Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение

высшего образования

«Новгородский государственный университет имени Ярослава Мудрого»

ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ

ПОЛИТЕХНИЧЕСКИЙ КОЛЛЕДЖ

УТВЕРЖДАЮ

Зам. директора по УМ и ВР

Л.Н. Иванова

« » 2025 г.

Методология безопасной разработки

Пояснительная записка к дипломному проекту по специальности

09.02.07 Информационные системы и программирование

ПТК. ДП 1991 16. 000ПЗ

|  |  |
| --- | --- |
| Согласовано: |  |
| Консультант по спец. части  Л.Н. Цымбалюк  « » 2025 года  Консультант по экон. части  Г.В. Лебедева  « » 2025 года  Нормоконтроль  А.М. Чернега  « » 2025 года  Заместитель директора по УПР  А.М. Чернега  « » 2025 года | Руководитель  А.Е. Цымбалюк  « » 2025 года  Выполнил:  обучающийся группы 1991  М.Э. Нор  « » 2025 года |

Содержание

[Введение 3](#_Toc200291172)

[1 Общая часть 6](#_Toc200291173)

[1.1 Постановка задачи 6](#_Toc200291174)

[1.2 Обоснование проектных решений 20](#_Toc200291175)

[1.3 Обзор и анализ существующих программных систем 31](#_Toc200291176)

[2 Специальная, практическая, опытно-экспериментальная часть 36](#_Toc200291177)

[2.1 Анализ задачи 36](#_Toc200291178)

[2.2 Описание логической структуры 41](#_Toc200291179)

[2.3 Описание работы программы 53](#_Toc200291180)

[2.4 Руководство оператора 72](#_Toc200291181)

[3 Экономическая часть 82](#_Toc200291182)

[3.1 Расчёт заработной платы с отчислением на социальное страхование 82](#_Toc200291183)

[3.2 Расчёт стоимости материалов и лицензионного обеспечения 87](#_Toc200291184)

[3.3 Расчёт накладных расходов 89](#_Toc200291185)

[3.4 Составление и расчёт цены реализации программного продукта 90](#_Toc200291186)

[Заключение 92](#_Toc200291187)

[Методология 94](#_Toc200291188)

[Список литературы 97](#_Toc200291189)

[Приложение А 99](#_Toc200291190)

[Приложение Б 104](#_Toc200291191)

[Приложение В 106](#_Toc200291192)

[Приложение Г 108](#_Toc200291193)

# Введение

В условиях стремительного роста киберугроз и усложнения архитектуры программного обеспечения (ПО) обеспечение безопасности на этапе разработки становится не просто важным, а критически необходимым. Согласно данным OWASP (2023), более 80% уязвимостей веб-приложений возникают на уровне исходного кода, причём основными причинами являются ошибки валидации входных данных, некорректная обработка пользовательского ввода и использование уязвимых библиотек. Последствия таких недостатков могут быть разрушительными: от массовых утечек персональных данных и финансовых потерь до судебных исков и утраты доверия со стороны пользователей. При этом подавляющее большинство этих проблем предотвратимо, если внедрить строгую методологию безопасного программирования, ориентированную на минимизацию рисков ещё на этапе написания кода.

Целью исследования является разработка методологии безопасной разработки программного обеспечения, которая будет сочетать теоретические принципы защиты информации с практическими рекомендациями для реализации в реальных проектах. Для достижения этой цели были поставлены следующие задачи:

* Автоматическое создание документации : отображаются все доступные эндпоинты;
* провести анализ распространённых уязвимостей в исходном коде, включая SQL-инъекции, межсайтовый скриптинг (XSS), уязвимости в сторонних библиотеках и ошибки управления сессиями;
* сформировать свод правил безопасного программирования, охватывающий ключевые аспекты: валидацию данных, криптографические практики, обработку ошибок и управление зависимостями;
* реализовать демонстрационные примеры кода на языках Java (Spring Boot), иллюстрирующие применение предложенных правил в контексте создания REST API;
* оценить влияние методологии на качество кода, включая производительность, читаемость и устойчивость к атакам, с использованием инструментов статического анализа (SAST) и динамического тестирования (DAST).

Объектом исследования выбран процесс разработки программного обеспечения, охватывающий этапы написания кода, интеграции внешних библиотек и верификации безопасности. Предметом исследования выступают методы и практики предотвращения уязвимостей на уровне исходного кода, включая правила валидации данных, защиту конфиденциальной информации и автоматизированный мониторинг зависимостей.

Для решения задач использовались следующие методы:

* теоретический анализ стандартов безопасности (OWASP Secure Coding Practices, CERT Coding Standards, NIST SP 800-53), направленный на выявление универсальных принципов безопасной разработки;
* экспериментальная проверка предложенных правил через реализацию защищённых REST API на Java (Spring Boot) с акцентом на защиту от SQLi, XSS и CSRF-атак;
* сравнительный анализ существующих подходов, включая Microsoft SDL (Security Development Lifecycle) и OWASP ASVS (Application Security Verification Standard), для выявления их сильных и слабых сторон;
* тестирование кода с помощью инструментов SAST (SonarQube, OWASP Dependency-Check) и DAST (Burp Suite, OWASP ZAP) для количественной оценки эффективности методологии.

Научная новизна работы заключается в предложении комплексной методологии, объединяющей теоретические принципы безопасности с практическими примерами реализации на популярных языках программирования. В отличие от существующих подходов, методология фокусируется не только на общих рекомендациях, но и на конкретных паттернах кодирования, которые могут быть внедрены непосредственно на уровне исходного кода.

Практическая значимость исследования выражается в возможности использования разработанной методологии в реальных проектах для снижения рисков уязвимостей на ранних этапах разработки. Результаты работы могут быть применены как руководство для разработчиков, DevOps-команд и архитекторов систем, стремящихся повысить уровень безопасности своего ПО без значительного усложнения процессов.

Работа состоит из четырех частей:

* общая часть — анализ уязвимостей, существующих подходов и роли библиотек в безопасности;
* специальная часть — разработка методологии, примеры кода на Java, оценка их эффективности;
* заключение — выводы, рекомендации по внедрению и направления дальнейших исследований;
* приложения — полные листинги примеров, результаты тестирования и диаграммы.

# 1 Общая часть

## 1.1 Постановка задачи

### 1.1.1 Проблема

Современные программные системы сталкиваются с множеством угроз, возникающих на этапе разработки. По данным OWASP (2023), 80% уязвимостей связаны с ошибками в коде, такими как SQL-инъекции, XSS, CSRF, IDOR, использование уязвимых библиотек и некорректное управление секретами. Эти проблемы возникают как из-за человеческого фактора (например, недостаточная экспертиза разработчиков), так и из-за организационных и технических недочётов (недостатки в тестировании, отсутствие мониторинга зависимостей). Анализ показывает, что уязвимости часто являются следствием не только отдельных ошибок, но и системного отсутствия комплексного подхода к обеспечению безопасности на всех этапах жизненного цикла ПО.

SQL-инъекции, например, возникают из-за некорректной обработки пользовательского ввода. Атакующий может внедрить вредоносный SQL-запрос через поля формы или URL-параметры, что позволяет ему получить доступ к базе данных, изменить данные или даже удалить информацию. Примером служит передача строки ' OR '1'='1 в поле логина, которая может обойти проверку авторизации и предоставить доступ к системе без пароля. Подобные атаки подчёркивают важность использования параметризованных запросов и строгой валидации входных данных.

Межсайтовый скриптинг (XSS) представляет собой внедрение вредоносного JavaScript-кода через недостаточно защищённые формы или поля ввода. Например, отправка скрипта <script>alert('xss')</script> в поле комментария может привести к выполнению кода в браузере других пользователей, что открывает возможность кражи сессионных кук, перенаправлению на вредоносные сайты или сбору конфиденциальной информации. Для предотвращения XSS необходимо очищать входные данные от HTML-тегов и использовать механизмы экранирования при выводе информации.

CSRF (межсайтовая подделка запроса) позволяет злоумышленнику выполнять действия от имени пользователя без его ведома через поддельные формы или ссылки. Например, пользователь, авторизованный в банковском приложении, может случайно перейти по ссылке, которая отправляет запрос на перевод средств с его счета. Защита от CSRF включает использование токенов проверки (CSRF tokens) и проверку заголовков SameSite в куках.

IDOR (прямой несанкционированный доступ к объектам) возникает при манипуляциях с идентификаторами ресурсов. Например, изменение URL с /user/1 на /user/2 может позволить пользователю получить доступ к чужому профилю, если система не проверяет права доступа. Для предотвращения IDOR требуется строгая проверка прав на уровне бизнес-логики и использование несуществующих идентификаторов (например, UUID вместо последовательных чисел).

Уязвимости в зависимостях — ещё одна распространённая проблема, связанная с использованием библиотек с известными проблемами. Например, уязвимость log4j в Java позволяла удалённое выполнение кода через логирование пользовательских данных. Это подчёркивает необходимость регулярного обновления зависимостей и использования инструментов вроде OWASP Dependency-Check для анализа безопасности сторонних модулей.

Некорректное управление секретами, такое как хранение паролей или токенов в открытом виде в коде или конфигурационных файлах, также является критическим риском. Например, жёстко зашитые API-ключи в исходном коде могут быть раскрыты через утечки репозиториев, что приведёт к компрометации систем. Для минимизации рисков рекомендуется использовать переменные окружения, специализированные менеджеры секретов (например, HashiCorp Vault) и шифрование конфиденциальной информации.

Эти проблемы приводят к утечкам данных, финансовым потерям и юридическим последствиям. Например, утечка персональных данных через SQL-инъекции может привести к штрафам по GDPR в размере до 4% от годового оборота компании. По данным Verizon DBIR (2023), 75% атак начинаются с эксплуатации уязвимостей на уровне исходного кода, что подчеркивает необходимость внедрения безопасной разработки на ранних этапах. Кроме того, репутационные потери и потеря доверия клиентов могут нанести урон компании не менее серьёзный, чем прямые финансовые издержки.

С ростом сложности программного обеспечения и увеличением числа подключаемых библиотек, проблема безопасности становится ещё более актуальной. Автоматизация разработки, внедрение CI/CD и активное использование open-source решений создают новые точки проникновения для злоумышленников. Это требует системного подхода к обеспечению безопасности, включающего обучение разработчиков, автоматизированный анализ кода и внедрение стандартов безопасного программирования на всех этапах разработки.

### 1.1.2 Цель и задачи

Понимание причин возникновения уязвимостей в коде позволяет выявить системные слабые места в процессе разработки программного обеспечения. Основные источники рисков связаны как с человеческим фактором (например, недостаточная экспертиза разработчиков), так и с организационными и техническими аспектами (недостатки в тестировании, использование уязвимых библиотек). Анализ этих причин демонстрирует, что уязвимости часто являются следствием не только отдельных ошибок, но и отсутствия комплексного подхода к обеспечению безопасности на всех этапах жизненного цикла ПО.

Уязвимости в коде возникают по следующим причинам:

* ошибка разработчик:

1. недостаточная экспертиза в области безопасности;
2. использование устаревших или небезопасных практик (например, динамическое исполнение кода без проверок).

* недостатки в процессе разработки:

1. отсутствие этапов тестирования на безопасность (Без регулярной проверки кода на уязвимости (например, через SAST/DAST) многие ошибки остаются незамеченными до поздних стадий разработки или даже после запуска. Это увеличивает риск эксплуатации уязвимостей злоумышленниками);
2. несоблюдение стандартов безопасности (например, OWASP Secure Coding Practices) приводит к использованию небезопасных методов, таких как динамическое исполнение кода или слабая валидация данных. Это создает предпосылки для атак, которые можно было бы избежать при следовании проверенным рекомендациям.

* проблемы с зависимостями:

1. использование библиотек без анализа их безопасности;
2. задержки в обновлении компонентов после публикации уязвимостей;
3. использование несуществующих зависимостей (Злоумышленники могут создать библиотеку и загрузить в нее вредоносное ПО, а разработчик укажет неверное название, и скачает ее. Частая проблема при использовании нейросетей для генерации кода).

* архитектурные ошибки:

1. неправильная реализация механизмов аутентификации и шифрования (Ошибки в выборе алгоритмов (например, использование MD5 для хранения паролей) или некорректная настройка протоколов (например, HTTP вместо HTTPS) делают систему уязвимой для перехвата данных или подделки токенов);
2. недостаточная изоляция компонентов системы (Отсутствие четкого разделения зон ответственности между модулями (например, фронтенд и бэкенд) позволяет атакующему использовать уязвимость в одном компоненте для доступа к критическим данным всей системы.).

Таким образом, источники уязвимостей в коде многообразны и охватывают как технические аспекты (например, использование слабых алгоритмов или устаревших библиотек), так и организационные проблемы (недостатки в процессах разработки и обучении). Выявление этих причин позволяет сформулировать ключевые требования к методологии безопасной разработки, направленной на устранение системных рисков. Полученные выводы будут использованы в специальной части работы для разработки практических рекомендаций, направленных на минимизацию уязвимостей на этапе написания кода.

### 1.1.3 Обоснование необходимости разработки

В условиях цифровой трансформации общества и стремительного роста числа онлайн-сервисов, безопасность программного обеспечения становится одной из ключевых задач при разработке любого продукта. Уязвимости в коде могут привести к утечкам персональных данных, финансовым потерям, нарушению целостности информации и снижению доверия со стороны пользователей. В связи с этим всё большее значение приобретает методология безопасной разработки ПО, которая подразумевает не просто реакцию на уже выявленные угрозы, но проактивное внедрение защитных мер на всех этапах жизненного цикла разработки.

Согласно данным OWASP (Open Web Application Security Project), основными уязвимостями веб-приложений остаются:

* неправильная реализация аутентификации и управления сессией;
* недостаточная защита данных;
* отсутствие проверок входных данных (например, SQL-инъекции, XSS);
* некорректная обработка ошибок;
* несоблюдение политики доступа;
* использование устаревших или небезопасных библиотек.

Эти проблемы показывают, что даже опытные разработчики могут допустить ошибки, которые в будущем станут причинами серьёзных инцидентов. Поэтому необходима систематизация знаний и формирование чёткой методологии безопасной разработки, которая поможет минимизировать такие риски ещё на этапе написания кода.

Настоящий дипломный проект направлен на разработку REST API с использованием языка Java и фреймворка Spring Boot, реализованного с соблюдением принципов безопасной разработки. Основной задачей является демонстрация того, как можно применять практики информационной безопасности в реальных условиях разработки программного обеспечения, чтобы предотвратить наиболее распространённые виды атак и повысить надёжность системы.

Проект предназначен для следующих категорий пользователей:

* разработчики программного обеспечения — особенно те, кто только начинает осваивать вопросы безопасности в коде. Проект может служить примером того, как правильно организовать аутентификацию, авторизацию, валидацию данных и работу с чувствительной информацией;
* студенты и преподаватели технических вузов — работа может быть использована как учебное пособие или практическое руководство по применению безопасных практик в разработке ПО;
* технические специалисты, занимающиеся тестированием и аудитом безопасности — результаты проекта помогут лучше понимать, какие меры защиты должны быть внедрены на уровне исходного кода;
* предприниматели и заказчики ПО — проект демонстрирует, что безопасность не должна рассматриваться как отдельный этап после разработки, а должна быть частью самого процесса создания программного обеспечения.

Данная работа имеет как научную, так и практическую ценность. С научной точки зрения, она способствует развитию методологий безопасной разработки программного обеспечения и позволяет структурировать существующие подходы в виде свода правил, применимых в реальной практике. С практической стороны, представленное REST API может быть адаптировано и использовано как прототип для более сложных систем, таких как интернет-магазины, CRM-системы или платформы электронного обучения.

Таким образом, разработка данного проекта оправдана с точки зрения повышения уровня безопасности современных веб-приложений и формирования у разработчиков культуры безопасного программирования

### 1.1.4 Технико-математическое описание задачи

Для реализации информационной системы в виде безопасного REST API используется современный стек технологий, ориентированный на надёжность, масштабируемость и высокую степень защиты данных. Основным языком программирования выбран Java, благодаря своей популярности в enterprise-разработке, широкой экосистеме и встроенной поддержке безопасности.

Серверная часть приложения построена на основе фреймворка Spring Boot, что позволило быстро создать архитектуру, соответствующую принципам RESTful API. Spring предоставляет мощные инструменты для управления зависимостями, маршрутизации запросов, работы с базами данных и обеспечения безопасности, что делает его идеальным выбором для данного проекта.

Для взаимодействия с базой данных используется ORM-библиотека Hibernate, интегрированная через модуль Spring Data JPA. Такой подход позволяет работать с объектами, не используя SQL напрямую, что упрощает разработку и снижает риск SQL-инъекций

Для управления базой данных используется система PostgreSQL — стабильная и производительная СУБД, поддерживающая сложные реляционные структуры и запросы. Она обеспечивает надёжное хранение информации о нейросетях, их конфигурациях, статусах и результатах работы. Связь между Java-приложением и базой данных реализуется средствами Spring Data JPA, что упрощает работу с базой.

Одним из ключевых элементов технической реализации является использование Spring Security — самого популярного фреймворка для обеспечения безопасности в Java-приложениях. С его помощью реализованы механизмы:

* аутентификации и авторизации пользователей;
* разграничения ролей (ADMIN, EMPLOYEE, CLIENT);
* защита от CSRF-атак;
* ограничение доступа к определённым эндпоинтам.

Для stateless-архитектуры используется JWT (JSON Web Token), который позволяет передавать данные о пользователе в зашифрованном виде между клиентом и сервером без необходимости хранения сессий на сервере. Это повышает уровень безопасности и упрощает масштабирование приложения. Генерация, проверка и обработка токенов реализована с помощью библиотеки io.jsonwebtoken, а также собственного компонента JwtUtils

Для безопасного хранения пользовательских паролей применяется алгоритм BCrypt, реализованный через класс BCryptPasswordEncoder. Этот алгоритм считается одним из самых безопасных на сегодняшний день, поскольку он включает в себя механизм "соления", предотвращающий использование Rainbow-таблиц при взломе.

Для предотвращения ошибок и потенциальных уязвимостей (например, некорректных данных) используется встроенная валидация с помощью аннотаций JSR-380 (@NotBlank, @Positive и др.). Это позволяет ещё на уровне контроллера проверять корректность входных данных и возвращать понятные сообщения об ошибках.

Для обеспечения качества кода и возможности дальнейшего развития приложения реализованы unit-тесты с использованием фреймворков JUnit 5 и Mockito. Тестирование проводится на уровне сервисов и контроллеров, что позволяет проверить логику работы системы без зависимости от внешних ресурсов.

Для реализации проекта используются следующие программные средства:

* IntelliJ IDEA — основная IDE для разработки, отладки и тестирования Java-приложения;
* Maven — система управления зависимостями, обеспечивающая простое подключение необходимых библиотек;
* Docker — хотя в текущей версии проекта контейнеризация не реализована, она может быть добавлена для упрощения деплоя и изоляции сервисов;
* Postman — инструмент для тестирования HTTP-запросов к API.
* Swagger UI (опционально) — может быть интегрирован для документирования и тестирования REST API;
* pgAdmin – графическая утилита для администрирования базы данных PostgreSQL.

Выбранный стек технологий и методов разработки позволяет создать легковесное, безопасное и расширяемое REST API, соответствующее современным требованиям к веб-приложениям. Использование Spring Security, JWT, Hibernate, BCrypt и других компонентов обеспечивает защиту данных, гибкость архитектуры и простоту сопровождения, что делает систему пригодной как для учебных целей, так и для использования в реальных условиях.

### 1.1.5 Характеристика бизнес-процессов

Информационная система предназначена для демонстрации методологии безопасной разработки программного обеспечения на примере реализации простого REST API. Основное назначение системы — показать, как можно организовать регистрацию, авторизацию, работу с данными и ограничение доступа в рамках веб-приложения, соблюдая современные стандарты безопасности. Система может быть использована как шаблон для построения более сложных решений, таких как интернет-магазины или сервисы управления продуктами, где важна защита информации и контроль прав пользователей.

Для описания бизнес-процессов используется методология IDEF0, которая позволяет формализовать функциональные зависимости и показать иерархию взаимодействий между основными компонентами системы. На рисунке А.1 представлена IDEF0-диаграмма, отражающая общий процесс управления пользователями и товарами. Рисунок А.2 содержит детализацию этих процессов, включая этапы регистрации, аутентификации, получения списка товаров и добавления новых записей. Диаграмма прецедентов, представленная на рисунке А.3, демонстрирует варианты использования системы различными категориями пользователей.

