МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет
вычислительной техники

Кафедра
«Вычислительная техника»

Направление подготовки: 09.03.01 «Информатика и вычислительная техника»

Профиль подготовки: «Программное обеспечение средств вычислительной техники и автоматизированных систем»

БАКАЛАВРСКАЯ РАБОТА

на тему

WEB-ПОРТАЛ МАРКЕТПЛЕЙС-АГРЕГАТОРА

Студент _____ Юдин Денис Александрович
(подпись, дата) (ФИО полностью)

Руководитель _____ Дубравин А. В.
(подпись, дата) (фамилия, инициалы)

Нормоконтролёр _____ А.С. Бычков
(подпись, дата) (фамилия, инициалы)

Работа допущена к защите (протокол заседания кафедры от . .2023 г. №)

Заведующий кафедрой _____ М.А. Митрохин
(подпись) (фамилия, инициалы)

Работа защищена с отметкой удовлетворительно (протокол заседания ГЭК от 27.06.2023г. № 33)

Секретарь ГЭК _____ Е.И. Гурин
(подпись) (фамилия, инициалы)

Пенза 2023

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет	*Кафедра*
вычислительной техники	«Вычислительная техника»

«Утверждаю»
Заведующий кафедрой
_____М.А. Митрохин
«26» 04 2023г.

ЗАДАНИЕ
НА БАКАЛАВРСКУЮ РАБОТУ

1. Студент Юдин Денис Александрович гр. 19ВВП1 факультета вычислительной техники.

2. Тема работы «Web – портал маркетплейс - агрегатора»

Тема утверждена приказом ПГУ №458/о от "26" 04 2023 г.

3. Руководитель бакалаврской работы к.т.н., доцент Дубравин А.В.

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

Разработать приложение «*Web*-портал маркетплейс-агрегатора», выполняющее следующие функции:

– мульпользовательский доступ к приложению;

– возможность просмотра, загрузки и редактирования товаров и их категорий;

– возможность поиска, бронирования товара и оформление заказа, с возможностью установки и просмотра статуса заказа.

Данное приложение должно быть спроектировано на основе модели клиент серверной архитектуры с использованием паттернов *MVC* и *REST*. Клиентская часть должна обладать визуальным графическим интерфейсом, а серверная должна обрабатывать запросы и работать с базой данных.

5. Перечень подлежащих разработке вопросов

Требуется произвести анализ предметной области: изучить аналоги существующих маркетплейс-площадок, изучить способы проектирования и разработки клиент-серверного приложения с применением *REST API* методологии. Необходимо провести проектирование и разработку приложения с использованием фреймворка *Spring* и языка программирования Java.

6. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1.	Изучение предметной области	50	26.04.2023-30.04.2023	
2.	Анализ требований к проектированию и разработки клиент – серверных приложений	10	31.04.2023-02.05.2023	
3.	Выбор технологий, применяемых при проектировании и разработке клиент – серверных приложений	50	03.05.2023-07.05.2023	
4.	Описание структур, алгоритмов проектирования и реализации маркетплейс – площадки на базе клиент – серверного приложения	40	08.05.2023-11.05.2023	
5.	Разработка приложения «*Web*-портал маркетплейс-агрегатора»	130	12.05.2023-27.05.2023	
6.	Тестирование разработанного ПО	20	27.05.2023-29.05.2023	
7.	Оформление пояснительной записки	24	30.05.2023-31.05.2023	

Дата выдачи задания "26" апреля 2023 г.

Руководитель бакалаврской работы_____Дубравин А.В.
(подпись, дата)

Задание к исполнению принял студент_____Юдин Д.А.
(подпись, дата)

Работу к защите допустить
Декан факультета _____Фионова Л.Р.
(подпись, дата)
27.06.2023

Реферат

Работа содержит 92 страницы формата А4, 29 рисунков, 10 источников, 2 приложения.

WEB - ПОРТАЛ, *BOOTSTRAP*, *MVC*, *ANGULAR*, *HIBERNATE*, *POSTGESQL*, *JAVA*, *SPRING*.

Объект разработки – *Web* - портал маркетплейс-агрегатора.

Целью дипломной работы является разработка приложения для возможности осуществления интернет - торговли в рамках *web* - ресурса.

В процессе работы над программным обеспечением был проведен анализ требований к разрабатываемому продукту, анализ существующих современных средств проектирования ПО, анализ аналогичных систем, изучены методологии написания *web* - проектов.

В результате проведенной работы создано и протестировано клиент-серверное приложение, позволяющее покупателю искать товары, добавлять их в свою корзину и оформлять оплату, а администратору маркетплейса загружать товары, назначать им характеристики и редактировать описание товаров.

В пояснительной записке представлены: анализ предметной области, проектирование и реализация приложения, технология разработки и отладки программы, раздел БЖД.

Графическая часть содержит модели БД и *UML*-диаграммы приложени

Содержание

Перечень сокращений и обозначений 4
Введение 5
1 Web – портал маркетплейс - агрегатора 6
1.1 Предпроектный анализ 6
1.1.1 Анализ предметной области 6
1.1.2 Анализ аналогичных решений web-портала маркетплейс- агрегатора.8
1.1.3 Анализ стека технологий 10
1.1.4 Выбор СУБД 15
1.1.5 Выбор фреймворка для реализации клиентской части приложения.17
1.2 Проектирование и разработка 19
1.2.1 Реализация серверной части. 19
1.2.2 Разработка БД 22
1.2.3 Разработка клиентской части 30
1.3 Анализ результатов 33
1.3.1 Тестирование приложения 34
2 Охрана труда 39
2.1 Характеристики потенциальных опасностей и вредностей, которые могут возникнуть при работе с ЭВМ 40
Заключение 41
Список использованных источников 42
Приложение А – UML диаграммы 44
Приложение Б – Листинг программы 48
Приложение В - Презентация 93

Перечень сокращений и обозначений

В настоящей работе применяют следующие сокращения и обозначения

БД – база данных

СУБД – система управления базами данных

API – программный интерфейс приложения

CRUD – акроним, обозначающий четыре базовые функции, используемые при работе с базами данных: создание (англ. create), чтение (read), модификация (update), удаление (delete)

MVC – «Модель-Представление-Контроллер» или «Модель-Вид-Контроллер»

MVP – минимально жизнеспособный продукт

ORM – способ доступа к базе данных с помощью объектно – ориентированного языка

REST – архитектурный стиль взаимодействия компонентов распределенного приложения в сети

Введение

В современном мире электронная коммерция представляет собой одну из самых активных для бурного роста и развития сфер, перспективность которой на сегодняшний день сложно отрицать. Ежедневно миллионы людей предпочитают просматривать и оформлять онлайн сотни и тысячи товаров через различные интернет-магазины.

В России маркетплейсы показывают значительный рост. В 2022 году, несмотря на ожидаемый спад в силу уходов многих брендов, оборот данного сегмента интернет – торговли составил 2.7 трлн. рублей, увеличившись на 80%.

Вышеописанное демонстрируют нам актуальность выбранного в рамках выпускной квалификационной работы проекта.

Архитектурно данное приложение должно быть реализовано по классическом принципу клиент – серверное приложение, где клиентская часть представлена в виде *web* – интерфейса в браузере (фронтенд), а серверная часть непрерывно “слушает” запросы с клиентской и производит необходимые манипуляции с записью в базы данных (бэкенд).

Для реализации соответствующей модели были проведены определенные этапы разработки, по итогам которой демонстрировались результаты работы, а именно:

– исследована предметная область;

– исследованы аналоги данного приложения;

– исследован и выбран стек для написания каждой из частей приложения;

– разработана кодовая база;

– проведена отладка и тестирование приложения и соответствующих инструментов, на которых она разрабатывалось;

- описан раздел безопасности жизнедеятельности.

1 Web – портал маркетплейс - агрегатора

1.1 Предпроектный анализ

1.1.1 Анализ предметной области

Для того, чтобы успешно и в обозначенные сроки реализовать задачи, поставленные во введении данной выпускной квалификационной работы, необходимо провести анализ предметной области выбранной темы и представлять, из каких этапов будет состоять разработка данного проекта.

WEB - портал маркетплейс-агрегатора – это крайне большой по своему объему и содержанию ресурс, для проектирования которого необходима большая команда опытных инженеров, имеющих экспертизу в рамках работы с подобными по объему проектами и хорошо владеющих предметной частью, поэтому в условиях ограниченных ресурсов, а именно времени для проекта и количестве разработчиков, было принято решение предварительно создать *MVP* версию продукта.

MVP (*minium viable product*) – это один из современных подходов к созданию различных продуктов и инженерных решений, в том числе и продуктов программного обеспечения, состоящий в том, чтобы создать минимально жизнеспособную версию продукта, имеющую самый базовый набор функциональности и требующую небольшое время на разработку. Смысл данного решения состоит в том, чтобы сразу оценить потребность в таком продукте у широкой аудитории, не тратить слишком много времени на разработку и тестировании второстепенных гипотез, а также быстро выйти на рынок и начать получать какой-либо доход, необходимый хотя бы на дальнейшее поддержание ресурсов разработки, таких как сервер и т. д.

В случае неудачи и неуспеха, как уже было выше сказано, на такой подход было затрачено довольно мало ресурсов, что снижает риски его разработки, и даже иногда дает возможность использовать наработки в новых, уже иных проектах разработчика. Также подход *MVC* позволяет

иметь постоянную обратную связь с потребителем продукта, что позволяет итеративно наращивать необходимый для пользователя функционал и оперативно закрывать возникающие баги и проблемы. Наглядная демонстрация данного подхода представлена в иллюстрации на рисунке 1.1.

Рисунок 1.1 – Разница между стандартным принятым к разработке продуктов подходом и подходом *MVP*, выбранным в качестве основного для данного проекта

В рамках данного подхода для проекта *web* - портала маркетплейс-агрегатора было решено выделить две группы пользователей, покупателей и администратора, и дать им возможность приобретать и загружать товары на сайт соответственно.

1.1.2 Анализ аналогичных решений web-портала маркетплейс-агрегатора.

На сегодняшний день уже существует несколько популярных маркетплейсов - агрегаторов, которые мы можем проанализировать для того, чтобы понять, на какие основные моменты нам нужно опираться при разработке своего продукта, а также выделить их конкурентные преимущества и недостатки для конечного потребителя.

Первым для анализа рассмотрим один из самых популярных, если не самый, маркетплейсов в России *Wildberries*. На рисунке 1.2 представлен дизайн главной страницы данного маркетплейса.

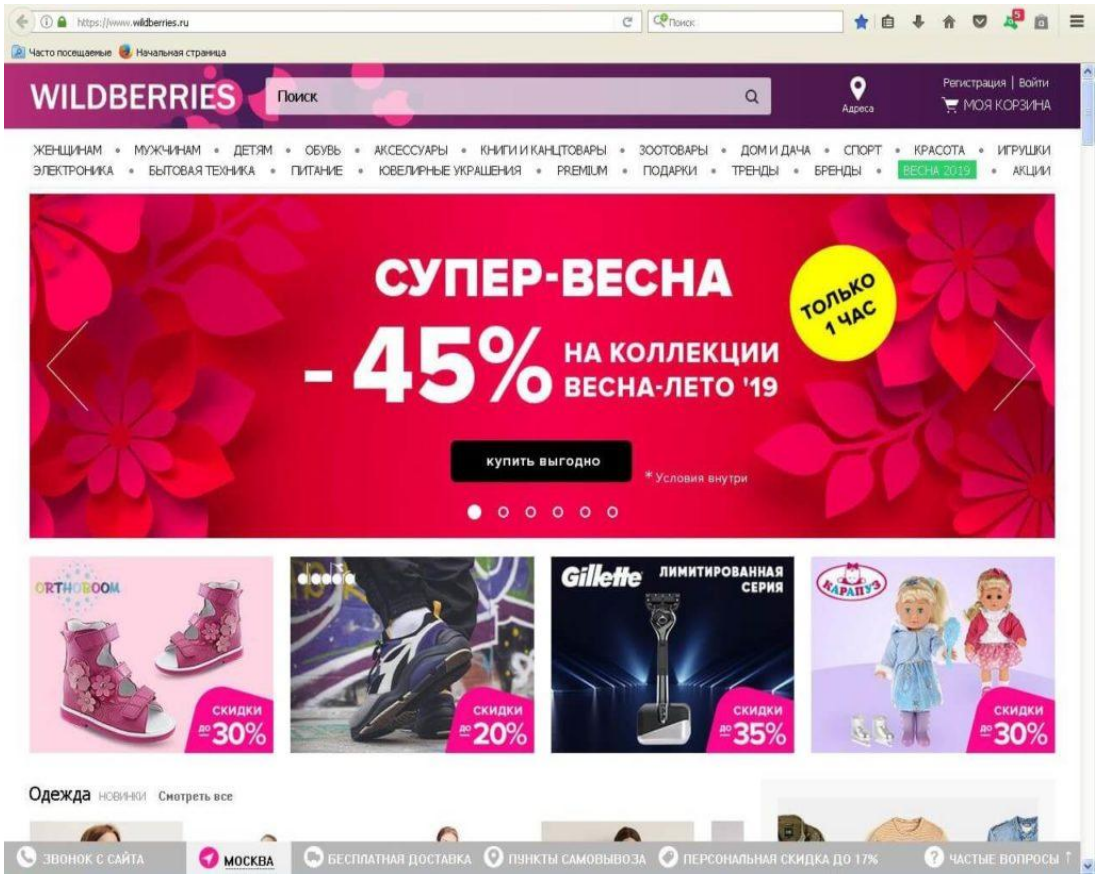


Рисунок 1.2 – Главная страница маркетплейса *Wildberries*

Wildberries был основан в 2004 году российской предпринимательницей Татьяной Бакальчукой и насчитывает уже довольно большую и богатую историю.

К основным достоинствам этого магазина можно отнести огромный выбор товаров в самых различных категориях, от бытовых принадлежностей до велосипедов и моторных лодок, конкурентные цены, при этом часто сопровождающиеся большими сезонными скидками, а также быструю и удобную доставку товаров по всей России.

К недостаткам же мы можем отнести большую комиссию площадки за возможность размещаться на ней, что повышает “порог входа” для небольших магазинов, а также порой долгое ожидание подтверждения заказа продавцом на площадке из-за большого числа клиентов.

Еще одним очень популярным среди потребителей в сфере *e* - commerce торговли в России является маркетплейс - агрегатор *Ozon*, появившийся даже раньше, чем *Wildberries*, в далеком 1998 году. На рисунке 1.3 представлена главная страница данного маркетплейса.

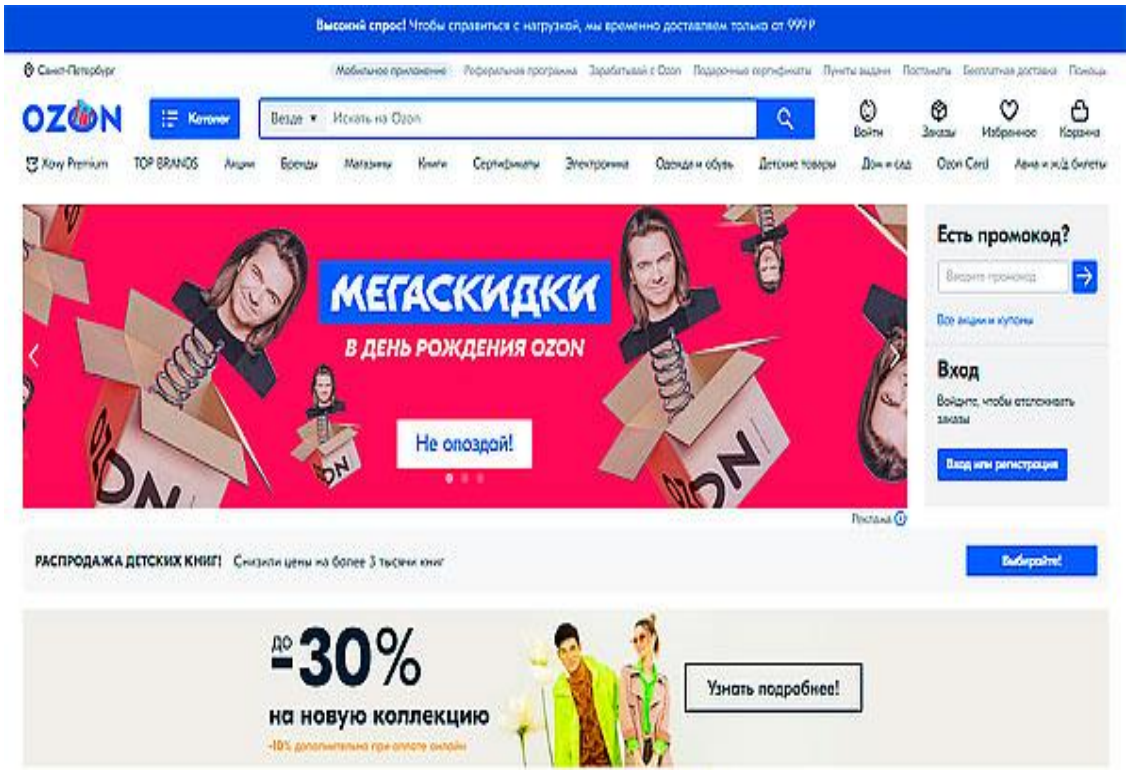


Рисунок 1.3 – Главная страница маркетплейса *Ozon*

К его преимуществам можно отнести приятный дизайн, то, что площадка работает с прямыми поставщиками, поэтому на нем в большем

количестве представлены небольшие аутентичные магазины, а также большое количество рекламных предложений для привлечения новых покупателей и продавцов, что обеспечивает наличие большого числа различных бонусных программ.

Недостатки частично схожи с уже упомянутым выше *Wildberries*: высокая цена за комиссию за размещение (порой может доходить до 40%), что ограничивает на нем размещение производств до 1000 товаров в силу нерентабельности, а также необходимость продавцов быть юридическим лицом.

1.1.3 Анализ стека технологий

На текущий момент есть широкий выбор языков программирования, как правило высокоуровневых, но бывают и исключения, для написания клиент-серверных программ: *Java, C#, Python, C++* и др.

Архитектура *Web* - портала состоит из трех частей, и этим определяется то, какой стек технологий будет выбран для данного проекта. Основные части архитектуры:

– клиентская часть;

– серверная часть;

– база данных.

Рассмотрим основные языки, которые могут быть использованы в качестве инструмента для разработки серверной части предложения.

