



МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение  
высшего образования

«МИРЭА – Российский технологический университет»

РТУ МИРЭА

Институт информационных технологий (ИТ)

Кафедра инструментального и прикладного программного обеспечения (ИиППО)

### КУРСОВАЯ РАБОТА

по дисциплине: Шаблоны программных платформ языка Java

по профилю: Разработка программных продуктов и проектирование информационных систем

направления профессиональной подготовки: 09.03.04 «Программная инженерия»

Тема: Приложение «Магазин футбольной атрибутики»

Студент: Белослудцев Егор Денисович

Группа: ИКБО-16-21

Работа представлена к защите 13.06.23 (дата) [подпись] / Белослудцев Е. Д. /  
(подпись и ф.и.о. студента)

Руководитель: Андрей Владимирович Рачков, старший преподаватель

Работа допущена к защите 13.06.23 (дата) [подпись] / Рачков А. В. /  
(подпись и ф.и.о. рук-ля)

Оценка по итогам защиты: 5,0

13.06.23 [подпись] 18.03 к.т.н. Кузнецов

13.06.23 [подпись] ст. преп. Зорин Н.В.

(подписи, дата, ф.и.о., должность, звание, уч. степень двух преподавателей, принявших защиту)

М. РТУ МИРЭА. 2023 г.



МИНОБРНАУКИ РОССИИ  
Федеральное государственное бюджетное образовательное учреждение  
высшего образования  
«МИРЭА – Российский технологический университет»  
**РТУ МИРЭА**

**Институт информационных технологий (ИТ)**  
**Кафедра инструментального и прикладного программного обеспечения (ИиППО)**

**ЗАДАНИЕ**  
**на выполнение курсовой работы**

по дисциплине: Шаблоны программных платформ языка Джава  
по профилю: Разработка программных продуктов и проектирование информационных систем  
направления профессиональной подготовки: Программная инженерия (09.03.04)

Студент: Белослудцев Егор Денисович

Группа: ИКБО-16-21

Срок представления к защите: 11.05.2023 г.

Руководитель: старший преподаватель Рачков Андрей Владимирович

**Тема:** Приложение «Магазин футбольной атрибутики»

**Исходные данные:** индивидуальное задание на разработку; документация по Spring Framework и JEE, документация по языку Java (версия не ниже 8); инструменты и технологии: JDK (не ниже 8), создание Spring MVC web-приложений, RESTful web-сервисов, Spring ORM и Spring DAO, Gradle, gitHub, IntelliJIDEA. Нормативный документ: инструкция по организации и проведению курсового проектирования СМКО МИРЭА 7.5.1/04.И.05-18.

**Перечень вопросов, подлежащих разработке, и обязательного графического материала:**  
1. Провести анализ предметной области и формирование основных требований к приложению.  
2. Обосновать выбор средств ведения разработки. 3. Разработать приложение с использованием фреймворка Spring, выбранной технологии и инструментария. 4. Провести тестирование приложения. 5. Оформить пояснительную записку по курсовой работе в соответствии с ГОСТ 7.32-2017. 6. Провести анализ текста на антиплагиат 7. Создать презентацию по выполненной курсовой работе.

Руководителем произведён инструктаж по технике безопасности, противопожарной технике и правилам внутреннего распорядка.

Зав. кафедрой ИиППО: Р. Г. Болбаков, « 20 » 02 2023 г.

Задание на КР выдал: А.В. Рачков, « 20 » 02 2023 г.

Задание на КР получил: Е.Д. Белослудцев, « 20 » 02 2023 г.

## РЕФЕРАТ

Проект курсовой работы содержит 50 страниц отчета, 41 иллюстраций, 1 таблицу, 15 информационных источников.

Ключевые слова: МАГАЗИН ФУТБОЛЬНОЙ АТТРИБУТИКИ, УПРАВЛЕНИЕ, JAVA, SPRING, SPRING SECURITY, SPRING MVC.

Объектом исследования данной работы является разработка программного приложения для автоматизации работы магазина футбольной атрибутики.

Цель работы – разработать серверное приложение для интернет-магазина футбольной атрибутики на языке Java с использованием фреймворка Spring.

В процессе работы проведен анализ предметной области и сайтов с подобной тематикой.

В результате работы разработано приложение, которое позволяет управлять заказами, продуктами и клиентской базой магазина футбольной атрибутики. Новизной данной работы является использование фреймворка Spring для упрощения процесса разработки и увеличения надежности приложения.

The project of the course work contains 46 pages of the report, 54 illustrations, 1 table, 15 information sources.

Keywords: FOOTBALL EQUIPMENT SHOP, MANAGEMENT, JAVA, SPRING, SPRING SECURITY, SPRING MVC.

