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 «Конструирование программного обеспечения» для студентов
направления подготовки 09.03.04 Программная инженерия
Направленность (профиль)
«Инженерия сложных программных систем»

Ставрополь 2026

Содержание

Введение

1. Цели и задачи лабораторного практикума

Лабораторные работы по дисциплине «Конструирование программного обеспечения» направлены на формирование у студентов практических навыков разработки, тестирования, рефакторинга и обеспечения безопасности современных программных систем на платформе Java (Spring Boot).

Цель лабораторного практикума – закрепление теоретических знаний и получение практических компетенций в области:

- модульного, интеграционного и нагрузочного тестирования;
- рефакторинга и повышения качества кода;
- аутентификации, авторизации и информационной безопасности;
- объектно-реляционного отображения (ORM) и оптимизации запросов;
- реактивного и асинхронного программирования;
- альтернативных API (GraphQL).

Задачи лабораторного практикума:

- Освоить методологию Test-Driven Development (TDD) и инструменты тестирования (JUnit 5, Mockito, TestContainers, JMeter/k6).
- Научиться выявлять и устранять «запахи кода» с использованием статического анализа (SonarQube) и средств рефакторинга IntelliJ IDEA.
- Сформировать навыки реализации аутентификации (JWT, OAuth2) и авторизации (RBAC) с применением Spring Security.
- Освоить работу с Spring Data JPA и методы решения проблемы N+1 запросов.
- Изучить реактивное программирование на Spring WebFlux и Project Reactor.

- Научиться разрабатывать GraphQL API и сравнивать его с REST.

2. Перечень формируемых компетенций

В результате выполнения лабораторных работ студент должен сформировать следующие компетенции:

Код компетенции	Формулировка компетенции
ПК-3	Владеет навыками разработки программного обеспечения: способен реализовывать алгоритмы, писать код на современных языках программирования, создавать веб- и desktop-приложения, работать с базами данных.
ПК-4	Способен проводить тестирование, отладку и оценивать качество программного обеспечения, применяя инструменты автоматизированного тестирования и методы обеспечения надежности и безопасности.

3. Требования к подготовке к лабораторным работам

Для успешного выполнения лабораторных работ студент должен:

1. Владеть основами языка Java (синтаксис, ООП, коллекции, исключения, потоки ввода-вывода).
2. Иметь базовое представление о фреймворке Spring (IoC, DI, Spring Boot).
3. Понимать принципы работы REST API и баз данных (SQL, транзакции).
4. Уметь работать с системой контроля версий Git.
5. Иметь установленное программное обеспечение согласно перечню (раздел 9 РПД).

Подготовка к каждой работе включает:

1. Изучение теоретического материала по указанной теме.

2. Ознакомление с методическими указаниями и индивидуальным вариантом задания.
3. Написание исходного кода в среде разработки (IntelliJ IDEA / VS Code).
4. Подготовку отчета по установленной форме.

4. Порядок выполнения лабораторных работ

Каждая лабораторная работа выполняется в следующем порядке:

Этап	Содержание
1	Получение индивидуального варианта у преподавателя.
2	Изучение теоретических основ (лекции, учебная литература, документация).
3	Разработка кода согласно заданию.
4	Отладка и тестирование разработанного кода.
5	Оформление отчета по лабораторной работе.
6	Защита лабораторной работы (собеседование с преподавателем, демонстрация кода).

5. Структура отчета по лабораторной работе

Отчет по каждой лабораторной работе должен содержать следующие разделы:

- Титульный лист (с указанием дисциплины, темы работы, ФИО студента, группы, варианта).
- Цель работы (формулируется в соответствии с методическими указаниями).
- Постановка задачи (текст индивидуального задания).

- Теоретическая часть (краткое изложение основных понятий и инструментов, используемых в работе).

Практическая часть:

1. описание разработанных классов и методов;
2. листинги ключевых фрагментов кода;
3. описание тестов (для работ №1–3, 7–9);
4. скриншоты результатов выполнения.
5. Заключение (выводы о полученных навыках и решенных задачах).
6. Ответы на контрольные вопросы (письменно).
7. Список использованных источников.

Отчет оформляется в текстовом редакторе (Word, LibreOffice) или в формате Markdown с последующей конвертацией в PDF. Исходный код работы размещается в репозитории GitHub/GitLab/GitFlic, ссылка указывается в отчете.

6. Критерии оценивания лабораторных работ

Оценка	Критерий
5 (отлично)	Работа выполнена в полном объеме, код соответствует заданию, тесты успешно проходят, даны полные и аргументированные ответы на контрольные вопросы, отчет оформлен аккуратно.
4 (хорошо)	Работа выполнена в полном объеме, но есть незначительные замечания по оформлению кода или отчета, допущены неточности при ответах на контрольные вопросы.
3	Работа выполнена частично (не менее 70%

Оценка	Критерий
(удовлетворительно)	требований), код содержит ошибки, тесты проходят не полностью, ответы на контрольные вопросы неполные.
2 (неудовлетворительно)	Работа не выполнена или выполнено менее 50% требований, код неработоспособен, студент не может ответить на контрольные вопросы.

7. Перечень программного обеспечения и инструментов

Для выполнения лабораторных работ используется следующий стек технологий:

1. Системное ПО и среда разработки

Категория	Наименование	Статус в РФ / Лицензия
Операционная система	Astra Linux (версия «Орел»)	Отечественная ОС, включена в реестр ПО. Бесплатна для образования.
	Ubuntu / Debian Linux	Доступна. Свободная лицензия (GPL).
	Microsoft Windows	Доступна, но импортозамещение рекомендуется.
Среда разработки (IDE)	IntelliJ IDEA Community Edition	Доступна. Свободная лицензия Apache 2.0.
	Visual Studio	Доступен. Свободная

Категория	Наименование	Статус в РФ / Лицензия
	Code	лицензия MIT.
	Eclipse IDE	Доступна. Свободная лицензия EPL.

2. Язык программирования и сборка

Категория	Наименование	Статус в РФ / Лицензия
JDK	Liberica JDK (BellSoft)	Доступен. Свободная лицензия GPL. Сертифицирован для Astra Linux.
	OpenJDK	Доступен. Свободная лицензия GPL.
Система сборки	Apache Maven	Доступна. Лицензия Apache 2.0.
	Gradle	Доступен. Лицензия Apache 2.0.

3. Фреймворки и библиотеки (Java-стек)

Категория	Наименование	Статус в РФ / Лицензия
Основной фреймворк	Spring Boot (Open Source)	Доступен. Лицензия Apache 2.0. Срок бесплатной

Категория	Наименование	Статус в РФ / Лицензия
		поддержки Community версии – до июня 2026 г.
	Axiom Spring (рекомендовано)	Отечественный форк с полной поддержкой. Обеспечивает выпуск патчей безопасности после 2026 г.
Безопасность	Spring Security	Доступно. Лицензия Apache 2.0.
	JJWT (JWT)	Доступно. Лицензия Apache 2.0.
ORM	Hibernate	Доступно. Лицензия LGPL.
Реактивное программирование	Project Reactor	Доступно. Лицензия Apache 2.0.
	Spring WebFlux	Доступно. Лицензия Apache 2.0.
API	Spring for GraphQL	Доступно. Лицензия Apache 2.0.
	WebClient /	Доступно. Лицензия

Категория	Наименование	Статус в РФ / Лицензия
	RestTemplate	Apache 2.0.