На сегодняшний день, учитывая огромную популярность веб-разработки, существует довольно много высокоуровневых языков программирования, на которых можно написать *web* - портал, однако помимо самого языка нужно также учитывать, сложилась ли вокруг него удобная экосистема, позволяющая наиболее эффективно и быстро писать программный код, что вынуждает нас ответственно подойти к выборам инструментов разработки, причем речь не только о языке программирования,

но также о фреймворке для данного языка, который уже готов выполнять большую часть работы за нас.

Web-портал можно реализовать на таких современных языка программирования, как *Java, Python, C#, Go*. Ниже мы рассмотрим данные языки программирования и выберем наиболее подходящий для наших задач инструмент.

Java – высокоуровневый объектно – ориентированный язык программирования, являющийся сегодня одним из самых популярных в мире языков. Язык появился еще в 1996 году и проделал большой путь, что породило огромную систему библиотек вокруг него, набор фреймворков для различных видов разработки, определенную экосистему и т. д. В настоящее время текущей версией языка является *Java* 20, вышедшая в марте 2023 года. Ключевой особенностью языка является наличие виртуальной машины *JVM* (*Java Virtual Machine*), что позволяет транслировать код данного языка сначала в специальный байт – код, независимый от платформы, который затем будет выполняться данной виртуальной машиной, что снимает с программиста задачу управления памятью и позволяет ему сконцентрироваться сугубо на бизнес – логике. Подобная архитектура языка обеспечивает кроссплатформенность и потокобезопасность.

Python – высокоуровневый объектно-ориентированный, который предназначен для приложений различного типа. Язык является интерпретируемым, то есть не компилируется напрямую в машинный код, а исполняется построчно напрямую на машине пользователя построчно или блочно встроенным интерпретатором. Это делает язык довольно медленным, что затрудняет его применение, например, в сфере разработки компьютерных игр, где необходимо выжимать максимум производительности из устройства, однако язык нашел применение в таких сферах как *Data science* и *Machine Learning* за счет обширных библиотек, например, *NumPy* и *Pandas*, а также в сфере разработки веб-приложений за счет возможности быстро написать

минимальный прототип приложения используя фреймоврки *Django* и *Flask*, например.

C# – высокоуровневый язык программирования от компании Microsoft. К его основным преимуществам можно отнести платформенную независимость (язык подходит для создания приложений на любой платформе, поддерживающей *CLR – Common Language Runtime*, а именно *Windows, Linux* и *MacOs*), типобезопасность за счет проверки типов в момент компиляции, а также обширную библиотеку классов *.NET Framework и .NET Core*, которая часто используется для создания приложений и веб – сервисов.

Go – язык программированый, представленный компанией Google. Крайне быстрый язык с си-подобным синтаксисом, позволяющий писать легкие и быстрые многопоточные программы. Несмотря на свой относительно молодой возраст и лаконичность, *Golang* уже оброс большим количеством библиотек, преимущественно направленных на его использование в сетевой и *web* – разработке, что позволяет легко и быстро писать на нем подобного рода приложения. К недостаткам языка можно отнести отсутствие ярко выраженной объектно – ориентированной парадигмы, а также лаконичность синтаксиса ведет к тому, что разработчикам часто приходится реализовывать набор функций, представленных в других языках по умолчанию, вручную.

Беря во внимание все вышеперечисленные преимущества и недостатки каждого из языков, было решено выбрать язык *Java* для работы над проектом.

Для этого язык существует ряд фреймворков для веб-разработки, но общепринятым стандартом в индустрии стал фреймворк *Spring* и возникшая вокруг него экосистема.

Spring – это легковесный и мощный опен соурс фреймворк для разработки веб – приложений. Он был создан в 2004 году и быстро стал использоваться в enterprise – разработке (вид разработки, создающий

комплексные, масштабируемые системы для крупных организаций и большого числа пользователей, например, банковские приложения, приложения для заказа такси по всему миру и т. д.). *Spring* был призван сделать подобную разработку более управляемой, а также решить ряд встающих перед ней типовых задач, например, быстрое развертывание веб – сервера, уменьшение жесткой связности многих различных компонентов в системе, удобное разделение веб – приложения на отдельные слои, такие как слой контроллера, бизнес –логики и доступа к базе данных. *Spring* позволяет создавать масштабируемые, отказоустойчивые, безопасные и высокопроизводительные приложения, которые могут легко обращаться по *API* к сторонним приложениям, а также использовать различные библиотеки.

Для того, чтобы эффективно использовать *Spring*, необходимо понимать основные концепции, которые закладывались его создателями в этот язык. *Spring* состоит из трех наиболее важных частей: инверсия управления (*IoC*) и внедрение зависимостей. Принцип инверсии управления означает, что Spring сам управляет компонентами приложения, используя принципе внедрения зависимостей и уменьшая количество кода, необходимого для написания.

Также помимо непосредственно *Spring Core*, вокруг *Spring* сложилась определенная экосистема, позволяющая эффективно деплоить приложения на облачные сервера, обращаться к базе данных и следовать паттерну *MVC* при разработке веб – приложений. Рассмотрим некоторые из них.

Spring MVC - модуль, исходя из названия, позволяет создавать web – приложение с использованием архитектуры *MVC* и стандарту *REST* – современному стандарту для обращения к ресурсам *web* – приложения через его *API*.

Spring Data – модуль позволяет легко обращаться к базе данных и манипулировать лежащими там записями, используя такие технологии как *JPA*, *JDBC* и даже *NoSQL*.

Spring Security – модуль обеспечивает защиту приложения через надежные механизмы аутентификации и авторизации, позволяя использовать защиту основываясь на статусе пользователя приложения, систему входа/выхода и т. д.

Spring Batch – модуль позволяет активно работать с огромными потоками данных через механизм батчей –обработки данных не целиком(то есть разбивая большой пласт данных на маленькие кусочки и обрабатывая каждую транзакцию отдельно), а путем деления данных на пакеты или группы, что облегчает их отработку и при этом позволяет делать это достаточно быстро.

Spring Cloud - данный модуль позволяет удобно использовать облачные сервисы и инфраструктуру, например, *AWS* или *Google Clouds*.

Spring Boot – фреймворк, позволяющий легко и эффективно создавать, настраивать и запускать проекты на фреймворке Spring, берущий большую часть работы в этих аспектах на себя.

Рассмотрим кратко принцип *MVC*, о котором шла речь выше.

MVC (*Model – View – Controller*) – паттерн для разработки *web* – приложений, разделяющих их на три архитектурных слоя: слой модели, слой представления и слой контроллера. Каждый слой имеет четко определенный круг задач и зону ответственности.

Слой модели - это компонент для представления слоя работы с данными приложениями, который необходим для операции с ними, их валидации и обработки. Основная задача этого слоя – работа с базой данных.

Слой представления – данный компонент необходим для того, чтобы отображать итог работы с данными в удобном и понятном для юзера виде. Данный слой взаимодействует с юзером и создает для него удобный интерфейс в виде форм, таблиц, описаний и т. д.

Контроллер – компонент, который представляет собой промежуточную связь между слоями модели и представления. Он принимает запросы от

пользователя, обрабатывает их в соответствии с этими запросами обновляет компоненты модели и представления.

Преимущества использования *MVC*: четко виден уровень ответственности каждого компонента, возможность переиспользования кода, гибкость в разработке путем уменьшения связности частей приложения.
Схема архитектуры *MVC* представлена на рисунке 1.4.

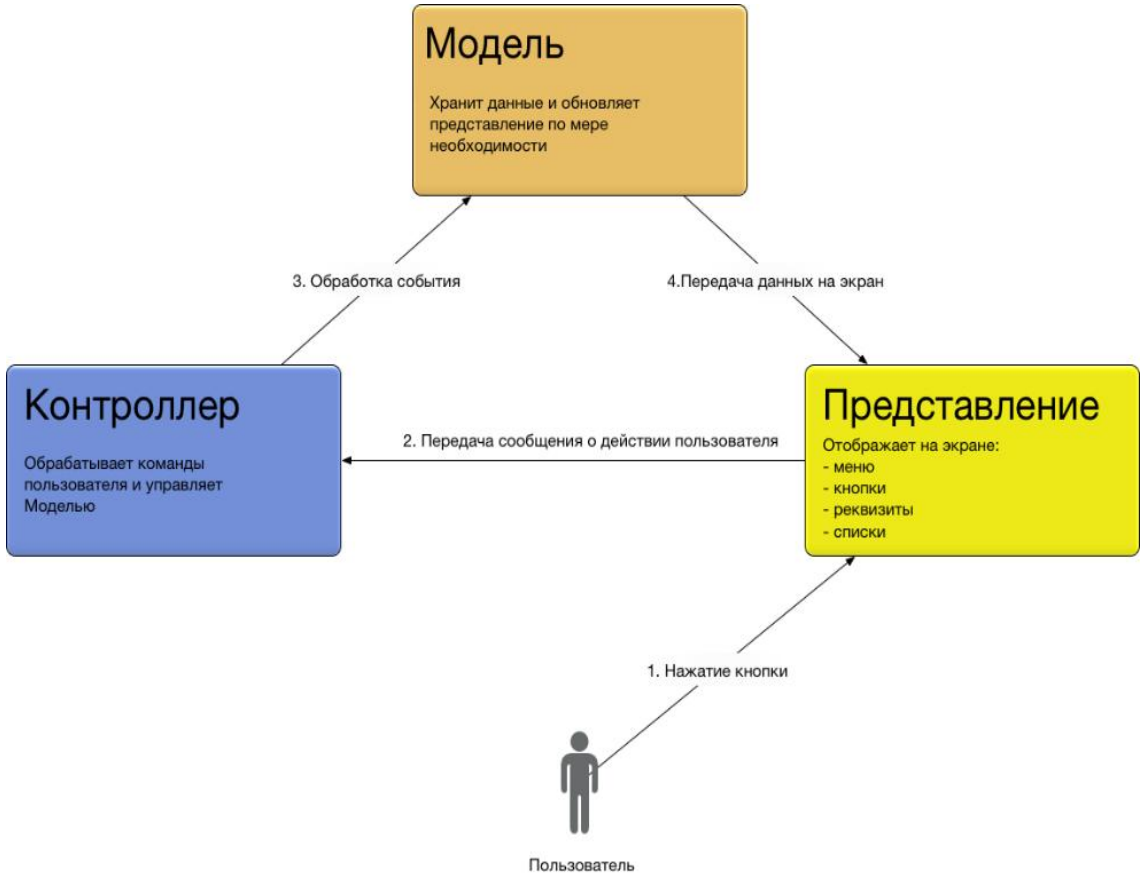


Рисунок 1.4 – Схема архитектуры *MVC*

1.1.4 Выбор СУБД

Для проектирования базы данных *web* - портала необходимо выбрать СУБД. Основные критерии для выбора СУБД для данного портала: подробная документация, легкость развертывания, отказоустойчивость и стабильность. Рассмотрим наиболее популярные на сегодняшний день СУБД и оценим, какая из них наиболее подходит для выбранного проекта.

PostgreSQL – это современная мощная опен – соурс реляционная база данных которая используется для хранения, управления и извлечения данных. *PostgreSQL* использует клиент – серверную модель для для хранения и обработки данных. Чтобы работать с ней, чаще всего программист должен указать номер хоста, на котором БД расположена, порт, а также логин и пароль. Она поддерживает большой набор языков программирования, таких как *Python, C/C++*, *Ruby*, *PHP* и многие другие, в том числе и *Java*, который будет использоваться для работы над проектом, что делает ее прекрасным выбором для поставленных задач. *PostgreSQL* обладает большим набором функций, включая поддержку *JSON*, полнотекстовой поиск, поддержку пользовательских типов данных и расширенную индексацию. Также она поддерживает хранимые процедуры и триггеры.

Oracle – это СУБД от одноименной компании *Oracle*, является корпоративным стандартном в индустрии. База данных изначально проектировалась для эффективного хранения и обработки большого количества данных, включая структурированные, частично – структурированные и неструктурированные данные. СУБД управляется собственным диалектом языка *SQL*, предназначенным для эффективной работы с большими нагрузками, что позволяет ей поддерживать более миллиона транзакций ежедневно.

SQL Server – данная СУБД разработана компанией *Microsoft.* Ее отличительной особенностью является поддержка специального диалекта *SQL* под названием *T –SQL*. В зависимости от того, какую задача выполняет команда на данном языке, их подразделяют на следующие типы: *DDL* для определения (т.е. создания) данных, *DML* для управления данными и *DCL* для управления правами по доступам к данным. Также данная СУБД отличается производительностью, надежностью и простотой.

На основе вышеизложенного, в качестве СУБД для проекта было решено выбрать *PostgreSQL*.

1.1.5 Выбор фреймворка для реализации клиентской части приложения.

На сегодняшний день для выбора инструментов создания клиентской части приложения существует не так много вариантов. Не считая неких экзотических вариантов, таких как, например, фреймворк для *Java Vaadin*, с помощью которого на языке *Java* можно создавать клиентский интерфейс в браузере, существует по сути всего три инструмента для оформления внешнего вида страница: *HTML*, *CSS* и *Javascript*.

HTML представляет из себя язык разметки, используемый в веб – разработке для создания структурированного содержимого страниц в браузере. Он использует стандартный набор тегов, с помощью которых можно задать, какие элементы (например, картинка или абзац) должны быть размещены на странице в том или ином месте.

CSS необходим нам для оформления созданных с помощью *HTML* тегов страниц. Этот язык с помощью специальной структуры позволяет задавать набор оформлений для страниц, таких как, например, размер шрифта, цвет текста и т. д.

После того, как мы сделали разметку для страницы и украсили ее с помощью *CSS*, перед нами может встать необходимость сделать страницу интерактивной, для того, чтобы реагировать на те или иные действия пользователей. Для решения этой проблемы существует язык *Javascript*, который позволяет изменять элементы в соответствии с действиями пользователя. На сегодняшний день никто не использует данный язык в его “чистом” виде, существует набор фреймворков, которые берут на себя часть работы и позволяют делать веб – страницы клиентской части более гибкими и динамичными, разделяя их на компоненты. По сути выбор инструментов для клиентской части и состоит в выборе одного из фреймворков для языка *Javascript*. Рассмотрим некоторые из этих фреймворков.

React – фреймворк, разработанный компанией *Facebook*. Он быстро стал популярен среди программистов клиентской части приложений, поскольку сочетает в себе высокую производительность с удобным подходом к разработке веб – приложений. Модульная архитектура *React* позволяет разбивать приложение на много маленьких компонентов, что делает его более гибким и удобным при разработке. Еще удобным *React* делает то, что он работает с виртуальным, а не реальным *DOM* - деревом. Это позволяет не перерисовывать каждый раз всю страницу (т.е. все ее *DOM* – дерево), а перерисовывать только определенную конкретную часть.

AngularJS – фреймворк, созданный компанией *Google*, использующийся также для создания динамических пользовательских интерфейсов. Одним из наиболее важных преимуществ *Angular* является двухстороннее связывание (*two – way data binding*), означающее, что любые изменения, внесенные пользователем в пользовательский интерфейс, автоматически отображаются в модели данных, и наоборот. *AngularJS* также поддерживает создание пользовательских директив, что позволяет разработчикам создавать собственные элементы управления, приспособленные к их конкретным нуждам, что облегчает повторное использование кода.

Vue.js – это фреймворк, который основывается на практике объявления шаблона – это означает, что можно создать пользовательский интерфейс путем объявление шаблона внутри *html* – файла. Одной из главных особенностей *Vue.js* является "реактивность". Это означает, что *Vue.js* автоматически обновляет содержимое страницы при изменении данных, что делает код более понятным и легким для понимания.

Для создания клиентской части приложения было решено выбрать *AngularJS*.

1.2 Проектирование и разработка

Приложение, которое являются целью разработки данной выпускной квалификационной работы, состоит из клиентской и серверной частей. Сервер будет предоставлять *API* для обращений со стороны клиентской части, и, реагируя на поступающие запросы, будет работать с моделями, хранящимися в базе данных. Клиентская часть будет представлять из себя стандартный *UI*, написанный на языке *Angular*, и будет служить для того, чтобы пользователи приложения могли осуществлять необходимые им действия по покупке или продаже товаров соответственно. Клиентская часть должна представлять из себя удобный интуитивно – понятный интерфейс.

Так как на сайте будет две группы пользователей, обсудим каждую из них.

Покупатель будет способен осуществлять поиск по сайту, выбирать понравившиеся товары, добавлять их в корзину и оплачивать. Администратор сайта будет способен выкладывать на сайт новые товары, указывая необходимые товарные характеристики, осуществлять поиск по товарам сайта, редактировать добавленные товары, а также менять статус доставки товара, заказанного пользователем.

Перед запуском сервера происходит процесс инициализации всех модулей и компонентов, который включает в себя внедрение зависимостей и установление соединения с базой данных.

1.2.1 Реализация серверной части.

Серверная часть приложения была разработана с использование фреймворка *Spring*. Для подключения сторонних библиотеке и сборки приложения был использован фреймворк *Maven*.

Сначала была произведена предварительная конфигурация приложения с использованием сервиса *Spring Initializer*. В данном сервисе необходимо выбрать основные части фреймворка *Spring*, которые будут использоваться в приложении, а также иные зависимости, такие как, например, драйвер

PostgreSQL для взаимодействия с базой данных. Затем была произведена настройка приложения для первого запуска через файл *application.properties*. Результаты настройки приложения продемонстрированы на рисунке 1.5.

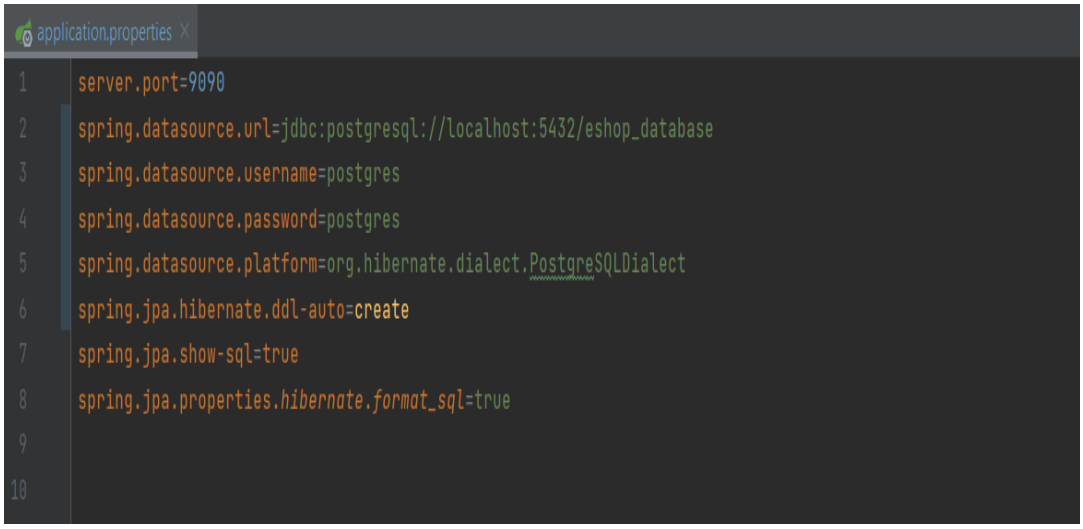


Рисунок 1.5 – Результат конфигурации приложения

Затем были созданы контроллеры и в них прописаны соответствующие эндопониты для основных сущностей в приложении, таких как *Cart*, *Product*, *Order Detail* и т. д. Пример такого эндпоинта в классе контроллер представлен на рисунке 1.6.

