

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

МЕТОДИЧЕСКИЕ УКАЗАНИЯ
по выполнению лабораторных работ
по дисциплине «Проектирование архитектуры проектных систем»
для студентов направления подготовки 09.03.04 Программная инженерия
Направленность (профиль)
«Инженерия сложных программных систем»

Ставрополь 2026

Содержание

Введение.....	3
Лабораторная работа №1	10
Лабораторная работа №2	21
Лабораторная работа №3	30
Лабораторная работа №4	44
Лабораторная работа №5	61
Лабораторная работа №6	84
Лабораторная работа №7	99
Лабораторная работа №8	114
Лабораторная работа №9	140
Лабораторная работа №10	174
Лабораторная работа №11	207
Лабораторная работа №12	243
Лабораторная работа №13	276
Лабораторная работа №14	321
Лабораторная работа №15	350
Лабораторная работа №16	384
Лабораторная работа №17	419
Лабораторная работа №18	461

Введение

1. Место лабораторных работ в структуре дисциплины

Лабораторный практикум является ключевой составляющей дисциплины «Проектирование архитектуры программных систем» и направлен на формирование у студентов системных компетенций в области проектирования, оценки и эволюции архитектуры программных систем среднего и крупного масштаба.

В ходе выполнения лабораторных работ студенты последовательно проходят все этапы архитектурного проектирования: от анализа нефункциональных требований и документирования архитектуры до реализации сложных архитектурных паттернов (Clean Architecture, DDD, CQRS, Event Sourcing, Saga) и оценки архитектурного технического долга.

2. Структура лабораторного практикума

Лабораторный практикум состоит из 18 работ, сгруппированных в 6 модулей:

Модуль	№ ЛР	Тематика
Модуль 1	1–3	Нефункциональные требования и документирование архитектуры
Модуль 2	4–6	Clean Architecture (рефакторинг PCMEF → Clean Architecture)
Модуль 3	7–10	Domain-Driven Design (стратегический и тактический)
Модуль 4	11–13	CQRS и Event Sourcing
Модуль 5	14–16	Архитектурная эволюция и оценка

Модуль	№ ЛР	Тематика
Модуль 6	17–18	Архитектурный технический долг и итоговый проект

3. Сквозной принцип проектирования

Все лабораторные работы выполняются на основе единого сквозного проекта, который студент выбирает в начале семестра. Это позволяет:

проследить эволюцию архитектуры от простого монолита до микросервисной системы;

применять на практике все изучаемые паттерны и подходы на одном примере;

получить к концу семестра полноценный архитектурный проект, готовый к защите.

Варианты сквозных проектов (15 вариантов):

	Название проекта	Домен
	Logistics Management System	Логистика и доставка
	E-Commerce Platform	Интернет-торговля
	Digital Wallet	Финансовые технологии
	Library Management System	Библиотечное дело
	Hotel Booking System	Гостиничный бизнес
	Online Courses Platform	Образовательные технологии

	Название проекта	Домен
	Medical Information System	Здравоохранение
	CRM for Retail	Управление взаимоотношениями с клиентами
	Payment Gateway	Платёжные системы
0	Warehouse Management System	Управление складом
1	Social Network	Социальные сети
2	Video Hosting Platform	Видеохостинг
3	IoT Monitoring System	Интернет вещей
4	Cryptocurrency Exchange	Криптовалютная биржа
5	Food Delivery Service	Сервис доставки еды

4. Формат лабораторных работ

Каждая лабораторная работа имеет единую структуру:

Раздел	Содержание
Цель работы	Формулировка формируемых компетенций
Задачи работы	Конкретные шаги для достижения цели
Теоретическое обоснование	Краткое изложение ключевых концепций и паттернов
Пример выполнения	Детальный разбор на примере сквозного проекта (вариант 1)
Индивидуальные задания	15 вариантов для самостоятельного выполнения
Контрольные вопросы	Вопросы для самопроверки и защиты
Требования к отчёту	Состав и формат отчёта по работе
Критерии оценивания	Система оценки с указанием максимальных баллов

5. Требования к выполнению

5.1. Код и технологии

Язык программирования: Java 17+

Фреймворк: Spring Boot 3.x

Система сборки: Maven или Gradle

Тестирование: JUnit 5, Mockito, TestContainers

Документирование диаграмм: PlantUML или Structurizr

5.2. Отчётность

По каждой лабораторной работе оформляется отчёт в формате PDF, содержащий:

титульный лист;

цель и задачи;

описание выполнения работы;

код ключевых компонентов (фрагменты);

диаграммы (при наличии);

ответы на контрольные вопросы;

вывод.

5.3. Допуск к экзамену

Для получения допуска к экзамену необходимо:

выполнить и защитить не менее 16 из 18 лабораторных работ;

защитить итоговый проект (ЛР 18).

6. Связь с теоретической частью

Лабораторные работы синхронизированы с лекционным курсом:

Лекция	Тема	Соответствующие ЛР
1–2	Нефункциональные требования, архитектурные стили	ЛР 1, 2, 3
3–4	Документирование, C4	ЛР 2, 3
4–6	Clean Architecture, DDD	ЛР 4–10
7–9	CQRS, Event Sourcing, эволюция	ЛР 11–16
—	Технический долг, оценка	ЛР 17, 18

7. Компетенции, формируемые в ходе лабораторных работ

Категория	Компетенции
Аналитические	Анализ нефункциональных требований, оценка архитектурных рисков (АТАМ), расчёт технического долга
Проектные	Выбор архитектурного стиля, проектирование ограниченных контекстов (DDD), версионирование API
Технические	Реализация Clean Architecture, CQRS, Event Sourcing, Saga, Strangler pattern, Feature Toggles
Документационные	С4 диаграммы, ADR, архитектурная документация

8. Методические рекомендации для студентов

Выберите проект в начале семестра и используйте его во всех работах — это сэкономит время и повысит качество итогового проекта.

Ведите репозиторий на GitHub/GitLab с кодом и диаграммами.

Не пропускайте теоретическое обоснование — оно содержит ключевые концепции, необходимые для выполнения задания.

Используйте пример выполнения как образец, но адаптируйте под свой проект.

Тестируйте код — многие работы требуют написания модульных тестов.

Сдавайте работы последовательно — каждая следующая работа опирается на предыдущую.

Выполнение лабораторных работ по дисциплине «Проектирование архитектуры программных систем» позволит студентам не только освоить современные архитектурные паттерны и подходы, но и получить практический опыт проектирования реальных систем среднего и крупного масштаба.

Успешное выполнение всех 18 работ обеспечит готовность к решению архитектурных задач в профессиональной деятельности, включая выбор архитектурного стиля, документирование, оценку рисков и управление техническим долгом.

Лабораторная работа №1

Анализ нефункциональных требований (FURPS+) на примере «Системы управления логистикой и доставкой»

1. Цель работы

Сформировать компетенции по выявлению, классификации и документированию нефункциональных требований для логистической системы, а также связыванию их с архитектурными решениями с использованием метода FURPS+ и сценариев качества.

2. Задачи работы

1. Изучить метод FURPS+ для классификации нефункциональных требований.
2. На основе сквозного примера «Система управления логистикой и доставкой» выделить нефункциональные требования.
3. Для каждого требования сформулировать сценарий качества (стимул → источник → среда → ответ → мера).
4. Связать требования с конкретными архитектурными решениями (EDA, CQRS, Event Sourcing, Saga, WebSocket, кэширование карт) и обосновать выбор.
5. Оформить результаты в виде таблицы.

3. Теоретическое обоснование

3.1. Нефункциональные требования в логистических системах

Для логистики критичны следующие атрибуты качества:

Атрибут	Значение для логистики
Производительность	Трекинг курьеров в реальном времени, расчёт маршрутов < 2 сек
Масштабируемость	Рост числа заказов в час пик

Атрибут	Значение для логистики
	(19:00–21:00)
Отказоустойчивость	При сбое сервиса маршрутизации курьер должен получить последний известный маршрут
Безопасность	Данные о местоположении курьеров и адресах клиентов
Консистентность	Один заказ не может быть назначен двум курьерам (Saga/Compensating transaction)
Поддерживаемость	Лёгкость добавления нового картографического провайдера

3.2. Метод FURPS+ (применительно к логистике)

Категория	Пример требования
Functionality	(Не НФТ, опускаем)
Usability	Диспетчер видит карту с курьерами с задержкой не более 3 сек
Reliability	Система продолжает назначать заказы при падении 1 из 5 экземпляров сервиса

Категория	Пример требования
Performance	Обновление координат курьера → UI диспетчера < 1 сек (p99)
Supportability	Возможность переключения провайдера карт без передеплойа (Map Provider Adapter)
+ Security	Токен доступа к API трекинга истекает через 15 минут
+ Operational	Система логирует каждый этап жизни заказа (создан → назначен → забран → доставлен)

3.3. Сценарий качества для логистики (шаблон)

Элемент	Пример для логистики
Стимул	500 курьеров одновременно отправляют координаты GPS
Источник	Мобильные приложения курьеров
Среда	Нормальная нагрузка, Redis для координат работает
Ответ	Система принимает и обрабатывает все данные
Мера	99.9% сообщений обработаны за < 500 мс

4. Пример выполнения (на основе логистики)

4.1. Описание сквозного примера

«Система управления логистикой и доставкой»

Функционал:

1. Курьер получает заказы через мобильное приложение
2. Диспетчер видит всех курьеров на карте в реальном времени
3. Автоматическое построение оптимального маршрута с учётом пробок
4. Назначение заказа ближайшему свободному курьеру
5. Трекинг доставки для клиента
6. Отчёты по эффективности курьеров

4.2. Применение FURPS+

Категория	Требование (ID)	Описание
Usability	U-1	Диспетчер видит перемещение курьеров на карте с задержкой не более 3 секунд
Reliability	R-1	Система продолжает назначать заказы при отказе 1 из 3 экземпляров сервиса назначения
Performance	P-1	Расчёт оптимального маршрута для заказа с 10 точками выполняется < 2 секунд
Supportability	S-1	Замена провайдера карт (Google Maps →

Категория	Требование (ID)	Описание
		OSRM) без изменения кода Use Cases
+ Security	Sec-1	Данные о местоположении курьеров передаются только авторизованному диспетчеру
+ Operational	O-1	Каждое событие жизни заказа (OrderAssigned, OrderPickedUp, OrderDelivered) сохраняется в Event Store

4.3. Сценарии качества

Сценарий 1 (Производительность: трекинг курьеров)

Стимул: 1000 курьеров отправляют GPS-координаты каждые 3 секунды

Источник: Мобильные приложения курьеров (WebSocket)

Среда: Система работает штатно, Redis для координат включён

Ответ: Система обновляет позиции курьеров на карте диспетчера

Мера: p99 задержка от получения координат до отображения на UI диспетчера ≤ 1 секунда

Сценарий 2 (Отказоустойчивость: сервис маршрутизации)

Стимул: Сервис маршрутизации (расчёт оптимального пути) становится недоступен

Источник: Сбой оборудования или сети

Среда: Есть заказ, ожидающий назначения

Ответ: Система назначает заказ курьеру по упрощённому алгоритму (по прямой дистанции)

Мера: Время назначения заказа увеличивается не более чем на 20% (с 5 до 6 секунд)

Сценарий 3 (Безопасность: доступ к трекингу)

Стимул: Неавторизованный запрос к API трекинга курьеров

Источник: Злоумышленник или скомпрометированный клиент

Среда: API защищено JWT-токенами

Ответ: Система отклоняет запрос с кодом 401 Unauthorized

Мера: 100% запросов без валидного токена отклоняются, событие логируется

Сценарий 4 (Масштабируемость: вечерний час пик)

Стимул: Резкий рост заказов до 5000 в минуту (19:00–20:00)

Источник: Клиенты, заказывающие доставку ужина

Среда: Обычная нагрузка — 500 заказов в минуту

Ответ: Система горизонтально масштабирует сервис назначения заказов

Мера: Время назначения заказа (p95) не превышает 5 секунд (при норме 3 сек)

Сценарий 5 (Консистентность: двойное назначение)

Стимул: Один и тот же заказ пытаются назначить два разных экземпляра сервиса

Источник: Параллельные обработчики очереди заказов (Kafka)

Среда: Высокая нагрузка, несколько консьюмеров

Ответ: Только один экземпляр успешно назначает заказ

Мера: 0% заказов, назначенных более чем одному курьеру
(проверяется журналом)

4.4. Связь с архитектурными решениями

Требование	Архитектурное решение	Обоснование
P-1 (расчёт маршрута < 2 сек)	Кэширование матрицы расстояний (Redis) + предвычисление популярных маршрутов	Тяжёлые вычисления не повторяются для одинаковых пар адресов
R-1 (отказоустойчивость назначения)	Паттерн Circuit Breaker (Resilience4j) + реплики сервиса (K8s) + Graceful degradation	При падении маршрутизатора используем fallback (прямое расстояние)
O-1 (аудит событий)	Event Sourcing (Event Store) — каждое изменение статуса заказа — событие	Полная история заказа, возможность восстановления состояния
Sec-1	JWT-токены +	Минимизация

Требование	Архитектурное решение	Обоснование
(безопасность трекинга)	TLS 1.3 + короткий TTL (15 мин) + refresh token	последствий при компрометации токена
Консистентность назначения	Saga orchestration (Orchestrator через Kafka) + идемпотентность	Предотвращение двойного назначения в распределённой среде
U-1 (карта курьеров < 3 сек)	WebSocket (STOMP) + CQRS (отдельная read-модель для позиций)	Push-обновления вместо polling, оптимизация чтения
S-1 (смена провайдера карт)	Clean Architecture + интерфейс MapProviderPort в слое Use Cases	Адаптер под конкретный провайдер — в Infrastructure

5. Индивидуальные задания (15 вариантов)

Каждый вариант содержит специализированное требование к логистической системе. Студент должен:

Добавить это требование в FURPS+.

Составить для него сценарий качества.

Заполнить строку в таблице «Требование → Архитектурное решение → Обоснование».

Вариант	Дополнительное НФТ
1	Умная маршрутизация: учёт пробок в реальном времени, пересчёт маршрута каждые 5 минут
2	Холодное хранение: архив заказов старше 1 года выгружается в S3, но должен быть доступен за < 1 мин
3	Офлайн-режим курьера: при потере связи курьер должен завершить доставку, а после синхронизировать данные
4	Приоритетные заказы: заказы с пометкой «эспресс» обрабатываются в первую очередь (latency < 30 сек)
5	Геозонирование: при въезде курьера в зону ресторана автоматически отправляется push-уведомление
6	Мультиязычность: интерфейс диспетчера на русском, английском, китайском (переключение без перезапуска)
7	Аудит изменений диспетчера: кто и когда переназначил заказ другому курьеру
8	Прогнозирование времени доставки: ML-модель с точностью ± 5 минут (обновление модели без

Вариант	Дополнительное НФТ
	деплой)
9	Rate limiting для API курьеров: не более 1 запроса координат в 2 секунды с одного устройства
10	Тенантность (multi-tenant): разные логистические компании работают в изолированных контекстах
11	Интеграция с внешними агрегаторами (Яндекс.Еда, Delivery Club) через API с таймаутом 3 сек
12	Качество GPS-данных: отбрасывание выбросов (шум), сглаживание трека по алгоритму Калмана
13	Назначение с учётом рейтинга курьера: лучший курьер получает приоритет на дорогие заказы
14	Отказоустойчивость брокера сообщений: дублирование Kafka между 3 дата-центрами
15	Мониторинг ETA: если курьер опаздывает более чем на 10 минут — уведомление диспетчеру

6. Контрольные вопросы

1. Почему в логистической системе критична консистентность при назначении заказа? Как её обеспечить в распределённой архитектуре?

2. Для чего в трекинге курьеров используется WebSocket, а не HTTP polling?
3. Какой архитектурный паттерн позволяет сохранять полную историю изменений статуса заказа?
4. Почему в требовании «смена провайдера карт» помогает Clean Architecture?
5. В чём разница между отказоустойчивостью (fault tolerance) и устойчивостью к сбоям (resilience) для сервиса маршрутизации?
6. Приведите пример компромисса (trade-off) между производительностью трекинга и консистентностью.
7. Какой паттерн распределённых транзакций подходит для сценария: «списать деньги с клиента → создать заказ → назначить курьера»?

7. Требования к отчёту

Отчёт должен содержать:

Титульный лист (название работы, ФИО, группа, вариант, сквозной пример «Логистика»).

Цель и задачи работы.

Описание логистической системы (основные функции).

Таблица FURPS+ (минимум 4 требования + 1 из варианта).

Сценарии качества (по одному на каждое требование из п.4).

Таблица «Требование → Архитектурное решение → Обоснование».

Вывод (какие атрибуты качества оказались критичными для логистики, какие архитектурные стили выбраны).

Ответы на контрольные вопросы (письменно).

Формат: **PDF**, шрифт 12 pt Times New Roman, 1.5 интервала.

8. Критерии оценивания

Критерий	max балл

Критерий	max балл
Полнота выделения НФТ (не менее 4 базовых + 1 из варианта)	3
Соответствие требований логистическому домену	2
Корректность сценариев качества (все 5 элементов)	3
Логичность связки «требование → архитектурное решение»	3
Обоснование выбора решения (почему именно Saga/CQRS/ES и т.д.)	3
Качество оформления отчёта, грамотность	1
Ответы на контрольные вопросы	1
Итого	16

Шкала перевода: 14-16 — «отлично», 11-13 — «хорошо», 8-10 — «удовлетворительно», <8 — «неудовлетворительно»

Лабораторная работа №2

Документирование архитектуры: диаграмма контекста (C4 Level 1)

1. Цель работы

Сформировать компетенции по документированию архитектуры программных систем с использованием нотации C4 (уровень 1 - System Context Diagram), включая идентификацию акторов, внешних систем и взаимодействий для выбранного сквозного проекта.

2. Задачи работы

1. Изучить нотацию C4 и правила построения диаграммы контекста (Level 1).
2. Для **своего сквозного проекта** (выбранного в ЛР 1) определить внешних акторов и внешние системы.
3. Построить диаграмму контекста с использованием PlantUML или Structurizr.
4. Описать взаимодействия (протоколы, форматы данных, направления потоков).
5. Оформить результаты в виде отчёта с диаграммой и пояснениями.

3. Теоретическое обоснование

3.1. Нотация C4 (напоминание)

Урове нь	Название	Что показывается
1	System Context	Система, пользователи (акторы), внешние системы
2	Containers	Сервисы, БД, очереди
3	Components	Внутреннее устройство контейнера

Урове нь	Название	Что показывается
4 Level	Code	Классы, интерфейсы

3.2. Правила построения диаграммы контекста

В центре - **одна** разрабатываемая система.

Person - человек или роль (пользователь).

System_Ext - внешняя система (не разрабатываемая, но взаимодействующая).

Каждая связь подписывается: [протокол] действие / данные.

Не показывать внутренние БД, очереди, сервисы.

Обязательно добавить **легенду** (условные обозначения).

3.3. Общие протоколы для разных доменов

Домен	Типичные протоколы
Логистика	WebSocket (GPS), HTTPS/REST (маршруты), FCM (push)
E-Commerce	HTTPS/REST, Kafka (события заказов), SMTP (email)
Финтех	HTTPS/mTLS, ISO 8583, Webhook
Библиотека	HTTPS, REST, RFID (низкоуровневый)
Отели	HTTPS, GraphQL, AMQP

4. Пример выполнения

Сквозной пример (для иллюстрации): «Система управления логистикой и доставкой»

Внимание! Студент выполняет работу для **своего проекта**, выбранного в ЛР 1. Пример ниже - только образец.

4.1. Описание системы (логистика)

1. Система управления логистикой и доставкой:
2. Приём заказов от клиентов
3. Назначение заказа курьеру
4. Трекинг курьеров в реальном времени
5. Построение оптимального маршрута
6. Уведомление клиентов о статусе

4.2. Акторы и внешние системы

Акторы (люди):

1. Курьер (отправляет GPS, получает заказы)
2. Диспетчер (видит карту, назначает заказы)
3. Клиент (отслеживает доставку)
4. **Внешние системы:**
5. Картографический провайдер (OSRM / Google)
6. Push-сервис (Firebase FCM)
7. Платёжный шлюз (Stripe)
8. Складская система (WMS)

4.3. Диаграмма контекста (PlantUML)

```
plantuml
@startuml
!include https://raw.githubusercontent.com/plantuml-stdlib/C4-PlantUML/master/C4_Context.puml
LAYOUT_WITH_LEGEND()

Person(courier, "Курьер", "Доставляет заказы, отправляет GPS")
Person(dispatcher, "Диспетчер", "Управляет курьерами")
```


Person(client, "Клиент", "Отслеживает доставку")

System(logisticsSystem, "Logistics Management System", "Управление доставкой")

System_Ext(mapProvider, "Картографический провайдер", "Расчёт маршрутов")

System_Ext(pushService, "Push-сервис", "Уведомления")

System_Ext(paymentGateway, "Платёжный шлюз", "Оплата")

System_Ext(wms, "Складская система", "Готовность заказов")

Rel(courier, logisticsSystem, "Отправляет GPS, получает заказы", "WebSocket")

Rel(dispatcher, logisticsSystem, "Управляет заказами", "HTTPS")

Rel(client, logisticsSystem, "Отслеживает статус", "HTTPS")

Rel(logisticsSystem, mapProvider, "Расчёт маршрута", "REST")

Rel(logisticsSystem, pushService, "Отправка уведомлений", "FCM")

Rel(logisticsSystem, paymentGateway, "Списание оплаты", "HTTPS")

Rel(logisticsSystem, wms, "Статус заказа", "RabbitMQ")

@enduml

4.4. Таблица взаимодействий

О т кого	К кому	Про токол	Действие	Данные
К	С	Web	Отправка	{lat, lng,

О т кого	К кому	Про токол	Действие	Данные
Курьер	Система	Socket	GPS	courier_id}
Система	Карты	REST	Расчёт маршрута	{origin, destination}
Клиент	Система	HTTP	Запрос статуса	{order_id}

5. Индивидуальные задания (15 вариантов сквозных проектов)

Важно: Каждый студент выполняет работу для того же проекта, который он выбрал в ЛР 1. Ниже приведены 15 сквозных проектов. Студент работает со своим вариантом на протяжении всех ЛР 1–18.

В ариант	Сквозной проект	Краткое описание
1	Система управления логистикой и доставкой	Трекинг курьеров, маршрутизация, назначение заказов
2	Интернет-магазин (E-Commerce)	Каталог, корзина, заказы, оплата, доставка
3	Финтех-кошелёк (Digital Wallet)	Переводы, пополнение, выписки, аудит
4	Система управления	Каталог книг, выдача,

В ариант	Сквозной проект	Краткое описание
	библиотекой	штрафы, бронирование
5	Система бронирования отелей	Поиск, бронирование, оплата, отзывы
6	Платформа онлайн-курсов	Видео, тесты, сертификаты, прогресс
7	Медицинская информационная система	Запись к врачу, электронная карта, рецепты
8	CRM для ритейла	Клиенты, заказы, скидки, коммуникации
9	Платёжный шлюз	Обработка транзакций, фрод-мониторинг, 3DS
10	Система управления складом (WMS)	Приёмка, отгрузка, инвентаризация, интеграция с ERP
11	Социальная сеть	Посты, лента, мессенджер, уведомления
12	Видеохостинг	Загрузка, стриминг, рекомендации, комментарии

В ариант	Сквозной проект	Краткое описание
13	Система мониторинга IoT-устройств	Сбор данных, алармы, дашборды
14	Биржа криптовалют	Ордера, стаканы, сделки, КУС
15	Сервис доставки еды	Заказ, трекинг курьеров, оплата, отзывы

Задание для студента:

Для **своего проекта** (по варианту):

Выделить **не менее 2 акторов** (ролей пользователей) и **не менее 2 внешних систем**.

Построить диаграмму контекста C4 Level 1.

Описать все взаимодействия в таблице.

Обосновать границы системы (что внутри, что снаружи и почему).

6. Контрольные вопросы

1. Что показывает диаграмма контекста (C4 Level 1) и чем она отличается от диаграммы контейнеров?
2. Почему на диаграмме контекста нельзя показывать базы данных?
3. В каком случае внешняя система может стать актором (Person)?
4. Для чего нужна легенда на диаграмме C4?
5. Как на диаграмме контекста отразить, что система поддерживает несколько внешних API одного типа (например, два платёжных шлюза)?
6. Приведите пример компромисса при вынесении функции во внешнюю систему.

7. Какие протоколы вы выбрали для вашего проекта и почему?

7. Требования к отчёту

Отчёт должен содержать:

Титульный лист (название работы, ФИО, группа, вариант, название сквозного проекта).

Цель и задачи работы.

Описание сквозного проекта (из варианта, 5–7 предложений).

Таблица акторов и внешних систем (с обоснованием).

Диаграмма контекста (C4 Level 1) - изображение или код PlantUML.

Таблица взаимодействий (от кого → к кому → протокол → действие → данные).

Обоснование границ системы (что входит в разрабатываемую систему, что вынесено вовне).

Вывод (ключевые внешние зависимости).

Ответы на контрольные вопросы (письменно).

Формат: **PDF**, шрифт 12 pt Times New Roman, 1.5 интервала.

8. Критерии оценивания

Критерий	max балл
Правильность выделения акторов (не менее 2) и внешних систем (не менее 2)	2
Корректность диаграммы C4 Level 1 (синтаксис, элементы, легенда)	4
Наличие подписей на всех связях	2

Критерий	max балл
(протокол/действие)	
Текстовое описание взаимодействий (таблица)	2
Обоснование границ системы (почему внешние системы вынесены)	2
Соответствие выбранному сквозному проекту (варианту)	2
Качество оформления отчёта	1
Ответы на контрольные вопросы	1
Итого	16

Шкала перевода: 14-16 - «отлично», 11-13 - «хорошо», 8-10 - «удовлетворительно», <8 - «неудовлетворительно»

Лабораторная работа №3

Документирование архитектуры: диаграмма контейнеров (C4 Level 2)

1. Цель работы

Сформировать компетенции по документированию архитектуры программных систем на уровне контейнеров (C4 Level 2), включая выделение технологических блоков (сервисов, баз данных, очередей, кэшей),

определение протоколов взаимодействия и технологического стека для выбранного сквозного проекта.

2. Задачи работы

1. Изучить уровень C4 Level 2 (диаграмма контейнеров) и его отличия от Level 1.
2. Для своего сквозного проекта (выбранного в ЛР 1) выделить контейнеры (технологические блоки).
3. Определить технологии для каждого контейнера (язык, фреймворк, БД, брокер и т.д.).
4. Построить диаграмму контейнеров с использованием PlantUML или Structurizr.
5. Показать протоколы взаимодействия между контейнерами (REST, gRPC, AMQP, JDBC, WebSocket и др.).
6. Оформить результаты в виде отчёта с диаграммой и пояснениями.

3. Теоретическое обоснование

3.1. Уровень C4 Level 2: диаграмма контейнеров

Назначение: показать технологические блоки (контейнеры), из которых состоит система, и их взаимодействие.

Что такое контейнер в C4:

Не Docker-контейнер (хотя может им быть), а логическая единица: веб-приложение, база данных, очередь сообщений, кэш, файловое хранилище, мобильное приложение и т.д.

Каждый контейнер имеет свою технологию и отвечает за определённую функцию.

Что показывается:

Контейнеры (сервисы, БД, очереди, кэши)

Взаимодействия между контейнерами (протоколы, API)

Связи с внешними системами (из Level 1)

Что НЕ показывается:

Внутреннее устройство контейнера (классы, компоненты) — это Level 3

Отдельные функции внутри сервиса

3.2. Типичные контейнеры для разных доменов

Тип контейнера	Примеры технологий	Протоколы
Web фронтенд	React, Vue, Angular	HTTPS
Мобильное приложение	iOS (Swift), Android (Kotlin), Flutter	HTTPS, WebSocket
Backend сервис	Spring Boot, Node.js, FastAPI, Go	REST, gRPC, GraphQL
База данных	PostgreSQL, MySQL, MongoDB	JDBC, TCP
Кэш	Redis, Memcached	RESP (Redis protocol)
Очередь сообщений	RabbitMQ, Kafka, AWS SQS	AMQP, Kafka protocol
Хранилище файлов	S3, MinIO, NFS	S3 API, NFS
Event Store	EventStoreDB, Kafka	gRPC, Kafka protocol
Поисковый	Elasticsearch,	REST

Тип контейнера	Примеры технологий	Протоколы
движок	OpenSearch	

3.3. Принципы построения диаграммы контейнеров

Каждый контейнер должен иметь название и технологию (например, «Order Service [Spring Boot]»).

Стрелки подписываются: [протокол] действие (например, [REST] Создать заказ).

Показываются все значимые взаимодействия (синхронные и асинхронные).

Базы данных и очереди — тоже контейнеры.

Внешние системы (из Level 1) можно показать серым цветом или отдельно.

4. Пример выполнения

Сквозной пример (для иллюстрации): «Система управления логистикой и доставкой» (вариант 1)

Студент выполняет работу для своего проекта по своему варианту.

4.1. Описание системы (логистика)

1. Система управления логистикой и доставкой:
2. Приём заказов через API
3. Трекинг курьеров в реальном времени
4. Автоматическое назначение заказа ближайшему курьеру
5. Расчёт маршрутов через внешний картографический провайдер
6. Уведомление клиентов через push

4.2. Выделение контейнеров

Контейнер	Технология	Назначение

Контейнер	Технология	Назначение
Web Dashboard	React + Vite	Интерфейс диспетчера (карта, управление)
Mobile App (Courier)	Flutter / Kotlin	Приложение курьера (GPS, заказы)
API Gateway	Spring Cloud Gateway / Kong	Маршрутизация, аутентификация
Order Service	Spring Boot	Управление заказами
Dispatch Service	Spring Boot	Алгоритм назначения заказов
Tracking Service	Spring Boot + WebSocket	Приём GPS, трансляция позиций
Notification Service	Node.js	Отправка push через FCM
PostgreSQL (Orders)	PostgreSQL 15	Хранение заказов, агрегатов
PostgreSQL (Tracking)	TimescaleDB	Хранение GPS-треков (time-series)
Redis	Redis 7	Кэш позиций

Контейнер	Технология	Назначение
		курьеров, сессии
Kafka	Apache Kafka	События между сервисами (OrderCreated, CourierMoved)
Event Store	EventStoreDB	Хранение событий (Event Sourcing)

4.3. Диаграмма контейнеров (PlantUML)

```
plantuml
```

```
@startuml
```

```
!include https://raw.githubusercontent.com/plantuml-stdlib/C4-PlantUML/master/C4_Container.puml
```

```
LAYOUT_WITH_LEGEND()
```

```
Person(courier, "Курьер", "Мобильное приложение")
```

```
Person(dispatcher, "Диспетчер", "Web-дашборд")
```

```
Person(client, "Клиент", "Веб/мобильное приложение")
```

```
System_Ext(mapProvider, "Картографический провайдер", "OSRM / Google Maps")
```

```
System_Ext(pushService, "Push-сервис", "Firebase FCM")
```

Container(webDashboard, "Web Dashboard", "React + Vite",
"Диспетчерская панель (карта курьеров)")

Container(mobileCourier, "Mobile App (Courier)", "Flutter / Kotlin",
"Приложение курьера")

Container(apiGateway, "API Gateway", "Spring Cloud Gateway",
"Маршрутизация, аутентификация")

Container(orderService, "Order Service", "Spring Boot", "Управление
заказами")

Container(dispatchService, "Dispatch Service", "Spring Boot", "Назначение
заказов курьерам")

Container(trackingService, "Tracking Service", "Spring Boot + WebSocket",
"Приём GPS, трекинг")

Container(notificationService, "Notification Service", "Node.js", "Push-
уведомления")

ContainerDb(orderDb, "Order Database", "PostgreSQL", "Заказы,
агрегаты")

ContainerDb(trackingDb, "Tracking Database", "TimescaleDB", "GPS-
треки (time-series)")

ContainerDb(redis, "Redis Cache", "Redis", "Кэш позиций курьеров,
сессии")

ContainerDb(kafka, "Kafka", "Apache Kafka", "Событийная шина")

ContainerDb(eventStore, "Event Store", "EventStoreDB", "События (Event
Sourcing)")

' Связи: люди → контейнеры

Rel(dispatcher, webDashboard, "Использует", "HTTPS")

Rel(courier, mobileCourier, "Использует", "HTTPS/WSS")

Rel(client, apiGateway, "Отслеживает заказ", "HTTPS")

' Связи: API Gateway → сервисы

Rel(apiGateway, orderService, "Маршрутизирует", "gRPC")

Rel(apiGateway, dispatchService, "Маршрутизирует", "gRPC")

Rel(apiGateway, trackingService, "Маршрутизирует (WebSocket upgrade)", "WS/WSS")

' Связи между сервисами и БД/очередями

Rel(orderService, orderDb, "Читает/пишет", "JDBC")

Rel(orderService, kafka, "Публикует события", "Kafka Protocol")

Rel(dispatchService, kafka, "Подписывается на события", "Kafka Protocol")

Rel(dispatchService, redis, "Читает позиции курьеров", "RESP")

Rel(trackingService, trackingDb, "Сохраняет треки", "JDBC")

Rel(trackingService, redis, "Обновляет кэш позиций", "RESP")

Rel(trackingService, kafka, "Публикует CourierMoved", "Kafka Protocol")

Rel(notificationService, kafka, "Подписывается на события", "Kafka Protocol")

Rel(notificationService, pushService, "Отправляет push", "HTTPS/FCM")

' Связи сервисов с внешними системами

Rel(dispatchService, mapProvider, "Расчёт маршрута/дистанции", "REST")

' Event Store для аудита (опционально)

Rel(orderService, eventStore, "Сохраняет события (ES)", "gRPC")

@enduml

4.4. Таблица контейнеров с технологиями и протоколами

Контейнер	Технология	Протоколы входящие	Протоколы исходящие
Web Dashboard	React + Vite	HTTPS (на пользователя)	HTTPS (к API Gateway)
Mobile App (Courier)	Flutter	HTTPS, WSS	HTTPS, WSS
API Gateway	Spring Cloud Gateway	HTTPS, WSS	gRPC, Kafka
Order Service	Spring Boot	gRPC	JDBC, Kafka, gRPC (ES)
Dispatch Service	Spring Boot	gRPC, Kafka	REST (карты), Kafka, RESP (Redis)
Tracking Service	Spring Boot	WSS, gRPC	JDBC, RESP, Kafka
Notification Service	Node.js	Kafka	HTTPS (FCM)
Order Database	PostgreSQL	JDBC	—
Tracking Database	Timescale DB	JDBC	—

Контейнер	Технология	Протоколы входящие	Протоколы исходящие
Redis Cache	Redis	RESP	—
Kafka	Apache Kafka	Kafka Protocol	Kafka Protocol
Event Store	EventStoreDB	gRPC	—

4.5. Обоснование выбора технологий

Контейнер	Почему выбрано
Spring Boot	Стабильный фреймворк, большая экосистема, интеграция с Kafka и WebSocket
TimescaleDB	Лучше подходит для временных рядов (GPS-треки), чем обычный PostgreSQL
Redis	Быстрый in-memory кэш для позиций курьеров (микросекундная задержка)
Kafka	Асинхронная связь между сервисами, replay событий
WebSocket	Низкая задержка для трекинга GPS (push, а не polling)

5. Индивидуальные задания (15 вариантов сквозных проектов)

Важно: Каждый студент выполняет работу для того же проекта, который он выбрал в ЛР 1 и ЛР 2.

Вариант	Сквозной проект	Особые контейнеры (обязательно включить)
1	Система управления логистикой и доставкой	WebSocket-сервер трекинга, TimescaleDB, Redis
2	Интернет-магазин (E-Commerce)	Сервис каталога, Сервис корзины, Сервис заказов, Redis (сессии)
3	Финтех-кошелёк (Digital Wallet)	Сервис транзакций, Event Store (обязательно), Антифрод-сервис
4	Система управления библиотекой	Сервис каталога, Сервис выдачи, Сервис штрафов, Elasticsearch
5	Система бронирования отелей	Сервис поиска, Сервис бронирования, Сервис оплаты, Kafka
6	Платформа онлайн-курсов	Сервис видео (CDN), Сервис тестов, Сервис прогресса, RabbitMQ

Вариант	Сквозной проект	Особые контейнеры (обязательно включить)
7	Медицинская информационная система	Сервис записи, HL7-адаптер (внешний), БД с шифрованием
8	CRM для ритейла	Сервис клиентов, Сервис заказов, Сервис коммуникаций (email/SMS)
9	Платёжный шлюз	Сервис транзакций, Сервис 3DS, БД с PCI DSS-требованиями
10	Система управления складом (WMS)	Сервис приёмки, Сервис отгрузки, Сервис инвентаризации, интеграция с весами
11	Социальная сеть	Сервис постов, Сервис ленты (таймлайна), Сервис мессенджера, CDN
12	Видеохостинг	Сервис загрузки, Сервис транскодинга, CDN, Сервис рекомендаций (ML)
13	Система мониторинга IoT-	MQTT-брокер, Сервис алармов, InfluxDB, Grafana

Вариант	Сквозной проект	Особые контейнеры (обязательно включить)
	устройств	
14	Биржа криптовалют	Сервис ордеров, Сервис стакана (order book), WebSocket для клиентов
15	Сервис доставки еды	Сервис заказов, Сервис курьеров, Сервис ресторанов, Kafka

Задание для студента:

1. Для своего проекта (по варианту)
2. Выделить не менее 6 контейнеров (включая БД, очереди, кэши).
3. Для каждого указать технологию (конкретную, например, PostgreSQL 15, а не просто «БД»).
4. Построить диаграмму контейнеров C4 Level 2.
5. Описать протоколы взаимодействия между контейнерами.
6. Обосновать выбор ключевых технологий.

6. Контрольные вопросы

1. В чём разница между диаграммой контейнеров (C4 Level 2) и диаграммой контекста (Level 1)?
2. Почему база данных — это контейнер, а не компонент?
3. Как на диаграмме контейнеров показать асинхронное взаимодействие через очередь сообщений?
4. Какие протоколы лучше подходят для синхронного взаимодействия между сервисами, а какие для асинхронного?
5. Что делать, если один и тот же контейнер общается с другим по нескольким протоколам (например, gRPC и Kafka)?
6. Почему для трекинга GPS выбран WebSocket, а не HTTP polling? Как это отражено на диаграмме?

7. Как на диаграмме контейнеров показать, что сервис горизонтально масштабируется (несколько экземпляров)?

7. Требования к отчёту

Отчёт должен содержать:

Титульный лист (название работы, ФИО, группа, вариант, название сквозного проекта).

Цель и задачи работы.

Описание сквозного проекта (из варианта, 5–7 предложений).

Таблица контейнеров (название, технология, назначение).

Диаграмма контейнеров (C4 Level 2) — изображение или код PlantUML.

Таблица протоколов взаимодействия (от контейнера → к контейнеру → протокол → действие).

Обоснование выбора ключевых технологий (3–5 позиций).

Вывод (какие архитектурные стили видны на уровне контейнеров).

Ответы на контрольные вопросы (письменно).

Формат: PDF, шрифт 12 pt Times New Roman, 1.5 интервала.

8. Критерии оценивания

Критерий	max балл
Полнота выделения контейнеров (не менее 6, включая БД/очереди)	2
Корректность указания технологий (конкретные, не общие)	2
Корректность диаграммы C4 Level 2 (синтаксис,	4

Критерий	max балл
обозначения, легенда)	
Наличие подписей на всех связях (протокол/действие)	2
Таблица протоколов взаимодействия (полнота)	2
Обоснование выбора технологий (логистическая/доменная специфика)	2
Соответствие выбранному сквозному проекту (варианту)	1
Качество оформления отчёта	1
Итого	16

Шкала перевода: 14-16 — «отлично», 11-13 — «хорошо», 8-10 — «удовлетворительно», <8 — «неудовлетворительно»

Лабораторная работа №4

Рефакторинг PCMEF → Clean Architecture. Часть 1: выделение слоёв

1. Цель работы

Сформировать компетенции по трансформации традиционной слоистой архитектуры (PCMEF) в Clean Architecture путём выделения независимых

слоёв: Entities, Use Cases, Interface Adapters, Frameworks/Drivers для выбранного сквозного проекта.

2. Задачи работы

1. Изучить структуру Clean Architecture и её отличия от PCMEF.
2. Взять существующий код проекта из дисциплины «Программная инженерия» (PCMEF-архитектура на Spring Boot).
3. Выделить слои Clean Architecture в соответствии с правилами Dependency Rule.
4. Переименовать и реорганизовать пакеты в проекте.
5. Убедиться, что после рефакторинга код продолжает работать (все тесты проходят).
6. Оформить результаты в виде отчёта с описанием изменений.

3. Теоретическое обоснование

3.1. Архитектура PCMEF

PCMEF - это слоистая архитектура, используемая в курсе «Программная инженерия»:

Слой	Назначение	Пример (логистика)
Presentation	UI, контроллеры	CourierController
Control	Бизнес-логика, сервисы	CourierService
Model	Сущности (JPA)	CourierEntity
Entity (Domain)	Доменные сущности (редко)	Courier
Foundation	Репозитории,	CourierRepository

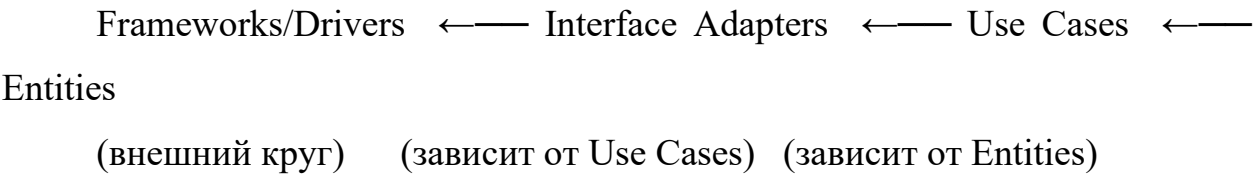
Слой	Назначение	Пример (логистика)
	работа с БД	

Проблемы PCMEF:

1. Зависимости идут сверху вниз (Presentation → Control → Model → Foundation)
2. Жёсткая привязка к фреймворкам (Spring, JPA)
3. Сложность тестирования бизнес-логики без БД
4. Нарушение принципа инверсии зависимостей

3.2. Clean Architecture (Robert C. Martin)

Ключевой принцип: Dependency Rule - зависимости направлены внутрь (к слою Entities).



Слой	Назначение	Что запрещено
Entities	Бизнес-сущности (чистая логика)	Зависимости от фреймворков, БД, UI
Use Cases	Сценарии использования (сервисы)	Зависимости от БД, фреймворков (только интерфейсы)

Слой	Назначение	Что запрещено
Interface Adapters	Контроллеры, репозитории (реализации)	Может зависеть от фреймворков
Frameworks/Drivers	Spring Boot, JPA, БД, Kafka	Самая внешняя часть

3.3. Структура пакетов в Clean Architecture

java

com logistics/

```

├── domain/           // Entities (ядро)
│   ├── model/        // сущности (Courier, Order, Location)
│   ├── valueobject/   // Value Objects (Coordinates, Money)
│   └── repository/    // интерфейсы репозиториев (порт)
├── application/      // Use Cases
│   ├── usecase/       // AssignOrderUseCase, TrackCourierUseCase
│   ├── port/          // порты (входящие/исходящие)
│   └── dto/           // DTO для Use Cases
├── infrastructure/   // Frameworks/Drivers + Adapters
│   ├── web/           // контроллеры (REST)
│   ├── persistence/   // JPA-репозитории, сущности БД
│   ├── messaging/     // Kafka/RabbitMQ
│   └── config/         // Spring конфигурации
└── shared/           // общие утилиты

```

4. Пример выполнения

Сквозной пример (для иллюстрации): «Система управления логистикой и доставкой» (вариант 1)

Студент выполняет работу для своего проекта по своему варианту.

4.1. Исходный код (PCMEF-структура)

До рефакторинга (PCMEF):

```
com.logistics/  
├── controller/  
│   └── CourierController.java    // Presentation  
├── service/  
│   └── CourierService.java      // Control  
├── entity/  
│   └── CourierEntity.java       // Model  
├── domain/  
│   └── Courier.java             // Entity (доменная)  
└── repository/  
    └── CourierRepository.java    // Foundation
```

Пример кода (PCMEF-стиль):

```
// CourierController.java (Presentation)  
  
@RestController  
@RequestMapping("/api/couriers")  
public class CourierController {  
    @Autowired  
    private CourierService courierService; // прямая зависимость  
  
    @PostMapping("/{id}/location")
```



```

        public ResponseEntity<?> updateLocation(@PathVariable Long id,
        @RequestBody LocationDto dto) {
            courierService updateLocation(id, dto);
            return ResponseEntity.ok().build();
        }
    }
}

```

// CourierService.java (Control)

```

@Service
public class CourierService {
    @Autowired
    private CourierRepository repository; // прямая зависимость от JPA

    @Transactional
    public void updateLocation(Long id, LocationDto dto) {
        CourierEntity entity = repository.findById(id).orElseThrow();
        entity.setLatitude(dto.getLat());
        entity.setLongitude(dto.getLng());
        repository.save(entity);
    }
}

```

4.2. Рефакторинг: выделение слоёв Clean Architecture

Шаг 1. Создаём новую структуру пакетов

```

com.logistics/
├── domain/
│   ├── model/
│   │   └── Courier.java

```

```

| |   └── Location.java
| |   └── valueobject/
| |       └── Coordinates.java
| |   └── repository/
| |       └── CourierRepositoryPort.java
| └── application/
|     └── usecase/
|         └── UpdateCourierLocationUseCase.java
|     └── dto/
|         └── UpdateLocationCommand.java
| └── infrastructure/
|     └── web/
|         └── CourierController.java
|     └── persistence/
|         ├── CourierJpaEntity.java
|         └── CourierRepositoryAdapter.java
|     └── config/
|         └── BeanConfig.java

```

Шаг 2. Переносим сущности в слой Domain (Entities)

// domain/model/Courier.java (чистая бизнес-сущность)

`package` com logistics domain model;

`import` com logistics domain valueobject `Coordinates`;

```

public class Courier {
    private Long id;
    private String name;
    private Coordinates currentLocation;
}

```

```

private CourierStatus status;

public void updateLocation(Coordinates newLocation) {
    this.currentLocation = newLocation;
    // бизнес-правило: при обновлении локации сбрасываем статус
    "ожидания"

    if (this.status == CourierStatus.WAITING) {
        this.status = CourierStatus.ACTIVE;
    }
}

// геттеры, конструктор...
}

// domain/valueobject/Coordinates.java (Value Object)
package com.logistics.domain.valueobject;

public record Coordinates(double lat, double lng) {
    public Coordinates {
        if (lat < -90 || lat > 90 || lng < -180 || lng > 180) {
            throw new IllegalArgumentException("Invalid coordinates");
        }
    }
}

public double distanceTo(Coordinates other) {
    // формула гаверсинуса
    return calculateHaversine(this.lat, this.lng, other.lat, other.lng);
}
}

```

Шаг 3. Создаём Use Case (слой Application)

java

```
// application/usecase/UpdateCourierLocationUseCase.java
```

```
package com.logistics.application.usecase;
```

```
import com.logistics.domain.model Courier;
```

```
import com.logistics.domain.repository CourierRepositoryPort;
```

```
import com.logistics.domain.valueobject Coordinates;
```

```
import com.logistics.application.dto.UpdateLocationCommand;
```

```
@Component // или @Service - но это уже деталь фреймворка
```

```
public class UpdateCourierLocationUseCase {
```

```
    private final CourierRepositoryPort courierRepository;
```

```
    public UpdateCourierLocationUseCase(CourierRepositoryPort  
courierRepository) {
```

```
        this.courierRepository = courierRepository;
```

```
    }
```

```
    public void execute(UpdateLocationCommand command) {
```

```
        Courier courier = courierRepository.findById(command.getCourierId())
```

```
        .orElseThrow(() -> new
```

```
CourierNotFoundException(command.getCourierId()));
```

```
        Coordinates newLocation = new Coordinates(command.getLat(),  
command.getLng());
```

```
        courier.updateLocation(newLocation);
```

```
        courierRepository.save(courier);
```

```
}  
}
```

Шаг 4. Создаём порт (интерфейс репозитория) в Domain

```
// domain/repository/CourierRepositoryPort.java
```

```
package com.logistics.domain.repository;
```

```
import com.logistics.domain.model Courier;
```

```
import java.util Optional;
```

```
public interface CourierRepositoryPort {
```

```
    Optional<Courier> findById(Long id);
```

```
    Courier save(Courier courier);
```

```
    void delete(Long id);
```

```
}
```

Шаг 5. Реализуем адаптер в Infrastructure

```
// infrastructure/persistence/CourierJpaEntity.java (отдельная JPA-  
сущность)
```

```
package com.logistics.infrastructure.persistence;
```

```
import jakarta.persistence *;
```

```
import lombok Data;
```

```
@Entity
```

```
@Table(name = "couriers")
```

```
@Data
```

```
public class CourierJpaEntity {
```

```
@Id
@GeneratedValue(strategy = GenerationType.IDENTITY)
private Long id;
private String name;
private double lat;
private double lng;
private String status;
}
```

```
// infrastructure/persistence/CourierRepositoryAdapter.java
```

```
package com.logistics.infrastructure.persistence;
```

```
import com.logistics.domain.model Courier;
import com.logistics.domain.repository.CourierRepositoryPort;
import com.logistics.domain.valueobject Coordinates;
import lombok.RequiredArgsConstructor;
import org.springframework.stereotype.Component;
import java.util Optional;
```

```
@Component
```

```
@RequiredArgsConstructor
```

```
public class CourierRepositoryAdapter implements CourierRepositoryPort {
    private final JpaCourierRepository jpaRepository;
```

```
@Override
```

```
public Optional<Courier> findById(Long id) {
    return jpaRepository.findById(id).map(this::toDomain);
}
```

@Override

```
public Courier save(Courier courier) {  
    CourierJpaEntity entity = toJpa(courier);  
    CourierJpaEntity saved = jpaRepository.save(entity);  
    return toDomain(saved);  
}
```

```
private Courier toDomain(CourierJpaEntity entity) {  
    return new Courier(  
        entity.getId(),  
        entity.getName(),  
        new Coordinates(entity.getLat(), entity.getLng()),  
        CourierStatus.valueOf(entity.getStatus())  
    );  
}
```

```
private CourierJpaEntity toJpa(Courier courier) {  
    CourierJpaEntity entity = new CourierJpaEntity();  
    entity.setId(courier.getId());  
    entity.setName(courier.getName());  
    entity.setLat(courier.getCurrentLocation().lat());  
    entity.setLng(courier.getCurrentLocation().lng());  
    entity.setStatus(courier.getStatus().name());  
    return entity;  
}
```

// infrastructure/persistence/JpaCourierRepository.java (Spring Data)

package com.logistics.infrastructure.persistence;

```
import org.springframework.data.jpa.repository.JpaRepository;

public interface JpaCourierRepository extends
JpaRepository<CourierJpaEntity, Long> {
}
```

Шаг 6. Адаптер контроллера (Interface Adapters)

```
// infrastructure/web/CourierController.java
package com.logistics.infrastructure.web;

import com.logistics.application.dto.UpdateLocationCommand;
import com.logistics.application.usecase.UpdateCourierLocationUseCase;
import lombok.RequiredArgsConstructor;
import org.springframework.web.bind.annotation.*;

@RestController
@RequestMapping("/api/couriers")
@RequiredArgsConstructor
public class CourierController {
    private final UpdateCourierLocationUseCase updateLocationUseCase;

    @PostMapping("/{id}/location")
    public void updateLocation(@PathVariable Long id, @RequestBody
LocationRequest request) {
        UpdateLocationCommand command = new
UpdateLocationCommand(id, request.getLat(), request.getLng());
        updateLocationUseCase.execute(command);
    }
}
```


}

}

4.3. Сравнение архитектур (таблица)

Критерий	PCMEF (было)	Clean Architecture (стало)
Количество слоёв	5 (P, C, M, E, F)	4 (Entities, Use Cases, Adapters, Frameworks)
Направление зависимостей	Сверху вниз ($P \rightarrow C \rightarrow M \rightarrow F$)	Внутрь (к Entities)
Зависимость от Spring	Прямая (@Service, @Autowired)	Только в Infrastructure слое
Зависимость от JPA	Прямая (в сервисах)	Через интерфейс (порт)
Тестируемость бизнес-логики	Требует поднятия контекста Spring	Мокинг портов, JUnit только
Где находится бизнес-правило	В сервисе (Control)	В сущности (Courier.updateLocation)

5. Индивидуальные задания (15 вариантов сквозных проектов)

Важно: Каждый студент выполняет работу для того же проекта, который он выбрал в ЛР 1.

Вариант	Сквозной проект	Какая сущность/Use Case рефакторится
1	Логистика	Courier.updateLocation() + AssignOrderUseCase
2	Интернет-магазин	Product.updateStock() + CreateOrderUseCase
3	Финтех-кошелёк	Wallet.deductFunds() + TransferMoneyUseCase
4	Библиотека	Book.borrow() + BorrowBookUseCase
5	Отели	Room.book() + CreateBookingUseCase
6	Онлайн-курсы	Course.enrollStudent() + EnrollUseCase
7	Медицина	Appointment.schedule() + ScheduleAppointmentUseCase
8	CRM	Customer.addOrder() + CreateCustomerOrderUseCase
9	Платёжный шлюз	Transaction.process() + ProcessPaymentUseCase

Вариант	Сквозной проект	Какая сущность/Use Case рефакторится
10	Склад (WMS)	InventoryItem.reserve() + ReserveItemUseCase
11	Социальная сеть	Post.publish() + PublishPostUseCase
12	Видеохостинг	Video.upload() + UploadVideoUseCase
13	IoT-мониторинг	Device.updateReading() + ProcessReadingUseCase
14	Криптовалюта	OrderBook.match() + MatchOrderUseCase
15	Доставка еды	Restaurant.acceptOrder() + AcceptOrderUseCase

Задание для студента:

Для своего проекта (по варианту):

1. Взять существующий РСМЕФ-код из «Программной инженерии» (или создать упрощённую версию).
2. Выделить 4 слоя Clean Architecture.
3. Переименовать пакеты в соответствии со структурой.
4. Перенести бизнес-правило из сервиса в доменную сущность (Entity).
5. Создать интерфейс порта (репозитория) в слое Domain.
6. Реализовать адаптер в Infrastructure.
7. Убедиться, что код компилируется и (при наличии) тесты проходят.

6. Контрольные вопросы

1. В чём главное отличие Clean Architecture от PCMEF по направлению зависимостей?
2. Почему слой Entities не должен зависеть от JPA или Spring?
3. Где в Clean Architecture размещаются интерфейсы репозитория (порты), а где их реализации?
4. Как бизнес-правило «курьер не может сменить статус, если он уже доставил заказ» должно быть реализовано: в сущности или в Use Case?
5. Что произойдёт, если Use Case напрямую вызовет JPA-репозиторий? Нарушится ли Dependency Rule?
6. Как протестировать Use Case без поднятия Spring-контекста?
7. В каком слое должен находиться @Transactional и почему?

7. Требования к отчёту

Отчёт должен содержать:

Титульный лист (название работы, ФИО, группа, вариант, сквозной проект).

Цель и задачи работы.

Описание исходной PCMEF-структуры (пакеты, зависимости, код до рефакторинга).

Новая структура пакетов Clean Architecture (диаграмма или дерево).

Код ключевых классов (сущность, Use Case, порт, адаптер) - фрагменты.

Таблица сравнения PCMEF vs Clean Architecture (для своего проекта).

Вывод (что изменилось, какие преимущества получены).

Ответы на контрольные вопросы (письменно).

Формат: PDF, шрифт 12 pt Times New Roman, 1.5 интервала.

Приложение: архив с исходным кодом до и после рефакторинга (по желанию).

8. Критерии оценивания

Критерий	max балл
Правильность выделения 4 слоёв Clean Architecture	2
Переименование пакетов в соответствии с Clean Architecture	2
Вынос бизнес-правила из сервиса в доменную сущность	3
Создание порта (интерфейса репозитория) в Domain	2
Реализация адаптера в Infrastructure (маппинг JPA ↔ Domain)	3
Отсутствие прямых зависимостей Use Cases от фреймворков	2
Соответствие выбранному сквозному проекту (варианту)	1
Качество оформления отчёта	1
Итого	16

Шкала перевода: 14-16 - «отлично», 11-13 - «хорошо», 8-10 - «удовлетворительно», <8 - «неудовлетворительно»

Лабораторная работа №5

Рефакторинг PCMEF → Clean Architecture. Часть 2: Dependency Inversion (инверсия зависимостей)

1. Цель работы

Сформировать компетенции по обеспечению независимости бизнес-логики от фреймворков и инфраструктуры через применение принципа инверсии зависимостей (Dependency Inversion Principle) в Clean Architecture для выбранного сквозного проекта.

2. Задачи работы

1. Изучить принцип Dependency Inversion (DIP) и его роль в Clean Architecture.
2. В проекте (после ЛР 4) заменить прямые вызовы БД и внешних сервисов на интерфейсы (порты).
3. Вынести интерфейсы репозитория в слой Domain (Entities).
4. Реализовать Dependency Injection через конструктор (без @Autowired над полями).
5. Обеспечить, чтобы Use Cases не зависели от Spring, JPA, Kafka и других фреймворков напрямую.
6. Написать модульные тесты для Use Cases с использованием моков (Mockito).
7. Оформить результаты в виде отчёта.

3. Теоретическое обоснование

3.1. Принцип инверсии зависимостей (DIP)

Формулировка (Robert C. Martin):

Модули верхних уровней не должны зависеть от модулей нижних уровней. Оба типа модулей должны зависеть от абстракций.

Абстракции не должны зависеть от деталей. Детали должны зависеть от абстракций.