Первый ключевой процесс — это регистрация нового пользователя. Пользователь предоставляет свои данные: email, пароль, имя и роль (ADMIN, EMPLOYEE или CLIENT). При этом проводится проверка уникальности email, а также корректности введённых данных. Пароль перед сохранением хэшируется с помощью алгоритма BCrypt, что обеспечивает высокую степень защиты конфиденциальной информации.

Второй важный процесс — авторизация и выдача JWT-токена. После успешной регистрации пользователь может войти в систему, указав email и пароль. Сервер проверяет учетные данные, и, если они корректны, генерируется JWT-токен. Этот токен содержит информацию о пользователе: его ID, роль и срок действия. Токен должен быть передан в заголовке Authorization при каждом последующем запросе к защищённым ресурсам.

Третьим значимым процессом является получение списка товаров. Авторизованные пользователи с ролью ADMIN или EMPLOYEE могут получить список всех доступных товаров. Доступ к данным контролируется через Spring Security, а также аннотации, такие как @PreAuthorize. Ответ системы возвращается в формате JSON, что делает его удобным для дальнейшей обработки клиентскими приложениями.

Четвёртый процесс — добавление нового товара. Пользователи с ролью ADMIN или EMPLOYEE имеют возможность добавлять новые товары в систему. При этом обязательными являются следующие поля: название товара и его цена. Название не должно быть пустым, а цена должна быть положительным числом. Перед сохранением данных в базу проводится строгая валидация входных данных с использованием JSR-380 (@NotBlank, @Positive), что исключает возможность некорректной записи информации.

Система реализует несколько ключевых функций, направленных на обеспечение безопасности и устойчивости работы. Она использует JWT для stateless-аутентификации, что повышает масштабируемость и упрощает распределённую архитектуру. Для безопасного хранения паролей применяется алгоритм BCrypt, который считается одним из самых надёжных на сегодняшний день. Также реализована защита от CSRF-атак, включены заголовки безопасности, такие как Content-Security-Policy и Cache-Control, а также выполнена строгая валидация входных данных на уровне контроллеров.

Все эти меры обеспечивают защиту от распространённых уязвимостей, таких как SQL-инъекции, XSS-атаки и подделка межсайтовых запросов. Кроме того, реализовано ограничение доступа по ролям, что позволяет контролировать, какие действия может выполнять тот или иной пользователь.

Система предполагает наличие нескольких ролей пользователей:

* CLIENT — в текущей версии эта роль не предоставляет никаких действий, но может быть расширена в будущем;
* EMPLOYEE — имеет возможность просматривать список товаров и добавлять новые;
* ADMIN — обладает теми же правами, что и EMPLOYEE, и может использоваться для расширения системы в дальнейшем.

Взаимодействие с системой осуществляется через HTTP-запросы к REST API, которые можно выполнять с помощью таких инструментов, как Postman, curl или HTTP-клиенты из мобильных и веб-приложений.

При проектировании и разработке системы были применены современные подходы:

* модульная архитектура, основанная на разделении на слои: контроллеры, сервисы, репозитории;
* безопасная разработка, включающая использование JWT, шифрование паролей, валидацию данных и ограничение прав доступа;
* обработка ошибок, предусматривающая возврат понятных сообщений об ошибках и корректную обработку исключений;
* тестирование, выполненное с помощью JUnit и Mockito, чтобы убедиться в надёжности и корректности работы каждого компонента системы.

Таким образом, система охватывает ключевые процессы регистрации, авторизации, работы с данными и управления правами доступа. Все операции реализованы с соблюдением принципов безопасной разработки, что делает данное решение пригодным как для учебных целей, так и для практического применения. Представленная архитектура обеспечивает высокую степень защищённости, читаемость кода и удобство сопровождения, что особенно важно при создании реальных коммерческих продуктов

### 1.1.6 Требования к программе

Для успешной реализации проекта, направленного на демонстрацию методологии безопасной разработки программного обеспечения, были определены следующие требования к программе. Они включают технические средства, используемые фреймворки и библиотеки, функциональные возможности, а также дополнительные аспекты, обеспечивающие надёжность, безопасность и удобство использования системы.

Технические средства:

* серверное оборудование – минимальные системные требования позволяют запускать приложение на локальной машине или VPS, поскольку система представляет собой легковесный REST API сервис;
* поддержка контейнеризации (Docker) – хотя в текущей версии проекта не используется, предусмотрена возможность дальнейшего внедрения Docker для упрощения деплоя и изоляции компонентов;
* интеграция с СУБД – поддерживается работа с H2 (встроенной базой данных) для тестирования и PostgreSQL для продакшена.

Фреймворки и библиотеки:

* Spring Boot – основной фреймворк для построения REST API и управления зависимостями;
* Spring Security – используется для реализации механизма аутентификации, авторизации и ограничения доступа по ролям;
* JWT (JSON Web Token) – применяется для stateless-аутентификации, что позволяет исключить хранение сессий на сервере;
* BCryptPasswordEncoder – используется для безопасного хэширования паролей перед сохранением в БД;
* Hibernate / Spring Data JPA – ORM-библиотека для работы с базами данных и маппинга объектов в таблицы;
* Lombok – упрощает написание кода за счёт автоматической генерации геттеров, сеттеров и других элементов;
* JUnit 5 + Mockito – инструменты модульного тестирования для проверки контроллеров и сервисов;
* Jakarta Validation (JSR-380) – используется для валидации входных данных на уровне контроллеров.

Требования к функциональности:

* регистрация пользователей – возможность создания новых учётных записей с указанием email, пароля, имени и роли (ADMIN, EMPLOYEE, CLIENT);
* авторизация и выдача JWT-токена – после регистрации пользователь может войти в систему и получить токен, содержащий его ID и роль;
* получение списка товаров – авторизованные пользователи с ролью ADMIN или EMPLOYEE могут просматривать список всех товаров;
* добавление нового товара – пользователи с соответствующими правами могут добавлять новые товары в систему, проходя строгую валидацию данных;
* ограничение доступа по ролям – защита эндпоинтов с помощью аннотаций @PreAuthorize и конфигураций Spring Security;
* валидация входных данных – обязательные поля проверяются на корректность с использованием JSR-380 (@NotBlank, @Positive);
* обработка ошибок – понятные сообщения об ошибках возвращаются клиенту в случае некорректного запроса или исключения в системе;
* шифрование паролей – использование BCrypt для безопасного хранения паролей;
* защита от CSRF-атак – отключение CSRF защиты в Spring Security, так как используется stateless-архитектура;
* заголовки безопасности – добавлены заголовки Content-Security-Policy, Cache-Control и Pragma для повышения уровня безопасности HTTP-ответов.

Дополнительные требования:

* обеспечение безопасности данных – реализация механизмов шифрования, валидации, ограничения доступа и защиты от распространённых уязвимостей (XSS, SQL-инъекции, CSRF);
* модульное тестирование – покрытие бизнес-логики unit-тестами с использованием JUnit и Mockito;
* логирование действий – использование @Slf4j для логирования событий, таких как регистрация, вход и операции с товарами;
* масштабируемость – применение stateless-архитектуры с JWT позволяет легко масштабировать систему при увеличении нагрузки;
* гибкость и расширяемость – система спроектирована таким образом, чтобы её можно было легко расширять, например, добавлять новые сущности, роли, функции и интерфейсы;
* интеграция с внешними системами – API разработан в соответствии со стандартами RESTful, что позволяет использовать его как backend для мобильных и веб-приложений;
* документирование API – возможность интеграции Swagger UI для автоматического документирования и тестирования эндпоинтов.

## 1.2 Обоснование проектных решений

### 1.2.1 Обоснование выбора языков программирования

В рамках реализации дипломного проекта в качестве основного языка программирования был выбран Java . Выбор этого языка обусловлен его широким применением в enterprise-разработке, наличием мощной экосистемы, строгой типизацией, автоматическим управлением памятью и высоким уровнем поддержки безопасности.

Основные преимущества Java для данного проекта:

* Java является одним из самых популярных языков программирования в мире, особенно в области разработки серверных приложений. Согласно многочисленным рейтингам (TIOBE Index, Stack Overflow Survey), Java стабильно занимает одни из лидирующих позиций среди языков, используемых в профессиональной среде. Это делает её привлекательной для обучения, исследования и практического применения.
* Java предоставляет мощные средства работы с потоками, что позволяет эффективно использовать ресурсы процессора при обработке множества параллельных HTTP-запросов. Для REST API это особенно важно, поскольку система должна уметь обрабатывать множество запросов от разных клиентов одновременно.
* Строгая типизация в Java помогает находить ошибки ещё на этапе компиляции, что повышает надёжность кода и снижает риск возникновения ошибок во время выполнения. Это особенно ценно при разработке безопасных систем, где некорректное поведение может привести к утечкам данных или другим уязвимостям.
* Механизм сборки мусора (Garbage Collection) позволяет разработчику не заботиться о ручном освобождении памяти, что снижает вероятность возникновения утечек памяти и других низкоуровневых проблем. Это особенно важно при длительной работе серверного приложения.
* Java имеет огромное количество библиотек, фреймворков и инструментов, которые значительно упрощают разработку сложных систем. Особенно важными для данного проекта являются:

1. Spring Framework — для построения REST API и обеспечения безопасности.
2. Hibernate / Spring Data JPA — для работы с базой данных.
3. BCryptPasswordEncoder — для безопасного хэширования паролей.
4. JWT (JSON Web Token) — для stateless-аутентификации.
5. JUnit и Mockito — для модульного тестирования.

* Java активно используется в корпоративной сфере, особенно там, где важны масштабируемость, производительность и безопасность. Многие крупные компании используют Java для создания критически важных систем, что говорит о её надёжности и долгосрочной перспективе.

Использование Spring Boot

Для реализации REST API был выбран фреймворк Spring Boot , который представляет собой расширение Spring Framework и позволяет быстро создавать автономные, готовые к запуску приложения без необходимости ручной настройки контейнеров сервлетов или XML-конфигураций.

Причины выбора Spring Boot:

* Встроенный сервер : Не требуется внешний сервер (например, Tomcat) — Spring Boot поставляется с embedded-сервером;
* Поддержка RESTful API : Удобные аннотации (@RestController, @RequestMapping) позволяют легко создавать веб-сервисы;
* Интеграция с Security : Легко подключается Spring Security для реализации аутентификации и авторизации;
* Простота тестирования : Хорошая совместимость с JUnit и Mockito;
* Логирование и мониторинг : Возможность использования Logback, SLF4J и Actuator для наблюдения за состоянием системы.

Таким образом, Java в сочетании с Spring Boot была выбрана как наиболее подходящий язык для реализации безопасного REST API, соответствующего современным требованиям к разработке программного обеспечения.

### 1.2.2 Инструментальные средства

Для реализации дипломного проекта, представляющего собой REST API с акцентом на безопасную разработку, были выбраны следующие инструментальные средства. Каждый из выбранных компонентов обладает определёнными преимуществами, которые позволили создать надёжное, защищённое и расширяемое приложение, соответствующее современным стандартам веб-разработки.

Java и Spring Boot для создания REST API

Spring Boot был выбран как основной фреймворк для построения серверной части приложения. Он предоставляет мощные механизмы для быстрой разработки, автоматической конфигурации и масштабируемости:

* простота настройки : благодаря автоконфигурации и встроенному серверу, не требуется ручная настройка контейнеров сервлетов;
* поддержка RESTful API : удобные аннотации (@RestController, @RequestMapping) позволяют легко строить эндпоинты;
* интеграция с Security : лёгкая интеграция Spring Security обеспечивает защиту от распространённых уязвимостей;
* тестирование : хорошая совместимость с JUnit и Mockito позволяет проводить модульное тестирование контроллеров и сервисов.

Spring Security для обеспечения безопасности

Spring Security используется для реализации механизмов аутентификации, авторизации и ограничения доступа к ресурсам:

* аутентификация и авторизация : система проверяет учетные данные пользователя и выдаёт ему роль;
* ограничение доступа по ролям : с помощью аннотаций @PreAuthorize реализован контроль доступа к эндпоинтам;
* защита от CSRF-атак : в данном случае отключена, так как используется stateless-архитектура с JWT;
* заголовки безопасности : добавлены заголовки Content-Security-Policy, Cache-Control, Pragma для повышения уровня защиты HTTP-ответов.

JWT (JSON Web Token) для stateless-аутентификации

JWT применяется для передачи данных между клиентом и сервером в зашифрованном виде без необходимости хранения сессий на сервере:

* Stateless-архитектура : токены не требуют хранения состояния на сервере, что упрощает масштабирование;
* безопасность передачи данных : информация о пользователе шифруется и подписывается секретным ключом;
* гибкость использования : JWT может быть легко интегрирован в мобильные и веб-приложения.

BCryptPasswordEncoder для безопасного хэширования паролей

Для безопасного хранения паролей используется алгоритм BCrypt, реализованный через класс BCryptPasswordEncoder:

* соление : предотвращает использование Rainbow-таблиц;
* надежность : BCrypt считается одним из самых безопасных способов хэширования паролей;
* интеграция : легко подключается через Spring Security и работает "из коробки".

Hibernate / Spring Data JPA для работы с базой данных

Hibernate используется как ORM-библиотека для взаимодействия с реляционной базой данных:

* маппинг объектов в таблицы : позволяет работать с данными как с объектами, исключая необходимость написания SQL-запросов;
* поддержка валидации : автоматическая проверка данных перед сохранением;
* интеграция с Spring Boot : простая настройка и использование репозиториев.

H2/PostgreSQL для хранения данных

Для хранения информации о пользователях и товарах используется реляционная СУБД;

* H2 – встроенная БД для тестирования и демонстрации функционала;
* PostgreSQL – возможна замена для продакшена, обеспечивает стабильность и производительность;
* безопасность хранения : только хэши паролей хранятся в БД, что минимизирует риск утечки данных.

Lombok для упрощения кода

Lombok используется для автоматической генерации boilerplate-кода (геттеров, сеттеров, toString, equals и т.д.):

* Повышает читаемость кода : уменьшает объём повторяющегося кода;
* Ускоряет разработку : экономит время на написание вспомогательных методов;
* Лёгкая интеграция : поддерживается большинством IDE, включая IntelliJ IDEA.

JUnit 5 и Mockito для модульного тестирования

Для проверки работоспособности сервисов и контроллеров используются библиотеки JUnit и Mockito:

* Проверка бизнес-логики : тестирование методов сервисов без зависимости от внешних систем;
* Мокирование зависимостей : возможность имитировать поведение компонентов;
* Обеспечение качества кода : тесты помогают находить ошибки на раннем этапе.

IntelliJ IDEA как среда разработки

В качестве основной IDE используется IntelliJ IDEA Community Edition:

* Поддержка Spring и Java : отличная интеграция с фреймворками, автодополнение, диагностика ошибок;
* Отладка и профилирование : удобство анализа выполнения кода;
* Git-интеграция : управление версиями и командная разработка.

Postman для тестирования API

Postman используется для тестирования HTTP-запросов к API:

* Интерфейс для проверки эндпоинтов : удобно отправлять запросы и проверять ответы;
* Автоматизация тестов : можно писать скрипты для проверки логики;
* Интеграция с CI/CD – возможно использовать в составе автоматизированных процессов.

Swagger UI

* Хотя в текущей реализации Swagger не подключен, его можно интегрировать для документирования API и тестирования эндпоинтов:
* Автоматическое создание документации : отображаются все доступные эндпоинты;
* Интерактивное тестирование : можно выполнять запросы прямо из интерфейса.

Выбор указанных инструментальных средств обоснован их популярностью, стабильностью, развитой экосистемой и высоким уровнем поддержки сообщества. Использование этих технологий позволило реализовать безопасное, модульное и расширяемое REST API, которое может служить примером применения методологии безопасной разработки программного обеспечения.

Таким образом, применение перечисленных технологий обеспечивает надёжную основу для построения безопасного веб-сервиса, способного эффективно решать задачи регистрации, авторизации, управления правами и работы с данными в различных условиях эксплуатации.

### 1.2.3 Обоснование выбора среды программирования

Для реализации дипломного проекта в качестве основной интегрированной среды разработки (IDE) была выбрана IntelliJ IDEA , что обусловлено её высокой функциональностью, удобством работы и полной совместимостью с используемыми технологиями, такими как Java , Spring Boot и Spring Security . Это позволило эффективно реализовать серверную часть приложения с акцентом на безопасную разработку.

Основные преимущества IntelliJ IDEA для данного проекта:

* IntelliJ IDEA предоставляет мощные инструменты для работы с фреймворками Spring и Spring Boot:

1. автоматическое создание проектов через Spring Initializr ;
2. поддержка автодополнения, рефакторинга и диагностики ошибок;
3. упрощённая настройка конфигураций Spring Security;
4. возможность запуска и тестирования REST API прямо из IDE без дополнительных действий;
5. эти возможности значительно ускорили процесс разработки и помогли сосредоточиться на реализации логики безопасности.

* проект использует систему управления зависимостями Maven , которая позволяет легко подключать библиотеки, такие как Spring Security , JWT , Lombok , Hibernate и другие. IntelliJ IDEA обеспечивает:

1. удобное редактирование pom.xml;
2. автозагрузку зависимостей;
3. контроль версий и предупреждения о возможных конфликтах;
4. такая интеграция сделала работу с проектом более стабильной и простой.

* благодаря встроенной поддержке Lombok , IDE корректно распознаёт аннотации, такие как @Data, @AllArgsConstructor и другие, что позволило:

1. сократить объём повторяющегося кода;
2. повысить читаемость и поддерживаемость проекта;
3. избежать необходимости писать вручную геттеры, сеттеры, toString и другие методы.

* IntelliJ IDEA имеет мощные встроенные средства для работы с Git:

1. отслеживание изменений;
2. работа с ветками;
3. интеграция с GitHub/GitLab;
4. визуальное сравнение файлов и откат изменений;
5. это дало возможность контролировать изменения в коде, сохранять историю разработки и работать над проектом в командном режиме.

* для реализации безопасной разработки важно не только правильно написать код, но и протестировать его поведение. IntelliJ IDEA предоставляет:

1. пошаговую отладку с точками останова;
2. просмотр значений переменных и стека вызовов;
3. интеграцию с JUnit для выполнения и анализа тестов;
4. эти инструменты помогли выявлять ошибки ещё на этапе разработки и повышать качество кода.

* IntelliJ IDEA содержит встроенную утилиту HTTP Client , которая позволяет:

1. отправлять HTTP-запросы к эндпоинтам;
2. тестировать REST API без использования сторонних инструментов;
3. сохранять запросы и использовать переменные окружения;
4. это сделало процесс тестирования API более удобным и интегрированным в рабочий процесс.

* хотя в проекте используется встроенная БД H2, IntelliJ IDEA также поддерживает:

1. подключение к PostgreSQL, MySQL и другим СУБД;
2. SQL-автодополнение и проверку синтаксиса;
3. просмотр таблиц и выполнение запросов прямо из IDE;
4. это открывает возможность дальнейшего масштабирования системы.

Использование IntelliJ IDEA в качестве основной среды разработки позволило реализовать проект с высоким уровнем качества и безопасности. Её мощные функции по работе с Java, Spring, Git, тестированию и отладке, а также интеграция с REST API и системами контроля версий сделали процесс разработки максимально эффективным и комфортным. Таким образом, выбор данной IDE полностью соответствует целям и задачам дипломного проекта, связанным с демонстрацией принципов безопасной разработки программного обеспечения.

### 1.2.4 Информационное обеспечение

Для реализации дипломного проекта, представляющего собой REST API с акцентом на безопасную разработку, используется набор программных средств и инструментов, обеспечивающих эффективное хранение данных, управление кодом, тестирование и документирование. Выбранные технологии позволяют создать надёжную, защищённую и расширяемую систему, соответствующую современным требованиям к веб-приложениям.

H2/PostgreSQL – система управления базами данных

В качестве основной СУБД используется H2 — легковесная встроенная реляционная база данных, подходящая для тестирования и демонстрации функционала. Также предусмотрена возможность замены H2 на PostgreSQL для использования в продакшне:

* обеспечивает надёжное хранение информации о пользователях и товарах;
* поддерживает SQL-запросы и транзакции;
* удобна для работы с ORM (Hibernate / Spring Data JPA);
* в случае перехода на PostgreSQL — обеспечивается масштабируемость и высокая производительность.