4. Базы данных

Категория	**Наименование**	**Статус в РФ / Лицензия**
Основная СУБД (рекомендовано)	Postgres Pro Enterprise	Отечественная СУБД на основе PostgreSQL. Бесплатная академическая лицензия для вузов. Включена в реестр ПО.
Альтернативная СУБД	PostgreSQL	Доступна. Свободная лицензия PostgreSQL License.
СУБД для тестов	H2 Database	Доступна. Лицензия EPL / MPL.

5. Тестирование и качество кода

Категория	Наименование	Статус в РФ / Лицензия
Модульное тестирование	JUnit 5	Доступно. Лицензия EPL 2.0.
	Mockito	Доступно. Лицензия MIT.
	AssertJ	Доступно. Лицензия

Категория	Наименование	Статус в РФ / Лицензия
		Apache 2.0.
Интеграционное тестирование	TestContainers	Доступно. Лицензия MIT.
Нагрузочное тестирование	Apache JMeter	Доступно. Лицензия Apache 2.0.
	k6	Доступно. Лицензия AGPLv3.
Анализ качества кода	SonarQube (Community Build)	Доступно. Лицензия LGPLv3.
Покрытие кода	JaCoCo	Доступно. Лицензия EPL.

6. Контейнеризация и оркестрация

Категория	Наименование	Статус в РФ / Лицензия
Контейнеризация	Docker Desktop	Доступен. Бесплатная лицензия для образовательных учреждений.
	Docker Engine (Linux)	Доступен.

Категория	Наименование	Статус в РФ / Лицензия
		Свободная лицензия Apache 2.0.
Отечественный аналог	«Среда контейнеризации для Astra Linux»	Включена в состав Astra Linux.

7. Система контроля версий и совместная разработка

Категория	Наименование	Статус в РФ / Лицензия
Система контроля версий	Git	Доступна. Лицензия GPL.
Отечественная платформа	GitFlic	Российская платформа. Аналог GitHub. Включена в реестр ПО.
Зарубежные платформы	GitHub / GitLab	Доступны, но не рекомендуются для хранения критических данных.

8. Рекомендуемая литература

Основная:

1. Соммервилл, И. Инженерия программного обеспечения = Software Engineering : учебник / И. Соммервилл ; пер. с англ. – 10-

е изд. – Москва : Вильямс, 2021. – 816 с. – ISBN 978-5-907144-60-2. – Текст : непосредственный.

2. Мартин, Р. С. Чистый код: создание, анализ и рефакторинг = Clean Code : A Handbook of Agile Software Craftsmanship / Р. С. Мартин ; пер. с англ. – Санкт-Петербург : Питер, 2020. – 464 с. – ISBN 978-5-4461-0960-9. – Текст : непосредственный.
3. Программная инженерия. Визуальное моделирование программных систем : учебник для вузов / под науч. ред. проф. А. Н. Громова. – 3-е изд., пер. и доп. – Москва : Издательство Юрайт, 2026. – 285 с. – (Высшее образование). – ISBN 978-5-534-17952-6. – Текст : электронный // Образовательная платформа Юрайт [сайт]. – URL: https://urait.ru/bcode/545309
4. Базы данных. PostgreSQL : учебное пособие для вузов / под ред. С. В. Кузнецова. – Москва : Издательство Юрайт, 2026. – 312 с. – (Высшее образование). – ISBN 978-5-534-18923-5. – Текст : электронный // Образовательная платформа Юрайт [сайт]. – URL: https://urait.ru/bcode/547123
5. Spring в действии (Spring in Action) : Крейг Уоллс. – 6-е изд. – М. : ДМК Пресс, 2022. – 520 с. – ISBN 978-5-97060-985-7. – Текст : непосредственный.

Дополнительная:

1. Бейзер, Б. Тестирование черного ящика. Технологии функционального тестирования программного обеспечения и систем / Б. Бейзер ; пер. с англ. – Санкт-Петербург : Питер, 2018. – 320 с. – ISBN 978-5-4461-0642-4. – Текст : непосредственный.
2. Гамма, Э. Приемы объектно-ориентированного проектирования. Паттерны проектирования = Design Patterns: Elements of Reusable Object-Oriented Software / Э. Гамма, Р. Хелм, Р. Джонсон, Дж.

Влиссидес ; пер. с англ. – Санкт-Петербург : Питер, 2020. – 448 с. – ISBN 978-5-4461-1329-3. – Текст : непосредственный.

3. Архитектура ЭВМ и системное программное обеспечение : учебник для вузов / под ред. В. А. Миронова. – 6-е изд., пер. и доп. – Москва : Издательство Юрайт, 2025. – 480 с. – (Высшее образование). – ISBN 978-5-534-14567-8. – Текст : электронный // Образовательная платформа Юрайт [сайт]. – URL: https://urait.ru/bcode/545009

Успешное выполнение всех лабораторных работ позволит студенту сформировать практические навыки конструирования программного обеспечения на современном технологическом стеке Java/Spring, необходимые для профессиональной деятельности в области программной инженерии. Результаты лабораторных работ могут быть использованы при выполнении курсовых и выпускных квалификационных работ, а также в последующей профессиональной деятельности.

Лабораторная работа № 1

Тест-дизайн: TDD с JUnit 5 и Mockito

Цель работы: освоить методологию Test-Driven Development (TDD) и приобрести навыки написания модульных тестов с использованием JUnit 5, Mockito и AssertJ.

Задачи:

1. Изучить принципы «красный-зеленый-рефакторинг».
2. Научиться создавать mock-объекты с помощью Mockito.
3. Освоить написание параметризованных тестов.
4. Научиться проверять исключительные ситуации.

Краткое содержание темы:

TDD - техника разработки программного обеспечения, при которой тесты пишутся до реализации кода. Цикл TDD: написание падающего теста (красный), написание минимального кода для прохождения теста (зеленый), рефакторинг с сохранением работоспособности тестов. JUnit 5 - фреймворк для модульного тестирования Java-приложений. Mockito - библиотека для создания mock-объектов и проверки взаимодействий. AssertJ - библиотека для выразительных утверждений (fluent assertions).