В контексте Clean Architecture:

Что зависит от чего	Направление
Use Cases (высокий уровень)	Зависят от интерфейсов (портов)
Репозитории (низкий уровень)	Реализуют эти интерфейсы
Контроллеры	Зависят от Use Cases (через интерфейсы)

3.2. Проблема прямых зависимостей (до рефакторинга)

```
// ПЛОХО: Use Case напрямую зависит от JPA-репозитория
@Service
public class AssignOrderUseCase {
    @Autowired
    private JpaCourierRepository courierRepository; // прямая зависимость
от JPA

    @Autowired
    private JpaOrderRepository orderRepository; // прямая зависимость
от JPA

    public void execute(Long orderId) {
        // бизнес-логика, перемешанная с JPA
    }
}
```

Проблемы:

Невозможно тестировать бизнес-логику без БД

Смена БД требует изменения кода Use Case

Нарушение Dependency Rule

3.3. Решение: порты и адаптеры (Hexagonal Architecture)

// ХОРОШО: Use Case зависит от интерфейса (порта)

@Component

public class AssignOrderUseCase {

private final CourierRepositoryPort courierRepository; // интерфейс из
Domain

private final OrderRepositoryPort orderRepository; // интерфейс из
Domain

public AssignOrderUseCase(CourierRepositoryPort courierRepository,
OrderRepositoryPort orderRepository) {

this.courierRepository = courierRepository;

this.orderRepository = orderRepository;

}

public void execute(AssignOrderCommand command) {

// ТОЛЬКО бизнес-логика

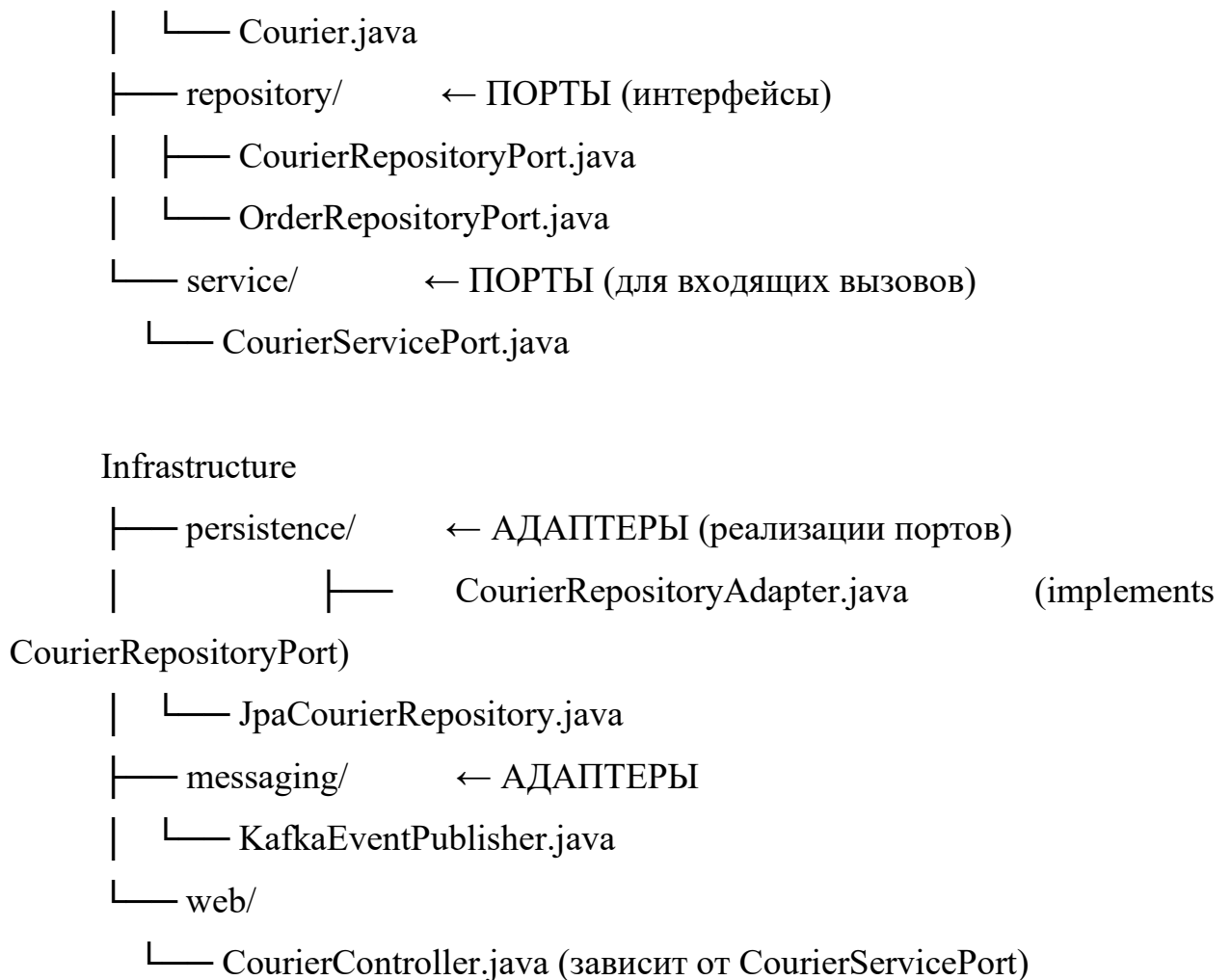
}

}

3.4. Структура портов и адаптеров

Domain (ядро)

└─ model/



4. Пример выполнения

Сквозной пример (для иллюстрации): «Система управления логистикой и доставкой» (вариант 1)

Студент выполняет работу для своего проекта по своему варианту.

4.1. Исходный код (после ЛР 4) — ещё есть прямые зависимости

// application/usecase/AssignOrderUseCase.java (после ЛР 4)

`package` com logistics application usecase;

`import` com logistics domain model Courier;

`import` com logistics domain model Order;

```
import com logistics infrastructure persistence JpaCourierRepository;    //
```

ПРЯМАЯ ЗАВИСИМОСТЬ!

```
import com logistics infrastructure persistence JpaOrderRepository;    //
```

ПРЯМАЯ ЗАВИСИМОСТЬ!

```
import org springframework stereotype Component;
```

```
@Component
```

```
public class AssignOrderUseCase {
```

```
    private final JpaCourierRepository courierRepository; // проблема
```

```
    private final JpaOrderRepository orderRepository;    // проблема
```

```
    // Dependency Injection через конструктор (хорошо), но тип —  
    конкретный класс (плохо)
```

```
    public AssignOrderUseCase(JpaCourierRepository courierRepository,  
                              JpaOrderRepository orderRepository) {
```

```
        this.courierRepository = courierRepository;
```

```
        this.orderRepository = orderRepository;
```

```
    }
```

```
    public void execute(Long orderId) {
```

```
        Order order = orderRepository findById(orderId)
```

```
        .orElseThrow(() -> new RuntimeException("Order not found"));
```

```
        Courier nearest =
```

```
        courierRepository.findNearest(order.getPickupLocation())
```

```
        .orElseThrow(() -> new RuntimeException("No courier available"));
```

```
        order.assignToCourier(nearest);
```

```
        orderRepository.save(order);
```

```
        courierRepository save(nearest);  
    }  
}
```

4.2. Рефакторинг: создание портов в слое Domain

Шаг 1. Создаём интерфейсы портов в Domain

```
// domain/repository/CourierRepositoryPort.java
```

```
package com.logistics.domain.repository;
```

```
import com.logistics.domain.model Courier;
```

```
import com.logistics.domain.valueobject Coordinates;
```

```
import java.util Optional;
```

```
public interface CourierRepositoryPort {  
    Optional<Courier> findById(Long id);  
    Optional<Courier> findNearest(Coordinates location);  
    Courier save(Courier courier);  
    void delete(Long id);  
    boolean existsAvailable();  
}
```

```
// domain/repository/OrderRepositoryPort.java
```

```
package com.logistics.domain.repository;
```

```
import com.logistics.domain.model Order;
```

```
import java.util Optional;
```

```
public interface OrderRepositoryPort {
```

```

Optional<Order> findById(Long id);
Order save(Order order);
void delete(Long id);
}

```

Шаг 2. Создаём Use Case, зависящий только от портов

```

// application/usecase/AssignOrderUseCase.java (рефакторинг)
package com.logistics.application.usecase;

import com.logistics.domain.model Courier;
import com.logistics.domain.model Order;
import com.logistics.domain.repository CourierRepositoryPort;           //
интерфейс
import com.logistics.domain.repository OrderRepositoryPort;           //
интерфейс
import com.logistics.application.dto AssignOrderCommand;
import com.logistics.application.exception CourierNotFoundException;
import com.logistics.application.exception OrderNotFoundException;

@Component // можно убрать @Component, добавить в конфигурацию
позже

public class AssignOrderUseCase {
    private final CourierRepositoryPort courierRepository;
    private final OrderRepositoryPort orderRepository;

    // Инверсия: зависим от абстракций, а не от конкретных реализаций
    public AssignOrderUseCase(CourierRepositoryPort courierRepository,
                              OrderRepositoryPort orderRepository) {
        this.courierRepository = courierRepository;
    }
}

```

```

    this orderRepository = orderRepository;
}

public void execute(AssignOrderCommand command) {
    // 1. Загружаем заказ через порт
    Order order = orderRepository findById(command getOrderId())
        .orElseThrow(() -> new
OrderNotFoundException(command getOrderId()));

    // 2. Проверяем бизнес-правила (в домене)
    if (!order isPending()) {
        throw new IllegalStateException("Order is not pending");
    }

    // 3. Ищем ближайшего курьера
    Courier nearest =
courierRepository.findNearest(order getPickupLocation())
        .orElseThrow(() -> new CourierNotFoundException("No courier
available"));

    // 4. Бизнес-логика назначения (в доменной сущности)
    order assignToCourier(nearest);
    nearest assignOrder(order);

    // 5. Сохраняем через порты
    orderRepository.save(order);
    courierRepository.save(nearest);
}
}

```

Шаг 3. Реализуем адаптеры в Infrastructure

```
// infrastructure/persistence/CourierRepositoryAdapter.java
package com.logistics.infrastructure.persistence;

import com.logistics.domain.model Courier;
import com.logistics.domain.repository CourierRepositoryPort;
import com.logistics.domain.valueobject Coordinates;
import lombok.RequiredArgsConstructor;
import org.springframework.stereotype.Component;
import java.util Optional;

@Component // реализация порта
@RequiredArgsConstructor

public class CourierRepositoryAdapter implements CourierRepositoryPort {

    private final JpaCourierRepository jpaRepository;

    @Override
    public Optional<Courier> findById(Long id) {
        return jpaRepository.findById(id).map(this::toDomain);
    }

    @Override
    public Optional<Courier> findNearest(Coordinates location) {
        // реальная реализация с гео-запросом
        return jpaRepository.findNearest(location.lat(), location.lng())
            .map(this::toDomain);
    }
}
```

@Override

```
public Courier save(Courier courier) {  
    CourierJpaEntity entity = toJpa(courier);  
    CourierJpaEntity saved = jpaRepository.save(entity);  
    return toDomain(saved);  
}
```

@Override

```
public void delete(Long id) {  
    jpaRepository.deleteById(id);  
}
```

@Override

```
public boolean existsAvailable() {  
    return jpaRepository.countByStatus("ACTIVE") > 0;  
}
```

```
private Courier toDomain(CourierJpaEntity entity) {  
    return new Courier(  
        entity.getId(),  
        entity.getName(),  
        new Coordinates(entity.getLat(), entity.getLng()),  
        CourierStatus.valueOf(entity.getStatus())  
    );  
}
```

```
private CourierJpaEntity toJpa(Courier courier) {  
    CourierJpaEntity entity = new CourierJpaEntity();  
    entity.setId(courier.getId());  
}
```

```

        entity.setName(courier.getName());
        entity.setLat(courier.getCurrentLocation().lat());
        entity.setLng(courier.getCurrentLocation().lng());
        entity.setStatus(courier.getStatus().name());
        return entity;
    }
}

```

// infrastructure/persistence/JpaCourierRepository.java

package com logistics infrastructure persistence;

```

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 JpaCourierRepository extends
JpaRepository<CourierJpaEntity, Long> {

    @Query(value = "SELECT * FROM couriers WHERE status = 'ACTIVE'
" +
        "ORDER BY (lat - :lat)^2 + (lng - :lng)^2 LIMIT 1", nativeQuery =
true)
    Optional<CourierJpaEntity> findNearest(@Param("lat") double lat,
@Param("lng") double lng);

    long countByStatus(String status);
}

```

Шаг 4. Конфигурация Spring (явное связывание портов и адаптеров)


```
// infrastructure/config/BeanConfig.java
package com logistics infrastructure config;

import com logistics application usecase AssignOrderUseCase;
import com logistics domain repository CourierRepositoryPort;
import com logistics domain repository OrderRepositoryPort;
import org springframework context annotation Bean;
import org springframework context annotation Configuration;

@Configuration
public class BeanConfig {

    @Bean
    public AssignOrderUseCase assignOrderUseCase(
        CourierRepositoryPort courierRepository,
        OrderRepositoryPort orderRepository) {
        return new AssignOrderUseCase(courierRepository, orderRepository);
    }
}
```

4.3. Модульное тестирование Use Cases с моками

```
// application/usecase/AssignOrderUseCaseTest.java
package com logistics application usecase;

import com logistics domain model Courier;
import com logistics domain model Order;
import com logistics domain repository CourierRepositoryPort;
import com logistics domain repository OrderRepositoryPort;
```

```
import com logistics domain valueobject Coordinates;
import com logistics application dto AssignOrderCommand;
import org junit jupiter api BeforeEach;
import org junit jupiter api Test;
import org junit jupiter api extension ExtendWith;
import org mockito Mock;
import org mockito junit jupiter MockitoExtension;
```

```
import java util Optional;
```

```
import static org assertj core api Assertions assertThatThrownBy;
import static org mockito ArgumentMatchers any;
import static org mockito Mockito.*;
```

```
@ExtendWith(MockitoExtension class)
```

```
class AssignOrderUseCaseTest {
```

```
    @Mock
```

```
    private CourierRepositoryPort courierRepository;
```

```
    @Mock
```

```
    private OrderRepositoryPort orderRepository;
```

```
    private AssignOrderUseCase useCase;
```

```
    @BeforeEach
```

```
    void setUp() {
```

```
        useCase = new AssignOrderUseCase(courierRepository,
orderRepository);
```

```
}
```

```
@Test
```

```
void shouldAssignOrderToNearestCourier() {
```

```
    // given
```

```
    Long orderId = 1L;
```

```
    Order order = new Order(orderId, new Coordinates(55.75, 37.62));
```

```
    order.setPending(true);
```

```
    Courier courier = new Courier(1L, "John", new Coordinates(55.76,  
37.63), CourierStatus.ACTIVE);
```

```
    when(orderRepository.findById(orderId)).thenReturn(Optional.of(order));
```

```
    when(courierRepository.findNearest(any(Coordinates.class))).thenReturn(Optional  
.of(courier));
```

```
    when(orderRepository.save(any(Order.class))).thenReturn(order);
```

```
    when(courierRepository.save(any(Courier.class))).thenReturn(courier);
```

```
    // when
```

```
    useCase.execute(new AssignOrderCommand(orderId));
```

```
    // then
```

```
    verify(orderRepository).save(order);
```

```
    verify(courierRepository).save(courier);
```

```
    verify(orderRepository, times(1)).findById(orderId);
```

```
    verify(courierRepository,  
times(1)).findNearest(any(Coordinates.class));
```

```
}
```

```
@Test
```

```
void shouldThrowExceptionWhenOrderNotFound() {
```

```
    // given
```

```
    Long orderId = 999L;
```

```
    when(orderRepository.findById(orderId)).thenReturn(Optional.empty());
```

```
    // when & then
```

```
    assertThatThrownBy(() -> useCase.execute(new  
AssignOrderCommand(orderId)))  
        .isInstanceOf(OrderNotFoundException.class);
```

```
    verify(orderRepository, never()).save(any());
```

```
    verify(courierRepository, never()).findNearest(any());
```

```
}
```

```
@Test
```

```
void shouldThrowExceptionWhenNoCourierAvailable() {
```

```
    // given
```

```
    Long orderId = 1L;
```

```
    Order order = new Order(orderId, new Coordinates(55.75, 37.62));
```

```
    order.setPending(true);
```

```
    when(orderRepository.findById(orderId)).thenReturn(Optional.of(order));
```

```

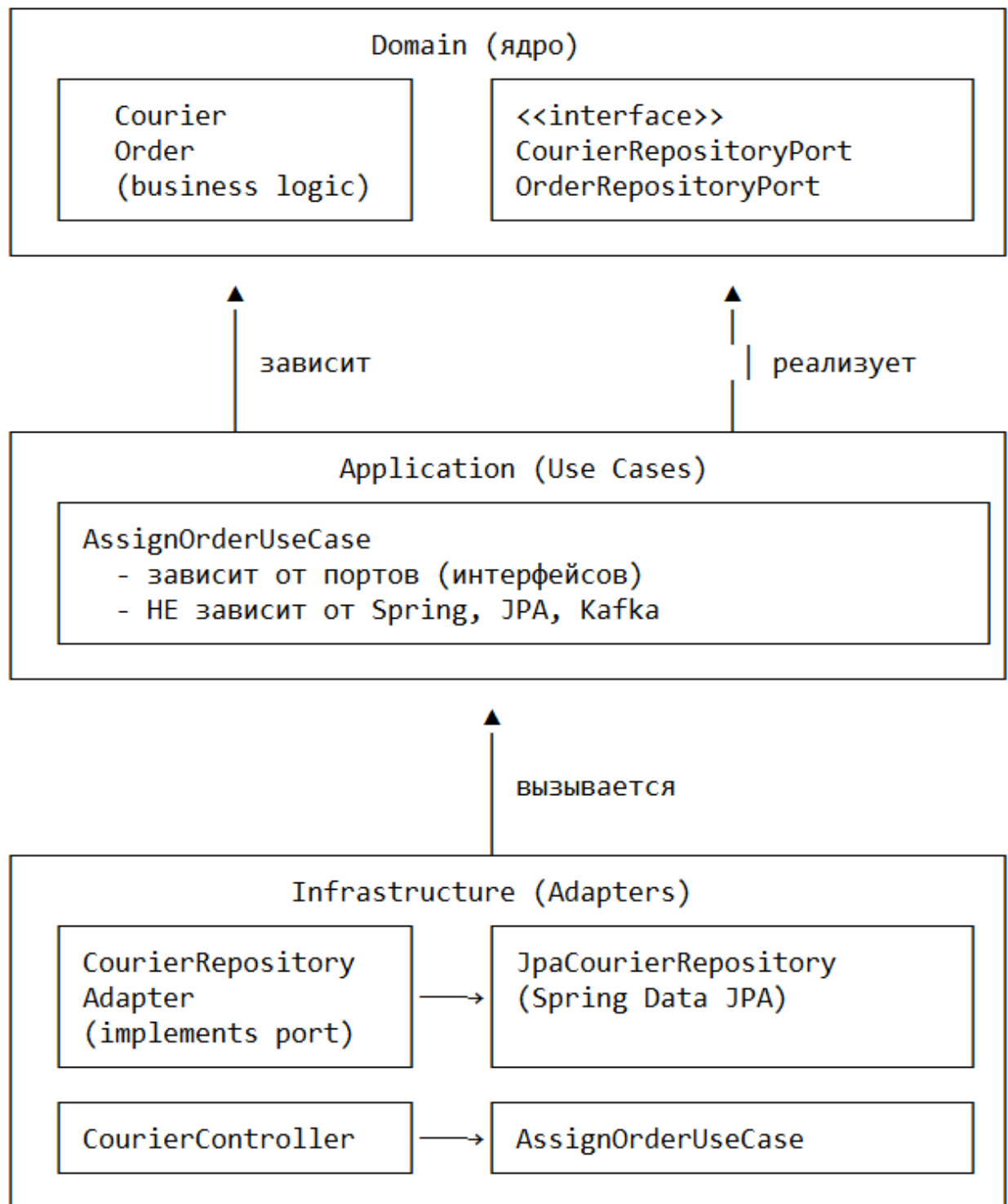
when(courierRepository.findNearest(any(Coordinates class))).thenReturn(Optional
.empty());

// when & then
assertThatThrownBy(() -> useCase execute(new
AssignOrderCommand(orderId)))
.isInstanceOf(CourierNotFoundException class);

verify(orderRepository, never()).save(any());
verify(courierRepository, never()).save(any());
}
}

```

4.4. Диаграмма зависимостей после рефакторинга



4.5. Проверка независимости от фреймворков

Критерии успешного рефакторинга:

Проверка	Результат

Проверка	Результат
Use Case не импортирует org.springframework.*	Да
Use Case не импортирует jakarta.persistence.*	Да
Use Case не импортирует org.apache.kafka.*	Да
Use Case можно запустить в plain Java (без Spring)	Да
Модульные тесты не требуют @SpringBootTest	Да

5. Индивидуальные задания (15 вариантов сквозных проектов)

Важно: Каждый студент выполняет работу для того же проекта, который он выбрал в ЛР 1–4.

Вариант	Сквозной проект	UseCase для рефакторинга	Какие порты создать
1	Логистика	AssignOrderUseCase	CourierRepositoryPort, OrderRepositoryPort
2	Интернет-магазин	CreateOrderUseCase	ProductRepositoryPort, CartRepositoryPort, OrderRepositoryPort
3	Финтех-кошелёк	TransferMoneyUseCase	WalletRepositoryPort, TransactionRepositoryPort

Вариант	Сквозной проект	UseCase для рефакторинга	Какие порты создать
			Port
4	Библиотека	BorrowBookUseCase	BookRepositoryPort, ReaderRepositoryPort, LoanRepositoryPort
5	Отели	CreateBookingUseCase	RoomRepositoryPort, BookingRepositoryPort, PaymentRepositoryPort
6	Онлайн-курсы	EnrollStudentUseCase	CourseRepositoryPort, StudentRepositoryPort, EnrollmentRepositoryPort
7	Медицина	ScheduleAppointmentUseCase	DoctorRepositoryPort, PatientRepositoryPort, AppointmentRepositoryPort
8	CRM	CreateCustomerOrderUseCase	CustomerRepositoryPort, OrderRepositoryPort,

Вариант	Сквозной проект	UseCase для рефакторинга	Какие порты создать
			ProductRepositoryPort
9	Платёжный шлюз	ProcessPaymentUse Case	TransactionRepositoryPort, MerchantRepositoryPort
10	Склад (WMS)	ReserveItemUse Case	InventoryRepositoryPort, ReservationRepositoryPort
11	Социальная сеть	PublishPostUse Case	PostRepositoryPort, UserRepositoryPort, FeedRepositoryPort
12	Видео хостинг	UploadVideoUse Case	VideoRepositoryPort, UserRepositoryPort, CdnPort
13	IoT-мониторинг	ProcessReadingUse Case	DeviceRepositoryPort, ReadingRepositoryPort, AlertPort
14	Криптовалюта	MatchOrderUse Case	OrderBookRepositoryPort, TradeRepositoryPort

Вариант	Сквозной проект	UseCase для рефакторинга	Какие порты создать
15	Доставка еды	AcceptOrderUseCase	RestaurantRepositoryPort, OrderRepositoryPort, CourierRepositoryPort

Задание для студента:

Для своего проекта (по варианту):

1. Выделить интерфейсы портов (репозиториев) в слой Domain.
2. Переписать Use Case, чтобы он зависел только от портов (не от конкретных реализаций).
3. Реализовать адаптеры в Infrastructure (JPA-реализации портов).
4. Настроить Dependency Injection через конструктор.
5. Написать минимум 3 модульных теста для Use Case с использованием Mockito.
6. Убедиться, что Use Case не имеет прямых зависимостей от Spring/JPA.
7. Зафиксировать результаты в отчёте.

6. Контрольные вопросы

1. Сформулируйте принцип Dependency Inversion (DIP) своими словами.
2. Чем отличается Dependency Inversion от Dependency Injection?
3. Почему Use Case не должен напрямую вызывать JpaRepository?
4. Где в Clean Architecture размещаются интерфейсы портов, а где их реализации?
5. Как протестировать Use Case без поднятия Spring-контекста? Приведите пример.
6. Что такое «адаптер» в контексте Clean/Hexagonal Architecture?
7. Можно ли использовать @Autowired в Use Case после рефакторинга? Почему?
8. Какой паттерн используется для преобразования между JPA-сущностью и доменной сущностью?

7. Требования к отчёту

Отчёт должен содержать:

Титульный лист (название работы, ФИО, группа, вариант, сквозной проект).

Цель и задачи работы.

Описание Use Case до рефакторинга (код с прямыми зависимостями).

Интерфейсы портов (код из слоя Domain).

Use Case после рефакторинга (зависит только от портов).

Реализация адаптеров (код из Infrastructure).

Модульные тесты (3 примера с Mockito).

Диаграмма зависимостей (схема портов и адаптеров).

Таблица проверки независимости (отсутствие импортов фреймворков в Use Case).

Вывод (что дала инверсия зависимостей).

Ответы на контрольные вопросы.

Формат: PDF, шрифт 12 pt Times New Roman, 1.5 интервала.

Приложение: ссылка на репозиторий с кодом (по желанию).

8. Критерии оценивания

Критерий	max балл
Создание интерфейсов портов в слое Domain (минимум 2)	2
Use Case зависит только от портов (нет прямых зависимостей от JPA/Spring)	3
Реализация адаптеров в Infrastructure	2

Критерий	max балл
(маппинг JPA ↔ Domain)	
Dependency Injection через конструктор (без @Autowired над полями)	2
Модульные тесты с Mockito (минимум 3 теста, работают без Spring)	3
Отсутствие фреймворк-специфичных импортов в Application/Domain	2
Соответствие выбранному сквозному проекту (варианту)	1
Качество оформления отчёта	1
Итого	16

Шкала перевода: 14-16 — «отлично», 11-13 — «хорошо», 8-10 — «удовлетворительно», <8 — «неудовлетворительно»

Лабораторная работа №6 Сравнение PCMEF и Clean Architecture

1. Цель работы

Сформировать компетенции по сравнительному анализу архитектурных подходов (PCMEF и Clean Architecture), оценке их применимости для конкретного домена и обоснованному выбору архитектурного стиля для будущего развития выбранного сквозного проекта.

2. Задачи работы

1. Изучить ключевые характеристики PCMEF и Clean Architecture.
2. Заполнить таблицу сравнения по заданным критериям для своего сквозного проекта.
3. Выявить преимущества и недостатки каждого подхода в контексте выбранного домена.
4. Выбрать архитектурный подход для дальнейшего развития проекта, обосновать выбор.
5. Оформить результаты в виде аналитического отчёта.

3. Теоретическое обоснование

3.1. Сравнительная характеристика архитектур

Критерий	PCMEF	Clean Architecture
Происхождение	Учебная архитектура (курс «Программная инженерия»)	Robert C. Martin (Uncle Bob), 2012
Количество слоёв	5 (Presentation, Control, Model, Entity, Foundation)	4 (Entities, Use Cases, Interface Adapters, Frameworks/Drivers)
Направление зависимостей	Сверху вниз ($P \rightarrow C \rightarrow M \rightarrow E \rightarrow F$)	Внутрь (к Entities)
Зависимость от фреймворков	Высокая (Spring, JPA пронизывают все слои)	Низкая (фреймворки только во внешнем слое)
Тестируемость	Требует поднятия	Чистые unit-

Критерий	PCMEF	Clean Architecture
бизнес-логики	Spring-контекста	тесты (без фреймворков)
Сложность	Низкая (быстрый старт)	Средняя/Высокая (больше абстракций)
Скорость разработки (первые недели)	Высокая	Низкая
Скорость разработки (долгосрочно)	Снижается из-за связанности	Высокая (независимость)
Поддерживаемость	Средняя	Высокая
Смена БД	Трудоёмко (код везде зависит от JPA)	Легко (через порты/адаптеры)
Смена UI (например, с Web на Mobile)	Трудоёмко (логика в контроллерах)	Легко (UI — внешний адаптер)

3.2. Диаграмма направлений зависимостей

PCMEF (зависимости сверху вниз):

Presentation (контроллеры)

↓ зависит

Control (сервисы)

↓ зависит

Model (JPA-сущности)

↓ зависит

Entity (доменные сущности — часто пустые)

↓ зависит

Foundation (репозитории)

Clean Architecture (зависимости внутрь):

Frameworks/Drivers (Spring, JPA, Kafka)

↑ зависит (реализует)

Interface Adapters (контроллеры, репозитории-адаптеры)

↑ зависит (реализует)

Use Cases (сервисы бизнес-логики)

↑ зависит

Entities (чистые бизнес-сущности)

3.3. Ключевые отличия в организации кода

Аспект	PCMEF	Clean Architecture
Где находится бизнес-правило?	В Control (сервисе)	В Entity или Use Case
Как контроллер вызывает логику?	<code>courierService.updateLocation(...)</code>	<code>updateLocationUseCase.execute(...)</code>

Аспект	PCMEF	Clean Architecture
Как репозиторий объявлен?	interface JpaCourierRepository extends JpaRepository	interface CourierRepositoryPort (в Domain)
Где реализация репозитория?	В Foundation (JPA)	В Infrastructure (Adapter)
DTO/Command объекты	Часто отсутствуют (прямая передача Entity)	Явные Command/Query объекты
Транзакции	@Transactional на методах сервиса	@Transactional на адаптерах репозитория

4. Пример выполнения

Сквозной пример (для иллюстрации): «Система управления логистикой и доставкой» (вариант 1)

Студент выполняет работу для своего проекта по своему варианту.

4.1. Таблица сравнения для логистики

Критерий	PCMEF (было)	Clean Architecture (стало)
Количество слоёв	5 (P, C, M, E, F)	4 (Entities, Use Cases, Adapters, Frameworks)
Направление	Контроллер → Сервис → JPA-сущность	Внутри: Adapters → Use Cases → Entities

Критерий	PCMEF (было)	Clean Architecture (стало)
зависимостей	→ Репозиторий	
Независимость от фреймворков	Нет (Spring везде: @Service, @Autowired, @Transactional)	Да (Use Cases не знают о Spring)
Тестируемость	Средняя (требуется @SpringBootTest, БД в памяти)	Отличная (чистые unit-тесты с моками)
Сложность	Низкая (понятно новичкам)	Средняя (требуется понимание DIP)
Где бизнес-правило «курьер не может получить заказ, если он уже занят»?	В CourierService.assignOrder()	В методе Courier.assignOrder() (Entity)
Смена картографического провайдера	Трудно (код расчёта дистанции в сервисе)	Легко (замена адаптера MapProviderPort)
Смена БД (PostgreSQL →	Трудно (везде JPA)	Легко (новый адаптер репозитория)

Критерий	PCMEF (было)	Clean Architecture (стало)
MongoDB)		
Дублирование кода	Часто (логика в разных сервисах)	Минимально (Use Cases переиспользуются)
Порог входа для разработчика	Низкий	Средний (нужно понять порты/адаптеры)

4.2. Пример кода: сравнение на фрагменте

PCMEF-стиль (логистика):

```

@Service
public class CourierService {
    @Autowired
    private JpaCourierRepository courierRepository; // прямая зависимость

    @Transactional
    public void updateLocation(Long id, double lat, double lng) {
        CourierEntity entity = courierRepository.findById(id).orElseThrow();
        entity.setLat(lat);
        entity.setLng(lng);
        courierRepository.save(entity);
        // бизнес-правило потеряно в контроллере
    }
}

```

Clean Architecture (после ЛР 4–5):

// Domain Entity — бизнес-правило здесь

```
public class Courier {  
    private Coordinates location;  
  
    public void updateLocation(Coordinates newLocation) {  
        if (newLocation distanceTo(this location) > 500) {  
            throw new IllegalStateException("Jump too large, possible GPS  
error");  
        }  
        this.location = newLocation;  
    }  
}
```

// Use Case — оркестрация

@Component

```
public class UpdateLocationUseCase {  
    private final CourierRepositoryPort repository; // зависит от порта  
  
    public void execute(UpdateLocationCommand command) {  
        Courier courier = repository.findById(command.getCourierId())  
            .orElseThrow();  
        courier.updateLocation(new Coordinates(command.getLat(),  
command.getLng()));  
        repository.save(courier);  
    }  
}
```

4.3. Анализ применимости для логистики

Фактор	Оценка PCMEF	Оценка Clean Architecture	Обоснование
Сложность домена	Средняя	Высокая	Логистика имеет сложные бизнес-правила (назначение, маршрутизация) → Clean Architecture лучше изолирует их
Частота изменений	Средняя	Высокая	Требования к алгоритмам назначения часто меняются → Clean Architecture позволяет тестировать без БД
Интеграции	Средняя	Высокая	Много внешних API (карты, push, оплата) → порты/адаптеры упрощают замену
Размер команды	Низкая	Средняя	При 3+ разработчиках Clean Architecture даёт чёткие границы
Время до первого релиза	Низкая	Высокая (больше)	Прототип быстрее на PCMEF

Фактор	Оценка PCMEF	Оценка Clean Architecture	Обоснование
	(быстрый старт)	абстракций)	
Долгосрочная поддержка	Средняя	Высокая	Clean Architecture окупается через 6-12 месяцев

4.4. Выбор архитектуры для логистики

Решение: Clean Architecture (с элементами DDD)

Обоснование:

Сложные бизнес-правила: логистика имеет нетривиальную логику назначения заказов, расчёта ETA, обработки геозон.

Множество внешних интеграций: карты, push-уведомления, WMS, платёжные системы — порты/адаптеры позволяют легко их заменять.

Высокие требования к тестируемости: критично тестировать сценарии назначения без реальной БД и карт.

Долгосрочная эволюция: планируется переход к микросервисам (Strangler pattern) — Clean Architecture упрощает вынос Use Cases в отдельные сервисы.

Аудит и Event Sourcing: в логистике важно сохранять историю перемещений — Clean Architecture легко интегрируется с ES.

Компромиссы:

Более медленный старт (первые 2 недели)

Больше шаблонного кода (порты, адаптеры, Command/Query объекты)

Требуется обучение команды принципам DIP

5. Индивидуальные задания (15 вариантов сквозных проектов)

Важно: Каждый студент выполняет работу для того же проекта, который он выбрал в ЛР 1–5.

Вариант	Сквозной проект	Ключевые факторы для анализа
1	Логистика	Сложность маршрутизации, интеграция карт, Event Sourcing для треков
2	Интернет-магазин	Высокая нагрузка на каталог, частые акции, интеграция с платёжками
3	Финтех-кошелёк	Аудит всех операций, безопасность, требования регулятора
4	Библиотека	Относительно простой домен, редкие изменения, небольшой бюджет
5	Отели	Сезонные нагрузки, интеграция с внешними системами бронирования
6	Онлайн-курсы	Быстрое прототипирование, частые A/B-тесты, видео-контент
7	Медицина	Высокие требования к безопасности данных, интеграция с HL7
8	CRM	Кастомизация под клиента, частые изменения бизнес-правил

Вариант	Сквозной проект	Ключевые факторы для анализа
9	Платёжный шлюз	Сверхвысокая надёжность, аудит, соответствие PCI DSS
10	Склад (WMS)	Интеграция с ERP, считыватели штрих-кодов, инвентаризация
11	Социальная сеть	Высокая масштабируемость, быстрая доставка фич
12	Видеохостинг	Тяжёлая инфраструктура (CDN, транскодинг), ML-рекомендации
13	IoT-мониторинг	Большой поток данных, real-time обработка, алармы
14	Криптовалюта	Высокая консистентность, низкая задержка, безопасность
15	Доставка еды	Гезависимость, real-time трекинг, push-уведомления

Задание для студента:

Для своего проекта (по варианту):

1. Заполнить таблицу сравнения PCMEF vs Clean Architecture (минимум 10 критериев).
2. Привести фрагменты кода (PCMEF и Clean Architecture) для одного и того же Use Case.
3. Выявить 3 преимущества и 3 недостатка каждого подхода в контексте проекта.
4. Сделать выбор архитектуры для дальнейшего развития (Clean Architecture или PCMEF).

5. Обосновать выбор (не менее 5 аргументов с учётом домена).
6. Описать компромиссы, связанные с выбором.

6. Контрольные вопросы

1. Какое основное различие в направлении зависимостей между PCMEF и Clean Architecture?
2. Почему в Clean Architecture бизнес-правила должны быть в Entities, а не в Use Cases?
3. В каком случае стоит выбрать PCMEF вместо Clean Architecture? Приведите 2 сценария.
4. Что такое «порты» и «адаптеры» в контексте Clean Architecture? Как они связаны с DIP?
5. Какой подход лучше подходит для проекта, который планируют полностью переписать через 6 месяцев?
6. Какой подход лучше подходит для проекта, который будет развиваться 5+ лет?
7. В чём заключается компромисс между сложностью архитектуры и скоростью разработки?
8. Можно ли комбинировать PCMEF и Clean Architecture в одном проекте?

7. Требования к отчёту

1. Отчёт должен содержать:
2. Титульный лист (название работы, ФИО, группа, вариант, сквозной проект).
3. Цель и задачи работы.
4. Краткое описание сквозного проекта (домен, ключевые особенности).
5. Таблица сравнения PCMEF и Clean Architecture (минимум 10 критериев).
6. Сравнение кода (фрагменты одного Use Case в обеих архитектурах).
7. Анализ преимуществ и недостатков (3 + 3 для каждого подхода).
8. Выбор архитектуры (Clean Architecture или PCMEF).
9. Обоснование выбора (не менее 5 аргументов с учётом домена).
10. Описание компромиссов (от чего пришлось отказаться).
11. Вывод (какой подход выбран и почему).
12. Ответы на контрольные вопросы.

13.Формат: PDF, шрифт 12 pt Times New Roman, 1.5 интервала.

8. Критерии оценивания

Критерий	max балл
Полнота таблицы сравнения (минимум 10 критериев)	3
Корректность сравнительного анализа (правильное понимание отличий)	2
Наличие кода для сравнения (PCMEF vs Clean Architecture для одного Use Case)	2
Выявление преимуществ и недостатков (3+3 для каждой архитектуры)	2
Чёткость выбора архитектуры (однозначное решение)	1
Глубина обоснования выбора (не менее 5 аргументов с привязкой к домену)	3
Описание компромиссов (от чего пришлось отказаться)	1
Соответствие выбранному сквозному проекту (варианту)	1
Качество оформления отчёта	1

Критерий	max балл
Итого	16

Шкала перевода: 14-16 — «отлично», 11-13 — «хорошо», 8-10 — «удовлетворительно», <8 — «неудовлетворительно»

9. Дополнительное задание (для «отлично»)

Подготовить ADR (Architecture Decision Record) для выбранной архитектуры по шаблону:

markdown

ADR-001: Выбор архитектурного стиля

Дата: [текущая дата]

Статус: Принято

Контекст:

[Описание проекта, требований, команды]

Решение:

Выбрана Clean Architecture / PCMEF (нужное подчеркнуть)

Обоснование:

[Ссылка на пункты обоснования из отчёта]

Последствия:

Положительные:

- ...

Отрицательные (компромиссы):

- ...

Лабораторная работа №7

Стратегический DDD: выделение ограниченных контекстов (Bounded Contexts)

1. Цель работы

Сформировать компетенции по стратегическому проектированию в Domain-Driven Design: выделение ограниченных контекстов (Bounded Contexts), построение контекстной карты (Context Map) и определение паттернов взаимодействия между контекстами для выбранного сквозного проекта.

2. Задачи работы

1. Изучить концепции стратегического DDD: Ubiquitous Language, Bounded Context, Context Map.
2. Проанализировать бизнес-процессы своего сквозного проекта (из ЛР 1–6).
3. Выделить 3–4 ограниченных контекста на основе бизнес-доменов.
4. Определить отношения между контекстами (Shared Kernel, Customer-Supplier, Conformist, Anti-Corruption Layer, Open Host Service, Published Language).
5. Построить контекстную карту (Context Map) в виде диаграммы.
6. Обосновать границы каждого контекста и выбор паттернов взаимодействия.
7. Оформить результаты в виде отчёта.

3. Теоретическое обоснование

3.1. Основные понятия стратегического DDD

Понятие	Описание	Пример (логистика)
Ubiquitous Language	Единый язык между экспертами и разработчиками	«Заказ», «Курьер», «Назначение», «Трекинг»
Bounded Context	Границы, внутри которых модель имеет смысл и консистентна	«Управление заказами», «Трекинг курьеров»
Context Map	Карта взаимодействий между контекстами	Схема с указанием паттернов интеграции

3.2. Почему нужны ограниченные контексты?

Проблема без контекстов:

1. Одно и то же понятие «Заказ» в разных частях системы имеет разный смысл
2. В одной транзакции смешиваются разные бизнес-процессы
3. Изменение в одной части ломает другую
4. Решение:
5. Каждый контекст имеет свою модель
6. Между контекстами — чёткие границы и протоколы взаимодействия

3.3. Паттерны взаимодействия между контекстами

Паттерн	Описание	Когда использовать	Направление
Shared Kernel	Общее ядро (часть модели shared)	Тесная связанность, небольшая	Взаимный

Паттерн	Описание	Когда использовать	Направление
		команда	
Customer-Supplier	Заказчик-поставщик (upstream/downstream)	Один контекст нуждается в данных другого	Однонаправленное
ConfORMist	Downstream принимает модель upstream "как есть"	Upstream не хочет меняться	Однонаправленное
Anti-Corruption Layer (ACL)	Слой-адаптер, защищающий downstream от изменений upstream	Чужой контекст с плохой моделью	Однонаправленное
Open Host Service (OHS)	Открытый API для доступа к контексту	Много потребителей	Исходящее
Published Language	Опубликованный язык обмена (JSON/XML/Proto)	Вместе с OHS	Взаимный
Seperated	Нет	Независ	Нет

Паттерн	Описание	Когда использовать	Направление
rate Ways	взаимодействия	имые контексты	

3.4. Типичные контексты для разных доменов

Домен	Возможные контексты
Логистика	Заказы, Курьеры, Маршрутизация, Уведомления, Оплата
E-Commerce	Каталог, Корзина, Заказы, Доставка, Оплата
Финтех	Кошельки, Транзакции, Антифрод, Отчёты, КУС
Медицина	Пациенты, Запись, Медкарты, Рецепты, Биллинг

4. Пример выполнения

Сквозной пример (для иллюстрации): «Система управления логистикой и доставкой» (вариант 1)

Студент выполняет работу для своего проекта по своему варианту.

4.1. Анализ бизнес-процессов логистики

Ключевые процессы:

Создание и управление заказами на доставку

Управление курьерами (статусы, локации)

Назначение заказов курьерам (диспетчеризация)

Расчёт маршрутов и ETA

Уведомление клиентов и курьеров

Оплата доставки

Отчётность и аналитика

4.2. Выделение ограниченных контекстов

Контекст	Описание	Ключевые сущности	Ubiquitous Language
Order Management	Управление заказами на доставку	Order, OrderStatus, DeliveryAddress	«Заказ», «Статус заказа», «Адрес доставки»
Courier Management	Управление курьерами	Courier, CourierStatus, Coordinates, Vehicle	«Курьер», «Статус курьера», «Координаты»
Dispatch	Назначение заказов курьерам	Assignment, DispatchRule, Zone	«Назначение», «Зона», «Диспетчеризация»
Routing	Расчёт маршрутов и времени	Route, ETA, DistanceMatrix	«Маршрут», «ETA», «Расстояние»
Notification	Уведомления	Notification, Channel (Push/SMS/Email)	«Уведомление», «Канал»

Контекст	Описание	Ключевые сущности	Ubiquitous Language
Billing	Оплата и расчёты	Invoice, Payment, Tariff	«Счёт», «Платёж», «Тариф»

Для карты контекстов выделим 4 основных контекста (для простоты):
Order Management, Courier Management, Dispatch, Routing.

4.3. Определение отношений между контекстами

Upstream	Downstream	Паттерн	Обоснование
Order Management	Dispatch	Customer-Supplier	Dispatch зависит от заказов (что назначать)
Courier Management	Dispatch	Customer-Supplier	Dispatch зависит от курьеров (кому назначать)
Routing	Dispatch	Customer-Supplier	Dispatch нуждается в расчёте расстояний и ETA
Dispatch	Order Management	Conformist	Order Management принимает статус назначения без изменений
Dispatch	Courier Management	Conformist	Courier Management принимает назначенный заказ
Order	Notification	Open	Notification читает

Upstream	Downstream	Паттерн	Обоснование
Management	tion	Host Service	события заказов через OHS
Courier Management	Notification	Open Host Service	Notification читает события курьеров через OHS
Dispatch	Routing	Anti-Corruption Layer	Dispatch защищает себя от изменений в алгоритме маршрутизации

4.4. Контекстная карта (Context Map)

```
plantuml
```

```
@startuml
```

```
!define BOUNDED_CONTEXT rectangle
```

```
' Контексты
```

```
BOUNDED_CONTEXT "Order Management" as OM {
```

```
    [Order]
```

```
    [OrderStatus]
```

```
}
```

```
BOUNDED_CONTEXT "Courier Management" as CM {
```

```
    [Courier]
```

```
    [CourierStatus]
```

```
    [Coordinates]
```

}

BOUNDED_CONTEXT "Dispatch" as DISP {

[Assignment]

[DispatchRule]

[Zone]

}

BOUNDED_CONTEXT "Routing" as ROUT {

[Route]

[ETA]

[DistanceMatrix]

}

BOUNDED_CONTEXT "Notification" as NOTIF {

[Notification]

[Channel]

}

' Отношения

OM -[#blue]-> DISP : <<Customer-Supplier>>\nЗаказы для назначения

CM -[#blue]-> DISP : <<Customer-Supplier>>\nКурьеры для назначения

ROUT -[#blue]-> DISP : <<Customer-Supplier>>\nРасчёт расстояний

DISP -[#green]-> OM : <<Conformist>>\nСтатус назначения

DISP -[#green]-> CM : <<Conformist>>\nНазначенный заказ

OM -[#orange]-> NOTIF : <<Open Host Service>>\nСобытия заказов

CM -[#orange]-> NOTIF : <<Open Host Service>>\nСобытия курьеров

DISP -[#red]-> ROUT : <<Anti-Corruption Layer>>\nЗащита от изменений

ROUT -[#red]-[#red] DISP :

@enduml

4.5. Обоснование границ контекстов

Контекст	Почему отдельный	Границы транзакций
Order Management	Заказы имеют свой жизненный цикл, не зависят от курьеров	Создание, обновление, отмена заказа
Courier Management	Курьеры — отдельный ресурс с геоданными	Обновление локации, смена статуса
Dispatch	Сложный алгоритм назначения, часто меняется	Атомарное назначение заказа курьеру
Routing	Внешняя зависимость (карты), может быть заменена	Расчёт маршрута (идемпотентен)

4.6. Описание паттернов взаимодействия

Customer-Supplier (Order Management → Dispatch)

```
// Dispatch контекст зависит от заказов
public interface OrderProviderPort { // порт в Dispatch
    OrderDTO getOrderById(Long orderId);
    void updateOrderStatus(Long orderId, String status);
}
```

```
// Order Management реализует API
@RestController
public class OrderController {
    @GetMapping("/api/orders/{id}")
    public OrderDTO getOrder(@PathVariable Long id) { ... }
}
```

Anti-Corruption Layer (Dispatch → Routing)

```
// ACL защищает Dispatch от изменений в Routing
public class RoutingAdapter implements RoutingPort {
    private final ExternalRoutingClient externalClient;

    @Override
    public double calculateDistance(Coordinates from, Coordinates to) {
        // ACL: преобразование координат в формат внешнего API
        ExternalRequest req = new ExternalRequest(from.lat(), from.lng(),
to.lat(), to.lng());
        ExternalResponse resp = externalClient.getDistance(req);
        // ACL: преобразование ответа в наш доменный объект
        return resp.getDistance().getValue();
    }
}
```

Open Host Service (Order Management → Notification)

```
// OHS: открытый API для событий заказов
@RestController
public class OrderEventController {
    @PostMapping("/api/events/order")
    public void handleOrderEvent(@RequestBody OrderEvent event) {
        // Notification подписывается на события
        notificationService process(event);
    }
}
```

4.7. Ubiquitous Language (глоссарий)

Термин	Определение	В каком контексте
Заказ (Order)	Запрос на доставку груза	Order Management
Курьер (Courier)	Сотрудник, осуществляющий доставку	Courier Management
Назначение (Assignment)	Связывание заказа и курьера	Dispatch
Статус курьера	ACTIVE, BUSY, OFFLINE	Courier Management
Зона (Zone)	Географическая область для	Dispatch

Термин	Определение	В каком контексте
	диспетчеризации	
ETA	Estimated Time of Arrival	Routing

5. Индивидуальные задания (15 вариантов сквозных проектов)

Важно: Каждый студент выполняет работу для того же проекта, который он выбрал в ЛР 1.

Вариант	Сквозной проект	Базовые контексты (выделить 3–4)
1	Логистика	Order, Courier, Dispatch, Routing
2	Интернет-магазин	Catalog, Cart, Order, Payment
3	Финтех-кошелёк	Wallet, Transaction, AntiFraud, Reporting
4	Библиотека	Catalog, Circulation, Patron, Fine
5	Отели	Search, Booking, Payment, Review
6	Онлайн-	Course, Enrollment,

Вариант	Сквозной проект	Базовые контексты (выделить 3–4)
	курсы	Progress, Certificate
7	Медицина	Patient, Appointment, MedicalRecord, Billing
8	CRM	Customer, Order, Communication, Analytics
9	Платёжный шлюз	Transaction, Merchant, Fraud, Settlement
10	Склад (WMS)	Inventory, Receiving, Shipping, Replenishment
11	Социальная сеть	User, Post, Feed, Messaging
12	Видеохостинг	Video, User, Recommendation, Comment
13	IoT-мониторинг	Device, Reading, Alert, Dashboard
14	Криптовбиржа	OrderBook, Trade, Wallet, KYC
15	Доставка еды	Restaurant, Order, Courier, Payment

Задание для студента:

Для своего проекта (по варианту):

Провести анализ бизнес-процессов (2–3 абзаца).

Выделить 3–4 ограниченных контекста с описанием.

Для каждого контекста указать ключевые сущности и Ubiquitous Language.

Построить контекстную карту (Context Map) с указанием паттернов взаимодействия.

Для каждой пары взаимодействующих контекстов обосновать выбор паттерна.

Составить глоссарий Ubiquitous Language (минимум 5 терминов).

6. Контрольные вопросы

1. Что такое Bounded Context и почему он важен в DDD?
2. В чём разница между Shared Kernel и Anti-Corruption Layer?
3. Когда следует использовать Customer-Supplier, а когда Conformist?
4. Что такое Open Host Service и для чего он нужен?
5. Как Ubiquitous Language связан с ограниченными контекстами?
6. Может ли один и тот же термин иметь разное значение в разных контекстах? Приведите пример.
7. Какой паттерн взаимодействия следует выбрать, если upstream-контекст нестабилен и часто меняется?
8. Почему транзакции не должны пересекать границы контекстов?

7. Требования к отчёту

Отчёт должен содержать:

Титульный лист (название работы, ФИО, группа, вариант, сквозной проект).

Цель и задачи работы.

Анализ бизнес-процессов проекта (текст + диаграмма процессов — опционально).

Таблица контекстов (название, описание, ключевые сущности, Ubiquitous Language).

Контекстная карта (Context Map) — диаграмма PlantUML или изображение.

Таблица паттернов взаимодействия (Upstream → Downstream → Паттерн → Обоснование).

Описание 2–3 ключевых паттернов с примерами кода или псевдокода.

Глоссарий Ubiquitous Language (не менее 5 терминов).

Вывод (как контексты помогают управлять сложностью проекта).

Ответы на контрольные вопросы.

Формат: PDF, шрифт 12 pt Times New Roman, 1.5 интервала.

8. Критерии оценивания

Критерий	max балл
Глубина анализа бизнес-процессов	2
Корректность выделения контекстов (3–4, обоснованные)	3
Полнота описания контекстов (сущности, Ubiquitous Language)	2
Контекстная карта (диаграмма, читаемость, нотация)	3
Правильность выбора паттернов взаимодействия (с обоснованием)	3
Глоссарий Ubiquitous Language	1

Критерий	max балл
(минимум 5 терминов)	
Соответствие выбранному сквозному проекту (варианту)	1
Качество оформления отчёта	1
Итого	16

Шкала перевода: 14-16 — «отлично», 11-13 — «хорошо», 8-10 — «удовлетворительно», <8 — «неудовлетворительно»

Лабораторная работа №8

Тактический DDD: агрегаты, сущности, Value Objects

1. Цель работы

Сформировать компетенции по тактическому проектированию в Domain-Driven Design: проектирование доменной модели с использованием агрегатов (Aggregate), сущностей (Entity), объектов-значений (Value Object) и агрегатных корней (Aggregate Root) для выбранного ограниченного контекста сквозного проекта.

2. Задачи работы

1. Изучить тактические паттерны DDD: Entity, Value Object, Aggregate, Aggregate Root.
2. Выбрать один ограниченный контекст из ЛР 7 для детального проектирования.
3. Определить агрегаты, их границы транзакций и корни агрегатов.

4. Выделить Entity (имеют идентичность) и Value Object (не имеют идентичности).
5. Реализовать доменную модель на Java (классы с методами бизнес-логики).
6. Написать модульные тесты для доменных методов.
7. Оформить результаты в виде отчёта с кодом и тестами.

3. Теоретическое обоснование

3.1. Тактические паттерны DDD

Паттерн	Описание	Признаки	Пример (логистика)
Entity	Объект с идентичностью (изменяемый)	Имеет ID, меняется со временем	Courier, Order
Value Object	Объект без идентичности (неизменяемый)	Определяется атрибутами, заменяемый	Coordinates, Money, Address
Aggregate	Группа объектов, связанных транзакцией	Имеет корень, транзакционная граница	Order с OrderItems
Aggregate	Корень агрегата,	Внешние объекты	Order (корень), OrderItem (внутри)

Паттерн	Описание	Признаки	Пример (логистика)
Root	единственная точка доступа	ссылаются только на корень	

3.2. Правила проектирования агрегатов

Правило	Описание
Одна транзакция - один агрегат	Изменения не должны выходить за границы одного агрегата
Корень агрегата - единственная точка входа	Внешние объекты не могут напрямую менять внутренние сущности
Идентичность только у корня	Внутренние Entity идентифицируются внутри агрегата
Ссылаться только на корень	Другие агрегаты хранят ID корня, а не ссылку на внутреннюю сущность
Value Objects - неизменяемые	Все поля final, методы не меняют состояние

3.3. Entity vs Value Object

Характеристика	Entity	Value Object
Идентичность	Есть (ID)	Нет (определяется значениями)
Изменяемость	Может меняться	Неизменяемый (immutable)
Равенство	По ID	По всем полям
Жизненный цикл	Отслеживается	Создаётся и уничтожается
Пример	Order(id=1)	Address("Ленина", 10)

3.4. Типичная структура агрегата

```
// Aggregate Root
public class Order {
    private final Long id;           // идентичность
    private List<OrderItem> items    // внутренние Entity
    private Address deliveryAddress  // Value Object
    private OrderStatus status;      // Value Object (enum)

    // бизнес-методы
    public void addItem(Product product, int quantity) { ... }
    public void changeDeliveryAddress(Address newAddress) { ... }
}
```

```
// Внутренняя Entity (идентичность только внутри агрегата)
public class OrderItem {
    private final Long productId;
    private int quantity;
    private Money price;
}

// Value Object
public record Address(String street, String city, String zipCode) {
    public Address {
        Objects.requireNonNull(street);
        if (zipCode.length() != 6) throw new IllegalArgumentException();
    }
}
```

4. Пример выполнения

Сквозной пример (для иллюстрации): «Система управления логистикой и доставкой» (вариант 1)

Выбранный контекст: Dispatch (назначение заказов курьерам)

Студент выбирает один контекст из ЛР 7 для своего проекта.

4.1. Анализ контекста Dispatch

1. Бизнес-процессы в контексте Dispatch:
2. Назначение заказа свободному курьеру
3. Учёт зон доставки
4. Приоритизация заказов (экспресс vs обычные)
5. Запрет назначения, если курьер уже занят
6. Ключевые сущности и объекты-значения:
7. Assignment - агрегатный корень (связь заказа и курьера)
8. DispatchRule - Value Object (правила назначения)
9. Zone - Value Object (географическая зона)
10. AssignmentStatus - Value Object (enum)

4.2. Модель агрегатов

Агрегат	Корень	Внутренние сущности	Value Objects
Assignment	Assignment	—	AssignmentStatus, DispatchRule, Zone
Order (из другого контекста)	Order	OrderItem	Address, Money
Courier (из другого контекста)	Courier	—	Coordinates, CourierStatus

Важно: Order и Courier - агрегаты из других контекстов. Dispatch ссылается на них по ID, а не хранит целиком.