Git – система контроля версий

Для управления исходным кодом используется Git , что позволяет:

* Отслеживать изменения в коде;
* Организовать командную разработку;
* Сохранять историю изменений;
* Интегрироваться с платформами, такими как GitHub или GitLab.

Git обеспечивает полный контроль над состоянием проекта на всех этапах его жизненного цикла.

IntelliJ IDEA – интегрированная среда разработки

В качестве основной IDE используется IntelliJ IDEA , которая предоставляет:

* автоматическую загрузку зависимостей через Maven;
* интеграцию с Git и системами контроля версий;
* поддержку Spring Boot, Lombok, Hibernate и других используемых технологий;
* мощные инструменты отладки и анализа кода.

Эти возможности делают IntelliJ IDEA идеальным выбором для создания и сопровождения безопасного Java-приложения.

JUnit 5 и Mockito – инструменты модульного тестирования

Для проверки корректности работы контроллеров и сервисов используются библиотеки:

* JUnit 5 – для написания и запуска тестовых сценариев;
* Mockito – для мокирования зависимостей и изоляции логики при тестировании.

Тестирование позволяет выявлять ошибки на ранних этапах разработки и повышает качество кода.

Postman – инструмент для тестирования REST API

Для тестирования HTTP-запросов к эндпоинтам используется Postman :

* позволяет отправлять GET, POST и другие запросы;
* обеспечивает удобное тестирование JWT-авторизации;
* используется для проверки работы продуктового API.

Это делает процесс отладки более наглядным и упрощает взаимодействие с backend-частью.

Swagger UI

Хотя в текущей реализации Swagger не подключен, он может быть использован в дальнейшем для:

* автоматического создания документации по API;
* интерактивного тестирования эндпоинтов;
* упрощения взаимодействия между frontend и backend.

Swagger особенно полезен при расширении системы и добавлении новых компонентов.

Draw.io – инструмент для проектирования архитектуры

Для визуализации структуры проекта и бизнес-процессов применяется онлайн-сервис Draw.io:

* создание IDEF0-диаграмм;
* проектирование ER-моделей БД;
* формирование диаграмм прецедентов и последовательностей.

Эти инструменты помогают лучше понять логику работы системы и подготовить качественную техническую документацию.

Microsoft Office – офисный пакет для документирования

Для подготовки технической документации, презентаций и отчётов используется Microsoft Office :

* Microsoft Word — для оформления пояснительной записки;
* Excel — для составления таблиц требований и характеристик;
* PowerPoint — для создания презентации к защите диплома.

Офисный пакет обеспечивает структурированное представление материалов и их наглядность при защите проекта.

Выбор указанных инструментов обусловлен их популярностью, открытой документацией, развитой экосистемой и высоким уровнем поддержки сообщества. Все перечисленные средства обеспечивают полное покрытие задач, связанных с проектированием, реализацией, тестированием и сопровождением безопасного REST API. Они являются ключевыми компонентами информационного обеспечения проекта и позволяют достичь высокого уровня качества и безопасности разработки.

## 1.3 Обзор и анализ существующих программных систем

Анализ существующих подходов к обеспечению безопасности на этапе разработки позволяет выявить ключевые принципы, на которых можно основывать новую методологию. Современные стандарты и методологии, такие как Microsoft SDL, OWASP ASVS, CERT Secure Coding и ISO/IEC 27001, предлагают различные стратегии для минимизации рисков уязвимостей в коде. Однако каждая из них имеет свои ограничения, которые необходимо учитывать при создании универсального решения.

Microsoft SDL (Security Development Lifecycle)

Microsoft SDL представляет собой структурированный процесс разработки программного обеспечения, включающий этапы обучения разработчиков, угрозный анализ, код-ревью и тестирование на безопасность. SDL фокусируется на интеграции мер безопасности в каждый этап жизненного цикла ПО, начиная с планирования и заканчивая поддержкой. Основные элементы SDL включают:

* обучение разработчиков: регулярные тренинги по безопасности, направленные на повышение осведомлённости о типовых угрозах;
* анализ угроз: идентификация потенциальных рисков на ранних этапах проектирования;
* код-ревью и статический анализ: проверка исходного кода на соответствие стандартам безопасности с использованием автоматизированных инструментов;
* тестирование на проникновение: имитация атак для выявления уязвимостей перед выпуском продукта.

Сильные стороны:

* четко определённые этапы, обеспечивающие комплексный подход к безопасности;
* поддержка регулярных аудитов и обновлений в ходе эксплуатации;
* фокус на предотвращение уязвимостей на ранних этапах разработки.

Слабые стороны:

* высокая сложность внедрения, требующая значительных ресурсов и времени;
* ориентация на крупные корпоративные проекты, что делает SDL менее применимым для малых команд или стартапов;
* отсутствие конкретных рекомендаций по реализации правил в коде (например, как валидировать данные в конкретном языке программирования).

OWASP ASVS (Application Security Verification Standard) OWASP ASVS — это международный стандарт проверки безопасности веб-приложений, включающий уровни проверок (Low, Medium, High) и критерии (валидация данных, управление сессиями, шифрование). ASVS предоставляет гибкие рамки для оценки защищённости систем, которые можно адаптировать под разные проекты.

Сильные стороны:

* гибкость: можно настроить уровень проверок в зависимости от масштаба проекта.
* поддержка автоматизированных тестов, что ускоряет процесс выявления уязвимостей.
* акцент на веб-специфические угрозы (XSS, CSRF, IDOR).

Слабые стороны:

* требует глубоких знаний безопасности от разработчиков для корректной реализации;
* отсутствие технических рекомендаций по реализации мер защиты (например, как использовать параметризованные запросы в SQL);
* ограниченное покрытие низкоуровневых ошибок (например, переполнение буфера в C/C++).

CERT Secure Coding StandardsCERT Secure Coding — это набор правил по безопасному программированию на конкретных языках (C, C++, Java, Python). Каждое правило сопровождается примерами кода и объяснением последствий нарушения.

Сильные стороны:

* упор на конкретные языки и типы ошибок (например, некорректная обработка строк в C);
* интеграция с инструментами статического анализа (например, Coverity, Klocwork);
* чёткая классификация уязвимостей по категориям (например, «Memory Management Errors»).

Слабые стороны:

* нет охвата веб-специфических угроз (XSS, CSRF). Фокус на low-level ошибки, игнорируя архитектурные аспекты безопасности;
* сложность внедрения в межъязыковые проекты (например, микросервисы на разных языках).

ISO/IEC 27001ISO/IEC 27001 — международный стандарт управления информационной безопасностью, включающий политики и процессы для защиты данных. Стандарт ориентирован на организационные аспекты, такие как управление рисками, контроль доступа и инцидент-менеджмент.

Сильные стороны:

* общий подход к защите данных и процессам, применимый к любым системам;
* возможность сертификации компаний, что усиливает доверие клиентов;
* акцент на документирование политик и процедур.

Слабые стороны:

* отсутствие технических рекомендаций для разработчиков (например, как шифровать пароли);
* высокая стоимость внедрения и сертификации;
* недостаточная гибкость для быстрого реагирования на новые угрозы.

DevSecOpsDevSecOps — это подход, интегрирующий безопасность в DevOps-процессы через автоматизацию. Основные элементы включают:

* автоматизированный анализ зависимостей (например, Dependabot, Snyk).
* статический и динамический анализ кода (SonarQube, OWASP ZAP).
* проверку безопасности в CI/CD (например, запрет слияния кода с уязвимостями).

Сильные стороны:

* быстрая интеграция мер безопасности в существующие процессы;
* поддержка регулярного мониторинга зависимостей и уязвимостей;
* упрощение внедрения для малых команд.

Слабые стороны:

* ограниченная глубина анализа (например, не выявляет логические ошибки в коде);
* зависимость от качества используемых инструментов;
* требует предварительного обучения команды.

Выводы по анализу

Существующие методологии имеют следующие ограничения:

* отсутствие баланса между теорией и практикой: стандарты вроде OWASP ASVS дают общие критерии, но не объясняют, как их реализовать в коде;
* высокая сложность внедрения: SDL и ISO 27001 требуют ресурсоемких процессов, что делает их неприменимыми для малых проектов;
* узкая специализация: CERT и OWASP ASVS фокусируются либо на языковых ошибках, либо на веб-уязвимостях, но не объединяют их в единую методологию.

Обоснование необходимости новой методологии. Разрабатываемая методология должна:

* комбинировать лучшие практики: использовать подходы SDL (цикл разработки), OWASP ASVS (тестирование) и CERT (языковые рекомендации);
* быть применимой к реальным проектам: включать конкретные правила валидации данных, работы с зависимостями и шифрования;
* поддерживать автоматизацию: интеграция с инструментами SAST/DAST и CI/CD.

Этот подход позволит создать универсальный свод правил, который можно внедрить как в крупные корпоративные системы, так и в стартапы, без значительных изменений в процессах разработки

# 2 Специальная, практическая, опытно-экспериментальная часть

## 2.1 Анализ задачи

### 2.1.1 Информационное моделирование предметной области

Проект направлен на демонстрацию методологии безопасной разработки программного обеспечения на примере реализации REST API. Основная цель — показать, как можно организовать регистрацию, аутентификацию, работу с данными и ограничение доступа по ролям в рамках простой, но защищённой архитектуры веб-приложения.

Система разрабатывается с использованием реляционной базы данных, где данные о пользователях и товарах хранятся в отдельных таблицах, связанных слабыми зависимостями. Структура базы данных представлена в виде ER-модели (рисунок А.4), которая отражает ключевые сущности предметной области.

В ходе проектирования были созданы следующие таблицы:

* users — хранит информацию о пользователях системы:

1. id — уникальный идентификатор пользователя;
2. email — уникальный email для входа;
3. password\_hash — хэшированный пароль (хранится с использованием BCrypt);
4. name — имя пользователя;
5. role — роль пользователя (ADMIN, EMPLOYEE, CLIENT).

* products — содержит данные о товарах:

1. id — уникальный идентификатор товара;
2. name — название товара;
3. price — цена товара.

* cart — корзина пользователя:

1. id — уникальный идентификатор корзины;
2. user\_id — внешний ключ на таблицу users, указывает владельца корзины;
3. cart\_item — элементы корзины;
4. id — уникальный идентификатор элемента;
5. cart\_id — внешний ключ на таблицу cart;
6. product\_id — внешний ключ на таблицу products;
7. quantity — количество товара в корзине.

* orders — заказы пользователей:

1. id — уникальный идентификатор заказа;
2. user\_id — внешний ключ на таблицу users, указывает, кому принадлежит заказ;
3. total\_price — общая стоимость заказа;
4. order\_date — дата оформления заказа.

* order\_item — позиции в заказе:

1. id — уникальный идентификатор позиции;
2. order\_id — внешний ключ на таблицу orders;
3. product\_id — внешний ключ на таблицу products;
4. quantity — количество товара в заказе;
5. price\_at\_time — цена товара на момент оформления заказа.

Связи между таблицами устанавливаются через внешние ключи при необходимости расширения системы. В текущей версии реализованы минимальные связи, чтобы сосредоточиться на демонстрации принципов безопасности: регистрации, аутентификации, авторизации, ограничении доступа по ролям и защите данных.

Таблица users связана с таблицей cart через отношение OneToOne — у каждого пользователя может быть только одна корзина.

Таблица cart связана с таблицей cart\_item через отношение OneToMany — корзина может содержать несколько товаров.

Таблица products участвует в двух связях: с cart\_item и order\_item через отношение ManyToOne — один товар, может быть, во многих корзинах и заказах.

Таблица orders связана с таблицей order\_item через отношение OneToMany — один заказ может содержать несколько товаров.

Таблица «Продукты» – содержит информацию о продуктах, доступных в системе.

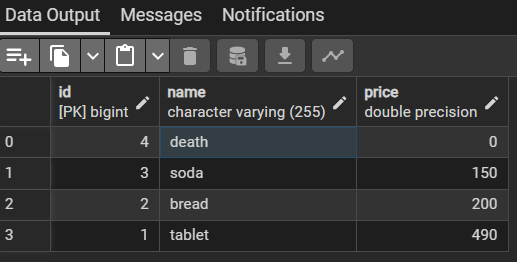


Рисунок 1 – Данные из таблицы продуктов

Таблица «Пользователи» – содержит данные о пользователях системы

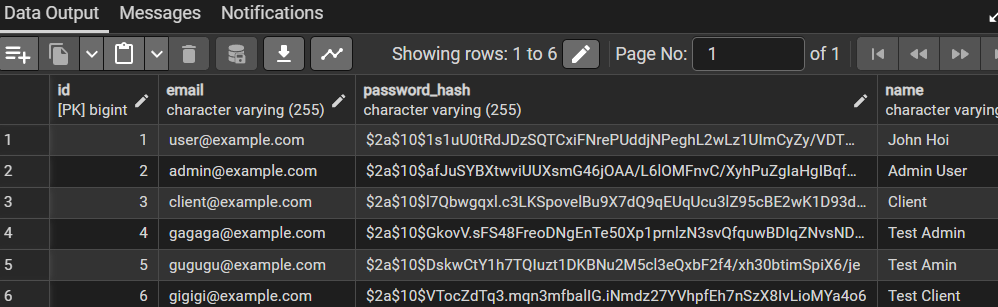


Рисунок 2 – Данные из таблицы пользователей

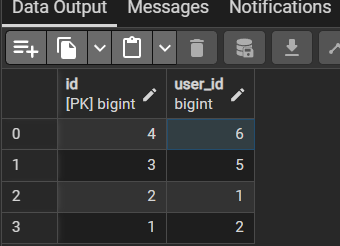


Рисунок 3 – Данные из таблицы корзины

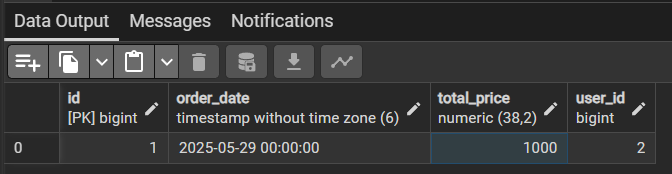


Рисунок 4 – Данные из таблицы Заказ\_продукты

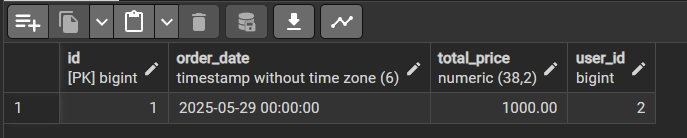


Рисунок 5 – Данные из таблицы заказы

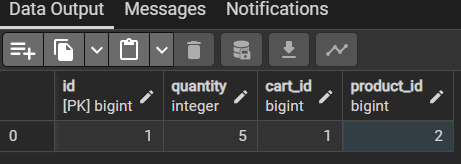


Рисунок 6 – Данные из таблицы Корзина\_продукты

Такая структура позволяет легко расширять систему, добавляя новые функциональные возможности, такие как история заказов, отслеживание статусов, оплата, фильтрация товаров и т.д.

Индексы в базе данных

Для повышения скорости выборки данных и ускорения операций поиска были созданы индексы:

* на поле email в таблице users, чтобы ускорить процесс авторизации;
* на поле id во всех таблицах (users, products, cart, orders) для эффективного поиска записей;
* на полях cart\_id, product\_id, order\_id для ускорения запросов к связанным данным.

Эти меры позволяют избежать замедления работы системы даже при увеличении объёма данных и активном использовании корзины и истории заказов.

Схема данных представлена на рисунке А.5 и может быть легко расширена для добавления новых сущностей (истории изменений) в случае масштабирования системы.

Безопасная обработка запросов

Все HTTP-запросы обрабатываются с соблюдением современных стандартов безопасности:

* использование JWT для stateless-аутентификации, исключающей необходимость хранения сессий;
* шифрование паролей с помощью BCryptPasswordEncoder;
* защита от CSRF-атак (отключена, так как используется stateless-архитектура);
* проверка входных данных на уровне контроллеров (@NotBlank, @Positive и другие аннотации JSR-380);
* заголовки безопасности: Content-Security-Policy, Cache-Control, Pragma.

Ограничение доступа по ролям

Для управления правами доступа используются аннотации Spring Security (@PreAuthorize) и конфигурации SecurityFilterChain. Доступ к эндпоинтам /products/\*\* ограничен только для пользователей с ролью ADMIN или EMPLOYEE.

Тестирование и модульная проверка

Проект покрыт unit-тестами с использованием JUnit 5 и Mockito, что позволяет проверять логику сервисов и контроллеров без зависимости от внешних систем. Это гарантирует надёжность и стабильность кода при дальнейшем развитии системы.

Реактивный подход (опционально)

Хотя в текущей реализации не используется реактивная модель обработки запросов, система спроектирована таким образом, чтобы её можно было легко адаптировать под Spring WebFlux в будущем. Такое решение позволит:

* обрабатывать множество запросов одновременно без блокировок;
* увеличить отзывчивость системы;
* поддерживать высокую нагрузку при масштабировании.

## 2.2 Описание логической структуры

### 2.2.1 Алгоритм программы

Программа представляет собой REST API, реализованное на языке Java с использованием фреймворка Spring Boot. Основное назначение системы — демонстрация методологии безопасной разработки программного обеспечения через реализацию защищённых механизмов регистрации, аутентификации, авторизации и работы с данными. Система спроектирована как простое, но функциональное веб-приложение, которое может быть расширено и применено в более сложных проектах, таких как интернет-магазины или CRM-системы.

Алгоритм работы программы охватывает жизненный цикл HTTP-запросов, начиная с регистрации пользователя и заканчивая выполнением защищённых операций над данными. Весь процесс реализован таким образом, чтобы обеспечить максимальную безопасность, соответствуя современным стандартам разработки веб-приложений. Архитектура приложения построена на основе принципов модульности и безопасности, что позволяет легко масштабировать систему и поддерживать её на высоком уровне качества.

Регистрация пользователей происходит путём отправки POST-запроса на эндпоинт /auth/register. При этом клиент передаёт email, пароль, имя и роль, которые проходят предварительную проверку на корректность формата и уникальность. Пароль шифруется с помощью BCryptPasswordEncoder перед сохранением в базу данных, что гарантирует его хранение только в зашифрованном виде. Если email уже используется, система генерирует исключение, информирующее клиента о наличии такого пользователя. Это позволяет избежать дублирования записей и повышает уровень безопасности.

Аутентификация реализована через JWT (JSON Web Token), что делает систему stateless и упрощает управление сессиями. После успешной регистрации пользователь может войти в систему, отправив POST-запрос на /auth/login с указанием email и пароля. Сервер проверяет учетные данные, сравнивает хэш пароля с сохранённым в БД и, в случае успеха, генерирует JWT-токен. Токен содержит информацию о пользователе: его ID, роль и срок действия. Данный токен должен передаваться в заголовке Authorization при каждом последующем запросе к защищённым ресурсам.

Авторизация реализована с учётом ролевой модели доступа. Пользователи могут иметь одну из трёх ролей: ADMIN, EMPLOYEE или CLIENT. Доступ к эндпоинтам /products/\*\* ограничен и предоставляется только пользователям с ролью ADMIN или EMPLOYEE. Контроль доступа осуществляется с помощью аннотации @PreAuthorize, а также настройками в конфигурационном классе SecurityConfig. Таким образом, система предотвращает несанкционированный доступ к данным и обеспечивает разграничение прав между различными категориями пользователей.

Управление товарами реализовано через контроллер ProductController, который предоставляет два основных метода: получение списка товаров и добавление новых позиций. Получение списка товаров доступно только авторизованным пользователям с соответствующими правами и реализовано через GET-запрос на /products. Перед выполнением бизнес-логики система проверяет наличие и валидность JWT-токена, после чего извлекает информацию о пользователе и устанавливает его аутентификацию в контексте Spring Security. Все входные данные проходят строгую валидацию, а также проверку на типы и значения, что снижает риск ошибок ввода и потенциальных уязвимостей.

Добавление товара выполняется через POST-запрос на тот же эндпоинт /products. При этом обязательные поля name и price проходят дополнительную проверку на стороне сервера. Имя не должно быть пустым, а цена должна быть положительным числом. Если все проверки успешно пройдены, товар сохраняется в базе данных, и клиент получает ответ в формате JSON с кодом 201 Created. Такой подход гарантирует целостность данных и минимизирует возможность попадания некорректных значений в систему.

