МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

МЕТОДИЧЕСКИЕ УКАЗАНИЯ

по выполнению практических работ
по дисциплине «Программная инженерия» для студентов направления подготовки
09.03.04 Программная инженерия
Направленность (профиль)
«Разработка и сопровождение программного обеспечения»

Ставрополь 2026

Содержание

ВВЕДЕНИЕ

1. Общая характеристика дисциплины

Программная инженерия - это дисциплина, изучающая систематический, дисциплинированный и измеримый подход к разработке, эксплуатации и сопровождению программного обеспечения. В отличие от программирования, которое фокусируется на написании кода, программная инженерия охватывает все этапы жизненного цикла программного продукта: от бизнес-анализа и сбора требований до проектирования, реализации, тестирования, развертывания и сопровождения.

Цель изучения дисциплины - сформировать у студентов комплексное понимание процессов, методов и инструментов создания качественного программного обеспечения, а также развить практические навыки проектирования, разработки и документирования программных систем с использованием современных архитектурных паттернов и технологий.

2. Структура и содержание лабораторного практикума

Лабораторный практикум состоит из 27 лабораторных работ, распределённых по двум семестрам:

Семестр	Количество работ	Фокус
4 семестр	9 работ	Основы программной инженерии
5 семестр	18 работ	Углублённая разработка и Enterprise-технологии

2.1. 4 семестр: Базовый курс (9 лабораторных работ)

Работы первого семестра закладывают фундамент: студенты осваивают бизнес-анализ, системные требования, архитектурное проектирование, проектирование баз данных и реализацию ядра системы.

№	Название	Ключевые артефакты
ЛР1	Инициация проекта и модель бизнес-объектов	IDEF0, BUC, глоссарий, SWOT, ROI

№	Название	Ключевые артефакты
ЛР2	Объектная модель предметной области и требования	Use Case диаграмма, Domain Model
ЛР3	Проектирование архитектурного каркаса PCMEF	Диаграмма пакетов, ADR
ЛР4	Проектирование БД (ER-диаграмма, нормализация)	ER-диаграмма, логическая модель
ЛР5	Создание DDL-скриптов	SQL-скрипты, индексы, ограничения
ЛР6	Реализация Entity-слоя (JPA-сущности)	JPA-аннотации, бизнес-методы
ЛР7	Реализация Foundation-слоя (репозитории)	Интерфейсы и реализации репозиториев
ЛР8	Модульное тестирование (JUnit + Mockito)	Тесты, отчёт JaCoCo
ЛР9	Итоговая защита (промежуточная)	Презентация, допуск к 5 семестру

2.2. 5 семестр: Продвинутый курс (18 лабораторных работ)

Работы второго семестра углубляют знания и расширяют технологический стек: бизнес-логика, REST API, веб-интерфейс, безопасность, тестирование, контейнеризация.

№	Название	Ключевые артефакты
ЛР10	Спецификация двух ключевых прецедентов	Детализированные Use Case спецификации
ЛР11	Диаграммы последовательности	Sequence diagrams

№	Название	Ключевые артефакты
	для 3 сценариев	
ЛР12	Применение паттернов GoF (Strategy, Observer)	Реализация паттернов в коде
ЛР13	Уточнение диаграммы классов проектирования	Design Class Diagram
ЛР14	JPA: связи @OneToMany, @ManyToOne	JPA-аннотации, тесты связей
ЛР15	Реализация Mediator-слоя (Spring @Service)	Сервисы, бизнес-логика, транзакции
ЛР16	Реализация Control-слоя (Spring @RestController)	REST API, Swagger, валидация
ЛР17	Интеграционное тестирование (TestContainers)	Интеграционные тесты с реальной БД
ЛР18	Настройка JWT аутентификации в Spring Security	JWT, SecurityConfig, фильтры
ЛР19	Создание REST API (CRUD для основной сущности)	Полноценный CRUD API
ЛР20	Создание React-приложения (инициализация, роутинг)	React + Vite, React Router
ЛР21	Разработка компонентов (список, карточка, форма)	EventCard, EventForm, EventList
ЛР22	Интеграция React с REST API (Axios)	Axios, интерцепторы, API-сервисы

№	Название	Ключевые артефакты
ЛР23	Валидация форм (React Hook Form + Zod)	Схемы валидации, кастомные компоненты
ЛР24	Адаптивная вёрстка (Bootstrap/Tailwind)	Бургер-меню, адаптивная сетка
ЛР25	Обработка ошибок и загрузки	Error Boundary, Skeleton, Retry
ЛР26	Сборка проекта в Docker (Dockerfile)	Dockerfile, docker-compose.yml
ЛР27	Итоговая защита проекта	Пояснительная записка, презентация

3. Сквозной проект

Все лабораторные работы объединены единым сквозным проектом — программной системой, которая развивается от лабораторной работы к лабораторной работе и в дальнейшем служат основой к курсовому проекту. Студент выбирает один проект из 15 вариантов (или предлагает свой) и последовательно реализует его на протяжении всего курса.

Рекомендуемые темы проектов:

	Категория	Проект
	Управление ресурсами	Система управления складом
	Управление ресурсами	Платформа аренды оборудования
	Управление ресурсами	Система управления парком автомобилей
	Образование	Система дистанционного обучения (LMS)

	Категория	Проект
	Образование	Платформа для проведения онлайн-курсов
	Образование	Система записи на кружки и секции
	Здравоохранение	Система управления медицинскими назначениями
	Здравоохранение	Платформа записи на диагностические процедуры
	Здравоохранение	Система мониторинга здоровья пациентов
0	Торговля и услуги	Система управления доставкой еды
1	Торговля и услуги	Платформа бронирования услуг красоты
2	Торговля и услуги	Система проката спортивного инвентаря
3	Производство	Система управления производственными заказами
4	Логистика	Платформа логистики и отслеживания грузов
5	Управление	Система управления проектами в строительстве

4. Траектории выполнения

В зависимости от интересов и уровня подготовки студент может выбрать одну из четырёх траекторий выполнения курсового проекта:

Траектория	Фокус	Технологии	Сложность
А. Десктопная	Толстый клиент, локальная БД	Java Swing/JavaFX, SQLite/PostgreSQL	★★☆
Б. Веб-ориентированная	Тонкий клиент, серверное приложение	JSP/Servlets/Spring MVC, PostgreSQL	★★★
В. Мобильная	Нативное/кроссплатформенное приложение	Android/Kotlin, Flutter, React Native + Java Backend	★★★☆
Г. Enterprise	Полный стек + API + контейнеризация	Full Stack, REST, Docker, Cloud	★★★★

5. Архитектурная основа: PCMEF

Все лабораторные работы базируются на архитектурном паттерне PCMEF (Presentation-Control-Mediator-Entity-Foundation), который обеспечивает:

1. Чёткое разделение ответственности между компонентами
2. Направленность зависимостей сверху вниз
3. Возможность тестирования каждого слоя изолированно
4. Адаптацию под различные среды выполнения (Desktop, Web, Mobile)

Структура PCMEF:

Слой	Назначение
Presentation (P)	Интерфейс пользователя, отображение данных

Слой	Назначение
Control (C)	Обработка запросов, координация, валидация
Mediator (M)	Бизнес-логика, транзакции, правила
Entity (E)	Бизнес-сущности, состояние, методы предметной области
Foundation (F)	Доступ к данным, инфраструктура, репозитории

6. Технологический стек

Компонент	Технология	Применение
Бэкенд	Java 17+, Spring Boot 3.x	REST API, бизнес-логика, безопасность
База данных	PostgreSQL, H2 (тесты)	Хранение данных, миграции
ORM	Spring Data JPA / Hibernate	Объектно-реляционное отображение
Фронтенд (Web)	React 18+, Vite, Axios	Веб-интерфейс
Стилизация	Bootstrap / Tailwind CSS	Адаптивная вёрстка
Тестирование	JUnit 5, Mockito, TestContainers	Модульное и интеграционное тестирование
Контейнеризация	Docker, Docker Compose	Развёртывание

Компонент	Технология	Применение
Сборка	Maven / Gradle	Управление зависимостями
Контроль версий	Git, GitHub/GitLab	Версионность, совместная работа

7. Структура лабораторной работы

Каждая лабораторная работа имеет единую структуру:

Цель работы — что студент должен освоить

Задачи — конкретные шаги для достижения цели

Теоретическое обоснование — краткий обзор необходимых теоретических знаний

Пример выполнения — сквозной пример на проекте EMS

Методика выполнения — пошаговая инструкция

Варианты заданий — 15 вариантов для индивидуального выполнения

Контрольные вопросы — для самопроверки и защиты

Требования к отчёту — структура и оформление

Критерии оценивания — система баллов

Связь с последующими работами — преемственность

8. Требования к репозиторию и документации

8.1. Репозиторий

Все лабораторные работы и курсовой проект должны храниться в Git-репозитории (GitHub/GitLab) со следующей структурой:

```
course-project/
├── .gitignore
├── README.md
├── pom.xml / build.gradle
├── docs/
│   ├── 00-project-charter/
│   ├── 01-requirements/
```

```
│   ├── 02-architecture/
│   ├── 03-database/
│   ├── 04-detailed-design/
│   ├── 05-implementation/
│   ├── 06-testing/
│   ├── 07-refactoring/
│   ├── 08-ui/
│   ├── 09-api/
│   ├── 10-deployment/
│   ├── 11-user-guide/
│   └── 12-final-report/
├── src/
│   ├── main/
│   └── test/
├── docker/
└── scripts/
```

8.2. Документация

Вся документация оформляется в формате Markdown (.md) и хранится в репозитории. README.md в корне репозитория должен содержать:

Название и описание проекта

Выбранную траекторию

Технологический стек

Инструкцию по установке и запуску

Ссылки на документацию по этапам

Статистику разработки (Git Insights)

9. Рекомендации по выполнению

Не откладывайте на последний день — каждая работа требует времени на осмысление и выполнение

Ведите репозиторий в актуальном состоянии — коммитьте изменения регулярно

Пользуйтесь теоретическим обоснованием — оно содержит ключевые концепции

Следуйте примеру — сквозной проект EMS поможет понять, что нужно делать

Проверяйте себя по контрольным вопросам — они отражают суть работы

Оформляйте отчёты сразу — не откладывайте документацию на потом

Используйте инструменты — PlantUML для диаграмм, JaCoCo для покрытия

Данный лабораторный практикум охватывает полный цикл разработки программного обеспечения — от бизнес-анализа до развёртывания в Docker. Выполнение всех работ позволит сформировать компетенции, необходимые для профессиональной деятельности в области разработки программного обеспечения, а также подготовит к выполнению дипломного проекта на 4 курсе.

Лабораторная работа № 1

Инициация проекта и модель бизнес-объектов

Цель работы: Определить границы автоматизации и построить бизнес-модель

1. Теоретическое обоснование

1.1. Важность предпроектного анализа в программной инженерии

Перед тем как приступить к проектированию и разработке программного обеспечения, необходимо чётко понять бизнес-контекст системы. Согласно исследованиям Standish Group, 47% неудач IT-проектов связаны с неполными или неясными требованиями. Модель бизнес-объектов (Business Object Model, BOM) - это инструмент, который позволяет:

1. Установить общий язык между заказчиком, бизнес-аналитиками и разработчиками.

2. Определить границы автоматизации - что будет делать система, а что останется за её пределами.

3. Выявить ключевые бизнес-сущности и процессы, которые нужно поддерживать.

4. Предотвратить фундаментальные ошибки на ранних этапах, когда их исправление наименее затратно.

1.2. Концепция модели бизнес-объектов

Практическая программная инженерия основывается на модели бизнес-объектов, которая включает четыре взаимосвязанных компонента:

1. Диаграмма бизнес-контекста - определяет систему как «чёрный ящик» в окружении внешних сущностей.

2. Модель бизнес-прецедентов - показывает, какие бизнес-действия система должна поддерживать.

3. Бизнес-глоссарий - формализует терминологию предметной области.

4. Модель бизнес-классов - идентифицирует ключевые бизнес-сущности и их отношения.

Эти компоненты образуют иерархию уточнения: от общего контекста к конкретным понятиям.

Разберем пример проекта: «Система управления событиями (Event Management System)»

2.1. Диаграмма бизнес-контекста (аналог IDEF0 A-0)

Цель этого шага - показать систему как единый функциональный блок в бизнес-окружении.

Методика построения:

1. Определить основную функцию системы в бизнес-терминах.

2. Идентифицировать входные потоки (что преобразуется).

3. Идентифицировать выходные потоки (результаты преобразования).

4. Определить управляющие воздействия (правила, ограничения).

5. Определить механизмы выполнения (кто/что реализует функцию).

Пример для Event Management System (EMS):

Элементы диаграммы:

- Основная функция (A0): «Организация и проведение мероприятий»

- Входы (Inputs):

- Запрос на проведение мероприятия (цель, ожидания)
- Бюджет

- Выходы (Outputs):
  - Проведённое мероприятие с отчётом
  - Удовлетворённые участники
- Управление (Controls):
  - Нормативные требования (безопасность, законодательство)
- Механизмы (Mechanisms):
  - Организаторы
  - Волонтёры

2.2. Модель бизнес-прецедентов (BUC-диаграмма)

Цель второго шага - показать, какие бизнес-действия система должна поддерживать.

Методика построения:

1. Выявить бизнес-акторов (роли, вовлечённые в бизнес-процессы).
2. Определить бизнес-прецеденты (значимые бизнес-действия).
3. Установить связи между акторами и прецедентами.

Например,

Организатор

1. Планирование мероприятия ® Подбор места и ресурсов ® Координация логистики
2. Регистрация участников ® Сбор и оплата подтверждений
3. Управление бюджетом ® Формирование отчетов

Ключевые элементы:

- Бизнес-акторы: Организатор, Участник, Спонсор, Поставщик услуг

- Бизнес-прецеденты:

- Планирование мероприятия (включает подбор места и координацию логистики)

- Регистрация участников (включает сбор оплат)

- Управление бюджетом (включает формирование отчётов)

2.3. Бизнес-глоссарий

Цель третьего шага - создать единый словарь терминов предметной области.

Методика составления:

1. Выписать все ключевые термины из описания предметной области.
2. Дать чёткие, однозначные определения.
3. Указать связи между терминами.

Пример бизнес-глоссария:

Термин	**Определение**	**Связи/Комментарии**
Мероприятие (Event)	Запланированное собрание людей с определённой целью (конференция, семинар, встреча).	Имеет Организатора, Место проведения, Расписание
Организатор (Organizer)	Лицо или организация, ответственные за планирование и проведение мероприятия.	Создает Мероприятия, управляет Бюджетом

Участник (Participant)	Лицо, зарегистрированное для посещения мероприятия.	Регистрируется на Мероприятие, может вносить Оплату
Место проведения (Venue)	Физическое расположение, где проводится мероприятие.	Имеет Вместимость, Оснащение, Стоимость аренды
Сессия (Session)	Отдельный блок программы мероприятия (доклад, мастер-класс).	Часть Мероприятия, имеет Время начала/окончания, Докладчика
Бюджет мероприятия (Event Budget	План доходов и расходов, связанных с проведением мероприятия.	Включает Статьи расходов, Источники финансирования
Регистрация (Registration)	Процесс записи участника на мероприятие.	Создаёт Учётную запись Участника, генерирует Билет

2.4. Модель бизнес-классов (высокоуровневая)

Цель четвертого шага - идентифицировать ключевые бизнес-сущности и их отношения.

Методика построения:

1. Выделить существительные из бизнес-глоссария и описаний прецедентов.

2. Определить, какие из них являются бизнес-сущностями (имеют состояние и поведение).

3. Установить основные отношения между сущностями.

4. Определить ключевые атрибуты каждой сущности.

Пример модели бизнес-классов:

Организатор (actor):

- Организует:

- Мероприятие: название, дата начала, дата окончания, статус

- Имеет:

    - Место проведения: название, адрес, вместимость;

    - Сессия: тема, время_начала, время_окончания, докладчик.

Участник (actor):

- Участвует:

    - Регистрация: статус, дата_регистрации, сумма_оплаты

- Имеет:

    - Билет: номер, QR_код, статус проверки.

Ключевые отношения:

1. Агрегация (<>----->): Мероприятие состоит из множества Регистраций

2. Ассоциация (----->): Участник участвует в Мероприятии через Регистрацию

3. Композиция (◆----->): Сессия является неотъемлемой частью Мероприятия

Методические указания к выполнению лабораторной работы

Этап 1: Подготовка

- Изучите список из 15 проектов

- Выберите проект, соответствующий вашим интересам и знаниям

- Сформулируйте краткое описание проекта (2-3 предложения)

- Установите необходимые инструменты: Draw.io или др.

Этап 2: Основное выполнение

Блок 2.1: Бизнес-анализ (4 часа)

Шаг 2.1.1. Резюме проекта (Executive Summary)

Структура резюме:

1. Проблема (2-3 предложения): Что не так сейчас?

2. Решение (2-3 предложения): Что предлагаете вы?

3. Цели проекта (3-4 пункта): Чего планируете достичь?

4. Ожидаемые результаты (2-3 пункта): Что получит заказчик?

5. Ключевые показатели успеха (3-4 пункта): Как измерим успех?

6. Сроки и бюджет (1 предложение): Общая оценка

Пример:

Проблема: Организация мероприятий требует ручного труда 3 сотрудников по 40 часов в месяц, возникают ошибки при регистрации участников.

Решение: Разработка системы EventPro, автоматизирующей планирование, регистрацию и отчётность.

Цели: Сократить время организации на 40%, увеличить регистрации на 25%, снизить ошибки на 90%.

Ожидаемые результаты: Автоматизация 80% рутинных операций, интеграция с платежными системами, мобильный доступ для организаторов.

Ключевые показатели: Время обработки заявки < 5 минут, удовлетворённость участников > 85%, окупаемость < 12 месяцев.

Сроки: 6 месяцев разработки, бюджет: 1.5 млн рублей.

Шаг 2.1.2: Бизнес-контекст (1 час)

- Нарисуйте диаграмму IDEF0 A-0

- Подготовьте описание:

1. Основная функция системы: [глагол + существительное]
2. Входные потоки: [что преобразуется, 3-5 пунктов]
3. Выходные потоки: [результаты, 3-5 пунктов]
4. Управляющие факторы: [правила, ограничения, 2-3 пункта]
5. Исполнители: [кто/что выполняет, 2-3 пункта]

Шаг 2.1.3: Матрица стейкхолдеров

Стейкхолдер	Роль/Влияние	Интересы/Ожидания	Влияние (1-5)	Отношение	Стратегия работы
Организатор	Заказчик/Инициатор	Автоматизация рутины, контроль бюджета	5	+	Регулярные встречи, демо
Участник	Конечный пользователь	Удобство, информация, поддержка	4	+	UX-исследования, фидбек
Бухгалтерия	Внутренний потребитель	Корректные отчёты, интеграция с 1С	3	0	Совместные сессии
ИТ-отдел	Техническая поддержка	Стабильность, документация	2	-	Технические брифинги

Шаг 2.1.4: SWOT-анализ

СИЛЬНЫЕ СТОРОНЫ (внутренние +):

- Опытная команда организаторов
- Существующая клиентская база

СЛАБЫЕ СТОРОНЫ (внутренние -):

- Ручная обработка данных
- Высокая текучесть персонала

ВОЗМОЖНОСТИ (внешние +):

- Рост рынка онлайн-мероприятий
- Пример: Новые технологии доступны

УГРОЗЫ (внешние -):

- Конкуренция с платформами-гигантами
- Изменение законодательства

Блок 2.2: Требования и качество

Шаг 2.2.1: BUC-диаграмма

- Нарисуйте в PlantUML или Draw.io
- Включите 3-5 бизнес-акторов
- Определите 6-10 бизнес-прецедентов
- Добавьте пояснения:

Пример . Пояснения к диаграмме:

Актор Организатор взаимодействует с системой через три основных бизнес-прецедента:

Планирование мероприятия: Организатор определяет основные параметры события - дату, время, место, целевую аудиторию, бюджет и ключевые показатели успеха. Этот прецедент является инициирующим для всего процесса.

Управление бюджетом: Организатор контролирует финансовые потоки, утверждает статьи расходов, отслеживает оплаты от участников и спонсоров, распределяет средства между различными направлениями мероприятия.

Оценка эффективности: После завершения мероприятия организатор анализирует ключевые метрики (посещаемость, удовлетворённость участников, финансовый результат), что позволяет принимать решения для улучшения будущих событий.

Бизнес-прецедент Планирование мероприятия включает три подпроцесса:

Подбор места - выбор и бронирование подходящей локации с учётом количества участников, технических требований, географического

расположения и стоимости. Этот процесс может включать несколько итераций согласования с владельцами помещений.

Определение программы - разработка содержательной части мероприятия: подбор спикеров, формирование расписания, определение форматов выступлений (доклады, мастер-классы, панельные дискуссии), планирование кофе-брейков и нетворкинга.

Назначение ответственных - распределение ролей и зон ответственности среди членов команды: кто отвечает за логистику, кто за контент, кто за связь с участниками, кто за техническое обеспечение.

Отношение include между Регистрация участников и Сбор оплат означает, что процесс регистрации обязательно содержит в себе этап финансового взаимодействия. Это обязательная часть основного бизнес-прецедента. Без сбора оплат регистрация не может считаться завершённой. Такое отношение показывает, что:

Сбор оплат - это не самостоятельный бизнес-процесс, а компонент более крупного процесса регистрации

При каждом выполнении Регистрации участников обязательно выполняется Сбор оплат (если мероприятие платное)

Логика и последовательность Сбора оплат могут меняться (разные платёжные системы, валюты, скидки), но сам факт финансового подтверждения присутствует всегда

Шаг 2.2.2: Карта пути клиента (CJM) (1 час)

Выберите один ключевой сценарий и детализируйте его:

Этап	Цель клиента	Действия	Мысли	Чувства	Болевые точки	Возможности
Осведомленность	Узнать о мероприятии	Видит рекламу	"Интересно, но..."	Любопытство, сомнение	Мало информации	Детальная программа
Рассмотрение	Оценить ценность	Читает программу	"Что я получу?"	Интерес, анализ	Нет отзывов	Видео с прошлых мероприят

						ий
Регистрация	Записатьс я	Заполня ет форму	"Сколько времени?"	Нетерпени е	Сложная форма	Упрощённ ый процесс
Оплата	Подтверд ить участие	Вводит данные	"Безопасно ли?"	Осторожно сть	Нет SSL	PCI DSS compliance
Участие	Получить опыт	Посеща ет	"Соответств ует ли ожиданиям ?"	Надежда	Плохая организац ия	Мобильно е приложени е

Шаг 2.2.3: Матрица K2 (Качество-Функция)

Функциональн ые требования	Надёжнос ть	Производ ит-сть	Безопаснос ть	Удобст во	Поддер ж-ка	Ве с
Онлайн-регистрация	9	7	8	10	6	0.2 5
Управление бюджетом	10	6	9	5	8	0.2 0
Рассылка уведомлений	8	8	7	7	9	0.1 5
Формировани е отчётов	9	5	8	6	10	0.2 0
Интеграция с платежами	10	7	10	6	7	0.2 0
Итого (средневзвеш.)	9.2	6.6	8.4	6.8	8.0	1.0 0

Пояснение: Оцените по шкале 1-10, где 10 — критически важно

Шаг 2.2.4: Критерии приёмки

Для 3 ключевых требований определите критерии:

1. ТРЕБОВАНИЕ: Регистрация участника

   КРИТЕРИИ ПРИЁМКИ:

   - Система принимает регистрацию за ≤ 3 минут

   - Отправляет подтверждение в течение 60 секунд

   - Валидирует email и телефон

- Сохраняет черновик при прерывании

2. ТРЕБОВАНИЕ: Приём оплаты

КРИТЕРИИ ПРИЁМКИ:

- Поддерживает 3+ платёжных системы
- Автоматически генерирует чек
- Уведомляет при успешной оплате
- Даёт 3 попытки при ошибке

3. ТРЕБОВАНИЕ: Формирование отчёта

КРИТЕРИИ ПРИЁМКИ:

- Генерирует PDF/Excel за ≤ 30 секунд
- Включает все требуемые разделы
- Соответствует корпоративному стилю
- Отправляет на email автоматически

Блок 2.3: Экономика и планирование

Шаг 2.3.1: Расчёт ROI

РАСЧЁТ ОКУПАЕМОСТИ ИНВЕСТИЦИЙ

1. ТЕКУЩИЕ ЗАТРАТЫ (без системы):

- Зарплата персонала: 3 чел × 40 ч/мес × 500 руб/ч = 60,000 руб/мес
- Ошибки и переделки: 15,000 руб/мес
-Итого: 75,000 руб/мес × 12 = 900,000 руб/год

2. ЗАТРАТЫ ПОСЛЕ ВНЕДРЕНИЯ:

-Зарплата: 1 чел × 20 ч/мес × 500 руб/ч = 10,000 руб/мес
- Поддержка системы: 5,000 руб/мес
- Амортизация системы: 300,000 руб / 3 года / 12 = 8,333 руб/мес

- Итого: 23,333 руб/мес × 12 = 280,000 руб/год

3. ПРЯМАЯ ЭКОНОМИЯ:

- 900,000 - 280,000 = 620,000 руб/год

4. ДОПОЛНИТЕЛЬНЫЕ ВЫГОДЫ:

- Увеличение регистраций на 25%: +200,000 руб/год
- Снижение отмен: +50,000 руб/год
-Итого дополнительно: 250,000 руб/год

5. ОБЩАЯ ВЫГОДА:

• 620,000 + 250,000 = 870,000 руб/год

6. ИНВЕСТИЦИИ:

- Разработка: 1,500,000 руб
- Внедрение: 200,000 руб
- Обучение: 100,000 руб
- Итого: 1,800,000 руб

7. ROI:

• Простой срок окупаемости: 1,800,000 / (870,000/12) = 25 месяцев
• ROI за 3 года: ((870,000×3 - 1,800,000) / 1,800,000) × 100% = 45%

Шаг 2.3.2: Матрица приоритизации MoSCoW

ПРИОРИТЕТЫ ТРЕБОВАНИЙ (MoSCoW)

MUST HAVE (должны быть в Release 1):

- Регистрация участников с базовыми полями
- Приём онлайн-оплат через 1 платёжную систему
- Формирование списков участников в Excel

- Базовая аутентификация (логин/пароль)

SHOULD HAVE (желательно в Release 1):

- Интеграция с корпоративной почтой
- Шаблоны сертификатов
- Модуль уведомлений (email/SMS)
- Простая аналитика (статистика регистраций)

COULD HAVE (могли бы быть, если останется время):

- Мобильное приложение для организаторов
- Социальные функции (чат участников)
- AI-рекомендации спикеров
- Интеграция с календарями

WON'T HAVE (не в этой версии):

- VR-трансляции мероприятий
- Мультиязычная поддержка
- Блокчейн для сертификатов
- Интеграция с CRM конкурентов

Шаг 2.3.3: Рамка системы

Нарисуйте схему взаимодействий:

Шаг 2.3.4: План коммуникации

Этап	**Цель коммуникации**	**Участники**	**Формат**	**Частота**	**Ответственный**	**Артефакты**
Инициация	Утверждение границ проекта	Заказчик, спонсор	Встреча	1 раз	Менеджер	Устав проекта
Анализ	Сбор требований	Пользователи, эксперты	Интервью	2 раза/нед	Аналитик	Требования
Обзор	Проверка результатов	Команда, заказчик	Демо	Каждые 2 недели	Тимлид	Прототипы
Риски	Мониторинг проблем	Руководство	Отчёт	Ежемесячно	Менеджер	Risk Register
Закрытие	Приёмка результатов	Все стейкхолдеры	Презентация	По завершении	Менеджер	Отчёт

Шаг 2.3.5: Прототип видения

Создайте одностраничный документ:

[НАЗВАНИЕ СИСТЕМЫ]

Vision Document v1.0

ДЛЯ [целевая аудитория]

КОТОРЫЕ [проблема/потребность]

НАША СИСТЕМА [категория системы]

КОТОРАЯ [ключевое преимущество]

В ОТЛИЧИЕ ОТ [конкуренты/альтернативы]

НАШЕ РЕШЕНИЕ [уникальное предложение]

ЦЕННОСТЬ ДЛЯ БИЗНЕСА

✓ Сокращение времени организации на 40%

✓ Увеличение регистраций на 25%

✓ Снижение ошибок в отчётах на 90%

✓ ROI: 45% за 3 года

КЛЮЧЕВЫЕ ВОЗМОЖНОСТИ

• Автоматическая рассылка персонализированных приглашений

• Интеграция с 3+ платёжными системами

• Мобильное приложение для организаторов

• Аналитика в реальном времени

ТЕХНОЛОГИЧЕСКИЙ СТЕК

Backend: Java 17, Spring Boot 3, PostgreSQL

Frontend: React 18, TypeScript, Tailwind CSS

Mobile: Flutter 3 (iOS/Android)

Infra: Docker, AWS, GitHub Actions

СРОКИ И БЮДЖЕТ

• Разработка: 6 месяцев (итерации по 2 недели)

• Бюджет: 1.5 млн рублей

• Окупаемость: 25 месяцев

• Команда: 5 человек (PO, Dev ×2, QA, BA)

КЛЮЧЕВЫЕ МЕТРИКИ УСПЕХА:

• NPS пользователей > 70

• Время обработки заявки < 5 минут

• Доступность системы > 99.5%

• Удовлетворённость заказчика > 8/10

Структура итогового отчета:

1. Титульный лист

2. Содержание (с номерами страниц)

3. Резюме проекта (1 стр.)

4. 1. Введение и контекст

   - 1.1. Описание проекта

   - 1.2. Бизнес-контекст (диаграмма + описание)

   - 1.3. Матрица стейкхолдеров

5. 2. Анализ текущего состояния

   - 2.1. SWOT-анализ

   - 2.2. Карта пути клиента (CJM)

6. 3. Требования и качество

   - 3.1. BUC-диаграмма

   - 3.2. Матрица K2 (Качество-Функция)

   - 3.3. Критерии приёмки

7. 4. Экономическое обоснование

   - 4.1. Расчёт ROI

   - 4.2. Матрица MoSCoW

8. 5. Техническое видение

   - 5.1. Рамка системы

   - 5.2. Прототип видения

   - 5.3. План коммуникации

Варианты индивидуальных заданий

Вариант	Категория	Проект
1.	**Управление ресурсами**	Система управления складом - учёт товаров, инвентаризация, приёмка/отпуск
2.		Платформа аренды оборудования - бронирование строительной/производственной техники
3.		Система управления парком автомобилей - учёт ТС, планирование ремонтов, распределение водителей
4.	Образование и обучение	Система дистанционного обучения (LMS) - управление курсами, тестирование, отслеживание прогресса
5.		Платформа для проведения онлайн-курсов - регистрация, доступ к материалам, проверка заданий
6.		Система записи на кружки и секции - расписание, оплата, контроль посещаемости
7.	Здравоохранение	Система управления медицинскими назначениями — учёт пациентов, назначения, результаты анализов
8.		Платформа записи на диагностические процедуры — МРТ, КТ, УЗИ с учётом оборудования и специалистов
9.		Система мониторинга здоровья пациентов — сбор данных с устройств, алерты для врачей
10.	Торговля и услуги	Система управления доставкой еды — заказы, курьеры, маршрутизация, отслеживание
11.		Платформа бронирования услуг красоты и здоровья — салоны,

		мастера, расписание
12.		Система проката спортивного инвентаря — лыжи, велосипеды, снаряжение с учётом сезона
13.	Производство и логистика	Система управления производственными заказами — планирование, отслеживание этапов, контроль качества
14.		Платформа логистики и отслеживания грузов — маршруты, транспорт, статусы доставки
15.	Управление	Система управления проектами

Лабораторная работа №2

Объектная модель предметной области и требования

1. Цель и задачи работы:

Цель: Освоить методику преобразования бизнес-требований в формальные системные спецификации посредством создания объектной модели предметной области (Domain Model) и диаграммы вариантов использования (Use Case Diagram).

Задачи:

1. На основе бизнес-модели (ЛР1) выделить системных акторов и прецеденты
2. Построить диаграмму вариантов использования (Use Case Diagram)
3. Создать объектную модель предметной области (Domain Model) — диаграмму классов с атрибутами, методами и бизнес-правилами
4. Разработать глоссарий предметной области (20+ терминов)

5. Подготовить таблицу трассировки требований

2. ТЕОРЕТИЧЕСКОЕ ОБОСНОВАНИЕ

2.1. Переход от бизнес-модели к системным требованиям

В лабораторной работе №1 была построена модель бизнес-объектов (BOM). Следующий критически важный этап — это переход от бизнес-требований к системным требованиям.

Согласно исследованию IBM Systems Sciences Institute, исправление ошибки на этапе требований в 100 раз дешевле, чем на этапе тестирования.

Ключевые различия между бизнес-моделью и системными требованиями:

Аспект	**Бизнес-модель (ЛР1)**	**Системные требования (ЛР2)**
Уровень абстракции	Что должно быть сделано в бизнесе	Как система будет это делать
Акторы	Бизнес-роли (Организатор, Клиент)	Пользователи системы (Администратор, Пользователь)
Прецеденты	Бизнес-процессы	Функциональность системы
Язык	Бизнес-терминология	Технические спецификации
Цель	Понимание проблемы и контекста	Создание инструкций для разработки

2.2. Диаграмма вариантов использования (Use Case Diagram)

Use Case Diagram — это UML-диаграмма, показывающая, какие функции система предоставляет внешним пользователям (акторам).

Основные элементы:

Элемент	Обозначение	Описание
Актор (Actor)	[Человечек]	Роль пользователя, взаимодействующего с системой
Прецедент (Use Case)	(Название)	Конкретная функция, которую система выполняет
Отношение include	<<include>>	Один прецедент всегда вызывает другой
Отношение extend	<<extend>>	Один прецедент может дополнить другой при условии

Правила построения:

2. Акторы находятся вне системы (за границей)
3. Прецеденты — внутри системы
4. Акторы соединяются с прецедентами линиями (ассоциациями)

2.3. Объектная модель предметной области (Domain Model)

Domain Model — это структурное представление ключевых понятий и их отношений в предметной области. В отличие от бизнес-модели из ЛР1, Domain Model:

1. Фокусируется на сущностях и их поведении, а не на процессах
2. Использует UML Class Diagram с атрибутами и методами
3. Включает бизнес-правила в виде ограничений и валидаций

4. Является основой для проектирования БД и объектно-ориентированной архитектуры

Принципы построения Domain Model:

Принцип	Описание
Ubiquitous Language	Использование терминов, понятных и бизнесу, и разработчикам
Bounded Context	Чёткие границы моделируемой области
Aggregate Root	Определение главных сущностей, через которые происходит доступ

2.4. Глоссарий предметной области

Глоссарий — это словарь ключевых терминов проекта, который обеспечивает единое понимание предметной области всеми участниками.

Требования к глоссарию:

1. 20+ терминов
2. Каждый термин имеет чёткое определение
3. Указаны связи между терминами (где уместно)

2.5. Таблица трассировки требований

Таблица трассировки (Traceability Matrix) показывает связь между бизнес-требованиями (из ЛР1) и системными требованиями (из ЛР2).

Бизнес-требование (из ЛР1)	Системное требование (Use Case)	Статус
Управление мероприятиями	UC-001: Создать мероприятие	Реализовано

3. ПРИМЕР ВЫПОЛНЕНИЯ (СКВОЗНОЙ ПРОЕКТ)

Проект: «Система управления мероприятиями» (Event Management System)

3.1. Исходные данные из ЛР1

Бизнес-акторы: Организатор, Участник, Бухгалтер, Спонсор

Бизнес-прецеденты:

1. Планирование мероприятия
2. Регистрация участников
3. Управление бюджетом
4. Формирование отчётов

3.2. Шаг 1: Построение диаграммы вариантов использования (Use Case)

Трансформация бизнес-прецедентов в системные Use Cases:

Бизнес-прецедент	Системные Use Cases
Планирование мероприятия	UC-001: Создать мероприятие UC-002: Редактировать мероприятие UC-003: Удалить мероприятие
Регистрация участников	UC-004: Зарегистрироваться на мероприятие UC-005: Отменить регистрацию
Управление бюджетом	UC-006: Оплатить участие UC-007: Просмотреть финансовый отчёт
Формирование отчётов	UC-008: Сформировать отчёт по участникам

Диаграмма Use Case для EMS (PlantUML):

```
@startuml
left to right direction
actor "Администратор" as Admin
actor "Участник" as User
actor "Бухгалтер" as Accountant
rectangle "Event Management System" {
  usecase "UC-001: Создать мероприятие" as UC1
  usecase "UC-002: Редактировать мероприятие" as UC2
  usecase "UC-003: Удалить мероприятие" as UC3
  usecase "UC-004: Зарегистрироваться на мероприятие" as UC4
  usecase "UC-005: Отменить регистрацию" as UC5
  usecase "UC-006: Оплатить участие" as UC6
  usecase "UC-007: Просмотреть финансовый отчёт" as UC7
  usecase "UC-008: Сформировать отчёт по участникам" as UC8
  UC4 ..> UC6 : <<include>>
}
Admin --> UC1
Admin --> UC2
Admin --> UC3
User --> UC4
User --> UC5
Accountant --> UC7
Accountant --> UC8

@enduml
```

Пояснения к диаграмме (рисунок 1):

Актор «Администратор» заменяет бизнес-актора «Организатор» для уровня системы. Администратор отвечает за управление мероприятиями (создание, редактирование, удаление).

Актор «Участник» — это пользователь системы, который может регистрироваться на мероприятия и отменять регистрацию.

Актор «Бухгалтер» имеет доступ к финансовым отчётам и отчётам по участникам.

Отношение <<include>> между «Зарегистрироваться на мероприятие» и «Оплатить участие» означает, что регистрация на платное мероприятие всегда включает этап оплаты.

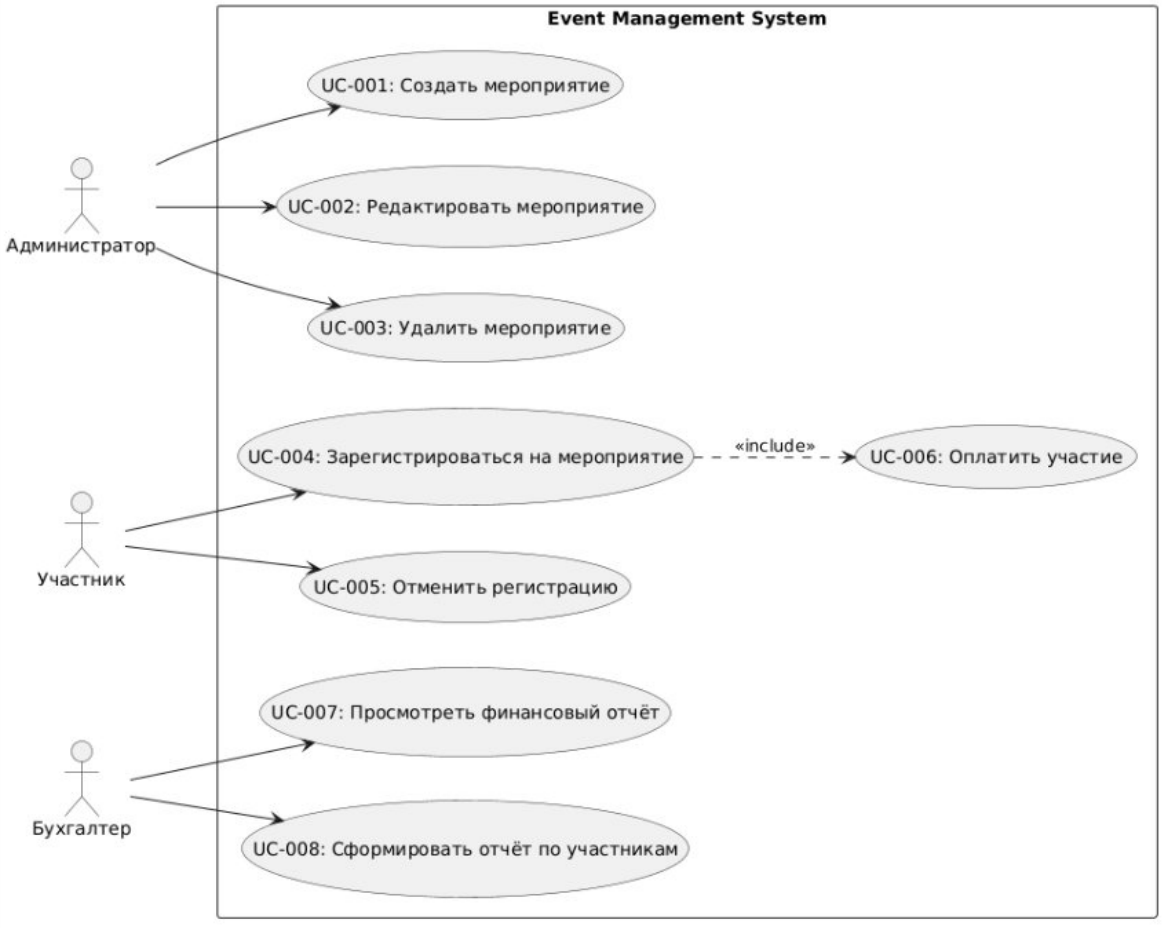


Рисунок - 1

3.3. Шаг 2: Построение объектной модели предметной области (Domain Model)

Методика построения Domain Model:

1. Выделение существительных из глоссария и описаний прецедентов
2. Классификация на сущности (Entity), значения (Value Object), перечисления (Enum)
3. Определение атрибутов с типами данных
4. Установление отношений (ассоциация, агрегация, композиция, наследование)
5. Добавление бизнес-правил в виде ограничений

Domain Model для EMS (PlantUML):

```
@startuml

' Сущности (Entities)
class Event <<Entity>> {
  - id: Long
  - title: String
  - description: String
  - startDate: LocalDateTime
  - endDate: LocalDateTime
  - maxParticipants: Integer
  - status: EventStatus
  + create()
  + update()
  + cancel()
  + checkAvailability()
}

class User <<Entity>> {
  - id: Long
```

```
  - email: String
  - name: String
  - role: String
  + register()
  + authenticate()
}

class Registration <<Entity>> {
  - id: Long
  - registrationDate: LocalDateTime
  - status: RegistrationStatus
  + confirm()
  + cancel()
}

class Payment <<Entity>> {
  - id: Long
  - amount: BigDecimal
  - paymentDate: LocalDateTime
  - method: String
  + process()
  + refund()
}

' Перечисления (Enums)
enum EventStatus {
  DRAFT
  PUBLISHED
  CANCELLED
```

```
  COMPLETED
}

enum RegistrationStatus {
  PENDING
  CONFIRMED
  CANCELLED
}

' Отношения
Event "1" -- "0..*" Registration : has
User "1" -- "0..*" Registration : makes
Registration "1" -- "0..1" Payment : includes

' Бизнес-правила
note top of Event
  Правило 1: Дата начала должна быть
  в будущем относительно создания

  Правило 2: Максимальное количество
  участников должно быть >= 1
end note

note right of Registration
  Правило 3: Отмена регистрации
  возможна за 24 часа до начала
  мероприятия

  Правило 4: Статус CONFIRMED
```

```
  может быть только после успешной
  оплаты (для платных мероприятий)
end note

@enduml
```

Пояснения к Domain Model (рисунок 2):

Классы-сущности (<<Entity>>) имеют идентичность и жизненный цикл:

Event — идентифицируется по id, может менять статус

Registration — имеет жизненный цикл (создание → подтверждение → отмена)

Сущности содержат бизнес-логику в методах

Классы-значения (Value Object) не имеют идентичности (в данном примере не используются, но могут быть добавлены: Address, Price)

Типы отношений:

Агрегация (o--): Event имеет множество Registration (регистрации могут существовать независимо)

Ассоциация (--): User и Registration связаны, но независимы

Бизнес-правила:

Валидация дат (Event.create())

Проверка доступности мест (Event.checkAvailability())

Условия отмены регистрации

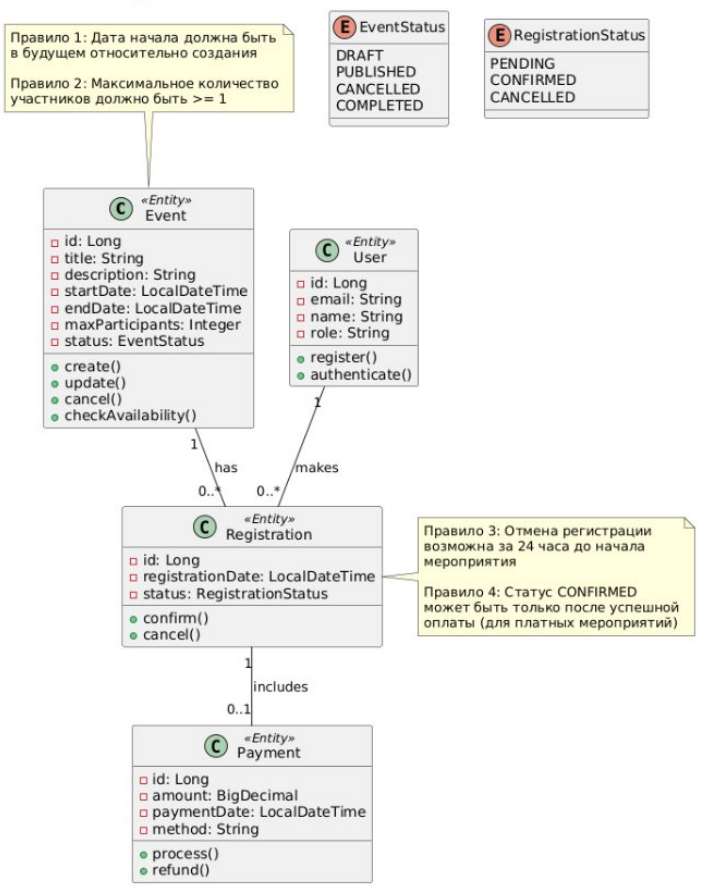


Рисунок -2

3.4. Шаг 3: Разработка глоссария предметной области

Термин	Определение	Связи/Комментарии
Мероприятие (Event)	Запланированное событие с определённой датой, местом и максимальным количеством участников	Имеет статус (DRAFT, PUBLISHED, CANCELLED, COMPLETED)
Пользователь (User)	Лицо, имеющее учётную запись в системе	Может иметь роль: USER или ADMIN

Термин	Определение	Связи/Комментарии
Регистрация (Registration)	Факт записи пользователя на мероприятие	Имеет статус (PENDING, CONFIRMED, CANCELLED)
Платёж (Payment)	Финансовая транзакция за участие в мероприятии	Связан с регистрацией
Категория (Category)	Группировка мероприятий по тематике	Например: Конференция, Вебинар, Мастер-класс
Администратор (Admin)	Пользователь с полными правами на управление системой	Роль ADMIN
Участник (Participant)	Пользователь, зарегистрированный на мероприятие	Роль USER
Статус мероприятия (EventStatus)	Состояние мероприятия: черновик, опубликовано, отменено, завершено	Перечисление
Статус регистрации (RegistrationStatus)	Состояние регистрации: ожидание, подтверждена, отменена	Перечисление

Термин	Определение	Связи/Комментарии
Бизнес-правило (Business Rule)	Ограничение или условие, которым должна удовлетворять система	Например: отмена за 24 часа

3.5. Шаг 4: Таблица трассировки требований

ID	Бизнес-требование (из ЛР1)	Системное требование (Use Case)	Тип связи	Статус
BR-01	Организация мероприятий	UC-001: Создать мероприятие	Реализует	+
BR-01	Организация мероприятий	UC-002: Редактировать мероприятие	Реализует	+
BR-01	Организация мероприятий	UC-003: Удалить мероприятие	Реализует	+
BR-02	Регистрация участников	UC-004: Зарегистрироваться	Реализует	+
BR-02	Регистрация	UC-005:	Реализу	+

ID	Бизнес-требование (из ЛР1)	Системное требование (Use Case)	Тип связи	Статус
	участников	Отменить регистрацию	ет	
BR-03	Финансовый учёт	UC-006: Оплатить участие	Реализует	+
BR-03	Финансовый учёт	UC-007: Просмотреть финансовый отчёт	Реализует	+
BR-04	Формирование отчётов	UC-008: Сформировать отчёт по участникам	Реализует	+

4. МЕТОДИКА ВЫПОЛНЕНИЯ (ПОШАГОВАЯ ИНСТРУКЦИЯ)

Шаг 1: Анализ результатов ЛР1

1. Откройте отчёт по лабораторной работе №1
2. Выделите бизнес-акторов (действующие лица бизнеса)
3. Выделите бизнес-прецеденты (что бизнес делает)
4. Определите ключевые бизнес-сущности

Шаг 2: Построение диаграммы Use Case (2 часа)

Определите системных акторов:

1. Кто будет непосредственно взаимодействовать с системой?

2. Каждому бизнес-актору может соответствовать один или несколько системных акторов
3. Для каждого бизнес-прецедента определите 1-3 системных Use Case:
4. Бизнес: «Планирование мероприятия» → Системные: «Создать мероприятие», «Редактировать мероприятие», «Удалить мероприятие»
5. Постройте диаграмму с использованием отношений <<include>> и <<extend>>
6. Добавьте пояснения к каждому актору и Use Case (минимум 3 пояснения)

Примерная таблица для заполнения:

Бизнес-актор (ЛР1)	**Системный актор**	**Бизнес-прецедент**	**Системные Use Cases**

Шаг 3: Построение Domain Model

Выпишите все существительные из глоссария ЛР1 и описаний прецедентов

Классифицируйте на:

1. Сущности (Entity) — имеют идентичность (id) и жизненный цикл
2. Значения (Value Object) — не имеют идентичности (например, Address, Price)
3. Перечисления (Enum) — фиксированный набор значений
4. Определите атрибуты с типами данных (Java-типы: Long, String, LocalDateTime, BigDecimal)

Установите отношения с указанием кратностей:

1. Ассоциация (--) — простое отношение

2. Агрегация (o--) — отношение «часть-целое» с независимым существованием частей
3. Композиция (*--) — отношение «часть-целое» с зависимым существованием частей
4. Наследование (<|--) — иерархия классов
5. Добавьте ключевые методы и бизнес-правила

Шаг 4: Разработка глоссария (1 час)

Соберите 20+ терминов из Domain Model и Use Case диаграммы

Для каждого термина дайте чёткое определение

Укажите связи между терминами (где уместно)

Формат глоссария:

Термин	Определение	Связи
...	...	...

Шаг 5: Таблица трассировки требований (1 час)

Установите связи между бизнес-требованиями (из ЛР1) и системными Use Cases

Заполните таблицу трассировки

Формат таблицы трассировки:

ID	Бизнес-требование	Системное требование	Тип связи	Статус
...	...	...	Реализует / Уточняет / ...	✓/✗/🔄

5. ВАРИАНТЫ ДЛЯ ИНДИВИДУАЛЬНОГО ВЫПОЛНЕНИЯ

Варианты соответствуют проектам из лабораторной работы №1:

	Проект	Ключевые сущности (минимум)
	Система управления складом	Product, Warehouse, StockMovement, Supplier, Category
	Платформа аренды оборудования	Equipment, Rental, Customer, Maintenance
	Система управления парком автомобилей	Vehicle, Driver, Trip, Maintenance, Fueling
	Система дистанционного обучения (LMS)	Course, Module, Lesson, Student, Enrollment, Progress
	Платформа для проведения онлайн-курсов	Course, Instructor, Student, Video, Quiz, Certificate
	Система записи на кружки и секции	Circle, Schedule, Student, Teacher, Attendance, Payment
	Система управления медицинскими назначениями	Patient, Doctor, Appointment, Prescription, Diagnosis
	Платформа записи на диагностические процедуры	Patient, Procedure, Equipment, Schedule, Result
	Система мониторинга	Patient, Measurement, Alert, Device, Doctor

	Проект	Ключевые сущности (минимум)
	здоровья пациентов	
0	Система управления доставкой еды	Restaurant, Dish, Order, Courier, Customer, Payment
1	Платформа бронирования услуг красоты	Salon, Master, Service, Client, Booking, Review
2	Система проката спортивного инвентаря	Item, Rental, Customer, Category, Maintenance
3	Система управления производственными заказами	Order, Product, Operation, Worker, Equipment
4	Платформа логистики и отслеживания грузов	Shipment, Route, Vehicle, Driver, Tracking, Customer
5	Система управления проектами в строительстве	Project, Task, Worker, Material, Equipment, Report

6. КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Чем отличается бизнес-актор от системного актора?
2. Что такое Use Case диаграмма и для чего она нужна?
3. В чём разница между отношениями <<include>> и <<extend>>?

4. Что такое Domain Model и чем она отличается от бизнес-модели?
5. Какие типы отношений между классами существуют в UML?
6. Что такое Value Object и чем он отличается от Entity?
7. Как определить, что класс должен быть сущностью, а не значением?
8. Какие бизнес-правила можно заложить в Domain Model?
9. Как обеспечить единообразие терминов между бизнес-аналитиками и разработчиками?
10. Что такое таблица трассировки требований и зачем она нужна?
11. Как связаны Use Case диаграмма и Domain Model?
12. Какие проблемы могут возникнуть, если пропустить этап построения Domain Model?
13. Как проверить, что Domain Model соответствует принципам DDD (Domain-Driven Design)?
14. Что такое агрегат (Aggregate) и корень агрегата (Aggregate Root)?
15. Как ограниченный контекст (Bounded Context) влияет на Domain Model?

7. ТРЕБОВАНИЯ К ОТЧЕТУ

Структура отчета

1. Титульный лист
2. Содержание
3. Введение (цель работы, краткое описание проекта)
4. Диаграмма вариантов использования (Use Case)
   4.1. Список акторов с описанием
   4.2. Диаграмма (вставка изображения)
   4.3. Пояснения к диаграмме (минимум 3 пояснения)
5. Объектная модель предметной области (Domain Model)
   5.1. Описание методологии построения

5.2. Диаграмма классов (вставка изображения)

5.3. Описание ключевых сущностей (3-5 сущностей)

5.4. Бизнес-правила и ограничения (минимум 3 правила)

6. Глоссарий предметной области (20+ терминов)

7. Таблица трассировки требований

8. Заключение (выводы, связь с последующими работами)

9. Список использованных источников

10. Приложения (исходный код диаграмм PlantUML)

Требования к оформлению

Элемент	Требование
Формат файла	PDF
Объём	10-15 страниц
Диаграммы	Векторный формат (PNG/SVG), читаемые подписи
Имя файла	Фамилия_Имя_ЛР2_ПИЖ.pdf

8. КРИТЕРИИ ОЦЕНИВАНИЯ

Критерий	**Макс. балл**	**Описание**
Use Case диаграмма	4	Корректность акторов, прецедентов, отношений (include/extend)
Domain Model	6	Полнота сущностей (5-7), корректность отношений, наличие

Критерий	Макс. балл	Описание
		бизнес-правил
Глоссарий	3	20+ терминов, чёткие определения, указаны связи
Таблица трассировки	3	Связь бизнес-требований с системными Use Cases
Пояснения	3	Ясность, полнота, грамотность
Связь с ЛР1	2	Чёткая трассировка от бизнес-модели к системным требованиям
Оформление отчета	2	Соответствие требованиям
Ответы на контрольные вопросы	2	Понимание материала при защите
ИТОГО	25	

Лабораторная работа №3

Проектирование архитектурного каркаса

Цель работы: Спроектировать многослойную архитектуру программной системы на основе паттерна PCMEF с учётом выбранной траектории разработки (Desktop/Web/Mobile).

1. Теоретическое обоснование

1.1. Построение архитектуры на основе объектной модели

В лабораторной работе №2 вы построили модель предметной области (Domain Model) - статическое представление ключевых сущностей и их отношений. Однако Domain Model не отвечает на вопросы:

- Как эти сущности будут взаимодействовать?
- Как организовать код, чтобы его можно было тестировать?
- Как отделить пользовательский интерфейс от бизнес-логики?
- Как обеспечить возможность замены базы данных или интерфейса?

Эти вопросы решает архитектура программного обеспечения.

Архитектура ПО - это набор структурных решений, которые определяют:

Аспект	Описание
Компоненты	Крупные блоки системы (пакеты, модули, сервисы)
Связи	Как компоненты взаимодействуют друг с другом
Зависимости	Кто от кого зависит и в каком направлении
Границы	Что находится внутри системы, а что снаружи

Хорошая архитектура должна обеспечивать:

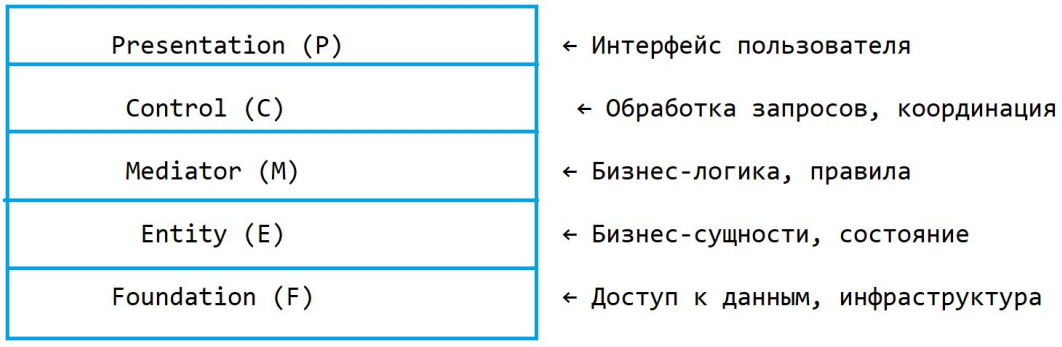
1. Разделение ответственности - каждый компонент решает одну задачу
2. Слабое зацепление - компоненты можно менять независимо
3. Высокую связность - внутри компонента всё связано по смыслу

4. Тестируемость - возможность тестировать компоненты изолированно

5. Поддерживаемость - лёгкость внесения изменений

1.3. Архитектурная концепция - PCMEF

1.3.1. Классическая структура PCMEF (монолит)



Правило зависимостей: Зависимости направлены строго вниз. Вышележащие слои могут зависеть от нижележащих, но не наоборот. Это обеспечивает стабильность архитектуры.

1.4. Адаптация PCMEF под разные траектории

PCMEF это единый концептуальный паттерн, который по-разному реализуется в разных средах:

Слой	**Классический (Desktop)**	**Веб-приложение**	**Мобильное приложение (клиент-сервер)**
P	JVM (Swing/JavaFX)	В браузере (HTML/JS)	На клиенте (Activity/Widgets)
C	JVM (Swing/JavaFX)	На сервере	На сервере

		(Controllers)	(REST Controllers)
M	JVM (Swing/JavaFX)	На сервере (Services)	На сервере (Services)
E	JVM (Swing/JavaFX)	На сервере (JPA Entities)	На сервере (JPA Entities)
F	В том же JVM	На сервере (Repositories)	На сервере (Repositories)
Доп. слои	--	-	На клиенте: State Management, API Client, Local Cache

Важнейший принцип: Несмотря на разное физическое расположение слоёв, логика их взаимодействия и направление зависимостей остаются теми же

2. Методика выполнения работы

Шаг 1. Анализ исходных данных

Возьмите результаты лабораторные работы 1 и 2:

- Use Case диаграмму - чтобы понять, какие сценарии должна поддерживать система

- Domain Model - чтобы определить ключевые сущности

- Спецификации прецедентов - чтобы понять последовательность действий

Вопросы для анализа:

- Какие акторы взаимодействуют с системой?

- Какие данные должны сохраняться?

- Какие бизнес-правила нужно реализовать?

- Какие внешние системы будут интегрироваться?

Шаг 2: Выбор и обоснование архитектурного стиля

Для всех траекторий базовым стилем является PCMEF. Однако в зависимости от траектории вы должны обосновать, как именно PCMEF адаптируется:

Примерные варианты обоснования.

Для траектории А (Desktop):

Выбран классический монолитный PCMEF, так как приложение работает на одном компьютере пользователя, не требует сетевого взаимодействия и может быть реализовано в рамках одного JVM-процесса. Все пять слоёв будут расположены в одном проекте, но разделены на пакеты для обеспечения модульности.

Для траектории Б (Web):

Выбран распределённый PCMEF с тонким клиентом. Presentation-слой реализуется в браузере на HTML/CSS/JavaScript, остальные слои - на сервере. Взаимодействие через HTTP-запросы. Такая архитектура обеспечивает централизованное хранение данных и возможность доступа с любого устройства.

Для траектории В (Mobile):

Выбран распределённый PCMEF в конфигурации клиент-сервер. Presentation-слой вынесен на мобильное устройство и дополнен слоями State Management, API Client и Local Cache для обеспечения оффлайн-режима и реактивного UI. Control, Mediator, Entity и Foundation реализованы на сервере. Взаимодействие через REST API.

Для траектории Г (Enterprise):

Выбран полный стек PCMEF с обоими типами интерфейсов (Desktop и Web) и REST API для интеграции. Такая архитектура обеспечивает максимальную гибкость и возможность расширения системы в будущем.

Структура обоснования:

1. Краткая характеристика выбранной траектории
2. Почему PCMEF подходит для данной траектории
3. Какие модификации/адаптации вносятся
4. Какие альтернативы рассматривались и почему отклонены

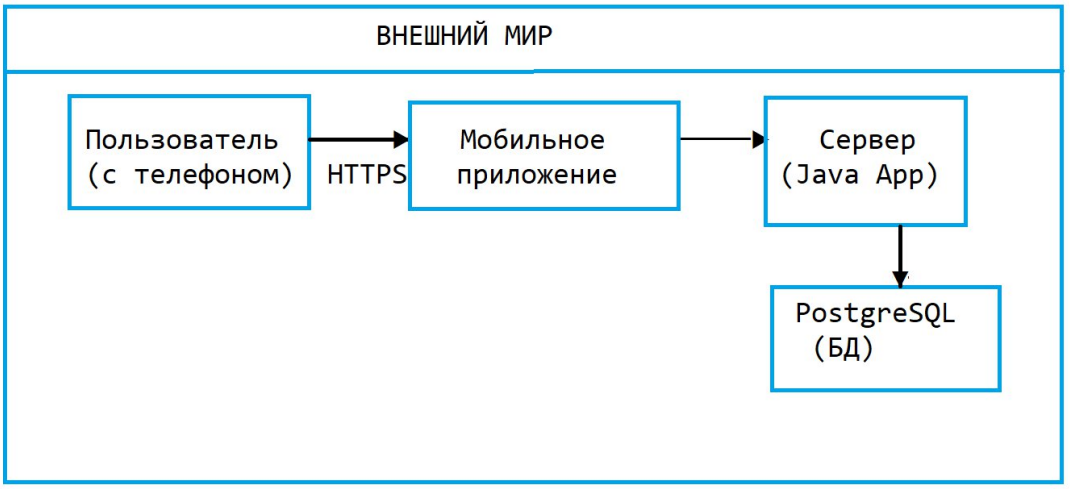
Шаг 3: Определение границ системы

Уточните контекстную диаграмму из лабораторной работы 1, добавив технические детали:

Что должно быть отражено:

- Система как единое целое
- Внешние системы (платёжные шлюзы, почтовые серверы, внешние API)
- Пользователи с указанием способов взаимодействия
- Протоколы взаимодействия (HTTP, SMTP, JDBC)

Пример для мобильной траектории:



Шаг 4: Построение диаграммы пакетов (ключевой шаг)

4.1. Общие требования к диаграмме пакетов

Диаграмма пакетов должна:

- Показывать все архитектурные слои в соответствии с выбранной траекторией
- Демонстрировать направление зависимостей (стрелками)
- Содержать краткие пояснения ответственности каждого пакета
- Быть читаемой (не перегруженной деталями)

4.2. Шаблоны диаграмм по траекториям

Для всех траекторий мы будем использовать PlantUML (https://www.plantuml.com/plantuml/uml/)

Для траектории A (Desktop — монолит):

@startuml

!define RECTANGLE rectangle

package "Desktop Application" {

```
package presentation {
  [MainFrame]
  [Dialogs]
  [Forms]
  note top of presentation
    Отвечает за отображение и ввод данных
  end note
}

package control {
  [MainController]
  [AuthController]
  [EntityController]
  note top of control
    Координирует потоки данных, вызывает медиаторы
  end note
}

package mediator {
  [UserService]
  [OrderService]
  [ReportService]
  note top of mediator
    Содержит бизнес-логику, транзакции
  end note
}

package entity {
  [User]
```

```
  [Order]
  [Product]
  note top of entity
    Классы-сущности с бизнес-методами
  end note
}

package foundation {
  [UserRepository]
  [OrderRepository]
  [DatabaseConnection]
  note top of foundation
    Доступ к данным, DAO, работа с БД
  end note
}

' Зависимости (только вниз!)
presentation --> control
control --> mediator
mediator --> entity
mediator --> foundation
entity --> foundation

' Пакет acquaintance (общие интерфейсы) - опционально
package acquaintance {
  [IUser]
  [IOrder]
  note top of acquaintance
    Интерфейсы для слабой связанности
```

```
    end note
  }

  presentation --> acquaintance
  entity --> acquaintance
}
@enduml
```

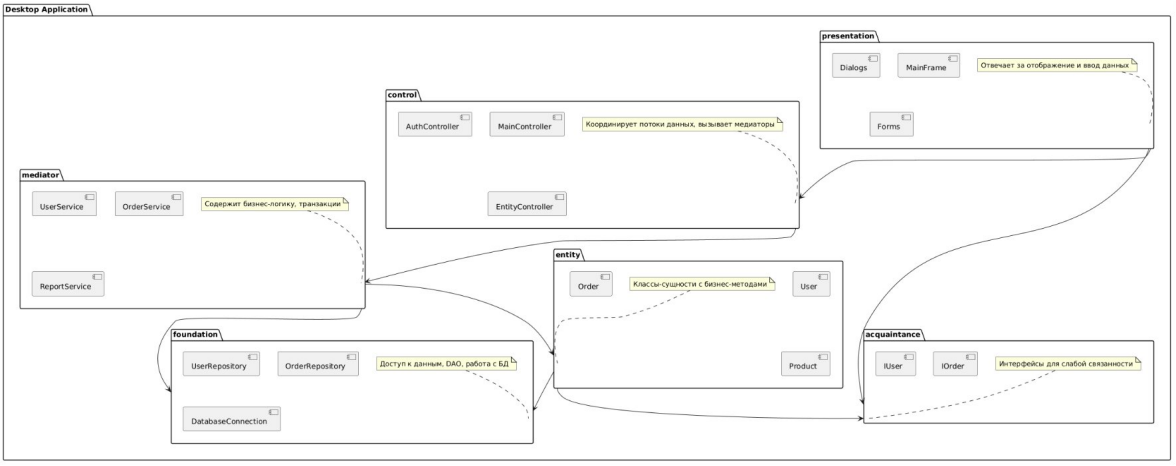


Рисунок 1

Для траектории Б (Web - тонкий клиент):

```
@startuml
!define RECTANGLE rectangle

left to right direction
title Пример архитектуры веб-приложения (Browser + Spring Boot)

package "Browser (Клиент)" {
    package "presentation" {
        [HTML Pages]
        [CSS Styles]
        [JavaScript]
```

```
        note right of [HTML Pages] : Отображение, валидация на клиенте
    }
}

package "Server (Java Application)" {

  package "control" {
    [UserController]
    [OrderController]
    [AuthController]
    note right of [UserController] : REST-контроллеры, @RestController
  }

  package "mediator" {
    [UserService]
    [OrderService]
    note right of [UserService] : Бизнес-логика, @Service
  }

  package "entity" {
    [User]
    [Order]
    [Product]
    note right of [User] : JPA-сущности, @Entity
  }

  package "foundation" {
    [UserRepository]
    [OrderRepository]
```

```
    note right of [UserRepository] : Spring Data JPA, @Repository
  }

  package "dto" {
    [UserRequest]
    [UserResponse]
    note right of [UserRequest] : Data Transfer Objects для API
  }

  presentation --> control : HTTP/JSON
  control --> dto : использует
  control --> mediator : вызывает
  mediator --> entity : управляет
  mediator --> foundation : использует
  entity --> foundation : сохраняется через
}

package "Database" {
  [PostgreSQL]
  note right of [PostgreSQL] : Реляционная БД
}

foundation --> Database : JDBC

@enduml
```

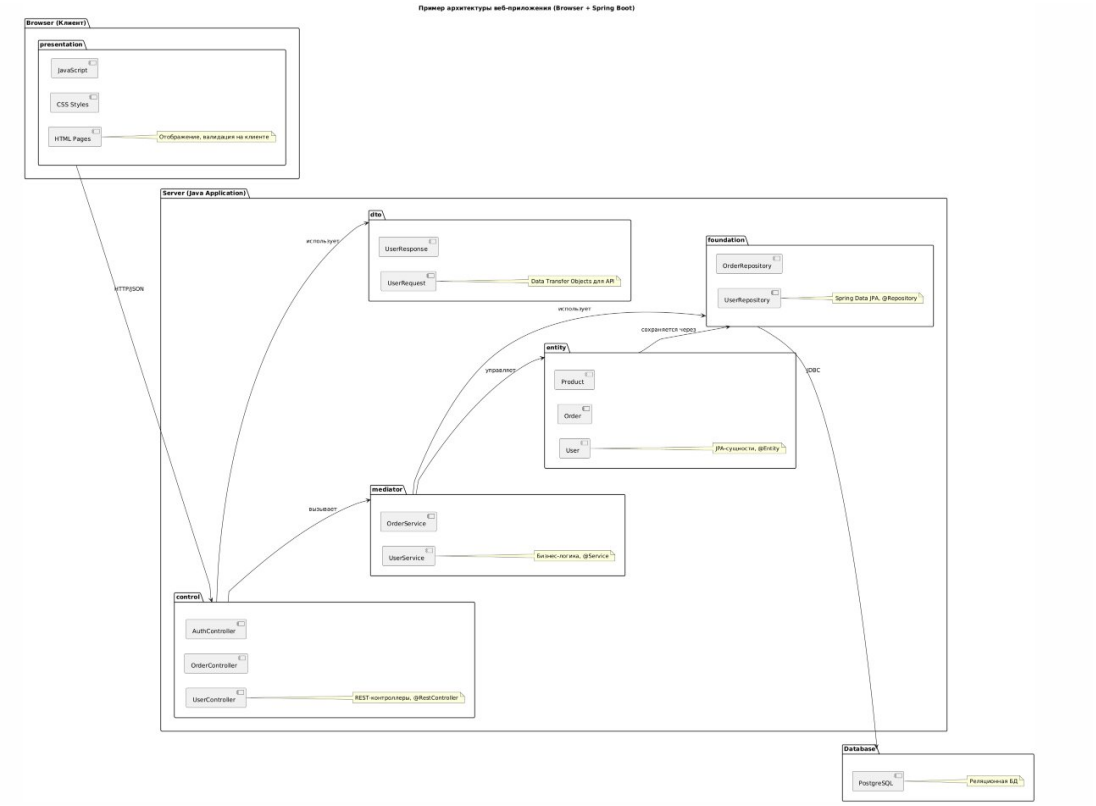


Рисунок 2

Для траектории B (Mobile — клиент-сервер):

```
@startuml
!define RECTANGLE rectangle

left to right direction

package "Mobile Client" {

  package "presentation" {
    [Activities/Fragments]
    [Compose Widgets]
    [XML Layouts]
    note top of presentation
      UI, отрисовка, навигация
    end note
```

```
}

package "state_management" {
  [ViewModels]
  [States]
  [Events]
  note top of state_management
    Хранит состояние экрана, реакция на события
  end note
}

package "api_client" {
  [Retrofit Service]
  [API Interfaces]
  [DTOs]
  note top of api_client
    HTTP-запросы, сериализация JSON
  end note
}

package "local_cache" {
  [Room Database]
  [DAOs]
  [Entities (local)]
  note top of local_cache
    Кэширование, оффлайн-режим
  end note
}
```

```
  ' Зависимости на клиенте
  presentation --> state_management
  state_management --> api_client
  state_management --> local_cache
}

package "Server (Java)" {

  package "control" {
    [REST Controllers]
    note top of control
      @RestController, входная точка
    end note
  }

  package "mediator" {
    [Services]
    note top of mediator
      Бизнес-логика, @Service
    end note
  }

  package "entity" {
    [JPA Entities]
    note top of entity
      Бизнес-сущности, @Entity
    end note
  }
```

```
  package "foundation" {
    [Repositories]
    note top of foundation
      Доступ к данным, @Repository
    end note
  }
}

package "Database" {
  [PostgreSQL]
}

' Межпроцессное взаимодействие
api_client --> control : REST/JSON/HTTPS

' Зависимости на сервере
control --> mediator
mediator --> entity
mediator --> foundation
entity --> foundation
foundation --> Database

' Примечание: Entity на клиенте (local) и на сервере - разные!
note right of local_cache
  Локальные entity (Room) могут
  отличаться от серверных JPA-entity
end note
@enduml
```

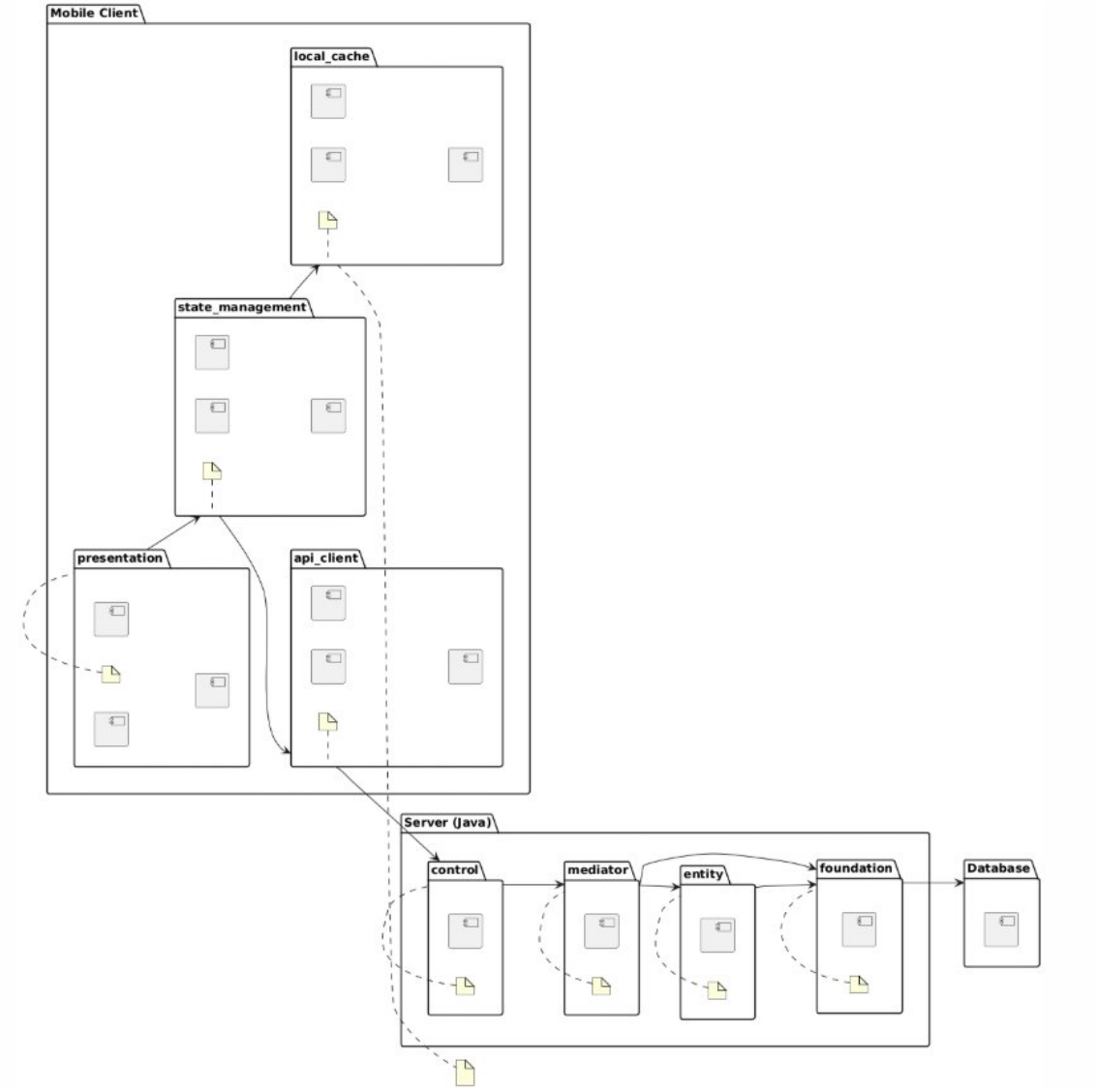


Рисунок 3

4.3. Что должно быть на диаграмме обязательно:

1. Все пакеты согласно выбранной траектории
2. Стрелки зависимостей с чёткими направлениями
3. Краткие пояснения (ноты) для каждого пакета
4. Границы системы (что внутри, что снаружи)
5. Внешние системы (БД, внешние API)

Шаг 5: Определение и визуализация зависимостей

После построения диаграммы пакетов необходимо детализировать зависимости:

5.1. Типы зависимостей

Тип зависимости	**Обозначение**	**Пример**	**Пояснение**
Прямая зависимость	A -> B	Controller зависит от Service	A использует B напрямую
Интерфейсная зависимость	A ---> I <|.. B	Controller зависит от интерфейса I, B реализует I	Слабая связанность
Ассоциация	A -- B	Client и Server	Постоянная связь
Зависимость через границу	A ---> B : HTTP/JSON	Клиент и сервер	Сетевой протокол

5.2. Спецификация интерфейсов между слоями

Общий подход к визуализации зависимостей

Для всех траекторий необходимо использовать PlantUML для создания диаграмм, показывающих:

1. Направление зависимостей между слоями
2. Типы связей (вызов методов, HTTP-запросы, JDBC-соединения)
3. Интерфейсы взаимодействия между компонентами
4. Потоки данных в системе

5.2.1. Диаграмма зависимостей для Desktop

Для каждой ключевой зависимости необходимо определить **контракт взаимодействия**:

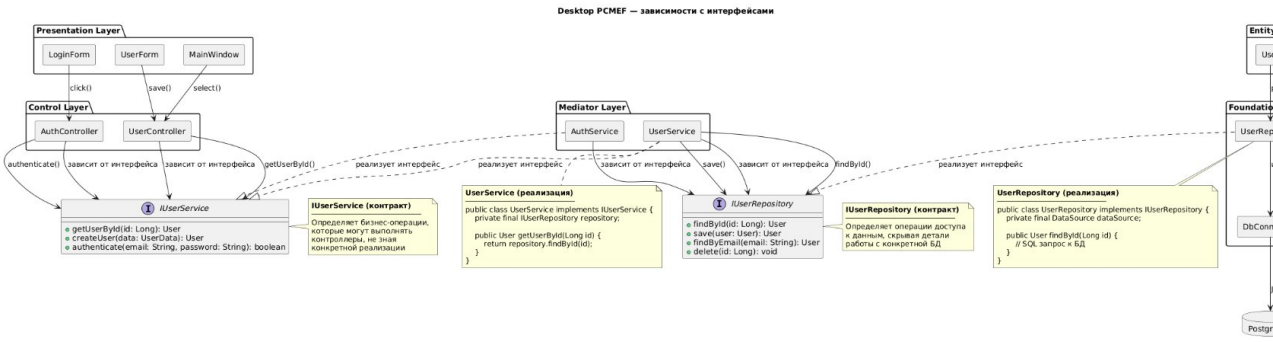
@startuml

```
allowmixing
title Desktop PCMEF - зависимости с интерфейсами

' ===== ИНТЕРФЕЙСЫ =====
interface "IUserService" as IUS {
   +getUserById(id: Long): User
   +createUser(data: UserData): User
   +authenticate(email: String, password: String): boolean
}

interface "IUserRepository" as IUR {
   +findById(id: Long): User
   +save(user: User): User
   +findByEmail(email: String): User
   +delete(id: Long): void
}

' ===== СЛОИ =====
package "Presentation Layer" {
   rectangle "LoginForm" as LoginForm
   rectangle "MainWindow" as MainWindow
   rectangle "UserForm" as UserForm
}

package "Control Layer" {
   rectangle "AuthController" as AuthCtrl
   rectangle "UserController" as UserCtrl

   AuthCtrl --> IUS : "зависит от интерфейса"
```

```
    UserCtrl --> IUS : "зависит от интерфейса"
}

package "Mediator Layer" {
    rectangle "UserService" as UserSvc
    rectangle "AuthService" as AuthSvc

    UserSvc ..|> IUS : "реализует интерфейс"
    AuthSvc ..|> IUS : "реализует интерфейс"

    UserSvc --> IUR : "зависит от интерфейса"
    AuthSvc --> IUR : "зависит от интерфейса"
}

package "Entity Layer" {
    rectangle "User" as UserEnt
    rectangle "Role" as RoleEnt
}

package "Foundation Layer" {
    rectangle "UserRepository" as UserRepo
    rectangle "DbConnection" as DbConn

    UserRepo ..|> IUR : "реализует интерфейс"
    UserRepo --> DbConn : "использует"
}

database "PostgreSQL" as DB
```

```
' ===== СВЯЗИ МЕЖДУ СЛОЯМИ =====
LoginForm --> AuthCtrl : "click()"
MainWindow --> UserCtrl : "select()"
UserForm --> UserCtrl : "save()"

AuthCtrl --> IUS : "authenticate()"
UserCtrl --> IUS : "getUserById()"

UserSvc --> IUR : "findById()"
UserSvc --> IUR : "save()"

UserEnt --> UserRepo : "persists via"

DbConn --> DB : "JDBC"

' ===== ПОЯСНЕНИЯ =====
note right of IUS
   **IUserService (контракт)**
   Определяет бизнес-операции,
   которые могут выполнять
   контроллеры, не зная
   конкретной реализации
end note

note right of IUR
   **IUserRepository (контракт)**
   Определяет операции доступа
   к данным, скрывая детали
   работы с конкретной БД
```

```
end note

note bottom of UserSvc
   **UserService (реализация)**
   public class UserService implements IUserService {
      private final IUserRepository repository;

      public User getUserById(Long id) {
         return repository.findById(id);
      }
   }
end note

note bottom of UserRepo
   **UserRepository (реализация)**
   public class UserRepository implements IUserRepository {
      private final DataSource dataSource;

      public User findById(Long id) {
         // SQL запрос к БД
      }
   }
end note
@enduml
```



5.2. Таблица зависимостей для Desktop (с интерфейсами)

От	Через что	К	Тип связи	Направление	Описание
Presentation	Прямая ссылка	Control	Ассоциация	–	UI вызывает методы контроллера
Control	Интерфейс IUserService	Mediator	Зависимость от абстракции	→	Контроллер знает только интерфейс сервиса
Mediator (Service)	Реализация	IUserService	Реализация интерфейса	←	Сервис реализует контракт
Mediator (Service)	Интерфейс IUserRepository	Foundation	Зависимость от абстракции	→	Сервис использует репозиторий через интерфейс
Foundation (Reposito	Реализация	IUserRepository	Реализация интерфейса	←	Репозиторий реализует контракт

От	Через что	К	Тип связи	Направление	Описание
ry)					
Foundation (Repository)	Прямая ссылка	DbConnection	Ассоциация	→	Репозиторий использует соединение
DbConnection	JDBC	Database	Зависимость	→	Соединение с БД
Entity	Прямая ссылка	Foundation	Ассоциация	→	Сущность пер

Для каждой ключевой зависимости необходимо определить контракт взаимодействия:

Пример для мобильной траектории:

```
@startuml
left to right direction

rectangle "state_management" {
  component state
}

rectangle "api_client" {
  interface UserApi
}

rectangle "local_cache" {
```

```
  interface UserDao
}

state --> UserApi
state --> UserDao

@enduml
```

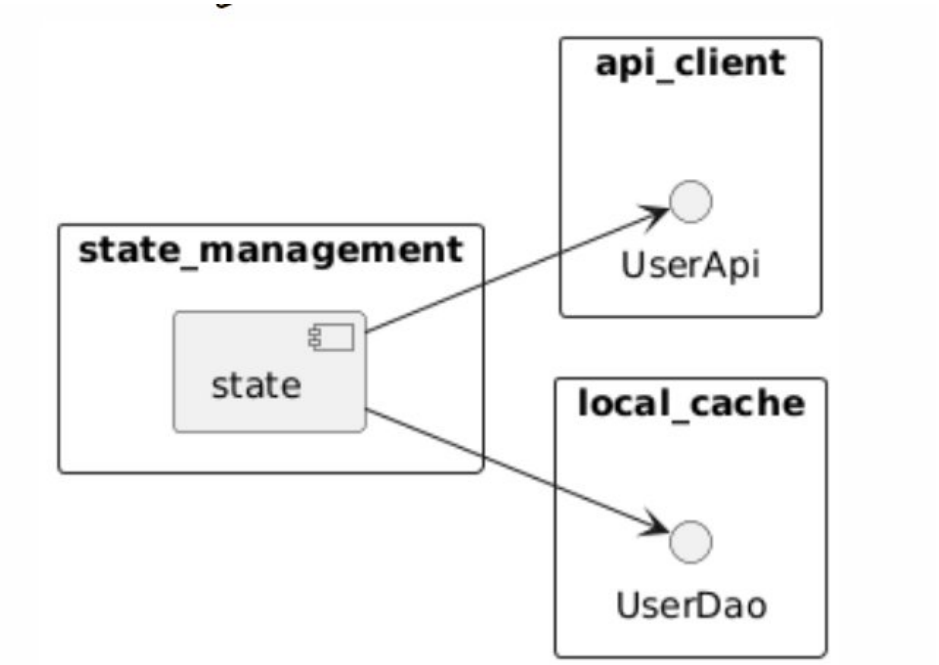


Рисунок 4

5.3. Проверка циклических зависимостей

Критическое правило: В диаграмме пакетов НЕ ДОЛЖНО БЫТЬ ЦИКЛИЧЕСКИХ ЗАВИСИМОСТЕЙ!

Неверно: presentation → control → mediator → presentation

Верно: presentation → control → mediator → entity → foundation

Как проверить:

1. Мысленно пройдите по стрелкам

2. Если можно вернуться в исходный пакет, не меняя направления - это цикл

3. Циклы устраняются введением интерфейсов или реорганизацией пакетов

Шаг 6: Оформление архитектурного документа

2.3. Структура архитектурного документа

АРХИТЕКТУРНЫЙ ДОКУМЕНТ

Проект: [Название проекта]

Версия: 1.0

Дата: [Дата]

Автор: [ФИО]

Траектория: [Desktop / Web / Mobile / Enterprise]

1. ВВЕДЕНИЕ

1.1. Цель документа

1.2. Область применения

1.3. Определения и сокращения

1.4. Ссылочные документы (ЛР1, ЛР2)

2. КОНТЕКСТ СИСТЕМЫ

2.1. Бизнес-контекст (кратко из ЛР1)

2.2. Технический контекст (внешние системы, интеграции)

2.3. Диаграмма контекста (уточнённая)

3. АРХИТЕКТУРНЫЙ СТИЛЬ

3.1. Выбор архитектурного стиля

3.2. Обоснование выбора PCMEF

3.3. Адаптация PCMEF под выбранную траекторию

3.4. Рассмотренные альтернативы и причины отказа

4. СТРУКТУРА СИСТЕМЫ

4.1. Общая диаграмма пакетов (вставка изображения)

4.2. Описание слоёв и пакетов

4.2.1. Presentation-слой

4.2.2. Control-слой (или State Management для Mobile)

4.2.3. Mediator-слой (Services)

4.2.4. Entity-слой

4.2.5. Foundation-слой

4.2.6. Дополнительные слои (API Client, Local Cache, DTO)

5. ЗАВИСИМОСТИ И ВЗАИМОДЕЙСТВИЯ

5.1. Правила зависимостей между слоями

5.2. Диаграмма зависимостей с пояснениями

5.3. Спецификация ключевых интерфейсов

5.3.1. Интерфейс между Presentation и Control

5.3.2. Интерфейс между Control и Mediator

5.3.3. Интерфейс между Mediator и Foundation

5.4. Обработка циклических зависимостей (если есть)

6. РАЗВЁРТЫВАНИЕ (ДЛЯ РАСПРЕДЕЛЁННЫХ СИСТЕМ)

6.1. Диаграмма развёртывания (deployment diagram)

6.2. Требования к окружению

6.3. Конфигурация взаимодействия

7. ОБОСНОВАНИЕ АРХИТЕКТУРНЫХ РЕШЕНИЙ

7.1. Ключевые ADR (Architecture Decision Records)

7.2. Компромиссы и допущения

8. ЗАКЛЮЧЕНИЕ

8.1. Выводы

8.2. План перехода к реализации (ЛР4+)

ПРИЛОЖЕНИЯ

Приложение А. Исходный код диаграмм (PlantUML)

3. Пример выполнения для Event Management System

3.1. Исходные данные (из ЛР1-ЛР2)

Проект: Event Management System

Траектория: Mobile (Android + Java Backend)

Ключевые требования из ЛР2:

- UC-001: Создать мероприятие
- UC-004: Зарегистрироваться на мероприятие
- UC-006: Оплатить участие

Domain Model: Содержит классы Event, User, Registration, Payment, Session

3.2. Обоснование выбора архитектурного стиля

3.1. Выбор архитектурного стиля

Для проекта Event Management System выбран архитектурный паттерн PCMEF

в его клиент-серверной адаптации, предназначенной для мобильных приложений.

Обоснование выбора:

1. Разделение ответственности: Система требует чёткого разделения между

интерфейсом пользователя (мобильное приложение), бизнес-логикой (сервер) и хранением данных (БД). PCMEF обеспечивает это разделение на уровне

концептуальных слоёв.

2. Масштабируемость: В будущем система может быть расширена веб-интерфейсом или публичным API. Распределённая реализация PCMEF позволяет добавлять новые типы клиентов без изменения серверной части.

3. Тестируемость: Бизнес-логика (Mediator, Entity) может тестироваться

изолированно от инфраструктуры (Foundation) и интерфейса.

4. Соответствие требованиям курса: Методические указания предписывают

использование PCMEF как единой архитектурной основы.

Адаптация PCMEF под мобильную траекторию:

В соответствии с Лекцией 1, слои распределяются следующим образом:

- Presentation (с доп. слоями State Management, API Client, Local Cache) - на клиенте

- Control, Mediator, Entity, Foundation - на сервере

Рассмотренные альтернативы:

1. Традиционный MVC: Отвергнут, так как не обеспечивает достаточной изоляции бизнес-логики и ведёт к "толстым" контроллерам.

2. Clean Architecture: Более сложная для реализации в рамках учебного проекта, хотя концептуально близка к PCMEF. Оставлена для траектории Enterprise.

3. Только мобильное приложение с SQLite: Невозможна, так как противоречит

методическим указаниям (требуется серверная часть).