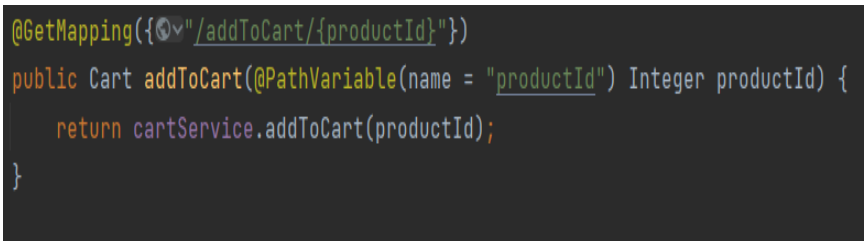


Рисунок 1.6 – Пример эндпоинта, обрабатывающего добавление товара в корзину

Для того, чтобы доступ к данному и иным методам имели только предварительно авторизованные юзеры, то есть, проще говоря, в целях обеспечения безопасности, была добавлена аннотация *@PreAuthorize(“hasRole(‘User’))* из пакета *Spring Security*, проверяющая,

является ли клиент пользователем приложения с помощью метода *.hasRole*(). Пример использования данной аннотации представлен на рисунке 1.7.

```
@PreAuthorize("hasRole('User')")
@GetMapping({"/getCartDetails"})
public List<Cart> getCartDetails() {
    return cartService.getCartDetails();
}
```

Рисунок 1.7 – Пример использования аннотации *@PreAuthorize*

В целом, классы контроллера опираются на созданную структуру базы данных и необходимы для обращения к соответствующим сущностям. На рисунке 1.8 представлены *UML* – диаграмма контроллеров.

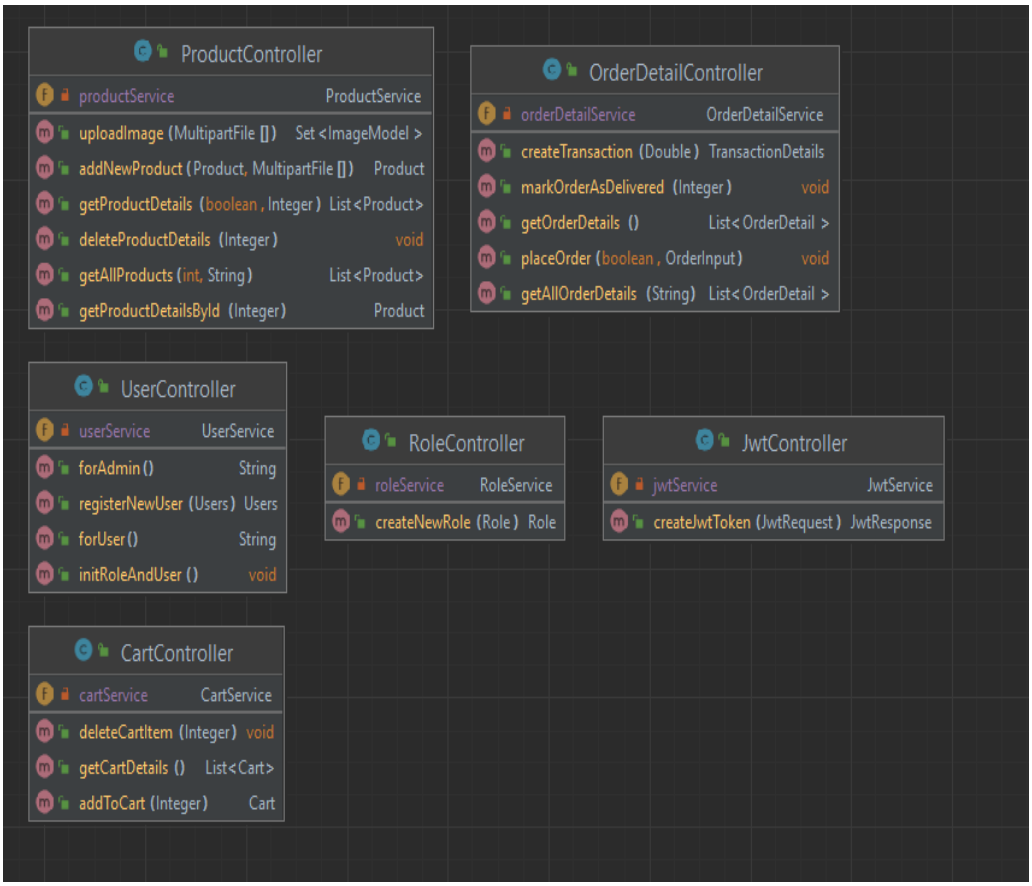


Рисунок 1.8 – *UML* диаграмма классов контроллеров

Контроллеры не взаимодействую напрямую с базой данных и классами моделей, а используют для этого прослойку сервисов, в которых реализована

основная бизнес – логика в виде соответствующих методов. Основные поля и методы классов сервисов представлены на рисунке 1.9.

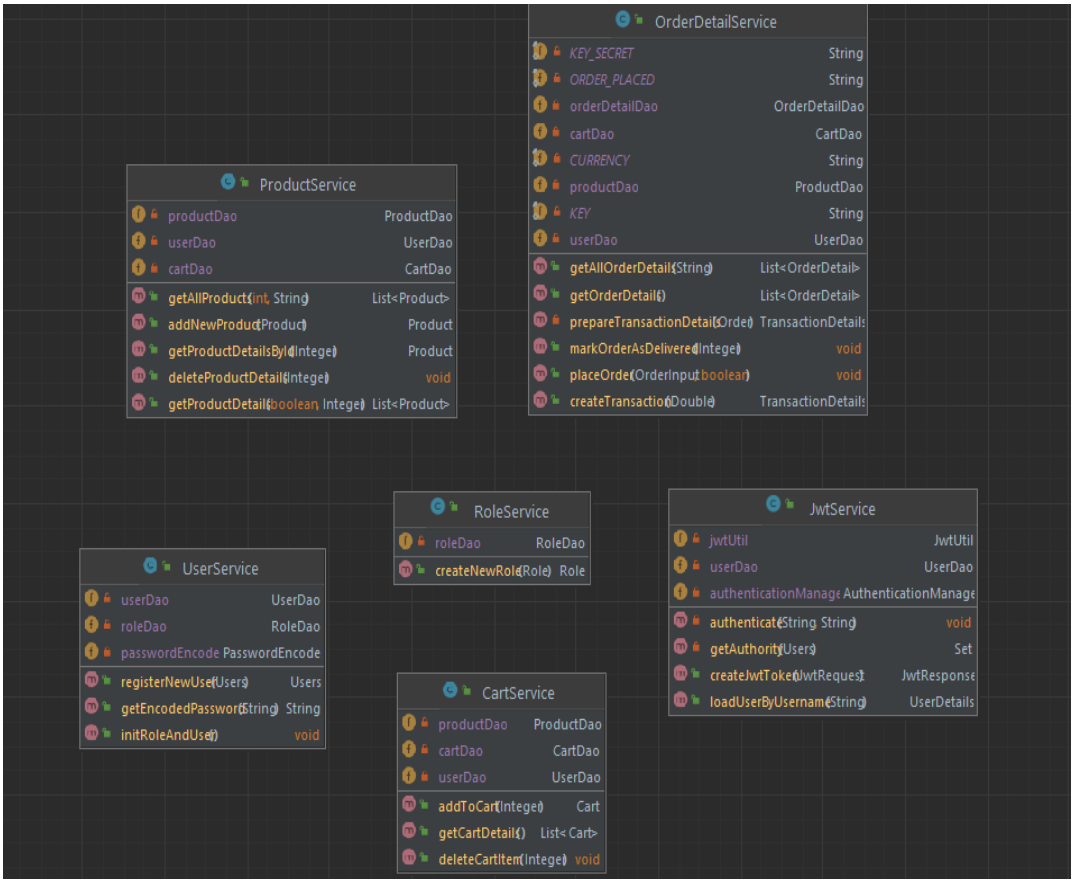


Рисунок 1.9 – *UML* диаграмма классов сервисов

1.2.2 Разработка БД

Spring позволяет не писать напрямую запросы к базе данных на языке *SQL*, а работать с объектами программы как с записями в базе данных. Для этого используется прослойка в виде *Spring Data JPA* и *Hibernate*. *Spring Data JPA* предоставляет высокоуровневые абстракции к базе данных, позволяя, например, искать нужные данные с помощью вызова соответствующего метода у объекта соответствующего класса и сокращая количество *boilerplate* кода, а *Hibernate* выполняет низкоуровневую работу, превращая эти методы в конкретные *SQL* запросы.

Для того, чтобы использовать *Hibernate* и *Spring Data JPA*, нам необходимо сначала подключить последнюю в *Maven*. Отдельно *Hibernate*

подключить не нужно, поскольку он является часть *Spring Data JPA*. На рисунке 1.10 показано соответствующее подключение к проекту.

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>
```

Рис 1.10 – Подключаем к проекту *Spring-data-jpa*

Для того, что описать базу данных в самом приложении, а использовать механизм *ORM* (соотносит сущности в приложении с таблицами в базе данных, тем самым позволяя легко манипулировать записями как будто это объекты в программе), нам необходимо сначала выделять основные сущности и их поля, на которых и будет строиться база данных в нашем приложении. Мы можем явно выделить три основных сущностей: пользователи и роли, продукты и их изображения, заказы.

Разделы пользователей и ролей представлен двумя сущностями (таблица 1.1 и таблица 1.2): “Пользователи”, “Роли”.

Таблица 1.1 – Атрибуты сущности “Пользователи”

№	Наименование атрибута	Тип	Обязательный
1	Идентификатор пользователя	Целый	Да
2	Юзернейм пользователя	Строка	Да
3	Имя пользователя	Строка	Да
4	Фамилия пользователя	Строка	Да
5	Пользователь пароля	Строка	Да

Таблица 1.2 – Атрибуты сущности “Роли”

№	Наименование атрибута	Тип	Обязательный
1	Идентификатор роли	Целый	Да
2	Название роли	Строка	Да
3	Описание пользователя	Строка	Да

Следующий описываемый раздел – “Продукты и их изображения”. Описание полей моделей данного раздела представлены ниже в таблицах 1.3 и 1.4.

Таблица 1.3 – Атрибуты сущности “Продукты”

№	Наименование атрибута	Тип	Обязательный
1	Идентификатор товара	Целый	Да
2	Актуальная цена	Числовое значение	Да
3	Описание товара	Строка	Да
4	Цена со скидкой	Числовое значение	Да
5	Название товара	Строка	Да

Таблица 1.4 – Атрибуты сущности “Изображения”

№	Наименование атрибута	Тип	Обязательный
1	Идентификатор изображения	Целый	Да
2	Название изображения	Строка	Да
3	Двоичной код изображения	Бинарный	Да
4	Формат изображения	Строка	Да

После того, как пользователем будет выбран товар на сайте, следующий раздел, к которому он перейдет, будет раздел заказа товара. С данными разделом связана сущность “Детали заказа”. Поля данной сущности описаны в таблице 1.5

Таблица 1.5 – Атрибуты сущности “Детали заказа”

№	Наименование атрибута	Тип	Обязательный
1	Идентификатор заказа	Целый	Да
2	Номер заказа	Строка	Да
3	Количество товаров в заказе	Целое	Да
4	Дополнительный номер заказ	Строка	Да
5	Полное название заказа	Строка	Да
6	Статус заказа	Строка	Да
7	Идентификатор продукта		Внешний ключ
8	Имя пользователя		Внешний ключ

После того, как описаны основные сущности, на их основе мы можем понять, из каких фактических таблиц будет реализована физическая модель БД.

Связь таблиц пользователей и их ролей представлена через таблицу “user_role”. Логические модули базы данных для данных сущностей продемонстрированы в таблицах 1.6, 1.7 и 1.8.

Таблица 1.6 – Атрибуты сущности “users”

№	Наименование атрибута	Тип	Обязательный
1	id	Int	Первичный ключ, обязательный
2	user_name	varchar(128)	Да
3	user_first_name	varchar(128)	Да
4	user_last_name	varchar(128)	Да
5	user_password	varchar(255)	Да

Таблица 1.7 – Атрибуты сущности “role”

№	Наименование атрибута	Тип	Обязательный
1	id	int	Первичный ключ, обязательный
2	role_name	varchar(255)	Да
3	role_description	varchar(255)	Да

Таблица 1.8 – Атрибуты сущности “user_role”

№	Наименование атрибута	Тип	Обязательный
1	id	int	Первичный ключ, обязательный
2	user_id		Внешний ключ
3	role_id		Внешний ключ

Далее рассмотрим представление в базе данных в виде таблиц товаров, изображений и связь между ними. Их поля представлены в таблицах 1.9, 1.10 и 1.11.

Таблица 1.9 – Атрибуты сущности “product”

№	Наименование атрибута	Тип	Обязательный
1	product_id	int	Первичный ключ, обязательный
2	product_actual_price	double	Да
3	product_description	varchar(200)	Да
4	product_discounted_price	double	Да
5	product_name	varchar(255)	Да

Таблица 1.10 – Атрибуты сущности “image_model”

№	Наименование атрибута	Тип	Обязательный
1	id	int	Первичный ключ, обязательный
2	name	varchar(255)	Да
3	pic_byte	bytea	Да
4	type	varchar(255)	Да

Таблица 1.11 – Атрибуты сущности “product_images ”

№	Наименование атрибута	Тип	Обязательный
1	product_id		Внешний ключ
2	image_id		Внешний ключ

Последнюю таблицу, которую мы рассмотрим, будет таблица “order_detail”, хранящая информацию о заказе, и таблица “cart”, связывающая ее с информацией о пользователях. Рассмотрим их поля в таблицах 1.12 и 1.13.

Таблица 1.12 – Атрибуты сущности “order_detail ”

№	Наименование атрибута	Тип	Обязательный
1	order_id	int	Первичный ключ, обязательный
2	order_alternate_contact_number	varchar(255)	Да
3	order_amount	int	Да
4	order_contact_number	varchar(255)	Да
5	order_full_name	varchar(255)	Да
6	order_full_order	varchar(255)	Да

Таблица 1.13 – Атрибуты сущности “cart ”

№	Наименование атрибута	Тип	Обязательный
1	cart_id		Внешний ключ
2	product_product_id		Внешний ключ
3	users_user_name		Внешний ключ

Нам не обязательно вручную создавать все эти таблицы. Эту задачу можно решить помощью добавления аннотации *@Entity* к классам моделей, *Spring* автоматически создаст соответствующие таблицы в базе данных. Можно также с помощью дополнительных аннотаций, например, таких как *@OneToOne* более детально производить настройку классов и, соответственно, таблиц в базе данных. Например, уже упомянутая выше *@OneToOne* позволяет создавать связи между таблицами используя *Primary Key* и *Foreign Key* соответственно. Пример использования аннотации продемонстрирован на рисунке 1.11.

```
@Entity
public class OrderDetail {

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private Integer orderId;
```

Рисунок 1.11 – Пример использования аннотации *@Entity* для создания таблицы из класса в базе данных

Таким образом, мы создаем необходимые таблицы в базе, но для того, чтобы к ним обращаться, нам необходим слой репозитория, через который мы непосредственно будем этими записями манипулировать. Для этого мы создаем соответствующие классы, каждый из которых наследуем от класса *CrudRepository*, который, исходя из названия, уже содержит соответствующий набор методов для манипулирования данными. Мы также можем расширять эти методы, просто указываем необходимые нам методы в

сигнатуре этого класса. Слой репозитория также по-другому называется слоем *DAO – data access object*, то есть слой доступа к данным, однако никакой разницы по – большому счету нет. На рисунке 1.12 приведен пример создания класса репозитория для корзины с собственным методом *findByUser (User user)*.