Индивидуальные варианты

Вариант	Задание
1	Реализовать класс Calculator с методами: add, subtract, multiply, divide. Написать тесты для всех методов (деление на ноль – исключение).
2	Реализовать класс StringUtils с методами: reverse, isPalindrome, countVowels. Написать

Вариант	Задание
	параметризованные тесты.
3	Реализовать класс BankAccount с методами: deposit, withdraw, getBalance. Использовать Mockito для мока внешнего сервиса аудита.
4	Реализовать класс UserService (регистрация, поиск по email). UserRepository – mock. Проверить, что при дублировании email выбрасывается исключение.
5	Реализовать класс OrderService (расчет скидки). Скидка зависит от класса клиента и суммы заказа. Использовать параметризованные тесты.
6	Реализовать класс PasswordValidator (проверка сложности пароля: длина ≥8, цифра, заглавная буква, спецсимвол). Параметризованные тесты.
7	Реализовать класс Cache (LRU кэш) с методами put и get. Тесты должны проверять вытеснение наименее используемых элементов.
8	Реализовать класс Fraction (дробь) с методами сложения, вычитания, умножения, деления. Тесты – сравнение с плавающей точкой (Delta).
9	Реализовать класс TennisGame (подсчет очков теннисного матча: 0-15-30-40-game-deuce-advantage). Покрыть все состояния.

Вариант	Задание
10	Реализовать класс ShoppingCart с товарами и купонами. Mock для CouponService. Тест – применение скидки по купону.
11	Реализовать класс EmailSender. Mock для SmtpClient. Проверить, что при некорректном email выбрасывается исключение.
12	Реализовать класс PrimeChecker. Параметризованные тесты для проверки простых чисел. TDD – сначала тесты, потом реализация.
13	Реализовать класс RomanNumerals (конвертер арабских чисел в римские). Тесты для диапазона 1–3999.
14	Реализовать класс VotingMachine (голосование за кандидатов). Проверка на повторное голосование – исключение.
15	Реализовать класс JsonParser (парсинг JSON). Mock для HttpClient. Тест – обработка ошибочного JSON.

Контрольные вопросы:

1. Что такое TDD и какие этапы включает «красный-зеленый-рефакторинг»?
2. В чем отличие mock от stub?
3. Как проверить, что метод был вызван определенное количество раз (Mockito.verify)?

4. Что такое параметризованные тесты в JUnit 5? Приведите пример.
5. Как тестировать методы, выбрасывающие исключения?
6. Какие преимущества дает AssertJ по сравнению с встроенными assert?
7. Что такое code coverage и какие значения считаются достаточными?

Лабораторная работа № 2

Интеграционное тестирование: TestContainers + PostgreSQL

Цель работы: освоить интеграционное тестирование Spring Boot приложений с использованием TestContainers для изоляции тестовой базы данных PostgreSQL.

Задачи:

1. Настроить TestContainers для PostgreSQL в тестах Spring Boot.
2. Написать интеграционные тесты для репозиториев Spring Data JPA.
3. Проверить корректность работы кастомных запросов и транзакций.
4. Настроить автоматическое создание и удаление контейнеров.

Краткое содержание темы:

TestContainers - библиотека для Java, позволяющая запускать Docker-контейнеры (базы данных, брокеры сообщений, браузеры) в тестах. В сочетании с Spring Boot Test и @DataJpaTest обеспечивает изолированное тестирование слоя доступа к данным. PostgreSQL - реляционная СУБД. JPA (Java Persistence API) - стандарт для ORM, Hibernate - реализация.

Индивидуальные варианты (15 вариантов):

Вариант	Задание (сущности и запросы)
1	User (id, name, email) – поиск по email, проверка уникальности.
2	Product (id, name, price, stock) – поиск товаров с ценой выше заданной.
3	Order (id, userId, total, status) + OrderItem – проверка каскадных операций.
4	Category (id, name, parent) + ProductCategory – иерархический запрос.
5	Employee (id, name, department, salary) – поиск сотрудников с зарплатой выше средней.
6	Book (id, title, author, year) – поиск книг, изданных после заданного года.
7	Student (id, name, groupNumber, gpa) – поиск студентов со средним баллом >4.0.
8	Hotel (id, name, city, stars) + Room – поиск свободных номеров в отеле.
9	Car (id, brand, model, year, price) – поиск автомобилей по диапазону цены.
10	Course (id, title, credits) + Enrollment (studentId, courseId, grade) – подсчет среднего балла по курсу.
11	Article (id, title, author, views, likes) – поиск топ-5

Вариант	Задание (сущности и запросы)
	популярных статей.
12	Sensor (id, type, location, value, timestamp) – выборка данных за последние 24 часа.
13	Transaction (id, fromAccount, toAccount, amount, date) – поиск транзакций по счету.
14	Task (id, title, assignee, status, dueDate) – поиск просроченных задач.
15	Log (id, level, message, timestamp) – поиск ошибок (ERROR) за период.

Контрольные вопросы:

1. Что такое TestContainers и для чего он используется?
2. Как настроить TestContainers для PostgreSQL в Spring Boot тесте?
3. Чем отличается @DataJpaTest от @SpringBootTest?
4. Как обеспечить изоляцию тестов при использовании реальной базы данных?
5. Что такое @Sql и @SqlGroup? Когда их используют?
6. Как тестировать кастомные запросы в Spring Data JPA?
7. Какие проблемы решает использование TestContainers по сравнению с H2 в памяти?

Лабораторная работа № 3

Нагрузочное тестирование: JMeter против Spring Boot API

Цель работы: освоить нагрузочное тестирование REST API с использованием Apache JMeter (или k6), научиться анализировать результаты и выявлять узкие места.

Задачи:

1. Создать тестовый REST API на Spring Boot с несколькими эндпоинтами.
2. Настроить Spring Boot Actuator (метрики, health, info).
3. Разработать нагрузочный сценарий в JMeter/k6.
4. Провести тестирование при разной нагрузке и проанализировать результаты.

Краткое содержание темы:

Нагрузочное тестирование - оценка поведения системы под ожидаемой нагрузкой. JMeter - инструмент для нагрузочного тестирования с графическим интерфейсом. k6 - современный инструмент для тестирования производительности с написанием сценариев на JavaScript. Spring Boot Actuator предоставляет метрики (запросы в секунду, время ответа, использование памяти). Ключевые метрики: throughput (пропускная способность), latency (задержка), error rate (процент ошибок).

