

Пенза, 2024

Федеральное государственное бюджетное образовательное учреждение
высшего образования
ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ

Факультет
Вычислительной техники

Кафедра
МО и ПЭВМ

«Утверждаю»
Заведующий кафедрой
_____ А. Ю. Козлов
«25» декабря 2023 г

**ЗАДАНИЕ
НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ
БАКАЛАВРА**

1. Студент Лобанов Артём Дмитриевич гр. 20ВП1 факультета ВТ
<Фамилия, имя, отчество полностью>

2. Тема работы Программные средства администрирования для приложения
электронной коммерции по продаже 3d-продукции

Тема утверждена приказом ПГУ № _____ от "___" _____ 2024 г.

3. Руководитель работы к.т.н., доцент кафедры «МО и ПЭВМ» Балашова И.Ю.

4. Задание на работу (назначение разработки, исходные данные и т. п.)

1) *Назначение: сервисы регистрации, авторизации и аутентификации пользователя*

2) *Среда выполнения: веб-браузер, Linux, Windows*

Языковые средства разработки: Java, TypeScript, UML

Инструментальные средства: IntelliJ idea, WebStorm, PostgreSQL 16, npm, Draw.io

Технологии разработки: ООП, сервисно-ориентированный подход

3) *Основные функции:*

а) регистрация, авторизация и аутентификация;

б) просмотр профиля пользователя;

в) обеспечение безопасной передачи токенов доступа;

г) администрирование пользователей;

е) принятие и отклонение заказов;

д) завершение заказа.

5. Перечень подлежащих разработке вопросов:

1) <i>Анализ предметной области и постановка задачи</i>
2) <i>Проектирование</i>
3) <i>Реализация</i>
4) <i>Контроль качества программного обеспечения</i>
5) <i>План разработки бюджета</i>
6) <i>Оформление пояснительной записки</i>

6. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1	Анализ предметной области и требований к системе	10%	08.04.2024 – 13.04.2024	
2	Проектирование программных средств	20%	15.04.2024 – 25.04.2024	
3	Реализация программных средств	30%	26.04.2024 – 12.05.2024	
4	Тестирование программных средств	20%	13.05.2024 – 22.05.2024	
5	Оформление пояснительной записки	20%	23.05.2024 – 03.06.2024	

Дата выдачи задания «21» декабря 2023г.

Руководитель бакалаврской работы _____
(подпись, дата)

Задание к исполнению принял студент _____
(подпись, дата)

Работу к защите допустить
Декан факультета _____ Л.Р. Фионова
(подпись, дата)

Реферат

Пояснительная записка содержит 74 листа, 40 рисунков, 13 таблиц, 22 использованных источника и 4 приложения.

ПРИЛОЖЕНИЕ ЭЛЕКТРОННОЙ КОММЕРЦИИ, КЛИЕНТ, СЕРВЕР АВТОРИЗАЦИИ, ПРОКСИ СЕРВЕР, СЕРВЕР БАЗЫ ДАННЫХ, UML, REST, MVC.

Объект исследования – приложение электронной коммерции.

Цель выпускной квалификационной работы – разработка программных средств администрирования для приложения электронной коммерции по продаже 3d-изделий.

Технологии разработки – разработка программных средств проводилась на языке программирования Java с использованием фреймворка Spring и ORM Hibernate, клиентская часть разрабатывалась на языке JavaScript с использованием фреймворка React.js, для создания сервера БД использовался PostgreSQL 16. В качестве среды разработки использовался редакторы программного кода IntelliJ idea и WebStorm.

Среда исполнения – Linux, веб-браузер, Windows.

Результаты работы – в ходе выполнения выпускной квалификационной работы были созданы клиентская и серверная часть программных средств администрирования электронной коммерции по продаже 3d-изделий. Все требования к разработке приложения соблюдены.

					ПГУ 09.03.04– 8BP201.15 ПЗ			
И	Ли	№ докум.	Подп	Да	Программные средства администрирования для приложения электронной коммерции по продаже 3d-продукции	Лит.	Лист	Листов
Разраб.		Лобанов А.Д.						
Пров.		Балашова И.Ю.					4	74
						Группа 20ВП1		
Н. Контр.		Такташкин Д.В						
Утв.		Козлов А. Ю.						

Содержание

Введение.....	7
1 Анализ приложений электронной коммерции	8
1.1 Понятие приложений электронной коммерции	8
1.2 Сравнительный анализ платформ электронной коммерции	9
1.3 Постановка задачи	13
1.4 Анализ требований.....	13
1.4.1 Функциональные требования	14
1.4.2 Нефункциональные требования	19
2 Проектирование программных средств администрирования для приложения электронной коммерции	21
2.1 Архитектура приложения.....	21
2.2 Проектирование структур данных	23
2.2.1 Логическая модель базы данных	23
2.2.2 Физическая модель базы данных	25
2.3 Проектирование программных средств.....	26
2.3.1 Проектирование алгоритма доступа к защищенным данным.....	26
2.3.2 Проектирование классов прокси сервера	27
2.3.3 Проектирование классов сервера авторизации.....	28
2.4 Проектирование пользовательского интерфейса веб-приложения	29
2.5 Проектирование клиентского приложения	32
3 Реализация программных средств администрирования для приложения электронной коммерции	34
3.1 Реализация сервера авторизации.....	34
3.2 Реализация прокси сервера	35

3.3 Развертывание базы данных	36
3.4 Спецификация конечных точек	37
3.5 Развертывание программных средств.....	37
4 Контроль качества программного обеспечения.....	39
4.1 Функциональное тестирование.....	39
4.2 Метрики кода.....	42
4.2.1 Метрика SLOC	42
4.2.2 Цикломатическая сложность	43
4.2.3 Метрика Чепина	44
4.3 Планирование разработки и оценка бюджета.....	45
Заключение	48
Список использованных источников	49
Приложение А (справочное) Глоссарий	52
Приложение Б (рекомендуемое) Листинг приложения	54
Приложение В (рекомендуемое) Результаты тестирования.....	69
Приложение Г (рекомендуемое) Диаграмма классов сервера авторизации	73

Введение

Развитие технологий на современном этапе привело к цифровизации множества различных сфер нашей жизни. Торговля также не осталась в стороне. На сегодняшний день большинство товаров и услуг можно приобрести на различных площадках электронной коммерции. Множество компаний приспособиливаются к прогрессивному рынку торговли и создают собственные площадки электронной коммерции.

Электронная коммерция (e-commerce) – это сфера цифровой экономики, которая включает в себя все финансовые и торговые транзакции, осуществляемые при помощи компьютерных сетей, и бизнес-процессы, связанные с проведением таких транзакций. В e-commerce входят процессы купли-продажи, интернет-маркетинг, электронный обмен данными, финансовые операции и другое.

В постоянно растущем мире электронной коммерции безопасность данных покупателей стала главным приоритетом для бизнеса и потребителей. Угрозы онлайн-безопасности постоянно развиваются, и предприятия электронной коммерции должны быть в курсе лучших практик и новейших технологий для защиты конфиденциальной информации своих клиентов [1]. Таким образом, актуальной задачей является разработка программных средств администрирования для приложения электронной коммерции.

Объектом исследования являются приложения электронной коммерции.

Предметом исследования является программные средства администрирования для приложения электронной коммерции по продаже 3d-изделий.

Целью выпускной квалификационной работы является разработка программных средств администрирования пользователей для приложения электронной коммерции по продаже 3d-изделий. Разработка включает в себя анализ требований, проектирование, реализацию и тестирование программного обеспечения.

1 Анализ приложений электронной коммерции

1.1 Понятие приложений электронной коммерции

Приложение электронной коммерции – это платформа для реализации различных товаров и услуг. Широко известны такие виды электронной коммерции, как:

- глобальные маркетплейсы мирового масштаба, такие как Aliexpress и Amazon или локального, например Ozon и Wildberries;
- онлайн-магазины, реализующие товары в определенной категории, такие как интернет-магазины одежды, стройматериалов или электроники, а также интернет-магазины по продаже 3d-изделий;
- магазины программного обеспечения, продающие приложения или программные средства для ПК и смартфонов, такие, как AppStore, Google Play, глобальная корпорация Microsoft и ряд других;
- специализированные сервисы продажи бизнес-услуг и программ для корпоративных клиентов, продающие товары, необходимые для обеспечения различных бизнес-процессов клиентов, например, российская компания «1С», магазины разработчиков CMS и тем оформления сайтов, брокерских программ;
- службы доставки товаров, например, доставка готовой еды из ресторанов или продуктов из магазина, а также многие другие категории товаров;
- онлайн-банкинг, системы электронных платежей, а также банковские приложения или услуги кредитных организаций;
- услуги перевозок, такие как службы онлайн вызова такси и каршеринга или же службы перевозок различного типа грузов;
- сервисы для интернет-маркетинга, организованные в качестве частей крупных социальных сетей, дающие возможность вести бизнес с использованием специальных категорий аккаунтов и прочие виды электронной коммерции.

Большинство приложений электронной коммерции имеют программные средства администрирования. Основными задачами программных средств администрирования для приложения электронной коммерции являются:

- организация безопасности приложения электронной коммерции, хранение конфиденциальных данных пользователей;
- разграничение доступа к ресурсам;
- регистрация новых пользователей в приложении;
- аутентификация и авторизация пользователей с использованием одного из распространенных протоколов OAuth 2.1 [2] и его расширения OpenID Connect 1.0 [3];
- предоставление личных данных пользователя.

1.2 Сравнительный анализ платформ электронной коммерции

Рассмотрим популярные сервисы электронной коммерции по продаже 3d-изделий.

Целью веб-приложения 3DMF является реализация товаров, напечатанных на заказ [4]. Интерфейс приложения представлен на рисунке 1.

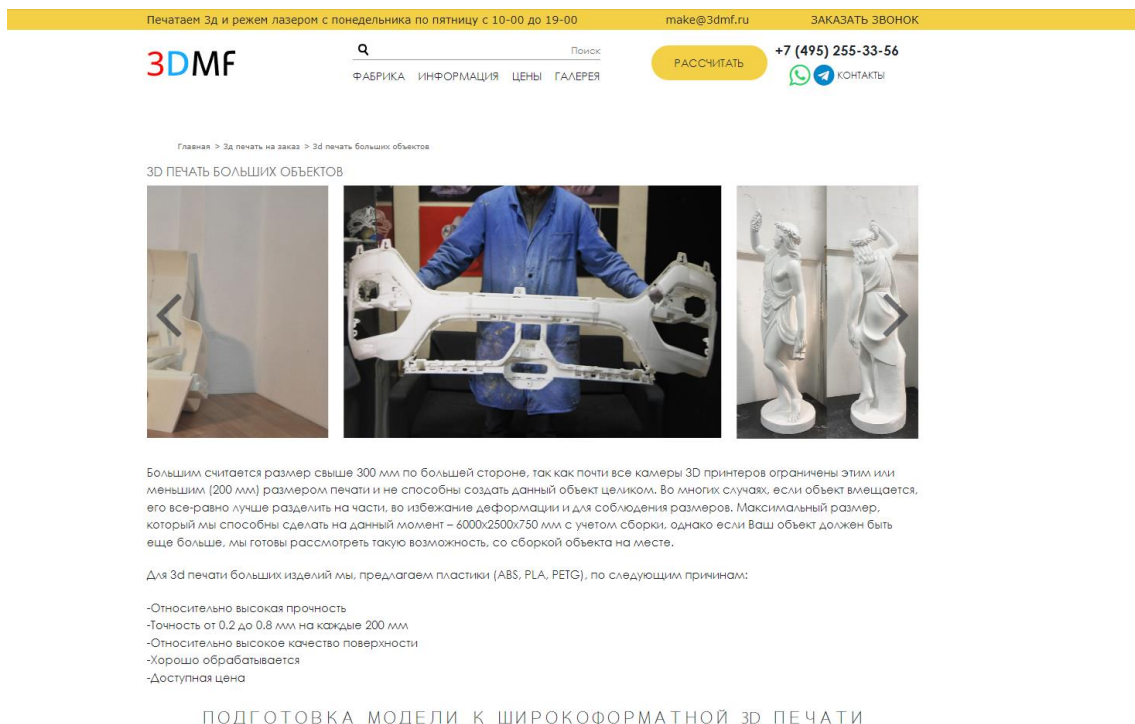


Рисунок 1 – Главная страница 3DMF

Преимуществом данного приложения является удобство при оформлении индивидуального заказа. Вместе с тем можно отметить следующие недостатки:

- отсутствие возможности покупки типовых товаров;
- отсутствие технологии OAuth2.1 при аутентификации и авторизации пользователей;
- отсутствие возможности создания личного кабинета пользователя.

Приложение 3dtoday является порталом для любителей и профессионалов, заинтересованных в 3d-печати и сопутствующих технологиях [5].

На рисунке 2 представлена главная страница приложения.

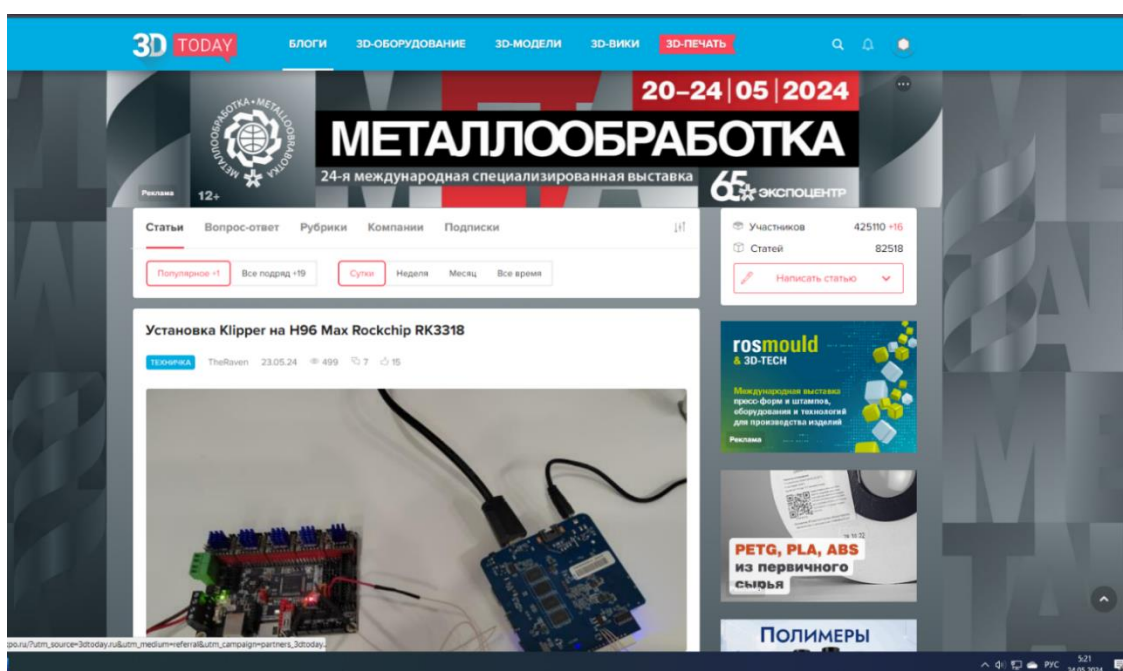


Рисунок 2 – Главная страница 3dtoday

Преимущества:

- большой выбор типовых товаров;
- реализована регистрация, аутентификация и авторизация пользователей;
- реализована регистрация с подтверждением через почту.