```
@Repository
public interface CartDao extends CrudRepository<Cart, Integer > {
    4 usages  new *
    public List<Cart> findByUsers(Users users);
}
```

Рисунок 1.12 – Пример класса репозитория для доступа к таблице "*Cart*"

Таким образом, используя данные абстракции, мы и манипулируем записями в базе данных. На рисунках 13 и 14 представлены *UML*-диаграммы сущностей и репозиториев для взаимодействия с ними.

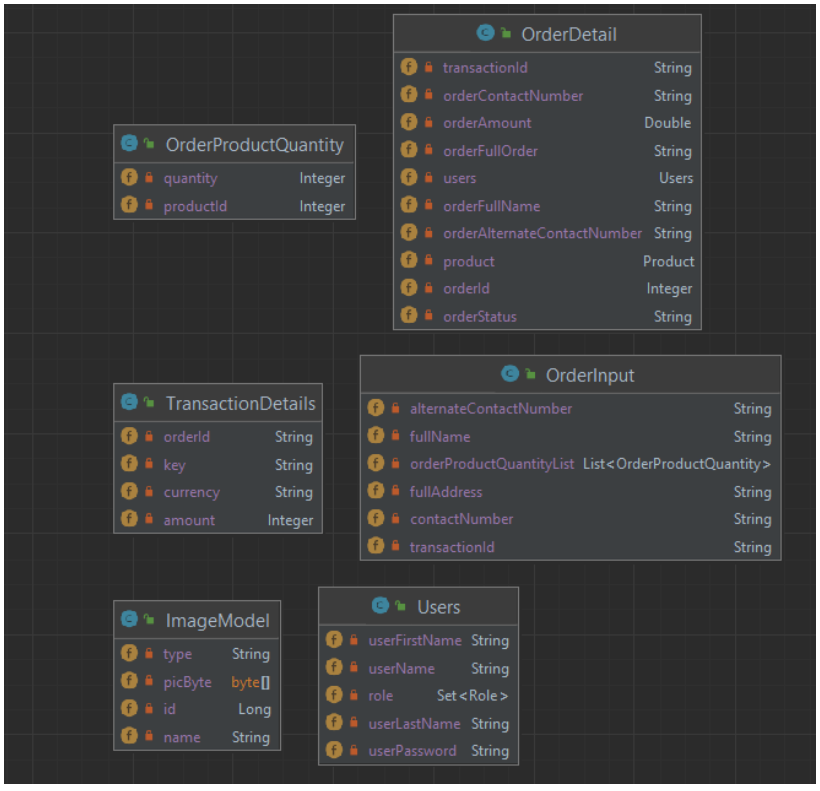


Рисунок 1.13 – *UML* – диаграммы сущностей в пакете "*Entity*"

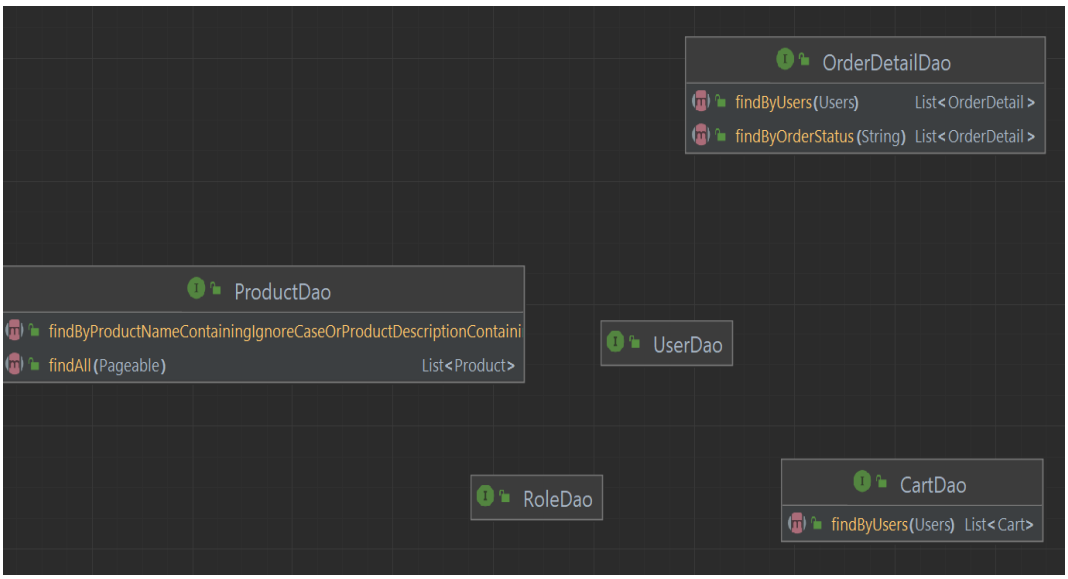


Рисунок 1.14 – *UML* – диаграммы классов *DAO* для доступа к базе данных

1.2.3 Разработка клиентской части

В качестве клиентской части приложения используется фреймворк *Angular*. Первоначально был создан сам проект с помощью команды для *Angular CLI ng new E-Commerce*. Затем в файле *angular.json* была произведена настройка этого приложения. Сам файл конфигурации представлен на рисунке 1.15.

```
{
  "$schema": "./node_modules/@angular/cli/lib/config/schema.json",
  "version": 1,
  "newProjectRoot": "projects",
  "projects": {
    "jwt-youtube-ui": {
      "projectType": "application",
      "schematics": {},
      "root": "",
      "sourceRoot": "src",
      "prefix": "app",
      "architect": {
        "build": {
          "builder": "@angular-devkit/build-angular:browser",
          "options": {
            "outputPath": "dist/jwt-youtube-ui",
            "index": "src/index.html",
            "main": "src/main.ts",
            "polyfills": "src/polyfills.ts",
            "tsConfig": "tsconfig.app.json",
            "aot": true,
            "assets": [
              "src/favicon.ico",
              "src/assets"
```

Рисунок 1.15 – Файл конфигурации *angular.json*

После этого были созданы компоненты приложения, каждый компонент отвечает за соответствующие часть страницы. Для компонентов были указаны свойства, прописан конструктор по умолчанию, а также определены основные методы, отображаемые на клиентской части. Пример такого компонента – корзина – представлен на рисунке 1.16. На примере данного компонента можно также упомянуть, что каждый компонент связывается непосредственно с *html* – шаблоном с помощью свойства templateUrl и также, соответственно, с *css* – кодом с помощью свойства styleUrls, которые во многом и задают внешний вид страны, а сам компонент служит для реализации ее логики работы и разделения клиентской части на отдельные модули, которые проще поддерживать. Язык *AngularJS* придерживается парадигмы написания кода в объектно – ориентированном стиле, что позволяет использовать в языке классы, наследование, конструкторы классов и прочие элементы данной парадигмы.

```
import { Component, OnInit } from '@angular/core';
import { Router } from '@angular/router';
import { ProductService } from '../_services/product.service';

@Component({
  selector: 'app-cart',
  templateUrl: './cart.component.html',
  styleUrls: ['./cart.component.css']
})
export class CartComponent implements OnInit {

  displayedColumns: string[] = ['Name', 'Description', 'Price', 'Discounted Price', 'Action'];

  cartDetails: any[] = [];

  constructor(private productService: ProductService,
    private router: Router) { }

  ngOnInit(): void {
    this.getCartDetails();
  }

  delete(cartId) {
    console.log(cartId);
```

Рисунок 1.16 – Пример компонента *cart.component.ts*

После того, как все основные компоненты были созданы, в модуле app-*routing.module.ts* были определены маршруты, связывающие эти компоненты с конкретными путями на странице сайта. Код в соответствующем файле изображен на рисунке 1.17. После запуска клиентской части приложения командой ng server, она стала доступна по адресу *http://localhost:4200/* и успешно была проверена.

```
import { NgModule } from "@angular/core";
import { Routes, RouterModule } from "@angular/router";
import { AddNewProductComponent } from "./add-new-product/add-new-product.component";
import { AdminComponent } from "./admin/admin.component";
import { BuyProductResolverService } from "./buy-product-resolver.service";
import { BuyProductComponent } from "./buy-product/buy-product.component";
import { CartComponent } from "./cart/cart.component";
import { ForbiddenComponent } from "./forbidden/forbidden.component";
import { HomeComponent } from "./home/home.component";
import { LoginComponent } from "./login/login.component";
import { MyOrdersComponent } from "./my-orders/my-orders.component";
import { OrderConfirmationComponent } from "./order-confirmation/order-confirmation.component";
import { OrderDetailsComponent } from "./order-details/order-details.component";
import { ProductResolveService } from "./product-resolve.service";
import { ProductViewDetailsComponent } from "./product-view-details/product-view-details.component";
import { RegisterComponent } from "./register/register.component";
import { ShowProductDetailsComponent } from "./show-product-details/show-product-details.component";
import { UserComponent } from "./user/user.component";
import { AuthGuard } from "./_auth/auth.guard";