Индивидуальные варианты

Вариант	Эндпоинт	Характеристика нагрузки
1	GET /fibonacci/{n} (числа Фибоначчи)	1000 запросов, 50 потоков, n=30
2	POST /prime (проверка на простоту)	5000 запросов, 100 потоков, числа 10^6–10^7
3	GET /factorial/{n}	2000 запросов, 50

Вариант	Эндпоинт	Характеристика нагрузки
	(факториал)	потоков, n=20
4	GET /sort (сортировка массива 1000 чисел)	3000 запросов, 100 потоков
5	POST /encrypt (шифрование строки)	2000 запросов, 50 потоков
6	GET /products/{id} (поиск товара)	10000 запросов, 200 потоков
7	POST /search (полнотекстовый поиск)	5000 запросов, 100 потоков
8	GET /report/generate (генерация отчета)	500 запросов, 10 потоков (тяжелый)
9	POST /calculate (сложные вычисления)	2000 запросов, 50 потоков
10	GET /db-query (сложный SQL-запрос)	4000 запросов, 100 потоков
11	GET /cache-test (чтение с кэшем Redis)	15000 запросов, 300 потоков
12	POST /upload (загрузка файла 1 МБ)	1000 запросов, 30 потоков

Вариант	Эндпоинт	Характеристика нагрузки
13	GET /heavy-math (численный метод)	1500 запросов, 40 потоков
14	POST /graphql (сложный GraphQL-запрос)	3000 запросов, 60 потоков
15	GET /stream (Server-Sent Events)	500 соединений, длительность 60 сек

Контрольные вопросы:

1. Чем нагрузочное тестирование отличается от стресс-тестирования?
2. Какие метрики производительности являются ключевыми?
3. Как настроить JMeter для имитации 100 одновременных пользователей?
4. Как интерпретировать результаты: 95-й перцентиль, среднее время, пропускная способность?
5. Что такое «узкое место» (bottleneck) и как его выявить?
6. Какие возможности предоставляет Spring Boot Actuator для мониторинга?
7. Сравните JMeter и k6: преимущества и недостатки.

Лабораторная работа № 4

Рефакторинг: устранение запахов кода в legacy-проекте

Цель работы: освоить методы рефакторинга, научиться выявлять «запахи кода» и устранять их с использованием статического анализа (SonarQube) и инструментов IntelliJ IDEA.

Задачи:

1. Проанализировать legacy-код с помощью SonarQube.
2. Выявить «запахи кода» (code smells): длинные методы, дублирование, примитивную одержимость, завистливые функции и др.
3. Выполнить рефакторинг: извлечение методов, замена условного оператора полиморфизмом, встраивание класса и др.
4. Проверить, что после рефакторинга функциональность не нарушена (тесты).
5. Снизить технический долг (technical debt) проекта.

Краткое содержание темы:

«Запахи кода» (code smells) - поверхностные индикаторы проблем в коде, которые могут затруднять поддержку и развитие. Рефакторинг - процесс изменения внутренней структуры кода без изменения его внешнего поведения. SonarQube - платформа для непрерывного анализа качества кода. Инструменты рефакторинга в IntelliJ IDEA: Extract Method, Extract Constant, Inline, Rename, Move, Pull Members Up/Down, Introduce Parameter Object и др.

Индивидуальные варианты

Каждый вариант содержит исходный код с 5-7 запахами (выдается преподавателем отдельно). Студент должен:

1. Запустить SonarQube и выявить все проблемы.
2. Классифицировать проблемы по типам.
3. Выполнить рефакторинг.
4. Предоставить отчет: «было – стало» (листинг кода до/после).

Типы заданий по вариантам:

Вариант	Тип legacy-кода	Основные запахи
1	Расчет зарплаты с кучей if-else	Длинный метод, дублирование, магические числа
2	Обработка заказов с типами оплаты	Примитивная одержимость (строковые коды), switch-case
3	Класс Customer с 25 полями и 15 методами	God object (божественный объект), длинный метод
4	Функция валидации формы (300 строк)	Длинный метод, дублирование, комментарии вместо кода
5	Иерархия фигур (Circle, Square, Triangle)	Завистливая функция (Feature Envy), неправильное наследование
6	Парсинг логов с повторами	Дублирование кода (copy-paste)
7	Работа с датами (своя реализация)	Переизобретение колеса, примитивная одержимость
8	Класс ReportGenerator с методом 400 строк	God object, длинный метод,

Вариант	Тип legacy-кода	Основные запахи
		временное поле
9	Обработка ошибок через коды возврата (-1, 0, 1)	Магические числа, отсутствие исключений
10	Singleton с 10 методами и глобальным состоянием	Скрытая зависимость, глобальное состояние
11	Класс-утилита из 30 статических методов	Ленивый класс, отсутствие ООП
12	Работа с файлами: дублирование try-catch-finally	Дублирование, длинный метод
13	Конфигурация через Properties (строковые ключи)	Примитивная одержимость, магические строки
14	Класс Order содержит методы работы с БД	Раздутый интерфейс, смешение слоев
15	Цепочка if-else для расчета скидки (7 уровней)	Длинный метод, усложненная логика

Контрольные вопросы:

1. Что такое «запахи кода» (code smells)? Приведите 5 примеров.
2. Чем рефакторинг отличается от переписывания кода?

3. Как проверить, что рефакторинг не изменил поведение программы?
4. Какие инструменты рефакторинга есть в IntelliJ IDEA?
5. Что такое технический долг (technical debt) и как его измерить?
6. Как SonarQube помогает в рефакторинге?
7. В чем разница между Extract Method и Inline Method?

Лабораторная работа № 5

Безопасность: JWT + Spring Security + RBAC

Цель работы: освоить реализацию аутентификации и авторизации в Spring Boot приложении с использованием JWT (JSON Web Token) и ролевой модели доступа (RBAC).

Задачи:

1. Настроить Spring Security для защиты REST API.
2. Реализовать регистрацию и аутентификацию пользователей.
3. Реализовать генерацию и валидацию JWT (библиотека jjwt).
4. Реализовать ролевую модель (ROLE_USER, ROLE_ADMIN, ROLE_MODERATOR).
5. Настроить доступ к эндпоинтам в зависимости от роли.
6. Реализовать хеширование паролей с помощью BCrypt.

Краткое содержание темы:

Spring Security - фреймворк для аутентификации и авторизации в Java-приложениях. JWT (JSON Web Token) - компактный URL-безопасный токен для передачи информации между сторонами. RBAC (Role-Based Access Control) - модель управления доступом, основанная на ролях. BCrypt - адаптивная хеш-функция для надежного хеширования паролей.

Индивидуальные варианты

Вариант	Роли	Эндпоинты и права
1	USER, ADMIN	USER – чтение своего профиля; ADMIN – чтение всех пользователей
2	USER, MODERATOR, ADMIN	MODERATOR – модерация контента; ADMIN – управление пользователями
3	EMPLOYEE, MANAGER, HR	EMPLOYEE – просмотр своего расписания; MANAGER – просмотр отдела; HR – управление сотрудниками
4	READER, AUTHOR, EDITOR	READER – чтение статей; AUTHOR – создание статей; EDITOR – редактирование любых статей
5	CUSTOMER, SELLER, ADMIN	CUSTOMER – просмотр товаров; SELLER – управление своими товарами; ADMIN – управление всеми
6	STUDENT, TEACHER, ADMIN	STUDENT – просмотр своих оценок; TEACHER – выставление оценок; ADMIN – управление курсами
7	USER, VIP, ADMIN	USER – базовый доступ; VIP – доступ к премиум-контенту; ADMIN – управление пользователями
8	CLIENT, ACCOUNTANT,	CLIENT – просмотр счета; ACCOUNTANT – проведение