3.3. Диаграмма пакетов

PlantUML код:

```
@startuml
left to right direction
skinparam packageStyle rectangle
skinparam componentStyle rectangle

title Архитектура Event Management System (Mobile-Desktop)

package "Mobile Client (Android/Kotlin)" {

  package "presentation" <<P>> {
    class MainActivity
```

```
  class EventDetailActivity
  class RegistrationActivity
  class "Compose UI"

  note "Отвечает за отображение и навигацию" as N1
  MainActivity .. N1
  EventDetailActivity .. N1
  RegistrationActivity .. N1
  "Compose UI" .. N1
}

package "state_management" {
  class EventViewModel
  class RegistrationViewModel
  class AuthViewModel
  class "UI State"
  class Events

  note "Хранит состояние экранов, LiveData/StateFlow" as N2
  EventViewModel .. N2
  RegistrationViewModel .. N2
  AuthViewModel .. N2
  "UI State" .. N2
  Events .. N2
}

package "api_client" {
  interface EventApi
  interface UserApi
```

```
  class RetrofitClient
  class AuthInterceptor
  class EventDto
  class UserDto

  note "HTTP-запросы, JSON сериализация, JWT" as N3
  EventApi .. N3
  UserApi .. N3
  RetrofitClient .. N3
  AuthInterceptor .. N3
  EventDto .. N3
  UserDto .. N3
}

package "local_cache" {
  class AppDatabase
  class EventDao
  class UserDao
  class LocalEvent
  class LocalUser

  note "Room, кэширование, оффлайн-режим" as N4
  AppDatabase .. N4
  EventDao .. N4
  UserDao .. N4
  LocalEvent .. N4
  LocalUser .. N4
}
```

```
  ' Зависимости на клиенте
  presentation --> state_management : "наблюдает"
  state_management --> api_client : "вызывает"
  state_management --> local_cache : "читает/пишет"
}

package "Server (Java + Spring Boot)" {

  package "control" <<C>> {
    class EventController <<@RestController>>
    class UserController <<@RestController>>
    class RegistrationController <<@RestController>>
    class PaymentController <<@RestController>>

    note "REST-контроллеры, валидация DTO" as N5
    EventController .. N5
    UserController .. N5
    RegistrationController .. N5
    PaymentController .. N5
  }

  package "dto" {
    class EventRequest
    class EventResponse
    class UserRequest
    class UserResponse
    class RegistrationRequest

    note "Data Transfer Objects для API" as N6
```

```
  EventRequest .. N6
  EventResponse .. N6
  UserRequest .. N6
  UserResponse .. N6
  RegistrationRequest .. N6
}

package "mediator" <<M>> {
  class EventService <<@Service>>
  class UserService <<@Service>>
  class RegistrationService <<@Service>>
  class PaymentService <<@Service>>

  note "Бизнес-логика, @Transactional" as N7
  EventService .. N7
  UserService .. N7
  RegistrationService .. N7
  PaymentService .. N7
}

package "entity" <<E>> {
  class Event <<@Entity>>
  class User <<@Entity>>
  class Registration <<@Entity>>
  class Payment <<@Entity>>
  class Session <<@Entity>>

  note "JPA-сущности, ORM" as N8
  Event .. N8
```

```
    User .. N8
    Registration .. N8
    Payment .. N8
    Session .. N8
  }

  package "foundation" <<F>> {
    class EventRepository <<@Repository>>
    class UserRepository <<@Repository>>
    class RegistrationRepository <<@Repository>>
    class PaymentRepository <<@Repository>>

    note "Spring Data JPA, доступ к данным" as N9
    EventRepository .. N9
    UserRepository .. N9
    RegistrationRepository .. N9
    PaymentRepository .. N9
  }

  ' Зависимости на сервере
  control --> dto
  control --> mediator
  mediator --> entity
  mediator --> foundation
  entity --> foundation
}

database "PostgreSQL" {
  class events
```

```
  class users
  class registrations
  class payments
  class sessions

  note "Реляционная БД, ACID" as N10
  events .. N10
  users .. N10
  registrations .. N10
  payments .. N10
  sessions .. N10
}

' Межпроцессное взаимодействие
api_client --> control : "REST/JSON/HTTPS"

foundation --> PostgreSQL : "JDBC"

' Пояснения
note top of api_client
  Единственный слой на клиенте,
  который знает о сервере
end note

note bottom of mediator
  Здесь реализованы бизнес-правила:
  - проверка доступности мест
  - расчёт стоимости
  - валидация дат
```

```
end note

note right of local_cache
  Обеспечивает работу приложения
  без интернета (оффлайн-режим)
end note

@enduml
```

3.4. Описание зависимостей

Таблица ключевых зависимостей:

От	К	Тип	Описание
presentation	state_management	Наблюдение	UI подписывается на изменения состояния
state_management	api_client	Вызов	ViewModel вызывает методы API при действиях пользователя
state_management	local_cache	Чтение/Запись	Кэширование данных, оффлайн-режим
api_client	control	HTTP/REST	Сетевые запросы к серверу
control	mediator	Вызов метода	Контроллеры вызывают сервисы
mediator	entity	Чтение/Изменение	Сервисы работают

			с сущностями
mediator	foundation	Вызов	Сервисы вызывают репозитории для доступа к данным
foundation	PostgreSQL	JDBC	Репозитории выполняют SQL-запросы

3.5. Спецификация ключевых интерфейсов

Интерфейс между state_management и api_client:

```
// API Client (Kotlin)
interface EventApi {
    @GET("events")
    suspend fun getEvents(): List<EventDto>

    @GET("events/{id}")
    suspend fun getEvent(@Path("id") id: Long): EventDto

    @POST("events")
    suspend fun createEvent(@Body event: EventCreateRequest): EventResponse

    @POST("events/{id}/register")
    suspend fun register(
        @Path("id") eventId: Long,
        @Body request: RegistrationRequest
```

```
    ): RegistrationResponse
}
```

Интерфейс между control и mediator (Java):

```
// Service Interface (Java)
public interface EventService {
    EventDto createEvent(EventCreateRequest request);
    List<EventDto> getAllEvents();
    EventDto getEventById(Long id);
    RegistrationDto registerForEvent(Long eventId, RegistrationRequest request);
    boolean checkAvailability(Long eventId);
}
```

Интерфейс между mediator и foundation:

```
// Repository Interface (Java)
public interface EventRepository extends JpaRepository<Event, Long> {
    List<Event> findByStartDateAfter(LocalDateTime date);
    Optional<Event> findByTitleAndStartDate(String title, LocalDateTime startDate);

    @Query("SELECT e FROM Event e WHERE e.status = :status")
    List<Event> findByStatus(@Param("status") EventStatus status);
}
```

Отчет по лабораторной работе должен содержать:

- Цель лабораторной работы
- Краткое описание проекта (2-3 предложения)
- Перечень решаемых задач (4-6 пунктов)

Анализ исходных данных

- Краткая характеристика проекта из ЛР1
- Ключевые бизнес-процессы (из ЛР1)
- Основные акторы системы (из ЛР2)
- Ключевые Use Cases (из ЛР2)
- Основные сущности Domain Model (из ЛР2)
- Таблица исходных артефактов со ссылками на ЛР1-ЛР2

Выбор и обоснование архитектурного стиля

- Обоснование выбора PCMEF (3-5 причин)
- Рассмотренные альтернативы (минимум 2)
- Причины отказа от альтернатив
- Таблица распределения слоёв PCMEF для выбранной траектории

Диаграмма пакетов PCMEF

- Графическое представление диаграммы пакетов (PlantUML/Draw.io)
- Описание каждого слоя (назначение, ключевые классы)
- Для распределённых систем - разделение на клиент и сервер
- Пояснения к диаграмме

Спецификация интерфейсов между слоями

- Интерфейс Control → Mediator (I*Service)
- Интерфейс Mediator → Foundation (I*Repository)
- Другие интерфейсы (при наличии)
- Таблица интерфейсов с описанием методов

Анализ зависимостей

- Таблица зависимостей между слоями
- Граф зависимостей в текстовом виде
- Проверка отсутствия циклических зависимостей
- Вывод о корректности направленности зависимостей

Адаптация PCMEF под выбранную траекторию

- Особенности адаптации для конкретной траектории
- Сравнительная таблица с классическим PCMEF
- Обоснование необходимости адаптации
- Дополнительные слои (при наличии)

ПРИЛОЖЕНИЯ

- Исходный код диаграмм (PlantUML)
- Дополнительные схемы (при наличии)
- Скриншоты (при необходимости)

Лабораторная работа №4

Проектирование базы данных - ER-диаграмма и нормализация

1. Цель работы и задачи

Цель: Освоить методику проектирования реляционной базы данных на основе объектной модели предметной области, включая построение ER-диаграммы и нормализацию до третьей нормальной формы (3НФ).

Задачи:

1. Преобразовать Domain Model (из ЛР2) в логическую модель данных (ER-диаграмму)

2. Выделить все сущности, атрибуты и связи
3. Определить первичные и внешние ключи
4. Выполнить нормализацию таблиц до 3 нормальной формы (3НФ)
5. Устранить избыточность и потенциальные аномалии модификации данных
6. Подготовить спецификацию связей между таблицами

2. ТЕОРЕТИЧЕСКОЕ ОБОСНОВАНИЕ

2.1. Место проектирования БД в жизненном цикле

Проектирование базы данных - это этап, на котором объектная модель предметной области (Domain Model) преобразуется в структуру, пригодную для хранения в реляционной СУБД.

Domain Model	
[Шаг 1]	Выделение сущностей и атрибутов
[Шаг 2]	Определение первичных ключей
[Шаг 3]	Нормализация (1НФ → 2НФ → 3НФ)
[Шаг 4]	Определение связей и внешних ключей
	ER-диаграмма (логическая модель)
	DDL-скрипты (ЛР5)

2.2. От Domain Model к реляционной модели

Модель	**Назначение**	**Элементы**
Domain Model (ЛР2)	Объектная модель предметной области	Классы, атрибуты, методы, ассоциации
ER-диаграмма	Логическая модель данных	Сущности, атрибуты, связи, ключи

Модель	Назначение	Элементы
Реляционная модель	Физическая модель данных	Таблицы, столбцы, типы данных, индексы

Ключевые отличия:

Domain Model (ООП)	Реляционная модель (БД)
Класс	Таблица
Объект	Строка (кортеж)
Атрибут	Столбец (поле)
Ссылка на объект	Внешний ключ
Наследование	Стратегии отображения
Методы	Хранимые процедуры (реже)

2.3. Сущности, атрибуты и связи

Сущность (Entity) - это объект реального мира, о котором необходимо хранить данные. В БД сущность становится таблицей.

Атрибут - это характеристика сущности. В БД атрибут становится столбцом таблицы.

Связь (Relationship) - это логическая связь между двумя или более сущностями.

Типы связей:

Тип связи	Обозначение	Пример	В реляционной БД
Один-к-одному (1:1)	1 —— 1	User ↔ Passport	Внешний ключ + UNIQUE
Один-ко-многим (1:N)	1 —— 0..*	User → Order	Внешний ключ в таблице «многие»
Многие-ко-многим (M:N)	0..* —— 0..*	Student ↔ Course	Связующая таблица

2.4. Первичные и внешние ключи

Первичный ключ (Primary Key) - это столбец (или комбинация столбцов), который уникально идентифицирует каждую строку в таблице.

Требования к первичному ключу:

1. Уникальность (нет двух одинаковых значений)
2. Непустота (NOT NULL)
3. Стабильность (не должен меняться со временем)

Типы первичных ключей:

Тип	Пример	Плюсы	Минусы
Суррогатный	id (автоинкремент)	Простота, стабильность	Не имеет бизнес-смысла
Естественный	email, passport_series	Понятен бизнесу	Может измениться

Внешний ключ (Foreign Key) - это столбец, который ссылается на первичный ключ другой таблицы.

2.5. Нормализация базы данных

Нормализация - это процесс организации данных в базе данных для уменьшения избыточности и предотвращения аномалий.

Аномалии модификации данных:

Аномалия	Описание	Пример
Вставки	Нельзя добавить данные без других данных	Нельзя добавить преподавателя без курса
Обновления	Изменение одних данных требует изменения в нескольких строках	Смена адреса филиала обновляется во всех заказах
Удаления	При удалении теряются нужные данные	Удаление курса удаляет информацию о преподавателе

2.5.1. Первая нормальная форма (1НФ)

Требования:

Все атрибуты атомарны (не содержат множественных значений)

Каждая запись уникальна (есть первичный ключ)

Нарушение 1НФ (рисунок 3):

Student	Courses
Иванов	Матан, Физика, Прог

← не атомарно!

Рисунок 3

Исправление:

Student	Course
Иванов Иванов Иванов	Матан Физика Прог

2.5.2. Вторая нормальная форма (2НФ)

Требования:

Находится в 1НФ

Все неключевые атрибуты зависят от полного первичного ключа (актуально для составных ключей)

Нарушение 2НФ (составной ключ (StudentID, CourseID)):

StudentID	CourseID	StudentName	CourseName
1 1 2	101 102 101	Иванов Иванов Петров	Матан Физика Матан

```
StudentName зависит только от StudentID (часть ключа) – нарушение!
CourseName зависит только от CourseID (часть ключа) – нарушение!
```

StudentName зависит только от StudentID (часть ключа) - нарушение!

CourseName зависит только от CourseID (часть ключа) - нарушение!

Исправление (разделение на 3 таблицы):

Students: (StudentID, StudentName)

Courses: (CourseID, CourseName)

Enroll: (StudentID, CourseID)

2.5.3. Третья нормальная форма (3НФ)

Требования:

Находится в 2НФ

Отсутствуют транзитивные зависимости (неключевой атрибут не зависит от другого неключевого)

Нарушение 3НФ:

OrderID	CustomerID	City	PostalCode
101 102	1 1	Москва Москва	101000 101000

```
PostalCode зависит от City (а City зависит от CustomerID) — транзитивная
зависимость!
```

PostalCode зависит от City (а City зависит от CustomerID) — транзитивная зависимость!

Исправление:

Orders: (OrderID, CustomerID)

Customers: (CustomerID, City, PostalCode)

2.6. Действия при удалении (Referential Actions)

Действие	**Описание**	**Когда использовать**
CASCADE	При удалении родителя удаляются все дети	Зависимые данные не имеют смысла без родителя
SET NULL	Внешний ключ устанавливается в	Связь опциональна

Действие	Описание	Когда использовать
	NULL	
RESTRICT	Запрещает удаление, если есть дети	Целостность критична
NO ACTION	Аналогично RESTRICT (в PostgreSQL)	

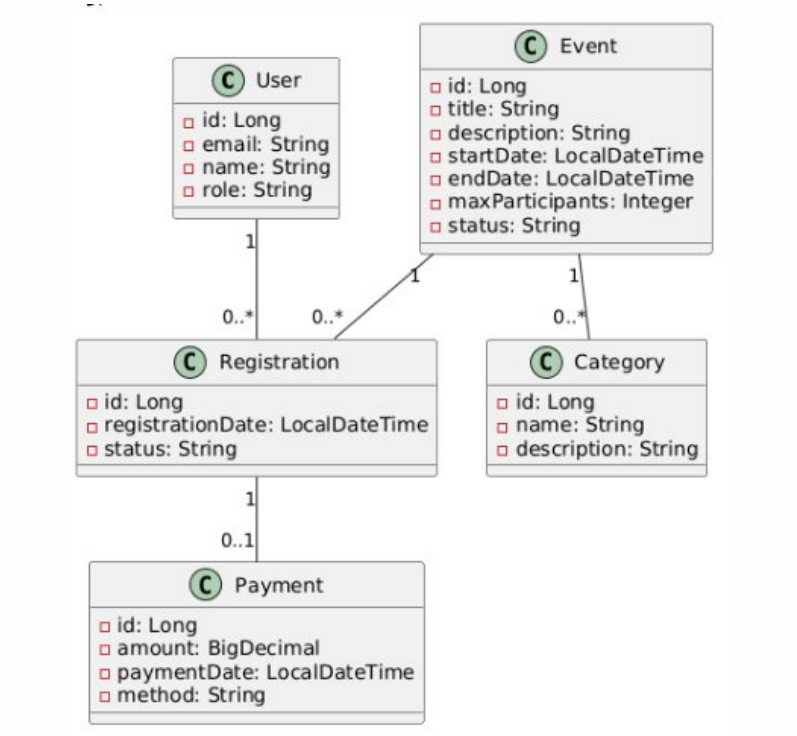
3. ПРИМЕР ВЫПОЛНЕНИЯ (СКВОЗНОЙ ПРОЕКТ)

Проект: «Система управления мероприятиями» (Event Management System)

3.1. Исходная Domain Model (из ЛР2)

```
@startuml
class User {
  - id: Long
  - email: String
  - name: String
  - role: String
}

class Event {
  - id: Long
  - title: String
  - description: String
  - startDate: LocalDateTime
  - endDate: LocalDateTime
```

```
  - maxParticipants: Integer
  - status: String
}

class Registration {
  - id: Long
  - registrationDate: LocalDateTime
  - status: String
}

class Payment {
  - id: Long
  - amount: BigDecimal
  - paymentDate: LocalDateTime
  - method: String
}

class Category {
  - id: Long
  - name: String
  - description: String
}

User "1" -- "0..*" Registration
Event "1" -- "0..*" Registration
Registration "1" -- "0..1" Payment
Event "1" -- "0..*" Category
@enduml
```



3.2. Шаг 1: Выделение сущностей и атрибутов

Таблица сущностей и атрибутов:

Сущность	Атрибуты	Типы данных (SQL)
User	id, email, password_hash, name, role, created_at	BIGINT, VARCHAR(255), VARCHAR(60), VARCHAR(100), VARCHAR(20), TIMESTAMP
Event	id, title, description, start_date, end_date, max_participants, status	BIGINT, VARCHAR(200), TEXT, TIMESTAMP,

Сущность	Атрибуты	Типы данных (SQL)
		TIMESTAMP, INT, VARCHAR(20)
Registration	id, user_id, event_id, registration_date, status	BIGINT, BIGINT, BIGINT, TIMESTAMP, VARCHAR(20)
Payment	id, registration_id, amount, payment_date, method, transaction_id	BIGINT, BIGINT, DECIMAL(10,2), TIMESTAMP, VARCHAR(50), VARCHAR(100)
Category	id, name, description	BIGINT, VARCHAR(100), TEXT

3.3. Шаг 2: Определение первичных ключей

Для каждой таблицы используется суррогатный первичный ключ id (автоинкремент / BIGSERIAL).

sql

id BIGSERIAL PRIMARY KEY

3.4. Шаг 3: Нормализация

Проверка 1НФ

Требование 1: Все атрибуты атомарны

Таблица	Проверка	Результат
User	email — строка, name — строка	+
Event	title — строка, description — текст	+
Registration	registration_date — дата, status — строка	+
Payment	amount — число, method — строка	+
Category	name — строка	+

Требование 2: Есть первичный ключ

У всех таблиц есть id — первичный ключ. +

Вывод: Все таблицы находятся в 1НФ.

Проверка 2НФ

Требование: Все неключевые атрибуты зависят от полного первичного ключа.

У всех таблиц простой первичный ключ (одно поле id), поэтому частичная зависимость невозможна.

Вывод: Все таблицы находятся в 2НФ.

Проверка 3НФ

Требование: Отсутствие транзитивных зависимостей.

Таблица	**Проверка**	**Результат**
User	Все поля зависят только от id	+

Таблица	Проверка	Результат
Event	Все поля зависят только от id; category_id — внешний ключ	+
Registration	Все поля зависят только от id; user_id, event_id — внешние ключи	+
Payment	Все поля зависят только от id; registration_id — внешний ключ	+
Category	Все поля зависят только от id	+

Вывод: Все таблицы находятся в 3НФ.

3.5. Шаг 4: Определение связей и внешних ключей

Связь	ип	Внешний ключ	Ограничения
User → Registration	:N	registration.user_id → user.id	ON DELETE CASCADE
Event → Registration	:N	registration.event_id → event.id	ON DELETE CASCADE
Regi		payment.registration_id → regist	UNIQUE,

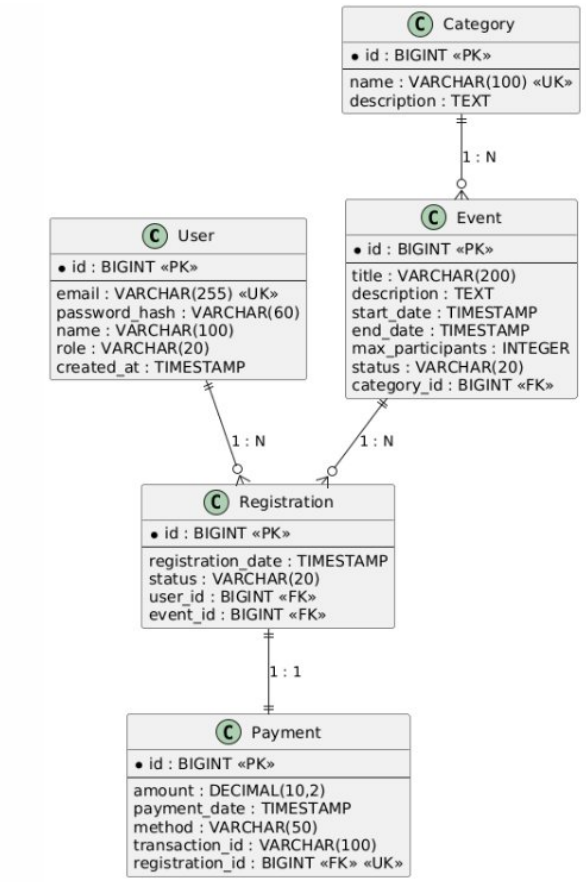
Связь	ип	Внешний ключ	Ограничения
stration → Payment	:1	ration.id	ON DELETE CASCADE
Category → Event	:N	event.category_id → category.id	ON DELETE SET NULL

3.6. ER-диаграмма (логическая модель)

```
@startuml
!define TABLE class

TABLE "User" as users {
  * id : BIGINT <<PK>>
  --
  email : VARCHAR(255) <<UK>>
  password_hash : VARCHAR(60)
  name : VARCHAR(100)
  role : VARCHAR(20)
  created_at : TIMESTAMP
}

TABLE "Event" as events {
  * id : BIGINT <<PK>>
  --
  title : VARCHAR(200)
```

```
  description : TEXT
  start_date : TIMESTAMP
  end_date : TIMESTAMP
  max_participants : INTEGER
  status : VARCHAR(20)
  category_id : BIGINT <<FK>>
}

TABLE "Category" as categories {
  * id : BIGINT <<PK>>
  --
  name : VARCHAR(100) <<UK>>
  description : TEXT
}

TABLE "Registration" as registrations {
  * id : BIGINT <<PK>>
  --
  registration_date : TIMESTAMP
  status : VARCHAR(20)
  user_id : BIGINT <<FK>>
  event_id : BIGINT <<FK>>
}

TABLE "Payment" as payments {
  * id : BIGINT <<PK>>
  --
  amount : DECIMAL(10,2)
  payment_date : TIMESTAMP
```

```
  method : VARCHAR(50)
  transaction_id : VARCHAR(100)
  registration_id : BIGINT <<FK>> <<UK>>
}

users ||--o{ registrations : "1 : N"
events ||--o{ registrations : "1 : N"
categories ||--o{ events : "1 : N"
registrations ||--|| payments : "1 : 1"

@enduml
```



3.7. Спецификация связей

Связь	Тип	Таблица	Поле	Связанная таблица	Действие при удалении
User → Registration	:N	registrations	user_id	users	CASCADE
Event → Registration	:N	registrations	event_id	events	CASCADE
Category → Event	:N	events	category_id	categories	SET NULL
Registration → Payment	:1	payments	registration_id	registrations	CASCADE

Пояснение выбора действий при удалении:

CASCADE для User и Registration: Если пользователь удаляется, его регистрации теряют смысл

CASCADE для Event и Registration: Если мероприятие удаляется, регистрации на него теряют смысл

SET NULL для Category и Event: Категория может быть удалена, но мероприятия должны остаться

CASCADE для Registration и Payment: Платёж не имеет смысла без регистрации

4. МЕТОДИКА ВЫПОЛНЕНИЯ (ПОШАГОВАЯ ИНСТРУКЦИЯ)

Шаг 1: Анализ Domain Model из ЛР2 (1 час)

1. Откройте Domain Model, созданную в лабораторной работе №2
2. Выпишите все классы-сущности

3. Для каждого класса выпишите атрибуты с типами (Java -> SQL маппинг)
4. Определите все ассоциации между классами
5. Таблица для заполнения:

Класс (Domain Model)	Таблица (БД)	Атрибуты (Java → SQL)

Шаг 2: Определение первичных ключей (30 мин)

1. Для каждой таблицы определить первичный ключ (обычно id)
2. Выбрать стратегию генерации (автоинкремент / sequence)
3. Отметить первичные ключи на диаграмме (звёздочка *)

Пример:

id BIGSERIAL PRIMARY KEY

Шаг 3: Нормализация

Проверка 1НФ

Все атрибуты атомарны? (нет массивов, списков в одной ячейке)

Есть первичный ключ?

Проверка 2НФ

Если составной ключ — все неключевые атрибуты зависят от полного ключа?

При нарушении — разделить на несколько таблиц

Проверка 3НФ

Отсутствуют транзитивные зависимости (атрибут зависит от другого неключевого)?

При нарушении — вынести зависимые атрибуты в отдельную таблицу

Чек-лист нормализации:

Таблица	1НФ	2НФ	3НФ	Примечания
...	☐	☐	☐	

Шаг 4: Определение связей и внешних ключей (1 час)

1. Для каждой связи определить тип (1:1, 1:N, M:N)
2. Определить, в какой таблице будет внешний ключ
3. Выбрать действие при удалении (CASCADE, SET NULL, RESTRICT)

Таблица для заполнения:

Связь	Тип	Таблица	Поле	Связанная таблица	Действие при удалении
...	...	...	...	...	...

Шаг 5: Построение ER-диаграммы (1 час)

1. Используйте PlantUML, Draw.io или Lucidchart
2. Укажите все сущности (прямоугольники)
3. Укажите все атрибуты с типами
4. Отметьте первичные ключи (* или <<PK>>)
5. Отметьте внешние ключи (<<FK>>)
6. Покажите связи с указанием кратностей

Шаг 6: Оформление отчёта (1 час)

1. Вставьте ER-диаграмму в отчёт
2. Добавьте таблицы с атрибутами
3. Добавьте таблицу связей
4. Напишите выводы о нормализации

5. ВАРИАНТЫ ДЛЯ ИНДИВИДУАЛЬНОГО ВЫПОЛНЕНИЯ

Варианты соответствуют проектам из лабораторной работы №1:

	Проект	Ключевые сущности (минимум 5)
	Система управления складом	Product, Warehouse, StockMovement, Supplier, Category
	Платформа аренды оборудования	Equipment, Rental, Customer, Maintenance, Category
	Система управления парком автомобилей	Vehicle, Driver, Trip, Maintenance, Fueling
	Система дистанционного обучения (LMS)	Course, Module, Lesson, Student, Enrollment, Progress
	Платформа для проведения онлайн-курсов	Course, Instructor, Student, Video, Quiz, Certificate
	Система записи на кружки и секции	Circle, Schedule, Student, Teacher, Attendance, Payment
	Система управления медицинскими назначениями	Patient, Doctor, Appointment, Prescription,

	Проект	Ключевые сущности (минимум 5)
		Diagnosis
	Платформа записи на диагностические процедуры	Patient, Procedure, Equipment, Schedule, Result
	Система мониторинга здоровья пациентов	Patient, Measurement, Alert, Device, Doctor
0	Система управления доставкой еды	Restaurant, Dish, Order, Courier, Customer, Payment
1	Платформа бронирования услуг красоты	Salon, Master, Service, Client, Booking, Review
2	Система проката спортивного инвентаря	Item, Rental, Customer, Category, Maintenance
3	Система управления производственными заказами	Order, Product, Operation, Worker, Equipment
4	Платформа логистики и отслеживания грузов	Shipment, Route, Vehicle, Driver, Tracking,

	Проект	Ключевые сущности (минимум 5)
		Customer
5	Система управления проектами в строительстве	Project, Task, Worker, Material, Equipment, Report

6. КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Чем отличается логическая модель данных от физической?
2. Что такое сущность, атрибут, связь в контексте ER-диаграммы?
3. Какие требования предъявляются к таблице, находящейся в 1НФ?
4. Какие требования предъявляются к таблице, находящейся в 2НФ?
5. Какие требования предъявляются к таблице, находящейся в 3НФ?
6. Как в реляционной БД реализуется связь «один-ко-многим»?
7. Как в реляционной БД реализуется связь «многие-ко-многим»?
8. Для чего нужны внешние ключи (FOREIGN KEY)?
9. Что такое первичный ключ и каким требованиям он должен удовлетворять?
10. В чём разница между CASCADE, SET NULL и RESTRICT при удалении
11. В каких случаях следует использовать суррогатный ключ вместо естественного?
12. Как обеспечить уникальность пары полей (user_id, event_id) в таблице registrations?
13. Для чего нужны ограничения CHECK? Приведите пример.
14. Какие аномалии возникают при нарушении нормальных форм?
15. Что такое транзитивная зависимость и как её устранить?

16. Как спроектировать БД для системы с мультиязычным контентом?
17. Как реализовать мягкое удаление (soft delete) на уровне БД?
18. Какие существуют паттерны для аудита изменений данных?
19. Как обеспечить версионирование записей для оптимистичной блокировки?
20. Как спроектировать БД для хранения иерархических структур (деревьев категорий)?

7. ТРЕБОВАНИЯ К ОТЧЕТУ

Структура отчета

1. Титульный лист
2. Содержание
3. Введение (цель работы, краткое описание проекта)
4. Исходная Domain Model (из ЛР2)
5. Проектирование базы данных
   5.1. Таблица сущностей и атрибутов
   5.2. ER-диаграмма (логическая модель)
   5.3. Обоснование нормализации (проверка 1НФ, 2НФ, 3НФ)
   5.4. Спецификация связей и внешних ключей
6. Заключение (выводы, план дальнейшей работы — DDL-скрипты)
7. Список использованных источников
8. Приложения (исходный код ER-диаграммы PlantUML)

Требования к оформлению

Элемент	Требование
Формат файла	PDF
Объём	8-12 страниц

Элемент	Требование
ER-диаграмма	Векторный формат (PNG/SVG), читаемые подписи
Имя файла	Фамилия_Имя_ЛР4_ПИ.pdf

8. КРИТЕРИИ ОЦЕНИВАНИЯ

Критерий	Макс. балл	Описание
Полнота анализа Domain Model	2	Все сущности и атрибуты корректно выделены
Качество ER-диаграммы	5	Корректность, читаемость, полнота (сущности, атрибуты, ключи, связи)
Нормализация до 3НФ	4	Отсутствие избыточности, правильные зависимости
Спецификация связей	3	Все внешние ключи, типы связей, действия при удалении
Обоснование решений	3	Пояснения к выбору типов данных, ключей, действий при удалении

Критерий	Макс. балл	Описание
Оформление отчета	2	Соответствие требованиям, структура
Ответы на контрольные вопросы	2	Понимание материала при защите
ИТОГО	21	

9. ИНСТРУМЕНТЫ ДЛЯ ВЫПОЛНЕНИЯ

Назначение	Инструмент	Ссылка
ER-диаграммы	Draw.io	https://app.diagrams.net
ER-диаграммы	PlantUML	https://plantuml.com
ER-диаграммы	Lucidchart	https://lucid.app
Проверка нормализации	Онлайн-калькуляторы	Ручная проверка

11. ЗАКЛЮЧЕНИЕ

Выполнение лабораторной работы №4 позволяет:

Освоить этапы проектирования реляционной базы данных

Научиться строить ER-диаграммы и выделять сущности

Проводить нормализацию до 3НФ и устранять избыточность

Подготовить основу для создания DDL-скриптов (ЛР5)

Закрепить связь между объектной моделью и реляционной БД

Лабораторная работа №5

Создание DDL-скриптов

1. Цель и задачи работы

Цель: Освоить методику создания физической модели базы данных на основе ER-диаграммы (из ЛР4) путём разработки DDL-скриптов (Data Definition Language) для создания схемы базы данных в целевой СУБД.

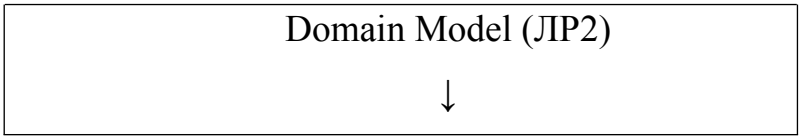
Задачи:

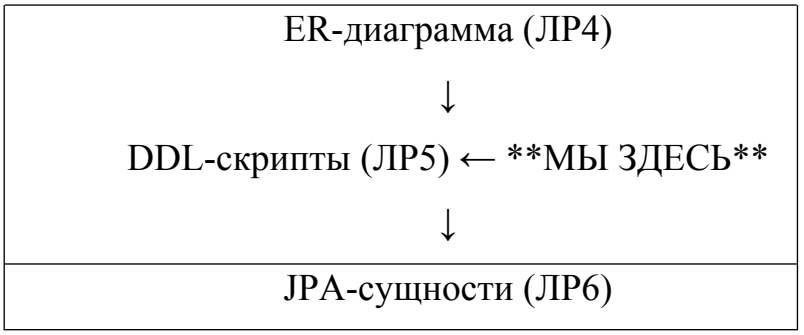
1. Преобразовать логическую модель данных (ER-диаграмму) в физическую (DDL-скрипты)
2. Написать SQL-скрипты CREATE TABLE для всех сущностей
3. Определить типы данных с учётом особенностей целевой СУБД (PostgreSQL)
4. Добавить ограничения целостности: PRIMARY KEY, FOREIGN KEY, UNIQUE, CHECK
5. Создать индексы для часто используемых полей
6. Обеспечить правильный порядок создания таблиц (с учётом внешних ключей)

2. ТЕОРЕТИЧЕСКОЕ ОБОСНОВАНИЕ

2.1. Место DDL-скриптов в жизненном цикле

DDL-скрипты - это этап перехода от логической модели данных (ER-диаграмма) к физической реализации в конкретной СУБД.





2.2. Что такое DDL?

DDL (Data Definition Language) - это подмножество SQL для определения структуры базы данных.

Основные команды DDL:

Команда	**Назначение**	**Пример**
CREATE TABLE	Создание таблицы	CREATE TABLE users (...)
ALTER TABLE	Изменение структуры таблицы	ALTER TABLE users ADD COLUMN phone VARCHAR(20)
DROP TABLE	Удаление таблицы	DROP TABLE users
CREATE INDEX	Создание индекса	CREATE INDEX idx_user_email ON users(email)
CREATE SEQUENCE	Создание последовательности	CREATE SEQUENCE user_id_seq

2.3. Типы данных в PostgreSQL

SQL тип	Java тип	Описание	Размер	Пример
BIGSERIAL	Long	Автоинкрементируемый BIGINT	8 байт	id BIGSERIAL PRIMARY KEY
BIGINT	Long	Целое число	8 байт	user_id BIGINT
INTEGER	Integer	Целое число	4 байта	age INTEGER
VARCHAR(n)	String	Строка фиксированной макс. длины	до n	email VARCHAR(255)
TEXT	String	Строка неограниченной длины	переменный	description TEXT
TIMESTAMP	LocalDateTime	Дата и время	8 байт	created_at TIMESTAMP
DATE	LocalDate	Только дата	4 байта	birth_date DATE
DECIMAL(p,s)	BigDecimal	Точное число	переменный	amount DECIMAL(10,2)
BOOLEAN	Boolean	true/false	1 байт	is_active BOOLEAN

2.4. Ограничения целостности (Constraints)

Ограничение	Синтаксис	Назначение
PRIMARY KEY	id BIGSERIAL PRIMARY KEY	Уникальный идентификатор строки

Ограничение	Синтаксис	Назначение
FOREIGN KEY	user_id BIGINT REFERENCES users(id)	Ссылка на другую таблицу
UNIQUE	email VARCHAR(255) UNIQUE	Запрет дублирования значений
NOT NULL	name VARCHAR(100) NOT NULL	Запрет NULL-значений
CHECK	CHECK (amount > 0)	Проверка условия
DEFAULT	status VARCHAR(20) DEFAULT 'ACTIVE'	Значение по умолчанию

2.5. Индексы

Индекс - это структура данных, ускоряющая поиск и сортировку.

Когда создавать индексы:

1. Поля, часто используемые в WHERE (например, email)
2. Поля, используемые в JOIN (внешние ключи)
3. Поля, используемые в ORDER BY
4. Поля, используемые в GROUP BY

Синтаксис создания индекса:

sql

```
-- Обычный индекс
CREATE INDEX idx_users_email ON users(email);

-- Составной индекс
CREATE INDEX idx_events_dates ON events(start_date, end_date);
```

```
-- Уникальный индекс
CREATE UNIQUE INDEX idx_users_email_unique ON users(email);
```

2.6. Порядок создания таблиц

При наличии внешних ключей таблицы должны создаваться в определённом порядке:

1. Таблицы без внешних ключей (независимые)
2. Таблицы, ссылающиеся на уже созданные

Пример порядка:

```
-- 1. Независимые таблицы (без внешних ключей)
CREATE TABLE users (...);      -- нет внешних ключей
CREATE TABLE categories (...); -- нет внешних ключей

-- 2. Таблицы, зависящие от users и categories
CREATE TABLE events (
    ...
    created_by BIGINT REFERENCES users(id),
    category_id BIGINT REFERENCES categories(id)
);

-- 3. Таблицы, зависящие от events
CREATE TABLE registrations (
    ...
    event_id BIGINT REFERENCES events(id)
);
```

```
-- 4. Самые зависимые таблицы
CREATE TABLE payments (
    ...
    registration_id BIGINT REFERENCES registrations(id)
);
```

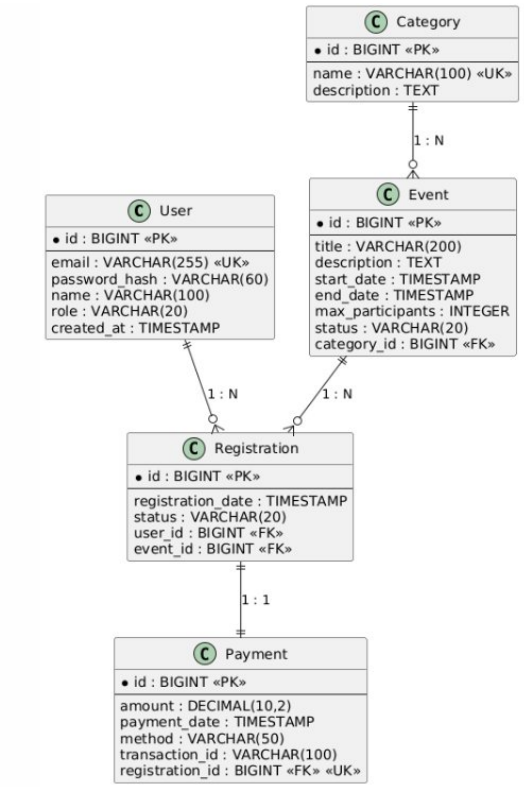
2.7. Действия при удалении (ON DELETE)

Действие	Описание	Синтаксис
CASCADE	Удаление родителя удаляет детей	ON DELETE CASCADE
SET NULL	Внешний ключ становится NULL	ON DELETE SET NULL
RESTRICT	Запрещает удаление, если есть дети	ON DELETE RESTRICT
NO ACTION	Аналогично RESTRICT (в PostgreSQL)	ON DELETE NO ACTION

3. ПРИМЕР ВЫПОЛНЕНИЯ (СКВОЗНОЙ ПРОЕКТ)

Проект: «Система управления мероприятиями» (Event Management System)

3.1. Исходная ER-диаграмма (из ЛР4)



3.2. Шаг 1: Создание независимых таблиц

Таблица users:

```
-- Создание таблицы пользователей
CREATE TABLE users (
    id BIGSERIAL PRIMARY KEY,
    email VARCHAR(255) NOT NULL UNIQUE,
    password_hash VARCHAR(60) NOT NULL,
    name VARCHAR(100) NOT NULL,
    role VARCHAR(20) NOT NULL DEFAULT 'USER',
    created_at TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP
);
```

```
-- Комментарии к столбцам
COMMENT ON TABLE users IS 'Пользователи системы';
COMMENT ON COLUMN users.email IS 'Email пользователя (уникальный)';
COMMENT ON COLUMN users.password_hash IS 'Хеш пароля (BCrypt)';
COMMENT ON COLUMN users.role IS 'Роль: USER, ADMIN';
```

Таблица categories:

```
-- Создание таблицы категорий
CREATE TABLE categories (
    id BIGSERIAL PRIMARY KEY,
    name VARCHAR(100) NOT NULL UNIQUE,
    description TEXT,
    created_at TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP
);

COMMENT ON TABLE categories IS 'Категории мероприятий';
```

3.3. Шаг 2: Создание таблиц с внешними ключами

Таблица events:

```
-- Создание таблицы мероприятий
CREATE TABLE events (
    id BIGSERIAL PRIMARY KEY,
    title VARCHAR(200) NOT NULL,
    description TEXT,
```

```
    start_date TIMESTAMP NOT NULL,
    end_date TIMESTAMP NOT NULL,
    max_participants INTEGER NOT NULL CHECK (max_participants > 0),
    status VARCHAR(20) NOT NULL DEFAULT 'DRAFT',
    category_id BIGINT,
    created_by BIGINT NOT NULL,
    created_at TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,

    -- Внешние ключи
    CONSTRAINT fk_event_category FOREIGN KEY (category_id)
      REFERENCES categories(id) ON DELETE SET NULL,
    CONSTRAINT fk_event_creator FOREIGN KEY (created_by)
      REFERENCES users(id) ON DELETE CASCADE,

    -- Проверочные ограничения
    CONSTRAINT chk_event_dates CHECK (start_date < end_date),
    CONSTRAINT chk_event_status CHECK (status IN ('DRAFT', 'PUBLISHED', 'CANCELLED', 'COMPLETED'))
);

COMMENT ON TABLE events IS 'Мероприятия';
COMMENT ON COLUMN events.status IS 'Статус мероприятия';
```

Таблица registrations:

```
-- Создание таблицы регистраций
```

```
CREATE TABLE registrations (
    id BIGSERIAL PRIMARY KEY,
    user_id BIGINT NOT NULL,
    event_id BIGINT NOT NULL,
    registration_date TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
    status VARCHAR(20) NOT NULL DEFAULT 'PENDING',
    created_at TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,

    -- Внешние ключи
    CONSTRAINT fk_reg_user FOREIGN KEY (user_id)
      REFERENCES users(id) ON DELETE CASCADE,
    CONSTRAINT fk_reg_event FOREIGN KEY (event_id)
      REFERENCES events(id) ON DELETE CASCADE,

    -- Проверочные ограничения
    CONSTRAINT chk_reg_status CHECK (status IN ('PENDING', 'CONFIRMED', 'CANCELLED')),

    -- Уникальность пары (пользователь не может зарегистрироваться дважды)
    CONSTRAINT uk_reg_user_event UNIQUE (user_id, event_id)
);

COMMENT ON TABLE registrations IS 'Регистрации пользователей на мероприятия';
```

3.4. Шаг 3: Создание таблицы с уникальным внешним ключом (1:1)

Таблица payments:

```
-- Создание таблицы платежей
CREATE TABLE payments (
    id BIGSERIAL PRIMARY KEY,
    registration_id BIGINT NOT NULL UNIQUE, -- UNIQUE обеспечивает связь 1:1
    amount DECIMAL(10,2) NOT NULL CHECK (amount >= 0),
    payment_date TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
    method VARCHAR(50) NOT NULL,
    transaction_id VARCHAR(100),
    created_at TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,

    -- Внешний ключ
    CONSTRAINT fk_payment_registration FOREIGN KEY (registration_id)
        REFERENCES registrations(id) ON DELETE CASCADE,

    -- Проверочные ограничения
    CONSTRAINT chk_payment_method CHECK (method IN ('CARD', 'CASH', 'ONLINE'))
);

COMMENT ON TABLE payments IS 'Платежи за регистрации';
```

3.5. Шаг 4: Создание индексов

```
-- Индексы для users
CREATE INDEX idx_users_email ON users(email);
CREATE INDEX idx_users_role ON users(role);

-- Индексы для events
CREATE INDEX idx_events_dates ON events(start_date, end_date);
CREATE INDEX idx_events_status ON events(status);
CREATE INDEX idx_events_category ON events(category_id);

-- Индексы для registrations
CREATE INDEX idx_registrations_user ON registrations(user_id);
CREATE INDEX idx_registrations_event ON registrations(event_id);
CREATE INDEX idx_registrations_status ON registrations(status);
CREATE INDEX idx_registrations_date ON registrations(registration_date);

-- Индексы для payments
CREATE INDEX idx_payments_registration ON payments(registration_id);
CREATE INDEX idx_payments_date ON payments(payment_date);
```

3.6. Полный DDL-скрипт (в правильном порядке)

```
-- ============================================================
-- DDL-скрипт для системы управления мероприятиями
-- СУБД: PostgreSQL 15+
-- ============================================================

-- 1. Независимые таблицы
```

```
-- ============================================================

-- Таблица пользователей
CREATE TABLE users (
    id BIGSERIAL PRIMARY KEY,
    email VARCHAR(255) NOT NULL UNIQUE,
    password_hash VARCHAR(60) NOT NULL,
    name VARCHAR(100) NOT NULL,
    role VARCHAR(20) NOT NULL DEFAULT 'USER',
    created_at TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP
);

-- Таблица категорий
CREATE TABLE categories (
    id BIGSERIAL PRIMARY KEY,
    name VARCHAR(100) NOT NULL UNIQUE,
    description TEXT,
    created_at TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP
);

-- 2. Таблицы, зависящие от users и categories
-- ============================================================

-- Таблица мероприятий
CREATE TABLE events (
```

```
    id BIGSERIAL PRIMARY KEY,
    title VARCHAR(200) NOT NULL,
    description TEXT,
    start_date TIMESTAMP NOT NULL,
    end_date TIMESTAMP NOT NULL,
    max_participants INTEGER NOT NULL CHECK (max_participants > 0),
    status VARCHAR(20) NOT NULL DEFAULT 'DRAFT',
    category_id BIGINT,
    created_by BIGINT NOT NULL,
    created_at TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,

    CONSTRAINT fk_event_category FOREIGN KEY (category_id)
      REFERENCES categories(id) ON DELETE SET NULL,
    CONSTRAINT fk_event_creator FOREIGN KEY (created_by)
      REFERENCES users(id) ON DELETE CASCADE,
    CONSTRAINT chk_event_dates CHECK (start_date < end_date),
    CONSTRAINT chk_event_status CHECK (status IN ('DRAFT', 'PUBLISHED', 'CANCELLED', 'COMPLETED'))
);

-- 3. Таблицы, зависящие от events
-- ================================================

-- Таблица регистраций
CREATE TABLE registrations (
    id BIGSERIAL PRIMARY KEY,
```

```
    user_id BIGINT NOT NULL,
    event_id BIGINT NOT NULL,
    registration_date TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
    status VARCHAR(20) NOT NULL DEFAULT 'PENDING',
    created_at TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,

    CONSTRAINT fk_reg_user FOREIGN KEY (user_id)
      REFERENCES users(id) ON DELETE CASCADE,
    CONSTRAINT fk_reg_event FOREIGN KEY (event_id)
      REFERENCES events(id) ON DELETE CASCADE,
    CONSTRAINT chk_reg_status CHECK (status IN ('PENDING', 'CONFIRMED', 'CANCELLED')),
    CONSTRAINT uk_reg_user_event UNIQUE (user_id, event_id)
);

-- 4. Таблицы, зависящие от registrations
-- ============================================================

-- Таблица платежей
CREATE TABLE payments (
    id BIGSERIAL PRIMARY KEY,
    registration_id BIGINT NOT NULL UNIQUE,
    amount DECIMAL(10,2) NOT NULL CHECK (amount >= 0),
    payment_date TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
    method VARCHAR(50) NOT NULL,
    transaction_id VARCHAR(100),
```

```
    created_at TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,

    CONSTRAINT fk_payment_registration FOREIGN KEY (registration_id)
        REFERENCES registrations(id) ON DELETE CASCADE,
    CONSTRAINT chk_payment_method CHECK (method IN ('CARD', 'CASH', 'ONLINE'))
);

-- 5. Индексы
-- ============================================================

-- Индексы для users
CREATE INDEX idx_users_email ON users(email);
CREATE INDEX idx_users_role ON users(role);

-- Индексы для events
CREATE INDEX idx_events_dates ON events(start_date, end_date);
CREATE INDEX idx_events_status ON events(status);
CREATE INDEX idx_events_category ON events(category_id);

-- Индексы для registrations
CREATE INDEX idx_registrations_user ON registrations(user_id);
CREATE INDEX idx_registrations_event ON registrations(event_id);
CREATE INDEX idx_registrations_status ON registrations(status);
CREATE INDEX idx_registrations_date ON registrations(registration_date);

-- Индексы для payments
```

```
CREATE INDEX idx_payments_registration ON payments(registration_id);
CREATE INDEX idx_payments_date ON payments(payment_date);

-- 6. Комментарии (документация)
-- ============================================================

COMMENT ON TABLE users IS 'Пользователи системы';
COMMENT ON TABLE categories IS 'Категории мероприятий';
COMMENT ON TABLE events IS 'Мероприятия';
COMMENT ON TABLE registrations IS 'Регистрации пользователей на мероприятия';
COMMENT ON TABLE payments IS 'Платежи за регистрации';
```

4. МЕТОДИКА ВЫПОЛНЕНИЯ (ПОШАГОВАЯ ИНСТРУКЦИЯ)

Шаг 1: Анализ ER-диаграммы из ЛР4

Откройте ER-диаграмму, созданную в ЛР4

Выпишите список всех сущностей

Определите порядок создания таблиц (сначала независимые, потом зависимые)

Таблица порядка создания:

Порядок	Таблица	Зависимости (FK)
1	...	Нет
2	...	Нет
3	...	Зависит от таблиц 1-2
...	...	...

Шаг 2: Определение типов данных (1 час)

Для каждого атрибута определите SQL-тип (см. таблицу типов выше)

Учитывайте особенности PostgreSQL

Для строк — предпочтительно VARCHAR с ограничением длины

Таблица типов данных:

Таблица	**Атрибут**	**Тип в Java**	**Тип в SQL**	**Ограничения**
users	id	Long	BIGSERIAL	PRIMARY KEY
users	email	String	VARCHAR(255)	NOT NULL, UNIQUE
...	...	...	...	...

Шаг 3: Написание CREATE TABLE

Для каждой таблицы написать CREATE TABLE

Добавить первичные ключи

Добавить внешние ключи (REFERENCES)

Добавить NOT NULL где необходимо

Добавить DEFAULT для значений по умолчанию

Добавить CHECK для проверок

Добавить UNIQUE для уникальных полей

Шаблон CREATE TABLE:

```
CREATE TABLE table_name (
    id BIGSERIAL PRIMARY KEY,
    column1 TYPE NOT NULL,
    column2 TYPE,
```

```
    column3 TYPE DEFAULT 'default_value',
    column4 TYPE CHECK (condition),

    CONSTRAINT fk_name FOREIGN KEY (column4) REFERENCES other_table(id) ON DELETE ACTION,
    CONSTRAINT uk_name UNIQUE (column1, column2),
    CONSTRAINT chk_name CHECK (condition)
);
```

Шаг 4: Создание индексов

Добавить индекс для полей, часто используемых в WHERE

Добавить индекс для внешних ключей

Добавить составные индексы для часто используемых комбинаций

Таблица индексов:

Таблица	Поле(я)	Тип индекса	Обоснование
users	email	обычный	Поиск по email при входе
events	start_date, end_date	составной	Фильтрация по датам
...	...	...	...

Шаг 5: Проверка синтаксиса

Проверьте, что все скобки закрыты

Проверьте, что все запятые на месте

Проверьте порядок создания таблиц (внешние ключи ссылаются на существующие)

Проверьте, что все типы данных корректны

Шаг 6: Оформление отчёта

Вставьте полный DDL-скрипт в отчёт

Добавьте комментарии к сложным местам

Опишите обоснование выбора типов данных

Добавьте таблицу индексов с пояснениями

5. ВАРИАНТЫ ДЛЯ ИНДИВИДУАЛЬНОГО ВЫПОЛНЕНИЯ

Варианты соответствуют проектам из предыдущих лабораторных работ:

	Проект	Минимальное количество таблиц
	Система управления складом	5 (Products, Warehouse, StockMovement, Supplier, Category)
	Платформа аренды оборудования	5 (Equipment, Rental, Customer, Maintenance, Category)
	Система управления парком автомобилей	5 (Vehicle, Driver, Trip, Maintenance, Fueling)
	Система дистанционного обучения (LMS)	6 (Course, Module, Lesson, Student, Enrollment, Progress)
	Платформа для проведения онлайн-курсов	6 (Course, Instructor, Student, Video, Quiz, Certificate)

	Проект	Минимальное количество таблиц
	Система записи на кружки и секции	6 (Circle, Schedule, Student, Teacher, Attendance, Payment)
	Система управления медицинскими назначениями	5 (Patient, Doctor, Appointment, Prescription, Diagnosis)
	Платформа записи на диагностические процедуры	5 (Patient, Procedure, Equipment, Schedule, Result)
	Система мониторинга здоровья пациентов	5 (Patient, Measurement, Alert, Device, Doctor)
0	Система управления доставкой еды	6 (Restaurant, Dish, Order, Courier, Customer, Payment)
1	Платформа бронирования услуг красоты	6 (Salon, Master, Service, Client, Booking, Review)
2	Система проката спортивного инвентаря	5 (Item, Rental, Customer, Category, Maintenance)
	Система	5 (Order, Product, Operation, Worker,

	Проект	Минимальное количество таблиц
3	управления производственными заказами	Equipment)
4	Платформа логистики и отслеживания грузов	6 (Shipment, Route, Vehicle, Driver, Tracking, Customer)
5	Система управления проектами в строительстве	6 (Project, Task, Worker, Material, Equipment, Report)

6. КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Что такое DDL и какие команды к нему относятся?
2. В чём разница между VARCHAR(255) и TEXT?
3. Какой тип данных в PostgreSQL соответствует Java-типу LocalDateTime?
4. Что такое BIGSERIAL и для чего он используется?
5. Какой синтаксис создания внешнего ключа?
6. Чем отличается ON DELETE CASCADE от ON DELETE SET NULL?
7. Как создать уникальный индекс на поле?
8. Что такое составной индекс и когда он нужен?
9. Почему важен порядок создания таблиц при наличии внешних ключей?
10. В каких случаях стоит использовать CHECK-ограничения?

11. Какой тип данных выбрать для хранения денежных сумм и почему?
12. Как создать индекс для ускорения поиска по диапазону дат?
13. Что произойдёт, если попытаться создать таблицу с внешним ключом на ещё не созданную таблицу?
14. Как обеспечить версионирование записей (оптимистичную блокировку) на уровне DDL?
15. Как реализовать мягкое удаление (soft delete) с использованием DDL?
16. Какие существуют стратегии партиционирования (шардирования) больших таблиц?
17. Как в PostgreSQL создать индекс по выражению (например, LOWER(email))?
18. Что такое EXCLUDE-ограничения и когда они применяются?

7. ТРЕБОВАНИЯ К ОТЧЕТУ

Структура отчета

1. Титульный лист
2. Содержание
3. Введение (цель работы, краткое описание проекта)
4. Исходная ER-диаграмма (из ЛР4)
5. DDL-скрипты
   5.1. Порядок создания таблиц (обоснование)
   5.2. CREATE TABLE для каждой таблицы
   5.3. Индексы (с пояснениями)
6. Обоснование решений
   6.1. Выбор типов данных
   6.2. Выбор действий ON DELETE
   6.3. Выбор полей для индексации

7. Заключение (выводы, план дальнейшей работы — JPA)

8. Список использованных источников

9. Приложения (полный DDL-скрипт)

8. КРИТЕРИИ ОЦЕНИВАНИЯ

Критерий	**Макс. балл**	**Описание**
Полнота DDL-скриптов	4	Все таблицы созданы, все поля определены
Корректность типов данных	3	Правильный выбор SQL-типов под Java-типы
Ограничения целостности	4	PRIMARY KEY, FOREIGN KEY, UNIQUE, CHECK, NOT NULL, DEFAULT
Индексы	3	Созданы для часто используемых полей, обоснованы
Порядок создания таблиц	2	Правильный порядок (нет ошибок FK)
Обоснование решений	3	Пояснения к типам, индексам, ON DELETE
Оформление отчета	2	Соответствие требованиям
Ответы на контрольные вопросы	2	Понимание материала при защите

Критерий	Макс. балл	Описание
ИТОГО	23	

9. ИНСТРУМЕНТЫ ДЛЯ ВЫПОЛНЕНИЯ

Назначение	Инструмент	Ссылка
Редактор SQL	DBeaver	https://dbeaver.io
Редактор SQL	DataGrip	https://www.jetbrains.com/datagrip/
Редактор SQL	pgAdmin	https://www.pgadmin.org
Проверка синтаксиса	PostgreSQL	https://www.postgresql.org
ER-диаграмма	Draw.io / PlantUML	из ЛР4

Лабораторная работа №6

Реализация Entity-слоя (JPA-сущности)

1. Цель и задачи работы

Цель: Освоить методику создания JPA-сущностей (Entity-слоя) на основе спроектированной базы данных (DDL-скрипты из ЛР5) с использованием объектно-реляционного отображения (ORM) в контексте архитектуры PCMEF.

Задачи:

1. Настроить Maven-проект с зависимостями для JPA/Hibernate
2. Настроить подключение к базе данных PostgreSQL
3. Создать JPA-сущности для всех таблиц из ЛР5
4. Реализовать связи между сущностями (@OneToMany, @ManyToOne, @OneToOne)
5. Настроить стратегии генерации первичных ключей
6. Создать базовые бизнес-методы в сущностях

2. ТЕОРЕТИЧЕСКОЕ ОБОСНОВАНИЕ

2.1. Место Entity-слоя в архитектуре PCMEF

В архитектуре PCMEF Entity-слой отвечает за представление бизнес-сущностей и их состояния. Он является связующим звеном между реляционной базой данных (Foundation-слой) и бизнес-логикой (Mediator-слой).

Database (PostgreSQL) ↓
Foundation (репозитории) ↓
Entity (JPA-сущности) ← **МЫ ЗДЕСЬ** ↓
Mediator (бизнес-логика) ↓
Control (REST API)

↓
Presentation (UI)

2.2. Что такое JPA и Hibernate?

JPA (Java Persistence API) - это спецификация Java для объектно-реляционного отображения (ORM).

Hibernate - самая популярная реализация спецификации JPA.

Основные аннотации JPA:

Аннотация	**Назначение**	**Пример**
@Entity	Объявляет класс сущностью	@Entity public class User { ... }
@Table	Задаёт имя таблицы в БД	@Table(name = "users")
@Id	Объявляет первичный ключ	@Id private Long id;
@GeneratedValue	Стратегия генерации ID	@GeneratedValue(strategy = GenerationType.IDENTITY)
@Column	Настройка отображения поля	@Column(unique = true, nullable = false)
@Transient	Поле не сохраняется в БД	@Transient private String temp;

2.3. Стратегии генерации первичных ключей

Стратегия	Описание	Синтаксис	Когда использовать
IDENTITY	Автоинкремент (AUTO_INCREMENT)	@GeneratedValue(strategy = GenerationType.IDENTITY)	PostgreSQL (BIGSERIAL), MySQL
SEQUENCE	Последовательность	@GeneratedValue(strategy = GenerationType.SEQUENCE)	Oracle, PostgreSQL
TABLE	Отдельная таблица для ID	@GeneratedValue(strategy = GenerationType.TABLE)	Кроссплатформенно
UUID	Универсальный уникальный ID	@GeneratedValue(strategy = GenerationType.UUID)	Распределённые системы

2.4. Отображение связей (Relationships)

Тип связи	**JPA-аннотация**	**Где находится**	**Пример**
Один-к-одному	@OneToOne	Оба класса	User ↔ Passport

Тип связи	JPA-аннотация	Где находится	Пример
Один-ко-многим	@OneToMany	Сторона "один"	User → List<Order>
Многие-к-одному	@ManyToOne	Сторона "многие"	Order → User
Многие-ко-многим	@ManyToMany	Оба класса	Student ↔ Course

Пример двунаправленной связи 1:N:

```
// Сторона "один" (Event)
@Entity
public class Event {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @OneToMany(mappedBy = "event", cascade = CascadeType.ALL)
    private List<Registration> registrations = new ArrayList<>();
}

// Сторона "многие" (Registration)
@Entity
public class Registration {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
```

```
    private Long id;

    @ManyToOne
    @JoinColumn(name = "event_id", nullable = false)
    private Event event
}
```

2.5. Типы загрузки (Fetch Types)

Тип	Описание	Когда использовать	Производительность
EAGER	Жадная загрузка (все данные сразу)	Маленькие объёмы, всегда нужны	Ниже
LAZY	Ленивая загрузка (при первом обращении)	Большие объёмы, не всегда нужны	Выше

Рекомендация: По умолчанию использовать LAZY и загружать данные явно через JOIN FETCH.

2.6. Каскадные операции (Cascade Types)

Тип	**Описание**	**Пример**
PERSIST	Сохранять связанные объекты	cascade = CascadeType.PERSIST

Тип	Описание	Пример
MERGE	Обновлять связанные объекты	cascade = CascadeType.MERGE
REMOVE	Удалять связанные объекты	cascade = CascadeType.REMOVE
REFRESH	Обновлять состояние	cascade = CascadeType.REFRESH
DETACH	Отсоединять от контекста	cascade = CascadeType.DETACH
ALL	Все операции	cascade = CascadeType.ALL

2.7. Бизнес-методы в сущностях

Сущности могут содержать бизнес-методы, инкапсулирующие логику, связанную с состоянием объекта.

```
@Entity
public class Event {
    // поля...

    public boolean checkAvailability() {
        return registrations.size() < maxParticipants;
    }

    public Registration registerUser(User user) {
```

```
        if (!checkAvailability()) {
            throw new IllegalStateException("No seats available");
        }
        Registration registration = new Registration();
        registration.setEvent(this);
        registration.setUser(user);
        registration.setRegistrationDate(LocalDateTime.now());
        this.registrations.add(registration);
        return registration;
    }
}
```

3. ПРИМЕР ВЫПОЛНЕНИЯ (СКВОЗНОЙ ПРОЕКТ)

Проект: «Система управления мероприятиями» (Event Management System)

3.1. Шаг 1: Настройка Maven-проекта

pom.xml:

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
         http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>ru.edu</groupId>
    <artifactId>event-management</artifactId>
    <version>1.0-SNAPSHOT</version>
```

```
<properties>
    <maven.compiler.source>17</maven.compiler.source>
    <maven.compiler.target>17</maven.compiler.target>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
</properties>

<dependencies>
    <!-- JPA API -->
    <dependency>
        <groupId>jakarta.persistence</groupId>
        <artifactId>jakarta.persistence-api</artifactId>
        <version>3.1.0</version>
    </dependency>

    <!-- Hibernate (реализация JPA) -->
    <dependency>
        <groupId>org.hibernate.orm</groupId>
        <artifactId>hibernate-core</artifactId>
        <version>6.4.0.Final</version>
    </dependency>

    <!-- PostgreSQL Driver -->
    <dependency>
        <groupId>org.postgresql</groupId>
        <artifactId>postgresql</artifactId>
        <version>42.7.1</version>
    </dependency>

    <!-- HikariCP (пул соединений) -->
```

```
        <dependency>
            <groupId>com.zaxxer</groupId>
            <artifactId>HikariCP</artifactId>
            <version>5.0.1</version>
        </dependency>

        <!-- Lombok (сокращение кода) -->
        <dependency>
            <groupId>org.projectlombok</groupId>
            <artifactId>lombok</artifactId>
            <version>1.18.30</version>
            <scope>provided</scope>
        </dependency>

        <!-- JUnit (для тестов) -->
        <dependency>
            <groupId>org.junit.jupiter</groupId>
            <artifactId>junit-jupiter</artifactId>
            <version>5.10.0</version>
            <scope>test</scope>
        </dependency>
    </dependencies>

</project>
```

3.2. Шаг 2: Настройка подключения к БД

persistence.xml (в src/main/resources/META-INF/):

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<persistence xmlns="https://jakarta.ee/xml/ns/persistence"
             xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
             xsi:schemaLocation="https://jakarta.ee/xml/ns/persistence
             https://jakarta.ee/xml/ns/persistence/persistence_3_0.xsd"
             version="3.0">

    <persistence-unit name="event-management-pu">
        <provider>org.hibernate.jpa.HibernatePersistenceProvider</provider>

        <properties>
            <!-- Подключение к БД -->
            <property name="jakarta.persistence.jdbc.driver"
value="org.postgresql.Driver"/>
            <property name="jakarta.persistence.jdbc.url"
value="jdbc:postgresql://localhost:5432/event_db"/>
            <property name="jakarta.persistence.jdbc.user" value="postgres"/>
            <property name="jakarta.persistence.jdbc.password"
value="password"/>

            <!-- Hibernate свойства -->
            <property name="hibernate.dialect"
value="org.hibernate.dialect.PostgreSQLDialect"/>
            <property name="hibernate.show_sql" value="true"/>
            <property name="hibernate.format_sql" value="true"/>
            <property name="hibernate.hbm2ddl.auto" value="validate"/>
        </properties>
    </persistence-unit>
</persistence>
```

EntityManagerFactory (фабрика):

java

```
package ru.edu.project.foundation.config;

import jakarta.persistence.EntityManager;
import jakarta.persistence.EntityManagerFactory;
import jakarta.persistence.Persistence;

public class JpaUtil {

    private static final EntityManagerFactory ENTITY_MANAGER_FACTORY;

    static {
        try {
            ENTITY_MANAGER_FACTORY = Persistence.createEntityManagerFactory("event-management-pu");
        } catch (Throwable ex) {
            throw new ExceptionInInitializerError(ex);
        }
    }

    public static EntityManager getEntityManager() {
        return ENTITY_MANAGER_FACTORY.createEntityManager();
    }

    public static void close() {
        if (ENTITY_MANAGER_FACTORY != null && ENTITY_MANAGER_FACTORY.isOpen()) {
            ENTITY_MANAGER_FACTORY.close();
```

```
        }
    }
}
```

3.3. Шаг 3: Создание JPA-сущностей

User.java:

```
package ru.edu.project.entity;

import jakarta.persistence.*;
import lombok.*;
import java.time.LocalDateTime;
import java.util.ArrayList;
import java.util.List;

@Entity
@Table(name = "users")
@Getter
@Setter
@NoArgsConstructor
@AllArgsConstructor
@Builder
public class User {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(unique = true, nullable = false, length = 255)
    private String email;
```

```
    @Column(name = "password_hash", nullable = false, length = 60)
    private String passwordHash;

    @Column(nullable = false, length = 100)
    private String name;

    @Column(nullable = false, length = 20)
    @Builder.Default
    private String role = "USER";

    @Column(name = "created_at", nullable = false)
    @Builder.Default
    private LocalDateTime createdAt = LocalDateTime.now();

    @Column(name = "updated_at", nullable = false)
    @Builder.Default
    private LocalDateTime updatedAt = LocalDateTime.now();

    @OneToMany(mappedBy = "user", cascade = CascadeType.ALL, fetch =
FetchType.LAZY)
    @Builder.Default
    private List<Registration> registrations = new ArrayList<>();

    // Бизнес-методы
    public boolean checkPassword(String rawPassword) {
        // В реальном проекте используется BCrypt
        return this.passwordHash.equals(rawPassword);
    }
```

```
    public void addRegistration(Registration registration) {
        registrations.add(registration);
        registration.setUser(this);
    }
}
```

Category.java:

```
package ru.edu.project.entity;

import jakarta.persistence.*;
import lombok.*;
import java.time.LocalDateTime;
import java.util.ArrayList;
import java.util.List;

@Entity
@Table(name = "categories")
@Getter
@Setter
@NoArgsConstructor
@AllArgsConstructor
@Builder
public class Category {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
```

```
    @Column(unique = true, nullable = false, length = 100)
    private String name;

    @Column(columnDefinition = "TEXT")
    private String description;

    @Column(name = "created_at", nullable = false)
    @Builder.Default
    private LocalDateTime createdAt = LocalDateTime.now();

    @OneToMany(mappedBy = "category", cascade = CascadeType.ALL,
fetch = FetchType.LAZY)
    @Builder.Default
    private List<Event> events = new ArrayList<>();
}
```

Event.java:

```
package ru.edu.project.entity;

import jakarta.persistence.*;
import lombok.*;
import java.math.BigDecimal;
import java.time.LocalDateTime;
import java.util.ArrayList;
import java.util.List;

@Entity
@Table(name = "events")
@Getter
```

```java
@Setter
@NoArgsConstructor
@AllArgsConstructor
@Builder
public class Event {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(nullable = false, length = 200)
    private String title;

    @Column(columnDefinition = "TEXT")
    private String description;

    @Column(name = "start_date", nullable = false)
    private LocalDateTime startDate;

    @Column(name = "end_date", nullable = false)
    private LocalDateTime endDate;

    @Column(name = "max_participants", nullable = false)
    private Integer maxParticipants;

    @Column(nullable = false, length = 20)
    @Builder.Default
    private String status = "DRAFT";
```

```
    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "category_id")
    private Category category;

    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "created_by", nullable = false)
    private User createdBy;

    @Column(name = "created_at", nullable = false)
    @Builder.Default
    private LocalDateTime createdAt = LocalDateTime.now();

    @Column(name = "updated_at", nullable = false)
    @Builder.Default
    private LocalDateTime updatedAt = LocalDateTime.now();

    @OneToMany(mappedBy = "event", cascade = CascadeType.ALL, fetch
= FetchType.LAZY)
    @Builder.Default
    private List<Registration> registrations = new ArrayList<>();

    // Бизнес-методы
    public boolean checkAvailability() {
        return registrations.size() < maxParticipants;
    }

    public int getAvailableSeats() {
        return maxParticipants - registrations.size();
    }
```

```
    public Registration registerUser(User user) {
        if (!checkAvailability()) {
            throw new IllegalStateException("No seats available for event: " +
this.id);
        }

        Registration registration = Registration.builder()
                .event(this)
                .user(user)
                .registrationDate(LocalDateTime.now())
                .status("PENDING")
                .build();

        this.registrations.add(registration);
        user.addRegistration(registration);

        return registration;
    }

    public void publish() {
        this.status = "PUBLISHED";
    }

    public void cancel() {
        this.status = "CANCELLED";
    }

    public void complete() {
```

```
            this status = "COMPLETED";
        }
    }
```

Registration.java:

```
package ru edu project entity;

import jakarta persistence *;
import lombok.*;
import java time LocalDateTime;

@Entity
@Table(name = "registrations",
    uniqueConstraints = @UniqueConstraint(columnNames = {"user_id",
"event_id"}))
@Getter
@Setter
@NoArgsConstructor
@AllArgsConstructor
@Builder
public class Registration {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(name = "registration_date", nullable = false)
    @Builder.Default
    private LocalDateTime registrationDate = LocalDateTime.now();
```

```
    @Column(nullable = false, length = 20)
    @Builder.Default
    private String status = "PENDING";

    @Column(name = "created_at", nullable = false)
    @Builder.Default
    private LocalDateTime createdAt = LocalDateTime.now();

    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "user_id", nullable = false)
    private User user;

    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "event_id", nullable = false)
    private Event event

    @OneToOne(mappedBy = "registration", cascade = CascadeType.ALL,
fetch = FetchType.LAZY)
    private Payment payment;

    // Бизнес-методы
    public void confirm() {
        this.status = "CONFIRMED";
    }

    public void cancel() {
        this.status = "CANCELLED";
    }
```

```
    public boolean isConfirmed() {
        return "CONFIRMED".equals(this.status);
    }

    public void addPayment(Payment payment) {
        this.payment = payment;
        payment.setRegistration(this);
    }
}
```

Payment.java:

```
package ru.edu.project.entity;

import jakarta.persistence.*;
import lombok.*;
import java.math.BigDecimal;
import java.time.LocalDateTime;

@Entity
@Table(name = "payments")
@Getter
@Setter
@NoArgsConstructor
@AllArgsConstructor
@Builder
public class Payment {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
```

```
    private Long id;

    @Column(nullable = false, precision = 10, scale = 2)
    private BigDecimal amount;

    @Column(name = "payment_date", nullable = false)
    @Builder.Default
    private LocalDateTime paymentDate = LocalDateTime.now();

    @Column(nullable = false, length = 50)
    private String method;

    @Column(name = "transaction_id", length = 100)
    private String transactionId;

    @Column(name = "created_at", nullable = false)
    @Builder.Default
    private LocalDateTime createdAt = LocalDateTime.now();

    @OneToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "registration_id", nullable = false, unique = true)
    private Registration registration;

    // Бизнес-методы
    public void refund() {
        // Логика возврата платежа
    }

    public boolean isCompleted() {
```

```
        return this.transactionId != null && !this.transactionId.isEmpty();
    }
}
```

3.4. Шаг 4: Пример использования сущностей

```
package ru.edu.project;

import ru.edu.project.entity.*;
import ru.edu.project.foundation.config.JpaUtil;
import jakarta.persistence.EntityManager;
import jakarta.persistence.EntityTransaction;
import java.math.BigDecimal;
import java.time.LocalDateTime;

public class Main {

    public static void main(String[] args) {
        EntityManager em = JpaUtil.getEntityManager();
        EntityTransaction tx = em.getTransaction();

        try {
            tx.begin();

            // Создание пользователя
            User admin = User.builder()
                    .email("admin@example.com")
                    .passwordHash("hashed_password")
                    .name("Администратор")
                    .role("ADMIN")
                    .build();
```

```
em.persist(admin);

// Создание категории
Category category = Category.builder()
        .name("Конференция")
        .description("Деловые конференции и форумы")
        .build();
em.persist(category);

// Создание мероприятия
Event event = Event.builder()
        .title("Java Conference 2025")
        .description("Главная Java-конференция года")
        .startDate(LocalDateTime.now().plusDays(30))
        .endDate(LocalDateTime.now().plusDays(31))
        .maxParticipants(100)
        .status("DRAFT")
        .category(category)
        .createdBy(admin)
        .build();
em.persist(event);

// Создание пользователя-участника
User user = User.builder()
        .email("user@example.com")
        .passwordHash("hashed_password")
        .name("Иван Иванов")
        .build();
em.persist(user);
```

```
        // Регистрация на мероприятие
        Registration registration = event.registerUser(user);
        em.persist(registration);

        // Создание платежа
        Payment payment = Payment.builder()
                .amount(new BigDecimal("5000.00"))
                .method("CARD")
                .transactionId("TXN-12345")
                .registration(registration)
                .build();
        em.persist(payment);

        // Подтверждение регистрации
        registration.confirm();

        tx.commit();

        System.out.println("Данные успешно сохранены!");

    } catch (Exception e) {
        if (tx.isActive()) {
            tx.rollback();
        }
        e.printStackTrace();
    } finally {
        em.close();
        JpaUtil.close();
```

}
}
}

4. МЕТОДИКА ВЫПОЛНЕНИЯ (ПОШАГОВАЯ ИНСТРУКЦИЯ)

Шаг 1: Подготовка окружения

Создать Maven-проект

Добавить зависимости в pom.xml

Создать файл persistence.xml с настройками подключения к БД

Создать класс JpaUtil для управления EntityManager

Шаг 2: Создание сущностей

Для каждой таблицы из ЛР5:

Создать Java-класс с аннотацией @Entity

Добавить аннотацию @Table с именем таблицы

Добавить поле id с аннотациями @Id и @GeneratedValue

Добавить остальные поля с аннотацией @Column

Добавить связи с помощью @ManyToOne, @OneToMany, @OneToOne

Добавить бизнес-методы

Шаг 3: Настройка связей

Таблица связей для заполнения:

Таблица (класс)	**Связь**	**Связанная таблица**	**Тип**	**В классе**
Event	category	Category	@ManyToOne	Event
Event	createdBy	User	@ManyToOne	Event

Таблица (класс)	Связь	Связанная таблица	Тип	В классе
Event	registrations	Registration	@OneToMany	Event
Registration	user	User	@ManyToOne	Registration
Registration	event	Event	@ManyToOne	Registration
Registration	payment	Payment	@OneToOne	Registration
Payment	registration	Registration	@OneToOne	Payment

Шаг 4: Добавление бизнес-методов

Для каждой сущности добавьте методы, отражающие бизнес-логику:

1. checkAvailability() - проверка мест
2. registerUser() - регистрация пользователя
3. confirm() / cancel() - изменение статуса
4. publish() / complete() - изменение статуса мероприятия

Шаг 5: Тестирование

Напишите простой main()-метод для сохранения данных

Проверьте, что данные корректно сохраняются в БД

Проверьте, что связи работают правильно

5. ВАРИАНТЫ ДЛЯ ИНДИВИДУАЛЬНОГО ВЫПОЛНЕНИЯ

Варианты соответствуют проектам из предыдущих лабораторных работ.

Минимальный набор JPA-сущностей для каждого проекта:

	Проект	**Сущности (минимум)**
	Система управления складом	Product, Warehouse, StockMovement, Supplier, Category
	Платформа аренды оборудования	Equipment, Rental, Customer, Maintenance, Category
	Система управления парком автомобилей	Vehicle, Driver, Trip, Maintenance, Fueling
	Система дистанционного обучения	Course, Module, Lesson, Student, Enrollment, Progress
	Платформа для проведения онлайн-курсов	Course, Instructor, Student, Video, Quiz, Certificate
	Система записи на кружки и секции	Circle, Schedule, Student, Teacher, Attendance, Payment
	Система медицинских назначений	Patient, Doctor, Appointment, Prescription, Diagnosis

	Проект	Сущности (минимум)
	Платформа записи на процедуры	Patient, Procedure, Equipment, Schedule, Result
	Система мониторинга здоровья	Patient, Measurement, Alert, Device, Doctor
0	Система управления доставкой еды	Restaurant, Dish, Order, Courier, Customer, Payment
1	Платформа бронирования услуг красоты	Salon, Master, Service, Client, Booking, Review
2	Система проката спортивного инвентаря	Item, Rental, Customer, Category, Maintenance
3	Система производственных заказов	Order, Product, Operation, Worker, Equipment
4	Платформа логистики	Shipment, Route, Vehicle, Driver, Tracking, Customer
5	Система управления проектами	Project, Task, Worker, Material, Equipment, Report

6. КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Что такое JPA и для чего он нужен?
2. Какие аннотации JPA используются для маппинга класса на таблицу?
3. Какие стратегии генерации первичных ключей существуют в JPA?
4. В чём разница между @OneToMany и @ManyToOne?
5. Что означают параметры fetch = FetchType.LAZY и fetch = FetchType.EAGER?
6. Для чего используется mappedBy в двунаправленной связи?
7. Что такое CascadeType и какие значения он принимает?
8. Как создать связь «многие-ко-многим» в JPA?
9. В чём разница между @JoinColumn и mappedBy?
10. Как обеспечить уникальность пары полей в JPA?
11. Почему рекомендуется использовать FetchType.LAZY по умолчанию?
12. Как решить проблему LazyInitializationException?
13. Как выполнить запрос с JOIN FETCH для загрузки связанных сущностей?
14. Как реализовать наследование в JPA (стратегии SINGLE_TABLE, JOINED, TABLE_PER_CLASS)?
15. Что такое @Embeddable и @Embedded и когда их использовать?
16. Как реализовать мягкое удаление (soft delete) в JPA?
17. Как реализовать версионирование для оптимистичной блокировки (@Version)?
18. Как создать кастомный тип данных в JPA?

7. ТРЕБОВАНИЯ К ОТЧЕТУ

1. Титульный лист
2. Содержание
3. Введение (цель работы, краткое описание проекта)
4. Настройка проекта
   4.1. pom.xml (зависимости)
   4.2. persistence.xml (настройки БД)
   4.3. JpaUtil (фабрика EntityManager)
5. JPA-сущности
   5.1. User (листинг кода + пояснения)
   5.2. Category (листинг кода + пояснения)
   5.3. Event (листинг кода + пояснения)
   5.4. Registration (листинг кода + пояснения)
   5.5. Payment (листинг кода + пояснения)
6. Бизнес-методы (описание ключевых методов)
7. Тестирование (пример сохранения данных)
8. Заключение (выводы, связь с последующими работами)
9. Список использованных источников
10. Приложения (полный код всех сущностей)

8. КРИТЕРИИ ОЦЕНИВАНИЯ

Критерий	**Макс. балл**	**Описание**
Полнота сущностей	4	Все таблицы из ЛР5 отображены в JPA
Корректность аннотаций	4	Правильное использование @Entity, @Table, @Column, @Id,

Критерий	Макс. балл	Описание
		@GeneratedValue
Правильность связей	5	Корректные @OneToMany, @ManyToOne, @OneToOne с указанием fetch, cascade
Бизнес-методы	3	Наличие ключевых бизнес-методов в сущностях
Настройка проекта	2	Корректные pom.xml, persistence.xml, JpaUtil
Оформление отчета	2	Соответствие требованиям
Ответы на контрольные вопросы	2	Понимание материала при защите
ИТОГО	22	

Лабораторная работа №7

Реализация Foundation-слоя (репозитории)

1. Цель и задачи работы

Цель: Освоить методику создания слоя доступа к данным (Foundation) с использованием паттерна Repository на основе JPA-сущностей, созданных в лабораторной работе №6.

Задачи:

1. Создать интерфейсы репозиториев для всех JPA-сущностей
2. Реализовать базовые CRUD-операции (create, read, update, delete)
3. Реализовать специфические методы поиска (findByEmail, findByStatus и т.д.)
4. Настроить управление транзакциями
5. Написать модульные тесты для репозиториев

2. ТЕОРЕТИЧЕСКОЕ ОБОСНОВАНИЕ

2.1. Место Foundation-слоя в архитектуре PCMEF

В архитектуре PCMEF Foundation-слой отвечает за доступ к данным. Он является посредником между бизнес-логикой (Mediator) и базой данных.

Database (PostgreSQL) ↓
Foundation (репозитории) ← **МЫ ЗДЕСЬ** ↓
Entity (JPA-сущности) ↓
Mediator (бизнес-логика)

2.2. Паттерн Repository

Repository - это паттерн проектирования, который абстрагирует доступ к данным, создавая коллекцию-подобный интерфейс для работы с сущностями.

Преимущества:

1. Изоляция бизнес-логики от деталей доступа к данным

2. Упрощение тестирования (можно мокировать репозиторий)
3. Централизация логики запросов
4. Лёгкость замены источника данных

Структура Repository:

```
public interface UserRepository {
    // Базовые CRUD
    User save(User user);
    Optional<User> findById(Long id);
    List<User> findAll();
    void delete(User user);
    void deleteById(Long id);

    // Специфические методы
    Optional<User> findByEmail(String email);
    List<User> findByRole(String role);
}
```

2.3. Управление транзакциями

Транзакция - это последовательность операций, которая выполняется как единое целое (ACID).

```
public User save(User user) {
    EntityTransaction transaction = entityManager.getTransaction();
    try {
        transaction.begin();
        entityManager.persist(user);
        transaction.commit();
        return user;
    } catch (Exception e) {
        if (transaction.isActive()) {
```

```
            transaction.rollback();
        }
        throw new RuntimeException("Failed to save user", e);
    }
}
```

2.4. Шаблоны методов репозитория

Тип метода	Синтаксис JPA	Пример
По ID	entityManager.find(Class, id)	findById(Long id)
Все записи	createQuery("SELECT e FROM Entity e")	findAll()
По условию	createQuery("... WHERE ...")	findByEmail(String email)
С пагинацией	setFirstResult() / setMaxResults()	findAll(int page, int size)
С сортировкой	ORDER BY field ASC/DESC	findAllSorted(String field)

2.5. Типы запросов в JPA

Тип запроса	**Описание**	**Пример**
JPQL	Java Persistence Query	SELECT u FROM

Тип запроса	Описание	Пример
	Language	User u WHERE u.email = :email
Criteria API	Типобезопасные запросы	cb.equal(root.get("email"), email)
Native Query	SQL-запросы	SELECT * FROM users WHERE email = ?

3. ПРИМЕР ВЫПОЛНЕНИЯ (СКВОЗНОЙ ПРОЕКТ)

Проект: «Система управления мероприятиями» (Event Management System)

3.1. Шаг 1: Создание базового интерфейса репозитория

BaseRepository.java:

```
package ru.edu.project.foundation.repositories;

import java.util.List;
import java.util.Optional;

/**
 * Базовый интерфейс репозитория с общими CRUD-операциями
 * @param <T> тип сущности
 * @param <ID> тип первичного ключа
 */
public interface BaseRepository<T, ID> {
```

```
/**
 * Сохранить сущность (INSERT)
 */
T save(T entity);

/**
 * Обновить сущность (UPDATE)
 */
T update(T entity);

/**
 * Найти по ID
 */
Optional<T> findById(ID id);

/**
 * Найти все записи
 */
List<T> findAll();

/**
 * Удалить сущность
 */
void delete(T entity);

/**
 * Удалить по ID
 */
```

```
    void deleteById(ID id);

    /**
     * Проверить существование по ID
     */
    boolean existsById(ID id);

    /**
     * Получить количество записей
     */
    long count();
}
```

3.2. Шаг 2: Реализация базового репозитория

BaseRepositoryImpl.java:

```
package ru.edu.project.foundation.repositories;

import jakarta.persistence.EntityManager;
import jakarta.persistence.EntityTransaction;
import jakarta.persistence.TypedQuery;
import java.util.List;
import java.util.Optional;

public abstract class BaseRepositoryImpl<T, ID> implements BaseRepository<T, ID> {

    protected final EntityManager entityManager;
    protected final Class<T> entityClass;
```

```
    public BaseRepositoryImpl(EntityManager entityManager, Class<T> entityClass) {
        this.entityManager = entityManager;
        this.entityClass = entityClass;
    }

    @Override
    public T save(T entity) {
        EntityTransaction transaction = entityManager.getTransaction();
        try {
            transaction.begin();
            entityManager.persist(entity);
            transaction.commit();
            return entity;
        } catch (Exception e) {
            if (transaction.isActive()) {
                transaction.rollback();
            }
            throw new RuntimeException("Failed to save entity: " + entityClass.getSimpleName(), e);
        }
    }

    @Override
    public T update(T entity) {
        EntityTransaction transaction = entityManager.getTransaction();
        try {
            transaction.begin();
```

```
            T merged = entityManager.merge(entity);
            transaction.commit();
            return merged;
        } catch (Exception e) {
            if (transaction.isActive()) {
                transaction.rollback();
            }
            throw new RuntimeException("Failed to update entity: " +
entityClass.getSimpleName(), e);
        }
    }

    @Override
    public Optional<T> findById(ID id) {
        return Optional.ofNullable(entityManager.find(entityClass, id));
    }

    @Override
    public List<T> findAll() {
        TypedQuery<T> query = entityManager.createQuery(
            "SELECT e FROM " + entityClass.getSimpleName() + " e",
entityClass);
        return query.getResultList();
    }

    @Override
    public void delete(T entity) {
        EntityTransaction transaction = entityManager.getTransaction();
        try {
```

```
                transaction.begin();
                entityManager.remove(entityManager.contains(entity) ? entity : entityManager.merge(entity));
                transaction.commit();
            } catch (Exception e) {
                if (transaction.isActive()) {
                    transaction.rollback();
                }
                throw new RuntimeException("Failed to delete entity: " + entityClass.getSimpleName(), e);
            }
        }

        @Override
        public void deleteById(ID id) {
            findById(id).ifPresent(this::delete);
        }

        @Override
        public boolean existsById(ID id) {
            return findById(id).isPresent();
        }

        @Override
        public long count() {
            TypedQuery<Long> query = entityManager.createQuery(
                "SELECT COUNT(e) FROM " + entityClass.getSimpleName() + " e", Long.class);
            return query.getSingleResult();
```

```
    }
}
```

3.3. Шаг 3: Создание специфических репозиториев

UserRepository.java:

```
package ru edu project foundation repositories;

import ru edu project entity.User;
import java util List;
import java util Optional;

public interface UserRepository extends BaseRepository<User, Long> {

    /**
     * Поиск пользователя по email
     */
    Optional<User> findByEmail(String email);

    /**
     * Поиск пользователей по роли
     */
    List<User> findByRole(String role);

    /**
     * Поиск пользователей по части имени (like)
     */
    List<User> findByNameContaining(String namePart);
```

```
    /**
     * Поиск пользователей, зарегистрированных после даты
     */
    List<User> findByCreatedAtAfter(java.time.LocalDateTime date);

    /**
     * Проверка существования email
     */
    boolean existsByEmail(String email);
}
```

UserRepositoryImpl.java:

```
package ru.edu.project.foundation.repositories;

import ru.edu.project.entity.User;
import jakarta.persistence.EntityManager;
import jakarta.persistence.TypedQuery;
import java.time.LocalDateTime;
import java.util.List;
import java.util.Optional;

public class UserRepositoryImpl extends BaseRepositoryImpl<User, Long>
implements UserRepository {

    public UserRepositoryImpl(EntityManager entityManager) {
        super(entityManager, User.class);
    }

    @Override
```

```
    public Optional<User> findByEmail(String email) {
        TypedQuery<User> query = entityManager.createQuery(
            "SELECT u FROM User u WHERE u.email = :email", User.class);
        query.setParameter("email", email);
        return query.getResultStream().findFirst();
    }

    @Override
    public List<User> findByRole(String role) {
        TypedQuery<User> query = entityManager.createQuery(
            "SELECT u FROM User u WHERE u.role = :role", User.class);
        query.setParameter("role", role);
        return query.getResultList();
    }

    @Override
    public List<User> findByNameContaining(String namePart) {
        TypedQuery<User> query = entityManager.createQuery(
            "SELECT u FROM User u WHERE u.name LIKE :name",
User.class);
        query.setParameter("name", "%" + namePart + "%");
        return query.getResultList();
    }

    @Override
    public List<User> findByCreatedAtAfter(LocalDateTime date) {
        TypedQuery<User> query = entityManager.createQuery(
            "SELECT u FROM User u WHERE u.createdAt > :date",
User.class);
```

```
        query.setParameter("date", date);
        return query.getResultList();
    }

    @Override
    public boolean existsByEmail(String email) {
        TypedQuery<Long> query = entityManager.createQuery(
            "SELECT COUNT(u) FROM User u WHERE u.email = :email",
Long.class);
        query.setParameter("email", email);
        return query.getSingleResult() > 0;
    }
}
```

EventRepository.java:

```
java
package ru.edu.project.foundation.repositories;

import ru.edu.project.entity.Event;
import java.time.LocalDateTime;
import java.util.List;
import java.util.Optional;

public interface EventRepository extends BaseRepository<Event, Long> {

    /**
     * Поиск мероприятий по статусу
     */
    List<Event> findByStatus(String status);
```

```
    /**
     * Поиск опубликованных мероприятий
     */
    List<Event> findPublishedEvents();

    /**
     * Поиск мероприятий по категории
     */
    List<Event> findByCategoryId(Long categoryId);

    /**
     * Поиск мероприятий, начинающихся после даты
     */
    List<Event> findByStartDateAfter(LocalDateTime date);

    /**
     * Поиск мероприятий в диапазоне дат
     */
    List<Event> findByStartDateBetween(LocalDateTime start,
LocalDateTime end);

    /**
     * Поиск мероприятий, созданных пользователем
     */
    List<Event> findByCreatedById(Long userId);

    /**
     * Поиск мероприятий с доступными местами
     */
```

```
    List<Event> findEventsWithAvailableSeats();
}
```

EventRepositoryImpl.java:

```
package ru.edu.project.foundation.repositories;

import ru.edu.project.entity.Event;
import jakarta.persistence.EntityManager;
import jakarta.persistence.TypedQuery;
import java.time.LocalDateTime;
import java.util.List;

public class EventRepositoryImpl extends BaseRepositoryImpl<Event, Long> implements EventRepository {

    public EventRepositoryImpl(EntityManager entityManager) {
        super(entityManager, Event.class);
    }

    @Override
    public List<Event> findByStatus(String status) {
        TypedQuery<Event> query = entityManager.createQuery(
            "SELECT e FROM Event e WHERE e.status = :status", Event.class);
        query.setParameter("status", status);
        return query.getResultList();
    }

    @Override
```

```
    public List<Event> findPublishedEvents() {
        TypedQuery<Event> query = entityManager.createQuery(
            "SELECT e FROM Event e WHERE e.status = 'PUBLISHED'",
Event.class);
        return query.getResultList();
    }

    @Override
    public List<Event> findByCategoryId(Long categoryId) {
        TypedQuery<Event> query = entityManager.createQuery(
            "SELECT e FROM Event e WHERE e.category.id = :categoryId",
Event.class);
        query.setParameter("categoryId", categoryId);
        return query.getResultList();
    }

    @Override
    public List<Event> findByStartDateAfter(LocalDateTime date) {
        TypedQuery<Event> query = entityManager.createQuery(
            "SELECT e FROM Event e WHERE e.startDate > :date",
Event.class);
        query.setParameter("date", date);
        return query.getResultList();
    }

    @Override
    public List<Event> findByStartDateBetween(LocalDateTime start,
LocalDateTime end) {
        TypedQuery<Event> query = entityManager.createQuery(
```

```
                "SELECT e FROM Event e WHERE e.startDate BETWEEN :start
AND :end", Event.class);
            query.setParameter("start", start);
            query.setParameter("end", end);
            return query.getResultList();
        }

        @Override
        public List<Event> findByCreatedById(Long userId) {
            TypedQuery<Event> query = entityManager.createQuery(
                "SELECT e FROM Event e WHERE e.createdBy.id = :userId",
Event.class);
            query.setParameter("userId", userId);
            return query.getResultList();
        }

        @Override
        public List<Event> findEventsWithAvailableSeats() {
            TypedQuery<Event> query = entityManager.createQuery(
                "SELECT e FROM Event e WHERE SIZE(e.registrations) <
e.maxParticipants " +
                "AND e.status = 'PUBLISHED'", Event.class);
            return query.getResultList();
        }
    }
```

RegistrationRepository.java:

```
package ru.edu.project.foundation.repositories;
```

```
import ru.edu.project.entity.Registration;
import java.time.LocalDateTime;
import java.util.List;
import java.util.Optional;

public interface RegistrationRepository extends BaseRepository<Registration, Long> {

    /**
     * Поиск регистраций пользователя
     */
    List<Registration> findByUserId(Long userId);

    /**
     * Поиск регистраций на мероприятие
     */
    List<Registration> findByEventId(Long eventId);

    /**
     * Поиск регистрации пользователя на мероприятие
     */
    Optional<Registration> findByUserIdAndEventId(Long userId, Long eventId);

    /**
     * Поиск регистраций по статусу
     */
    List<Registration> findByStatus(String status);
```

```
    /**
     * Поиск подтверждённых регистраций на мероприятие
     */
    List<Registration> findConfirmedByEventId(Long eventId);

    /**
     * Подсчёт участников мероприятия
     */
    long countConfirmedByEventId(Long eventId);

    /**
     * Поиск регистраций, созданных после даты
     */
    List<Registration> findByRegistrationDateAfter(LocalDateTime date);
}
```

RegistrationRepositoryImpl.java:

```
package ru.edu.project.foundation.repositories;

import ru.edu.project.entity.Registration;
import jakarta.persistence.EntityManager;
import jakarta.persistence.TypedQuery;
import java.time.LocalDateTime;
import java.util.List;
import java.util.Optional;

public class RegistrationRepositoryImpl extends
BaseRepositoryImpl<Registration, Long>
        implements RegistrationRepository {
```

```
    public RegistrationRepositoryImpl(EntityManager entityManager) {
        super(entityManager, Registration.class);
    }

    @Override
    public List<Registration> findByUserId(Long userId) {
        TypedQuery<Registration> query = entityManager.createQuery(
            "SELECT r FROM Registration r WHERE r.user.id = :userId",
Registration.class);
        query.setParameter("userId", userId);
        return query.getResultList();
    }

    @Override
    public List<Registration> findByEventId(Long eventId) {
        TypedQuery<Registration> query = entityManager.createQuery(
            "SELECT r FROM Registration r WHERE r.event.id = :eventId",
Registration.class);
        query.setParameter("eventId", eventId);
        return query.getResultList();
    }

    @Override
    public Optional<Registration> findByUserIdAndEventId(Long userId,
Long eventId) {
        TypedQuery<Registration> query = entityManager.createQuery(
            "SELECT r FROM Registration r WHERE r.user.id = :userId AND
r.event.id = :eventId",
```

```
                Registration.class);
        query.setParameter("userId", userId);
        query.setParameter("eventId", eventId);
        return query.getResultStream().findFirst();
    }

    @Override
    public List<Registration> findByStatus(String status) {
        TypedQuery<Registration> query = entityManager.createQuery(
                "SELECT r FROM Registration r WHERE r.status = :status",
Registration.class);
        query.setParameter("status", status);
        return query.getResultList();
    }

    @Override
    public List<Registration> findConfirmedByEventId(Long eventId) {
        TypedQuery<Registration> query = entityManager.createQuery(
                "SELECT r FROM Registration r WHERE r.event.id = :eventId
AND r.status = 'CONFIRMED'",
                Registration.class);
        query.setParameter("eventId", eventId);
        return query.getResultList();
    }

    @Override
    public long countConfirmedByEventId(Long eventId) {
        TypedQuery<Long> query = entityManager.createQuery(
```

```
                "SELECT COUNT(r) FROM Registration r WHERE r.event.id
= :eventId AND r.status = 'CONFIRMED'",
                Long.class);
        query.setParameter("eventId", eventId);
        return query.getSingleResult();
    }

    @Override
    public List<Registration> findByRegistrationDateAfter(LocalDateTime
date) {
        TypedQuery<Registration> query = entityManager.createQuery(
                "SELECT r FROM Registration r WHERE
r.registrationDate > :date", Registration.class);
        query.setParameter("date", date);
        return query.getResultList();
    }
}
```

3.4. Шаг 4: Модульное тестирование репозиториев

UserRepositoryTest.java:

```
package ru.edu.project.foundation.repositories;

import ru.edu.project.entity.User;
import ru.edu.project.foundation.config.JpaUtil;
import jakarta.persistence.EntityManager;
import org.junit.jupiter.api.*;

import java.time.LocalDateTime;
```

```java
import java.util.List;

import static org.junit.jupiter.api.Assertions.*;

@TestInstance(TestInstance.Lifecycle.PER_CLASS)
class UserRepositoryTest {

    private UserRepository userRepository;
    private EntityManager entityManager;

    @BeforeAll
    void setUp() {
        entityManager = JpaUtil.getEntityManager();
        userRepository = new UserRepositoryImpl(entityManager);
    }

    @AfterAll
    void tearDown() {
        entityManager.close();
        JpaUtil.close();
    }

    @BeforeEach
    void cleanUp() {
        // Очистка таблицы перед каждым тестом
        userRepository.findAll().forEach(userRepository::delete);
    }

    @Test
```

```
void testSaveAndFindById() {
    // given
    User user = User.builder()
            .email("test@example.com")
            .passwordHash("hashed_password")
            .name("Test User")
            .role("USER")
            .build();

    // when
    User saved = userRepository.save(user);

    // then
    assertNotNull(saved.getId());
    assertEquals("test@example.com", saved.getEmail());

    Optional<User> found = userRepository.findById(saved.getId());
    assertTrue(found.isPresent());
    assertEquals("Test User", found.get().getName());
}

@Test
void testFindByEmail() {
    // given
    User user = User.builder()
            .email("unique@example.com")
            .passwordHash("hashed_password")
            .name("Unique User")
            .build();
```

```
        userRepository.save(user);

        // when
        Optional<User> found = userRepository.findByEmail("unique@example.com");

        // then
        assertTrue(found.isPresent());
        assertEquals("Unique User", found.get().getName());
    }

    @Test
    void testFindByRole() {
        // given
        User admin = User.builder()
                .email("admin@example.com")
                .passwordHash("hashed")
                .name("Admin")
                .role("ADMIN")
                .build();
        User user = User.builder()
                .email("user@example.com")
                .passwordHash("hashed")
                .name("User")
                .role("USER")
                .build();
        userRepository.save(admin);
        userRepository.save(user);
```

```
        // when
        List<User> admins = userRepository.findByRole("ADMIN");
        List<User> users = userRepository.findByRole("USER");

        // then
        assertEquals(1, admins.size());
        assertEquals(1, users.size());
        assertEquals("Admin", admins.get(0).getName());
        assertEquals("User", users.get(0).getName());
    }

    @Test
    void testExistsByEmail() {
        // given
        User user = User.builder()
                .email("exists@example.com")
                .passwordHash("hashed")
                .name("Exists")
                .build();
        userRepository.save(user);

        // when
        boolean exists = userRepository.existsByEmail("exists@example.com");
        boolean notExists = userRepository.existsByEmail("nonexistent@example.com");

        // then
        assertTrue(exists);
```

```
    assertFalse(notExists);
}

@Test
void testUpdate() {
    // given
    User user = User.builder()
            .email("update@example.com")
            .passwordHash("old_hash")
            .name("Old Name")
            .build();
    user = userRepository.save(user);

    // when
    user.setName("New Name");
    user.setPasswordHash("new_hash");
    User updated = userRepository.update(user);

    // then
    assertEquals("New Name", updated.getName());
    assertEquals("new_hash", updated.getPasswordHash());

    Optional<User> found = userRepository.findById(user.getId());
    assertTrue(found.isPresent());
    assertEquals("New Name", found.get().getName());
}

@Test
void testDelete() {
```

```
        // given
        User user = User.builder()
                .email("delete@example.com")
                .passwordHash("hashed")
                .name("To Delete")
                .build();
        user = userRepository.save(user);
        Long id = user.getId();

        // when
        userRepository.delete(user);

        // then
        Optional<User> found = userRepository.findById(id);
        assertFalse(found.isPresent());
    }

    @Test
    void testCount() {
        // given
        long beforeCount = userRepository.count();

        User user1 = User.builder().email("count1@example.com").passwordHash("hash").name("Count1").build();
        User user2 = User.builder().email("count2@example.com").passwordHash("hash").name("Count2").build();
        userRepository.save(user1);
```

```
        userRepository.save(user2);

        // when
        long afterCount = userRepository.count();

        // then
        assertEquals(beforeCount + 2, afterCount);
    }
}
```

4. МЕТОДИКА ВЫПОЛНЕНИЯ (ПОШАГОВАЯ ИНСТРУКЦИЯ)

Шаг 1: Создание базового репозитория

Создать интерфейс BaseRepository<T, ID> с общими методами

Создать абстрактный класс BaseRepositoryImpl<T, ID>, реализующий интерфейс

Реализовать базовые CRUD-операции через EntityManager

Шаг 2: Создание специфических репозиториев

Для каждой сущности из ЛР6:

Создать интерфейс XxxRepository extends BaseRepository<Xxx, Long>

Добавить специфические методы поиска (по уникальным полям, по условиям)

Создать реализацию XxxRepositoryImpl extends BaseRepositoryImpl<Xxx, Long> implements XxxRepository

Реализовать все специфические методы

Таблица методов для заполнения:

Сущность	Специфические методы	Обоснование
User	findByEmail, findByRole,	Поиск по

Сущность	Специфические методы	Обоснование
	findByNameContaining	бизнес-полям
Event	findByStatus, findByCategoryId, findByStartDateAfter	Фильтрация мероприятий
Registration	findByUserId, findByEventId, findByUserIdAndEventId	Уникальность регистрации
Category	findByName	Поиск по имени
Payment	findByRegistrationId	Связь с регистрацией

Шаг 3: Настройка управления транзакциями

Реализовать транзакционную обёртку для каждого метода модификации данных

Обеспечить корректный откат (rollback) при ошибках

```
EntityTransaction transaction = entityManager getTransaction();
try {
    transaction begin();
    // операции с БД
    transaction commit();
} catch (Exception e) {
    if (transaction isActive()) {
        transaction rollback();
    }
```

```
    throw e;
}
```

Шаг 4: Модульное тестирование

Настроить тестовую базу данных (H2)

Написать тесты для CRUD-операций

Написать тесты для специфических методов поиска

Проверить корректность работы транзакций

5. ВАРИАНТЫ ДЛЯ ИНДИВИДУАЛЬНОГО ВЫПОЛНЕНИЯ

Варианты соответствуют проектам из предыдущих лабораторных работ.