const routes: Routes = [
  { path: "", component: HomeComponent },
  {
    path: "admin",
    component: AdminComponent,
    canActivate: [AuthGuard],
    data: { roles: ["Admin"] },
  },
  {
    path: "user",
```

Рисунок 1.17 – Файл *app-routing.module.ts*

1.3 Анализ результатов

После выполнения всех вышеперечисленных пунктов был разработан *web* - портал маркетплейс-агрегатора на языке программирования *Java* с применением *Spring* для серверной части и использованием *AngularJs* для клиентской части.

Покупателю предоставляется следующий набор возможностей:

– оформлять доставку товаров;

– отслеживать статус доставки;

– просматривать товары на главной странице;

– добавлять товары в корзину.

На рисунке 1.18 представлена *UML* – диаграмма вариантов использования приложения покупателем.

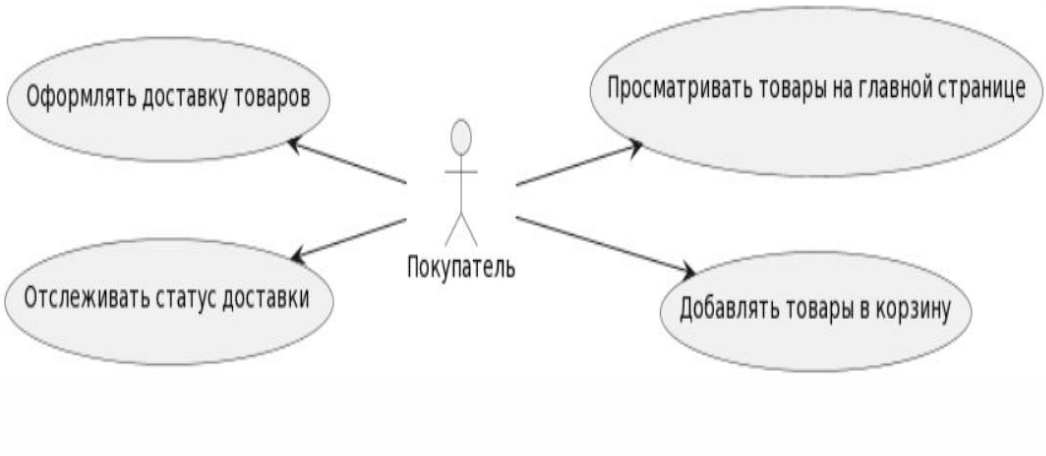


Рисунок 1.18 - *UML*-диаграмма вариантов использования приложения покупателем

Администратору предоставляется следующий набор возможностей:

– просматривать товары на главной странице;

– добавлять новые товары на сайт;

– менять статус доставки и фильтровать по этим статусам товары.

На рисунке 1.19 представлена *UML* – диаграмма вариантов использования приложения администратором.

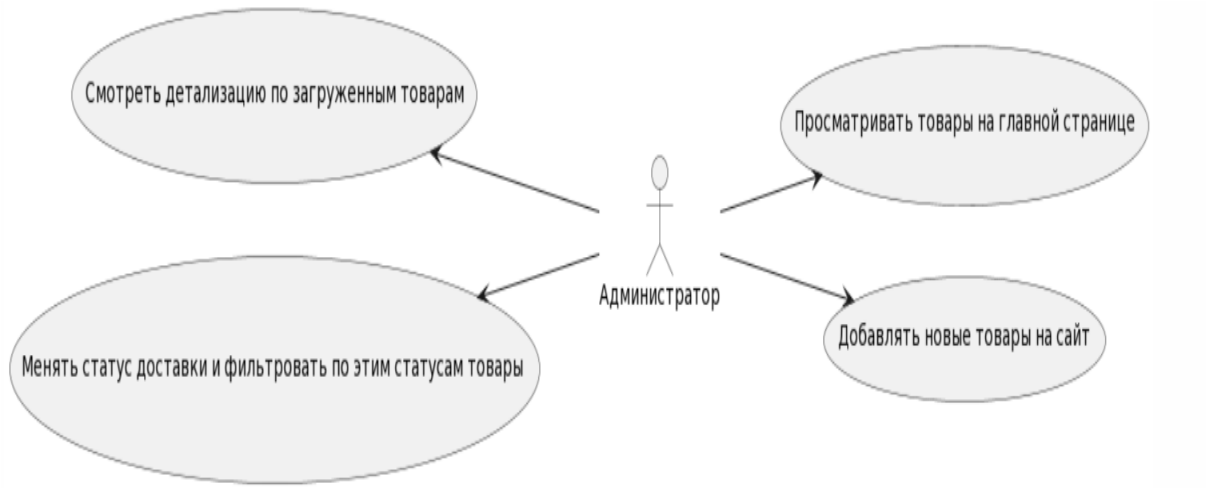


Рисунок 1.19 - *UML*-диаграмма вариантов использования приложения администратором

1.3.1 Тестирование приложения

Для проверки работоспособности приложения было проведено ручное тестирование.

Для тестирования администратора заранее был создан аккаунт с данными *admin123/admin* соответственно. Залогинимся под этими данными, перед нами открывается личный кабинет администратора, представленный на рисунке 1.20.

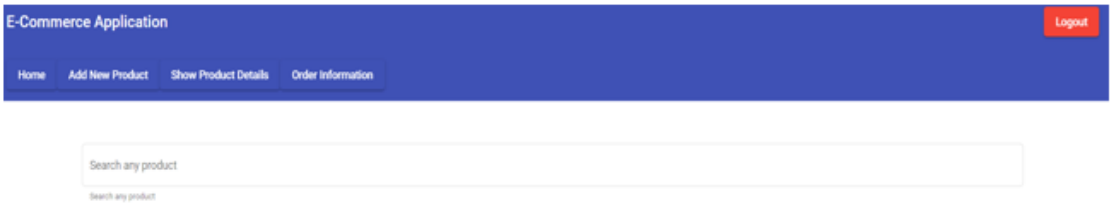


Рисунок 1.20 – Личный кабинет администратора

После этого проверим добавление товара. Перейдем в раздел “*Add New Product*” и выберем новый товар для добавления. Процесс добавления представлен на рисунке 1.21.

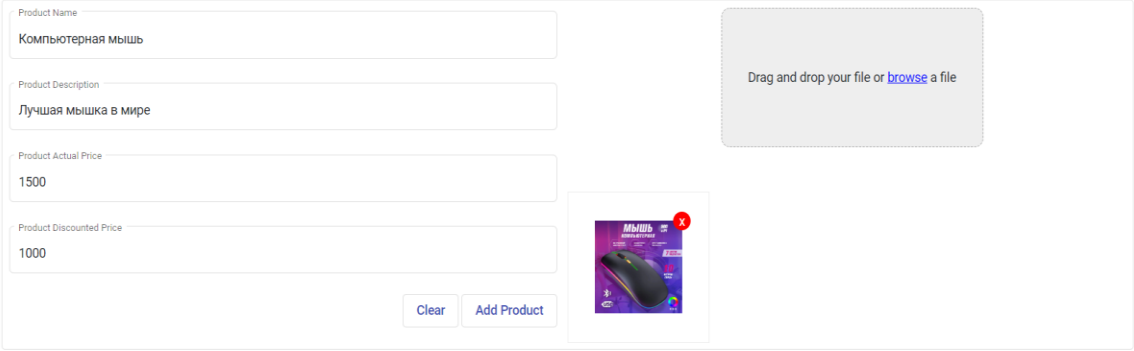


Рисунок 1.21 – Добавление товара на сайт

После этого переходим на главную страницу сайта и убеждаемся, что товар был успешно добавлен. Результат добавления отображен на рисунке 1.22.

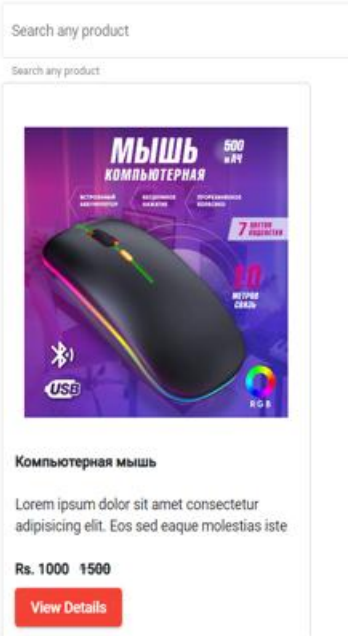


Рисунок 1.22 – Товар был успешно добавлен на главную страницу сайта

Проверим также серверную часть и убедимся, что товар был успешно добавлен в базу данных. Результат добавлен отображен на рисунке 1.23.

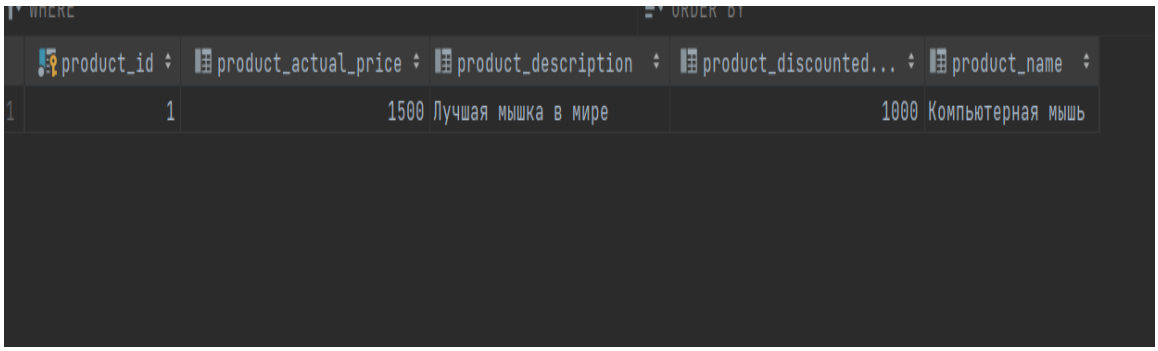


Рисунок 1.23 – Компьютерная мышь успешно добавлена

Проверим редактирование товара и изменим название продукта. На рисунке 1.24 видно, как происходит изменение названия. На рисунке 1.25 видим, что название было успешно изменено.

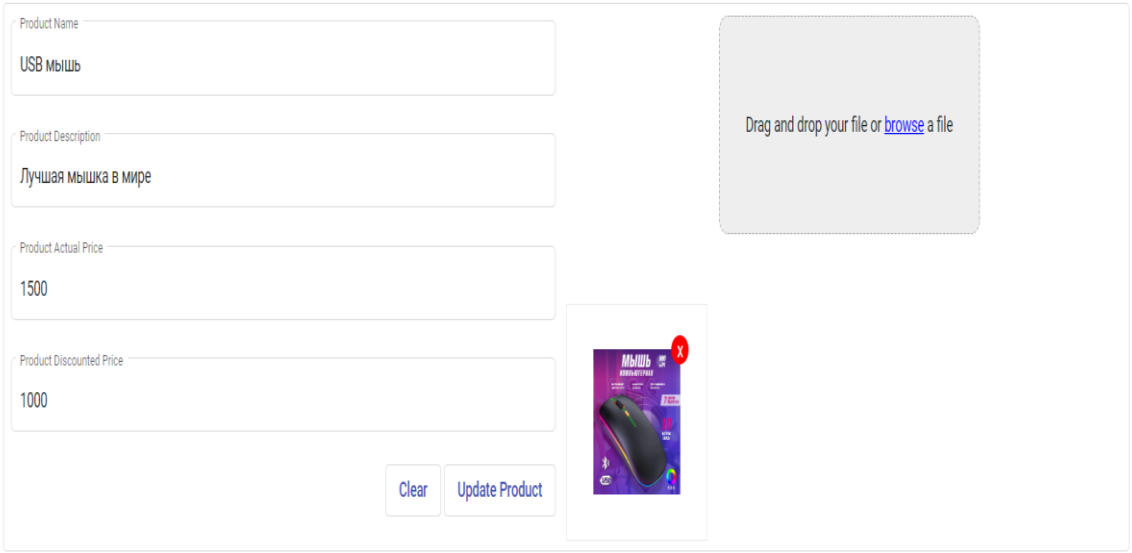


Рисунок 1.24 – Меняем названием товара на специальной странице

Рисунок 1.25 – Название товара было успешно изменено

Проверим теперь взаимодействие с сайтом с точки зрения потребителя. Зарегистрируемся на сайте и зайдем под соответствующим аккаунтом, на рисунке 1.26 можем видеть, что дизайн главной страницы изменился.

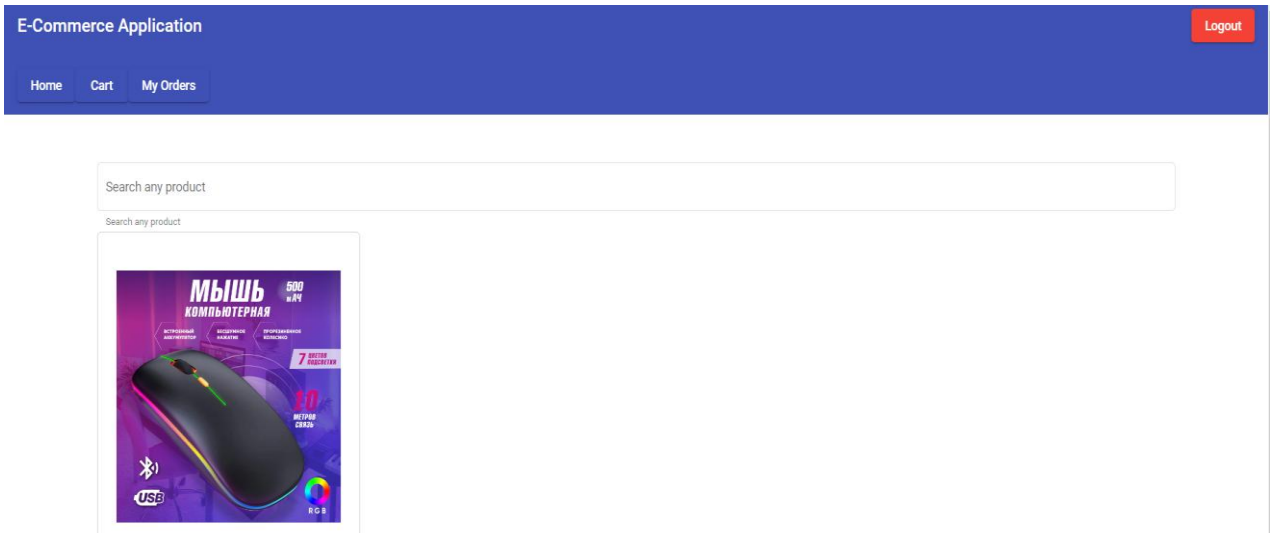


Рисунок 1.26 – Дизайн главной страницы у покупателя

Добавим товар в корзину. На рисунке 1.27 можем убедиться, что товар был успешно добавлен.

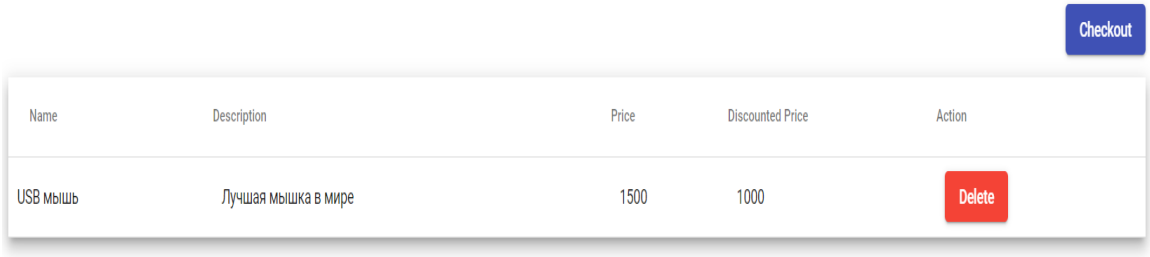


Рисунок 1.27 – Товар был успешно добавлен в корзину

Оформим доставку товара и убедимся, что товар появился в разделе “*My Orders*”. На рисунке 1.28 мы можем видеть, что заказ был успешно оформлен. Также на рисунке 1.29 видим, что в базе данных успешно оформленный заказ также появился.

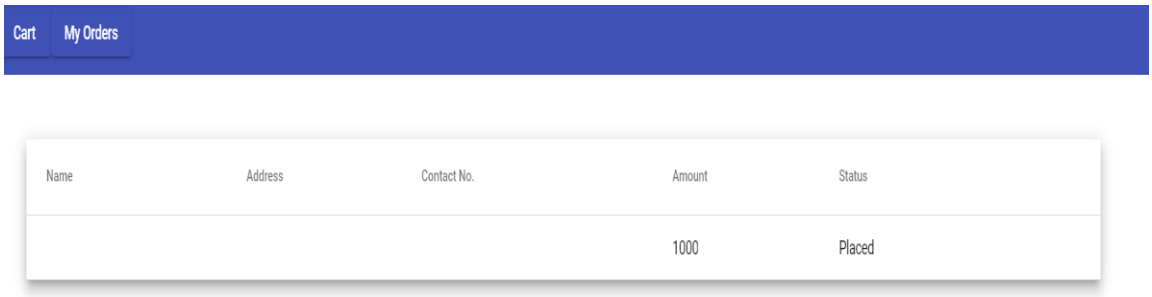


Рисунок 1.28 – Заказ товара был успешно оформлен

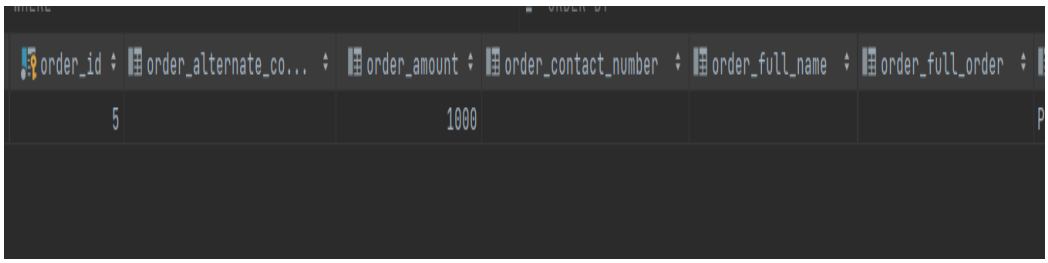


Рисунок 1.29 – Данные о заказе были успешно добавлены в базу

Таким образом, сайт был протестирован, и мы убедились, что он успешно работает.

2 Охрана труда

Охрана труда в сфере разработки приложений является важным аспектом, который необходимо учитывать при работе над проектами. Как и любая другая профессия, разработчики приложений тоже подвергаются определенным рискам и опасностям в рабочей среде.

Соблюдая ряд пунктов, мы можем обеспечить высокую культуру труда в сфере *IT* и тем самым снизить возможные риски для здоровья:

– необходимо обеспечить удобный и безопасный рабочий процесс. Все рабочие места должны соответствовать требованиям безопасности и быть оснащены средствами защиты, такими как наушники, мониторы с защитой глаз от излучения, специальное кресло с регулируемой высотой и удобной спинкой и т.д;

– необходимо обеспечить регулярное обновление рабочего места и оборудования, чтобы избежать возможных поломок мебели и повреждений, что может привести к производственным травмам. Разработчики должны также соблюдать правила электробезопасности, не допускать самостоятельных ремонтов и перестановок электропроводки;

– окружающая среда также должна быть безопасной: помещение должно обеспечивать достаточное освещение и вентиляцию, а также быть чистым и убранным;

– важным элементом охраны труда является профилактический медицинский осмотр, который помогает выявить наличие заболеваний, связанных с работой за компьютером. Этот мероприятие позволяет выявить и предотвратить развитие остеохондроза, нарушения зрения, нарушения нервной системы и др.

В итоге, охрана труда в области разработки приложений является не менее важной, чем в других профессиях и помогает создать безопасную и комфортную среду работы, что положительно влияет на качество и сроки выполнения проектов.

2.1 Характеристики потенциальных опасностей и вредностей, которые могут возникнуть при работе с ЭВМ

Даже соблюдая вышеперечисленные правила и соблюдая все меры предосторожности, *IT* – специалисты тем не менее все равно могут столкнуться с определенные ущербом для своего здоровья :

– длительная работа за компьютером легко приводит к появлению проблем со зрением, поскольку глаза специалиста постоянно направлены на мерцающий экран, что постепенно приводит к их усталости, сухости, и раздражению;

– отсутствие активности и длительная стационарная работа могут способствовать нарушению осанки, появлению болей в спине, шее, руках и ногах. В связи с этим возникают риски для здоровья опорно-двигательного аппарата, которые могут ухудшиться из-за недостатка физических нагрузок;

– инфекционными заболеваниями: клавиатуры и мыши могут стать носителями бактерий и вирусов, которые могут вызвать разные инфекционные заболевания;

– работа с мониторами может подвергать человека воздействию ионизирующего излучения, что может негативно сказываться на его здоровье;

– использование клавиатур и мышей может привести к распространению бактерий и вирусов, которые могут стать причиной возникновения различных инфекционных заболеваний.

Основной причиной возникновения различных проблем со здоровьем может стать продолжительная работа на компьютере, следовательно, для поддержания здоровья необходимо регулярно принимать профилактические меры, такие как соблюдение правильной позы при работе, регулярные перерывы и физические упражнения.

Заключение

В результате дипломной работы была разработана *MVP* – версия *web* – портала маркетплейс – агрегатора, реализующая функционал в соответствии с техническим заданием.

Разработка данного ПО позволила автоматизировать и упростить поиск, покупку, продажу и заказ товара.

В процессе разработки программного обеспечения были изучены и применены на практике технологии: *Spring Framework*, *Maven*, *PostgreSQL*. Выработан навык работы с интегрированной средой разработки. Были освоены современные практики и парадигмы проектирования web – приложений, такие как *REST*, *MVC*, передача по *http* и т. д., получены навыки разработки *API*. Для разработки клиентской части приложения применялся фреймворк *AngularJS*.

На следующем этапе разработки данного проекта предполагается: добавить на сайт необходимые фильтры и сортировки, категории товаров, сделать отдельную роль продавцов для добавления товаров.

На основании вышесказанного, сделан вывод, что все задачи, поставленные в рамках данной выпускной квалификационной работы, были выполнены успешно.

Список использованных источников

1. Блох Д. Эффективная *Java*; Диалектика – Вильямс – М., 2019. – 464 с.

2. Кларенс Х., Харроп Р. *Spring 5* для профессионалов; Диалектика – Вильямс – М., 2020. – 1120 с.

3. Берт Б., Сьерра К. Изучаем *Java*; Эксмо – М., 2022. – 720 с.

4. Мартин Р. Чистый код: создание, анализ и рефакторинг. Библиотека программиста; Питер – М., 2022. – 464 с.

5. Хеклер М. *Spring Boot* по – быстрому; Эксмо – М., 2022. – 352 с.

6. Фримен Э., Робсон Э. *Head First*. Паттерны проектирования. 2 – е издание; Питер – М., 2022. – 640 с.

7. Эккель Б. Философия *Java*. 4 – е полное изд.; Эксмо – М., 2020. – 1242 с.

8. Уоллс К. *Spring* в действии. Шестое издание; ДМК Пресс – М., 2022. – 544 с.

9. Бауэр К., Кинг Г., Грэгори Г. *Java Persistence API* и *Hibernate*; ДМК Пресс – М., 2018. – 632 с.

10. Документация к фреймворку *Spring*. [Электронный ресурс] / Режим доступа: https://docs.spring.io/spring-framework/reference/ свободный

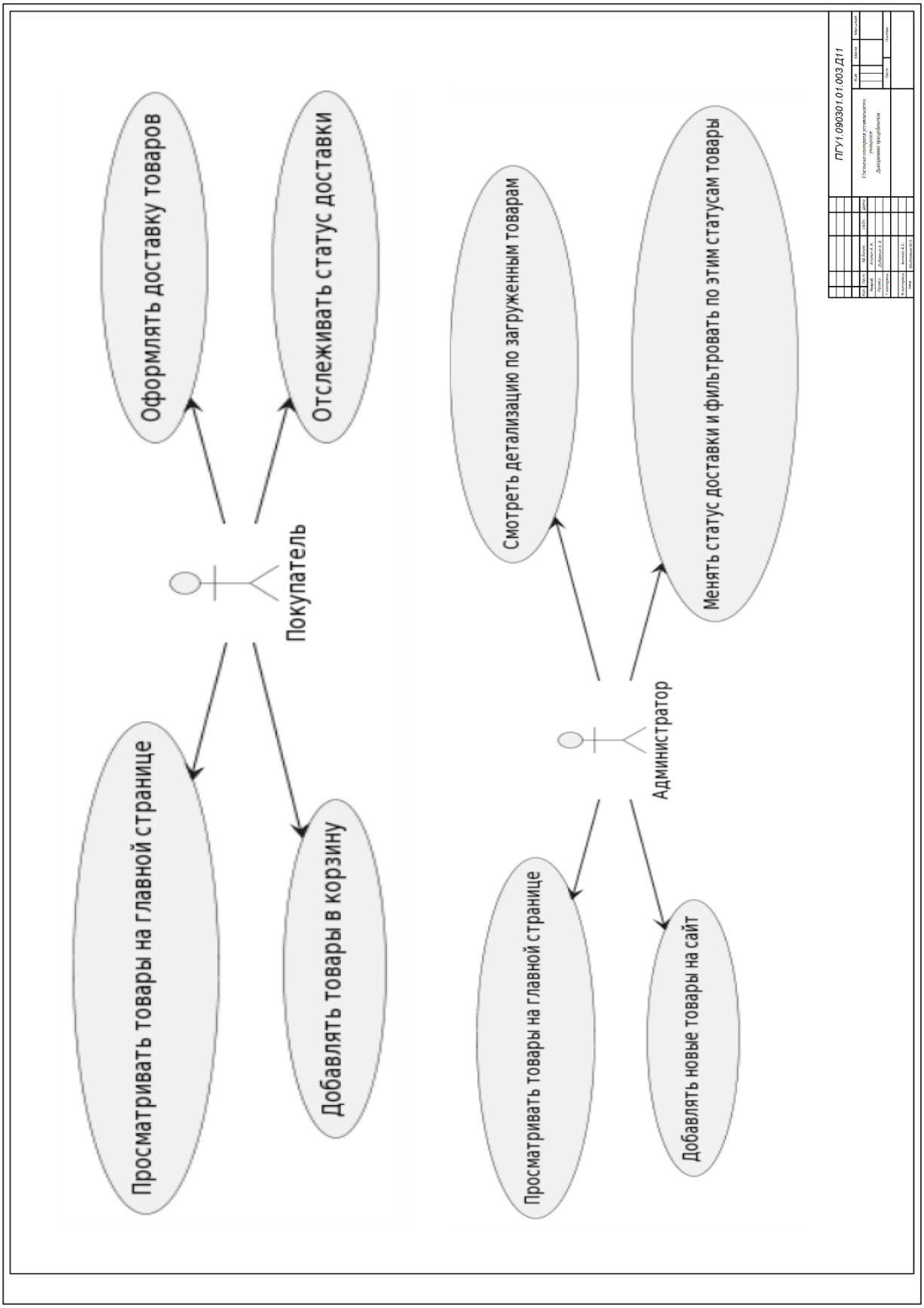
Приложение А

(обязательное)

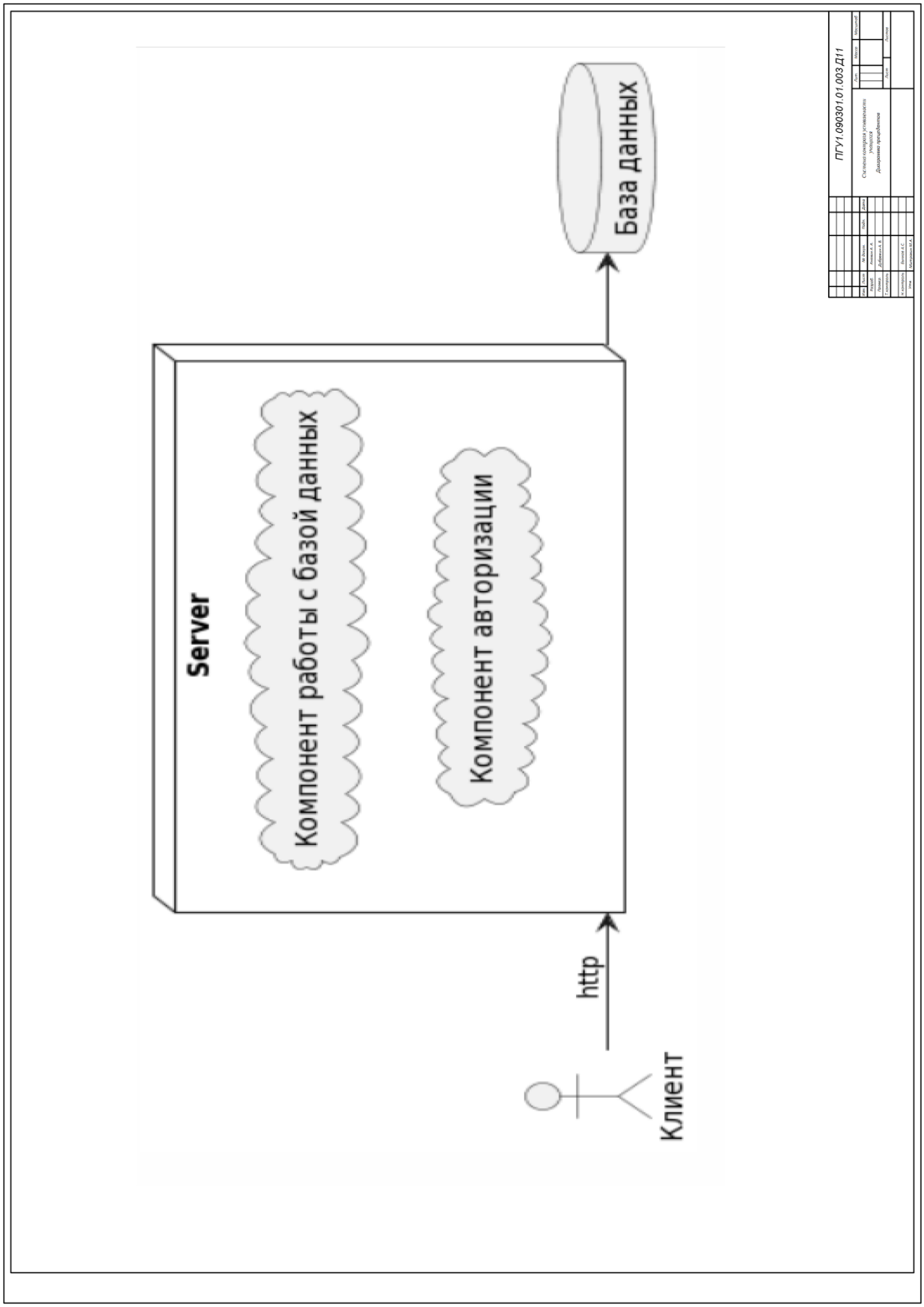
Web – портал маркетплейс - агрегатора

UML - диаграммы

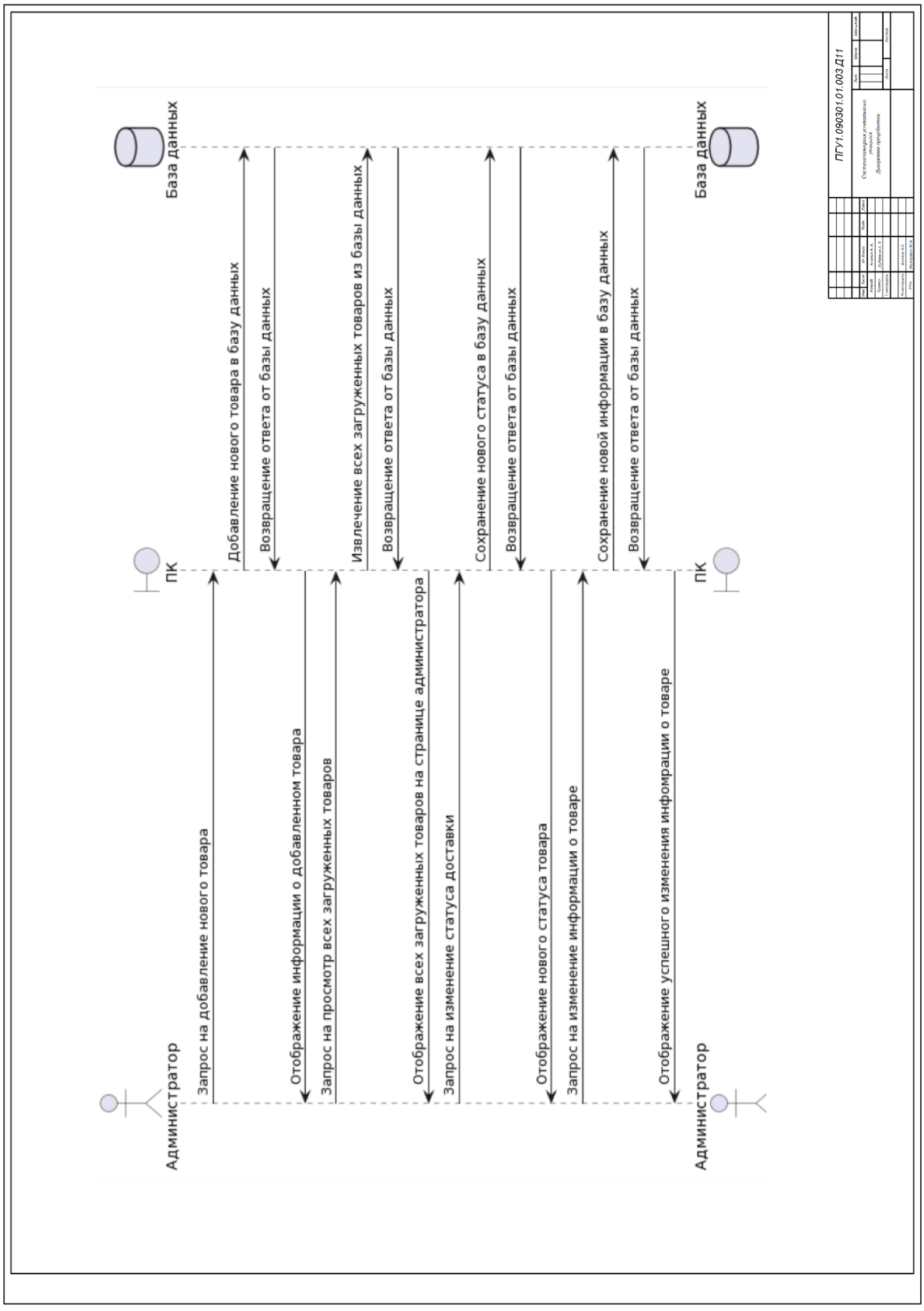
Приложение А.1 – Диаграмма вариантов использовани



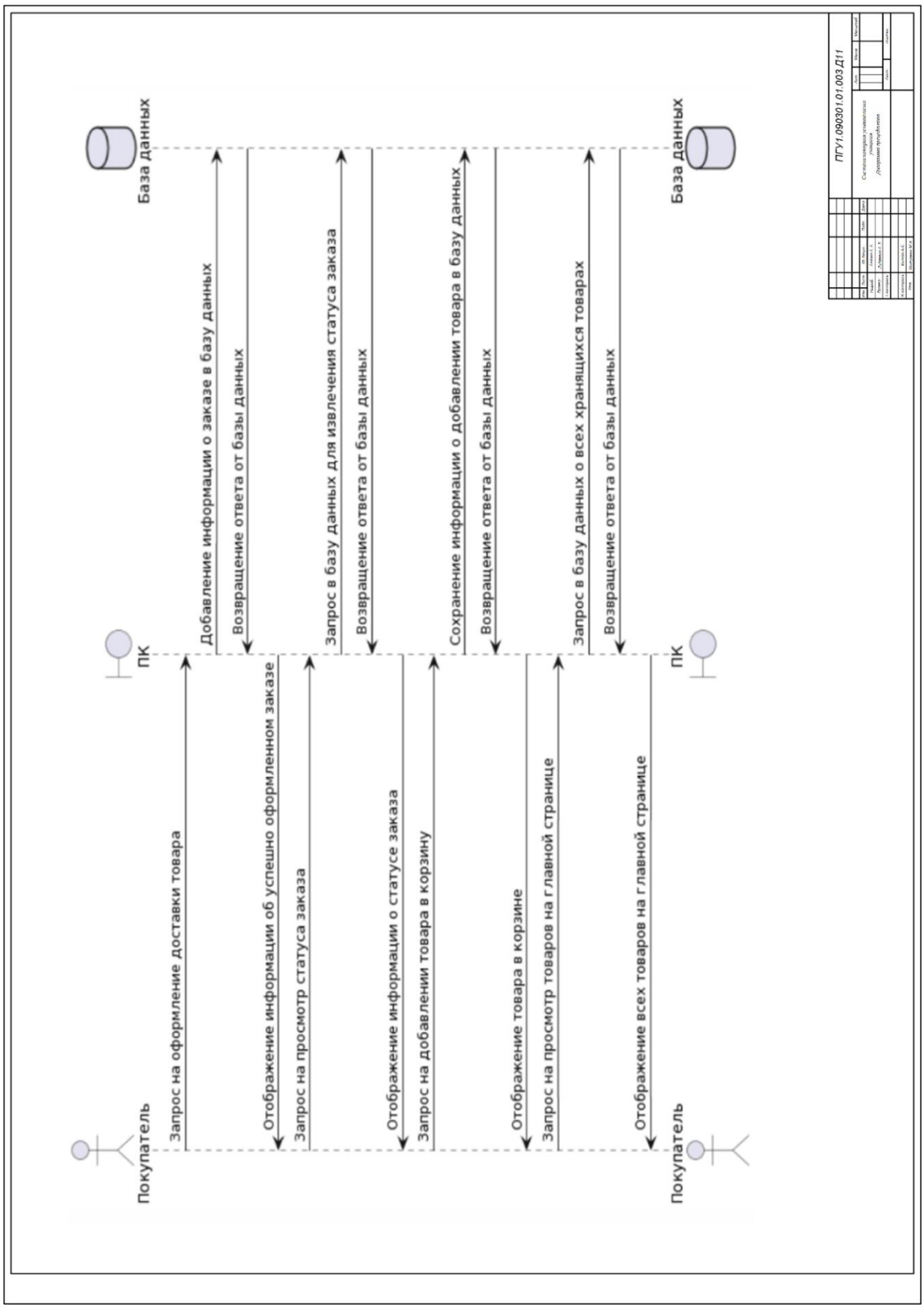
Приложение А.2 – Диаграмма развертывания



Приложение А.3 – Диаграмма развертывания



Приложение А.4 – Диаграмма развертывания



Приложение Б

(обязательное)

Web - портал маркетплейс - агрегатора

Листинг программы

Файл ECommerceApplication.java