Вариант	Роли	Эндпоинты и права
	DIRECTOR	платежей; DIRECTOR – просмотр всех счетов
9	USER, SUPPORT, ADMIN	USER – создание тикетов; SUPPORT – обработка тикетов; ADMIN – управление системой
10	VIEWER, OPERATOR, SUPERADMIN	VIEWER – просмотр логов; OPERATOR – выполнение операций; SUPERADMIN – любые действия
11	RESIDENT, SECURITY, MANAGER	RESIDENT – доступ в жилую зону; SECURITY – управление доступом; MANAGER – управление зонами
12	DRIVER, DISPATCHER, ADMIN	DRIVER – обновление статуса; DISPATCHER – распределение заказов; ADMIN – управление системой
13	PATIENT, DOCTOR, ADMIN	PATIENT – запись на прием; DOCTOR – просмотр расписания; ADMIN – управление врачами
14	BUYER, SUPPLIER, ADMIN	BUYER – оформление заказов; SUPPLIER – управление поставками; ADMIN – управление всеми
15	GUEST,	GUEST – только чтение;

Вариант	Роли	Эндпоинты и права
	REGISTERED, PREMIUM	REGISTERED – комментирование; PREMIUM – создание контента

Контрольные вопросы:

1. Что такое JWT и из каких частей он состоит (Header, Payload, Signature)?
2. Как Spring Security интегрируется с JWT?
3. В чем разница между authentication и authorization?
4. Что такое RBAC и какие у него преимущества?
5. Почему пароли нужно хешировать с солью (BCrypt), а не хранить в открытом виде?
6. Как настроить доступ к эндпоинту только для ROLE_ADMIN?
7. Как обновлять JWT при истечении срока действия (refresh token)?

Лабораторная работа № 6

Безопасность: OAuth2 (вход через GitHub / Google)

Цель работы: освоить реализацию OAuth2-аутентификации в Spring Boot приложении (вход через внешних провайдеров: GitHub, Google, Yandex, VK).

Задачи:

1. Зарегистрировать OAuth2-приложение у провайдера (GitHub / Google).
2. Настроить Spring Security OAuth2 Client.
3. Реализовать вход через внешнего провайдера (Single Sign-On).
4. Получить профиль пользователя (имя, email, аватар).
5. Реализовать выход (logout) и управление сессиями.

Краткое содержание темы:

OAuth 2.0 - протокол авторизации, позволяющий приложениям получать ограниченный доступ к данным пользователя без передачи пароля. Spring Security OAuth2 Client - модуль для интеграции с внешними провайдерами (GitHub, Google, Facebook, Yandex, VK). SSO (Single Sign-On) - технология единого входа.

Индивидуальные варианты

Вариант	Провайдер	Дополнительные требования
1	GitHub	Получение списка репозиториев пользователя
2	GitHub	Проверка членства в организации
3	Google	Получение календаря (Google Calendar API)
4	Google	Получение контактов (Google People API)
5	Yandex	Получение информации о диске (Yandex Disk)
6	VK	Получение списка друзей (VK API)
7	GitHub + Google	Два провайдера, объединение аккаунтов
8	Google	Вход только для корпоративного домена

Вариант	Провайдер	Дополнительные требования
		(@mycompany.com)
9	GitHub	Получение email (даже если скрыт)
10	Yandex	Обновление аватара через Yandex API
11	VK	Публикация поста на стене от имени пользователя
12	Google	Сохранение refresh token для офлайн-доступа
13	GitHub	Создание gist от имени пользователя
14	Yandex	Получение списка писем (Yandex Mail API)
15	VK	Получение статистики группы (groups.getById)

Контрольные вопросы:

1. Для чего используется OAuth2? Чем он отличается от базовой аутентификации?
2. Что такое Authorization Code Flow и Implicit Flow?
3. Что такое Client ID и Client Secret?
4. Что такое access token и refresh token?

5. Как Spring Security OAuth2 Client упрощает интеграцию с провайдерами?
6. Как добавить нового провайдера (например, Яндекс или VK) через application.yml?
7. Как обрабатывать ошибки авторизации (отказ пользователя, истекший токен)?

Лабораторная работа № 7

ORM: устранение N+1 в Spring Data JPA

Цель работы: освоить методы устранения проблемы N+1 запросов в Spring Data JPA / Hibernate.

Задачи:

1. Смоделировать сущности с отношениями @OneToMany и @ManyToOne.
2. Воспроизвести проблему N+1 запросов.
3. Решить проблему с помощью @EntityGraph, JOIN FETCH, @NamedEntityGraph.
4. Сравнить производительность различных подходов.
5. Настроить логирование SQL-запросов.

Краткое содержание темы:

Проблема N+1 запросов возникает при загрузке коллекций в JPA/Hibernate: один запрос для родительской сущности и N запросов для каждой дочерней. Решения: JOIN FETCH (один запрос с JOIN), @EntityGraph (декларативное описание графа загрузки), @NamedEntityGraph (многократное использование), Batch Fetching (пакетная загрузка).

Индивидуальные варианты

Вариант	Сущности и связи	Запрос (выгрузка)
1	Author ↔ Book (OneToMany)	Все авторы с книгами
2	Category ↔ Product (OneToMany)	Категории с товарами (где цена > 100)
3	User ↔ Order ↔ OrderItem (цепочка)	Пользователь → заказы → товары
4	Department ↔ Employee (OneToMany)	Отдел со всеми сотрудниками
5	Post ↔ Comment (OneToMany)	Пост с комментариями (первые 10)
6	Student ↔ Course (ManyToMany)	Студенты с курсами
7	Library ↔ Book (OneToMany)	Библиотека со списком книг
8	Team ↔ Player (OneToMany)	Команда с игроками (фильтр по позиции)
9	Parent ↔ Child (OneToMany)	Родители с детьми (сортировка по

Вариант	Сущности и связи	Запрос (выгрузка)
		имени)
10	Doctor ↔ Patient (ManyToMany)	Пациенты с лечащим врачом
11	School ↔ Class ↔ Student (два уровня)	Школа → классы → ученики
12	Forum ↔ Topic ↔ Message (три уровня)	Форум → топики → сообщения
13	Warehouse ↔ Product (ManyToMany)	Склад с товарами (количество > 0)
14	Hotel ↔ Room ↔ Booking (цепочка)	Отель → комнаты → бронирования
15	Movie ↔ Actor (ManyToMany)	Фильмы с актерами (фильтр по году)

Контрольные вопросы:

1. В чем заключается проблема N+1 запросов в JPA/Hibernate?
2. Чем JOIN FETCH отличается от обычного JOIN?
3. Что такое EntityGraph и как его использовать?
4. Когда использовать @EntityGraph, а когда - JOIN FETCH?
5. Как включить логирование SQL-запросов в Spring Boot?
6. Что такое Batch Fetching и когда его применяют?
7. Как проверить, что проблема N+1 устранена?

Лабораторная работа № 8

Реактивное программирование: WebFlux + Reactor (сравнение с MVC)

Цель работы: освоить реактивное программирование на Spring WebFlux и Project Reactor, сравнить производительность с традиционным Spring MVC.