Недостатки:

- отсутствие технологии OAuth2.1 при аутентификации и авторизации пользователей;
- отсутствие истории заказов пользователя.

Веб-приложение Volprint позволяет оформить заказы на 3d-изделия разных типов и цветов [6]. На рисунке 3 представлена главная страница приложения.

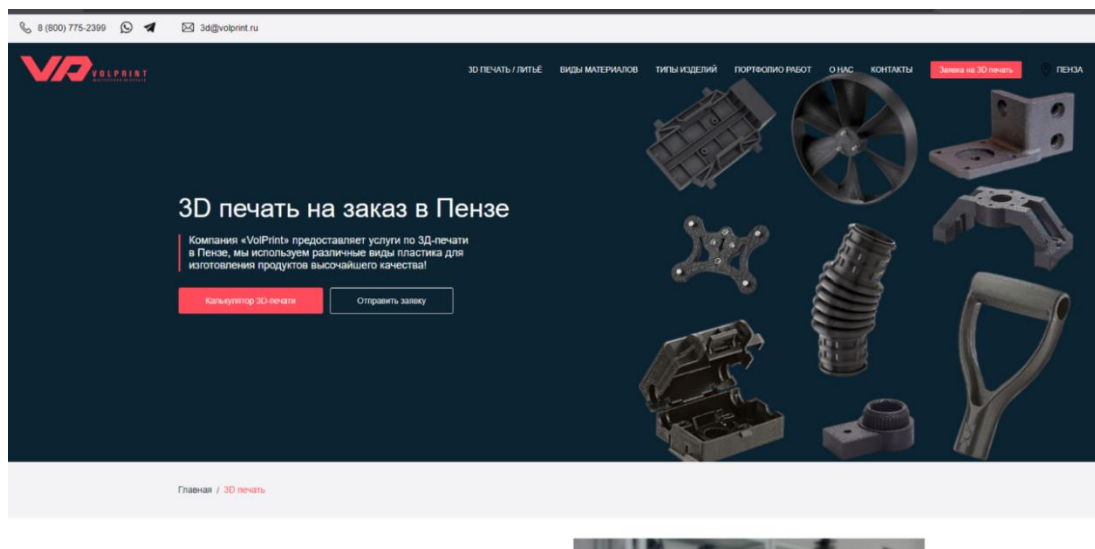


Рисунок 3 – Главная страница Volprint

Преимущества:

- различный выбор средств печати изделий;
- расчет индивидуальных заказов;
- наличие таблицы отзывов о заказах.

Недостатки:

- отсутствие личного кабинета пользователя;
- отсутствие регистрации пользователя;
- отсутствие аутентификации и авторизации.

Данный сервис обладает достаточно богатым выбором реализации печати изделий. Но данный портал обладает недостатками, описанными выше, что делает его не дружественным к пользователю.

Далее рассмотрим веб-приложение asteri-3d.ru [7]. На рисунке 4 представлена главная страница приложения.

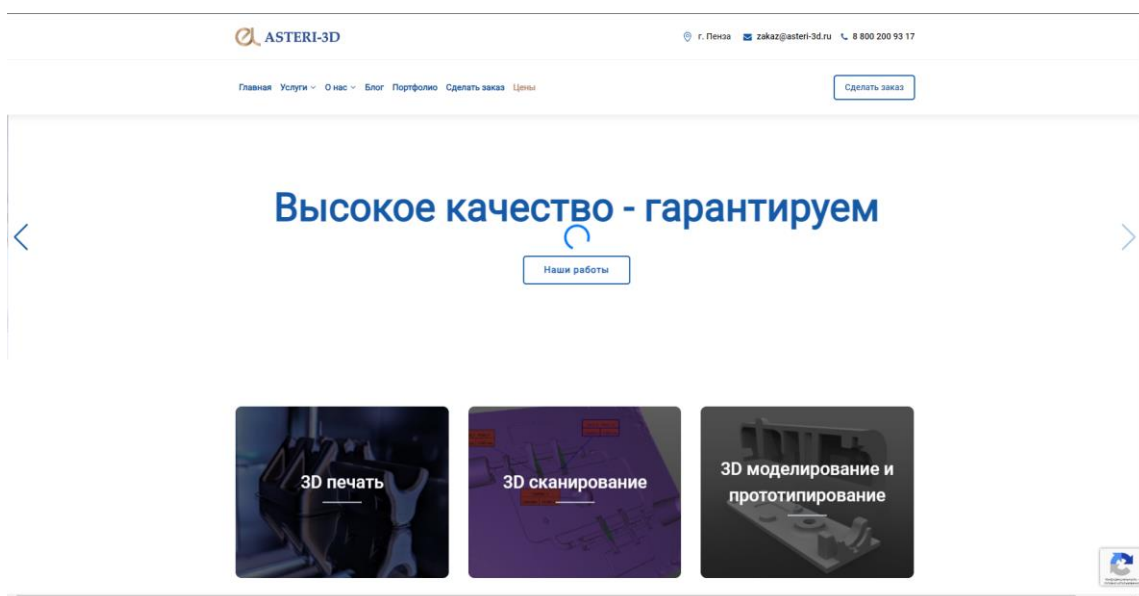


Рисунок 4 – Главная страница asteri-3d

Преимуществом данного приложения является удобство при оформлении индивидуального заказа.

Недостатки:

- отсутствие личного кабинета пользователя;
- отсутствие регистрации пользователя;
- отсутствие аутентификации и авторизации.

Проведем сравнение приложений по трехбалльной шкале оценивания, где 0 означает «полное отсутствие реализации», 1 – «частичную реализацию», а 2 – «полную реализацию» рассматриваемой функции. Результаты сравнения представлены в таблице 1.

Таблица 1 – Сравнение программных средств электронной коммерции

Критерий \ Приложение	Asteri-3D	VolPrint	3DMF	3dtoday
Наличие регистрации	0	0	0	1
Разграничение доступа к ресурсам	0	0	0	2
Наличие авторизации и аутентификации	0	0	0	1
Предоставление личных данных пользователя	0	0	0	1
Использование протокола OAuth2.1	0	0	0	0
ИТОГО	0	0	0	5

Подводя итог сравнения, можно сделать вывод, что ни одно из рассмотренных приложений не соответствует объявленным критериям в полной мере. Среди всех продуктов-аналогов большее количество баллов набрало веб-приложение «3dtoday», однако у него отсутствует протокол OAuth2.1 при авторизации и аутентификации пользователей, следовательно приводит к ослаблению безопасности веб-приложения.

1.3 Постановка задачи

В результате анализа предметной области было принято решение разработать собственный веб-сервис для реализации 3d-изделий.

Сервис должен иметь функционал оформления стандартного и индивидуального заказов, регистрации, аутентификации и авторизации пользователя. Менеджер должен иметь функционал обработки заказов. Сотрудник должен иметь функционал завершения заказов. Также сервис должен иметь реализацию личного кабинета пользователя.

В личном кабинете должны отображаться:

- профиль пользователя;
- история заказов пользователя.

Разрабатываемая система должна быть удобна в использовании и предоставлять программный интерфейс для взаимодействия с основным функционалом системы.

1.4 Анализ требований

Анализ требований является важным этапом проектирования, на котором в соответствии с бизнес-процессами можно сформулировать функциональные требования к системе [8].

Проектируемый сервис должен соответствовать протоколу OAuth2.1, а следовательно, должен иметь отдельный сервер для авторизации, аутентификации и регистрации пользователей. Для получения данных о пользователе и для организации доступа клиентских приложений к ресурсам или

данным учетных записей пользователя должно использоваться расширение протокола OAuth – OpenID Connect 1.0. Все получаемые токены должны храниться на достоверном ресурсе. Для защиты сервисов приложения он несанкционированного доступа должен быть реализован обратный прокси сервер.

1.4.1 Функциональные требования

Согласно поставленным задачам, были выделены следующие функциональные требования:

- регистрация и авторизация;
- оформление типового заказа;
- оформление индивидуального заказа;
- отправка активных заказов мастеру по печати;
- получение активных заказов сотрудником;
- получение менеджером всех активных заказов;
- принятие заказа менеджером;
- отклонение заказа менеджером;
- завершение заказа сотрудником;
- просмотр личного профиля.

В соответствии с выявленными требованиями была разработана диаграмма вариантов использования UML.

Диаграмма вариантов использования иллюстрирует различные варианты использования и различных типов пользователей, которые существуют системе, а также помогают фиксировать системные требования и отображать поведение системы. Варианты использования описывают функциональность системы с точки зрения пользователей, а пользователи представляют внешние сущности, которые используют систему. Диаграмма вариантов использования представлена на рисунке 5.

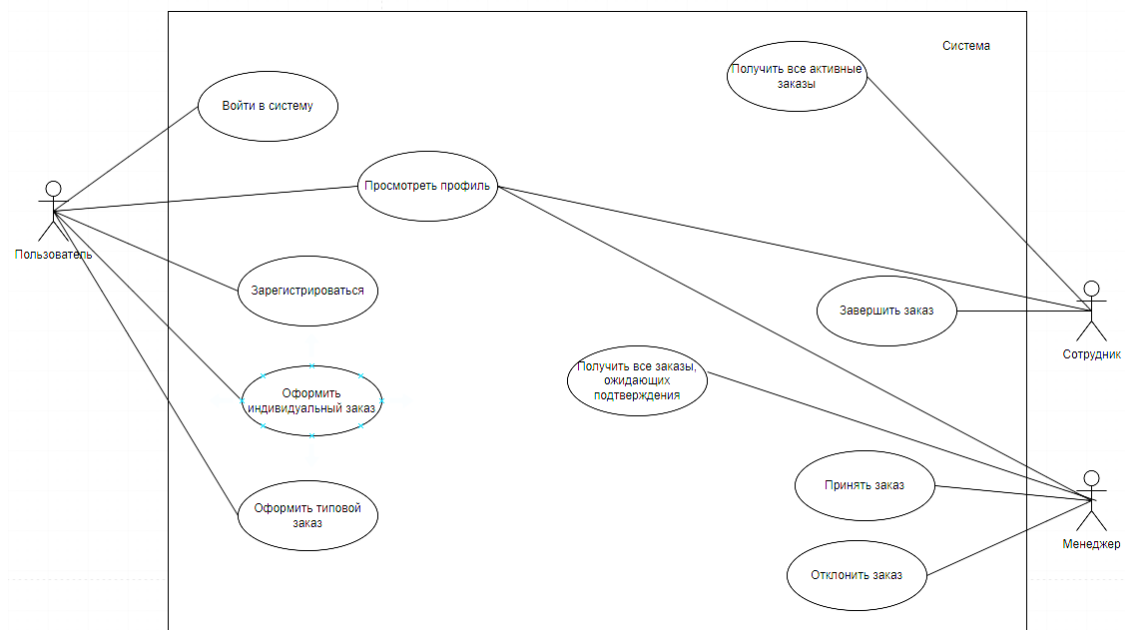


Рисунок 5 – Диаграмма вариантов использования

Спецификация прецедента «Войти в систему» представлена на рисунке 6.

Наименование: Войти в систему
ID: 1
Краткое описание: пользователь проходит авторизацию и аутентификацию
Предусловие: пользователь зарегистрирован
Основной поток: <ol style="list-style-type: none"> 1. Пользователь вводит данные в форму 2. Система проверяет валидность данных 3. Система перенаправляет пользователя на главную страницу приложения
Постусловие: пользователь вошел в систему

Рисунок 6 – Спецификация прецедента «Войти в систему»

На рисунке 7 представлена спецификация прецедента «Зарегистрироваться».

Наименование: Зарегистрироваться
ID: 2
Краткое описание: пользователь проходит регистрацию
Предусловие: нет
Основной поток: <ol style="list-style-type: none"> 1. Пользователь вводит данные 2. Система проверяет существует ли такой пользователь в базе данных 3. Система перенаправляет пользователя на страницу авторизации
Постусловие: пользователь зарегистрирован

Рисунок 7 – Спецификация прецедента «Зарегистрироваться»

На рисунке 8 представлена спецификация прецедента «Получить все активные заказы».

Наименование: Получить все активные заказы
ID: 3
Краткое описание: сотрудник получает все активные заказы
Предусловие: пользователь зашел в систему как сотрудник
Основной поток: <ol style="list-style-type: none"> 1. Пользователь выбирает соответствующий пункт меню 2. Система отображает все активные заказы
Постусловие: Получены все активные заказы

Рисунок 8 – Спецификация прецедента «Посмотреть профиль»

На рисунке 9 представлена спецификация прецедента «Посмотреть историю заказов».

Наименование: Посмотреть историю заказов
ID: 4
Краткое описание: пользователь просматривает историю заказов
Предусловие: пользователь зашел в систему
Основной поток: <ol style="list-style-type: none"> 1. Пользователь выбирает соответствующий пункт меню 2. Система отображает историю заказов пользователя
Постусловие: Отображена история заказов

Рисунок 9 – Спецификация прецедента «Посмотреть историю заказов»

На рисунке 10 представлена спецификация прецедента «Принять заказ».

Наименование: Принять заказ
ID: 5
Краткое описание: менеджер принимает заказ
Предусловие: пользователь зашел в систему как менеджер
Основной поток: <ol style="list-style-type: none"> 1. Менеджер выбирает соответствующий пункт меню 2. Система отображает активные заказы 3. Менеджер принимает заказ
Постусловие: Менеджер принял заказ

Рисунок 10 – Спецификация прецедента «Принять заказ»

На рисунке 11 представлена спецификация прецедента «Принять заказ»

Наименование: Отклонить заказ
ID: 6
Краткое описание: менеджер отклоняет заказ
Предусловие: пользователь зашел в систему как менеджер
Основной поток: <ol style="list-style-type: none"> 1. Менеджер выбирает соответствующий пункт меню 2. Система отображает активные заказы 3. Менеджер отклоняет заказ
Постусловие: Менеджер отклонил заказ

Рисунок 11 – Спецификация прецедента «Отклонить заказ»

На рисунке 12 представлена спецификация прецедента «Завершение заказа».