Безопасность HTTP-запросов обеспечивается несколькими уровнями защиты. CSRF-атаки отключены, так как используется stateless-архитектура. Настроен набор заголовков безопасности, включающий Content-Security-Policy, Cache-Control и Pragma, что помогает защитить приложение от XSS и других видов атак. Все запросы проверяются на уровне Spring Security, а в случае возникновения ошибок возвращаются понятные сообщения с соответствующим HTTP-статусом. Вся логика безопасности реализована в классе SecurityConfig, где определены политики доступа, параметры сессии и заголовки.

Логирование действий пользователей реализовано с помощью библиотеки Lombok и аннотации @Slf4j. Все ключевые события, такие как регистрация, вход и добавление товаров, записываются в логи. Для точного отслеживания действий конкретного пользователя его ID сохраняется в MDC-контексте, что позволяет удобно анализировать логи и диагностиировать возможные проблемы.

Обработка исключений выполнена централизованно с помощью механизма @ControllerAdvice. Примеры исключений включают InvalidCredentialsException, UserAlreadyExistsException и InvalidProductDataException. Все эти исключения перехватываются глобальным обработчиком, который возвращает клиенту понятное сообщение и соответствующий HTTP-код. Это делает взаимодействие с API более прозрачным и устойчивым к ошибкам со стороны клиента.

Проект имеет чётко разделённую на модули структуру, включающую контроллеры, DTO, сущности, репозитории, сервисы, фильтры и компоненты безопасности. Такая организация позволяет легко тестировать, расширять и поддерживать систему. Каждый слой выполняет свою функцию: контроллеры принимают запросы, сервисы содержат бизнес-логику, репозитории работают с базой данных, а фильтры обеспечивают проверку токена и установку аутентификации.

Модульное тестирование покрывает ключевые части системы, включая работу контроллеров и сервисов. Для написания тестов используются JUnit 5 и Mockito, что позволяет мокировать зависимости и проверять логику работы системы без запуска сервера. Тесты проверяют корректность регистрации и аутентификации, ограничение доступа по ролям, обработку ошибок и работу с данными. Это даёт уверенность в том, что система работает правильно и безопасно.

Система использует реляционную базу данных, в которой информация о пользователях и товарах хранится в двух таблицах: users и products. Для ускорения выборки данных созданы индексы на поле email и id. Хотя связи между таблицами минимальны, архитектура спроектирована так, чтобы их можно было легко добавить при расширении системы.

Ещё одной важной особенностью является использование JWT для stateless-аутентификации. Эта задача решена с помощью собственного фильтра JwtFilter, который извлекает и проверяет токен, устанавливая аутентификацию в контексте Spring Security. Фильтр встроен в цепочку фильтров Spring и вызывается до выполнения бизнес-логики. Это обеспечивает защиту всех эндпоинтов, кроме /auth/\*\*.

Также стоит отметить использование аннотаций JSR-380 (@NotBlank, @Positive) на уровне контроллеров, что позволяет выполнять валидацию данных ещё на этапе получения запроса. Это предотвращает обработку невалидных данных в бизнес-логике и снижает риск возникновения ошибок на уровне приложения.

Таким образом, алгоритм программы охватывает весь жизненный цикл взаимодействия пользователя с системой — от регистрации до выполнения защищённых операций над данными. Реализация выполнена с соблюдением принципов безопасной разработки: используются современные механизмы аутентификации, шифрования, валидации и ограничения доступа. Архитектура системы обеспечивает читаемость, расширяемость и безопасность, что делает её пригодной как для учебных целей, так и для практического применения.

2.2.2 Используемые методы

случае перехода к продакшн-среде предусмотрена возможность замены H2 на PostgreSQL, что обеспечит высокую производительность и надёжность при работе с данными в реальных условиях. База данных организована с учётом принципов нормализации, что позволило минимизировать избыточность информации и обеспечить целостность данных. Основные таблицы включают данные о пользователях и товарах, связанные между собой минимальными зависимостями, что упрощает расширение системы в будущем.

Методы работы с базой данных:

* проектирование базы данных: структура спроектирована таким образом, чтобы система была легко расширяемой. Таблицы users и products содержат минимальное количество полей, необходимых для демонстрации функционала;
* оптимизация SQL-запросов: для ускорения выборки данных созданы индексы на ключевых полях, таких как email и id. Использование Spring Data JPA позволяет автоматически формировать эффективные запросы к базе данных, исключая необходимость ручной оптимизации;
* обеспечение целостности данных: все операции добавления и обновления данных проходят строгую проверку на уровне бизнес-логики и ORM (Hibernate), что предотвращает запись некорректных или повреждённых записей.

Методы защиты данных играют ключевую роль в архитектуре приложения. Безопасность реализована на всех уровнях взаимодействия с системой. Прежде всего, защита от SQL-инъекций обеспечивается за счёт использования Hibernate / Spring Data JPA, которые по умолчанию применяют параметризованные запросы. Это исключает возможность внедрения вредоносного SQL-кода через входные данные.

Кроме того, пароли пользователей никогда не сохраняются в открытом виде. Все пароли шифруются перед записью в базу данных с использованием алгоритма BCrypt, который считается одним из самых безопасных способов хэширования. Такой подход гарантирует защиту конфиденциальной информации даже в случае компрометации базы данных.

Работа с HTTP-запросами также реализована с соблюдением принципов безопасности. Для аутентификации и авторизации используется JWT (JSON Web Token), что делает систему stateless и удобной для масштабирования. Генерация токена выполняется с применением секретного ключа, достаточной длины для обеспечения криптографической стойкости. Проверка токена осуществляется на каждом запросе к защищённым эндпоинтам, что исключает возможность несанкционированного доступа.

Авторизация выполнена с использованием аннотаций @PreAuthorize и конфигурации Spring Security, позволяя ограничивать доступ к различным частям системы в зависимости от роли пользователя. Это даёт возможность гибко управлять правами без необходимости изменения логики контроллеров.

Для обработки входных данных применяются стандарты JSR-380, такие как @NotBlank, @Positive и другие. Все данные, передаваемые клиентом, проверяются до начала выполнения бизнес-логики, что снижает вероятность ошибок и потенциальных уязвимостей. Исключения, возникающие при невалидных данных, обрабатываются централизованно с помощью механизма @ControllerAdvice, обеспечивая понятную и унифицированную реакцию системы на некорректный ввод.

Логирование действий пользователей реализовано с использованием библиотеки Lombok и аннотации @Slf4j. Все ключевые события, такие как регистрация, вход и добавление товаров, фиксируются в логах. ID пользователя сохраняется в MDC (Mapped Diagnostic Context), что позволяет точно отслеживать действия конкретного пользователя и анализировать логи в системах централизованного логирования.

Взаимодействие с базой данных осуществляется через Spring Data JPA, что обеспечивает простоту работы с объектами, исключая необходимость ручного составления SQL-запросов. Методы findAll(), save() и findByEmail() используются в сервисах ProductServiceImpl и UserServiceImpl для выполнения операций CRUD над данными. Такой подход повышает читаемость кода и упрощает его дальнейшее сопровождение.

Система покрыта модульными тестами, написанными с использованием JUnit 5 и Mockito. Эти тесты проверяют корректность регистрации, аутентификации, авторизации и работы с товарами. Тестирование проводится без запуска сервера с помощью MockMvc, что ускоряет процесс проверки и позволяет избежать внешних зависимостей. Таким образом, качество кода поддерживается на высоком уровне, а риск появления regressions сводится к минимуму.

Особое внимание уделено обработке исключений. Все ошибки, возникающие в процессе выполнения, перехватываются глобальным обработчиком, который возвращает клиенту понятные сообщения и соответствующие HTTP-статусы. Например, попытка зарегистрировать существующего пользователя приводит к выбросу UserAlreadyExistsException, а некорректные данные о товаре — к InvalidProductDataException. Это делает работу с API более прозрачной и предсказуемой.

Ещё одной важной частью является реализация фильтра JwtFilter, который встроен в цепочку фильтров Spring Security. Этот фильтр извлекает и проверяет JWT-токен, устанавливая аутентификацию пользователя в контексте Spring. Фильтр работает до выполнения любого контроллера, обеспечивая защиту всех защищённых эндпоинтов. Также в него встроена запись ID пользователя в MDC для последующего логирования.

В системе также применяются современные заголовки безопасности, такие как Content-Security-Policy, Cache-Control и Pragma, что защищает приложение от XSS и других видов атак. CSRF-атаки отключены, так как система использует stateless-архитектуру, основанную на токенах.

Таким образом, в проекте задействованы современные и проверенные практики работы с базами данных, безопасности, валидации и логирования. Реализация выполнена с учётом требований к защите данных и устойчивости к распространённым уязвимостям, что делает систему не только функциональной, но и безопасной для практического применения.

### 2.2.3 Составные части программы и связи между ними

Разработанная программа представляет собой REST API, реализованное на языке Java с использованием фреймворка Spring Boot. Архитектура системы построена на основе принципов модульности и разделения ответственности, что обеспечивает простоту сопровождения, расширяемость и безопасность. Программа состоит из логически завершённых компонентов, каждый из которых выполняет свою функцию и взаимодействует с другими частями системы.

Все элементы программы связаны между собой и работают совместно для обеспечения корректного выполнения бизнес-процессов: регистрации пользователей, аутентификации через JWT, ограничения доступа по ролям и работы с данными о товарах. Взаимодействие между компонентами организовано с использованием стандартных практик разработки на Spring Boot и соответствует принципам проектирования MVC (Model-View-Controller), хотя View-слой отсутствует как самостоятельный элемент, так как система является backend-компонентом.

Составные части программы:

Контроллеры

Классы AuthController и ProductController являются точкой входа для HTTP-запросов. Они обрабатывают входящие запросы, проводят начальную валидацию данных и передают управление на уровень сервисов. Контроллеры аннотированы как @RestController, что указывает на их предназначение — возвращать данные в формате JSON. Все эндпоинты защищены механизмами безопасности и имеют строго определённые маршруты.

DTO (Data Transfer Objects)

Объекты передачи данных представлены классами AuthRequest, RegisterRequest, AuthResponse и ProductDTO. Использование DTO позволяет отделить внешний интерфейс API от внутренней структуры модели данных. Это повышает безопасность и упрощает поддержку кода. Для всех DTO-классов применяется аннотация @Data из Lombok, обеспечивающая автоматическую генерацию методов getter, setter, toString и других.

Модели (Entity)

Классы User и Product представляют собой сущности, отображаемые на таблицы базы данных. Они используются JPA для хранения и извлечения информации. Эти классы содержат поля, аннотированные с помощью @Entity, @Table, @Id, @GeneratedValue, что позволяет Spring Data JPA корректно маппить объекты в БД.

Репозитории

Интерфейсы UserRepository и ProductRepository предоставляют доступ к данным и реализуют операции CRUD над сущностями User и Product. Они основаны на механизме Spring Data JPA и не требуют реализации методов вручную. Репозитории используются сервисами для сохранения, получения и проверки данных.

Сервисы

Классы UserServiceImpl и ProductServiceImpl содержат бизнес-логику приложения. Сервисы инкапсулируют операции по регистрации, авторизации, созданию товаров и другим действиям. Они взаимодействуют с репозиториями, применяют правила валидации, записывают логи и формируют ответы для контроллеров. Сервисы аннотированы как @Service и внедряются в контроллеры через конструктора.

Фильтры

Класс JwtFilter реализует проверку JWT-токена на каждом запросе, кроме тех, которые относятся к /auth/\*\*. Он встроен в цепочку фильтров Spring Security и отвечает за установку аутентификации в контексте Spring. Фильтр извлекает ID и роль пользователя из токена и сохраняет их в MDC для последующего логирования.

Утилиты безопасности

Класс JwtUtils отвечает за генерацию, извлечение и проверку JWT-токенов. Он использует библиотеку io.jsonwebtoken и реализует криптографическую подпись с алгоритмом HS512. Утилита также содержит секретный ключ, который используется для подписи токенов, и обеспечивает защиту от подделки.

Конфигурация безопасности

Класс SecurityConfig содержит настройки Spring Security, включая политики сессий, заголовки безопасности, параметры аутентификации и авторизации. Здесь настраивается stateless-архитектура, отключается CSRF-защита, задаются права доступа к эндпоинтам и создаётся бин BCryptPasswordEncoder для шифрования паролей.

Исключения и обработка ошибок

Программа содержит собственные исключения: InvalidCredentialsException, UserAlreadyExistsException и InvalidProductDataException. Они используются для обработки ошибочных ситуаций, таких как неверные учётные данные, дублирование email или некорректные данные о товаре. Исключения обрабатываются централизованно с помощью механизма @ControllerAdvice.

Тестирование

Проект покрыт unit-тестами, реализованными с использованием JUnit 5 и Mockito. Тесты находятся в классах AuthControllerTest и ProductControllerTest. Они проверяют работу контроллеров, валидацию входных данных и правильность ответов сервера. Для тестирования используются моки сервисов, а также MockMvc для имитации HTTP-запросов без запуска сервера.

Логирование

Для записи событий в системе используется библиотека SLF4J с аннотацией @Slf4j из Lombok. Логируются события регистрации, входа и добавления товаров. Для точного трассирования действий пользователей используется MDC (Mapped Diagnostic Context), в который записывается userId.

Безопасность и защита данных

Для шифрования паролей используется BCryptPasswordEncoder, который внедряется в сервис UserServiceImpl. Все пароли хранятся только в зашифрованном виде. Доступ к защищённым эндпоинтам ограничен с помощью аннотаций @PreAuthorize и конфигурации Spring Security. На уровне HTTP-запросов используются заголовки безопасности, такие как Content-Security-Policy, Cache-Control, Pragma.

Валидация данных

На стороне контроллеров применяются аннотации JSR-380 (@NotBlank, @Positive) для проверки входных данных до начала выполнения бизнес-логики. Это предотвращает обработку некорректных или потенциально опасных данных и снижает вероятность возникновения ошибок на уровне приложения.

База данных

Для хранения информации используется реляционная база данных H2 в режиме in-memory, что удобно для тестирования и демонстрации. При необходимости система может быть адаптирована для использования PostgreSQL. Индексы созданы на ключевых полях, таких как email и id, что ускоряет выборку данных.

Связь между компонентами

Все компоненты программы взаимодействуют согласованно и строго по назначению:

* контроллеры принимают HTTP-запросы и вызывают сервисы;
* сервисы обращаются к репозиториям для работы с данными;
* репозитории взаимодействуют с базой данных;
* фильтры проверяют наличие и валидность JWT-токена до передачи управления контроллерам;
* утилиты безопасности отвечают за генерацию и проверку токенов;
* конфигурация безопасности определяет политики доступа, заголовки и типы сессий;
* исключения перехватываются глобальным обработчиком, который возвращает клиенту понятные сообщения;
* логирование происходит на уровне сервисов и фильтров, с возможностью трассировки по ID пользователя.

Такое построение системы позволяет легко расширять её функциональность, добавлять новые сущности и улучшать безопасность без изменения существующих модулей. Все зависимости внедряются через конструкторов, что делает систему тестируемой и независимой от конкретной реализации.

## 2.3 Описание работы программы

### 2.3.1 Общие сведения

Данная система представляет собой REST API, реализованное на языке Java с использованием фреймворка Spring Boot. Программа предназначена для демонстрации методологии безопасной разработки программного обеспечения и показывает, как можно организовать регистрацию, аутентификацию, авторизацию и работу с данными в рамках простой, но защищённой архитектуры веб-приложения. Система может быть использована как шаблон для построения более сложных решений, таких как интернет-магазины или сервисы управления продуктами, где важна защита информации и контроль прав доступа.

REST API предоставляет следующие функции:

* регистрация пользователей с указанием email, пароля, имени и роли;
* авторизация с выдачей JWT-токена;
* получение списка товаров;
* добавление новых товаров с валидацией входных данных;
* ограничение доступа по ролям (ADMIN, EMPLOYEE);
* безопасное хранение паролей с помощью BCrypt;
* защита от распространённых уязвимостей (XSS, CSRF, SQL-инъекции);
* валидация данных на уровне контроллеров;
* логирование действий пользователей.

Программа обеспечивает удобное взаимодействие с backend-частью через HTTP-запросы и может использоваться мобильными и веб-приложениями как основа для клиентской части. API разработан в соответствии со стандартами RESTful, что делает его легко интегрируемым в любые клиентские приложения.

Система разработана с учетом принципов модульности, безопасности и расширяемости. Все компоненты строго разделены: контроллеры обрабатывают запросы, сервисы содержат бизнес-логику, репозитории работают с базой данных, а фильтры обеспечивают проверку токена и установку аутентификации. Такая организация повышает читаемость кода, упрощает тестирование и позволяет быстро масштабировать функционал.

Безопасность является ключевым элементом данной системы. Для обеспечения защиты используются современные механизмы:

* аутентификация через JWT;
* шифрование паролей с помощью BCryptPasswordEncoder;
* валидация входных данных с использованием JSR-380;
* заголовки безопасности (Content-Security-Policy, Cache-Control, Pragma);
* ограничение доступа по ролям (@PreAuthorize);
* отключение CSRF-атак в stateless-архитектуре.

Тестирование проводится с применением JUnit 5 и Mockito, что гарантирует надёжность работы всех компонентов без необходимости запуска сервера. Контроллеры покрыты unit-тестами, которые проверяют корректность регистрации, входа в систему, добавления товаров и обработки ошибок.

Логирование выполнено с использованием @Slf4j из Lombok, что позволяет отслеживать ключевые события, такие как регистрация, вход и операции над товарами. ID пользователя записывается в MDC для последующего логирования действий конкретного пользователя, что особенно полезно при анализе логов и диагностике проблем.

Функциональность системы спроектирована таким образом, чтобы она могла быть легко расширена за счёт добавления новых сущностей, ролей и интерфейсов. Благодаря использованию Spring Data JPA и нормализованной структуры базы данных, система поддерживает гибкую настройку и дальнейшее развитие.

Таким образом, программа представляет собой защищённое и модульное REST API, которое может служить образцом реализации безопасной разработки программного обеспечения. Она сочетает в себе современные подходы к проектированию, безопасности и тестированию, что делает её пригодной как для учебных целей, так и для практического применения.

### 2.3.2 Функциональное назначение

Программа выполняет ключевые функции, демонстрирующие реализацию принципов безопасной разработки программного обеспечения в контексте создания REST API. Она предоставляет пользователям возможность регистрироваться, авторизироваться, а также выполнять операции над данными с учётом ролевого ограничения доступа. Все функции реализованы с применением современных практик безопасности, таких как шифрование паролей, stateless-аутентификация через JWT, защита от распространённых уязвимостей и строгая валидация входных данных.

Регистрация пользователей

Система позволяет новым пользователям зарегистрироваться, указав email, пароль, имя и роль (ADMIN, EMPLOYEE, CLIENT). При регистрации:

* проверяется уникальность email;
* пароль шифруется с использованием алгоритма BCrypt перед сохранением в базе данных;
* создаётся новая запись в таблице users.

Эта функция гарантирует, что учетные данные хранятся безопасно и не могут быть восстановлены из базы данных даже при её компрометации.

Аутентификация через JWT

После успешной регистрации пользователь может войти в систему, отправив email и пароль. В случае корректных данных система генерирует JWT-токен, содержащий:

* ID пользователя;
* роль;
* срок действия токена.

Токен должен быть передан в заголовке Authorization при каждом последующем запросе к защищённым ресурсам. Использование JWT делает систему stateless, что повышает масштабируемость и упрощает управление сессиями.

Авторизация по ролям

Система реализует механизм авторизации, основанный на ролях пользователей. Пользователи могут иметь одну из следующих ролей:

* ADMIN;
* EMPLOYEE;
* CLIENT.

Доступ к эндпоинтам /products/\*\* ограничен только пользователям с ролью ADMIN или EMPLOYEE. Это ограничение реализовано с помощью аннотаций @PreAuthorize("hasRole('ADMIN') or hasRole('EMPLOYEE')"), а также конфигурации Spring Security. Такой подход предотвращает несанкционированный доступ и позволяет гибко управлять правами без изменения логики контроллеров.

Управление товарами

Пользователи с соответствующими правами могут просматривать список товаров и добавлять новые элементы в систему. Основные функции:

* получение списка товаров через GET-запрос на /products;
* добавление товара через POST-запрос на /products.

Перед выполнением операций система проверяет наличие и валидность JWT-токена, а также права пользователя. Все входные данные проходят строгую валидацию:

* название товара не должно быть пустым (@NotBlank);
* цена должна быть положительной (@Positive).