Минимальный набор репозиториев для каждого проекта:

№	Проект	Репозитории
1	Система управления складом	Product, Warehouse, StockMovement, Supplier, Category
2	Платформа аренды оборудования	Equipment, Rental, Customer, Maintenance, Category
3	Система управления парком автомобилей	Vehicle, Driver, Trip, Maintenance, Fueling
4	Система дистанционного обучения	Course, Module, Lesson, Student, Enrollment, Progress
5	Платформа для проведения онлайн-	Course, Instructor, Student, Video, Quiz, Certificate

№	Проект	Репозитории
	курсов	
6	Система записи на кружки и секции	Circle, Schedule, Student, Teacher, Attendance, Payment
7	Система медицинских назначений	Patient, Doctor, Appointment, Prescription, Diagnosis
8	Платформа записи на процедуры	Patient, Procedure, Equipment, Schedule, Result
9	Система мониторинга здоровья	Patient, Measurement, Alert, Device, Doctor
10	Система управления доставкой еды	Restaurant, Dish, Order, Courier, Customer, Payment
11	Платформа бронирования услуг красоты	Salon, Master, Service, Client, Booking, Review
12	Система проката спортивного инвентаря	Item, Rental, Customer, Category, Maintenance
13	Система производственных заказов	Order, Product, Operation, Worker, Equipment

№	Проект	Репозитории
14	Платформа логистики	Shipment, Route, Vehicle, Driver, Tracking, Customer
15	Система управления проектами	Project, Task, Worker, Material, Equipment, Report

6. КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Что такое паттерн Repository и для чего он нужен?
2. Какие базовые CRUD-операции должны быть в репозитории?
3. Как в JPA выполнить поиск по уникальному полю?
4. Что такое JPQL и чем он отличается от SQL?
5. Как управлять транзакциями в JPA?
6. Почему методы репозитория возвращают Optional<T> для поиска по ID?
7. Как реализовать поиск с пагинацией?
8. Как реализовать поиск с сортировкой?
9. Что такое LazyInitializationException и как её избежать?
10. Как протестировать репозиторий с реальной базой данных?
11. Как реализовать динамические запросы с Criteria API?
12. Как реализовать поиск с несколькими параметрами фильтрации?
13. Как реализовать пагинацию с общим количеством страниц?
14. Как реализовать кэширование запросов на уровне репозитория?
15. Как реализовать аудит изменений (кто и когда изменил запись)?

7. ТРЕБОВАНИЯ К ОТЧЕТУ

Структура отчета

1. Титульный лист

2. Содержание

3. Введение (цель работы, краткое описание проекта)

4. Базовый репозиторий

   4.1. BaseRepository (интерфейс)

   4.2. BaseRepositoryImpl (реализация)

5. Специфические репозитории

   5.1. UserRepository (интерфейс + реализация)

   5.2. EventRepository (интерфейс + реализация)

   5.3. RegistrationRepository (интерфейс + реализация)

   5.4. Другие репозитории по необходимости

6. Транзакционное управление (объяснение)

7. Модульное тестирование

   7.1. Тесты для UserRepository

   7.2. Тесты для EventRepository

   7.3. Результаты тестирования

8. Заключение (выводы, связь с последующими работами)

9. Список использованных источников

10. Приложения (полный код репозиториев)

8. КРИТЕРИИ ОЦЕНИВАНИЯ

Критерий	**Макс. балл**	**Описание**
Базовый репозиторий	3	Корректная реализация BaseRepository и BaseRepositoryImpl
Специфические репозитории	5	Все необходимые репозитории созданы, методы реализованы

Критерий	Макс. балл	Описание
Специфические методы поиска	4	Корректные JPQL-запросы, параметры, типы возвращаемых значений
Управление транзакциями	2	begin/commit/rollback, обработка ошибок
Модульное тестирование	4	Наличие тестов, покрытие основных операций
Оформление отчета	2	Соответствие требованиям
Ответы на контрольные вопросы	2	Понимание материала при защите
ИТОГО	22	

Лабораторная работа №8

Модульное тестирование (JUnit + Mockito)

1. Цель и задачи работы

Цель: Освоить методику модульного тестирования Java-приложений с использованием фреймворков JUnit 5 и Mockito для проверки работоспособности слоёв Entity и Foundation в архитектуре PCMEF.

Задачи:

Настроить тестовое окружение с JUnit 5 и Mockito

Написать модульные тесты для бизнес-методов сущностей (Entity)

Написать модульные тесты для репозиториев (Foundation) с использованием моков

Обеспечить покрытие кода тестами не менее 40%

Сгенерировать отчёт о покрытии (JaCoCo)

2. ТЕОРЕТИЧЕСКОЕ ОБОСНОВАНИЕ

2.1. Что такое модульное тестирование?

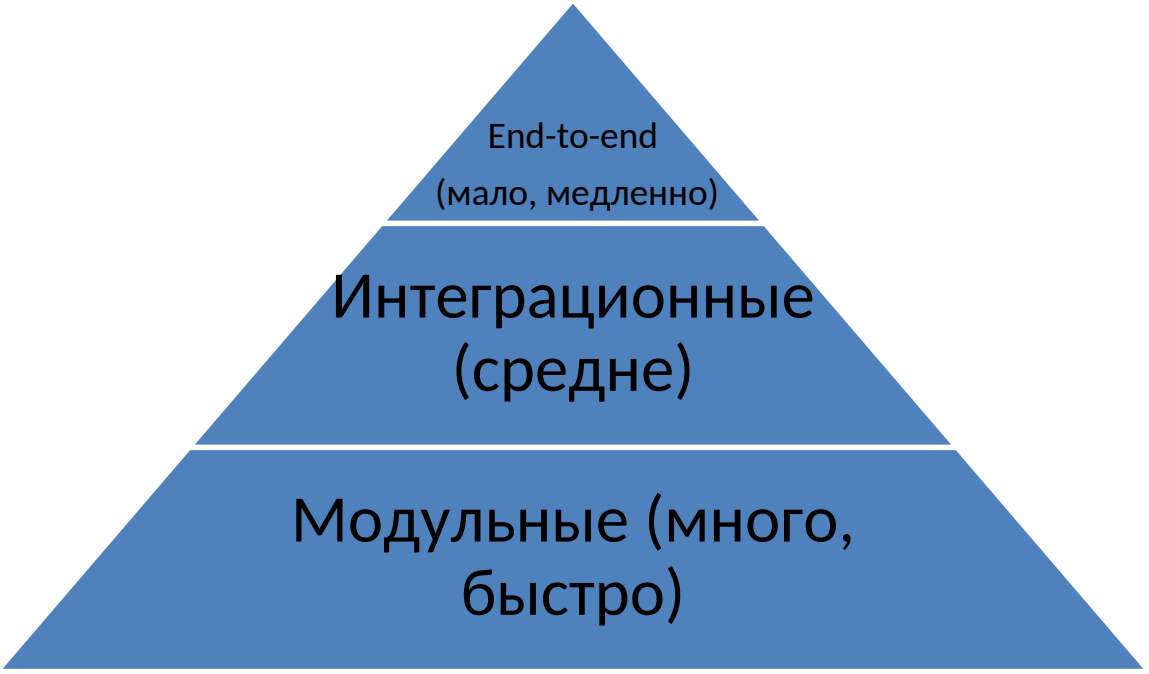
Модульное тестирование (Unit Testing) - это процесс проверки отдельных компонентов (модулей) программы в изоляции от остальной системы.

Цели модульного тестирования:

1. Обнаружение ошибок на ранних этапах
2. Обеспечение корректности работы при изменении кода (регрессия)
3. Документирование ожидаемого поведения
4. Упрощение рефакторинга

2.2. Пирамида тестирования

-



2.3. JUnit 5 — основные аннотации

Аннотация	**Описание**	**Пример**
@Test	Объявляет тестовый метод	@Test void testSomething() { ... }
@BeforeEach	Выполняется перед каждым тестом	@BeforeEach void setUp() { ... }
@AfterEach	Выполняется после каждого теста	@AfterEach void tearDown() { ... }
@BeforeAll	Выполняется один раз перед всеми тестами	@BeforeAll static void init() { ... }
@AfterA	Выполняется один раз	@AfterAll static

Аннотация	Описание	Пример
ll	после всех тестов	void cleanup() { ... }
@Disabled	Отключает тест	@Disabled("Not implemented yet")
@DisplayName	Задаёт читаемое имя	@DisplayName("Проверка регистрации")
@Nested	Вложенный тестовый класс	@Nested class UserRegistrationTest { ... }

2.4. Основные assertions (проверки)

Метод	Описание	Пример
assertEquals(expected, actual)	Проверка равенства	assertEquals(5, result)
assertNotEquals(expected, actual)	Проверка неравенства	assertNotEquals(0, result)
assertTrue(condition)	Проверка истинности	assertTrue(user.isActive())
assertFalse(condition)	Проверка ложности	assertFalse(list.isEmpty())

Метод	Описание	Пример
assertNotNull(object)	Проверка, что не null	assertNotNull(user)
assertNull(object)	Проверка, что null	assertNull(user.getDeletedAt()
assertThrows(exception, executable)	Проверка исключения	assertThrows(IllegalStateException, () -> { ... })
assertAll(executables...)	Групповая проверка	assertAll(() -> ..., () -> ...)

2.5. Mockito - создание моков

Mock — это объект-заглушка, имитирующий поведение реального объекта для изолированного тестирования.

Аннотация/Метод	Описание	Пример
@Mock	Создаёт мок-объект	@Mock private UserRepository userRepository;
@InjectMocks	Внедряет моки в тестируемый объект	@InjectMocks private UserService userService;

Аннотация/Метод	Описание	Пример
@ExtendWith(MockitoExtension.class)	Включает поддержку Mockito	
when(x).thenReturn(y)	Настройка поведения мока	when(repo.findById(1L)).thenReturn(Optional.of(user));
verify(x).method()	Проверка вызова метода	verify(repo).save(any());
any()	Любой аргумент	any(User.class)
never()	Проверка, что метод не вызывался	verify(repo, never()).delete(any());

2.6. Параметризованные тесты

```
@ParameterizedTest
@ValueSource(strings = {"user1@example.com", "user2@example.com"})
void testEmails(String email) {
    assertTrue(email.contains("@"));
}

@ParameterizedTest
@CsvSource({"1, 2, 3", "4, 5, 9"})
void testSum(int a, int b, int expected) {
```

```
    assertEquals(expected, a + b);
}
```

2.7. JaCoCo — отчёт о покрытии

JaCoCo (Java Code Coverage) — инструмент для измерения покрытия кода тестами.

Типы покрытия:

1. Instruction coverage — покрытие инструкций (байт-кода)
2. Branch coverage — покрытие ветвей (if/else, switch)
3. Line coverage — покрытие строк кода
4. Method coverage — покрытие методов

3. ПРИМЕР ВЫПОЛНЕНИЯ (СКВОЗНОЙ ПРОЕКТ)

Проект: «Система управления мероприятиями» (Event Management System)

3.1. Шаг 1: Добавление зависимостей в pom.xml

```
<dependencies>
    <!-- ... существующие зависимости ... -->

    <!-- JUnit 5 -->
    <dependency>
        <groupId>org.junit.jupiter</groupId>
        <artifactId>junit-jupiter</artifactId>
        <version>5.10.0</version>
        <scope>test</scope>
    </dependency>

    <!-- Mockito -->
    <dependency>
        <groupId>org.mockito</groupId>
```

```
        <artifactId>mockito-core</artifactId>
        <version>5.7.0</version>
        <scope>test</scope>
    </dependency>

    <!-- Mockito JUnit 5 integration -->
    <dependency>
        <groupId>org.mockito</groupId>
        <artifactId>mockito-junit-jupiter</artifactId>
        <version>5.7.0</version>
        <scope>test</scope>
    </dependency>

    <!-- H2 Database for testing -->
    <dependency>
        <groupId>com.h2database</groupId>
        <artifactId>h2</artifactId>
        <version>2.2.224</version>
        <scope>test</scope>
    </dependency>
</dependencies>

<build>
    <plugins>
        <!-- JaCoCo plugin -->
        <plugin>
            <groupId>org.jacoco</groupId>
            <artifactId>jacoco-maven-plugin</artifactId>
            <version>0.8.10</version>
```

```
            <executions>
                <execution>
                    <goals>
                        <goal>prepare-agent</goal>
                        <goal>report</goal>
                    </goals>
                </execution>
            </executions>
        </plugin>

        <!-- Surefire plugin for tests -->
        <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-surefire-plugin</artifactId>
            <version>3.1.2</version>
        </plugin>
    </plugins>
</build>
```

3.2. Шаг 2: Тестирование Entity-слоя (бизнес-методы)

EventTest.java:

```
package ru.edu.project.entity;

import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.DisplayName;
import org.junit.jupiter.api.Nested;
import org.junit.jupiter.api.Test;
import java.math.BigDecimal;
```

```java
import java.time.LocalDateTime;
import static org.junit.jupiter.api.Assertions.*;

@DisplayName("Тестирование сущности Event")
class EventTest {

    private Event event;
    private User user;

    @BeforeEach
    void setUp() {
        user = User.builder()
                .id(1L)
                .email("test@example.com")
                .name("Test User")
                .build();

        event = Event.builder()
                .id(1L)
                .title("Java Conference 2025")
                .description("Main Java conference")
                .startDate(LocalDateTime.now().plusDays(30))
                .endDate(LocalDateTime.now().plusDays(31))
                .maxParticipants(10)
                .status("DRAFT")
                .build();
    }

    @Nested
```

```
@DisplayName("Тесты проверки доступности мест")
class AvailabilityTests {

    @Test
    @DisplayName("Должно вернуть true при наличии мест")
    void shouldReturnTrueWhenSeatsAvailable() {
        assertTrue(event.checkAvailability());
    }

    @Test
    @DisplayName("Должно вернуть false при отсутствии мест")
    void shouldReturnFalseWhenNoSeats() {
        for (int i = 0; i < event.getMaxParticipants(); i++) {
            event.registerUser(User.builder().id((long) i).build());
        }
        assertFalse(event.checkAvailability());
    }

    @Test
    @DisplayName("Должен вернуть правильное количество свободных мест")
    void shouldReturnCorrectAvailableSeats() {
        assertEquals(10, event.getAvailableSeats());

        event.registerUser(User.builder().id(1L).build());
        assertEquals(9, event.getAvailableSeats());

        event.registerUser(User.builder().id(2L).build());
        assertEquals(8, event.getAvailableSeats());
```

```
    }
}

@Nested
@DisplayName("Тесты регистрации пользователей")
class RegistrationTests {

    @Test
    @DisplayName("Должен создать регистрацию при наличии мест")
    void shouldCreateRegistrationWhenSeatsAvailable() {
        Registration registration = event.registerUser(user);

        assertNotNull(registration);
        assertEquals(event, registration.getEvent());
        assertEquals(user, registration.getUser());
        assertNotNull(registration.getRegistrationDate());
        assertEquals("PENDING", registration.getStatus());
    }

    @Test
    @DisplayName("Должен выбросить исключение при отсутствии
мест")
    void shouldThrowExceptionWhenNoSeats() {
        for (int i = 0; i < event.getMaxParticipants(); i++) {
            event.registerUser(User.builder().id((long) i).build());
        }

        assertThrows(IllegalStateException.class, () -> {
            event.registerUser(user);
```

```
        });
    }
}

@Nested
@DisplayName("Тесты изменения статуса")
class StatusTests {

    @Test
    @DisplayName("Должен опубликовать мероприятие")
    void shouldPublishEvent() {
        event.publish();
        assertEquals("PUBLISHED", event.getStatus());
    }

    @Test
    @DisplayName("Должен отменить мероприятие")
    void shouldCancelEvent() {
        event.cancel();
        assertEquals("CANCELLED", event.getStatus());
    }

    @Test
    @DisplayName("Должен завершить мероприятие")
    void shouldCompleteEvent() {
        event.complete();
        assertEquals("COMPLETED", event.getStatus());
    }
}
```

}

UserTest.java:

```
package ru edu project entity;

import org junit jupiter api BeforeEach;
import org junit jupiter api DisplayName;
import org junit jupiter api Test;
import static org junit jupiter api Assertions.*;

@DisplayName("Тестирование сущности User")
class UserTest {

    private User user;

    @BeforeEach
    void setUp() {
        user = User.builder()
                .id(1L)
                .email("test@example.com")
                .passwordHash("hashed_password_123")
                .name("Иван Иванов")
                .role("USER")
                .build();
    }

    @Test
    @DisplayName("Должен проверить пароль")
    void shouldCheckPassword() {
```

```
        // given
        user.setPasswordHash("$2a$10$encrypted_password");

        // then
        // В реальном проекте пароль проверяется через BCrypt
        assertTrue(user.checkPassword("$2a$10$encrypted_password"));
        assertFalse(user.checkPassword("wrong_password"));
    }

    @Test
    @DisplayName("Должен добавить регистрацию")
    void shouldAddRegistration() {
        Registration registration = Registration.builder().build();
        user.addRegistration(registration);

        assertTrue(user.getRegistrations().contains(registration));
        assertEquals(user, registration.getUser());
    }
}
```

RegistrationTest.java:

```
package ru.edu.project.entity;

import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.DisplayName;
import org.junit.jupiter.api.Nested;
import org.junit.jupiter.api.Test;
import java.time.LocalDateTime;
import static org.junit.jupiter.api.Assertions.*;
```

```java
@DisplayName("Тестирование сущности Registration")
class RegistrationTest {

    private Registration registration;

    @BeforeEach
    void setUp() {
        registration = Registration.builder()
                .id(1L)
                .registrationDate(LocalDateTime.now())
                .status("PENDING")
                .build();
    }

    @Nested
    @DisplayName("Тесты изменения статуса")
    class StatusChangeTests {

        @Test
        @DisplayName("Должен подтвердить регистрацию")
        void shouldConfirmRegistration() {
            registration.confirm();
            assertEquals("CONFIRMED", registration.getStatus());
        }

        @Test
        @DisplayName("Должен отменить регистрацию")
        void shouldCancelRegistration() {
```

```
        registration.cancel();
        assertEquals("CANCELLED", registration.getStatus());
    }
}

@Test
@DisplayName("Должен проверить, что регистрация подтверждена")
void shouldCheckConfirmed() {
    assertFalse(registration.isConfirmed());

    registration.confirm();
    assertTrue(registration.isConfirmed());
}

@Test
@DisplayName("Должен добавить платёж")
void shouldAddPayment() {
    Payment payment = Payment.builder().amount(new
java.math.BigDecimal("1000")).build();
    registration.addPayment(payment);

    assertEquals(registration, payment.getRegistration());
    assertNotNull(registration.getPayment());
}
}
```

3.3. Шаг 3: Тестирование Foundation-слоя (репозиториев) с моками

UserRepositoryTest.java (с моками):

```
package ru.edu.project.foundation.repositories;
```

```
import ru.edu.project.entity.User;
import jakarta.persistence.EntityManager;
import jakarta.persistence.EntityTransaction;
import jakarta.persistence.TypedQuery;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.DisplayName;
import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.extension.ExtendWith;
import org.mockito.InjectMocks;
import org.mockito.Mock;
import org.mockito.junit.jupiter.MockitoExtension;

import java.util.List;
import java.util.Optional;

import static org.junit.jupiter.api.Assertions.*;
import static org.mockito.ArgumentMatchers.*;
import static org.mockito.Mockito.*;

@ExtendWith(MockitoExtension.class)
@DisplayName("Тестирование UserRepository")
class UserRepositoryTest {

    @Mock
    private EntityManager entityManager;

    @Mock
    private EntityTransaction transaction;
```

```
    @Mock
    private TypedQuery<User> query;

    @InjectMocks
    private UserRepositoryImpl userRepository;

    private User testUser;

    @BeforeEach
    void setUp() {
        testUser = User.builder()
                .id(1L)
                .email("test@example.com")
                .passwordHash("hashed")
                .name("Test User")
                .role("USER")
                .build();
    }

    @Test
    @DisplayName("Сохранение пользователя должно вызвать persist и commit")
    void saveShouldCallPersistAndCommit() {
        // given
        when(entityManager.getTransaction()).thenReturn(transaction);
        when(transaction.isActive()).thenReturn(true);

        // when
```

```
        User saved = userRepository.save(testUser);

        // then
        verify(entityManager).persist(testUser);
        verify(transaction).begin();
        verify(transaction).commit();
        assertEquals(testUser, saved);
    }

    @Test
    @DisplayName("Поиск пользователя по ID должен вернуть Optional с пользователем")
    void findByIdShouldReturnOptionalWithUser() {
        // given
        when(entityManager.find(eq(User.class), eq(1L))).thenReturn(testUser);

        // when
        Optional<User> found = userRepository.findById(1L);

        // then
        assertTrue(found.isPresent());
        assertEquals("test@example.com", found.get().getEmail());
        verify(entityManager).find(User.class, 1L);
    }

    @Test
    @DisplayName("Поиск пользователя по несуществующему ID должен вернуть пустой Optional")
```

```
    void findByIdShouldReturnEmptyOptionalWhenNotFound() {
        // given
        when(entityManager.find(eq(User.class), eq(999L))).thenReturn(null);

        // when
        Optional<User> found = userRepository.findById(999L);

        // then
        assertFalse(found.isPresent());
    }

    @Test
    @DisplayName("Получение всех пользователей должно вернуть список")
    void findAllShouldReturnList() {
        // given
        List<User> expectedUsers = List.of(testUser);
        when(entityManager.createQuery(anyString(),
eq(User.class))).thenReturn(query);
        when(query.getResultList()).thenReturn(expectedUsers);

        // when
        List<User> users = userRepository.findAll();

        // then
        assertEquals(1, users.size());
        assertEquals("test@example.com", users.get(0).getEmail());
    }
```

```
    @Test
    @DisplayName("Удаление пользователя должно вызвать remove и commit")
    void deleteShouldCallRemoveAndCommit() {
        // given
        when(entityManager.getTransaction()).thenReturn(transaction);
        when(entityManager.contains(testUser)).thenReturn(true);

        // when
        userRepository.delete(testUser);

        // then
        verify(entityManager).remove(testUser);
        verify(transaction).begin();
        verify(transaction).commit();
    }

    @Test
    @DisplayName("Поиск пользователя по email должен вернуть Optional")
    void findByEmailShouldReturnOptional() {
        // given
        when(entityManager.createQuery(anyString(), eq(User.class))).thenReturn(query);
        when(query.setParameter(eq("email"), eq("test@example.com"))).thenReturn(query);
        when(query.getResultStream()).thenReturn(List.of(testUser).stream());

        // when
```

```
        Optional<User> found = userRepository.findByEmail("test@example.com");

        // then
        assertTrue(found.isPresent());
        assertEquals("test@example.com", found.get().getEmail());
    }

    @Test
    @DisplayName("Проверка существования email должен вернуть true")
    void existsByEmailShouldReturnTrue() {
        // given
        when(entityManager.createQuery(anyString(), eq(Long.class))).thenReturn(query);
        when(query.setParameter(eq("email"), eq("test@example.com"))).thenReturn(query);
        when(query.getSingleResult()).thenReturn(1L);

        // when
        boolean exists = userRepository.existsByEmail("test@example.com");

        // then
        assertTrue(exists);
    }
}
```

3.4. Шаг 4: Интеграционное тестирование с H2 (опционально)

UserRepositoryIntegrationTest.java:

```java
package ru.edu.project.foundation.repositories;

import ru.edu.project.entity.User;
import ru.edu.project.foundation.config.JpaUtil;
import jakarta.persistence.EntityManager;
import org.junit.jupiter.api.*;

import static org.junit.jupiter.api.Assertions.*;

@TestInstance(TestInstance.Lifecycle.PER_CLASS)
class UserRepositoryIntegrationTest {

    private UserRepository userRepository;
    private EntityManager entityManager;

    @BeforeAll
    void setUp() {
        entityManager = JpaUtil.getEntityManager();
        userRepository = new UserRepositoryImpl(entityManager);
    }

    @AfterAll
    void tearDown() {
        entityManager.close();
        JpaUtil.close();
    }

    @BeforeEach
    void cleanUp() {
```

```
        userRepository.findAll().forEach(userRepository::delete);
    }

    @Test
    void testSaveAndFind() {
        User user = User.builder()
                .email("integration@example.com")
                .passwordHash("hashed")
                .name("Integration Test")
                .build();

        User saved = userRepository.save(user);

        assertNotNull(saved.getId());

        var found = userRepository.findById(saved.getId());
        assertTrue(found.isPresent());
        assertEquals("Integration Test", found.get().getName());
    }
}
```

3.5. Шаг 5: Запуск тестов и генерация отчёта JaCoCo

Команды Maven:

```
# Запуск всех тестов
mvn test

# Генерация отчёта JaCoCo
mvn jacoco:report
```

```
# Полный цикл: тесты + отчёт
mvn clean test jacoco:report
```

Отчёт JaCoCo будет доступен по адресу: target/site/jacoco/index.html

4. МЕТОДИКА ВЫПОЛНЕНИЯ (ПОШАГОВАЯ ИНСТРУКЦИЯ)

Шаг 1: Настройка тестового окружения

Добавить зависимости JUnit 5, Mockito, H2 в pom.xml

Настроить плагин JaCoCo в pom.xml

Создать тестовую директорию src/test/java

Шаг 2: Тестирование Entity-слоя

Для каждой сущности:

Создать тестовый класс XxxTest.java

Написать тесты для конструкторов и сеттеров

Написать тесты для бизнес-методов (checkAvailability, registerUser и др.)

Написать тесты для проверки исключений

Шаг 3: Тестирование Foundation-слоя

Для каждого репозитория:

Создать тестовый класс XxxRepositoryTest.java

Настроить моки с помощью @Mock и @InjectMocks

Написать тесты для CRUD-операций

Написать тесты для специфических методов поиска

Шаг 4: Проверка покрытия

Запустить тесты: mvn clean test

Сгенерировать отчёт JaCoCo: mvn jacoco:report

Открыть target/site/jacoco/index.html

Проверить, что покрытие >40%

При необходимости добавить тесты

5. ВАРИАНТЫ ДЛЯ ИНДИВИДУАЛЬНОГО ВЫПОЛНЕНИЯ

Варианты соответствуют проектам из предыдущих лабораторных работ.

Минимальный набор тестов для каждого проекта:

	Проект	**Тестируемые классы**
	Система управления складом	Product, Warehouse, StockMovement, ProductRepository, WarehouseRepository
	Платформа аренды оборудования	Equipment, Rental, Customer, EquipmentRepository, RentalRepository
	Система управления парком автомобилей	Vehicle, Driver, Trip, VehicleRepository, DriverRepository
	Система дистанционного обучения	Course, Lesson, Student, Enrollment, CourseRepository, EnrollmentRepository
	Платформа онлайн-курсов	Course, Video, Quiz, Certificate, CourseRepository

	Проект	Тестируемые классы
	Система записи на кружки	Circle, Schedule, Attendance, CircleRepository, AttendanceRepository
	Система медицинских назначений	Patient, Appointment, Prescription, PatientRepository
	Платформа записи на процедуры	Patient, Procedure, Schedule, PatientRepository
	Система мониторинга здоровья	Patient, Measurement, Alert, PatientRepository
0	Система доставки еды	Restaurant, Order, Courier, OrderRepository
1	Платформа бронирования услуг	Salon, Master, Booking, SalonRepository
2	Система проката инвентаря	Item, Rental, ItemRepository
3	Система производственных заказов	Order, Product, Operation, OrderRepository
4	Платформа логистики	Shipment, Route, Vehicle, ShipmentRepository
	Система управления	Project, Task, Worker,

	Проект	Тестируемые классы
5	проектами	ProjectRepository

6. КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Что такое модульное тестирование и зачем оно нужно?
2. Какие аннотации JUnit 5 используются для тестирования?
3. Что такое мок-объект и для чего он нужен?
4. Как проверить, что метод выбросил исключение?
5. Как запустить все тесты в Maven-проекте?
6. В чём разница между @BeforeEach и @BeforeAll?
7. Как проверить, что метод был вызван определённое количество раз?
8. Как протестировать метод с параметрами?
9. Что такое покрытие кода (code coverage)?
10. Какой минимальный процент покрытия требуется в курсовом проекте?
11. Как протестировать метод с базой данных без реального подключения?
12. Как протестировать многопоточный код?
13. Как настроить параметризованные тесты с несколькими аргументами?
14. Как интегрировать тесты в CI/CD пайплайн?
15. Как измерить покрытие ветвей (branch coverage) в JaCoCo?

7. ТРЕБОВАНИЯ К ОТЧЕТУ

Структура отчета

1. Титульный лист
2. Содержание

3. Введение (цель работы, краткое описание проекта)

4. Настройка тестового окружения

   4.1. Зависимости в pom.xml

   4.2. Настройка JaCoCo

5. Тестирование Entity-слоя

   5.1. Тесты для Event (листинг + пояснения)

   5.2. Тесты для User (листинг + пояснения)

   5.3. Тесты для Registration (листинг + пояснения)

6. Тестирование Foundation-слоя

   6.1. Тесты для UserRepository (листинг + пояснения)

   6.2. Тесты для EventRepository (листинг + пояснения)

7. Отчёт о покрытии JaCoCo

   7.1. Скриншот отчёта

   7.2. Анализ покрытия

8. Заключение (выводы)

9. Список использованных источников

10. Приложения (полный код тестов)

8. КРИТЕРИИ ОЦЕНИВАНИЯ

Критерий	Мак с. балл	Описание
Настройка тестового окружения	2	Корректные зависимости в pom.xml, настройка JaCoCo
Тесты Entity-слоя	6	3+ сущностей, бизнес-методы, исключения
Тесты Foundation-слоя	6	3+ репозиториев, CRUD,

Критерий	Макс. балл	Описание
(моки)		специфические методы
Покрытие кода (JaCoCo)	3	Покрытие >40% инструкций
Оформление отчета	2	Соответствие требованиям
Ответы на контрольные вопросы	3	Понимание материала при защите
ИТОГО	22	

Лабораторная работа №9

Итоговая защита (промежуточная)

1. Цель и задачи работы

Цель: Систематизировать и представить результаты работы за 4 семестр по курсу «Программная инженерия», продемонстрировать работоспособность разработанного ядра системы, защитить промежуточные результаты перед преподавателем.

Задачи:

Подготовить презентацию, отражающую все этапы работы за семестр

Продемонстрировать работающее ядро системы (Entity + Foundation слои)

Показать результаты тестирования (JUnit + JaCoCo)

Ответить на вопросы по теоретическому материалу

Получить допуск к продолжению работы в 5 семестре

2. ТЕОРЕТИЧЕСКОЕ ОБОСНОВАНИЕ

2.1. Что такое промежуточная защита?

Промежуточная защита - это контрольная точка в конце 4 семестра, на которой студент демонстрирует результаты выполнения лабораторных работ ЛР1-ЛР8. Успешная защита является допуском к продолжению курса в 5 семестре.

2.2. Что проверяется на защите

Аспект	Что проверяется	Как проверяется
Бизнес-анализ	Корректность IDEF0, BUC, глоссария	Пояснения по диаграммам
Требования	Use Case диаграмма, Domain Model	Обоснование выбора сущностей
Архитектура	PCMEF, диаграмма пакетов, ADR	Объяснение слоёв и зависимостей
База данных	ER-диаграмма, DDL-скрипты	Обоснование нормализации
JPA-сущности	Корректность аннотаций, связей	Демонстрация кода
Репозитории	CRUD-операции, специфические методы	Демонстрация кода

Аспект	Что проверяется	Как проверяется
Тестирование	JUnit тесты, покрытие JaCoCo >40%	Демонстрация отчёта

2.3. Критерии успешной защиты

Критерий	Требование
Полнота выполнения	Выполнены все ЛР (ЛР1-ЛР8)
Работоспособность	Код компилируется, тесты проходят
Покрытие тестами	>40% (JaCoCo)
Архитектура	Соответствие PCMEF
Документация	Вся документация в Git

3. ПРИМЕР ВЫПОЛНЕНИЯ (ШАБЛОН ПРЕЗЕНТАЦИИ)

3.1. Структура презентации (10-12 слайдов)

Слайд 1: Титульный

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ

ФГАОУ ВО «СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

ПРОМЕЖУТОЧНАЯ ЗАЩИТА

по дисциплине «Программная инженерия»

Тема: [Название проекта]

Выполнил: студент группы [номер] [ФИО]

Преподаватель: [ФИО]

Ставрополь, 2026

Слайд 2: Краткое описание проекта

📌 НАЗВАНИЕ ПРОЕКТА: [Название]

🎯 ЦЕЛЬ: [Краткое описание, 1-2 предложения]

👥 ПОЛЬЗОВАТЕЛИ:

• [Роль 1] — [что делает]

• [Роль 2] — [что делает]

📊 ОСНОВНЫЕ ФУНКЦИИ:

• [Функция 1]

• [Функция 2]

• [Функция 3]

Слайд 3: Бизнес-модель (ЛР1)

🏢 БИЗНЕС-МОДЕЛЬ

Диаграмма бизнес-контекста (IDEF0 A-0):

[ВСТАВИТЬ ДИАГРАММУ]

Основная функция: [глагол + существительное]

Входы: [вход 1], [вход 2], [вход 3]

Выходы: [выход 1], [выход 2]

Ключевые бизнес-акторы:

• [Актор 1] — [роль]

• [Актор 2] — [роль]

Слайд 4: Системные требования (ЛР2)

📋 СИСТЕМНЫЕ ТРЕБОВАНИЯ

Диаграмма Use Case:

[ВСТАВИТЬ ДИАГРАММУ]

Основные акторы:

• Администратор — [возможности]

• Пользователь — [возможности]

Ключевые прецеденты:

• UC-001: [Название]

• UC-002: [Название]

• UC-003: [Название]

Domain Model (ключевые сущности):

• [Сущность 1] — [атрибуты]

• [Сущность 2] — [атрибуты]

• [Сущность 3] — [атрибуты]

Слайд 5: Архитектура PCMEF (ЛР3)

🏗 АРХИТЕКТУРА PCMEF

Диаграмма пакетов:

[ВСТАВИТЬ ДИАГРАММУ]

Траектория: [Desktop / Web / Mobile / Enterprise]

Распределение слоёв:
Presentation (P) — [технология]
Control (C) — [технология]
Mediator (M) — [технология]
Entity (E) — JPA
Foundation (F) — репозитории

Ключевые ADR:
• ADR-001: [Решение]
• ADR-002: [Решение]

Слайд 6: База данных (ЛР4-ЛР5)
🗄 БАЗА ДАННЫХ

ER-диаграмма:

[ВСТАВИТЬ ER-ДИАГРАММУ]

Таблицы:
• users — пользователи
• events — мероприятия

• registrations — регистрации
• payments — платежи
• categories — категории

Нормализация: 3НФ
СУБД: PostgreSQL

Слайд 7: JPA-сущности (ЛР6)
📦 JPA-СУЩНОСТИ

Пример сущности Event:

```
@Entity
@Table(name = "events")
public class Event {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(nullable = false)
    private String title;

    @ManyToOne
    @JoinColumn(name = "category_id")
    private Category category;

    @OneToMany(mappedBy = "event")
    private List<Registration> registrations;
```

```
    public boolean checkAvailability() { ... }
    public Registration registerUser(User user) { ... }
}
```

Связи:

• Event ↔ Category — @ManyToOne

• Event ↔ Registration — @OneToMany

Слайд 8: Репозитории (ЛР7)

📎 FOUNDATION-СЛОЙ (РЕПОЗИТОРИИ)

Пример репозитория:

```
public interface UserRepository {
    Optional<User> findById(Long id);
    Optional<User> findByEmail(String email);
    List<User> findByRole(String role);
    User save(User user);
    void delete(User user);
}

public class UserRepositoryImpl implements UserRepository {
    private final EntityManager entityManager;
    // реализация
}
```

Основные методы:

• save() — сохранение

• findById() — поиск по ID

• findByEmail() — поиск по email
• delete() — удаление

Слайд 9: Тестирование (ЛР8)

ТЕСТИРОВАНИЕ

JUnit 5 + Mockito

Пример теста:

```
@Test
void testRegisterUserWhenSeatsAvailable() {
    Registration registration = event.registerUser(user);
    assertNotNull(registration);
    assertEquals(event, registration.getEvent());
    assertEquals("PENDING", registration.getStatus());
}

@Test
void testRegisterUserWhenNoSeats() {
    // заполняем все места
    for (int i = 0; i < 10; i++) {
        event.registerUser(User.builder().id((long)i).build());
    }
    assertThrows(IllegalStateException.class, () -> {
        event.registerUser(user);
    });
}
```

Отчёт JaCoCo:

[ВСТАВИТЬ СКРИНШОТ ОТЧЁТА JaCoCo]

Покрытие инструкций: XX% (>40%)

Слайд 10: Демонстрация работы

📄 ДЕМОНСТРАЦИЯ РАБОТЫ

1. Сохранение пользователя в БД
2. Сохранение мероприятия
3. Регистрация пользователя на мероприятие
4. Выполнение тестов
5. Отчёт JaCoCo

[ДЕМОНСТРАЦИЯ КОДА И/ИЛИ КОНСОЛИ]

Слайд 11: Итоги и план на 5 семестр

📊 ИТОГИ 4 СЕМЕСТРА

✓ Выполненные работы:

• ЛР1: Бизнес-моделирование
• ЛР2: Системные требования
• ЛР3: Архитектура PCMEF
• ЛР4: ER-диаграмма
• ЛР5: DDL-скрипты
• ЛР6: JPA-сущности
• ЛР7: Репозитории
• ЛР8: Модульное тестирование (покрытие XX%)

📈 ПЛАН НА 5 СЕМЕСТР

• Реализация Mediator-слоя (бизнес-логика)

• Реализация Control-слоя (REST API)

• Создание Web-интерфейса (React)

• Интеграционное тестирование

• Контейнеризация (Docker)

[ССЫЛКА НА GIT-РЕПОЗИТОРИЙ]

Слайд 12: Спасибо за внимание

text

СПАСИБО ЗА ВНИМАНИЕ!

GitHub: https://github.com/username/project

4. МЕТОДИКА ВЫПОЛНЕНИЯ

Шаг 1: Подготовка презентации

Создать презентацию по шаблону (10-12 слайдов)

Вставить все диаграммы из ЛР1-ЛР8

Подготовить демонстрационные примеры кода

Сделать скриншот отчёта JaCoCo

Шаг 2: Проверка работоспособности

Убедиться, что проект компилируется без ошибок

Запустить все тесты (mvn test)

Проверить отчёт JaCoCo (mvn jacoco:report)

Убедиться, что покрытие >40%

Шаг 3: Подготовка к вопросам

Повторить теоретический материал по темам:

Бизнес-анализ:

Что такое IDEF0? Основные элементы (вход, выход, управление, механизм)

Чем бизнес-актор отличается от системного?

Что такое BUC-диаграмма?

Требования:

Что такое Use Case? Отношения include/extend

Что такое Domain Model? Чем отличается от бизнес-модели?

Что такое Value Object и Entity?

Архитектура:

Что такое PCMEF? Назначение каждого слоя

Правило зависимостей в PCMEF

Чем отличается классический PCMEF от адаптированного для Web/Mobile?

База данных:

Что такое нормализация? 1НФ, 2НФ, 3НФ

Типы связей в БД (1:1, 1:N, M:N)

Что такое первичный и внешний ключ?

JPA:

Что такое JPA и Hibernate?

Основные аннотации (@Entity, @Id, @GeneratedValue, @Column)

Типы связей (@OneToMany, @ManyToOne, @OneToOne)

EAGER vs LAZY загрузка

Репозитории:

Что такое паттерн Repository?

Чем отличается save() от update()?

Что такое JPQL?

Тестирование:

Что такое модульное тестирование?

Что такое мок-объект (Mockito)?

Что такое покрытие кода (JaCoCo)?

Шаг 4: Защита (15-20 минут)

Презентация (5-7 минут)

Демонстрация кода (3-5 минут)

Ответы на вопросы (5-10 минут)

5. ЧЕК-ЛИСТ ГОТОВНОСТИ К ЗАЩИТЕ

Документация в репозитории

Файл	Наличие	Комментарий
README.md	☐	Визитка проекта
docs/00-project-charter/	☐	ЛР1: бизнес-модель
docs/01-requirements/	☐	ЛР2: требования
docs/02-architecture/	☐	ЛР3: архитектура
docs/03-database/	☐	ЛР4-ЛР5: БД и DDL
docs/04-implementation/entity/	☐	ЛР6: JPA-сущности
docs/04-implementation/foundation/	☐	ЛР7: репозитории
docs/05-testing/jacoco-report/	☐	ЛР8: отчёт JaCoCo

Файл	Наличие	Комментарий
git-stats.png	☐	Статистика коммитов

Код

Компонент	Наличие	Комментарий
pom.xml	☐	Зависимости, плагины
src/main/java/.../entity/	☐	Все JPA-сущности
src/main/java/.../foundation/repositories/	☐	Все репозитории
src/main/java/.../foundation/config/	☐	JpaUtil
src/main/resources/META-INF/persistence.xml	☐	Настройки JPA
src/test/java/.../entity/	☐	Тесты сущностей
src/test/java/.../foundation/	☐	Тесты репозиториев

Технические требования

Требование	Статус	Комментарий
Проект компилируется	☐	mvn clean compile
Все тесты проходят	☐	mvn test
Покрытие >40%	☐	mvn jacoco:report
Git репозиторий	☐	Регулярные коммиты

6. КОНТРОЛЬНЫЕ ВОПРОСЫ (ДЛЯ ЗАЩИТЫ)

По бизнес-анализу (ЛР1)

Что такое IDEF0 и зачем он нужен?

Какие элементы входят в диаграмму IDEF0 A-0?

Чем бизнес-актор отличается от системного актора?

Что такое BUC-диаграмма?

По требованиям (ЛР2)

Что такое Use Case диаграмма? Что означают отношения include/extend?

Что такое Domain Model? Чем отличается от бизнес-модели?

Что такое Entity и Value Object в Domain Model?

По архитектуре (ЛР3)

Что означает аббревиатура PCMEF? Назначение каждого слоя?

Какое правило зависимостей в PCMEF?

Как адаптируется PCMEF для мобильной траектории?

По базе данных (ЛР4-ЛР5)

Что такое нормализация? Какие требования к 1НФ, 2НФ, 3НФ?

Как в реляционной БД реализуется связь 1:N?

Для чего нужны индексы?

По JPA (ЛР6)

Что такое JPA и Hibernate?

Какие аннотации используются для маппинга класса на таблицу?

Как реализовать связь @OneToMany?

По репозиториям (ЛР7)

Что такое паттерн Repository?

Чем отличается save() от update()?

Что такое JPQL? Чем отличается от SQL?

По тестированию (ЛР8)

Что такое модульное тестирование?

Что такое мок-объект и зачем он нужен?

Что такое покрытие кода? Какой минимальный процент требуется?

7. КРИТЕРИИ ОЦЕНИВАНИЯ

Критерий	**Макс. балл**	**Описание**
Полнота выполнения ЛР1-ЛР8	5	Все работы выполнены в полном объёме
Качество презентации	3	Структура, наглядность, читаемость
Демонстрация кода	4	Работоспособность, чистота кода

Критерий	Макс. балл	Описание
Тестирование (JaCoCo)	3	Покрытие >40%
Ответы на вопросы	5	Глубина понимания, аргументация
ИТОГО	20	

Штрафы:

Нарушение	Штраф
Отсутствие Git репозитория	-5 баллов
Отсутствие JaCoCo отчёта	-3 балла
Покрытие <40%	-2 балла
Не компилируется проект	недопуск

Лабораторная работа №10

Спецификация двух ключевых прецедентов (Use Cases)

1. Цель и задачи работы

Цель: Освоить методику детальной спецификации вариантов использования (Use Cases) для ключевых сценариев взаимодействия пользователей с программной системой на основе требований, выявленных в лабораторной работе №2 (4 семестр).

Задачи:

1. Выбрать два ключевых прецедента из Use Case диаграммы (ЛР2)
2. Детализировать каждый прецедент по шаблону, включая:
3. Краткое описание
4. Предусловия и постусловия
5. Основной поток (счастливый путь)
6. Альтернативные потоки (подпотоки)
7. Потоки исключений (обработка ошибок)
8. Определить приоритет, частоту использования и сложность каждого прецедента
9. Подготовить документ спецификации прецедентов

2. ТЕОРЕТИЧЕСКОЕ ОБОСНОВАНИЕ

2.1. Что такое Use Case Specification?

Use Case Specification (Спецификация прецедента) - это детализированное текстовое описание варианта использования, которое дополняет Use Case диаграмму. Если диаграмма показывает «кто что делает», то спецификация объясняет «как именно это происходит».

2.2. Зачем нужна спецификация прецедентов?

Причина	Объяснение
Уточнение требований	Диаграмма не показывает шаги и детали
Основа для тестирования	Каждый шаг основного потока — потенциальный тест-кейс
Коммуникация с заказчиком	Заказчик понимает текст

Причина	Объяснение
	лучше, чем диаграммы
Основа для проектирования	Разработчик понимает, что нужно реализовать
Документирование альтернатив	Учёт различных сценариев развития событий

2.3. Структура спецификации прецедента

Раздел	Описание	Обязательность
ID	Уникальный идентификатор (UC-001, UC-002...)	Да
Название	Краткое, глагол + существительное	Да
Акторы	Кто инициирует прецедент	Да
Краткое описание	Цель прецедента (1-2 предложения)	Да
Предусловия	Что должно быть истинно до начала	Да
Постусловия	Что будет	Да

Раздел	Описание	Обязательность
	истинно после успешного выполнения	
Основной поток	Нормальный сценарий (шаг 1, 2, 3...)	Да
Альтернативные потоки	Варианты развития событий	Да
Потоки исключений	Что происходит при ошибках	Да
Приоритет	High / Medium / Low	Рекомендуется
Частота использования	Редко / Иногда / Часто / Очень часто	Рекомендуется
Сложность	Низкая / Средняя / Высокая	Рекомендуется

2.4. Требования к хорошей спецификации

Требование	Описание	Пример
Однозначность	Каждый шаг понятен	✕ «Система обрабатывает данные»

Требование	Описание	Пример
	однозначно	✓ «Система проверяет формат email»
Полнота	Учтены все значимые шаги	Не должно быть пропусков в логике
Непротиворечивость	Шаги не противоречат друг другу	—
Тестируемость	Каждый шаг можно проверить	—
Уровень детализации	Не слишком мелко, не слишком крупно	Шаг — одно действие пользователя или системы

2.5. Пример нумерации шагов

4. ОСНОВНОЙ ПОТОК

1. Пользователь открывает страницу мероприятий
2. Система отображает список мероприятий
3. Пользователь выбирает мероприятие
4. Пользователь нажимает кнопку «Зарегистрироваться»
5. Система проверяет наличие свободных мест
6. Система создаёт регистрацию

7. Система отправляет email-подтверждение

8. Система отображает сообщение об успехе

2.6. Обозначение альтернативных потоков и исключений

Обозначение	Тип	Описание
A1	Альтернативный поток	Вариант развития, который не является ошибкой
A1.1, A1.2	—	Шаги альтернативного потока
E1	Исключение	Ошибочная ситуация, прерывающая основной поток
E1.1, E1.2	—	Шаги обработки исключения

3. ПРИМЕР ВЫПОЛНЕНИЯ (СКВОЗНОЙ ПРОЕКТ)

Проект: «Система управления мероприятиями» (Event Management System)

3.1. Исходная Use Case диаграмма (из ЛР2)

Акторы: Администратор, Участник, Бухгалтер

Прецеденты:

UC-001: Создать мероприятие

UC-002: Редактировать мероприятие

UC-003: Удалить мероприятие

UC-004: Зарегистрироваться на мероприятие

UC-005: Отменить регистрацию

UC-006: Оплатить участие

UC-007: Сформировать отчёт

3.2. Выбор прецедентов для детализации

Для спецификации выбраны два ключевых прецедента:

ID	Название	Причина выбора
UC-001	Создать мероприятие	Основной прецедент для администратора, включает валидацию данных
UC-004	Зарегистрироваться на мероприятие	Включает бизнес-правила (проверка мест), взаимодействие с внешней платежной системой

3.3. Спецификация прецедента UC-001: Создать мероприятие

ID: UC-001

Название: Создать мероприятие

Акторы: Администратор (инициатор), Система (исполнитель)

Уровень: Пользовательский

Приоритет: Высокий

Частота использования: 5-10 раз в неделю

Сложность: Средняя

1. Краткое описание

Администратор создаёт новое мероприятие в системе, заполняя информацию о нём: название, описание, даты проведения, максимальное количество участников, категорию. Мероприятие создаётся в статусе «Черновик» и может быть опубликовано позже.

2. Предусловия

Пользователь аутентифицирован в системе как Администратор

У пользователя есть права на создание мероприятий

Система находится в рабочем состоянии

База данных доступна

3. Постусловия

Новое мероприятие сохранено в базе данных

Мероприятию присвоен уникальный идентификатор (ID)

Мероприятие имеет статус «DRAFT» (Черновик)

В журнале аудита зафиксировано создание мероприятия

Администратору отправлено уведомление о создании

4. Основной поток (счастливый путь)

Администратор открывает страницу «Мои мероприятия»

Система отображает список существующих мероприятий и кнопку «Создать мероприятие»

Администратор нажимает кнопку «Создать мероприятие»

Система отображает форму создания мероприятия со следующими полями:

Название (обязательное, 3-200 символов)

Описание (необязательное, до 5000 символов)

Дата и время начала (обязательное, будущая дата)

Дата и время окончания (обязательное, после даты начала)

Максимальное количество участников (обязательное, 1-10000)

Категория (необязательное, выбор из списка)

Администратор заполняет обязательные поля и нажимает «Сохранить»

Система выполняет валидацию введённых данных:

Проверяет, что все обязательные поля заполнены

Проверяет, что дата начала в будущем

Проверяет, что дата окончания позже даты начала

Проверяет, что максимальное количество участников > 0

Система создаёт новое мероприятие со статусом «DRAFT»

Система генерирует уникальный ID мероприятия

Система сохраняет мероприятие в базе данных

Система отправляет email-подтверждение администратору

Система отображает сообщение «Мероприятие успешно создано»

Система перенаправляет администратора на страницу деталей созданного мероприятия

5. Альтернативные потоки

A1: Создание мероприятия на основе шаблона

Шаг	Действие
A1.1	На шаге 3 администратор выбирает «Создать из шаблона» вместо «Создать мероприятие»
A1.2	Система отображает список доступных шаблонов мероприятий
A1.3	Администратор выбирает шаблон

Шаг	Действие
A1.4	Система предзаполняет форму данными из выбранного шаблона
A1.5	Администратор редактирует поля по необходимости
A1.6	Продолжение с шага 5 основного потока

A2: Добавление сессий при создании

Шаг	Действие
A2.1	После шага 11 администратор выбирает «Добавить сессию»
A2.2	Выполняется прецедент UC-009 «Добавить сессию»
A2.3	После добавления сессии администратор возвращается к странице мероприятия

6. Потоки исключений

E1: Ошибка валидации данных

Шаг	Действие
E1.	На шаге 6 система обнаруживает ошибки валидации

Шаг	Действие
1	
E1.2	Система подсвечивает поля с ошибками
E1.3	Система отображает сообщения об ошибках рядом с соответствующими полями
E1.4	Администратор исправляет ошибки
E1.5	Продолжение с шага 5 основного потока

E2: Сбой базы данных

Шаг	Действие
E2.1	На шаге 9 происходит ошибка при сохранении в базу данных
E2.2	Система откатывает транзакцию
E2.3	Система сохраняет введённые данные в локальное хранилище браузера
E2.	Система отображает сообщение «Ошибка сохранения.

Шаг	Действие
4	Попробуйте позже.»
E2.5	Прецедент завершается неудачей

E3: Превышение лимита мероприятий (для бесплатного тарифа)

Шаг	Действие
E3.1	На шаге 6 система определяет, что у администратора превышен лимит активных мероприятий
E3.2	Система отображает сообщение «Достигнут лимит активных мероприятий. Обновите тариф.»
E3.3	Система предлагает перейти на страницу управления подпиской
E3.4	Прецедент завершается неудачей

E4: Потеря соединения с интернетом

Шаг	Действие
E4.1	На любом шаге теряется интернет-соединение

Шаг	Действие
E4.2	Система обнаруживает потерю соединения
E4.3	Система сохраняет введённые данные в локальное хранилище
E4.4	Система отображает сообщение «Нет соединения с сервером. Данные сохранены локально.»
E4.5	При восстановлении соединения система предлагает восстановить сохранённые данные

3.4. Спецификация прецедента UC-004: Зарегистрироваться на мероприятие

ID: UC-004

Название: Зарегистрироваться на мероприятие

Акторы: Участник (инициатор), Система (исполнитель), Платёжная система (внешняя)

Уровень: Пользовательский

Приоритет: Высокий

Частота использования: 100-500 раз в день

Сложность: Высокая

1. Краткое описание

Пользователь регистрируется на опубликованное мероприятие. Если мероприятие платное, регистрация включает этап оплаты. После успешной регистрации пользователь получает подтверждение по email.

2. Предусловия

Пользователь аутентифицирован в системе (или регистрация доступна для гостей)

Мероприятие существует и имеет статус «PUBLISHED» (Опубликовано)

Дата начала мероприятия ещё не наступила

Есть доступные места для регистрации

Для платных мероприятий: платёжная система доступна

3. Постусловия

Создана регистрация со статусом «PENDING» или «CONFIRMED»

Количество доступных мест уменьшено на 1

Пользователь получил email-подтверждение регистрации

Для платных мероприятий: создана запись о платеже

Мероприятие добавлено в календарь пользователя (опционально)

4. Основной поток (счастливый путь)

Пользователь открывает страницу «Список мероприятий»

Система отображает список опубликованных мероприятий (с фильтрацией и пагинацией)

Пользователь выбирает мероприятие и переходит на страницу его деталей

Система отображает подробную информацию о мероприятии:

Название, описание, даты проведения

Местоположение (адрес)

Количество свободных мест

Стоимость участия (если мероприятие платное)

Кнопку «Зарегистрироваться»

Пользователь нажимает кнопку «Зарегистрироваться»

Если пользователь не аутентифицирован:

Система перенаправляет на страницу входа/регистрации

После успешной аутентификации возвращает на страницу мероприятия

Система проверяет возможность регистрации:

Статус мероприятия = PUBLISHED

Текущая дата < дата начала мероприятия

Количество регистраций < максимальное количество участников

Если мероприятие бесплатное:

Система создаёт регистрацию со статусом «CONFIRMED»

Переход к шагу 12

Если мероприятие платное:

Система отображает форму оплаты (сумма, способ оплаты)

Пользователь выбирает способ оплаты (карта, электронные деньги)

Система перенаправляет на страницу платёжного шлюза

Пользователь вводит платёжные данные

Платёжная система обрабатывает платеж

Система получает подтверждение от платёжной системы

Система создаёт запись о платеже

Система создаёт регистрацию со статусом «CONFIRMED»

Система сохраняет регистрацию в базе данных

Система уменьшает счётчик доступных мест

Система отправляет email с подтверждением регистрации (и чеком, если была оплата)

Система добавляет мероприятие в календарь пользователя (Google Calendar / Яндекс.Календарь)

Система отображает сообщение «Вы успешно зарегистрированы на мероприятие»

Система перенаправляет на страницу «Мои регистрации»

5. Альтернативные потоки

A1: Регистрация нескольких участников одним пользователем

Шаг	Действие
A1.1	На шаге 5 пользователь выбирает «Зарегистрировать нескольких участников»
A1.2	Система отображает форму для добавления дополнительных участников
A1.3	Пользователь добавляет данные для каждого участника (ФИО, email, телефон)
A1.4	Шаги A1.2-A1.3 повторяются для каждого дополнительного участника
A1.5	При оплате общая сумма = стоимость × количество участников
A1.6	Продолжение с шага 6 основного потока

A2: Добавление промокода (если мероприятие платное)

Шаг	Действие
A2.1	На шаге 9 пользователь вводит промокод в специальное поле
A2.2	Система проверяет промокод (активен, не истёк, соответствует мероприятию)

Шаг	Действие
A2.3	Система пересчитывает сумму со скидкой
A2.4	Пользователь подтверждает оплату со скидкой
A2.5	Продолжение с шага 9 основного потока

A3: Регистрация через социальные сети

Шаг	Действие
A3.1	На шаге 6 (аутентификация) пользователь выбирает «Войти через Google/Facebook/VK»
A3.2	Система перенаправляет на страницу OAuth-авторизации выбранной социальной сети
A3.3	Пользователь авторизуется в социальной сети
A3.4	Система получает базовые данные профиля (email, имя)
A3.5	Система создаёт учётную запись (если впервые) и выполняет вход
A3.6	Продолжение с шага 5 основного потока

6. Потоки исключений

E1: Нет доступных мест

Шаг	Действие
E1.1	На шаге 7 система обнаруживает, что свободных мест нет
E1.2	Система отображает сообщение «Свободных мест нет»
E1.3	Система предлагает встать в лист ожидания
E1.4	Если пользователь согласен: создаётся регистрация со статусом «WAITLISTED»
E1.5	Если пользователь отказывается: прецедент завершается
E1.6	При освобождении места пользователь из листа ожидания автоматически регистрируется и получает уведомление

E2: Истекло время сессии при оплате

Шаг	Действие
E2.1	На шаге 9 пользователь слишком долго заполняет платёжную форму

Шаг	Действие
E2.2	Время сессии истекает
E2.3	Система отображает сообщение «Время сессии истекло. Повторите попытку.»
E2.4	Система возвращает пользователя на шаг 4
E2.5	Прецедент завершается неудачей

E3: Недостаточно средств на счёте (для платных мероприятий)

Шаг	Действие
E3.1	На шаге 9 платёжная система отклоняет платёж из-за недостатка средств
E3.2	Система отображает сообщение «Недостаточно средств»
E3.3	Система предлагает выбрать другой способ оплаты
E3.4	Пользователь выбирает другой способ оплаты
E3.5	Продолжение с шага 9 основного потока

E4: Пользователь уже зарегистрирован на это мероприятие

Шаг	Действие
E4.1	На шаге 7 система обнаруживает, что пользователь уже зарегистрирован
E4.2	Система отображает сообщение «Вы уже зарегистрированы на это мероприятие»
E4.3	Система перенаправляет на страницу «Мои регистрации»
E4.4	Прецедент завершается

E5: Платёжная система недоступна

Шаг	Действие
E5.1	На шаге 9 система не может установить соединение с платёжной системой
E5.2	Система отображает сообщение «Платёжная система временно недоступна. Попробуйте позже.»
E5.3	Регистрация сохраняется со статусом «PENDING»
E5.4	Система отправляет напоминание об оплате через 1 час

Шаг	Действие
E5.5	Прецедент завершается неудачей

4. МЕТОДИКА ВЫПОЛНЕНИЯ (ПОШАГОВАЯ ИНСТРУКЦИЯ)

Шаг 1: Выбор прецедентов для детализации

Откройте Use Case диаграмму из лабораторной работы №2

Выберите два ключевых прецедента для детализации

Критерии выбора:

Прецедент с бизнес-логикой (не просто CRUD)

Прецедент, затрагивающий несколько слоёв системы

Прецедент с альтернативными потоками и исключениями

Таблица для заполнения:

ID	Название	Причина выбора
UC-...	...	...
UC-...	...	...

Шаг 2: Заполнение спецификации прецедента

Для каждого выбранного прецедента заполните:

ID, название, акторы

Краткое описание (1-2 предложения)

Предусловия (минимум 2)

Постусловия (минимум 2)

Основной поток (8-15 шагов)

Альтернативные потоки (минимум 2)

Потоки исключений (минимум 3)

Приоритет, частота, сложность

Шаг 3: Оформление документа

Объедините спецификации двух прецедентов в один документ

Добавьте титульный лист

Добавьте таблицу выбора прецедентов

Проверьте нумерацию шагов и обозначения

5. ВАРИАНТЫ ДЛЯ ИНДИВИДУАЛЬНОГО ВЫПОЛНЕНИЯ

Варианты соответствуют проектам из лабораторной работы №2.

	Проект	Рекомендуемые прецеденты для детализации
	Система управления складом	Приёмка товара, Отгрузка товара, Списание товара
	Платформа аренды оборудования	Бронирование оборудования, Возврат оборудования
	Система управления парком автомобилей	Назначение водителя, Завершение поездки
	Система дистанционного обучения	Запись на курс, Прохождение теста, Получение сертификата

	Проект	Рекомендуемые прецеденты для детализации
	Платформа для проведения онлайн-курсов	Создание курса, Проверка задания
	Система записи на кружки и секции	Запись ребёнка, Отметка посещения
	Система управления медицинскими назначениями	Создание назначения, Выполнение назначения
	Платформа записи на диагностические процедуры	Запись на процедуру, Получение результатов
	Система мониторинга здоровья пациентов	Добавление показателя, Генерация алерта
0	Система управления доставкой еды	Оформление заказа, Назначение курьера
1	Платформа бронирования услуг красоты	Выбор мастера, Бронирование услуги
2	Система проката спортивного инвентаря	Оформление проката, Возврат с штрафом

	Проект	Рекомендуемые прецеденты для детализации
3	Система управления производственными заказами	Создание заказа, Завершение операции
4	Платформа логистики и отслеживания грузов	Создание отправления, Обновление статуса
5	Система управления проектами в строительстве	Создание задачи, Отчёт о выполнении

6. КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Что такое Use Case Specification и зачем она нужна?
2. Какие разделы обязательно должны быть в спецификации прецедента?
3. Чем предусловия отличаются от постусловий?
4. Что такое «основной поток» (happy path)?
5. В чём разница между альтернативным потоком и потоком исключений?
6. Как правильно нумеровать шаги в альтернативных потоках и исключениях?
7. Какие критерии выбора прецедентов для детализации?
8. Как определить, что спецификация прецедента достаточно детализирована?
9. Какие ошибки часто встречаются при написании спецификаций?
10. Как связана спецификация прецедентов с тест-кейсами?

11. Как спецификация прецедентов используется при проектировании архитектуры?
12. Как учесть в спецификации бизнес-правила?
13. Что такое «расширяющие точки» (extension points) в Use Case?
14. Как документировать прецеденты с длительными процессами (асинхронными)?
15. Как спецификация прецедентов связана с user stories в Agile?

7. ТРЕБОВАНИЯ К ОТЧЕТУ

Структура отчета

1. Титульный лист
2. Содержание
3. Введение (цель работы, краткое описание проекта)
4. Выбор прецедентов для детализации (таблица с обоснованием)
5. Спецификация прецедента 1
   5.1. ID и название
   5.2. Акторы, приоритет, частота, сложность
   5.3. Краткое описание
   5.4. Предусловия и постусловия
   5.5. Основной поток
   5.6. Альтернативные потоки
   5.7. Потоки исключений
6. Спецификация прецедента 2
   (аналогичная структура)
7. Заключение (выводы, связь с последующими работами)
8. Список использованных источников
9. Приложения (при необходимости)

8. КРИТЕРИИ ОЦЕНИВАНИЯ

Критерий	Макс. балл	Описание
Выбор прецедентов	2	Обоснованный выбор двух ключевых прецедентов
Спецификация прецедента 1	8	Полнота, корректность, детализация
Спецификация прецедента 2	8	Полнота, корректность, детализация
Альтернативные потоки	2	Минимум 2 на каждый прецедент
Потоки исключений	2	Минимум 3 на каждый прецедент
Оформление отчета	2	Соответствие требованиям
Ответы на контрольные вопросы	2	Понимание материала
ИТОГО	26	

Лабораторная работа №11

Диаграммы последовательности для 3 сценариев

1. Цель и задачи работы

Цель: Освоить методику детального проектирования поведения программной системы путем построения диаграмм последовательности (sequence diagrams) для ключевых сценариев использования на основе спецификаций прецедентов, разработанных в лабораторной работе №10.

Задачи:

1. Выбрать 3 ключевых сценария из спецификаций прецедентов (ЛР10)
2. Построить диаграммы последовательности, показывающие взаимодействие объектов между слоями PCMEF
3. Учесть альтернативные потоки и исключения на диаграммах
4. Добавить пояснения к каждой диаграмме
5. Подготовить отчёт с тремя диаграммами и их анализом

2. ТЕОРЕТИЧЕСКОЕ ОБОСНОВАНИЕ

2.1. Место диаграмм последовательности в жизненном цикле

Диаграммы последовательности (Sequence Diagrams) — это мост между требованиями (спецификациями прецедентов) и реализацией (кодом).

Спецификация прецедента (ЛР10) ↓
Диаграмма последовательности (ЛР11) ← **МЫ ЗДЕСЬ** ↓
Проектирование классов (ЛР13) ↓
Реализация (Mediator, Control)

2.2. Что такое диаграмма последовательности?

Диаграмма последовательности (Sequence Diagram) - это UML-диаграмма, показывающая взаимодействие объектов во времени. Она

является основным инструментом детального проектирования поведения системы.

2.3. Основные элементы диаграммы последовательности

Элемент	Обозначение	Описание
Актор	actor "Пользователь" as User	Внешний пользователь системы
Линия жизни (Lifeline)	participant "UI" as UI	Представляет отдельный объект в процессе
Синхронное сообщение	User -> UI : действие	Вызов метода, ожидание ответа
Асинхронное сообщение	UI ->> API : запрос	Вызов без ожидания ответа
Возврат (Reply)	API --> UI : результат	Возврат результата (опционально)
Фрагмент alt	alt условие	Условное выполнение (if-else)

Элемент	Обозначение	Описание
Фрагмент opt	opt условие	Опциональное выполнение (if)
Фрагмент loop	loop [N раз]	Циклическое выполнение
Создание объекта	create NewObject	Создание нового экземпляра
Уничтожение объекта	destroy	Завершение жизни объекта

2.4. Связь диаграмм последовательности с архитектурой PCMEF

Диаграммы последовательности должны отражать взаимодействие между слоями PCMEF строго сверху вниз:

Presentation (UI) → Control (Controllers) → Mediator (Services) → Entity (Сущности) → Foundation (Репозитории) → Database

Правила взаимодействия в PCMEF:

Правило	Описание
1	Presentation вызывает методы Control

Правило	Описание
2	Control вызывает методы Mediator
3	Mediator вызывает методы Foundation и модифицирует Entity
4	Foundation обращается к БД
5	Ответ возвращается по цепочке в обратном порядке

2.5. От спецификации прецедента к диаграмме последовательности

Элемент спецификации	**Элемент диаграммы**
Шаг основного потока	Последовательность сообщений
Условие "если"	Фрагмент alt
Действие "если возможно"	Фрагмент opt
Повторяющееся действие	Фрагмент loop
Участник (актор, система, компонент)	Линия жизни (lifeline)
Обработка ошибки	Фрагмент alt с условием ошибки

3. ПРИМЕР ВЫПОЛНЕНИЯ (СКВОЗНОЙ ПРОЕКТ)

Проект: «Система управления мероприятиями» (Event Management System)

3.1. Выбор сценариев для детализации

На основе спецификаций прецедентов из ЛР10 выбраны три сценария:

	Сценарий	Источник	Обоснование
	Успешная регистрация на мероприятие (бесплатное)	UC-004 основной поток	Показывает стандартное взаимодействие
	Ошибка при регистрации (нет мест)	UC-004 исключение E1	Показывает обработку ошибок
	Успешное создание мероприятия	UC-001 основной поток	Показывает взаимодействие администратора

3.2. Диаграмма последовательности 1: Успешная регистрация на бесплатное мероприятие

Сценарий: Пользователь успешно регистрируется на бесплатное мероприятие.

@startuml

title Диаграмма последовательности: Успешная регистрация на бесплатное мероприятие

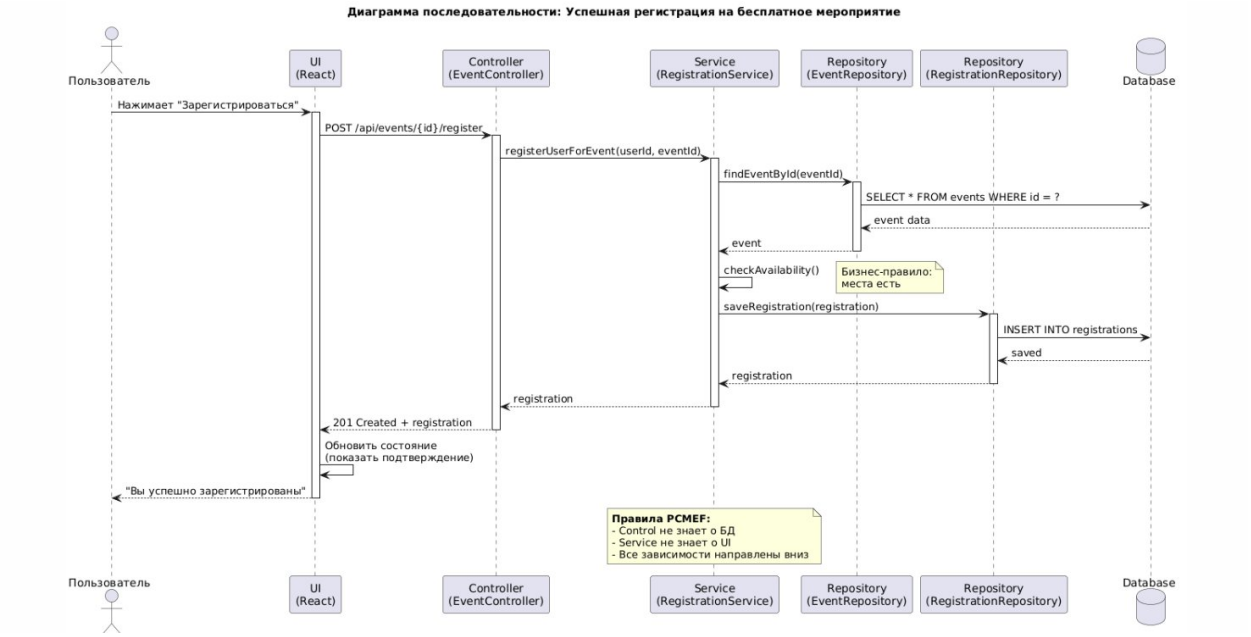
actor "Пользователь" as User

participant "UI\n(React)" as UI

participant "Controller\n(EventController)" as Controller

```
participant "Service\n(RegistrationService)" as Service
participant "Repository\n(EventRepository)" as EventRepo
participant "Repository\n(RegistrationRepository)" as RegRepo
database "Database" as DB

User -> UI: Нажимает "Зарегистрироваться"
activate UI

UI -> Controller: POST /api/events/{id}/register
activate Controller

Controller -> Service: registerUserForEvent(userId, eventId)
activate Service

Service -> EventRepo: findEventById(eventId)
activate EventRepo
EventRepo -> DB: SELECT * FROM events WHERE id = ?
DB --> EventRepo: event data
EventRepo --> Service: event
deactivate EventRepo

Service -> Service: checkAvailability()
note right: Бизнес-правило:\nместа есть

Service -> RegRepo: saveRegistration(registration)
activate RegRepo
RegRepo -> DB: INSERT INTO registrations
DB --> RegRepo: saved
RegRepo --> Service: registration
```

```
deactivate RegRepo

Service --> Controller: registration
deactivate Service

Controller --> UI: 201 Created + registration
deactivate Controller

UI -> UI: Обновить состояние\n(показать подтверждение)
UI --> User: "Вы успешно зарегистрированы"

deactivate UI

note over Service
  **Правила PCMEF:**
  - Control не знает о БД
  - Service не знает о UI
  - Все зависимости направлены вниз
end note

@enduml
```



Пояснения к диаграмме:

Актор «Пользователь» инициирует действие через UI

UI (React) отправляет HTTP-запрос к контроллеру

Контроллер делегирует бизнес-логику сервису

Сервис:

Запрашивает мероприятие из репозитория

Проверяет наличие мест (бизнес-правило в сущности)

Сохраняет регистрацию

Ответ возвращается по цепочке пользователю

3.3. Диаграмма последовательности 2: Ошибка при регистрации (нет свободных мест)

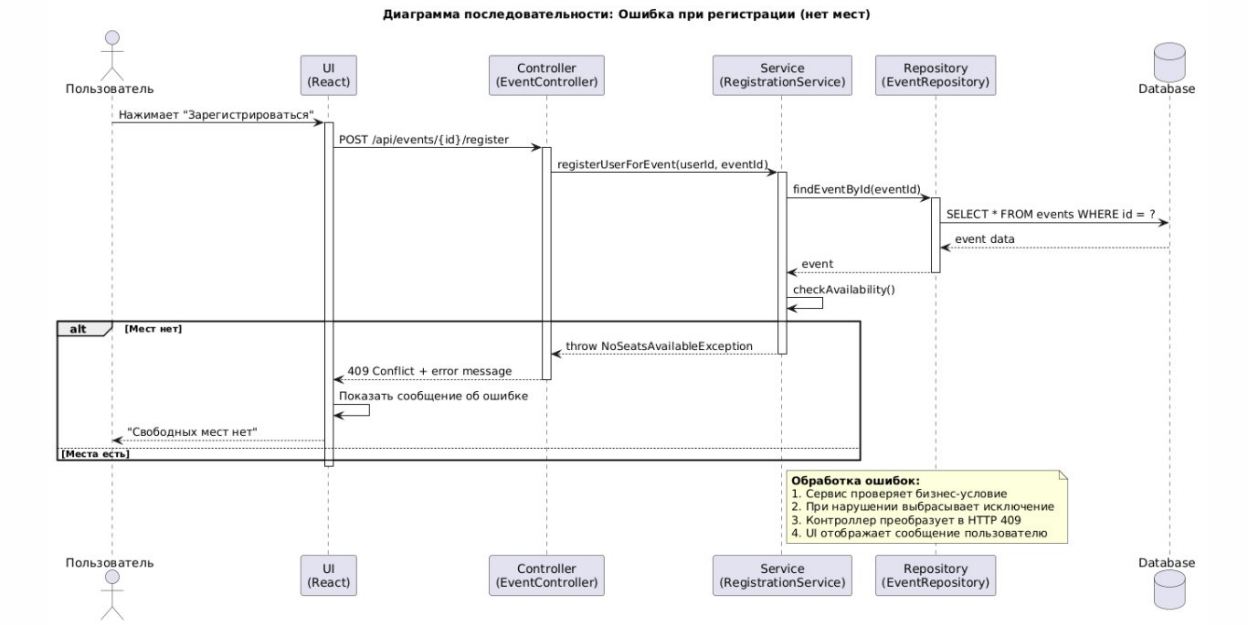
Сценарий: Пользователь пытается зарегистрироваться на мероприятие, но свободных мест нет.

@startuml

title Диаграмма последовательности: Ошибка при регистрации (нет мест)

```
actor "Пользователь" as User
participant "UI\n(React)" as UI
participant "Controller\n(EventController)" as Controller
participant "Service\n(RegistrationService)" as Service
participant "Repository\n(EventRepository)" as EventRepo
database "Database" as DB

User -> UI: Нажимает "Зарегистрироваться"
activate UI

UI -> Controller: POST /api/events/{id}/register
activate Controller

Controller -> Service: registerUserForEvent(userId, eventId)
activate Service

Service -> EventRepo: findEventById(eventId)
activate EventRepo
EventRepo -> DB: SELECT * FROM events WHERE id = ?
DB --> EventRepo: event data
EventRepo --> Service: event
deactivate EventRepo

Service -> Service: checkAvailability()

alt Мест нет
    Service --> Controller: throw NoSeatsAvailableException
    deactivate Service
```

```
    Controller --> UI: 409 Conflict + error message
    deactivate Controller

    UI -> UI: Показать сообщение об ошибке
    UI --> User: "Свободных мест нет"

else Места есть
    ' Этот вариант не выполняется
end

deactivate UI

note right of Service
  **Обработка ошибок:**
  1. Сервис проверяет бизнес-условие
  2. При нарушении выбрасывает исключение
  3. Контроллер преобразует в HTTP 409
  4. UI отображает сообщение пользователю
end note

@enduml
```



Пояснения к диаграмме:

1. Фрагмент alt показывает два варианта развития событий
2. При отсутствии мест сервис выбрасывает исключение
3. Контроллер преобразует исключение в HTTP-статус 409 (Conflict)
4. UI отображает сообщение об ошибке

Важно: Ответ не возвращается, прецедент завершается неудачей

3.4. Диаграмма последовательности 3: Успешное создание мероприятия администратором

Сценарий: Администратор создаёт новое мероприятие.

@startuml

title Диаграмма последовательности: Создание мероприятия администратором

actor "Администратор" as Admin

participant "UI\n(React)" as UI

participant "Controller\n(EventController)" as Controller

participant "Service\n(EventService)" as Service

participant "Repository\n(EventRepository)" as EventRepo

```
participant "Repository\n(CategoryRepository)" as CatRepo
participant "Repository\n(UserRepository)" as UserRepo
database "Database" as DB

Admin -> UI: Заполняет форму и нажимает "Сохранить"
activate UI

UI -> Controller: POST /api/events
activate Controller

Controller -> Controller: validateEventData()
note right: Валидация на уровне Control

alt Ошибка валидации
    Controller --> UI: 400 Bad Request + errors
    deactivate Controller
    UI --> Admin: "Проверьте введённые данные"
    deactivate UI
else Данные корректны
    Controller -> Service: createEvent(eventData, userId)
    activate Service

    Service -> UserRepo: findById(createdBy)
    activate UserRepo
    UserRepo -> DB: SELECT * FROM users WHERE id = ?
    DB --> UserRepo: user
    UserRepo --> Service: creator
    deactivate UserRepo
```

```
    opt Если указана категория
        Service -> CatRepo: findById(categoryId)
        activate CatRepo
        CatRepo -> DB: SELECT * FROM categories WHERE id = ?
        DB --> CatRepo: category
        CatRepo --> Service: category
        deactivate CatRepo
    end

    Service -> Service: create Event entity
    note right: Бизнес-правила:\n- статус "DRAFT"\n- валидация дат

    Service -> EventRepo: save(event)
    activate EventRepo
    EventRepo -> DB: INSERT INTO events
    DB --> EventRepo: saved event
    EventRepo --> Service: savedEvent
    deactivate EventRepo

    Service --> Controller: EventDto
    deactivate Service

    Controller --> UI: 201 Created + event
    deactivate Controller

    UI -> UI: Обновить список мероприятий
    UI --> Admin: "Мероприятие создано"
    deactivate UI
end
```

note over Service

Распределение ответственности:

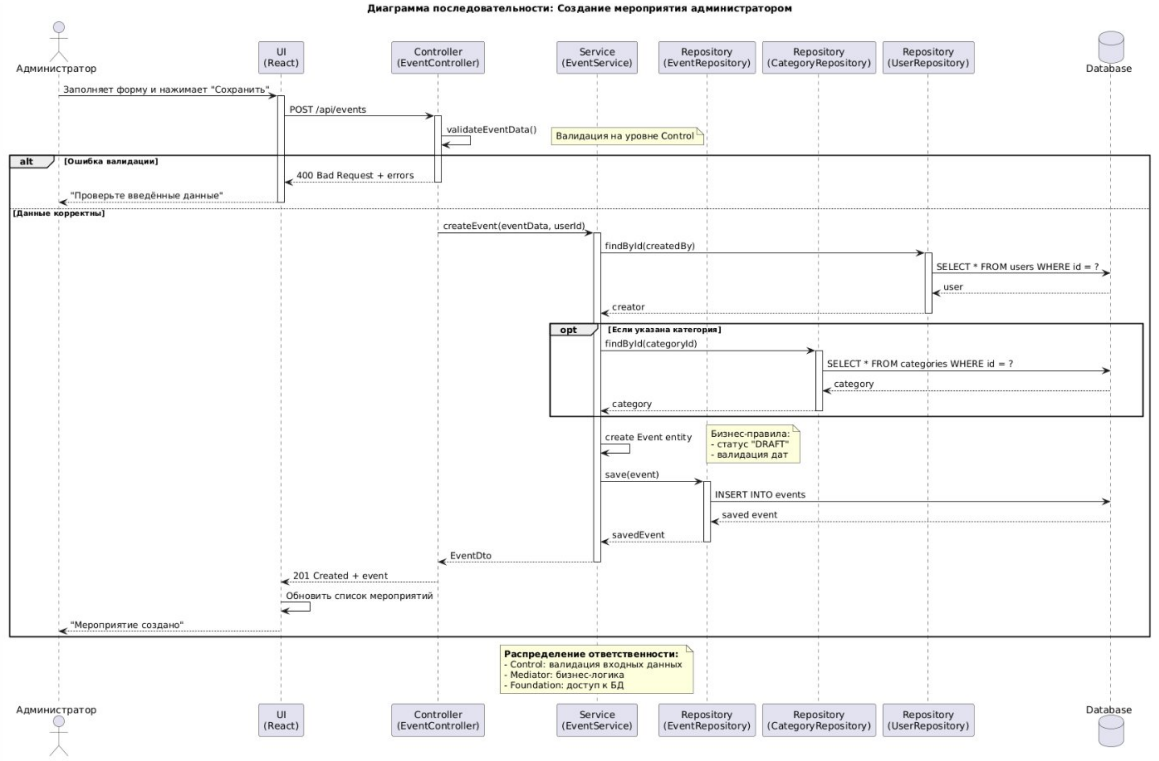
- Control: валидация входных данных

- Mediator: бизнес-логика

- Foundation: доступ к БД

end note

@enduml



Пояснения к диаграмме:

Администратор заполняет форму и отправляет данные

Контроллер сначала выполняет валидацию на уровне DTO

При ошибке валидации сразу возвращается 400 Bad Request

При успешной валидации вызывается сервис

Сервис:

Проверяет существование пользователя-создателя

При наличии категории — проверяет её существование

Создаёт сущность Event (статус "DRAFT")

Сохраняет через репозиторий

Фрагмент opt показывает опциональное действие (поиск категории, если она указана)

4. МЕТОДИКА ВЫПОЛНЕНИЯ (ПОШАГОВАЯ ИНСТРУКЦИЯ)

Шаг 1: Выбор сценариев для визуализации

Откройте спецификации прецедентов из лабораторной работы №10

Выберите 3 сценария для построения диаграмм

Рекомендуемый набор:

Один успешный сценарий (happy path)

Один сценарий с альтернативным потоком

Один сценарий с обработкой исключения

Таблица для заполнения:

	Сценарий	Источник (UC)	Тип сценария	Почему выбран
	...	UC-...	Основной поток	...
	...	UC-...	Альтернативный поток	...
	...	UC-...	Исключение	...

Шаг 2: Построение диаграмм последовательности

Для каждого выбранного сценария:

Определите участников (акторов и компоненты)

Расположите участников в порядке PCMEF сверху вниз

Нанесите линию жизни для каждого участника

Постройте последовательность сообщений из основного потока

Добавьте alt для условных ветвлений

Добавьте opt для опциональных действий

Добавьте loop для циклов

Покажите обработку исключений

Добавьте пояснения (ноты) к ключевым моментам

Шаг 3: Добавление пояснений

Для каждой диаграммы:

Объясните, какой сценарий показан

Укажите, какие слои PCMEF участвуют

Поясните ключевые бизнес-правила

Объясните обработку ошибок

Шаг 4: Оформление отчёта

Вставьте все три диаграммы в отчёт

Добавьте пояснения под каждой диаграммой

Проверьте читаемость (не перегружены ли диаграммы)

5. ВАРИАНТЫ ДЛЯ ИНДИВИДУАЛЬНОГО ВЫПОЛНЕНИЯ

Варианты соответствуют проектам из лабораторной работы №10.

№	**Проект**	**Рекомендуемые сценарии для диаграмм**
1	Система управления складом	Приёмка товара, Списание при браке, Отгрузка (недостаточно товара)
2	Платформа аренды оборудования	Бронирование, Возврат с повреждением, Продление аренды
3	Система	Назначение водителя, Завершение с

№	Проект	Рекомендуемые сценарии для диаграмм
	управления парком автомобилей	нарушением, Ремонт
4	Система дистанционного обучения	Запись на курс, Прохождение теста (не сдан), Получение сертификата
5	Платформа онлайн-курсов	Создание курса, Проверка задания (отклонено), Публикация
6	Система записи на кружки	Запись ребёнка, Отмена записи, Ошибка при оплате
7	Система медицинских назначений	Создание назначения, Отмена назначения, Просроченное назначение
8	Платформа записи на процедуры	Запись на процедуру, Отмена записи, Перенос записи
9	Система мониторинга здоровья	Добавление показателя, Превышение нормы, Генерация отчёта
10	Система доставки еды	Оформление заказа, Отмена заказа, Нет курьеров
1	Платформа	Бронирование, Отмена брони,

№	Проект	Рекомендуемые сценарии для диаграмм
1	бронирования услуг	Занятое время
2	1 Система проката инвентаря	Оформление проката, Возврат с штрафом, Продление
3	1 Система производственных заказов	Создание заказа, Завершение операции, Брак
4	1 Платформа логистики	Создание отправления, Задержка доставки, Потеря груза
5	1 Система управления проектами	Создание задачи, Отчёт о выполнении, Срыв срока

6. КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Что такое диаграмма последовательности и для чего она используется?
2. Назовите основные элементы диаграммы последовательности.
3. Как на диаграмме последовательности показать условное выполнение?
4. Как показать цикл на диаграмме последовательности?
5. В чем разница между синхронным и асинхронным сообщением?
6. Как диаграммы последовательности связаны с архитектурой PCMEF?

7. Почему важно соблюдать направление вызовов сверху вниз (P → C → M → E → F)?
8. Как показать обработку исключения на диаграмме последовательности?
9. Как на диаграмме последовательности показать создание нового объекта?
10. Как обеспечить соответствие между диаграммами последовательности и спецификациями прецедентов?
11. Какие проблемы могут возникнуть, если не построить диаграммы последовательности до реализации?
12. Как проверить, что диаграмма последовательности не нарушает архитектурные принципы?
13. Как показать на диаграмме последовательности взаимодействие с внешним сервисом (например, платёжным шлюзом)?

7. ТРЕБОВАНИЯ К ОТЧЕТУ

Структура отчета

1. Титульный лист
2. Содержание
3. Введение (цель работы, краткое описание проекта)
4. Выбор сценариев для визуализации (таблица с обоснованием)
5. Диаграмма последовательности 1
   5.1. Графическое представление (вставка изображения)
   5.2. Пояснения к диаграмме
6. Диаграмма последовательности 2
   6.1. Графическое представление (вставка изображения)
   6.2. Пояснения к диаграмме
7. Диаграмма последовательности 3

7.1. Графическое представление (вставка изображения)

7.2. Пояснения к диаграмме

8. Анализ соответствия архитектуре PCMEF

9. Заключение (выводы, связь с последующими работами)

10. Список использованных источников

11. Приложения (исходный код PlantUML)

8. КРИТЕРИИ ОЦЕНИВАНИЯ

Критерий	**Мак с. балл**	**Описание**
Выбор сценариев	2	3 сценария, обоснованный выбор
Диаграмма 1	6	Корректность, полнота, соответствие PCMEF
Диаграмма 2	6	Корректность, полнота, соответствие PCMEF
Диаграмма 3	6	Корректность, полнота, соответствие PCMEF
Пояснения	3	Ясность, полнота, знание терминологии
Анализ PCMEF	2	Выводы о соответствии архитектуре
Оформление	2	Соответствие требованиям

Критерий	Макс. балл	Описание
отчета		
Ответы на контрольные вопросы	3	Понимание материала
ИТОГО	30	

9. ИНСТРУМЕНТЫ ДЛЯ ВЫПОЛНЕНИЯ

Инструмент	Ссылка	Особенности
PlantUML Online	https://www.plantuml.com/plantuml/uml/	Бесплатно, текст → диаграмма
PlantUML for VS Code	marketplace.visualstudio.com	Локальная работа
Draw.io	https://app.diagrams.net	GUI-редактор

Лабораторная работа №12

Применение паттернов GoF (Strategy, Observer)

1. Цель и задачи работы

Цель: Освоить применение поведенческих паттернов проектирования GoF (Gang of Four) — Strategy и Observer — для решения типовых задач в архитектуре PCMEF.