```
@SpringBootApplication
public class ECommerceApplication {

    public static void main(String[] args) {
        SpringApplication.run(ECommerceApplication.class, args);
    }

}
```

Файл CorsConfiguration.java

```
@Configuration
public class CorsConfiguration {

    private static final String GET = "GET";
    private static final String POST = "POST";
    private static final String PUT = "PUT";
    private static final String DELETE = "DELETE";

    @Bean
    public WebMvcConfigurer corsConfigurer() {
        return new WebMvcConfigurer() {
            @Override
            public void addCorsMappings(CorsRegistry registry) {
                registry.addMapping("/**")
                        .allowedMethods(GET, POST, PUT, DELETE)
                        .allowedHeaders("*")
                        .allowedOriginPatterns("*")
                        .allowCredentials(true);
            }
        };
```

```
    }
}
```

Файл JwtAuthenticationEntryPoint.java

```
@Component
public class JwtAuthenticationEntryPoint implements
AuthenticationEntryPoint {


    @Override
    public void commence(HttpServletRequest request,
HttpServletResponse response, AuthenticationException
authException) throws IOException, ServletException {
        response.sendError(HttpServletResponse.SC_UNAUTHORIZED,
"Unauthorized");
    }


}
```

Файл JwtRequestFilter.java

```
@Component
public class JwtRequestFilter extends OncePerRequestFilter {


    public static String CURRENT_USER = "";


    @Autowired
    private JwtUtil jwtUtil;


    @Autowired
    private JwtService jwtService;


    @Override
    protected void doFilterInternal(HttpServletRequest request,
HttpServletResponse response, FilterChain filterChain) throws
ServletException, IOException {
```

```
        final String requestTokenHeader =
request.getHeader("Authorization");


        String username = null;
        String jwtToken = null;


        if (requestTokenHeader != null &&
requestTokenHeader.startsWith("Bearer ")) {
            jwtToken = requestTokenHeader.substring(7);
            try {
                username =
jwtUtil.getUsernameFromToken(jwtToken);
                CURRENT_USER = username;
            } catch (IllegalArgumentException e) {
                System.out.println("Unable to get JWT Token");
            } catch (ExpiredJwtException e) {
                System.out.println("JWT Token has expired");
            }
        } else {
            System.out.println("JWT token does not start with
Bearer");
        }


        if (username != null &&
SecurityContextHolder.getContext().getAuthentication() == null)
{


            UserDetails userDetails =
jwtService.loadUserByUsername(username);


            if (jwtUtil.validateToken(jwtToken, userDetails)) {


                UsernamePasswordAuthenticationToken
usernamePasswordAuthenticationToken = new
```

```
UsernamePasswordAuthenticationToken(userDetails, null,
userDetails.getAuthorities());

usernamePasswordAuthenticationToken.setDetails(new
WebAuthenticationDetailsSource().buildDetails(request));

SecurityContextHolder.getContext().setAuthentication(usernamePas
swordAuthenticationToken);
            }
        }
        filterChain.doFilter(request, response);

    }

}
```

Файл WebSecurityConfiguration.java

```
@Configuration
@EnableWebSecurity
@EnableGlobalMethodSecurity(prePostEnabled = true)
public class WebSecurityConfiguration extends
WebSecurityConfigurerAdapter {

    @Autowired
    private JwtAuthenticationEntryPoint
jwtAuthenticationEntryPoint;

    @Autowired
    private JwtRequestFilter jwtRequestFilter;

    @Autowired
    private UserDetailsService jwtService;

    @Bean
    @Override
```

```
    public AuthenticationManager authenticationManagerBean()
throws Exception {
        return super.authenticationManagerBean();
    }

    @Override
    protected void configure(HttpSecurity httpSecurity) throws
Exception {
        httpSecurity.cors();
        httpSecurity.csrf().disable()

.authorizeRequests().antMatchers("/authenticate",
"/registerNewUser", "/getAllProducts",
"/getProductDetailsById/{productId}").permitAll()
                .antMatchers(HttpHeaders.ALLOW).permitAll()
                .anyRequest().authenticated()
                .and()

.exceptionHandling().authenticationEntryPoint(jwtAuthenticationE
ntryPoint)
                .and()

.sessionManagement().sessionCreationPolicy(SessionCreationPolicy
.STATELESS)
        ;

        httpSecurity.addFilterBefore(jwtRequestFilter,
UsernamePasswordAuthenticationFilter.class);
    }

    @Bean
    public PasswordEncoder passwordEncoder() {
        return new BCryptPasswordEncoder();
    }
```

```
    @Autowired
    public void configureGlobal(AuthenticationManagerBuilder
authenticationManagerBuilder) throws Exception {

authenticationManagerBuilder.userDetailsService(jwtService).pass
wordEncoder(passwordEncoder());
    }
}
```

Файл CartController.java

```
@RestController
public class CartController {

    @Autowired
    private CartService cartService;

    @PreAuthorize("hasRole('User')")
    @GetMapping({"/addToCart/{productId}"})
    public Cart addToCart(@PathVariable(name = "productId")
Integer productId) {
        return cartService.addToCart(productId);
    }

    @PreAuthorize("hasRole('User')")
    @DeleteMapping({"/deleteCartItem/{cartId}"})
    public void deleteCartItem(@PathVariable(name = "cartId")
Integer cartId) {
        cartService.deleteCartItem(cartId);
    }

    @PreAuthorize("hasRole('User')")
    @GetMapping({"/getCartDetails"})
    public List<Cart> getCartDetails() {
        return cartService.getCartDetails();
```

```
    }
}
```

Файл JwtController.java

```
@RestController
@CrossOrigin
public class JwtController {

    @Autowired
    private JwtService jwtService;

    @PostMapping({"/authenticate"})
    public JwtResponse createJwtToken(@RequestBody JwtRequest
jwtRequest) throws Exception {
        return jwtService.createJwtToken(jwtRequest);
    }
}
```

Файл OrderDetailController.java

```
@RestController
public class OrderDetailController {

    @Autowired
    private OrderDetailService orderDetailService;

    @PreAuthorize("hasRole('User')")
    @PostMapping({"/placeOrder/{isSingleProductCheckout}"})
    public void placeOrder(@PathVariable(name =
"isSingleProductCheckout") boolean isSingleProductCheckout,
            @RequestBody OrderInput orderInput) {
        orderDetailService.placeOrder(orderInput,
isSingleProductCheckout);
    }
```

```
    @PreAuthorize("hasRole('User')")
    @GetMapping({"/getOrderDetails"})
    public List<OrderDetail> getOrderDetails() {
        return orderDetailService.getOrderDetails();
    }

    @PreAuthorize("hasRole('Admin')")
    @GetMapping({"/getAllOrderDetails/{status}"})
    public List<OrderDetail>
getAllOrderDetails(@PathVariable(name = "status") String status)
{
        return orderDetailService.getAllOrderDetails(status);
    }

    @PreAuthorize("hasRole('Admin')")
    @GetMapping({"/markOrderAsDelivered/{orderId}"})
    public void markOrderAsDelivered(@PathVariable(name =
"orderId") Integer orderId) {
        orderDetailService.markOrderAsDelivered(orderId);
    }

    @PreAuthorize("hasRole('User')")
    @GetMapping({"/createTransaction/{amount}"})
    public TransactionDetails
createTransaction(@PathVariable(name = "amount") Double amount)
{
        return orderDetailService.createTransaction(amount);
    }
}
```

Файл ProductController.java

```
@RestController
public class ProductController {
```

```
    @Autowired
    private ProductService productService;


    @PreAuthorize("hasRole('Admin')")
    @PostMapping(value = {"/addNewProduct"}, consumes =
{MediaType.MULTIPART_FORM_DATA_VALUE})
    public Product addNewProduct(@RequestPart("product") Product
product,
                                 @RequestPart("imageFile")
MultipartFile[] file) {
        try {
            Set<ImageModel> images = uploadImage(file);
            product.setProductImages(images);
            return productService.addNewProduct(product);
        } catch (Exception e) {
            System.out.println(e.getMessage());
            return null;
        }

    }


    public Set<ImageModel> uploadImage(MultipartFile[]
multipartFiles) throws IOException {
        Set<ImageModel> imageModels = new HashSet<>();

        for (MultipartFile file: multipartFiles) {
            ImageModel imageModel = new ImageModel(
                    file.getOriginalFilename(),
                    file.getContentType(),
                    file.getBytes()
            );
            imageModels.add(imageModel);
```

```
        }


        return imageModels;
    }


    @GetMapping({"/getAllProducts"})
    public List<Product>
getAllProducts(@RequestParam(defaultValue = "0") int pageNumber,

@RequestParam(defaultValue = "") String searchKey) {
        List<Product> result =
productService.getAllProducts(pageNumber, searchKey);
        System.out.println("Result size is "+ result.size());
        return result;
    }


    @GetMapping({"/getProductDetailsById/{productId}"})
    public Product
getProductDetailsById(@PathVariable("productId") Integer
productId) {
        return productService.getProductDetailsById(productId);
    }


    @PreAuthorize("hasRole('Admin')")
    @DeleteMapping({"/deleteProductDetails/{productId}"})
    public void deleteProductDetails(@PathVariable("productId")
Integer productId) {
        productService.deleteProductDetails(productId);
    }


    @PreAuthorize("hasRole('User')")

@GetMapping({"/getProductDetails/{isSingleProductCheckout}/{prod
uctId}"})
```

```
    public List<Product> getProductDetails(@PathVariable(name =
"isSingleProductCheckout" ) boolean isSingleProductCheckout,
                                           @PathVariable(name =
"productId")  Integer productId) {
        return
productService.getProductDetails(isSingleProductCheckout,
productId);
    }
}
```

Файл RoleController.java

```
@RestController
public class RoleController {

    @Autowired
    private RoleService roleService;

    @PostMapping({"/createNewRole"})
    public Role createNewRole(@RequestBody Role role) {
        return roleService.createNewRole(role);
    }
}
```

Файл UserController.java

```
import javax.annotation.PostConstruct;