Если данные некорректны, пользователь получает понятное сообщение об ошибке с кодом 400 Bad Request.

Безопасность HTTP-запросов

Для защиты от внешних угроз применяются следующие меры:

* отключение CSRF-атак (stateless архитектура);
* заголовки безопасности:

1. Content-Security-Policy;
2. Cache-Control: no-store;
3. Pragma: no-cache.

* проверка JWT-токена на каждом запросе, кроме /auth/\*\*;
* централизованная обработка исключений;
* логирование действий пользователей с использованием MDC (Mapped Diagnostic Context).

Эти меры обеспечивают защиту от XSS, CSRF, подделки запросов и других распространённых видов атак.

Логирование действий

Система логирует все ключевые события:

* регистрация новых пользователей;
* авторизация и вход в систему;
* добавление товаров.

Для логирования используется библиотека SLF4J с аннотацией @Slf4j из Lombok. ID пользователя сохраняется в MDC, что позволяет точно отслеживать действия конкретного пользователя. Это особенно полезно при анализе логов и диагностике возможных проблем.

Обработка исключений

Все исключения обрабатываются централизованно с помощью механизма @ControllerAdvice. Примеры обрабатываемых исключений:

* InvalidCredentialsException — при неверных учетных данных;
* UserAlreadyExistsException — при попытке зарегистрировать существующего пользователя;
* InvalidProductDataException — при некорректных данных о товаре.

Клиент получает понятное сообщение об ошибке и соответствующий HTTP-статус, что делает использование API более прозрачным и надёжным.

Модульное тестирование

Контроллеры и сервисы покрыты unit-тестами, написанными с использованием JUnit 5 и Mockito. Тесты проверяют:

* корректность регистрации и аутентификации;
* обработку ошибок;
* работу с данными.

MockMvc используется для тестирования HTTP-запросов без запуска сервера, что ускоряет процесс проверки и позволяет избежать внешних зависимостей. Это обеспечивает высокое качество кода и снижает вероятность regressions.

Работа с базой данных

Для хранения информации используются две таблицы:

* users: id, email, password\_hash, name, role;
* products: id, name, price.

Для ускорения работы с данными созданы индексы на поля email и id. База данных организована таким образом, чтобы обеспечить целостность и согласованность данных. В текущей реализации используется H2, но предусмотрена возможность перехода на PostgreSQL для использования в продакшене.

Безопасная передача данных

Все чувствительные данные, такие как пароли, никогда не передаются в открытом виде. Пароли шифруются до сохранения в БД, а токены содержат только необходимую информацию, подписанную секретным ключом. Это обеспечивает защиту данных как при хранении, так и при передаче между клиентом и сервером.

Валидация данных

Входные данные проверяются на уровне контроллеров с использованием аннотаций JSR-380 (@NotBlank, @Positive). Это позволяет исключить обработку невалидных данных на уровне бизнес-логики и повысить уровень отказоустойчивости системы.

Функциональное назначение программы заключается в демонстрации методологии безопасной разработки программного обеспечения на примере реализации REST API. Реализованные функции охватывают жизненный цикл взаимодействия пользователя с системой — от регистрации и аутентификации до выполнения защищённых операций над данными. Все действия выполнены с соблюдением современных стандартов безопасности, что делает систему не только функциональной, но и защищённой от большинства распространённых уязвимостей.

### 2.3.3 Связи с другими программами

Разработанная программа представляет собой REST API, реализованное на языке Java с использованием фреймворка Spring Boot. В рамках данного проекта основное внимание уделено демонстрации методологии безопасной разработки программного обеспечения, поэтому прямое взаимодействие с внешними системами ограничено. Тем не менее, система спроектирована таким образом, чтобы её можно было легко интегрировать с различными внешними компонентами при дальнейшем расширении функционала.

Программа взаимодействует со следующими системами и библиотеками:

* Spring Security – используется для реализации механизмов аутентификации и авторизации. Через данный фреймворк осуществляется защита эндпоинтов, проверка ролей пользователей и управление политикой доступа;
* JWT (JSON Web Token) – применяется для stateless-аутентификации и передачи данных о пользователе между клиентом и сервером. Библиотека io.jsonwebtoken обеспечивает генерацию, проверку и извлечение информации из токена;
* Hibernate / Spring Data JPA – используется для работы с реляционной базой данных. ORM-библиотека обеспечивает маппинг объектов в таблицы, выполнение CRUD-операций и автоматическую валидацию запросов к БД;
* BCryptPasswordEncoder – применяется для безопасного хэширования паролей перед сохранением их в базе данных. Это позволяет исключить хранение паролей в открытом виде и повысить уровень защищённости системы;
* Jakarta Validation (JSR-380) – используется для валидации входных данных на уровне контроллеров. Аннотации, такие как @NotBlank и @Positive, позволяют проверять корректность формата и значения полей до начала выполнения бизнес-логики;

Помимо этого, программа имеет потенциальную возможность интеграции со следующими системами:

* PostgreSQL – хотя в текущей реализации используется встроенная БД H2, предусмотрена возможность замены на PostgreSQL, что повысит производительность и надёжность при переходе в продакшен;
* Swagger UI – может быть интегрирован для документирования и тестирования REST API. Это упростит работу с эндпоинтами и позволит пользователям просматривать и выполнять запросы прямо в браузере;
* Docker – хотя в текущей версии контейнеризация не реализована, предусмотрен путь к интеграции с Docker для удобства деплоя и масштабирования сервиса;
* Postman – используется как инструмент для ручного тестирования REST API. Позволяет отправлять HTTP-запросы, проверять ответы и анализировать поведение системы при различных сценариях использования.

Также в проекте используются стандартные средства разработки и контроля версий:

* IntelliJ IDEA – как основная IDE для написания, отладки и тестирования кода;
* Git – для управления версиями кода, отслеживания изменений и организации командной разработки;
* JUnit 5 + Mockito – для модульного тестирования контроллеров и сервисов. Тесты проверяют корректность регистрации, аутентификации и операций с товарами без необходимости запуска сервера.

Таким образом, программа имеет чётко определённые связи с используемыми библиотеками и фреймворками, а также спроектирована с учетом возможности дальнейшей интеграции с внешними системами. Такой подход делает её расширяемой, переносимой и готовой к применению в реальных условиях эксплуатации.

### 2.3.4 Входные данные

Разработанная программа представляет собой REST API, предназначенное для демонстрации методологии безопасной разработки программного обеспечения. На вход система получает данные, необходимые для выполнения ключевых операций: регистрации пользователей, авторизации через JWT, ограничения доступа по ролям и работы с товарами. Все входные данные проходят строгую проверку на корректность и безопасность перед началом обработки, что позволяет предотвратить ошибки, защититься от некорректных или злонамеренных запросов и обеспечить целостность данных.

Данные для регистрации пользователя

При регистрации пользователь предоставляет следующие данные:

* email — уникальный адрес электронной почты, который используется для последующего входа в систему;
* password — пароль, который шифруется с помощью алгоритма BCrypt перед сохранением в базу данных;
* name — имя пользователя;
* role — роль (ADMIN, EMPLOYEE, CLIENT), определяющая уровень доступа к защищённым ресурсам.

Система проверяет уникальность email и формат ввода, а также гарантирует, что роль принадлежит к одному из допустимых значений. Если данные не соответствуют требованиям, клиент получает понятное сообщение об ошибке с HTTP-статусом 400 Bad Request.

Данные для авторизации

Для входа в систему пользователь отправляет:

* email — указанный при регистрации;
* password — пароль, который сравнивается с хэшированным значением в базе данных.

Если учетные данные корректны, сервер генерирует JWT-токен, содержащий ID пользователя, его роль и срок действия. Токен должен быть использован во всех последующих запросах к защищённым эндпоинтам.

JWT-токен

JWT-токен является важным входным элементом системы, так как он используется для stateless-аутентификации. Он передаётся в заголовке Authorization и содержит:

* идентификатор пользователя;
* роль пользователя;
* срок действия.

Токен проверяется фильтром JwtFilter на каждом запросе, кроме тех, которые относятся к /auth/\*\*. Это позволяет системе идентифицировать пользователя и установить его аутентификацию без использования сессий.

Данные для добавления товара

Пользователи с ролью ADMIN или EMPLOYEE могут добавлять новые товары в систему, предоставляя следующие данные:

* name — название товара, которое не должно быть пустым (@NotBlank);
* price — цена товара, которая должна быть положительным числом (@Positive).

Эти данные передаются в теле POST-запроса в формате JSON и проверяются до начала выполнения бизнес-логики. Невалидные данные приводят к выбросу исключения и возврату клиенту соответствующего ответа.

HTTP-запросы к защищённым эндпоинтам

Система принимает входящие HTTP-запросы к следующим эндпоинтам:

* POST /auth/register — регистрация нового пользователя;
* POST /auth/login — получение JWT-токена;
* GET /products — получение списка товаров;
* POST /products — добавление нового товара.

Каждый запрос проверяется на соответствие ожидаемому формату и наличию прав доступа. Для защиты от уязвимостей применяются аннотации JSR-380, параметризованные запросы, заголовки безопасности и ограничения Spring Security.

Исключения и ошибочные данные

Программа обрабатывает различные виды невалидных входных данных, такие как:

* Неверный email или пароль при входе;
* Попытка зарегистрировать существующего пользователя;
* Некорректные данные о товаре.

Все исключения перехватываются централизованно с помощью механизма @ControllerAdvice, что позволяет возвращать клиенту понятные сообщения и соответствующие HTTP-коды ошибок.

Логирование действий

На уровне сервисов и фильтров система логирует все ключевые события, связанные с регистрацией, входом и добавлением товаров. Для трассировки действий конкретного пользователя используется MDC (Mapped Diagnostic Context), в который записывается userId.

Таким образом, входные данные охватывают все основные аспекты функционирования программы: от регистрации и аутентификации до выполнения операций над товарами. Все данные подвергаются строгой валидации и проверке, что делает систему устойчивой к ошибкам и защищённой от большинства распространённых уязвимостей. Реализация входных интерфейсов выполнена с соблюдением принципов безопасной разработки, что делает систему удобной как для тестирования, так и для практического применения.

### 2.3.5 Выходные данные

Работа разработанной программы направлена на формирование выходных данных, которые играют ключевую роль в обеспечении прозрачности работы системы, проверке корректности выполнения операций и поддержании высокого уровня безопасности. Эти данные предоставляют пользователю информацию о состоянии системы, результатах выполнения запросов и возможных ошибках, что позволяет эффективно использовать API и интегрировать его в клиентские приложения.

Ответы на регистрацию пользователя

После успешной регистрации система возвращает HTTP-ответ со статусом 201 Created и сообщением "User registered". В случае ошибки (например, дублирование email или некорректные данные) возвращается статус 400 Bad Request с понятным описанием проблемы. Такой подход делает процесс регистрации безопасным и предсказуемым, исключая возможность появления недопустимых записей в базе данных.

JWT-токен после авторизации

При успешной аутентификации система возвращает JSON-объект, содержащий:

* token — сгенерированный JWT-токен, который должен быть использован при последующих запросах;
* userId — уникальный идентификатор пользователя;
* role — роль пользователя (ADMIN, EMPLOYEE, CLIENT), определяющую уровень доступа к защищённым ресурсам.

Эти данные обеспечивают stateless-аутентификацию и позволяют системе устанавливать права доступа без использования сессий. Формат токена предусматривает защиту от подделки и содержит срок действия, что повышает уровень безопасности.

Список товаров

Для пользователей с ролью ADMIN или EMPLOYEE система возвращает список всех товаров в формате JSON. Каждый товар представлен объектом, содержащим:

* id — уникальный идентификатор;
* name — название товара;
* price — стоимость товара.

Ответ сопровождается заголовками Cache-Control: no-store и Pragma: no-cache, что исключает кэширование чувствительной информации. Это делает систему более безопасной при передаче данных через сети общего пользования.

Добавление нового товара

При успешном добавлении товара система возвращает HTTP-статус 201 Created и JSON-объект, соответствующий созданному товару. Данные включают:

* id — присвоенный товару идентификатор;
* name — указанное название;
* price — установленную цену.

Если данные не прошли валидацию (например, цена отрицательная или название отсутствует), возвращается статус 400 Bad Request с подробным описанием ошибки. Это позволяет клиентской части сразу выявить проблему и исправить запрос.

Информация об ошибках

Программа предоставляет централизованную обработку исключений, которая формирует унифицированные ответы при возникновении ошибок. Примеры:

* InvalidCredentialsException — возвращается при неверных учетных данных с HTTP-статусом 401 Unauthorized;
* UserAlreadyExistsException — генерируется при попытке зарегистрировать существующего пользователя, с кодом 400 Bad Request;
* InvalidProductDataException — вызывается при некорректных данных о товаре, также возвращается статус 400.

Клиент получает понятное сообщение об ошибке в JSON-формате, что делает использование API более прозрачным и удобным для диагностики.

Логирование действий

На уровне сервисов и фильтров программа логирует все ключевые события:

* регистрация новых пользователей;
* авторизация и вход в систему;
* операции над товарами.

ID пользователя сохраняется в MDC (Mapped Diagnostic Context), что позволяет точно отслеживать действия конкретного пользователя. Логи могут быть собраны и проанализированы в дальнейшем с помощью систем централизованного логирования, таких как ELK Stack.

Тестовые результаты

Модульные тесты, реализованные с использованием JUnit 5 и Mockito, формируют выходные данные, которые позволяют проверить корректность работы контроллеров и сервисов без запуска сервера. Тесты проверяют:

* корректность регистрации и авторизации;
* обработку ошибок;
* работу с данными.

MockMvc используется для имитации HTTP-запросов и проверки возвращаемых ответов, что гарантирует стабильность и предсказуемость поведения системы.

Выходные данные программы охватывают все основные аспекты взаимодействия с REST API: от регистрации и аутентификации до выполнения операций над товарами и обработки ошибок. Все ответы возвращаются в формате JSON, что делает их легко обрабатываемыми клиентскими приложениями. Реализация выходных интерфейсов выполнена с соблюдением принципов безопасной разработки, что делает систему надёжной, читаемой и готовой к практическому применению.

### 2.3.6 Тестирование

Тестирование программы проводилось с целью проверки корректности реализации ключевых функций REST API и обеспечения безопасности выполнения операций. Были использованы различные уровни тестирования: модульное, интеграционное и функциональное, что позволило охватить все слои приложения и убедиться в надёжности и предсказуемости его поведения.

Модульное тестирование

Для проверки работы отдельных компонентов системы были созданы unit-тесты, написанные с использованием JUnit 5 и Mockito. В рамках данного подхода тестируются:

* сервисы (UserServiceImpl, ProductServiceImpl);
* контроллеры (AuthController, ProductController);
* утилиты безопасности (JwtUtils);
* обработка исключений.

Тесты выполнялись без запуска сервера, с использованием MockMvc, что позволило имитировать HTTP-запросы и проверять работу контроллеров в изолированной среде. Все сервисы мокались с помощью Mockito, чтобы исключить зависимости от внешних систем и ускорить процесс тестирования.

Примеры проверяемых сценариев:

* корректная регистрация пользователя;
* проверка уникальности email;
* аутентификация с выдачей JWT-токена;
* получение списка товаров;
* добавление товара с валидацией данных.

Каждый тест проверяет не только успешное выполнение операции, но и обработку ошибочных ситуаций, таких как дублирование email или некорректные данные о товаре.

Интеграция с MockMvc

Для тестирования HTTP-запросов использовался фреймворк MockMvc, позволяющий отправлять запросы к контроллерам без запуска сервера. Это дало возможность проверить:

* корректность маршрутов;
* работу аннотаций @PostMapping, @GetMapping, @PreAuthorize;
* обработку входящих JSON-данных;
* возврат правильного HTTP-статуса и формата ответа.

Также проверялось, что заголовки безопасности (Cache-Control, Pragma) передаются правильно при работе с защищёнными эндпоинтами.

Тестирование авторизации и безопасности

Особое внимание было уделено тестированию механизмов аутентификации и авторизации:

* генерация и валидация JWT-токена;
* проверка ограничения доступа по ролям (ADMIN, EMPLOYEE);
* отключение CSRF-атак и stateless-архитектура.

Было проведено несколько тестовых сценариев, подтверждающих, что:

* только зарегистрированные пользователи могут получить доступ к /products/\*\*;
* пользователям с ролью CLIENT доступ запрещён;
* фильтр JwtFilter корректно извлекает и проверяет токен;
* заголовки безопасности добавляются к каждому ответу;
* тестирование валидации данных.

Все входные данные проходят строгую валидацию на уровне контроллеров с использованием аннотаций JSR-380 (@NotBlank, @Positive). Для проверки этого механизма были разработаны тесты, которые имитируют:

* отправку пустого названия товара;
* некорректную цену (отрицательное число, ноль);
* неверный email или пароль при регистрации и авторизации.

В случае невалидных данных система возвращает понятное сообщение об ошибке и HTTP-статус 400 Bad Request. Все эти сценарии были протестированы и подтверждены.

Централизованная обработка исключений

Система использует @ControllerAdvice для централизованной обработки исключений. Проверялись следующие ситуации:

* попытка зарегистрировать существующего пользователя (UserAlreadyExistsException);
* неверные учетные данные (InvalidCredentialsException);
* некорректные данные о товаре (InvalidProductDataException).

Тестирование показало, что все исключения перехватываются корректно, а клиент получает понятное сообщение и соответствующий HTTP-статус.

Тестирование логики бизнес-процессов

Проверка бизнес-логики проводилась на уровне сервисов. Например:

* при регистрации пользователя пароль шифруется с помощью BCryptPasswordEncoder;
* при добавлении товара цена проверяется на положительное значение;
* при авторизации генерируется JWT-токен с правильным набором полей.

Эти проверки гарантируют, что основные функции системы работают так, как задумано, и обеспечивают защиту от распространённых ошибок.

Интеграционное тестирование

Хотя система представляет собой одно REST API без внешних микросервисов, была проведена проверка взаимодействия между уровнями:

* контроллер → Сервис → Репозиторий;
* фильтры безопасности → Spring Security → Контроллеры.

Это позволило убедиться, что все зависимости внедряются корректно, а политики доступа соблюдаются.

Тестирование логирования

Система логирует все ключевые события, такие как регистрация, вход и добавление товаров. ID пользователя записывается в MDC, что позволяет точно трассировать действия конкретного пользователя. Логирование проверялось через unit-тесты и анализировалось на предмет точности и полноты информации.

### 2.3.7 Вызов и загрузка

Запуск разработанной программы осуществляется стандартным образом для приложений на Spring Boot. Система представляет собой REST API, которое может быть развернуто как локально, так и на сервере. Для запуска не требуется сложных инфраструктур или контейнеров, хотя предусмотрена возможность дальнейшего внедрения Docker для упрощения деплоя.

После этого приложение запускается и становится доступным по адресу http://localhost:8081. Все компоненты системы начинают работу:

* загружается контекст Spring;
* подключается база данных (PostgreSQL);
* инициализируются репозитории, сервисы и контроллеры;
* настройки безопасности применяются к HTTP-запросам;
* внедряются зависимости через механизм DI.

Сразу после запуска система готова принимать HTTP-запросы. Основные эндпоинты становятся доступны:

* POST /auth/register — регистрация нового пользователя;
* POST /auth/login — получение JWT-токена;
* GET /products — получение списка товаров (только для авторизованных);
* POST /products — добавление товара (только для ADMIN и EMPLOYEE).

Все данные, передаваемые между клиентом и сервером, имеют формат JSON, что делает систему удобной для взаимодействия как с фронтенд-приложениями, так и мобильными клиентами.

На этапе запуска программа автоматически подключается к используемой БД. По умолчанию используется встроенная H2-база данных, которая создаётся динамически и не требует предварительной настройки. При необходимости перехода к продакшн-среде можно изменить настройки в файле application.properties или application.yml для использования PostgreSQL.

Таблицы users и products создаются автоматически при первом запуске, если включена опция spring.jpa.hibernate.ddl-auto=update. Это позволяет избежать ручного создания структуры БД и упрощает процесс развертывания.

Использование внешних инструментов

Взаимодействие с API можно выполнять с помощью различных инструментов, таких как:

* Postman — удобен для тестирования запросов, проверки ответов и отладки токенов;
* curl — позволяет отправлять запросы через терминал, что полезно при автоматизации или работе в CI/CD;
* HTTP клиенты из мобильных и веб-приложений — могут использоваться для интеграции с клиентскими приложениями.