Задачи:

1. Изучить теоретические основы паттернов Strategy и Observer
2. Выявить в проекте места, где целесообразно применить эти паттерны
3. Реализовать паттерн Strategy для вариативных алгоритмов (расчёт стоимости, скидки, валидация)
4. Реализовать паттерн Observer для уведомлений об изменениях (email, push, логирование)
5. Интегрировать паттерны в существующую архитектуру PCMEF
6. Написать модульные тесты для проверки реализованных паттернов

2. ТЕОРЕТИЧЕСКОЕ ОБОСНОВАНИЕ

2.1. Что такое паттерны проектирования GoF?

Паттерны проектирования (Design Patterns) - это типовые решения часто встречающихся проблем при проектировании объектно-ориентированного программного обеспечения.

Классификация паттернов GoF:

Группа	Назначение	Паттерны
Порождающие	Создание объектов	Singleton, Factory Method, Abstract Factory, Builder, Prototype
Структурные	Компоновка классов и объектов	Adapter, Bridge, Composite, Decorator,

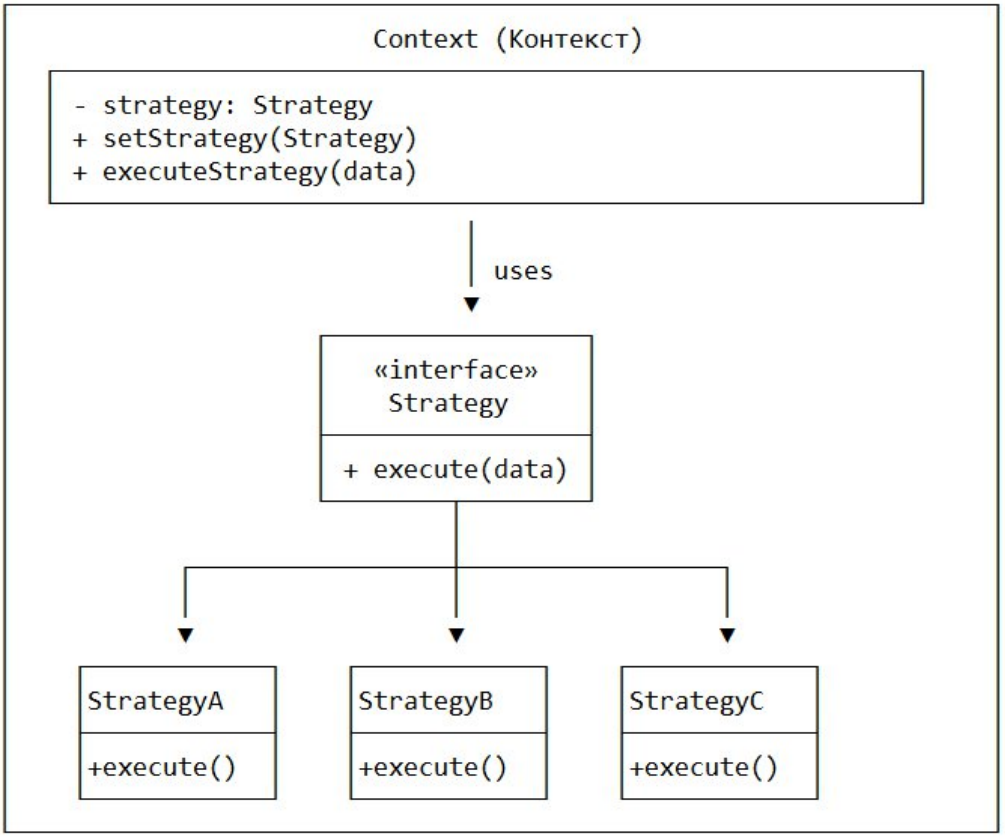
Группа	Назначение	Паттерны
		Facade, Flyweight, Proxy
Поведенческие	Взаимодействие объектов	Strategy, Observer, Command, Chain of Responsibility, State, Template Method, Visitor, Iterator, Mediator, Memento, Interpreter

2.2. Паттерн Strategy (Стратегия)

Назначение: Определяет семейство алгоритмов, инкапсулирует каждый из них и делает их взаимозаменяемыми. Стратегия позволяет изменять алгоритм независимо от клиентов, которые его используют.

Проблема: В системе есть несколько вариантов выполнения одного действия (расчёт цены со скидкой, разные способы валидации, разные алгоритмы сортировки). При добавлении нового варианта приходится изменять существующий код (нарушение OCP).

Решение:



Применение в курсовом проекте:

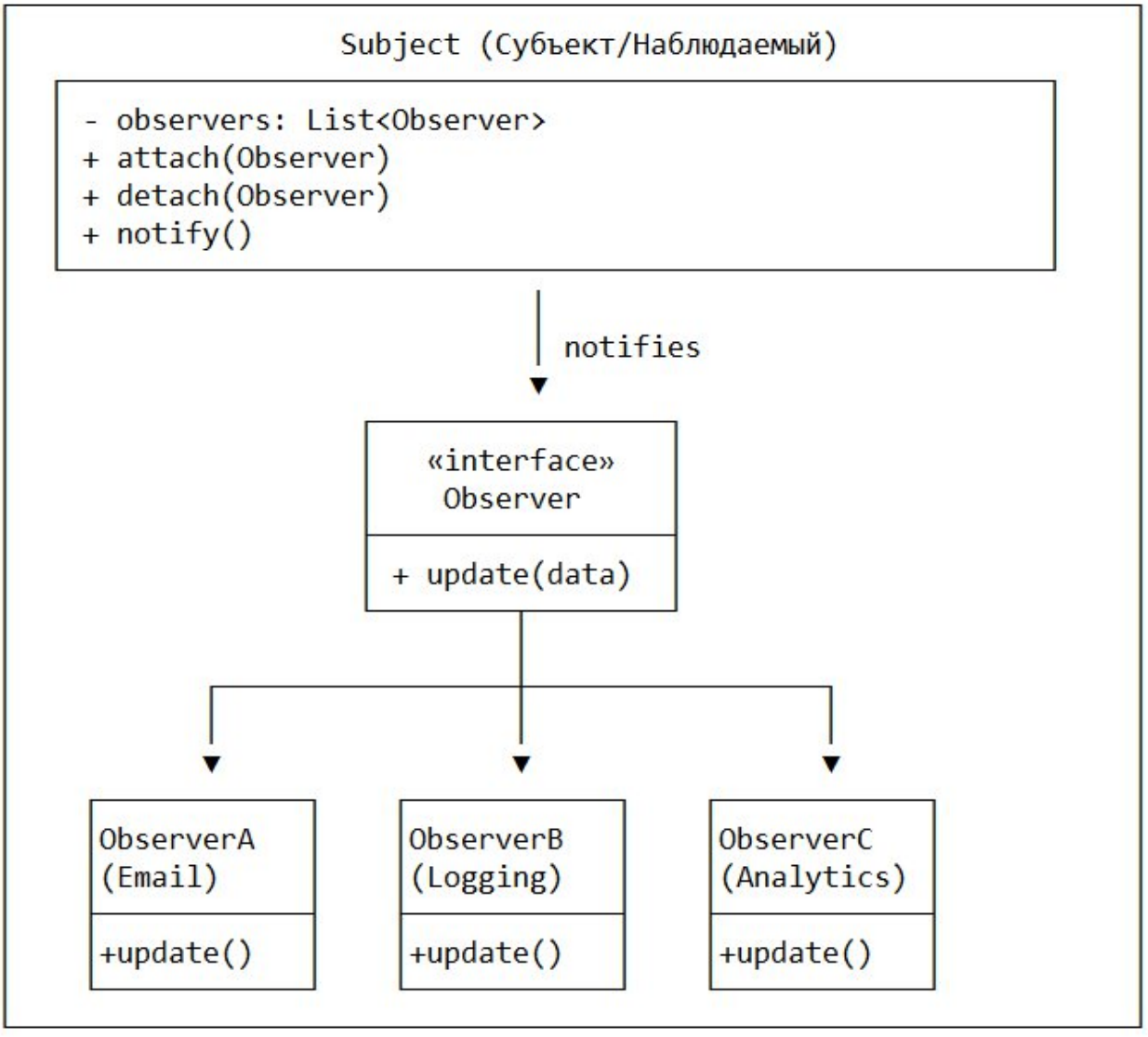
Задача	**Варианты стратегий**
Расчёт стоимости участия	Стандартная цена, Раннее бронирование (скидка), Групповая скидка
Валидация данных	Валидация email, Валидация телефона, Валидация дат
Формирование отчёта	PDF, Excel, CSV
Уведомления	Email, SMS, Push

2.3. Паттерн Observer (Наблюдатель)

Назначение: Определяет зависимость "один-ко-многим" между объектами так, что при изменении состояния одного объекта все зависящие от него объекты уведомляются и обновляются автоматически.

Проблема: При наступлении события (регистрация пользователя, создание заказа) нужно уведомить несколько компонентов (отправить email, записать в лог, обновить статистику). Прямые вызовы приводят к жёсткой связанности.

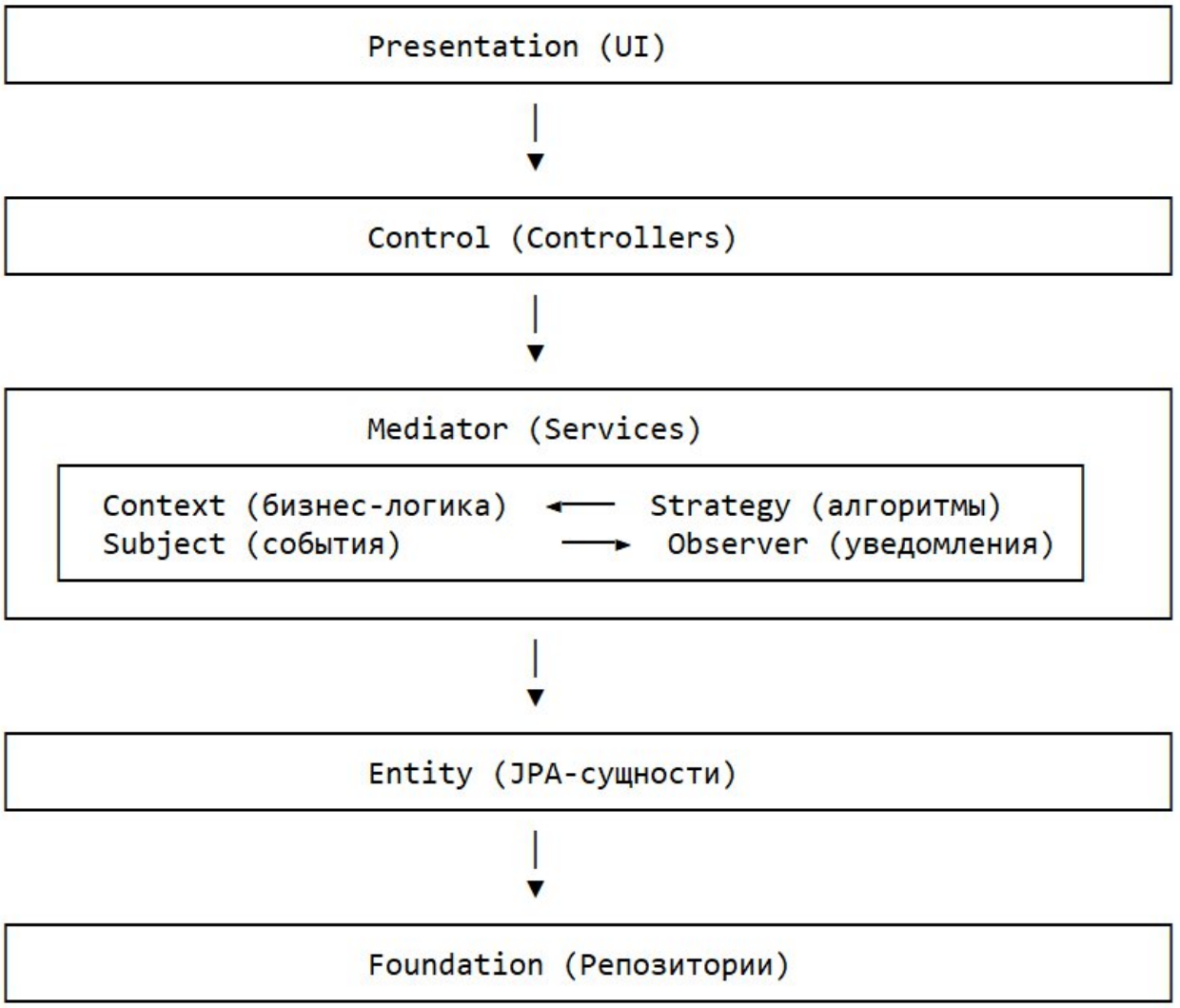
Решение:



Применение в проекте:

Событие	Наблюдатели
Регистрация на мероприятие	Email уведомление, Логирование, Обновление статистики
Создание мероприятия	Уведомление администратора, Индексация поиска
Отмена регистрации	Email уведомление, Освобождение места
Оплата	Email с чеком, Обновление статуса, Аналитика

2.4. Место паттернов в архитектуре PCMEF



3. ПРИМЕР ВЫПОЛНЕНИЯ (СКВОЗНОЙ ПРОЕКТ)

Проект: «Система управления мероприятиями» (Event Management System)

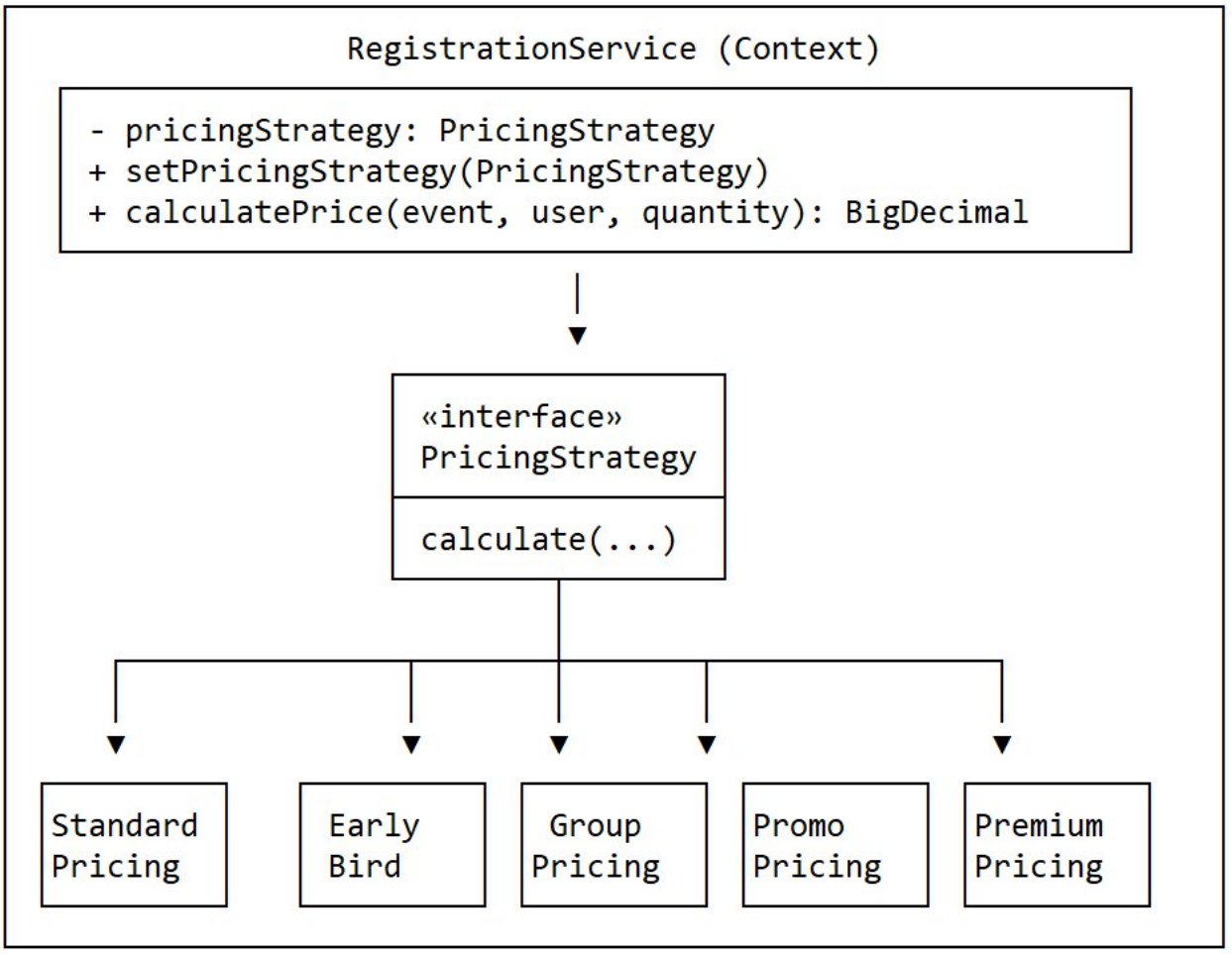
3.1. Паттерн Strategy: Расчёт стоимости участия

Проблема: Стоимость участия в мероприятии может рассчитываться по-разному:

1. Стандартная цена
2. Скидка за раннее бронирование (за 30+ дней до мероприятия)
3. Групповая скидка (при регистрации 5+ человек)
4. Скидка по промокоду

Решение: Реализовать интерфейс PricingStrategy и несколько конкретных стратегий.

Структура:



Интерфейс стратегии:

```
package ru.edu.project.mediator.strategy;

import ru.edu.project.entity.Event;
import ru.edu.project.entity.User;
import java.math.BigDecimal;

/**
 * Стратегия расчёта стоимости участия в мероприятии
```

```
 */
public interface PricingStrategy {

    /**
     * Рассчитать стоимость участия
     * @param event мероприятие
     * @param user пользователь
     * @param quantity количество участников
     * @return итоговая стоимость
     */
    BigDecimal calculatePrice(Event event, User user, int quantity);

    /**
     * Название стратегии (для отображения в UI)
     */
    String getStrategyName();
}
```

Конкретные стратегии:

java

```
package ru.edu.project.mediator.strategy.impl;

import ru.edu.project.entity.Event;
import ru.edu.project.entity.User;
import ru.edu.project.mediator.strategy.PricingStrategy;
import java.math.BigDecimal;
import java.time.LocalDateTime;

/**
 * Стандартная стратегия ценообразования (без скидок)
```

```
 */
public class StandardPricing implements PricingStrategy {

    @Override
    public BigDecimal calculatePrice(Event event, User user, int quantity) {
        if (event.getBasePrice() == null) {
            return BigDecimal.ZERO;
        }
        return event.getBasePrice().multiply(BigDecimal.valueOf(quantity));
    }

    @Override
    public String getStrategyName() {
        return "Стандартная цена";
    }
}

package ru.edu.project.mediator.strategy.impl;

import ru.edu.project.entity.Event;
import ru.edu.project.entity.User;
import ru.edu.project.mediator.strategy.PricingStrategy;
import java.math.BigDecimal;
import java.time.LocalDateTime;

/**
 * Стратегия "Раннее бронирование" (скидка 20% при бронировании за 30+ дней)
 */
```

```
public class EarlyBirdPricing implements PricingStrategy {

    private static final BigDecimal DISCOUNT = new BigDecimal("0.8"); // 20% скидка
    private static final int DAYS_THRESHOLD = 30;

    @Override
    public BigDecimal calculatePrice(Event event, User user, int quantity) {
        if (event.getBasePrice() == null) {
            return BigDecimal.ZERO;
        }

        BigDecimal baseTotal = event.getBasePrice().multiply(BigDecimal.valueOf(quantity));

        // Проверка даты мероприятия
        LocalDateTime now = LocalDateTime.now();
        LocalDateTime eventStart = event.getStartDate();

        if (eventStart.minusDays(DAYS_THRESHOLD).isAfter(now)) {
            return baseTotal.multiply(DISCOUNT);
        }

        return baseTotal;
    }

    @Override
    public String getStrategyName() {
        return "Раннее бронирование (скидка 20%)";
```

```
    }
}
```

```
package ru.edu.project.mediator.strategy.impl;

import ru.edu.project.entity.Event;
import ru.edu.project.entity.User;
import ru.edu.project.mediator.strategy.PricingStrategy;
import java.math.BigDecimal;

/**
 * Групповая скидка (скидка 10% при регистрации 5+ человек)
 */
public class GroupPricing implements PricingStrategy {

    private static final BigDecimal DISCOUNT = new BigDecimal("0.9"); // 10% скидка
    private static final int GROUP_SIZE = 5;

    @Override
    public BigDecimal calculatePrice(Event event, User user, int quantity) {
        if (event.getBasePrice() == null) {
            return BigDecimal.ZERO;
        }

        BigDecimal baseTotal = event.getBasePrice().multiply(BigDecimal.valueOf(quantity));

        if (quantity >= GROUP_SIZE) {
```

```
            return baseTotal.multiply(DISCOUNT);
        }

        return baseTotal;
    }

    @Override
    public String getStrategyName() {
        return "Групповая скидка (10% при регистрации 5+)";
    }
}
```

```
package ru.edu.project.mediator.strategy.impl;

import ru.edu.project.entity.Event;
import ru.edu.project.entity.User;
import ru.edu.project.mediator.strategy.PricingStrategy;
import java.math.BigDecimal;
import java.util.Map;
import java.util.concurrent.ConcurrentHashMap;

/**
 * Стратегия скидки по промокоду
 */
public class PromoPricing implements PricingStrategy {

    private final Map<String, BigDecimal> promoCodes = new
ConcurrentHashMap<>();
```

```
    public PromoPricing() {
        // Инициализация промокодов (в реальном проекте — из БД)
        promoCodes.put("WELCOME10", new BigDecimal("0.9"));  // 10% скидка
        promoCodes.put("SUMMER20", new BigDecimal("0.8"));  // 20% скидка
        promoCodes.put("EARLY30", new BigDecimal("0.7"));  // 30% скидка
    }

    public void addPromoCode(String code, BigDecimal discount) {
        promoCodes.put(code, discount);
    }

    public boolean isValidPromoCode(String code) {
        return promoCodes.containsKey(code);
    }

    @Override
    public BigDecimal calculatePrice(Event event, User user, int quantity) {
        // Этот метод не используется напрямую — требуется промокод
        throw new UnsupportedOperationException(
            "Для PromoPricing используйте calculatePriceWithPromoCode()"
        );
    }

    public BigDecimal calculatePriceWithPromoCode(Event event, User user,
                                                  int quantity, String promoCode) {
        if (event.getBasePrice() == null) {
```

```
            return BigDecimal.ZERO;
        }

        BigDecimal baseTotal = event.getBasePrice().multiply(BigDecimal.valueOf(quantity));
        BigDecimal discount = promoCodes.getOrDefault(promoCode, BigDecimal.ONE);

        return baseTotal.multiply(discount);
    }

    @Override
    public String getStrategyName() {
        return "Промокод";
    }
}
```

Контекст (использование стратегии в сервисе):

```
package ru.edu.project.mediator.services;

import ru.edu.project.entity.Event;
import ru.edu.project.entity.User;
import ru.edu.project.mediator.strategy.PricingStrategy;
import ru.edu.project.mediator.strategy.impl.StandardPricing;
import java.math.BigDecimal;

@Service
public class RegistrationService {
```

```
    private PricingStrategy pricingStrategy;

    public RegistrationService() {
        this.pricingStrategy = new StandardPricing(); // стратегия по умолчанию
    }

    /**
     * Установка стратегии ценообразования
     */
    public void setPricingStrategy(PricingStrategy strategy) {
        this.pricingStrategy = strategy;
    }

    /**
     * Получение текущей стратегии
     */
    public PricingStrategy getPricingStrategy() {
        return pricingStrategy;
    }

    /**
     * Расчёт стоимости регистрации с использованием текущей стратегии
     */
    public BigDecimal calculateRegistrationPrice(Event event, User user, int quantity) {
        return pricingStrategy.calculatePrice(event, user, quantity);
    }
```

```
    // ... остальные методы
}
```

Использование в контроллере:

```
@RestController
@RequestMapping("/api/events")
public class EventController {

    private final RegistrationService registrationService;

    @PostMapping("/{id}/calculate-price")
    public ResponseEntity<PriceResponse> calculatePrice(
            @PathVariable Long id,
            @RequestBody PriceRequest request) {

        Event event = eventService.getEventById(id);
        User user = userService.getUserById(request.getUserId());

        // Выбор стратегии на основе параметров запроса
        PricingStrategy strategy = selectStrategy(request);
        registrationService.setPricingStrategy(strategy);

        BigDecimal price = registrationService.calculateRegistrationPrice(
            event, user, request.getQuantity()
        );

        return ResponseEntity.ok(new PriceResponse(price,
strategy.getStrategyName()));
```

```
    }

    private PricingStrategy selectStrategy(PriceRequest request) {
        if (request.getPromoCode() != null
&& !request.getPromoCode().isEmpty()) {
            return new PromoPricing();
        }
        if (request.getQuantity() >= 5) {
            return new GroupPricing();
        }
        if (request.isEarlyBird()) {
            return new EarlyBirdPricing();
        }
        return new StandardPricing();
    }
}
```

3.2. Паттерн Observer: Уведомления о событиях

Проблема: При регистрации пользователя на мероприятие нужно выполнить несколько действий:

Отправить email-подтверждение

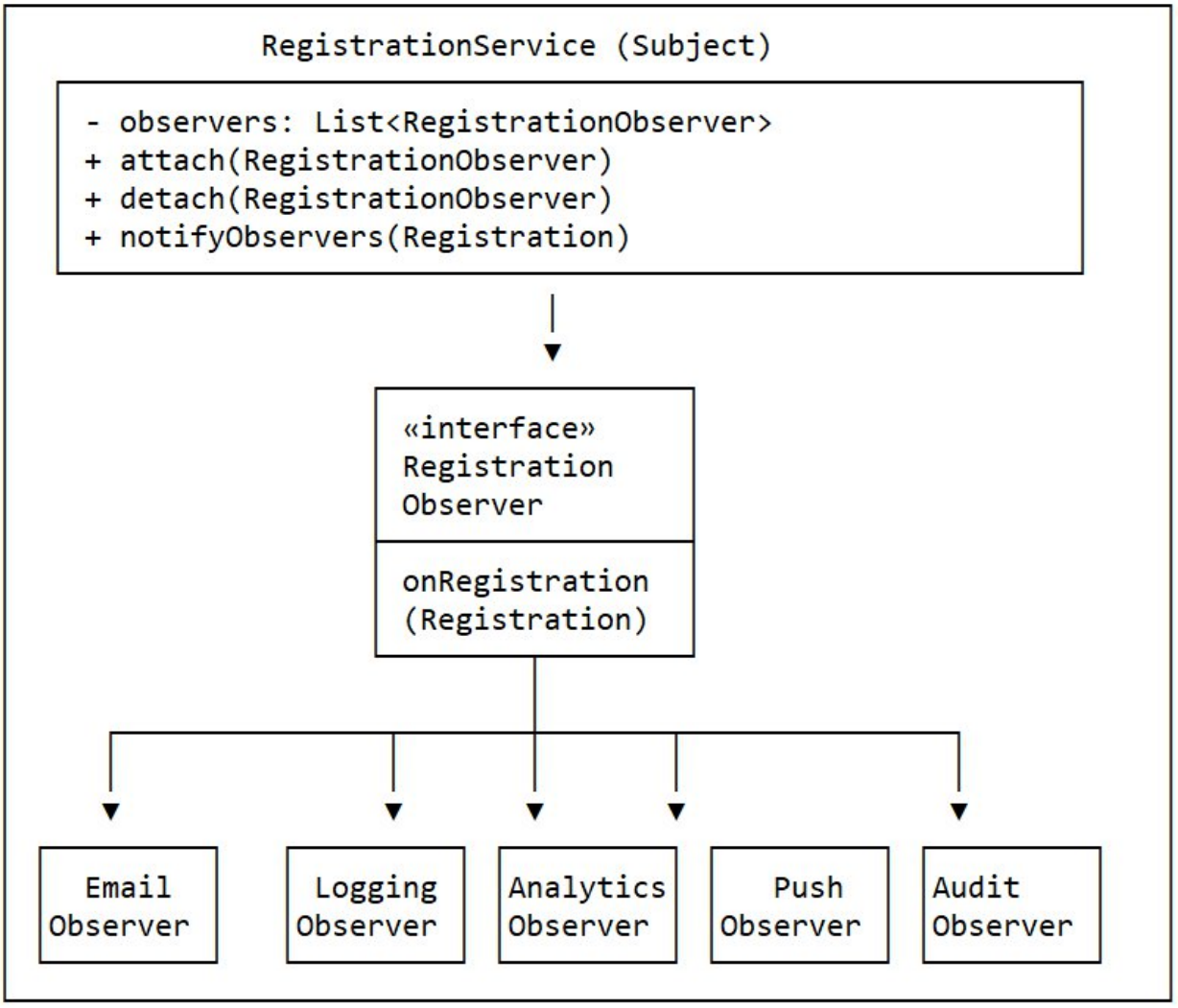
Записать в лог

Обновить статистику мероприятий

Отправить push-уведомление (если есть)

Решение: Реализовать паттерн Observer.

Структура:



Интерфейс наблюдателя:

```
package ru.edu.project.mediator.observer;

import ru.edu.project.entity.Registration;

/**
 * Наблюдатель за событиями регистрации
 */
public interface RegistrationObserver {

    /**
```

```
     * Вызывается при создании новой регистрации
     */
    void onRegistrationCreated(Registration registration);

    /**
     * Вызывается при подтверждении регистрации (после оплаты)
     */
    void onRegistrationConfirmed(Registration registration);

    /**
     * Вызывается при отмене регистрации
     */
    void onRegistrationCancelled(Registration registration);
}
```

Субъект (наблюдаемый объект):

```
package ru.edu.project.mediator.services;

import ru.edu.project.entity.Registration;
import ru.edu.project.mediator.observer.RegistrationObserver;
import org.springframework.stereotype.Service;

import java.util.ArrayList;
import java.util.List;

@Service
public class RegistrationEventPublisher {

    private final List<RegistrationObserver> observers = new ArrayList<>();
```

```
/**
 * Подписка на события
 */
public void attach(RegistrationObserver observer) {
    observers.add(observer);
}

/**
 * Отписка от событий
 */
public void detach(RegistrationObserver observer) {
    observers.remove(observer);
}

/**
 * Уведомление о создании регистрации
 */
public void notifyRegistrationCreated(Registration registration) {
    for (RegistrationObserver observer : observers) {
        observer.onRegistrationCreated(registration);
    }
}

/**
 * Уведомление о подтверждении регистрации
 */
public void notifyRegistrationConfirmed(Registration registration) {
    for (RegistrationObserver observer : observers) {
```

```
            observer.onRegistrationConfirmed(registration);
        }
    }

    /**
     * Уведомление об отмене регистрации
     */
    public void notifyRegistrationCancelled(Registration registration) {
        for (RegistrationObserver observer : observers) {
            observer.onRegistrationCancelled(registration);
        }
    }
}
```

Конкретные наблюдатели:

```
package ru.edu.project.mediator.observer.impl;

import ru.edu.project.entity.Registration;
import ru.edu.project.mediator.observer.RegistrationObserver;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.stereotype.Component;

/**
 * Наблюдатель для логирования событий регистрации
 */
@Component
public class LoggingObserver implements RegistrationObserver {
```

```
    private static final Logger log = LoggerFactory.getLogger(LoggingObserver.class);

    @Override
    public void onRegistrationCreated(Registration registration) {
        log.info("Создана регистрация ID={}, пользователь={}, мероприятие={}",
                registration.getId(),
                registration.getUser().getEmail(),
                registration.getEvent().getTitle()
        );
    }

    @Override
    public void onRegistrationConfirmed(Registration registration) {
        log.info("Подтверждена регистрация ID={}, пользователь={}, мероприятие={}",
                registration.getId(),
                registration.getUser().getEmail(),
                registration.getEvent().getTitle()
        );
    }

    @Override
    public void onRegistrationCancelled(Registration registration) {
        log.warn("Отменена регистрация ID={}, пользователь={}, мероприятие={}",
                registration.getId(),
                registration.getUser().getEmail(),
```

```
            registration.getEvent().getTitle()
        );
    }
}
```

java

```
package ru.edu.project.mediator.observer.impl;

import ru.edu.project.entity.Registration;
import ru.edu.project.mediator.observer.RegistrationObserver;
import org.springframework.stereotype.Component;

/**
 * Наблюдатель для отправки email-уведомлений
 */
@Component
public class EmailObserver implements RegistrationObserver {

    private final EmailService emailService;

    public EmailObserver(EmailService emailService) {
        this.emailService = emailService;
    }

    @Override
    public void onRegistrationCreated(Registration registration) {
        String subject = "Подтверждение регистрации на мероприятие";
        String body = String.format(
            "Вы зарегистрировались на мероприятие \"%s\".\n" +
            "Дата: %s\n" +
```

```
                "Для подтверждения перейдите по ссылке: ...",
                registration.getEvent().getTitle(),
                registration.getEvent().getStartDate()
            );
            emailService.sendEmail(registration.getUser().getEmail(), subject,
body);
        }

        @Override
        public void onRegistrationConfirmed(Registration registration) {
            String subject = "Регистрация подтверждена";
            String body = String.format(
                "Ваша регистрация на мероприятие \"%s\" подтверждена!\n" +
                "Ждём вас %s",
                registration.getEvent().getTitle(),
                registration.getEvent().getStartDate()
            );
            emailService.sendEmail(registration.getUser().getEmail(), subject,
body);
        }

        @Override
        public void onRegistrationCancelled(Registration registration) {
            String subject = "Регистрация отменена";
            String body = String.format(
                "Регистрация на мероприятие \"%s\" отменена.\n" +
                "Если вы не инициировали отмену, свяжитесь с поддержкой.",
                registration.getEvent().getTitle()
            );
```

```
            emailService.sendEmail(registration.getUser().getEmail(),          subject,
body);
        }
    }

    package ru.edu.project.mediator.observer.impl;

    import ru.edu.project.entity.Registration;
    import ru.edu.project.mediator.observer.RegistrationObserver;
    import org.springframework.stereotype.Component;

    /**
     * Наблюдатель для обновления аналитики
     */
    @Component
    public class AnalyticsObserver implements RegistrationObserver {

        private final StatisticsService statisticsService;

        public AnalyticsObserver(StatisticsService statisticsService) {
            this.statisticsService = statisticsService;
        }

        @Override
        public void onRegistrationCreated(Registration registration) {
            // Обновление счётчика регистраций для мероприятия
            statisticsService.incrementRegistrationCount(
                registration.getEvent().getId()
            );
```

```
    }

    @Override
    public void onRegistrationConfirmed(Registration registration) {
        // Обновление счётчика подтверждённых регистраций
        statisticsService.incrementConfirmedCount(
            registration.getEvent().getId()
        );
    }

    @Override
    public void onRegistrationCancelled(Registration registration) {
        // Обновление счётчика отмен
        statisticsService.decrementRegistrationCount(
            registration.getEvent().getId()
        );
    }
}
```

Интеграция в сервис:

```
package ru.edu.project.mediator.services;

import ru.edu.project.entity.Event;
import ru.edu.project.entity.Registration;
import ru.edu.project.entity.User;
import ru.edu.project.foundation.repositories.EventRepository;
import ru.edu.project.foundation.repositories.RegistrationRepository;
import ru.edu.project.foundation.repositories.UserRepository;
import ru.edu.project.mediator.observer.RegistrationObserver;
```

```java
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;

import java.util.ArrayList;
import java.util.List;

@Service
public class RegistrationService {

    private final UserRepository userRepository;
    private final EventRepository eventRepository;
    private final RegistrationRepository registrationRepository;
    private final RegistrationEventPublisher eventPublisher;

    public RegistrationService(UserRepository userRepository,
                               EventRepository eventRepository,
                               RegistrationRepository registrationRepository,
                               RegistrationEventPublisher eventPublisher) {
        this.userRepository = userRepository;
        this.eventRepository = eventRepository;
        this.registrationRepository = registrationRepository;
        this.eventPublisher = eventPublisher;
    }

    @Transactional
    public Registration registerUserForEvent(Long userId, Long eventId) {
        // Бизнес-логика регистрации
        User user = userRepository.findById(userId)
            .orElseThrow(() -> new UserNotFoundException(userId));
```

```
        Event event = eventRepository.findById(eventId)
            .orElseThrow(() -> new EventNotFoundException(eventId));

        if (!event.checkAvailability()) {
            throw new NoSeatsAvailableException(eventId);
        }

        Registration registration = event.registerUser(user);
        Registration saved = registrationRepository.save(registration);

        // Уведомление наблюдателей о создании регистрации
        eventPublisher.notifyRegistrationCreated(saved);

        return saved;
    }

    @Transactional
    public void confirmRegistration(Long registrationId) {
        Registration registration = registrationRepository.findById(registrationId)
            .orElseThrow(() -> new RegistrationNotFoundException(registrationId));

        registration.confirm();
        registrationRepository.update(registration);

        // Уведомление наблюдателей о подтверждении
        eventPublisher.notifyRegistrationConfirmed(registration);
```

```
    }

    @Transactional
    public void cancelRegistration(Long registrationId) {
        Registration registration =
registrationRepository.findById(registrationId)
            .orElseThrow(() -> new
RegistrationNotFoundException(registrationId));

        registration.cancel();
        registrationRepository.update(registration);

        // Уведомление наблюдателей об отмене
        eventPublisher.notifyRegistrationCancelled(registration);
    }
}
```

Настройка подписки наблюдателей:

```
package ru.edu.project.config;

import ru.edu.project.mediator.observer.RegistrationObserver;
import ru.edu.project.mediator.services.RegistrationEventPublisher;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Configuration;

import javax.annotation.PostConstruct;
import java.util.List;

@Configuration
```

```
public class ObserverConfig {

    @Autowired
    private RegistrationEventPublisher eventPublisher;

    @Autowired
    private List<RegistrationObserver> observers;

    @PostConstruct
    public void registerObservers() {
        for (RegistrationObserver observer : observers) {
            eventPublisher.attach(observer);
        }
    }
}
```

4. МЕТОДИКА ВЫПОЛНЕНИЯ

Шаг 1: Выбор места для применения паттернов (

Проанализируйте бизнес-логику вашего проекта

Найдите места, где используются разные алгоритмы расчёта

Найдите места, где одно событие требует множественных реакций

Таблица для Strategy:

Алгоритм	**Варианты**	**Где используется**
Расчёт стоимости	Стандартный, Со скидкой, Групповой	Registration Service
...	...	...

Таблица для Observer:

Событие	Реакции (наблюдатели)	Где возникает
Создание заказа	Email, Логирование, Склад	Order Service
...	...	...

Шаг 2: Реализация паттерна Strategy

Создать интерфейс стратегии (XxxStrategy)

Реализовать 2-3 конкретные стратегии

Модифицировать сервис для использования стратегии

Добавить возможность переключения стратегий

Шаг 3: Реализация паттерна Observer

Создать интерфейс наблюдателя (XxxObserver)

Создать класс-субъект (издатель событий)

Реализовать 2-3 конкретных наблюдателя

Интегрировать уведомления в бизнес-логику

Шаг 4: Тестирование

Написать тесты для каждой стратегии

Написать тесты для каждого наблюдателя

Проверить, что уведомления работают

5. КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Что такое паттерн Strategy? Когда его следует применять?
2. Что такое паттерн Observer? Когда его следует применять?
3. В чём разница между attach() и detach() в Observer?
4. Какие принципы SOLID реализует паттерн Strategy?

5. Как паттерн Strategy помогает соблюдать принцип открытости/закрытости (OCP)?
6. Можно ли комбинировать несколько стратегий одновременно?
7. Как избежать утечек памяти при использовании Observer?
8. В чём разница между pull и push моделями в Observer?
9. Как паттерны Strategy и Observer связаны с архитектурой PCMEF?
10. Как протестировать Observer без реальных внешних сервисов (email, API)?
11. Какие альтернативы паттерну Observer существуют в современном Java (Spring Events)?

6. ТРЕБОВАНИЯ К ОТЧЕТУ

Структура отчета

1. Титульный лист
2. Содержание
3. Введение (цель работы, краткое описание проекта)
4. Паттерн Strategy
   4.1. Обоснование выбора (диаграмма классов)
   4.2. Реализация интерфейса стратегии
   4.3. Реализация конкретных стратегий (2-3)
   4.4. Интеграция в сервис (код)
   4.5. Тестирование
5. Паттерн Observer
   5.1. Обоснование выбора (диаграмма классов)
   5.2. Реализация интерфейса наблюдателя
   5.3. Реализация конкретных наблюдателей (2-3)
   5.4. Реализация субъекта (издателя)
   5.5. Интеграция в сервис (код)

5.6. Тестирование

6. Заключение (выводы, плюсы применения паттернов)

7. Список использованных источников

8. Приложения (полный код)

7. КРИТЕРИИ ОЦЕНИВАНИЯ

Критерий	**Макс. балл**	**Описание**
Обоснование выбора Strategy	3	Проблема → решение, диаграмма
Реализация Strategy	6	Интерфейс + 2-3 реализации
Интеграция Strategy	3	Внедрение в сервис
Обоснование выбора Observer	3	Проблема → решение, диаграмма
Реализация Observer	6	Интерфейс + 2-3 реализации
Интеграция Observer	3	Внедрение в сервис
Тестирование	3	Тесты для стратегий и наблюдателей

Критерий	Макс. балл	Описание
Оформление отчета	2	Соответствие требованиям
Ответы на вопросы	3	Понимание материала
ИТОГО	32	

Лабораторная работа №13

Уточнение диаграммы классов проектирования

1. Цель и задачи работы

Цель: Уточнить диаграмму классов проектирования (Design Class Diagram) на основе разработанных спецификаций прецедентов (ЛР10), диаграмм последовательности (ЛР11) и реализованных паттернов (ЛР12), а также детализировать сигнатуры методов и типы данных.

Задачи:

1. Проанализировать спецификации прецедентов и диаграммы последовательности
2. Выявить все классы, участвующие в реализации сценариев
3. Уточнить атрибуты классов с типами данных
4. Детализировать методы с полными сигнатурами (параметры, возвращаемые значения, исключения)
5. Уточнить отношения между классами (ассоциации, агрегации, композиции, наследование)

6. Добавить классы DTO, исключений, паттернов (Strategy, Observer)
7. Построить итоговую диаграмму классов проектирования

2. ТЕОРЕТИЧЕСКОЕ ОБОСНОВАНИЕ

2.1. Место диаграммы классов в жизненном цикле

Диаграмма классов проектирования - это мост между анализом (требования, спецификации) и реализацией (код). Она отвечает на вопрос: «Из каких классов состоит система и как они связаны?».

Спецификации прецедентов (ЛР10) ↓
Диаграммы последовательности (ЛР11) ↓
Применение паттернов (ЛР12) ↓
Диаграмма классов проектирования (ЛР13) ← **МЫ ЗДЕСЬ** ↓
Реализация (Mediator, Control, Entity, Foundation)

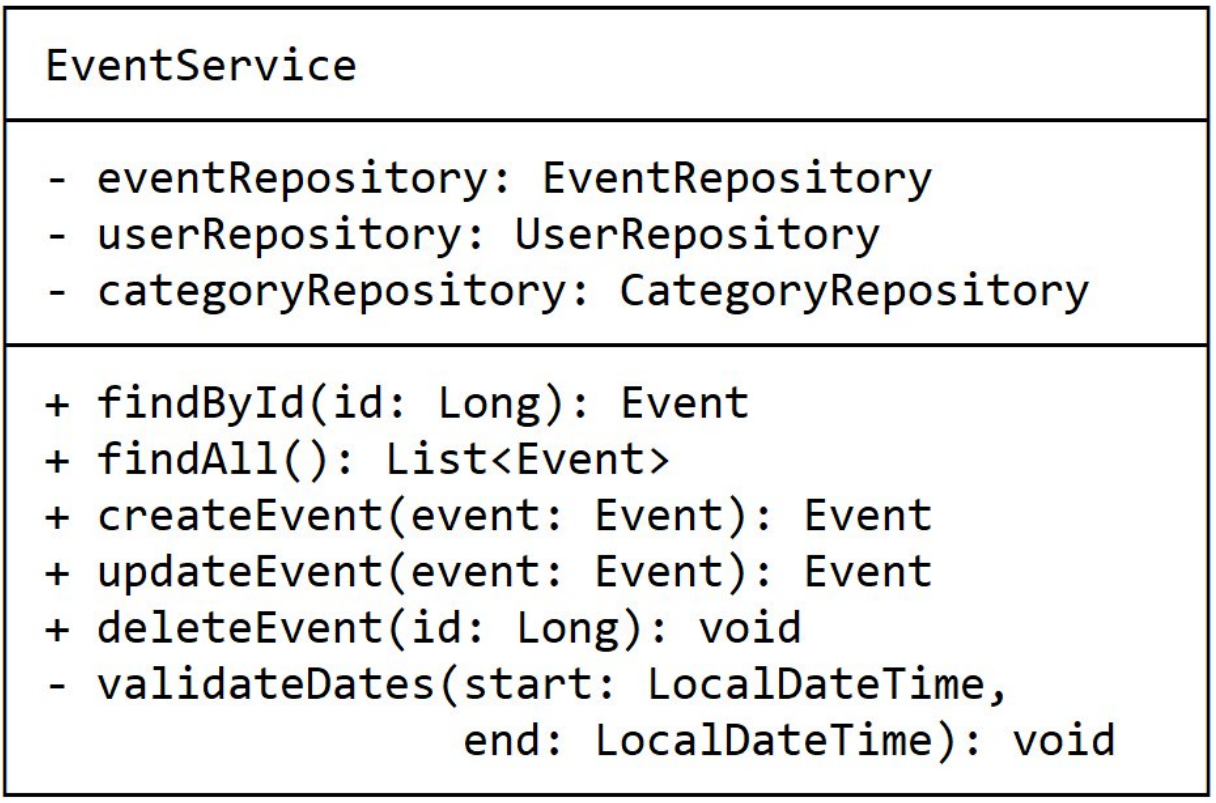
2.2. Отличие Domain Model от Design Class Diagram

Аспект	**Domain Model (ЛР2)**	**Design Class Diagram (ЛР13)**
Уровень абстракции	Концептуальный	Технический (близкий к коду)

Аспект	Domain Model (ЛР2)	Design Class Diagram (ЛР13)
Методы	Только ключевые бизнес-методы	Полные сигнатуры (параметры, возвращаемые типы, исключения)
Типы данных	Бизнес-типы (Email, Phone)	Java-типы (String, LocalDateTime, Long)
Отношения	Бизнес-связи	Технические зависимости (внедрение через конструктор)
Дополнительные классы	Только сущности	DTO, исключения, репозитории, сервисы, контроллеры, паттерны

2.3. Элементы диаграммы классов проектирования

Класс (детализированный):



Типы отношений:

Отношение	Обозначение	Описание	Пример
Ассоциация	---	Простая связь	User --- Registration
Агрегация	◇---	Часть-целое (части могут существовать отдельно)	Event ◇--- Registration
Композиция	◆---	Часть-целое (части не могут существовать отдельно)	Event ◆--- Session

Отношение	Обозначение	Описание	Пример
Наследование	▷---	Расширение класса	EventService ▷--- BaseService
Реализация	◁ - - - (пунктир)	Реализация интерфейса	EventServiceImpl ◁--- IEventService
Зависимость	- - -> (пунктир)	Временная связь (например, параметр метода)	—

2.4. Стереотипы классов в PCMEF

Стереотип	Назначение	Пример
<<Entity>>	JPA-сущность	class User <<Entity>>
<<Repository>>	Доступ к данным	interface UserRepository <<Repository>>
<<Service>>	Бизнес-логика	class UserService <<Service>>
<<Contro	REST-контроллер	class UserController

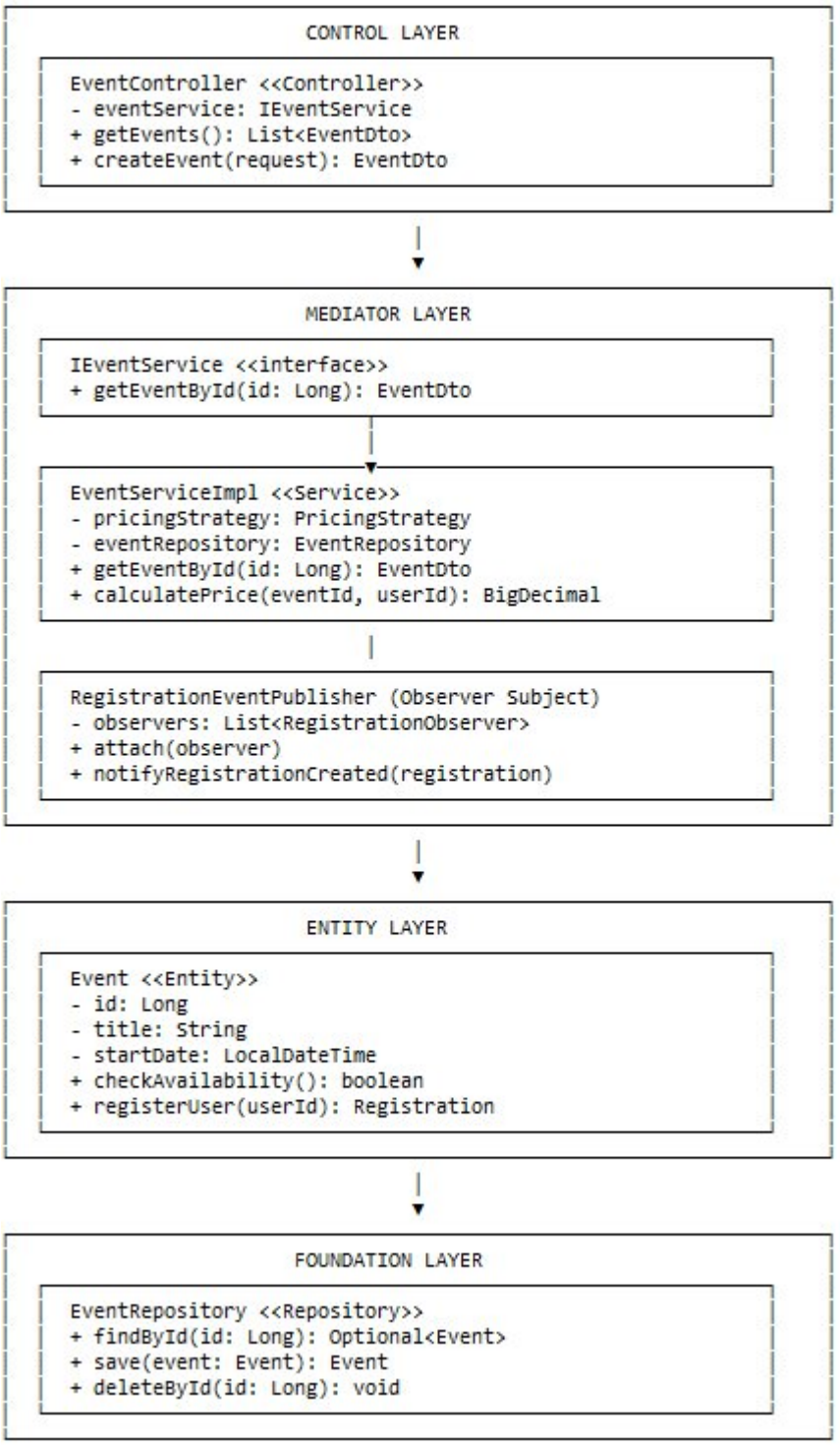
Стереотип	Назначение	Пример
<<Controller>>		class UserController <<Controller>>
<<DTO>>	Data Transfer Object	class UserDto <<DTO>>
<<Exception>>	Исключение	class UserNotFoundException <<Exception>>
<<Strategy>>	Стратегия	interface PricingStrategy <<Strategy>>
<<Observer>>	Наблюдатель	interface RegistrationObserver <<Observer>>

2.5. Правила построения Design Class Diagram

Правило	Описание
1	Показывать только классы, относящиеся к текущему контексту (не перегружать)
2	Указывать полные сигнатуры методов (параметры + возвращаемый тип)

Правило	Описание
3	Указывать видимость: + (public), - (private), # (protected)
4	Для атрибутов указывать тип данных
5	Для связей указывать кратности (1, 0.., 1..)
6	Для зависимостей указывать направление
7	Группировать классы по пакетам/слоям

2.6. Связь с компонентами PCMEF



3. ПРИМЕР ВЫПОЛНЕНИЯ (СКВОЗНОЙ ПРОЕКТ)

Проект: «Система управления мероприятиями» (Event Management System)

3.1. Анализ исходных данных

На основе ЛР10-ЛР12 выявлены следующие классы:

Из Domain Model (ЛР2):

Event, User, Registration, Payment, Category

Из спецификаций прецедентов (ЛР10):

Необходимость в DTO (EventDto, UserDto, RegistrationDto)

Из диаграмм последовательности (ЛР11):

Контроллеры, сервисы, репозитории

Из паттернов (ЛР12):

Стратегии (PricingStrategy)

Наблюдатели (RegistrationObserver)

3.2. Полная диаграмма классов проектирования

```
@startuml
!define ENTITY <<Entity>>
!define REPOSITORY <<Repository>>
!define SERVICE <<Service>>
!define CONTROLLER <<Controller>>
!define DTO <<DTO>>
!define EXCEPTION <<Exception>>
!define STRATEGY <<Strategy>>
!define OBSERVER <<Observer>>
!define INTERFACE <<interface>>

left to right direction

' ========== EXCEPTION LAYER ==========
package "Exceptions" {
  class BusinessException <<EXCEPTION>> {
```

```
        - message: String
        + BusinessException(message: String)
    }

    class UserNotFoundException <<EXCEPTION>> {
        + UserNotFoundException(userId: Long)
        + UserNotFoundException(email: String)
    }

    class EventNotFoundException <<EXCEPTION>> {
        + EventNotFoundException(eventId: Long)
    }

    class NoSeatsAvailableException <<EXCEPTION>> {
        + NoSeatsAvailableException(eventId: Long)
    }

    class DuplicateRegistrationException <<EXCEPTION>> {
        + DuplicateRegistrationException(userId: Long, eventId: Long)
    }

    BusinessException <|-- UserNotFoundException
    BusinessException <|-- EventNotFoundException
    BusinessException <|-- NoSeatsAvailableException
    BusinessException <|-- DuplicateRegistrationException
}

' ==================== DTO LAYER ====================
package "DTO" {
```

```
class UserDto <<DTO>> {
    - id: Long
    - email: String
    - name: String
    - role: String
    + fromEntity(user: User): UserDto
}

class EventDto <<DTO>> {
    - id: Long
    - title: String
    - description: String
    - startDate: LocalDateTime
    - endDate: LocalDateTime
    - maxParticipants: Integer
    - registeredCount: Integer
    - availableSeats: Integer
    - status: String
    - categoryName: String
    - createdBy: String
    + fromEntity(event: Event): EventDto
}

class RegistrationDto <<DTO>> {
    - id: Long
    - registrationDate: LocalDateTime
    - status: String
    - userId: Long
    - userName: String
```

```
        - eventId: Long
        - eventTitle: String
        + fromEntity(registration: Registration): RegistrationDto
    }
}

' ========== STRATEGY PATTERN ==========
package "Strategy" {
    interface PricingStrategy <<STRATEGY>> {
        + calculatePrice(event: Event, user: User, quantity: int): BigDecimal
        + getStrategyName(): String
    }

    class StandardPricing <<STRATEGY>> {
        + calculatePrice(event: Event, user: User, quantity: int): BigDecimal
        + getStrategyName(): String
    }

    class EarlyBirdPricing <<STRATEGY>> {
        - DISCOUNT: BigDecimal
        - DAYS_THRESHOLD: int
        + calculatePrice(event: Event, user: User, quantity: int): BigDecimal
        + getStrategyName(): String
    }

    class GroupPricing <<STRATEGY>> {
        - DISCOUNT: BigDecimal
        - GROUP_SIZE: int
```

```
        + calculatePrice(event: Event, user: User, quantity: int): BigDecimal
        + getStrategyName(): String
    }

    class PromoPricing <<STRATEGY>> {
        - promoCodes: Map<String, BigDecimal>
        + calculatePrice(event: Event, user: User, quantity: int): BigDecimal
        + calculatePriceWithPromoCode(event: Event, user: User, quantity: int,
promoCode: String): BigDecimal
        + isValidPromoCode(code: String): boolean
        + getStrategyName(): String
    }

    PricingStrategy <|.. StandardPricing
    PricingStrategy <|.. EarlyBirdPricing
    PricingStrategy <|.. GroupPricing
    PricingStrategy <|.. PromoPricing
}

' ==================== OBSERVER PATTERN
====================
package "Observer" {
    interface RegistrationObserver <<OBSERVER>> {
        + onRegistrationCreated(registration: Registration): void
        + onRegistrationConfirmed(registration: Registration): void
        + onRegistrationCancelled(registration: Registration): void
    }

    class EmailObserver <<OBSERVER>> {
```

```
    - emailService: EmailService
    + onRegistrationCreated(registration: Registration): void
    + onRegistrationConfirmed(registration: Registration): void
    + onRegistrationCancelled(registration: Registration): void
}

class LoggingObserver <<OBSERVER>> {
    - log: Logger
    + onRegistrationCreated(registration: Registration): void
    + onRegistrationConfirmed(registration: Registration): void
    + onRegistrationCancelled(registration: Registration): void
}

class AnalyticsObserver <<OBSERVER>> {
    - statisticsService: StatisticsService
    + onRegistrationCreated(registration: Registration): void
    + onRegistrationConfirmed(registration: Registration): void
    + onRegistrationCancelled(registration: Registration): void
}

RegistrationObserver <|.. EmailObserver
RegistrationObserver <|.. LoggingObserver
RegistrationObserver <|.. AnalyticsObserver

class RegistrationEventPublisher {
    - observers: List<RegistrationObserver>
    + attach(observer: RegistrationObserver): void
    + detach(observer: RegistrationObserver): void
    + notifyRegistrationCreated(registration: Registration): void
```

```
        + notifyRegistrationConfirmed(registration: Registration): void
        + notifyRegistrationCancelled(registration: Registration): void
    }

    RegistrationEventPublisher o-- RegistrationObserver
}

' ====================== ENTITY LAYER ======================
package "Entity" {
    class User <<ENTITY>> {
        - id: Long
        - email: String
        - passwordHash: String
        - name: String
        - role: String
        - createdAt: LocalDateTime
        - updatedAt: LocalDateTime
        - registrations: List<Registration>
        + checkPassword(rawPassword: String): boolean
        + addRegistration(registration: Registration): void
        + getRegistrations(): List<Registration>
    }

    class Category <<ENTITY>> {
        - id: Long
        - name: String
        - description: String
        - createdAt: LocalDateTime
        - events: List<Event>
```

```
}

class Event <<ENTITY>> {
    - id: Long
    - title: String
    - description: String
    - startDate: LocalDateTime
    - endDate: LocalDateTime
    - maxParticipants: Integer
    - status: String
    - basePrice: BigDecimal
    - createdAt: LocalDateTime
    - updatedAt: LocalDateTime
    - category: Category
    - createdBy: User
    - registrations: List<Registration>
    + checkAvailability(): boolean
    + getAvailableSeats(): int
    + registerUser(user: User): Registration
    + publish(): void
    + cancel(): void
    + complete(): void
}

class Registration <<ENTITY>> {
    - id: Long
    - registrationDate: LocalDateTime
    - status: String
    - createdAt: LocalDateTime
```

```
    - user: User
    - event: Event
    - payment: Payment
    + confirm(): void
    + cancel(): void
    + isConfirmed(): boolean
    + addPayment(payment: Payment): void
  }

  class Payment <<ENTITY>> {
    - id: Long
    - amount: BigDecimal
    - paymentDate: LocalDateTime
    - method: String
    - transactionId: String
    - createdAt: LocalDateTime
    - registration: Registration
    + refund(): void
    + isCompleted(): boolean
  }

  User "1" -- "0..*" Registration
  Event "1" -- "0..*" Registration
  Event "1" -- "1" Category
  Event "1" -- "1" User : created by
  Registration "1" -- "0..1" Payment
}
```

```
    '        ==================== FOUNDATION LAYER ====================
        package "Foundation" {
            interface UserRepository <<REPOSITORY>> {
                + findById(id: Long): Optional<User>
                + findAll(): List<User>
                + findByEmail(email: String): Optional<User>
                + findByRole(role: String): List<User>
                + existsByEmail(email: String): boolean
                + save(user: User): User
                + update(user: User): User
                + delete(user: User): void
                + deleteById(id: Long): void
            }

            interface EventRepository <<REPOSITORY>> {
                + findById(id: Long): Optional<Event>
                + findAll(): List<Event>
                + findByStatus(status: String): List<Event>
                + findByCategoryId(categoryId: Long): List<Event>
                + findByStartDateBetween(start: LocalDateTime, end: LocalDateTime): List<Event>
                + findByCreatedById(userId: Long): List<Event>
                + existsById(id: Long): boolean
                + save(event: Event): Event
                + update(event: Event): Event
                + delete(event: Event): void
                + deleteById(id: Long): void
            }
```

```
interface RegistrationRepository <<REPOSITORY>> {
    + findById(id: Long): Optional<Registration>
    + findAll(): List<Registration>
    + findByUserId(userId: Long): List<Registration>
    + findByEventId(eventId: Long): List<Registration>
    + findByUserIdAndEventId(userId: Long, eventId: Long): Optional<Registration>
    + findByStatus(status: String): List<Registration>
    + countConfirmedByEventId(eventId: Long): long
    + save(registration: Registration): Registration
    + update(registration: Registration): Registration
    + delete(registration: Registration): void
    + deleteById(id: Long): void
}

interface CategoryRepository <<REPOSITORY>> {
    + findById(id: Long): Optional<Category>
    + findAll(): List<Category>
    + findByName(name: String): Optional<Category>
    + save(category: Category): Category
    + update(category: Category): Category
    + delete(category: Category): void
    + deleteById(id: Long): void
}

interface PaymentRepository <<REPOSITORY>> {
    + findById(id: Long): Optional<Payment>
    + findAll(): List<Payment>
```

```
        + findByRegistrationId(registrationId: Long): Optional<Payment>
        + save(payment: Payment): Payment
        + update(payment: Payment): Payment
        + delete(payment: Payment): void
        + deleteById(id: Long): void
    }
}

' ========== MEDIATOR LAYER ==========
package "Mediator" {
    interface IUserService <<SERVICE>> {
        + getUserById(id: Long): UserDto
        + getUserByEmail(email: String): UserDto
        + getAllUsers(): List<UserDto>
        + getUsersByRole(role: String): List<UserDto>
        + createUser(email: String, password: String, name: String): UserDto
        + updateUser(id: Long, name: String, role: String): UserDto
        + deleteUser(id: Long): void
        + authenticate(email: String, password: String): UserDto
    }

    interface IEventService <<SERVICE>> {
        + getEventById(id: Long): EventDto
        + getAllEvents(): List<EventDto>
        + getPublishedEvents(): List<EventDto>
        + getEventsByCategory(categoryId: Long): List<EventDto>
        + getEventsInDateRange(start: LocalDateTime, end: LocalDateTime): List<EventDto>
```

```
        + createEvent(title: String, description: String, startDate: LocalDateTime, endDate: LocalDateTime, maxParticipants: Integer, categoryId: Long, createdBy: Long): EventDto
        + updateEvent(id: Long, title: String, description: String, startDate: LocalDateTime, endDate: LocalDateTime, maxParticipants: Integer, categoryId: Long): EventDto
        + publishEvent(id: Long): void
        + cancelEvent(id: Long): void
        + deleteEvent(id: Long): void
    }

    interface IRegistrationService <<SERVICE>> {
        + getRegistrationById(id: Long): RegistrationDto
        + getRegistrationsByUser(userId: Long): List<RegistrationDto>
        + getRegistrationsByEvent(eventId: Long): List<RegistrationDto>
        + registerUserForEvent(userId: Long, eventId: Long): RegistrationDto
        + cancelRegistration(registrationId: Long): void
        + confirmRegistration(registrationId: Long): void
        + calculatePrice(eventId: Long, userId: Long, quantity: int): BigDecimal
    }

    class UserServiceImpl <<SERVICE>> {
        - userRepository: UserRepository
        + UserServiceImpl(userRepository: UserRepository)
        + getUserById(id: Long): UserDto
        + authenticate(email: String, password: String): UserDto
        - validateUserData(email: String, name: String): void
    }
```

```
class EventServiceImpl <<SERVICE>> {
  - eventRepository: EventRepository
  - userRepository: UserRepository
  - categoryRepository: CategoryRepository
  + EventServiceImpl(eventRepository: EventRepository, userRepository: UserRepository, categoryRepository: CategoryRepository)
  + createEvent(...): EventDto
  - validateEventDates(start: LocalDateTime, end: LocalDateTime): void
}

class RegistrationServiceImpl <<SERVICE>> {
  - userRepository: UserRepository
  - eventRepository: EventRepository
  - registrationRepository: RegistrationRepository
  - pricingStrategy: PricingStrategy
  - eventPublisher: RegistrationEventPublisher
  + RegistrationServiceImpl(userRepository: UserRepository, eventRepository: EventRepository, registrationRepository: RegistrationRepository, eventPublisher: RegistrationEventPublisher)
  + registerUserForEvent(userId: Long, eventId: Long): RegistrationDto
  + calculatePrice(eventId: Long, userId: Long, quantity: int): BigDecimal
  + setPricingStrategy(strategy: PricingStrategy): void
  - validateRegistration(userId: Long, eventId: Long): void
}

IUserService <|.. UserServiceImpl
IEventService <|.. EventServiceImpl
```

```
        IRegistrationService <|.. RegistrationServiceImpl

        RegistrationServiceImpl --> PricingStrategy : uses
        RegistrationServiceImpl --> RegistrationEventPublisher : uses
        RegistrationEventPublisher --> RegistrationObserver : notifies

        UserServiceImpl --> UserRepository
        EventServiceImpl --> EventRepository
        EventServiceImpl --> UserRepository
        EventServiceImpl --> CategoryRepository
        RegistrationServiceImpl --> UserRepository
        RegistrationServiceImpl --> EventRepository
        RegistrationServiceImpl --> RegistrationRepository
    }

    ' ========== CONTROL LAYER ==========
    package "Control" {
        class UserController <<CONTROLLER>> {
            - userService: IUserService
            + UserController(userService: IUserService)
            + register(request: RegisterRequest): ResponseEntity<UserDto>
            + login(request: LoginRequest): ResponseEntity<AuthResponse>
            + getUserProfile(): ResponseEntity<UserDto>
            + updateUserProfile(request: UpdateUserRequest): ResponseEntity<UserDto>
        }

        class EventController <<CONTROLLER>> {
```

- eventService: IEventService
- registrationService: IRegistrationService
+ EventController(eventService: IEventService, registrationService: IRegistrationService)
+ getAllEvents(): ResponseEntity<List<EventDto>>
+ getEventById(id: Long): ResponseEntity<EventDto>
+ createEvent(request: CreateEventRequest): ResponseEntity<EventDto>
+ updateEvent(id: Long, request: UpdateEventRequest): ResponseEntity<EventDto>
+ deleteEvent(id: Long): ResponseEntity<Void>
+ publishEvent(id: Long): ResponseEntity<Void>
+ registerForEvent(id: Long): ResponseEntity<RegistrationDto>
+ calculatePrice(id: Long, request: PriceRequest): ResponseEntity<PriceResponse>
}

class PriceRequest {
- userId: Long
- quantity: int
- promoCode: String
- isEarlyBird: boolean
+ getters/setters
}

class PriceResponse {
- price: BigDecimal
- strategyName: String
+ PriceResponse(price: BigDecimal, strategyName: String)

```
    }

    EventController --> IEventService
    EventController --> IRegistrationService
    EventController --> PricingStrategy : selects
}

' ==================== DEPENDENCIES ====================
' Зависимости между слоями (только вниз)
Control --> Mediator
Mediator --> Entity
Mediator --> Foundation
Mediator --> Strategy
Mediator --> Observer
Foundation --> Entity

note top of Mediator
  **PCMEF Правила зависимостей:**
  1. Control → Mediator → Foundation → Entity
  2. Нет зависимостей снизу вверх
  3. Стратегии и наблюдатели — часть Mediator
end note

@enduml
```

3.3. Спецификация ключевых методов

Методы сущности Event:

Метод	Возвращаемый тип	Параметры	Исключения	Описание
checkAvailability	boolean	—	—	Проверяет наличие свободных мест
getAvailableSeats	int	—	—	Возвращает количество свободных мест
registerUser	Registration	user: User	IllegalStateException	Регистрирует пользователя на мероприятие
publish	void	—	—	Меняет статус на PUBLISHED
cancel	void	—	—	Меня

Метод	Возвращаемый тип	Параметры	Исключения	Описание
				ет статус на CANCELLED

Методы сервиса RegistrationServiceImpl:

Метод	**Возвращаемый тип**	**Параметры**	**Исключения**	**Описание**
registerUserForEvent	RegistrationDto	userId: Long, eventId: Long	UserNotFoundException, EventNotFoundException, NoSeatsAvailableException, DuplicateRegistrationException	Регистрирует пользователя
calculatePrice	BigDecimal	eventId: Long, userId: Long,	EventNotFoundException	Рассчитывает стоимость с учётом стратегии

Метод	Возвращаемый тип	Параметры	Исключения	Описание
		quantity: int		
setPricingStrategy	void	strategy: PricingStrategy	—	Устанавливает стратегию ценообразования

4. МЕТОДИКА ВЫПОЛНЕНИЯ

Шаг 1: Анализ исходных материалов

Откройте спецификации прецедентов (ЛР10)

Откройте диаграммы последовательности (ЛР11)

Откройте реализованные паттерны (ЛР12)

Выпишите все классы, участвующие в реализации

Шаг 2: Выделение классов по слоям

Определите классы Entity (JPA-сущности)

Определите классы Foundation (репозитории)

Определите классы Mediator (сервисы, DTO, исключения, паттерны)

Определите классы Control (контроллеры, Request/Response

Шаг 3: Уточнение атрибутов и методов

Для каждого класса определите атрибуты с типами данных

Определите методы с полными сигнатурами

Укажите видимость (+, -, #)

Укажите исключения, которые могут быть выброшены

Шаг 4: Определение отношений

Определите ассоциации между классами (---)

Определите агрегации (◇---) и композиции (◆---)

Определите наследование (▷---) и реализацию (◁---)

Укажите кратности (1, 0.., *1*..)

Шаг 5: Построение диаграммы

Используйте PlantUML или Draw.io

Сгруппируйте классы по пакетам/слоям

Нанесите все классы и связи

Добавьте стереотипы (<<Entity>>, <<Service>> и т.д.)

5. ВАРИАНТЫ ДЛЯ ИНДИВИДУАЛЬНОГО ВЫПОЛНЕНИЯ

Варианты соответствуют проектам из предыдущих лабораторных работ.

	Проект	Основные классы (минимум)
	Система управления складом	Product, Warehouse, StockMovement, Supplier, Category, ProductService, StockService
	Платформа аренды оборудования	Equipment, Rental, Customer, Maintenance, Category, RentalService

	Проект	Основные классы (минимум)
	Система управления парком автомобилей	Vehicle, Driver, Trip, Maintenance, Fueling, TripService
	Система дистанционного обучения	Course, Lesson, Student, Enrollment, Progress, CourseService
	Платформа онлайн-курсов	Course, Instructor, Student, Video, Quiz, Certificate, CourseService
	Система записи на кружки	Circle, Schedule, Student, Teacher, Attendance, CircleService
	Система медицинских назначений	Patient, Doctor, Appointment, Prescription, Diagnosis, AppointmentService
	Платформа записи на процедуры	Patient, Procedure, Equipment, Schedule, Result, ProcedureService
	Система мониторинга здоровья	Patient, Measurement, Alert, Device, Doctor, MeasurementService
0	Система доставки еды	Restaurant, Dish, Order, Courier, Customer, Payment, OrderService

	Проект	Основные классы (минимум)
1	Платформа бронирования услуг	Salon, Master, Service, Client, Booking, Review, BookingService
2	Система проката инвентаря	Item, Rental, Customer, Category, Maintenance, RentalService
3	Система производственных заказов	Order, Product, Operation, Worker, Equipment, OrderService
4	Платформа логистики	Shipment, Route, Vehicle, Driver, Tracking, Customer, ShipmentService
5	Система управления проектами	Project, Task, Worker, Material, Equipment, Report, ProjectService

6. КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Чем отличается Domain Model от Design Class Diagram?
2. Какие стереотипы классов используются в PCMEF?
3. Как на диаграмме классов обозначаются ассоциация, агрегация, композиция?
4. Что означает кратность 0..*?
5. Как обозначается наследование на диаграмме классов?
6. Какие классы относятся к слою Mediator? Foundation? Control?

7. Почему в Design Class Diagram нужно указывать полные сигнатуры методов?
8. Как показать зависимость (dependency) на диаграмме классов?
9. Что такое <<interface>> и как он обозначается?
10. Как показать, что класс реализует интерфейс?
11. Как на диаграмме классов показать внедрение зависимостей через конструктор?
12. Как показать generic-типы (например, List<Event>)?
13. Как показать, что метод выбрасывает исключение?
14. Как на диаграмме классов показать паттерн Strategy?
15. Как на диаграмме классов показать паттерн Observer?

7. ТРЕБОВАНИЯ К ОТЧЕТУ

Структура отчета

1. Титульный лист
2. Содержание
3. Введение (цель работы, краткое описание проекта)
4. Анализ исходных материалов (ЛР10-ЛР12)
5. Диаграмма классов проектирования (PlantUML)
   5.1. Entity слой
   5.2. Foundation слой (репозитории)
   5.3. Mediator слой (сервисы, DTO, исключения, паттерны)
   5.4. Control слой (контроллеры)
6. Спецификация ключевых классов (таблица)
   6.1. Классы Entity (атрибуты, методы)
   6.2. Классы Mediator (атрибуты, методы)
   6.3. Классы Control (атрибуты, методы)
7. Обоснование отношений (почему агрегация, а не композиция)
8. Заключение (выводы, связь с реализацией)

9. Список использованных источников

10. Приложения (исходный код PlantUML)

8. КРИТЕРИИ ОЦЕНИВАНИЯ

Критерий	Мак с. балл	Описание
Полнота классов	4	Все необходимые классы присутствуют
Корректность атрибутов	4	Типы данных, видимость
Корректность методов	5	Полные сигнатуры, параметры, исключения
Корректность отношений	4	Ассоциации, агрегации, композиции, наследование, кратности
Стереотипы и слои	3	Правильное распределение по слоям PCMEF
Связь с ЛР10-ЛР12	3	Учтены спецификации, диаграммы, паттерны
Оформление отчета	2	Соответствие требованиям
Ответы на вопросы	3	Понимание материала

Критерий	Мак с. балл	Описание
ИТОГО	28	

Лабораторная работа №14

JPA — связи @OneToMany, @ManyToOne

1. Цель и задачи работы

Цель: Освоить методику реализации объектно-реляционного отображения (ORM) связей между сущностями в JPA/Hibernate, в частности:

@ManyToOne (многие-к-одному)

@OneToMany (один-ко-многим)

@OneToOne (один-к-одному)

на основе спроектированной диаграммы классов из лабораторной работы №13.

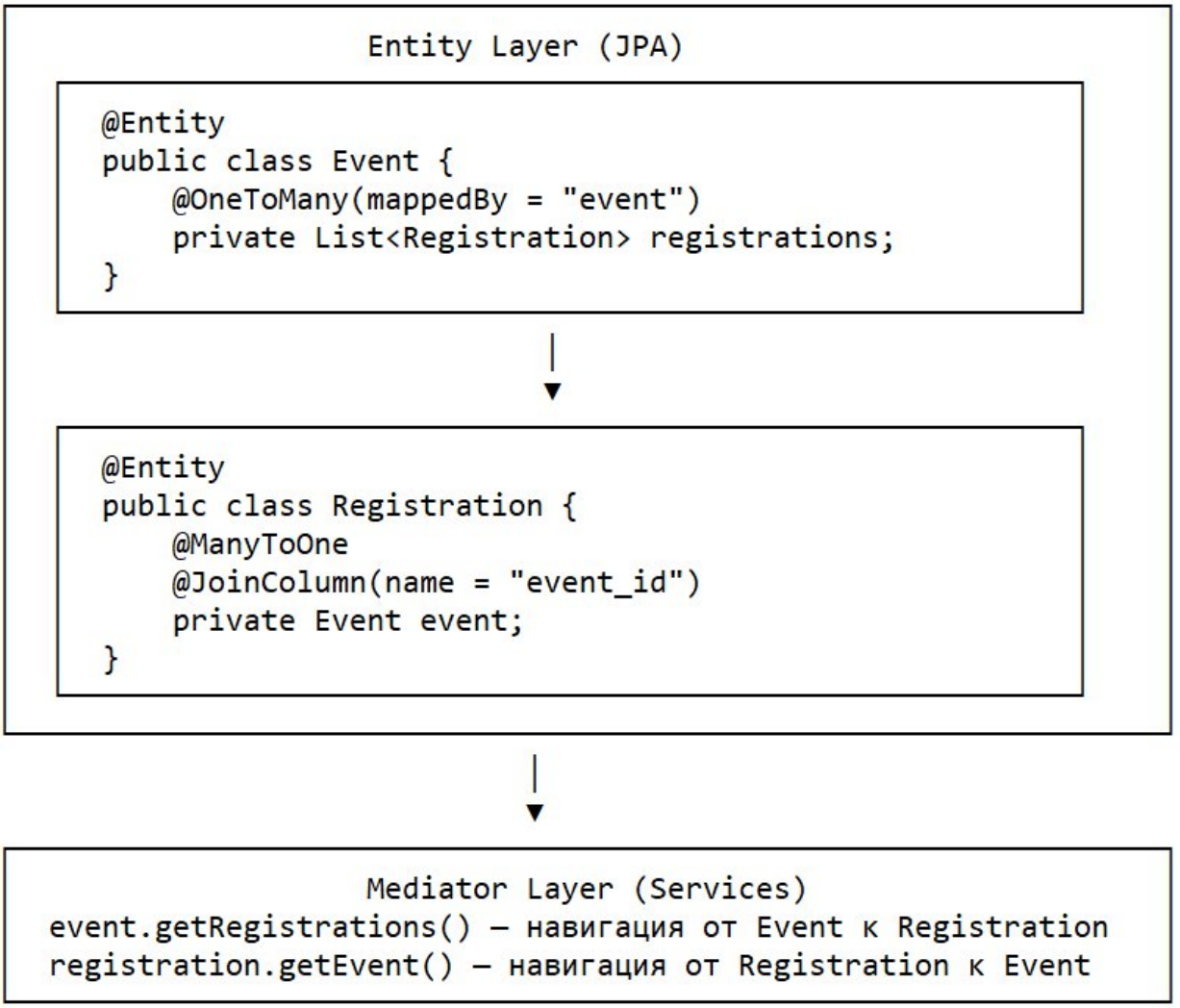
Задачи

1. Изучить теоретические основы связей между сущностями в JPA
2. Реализовать связь @ManyToOne (многие-к-одному)
3. Реализовать двунаправленную связь @OneToMany / @ManyToOne
4. Настроить каскадные операции (CascadeType) и типы загрузки (FetchType)
5. Реализовать связь @OneToOne (один-к-одному)
6. Написать тесты для проверки корректности связей

2. ТЕОРЕТИЧЕСКОЕ ОБОСНОВАНИЕ

2.1. Место JPA-связей в архитектуре PCMEF

JPA-связи реализуются в Entity-слое (JPA-сущности) и используются Mediator-слоем (сервисами) для навигации между связанными объектами.



2.2. Типы связей между сущностями

Тип связи	Аннотация	Описание	Пример
Один-к-одному	@OneToOne	Одна запись связана с одной записью в другой таблице	User ↔ Passport

Тип связи	Аннотация	Описание	Пример
Один-ко-многим	@OneToMany	Одна запись связана с несколькими записями в другой таблице	Event → Registration
Многие-к-одному	@ManyToOne	Несколько записей связаны с одной записью в другой таблице	Registration → Event
Многие-ко-многим	@ManyToMany	Несколько записей связаны с несколькими записями	Student ↔ Course

2.3. Направленность связей

Одностороння связь (Unidirectional):

```
// Только Event знает о Registration
@Entity
public class Event {
    @OneToMany
    @JoinColumn(name = "event_id")
    private List<Registration> registrations;
}
```

Двунаправленная связь (Bidirectional):

```
// Event знает о Registration
@Entity
```

```
public class Event {
    @OneToMany(mappedBy = "event")
    private List<Registration> registrations;
}

// Registration знает о Event
@Entity
public class Registration {
    @ManyToOne
    @JoinColumn(name = "event_id")
    private Event event;
}
```

2.4. Основные аннотации для связей

Аннотация	Описание	Атрибуты
@OneToMany	Связь один-ко-многим	mappedBy, cascade, fetch, orphanRemoval
@ManyToOne	Связь многие-к-одному	cascade, fetch, optional
@OneToOne	Связь один-к-одному	mappedBy, cascade, fetch, optional

Аннотация	Описание	Атрибуты
@ManyToMany	Связь многие-ко-многим	mappedBy, cascade, fetch
@JoinColumn	Настройка внешнего ключа	name, referencedColumnName, nullable, unique
@JoinTable	Настройка связующей таблицы	name, joinColumns, inverseJoinColumns

2.5. Типы загрузки (FetchType)

Тип	Описание	Производительность	Когда использовать
LAZY (ленивая)	Данные загружаются при первом обращении	Высокая	По умолчанию для @OneToMany, @ManyToMany
EAGER (жадная)	Данные загружают	Низкая (N+1 проблема)	Только для @ManyToOne, @OneToOne если данные всегда нужны

Тип	Описание	Производительность	Когда использовать
	ся сразу вместе с родителем		

Рекомендация: Использовать FetchType.LAZY для всех связей, кроме случаев, когда связанные данные нужны всегда.

2.6. Каскадные операции (CascadeType)

Тип	Описание	Пример
PERSIST	Сохранять связанные объекты	При сохранении Event сохраняются Registration
MERGE	Обновлять связанные объекты	При обновлении Event обновляются Registration
REMOVE	Удалять связанные объекты	При удалении Event удаляются Registration
REFRESH	Обновлять состояние связанных объектов	—
DETACH	Отсоединять связанные объекты	—

Тип	Описание	Пример
ALL	Все операции	cascade = CascadeType.ALL

2.7. Проблема N+1 запросов

Проблема: При загрузке коллекции с FetchType.EAGER Hibernate может выполнить N+1 запросов (1 запрос для родителя + N запросов для каждого ребёнка).

Решение: Использовать FetchType.LAZY и загружать данные через JOIN FETCH в JPQL:

```
@Query("SELECT e FROM Event e LEFT JOIN FETCH e.registrations WHERE e.id = :id")
Optional<Event> findByIdWithRegistrations(@Param("id") Long id);
```

2.8. Схема базы данных для связей

Тип связи	Схема БД
@OneToMany / @ManyToOne	Внешний ключ в таблице «многие»
@OneToOne	Внешний ключ + UNIQUE в одной из таблиц
@ManyToMany	Связующая таблица (два внешних ключа)

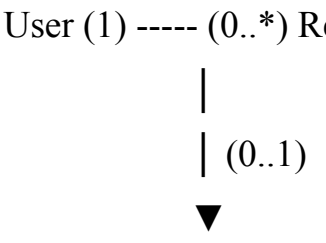
```
-- Event → Registration (1:N)
```

```
CREATE TABLE event (
    id BIGSERIAL PRIMARY KEY
);

CREATE TABLE registration (
    id BIGSERIAL PRIMARY KEY,
    event_id BIGINT REFERENCES event(id)  -- внешний ключ
);
```

3. ПРИМЕР ВЫПОЛНЕНИЯ (СКВОЗНОЙ ПРОЕКТ)

Проект: «Система управления мероприятиями» (Event Management System)

3.1. Исходная диаграмма классов (из ЛР13)



3.2. Реализация связи @ManyToOne (Unidirectional)

Registration → Event (многие-к-одному):

```
package ru.edu.project.entity;

import jakarta.persistence.*;
import lombok.*;
import java.time.LocalDateTime;

@Entity
```

```java
@Table(name = "registrations")
@Getter
@Setter
@NoArgsConstructor
@AllArgsConstructor
@Builder
public class Registration {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(name = "registration_date", nullable = false)
    @Builder.Default
    private LocalDateTime registrationDate = LocalDateTime.now();

    @Column(nullable = false, length = 20)
    @Builder.Default
    private String status = "PENDING";

    // Many-to-One: много регистраций → одно мероприятие
    @ManyToOne(fetch = FetchType.LAZY)  // LAZY — оптимально
    @JoinColumn(name = "event_id", nullable = false)
    private Event event;

    // Many-to-One: много регистраций → один пользователь
    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "user_id", nullable = false)
    private User user;
```

```
    // One-to-One: одна регистрация → один платёж
    @OneToOne(mappedBy = "registration", cascade = CascadeType.ALL,
            fetch = FetchType.LAZY)
    private Payment payment;
}
```

3.3. Реализация двунаправленной связи @OneToMany / @ManyToOne

Event ← Registration (один-ко-многим + многие-к-одному):

```
package ru.edu.project.entity;

import jakarta.persistence.*;
import lombok.*;
import java.math.BigDecimal;
import java.time.LocalDateTime;
import java.util.ArrayList;
import java.util.List;

@Entity
@Table(name = "events")
@Getter
@Setter
@NoArgsConstructor
@AllArgsConstructor
@Builder
public class Event {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
```

```
    private Long id;

    @Column(nullable = false, length = 200)
    private String title;

    @Column(columnDefinition = "TEXT")
    private String description;

    @Column(name = "start_date", nullable = false)
    private LocalDateTime startDate;

    @Column(name = "end_date", nullable = false)
    private LocalDateTime endDate;

    @Column(name = "max_participants", nullable = false)
    private Integer maxParticipants;

    @Column(nullable = false, length = 20)
    @Builder.Default
    private String status = "DRAFT";

    @Column(name = "base_price")
    private BigDecimal basePrice;

    // Many-to-One: событие → категория
    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "category_id")
    private Category category;
```

```
    // Many-to-One: событие → создатель
    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "created_by", nullable = false)
    private User createdBy;

    // One-to-Many: одно событие → много регистраций
    @OneToMany(mappedBy = "event",          // поле в Registration
               cascade = CascadeType.ALL,                    // каскадное
сохранение/удаление
               fetch = FetchType.LAZY,      // ленивая загрузка
               orphanRemoval = true)        // удаление "сирот"
    @Builder.Default
    private List<Registration> registrations = new ArrayList<>();

    // Вспомогательные методы для управления двунаправленной связью
    public void addRegistration(Registration registration) {
        registrations.add(registration);
        registration.setEvent(this);
    }

    public void removeRegistration(Registration registration) {
        registrations.remove(registration);
        registration.setEvent(null);
    }

    // Бизнес-методы
    public boolean checkAvailability() {
        return registrations.size() < maxParticipants;
    }
```

```
    public int getAvailableSeats() {
        return maxParticipants - registrations.size();
    }

    public Registration registerUser(User user) {
        if (!checkAvailability()) {
            throw new IllegalStateException("No seats available");
        }

        Registration registration = Registration.builder()
                .event(this)
                .user(user)
                .registrationDate(LocalDateTime.now())
                .status("PENDING")
                .build();

        addRegistration(registration);  // синхронизация обеих сторон
        return registration;
    }
}
```

3.4. Реализация связи @OneToOne (один-к-одному)

Registration → Payment (один-к-одному):

```
package ru.edu.project.entity;

import jakarta.persistence.*;
import lombok.*;
import java.math.BigDecimal;
```

```java
import java.time.LocalDateTime;

@Entity
@Table(name = "payments")
@Getter
@Setter
@NoArgsConstructor
@AllArgsConstructor
@Builder
public class Payment {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(nullable = false, precision = 10, scale = 2)
    private BigDecimal amount;

    @Column(name = "payment_date", nullable = false)
    @Builder.Default
    private LocalDateTime paymentDate = LocalDateTime.now();

    @Column(nullable = false, length = 50)
    private String method;

    @Column(name = "transaction_id", length = 100)
    private String transactionId;

    // One-to-One: один платёж → одна регистрация (владелец связи)
```

```
    @OneToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "registration_id", nullable = false, unique = true)
    private Registration registration;
}
```

Registration → Payment (другая сторона):

```
// В классе Registration (продолжение)
@OneToOne(mappedBy = "registration",  // поле в Payment
        cascade = CascadeType.ALL,   // каскадное сохранение платежа
        fetch = FetchType.LAZY)
private Payment payment;

// Вспомогательный метод
public void addPayment(Payment payment) {
    this.payment = payment;
    payment.setRegistration(this);
}
```

3.5. Реализация связи @ManyToOne с Optional=false

Registration → User:

```
// В классе Registration
@ManyToOne(fetch = FetchType.LAZY, optional = false)   // optional = false → NOT NULL
@JoinColumn(name = "user_id", nullable = false)
private User user;
```

3.6. Пример использования связей в сервисе

```
package ru.edu.project.mediator.services;

import org.springframework.stereotype.Service;
```

```
import org springframework transaction annotation Transactional;

@Service
public class RegistrationService {

    private final EventRepository eventRepository;
    private final UserRepository userRepository;
    private final RegistrationRepository registrationRepository;

    @Transactional
    public RegistrationDto registerUserForEvent(Long userId, Long eventId)
{
        // Загрузка с JOIN FETCH для оптимизации
        Event event = eventRepository.findByIdWithRegistrations(eventId)
            .orElseThrow(() -> new EventNotFoundException(eventId));

        User user = userRepository.findById(userId)
            .orElseThrow(() -> new UserNotFoundException(userId));

        // Использование бизнес-метода сущности (который использует
связь)
        Registration registration = event.registerUser(user);

        // Сохранение (каскадное, благодаря CascadeType.ALL)
        Registration saved = registrationRepository.save(registration);

        // Навигация по связям
        System.out.println("Мероприятие: " + saved.getEvent().getTitle());
```

```
        System.out.println("Всего регистраций: " +
saved.getEvent().getRegistrations().size());

        return RegistrationDto.fromEntity(saved);
    }
}
```

3.7. Репозиторий с JOIN FETCH для решения N+1 проблемы

```
package ru.edu.project.foundation.repositories;

import ru.edu.project.entity.Event;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.jpa.repository.Query;
import org.springframework.data.repository.query.Param;
import java.util.Optional;

public interface EventRepository extends JpaRepository<Event, Long> {

    // Без JOIN FETCH — будет N+1 проблема
    Optional<Event> findById(Long id);

    // С JOIN FETCH — одна оптимизированная загрузка
    @Query("SELECT e FROM Event e LEFT JOIN FETCH e.registrations
WHERE e.id = :id")
    Optional<Event> findByIdWithRegistrations(@Param("id") Long id);

    @Query("SELECT e FROM Event e LEFT JOIN FETCH e.registrations r
LEFT JOIN FETCH r.user WHERE e.id = :id")
```

```
    Optional<Event> findByIdWithRegistrationsAndUsers(@Param("id")
Long id);
}
```

3.8. Тестирование связей

```
package ru.edu.project.entity;

import org.junit.jupiter.api.Test;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.context.SpringBootTest;
import org.springframework.transaction.annotation.Transactional;
import ru.edu.project.foundation.repositories.EventRepository;
import ru.edu.project.foundation.repositories.UserRepository;

import java.math.BigDecimal;
import java.time.LocalDateTime;

import static org.junit.jupiter.api.Assertions.*;

@SpringBootTest
@Transactional
class RelationshipTest {

    @Autowired
    private EventRepository eventRepository;

    @Autowired
    private UserRepository userRepository;

    @Test
```

```
void testOneToManyRelationship() {
    // Создание пользователя
    User user = User.builder()
            .email("test@example.com")
            .passwordHash("hashed")
            .name("Test User")
            .build();
    user = userRepository.save(user);

    // Создание мероприятия
    Event event = Event.builder()
            .title("Test Event")
            .startDate(LocalDateTime.now().plusDays(7))
            .endDate(LocalDateTime.now().plusDays(8))
            .maxParticipants(10)
            .createdBy(user)
            .build();

    // Добавление регистраций через связь
    Registration reg1 = event.registerUser(user);
    Registration reg2 = event.registerUser(user);

    event = eventRepository.save(event);

    // Проверка связей
    assertNotNull(event.getId());
    assertEquals(2, event.getRegistrations().size());
    assertEquals(event, event.getRegistrations().get(0).getEvent());
    assertEquals(user, event.getRegistrations().get(0).getUser());
```

```
}

@Test
void testManyToOneRelationship() {
    User user = User.builder()
            .email("user@example.com")
            .passwordHash("hash")
            .name("User")
            .build();
    user = userRepository.save(user);

    Event event = Event.builder()
            .title("Event")
            .startDate(LocalDateTime.now().plusDays(7))
            .endDate(LocalDateTime.now().plusDays(8))
            .maxParticipants(10)
            .createdBy(user)
            .build();
    event = eventRepository.save(event);

    Registration registration = Registration.builder()
            .event(event)
            .user(user)
            .build();

    // Проверка @ManyToOne
    assertEquals(event, registration.getEvent());
    assertEquals(user, registration.getUser());
}
```

```
    @Test
    void testCascadeRemove() {
        User user = User.builder()
                .email("cascade@example.com")
                .passwordHash("hash")
                .name("Cascade Test")
                .build();
        user = userRepository.save(user);

        Event event = Event.builder()
                .title("Cascade Event")
                .startDate(LocalDateTime.now().plusDays(7))
                .endDate(LocalDateTime.now().plusDays(8))
                .maxParticipants(10)
                .createdBy(user)
                .build();

        Registration reg1 = event.registerUser(user);
        Registration reg2 = event.registerUser(user);

        event = eventRepository.save(event);
        Long eventId = event.getId();

        // При удалении Event, все Registration удаляются благодаря CascadeType.ALL
        eventRepository.delete(event);

        assertFalse(eventRepository.findById(eventId).isPresent());
```

}

}

4. МЕТОДИКА ВЫПОЛНЕНИЯ

Шаг 1: Анализ диаграммы классов (ЛР13)

Определите все связи между сущностями (1:1, 1:N, M:N)

Для каждой связи определите направленность

Определите, какие связи требуют каскадных операций

Таблица связей для заполнения:

Связь	ип	Одна сторона	Многие сторона	Каскад	Fetch
Event → Registration	:N	Event	Registration	ALL	LAZY
Registration → Event	:1	—	Registration	—	LAZY
User → Registration	:N	User	Registration	ALL	LAZY
Registration → User	:1	—	Registration	—	LAZY
Registration → Payment	:1	Registration	Payment	ALL	LAZY

Шаг 2: Реализация @ManyToOne

Добавьте аннотацию @ManyToOne на стороне "многие"

Добавьте @JoinColumn с правильным именем внешнего ключа

Укажите fetch = FetchType.LAZY

Укажите optional = false если поле обязательное

Шаг 3: Реализация @OneToMany

Добавьте аннотацию @OneToMany на стороне "один"

Укажите mappedBy — название поля на противоположной стороне

Укажите cascade = CascadeType.ALL

Укажите fetch = FetchType.LAZY

Добавьте вспомогательные методы (addXxx, removeXxx)

Шаг 4: Реализация @OneToOne

Выберите сторону-владельца связи (где будет внешний ключ)

На стороне-владельце: @OneToOne + @JoinColumn

На противоположной стороне: @OneToOne(mappedBy = "...")