The object of study of this work is the development of a software application for automating the work of a football paraphernalia store.

The purpose of the work is to develop a server application for an online store of football paraphernalia in Java using the Spring framework.

In the process of work, an analysis of the subject area and sites with similar topics was carried out.

As a result of the work, an application was developed that allows you to manage orders, products and the customer base of a football merchandise store. The novelty of this work is the use of the Spring framework to simplify the development process and increase the reliability of the application.

## СОДЕРЖАНИЕ

ПЕРЕЧЕНЬ СКОРАЩЕНИЙ.....	5
ВВЕДЕНИЕ .....	6
1 ОБЩИЕ СВЕДЕНИЯ .....	7
1.1 Обозначение и наименование интернет-ресурса .....	7
1.2 Основные преимущества использования Spring .....	7
2 ОПИСАНИЕ ЛОГИЧЕСКОЙ СТРУКТУРЫ .....	8
2.1 Анализ предметной области .....	8
2.2 Выбор технологии разработки приложения .....	10
3 РЕАЛИЗАЦИЯ ПРИЛОЖЕНИЯ .....	12
3.1 Использование библиотеки Lombok .....	12
3.2 Использование технологий Spring Boot .....	12
3.3 Создание моделей данных .....	13
3.3.1 Класс «Customer» .....	13
3.3.2 Класс «Order» .....	14
3.3.3 Класс «OrderItem» .....	15
3.3.4 Класс «Product» .....	16
3.3.5 Класс «ProductImage» .....	17
3.4 Создание JPA репозиторий для работы с базой данных .....	18
3.5 Создание сервисов для реализации логики работы приложения .....	20
3.5.1 Класс «CartService» .....	20
3.5.2 Класс «ProductImageService» .....	21
3.5.3 Класс «ProductService» .....	22
3.6 Использование Spring MVC для реализации веб-интерфейса .....	23
3.6.1 Класс «AdminController» .....	23
3.6.2 Класс «AuthenticationController» .....	25
3.6.3 Класс «CartController» .....	26
3.7 Использование паттерна DTO .....	28
3.8 Использование библиотеки-маппера .....	29
3.9 Использование Spring Security .....	31
4 ТЕСТИРОВАНИЕ ПРИЛОЖЕНИЯ .....	34
4.1 Тестирование приложения со стороны пользователя .....	34

4.2 Тестирование приложения с помощью Postman.....	41
ЗАКЛЮЧЕНИЕ .....	46
СПИСОК ИНФОРМАЦИОННЫХ ИСТОЧНИКОВ.....	48

## ПЕРЕЧЕНЬ СКОРАЩЕНИЙ

RESTful API	—	Representational State Transfer API
СУБД	—	Система Управления Базами Данных
JPA	—	Java Persistence API
JSON	—	JavaScript Object Notation
MVC	—	Model-View-Controller
REST	—	Representational State Transfer
AOP	—	Aspect-Oriented Programming
API	—	Application Programming Interface
CRUD	—	Create, Read, Update, Del
DTO	—	Data Transfer Object
DI	—	Dependency Injection
IoC	—	Inversion of Control
XML	—	International Air Transport Association

## **ВВЕДЕНИЕ**

В настоящее время информационные технологии широко используются в различных сферах деятельности, включая торговлю. Одним из наиболее востребованных видов торговли является продажа футбольной атрибутики. В связи с этим возникает потребность в создании удобных и эффективных программных решений для управления бизнес-процессами в магазинах футбольной атрибутики.

С каждым годом число интернет-магазинов увеличивается, поскольку это выгодно для владельцев и удобно для покупателей. Интернет-магазин может работать круглосуточно и осуществлять продажу товаров в автоматическом режиме без участия продавца, что позволяет сэкономить время покупателя.

Цель данной курсовой работы - разработать серверное приложение для интернет-магазина футбольной атрибутики на языке Java с использованием фреймворка Spring[1]. Для достижения данной цели были поставлены следующие задачи:

- 1) изучить основы фреймворка Spring, включая Spring Data JPA[2], Spring Security, Spring Core и Spring MVC;
- 2) провести анализ требований к серверному приложению;
- 3) спроектировать и разработать серверное приложение;
- 4) провести тестирование разработанного приложения.

Объектом исследования данной курсовой работы является серверное приложение для интернет-магазина футбольной атрибутики, а предметом исследования - применение фреймворка Spring при разработке данного

При разработке проекта использовались знания, полученные в ходе лекционных и практических занятий по курсу «Шаблоны программных платформ языка Java», а также дополнительные литературные источники, связанные с разработкой интернет-магазинов и футбольной атрибутики.