Наименование: Завершение заказа
ID: 7
Краткое описание: мастер сообщает о завершении заказа
Предусловие: пользователь зашел в систему как мастер
Основной поток: <ol style="list-style-type: none"> 4. Система отображает принятые заказы от менеджера 5. Мастер выбирает выполненный заказ 6. Мастер завершает заказ
Постусловие: Мастер завершил заказ

Рисунок 12 – Спецификация прецедента «Сообщить о завершении заказа»

На рисунке 13 представлена спецификация прецедента «Получить все заказы, ожидающих подтверждения».

Наименование: Получить все заказы, ожидающих подтверждения
ID: 8
Краткое описание: Менеджер получает все заказы, ожидающие подтверждения
Предусловие: пользователь зашел в систему как менеджер
Основной поток: <ol style="list-style-type: none"> 1. Пользователь выбирает определённый пункт меню 2. Система отображает все заказы, ожидающие подтверждения
Постусловие: Менеджер получил все заказы, ожидающие подтверждения

Рисунок 13 – Спецификация прецедента «Получить все заказы, ожидающих подтверждения»

В результате анализа функциональных требований были выделены варианты использования программы. Для основных вариантов использования были представлены их спецификации, которые позволяют отобразить последовательность действий пользователя и системы при выполнении определенных функций.

1.4.2 Нефункциональные требования

Серверная часть приложения должна быть реализована с помощью микросервисной архитектуры на языке Java с использованием фреймворка Spring [9]. Язык Java был выбран так, как он обладает кроссплатформенностью и множеством библиотек, способствующих разработке серверных приложений, а фреймворк Spring обладает множеством инструментов, позволяющие реализовывать веб-приложения, соответствующие современным стандартам в области разработки ПО. Необходимо реализовать RESTful веб-сервисы, позволяющие использовать реализованные сервисы на любом клиентском устройстве [10]. В качестве СУБД будет использоваться PostgreSQL.

Данная база данных была выбрана, так как обладает достаточной надежностью при хранении данных и высокой производительностью при чтении и модификации данных.

Для работы с БД необходимо использовать ORM Hibernate. Данная ORM позволяет сократить объемы низкоуровневого программирования при работе с реляционными базами данных, таких как PostgreSQL, а также решает задачу связи классов Java с таблицами базы данных.

Клиентская часть приложения будет реализована на языке JavaScript с использованием фреймворка React. Приложение должно работать в веб-браузерах, реализованных на движках Webkit и Blink (Chrome, Safari, Firefox).

При разработке сервисов необходимо использовать интегрированную среду разработки IntelliJ idea [11], а при разработке клиентского приложения использовать среду разработки WebStorm. Данные среды разработки позволяют эффективно проектировать, разрабатывать и отлаживать программные средства.

2 Проектирование программных средств администрирования для приложения электронной коммерции

2.1 Архитектура приложения

2.1.1 Архитектура сервера авторизации

Архитектура сервиса авторизации основана на REST архитектуре и шаблоне проектирования MVC [12]. MVC архитектура является наиболее распространенной при проектировании веб-приложений, так как разделение на слои Model, View, Controller естественно ложится на структуру подобных сервисов. На рисунке 14 представлена схема взаимодействия приложений с REST API, построенного по MVC архитектуре.

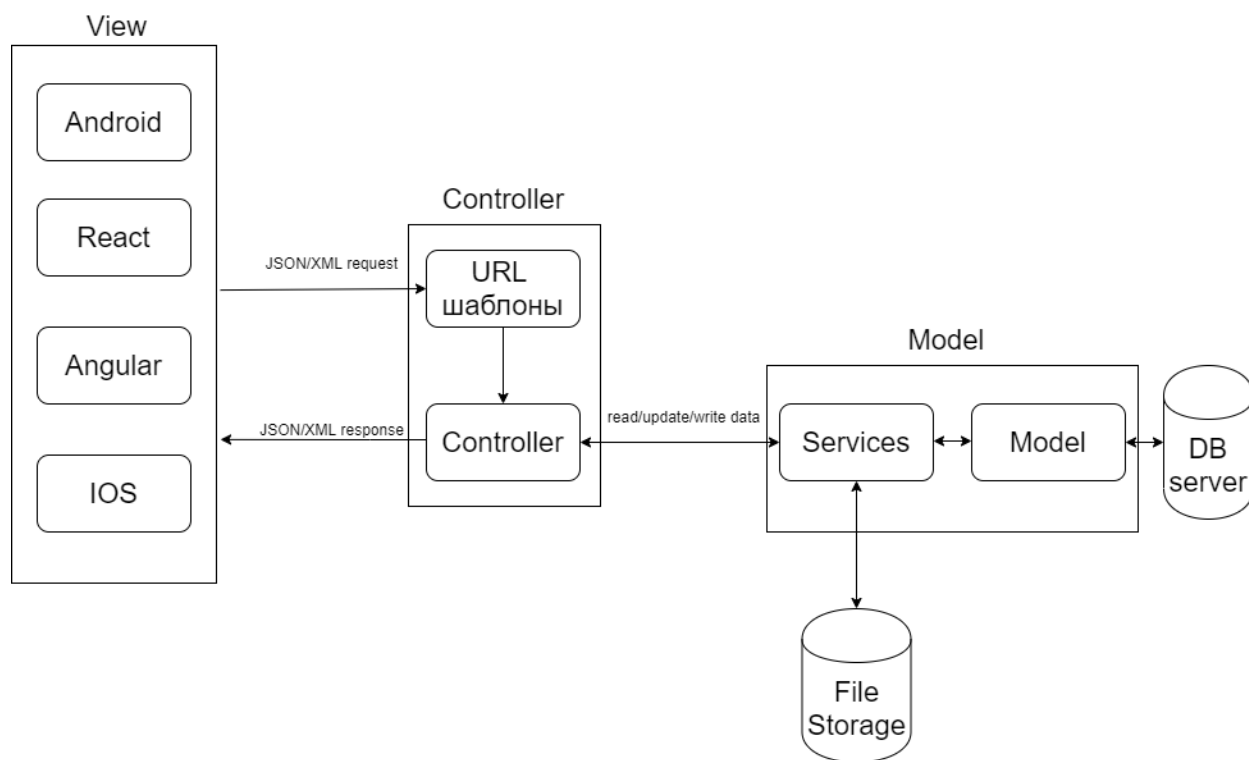


Рисунок 14 – Схема взаимодействий приложений с API в MVC архитектуре

Также из данной модели видно, что данный сервис основан на трехуровневой модели, состоящей из клиента, сервера приложений и сервера базы данных. Данная модель накладывает несколько основных ограничений на приложение:

- уровень клиента должна содержать минимум бизнес-логики приложения;
- уровень клиента не должен иметь прямой доступ к серверу БД;
- уровень сервера приложений должен обеспечивать возможность горизонтального масштабирования.

Клиентом в данной системе может являться любое клиентское устройство, например, Android, IOS или браузер.

2.1.2 Архитектура прокси сервера

API Gateway – базовый шаблон проектирования при разработке приложения с использованием микросервисервисной архитектуры [13]. Данный шаблон представлен на рисунке 15.

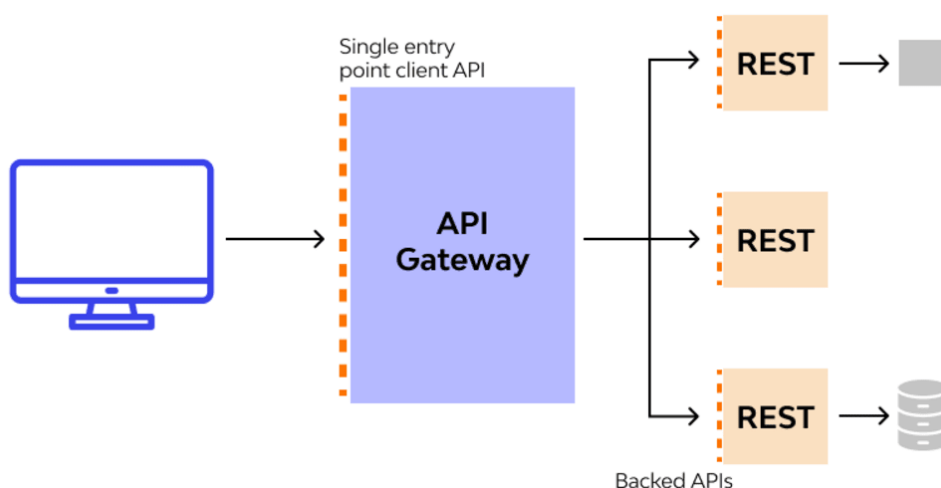


Рисунок 15 – Схема шаблона

Он представляет собой маршрутизатор API-запросов, который получает на вход все запросы клиента, обрабатывает их и рассылает разным сервисам, а после отправляет ответ обратно клиенту. Предоставляя центральную точку входа, управляя маршрутизацией, трансляцией протоколов, безопасностью, балансировкой нагрузки и другими сквозными задачами, API Gateway упрощает взаимодействие архитектуры клиент-сервер и повышает масштабируемость, безопасность и управляемость распределенных систем [14].

2.1.3 Архитектура взаимодействия

Клиент-серверная архитектура – это система распределения, в которой одни устройства обращаются к другим для получения необходимой информации или выполнения определенных задач [15]. Данная архитектура представлена на рисунке 16.

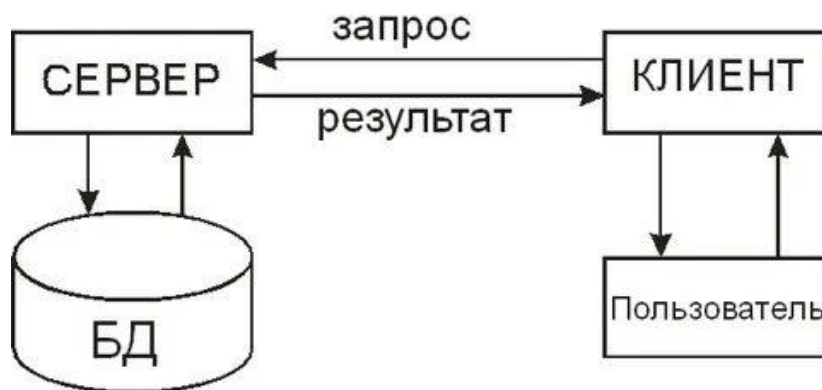


Рисунок 16 – Схема клиент – серверной архитектуры

Программы-серверы ожидают от клиентских программ запросы и предоставляют им свои ресурсы в виде данных или в виде сервисных функций.

2.1.4 Архитектура микросервисов

Микросервисная архитектура – архитектура программного обеспечения, направленная на взаимодействие насколько это возможно небольших, слабо связанных и легко изменяемых модулей, реализующие отдельные бизнес-процессы. Данный подход к разработке программного обеспечения позволяет разрабатывать, изменять и масштабировать отдельные части приложения независимо от других модулей [16].

2.2 Проектирование структур данных

2.2.1 Логическая модель базы данных

На основе проведенного анализа предметной области и функциональных требований построена логическая модель базы данных с использованием

нотации «сущность-связь». Логическая модель отражает суть выбранной предметной области и выделяет основные сущности, атрибуты и связи в системе.

В начале необходимо выделить список всех основных сущностей данной предметной области. Данный список представлен в таблице 2.

Таблица 2 – Описание сущностей предметной области

№	Наименование	Описание	Категория	Количество экземпляров	Режим обновления
1	Пользователь	Списочный состав пользователей, зарегистрированных в системе	Стержневая	1000	5% в неделю
2	Заказанные товары	Список заказанных товаров	Стержневая	100000	Очень часто
3	Предмет заказа	Список типов товаров и их количество	Стержневая	100000	5% в день
4	Заказ	Списочная информация заказов	Стержневая	100000	5% в день
5	Предмет каталога	Список товаров	Стержневая	50	Очень редко

На основании данных сущностей необходимо построить модель. Для создания модели использовался язык IDEF1X с использованием CASE-средства draw.io [17]. На рисунке 17 изображена построенная логическая модель сервиса.



Рисунок 17 – Логическая схема БД

Далее перейдем к проектированию физической модели базы данных.

2.2.2 Физическая модель базы данных

Физическая модель данных включает в себя все реализуемые сущности, технические спецификации таблиц, требования к хранению и типы индексов. Производительность базы данных, стратегия индексации, физическое хранилище и денормализация – важные параметры физической модели [18].

В качестве СУБД будет использована PostgreSQL версии 16. Плюсами данной СУБД является то, что она открыто распространяемая и имеет реализацию для множества платформ. Также данная СУБД имеет поддержку множества различных типов данных, такие как JSON, XML, массивы, uuid идентификаторы и т. д.

На основании логической модели была разработана физическая модель базы данных. Физическая модель БД в нотации IDEF1X представлена на рисунке 18.

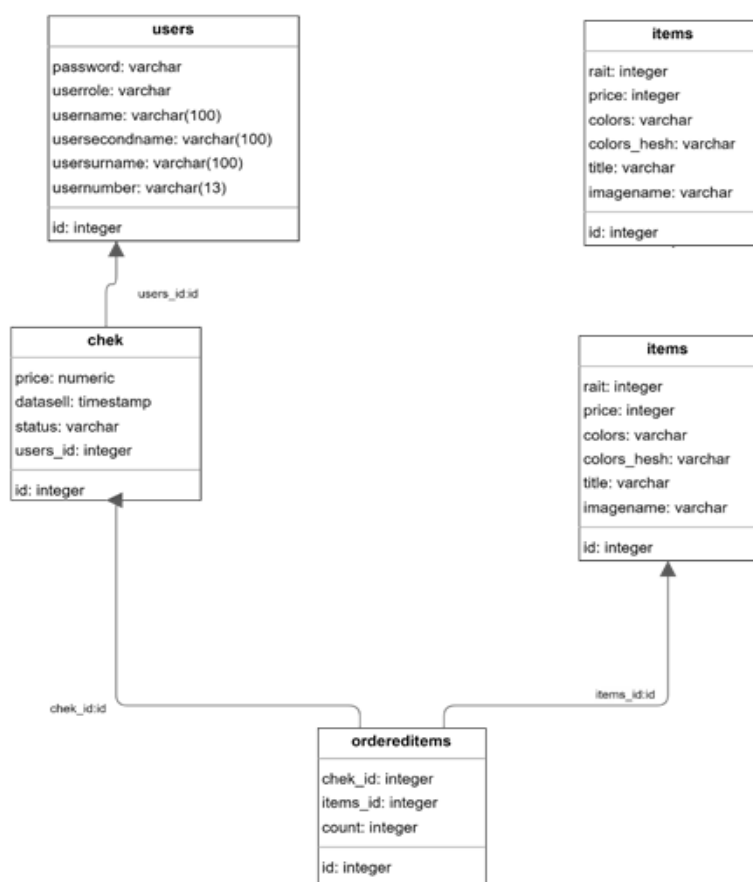


Рисунок 18 – Физическая схема БД

В процессе создания физической модели было принято решение использовать для создания модели БД миграции, а для описания БД ORM Hibernate. ORM основана на объектном подходе – представлении таблиц БД в виде классов. Поэтому построенную физическая модель необходимо описать в терминах ORM.