Шаг 5: Оптимизация с JOIN FETCH (1 час)

Добавьте в репозитории методы с JOIN FETCH

Используйте их в сервисах для загрузки связанных данных

5. КОНТРОЛЬНЫЕ ВОПРОСЫ

1. В чем разница между @OneToMany и @ManyToOne?
2. Что означает mappedBy в @OneToMany?
3. Какие значения может принимать FetchType и в чем их отличие?
4. Что такое cascade = CascadeType.ALL?
5. В чем разница между单向 и双向 связью?
6. Что такое проблема N+1 запросов и как её решить?
7. Что делает orphanRemoval = true?
8. Как правильно синхронизировать обе стороны двунаправленной связи?

9. Как реализовать связь @ManyToMany с дополнительными атрибутами?
10. В чем разница между cascade = CascadeType.REMOVE и orphanRemoval = true?
11. Как загрузить связанные сущности с пагинацией?

6. ТРЕБОВАНИЯ К ОТЧЕТУ

Структура отчета

1. Титульный лист
2. Содержание
3. Введение (цель работы, краткое описание проекта)
4. Анализ связей (диаграмма классов, таблица связей)
5. Реализация @ManyToOne (код, пояснения)
6. Реализация @OneToMany (код, пояснения)
7. Реализация @OneToOne (код, пояснения)
8. Реализация вспомогательных методов (add/remove)
9. Оптимизация с JOIN FETCH (репозитории, тесты)
10. Тестирование связей (листинг тестов)
11. Заключение
12. Список литературы
13. Приложения (полный код)

Лабораторная работа №15

Реализация Mediator-слоя (бизнес-логика)

1. Цель и задачи работы

Цель: Освоить методику создания слоя бизнес-логики (Mediator) в архитектуре PCMEF с использованием Spring Framework, реализовав

сервисы, которые координируют работу репозиториев и сущностей, управляют транзакциями и реализуют бизнес-правила предметной области.

Задачи:

1. Настроить Spring-контекст для внедрения зависимостей
2. Создать интерфейсы сервисов (I*Service) для всех бизнес-операций
3. Реализовать сервисы (Impl) с бизнес-логикой
4. Настроить управление транзакциями (@Transactional)
5. Реализовать обработку бизнес-исключений
6. Написать модульные тесты для сервисов с использованием моков репозиториев

2. ТЕОРЕТИЧЕСКОЕ ОБОСНОВАНИЕ

2.1. Место Mediator-слоя в архитектуре PCMEF

В архитектуре PCMEF Mediator-слой (Посредник) отвечает за бизнес-логику и координацию между Control-слоем и Foundation-слоем.

Control (REST API / Контроллеры) ↓
Mediator (сервисы) ← **МЫ ЗДЕСЬ** ↓
Entity (JPA-сущности) ↓
Foundation (репозитории)

Ответственность Mediator-слоя:

1. Реализация бизнес-правил
2. Управление транзакциями
3. Координация нескольких репозиториев
4. Валидация бизнес-условий

5. Выброс бизнес-исключений

2.2. Структура Mediator-слоя

```
mediator/
├── interfaces/
│   ├── IUserService.java
│   ├── IEventService.java
│   └── IRegistrationService.java
├── services/
│   ├── UserServiceImpl.java
│   ├── EventServiceImpl.java
│   └── RegistrationServiceImpl.java
├── exceptions/
│   ├── BusinessException.java
│   ├── UserNotFoundException.java
│   ├── EventNotFoundException.java
│   └── NoSeatsAvailableException.java
└── dto/
    ├── UserDto.java
    ├── EventDto.java
    └── RegistrationDto.java
```

2.3. Внедрение зависимостей (Dependency Injection)

Внедрение через конструктор (рекомендуемый способ):

```
@Service
public class UserServiceImpl implements IUserService {

    private final UserRepository userRepository;
```

```
    public UserServiceImpl(UserRepository userRepository) {
        this.userRepository = userRepository;
    }
}
```

Преимущества внедрения через конструктор:

- Поля можно сделать final
- Упрощает тестирование
- Позволяет обнаружить циклические зависимости

2.4. Управление транзакциями

Аннотация @Transactional:

```
@Service
public class RegistrationServiceImpl implements IRegistrationService {

    @Override
    @Transactional
    public Registration registerUserForEvent(Long userId, Long eventId) {
        // несколько операций с БД в рамках одной транзакции
        User user = userRepository.findById(userId)
            .orElseThrow(() -> new UserNotFoundException(...));
        Event event = eventRepository.findById(eventId)
            .orElseThrow(() -> new EventNotFoundException(...));

        Registration registration = event.registerUser(user);
        return registrationRepository.save(registration);
    }
}
```

Параметры @Transactional:

Параметр	Описание	Пример
propagation	Распространение транзакции	REQUIRED, REQUIRES_NEW, SUPPORTS
isolation	Уровень изоляции	READ_COMMITTED, REPEATABLE_READ
rollbackFor	При каких исключениях откат	rollbackFor = BusinessException.class
readOnly	Только чтение (оптимизация)	readOnly = true

2.5. Бизнес-исключения

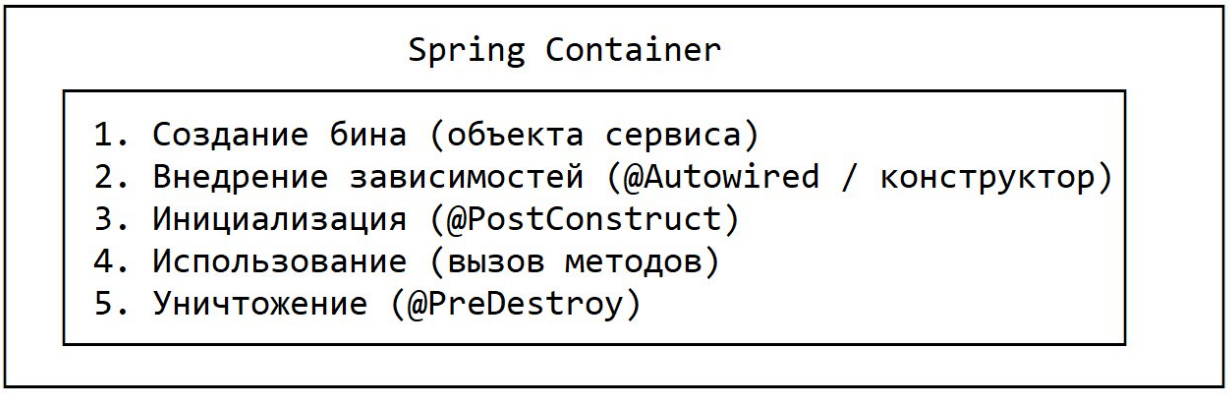
```
public class BusinessException extends RuntimeException {
    public BusinessException(String message) {
        super(message);
    }
}

public class UserNotFoundException extends BusinessException {
    public UserNotFoundException(Long userId) {
        super("User not found with id: " + userId);
    }
}
```

2.6. DTO (Data Transfer Object)

DTO используются для передачи данных между слоями, чтобы не экспортировать Entity напрямую.

```java
public class UserDto {
    private Long id;
    private String email;
    private String name;
    private String role;

    // Конструкторы, геттеры, сеттеры

    public static UserDto fromEntity(User user) {
        return new UserDto(user.getId(), user.getEmail(),
                           user.getName(), user.getRole());
    }
}
```

2.7. Жизненный цикл сервиса



3. ПРИМЕР ВЫПОЛНЕНИЯ (СКВОЗНОЙ ПРОЕКТ)

Проект: «Система управления мероприятиями» (Event Management System)

3.1. Добавление зависимостей Spring в pom.xml

```
    <dependencies>
        <!-- Существующие зависимости (JPA, Hibernate, PostgreSQL, JUnit, Mockito) -->

        <!-- Spring Context (для Dependency Injection) -->
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-context</artifactId>
            <version>6.1.0</version>
        </dependency>

        <!-- Spring Transactions -->
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-tx</artifactId>
            <version>6.1.0</version>
        </dependency>

        <!-- Spring ORM (интеграция с Hibernate) -->
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-orm</artifactId>
            <version>6.1.0</version>
        </dependency>

        <!-- Spring Test -->
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-test</artifactId>
```

```
            <version>6.1.0</version>
            <scope>test</scope>
        </dependency>
    </dependencies>
```

3.2. Настройка Spring-контекста (Java-конфигурация)

AppConfig.java:

```
package ru.edu.project.config;

import com.zaxxer.hikari.HikariConfig;
import com.zaxxer.hikari.HikariDataSource;
import jakarta.persistence.EntityManagerFactory;
import org.springframework.context.annotation.*;
import org.springframework.orm.jpa.JpaTransactionManager;
import
org.springframework.orm.jpa.LocalContainerEntityManagerFactoryBean;
import org.springframework.orm.jpa.vendor.HibernateJpaVendorAdapter;
import org.springframework.transaction.PlatformTransactionManager;
import
org.springframework.transaction.annotation.EnableTransactionManagement;

import javax.sql.DataSource;
import java.util.Properties;

@Configuration
@ComponentScan(basePackages = "ru.edu.project")
@EnableTransactionManagement
public class AppConfig {
```

```
    @Bean
    public DataSource dataSource() {
        HikariConfig config = new HikariConfig();
        config.setJdbcUrl("jdbc:postgresql://localhost:5432/event_db");
        config.setUsername("postgres");
        config.setPassword("password");
        config.setDriverClassName("org.postgresql.Driver");
        config.setMaximumPoolSize(10);
        return new HikariDataSource(config);
    }

    @Bean
    public LocalContainerEntityManagerFactoryBean entityManagerFactory(DataSource dataSource) {
        LocalContainerEntityManagerFactoryBean emf = new LocalContainerEntityManagerFactoryBean();
        emf.setDataSource(dataSource);
        emf.setPackagesToScan("ru.edu.project.entity");

        HibernateJpaVendorAdapter vendorAdapter = new HibernateJpaVendorAdapter();
        vendorAdapter.setShowSql(true);
        emf.setJpaVendorAdapter(vendorAdapter);

        Properties properties = new Properties();
        properties.setProperty("hibernate.dialect", "org.hibernate.dialect.PostgreSQLDialect");
        properties.setProperty("hibernate.hbm2ddl.auto", "validate");
        emf.setJpaProperties(properties);
```

```
        return emf;
    }

    @Bean
    public PlatformTransactionManager transactionManager(EntityManagerFactory entityManagerFactory) {
        return new JpaTransactionManager(entityManagerFactory);
    }
}
```

3.3. Реализация репозиториев с поддержкой Spring

UserRepository.java:

```
package ru.edu.project.foundation.repositories;

import ru.edu.project.entity.User;
import java.util.List;
import java.util.Optional;

public interface UserRepository {
    Optional<User> findById(Long id);
    List<User> findAll();
    Optional<User> findByEmail(String email);
    List<User> findByRole(String role);
    User save(User user);
    User update(User user);
    void delete(User user);
    void deleteById(Long id);
    boolean existsByEmail(String email);
```

```
}
```

UserRepositoryImpl.java (с Spring-аннотациями):

```
package ru.edu.project.foundation.repositories;

import jakarta.persistence.EntityManager;
import jakarta.persistence.PersistenceContext;
import jakarta.persistence.TypedQuery;
import org.springframework.stereotype.Repository;
import org.springframework.transaction.annotation.Transactional;
import ru.edu.project.entity.User;

import java.util.List;
import java.util.Optional;

@Repository
@Transactional
public class UserRepositoryImpl implements UserRepository {

    @PersistenceContext
    private EntityManager entityManager;

    @Override
    public Optional<User> findById(Long id) {
        return Optional.ofNullable(entityManager.find(User.class, id));
    }

    @Override
    public List<User> findAll() {
```

```
    TypedQuery<User> query = entityManager.createQuery(
        "SELECT u FROM User u", User.class);
    return query.getResultList();
}

@Override
public Optional<User> findByEmail(String email) {
    TypedQuery<User> query = entityManager.createQuery(
        "SELECT u FROM User u WHERE u.email = :email", User.class);
    query.setParameter("email", email);
    return query.getResultStream().findFirst();
}

@Override
public List<User> findByRole(String role) {
    TypedQuery<User> query = entityManager.createQuery(
        "SELECT u FROM User u WHERE u.role = :role", User.class);
    query.setParameter("role", role);
    return query.getResultList();
}

@Override
public User save(User user) {
    entityManager.persist(user);
    return user;
}

@Override
public User update(User user) {
```

```
        return entityManager.merge(user);
    }

    @Override
    public void delete(User user) {
        entityManager.remove(entityManager.contains(user) ? user : entityManager.merge(user));
    }

    @Override
    public void deleteById(Long id) {
        findById(id).ifPresent(this::delete);
    }

    @Override
    public boolean existsByEmail(String email) {
        TypedQuery<Long> query = entityManager.createQuery(
            "SELECT COUNT(u) FROM User u WHERE u.email = :email", Long.class);
        query.setParameter("email", email);
        return query.getSingleResult() > 0;
    }
}
```

3.4. Создание бизнес-исключений

BusinessException.java:

```
package ru.edu.project.mediator.exceptions;
```

```
public class BusinessException extends RuntimeException {
    public BusinessException(String message) {
        super(message);
    }

    public BusinessException(String message, Throwable cause) {
        super(message, cause);
    }
}
```

UserNotFoundException.java:

```
package ru.edu.project.mediator.exceptions;

public class UserNotFoundException extends BusinessException {
    public UserNotFoundException(Long userId) {
        super("User not found with id: " + userId);
    }

    public UserNotFoundException(String email) {
        super("User not found with email: " + email);
    }
}
```

EventNotFoundException.java:

```
package ru.edu.project.mediator.exceptions;

public class EventNotFoundException extends BusinessException {
    public EventNotFoundException(Long eventId) {
```

```
        super("Event not found with id: " + eventId);
    }
}
```

NoSeatsAvailableException.java:

```
package ru.edu.project.mediator.exceptions;

public class NoSeatsAvailableException extends BusinessException {
    public NoSeatsAvailableException(Long eventId) {
        super("No seats available for event: " + eventId);
    }
}
```

DuplicateRegistrationException.java:

```
package ru.edu.project.mediator.exceptions;

public class DuplicateRegistrationException extends BusinessException {
    public DuplicateRegistrationException(Long userId, Long eventId) {
        super("User " + userId + " is already registered for event " + eventId);
    }
}
```

3.5. Создание DTO

UserDto.java:

```
package ru.edu.project.mediator.dto;

import ru.edu.project.entity.User;
import java.util.Objects;

public class UserDto {
```

```
    private Long id;
    private String email;
    private String name;
    private String role;

    public UserDto() {}

    public UserDto(Long id, String email, String name, String role) {
        this.id = id;
        this.email = email;
        this.name = name;
        this.role = role;
    }

    public static UserDto fromEntity(User user) {
        if (user == null) return null;
        return new UserDto(user.getId(), user.getEmail(), user.getName(),
user.getRole());
    }

    // Getters and Setters
    public Long getId() { return id; }
    public void setId(Long id) { this.id = id; }

    public String getEmail() { return email; }
    public void setEmail(String email) { this.email = email; }

    public String getName() { return name; }
    public void setName(String name) { this.name = name; }
```

```
    public String getRole() { return role; }
    public void setRole(String role) { this.role = role; }

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;
        UserDto userDto = (UserDto) o;
        return Objects.equals(id, userDto.id);
    }

    @Override
    public int hashCode() {
        return Objects.hash(id);
    }

    @Override
    public String toString() {
        return "UserDto{" +
                "id=" + id +
                ", email='" + email + '\'' +
                ", name='" + name + '\'' +
                ", role='" + role + '\'' +
                '}';
    }
}
```

EventDto.java:

```
package ru.edu.project.mediator.dto;
```

```
import ru.edu.project.entity.Event;
import java.time.LocalDateTime;
import java.util.Objects;

public class EventDto {
    private Long id;
    private String title;
    private String description;
    private LocalDateTime startDate;
    private LocalDateTime endDate;
    private Integer maxParticipants;
    private Integer registeredCount;
    private Integer availableSeats;
    private String status;
    private String categoryName;
    private String createdBy;

    public EventDto() {}

    public static EventDto fromEntity(Event event) {
        if (event == null) return null;
        EventDto dto = new EventDto();
        dto.setId(event.getId());
        dto.setTitle(event.getTitle());
        dto.setDescription(event.getDescription());
        dto.setStartDate(event.getStartDate());
        dto.setEndDate(event.getEndDate());
        dto.setMaxParticipants(event.getMaxParticipants());
```

```
        dto.setRegisteredCount(event.getRegistrations().size());
        dto.setAvailableSeats(event.getAvailableSeats());
        dto.setStatus(event.getStatus());
        dto.setCategoryName(event.getCategory() != null ? event.getCategory().getName() : null);
        dto.setCreatedBy(event.getCreatedBy() != null ? event.getCreatedBy().getName() : null);
        return dto;
    }

    // Getters and Setters
    public Long getId() { return id; }
    public void setId(Long id) { this.id = id; }

    public String getTitle() { return title; }
    public void setTitle(String title) { this.title = title; }

    public String getDescription() { return description; }
    public void setDescription(String description) { this.description = description; }

    public LocalDateTime getStartDate() { return startDate; }
    public void setStartDate(LocalDateTime startDate) { this.startDate = startDate; }

    public LocalDateTime getEndDate() { return endDate; }
    public void setEndDate(LocalDateTime endDate) { this.endDate = endDate; }
```

```
    public Integer getMaxParticipants() { return maxParticipants; }
    public void setMaxParticipants(Integer maxParticipants) { this.maxParticipants = maxParticipants; }

    public Integer getRegisteredCount() { return registeredCount; }
    public void setRegisteredCount(Integer registeredCount) { this.registeredCount = registeredCount; }

    public Integer getAvailableSeats() { return availableSeats; }
    public void setAvailableSeats(Integer availableSeats) { this.availableSeats = availableSeats; }

    public String getStatus() { return status; }
    public void setStatus(String status) { this.status = status; }

    public String getCategoryName() { return categoryName; }
    public void setCategoryName(String categoryName) { this.categoryName = categoryName; }

    public String getCreatedBy() { return createdBy; }
    public void setCreatedBy(String createdBy) { this.createdBy = createdBy; }
}
```

3.6. Создание интерфейсов сервисов

IUserService.java:

```
package ru.edu.project.mediator.interfaces;

import ru.edu.project.mediator.dto.UserDto;
import java.util.List;
```

```
public interface IUserService {

    /**
     * Получить пользователя по ID
     */
    UserDto getUserById(Long id);

    /**
     * Получить пользователя по email
     */
    UserDto getUserByEmail(String email);

    /**
     * Получить всех пользователей
     */
    List<UserDto> getAllUsers();

    /**
     * Получить пользователей по роли
     */
    List<UserDto> getUsersByRole(String role);

    /**
     * Создать пользователя
     */
    UserDto createUser(String email, String password, String name);

    /**
```

```
    * Обновить пользователя
    */
    UserDto updateUser(Long id, String name, String role);

    /**
    * Удалить пользователя
    */
    void deleteUser(Long id);

    /**
    * Аутентификация пользователя
    */
    UserDto authenticate(String email, String password);
}
```

IEventService.java:

```
package ru.edu.project.mediator.interfaces;

import ru.edu.project.mediator.dto.EventDto;
import java.time.LocalDateTime;
import java.util.List;

public interface IEventService {

    /**
    * Получить мероприятие по ID
    */
    EventDto getEventById(Long id);

    /**
```

```
     * Получить все мероприятия
     */
    List<EventDto> getAllEvents();

    /**
     * Получить опубликованные мероприятия
     */
    List<EventDto> getPublishedEvents();

    /**
     * Получить мероприятия по категории
     */
    List<EventDto> getEventsByCategory(Long categoryId);

    /**
     * Получить мероприятия в диапазоне дат
     */
    List<EventDto> getEventsInDateRange(LocalDateTime start,
LocalDateTime end);

    /**
     * Создать мероприятие
     */
    EventDto createEvent(String title, String description, LocalDateTime
startDate,
                         LocalDateTime endDate, Integer maxParticipants,
                         Long categoryId, Long createdBy);

    /**
```

```
     * Обновить мероприятие
     */
    EventDto updateEvent(Long id, String title, String description,
                         LocalDateTime startDate, LocalDateTime endDate,
                         Integer maxParticipants, Long categoryId);

    /**
     * Опубликовать мероприятие
     */
    void publishEvent(Long id);

    /**
     * Отменить мероприятие
     */
    void cancelEvent(Long id);

    /**
     * Удалить мероприятие
     */
    void deleteEvent(Long id);
}
```

IRegistrationService.java:

```
package ru.edu.project.mediator.interfaces;

import ru.edu.project.mediator.dto.RegistrationDto;
import java.util.List;

public interface IRegistrationService {
```

```
    /**
     * Получить регистрацию по ID
     */
    RegistrationDto getRegistrationById(Long id);

    /**
     * Получить регистрации пользователя
     */
    List<RegistrationDto> getRegistrationsByUser(Long userId);

    /**
     * Получить регистрации на мероприятие
     */
    List<RegistrationDto> getRegistrationsByEvent(Long eventId);

    /**
     * Зарегистрировать пользователя на мероприятие
     */
    RegistrationDto registerUserForEvent(Long userId, Long eventId);

    /**
     * Отменить регистрацию
     */
    void cancelRegistration(Long registrationId);

    /**
     * Подтвердить регистрацию (после оплаты)
     */
    void confirmRegistration(Long registrationId);
```

}

3.7. Реализация сервисов

UserServiceImpl.java:

```
package ru.edu.project.mediator.services;

import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;
import ru.edu.project.entity.User;
import ru.edu.project.foundation.repositories.UserRepository;
import ru.edu.project.mediator.dto.UserDto;
import ru.edu.project.mediator.exceptions.UserNotFoundException;
import ru.edu.project.mediator.interfaces.IUserService;

import java.util.List;
import java.util.stream.Collectors;

@Service
@Transactional
public class UserServiceImpl implements IUserService {

    private final UserRepository userRepository;

    public UserServiceImpl(UserRepository userRepository) {
        this.userRepository = userRepository;
    }

    @Override
    @Transactional(readOnly = true)
    public UserDto getUserById(Long id) {
```

```
        User user = userRepository.findById(id)
                .orElseThrow(() -> new UserNotFoundException(id));
        return UserDto.fromEntity(user);
    }

    @Override
    @Transactional(readOnly = true)
    public UserDto getUserByEmail(String email) {
        User user = userRepository.findByEmail(email)
                .orElseThrow(() -> new UserNotFoundException(email));
        return UserDto.fromEntity(user);
    }

    @Override
    @Transactional(readOnly = true)
    public List<UserDto> getAllUsers() {
        return userRepository.findAll().stream()
                .map(UserDto::fromEntity)
                .collect(Collectors.toList());
    }

    @Override
    @Transactional(readOnly = true)
    public List<UserDto> getUsersByRole(String role) {
        return userRepository.findByRole(role).stream()
                .map(UserDto::fromEntity)
                .collect(Collectors.toList());
    }
```

```
    @Override
    public UserDto createUser(String email, String password, String name) {
        if (userRepository.existsByEmail(email)) {
            throw new BusinessException("Email already exists: " + email);
        }

        User user = User.builder()
                .email(email)
                .passwordHash(password) // в реальном проекте хешировать BCrypt
                .name(name)
                .role("USER")
                .build();

        User saved = userRepository.save(user);
        return UserDto.fromEntity(saved);
    }

    @Override
    public UserDto updateUser(Long id, String name, String role) {
        User user = userRepository.findById(id)
                .orElseThrow(() -> new UserNotFoundException(id));

        if (name != null && !name.isEmpty()) {
            user.setName(name);
        }
        if (role != null && !role.isEmpty()) {
            user.setRole(role);
        }
```

```
        User updated = userRepository.update(user);
        return UserDto.fromEntity(updated);
    }

    @Override
    public void deleteUser(Long id) {
        if (!userRepository.existsById(id)) {
            throw new UserNotFoundException(id);
        }
        userRepository.deleteById(id);
    }

    @Override
    @Transactional(readOnly = true)
    public UserDto authenticate(String email, String password) {
        User user = userRepository.findByEmail(email)
                .orElseThrow(() -> new UserNotFoundException(email));

        if (!user.checkPassword(password)) {
            throw new BusinessException("Invalid password for user: " + email);
        }

        return UserDto.fromEntity(user);
    }
}
```

EventServiceImpl.java:

```
package ru.edu.project.mediator.services;
```

```
import org springframework stereotype Service;
import org springframework transaction annotation Transactional;
import ru edu project entity Category;
import ru edu project entity Event;
import ru edu project entity User;
import ru edu project foundation repositories CategoryRepository;
import ru edu project foundation repositories EventRepository;
import ru edu project foundation repositories UserRepository;
import ru edu project mediator dto EventDto;
import ru edu project mediator exceptions EventNotFoundException;
import ru edu project mediator exceptions UserNotFoundException;
import ru edu project mediator interfaces IEventService;

import java time LocalDateTime;
import java util List;
import java util stream Collectors;

@Service
@Transactional
public class EventServiceImpl implements IEventService {

    private final EventRepository eventRepository;
    private final UserRepository userRepository;
    private final CategoryRepository categoryRepository;

    public EventServiceImpl(EventRepository eventRepository,
                            UserRepository userRepository,
                            CategoryRepository categoryRepository) {
        this eventRepository = eventRepository;
```

```
        this.userRepository = userRepository;
        this.categoryRepository = categoryRepository;
    }

    @Override
    @Transactional(readOnly = true)
    public EventDto getEventById(Long id) {
        Event event = eventRepository.findById(id)
                .orElseThrow(() -> new EventNotFoundException(id));
        return EventDto.fromEntity(event);
    }

    @Override
    @Transactional(readOnly = true)
    public List<EventDto> getAllEvents() {
        return eventRepository.findAll().stream()
                .map(EventDto::fromEntity)
                .collect(Collectors.toList());
    }

    @Override
    @Transactional(readOnly = true)
    public List<EventDto> getPublishedEvents() {
        return eventRepository.findByStatus("PUBLISHED").stream()
                .map(EventDto::fromEntity)
                .collect(Collectors.toList());
    }

    @Override
```

```
    @Transactional(readOnly = true)
    public List<EventDto> getEventsByCategory(Long categoryId) {
        return eventRepository.findByCategoryId(categoryId).stream()
                .map(EventDto::fromEntity)
                .collect(Collectors.toList());
    }

    @Override
    @Transactional(readOnly = true)
    public List<EventDto> getEventsInDateRange(LocalDateTime start,
LocalDateTime end) {
        return eventRepository.findByStartDateBetween(start, end).stream()
                .map(EventDto::fromEntity)
                .collect(Collectors.toList());
    }

    @Override
    public EventDto createEvent(String title, String description,
LocalDateTime startDate,
                                LocalDateTime endDate, Integer maxParticipants,
                                Long categoryId, Long createdBy) {

        if (startDate.isAfter(endDate)) {
            throw new BusinessException("Start date must be before end date");
        }

        User creator = userRepository.findById(createdBy)
                .orElseThrow(() -> new UserNotFoundException(createdBy));
```

```
        Category category = null;
        if (categoryId != null) {
            category = categoryRepository.findById(categoryId)
                    .orElseThrow(() -> new BusinessException("Category not found: " + categoryId));
        }

        Event event = Event.builder()
                .title(title)
                .description(description)
                .startDate(startDate)
                .endDate(endDate)
                .maxParticipants(maxParticipants)
                .status("DRAFT")
                .category(category)
                .createdBy(creator)
                .build();

        Event saved = eventRepository.save(event);
        return EventDto.fromEntity(saved);
    }

    @Override
    public EventDto updateEvent(Long id, String title, String description,
                                LocalDateTime startDate, LocalDateTime endDate,
                                Integer maxParticipants, Long categoryId) {

        Event event = eventRepository.findById(id)
                .orElseThrow(() -> new EventNotFoundException(id));
```

```
        if (startDate != null && endDate != null &&
startDate.isAfter(endDate)) {
            throw new BusinessException("Start date must be before end date");
        }

        if (title != null) event.setTitle(title);
        if (description != null) event.setDescription(description);
        if (startDate != null) event.setStartDate(startDate);
        if (endDate != null) event.setEndDate(endDate);
        if (maxParticipants != null) event.setMaxParticipants(maxParticipants);
        if (categoryId != null) {
            Category category = categoryRepository.findById(categoryId)
                    .orElseThrow(() -> new BusinessException("Category not
found: " + categoryId));
            event.setCategory(category);
        }

        Event updated = eventRepository.update(event);
        return EventDto.fromEntity(updated);
    }

    @Override
    public void publishEvent(Long id) {
        Event event = eventRepository.findById(id)
                .orElseThrow(() -> new EventNotFoundException(id));
        event.publish();
        eventRepository.update(event);
    }
```

```
    @Override
    public void cancelEvent(Long id) {
        Event event = eventRepository.findById(id)
                .orElseThrow(() -> new EventNotFoundException(id));
        event.cancel();
        eventRepository.update(event);
    }

    @Override
    public void deleteEvent(Long id) {
        if (!eventRepository.existsById(id)) {
            throw new EventNotFoundException(id);
        }
        eventRepository.deleteById(id);
    }
}
```

RegistrationServiceImpl.java:

```
package ru.edu.project.mediator.services;

import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;
import ru.edu.project.entity.Event;
import ru.edu.project.entity.Registration;
import ru.edu.project.entity.User;
import ru.edu.project.foundation.repositories.EventRepository;
import ru.edu.project.foundation.repositories.RegistrationRepository;
import ru.edu.project.foundation.repositories.UserRepository;
import ru.edu.project.mediator.dto.RegistrationDto;
```

```
import ru.edu.project.mediator.exceptions.*;
import ru.edu.project.mediator.interfaces.IRegistrationService;

import java.util.List;
import java.util.stream.Collectors;

@Service
@Transactional
public class RegistrationServiceImpl implements IRegistrationService {

    private final UserRepository userRepository;
    private final EventRepository eventRepository;
    private final RegistrationRepository registrationRepository;

    public RegistrationServiceImpl(UserRepository userRepository,
                                   EventRepository eventRepository,
                                   RegistrationRepository registrationRepository) {
        this.userRepository = userRepository;
        this.eventRepository = eventRepository;
        this.registrationRepository = registrationRepository;
    }

    @Override
    @Transactional(readOnly = true)
    public RegistrationDto getRegistrationById(Long id) {
        Registration registration = registrationRepository.findById(id)
                .orElseThrow(() -> new RegistrationNotFoundException(id));
        return RegistrationDto.fromEntity(registration);
    }
```

```
    @Override
    @Transactional(readOnly = true)
    public List<RegistrationDto> getRegistrationsByUser(Long userId) {
        return registrationRepository.findByUserId(userId).stream()
                .map(RegistrationDto::fromEntity)
                .collect(Collectors.toList());
    }

    @Override
    @Transactional(readOnly = true)
    public List<RegistrationDto> getRegistrationsByEvent(Long eventId) {
        return registrationRepository.findByEventId(eventId).stream()
                .map(RegistrationDto::fromEntity)
                .collect(Collectors.toList());
    }

    @Override
    public RegistrationDto registerUserForEvent(Long userId, Long eventId)
{
        // 1. Проверка существования пользователя
        User user = userRepository.findById(userId)
                .orElseThrow(() -> new UserNotFoundException(userId));

        // 2. Проверка существования мероприятия
        Event event = eventRepository.findById(eventId)
                .orElseThrow(() -> new EventNotFoundException(eventId));

        // 3. Проверка, не зарегистрирован ли пользователь уже
```

```
        if (registrationRepository.findByUserIdAndEventId(userId, eventId).isPresent()) {
            throw new DuplicateRegistrationException(userId, eventId);
        }

        // 4. Проверка наличия свободных мест (бизнес-правило в сущности)
        if (!event.checkAvailability()) {
            throw new NoSeatsAvailableException(eventId);
        }

        // 5. Регистрация пользователя (бизнес-правило в сущности)
        Registration registration = event.registerUser(user);

        // 6. Сохранение в БД
        Registration saved = registrationRepository.save(registration);

        return RegistrationDto.fromEntity(saved);
    }

    @Override
    public void cancelRegistration(Long registrationId) {
        Registration registration = registrationRepository.findById(registrationId)
                .orElseThrow(() -> new RegistrationNotFoundException(registrationId));

        // Бизнес-правило: отмена возможна не позднее чем за 24 часа
        Event event = registration.getEvent();
```

```
            if
(event.getStartDate().minusHours(24).isBefore(java.time.LocalDateTime.now())) {
                throw new BusinessException("Cancellation is not allowed within 24
hours of the event");
            }

            registration.cancel();
            registrationRepository.update(registration);
        }

        @Override
        public void confirmRegistration(Long registrationId) {
            Registration registration =
registrationRepository.findById(registrationId)
                    .orElseThrow(() -> new
RegistrationNotFoundException(registrationId));

            registration.confirm();
            registrationRepository.update(registration);
        }
    }
```

3.8. Модульное тестирование сервисов с Mockito

RegistrationServiceImplTest.java:

```
package ru.edu.project.mediator.services;

import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.DisplayName;
import org.junit.jupiter.api.Test;
```

```
import org.junit.jupiter.api.extension.ExtendWith;
import org.mockito.InjectMocks;
import org.mockito.Mock;
import org.mockito.junit.jupiter.MockitoExtension;
import ru.edu.project.entity.Event;
import ru.edu.project.entity.Registration;
import ru.edu.project.entity.User;
import ru.edu.project.foundation.repositories.EventRepository;
import ru.edu.project.foundation.repositories.RegistrationRepository;
import ru.edu.project.foundation.repositories.UserRepository;
import ru.edu.project.mediator.dto.RegistrationDto;
import ru.edu.project.mediator.exceptions.*;

import java.math.BigDecimal;
import java.time.LocalDateTime;
import java.util.Optional;

import static org.junit.jupiter.api.Assertions.*;
import static org.mockito.ArgumentMatchers.any;
import static org.mockito.Mockito.*;

@ExtendWith(MockitoExtension.class)
@DisplayName("Тестирование RegistrationServiceImpl")
class RegistrationServiceImplTest {

    @Mock
    private UserRepository userRepository;

    @Mock
```

```
    private EventRepository eventRepository;

    @Mock
    private RegistrationRepository registrationRepository;

    @InjectMocks
    private RegistrationServiceImpl registrationService;

    private User testUser;
    private Event testEvent;
    private Registration testRegistration;

    @BeforeEach
    void setUp() {
        testUser = User.builder()
                .id(1L)
                .email("user@example.com")
                .name("Test User")
                .build();

        testEvent = Event.builder()
                .id(1L)
                .title("Test Event")
                .maxParticipants(10)
                .startDate(LocalDateTime.now().plusDays(7))
                .endDate(LocalDateTime.now().plusDays(8))
                .build();

        testRegistration = Registration.builder()
```

```
                .id(1L)
                .user(testUser)
                .event(testEvent)
                .status("PENDING")
                .build();
    }

    @Test
    @DisplayName("Должен успешно зарегистрировать пользователя")
    void shouldRegisterUserSuccessfully() {
        // given
        when(userRepository.findById(1L)).thenReturn(Optional.of(testUser));

when(eventRepository.findById(1L)).thenReturn(Optional.of(testEvent));
        when(registrationRepository.findByUserIdAndEventId(1L, 1L))
                .thenReturn(Optional.empty());
        when(registrationRepository.save(any(Registration.class)))
                .thenReturn(testRegistration);

        // when
        RegistrationDto result = registrationService.registerUserForEvent(1L,
1L);

        // then
        assertNotNull(result);
        assertEquals(1L, result.getId());
        verify(registrationRepository).save(any(Registration.class));
    }
```

```
    @Test
    @DisplayName("Должен выбросить исключение при отсутствии пользователя")
    void shouldThrowExceptionWhenUserNotFound() {
        // given
        when(userRepository.findById(999L)).thenReturn(Optional.empty());

        // then
        assertThrows(UserNotFoundException.class, () -> {
            registrationService.registerUserForEvent(999L, 1L);
        });

        verify(eventRepository, never()).findById(any());
    }

    @Test
    @DisplayName("Должен выбросить исключение при отсутствии мероприятия")
    void shouldThrowExceptionWhenEventNotFound() {
        // given
        when(userRepository.findById(1L)).thenReturn(Optional.of(testUser));
        when(eventRepository.findById(999L)).thenReturn(Optional.empty());

        // then
        assertThrows(EventNotFoundException.class, () -> {
            registrationService.registerUserForEvent(1L, 999L);
        });
    }
```

```
    @Test
    @DisplayName("Должен выбросить исключение при повторной регистрации")
    void shouldThrowExceptionWhenDuplicateRegistration() {
        // given
        when(userRepository.findById(1L)).thenReturn(Optional.of(testUser));

when(eventRepository.findById(1L)).thenReturn(Optional.of(testEvent));
        when(registrationRepository.findByUserIdAndEventId(1L, 1L))
                .thenReturn(Optional.of(testRegistration));

        // then
        assertThrows(DuplicateRegistrationException.class, () -> {
            registrationService.registerUserForEvent(1L, 1L);
        });

        verify(registrationRepository, never()).save(any());
    }

    @Test
    @DisplayName("Должен выбросить исключение при отсутствии мест")
    void shouldThrowExceptionWhenNoSeatsAvailable() {
        // given
        testEvent.setMaxParticipants(1);
        testEvent.registerUser(testUser); // заполняем единственное место

        when(userRepository.findById(2L)).thenReturn(Optional.of(
                User.builder().id(2L).build()));
```

```
when(eventRepository.findById(1L)).thenReturn(Optional.of(testEvent));
        when(registrationRepository.findByUserIdAndEventId(2L, 1L))
                .thenReturn(Optional.empty());

        // then
        assertThrows(NoSeatsAvailableException.class, () -> {
            registrationService.registerUserForEvent(2L, 1L);
        });
    }

    @Test
    @DisplayName("Должен успешно отменить регистрацию")
    void shouldCancelRegistrationSuccessfully() {
        // given
        testEvent.setStartDate(LocalDateTime.now().plusDays(2));

when(registrationRepository.findById(1L)).thenReturn(Optional.of(testRegistratio
n));
        when(registrationRepository.update(any(Registration.class)))
                .thenReturn(testRegistration);

        // when
        registrationService.cancelRegistration(1L);

        // then
        assertEquals("CANCELLED", testRegistration.getStatus());
        verify(registrationRepository).update(testRegistration);
    }
```

```
    @Test
    @DisplayName("Должен выбросить исключение при отмене регистрации менее чем за 24 часа")
    void shouldThrowExceptionWhenCancellationTooLate() {
        // given
        testEvent.setStartDate(LocalDateTime.now().plusHours(12));  // менее 24 часов

        when(registrationRepository.findById(1L)).thenReturn(Optional.of(testRegistration));

        // then
        assertThrows(BusinessException.class, () -> {
            registrationService.cancelRegistration(1L);
        });

        verify(registrationRepository, never()).update(any());
    }
}
```

4. МЕТОДИКА ВЫПОЛНЕНИЯ (ПОШАГОВАЯ ИНСТРУКЦИЯ)

Шаг 1: Настройка Spring-контекста

Добавить зависимости Spring в pom.xml

Создать конфигурационный класс AppConfig

Настроить DataSource, EntityManagerFactory, TransactionManager

Включить сканирование компонентов (@ComponentScan)

Шаг 2: Адаптация репозиториев для Spring

Добавить аннотацию @Repository к репозиториям

Добавить аннотацию @PersistenceContext для EntityManager

Добавить аннотацию @Transactional к методам репозиториев

Шаг 3: Создание DTO

Создать DTO для каждой сущности (UserDto, EventDto, RegistrationDto)

Добавить статические методы fromEntity()

Добавить equals(), hashCode(), toString()

Шаг 4: Создание бизнес-исключений

Создать базовый класс BusinessException

Создать специфические исключения для каждой ситуации

Шаг 5: Создание интерфейсов сервисов

Создать IUserService

Создать IEventService

Создать IRegistrationService

Шаг 6: Реализация сервисов

Реализовать UserServiceImpl

Реализовать EventServiceImpl

Реализовать RegistrationServiceImpl

Добавить аннотации @Service и @Transactional

Шаг 7: Модульное тестирование

Настроить тесты с @ExtendWith(MockitoExtension.class)

Создать моки для репозиториев с @Mock

Внедрить тестируемый сервис с @InjectMocks

Написать тесты для всех методов (позитивные и негативные сценарии)

5. ВАРИАНТЫ ДЛЯ ИНДИВИДУАЛЬНОГО ВЫПОЛНЕНИЯ

Варианты соответствуют проектам из предыдущих лабораторных работ.

	Проект	Ключевые сервисы
	Система управления складом	ProductService, WarehouseService, StockMovementService
	Платформа аренды оборудования	EquipmentService, RentalService, CustomerService
	Система управления парком автомобилей	VehicleService, DriverService, TripService
	Система дистанционного обучения	CourseService, EnrollmentService, ProgressService
	Платформа онлайн-курсов	CourseService, QuizService, CertificateService
	Система записи на кружки	CircleService, AttendanceService, PaymentService
	Система медицинских назначений	PatientService, AppointmentService, PrescriptionService
	Платформа записи на процедуры	PatientService, ProcedureService,

	Проект	Ключевые сервисы
		ScheduleService
	Система мониторинга здоровья	PatientService, MeasurementService, AlertService
0	Система доставки еды	OrderService, CourierService, PaymentService
1	Платформа бронирования услуг	SalonService, BookingService, ReviewService
2	Система проката инвентаря	ItemService, RentalService, CustomerService
3	Система производственных заказов	OrderService, OperationService, WorkerService
4	Платформа логистики	ShipmentService, RouteService, TrackingService
5	Система управления проектами	ProjectService, TaskService, WorkerService

6. КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Какие задачи решает Mediator-слой в архитектуре PCMEF?
2. Что такое Dependency Injection и какие способы внедрения существуют?
3. Для чего нужна аннотация @Transactional?
4. Что такое DTO и зачем он нужен?
5. В чём разница между @Component, @Service, @Repository?
6. Что такое @PersistenceContext и как он работает?
7. Какие параметры можно задать в @Transactional?
8. Как протестировать сервис с использованием моков репозиториев?
9. Как реализовать транзакцию, затрагивающую несколько сервисов?
10. В чём разница между REQUIRED и REQUIRES_NEW propagation?
11. Как настроить rollbackFor для специфических исключений?
12. Как избежать проблемы N+1 запросов в сервисном слое?

7. ТРЕБОВАНИЯ К ОТЧЕТУ

Структура отчета

1. Титульный лист
2. Содержание
3. Введение (цель работы, краткое описание проекта)
4. Настройка Spring-контекста
   4.1. Зависимости в pom.xml
   4.2. Конфигурационный класс AppConfig
5. Адаптация репозиториев для Spring
   5.1. Аннотации @Repository, @PersistenceContext
6. DTO (UserDto, EventDto, RegistrationDto)

7. Бизнес-исключения

8. Интерфейсы сервисов (IUserService, IEventService, IRegistrationService)

9. Реализация сервисов (UserServiceImpl, EventServiceImpl, RegistrationServiceImpl)

10. Модульное тестирование

10.1. Тесты для RegistrationServiceImpl

10.2. Результаты тестирования

11. Заключение (выводы, связь с последующими работами)

12. Список использованных источников

13. Приложения (полный код)

8. КРИТЕРИИ ОЦЕНИВАНИЯ

Критерий	**Макс. балл**	**Описание**
Настройка Spring-контекста	3	Корректная конфигурация, зависимости
Адаптация репозиториев	3	Аннотации @Repository, @PersistenceContext
DTO	3	Все DTO созданы, метод fromEntity
Бизнес-исключения	2	Иерархия исключений
Интерфейсы сервисов	3	Полнота методов, JavaDoc
Реализация сервисов	5	Бизнес-логика, транзакции, обработка

Критерий	Макс. балл	Описание
		исключений
Модульное тестирование	5	Покрытие всех методов, моки, позитивные/негативные сценарии
Оформление отчета	2	Соответствие требованиям
Ответы на контрольные вопросы	2	Понимание материала
ИТОГО	28	

Лабораторная работа №16

Реализация Control-слоя (Spring @RestController)

1. Цель и задачи работы

Цель: Реализовать Control-слой (REST API) программной системы на основе спроектированной диаграммы классов (ЛР13) и реализованного Mediator-слоя (ЛР15) с использованием Spring MVC.

Задачи:

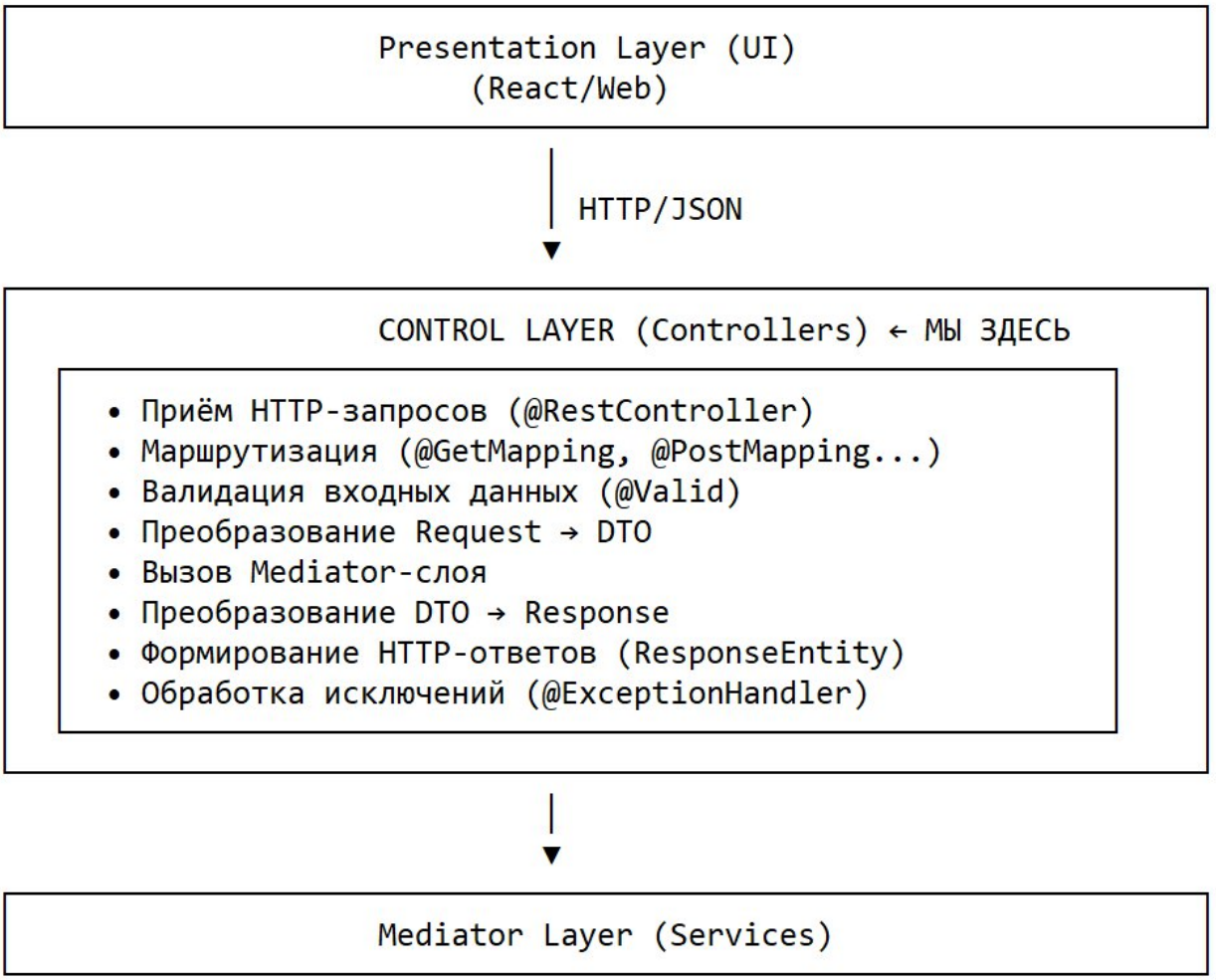
1. Создать REST-контроллеры с аннотацией @RestController
2. Реализовать CRUD-эндпоинты для всех основных сущностей

3. Настроить маршрутизацию запросов (@RequestMapping, @GetMapping, @PostMapping, @PutMapping, @DeleteMapping)
4. Реализовать валидацию входных данных (@Valid, @NotNull, @Size)
5. Преобразовать DTO в Request/Response объекты
6. Настроить обработку бизнес-исключений (@ControllerAdvice, @ExceptionHandler)
7. Написать интеграционные тесты для контроллеров (MockMvc)

2. ТЕОРЕТИЧЕСКОЕ ОБОСНОВАНИЕ

2.1. Место Control-слоя в архитектуре PCMEF

Control-слой — это входная точка в систему. Он принимает HTTP-запросы, валидирует входные данные, вызывает методы Mediator-слоя и возвращает HTTP-ответы.



2.2. Структура Control-слоя

control/

├── controllers/

│ ├── UserController.java

│ ├── EventController.java

│ ├── RegistrationController.java

│ ├── CategoryController.java

│ └── AuthController.java

├── dto/

│ ├── request/

│ │ ├── CreateUserRequest.java

│ │ ├── UpdateUserRequest.java

```
│   │   ├── CreateEventRequest.java
│   │   ├── UpdateEventRequest.java
│   │   ├── LoginRequest.java
│   │   └── PriceCalculationRequest.java
│   └── response/
│       ├── UserResponse.java
│       ├── EventResponse.java
│       ├── RegistrationResponse.java
│       ├── AuthResponse.java
│       ├── PriceResponse.java
│       └── ErrorResponse.java
├── advice/
│   └── GlobalExceptionHandler.java
└── config/
    ├── WebConfig.java (CORS)
    └── SwaggerConfig.java (OpenAPI)
```

2.3. Ключевые аннотации Spring Web

Аннотация	Назначение	Пример
@RestController	Объявляет класс REST-контроллером	@RestController
@RequestMapping	Базовый путь для всех эндпоинтов контроллера	@RequestMapping("/api/users")

Аннотация	Назначение	Пример
@GetMapping	Обработка GET-запросов	@GetMapping("/{id}")
@PostMapping	Обработка POST-запросов	@PostMapping
@PutMapping	Обработка PUT-запросов	@PutMapping("/{id}")
@DeleteMapping	Обработка DELETE-запросов	@DeleteMapping("/{id}")
@PathVariable	Извлечение параметра из URL	@PathVariable Long id
@RequestParam	Извлечение query-параметра	@RequestParam(required = false) String role
@RequestBody	Преобразование JSON в объект	@RequestBody CreateUserRequest request
@Valid	Включение валидации	@Valid @RequestBody CreateUserRequest request

2.4. HTTP-статусы (ResponseEntity)

Статус	од	Когда использовать	Пример
OK	00	Успешный GET, PUT, DELETE	ResponseEntity.ok(response)
CREATED	01	Успешный POST (создание)	ResponseEntity.status(HttpStatus.CREATED).body(response)
NO_CONTENT	04	Успешный DELETE без тела	ResponseEntity.noContent().build()
BAD_REQUEST	00	Ошибка валидации	ResponseEntity.badRequest().body(error)
NOT_FOUND	04	Ресурс не найден	ResponseEntity.notFound().build()
CONFLICT	09	Конфликт (дубликат)	ResponseEntity.status(409).body(error)

2.5. Валидация входных данных

Зависимости в pom.xml:

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-validation</artifactId>
</dependency>
```

Аннотации валидации:

Аннотация	Назначение	Пример
@NotNull	Поле не может быть null	@NotNull
@NotBlank	Строка не пустая и не null	@NotBlank(message = "Name is required")
@Size(min, max)	Ограничение длины строки	@Size(min = 3, max = 100)
@Email	Проверка формата email	@Email
@Min / @Max	Числовой диапазон	@Min(1) @Max(10000)
@Positive	Положительное число	@Positive
@Past / @Future	Дата в прошлом/будущем	@Future

2.6. Обработка исключений в Control-слое

```
@RestControllerAdvice
public class GlobalExceptionHandler {
```

```
    @ExceptionHandler(UserNotFoundException.class)
    public ResponseEntity<ErrorResponse> handleNotFound(UserNotFoundException e) {
        return ResponseEntity.status(HttpStatus.NOT_FOUND)
            .body(new ErrorResponse(e.getMessage()));
    }

    @ExceptionHandler(NoSeatsAvailableException.class)
    public ResponseEntity<ErrorResponse> handleConflict(BusinessException e) {
        return ResponseEntity.status(HttpStatus.CONFLICT)
            .body(new ErrorResponse(e.getMessage()));
    }

    @ExceptionHandler(MethodArgumentNotValidException.class)
    public ResponseEntity<ErrorResponse> handleValidation(MethodArgumentNotValidException e) {
        String errors = e.getBindingResult().getAllErrors().stream()
            .map(error -> error.getDefaultMessage())
            .collect(Collectors.joining(", "));
        return ResponseEntity.badRequest()
            .body(new ErrorResponse("Validation failed: " + errors));
    }
}
```

2.7. Структура Request и Response DTO

```
// Request (входные данные)
public class CreateEventRequest {
    @NotBlank(message = "Title is required")
```

```
    @Size(min = 3, max = 200)
    private String title;

    @Future(message = "Start date must be in the future")
    private LocalDateTime startDate;

    @Positive(message = "Max participants must be positive")
    private Integer maxParticipants;
    // getters/setters
}

// Response (выходные данные)
public class EventResponse {
    private Long id;
    private String title;
    private String status;
    // getters/setters
}
```

3. ПРИМЕР ВЫПОЛНЕНИЯ (СКВОЗНОЙ ПРОЕКТ)

Проект: «Система управления мероприятиями» (Event Management System)

3.1. Request/Response DTO

CreateEventRequest.java:

```
package ru.edu.project.control.dto.request;

import jakarta.validation.constraints.*;
import lombok.Data;
import java.time.LocalDateTime;
```

```
@Data
public class CreateEventRequest {

    @NotBlank(message = "Title is required")
    @Size(min = 3, max = 200, message = "Title must be between 3 and 200 characters")
    private String title;

    @Size(max = 5000, message = "Description cannot exceed 5000 characters")
    private String description;

    @NotNull(message = "Start date is required")
    @Future(message = "Start date must be in the future")
    private LocalDateTime startDate;

    @NotNull(message = "End date is required")
    @Future(message = "End date must be in the future")
    private LocalDateTime endDate;

    @NotNull(message = "Max participants is required")
    @Positive(message = "Max participants must be positive")
    @Max(value = 10000, message = "Max participants cannot exceed 10000")
    private Integer maxParticipants;

    private Long categoryId;
}
```

LoginRequest.java:

```
package ru.edu.project.control.dto.request;

import jakarta.validation.constraints.Email;
import jakarta.validation.constraints.NotBlank;
import lombok.Data;

@Data
public class LoginRequest {

    @NotBlank(message = "Email is required")
    @Email(message = "Email should be valid")
    private String email;

    @NotBlank(message = "Password is required")
    @Size(min = 6, message = "Password must be at least 6 characters")
    private String password;
}
```

EventResponse.java:

```
package ru.edu.project.control.dto.response;

import lombok.Builder;
import lombok.Data;
import ru.edu.project.mediator.dto.EventDto;
import java.time.LocalDateTime;

@Data
@Builder
public class EventResponse {
```

```
    private Long id;
    private String title;
    private String description;
    private LocalDateTime startDate;
    private LocalDateTime endDate;
    private Integer maxParticipants;
    private Integer registeredCount;
    private Integer availableSeats;
    private String status;
    private String categoryName;
    private String createdBy;

    public static EventResponse fromDto(EventDto dto) {
        if (dto == null) return null;
        return EventResponse.builder()
                .id(dto.getId())
                .title(dto.getTitle())
                .description(dto.getDescription())
                .startDate(dto.getStartDate())
                .endDate(dto.getEndDate())
                .maxParticipants(dto.getMaxParticipants())
                .registeredCount(dto.getRegisteredCount())
                .availableSeats(dto.getAvailableSeats())
                .status(dto.getStatus())
                .categoryName(dto.getCategoryName())
                .createdBy(dto.getCreatedBy())
                .build();
    }
}
```

ErrorResponse.java:

```
package ru.edu.project.control.dto.response;

import lombok.AllArgsConstructor;
import lombok.Data;
import java.time.LocalDateTime;

@Data
@AllArgsConstructor
public class ErrorResponse {
    private String message;
    private int status;
    private String path;
    private LocalDateTime timestamp;

    public ErrorResponse(String message) {
        this.message = message;
        this.timestamp = LocalDateTime.now();
    }
}
```

3.2. REST-контроллеры

EventController.java:

```
package ru.edu.project.control.controllers;

import io.swagger.v3.oas.annotations.Operation;
import io.swagger.v3.oas.annotations.tags.Tag;
import jakarta.validation.Valid;
import lombok.RequiredArgsConstructor;
import lombok.extern.slf4j.Slf4j;
```

```
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;
import ru.edu.project.control.dto.request.CreateEventRequest;
import ru.edu.project.control.dto.request.UpdateEventRequest;
import ru.edu.project.control.dto.response.EventResponse;
import ru.edu.project.mediator.dto.EventDto;
import ru.edu.project.mediator.interfaces.IEventService;

import java.util.List;

@Slf4j
@RestController
@RequestMapping("/api/events")
@RequiredArgsConstructor
@Tag(name = "Events", description = "API для управления мероприятиями")
public class EventController {

    private final IEventService eventService;

    @GetMapping
    @Operation(summary = "Получить список всех мероприятий")
    public ResponseEntity<List<EventResponse>> getAllEvents() {
        log.debug("GET /api/events");
        List<EventResponse> events = eventService.getAllEvents().stream()
                .map(EventResponse::fromDto)
                .toList();
        return ResponseEntity.ok(events);
```

```
    }

    @GetMapping("/published")
    @Operation(summary = "Получить список опубликованных мероприятий")
    public ResponseEntity<List<EventResponse>> getPublishedEvents() {
        log.debug("GET /api/events/published");
        List<EventResponse> events = eventService.getPublishedEvents().stream()
                .map(EventResponse::fromDto)
                .toList();
        return ResponseEntity.ok(events);
    }

    @GetMapping("/{id}")
    @Operation(summary = "Получить мероприятие по ID")
    public ResponseEntity<EventResponse> getEventById(@PathVariable Long id) {
        log.debug("GET /api/events/{}", id);
        EventDto event = eventService.getEventById(id);
        return ResponseEntity.ok(EventResponse.fromDto(event));
    }

    @PostMapping
    @Operation(summary = "Создать новое мероприятие")
    public ResponseEntity<EventResponse> createEvent(@Valid @RequestBody CreateEventRequest request) {
        log.info("POST /api/events - creating event: {}", request.getTitle());
```

```
        EventDto created = eventService.createEvent(
                request.getTitle(),
                request.getDescription(),
                request.getStartDate(),
                request.getEndDate(),
                request.getMaxParticipants(),
                request.getCategoryId(),
                1L  // TODO: получить из контекста безопасности
        );

        return ResponseEntity.status(HttpStatus.CREATED)
                .body(EventResponse.fromDto(created));
    }

    @PutMapping("/{id}")
    @Operation(summary = "Обновить мероприятие")
    public ResponseEntity<EventResponse> updateEvent(
            @PathVariable Long id,
            @Valid @RequestBody UpdateEventRequest request) {

        log.info("PUT /api/events/{} - updating event", id);

        EventDto updated = eventService.updateEvent(
                id,
                request.getTitle(),
                request.getDescription(),
                request.getStartDate(),
                request.getEndDate(),
                request.getMaxParticipants(),
```

```
            request.getCategoryId()
        );

        return ResponseEntity.ok(EventResponse.fromDto(updated));
    }

    @DeleteMapping("/{id}")
    @Operation(summary = "Удалить мероприятие")
    public ResponseEntity<Void> deleteEvent(@PathVariable Long id) {
        log.info("DELETE /api/events/{}", id);
        eventService.deleteEvent(id);
        return ResponseEntity.noContent().build();
    }

    @PatchMapping("/{id}/publish")
    @Operation(summary = "Опубликовать мероприятие")
    public ResponseEntity<Void> publishEvent(@PathVariable Long id) {
        log.info("PATCH /api/events/{}/publish", id);
        eventService.publishEvent(id);
        return ResponseEntity.ok().build();
    }

    @PatchMapping("/{id}/cancel")
    @Operation(summary = "Отменить мероприятие")
    public ResponseEntity<Void> cancelEvent(@PathVariable Long id) {
        log.info("PATCH /api/events/{}/cancel", id);
        eventService.cancelEvent(id);
        return ResponseEntity.ok().build();
    }
```

```
}
```

RegistrationController.java:

```
package ru.edu.project.control.controllers;

import io.swagger.v3.oas.annotations.Operation;
import io.swagger.v3.oas.annotations.tags.Tag;
import jakarta.validation.Valid;
import lombok.RequiredArgsConstructor;
import lombok.extern.slf4j.Slf4j;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;
import ru.edu.project.control.dto.request.PriceCalculationRequest;
import ru.edu.project.control.dto.response.PriceResponse;
import ru.edu.project.control.dto.response.RegistrationResponse;
import ru.edu.project.mediator.dto.RegistrationDto;
import ru.edu.project.mediator.interfaces.IRegistrationService;
import ru.edu.project.mediator.strategy.impl.EarlyBirdPricing;
import ru.edu.project.mediator.strategy.impl.GroupPricing;

import java.util.List;

@Slf4j
@RestController
@RequestMapping("/api/registrations")
@RequiredArgsConstructor
@Tag(name = "Registrations", description = "API для управления регистрациями")
public class RegistrationController {
```

```
    private final IRegistrationService registrationService;

    @GetMapping("/user/{userId}")
    @Operation(summary = "Получить регистрации пользователя")
    public ResponseEntity<List<RegistrationResponse>>
getRegistrationsByUser(
            @PathVariable Long userId) {
        log.debug("GET /api/registrations/user/{}", userId);
        List<RegistrationResponse> registrations = registrationService
                .getRegistrationsByUser(userId).stream()
                .map(RegistrationResponse::fromDto)
                .toList();
        return ResponseEntity.ok(registrations);
    }

    @GetMapping("/event/{eventId}")
    @Operation(summary = "Получить регистрации на мероприятие")
    public ResponseEntity<List<RegistrationResponse>>
getRegistrationsByEvent(
            @PathVariable Long eventId) {
        log.debug("GET /api/registrations/event/{}", eventId);
        List<RegistrationResponse> registrations = registrationService
                .getRegistrationsByEvent(eventId).stream()
                .map(RegistrationResponse::fromDto)
                .toList();
        return ResponseEntity.ok(registrations);
    }
```

```
    @PostMapping("/events/{eventId}/users/{userId}")
    @Operation(summary = "Зарегистрировать пользователя на мероприятие")
    public ResponseEntity<RegistrationResponse> registerUserForEvent(
        @PathVariable Long userId,
        @PathVariable Long eventId) {
        log.info("POST /api/registrations/events/{}/users/{}", eventId, userId);
        RegistrationDto registration = registrationService.registerUserForEvent(userId, eventId);
        return ResponseEntity.status(HttpStatus.CREATED)
            .body(RegistrationResponse.fromDto(registration));
    }

    @DeleteMapping("/{registrationId}")
    @Operation(summary = "Отменить регистрацию")
    public ResponseEntity<Void> cancelRegistration(@PathVariable Long registrationId) {
        log.info("DELETE /api/registrations/{}", registrationId);
        registrationService.cancelRegistration(registrationId);
        return ResponseEntity.noContent().build();
    }

    @PostMapping("/calculate-price")
    @Operation(summary = "Рассчитать стоимость регистрации")
    public ResponseEntity<PriceResponse> calculatePrice(
        @Valid @RequestBody PriceCalculationRequest request) {
        log.debug("POST /api/registrations/calculate-price");

        // Выбор стратегии на основе параметров запроса
```

```
        if (request.getQuantity() >= 5) {
            registrationService.setPricingStrategy(new GroupPricing());
        } else if (request.isEarlyBird()) {
            registrationService.setPricingStrategy(new EarlyBirdPricing());
        }
        // иначе остаётся StandardPricing (по умолчанию)

        java.math.BigDecimal price = registrationService.calculatePrice(
                request.getEventId(),
                request.getUserId(),
                request.getQuantity()
        );

        return ResponseEntity.ok(new PriceResponse(price));
    }
}
```

AuthController.java:

```
package ru.edu.project.control.controllers;

import io.swagger.v3.oas.annotations.Operation;
import io.swagger.v3.oas.annotations.tags.Tag;
import jakarta.validation.Valid;
import lombok.RequiredArgsConstructor;
import lombok.extern.slf4j.Slf4j;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;
import ru.edu.project.control.dto.request.LoginRequest;
import ru.edu.project.control.dto.request.RegisterRequest;
import ru.edu.project.control.dto.response.AuthResponse;
```

```
import ru.edu.project.mediator.dto.UserDto;
import ru.edu.project.mediator.interfaces.IUserService;

@Slf4j
@RestController
@RequestMapping("/api/auth")
@RequiredArgsConstructor
@Tag(name = "Authentication", description = "API для аутентификации")
public class AuthController {

    private final IUserService userService;

    @PostMapping("/login")
    @Operation(summary = "Вход в систему")
    public ResponseEntity<AuthResponse> login(@Valid @RequestBody
LoginRequest request) {
        log.info("POST /api/auth/login - user: {}", request.getEmail());
        UserDto user = userService.authenticate(request.getEmail(),
request.getPassword());
        // JWT токен будет добавлен в ЛР17
        return ResponseEntity.ok(new AuthResponse("fake-jwt-token", user));
    }

    @PostMapping("/register")
    @Operation(summary = "Регистрация нового пользователя")
    public ResponseEntity<AuthResponse> register(@Valid @RequestBody
RegisterRequest request) {
        log.info("POST /api/auth/register - user: {}", request.getEmail());
        UserDto user = userService.createUser(
```

```
                request.getEmail(),
                request.getPassword(),
                request.getName()
        );
        return ResponseEntity.ok(new AuthResponse("fake-jwt-token", user));
    }
}
```

3.3. Глобальная обработка исключений

GlobalExceptionHandler.java:

```
package ru.edu.project.control.advice;

import jakarta.servlet.http.HttpServletRequest;
import lombok.extern.slf4j.Slf4j;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.validation.FieldError;
import org.springframework.web.bind.MethodArgumentNotValidException;
import org.springframework.web.bind.annotation.ExceptionHandler;
import org.springframework.web.bind.annotation.RestControllerAdvice;
import ru.edu.project.control.dto.response.ErrorResponse;
import ru.edu.project.mediator.exceptions.*;

import java.time.LocalDateTime;
import java.util.stream.Collectors;

@Slf4j
@RestControllerAdvice
public class GlobalExceptionHandler {
```

```
    @ExceptionHandler(UserNotFoundException.class)
    public ResponseEntity<ErrorResponse> handleUserNotFound(UserNotFoundException e, HttpServletRequest request) {
        log.warn("User not found: {}", e.getMessage());
        return buildErrorResponse(e.getMessage(), HttpStatus.NOT_FOUND, request);
    }

    @ExceptionHandler(EventNotFoundException.class)
    public ResponseEntity<ErrorResponse> handleEventNotFound(EventNotFoundException e, HttpServletRequest request) {
        log.warn("Event not found: {}", e.getMessage());
        return buildErrorResponse(e.getMessage(), HttpStatus.NOT_FOUND, request);
    }

    @ExceptionHandler(RegistrationNotFoundException.class)
    public ResponseEntity<ErrorResponse> handleRegistrationNotFound(RegistrationNotFoundException e,
HttpServletRequest request) {
        log.warn("Registration not found: {}", e.getMessage());
        return buildErrorResponse(e.getMessage(), HttpStatus.NOT_FOUND, request);
    }

    @ExceptionHandler(NoSeatsAvailableException.class)
    public ResponseEntity<ErrorResponse> handleNoSeats(NoSeatsAvailableException e, HttpServletRequest request) {
```

```
        log.warn("No seats available: {}", e.getMessage());
        return buildErrorResponse(e.getMessage(), HttpStatus.CONFLICT,
request);
    }

    @ExceptionHandler(DuplicateRegistrationException.class)
    public ResponseEntity<ErrorResponse>
handleDuplicateRegistration(DuplicateRegistrationException e,
HttpServletRequest request) {
        log.warn("Duplicate registration: {}", e.getMessage());
        return buildErrorResponse(e.getMessage(), HttpStatus.CONFLICT,
request);
    }

    @ExceptionHandler(BusinessException.class)
    public ResponseEntity<ErrorResponse>
handleBusinessException(BusinessException e, HttpServletRequest request) {
        log.warn("Business exception: {}", e.getMessage());
        return buildErrorResponse(e.getMessage(),
HttpStatus.BAD_REQUEST, request);
    }

    @ExceptionHandler(MethodArgumentNotValidException.class)
    public ResponseEntity<ErrorResponse>
handleValidationExceptions(MethodArgumentNotValidException e,
HttpServletRequest request) {
        String errors = e.getBindingResult().getAllErrors().stream()
            .map(error -> {
                if (error instanceof FieldError fieldError) {
```

```
                    return fieldError.getField() + ": " +
fieldError.getDefaultMessage();
                }
                return error.getDefaultMessage();
            })
            .collect(Collectors.joining(", "));

        log.warn("Validation failed: {}", errors);
        return buildErrorResponse("Validation failed: " + errors,
HttpStatus.BAD_REQUEST, request);
    }

    @ExceptionHandler(Exception.class)
    public ResponseEntity<ErrorResponse>
handleGenericException(Exception e, HttpServletRequest request) {
        log.error("Unexpected error: ", e);
        return buildErrorResponse("Internal server error",
HttpStatus.INTERNAL_SERVER_ERROR, request);
    }

    private ResponseEntity<ErrorResponse> buildErrorResponse(String
message, HttpStatus status, HttpServletRequest request) {
        ErrorResponse response = new ErrorResponse(
            message,
            status.value(),
            request.getRequestURI(),
            LocalDateTime.now()
        );
        return ResponseEntity.status(status).body(response);
```

```
    }
}
```

3.4. Интеграционное тестирование контроллеров (MockMvc)

EventControllerTest.java:

```
package ru.edu.project.control.controllers;

import com.fasterxml.jackson.databind.ObjectMapper;
import org.junit.jupiter.api.Test;
import org.springframework.beans.factory.annotation.Autowired;
import
org.springframework.boot.test.autoconfigure.web.servlet.WebMvcTest;
import org.springframework.boot.test.mock.mockito.MockBean;
import org.springframework.http.MediaType;
import org.springframework.test.web.servlet.MockMvc;
import ru.edu.project.control.dto.request.CreateEventRequest;
import ru.edu.project.mediator.dto.EventDto;
import ru.edu.project.mediator.interfaces.IEventService;

import java.time.LocalDateTime;
import java.util.List;

import static org.mockito.ArgumentMatchers.any;
import static org.mockito.Mockito.when;
import
static
org.springframework.test.web.servlet.request.MockMvcRequestBuilders.*;
import
static
org.springframework.test.web.servlet.result.MockMvcResultMatchers.*;
```

```java
@WebMvcTest(EventController.class)
class EventControllerTest {

    @Autowired
    private MockMvc mockMvc;

    @Autowired
    private ObjectMapper objectMapper;

    @MockBean
    private IEventService eventService;

    @Test
    void getAllEvents_ShouldReturnEventsList() throws Exception {
        EventDto event = EventDto.builder()
                .id(1L)
                .title("Test Event")
                .status("DRAFT")
                .build();

        when(eventService.getAllEvents()).thenReturn(List.of(event));

        mockMvc.perform(get("/api/events"))
                .andExpect(status().isOk())
                .andExpect(jsonPath("$[0].id").value(1))
                .andExpect(jsonPath("$[0].title").value("Test Event"));
    }

    @Test
```

```
void getEventById_ShouldReturnEvent() throws Exception {
    EventDto event = EventDto.builder()
            .id(1L)
            .title("Test Event")
            .status("DRAFT")
            .build();

    when(eventService.getEventById(1L)).thenReturn(event);

    mockMvc.perform(get("/api/events/1"))
            .andExpect(status().isOk())
            .andExpect(jsonPath("$.id").value(1))
            .andExpect(jsonPath("$.title").value("Test Event"));
}

@Test
void createEvent_ShouldReturnCreatedEvent() throws Exception {
    CreateEventRequest request = new CreateEventRequest();
    request.setTitle("New Event");
    request.setStartDate(LocalDateTime.now().plusDays(7));
    request.setEndDate(LocalDateTime.now().plusDays(8));
    request.setMaxParticipants(100);

    EventDto created = EventDto.builder()
            .id(1L)
            .title("New Event")
            .status("DRAFT")
            .build();
```

```
        when(eventService.createEvent(any(), any(), any(), any(), any(), any(),
any()))
                .thenReturn(created);

        mockMvc.perform(post("/api/events")
                .contentType(MediaType.APPLICATION_JSON)
                .content(objectMapper.writeValueAsString(request)))
                .andExpect(status().isCreated())
                .andExpect(jsonPath("$.id").value(1))
                .andExpect(jsonPath("$.title").value("New Event"));
    }

    @Test
    void createEvent_WithInvalidData_ShouldReturnBadRequest() throws
Exception {
        CreateEventRequest request = new CreateEventRequest(); // пустой
запрос

        mockMvc.perform(post("/api/events")
                .contentType(MediaType.APPLICATION_JSON)
                .content(objectMapper.writeValueAsString(request)))
                .andExpect(status().isBadRequest());
    }

    @Test
    void deleteEvent_ShouldReturnNoContent() throws Exception {
        mockMvc.perform(delete("/api/events/1"))
                .andExpect(status().isNoContent());
    }
```

```
}
```

3.5. CORS Конфигурация

WebConfig.java:

```
package ru.edu.project.control.config;

import org.springframework.context.annotation.Configuration;
import org.springframework.web.servlet.config.annotation.CorsRegistry;
import
org.springframework.web.servlet.config.annotation.WebMvcConfigurer;

@Configuration
public class WebConfig implements WebMvcConfigurer {

    @Override
    public void addCorsMappings(CorsRegistry registry) {
        registry.addMapping("/api/**")
                .allowedOrigins("http://localhost:3000", "http://localhost:5173")
                .allowedMethods("GET", "POST", "PUT", "DELETE", "PATCH",
"OPTIONS")
                .allowedHeaders("*")
                .allowCredentials(true);
    }
}
```

4. МЕТОДИКА ВЫПОЛНЕНИЯ

Шаг 1: Добавление зависимостей

Добавить spring-boot-starter-web

Добавить spring-boot-starter-validation

Добавить spring-boot-starter-test

Шаг 2: Создание Request/Response DTO

Создать request DTO для каждой операции (Create, Update)

Добавить аннотации валидации (@NotBlank, @Size, @Min, @Future)

Создать response DTO (с методами fromDto)

Шаг 3: Создание контроллеров

Создать EventController (CRUD + publish/cancel)

Создать RegistrationController (регистрация, отмена, расчёт цены)

Создать AuthController (логин, регистрация)

Шаг 4: Настройка обработки исключений

Создать GlobalExceptionHandler с @RestControllerAdvice

Добавить @ExceptionHandler для всех бизнес-исключений

Создать ErrorResponse DTO

Шаг 5: Интеграционное тестирование

Написать тесты для GET эндпоинтов

Написать тесты для POST эндпоинтов

Написать тесты для DELETE эндпоинтов

Написать тесты для валидации

5. КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Какие задачи решает Control-слой в архитектуре PCMEF?
2. В чем разница между @Controller и @RestController?
3. Как обработать параметр из URL (/users/{id})?
4. Как получить JSON из тела запроса?
5. Как вернуть HTTP-статус 201 Created?

6. Как настроить валидацию входных данных?
7. Как глобально обработать исключения в контроллерах?
8. Как настроить CORS для React-приложения?
9. В чем разница между @RequestParam и @PathVariable?

6. ТРЕБОВАНИЯ К ОТЧЕТУ

Структура отчета:

Титульный лист

Содержание

Введение

Request/Response DTO (листинг)

REST-контроллеры (листинг)

Глобальная обработка исключений

Интеграционные тесты

7. КРИТЕРИИ ОЦЕНИВАНИЯ

Критерий	Макс. балл	Описание
Request/Response DTO	4	Валидация, преобразование
REST-контроллеры	8	Все эндпоинты, статусы
Обработка исключений	4	@RestControllerAdvice
CORS настройка	2	WebConfig
Интеграционные тесты	6	MockMvc,

Критерий	Макс. балл	Описание
		покрытие
Оформление отчета	2	Соответствие требованиям
ИТОГО	26	

Лабораторная работа №17

Интеграционное тестирование (TestContainers)

1. Цель и задачи

Цель: Освоить методику интеграционного тестирования Spring Boot приложений с использованием библиотеки TestContainers для работы с реальной базой данных в изолированной Docker-среде.

Задачи:

1. Изучить теоретические основы интеграционного тестирования
2. Настроить TestContainers для PostgreSQL в тестовой среде
3. Написать интеграционные тесты для репозиториев
4. Написать интеграционные тесты для сервисов (Mediator)
5. Написать интеграционные тесты для REST-контроллеров (MockMvc + TestContainers)
6. Настроить автоматическое управление миграциями схемы БД

2. ТЕОРЕТИЧЕСКОЕ ОБОСНОВАНИЕ

2.1. Что такое интеграционное тестирование?

Интеграционное тестирование (Integration Testing) - это уровень тестирования, на котором проверяются взаимодействия между компонентами системы (репозитории ↔ сервисы ↔ контроллеры).

Пирамида тестирования:

```
        /\
       /e2e\          End-to-end (мало, медленно)
      /----------\
     /integration\    Интеграционные (средне)
    /--------------\
   /    unit test    \  Модульные (много, быстро)
  /------------------\
```

2.2. Что такое TestContainers?

TestContainers - это Java-библиотека, которая позволяет запускать Docker-контейнеры с реальными сервисами (PostgreSQL, Redis, Kafka) во время выполнения тестов.

Преимущества:

Преимущество	Описание
Реальная БД	Тесты выполняются на настоящей PostgreSQL, а не на H2
Изоляция	Каждый тест запускает чистый контейнер
Воспроизводимость	Одинаковое окружение на всех машинах
Без моков	Проверяется реальное поведение БД (индексы, триггеры, транзакции)

2.3. Зависимости для TestContainers

```
<dependencies>
    <!-- Spring Boot Test -->
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-test</artifactId>
        <scope>test</scope>
    </dependency>

    <!-- TestContainers Core -->
    <dependency>
        <groupId>org.testcontainers</groupId>
        <artifactId>testcontainers</artifactId>
        <version>1.19.3</version>
        <scope>test</scope>
    </dependency>

    <!-- TestContainers PostgreSQL -->
    <dependency>
        <groupId>org.testcontainers</groupId>
        <artifactId>postgresql</artifactId>
        <version>1.19.3</version>
        <scope>test</scope>
    </dependency>

    <!-- TestContainers JUnit 5 integration -->
    <dependency>
        <groupId>org.testcontainers</groupId>
        <artifactId>junit-jupiter</artifactId>
```

```
        <version>1.19.3</version>
        <scope>test</scope>
    </dependency>
</dependencies>
```

2.4. Уровни интеграционного тестирования

Уровень	Что тестируем	Инструменты	Сложность
Репозитории	JPA-запросы, связи, транзакции	TestContainers, Spring Data JPA Test	Средняя
Сервисы	Бизнес-логика с реальной БД	TestContainers, @SpringBootTest	Высокая
Контроллеры	REST API, валидация, исключения	MockMvc, TestContainers, @WebMvcTest	Высокая

2.5. Аннотации для интеграционного тестирования

Аннотация	Назначение	Пример
@SpringBootTes	Загружает	@SpringBootTest

Аннотация	Назначение	Пример
t	полный контекст приложения	
@Testcontainers	Включает поддержку TestContainers	@Testcontainers
@Container	Объявляет контейнер	@Container static PostgreSQLContainer<?> postgres = ...
@DynamicPropertySource	Динамическая настройка свойств	@DynamicPropertySource
@Transactional	Откат транзакции после теста	@Transactional
@AutoConfigureMockMvc	Настройка MockMvc для тестирования контроллеров	@AutoConfigureMockMvc

3. ПРИМЕР ВЫПОЛНЕНИЯ (СКВОЗНОЙ ПРОЕКТ)

Проект: «Система управления мероприятиями» (Event Management System)

3.1. Базовая конфигурация TestContainers

AbstractIntegrationTest.java:

```
package ru.edu.project.test;

import org.springframework.boot.test.context.SpringBootTest;
import org.springframework.test.context.DynamicPropertyRegistry;
import org.springframework.test.context.DynamicPropertySource;
import org.springframework.test.context.TestConstructor;
import org.springframework.transaction.annotation.Transactional;
import org.testcontainers.containers.PostgreSQLContainer;
import org.testcontainers.junit.jupiter.Container;
import org.testcontainers.junit.jupiter.Testcontainers;

@Testcontainers
@SpringBootTest(webEnvironment =
SpringBootTest.WebEnvironment.RANDOM_PORT)
@Transactional
@TestConstructor(autowireMode = TestConstructor.AutowireMode.ALL)
public abstract class AbstractIntegrationTest {

    @Container
    protected static final PostgreSQLContainer<?>
POSTGRES_CONTAINER =
            new PostgreSQLContainer<>("postgres:15")
                    .withDatabaseName("testdb")
                    .withUsername("testuser")
                    .withPassword("testpass");

    @DynamicPropertySource
    static void registerPgProperties(DynamicPropertyRegistry registry) {
```

```
        registry.add("spring.datasource.url",
POSTGRES_CONTAINER::getJdbcUrl);
        registry.add("spring.datasource.username",
POSTGRES_CONTAINER::getUsername);
        registry.add("spring.datasource.password",
POSTGRES_CONTAINER::getPassword);
        registry.add("spring.jpa.hibernate.ddl-auto", () -> "create-drop");
        registry.add("spring.jpa.show-sql", () -> "true");
        registry.add("spring.jpa.properties.hibernate.format_sql", () -> "true");
    }
}
```

3.2. Тестирование репозиториев

EventRepositoryIntegrationTest.java:

```
package ru.edu.project.foundation.repositories;

import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.DisplayName;
import org.junit.jupiter.api.Test;
import org.springframework.beans.factory.annotation.Autowired;
import ru.edu.project.entity.Category;
import ru.edu.project.entity.Event;
import ru.edu.project.entity.User;
import ru.edu.project.test.AbstractIntegrationTest;

import java.math.BigDecimal;
import java.time.LocalDateTime;
import java.util.List;
```

```
import static org.junit.jupiter.api.Assertions.*;

@DisplayName("Интеграционные тесты EventRepository")
class EventRepositoryIntegrationTest extends AbstractIntegrationTest {

    @Autowired
    private EventRepository eventRepository;

    @Autowired
    private UserRepository userRepository;

    @Autowired
    private CategoryRepository categoryRepository;

    private User testUser;
    private Category testCategory;

    @BeforeEach
    void setUp() {
        // Создание тестового пользователя
        testUser = User.builder()
                .email("test@example.com")
                .passwordHash("hashed_password")
                .name("Test User")
                .role("USER")
                .build();
        testUser = userRepository.save(testUser);

        // Создание тестовой категории
```

```
    testCategory = Category.builder()
            .name("Конференция")
            .description("Деловые конференции")
            .build();
    testCategory = categoryRepository.save(testCategory);
}

@Test
@DisplayName("Сохранение мероприятия должно присвоить ID")
void saveEventShouldAssignId() {
    // given
    Event event = Event.builder()
            .title("Test Event")
            .description("Description")
            .startDate(LocalDateTime.now().plusDays(7))
            .endDate(LocalDateTime.now().plusDays(8))
            .maxParticipants(100)
            .status("DRAFT")
            .basePrice(new BigDecimal("5000"))
            .category(testCategory)
            .createdBy(testUser)
            .build();

    // when
    Event saved = eventRepository.save(event);

    // then
    assertNotNull(saved.getId());
    assertEquals("Test Event", saved.getTitle());
```

```
        assertEquals(testCategory.getId(), saved.getCategory().getId());
        assertEquals(testUser.getId(), saved.getCreatedBy().getId());
    }

    @Test
    @DisplayName("Поиск мероприятия по ID должен вернуть корректные данные")
    void findEventByIdShouldReturnCorrectData() {
        // given
        Event event = Event.builder()
                .title("Findable Event")
                .startDate(LocalDateTime.now().plusDays(7))
                .endDate(LocalDateTime.now().plusDays(8))
                .maxParticipants(50)
                .status("PUBLISHED")
                .createdBy(testUser)
                .build();
        Event saved = eventRepository.save(event);

        // when
        Event found = eventRepository.findById(saved.getId()).orElse(null);

        // then
        assertNotNull(found);
        assertEquals("Findable Event", found.getTitle());
        assertEquals("PUBLISHED", found.getStatus());
        assertEquals(50, found.getMaxParticipants());
    }
```

```
    @Test
    @DisplayName("Поиск мероприятий по статусу должен вернуть только нужные")
    void findByStatusShouldReturnOnlyMatchingEvents() {
        // given
        Event draftEvent = Event.builder()
                .title("Draft Event")
                .startDate(LocalDateTime.now().plusDays(7))
                .endDate(LocalDateTime.now().plusDays(8))
                .maxParticipants(10)
                .status("DRAFT")
                .createdBy(testUser)
                .build();

        Event publishedEvent = Event.builder()
                .title("Published Event")
                .startDate(LocalDateTime.now().plusDays(7))
                .endDate(LocalDateTime.now().plusDays(8))
                .maxParticipants(20)
                .status("PUBLISHED")
                .createdBy(testUser)
                .build();

        eventRepository.save(draftEvent);
        eventRepository.save(publishedEvent);

        // when
        List<Event> draftEvents = eventRepository.findByStatus("DRAFT");
```

```
        List<Event> publishedEvents = eventRepository.findByStatus("PUBLISHED");

        // then
        assertEquals(1, draftEvents.size());
        assertEquals("Draft Event", draftEvents.get(0).getTitle());

        assertEquals(1, publishedEvents.size());
        assertEquals("Published Event", publishedEvents.get(0).getTitle());
    }

    @Test
    @DisplayName("Обновление мероприятия должно изменить данные")
    void updateEventShouldChangeData() {
        // given
        Event event = Event.builder()
                .title("Original Title")
                .startDate(LocalDateTime.now().plusDays(7))
                .endDate(LocalDateTime.now().plusDays(8))
                .maxParticipants(30)
                .status("DRAFT")
                .createdBy(testUser)
                .build();
        Event saved = eventRepository.save(event);

        // when
        saved.setTitle("Updated Title");
        saved.setMaxParticipants(50);
```

```
        Event updated = eventRepository.update(saved);

        // then
        assertEquals("Updated Title", updated.getTitle());
        assertEquals(50, updated.getMaxParticipants());
    }

    @Test
    @DisplayName("Удаление мероприятия должно удалить запись из БД")
    void deleteEventShouldRemoveRecord() {
        // given
        Event event = Event.builder()
                .title("To Delete")
                .startDate(LocalDateTime.now().plusDays(7))
                .endDate(LocalDateTime.now().plusDays(8))
                .maxParticipants(10)
                .status("DRAFT")
                .createdBy(testUser)
                .build();
        Event saved = eventRepository.save(event);
        Long id = saved.getId();

        // when
        eventRepository.deleteById(id);

        // then
        assertFalse(eventRepository.findById(id).isPresent());
    }
```

```
    @Test
    @DisplayName("Поиск мероприятий по категории должен работать")
    void findByCategoryIdShouldReturnEvents() {
        // given
        Event event1 = Event.builder()
                .title("Event 1")
                .startDate(LocalDateTime.now().plusDays(7))
                .endDate(LocalDateTime.now().plusDays(8))
                .maxParticipants(10)
                .category(testCategory)
                .createdBy(testUser)
                .build();

        Event event2 = Event.builder()
                .title("Event 2")
                .startDate(LocalDateTime.now().plusDays(7))
                .endDate(LocalDateTime.now().plusDays(8))
                .maxParticipants(20)
                .category(testCategory)
                .createdBy(testUser)
                .build();

        eventRepository.save(event1);
        eventRepository.save(event2);

        // when
        List<Event>                                events                                =
eventRepository.findByCategoryId(testCategory.getId());
```

```
        // then
        assertEquals(2, events.size());
    }
}
```

3.3. Тестирование сервисов (Mediator)

RegistrationServiceIntegrationTest.java:

```
package ru.edu.project.mediator.services;

import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.DisplayName;
import org.junit.jupiter.api.Test;
import org.springframework.beans.factory.annotation.Autowired;
import ru.edu.project.entity.Category;
import ru.edu.project.entity.Event;
import ru.edu.project.entity.User;
import ru.edu.project.foundation.repositories.CategoryRepository;
import ru.edu.project.foundation.repositories.EventRepository;
import ru.edu.project.foundation.repositories.UserRepository;
import ru.edu.project.mediator.dto.RegistrationDto;
import ru.edu.project.mediator.exceptions.DuplicateRegistrationException;
import ru.edu.project.mediator.exceptions.NoSeatsAvailableException;
import ru.edu.project.mediator.interfaces.IRegistrationService;
import ru.edu.project.mediator.strategy.impl.EarlyBirdPricing;
import ru.edu.project.mediator.strategy.impl.GroupPricing;
import ru.edu.project.test.AbstractIntegrationTest;

import java.math.BigDecimal;
import java.time.LocalDateTime;
```

```
import static org.junit.jupiter.api.Assertions.*;

@DisplayName("Интеграционные тесты RegistrationService")
class RegistrationServiceIntegrationTest extends AbstractIntegrationTest {

    @Autowired
    private IRegistrationService registrationService;

    @Autowired
    private UserRepository userRepository;

    @Autowired
    private EventRepository eventRepository;

    @Autowired
    private CategoryRepository categoryRepository;

    private User testUser;
    private Event testEvent;

    @BeforeEach
    void setUp() {
        // Очистка данных перед каждым тестом
        eventRepository.deleteAll();
        userRepository.deleteAll();
        categoryRepository.deleteAll();

        // Создание пользователя
```

```
        testUser = User.builder()
                .email("user" + System.currentTimeMillis() + "@example.com")
                .passwordHash("hashed")
                .name("Test User")
                .role("USER")
                .build();
        testUser = userRepository.save(testUser);

        // Создание категории
        Category category = Category.builder()
                .name("Конференция")
                .build();
        category = categoryRepository.save(category);

        // Создание мероприятия
        testEvent = Event.builder()
                .title("Test Event")
                .startDate(LocalDateTime.now().plusDays(30))
                .endDate(LocalDateTime.now().plusDays(31))
                .maxParticipants(10)
                .status("PUBLISHED")
                .basePrice(new BigDecimal("5000"))
                .category(category)
                .createdBy(testUser)
                .build();
        testEvent = eventRepository.save(testEvent);
    }

    @Test
```

```
    @DisplayName("Регистрация пользователя должна создать запись в БД")
    void registerUserForEventShouldCreateRegistration() {
        // when
        RegistrationDto result = registrationService.registerUserForEvent(
                testUser.getId(), testEvent.getId());

        // then
        assertNotNull(result);
        assertNotNull(result.getId());
        assertEquals(testUser.getId(), result.getUserId());
        assertEquals(testEvent.getId(), result.getEventId());
        assertEquals("PENDING", result.getStatus());
    }

    @Test
    @DisplayName("Повторная регистрация должна выбросить исключение")
    void duplicateRegistrationShouldThrowException() {
        // given
        registrationService.registerUserForEvent(testUser.getId(),
testEvent.getId());

        // then
        assertThrows(DuplicateRegistrationException.class, () -> {
            registrationService.registerUserForEvent(testUser.getId(),
testEvent.getId());
        });
    }
```

```
    @Test
    @DisplayName("Регистрация при полном мероприятии должна выбросить исключение")
    void registrationWithFullEventShouldThrowException() {
        // given
        testEvent.setMaxParticipants(1);
        eventRepository.update(testEvent);
        registrationService.registerUserForEvent(testUser.getId(), testEvent.getId());

        // создаём второго пользователя
        User secondUser = User.builder()
                .email("second@example.com")
                .passwordHash("hashed")
                .name("Second User")
                .build();
        secondUser = userRepository.save(secondUser);

        // then
        assertThrows(NoSeatsAvailableException.class, () -> {
            registrationService.registerUserForEvent(secondUser.getId(), testEvent.getId());
        });
    }

    @Test
    @DisplayName("Расчёт цены со стандартной стратегией")
    void calculatePriceWithStandardStrategy() {
```

```
    // given
    BigDecimal price = registrationService.calculatePrice(
            testEvent.getId(), testUser.getId(), 2);

    // then
    assertEquals(new BigDecimal("10000"), price); // 5000 * 2
}

@Test
@DisplayName("Расчёт цены с групповой скидкой")
void calculatePriceWithGroupDiscount() {
    // given
    registrationService.setPricingStrategy(new GroupPricing());

    // when
    BigDecimal price = registrationService.calculatePrice(
            testEvent.getId(), testUser.getId(), 6);

    // then
    assertEquals(new BigDecimal("27000"), price); // 5000 * 6 * 0.9
}

@Test
@DisplayName("Расчёт цены со скидкой за раннее бронирование")
void calculatePriceWithEarlyBirdDiscount() {
    // given
    registrationService.setPricingStrategy(new EarlyBirdPricing());