Задачи:

1. Создать два проекта: Spring MVC (Tomcat, блокирующий) и Spring WebFlux (Netty, неблокирующий).
2. Реализовать эндпоинты с задержкой (Thread.sleep в MVC, delay в WebFlux).
3. Провести нагрузочное тестирование и сравнить throughput при разных сценариях.
4. Реализовать потоковую обработку данных (Flux) и клиент WebClient.

Краткое содержание темы:

Reactive Programming - парадигма асинхронного программирования, основанная на потоках данных и распространении изменений. Project Reactor - реализация reactive streams для JVM с типами Mono (0..1) и Flux (0..N). Spring WebFlux - реактивный веб-фреймворк на основе Netty, неблокирующий I/O. WebClient - неблокирующий HTTP-клиент.

Индивидуальные варианты

Вариант	Задание (сравнение MVC vs WebFlux)
1	Эндпоинт с задержкой 200 мс (Thread.sleep / delay). Сравнить RPS.
2	Параллельный вызов 3 внешних API (WebClient / RestTemplate).

Вариант	Задание (сравнение MVC vs WebFlux)
3	Загрузка файла 10 МБ (блокирующий vs неблокирующий I/O).
4	SSE (Server-Sent Events) – поток 1000 событий.
5	WebSocket – эхо-сервер (MVC vs WebFlux).
6	Обработка потока чисел (Flux.range 1..10000) с операторами map, filter.
7	Объединение двух потоков (zip, merge).
8	Поток из файла (чтение построчно, неблокирующее).
9	Backpressure – медленный consumer vs быстрый producer.
10	Обработка ошибок в реактивной цепочке (onErrorResume, fallback).
11	Таймауты в WebClient (успех/ошибка).
12	Повтор попыток (retry) при сбоях внешнего сервиса.
13	Потоковая загрузка данных из БД (R2DBC + Postgres).
14	Кастомный оператор (transform) для логирования.
15	Сравнение использования памяти при 1000 параллельных запросах.

Контрольные вопросы:

1. В чем разница между реактивным (WebFlux) и блокирующим (MVC) подходом?
2. Что такое Mono и Flux в Project Reactor?
3. Как работает backpressure?
4. Какие преимущества дает неблокирующий I/O (Netty) при высоких нагрузках?
5. Как выполнить параллельные вызовы в WebClient?
6. Чем отличается WebClient от RestTemplate?
7. В каких случаях WebFlux не дает выигрыша, а где – критически важен?

Лабораторная работа № 9

Альтернативные API: GraphQL vs REST (сравнение)

Цель работы: освоить GraphQL API на Spring Boot, сравнить с REST по гибкости запросов, количеству запросов и производительности.

Задачи:

1. Создать GraphQL API с использованием Spring for GraphQL.
2. Реализовать выборку полей (field selection) и связанные сущности.
3. Реализовать DataLoader для устранения N+1 в GraphQL.
4. Сравнить GraphQL и REST по: количеству запросов, объему данных, сложности клиента.
5. Реализовать мутации (create, update, delete).

Краткое содержание темы:

GraphQL - язык запросов к API, разработанный Facebook. Позволяет клиенту запрашивать именно те поля, которые нужны, и получать связанные данные за один запрос. Spring for GraphQL - официальная интеграция

GraphQL со Spring. DataLoader - решение проблемы N+1 запросов в GraphQL. Schema Definition Language (SDL) - описание схемы GraphQL.

Индивидуальные варианты

Вариант	Сущности и запросы
1	Book (id, title, author, year) – запрос с выбором полей
2	User (id, name, email, posts) + Post (title, body) – связанные данные
3	Product (id, name, price, category, reviews) – агрегация (avg rating)
4	Order (id, total, items) + OrderItem (productId, quantity) – вложенные
5	Movie (id, title, director, actors) – список актеров
6	University (name, departments) + Department (name, professors) – два уровня
7	GitHub-like: Repo (name, owner, stars, forks, issues) – связанные
8	SocialNetwork: User (friends, posts, likes) – графовые запросы
9	E-commerce: Category (products), Product (reviews), Review (author) – три уровня
10	Blog: Author (posts, comments) + Comment (author) – циклический запрос

Вариант	Сущности и запросы
11	CRM: Company (contacts, deals) + Deal (products) – цепочка
12	Library: Reader (books), Book (reviews) – обратные связи
13	Hotel: Hotel (rooms, bookings) + Booking (guest) – глубокий запрос
14	University 2: Student (courses, grades), Course (professor) – с агрегацией
15	Forum: Category (topics), Topic (posts), Post (author) – 3 уровня

Контрольные вопросы:

Чем GraphQL отличается от REST по модели запросов?

Что такое DataLoader и какую проблему он решает?

Как определить схему GraphQL через SDL?

Как в GraphQL реализовать пагинацию (connections, cursors)?

В чем разница между query и mutation?

Какие преимущества дает GraphQL для мобильных клиентов?

Когда REST всё еще лучше GraphQL? Какие у GraphQL недостатки?

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
«Конструирование программного обеспечения»
для студентов направления подготовки
09.03.04 Программная инженерия Направленность (профиль)
«Инженерия сложных программных систем»

Ставрополь 2026

ВВЕДЕНИЕ

Целью самостоятельной работы студентов по дисциплине «Конструирование программного обеспечения» является формирование и закрепление профессиональных компетенций в области разработки, тестирования, рефакторинга и обеспечения безопасности программных систем на платформе Java/Spring, а также развитие навыков самоорганизации, критического мышления и самостоятельного поиска информации.

Задачи самостоятельной работы:

1. Закрепление теоретических знаний, полученных на лекционных занятиях.
2. Формирование практических навыков разработки программного кода с использованием современных технологий (Spring Boot, JUnit, Mockito, TestContainers, JPA, Spring Security, WebFlux, GraphQL).
3. Развитие умения работать с научно-технической литературой, нормативной документацией и ресурсами сети «Интернет».
4. Подготовка к выполнению лабораторных работ, курсового проекта и промежуточной аттестации.
5. Формирование навыков рефлексии и самооценки результатов учебной деятельности.

Место самостоятельной работы в учебном процессе

Самостоятельная работа является обязательной частью учебного процесса и выполняется студентом во внеаудиторное время. Объем самостоятельной работы по дисциплине составляет 108 часов (3 зачетные единицы). Соотношение аудиторной и самостоятельной работы – 1:1.

Формируемые компетенции

В результате выполнения самостоятельной работы студент должен сформировать следующие компетенции:

Код компетенции	Формулировка компетенции

Код компетенции	Формулировка компетенции
ПК-3	Владеет навыками разработки программного обеспечения: способен реализовывать алгоритмы, писать код на современных языках программирования, создавать веб- и desktop-приложения, работать с базами данных.
ПК-4	Способен проводить тестирование, отладку и оценивать качество программного обеспечения, применяя инструменты автоматизированного тестирования и методы обеспечения надежности и безопасности.