2.3 Проектирование программных средств

2.3.1 Проектирование алгоритма доступа к защищенным данным

Диаграмма последовательности описывает коммуникационный аспект между основными объектами системы в некотором сценарии. На данной диаграмме описан процесс получения защищенных данных (рисунок 19).

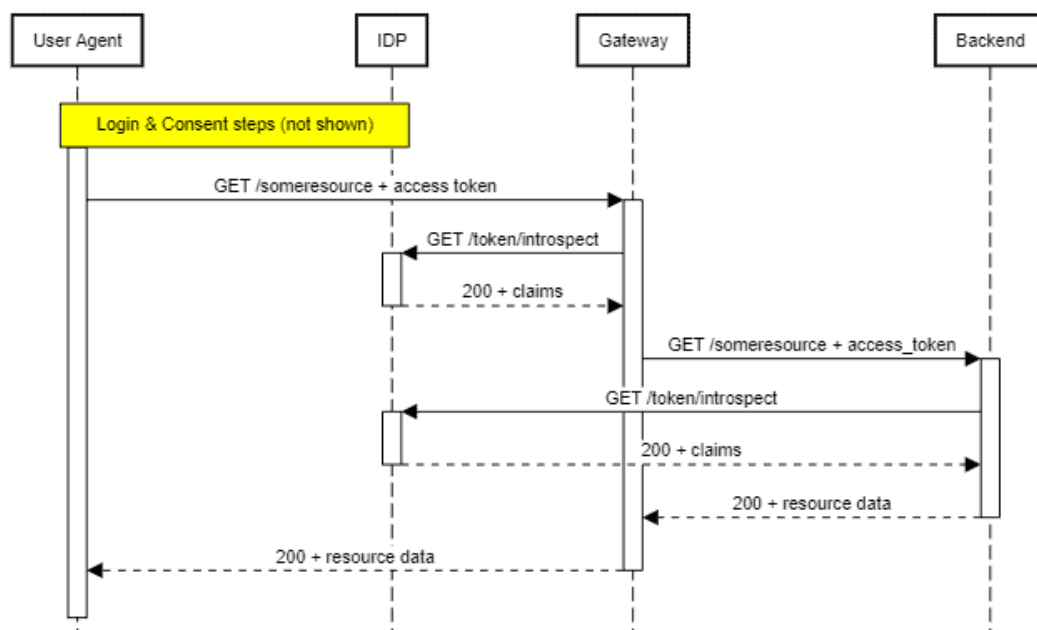


Рисунок 19 – Диаграмма последовательности доступа к защищенным данным

Эта диаграмма включает четыре основных объекта: пользователь, прокси сервер, сервер авторизации и сервер ресурсов. Первым действием пользователь входит в систему, после чего клиентское приложение формирует запрос о получении данных с токеном доступа. На стороне прокси сервера происходит определение валидности и прав доступа путем отправки токена доступа на проверку серверу авторизации. После проверки токена сервер перенаправляет запрос с ответом обратно на прокси сервер. Если валидность и права доступа

подтвердились, прокси сервер перенаправляет запрос на сервер ресурсов. В свою очередь, сервер ресурсов отправляет токен доступа на расшифровку и получение данных о пользователе. После получения ответа, сервер ресурсов формирует ответ с запрошенными данными и отправляет прокси серверу. После получения ответа прокси сервер перенаправляет этот ответ пользователю.

2.3.2 Проектирование классов прокси сервера

Согласно выбранной архитектуре прокси сервера было спроектировано три основных класса приложения. Класс `GatewayController` реализует перенаправление на клиентское приложение. В классе `SecConfig` содержится бизнес-логика проверки валидности и прав доступа пользователя. На рисунке 20 представлена диаграмма классов прокси сервера.

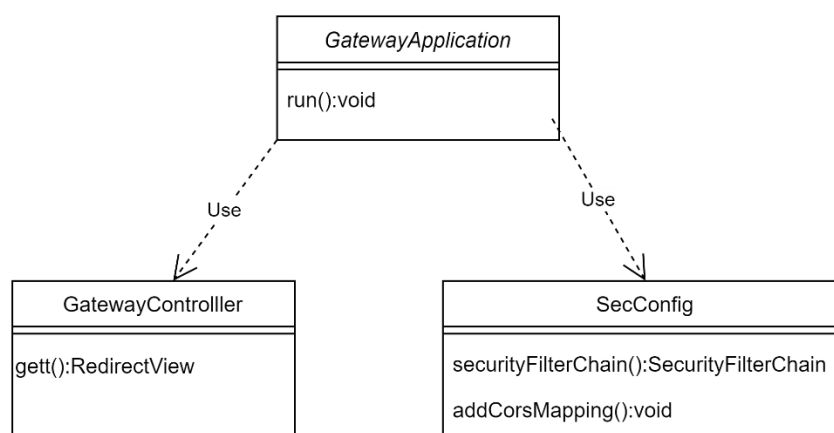


Рисунок 20 – Диаграмма классов для прокси сервера

В качестве примера, для классов прокси сервера была составлена спецификация. Спецификация класса `SecConfig` описана в таблице 3.

Таблица 3 – Спецификация класса «`SecConfig`»

Наименование класса: <code>SecConfig</code>
Назначение класса: содержит в себе бизнес логику проверку валидности и прав доступа пользователя
Методы класса: <code>securityFilterChain</code> – фильтрация запросов <code>addCorsMapping</code> – механизм, обеспечивающий контролируемый доступ к ресурсам, расположенным за пределами данного домена.

Спецификация класса `GatewayController` описана в таблице 4.

Таблица 4 – Спецификация класса «GatewayController»

Наименование класса: GatewayController
Назначение класса: реализует перенаправление на клиентское приложение
Методы класса: gett() – перенаправление на клиентское приложение

Данный сервис был спроектирован согласно алгоритму доступа, описанного выше.

2.3.3 Проектирование классов сервера авторизации

Согласно шаблону проектирования MVC и выбранному архитектурному подходу REST было спроектировано несколько основных модулей приложения. Диаграмма классов для сервера авторизации представлена в приложении Г.1.

В классе SecurityConfig содержится бизнес-логика процессов авторизации и регистрации пользователя. В классе AuthServerConfig содержится бизнес-логика сервера авторизации. Классы UserControllerDTO и UserDTO отвечают за слой представления данных о пользователе. Класс UserEntity реализует сущность пользователя в БД. Класс UserDetailsServiceImpl занимается формированием записей, которые будут сохранены или получены в БД. Интерфейс UserRepository реализует запросы к БД с использованием ORM.

В качестве примера, для классов сервера авторизации была составлена спецификация. В таблице 5 представлена спецификация класса AuthorizationService, ответственного за бизнес-логику процесса авторизации.

Таблица 5 – Спецификация класса «AuthorizationService»

Наименование класса: AuthServerConfig
Назначение класса: содержит в себе бизнес логику сервера авторизации
Поля класса: AuthorizationServerProperties: AuthorizationServerProperties – зависимости для сервера авторизации
Методы класса: http – настройка цепочки http запросов registeredClientRepository – настройка клиента OAuth2.1 jwtSource – настройка шифрования токена tokenCustomizer – настройка токена доступа

В таблице 6 представлена спецификация класса UserDetailsServiceImpl, содержащий в себе в методы конечных точек доступа к ресурсу.

Таблица 6 – Спецификация класса «UserDetailServiceImpl»

Наименование класса: UserDetailServiceImpl
Назначение класса: содержит в методы доступа к пользователю
Поля класса: userRepository: userRepository – получение данных о пользователе
Методы класса: saveUser – сохранение пользователя в БД loadUserByUsername – получение пользователя по имени

Спецификация класса RegistrationService описана в таблице 7.

Таблица 7 – Спецификация класса «RegistrationService»

Наименование класса: SecurityConfig
Назначение класса: содержит в себе бизнес логику регистрации пользователя
Поля класса: defaultSecurityFilterChain: void – цепочка фильтров аутентификации пользователя

Данный сервис был спроектирован согласно алгоритму доступа, приведенному выше.

2.4 Проектирование пользовательского интерфейса веб-приложения

На основе диаграммы вариантов использования, для клиентского приложения был спроектирован графический интерфейс. Графический интерфейс пользователя – это способ взаимодействия пользователя с компьютером с использованием графических элементов, таких как окна, кнопки и меню. В графическом пользовательском интерфейсе визуальные элементы представляют действия, которые могут выполнять пользователи, объекты, которыми пользователи могут манипулировать, и другую информацию, относящуюся к пользователю.

Пользовательский интерфейс должен обеспечивать простое и эффективное выполнение основных функций пользователей.

Интерфейс пользователя будет состоять из нескольких окон:

- окно профиля менеджера;
- окно профиля сотрудника;
- окно входа;
- окно регистрации.

На рисунке 21 представлен интерфейс окна входа.

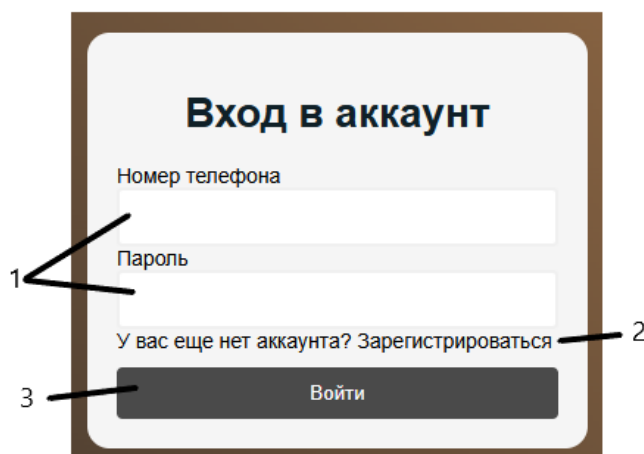


Рисунок 21 – Окно входа

Интерфейс окна входа состоит из:

- 1 – поля для ввода пользовательских данных;
- 2 – кнопка отправки формы;
- 3 – кнопка перехода на страницу регистрации.

При нажатии кнопки отправки формы, данные отсылаются на проверку серверной части приложения.

На рисунке 22 представлен интерфейс окна профиля сотрудника.



Рисунок 22 – Окно профиля сотрудника

Основные элементы интерфейса данного окна, представленного на рисунке 22:

- 1 – навигационная панель для перехода к главной странице;
- 2 – навигационная панель для перехода к аккаунту пользователя;
- 3 – навигационная панель для перехода к каталогу;
- 4 – профиль сотрудника;
- 5 – кнопка завершения заказа.

При нажатии на кнопку завершения заказа данные передадутся в базу данных и поменяется статус заказа.

На рисунке 23 представлен интерфейс окна регистрации.

Рисунок 23 – Окно регистрации

Интерфейс окна регистрации состоит из:

- 1 – поля для ввода пользовательских данных;
- 2 – кнопка перехода на страницу логина;
- 3 – кнопки отправки формы.

При нажатии кнопки отправки формы, данные отсылаются на проверку серверной части приложения.

На рисунке 24 представлен интерфейс окна менеджера.



Рисунок 24 – Окно профиля менеджера

Интерфейс окно профиля менеджера состоит из:

- 1 – навигационная панель для перехода к главной странице;
- 2 – навигационная панель для перехода к аккаунту пользователя;
- 3 – навигационная панель для перехода к каталогу;
- 4 – профиль сотрудника;
- 5 – кнопка подтверждения заказа;
- 6 – кнопка отклонения заказа;
- 7 – список заказов.

При нажатии на кнопку принятия заказа данные передадутся в базу данных и поменяется статус заказа.

Каждая страница и панель содержит в себе множество React компонентов. Клиентское приложение связывается с сервером приложений по средствам библиотеки axios, предоставляющий удобный механизм для составления http запросов.

2.5 Проектирование клиентского приложения

Для написания клиентского приложения был выбран фреймворк React.js. React приложение использует компонентный подход. Каждый компонент представлен в виде отдельной функции. В React.js есть возможность создавать

инкапсулированные компоненты, которые управляют своим состоянием, а затем компоновать их для создания сложных пользовательских интерфейсов [19].

React-компоненты реализуют метод `render()`, который принимает входные данные и возвращает некоторые данные для вывода. Данный метод определяет, что должно быть отображено на экране в ответ на изменение состояния или свойств компонента. Он вызывается автоматически, когда компонент монтируется или обновляется. Для описания используется XML-подобный синтаксис под названием TSX. Входные данные, передаваемые в компонент, доступны в `render()` через переменную `this.props`. Помимо входных данных, компонент поддерживает внутренние данные состояния которые могут быть доступны с помощью переменной `this.state`. Объект `state` описывает состояние компонента. Когда данные состояния компонента изменяются, React заново вызывает метод `render()` и обновляет разметку страницы.

3 Реализация программных средств администрирования для приложения электронной коммерции

3.1 Реализация сервера авторизации

Сервер авторизации реализован на языке Java, с помощью фреймворка для создания веб-приложений Spring. Компоненты данного сервиса представлены на рисунке 25.

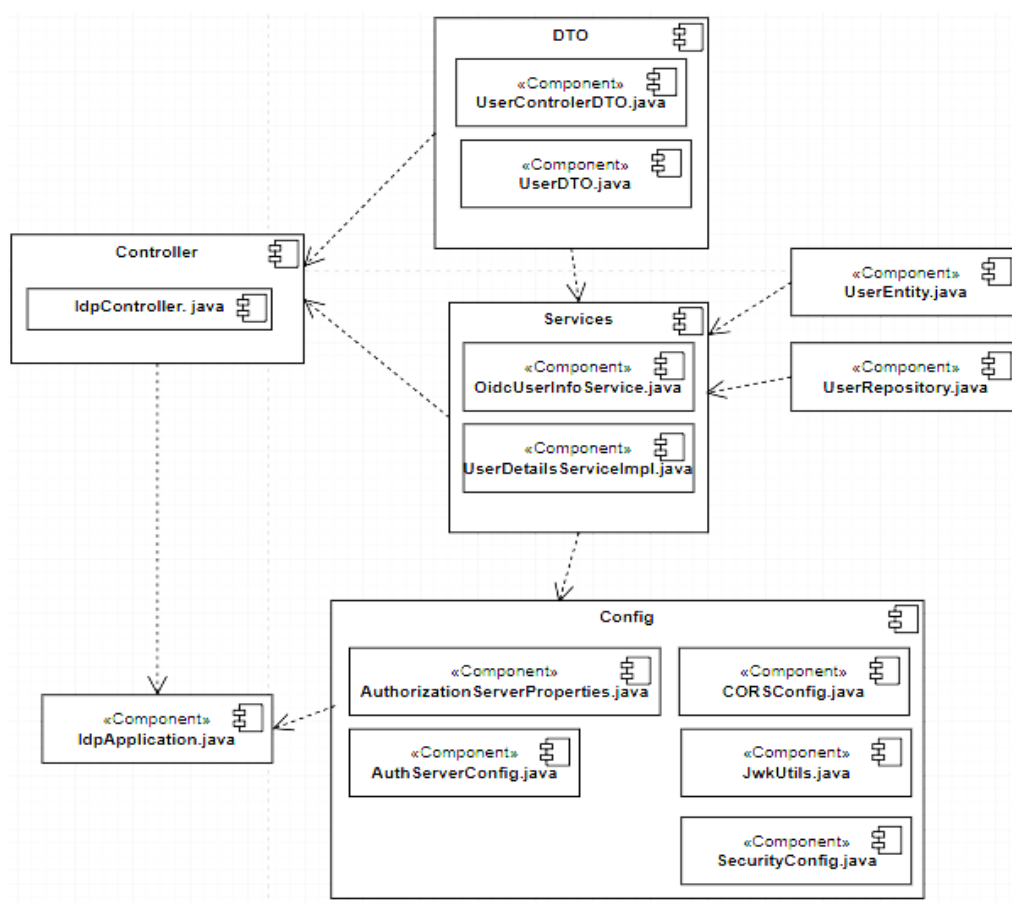


Рисунок 25 – Диаграмма компонентов сервера авторизации

В таблице 8 приведены описания основных компонентов системы.