@RestController
public class UserController {

    @Autowired
    private UserService userService;

    @PostConstruct
    public void initRoleAndUser() {
        userService.initRoleAndUser();
```

```
    }

    @PostMapping({"/registerNewUser"})
    public Users registerNewUser(@RequestBody Users users) {
        return userService.registerNewUser(users);
    }

    @GetMapping({"/forAdmin"})
    @PreAuthorize("hasRole('Admin')")
    public String forAdmin(){
        return "This URL is only accessible to the admin";
    }

    @GetMapping({"/forUser"})
    @PreAuthorize("hasRole('User')")
    public String forUser(){
        return "This URL is only accessible to the user";
    }
}
```

Файл CartDao.java

```
@Repository
public interface CartDao extends CrudRepository<Cart, Integer >
{
    public List<Cart> findByUsers(Users users);
}
```

Файл OrderDetailDao.java

```
public interface OrderDetailDao extends
CrudRepository<OrderDetail, Integer> {
```

```
    public List<OrderDetail> findByUsers(Users users);


    public List<OrderDetail> findByOrderStatus(String status);
}
```

Файл ProductDao.java

```
@Repository
public interface ProductDao extends CrudRepository<Product,
Integer> {
    public List<Product> findAll(Pageable pageable);


    public List<Product>
findByProductNameContainingIgnoreCaseOrProductDescriptionContain
ingIgnoreCase(
            String key1, String key2, Pageable pageable
    );
}
```

Файл RoleDao.java

```
@Repository
public interface RoleDao extends CrudRepository<Role, String> {


}
```

Файл UserDao.java

```
@Repository
public interface UserDao extends CrudRepository<Users, String> {
}
```

Файл Cart.java

```
@Entity
@Table
public class Cart {


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

```
private Integer cartId;
@OneToOne
private Product product;
@OneToOne
private Users users;

public Cart(){

}

public Cart(Product product, Users users) {
    this.product = product;
    this.users = users;
}

public Integer getCartId() {
    return cartId;
}

public void setCartId(Integer cartId) {
    this.cartId = cartId;
}

public Product getProduct() {
    return product;
}

public void setProduct(Product product) {
    this.product = product;
}
```

```
    public Users getUser() {
        return users;
    }

    public void setUser(Users users) {
        this.users = users;
    }
}
```

Файл ImageModel.java

```
@Entity
@Table(name = "image_model")
public class ImageModel {

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private Long id;
    private String name;
    private String type;
    @Column(length = 50000000)
    private byte[] picByte;

    public ImageModel() {

    }

    public ImageModel(String name, String type, byte[] picByte)
{
        this.name = name;
        this.type = type;
        this.picByte = picByte;
    }
```

```
public Long getId() {
    return id;
}

public void setId(Long id) {
    this.id = id;
}

public String getName() {
    return name;
}

public void setName(String name) {
    this.name = name;
}

public String getType() {
    return type;
}

public void setType(String type) {
    this.type = type;
}

public byte[] getPicByte() {
    return picByte;
}

public void setPicByte(byte[] picByte) {
    this.picByte = picByte;
}
```

```
}
```

Файл JwtRequest.java

```
package com.ecommerce.entity;

public class JwtRequest {

    private String userName;
    private String userPassword;

    public String getUserName() {
        return userName;
    }

    public void setUserName(String userName) {
        this.userName = userName;
    }

    public String getUserPassword() {
        return userPassword;
    }

    public void setUserPassword(String userPassword) {
        this.userPassword = userPassword;
    }
}
```

Файл JwtResponse.java

```
package com.ecommerce.entity;

public class JwtResponse {

    private Users users;
```

```
    private String jwtToken;

    public JwtResponse(Users users, String jwtToken) {
        this.users = users;
        this.jwtToken = jwtToken;
    }

    public Users getUser() {
        return users;
    }

    public void setUser(Users users) {
        this.users = users;
    }

    public String getJwtToken() {
        return jwtToken;
    }

    public void setJwtToken(String jwtToken) {
        this.jwtToken = jwtToken;
    }
}
```

Файл OrderDetail.java

```
package com.ecommerce.entity;

import javax.persistence.*;

@Entity
public class OrderDetail {
```

```
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private Integer orderId;
    private String orderFullName;
    private String orderFullOrder;
    private String orderContactNumber;
    private String orderAlternateContactNumber;
    private String orderStatus;
    private Double orderAmount;
    @OneToOne
    private Product product;
    @OneToOne
    private Users users;
    private String transactionId;


    public OrderDetail() {


    }


    public OrderDetail(String orderFullName, String
orderFullOrder, String orderContactNumber, String
orderAlternateContactNumber, String orderStatus, Double
orderAmount, Product product, Users users, String transactionId)
{
        this.orderFullName = orderFullName;
        this.orderFullOrder = orderFullOrder;
        this.orderContactNumber = orderContactNumber;
        this.orderAlternateContactNumber =
orderAlternateContactNumber;
        this.orderStatus = orderStatus;
        this.orderAmount = orderAmount;
        this.product = product;
        this.users = users;
```

```
        this.transactionId = transactionId;
    }

    public String getTransactionId() {
        return transactionId;
    }

    public void setTransactionId(String transactionId) {
        this.transactionId = transactionId;
    }

    public Product getProduct() {
        return product;
    }

    public void setProduct(Product product) {
        this.product = product;
    }

    public Users getUser() {
        return users;
    }

    public void setUser(Users users) {
        this.users = users;
    }

    public Integer getOrderId() {
        return orderId;
    }
```

```
    public void setOrderId(Integer orderId) {
        this.orderId = orderId;
    }

    public String getOrderFullName() {
        return orderFullName;
    }

    public void setOrderFullName(String orderFullName) {
        this.orderFullName = orderFullName;
    }

    public String getOrderFullOrder() {
        return orderFullOrder;
    }

    public void setOrderFullOrder(String orderFullOrder) {
        this.orderFullOrder = orderFullOrder;
    }

    public String getOrderContactNumber() {
        return orderContactNumber;
    }

    public void setOrderContactNumber(String orderContactNumber)
{
        this.orderContactNumber = orderContactNumber;
    }

    public String getOrderAlternateContactNumber() {
        return orderAlternateContactNumber;
    }
```

```
    public void setOrderAlternateContactNumber(String
orderAlternateContactNumber) {
        this.orderAlternateContactNumber =
orderAlternateContactNumber;
    }

    public String getOrderStatus() {
        return orderStatus;
    }

    public void setOrderStatus(String orderStatus) {
        this.orderStatus = orderStatus;
    }

    public Double getOrderAmount() {
        return orderAmount;
    }

    public void setOrderAmount(Double orderAmount) {
        this.orderAmount = orderAmount;
    }
}
```

Файл OrderInput.java

```
package com.ecommerce.entity;

import java.util.List;

public class OrderInput {

    private String fullName;
    private String fullAddress;
```

```
    private String contactNumber;
    private String alternateContactNumber;
    private String transactionId;
    private List<OrderProductQuantity> orderProductQuantityList;

    public String getTransactionId() {
        return transactionId;
    }

    public void setTransactionId(String transactionId) {
        this.transactionId = transactionId;
    }

    public String getFullName() {
        return fullName;
    }

    public void setFullName(String fullName) {
        this.fullName = fullName;
    }

    public String getFullAddress() {
        return fullAddress;
    }

    public void setFullAddress(String fullAddress) {
        this.fullAddress = fullAddress;
    }

    public String getContactNumber() {
        return contactNumber;
```

```
    }

    public void setContactNumber(String contactNumber) {
        this.contactNumber = contactNumber;
    }

    public String getAlternateContactNumber() {
        return alternateContactNumber;
    }

    public void setAlternateContactNumber(String
alternateContactNumber) {
        this.alternateContactNumber = alternateContactNumber;
    }

    public List<OrderProductQuantity>
getOrderProductQuantityList() {
        return orderProductQuantityList;
    }

    public void
setOrderProductQuantityList(List<OrderProductQuantity>
orderProductQuantityList) {
        this.orderProductQuantityList =
orderProductQuantityList;
    }
}
```

Файл OrderProductQuantity.java

```
package com.ecommerce.entity;

public class OrderProductQuantity {

    private Integer productId;
```

```
    private Integer quantity;

    public Integer getProductId() {
        return productId;
    }

    public void setProductId(Integer productId) {
        this.productId = productId;
    }

    public Integer getQuantity() {
        return quantity;
    }

    public void setQuantity(Integer quantity) {
        this.quantity = quantity;
    }
}
```

Файл Product.java

```
@Entity
public class Product {

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private Integer productId;
    private String productName;
    @Column(length = 2000)
    private String productDescription;
    private Double productDiscountedPrice;
    private Double productActualPrice;
    @ManyToMany(fetch = FetchType.EAGER, cascade =
CascadeType.ALL)
```

```
    @JoinTable(name = "product_images",
            joinColumns = {
                    @JoinColumn(name = "product_id")
            },
            inverseJoinColumns = {
                    @JoinColumn(name = "image_id")
            }
    )
    private Set<ImageModel> productImages;

    public Set<ImageModel> getProductImages() {
        return productImages;
    }

    public void setProductImages(Set<ImageModel> productImages)
{
        this.productImages = productImages;
    }


    public Integer getProductId() {
        return productId;
    }

    public void setProductId(Integer productId) {
        this.productId = productId;
    }

    public String getProductName() {
        return productName;
    }
```

```
    public void setProductName(String productName) {
        this.productName = productName;
    }

    public String getProductDescription() {
        return productDescription;
    }

    public void setProductDescription(String productDescription)
{
        this.productDescription = productDescription;
    }

    public Double getProductDiscountedPrice() {
        return productDiscountedPrice;
    }

    public void setProductDiscountedPrice(Double
productDiscountedPrice) {
        this.productDiscountedPrice = productDiscountedPrice;
    }

    public Double getProductActualPrice() {
        return productActualPrice;
    }

    public void setProductActualPrice(Double productActualPrice)
{
        this.productActualPrice = productActualPrice;
    }
}
```

Файл Role.java

```
@Entity
public class Role {

    @Id
    private String roleName;
    private String roleDescription;

    public String getRoleName() {
        return roleName;
    }

    public void setRoleName(String roleName) {
        this.roleName = roleName;
    }

    public String getRoleDescription() {
        return roleDescription;
    }

    public void setRoleDescription(String roleDescription) {
        this.roleDescription = roleDescription;
    }
}
```

Файл TransactionDetails.java

```
package com.ecommerce.entity;

public class TransactionDetails {

    private String orderId;
    private String currency;
    private Integer amount;
```

```
    private String key;

    public TransactionDetails(String orderId, String currency,
Integer amount, String key) {
        this.orderId = orderId;
        this.currency = currency;
        this.amount = amount;
        this.key = key;
    }

    public String getKey() {
        return key;
    }

    public void setKey(String key) {
        this.key = key;
    }

    public String getOrderId() {
        return orderId;
    }

    public void setOrderId(String orderId) {
        this.orderId = orderId;
    }

    public String getCurrency() {
        return currency;
    }

    public void setCurrency(String currency) {
        this.currency = currency;
```

```
    }

    public Integer getAmount() {
        return amount;
    }

    public void setAmount(Integer amount) {
        this.amount = amount;
    }
}
```

Файл Users.java

```
@Entity
public class Users {

    @Id
    private String userName;
    private String userFirstName;
    private String userLastName;
    private String userPassword;
    @ManyToMany(fetch = FetchType.EAGER, cascade =
CascadeType.ALL)
    @JoinTable(name = "USER_ROLE",
            joinColumns = {
                    @JoinColumn(name = "USER_ID")
            },
            inverseJoinColumns = {
                    @JoinColumn(name = "ROLE_ID")
            }
    )
    private Set<Role> role;

    public String getUserName() {
```

```
        return userName;
    }

    public void setUserName(String userName) {
        this.userName = userName;
    }

    public String getUserFirstName() {
        return userFirstName;
    }

    public void setUserFirstName(String userFirstName) {
        this.userFirstName = userFirstName;
    }

    public String getUserLastName() {
        return userLastName;
    }

    public void setUserLastName(String userLastName) {
        this.userLastName = userLastName;
    }

    public String getUserPassword() {
        return userPassword;
    }

    public void setUserPassword(String userPassword) {
        this.userPassword = userPassword;
    }
```

```
    public Set<Role> getRole() {
        return role;
    }

    public void setRole(Set<Role> role) {
        this.role = role;
    }
}
```

Файл CartService.java

```
@Service
public class CartService {

    @Autowired
    private CartDao cartDao;

    @Autowired
    private ProductDao productDao;

    @Autowired
    private UserDao userDao;

    public void deleteCartItem(Integer cartId) {
        cartDao.deleteById(cartId);
    }

    public Cart addToCart(Integer productId) {
        Product product = productDao.findById(productId).get();

        String username = JwtRequestFilter.CURRENT_USER;

        Users users = null;
```

```
        if(username != null) {
            users = userDao.findById(username).get();
        }

        List<Cart> cartList = cartDao.findByUsers(users);
        List<Cart> filteredList = cartList.stream().filter(x ->
x.getProduct().getProductId() ==
productId).collect(Collectors.toList());

        if(filteredList.size() > 0) {
            return null;
        }

        if(product != null && users != null) {
            Cart cart = new Cart(product, users);
            return cartDao.save(cart);
        }

        return null;
    }

    public List<Cart> getCartDetails() {
        String username = JwtRequestFilter.CURRENT_USER;
        Users users = userDao.findById(username).get();
        return cartDao.findByUsers(users);
    }
}
```

Файл JwtService.java

```
@Service
public class JwtService implements UserDetailsService {
```

```
    @Autowired
    private JwtUtil jwtUtil;

    @Autowired
    private UserDao userDao;

    @Autowired
    private AuthenticationManager authenticationManager;

    public JwtResponse createJwtToken(JwtRequest jwtRequest)
throws Exception {
        String userName = jwtRequest.getUserName();
        String userPassword = jwtRequest.getUserPassword();
        authenticate(userName, userPassword);

        UserDetails userDetails = loadUserByUsername(userName);
        String newGeneratedToken =
jwtUtil.generateToken(userDetails);

        Users users = userDao.findById(userName).get();
        return new JwtResponse(users, newGeneratedToken);
    }

    @Override
    public UserDetails loadUserByUsername(String username)
throws UsernameNotFoundException {
        Users users = userDao.findById(username).get();

        if (users != null) {
            return new
org.springframework.security.core.userdetails.User(
                    users.getUserName(),
```

```
                    users.getUserPassword(),
                    getAuthority(users)
            );
        } else {
            throw new UsernameNotFoundException("User not found
with username: " + username);
        }
    }

    private Set getAuthority(Users users) {
        Set<SimpleGrantedAuthority> authorities = new
HashSet<>();
        users.getRole().forEach(role -> {
            authorities.add(new SimpleGrantedAuthority("ROLE_" +
role.getRoleName()));
        });
        return authorities;
    }

    private void authenticate(String userName, String
userPassword) throws Exception {
        try {
            authenticationManager.authenticate(new
UsernamePasswordAuthenticationToken(userName, userPassword));
        } catch (DisabledException e) {
            throw new Exception("USER_DISABLED", e);
        } catch (BadCredentialsException e) {
            throw new Exception("INVALID_CREDENTIALS", e);
        }
    }
}
```

Файл OderDetailService.java

```
package com.ecommerce.service;
```

```
import com.ecommerce.configuration.JwtRequestFilter;
import com.ecommerce.entity.*;
import com.razorpay.Order;
import com.razorpay.RazorpayClient;
import com.ecommerce.dao.CartDao;

@Service
public class OrderDetailService {

    private static final String ORDER_PLACED = "Placed";

    private static final String KEY = "rzp_test_AXBzvN2fkD4ESK";
    private static final String KEY_SECRET =
"bsZmiVD7p1GMo6hAWiy4SHSH";
    private static final String CURRENCY = "INR";

    @Autowired
    private OrderDetailDao orderDetailDao;

    @Autowired
    private ProductDao productDao;

    @Autowired
    private UserDao userDao;

    @Autowired
    private CartDao cartDao;

    public List<OrderDetail> getAllOrderDetails(String status) {
        List<OrderDetail> orderDetails = new ArrayList<>();
```

```
        if(status.equals("All")) {
            orderDetailDao.findAll().forEach(
                    x -> orderDetails.add(x)
            );
        } else {
            orderDetailDao.findByOrderStatus(status).forEach(
                    x -> orderDetails.add(x)
            );
        }


         return orderDetails;
    }

    public List<OrderDetail> getOrderDetails() {
        String currentUser = JwtRequestFilter.CURRENT_USER;
        Users users = userDao.findById(currentUser).get();

        return orderDetailDao.findByUsers(users);
    }

    public void placeOrder(OrderInput orderInput, boolean
isSingleProductCheckout) {
        List<OrderProductQuantity> productQuantityList =
orderInput.getOrderProductQuantityList();

        for (OrderProductQuantity o: productQuantityList) {
            Product product =
productDao.findById(o.getProductId()).get();

            String currentUser = JwtRequestFilter.CURRENT_USER;
            Users users = userDao.findById(currentUser).get();
```

```
            OrderDetail orderDetail = new OrderDetail(
                  orderInput.getFullName(),
                  orderInput.getFullAddress(),
                  orderInput.getContactNumber(),
                  orderInput.getAlternateContactNumber(),
                    ORDER_PLACED,
                    product.getProductDiscountedPrice() *
o.getQuantity(),
                    product,
                    users,
                    orderInput.getTransactionId()
            );

            // empty the cart.
            if(!isSingleProductCheckout) {
                List<Cart> carts = cartDao.findByUsers(users);
                carts.stream().forEach(x ->
cartDao.deleteById(x.getCartId()));
            }

            orderDetailDao.save(orderDetail);
        }
    }

    public void markOrderAsDelivered(Integer orderId) {
        OrderDetail orderDetail =
orderDetailDao.findById(orderId).get();

        if(orderDetail != null) {
            orderDetail.setOrderStatus("Delivered");
            orderDetailDao.save(orderDetail);
        }
```

```
    }

    public TransactionDetails createTransaction(Double amount) {
        try {

            JSONObject jsonObject = new JSONObject();
            jsonObject.put("amount", (amount * 100) );
            jsonObject.put("currency", CURRENCY);

            RazorpayClient razorpayClient = new
RazorpayClient(KEY, KEY_SECRET);

            Order order =
razorpayClient.orders.create(jsonObject);

            TransactionDetails transactionDetails =
prepareTransactionDetails(order);
            return transactionDetails;
        } catch (Exception e) {
            System.out.println(e.getMessage());
        }
        return null;
    }

    private TransactionDetails prepareTransactionDetails(Order
order) {
        String orderId = order.get("id");
        String currency = order.get("currency");
        Integer amount = order.get("amount");

        TransactionDetails transactionDetails = new
TransactionDetails(orderId, currency, amount, KEY);
        return transactionDetails;
```

```
    }
}
```