    // when (тест запущен за 30+ дней до мероприятия)
```

```
        BigDecimal price = registrationService.calculatePrice(
                testEvent.getId(), testUser.getId(), 1);

        // then
        assertEquals(new BigDecimal("4000"), price); // 5000 * 0.8
    }

    @Test
    @DisplayName("Отмена регистрации должна изменить статус")
    void cancelRegistrationShouldChangeStatus() {
        // given
        RegistrationDto registration = registrationService.registerUserForEvent(
                testUser.getId(), testEvent.getId());

        // when
        registrationService.cancelRegistration(registration.getId());

        // then
        RegistrationDto cancelled = registrationService.getRegistrationById(registration.getId());
        assertEquals("CANCELLED", cancelled.getStatus());
    }

    @Test
    @DisplayName("Получение регистраций пользователя")
    void getRegistrationsByUserShouldReturnList() {
        // given
```

```
        registrationService.registerUserForEvent(testUser.getId(),
testEvent.getId());

        // when
        var registrations =
registrationService.getRegistrationsByUser(testUser.getId());

        // then
        assertEquals(1, registrations.size());
        assertEquals(testEvent.getId(), registrations.get(0).getEventId());
    }
}
```

3.4. Тестирование контроллеров (REST API)

EventControllerIntegrationTest.java:

```
package ru.edu.project.control.controllers;

import com.fasterxml.jackson.databind.ObjectMapper;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.DisplayName;
import org.junit.jupiter.api.Test;
import org.springframework.beans.factory.annotation.Autowired;
import
org.springframework.boot.test.autoconfigure.web.servlet.AutoConfigureMockMvc
;
import org.springframework.http.MediaType;
import org.springframework.test.web.servlet.MockMvc;
import ru.edu.project.control.dto.request.CreateEventRequest;
import ru.edu.project.entity.Event;
import ru.edu.project.entity.User;
```

```
import ru.edu.project.foundation.repositories.EventRepository;
import ru.edu.project.foundation.repositories.UserRepository;
import ru.edu.project.test.AbstractIntegrationTest;

import java.time.LocalDateTime;

import static org.hamcrest.Matchers.*;
import static org.springframework.test.web.servlet.request.MockMvcRequestBuilders.*;
import static org.springframework.test.web.servlet.result.MockMvcResultMatchers.*;

@AutoConfigureMockMvc
@DisplayName("Интеграционные тесты EventController")
class EventControllerIntegrationTest extends AbstractIntegrationTest {

    @Autowired
    private MockMvc mockMvc;

    @Autowired
    private ObjectMapper objectMapper;

    @Autowired
    private EventRepository eventRepository;

    @Autowired
    private UserRepository userRepository;

    private User testUser;
```

```
private Event testEvent;

@BeforeEach
void setUp() {
    eventRepository.deleteAll();
    userRepository.deleteAll();

    testUser = User.builder()
            .email("admin@example.com")
            .passwordHash("hashed")
            .name("Admin")
            .role("ADMIN")
            .build();
    testUser = userRepository.save(testUser);

    testEvent = Event.builder()
            .title("Initial Event")
            .description("Initial description")
            .startDate(LocalDateTime.now().plusDays(7))
            .endDate(LocalDateTime.now().plusDays(8))
            .maxParticipants(100)
            .status("DRAFT")
            .createdBy(testUser)
            .build();
    testEvent = eventRepository.save(testEvent);
}

@Test
```

```
    @DisplayName("GET /api/events должен вернуть список мероприятий")
    void getAllEventsShouldReturnList() throws Exception {
        mockMvc.perform(get("/api/events"))
                .andExpect(status().isOk())
                .andExpect(jsonPath("$", hasSize(1)))
                .andExpect(jsonPath("$[0].title").value("Initial Event"));
    }

    @Test
    @DisplayName("GET /api/events/{id} должен вернуть мероприятие по ID")
    void getEventByIdShouldReturnEvent() throws Exception {
        mockMvc.perform(get("/api/events/{id}", testEvent.getId()))
                .andExpect(status().isOk())
                .andExpect(jsonPath("$.id").value(testEvent.getId()))
                .andExpect(jsonPath("$.title").value("Initial Event"));
    }

    @Test
    @DisplayName("GET /api/events/{id} с несуществующим ID должен вернуть 404")
    void getEventByIdNotFoundShouldReturn404() throws Exception {
        mockMvc.perform(get("/api/events/{id}", 9999L))
                .andExpect(status().isNotFound());
    }

    @Test
    @DisplayName("POST /api/events должен создать мероприятие")
```

```
    void createEventShouldCreateNewEvent() throws Exception {
        CreateEventRequest request = new CreateEventRequest();
        request.setTitle("New Event");
        request.setDescription("New description");
        request.setStartDate(LocalDateTime.now().plusDays(14));
        request.setEndDate(LocalDateTime.now().plusDays(15));
        request.setMaxParticipants(200);

        mockMvc.perform(post("/api/events")
                .contentType(MediaType.APPLICATION_JSON)
                .content(objectMapper.writeValueAsString(request)))
                .andExpect(status().isCreated())
                .andExpect(jsonPath("$.title").value("New Event"))
                .andExpect(jsonPath("$.maxParticipants").value(200));
    }

    @Test
    @DisplayName("POST /api/events с невалидными данными должен вернуть 400")
    void createEventWithInvalidDataShouldReturn400() throws Exception {
        CreateEventRequest request = new CreateEventRequest(); // пустой запрос

        mockMvc.perform(post("/api/events")
                .contentType(MediaType.APPLICATION_JSON)
                .content(objectMapper.writeValueAsString(request)))
                .andExpect(status().isBadRequest());
    }
```

```
    @Test
    @DisplayName("PUT /api/events/{id} должен обновить мероприятие")
    void updateEventShouldUpdateExistingEvent() throws Exception {
        CreateEventRequest request = new CreateEventRequest();
        request.setTitle("Updated Title");
        request.setDescription("Updated description");
        request.setStartDate(LocalDateTime.now().plusDays(7));
        request.setEndDate(LocalDateTime.now().plusDays(8));
        request.setMaxParticipants(150);

        mockMvc.perform(put("/api/events/{id}", testEvent.getId())
                .contentType(MediaType.APPLICATION_JSON)
                .content(objectMapper.writeValueAsString(request)))
                .andExpect(status().isOk())
                .andExpect(jsonPath("$.title").value("Updated Title"))
                .andExpect(jsonPath("$.maxParticipants").value(150));
    }

    @Test
    @DisplayName("DELETE /api/events/{id} должен удалить мероприятие")
    void deleteEventShouldRemoveEvent() throws Exception {
        mockMvc.perform(delete("/api/events/{id}", testEvent.getId()))
                .andExpect(status().isNoContent());

        // Проверка, что мероприятие действительно удалено
        mockMvc.perform(get("/api/events/{id}", testEvent.getId()))
                .andExpect(status().isNotFound());
    }
```

```
    @Test
    @DisplayName("PATCH /api/events/{id}/publish должен опубликовать мероприятие")
    void publishEventShouldChangeStatus() throws Exception {
        mockMvc.perform(patch("/api/events/{id}/publish", testEvent.getId()))
                .andExpect(status().isOk());

        mockMvc.perform(get("/api/events/{id}", testEvent.getId()))
                .andExpect(status().isOk())
                .andExpect(jsonPath("$.status").value("PUBLISHED"));
    }
}
```

3.5. Конфигурация для тестов (application-test.yml)

```
spring:
  jpa:
    hibernate:
      ddl-auto: create-drop
    show-sql: true
    properties:
      hibernate:
        format_sql: true
        dialect: org.hibernate.dialect.PostgreSQLDialect
  datasource:
    driver-class-name: org.testcontainers.jdbc.ContainerDatabaseDriver

logging:
  level:
```

```
org.springframework.boot.test: DEBUG
org.testcontainers: INFO
```

4. МЕТОДИКА ВЫПОЛНЕНИЯ

Шаг 1: Добавление зависимостей

Добавить TestContainers зависимости в pom.xml

Добавить spring-boot-starter-test

Шаг 2: Создание базового класса для тестов

Создать AbstractIntegrationTest

Настроить PostgreSQLContainer

Настроить @DynamicPropertySource

Шаг 3: Тестирование репозиториев

Написать тесты для CRUD операций

Написать тесты для поисковых методов

Проверить каскадные операции

Шаг 4: Тестирование сервисов

Написать тесты для бизнес-логики

Проверить транзакционность

Проверить бизнес-исключения

Шаг 5: Тестирование контроллеров

Настроить @AutoConfigureMockMvc

Написать тесты для всех эндпоинтов

5. КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Чем интеграционное тестирование отличается от модульного?
2. Что такое TestContainers и зачем он нужен?
3. Как TestContainers помогает тестировать реальную базу данных?
4. Какие аннотации используются для настройки TestContainers?
5. Как TestContainers управляет жизненным циклом Docker-контейнера?
6. Как настроить @DynamicPropertySource для подключения к контейнеру?
7. В чём преимущество TestContainers перед H2 для интеграционных тестов?

6. ТРЕБОВАНИЯ К ОТЧЕТУ

Структура отчета:

Титульный лист

Содержание

Введение

Настройка TestContainers

Тесты репозиториев (листинг)

Тесты сервисов (листинг)

Тесты контроллеров (листинг)

Заключение

7. КРИТЕРИИ ОЦЕНИВАНИЯ

Критерий	Макс. балл	Описание
Настройка TestContainers	3	Корректная конфигурация

Критерий	Макс. балл	Описание
Тесты репозиториев	7	CRUD, поиск, каскады
Тесты сервисов	7	Бизнес-логика, транзакции
Тесты контроллеров	5	REST API, статусы
Оформление отчета	2	Соответствие требованиям
ИТОГО	24	

Лабораторная работа №18

Настройка JWT аутентификации в Spring Security

1. Цель и задачи работы

Цель: Освоить настройку JWT (JSON Web Token) аутентификации в Spring Security для защиты REST API и обеспечения безопасности веб-приложения.

Задачи:

1. Изучить теоретические основы JWT и Spring Security
2. Настроить Spring Security с JWT аутентификацией
3. Реализовать генерацию и валидацию JWT токенов
4. Создать эндпоинты для логина и регистрации

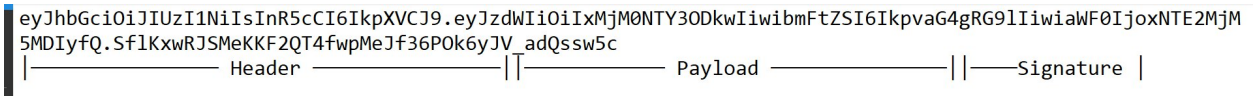
5. Настроить защиту эндпоинтов на основе ролей
6. Реализовать фильтр для проверки JWT токенов
7. Написать интеграционные тесты для аутентификации

2. ТЕОРЕТИЧЕСКОЕ ОБОСНОВАНИЕ

2.1. Что такое JWT?

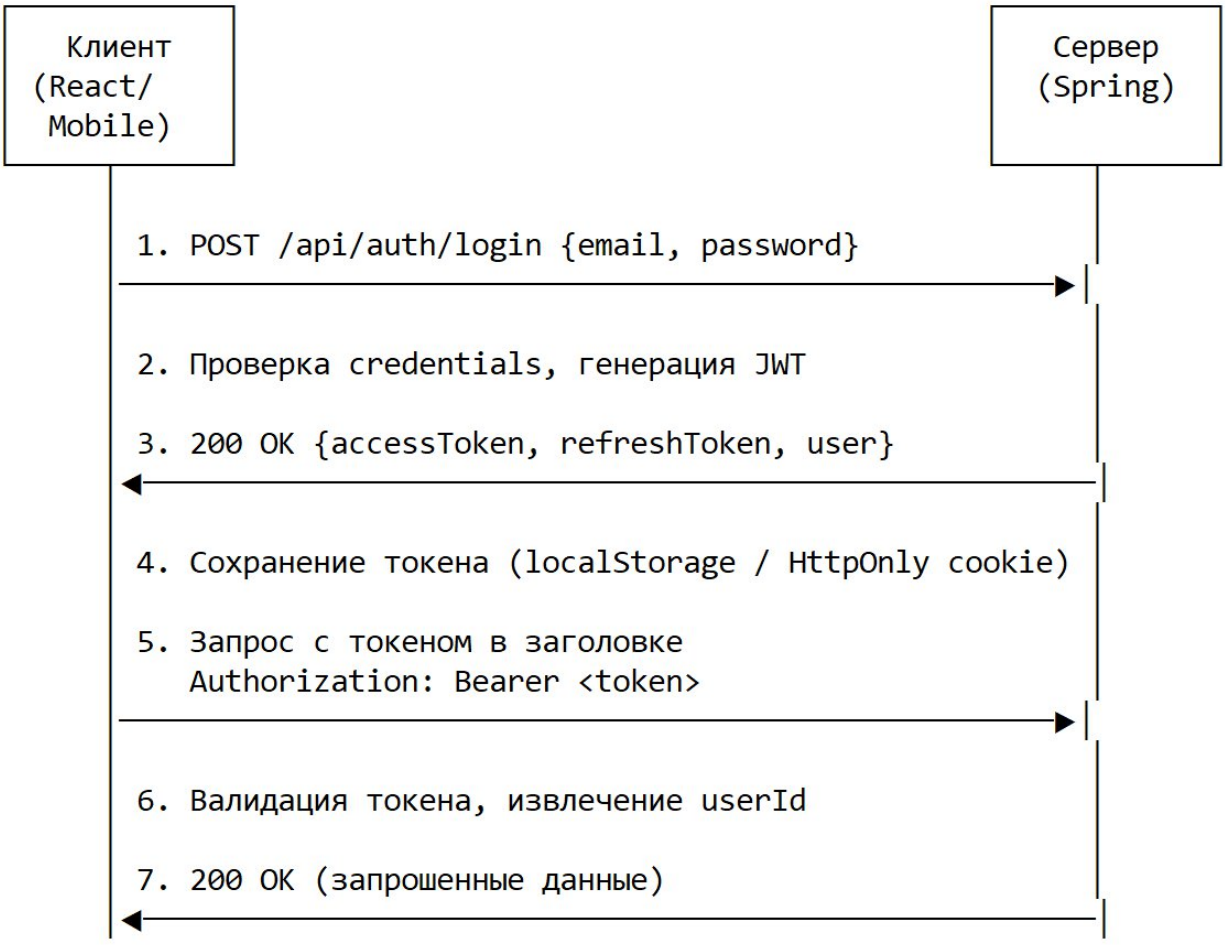
JWT (JSON Web Token) - это открытый стандарт (RFC 7519) для создания токенов доступа, который позволяет безопасно передавать информацию между сторонами в виде JSON-объекта.

Структура JWT:



Часть	Описание	Пример
Header	Алгоритм шифрования и тип токена	{"alg": "HS256", "typ": "JWT"}
Payload	Данные (claims) — userId, роли, срок действия	{"sub": "1", "role": "USER", "exp": 1735689600}
Signature	Подпись, подтверждающая подлинность токена	HMACSHA256(base64UrlEncode(header) + "." + base64UrlEncode(payload), secret)

2.2. Как работает JWT аутентификация?



2.3. Зависимости

```
<dependencies>
    <!-- Spring Security -->
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-security</artifactId>
    </dependency>

    <!-- JWT (JJWT - Java JWT) -->
    <dependency>
        <groupId>io.jsonwebtoken</groupId>
        <artifactId>jjwt-api</artifactId>
        <version>0.11.5</version>
```

```
    </dependency>
    <dependency>
        <groupId>io.jsonwebtoken</groupId>
        <artifactId>jjwt-impl</artifactId>
        <version>0.11.5</version>
        <scope>runtime</scope>
    </dependency>
    <dependency>
        <groupId>io.jsonwebtoken</groupId>
        <artifactId>jjwt-jackson</artifactId>
        <version>0.11.5</version>
        <scope>runtime</scope>
    </dependency>
</dependencies>
```

2.4. Компоненты JWT аутентификации

Компонент	Назначение
JwtUtils	Генерация, валидация, извлечение данных из токена
JwtAuthenticationFilter	Фильтр, проверяющий наличие и валидность JWT
UserDetailsService	Загрузка пользователя по email/username
SecurityConfig	Конфигурация Spring Security (цепочка фильтров, разрешенные эндпоинты)
AuthController	Эндпоинты для логина и регистрации

3. ПРИМЕР ВЫПОЛНЕНИЯ (СКВОЗНОЙ ПРОЕКТ)

3.1. JWT Utils

JwtUtils.java:

```
package ru.edu.project.security;

import io.jsonwebtoken.*;
import io.jsonwebtoken.security.Keys;
import lombok.extern.slf4j.Slf4j;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.security.core.Authentication;
import org.springframework.stereotype.Component;
import ru.edu.project.security.services.UserDetailsImpl;

import java.security.Key;
import java.util.Date;

@Slf4j
@Component
public class JwtUtils {

    @Value("${app.jwt.secret}")
    private String jwtSecret;

    @Value("${app.jwt.expiration-ms}")
    private int jwtExpirationMs;

    @Value("${app.jwt.refresh-expiration-ms}")
```

```
    private int jwtRefreshExpirationMs;

    private Key key() {
        return Keys.hmacShaKeyFor(jwtSecret.getBytes());
    }

    /**
     * Генерация access токена
     */
    public String generateAccessToken(Authentication authentication) {
        UserDetailsImpl userPrincipal = (UserDetailsImpl) authentication.getPrincipal();

        return Jwts.builder()
                .setSubject(userPrincipal.getId().toString())
                .claim("email", userPrincipal.getEmail())
                .claim("role", userPrincipal.getRole())
                .setIssuedAt(new Date())
                .setExpiration(new Date(System.currentTimeMillis() + jwtExpirationMs))
                .signWith(key(), SignatureAlgorithm.HS256)
                .compact();
    }

    /**
     * Генерация refresh токена
     */
    public String generateRefreshToken(Long userId) {
        return Jwts.builder()
```

```
                .setSubject(userId.toString())
                .setIssuedAt(new Date())
                .setExpiration(new Date(System.currentTimeMillis() + jwtRefreshExpirationMs))
                .claim("type", "refresh")
                .signWith(key(), SignatureAlgorithm.HS256)
                .compact();
    }

    /**
     * Генерация токена из UserDetails (для регистрации)
     */
    public String generateTokenFromUserDetails(UserDetailsImpl userDetails) {
        return Jwts.builder()
                .setSubject(userDetails.getId().toString())
                .claim("email", userDetails.getEmail())
                .claim("role", userDetails.getRole())
                .setIssuedAt(new Date())
                .setExpiration(new Date(System.currentTimeMillis() + jwtExpirationMs))
                .signWith(key(), SignatureAlgorithm.HS256)
                .compact();
    }

    /**
     * Извлечение userId из токена
     */
    public Long getUserIdFromToken(String token) {
```

```
    Claims claims = Jwts.parserBuilder()
            .setSigningKey(key())
            .build()
            .parseClaimsJws(token)
            .getBody();
    return Long.parseLong(claims.getSubject());
}

/**
 * Извлечение email из токена
 */
public String getEmailFromToken(String token) {
    Claims claims = Jwts.parserBuilder()
            .setSigningKey(key())
            .build()
            .parseClaimsJws(token)
            .getBody();
    return claims.get("email", String.class);
}

/**
 * Извлечение роли из токена
 */
public String getRoleFromToken(String token) {
    Claims claims = Jwts.parserBuilder()
            .setSigningKey(key())
            .build()
            .parseClaimsJws(token)
            .getBody();
```

```
        return claims.get("role", String.class);
    }

    /**
     * Валидация токена
     */
    public boolean validateToken(String authToken) {
        try {
            Jwts.parserBuilder().setSigningKey(key()).build().parseClaimsJws(authToken);
            return true;
        } catch (SecurityException e) {
            log.error("Invalid JWT signature: {}", e.getMessage());
        } catch (MalformedJwtException e) {
            log.error("Invalid JWT token: {}", e.getMessage());
        } catch (ExpiredJwtException e) {
            log.error("JWT token is expired: {}", e.getMessage());
        } catch (UnsupportedJwtException e) {
            log.error("JWT token is unsupported: {}", e.getMessage());
        } catch (IllegalArgumentException e) {
            log.error("JWT claims string is empty: {}", e.getMessage());
        }
        return false;
    }
}
```

3.2. UserDetailsImpl (Principal)

UserDetailsImpl.java:

```
package ru.edu.project.security.services;
```

```
import com.fasterxml.jackson.annotation.JsonIgnore;
import lombok.AllArgsConstructor;
import lombok.Getter;
import org.springframework.security.core.GrantedAuthority;
import
org.springframework.security.core.authority.SimpleGrantedAuthority;
import org.springframework.security.core.userdetails.UserDetails;
import ru.edu.project.entity.User;

import java.util.Collection;
import java.util.Collections;
import java.util.Objects;

@Getter
@AllArgsConstructor
public class UserDetailsImpl implements UserDetails {

    private static final long serialVersionUID = 1L;

    private Long id;
    private String email;
    private String name;
    @JsonIgnore
    private String password;
    private String role;

    public static UserDetailsImpl build(User user) {
        return new UserDetailsImpl(
```

```
                user.getId(),
                user.getEmail(),
                user.getName(),
                user.getPasswordHash(),
                user.getRole()
        );
    }

    @Override
    public Collection<? extends GrantedAuthority> getAuthorities() {
        return Collections.singletonList(new SimpleGrantedAuthority("ROLE_" + role));
    }

    @Override
    public String getPassword() {
        return password;
    }

    @Override
    public String getUsername() {
        return email;
    }

    @Override
    public boolean isAccountNonExpired() {
        return true;
    }
```

```
    @Override
    public boolean isAccountNonLocked() {
        return true;
    }

    @Override
    public boolean isCredentialsNonExpired() {
        return true;
    }

    @Override
    public boolean isEnabled() {
        return true;
    }

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;
        UserDetailsImpl that = (UserDetailsImpl) o;
        return Objects.equals(id, that.id);
    }

    @Override
    public int hashCode() {
        return Objects.hash(id);
    }
}
```

3.3. UserDetailsService

UserDetailsServiceImpl.java:

```
package ru.edu.project.security.services;

import lombok.RequiredArgsConstructor;
import lombok.extern.slf4j.Slf4j;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.security.core.userdetails.UserDetailsService;
import
org.springframework.security.core.userdetails.UsernameNotFoundException;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;
import ru.edu.project.entity.User;
import ru.edu.project.foundation.repositories.UserRepository;

@Slf4j
@Service
@RequiredArgsConstructor
public class UserDetailsServiceImpl implements UserDetailsService {

    private final UserRepository userRepository;

    @Override
    @Transactional(readOnly = true)
    public UserDetails loadUserByUsername(String email) throws
UsernameNotFoundException {
        log.debug("Loading user by email: {}", email);

        User user = userRepository.findByEmail(email)
```

```
                .orElseThrow(() -> new UsernameNotFoundException("User not
found with email: " + email));

        return UserDetailsImpl.build(user);
    }
}
```

3.4. JWT Authentication Filter

AuthTokenFilter.java:

```
package ru.edu.project.security;

import jakarta.servlet.FilterChain;
import jakarta.servlet.ServletException;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;
import lombok.RequiredArgsConstructor;
import lombok.extern.slf4j.Slf4j;
import
org.springframework.security.authentication.UsernamePasswordAuthenticationTok
en;
import org.springframework.security.core.context.SecurityContextHolder;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.security.core.userdetails.UserDetailsService;
import
org.springframework.security.web.authentication.WebAuthenticationDetailsSource
;
import org.springframework.stereotype.Component;
import org.springframework.util.StringUtils;
import org.springframework.web.filter.OncePerRequestFilter;
```

```
import java.io.IOException;

@Slf4j
@Component
@RequiredArgsConstructor
public class AuthTokenFilter extends OncePerRequestFilter {

    private final JwtUtils jwtUtils;
    private final UserDetailsService userDetailsService;

    @Override
    protected void doFilterInternal(HttpServletRequest request,
                                    HttpServletResponse response,
                                    FilterChain filterChain)
            throws ServletException, IOException {

        try {
            String jwt = parseJwt(request);
            if (jwt != null && jwtUtils.validateToken(jwt)) {
                Long userId = jwtUtils.getUserIdFromToken(jwt);
                String email = jwtUtils.getEmailFromToken(jwt);

                UserDetails userDetails =
userDetailsService.loadUserByUsername(email);

                UsernamePasswordAuthenticationToken authentication =
                        new UsernamePasswordAuthenticationToken(userDetails,
null, userDetails.getAuthorities());
```

```
                authentication.setDetails(new
WebAuthenticationDetailsSource().buildDetails(request));

SecurityContextHolder.getContext().setAuthentication(authentication);
                log.debug("User authenticated: {}", email);
            }
        } catch (Exception e) {
            log.error("Cannot set user authentication: {}", e.getMessage());
        }

        filterChain.doFilter(request, response);
    }

    private String parseJwt(HttpServletRequest request) {
        String headerAuth = request.getHeader("Authorization");

        if (StringUtils.hasText(headerAuth) && headerAuth.startsWith("Bearer
")) {
            return headerAuth.substring(7);
        }

        return null;
    }
}
```

3.5. Security Configuration

SecurityConfig.java:

```
package ru.edu.project.security.config;
```

```
import lombok.RequiredArgsConstructor;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.security.authentication.AuthenticationManager;
import
org.springframework.security.authentication.dao.DaoAuthenticationProvider;
import
org.springframework.security.config.annotation.authentication.configuration.Authe
nticationConfiguration;
import
org.springframework.security.config.annotation.method.configuration.EnableMeth
odSecurity;
import
org.springframework.security.config.annotation.web.builders.HttpSecurity;
import
org.springframework.security.config.annotation.web.configuration.EnableWebSec
urity;
import org.springframework.security.config.http.SessionCreationPolicy;
import org.springframework.security.core.userdetails.UserDetailsService;
import
org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.security.web.SecurityFilterChain;
import
org.springframework.security.web.authentication.UsernamePasswordAuthenticatio
nFilter;
import ru.edu.project.security.AuthTokenFilter;
```

```
@Configuration
@EnableWebSecurity
@EnableMethodSecurity(prePostEnabled = true)
@RequiredArgsConstructor
public class SecurityConfig {

    private final UserDetailsService userDetailsService;
    private final AuthTokenFilter authTokenFilter;

    @Bean
    public PasswordEncoder passwordEncoder() {
        return new BCryptPasswordEncoder();
    }

    @Bean
    public DaoAuthenticationProvider authenticationProvider() {
        DaoAuthenticationProvider authProvider = new
DaoAuthenticationProvider();
        authProvider.setUserDetailsService(userDetailsService);
        authProvider.setPasswordEncoder(passwordEncoder());
        return authProvider;
    }

    @Bean
    public AuthenticationManager
authenticationManager(AuthenticationConfiguration authConfig) throws Exception
{
        return authConfig.getAuthenticationManager();
    }
```

```
    @Bean
    public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {
        http
            .csrf(csrf -> csrf.disable())
            .cors(cors -> cors.configure(http))
            .sessionManagement(session -> session.sessionCreationPolicy(SessionCreationPolicy.STATELESS))
            .authorizeHttpRequests(auth -> auth
                // Публичные эндпоинты
                .requestMatchers("/api/auth/**").permitAll()
                .requestMatchers("/swagger-ui/**", "/v3/api-docs/**").permitAll()
                // Административные эндпоинты
                .requestMatchers("/api/admin/**").hasRole("ADMIN")
                // Пользовательские эндпоинты
                .anyRequest().authenticated()
            )
            .authenticationProvider(authenticationProvider())
            .addFilterBefore(authTokenFilter, UsernamePasswordAuthenticationFilter.class);

        return http.build();
    }
}
```

3.6. PasswordEncoder утилита (для хеширования при регистрации)

PasswordUtils.java:

```
package ru.edu.project.security;
```

```
import
org springframework security crypto bcrypt BCryptPasswordEncoder;
import org springframework security crypto password PasswordEncoder;
import org springframework stereotype Component;

@Component
public class PasswordUtils {

    private final PasswordEncoder passwordEncoder = new
BCryptPasswordEncoder();

    public String encodePassword(String rawPassword) {
        return passwordEncoder encode(rawPassword);
    }

    public boolean matches(String rawPassword, String encodedPassword) {
        return passwordEncoder matches(rawPassword, encodedPassword);
    }
}
```

3.7. Auth Controller (обновлённый)

AuthController.java:

```
package ru edu project control controllers;

import jakarta validation Valid;
import lombok RequiredArgsConstructor;
import lombok extern slf4j Slf4j;
import org springframework http HttpStatus;
```

```
import org.springframework.http.ResponseEntity;
import org.springframework.security.authentication.AuthenticationManager;
import
org.springframework.security.authentication.UsernamePasswordAuthenticationTok
en;
import org.springframework.security.core.Authentication;
import org.springframework.security.core.context.SecurityContextHolder;
import org.springframework.web.bind.annotation.*;
import ru.edu.project.control.dto.request.LoginRequest;
import ru.edu.project.control.dto.request.RegisterRequest;
import ru.edu.project.control.dto.response.AuthResponse;
import ru.edu.project.entity.User;
import ru.edu.project.foundation.repositories.UserRepository;
import ru.edu.project.mediator.exceptions.BusinessException;
import ru.edu.project.security.JwtUtils;
import ru.edu.project.security.PasswordUtils;
import ru.edu.project.security.services.UserDetailsImpl;

@Slf4j
@RestController
@RequestMapping("/api/auth")
@RequiredArgsConstructor
public class AuthController {

    private final AuthenticationManager authenticationManager;
    private final JwtUtils jwtUtils;
    private final UserRepository userRepository;
    private final PasswordUtils passwordUtils;
```

```
    @PostMapping("/login")
    public ResponseEntity<AuthResponse> authenticateUser(@Valid
@RequestBody LoginRequest request) {
        log.info("Login attempt for user: {}", request.getEmail());

        Authentication authentication = authenticationManager.authenticate(
            new UsernamePasswordAuthenticationToken(request.getEmail(),
request.getPassword())
        );

        SecurityContextHolder.getContext().setAuthentication(authentication);

        String accessToken = jwtUtils.generateAccessToken(authentication);
        String refreshToken =
jwtUtils.generateRefreshToken(getUserIdFromAuthentication(authentication));

        UserDetailsImpl userDetails = (UserDetailsImpl)
authentication.getPrincipal();

        return ResponseEntity.ok(new AuthResponse(accessToken,
refreshToken, userDetails));
    }

    @PostMapping("/register")
    public ResponseEntity<AuthResponse> registerUser(@Valid
@RequestBody RegisterRequest request) {
        log.info("Registration attempt for user: {}", request.getEmail());

        // Проверка существования email
```

```
        if (userRepository.existsByEmail(request.getEmail())) {
            throw new BusinessException("Email already exists: " +
request.getEmail());
        }

        // Создание нового пользователя
        User user = User.builder()
                .email(request.getEmail())
                .passwordHash(passwordUtils.encodePassword(request.getPasswo
rd()))
                .name(request.getName())
                .role("USER")  // по умолчанию роль USER
                .build();

        User savedUser = userRepository.save(user);
        log.info("User registered successfully: {}", savedUser.getEmail());

        // Автоматическая аутентификация после регистрации
        Authentication authentication = authenticationManager.authenticate(
                new UsernamePasswordAuthenticationToken(request.getEmail(),
request.getPassword())
        );

        SecurityContextHolder.getContext().setAuthentication(authentication);

        String accessToken = jwtUtils.generateAccessToken(authentication);
        String refreshToken =
jwtUtils.generateRefreshToken(savedUser.getId());
```

```
        UserDetailsImpl userDetails = UserDetailsImpl.build(savedUser);

        return ResponseEntity.status(HttpStatus.CREATED)
                .body(new AuthResponse(accessToken, refreshToken, userDetails));
    }

    @PostMapping("/refresh")
    public ResponseEntity<AuthResponse> refreshToken(@RequestHeader("Authorization") String refreshToken) {
        log.debug("Token refresh request");

        String token = refreshToken.substring(7); // "Bearer " удаляем
        if (!jwtUtils.validateToken(token)) {
            throw new BusinessException("Invalid refresh token");
        }

        Long userId = jwtUtils.getUserIdFromToken(token);
        User user = userRepository.findById(userId)
                .orElseThrow(() -> new BusinessException("User not found"));

        // Генерация нового access токена
        UserDetailsImpl userDetails = UserDetailsImpl.build(user);
        UsernamePasswordAuthenticationToken authentication =
                new UsernamePasswordAuthenticationToken(userDetails, null, userDetails.getAuthorities());

        String newAccessToken = jwtUtils.generateAccessToken(authentication);
```

```
        String newRefreshToken = jwtUtils.generateRefreshToken(userId);

        return ResponseEntity.ok(new AuthResponse(newAccessToken,
newRefreshToken, userDetails));
    }

    private Long getUserIdFromAuthentication(Authentication
authentication) {
        UserDetailsImpl userDetails = (UserDetailsImpl)
authentication.getPrincipal();
        return userDetails.getId();
    }
}
```

3.8. Обновлённые Request/Response DTO

LoginRequest.java:

```
package ru.edu.project.control.dto.request;

import jakarta.validation.constraints.Email;
import jakarta.validation.constraints.NotBlank;
import lombok.Data;

@Data
public class LoginRequest {

    @NotBlank(message = "Email is required")
    @Email(message = "Email should be valid")
    private String email;
```

```
    @NotBlank(message = "Password is required")
    private String password;
}
```

RegisterRequest.java:

```
package ru.edu.project.control.dto.request;

import jakarta.validation.constraints.Email;
import jakarta.validation.constraints.NotBlank;
import jakarta.validation.constraints.Size;
import lombok.Data;

@Data
public class RegisterRequest {

    @NotBlank(message = "Email is required")
    @Email(message = "Email should be valid")
    private String email;

    @NotBlank(message = "Password is required")
    @Size(min = 6, message = "Password must be at least 6 characters")
    private String password;

    @NotBlank(message = "Name is required")
    @Size(min = 2, max = 100, message = "Name must be between 2 and 100 characters")
    private String name;
}
```

AuthResponse.java:

```java
package ru.edu.project.control.dto.response;

import lombok.AllArgsConstructor;
import lombok.Data;
import ru.edu.project.security.services.UserDetailsImpl;

@Data
@AllArgsConstructor
public class AuthResponse {

    private String accessToken;
    private String refreshToken;
    private String tokenType = "Bearer";
    private Long id;
    private String email;
    private String name;
    private String role;

    public AuthResponse(String accessToken, String refreshToken,
UserDetailsImpl userDetails) {
        this.accessToken = accessToken;
        this.refreshToken = refreshToken;
        this.id = userDetails.getId();
        this.email = userDetails.getEmail();
        this.name = userDetails.getName();
        this.role = userDetails.getRole();
    }
}
```

3.9. Защита эндпоинтов с помощью ролей

EventController.java (фрагмент с аннотациями безопасности):

```
@RestController
@RequestMapping("/api/events")
@RequiredArgsConstructor
public class EventController {

    private final IEventService eventService;

    @GetMapping
    @PreAuthorize("hasRole('USER') or hasRole('ADMIN')")
    public ResponseEntity<List<EventResponse>> getAllEvents() {
        // ...
    }

    @PostMapping
    @PreAuthorize("hasRole('ADMIN')")
    public ResponseEntity<EventResponse> createEvent(@Valid
@RequestBody CreateEventRequest request) {
        // ...
    }

    @DeleteMapping("/{id}")
    @PreAuthorize("hasRole('ADMIN')")
    public ResponseEntity<Void> deleteEvent(@PathVariable Long id) {
        // ...
    }
}
```

3.10. Настройки в application.yml

```
app:
  jwt:
    secret:
404E635266556A586E3272357538782F413F4428472B4B6250645367566B5970
    expiration-ms: 3600000          # 1 час
    refresh-expiration-ms: 86400000  # 24 часа

spring:
  security:
    user:
      name: admin
      password: admin
```

3.11. Интеграционные тесты для аутентификации

AuthControllerIntegrationTest.java:

```
package ru edu project control controllers;

import com fasterxml jackson databind ObjectMapper;
import org junit jupiter api BeforeEach;
import org junit jupiter api DisplayName;
import org junit jupiter api Test;
import org springframework beans factory annotation Autowired;
import
org springframework boot test autoconfigure web servlet AutoConfigureMockMvc
;

import org springframework http MediaType;
```

```
import org.springframework.test.web.servlet.MockMvc;
import ru.edu.project.control.dto.request.LoginRequest;
import ru.edu.project.control.dto.request.RegisterRequest;
import ru.edu.project.entity.User;
import ru.edu.project.foundation.repositories.UserRepository;
import ru.edu.project.test.AbstractIntegrationTest;

import static org.hamcrest.Matchers.*;
import static org.springframework.test.web.servlet.request.MockMvcRequestBuilders.*;
import static org.springframework.test.web.servlet.result.MockMvcResultMatchers.*;

@AutoConfigureMockMvc
@DisplayName("Интеграционные тесты AuthController")
class AuthControllerIntegrationTest extends AbstractIntegrationTest {

    @Autowired
    private MockMvc mockMvc;

    @Autowired
    private ObjectMapper objectMapper;

    @Autowired
    private UserRepository userRepository;

    @BeforeEach
    void setUp() {
        userRepository.deleteAll();
```

```
    }

    @Test
    @DisplayName("Регистрация нового пользователя должна вернуть токен")
    void registerShouldReturnToken() throws Exception {
        RegisterRequest request = new RegisterRequest();
        request.setEmail("newuser@example.com");
        request.setPassword("password123");
        request.setName("New User");

        mockMvc.perform(post("/api/auth/register")
                .contentType(MediaType.APPLICATION_JSON)
                .content(objectMapper.writeValueAsString(request)))
                .andExpect(status().isCreated())
                .andExpect(jsonPath("$.accessToken", notNullValue()))
                .andExpect(jsonPath("$.email").value("newuser@example.com"))
                .andExpect(jsonPath("$.role").value("USER"));
    }

    @Test
    @DisplayName("Регистрация с существующим email должна вернуть ошибку")
    void registerWithExistingEmailShouldReturnError() throws Exception {
        // Создание существующего пользователя
        User existingUser = User.builder()
                .email("existing@example.com")
                .passwordHash("hashed")
                .name("Existing")
```

```
                .build();
        userRepository.save(existingUser);

        RegisterRequest request = new RegisterRequest();
        request.setEmail("existing@example.com");
        request.setPassword("password123");
        request.setName("New User");

        mockMvc.perform(post("/api/auth/register")
                .contentType(MediaType.APPLICATION_JSON)
                .content(objectMapper.writeValueAsString(request)))
                .andExpect(status().isBadRequest());
    }

    @Test
    @DisplayName("Логин с корректными данными должен вернуть токен")
    void loginWithValidCredentialsShouldReturnToken() throws Exception {
        // Создание пользователя
        User user = User.builder()
                .email("user@example.com")
                .passwordHash(passwordEncoder.encode("password123"))
                .name("Test User")
                .role("USER")
                .build();
        userRepository.save(user);

        LoginRequest request = new LoginRequest();
        request.setEmail("user@example.com");
```

```
        request.setPassword("password123");

        mockMvc.perform(post("/api/auth/login")
                .contentType(MediaType.APPLICATION_JSON)
                .content(objectMapper.writeValueAsString(request)))
                .andExpect(status().isOk())
                .andExpect(jsonPath("$.accessToken", notNullValue()))
                .andExpect(jsonPath("$.email").value("user@example.com"));
    }

    @Test
    @DisplayName("Логин с неверным паролем должен вернуть ошибку")
    void loginWithInvalidPasswordShouldReturnError() throws Exception {
        User user = User.builder()
                .email("user@example.com")
                .passwordHash(passwordEncoder.encode("correct"))
                .name("Test User")
                .build();
        userRepository.save(user);

        LoginRequest request = new LoginRequest();
        request.setEmail("user@example.com");
        request.setPassword("wrong");

        mockMvc.perform(post("/api/auth/login")
                .contentType(MediaType.APPLICATION_JSON)
                .content(objectMapper.writeValueAsString(request)))
                .andExpect(status().isUnauthorized());
```

```
    }

    @Test
    @DisplayName("Доступ к защищённому эндпоинту без токена должен вернуть 401")
    void accessProtectedEndpointWithoutTokenShouldReturn401() throws Exception {
        mockMvc.perform(get("/api/events"))
                .andExpect(status().isUnauthorized());
    }

    @Test
    @DisplayName("Доступ к защищённому эндпоинту с валидным токеном должен быть разрешён")
    void accessProtectedEndpointWithValidTokenShouldBeAllowed() throws Exception {
        // Регистрация и получение токена
        RegisterRequest registerRequest = new RegisterRequest();
        registerRequest.setEmail("test@example.com");
        registerRequest.setPassword("password123");
        registerRequest.setName("Test User");

        String response = mockMvc.perform(post("/api/auth/register")
                .contentType(MediaType.APPLICATION_JSON)
                .content(objectMapper.writeValueAsString(registerRequest)))
                .andExpect(status().isCreated())
                .andReturn()
                .getResponse()
                .getContentAsString();
```

```
        String token =
objectMapper.readTree(response).get("accessToken").asText();

        // Доступ к защищённому эндпоинту
        mockMvc.perform(get("/api/events")
                .header("Authorization", "Bearer " + token))
                .andExpect(status().isOk());
    }
}
```

4. МЕТОДИКА ВЫПОЛНЕНИЯ

Шаг 1: Добавление зависимостей

Добавить spring-boot-starter-security

Добавить jjwt-api, jjwt-impl, jjwt-jackson

Шаг 2: Создание JWT утилит

Создать JwtUtils (генерация, валидация, извлечение данных)

Создать PasswordUtils (BCrypt хеширование)

Шаг 3: Реализация UserDetailsService

Создать UserDetailsImpl (Principal)

Создать UserDetailsServiceImpl (загрузка пользователя по email)

Шаг 4: Создание JWT фильтра

Создать AuthTokenFilter (извлечение и валидация токена)

Шаг 5: Настройка SecurityConfig

Настроить SecurityFilterChain

Указать публичные эндпоинты (/api/auth/**, /swagger-ui/**)

Настроить защиту эндпоинтов по ролям

Добавить фильтр в цепочку

Шаг 6: Обновление AuthController

Добавить эндпоинт /login

Добавить эндпоинт /register

Добавить эндпоинт /refresh

Шаг 7: Защита эндпоинтов

Добавить @PreAuthorize к методам контроллеров

5. КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Что такое JWT и из каких частей он состоит?
2. В чем разница между access token и refresh token?
3. Как Spring Security интегрируется с JWT?
4. Что делает фильтр AuthTokenFilter?
5. Как настроить SecurityFilterChain для stateless аутентификации?
6. Как хешировать пароли с BCrypt в Spring Security?
7. Как защитить эндпоинты на основе ролей?

6. ТРЕБОВАНИЯ К ОТЧЕТУ

Структура отчета:

Титульный лист

Содержание

Введение

JWT Utils (листинг)

UserDetailsService (листинг)

JWT Filter (листинг)

SecurityConfig (листинг)

AuthController (листинг)

Интеграционные тесты

Заключение

7. КРИТЕРИИ ОЦЕНИВАНИЯ

Критерий	Макс. балл	Описание
JWT Utils	4	Генерация, валидация, извлечение
UserDetailsService	3	Загрузка пользователя
JWT Filter	4	Извлечение и проверка токена
SecurityConfig	4	Настройка безопасности
AuthController	4	Логин, регистрация, refresh
Интеграционные тесты	5	Тесты аутентификации
Оформление отчета	2	Соответствие требованиям
ИТОГО	26	

Лабораторная работа №19

Создание REST API (CRUD для основной сущности)

1. Цель и задачи работы

Цель: Создать полноценный REST API для основной сущности проекта с реализацией всех CRUD-операций (Create, Read, Update, Delete) с использованием Spring MVC, валидацией входных данных и правильными HTTP-статусами.

Задачи:

1. Изучить принципы построения REST API
2. Создать CRUD-эндпоинты для основной сущности (Event / Product / Order / Course)
3. Реализовать правильные HTTP-методы и статусы
4. Добавить валидацию входных данных
5. Реализовать пагинацию и фильтрацию списков
6. Написать интеграционные тесты для всех эндпоинтов
7. Документировать API (Swagger/OpenAPI)

2. ТЕОРЕТИЧЕСКОЕ ОБОСНОВАНИЕ

2.1. Принципы REST API

Принцип	Описание	Пример
Ресурсно-ориентированность	URL обозначают ресурсы (существительные)	/api/events, /api/events/{id}
Использовани	GET, POST,	GET — чтение,

Принцип	Описание	Пример
е HTTP-методов	PUT, DELETE, PATCH	POST — создание
Использовани е HTTP-статусов	200, 201, 204, 400, 404	201 Created — успешное создание
Stateless	Сервер не хранит состояние клиента	Каждый запрос содержит всю информацию
Единообразие интерфейса	Одинаковые паттерны для всех ресурсов	CRUD для всех сущностей

2.2. HTTP-методы и их значение в CRUD

О перация	H TTP-метод	Эндп оинт	Тело запроса	Тело ответа	С татус
C reate	P OST	/api/e vents	Creat eRequest	EventRes ponse	2 01
R ead (all)	G ET	/api/e vents	—	List<Even tResponse>	2 00
R ead	G ET	/api/e vents/{id}	—	EventRes ponse	2 00

Операция	HTTP-метод	Эндпоинт	Тело запроса	Тело ответа	Статус
(one)					
Update (full)	PUT	/api/events/{id}	UpdateRequest	EventResponse	200
Update (partial)	PATCH	/api/events/{id}	PatchRequest	EventResponse	200
Delete	DELETE	/api/events/{id}	—	—	204

2.3. Структура REST API для основной сущности

REST API для Event

```
POST   /api/events                - создание мероприятия
GET    /api/events                - список всех мероприятий
GET    /api/events?page=0&size=10 - список с пагинацией
GET    /api/events?status=PUBLISHED - фильтрация по статусу
GET    /api/events/{id}           - получение по ID
PUT    /api/events/{id}           - полное обновление
PATCH  /api/events/{id}           - частичное обновление
DELETE /api/events/{id}           - удаление
```

2.4. Пагинация и фильтрация

Параметры пагинации:

Параметр	Описание	Пример
page	Номер страницы (с 0)	page=0
size	Размер страницы	size=10
sort	Поле и направление сортировки	sort=startDate,desc

Параметры фильтрации:

Параметр	Описание	Пример
status	Статус мероприятия	status=PUBLISHED
categoryId	ID категории	categoryId=1
startDateFrom	Дата начала от	startDateFrom=2025-01-01T00:00:00

2.5. Зависимости

```
<dependencies>
    <!-- Spring Boot Web -->
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
```

```
        </dependency>

        <!-- Validation -->
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-validation</artifactId>
        </dependency>

        <!-- OpenAPI / Swagger -->
        <dependency>
            <groupId>org.springdoc</groupId>
            <artifactId>springdoc-openapi-starter-webmvc-ui</artifactId>
            <version>2.3.0</version>
        </dependency>
    </dependencies>
```

3. ПРИМЕР ВЫПОЛНЕНИЯ (СКВОЗНОЙ ПРОЕКТ)

Проект: «Система управления мероприятиями» (Event Management System)

3.1. Request DTO

CreateEventRequest.java:

```
package ru.edu.project.control.dto.request;

import jakarta.validation.constraints.*;
import lombok.Data;
import java.time.LocalDateTime;

@Data
public class CreateEventRequest {
```

```
    @NotBlank(message = "Title is required")
    @Size(min = 3, max = 200, message = "Title must be between 3 and 200 characters")
    private String title;

    @Size(max = 5000, message = "Description cannot exceed 5000 characters")
    private String description;

    @NotNull(message = "Start date is required")
    @Future(message = "Start date must be in the future")
    private LocalDateTime startDate;

    @NotNull(message = "End date is required")
    @Future(message = "End date must be in the future")
    private LocalDateTime endDate;

    @NotNull(message = "Max participants is required")
    @Positive(message = "Max participants must be positive")
    @Max(value = 10000, message = "Max participants cannot exceed 10000")
    private Integer maxParticipants;

    @Positive(message = "Price must be positive")
    private java.math.BigDecimal basePrice;

    private Long categoryId
}
```

UpdateEventRequest.java:

```
package ru.edu.project.control.dto.request;

import jakarta.validation.constraints.*;
import lombok.Data;
import java.time.LocalDateTime;

@Data
public class UpdateEventRequest {

    @Size(min = 3, max = 200, message = "Title must be between 3 and 200 characters")
    private String title;

    @Size(max = 5000, message = "Description cannot exceed 5000 characters")
    private String description;

    @Future(message = "Start date must be in the future")
    private LocalDateTime startDate;

    @Future(message = "End date must be in the future")
    private LocalDateTime endDate;

    @Positive(message = "Max participants must be positive")
    @Max(value = 10000, message = "Max participants cannot exceed 10000")
    private Integer maxParticipants;

    @Positive(message = "Price must be positive")
```

```
    private java.math.BigDecimal basePrice;

    private Long categoryId;
}
```

EventFilterRequest.java:

```
package ru.edu.project.control.dto.request;

import lombok.Data;
import java.time.LocalDateTime;

@Data
public class EventFilterRequest {

    private String status;
    private Long categoryId;
    private LocalDateTime startDateFrom;
    private LocalDateTime startDateTo;
    private String search;
}
```

3.2. Response DTO

EventResponse.java:

```
package ru.edu.project.control.dto.response;

import lombok.Builder;
import lombok.Data;
import ru.edu.project.mediator.dto.EventDto;
import java.time.LocalDateTime;

@Data
```

```java
@Builder
public class EventResponse {
    private Long id;
    private String title;
    private String description;
    private LocalDateTime startDate;
    private LocalDateTime endDate;
    private Integer maxParticipants;
    private Integer registeredCount;
    private Integer availableSeats;
    private String status;
    private java.math.BigDecimal basePrice;
    private String categoryName;
    private String createdBy;
    private LocalDateTime createdAt;
    private LocalDateTime updatedAt;

    public static EventResponse fromDto(EventDto dto) {
        if (dto == null) return null;
        return EventResponse.builder()
                .id(dto.getId())
                .title(dto.getTitle())
                .description(dto.getDescription())
                .startDate(dto.getStartDate())
                .endDate(dto.getEndDate())
                .maxParticipants(dto.getMaxParticipants())
                .registeredCount(dto.getRegisteredCount())
                .availableSeats(dto.getAvailableSeats())
                .status(dto.getStatus())
```

```
                .basePrice(dto.getBasePrice())
                .categoryName(dto.getCategoryName())
                .createdBy(dto.getCreatedBy())
                .createdAt(dto.getCreatedAt())
                .updatedAt(dto.getUpdatedAt())
                .build();
    }
}
```

PageResponse.java (обёртка для пагинации):

```
package ru.edu.project.control.dto.response;

import lombok.Builder;
import lombok.Data;
import java.util.List;

@Data
@Builder
public class PageResponse<T> {
    private List<T> content;
    private int pageNumber;
    private int pageSize;
    private long totalElements;
    private int totalPages;
    private boolean first;
    private boolean last;
}
```

3.3. REST Controller

EventController.java (полная версия с CRUD):

```
package ru.edu.project.control.controllers;
```

```
import io swagger v3 oas annotations Operation;
import io swagger v3 oas annotations Parameter;
import io swagger v3 oas annotations responses ApiResponse;
import io swagger v3 oas annotations responses ApiResponses;
import io swagger v3 oas annotations tags Tag;
import jakarta validation Valid;
import lombok RequiredArgsConstructor;
import lombok extern slf4j Slf4j;
import org springframework data domain Page;
import org springframework data domain PageRequest;
import org springframework data domain Pageable;
import org springframework data domain Sort;
import org springframework format annotation DateTimeFormat;
import org springframework http HttpStatus;
import org springframework http ResponseEntity;
import org springframework security access prepost PreAuthorize;
import org springframework web bind annotation *;
import ru edu project control dto request CreateEventRequest;
import ru edu project control dto request UpdateEventRequest;
import ru edu project control dto response EventResponse;
import ru edu project control dto response PageResponse;
import ru edu project mediator dto EventDto;
import ru edu project mediator interfaces IEventService;

import java time LocalDateTime;
import java util List;

@Slf4j
```

```
@RestController
@RequestMapping("/api/events")
@RequiredArgsConstructor
@Tag(name = "Events", description = "API для управления мероприятиями")
public class EventController {

    private final IEventService eventService;

    // ==================== CREATE ====================

    @PostMapping
    @PreAuthorize("hasRole('ADMIN')")
    @Operation(summary = "Создать новое мероприятие")
    @ApiResponses(value = {
        @ApiResponse(responseCode = "201", description = "Мероприятие создано"),
        @ApiResponse(responseCode = "400", description = "Неверные входные данные"),
        @ApiResponse(responseCode = "401", description = "Не авторизован"),
        @ApiResponse(responseCode = "403", description = "Доступ запрещён (не ADMIN)")
    })
    public ResponseEntity<EventResponse> createEvent(@Valid @RequestBody CreateEventRequest request) {
        log.info("Creating new event: {}", request.getTitle());
```

```
        // TODO: получить текущего пользователя из контекста безопасности
        Long currentUserId = 1L;

        EventDto created = eventService.createEvent(
                request.getTitle(),
                request.getDescription(),
                request.getStartDate(),
                request.getEndDate(),
                request.getMaxParticipants(),
                request.getBasePrice(),
                request.getCategoryId(),
                currentUserId
        );

        return ResponseEntity.status(HttpStatus.CREATED)
                .body(EventResponse.fromDto(created));
    }

    // ==================== READ (all) ====================

    @GetMapping
    @PreAuthorize("hasRole('USER') or hasRole('ADMIN')")
    @Operation(summary = "Получить список всех мероприятий (с пагинацией)")
    public ResponseEntity<PageResponse<EventResponse>> getAllEvents(
            @RequestParam(defaultValue = "0") int page,
            @RequestParam(defaultValue = "10") int size,
            @RequestParam(defaultValue = "startDate,asc") String sort,
```

```
            @RequestParam(required = false) String status,
            @RequestParam(required = false) Long categoryId,
            @RequestParam(required = false) @DateTimeFormat(iso = DateTimeFormat.ISO.DATE_TIME) LocalDateTime startDateFrom,
            @RequestParam(required = false) @DateTimeFormat(iso = DateTimeFormat.ISO.DATE_TIME) LocalDateTime startDateTo,
            @RequestParam(required = false) String search) {

        log.debug("GET /api/events - page={}, size={}, status={}, categoryId={}", page, size, status, categoryId);

        Pageable pageable = PageRequest.of(page, size, Sort.by(parseSort(sort)));
        Page<EventDto> eventPage = eventService.findEvents(pageable, status, categoryId, startDateFrom, startDateTo, search);

        PageResponse<EventResponse> response = PageResponse.<EventResponse>builder()
                .content(eventPage.getContent().stream()
                        .map(EventResponse::fromDto)
                        .toList())
                .pageNumber(eventPage.getNumber())
                .pageSize(eventPage.getSize())
                .totalElements(eventPage.getTotalElements())
                .totalPages(eventPage.getTotalPages())
                .first(eventPage.isFirst())
                .last(eventPage.isLast())
                .build();
```

```
        return ResponseEntity.ok(response);
    }

    @GetMapping("/all")
    @PreAuthorize("hasRole('ADMIN')")
    @Operation(summary = "Получить все мероприятия без пагинации (только для администратора)")
    public ResponseEntity<List<EventResponse>> getAllEventsNoPagination() {
        log.debug("GET /api/events/all");
        List<EventResponse> events = eventService.getAllEvents().stream()
                .map(EventResponse::fromDto)
                .toList();
        return ResponseEntity.ok(events);
    }

    // ==================== READ (one) ====================

    @GetMapping("/{id}")
    @PreAuthorize("hasRole('USER') or hasRole('ADMIN')")
    @Operation(summary = "Получить мероприятие по ID")
    @ApiResponses(value = {
        @ApiResponse(responseCode = "200", description = "Мероприятие найдено"),
        @ApiResponse(responseCode = "404", description = "Мероприятие не найдено")
    })
    public ResponseEntity<EventResponse> getEventById(@PathVariable Long id) {
```

```
        log.debug("GET /api/events/{}", id);
        EventDto event = eventService.getEventById(id);
        return ResponseEntity.ok(EventResponse.fromDto(event));
    }

    // ==================== UPDATE (full) ====================

    @PutMapping("/{id}")
    @PreAuthorize("hasRole('ADMIN')")
    @Operation(summary = "Полное обновление мероприятия")
    public ResponseEntity<EventResponse> updateEvent(
            @PathVariable Long id,
            @Valid @RequestBody UpdateEventRequest request) {

        log.info("PUT /api/events/{} - updating event", id);

        EventDto updated = eventService.updateEvent(
                id,
                request.getTitle(),
                request.getDescription(),
                request.getStartDate(),
                request.getEndDate(),
                request.getMaxParticipants(),
                request.getBasePrice(),
                request.getCategoryId()
        );

        return ResponseEntity.ok(EventResponse.fromDto(updated));
```

```
    }

    // ==================== UPDATE (partial) ====================

    @PatchMapping("/{id}")
    @PreAuthorize("hasRole('ADMIN')")
    @Operation(summary = "Частичное обновление мероприятия")
    public ResponseEntity<EventResponse> patchEvent(
            @PathVariable Long id,
            @RequestBody UpdateEventRequest request) {

        log.info("PATCH /api/events/{} - partially updating event", id);

        // Только не-null поля будут обновлены
        EventDto updated = eventService.patchEvent(
                id,
                request.getTitle(),
                request.getDescription(),
                request.getStartDate(),
                request.getEndDate(),
                request.getMaxParticipants(),
                request.getBasePrice(),
                request.getCategoryId()
        );

        return ResponseEntity.ok(EventResponse.fromDto(updated));
    }
```

```
    // ==================== DELETE ====================

    @DeleteMapping("/{id}")
    @PreAuthorize("hasRole('ADMIN')")
    @Operation(summary = "Удалить мероприятие")
    @ApiResponses(value = {
        @ApiResponse(responseCode = "204", description = "Мероприятие удалено"),
        @ApiResponse(responseCode = "404", description = "Мероприятие не найдено")
    })
    public ResponseEntity<Void> deleteEvent(@PathVariable Long id) {
        log.info("DELETE /api/events/{}", id);
        eventService.deleteEvent(id);
        return ResponseEntity.noContent().build();
    }

    // ==================== HELPER METHODS ====================

    private Sort parseSort(String sortParam) {
        String[] parts = sortParam.split(",");
        String field = parts[0];
        Sort.Direction direction = parts.length > 1 && parts[1].equalsIgnoreCase("desc")
                ? Sort.Direction.DESC
                : Sort.Direction.ASC;
        return Sort.by(direction, field);
    }
```

}

3.4. Расширение Mediator-слоя (Service)

IEventService.java (дополнение):

```
package ru.edu.project.mediator.interfaces;

import org.springframework.data.domain.Page;
import org.springframework.data.domain.Pageable;
import ru.edu.project.mediator.dto.EventDto;
import java.math.BigDecimal;
import java.time.LocalDateTime;

public interface IEventService {

    // Существующие методы...
    EventDto getEventById(Long id);
    List<EventDto> getAllEvents();

    // Новые методы для CRUD
    EventDto createEvent(String title, String description, LocalDateTime
startDate,
                         LocalDateTime endDate, Integer maxParticipants,
BigDecimal basePrice,
                         Long categoryId, Long createdBy);

    EventDto updateEvent(Long id, String title, String description,
                         LocalDateTime startDate, LocalDateTime endDate,
                         Integer maxParticipants, BigDecimal basePrice, Long
categoryId);
```

```
    EventDto patchEvent(Long id, String title, String description,
                        LocalDateTime startDate, LocalDateTime endDate,
                        Integer maxParticipants, BigDecimal basePrice, Long
categoryId);

    void deleteEvent(Long id);

    // Методы для пагинации и фильтрации
    Page<EventDto> findEvents(Pageable pageable, String status, Long
categoryId,
                              LocalDateTime startDateFrom, LocalDateTime
startDateTo, String search);
}
```

3.5. Интеграционные тесты для всех CRUD-операций

EventControllerCrudTest.java:

```
package ru.edu.project.control.controllers;

import com.fasterxml.jackson.databind.ObjectMapper;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.DisplayName;
import org.junit.jupiter.api.Test;
import org.springframework.beans.factory.annotation.Autowired;
import
org.springframework.boot.test.autoconfigure.web.servlet.AutoConfigureMockMvc
;
import org.springframework.http.MediaType;
import org.springframework.test.web.servlet.MockMvc;
import ru.edu.project.control.dto.request.CreateEventRequest;
import ru.edu.project.control.dto.request.UpdateEventRequest;
```

```
import ru.edu.project.entity.Event;
import ru.edu.project.entity.User;
import ru.edu.project.foundation.repositories.EventRepository;
import ru.edu.project.foundation.repositories.UserRepository;
import ru.edu.project.test.AbstractIntegrationTest;

import java.math.BigDecimal;
import java.time.LocalDateTime;

import static org.hamcrest.Matchers.*;
import static org.springframework.test.web.servlet.request.MockMvcRequestBuilders.*;
import static org.springframework.test.web.servlet.result.MockMvcResultMatchers.*;

@AutoConfigureMockMvc
@DisplayName("CRUD-тесты EventController")
class EventControllerCrudTest extends AbstractIntegrationTest {

    @Autowired
    private MockMvc mockMvc;

    @Autowired
    private ObjectMapper objectMapper;

    @Autowired
    private EventRepository eventRepository;

    @Autowired
```

```
    private UserRepository userRepository;

    private String adminToken;
    private Long testEventId;

    @BeforeEach
    void setUp() throws Exception {
        eventRepository.deleteAll();
        userRepository.deleteAll();

        // Регистрация администратора и получение токена
        adminToken = registerAndGetToken("admin@example.com",
"admin123", "Admin User", "ADMIN");

        // Создание тестового мероприятия для операций чтения и
обновления
        testEventId = createTestEvent();
    }

    private String registerAndGetToken(String email, String password, String
name, String role) throws Exception {
        // Регистрация пользователя
        String registerRequest = String.format("""
            {
                "email": "%s",
                "password": "%s",
                "name": "%s",
                "role": "%s"
            }
```

```
            """, email, password, name, role);

        String response = mockMvc.perform(post("/api/auth/register")
                .contentType(MediaType.APPLICATION_JSON)
                .content(registerRequest))
                .andExpect(status().isCreated())
                .andReturn()
                .getResponse()
                .getContentAsString();

        // Извлечение токена
        return objectMapper.readTree(response).get("accessToken").asText();
    }

    private Long createTestEvent() throws Exception {
        CreateEventRequest request = new CreateEventRequest();
        request.setTitle("Test Event");
        request.setDescription("Test Description");
        request.setStartDate(LocalDateTime.now().plusDays(7));
        request.setEndDate(LocalDateTime.now().plusDays(8));
        request.setMaxParticipants(100);
        request.setBasePrice(new BigDecimal("5000"));

        String response = mockMvc.perform(post("/api/events")
                .header("Authorization", "Bearer " + adminToken)
                .contentType(MediaType.APPLICATION_JSON)
                .content(objectMapper.writeValueAsString(request)))
                .andExpect(status().isCreated())
                .andReturn()
```

```
                .getResponse()
                .getContentAsString();

        return objectMapper.readTree(response).get("id").asLong();
    }

    // ==================== CREATE TESTS ====================

    @Test
    @DisplayName("POST /api/events - создание мероприятия должно вернуть 201")
    void createEventShouldReturn201() throws Exception {
        CreateEventRequest request = new CreateEventRequest();
        request.setTitle("New Event");
        request.setDescription("New Description");
        request.setStartDate(LocalDateTime.now().plusDays(14));
        request.setEndDate(LocalDateTime.now().plusDays(15));
        request.setMaxParticipants(200);
        request.setBasePrice(new BigDecimal("10000"));

        mockMvc.perform(post("/api/events")
                .header("Authorization", "Bearer " + adminToken)
                .contentType(MediaType.APPLICATION_JSON)
                .content(objectMapper.writeValueAsString(request)))
                .andExpect(status().isCreated())
                .andExpect(jsonPath("$.id", notNullValue()))
                .andExpect(jsonPath("$.title").value("New Event"))
                .andExpect(jsonPath("$.maxParticipants").value(200));
```

```
    }

    @Test
    @DisplayName("POST /api/events - без токена должно вернуть 401")
    void createEventWithoutTokenShouldReturn401() throws Exception {
        CreateEventRequest request = new CreateEventRequest();
        request.setTitle("New Event");
        request.setStartDate(LocalDateTime.now().plusDays(7));
        request.setEndDate(LocalDateTime.now().plusDays(8));
        request.setMaxParticipants(100);

        mockMvc.perform(post("/api/events")
                .contentType(MediaType.APPLICATION_JSON)
                .content(objectMapper.writeValueAsString(request)))
                .andExpect(status().isUnauthorized());
    }

    @Test
    @DisplayName("POST /api/events - с невалидными данными должно вернуть 400")
    void createEventWithInvalidDataShouldReturn400() throws Exception {
        CreateEventRequest request = new CreateEventRequest();
        request.setTitle(""); // пустое название

        mockMvc.perform(post("/api/events")
                .header("Authorization", "Bearer " + adminToken)
                .contentType(MediaType.APPLICATION_JSON)
                .content(objectMapper.writeValueAsString(request)))
                .andExpect(status().isBadRequest());
```

```
    }

    // ==================== READ TESTS ====================

    @Test
    @DisplayName("GET /api/events - получение списка должно вернуть 200")
    void getAllEventsShouldReturn200() throws Exception {
        mockMvc.perform(get("/api/events")
                .header("Authorization", "Bearer " + adminToken))
                .andExpect(status().isOk())
                .andExpect(jsonPath("$.content", hasSize(greaterThanOrEqualTo(1))))
                .andExpect(jsonPath("$.totalElements").value(1));
    }

    @Test
    @DisplayName("GET /api/events/{id} - получение по ID должно вернуть 200")
    void getEventByIdShouldReturn200() throws Exception {
        mockMvc.perform(get("/api/events/{id}", testEventId)
                .header("Authorization", "Bearer " + adminToken))
                .andExpect(status().isOk())
                .andExpect(jsonPath("$.id").value(testEventId))
                .andExpect(jsonPath("$.title").value("Test Event"));
    }

    @Test
```

```
    @DisplayName("GET /api/events/{id} - несуществующий ID должно вернуть 404")
    void getEventByIdNotFoundShouldReturn404() throws Exception {
        mockMvc.perform(get("/api/events/{id}", 9999L)
                .header("Authorization", "Bearer " + adminToken))
                .andExpect(status().isNotFound());
    }

    @Test
    @DisplayName("GET /api/events - пагинация должна работать")
    void paginationShouldWork() throws Exception {
        // Создание нескольких мероприятий
        for (int i = 0; i < 5; i++) {
            CreateEventRequest request = new CreateEventRequest();
            request.setTitle("Event " + i);
            request.setStartDate(LocalDateTime.now().plusDays(7 + i));
            request.setEndDate(LocalDateTime.now().plusDays(8 + i));
            request.setMaxParticipants(100);

            mockMvc.perform(post("/api/events")
                    .header("Authorization", "Bearer " + adminToken)
                    .contentType(MediaType.APPLICATION_JSON)
                    .content(objectMapper.writeValueAsString(request)));
        }

        // Проверка пагинации
        mockMvc.perform(get("/api/events")
                .header("Authorization", "Bearer " + adminToken)
                .param("page", "0")
```

```
                .param("size", "3"))
                .andExpect(status().isOk())
                .andExpect(jsonPath("$.content", hasSize(3)))
                .andExpect(jsonPath("$.pageNumber").value(0))
                .andExpect(jsonPath("$.pageSize").value(3));

        mockMvc.perform(get("/api/events")
                .header("Authorization", "Bearer " + adminToken)
                .param("page", "1")
                .param("size", "3"))
                .andExpect(status().isOk())
                .andExpect(jsonPath("$.pageNumber").value(1));
    }

    @Test
    @DisplayName("GET /api/events - фильтрация по статусу должна работать")
    void filterByStatusShouldWork() throws Exception {
        mockMvc.perform(get("/api/events")
                .header("Authorization", "Bearer " + adminToken)
                .param("status", "DRAFT"))
                .andExpect(status().isOk())
                .andExpect(jsonPath("$.content",
hasSize(greaterThanOrEqualTo(1))));
    }

    // ==================== UPDATE TESTS ====================
```

```
    @Test
    @DisplayName("PUT /api/events/{id} - полное обновление должно вернуть 200")
    void fullUpdateEventShouldReturn200() throws Exception {
        UpdateEventRequest request = new UpdateEventRequest();
        request.setTitle("Updated Title");
        request.setDescription("Updated Description");
        request.setStartDate(LocalDateTime.now().plusDays(7));
        request.setEndDate(LocalDateTime.now().plusDays(8));
        request.setMaxParticipants(150);
        request.setBasePrice(new BigDecimal("7500"));

        mockMvc.perform(put("/api/events/{id}", testEventId)
                .header("Authorization", "Bearer " + adminToken)
                .contentType(MediaType.APPLICATION_JSON)
                .content(objectMapper.writeValueAsString(request)))
                .andExpect(status().isOk())
                .andExpect(jsonPath("$.title").value("Updated Title"))
                .andExpect(jsonPath("$.maxParticipants").value(150))
                .andExpect(jsonPath("$.basePrice").value(7500));
    }

    @Test
    @DisplayName("PATCH /api/events/{id} - частичное обновление должно вернуть 200")
    void partialUpdateEventShouldReturn200() throws Exception {
        UpdateEventRequest request = new UpdateEventRequest();
        request.setTitle("Patched Title");
        // Другие поля не указаны — они не должны измениться
```

```
        mockMvc.perform(patch("/api/events/{id}", testEventId)
                .header("Authorization", "Bearer " + adminToken)
                .contentType(MediaType.APPLICATION_JSON)
                .content(objectMapper.writeValueAsString(request)))
                .andExpect(status().isOk())
                .andExpect(jsonPath("$.title").value("Patched Title"))
                .andExpect(jsonPath("$.maxParticipants").value(100)); // осталось неизменным
    }

    // ==================== DELETE TESTS ====================

    @Test
    @DisplayName("DELETE /api/events/{id} - удаление должно вернуть 204")
    void deleteEventShouldReturn204() throws Exception {
        mockMvc.perform(delete("/api/events/{id}", testEventId)
                .header("Authorization", "Bearer " + adminToken))
                .andExpect(status().isNoContent());

        // Проверка, что мероприятие действительно удалено
        mockMvc.perform(get("/api/events/{id}", testEventId)
                .header("Authorization", "Bearer " + adminToken))
                .andExpect(status().isNotFound());
    }
}
```

4. МЕТОДИКА ВЫПОЛНЕНИЯ

Шаг 1: Создание Request/Response DTO

Создать CreateXxxRequest для операции создания

Создать UpdateXxxRequest для операции обновления

Создать XxxResponse для ответов

Создать PageResponse<T> для пагинации

Добавить аннотации валидации

Шаг 2: Реализация REST Controller

Создать контроллер для основной сущности

Реализовать POST эндпоинт (создание) → 201 Created

Реализовать GET эндпоинт (список) → 200 OK

Реализовать GET эндпоинт (по ID) → 200 OK / 404

Реализовать PUT эндпоинт (полное обновление) → 200 OK

Реализовать PATCH эндпоинт (частичное обновление) → 200 OK

Реализовать DELETE эндпоинт (удаление) → 204 No Content

Шаг 3: Добавление пагинации и фильтрации

Добавить параметры page, size, sort

Добавить параметры фильтрации (status, categoryId, date range)

Реализовать метод findEvents в сервисе с поддержкой пагинации

Шаг 4: Документирование API

Добавить аннотации Swagger/OpenAPI

Описать эндпоинты, параметры, ответы

Шаг 5: Интеграционные тесты

Написать тесты для CREATE (успех + ошибки)

Написать тесты для READ (все + по ID + пагинация)

Написать тесты для UPDATE (PUT + PATCH)

Написать тесты для DELETE

5. КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Какие HTTP-методы используются для CRUD операций?
2. Какой HTTP-статус возвращается при успешном создании ресурса?
3. Какой HTTP-статус возвращается при успешном удалении?
4. В чем разница между PUT и PATCH?
5. Как реализовать пагинацию в Spring Boot?
6. Как добавить фильтрацию списка по параметрам?
7. Как реализовать частичное обновление (PATCH) для сущности?

6. ТРЕБОВАНИЯ К ОТЧЕТУ

Структура отчета:

Титульный лист

Содержание

Введение

Request/Response DTO (листинг)

REST Controller (листинг всех эндпоинтов)

Пагинация и фильтрация

Интеграционные тесты (листинг)

Заключение

7. КРИТЕРИИ ОЦЕНИВАНИЯ

Критерий	**Макс. балл**	**Описание**
Request/Response	4	Валидация,

Критерий	Макс. балл	Описание
DTO		корректные поля
POST эндпоинт	4	Создание, статус 201
GET эндпоинты	4	Список + по ID, статус 200
PUT эндпоинт	3	Полное обновление
PATCH эндпоинт	3	Частичное обновление
DELETE эндпоинт	3	Удаление, статус 204
Пагинация и фильтрация	3	Правильная работа
Интеграционные тесты	2	Покрытие всех операций
ИТОГО	26	

Лабораторная работа №20

Создание React-приложения (инициализация, роутинг)

1. Цель и задачи работы

Цель: Освоить создание современного React-приложения с использованием Vite, настройку маршрутизации (React Router) и организацию структуры проекта для последующей интеграции с REST API.

Задачи:

1. Создать React-проект с использованием Vite
2. Настроить структуру проекта (компоненты, страницы, API, стили)
3. Установить и настроить React Router DOM для маршрутизации
4. Создать основные страницы приложения (Главная, Список, Детали, Вход, Регистрация, Профиль)
5. Настроить навигацию между страницами
6. Создать общие компоненты (Navbar, Footer, Layout, Loader, ErrorBoundary)

2. ТЕОРЕТИЧЕСКОЕ ОБОСНОВАНИЕ

2.1. Почему Vite?

Vite - это современный инструмент сборки для фронтенд-проектов, который приходит на смену Create React App (CRA).

Характеристика	**Create React App (CRA)**	**Vite**
Скорость запуска dev-сервера	Медленная (бандлинг)	Мгновенная (ES modules)
Горячая замена модулей (HMR)	Медленная при больших проектах	Быстрая
Сборка для production	Webpack	Rollup (быстрее)

Характеристика	Create React App (CRA)	Vite
Размер бандла	Больше	Меньше
Актуальность	Устаревает	Современный стандарт

2.2. Структура React-приложения

```
src/
├── api/                    # API-клиенты (будет в ЛР22)
│   ├── axios.js            # Настройка Axios
│   ├── eventApi.js
│   └── authApi.js
├── assets/                 # Статические файлы (изображения, шрифты)
│   └── images/
├── components/             # Переиспользуемые компоненты
│   ├── common/             # Общие компоненты
│   │   ├── Navbar.jsx
│   │   ├── Footer.jsx
│   │   ├── Loader.jsx
│   │   ├── ErrorAlert.jsx
│   │   └── PrivateRoute.jsx
│   ├── events/             # Компоненты для мероприятий
│   │   ├── EventCard.jsx
│   │   ├── EventList.jsx
│   │   └── EventForm.jsx
│   └── auth/               # Компоненты для авторизации
│       ├── LoginForm.jsx
```

```
│       └── RegisterForm.jsx
├── pages/                  # Компоненты-страницы (для роутинга)
│   ├── HomePage.jsx
│   ├── EventsPage.jsx
│   ├── EventDetailsPage.jsx
│   ├── CreateEventPage.jsx
│   ├── EditEventPage.jsx
│   ├── LoginPage.jsx
│   ├── RegisterPage.jsx
│   └── ProfilePage.jsx
├── hooks/                  # Кастомные хуки
│   ├── useAuth.js
│   └── useEvents.js
├── store/                  # Управление состоянием (будет позже)
│   └── store.js
├── styles/                 # Глобальные стили
│   ├── global.css
│   └── variables.css
├── utils/                  # Вспомогательные функции
│   ├── dateFormatter.js
│   └── validation.js
├── App.jsx                 # Корневой компонент
├── App.css
├── main.jsx                # Точка входа
└── index.css
```

2.3. Маршрутизация в React Router DOM v6

Основные компоненты:

Компонент	Назначение	Пример
BrowserRouter	Обёртка для всего приложения	<BrowserRouter><App /></BrowserRouter>
Routes	Контейнер для маршрутов	<Routes>...</Routes>
Route	Отдельный маршрут	<Route path="/" element={<HomePage />} />
Link	Навигационная ссылка	<Link to="/events">События</Link>
NavLink	Ссылка с активным состоянием	<NavLink to="/events">События</NavLink>
useNavigate	Программная навигация	const navigate = useNavigate(); navigate('/login');
useParams	Получение параметров из URL	const { id } = useParams();
Outlet	Рендер вложенных маршрутов	<Outlet />

2.4. Вложенная маршрутизация

```
<Route path="/events" element={<EventsLayout />}>
   <Route index element={<EventsList />} />
   <Route path=":id" element={<EventDetails />} />
   <Route path="create" element={<CreateEvent />} />
   <Route path=":id/edit" element={<EditEvent />} />
</Route>
```

2.5. Защищённые маршруты (Private Route)

```
const PrivateRoute = ({ children }) => {
   const token = localStorage.getItem('token');
   return token ? children : <Navigate to="/login" />;
};
```

3. ПРИМЕР ВЫПОЛНЕНИЯ (СКВОЗНОЙ ПРОЕКТ)

3.1. Шаг 1: Инициализация проекта

```
# Создание проекта с Vite
npm create vite@latest event-management-frontend -- --template react

# Переход в папку проекта
cd event-management-frontend

# Установка зависимостей
npm install

# Установка дополнительных пакетов
npm install react-router-dom
npm install axios
npm install react-hook-form
```

npm install react-hot-toast

3.2. Шаг 2: Структура проекта (создание папок)

mkdir src/components

mkdir src/components/common

mkdir src/components/events

mkdir src/components/auth

mkdir src/pages

mkdir src/api

mkdir src/hooks

mkdir src/styles

mkdir src/utils

mkdir src/assets

mkdir src/assets/images

3.3. Шаг 3: Глобальные стили

src/styles/variables.css:

```
:root {
    /* Цветовая схема */
    --primary-color: #3498db;
    --primary-dark: #2980b9;
    --secondary-color: #2ecc71;
    --danger-color: #e74c3c;
    --warning-color: #f39c12;
    --dark-color: #2c3e50;
    --light-color: #ecf0f1;
    --text-color: #333;
    --text-light: #666;
```

```
    --white: #fff;
    --black: #000;

    /* Тени */
    --box-shadow: 0 2px 4px rgba(0,0,0,0.1);
    --box-shadow-hover: 0 4px 8px rgba(0,0,0,0.15);

    /* Отступы */
    --spacing-xs: 4px;
    --spacing-sm: 8px;
    --spacing-md: 16px;
    --spacing-lg: 24px;
    --spacing-xl: 32px;

    /* Шрифты */
    --font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;

    /* Брейкпоинты */
    --breakpoint-mobile: 768px;
    --breakpoint-tablet: 1024px;
}
```

src/styles/global.css:

```
css
@import './variables.css';

* {
    margin: 0;
    padding: 0;
    box-sizing: border-box;
```

```
}

body {
    font-family: var(--font-family);
    color: var(--text-color);
    background-color: var(--light-color);
    line-height: 1.6;
}

a {
    text-decoration: none;
    color: inherit;
}

.container {
    max-width: 1200px;
    margin: 0 auto;
    padding: 0 var(--spacing-md);
}

.btn {
    display: inline-block;
    padding: var(--spacing-sm) var(--spacing-lg);
    border: none;
    border-radius: 4px;
    cursor: pointer;
    font-size: 1rem;
    transition: all 0.3s ease;
}
```

```
.btn-primary {
    background-color: var(--primary-color);
    color: var(--white);
}

.btn-primary:hover {
    background-color: var(--primary-dark);
}

.btn-danger {
    background-color: var(--danger-color);
    color: var(--white);
}

.btn-sm {
    padding: var(--spacing-xs) var(--spacing-md);
    font-size: 0.875rem;
}

.form-group {
    margin-bottom: var(--spacing-md);
}

.form-group label {
    display: block;
    margin-bottom: var(--spacing-xs);
    font-weight: bold;
}
```

```
.form-control {
    width: 100%;
    padding: var(--spacing-sm);
    border: 1px solid #ddd;
    border-radius: 4px;
    font-size: 1rem;
}

.form-control:focus {
    outline: none;
    border-color: var(--primary-color);
    box-shadow: 0 0 0 2px rgba(52,152,219,0.2);
}

.alert {
    padding: var(--spacing-md);
    border-radius: 4px;
    margin-bottom: var(--spacing-md);
}

.alert-error {
    background-color: #f8d7da;
    color: #721c24;
    border: 1px solid #f5c6cb;
}

.alert-success {
    background-color: #d4edda;
```

```
    color: #155724;
    border: 1px solid #c3e6cb;
}
```

3.4. Шаг 4: Общие компоненты

src/components/common/Navbar.jsx:

```
jsx
import React from 'react';
import { Link, NavLink, useNavigate } from 'react-router-dom';
import './Navbar.css';

const Navbar = () => {
    const navigate = useNavigate();
    const token = localStorage.getItem('token');
    const user = JSON.parse(localStorage.getItem('user') || '{}');

    const handleLogout = () => {
        localStorage.removeItem('token');
        localStorage.removeItem('user');
        navigate('/login');
    };

    return (
        <nav className="navbar">
            <div className="container navbar-container">
                <Link to="/" className="navbar-brand">
                    🎉 EventManager
                </Link>

                <div className="navbar-menu">
```

```
                    <NavLink to="/" className={({ isActive }) => isActive ? 'active' : ''}>
                      Главная
                    </NavLink>
                    <NavLink to="/events" className={({ isActive }) => isActive ? 'active' : ''}>
                      Мероприятия
                    </NavLink>

                    {token ? (
                      <>
                        <NavLink to="/my-registrations" className={({ isActive }) => isActive ? 'active' : ''}>
                          Мои регистрации
                        </NavLink>
                        {user.role === 'ADMIN' && (
                          <NavLink to="/events/create" className={({ isActive }) => isActive ? 'active' : ''}>
                            + Создать
                          </NavLink>
                        )}
                        <NavLink to="/profile" className={({ isActive }) => isActive ? 'active' : ''}>
                          {user.name || 'Профиль'}
                        </NavLink>
                        <button onClick={handleLogout} className="btn-logout">
                          Выйти
                        </button>
```

```
                        </>
                    ) : (
                        <>
                            <NavLink to="/login" className={({ isActive }) =>
isActive ? 'active' : ''}>
                                Вход
                            </NavLink>
                            <NavLink to="/register" className={({ isActive }) =>
isActive ? 'active' : ''}>
                                Регистрация
                            </NavLink>
                        </>
                    )}
                </div>
            </div>
        </nav>
    );
};

export default Navbar;
```

src/components/common/Navbar.css:

```
.navbar {
    background-color: var(--dark-color);
    color: var(--white);
    padding: 1rem 0;
    position: sticky;
    top: 0;
    z-index: 1000;
```

```
}

.navbar-container {
    display: flex;
    justify-content: space-between;
    align-items: center;
}

.navbar-brand {
    font-size: 1.5rem;
    font-weight: bold;
    color: var(--white);
}

.navbar-menu {
    display: flex;
    align-items: center;
    gap: 1.5rem;
}

.navbar-menu a {
    color: var(--white);
    transition: color 0.3s ease;
}

.navbar-menu a:hover,
.navbar-menu a.active {
    color: var(--primary-color);
}
```

```
.btn-logout {
    background: none;
    border: none;
    color: var(--white);
    cursor: pointer;
    font-size: 1rem;
    padding: 0;
}

.btn-logout:hover {
    color: var(--danger-color);
}

@media (max-width: 768px) {
    .navbar-container {
        flex-direction: column;
        gap: 10px;
    }

    .navbar-menu {
        flex-wrap: wrap;
        justify-content: center;
        gap: 1rem;
    }
}
```

src/components/common/Footer.jsx:

```
import React from 'react';
```

```
import './Footer.css';

const Footer = () => {
  return (
    <footer className="footer">
      <div className="container">
        <p>&copy; 2025 Event Management System. Все права защищены.</p>
      </div>
    </footer>
  );
};

export default Footer;
```

src/components/common/Footer.css:

```
.footer {
  background-color: var(--dark-color);
  color: var(--white);
  text-align: center;
  padding: 1.5rem 0;
  margin-top: auto;
}
```

src/components/common/Loader.jsx:

```
import React from 'react';
import './Loader.css';
```

```
const Loader = () => {
    return (
        <div className="loader-container">
            <div className="loader"></div>
            <p>Загрузка...</p>
        </div>
    );
};

export default Loader;
```

src/components/common/Loader.css:

```
.loader-container {
    display: flex;
    flex-direction: column;
    align-items: center;
    justify-content: center;
    padding: 2rem;
}

.loader {
    width: 48px;
    height: 48px;
    border: 4px solid #f3f3f3;
    border-top: 4px solid var(--primary-color);
    border-radius: 50%;
    animation: spin 1s linear infinite;
}
```

```
@keyframes spin {
  0% { transform: rotate(0deg); }
  100% { transform: rotate(360deg); }
}
```

src/components/common/PrivateRoute.jsx:

```
import React from 'react';
import { Navigate, Outlet } from 'react-router-dom';

const PrivateRoute = () => {
  const token = localStorage.getItem('token');
  return token ? <Outlet /> : <Navigate to="/login" />;
};

export default PrivateRoute;
```

src/components/common/AdminRoute.jsx:

```
import React from 'react';
import { Navigate, Outlet } from 'react-router-dom';

const AdminRoute = () => {
  const token = localStorage.getItem('token');
  const user = JSON.parse(localStorage.getItem('user') || '{}');

  if (!token) {
    return <Navigate to="/login" />;
  }

  if (user.role !== 'ADMIN') {
```

```
        return <Navigate to="/" />;
    }

    return <Outlet />;
};

export default AdminRoute;
```

3.5. Шаг 5: Компоненты-страницы

src/pages/HomePage.jsx:

```
import React from 'react';
import { Link } from 'react-router-dom';
import './HomePage.css';

const HomePage = () => {
    const token = localStorage.getItem('token');

    return (
        <div className="home-page">
            <div className="hero">
                <h1>Добро пожаловать в EventManager</h1>
                <p>Управляйте мероприятиями легко и удобно</p>
                {!token && (
                    <div className="hero-buttons">
                        <Link to="/register" className="btn btn-primary">
                            Начать сейчас
                        </Link>
                        <Link to="/login" className="btn btn-outline">
                            Войти
```

```
            </Link>
          </div>
        )}
        {token && (
          <Link to="/events" className="btn btn-primary">
            Перейти к мероприятиям
          </Link>
        )}
      </div>

      <div className="features">
        <h2>Возможности системы</h2>
        <div className="features-grid">
          <div className="feature-card">
            <div className="feature-icon">📋</div>
            <h3>Создание мероприятий</h3>
            <p>Легко создавайте и управляйте мероприятиями</p>
          </div>
          <div className="feature-card">
            <div className="feature-icon">👥</div>
            <h3>Регистрация участников</h3>
            <p>Участники могут быстро зарегистрироваться</p>
          </div>
          <div className="feature-card">
            <div className="feature-icon">📊</div>
            <h3>Аналитика</h3>
            <p>Отслеживайте посещаемость и отзывы</p>
          </div>
```

```
            </div>
          </div>
        </div>
      );
    };

    export default HomePage;
```

src/pages/HomePage.css:

```
    .home-page {
      min-height: calc(100vh - 140px);
    }

    .hero {
      text-align: center;
      padding: 4rem 2rem;
      background: linear-gradient(135deg, var(--primary-color), var(--primary-dark));
      color: var(--white);
    }

    .hero h1 {
      font-size: 2.5rem;
      margin-bottom: 1rem;
    }

    .hero p {
      font-size: 1.2rem;
      margin-bottom: 2rem;
```

```
}

.hero-buttons {
    display: flex;
    gap: 1rem;
    justify-content: center;
}

.btn-outline {
    background: transparent;
    border: 2px solid var(--white);
    color: var(--white);
}

.btn-outline:hover {
    background: var(--white);
    color: var(--primary-color);
}

.features {
    padding: 3rem 2rem;
    max-width: 1200px;
    margin: 0 auto;
}

.features h2 {
    text-align: center;
    margin-bottom: 2rem;
}
```

```
.features-grid {
    display: grid;
    grid-template-columns: repeat(auto-fit, minmax(300px, 1fr));
    gap: 2rem;
}

.feature-card {
    text-align: center;
    padding: 2rem;
    background: var(--white);
    border-radius: 8px;
    box-shadow: var(--box-shadow);
    transition: transform 0.3s ease;
}

.feature-card:hover {
    transform: translateY(-5px);
    box-shadow: var(--box-shadow-hover);
}

.feature-icon {
    font-size: 3rem;
    margin-bottom: 1rem;
}
```

src/pages/LoginPage.jsx:

```
jsx
import React, { useState } from 'react';
import { useNavigate, Link } from 'react-router-dom';
```

```
import './AuthPages.css';

const LoginPage = () => {
    const navigate = useNavigate();
    const [formData, setFormData] = useState({
        email: '',
        password: ''
    });
    const [error, setError] = useState('');
    const [loading, setLoading] = useState(false);

    const handleChange = (e) => {
        setFormData({
            ...formData,
            [e.target.name]: e.target.value
        });
    };

    const handleSubmit = async (e) => {
        e.preventDefault();
        setError('');
        setLoading(true);

        try {
            // TODO: заменить на реальный API-вызов (ЛР22)
            // const response = await authApi.login(formData);

            // Временная имитация (удалить после подключения API)
            await new Promise(resolve => setTimeout(resolve, 1000));
```

```
            if (formData.email === 'admin@example.com' &&
formData.password === 'admin') {
                localStorage.setItem('token', 'fake-jwt-token');
                localStorage.setItem('user', JSON.stringify({
                  id: 1,
                  email: 'admin@example.com',
                  name: 'Admin',
                  role: 'ADMIN'
                }));
                navigate('/');
              } else {
                setError('Неверный email или пароль');
              }
            } catch (err) {
              setError(err.response?.data?.message || 'Ошибка входа');
            } finally {
              setLoading(false);
            }
          };

          return (
            <div className="auth-page">
              <div className="auth-card">
                <h2>Вход в систему</h2>

                {error && <div className="alert alert-error">{error}</div>}

                <form onSubmit={handleSubmit}>
```

```
                    <div className="form-group">
                        <label htmlFor="email">Email</label>
                        <input
                            type="email"
                            id="email"
                            name="email"
                            className="form-control"
                            value={formData.email}
                            onChange={handleChange}
                            required
                        />
                    </div>

                    <div className="form-group">
                        <label htmlFor="password">Пароль</label>
                        <input
                            type="password"
                            id="password"
                            name="password"
                            className="form-control"
                            value={formData.password}
                            onChange={handleChange}
                            required
                        />
                    </div>

                    <button type="submit" className="btn btn-primary"
disabled={loading}>
                        {loading ? 'Вход...' : 'Войти'}
```

```
                    </button>
                </form>

                <p className="auth-link">
                    Нет аккаунта? <Link to="/register">Зарегистрироваться</Link>
                </p>
            </div>
        </div>
    );
};

export default LoginPage;
```

src/pages/RegisterPage.jsx:

```
import React, { useState } from 'react';
import { useNavigate, Link } from 'react-router-dom';
import './AuthPages.css';

const RegisterPage = () => {
    const navigate = useNavigate();
    const [formData, setFormData] = useState({
        email: '',
        password: '',
        confirmPassword: '',
        name: ''
    });
    const [error, setError] = useState('');
    const [loading, setLoading] = useState(false);
```

```
const handleChange = (e) => {
    setFormData({
        ...formData,
        [e.target.name]: e.target.value
    });
};

const handleSubmit = async (e) => {
    e.preventDefault();
    setError('');

    if (formData.password !== formData.confirmPassword) {
        setError('Пароли не совпадают');
        return;
    }

    setLoading(true);

    try {
        // TODO: заменить на реальный API-вызов (ЛР22)
        await new Promise(resolve => setTimeout(resolve, 1000));

        localStorage.setItem('token', 'fake-jwt-token');
        localStorage.setItem('user', JSON.stringify({
            id: 1,
            email: formData.email,
            name: formData.name,
            role: 'USER'
        }));
```

```
      navigate('/');
    } catch (err) {
      setError(err.response?.data?.message || 'Ошибка регистрации');
    } finally {
      setLoading(false);
    }
  };

  return (
    <div className="auth-page">
      <div className="auth-card">
        <h2>Регистрация</h2>

        {error && <div className="alert alert-error">{error}</div>}

        <form onSubmit={handleSubmit}>
          <div className="form-group">
            <label htmlFor="name">Имя</label>
            <input
              type="text"
              id="name"
              name="name"
              className="form-control"
              value={formData.name}
              onChange={handleChange}
              required
            />
          </div>
```

```
                    <div className="form-group">
                        <label htmlFor="email">Email</label>
                        <input
                            type="email"
                            id="email"
                            name="email"
                            className="form-control"
                            value={formData.email}
                            onChange={handleChange}
                            required
                        />
                    </div>

                    <div className="form-group">
                        <label htmlFor="password">Пароль</label>
                        <input
                            type="password"
                            id="password"
                            name="password"
                            className="form-control"
                            value={formData.password}
                            onChange={handleChange}
                            required
                        />
                    </div>

                    <div className="form-group">
                        <label            htmlFor="confirmPassword">Подтверждение
пароля</label>
```

```
                        <input
                            type="password"
                            id="confirmPassword"
                            name="confirmPassword"
                            className="form-control"
                            value={formData.confirmPassword}
                            onChange={handleChange}
                            required
                        />
                    </div>

                    <button type="submit" className="btn btn-primary" disabled={loading}>
                        {loading ? 'Регистрация...' : 'Зарегистрироваться'}
                    </button>
                </form>

                <p className="auth-link">
                    Уже есть аккаунт? <Link to="/login">Войти</Link>
                </p>
            </div>
        </div>
    );
};

export default RegisterPage;
```

src/pages/AuthPages.css:

```
.auth-page {
```

```
    display: flex;
    justify-content: center;
    align-items: center;
    min-height: calc(100vh - 140px);
    padding: 2rem;
}