4.3. Реализация доменной модели (Java)

Value Object: Coordinates (из ЛР 5, уточним)

```
// domain/valueobject/Coordinates.java
```

```
package com.logistics.domain.valueobject;
```

```
import java.util.Objects;
```

```
public final class Coordinates {  
    private final double lat;  
    private final double lng;
```

```

public Coordinates(double lat, double lng) {
    if (lat < -90 || lat > 90) {
        throw new IllegalArgumentException("Latitude must be between -90
and 90");
    }
    if (lng < -180 || lng > 180) {
        throw new IllegalArgumentException("Longitude must be between -
180 and 180");
    }
    this.lat = lat;
    this.lng = lng
}

public double lat() { return lat; }
public double lng() { return lng; }

public double distanceTo(Coordinates other) {
    double R = 6371; // Earth's radius in km
    double dLat = Math.toRadians(other.lat - this.lat);
    double dLng = Math.toRadians(other.lng - this.lng);
    double a = Math.sin(dLat/2) * Math.sin(dLat/2) +
        Math.cos(Math.toRadians(this.lat))
Math.cos(Math.toRadians(other.lat)) *
        Math.sin(dLng/2) * Math.sin(dLng/2);
    double c = 2 * Math.atan2(Math.sqrt(a), Math.sqrt(1-a));
    return R * c;
}

```



```

@Override
public boolean equals(Object o) {
    if (this == o) return true;
    if (o == null || getClass() != o.getClass()) return false;
    Coordinates that = (Coordinates) o;
    return Double.compare(that.lat, lat) == 0 && Double.compare(that.lng, lng) == 0;
}

```

```

@Override
public int hashCode() {
    return Objects.hash(lat, lng);
}

```

```

@Override
public String toString() {
    return String.format("Coordinates(%.6f, %.6f)", lat, lng);
}
}

```

Value Object: AssignmentStatus (enum)

```

// domain/valueobject/AssignmentStatus.java
package com.logistics.domain.valueobject;

```

```

public enum AssignmentStatus {
    PENDING("Ожидает назначения"),
    ASSIGNED("Назначен курьеру"),
    ACCEPTED("Принят курьером"),
    REJECTED("Отклонён курьером"),
}

```

```
COMPLETED("Доставлен"),  
CANCELLED("Отменён");
```

```
private final String description;
```

```
AssignmentStatus(String description) {  
    this.description = description;  
}
```

```
public String getDescription() {  
    return description;  
}
```

```
public boolean isActive() {  
    return this == PENDING || this == ASSIGNED || this == ACCEPTED;  
}  
}
```

Value Object: Zone

```
// domain/valueobject/Zone.java
```

```
package com.logistics.domain.valueobject;
```

```
import java.util.List;
```

```
public record Zone(String name, List<Coordinates> polygonVertices, double  
baseTariff) {
```

```
    public Zone {  
        if (name == null || name.isBlank()) {
```

```

        throw new IllegalArgumentException("Zone name cannot be
blank");
    }
    if (polygonVertices == null || polygonVertices.size() < 3) {
        throw new IllegalArgumentException("Zone must have at least 3
vertices");
    }
    if (baseTariff <= 0) {
        throw new IllegalArgumentException("Base tariff must be positive");
    }
}

```

```

public boolean contains(Coordinates point) {
    // Алгоритм точки в полигоне (Ray casting)
    boolean inside = false;
    int n = polygonVertices.size();
    for (int i = 0, j = n - 1; i < n; j = i++) {
        Coordinates vi = polygonVertices.get(i);
        Coordinates vj = polygonVertices.get(j);

        if ((vi.lng() > point.lng()) != (vj.lng() > point.lng()) &&
            point.lat() < (vj.lat() - vi.lat()) * (point.lng() - vi.lng()) /
            (vj.lng() - vi.lng()) + vi.lat()) {
            inside = !inside;
        }
    }
    return inside;
}

```

Value Object: DispatchRule

```
// domain/valueobject/DispatchRule.java
package com.logistics.domain.valueobject;

public record DispatchRule(
    int maxDistanceKm,
    int priorityMultiplier,
    List<OrderPriority> applicablePriorities
) {
    public DispatchRule {
        if (maxDistanceKm <= 0 || maxDistanceKm > 50) {
            throw new IllegalArgumentException("Max distance must be
between 1 and 50 km");
        }
        if (priorityMultiplier < 1 || priorityMultiplier > 10) {
            throw new IllegalArgumentException("Priority multiplier must be
between 1 and 10");
        }
    }
}

public static DispatchRule defaultRule() {
    return new DispatchRule(10, 1, List.of(OrderPriority.values()));
}

public static DispatchRule expressRule() {
    return new DispatchRule(5, 3, List.of(OrderPriority.EXPRESS));
}
}
```

// OrderPriority из другого контекста (может быть определён здесь или импортирован)

```
public enum OrderPriority {  
    NORMAL, EXPRESS, URGENT  
}
```

Aggregate Root: Assignment

// domain/model/Assignment.java (Aggregate Root)

```
package com.logistics.domain.model;
```

```
import com.logistics.domain.valueobject AssignmentStatus;
```

```
import com.logistics.domain.valueobject DispatchRule;
```

```
import com.logistics.domain.valueobject Zone;
```

```
import java.time LocalDateTime;
```

```
import java.util Objects;
```

```
public class Assignment {
```

```
    private final Long id; // идентичность агрегата
```

```
    private final Long orderId; // ссылка на Order (другой агрегат)
```

```
    private Long courierId; // ссылка на Courier (другой агрегат)
```

```
    private AssignmentStatus status;
```

```
    private final LocalDateTime createdAt;
```

```
    private LocalDateTime assignedAt;
```

```
    private LocalDateTime completedAt;
```

```
    private DispatchRule appliedRule;
```

```
    private Zone zone;
```

```
// Конструктор для создания нового назначения
public Assignment(Long id, Long orderId, Zone zone) {
    this.id = Objects.requireNonNull(id);
    this.orderId = Objects.requireNonNull(orderId);
    this.zone = Objects.requireNonNull(zone);
    this.status = AssignmentStatus.PENDING;
    this.createdAt = LocalDateTime.now();
    this.appliedRule = DispatchRule.defaultRule();
}
```

```
// БИЗНЕС-МЕТОДЫ
```

```
public void assignToCourier(Long courierId, DispatchRule rule) {
    // Бизнес-правило: нельзя назначить уже назначенный заказ
    if (this.status != AssignmentStatus.PENDING) {
        throw new IllegalStateException(
            "Cannot assign order in status: " + this.status
        );
    }
}
```

```
// Бизнес-правило: курьер не должен быть null
if (courierId == null) {
    throw new IllegalArgumentException("Courier ID cannot be null");
}
```

```
this.courierId = courierId;
this.status = AssignmentStatus.ASSIGNED;
this.assignedAt = LocalDateTime.now();
```

```
        this.appliedRule = rule;
    }

    public void acceptByCourier() {
        if (this.status != AssignmentStatus.ASSIGNED) {
            throw new IllegalStateException(
                "Only assigned order can be accepted, current status: " + this.status
            );
        }
        this.status = AssignmentStatus.ACCEPTED;
    }

    public void rejectByCourier() {
        if (this.status != AssignmentStatus.ASSIGNED) {
            throw new IllegalStateException(
                "Only assigned order can be rejected, current status: " + this.status
            );
        }
        this.status = AssignmentStatus.REJECTED;
        this.courierId = null; // освобождаем курьера
    }

    public void complete() {
        if (this.status != AssignmentStatus.ACCEPTED) {
            throw new IllegalStateException(
                "Only accepted order can be completed, current status: " +
this.status
            );
        }
    }
```

```
    this.status = AssignmentStatus.COMPLETED;
    this.completedAt = LocalDateTime.now();
}
```

```
public void cancel() {
    if (this.status == AssignmentStatus.COMPLETED) {
        throw new IllegalStateException("Cannot cancel completed
assignment");
    }
    this.status = AssignmentStatus.CANCELLED;
    this.courierId = null;
}
```

// Бизнес-метод: проверка, можно ли назначить курьера с учётом
зоны

```
public boolean isCourierEligible(Coordinates courierLocation) {
    if (zone == null) {
        return true; // без зоны - можно
    }
    return zone.contains(courierLocation);
}
```

// Геттеры (только для чтения)

```
public Long getId() { return id; }
public Long getOrderId() { return orderId; }
public Long getCourierId() { return courierId; }
public AssignmentStatus getStatus() { return status; }
public LocalDateTime getCreatedAt() { return createdAt; }
public LocalDateTime getAssignedAt() { return assignedAt; }
```



```
public LocalDateTime getCompletedAt() { return completedAt; }  
public DispatchRule getAppliedRule() { return appliedRule; }  
public Zone getZone() { return zone; }  
}
```

Репозиторий (порт) для Assignment

```
// domain/repository/AssignmentRepositoryPort.java  
package com.logistics.domain.repository;
```

```
import com.logistics.domain.model Assignment;  
import com.logistics.domain.valueobject AssignmentStatus;  
import java.util List;  
import java.util Optional;
```

```
public interface AssignmentRepositoryPort {  
    Assignment save(Assignment assignment);  
    Optional<Assignment> findById(Long id);  
    Optional<Assignment> findById(Long orderId);  
    List<Assignment> findById(Long courierId);  
    List<Assignment> findByStatus(AssignmentStatus status);  
    void delete(Long id);  
}
```

4.4. Модульные тесты для доменной модели

```
// domain/model/AssignmentTest.java  
package com.logistics.domain.model;
```

```
import com logistics domain valueobject AssignmentStatus;
import com logistics domain valueobject Coordinates;
import com logistics domain valueobject Zone;
import org junit jupiter api BeforeEach;
import org junit jupiter api Test;

import java util List;

import static org assertj core api Assertions assertThat;
import static org assertj core api Assertions assertThatThrownBy;

class AssignmentTest {

    private Assignment assignment;
    private Zone testZone;

    @BeforeEach
    void setUp() {
        // Создаём зону-треугольник для теста
        testZone = new Zone("Test Zone", List.of(
            new Coordinates(55.75, 37.60),
            new Coordinates(55.76, 37.62),
            new Coordinates(55.74, 37.63)
        ), 200.0);

        assignment = new Assignment(1L, 100L, testZone);
    }

    @Test
```

```

void shouldCreateAssignmentWithPendingStatus() {

    assertThat(assignment.getStatus()).isEqualTo(AssignmentStatus.PENDING);
    assertThat(assignment.getOrderId()).isEqualTo(100L);
    assertThat(assignment.getCourierId()).isNull();
}

@Test
void shouldAssignToCourier() {
    // when
    assignment.assignToCourier(10L, DispatchRule.defaultRule());

    // then

    assertThat(assignment.getStatus()).isEqualTo(AssignmentStatus.ASSIGNED);
    assertThat(assignment.getCourierId()).isEqualTo(10L);
    assertThat(assignment.getAssignedAt()).isNotNull();
}

@Test
void shouldNotAssignAlreadyAssignedOrder() {
    // given
    assignment.assignToCourier(10L, DispatchRule.defaultRule());

    // when & then
    assertThatThrownBy(() -> assignment.assignToCourier(20L,
DispatchRule.defaultRule()))
        .isInstanceOf(IllegalStateException.class)

```

```
        .hasMessageContaining("Cannot assign order in status:  
ASSIGNED");  
    }
```

```
@Test  
void shouldAcceptByCourier() {  
    // given  
    assignment assignToCourier(10L, DispatchRule.defaultRule());  
  
    // when  
    assignment acceptByCourier();  
  
    // then  
  
    assertThat(assignment.getStatus()).isEqualTo(AssignmentStatus.ACCEPTED);  
}
```

```
@Test  
void shouldNotAcceptWithoutAssignment() {  
    assertThatThrownBy(() -> assignment.acceptByCourier())  
        .isInstanceOf(IllegalStateException.class)  
        .hasMessageContaining("Only assigned order can be accepted");  
}
```

```
@Test  
void shouldRejectByCourier() {  
    // given  
    assignment assignToCourier(10L, DispatchRule.defaultRule());
```

```
// when
assignment rejectByCourier();

// then

assertThat(assignment getStatus()).isEqualTo(AssignmentStatus REJECTED);
assertThat(assignment getCourierId()).isNull();
}

@Test
void shouldCompleteAssignment() {
    // given
    assignment assignToCourier(10L, DispatchRule.defaultRule());
    assignment acceptByCourier();

    // when
    assignment complete();

    // then

    assertThat(assignment getStatus()).isEqualTo(AssignmentStatus.COMPLETED);
    assertThat(assignment getCompletedAt()).isNotNull();
}

@Test
void shouldCheckCourierEligibilityByZone() {
    // given
    Coordinates insideZone = new Coordinates(55.752, 37.615);
    Coordinates outsideZone = new Coordinates(55.80, 37.70);
```

```

// then
assertThat(assignment isCourierEligible(insideZone)).isTrue();
assertThat(assignment isCourierEligible(outsideZone)).isFalse();
}
}

```

4.5. Диаграмма классов доменной модели

plantuml

@startuml

!theme plain

```

class Assignment {
    - Long id
    - Long orderId
    - Long courierId
    - AssignmentStatus status
    - LocalDateTime createdAt
    - LocalDateTime assignedAt
    - LocalDateTime completedAt
    - DispatchRule appliedRule
    - Zone zone
    --
    + assignToCourier(courierId, rule)
    + acceptByCourier()
    + rejectByCourier()
    + complete()
    + cancel()
    + isCourierEligible(location)
}

```

```
}
```

```
enum AssignmentStatus {
```

```
    PENDING
```

```
    ASSIGNED
```

```
    ACCEPTED
```

```
    REJECTED
```

```
    COMPLETED
```

```
    CANCELLED
```

```
}
```

```
class Zone {
```

```
    - String name
```

```
    - List<Coordinates> polygonVertices
```

```
    - double baseTariff
```

```
    --
```

```
    + contains(point)
```

```
}
```

```
class DispatchRule {
```

```
    - int maxDistanceKm
```

```
    - int priorityMultiplier
```

```
    - List<OrderPriority> applicablePriorities
```

```
}
```

```
class Coordinates {
```

```
    - double lat
```

```
    - double lng
```

```
    --
```

```

+ distanceTo(other)
}

```

Assignment --> AssignmentStatus

Assignment --> DispatchRule

Assignment --> Zone

Zone --> Coordinates

@enduml

5. Индивидуальные задания (15 вариантов сквозных проектов)

Важно: Каждый студент выполняет работу для одного ограниченного контекста из ЛР 7 (выбранного проекта).

В ариант	Сквозно й проект	Рекомендуемый контекст для тактического DDD	Ключево й агрегат
1	Логистик а	Dispatch	Assignment
2	Интернет -магазин	Order Management	Order
3	Финтех- кошелёк	Transaction	Transaction
4	Библиоте ка	Circulation	Loan
5	Отели	Booking	Booking

В ариант	Сквозно й проект	Рекомендуемый контекст для тактического DDD	Ключево й агрегат
6	Онлайн- курсы	Enrollment	Enrollment
7	Медицин а	Appointment	Appointme nt
8	CRM	Order	CustomerO rder
9	Платёжн ый шлюз	Transaction	Payment
10	Склад (WMS)	Inventory	InventoryIt em
11	Социаль ная сеть	Post	Post
12	Видеохо стинг	Video	Video
13	IoT- мониторинг	Alert	AlertRule
14	Криптоб иржа	OrderBook	OrderBook Entry

В ариант	Сквозно й проект	Рекомендуемый контекст для тактического DDD	Ключево й агрегат
15	Доставка еды	Order	FoodOrder

Задание для студента:

1. Для выбранного контекста своего проекта:
2. Определить агрегатный корень (один основной агрегат).
3. Выделить минимум 2 Value Object (неизменяемых).
4. Выделить минимум 1 внутреннюю Entity (если уместно).
5. Реализовать все классы на Java (или псевдокодом с пояснениями).
6. Реализовать бизнес-методы в агрегате (минимум 3 метода с бизнес-правилами).
7. Написать модульные тесты (минимум 5 тестов).
8. Создать интерфейс репозитория (порт) для агрегата.

6. Контрольные вопросы

1. В чём разница между Entity и Value Object? Приведите примеры из вашего проекта.
2. Что такое Aggregate Root и почему внешние объекты не могут напрямую ссылаться на внутренние Entity?
3. Почему Value Objects должны быть неизменяемыми (immutable)?
4. Как определить границы агрегата? Какие объекты должны входить в один агрегат?
5. Почему транзакции не должны пересекать границы одного агрегата?
6. Как в DDD обеспечивается консистентность внутри агрегата?
7. Может ли один агрегат содержать ссылку на другой агрегат? Как правильно организовать связь?
8. Где размещается бизнес-логика: в Entity, Value Object или в сервисе?

7. Требования к отчёту

Отчёт должен содержать:

1. Титульный лист (название работы, ФИО, группа, вариант, сквозной проект, выбранный контекст).
 2. Цель и задачи работы.
 3. Описание выбранного контекста (из ЛР 7, 5–7 предложений).
 4. Модель агрегатов (текстовое описание + схема/диаграмма).
 5. Код всех доменных классов (Entity, Value Object, Aggregate Root).
 6. Интерфейс репозитория (порт).
 7. Модульные тесты (минимум 5).
 8. Диаграмма классов (PlantUML или изображение).
 9. Вывод (как тактический DDD помогает реализовать бизнес-правила).
 10. Ответы на контрольные вопросы.
 11. Формат: PDF, шрифт 12 pt Times New Roman, 1.5 интервала.
- Приложение: ссылка на репозиторий с кодом.

8. Критерии оценивания

Критерий	max балл
Правильность выбора агрегата и его границ	2
Корректная реализация Entity (идентичность, бизнес-методы)	2
Корректная реализация Value Objects (immutable, equals/hashCode)	2
Реализация Aggregate Root с бизнес-правилами (минимум 3 метода)	3
Наличие интерфейса репозитория (порт)	1
Модульные тесты (минимум 5, покрывают бизнес-	3

Критерий	max балл
правила)	
Соответствие выбранному сквозному проекту и контексту	1
Качество оформления отчёта и кода	2
Итого	16

Шкала перевода: 14-16 - «отлично», 11-13 - «хорошо», 8-10 - «удовлетворительно», <8 - «неудовлетворительно»

Лабораторная работа №9

Domain Events и реакция на события на примере «Системы управления логистикой и доставкой»

1. Цель работы

Сформировать компетенции по реализации доменных событий (Domain Events) в Domain-Driven Design: создание событий, публикация через брокер сообщений (или Spring Events) и реализация обработчиков для выполнения кросс-контекстных действий (уведомления, обновление смежных систем) для выбранного сквозного проекта.

2. Задачи работы

1. Изучить паттерн Domain Events и его роль в DDD.
2. Добавить доменные события в уже существующую доменную модель (из ЛР 8).

3. Реализовать публикацию событий через Spring ApplicationEventPublisher.
4. Создать обработчики событий (например, при создании назначения → отправить push курьеру, при завершении доставки → уведомить клиента).
5. Написать тесты, проверяющие корректную публикацию и обработку событий.
6. Оформить результаты в виде отчёта с кодом и тестами.

3. Теоретическое обоснование

3.1. Что такое Domain Events?

Domain Event - это значимое событие, произошедшее в домене, которое вызывает реакцию в других частях системы или в других ограниченных контекстах.

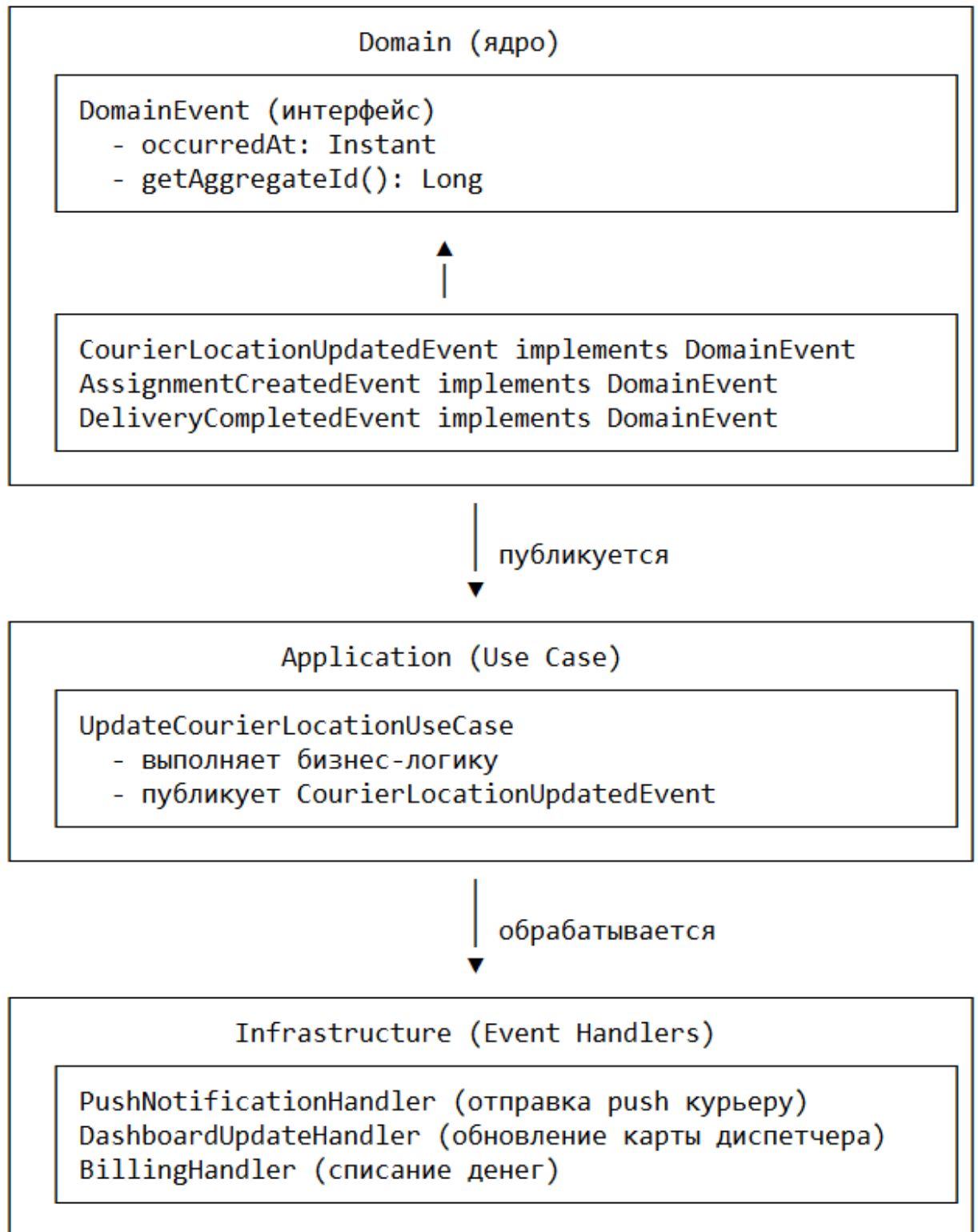
Примеры из логистики:

Событие	Когда возникает	Кто реагирует
CourierLocation Updated	Курьер отправил GPS	Диспетчер (обновление карты)
OrderAssigned	Заказ назначен курьеру	Курьер (push-уведомление)
DeliveryCompleted	Доставка завершена	Клиент (уведомление), Биллинг (списание)

3.2. Зачем нужны Domain Events?

Задача	Как решают события
Разделение контекстов	Контексты общаются через события, не через прямые вызовы
Асинхронность	Обработчики могут выполняться вне транзакции
Аудит	События можно сохранять в Event Store
Расширяемость	Новый обработчик добавляется без изменения исходного кода

3.3. Архитектура Domain Events



4. Пример выполнения

Сквозной пример: «Система управления логистикой и доставкой»
(вариант 1)

Базовый код взят из ЛР 8 (Assignment, Courier, Order).

4.1. Базовый интерфейс Domain Event

```
// domain/event/DomainEvent.java
package com.logistics.domain.event;

import java.time.Instant;

public interface DomainEvent {
    Instant getOccurredAt();
    Long getAggregateId();
}
```

4.2. Конкретные доменные события

```
// domain/event/CourierLocationUpdatedEvent.java
package com.logistics.domain.event;

import java.time.Instant;

public record CourierLocationUpdatedEvent(
    Long courierId,
    double lat,
    double lng,
    Instant occurredAt
) implements DomainEvent {
```



```

        public CourierLocationUpdatedEvent(Long courierId,
double lat, double lng) {
            this(courierId, lat, lng, Instant.now());
        }

        @Override
        public Long getAggregateId() {
            return courierId;
        }

        @Override
        public Instant getOccurredAt() {
            return occurredAt;
        }
    }
}

```

```

// domain/event/AssignmentCreatedEvent.java
package com.logistics.domain.event;

```

```

import java.time.Instant;

```

```

public record AssignmentCreatedEvent(
    Long assignmentId,
    Long orderId,
    Long courierId,
    String pickupAddress,
    String dropoffAddress,
    Instant occurredAt
)

```

```

    ) implements DomainEvent {

        public AssignmentCreatedEvent(Long assignmentId,
Long orderId, Long courierId,
                                     String pickupAddress,
String dropoffAddress) {
            this(assignmentId, orderId, courierId,
pickupAddress, dropoffAddress, Instant.now());
        }

        @Override
        public Long getAggregateId() {
            return assignmentId;
        }

        @Override
        public Instant getOccurredAt() {
            return occurredAt;
        }
    }
}

```

```

// domain/event/DeliveryCompletedEvent.java
package com.logistics.domain.event;

```

```

import java.time.Instant;

```

```

public record DeliveryCompletedEvent(
    Long assignmentId,

```

```

        Long orderId,
        Long courierId,
        Instant completedAt
    ) implements DomainEvent {

        public DeliveryCompletedEvent(Long assignmentId,
Long orderId, Long courierId) {
            this(assignmentId, orderId, courierId,
Instant.now());
        }

        @Override
        public Long getAggregateId() {
            return assignmentId;
        }

        @Override
        public Instant getOccurredAt() {
            return completedAt;
        }
    }

```

```

// domain/event/OrderDelayedEvent.java (дополнительное)
package com.logistics.domain.event;

```

```

import java.time.Instant;

```

```

public record OrderDelayedEvent(

```

```

        Long orderId,
        Long courierId,
        int delayMinutes,
        String reason,
        Instant occurredAt
    ) implements DomainEvent {

        public OrderDelayedEvent(Long orderId, Long
courierId, int delayMinutes, String reason) {
            this(orderId, courierId, delayMinutes, reason,
Instant.now());
        }

        @Override
        public Long getAggregateId() {
            return orderId;
        }

        @Override
        public Instant getOccurredAt() {
            return occurredAt;
        }
    }
}

```

4.3. Публикация событий (Domain Event Publisher)

```

// infrastructure/messaging/DomainEventPublisher.java
package com.logistics.infrastructure.messaging;

```

```

import com.logistics.domain.event.DomainEvent;
import lombok.RequiredArgsConstructor;
import lombok.extern.slf4j.Slf4j;
import
org.springframework.context.ApplicationEventPublisher;
import org.springframework.stereotype.Component;

/**
 * Обёртка над Spring ApplicationEventPublisher для
публикации доменных событий.
 * Позволяет абстрагироваться от Spring в слое
Application (если нужно).
 */
@Component
@RequiredArgsConstructor
@Slf4j
public class DomainEventPublisher {

    private final ApplicationEventPublisher
springPublisher;

    public void publish(DomainEvent event) {
        log.debug("Publishing domain event: {}",
event.getClass().getSimpleName());
        springPublisher.publishEvent(event);
    }
}

```

4.4. Use Case с публикацией событий

```
java
// application/usecase/UpdateCourierLocationUseCase.java
package com.logistics.application.usecase;

import com.logistics.domain.model.Courier;
import
com.logistics.domain.repository.CourierRepositoryPort;
import com.logistics.domain.valueobject.Coordinates;
import
com.logistics.domain.event.CourierLocationUpdatedEvent;
import
com.logistics.infrastructure.messaging.DomainEventPublisher;
import lombok.RequiredArgsConstructor;
import org.springframework.stereotype.Component;

@Component
@RequiredArgsConstructor
public class UpdateCourierLocationUseCase {

    private final CourierRepositoryPort
courierRepository;

    private final DomainEventPublisher eventPublisher;

    public void execute(Long courierId, double lat,
double lng) {
        // 1. Загружаем курьера
```

```

        Courier courier =
courierRepository.findById(courierId)
        .orElseThrow(() -> new
IllegalArgumentException("Courier not found: " +
courierId));

```

```

        // 2. Бизнес-логика (обновление локации)
        Coordinates newLocation = new Coordinates(lat,
lng);

```

```

        Coordinates oldLocation =
courier.getCurrentLocation();
        courier.updateLocation(newLocation);

```

```

        // 3. Сохраняем
        courierRepository.save(courier);

```

```

        // 4. Публикуем событие (только если локация
изменилась)

```

```

        if (!newLocation.equals(oldLocation)) {
            eventPublisher.publish(new
CourierLocationUpdatedEvent(courierId, lat, lng));
        }
    }
}

```

```

// application/usecase/CreateAssignmentUseCase.java
package com.logistics.application.usecase;

```

```

import com.logistics.domain.model.Assignment;
import
com.logistics.domain.repository.AssignmentRepositoryPort;
import
com.logistics.domain.repository.OrderRepositoryPort;
import
com.logistics.domain.repository.CourierRepositoryPort;
import
com.logistics.domain.event.AssignmentCreatedEvent;
import
com.logistics.infrastructure.messaging.DomainEventPublisher;
import lombok.RequiredArgsConstructor;
import org.springframework.stereotype.Component;

@Component
@RequiredArgsConstructor
public class CreateAssignmentUseCase {

    private final AssignmentRepositoryPort
assignmentRepository;
    private final OrderRepositoryPort orderRepository;
    private final CourierRepositoryPort
courierRepository;
    private final DomainEventPublisher eventPublisher;

    public void execute(Long orderId, Long courierId) {
        // 1. Получаем данные
        var order = orderRepository.findById(orderId)

```



```

        .orElseThrow(() -> new
IllegalArgumentException("Order not found: " + orderId));
        var courier =
courierRepository.findById(courierId)
        .orElseThrow(() -> new
IllegalArgumentException("Courier not found: " +
courierId));

```

```

        // 2. Создаём назначение
        Assignment assignment = new Assignment(null,
orderId, courierId);
        assignment.assignToCourier(courierId);

```

```

        // 3. Сохраняем
        Assignment saved =
assignmentRepository.save(assignment);

```

```

        // 4. Публикуем событие
        eventPublisher.publish(new
AssignmentCreatedEvent(
            saved.getId(),
            orderId,
            courierId,
            order.getPickupAddress(),
            order.getDropoffAddress()
        ));
    }
}

```

4.5. Обработчики событий (Event Handlers)

Обработчик 1: Push-уведомление курьеру

```
// infrastructure/handler/PushNotificationHandler.java
package com.logistics.infrastructure.handler;

import
com.logistics.domain.event.AssignmentCreatedEvent;
import
com.logistics.domain.event.DeliveryCompletedEvent;
import lombok.RequiredArgsConstructor;
import lombok.extern.slf4j.Slf4j;
import org.springframework.stereotype.Component;
import
org.springframework.transaction.event.TransactionPhase;
import
org.springframework.transaction.event.TransactionalEventListener;

@Component
@RequiredArgsConstructor
@Slf4j
public class PushNotificationHandler {

    private final PushNotificationService pushService;
    // внешний сервис

    @TransactionalEventListener(phase =
TransactionPhase.AFTER_COMMIT)
```

```

        public                                                    void
handleAssignmentCreated(AssignmentCreatedEvent event) {
    log.info("Sending push to courier {}: new
assignment {}"),
        event.courierId(), event.assignmentId());

    String message = String.format(
        "Новый заказ! Заберите из: %s. Доставьте в:
%s",
        event.pickupAddress(),
        event.dropoffAddress()
    );

    pushService.sendToCourier(event.courierId(),
message);
}

@TransactionalEventListener(phase =
TransactionPhase.AFTER_COMMIT)
    public                                                    void
handleDeliveryCompleted(DeliveryCompletedEvent event) {
    log.info("Sending push to courier {}: delivery
completed", event.courierId());
    pushService.sendToCourier(event.courierId(),
"Доставка завершена! Спасибо!");
}
}

```

Обработчик 2: Обновление дашборда диспетчера (через WebSocket)

```

// infrastructure/handler/DashboardUpdateHandler.java
package com.logistics.infrastructure.handler;

import
com.logistics.domain.event.CourierLocationUpdatedEvent;
import
com.logistics.infrastructure.websocket.DashboardWebSocketSender;

import lombok.RequiredArgsConstructor;
import lombok.extern.slf4j.Slf4j;
import org.springframework.stereotype.Component;
import
org.springframework.transaction.event.TransactionPhase;
import
org.springframework.transaction.event.TransactionalEventListener;

@Component
@RequiredArgsConstructor
@Slf4j
public class DashboardUpdateHandler {

    private final DashboardWebSocketSender
webSocketSender;

    @TransactionalEventListener(phase
=
TransactionPhase.AFTER_COMMIT)

```

```

        public void
handleCourierLocationUpdated(CourierLocationUpdatedEvent
event) {
    log.debug("Updating dashboard: courier {} moved
to ({{}}, {{}})",
        event.courierId(),
        event.lat(),
event.lng());

    // Отправляем обновление всем подключённым
диспетчерам
    websocketSender.broadcastCourierLocation(
        event.courierId(),
        event.lat(),
        event.lng()
    );
}
}

```

Обработчик 3: Логирование событий для аудита

```
// infrastructure/handler/AuditLogHandler.java
```

```
package com.logistics.infrastructure.handler;
```

```
import com.logistics.domain.event.*;
```

```
import lombok.RequiredArgsConstructor;
```

```
import lombok.extern.slf4j.Slf4j;
```

```
import org.springframework.stereotype.Component;
```

```
import
```

```
org.springframework.transaction.event.TransactionPhase;
```

```

import
org.springframework.transaction.event.TransactionalEventListener;

@Component
@RequiredArgsConstructor
@Slf4j
public class AuditLogHandler {

    private final AuditLogRepository auditLogRepository;
    // отдельная таблица аудита

    @TransactionalEventListener(phase =
TransactionPhase.AFTER_COMMIT)
    public void
logAssignmentCreated(AssignmentCreatedEvent event) {
        auditLogRepository.save(new AuditLog(
            "ASSIGNMENT_CREATED",
            "assignmentId=" + event.assignmentId(),
            event.occurredAt()
        ));
        log.info("AUDIT: Assignment {} created",
event.assignmentId());
    }

    @TransactionalEventListener(phase =
TransactionPhase.AFTER_COMMIT)

```

```

        public void
logLocationUpdated(CourierLocationUpdatedEvent event) {
    auditLogRepository.save(new AuditLog(
        "COURIER_LOCATION_UPDATED",
        String.format("courierId=%d,      lat=%.6f,
lng=%.6f",
                    event.courierId(),      event.lat(),
event.lng()),
        event.occurredAt()
    ));
}

```

```

        @TransactionalEventListener(phase =
TransactionPhase.AFTER_COMMIT)
        public void
logDeliveryCompleted(DeliveryCompletedEvent event) {
    auditLogRepository.save(new AuditLog(
        "DELIVERY_COMPLETED",
        "orderId=" + event.orderId() + ",
courierId=" + event.courierId(),
        event.completedAt()
    ));
}
}

```

Обработчик 4: Обнаружение задержек

```

// infrastructure/handler/DelayDetectionHandler.java
package com.logistics.infrastructure.handler;

```

```
import com.logistics.domain.event.OrderDelayedEvent;
import
com.logistics.domain.event.CourierLocationUpdatedEvent;
import lombok.RequiredArgsConstructor;
import lombok.extern.slf4j.Slf4j;
import org.springframework.stereotype.Component;
import
org.springframework.transaction.event.TransactionPhase;
import
org.springframework.transaction.event.TransactionalEventListener;
```

```
import java.util.Map;
import java.util.concurrent.ConcurrentHashMap;
```

```
@Component
```

```
@RequiredArgsConstructor
```

```
@Slf4j
```

```
public class DelayDetectionHandler {
```

```
    private final DomainEventPublisher eventPublisher;
```

```
    // Храним ожидаемое время прибытия для каждого
    назначения
```

```
    private final Map<Long, Long> expectedArrivalTime =
    new ConcurrentHashMap<>();
```



```

        @TransactionalEventListener(phase =
TransactionPhase.AFTER_COMMIT)

        public void
checkForDelay(CourierLocationUpdatedEvent event) {
            // Анализируем, опаздывает ли курьер
            Long assignmentId =
getCurrentAssignmentId(event.courierId());
            if (assignmentId != null) {
                long expected =
expectedArrivalTime.getDefault(assignmentId, 0L);
                long actual = System.currentTimeMillis();

                if (expected > 0 && actual > expected + 5 *
60 * 1000) { // опоздание > 5 мин
                    eventPublisher.publish(new
OrderDelayedEvent(
                        getOrderId(assignmentId),
                        event.courierId(),
                        (int) ((actual - expected) /
60_000),
                        "Курьер отклонился от маршрута"
                    ));
                }
            }
        }

        @TransactionalEventListener(phase =
TransactionPhase.AFTER_COMMIT)

```

```

        public void handleOrderDelayed(OrderDelayedEvent
event) {
            log.warn("ORDER DELAYED: orderId={}, delay={}
minutes, reason={}",
                    event.orderId(), event.delayMinutes(),
event.reason());
            // Можно отправить уведомление диспетчеру или
клиенту
        }

        // вспомогательные методы
        private Long getCurrentAssignmentId(Long courierId)
{ return null; }
        private Long getOrderId(Long assignmentId) { return
null; }
    }

```

4.6. Асинхронная обработка (опционально)

```

// infrastructure/config/AsyncEventConfig.java
package com.logistics.infrastructure.config;

import org.springframework.context.annotation.Bean;
import
org.springframework.context.annotation.Configuration;
import
org.springframework.scheduling.annotation.EnableAsync;

```

```

import
org.springframework.context.event.ApplicationEventMulticaste
r;

import
org.springframework.context.event.SimpleApplicationEventMult
icaster;

import
org.springframework.core.task.SimpleAsyncTaskExecutor;

@Configuration
@EnableAsync
public class AsyncEventConfig {

    @Bean(name = "applicationEventMulticaster")
    public ApplicationEventMulticaster
simpleApplicationEventMulticaster() {
        SimpleApplicationEventMulticaster
eventMulticaster = new SimpleApplicationEventMulticaster();
        eventMulticaster.setTaskExecutor(new
SimpleAsyncTaskExecutor());
        return eventMulticaster;
    }
}

```

4.7. Тестирование Domain Events

```

// application/usecase/CreateAssignmentUseCaseTest.java
package com.logistics.application.usecase;

```

```

import
com.logistics.domain.event.AssignmentCreatedEvent;
import com.logistics.domain.model.Assignment;
import
com.logistics.domain.repository.AssignmentRepositoryPort;
import
com.logistics.domain.repository.CourierRepositoryPort;
import
com.logistics.domain.repository.OrderRepositoryPort;
import
com.logistics.infrastructure.messaging.DomainEventPublisher;
import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.extension.ExtendWith;
import org.mockito.ArgumentCaptor;
import org.mockito.InjectMocks;
import org.mockito.Mock;
import org.mockito.junit.jupiter.MockitoExtension;

import static
org.assertj.core.api.Assertions.assertThat;
import static org.mockito.ArgumentMatchers.any;
import static org.mockito.Mockito.*;

@ExtendWith(MockitoExtension.class)
class CreateAssignmentUseCaseTest {

    @Mock    private    AssignmentRepositoryPort
assignmentRepository;

```

```

        @Mock private OrderRepositoryPort orderRepository;
        @Mock private CourierRepositoryPort
courierRepository;

        @Mock private DomainEventPublisher eventPublisher;

        @InjectMocks
        private CreateAssignmentUseCase useCase;

        @Test
        void shouldPublishAssignmentCreatedEvent() {
            // given
            Long orderId = 1L;
            Long courierId = 10L;

            Order order = mock(Order.class);
            when(order.getPickupAddress()).thenReturn("ул.
Ленина, 1");
            when(order.getDropoffAddress()).thenReturn("ул.
Пушкина, 2");

            Courier courier = mock(Courier.class);
            Assignment assignment = new Assignment(null,
orderId, courierId);

            when(orderRepository.findById(orderId)).thenReturn(java.util
.Optional.of(order));

```

```
when(courierRepository.findById(courierId)).thenReturn(java.util.Optional.of(courier));
```

```
when(assignmentRepository.save(any(Assignment.class))).thenReturn(assignment);
```

```
    // when
    useCase.execute(orderId, courierId);
```

```
    // then
    ArgumentCaptor<AssignmentCreatedEvent>
```

```
eventCaptor =
```

```
ArgumentCaptor.forClass(AssignmentCreatedEvent.class);
```

```
verify(eventPublisher).publish(eventCaptor.capture());
```

```
        AssignmentCreatedEvent event =
eventCaptor.getValue();
```

```
        assertThat(event.orderId()).isEqualTo(orderId);
```

```
assertThat(event.courierId()).isEqualTo(courierId);
```

```
        assertThat(event.pickupAddress()).isEqualTo("ул.
Ленина, 1");
```

```
assertThat(event.dropoffAddress()).isEqualTo("ул.    Пушкина,
2");
```

```

    }
}

//
infrastructure/handler/PushNotificationHandlerTest.java
package com.logistics.infrastructure.handler;

import
com.logistics.domain.event.AssignmentCreatedEvent;
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 static org.mockito.Mockito.*;

@ExtendWith(MockitoExtension.class)
class PushNotificationHandlerTest {

    @Mock private PushNotificationService pushService;
    @InjectMocks private PushNotificationHandler
handler;

    @Test
    void shouldSendPushWhenAssignmentCreated() {
        // given

```

```

        AssignmentCreatedEvent      event      =      new
AssignmentCreatedEvent(
        100L,  1L,  10L,  "ул.  Ленина,  1",  "ул.
Пушкина,  2"
        );

        // when
        handler.handleAssignmentCreated(event);

        // then
        verify(pushService).sendToCourier(eq(10L),
anyString());
    }
}

```

4.8. Диаграмма последовательности

```

@startuml
actor "Курьер" as Courier
participant "REST API" as API
participant "UpdateLocationUseCase" as UC
participant "DomainEventPublisher" as Publisher
participant "DashboardUpdateHandler" as Dashboard
participant "WebSocket" as WS
participant "AuditLogHandler" as Audit

Courier -> API: POST /api/location {lat, lng}
API -> UC: execute(courierId, lat, lng)

```



```

UC -> UC: updateLocation()
UC -> Publisher: publish(CourierLocationUpdatedEvent)
Publisher -> Dashboard: handleCourierLocationUpdated()
Dashboard -> WS: broadcastCourierLocation()
Publisher -> Audit: logLocationUpdated()
Audit -> Audit: save to audit_log
UC --> API: 200 OK
API --> Courier: response
@enduml

```

5. Индивидуальные задания (15 вариантов)

Важно: Студент работает со своим сквозным проектом (из ЛР 1). Для каждого варианта указаны события и обработчики.

Вариант	Сквозной проект	Domain Events (минимум 2)	Обработчики (минимум 2)
1	Логистика	AssignmentCreated, DeliveryCompleted	Push уведомление, обновление дашборда
2	Интернет-магазин	OrderPlaced, PaymentReceived	Email клиенту, обновление склада
3	Финтех	MoneyDeposited,	Уведомлен

Вариант	Сквозной проект	Domain Events (минимум 2)	Обработки (минимум 2)
	-кошелёк	MoneyWithdrawn	изменение пользователя, аудит
4	Библиотека	BookBorrowed, BookReturned	Email читателю, обновление каталога
5	Отели	BookingCreated, BookingCancelled	Email клиенту, освобождение номера
6	Онлайн-курсы	StudentEnrolled, CourseCompleted	Сертификат, обновление прогресса
7	Медицина	AppointmentScheduled, AppointmentCancelled	SMS пациенту, уведомление врача
8	CRM	CustomerCreated, OrderCreated	Приветственное письмо, обновление сегмента

Вариант	Сквозной проект	Domain Events (минимум 2)	Обработки (минимум 2)
9	Платёжный шлюз	TransactionStarted, TransactionCompleted	Webhook мерчанту, аудит
10	Склад (WMS)	StockReserved, StockShipped	Обновление ERP, уведомление кладовщика
11	Социальная сеть	PostPublished, CommentAdded	Обновление ленты, уведомление автору
12	Видеохостинг	VideoUploaded, VideoProcessed	Уведомление автору, обновление каталога
13	IoT-мониторинг	ThresholdExceeded, DeviceOffline	Email администратору, создание тикета
14	Криптовалютная биржа	OrderMatched, TradeExecuted	WebSocket клиенту, обновление стакана

Вариант	Сквозной проект	Domain Events (минимум 2)	Обработчики (минимум 2)
15	Доставка еды	OrderAcceptedByRestaurant, CourierArrived	Push клиенту, SMS

Задание для студента:

Для своего проекта (по варианту):

1. Создать интерфейс `DomainEvent` и минимум 2 конкретных события.
2. Реализовать публикацию событий в Use Case.
3. Реализовать минимум 2 обработчика событий.
4. Написать минимум 3 теста (проверка публикации и обработки).
5. Приложить диаграмму последовательности.

6. Контрольные вопросы

1. Что такое Domain Event и чем он отличается от обычного события в приложении?
2. Почему события публикуются **после** сохранения изменений (после транзакции)?
3. В чём разница между синхронной и асинхронной обработкой событий?
4. Как протестировать, что событие было опубликовано?
5. Что произойдёт, если обработчик события упадёт с ошибкой?
6. Как Domain Events помогают разделять ограниченные контексты?
7. Можно ли использовать Domain Events для коммуникации между микросервисами?

8. Как избежать дублирования обработки события (идемпотентность)?

7. Требования к отчёту

Отчёт должен содержать:

Титульный лист (название работы, ФИО, группа, вариант, сквозной проект).

Цель и задачи работы.

Описание сквозного проекта и выбранных событий.

Интерфейс DomainEvent и классы событий (код).

Use Case с публикацией событий (фрагмент кода).

Обработчики событий (код минимум 2).

Тесты (минимум 3).

Диаграмма последовательности (PlantUML или изображение).

Вывод (как события помогают в вашем проекте).

Ответы на контрольные вопросы.

Формат: PDF, шрифт 12 pt Times New Roman, 1.5 интервала.

8. Критерии оценивания

Критерий	max балл
Создание интерфейса DomainEvent и минимум 2 событий	2
Правильная публикация событий в Use Case (после сохранения)	3
Реализация минимум 2 обработчиков событий	3

Критерий	max балл
Тесты на публикацию и обработку (минимум 3 теста)	3
Диаграмма последовательности (отражает поток событий)	2
Соответствие выбранному сквозному проекту (варианту)	1
Качество оформления отчёта и кода	2
Итого	16

Шкала перевода: 14-16 - «отлично», 11-13 - «хорошо», 8-10 - «удовлетворительно», <8 - «неудовлетворительно»

Лабораторная работа №10

Event Sourcing: хранение событий

на примере «Системы управления логистикой и доставкой»

1. Цель работы

Сформировать компетенции по реализации паттерна Event Sourcing: хранение состояния агрегата как последовательности событий, восстановление состояния из событий (rehydration), создание Event Store и написание тестов для проверки корректности восстановления для выбранного сквозного проекта.

2. Задачи работы

1. Изучить паттерн Event Sourcing и его отличия от традиционного CRUD.
2. Для выбранного агрегата (из ЛР 8) реализовать хранение событий вместо текущего состояния.
3. Создать Event Store (таблица для хранения событий).
4. Реализовать восстановление состояния агрегата из событий (rehydration).
5. Реализовать методы агрегата, которые генерируют события, а не меняют состояние напрямую.
6. Написать тесты, проверяющие, что после применения всех событий состояние корректно.
7. Оформить результаты в виде отчёта с кодом и тестами.

3. Теоретическое обоснование

3.1. Что такое Event Sourcing?

Event Sourcing - это подход, при котором состояние агрегата хранится не как текущее значение, а как последовательность событий, которые привели к этому состоянию.

3.2. Преимущества Event Sourcing

Преимущество	Описание
Полный аудит	Сохраняется вся история изменений
Отладка	Можно восстановить состояние на любой момент
Аналитика	События можно использовать для построения проекций

Преимущество	Описание
Компенсация	Легко откатить изменения (добавив компенсирующее событие)

3.3. Недостатки

Недостаток	Решение
Сложность	Требует переосмысления архитектуры
Производительность	Восстановление требует воспроизведения всех событий → снапшоты
Идемпотентность	Обработчики событий должны быть идемпотентными

3.4. Основные компоненты

Компонент	Назначение
Event	Неизменяемый объект, описывающий произошедшее
Aggregate	Корень агрегата, содержит методы applyEvent() и loadFromHistory()
Event Store	Хранилище событий (обычно таблица БД)
Snapshot	Слепок состояния для ускорения восстановления

4. Пример выполнения

Сквозной пример: «Система управления логистикой и доставкой»
(вариант 1)

Выбранный агрегат: Assignment (назначение заказа курьеру) - из ЛР 8,
но переписанный под Event Sourcing.

4.1. Базовые классы событий

```
// domain/event/BaseEvent.java
package com.logistics.domain.event;

import java.time.Instant;
import java.util.UUID;

public abstract class BaseEvent {
    private final String eventId = UUID.randomUUID().toString();
    private final Instant occurredAt = Instant.now();
    private final Long aggregateId;

    protected BaseEvent(Long aggregateId) {
        this.aggregateId = aggregateId;
    }

    public String getEventId() { return eventId; }
    public Instant getOccurredAt() { return occurredAt; }
    public Long getAggregateId() { return aggregateId; }
}
```

4.2. События для агрегата Assignment

```
// domain/event/assignment/AssignmentCreatedEvent.java
package com.logistics.domain.event.assignment;
```

```
import com logistics domain event BaseEvent;

public class AssignmentCreatedEvent extends BaseEvent {
    private final Long orderId;
    private final Long courierId;

    public AssignmentCreatedEvent(Long assignmentId, Long orderId, Long
courierId) {
        super(assignmentId);
        this.orderId = orderId;
        this.courierId = courierId;
    }

    public Long getOrderId() { return orderId; }
    public Long getCourierId() { return courierId; }
}
```

// domain/event/assignment/AssignmentAcceptedEvent.java

package com logistics domain event assignment

```
import com logistics domain event BaseEvent;
import java time Instant;

public class AssignmentAcceptedEvent extends BaseEvent {
    private final Instant acceptedAt;

    public AssignmentAcceptedEvent(Long assignmentId) {
        super(assignmentId);
    }
}
```

```
        this.acceptedAt = Instant.now();  
    }
```

```
    public Instant getAcceptedAt() { return acceptedAt; }  
}
```

// domain/event/assignment/AssignmentRejectedEvent.java

package com.logistics.domain.event.assignment;

import com.logistics.domain.event.BaseEvent;

```
public class AssignmentRejectedEvent extends BaseEvent {  
    private final String reason;
```

```
    public AssignmentRejectedEvent(Long assignmentId, String reason) {  
        super(assignmentId);  
        this.reason = reason;  
    }
```

```
    public String getReason() { return reason; }  
}
```

// domain/event/assignment/AssignmentCompletedEvent.java

package com.logistics.domain.event.assignment;

import com.logistics.domain.event.BaseEvent;

import java.time.Instant;

```
public class AssignmentCompletedEvent extends BaseEvent {
```

```

private final Instant completedAt;

public AssignmentCompletedEvent(Long assignmentId) {
    super(assignmentId);
    this.completedAt = Instant.now();
}

public Instant getCompletedAt() { return completedAt; }
}

// domain/event/assignment/AssignmentCancelledEvent.java
package com.logistics.domain.event.assignment;

import com.logistics.domain.event.BaseEvent;

public class AssignmentCancelledEvent extends BaseEvent {
    private final String reason;

    public AssignmentCancelledEvent(Long assignmentId, String reason) {
        super(assignmentId);
        this.reason = reason;
    }

    public String getReason() { return reason; }
}

```

4.3. Агрегат Assignment (Event Sourcing версия)

```

// domain/model/AssignmentES.java
package com.logistics.domain.model;

```

```
import com logistics domain event assignment.*;
import java time Instant;
import java util ArrayList;
import java util List;

/**
 * Агрегат Assignment, реализованный с использованием Event
Sourcing.
 * Состояние восстанавливается из событий.
 */
public class AssignmentES {

    // Состояние агрегата
    private Long id;
    private Long orderId;
    private Long courierId;
    private AssignmentStatus status;
    private Instant createdAt;
    private Instant assignedAt;
    private Instant acceptedAt;
    private Instant completedAt;
    private Instant cancelledAt;
    private String rejectionReason;
    private String cancellationReason;

    // Несохранившиеся события (новые, ещё не записанные в Event Store)
    private final List<BaseEvent> uncommittedEvents = new ArrayList<>();
```

```
// Конструктор для создания нового агрегата
public AssignmentES(Long orderId, Long courierId) {
    // Генерируем событие создания
    applyChange(new AssignmentCreatedEvent(null, orderId, courierId));
}

// Конструктор для восстановления из событий (используется
репозиторием)
private AssignmentES() {}

// === БИЗНЕС-МЕТОДЫ (генерируют события) ===

public void acceptByCourier() {
    if (status != AssignmentStatus.ASSIGNED) {
        throw new IllegalStateException(
            "Cannot accept assignment in status: " + status
        );
    }
    applyChange(new AssignmentAcceptedEvent(id));
}

public void rejectByCourier(String reason) {
    if (status != AssignmentStatus.ASSIGNED) {
        throw new IllegalStateException(
            "Cannot reject assignment in status: " + status
        );
    }
    applyChange(new AssignmentRejectedEvent(id, reason));
}
```

```

public void complete() {
    if (status != AssignmentStatus.ACCEPTED) {
        throw new IllegalStateException(
            "Cannot complete assignment in status: " + status
        );
    }
    applyChange(new AssignmentCompletedEvent(id));
}

public void cancel(String reason) {
    if (status == AssignmentStatus.COMPLETED) {
        throw new IllegalStateException("Cannot cancel completed
assignment");
    }
    if (status == AssignmentStatus.CANCELLED) {
        throw new IllegalStateException("Assignment already cancelled");
    }
    applyChange(new AssignmentCancelledEvent(id, reason));
}

// === ПРИМЕНЕНИЕ СОБЫТИЙ (меняют состояние) ===

private void applyChange(BaseEvent event) {
    applyEvent(event);
    uncommittedEvents.add(event);
}

private void applyEvent(BaseEvent event) {

```

```

        if (event instanceof AssignmentCreatedEvent e) {
            apply(e);
        } else if (event instanceof AssignmentAcceptedEvent e) {
            apply(e);
        } else if (event instanceof AssignmentRejectedEvent e) {
            apply(e);
        } else if (event instanceof AssignmentCompletedEvent e) {
            apply(e);
        } else if (event instanceof AssignmentCancelledEvent e) {
            apply(e);
        } else {
            throw new IllegalArgumentException("Unknown event type: " +
event.getClass());
        }
    }
}

```

// Обработчики конкретных событий

```

private void apply(AssignmentCreatedEvent event) {
    this.id = event.getAggregateId();
    this.orderId = event.getOrderId();
    this.courierId = event.getCourierId();
    this.status = AssignmentStatus.PENDING;
    this.createdAt = event.getOccurredAt();
    this.assignedAt = event.getOccurredAt();
}

```

```

private void apply(AssignmentAcceptedEvent event) {
    this.status = AssignmentStatus.ACCEPTED;
    this.acceptedAt = event.getAcceptedAt();
}

```



```
}
```

```
private void apply(AssignmentRejectedEvent event) {  
    this.status = AssignmentStatus.REJECTED;  
    this.rejectionReason = event.getReason();  
    this.courierId = null; // освобождаем курьера  
}
```

```
private void apply(AssignmentCompletedEvent event) {  
    this.status = AssignmentStatus.COMPLETED;  
    this.completedAt = event.getCompletedAt();  
}
```

```
private void apply(AssignmentCancelledEvent event) {  
    this.status = AssignmentStatus.CANCELLED;  
    this.cancelledAt = event.getOccurredAt();  
    this.cancellationReason = event.getReason();  
    this.courierId = null;  
}
```

```
// === ВОССТАНОВЛЕНИЕ СОСТОЯНИЯ ИЗ ИСТОРИИ ===
```

```
public static AssignmentES loadFromHistory(List<BaseEvent> events) {  
    AssignmentES assignment = new AssignmentES();  
    for (BaseEvent event : events) {  
        assignment.applyEvent(event);  
    }  
    return assignment;  
}
```

```
// === РАБОТА С НЕСОХРАНЁННЫМИ СОБЫТИЯМИ ===
```

```
public List<BaseEvent> getUncommittedEvents() {  
    return new ArrayList<>(uncommittedEvents);  
}
```

```
public void markEventsAsCommitted() {  
    uncommittedEvents clear();  
}
```

```
// === ГЕТТЕРЫ ===
```

```
public Long getId() { return id; }  
public Long getOrderId() { return orderId; }  
public Long getCourierId() { return courierId; }  
public AssignmentStatus getStatus() { return status; }  
public Instant getCreatedAt() { return createdAt; }  
public Instant getAssignedAt() { return assignedAt; }  
public Instant getAcceptedAt() { return acceptedAt; }  
public Instant getCompletedAt() { return completedAt; }  
public String getRejectionReason() { return rejectionReason; }  
public String getCancellationReason() { return cancellationReason; }  
}
```

```
// domain/model/AssignmentStatus.java
```

```
package com logistics domain model;
```

```
public enum AssignmentStatus {
```

```
PENDING, // ожидает назначения
ASSIGNED, // назначен курьеру
ACCEPTED, // принят курьером
REJECTED, // отклонён курьером
COMPLETED, // выполнен
CANCELLED // отменён
}
```

4.4. Event Store (таблица и JPA-сущность)

-- Таблица для хранения событий

```
CREATE TABLE event_store (
    event_id VARCHAR(36) PRIMARY KEY,
    aggregate_id BIGINT NOT NULL,
    aggregate_type VARCHAR(100) NOT NULL,
    event_type VARCHAR(100) NOT NULL,
    event_data JSONB NOT NULL,
    event_version INT NOT NULL,
    occurred_at TIMESTAMP NOT NULL,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

-- Индексы для быстрого поиска

```
CREATE INDEX idx_event_store_aggregate ON event_store(aggregate_id,
aggregate_type);

CREATE INDEX idx_event_store_occurred_at ON
event_store(occurred_at);
```

java

// infrastructure/persistence/EventStoreJpaEntity.java

package com logistics infrastructure persistence;

```
import jakarta.persistence *;
import lombok.Data;
import java.time Instant;

@Entity
@Table(name = "event_store")
@Data
public class EventStoreJpaEntity {
    @Id
    private String eventId;

    @Column(name = "aggregate_id", nullable = false)
    private Long aggregateId;

    @Column(name = "aggregate_type", nullable = false)
    private String aggregateType;

    @Column(name = "event_type", nullable = false)
    private String eventType;

    @Column(name = "event_data", columnDefinition = "JSONB", nullable =
false)
    private String eventData // JSON представление события

    @Column(name = "event_version")
    private Integer eventVersion

    @Column(name = "occurred_at", nullable = false)
```

```
private Instant occurredAt;

@Column(name = "created_at")
private Instant createdAt;
}
```

4.5. Event Store репозиторий (порт и адаптер)