Файл RoleService.java

```
package com.ecommerce.service;

import com.ecommerce.dao.RoleDao;
import com.ecommerce.entity.Role;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

@Service
public class RoleService {

    @Autowired
    private RoleDao roleDao;

    public Role createNewRole(Role role) {
        return roleDao.save(role);
    }
}
```

Файл UserService.java

```
@Service
public class UserService {

    @Autowired
    private UserDao userDao;

    @Autowired
    private RoleDao roleDao;
```

```
    @Autowired
    private PasswordEncoder passwordEncoder;

    public void initRoleAndUser() {

        Role adminRole = new Role();
        adminRole.setRoleName("Admin");
        adminRole.setRoleDescription("Admin role");
        roleDao.save(adminRole);

        Role userRole = new Role();
        userRole.setRoleName("User");
        userRole.setRoleDescription("Default role for newly
created record");
        roleDao.save(userRole);

        Users adminUsers = new Users();
        adminUsers.setUserName("admin123");

adminUsers.setUserPassword(getEncodedPassword("admin@pass"));
        adminUsers.setUserFirstName("admin");
        adminUsers.setUserLastName("admin");
        Set<Role> adminRoles = new HashSet<>();
        adminRoles.add(adminRole);
        adminUsers.setRole(adminRoles);
        userDao.save(adminUsers);

        Users users = new Users();
        users.setUserName("raj123");
        users.setUserPassword(getEncodedPassword("raj@123"));
        users.setUserFirstName("raj");
        users.setUserLastName("sharma");
```

```
        Set<Role> userRoles = new HashSet<>();
        userRoles.add(userRole);
        users.setRole(userRoles);
        userDao.save(users);
    }

    public Users registerNewUser(Users users) {
        Role role = roleDao.findById("User").get();
        Set<Role> userRoles = new HashSet<>();
        userRoles.add(role);
        users.setRole(userRoles);

users.setUserPassword(getEncodedPassword(users.getUserPassword()
));

        return userDao.save(users);
    }

    public String getEncodedPassword(String password) {
        return passwordEncoder.encode(password);
    }
}
```

Файл JwtUtil.java

```
@Component
public class JwtUtil {

    private static final String SECRET_KEY =
"learn_programming_yourself";

    private static final int TOKEN_VALIDITY = 3600 * 5;

    public String getUsernameFromToken(String token) {
```

```
        return getClaimFromToken(token, Claims::getSubject);
    }

    public <T> T getClaimFromToken(String token,
Function<Claims, T> claimsResolver) {
        final Claims claims = getAllClaimsFromToken(token);
        return claimsResolver.apply(claims);
    }

    private Claims getAllClaimsFromToken(String token) {
        return
Jwts.parser().setSigningKey(SECRET_KEY).parseClaimsJws(token).ge
tBody();
    }

    public Boolean validateToken(String token, UserDetails
userDetails) {
        final String username = getUsernameFromToken(token);
        return (username.equals(userDetails.getUsername()) &&
!isTokenExpired(token));
    }

    private Boolean isTokenExpired(String token) {
        final Date expiration =
getExpirationDateFromToken(token);
        return expiration.before(new Date());
    }

    public Date getExpirationDateFromToken(String token) {
        return getClaimFromToken(token, Claims::getExpiration);
    }

    public String generateToken(UserDetails userDetails) {
```

```
        Map<String, Object> claims = new HashMap<>();


        return Jwts.builder()
                .setClaims(claims)
                .setSubject(userDetails.getUsername())
                .setIssuedAt(new
Date(System.currentTimeMillis()))
                .setExpiration(new
Date(System.currentTimeMillis() + TOKEN_VALIDITY * 1000))
                .signWith(SignatureAlgorithm.HS512, SECRET_KEY)
                .compact();
    }
}
```

Приложение В

(обязательное)

Web – портал маркетплейс - агрегатора

Презентация

Тема работы:
Web – портал
маркетплейс - агрегатора

Автор работы: студент группы 19ВВП1 Юдин Д. А.

Научный руководитель: к.т.н., доцент каф. ВТ ПГУ Дубравин

Пенза 2023

Актуальность работы

В России маркетплейсы показывают значительный рост. В 2022 году, несмотря на ожидаемый спад в силу уходов многих брендов, оборот данного сегмента интернет – торговли составил 2.7 трлн. рублей, увеличившись на 80%.

Ежедневно миллионы людей предпочитают просматривать и оформлять онлайн сотни и тысячи товаров через различные интернет-магазины.

Вышеописанное демонстрируют нам актуальность выбранного в рамках выпускной квалификационной работы проекта.

Предмет и объект работы

Предмет

- Разработка web – портала маркетплейс - агрегатора

Объект

- Разработка клиент – серверного web - приложения

Цель работы

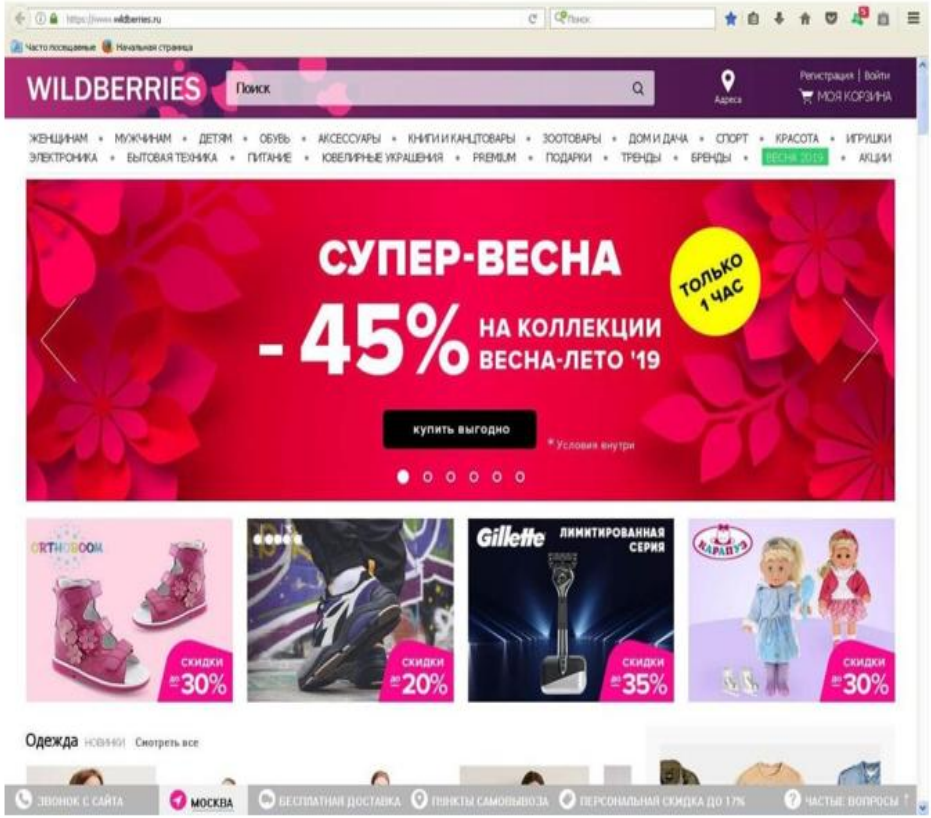
Цель работы – разработать web – портал маркетплейс – агрегатора. Данное приложение позволит администратору просматривать, загружать и редактировать описание продукции на сайте, а также менять статус заказа, потребителю же позволит просматривать заказы, а также оформлять и оплачивать их

Постановка задачи

Данное приложение должно быть спроектировано на основе модели клиент - серверной архитектуры с использованием паттернов *MVC* и *REST*.

Клиентская часть должна обладать визуальным графическим интерфейсом, а серверная должна обрабатывать запросы и работать с базой данных.

Аналоги
Wildberries



Стек технологий

Диаграммы развертывания

Диаграмма прецедентов (администратор)

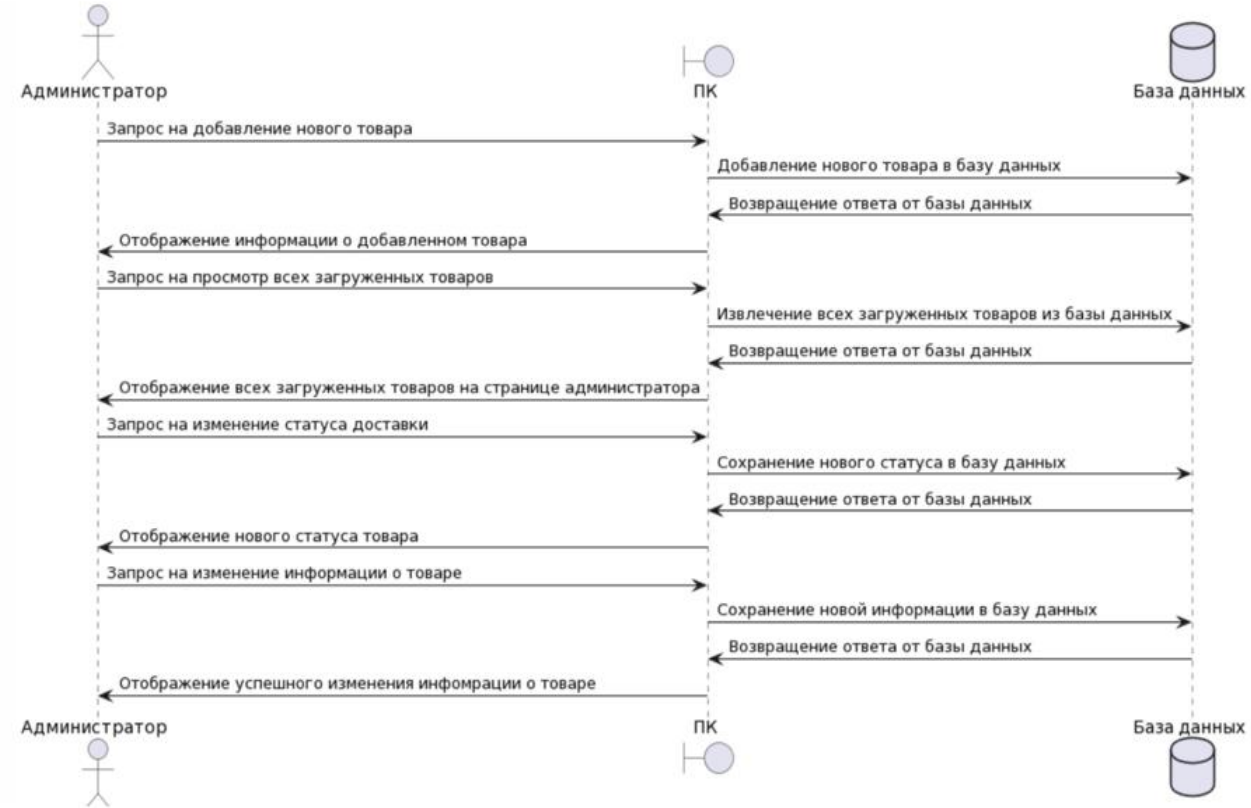
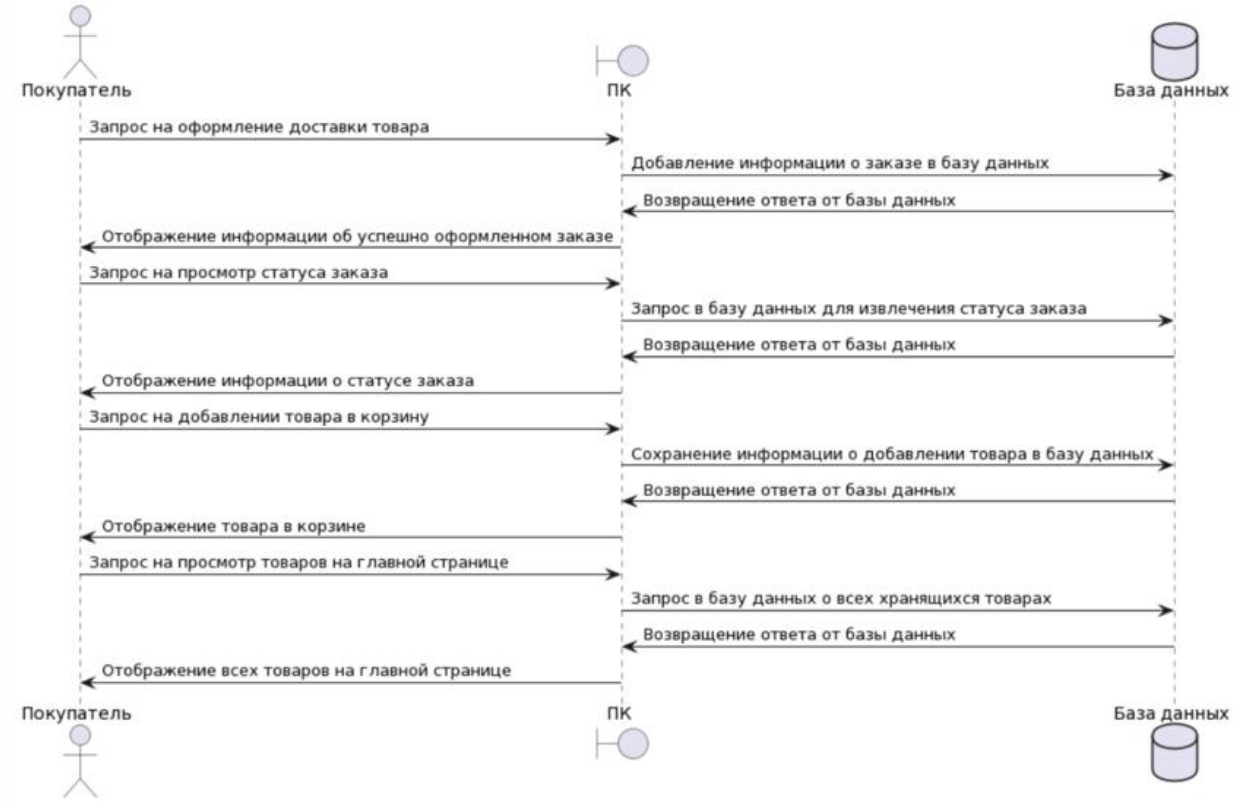
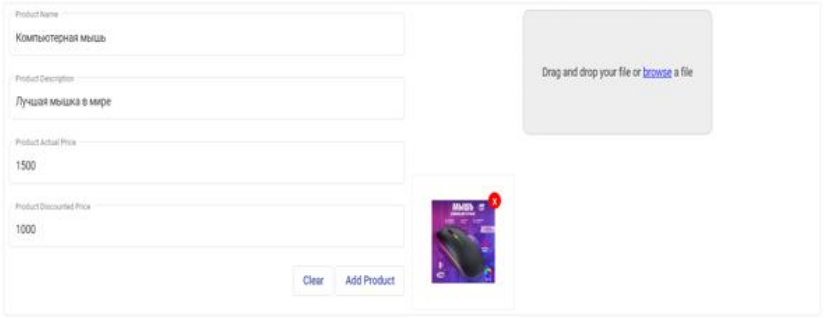


Диаграмма прецедентов (покупатель)



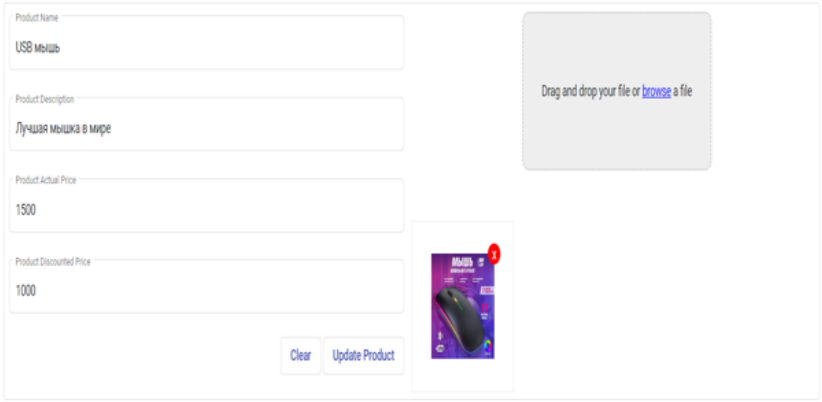
Демонстрация работы



Добавим товар на сайт

Товар был успешно добавлен на главную страницу сайта

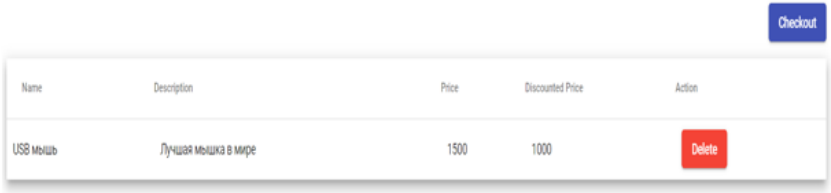
Демонстрация работы



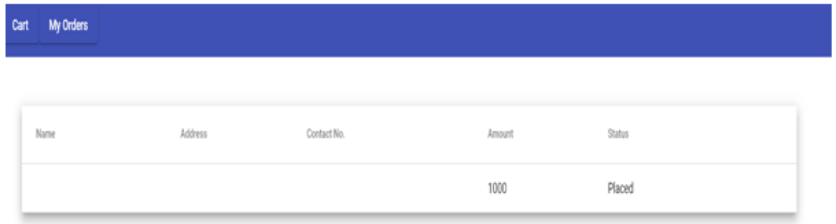
Меняем название товара в разделе редактирования

Название товара было успешно изменено

Демонстрация работы



Товар был успешно добавлен в корзину покупателем



Заказ был успешно оформлен

Заключение

В результате дипломной работы была разработана *MVP* – версия *web* – портала маркетплейс – агрегатора, реализующая функционал в соответствии с техническим заданием.

В процессе разработки программного обеспечения были изучены и применены на практике технологии: *Spring Framework*, *Maven*, *PostgreSQL*. Выработан навык работы с интегрированной средой разработки. Были освоены современные практики и парадигмы проектирования web – приложений, такие как *REST*, *MVC*, передача по *http* и т. д., получены навыки разработки *API*. Для разработки клиентской части приложения применялся фреймворк *AngularJS*.

На следующем этапе разработки данного проекта предполагается: добавить на сайт необходимые фильтры и сортировки, категории товаров, сделать отдельную роль продавцов для добавления товаров.

Спасибо за внимание!