.auth-card {
    background: var(--white);
    padding: 2rem;
    border-radius: 8px;
    box-shadow: var(--box-shadow);
    width: 100%;
    max-width: 400px;
}

.auth-card h2 {
    text-align: center;
    margin-bottom: 1.5rem;
}

.auth-link {
    text-align: center;
    margin-top: 1rem;
}

.auth-link a {
    color: var(--primary-color);
}
```

src/pages/EventsPage.jsx (заглушка):

```
import React from 'react';
import { Link } from 'react-router-dom';

const EventsPage = () => {
  return (
    <div className="container">
      <h1>Мероприятия</h1>
      <p>Страница в разработке. Будет реализована в ЛР21.</p>
      <Link to="/events/create" className="btn btn-primary">
        Создать мероприятие
      </Link>
    </div>
  );
};

export default EventsPage;
```

3.6. Шаг 6: Настройка маршрутизации (App.jsx)

src/App.jsx:

```
import React from 'react';
import { BrowserRouter as Router, Routes, Route } from 'react-router-dom';
import Navbar from './components/common/Navbar';
import Footer from './components/common/Footer';
import PrivateRoute from './components/common/PrivateRoute';
import AdminRoute from './components/common/AdminRoute';
import HomePage from './pages/HomePage';
import EventsPage from './pages/EventsPage';
import EventDetailsPage from './pages/EventDetailsPage';
```

```
import CreateEventPage from './pages/CreateEventPage';
import EditEventPage from './pages/EditEventPage';
import LoginPage from './pages/LoginPage';
import RegisterPage from './pages/RegisterPage';
import ProfilePage from './pages/ProfilePage';
import MyRegistrationsPage from './pages/MyRegistrationsPage';
import NotFoundPage from './pages/NotFoundPage';
import './styles/global.css';

function App() {
  return (
    <Router>
      <div className="app">
        <Navbar />
        <main className="main-content">
          <Routes>
            {/* Публичные маршруты */}
            <Route path="/" element={<HomePage />} />
            <Route path="/events" element={<EventsPage />} />
            <Route  path="/events/:id"  element={<EventDetailsPage  />}
/>
            <Route path="/login" element={<LoginPage />} />
            <Route path="/register" element={<RegisterPage />} />

            {/* Защищённые маршруты (требуют авторизации) */}
            <Route element={<PrivateRoute />}>
              <Route path="/profile" element={<ProfilePage />} />
              <Route                              path="/my-registrations"
element={<MyRegistrationsPage />} />
```

```
                    </Route>

                    {/* Административные маршруты (требуют роль ADMIN)
*/}
                    <Route element={<AdminRoute />}>
                      <Route path="/events/create" element={<CreateEventPage
/>} />
                      <Route path="/events/:id/edit" element={<EditEventPage
/>} />
                    </Route>

                    {/* 404 - страница не найдена */}
                    <Route path="*" element={<NotFoundPage />} />
                  </Routes>
                </main>
                <Footer />
              </div>
            </Router>
          );
        }

        export default App;
```

3.7. Шаг 7: Точка входа (main.jsx)

src/main.jsx:

```
import React from 'react';
import ReactDOM from 'react-dom/client';
import App from './App';
import './styles/global.css';
```

```
ReactDOM.createRoot(document.getElementById('root')).render(
  <React.StrictMode>
    <App />
  </React.StrictMode>
);
```

3.8. Шаг 8: Запуск приложения

```
# Запуск в режиме разработки
npm run dev

# Приложение будет доступно по адресу http://localhost:5173
```

4. МЕТОДИКА ВЫПОЛНЕНИЯ

Шаг 1: Создание проекта

Выполнить npm create vite@latest

Выбрать React + JavaScript

Установить зависимости

Шаг 2: Установка дополнительных пакетов

npm install react-router-dom

npm install axios

npm install react-hook-form

Шаг 3: Создание структуры папок

Создать папки: components, pages, api, hooks, styles, utils

Шаг 4: Создание общих компонентов (1 час)

Navbar, Footer, Loader, PrivateRoute, AdminRoute

Шаг 5: Создание страниц

HomePage, LoginPage, RegisterPage

EventsPage (заглушка), NotFoundPage

Шаг 6: Настройка маршрутизации

Настроить Routes в App.jsx

Проверить навигацию

Шаг 7: Стилизация

Создать глобальные стили

Добавить CSS для компонентов

5. КОНТРОЛЬНЫЕ ВОПРОСЫ

1. В чем разница между Vite и Create React App?
2. Какие компоненты используются для маршрутизации в React Router v6?
3. Как передать параметр из URL в компонент?
4. Как реализовать программную навигацию (переход по кнопке)?
5. Что такое вложенная маршрутизация и зачем она нужна?
6. Как защитить маршрут от неавторизованного доступа?
7. Как получить текущий URL и параметры в компоненте?

6. ТРЕБОВАНИЯ К ОТЧЕТУ

Структура отчета:

Титульный лист

Содержание

Введение

Структура проекта

Общие компоненты (листинг)

Компоненты-страницы (листинг)

Маршрутизация (App.jsx)

Скриншоты работы приложения

Заключение

7. КРИТЕРИИ ОЦЕНИВАНИЯ

Критерий	Макс. балл	Описание
Создание проекта	3	Vite, структура папок
Общие компоненты	5	Navbar, Footer, Loader, PrivateRoute
Страницы	6	HomePage, LoginPage, RegisterPage, NotFoundPage
Маршрутизация	5	Routes, вложенные маршруты, защита
Стилизация	4	CSS, адаптивность
Оформление отчета	2	Соответствие требованиям
ИТОГО	**25**	

Лабораторная работа №21

Разработка компонентов (список, карточка, форма)

1. Цель работы

Цель: Разработать основные React-компоненты для отображения списка сущностей (EventList), карточки сущности (EventCard) и формы создания/редактирования (EventForm) с использованием React Hook Form для управления состоянием формы и валидацией.

Задачи:

1. Разработать компонент EventCard для отображения карточки мероприятия
2. Разработать компонент EventList для отображения списка мероприятий с пагинацией
3. Разработать компонент EventForm для создания и редактирования мероприятий
4. Реализовать валидацию формы с помощью React Hook Form и Zod
5. Добавить обработку состояний загрузки (loading) и ошибок (error)
6. Интегрировать компоненты на страницы EventsPage, CreateEventPage, EditEventPage

2. ТЕОРЕТИЧЕСКОЕ ОБОСНОВАНИЕ

2.1. Компонентный подход в React

Компонент - это независимая, переиспользуемая часть интерфейса, которая инкапсулирует логику, структуру и стили.

Принципы разработки компонентов:

Принцип	Описание
Single Responsibility	Один компонент — одна

Принцип	Описание
	ответственность
Reusability	Компонент должен быть переиспользуемым
Composability	Компоненты могут комбинироваться
Props Drilling	Передача данных через пропсы

2.2. React Hook Form

React Hook Form - библиотека для управления формами в React, которая минимизирует количество ререндеров и упрощает валидацию.

Характеристика	Описание
Производительность	Минимальное количество ререндеров
Простота	Интуитивный API
Интеграция	Легко интегрируется с Zod, Yup
Типизация	Отличная поддержка TypeScript

Основные хуки:

Хук	Назначение
useForm	Основной хук для работы с формой
register	Регистрация поля ввода

Хук	Назначение
handleSubmit	Обработка отправки формы
formState.errors	Объект с ошибками валидации
reset	Сброс формы

2.3. Валидация с Zod

Zod - библиотека для декларативной валидации данных с TypeScript-выводом.

Схема валидации для мероприятия:

```
import { z } from 'zod';

const eventSchema = z.object({
  title: z.string()
    .min(3, 'Название должно содержать минимум 3 символа')
    .max(200, 'Название не должно превышать 200 символов'),
  description: z.string().max(5000, 'Описание не должно превышать 5000 символов').optional(),
  startDate: z.string()
    .min(1, 'Дата начала обязательна')
    .refine(date => new Date(date) > new Date(), 'Дата начала должна быть в будущем'),
  endDate: z.string()
    .min(1, 'Дата окончания обязательна'),
  maxParticipants: z.number()
    .min(1, 'Минимум 1 участник')
    .max(10000, 'Максимум 10000 участников'),
```

```
    basePrice: z.number().min(0, 'Цена не может быть отрицательной').optional(),
    categoryId: z.number().optional()
  }).refine(data => new Date(data.endDate) > new Date(data.startDate), {
    message: 'Дата окончания должна быть позже даты начала',
    path: ['endDate']
  });
```

2.4. Пагинация

Компонент пагинации позволяет пользователю перемещаться между страницами списка.

Параметр	Описание
currentPage	Текущая страница (0-index)
totalPages	Общее количество страниц
onPageChange	Callback при смене страницы
pageSize	Количество элементов на странице

3. ПРИМЕР ВЫПОЛНЕНИЯ (СКВОЗНОЙ ПРОЕКТ)

3.1. Установка зависимостей

```
npm install react-hook-form
npm install zod
npm install @hookform/resolvers
```

3.2. Компонент EventCard

src/components/events/EventCard.jsx:

```
import React from 'react';
import { useNavigate } from 'react-router-dom';
import { formatDate, formatDateTime } from '../../utils/dateFormatter';
import './EventCard.css';

const EventCard = ({ event, onDelete, isAdmin = false }) => {
    const navigate = useNavigate();

    const getStatusBadge = (status) => {
        const statusMap = {
            'DRAFT': { class: 'badge-draft', text: 'Черновик' },
            'PUBLISHED': { class: 'badge-published', text: 'Опубликовано' },
            'CANCELLED': { class: 'badge-cancelled', text: 'Отменено' },
            'COMPLETED': { class: 'badge-completed', text: 'Завершено' }
        };
        const statusInfo = statusMap[status] || { class: 'badge-draft', text:
status };
        return <span className={`badge
${statusInfo.class}`}>{statusInfo text}</span>;
    };

    const getAvailabilityClass = (availableSeats, maxParticipants) => {
        const ratio = availableSeats / maxParticipants;
        if (ratio === 0) return 'availability-full';
        if (ratio < 0.2) return 'availability-low';
        if (ratio < 0.5) return 'availability-medium';
        return 'availability-high';
    };
```

```
        return (
            <div className="event-card">
                <div className="event-card-header">
                    <h3>{event title}</h3>
                    {getStatusBadge(event status)}
                </div>

                <p className="event-description">
                    {event description?.substring(0, 120)}
                    {event description?.length > 120 ? '...' : ''}
                </p>

                <div className="event-details">
                    <div className="event-detail">
                        <span className="detail-label">📅 Дата:</span>
                        <span>{formatDate(event startDate)}</span>
                    </div>
                    <div className="event-detail">
                        <span className="detail-label">🕐 Время:</span>
                        <span>{formatDateTime(event startDate)}</span>
                    </div>
                    <div className="event-detail">
                        <span className="detail-label">👥 Участники:</span>
                        <span>{event registeredCount || 0} /
{event maxParticipants}</span>
                        <div className="availability-bar">
                            <div
```

```
                        className={`availability-fill
${getAvailabilityClass(event.availableSeats, event.maxParticipants)}`}
                        style={{ width: `${(event.registeredCount /
event.maxParticipants) * 100}%` }}
                      />
                    </div>
                  </div>
                  {event.basePrice > 0 && (
                    <div className="event-detail">
                      <span className="detail-label">💰 Цена:</span>
                      <span>{event.basePrice.toLocaleString()} ₽</span>
                    </div>
                  )}
                </div>

                <div className="event-card-actions">
                  <button
                    className="btn btn-sm btn-info"
                    onClick={() => navigate(`/events/${event.id}`)}
                  >
                    Подробнее
                  </button>

                  {isAdmin && (
                    <>
                      <button
                        className="btn btn-sm btn-warning"
                        onClick={() => navigate(`/events/${event.id}/edit`)}
                      >
```

```
                        Редактировать
                    </button>
                    <button
                        className="btn btn-sm btn-danger"
                        onClick={() => onDelete(event id)}
                    >
                        Удалить
                    </button>
                </>
            )}

            {!isAdmin && event status === 'PUBLISHED' && event availableSeats > 0 && (
                <button
                    className="btn btn-sm btn-primary"
                    onClick={() => navigate(`/events/${event id}/register`)}
                >
                    Зарегистрироваться
                </button>
            )}
        </div>
    </div>
  );
};

export default EventCard;
```

src/components/events/EventCard.css:

```
.event-card {
  background: var(--white);
```

```
    border-radius: 8px;
    box-shadow: var(--box-shadow);
    padding: 1.5rem;
    transition: transform 0.3s ease, box-shadow 0.3s ease;
}

.event-card:hover {
    transform: translateY(-4px);
    box-shadow: var(--box-shadow-hover);
}

.event-card-header {
    display: flex;
    justify-content: space-between;
    align-items: center;
    margin-bottom: 1rem;
    flex-wrap: wrap;
    gap: 0.5rem;
}

.event-card-header h3 {
    color: var(--dark-color);
    font-size: 1.25rem;
}

.badge {
    padding: 0.25rem 0.75rem;
    border-radius: 20px;
    font-size: 0.75rem;
```

```
    font-weight: bold;
}

.badge-draft {
    background-color: #95a5a6;
    color: white;
}

.badge-published {
    background-color: var(--primary-color);
    color: white;
}

.badge-cancelled {
    background-color: var(--danger-color);
    color: white;
}

.badge-completed {
    background-color: var(--secondary-color);
    color: white;
}

.event-description {
    color: var(--text-light);
    margin-bottom: 1rem;
    line-height: 1.5;
}
```

```css
.event-details {
    margin-bottom: 1rem;
}

.event-detail {
    display: flex;
    justify-content: space-between;
    margin-bottom: 0.5rem;
    font-size: 0.9rem;
}

.detail-label {
    color: var(--text-light);
}

.availability-bar {
    width: 100%;
    height: 4px;
    background-color: #e0e0e0;
    border-radius: 2px;
    margin-top: 0.25rem;
    overflow: hidden;
}

.availability-fill {
    height: 100%;
    transition: width 0.3s ease;
}
```

```
.availability-fill.availability-high {
    background-color: var(--secondary-color);
}

.availability-fill.availability-medium {
    background-color: var(--warning-color);
}

.availability-fill.availability-low {
    background-color: #e67e22;
}

.availability-fill.availability-full {
    background-color: var(--danger-color);
}

.event-card-actions {
    display: flex;
    gap: 0.5rem;
    flex-wrap: wrap;
    margin-top: 1rem;
}
```

3.3. Компонент EventForm

src/components/events/EventForm.jsx:

```
import React, { useEffect, useState } from 'react';
import { useForm } from 'react-hook-form';
import { zodResolver } from '@hookform/resolvers/zod';
import { z } from 'zod';
```

```
import './EventForm.css';

// Схема валидации
const eventSchema = z.object({
  title: z.string()
    .min(3, 'Название должно содержать минимум 3 символа')
    .max(200, 'Название не должно превышать 200 символов'),
  description: z.string().max(5000, 'Описание не должно превышать 5000 символов').optional(),
  startDate: z.string()
    .min(1, 'Дата начала обязательна')
    .refine(date => new Date(date) > new Date(), 'Дата начала должна быть в будущем'),
  endDate: z.string()
    .min(1, 'Дата окончания обязательна'),
  maxParticipants: z.number()
    .min(1, 'Минимум 1 участник')
    .max(10000, 'Максимум 10000 участников'),
  basePrice: z.number().min(0, 'Цена не может быть отрицательной').optional(),
  categoryId: z.number().optional()
}).refine(data => new Date(data.endDate) > new Date(data.startDate), {
  message: 'Дата окончания должна быть позже даты начала',
  path: ['endDate']
});

const EventForm = ({ initialData, onSubmit, isLoading, categories = [] }) => {
  const {
```

```
    register,
    handleSubmit,
    setValue,
    formState: { errors },
    watch
} = useForm({
    resolver: zodResolver(eventSchema),
    defaultValues: {
        title: '',
        description: '',
        startDate: '',
        endDate: '',
        maxParticipants: 1,
        basePrice: 0,
        categoryId: ''
    }
});

const startDate = watch('startDate');

// Заполнение формы при редактировании
useEffect(() => {
    if (initialData) {
        setValue('title', initialData.title || '');
        setValue('description', initialData.description || '');
        setValue('startDate', initialData.startDate?.slice(0, 16) || '');
        setValue('endDate', initialData.endDate?.slice(0, 16) || '');
        setValue('maxParticipants', initialData.maxParticipants || 1);
        setValue('basePrice', initialData.basePrice || 0);
```

```
            setValue('categoryId', initialData.categoryId || '');
        }
    }, [initialData, setValue]);

    // Форматирование данных перед отправкой
    const onFormSubmit = (data) => {
        const formattedData = {
            ...data,
            startDate: new Date(data.startDate).toISOString(),
            endDate: new Date(data.endDate).toISOString(),
            maxParticipants: Number(data.maxParticipants),
            basePrice: Number(data.basePrice),
            categoryId: data.categoryId ? Number(data.categoryId) : null
        };
        onSubmit(formattedData);
    };

    return (
        <form onSubmit={handleSubmit(onFormSubmit)} className="event-form">
            <div className="form-group">
                <label htmlFor="title">Название мероприятия *</label>
                <input
                    id="title"
                    type="text"
                    className={`form-control ${errors.title ? 'is-invalid' : ''}`}
                    placeholder="Введите название мероприятия"
                    {...register('title')}
                />
```

```
                {errors.title && (
                  <div                                    className="invalid-
feedback">{errors.title.message}</div>
                )}
              </div>

              <div className="form-group">
                <label htmlFor="description">Описание</label>
                <textarea
                  id="description"
                  className={`form-control ${errors.description ? 'is-invalid' :
''}`}
                  rows="5"
                  placeholder="Введите описание мероприятия"
                  {...register('description')}
                />
                {errors.description && (
                  <div                                    className="invalid-
feedback">{errors.description.message}</div>
                )}
              </div>

              <div className="form-row">
                <div className="form-group">
                  <label htmlFor="startDate">Дата и время начала *</label>
                  <input
                    id="startDate"
                    type="datetime-local"
```

```
                    className={`form-control ${errors.startDate ? 'is-invalid' : ''}`}
                    {...register('startDate')}
                  />
                  {errors.startDate && (
                    <div className="invalid-feedback">{errors.startDate.message}</div>
                  )}
                </div>

                <div className="form-group">
                  <label htmlFor="endDate">Дата и время окончания *</label>
                  <input
                    id="endDate"
                    type="datetime-local"
                    className={`form-control ${errors.endDate ? 'is-invalid' : ''}`}
                    {...register('endDate')}
                  />
                  {errors.endDate && (
                    <div className="invalid-feedback">{errors.endDate.message}</div>
                  )}
                </div>
              </div>

              <div className="form-row">
                <div className="form-group">
```

```
                    <label    htmlFor="maxParticipants">Максимум    участников
*</label>
                    <input
                      id="maxParticipants"
                      type="number"
                      className={`form-control    ${errors maxParticipants    ?    'is-
invalid' : ''}`}
                      {...register('maxParticipants', { valueAsNumber: true })}
                    />
                    {errors maxParticipants && (
                      <div                                    className="invalid-
feedback">{errors maxParticipants message}</div>
                    )}
                  </div>

                  <div className="form-group">
                    <label htmlFor="basePrice">Базовая цена (₽)</label>
                    <input
                      id="basePrice"
                      type="number"
                      step="0.01"
                      className={`form-control  ${errors basePrice  ?  'is-invalid'  :
''}`}
                      {...register('basePrice', { valueAsNumber: true })}
                    />
                    {errors basePrice && (
                      <div                                    className="invalid-
feedback">{errors basePrice message}</div>
                    )}
```

```
        </div>
      </div>

      {categories.length > 0 && (
        <div className="form-group">
          <label htmlFor="categoryId">Категория</label>
          <select
            id="categoryId"
            className="form-control"
            {...register('categoryId', { valueAsNumber: true })}
          >
            <option value="">Без категории</option>
            {categories.map(category => (
              <option key={category.id} value={category.id}>
                {category.name}
              </option>
            ))}
          </select>
        </div>
      )}

      <div className="form-actions">
        <button type="submit" className="btn btn-primary" disabled={isLoading}>
          {isLoading ? 'Сохранение...' : (initialData ? 'Обновить' : 'Создать')}
        </button>
        <button type="button" className="btn btn-secondary" onClick={() => window.history.back()}>
```

```
                Отмена
              </button>
            </div>
          </form>
    );
};

export default EventForm;
```

src/components/events/EventForm.css:

```
.event-form {
    max-width: 800px;
    margin: 0 auto;
    background: var(--white);
    padding: 2rem;
    border-radius: 8px;
    box-shadow: var(--box-shadow);
}

.form-row {
    display: flex;
    gap: 1rem;
}

.form-row .form-group {
    flex: 1;
}

.form-actions {
```

```
  display: flex;
  gap: 1rem;
  justify-content: flex-end;
  margin-top: 2rem;
}

.btn-secondary {
  background-color: #95a5a6;
  color: white;
}

.btn-secondary:hover {
  background-color: #7f8c8d;
}

@media (max-width: 768px) {
  .form-row {
    flex-direction: column;
    gap: 0;
  }
}
```

3.4. Компонент EventList

src/components/events/EventList.jsx:

```
import React from 'react';
import EventCard from './EventCard';
import Loader from '../common/Loader';
import Pagination from '../common/Pagination';
import './EventList.css';
```

```
const EventList = ({
    events,
    loading,
    error,
    onRetry,
    onDelete,
    isAdmin = false,
    pagination,
    onPageChange
}) => {
    if (loading) {
        return <Loader />;
    }

    if (error) {
        return (
            <div className="event-list-error">
                <div className="alert alert-error">
                    <p>Ошибка загрузки мероприятий: {error}</p>
                    <button onClick={onRetry} className="btn btn-primary">
                        Повторить
                    </button>
                </div>
            </div>
        );
    }

    if (!events || events.length === 0) {
```

```
    return (
      <div className="event-list-empty">
        <p>Мероприятия не найдены</p>
        {isAdmin && (
          <button
            className="btn btn-primary"
            onClick={() => window location href = '/events/create'}
          >
            Создать первое мероприятие
          </button>
        )}
      </div>
    );
  }

  return (
    <div className="event-list">
      <div className="events-grid">
        {events map(event => (
          <EventCard
            key={event id}
            event={event}
            onDelete={onDelete}
            isAdmin={isAdmin}
          />
        ))}
      </div>

      {pagination && pagination totalPages > 1 && (
```

```
            <Pagination
              currentPage={pagination.pageNumber}
              totalPages={pagination.totalPages}
              onPageChange={onPageChange}
            />
          )}
        </div>
    );
};

export default EventList;
```

src/components/events/EventList.css:

```
.event-list {
  width: 100%;
}

.events-grid {
  display: grid;
  grid-template-columns: repeat(auto-fill, minmax(350px, 1fr));
  gap: 1.5rem;
  margin-bottom: 2rem;
}

.event-list-error {
  text-align: center;
  padding: 2rem;
}
```

```
.event-list-empty {
    text-align: center;
    padding: 3rem;
    background: var(--white);
    border-radius: 8px
}

.event-list-empty p {
    margin-bottom: 1rem;
    color: var(--text-light);
}

@media (max-width: 768px) {
    .events-grid {
        grid-template-columns: 1fr;
    }
}
```

3.5. Компонент Pagination

src/components/common/Pagination.jsx:

```
import React from 'react';
import './Pagination.css';

const Pagination = ({ currentPage, totalPages, onPageChange }) => {
    const getVisiblePages = () => {
        const delta = 2;
        const range = [];
        const rangeWithDots = [];
        let l
```

```
    for (let i = 1; i <= totalPages; i++) {
      if (i === 1 || i === totalPages || (i >= currentPage - delta && i <=
currentPage + delta)) {
        range.push(i);
      }
    }

    range.forEach((i) => {
      if (l) {
        if (i - l === 2) {
          rangeWithDots.push(l + 1);
        } else if (i - l !== 1) {
          rangeWithDots.push('...');
        }
      }
      rangeWithDots.push(i);
      l = i;
    });

    return rangeWithDots;
  };

  if (totalPages <= 1) return null;

  return (
    <div className="pagination">
      <button
        className="pagination-btn"
```

```
                onClick={() => onPageChange(currentPage - 1)}
                disabled={currentPage === 0}
            >
                « Предыдущая
            </button>

            <div className="pagination-pages">
                {getVisiblePages().map((page, index) => (
                    <button
                        key={index}
                        className={`pagination-page ${page === currentPage + 1 ? 'active' : ''}`}
                        onClick={() => typeof page === 'number' && onPageChange(page - 1)}
                        disabled={page === '...'}
                    >
                        {page}
                    </button>
                ))}
            </div>

            <button
                className="pagination-btn"
                onClick={() => onPageChange(currentPage + 1)}
                disabled={currentPage + 1 >= totalPages}
            >
                Следующая »
            </button>
        </div>
```

```
    );
};

export default Pagination;
```

src/components/common/Pagination.css:

```css
.pagination {
    display: flex;
    justify-content: center;
    align-items: center;
    gap: 0.5rem;
    margin: 2rem 0;
    flex-wrap: wrap;
}

.pagination-btn {
    padding: 0.5rem 1rem;
    border: 1px solid #ddd;
    background: var(--white);
    border-radius: 4px;
    cursor: pointer;
    transition: all 0.3s ease;
}

.pagination-btn:hover:not(:disabled) {
    background: var(--primary-color);
    color: var(--white);
    border-color: var(--primary-color);
}
```

```css
.pagination-btn:disabled {
  opacity: 0.5;
  cursor: not-allowed;
}

.pagination-pages {
  display: flex;
  gap: 0.25rem;
}

.pagination-page {
  width: 36px;
  height: 36px;
  display: flex;
  align-items: center;
  justify-content: center;
  border: 1px solid #ddd;
  background: var(--white);
  border-radius: 4px;
  cursor: pointer;
  transition: all 0.3s ease;
}

.pagination-page:hover:not(.active) {
  background: #f0f0f0;
}

.pagination-page.active {
```

```
    background: var(--primary-color);
    color: var(--white);
    border-color: var(--primary-color);
}

.pagination-page:disabled {
    cursor: default;
    background: transparent;
}
```

3.6. Обновлённые страницы

src/pages/EventsPage.jsx:

```
import React, { useState, useEffect } from 'react';
import EventList from '../components/events/EventList';
import { eventApi } from '../api/eventApi'; // будет в ЛР22

const EventsPage = () => {
    const [events, setEvents] = useState([]);
    const [loading, setLoading] = useState(true);
    const [error, setError] = useState(null);
    const [pagination, setPagination] = useState({
        pageNumber: 0,
        pageSize: 10,
        totalElements: 0,
        totalPages: 0
    });

    const user = JSON.parse(localStorage.getItem('user') || '{}');
    const isAdmin = user.role === 'ADMIN';
```

```
  const loadEvents = async (page = 0) => {
    setLoading(true);
    setError(null);

    try {
      // TODO: заменить на реальный API-вызов (ЛР22)
      // const response = await eventApi.getEvents(page, 10);
      // setEvents(response.data.content);
      // setPagination(response.data);

      // Временная заглушка
      await new Promise(resolve => setTimeout(resolve, 500));
      setEvents([]);
      setPagination({
        pageNumber: 0,
        pageSize: 10,
        totalElements: 0,
        totalPages: 0
      });
    } catch (err) {
      setError(err.response?.data?.message || 'Ошибка загрузки мероприятий');
    } finally {
      setLoading(false);
    }
  };

  const handleDelete = async (id) => {
```

```
    if (!window.confirm('Вы уверены, что хотите удалить это мероприятие?')) return;

    try {
      // TODO: заменить на реальный API-вызов
      // await eventApi.deleteEvent(id);
      await loadEvents(pagination.pageNumber);
    } catch (err) {
      setError(err.response?.data?.message || 'Ошибка удаления');
    }
  };

  useEffect(() => {
    loadEvents();
  }, []);

  return (
    <div className="container">
      <div className="page-header">
        <h1>Мероприятия</h1>
        {isAdmin && (
          <button
            className="btn btn-primary"
            onClick={() => window.location.href = '/events/create'}
          >
            + Создать мероприятие
          </button>
        )}
      </div>
```

```
          <EventList
            events={events}
            loading={loading}
            error={error}
            onRetry={() => loadEvents(pagination.pageNumber)}
            onDelete={handleDelete}
            isAdmin={isAdmin}
            pagination={pagination}
            onPageChange={(newPage) => loadEvents(newPage)}
          />
        </div>
    );
};

export default EventsPage;
```

src/pages/CreateEventPage.jsx:

```
import React, { useState } from 'react';
import { useNavigate } from 'react-router-dom';
import EventForm from '../components/events/EventForm';
import { eventApi } from '../api/eventApi';

const CreateEventPage = () => {
    const navigate = useNavigate();
    const [isLoading, setIsLoading] = useState(false);
    const [error, setError] = useState(null);

    const handleSubmit = async (data) => {
        setIsLoading(true);
```

```
        setError(null);

        try {
          // TODO: заменить на реальный API-вызов (ЛР22)
          // await eventApi.createEvent(data);
          await new Promise(resolve => setTimeout(resolve, 1000));
          navigate('/events');
        } catch (err) {
          setError(err.response?.data?.message || 'Ошибка создания мероприятия');
        } finally {
          setIsLoading(false);
        }
      };

      return (
        <div className="container">
          <h1>Создание мероприятия</h1>
          {error && <div className="alert alert-error">{error}</div>}
          <EventForm onSubmit={handleSubmit} isLoading={isLoading} />
        </div>
      );
    };

    export default CreateEventPage;
```

src/pages/EditEventPage.jsx:

```
import React, { useState, useEffect } from 'react';
import { useParams, useNavigate } from 'react-router-dom';
import EventForm from '../components/events/EventForm';
```

```
import Loader from '../components/common/Loader';
import { eventApi } from '../api/eventApi';

const EditEventPage = () => {
    const { id } = useParams();
    const navigate = useNavigate();
    const [event, setEvent] = useState(null);
    const [loading, setLoading] = useState(true);
    const [isSubmitting, setIsSubmitting] = useState(false);
    const [error, setError] = useState(null);

    useEffect(() => {
        const loadEvent = async () => {
            try {
                // TODO: заменить на реальный API-вызов (ЛР22)
                // const response = await eventApi.getEventById(id);
                // setEvent(response.data);
                await new Promise(resolve => setTimeout(resolve, 500));
                setEvent({
                    id: parseInt(id),
                    title: 'Тестовое мероприятие',
                    description: 'Описание тестового мероприятия',
                    startDate: '2025-12-01T10:00:00',
                    endDate: '2025-12-01T18:00:00',
                    maxParticipants: 100,
                    basePrice: 5000,
                    categoryId: null
                });
            } catch (err) {
```

```
                setError(err.response?.data?.message || 'Мероприятие не найдено');
            } finally {
                setLoading(false);
            }
        };

        loadEvent();
    }, [id]);

    const handleSubmit = async (data) => {
        setIsSubmitting(true);

        try {
            // TODO: заменить на реальный API-вызов
            // await eventApi.updateEvent(id, data);
            await new Promise(resolve => setTimeout(resolve, 1000));
            navigate(`/events/${id}`);
        } catch (err) {
            setError(err.response?.data?.message || 'Ошибка обновления мероприятия');
        } finally {
            setIsSubmitting(false);
        }
    };

    if (loading) return <Loader />;
    if (error) return <div className="alert alert-error">{error}</div>;
```

```
    return (
        <div className="container">
            <h1>Редактирование мероприятия</h1>
            <EventForm
                initialData={event}
                onSubmit={handleSubmit}
                isLoading={isSubmitting}
            />
        </div>
    );
};

export default EditEventPage;
```

4. МЕТОДИКА ВЫПОЛНЕНИЯ

Шаг 1: Установка зависимостей

npm install react-hook-form zod @hookform/resolvers

Шаг 2: Разработка EventCard

Создать компонент с отображением информации

Добавить статус-бейдж

Добавить индикатор заполнения мест

Добавить кнопки действий

Шаг 3: Разработка EventForm

Создать схему валидации Zod

Настроить useForm с zodResolver

Добавить все поля формы

Реализовать отображение ошибок

Шаг 4: Разработка EventList

Создать компонент списка

[] Добавить обработку состояний (loading, error, empty)

Интегрировать EventCard

Шаг 5: Разработка Pagination

Создать компонент пагинации

Реализовать отображение страниц с многоточием

Шаг 6: Интеграция на страницы

Обновить EventsPage

Обновить CreateEventPage

Обновить EditEventPage

5. КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Для чего используется React Hook Form?
2. Как выполняется валидация с помощью Zod?
3. Что такое register в React Hook Form?
4. Как отобразить ошибки валидации в форме?
5. Как реализовать пагинацию с отображением текущей страницы?
6. Как передать данные из формы в родительский компонент?
7. Как загрузить данные мероприятия в форму для редактирования?

6. ТРЕБОВАНИЯ К ОТЧЕТУ

Структура отчета:

Титульный лист

Содержание

Введение

EventCard (листинг + скриншот)

EventForm (листинг + схема валидации)

EventList (листинг)

Pagination (листинг)

Скриншоты работы приложения

Заключение

Имя файла: Фамилия_Имя_ЛР21_ПИ.pdf

7. КРИТЕРИИ ОЦЕНИВАНИЯ

Критерий	Макс. балл	Описание
EventCard	5	Отображение, кнопки, статусы
EventForm	6	Валидация, обработка ошибок
EventList	5	Состояния (loading, error, empty)
Pagination	4	Навигация, отображение
Интеграция со страницами	3	Create, Edit, List
Оформление отчета	2	Соответствие требованиям

Критерий	Макс. балл	Описание
ИТОГО	25	

Лабораторная работа №22

Интеграция React с REST API (Axios)

1. Цель и задачи работы

Цель: Освоить интеграцию React-приложения с REST API бэкенда с использованием библиотеки Axios, включая настройку HTTP-клиента, обработку запросов, управление состояниями загрузки и ошибок, а также работу с JWT-токенами для аутентификации.

Задачи:

1. Настроить Axios-клиент с базовым URL и интерцепторами
2. Реализовать API-сервисы для работы с мероприятиями и аутентификацией
3. Интегрировать API-вызовы в компоненты (EventList, EventForm, Auth)
4. Реализовать управление состояниями (loading, error, success)
5. Настроить автоматическое обновление access-токена
6. Обрабатывать ошибки API и показывать уведомления пользователю

2. ТЕОРЕТИЧЕСКОЕ ОБОСНОВАНИЕ

2.1. Что такое Axios?

Axios -это Promise-based HTTP-клиент для браузера и Node.js, который упрощает выполнение HTTP-запросов к REST API.

Характеристика	Описание
Поддержка Promise	Работает с async/await
Автоматическая трансформация JSON	Не нужно вызывать response.json()
Интерцепторы	Перехват запросов и ответов
Отмена запросов	Возможность отмены запросов
Таймауты	Настройка времени ожидания

2.2. Сравнение Axios и Fetch API

Характеристика	Axios	Fetch API
Синтаксис	Проще	Сложнее
JSON по умолчанию	Да	Нет
Интерцепторы	Есть	Нет
Отмена запросов	Есть	AbortController
Прогресс загрузки	Есть	Нет

2.3. Структура API-сервисов

```
src/api/
├── axios.js          # Настройка Axios (базовый URL, интерцепторы)
```

```
├── eventApi.js        # API для мероприятий
├── authApi.js         # API для аутентификации
├── registrationApi.js # API для регистраций
└── categoryApi.js     # API для категорий
```

2.4. Интерцепторы Axios

Request Interceptor - добавляет JWT-токен к каждому запросу:

```javascript
api.interceptors.request.use((config) => {
    const token = localStorage.getItem('accessToken');
    if (token) {
        config.headers.Authorization = `Bearer ${token}`;
    }
    return config;
});
```

Response Interceptor — обрабатывает ошибки и обновляет токен:

```javascript
api.interceptors.response.use(
    (response) => response,
    async (error) => {
        if (error.response?.status === 401) {
            // Попытка обновить токен
            const newToken = await refreshToken();
            if (newToken) {
                return api.request(error.config);
            }
        }
        return Promise.reject(error);
    }
```

```
);
```

3. ПРИМЕР ВЫПОЛНЕНИЯ (СКВОЗНОЙ ПРОЕКТ)

3.1. Установка зависимостей

```
npm install axios
npm install react-hot-toast   # для уведомлений
```

3.2. Настройка Axios-клиента

src/api/axios.js:

javascript

```
import axios from 'axios';

// Создание экземпляра Axios с базовыми настройками
const api = axios.create({
    baseURL: import.meta.env.VITE_API_URL || 'http://localhost:8080/api',
    timeout: 10000,
    headers: {
        'Content-Type': 'application/json',
    },
});

// Флаг для предотвращения множественных запросов на обновление токена
let isRefreshing = false;
let failedQueue = [];

const processQueue = (error, token = null) => {
    failedQueue.forEach(prom => {
        if (error) {
            prom.reject(error);
```

```
        } else {
            prom resolve(token);
        }
    });
    failedQueue = [];
};

// Request Interceptor: добавление JWT токена
api interceptors request use(
    (config) => {
        const accessToken = localStorage getItem('accessToken');
        if (accessToken) {
            config headers Authorization = `Bearer ${accessToken}`;
        }
        return config;
    },
    (error) => Promise reject(error)
);

// Response Interceptor: обработка ошибок и обновление токена
api interceptors response use(
    (response) => response,
    async (error) => {
        const originalRequest = error config;

        // Если ошибка 401 и это не повторный запрос
        if (error response?.status === 401 && !originalRequest _retry) {
            if (isRefreshing) {
                // Если уже идёт обновление, добавить в очередь
```

```
        return new Promise((resolve, reject) => {
          failedQueue.push({ resolve, reject });
        })
          .then(() => api(originalRequest))
          .catch((err) => Promise.reject(err));
      }

      originalRequest._retry = true;
      isRefreshing = true;

      try {
        const refreshToken = localStorage.getItem('refreshToken');
        if (!refreshToken) {
          throw new Error('No refresh token');
        }

        const response = await axios.post(
          `${api.defaults.baseURL}/auth/refresh`,
          { refreshToken }
        );

        const { accessToken, refreshToken: newRefreshToken } =
response.data;
        localStorage.setItem('accessToken', accessToken);
        localStorage.setItem('refreshToken', newRefreshToken);

        processQueue(null, accessToken);
        return api(originalRequest);
```

```
            } catch (refreshError) {
                processQueue(refreshError);
                // Очистка токенов при ошибке обновления
                localStorage.removeItem('accessToken');
                localStorage.removeItem('refreshToken');
                localStorage.removeItem('user');
                window.location.href = '/login';
                return Promise.reject(refreshError);
            } finally {
                isRefreshing = false;
            }
        }

        return Promise.reject(error);
    }
);

export default api;
```

3.3. API-сервисы

src/api/eventApi.js:

```
import api from './axios';

export const eventApi = {
    // Получение списка мероприятий с пагинацией и фильтрацией
    getEvents: (page = 0, size = 10, filters = {}) => {
        const params = new URLSearchParams({
            page,
            size,
```

```
            ...filters
        });
        return api.get(`/events?${params.toString()}`);
    },

    // Получение мероприятия по ID
    getEventById: (id) => api.get(`/events/${id}`),

    // Создание мероприятия
    createEvent: (eventData) => api.post('/events', eventData),

    // Полное обновление мероприятия
    updateEvent: (id, eventData) => api.put(`/events/${id}`, eventData),

    // Частичное обновление мероприятия
    patchEvent: (id, eventData) => api.patch(`/events/${id}`, eventData),

    // Удаление мероприятия
    deleteEvent: (id) => api.delete(`/events/${id}`),

    // Публикация мероприятия
    publishEvent: (id) => api.patch(`/events/${id}/publish`),

    // Отмена мероприятия
    cancelEvent: (id) => api.patch(`/events/${id}/cancel`)
};
```

src/api/authApi.js:

```
import api from './axios';
```

```
export const authApi = {
  // Вход в систему
  login: (credentials) => api.post('/auth/login', credentials),

  // Регистрация
  register: (userData) => api.post('/auth/register', userData),

  // Обновление токена
  refreshToken: (refreshToken) => api.post('/auth/refresh', { refreshToken }),

  // Выход
  logout: () => api.post('/auth/logout'),

  // Получение профиля
  getProfile: () => api.get('/users/me'),

  // Обновление профиля
  updateProfile: (userData) => api.put('/users/me', userData)
};
```

src/api/registrationApi.js:

```
import api from './axios';

export const registrationApi = {
  // Получение регистраций пользователя
  getUserRegistrations: () => api.get('/registrations/my'),

  // Получение регистраций на мероприятие
```

```
    getEventRegistrations: (eventId) =>
api.get(`/registrations/event/${eventId}`),

    // Регистрация на мероприятие
    registerForEvent: (eventId) => api.post(`/events/${eventId}/register`),

    // Отмена регистрации
    cancelRegistration: (registrationId) =>
api.delete(`/registrations/${registrationId}`),

    // Расчёт цены
    calculatePrice: (eventId, userId, quantity, promoCode = null) => {
      const params = { userId, quantity };
      if (promoCode) params.promoCode = promoCode;
      return api.get(`/events/${eventId}/calculate-price`, { params });
    }
  };
```

src/api/categoryApi.js:

```
import api from './axios';

export const categoryApi = {
  // Получение всех категорий
  getCategories: () => api.get('/categories'),

  // Получение категории по ID
  getCategoryById: (id) => api.get(`/categories/${id}`),

  // Создание категории
  createCategory: (categoryData) => api.post('/categories', categoryData),
```

```
    // Обновление категории
    updateCategory: (id, categoryData) => api.put(`/categories/${id}`, categoryData),

    // Удаление категории
    deleteCategory: (id) => api.delete(`/categories/${id}`)
};
```

3.4. Файл с переменными окружения

.env:

```
env
VITE_API_URL=http://localhost:8080/api
```

.env.example:

```
env
VITE_API_URL=http://localhost:8080/api
```

3.5. Утилиты для уведомлений

src/utils/toastUtils.js:

```
import toast from 'react-hot-toast';

export const showSuccess = (message) => {
    toast.success(message, {
        duration: 3000,
        position: 'top-right',
    });
};

export const showError = (message) => {
    toast.error(message, {
```

```
        duration: 4000,
        position: 'top-right',
    });
};

export const showInfo = (message) => {
    toast(message, {
        duration: 3000,
        position: 'top-right',
        icon: 'ℹ',
    });
};

export const showLoading = (message) => {
    return toast.loading(message, {
        position: 'top-right',
    });
};
```

3.6. Обновлённые компоненты с API-интеграцией

src/pages/EventsPage.jsx:

```
import React, { useState, useEffect } from 'react';
import { eventApi } from '../api/eventApi';
import EventList from '../components/events/EventList';
import { showSuccess, showError } from '../utils/toastUtils';
import toast from 'react-hot-toast';

const EventsPage = () => {
    const [events, setEvents] = useState([]);
```

```
    const [loading, setLoading] = useState(true);
    const [error, setError] = useState(null);
    const [pagination, setPagination] = useState({
        pageNumber: 0,
        pageSize: 10,
        totalElements: 0,
        totalPages: 0
    });

    const user = JSON.parse(localStorage.getItem('user') || '{}');
    const isAdmin = user.role === 'ADMIN';

    const loadEvents = async (page = 0, filters = {}) => {
        setLoading(true);
        setError(null);

        try {
            const response = await eventApi.getEvents(page, 10, filters);
            setEvents(response.data.content);
            setPagination({
                pageNumber: response.data.pageNumber,
                pageSize: response.data.pageSize,
                totalElements: response.data.totalElements,
                totalPages: response.data.totalPages
            });
        } catch (err) {
            const errorMessage = err.response?.data?.message || 'Ошибка
загрузки мероприятий';
            setError(errorMessage);
```

```
            showError(errorMessage);
        } finally {
            setLoading(false);
        }
    };

    const handleDelete = async (id) => {
        if (!window.confirm('Вы уверены, что хотите удалить это мероприятие?')) return;

        const loadingToast = toast.loading('Удаление...');

        try {
            await eventApi.deleteEvent(id);
            showSuccess('Мероприятие успешно удалено');
            await loadEvents(pagination.pageNumber);
        } catch (err) {
            const errorMessage = err.response?.data?.message || 'Ошибка удаления';
            showError(errorMessage);
        } finally {
            toast.dismiss(loadingToast);
        }
    };

    const handlePublish = async (id) => {
        const loadingToast = toast.loading('Публикация...');

        try {
```

```
            await eventApi.publishEvent(id);
            showSuccess('Мероприятие опубликовано');
            await loadEvents(pagination.pageNumber);
        } catch (err) {
            const errorMessage = err.response?.data?.message || 'Ошибка публикации';
            showError(errorMessage);
        } finally {
            toast.dismiss(loadingToast);
        }
    };

    useEffect(() => {
        loadEvents();
    }, []);

    return (
        <div className="container">
            <div className="page-header">
                <h1>Мероприятия</h1>
                {isAdmin && (
                    <button
                        className="btn btn-primary"
                        onClick={() => window.location.href = '/events/create'}
                    >
                        + Создать мероприятие
                    </button>
                )}
            </div>
```

```
            <EventList
                events={events}
                loading={loading}
                error={error}
                onRetry={() => loadEvents(pagination.pageNumber)}
                onDelete={handleDelete}
                onPublish={handlePublish}
                isAdmin={isAdmin}
                pagination={pagination}
                onPageChange={(newPage) => loadEvents(newPage)}
            />
        </div>
    );
};

export default EventsPage;
```

src/pages/LoginPage.jsx:

```
import React, { useState } from 'react';
import { useNavigate, Link } from 'react-router-dom';
import { authApi } from '../api/authApi';
import { showSuccess, showError, showLoading } from '../utils/toastUtils';
import './AuthPages.css';

const LoginPage = () => {
    const navigate = useNavigate();
    const [formData, setFormData] = useState({
        email: '',
```

```
    password: ''
  });
  const [error, setError] = useState('');
  const [loading, setLoading] = useState(false);

  const handleChange = (e) => {
    setFormData({
      ...formData,
      [e.target.name]: e.target.value
    });
  };

  const handleSubmit = async (e) => {
    e.preventDefault();
    setError('');
    setLoading(true);

    try {
      const response = await authApi.login(formData);
      const { accessToken, refreshToken, user } = response.data;

      localStorage.setItem('accessToken', accessToken);
      localStorage.setItem('refreshToken', refreshToken);
      localStorage.setItem('user', JSON.stringify(user));

      showSuccess(`Добро пожаловать, ${user.name}!`);
      navigate('/');
    } catch (err) {
```

```
            const errorMessage = err.response?.data?.message || 'Неверный email или пароль';
            setError(errorMessage);
            showError(errorMessage);
        } finally {
            setLoading(false);
        }
    };

    return (
        <div className="auth-page">
            <div className="auth-card">
                <h2>Вход в систему</h2>

                {error && <div className="alert alert-error">{error}</div>}

                <form onSubmit={handleSubmit}>
                    <div className="form-group">
                        <label htmlFor="email">Email</label>
                        <input
                            type="email"
                            id="email"
                            name="email"
                            className="form-control"
                            value={formData.email}
                            onChange={handleChange}
                            required
                        />
                    </div>
```

```
                <div className="form-group">
                  <label htmlFor="password">Пароль</label>
                  <input
                    type="password"
                    id="password"
                    name="password"
                    className="form-control"
                    value={formData password}
                    onChange={handleChange}
                    required
                  />
                </div>

                <button type="submit" className="btn btn-primary" disabled={loading}>
                  {loading ? 'Вход...' : 'Войти'}
                </button>
              </form>

              <p className="auth-link">
                Нет аккаунта? <Link to="/register">Зарегистрироваться</Link>
              </p>
            </div>
          </div>
        );
      };
```

```
export default LoginPage;
```

src/pages/CreateEventPage.jsx:

```
import React, { useState, useEffect } from 'react';
import { useNavigate } from 'react-router-dom';
import { eventApi } from '../api/eventApi';
import { categoryApi } from '../api/categoryApi';
import EventForm from '../components/events/EventForm';
import { showSuccess, showError, showLoading } from '../utils/toastUtils';
import toast from 'react-hot-toast';

const CreateEventPage = () => {
  const navigate = useNavigate();
  const [isLoading, setIsLoading] = useState(false);
  const [categories, setCategories] = useState([]);
  const [error, setError] = useState(null);

  useEffect(() => {
    const loadCategories = async () => {
      try {
        const response = await categoryApi.getCategories();
        setCategories(response.data);
      } catch (err) {
        console.error('Failed to load categories:', err);
      }
    };
    loadCategories();
  }, []);
```

```
  const handleSubmit = async (data) => {
    setIsLoading(true);
    setError(null);
    const loadingToast = showLoading('Создание мероприятия...');

    try {
      const response = await eventApi createEvent(data);
      showSuccess('Мероприятие успешно создано');
      navigate(`/events/${response data id}`);
    } catch (err) {
      const errorMessage = err response?.data?.message || 'Ошибка создания мероприятия';
      setError(errorMessage);
      showError(errorMessage);
    } finally {
      toast dismiss(loadingToast);
      setIsLoading(false);
    }
  };

  return (
    <div className="container">
      <h1>Создание мероприятия</h1>
      {error && <div className="alert alert-error">{error}</div>}
      <EventForm
        onSubmit={handleSubmit}
        isLoading={isLoading}
        categories={categories}
      />
```

```
        </div>
    );
};

export default CreateEventPage;
```

3.7. Обновление App.jsx (добавление Toaster)

src/App.jsx (фрагмент):

```
import { Toaster } from 'react-hot-toast';
// ... остальные импорты

function App() {
    return (
        <Router>
            <Toaster position="top-right" />
            <div className="app">
                <Navbar />
                <main className="main-content">
                    <Routes>
                        {/* маршруты */}
                    </Routes>
                </main>
                <Footer />
            </div>
        </Router>
    );
}
```

3.8. Кастомный хук для работы с API

src/hooks/useApi.js:

```
import { useState, useCallback } from 'react';
import { showError } from '../utils/toastUtils';

export const useApi = (apiFunction, options = {}) => {
  const [data, setData] = useState(null);
  const [loading, setLoading] = useState(false);
  const [error, setError] = useState(null);

  const execute = useCallback(async (...args) => {
    setLoading(true);
    setError(null);

    try {
      const response = await apiFunction(...args);
      setData(response.data);
      if (options.onSuccess) {
        options.onSuccess(response.data);
      }
      return response.data;
    } catch (err) {
      const errorMessage = err.response?.data?.message || 'Произошла ошибка';
      setError(errorMessage);
      if (options.onError) {
        options.onError(errorMessage);
      }
      if (options.showError !== false) {
```

```
            showError(errorMessage);
          }
          throw err;
        } finally {
          setLoading(false);
        }
    }, [apiFunction, options]);

    return { data, loading, error, execute };
};
```

Использование хука:

```
import { useApi } from '../hooks/useApi';
import { eventApi } from '../api/eventApi';

const EventsPage = () => {
    const { data: events, loading, error, execute: loadEvents } = useApi(
        () => eventApi.getEvents(0, 10),
        { onSuccess: (data) => console.log('Loaded', data) }
    );

    useEffect(() => {
        loadEvents();
    }, []);

    // ...
};
```

4. МЕТОДИКА ВЫПОЛНЕНИЯ

Шаг 1: Настройка Axios

Создать файл src/api/axios.js

Настроить baseURL

Добавить интерцептор для токена

Добавить интерцептор для обновления токена

Шаг 2: Создание API-сервисов

Создать eventApi.js

Создать authApi.js

Создать registrationApi.js

Создать categoryApi.js

Шаг 3: Настройка уведомлений

Установить react-hot-toast

Создать toastUtils.js

Добавить Toaster в App

Шаг 4: Интеграция API в компоненты

Обновить LoginPage

Обновить RegisterPage

Обновить EventsPage

Обновить CreateEventPage

Обновить EditEventPage

Шаг 5: Создание кастомного хука useApi

Создать хук для унификации API-вызовов

5. КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Для чего нужны интерцепторы в Axios?

2. Как добавить JWT-токен к каждому запросу?
3. Как обработать ошибку 401 и обновить токен?
4. Как предотвратить множественные запросы на обновление токена?
5. Как использовать кастомный хук useApi для унификации запросов?
6. Как показать уведомление об успехе/ошибке?

6. ТРЕБОВАНИЯ К ОТЧЕТУ

Структура отчета:

Титульный лист

Содержание

Введение

Настройка Axios (листинг)

API-сервисы (листинг)

Кастомный хук useApi

Интеграция в компоненты (листинг)

Скриншоты работы приложения

Заключение

Имя файла: Фамилия_Имя_ЛР22_ПИ.pdf

7. КРИТЕРИИ ОЦЕНИВАНИЯ

Критерий	Макс. балл	Описание
Настройка Axios	4	baseURL, интерцепторы
API-сервисы	5	eventApi, authApi,

Критерий	Макс. балл	Описание
		registrationApi
Управление токенами	4	Обновление, очередь запросов
Интеграция в компоненты	6	EventsPage, LoginPage, CreateEventPage
Обработка ошибок	4	Уведомления, fallback
Кастомный хук useApi	3	Унификация запросов
Оформление отчета	2	Соответствие требованиям
ИТОГО	28	

Лабораторная работа №23

Валидация форм (React Hook Form + Zod)

1. Цель и задачи работы

Цель: Освоить продвинутые методы валидации форм в React с использованием библиотек React Hook Form и Zod для создания надёжных,

типобезопасных и удобных для пользователя форм регистрации, логина, создания и редактирования мероприятий.

Задачи:

1. Изучить интеграцию React Hook Form с Zod для схемной валидации
2. Реализовать сложную валидацию с условными правилами и кастомными валидаторами
3. Добавить отложенную валидацию (onBlur) и валидацию в реальном времени
4. Реализовать динамические формы (поля, добавляемые пользователем)
5. Настроить глобальную и локальную обработку ошибок
6. Создать переиспользуемые компоненты полей ввода с валидацией

2. ТЕОРЕТИЧЕСКОЕ ОБОСНОВАНИЕ

2.1. React Hook Form + Zod: преимущества

Характеристика	**React Hook Form без Zod**	**React Hook Form + Zod**
Типизация	Ручная	Автоматическая из схемы
Сложные правила	Кастомные функции	Декларативные методы
Переиспользование	Копирование кода	Единая схема

Характеристика	React Hook Form без Zod	React Hook Form + Zod
Читаемость	Ниже	Выше

2.2. Схема Zod для мероприятия (сложная)

```
import { z } from 'zod';

// Кастомный валидатор для будущей даты
const futureDate = (date: string) => {
  return new Date(date) > new Date();
};

// Схема для мероприятия
export const eventSchema = z.object({
  title: z.string()
    .min(3, 'Название должно содержать минимум 3 символа')
    .max(200, 'Название не должно превышать 200 символов')
    .regex(/^[а-яА-Яа-zA-Z0-9\s\-.,!?()]+$/, 'Название содержит недопустимые символы'),

  description: z.string()
    .max(5000, 'Описание не должно превышать 5000 символов')
    .optional()
    .transform(val => val || ''),

  startDate: z.string()
    .min(1, 'Дата начала обязательна')
```

```
    .refine(futureDate, 'Дата начала должна быть в будущем'),

  endDate: z.string()
    .min(1, 'Дата окончания обязательна'),

  maxParticipants: z.number({
    required_error: 'Максимальное количество участников обязательно',
    invalid_type_error: 'Введите число'
  })
    .int('Количество участников должно быть целым числом')
    .positive('Количество участников должно быть положительным')
    .max(10000, 'Максимум 10000 участников'),

  basePrice: z.number()
    .min(0, 'Цена не может быть отрицательной')
    .optional()
    .default(0),

  categoryId: z.number().optional().nullable(),

  sessions: z.array(z.object({
    title: z.string().min(1, 'Название сессии обязательно'),
    speaker: z.string().min(1, 'Спикер обязателен'),
    startTime: z.string().min(1, 'Время начала обязательно'),
    endTime: z.string().min(1, 'Время окончания обязательно')
  })).optional()
}).refine(data => new Date(data.endDate) > new Date(data.startDate), {
  message: 'Дата окончания должна быть позже даты начала',
  path: ['endDate']
```

```
}).refine(data => {
  if (data.sessions && data.sessions.length > 0) {
    return data.sessions.every(session =>
      new Date(session.endTime) > new Date(session.startTime)
    );
  }
  return true;
}, {
  message: 'Время окончания сессии должно быть позже времени начала',
  path: ['sessions']
});
```

2.3. Типы из схемы Zod

```
// Автоматический вывод типа из схемы
export type EventFormData = z.infer<typeof eventSchema>;
```

2.4. Настройка React Hook Form с Zod

```
import { useForm } from 'react-hook-form';
import { zodResolver } from '@hookform/resolvers/zod';
import { eventSchema } from '../schemas/eventSchema';

const { register, handleSubmit, formState: { errors }, watch, setValue, trigger } = useForm({
  resolver: zodResolver(eventSchema),
  defaultValues: {
    title: '',
    description: '',
    startDate: '',
```

```
    endDate: '',
    maxParticipants: 1,
    basePrice: 0,
    categoryId: null,
    sessions: []
  },
  mode: 'onChange' // валидация при каждом изменении
});
```

2.5. Режимы валидации

Режим	Описание	Когда использовать
onSubmit	Валидация только при отправке	Простые формы
onBlur	Валидация при потере фокуса	Средние формы
onChange	Валидация при каждом изменении	Требовательные пользователи
onTouched	Валидация после первого касания	Баланс между производительностью и UX

3. ПРИМЕР ВЫПОЛНЕНИЯ (СКВОЗНОЙ ПРОЕКТ)

3.1. Установка зависимостей

```
npm install react-hook-form zod @hookform/resolvers
```

3.2. Схемы валидации

src/schemas/authSchemas.js:

```
import { z } from 'zod';

// Схема для логина
export const loginSchema = z.object({
  email: z.string()
    .min(1, 'Email обязателен')
    .email('Введите корректный email'),
  password: z.string()
    .min(1, 'Пароль обязателен')
    .min(6, 'Пароль должен содержать минимум 6 символов')
});

// Схема для регистрации
export const registerSchema = z.object({
  name: z.string()
    .min(2, 'Имя должно содержать минимум 2 символа')
    .max(100, 'Имя не должно превышать 100 символов')
    .regex(/^[a-яА-Яa-zA-Z\s-]+$/, 'Имя может содержать только буквы, пробелы и дефисы'),

  email: z.string()
    .min(1, 'Email обязателен')
    .email('Введите корректный email'),

  password: z.string()
    .min(6, 'Пароль должен содержать минимум 6 символов')
```

```
            .regex(/[A-Z]/, 'Пароль должен содержать хотя бы одну заглавную букву')
            .regex(/[0-9]/, 'Пароль должен содержать хотя бы одну цифру'),

        confirmPassword: z.string()
            .min(1, 'Подтверждение пароля обязательно')
    }).refine(data => data.password === data.confirmPassword, {
        message: 'Пароли не совпадают',
        path: ['confirmPassword']
    });
```

src/schemas/eventSchema.js:

```
import { z } from 'zod';

// Кастомные валидаторы
const futureDate = (date) => {
    const inputDate = new Date(date);
    const now = new Date();
    now.setHours(0, 0, 0, 0);
    return inputDate > now;
};

const validUrl = (url) => {
    if (!url) return true;
    try {
        new URL(url);
        return true;
    } catch {
        return false;
```

```
    }
};

// Схема для мероприятия
export const eventSchema = z.object({
    title: z.string()
        .min(3, 'Название должно содержать минимум 3 символа')
        .max(200, 'Название не должно превышать 200 символов')
        .regex(/^[а-яА-Яа-zA-Z0-9\s\-.,!?()]+$/, 'Название содержит недопустимые символы'),

    description: z.string()
        .max(5000, 'Описание не должно превышать 5000 символов')
        .optional()
        .transform(val => val || ''),

    shortDescription: z.string()
        .max(300, 'Краткое описание не должно превышать 300 символов')
        .optional(),

    startDate: z.string()
        .min(1, 'Дата начала обязательна')
        .refine(futureDate, 'Дата начала должна быть в будущем'),

    endDate: z.string()
        .min(1, 'Дата окончания обязательна'),

    maxParticipants: z.number({
        required_error: 'Максимальное количество участников обязательно',
```

```
    invalid_type_error: 'Введите число'
  })
    .int('Количество участников должно быть целым числом')
    .positive('Количество участников должно быть положительным')
    .min(1, 'Минимум 1 участник')
    .max(10000, 'Максимум 10000 участников'),

  basePrice: z.number()
    .min(0, 'Цена не может быть отрицательной')
    .optional()
    .default(0),

  categoryId: z.number().nullable().optional(),

  location: z.object({
    address: z.string().min(1, 'Адрес обязателен'),
    city: z.string().min(1, 'Город обязателен'),
    coordinates: z.object({
      lat: z.number().optional(),
      lng: z.number().optional()
    }).optional()
  }).optional(),

  imageUrl: z.string()
    .optional()
    .refine(validUrl, 'Введите корректный URL изображения'),

  tags: z.array(z.string()).max(10, 'Максимум 10 тегов').optional(),
```

```
    sessions: z.array(z.object({
      title: z.string().min(1, 'Название сессии обязательно'),
      speaker: z.string().min(1, 'Спикер обязателен'),
      startTime: z.string().min(1, 'Время начала обязательно'),
      endTime: z.string().min(1, 'Время окончания обязательно'),
      room: z.string().optional(),
      description: z.string().max(500, 'Описание не должно превышать 500 символов').optional()
    })).optional().default([])
  }).refine(data => new Date(data.endDate) > new Date(data.startDate), {
    message: 'Дата окончания должна быть позже даты начала',
    path: ['endDate']
  }).refine(data => {
    if (data.sessions && data.sessions.length > 0) {
      return data.sessions.every(session =>
        new Date(session.endTime) > new Date(session.startTime)
      );
    }
    return true;
  }, {
    message: 'Время окончания сессии должно быть позже времени начала',
    path: ['sessions']
  });

export type EventFormData = z.infer<typeof eventSchema>;
```

3.3. Кастомный компонент поля ввода

src/components/common/FormInput.jsx:

```jsx
import React from 'react';
import './FormInput.css';

const FormInput = React.forwardRef(({
  label,
  name,
  type = 'text',
  placeholder,
  error,
  required = false,
  className = '',
  ...props
}, ref) => {
  return (
    <div className="form-group">
      <label htmlFor={name}>
        {label}
        {required && <span className="required-star">*</span>}
      </label>
      <input
        id={name}
        name={name}
        type={type}
        ref={ref}
        className={`form-control ${error ? 'is-invalid' : ''} ${className}`}
        placeholder={placeholder}
        aria-invalid={!!error}
        aria-describedby={error ? `${name}-error` : undefined}
```

```
                {...props}
            />
            {error && (
                <div id={`${name}-error`} className="invalid-feedback">
                    {error message}
                </div>
            )}
        </div>
    );
});

FormInput displayName = 'FormInput';

export default FormInput;
```

src/components/common/FormTextarea.jsx:

```
jsx
import React from 'react';

const FormTextarea = React forwardRef(({
    label,
    name,
    rows = 4,
    placeholder,
    error,
    required = false,
    ...props
}, ref) => {
    // Подсчёт символов
    const [charCount, setCharCount] = React useState(0);
```

```
const handleChange = (e) => {
  setCharCount(e.target.value.length);
  if (props.onChange) props.onChange(e);
};

return (
  <div className="form-group">
    <label htmlFor={name}>
      {label}
      {required && <span className="required-star">*</span>}
      {props.maxLength && (
        <span className="char-counter">
          {charCount} / {props.maxLength}
        </span>
      )}
    </label>
    <textarea
      id={name}
      name={name}
      ref={ref}
      rows={rows}
      className={`form-control ${error ? 'is-invalid' : ''}`}
      placeholder={placeholder}
      onChange={handleChange}
      aria-invalid={!!error}
      {...props}
    />
    {error && (
```

```
            <div className="invalid-feedback">{error.message}</div>
          )}
        </div>
    );
});

FormTextarea.displayName = 'FormTextarea';

export default FormTextarea;
```

src/components/common/FormSelect.jsx:

```
import React from 'react';

const FormSelect = React.forwardRef(({
    label,
    name,
    options,
    placeholder,
    error,
    required = false,
    ...props
}, ref) => {
    return (
        <div className="form-group">
            <label htmlFor={name}>
                {label}
                {required && <span className="required-star">*</span>}
            </label>
            <select
```

```
            id={name}
            name={name}
            ref={ref}
            className={`form-control ${error ? 'is-invalid' : ''}`}
            aria-invalid={!!error}
            {...props}
          >
            <option value="">{placeholder || 'Выберите...'}</option>
            {options.map(option => (
              <option key={option.value} value={option.value}>
                {option.label}
              </option>
            ))}
          </select>
          {error && (
            <div className="invalid-feedback">{error.message}</div>
          )}
        </div>
      );
    });

FormSelect.displayName = 'FormSelect';

export default FormSelect;
```

3.4. Форма логина

src/pages/LoginPage.jsx (с валидацией):

```
import React, { useState } from 'react';
import { useNavigate, Link } from 'react-router-dom';
```

```
import { useForm } from 'react-hook-form';
import { zodResolver } from '@hookform/resolvers/zod';
import { loginSchema } from '../schemas/authSchemas';
import { authApi } from '../api/authApi';
import { showSuccess, showError } from '../utils/toastUtils';
import FormInput from '../components/common/FormInput';

const LoginPage = () => {
  const navigate = useNavigate();
  const [serverError, setServerError] = useState('');
  const [isLoading, setIsLoading] = useState(false);

  const { register, handleSubmit, formState: { errors, isDirty, isValid } } =
useForm({
    resolver: zodResolver(loginSchema),
    defaultValues: {
      email: '',
      password: ''
    },
    mode: 'onChange'
  });

  const onSubmit = async (data) => {
    setIsLoading(true);
    setServerError('');

    try {
      const response = await authApi.login(data);
      const { accessToken, refreshToken, user } = response.data;
```

```
            localStorage.setItem('accessToken', accessToken);
            localStorage.setItem('refreshToken', refreshToken);
            localStorage.setItem('user', JSON.stringify(user));

            showSuccess(`Добро пожаловать, ${user.name}!`);
            navigate('/');
        } catch (err) {
            const errorMessage = err.response?.data?.message || 'Неверный email или пароль';
            setServerError(errorMessage);
            showError(errorMessage);
        } finally {
            setIsLoading(false);
        }
    };

    return (
        <div className="auth-page">
            <div className="auth-card">
                <h2>Вход в систему</h2>

                {serverError && (
                    <div className="alert alert-error">{serverError}</div>
                )}

                <form onSubmit={handleSubmit(onSubmit)} noValidate>
                    <FormInput
                        label="Email"
```

```
        name="email"
        type="email"
        placeholder="ivan@example.com"
        register={register}
        error={errors.email}
        required
      />

      <FormInput
        label="Пароль"
        name="password"
        type="password"
        placeholder="••••••"
        register={register}
        error={errors.password}
        required
      />

      <button
        type="submit"
        className="btn btn-primary"
        disabled={isLoading || !isDirty || !isValid}
      >
        {isLoading ? 'Вход...' : 'Войти'}
      </button>
    </form>

    <p className="auth-link">
```

```
                    Нет аккаунта? <Link to="/register">Зарегистрироваться</Link>
                </p>
            </div>
        </div>
    );
};

export default LoginPage;
```

3.5. Форма регистрации

src/pages/RegisterPage.jsx (с валидацией):

```
import React, { useState } from 'react';
import { useNavigate, Link } from 'react-router-dom';
import { useForm } from 'react-hook-form';
import { zodResolver } from '@hookform/resolvers/zod';
import { registerSchema } from '../schemas/authSchemas';
import { authApi } from '../api/authApi';
import { showSuccess, showError } from '../utils/toastUtils';
import FormInput from '../components/common/FormInput';

const RegisterPage = () => {
    const navigate = useNavigate();
    const [serverError, setServerError] = useState('');
    const [isLoading, setIsLoading] = useState(false);

    const { register, handleSubmit, watch, formState: { errors, isDirty, isValid } } = useForm({
        resolver: zodResolver(registerSchema),
```

```
    defaultValues: {
      name: '',
      email: '',
      password: '',
      confirmPassword: ''
    },
    mode: 'onChange'
  });

  const password = watch('password');

  const onSubmit = async (data) => {
    setIsLoading(true);
    setServerError('');

    try {
      const response = await authApi.register({
        name: data.name,
        email: data.email,
        password: data.password
      });
      const { accessToken, refreshToken, user } = response.data

      localStorage.setItem('accessToken', accessToken);
      localStorage.setItem('refreshToken', refreshToken);
      localStorage.setItem('user', JSON.stringify(user));

      showSuccess(`Добро пожаловать, ${user.name}!`);
      navigate('/');
```

```
    } catch (err) {
      const errorMessage = err.response?.data?.message || 'Ошибка регистрации';
      setServerError(errorMessage);
      showError(errorMessage);
    } finally {
      setIsLoading(false);
    }
  };

  return (
    <div className="auth-page">
      <div className="auth-card">
        <h2>Регистрация</h2>

        {serverError && (
          <div className="alert alert-error">{serverError}</div>
        )}

        <form onSubmit={handleSubmit(onSubmit)} noValidate>
          <FormInput
            label="Имя"
            name="name"
            placeholder="Иван Иванов"
            register={register}
            error={errors.name}
            required
          />
```

```
                    <FormInput
                        label="Email"
                        name="email"
                        type="email"
                        placeholder="ivan@example.com"
                        register={register}
                        error={errors.email}
                        required
                    />

                    <FormInput
                        label="Пароль"
                        name="password"
                        type="password"
                        placeholder="••••••"
                        register={register}
                        error={errors.password}
                        helperText="Минимум 6 символов, одна заглавная буква и одна цифра"
                        required
                    />

                    <FormInput
                        label="Подтверждение пароля"
                        name="confirmPassword"
                        type="password"
                        placeholder="••••••"
                        register={register}
                        error={errors.confirmPassword}
```

```
                required
              />

              {password && (
                <div className="password-strength">
                  <div                  className="strength-label">Сложность
пароля:</div>
                  <div className="strength-bar">
                    <div            className={`strength-fill            strength-
${getPasswordStrength(password)}`} />
                  </div>
                </div>
              )}

              <button
                type="submit"
                className="btn btn-primary"
                disabled={isLoading || !isDirty || !isValid}
              >
                {isLoading ? 'Регистрация...' : 'Зарегистрироваться'}
              </button>
            </form>

            <p className="auth-link">
              Уже есть аккаунт? <Link to="/login">Войти</Link>
            </p>
          </div>
        </div>
      );
```

```
};

// Функция оценки сложности пароля
const getPasswordStrength = (password) => {
    let strength = 0;
    if (password length >= 8) strength++;
    if (/[A-Z]/.test(password)) strength++;
    if (/[0-9]/.test(password)) strength++;
    if (/[^A-Za-z0-9]/.test(password)) strength++;

    const levels = ['weak', 'medium', 'good', 'strong'];
    return levels[strength - 1] || 'weak';
};

export default RegisterPage;
```

3.6. Форма мероприятия с динамическими полями (сессии)

src/components/events/EventForm.jsx (расширенная версия):

```
import React from 'react';
import { useForm, useFieldArray } from 'react-hook-form';
import { zodResolver } from '@hookform/resolvers/zod';
import { eventSchema } from '../../schemas/eventSchema';
import FormInput from '../common/FormInput';
import FormTextarea from '../common/FormTextarea';
import FormSelect from '../common/FormSelect';
import './EventForm.css';

const EventForm = ({ initialData, onSubmit, isLoading, categories = [] }) =>
{
```

```
    const { register, control, handleSubmit, watch, formState: { errors, isDirty, isValid } } = useForm({
      resolver: zodResolver(eventSchema),
      defaultValues: initialData || {
        title: '',
        description: '',
        shortDescription: '',
        startDate: '',
        endDate: '',
        maxParticipants: 1,
        basePrice: 0,
        categoryId: null,
        location: { address: '', city: '' },
        imageUrl: '',
        tags: [],
        sessions: []
      },
      mode: 'onChange'
    });

    const { fields, append, remove } = useFieldArray({
      control,
      name: 'sessions'
    });

    const [tagInput, setTagInput] = React.useState('');
    const tags = watch('tags') || [];
    const startDate = watch('startDate');
```

```
    const handleAddTag = () => {
        if (tagInput.trim() && !tags.includes(tagInput.trim()) && tags.length < 10) {
            const currentTags = watch('tags') || [];
            const updatedTags = [...currentTags, tagInput.trim()];
            // Устанавливаем через setValue, так как useFieldArray не для этого
            // Используем прямой вызов
        }
        setTagInput('');
    };

    const handleRemoveTag = (tagToRemove) => {
        const currentTags = watch('tags') || [];
        const updatedTags = currentTags.filter(tag => tag !== tagToRemove);
        // Устанавливаем через setValue
    };

    const addSession = () => {
        append({
            title: '',
            speaker: '',
            startTime: startDate ? startDate.slice(0, 10) + 'T10:00' : '',
            endTime: startDate ? startDate.slice(0, 10) + 'T11:00' : '',
            room: '',
            description: ''
        });
    };
```

```
    const categoryOptions = categories.map(cat => ({
      value: cat.id,
      label: cat.name
    }));

    return (
      <form onSubmit={handleSubmit(onSubmit)} className="event-form" noValidate>
        <h2>Основная информация</h2>

        <FormInput
          label="Название мероприятия"
          name="title"
          register={register}
          error={errors.title}
          required
        />

        <FormTextarea
          label="Краткое описание"
          name="shortDescription"
          register={register}
          error={errors.shortDescription}
          maxLength={300}
        />

        <FormTextarea
          label="Полное описание"
          name="description"
```

```
        register={register}
        error={errors description}
        rows={6}
        maxLength={5000}
      />

      <div className="form-row">
        <FormInput
          label="Дата и время начала"
          name="startDate"
          type="datetime-local"
          register={register}
          error={errors startDate}
          required
        />

        <FormInput
          label="Дата и время окончания"
          name="endDate"
          type="datetime-local"
          register={register}
          error={errors endDate}
          required
        />
      </div>

      <div className="form-row">
        <FormInput
          label="Максимум участников"
```

```
        name="maxParticipants"
        type="number"
        register={register}
        error={errors maxParticipants}
        required
      />

      <FormInput
        label="Базовая цена (₽)"
        name="basePrice"
        type="number"
        step="0.01"
        register={register}
        error={errors basePrice}
      />
    </div>

    <FormSelect
      label="Категория"
      name="categoryId"
      options={categoryOptions}
      register={register}
      error={errors categoryId}
    />

    <h2>Местоположение</h2>

    <div className="form-row">
      <FormInput
```

```
        label="Страна"
        name="location.country"
        register={register}
        error={errors.location?.country}
      />

      <FormInput
        label="Город"
        name="location.city"
        register={register}
        error={errors.location?.city}
      />
    </div>

    <FormInput
      label="Адрес"
      name="location.address"
      register={register}
      error={errors.location?.address}
    />

    <h2>Изображение и теги</h2>

    <FormInput
      label="URL изображения"
      name="imageUrl"
      type="url"
      register={register}
      error={errors.imageUrl}
```

```
                placeholder="https://example.com/image.jpg"
              />

              <div className="form-group">
                <label>Теги</label>
                <div className="tags-input">
                  <input
                    type="text"
                    value={tagInput}
                    onChange={(e) => setTagInput(e.target.value)}
                    onKeyPress={(e) => e.key === 'Enter' && handleAddTag()}
                    placeholder="Введите тег и нажмите Enter"
                    className="form-control"
                  />
                  <button type="button" onClick={handleAddTag}
className="btn btn-sm btn-secondary">
                    Добавить
                  </button>
                </div>
                <div className="tags-list">
                  {tags.map((tag, index) => (
                    <span key={index} className="tag">
                      {tag}
                      <button type="button" onClick={() =>
handleRemoveTag(tag)}>×</button>
                    </span>
                  ))}
                </div>
```

```
                {errors.tags && <div className="invalid-feedback">{errors.tags.message}</div>}
            </div>

            <h2>Программа мероприятия (сессии)</h2>

            {fields.map((field, index) => (
              <div key={field.id} className="session-card">
                <div className="session-header">
                  <h4>Сессия {index + 1}</h4>
                  <button type="button" onClick={() => remove(index)} className="btn-remove">
                    Удалить
                  </button>
                </div>

                <FormInput
                  label="Название сессии"
                  name={`sessions.${index}.title`}
                  register={register}
                  error={errors.sessions?.[index]?.title}
                  required
                />

                <FormInput
                  label="Спикер"
                  name={`sessions.${index}.speaker`}
                  register={register}
                  error={errors.sessions?.[index]?.speaker}
```

```
        required
      />

      <div className="form-row">
        <FormInput
          label="Время начала"
          name={`sessions.${index}.startTime`}
          type="datetime-local"
          register={register}
          error={errors.sessions?.[index]?.startTime}
          required
        />

        <FormInput
          label="Время окончания"
          name={`sessions.${index}.endTime`}
          type="datetime-local"
          register={register}
          error={errors.sessions?.[index]?.endTime}
          required
        />
      </div>

      <FormInput
        label="Аудитория"
        name={`sessions.${index}.room`}
        register={register}
        error={errors.sessions?.[index]?.room}
      />
```

```
                    <FormTextarea
                        label="Описание сессии"
                        name={`sessions.${index}.description`}
                        register={register}
                        error={errors.sessions?.[index]?.description}
                        rows={2}
                    />
                </div>
            ))}

            <button type="button" onClick={addSession} className="btn btn-secondary">
                + Добавить сессию
            </button>

            <div className="form-actions">
                <button
                    type="submit"
                    className="btn btn-primary"
                    disabled={isLoading || !isDirty || !isValid}
                >
                    {isLoading ? 'Сохранение...' : (initialData ? 'Обновить' : 'Создать')}
                </button>
                <button type="button" className="btn btn-secondary" onClick={() => window.history.back()}>
                    Отмена
                </button>
```

```
        </div>
      </form>
    );
};

export default EventForm;
```

4. МЕТОДИКА ВЫПОЛНЕНИЯ

Шаг 1: Установка зависимостей

npm install react-hook-form zod @hookform/resolvers

Шаг 2: Создание схем валидации

Создать schemas/authSchemas.js

Создать schemas/eventSchema.js

Шаг 3: Создание кастомных компонентов полей

FormInput

FormTextarea

FormSelect

Шаг 4: Обновление формы логина

Интегрировать loginSchema

Добавить кастомные компоненты

Шаг 5: Обновление формы регистрации

Интегрировать registerSchema

Добавить индикатор сложности пароля

Шаг 6: Обновление формы мероприятия

Интегрировать eventSchema

Добавить динамические сессии

Добавить теги

5. КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Для чего нужен zodResolver в React Hook Form?
2. Как выполнить условную валидацию (refine) в Zod?
3. Что такое useFieldArray и для чего он используется?
4. Как реализовать кастомное сообщение об ошибке для refine?
5. Как выполнить отложенную валидацию (debounce)?
6. Как синхронизировать валидацию дат начала и окончания сессий?

6. ТРЕБОВАНИЯ К ОТЧЕТУ

Структура отчета:

Титульный лист

Содержание

Введение

Схемы валидации (листинг)

Кастомные компоненты полей

Интеграция в формы

Скриншоты валидации

Заключение

7. КРИТЕРИИ ОЦЕНИВАНИЯ

Критерий	Макс. балл	Описание
Схемы валидации	6	loginSchema, registerSchema, eventSchema
Кастомные компоненты	5	FormInput, FormTextarea, FormSelect
Форма логина	3	Валидация, отображение ошибок
Форма регистрации	4	Валидация, сложность пароля
Форма мероприятия	5	Динамические поля, условная валидация
Оформление отчета	2	Соответствие требованиям
ИТОГО	25	

Лабораторная работа №24

Адаптивная вёрстка (Bootstrap/Tailwind)

1. Цель работы и задачи

Цель: Освоить технологии адаптивной вёрстки для создания веб-интерфейсов, корректно отображающихся на различных устройствах (десктоп, планшет, мобильный телефон), с использованием CSS-фреймворков Bootstrap и Tailwind CSS.

Задачи:

1. Изучить принципы адаптивного веб-дизайна (RWD)
2. Освоить систему сеток CSS-фреймворков (Bootstrap / Tailwind)
3. Реализовать адаптивную навигацию (бургер-меню)
4. Адаптировать существующие компоненты (EventCard, EventList, формы)
5. Использовать медиа-запросы для тонкой настройки
6. Реализовать мобильную версию страниц (список, детали, формы)

2. ТЕОРЕТИЧЕСКОЕ ОБОСНОВАНИЕ

2.1. Принципы адаптивного веб-дизайна (RWD)

Принцип	Описание
Fluid Grids	Гибкие сетки на основе процентов, а не фиксированных пикселей
Flexible Images	Изображения масштабируются относительно контейнера
Media Queries	CSS-правила, применяемые при определённых размерах экрана
Mobile First	Сначала дизайн для мобильных, затем наращивание для больших экранов

2.2. Основные брейкпоинты

Устройство	Ширина экрана	Bootstrap класс	Tailwind класс
Мобильные (портрет)	< 576px	col-	max-sm:
Мобильные (ландшафт)	576px - 768px	col-sm-	sm:
Планшеты	768px - 992px	col-md-	md:
Десктопы	992px - 1200px	col-lg-	lg:
Большие десктопы	> 1200px	col-xl-	xl:

2.3. Установка фреймворков

Bootstrap:

```
npm install bootstrap @popperjs/core
```

Tailwind CSS:

bash

```
npm install -D tailwindcss postcss autoprefixer
npx tailwindcss init -p
```

3. ПРИМЕР ВЫПОЛНЕНИЯ (СКВОЗНОЙ ПРОЕКТ)

3.1. Вариант А: Адаптивная вёрстка с Bootstrap

3.1.1. Настройка Bootstrap

src/main.jsx:

```
import React from 'react';
import ReactDOM from 'react-dom/client';
import App from './App';
import 'bootstrap/dist/css/bootstrap.min.css';
import 'bootstrap/dist/js/bootstrap.bundle.min.js';
import './styles/global.css';

ReactDOM.createRoot(document.getElementById('root')).render(
  <React.StrictMode>
    <App />
  </React.StrictMode>
);
```

3.1.2. Адаптивная навигация (бургер-меню)

src/components/common/Navbar.jsx (Bootstrap):

```
import React from 'react';
import { Link, useNavigate } from 'react-router-dom';
import { Container, Navbar as BsNavbar, Nav, NavDropdown, Button }
from 'react-bootstrap';

const Navbar = () => {
  const navigate = useNavigate();
  const token = localStorage.getItem('token');
  const user = JSON.parse(localStorage.getItem('user') || '{}');

  const handleLogout = () => {
    localStorage.removeItem('token');
    localStorage.removeItem('user');
    navigate('/login');
  };
```

```
        return (
          <BsNavbar bg="dark" variant="dark" expand="lg" sticky="top" 
className="mb-4">
            <Container>
              <BsNavbar.Brand as={Link} to="/">
                🎉 EventManager
              </BsNavbar.Brand>
              <BsNavbar.Toggle aria-controls="basic-navbar-nav" />
              <BsNavbar.Collapse id="basic-navbar-nav">
                <Nav className="ms-auto">
                  <Nav.Link as={Link} to="/">Главная</Nav.Link>
                  <Nav.Link as={Link} 
to="/events">Мероприятия</Nav.Link>

                  {token ? (
                    <>
                      <Nav.Link as={Link} to="/my-registrations">
                        Мои регистрации
                      </Nav.Link>
                      {user.role === 'ADMIN' && (
                        <Nav.Link as={Link} to="/events/create">
                          + Создать
                        </Nav.Link>
                      )}
                      <NavDropdown title={user.name || 'Профиль'} 
id="user-nav-dropdown">
                        <NavDropdown.Item as={Link} to="/profile">
                          Профиль
```

```
                                    </NavDropdown.Item>
                                    <NavDropdown.Divider />
                                    <NavDropdown.Item onClick={handleLogout}>
                                      Выйти
                                    </NavDropdown.Item>
                                  </NavDropdown>
                                </>
                              ) : (
                                <>
                                  <Nav.Link as={Link} to="/login">Вход</Nav.Link>
                                  <Nav.Link                                    as={Link}
to="/register">Регистрация</Nav.Link>
                                </>
                              )}
                            </Nav>
                          </BsNavbar.Collapse>
                        </Container>
                      </BsNavbar>
                    );
                  };

                  export default Navbar;
```

3.1.3. Адаптивный компонент EventCard

src/components/events/EventCard.jsx (Bootstrap):

```
import React from 'react';
import { Card, Button, Badge, Row, Col } from 'react-bootstrap';
import { useNavigate } from 'react-router-dom';
import { formatDate } from '../../utils/dateFormatter';
```

```
import './EventCard.css';

const EventCard = ({ event, onDelete, isAdmin = false }) => {
    const navigate = useNavigate();

    const getStatusVariant = (status) => {
        const variants = {
            'DRAFT': 'secondary',
            'PUBLISHED': 'primary',
            'CANCELLED': 'danger',
            'COMPLETED': 'success'
        };
        return variants[status] || 'secondary';
    };

    const getStatusText = (status) => {
        const texts = {
            'DRAFT': 'Черновик',
            'PUBLISHED': 'Опубликовано',
            'CANCELLED': 'Отменено',
            'COMPLETED': 'Завершено'
        };
        return texts[status] || status;
    };

    const getProgressVariant = (ratio) => {
        if (ratio === 1) return 'danger';
        if (ratio > 0.8) return 'warning';
        return 'success';
```

```
    };

    const registeredCount = event registeredCount || 0;
    const progress = registeredCount / event maxParticipants;
    const progressPercent = Math round(progress * 100);

    return (
      <Card className="event-card h-100 shadow-sm">
        <Card.Body>
          <div  className="d-flex  justify-content-between  align-items-start
mb-2">
            <Card.Title className="mb-0">
              {event title}
            </Card.Title>
            <Badge bg={getStatusVariant(event.status)}>
              {getStatusText(event status)}
            </Badge>
          </div>

          <Card.Text className="text-muted small mb-3">
            {event description?.substring(0, 100)}
            {event description?.length > 100 ? '...' : ''}
          </Card.Text>

          <div className="event-details mb-3">
            <div className="detail-item mb-2">
              <span className="detail-icon">🗓</span>
              <span                                    className="detail-
text">{formatDate(event startDate)}</span>
```

```
                </div>

                <div className="detail-item mb-2">
                  <span className="detail-icon">👥</span>
                  <span className="detail-text">
                    {registeredCount} / {event.maxParticipants} участников
                  </span>
                </div>

                <div className="progress mb-2" style={{ height: '6px' }}>
                  <div
                    className={`progress-bar bg-${getProgressVariant(progress)}`}
                    style={{ width: `${progressPercent}%` }}
                  />
                </div>

                {event.basePrice > 0 && (
                  <div className="detail-item">
                    <span className="detail-icon">💰</span>
                    <span className="detail-text fw-bold">
                      {event.basePrice.toLocaleString()} ₽
                    </span>
                  </div>
                )}
              </div>

              <div className="d-flex gap-2 flex-wrap">
```

```
<Button
    variant="outline-primary"
    size="sm"
    onClick={() => navigate(`/events/${event id}`)}
>
    Подробнее
</Button>

{isAdmin && (
    <>
        <Button
            variant="outline-warning"
            size="sm"
            onClick={() => navigate(`/events/${event id}/edit`)}
        >
            Редактировать
        </Button>
        <Button
            variant="outline-danger"
            size="sm"
            onClick={() => onDelete(event id)}
        >
            Удалить
        </Button>
    </>
)}

{!isAdmin && event status === 'PUBLISHED' && (
    <Button
```

```
                        variant="success"
                        size="sm"
                        onClick={() => navigate(`/events/${event.id}/register`)}
                    >
                        Зарегистрироваться
                    </Button>
                )}
            </div>
        </Card.Body>

        <Card.Footer className="text-muted small bg-white">
            Создано: {formatDate(event.createdAt)}
        </Card.Footer>
    </Card>
  );
};

export default EventCard;
```

3.1.4. Адаптивный список мероприятий

src/pages/EventsPage.jsx (Bootstrap):

```
import React, { useState, useEffect } from 'react';
import { Container, Row, Col, Button, Spinner, Alert, Form, InputGroup } from 'react-bootstrap';
import { eventApi } from '../api/eventApi';
import EventCard from '../components/events/EventCard';
import Pagination from '../components/common/Pagination';
import { showSuccess, showError } from '../utils/toastUtils';

const EventsPage = () => {
```

```
const [events, setEvents] = useState([]);
const [loading, setLoading] = useState(true);
const [error, setError] = useState(null);
const [filters, setFilters] = useState({ search: '', status: '' });
const [pagination, setPagination] = useState({
    pageNumber: 0,
    pageSize: 9,
    totalPages: 0
});

const user = JSON.parse(localStorage.getItem('user') || '{}');
const isAdmin = user.role === 'ADMIN';

const loadEvents = async (page = 0) => {
    setLoading(true);
    setError(null);

    try {
        const response = await eventApi.getEvents(page, 9, filters);
        setEvents(response.data.content);
        setPagination({
            pageNumber: response.data.pageNumber,
            pageSize: response.data.pageSize,
            totalPages: response.data.totalPages
        });
    } catch (err) {
        setError(err.response?.data?.message || 'Ошибка загрузки');
    } finally {
        setLoading(false);
```

```
    }
};

const handleDelete = async (id) => {
    if (!window.confirm('Удалить мероприятие?')) return;
    try {
        await eventApi.deleteEvent(id);
        showSuccess('Мероприятие удалено');
        loadEvents(pagination.pageNumber);
    } catch (err) {
        showError(err.response?.data?.message || 'Ошибка удаления');
    }
};

const handleFilterChange = (e) => {
    setFilters({ ...filters, [e.target.name]: e.target.value });
};

const applyFilters = () => {
    loadEvents(0);
};

useEffect(() => {
    loadEvents();
}, []);

if (loading && events.length === 0) {
    return (
        <Container className="text-center py-5">
```

```
            <Spinner animation="border" variant="primary" />
            <p className="mt-3">Загрузка мероприятий...</p>
        </Container>
    );
}

if (error) {
    return (
        <Container className="text-center py-5">
            <Alert variant="danger">{error}</Alert>
            <Button variant="primary" onClick={() => loadEvents()}>
                Повторить
            </Button>
        </Container>
    );
}

return (
    <Container className="py-4">
        <div  className="d-flex  flex-column  flex-sm-row  justify-content-
between align-items-sm-center mb-4 gap-3">
            <h1 className="mb-0">Мероприятия</h1>
            {isAdmin && (
                <Button variant="primary" href="/events/create">
                    + Создать мероприятие
                </Button>
            )}
        </div>
```

```
{/* Фильтры */}
<Row className="mb-4 g-3">
  <Col xs={12} md={6}>
    <InputGroup>
      <Form.Control
        placeholder="Поиск по названию..."
        name="search"
        value={filters.search}
        onChange={handleFilterChange}
        onKeyPress={(e) => e.key === 'Enter' && applyFilters()}
      />
      <Button variant="outline-secondary" onClick={applyFilters}>
        🔍 Поиск
      </Button>
    </InputGroup>
  </Col>
  <Col xs={12} md={4}>
    <Form.Select
      name="status"
      value={filters.status}
      onChange={handleFilterChange}
    >
      <option value="">Все статусы</option>
      <option value="PUBLISHED">Опубликованные</option>
      <option value="DRAFT">Черновики</option>
      <option value="COMPLETED">Завершённые</option>
    </Form.Select>
  </Col>
```

```
                    <Col xs={12} md={2}>
                      <Button variant="secondary" onClick={applyFilters}
classNameName="w-100">
                        Применить
                      </Button>
                    </Col>
                  </Row>

                  {/* Список мероприятий */}
                  {events.length === 0 ? (
                    <div className="text-center py-5 bg-light rounded">
                      <p className="text-muted">Мероприятия не найдены</p>
                      {isAdmin && (
                        <Button variant="primary" href="/events/create">
                          Создать первое мероприятие
                        </Button>
                      )}
                    </div>
                  ) : (
                    <>
                      <Row xs={1} md={2} lg={3} className="g-4">
                        {events.map(event => (
                          <Col key={event.id}>
                            <EventCard
                              event={event}
                              onDelete={handleDelete}
                              isAdmin={isAdmin}
                            />
                          </Col>
```

```
                    ))}
                </Row>

                {pagination.totalPages > 1 && (
                    <div className="d-flex justify-content-center mt-4">
                        <Pagination
                            currentPage={pagination.pageNumber}
                            totalPages={pagination.totalPages}
                            onPageChange={loadEvents}
                        />
                    </div>
                )}
            </>
        )}
    </Container>
  );
};

export default EventsPage;
```

3.1.5. Адаптивная форма

src/pages/CreateEventPage.jsx (Bootstrap):

```
import React, { useState, useEffect } from 'react';
import { useNavigate } from 'react-router-dom';
import { Container, Row, Col, Form, Button, Spinner, Alert } from 'react-bootstrap';
import { eventApi } from '../api/eventApi';
import { categoryApi } from '../api/categoryApi';
import { showSuccess, showError } from '../utils/toastUtils';
```

```
const CreateEventPage = () => {
    const navigate = useNavigate();
    const [isLoading, setIsLoading] = useState(false);
    const [categories, setCategories] = useState([]);
    const [error, setError] = useState(null);
    const [formData, setFormData] = useState({
        title: '',
        description: '',
        startDate: '',
        endDate: '',
        maxParticipants: 1,
        basePrice: 0,
        categoryId: ''
    });

    useEffect(() => {
        const loadCategories = async () => {
            try {
                const response = await categoryApi.getCategories();
                setCategories(response.data);
            } catch (err) {
                console.error(err);
            }
        };
        loadCategories();
    }, []);

    const handleChange = (e) => {
        const { name, value, type } = e.target;
```

```
    setFormData({
      ...formData,
      [name]: type === 'number' ? Number(value) : value
    });
  };

  const handleSubmit = async (e) => {
    e.preventDefault();
    setIsLoading(true);
    setError(null);

    try {
      await eventApi.createEvent(formData);
      showSuccess('Мероприятие создано');
      navigate('/events');
    } catch (err) {
      setError(err.response?.data?.message || 'Ошибка создания');
      showError(err.response?.data?.message || 'Ошибка создания');
    } finally {
      setIsLoading(false);
    }
  };

  return (
    <Container className="py-4">
      <Row className="justify-content-center">
        <Col xs={12} md={8} lg={6}>
          <div className="bg-white p-4 p-md-5 rounded-4 shadow-sm">
            <h2 className="mb-4">Создание мероприятия</h2>
```