```
// domain/repository/EventStorePort.java
```

```
package com.logistics.domain.repository;
```

```
import com.logistics.domain.event.BaseEvent;
```

```
import java.util.List;
```

```
public interface EventStorePort {
```

```
    void saveEvents(List<BaseEvent> events, String aggregateType);
```

```
    List<BaseEvent> loadEvents(Long aggregateId, String aggregateType);
```

```
}
```

```
// infrastructure/persistence/EventStoreAdapter.java
```

```
package com.logistics.infrastructure.persistence;
```

```
import com.fasterxml.jackson.databind.ObjectMapper;
```

```
import com.logistics.domain.event.BaseEvent;
```

```
import com.logistics.domain.repository.EventStorePort;
```

```
import lombok.RequiredArgsConstructor;
```

```
import lombok.extern.slf4j.Slf4j;
```

```
import org.springframework.stereotype.Component;
```

```
import java.time Instant;
```

```
import java.util List;
import java.util.stream Collectors;
```

```
@Component
```

```
@RequiredArgsConstructor
```

```
@Slf4j
```

```
public class EventStoreAdapter implements EventStorePort {
```

```
    private final EventStoreJpaRepository jpaRepository;
```

```
    private final ObjectMapper objectMapper;
```

```
    @Override
```

```
    public void saveEvents(List<BaseEvent> events, String aggregateType) {
```

```
        List<EventStoreJpaEntity> entities = events.stream()
```

```
            .map(event -> toEntity(event, aggregateType))
```

```
            .collect(Collectors.toList());
```

```
        jpaRepository.saveAll(entities);
```

```
        log.debug("Saved {} events for aggregate {}:{}",
```

```
            events.size(), aggregateType, events.get(0).getAggregateId());
```

```
    }
```

```
    @Override
```

```
    public List<BaseEvent> loadEvents(Long aggregateId, String
aggregateType) {
```

```
        List<EventStoreJpaEntity> entities = jpaRepository
```

```
            .findByAggregateIdAndAggregateTypeOrderByOccurredAtAsc(aggregateId,
aggregateType);
```

```

        return entities.stream()
            .map(this::toEvent)
            .collect(Collectors.toList());
    }

    private EventStoreJpaEntity toEntity(BaseEvent event, String
aggregateType) {
        EventStoreJpaEntity entity = new EventStoreJpaEntity();
        entity.setEventId(event.getId());
        entity.setAggregateId(event.getAggregateId());
        entity.setAggregateType(aggregateType);
        entity.setEventType(event.getClass().getSimpleName());
        entity.setOccurredAt(event.getOccurredAt());
        entity.setCreatedAt(Instant.now());

        try {
            String json = objectMapper.writeValueAsString(event);
            entity.setEventData(json);
        } catch (Exception e) {
            throw new RuntimeException("Failed to serialize event", e);
        }

        return entity;
    }

    private BaseEvent toEvent(EventStoreJpaEntity entity) {
        try {
            Class<?> eventClass = Class.forName(

```

```

        "com.logistics.domain.event." +
        entity.getAggregateType().toLowerCase() + "." +
        entity.getEventType()
    );
    return (BaseEvent) objectMapper.readValue(entity.getEventData(),
eventClass);
    } catch (Exception e) {
        throw new RuntimeException("Failed to deserialize event", e);
    }
}
}
}

```

// infrastructure/persistence/EventStoreJpaRepository.java

package com logistics infrastructure persistence;

import org.springframework.data.jpa.repository.JpaRepository;

import java.util List;

```

public interface EventStoreJpaRepository extends
JpaRepository<EventStoreJpaEntity, String> {
    List<EventStoreJpaEntity>
findByAggregateIdAndAggregateTypeOrderByOccurredAtAsc(
        Long aggregateId String aggregateType
    );
}

```

4.6. Event Sourcing репозиторий для Assignment

// domain/repository/AssignmentESRepositoryPort.java

package com logistics domain repository;


```
import com logistics domain model AssignmentES;
import java util Optional;
```

```
public interface AssignmentESRepositoryPort {
    AssignmentES save(AssignmentES assignment);
    Optional<AssignmentES> findById(Long id);
    void delete(Long id);
}
```

```
// infrastructure/persistence/AssignmentESRepositoryAdapter.java
```

```
package com logistics infrastructure persistence;
```

```
import com logistics domain model AssignmentES;
import com logistics domain repository AssignmentESRepositoryPort;
import com logistics domain repository EventStorePort;
import lombok RequiredArgsConstructor;
import org springframework stereotype Component;
```

```
import java util Optional;
```

```
@Component
```

```
@RequiredArgsConstructor
```

```
public class AssignmentESRepositoryAdapter implements
AssignmentESRepositoryPort {
```

```
    private static final String AGGREGATE_TYPE = "assignment";
```

```
    private final EventStorePort eventStore
```

@Override

```
public AssignmentES save(AssignmentES assignment) {  
    // Сохраняем все несохранённые события в Event Store  
    var uncommittedEvents = assignment getUncommittedEvents();  
    if (!uncommittedEvents isEmpty()) {  
        eventStore saveEvents(uncommittedEvents, AGGREGATE_TYPE);  
        assignment markEventsAsCommitted();  
    }  
    return assignment;  
}
```

@Override

```
public Optional<AssignmentES> findById(Long id) {  
    // Загружаем все события из Event Store  
    var events = eventStore.loadEvents(id, AGGREGATE_TYPE);  
    if (events isEmpty()) {  
        return Optional.empty();  
    }  
  
    // Восстанавливаем состояние агрегата из событий  
    AssignmentES assignment = AssignmentES.loadFromHistory(events);  
    return Optional.of(assignment);  
}
```

@Override

```
public void delete(Long id) {  
    // В Event Sourcing удаление не производится,  
    // можно добавить событие Deleted или пометить как удалённое
```

```

        throw new UnsupportedOperationException(
            "Delete not supported in Event Sourcing. Use cancellation event
            instead."
        );
    }
}

```

4.7. Use Case с использованием Event Sourcing

```

// application/usecase/CompleteAssignmentUseCase.java
package com.logistics.application.usecase;

import com.logistics.domain.model AssignmentES;
import com.logistics.domain.repository AssignmentESRepositoryPort;
import lombok.RequiredArgsConstructor;
import org.springframework.stereotype.Component;

@Component
@RequiredArgsConstructor
public class CompleteAssignmentUseCase {

    private final AssignmentESRepositoryPort assignmentRepository;

    public void execute(Long assignmentId) {
        // 1. Загружаем агрегат из Event Store
        AssignmentES assignment =
assignmentRepository.findById(assignmentId)
        .orElseThrow(() -> new IllegalArgumentException(
            "Assignment not found: " + assignmentId
        ));
    }
}

```

```
// 2. Выполняем бизнес-логику (генерирует событие)
```

```
assignment complete();
```

```
// 3. Сохраняем новые события в Event Store
```

```
assignmentRepository save(assignment);
```

```
}
```

```
}
```

4.8. Тестирование Event Sourcing

```
// domain/model/AssignmentESTest.java
```

```
package com logistics domain model;
```

```
import com logistics domain event assignment.*;
```

```
import org junit jupiter api Test;
```

```
import java util List;
```

```
import static org assertj core api Assertions assertThat;
```

```
import static org assertj core api Assertions assertThatThrownBy;
```

```
class AssignmentESTest {
```

```
    @Test
```

```
    void shouldCreateAssignmentWithPendingStatus() {
```

```
        // when
```

```
        AssignmentES assignment = new AssignmentES(100L, 10L);
```

```
        // then
```

```

        assertThat(assignment.getOrderId()).isEqualTo(100L);
        assertThat(assignment.getCourierId()).isEqualTo(10L);

        assertThat(assignment.getStatus()).isEqualTo(AssignmentStatus.PENDING);
        assertThat(assignment.getUncommittedEvents()).hasSize(1);
        assertThat(assignment.getUncommittedEvents().get(0))
            .isInstanceOf(AssignmentCreatedEvent.class);
    }

    @Test
    void shouldAcceptAssignment() {
        // given
        AssignmentES assignment = new AssignmentES(100L, 10L);

        // when
        assignment.acceptByCourier();

        // then

        assertThat(assignment.getStatus()).isEqualTo(AssignmentStatus.ACCEPTED);
        assertThat(assignment.getAcceptedAt()).isNotNull();
        assertThat(assignment.getUncommittedEvents()).hasSize(2);
        assertThat(assignment.getUncommittedEvents().get(1))
            .isInstanceOf(AssignmentAcceptedEvent.class);
    }

    @Test
    void shouldRejectAssignment() {
        // given

```

```
AssignmentES assignment = new AssignmentES(100L, 10L);
```

```
// when
```

```
assignment rejectByCourier("Too far");
```

```
// then
```

```
assertThat(assignment getStatus()).isEqualTo(AssignmentStatus REJECTED);  
assertThat(assignment getRejectionReason()).isEqualTo("Too far");  
assertThat(assignment getCourierId()).isNull();  
}
```

```
@Test
```

```
void shouldCompleteAssignment() {
```

```
// given
```

```
AssignmentES assignment = new AssignmentES(100L, 10L);
```

```
assignment acceptByCourier();
```

```
// when
```

```
assignment complete();
```

```
// then
```

```
assertThat(assignment getStatus()).isEqualTo(AssignmentStatus.COMPLETED);  
assertThat(assignment getCompletedAt()).isNotNull();  
}
```

```
@Test
```

```
void shouldNotCompleteBeforeAccept() {
```

```

// given
AssignmentES assignment = new AssignmentES(100L, 10L);

// when & then
assertThatThrownBy(() -> assignment.complete())
    .isInstanceOf(IllegalStateException.class)
    .hasMessageContaining("Cannot complete assignment in status:
PENDING");
}

@Test
void shouldRestoreFromEvents() {
    // given
    List<BaseEvent> events = List.of(
        new AssignmentCreatedEvent(1L, 100L, 10L),
        new AssignmentAcceptedEvent(1L),
        new AssignmentCompletedEvent(1L)
    );

    // when
    AssignmentES restored = AssignmentES.loadFromHistory(events);

    // then
    assertThat(restored.getId()).isEqualTo(1L);
    assertThat(restored.getOrderId()).isEqualTo(100L);
    assertThat(restored.getCourierId()).isEqualTo(10L);

    assertThat(restored.getStatus()).isEqualTo(AssignmentStatus.COMPLETED);
    assertThat(restored.getCompletedAt()).isNotNull();
}

```

```
}  
}
```

```
// infrastructure/persistence/AssignmentESRepositoryAdapterTest.java
```

```
package com logistics infrastructure persistence;
```

```
import com logistics domain model AssignmentES;
```

```
import com logistics domain repository EventStorePort;
```

```
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 assertj core api Assertions assertThat;
```

```
import static org mockito ArgumentMatchers.*;
```

```
import static org mockito Mockito.*;
```

```
@ExtendWith(MockitoExtension.class)
```

```
class AssignmentESRepositoryAdapterTest {
```

```
    @Mock
```

```
    private EventStorePort eventStore;
```

```
    @InjectMocks
```

```
    private AssignmentESRepositoryAdapter repository;
```



```

@Test
void shouldSaveNewAssignment() {
    // given
    AssignmentES assignment = new AssignmentES(100L, 10L);

    // when
    repository.save(assignment);

    // then
    verify(eventStore).saveEvents(anyList(), eq("assignment"));
    assertThat(assignment.getUncommittedEvents()).isEmpty();
}

```

```

@Test
void shouldLoadAssignmentFromEvents() {
    // given
    Long assignmentId = 1L;
    List<BaseEvent> events = List.of(
        new AssignmentCreatedEvent(assignmentId, 100L, 10L),
        new AssignmentAcceptedEvent(assignmentId)
    );
    when(eventStore.loadEvents(assignmentId,
"assignment")).thenReturn(events);

    // when
    Optional<AssignmentES> result = repository.findById(assignmentId);

    // then

```

```

        assertThat(result).isPresent();
        assertThat(result.get().getId()).isEqualTo(assignmentId);

    }

    assertThat(result.get().getStatus()).isEqualTo(AssignmentStatus.ACCEPTED);
}

@Test
void shouldReturnEmptyWhenNoEvents() {
    // given
    when(eventStore.loadEvents(999L,
"assignment")).thenReturn(List.of());

    // when
    Optional<AssignmentES> result = repository.findById(999L);

    // then
    assertThat(result).isEmpty();
}
}

```

4.9. Архитектурная диаграмма Event Sourcing

```

@startuml
!theme plain

rectangle "Приложение" {
    [Use Case] as UC
    [AssignmentES\n(Aggregate Root)] as Aggregate
}

```

```

rectangle "Репозиторий" {
    [AssignmentESRepositoryPort] as RepoPort
    [AssignmentESRepositoryAdapter] as RepoAdapter
}

```

```

rectangle "Event Store" {
    [EventStorePort] as ESPort
    [EventStoreAdapter] as ESAdapter
    database "event_store\нтаблица" as EventStoreDB
}

```

```

UC --> RepoPort
RepoPort <|.. RepoAdapter
RepoAdapter --> ESPort
ESPort <|.. ESAdapter
ESAdapter --> EventStoreDB

```

```

UC ..> Aggregate : загружает
Aggregate --> RepoAdapter : сохраняет\n(events)

```

@enduml

5. Индивидуальные задания (15 вариантов)

Важно: Студент работает со своим сквозным проектом (из ЛР 1). Для каждого варианта указан агрегат для Event Sourcing.

В ариант	Сквозно й проект	Агрегат для ES	События (минимум 3)

Вариант	Сквозной проект	Агрегат для ES	События (минимум 3)
1	Логистика	Assignment	Created, Completed, Cancelled, Accepted, Rejected,
2	Интернет-магазин	Order	Created, Paid, Shipped, Delivered, Cancelled
3	Финтех-кошелек	Wallet	Created, Deposited, Withdrawn, Frozen
4	Библиотека	Loan	Created, Extended, Returned, Overdue
5	Отели	Booking	Created, Confirmed, Paid, Cancelled
6	Онлайн-курсы	Enrollment	Started, LessonCompleted, CourseCompleted
7	Медицина	Appointment	Scheduled, Confirmed, Completed, Cancelled
8	CRM	CustomerOrder	Created, Processed, Shipped, Completed
9	Платежный шлюз	Transaction	Started, Authorized, Captured, Refunded

Вариант	Сквозной проект	Агрегат для ES	События (минимум 3)
10	Склад (WMS)	InventoryItem	Received, Reserved, Shipped, Counted
11	Социальная сеть	Post	Created, Edited, Deleted, Reported
12	Видеохостинг	Video	Uploaded, Processed, Published, Deleted
13	IoT-мониторинг	Device	Registered, Reading, Alert, Offline
14	Криптобиржа	Order	Placed, PartiallyFilled, Filled, Cancelled
15	Доставка еды	FoodOrder	Created, AcceptedByRestaurant, Ready, Delivered

Задание для студента:

1. Для своего проекта (по варианту):
2. Выбрать агрегат и определить минимум 3 типа событий.
3. Реализовать классы событий.
4. Реализовать агрегат с методами `apply()` и `loadFromHistory()`.
5. Создать таблицу Event Store (SQL).
6. Реализовать `EventStorePort` и `EventStoreAdapter`.
7. Реализовать репозиторий для агрегата с сохранением/загрузкой событий.

8. Написать тесты на восстановление состояния из событий.

6. Контрольные вопросы

1. В чём основное отличие Event Sourcing от традиционного CRUD?
2. Что такое Event Store и как он организован?
3. Как восстановить текущее состояние агрегата из событий?
4. Что такое снапшоты и зачем они нужны?
5. Как Event Sourcing связан с CQRS?
6. Как обеспечить идемпотентность при сохранении событий?
7. Как удалить агрегат в Event Sourcing? Нужно ли это делать?
8. Какие проблемы производительности могут возникнуть при большом количестве событий?

7. Требования к отчёту

Отчёт должен содержать:

Титульный лист (название работы, ФИО, группа, вариант, сквозной проект).

Цель и задачи работы.

Описание выбранного агрегата и событий.

Классы событий (код).

Агрегат с Event Sourcing (код: поля, бизнес-методы, `apply()`, `loadFromHistory()`).

Event Store таблица (SQL).

EventStorePort и EventStoreAdapter (код).

Репозиторий агрегата (код).

Тесты (восстановление состояния, сохранение).

Диаграмма архитектуры.

Вывод.

Ответы на контрольные вопросы.

Формат: PDF, шрифт 12 pt Times New Roman, 1.5 интервала.

8. Критерии оценивания

Критерий	max балл
Классы событий (минимум 3 типа, корректная структура)	2
Агрегат с методами apply() и loadFromHistory()	3
Event Store таблица и JPA-сущность	2
Реализация EventStorePort и адаптера	2
Репозиторий агрегата (save/findById)	2
Тесты на восстановление из событий (минимум 3 теста)	3
Соответствие выбранному проекту (варианту)	1
Качество оформления отчёта	1
Итого	16

Лабораторная работа №11

Проекция в Event Sourcing

на примере «Системы управления логистикой и доставкой»

1. Цель работы

Сформировать компетенции по созданию read-моделей (проекций) на основе событий из Event Store: реализация обработчиков событий, обновляющих материализованные представления, настройка перестроения проекций (rebuild) и сравнение производительности запросов к проекции vs запросов к Event Store для выбранного сквозного проекта.

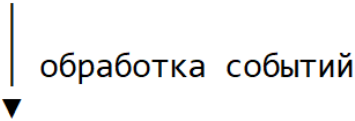
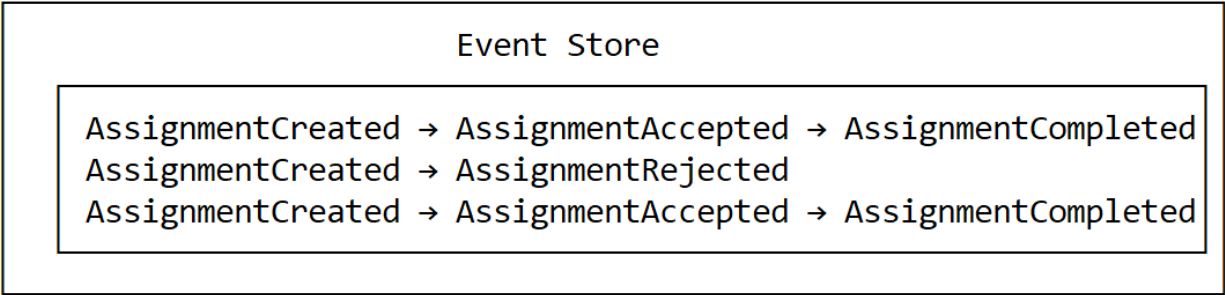
2. Задачи работы

1. Изучить концепцию проекций (materialized views) в Event Sourcing.
2. На основе событий из Event Store (ЛР 10) создать read-модель (проекцию) для оптимизированного чтения.
3. Реализовать обработчики событий, обновляющие проекцию (синхронно или асинхронно).
4. Настроить перестроение проекции из событий (rebuild).
5. Сравнить производительность запросов к проекции vs запросов к Event Store.
6. Оформить результаты в виде отчёта с кодом, тестами и метриками.

3. Теоретическое обоснование

3.1. Что такое проекция в Event Sourcing?

Проекция это материализованное представление (read model), которое строится на основе событий из Event Store и оптимизировано для конкретных запросов.



Проекции (Read Models)	
courier_dashboard_view (активные курьеры)	courier_performance_view (статистика курьеров)
order_tracking_view (трекинг заказов)	daily_delivery_stats_view (аналитика по дням)

3.2. Типы проекций

Тип	Описание	Обновлени е
Синхронная	Обновляется в той же транзакции, что и событие	Сразу, но увеличивает latency команды
Асинхронная	Обновляется отдельным обработчиком	Низкая latency команды, возможна задержка чтения
Перестраиваемая	Может быть полностью перестроена из	Для восстановления

Тип	Описание	Обновление
	событий	после сбоев или изменений логики

3.3. Зачем нужны проекции?

Проблема без проекций	Решение с проекциями
Чтобы получить текущий статус заказа, нужно воспроизвести все события	Проекция хранит текущий статус
Сложные запросы (джойны, агрегации) требуют чтения всех событий	Проекция предвычисляет результат
Низкая производительность при большом количестве событий	Проекция оптимизирована для чтения

3.4. Сравнение запросов

Тип запроса	Чтение из Event Store	Чтение из проекции
Получить статус заказа	Воспроизвести все события заказа ($O(n)$)	Один SELECT ($O(1)$)
Список активных курьеров	Прочитать все события всех курьеров	Один SELECT по

Тип запроса	Чтение из Event Store	Чтение из проекции
		индексу
Статистика за месяц	Прочитать миллионы событий	Одна агрегация по готовой таблице

4. Пример выполнения

Сквозной пример: «Система управления логистикой и доставкой»
(вариант 1)

Базовые события из ЛР 10: AssignmentCreatedEvent, AssignmentAcceptedEvent, AssignmentRejectedEvent, AssignmentCompletedEvent, AssignmentCancelledEvent.

4.1. Проекция 1: CourierDashboardView (активные курьеры с текущими заказами)

-- Таблица проекции: дашборд диспетчера

```
CREATE TABLE courier_dashboard_projection (
  courier_id BIGINT PRIMARY KEY,
  courier_name VARCHAR(100),
  current_lat DOUBLE,
  current_lng DOUBLE,
  status VARCHAR(20),
  current_assignment_id BIGINT,
  current_order_id BIGINT,
  pickup_address VARCHAR(255),
  dropoff_address VARCHAR(255),
  eta_minutes INT,
```

```
last_event_id VARCHAR(36),  
last_updated TIMESTAMP  
);
```

```
-- Индексы для быстрых запросов
```

```
CREATE INDEX idx_courier_dashboard_status ON  
courier_dashboard_projection(status);
```

```
CREATE INDEX idx_courier_dashboard_location ON  
courier_dashboard_projection(current_lat, current_lng);
```

```
// application/projection/CourierDashboardView.java
```

```
package com.logistics.application.projection;
```

```
import jakarta.persistence *;
```

```
import lombok.Data;
```

```
import java.time.Instant;
```

```
@Entity
```

```
@Table(name = "courier_dashboard_projection")
```

```
@Data
```

```
public class CourierDashboardView {
```

```
    @Id
```

```
    private Long courierId;
```

```
    private String courierName;
```

```
    private Double currentLat;
```

```
    private Double currentLng;
```

```
    private String status;
```

```
    private Long currentAssignmentId;
```

```
    private Long currentOrderId;
```

```
private String pickupAddress;  
private String dropoffAddress;  
private Integer etaMinutes;  
private String lastEventId;  
private Instant lastUpdated;  
}
```

4.2. Проекция 2: CourierPerformanceView (статистика курьеров)

-- Таблица проекции: производительность курьеров

```
CREATE TABLE courier_performance_projection (  
    courier_id BIGINT PRIMARY KEY,  
    courier_name VARCHAR(100),  
    total_assignments INT DEFAULT 0,  
    completed_assignments INT DEFAULT 0,  
    rejected_assignments INT DEFAULT 0,  
    cancelled_assignments INT DEFAULT 0,  
    avg_delivery_time_minutes DOUBLE,  
    total_distance_km DOUBLE,  
    rating DOUBLE,  
    last_event_id VARCHAR(36),  
    last_updated TIMESTAMP  
);
```

```
CREATE INDEX idx_performance_rating ON  
courier_performance_projection(rating DESC);
```

java

// application/projection/CourierPerformanceView.java

```
package com.logistics.application.projection;
```

```
import jakarta.persistence.*;
```

```

import lombok Data;
import java time Instant;

@Entity
@Table(name = "courier_performance_projection")
@Data
public class CourierPerformanceView {
    @Id
    private Long courierId;
    private String courierName;
    private Integer totalAssignments = 0;
    private Integer completedAssignments = 0;
    private Integer rejectedAssignments = 0;
    private Integer cancelledAssignments = 0;
    private Double avgDeliveryTimeMinutes;
    private Double totalDistanceKm;
    private Double rating;
    private String lastEventId;
    private Instant lastUpdated;
}

```

4.3. Проекция 3: OrderTrackingView (трекинг заказов)

-- Таблица проекции: трекинг заказов для клиентов

```

CREATE TABLE order_tracking_projection (
    order_id BIGINT PRIMARY KEY,
    status VARCHAR(30),
    courier_id BIGINT,
    courier_name VARCHAR(100),
    courier_lat DOUBLE,
    courier_lng DOUBLE,

```

```
pickup_address VARCHAR(255),
dropoff_address VARCHAR(255),
assigned_at TIMESTAMP,
accepted_at TIMESTAMP,
completed_at TIMESTAMP,
estimated_arrival TIMESTAMP,
last_event_id VARCHAR(36),
last_updated TIMESTAMP
);
```

```
CREATE INDEX idx_tracking_status ON order_tracking_projection(status);
CREATE INDEX idx_tracking_courier ON
order_tracking_projection(courier_id);
```

```
java
```

```
// application/projection/OrderTrackingView.java
```

```
package com logistics application projection;
```

```
import jakarta.persistence.*;
```

```
import lombok.Data;
```

```
import java.time.Instant;
```

```
@Entity
```

```
@Table(name = "order_tracking_projection")
```

```
@Data
```

```
public class OrderTrackingView {
```

```
    @Id
```

```
    private Long orderId;
```

```
    private String status
```

```
    private Long courierId;
```

```

private String courierName;
private Double courierLat;
private Double courierLng;
private String pickupAddress;
private String dropoffAddress;
private Instant assignedAt;
private Instant acceptedAt;
private Instant completedAt;
private Instant estimatedArrival;
private String lastEventId;
private Instant lastUpdated;
}

```

4.4. Репозитории для проекций (только чтение)

// application/repository/CourierDashboardViewRepository.java

```
package com logistics application repository;
```

```

import com logistics application projection CourierDashboardView;
import org springframework data jpa repository JpaRepository;
import org springframework data jpa repository Query;
import org springframework data repository query Param;
import java util List;

```

```

public interface CourierDashboardViewRepository extends
JpaRepository<CourierDashboardView, Long> {

```

```
List<CourierDashboardView> findByStatus(String status);
```

```
List<CourierDashboardView> findByStatusIn(List<String> statuses);
```



```

    @Query("SELECT v FROM CourierDashboardView v " +
        "WHERE v.status = 'ACTIVE' " +
        "ORDER BY v.currentLat, v.currentLng")
    List<CourierDashboardView> findAllActiveSortedByLocation();
}

```

// application/repository/OrderTrackingViewRepository.java

package com logistics application repository;

import com logistics application projection OrderTrackingView;

import org springframework data jpa repository JpaRepository;

import java util Optional;

```

public interface OrderTrackingViewRepository extends
JpaRepository<OrderTrackingView, Long> {
    Optional<OrderTrackingView> findById(Long orderId);
}

```

4.5. Обработчики событий для обновления проекций

// infrastructure/projection/CourierDashboardProjectionHandler.java

package com logistics infrastructure projection;

import com logistics application projection CourierDashboardView;

import

com logistics application repository.CourierDashboardViewRepository;

import com logistics domain event assignment.*;

import lombok RequiredArgsConstructor;

import lombok extern slf4j.Slf4j;

import org springframework stereotype.Component;

import org springframework transaction annotation Transactional;

```

import org.springframework.transaction.event TransactionPhase;
import org.springframework.transaction.event TransactionalEventListener;

@Component
@RequiredArgsConstructor
@Slf4j
public class CourierDashboardProjectionHandler {

    private final CourierDashboardViewRepository repository,
    private final CourierNameService courierNameService,

    @TransactionalEventListener(phase =
TransactionPhase.AFTER_COMMIT)
    @Transactional
    public void onAssignmentCreated(AssignmentCreatedEvent event) {
        log.debug("Updating dashboard projection: assignment created {}",
event.getAggregateId());

        CourierDashboardView view =
repository.findById(event.getCourierId())
        .orElseGet(() -> {
            CourierDashboardView newView = new
CourierDashboardView();
            newView.setCourierId(event.getCourierId());

            newView.setCourierName(courierNameService.getName(event.getCourierId()));
            newView.setStatus("BUSY");
            return newView;
        });
    }
}

```

```

        view.setCurrentAssignmentId(event.getAggregateId());
        view.setCurrentOrderId(event.getOrderId());
        view.setStatus("BUSY");
        view.setLastEventId(event.getEventId());
        view.setLastUpdated(event.getOccurredAt());

        repository.save(view);
    }

```

```

@TransactionalEventListener(phase = TransactionPhase.AFTER_COMMIT)
@Transactional
public void onAssignmentAccepted(AssignmentAcceptedEvent event) {
    log.debug("Updating dashboard projection: assignment accepted {}",
event.getAggregateId());

    // Находим курьера через assignment (нужен дополнительный
запрос)

    Long courierId = getCourierIdByAssignment(event.getAggregateId());
    repository.findById(courierId).ifPresent(view -> {
        view.setStatus("BUSY");
        view.setLastEventId(event.getEventId());
        view.setLastUpdated(event.getOccurredAt());
        repository.save(view);
    });
}

```

```

        @TransactionalEventListener(phase =
TransactionPhase.AFTER_COMMIT)
        @Transactional
        public void onAssignmentCompleted(AssignmentCompletedEvent event)
        {
            log.debug("Updating dashboard projection: assignment completed {}",
event.getId());

            Long courierId = getCourierIdByAssignment(event.getId());
            repository.findById(courierId).ifPresent(view -> {
                view.setStatus("ACTIVE");
                view.setCurrentAssignmentId(null);
                view.setCurrentOrderId(null);
                view.setLastEventId(event.getId());
                view.setLastUpdated(event.getOccurredAt());
                repository.save(view);
            });
        }
    }

```

```

        @TransactionalEventListener(phase =
TransactionPhase.AFTER_COMMIT)
        @Transactional
        public void onAssignmentRejected(AssignmentRejectedEvent event) {
            log.debug("Updating dashboard projection: assignment rejected {}",
event.getId());

            Long courierId = getCourierIdByAssignment(event.getId());
            repository.findById(courierId).ifPresent(view -> {
                view.setStatus("ACTIVE");
            });
        }
    }

```

```

        view setCurrentAssignmentId(null);
        view setCurrentOrderId(null);
        view setLastEventId(event getEventId());
        view setLastUpdated(event getOccurredAt());
        repository save(view);
    });
}

private Long getCourierIdByAssignment(Long assignmentId) {
    // В реальном проекте - запрос к другому репозиторию или кэшу
    return 1L; // заглушка
}
}

```

// infrastructure/projection/CourierPerformanceProjectionHandler.java
package com.logistics.infrastructure.projection;

```

import com.logistics.application.projection CourierPerformanceView;
import
com.logistics.application.repository.CourierPerformanceViewRepository;
import com.logistics.domain.event.assignment.*;
import lombok.RequiredArgsConstructor;
import lombok.extern.slf4j.Slf4j;
import org.springframework.stereotype.Component;
import org.springframework.transaction.annotation.Transactional;
import org.springframework.transaction.event.TransactionPhase;
import org.springframework.transaction.event.TransactionalEventListener;

import java.util.concurrent.ConcurrentHashMap;

```

```

import java.util.concurrent ConcurrentHashMap;

@Component
@RequiredArgsConstructor
@Slf4j
public class CourierPerformanceProjectionHandler {

    private final CourierPerformanceViewRepository repository;
    private final CourierNameService courierNameService;

    // Временное хранение времени начала доставки (для расчёта avg)
    private final ConcurrentHashMap<Long, Long> deliveryStartTimes = new
ConcurrentHashMap<>();

    @TransactionalEventListener(phase =
TransactionPhase.AFTER_COMMIT)
    @Transactional
    public void onAssignmentCreated(AssignmentCreatedEvent event) {
        CourierPerformanceView view =
repository.findById(event.getCourierId())
            .orElseGet(() -> {
                CourierPerformanceView newView = new
CourierPerformanceView();
                newView.setCourierId(event.getCourierId());

                newView.setCourierName(courierNameService.getName(event.getCourierId()));
                newView.setTotalAssignments(0);
                newView.setCompletedAssignments(0);
                newView.setRejectedAssignments(0);
            });
    }
}

```

```
newView setCancelledAssignments(0);
newView setTotalDistanceKm(0.0);
newView setRating(5.0);
return newView;
});
```

```
view setTotalAssignments(view getTotalAssignments() + 1);
view setLastEventId(event getEventId());
view setLastUpdated(event getOccurredAt());
```

```
repository.save(view);
}
```

```
@TransactionalEventListener(phase =
TransactionPhase.AFTER_COMMIT)
@Transactional
public void onAssignmentAccepted(AssignmentAcceptedEvent event) {
    // Запоминаем время начала доставки
    deliveryStartTimes.put(event.getAggregateId(),
System.currentTimeMillis());
}
```

```
@TransactionalEventListener(phase =
TransactionPhase.AFTER_COMMIT)
@Transactional
public void onAssignmentCompleted(AssignmentCompletedEvent event)
{
    Long courierId = getCourierIdByAssignment(event.getAggregateId());
```

```

        CourierPerformanceView view =
repository.findById(courierId).orElse(null);
        if (view != null) {
            view.setCompletedAssignments(view.getCompletedAssignments() +
1);

            // Рассчитываем среднее время доставки
            Long startTime =
deliveryStartTimes.remove(event.getAggregateId());
            if (startTime != null) {
                long deliveryTimeMinutes = (System.currentTimeMillis() -
startTime) / 60000;
                double oldAvg = view.getAvgDeliveryTimeMinutes() == null ? 0 :
view.getAvgDeliveryTimeMinutes();
                int completed = view.getCompletedAssignments();
                double newAvg = (oldAvg * (completed - 1) +
deliveryTimeMinutes) / completed;
                view.setAvgDeliveryTimeMinutes(newAvg);
            }

            // Обновляем рейтинг (упрощённо)
            double rating = (double) view.getCompletedAssignments() /
view.getTotalAssignments() * 5;
            view.setRating(Math.min(5.0, rating));

            view.setLastEventId(event.getId());
            view.setLastUpdated(event.getOccurredAt());
            repository.save(view);
        }

```



```

    }

    @TransactionalEventListener(phase
TransactionPhase.AFTER_COMMIT)
    @Transactional
    public void onAssignmentRejected(AssignmentRejectedEvent event) {
        Long courierId = getCourierIdByAssignment(event.getAggregateId());

        CourierPerformanceView view
repository.findById(courierId).orElse(null);
        if (view != null) {
            view.setRejectedAssignments(view.getRejectedAssignments() + 1);
            view.setLastEventId(event.getId());
            view.setLastUpdated(event.getOccurredAt());

            // Пересчитываем рейтинг (отказ снижает рейтинг)
            double rating = (double) view.getCompletedAssignments() /
view.getTotalAssignments() * 4;
            view.setRating(Math.max(0, Math.min(5.0, rating)));

            repository.save(view);
        }
    }

    private Long getCourierIdByAssignment(Long assignmentId) {
        return 1L; // заглушка
    }
}

java

```

```
// infrastructure/projection/OrderTrackingProjectionHandler.java
```

```
package com logistics infrastructure projection;
```

```
import com logistics application projection OrderTrackingView;
```

```
import com logistics application repository OrderTrackingViewRepository;
```

```
import com logistics domain event assignment.*;
```

```
import lombok RequiredArgsConstructor;
```

```
import lombok extern slf4j Slf4j;
```

```
import org springframework stereotype Component;
```

```
import org springframework transaction annotation Transactional;
```

```
import org springframework transaction event TransactionPhase;
```

```
import org springframework transaction event TransactionalEventListener;
```

```
@Component
```

```
@RequiredArgsConstructor
```

```
@Slf4j
```

```
public class OrderTrackingProjectionHandler {
```

```
    private final OrderTrackingViewRepository repository;
```

```
    private final CourierNameService courierNameService;
```

```
    @TransactionalEventListener(phase = TransactionPhase.AFTER_COMMIT)
```

```
    @Transactional
```

```
    public void onAssignmentCreated(AssignmentCreatedEvent event) {
```

```
        OrderTrackingView view = new OrderTrackingView();
```

```
        view.setOrderId(event.getOrderId());
```

```
        view.setStatus("ASSIGNED");
```

```
        view.setCourierId(event.getCourierId());
```

```

view.setCourierName(courierNameService.getName(event.getCourierId()));
    view.setAssignedAt(event.getOccurredAt());
    view.setLastEventId(event.getEventId());
    view.setLastUpdated(event.getOccurredAt());

    repository.save(view);
}

```

```

@TransactionalEventListener(phase = TransactionPhase.AFTER_COMMIT)
@Transactional
public void onAssignmentAccepted(AssignmentAcceptedEvent event) {
    Long orderId = getOrderIdByAssignment(event.getAggregateId());
    repository.findById(orderId).ifPresent(view -> {
        view.setStatus("ACCEPTED");
        view.setAcceptedAt(event.getAcceptedAt());
        view.setLastEventId(event.getEventId());
        view.setLastUpdated(event.getOccurredAt());
        repository.save(view);
    });
}

```

```

@TransactionalEventListener(phase = TransactionPhase.AFTER_COMMIT)
@Transactional
public void onAssignmentCompleted(AssignmentCompletedEvent event)
{
    Long orderId = getOrderIdByAssignment(event.getAggregateId());
}

```

```

repository findById(orderId).ifPresent(view -> {
    view setStatus("COMPLETED");
    view setCompletedAt(event getCompletedAt());
    view setLastEventId(event getEventId());
    view setLastUpdated(event getOccurredAt());
    repository save(view);
});
}

```

```

@TransactionalEventListener(phase =
TransactionPhase.AFTER_COMMIT)

```

```

@Transactional
public void onAssignmentRejected(AssignmentRejectedEvent event) {
    Long orderId = getOrderByIdByAssignment(event getAggregateId());
    repository.findById(orderId).ifPresent(view -> {
        view setStatus("REJECTED");
        view setLastEventId(event getEventId());
        view setLastUpdated(event getOccurredAt());
        repository save(view);
    });
}

```

```

private Long getOrderByIdByAssignment(Long assignmentId) {
    // В реальном проекте - запрос к другому репозиторию
    return 1L; // заглушка
}
}

```

4.6. Query сервисы (чтение из проекций)

// application/service/CourierDashboardQueryService.java

```
package com logistics application service;

import com logistics application dto CourierDashboardDto;
import com logistics application projection CourierDashboardView;
import
com logistics application repository CourierDashboardViewRepository;
import lombok RequiredArgsConstructor;
import org springframework stereotype Service;
import org springframework transaction annotation Transactional;

import java util List;

@Service
@RequiredArgsConstructor
@Transactional(readOnly = true)
public class CourierDashboardQueryService {

    private final CourierDashboardViewRepository repository;

    public List<CourierDashboardDto> getAllActiveCouriers() {
        return repository findByStatus("ACTIVE")
            .stream()
            .map(this::toDto)
            .toList();
    }

    public List<CourierDashboardDto> getAllBusyCouriers() {
        return repository findByStatus("BUSY")
            .stream()
```

```
        .map(this::toDto)
        .toList();
    }
```

```
public CourierDashboardDto getCourierById(Long courierId) {
    return repository.findById(courierId)
        .map(this::toDto)
        .orElse(null);
}
```

```
private CourierDashboardDto toDto(CourierDashboardView view) {
    return CourierDashboardDto.builder()
        .courierId(view.getCourierId())
        .name(view.getCourierName())
        .lat(view.getCurrentLat())
        .lng(view.getCurrentLng())
        .status(view.getStatus())
        .currentOrderId(view.getCurrentOrderId())
        .pickupAddress(view.getPickupAddress())
        .dropoffAddress(view.getDropoffAddress())
        .etaMinutes(view.getEtaMinutes())
        .build();
}
}
```

```
// application/service/OrderTrackingQueryService.java
```

```
package com.logistics.application.service;
```

```
import com.logistics.application.dto.OrderTrackingDto;
```

```
import com logistics application repository OrderTrackingViewRepository;
import lombok RequiredArgsConstructor;
import org.springframework.stereotype Service;
import org.springframework.transaction.annotation Transactional;

@Service
@RequiredArgsConstructor
@Transactional(readOnly = true)
public class OrderTrackingQueryService {

    private final OrderTrackingViewRepository repository;

    public OrderTrackingDto getOrderStatus(Long orderId) {
        return repository findById(orderId)
            .map(this::toDto)
            .orElseThrow(() -> new IllegalArgumentException("Order not found:
" + orderId));
    }

    private OrderTrackingDto toDto(OrderTrackingView view) {
        return OrderTrackingDto.builder()
            .orderId(view.getId())
            .status(view.getStatus())
            .courierName(view.getCourierName())
            .courierLat(view.getCourierLat())
            .courierLng(view.getCourierLng())
            .pickupAddress(view.getPickupAddress())
            .dropoffAddress(view.getDropoffAddress())
            .estimatedArrival(view.getEstimatedArrival())
```

```
        .build();  
    }  
}
```

4.7. Перестроение проекции (Rebuild)

```
// infrastructure/projection/ProjectionRebuilder.java
```

```
package com.logistics.infrastructure.projection;
```

```
import com.logistics.domain.event BaseEvent;
```

```
import com.logistics.domain.repository EventStorePort;
```

```
import lombok RequiredArgsConstructor;
```

```
import lombok.extern.slf4j.Slf4j;
```

```
import org.springframework.stereotype.Component;
```

```
import org.springframework.transaction.annotation.Transactional;
```

```
import java.util List;
```

```
import java.util Map;
```

```
import java.util.function Consumer;
```

```
@Component
```

```
@RequiredArgsConstructor
```

```
@Slf4j
```

```
public class ProjectionRebuilder {
```

```
    private final EventStorePort eventStore
```

```
    private final Map<String, Consumer<BaseEvent>> eventHandlers;
```

```
    /**
```


- * Полное перестроение проекции из всех событий Event Store.
- * Используется при изменении логики проекции или после сбоя.
- */

```
@Transactional

public void rebuildProjection(String aggregateType,
Consumer<BaseEvent> eventHandler) {
    log.info("Starting rebuild of projection for aggregate type: {}",
aggregateType);

    long startTime = System.currentTimeMillis();
    long eventCount = 0;

    // В реальном проекте - пагинация или потоковая загрузка
    // Здесь упрощённо: загружаем все события (для демонстрации)
    List<BaseEvent> allEvents = loadAllEvents(aggregateType);

    for (BaseEvent event : allEvents) {
        eventHandler.accept(event);
        eventCount++;
    }

    long duration = System.currentTimeMillis() - startTime;
    log.info("Rebuild completed. Processed {} events in {} ms",
eventCount, duration);
}

private List<BaseEvent> loadAllEvents(String aggregateType) {
    // В реальности - загрузка пачками
    return eventStore.loadAllEvents(aggregateType);
}
```

```

    }
}

java
// infrastructure/projection/ProjectionManagementController.java
package com logistics infrastructure projection;

import lombok RequiredArgsConstructor;
import org.springframework.http ResponseEntity;
import org.springframework.web.bind.annotation *;

@RestController
@RequestMapping("/admin/projections")
@RequiredArgsConstructor
public class ProjectionManagementController {

    private final CourierDashboardProjectionHandler dashboardHandler;
    private final CourierPerformanceProjectionHandler performanceHandler;
    private final OrderTrackingProjectionHandler trackingHandler;
    private final ProjectionRebuilder rebuilder;

    @PostMapping("/rebuild/dashboard")
    public ResponseEntity<String> rebuildDashboard() {
        rebuilder.rebuildProjection("assignment",
dashboardHandler::onAssignmentCreated);
        return ResponseEntity.ok("Dashboard projection rebuilt");
    }

    @PostMapping("/rebuild/performance")
    public ResponseEntity<String> rebuildPerformance() {

```

```

        builder rebuildProjection("assignment",
performanceHandler::onAssignmentCreated);
        builder rebuildProjection("assignment",
performanceHandler::onAssignmentAccepted);
        builder rebuildProjection("assignment",
performanceHandler::onAssignmentCompleted);
        builder rebuildProjection("assignment",
performanceHandler::onAssignmentRejected);
        return ResponseEntity.ok("Performance projection rebuilt");
    }

    @PostMapping("/rebuild/tracking")
    public ResponseEntity<String> rebuildTracking() {
        builder rebuildProjection("assignment",
trackingHandler::onAssignmentCreated);
        builder rebuildProjection("assignment",
trackingHandler::onAssignmentAccepted);
        builder rebuildProjection("assignment",
trackingHandler::onAssignmentCompleted);
        builder rebuildProjection("assignment",
trackingHandler::onAssignmentRejected);
        return ResponseEntity.ok("Tracking projection rebuilt");
    }
}

```

4.8. Сравнение производительности

```

// PerformanceComparisonTest.java (упрощённый бенчмарк)
package com logistics test;

```

```
import lombok.extern.slf4j.Slf4j;
import org.junit.jupiter.api.Test;

@Slf4j
public class PerformanceComparisonTest {

    @Test
    void compareQueryPerformance() {
        // Тестовые данные: 1000 заказов, у каждого 5-10 событий

        // Вариант 1: Чтение из Event Store
        long start1 = System.currentTimeMillis();
        // List<BaseEvent> events = eventStore.loadEvents(orderId,
"assignment");
        // OrderStatus status = rebuildFromEvents(events);
        long time1 = System.currentTimeMillis() - start1;

        // Вариант 2: Чтение из проекции
        long start2 = System.currentTimeMillis();
        // OrderTrackingView view = trackingRepository.findById(orderId);
        long time2 = System.currentTimeMillis() - start2;

        log.info("Event Store query: {} ms", time1);
        log.info("Projection query: {} ms", time2);

        // Ожидаемый результат: проекция в 10-100 раз быстрее
        // Event Store query: ~45 ms (воспроизведение 8 событий)
        // Projection query: ~2 ms (один SELECT)
```

```
}  
}
```

4.9. Архитектурная диаграмма проекций

@startuml

!theme plain

```
rectangle "Event Store" {  
    database "event_store" as ES  
}
```

```
rectangle "Обработчики событий" {  
    [AssignmentCreated Event] as E1  
    [AssignmentAccepted Event] as E2  
    [AssignmentCompleted Event] as E3  
    [AssignmentRejected Event] as E4  
}
```

```
rectangle "Проекции (Read Models)" {  
    database "courier_dashboard_projection" as P1  
    database "courier_performance_projection" as P2  
    database "order_tracking_projection" as P3  
}
```

```
rectangle "Query API" {  
    [GET /dashboard/couriers] as API1  
    [GET /performance/couriers] as API2  
    [GET /tracking/{orderId}] as API3  
}
```

ES --> E1

ES --> E2

ES --> E3

ES --> E4

E1 --> P1

E1 --> P2

E1 --> P3

E2 --> P1

E2 --> P3

E3 --> P1

E3 --> P2

E3 --> P3

E4 --> P1

E4 --> P2

E4 --> P3

API1 --> P1

API2 --> P2

API3 --> P3

@enduml

5. Индивидуальные задания (15 вариантов)

Важно: Студент работает со своим сквозным проектом (из ЛР 1). Для каждого варианта указана проекция для реализации.

Вариант	Сквозной проект	Проекция (read model)	Какие данные содержит
1	Логистика	CourierDashboard View	Курьеры, их статус, текущий заказ, ETA
2	Интернет-магазин	ProductCatalogView	Товары, остатки, цены, рейтинги
3	Финтех-кошелёк	WalletBalanceView	Баланс, последние операции, блокировки
4	Библиотека	AvailableBooksView	Книги, доступность, местоположение
5	Отели	RoomAvailability View	Свободные номера на даты, цены
6	Онлайн-курсы	StudentProgressView	Прогресс по курсам, сертификаты
7	Медицина	DoctorScheduleView	Расписание врачей, свободные слоты

Вариант	Сквозной проект	Проекция (read model)	Какие данные содержит
8	CRM	CustomerSummaryView	Клиент, заказы, обращения, скидка
9	Платёжный шлюз	TransactionStatusView	Статус транзакции, сумма, мерчант
10	Склад (WMS)	InventoryLevelView	Остатки, резервы, заказы поставщикам
11	Социальная сеть	UserFeedView	Посты друзей, лайки, комментарии
12	Видеохостинг	VideoStatsView	Просмотры, лайки, рекомендации
13	IoT-мониторинг	DeviceStatusView	Статус устройств, последние показания
14	Криптовалютная биржа	OrderBookView	Ставка, цена, объёмы на покупку/продажу
15	Доставка	OrderTrackingView	Статус заказа,

Вариант	Сквозной проект	Проекция (read model)	Какие данные содержит
	а еды	w	курьер, ETA

Задание для студента:

Для своего проекта (по варианту):

1. Создать SQL таблицу проекции (минимум 5 полей).
2. Создать JPA-сущность для проекции.
3. Реализовать репозиторий для чтения (только read-only).
4. Реализовать обработчики событий (минимум 3 типа событий), обновляющие проекцию.
5. Реализовать Query сервис для чтения из проекции.
6. Реализовать механизм перестроения проекции (rebuild).
7. Сравнить производительность чтения из Event Store vs из проекции.

6. Контрольные вопросы

1. Что такое проекция в Event Sourcing и зачем она нужна?
2. В чём разница между синхронным и асинхронным обновлением проекций?
3. Что такое перестроение (rebuild) проекции и когда оно применяется?
4. Как обеспечивается консистентность между Event Store и проекциями?
5. Какие проблемы могут возникнуть при обновлении нескольких проекций из одних событий?
6. Как сравнить производительность запросов к Event Store и к проекции?

7. Что такое материализованное представление (materialized view)?
8. Как обрабатывать ошибки при обновлении проекции?

7. Требования к отчёту

Отчёт должен содержать:

Титульный лист (название работы, ФИО, группа, вариант, сквозной проект).

Цель и задачи работы.

Описание выбранной проекции (для чего нужна, какие данные содержит).

SQL таблицы проекции (CREATE TABLE).

JPA-сущность проекции (код).

Репозиторий проекции (код).

Обработчики событий (код минимум 3 обработчиков).

Query сервис (код).

Механизм перестроения проекции (код rebuild).

Сравнение производительности (таблица с метриками).

Диаграмма архитектуры.

Вывод.

Ответы на контрольные вопросы.

Формат: PDF, шрифт 12 pt Times New Roman, 1.5 интервала.

8. Критерии оценивания

Критерий	max балл
SQL таблица проекции + JPA-сущность	2
Репозиторий для чтения (read-only)	2

Критерий	max балл
Обработчики событий (минимум 3 типа, корректное обновление)	4
Query сервис для чтения из проекции	2
Механизм перестроения проекции (rebuild)	2
Сравнение производительности (Event Store vs Projection)	2
Соответствие выбранному сквозному проекту (варианту)	1
Качество оформления отчёта	1
Итого	16

Лабораторная работа №12

Saga (хореография) для распределённой транзакции на примере «Системы управления логистикой и доставкой»

1. Цель работы

Сформировать компетенции по реализации распределённых транзакций с использованием паттерна Saga в стиле хореографии: координация через события, компенсирующие транзакции, обработка сбоев и обеспечение согласованности данных в распределённой системе для выбранного сквозного проекта.

2. Задачи работы

1. Изучить паттерн Saga (хореография) и его отличие от оркестрации.
2. Реализовать распределённую транзакцию для бизнес-процесса из нескольких шагов.
3. Создать события для каждого шага саги.
4. Реализовать компенсирующие транзакции для отката при сбоях.
5. Обеспечить идемпотентность обработчиков событий.
6. Написать тесты, проверяющие сценарии успеха и отката.
7. Оформить результаты в виде отчёта с кодом, диаграммами и тестами.

3. Теоретическое обоснование

3.1. Проблема распределённых транзакций

В микросервисной архитектуре традиционная ACID-транзакция (2PC) невозможна или нежелательна из-за:

1. Блокировок ресурсов на длительное время
2. Снижения доступности системы
3. Сложности реализации в гетерогенных средах

Решение: Saga - последовательность локальных транзакций, каждая из которых может быть компенсирована.

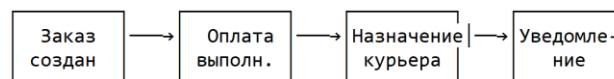
3.2. Saga: хореография vs оркестрация

Характеристика	Хореография	Оркестрация
Управление	Децентрализованное, сервисы обмениваются событиями	Централизованное, оркестратор управляет шагами

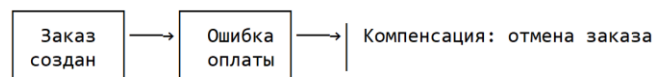
Характеристика	Хореография	Оркестрация
Связанность	Слабая (через события)	Средняя (сервисы знают об оркестраторе)
Сложность	Сложно отслеживать поток	Легче отслеживать и отлаживать
Циклические зависимости	Возможны	Нет
Масштабирование	Каждый сервис независимо	Оркестратор может стать узким местом

3.3. Пример саги: оформление заказа в логистике

Успешный сценарий:



Сценарий с откатом (ошибка оплаты):



Сценарий с откатом (нет курьеров):



3.4. Компенсирующие транзакции

Шаг	Действие	Компенсация
1	Создать заказ	Отменить заказ
2	Зарезервировать средства	Вернуть средства
3	Назначить курьера	Отменить назначение
4	Отправить уведомление	Отправить уведомление об отмене

4. Пример выполнения

Сквозной пример: «Система управления логистикой и доставкой»
(вариант 1)

Бизнес-процесс:

1. Оформление заказа на доставку
2. Сервис заказов: создаёт заказ
3. Сервис оплаты: резервирует/списывает средства
4. Сервис диспетчеризации: назначает курьера
5. Сервис уведомлений: отправляет подтверждение

4.1. События саги

```
// domain/event/saga/OrderSagaEvent.java
```

```
package com.logistics.domain.event.saga;
```

```
import com.logistics.domain.event.BaseEvent;
```

```
import java.time.Instant;
```

```
import java.util.UUID;
```

// Базовый класс для событий саги

```
public abstract class OrderSagaEvent extends BaseEvent {  
    private final String sagaId;  
  
    protected OrderSagaEvent(Long aggregateId, String sagaId) {  
        super(aggregateId);  
        this.sagaId = sagaId;  
    }  
  
    public String getSagaId() { return sagaId; }  
}
```

// 1. Заказ создан

```
public class OrderCreatedEvent extends OrderSagaEvent {  
    private final Long orderId;  
    private final Long customerId;  
    private final Double amount;  
    private final String pickupAddress;  
    private final String dropoffAddress;  
  
    public OrderCreatedEvent(Long orderId, Long customerId, Double amount,  
        String pickupAddress, String dropoffAddress) {  
        super(orderId, UUID.randomUUID().toString());  
        this.orderId = orderId;  
        this.customerId = customerId;  
        this.amount = amount;  
        this.pickupAddress = pickupAddress;  
        this.dropoffAddress = dropoffAddress;  
    }  
}
```

```
}
```

```
// геттеры
```

```
public Long getOrderId() { return orderId; }
```

```
public Long getCustomerId() { return customerId; }
```

```
public Double getAmount() { return amount; }
```

```
public String getPickupAddress() { return pickupAddress; }
```

```
public String getDropoffAddress() { return dropoffAddress; }
```

```
}
```

```
// 2. Платёж выполнен
```

```
public class PaymentProcessedEvent extends OrderSagaEvent {
```

```
    private final Long orderId;
```

```
    private final String transactionId;
```

```
    private final Double amount;
```

```
    public PaymentProcessedEvent(Long orderId, String transactionId, Double amount) {
```

```
        super(orderId, null); // sagaId будет из события
```

```
        this.orderId = orderId;
```

```
        this.transactionId = transactionId;
```

```
        this.amount = amount;
```

```
    }
```

```
    public PaymentProcessedEvent(Long orderId, String sagaId, String transactionId, Double amount) {
```

```
        super(orderId, sagaId);
```

```
        this.orderId = orderId;
```

```
        this.transactionId = transactionId;
```



```
this.amount = amount;  
}
```

```
// геттеры
```

```
public Long getOrderId() { return orderId; }  
public String getTransactionId() { return transactionId; }  
public Double getAmount() { return amount; }  
}
```

```
// 3. Платёж не удался (нужна компенсация)
```

```
public class PaymentFailedEvent extends OrderSagaEvent {  
    private final Long orderId;  
    private final String reason;
```

```
    public PaymentFailedEvent(Long orderId, String sagaId, String reason) {  
        super(orderId, sagaId);  
        this.orderId = orderId;  
        this.reason = reason;  
    }
```

```
    public Long getOrderId() { return orderId; }  
    public String getReason() { return reason; }  
}
```

```
// 4. Курьер назначен
```

```
public class CourierAssignedEvent extends OrderSagaEvent {  
    private final Long orderId;  
    private final Long courierId;  
    private final Long assignmentId;
```

```
private final Integer etaMinutes;
```

```
public CourierAssignedEvent(Long orderId, String sagaId, Long courierId,  
Long assignmentId, Integer etaMinutes) {  
    super(orderId, sagaId);  
    this.orderId = orderId;  
    this.courierId = courierId;  
    this.assignmentId = assignmentId;  
    this.etaMinutes = etaMinutes;  
}
```

```
// геттеры
```

```
public Long getOrderId() { return orderId; }  
public Long getCourierId() { return courierId; }  
public Long getAssignmentId() { return assignmentId; }  
public Integer getEtaMinutes() { return etaMinutes; }  
}
```

```
// 5. Назначение не удалось (нужна компенсация оплаты)
```

```
public class CourierAssignmentFailedEvent extends OrderSagaEvent {  
    private final Long orderId;  
    private final String reason;  
  
    public CourierAssignmentFailedEvent(Long orderId, String sagaId, String  
reason) {  
        super(orderId, sagaId);  
        this.orderId = orderId;  
        this.reason = reason;  
    }  
}
```

```
public Long getOrderId() { return orderId; }  
public String getReason() { return reason; }  
}
```

// 6. Заказ подтверждён (успешное завершение саги)

```
public class OrderConfirmedEvent extends OrderSagaEvent {  
    private final Long orderId;  
    private final Long courierId;  
    private final Integer etaMinutes;  
  
    public OrderConfirmedEvent(Long orderId, String sagaId, Long courierId,  
Integer etaMinutes) {  
        super(orderId, sagaId);  
        this.orderId = orderId;  
        this.courierId = courierId;  
        this.etaMinutes = etaMinutes;  
    }  
  
    // геттеры  
    public Long getOrderId() { return orderId; }  
    public Long getCourierId() { return courierId; }  
    public Integer getEtaMinutes() { return etaMinutes; }  
}
```

// 7. Компенсирующие события