Таблица 8 – Описание компонентов системы

№	Название	Описание
1	idpApplication.java	Является точкой входа, содержит все сервисы приложения.
2	UserRepository.java	Репозиторий для получения данных пользователя из базы данных
3	UserEntity.java	Описание таблицы пользователя БД с помощью языка ORM

Продолжение таблицы 8

№	Название	Описание
4	UserDTO.java	Объект представления пользователя.
5	UserControllerDTO.java	Объект представления пользователя.
6	AuthorizationServerProperties.java	Класс зависимостей для сервера авторизации.
7	AuthServerConfig.java	Описание таблиц БД с помощью языка ORM.
8	CORSConfig.java	Класс конфигурации политики cors.
9	JwkUtils.java	Класс конфигурации шифрования токена доступа.
10	SecurityConfig.java	Класс конфигурации цепочки фильтров входа в систему.
11	OidcUserInfoService.java	Сервис создания пользователя OIDC Connect.
12	UserDetailsServiceImpl.java	Бизнес логика сервиса, связанного с пользователем.
13	IdpController.java	Класс, содержащий конечные точки обращения к сервису

Данный сервис был реализован с учетом критериев, установленных ранее.

3.2 Реализация прокси сервера

Прокси сервер реализован на языке Java, с помощью фреймворка для создания веб-приложений Spring. Компоненты данного сервиса представлены на рисунке 26.

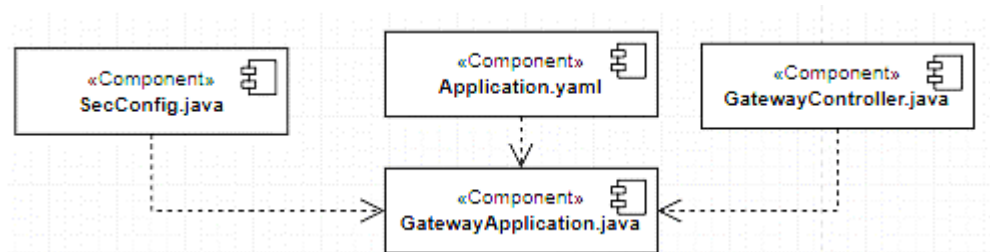


Рисунок 26 – Диаграмма компонентов прокси сервера

В таблице 9 приведены описания основных компонентов системы.

Таблица 9 – Описание компонентов системы

z	Название	Описание
1	GatewayController.java	Класс, содержащий конечные точки обращения к сервису
2	SecConfig.java	Класс конфигурации цепочки запросов
3	GatewayApplication.java	Является точкой входа, содержит все сервисы приложения
4	Applicational.yaml	Конфигурационный файл прокси сервера

Данный сервис был реализован с учетом критериев, установленных ранее.

3.3 Развертывание базы данных

БД была развернута с помощью механизма миграции. Миграция – обновление структуры базы данных до определённой версии, чаще всего до более новой.

Перед первичной миграцией необходимо создать БД, к которой подключится сервер авторизации. Для данной цели был написан скрипт на языке SQL. Механизм миграций необходим для версионирования БД и поддержания ее консистентности. В данной работе использовалась библиотека Flyway [20] для миграции БД.

Результат развертывания БД представлен на рисунке 27.

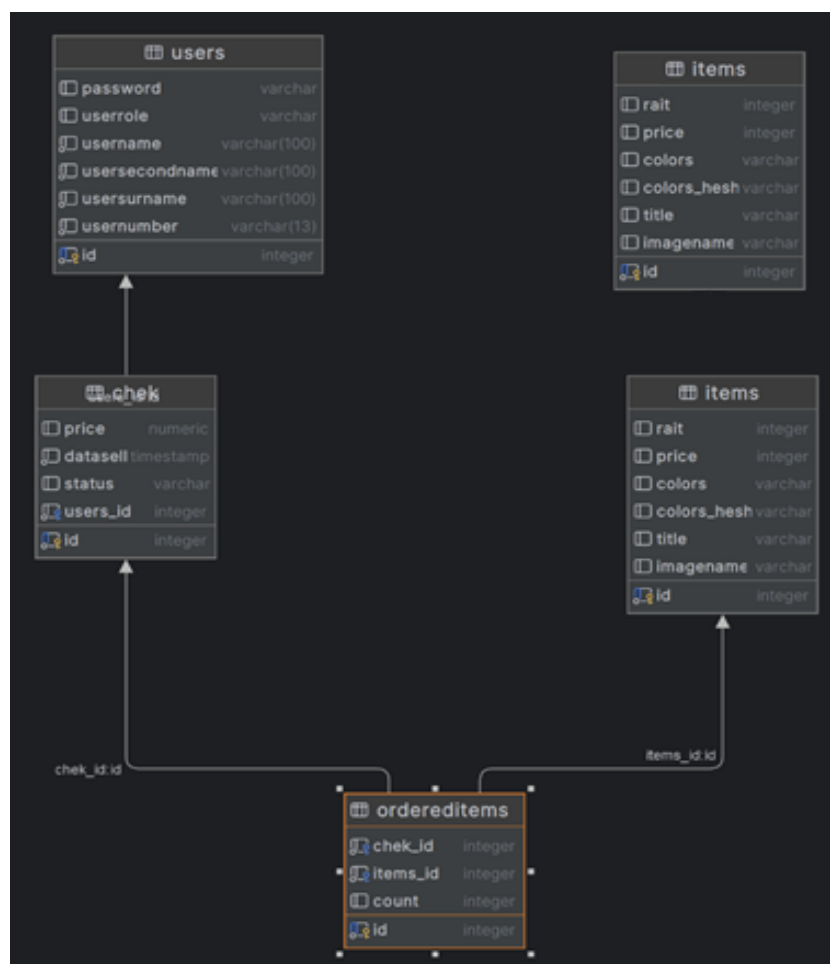


Рисунок 27 – Результат развертывания БД

База данных была развернута с учетом требований, указанных ранее.

3.4 Спецификация конечных точек

Так как сервисы представляют собой REST API приложения, то необходимо представить спецификацию для каждого сервиса. К каждому сервису можно обратиться по специализированному URI, а основным средством обмена информацией является JSON объекты. Спецификация сервисов представлена в таблице 10.

Таблица 10 – Спецификация REST сервисов

№	Назначение сервиса	URI сервиса	Класс контроллера
1	Точка регистрации	http://localhost:9000/register	IdpController
2	Точка входа в систему	http://localhost:9000/login	IdpController
3	Обновление токена доступа	http://localhost:9000/oauth2/token	IdpController
4	Получение информации о пользователе	http://localhost:9000/userinfo	IdpController
5	Получение токена доступа	http://localhost:9000/oauth2/token	IdpController
6	Переход на главную страницу приложения	http://localhost:8080/	GatewayController
7	Получение профиля менеджера и всех заказов	http://localhost:8080/manager/getallorders	GatewayController
8	Получение профиля сотрудника и всех активных заказов	http://localhost:8080/worker/getallorders	GatewayController

Данные конечные точки разработаны с учетом требований, указанных ранее.

3.5 Развертывание программных средств

На рисунке 28 представлена диаграмма развертывания программных средств, которая изображает основные компоненты системы.

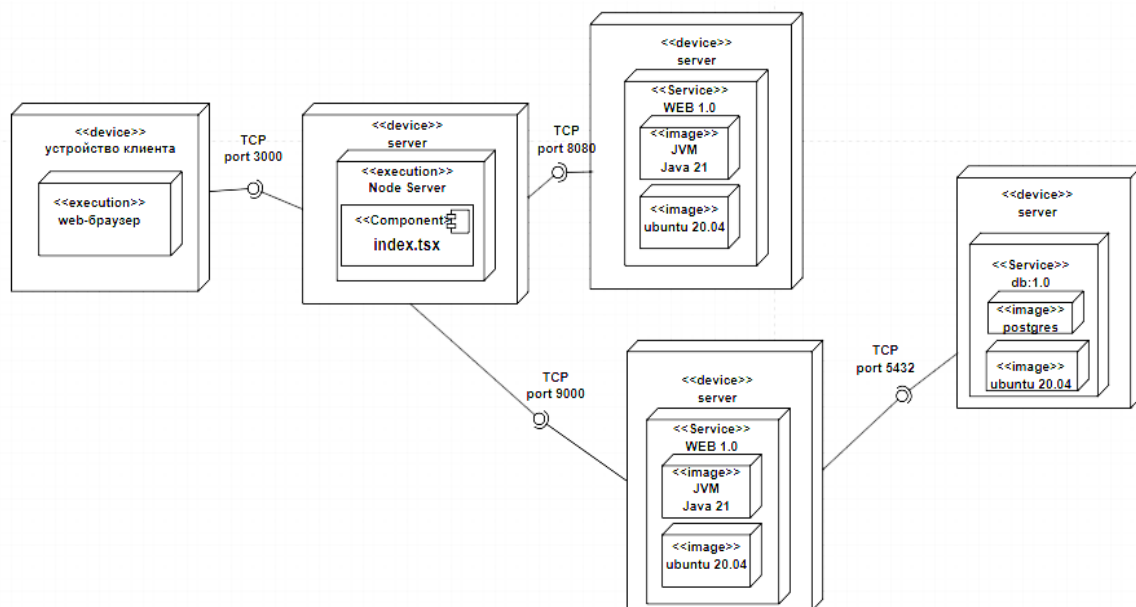


Рисунок 28 – Диаграмма развертывания

На диаграмме видно, что программные средства соответствуют трехуровневой архитектуре. Сервера программных средств и сервер БД могут быть обернуты в docker контейнеры, что позволяет в дальнейшем беспрепятственно переносить данные сервисы на другие платформы. Клиентское приложение должно работать на отдельном сервере.

4 Контроль качества программного обеспечения

4.1 Функциональное тестирование

Функциональное тестирование методом черного ящика – это тестирование функциональных возможностей ПО, в условиях, когда внутренняя структура и реализация неизвестна.

Произведем функциональное тестирование авторизации и аутентификации. Пользователь должен иметь возможность произвести авторизацию и аутентификацию. Номер телефона должен содержать только 11 символов разрешенного формата. В пароле допускаются только символы латинского и русского алфавитов, а также арабские цифры. Размерность пароля не должна быть меньше 10 корректных символов. Пользователь должен быть зарегистрирован перед входом в систему, а следовательно номер должен быть найден в базе данных.

В таблице 11 представлены классы эквивалентности.

Таблица 11 – Классы эквивалентности

Входные условия	Правильные классы эквивалентности	Неправильные классы эквивалентности
Длина номера	11	<11 или >11
Длина пароля	от 10	<10
Номер телефона	Состоит только цифр	Содержит другие символы
Номер в базе данных	номер найден	номер не найден
пароль	Состоит из букв латинского или русского алфавита, цифр	Содержит другие символы

Определим граничные условия. Длина пароля не менее 10 символов. Следовательно, длина в 10 символов находится на границе допустимых значений. Это значение следует проверить. Также имеет смысл проверить ближайшие неверное значение 9.

Также следует проверить работу программы на корректных последовательностях символов в номере, так и последовательности, содержащие некорректные символы.

Для признака нахождения номера существует только два возможных варианта. Либо номер найден, либо не найден. Следует протестировать оба варианта.

Для того, чтобы построить эффективно-результативные тесты, можно воспользоваться методом функциональных диаграмм. При построении тестов с использованием данного метода в спецификации определяются причины и следствия. Причина – входное условие, а следствие – выход системы. Далее спецификация преобразуется в управляющий граф.

Для процесса загрузки набора данных была построена функциональная диаграмма, представленная на рисунке 29.

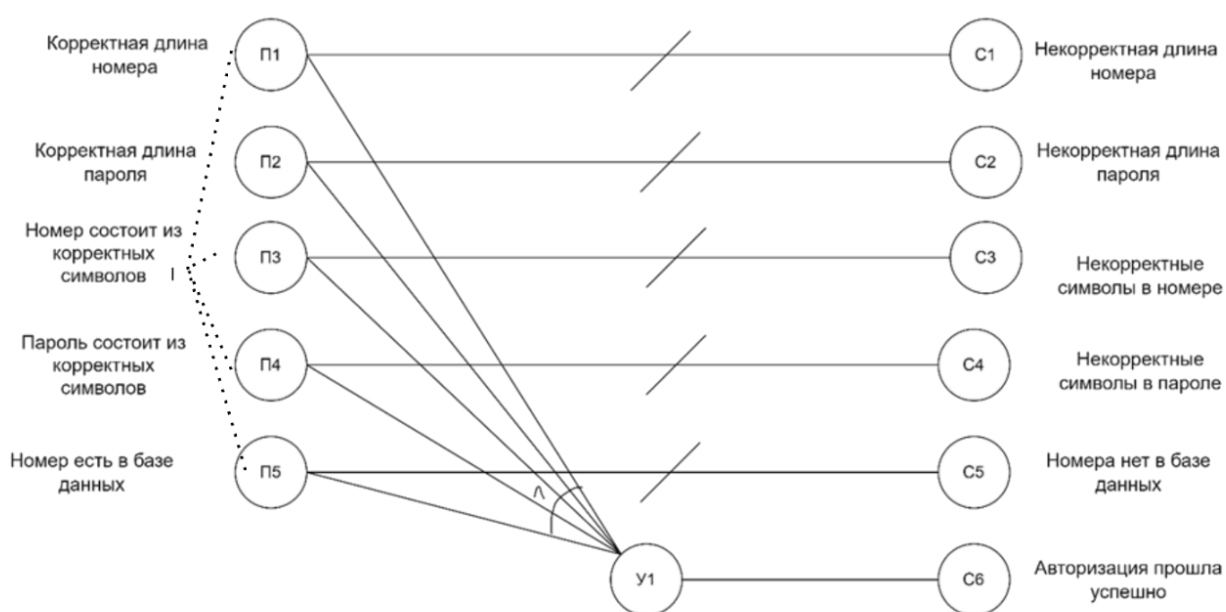


Рисунок 29 – Функциональная диаграмма

Причины:

- корректная длина номера;
- корректная длина пароля;
- номер состоит из корректных символов;
- пароль состоит из корректных символов;
- номер есть в базе данных.