Сервер проверяет токен с помощью фильтра JwtFilter, и, если он действителен, запрос направляется в контроллер ProductController.

Логирование и диагностика

Во время работы система записывает ключевые события в логи. Каждый запрос к защищённым эндпоинтам логируется, включая ID пользователя, тип операции и результат выполнения. Логирование выполняется с использованием @Slf4j из Lombok, а ID пользователя сохраняется в MDC для точного трассирования действий конкретного клиента.

Безопасность и производительность

Система спроектирована таким образом, чтобы быть легковесной и быстро запускающейся. Так как используется stateless-архитектура с JWT, нет необходимости управлять сессиями, что снижает нагрузку на сервер. Все механизмы безопасности активируются сразу после запуска:

* проверка токена;
* валидация входных данных;
* защита от CSRF;
* заголовки безопасности.

Это гарантирует, что система работает безопасно с самого начала и не требует дополнительных действий со стороны администратора.

Таким образом, вызов и загрузка программы реализованы простым и понятным способом. Достаточно запустить приложение через Maven или JAR-файл, и система будет готова к работе. Простота настройки, минимальные требования к окружению и использование стандартных протоколов делают программу удобной для тестирования, демонстрации и последующего расширения. Предусмотрена возможность интеграции с Docker, что позволит в будущем масштабировать систему и использовать её в облачных и микросервисных архитектурах.

## 2.4 Руководство оператора

### 2.4.1 Назначение программы

Программа предназначена для демонстрации методологии безопасной разработки программного обеспечения на примере реализации REST API. Она может быть использована студентами, преподавателями, начинающими и опытными разработчиками, которые хотят изучить или применять на практике современные подходы к созданию защищённых веб-приложений.

Целевой аудиторией являются:

* студенты и преподаватели технических специальностей — могут использовать программу как учебный пример реализации безопасной аутентификации, авторизации, шифрования паролей и ограничения доступа по ролям;
* разработчики backend-приложений — могут использовать данный проект как шаблон для создания собственных безопасных REST API сервисов;
* специалисты по безопасности — могут использовать систему как тестовую площадку для анализа распространённых уязвимостей и проверки механизмов защиты;
* тестировщики ПО — система может служить базой для написания и отладки тест-кейсов, связанных с безопасностью и обработкой HTTP-запросов;
* предприниматели и заказчики ПО — могут использовать программу как прототип для дальнейшего масштабирования и внедрения в реальные продукты, такие как интернет-магазины, CRM-системы или платформы управления продуктами.

Основная цель программы — показать, как можно реализовать безопасное REST API, соблюдая принципы современной разработки ПО:

* использование JWT для stateless-аутентификации;
* шифрование паролей через BCryptPasswordEncoder;
* валидация входных данных с помощью JSR-380;
* ограничение доступа по ролям пользователей;
* защита от распространённых уязвимостей (XSS, CSRF, SQL-инъекции);
* централизованная обработка ошибок;
* логирование действий пользователей.
* программа предоставляет пользователям возможность:
* регистрироваться и авторизироваться с выдачей JWT-токена;
* просматривать список товаров;
* добавлять новые товары в зависимости от роли пользователя;
* получать понятные сообщения об ошибках;
* убедиться в корректной обработке запросов и надёжности системы благодаря модульным тестам.

REST API спроектировано таким образом, чтобы его можно было легко интегрировать в мобильные приложения, фронтенд-платформы и микросервисную архитектуру. Все компоненты системы строго разделены: контроллеры, DTO, модели, репозитории, сервисы и фильтры, что делает её читаемой, тестируемой и расширяемой.

Кроме того, программа может быть основой для более сложных решений, таких как интернет-магазины, платформы электронной коммерции, системы управления продуктами и другими сущностями. При необходимости она может быть дополнена новыми эндпоинтами, моделями и функциями, такими как оплата, корзина, история заказов и интеграция с внешними системами.

Таким образом, программа представляет собой готовый пример реализации безопасного REST API, который может быть использован как в образовательных целях, так и в качестве прототипа для последующего развития в профессиональной среде.

### 2.4.2 Условия выполнения

Для корректной работы разработанной программы, реализованной в виде REST API на языке Java с использованием фреймворка Spring Boot, необходимо соблюдение определённых условий программной и аппаратной среды. Хотя система предназначена для демонстрации методологии безопасной разработки и не предъявляет высоких требований к производительности, важно обеспечить минимальные параметры оборудования и окружения для успешного запуска и тестирования.

Требования к оборудованию

Программа может быть запущена на стандартном персональном компьютере или сервере. Рекомендуемые характеристики:

* процессор: минимум 2 ядра (рекомендуется 4 ядра). Подходит любая современная модель Intel или AMD, так как приложение не требует высокой вычислительной мощности;
* оперативная память: минимум 4 ГБ. Рекомендуется 8 ГБ для комфортной работы с IDE и одновременным запуском базы данных;
* место на диске: минимум 5 ГБ свободного места на жёстком диске. Рекомендуется использование SSD для ускорения загрузки приложения и работы с базой данных;
* графическая карта: не требуется, так как программа не связана с обработкой графических данных или обучением нейросетей.

Требования к программному обеспечению

Для запуска и разработки системы необходим следующий набор программного обеспечения:

* Java 17 или выше — основная платформа, на которой работает приложение;
* Maven 3.x — система сборки проекта, обеспечивающая простое подключение зависимостей;
* IntelliJ IDEA (Community или Ultimate) — рекомендуемая интегрированная среда разработки для работы с Spring Boot проектами;
* Git — для управления версиями кода и командной разработки;
* Postman — инструмент для тестирования HTTP-запросов и проверки работы API;
* PostgreSQL — В случае использования PostgreSQL необходим соответствующий сервер баз данных;
* HTTP-сервер — встроен в Spring Boot, поэтому внешний сервер не требуется.

Системные требования

Программа совместима с различными операционными системами:

Windows 10/11

Linux (Ubuntu, Fedora и другие дистрибутивы)

macOS 10.14 и выше

Это позволяет использовать систему в различных средах, что делает её удобной как для локальной разработки, так и для последующего деплоя на сервере.

Платформа исполнения

REST API реализовано с использованием:

* Spring Boot 3.x;
* Spring Security;
* Spring Data JPA;
* Hibernate ORM;
* JWT (JSON Web Token);
* BCryptPasswordEncoder;
* Jakarta Validation (JSR-380).

Все зависимости указаны в pom.xml, что позволяет автоматически подключать библиотеки через Maven.

Режимы работы

Программа может работать в двух режимах:

* Локальная разработка: запуск через IDE (например, IntelliJ IDEA) или с помощью команды. /mvnw spring-boot:run;
* Использование JAR-файла: после сборки проекта с помощью команды. /mvnw package можно запустить приложение с помощью java -jar secure-api.jar.

Условия тестирования

* Для модульного тестирования используются:
* JUnit 5 — для написания и запуска тестов;
* Mockito — для мокирования сервисов и репозиториев;
* MockMvc — для тестирования контроллеров без запуска сервера.

Тесты выполняются в рамках Maven-проекта и запускаются командой. /mvnw test. Все тесты покрывают ключевые сценарии работы системы: регистрацию, авторизацию, работу с товарами и обработку ошибок.

Таким образом, условия выполнения программы являются минимальными и доступными для большинства пользователей. Приложение легко запускается на современных ПК и не требует специализированного оборудования или сложной настройки окружения. Это делает его подходящим как для образовательных целей, так и для практического применения в качестве шаблона безопасного REST API.

### 2.4.3 Работа пользователя

После запуска программы пользователь может взаимодействовать с REST API через HTTP-запросы. Приложение по умолчанию доступно на порту 8080, что позволяет использовать его локально или на сервере без дополнительных изменений конфигурации. Все функции системы реализованы через стандартные методы HTTP, а обмен данными осуществляется в формате JSON.

Доступ к системе

REST API предоставляет следующие эндпоинты:

* POST /auth/register — регистрация нового пользователя;
* POST /auth/login — получение JWT-токена после успешной авторизации;
* GET /products — получение списка товаров (только для авторизованных);
* POST /products — добавление нового товара (только для ADMIN и EMPLOYEE).

Функционал системы

Программа предоставляет пользователю следующие возможности:

* регистрация — создание новой учётной записи с указанием email, пароля, имени и роли (ADMIN, EMPLOYEE, CLIENT);
* авторизация — получение JWT-токена, который используется для последующих запросов;
* получение списка товаров — просмотр всех доступных товаров (только для пользователей с ролью ADMIN или EMPLOYEE);
* добавление товара — создание новых записей о товарах с проверкой данных на корректность;
* обработка ошибок — система возвращает понятные сообщения об ошибках в случае некорректных данных или попытки несанкционированного доступа;
* логирование действий — все ключевые события, такие как регистрация, вход и операции над товарами, записываются в логи для последующего анализа.

Управление пользователями

Система позволяет регистрировать пользователей с разными уровнями доступа:

* CLIENT — в текущей версии имеет только возможность регистрации и авторизации, но не может выполнять никаких действий над товарами;
* EMPLOYEE — может просматривать список товаров и добавлять новые позиции;
* ADMIN — обладает теми же правами, что и EMPLOYEE, и может быть расширен для выполнения административных задач.

Роль пользователя указывается при регистрации и проверяется при каждом обращении к защищённым ресурсам с помощью аннотаций @PreAuthorize("hasRole('ADMIN') or hasRole('EMPLOYEE')").

Ограничение доступа

Система обеспечивает ограничение доступа по ролям:

* эндпоинт /auth/\*\* — открыт для всех (регистрация и вход);
* эндпоинт /products/\*\* — доступен только пользователям с ролью ADMIN или EMPLOYEE;
* для других ролей (например, CLIENT) доступ запрещён, и возвращается HTTP-код 403 Forbidden.

Безопасная передача данных

Программа использует JWT для stateless-аутентификации, что исключает необходимость хранения сессий на сервере и повышает масштабируемость. Пароли шифруются с помощью алгоритма BCrypt перед сохранением в БД. На уровне контроллеров применяются аннотации JSR-380 (@NotBlank, @Positive) для валидации входных данных.

На стороне клиента используются заголовки безопасности:

* Cache-Control: no-store;
* Pragma: no-cache.

Это предотвращает кэширование чувствительной информации браузером или промежуточными прокси.

Централизованная обработка ошибок

Все исключения обрабатываются централизованно с помощью механизма @ControllerAdvice. Это гарантирует, что клиент всегда получает понятное сообщение и соответствующий HTTP-статус. Например:

* UserAlreadyExistsException — возвращает 400 Bad Request;
* InvalidCredentialsException — возвращает 401 Unauthorized;
* InvalidProductDataException — возвращает 400 Bad Request.

Модульное тестирование

Проект покрыт unit-тестами, написанными с использованием JUnit 5 и Mockito. Тестирование проводится без запуска сервера, с помощью MockMvc. Это позволяет проверять работу контроллеров и сервисов в изолированной среде.

Примеры тестовых сценариев:

* проверка успешной регистрации;
* обработка дублирования email;
* валидация входных данных при добавлении товара;
* корректная работа аутентификации и выдача JWT-токена.

Тесты находятся в пакете com.example.SecureAPI.controller и охватывают основные функции программы.

Интеграция с внешними системами

REST API легко интегрируется с другими приложениями:

* мобильные приложения могут использовать API для работы с продуктами и пользователями;
* фронтенд-приложения (React, Vue, Angular) могут использовать данные из /products для отображения каталога;
* системы мониторинга и логирования могут быть интегрированы для сбора метрик и трассировки действий пользователей.

Примеры API-запросов

Ниже представлены основные API-методы, доступные пользователю:

Работа с пользователями

* POST /auth/register — регистрация пользователя с ролью;
* POST /auth/login — получение JWT-токена.

Работа с товарами

* GET /products — получить список товаров (требуется роль ADMIN или EMPLOYEE);
* POST /products — добавить новый товар (требуется роль ADMIN или EMPLOYEE).

Обработка ошибок

* 400 Bad Request — при невалидных данных (например, цена меньше нуля);
* 401 Unauthorized — при попытке входа с неверными учетными данными;
* 403 Forbidden — при попытке доступа к защищённому ресурсу без нужной роли.

### 2.4.4 Техника безопасности при работе на компьютере

Перед началом работы за рабочим местом следует соблюсти ряд требований:

* отрегулировать освещение, сделать удобным для работы;
* проверить правильность подключения оборудования к электросети;
* проверить исправность всех проводов питания и отсутствия оголённых участков проводов;
* протереть антистатической салфеткой поверхность экрана монитора и защитного экран;
* посмотреть правильность расположения экрана, клавиатуры, мышки, при необходимости отрегулировать, а также расположение элементов компьютера;
* убрать рабочее место от грязи и пыли.

Обязательная техника безопасности при работе на ПК требует выполнение следующих нормативов при размещении монитора:

* расстояние между столами и работника составляет не мене 2 метров, а между боковыми поверхностями мониторов – не менее 1,2 метра;
* экран используемого монитора находится на расстоянии от 0,6 до 0,7 метра от глаза работника.

Безопасные правила поведения и техники выполнения трудовых операций приведены в Типовой инструкции ТОИ Р-45-084-01. В зависимости от категории выполняемой работы и уровня зрительной нагрузки, приходящейся на сотрудника в течение смены, суммарное время перерывов в зависимости от ее продолжительности составляет:

* при 8-часовой смене – от 30 до 70 минут;
* при 12-часовой смене – от 70 до 120 минут.

# 3 Экономическая часть

## 3.1 Расчёт заработной платы с отчислением на социальное страхование

### 3.1.1 Определение перечня работ и трудоёмкости разработки программного продукта

Смета затрат на производство представляет собой общий свод плановых расходов в стоимостном выражении, необходимых для выпуска продукции, выполнения работ и оказания услуг в рамках производственной программы предприятия. Разработка сметы осуществляется по единой номенклатуре экономических элементов.

Структура затрат организации зависит от особенностей отрасли и определяется типовыми рекомендациями по учёту и калькулированию себестоимости, разработанными министерствами и ведомствами при согласовании с Минфином РФ. Эти рекомендации применяются в промышленности, сельском хозяйстве, строительстве, торговле, общественном питании и других сферах деятельности.

Затраты включают:

* расходы, связанные с выпуском продукции, выполнением работ и оказанием услуг;
* расходы на управление и реализацию продукции, за исключением капитальных вложений, средств, направленных на освоение новых видов продукции, а также непроизводственных расходов.

Группировка затрат в смете проводится по следующим экономическим элементам:

* материальные затраты, включающие сырьё и материалы, покупные комплектующие, полуфабрикаты, топливо, запасные части и прочие аналогичные расходы;
* оплата труда, которая состоит из заработной платы, премий за достижение показателей, выплат за выслугу лет, резервов на оплату отпусков;

отчисления в социальные фонды;

* амортизационные отчисления, относящиеся к основным фондам, нематериальным активам, малоценным и быстроизнашивающимся предметам;
* прочие расходы, такие как оплата услуг сторонних организаций, хозяйственные издержки, затраты на рекламу и сбыт продукции.

Расчёт сметы выполняется на основе данных бизнес-плана, связанного с производственной программой на будущий финансовый год, анализа фактических показателей предыдущего периода и действующих нормативов по статьям расходов, принятых в организации.

Нормативы по статьям затрат устанавливаются с учётом рыночных цен на продукцию прошедший период. По этим данным рассчитывается максимально допустимый уровень издержек на производство, соответствующий текущему уровню рыночной или договорной цены. Полученные значения позволяют спрогнозировать объём затрат на единицу продукции и составить плановую смету с детализацией по статьям и элементам.

С учётом специфики отраслевой деятельности смета затрат формируется по направлениям работы, видам продукции, услуг или категориям деятельности. Дополнительно возможна группировка по структурным подразделениям.

На завершающем этапе планирования составляется сводная смета, ориентированная на производственную программу и объём выпуска продукции.

На начальном этапе создания программного продукта необходимо определить этапы разработки и оценить трудоёмкость каждого из них.

В таблице 1 представлены все основные показатели.

Таблица 1 – Этапы и трудоёмкость выполнения работ В часах

|  |  |  |
| --- | --- | --- |
| Этапы разработки | Руководитель | Техник |
| Постановка задачи | 3 | 9 |
| Выбор инструментальных средств | 1 | 1 |
| Проектирование | 3 | 15 |
| Построение алгоритма | 2 | 20 |
| Программирование | 8 | 36 |
| Тестирование | 4 | 32 |
| Отладка | 4 | 14 |
| Написание пояснительной записки | 3 | 14 |
| Итого | 28 | 141 |
| 135 | |

Минимальный размер заработной платы в соответствии с установленными нормами составляет 22440 рубля.

Уровень заработной платы специалистов в сфере информационных технологий определён исходя из анализа текущего состояния рынка труда в Новгородской области. Были рассмотрены данные по вакансиям и предложениям о работе, публикуемые в открытых источниках, включая региональные сайты [7] по трудоустройству и отчёты служб занятости населения [5].

Месячный оклад руководителя проекта принят равным 32450 руб., техника – 22800 руб.

### 3.1.2 Расчёт основной заработной платы

При расчёте основной заработной платы учитывается заработная плата всех категорий работников, непосредственно занятых разработкой программного продукта. Размер заработной платы определяется исходя из количества исполнителей и их квалифицированного уровня, а также затраченного ими времени в целом на разработку программного продукта и отдельных его этапов.

Основная заработная плата работников определяется по повременной системе оплаты труда.

Рассчитаем цену одного часа работы сотрудника по формуле:

|  |  |
| --- | --- |
| Цч = Ом/(Др×Ч), | (1) |

где Др – среднее число рабочих дней в месяце;

Ч – часов в смене.

В соответствии с установленными окладами и нормативами рабочего времени, для расчёта часовой ставки используется число рабочих дней в мае 2025 года – 18 дней, согласно производственному календарю Российской Федерации на 2025 год, утверждённому Министерством труда.

Продолжительность рабочей смены принята равной 8 часам, что соответствует стандартной 40-часовой рабочей неделе. Таким образом, произведем расчёт оплаты труда за час.

Руководитель проекта:

Цч = 32450/(18×8) = 225,35 руб.

Техник:

Цч = 22800/(18×8) = 158,33 руб.

Заработная плата работников по окладу рассчитывается по формуле:

|  |  |
| --- | --- |
| ЗПосн = Цч×Ч, | (2) |

где Ч – количество часов, затраченных на разработку программного продукта.

Таблица 2 – Основная заработная плата работников В рублях

|  |  |  |  |
| --- | --- | --- | --- |
| Должность | Количество часов, затраченных на разработку продукта, час | Оплата труда за 1 час | Заработная плата |
| Руководитель проекта | 28 | 225,35 | 6 309,80 |
| Техник | 141 | 158,33 | 22 324,53 |
| Итого | | | 28 634,33 |

Таким образом, по показателям, представленным в таблице 2, основная заработная плата работников составила 28634,33 руб.

Дополнительная заработная плата включает различные виды доплат сверх основной заработной платы: премии, доплату за работу в сверхурочное время, надбавки за профессиональное мастерство, оплату очередного и учебных отпусков и прочие виды доплат.

Дополнительная заработная плата устанавливается на предприятии в процентах от суммы основной заработной платы и составляет от 16 до 20%.

Таким образом, дополнительная заработная плата определяется по следующей формуле:

|  |  |
| --- | --- |
| ЗПдоп = ЗПосн×20%/100%, | (3) |

где ЗПосн – основная заработная плата, руб.

Значение ЗПосн, взятое из таблицы 2, используется для расчёта дополнительной заработной платы:

|  |
| --- |
| ЗПдоп = 28634,33×0,20 = 5 726,86 руб. |

На основании вышеприведенных расчетов, можно сделать вывод, что основная и дополнительно заработная плата работников по созданию информационной системы для кофейни составит 28634,33+5 726,86 = 34361,19 руб.

### 3.1.4 Расчёт отчислений на социальное страхование

Отчисления на социальное страхование составляют 30% от суммы основной и дополнительной заработной платы.

Отчисления на социальное страхование рассчитываются по формуле:

|  |  |
| --- | --- |
| Зстр = (ЗПосн+ЗПдоп)×30%/100%, | (4) |

где ЗПосн – основная заработная плата, руб.;

ЗПдоп – дополнительная заработная плата, руб.

Значение ЗПосн взято из таблицы 2, а дополнительная заработная плата рассчитывается по формуле 3. На основании полученных данных вычислим величину отчислений с использованием формулы 4:

|  |
| --- |
| Зстр = 34361,19×30%/100% = 10308,35 руб. |

Таким образом, из расчетов видно, что отчисления на социальное страхование составят 10308,35 руб.