```
public class OrderCancelledEvent extends OrderSagaEvent {  
    private final Long orderId;  
    private final String reason;
```

```
public OrderCancelledEvent(Long orderId, String sagaId, String reason) {  
    super(orderId, sagaId);  
    this orderId = orderId;  
    this reason = reason;  
}
```

```
public Long getOrderId() { return orderId; }  
public String getReason() { return reason; }  
}
```

```
public class PaymentRefundedEvent extends OrderSagaEvent {  
    private final Long orderId;  
    private final String originalTransactionId;  
    private final String refundTransactionId;
```

```
    public PaymentRefundedEvent(Long orderId, String sagaId, String  
originalTransactionId,  
String refundTransactionId) {  
        super(orderId, sagaId);  
        this orderId = orderId;  
        this originalTransactionId = originalTransactionId;  
        this refundTransactionId = refundTransactionId;  
    }
```

```
// геттеры
```

```
public Long getOrderId() { return orderId; }  
public String getOriginalTransactionId() { return originalTransactionId; }  
public String getRefundTransactionId() { return refundTransactionId; }
```

```
}
```

4.2. Сервис заказов (инициатор саги)

// application/service/OrderSagaOrchestrator.java (по факту - координатор через события)

```
package com.logistics.application.service;
```

```
import com.logistics.domain.event.saga OrderCreatedEvent;
```

```
import com.logistics.infrastructure.messaging DomainEventPublisher;
```

```
import lombok.RequiredArgsConstructor;
```

```
import lombok.extern.slf4j.Slf4j;
```

```
import org.springframework.stereotype.Service;
```

```
@Service
```

```
@RequiredArgsConstructor
```

```
@Slf4j
```

```
public class OrderSagaInitiator {
```

```
    private final DomainEventPublisher eventPublisher;
```

```
    private final OrderRepository orderRepository;
```

```
    /**
```

```
     * Начало саги: создание заказа
```

```
     */
```

```
    public Long startOrderSaga(Long customerId, Double amount,
```

```
                                String pickupAddress, String dropoffAddress) {
```

```
        // 1. Создаём заказ со статусом PENDING
```

```
        Order order = new Order(customerId, amount, pickupAddress,  
dropoffAddress);
```

```
        order.setStatus(OrderStatus.PENDING);
```

```

    Order savedOrder = orderRepository.save(order);

    log.info("Saga started: order {} created, waiting for payment",
savedOrder.getId());

// 2. Публикуем событие для сервиса оплаты
OrderCreatedEvent event = new OrderCreatedEvent(
    savedOrder.getId(),
    customerId,
    amount,
    pickupAddress,
    dropoffAddress
);
eventPublisher.publish(event);

return savedOrder.getId();
}

/**
 * Компенсация: отмена заказа (если оплата не удалась)
 */
public void cancelOrder(Long orderId, String reason) {
    Order order = orderRepository.findById(orderId).orElse(null);
    if (order != null && order.getStatus() == OrderStatus.PENDING) {
        order.setStatus(OrderStatus.CANCELLED);
        order.setCancellationReason(reason);
        orderRepository.save(order);
        log.info("Order {} cancelled due to: {}", orderId, reason);
    }
}

```

```
}  
}
```

4.3. Сервис оплаты (обработчик событий)

```
// application/service/PaymentSagaHandler.java
```

```
package com.logistics.application.service;
```

```
import com.logistics.domain.event.saga *;
```

```
import com.logistics.infrastructure.messaging DomainEventPublisher;
```

```
import lombok.RequiredArgsConstructor;
```

```
import lombok.extern.slf4j.Slf4j;
```

```
import org.springframework.stereotype.Service;
```

```
import org.springframework.transaction.event TransactionPhase;
```

```
import org.springframework.transaction.event TransactionalEventListener;
```

```
@Service
```

```
@RequiredArgsConstructor
```

```
@Slf4j
```

```
public class PaymentSagaHandler {
```

```
    private final PaymentService paymentService;
```

```
    private final DomainEventPublisher eventPublisher;
```

```
    /**
```

```
     * Обработка события создания заказа
```

```
     */
```

```
    @TransactionalEventListener(phase
```

```
TransactionPhase.AFTER_COMMIT)
```

```
    public void handleOrderCreated(OrderCreatedEvent event) {
```

=

```
log info("Payment service received OrderCreatedEvent for order {}",  
event.getId());
```

```
try {  
    // Пытаемся провести платёж  
    String transactionId = paymentService.processPayment(  
        event.getId(),  
        event.getAmount()  
    );  
  
    // Успех - публикуем PaymentProcessedEvent  
    PaymentProcessedEvent successEvent = new PaymentProcessedEvent(  
        event.getId(),  
        event.getId(),  
        transactionId,  
        event.getAmount()  
    );  
    eventPublisher.publish(successEvent);  
  
    log info("Payment successful for order {}, transaction {}",  
        event.getId(), transactionId);
```

```
} catch (PaymentFailedException e) {  
    // Ошибка - публикуем PaymentFailedEvent (нужна компенсация)  
    PaymentFailedEvent failedEvent = new PaymentFailedEvent(  
        event.getId(),  
        event.getId(),  
        e.getMessage()  
    );
```



```

eventPublisher.publish(failedEvent);

log.error("Payment failed for order {}: {}", event.getOrderId(),
e.getMessage());
}
}

/**
 * Компенсация: возврат средств (если назначение курьера не удалось)
 */

@TransactionalEventListener(phase =
TransactionPhase.AFTER_COMMIT)
public void handleCourierAssignmentFailed(CourierAssignmentFailedEvent
event) {
    log.info("Payment service received CourierAssignmentFailedEvent for order
{}",
event.getOrderId());

    // Возвращаем средства
    String refundTransactionId = paymentService.refundPayment(
event.getOrderId(),
event.getReason()
);

    // Публикуем событие о возврате
    PaymentRefundedEvent refundEvent = new PaymentRefundedEvent(
event.getOrderId(),
event.getSagaId(),
null, // originalTransactionId можно получить из БД

```

```

refundTransactionId
);
eventPublisher publish(refundEvent);

log info("Refund processed for order {}, refundId {}",
event getOrderId(), refundTransactionId);
}
}

```

4.4. Сервис диспетчеризации (назначение курьера)

// application/service/DispatchSagaHandler.java

```

package com logistics application service;

import com logistics domain event saga *;
import com logistics infrastructure messaging DomainEventPublisher;
import lombok RequiredArgsConstructor;
import lombok extern slf4j Slf4j;
import org springframework stereotype Service;
import org springframework transaction event TransactionPhase;
import org springframework transaction event TransactionalEventListener;

```

```
@Service
```

```
@RequiredArgsConstructor
```

```
@Slf4j
```

```
public class DispatchSagaHandler {
```

```
private final CourierAssignmentService assignmentService;
```

```
private final DomainEventPublisher eventPublisher;
```

```
/**
```

* Обработка успешной оплаты: назначение курьера

*/

```
@TransactionalEventListener(phase = TransactionPhase.AFTER_COMMIT)

public void handlePaymentProcessed(PaymentProcessedEvent event) {
    log.info("Dispatch service received PaymentProcessedEvent for order {}",
        event.getId());

    try {
        // Пытаемся назначить курьера
        AssignmentResult result = assignmentService.assignCourier(
            event.getId(),
            event.getPickupAddress(), // нужно добавить в событие
            event.getDropoffAddress()
        );

        // Успех - публикуем CourierAssignedEvent
        CourierAssignedEvent successEvent = new CourierAssignedEvent(
            event.getId(),
            event.getSagaId(),
            result.getCourierId(),
            result.getAssignmentId(),
            result.getEtaMinutes()
        );
        eventPublisher.publish(successEvent);

        log.info("Courier {} assigned to order {}", result.getCourierId(),
            event.getId());
    }
}
```

```

    } catch (NoCourierAvailableException e) {
        // Ошибка - публикуем CourierAssignmentFailedEvent (нужна
        компенсация оплаты)
        CourierAssignmentFailedEvent failedEvent = new
        CourierAssignmentFailedEvent(
            event getId(),
            event getSagaId(),
            e.getMessage()
        );
        eventPublisher.publish(failedEvent);

        log.error("No courier available for order {}", event getId());
    }
}

/**
 * Компенсация: отмена назначения (если сага отменяется)
 */
@TransactionalEventListener(phase =
TransactionPhase.AFTER_COMMIT)
public void handleOrderCancelled(OrderCancelledEvent event) {
    log.info("Dispatch service received OrderCancelledEvent for order {}",
        event.getId());

    // Отменяем назначение, если оно было
    assignmentService.cancelAssignment(event.getId(),
event.getReason());
}
}

```

4.5. Сервис уведомлений (завершение саги)

```
// application/service/NotificationSagaHandler.java
```

```
package com.logistics.application.service;
```

```
import com.logistics.domain.event.saga *;
```

```
import com.logistics.infrastructure.messaging DomainEventPublisher;
```

```
import lombok.RequiredArgsConstructor;
```

```
import lombok.extern.slf4j.Slf4j;
```

```
import org.springframework.stereotype.Service;
```

```
import org.springframework.transaction.event TransactionPhase;
```

```
import org.springframework.transaction.event TransactionalEventListener;
```

```
@Service
```

```
@RequiredArgsConstructor
```

```
@Slf4j
```

```
public class NotificationSagaHandler {
```

```
    private final NotificationService notificationService;
```

```
    private final DomainEventPublisher eventPublisher;
```

```
    /**
```

```
     * Обработка успешного назначения курьера: отправка уведомления
```

```
     */
```

```
    @TransactionalEventListener(phase
```

```
TransactionPhase.AFTER_COMMIT)
```

```
    public void handleCourierAssigned(CourierAssignedEvent event) {
```

```
        log.info("Notification service received CourierAssignedEvent for order {}",
```

```
event.getId());
```

```

// Отправляем уведомление клиенту
notificationService sendOrderConfirmed(
    event getOrderId(),
    event getCourierId(),
    event getEtaMinutes()
);

// Публикуем финальное событие успешного завершения саги
OrderConfirmedEvent confirmedEvent = new OrderConfirmedEvent(
    event getOrderId(),
    event getSagaId(),
    event getCourierId(),
    event getEtaMinutes()
);
eventPublisher.publish(confirmedEvent);

log info("Order confirmed notification sent for order {}",
event.getId());
}

/**
 * Обработка отмены заказа: уведомление клиента
 */
@TransactionalEventListener(phase =
TransactionPhase.AFTER_COMMIT)
public void handleOrderCancelled(OrderCancelledEvent event) {
    log info("Notification service received OrderCancelledEvent for order {}",
    event.getId());
}

```

```

        notificationService.sendOrderCancelled(event.getId(),
event.getReason());
    }

    /**
     * Обработка возврата средств: уведомление клиента
     */
    @TransactionalEventListener(phase =
TransactionPhase.AFTER_COMMIT)
    public void handlePaymentRefunded(PaymentRefundedEvent event) {
        log.info("Notification service received PaymentRefundedEvent for order
{}",
event.getId());

        notificationService.sendRefundNotification(event.getId(),
event.getRefundTransactionId());
    }
}

```

4.6. Идемпотентность обработчиков

// infrastructure/persistence/ProcessedEventTracker.java

package com.logistics.infrastructure.persistence;

import jakarta.persistence.*;

import lombok.Data;

import java.time.Instant;

@Entity

@Table(name = "processed_events")

@Data

```
public class ProcessedEvent {  
    @Id  
    private String eventId;  
    private String handlerName;  
    private String sagaId;  
    private Instant processedAt;  
}
```

```
// infrastructure/messaging/IdempotentEventHandler.java
```

```
package com logistics infrastructure messaging;
```

```
import lombok RequiredArgsConstructor;
```

```
import org springframework stereotype Component;
```

```
@Component
```

```
@RequiredArgsConstructor
```

```
public class IdempotentEventHandler {
```

```
    private final ProcessedEventRepository processedEventRepository;
```

```
    /**
```

```
     * Проверяет, было ли событие уже обработано
```

```
     */
```

```
    public boolean isAlreadyProcessed(String eventId, String handlerName) {
```

```
        return
```

```
        processedEventRepository.existsByEventIdAndHandlerName(eventId,  
        handlerName);
```

```
    }
```



```

/**
 * Отмечает событие как обработанное
 */

public void markAsProcessed(String eventId, String handlerName, String
sagaId) {
    ProcessedEvent event = new ProcessedEvent();
    event.setEventId(eventId);
    event.setHandlerName(handlerName);
    event.setSagaId(sagaId);
    event.setProcessedAt(Instant.now());
    processedEventRepository.save(event);
}

// Пример использования в обработчике
@TransactionalEventListener(phase
TransactionPhase.AFTER_COMMIT)
public void handlePaymentProcessed(PaymentProcessedEvent event) {
    String eventId = event.getEventId();
    String handlerName = "PaymentSagaHandler.handlePaymentProcessed";

    if (idempotentHandler.isAlreadyProcessed(eventId, handlerName)) {
        log.info("Event {} already processed, skipping", eventId);
        return;
    }

    // бизнес-логика...

```

```
        idempotentHandler markAsProcessed(eventId, handlerName,
event getSagaId());
    }
```

4.7. Тестирование саги

```
// test/saga/OrderSagaChoreographyTest.java
```

```
package com.logistics.test.saga;
```

```
import com.logistics.domain.event.saga *;
import com.logistics.infrastructure.messaging DomainEventPublisher;
import org.junit.jupiter.api Test;
import org.junit.jupiter.api.extension ExtendWith;
import org.mockito ArgumentCaptor;
import org.mockito InjectMocks;
import org.mockito Mock;
import org.mockito.junit.jupiter MockitoExtension;

import static org.assertj.core.api Assertions assertThat;
import static org.mockito.ArgumentMatchers any;
import static org.mockito.Mockito.*;
```

```
@ExtendWith(MockitoExtension class)
```

```
class OrderSagaChoreographyTest {
```

```
@Mock private PaymentService paymentService;
```

```
@Mock private CourierAssignmentService assignmentService;
```

```
@Mock private NotificationService notificationService;
```

```
@Mock private DomainEventPublisher eventPublisher;
```

```
@Mock private OrderRepository orderRepository;
```

```

@InjectMocks private OrderSagaInitiator sagaInitiator;
@InjectMocks private PaymentSagaHandler paymentHandler;
@InjectMocks private DispatchSagaHandler dispatchHandler;
@InjectMocks private NotificationSagaHandler notificationHandler;

@Test
void shouldCompleteSagaSuccessfully() {
    // given
    Long orderId = 1L;
    String sagaId = "saga-123";
    String transactionId = "tx-456";
    Long courierId = 10L;
    Long assignmentId = 100L;
    Integer eta = 15;

    OrderCreatedEvent orderEvent = new OrderCreatedEvent(orderId, 100L,
50.0, "Addr1", "Addr2");
    PaymentProcessedEvent paymentEvent = new
PaymentProcessedEvent(orderId, sagaId, transactionId, 50.0);
    CourierAssignedEvent assignEvent = new CourierAssignedEvent(orderId,
sagaId, courierId, assignmentId, eta);

    // when
    when(paymentService.processPayment(anyLong(), anyDouble()))
        .thenReturn(transactionId);
    when(assignmentService.assignCourier(anyLong(), anyString(),
anyString()))
        .thenReturn(new AssignmentResult(courierId, assignmentId, eta));

```

```

paymentHandler handleOrderCreated(orderEvent);
dispatchHandler handlePaymentProcessed(paymentEvent);
notificationHandler handleCourierAssigned(assignEvent);

// then
verify(paymentService).processPayment(100L, 50.0);
verify(assignmentService).assignCourier(eq(orderId), anyString(),
anyString());
verify(notificationService).sendOrderConfirmed(eq(orderId), eq(courierId),
eq(eta));

ArgumentCaptor<OrderConfirmedEvent> captor =
ArgumentCaptor.forClass(OrderConfirmedEvent.class);
verify(eventPublisher, atLeastOnce()).publish(captor.capture());

assertThat(captor.getValue().getOrderId()).isEqualTo(orderId);
assertThat(captor.getValue().getCourierId()).isEqualTo(courierId);
}

@Test
void shouldCompensateWhenPaymentFails() {
// given
Long orderId = 1L;
OrderCreatedEvent orderEvent = new OrderCreatedEvent(orderId, 100L,
50.0, "Addr1", "Addr2");

// when
when(paymentService.processPayment(anyLong(), anyDouble()))
.thenThrow(new PaymentFailedException("Insufficient funds"));

```

```

paymentHandler handleOrderCreated(orderEvent);

// then
ArgumentCaptor<PaymentFailedEvent> captor =
ArgumentCaptor.forClass(PaymentFailedEvent.class);
verify(eventPublisher).publish(captor.capture());

assertThat(captor.getValue().getOrderId()).isEqualTo(orderId);
assertThat(captor.getValue().getReason()).contains("Insufficient funds");

// Проверяем, что отмена заказа была вызвана
verify(orderRepository).save(any(Order.class));
}

@Test
void shouldCompensateWhenNoCourierAvailable() {
// given
Long orderId = 1L;
String sagaId = "saga-123";
String transactionId = "tx-456";

PaymentProcessedEvent paymentEvent = new
PaymentProcessedEvent(orderId, sagaId, transactionId, 50.0);

// when
when(assignmentService assignCourier(anyLong(), anyString(),
anyString()))
.thenThrow(new NoCourierAvailableException("No courier in zone"));

```

```

dispatchHandler handlePaymentProcessed(paymentEvent);

// then
ArgumentCaptor<CourierAssignmentFailedEvent> captor =
ArgumentCaptor.forClass(CourierAssignmentFailedEvent.class);
verify(eventPublisher).publish(captor.capture());

assertThat(captor.getValue().getOrderId()).isEqualTo(orderId);
assertThat(captor.getValue().getReason()).contains("No courier in zone");

// Проверяем, что компенсация оплаты была вызвана
verify(paymentService).refundPayment(eq(orderId), anyString());
}
}

```

4.8. Диаграмма последовательности (хореография)

@startuml

!theme plain

participant "Заказчик" as Client

participant "Сервис\nЗаказов" as OrderService

participant "Сервис\nОплаты" as PaymentService

participant "Сервис\nДиспетчеризации" as DispatchService

participant "Сервис\nУведомлений" as NotificationService

Client -> OrderService: Создать заказ

activate OrderService

OrderService -> OrderService: Создать заказ (PENDING)

OrderService -> PaymentService: OrderCreatedEvent

deactivate OrderService

activate PaymentService

PaymentService -> PaymentService: Обработка платежа

alt Успешная оплата

PaymentService -> DispatchService: PaymentProcessedEvent

deactivate PaymentService

activate DispatchService

DispatchService -> DispatchService: Назначить курьера

alt Курьер назначен

DispatchService -> NotificationService: CourierAssignedEvent

deactivate DispatchService

activate NotificationService

NotificationService -> NotificationService: Отправить уведомление

NotificationService -> OrderService: OrderConfirmedEvent

deactivate NotificationService

else Нет курьера

DispatchService -> PaymentService: CourierAssignmentFailedEvent

deactivate DispatchService

activate PaymentService

PaymentService -> PaymentService: Возврат средств

PaymentService -> NotificationService: PaymentRefundedEvent

deactivate PaymentService

activate NotificationService

NotificationService -> OrderService: OrderCancelledEvent
deactivate NotificationService
end

else Ошибка оплаты
PaymentService -> OrderService: PaymentFailedEvent
deactivate PaymentService

activate OrderService
OrderService -> OrderService: Отмена заказа
OrderService -> NotificationService: OrderCancelledEvent
deactivate OrderService

activate NotificationService
NotificationService -> NotificationService: Уведомление об отмене
deactivate NotificationService
end

@enduml

5. Индивидуальные задания (15 вариантов)

Важно: Студент работает со своим сквозным проектом. Для каждого варианта указан бизнес-процесс для саги.

В ариант	Сквозно й проект	Бизнес- процесс (Saga)	Шаги саги
1	Логисти ка	Оформле ние доставки	Заказ → Оплата → Назначение курьера →

Вариант	Сквозной проект	Бизнес-процесс (Saga)	Шаги саги
			Уведомление
2	Интернет-магазин	Оформление заказа	Корзина → Оплата → Резерв товара → Доставка
3	Финтех-кошелёк	Перевод средств	Списание → Блокировка → Перевод → Зачисление
4	Библиотека	Выдача книги	Проверка читателя → Резерв книги → Выдача → Уведомление
5	Отели	Бронирование номера	Поиск → Резерв → Оплата → Подтверждение
6	Онлайн-курсы	Запись на курс	Проверка мест → Оплата → Зачисление → Выдача доступа
7	Медицина	Запись к врачу	Выбор врача → Резерв времени → Подтверждение → Напоминание
8	CRM	Создание заказа	Проверка клиента → Создание заказа → Резерв товара → Счёт

Вариант	Сквозной проект	Бизнес-процесс (Saga)	Шаги саги
9	Платёжный шлюз	Обработка транзакции	Авторизация → Списание → Клиринг → Уведомление
10	Склад (WMS)	Отгрузка товара	Резерв → Сборка → Упаковка → Отгрузка
11	Социальная сеть	Публикация поста	Создание → Модерация → Публикация → Уведомление
12	Видеохостинг	Загрузка видео	Загрузка → Транскодинг → Проверка → Публикация
13	IoT-мониторинг	Обработка аларма	Получение → Фильтрация → Оповещение → Эскалация
14	Криптовалюта	Исполнение ордера	Проверка счёта → Блокировка → Сопоставление → Исполнение
15	Доставка еды	Заказ в ресторане	Приём заказа → Подтверждение → Готовка → Доставка

Задание для студента:

Для своего проекта (по варианту):

1. Определить бизнес-процесс из 3–5 шагов.
2. Создать события для каждого шага (успех + отказ).
3. Реализовать обработчики для каждого сервиса.
4. Реализовать компенсирующие транзакции для отката.
5. Реализовать идемпотентность обработчиков.
6. Написать тесты для успешного сценария и сценария с откатом.
7. Построить диаграмму последовательности.

6. Контрольные вопросы

1. В чём отличие Saga (хореография) от Saga (оркестрация)?
2. Что такое компенсирующая транзакция? Приведите пример.
3. Зачем нужна идемпотентность в обработчиках событий саги?
4. Как обеспечить идемпотентность при публикации событий?
5. Как обрабатывать долго выполняющиеся шаги саги?
6. Что произойдёт, если обработчик события упадёт после частичного выполнения?
7. Как отслеживать состояние саги в хореографии?
8. В каких случаях хореография предпочтительнее оркестрации?

7. Требования к отчёту

Отчёт должен содержать:

Титульный лист.

Цель и задачи работы.

Описание бизнес-процесса (диаграмма шагов).

События саги (код).

Обработчики каждого сервиса (код).

Компенсирующие транзакции.

Идемпотентность (реализация).

Тесты (успех + откат).

Диаграмма последовательности.

Вывод.

Ответы на вопросы.

8. Критерии оценивания

Критерий	max балл
Описание бизнес-процесса (3–5 шагов)	1
События для каждого шага (минимум 3 типа)	2
Обработчики событий (все сервисы)	3
Компенсирующие транзакции (откат при сбоях)	3
Идемпотентность обработчиков	2
Тесты (успешный сценарий + откат)	3
Диаграмма последовательности	1
Соответствие варианту	1
Итого	16

Лабораторная работа №13

Saga (оркестрация) с координатором

на примере «Системы управления логистикой и доставкой»

1. Цель работы

Сформировать компетенции по реализации распределённых транзакций с использованием паттерна Saga в стиле оркестрации: централизованное управление шагами саги через координатор (оркестратор), обработка сбоев, выполнение компенсирующих транзакций и сравнение с хореографией для выбранного сквозного проекта.

2. Задачи работы

1. Изучить паттерн Saga (оркестрация) и его отличие от хореографии.
2. Реализовать оркестратор, управляющий шагами распределённой транзакции.
3. Реализовать шаги саги и их компенсации.
4. Реализовать сохранение состояния саги для восстановления после сбоев.
5. Обеспечить идемпотентность выполнения шагов.
6. Написать тесты, проверяющие успешное выполнение и откат саги.
7. Сравнить с хореографией (ЛР 12) и сделать выводы.
8. Оформить результаты в виде отчёта с кодом, диаграммами и сравнением.

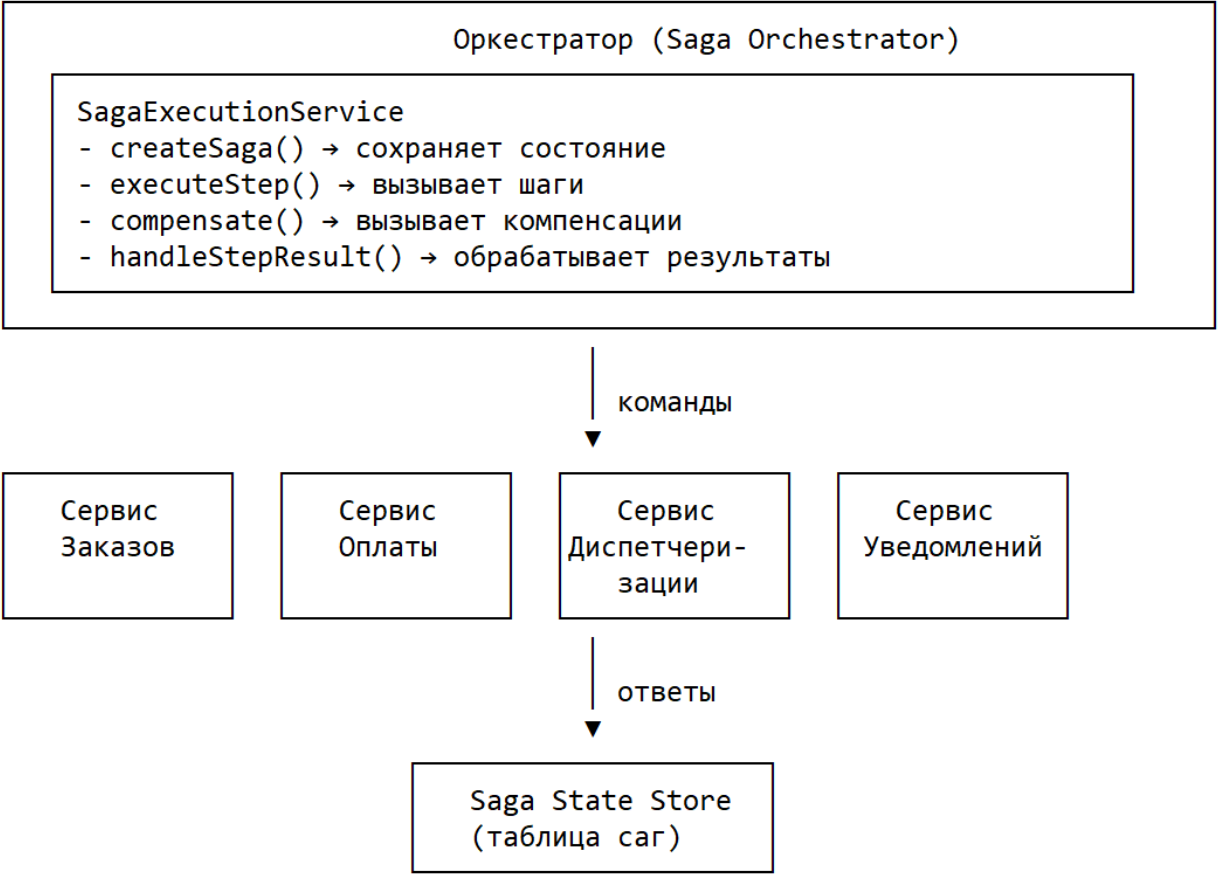
3. Теоретическое обоснование

3.1. Оркестрация vs Хореография

Характеристика	Хореография (ЛР 12)	Оркестрация (ЛР 13)
Управление	Децентрализованное	Централизованное

Характеристика	Хореография (ЛР 12)	Оркестрация (ЛР 13)
		нное (оркестратор)
Знание процесса	Каждый сервис знает только свои шаги	Оркестратор знает весь процесс
Отслеживание	Сложно (события разбросаны)	Легко (один оркестратор)
Отладка	Трудно	Проще
Точка отказа	Нет единой точки	Оркестратор может стать точкой отказа
Связанность	Слабая	Средняя
Изменение процесса	Нужно менять несколько сервисов	Только оркестратор

3.2. Архитектура саги с оркестратором



3.3. Состояния саги

Состояние	Описание
STARTED	Сага создана, начато выполнение
STEP_EXECUTING	Выполняется текущий шаг
COMPLETED	Все шаги успешно выполнены
COMPENSATING	Выполняется компенсация (откат)
FAILED	Сага завершилась с ошибкой

Состояние	Описание
COMPENSATED	Компенсация завершена

3.4. Команды vs События

Хореография	Оркестрация
Публикует события (OrderCreatedEvent)	Отправляет команды (CreateOrderCommand)
Сервисы реагируют на события	Сервисы выполняют команды и возвращают результат
Компенсация - отдельное событие	Компенсация - команда отката

4. Пример выполнения

Сквозной пример: «Система управления логистикой и доставкой»
(вариант 1)

Бизнес-процесс: Оформление заказа на доставку (те же шаги, что в ЛР 12)

4.1. Модель саги (состояние)

// domain/model/SagaInstance.java

package com.logistics.domain.model;

import jakarta.persistence *;

import lombok.Data;

import java.time.Instant;

import java.util.ArrayList;


```
import java.util List;

@Entity
@Table(name = "saga_instances")
@Data
public class SagaInstance {

    @Id
    private String sagaId;

    private String sagaType;        // OrderSaga

    private Long businessKey;       // orderId

    @Enumerated(EnumType.STRING)
    private SagaState state;        // STARTED, COMPLETED,
    COMPENSATING, etc.

    private Integer currentStepIndex; // индекс текущего шага (0-based)

    @Enumerated(EnumType.STRING)
    private StepStatus currentStepStatus, // EXECUTING, COMPLETED,
    FAILED

    @Column(length = 2000)
    private String sagaData,        // JSON с данными саги

    private String lastError,
```

```
private Integer retryCount;
```

```
private Instant createdAt;
```

```
private Instant updatedAt;
```

```
@OneToMany(cascade = CascadeType.ALL, fetch = FetchType.EAGER)
```

```
@JoinColumn(name = "saga_id")
```

```
private List<SagaStepExecution> stepExecutions = new ArrayList<>();
```

```
}
```

```
// domain/model/SagaState.java
```

```
public enum SagaState {
```

```
    STARTED,    // saga запущена
```

```
    EXECUTING,  // выполняется шаг
```

```
    COMPLETED, // успешно завершена
```

```
    COMPENSATING, // выполняется компенсация
```

```
    COMPENSATED, // компенсация завершена
```

```
    FAILED      // saga провалилась
```

```
}
```

```
// domain/model/StepStatus.java
```

```
public enum StepStatus {
```

```
    PENDING, EXECUTING, COMPLETED, FAILED, COMPENSATING,  
    COMPENSATED
```

```
}
```

```
// domain/model/SagaStepExecution.java
```

```
@Entity
```

```
@Table(name = "saga_step_executions")
```

```

@Data
public class SagaStepExecution {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private String sagaId;
    private Integer stepIndex;
    private String stepName;

    @Enumerated(EnumType.STRING)
    private StepStatus status;

    private String requestData; // JSON запроса
    private String responseData; // JSON ответа
    private String errorMessage;
    private Instant startedAt;
    private Instant completedAt;
}

```

4.2. SQL для таблиц саги

-- Таблица экземпляров саг

```

CREATE TABLE saga_instances (
    saga_id VARCHAR(36) PRIMARY KEY,
    saga_type VARCHAR(50) NOT NULL,
    business_key BIGINT NOT NULL,
    state VARCHAR(20) NOT NULL,
    current_step_index INT DEFAULT 0,
    current_step_status VARCHAR(20),

```

```

saga_data TEXT,
last_error TEXT,
retry_count INT DEFAULT 0,
created_at TIMESTAMP NOT NULL,
updated_at TIMESTAMP NOT NULL
);

CREATE INDEX idx_saga_state ON saga_instances(state);
CREATE INDEX idx_saga_business_key ON
saga_instances(business_key);

```

-- Таблица выполнения шагов

```

CREATE TABLE saga_step_executions (
  id BIGINT AUTO_INCREMENT PRIMARY KEY,
  saga_id VARCHAR(36) NOT NULL,
  step_index INT NOT NULL,
  step_name VARCHAR(100) NOT NULL,
  status VARCHAR(20) NOT NULL,
  request_data TEXT,
  response_data TEXT,
  error_message TEXT,
  started_at TIMESTAMP,
  completed_at TIMESTAMP,
  FOREIGN KEY (saga_id) REFERENCES saga_instances(saga_id)
);

```

```

CREATE INDEX idx_step_saga ON saga_step_executions(saga_id);

```

4.3. Базовая команда (общий интерфейс)

```
// application/command/SagaCommand.java  
package com logistics application command;
```

```
public interface SagaCommand {  
    String getCommandType();  
    String getSagaId();  
    Long getBusinessKey();  
}
```

```
// application/command/CreateOrderCommand.java  
package com logistics application command;
```

```
import lombok AllArgsConstructor;  
import lombok Data;  
import lombok NoArgsConstructor;
```

```
@Data
```

```
@AllArgsConstructor
```

```
@NoArgsConstructor
```

```
public class CreateOrderCommand implements SagaCommand {
```

```
    private String sagaId;  
    private Long businessKey; // orderId  
    private Long customerId;  
    private Double amount;  
    private String pickupAddress;  
    private String dropoffAddress;
```

```
@Override
```

```
public String getCommandType() {
```

```
        return "CREATE_ORDER";
    }
}
```

```
// application/command/ProcessPaymentCommand.java
```

```
@Data
```

```
@AllArgsConstructor
```

```
@NoArgsConstructor
```

```
public class ProcessPaymentCommand implements SagaCommand {
```

```
    private String sagaId;
```

```
    private Long businessKey;
```

```
    private Long customerId;
```

```
    private Double amount;
```

```
@Override
```

```
public String getCommandType() {
```

```
    return "PROCESS_PAYMENT";
```

```
}
```

```
}
```

```
// application/command/AssignCourierCommand.java
```

```
@Data
```

```
@AllArgsConstructor
```

```
@NoArgsConstructor
```

```
public class AssignCourierCommand implements SagaCommand {
```

```
    private String sagaId;
```

```
    private Long businessKey;
```

```
    private String pickupAddress;
```

```
    private String dropoffAddress;
```

```

@Override
public String getCommandType() {
    return "ASSIGN_COURIER";
}
}

```

// application/command/SendNotificationCommand.java

```

@Data
@AllArgsConstructor
@NoArgsConstructor
public class SendNotificationCommand implements SagaCommand {
    private String sagaId;
    private Long businessKey;
    private Long courierId;
    private Integer etaMinutes;

    @Override
    public String getCommandType() {
        return "SEND_NOTIFICATION";
    }
}

```

4.4. Результаты команд (успех/ошибка)

// application/result/SagaStepResult.java

```
package com logistics application result;
```

```
import lombok AllArgsConstructor;
```

```
import lombok Data;
```

```
@Data
@AllArgsConstructor
public class SagaStepResult {
    private boolean success;
    private String stepName;
    private Object data;
    private String errorMessage;

    public static SagaStepResult success(String stepName, Object data) {
        return new SagaStepResult(true, stepName, data, null);
    }

    public static SagaStepResult failure(String stepName, String
errorMessage) {
        return new SagaStepResult(false, stepName, null, errorMessage);
    }
}
```

// application/result/CreateOrderResult.java

```
@Data
@AllArgsConstructor
public class CreateOrderResult {
    private Long orderId;
    private String status;
}
```

// application/result/ProcessPaymentResult.java

```
@Data
```



```
@AllArgsConstructor  
public class ProcessPaymentResult {  
    private String transactionId;  
    private Double amount;  
    private String status;  
}
```

```
// application/result/AssignCourierResult.java
```

```
@Data  
@AllArgsConstructor  
public class AssignCourierResult {  
    private Long courierId;  
    private Long assignmentId;  
    private Integer etaMinutes;  
}
```

4.5. Сервисы-исполнители (выполняют команды)

```
// application/service/OrderCommandExecutor.java
```

```
package com logistics application service;
```

```
import com logistics application command CreateOrderCommand;  
import com logistics application result CreateOrderResult;  
import com logistics application result SagaStepResult;  
import lombok RequiredArgsConstructor;  
import lombok extern slf4j Slf4j;  
import org springframework stereotype Service;
```

```
@Service  
@RequiredArgsConstructor
```

@Slf4j

```
public class OrderCommandExecutor {

    private final OrderRepository orderRepository;

    public SagaStepResult execute(CreateOrderCommand command) {
        try {
            log info("Executing CREATE_ORDER for saga {}, order {}",
                command.getSagaId(), command.getBusinessKey());

            Order order = new Order(
                command.getBusinessKey(),
                command.getCustomerId(),
                command.getAmount(),
                command.getPickupAddress(),
                command.getDropoffAddress()
            );
            order.setStatus(OrderStatus.PENDING);
            orderRepository.save(order);

            CreateOrderResult result = new CreateOrderResult(
                order.getId(),
                "PENDING"
            );

            return SagaStepResult.success("CREATE_ORDER", result);

        } catch (Exception e) {
            log error("CREATE_ORDER failed: {}", e.getMessage());
        }
    }
}
```

```

        return SagaStepResult.failure("CREATE_ORDER", e.getMessage());
    }
}

public SagaStepResult compensate(CreateOrderCommand command) {
    try {
        log.info("Compensating CREATE_ORDER for saga {}, order {}",
            command.getSagaId(), command.getBusinessKey());

        orderRepository.findById(command.getBusinessKey()).ifPresent(order -> {
            order.setStatus(OrderStatus.CANCELLED);
            order.setCancellationReason("Saga compensation");
            orderRepository.save(order);
        });

        return
        SagaStepResult.success("COMPENSATE_CREATE_ORDER", null);

    } catch (Exception e) {
        log.error("Compensation failed: {}", e.getMessage());
        return
        SagaStepResult.failure("COMPENSATE_CREATE_ORDER", e.getMessage());
    }
}

// application/service/PaymentCommandExecutor.java
package com.logistics.application.service;

```

```
import com logistics application command ProcessPaymentCommand;
import com logistics application result ProcessPaymentResult;
import com logistics application result SagaStepResult;
import lombok RequiredArgsConstructor;
import lombok.extern slf4j.Slf4j;
import org.springframework.stereotype Service;

@Service
@RequiredArgsConstructor
@Slf4j
public class PaymentCommandExecutor {

    private final PaymentService paymentService;

    public SagaStepResult execute(ProcessPaymentCommand command) {
        try {
            log.info("Executing PROCESS_PAYMENT for saga {}, order {}",
                command.getSagaId(), command.getBusinessKey());

            String transactionId = paymentService.processPayment(
                command.getCustomerId(),
                command.getAmount()
            );

            ProcessPaymentResult result = new ProcessPaymentResult(
                transactionId,
                command.getAmount(),
                "SUCCESS"
            );
        }
    }
}
```

```

    );

    return SagaStepResult.success("PROCESS_PAYMENT", result);

} catch (PaymentFailedException e) {
    log.error("PROCESS_PAYMENT failed: {}", e.getMessage());
    return SagaStepResult.failure("PROCESS_PAYMENT",
e.getMessage());
}
}

public SagaStepResult compensate(ProcessPaymentCommand command)
{
    try {
        log.info("Compensating PROCESS_PAYMENT for saga {}, order
{}",

        command.getSagaId(), command.getBusinessKey());

        String refundId = paymentService.refundPayment(
            command.getBusinessKey(),
            "Saga compensation"
        );

        return
SagaStepResult.success("COMPENSATE_PROCESS_PAYMENT", refundId);

    } catch (Exception e) {
        log.error("Payment compensation failed: {}", e.getMessage());
    }
}

```

```

        return
        SagaStepResult.failure("COMPENSATE_PROCESS_PAYMENT",
        e.getMessage());
    }
}
}

```

// application/service/DispatchCommandExecutor.java

package com logistics application service;

import com logistics application command AssignCourierCommand;

import com logistics application result AssignCourierResult;

import com logistics application result SagaStepResult;

import lombok RequiredArgsConstructor;

import lombok.extern slf4j.Slf4j;

import org springframework stereotype.Service;

@Service

@RequiredArgsConstructor

@Slf4j

public class DispatchCommandExecutor {

private final CourierAssignmentService assignmentService;

public SagaStepResult execute(AssignCourierCommand command) {

try {

log.info("Executing ASSIGN_COURIER for saga {}, order {}",

command.getSagaId(), command.getBusinessKey());

```

AssignmentResult result = assignmentService assignCourier(
    command getBusinessKey(),
    command getPickupAddress(),
    command getDropoffAddress()
);

AssignCourierResult assignResult = new AssignCourierResult(
    result getCourierId(),
    result getAssignmentId(),
    result getEtaMinutes()
);

return SagaStepResult.success("ASSIGN_COURIER",
assignResult);

} catch (NoCourierAvailableException e) {
    log error("ASSIGN_COURIER failed: {}", e.getMessage());
    return SagaStepResult.failure("ASSIGN_COURIER",
e.getMessage());
}
}

public SagaStepResult compensate(AssignCourierCommand command) {
    try {
        log info("Compensating ASSIGN_COURIER for saga {}, order {}",
            command getSagaId(), command getBusinessKey());

        assignmentService cancelAssignment(command getBusinessKey(),
"Saga compensation");
    }
}

```

```

        return
SagaStepResult success("COMPENSATE_ASSIGN_COURIER", null);

    } catch (Exception e) {
        log error("Dispatch compensation failed: {}", e.getMessage());
        return
SagaStepResult failure("COMPENSATE_ASSIGN_COURIER", e.getMessage());
    }
}
}
}

```

4.6. Оркестратор (Saga Orchestrator)

```

// application/orchestrator/OrderSagaOrchestrator.java
package com logistics application orchestrator;

import com fasterxml jackson databind.ObjectMapper;
import com logistics application command *;
import com logistics application result SagaStepResult;
import com logistics application service *;
import com logistics domain model SagaInstance;
import com logistics domain model SagaState;
import com logistics domain model StepStatus;
import com logistics infrastructure persistence SagaInstanceRepository;
import lombok RequiredArgsConstructor;
import lombok extern slf4j Slf4j;
import org springframework stereotype Service;
import org springframework transaction annotation Transactional;

```



```
import java time Instant;
import java util HashMap;
import java util Map;
import java util UUID;
```

```
@Service
```

```
@RequiredArgsConstructor
```

```
@Slf4j
```

```
public class OrderSagaOrchestrator {
```

```
    private final SagaInstanceRepository sagaRepository;
    private final OrderCommandExecutor orderExecutor;
    private final PaymentCommandExecutor paymentExecutor;
    private final DispatchCommandExecutor dispatchExecutor;
    private final NotificationCommandExecutor notificationExecutor;
    private final ObjectMapper objectMapper;
```

```
    // Определение шагов саги
```

```
    private enum SagaStep {
        CREATE_ORDER(0, "CREATE_ORDER", true),
        PROCESS_PAYMENT(1, "PROCESS_PAYMENT", true),
        ASSIGN_COURIER(2, "ASSIGN_COURIER", true),
        SEND_NOTIFICATION(3, "SEND_NOTIFICATION", false);    //
```

финальный шаг без компенсации

```
        final int index;
        final String name;
        final boolean compensable;
```

```

SagaStep(int index, String name, boolean compensable) {
    this.index = index;
    this.name = name;
    this.compensable = compensable;
}

static SagaStep fromIndex(int index) {
    for (SagaStep step : values()) {
        if (step.index == index) return step;
    }
    return null;
}

}

/**
 * Запуск саги
 */
@Transactional
public String startSaga(Long orderId, Long customerId, Double amount,
                        String pickupAddress, String dropoffAddress) {
    String sagaId = UUID.randomUUID().toString();

    // Сохраняем данные саги
    Map<String, Object> sagaData = new HashMap<>();
    sagaData.put("orderId", orderId);
    sagaData.put("customerId", customerId);
    sagaData.put("amount", amount);
    sagaData.put("pickupAddress", pickupAddress);

```

```
sagaData.put("dropoffAddress", dropoffAddress);
```

```
SagaInstance saga = new SagaInstance();
```

```
saga.setSagaId(sagaId);
```

```
saga.setSagaType("ORDER_SAGA");
```

```
saga.setBusinessKey(orderId);
```

```
saga.setState(SagaState.STARTED);
```

```
saga.setCurrentStepIndex(0);
```

```
saga.setCurrentStepStatus(StepStatus.PENDING);
```

```
saga.setSagaData(toJson(sagaData));
```

```
saga.setRetryCount(0);
```

```
saga.setCreatedAt(Instant.now());
```

```
saga.setUpdatedAt(Instant.now());
```

```
sagaRepository.save(saga);
```

```
log.info("Saga started: {} for order {}", sagaId, orderId);
```

```
// Асинхронно начинаем выполнение
```

```
executeSagaAsync(sagaId);
```

```
return sagaId;
```

```
}
```

```
/**
```

```
 * Выполнение саги (пошагово)
```

```
 */
```

```
@Transactional
```

```
public void executeSaga(String sagaId) {
```

```
SagaInstance saga = sagaRepository.findById(sagaId)
    .orElseThrow(() -> new IllegalArgumentException("Saga not found:
" + sagaId));
```

```
if (saga.getState() == SagaState.COMPLETED) {
    log.info("Saga {} already completed", sagaId);
    return;
}
```

```
if (saga.getState() == SagaState.COMPENSATING) {
    log.info("Saga {} is compensating, skipping execution", sagaId);
    return;
}
```

```
int currentStep = saga.getCurrentStepIndex();
SagaStep step = SagaStep.fromIndex(currentStep);
```

```
if (step == null) {
    // Все шаги выполнены
    completeSaga(saga);
    return;
}
```

```
log.info("Executing step {} for saga {}", step.name, sagaId);
```

```
// Обновляем статус
saga.setCurrentStepStatus(StepStatus.EXECUTING);
saga.setState(SagaState.EXECUTING);
saga.setUpdatedAt(Instant.now());
```

```
sagaRepository save(saga);

// Выполняем шаг
SagaStepResult result = executeStep(step, saga);

if (result.isSuccess()) {
    // Шаг успешен - переходим к следующему
    saga setCurrentStepIndex(currentStep + 1);
    saga setCurrentStepStatus(StepStatus.COMPLETED);
    saga setUpdatedAt(Instant.now());
    sagaRepository save(saga);

    // Сохраняем результат в sagaData
    updateSagaDataWithResult(saga, step, result);

    // Продолжаем выполнение
    executeSaga(sagaId);

} else {
    // Шаг провалился - начинаем компенсацию
    log error("Step {} failed for saga {}: {}", step name, sagaId,
result getErrorMessage());

    saga setLastError(result getErrorMessage());
    saga setState(SagaState.COMPENSATING);
    saga setUpdatedAt(Instant.now());
    sagaRepository save(saga);

    startCompensation(sagaId);
}
```

```
}
```

```
/**
```

```
 * Выполнение конкретного шага
```

```
 */
```

```
private SagaStepResult executeStep(SagaStep step, SagaInstance saga) {
```

```
    Map<String, Object> data = fromJson(saga.getSagaData());
```

```
    switch (step) {
```

```
        case CREATE_ORDER:
```

```
            CreateOrderCommand cmd = new CreateOrderCommand(
```

```
                saga.getSagaId(),
```

```
                (Long) data.get("orderId"),
```

```
                (Long) data.get("customerId"),
```

```
                (Double) data.get("amount"),
```

```
                (String) data.get("pickupAddress"),
```

```
                (String) data.get("dropoffAddress")
```

```
            );
```

```
            return orderExecutor.execute(cmd);
```

```
        case PROCESS_PAYMENT:
```

```
            ProcessPaymentCommand paymentCmd = new
```

```
ProcessPaymentCommand(
```

```
                saga.getSagaId(),
```

```
                (Long) data.get("orderId"),
```

```
                (Long) data.get("customerId"),
```

```
                (Double) data.get("amount")
```

```
            );
```

```
            return paymentExecutor.execute(paymentCmd);
```

```

        case ASSIGN_COURIER:

            AssignCourierCommand assignCmd = new
AssignCourierCommand(
            saga.getSagaId(),
            (Long) data.get("orderId"),
            (String) data.get("pickupAddress"),
            (String) data.get("dropoffAddress")
        );

        return dispatchExecutor.execute(assignCmd);

        case SEND_NOTIFICATION:

            // Извлекаем ID курьера из предыдущего шага
            Long courierId = (Long) data.get("assignedCourierId");
            Integer eta = (Integer) data.get("etaMinutes");

            SendNotificationCommand notifyCmd = new
SendNotificationCommand(
            saga.getSagaId(),
            (Long) data.get("orderId"),
            courierId
            eta
        );

        return notificationExecutor.execute(notifyCmd);

        default:

            return SagaStepResult.failure(step name "Unknown step");
    }
}

```

```
/**
 * Запуск компенсации (откат)
 */

@Transactional
public void startCompensation(String sagaId) {
    SagaInstance saga = sagaRepository.findById(sagaId).orElse(null);
    if (saga == null) return;

    log.info("Starting compensation for saga {}", sagaId);
    saga.setState(SagaState.COMPENSATING);
    saga.setUpdatedAt(Instant.now());
    sagaRepository.save(saga);

    compensate(sagaId, saga.getCurrentStepIndex() - 1);
}

/**
 * Компенсация шагов в обратном порядке
 */

@Transactional
public void compensate(String sagaId, int stepIndex) {
    if (stepIndex < 0) {
        // Все шаги скомпенсированы
        SagaInstance saga = sagaRepository.findById(sagaId).orElse(null);
        if (saga != null) {
            saga.setState(SagaState.COMPENSATED);
            saga.setUpdatedAt(Instant.now());
            sagaRepository.save(saga);
        }
    }
}
```



```

        log info("Saga {} fully compensated", sagaId);
    }

    return;
}

SagaInstance saga = sagaRepository.findById(sagaId).orElse(null);
if (saga == null) return;

SagaStep step = SagaStep.fromIndex(stepIndex);
if (step == null || !step.compensable) {
    // Пропускаем некомпенсируемые шаги
    compensate(sagaId, stepIndex - 1);
    return;
}

log info("Compensating step {} for saga {}", step.name, sagaId);

SagaStepResult result = compensateStep(step, saga);

if (result.isSuccess()) {
    // Компенсация успешна - переходим к следующему шагу
    compensate(sagaId, stepIndex - 1);
} else {
    // Компенсация провалилась - требуется ручное вмешательство
    saga.setState(SagaState.FAILED);
    saga.setLastError("Compensation failed for step " + step.name + ": "
+ result.getErrorMessage());
    saga.setUpdatedAt(Instant.now());
    sagaRepository.save(saga);
}

```

```

        log error("Compensation failed for saga {} at step {}", sagaId,
step name);
    }
}

/**
 * Компенсация конкретного шага
 */
private SagaStepResult compensateStep(SagaStep step, SagaInstance
saga) {
    Map<String, Object> data = fromJson(saga getSagaData());

    switch (step) {
        case CREATE_ORDER:
            CreateOrderCommand cmd = new CreateOrderCommand(
                saga getSagaId(),
                (Long) data get("orderId"),
                (Long) data get("customerId"),
                (Double) data get("amount"),
                (String) data get("pickupAddress"),
                (String) data get("dropoffAddress")
            );
            return orderExecutor.compensate(cmd);

        case PROCESS_PAYMENT:
            ProcessPaymentCommand paymentCmd = new
ProcessPaymentCommand(
                saga getSagaId(),
                (Long) data get("orderId"),

```

```

        (Long) data get("customerId"),
        (Double) data get("amount")
    );
    return paymentExecutor.compensate(paymentCmd);

    case ASSIGN_COURIER:
        AssignCourierCommand assignCmd = new
AssignCourierCommand(
        saga.getSagaId(),
        (Long) data get("orderId"),
        (String) data get("pickupAddress"),
        (String) data get("dropoffAddress")
    );
    return dispatchExecutor.compensate(assignCmd);

    default:
        return SagaStepResult.success("NO_OP", null);
    }
}

/**
 * Завершение саги (успех)
 */
private void completeSaga(SagaInstance saga) {
    saga.setState(SagaState.COMPLETED);
    saga.setCurrentStepStatus(StepStatus.COMPLETED);
    saga.setUpdatedAt(Instant.now());
    sagaRepository.save(saga);
    log.info("Saga {} completed successfully", saga.getSagaId());
}

```

```

    }

    /**
     * Асинхронное выполнение (упрощённо - синхронно, в реальности -
    через @Async)
     */

    private void executeSagaAsync(String sagaId) {
        // В реальном проекте - @Async или через очередь
        executeSaga(sagaId);
    }

    private void updateSagaDataWithResult(SagaInstance saga, SagaStep
step, SagaStepResult result) {
        Map<String, Object> data = fromJson(saga.getSagaData());

        switch (step) {
            case PROCESS_PAYMENT:
                if (result.getData() instanceof ProcessPaymentResult
paymentResult) {
                    data.put("transactionId", paymentResult.getTransactionId());
                }
                break;

            case ASSIGN_COURIER:
                if (result.getData() instanceof AssignCourierResult assignResult) {
                    data.put("assignedCourierId", assignResult.getCourierId());
                    data.put("assignmentId", assignResult.getAssignmentId());
                    data.put("etaMinutes", assignResult.getEtaMinutes());
                }
        }
    }

```

```

        break;
    }

    saga.setSagaData(toJson(data));
}

private String toJson(Object obj) {
    try {
        return objectMapper.writeValueAsString(obj);
    } catch (Exception e) {
        return "{}";
    }
}

@SuppressWarnings("unchecked")
private Map<String, Object> fromJson(String json) {
    try {
        return objectMapper.readValue(json, Map.class);
    } catch (Exception e) {
        return new HashMap<>();
    }
}
}

```

4.7. REST API для запуска саги

```

// infrastructure/web/SagaController.java
package com.logistics.infrastructure.web;

import com.logistics.application.dto.StartSagaRequest;

```

```
import com logistics application orchestrator OrderSagaOrchestrator;
import com logistics domain model SagaInstance;
import com logistics infrastructure persistence SagaInstanceRepository;
import lombok RequiredArgsConstructor;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation *;
```

```
import java.util Map;
```

```
@RestController
@RequestMapping("/api/saga")
@RequiredArgsConstructor
public class SagaController {
```

```
    private final OrderSagaOrchestrator orchestrator;
    private final SagaInstanceRepository sagaRepository;
```

```
    @PostMapping("/order")
    public ResponseEntity<Map<String, String>>
startOrderSaga(@RequestBody StartSagaRequest request) {
        String sagaId = orchestrator.startSaga(
            request.getId(),
            request.getCustomerId(),
            request.getAmount(),
            request.getPickupAddress(),
            request.getDropoffAddress()
        );

        return ResponseEntity.ok(Map.of("sagaId", sagaId));
    }
}
```

```

    }

    @GetMapping("/{sagaId}")
    public ResponseEntity<SagaInstance> getSagaStatus(@PathVariable
String sagaId) {
        return sagaRepository findById(sagaId)
            .map(ResponseEntity::ok)
            .orElse(ResponseEntity.notFound().build());
    }

    @PostMapping("/{sagaId}/retry")
    public ResponseEntity<Void> retrySaga(@PathVariable String sagaId) {
        orchestrator.executeSaga(sagaId);
        return ResponseEntity.ok().build();
    }
}

```

4.8. Тестирование саги

// test/orchestrator/OrderSagaOrchestratorTest.java

package com logistics test orchestrator;

```

import com logistics application orchestrator OrderSagaOrchestrator;
import com logistics application service.*;
import com logistics domain model SagaInstance;
import com logistics domain model SagaState;
import com logistics infrastructure persistence SagaInstanceRepository;
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 Optional;

import static org.assertj.core.api Assertions assertThat;
import static org.mockito.ArgumentMatchers any;
import static org.mockito Mockito.*;

@ExtendWith(MockitoExtension.class)
class OrderSagaOrchestratorTest {

    @Mock private SagaInstanceRepository sagaRepository;
    @Mock private OrderCommandExecutor orderExecutor;
    @Mock private PaymentCommandExecutor paymentExecutor;
    @Mock private DispatchCommandExecutor dispatchExecutor;
    @Mock private NotificationCommandExecutor notificationExecutor;

    @InjectMocks
    private OrderSagaOrchestrator orchestrator;

    @Test
    void shouldCompleteSagaSuccessfully() {
        // given
        String sagaId = "saga-123";
        SagaInstance saga = new SagaInstance();
        saga.setSagaId(sagaId);
        saga.setState(SagaState.STARTED);
        saga.setCurrentStepIndex(0);
    }
}
```



```
when(sagaRepository.findById(sagaId)).thenReturn(Optional.of(saga));  
when(sagaRepository.save(any(SagaInstance.class))).thenReturn(saga);
```

```
when(orderExecutor.execute(any())).thenReturn(SagaStepResult.success("CREATE_ORDER", null));
```

```
when(paymentExecutor.execute(any())).thenReturn(SagaStepResult.success("PROCESS_PAYMENT", null));
```

```
when(dispatchExecutor.execute(any())).thenReturn(SagaStepResult.success("ASSIGN_COURIER", null));
```

```
when(notificationExecutor.execute(any())).thenReturn(SagaStepResult.success("SEND_NOTIFICATION", null));
```

```
// when
```

```
orchestrator.executeSaga(sagaId);
```

```
// then
```

```
assertThat(saga.getState()).isEqualTo(SagaState.COMPLETED);
```

```
verify(orderExecutor).execute(any());
```

```
verify(paymentExecutor).execute(any());
```

```
verify(dispatchExecutor).execute(any());
```

```
verify(notificationExecutor).execute(any());
```

```
}
```

```
@Test
```

```

void shouldCompensateWhenPaymentFails() {
    // given
    String sagaId = "saga-456";
    SagaInstance saga = new SagaInstance();
    saga setSagaId(sagaId);
    saga setState(SagaState.STARTED);
    saga setCurrentStepIndex(1); // на шаге оплаты

    when(sagaRepository.findById(sagaId)).thenReturn(Optional of(saga));
    when(sagaRepository.save(any(SagaInstance.class))).thenReturn(saga);

    when(paymentExecutor.execute(any())).thenReturn(
        SagaStepResult.failure("PROCESS_PAYMENT", "Insufficient
funds")
    );
    when(orderExecutor.compensate(any())).thenReturn(
        SagaStepResult.success("COMPENSATE_CREATE_ORDER",
null)
    );