Следствия:

- сообщение об ошибке «Некорректная длина номера»;

- сообщение об ошибке «Некорректная длина пароля»;
- сообщение об ошибке «Некорректные символы в номере»;
- сообщение об ошибке «Некорректные символы в пароле»;
- сообщение об ошибке «Номера нет в базе данных»;
- успех.

На основании функциональной диаграммы была построена таблица решений. Таблица решений представлена в таблице 12.

Таблица 12 – Таблица решений

	1	2	3	4	5	6	7	9	10
П1	0	1	1	0	1	0	1	-	1
П2	0	0	1	0	0	1	1	-	1
П3	0	0	0	1	1	1	1	-	1
П4	0	0	0	0	0	0	0	-	1
П5	0	0	0	0	0	0	0	0	1
С1	1	0	0	1	0	1	0	0	0
С2	1	1	0	1	1	0	0	0	0
С3	1	1	1	0	0	0	0	0	0
С4	1	1	1	1	1	1	1	0	0
С5	0	0	0	0	0	0	0	1	0
С6	0	0	0	0	0	0	0	0	1

В таблице 13 представлен набор тестов, составленные на основе таблицы решений.

Таблица 13 – Набор тестов

№	Входные данные	Ожидаемый результат	Результат
1	номер: Со! пароль: .re!.csv	(С1, С2, С3, С4)	Тест пройден (рисунок В.1)
2	номер: 89085220326 пароль: .re@.csv	(С2, С4)	Тест пройден (рисунок В.2)
3	номер: 908526759 пароль: recsv	(С1, С2, С4)	Тест пройден
4	номер: 89065330232 пароль: re2021.csv	(С4)	Тест пройден
5	номер: 890651330232 пароль: re2021csv	(С3)	Тест пройден (рисунок В.3)
6	номер: 8906533г0232 пароль: re2021.csv	(С3, С4)	Тест пройден

Продолжение таблицы 13

№	Входные данные	Ожидаемый результат	Результат
7	Псевдоним: 89064563220 пароль: re2021csv	(C5)	Тест пройден (рисунок В.4)
8	номер: 909009 пароль: qasefflq23qw	(C1)	Тест пройден (рисунок В.5)
9	номер: 50644553 пароль: recsv	(C1, C2)	Тест пройден
10	номер: 890000000000 пароль: 000	(C6)	Тест пройден

Функциональное тестирование показало, что при загрузке набора данных программа работает правильно. Результаты тестирования представлены в приложении В.

4.2 Метрики кода

4.2.1 Метрика SLOC

Количество строк кода (англ. Source Lines of Code – SLOC) – это метрика программного обеспечения, используемая для измерения его объема с помощью подсчета количества строк в тексте исходного кода [21].

Был проведен подсчет данной метрики в методе loadUserByUsername класса UserDetailsServiceImpl который проверяет наличие в базе данных пользователя с указанным номером телефона и возвращает объект данного пользователя. Код функции приведен в листинге 1.

Листинг 1 – Метод LoadUserByUsername

```
public UserDetails loadUserByUsername(String username) throws
UsernameNotFoundException {

    //получение данных пользователя из бд
    UserEntity entity =
    userRepository.findUserEntitiesByUserNumber(username);

    if (entity == null) {
        throw new UsernameNotFoundException("User with username = " +
        username + " not found");
    }
    //Создание объекта пользователя
    UserDTO userDTO = new UserDTO(entity.getUserName(),
    entity.getPassword(), Collections.emptyList());
    userDTO.setId(entity.getId());
    userDTO.setUserNumber(entity.getUserNumber());
    userDTO.setUserSecondName(entity.getUserSecondName());
    userDTO.setUserSurName(entity.getUserSurName());
```

```

userDTO.setUserRole(entity.getUserRole());
return userDTO;
}

```

Физическими строками считаются все непустые строки текстового файла.

Измеряя логические строки кода, предпринимается попытка посчитать количество собственно операторов в программе, но, разумеется, их определение зависит от конкретного языка программирования.

В данной функции имеем:

- 16 физических строк кода;
- 6 логических строк кода;
- 2 строки комментариев.

Значение метрики SLOC равняется 16 физическим строкам кода и 46 логическим.

4.2.2 Цикломатическая сложность

При вычислении цикломатической сложности используется граф потока управления программы. Узлы графа соответствуют неделимым группам команд программы, они соединены ориентированными ребрами, если группа команд, соответствующая второму узлу, может быть выполнена непосредственно после группы команд первого узла.

Граф потока управления программы для функции, представленной в листинге 1, приведен на рисунке 34.

Для вычисления цикломатической сложности $V(G)$ применяется следующая формула:

$$V(G) = e - n + 2p = 8 - 7 + 2 = 3,$$

где e – количество дуг;

n – количество вершин;

p – число компонент связности.

Граф содержит 9 вершин, 13 дуг и один компонент связности. Граф представлен на рисунке 30.

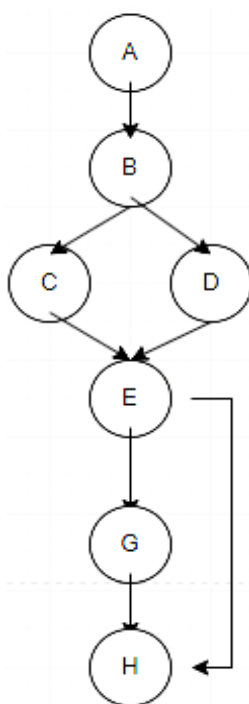


Рисунок 30 – Граф потока

Таким образом значение цикломатической сложности равно 3. Это значение не превышает 10, следовательно, функция не является структурно ложной.

4.2.3 Метрика Чепина

Суть метода состоит в оценке информационной прочности отдельно взятого программного модуля с помощью анализа характера использования переменных из списка ввода-вывода [21].

Все множество переменных, составляющих список ввода-вывода, разбивается на 4 функциональные группы:

- вводимые переменные для расчетов и для обеспечения вывода;
- модифицируемые или создаваемые внутри программы переменные;
- переменные, участвующие в управлении работой программного модуля (управляющие переменные);
- не используемые в программе («паразитные») переменные.

Каждая переменная может выполнять несколько функций одновременно, поэтому необходимо учитывать ее в каждой соответствующей функциональной группе. Метрика Чепина рассчитывается следующим образом:

$$Q = a_1 \times P + a_2 \times M + a_3 \times C + a_4 \times T,$$

где a_1, a_2, a_3, a_4 – весовые коэффициенты.

Для отражения различного влияния на сложность программы каждой группы использованы весовые коэффициенты. Наибольший вес, равный 3, по мнению Чепина, имеет функциональная группа С, так как она влияет на поток управления программы. Весовые коэффициенты остальных групп распределяются следующим образом: $a_1 = 1$, $a_2 = 2$, $a_4 = 0.5$. Неиспользуемые переменные имеют ненулевой коэффициент. Несмотря на то, что они не увеличивают сложность программы, иногда они затрудняют ее понимание. В представленной в листинге 1 функции $P = 1$, $M = 2$, $C = 0$, $T = 0$ [18]. Учитывая весовые коэффициенты:

$$Q = 1 + 2 \times 2 + 3 \times 0 + 0.5 \times 0 = 5.$$

Значение метрики Чепина равняется 5.

Если в результате изменения рассматриваемых функций значение метрики значительно увеличиться, это будет означать низкое качество изменений.

4.3 Планирование разработки и оценка бюджета

Планирование разработки и оценка бюджета является важной частью реализации проекта. С помощью планирования можно установить цели проекта, а также определить материальные и временные затраты на реализацию отдельных частей системы.

Планирование разработки и оценка бюджета была выполнена при помощи программы для управления проектами Microsoft Project [22]. Проект был начат 8 апреля 2024 года.

При планировании разработки были основные задачи. Список задач и их описание представлено на рисунке 31.

На рисунке 34 приведена статистика по проекту.

Статистика проекта для 'Проект1'				×
	Начало		Окончание	
Текущее	Пн 08.04.24		Пн 03.06.24	
Базовое	НД		НД	
Фактическое	НД		НД	
Отклонение	0 дней		0 дней	
	Длительность	Трудозатраты	Затраты	
Текущие	40 дней	784 ч	196 000,00 Р	
Базовые	0 дней	0 ч	0,00 Р	
Фактические	0 дней	0 ч	0,00 Р	
Оставшиеся	40 дней	784 ч	196 000,00 Р	
Процент завершения				
Длительность: 0%		Трудозатраты: 0%		Заккрыть

Рисунок 34 – Статистика проекта

Таким образом разработка проекта составит 40 полных рабочих дней с 8.04.2024 по 3.06.2024 года. Трудозатраты составят 748 человеко-часов. Оценка бюджета проекта составляет 196 000 рублей.

Заключение

В ходе выполнения выпускной квалификационной работы разработаны программные средства администрирования для приложения электронной коммерции по продаже 3d-изделий.

Проведен анализ приложений электронной коммерции и выделены основные достоинства и недостатки их реализации. На данной основе выделены функциональные и нефункциональные требования к программным средствам.

Для реализации данных программных средств выбрана архитектура системы, спроектирован алгоритм доступа к защищенным данным. Также спроектирована логическая и физическая схема БД, отображающая основные сущности и связи в выбранной предметной области. Для основных модулей построены диаграммы классов. Работа основных сервисов системы представлена на диаграммах UML, таких как диаграмма вариантов использования и последовательности.

Разработка программных средств проводилась в средах разработки IntelliJ idea и WebStorm на языках программирования Java и JavaScript. Разработанные программные средства функционирует на любом современном веб-браузере.

Разработанные программные средства протестированы. Выполнено функциональное тестирование. Все тесты пройдены успешно.

Также проведено планирование и оценка бюджета проекта при помощи программы для управления проектами Microsoft Project. Затраты на проект составили 196 000 рублей. Время разработки 40 рабочих дня.

Таким образом, в результате выполнения выпускной квалификационной работы разработаны программные средства администрирования, отвечающие всем требованиям технического задания.

Список использованных источников

1. Все, что вам нужно знать о защите платежей в электронной коммерции [Электронный ресурс]. – URL: <https://appmaster.io/ru/blog/bezopasnost-platezhei-v-elektronnoi-kommertsii> (дата обращения: 24.05.2024).
2. OAuth2.1 // OAuth.net [Электронный ресурс]. – URL: <https://oauth.net/2.1/> (дата обращения: 24.05.2024).
3. OpenID Connect Core 1.0 // OpenID [Электронный ресурс]. – URL: https://openid.net/specs/openid-connect-core-1_0-final.html (дата обращения: 24.05.2024).
4. 3д печать в Москве // 3dmf [Электронный ресурс]. – URL: <https://3dmf.ru/> (дата обращения: 24.05.2024).
5. Сообщество владельцев 3D-принтеров –3DToday – описание 3D-принтеров, отзывы о 3D-принтерах, характеристики, фотографии и описания, распечатанные 3D-модели и многое другое // 3dtoday [Электронный ресурс]. – URL: <https://3dtoday.ru/> (дата обращения: 24.05.2024).
6. Заказать производство/изготовление деталей из пластика и полиуретанов в Пензе// Volprint [Электронный ресурс]. – URL: <https://penza.volprint.ru/> (дата обращения: 24.05.2024).
7. 3D печать в Пензе – услуги 3D-принтера, цены // Asteri–3D [Электронный ресурс]. – URL: <https://asteri-3d.ru/cities/penza/> (дата обращения: 24.05.2024).
8. Анализ требований // Хабр [Электронный ресурс]. – URL: <https://habr.com/ru/articles/340956/> (дата обращения: 24.05.2024).
9. Spring | Why Spring // Broadcom [Электронный ресурс]. – URL: <https://spring.io/microservices> (дата обращения: 24.05.2024).
10. Что такое RESTful на самом деле // Habr [Электронный ресурс]. – URL: <https://habr.com/ru/companies/hexlet/articles/274675/> (дата обращения: 24.05.2024).

11. IntelliJ IDEA – the Leading Java and Kotlin IDE //JetBrains [Электронный ресурс]. – URL: <https://www.jetbrains.com/idea/> (дата обращения: 24.05.2024).
12. Что такое MVC: базовые концепции // skillbox [Электронный ресурс]. – URL: https://skillbox.ru/media/code/chto_takoe_mvc_bazovye_kontseptsii_i_primer_prilozheniya/ (дата обращения: 24.05.2024).
13. Использование шлюзов API в микрослужбах. // Microsoft [Электронный ресурс]. – URL: <https://learn.microsoft.com/ru-ru/azure/architecture/microservices/design/gateway> (дата обращения: 24.05.2024).
14. <https://microservices.io/patterns/apigateway.html> //microservices.io [Электронный ресурс]. – URL: <https://microservices.io/patterns/apigateway.html> (дата обращения: 25.05.2024)
15. Основные особенности и виды архитектур клиент-сервер [Электронный ресурс]. – URL: <https://www.ittelo.ru/news/osnovnye-osobennosti-i-vidy-arkhitektur-klient-server/> (дата обращения: 25.05.2024)
16. О микросервисной архитектуре простыми словами // skillbox [Электронный ресурс]. – URL: <https://skillbox.ru/media/code/o-mikroservisnoy-arkhitecture-prostymi-slovami/> (дата обращения: 25.05.2021)
17. draw.io // draw.io [Электронный ресурс]. – URL: <https://www.draw.io/> (дата обращения: 25.05.2024)
18. <https://habr.com/ru/articles/556790/>
19. React.js: понятное руководство для начинающих // Хабр [Электронный ресурс]. – URL: <https://habr.com/ru/companies/ruvds/articles/428077/> (дата обращения: 25.05.2024)
20. Миграции баз данных с Flyway // Хабр [Электронный ресурс]. – URL: <https://habr.com/ru/companies/otus/articles/506788/> (дата обращения: 25.05.2024)
21. Программный код и его метрики // Хабр [Электронный ресурс]. – URL: <https://habr.com/ru/companies/intel/articles/106082/> (дата обращения: 27.05.2024)

22. ПО для управления проектами // Microsoft Project [Электронный ресурс]. – URL: <https://www.microsoft.com/ru-ru/microsoft-365/project/project-management-software> (дата обращения: 25.05.2024)

Приложение А
(справочное)
Глоссарий

База данных (БД) – это информативная модель, позволяющая организованно поддерживать данные о группе объектов, имеющих один и тот же набор параметров.