## 3.2 Расчёт стоимости материалов и лицензионного обеспечения

При создании программного продукта используется лицензионное программное обеспечение, поэтому необходимо включить стоимость лицензионных программ, используемых при разработке программного продукта, в смету затрат.

Перечень лицензионных программ, срок их действия и нормы установки представлены в таблице 3.

Таблица 3 – Перечень программного обеспечения для выполнения работ

|  |  |  |
| --- | --- | --- |
| Наименование лицензионных программ | Норма установки, шт. | Цена, руб. |
| Microsoft Windows 10 Home | 1 | 3150 |
| Intellij IDEA Ultimate | 1 | 47700 |
| МойОфис Персональный | 1 | 1495 |
| Итого | | 52345 |

Стоимость лицензионного программного обеспечения для разработки программного продукта составляет 52345 руб.

Расчёт стоимости необходимых материалов для разработки программного продукта рассчитывается на основе норм расхода материальных ресурсов и оптовых цен на их приобретение по формуле:

|  |  |
| --- | --- |
| См = Нр×Ц, | (5) |

где Нр – норма расхода материальных ресурсов в натуральных единицах;

Ц – цена приобретения за единицу, руб.

Расчёт стоимости материалов представлен в таблице 4.

Таблица 4 – Расчёт стоимости материалов В рублях

|  |  |  |  |
| --- | --- | --- | --- |
| Наименование материала | Норма расхода | Цена за единицу | Сумма |
| Флеш-накопитель USB 8Гб, шт. | 1 | 850 | 850 |
| Бумага, уп. | 0,30 | 347 | 104,10 |
| Картридж, шт. | 1 | 1500 | 1500 |
| Папка, шт. | 1 | 215 | 215 |
| Итого | | | 2 669,10 |

Общая стоимость материалов составила 2 669,10 руб.

Для расчета затрат на электроэнергию учитываются мощность используемого оборудования и время работы над проектом. В данном случае использовался ноутбук Aspire 1 A114-33-P7VD с мощностью 45 Вт, что эквивалентно 0,045 кВт. Общее время работы над проектом для техника составило 141 часов. Потребление электроэнергии за это время рассчитывается как произведение мощности устройства на время работы: 0,045×141 = 6,345 кВт/ч.

Для городских домов с электроплитами, одноставочный тариф второго диапазона равен 4 руб. 21 коп. [8], затраты на электроэнергию составят: 6,345×4,21 = 26,71 руб.

Для обеспечения стабильного подключения к сети использовался безлимитный тариф «Просто Интернет» от компании Ростелеком стоимостью 400 рублей в месяц. Учитывая, что работа над проектом заняла 141 часов, а среднее количество рабочих часов в месяце составляет 160 часов, доля времени работы над проектом относительно общего рабочего времени в месяце определяется как: 125/160 = 0,88.

Таким образом, затраты на интернет за время работы над проектом рассчитываются следующим образом: 400×0,88 = 352 руб.

Общие затраты на электроэнергию и интернет за период работы над проектом составили: 26,71+352 = 378,71 руб.

Итого расходов: 34361,19 + 10308,35 + 52345 + 2 669,10 + 378,71 = 100062,35 руб.

## 3.3 Расчёт накладных расходов

Накладные расходы представляют собой дополнительные к основным расходам затраты на управление, организацию и обслуживание производства. Не связаны напрямую с основным производством товаров или предоставлением услуг, не входят в стоимость материалов и оплату труда.

Накладные расходы закладываются в себестоимость товара, издержки его производства и обращения, но не прямо, а косвенно, пропорционально стоимости материалов и сырья, сумме заработной платы и так далее.

Накладные расходы составляют 10% от расходов на эксплуатацию и рассчитываются по формуле:

|  |  |
| --- | --- |
| Нр = (Рэксп×10%)/100%, | (6) |

где Рэксп – величина расходов на эксплуатацию, руб.

|  |
| --- |
| Нр = (100062,35×10%)/100% = 10006,23 руб. |

На основании выполненных расчетов представлена смета затрат на разработку программного продукта, приведенная в таблице 5.

Таблица 5 – Смета затрат на программный продукт

|  |  |  |
| --- | --- | --- |
| Статьи затрат | Сумма, руб. | Структура затрат, % |
| Основная и дополнительная заработная плата | 34361,19 | 31,22 |
| Отчисления на социальное страхование | 10308,35 | 9,37 |
| Стоимость материалов | 2 669,10 | 2,42 |
| Стоимость лицензионного обеспечения | 52345 | 47,56 |
| Стоимость электроэнергии и интернета | 378,71 | 0,34 |
| Итог расходы на эксплуатацию | 100062,35 | 90,91 |
| Накладные расходы | 10006,23 | 9,09 |
| Итого | 110068,58 | 100 |

Таким образом, общая сумма затрат на разработку информационной системы для кофейни составит 110068,58 руб.

Результаты расчетов визуализированы в виде диаграммы структуры затрат на программный продукт, показанной на рисунке 7.

Рисунок 7 – Структура затрат на программный продукт

## 3.4 Составление и расчёт цены реализации программного продукта

Цена является одним из самых важных экономических показателей. Её основная функция состоит в обеспечении выручки от реализации программного продукта, поэтому цена определяет прибыль и финансовую стабильность предприятия, его жизнеспособность.

Так как целью предпринимательской деятельности является получение прибыли, то цена реализации программного продукта представляет собой сумму затрат на его разработку и запланированную величину прибыли. Установлен уровень рентабельности в размере 25%. Общие затраты на разработку программного продукта принимаются из таблицы 5, а данные о стоимости материалов из таблицы 4.

Расчёт оптовой цены производится по формуле:

|  |  |
| --- | --- |
| Цопт = З×(1+Р), | (7) |

где З – общие затраты на разработку программного продукта, руб.;

Р – уровень рентабельности проекта, коэффициент.

Расчет оптовой цены выполнен с использованием формулы 7:

|  |  |
| --- | --- |
| Цопт = 110068,58×(1+0,25) = 137585,72 руб. |  |

Расчёт прибыли от продажи продукта производится по формуле:

|  |  |
| --- | --- |
| П = Цопт - З, | (8) |
| П = 137585,72-110068,58 = 27517,14 руб. |  |

Таким образом, можно заключить, что разработка программного продукта является экономически целесообразной, поскольку обеспечивает получение прибыли. Величина прибыли от реализации составила 27 517 руб. 14 коп.

# Заключение

Разработанная программа представляет собой REST API, реализованное на языке Java с использованием фреймворка Spring Boot. Основной целью проекта являлась демонстрация методологии безопасной разработки программного обеспечения, а также создание примера системы, в которой все ключевые аспекты безопасности внедрены ещё на этапе проектирования и реализации.

Программа успешно решает задачи регистрации пользователей, авторизации через JWT, ограничения доступа по ролям, шифрования паролей, валидации входных данных и защиты от распространённых уязвимостей, таких как XSS и CSRF-атаки. Все эти меры обеспечивают высокий уровень защищённости при работе с чувствительными данными и позволяют использовать систему как прототип для более сложных решений — интернет-магазине, CRM-систем или сервисов управления продуктами.

Архитектура приложения спроектирована с учётом принципов модульности и разделения ответственности. Контроллеры, сервисы, репозитории, DTO, модели и фильтры строго изолированы друг от друга, что делает код читаемым, тестируемым и расширяемым. Такой подход позволяет легко добавлять новые функциональные модули и интегрировать дополнительные компоненты без изменения существующих частей программы.

Для хранения данных используется реляционная система управления базами данных PostgreSQL , которая может быть применена вместо встроенной H2. Это гарантирует стабильность, производительность и надёжность при переходе к продакшн-среде. Благодаря использованию ORM (Hibernate / Spring Data JPA) работа с базой данных осуществляется простым и безопасным способом, исключая необходимость написания SQL-запросов вручную и снижая риск SQL-инъекций.

Особое внимание было уделено тестированию. Программа покрыта unit-тестами, написанными с использованием JUnit 5 и Mockito, которые проверяют корректность регистрации, аутентификации, работы с товарами и обработки ошибок. Тестирование проводилось без запуска сервера с помощью MockMvc, что позволило заранее выявить возможные проблемы и гарантировать стабильность работы всех компонентов.

Система использует JWT (JSON Web Token) для stateless-аутентификации, что повышает масштабируемость и упрощает управление сессиями. Пароли пользователей сохраняются в зашифрованном виде с использованием алгоритма BCrypt, что предотвращает их восстановление даже в случае компрометации базы данных. Входные данные проверяются на уровне контроллеров с помощью аннотаций JSR-380, такие как @NotBlank и @Positive, что исключает возможность передачи некорректных значений в бизнес-логику.

Настройки безопасности централизованно управляются через класс SecurityConfig. Здесь определены политики доступа, заголовки безопасности (Content-Security-Policy, Cache-Control, Pragma) и параметры сессии (stateless). Фильтр JwtFilter проверяет наличие и валидность токена до начала выполнения контроллера, что обеспечивает защиту всех защищённых эндпоинтов.

Также реализована централизованная обработка ошибок с помощью механизма @ControllerAdvice. Все исключения, возникающие в процессе работы системы, перехватываются и преобразуются в понятные HTTP-ответы с соответствующими статусами. Это делает использование API более прозрачным и удобным как для разработчиков, так и для клиентских приложений.

Логирование действий пользователей выполнено с использованием библиотеки SLF4J и аннотации @Slf4j из Lombok. ID пользователя записывается в MDC, что позволяет точно трассировать действия конкретного клиента и анализировать логи для диагностики проблем.

# Методология

В рамках данного дипломного проекта была разработана и исследована методология безопасной разработки программного обеспечения на примере реализации простого, но защищённого REST API на языке Java с использованием Spring Boot. В процессе создания системы были выделены ключевые принципы и практики, которые следует соблюдать при разработке любого веб-приложения, особенно если оно работает с чувствительными данными или требует авторизации пользователей.

* Аутентификация и авторизация

1. используйте JWT для stateless-аутентификации, особенно если планируется масштабируемость;
2. не храните сессии на сервере, это снижает нагрузку и повышает отказоустойчивость;
3. ограничивайте доступ к эндпоинтам через аннотации @PreAuthorize и политики Spring Security;
4. разграничивайте роли пользователей (например, ADMIN, EMPLOYEE, CLIENT) и предоставляйте доступ только в зависимости от роли.

* Шифрование и безопасное хранение данных

1. пароли никогда не сохраняются в открытом виде. Используйте BCryptPasswordEncoder или аналогичные механизмы;
2. используйте реляционную БД с поддержкой транзакций и целостности данных (PostgreSQL, MySQL);
3. храните секретный ключ для JWT в защищённом месте , избегая его жёсткого кодирования в классах;
4. убирайте кэширование чувствительных данных.

* Защита от распространённых уязвимостей

1. отключайте CSRF , если используете stateless-аутентификацию;
2. используйте Content-Security-Policy для защиты от XSS;
3. всегда проверяйте входящие данные на уровне контроллеров , чтобы предотвратить SQL-инъекции и некорректный ввод.

* Работа с ролями и фильтрами

1. создавайте собственный фильтр (например, JwtFilter) , который проверяет наличие и валидность токена до начала выполнения бизнес-логики;
2. добавляйте роль в токен и используйте её в политике доступа;
3. записывайте ID пользователя в MDC для последующего логирования действий.

* Логирование и диагностика

1. логируйте все ключевые события (регистрация, вход, добавление товаров);
2. используйте MDC для трассировки действий конкретного пользователя ;
3. не записывайте в логи чувствительные данные , такие как пароли или токены.

* Обработка ошибок

1. централизованно обрабатывайте исключения с помощью @ControllerAdvice;
2. возвращайте клиенту понятные сообщения об ошибках и соответствующие HTTP-статусы (400, 401, 403, 500 и т.д.);

* Тестирование

1. покрывайте контроллеры и сервисы unit-тестами с использованием JUnit 5 и Mockito;
2. тестируйте работу с токенами, валидацию данных и политики безопасности ;
3. используйте MockMvc для тестирования HTTP-запросов без запуска сервера.

* Архитектурные рекомендации

1. разделяйте уровни приложения : контроллеры, DTO, модели, репозитории, сервисы, фильтры;
2. не передавайте модель напрямую клиенту — используйте DTO для контроля выходных данных;
3. внедряйте зависимости через конструкторы, чтобы повысить тестируемость и читаемость кода.

* Безопасная разработка

1. применяйте принцип минимальных привилегий : каждый пользователь должен иметь доступ только к тем ресурсам, которые ему действительно нужны;
2. используйте строгую типизацию и валидацию — это помогает находить ошибки ещё на этапе компиляции и ввода данных;
3. используйте современные библиотеки безопасности — Spring Security, JWT, Lombok.

* Расширяемость и переносимость

1. проектируйте систему так, чтобы она могла быть расширена : добавление новых ролей, сущностей и прав доступа не должно требовать глубоких изменений;
2. поддерживайте гибкость конфигурации : замена H2 на PostgreSQL должна быть простой;
3. п редусматривайте возможность контейнеризации с помощью Docker.

Данную методологию нельзя считать завершенной, и в последствии набор правил для безопасной разработки необходимо будет расширять и редактировать. Также возможно расширить данный сборник на другие языки программирования, плюс ещё добавить пункты, связанные с работой экосистем языков программирования.

# Список литературы

1. ГОСТ 2.701—2008 Единая система конструкторской документации. Схемы. Виды и типы. Общие требования к выполнению. – М.: Стандартинформ, 2009. – 15 с. (Дата обращения: 15.03.2025);
2. ГОСТ 2.743—91 Единая система конструкторской документации. Обозначения условные графические в схемах. Элементы цифровой техники. – М.: Стандартинформ, 2009. – 62 с. (Дата обращения: 15.04.2025);
3. ГОСТ Р 7.0.97—2016. Требования к оформлению документов. (Дата обращения: 15.09.2024) 1. ГОСТ 2.701—2008 Единая система конструкторской документации. Схемы. Виды и типы. Общие требования к выполнению. – М.: Стандартинформ, 2009. – 15 с.
4. ГОСТ 2.743—91 Единая система конструкторской документации. Обозначения условные графические в схемах. Элементы цифровой техники. – М.: Стандартинформ, 2009. – 62 с.
5. ГОСТ Р 7.0.97—2016. Требования к оформлению документов.
6. Walls C. Spring Boot in Action . – Shelter Island: Manning Publications, 2016. – 264 с.
7. Sharma S., Verma V. Spring Boot: Beginner's Guide . – Birmingham: Packt Publishing, 2018. – 316 с.
8. Hunter R. Spring Security Essentials . – Birmingham: Packt Publishing, 2015. – 150 с.
9. Heffner J., Laranjeira R. PostgreSQL: Up and Running . – Sebastopol: O'Reilly Media, 2017. – 322 с.
10. Zed Y. Practical API Security: Securing REST APIs from Attackers . – Berkeley: Apress, 2016. – 160 с.
11. Steel C., Nagappan M., Woods D. Securing DevOps: Security for Cloud-based Applications and Infrastructures . – San Francisco: No Starch Press, 2017. – 342 с.
12. OWASP Foundation. OWASP API Security Top 10 . – [Электронный ресурс] – URL: https://owasp.org/www-project-api-security/ (Дата обращения: 30.10.2024).

# Приложение А

(Обязательное)