    // when
    orchestrator.executeSaga(sagaId);

    // then
    verify(paymentExecutor).execute(any());
    verify(orderExecutor).compensate(any());
    verify(dispatchExecutor, never()).execute(any());

    assertThat(saga.getState()).isEqualTo(SagaState.COMPENSATED);
}

```

4.9. Сравнение хореографии и оркестрации

Критерий	Хореография (ЛР 12)	Оркестрация (ЛР 13)
Сложность реализации	Средняя (много событий)	Средняя (один оркестратор)
Отслеживание состояния	Сложно (события разбросаны)	Легко (таблица sag)
Отладка	Трудно	Проще
Точка отказа	Нет	Оркестратор
Изменение процесса	Менять несколько сервисов	Только оркестратор
Повторные попытки	Сложно (нужны dead-letter)	Просто (retry в оркестраторе)
Масштабирование	Каждый сервис независимо	Оркестратор может быть узким местом

Критерий	Хореография (ЛР 12)	Оркестрация (ЛР 13)
Когда выбирать	Простые процессы, слабая связанность	Сложные процессы, нужен контроль

4.10. Архитектурная диаграмма

@startuml

!theme plain

rectangle "Клиент" as Client

rectangle "Оркестратор" {
 [OrderSagaOrchestrator] as Orchestrator
 database "Saga State Store" as SagaDB
}

rectangle "Сервисы" {
 [Order Service] as OrderSvc
 [Payment Service] as PaymentSvc
 [Dispatch Service] as DispatchSvc
 [Notification Service] as NotifySvc
}

Client --> Orchestrator : startSaga()

Orchestrator --> SagaDB : сохраняет состояние

Orchestrator --> OrderSvc : CreateOrderCommand

OrderSvc --> Orchestrator : OrderCreatedResult

Orchestrator --> PaymentSvc : ProcessPaymentCommand

PaymentSvc --> Orchestrator : PaymentResult

Orchestrator --> DispatchSvc : AssignCourierCommand

DispatchSvc --> Orchestrator : AssignmentResult

Orchestrator --> NotifySvc : SendNotificationCommand

NotifySvc --> Orchestrator : NotificationResult

Orchestrator --> Client : SagaCompleted

@enduml

5. Индивидуальные задания (15 вариантов)

Важно: Студент работает со своим сквозным проектом (тот же, что в ЛР 12).

Вариант	Сквозной проект	Шаги саги (3–5)
1	Логистика	Создать заказ → Оплатить → Назначить курьера → Уведомить
2	Интернет-магазин	Резерв товара → Оплата →

Вариант	Сквозной проект	Шаги саги (3–5)
		Создание заказа → Доставка
3	Финтех-кошелёк	Проверка счёта → Блокировка → Перевод → Разблокировка
4	Библиотека	Проверка читателя → Резерв книги → Выдача → Уведомление
5	Отели	Поиск номера → Резерв → Оплата → Подтверждение
6	Онлайн-курсы	Проверка мест → Оплата → Зачисление → Доступ
7	Медицина	Поиск врача → Резерв времени → Запись → Напоминание
8	CRM	Проверка клиента → Создание заказа → Резерв → Счёт
9	Платёжный шлюз	Авторизация → Списание → Клиринг → Уведомление
10	Склад (WMS)	Резерв → Сборка →

Вариант	Сквозной проект	Шаги саги (3–5)
		Упаковка → Отгрузка
11	Социальная сеть	Создание поста → Модерация → Публикация → Уведомление
12	Видеохостинг	Загрузка → Транскодинг → Проверка → Публикация
13	IoT-мониторинг	Получение → Фильтрация → Оповещение → Эскалация
14	Криптовалюта	Проверка счёта → Блокировка → Сопоставление → Исполнение
15	Доставка еды	Приём заказа → Подтверждение → Готовка → Доставка

Задание для студента:

Для своего проекта (по варианту):

1. Определить бизнес-процесс из 3–5 шагов (можно взять из ЛР 12).
2. Создать модель саги (таблицы, состояния).
3. Реализовать команды для каждого шага.
4. Реализовать исполнители команд (сервисы).
5. Реализовать оркестратор с логикой выполнения и компенсации.
6. Реализовать REST API для запуска и проверки статуса саги.

7. Написать тесты (успех + откат).
8. Сравнить с хореографией в выводе.

6. Контрольные вопросы

1. В чём основное отличие оркестрации от хореографии?
2. Где хранится состояние саги в оркестрации?
3. Как оркестратор восстанавливается после сбоя?
4. Что такое компенсирующая транзакция в оркестрации?
5. Как обеспечить идемпотентность команд в оркестрации?
6. В каких случаях оркестрация предпочтительнее хореографии?
7. Как тестировать оркестратор без реальных сервисов?
8. Как обрабатывать долго выполняющиеся шаги в оркестрации?

7. Требования к отчёту

Отчёт должен содержать:

Титульный лист.

Цель и задачи работы.

Описание бизнес-процесса (шаги, компенсации).

Модель саги (таблицы, состояния).

Команды (код).

Исполнители команд (код).

Оркестратор (код).

REST API (контроллер).

Тесты (успех + откат).

Сравнение с хореографией (таблица + вывод).

Диаграмма последовательности.

Вывод.

Ответы на вопросы.

8. Критерии оценивания

Критерий	max балл
Модель саги (таблицы, состояния)	1
Команды для всех шагов (3–5)	2
Исполнители команд (все шаги + компенсации)	3
Оркестратор (execute + compensate)	4
REST API (запуск, статус)	1
Тесты (успех + откат)	2
Сравнение с хореографией (таблица + вывод)	2
Соответствие варианту	1
Итого	16

Лабораторная работа №14

Версионирование API и обратная совместимость на примере «Системы управления логистикой и доставкой»

1. Цель работы

Сформировать компетенции по проектированию и реализации версионирования API для обеспечения обратной совместимости при эволюции архитектуры: использование различных стратегий версионирования (URL, header, content negotiation), управление устаревшими

версиями, миграция клиентов и депрекация API для выбранного сквозного проекта.

2. Задачи работы

1. Изучить стратегии версионирования API и их компромиссы.
2. Реализовать 2 версии API для одного и того же ресурса.
3. Реализовать разные стратегии версионирования (URL, header).
4. Обеспечить обратную совместимость между версиями.
5. Реализовать механизм депрекации (deprecation) API.
6. Создать заглушку (adapter) для старой версии, вызывающую новую.
7. Написать тесты, проверяющие работу обеих версий.
8. Оформить результаты в виде отчёта с кодом, тестами и сравнением стратегий.

3. Теоретическое обоснование

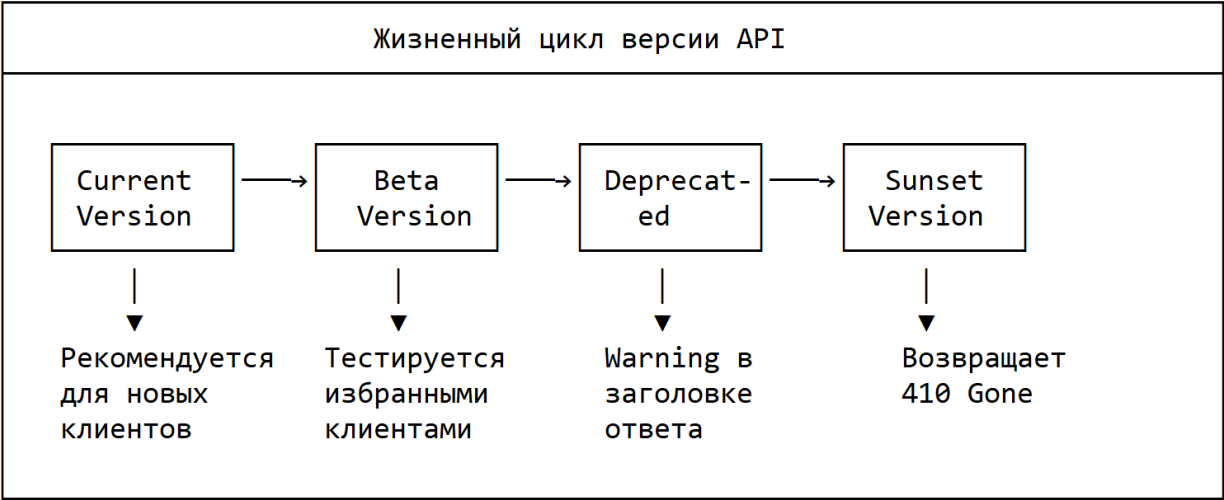
3.1. Зачем нужно версионирование API?

Проблема	Решение
Изменение формата ответа	Новая версия API
Удаление поля	Клиенты на старой версии продолжают работать
Добавление обязательного параметра	Старые клиенты не ломаются
Изменение бизнес-логики	Обе версии сосуществуют

3.2. Стратегии версионирования

Стратегия	Пример	Преимущества	Недостатки
URI Path	/v1/orders, /v2/orders	Простота, видимость в URL	"Загрязнение" URL, кэширование по разным URL
Query Parameter	/orders?version=1	Простота	Нестандартно, сложно с POST
Custom Header	Accept-version: v1	Чистый URL, стандартно	Требуется клиентская поддержка
Content Negotiation	Accept: application/vnd.myapp.v1+json	RESTful, гибкость	Сложнее в реализации
Media Type	application/vnd.myapp.order.v2+json	Полный контроль	Переусложнение

3.3. Жизненный цикл версии API



3.4. Обратная совместимость: правила

Изменение	Совместимо?	Как обработать
Добавление поля в ответ	Да	Старые клиенты игнорируют
Добавление необязательного параметра	Да	Используют default
Добавление обязательного параметра	Нет	Нужна новая версия
Удаление поля	Нет	Нужна новая версия
Изменение типа поля	Нет	Нужна новая версия
Изменение названия поля	Нет	Нужна новая версия

4. Пример выполнения

Сквозной пример: «Система управления логистикой и доставкой»
(вариант 1)

Ресурс: GET /orders/{id} — получение информации о заказе.

4.1. Версия 1 (исходная)

```
// dto/v1/OrderResponseV1.java
```

```
package com logistics api dto v1;
```

```
import lombok AllArgsConstructor;
```

```
import lombok Data;
```

```
import lombok NoArgsConstructor;
```

```
import java time Instant;
```

```
@Data
```

```
@AllArgsConstructor
```

```
@NoArgsConstructor
```

```
public class OrderResponseV1 {
```

```
    private Long id;
```

```
    private String status; // "PENDING", "ASSIGNED",
```

```
"COMPLETED"
```

```
    private String pickupAddress;
```

```
    private String dropoffAddress;
```

```
    private Double amount;
```

```
    private Instant createdAt;
```

```
}
```

```
// controller/v1/OrderControllerV1.java
```

```
package com logistics api controller.v1;
```

```
import com logistics api dto v1 OrderResponseV1;
import com logistics application service OrderQueryService;
import lombok RequiredArgsConstructor;
import org springframework web bind annotation GetMapping;
import org springframework web bind annotation PathVariable;
import org springframework web bind annotation RequestMapping;
import org springframework web bind annotation RestController;
```

```
@RestController
@RequestMapping("/api/v1/orders")
@RequiredArgsConstructor
public class OrderControllerV1 {
```

```
    private final OrderQueryService orderService;
```

```
    @GetMapping("/{id}")
    public OrderResponseV1 getOrder(@PathVariable Long id) {
        Order order = orderService.findById(id);

        return new OrderResponseV1(
            order.getId(),
            order.getStatus().name(),
            order.getPickupAddress(),
            order.getDropoffAddress(),
            order.getAmount(),
            order.getCreatedAt()
        );
    }
}
```

4.2. Версия 2 (с изменениями)

Изменения в версии 2:

Добавлено поле `customerName`

Добавлено поле `etaMinutes` (для назначенных заказов)

Поле `status` изменено с `enum` на объект с описанием

Поле `amount` переименовано в `totalAmount`

Добавлен вложенный объект `courier`

```
// dto/v2/OrderResponseV2.java
```

```
package com.logistics.api.dto.v2;
```

```
import lombok.AllArgsConstructor;
```

```
import lombok.Data;
```

```
import lombok.NoArgsConstructor;
```

```
import java.time.Instant;
```

```
@Data
```

```
@AllArgsConstructor
```

```
@NoArgsConstructor
```

```
public class OrderResponseV2 {
```

```
    private Long id;
```

```
    private OrderStatusV2 status;    // объект, а не строка
```

```
    private String customerName;
```

```
    private String pickupAddress;
```

```
    private String dropoffAddress;
```

```
    private Double totalAmount;    // переименовано
```

```
    private CourierInfo courier;    // новый вложенный объект
```

```
    private Integer etaMinutes;    // новое поле
```

```
    private Instant createdAt;
```

```
        private Instant updatedAt, // новое поле
    }

    @Data
    @AllArgsConstructor
    @NoArgsConstructor
    public class OrderStatusV2 {
        private String code, // "PENDING", "ASSIGNED", "COMPLETED"
        private String description, // "Ожидает обработки", "Назначен курьеру"
    }
```

```
    @Data
    @AllArgsConstructor
    @NoArgsConstructor
    public class CourierInfo {
        private Long id;
        private String name;
        private String phone;
        private Double currentLat;
        private Double currentLng;
    }
```

```
// controller/v2/OrderControllerV2.java
```

```
package com.logistics.api.controller.v2;
```

```
import com.logistics.api.dto.v2 OrderResponseV2;
```

```
import com.logistics.application.service OrderQueryService;
```

```
import lombok RequiredArgsConstructor;
```

```
import org.springframework.web.bind.annotation.GetMapping;
```



```
import org.springframework.web.bind.annotation PathVariable;
import org.springframework.web.bind.annotation RequestMapping;
import org.springframework.web.bind.annotation.RestController;
```

```
@RestController
@RequestMapping("/api/v2/orders")
@RequiredArgsConstructor
```

```
public class OrderControllerV2 {
```

```
    private final OrderQueryService orderService;
    private final CourierQueryService courierService;
    private final AssignmentQueryService assignmentService;
```

```
    @GetMapping("/{id}")
```

```
    public OrderResponseV2 getOrder(@PathVariable Long id) {
```

```
        Order order = orderService.findById(id);
```

```
        Assignment currentAssignment =
```

```
assignmentService.findCurrentByOrderId(id);
```

```
        Courier courier = currentAssignment != null ?
```

```
        courierService.findById(currentAssignment.getCourierId()) : null;
```

```
        return new OrderResponseV2(
```

```
            order.getId(),
```

```
            new OrderStatusV2(order.getStatus().name(),
```

```
getStatusDescription(order.getStatus())),
```

```
            order.getCustomerName(),
```

```
            order.getPickupAddress(),
```

```
            order.getDropoffAddress(),
```

```
            order.getAmount(),
```

```

courier != null ? new CourierInfo(
    courier.getId(),
    courier.getName(),
    courier.getPhone(),
    courier.getCurrentLat(),
    courier.getCurrentLng()
) : null,
currentAssignment != null ? currentAssignment.getEtaMinutes() :
null,
order.getCreatedAt(),
order.getUpdatedAt()
);
}

private String getStatusDescription(OrderStatus status) {
    return switch (status) {
        case PENDING -> "Ожидает обработки";
        case ASSIGNED -> "Назначен курьеру";
        case COMPLETED -> "Доставлен";
        case CANCELLED -> "Отменён";
    };
}
}

```

4.3. Стратегия 1: Версионирование через URL (использовано выше)

```

// Конфигурация для URL-версионирования (минимальная)
// Просто разные маппинги в контроллерах

```

4.4. Стратегия 2: Версионирование через Header

```

// controller/versioning/OrderControllerHeaderVersioning.java

```

```
package com logistics api controller versioning
```

```
import com logistics api dto v1 OrderResponseV1;
```

```
import com logistics api dto v2 OrderResponseV2;
```

```
import com logistics application service OrderQueryService;
```

```
import lombok RequiredArgsConstructor;
```

```
import org springframework web bind annotation *;
```

```
@RestController
```

```
@RequestMapping("/api/orders")
```

```
@RequiredArgsConstructor
```

```
public class OrderControllerHeaderVersioning {
```

```
    private final OrderQueryService orderService;
```

```
    private final OrderV2Mapper v2Mapper;
```

```
    @GetMapping("/{id}")
```

```
    public Object getOrder(@PathVariable Long id,
```

```
                           @RequestHeader(value = "API-Version", defaultValue =  
"1") String version) {
```

```
        Order order = orderService.findById(id);
```

```
        switch (version) {
```

```
            case "2":
```

```
                return v2Mapper.toV2(order);
```

```
            case "1":
```

```
            default:
```

```
                return new OrderResponseV1(
```

```

        order getId(),
        order getStatus().name(),
        order getPickupAddress(),
        order getDropoffAddress(),
        order getAmount(),
        order getCreatedAt()
    );
}
}
}

```

4.5. Стратегия 3: Content Negotiation (Accept header)

// controller/versioning/OrderControllerContentNegotiation.java

package com.logistics.api.controller.versioning;

```

import com.logistics.api.dto.v1.OrderResponseV1;
import com.logistics.api.dto.v2.OrderResponseV2;
import com.logistics.application.service.OrderQueryService;
import lombok.RequiredArgsConstructor;
import org.springframework.http.MediaType;
import org.springframework.web.bind.annotation.*;

```

```
@RestController
```

```
@RequestMapping("/api/orders")
```

```
@RequiredArgsConstructor
```

```
public class OrderControllerContentNegotiation {
```

```
    private final OrderQueryService orderService;
```

```
    private final OrderV2Mapper v2Mapper;
```

```

@GetMapping(value =("/{id}", produces = {
    "application/vnd.logistics.v1+json",
    "application/json"
}))

public OrderResponseV1 getOrderV1(@PathVariable Long id) {
    Order order = orderService findById(id);
    return new OrderResponseV1(
        order.getId(),
        order.getStatus() name(),
        order.getPickupAddress(),
        order.getDropoffAddress(),
        order.getAmount(),
        order.getCreatedAt()
    );
}

```

```

@GetMapping(value =("/{id}", produces =
"application/vnd.logistics.v2+json")

public OrderResponseV2 getOrderV2(@PathVariable Long id) {
    Order order = orderService findById(id);
    return v2Mapper toV2(order);
}
}

```

4.6. Адаптер (заглушка) для обратной совместимости

// adapter/OrderV1ToV2Adapter.java

```
package com logistics api adapter;
```

```
import com logistics api dto v1 OrderResponseV1;
```

```
import com logistics api dto v2 OrderResponseV2;
```

```

import com logistics application service OrderQueryService;
import lombok RequiredArgsConstructor;
import org springframework stereotype Component;

/**
 * Адаптер позволяет вызывать V2 реализацию для V1 клиентов.
 * Используется, когда старая версия API может быть реализована через
новую.
 */
@Component
@RequiredArgsConstructor
public class OrderV1ToV2Adapter {

    private final OrderQueryService orderService;
    private final OrderV2Mapper v2Mapper;

    /**
     * Получение V1 ответа через V2 логику (с трансформацией)
     */
    public OrderResponseV1 getOrderV1FromV2(Long orderId) {
        // Вызываем V2 логику
        Order order = orderService.findById(orderId);
        OrderResponseV2 v2Response = v2Mapper.toV2(order);

        // Преобразуем V2 ответ в V1 формат
        return transformToV1(v2Response);
    }

    private OrderResponseV1 transformToV1(OrderResponseV2 v2) {

```

```

return new OrderResponseV1(
    v2 getId(),
    v2 getStatus().getCode(),      // берём код из объекта
    v2 getPickupAddress(),
    v2 getDropoffAddress(),
    v2 getTotalAmount(),          // renamed field
    v2 getCreatedAt()
    // customerName, courier, etaMinutes — игнорируем (нет в V1)
);
}
}

```

// controller/v1/OrderControllerV1WithAdapter.java (альтернативный)

@RestController

@RequestMapping("/api/v1/orders")

@RequiredArgsConstructor

public class OrderControllerV1WithAdapter {

private final OrderV1ToV2Adapter adapter; // использует V2 логику

@GetMapping("/{id}")

public OrderResponseV1 getOrder(@PathVariable Long id) {

return adapter.getOrderV1FromV2(id);

}

}

4.7. Механизм депрекации (Deprecation)

// annotation/DeprecatedApi.java

package com.logistics.api.annotation;

```
import java.lang.annotation.*;

@Target(ElementType.METHOD)
@Retention(RetentionPolicy.RUNTIME)
@Documented
public @interface DeprecatedApi {
    String since() default "";
    String until() default "";
    String reason() default "";
    String alternative() default "";
}

// aspect/DeprecationAspect.java
package com.logistics.api.aspect;

import com.logistics.api.annotation.DeprecatedApi;
import jakarta.servlet.http.HttpServletResponse;
import lombok.extern.slf4j.Slf4j;
import org.aspectj.lang.annotation.AfterReturning;
import org.aspectj.lang.annotation.Aspect;
import org.springframework.stereotype.Component;
import org.springframework.web.context.request.RequestContextHolder;
import org.springframework.web.context.request.ServletRequestAttributes;

@Aspect
@Component
@Slf4j
public class DeprecationAspect {
```



```

    @AfterReturning("@annotation(deprecatedApi)")
    public void addDeprecationWarning(DeprecatedApi deprecatedApi) {
        ServletRequestAttributes attributes =
            (ServletRequestAttributes)
RequestContextHolder.getRequestAttributes();

        if (attributes != null) {
            HttpServletResponse response = attributes.getResponse();
            if (response != null) {
                // Добавляем заголовок с предупреждением
                String warning = String.format(
                    "299 - \"API is deprecated. Since: %. Alternative: %s\"",
                    deprecatedApi.since(),
                    deprecatedApi.alternative()
                );
                response.addHeader("Warning", warning);

                log.warn("Deprecated API called: {}, alternative: {}",
                    deprecatedApi.reason(), deprecatedApi.alternative());
            }
        }
    }
}

// controller/v1/OrderControllerV1WithDeprecation.java
@RestController
@RequestMapping("/api/v1/orders")
@RequiredArgsConstructor
public class OrderControllerV1WithDeprecation {

```

```
private final OrderV1ToV2Adapter adapter;
```

```
@GetMapping("/{id}")
```

```
@DeprecatedApi(
```

```
    since = "2024-01-15",
```

```
    until = "2024-06-30",
```

```
    reason = "V1 API is deprecated, please migrate to V2",
```

```
    alternative = "/api/v2/orders/{id}"
```

```
)
```

```
public OrderResponseV1 getOrder(@PathVariable Long id) {
```

```
    return adapter getOrderV1FromV2(id);
```

```
}
```

```
@GetMapping("/{id}/status")
```

```
@DeprecatedApi(
```

```
    since = "2024-01-15",
```

```
    reason = "Use GET /api/v2/orders/{id} and read status field",
```

```
    alternative = "/api/v2/orders/{id}"
```

```
)
```

```
public String getOrderStatus(@PathVariable Long id) {
```

```
    // старый способ получения статуса
```

```
    return adapter getOrderV1FromV2(id).getStatus();
```

```
}
```

```
}
```

4.8. Политика sunset (удаление старой версии)

```
// config/SunsetVersionInterceptor.java
```

```
package com.logistics.api.interceptor;
```

```
import jakarta.servlet.http.HttpServletRequest;
```

```
import jakarta.servlet.http HttpServletResponse;
import org.springframework.stereotype Component;
import org.springframework.web.servlet HandlerInterceptor;
```

```
import java.time LocalDate;
import java.util Map;
```

```
@Component
```

```
public class SunsetVersionInterceptor implements HandlerInterceptor {
```

```
    // Карта версий и дат их удаления
```

```
    private final Map<String, LocalDate> sunsetDates = Map.of(
        "/api/v1", LocalDate.of(2024, 6, 30),
        "/api/v1/orders/status", LocalDate.of(2024, 3, 31)
    );
```

```
@Override
```

```
public boolean preHandle(HttpServletRequest request,
    HttpServletResponse response,
    Object handler) {
```

```
    String path = request.getRequestURI();
```

```
    for (Map.Entry<String, LocalDate> entry : sunsetDates.entrySet()) {
        if (path.startsWith(entry.getKey())) {
            LocalDate sunsetDate = entry.getValue();

            if (LocalDate.now().isAfter(sunsetDate)) {
                response.setStatus(410); // Gone
            }
        }
    }
```

```

        response.setContentType("application/json");
        response.addHeader("Sunset", sunsetDate.toString());
        return false;
    }

    if (LocalDate.now().isAfter(sunsetDate.minusMonths(1))) {
        response.addHeader("Warning",
            "299 - \"API will be removed on \" + sunsetDate + "\"");
        response.addHeader("Sunset", sunsetDate.toString());
    }
}

return true;
}
}

```

4.9. Тестирование версионирования

```
// test/api/VersioningTest.java
```

```
package com.logistics.test.api;
```

```
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.boot.test.context.SpringBootTest;
```

```
import org.springframework.test.web.servlet.MockMvc;
```

```
import org.springframework.test.web.servlet.request MockMvcRequestBuilders get;  
import org.springframework.test.web.servlet.result MockMvcResultMatchers.*;
```

```
@SpringBootTest  
@AutoConfigureMockMvc  
class VersioningTest {
```

```
    @Autowired  
    private MockMvc mockMvc;
```

```
    @Test  
    void shouldReturnV1ResponseForV1Url() throws Exception {  
        mockMvc.perform(get("/api/v1/orders/1"))  
            .andExpect(status().isOk())  
            .andExpect(jsonPath("$.id").value(1))  
            .andExpect(jsonPath("$.status").exists())  
            .andExpect(jsonPath("$.customerName").doesNotExist())  
            .andExpect(jsonPath("$.totalAmount").doesNotExist());  
    }
```

```
    @Test  
    void shouldReturnV2ResponseForV2Url() throws Exception {  
        mockMvc.perform(get("/api/v2/orders/1"))  
            .andExpect(status().isOk())  
            .andExpect(jsonPath("$.id").value(1))  
            .andExpect(jsonPath("$.status.code").exists())  
            .andExpect(jsonPath("$.customerName").exists())  
    }
```

```
        .andExpect(jsonPath("$.totalAmount").exists());
    }
}
```

```
@Test
```

```
void shouldReturnV1ByDefaultForHeaderVersioning() throws Exception
```

```
{
    mockMvc.perform(get("/api/orders/1"))
        .andExpect(status().isOk())
        .andExpect(jsonPath("$.status").exists())
        .andExpect(jsonPath("$.status.code").doesNotExist());
}
```

```
@Test
```

```
void shouldReturnV2WhenHeaderSpecified() throws Exception {
```

```
    mockMvc.perform(get("/api/orders/1")
        .header("API-Version", "2"))
        .andExpect(status().isOk())
        .andExpect(jsonPath("$.status.code").exists());
}
```

```
@Test
```

```
void shouldReturnDeprecationWarningForV1() throws Exception {
```

```
    mockMvc.perform(get("/api/v1/orders/1"))
        .andExpect(status().isOk())
        .andExpect(header().exists("Warning"));
}
```

```
@Test
```

```
void shouldReturn410ForSunsetApi() throws Exception {
```

```

// предполагаем, что версия v1/status уже удалена
mockMvc.perform(get("/api/v1/orders/1/status"))
        .andExpect(status().isGone());
}

@Test
void shouldReturnV2ForContentNegotiation() throws Exception {
    mockMvc.perform(get("/api/orders/1")
        .accept("application/vnd.logistics.v2+json"))
        .andExpect(status().isOk())
        .andExpect(jsonPath("$.totalAmount").exists())
        .andExpect(jsonPath("$.customerName").exists());
}
}

```

4.10. Сравнение стратегий версионирования

Стратегия	Сложность	Видимость	Кэширование	RESTful	Рекомендация
URI Path	Низкая	Высокая	Раздельное	Средняя	Для простых API
Header	Средняя	Низкая	По URL	Высокая	Для внутренних API
Content Negotiation	Высокая	Низкая	По Accept	Высокая	Для публичных API

Ст ратегия	Сл ожность	Ви димость	Кэш ирование	R ESTful	Реко мендация
Qu ery Paramete r	Ни зкая	Ср едняя	Зави сит	Н изкая	Не рекомендуется

4.11. Миграция клиентов: план

```
// service/ClientMigrationService.java
```

```
package com.logistics.api.service;
```

```
import lombok.extern.slf4j.Slf4j;
```

```
import org.springframework.stereotype.Service;
```

```
import java.util.Map;
```

```
import java.util.concurrent.ConcurrentHashMap;
```

```
@Service
```

```
@Slf4j
```

```
public class ClientMigrationService {
```

```
    private final Map<String, String> clientVersions = new  
    ConcurrentHashMap<>();
```

```
    /**
```

```
     * Регистрация клиента и его версии
```

```
     */
```

```
    public void registerClient(String clientId, String apiVersion) {
```

```
        clientVersions.put(clientId, apiVersion);
```



```

        log info("Client {} registered with API version {}", clientId,
apiVersion);
    }

    /**
     * Получение статистики по версиям
     */
    public Map<String, Long> getVersionStats() {
        return clientVersions values().stream()
            .collect(Collectors.groupingBy(v -> v, Collectors.counting()));
    }

    /**
     * Проверка, что все клиенты мигрировали с версии
     */
    public boolean isVersionReadyForSunset(String version) {
        return clientVersions values().stream()
            .noneMatch(v -> v.equals(version));
    }

    /**
     * Отправка уведомления о необходимости миграции
     */
    public void notifyClientsToMigrate(String oldVersion, String
newVersion) {
        clientVersions entrySet().stream()
            .filter(e -> e.getValue().equals(oldVersion))
            .forEach(e -> {
                log warn("Client {} still using deprecated API version {}",

```

```

        e.getKey(), oldVersion);

        // В реальности — отправка email или уведомления

    });
}
}

```

5. Индивидуальные задания (15 вариантов)

Важно: Студент работает со своим сквозным проектом (из ЛР 1).

В ариант	Сквозно й проект	Ресурс для версионирования	Изменения в V2
1	Логисти ка	GET /orders/{id}	Добавить customerName, etaMinutes, courier
2	Интерне т-магазин	GET /products/{id}	Добавить rating, reviews, stock статус
3	Финтех- кошелёк	GET /wallets/{id}	Добавить frozen_amount, last_transactions
4	Библиот ека	GET /books/{id}	Добавить availability, reservation_status
5	Отели	GET /rooms/{id}	Добавить amenities, cancellation_policy
6	Онлайн-	GET	Добавить progress,

В ариант	Сквозно й проект	Ресурс для версионирования	Изменения в V2
	курсы	/courses/{id}	certificate_url
7	Медици на	GET /appointments/{id}	Добавить doctor_info, telemedicine_link
8	CRM	GET /customers/{id}	Добавить loyalty_tier, total_spent
9	Платёжн ый шлюз	GET /transactions/{id}	Добавить fee, merchant_info, status_details
10	Склад (WMS)	GET /inventory/{id}	Добавить reserved_quantity, reorder_level
11	Социаль ная сеть	GET /posts/{id}	Добавить shares_count, is_edited, media_urls
12	Видеохо стинг	GET /videos/{id}	Добавить duration, like_ratio, comments_count
13	IoT- мониторинг	GET /devices/{id}	Добавить battery_level, last_offline_at

В ариант	Сквозно й проект	Ресурс для версионирования	Изменения в V2
14	Криптоб иржа	GET /orders/{id}	Добавить fee, executed_volume, avg_price
15	Доставк а еды	GET /orders/{id}	Добавить restaurant_info, delivery_tracking_link

Задание для студента:

Для своего проекта (по варианту):

1. Реализовать V1 API (исходная версия).
2. Реализовать V2 API с указанными изменениями.
3. Реализовать минимум 2 стратегии версионирования.
4. Создать адаптер для обратной совместимости.
5. Реализовать механизм депрекации (заголовок Warning).
6. Реализовать проверку sunset (410 Gone).
7. Написать тесты для всех версий.
8. Сравнить стратегии в выводе.

6. Контрольные вопросы

1. Какие стратегии версионирования API существуют? Их优缺点?
2. Что такое обратная совместимость и как её обеспечить?
3. Какой заголовок используется для предупреждения о депрекации?
4. Какой HTTP статус возвращается при удалении старой версии?
5. Что такое Sunset header и для чего он нужен?

- 6. В чём разница между депрекацией и sunset?
- 7. Как мигрировать клиентов на новую версию API?
- 8. Как тестировать разные версии API?

7. Требования к отчёту

Отчёт должен содержать:

Титульный лист.

Цель и задачи работы.

Описание изменений между V1 и V2.

V1 API (код контроллера и DTO).

V2 API (код контроллера и DTO).

Стратегии версионирования (минимум 2).

Адаптер для обратной совместимости.

Механизм депрекации.

Политика sunset (410 Gone).

Тесты.

Сравнение стратегий.

Вывод.

Ответы на вопросы.

8. Критерии оценивания

Критерий	max балл
V1 API (контроллер + DTO)	2
V2 API (контроллер + DTO с изменениями)	2
Стратегии версионирования (минимум 2)	3

Критерий	max балл
Адаптер для обратной совместимости	2
Депрекация (заголовок Warning)	2
Sunset (410 Gone)	2
Тесты (проверка всех версий)	2
Соответствие варианту	1
Итого	16

Лабораторная работа №15

Strangler pattern – постепенная замена модуля на примере «Системы управления логистикой и доставкой»

1. Цель работы

Сформировать компетенции по миграции монолитного приложения к микросервисной архитектуре с использованием паттерна Strangler Fig: поэтапная замена функциональности, настройка reverse проху для маршрутизации, реализация антикоррупционного слоя (Anti-Corruption Layer) и обеспечение бесшовной работы системы в процессе миграции для выбранного сквозного проекта.

2. Задачи работы

1. Изучить паттерн Strangler Fig и его применение для миграции.
2. Выделить один модуль монолита для вынесения в отдельный микросервис.

3. Настроить reverse proxy (Nginx/Spring Cloud Gateway) для маршрутизации запросов.
4. Реализовать антикоррупционный слой (ACL) для доступа к данным монолита.
5. Обеспечить поэтапный перенос функциональности.
6. Написать тесты, проверяющие корректную маршрутизацию.
7. Оформить результаты в виде отчёта с кодом, конфигурациями и диаграммами.

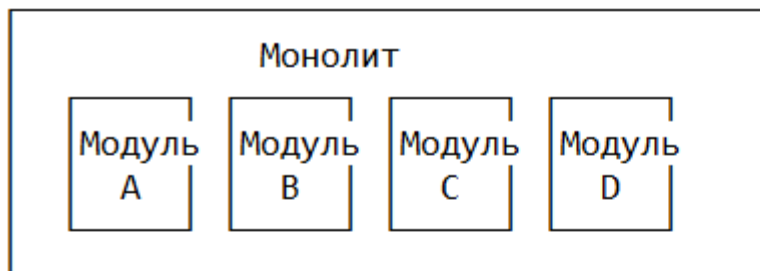
3. Теоретическое обоснование

3.1. Паттерн Strangler Fig (Strangler Application)

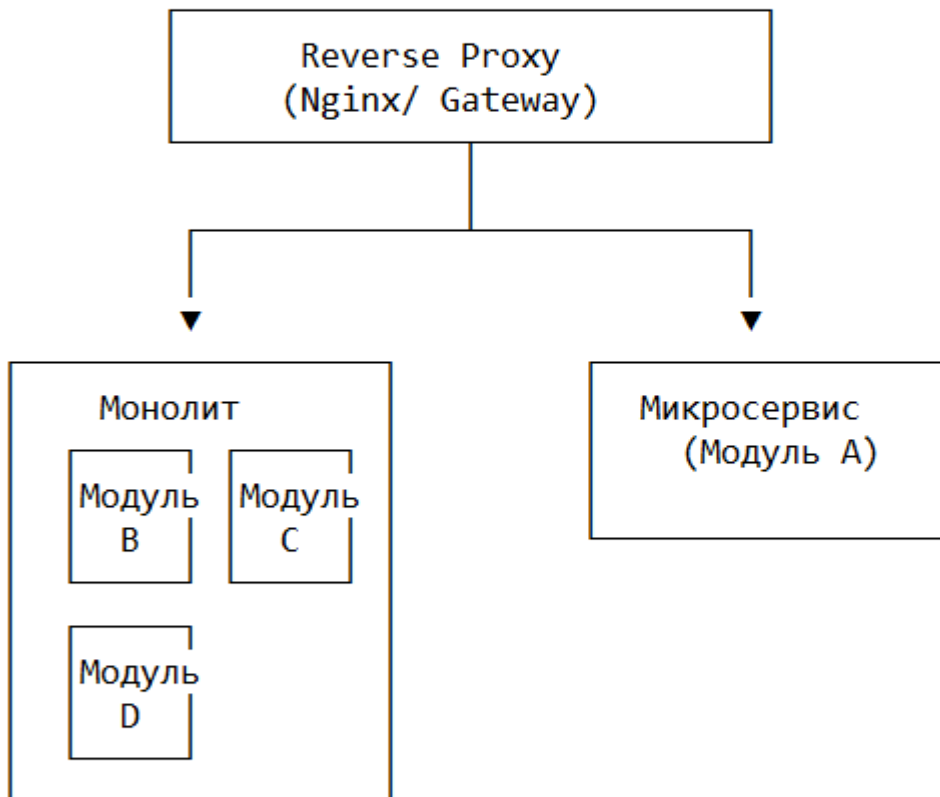
Название: по аналогии с фиговым деревом-удушителем (strangler fig), которое обвивает дерево-хозяина и постепенно заменяет его.

Этап 1: Монолит

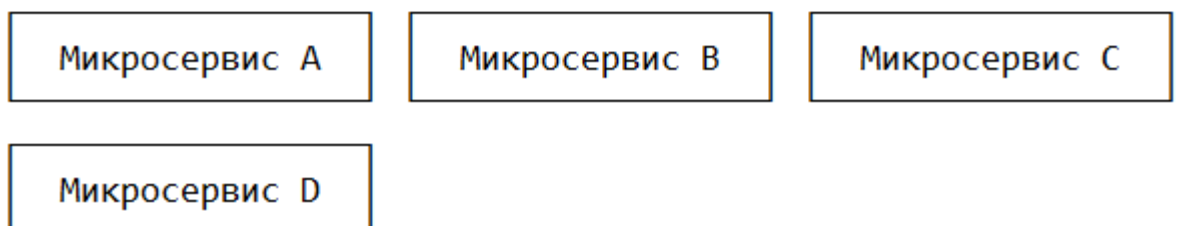
Этап 1: Монолит



Этап 2: Reverse proxy + первый микросервис



Этап 3: Полная замена монолита



3.2. Стратегии миграции

Стратегия	Описание	Когда применять
По API	Замена по одному endpoint'у	API-ориентированные системы
По модулю	Вынесение целого модуля	Слабо связанные модули
По таблицам	Замена доступа к таблицам	При смене БД
По функциям	Замена функции за функцией	Для критичных систем

3.3. Компоненты Strangler pattern

Компонент	Назначение
Reverse Proxy	Маршрутизирует запросы на монолит или микросервис
Feature Toggle	Управляет переключением (можно использовать ЛР 16)
Anti-Corruption Layer (ACL)	Адаптер между старым и новым кодом
Proxy Facade	Промежуточный слой для миграции данных

4. Пример выполнения

Сквозной пример: «Система управления логистикой и доставкой»
(вариант 1)

Выносимый модуль: Courier Management (управление курьерами)

4.1. Исходный монолит (структура)

monolith/

```
|— src/main/java/com/logistics/
|   |— controller/
|   |   |— CourierController.java
|   |   |— OrderController.java
|   |   |— DispatchController.java
|   |— service/
|   |   |— CourierService.java
|   |   |— OrderService.java
|   |   |— DispatchService.java
|   |— repository/
|   |   |— CourierRepository.java
|   |   |— OrderRepository.java
|   |   |— AssignmentRepository.java
|   |— entity/
|   |   |— Courier.java
|   |   |— Order.java
|   |   |— Assignment.java
```

4.2. Создаём новый микросервис (Courier Service)

```
// courier-
service/src/main/java/com/logistics/courier/CourierServiceApplication.java
package com logistics courier;

import org.springframework.boot.SpringApplication;
```

```

import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class CourierServiceApplication {
    public static void main(String[] args) {
        SpringApplication.run(CourierServiceApplication.class, args);
    }
}

// courier-
service/src/main/java/com/logistics/courier/controller/CourierController.java
package com.logistics.courier.controller;

import com.logistics.courier.dto.CourierRequest;
import com.logistics.courier.dto.CourierResponse;
import com.logistics.courier.dto.CourierLocationRequest;
import com.logistics.courier.service.CourierService;
import lombok.RequiredArgsConstructor;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;

import java.util.List;

@RestController
@RequestMapping("/api/couriers")
@RequiredArgsConstructor
public class CourierController {

    private final CourierService courierService;

```

@GetMapping

```
public ResponseEntity<List<CourierResponse>> getAllCouriers() {  
    return ResponseEntity.ok(courierService.getAllCouriers());  
}
```

@GetMapping("/{id}")

```
public ResponseEntity<CourierResponse> getCourier(@PathVariable  
Long id) {  
    return ResponseEntity.ok(courierService.getCourier(id));  
}
```

@PostMapping

```
public ResponseEntity<CourierResponse> createCourier(@RequestBody  
CourierRequest request) {  
    return ResponseEntity.ok(courierService.createCourier(request));  
}
```

@PutMapping("/{id}/location")

```
public ResponseEntity<Void> updateLocation(@PathVariable Long id  
@RequestBody CourierLocationRequest  
request) {  
    courierService.updateLocation(id, request.getLat(), request.getLng());  
    return ResponseEntity.ok().build();  
}
```

@GetMapping("/{id}/location")

```
public ResponseEntity<CourierLocationRequest>  
getLocation(@PathVariable Long id) {
```

```

        return ResponseEntity.ok(courierService.getLocation(id));
    }

    @GetMapping("/nearest")
    public ResponseEntity<CourierResponse> findNearest(@RequestParam
double lat,
                                                    @RequestParam double lng) {
        return ResponseEntity.ok(courierService.findNearest(lat, lng));
    }
}

// courier-
service/src/main/java/com/logistics/courier/dto/CourierResponse.java
package com logistics courier dto;

import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

@Data
@AllArgsConstructor
@NoArgsConstructor
public class CourierResponse {
    private Long id;
    private String name;
    private String phone;
    private Double lat;
    private Double lng;
    private String status // ACTIVE, BUSY, OFFLINE

```

```
private Double rating;  
private Integer totalDeliveries;  
}
```

4.3. Антикоррупционный слой (ACL) в монолите

```
// monolith/src/main/java/com/logistics/acl/CourierServiceACL.java  
package com.logistics.acl;
```

```
import com.fasterxml.jackson.databind.ObjectMapper;  
import com.logistics.entity Courier;  
import com.logistics.repository CourierRepository;  
import lombok.RequiredArgsConstructor;  
import lombok.extern.slf4j.Slf4j;  
import org.springframework.beans.factory.annotation.Value;  
import org.springframework.http.*;  
import org.springframework.stereotype.Component;  
import org.springframework.web.client.RestTemplate;  
  
import java.util.HashMap;  
import java.util.Map;  
import java.util.Optional;
```

```
/**
```

* Anti-Corruption Layer для доступа к новому Courier микросервису.

* Позволяет монолиту обращаться к новому сервису, абстрагируя различия в API.

```
*/
```

```
@Component
```

```
@RequiredArgsConstructor
```

@Slf4j

public class CourierServiceACL {

private final RestTemplate restTemplate;

private final ObjectMapper objectMapper;

private final CourierRepository legacyCourierRepository;

@Value("\${courier.service.url:http://localhost:8081}")

private String courierServiceUrl;

@Value("\${courier.service.enabled:false}")

private boolean courierServiceEnabled;

/**

* Получение курьера через ACL (с проверкой: новый сервис или legacy)

*/

public Optional<CourierDTO> getCourier(Long id) {

if (courierServiceEnabled) {

return getCourierFromNewService(id);

} else {

return getCourierFromLegacy(id);

}

}

/**

* Запрос к новому микросервису

*/

private Optional<CourierDTO> getCourierFromNewService(Long id) {

```

try {
    String url = courierServiceUrl + "/api/couriers/" + id;
    ResponseEntity<String> response = restTemplate.getForEntity(url,
String class);

    if (response.getStatusCode() == HttpStatus.OK) {
        // Трансформация ответа нового сервиса в формат, понятный
МОНОЛИТУ

        Map<String, Object> responseMap =
objectMapper.readValue(response.getBody(), Map.class);
        return Optional.of(transformToCourierDTO(responseMap));
    }
} catch (Exception e) {
    log.error("Failed to get courier from new service: {}",
e.getMessage());
    // Fallback на legacy
    return getCourierFromLegacy(id);
}
return Optional.empty();
}

/**
 * Запрос к legacy (монолит)
 */
private Optional<CourierDTO> getCourierFromLegacy(Long id) {
    return legacyCourierRepository.findById(id)
.map(this::transformToCourierDTO);
}

```



```

/**
 * Обновление локации курьера
 */
public void updateLocation(Long id, double lat, double lng) {
    if (courierServiceEnabled) {
        updateLocationInNewService(id, lat, lng);
    } else {
        updateLocationInLegacy(id, lat, lng);
    }
}

private void updateLocationInNewService(Long id, double lat, double
lng) {
    try {
        String url = courierServiceUrl + "/api/couriers/" + id + "/location";
        Map<String, Double> request = new HashMap<>();
        request.put("lat", lat);
        request.put("lng", lng);

        HttpHeaders headers = new HttpHeaders();
        headers.setContentType(MediaType.APPLICATION_JSON);
        HttpEntity<Map<String, Double>> entity = new
HttpEntity<>(request, headers);

        restTemplate.put(url, entity);
        log.info("Updated courier {} location in new service", id);
    } catch (Exception e) {

```

```

        log error("Failed to update location in new service, falling back to
legacy: {}"),

        e.getMessage());
    updateLocationInLegacy(id, lat, lng);
    }
}

```

```

private void updateLocationInLegacy(Long id, double lat, double lng) {
    legacyCourierRepository findById(id).ifPresent(courier -> {
        courier.setLat(lat);
        courier.setLng(lng);
        legacyCourierRepository save(courier);
        log info("Updated courier {} location in legacy", id);
    });
}

```

```

/**
 * Поиск ближайшего курьера
 */
public Optional<CourierDTO> findNearest(double lat, double lng) {
    if (courierServiceEnabled) {
        return findNearestInNewService(lat, lng);
    } else {
        return findNearestInLegacy(lat, lng);
    }
}

```

```

private Optional<CourierDTO> findNearestInNewService(double lat,
double lng) {

```

```

    try {
        String url = courierServiceUrl + "/api/couriers/nearest?lat=" + lat +
"&lng=" + lng;
        ResponseEntity<String> response = restTemplate getForEntity(url,
String class);

        if (response.getStatusCode() == HttpStatus OK) {
            Map<String, Object> responseMap =
objectMapper.readValue(response.getBody(), Map class);
            return Optional.of(transformToCourierDTO(responseMap));
        }
    } catch (Exception e) {
        log error("Failed to find nearest in new service: {}", e.getMessage());
        return findNearestInLegacy(lat, lng);
    }
    return Optional.empty();
}

```

```

private Optional<CourierDTO> findNearestInLegacy(double lat, double
lng) {
    return legacyCourierRepository findNearest(lat, lng)
        .map(this::transformToCourierDTO);
}

```

```

// Трансформация между форматами
private CourierDTO transformToCourierDTO(Map<String, Object>
newServiceResponse) {
    CourierDTO dto = new CourierDTO();
    dto.setId(((Number) newServiceResponse get("id")).longValue());
}

```

```
    dto.setName(((String) newServiceResponse get("name")));
    dto.setPhone(((String) newServiceResponse get("phone")));
    dto.setLat(((Number) newServiceResponse get("lat")).doubleValue());
    dto.setLng(((Number) newServiceResponse get("lng")).doubleValue());
    dto.setStatus(((String) newServiceResponse get("status")));
    return dto;
}
```

```
private CourierDTO transformToCourierDTO(Courier legacyEntity) {
    CourierDTO dto = new CourierDTO();
    dto.setId(legacyEntity getId());
    dto.setName(legacyEntity getName());
    dto.setPhone(legacyEntity getPhone());
    dto.setLat(legacyEntity getLat());
    dto.setLng(legacyEntity getLng());
    dto.setStatus(legacyEntity getStatus());
    return dto;
}
```

// DTO для обмена между слоями

```
@Data
public static class CourierDTO {
    private Long id;
    private String name;
    private String phone;
    private Double lat;
    private Double lng;
    private String status;
}
```

```
}
```

4.4. Reverse проху конфигурация (Spring Cloud Gateway)

```
# gateway/src/main/resources/application.yml
```

```
spring:
```

```
  cloud:
```

```
    gateway:
```

```
      routes:
```

```
        # Маршруты для нового микросервиса курьеров
```

```
        - id: courier service
```

```
          uri: http //localhost:8081
```

```
          predicates:
```

```
            - Path=/api/couriers/**
```

```
          filters:
```

```
            - name: CircuitBreaker
```

```
              args:
```

```
                name: courierService
```

```
                fallbackUri: forward /fallback/couriers
```

```
        # Маршруты для монолита (всё остальное)
```

```
        - id: monolith
```

```
          uri: http //localhost:8080
```

```
          predicates:
```

```
            - Path=/api/**
```

```
java
```

```
// gateway/src/main/java/com/logistics/gateway/GatewayApplication.java
```

```
package com logistics gateway;
```

```
import org springframework boot SpringApplication;
```

```
import org springframework boot autoconfigure SpringBootApplication;
```

```

import org.springframework.cloud.gateway.route RouteLocator;
import
org.springframework.cloud.gateway.route.builder RouteLocatorBuilder;
import org.springframework.context.annotation Bean;

@SpringBootApplication

public class GatewayApplication {

    public static void main(String[] args) {
        SpringApplication.run(GatewayApplication.class, args);
    }

    @Bean
    public RouteLocator customRouteLocator(RouteLocatorBuilder builder) {
        return builder.routes()
            // Маршрут для нового сервиса курьеров
            .route("courier-service", r -> r
                .path("/api/couriers/**")
                .uri("http://localhost:8081"))
            // Маршрут для монолита
            .route("monolith", r -> r
                .path("/api/**")
                .uri("http://localhost:8080"))
            .build();
    }
}

```

4.5. Nginx конфигурация (альтернатива)

```
# /etc/nginx/nginx.conf
```

```
upstream monolith {
    server localhost:8080;
}

upstream courier_service {
    server localhost:8081;
}

server {
    listen 80;
    server_name logistics.example.com;

    location /api/couriers/ {
        # Запросы к курьерам идут в новый микросервис
        proxy_pass http://courier_service/api/couriers/;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;

        # Fallback на монолит при ошибке
        proxy_next_upstream error timeout http_500;
        proxy_next_upstream_tries 1;
    }

    location /api/ {
        # Все остальные запросы в монолит
        proxy_pass http://monolith/api/;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
    }
}
```

```
}
```

4.6. Feature Toggle для управления миграцией

```
// monolith/src/main/java/com/logistics/config/FeatureToggleConfig.java
```

```
package com logistics config
```

```
import org springframework beans factory annotation Value;
```

```
import org springframework context annotation Configuration;
```

```
@Configuration
```

```
public class FeatureToggleConfig {
```

```
    @Value("${feature.courier-service.enabled:false}")
```

```
    private boolean courierServiceEnabled;
```

```
    @Value("${feature.courier-service.migrate.percentage:0}")
```

```
    private int migratePercentage;
```

```
    /**
```

```
     * Определяет, нужно ли направлять запрос на новый сервис
```

```
     * (на основе процента миграции)
```

```
     */
```

```
    public boolean useNewService(Long courierId) {
```

```
        if (!courierServiceEnabled) {
```

```
            return false;
```

```
        }
```

```
        if (migratePercentage >= 100) {
```

```
            return true;
```



```

    }

    if (migratePercentage <= 0) {
        return false;
    }

    // Постепенная миграция: по ID курьера
    return (courierId % 100) < migratePercentage;
}
}

// использование в ACL
@Service
@RequiredArgsConstructor
public class CourierServiceFacade {

    private final FeatureToggleConfig featureToggle;
    private final CourierServiceACL acl;

    public Optional<CourierDTO> getCourier(Long id) {
        if (featureToggle.useNewService(id)) {
            log debug("Using new courier service for id: {}", id);
            return acl.getCourierFromNewService(id);
        } else {
            log debug("Using legacy courier service for id: {}", id);
            return acl.getCourierFromLegacy(id);
        }
    }
}

```

4.7. Миграция данных (синхронизация)

```
//  
monolith/src/main/java/com/logistics/migration/CourierDataMigrationService.java  
  
package com logistics migration;  
  
import com logistics acl CourierServiceACL;  
import com logistics entity Courier;  
import com logistics repository CourierRepository;  
import lombok RequiredArgsConstructor;  
import lombok extern slf4j Slf4j;  
import org springframework scheduling annotation Scheduled;  
import org springframework stereotype Service;  
  
import java util List;  
  
@Service  
@RequiredArgsConstructor  
@Slf4j  
public class CourierDataMigrationService {  
  
    private final CourierRepository legacyRepository;  
    private final CourierServiceACL acl;  
  
    private boolean migrationComplete = false;  
  
    /**  
     * Миграция данных из монолита в новый сервис  
     */
```

```

@Scheduled(fixedDelay = 60000) // каждую минуту
public void migrateData() {
    if (migrationComplete) {
        return;
    }

    log info("Starting courier data migration...");

    List<Courier> couriers = legacyRepository findAll();
    int migrated = 0;

    for (Courier courier : couriers) {
        try {
            // Отправляем курьера в новый сервис
            acl createCourierInNewService(courier);
            migrated++;
        } catch (Exception e) {
            log error("Failed to migrate courier {}: {}", courier.getId(),
e.getMessage());
        }
    }

    log info("Migration complete. Migrated {} couriers", migrated);
    migrationComplete = true;
}

/**
 * Синхронизация изменений (двунаправленная)
 */

```

```

public void syncCourierUpdate(Courier courier) {
    if (migrationComplete) {
        acl updateCourierInNewService(courier);
    }
}
}
}

```

4.8. Docker Compose для локального тестирования

```

# docker-compose.yml
version: '3.8'

services:
  gateway:
    build: ./gateway
    ports:
      - "80:8080"
    depends_on:
      - monolith
      - courier service

  monolith:
    build: ./monolith
    ports:
      - "8080:8080"
    environment:
      - SPRING_PROFILES_ACTIVE=legacy
      - COURIER_SERVICE_URL=http://courier service:8081
      - FEATURE_COURIER_SERVICE_ENABLED=false
    depends_on:

```

- postgres monolith

courier-service:

build: ./courier service

ports:

- "8081:8081"

environment:

- SPRING_PROFILES_ACTIVE=microservice
- DB_HOST=postgres courier

depends_on:

- postgres courier

postgres-monolith:

image: postgres 15

environment:

POSTGRES_DB: logistics_monolith

POSTGRES_USER: logistics

POSTGRES_PASSWORD: password

postgres-courier:

image: postgres 15

environment:

POSTGRES_DB: logistics_courier

POSTGRES_USER: logistics

POSTGRES_PASSWORD: password

4.9. Тестирование миграции

```
// test/strangler/StranglerPatternTest.java
```

```
package com.logistics.test.strangler
```

```
import com logistics acl CourierServiceACL;
import org junit jupiter api Test;
import org springframework beans factory annotation Autowired;
import org springframework boot test context SpringBootTest;
import org springframework boot test web client TestRestTemplate;
import org springframework http ResponseEntity;
```

```
import static org assertj core api Assertions assertThat;
```

```
@SpringBootTest(webEnvironment =
SpringBootTest.WebEnvironment.RANDOM_PORT)
class StranglerPatternTest {
```

```
@Autowired
```

```
private TestRestTemplate restTemplate;
```

```
@Autowired
```

```
private CourierServiceACL acl;
```

```
@Test
```

```
void shouldRouteCourierRequestsToNewService() {
```

```
// При включённом feature toggle
```

```
System.setProperty("courier.service.enabled", "true");
```

```
ResponseEntity<String> response = restTemplate
```

```
.getForEntity("/api/couriers/1", String.class);
```

```
assertThat(response.getStatusCode().is2xxSuccessful()).isTrue();
```

```

// Проверяем, что ответ содержит поля из нового сервиса
assertThat(response.getBody()).contains("phone", "rating");
}

@Test
void shouldFallbackToLegacyWhenNewServiceFails() {
    // Отключаем новый сервис
    System.setProperty("courier.service.enabled", "false");

    ResponseEntity<String> response = restTemplate
        .getForEntity("/api/couriers/1", String.class);

    assertThat(response.getStatusCode().is2xxSuccessful()).isTrue();
}

```

```

@Test
void shouldMigrateDataCorrectly() {
    // Проверяем, что данные из монолита доступны в новом сервисе
    Long testCourierId = 100L;