1. ОБЩАЯ ХАРАКТЕРИСТИКА САМОСТОЯТЕЛЬНОЙ РАБОТЫ

1.1. Виды самостоятельной работы

Самостоятельная работа по дисциплине «Конструирование программного обеспечения» включает следующие виды:

№	Вид самостоятельной работы	Трудоемкость (час.)	Формируемые компетенции
1	Подготовка к лекционным занятиям (изучение теоретического материала)	5	ПК-3, ПК-4
2	Подготовка к лабораторным работам (изучение методических указаний, написание кода, отладка, тестирование)	5	ПК-3, ПК-4

№	Вид самостоятельной работы	Трудоемкость (час.)	Формируемые компетенции
3	Выполнение индивидуальных заданий (9 лабораторных работ по 15 вариантов)	20	ПК-3, ПК-4
4	Оформление отчетов по лабораторным работам	6	ПК-3
Итого		36	

1.2. Планирование самостоятельной работы по модулям

Модуль	Тема	Виды СРС	Неделя семестра
1	Тест-дизайн. TDD с JUnit 5 и Mockito	Подготовка к ЛР №1, выполнение, оформление отчета	1–2
2	Интеграционное тестирование. TestContainers + PostgreSQL	Подготовка к ЛР №2, выполнение, оформление отчета	3–4
3	Нагрузочное тестирование. JMeter vs Spring Boot	Подготовка к ЛР №3, выполнение, оформление отчета	5–6

Модуль	Тема	Виды СРС	Неделя семестра
4	Рефакторинг. Устранение запахов кода	Подготовка к ЛР №4, выполнение, оформление отчета	7–8
5	Безопасность. JWT + Spring Security + RBAC	Подготовка к ЛР №5, выполнение, оформление отчета	9–10
6	Безопасность. OAuth2 (GitHub/Google)	Подготовка к ЛР №6, выполнение, оформление отчета	11–12
7	ORM. Устранение N+1 в Spring Data JPA	Подготовка к ЛР №7, выполнение, оформление отчета	13–14
8	Реактивное программирование. WebFlux + Reactor	Подготовка к ЛР №8, выполнение, оформление отчета	15–16
9	Альтернативные API. GraphQL vs REST	Подготовка к ЛР №9, выполнение, оформление отчета	17–18

1.3. Рекомендации по организации времени

Для эффективной организации самостоятельной работы рекомендуется:

Составить индивидуальный план-график выполнения работ на семестр.

Выделять на самостоятельную работу не менее 6 часов в неделю.

Чередовать виды деятельности: изучение теории – программирование – тестирование – оформление отчета.

Использовать инструменты тайм-менеджмента (Trello, YouTrack, Jira) для отслеживания прогресса.

Своевременно обращаться к преподавателю за консультацией.

2. МЕТОДИЧЕСКИЕ РЕКОМЕНДАЦИИ ПО ИЗУЧЕНИЮ ТЕОРЕТИЧЕСКОГО МАТЕРИАЛА

2.1. Работа с учебной литературой

При изучении теоретического материала рекомендуется:

Ознакомительное чтение – просмотр текста для получения общего представления о теме.

Изучающее чтение – детальное прочтение с выделением ключевых понятий, формул, алгоритмов.

Аналитическое чтение – критический анализ материала, сравнение разных точек зрения.

Реферативное чтение – составление конспекта, тезисов, аннотации.

2.2. Работа с электронными ресурсами

Рекомендуемые электронные ресурсы для самостоятельного изучения (доступны на территории РФ):

	Ресурс	Назначение
	https://ru.spring.io	Официальная документация Spring (русскоязычный раздел)
	https://www.postgrespro.ru/docs	Документация Postgres Pro
	https://gitflic.ru	Российская платформа

	Ресурс	Назначение
		совместной разработки
	https://metanit.com/java/	Учебник по Java и Spring (бесплатно, на русском)
	https://habr.com/ru/hub/spring	Статьи и туториалы по Spring (русскоязычное сообщество)
	https://roadmap.sh/spring	Spring Roadmap (образовательные материалы)

2.3. Конспектирование лекций

Рекомендуемая форма конспекта:

Структурный элемент	Содержание
Тема	Название темы лекции
Ключевые понятия	Определения основных терминов
Основные положения	Теоремы, законы, принципы (в тезисной форме)
Алгоритмы / схемы	Блок-схемы, диаграммы, последовательности действий
Вопросы для самопроверки	3–5 вопросов по теме для последующего повторения
Источники	Ссылки на литературу и интернет-

Структурный элемент	Содержание
	ресурсы

2.4. Перечень тем для самостоятельного изучения

№	Тема для самостоятельного изучения	Рекомендуемая литература	Трудоемкость (час.)
1	Принципы SOLID и их применение в Spring	Мартин Р.С. «Чистый код», гл. 10–11	2
2	Паттерны проектирования GoF в Java-приложениях	Гамма Э. «Паттерны проектирования»	2
3	Жизненный цикл бинов в Spring IoC контейнере	Документация Spring	2
4	Аспектно-ориентированное программирование (AOP)	Документация Spring	2
5	Транзакции в Spring: @Transactional, Propagation, Isolation	Документация Spring	2
6	Кэширование в Spring Boot: @Cacheable, Redis	Документация Spring	2

№	Тема для самостоятельного изучения	Рекомендуемая литература	Трудоемкость (час.)
7	Микросервисная архитектура и Spring Cloud	Статьи на Habr	2
8	Сравнение СУБД: PostgreSQL vs MySQL vs MongoDB	Учебник «Базы данных»	2
9	Безопасность REST API: CORS, CSRF, Rate Limiting	Документация Spring Security	2
10	Тест-дизайн: классы эквивалентности, граничные значения	Бейзер Б. «Тестирование черного ящика»	2
Итого			20

3. МЕТОДИЧЕСКИЕ УКАЗАНИЯ ПО ВЫПОЛНЕНИЮ ЛАБОРАТОРНЫХ РАБОТ

3.1. Перечень лабораторных работ

В рамках дисциплины «Конструирование программного обеспечения» студент выполняет 9 лабораторных работ (Java-стек).

№	Название лабораторной работы	Формируемые компетенции

№	Название лабораторной работы	Формируемые компетенции
1	TDD с JUnit 5 и Mockito	ПК-3, ПК-4
2	Интеграционное тестирование: TestContainers + PostgreSQL	ПК-3, ПК-4
3	Нагрузочное тестирование: JMeter против Spring Boot API	ПК-4
4	Рефакторинг: устранение запахов кода в legacy-проекте	ПК-3, ПК-4
5	JWT + Spring Security + RBAC	ПК-3, ПК-10
6	OAuth2 (вход через GitHub/Google)	ПК-3, ПК-10
7	Устранение N+1 в Spring Data JPA	ПК-3
8	WebFlux + Reactor (сравнение с MVC)	ПК-3, ПК-8
9	GraphQL vs REST (сравнение)	ПК-3, ПК-8

3.2. Алгоритм выполнения лабораторной работы

Этап	Содержание
1	Получение индивидуального варианта у преподавателя.
2	Изучение теоретического материала по теме работы (лекции, литература, документация).
3	Разработка алгоритма и архитектуры решения.