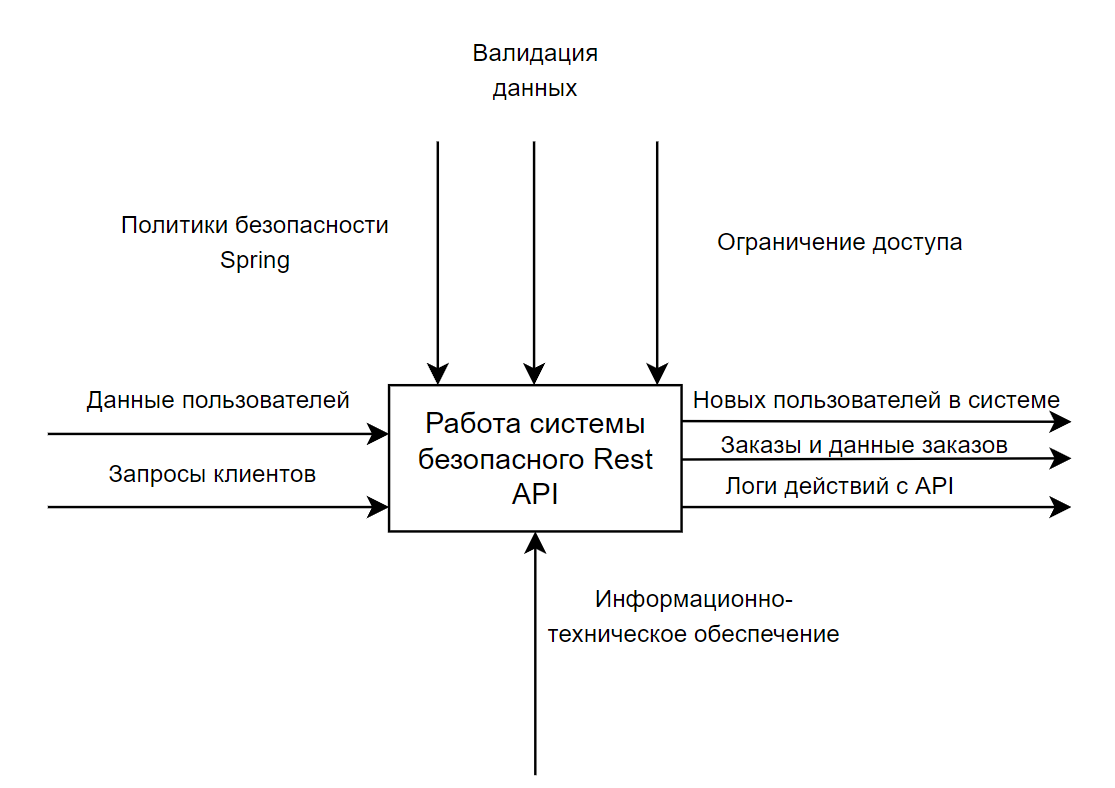


Рисунок А.1 – IDEF0 диаграмма

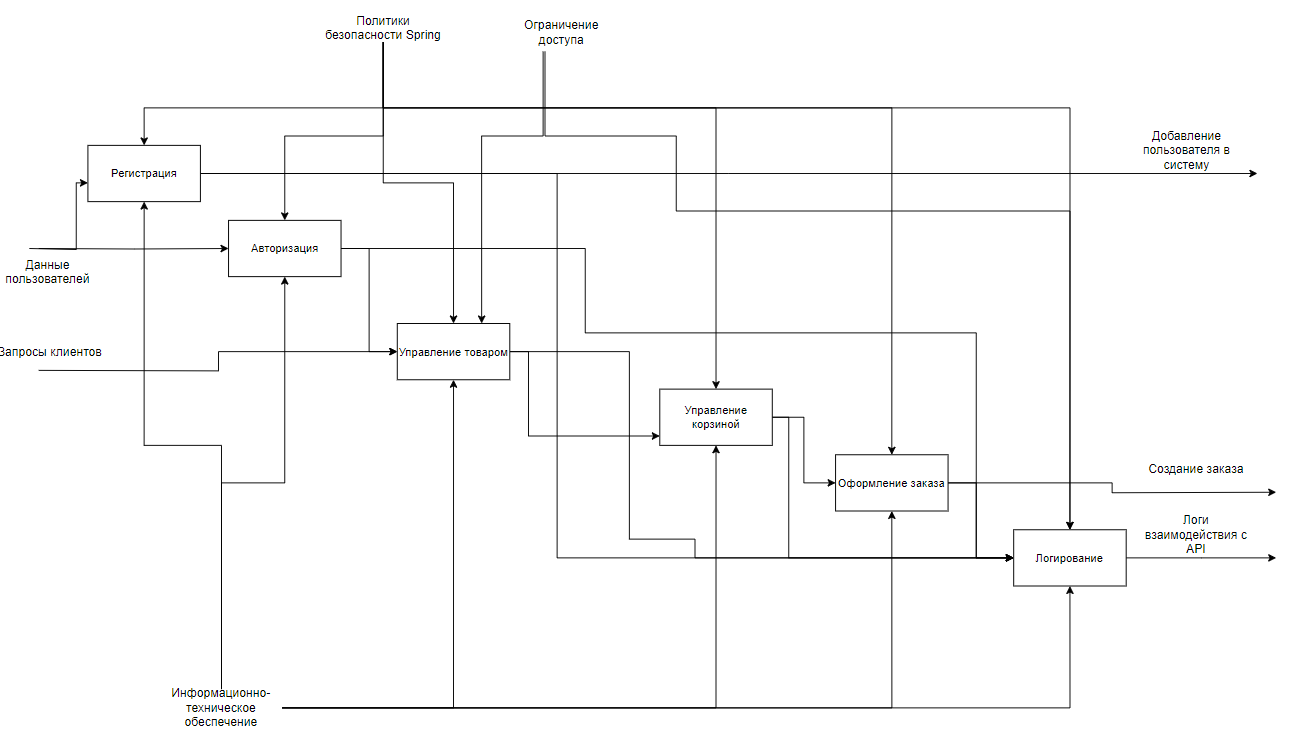


Рисунок А.2 – Декомпозиция IDEF0 модели расчета стоимости заказа

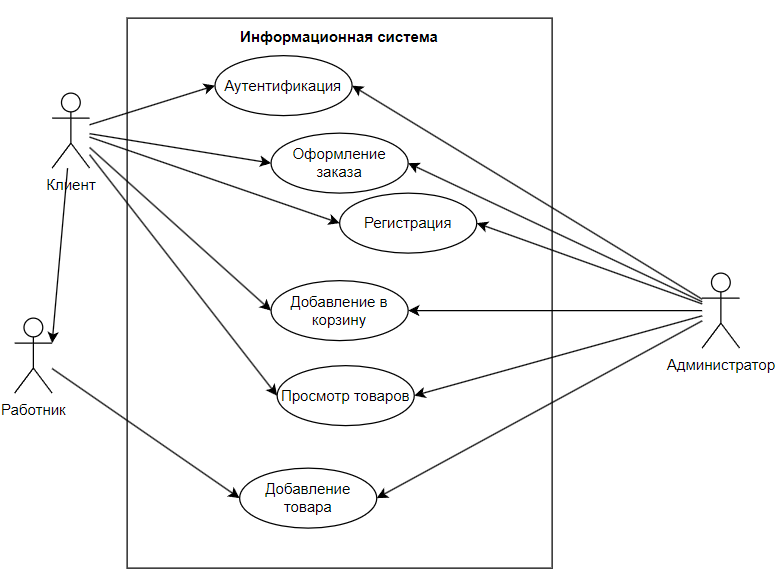


Рисунок А.3 – Диаграмма прецендентов

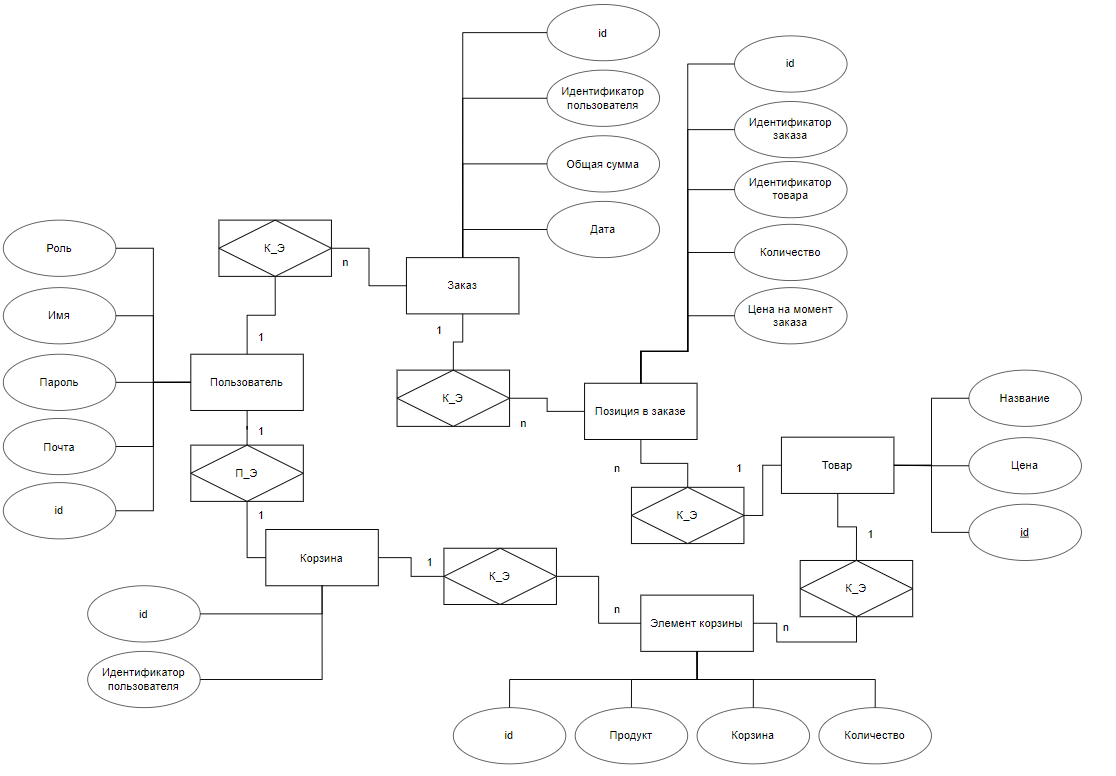


Рисунок А.4 – ER модель базы данных

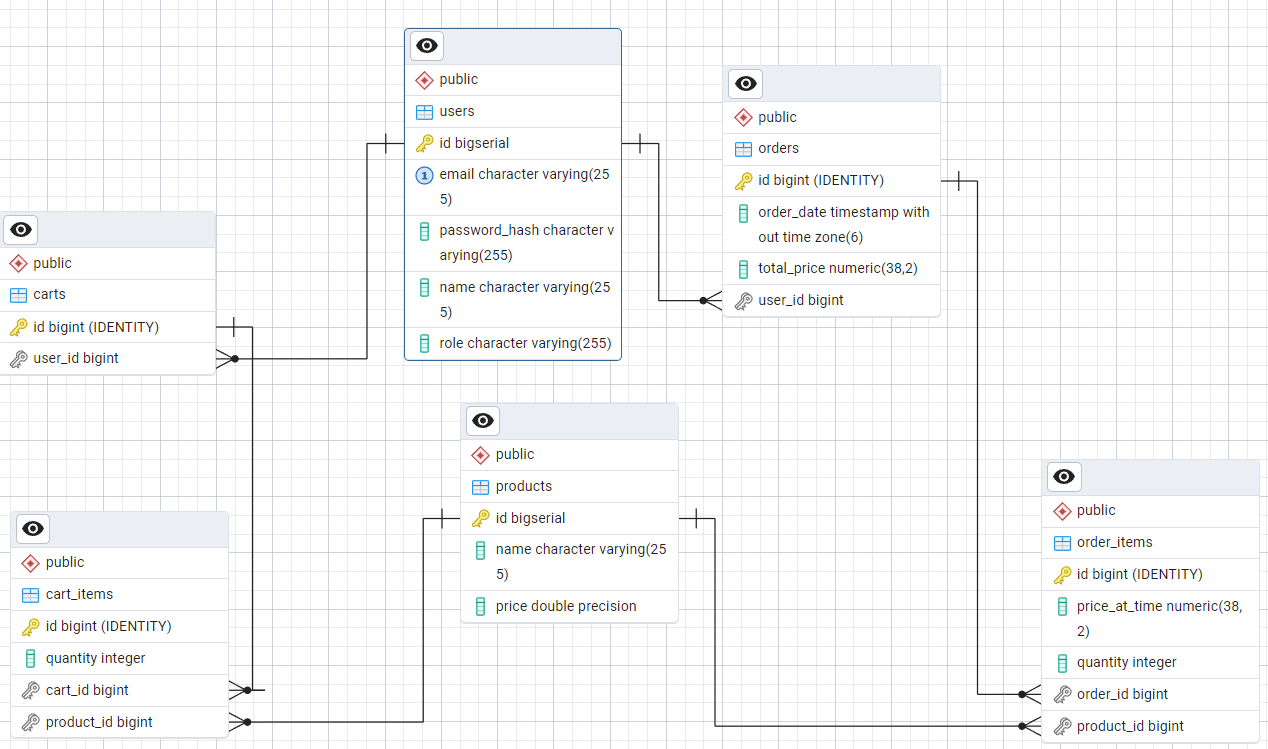


Рисунок А.5 – Схема данных

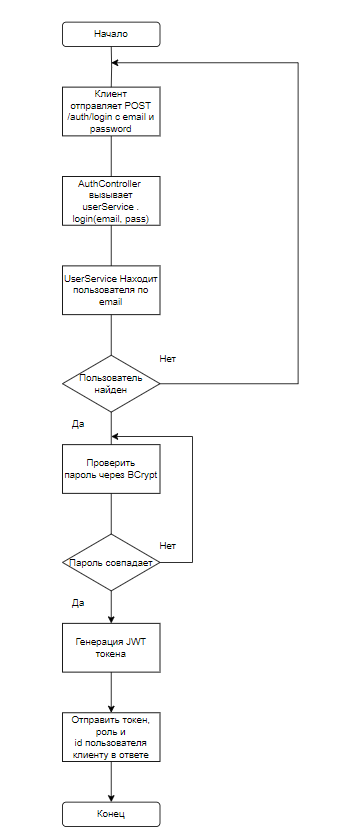


Рисунок А.6 – Блок-схема алгоритма JWT аутентификации

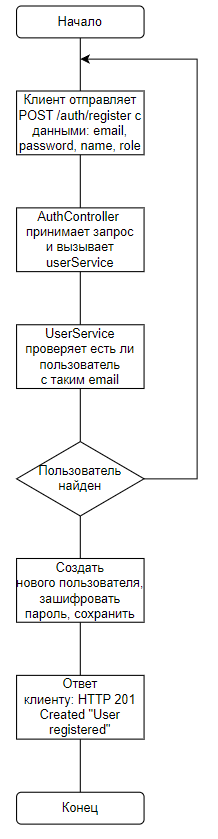


Рисунок А.7 – Блок-схема алгоритма регистрации пользователя

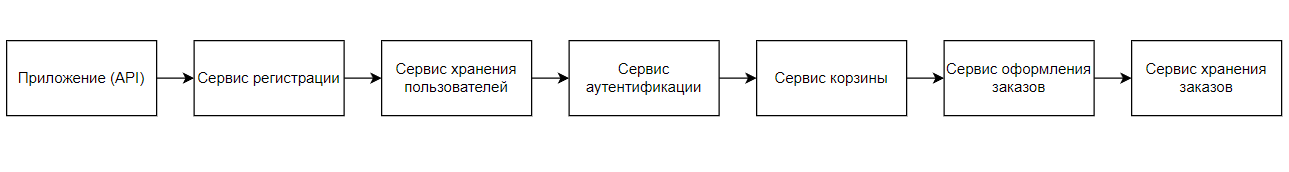


Рисунок А.8 – Диаграмма взаимодействие модулей клиента

# Приложение Б

(обязательное)

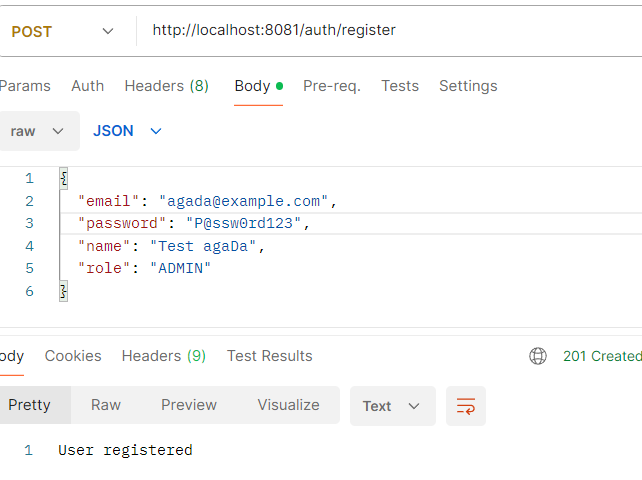


Рисунок Б.1 – тестирование регистрации в Postman

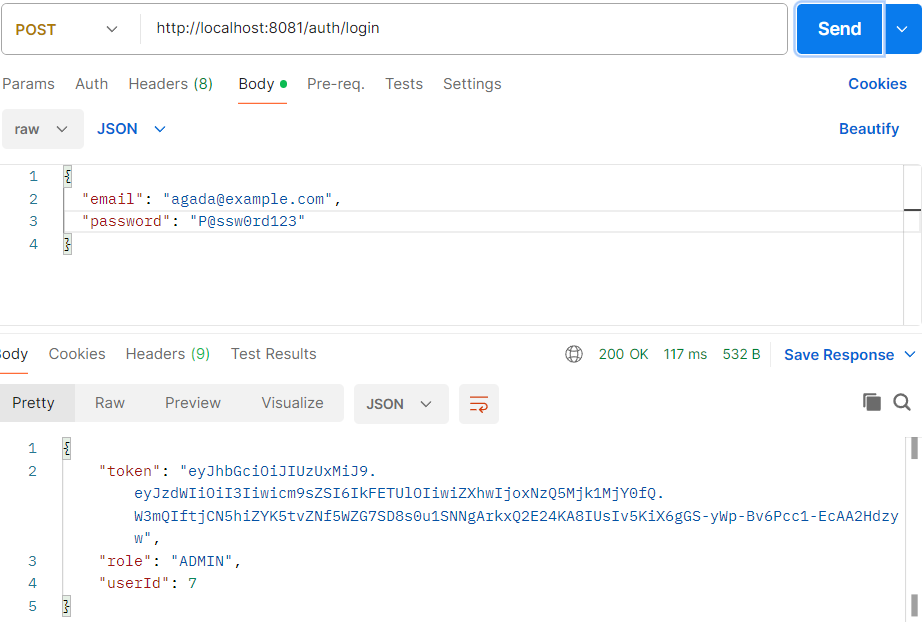


Рисунок Б.2 – тестирование авторизации в Postman

# Приложение В

(обязательное)

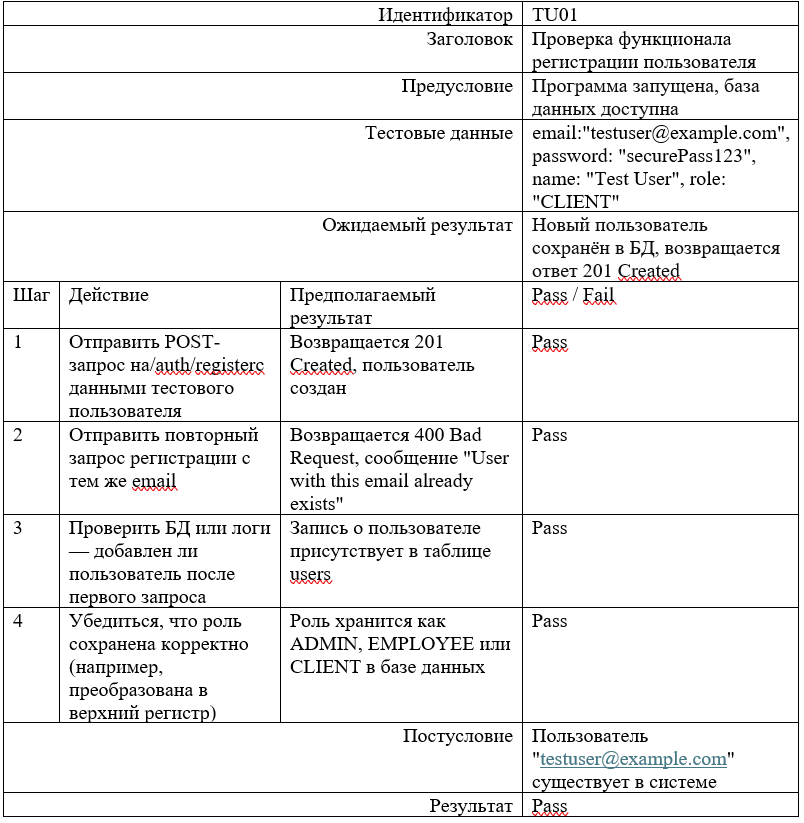


Рисунок В.1 – Тест-кейс для проверки функционала регистрации

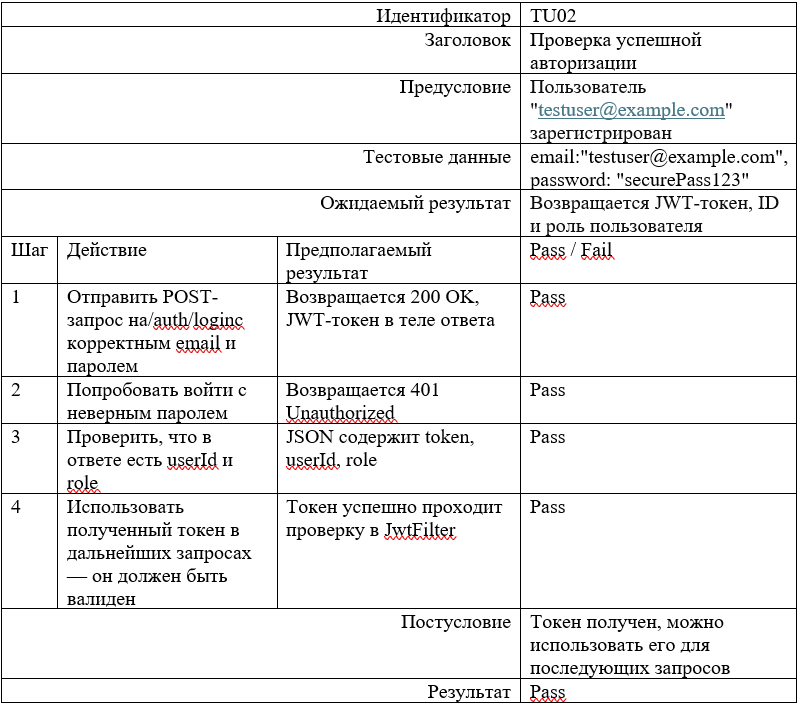


Рисунок В.2 – Тест-кейс для проверки функционала авторизации

# Приложение Г

(обязательное)

Г.1 Листинг. SQL код для создания объектов базы данных

-- 1. Таблица пользователей

CREATE TABLE users (

id BIGSERIAL PRIMARY KEY,

email VARCHAR(255) UNIQUE NOT NULL,

password\_hash VARCHAR(255) NOT NULL,

name VARCHAR(255),

role VARCHAR(50) NOT NULL CHECK (role IN ('CLIENT', 'EMPLOYEE', 'ADMIN'))

);

-- 2. Таблица товаров

CREATE TABLE products (

id BIGSERIAL PRIMARY KEY,

name VARCHAR(255) NOT NULL,

price DOUBLE PRECISION NOT NULL CHECK (price > 0)

);

-- 3. Таблица корзин

CREATE TABLE carts (

id BIGSERIAL PRIMARY KEY,

user\_id BIGINT UNIQUE NOT NULL REFERENCES users(id) ON DELETE CASCADE

);

-- 4. Таблица элементов корзины

CREATE TABLE cart\_items (

id BIGSERIAL PRIMARY KEY,

cart\_id BIGINT NOT NULL REFERENCES carts(id) ON DELETE CASCADE,

product\_id BIGINT NOT NULL REFERENCES products(id) ON DELETE RESTRICT,

quantity INTEGER NOT NULL CHECK (quantity > 0)

);

-- Индекс для поиска по cart\_id

CREATE INDEX idx\_cart\_items\_cart\_id ON cart\_items(cart\_id);

-- 5. Таблица заказов

CREATE TABLE orders (

id BIGSERIAL PRIMARY KEY,

user\_id BIGINT NOT NULL REFERENCES users(id) ON DELETE CASCADE,

total\_price NUMERIC(19, 2) NOT NULL CHECK (total\_price >= 0),

order\_date TIMESTAMP NOT NULL DEFAULT NOW()

);

-- 6. Таблица элементов заказа

CREATE TABLE order\_items (

id BIGSERIAL PRIMARY KEY,

order\_id BIGINT NOT NULL REFERENCES orders(id) ON DELETE CASCADE,

product\_id BIGINT NOT NULL REFERENCES products(id) ON DELETE RESTRICT,

quantity INTEGER NOT NULL CHECK (quantity > 0),

price\_at\_time NUMERIC(19, 2) NOT NULL CHECK (price\_at\_time >= 0)

);