    // Через ACL (должен сходить в новый сервис)
    var courier = acl.getCourier(testCourierId);

    assertThat(courier).isPresent();
    assertThat(courier.get().getId()).isEqualTo(testCourierId);
}
}

```

4.10. Диаграмма последовательности Strangler pattern

@startuml

!theme plain

actor "Клиент" as Client

participant "Reverse Proxy\n(Nginx/Gateway)" as Proxy

participant "Микросервис\nCourier Service" as NewService

participant "Монолит" as Monolith

participant "Anti-Corruption\nLayer" as ACL

database "Legacy DB" as LegacyDB

database "New DB" as NewDB

Client -> Proxy: GET /api/couriers/1

alt Запрос к курьеру

Proxy -> NewService: /api/couriers/1

NewService -> NewDB: SELECT * FROM couriers

NewDB --> NewService: Courier data

NewService --> Proxy: CourierResponse

Proxy --> Client: 200 OK

else Остальные запросы

Proxy -> Monolith: /api/orders/1

Monolith -> LegacyDB: SELECT * FROM orders

LegacyDB --> Monolith: Order data

Monolith --> Proxy: OrderResponse

Proxy --> Client: 200 OK

end

note over Monolith, ACL

В монолите ACL определяет,
обращаться к legacy или к новому сервису
end note

Client -> Proxy: PUT /api/couriers/1/location
Proxy -> Monolith: PUT /api/couriers/1/location
Monolith -> ACL: updateLocation(1, lat, lng)

alt Новый сервис включён

ACL -> NewService: PUT /api/couriers/1/location

NewService -> NewDB: UPDATE couriers

ACL --> Monolith: Success

else Новый сервис выключен

ACL -> LegacyDB: UPDATE couriers

ACL --> Monolith: Success

end

Monolith --> Proxy: 200 OK

Proxy --> Client: 200 OK

@enduml

4.11. План поэтапной миграции

migration-plan.yaml

phases:

phase1:

name: "Подготовка"

tasks:

- Создать новый микросервис Courier Service

- Настроить reverse proxy
- Реализовать ACL в монолите

duration: "2 недели"

risk: "Низкий"

phase2:

name: "Дуальное письмо (dual-write)"

tasks:

- Записывать данные и в монолит, и в новый сервис
- Синхронизировать существующие данные
- Включить feature toggle на 0%

duration: "1 неделя"

risk: "Средний"

phase3:

name: "Постепенное переключение"

tasks:

- Увеличить процент миграции до 10%
- Мониторинг ошибок
- Постепенное увеличение до 100%

duration: "2 недели"

risk: "Средний"

phase4:

name: "Отключение монолитного модуля"

tasks:

- Отключить код модуля в монолите
- Удалить таблицы курьеров из монолита
- Удалить ACL

duration: "1 неделя"

risk: "Низкий"

phase5:

name: "Завершение"

tasks:

- Удалить feature toggle
- Обновить документацию

duration: "3 дня"

risk: "Низкий"

4.12. Метрики мониторинга миграции

```
// monitoring/MigrationMetricsService.java
```

```
package com logistics monitoring;
```

```
import io micrometer core instrument MeterRegistry;
```

```
import lombok RequiredArgsConstructor;
```

```
import org springframework stereotype Service;
```

```
@Service
```

```
@RequiredArgsConstructor
```

```
public class MigrationMetricsService {
```

```
    private final MeterRegistry meterRegistry;
```

```
    public void recordRequestRouting(String serviceType, boolean success) {
```

```
        meterRegistry.counter("migration.requests",
```

```
            "service", serviceType
```

```
            "success", String.valueOf(success))
```

```

        ).increment();
    }

    public void recordMigrationPercentage(int percentage) {
        meterRegistry.gauge("migration.percentage", percentage);
    }

    public void recordFallbackCount(String operation) {
        meterRegistry.counter("migration.fallback",
            "operation", operation
        ).increment();
    }

    public void recordSyncLatency(String direction, long latencyMs) {
        meterRegistry.timer("migration.sync.latency",
            "direction", direction
        ).record(java.time.Duration.ofMillis(latencyMs));
    }
}

```

5. Индивидуальные задания (15 вариантов)

Важно: Студент работает со своим сквозным проектом (из ЛР 1).

В ариант	Сквозной проект	Выносимый модуль	Новый микросервис
1	Логистика	Courier Management	courier-service
2	Интернет-	Product Catalog	product-service

В ариант	Сквозной проект	Выносимый модуль	Новый микросервис
	магазин		
3	Финтех- кошелёк	Wallet Management	wallet-service
4	Библиотека	Book Catalog	book-service
5	Отели	Room Management	room-service
6	Онлайн- курсы	Course Management	course-service
7	Медицина	Doctor Schedule	schedule- service
8	CRM	Customer Management	customer- service
9	Платёжный шлюз	Transaction Processing	transaction- service
10	Склад (WMS)	Inventory Management	inventory- service
11	Социальна я сеть	User Profile	user-service
12	Видеохост	Video Storage	video-service

В ариант	Сквозной проект	Выносимый модуль	Новый микросервис
	инг		
13	IoT- мониторинг	Device Registry	device-service
14	Криптобир жа	Order Book	orderbook- service
15	Доставка еды	Restaurant Management	restaurant- service

Задание для студента:

Для своего проекта (по варианту):

1. Выделить модуль для вынесения в микросервис.
2. Создать новый микросервис (базовый контроллер, DTO, сервис).
3. Реализовать ACL в монолите (адаптер к новому сервису).
4. Настроить reverse proxy (Nginx или Spring Cloud Gateway).
5. Реализовать feature toggle для управления миграцией.
6. Написать тесты.
7. Составить план поэтапной миграции.

6. Контрольные вопросы

1. В чём суть паттерна Strangler Fig?
2. Какие компоненты необходимы для реализации Strangler pattern?
3. Что такое Anti-Corruption Layer и зачем он нужен?
4. Как reverse proxy помогает в миграции?
5. Какие стратегии миграции существуют?

6. Как обеспечить бесшовную работу системы во время миграции?
7. Как тестировать систему в процессе миграции?
8. Что делать после полного вытеснения модуля?

7. Требования к отчёту

Отчёт должен содержать:

Титульный лист.

Цель и задачи работы.

Описание выносимого модуля.

Новый микросервис (код контроллера, DTO).

ACL в монолите (код).

Конфигурация reverse проху.

Feature toggle.

Тесты.

План миграции (таблица этапов).

Диаграмма архитектуры.

Вывод.

Ответы на вопросы.

8. Критерии оценивания

Критерий	max балл
Выделение модуля для миграции	1
Новый микросервис (контроллер + DTO)	2
ACL в монолите (полнота, fallback)	3
Reverse проху конфигурация	2

Критерий	max балл
Feature toggle для управления	2
Тесты (маршрутизация, fallback)	2
План миграции (этапы, риски)	2
Соответствие варианту	1
Качество оформления	1
Итого	16

Лабораторная работа №16

Feature Toggles для архитектурных изменений

на примере «Системы управления логистикой и доставкой»

1. Цель работы

Сформировать компетенции по управлению архитектурными изменениями с использованием механизма Feature Toggles (флагов функций): реализация централизованного управления тоглами, переключение между старой и новой реализацией сервисов без изменения кода и передеплоя, обеспечение безопасного развёртывания и возможности отката для выбранного сквозного проекта.

2. Задачи работы

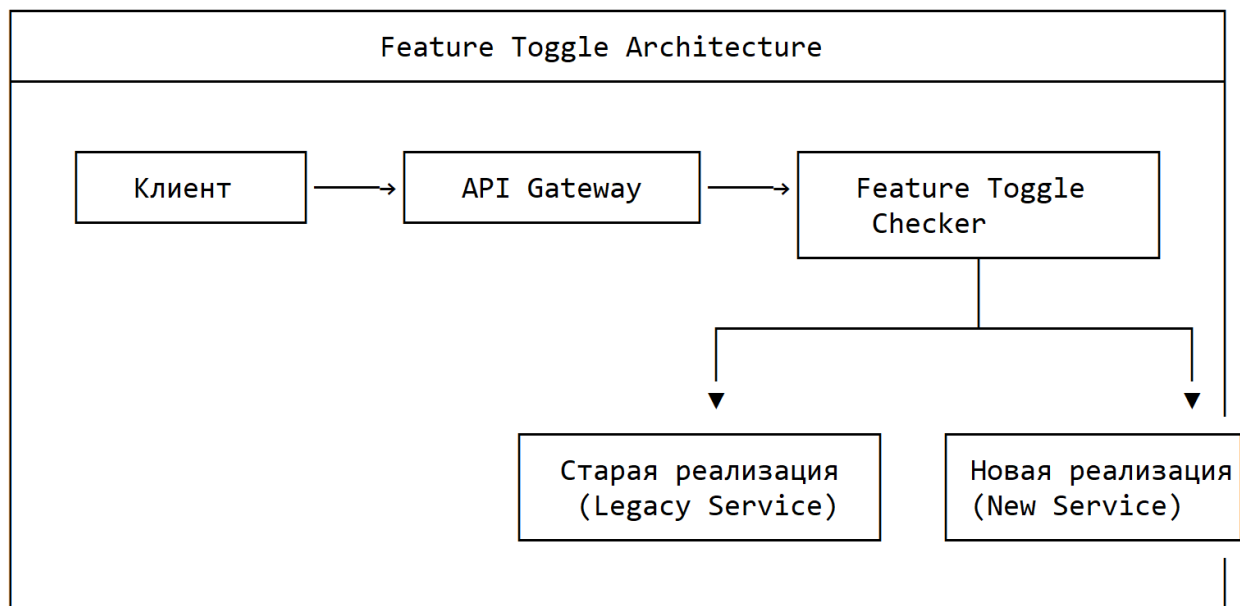
1. Изучить паттерн Feature Toggles и его применение для архитектурных изменений.
2. Реализовать механизм Feature Toggles (через конфигурационный сервер или переменные окружения).

3. Реализовать две альтернативные реализации одного сервиса (новая и старая).
4. Добавить toggle для переключения между реализациями.
5. Продемонстрировать возможность переключения без передеплоя.
6. Реализовать постепенное включение (percentage-based rollout).
7. Написать тесты для проверки работы кода с включённым и выключенным toggle.
8. Оформить результаты в виде отчёта с кодом, конфигурациями и тестами.

3. Теоретическое обоснование

3.1. Что такое Feature Toggles?

Feature Toggle - это механизм, позволяющий включать или выключать функциональность во время выполнения без изменения кода и перезапуска приложения.



3.2. Типы Feature Toggles

Тип	Описание	Жизненный цикл	Пример
Release Toggle	Включение новой фичи к релизу	Короткий (дни/недели)	Новый алгоритм расчёта ETA
Experiment Toggle	A/B тестирование	Средний (недели)	Разные стратегии назначения курьера
Ops Toggle	Управление операционными параметрами	Долгий (месяцы/годы)	Переключение БД, graceful degradation
Permission Toggle	Доступ для разных групп пользователей	Долгий	Beta-доступ для избранных клиентов

3.3. Стратегии развёртывания

Стратегия	Описание	Преимущества	Недостатки
Переменные окружения	Toggle в env	Простота	Требуется перезапуск
Файл	Toggle в	Простота	Требуется

Стратегия	Описание	Преимущества	Недостатки
конфигурации	application.yml		перезапуска
База данных	Toggle в DB	Без перезапуска	Задержка, нагрузка
Config Server	Централизованный (Spring Cloud Config)	Без перезапуска, версионирование	Сложность инфраструктуры
Feature Management Service	Специализированный сервис (LaunchDarkly, FF4J)	Мощный UI, аналитика	Внешняя зависимость

4. Пример выполнения

Сквозной пример: «Система управления логистикой и доставкой» (вариант 1)

Архитектурное изменение: новый алгоритм назначения курьера (на основе ML вместо простой гео-близости).

4.1. Feature Toggle через конфигурационный сервер (Spring Cloud Config)

Конфигурационный сервер

```
// config-server/src/main/java/com/logistics/config/ConfigServerApplication.java
package com logistics config
```

```
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cloud.config.server.EnableConfigServer;
```

```
@SpringBootApplication
```

```
@EnableConfigServer
```

```
public class ConfigServerApplication {
    public static void main(String[] args) {
        SpringApplication.run(ConfigServerApplication.class, args);
    }
}
```

```
# config-server/src/main/resources/application.yml
```

```
server:
```

```
  port: 8888
```

```
spring:
```

```
  cloud:
```

```
    config:
```

```
      server:
```

```
        git:
```

```
          uri: https://github.com/logistics/config-repo
```

```
          search-paths: '{application}'
```

```
yaml
```

```
# config-repo/logistics-service.yml (репозиторий конфигураций)
```

```
features:
```

```
  dispatch:
```

```
    new-algorithm:
```

```
enabled: false
percentage: 0
fallback-enabled: true
metrics-enabled: true
```

```
priority-queue:
  enabled: true
```

```
batch-dispatch:
  enabled: false
  batch-size: 10
```

```
algorithm:
  type: "distance" # distance, ml, hybrid
  ml-model-url: "http://ml-service:8080/predict"
  cache-ttl-seconds: 300
```

4.2. Интерфейс и реализации алгоритма назначения

```
// domain/service/DispatchAlgorithm.java
```

```
package com.logistics.domain.service;
```

```
import com.logistics.domain.model Courier;
```

```
import com.logistics.domain.model Order;
```

```
import java.util List;
```

```
import java.util Optional;
```

```
public interface DispatchAlgorithm {
```

```
    Optional<Courier> findBestCourier(Order order, List<Courier>
availableCouriers);
```

```

        String getAlgorithmType();
    }

    //      infrastructure/algorithm/LegacyDistanceAlgorithm.java      (старая
реализация)
    package com.logistics.infrastructure.algorithm;

    import com.logistics.domain.model Courier;
    import com.logistics.domain.model Order;
    import com.logistics.domain.service DispatchAlgorithm;
    import lombok.extern.slf4j.Slf4j;
    import org.springframework.stereotype.Component;

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

    @Component("legacyAlgorithm")
    @Slf4j
    public class LegacyDistanceAlgorithm implements DispatchAlgorithm {

        @Override
        public Optional<Courier> findBestCourier(Order order, List<Courier>
availableCouriers) {
            log.info("Using LEGACY distance-based algorithm");

            if (availableCouriers.isEmpty()) {
                return Optional.empty();
            }

```

```

// Простейший алгоритм: ближайший по прямой дистанции
return availableCouriers stream()
    .min(Comparator.comparingDouble(c ->
        c getCurrentLocation().distanceTo(order getPickupLocation())))
    .map(courier -> {
        log debug("Selected courier {} by distance", courier getId());
        return courier;
    });
}

@Override
public String getAlgorithmType() {
    return "legacy-distance";
}
}

// infrastructure/algorithm/NewMLAlgorithm.java (новая ML-реализация)
package com logistics infrastructure algorithm;

import com logistics domain model Courier;
import com logistics domain model Order;
import com logistics domain service DispatchAlgorithm;
import lombok RequiredArgsConstructor;
import lombok extern slf4j Slf4j;
import org springframework beans factory annotation Value;
import org springframework stereotype Component;
import org springframework web client RestTemplate;

```

```

import java.util HashMap;
import java.util List;
import java.util Map;
import java.util Optional;
import java.util stream Collectors;

@Component("mlAlgorithm")
@RequiredArgsConstructor
@Slf4j
public class NewMLAlgorithm implements DispatchAlgorithm {

    private final RestTemplate restTemplate;

    @Value("${algorithm.ml-model-url:http://ml-service:8080/predict}")
    private String mlModelUrl;

    @Override
    public Optional<Courier> findBestCourier(Order order, List<Courier>
availableCouriers) {
        log.info("Using NEW ML-based algorithm");

        if (availableCouriers.isEmpty()) {
            return Optional.empty();
        }

        try {
            // Подготовка данных для ML модели
            List<Map<String, Object>> courierFeatures =
availableCouriers.stream()

```



```

.map(c -> {
    Map<String, Object> features = new HashMap<>();
    features.put("courier_id", c.getId());
    features.put("distance_km", c.getCurrentLocation()
        .distanceTo(order.getPickupLocation()));
    features.put("current_load", c.getCurrentLoad());
    features.put("rating", c.getRating());
    features.put("avg_speed_kmh", c.getAverageSpeed());
    features.put("order_history_count", c.getTotalDeliveries());
    features.put("time_of_day",
java.time.LocalDateTime.now().getHour());
    return features;
})
.collect(Collectors.toList());

// Запрос к ML сервису
Map<String, Object> request = Map.of(
    "pickup_lat", order.getPickupLocation().lat(),
    "pickup_lng", order.getPickupLocation().lng(),
    "dropoff_lat", order.getDropoffLocation().lat(),
    "dropoff_lng", order.getDropoffLocation().lng(),
    "couriers", courierFeatures
);

@SuppressWarnings("unchecked")
Map<String, Object> response = restTemplate.postForObject(
    mlModelUrl, request, Map.class);

if (response != null && response.containsKey("best_courier_id")) {

```

```

        Long bestCourierId = ((Number)
response.get("best_courier_id")).longValue();

        return availableCouriers.stream()
            .filter(c -> c.getId().equals(bestCourierId))
            .findFirst();
    }

} catch (Exception e) {
    log.error("ML algorithm failed: {}", e.getMessage());
    // Fallback на legacy алгоритм
    return fallbackToLegacy(order, availableCouriers);
}

return Optional.empty();
}

private Optional<Courier> fallbackToLegacy(Order order, List<Courier>
availableCouriers) {
    log.warn("Falling back to legacy distance algorithm");
    return availableCouriers.stream()
        .min(Comparator.comparingDouble(c ->
            c.getCurrentLocation().distanceTo(order.getPickupLocation())));
}

@Override
public String getAlgorithmType() {
    return "ml-enhanced";
}
}

```

4.3. Фабрика с поддержкой Feature Toggle

```
// infrastructure/factory/DispatchAlgorithmFactory.java
package com.logistics.infrastructure.factory;

import com.logistics.domain.service DispatchAlgorithm;
import com.logistics.infrastructure.config FeatureToggleProperties;
import lombok.RequiredArgsConstructor;
import lombok.extern.slf4j.Slf4j;
import org.springframework.stereotype.Component;

import java.util.Map;
import java.util.concurrent.ThreadLocalRandom;

@Component
@RequiredArgsConstructor
@Slf4j
public class DispatchAlgorithmFactory {

    private final Map<String, DispatchAlgorithm> algorithms;
    private final FeatureToggleProperties featureToggle;

    /**
     * Получение алгоритма на основе feature toggle
     */
    public DispatchAlgorithm getAlgorithm() {
        String featureName = "dispatch.new-algorithm";

        // 1. Проверяем глобальный toggle
```

```
if (!featureToggle isEnabled(featureName)) {  
    log debug("Feature {} is disabled, using legacy algorithm",  
featureName);  
    return algorithms.get("legacyAlgorithm");  
}
```

// 2. Проверяем процент включения (canary rollout)

```
int percentage = featureToggle getPercentage(featureName);  
if (percentage > 0 && percentage < 100) {  
    int random = ThreadLocalRandom.current().nextInt(100);  
    if (random < percentage) {  
        log debug("Feature {} enabled for this request ({}% rollout)",  
featureName, percentage);  
        return algorithms.get("mlAlgorithm");  
    } else {  
        log debug("Feature {} disabled for this request ({}% rollout)",  
featureName, percentage);  
        return algorithms.get("legacyAlgorithm");  
    }  
}
```

// 3. Полностью включено

```
if (percentage >= 100) {  
    log debug("Feature {} fully enabled", featureName);  
    return algorithms.get("mlAlgorithm");  
}  
  
return algorithms.get("legacyAlgorithm");  
}
```

```

/**
 * Получение алгоритма на основе ID заказа (consistent rollout)
 */
public DispatchAlgorithm getAlgorithmForOrder(Long orderId) {
    String featureName = "dispatch.new-algorithm";

    if (!featureToggle isEnabled(featureName)) {
        return algorithms.get("legacyAlgorithm");
    }

    int percentage = featureToggle getPercentage(featureName);

    if (percentage >= 100) {
        return algorithms.get("mlAlgorithm");
    }

    // Consistent: один и тот же заказ всегда получает один алгоритм
    boolean useNew = (orderId % 100) < percentage;

    return useNew ? algorithms.get("mlAlgorithm") :
algorithms.get("legacyAlgorithm");
}
}

```

4.4. Feature Toggle конфигурация и обновление

// infrastructure/config/FeatureToggleProperties.java

```
package com logistics infrastructure config;
```

```
import lombok Data;
```

```

import
org.springframework.boot.context.properties ConfigurationProperties;
import org.springframework.cloud.context.config.annotation RefreshScope;
import org.springframework.stereotype Component;

import java.util HashMap;
import java.util Map;

@Component
@ConfigurationProperties(prefix = "features")
@RefreshScope // Позволяет обновлять конфигурацию без перезапуска
@Data
public class FeatureToggleProperties {

    private Map<String, FeatureConfig> dispatch = new HashMap<>();
    private Map<String, FeatureConfig> payment = new HashMap<>();
    private Map<String, FeatureConfig> notification = new HashMap<>();

    @Data
    public static class FeatureConfig {
        private boolean enabled = false;
        private int percentage = 0;
        private boolean fallbackEnabled = true;
        private boolean metricsEnabled = true;
        private String description = "";
    }

    public boolean isEnabled(String featurePath) {
        String[] parts = featurePath.split("\\.");
    }

```

```

        if (parts.length != 2) return false;

        String category = parts[0];
        String feature = parts[1];

        return switch (category) {
            case "dispatch" -> dispatch.getOrDefault(feature, new
FeatureConfig()).isEnabled();
            case "payment" -> payment.getOrDefault(feature, new
FeatureConfig()).isEnabled();
            case "notification" -> notification.getOrDefault(feature, new
FeatureConfig()).isEnabled();
            default -> false;
        };
    }
}

```

```

public int getPercentage(String featurePath) {
    String[] parts = featurePath.split("\\.");
    if (parts.length != 2) return 0;

    String category = parts[0];
    String feature = parts[1];

    return switch (category) {
        case "dispatch" -> dispatch.getOrDefault(feature, new
FeatureConfig()).getPercentage();
        case "payment" -> payment.getOrDefault(feature, new
FeatureConfig()).getPercentage();
    };
}

```

```
        case "notification" -> notification getOrDefault(feature, new  
FeatureConfig()).getPercentage();  
        default -> 0;  
    };  
}  
}
```

application.yml (локальная конфигурация)

features:

dispatch:

new-algorithm:

enabled: false

percentage: 0

fallback-enabled: true

metrics-enabled: true

description: "New ML-based dispatch algorithm"

priority-queue:

enabled: true

percentage: 100

description: "Priority queue for express orders"

batch-dispatch:

enabled: false

percentage: 0

description: "Batch dispatch for multiple orders"

management:

endpoints:


```
web:
  exposure:
    include: refresh,health,info,metrics
```

4.5. REST API для управления тоглами

```
// infrastructure/web/FeatureToggleController.java
```

```
package com.logistics.infrastructure.web;
```

```
import com.logistics.infrastructure.config FeatureToggleProperties;
import lombok RequiredArgsConstructor;
import org.springframework.cloud.context.refresh ContextRefresher;
import org.springframework.http ResponseEntity;
import org.springframework.web.bind.annotation *;
```

```
import java.util HashMap;
import java.util Map;
```

```
@RestController
@RequestMapping("/admin/features")
@RequiredArgsConstructor
public class FeatureToggleController {
```

```
    private final FeatureToggleProperties featureToggle;
    private final ContextRefresher contextRefresher;
```

```
@GetMapping
public ResponseEntity<Map<String, Object>> getAllFeatures() {
    Map<String, Object> features = new HashMap<>();
    features.put("dispatch", featureToggle.getDispatch());
    features.put("payment", featureToggle.getPayment());
```

```

        features.put("notification", featureToggle.getNotification());
        return ResponseEntity.ok(features);
    }

    @GetMapping("/{category}/{feature}")
    public ResponseEntity<FeatureToggleProperties.FeatureConfig>
getFeature(
        @PathVariable String category,
        @PathVariable String feature) {

        Map<String, FeatureToggleProperties.FeatureConfig> categoryMap =
switch (category) {
    case "dispatch" -> featureToggle.getDispatch();
    case "payment" -> featureToggle.getPayment();
    case "notification" -> featureToggle.getNotification();
    default -> null;
};

        if (categoryMap == null || !categoryMap.containsKey(feature)) {
            return ResponseEntity.notFound().build();
        }

        return ResponseEntity.ok(categoryMap.get(feature));
    }

    @PostMapping("/refresh")
    public ResponseEntity<Void> refreshConfig() {
        // Обновление конфигурации из Config Server
        contextRefresher.refresh();
    }

```

```
        return ResponseEntity.ok().build();
    }
}
```

4.6. Use Case с поддержкой Feature Toggle

// application/usecase/AssignOrderUseCase.java

package com.logistics.application.usecase;

```
import com.logistics.domain.model Courier;
import com.logistics.domain.model Order;
import com.logistics.domain.repository.CourierRepositoryPort;
import com.logistics.domain.repository.OrderRepositoryPort;
import com.logistics.domain.service.DispatchAlgorithm;
import com.logistics.infrastructure.factory.DispatchAlgorithmFactory;
import com.logistics.infrastructure.metrics.DispatchMetrics;
import lombok.RequiredArgsConstructor;
import lombok.extern.slf4j.Slf4j;
import org.springframework.stereotype.Component;

import java.util.List;
```

@Component

@RequiredArgsConstructor

@Slf4j

public class AssignOrderUseCase {

```
    private final OrderRepositoryPort orderRepository;
    private final CourierRepositoryPort courierRepository;
    private final DispatchAlgorithmFactory algorithmFactory;
    private final DispatchMetrics metrics;
```

```

public void execute(Long orderId) {
    // 1. Получаем заказ
    Order order = orderRepository findById(orderId)
        .orElseThrow(() -> new IllegalArgumentException("Order not found:
" + orderId));

    // 2. Получаем доступных курьеров
    List<Courier> availableCouriers = courierRepository findAvailable();

    if (availableCouriers isEmpty()) {
        throw new IllegalStateException("No couriers available");
    }

    // 3. Выбираем алгоритм через factory (с учётом feature toggle)
    long startTime = System.currentTimeMillis();
    DispatchAlgorithm algorithm =
algorithmFactory getAlgorithmForOrder(orderId);

    log info("Using algorithm: {} for order {}",
algorithm getAlgorithmType(), orderId);

    // 4. Находим лучшего курьера
    var bestCourier = algorithm.findBestCourier(order, availableCouriers);

    // 5. Метрики
    long duration = System.currentTimeMillis() - startTime;
    metrics recordDispatchAttempt(algorithm getAlgorithmType(),
bestCourier.isPresent(), duration);

```

```

    if (bestCourier isEmpty()) {
        throw new IllegalStateException("No suitable courier found");
    }

    // 6. Назначаем курьера
    order assignToCourier(bestCourier get());
    orderRepository save(order);

    log info("Order {} assigned to courier {} using {}",
        orderId, bestCourier get().getId(), algorithm getAlgorithmType());
}
}

```

4.7. Переменные окружения (альтернативный простой способ)

// infrastructure/config/SimpleFeatureToggle.java

```
package com logistics infrastructure config;
```

```
import org springframework beans factory annotation Value;
```

```
import org springframework stereotype Component;
```

```
/**
```

```
 * Простой Feature Toggle через переменные окружения.
```

```
 * Требуется перезапуска при изменении.
```

```
 */
```

```
@Component
```

```
public class SimpleFeatureToggle {
```

```
    @Value("${FEATURE_NEW_DISPATCH:false}")
```

```
    private boolean newDispatchEnabled;
```

```
@Value("${FEATURE_ML_ALGORITHM:false}")
```

```
private boolean mlAlgorithmEnabled;
```

```
@Value("${FEATURE_BATCH_DISPATCH:false}")
```

```
private boolean batchDispatchEnabled;
```

```
@Value("${FEATURE_NEW_DISPATCH_PERCENTAGE:0}")
```

```
private int newDispatchPercentage;
```

```
public boolean isNewDispatchEnabled() {
```

```
    return newDispatchEnabled;
```

```
}
```

```
public boolean isMlAlgorithmEnabled() {
```

```
    return mlAlgorithmEnabled;
```

```
}
```

```
public boolean isBatchDispatchEnabled() {
```

```
    return batchDispatchEnabled;
```

```
}
```

```
public boolean shouldUseNewDispatchForOrder(Long orderId) {
```

```
    if (!newDispatchEnabled) return false;
```

```
    if (newDispatchPercentage >= 100) return true;
```

```
    if (newDispatchPercentage <= 0) return false;
```

```
    return (orderId % 100) < newDispatchPercentage;
```

```
}
```

```
}
```

```
# .env file

FEATURE_NEW_DISPATCH=false
FEATURE_ML_ALGORITHM=false
FEATURE_BATCH_DISPATCH=false
FEATURE_NEW_DISPATCH_PERCENTAGE=0

# Для включения новой фичи:
# export FEATURE_NEW_DISPATCH=true
# export FEATURE_NEW_DISPATCH_PERCENTAGE=10
# docker restart logistics-service
```

4.8. Тестирование Feature Toggles

```
// test/feature/FeatureToggleTest.java

package com.logistics.test.feature;

import com.logistics.application.usecase.AssignOrderUseCase;
import com.logistics.infrastructure.config.FeatureToggleProperties;
import com.logistics.infrastructure.factory.DispatchAlgorithmFactory;
import org.junit.jupiter.api.Test;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.context.SpringBootTest;
import org.springframework.test.context.TestPropertySource;

import static org.assertj.core.api.Assertions.assertThat;

@SpringBootTest
class FeatureToggleTest {
```

```

@Autowired
private DispatchAlgorithmFactory algorithmFactory;

@Autowired
private FeatureToggleProperties featureToggle;

@Test
@TestPropertySource(properties = {"features.dispatch.new-
algorithm.enabled=false"})
void shouldUseLegacyAlgorithmWhenToggleDisabled() {
    var algorithm = algorithmFactory.getAlgorithm();
    assertThat(algorithm.getAlgorithmType()).isEqualTo("legacy-
distance");
}

@Test
@TestPropertySource(properties = {
    "features.dispatch.new-algorithm.enabled=true",
    "features.dispatch.new-algorithm.percentage=100"
})
void shouldUseNewAlgorithmWhenToggleEnabled() {
    var algorithm = algorithmFactory.getAlgorithm();
    assertThat(algorithm.getAlgorithmType()).isEqualTo("ml-enhanced");
}

@Test
@TestPropertySource(properties = {
    "features.dispatch.new-algorithm.enabled=true",

```



```

        "features.dispatch.new-algorithm.percentage=30"
    })

    void shouldDistributeByPercentage() {
        int newCount = 0;
        int legacyCount = 0;

        for (int i = 0; i < 1000; i++) {
            var algorithm = algorithmFactory getAlgorithm();
            if ("ml-enhanced" equals(algorithm getAlgorithmType())) {
                newCount++;
            } else {
                legacyCount++;
            }
        }
    }
}

```

```

// При 30% должно быть около 300 вызовов новой версии
assertThat(newCount).isBetween(250, 350);
assertThat(legacyCount).isBetween(650, 750);
}

```

```

@Test
@TestPropertySource(properties = {
    "features.dispatch.new-algorithm.enabled=true",
    "features.dispatch.new-algorithm.percentage=50"
})

void shouldBeConsistentForSameOrder() {
    Long orderId = 123L;
}

```

```

// Один и тот же заказ должен всегда получать один алгоритм

```

```

        String first =
algorithmFactory getAlgorithmForOrder(orderId).getAlgorithmType();

        for (int i = 0; i < 100; i++) {
            String current =
algorithmFactory getAlgorithmForOrder(orderId).getAlgorithmType();
            assertThat(current).isEqualTo(first);
        }
    }
}

```

// test/feature/FeatureToggleIntegrationTest.java

@SpringBootTest

@AutoConfigureMockMvc

class FeatureToggleIntegrationTest {

@Autowired

private MockMvc mockMvc;

@Test

void shouldRefreshConfigViaEndpoint() throws Exception {

// Изменяем конфигурацию через Config Server (имитация)

// Обновляем via endpoint

mockMvc.perform(post("/admin/features/refresh"))
 .andExpect(status().isOk());

// Проверяем, что конфигурация обновилась

mockMvc.perform(get("/admin/features/dispatch/new-algorithm"))

```

        .andExpect(status().isOk());
    }

    @Test
    void shouldReturnCurrentFeatureState() throws Exception {
        mockMvc.perform(get("/admin/features"))
            .andExpect(status().isOk())
            .andExpect(jsonPath("$.dispatch.new-algorithm.enabled").exists());
    }
}

```

4.9. Мониторинг Feature Toggles

// infrastructure/monitoring/FeatureToggleMetrics.java

package com logistics infrastructure monitoring;

```

import com logistics infrastructure config FeatureToggleProperties;
import io micrometer core instrument MeterRegistry;
import lombok RequiredArgsConstructor;
import org springframework scheduling annotation Scheduled;
import org springframework stereotype Component;

```

```

@Component
@RequiredArgsConstructor
public class FeatureToggleMetrics {

```

```

    private final FeatureToggleProperties featureToggle;
    private final MeterRegistry meterRegistry;

```

```

    @Scheduled(fixedDelay = 30000) // каждые 30 секунд
    public void recordFeatureToggles() {

```

```

// Записываем состояние каждого toggle в метрики
featureToggle.getDispatch().foreach((name, config) -> {
    meterRegistry gauge("feature.toggle.enabled",
        java.util.Collections.singletonList("feature=" + name),
        config
        FeatureToggleProperties FeatureConfig::isEnabled
    );

    meterRegistry gauge("feature.toggle.percentage",
        java.util.Collections.singletonList("feature=" + name),
        config
        FeatureToggleProperties FeatureConfig::getPercentage
    );
});
}
}
}

```

4.10. Диаграмма последовательности с Feature Toggle

@startuml

!theme plain

```

actor "Диспетчер" as Dispatcher
participant "API" as API
participant "AssignOrderUseCase" as UseCase
participant "FeatureToggle\nChecker" as Toggle
participant "AlgorithmFactory" as Factory
participant "LegacyAlgorithm" as Legacy
participant "MLAlgorithm" as ML

```

Dispatcher -> API: POST /assign/{orderId}

API -> UseCase: execute(orderId)

UseCase -> Toggle: isEnabled("dispatch.new-algorithm")

Toggle --> UseCase: true

UseCase -> Toggle: getPercentage("dispatch.new-algorithm")

Toggle --> UseCase: 10%

alt orderId % 100 < 10

UseCase -> Factory: getAlgorithm()

Factory -> ML: new MLAlgorithm()

Factory --> UseCase: mlAlgorithm

UseCase -> ML: findBestCourier(order, couriers)

ML -> ML: ML prediction

ML --> UseCase: bestCourier

else

UseCase -> Factory: getAlgorithm()

Factory -> Legacy: new LegacyAlgorithm()

Factory --> UseCase: legacyAlgorithm

UseCase -> Legacy: findBestCourier(order, couriers)

Legacy -> Legacy: distance calculation

Legacy --> UseCase: bestCourier

end

UseCase -> UseCase: assign courier

UseCase --> API: success

API --> Dispatcher: 200 OK

@enduml

4.11. Canary rollout план

```
// infrastructure/rollout/CanaryRolloutManager.java
```

```
package com logistics infrastructure rollout;
```

```
import lombok extern slf4j Slf4j;
```

```
import org springframework stereotype Component;
```

```
@Component
```

```
@Slf4j
```

```
public class CanaryRolloutManager {
```

```
    /**
```

```
    * План постепенного увеличения процента включения новой фичи
```

```
    */
```

```
public enum RolloutStage {
```

```
    STAGE_0(0, "Feature disabled, use legacy"),
```

```
    STAGE_1(1, "1% - Internal testing only"),
```

```
    STAGE_2(5, "5% - Selected beta customers"),
```

```
    STAGE_3(10, "10% - Monitoring errors and metrics"),
```

```
    STAGE_4(25, "25% - Half of non-critical traffic"),
```

```
    STAGE_5(50, "50% - A/B test with control group"),
```

```
    STAGE_6(90, "90% - Almost full rollout"),
```

```
    STAGE_7(100, "100% - Full rollout, legacy can be removed");
```

```
private final int percentage;
```

```
private final String description;
```

```
RolloutStage(int percentage, String description) {
    this.percentage = percentage;
    this.description = description;
}

/**
 * Проверка готовности к переходу на следующий этап
 */
public boolean isReadyForNextStage(String featureName, int
currentPercentage) {
    // Проверяем метрики ошибок
    double errorRate = getErrorRateForFeature(featureName);
    if (errorRate > 0.01) { // > 1% ошибок
        log.warn("Error rate {}% too high for feature {}, not advancing",
            errorRate * 100, featureName);
        return false;
    }

    // Проверяем латентность
    long p95Latency = getP95LatencyForFeature(featureName);
    if (p95Latency > 500) { // > 500ms
        log.warn("Latency {}ms too high for feature {}, not advancing",
            p95Latency, featureName);
        return false;
    }

    return true;
}
```

```
private double getErrorRateForFeature(String featureName) {
    // В реальности - из метрик
    return 0.001;
}
```

```
private long getP95LatencyForFeature(String featureName) {
    // В реальности - из метрик
    return 200L;
}
```

5. Индивидуальные задания (15 вариантов)

Важно: Студент работает со своим сквозным проектом (из ЛР 1).

Вариант	Сквозной проект	Feature Toggle (архитектурное изменение)
1	Логистика	Новый ML-алгоритм назначения курьера
2	Интернет-магазин	Новая система рекомендаций товаров
3	Финтех-кошелёк	Новая скоринговая система для транзакций
4	Библиотека	Новая система поиска книг (Elasticsearch)
5	Отели	Новый алгоритм ценообразования

Вариант	Сквозной проект	Feature Toggle (архитектурное изменение)
6	Онлайн-курсы	Новая система выдачи сертификатов
7	Медицина	Новая система записи к врачу
8	CRM	Новая сегментация клиентов (ML)
9	Платёжный шлюз	Новая антифрод-система
10	Склад (WMS)	Новая стратегия пополнения запасов
11	Социальная сеть	Новая формула ранжирования ленты
12	Видеохостинг	Новый алгоритм рекомендаций видео
13	IoT-мониторинг	Новое обнаружение аномалий
14	Криптовалюта	Новый механизм сопоставления ордеров
15	Доставка еды	Новый алгоритм группировки заказов

Задание для студента:

1. Для своего проекта (по варианту):

2. Создать Feature Toggle (через Config Server или переменные окружения).
3. Реализовать две альтернативные реализации (старая и новая).
4. Создать фабрику для выбора реализации на основе toggle.
5. Реализовать REST API для управления тоглами.
6. Реализовать постепенное включение (percentage-based rollout).
7. Написать тесты (проверка включения/выключения).
8. Составить canary rollout план.

6. Контрольные вопросы

1. Что такое Feature Toggle и для чего он используется?
2. Какие типы Feature Toggles существуют?
3. В чём разница между release toggle и experiment toggle?
4. Как реализовать процентное включение (canary release)?
5. Как обеспечить консистентность (один пользователь всегда видит одну версию)?
6. Как обновить toggle без перезапуска приложения?
7. Какие метрики нужно мониторить при использовании feature toggles?
8. Как удалять старый код после полного включения toggle?

7. Требования к отчёту

Отчёт должен содержать:

Титульный лист.

Цель и задачи работы.

Описание архитектурного изменения.

Две реализации (код старой и новой).

Factory с toggle.

Конфигурация toggles (application.yml или переменные окружения).

REST API для управления.

Процентное включение (rollout).

Тесты (включено/выключено, процентное распределение).

Canary rollout план.

Диаграмма последовательности.

Вывод.

Ответы на вопросы.

8. Критерии оценивания

Критерий	max балл
Feature Toggle конфигурация (2 способа)	2
Две альтернативные реализации (старая + новая)	2
Фабрика с выбором на основе toggle	2
Процентное включение (percentage-based)	2
REST API для управления тоглами	2
Тесты (включено/выключено, процент)	3
Canary rollout план	2
Соответствие варианту	1
Итого	16

Оценка архитектурного технического долга на примере «Системы управления логистикой и доставкой»

1. Цель работы

Сформировать компетенции по выявлению, инвентаризации и оценке архитектурного технического долга: анализ кода и архитектуры проекта, выявление проблем (циклические зависимости, несоответствие слоям, дублирование логики, отсутствие интерфейсов), оценка стоимости исправления vs стоимость владения, приоритизация и составление плана рефакторинга для выбранного сквозного проекта.

2. Задачи работы

1. Изучить понятие архитектурного технического долга и его виды.
2. Провести анализ кода и архитектуры своего сквозного проекта.
3. Выявить минимум 5 проблем (циклические зависимости, несоответствие слоям, дублирование, отсутствие интерфейсов, жёсткие связи).
4. Оценить стоимость исправления и стоимость владения каждой проблемы.
5. Составить бэклог технического долга с приоритизацией (Cost of Delay).
6. Предложить план рефакторинга на 2–3 итерации.
7. Оформить результаты в виде отчёта с таблицами, диаграммами и планом.

3. Теоретическое обоснование

3.1. Что такое архитектурный технический долг?

Технический долг - это метафора, описывающая последствия принятия быстрых, но неоптимальных архитектурных решений, которые требуют будущих доработок.



3.2. Признаки архитектурного технического долга

Признак	Описание	Приме р
Циклические зависимости	Модули зависят друг от друга	$A \rightarrow B \rightarrow A$
Нарушение слоёв	UI вызывает БД напрямую	Контрол лер использует Repository
Дублирование логики	Одна и та же логика в разных местах	Расчёт ETA в 3 сервисах

Признак	Описание	Приме р
Отсутствие интерфейсов	Жёсткие связи с реализациями	private JpaRepository в Use Case
God Object	Класс, который делает всё	Utils с 50 методами
Shotgun Surgery	Изменение требует правок в 10+ местах	Добавле ние поля в заказ
Библиотеки- монстры	Непропорционально большие зависимости	commons .jar на 20 MB

3.3. Оценка технического долга

Параметр	Описание	Шкала
Severity (Серьёзность)	Влияние на систему	1-5 (1=низкое, 5=критическое)
Effort (Усилия)	Время на исправление	1-5 (1=1 час, 5=2+ недели)
Cost of Delay (CoD)	Потери от отсрочки	1-5 (1=низкие, 5=критические)

Параметр	Описание	Шкала
Priority	Итоговый приоритет	CoD × Severity / Effort

4. Пример выполнения

Сквозной пример: «Система управления логистикой и доставкой»
(вариант 1)

4.1. Анализ архитектуры (диаграмма зависимостей)

@startuml

!theme plain

' Анализ текущих зависимостей в проекте

```
package "Presentation Layer" {
    [CourierController]
    [OrderController]
    [DispatchController]
    [AdminController]
}
```

```
package "Application Layer" {
    [CourierService]
    [OrderService]
    [DispatchService]
    [AssignmentService]
    [NotificationService]
    [PaymentService]
}
```

```
package "Domain Layer" {  
    [Courier]  
    [Order]  
    [Assignment]  
    [DomainEvent]  
    [RepositoryPorts]  
}
```

```
package "Infrastructure Layer" {  
    [CourierRepository]  
    [OrderRepository]  
    [AssignmentRepository]  
    [KafkaProducer]  
    [RedisCache]  
    [MLClient]  
}
```

' Проблемные зависимости (красные)

CourierController --> [CourierRepository] : ❌ Прямой вызов
репозитория

OrderController --> [OrderRepository] : ❌ Прямой вызов репозитория

[CourierService] --> [CourierRepository] : ✅ Через порт

[OrderService] --> [OrderRepository] : ✅ Через порт

[DispatchService] --> [OrderService] : ❌ Циклическая зависимость

[OrderService] --> [DispatchService] : ❌ Циклическая зависимость

[KafkaProducer] --> [RedisCache] : ❌ Неправильное направление

[RedisCache] --> [MLClient] : ❌ Спагетти-код

@enduml

4.2. Инструменты анализа кода

// tool/ArchitectureAnalyzer.java

package com.logistics.tool;

import com.github.javaparser.StaticJavaParser;

import com.github.javaparser.ast.CompilationUnit;

import com.github.javaparser.ast.body.ClassOrInterfaceDeclaration;

import com.github.javaparser.ast.expr.AnnotationExpr;

import com.github.javaparser.ast.type.ClassOrInterfaceType;

import lombok.Data;

import org.springframework.stereotype.Component;

import java.io.File;

import java.nio.file.Files;

import java.nio.file.Path;

import java.nio.file.Paths;

import java.util.*;

import java.util.stream.Collectors;

@Component

public class ArchitectureAnalyzer {

private final Map<String, Set<String>> dependencies = new
HashMap<>();

```
private final List<String> layers = List of("controller", "service",  
"repository", "entity");
```

```
public AnalysisResult analyze(String projectPath) throws Exception {  
    AnalysisResult result = new AnalysisResult();
```

```
    // 1. Собираем все Java файлы
```

```
    List<Path> javaFiles = Files.walk(Paths.get(projectPath))  
        .filter(p -> p.toString().endsWith(".java"))  
        .collect(Collectors.toList());
```

```
    // 2. Анализируем зависимости
```

```
    for (Path file : javaFiles) {  
        analyzeFile(file, result);  
    }
```

```
    // 3. Ищем циклические зависимости
```

```
    result.setCyclicDependencies(findCyclicDependencies());
```

```
    // 4. Ищем нарушение слоёв
```

```
    result.setLayerViolations(findLayerViolations());
```

```
    // 5. Ищем дублирование кода
```

```
    result.setCodeDuplication(findCodeDuplication(javaFiles));
```

```
    // 6. Ищем God Objects
```

```
    result.setGodObjects(findGodObjects(javaFiles));
```

```
    return result;
```

```
}
```

```
private void analyzeFile(Path file, AnalysisResult result) {  
    try {  
        CompilationUnit cu = StaticJavaParser parse(file toFile());  
  
        cu findAll(ClassOrInterfaceDeclaration class).forEach(cls -> {  
            String className = cls getNameAsString();  
            String packageName = cu getPackageDeclaration()  
                .map(p -> p getNameAsString())  
                .orElse("");  
  
            String fullName = packageName + "." + className,  
  
            // Ищем зависимости  
            Set<String> deps = new HashSet<>();  
            cls.findAll(ClassOrInterfaceType.class).forEach(type -> {  
                String depName = type getNameAsString();  
                if (!depName.startsWith("java")) &&  
                    !depName.equals(className)) {  
                    deps add(depName);  
                }  
            });  
  
            dependencies put(fullName, deps);  
  
            // Проверяем аннотации для определения слоя  
            String layer = detectLayer(cls);  
            String layerIssue = checkLayerViolation(cls, layer);
```

```

        if (layerIssue != null) {
            result addLayerViolation(fullName, layerIssue);
        }
    });

} catch (Exception e) {
    System.err.println("Failed to parse: " + file);
}
}

private String detectLayer(ClassOrInterfaceDeclaration cls) {
    for (AnnotationExpr ann : cls.getAnnotations()) {
        String name = ann.getNameAsString();
        if (name.contains("Controller")) return "controller";
        if (name.contains("Service")) return "service";
        if (name.contains("Repository")) return "repository";
        if (name.contains("Entity") || name.contains("Table")) return "entity";
    }
    return "unknown";
}

private String checkLayerViolation(ClassOrInterfaceDeclaration cls,
String layer) {
    // Контроллер не должен напрямую использовать репозиторий
    if ("controller".equals(layer)) {
        boolean usesRepository = cls.toString().contains("Repository");
        if (usesRepository) {
            return "Controller should not use Repository directly. Use Service
instead.";
        }
    }
}

```

```
    }  
}
```

// Сервис не должен использовать другой сервис из другого модуля
напрямую

```
if ("service".equals(layer)) {  
    // Проверка на циклические зависимости между сервисами  
}  
  
return null;  
}
```

```
private List<CyclicDependency> findCyclicDependencies() {  
    List<CyclicDependency> cycles = new ArrayList<>();  
    // Алгоритм поиска циклов в графе зависимостей  
    // (реализация упрощена)  
    return cycles;  
}
```

```
private List<LayerViolation> findLayerViolations() {  
    List<LayerViolation> violations = new ArrayList<>();  
    // Анализ нарушения слоёв (controller → repository и т.д.)  
    return violations;  
}
```

```
private List<CodeDuplication> findCodeDuplication(List<Path> files) {  
    List<CodeDuplication> duplications = new ArrayList<>();  
    // Поиск дублирования кода (например, с помощью CPD)  
    return duplications;  
}
```

```
}
```

```
private List<GodObject> findGodObjects(List<Path> files) {  
    List<GodObject> godObjects = new ArrayList<>();  
    // Поиск классов с > 30 методами или > 1000 строк  
    return godObjects;  
}
```

```
@Data
```

```
public static class AnalysisResult {  
    private List<CyclicDependency> cyclicDependencies = new  
ArrayList<>();  
    private List<LayerViolation> layerViolations = new ArrayList<>();  
    private List<CodeDuplication> codeDuplication = new ArrayList<>();  
    private List<GodObject> godObjects = new ArrayList<>();  
  
    public void addLayerViolation(String className, String issue) {  
        layerViolations add(new LayerViolation(className, issue));  
    }  
}
```

```
@Data
```

```
public static class CyclicDependency {  
    private List<String> cycle;  
}
```

```
@Data
```

```
public static class LayerViolation {  
    private final String className;
```

```
    private final String issue;
}
```

```
@Data
```

```
public static class CodeDuplication {
    private String fragment;
    private List<String> locations;
    private int lines;
}
```

```
@Data
```

```
public static class GodObject {
    private String className;
    private int methodCount;
    private int lineCount;
}
}
```

4.3. Инвентаризация технического долга

```
// model/TechnicalDebtItem.java
```

```
package com logistics techdebt
```

```
import lombok.Builder;
```

```
import lombok.Data;
```

```
@Data
```

```
@Builder
```

```
public class TechnicalDebtItem {
```

```
    private String id;
```

```
    private String category;           // architecture, code, test, documentation
```

```
private String title;
private String description;
private String location;           // файл или класс

// Оценки
private int severity;              // 1-5 (влияние на систему)
private int effort;               // 1-5 (часы/дни)
private int costOfDelay;          // 1-5 (потери от отсрочки)
private double priority;          // (severity + costOfDelay) / effort

private String suggestedSolution;
private String estimatedHours;
private String assignedTo;
private DebtStatus status;

public enum DebtStatus {
    IDENTIFIED, IN_PROGRESS, FIXED, ACCEPTED
}

}

// service/TechDebtInventoryService.java
package com logistics techdebt

import lombok.extern slf4j.Slf4j;
import org.springframework.stereotype.Service;

import java.util ArrayList;
import java.util List;
import java.util stream Collectors;
```



```

@Service
@Slf4j
public class TechDebtInventoryService {

    /**
     * Полный бэклог технического долга для логистической системы
     */

    public List<TechnicalDebtItem> getBacklog() {
        List<TechnicalDebtItem> backlog = new ArrayList<>();

        // 1. Циклическая зависимость между сервисами
        backlog.add(TechnicalDebtItem.builder()
            .id("ARCH-001")
            .category("architecture")
            .title("Циклическая зависимость между DispatchService и
OrderService")
            .description("""
                DispatchService вызывает OrderService для получения
информации о заказе,
                а OrderService вызывает DispatchService для проверки статуса
назначения.
                Это создаёт циклическую зависимость и усложняет
тестирование.
                """)
            .location("DispatchService.java, OrderService.java")
            .severity(5) // критично
            .effort(4) // 2-3 дня
            .costOfDelay(4) // высокие потери

```

```
.priority(2.25) // (5+4)/4 = 2.25
```

```
.suggestedSolution("""
```

Ввести промежуточный DomainEvent для связи между сервисами.

DispatchService публикует AssignmentCreatedEvent,
OrderService подписывается и обновляет статус заказа.

```
""")
```

```
.estimatedHours("16-20 часов")
```

```
.status(TechnicalDebtItem.DebtStatus.IDENTIFIED)
```

```
.build());
```

// 2. Нарушение слоёв (Controller → Repository)

```
backlog.add(TechnicalDebtItem.builder()
```

```
.id("ARCH-002")
```

```
.category("architecture")
```

```
.title("Контроллер напрямую вызывает Repository")
```

```
.description("""
```

CourierController напрямую использует CourierRepository, минуя сервисный слой. Бизнес-логика размазана по контроллерам.

```
""")
```

```
.location("CourierController.java:45")
```

```
.severity(4)
```

```
.effort(2) // 4-6 часов
```

```
.costOfDelay(3)
```

```
.priority(3.5) // (4+3)/2 = 3.5
```

```
.suggestedSolution("""
```

Перенести бизнес-логику из контроллера в CourierService.

Контроллер должен вызывать только сервис.

```
""")
.estimatedHours("4-6 часов")
.status(TechnicalDebtItem.DebtStatus.IDENTIFIED)
.build());
```

// 3. Дублирование логики расчёта ETA

```
backlog add(TechnicalDebtItem.builder()
.id("CODE-001")
.category("code")
.title("Дублирование логики расчёта ETA в 3 местах")
.description("""
```

Расчёт ETA (Estimated Time of Arrival) реализован в:

- DispatchService.calculateETA()
- TrackingService.estimateArrival()
- NotificationService.formatETA()

При изменении формулы нужно править 3 места.

```
""")
.location("DispatchService:234, TrackingService:89,
NotificationService:56")
.severity(3)
.effort(2)
.costOfDelay(2)
.priority(2.5)
.suggestedSolution("""
```

Создать отдельный сервис ETACalculator с единственной ответственностью.

Использовать его во всех трёх местах.

```
""")
```

```
.estimatedHours("3-5 часов")
.status(TechnicalDebtItem.DebtStatus.IDENTIFIED)
.build();
```

// 4. Отсутствие интерфейсов для репозиториев

```
backlog.add(TechnicalDebtItem.builder()
    .id("ARCH-003")
    .category("architecture")
    .title("Отсутствие интерфейсов для репозиториев")
    .description("""
```

В Clean Architecture репозитории должны быть интерфейсами
в Domain слое, но в проекте используются конкретные JPA-
репозитории

напрямую в Use Cases.

```
""")
.location("com.logistics.application.usecase.*")
.severity(4)
.effort(3) // 8-12 часов
.costOfDelay(3)
.priority(2.33)
.suggestedSolution("""
```

Создать интерфейсы репозиториев в Domain слое.

Реализовать адаптеры в Infrastructure.

Использовать Dependency Injection через конструктор.

```
""")
.estimatedHours("8-12 часов")
.status(TechnicalDebtItem.DebtStatus.IDENTIFIED)
.build();
```

// 5. God Object в классе Utils

```
backlog add(TechnicalDebtItem.builder()
    .id("CODE-002")
    .category("code")
    .title("God Object: класс LogisticsUtils содержит 45 методов")
    .description("""
        Класс LogisticsUtils содержит утилиты для:
        - форматирования дат
        - расчёта расстояний
        - валидации адресов
        - работы с JSON
        - логирования
        Нарушает Single Responsibility Principle.
        """)
    .location("LogisticsUtils.java")
    .severity(3)
    .effort(3) // 8 часов
    .costOfDelay(2)
    .priority(1.67)
    .suggestedSolution("""
        Разделить на отдельные утилитарные классы:
        - DateFormatter
        - DistanceCalculator
        - AddressValidator
        - JsonHelper
        - LoggingUtils
        """)
    .estimatedHours("6-8 часов")
    .status(TechnicalDebtItem.DebtStatus.IDENTIFIED)
```

```
.build());
```

// 6. Отсутствие модульных тестов для Use Cases

```
backlog add(TechnicalDebtItem.builder()
    .id("TEST-001")
    .category("test")
    .title("Use Cases не покрыты модульными тестами")
    .description("""
        AssignOrderUseCase, UpdateLocationUseCase и другие
        не имеют unit-тестов. Покрытие бизнес-логики отсутствует.
        """)
    .location("src/test/java/application/usecase/")
    .severity(4)
    .effort(4) // 2 дня
    .costOfDelay(4)
    .priority(2.0)
    .suggestedSolution("""
        Написать unit-тесты для всех Use Cases с использованием
        Mockito.

        Минимальное покрытие: 80% для Application слоя.
        """)
    .estimatedHours("12-16 часов")
    .status(TechnicalDebtItem.DebtStatus.IDENTIFIED)
    .build());
```

// 7. Жёсткая привязка к конкретному картографическому провайдеру

```
backlog add(TechnicalDebtItem.builder()
    .id("ARCH-004")
```

```
.category("architecture")
.title("Жёсткая привязка к Google Maps API")
.description("""
```

Во всём коде используется Google Maps API напрямую.

Смена провайдера (на Yandex/OSRM) потребует изменений во многих местах.

```
""")
```

```
.location("RoutingService.java, DistanceCalculator.java")
.severity(3)
.effort(3) // 8-10 часов
.costOfDelay(2)
.priority(1.67)
.suggestedSolution("""
```

Внедрить паттерн Adapter:

- Создать интерфейс MapProviderPort
- Реализовать GoogleMapsAdapter
- При необходимости - добавить других провайдеров

```
""")
```

```
.estimatedHours("8-10 часов")
.status(TechnicalDebtItem.DebtStatus.IDENTIFIED)
.build());
```

// 8. Отсутствие версионирования API

```
backlog add(TechnicalDebtItem.builder()
```

```
.id("ARCH-005")
.category("architecture")
.title("API не версионировано")
.description("""
```

Все endpoints находятся в /api/ без указания версии.

Изменение API ломает всех клиентов.