Unified Modeling Language (UML) – графический унифицированный язык моделирования, использующий объектно-ориентированный подход для моделирования в сфере разработки программного обеспечения.

ORM – технология, которая связывает базы данных с концепциями объектно-ориентированных языков программирования, создавая «виртуальную объектную базу данных».

REST – Representation State Transfer (передача репрезентативного состояния), архитектурный стиль взаимодействия компонентов распределенного приложения в сети.

API – Application Programming Interface (интерфейс программирования приложений), совокупность инструментов и функций в виде интерфейса для создания новых приложений, благодаря которому одна программа будет взаимодействовать с другой.

PostgreSQL – свободная объектно-реляционная система управления базами данных.

Hibernate – библиотека для языка программирования Java, предназначенная для решения задач объектно-реляционного отображения.

Приложение Б
(рекомендуемое)
Листинг приложения

Листинг Б.1 – Файл SecConfig.java

```

@Configuration
@EnableWebSecurity
@EnableMethodSecurity
public class SecConfig implements WebMvcConfigurer {

    @Bean
    public SecurityFilterChain securityFilterChain(HttpSecurity http)
    throws Exception {
        http

        .authorizeHttpRequests(authorizationManagerRequestMatcherRegistry ->
            authorizationManagerRequestMatcherRegistry
                .requestMatchers("/").permitAll()
                .requestMatchers("/items/**").permitAll()
                .anyRequest().authenticated())

        .oauth2ResourceServer(httpSecurityOAuth2ResourceServerConfigurer ->
            httpSecurityOAuth2ResourceServerConfigurer
                .jwt(Customizer.withDefaults()));

        return http.build();
    }

    @Bean
    public FilterRegistrationBean<CorsFilter>
    corsFilterFilterRegistrationBean() {
        UrlBasedCorsConfigurationSource source = new
        UrlBasedCorsConfigurationSource();
        CorsConfiguration config = new CorsConfiguration();

        config.setAllowCredentials(true);

        // Указываем список адресов для которых разрешены кросс-доменные
        запросы

        config.addAllowedOrigin("http://127.0.0.1:3000,http://localhost:3000");
        config.addAllowedHeader(CorsConfiguration.ALL);
        config.addExposedHeader(CorsConfiguration.ALL);
        config.addAllowedMethod(CorsConfiguration.ALL);

        source.registerCorsConfiguration("/**", config);
        FilterRegistrationBean<CorsFilter> bean = new
        FilterRegistrationBean<>(new CorsFilter(source));
        bean.setOrder(Ordered.HIGHEST_PRECEDENCE);
        return bean;
    }
}

```

Листинг Б.2 – Файл application.yaml

```

spring:
  cloud:
    gateway:
      mvc:

```

```

routes:
- id: catalog
  uri: http://localhost:8081
  predicates:
    - Path=/items
  filters:
    - RemoveRequestHeader=Cookie
- id: order
  uri: http://localhost:8086
  predicates:
    - Path=/worker/**
  filters:
    - TokenRelay=
    - RemoveRequestHeader=Cookie
- id: resource
  uri: http://localhost:8085
  predicates:
    - Path=/user/**
  filters:
    - TokenRelay=
    - RemoveRequestHeader=Cookie
- id: Manager
  uri: http://localhost:8084
  predicates:
    - Path=/manager/**
  filters:
    - TokenRelay=
    - RemoveRequestHeader=Cookie

security:
  oauth2:
    resourceserver:
      jwt:
        issuer-uri: "http://localhost:9000"
        jwk-set-uri: "http://localhost:9000/oauth2/jwks"
  application:
    name: gateway

server:
  port: 8080

servlet:
  session:
    cookie:
      name: "GATEWAY_SESSION"
logging:
  level:
  org:
  springframework:
    security: debug
    session: debug
    reactor.netty: debug

```

Листинг Б.3 – Файл AuthServerConfig.java

```
package com.sv.idp.config;
```

```

@RequiredArgsConstructor
@Configuration(proxyBeanMethods = false)
@EnableWebSecurity
@EnableMethodSecurity
public class AuthServerConfig {
    private final AuthorizationServerProperties
authorizationServerProperties;

    @Bean
    @Order(Ordered.HIGHEST_PRECEDENCE)
    public SecurityFilterChain authServerSecurityFilterChain(HttpSecurity
http) throws Exception {

OAuth2AuthorizationServerConfiguration.applyDefaultSecurity(http);
    Function<OidcUserInfoAuthenticationContext, OidcUserInfo>
userInfoMapper = (context) -> {
        OidcUserInfoAuthenticationToken authentication =
context.getAuthentication();
        JwtAuthenticationToken principal = (JwtAuthenticationToken)
authentication.getPrincipal();
        return new
OidcUserInfo(userInfoMapper.getClaims(principal.getTokenAttributes()));
    };
    http
        .getConfigurer(OAuth2AuthorizationServerConfigurer.class)
        .oidc(
            oidcConfigurer -> oidcConfigurer
                .logoutEndpoint(Customizer.withDefaults())
                .userInfoEndpoint(oidcUserInfoEndpointConfigurer
-> oidcUserInfoEndpointConfigurer
                    .userInfoMapper(userInfoMapper))
        );
    http.oauth2ResourceServer((oauth2) ->
oauth2.jwt(Customizer.withDefaults()));
    return http.exceptionHandling(exceptions -> exceptions
        .authenticationEntryPoint(new
LoginUrlAuthenticationEntryPoint("/login"))
    ).build();
}
@Bean
    public RegisteredClientRepository registeredClientRepository() {
        RegisteredClient registeredClient =
RegisteredClient.withId(UUID.randomUUID().toString())
            .clientId("gateway")
            .clientSecret("{noop}secret")

//.clientSettings(ClientSettings.builder().requireAuthorizationConsent(tr
ue).build())

        .clientAuthenticationMethod(ClientAuthenticationMethod.CLIENT_SECRET_BASI
C)

        .authorizationGrantType(AuthorizationGrantType.AUTHORIZATION_CODE)

        .authorizationGrantType(AuthorizationGrantType.REFRESH_TOKEN)
            .redirectUri("http://localhost:3000/")
            .postLogoutRedirectUri("http://localhost:9000/login")
    }
}

```

```

        .scope(OidcScopes.OPENID)
        .scope(OidcScopes.PROFILE)
        .scope("message.read")
        .scope("message.write")
        .build();

    return new InMemoryRegisteredClientRepository(registeredClient);
}

@Bean
public JWKSource<SecurityContext> jwkSource() {
    KeyPair keyPair = JwkUtils.generateRsaKey();
    RSAPublicKey publicKey = (RSAPublicKey) keyPair.getPublic();
    RSAPrivateKey privateKey = (RSAPrivateKey) keyPair.getPrivate();
    RSAKey rsaKey = new RSAKey.Builder(publicKey)
        .privateKey(privateKey)
        .keyID(UUID.randomUUID().toString())
        .build();
    JWKSet jwkSet = new JWKSet(rsaKey);
    return new ImmutableJWKSet<>(jwkSet);
}

@Bean
public AuthorizationServerSettings authorizationServerSettings() {
    return AuthorizationServerSettings.builder()
        .issuer(authorizationServerProperties.getIssuerUrl())
        .build();
}

@Bean
public OAuth2TokenCustomizer<JwtEncodingContext> tokenCustomizer() {
    return (context) -> {
        if
(OAuth2TokenType.ACCESS_TOKEN.equals(context.getTokenType())) {
            OidcUserInfo userInfo =
OidcUserInfoService.oidcUserInfo((UserDTO)
context.getPrincipal().getPrincipal());
            context.getClaims().claims(claims -> claims
                .putAll(userInfo.getClaims()));
        }
    };
}
}
}

```

Листинг Б.4 – Файл SecurityConfig.java

```

package com.sv.idp.config;

@EnableMethodSecurity
@EnableWebSecurity
@RequiredArgsConstructor
@Configuration(proxyBeanMethods = false)
public class SecurityConfig implements WebMvcConfigurer {
    private final UserDetailsServiceImpl userDetailsService;

    @Bean
    public SecurityFilterChain defaultSecurityFilterChain(HttpSecurity
http) throws Exception {
        http.csrf(AbstractHttpConfigurer::disable)

```

```

        .getSharedObject(AuthenticationManagerBuilder.class)
        .userDetailsService(userDetailsService)
        .passwordEncoder(new BCryptPasswordEncoder());

http.authorizeHttpRequests(authorizationManagerRequestMatcherRegistry ->
    authorizationManagerRequestMatcherRegistry

    .requestMatchers("/registration/**").permitAll()
        .requestMatchers("/login").permitAll()

    .requestMatchers("/register/**").permitAll()
        .requestMatchers("/css/**").permitAll()
        .requestMatchers(HttpMethod.OPTIONS,
"/oauth2/token").permitAll()
        .anyRequest().authenticated())
    .formLogin(httpSecurityFormLoginConfigurer ->
httpSecurityFormLoginConfigurer
        .defaultSuccessUrl("/")
        .loginPage("/login"))
    .logout(withDefaults());
return http.build();
}

}

```

Листинг Б.5 – Файл ManagerService.java

```

@Service
public class ManagerService {
    private final RestClient restClient;
    private final PlasticRepository plasticRepository;
    private final ItemRepository itemRepository;
    public ManagerService(RestClient.Builder restClientbuilder,
PlasticRepository plasticRepository, ItemRepository itemRepository) {
        this.restClient =
restClientbuilder.baseUrl("http://localhost:8085").build();
        this.plasticRepository = plasticRepository;
        this.itemRepository = itemRepository;
    }
    @Transactional
    public List<CheckDTO> gettAll(String token) {
        List<OrderDTO> orderDTOs =
restClient.method(HttpMethod.GET).uri("manager/allorders").headers(httpHe
aders -> httpHeaders.setBearerAuth(token)).retrieve().body(new
ParameterizedTypeReference<>() {
        });
        List<CheckDTO> checkDTOs = new ArrayList<>();
        assert orderDTOs != null;
        for ( OrderDTO orderDTO: orderDTOs){
            UserDTO userDTO = new UserDTO();
            userDTO.setId(orderDTO.getUser_id());
            userDTO.setUserSecondName(orderDTO.getUsersecondname());
            userDTO.setUsername(orderDTO.getUsername());
            userDTO.setUserSurName(orderDTO.getUsersurname());
            CheckDTO checkDTO = new CheckDTO();
            checkDTO.setId(orderDTO.getChek_id());
            checkDTO.setStatus(orderDTO.getStatus());
            checkDTO.setPrice(orderDTO.getPrice());

```

```

        checkDTO.setDataseil(orderDTO.getDataseil());
        checkDTO.setUser(userDTO);
        Item item =
itemRepository.getItemByNameAndColors(orderDTO.getTitle(), new
Color(orderDTO.getColors()));
        Plastic plastic = plasticRepository.getPlasticByColor(new
Color(orderDTO.getColors()));
        if (plastic.getPlasticweight() - item.getPlasticweight() >= 0){
            checkDTO.setAcc(true);
            plastic.setPlasticweight(plastic.getPlasticweight() -
item.getPlasticweight());
        }
        else {
            checkDTO.setAcc(false);
        }
    }
    return checkDTOs;
}
    public void setNewStatus(ReqBody reqBody, String token){
        restClient.post().uri("manager/allorders").headers(httpHeaders ->
httpHeaders.setBearerAuth(token)).body(reqBody).retrieve().toBodilessEnti
ty();
    }
}

```

Листинг Б.6 – Файл CheckService.java

```

Service
public class CheckService {
    @Autowired
    ItemsRepository itemsRepository;
    @Autowired
    CheckRepository checkRepository;
    @Autowired
    UserRepository userRepository;
    @Autowired
    OrderedItemsRepository orderedItemsRepository;
    @Autowired
    OrdersRepository ordersRepository;

    @Transactional
    public boolean create(CheckDTO checkDTO, Long id){
        List<ItemsDTO> e = checkDTO.getItemsDTOS();
        List<Ordereditem> ordereditems = new ArrayList<>();
        Ordereditem ordereditem;
        Date date = new Date();
        UserEntity userEntity = userRepository.getReferenceById(id);
        CheckEntity checkEntity = new CheckEntity();
        checkEntity.setDataseil(date);
        checkEntity.setUser(userEntity);
        checkEntity.setStatus("ACTIVE");
        checkEntity.setPrice((long) checkDTO.getTotalPrice());

        for (ItemsDTO itemsDTO: e) {
            ItemsEntity itemsEntity =
itemsRepository.getItemsEntitiesByColorsAndTitle(itemsDTO.getColorName(),
itemsDTO.getTitle());
            if(itemsEntity!=null){
                ordereditem = new Ordereditem();

```

```

        ordereditem.setChek(checkEntity);
        ordereditem.setItems(itemsEntity);
        ordereditem.setCount(itemsDTO.getCount());
        ordereditems.add(ordereditem);
    }
} checkRepository.save(checkEntity);
orderedItemsRepository.saveAll(ordereditems);
return true;
}

public CheckEntity getByOne(Long id){
    return checkRepository.getCheckEntityById(id);
}

public List<CheckEntity> gettall(Long id){
    return checkRepository.getCheckEntitiesByUser(userRepository
.getReferenceById(id));
}
public List<CheckEntity> gettallforworker(){
    return checkRepository.getAllByStatus("PENDING");
}
public List<OrderView> gettallforManager(){
    return ordersRepository.getAll();
}

@Transactional
public String refreshCheckStatus(Long id){
    CheckEntity checkEntity = checkRepository.getReferenceById(id);
    checkEntity.setStatus("DONE");
    checkRepository.save(checkEntity);
    return "done";
}

@Transactional
public String setNewStatus(Long id, String status){
    CheckEntity checkEntity = checkRepository.getCheckEntityById(id);
    if (checkEntity == null) {
        return "not found";
    }
    if(status.equals("PENDING")){
        checkEntity.setStatus("PENDING");
        return "true-pending";
    }

    if(status.equals("REJECTED")){
        checkEntity.setStatus("REJECTED");
        return "true-rejected";
    }
    return "false";
}
}

```

Листинг Б.7 – Файл UserDetailsServiceImpl.java

```

package com.sv.idp.Service;

@Service
@RequiredArgsConstructor