```
{error && <Alert variant="danger">{error}</Alert>}

<Form onSubmit={handleSubmit}>
  <Form.Group className="mb-3">
    <Form.Label>Название *</Form.Label>
    <Form.Control
      type="text"
      name="title"
      value={formData.title}
      onChange={handleChange}
      required
      placeholder="Введите название"
    />
  </Form.Group>

  <Form.Group className="mb-3">
    <Form.Label>Описание</Form.Label>
    <Form.Control
      as="textarea"
      name="description"
      rows={4}
      value={formData.description}
      onChange={handleChange}
      placeholder="Введите описание"
    />
  </Form.Group>

  <Row className="g-3">
```

```
                <Col xs={12} md={6}>
                    <Form.Group>
                        <Form.Label>Дата начала *</Form.Label>
                        <Form.Control
                            type="datetime-local"
                            name="startDate"
                            value={formData startDate}
                            onChange={handleChange}
                            required
                        />
                    </Form.Group>
                </Col>
                <Col xs={12} md={6}>
                    <Form.Group>
                        <Form.Label>Дата окончания *</Form.Label>
                        <Form.Control
                            type="datetime-local"
                            name="endDate"
                            value={formData endDate}
                            onChange={handleChange}
                            required
                        />
                    </Form.Group>
                </Col>
            </Row>

            <Row className="g-3 mt-2">
                <Col xs={12} md={6}>
                    <Form.Group>
```

```
                    <Form.Label>Максимум участников *</Form.Label>
                    <Form.Control
                      type="number"
                      name="maxParticipants"
                      value={formData.maxParticipants}
                      onChange={handleChange}
                      min="1"
                      max="10000"
                      required
                    />
                  </Form.Group>
                </Col>
                <Col xs={12} md={6}>
                  <Form.Group>
                    <Form.Label>Базовая цена (₽)</Form.Label>
                    <Form.Control
                      type="number"
                      name="basePrice"
                      value={formData.basePrice}
                      onChange={handleChange}
                      min="0"
                      step="0.01"
                    />
                  </Form.Group>
                </Col>
              </Row>

              <Form.Group className="mb-4">
```

```
        <Form.Label>Категория</Form.Label>
        <Form.Select
          name="categoryId"
          value={formData.categoryId}
          onChange={handleChange}
        >
          <option value="">Без категории</option>
          {categories.map(cat => (
            <option key={cat.id} value={cat.id}>
              {cat.name}
            </option>
          ))}
        </Form.Select>
      </Form.Group>

      <div className="d-flex gap-3">
        <Button
          type="submit"
          variant="primary"
          disabled={isLoading}
          className="flex-grow-1"
        >
          {isLoading ? 'Создание...' : 'Создать мероприятие'}
        </Button>
        <Button
          type="button"
          variant="secondary"
          onClick={() => navigate('/events')}
        >
```

```
                        Отмена
                    </Button>
                </div>
            </Form>
        </div>
    </Col>
</Row>
</Container>
  );
};

export default CreateEventPage;
```

3.2. Вариант Б: Адаптивная вёрстка с Tailwind CSS

3.2.1. Настройка Tailwind

tailwind.config.js:

```
/** @type {import('tailwindcss').Config} */
export default {
  content: [
    "./index.html",
    "./src/**/*.{js,ts,jsx,tsx}",
  ],
  theme: {
    extend: {
      colors: {
        primary: {
          50: '#eff6ff',
          500: '#3b82f6',
          600: '#2563eb',
```

```
                700: '#1d4ed8',
            }
        },
        screens: {
            'xs': '475px',
            'sm': '640px',
            'md': '768px',
            'lg': '1024px',
            'xl': '1280px',
            '2xl': '1536px',
        }
    },
  },
  plugins: [],
}
```

src/index.css (Tailwind):

```
css
@tailwind base;
@tailwind components;
@tailwind utilities;

@layer base {
  body {
    @apply bg-gray-100 text-gray-900;
  }
}

@layer components {
  .btn-primary {
```

```
    @apply bg-blue-600 hover:bg-blue-700 text-white font-medium py-2 px-4 rounded-lg transition duration-200;
  }

  .btn-secondary {
    @apply bg-gray-500 hover:bg-gray-600 text-white font-medium py-2 px-4 rounded-lg transition duration-200;
  }

  .btn-danger {
    @apply bg-red-600 hover:bg-red-700 text-white font-medium py-2 px-4 rounded-lg transition duration-200;
  }

  .btn-outline {
    @apply border border-blue-600 text-blue-600 hover:bg-blue-50 font-medium py-2 px-4 rounded-lg transition duration-200;
  }

  .card {
    @apply bg-white rounded-xl shadow-md hover:shadow-lg transition-shadow duration-300 overflow-hidden;
  }

  .form-input {
    @apply w-full px-4 py-2 border border-gray-300 rounded-lg focus:outline-none focus:ring-2 focus:ring-blue-500 focus:border-transparent;
  }
```

```
  .form-label {
    @apply block text-sm font-medium text-gray-700 mb-1;
  }
}
```

3.2.2. Адаптивная навигация (бургер-меню) — Tailwind

src/components/common/Navbar.jsx (Tailwind):

```
import React, { useState } from 'react';
import { Link, useNavigate } from 'react-router-dom';

const Navbar = () => {
  const navigate = useNavigate();
  const [isMenuOpen, setIsMenuOpen] = useState(false);
  const token = localStorage.getItem('token');
  const user = JSON.parse(localStorage.getItem('user') || '{}');

  const handleLogout = () => {
    localStorage.removeItem('token');
    localStorage.removeItem('user');
    navigate('/login');
  };

  const toggleMenu = () => {
    setIsMenuOpen(!isMenuOpen);
  };

  return (
    <nav className="bg-gray-900 text-white sticky top-0 z-50 shadow-
lg">
      <div className="container mx-auto px-4">
```

```
                <div className="flex justify-between items-center py-4">
                    {/* Logo */}
                    <Link to="/" className="text-2xl font-bold hover:text-blue-400 transition">
                        🎉 EventManager
                    </Link>

                    {/* Desktop Menu */}
                    <div className="hidden md:flex items-center space-x-6">
                        <NavLink to="/">Главная</NavLink>
                        <NavLink to="/events">Мероприятия</NavLink>

                        {token ? (
                            <>
                                <NavLink to="/my-registrations">Мои регистрации</NavLink>
                                {user.role === 'ADMIN' && (
                                    <NavLink to="/events/create">+ Создать</NavLink>
                                )}
                                <div className="relative group">
                                    <button className="flex items-center space-x-1 hover:text-blue-400">
                                        <span>{user.name || 'Профиль'}</span>
                                        <svg className="w-4 h-4" fill="none" stroke="currentColor" viewBox="0 0 24 24">
                                            <path strokeLinecap="round" strokeLinejoin="round" strokeWidth={2} d="M19 9l-7 7-7-7" />
                                        </svg>
                                    </button>
```

```
                                <div className="absolute right-0 mt-2 w-48 bg-white rounded-lg shadow-xl hidden group-hover:block">
                                    <Link to="/profile" className="block px-4 py-2 text-gray-800 hover:bg-gray-100">
                                        Профиль
                                    </Link>
                                    <button
                                        onClick={handleLogout}
                                        className="block w-full text-left px-4 py-2 text-red-600 hover:bg-gray-100"
                                    >
                                        Выйти
                                    </button>
                                </div>
                            </div>
                        </>
                    ) : (
                        <>
                            <NavLink to="/login">Вход</NavLink>
                            <NavLink to="/register">Регистрация</NavLink>
                        </>
                    )}
                </div>

                {/* Mobile Menu Button */}
                <button
                    onClick={toggleMenu}
                    className="md:hidden text-white focus:outline-none"
                >
```

```
                        <svg            className="w-6            h-6"            fill="none"
stroke="currentColor" viewBox="0 0 24 24">
                          {isMenuOpen ? (
                            <path    strokeLinecap="round"    strokeLinejoin="round"
strokeWidth={2} d="M6 18L18 6M6 6l12 12" />
                          ) : (
                            <path    strokeLinecap="round"    strokeLinejoin="round"
strokeWidth={2} d="M4 6h16M4 12h16M4 18h16" />
                          )}
                        </svg>
                      </button>
                    </div>

                    {/* Mobile Menu */}
                    {isMenuOpen && (
                      <div className="md:hidden pb-4 space-y-3">
                        <MobileNavLink                                                to="/"
onClick={toggleMenu}>Главная</MobileNavLink>
                        <MobileNavLink                                          to="/events"
onClick={toggleMenu}>Мероприятия</MobileNavLink>

                        {token ? (
                          <>
                            <MobileNavLink                    to="/my-registrations"
onClick={toggleMenu}>
                              Мои регистрации
                            </MobileNavLink>
                            {user role === 'ADMIN' && (
```

```
                        <MobileNavLink to="/events/create" onClick={toggleMenu}>
                          + Создать
                        </MobileNavLink>
                      )}
                      <MobileNavLink to="/profile" onClick={toggleMenu}>
                        Профиль
                      </MobileNavLink>
                      <button
                        onClick={() => { handleLogout(); toggleMenu(); }}
                        className="block w-full text-left text-red-400 hover:text-red-300 py-2"
                      >
                        Выйти
                      </button>
                    </>
                  ) : (
                    <>
                      <MobileNavLink to="/login" onClick={toggleMenu}>Вход</MobileNavLink>
                      <MobileNavLink to="/register" onClick={toggleMenu}>Регистрация</MobileNavLink>
                    </>
                  )}
                </div>
              )}
            </div>
          </nav>
        );
```

```
};

const NavLink = ({ to, children }) => (
  <Link to={to} className="hover:text-blue-400 transition">
    {children}
  </Link>
);

const MobileNavLink = ({ to, onClick, children }) => (
  <Link
    to={to}
    onClick={onClick}
    className="block hover:text-blue-400 transition py-1"
  >
    {children}
  </Link>
);

export default Navbar;
```

3.2.3. Адаптивный компонент EventCard (Tailwind)

src/components/events/EventCard.jsx (Tailwind):

```
import React from 'react';
import { useNavigate } from 'react-router-dom';
import { formatDate } from '../../utils/dateFormatter';

const EventCard = ({ event, onDelete, isAdmin = false }) => {
  const navigate = useNavigate();
```

```
const getStatusClasses = (status) => {
    const classes = {
        'DRAFT': 'bg-gray-500',
        'PUBLISHED': 'bg-blue-600',
        'CANCELLED': 'bg-red-600',
        'COMPLETED': 'bg-green-600'
    };
    return classes[status] || 'bg-gray-500';
};

const getStatusText = (status) => {
    const texts = {
        'DRAFT': 'Черновик',
        'PUBLISHED': 'Опубликовано',
        'CANCELLED': 'Отменено',
        'COMPLETED': 'Завершено'
    };
    return texts[status] || status;
};

const getProgressBarColor = (ratio) => {
    if (ratio === 1) return 'bg-red-500';
    if (ratio > 0.8) return 'bg-yellow-500';
    return 'bg-green-500';
};

const registeredCount = event registeredCount || 0;
const progress = registeredCount / event maxParticipants;
const progressPercent = Math round(progress * 100);
```

```
        return (
            <div className="card h-full flex flex-col">
                <div className="p-5 flex-grow">
                    {/* Header */}
                    <div className="flex justify-between items-start mb-3">
                        <h3 className="text-xl font-semibold text-gray-800 line-clamp-2">
                            {event title}
                        </h3>
                        <span className={`${getStatusClasses(event.status)} text-white text-xs px-2 py-1 rounded-full whitespace-nowrap ml-2`}>
                            {getStatusText(event status)}
                        </span>
                    </div>

                    {/* Description */}
                    <p className="text-gray-600 text-sm mb-4 line-clamp-3">
                        {event description?.substring(0, 120)}
                        {event description?.length > 120 ? '...' : ''}
                    </p>

                    {/* Details */}
                    <div className="space-y-2 text-sm">
                        <div className="flex items-center text-gray-600">
                            <span className="mr-2">🗓</span>
                            <span>{formatDate(event startDate)}</span>
                        </div>
```

```
                    <div className="flex items-center text-gray-600">
                      <span className="mr-2">👥</span>
                      <span>{registeredCount} / {event.maxParticipants} участников</span>
                    </div>

                    {/* Progress Bar */}
                    <div className="w-full bg-gray-200 rounded-full h-2">
                      <div
                        className={`${getProgressBarColor(progress)} h-2 rounded-full transition-all duration-300`}
                        style={{ width: `${progressPercent}%` }}
                      />
                    </div>

                    {event.basePrice > 0 && (
                      <div className="flex items-center text-gray-800 font-semibold">
                        <span className="mr-2">💰</span>
                        <span>{event.basePrice.toLocaleString()} ₽</span>
                      </div>
                    )}
                  </div>
                </div>

                {/* Actions */}
                <div className="px-5 pb-5 pt-2 border-t border-gray-100">
                  <div className="flex flex-wrap gap-2">
```

```
                    <button
                      onClick={() => navigate(`/events/${event.id}`)}
                      className="flex-1 bg-blue-600 hover:bg-blue-700 text-white text-sm py-2 px-3 rounded-lg transition"
                    >
                      Подробнее
                    </button>

                    {isAdmin && (
                      <>
                        <button
                          onClick={() => navigate(`/events/${event.id}/edit`)}
                          className="flex-1 bg-yellow-600 hover:bg-yellow-700 text-white text-sm py-2 px-3 rounded-lg transition"
                        >
                          Изменить
                        </button>
                        <button
                          onClick={() => onDelete(event.id)}
                          className="flex-1 bg-red-600 hover:bg-red-700 text-white text-sm py-2 px-3 rounded-lg transition"
                        >
                          Удалить
                        </button>
                      </>
                    )}

                    {!isAdmin && event.status === 'PUBLISHED' && (
                      <button
```

```
                        onClick={() => navigate(`/events/${event.id}/register`)}
                        className="flex-1 bg-green-600 hover:bg-green-700 text-
white text-sm py-2 px-3 rounded-lg transition"
                      >
                        Зарегистрироваться
                      </button>
                    )}
                  </div>
                </div>
              </div>
            );
          };

          export default EventCard;
```

3.2.4. Адаптивный список (Tailwind)

src/pages/EventsPage.jsx (Tailwind):

```
import React, { useState, useEffect } from 'react';
import { eventApi } from '../api/eventApi';
import EventCard from '../components/events/EventCard';
import { showSuccess, showError } from '../utils/toastUtils';

const EventsPage = () => {
  const [events, setEvents] = useState([]);
  const [loading, setLoading] = useState(true);
  const [error, setError] = useState(null);
  const [filters, setFilters] = useState({ search: '', status: '' });
  const [pagination, setPagination] = useState({
    pageNumber: 0,
```

```
    pageSize: 9,
    totalPages: 0
});

const user = JSON.parse(localStorage.getItem('user') || '{}');
const isAdmin = user.role === 'ADMIN';

const loadEvents = async (page = 0) => {
    setLoading(true);
    try {
        const response = await eventApi.getEvents(page, 9, filters);
        setEvents(response.data.content);
        setPagination({
            pageNumber: response.data.pageNumber,
            pageSize: response.data.pageSize,
            totalPages: response.data.totalPages
        });
    } catch (err) {
        setError(err.response?.data?.message || 'Ошибка загрузки');
    } finally {
        setLoading(false);
    }
};

const handleDelete = async (id) => {
    if (!window.confirm('Удалить мероприятие?')) return;
    try {
        await eventApi.deleteEvent(id);
        showSuccess('Мероприятие удалено');
```

```
            loadEvents(pagination.pageNumber);
        } catch (err) {
            showError(err.response?.data?.message || 'Ошибка удаления');
        }
    };

    useEffect(() => {
        loadEvents();
    }, []);

    if (loading && events.length === 0) {
        return (
            <div className="flex justify-center items-center py-20">
                <div className="animate-spin rounded-full h-12 w-12 border-b-2 border-blue-600"></div>
            </div>
        );
    }

    if (error) {
        return (
            <div className="container mx-auto px-4 py-10 text-center">
                <div  className="bg-red-100  border  border-red-400  text-red-700 px-4 py-3 rounded mb-4">
                    {error}
                </div>
                <button onClick={() => loadEvents()} className="btn-primary">
                    Повторить
                </button>
```

```
        </div>
      );
    }

    return (
      <div className="container mx-auto px-4 py-8">
        {/* Header */}
        <div  className="flex  flex-col  sm:flex-row  justify-between  items-
center mb-6 gap-4">
          <h1          className="text-3xl          font-bold          text-gray-
800">Мероприятия</h1>
          {isAdmin && (
            <a href="/events/create" className="btn-primary">
              + Создать мероприятие
            </a>
          )}
        </div>

        {/* Filters */}
        <div className="flex flex-col md:flex-row gap-3 mb-6">
          <input
            type="text"
            placeholder="Поиск по названию..."
            className="form-input flex-1"
            value={filters.search}
            onChange={(e)       =>       setFilters({       ...filters,       search:
e.target.value })}
            onKeyPress={(e) => e.key === 'Enter' && loadEvents(0)}
          />
```

```
                    <select
                        className="form-input w-full md:w-48"
                        value={filters status}
                        onChange={(e) => setFilters({ ...filters, status: e target value })}
                    >
                        <option value="">Все статусы</option>
                        <option value="PUBLISHED">Опубликованные</option>
                        <option value="DRAFT">Черновики</option>
                        <option value="COMPLETED">Завершённые</option>
                    </select>
                    <button onClick={() => loadEvents(0)} className="btn-primary whitespace-nowrap">
                        Применить фильтры
                    </button>
                </div>

                {/* Events Grid */}
                {events length === 0 ? (
                    <div className="text-center py-12 bg-gray-50 rounded-xl">
                        <p className="text-gray-500">Мероприятия не найдены</p>
                        {isAdmin && (
                            <a href="/events/create" className="btn-primary inline-block mt-4">
                                Создать первое мероприятие
                            </a>
                        )}
                    </div>
                ) : (
                    <>
```

```
                    <div className="grid grid-cols-1 md:grid-cols-2 lg:grid-cols-3
gap-6">
                        {events.map(event => (
                            <EventCard
                                key={event.id}
                                event={event}
                                onDelete={handleDelete}
                                isAdmin={isAdmin}
                            />
                        ))}
                    </div>

                    {/* Pagination */}
                    {pagination.totalPages > 1 && (
                        <div className="flex justify-center mt-8 space-x-2">
                            <button
                                onClick={() => loadEvents(pagination.pageNumber -
1)}
                                disabled={pagination.pageNumber === 0}
                                className="px-4 py-2 border rounded-lg
disabled:opacity-50 disabled:cursor-not-allowed hover:bg-gray-50"
                            >
                                ← Назад
                            </button>
                            <span className="px-4 py-2 text-gray-600">
                                {pagination.pageNumber + 1} / {pagination.totalPages}
                            </span>
                            <button
```

```
                        onClick={() => loadEvents(pagination.pageNumber +
1)}
                        disabled={pagination.pageNumber + 1 >=
pagination.totalPages}
                        className="px-4 py-2 border rounded-lg
disabled:opacity-50 disabled:cursor-not-allowed hover:bg-gray-50"
                    >
                        Вперёд →
                    </button>
                </div>
            )}
        </>
    )}
</div>
  );
};

export default EventsPage;
```

4. МЕТОДИКА ВЫПОЛНЕНИЯ

Шаг 1: Выбор и установка фреймворка

Bootstrap:

npm install bootstrap @popperjs/core

Импорт CSS в main.jsx

Tailwind CSS:

npm install -D tailwindcss postcss autoprefixer

npx tailwindcss init -p

Настройка content в tailwind.config.js

Шаг 2: Адаптивная навигация

Создать компонент навигации с бургер-меню

Реализовать скрытие/показ меню на мобильных

Адаптировать под выбранный фреймворк

Шаг 3: Адаптация EventCard

Использовать grid/flex для расположения

Добавить адаптивные отступы

Оптимизировать для мобильных

Шаг 4: Адаптация форм

Сделать поля на всю ширину на мобильных

Использовать стековую компоновку

Адаптировать кнопки

Шаг 5: Тестирование

Проверить на мобильном эмуляторе

Проверить на разных разрешениях

5. КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Что такое адаптивная вёрстка и чем она отличается от отзывчивой?
2. Что такое брейкпоинты? Назовите основные.
3. Как работает система сеток Bootstrap/Tailwind?
4. Как реализовать бургер-меню для мобильной версии?
5. В чём преимущества подхода Mobile First?
6. Как сделать изображение адаптивным?

6. ТРЕБОВАНИЯ К ОТЧЕТУ

Структура отчета:

Титульный лист

Содержание

Введение

Выбранный фреймворк (обоснование)

Адаптивная навигация (код + скриншоты)

Адаптация компонентов (код + скриншоты)

Скриншоты на разных разрешениях

Заключение

Имя файла: Фамилия_Имя_ЛР24_ПИ.pdf

7. КРИТЕРИИ ОЦЕНИВАНИЯ

Критерий	Мак с. балл	Описание
Выбор и настройка фреймворка	3	Правильная установка, конфигурация
Адаптивная навигация	6	Бургер-меню, корректная работа на всех разрешениях
Адаптация компонентов	6	EventCard, формы, список
Использование сетки	4	Grid/Flex, брейкпоинты
Медиа-запросы	3	Дополнительная настройка

Критерий	Макс. балл	Описание
Оформление отчета	3	Соответствие требованиям
ИТОГО	25	

Лабораторная работа №25

Обработка ошибок и загрузки

1. Цель и задачи работы

Цель: Освоить методики обработки ошибок и управления состояниями загрузки в React-приложении для создания надёжного, отказоустойчивого и удобного для пользователя интерфейса.

Задачи:

1. Изучить стратегии обработки ошибок в React
2. Реализовать глобальную обработку ошибок через Error Boundary
3. Создать компоненты для отображения состояний загрузки (skeleton screens, spinners)
4. Реализовать механизмы повторных запросов (retry logic)
5. Обрабатывать различные HTTP-статусы ошибок (400, 401, 403, 404, 500)
6. Создать кастомные хуки для управления состояниями загрузки и ошибок

2. ТЕОРЕТИЧЕСКОЕ ОБОСНОВАНИЕ

2.1. Типы ошибок в React-приложении

Тип ошибки	Описание	Пример
Синтаксические ошибки	Ошибки компиляции	Некорректный JSX
Runtime ошибки	Ошибки во время выполнения	Обращение к undefined
Сетевые ошибки	Проблемы с API	Нет соединения, таймаут
Бизнес-ошибки	Ошибки логики приложения	Недостаточно средств
HTTP-ошибки	Ответы сервера с ошибками	400, 401, 403, 404, 500

2.2. Компонент Error Boundary

Error Boundary - это React-компонент, который перехватывает JavaScript-ошибки в дочерних компонентах и отображает fallback-UI вместо упавшего компонента.

```
class ErrorBoundary extends React Component {
  constructor(props) {
    super(props);
    this state = { hasError: false, error: null };
  }
```

```
  static getDerivedStateFromError(error) {
    return { hasError: true, error };
  }

  componentDidCatch(error, errorInfo) {
    console.error('Error caught:', error, errorInfo);
    // Логирование ошибки в сервис аналитики
  }

  render() {
    if (this.state.hasError) {
      return <ErrorFallback error={this.state.error} />;
    }
    return this.props.children;
  }
}
```

2.3. Стратегии обработки загрузки

Стратегия	Описание	Когда использовать
Spinner	Анимированный индикатор загрузки	Быстрые операции (< 1 сек)
Skeleton Screen	Скелет контента	Длительные загрузки, списки

Стратегия	Описание	Когда использовать
Progress Bar	Индикатор прогресса	Загрузка файлов, длительные процессы
Debounced Loading	Задержка показа индикатора	Предотвращение мерцания

2.4. HTTP-статусы и их обработка

Статус	Описание	Действие
200	OK	Обработать данные
201	Created	Показать успех, редирект
400	Bad Request	Показать ошибку валидации
401	Unauthorized	Редирект на логин
403	Forbidden	Показать сообщение о доступе

Статус	Описание	Действие
404	Not Found	Показать страницу 404
409	Conflict	Показать конфликт (дубликат)
500	Internal Error	Показать сообщение об ошибке сервера

3. ПРИМЕР ВЫПОЛНЕНИЯ (СКВОЗНОЙ ПРОЕКТ)

3.1. Компонент Error Boundary

src/components/common/ErrorBoundary.jsx:

```
import React from 'react';
import { Button, Container, Row, Col } from 'react-bootstrap';

class ErrorBoundary extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      hasError: false,
      error: null,
      errorInfo: null
    };
  }
```

```
static getDerivedStateFromError(error) {
    return { hasError: true, error };
}

componentDidCatch(error, errorInfo) {
    console.error('Error caught by boundary:', error, errorInfo);
    this.setState({ errorInfo });

    // Логирование ошибки в сервис аналитики
    if (this.props.onError) {
        this.props.onError(error, errorInfo);
    }
}

handleReset = () => {
    this.setState({ hasError: false, error: null, errorInfo: null });
    if (this.props.onReset) {
        this.props.onReset();
    }
};

render() {
    if (this.state.hasError) {
        return (
            <Container className="py-5">
                <Row className="justify-content-center">
                    <Col xs={12} md={8} lg={6}>
                        <div className="text-center">
                            <div className="mb-4">
```

```
            <span className="display-1">⚠</span>
          </div>
          <h1 className="h3 mb-3">Что-то пошло не так</h1>
          <p className="text-muted mb-4">
            Произошла ошибка при загрузке страницы.
            Попробуйте перезагрузить страницу или вернуться позже.
          </p>
          {process.env.NODE_ENV === 'development' && this.state.error && (
            <div className="alert alert-danger text-start mb-4">
              <strong>Ошибка:</strong> {this.state.error.toString()}
              {this.state.errorInfo && (
                <pre className="mt-2 mb-0 small">
                  {this.state.errorInfo.componentStack}
                </pre>
              )}
            </div>
          )}
          <div className="d-flex gap-3 justify-content-center">
            <Button variant="primary" onClick={this.handleReset}>
              Попробовать снова
            </Button>
            <Button variant="secondary" onClick={() => window.location.href = '/'}>
              На главную
            </Button>
```

```
                        </div>
                    </div>
                </Col>
            </Row>
        </Container>
      );
    }

    return this.props.children;
  }
}

export default ErrorBoundary;
```

3.2. Компоненты для отображения загрузки

src/components/common/Loader.jsx (Spinner):

```
import React from 'react';
import { Spinner } from 'react-bootstrap';

const Loader = ({ size = 'md', text = 'Загрузка...', fullPage = false }) => {
  const spinnerSize = {
    sm: { width: '1rem', height: '1rem' },
    md: { width: '2rem', height: '2rem' },
    lg: { width: '3rem', height: '3rem' }
  };

  const content = (
    <div className="text-center">
      <Spinner
```

```
        animation="border"
        variant="primary"
        style={spinnerSize[size]}
      />
      {text && <p className="mt-3 text-muted">{text}</p>}
    </div>
  );

  if (fullPage) {
    return (
      <div  className="d-flex  justify-content-center  align-items-center" style={{ minHeight: '50vh' }}>
        {content}
      </div>
    );
  }

  return content;
};

export default Loader;
```

src/components/common/SkeletonCard.jsx:

```
import React from 'react';
import { Card, Placeholder } from 'react-bootstrap';

const SkeletonCard = () => {
  return (
    <Card className="h-100 shadow-sm">
      <Card.Body>
```

```
            <Placeholder as={Card.Title} animation="glow">
              <Placeholder xs={8} />
            </Placeholder>
            <Placeholder as={Card.Text} animation="glow">
              <Placeholder xs={12} className="mb-2" />
              <Placeholder xs={10} className="mb-2" />
              <Placeholder xs={8} />
            </Placeholder>
            <Placeholder as="div" animation="glow">
              <Placeholder xs={4} className="mb-2" />
              <Placeholder xs={6} className="mb-2" />
              <Placeholder xs={8} />
            </Placeholder>
          </Card.Body>
          <Card.Footer className="bg-white">
            <Placeholder as="div" animation="glow">
              <Placeholder xs={6} />
            </Placeholder>
          </Card.Footer>
        </Card>
    );
};

export default SkeletonCard;
```

src/components/common/SkeletonList.jsx:

```
import React from 'react';
import { Row, Col } from 'react-bootstrap';
import SkeletonCard from './SkeletonCard';
```

```
const SkeletonList = ({ count = 6, cols = { xs: 1, md: 2, lg: 3 } }) => {
  return (
    <Row xs={cols.xs} md={cols.md} lg={cols.lg} className="g-4">
      {Array.from({ length: count }).map((_, index) => (
        <Col key={index}>
          <SkeletonCard />
        </Col>
      ))}
    </Row>
  );
};

export default SkeletonList;
```

3.3. Обработка HTTP-статусов в API-клиенте

src/api/axios.js (расширенный):

```
import axios from 'axios';
import { showError } from '../utils/toastUtils';

const api = axios.create({
  baseURL: import.meta.env.VITE_API_URL || 'http://localhost:8080/api',
  timeout: 10000,
  headers: {
    'Content-Type': 'application/json',
  },
});

// Request interceptor: добавление токена
api.interceptors.request.use(
```

```
  (config) => {
    const accessToken = localStorage getItem('accessToken');
    if (accessToken) {
      config headers Authorization = `Bearer ${accessToken}`;
    }
    return config;
  },
  (error) => Promise reject(error)
);

// Response interceptor: обработка ошибок
api interceptors response use(
  (response) => response,
  async (error) => {
    const originalRequest = error config;
    const status = error response?.status;
    const message = error response?.data?.message;

    // 401 Unauthorized — попытка обновить токен
    if (status === 401 && !originalRequest _retry) {
      originalRequest _retry = true;
      try {
        const refreshToken = localStorage getItem('refreshToken');
        if (!refreshToken) throw new Error('No refresh token');

        const response = await axios post(
          `${api defaults baseURL}/auth/refresh`,
          { refreshToken }
        );
```

```
                localStorage.setItem('accessToken', response.data.accessToken);
                localStorage.setItem('refreshToken', response.data.refreshToken);

                originalRequest.headers.Authorization = `Bearer ${response.data.accessToken}`;
                return api(originalRequest);
            } catch (refreshError) {
                localStorage.removeItem('accessToken');
                localStorage.removeItem('refreshToken');
                localStorage.removeItem('user');
                window.location.href = '/login';
                return Promise.reject(refreshError);
            }
        }

        // 403 Forbidden
        if (status === 403) {
            showError('У вас нет доступа к этому ресурсу');
            // Опционально: редирект на главную
            // window.location.href = '/';
        }

        // 404 Not Found
        if (status === 404 && !originalRequest._isNotFoundHandled) {
            // Не показываем toast для каждого 404, только если нужно
            if (originalRequest._showNotFound !== false) {
                showError(message || 'Ресурс не найден');
            }
```

```
    }

    // 409 Conflict
    if (status === 409) {
      showError(message || 'Конфликт данных');
    }

    // 500 Internal Server Error
    if (status >= 500) {
      showError('Ошибка сервера. Попробуйте позже.');
    }

    // 0 — нет соединения
    if (!status && error code === 'ECONNABORTED') {
      showError('Превышено время ожидания ответа от сервера');
    }

    if (!status && !error response) {
      showError('Нет соединения с сервером. Проверьте интернет.');
    }

    return Promise.reject(error);
  }
);

export default api;
```

3.4. Кастомные хуки для обработки загрузки и ошибок

src/hooks/useApiRequest.js:

```
import { useState, useCallback } from 'react';
import { showError } from '../utils/toastUtils';

/**
 * Хук для выполнения API-запросов с обработкой состояний загрузки и ошибок
 */
export const useApiRequest = (options = {}) => {
  const { showToast = true, retryCount = 3, retryDelay = 1000 } = options;
  const [loading, setLoading] = useState(false);
  const [error, setError] = useState(null);
  const [data, setData] = useState(null);

  const execute = useCallback(async (apiFunction, ...args) => {
    setLoading(true);
    setError(null);

    let lastError = null;

    for (let attempt = 1; attempt <= retryCount; attempt++) {
      try {
        const response = await apiFunction(...args);
        setData(response.data);
        setLoading(false);
        return { success: true, data: response.data };
      } catch (err) {
        lastError = err;
        const status = err.response?.status;
```

```
                // Не повторяем при ошибках клиента (4xx, кроме 429)
                if (status && status >= 400 && status < 500 && status !== 429) {
                    break;
                }

                // Задержка перед повтором
                if (attempt < retryCount) {
                    await new Promise(resolve => setTimeout(resolve, retryDelay *
attempt));
                }
            }
        }

        const errorMessage = lastError?.response?.data?.message ||
                             lastError?.message ||
                             'Произошла ошибка';

        setError(errorMessage);
        setLoading(false);

        if (showToast) {
            showError(errorMessage);
        }

        return { success: false, error: errorMessage };
    }, [retryCount, retryDelay, showToast]);

    const reset = useCallback(() => {
        setLoading(false);
```

```
        setError(null);
        setData(null);
      }, []);

      return { loading, error, data, execute, reset };
    };
```

src/hooks/useDebouncedLoading.js:

```
    import { useState, useEffect, useRef } from 'react';

    /**
     * Хук для отложенного отображения индикатора загрузки (предотвращает мерцание)
     */
    export const useDebouncedLoading = (isLoading, delay = 300) => {
      const [showLoader, setShowLoader] = useState(false);
      const timeoutRef = useRef(null);

      useEffect(() => {
        if (isLoading) {
          timeoutRef.current = setTimeout(() => {
            setShowLoader(true);
          }, delay);
        } else {
          if (timeoutRef.current) {
            clearTimeout(timeoutRef.current);
          }
          setShowLoader(false);
        }
```

```
        return () => {
            if (timeoutRef.current) {
                clearTimeout(timeoutRef.current);
            }
        };
    }, [isLoading, delay]);

    return showLoader;
};
```

3.5. Компонент страницы 404

src/pages/NotFoundPage.jsx:

jsx

```
import React from 'react';
import { Link } from 'react-router-dom';
import { Container, Row, Col, Button } from 'react-bootstrap';

const NotFoundPage = () => {
    return (
        <Container className="py-5">
            <Row className="justify-content-center text-center">
                <Col xs={12} md={8} lg={6}>
                    <div className="mb-4">
                        <span className="display-1">🔍</span>
                    </div>
                    <h1 className="display-4 mb-3">404</h1>
                    <h2 className="h4 mb-3">Страница не найдена</h2>
                    <p className="text-muted mb-4">
```

```
                Запрашиваемая страница не существует или была перемещена.
              </p>
              <div className="d-flex gap-3 justify-content-center">
                <Button as={Link} to="/" variant="primary">
                  На главную
                </Button>
                <Button as={Link} to="/events" variant="outline-secondary">
                  К мероприятиям
                </Button>
              </div>
            </Col>
          </Row>
        </Container>
      );
    };

    export default NotFoundPage;
```

3.6. Обновлённый список мероприятий с обработкой ошибок и загрузки

src/pages/EventsPage.jsx (полная версия):

```
    import React, { useState, useEffect, useCallback } from 'react';
    import { Container, Row, Col, Button, Alert, Form, InputGroup } from 'react-bootstrap';
    import { eventApi } from '../api/eventApi';
    import EventCard from '../components/events/EventCard';
    import SkeletonList from '../components/common/SkeletonList';
```

```
import Loader from '../components/common/Loader';
import { useApiRequest } from '../hooks/useApiRequest';
import { useDebouncedLoading } from '../hooks/useDebouncedLoading';
import { showSuccess, showError } from '../utils/toastUtils';

const EventsPage = () => {
  const [events, setEvents] = useState([]);
  const [filters, setFilters] = useState({ search: '', status: '' });
  const [pagination, setPagination] = useState({
    pageNumber: 0,
    pageSize: 9,
    totalPages: 0
  });

  const { loading, error, execute: loadEvents } = useApiRequest({
    showToast: false,
    retryCount: 2
  });

  const user = JSON.parse(localStorage.getItem('user') || '{}');
  const isAdmin = user.role === 'ADMIN';
  const showLoader = useDebouncedLoading(loading, 300);

  const fetchEvents = useCallback(async (page = 0) => {
    const result = await loadEvents(
      () => eventApi.getEvents(page, 9, filters),
      page, filters
    );
```

```
            if (result.success) {
                setEvents(result.data.content);
                setPagination({
                    pageNumber: result.data.pageNumber,
                    pageSize: result.data.pageSize,
                    totalPages: result.data.totalPages
                });
            }
        }, [loadEvents, filters]);

        const handleDelete = async (id) => {
            if (!window.confirm('Вы уверены, что хотите удалить это мероприятие?')) return;

            try {
                await eventApi.deleteEvent(id);
                showSuccess('Мероприятие удалено');
                fetchEvents(pagination.pageNumber);
            } catch (err) {
                const status = err.response?.status;
                if (status === 403) {
                    showError('У вас нет прав для удаления этого мероприятия');
                } else if (status === 404) {
                    showError('Мероприятие не найдено');
                } else {
                    showError('Ошибка при удалении мероприятия');
                }
            }
        };
```

```
  const handleRetry = () => {
    fetchEvents(pagination.pageNumber);
  };

  useEffect(() => {
    fetchEvents();
  }, []);

  // Отображение скелетона при первой загрузке
  if (loading && events.length === 0 && showLoader) {
    return (
      <Container className="py-4">
        <div    className="d-flex    justify-content-between    align-items-
center mb-4">
          <h1>Мероприятия</h1>
          {isAdmin && (
            <Button variant="primary" href="/events/create">
              + Создать
            </Button>
          )}
        </div>
        <SkeletonList count={6} />
      </Container>
    );
  }

  // Отображение ошибки
  if (error && events.length === 0) {
```

```
    return (
      <Container className="py-5">
        <Alert variant="danger" className="text-center">
          <Alert.Heading>Ошибка загрузки</Alert.Heading>
          <p>{error}</p>
          <hr />
          <div className="d-flex justify-content-center gap-3">
            <Button variant="primary" onClick={handleRetry}>
              Повторить загрузку
            </Button>
            <Button variant="secondary" onClick={() => window.location.href = '/'}>
              На главную
            </Button>
          </div>
        </Alert>
      </Container>
    );
  }

  return (
    <Container className="py-4">
      {/* Header */}
      <div className="d-flex flex-column flex-sm-row justify-content-between align-items-sm-center mb-4 gap-3">
        <h1 className="mb-0">Мероприятия</h1>
        {isAdmin && (
          <Button variant="primary" href="/events/create">
            + Создать мероприятие
```

```
                    </Button>
                  )}
                </div>

                {/* Filters */}
                <div className="bg-white p-3 rounded-3 shadow-sm mb-4">
                  <Row className="g-3">
                    <Col xs={12} md={6}>
                      <InputGroup>
                        <Form.Control
                          placeholder="Поиск по названию..."
                          value={filters.search}
                          onChange={(e) => setFilters({ ...filters, search: e.target.value })}
                          onKeyPress={(e) => e.key === 'Enter' && fetchEvents(0)}
                        />
                        <Button variant="outline-secondary" onClick={() => fetchEvents(0)}>
                          🔍
                        </Button>
                      </InputGroup>
                    </Col>
                    <Col xs={12} md={4}>
                      <Form.Select
                        value={filters.status}
                        onChange={(e) => setFilters({ ...filters, status: e.target.value })}
                      >
```

```
                        <option value="">Все статусы</option>
                        <option
value="PUBLISHED">Опубликованные</option>
                        <option value="DRAFT">Черновики</option>
                        <option value="COMPLETED">Завершённые</option>
                        <option value="CANCELLED">Отменённые</option>
                      </Form.Select>
                    </Col>
                    <Col xs={12} md={2}>
                      <Button variant="secondary" onClick={() => fetchEvents(0)}
className="w-100">
                        Применить
                      </Button>
                    </Col>
                  </Row>
                </div>

                {/* Events Grid */}
                {events length === 0 && !loading ? (
                  <div className="text-center py-5 bg-light rounded-3">
                    <p      className="text-muted      mb-3">Мероприятия      не
найдены</p>
                    {isAdmin && (
                      <Button variant="primary" href="/events/create">
                        Создать первое мероприятие
                      </Button>
                    )}
                  </div>
                ) : (
```

```
<>
  <Row xs={1} md={2} lg={3} className="g-4">
    {events.map(event => (
      <Col key={event.id}>
        <EventCard
          event={event}
          onDelete={handleDelete}
          isAdmin={isAdmin}
        />
      </Col>
    ))}
  </Row>

  {/* Loading overlay для пагинации */}
  {loading && events.length > 0 && (
    <div className="text-center mt-4">
      <Loader size="sm" text="Загрузка..." />
    </div>
  )}

  {/* Pagination */}
  {pagination.totalPages > 1 && !loading && (
    <div className="d-flex justify-content-center mt-4">
      <Pagination
        currentPage={pagination.pageNumber}
        totalPages={pagination.totalPages}
        onPageChange={fetchEvents}
      />
    </div>
```

```
                    )}
                </>
            )}
        </Container>
    );
};

// Компонент пагинации
const Pagination = ({ currentPage, totalPages, onPageChange }) => {
    const getPageNumbers = () => {
        const delta = 2;
        const range = [];
        for (let i = Math.max(0, currentPage - delta); i <= Math.min(totalPages - 1, currentPage + delta); i++) {
            range.push(i);
        }
        return range;
    };

    return (
        <div className="d-flex gap-2 align-items-center">
            <button
                className="btn btn-outline-secondary"
                onClick={() => onPageChange(currentPage - 1)}
                disabled={currentPage === 0}
            >
                «
            </button>
```

```
                {getPageNumbers().map(pageNum => (
                  <button
                    key={pageNum}
                    className={`btn ${pageNum === currentPage ? 'btn-primary' :
'btn-outline-secondary'}`}
                    onClick={() => onPageChange(pageNum)}
                  >
                    {pageNum + 1}
                  </button>
                ))}

                <button
                  className="btn btn-outline-secondary"
                  onClick={() => onPageChange(currentPage + 1)}
                  disabled={currentPage + 1 >= totalPages}
                >
                  »
                </button>
              </div>
            );
          };

          export default EventsPage;
```

3.7. Интеграция Error Boundary в App

src/App.jsx:

```
import React from 'react';
import { BrowserRouter as Router, Routes, Route } from 'react-router-dom';
import { Toaster } from 'react-hot-toast';
```

```
import ErrorBoundary from './components/common/ErrorBoundary';
import ErrorFallback from './components/common/ErrorFallback';
import Navbar from './components/common/Navbar';
import Footer from './components/common/Footer';
import PrivateRoute from './components/common/PrivateRoute';
import AdminRoute from './components/common/AdminRoute';
import HomePage from './pages/HomePage';
import EventsPage from './pages/EventsPage';
import EventDetailsPage from './pages/EventDetailsPage';
import CreateEventPage from './pages/CreateEventPage';
import EditEventPage from './pages/EditEventPage';
import LoginPage from './pages/LoginPage';
import RegisterPage from './pages/RegisterPage';
import ProfilePage from './pages/ProfilePage';
import MyRegistrationsPage from './pages/MyRegistrationsPage';
import NotFoundPage from './pages/NotFoundPage';
import './styles/global.css';

function App() {
  // Глобальный обработчик ошибок для Error Boundary
  const handleGlobalError = (error, errorInfo) => {
    console.error('Global error:', error, errorInfo);
    // Здесь можно отправить ошибку в сервис аналитики
    // analytics.logError(error, errorInfo);
  };

  const handleGlobalReset = () => {
    // Опциональные действия при сбросе ошибки
    console.log('Error boundary reset');
```

```
  };

  return (
    <ErrorBoundary onError={handleGlobalError} onReset={handleGlobalReset}>
      <Router>
        <Toaster
          position="top-right"
          toastOptions={{
            success: { duration: 3000 },
            error: { duration: 5000 },
            loading: { duration: Infinity }
          }}
        />
        <div className="app d-flex flex-column min-vh-100">
          <Navbar />
          <main className="flex-grow-1">
            <Routes>
              <Route path="/" element={<HomePage />} />
              <Route path="/events" element={<EventsPage />} />
              <Route path="/events/:id" element={<EventDetailsPage />} />
              <Route path="/login" element={<LoginPage />} />
              <Route path="/register" element={<RegisterPage />} />

              <Route element={<PrivateRoute />}>
                <Route path="/profile" element={<ProfilePage />} />
                <Route path="/my-registrations" element={<MyRegistrationsPage />} />
```

```
                        </Route>

                        <Route element={<AdminRoute />}>
                          <Route                                    path="/events/create"
element={<CreateEventPage />} />
                          <Route                                  path="/events/:id/edit"
element={<EditEventPage />} />
                        </Route>

                        <Route path="*" element={<NotFoundPage />} />
                      </Routes>
                    </main>
                    <Footer />
                  </div>
                </Router>
              </ErrorBoundary>
            );
          }

          export default App;
```

4. МЕТОДИКА ВЫПОЛНЕНИЯ

Шаг 1: Создание Error Boundary

Создать компонент ErrorBoundary

Реализовать getDerivedStateFromError и componentDidCatch

Создать fallback UI

Шаг 2: Компоненты загрузки

Создать Loader (spinner)

Создать SkeletonCard и SkeletonList

Шаг 3: Обработка HTTP-статусов

Расширить интерцепторы Axios

Обработать 401 (обновление токена)

Обработать 403, 404, 409, 500

Шаг 4: Кастомные хуки

Создать useApiRequest (загрузка + ошибки + retry)

Создать useDebouncedLoading

Шаг 5: Интеграция в компоненты

Обновить EventsPage

Обновить другие страницы

5. КОНТРОЛЬНЫЕ ВОПРОСЫ

Что такое Error Boundary и для чего он нужен?

Какие HTTP-статусы требуют особой обработки?

Чем skeleton screen отличается от spinner?

Как реализовать механизм повторных запросов (retry)?

Как предотвратить мерцание индикатора загрузки?

Как обработать ошибку 401 и обновить JWT-токен?

6. КРИТЕРИИ ОЦЕНИВАНИЯ

Критерий	Мак с. балл	Описание
Error Boundary	5	Корректная работа, fallback UI

Критерий	Макс. балл	Описание
Компоненты загрузки	5	Spinner, Skeleton
Обработка HTTP-статусов	6	401, 403, 404, 409, 500
Кастомные хуки	4	useApiRequest, useDebouncedLoading
Интеграция в компоненты	5	EventsPage и др.
Оформление отчета	2	Соответствие требованиям
ИТОГО	27	

Лабораторная работа №26

Сборка проекта в Docker (Dockerfile)

1. Цель и задачи работы

Цель: Освоить контейнеризацию Java-приложения (бэкенда) с использованием Docker для упрощения развёртывания, обеспечения воспроизводимости окружения и подготовки к промышленной эксплуатации.

Задачи:

1. Изучить основы Docker: образы, контейнеры, Dockerfile
2. Создать Dockerfile для Spring Boot приложения

3. Оптимизировать сборку с использованием multi-stage builds
4. Настроить переменные окружения для конфигурации
5. Создать docker-compose.yml для запуска приложения с PostgreSQL
6. Написать инструкцию по сборке и запуску

2. ТЕОРЕТИЧЕСКОЕ ОБОСНОВАНИЕ

2.1. Что такое Docker?

Docker - это платформа для разработки, доставки и запуска приложений в изолированных контейнерах.

Понятие	Описание
Образ (Image)	Шаблон для создания контейнеров (содержит приложение и зависимости)
Контейнер (Container)	Запущенный экземпляр образа
Dockerfile	Инструкции для сборки образа
Docker Hub	Реестр образов
docker-compose	Инструмент для запуска многоконтейнерных приложений

2.2. Преимущества Docker для курсового проекта

Преимущество	Описание
Воспроизводимост	Приложение работает одинаково на любом окруж

Преимущество	Описание
ь	
Изоляция	Нет конфликтов с другими приложениями
Простота развёртывания	Одна команда для запуска
Масштабирование	Легко увеличить количество экземпляров

2.3. Базовая структура Dockerfile

dockerfile

```
# Базовый образ
FROM openjdk:17-jdk-slim

# Информация о maintainer
LABEL maintainer="your.email@example.com"

# Рабочая директория
WORKDIR /app

# Копирование JAR-файла
COPY target/*.jar app.jar

# Открываем порт
EXPOSE 8080

# Команда запуска
ENTRYPOINT ["java", "-jar", "app.jar"]
```

2.4. Multi-stage build (многостадийная сборка)

dockerfile

```
# Стадия 1: сборка
FROM maven:3.9-openjdk-17 AS build
WORKDIR /app
COPY pom.xml .
RUN mvn dependency:go-offline
COPY src ./src
RUN mvn clean package -DskipTests

# Стадия 2: запуск
FROM openjdk:17-jdk-slim
WORKDIR /app
COPY --from=build /app/target/*.jar app.jar
EXPOSE 8080
ENTRYPOINT ["java", "-jar", "app.jar"]
```

2.5. docker-compose.yml

```
version: '3.8'

services:
  app:
    build: .
    ports:
      - "8080:8080"
    environment:
      - SPRING_DATASOURCE_URL=jdbc postgresql //db 5432/eventsdb
      - SPRING_DATASOURCE_USERNAME=postgres
```

```
      - SPRING_DATASOURCE_PASSWORD=password
    depends_on:
      - db
    networks:
      - app network

  db:
    image: postgres 15
    environment:
      - POSTGRES_DB=eventsdb
      - POSTGRES_USER=postgres
      - POSTGRES_PASSWORD=password
    ports:
      - "5432:5432"
    volumes:
      - postgres_data /var/lib/postgresql/data
    networks:
      - app network

networks:
  app-network:
    driver: bridge

volumes:
  postgres_data:
```

3. ПРИМЕР ВЫПОЛНЕНИЯ (СКВОЗНОЙ ПРОЕКТ)

3.1. Оптимизированный Dockerfile (multi-stage)

Dockerfile (backend):

dockerfile

```
#================ СТАДИЯ 1: СБОРКА ====================
FROM maven:3.9-openjdk-17-slim AS builder

# Установка рабочей директории
WORKDIR /app

# Копирование POM-файла и загрузка зависимостей (кэширование)
COPY pom.xml .
RUN mvn dependency:go-offline -B

# Копирование исходного кода и сборка
COPY src ./src
RUN mvn clean package -DskipTests -B

# ================ СТАДИЯ 2: ЗАПУСК ====================
FROM openjdk:17-jdk-slim

# Установка информации о метках
LABEL maintainer="student@university.ru"
LABEL description="Event Management System Backend"
LABEL version="1.0.0"

# Установка не-root пользователя для безопасности
RUN addgroup --system --gid 1001 appgroup && \
    adduser --system --uid 1001 --gid 1001 appuser

# Рабочая директория
WORKDIR /app
```

```
# Копирование JAR-файла из стадии сборки
COPY --from=builder /app/target/*.jar app.jar

# Копирование скрипта для ожидания БД (опционально)
COPY wait-for-it.sh /wait-for-it.sh
RUN chmod +x /wait-for-it.sh

# Смена пользователя
USER appuser

# Открытие порта
EXPOSE 8080

# Переменные окружения (могут быть переопределены)
ENV SPRING_PROFILES_ACTIVE=docker
ENV JAVA_OPTS="-Xmx512m -Xms256m"

# Точка входа
ENTRYPOINT ["sh", "-c", "java $JAVA_OPTS -jar app.jar"]
```

3.2. Скрипт ожидания базы данных

wait-for-it.sh:

bash

```
#!/bin/bash
# wait-for-it.sh - ожидание доступности TCP-хоста

set -e

host="$1"
```

```
shift
port="$1"
shift
cmd="$@"

until nc -z "$host" "$port"; do
  >&2 echo "Waiting for $host:$port..."
  sleep 2
done

>&2 echo "$host:$port is available - executing command"
exec $cmd
```

3.3. Файлы конфигурации для Docker-окружения

application-docker.yml:

```
spring:
  datasource:
    url:
${SPRING_DATASOURCE_URL:jdbc:postgresql://db:5432/eventsdb}
    username: ${SPRING_DATASOURCE_USERNAME:postgres}
    password: ${SPRING_DATASOURCE_PASSWORD:password}
    driver-class-name: org.postgresql.Driver
  jpa:
    hibernate:
      ddl-auto: validate
    show-sql: false
    properties:
      hibernate:
        dialect: org.hibernate.dialect.PostgreSQLDialect
```

```
    format_sql: true
  jackson:
    serialization:
      fail-on-empty-beans: false

logging:
  level:
    ru.edu.project: DEBUG
  pattern:
    console: "%d{ISO8601} %highlight(%-5level) [%thread] %logger{36} - %msg%n"

server:
  port: 8080
  error:
    include-message: always
    include-binding-errors: always

app:
  jwt:
    secret: ${JWT_SECRET:404E635266556A586E3272357538782F413F4428472B4B6250645367566B5970}
    expiration-ms: 3600000
    refresh-expiration-ms: 86400000
```

3.4. Docker Compose для полного стека

docker-compose.yml:

yaml

```yaml
version: '3.8'

services:
  # PostgreSQL база данных
  postgres:
    image: postgres:15-alpine
    container_name: event-management-postgres
    restart: unless-stopped
    environment:
      POSTGRES_DB: ${POSTGRES_DB:-eventsdb}
      POSTGRES_USER: ${POSTGRES_USER:-postgres}
      POSTGRES_PASSWORD: ${POSTGRES_PASSWORD:-postgres}
      PGDATA: /var/lib/postgresql/data/pgdata
    ports:
      - "5432:5432"
    volumes:
      - postgres_data:/var/lib/postgresql/data
      - ./init.sql:/docker-entrypoint-initdb.d/init.sql
    networks:
      - event-network
    healthcheck:
      test: ["CMD-SHELL", "pg_isready -U postgres"]
      interval: 10s
      timeout: 5s
      retries: 5

  # Backend приложение
  backend:
    build:
```

```
    context: .
    dockerfile: Dockerfile
  container_name: event-management-backend
  restart: unless-stopped
  environment:
    SPRING_PROFILES_ACTIVE: docker
    SPRING_DATASOURCE_URL:
jdbc:postgresql://postgres:5432/${POSTGRES_DB:-eventsdb}
    SPRING_DATASOURCE_USERNAME: ${POSTGRES_USER:-
postgres}
    SPRING_DATASOURCE_PASSWORD:
${POSTGRES_PASSWORD:-postgres}
    JWT_SECRET: ${JWT_SECRET:-
404E635266556A586E3272357538782F413F4428472B4B6250645367566B5970
}

  ports:
    - "8080:8080"
  depends_on:
    postgres:
      condition: service_healthy
  networks:
    - event-network
  healthcheck:
    test: ["CMD", "curl", "-f", "http://localhost:8080/actuator/health"]
    interval: 30s
    timeout: 10s
    retries: 3

# Опционально: Nginx для reverse proxy
```

```
  nginx:
    image: nginx:alpine
    container_name: event-management-nginx
    restart: unless-stopped
    ports:
      - "80:80"
      - "443:443"
    volumes:
      - ./nginx.conf:/etc/nginx/nginx.conf:ro
      - ./ssl:/etc/nginx/ssl:ro
    depends_on:
      - backend
    networks:
      - event-network
    profiles:
      - production

  # Опционально: pgAdmin для управления БД
  pgadmin:
    image: dpage/pgadmin4
    container_name: event-management-pgadmin
    restart: unless-stopped
    environment:
      PGADMIN_DEFAULT_EMAIL: ${PGADMIN_EMAIL:-admin@example.com}
      PGADMIN_DEFAULT_PASSWORD: ${PGADMIN_PASSWORD:-admin}
    ports:
      - "5050:80"
```

```
    depends_on:
      - postgres
    networks:
      - event-network
    profiles:
      - dev

networks:
  event-network:
    driver: bridge

volumes:
  postgres_data:
```

3.5. Nginx конфигурация (опционально)

nginx.conf:

```
nginx
events {
    worker_connections 1024;
}

http {
    upstream backend {
        server backend:8080;
    }

    server {
        listen 80;
        server_name localhost;
```

```
        location /api/ {
            proxy_pass http://backend;
            proxy_set_header Host $host;
            proxy_set_header X-Real-IP $remote_addr;
            proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
            proxy_set_header X-Forwarded-Proto $scheme;
        }

        location /swagger-ui/ {
            proxy_pass http://backend/swagger-ui/;
        }

        location /v3/api-docs/ {
            proxy_pass http://backend/v3/api-docs/;
        }

        location / {
            root /usr/share/nginx/html;
            try_files $uri /index.html;
        }
    }
}
```

3.6. Файл .env для переменных окружения

.env:

```
env
# Database
POSTGRES_DB=eventsdb
```

```
POSTGRES_USER=postgres
POSTGRES_PASSWORD=secure_password_here

# JWT
JWT_SECRET=your-256-bit-secret-key-for-jwt-here

# pgAdmin
PGADMIN_EMAIL=admin@example.com
PGADMIN_PASSWORD=admin123
```

3.7. Скрипты для управления Docker

scripts/docker-start.sh:

bash

```
#!/bin/bash

# Цвета для вывода
GREEN='\033[0;32m'
YELLOW='\033[1;33m'
NC='\033[0m'

echo -e "${YELLOW}Starting Event Management System with Docker...${NC}"

# Проверка наличия .env файла
if [ ! -f .env ]; then
    echo -e "${YELLOW}.env file not found. Creating from .env.example...${NC}"
    cp .env.example .env
fi
```

```
# Сборка и запуск
docker-compose up -d --build

# Проверка статуса
echo -e "${YELLOW}Checking container status...${NC}"
docker-compose ps

echo -e "${GREEN}Application started!${NC}"
echo -e "Backend: http://localhost:8080"
echo -e "Swagger UI: http://localhost:8080/swagger-ui.html"
echo -e "pgAdmin: http://localhost:5050 (email: admin@example.com, password: admin123)"

# Просмотр логов
echo -e "${YELLOW}Showing logs (Ctrl+C to exit)...${NC}"
docker-compose logs -f
```

scripts/docker-stop.sh:

```
bash
#!/bin/bash

YELLOW='\033[1;33m'
GREEN='\033[0;32m'
NC='\033[0m'

echo -e "${YELLOW}Stopping Event Management System...${NC}"
docker-compose down

echo -e "${GREEN}Application stopped!${NC}"
```

scripts/docker-clean.sh:

bash

```
#!/bin/bash

YELLOW='\033[1;33m'
RED='\033[0;31m'
NC='\033[0m'

echo -e "${RED}WARNING: This will remove all containers, volumes, and images!${NC}"
read -p "Are you sure? (y/n) " -n 1 -r
echo

if [[ $REPLY =~ ^[Yy]$ ]]; then
    echo -e "${YELLOW}Stopping containers...${NC}"
    docker-compose down -v

    echo -e "${YELLOW}Removing images...${NC}"
    docker rmi event-management-backend

    echo -e "${YELLOW}Cleaning up...${NC}"
    docker system prune -f

    echo -e "${GREEN}Cleanup complete!${NC}"
fi
```

3.8. Инструкция по сборке и запуску

README-docker.md:

markdown

```
# Docker-развертывание Event Management System
```

Требования

- Docker 20.10+
- Docker Compose 2.20+

Быстрый старт

1. Клонирование репозитория

```bash
git clone https://github.com/username/event-management.git
cd event-management
2. Настройка переменных окружения
bash
cp .env.example .env
# Отредактируйте .env при необходимости
3. Сборка и запуск
bash
# Дать права на выполнение скриптов
chmod +x scripts/*.sh

# Запуск
./scripts/docker-start.sh
4. Проверка работы
bash
# Статус контейнеров
docker-compose ps
```

```
# Логи
docker-compose logs -f backend

# Проверка API
curl http://localhost:8080/api/events
```

5. Остановка

```
bash
./scripts/docker-stop.sh
```

Доступные сервисы

Сервис	URL	Credentials
Backend API	http://localhost:8080	—
Swagger UI	http://localhost:8080/swagger-ui.html	—
pgAdmin	http://localhost:5050	admin@example.com / admin123

Команды Docker

```
bash
# Просмотр логов
docker-compose logs -f backend

# Перезапуск конкретного сервиса
docker-compose restart backend

# Вход в контейнер
```

```
docker exec -it event-management-backend /bin/sh

# Запуск только БД
docker-compose up -d postgres

# Остановка с удалением томов
docker-compose down -v
```

3.9. Файл .dockerignore

.dockerignore:

```gitignore
# Git
.git/
.gitignore

# IDE
.idea/
.vscode/
*.iml

# Logs
*.log
logs/

# Docker
docker-compose*.yml
Dockerfile*
```

.dockerignore

```
# Build
target/
build/
*.jar
!.mvn/wrapper/maven-wrapper.jar

# Tests
src/test/

# Documentation
docs/
README.md

# Environment
.env
.env.*
!.env.example

# Node (для фронтенда)
node_modules/
dist/
```

3.10. Альтернативный Dockerfile для фронтенда (React)

frontend/Dockerfile:

dockerfile

```
# ================ СТАДИЯ 1: СБОРКА ====================
FROM node:18-alpine AS builder
```

```
WORKDIR /app

# Копирование package.json и установка зависимостей
COPY package*.json ./
RUN npm ci --only=production

# Копирование исходного кода и сборка
COPY . .
RUN npm run build

# ================ СТАДИЯ 2: ЗАПУСК ====================
FROM nginx:alpine

# Копирование собранных файлов
COPY --from=builder /app/dist /usr/share/nginx/html

# Копирование конфигурации nginx
COPY nginx.conf /etc/nginx/nginx.conf

EXPOSE 80

CMD ["nginx", "-g", "daemon off;"]
```

4. МЕТОДИКА ВЫПОЛНЕНИЯ

Шаг 1: Установка Docker

Установить Docker Desktop на Windows/Mac

Установить Docker Engine на Linux

Проверить установку: docker –version

Шаг 2: Создание Dockerfile

Создать файл Dockerfile

Настроить multi-stage сборку

Добавить не-root пользователя

Шаг 3: Создание docker-compose.yml

Добавить сервис PostgreSQL

Добавить сервис backend

Настроить сети и тома

Добавить healthcheck

Шаг 4: Файлы конфигурации

Создать application-docker.yml

Создать .env файл

Создать .dockerignore

Шаг 5: Тестирование

Собрать образ: docker build -t event-app .

Запустить: docker-compose up -d

Проверить работоспособность API

5. КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Что такое Docker и чем контейнер отличается от виртуальной машины?
2. Для чего нужен Dockerfile?
3. Какая разница между COPY и ADD в Dockerfile?
4. Что такое multi-stage build и зачем он нужен?
5. Как передать переменные окружения в контейнер?

6. Как организовать ожидание запуска БД перед стартом приложения?

6. ТРЕБОВАНИЯ К ОТЧЕТУ

Структура отчета:

Титульный лист

Содержание

Введение

Dockerfile (листинг и пояснения)

docker-compose.yml (листинг и пояснения)

Конфигурационные файлы

Скрипты для управления

Скриншоты работы

Инструкция по запуску

Заключение

7. КРИТЕРИИ ОЦЕНИВАНИЯ

Критерий	Макс. балл	Описание
Dockerfile	6	Multi-stage, оптимизация, non-root user
docker-compose.yml	6	PostgreSQL, backend, сети, тома
Переменные окружения	3	.env, конфигурация

Критерий	Макс. балл	Описание
Документация	4	Инструкция, скрипты
Тестирование	3	Работоспособность
Оформление отчета	3	Соответствие требованиям
ИТОГО	25	

Лабораторная работа №27

Итоговая защита проекта

1. ЦЕЛЬ РАБОТЫ

Цель: Систематизировать и представить результаты полного цикла разработки программного проекта, выполненного в рамках дисциплины «Программная инженерия», продемонстрировать работоспособность разработанной системы, защитить итоговые результаты перед преподавателем.

Задачи:

1. Подготовить итоговый отчёт (пояснительную записку) по курсовому проекту
2. Подготовить презентацию, отражающую все этапы разработки
3. Продемонстрировать работающую систему (бэкенд + фронтенд)
4. Показать результаты тестирования и покрытие кода
5. Представить Docker-контейнеризацию

6. Ответить на вопросы по теоретическому и практическому материалу
7. Получить итоговую оценку

2. ТРЕБОВАНИЯ К ИТОГОВОМУ ОТЧЁТУ (ПОЯСНИТЕЛЬНАЯ ЗАПИСКА)

2.1. Структура пояснительной записки

1. ТИТУЛЬНЫЙ ЛИСТ
2. РЕФЕРАТ
3. СОДЕРЖАНИЕ
4. ВВЕДЕНИЕ
   - 4.1. Актуальность темы
   - 4.2. Цель и задачи курсового проекта
   - 4.3. Объект и предмет исследования
   - 4.4. Методы исследования

5. АНАЛИТИЧЕСКАЯ ЧАСТЬ (по ЛР1)
   - 5.1. Описание предметной области
   - 5.2. Анализ бизнес-процессов (IDEF0)
   - 5.3. SWOT-анализ текущего состояния
   - 5.4. Матрица стейкхолдеров
   - 5.5. Анализ аналогов и конкурентов
   - 5.6. Обоснование необходимости разработки
   - 5.7. Экономическое обоснование (ROI)

6. ПРОЕКТНАЯ ЧАСТЬ
   - 6.1. Модель требований к ПО (ЛР2)
     - 6.1.1. Use Case диаграмма
     - 6.1.2. Спецификация прецедентов

10.1. WBS (иерархическая структура работ)

10.2. Диаграмма Ганта

10.3. Оценка трудоёмкости (COCOMO)

10.4. Управление рисками

11. ЗАКЛЮЧЕНИЕ

11.1. Результаты выполнения проекта

11.2. Достигнутые цели и решённые задачи

11.3. Перспективы развития системы

11.4. Выводы

12. СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

13. ПРИЛОЖЕНИЯ

13.1. Листинги ключевых модулей

13.2. Скриншоты интерфейсов

13.3. Результаты тестирования

13.4. Отчёт JaCoCo

2.2. Требования к оформлению

Параметр	Требование
Формат файла	PDF
Объём	50-80 страниц
Шрифт	Times New Roman, 14 pt
Межстрочный интервал	1.5

Параметр	Требование
Поля	Левое 30 мм, остальные 15 мм
Нумерация страниц	Снизу по центру
Иллюстрации	Подпись снизу: «Рисунок 1 — Название»
Таблицы	Подпись сверху: «Таблица 1 — Название»
Список литературы	Не менее 15 источников

3. ТРЕБОВАНИЯ К ПРЕЗЕНТАЦИИ

3.1. Структура презентации (12-15 слайдов)

Слайд 1: Титульный

├── Полное название проекта

├── Автор (ФИО, группа)

├── Научный руководитель

├── СКФУ (логотип)

└── Год

Слайд 2: Актуальность и цель

├── Проблема (2-3 предложения)

├── Цель проекта

├── Задачи (4-6 пунктов)

└── Объект и предмет исследования

Слайд 3: Бизнес-модель (ЛР1)

├── IDEF0 A-0 диаграмма
├── Основные бизнес-процессы
└── Ключевые стейкхолдеры

Слайд 4: Системные требования (ЛР2)

├── Use Case диаграмма
├── Ключевые акторы
├── Основные прецеденты
└── Domain Model (кратко)

Слайд 5: Архитектура PCMEF (ЛР3)

├── Диаграмма пакетов
├── Распределение слоёв
├── Ключевые ADR
└── Выбор технологий

Слайд 6: База данных (ЛР4-ЛР5)

├── ER-диаграмма
├── Ключевые таблицы
├── Нормализация до 3НФ
└── ORM-маппинг

Слайд 7: Реализация бэкенда (ЛР6-ЛР7, ЛР14-ЛР16)

├── Структура проекта (скриншот)
├── JPA-сущности со связями
├── REST API (таблица эндпоинтов)
├── Безопасность (JWT)

Слайд 8: Рефакторинг и качество (ЛР8-ЛР12)

├── Применённые паттерны GoF
├── Результаты статического анализа (SonarQube)
├── Покрытие тестами (JaCoCo)
└── Ключевые метрики

Слайд 9: Фронтенд (ЛР20-ЛР25)
├── Структура React-приложения
├── Основные компоненты
├── Скриншоты интерфейса (2-3)
└── Адаптивная вёрстка

Слайд 10: Интеграция и тестирование (ЛР17, ЛР22, ЛР25)
├── API-интеграция (Axios)
├── Обработка ошибок
├── Интеграционные тесты
├── Нагрузочное тестирование

Слайд 11: Развёртывание (ЛР26)
├── Dockerfile (многостадийная сборка)
├── docker-compose.yml
├── Схема развёртывания
└── Инструкция по запуску

Слайд 12: Демонстрация работы
├── Скриншоты работающего приложения
├── Демонстрация API (Swagger)
├── Демонстрация фронтенда
└── (или видеозапись)

Слайд 13: Результаты и выводы

├── Достигнутые результаты

├── Выполненные задачи

├── Решённые проблемы

├── Перспективы развития

Слайд 14: Заключение

├── Спасибо за внимание

├── Вопросы?

├── Ссылка на репозиторий (QR-код)

└── Контакты

3.2. Требования к оформлению презентации

Параметр	Требование
Формат	PPTX, PDF
Количество слайдов	12-15
Шрифт	Arial, Calibri, 24-28 pt (заголовки), 18-20 pt (текст)
Изображения	Высокое разрешение, читаемые
Диаграммы	Векторные (не скриншоты)
Анимация	Минимальная (не отвлекающая)
QR-код	Ссылка на репозиторий

4. ТРЕБОВАНИЯ К ДЕМОНСТРАЦИИ

4.1. Что должно быть продемонстрировано

Компонент	Что показывать
Бэкенд	Запуск приложения, Swagger UI, вызов API через Postman/браузер
База данных	Подключение к PostgreSQL, структура таблиц, данные
Фронтенд	Регистрация, логин, просмотр списка, создание, редактирование, удаление
Тестирование	Запуск тестов (mvn test), отчёт JaCoCo
Docker	Сборка образа, запуск через docker-compose

4.2. Чек-лист демонстрации

Бэкенд запущен и работает

Swagger UI доступен (http://localhost:8080/swagger-ui.html)

API возвращает корректные данные

Фронтенд загружается и работает

Регистрация нового пользователя

Вход в систему

Просмотр списка мероприятий

Создание нового мероприятия

Редактирование мероприятия

Удаление мероприятия

Регистрация на мероприятие

Просмотр моих регистраций

Адаптивная вёрстка (на мобильном/эмуляторе)

Запуск тестов (зелёные)

Docker-сборка и запуск

5. КОНТРОЛЬНЫЕ ВОПРОСЫ ДЛЯ ЗАЩИТЫ

По бизнес-анализу (ЛР1)

1. Что такое IDEF0 и какие элементы входят в диаграмму?
2. Как рассчитывался ROI? Какие показатели учитывались?
3. По требованиям и моделированию (ЛР2-ЛР3)
4. Чем отличается Domain Model от Design Class Diagram?
5. Что такое PCMEF? Назначение каждого слоя?
6. Какие ADR были приняты и почему?

По базе данных и ORM (ЛР4-ЛР5, ЛР14)

1. Какие связи между сущностями реализованы в JPA?
2. Какие стратегии нормализации применялись?

По реализации бэкенда (ЛР6-ЛР7, ЛР15-ЛР16)

1. Как реализована аутентификация и авторизация?
2. Какие паттерны GoF использованы и где?

По тестированию (ЛР8, ЛР17, ЛР25)

1. Какое покрытие кода тестами? Как достигнуто?
2. Как организовано интеграционное тестирование?

По фронтенду (ЛР20-ЛР25)

1. Как организована маршрутизация в React?
2. Как выполняется валидация форм?

По развёртыванию (ЛР26)

1. Как устроен Dockerfile? Почему multi-stage?
2. Как настроен docker-compose для приложения с БД?

Общие вопросы

1. Какие трудности возникли при разработке?

2. Что планируется улучшить в будущем?

6. КРИТЕРИИ ОЦЕНИВАНИЯ

6.1. Оценка пояснительной записки (30 баллов)

Критерий	**Макс. балл**	**Описание**
Полнота содержания	8	Все разделы, соответствие структуре
Качество оформления	5	ГОСТ, читаемость, иллюстрации
Глубина анализа	6	Обоснованность выводов, качество диаграмм
Стиль изложения	5	Грамотность, логичность, научный стиль
Оригинальность	6	Антиплагиат >70%

6.2. Оценка презентации (20 баллов)

Критерий	Макс. балл	Описание
Структура и наглядность	5	Логичность, визуализация
Полнота информации	5	Охват всех этапов

Критерий	Макс. балл	Описание
		разработки
Качество слайдов	5	Читаемость, дизайн, отсутствие ошибок
Соответствие регламенту	5	10-15 слайдов, тайминг

6.3. Оценка демонстрации (30 баллов)

Критерий	Макс. балл	Описание
Работоспособность бэкенда	6	API работает, Swagger доступен
Работоспособность фронтенда	6	Все функции работают
Тестирование	6	Тесты проходят, покрытие >40%
Docker	6	Сборка и запуск работают
Полнота демонстрации	6	Показаны все

Критерий	Макс. балл	Описание
		ключевые сценарии

6.4. Оценка ответов на вопросы (20 баллов)

Критерий	Макс. балл	Описание
Понимание архитектуры	5	Глубокое понимание PCMEF
Понимание кода	5	Способность объяснить реализацию
Понимание тестирования	5	Знание покрытия, видов тестов
Профессиональный кругозор	5	Ответы на общие вопросы

6.5. Итоговая шкала

Баллы	**Оценка (традиционная)**	**Оценка (ECTS)**
90-100	Отлично	A

Баллы	Оценка (традиционная)	Оценка (ECTS)
75-89	Хорошо	B-C
60-74	Удовлетворительно	D-E
0-59	Неудовлетворительно	FX-F

7. ЧЕК-ЛИСТ ГОТОВНОСТИ К ЗАЩИТЕ

1. Документация
2. Пояснительная записка (50-80 стр.) в формате PDF
3. Презентация (12-15 слайдов)
4. Техническое задание (приложение к записке)
5. Руководство пользователя
6. Руководство администратора
7. Видеодемонстрация (запасной вариант)
8. Код
9. Репозиторий на GitHub/GitLab
10. Код откомпилирован и запускается
11. Все тесты проходят (mvn test)
12. JaCoCo отчёт (покрытие >40%)
13. Dockerfile и docker-compose.yml
14. Инфраструктура
15. Бэкенд запущен
16. Фронтенд запущен

17. PostgreSQL запущена
18. Docker-образ собран
19. Личные материалы
20. Зачётная книжка
21. Распечатанный титульный лист (для подписи)
22. Флешка с проектом (запасной вариант)

8. РЕКОМЕНДАЦИИ ПО ПОДГОТОВКЕ

8.1. За неделю до защиты

Завершить пояснительную записку

Собрать все артефакты в репозитории

Проверить работоспособность на чистом окружении

Подготовить презентацию

8.2. За день до защиты

Распечатать пояснительную записку

Проверить демонстрацию (все сценарии)

Проверить Docker-запуск

Подготовить ответы на вопросы

8.3. В день защиты

Прийти за 15 минут

Подготовить ноутбук/флешку

Прогнать демонстрацию ещё раз

Дышать глубоко, говорить уверенно

9. ПРИМЕР РАСПРЕДЕЛЕНИЯ ВРЕМЕНИ НА ЗАЩИТЕ

Этап	Длительность	Что делать
Вступление	30 сек	Представиться, назвать тему

Этап	Длительность	Что делать
Презентация	5-7 мин	Краткий обзор проекта (6-8 слайдов)
Демонстрация	5-7 мин	Показать ключевые функции
Заключение	30 сек	Итоги, ответы на вопросы
Вопросы	5-10 мин	Ответы на вопросы комиссии
ИТОГО	12-18 мин	

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

МЕТОДИЧЕСКИЕ УКАЗАНИЯ

по организации и проведению самостоятельной работы по дисциплине «Программная инженерия» для студентов направления подготовки 09.03.04 Программная инженерия
Направленность (профиль)
«Разработка и сопровождение программного обеспечения»

Ставрополь 2026

ВВЕДЕНИЕ

1. Цель и задачи самостоятельной работы

Самостоятельная работа студентов (СРС) является неотъемлемой частью образовательного процесса по дисциплине «Программная инженерия» и направлена на формирование у обучающихся профессиональных компетенций, необходимых для разработки качественного программного обеспечения.

Цель самостоятельной работы — закрепление и углубление теоретических знаний, полученных на лекционных занятиях, а также формирование практических навыков проектирования, разработки и документирования программных систем.

Задачи самостоятельной работы:

№	Задача	Формируемая компетенция
1	Изучение теоретического материала по программной инженерии	Способность применять системный подход для решения поставленных задач
2	Выполнение практических заданий (лабораторных работ)	Способность разрабатывать алгоритмы и программы, пригодные для практического использования
3	Работа с научно-технической литературой	Способность осуществлять поиск, критический анализ и синтез информации
4	Подготовка отчётов и презентаций	Способность осуществлять деловую коммуникацию в профессиональной среде
5	Управление временем и ресурсами проекта	Способность управлять своим временем, выстраивать траекторию саморазвития

2. Объём и распределение самостоятельной работы

Семестр	Лекции	Лабораторные работы	Самостоятельная работа	Всего
4 семестр	18 ч	36 ч	54 ч	108 ч

Семестр	Лекции	Лабораторные работы	Самостоятельная работа	Всего
5 семестр	18 ч	54 ч	72 ч	144 ч
ИТОГО	36 ч	90 ч	126 ч	252 ч

1. ОБЩАЯ ХАРАКТЕРИСТИКА САМОСТОЯТЕЛЬНОЙ РАБОТЫ

1.1. Виды самостоятельной работы

Вид СРС	Описание	Трудоёмкость
Подготовка к лекциям	Изучение конспектов, рекомендуемой литературы	0.5 часа в неделю
Выполнение лабораторных работ	Разработка кода, оформление отчётов	4-6 часов в неделю
Подготовка к защите	Повторение материала, подготовка ответов	2-3 часа перед защитой
Изучение дополнительной литературы	Работа с источниками по темам курса	1-2 часа в неделю
Работа над курсовым проектом	Сквозная разработка программной системы	3-5 часов в неделю
Подготовка презентаций	Оформление результатов, репетиция выступления	2-3 часа перед защитой

1.2. Этапы самостоятельной работы по семестрам

4 семестр (9 лабораторных работ)

Неделя	Этап	Содержание СРС	Отчётность
1-2	ЛР1	Анализ предметной области, бизнес-моделирование	Сдача отчёта
3-4	ЛР2	Построение Domain Model, Use Case диаграммы	Сдача отчёта
5-6	ЛР3	Проектирование архитектуры PCMEF, ADR	Сдача отчёта
7-8	ЛР4	Проектирование БД, ER-диаграмма	Сдача отчёта
9-10	ЛР5	Создание DDL-скриптов	Сдача отчёта
11-12	ЛР6	Реализация JPA-сущностей	Сдача отчёта
13-14	ЛР7	Реализация репозиториев	Сдача отчёта
15-16	ЛР8	Модульное тестирование	Сдача отчёта
17-18	ЛР9	Промежуточная защита	Защита

5 семестр (18 лабораторных работ)

Неделя	Этап	Содержание СРС	Отчётность
1-2	ЛР10-ЛР11	Спецификации прецедентов, диаграммы последовательности	Сдача отчёта
3-4	ЛР12-ЛР13	Паттерны GoF, уточнение классов	Сдача отчёта
5-6	ЛР14-ЛР16	JPA-связи, Mediator, Control слои	Сдача отчёта

Неделя	Этап	Содержание СРС	Отчётность
7-8	ЛР17-ЛР18	Интеграционное тестирование, JWT	Сдача отчёта
9-10	ЛР19	REST API	Сдача отчёта
11-12	ЛР20-ЛР22	React, компоненты, интеграция	Сдача отчёта
13-14	ЛР23-ЛР25	Валидация, адаптивная вёрстка, ошибки	Сдача отчёта
15-16	ЛР26	Docker-контейнеризация	Сдача отчёта
17-18	ЛР27	Итоговая защита	Защита

1.3. Требования к организации рабочего пространства

Для эффективной самостоятельной работы студенту необходимо:

Инструмент	Назначение	Ссылка
Git + GitHub/GitLab	Контроль версий, хранение кода	https://github.com
IntelliJ IDEA / VS Code	Разработка кода	https://www.jetbrains.com/idea/
PlantUML / Draw.io	Создание диаграмм	https://plantuml.com
Postman	Тестирование API	https://www.postman.com
Docker Desktop	Контейнеризация	https://www.docker.com
DBeaver / pgAdmin	Работа с БД	https://dbeaver.io

2. МЕТОДИЧЕСКИЕ РЕКОМЕНДАЦИИ ПО ИЗУЧЕНИЮ ТЕОРЕТИЧЕСКОГО МАТЕРИАЛА

2.1. План самостоятельного изучения тем

Тема лекции	Вопросы для самостоятельного изучения	Литература
Введение в ПИ	История развития ПИ, сравнение с другими инженерными дисциплинами	[1, гл.1]
ЖЦ ПО	Сравнение Waterfall, Agile, Spiral, RUP	[1, гл.2]
Инструментарий ПИ	Обзор Jira, Trello, SonarQube, Jenkins	[1, гл.3]
Управление требованиями	Методы FURPS+, ATAM, трассировка требований	[2, гл.4]
UML	Диаграммы компонентов, развёртывания, пакетов	[3]
Архитектура PCMEF	Сравнение с Clean Architecture, Hexagonal	[4]
Паттерны GoF	Детальное изучение 23 паттернов	[5]
JPA/Hibernate	Стратегии наследования, кэширование	[6]
Spring Security	OAuth2, OpenID Connect, SSO	[7]
React	Хуки, контекст, оптимизация	[8]
Docker	Сети, тома, оркестрация	[9]

2.2. Рекомендуемая литература

Основная литература:

1. Мацяшек, Л. Практическая программная инженерия на основе учебного примера / Л. Мацяшек. — М.: Вильямс, 2021.
2. Соммервилл, И. Инженерия программного обеспечения / И. Соммервилл. — 10-е изд. — М.: Вильямс, 2018.
3. Ларман, К. Применение UML и шаблонов проектирования / К. Ларман. — 3-е изд. — М.: Вильямс, 2018.

4. Мартин, Р. Чистая архитектура / Р. Мартин. — СПб.: Питер, 2018.
5. Гамма, Э., Хелм, Р., Джонсон, Р., Влиссидес, Дж. Приёмы объектно-ориентированного проектирования. Паттерны проектирования. — СПб.: Питер, 2020.
6. Бауэр, К., Кинг, Г. Java Persistence API и Hibernate. — М.: ДМК Пресс, 2021.
7. Спэлл, Б., Оттвилл, Д. Spring Security в действии. — М.: ДМК Пресс, 2020.
8. Банков, Д. React в действии. — М.: ДМК Пресс, 2022.
9. Митчелл, Д. Docker в действии. — 2-е изд. — М.: ДМК Пресс, 2021.

Дополнительная литература:

1. Фаулер, М. Шаблоны корпоративных приложений. — М.: Вильямс, 2016.
2. Брукс, Ф. Мифический человеко-месяц. — СПб.: Символ-Плюс, 2019.
3. Стиллмен, М., Грин, Д. API в действии. — М.: ДМК Пресс, 2021.

2.3. Электронные ресурсы

Ресурс	Назначение	Ссылка
IEEE Xplore	Научные статьи по ПИ	https://ieeexplore.ieee.org
ACM Digital Library	Публикации по программной инженерии	https://dl.acm.org
Stack Overflow	Решение практических задач	https://stackoverflow.com
GitHub	Примеры кода, open-source проекты	https://github.com
Baeldung	Учебные материалы по Spring	https://www.baeldung.com
React Documentation	Официальная документация React	https://react.dev

2.4. Рекомендации по конспектированию

При самостоятельном изучении теоретического материала рекомендуется:

1. Вести электронный конспект в формате Markdown в репозитории (папка notes/)
2. Фиксировать ключевые определения и термины (глоссарий)
3. Создавать схемы и диаграммы для визуализации сложных концепций
4. Сравнивать и систематизировать информацию в таблицах
5. Формулировать вопросы к преподавателю для уточнения на лекциях

3. МЕТОДИЧЕСКИЕ УКАЗАНИЯ ПО ВЫПОЛНЕНИЮ ПРАКТИЧЕСКИХ РАБОТ

3.1. Общие требования

Самостоятельность выполнения — работы должны выполняться индивидуально (кроме специально оговорённых групповых проектов)

Своевременность сдачи — каждая работа сдаётся в установленный срок

Полнота реализации — работа должна быть выполнена в полном объёме

Документирование — каждая работа сопровождается отчётом

Контроль версий — код и отчёты хранятся в Git-репозитории

3.2. Порядок выполнения лабораторной работы

Этап	Действие	Время	Результат
1	Ознакомление с заданием	15 мин	Понимание цели и задач
2	Изучение теоретического материала	1-2 часа	Конспект, глоссарий
3	Выполнение практической части	4 - 8 часов	Реализация, код
4	Тестирование и отладка	1-2 часа	Работоспособный код
5	Оформление отчёта	1-2 часа	Отчёт в формате PDF
6	Пуш в репозиторий	10 мин	Коммит с кодом и отчётом
7	Подготовка к защите	1 час	Понимание работы, ответы на вопросы

Этап	Действие	Время	Результат
8	Защита	10-15 мин	Оценка

3.3. Связь лабораторных работ с курсовым проектом

Все лабораторные работы являются последовательными этапами выполнения курсового проекта:

ЛР1 → Бизнес-модель
↓
ЛР2 → Системные требования
↓
ЛР3 → Архитектура PCMEF
↓
ЛР4-ЛР5 → База данных
↓
ЛР6-ЛР7 → Реализация ядра
↓
ЛР8 → Тестирование
↓
ЛР10-ЛР19 → Расширение функционала
↓
ЛР20-ЛР25 → Веб-интерфейс
↓
ЛР26 → Docker
↓
ЛР27 → Итоговая защита

3.4. Рекомендации по отладке

Проблема	Рекомендации
Код не компилируется	Проверить синтаксис, зависимости, версии JDK
Тесты не проходят	Использовать отладчик (debugger), проверить моки
API не отвечает	Проверить порт, CORS-настройки, логи сервера
БД не подключается	Проверить credentials, URL, запущен ли

Проблема	Рекомендации
	сервер БД
Фронтенд не загружается	Проверить консоль браузера, сетевые запросы

4. ВОПРОСЫ ДЛЯ СОБЕСЕДОВАНИЯ ПО САМОСТОЯТЕЛЬНОМУ ИЗУЧЕНИЮ ЛИТЕРАТУРЫ

4.1. Раздел «Введение в программную инженерию»

1. Каковы основные причины возникновения «кризиса программного обеспечения»?
2. Чем программная инженерия отличается от традиционной инженерии?
3. Какие стандарты регламентируют процессы жизненного цикла ПО?
4. Каковы основные стадии жизненного цикла ПО?
5. В чём преимущества и недостатки каскадной модели?

4.2. Раздел «Методологии разработки»

6. Каковы основные ценности Agile-манифеста?
7. В чём разница между Scrum и Kanban?
8. Какие роли определены в Scrum?
9. Что такое «спринт» и «бэклог»?
10. Как выбирать методологию разработки для конкретного проекта?

4.3. Раздел «Управление требованиями»

11. Чем функциональные требования отличаются от нефункциональных?
12. Что такое FURPS+?
13. Какие методы сбора требований наиболее эффективны?
14. Что такое трассировка требований?
15. Как формулируются критерии приёмки?

4.4. Раздел «Архитектура ПО и PCMEF»

16. Что такое архитектура ПО и из каких элементов она состоит?
17. Каковы принципы «хорошей архитектуры»?
18. Что такое PCMEF? Опишите каждый слой.
19. Каковы правила зависимостей в PCMEF?
20. Как адаптируется PCMEF для веб-приложений? Для мобильных?

4.5. Раздел «Паттерны проектирования GoF»

21. Что такое паттерн проектирования? Какие группы паттернов выделяют GoF?
22. В чём отличие Singleton от Factory Method?

23. Когда применяется паттерн Strategy?
24. Какие задачи решает паттерн Observer?
25. Приведите пример использования паттерна Facade.

4.6. Раздел «Базы данных и ORM»

26. Что такое нормализация? Какие существуют нормальные формы?
27. Как в реляционной БД реализуются связи 1:N и M:N?
28. Что такое JPA и Hibernate?
29. В чём разница между LAZY и EAGER загрузкой?
30. Как решить проблему N+1 запросов в JPA?

4.7. Раздел «Тестирование и качество»

31. Какие уровни тестирования существуют?
32. Что такое модульное тестирование и чем оно отличается от интеграционного?
33. Что такое TDD (Test-Driven Development)?
34. Как JaCoCo измеряет покрытие кода?
35. Какие инструменты используются для интеграционного тестирования?

4.8. Раздел «Веб-разработка и REST API»

36. Что такое REST API? Каковы основные принципы REST?
37. Какие HTTP-методы соответствуют CRUD-операциям?
38. Что такое JWT и как он устроен?
39. Как реализована маршрутизация в React Router?
40. Для чего нужны хуки useState и useEffect?

4.9. Раздел «DevOps и контейнеризация»

41. Что такое Docker и чем контейнер отличается от виртуальной машины?
42. Зачем нужен multi-stage build в Dockerfile?
43. Что такое docker-compose и для чего он используется?
44. Каковы принципы CI/CD?
45. Какие инструменты используются для непрерывной интеграции?

5. КРИТЕРИИ ОЦЕНИВАНИЯ КОМПЕТЕНЦИЙ

5.1. Критерии оценки лабораторных работ

Критерий	Отлично	Хорошо	Удовлетворительно	Неудовлетворительно
Полнота выполнения	100% задач	80-90% задач	60-70% задач	<60% задач
Качество	Соответст	Незначител	Существенны	Код не читаем

Критерий	Отлично	Хорошо	Удовлетворительно	Неудовлетворительно
кода	вует стандартам, чистый	ьные нарушения	е нарушения	
Тестирование	Покрытие >60%	Покрытие 40-60%	Покрытие 20-40%	Покрытие <20%
Оформление отчёта	Полностью соответствует	Незначительные недочёты	Существенные недочёты	Отсутствует
Защита	Глубокое понимание	Понимание с подсказками	Поверхностное понимание	Не понимает

5.2. Шкала перевода баллов в оценку

Процент выполнения	Баллы	Оценка
90-100%	90-100	Отлично
75-89%	75-89	Хорошо
60-74%	60-74	Удовлетворительно
0-59%	0-59	Неудовлетворительно

5.3. Критерии оценки устного собеседования

Критерий	Отлично	Хорошо	Удовлетворительно	Неудовлетворительно
Полнота ответа	Раскрыты все вопросы	Раскрыты основны	Раскрыты не все вопросы	Ответ отсутствует

Критерий	Отлично	Хорошо	Удовлетворительно	Неудовлетворительно
		е вопросы		
Глубина понимания	Свободное оперирование понятиями	Понимание с небольшими уточнениями	Поверхностное понимание	Не понимает материал
Приведение примеров	Самостоятельные примеры	Примеры из лекций	Примеры с подсказками	Нет примеров
Аргументация	Логичная, обоснованная	Логичная	Не всегда логичная	Отсутствует

5.4. Индикаторы достижения компетенций

Компетенция	Индикатор	Критерий оценки
УК-1	Способен осуществлять поиск, критический анализ и синтез информации	Правильно выбранные источники, структурированный анализ
ОПК-6	Способен разрабатывать алгоритмы и программы	Работоспособный код, соответствие стандартам
ОПК-8	Способен осуществлять поиск, хранение, обработку и	Работа с БД, API, корректные запросы

Компетенция	Индикатор	Критерий оценки
	анализ информации	
ПК-2	Способен выполнять проектирование и разработку архитектуры	Корректная диаграмма пакетов PCMEF, ADR
ПК-4	Способен проводить тестирование, отладку и оценивать качество ПО	Наличие тестов, покрытие >40%, отладка

5.5. Итоговая оценка за семестр

4 семестр:

Итоговая оценка = (Σ баллов за ЛР1-ЛР8) × 0.7 + (Баллы за ЛР9) × 0.3

5 семестр:

text

Итоговая оценка = (Σ баллов за ЛР10-ЛР26) × 0.6 + (Баллы за ЛР27) × 0.4

ПРИЛОЖЕНИЕ А. Примерный календарный план самостоятельной работы

Неделя	4 семестр	5 семестр
1	ЛР1 — анализ предметной области	ЛР10 — спецификации прецедентов
2	ЛР1 — завершение, отчёт	ЛР11 — диаграммы последовательности

Неделя	4 семестр	5 семестр
3	ЛР2 — требования	ЛР12 — паттерны GoF
4	ЛР2 — Domain Model	ЛР13 — уточнение классов
5	ЛР3 — архитектура PCMEF	ЛР14 — JPA-связи
6	ЛР3 — ADR, интерфейсы	ЛР15 — Mediator-слой
7	ЛР4 — ER-диаграмма	ЛР16 — Control-слой
8	ЛР4 — нормализация	ЛР17 — интеграционное тестирование
9	ЛР5 — DDL-скрипты	ЛР18 — JWT
10	ЛР6 — JPA-сущности	ЛР19 — REST API
11	ЛР6 — бизнес-методы	ЛР20 — React инициализация
12	ЛР7 — репозитории	ЛР21 — компоненты
13	ЛР7 — тестирование репозиториев	ЛР22 — Axios интеграция
14	ЛР8 — модульные тесты	ЛР23 — валидация форм
15	ЛР8 — JaCoCo	ЛР24 — адаптивная вёрстка

Неделя	4 семестр	5 семестр
16	ЛР9 — подготовка презентации	ЛР25 — обработка ошибок
17	ЛР9 — промежуточная защита	ЛР26 — Docker
18	—	ЛР27 — итоговая защита

ПРИЛОЖЕНИЕ Б. Чек-лист готовности к защите лабораторной работы

1. Код выполнен в полном объёме
2. Код соответствует архитектуре PCMEF
3. Тесты проходят (mvn test)
4. JaCoCo отчёт сгенерирован (покрытие >40%)
5. Отчёт оформлен по шаблону
6. Отчёт загружен в репозиторий
7. Код закоммичен и запушен
8. README.md обновлён
9. Диаграммы встроены в отчёт
10. Контрольные вопросы проработаны