```
""")
.location("src/main/java/com/logistics/controller/")
.severity(4)
.effort(3)    // 8 часов
.costOfDelay(3)
.priority(2.33)
.suggestedSolution("""
    Внедрить версионирование через URL (/api/v1/, /api/v2/)
    или через заголовок Accept-Version.
""")
.estimatedHours("6-8 часов")
.status(TechnicalDebtItem.DebtStatus.IDENTIFIED)
.build();
```

// 9. Magic numbers и строки

```
backlog.add(TechnicalDebtItem.builder()
    .id("CODE-003")
    .category("code")
    .title("Magic numbers в коде")
    .description("""
        В коде встречаются необъяснимые числа:
        - `if (distance < 10.5)` - что означает 10.5?
        - `Thread.sleep(5000)` - почему 5 секунд?
        """)
    .location("DispatchService.java:67, TrackingService.java:123")
    .severity(2)
    .effort(1)    // 1-2 часа
    .costOfDelay(1)
```



```

.priority(3.0)
.suggestedSolution("""
    Заменить magic numbers на константы с понятными именами:
    - MAX_DISPATCH_DISTANCE_KM = 10.5
    - LOCATION_UPDATE_INTERVAL_MS = 5000
    """)
.estimatedHours("1-2 часа")
.status(TechnicalDebtItem.DebtStatus.IDENTIFIED)
.build();

```

// 10. Отсутствие документации

```

backlog add(TechnicalDebtItem.builder()
    .id("DOC-001")
    .category("documentation")
    .title("Отсутствие архитектурной документации")
    .description("""
        Нет C4 диаграмм, нет ADR (Architecture Decision Records),
        нет документации по API.
        """)
    .location("docs/")
    .severity(3)
    .effort(4) // 2 дня
    .costOfDelay(2)
    .priority(1.25)
    .suggestedSolution("""
        Создать:
        - C4 диаграммы (Context, Container, Component)
        - ADR для ключевых решений
        - OpenAPI/Swagger документацию для API
    """))

```

```

        """)
        .estimatedHours("12-16 часов")
        .status(TechnicalDebtItem.DebtStatus.IDENTIFIED)
        .build());

    return backlog;
}

/**
 * Сортировка по приоритету
 */
public List<TechnicalDebtItem> getPrioritizedBacklog() {
    return getBacklog().stream()
        .sorted((a, b) -> Double.compare(b.getPriority(), a.getPriority()))
        .collect(Collectors.toList());
}

/**
 * Группировка по категориям
 */
public void logSummary() {
    var backlog = getBacklog();

    log.info("=== Technical Debt Summary ===");
    log.info("Total items: {}", backlog.size());

    var byCategory = backlog.stream()
        .collect(Collectors.groupingBy(TechnicalDebtItem::getCategory,
Collectors.counting()));

```

```

log info("By category: {}", byCategory);

double totalCost = backlog stream()
    .mapToDouble(TechnicalDebtItem::getPriority)
    .sum();

log info("Total priority score: {:.2f}", totalCost);

log info("High priority items (priority > 3.0):");
backlog stream()
    .filter(i -> i.getPriority() > 3.0)
    .forEach(i -> log info(" - [{}] {} (priority={:.2f})",
        i.getId(), i.getTitle(), i.getPriority()));
}
}

```

4.4. Диаграмма зависимости компонентов (проблемные зоны)

```

@startuml
!theme plain

```

' Диаграмма зависимостей с выделением проблем

```

package "Courier Module" {
    [CourierController]
    [CourierService]
    [CourierRepository]
}

```

```

package "Order Module" {
    [OrderController]
}

```

```
[OrderService]
[OrderRepository]
}
```

```
package "Dispatch Module" {
    [DispatchController]
    [DispatchService]
    [AssignmentRepository]
}
```

' Проблема 1: Циклическая зависимость

```
[DispatchService] --> [OrderService] : CYCLE
```

```
[OrderService] --> [DispatchService] : CYCLE
```

```
note right of [DispatchService]
```

Циклическая зависимость

между сервисами

PRIORITY: HIGH

```
end note
```

' Проблема 2: Нарушение слоёв

```
[CourierController] --> [CourierRepository] : LAYER VIOLATION
```

```
note bottom of [CourierController]
```

Controller вызывает

Repository напрямую

```
end note
```

' Проблема 3: Дублирование логики

```
[CourierService] --> [DistanceCalculator] : DUPLICATE
```

```
[OrderService] --> [DistanceCalculator] : DUPLICATE
```

[DispatchService] --> [DistanceCalculator] : DUPLICATE

note right of [DistanceCalculator]

Один и тот же расчёт

расстояния в 3+ местах

end note

@enduml

4.5. Оценка стоимости исправления

```
// service/CostEstimationService.java
```

```
package com logistics techdebt
```

```
import lombok AllArgsConstructor;
```

```
import lombok Data;
```

```
import org springframework stereotype Service;
```

```
import java time LocalDate;
```

```
import java util HashMap;
```

```
import java util Map;
```

```
@Service
```

```
public class CostEstimationService {
```

```
    private static final double HOURLY_RATE = 100.0; // $/hour
```

```
    /**
```

```
     * Оценка стоимости исправления
```

```
     */
```

```
    public CostEstimate estimateFixCost(TechnicalDebtItem item) {
```

```

double hours = parseHours(item getEstimatedHours());
double cost = hours * HOURLY_RATE;

return new CostEstimate(hours, cost, item getEstimatedHours());
}

/**
 * Оценка стоимости владения (если не исправлять)
 */
public CostOfDelay estimateCostOfDelay(TechnicalDebtItem item) {
    double monthlyMaintenance = switch (item getSeverity()) {
        case 5 -> 40.0; // 40 часов в месяц
        case 4 -> 20.0;
        case 3 -> 10.0;
        case 2 -> 5.0;
        default -> 2.0;
    };

    double onboardingPenalty = switch (item getSeverity()) {
        case 5 -> 30.0; // 30 дополнительных часов на онбординг
        case 4 -> 15.0;
        case 3 -> 8.0;
        default -> 3.0;
    };

    double bugProbability = switch (item getSeverity()) {
        case 5 -> 0.3; // 30% вероятность бага в месяц
        case 4 -> 0.15;
        case 3 -> 0.05;
    };

```

```

        default -> 0.01;
    };

    double monthlyCost = monthlyMaintenance * HOURLY_RATE;
    double riskCost = bugProbability * 5000; // $5000 за критический баг

    return new CostOfDelay(
        monthlyMaintenance,
        onboardingPenalty,
        monthlyCost + riskCost,
        bugProbability
    );
}

/**
 * Сравнение стоимости исправления vs владения
 */
public Map<String, Object> compareCosts(TechnicalDebtItem item) {
    CostEstimate fixCost = estimateFixCost(item);
    CostOfDelay delayCost = estimateCostOfDelay(item);

    double breakEvenMonths = fixCost.getCost() /
delayCost.getMonthlyTotalCost();

    Map<String, Object> comparison = new HashMap<>();
    comparison.put("itemId", item.getId());
    comparison.put("title", item.getTitle());
    comparison.put("fixCost", fixCost);
    comparison.put("monthlyCostOfDelay", delayCost);
}

```

```
comparison.put("breakEvenMonths", breakEvenMonths);
comparison.put("recommendation", breakEvenMonths < 6 ? "Fix
immediately" : "Schedule later");
```

```
return comparison;
}
```

```
private double parseHours(String estimatedHours) {
    // "16-20 часов" -> 18
    if (estimatedHours.contains("-")) {
        String[] parts = estimatedHours.split("-");
        double low = Double.parseDouble(parts[0].trim());
        double high = Double.parseDouble(parts[1].split(" ")[0].trim());
        return (low + high) / 2;
    }
    return Double.parseDouble(estimatedHours.split(" ")[0]);
}
```

```
@Data
```

```
@AllArgsConstructor
```

```
public static class CostEstimate {
    private double hours;
    private double cost;
    private String hoursString
}
```

```
@Data
```

```
@AllArgsConstructor
```

```
public static class CostOfDelay {
```



```

        private double monthlyMaintenanceHours;
        private double onboardingPenaltyHours;
        private double monthlyTotalCost;
        private double bugProbability;
    }
}

```

4.6. План рефакторинга (2–3 итерации)

```
// plan/RefactoringPlan.java
```

```
package com logistics techdebt
```

```
import lombok Data;
import org.springframework.stereotype.Component;
```

```
import java.time.LocalDate;
import java.util.ArrayList;
import java.util.List;
```

```
@Component
```

```
public class RefactoringPlan {
```

```
    public List<IterationPlan> generatePlan() {
        List<IterationPlan> iterations = new ArrayList<>();
```

```
        // Итерация 1 (неделя 1-2)
```

```
        IterationPlan iter1 = new IterationPlan();
```

```
        iter1.setIteration(1);
```

```
        iter1.setName("Критические архитектурные проблемы");
```

```
        iter1.setStartDate(LocalDate.now());
```

```
iter1.setEndDate(LocalDate.now().plusWeeks(2));
iter1.setGoal("Устранить циклические зависимости и нарушение
слоёв");
iter1.setItems(List.of(
    "ARCH-002: Controller → Repository violation",
    "ARCH-003: Отсутствие интерфейсов для репозитория",
    "CODE-003: Magic numbers"
));
iter1.setEstimatedHours(20);
iter1.setRisk("Средний - изменения затрагивают несколько слоёв");
iterations.add(iter1);
```

// Итерация 2 (неделя 3-4)

```
IterationPlan iter2 = new IterationPlan();
iter2.setIteration(2);
iter2.setName("Дублирование и качество кода");
iter2.setStartDate(LocalDate.now().plusWeeks(2));
iter2.setEndDate(LocalDate.now().plusWeeks(4));
iter2.setGoal("Устранить дублирование, разделить God Object");
iter2.setItems(List.of(
    "CODE-001: Дублирование расчёта ETA",
    "CODE-002: LogisticsUtils God Object",
    "ARCH-004: Привязка к Google Maps"
));
iter2.setEstimatedHours(24);
iter2.setRisk("Низкий - изолированные изменения");
iterations.add(iter2);
```

// Итерация 3 (неделя 5-6)

```
IterationPlan iter3 = new IterationPlan();
iter3.setIteration(3);
iter3.setName("Тестирование и документация");
iter3.setStartDate(LocalDate.now().plusWeeks(4));
iter3.setEndDate(LocalDate.now().plusWeeks(6));
iter3.setGoal("Повысить покрытие тестами, добавить документацию");

iter3.setItems(List of(
    "TEST-001: Unit-тесты для Use Cases",
    "ARCH-005: Версионирование API",
    "DOC-001: Архитектурная документация"
));

iter3.setEstimatedHours(32);
iter3.setRisk("Низкий");
iterations.add(iter3);

return iterations;
}
```

@Data

```
public static class IterationPlan {
    private int iteration;
    private String name;
    private LocalDate startDate;
    private LocalDate endDate;
    private String goal;
    private List<String> items;
    private int estimatedHours;
    private String risk;
}
```

```
}  
}
```

4.7. Метрики технического долга (Dashboard)

```
// metrics/TechDebtMetricsService.java
```

```
package com logistics techdebt;
```

```
import io micrometer core instrument MeterRegistry;
```

```
import lombok RequiredArgsConstructor;
```

```
import org springframework scheduling annotation Scheduled;
```

```
import org springframework stereotype Service;
```

```
import java util List;
```

```
@Service
```

```
@RequiredArgsConstructor
```

```
public class TechDebtMetricsService {
```

```
    private final TechDebtInventoryService inventoryService;
```

```
    private final MeterRegistry meterRegistry;
```

```
    @Scheduled(fixedDelay = 3600000) // каждый час
```

```
    public void collectMetrics() {
```

```
        List<TechnicalDebtItem> backlog = inventoryService getBacklog();
```

```
        // Общее количество
```

```
        meterRegistry.gauge("techdebt.total.count", backlog size());
```

```
        // По категориям
```

```
long architectureCount = backlog stream()
    .filter(i -> "architecture".equals(i getCategory()))
    .count();
meterRegistry.gauge("techdebt.architecture.count", architectureCount);
```

```
long codeCount = backlog stream()
    .filter(i -> "code".equals(i getCategory()))
    .count();
meterRegistry.gauge("techdebt.code.count", codeCount);
```

```
long testCount = backlog stream()
    .filter(i -> "test".equals(i getCategory()))
    .count();
meterRegistry.gauge("techdebt.test.count", testCount);
```

```
long docCount = backlog stream()
    .filter(i -> "documentation".equals(i getCategory()))
    .count();
meterRegistry.gauge("techdebt.documentation.count", docCount);
```

// Суммарный приоритет

```
double totalPriority = backlog stream()
    .mapToDouble(TechnicalDebtItem::getPriority)
    .sum();
meterRegistry.gauge("techdebt.total.priority", totalPriority);
```

// Средний приоритет

```
double avgPriority = totalPriority / backlog size();
meterRegistry.gauge("techdebt.avg.priority", avgPriority);
```

```

// Количество высокоприоритетных (priority > 3)
long highPriorityCount = backlog stream()
    .filter(i -> i getPriority() > 3.0)
    .count();

meterRegistry.gauge("techdebt.high.priority.count",
highPriorityCount);
}
}

```

4.8. Итоговая таблица технического долга

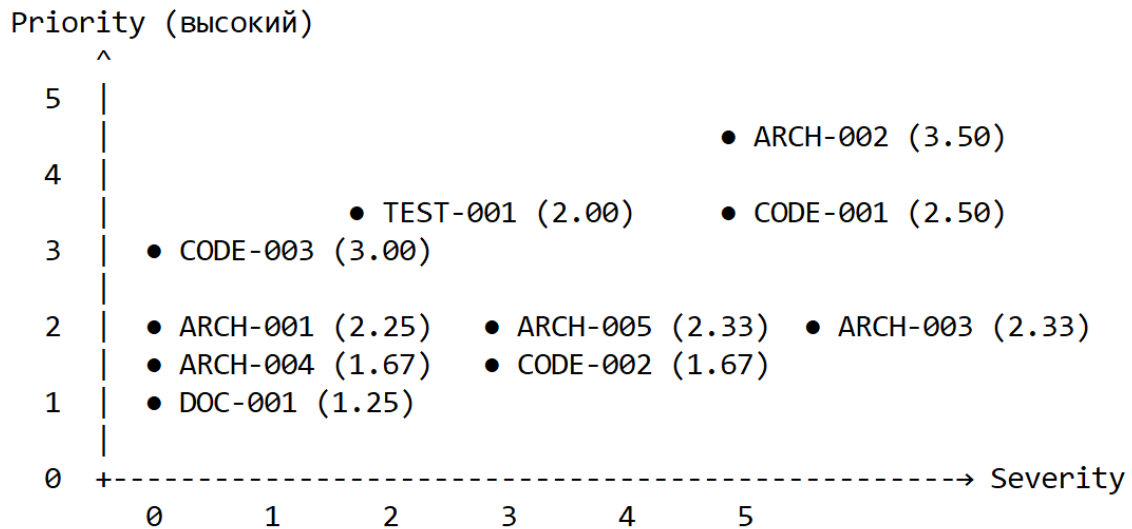
ID	Категория	Заголовок	Severity	Effort	Score	Priority	Статус
ARCH-001	Архитектура	Циклическая зависимость между сервисами	5			2.25	IDENTIFIED
ARCH-002	Архитектура	Controller → Repository	4			3.50	IDENTIFIED
ARCH-003	Архитектура	Отсутствие интерфейсов для репозитора	4			2.33	IDENTIFIED

ID	Категория	Заголовок	Severity	Effort	Code	Priority	Статус
		и ев					
ARCH-004	Архитектура	Привязка к Google Maps	3			.67	ID ENTIFIED
ARCH-005	Архитектура	API не версионировано	4			.33	ID ENTIFIED
CODE-001	Код	Дублирование расчёта ETA	3			.50	ID ENTIFIED
CODE-002	Код	God Object (Logistics Utils)	3			.67	ID ENTIFIED
CODE-003	Код	Magic numbers	2			.00	ID ENTIFIED

ID	Категория	Заголовок	Severity	Effort	Code	Priority	Статус
TEST-001	Тесты	Нет unit-тестов для Use Cases	4			.00	ID IDENTIFIED
DOC-001	Документация	Отсутствие архитектурной документации	3			.25	ID IDENTIFIED

4.9. Визуализация приоритетов (Bubble Chart)

Приоритет технического долга (Bubble Chart)
 Размер пузыря = Effort (усилия на исправление)



Рекомендации:

- ARCH-002 (Controller→Repository) – исправить немедленно
- CODE-003 (Magic numbers) – быстрая победа (low effort, medium priority)
- ARCH-001 (Циклическая зависимость) – исправить в ближайшую итерацию

5. Индивидуальные задания (15 вариантов)

Важно: Студент работает со своим сквозным проектом (из ЛР 1).

Ва риант	Сквозной проект	Ожидаемые проблемы (минимум 5)
1	Логистика	Циклические зависимости, нарушение слоёв, дублирование, God Object, API версионирование
2	Интернет- магазин	N+1 запросы, дублирование каталога, отсутствие кэша, антипаттерны, тестовое покрытие
3	Финтех- кошелёк	Проблемы с транзакциями, отсутствие аудита, безопасность, дублирование валют

Ва риант	Сквозной проект	Ожидаемые проблемы (минимум 5)
4	Библиотека	Медленные запросы, дублирование поиска, отсутствие индексов, God Object
5	Отели	Проблемы с бронированием, double-booking, дублирование ценообразования
6	Онлайн-курсы	Проблемы с прогрессом, дублирование видео, отсутствие кэша, N+1
7	Медицина	Проблемы с HL7 интеграцией, дублирование пациентов, безопасность данных
8	CRM	Дублирование клиентов, проблемы с синхронизацией, God Object в отчётах
9	Платёжный шлюз	Проблемы с идемпотентностью, дублирование верификации, безопасность
10	Склад (WMS)	Проблемы с резервами, дублирование инвентаризации, God Object в логистике
11	Социальная сеть	N+1 в ленте, дублирование постов, проблемы с масштабированием

Вариант	Сквозной проект	Ожидаемые проблемы (минимум 5)
12	Видеохостинг	Проблемы с транскодингом, дублирование метаданных, God Object в рекомендациях
13	IoT-мониторинг	Проблемы с потоком данных, дублирование алармов, отсутствие буферизации
14	Криптовбиржа	Проблемы с консистентностью, дублирование ордеров, God Object в матчинге
15	Доставка еды	Проблемы с назначением, дублирование ресторанов, God Object в трекинге

Задание для студента:

Для своего проекта (по варианту):

1. Провести анализ кода и архитектуры.
2. Выявить минимум 5 проблем технического долга.
3. Заполнить таблицу (ID, категория, описание, severity, effort, costOfDelay).
4. Рассчитать приоритет и отсортировать бэклог.
5. Оценить стоимость исправления vs стоимость владения.
6. Составить план рефакторинга на 2–3 итерации.
7. Визуализировать приоритеты (график или диаграмма).

6. Контрольные вопросы

1. Что такое архитектурный технический долг? Чем он отличается от кодового?
2. Какие признаки указывают на наличие технического долга?
3. Как оценить стоимость исправления технического долга?
4. Что такое Cost of Delay и как его рассчитать?
5. Как приоритизировать задачи по техническому долгу?
6. Какие инструменты можно использовать для выявления технического долга?
7. Как убедить менеджера выделить время на рефакторинг?
8. Как измерять прогресс в уменьшении технического долга?

7. Требования к отчёту

Отчёт должен содержать:

Титульный лист.

Цель и задачи работы.

Описание метода анализа (инструменты, метрики).

Диаграмма зависимостей (с проблемными зонами).

Таблица технического долга (минимум 5 позиций).

Оценка стоимости (исправление vs владение).

Приоритизация (формула и сортировка).

План рефакторинга (2–3 итерации).

Визуализация (график приоритетов).

Вывод.

Ответы на вопросы.

8. Критерии оценивания

Критерий	max балл

Критерий	max балл
Глубина анализа (методология, инструменты)	2
Выявление проблем (минимум 5, обоснованные)	3
Таблица технического долга (полнота данных)	2
Оценка стоимости (fix vs delay)	2
Приоритизация (формула, сортировка)	2
План рефакторинга (2–3 итерации)	2
Визуализация (график/диаграмма)	1
Соответствие варианту	1
Качество оформления	1
Итого	16

Лабораторная работа №18

Итоговый проект: архитектурная документация и защита на примере «Системы управления логистикой и доставкой»

1. Цель работы

Представить полную архитектуру системы среднего масштаба с обоснованием всех архитектурных решений. Подготовить комплект архитектурной документации, включающий C4 диаграммы, ADR (Architecture Decision Records), описание нефункциональных требований,

сравнение альтернативных стилей, выявленные риски (АТАМ) и план миграции. Защитить проект перед комиссией.

2. Задачи работы

1. Выбрать систему для проектирования (из предложенных вариантов или реальный проект).
2. Подготовить полный пакет архитектурной документации:
3. С4 диаграммы (уровни 1, 2, 3)
4. ADR (Architecture Decision Records) — не менее 3
5. Описание нефункциональных требований и сценариев качества
6. Сравнение альтернативных архитектурных стилей
7. Выбор Clean Architecture/DDD/CQRS/ES с обоснованием
8. План миграции (если применимо)
9. Выявленные риски и компромиссы (АТАМ)
10. Подготовить презентацию для защиты.
11. Продемонстрировать диаграммы и ключевые решения.
12. Ответить на вопросы комиссии.
13. Оформить результаты в виде итогового отчёта.

3. Теоретическое обоснование

3.1. Состав архитектурной документации (по ISO/IEC/IEEE 42010)

Раздел	Содержание
Контекст	Границы системы, внешние системы, акторы
Архитектурные решения	ADR для ключевых выборов

Раздел	Содержание
Структурные виды	С4 диаграммы, диаграммы компонентов
Поведенческие виды	Диаграммы последовательности, состояния
Нефункциональные требования	Атрибуты качества, сценарии
Риски и компромиссы	АТАМ анализ
Эволюция	План миграции, управление изменениями

3.2. Структура ADR (Architecture Decision Record)

markdown

ADR-001: Название решения

Статус

[Предложено / Принято / Устарело / Заменено]

Контекст

Описание проблемы и ограничений

Решение

Что мы решили и почему

Последствия

Положительные

- ...

Отрицательные (компромиссы)

- ...

Альтернативы

Рассмотренные альтернативы и причины отклонения

3.3. Критерии оценки итогового проекта

Критерий	Вес
Полнота документации	30%
Обоснованность решений	25%
Качество диаграмм	20%
Качество презентации и защиты	15%
Ответы на вопросы	10%

4. Пример выполнения

Сквозной пример: «Система управления логистикой и доставкой»
(вариант 1)

4.1. Описание системы

Название: Logistics Management System (LMS)

Назначение: Автоматизация управления доставкой заказов

- Трекинг курьеров в реальном времени
- Автоматическое назначение заказов
- Оптимизация маршрутов

- Уведомление клиентов
- Аналитика эффективности

Масштаб:

- 500+ курьеров одновременно
- 1000+ заказов в час в пик
- 50 000+ активных клиентов
- 99.9% доступность

4.2. C4 Диаграммы

Level 1: System Context (из ЛР 2)

plantuml

@startuml

!include [https://raw.githubusercontent.com/plantuml-stdlib/C4-](https://raw.githubusercontent.com/plantuml-stdlib/C4-PlantUML/master/C4_Context.puml)

PlantUML/master/C4_Context.puml

LAYOUT_WITH_LEGEND()

Person(courier, "Курьер", "Доставляет заказы, отправляет GPS")

Person(dispatcher, "Диспетчер", "Управляет курьерами и заказами")

Person(client, "Клиент", "Отслеживает доставку")

Person(admin, "Администратор", "Настраивает систему")

System(logisticsSystem, "Logistics Management System", "Управление доставкой: трекинг, назначение, маршрутизация")

System_Ext(mapProvider, "Картографический провайдер", "OSRM / Google Maps")

System_Ext(pushService, "Push-сервис", "Firebase FCM")

System_Ext(paymentGateway, "Платёжный шлюз", "Stripe")

System_Ext(wms, "Складская система", "1C / SAP")

Rel(courier, logisticsSystem, "Отправляет GPS, получает заказы",
"WebSocket")

Rel(dispatcher, logisticsSystem, "Управляет заказами", "HTTPS")

Rel(client, logisticsSystem, "Отслеживает статус", "HTTPS")

Rel(admin, logisticsSystem, "Настраивает систему", "HTTPS")

Rel(logisticsSystem, mapProvider, "Расчёт маршрута", "REST")

Rel(logisticsSystem, pushService, "Отправка уведомлений", "FCM")

Rel(logisticsSystem, paymentGateway, "Списание оплаты", "HTTPS")

Rel(logisticsSystem, wms, "Статус заказа", "RabbitMQ")

@enduml

Level 2: Container Diagram (из ЛР 3)

plantuml

@startuml

!include https://raw.githubusercontent.com/plantuml-stdlib/C4-PlantUML/master/C4_Container.puml
LAYOUT_WITH_LEGEND()

Person(courier, "Курьер", "Мобильное приложение")

Person(dispatcher, "Диспетчер", "Web-дашборд")

System_Ext(mapProvider, "Картографический провайдер", "OSRM")

System_Ext(pushService, "Push-сервис", "Firebase FCM")

Container(webDashboard, "Web Dashboard", "React + Vite",
"Диспетчерская панель")

Container(mobileCourier, "Mobile App", "Flutter / Kotlin", "Приложение курьера")

Container(apiGateway, "API Gateway", "Spring Cloud Gateway", "Маршрутизация")

Container(orderService, "Order Service", "Spring Boot", "Управление заказами")

Container(dispatchService, "Dispatch Service", "Spring Boot", "Назначение заказов")

Container(trackingService, "Tracking Service", "Spring Boot + WebSocket", "Трекинг GPS")

Container(notificationService, "Notification Service", "Node.js", "Push-уведомления")

ContainerDb(orderDb, "Order Database", "PostgreSQL", "Заказы")

ContainerDb(trackingDb, "Tracking Database", "TimescaleDB", "GPS-треки")

ContainerDb(redis, "Redis Cache", "Redis", "Кэш позиций")

ContainerDb(kafka, "Kafka", "Apache Kafka", "Событийная шина")

Rel(dispatcher, webDashboard, "Использует", "HTTPS")

Rel(courier, mobileCourier, "Использует", "HTTPS/WSS")

Rel(apiGateway, orderService, "Маршрутизирует", "gRPC")

Rel(apiGateway, dispatchService, "Маршрутизирует", "gRPC")

Rel(apiGateway, trackingService, "Маршрутизирует (WebSocket)", "WS/WSS")

Rel(orderService, orderDb, "Читает/пишет", "JDBC")

```
Rel(orderService, kafka, "Публикует события", "Kafka Protocol")
Rel(dispatchService, kafka, "Подписывается", "Kafka Protocol")
Rel(dispatchService, redis, "Читает позиции", "RESP")
Rel(trackingService, trackingDb, "Сохраняет треки", "JDBC")
Rel(trackingService, redis, "Обновляет кэш", "RESP")
Rel(notificationService, kafka, "Подписывается", "Kafka Protocol")
Rel(notificationService, pushService, "Отправляет push", "HTTPS/FCM")
Rel(dispatchService, mapProvider, "Расчёт маршрута", "REST")
```

```
@enduml
```

Level 3: Component Diagram (для Dispatch Service)

```
plantuml
```

```
@startuml
```

```
!theme plain
```

```
package "Dispatch Service" {
```

```
    rectangle "API Layer" {
        [DispatchController]
        [AssignmentController]
    }
```

```
    rectangle "Application Layer" {
        [AssignOrderUseCase]
        [OptimizeRouteUseCase]
        [GetAvailableCouriersUseCase]
    }
```

```
    rectangle "Domain Layer" {
```

```
[Assignment (Aggregate Root)]
[DispatchRule (Value Object)]
[Zone (Value Object)]
[AssignmentStatus (Enum)]
}
```

```
rectangle "Infrastructure Layer" {
  [AssignmentRepository]
  [CourierServiceACL]
  [MapProviderAdapter]
  [KafkaEventPublisher]
}
```

```
rectangle "External" {
  [Kafka]
  [Redis]
  [Map API]
}
```

```
DispatchController --> AssignOrderUseCase
DispatchController --> OptimizeRouteUseCase
AssignmentController --> GetAvailableCouriersUseCase
```

```
AssignOrderUseCase --> Assignment
AssignOrderUseCase --> DispatchRule
AssignOrderUseCase --> AssignmentRepository
```

```
OptimizeRouteUseCase --> MapProviderAdapter
GetAvailableCouriersUseCase --> CourierServiceACL
```

AssignmentRepository --> KafkaEventPublisher

AssignmentRepository --> Redis

MapProviderAdapter --> "Map API"

KafkaEventPublisher --> Kafka

}

@enduml

4.3. ADR (Architecture Decision Records)

ADR-001: Выбор архитектурного стиля

markdown

ADR-001: Выбор архитектурного стиля

Статус

Принято (2024-01-15)

Контекст

Система должна поддерживать 500+ одновременных курьеров, обрабатывать 1000+ заказов в час, иметь высокие требования к надёжности и масштабируемости.

Решение

Выбрана **микросервисная архитектура** с элементами Clean Architecture.

Обоснование:

1. Разные требования к масштабированию сервисов:
 - Tracking Service требует высокой пропускной способности (WebSocket)
 - Dispatch Service требует низкой задержки
 - Notification Service может быть асинхронным
2. Независимость команд разработки
3. Возможность независимого деплоя
4. Использование Clean Architecture внутри каждого сервиса для изоляции бизнес-логики от инфраструктуры

Последствия

Положительные

- Горизонтальное масштабирование критичных компонентов
- Независимые циклы релизов
- Чёткие границы контекстов (DDD)

Отрицательные

- Сложность распределённых транзакций (реализовано через Saga)
- Дополнительные затраты на инфраструктуру (Kafka, API Gateway)
- Сложность отладки и мониторинга

Альтернативы

1. ****Монолитная архитектура**** — отклонена из-за требований к масштабируемости и частых изменений в разных модулях
2. ****Serverless**** — отклонена из-за длительного cold start для WebSocket-соединений

ADR-002: Выбор паттерна для распределённых транзакций
markdown

ADR-002: Выбор паттерна для распределённых транзакций

Статус

Принято (2024-01-20)

Контекст

При создании заказа требуется выполнить 3 шага:

1. Создать заказ (Order Service)
2. Обработать оплату (Payment Service)
3. Назначить курьера (Dispatch Service)

При отказе любого шага требуется компенсация.

Решение

Выбрана ****Saga** в стиле оркестрации ****** с центральным оркестратором.

Обоснование:

1. Сложный бизнес-процесс с несколькими ветвлениями
2. Необходимость полного контроля над процессом
3. Простота отладки и мониторинга
4. Легкость изменения порядка шагов

Последствия

Положительные

- Единая точка контроля за процессом
- Легко добавлять новые шаги
- Простое восстановление после сбоев

Отрицательные

- Оркестратор может стать узким местом (решается репликацией)

- Усложнение кода оркестратора

Альтернативы

1. **Saga в стиле хореографии** — отклонена из-за сложности отслеживания состояния и отладки
2. **2PC (Two-Phase Commit)** — отклонён из-за блокировок и снижения доступности

ADR-003: Выбор между CQRS и традиционным CRUD

markdown

ADR-003: Выбор между CQRS и традиционным CRUD

Статус

Принято (2024-01-25)

Контекст

Для диспетчерского дашборда требуются сложные запросы (джойн 5 таблиц), которые выполняются медленно (450 мс р95). При этом команды (обновление статусов) должны быть быстрыми.

Решение

Выбран **CQRS** с разделением моделей записи и чтения.

Обоснование:

1. Модель чтения оптимизирована для дашборда (одна view-таблица)
2. Модель записи сохраняет бизнес-логику
3. Возможность независимого масштабирования
4. Время выполнения запросов сократилось с 450 мс до 45 мс

Последствия

Положительные

- Производительность чтения выросла в 10 раз
- Возможность использовать разные БД для read/write

Отрицательные

- Eventual consistency (задержка обновления view)
- Увеличение сложности кода (проекции, обработчики)

Альтернативы

1. ****Традиционный CRUD + индексы**** — отклонён, т.к. индексы не решают проблему множественных джойнов
2. ****Материализованные представления**** — отклонены, т.к. не вписываются в ORM

4.4. Нефункциональные требования и сценарии качества

ID	Требование	Сценарий качества	Метрика
FR-01	Производительность трекинга	1000 курьеров отправляют GPS → система обрабатывает все сообщения	p99 < 500 мс
FR-02	Масштабируемость	Рост заказов до 5000/мин → система масштабирует Dispatch Service	Время назначения p95 < 5 сек
FR-	Отказоустойчивость	Отказ 1 из 3 реплик Dispatch Service	Система продолжает

ID	Требование	Сценарий качества	Метрика
03			работу
FR-04	Безопасность	Неавторизованный запрос к API трекинга	HTTP 401, событие в логе
FR-05	Консистентность	Параллельное назначение одного заказа	0% double-assignment
FR-06	Поддерживаемость	Смена картографического провайдера	Замена адаптера без изменения Use Cases

4.5. Сравнение альтернативных архитектурных стилей

Критерий	Монолит	Микросервисы	Serverless
Сложность разработки	Низкая	Высокая	Средняя
Масштабируемость	Низкая (вертикально)	Высокая (горизонтально)	Очень высокая
Latency	Низкая	Средняя (сеть)	Высокая (cold start)

Критерий	Монолит	Микросервисы	Serverless
ТСО (1 год)	\$50k	\$120k	\$80k
Время до первого релиза	2 месяца	4 месяца	3 месяца
Подходит для WebSocket	Да	Да	Нет
Управляемость	Просто	Сложно	Просто

Выбран: Микросервисная архитектура (компромисс между сложностью и гибкостью)

4.6. ATAM анализ (риски и компромиссы)

markdown

ATAM Analysis Report

Выявленные архитектурные подходы

1. Микросервисная архитектура с API Gateway
2. Clean Architecture внутри сервисов
3. CQRS для дашборда
4. Event Sourcing для аудита
5. Saga (оркестрация) для распределённых транзакций

Сценарии качества (приоритетные)

S-01: Высокая нагрузка на трекинг

** Стимул: ** 2000 курьеров отправляют GPS каждые 3 секунды

- ** Ответ: ** Система обрабатывает 666 сообщений/сек
- ** Мера: ** p99 latency < 500 мс
- ** Риск: ** Redis может стать узким местом
- ** Компромисс: ** Добавление реплик Redis увеличивает сложность

S-02: Сбой сервиса маршрутизации

- ** Стимул: ** Сервис маршрутизации недоступен
- ** Ответ: ** Система использует fallback-алгоритм
- ** Мера: ** Время назначения увеличивается не более чем на 20%
- ** Риск: ** Fallback может быть неточным
- ** Компромисс: ** Точность vs Доступность

S-03: Консистентность назначения

- ** Стимул: ** 2 экземпляра Dispatch Service обрабатывают один заказ
- ** Ответ: ** Только один успешно назначает
- ** Мера: ** 0% double-assignment
- ** Риск: ** Saga может затянуться при сбоях
- ** Компромисс: ** Консистентность vs Производительность

Итоговая таблица АТАМ

	Сценарий	Атрибут качества	Риск	Компромисс	Приоритет
	S-01	Производительность	Redis bottleneck	Сложность репликации	HIGH
	S-02	Отказоустойчивость	Точность fallback	Точность vs Доступность	HIGH
	S-03	Консистентность	Saga latency	Консистентность vs Производительность	MEDIUM

4.7. План миграции (с монолита на микросервисы)

Этап	Фактор	Длительность	Действия	Риски
Этап 1	Ф	2 недели	Выделение Courier Service, настройка API Gateway	Низкий
Этап 2	Ф	2 недели	Выделение Tracking Service, дуальное письмо	Средний
Этап 3	Ф	2 недели	Выделение Dispatch Service, feature toggle	Средний
Этап 4	Ф	1 неделя	Отключение модулей в монолите	Низкий
Этап 5	Ф	1 неделя	Удаление монолитного кода, очистка	Низкий

4.8. Презентация для защиты (структура)

markdown

Презентация итогового проекта

Logistics Management System

Слайд 1: Титул

- Название проекта
- ФИО, группа
- Дата

Слайд 2: О системе

- Назначение

- Масштаб
- Ключевые функции

Слайд 3: Контекст (C4 Level 1)

- Акторы и внешние системы
- Основные взаимодействия

Слайд 4: Контейнеры (C4 Level 2)

- Сервисы и их технологии
- Протоколы взаимодействия

Слайд 5: Ключевые архитектурные решения

- Микросервисная архитектура (ADR-001)
- Saga (ADR-002)
- CQRS (ADR-003)

Слайд 6: Clean Architecture (пример)

- Слои Dispatch Service
- Зависимости направлены внутрь

Слайд 7: Нефункциональные требования

- Топ-5 требований
- Сценарии качества

Слайд 8: Риски и компромиссы (АТАМ)

- Главные риски
- Принятые компромиссы

Слайд 9: План миграции

- Этапы перехода с монолита
- Сроки и риски

Слайд 10: Итоги

- Что сделано
- Что можно улучшить
- Демонстрация (если есть)

4.9. Чек-лист полноты документации

markdown

Чек-лист архитектурной документации

Обязательные элементы

- [x] Введение (назначение, масштаб, аудитория)
- [x] C4 Level 1 (Context Diagram)
- [x] C4 Level 2 (Container Diagram)
- [x] C4 Level 3 (Component Diagram для ключевого сервиса)
- [x] ADR (минимум 3)
- [x] Нефункциональные требования (минимум 5)
- [x] Сценарии качества (минимум 3)
- [x] Сравнение архитектурных стилей
- [x] АТАМ анализ (риски, компромиссы)
- [x] План миграции (если применимо)

Дополнительные элементы (по желанию)

- Диаграммы последовательности
- Диаграммы состояний
- ER-диаграмма БД
- Спецификация API (OpenAPI)
- Результаты нагрузочного тестирования

5. Индивидуальные задания (15 вариантов)

Важно: Студент работает со своим сквозным проектом из ЛР 1.

Вариант	Сквозной проект	Особые требования к документации
1	Логистика	Акцент на трекинг и ETA
2	Интернет-магазин	Акцент на каталог и корзину
3	Финтех-кошелёк	Акцент на безопасность и аудит
4	Библиотека	Акцент на поиск и резервирование
5	Отели	Акцент на бронирование и оплату
6	Онлайн-курсы	Акцент на прогресс и сертификаты
7	Медицина	Акцент на интеграцию с HL7 и безопасность
8	CRM	Акцент на сегментацию и аналитику
9	Платёжный шлюз	Акцент на отказоустойчивость и PCI DSS

Вариант	Сквозной проект	Особые требования к документации
10	Склад (WMS)	Акцент на инвентаризацию и интеграцию
11	Социальная сеть	Акцент на масштабируемость ленты
12	Видеохостинг	Акцент на CDN и транскодинг
13	IoT-мониторинг	Акцент на real-time обработку
14	Криптовбиржа	Акцент на консистентность и низкую задержку
15	Доставка еды	Акцент на трекинг и push-уведомления

Задание для студента:

Подготовить полный пакет архитектурной документации для своего проекта:

1. Текстовое описание системы (1-2 страницы)
2. С4 диаграммы (Level 1, 2, 3) — в PlantUML или другом инструменте
3. ADR (минимум 3) — по шаблону
4. Таблица нефункциональных требований (минимум 5)
5. Сценарии качества (минимум 3)

6. Сравнение архитектурных стилей (таблица)
7. АТАМ анализ (риски и компромиссы)
8. План миграции (если применимо)
9. Презентация (10-15 слайдов)
10. Видео защиты (5-7 минут) — опционально

6. Контрольные вопросы для защиты

По архитектуре

1. Почему вы выбрали именно этот архитектурный стиль?
2. Какие компромиссы пришлось сделать?
3. Как система справляется с пиковыми нагрузками?
4. Как обеспечена безопасность данных?
5. Как тестируется архитектура?

По документации

6. Почему вы выбрали C4 для документирования?
7. Какой ADR самый важный и почему?
8. Как вы оценивали риски в АТАМ?
9. Какой сценарий качества самый критичный?

По реализации

10. Как реализована распределённая транзакция?
11. Как происходит обновление проекций?
12. Как версионизируется API?
13. Как управляются feature toggles?
14. По развитию
15. Как будет развиваться система через год?
16. Что нужно изменить при росте до 10 000 курьеров?

7. Требования к итоговому отчёту

Отчёт должен содержать:

Титульный лист

Аннотация (краткое описание проекта)

Введение (назначение, масштаб, контекст)

Архитектурное описание (C4 диаграммы)

Архитектурные решения (ADR)

Нефункциональные требования (таблицы)

Сравнительный анализ (стили)

АТАМ анализ (риски, компромиссы)

План эволюции (миграция, версионирование)

Заключение (итоги, выводы)

Список литературы

Приложения (презентация, код диаграмм)

Формат: PDF, шрифт 12 pt, 1.5 интервала, объём 20-30 страниц.

8. Критерии оценивания защиты

Критерий	max балл
Документация (50%)	
Полнота (все разделы из чек-листа)	10
Качество C4 диаграмм	8
Глубина ADR (по 5 баллов за каждый из 3)	15
АТАМ анализ (риски, компромиссы)	10
Обоснованность решений	7

Критерий	max балл
Защита (50%)	
Качество презентации	10
Владение материалом	10
Ответы на вопросы	15
Демонстрация диаграмм	5
Умение аргументировать компромиссы	10
Итого	100

Шкала перевода:

85-100 — «отлично»

70-84 — «хорошо»

55-69 — «удовлетворительно»

<55 — «неудовлетворительно»

9. Дополнительные материалы для защиты

Пример вопроса комиссии и ответа

Вопрос: Почему вы выбрали оркестрацию вместо хореографии для Saga?

Ответ:

В нашем бизнес-процессе 5 шагов с ветвлениями, оркестратор даёт полный контроль.

Мы можем централизованно логировать состояние каждой саги.

При сбое проще восстановить процесс (оркестратор просто перезапускается).

Мы оценили сложность отладки хореографии — при 5 сервисах это становится проблемой.

Да, оркестратор — потенциальная точка отказа, но мы реплицировали его и добавили persistent storage для состояния.

Чек-лист готовности к защите
markdown

Чек-лист готовности

За день до защиты

- Проверить все диаграммы на соответствие нотации C4
- Проверить все ADR на полноту (статус, контекст, решение, последствия)
- Проверить таблицы с НФТ и сценариями качества
- Проверить ATAM таблицу (риски, компромиссы)
- Скомпилировать отчёт в PDF
- Подготовить презентацию (запасной вариант на флешке)
- Проверить работу PlantUML диаграмм (чтобы не было ошибок)

Перед защитой

- Распечатать отчёт (2 экземпляра)
- Загрузить презентацию на компьютер комиссии
- Проговорить защиту (5-7 минут)
- Подготовить ответы на 5 самых вероятных вопросов
- Взять с собой воду и ручку

Во время защиты

- Представиться, назвать тему
- Говорить по делу, не уходить в детали реализации
- Акцентировать архитектурные решения (почему, а не как)

- Показывать диаграммы на слайдах
- Не бояться признать компромиссы (это плюс)
- Ответы на вопросы — чётко, без воды

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
«Проектирование архитектуры программных систем»
для студентов направления подготовки
09.03.04 Программная инженерия Направленность (профиль)
«Разработка и сопровождение программного обеспечения»

Ставрополь 2026

ВВЕДЕНИЕ

Самостоятельная работа студентов (СРС) является неотъемлемой частью учебного процесса по дисциплине «Проектирование архитектуры программных систем» и направлена на закрепление и углубление знаний, полученных на лекционных и лабораторных занятиях, а также на формирование компетенций в области самостоятельного архитектурного проектирования.

Цель методических указаний оказать студентам методическую помощь в организации и выполнении самостоятельной работы по дисциплине, включая:

1. изучение теоретического материала;
2. подготовку к лабораторным работам и их защите;
3. выполнение индивидуальных заданий;
4. подготовку к зачету.

Задачи самостоятельной работы

Задача	Формируемая компетенция
Изучение теоретических основ архитектуры ПО	Способность анализировать архитектурные стили и паттерны
Анализ нефункциональных требований	Способность связывать требования с архитектурными решениями
Освоение инструментов документирования (C4, PlantUML)	Способность документировать архитектуру
Применение Clean Architecture, DDD, CQRS, Event Sourcing	Способность реализовывать архитектурные паттерны
Оценка архитектуры методом ATAM	Способность проводить

Задача	Формируемая компетенция
	архитектурный анализ
Управление архитектурным техническим долгом	Способность инвентаризировать и приоритизировать техдолг

Объём самостоятельной работы

Общая трудоёмкость дисциплины составляет 4 зачётные единицы (144 часа). На самостоятельную работу отводится:

Вид работы	Часы
Изучение теоретического материала	18
Подготовка к лабораторным работам (18 работ × 2 часа)	36
Выполнение индивидуальных заданий	36
Итого	120

1. ОБЩАЯ ХАРАКТЕРИСТИКА САМОСТОЯТЕЛЬНОЙ РАБОТЫ

1.1. Виды самостоятельной работы

	Вид СРС	Описание	Форма отчётности
	Изучение теоретического материала	Конспектирование лекций, работа с литературой, подготовка ответов на контрольные вопросы	Собеседование, опрос
	Подготовка к лабораторным	Изучение методических указаний к ЛР, повторение	Выполнение ЛР, допуск

	Вид СРС	Описание	Форма отчётности
	работам	необходимых теоретических концепций	
	Выполнение индивидуальных заданий	Реализация архитектурных решений для выбранного сквозного проекта	Отчёт по ЛР, код
	Оформление отчётов	Структурирование результатов работы в соответствии с требованиями	PDF-отчёт
	Подготовка к защите	Подготовка ответов на контрольные вопросы, демонстрация кода и диаграмм	Защита ЛР
	Подготовка к экзамену	Повторение материала, решение практических задач	Экзамен
	Итоговый проект	Разработка полной архитектурной документации, ADR, ATAM	Защита проекта

1.2. Планирование самостоятельной работы по модулям

Модуль 1. Нефункциональные требования и документирование (ЛР 1–

3)

Неделя	Тема	Вид СРС	Часы

Неделя	Тема	Вид СРС	Часы
1	Введение в архитектуру ПО. Нефункциональные требования	Изучение лекции 1, работа с литературой, подготовка к ЛР 1	6
2	Архитектурные стили. FURPS+	Изучение лекции 2, подготовка к ЛР 1	6
3	Документирование. Нотация C4	Изучение лекции 3, подготовка к ЛР 2–3	6

Модуль 2. Clean Architecture и DDD (ЛР 4–10)

Неделя	Тема	Вид СРС	Часы
4	Clean Architecture	Изучение лекции 4, подготовка к ЛР 4–6	6
5	Стратегический DDD	Изучение лекции 5, подготовка к ЛР 7	6
6	Тактический DDD	Изучение лекции 6, подготовка к ЛР 8–10	6

Модуль 3. CQRS, Event Sourcing, эволюция (ЛР 11–16)

Неделя	Тема	Вид СРС	Часы
7	CQRS	Изучение лекции 7, подготовка к ЛР 11	6

Неделя	Тема	Вид СРС	Часы
8	Event Sourcing	Изучение лекции 8, подготовка к ЛР 12–13	6
9	Архитектурная эволюция	Изучение лекции 9, подготовка к ЛР 14–16	6

Модуль 4. Архитектурный технический долг и итоговый проект (ЛР 17–18)

Неделя	Тема	Вид СРС	Часы
10	Технический долг. АТАМ	Изучение лекции 9, подготовка к ЛР 16–17	6
11	Итоговый проект	Разработка архитектурной документации, ADR	12
12	Подготовка к экзамену	Повторение, решение задач	24

1.3. Рекомендации по организации времени

Принцип	Рекомендация
Регулярность	Занимайтесь самостоятельной работой ежедневно по 1–2 часа
Планирование	Составьте личный график на неделю с указанием конкретных задач
Сквозной проект	Выберите проект в начале семестра и ведите его на

Принцип	Рекомендация
	протяжении всех ЛР
Репозиторий	Ведите GitHub/GitLab репозиторий с кодом и диаграммами
Консультации	При возникновении вопросов обращайтесь к преподавателю

2. МЕТОДИЧЕСКИЕ РЕКОМЕНДАЦИИ ПО ИЗУЧЕНИЮ ТЕОРЕТИЧЕСКОГО МАТЕРИАЛА

2.1. Работа с лекционным материалом

Алгоритм изучения лекции:

Предварительный просмотр (15 минут): ознакомьтесь с планом лекции и ключевыми терминами.

Конспектирование (1 час): запишите основные определения, паттерны, диаграммы.

Визуализация (30 минут): перерисуйте ключевые диаграммы в PlantUML или на бумаге.

Закрепление (30 минут): ответьте на контрольные вопросы в конце лекции.

Связь с практикой (30 минут): подумайте, как применить материал к своему проекту.

Рекомендуемая структура конспекта лекции:

Лекция №[N]: [Название]

Ключевые термины

- [Термин 1]: [определение]

- [Термин 2]: [определение]

Основные концепции

[Концепция 1]

- Описание
- Пример
- Диаграмма

Связь с лабораторными работами

- ЛР [№]: как применяется

Вопросы для самопроверки

1. ...

2. ...

2.2. Работа с литературой

Рекомендуемый порядок изучения литературы

Тема	Приоритет	Источник
Введение в архитектуру ПО	★★ ★★★	Мартин Клеппман, главы 1–2
Clean Architecture	★★★★★	Роберт Мартин, главы 15–22
Domain-Driven Design	★★★★★	Эрик Эванс, главы 1–6, 9–14
Микросервисы	★★★★☆	Сэм Ньюмен, главы 1–4
CQRS и Event	★★★★☆	Мартин Клеппман,

Тема	Приоритет	Источник
Sourcing		глава 11

2.3. Работа с научными статьями и документацией

Тип источника	Где искать	Что читать
Официальная документация	Spring.io, MartinFowler.com	Spring Boot, Spring Cloud, C4 model
Научные статьи	Elibrary.ru, Google Scholar	АТАМ, архитектурные паттерны
Блоги экспертов	Habr.com, Medium.com	Разборы архитектурных решений

2.4. Подготовка к собеседованию по литературе

План подготовки:

Прочитать источник (2–3 часа).

Составить краткий конспект (0.5–1 страница).

Выделить 3–5 ключевых идей, которые можно применить в проекте.

Сформулировать 2–3 вопроса для обсуждения.

Подготовить пример из своего проекта, иллюстрирующий идеи автора.

3. МЕТОДИЧЕСКИЕ УКАЗАНИЯ ПО ВЫПОЛНЕНИЮ ПРАКТИЧЕСКИХ РАБОТ

3.1. Подготовка к лабораторной работе

Универсальный алгоритм подготовки к ЛР:

Изучите теоретическое обоснование (раздел 3 каждой ЛР) - 30 минут.

Разберите пример выполнения (раздел 4) - 30 минут.

Определите свой вариант сквозного проекта и задания - 10 минут.

Спланируйте реализацию (разбейте на подзадачи) - 20 минут.

Подготовьте инструменты (IDE, PlantUML, Docker и т.д.) - 10 минут.

3.2. Выполнение лабораторной работы

Рекомендации по написанию кода:

Рекомендация	Пояснение
Именован	Используйте понятные имена классов, методов, переменных
Комментарии	Комментируйте сложные архитектурные решения
Тесты	Пишите тесты параллельно с кодом (TDD)
Коммиты	Делайте частые коммиты с осмысленными сообщениями

Рекомендации по созданию диаграмм (PlantUML):

@startuml

' Рекомендуемый заголовок диаграммы

' Название: [Диаграмма контекста]

' Проект: [Логистика]

' Вариант: [1]

' Автор: [ФИО]

' Дата: [2024-03-15]

@enduml

3.3. Оформление отчёта

Структура отчёта по ЛР (шаблон):

markdown

Лабораторная работа №[N]

[Название работы]

****ФИО:**** _____

****Группа:**** _____

****Вариант:**** _____

****Сквозной проект:**** _____

1. Цель работы

[Из методических указаний]

2. Задачи работы

[Из методических указаний]

3. Ход выполнения

3.1. Описание проекта

[Краткое описание вашего сквозного проекта]

3.2. Выполнение задания 1

[Код, диаграммы, пояснения]

3.3. Выполнение задания 2

[Код, диаграммы, пояснения]

...

4. Тестирование

[Результаты тестов]

5. Вывод

[Что сделано, какие компетенции сформированы]

6. Ответы на контрольные вопросы

1. ...

2. ...

3.4. Защита лабораторной работы

Алгоритм подготовки к защите:

Шаг	Действие	Время
1	Проверить работоспособность кода	15 мин
2	Пересмотреть отчёт (цель, задачи, вывод)	10 мин
3	Повторить ответы на контрольные вопросы	20 мин
4	Подготовить демонстрацию (диаграммы, код)	15 мин
5	Проговорить защиту вслух	10 мин

4. ВОПРОСЫ ДЛЯ СОБЕСЕДОВАНИЯ ПО САМОСТОЯТЕЛЬНОМУ ИЗУЧЕНИЮ ЛИТЕРАТУРЫ

4.1. По книге Эрика Эванса «Domain-Driven Design»

№	Вопрос
1	Что такое Ubiquitous Language и почему он важен? Приведите пример из вашего проекта.
2	В чём разница между Entity и Value Object? Приведите 2 примера

№	Вопрос
	для вашего проекта.
3	Что такое Bounded Context и как определить его границы?
4	Опишите паттерны взаимодействия между контекстами (Shared Kernel, ACL, Customer-Supplier).
5	Что такое Aggregate и Aggregate Root? Как определить границы агрегата?
6	В чём отличие стратегического DDD от тактического?
7	Как Domain Events помогают в разделении контекстов?
8	Какие антипаттерны DDD вы знаете?

4.2. По книге Роберта Мартина «Clean Architecture»

№	Вопрос
1	Сформулируйте Dependency Rule в Clean Architecture.
2	Назовите 4 слоя Clean Architecture и их назначение.
3	Почему слой Entities не должен зависеть от фреймворков?
4	Что такое «The Clean Architecture scream»?
5	Как Clean Architecture связана с принципами SOLID?
6	В чём отличие Clean Architecture от трёхуровневой

№	Вопрос
	архитектуры?
7	Как тестировать Use Cases в Clean Architecture?
8	Какие компромиссы возникают при использовании Clean Architecture?

4.3. По книге Мартина Клеппмана «Высоконагруженные приложения»

№	Вопрос
1	Что такое CAP-теорема? Приведите пример компромисса для вашего проекта.
2	В чём разница между ACID и BASE транзакциями?
3	Что такое event sourcing? Чем он отличается от CRUD?
4	Когда применяется CQRS? Приведите пример для вашего проекта.
5	Что такое Saga и чем она отличается от 2PC?
6	Какие паттерны распределённых транзакций вы знаете?
7	Что такое компенсирующая транзакция?
8	Как обеспечить идемпотентность в распределённой системе?

4.4. По книге Сэма Ньюмена «Создание микросервисов»

№	Вопрос

№	Вопрос
1	В чём основные преимущества и недостатки микросервисной архитектуры?
2	Как организована коммуникация между микросервисами (синхронная/асинхронная)?
3	Что такое API Gateway и зачем он нужен?
4	Как управлять транзакциями в микросервисной архитектуре?
5	Что такое Strangler pattern и когда он применяется?
6	Как версионировать API в микросервисной архитектуре?
7	Что такое Circuit Breaker и зачем он нужен?
8	Как обеспечить observability (мониторинг, трейсинг, логирование) микросервисов?

4.5. По статьям и документации

№	Вопрос
1	Что такое нотация C4? Назовите 4 уровня.
2	Какой инструмент вы использовали для создания C4-диаграмм?
3	Что такое ADR (Architecture Decision Record)? Приведите структуру.

№	Вопрос
4	Что такое АТАМ и какие этапы он включает?
5	Как оценить архитектурный технический долг?
6	Что такое Feature Toggle? Какие стратегии развёртывания существуют?
7	Как обеспечить обратную совместимость API?
8	Что такое Serverless архитектура? Каковы её ограничения?

4.6. Критерии оценки ответа на собеседовании

Критерий	Отлично (5)	Хорошо (4)	Удовлетворительно (3)
Полнота ответа	Раскрыты все аспекты	Раскрыты основные аспекты	Раскрыты не все аспекты
Примеры	Приведены примеры из своего проекта	Приведены общие примеры	Примеры отсутствуют
Понимание	Показывает глубокое понимание	Понимает основные концепции	Поверхностное понимание
Связь с практикой	Чётко связывает теорию с ЛР	Частичная связь	Связь отсутствует

5. КРИТЕРИИ ОЦЕНИВАНИЯ КОМПЕТЕНЦИЙ

5.1. Система оценивания самостоятельной работы

Вид работы	Макс. балл	Минимальный порог
Выполнение ЛР (18 работ)	288 (16 × 18)	224 (14 × 16)
Защита ЛР (18 работ)	144 (8 × 18)	108 (6 × 18)
Активность на занятиях	20	10
Итоговый проект (ЛР 18)	100	70
Экзамен	100	60

5.2. Шкала оценивания компетенций

Уровень	Баллы (в % от максимума)	Характеристика
Высокий (продвинутый)	85–100%	Студент самостоятельно и полностью выполняет все виды СРС, глубоко понимает материал, способен аргументировать архитектурные решения, успешно защищает все работы
Базовый (пороговый)	55–84%	Студент выполняет основные виды СРС, допускает незначительные ошибки, понимает ключевые концепции, защищает работы с небольшими замечаниями

Уровень	Баллы (в % от максимума)	Характеристика
Ниже базового	<55%	Студент не выполняет СРС в требуемом объёме, имеет пробелы в понимании, не защитил необходимые работы

5.3. Критерии оценивания лабораторных работ

Критерий	Вес	Отлично	Хорошо	Удовлетворительно
Выполнение задания	40 %	Полностью, без ошибок	С незначительными ошибками	С существенными ошибками
Качество кода	15 %	Чистый код, следует принципам	Есть замечания по стилю	Код трудно читаем
Наличие тестов	5%	Тесты покрывают все сценарии	Тесты покрывают основные сценарии	Тесты отсутствуют
Оформление отчёта	15 %	Полностью соответствует требованиям	Есть незначительные отклонения	Существенные отклонения

Критерий	Вес	Отлично	Хорошо	Удовлетворительно
Ответы на вопросы	15 %	Полные, с примерами из проекта	Основные ответы верны	Ответы неполные

5.4. Критерии оценивания итогового проекта (ЛР 18)

Критерий	Макс. балл	Отлично	Хорошо	Удовлетворительно
Полнота документации	20	18–20	14–17	10–13
Качество С4 диаграмм	15	14–15	10–13	7–9
ADR (минимум 3)	15	14–15	10–13	7–9
АТАМ анализ	15	14–15	10–13	7–9
Обоснованность решений	15	14–15	10–13	7–9
Качество презентации	10	9–10	7–8	5–6
Ответы на вопросы	10	9–10	7–8	5–6
Итого	100	85–100	70–84	55–69

5.5. Рекомендации по повышению уровня компетенций

Уровень	Рекомендации
Ниже базового	Посетите консультации преподавателя; пересмотрите лекции; начните с выполнения более простых лабораторных работ; работайте в паре с более успешным студентом
Базовый	Углублённо изучите темы, где были ошибки; уделите больше внимания тестированию; используйте дополнительные источники литературы
Высокий	Участвуйте в научных конференциях; пишите статьи по архитектурным решениям; станьте наставником для отстающих студентов