Этап	Содержание
4	Написание кода в среде разработки (IntelliJ IDEA / VS Code).
5	Отладка и тестирование разработанного кода.
6	Оформление отчета по лабораторной работе (см. п. 3.3).
7	Загрузка исходного кода в репозиторий (GitHub/GitLab/GitFlic).
8	Защита лабораторной работы (собеседование с преподавателем, демонстрация кода).

3.3. Структура отчета по лабораторной работе

Отчет оформляется в текстовом редакторе (Word, LibreOffice) или в формате Markdown и должен содержать:

Титульный лист (с указанием дисциплины, темы работы, ФИО студента, группы, варианта).

Цель работы (формулируется в соответствии с методическими указаниями).

Постановка задачи (текст индивидуального задания).

Теоретическая часть (краткое изложение основных понятий и инструментов).

Практическая часть:

описание разработанных классов и методов;

листинги ключевых фрагментов кода;

описание тестов (для работ №1–3, 7–9);

скриншоты результатов выполнения.

Заключение (выводы о полученных навыках и решенных задачах).

Ответы на контрольные вопросы (письменно).

Список использованных источников.

Ссылка на репозиторий с исходным кодом.

3.4. Рекомендации по написанию кода

Соблюдать стандарты оформления кода (Java Code Conventions).

Использовать meaningful naming (осмысленные имена переменных, методов, классов).

Комментировать сложные алгоритмы и нетривиальные решения.

Писать модульные тесты (unit tests) для всех публичных методов.

Соблюдать принципы SOLID и паттерны проектирования.

Регулярно выполнять commit с осмысленными сообщениями.

Использовать .gitignore для исключения временных файлов и настроек среды.

3.5. Работа с Git и GitFlic

Рекомендуемый порядок работы с системой контроля версий:

Зарегистрироваться на платформе GitFlic (или GitHub/GitLab).

Создать приватный или публичный репозиторий для дисциплины.

Для каждой лабораторной работы создать отдельную ветку (feature/).

Выполнять commit после каждого завершенного этапа работы.

По завершении работы создать pull request и запросить code review у преподавателя.

После одобрения влить изменения в основную ветку (main).

4. ВОПРОСЫ ДЛЯ СОБЕСЕДОВАНИЯ ПО САМОСТОЯТЕЛЬНОМУ ИЗУЧЕНИЮ ЛИТЕРАТУРЫ

4.1. Общие вопросы по дисциплине

1. Что такое TDD (Test-Driven Development)? Опишите цикл «красный-зеленый-рефакторинг».
2. В чем разница между unit-тестом и интеграционным тестом?
3. Что такое mock-объект и для чего он используется (Mockito)?
4. Как TestContainers решает проблему изоляции тестовой базы данных?

5. Какие метрики производительности являются ключевыми при нагрузочном тестировании?
6. Что такое «запахи кода» (code smells)? Приведите 5 примеров.
7. Что такое рефакторинг и чем он отличается от переписывания кода?
8. Какие инструменты рефакторинга есть в IntelliJ IDEA?
9. Что такое технический долг (technical debt) и как его измерить?

4.2. Вопросы по безопасности (ПК-10)

1. Из каких частей состоит JWT (Header, Payload, Signature)?
2. В чем разница между аутентификацией и авторизацией?
3. Что такое RBAC (Role-Based Access Control)? Приведите пример.
4. Почему пароли нужно хешировать с помощью BCrypt, а не MD5 или SHA-1?
5. Как работает OAuth2? Чем он отличается от базовой аутентификации?
6. Что такое Access Token и Refresh Token?
7. Назовите 5 наиболее критичных уязвимостей OWASP Top 10.
8. Как защититься от SQL-инъекций в Spring Data JPA?
9. Как защититься от XSS (межсайтового скриптинга)?

4.3. Вопросы по ORM и базам данных

1. В чем заключается проблема N+1 запросов в JPA/Hibernate?
2. Какие способы решения проблемы N+1 существуют (@EntityGraph, JOIN FETCH)?
3. Что такое сессия Hibernate и кэш первого уровня?
4. В чем разница между FetchType.LAZY и FetchType.EAGER?
5. Что такое транзакция? Какие уровни изоляции существуют?

4.4. Вопросы по реактивному программированию

1. В чем разница между реактивным (WebFlux) и блокирующим (MVC) подходом?
2. Что такое Mono и Flux в Project Reactor?
3. Как работает backpressure?
4. Какие преимущества дает неблокирующий I/O (Netty) при высоких нагрузках?

4.5. Вопросы по GraphQL

1. Чем GraphQL отличается от REST по модели запросов?
2. Что такое DataLoader и какую проблему он решает?
3. Как определить схему GraphQL через SDL (Schema Definition Language)?

4.6. Вопросы по правовым нормам и ИБ (ПК-10)

4. Какие правовые нормы регулируют обработку персональных данных в РФ (ФЗ-152)?
5. Какие требования предъявляет ФЗ-149 «Об информации» к программному обеспечению?
6. Какие лицензии на ПО существуют (MIT, GPL, Apache, проприетарная)?
7. Что такое модель угроз информационной безопасности?
8. Какие требования предъявляет ФСТЭК к разработке программного обеспечения?

5. КРИТЕРИИ ОЦЕНИВАНИЯ КОМПЕТЕНЦИЙ

5.1. Оценка выполнения лабораторных работ

Оценка	Критерии
5 (отлично)	Работа выполнена в полном объеме, код соответствует заданию, тесты успешно проходят (покрытие ≥80%), оформление отчета аккуратное, студент дает полные ответы на контрольные вопросы.
4 (хорошо)	Работа выполнена в полном объеме, но есть незначительные замечания по оформлению кода или отчета, допущены неточности при ответах на контрольные вопросы.
3 (удовлетворительно)	Работа выполнена частично (не менее 70% требований), код содержит ошибки, тесты проходят не полностью, ответы на контрольные вопросы неполные.
2 (неудовлетворительно)	Работа не выполнена или выполнено менее 50% требований, код

Оценка	Критерии
	неработоспособен, студент не может ответить на контрольные вопросы.

5.2. Оценка самостоятельной работы

Вид СРС	Оцениваемые параметры	Вес в итоговой оценке
Выполнение лабораторных работ	Качество кода, полнота реализации, прохождение тестов	50%
Оформление отчетов	Полнота, аккуратность, соответствие требованиям	20%
Защита работ (собеседование)	Полнота ответов, понимание материала	20%
Работа с Git (репозиторий)	Регулярность commit, структура веток, качество сообщений	10%

5.3. Шкала перевода баллов в оценку

Процент выполнения	Баллы (100-балльная шкала)	Оценка (5-балльная шкала)
90–100%	90–100	5 (отлично)
75–89%	75–89	4 (хорошо)
60–74%	60–74	3 (удовлетворительно)

Процент выполнения	Баллы (100-балльная шкала)	Оценка (5-балльная шкала)
0–59%	0–59	2 (неудовлетворительно)