```

```

public class UserDetailsServiceImpl implements UserDetailsService {
    private final UserRepository userRepository;

    @Transactional
    public boolean saveUser(UserControllerDTO userDTO) {
        UserEntity entity =
userRepository.findUserEntitiesByUserNumber(userDTO.getUserNumber());
        if (entity == null) {
            entity = new UserEntity();
            entity.setUsername(userDTO.getUsername());
            entity.setUserSecondName(userDTO.getUserSecondName());
            entity.setUserRole("USER");
            entity.setUserNumber(userDTO.getUserNumber());
            entity.setUserSurName(userDTO.getUserSurName());
            entity.setPassword( new
BCryptPasswordEncoder().encode(userDTO.getPassword()));
            userRepository.save(entity);

            return true;
        }
        else return false;
    }

    @Override
    public UserDetails loadUserByUsername(String username) throws
UsernameNotFoundException {
        UserEntity entity =
userRepository.findUserEntitiesByUserNumber(username);
        if (entity == null) {
            throw new UsernameNotFoundException("User with username = " +
username + " not found");
        }

        UserDTO userDTO = new UserDTO(entity.getUserName(),
entity.getPassword(), Collections.emptyList());
        userDTO.setId(entity.getId());
        userDTO.setUserNumber(entity.getUserNumber());
        userDTO.setUserSecondName(entity.getUserSecondName());
        userDTO.setUserSurName(entity.getUserSurName());
        userDTO.setUserRole(entity.getUserRole());
        return userDTO;
//
//            User.builder()
//            .username(entity.getUserName())
//            .password(entity.getPassword())
//            .roles(entity.getUserRole())
//            .build();
//            new UserDTO(entity.getUserName(),
entity.getPassword(), Collections.emptyList(), entity.getId(),
entity.getUserRole(),
entity.getUserSecondName(), entity.getUserSurName(), entity.getUserNumber()
);
    }

}

```

Листинг Б.8 – Файл JwkUtils.java

```
package com.sv.idp.config;

import java.security.KeyPair;
import java.security.KeyPairGenerator;

public class JwkUtils {

    public static KeyPair generateRsaKey() {
        KeyPair keyPair;
        try {
            KeyPairGenerator keyPairGenerator =
                KeyPairGenerator.getInstance("RSA");
            keyPairGenerator.initialize(2048);
            keyPair = keyPairGenerator.generateKeyPair();
        } catch (Exception ex) {
            throw new IllegalStateException(ex);
        }
        return keyPair;
    }
}
```

Листинг Б.9 – Файл Login.html

```
</head>
<body>
<div class="container">

    <form th:action="@{/login}" id="form" method="post">
        <h1>Вход в аккаунт</h1>
        <div class="input-control">
            <label for="UserName">Номер телефона</label>
            <input id="UserMame" name="Username" type="text">
        </div>
        <div class="input-control">
            <label for="passsword">Пароль</label>
            <input id="passsword" name="passsword" type="text">
            <div th:if="${param.error}" class="error">

                </div>
        </div>
        <div>У вас еще нет аккаунта?
            <a th:href="@{/registration}">Зарегистрироваться</a>
        </div>
        <button type="submit">Войти</button>
        <span></span>
    </div>
</form>
</div>
</body>
</html>
```

Листинг Б.10 – Файл register.html

```
</head>
<body>
<div class="container">
    <form th:action="@{/registration}"
```

```

        th:object="${user}" id="form" method="post">
<h1>Регистрация</h1>
<div class="input-control">
    <label for="name">Имя</label>
    <input th:field = "{username}" id="name" name="name"
type="text">
    <div class="error">Некоректно введенное имя</div>
</div><div class="input-control">
    <label for="Lastname">Фамилия</label>
    <input th:field = "{userSecondName}" id="Lastname"
name="Lastname" type="text">
    <div class="error">Некоректно введенная фамилия</div>
</div>
<div class="input-control">
    <label for="Surname">Отчество</label>
    <input th:field = "{userSurName}" id="Surname"
name="Surname" type="text">
    <div class="error">Некоректно введенное отчество</div>
</div>
<div class="input-control">
    <label for="phone">Номер телефона</label>
    <input th:field = "{userNumber}" id="phone" name="phone"
type="text">
    <div class="error">Некоректно введенный номер</div>
</div>
<div class="input-control">
    <label for="password">Пароль</label>
    <input th:field = "{password}" id="password" name="password"
type="password">
</div>
<div>У вас уже есть аккаунт?
    <a th:href="@{/login}">Войти</a>
</div>
<div class="error"></div>
<div th:if="{create == false}" class="error">данный номер
телефона уже зарегистрирован</div>
<button type="submit">Зарегистрироваться</button>
<span></span>
</form>
</div>
</body>
</html>

```

Листинг Б.11 – Файл home.tsx

```

export interface IAccWorker {
    id: number
    datasell: string
    price: number
    status: string
    user: IUserAcc
}

export interface IAccManager extends IAccWorker {
    acc: boolean
}

export interface IUserAcc {
    password:string
}

```

```

    userName: string
    userNumber: string
    userRole: string
    userSecondName:string
    id:number
    userSurName:string
  }

const Account = () => {
  const dispatch = useAppDispatch()
  const {account} = useSelector(selectAccountData)
  const [managerRes, setMangerRes] = useState<IAccManager[]>([])
  const [workerRes, setWorkerRes] = useState<IAccWorker[]>([])
  let managerData;

  useEffect(()=> {
    onStartAccount()
  }, [])

  const onStartAccount = async () => {
    await getAcc()
    if (account.roles === "ROLE_MANAGER"){
      await managerOrders()
    }
    else if (account.roles === "ROLE_WORKER"){
      await workerOrders()
    }
  }

  const workerOrders = async () => {
    try{
      const res = await
    $api.get<IAccWorker[]>('http://localhost:8080/worker/getallorders',{
      headers: { 'Content-Type': "multipart/form-data"},
      withCredentials:true,
    })
      setWorkerRes(res.data)
    }catch (e){
      console.log(e)
    }
  }

  const managerOrders = async () => {
    try{
      const res = await
    $api.get<IAccManager[]>('http://localhost:8080/manager/getallorders',{
      headers: { 'Content-Type': "multipart/form-data"},
      withCredentials:true,
    })
      setMangerRes(res.data)
      console.log(res.data)
    }catch (e){
      console.log(e)
    }
  }
}

```

```

    }

    const getAcc = async ()=>{
      try{
        const res = await
$api.get<IAccount>('http://localhost:9000/userinfo',{
          headers: { 'Content-Type': "multipart/form-data"},
          withCredentials:true,
        })
        const data = res.data
        dispatch(setAccount(data))
      }catch (e){
        console.log(e)
      }
    }

    if (account.roles === "ROLE_MANAGER"){
      managerData = managerRes.map(obj => <ManagerData key={obj.id}
value={obj}/>)
    }

    else if ( account.roles === "ROLE_WORKER"){
      managerData = workerRes.map(obj => <WorkerData key={obj.id}
value={obj}/>)
    }

    return (
      <div>
        <AccountData />
        {managerData}
      </div>
    );
  };
};

export default Account;

```

Листинг Б.12 – OidcUserInfoService.java

```

@Service
public class OidcUserInfoService {

    public static OidcUserInfo oidcUserInfo(UserDTO userDTO) {
        return OidcUserInfo.builder()
            .subject(userDTO.getName())
            .name(userDTO.getName())
            .claim("id", userDTO.getId())
            .claim("roles", "ROLE_"+userDTO.getUserRole())
            .claim("Last_name", userDTO.getUserSecondName())
            .middleName(userDTO.getUserSurName())
            .phoneNumber(userDTO.getUserNumber())
            .build();
    }
}

```

Листинг Б.13 – OidcUserInfoService.java

```

@Service
public class WorkerService {
    private final RestClient restClient;
    private final OrderRepository orderRepository;
    public WorkerService(RestClient.Builder restClientbuilder,
        OrderRepository orderRepository) {
        this.restClient =
restClientbuilder.baseUrl("http://localhost:8085").build();
        this.orderRepository = orderRepository;
    }
    @Transactional
    public String setOrder(Long id, String token){
        CheckDTO checkDTO =
restClient.method(HttpMethod.GET).uri("worker/order/"+id).headers(httpHea
ders ->
httpHeaders.setBearerAuth(token)).retrieve().body(CheckDTO.class);
        assert checkDTO != null;
        Order order = new Order();
        order.setTimeComp(new Date());
        order.setTimeAcc(checkDTO.getDatase11());
        order.setCheks_id(checkDTO.getId());
        orderRepository.save(order);
        restClient.post().uri("worker/allorders")
.headers(httpHeaders -> httpHeaders.setBearerAuth(token)).body("id: " +
id).retrieve().toBodilessEntity();
        return "true";
    }

    public List<CheckDTO> gettAll(String token) {
List<CheckDTO> checkDTOS =
restClient.method(HttpMethod.GET).uri("worker/allorders")
.headers(httpHeaders -> httpHeaders.setBearerAuth(token)).retrieve()
.body(new ParameterizedTypeReference<>() {
    });
        return checkDTOS;
    }
}

```

Листинг Б.14 – ItemsMapper.java

```

@Service
public class ItemsMapper {
    public Items toItemS(ItemsDTO itemsDTO){
        Items items = new Items();
        items.setId(itemsDTO.getId());
        items.setName(itemsDTO.getTitle());
        items.setColors(itemsDTO.getColorsName());
        items.setPrice(itemsDTO.getPrice());
        items.setColors_hash(itemsDTO.getColorsHash());
        items.setImageName(itemsDTO.getImageURL());
        items.setRating(itemsDTO.getRating());
        return items;
    }

    public ItemsDTO toItemSDto(Items items){

```

```
if(items!= null) {  
    ItemsDTO itemsDto = new ItemsDTO();  
    itemsDto.setId(items.getId());  
    itemsDto.setTitle(items.getName());  
    itemsDto.setColorsName(items.getColors());  
    itemsDto.setPrice(items.getPrice());  
    itemsDto.setColorsHash(items.getColors_hash());  
    itemsDto.setImageURL(items.getImageName());  
    itemsDto.setRating(items.getRating());  
    return itemsDto;  
}  
return new ItemsDTO();}
```

Приложение В
(рекомендуемое)
Результаты тестирования

Вход в аккаунт

Номер телефона

Co!

Пароль

.re!.csv

Некорректная длина номера
Некорректная длина пароля
Некорректные символы в пароле
HНекорректные символы в номере
У вас еще нет аккаунта? Зарегистрироваться

Войти

Рисунок В.1 – Тест 1, покрывающий таблицу решений

Вход в аккаунт

Номер телефона

89085220326

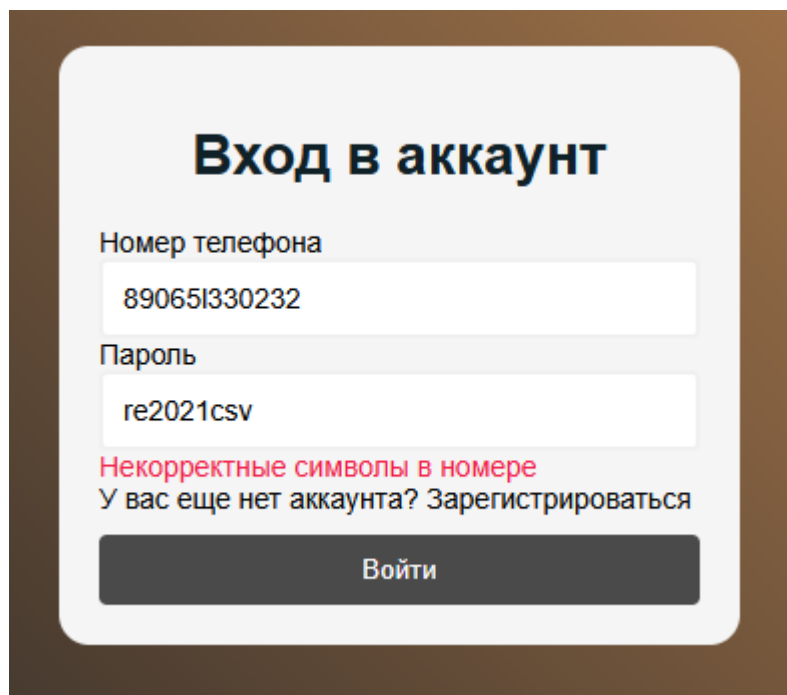
Пароль

.re@.csv

Некорректная длина пароля
Некорректные символы в пароле
У вас еще нет аккаунта? Зарегистрироваться

Войти

Рисунок В.2 – Тест 2, покрывающий таблицу решений



Вход в аккаунт

Номер телефона

890651330232

Пароль

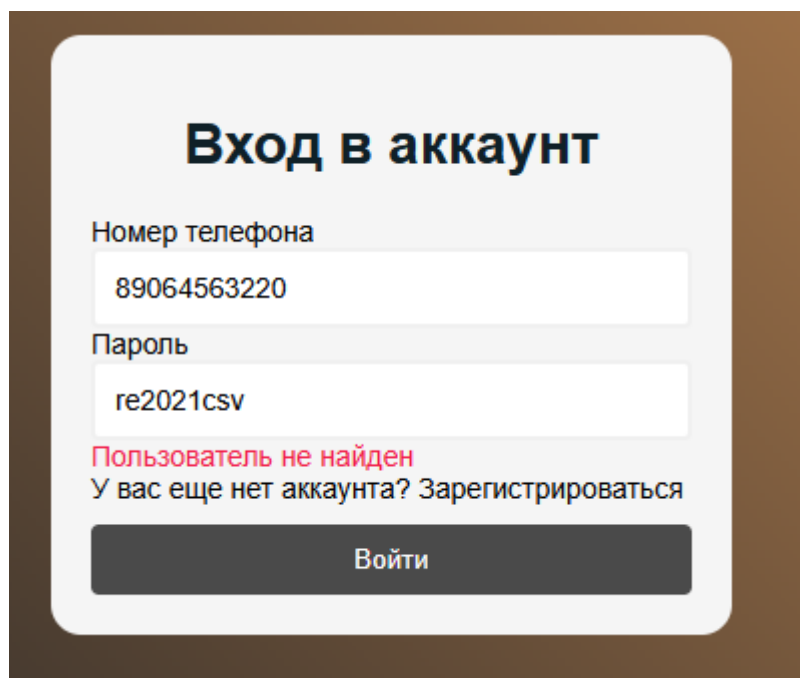
re2021csv

Некорректные символы в номере

У вас еще нет аккаунта? Зарегистрироваться

Войти

Рисунок В.3 – Тест 3, покрывающий таблицу решений



Вход в аккаунт

Номер телефона

89064563220

Пароль

re2021csv

Пользователь не найден

У вас еще нет аккаунта? Зарегистрироваться

Войти

Рисунок В.4 – Тест 4, покрывающий таблицу решений

Вход в аккаунт

Номер телефона

909009

Пароль

qasefflq23qw

Некорректная длина номера

У вас еще нет аккаунта? [Зарегистрироваться](#)

Войти

Рисунок В.5 – Тест 5, покрывающий таблицу решений

Приложение Г
(рекомендуемое)
Диаграмма классов сервера авторизации