## **1 ОБЩИЕ СВЕДЕНИЯ**

### **1.1 Обозначение и наименование интернет-ресурса**

Темой разработанного приложения является «Магазин футбольной атрибутики». Разработанное приложение для данного магазина получило название "FootballMart".

### **1.2 Основные преимущества использования Spring**

Spring является одним из самых популярных фреймворков для создания приложений на языке Java. Он предоставляет ряд преимуществ, которые делают его очень удобным для разработки приложений любой сложности:

1) инверсия контроля и внедрение зависимостей. Spring позволяет реализовать инверсию контроля и внедрение зависимостей, что делает код более гибким, уменьшает зависимости между классами и упрощает тестирование приложения;

2) Spring Framework предоставляет возможность легко подключать новые библиотеки и компоненты, что позволяет разрабатывать более сложные и функциональные приложения;

3) архитектура Spring Framework основана на модульности, что позволяет использовать только необходимые модули и библиотеки, что делает приложение более легким и быстрым;

4) Spring обладает высокой производительностью благодаря использованию компонентов, таких как Spring Boot, Spring Data JPA и Spring Security;



5) с помощью Spring Framework можно легко создавать RESTful API и микросервисы.

Использование Spring может значительно ускорить процесс разработки и улучшить качество кода.

## 2 ОПИСАНИЕ ЛОГИЧЕСКОЙ СТРУКТУРЫ

### 2.1 Анализ предметной области

Для создания серверного приложения для интернет-магазина футбольной атрибутики проведен анализ трех сайтов со схожей тематикой:

- <https://gnk-store.ru> – Gloves n' Kit
- <https://shop.premier-football.ru> – Премьер футбол
- <https://olimpijka.ru> – Olimpijka

Результаты анализа приведены в таблице 1.

Таблица 1 – Результат анализа сервисов со схожей тематикой.

Критерий	Gloves n' Kit	Премьер футбол	Olimpijka	Вывод
Ассортимент товаров	Сервис предоставляет огромный ассортимент товаров, благодаря которому даже самые придирчивые клиенты смогут найти товар, соответствующий их высоким требованиям.	Данный интернет магазин также удивляет приятный набор товаров и их разнообразием. Здесь вы сможете приобрести начиная от носков до целых спортивных комплектов, предназначенные для большого коллектива.	Olimpijka может обрадовать большим выбором товаров, которые необходимы для каждого клиента, интересующего таким видом спорта, как футбол. Также здесь есть функции приобретения	Реализуемый сервис должен иметь достаточный ассортимент.

			подарочных сертификатов.	
Цены	Сервер имеет средний ценовой диапазон, а также множество скидок, которые помогут клиентам сэкономить на некоторых товарах.	Более доступные цены и приятные условия доставки товаров.	Основной каталог имеет довольно высокие ценовой порог, но также присутствует множество товаров по более низким ценам.	Сервисное приложение должно предлагать более доступные цены.
Удобство использования сайта	Gloves n' Kit имеет простой и интуитивно понятный дизайн.	Данный сайт также имеет простой и понятный, но плохо адаптированный дизайн.	Имеет менее удобный дизайн сайта.	Сервисное приложение должно иметь понятный и удобный интерфейс.

Продолжение таблицы 1

Информативность сайта	Чтобы узнать подробную информацию по каждому товару необходимо перейти во вкладку описание, где клиент сможет максимально подробно	Премьер футбол предоставляет широкий ассортимент товаров, но зачастую они не играют данному сайту на руку. Некоторые	Имеет менее информативный сайт, где описания товаров более краткие.	Разрабатываемое сервисное приложение должно быть информативно для пользователя.
-----------------------	------------------------------------------------------------------------------------------------------------------------------------	----------------------------------------------------------------------------------------------------------------------	---------------------------------------------------------------------	---------------------------------------------------------------------------------

	ознакомиться с содержанием выбранного товара.	страницы переполнены ненужной информацией, которая мешает фокусировке на действительно важной информации.		
Качество услуг	Качество услуг, предоставляемых находится на высоком уровне.	Качество услуг находится на среднем уровне.	Высокое качество предоставляемых услуг, также присутствуют отзывы к каждому товару.	Сервисное приложено предоставлять услуги высокого качества.

Таким образом, разрабатываемый сервис должен иметь достаточный ассортимент, более доступные цены, понятный и удобный интерфейс, быть информативным для пользователя и предоставлять услуги высокого качества.

## 2.2 Выбор технологии разработки приложения

Для разработки приложения использовалась JDK (Java Development Kit)[3] – набор инструментов для разработки приложений на Java, включая компилятор, отладчик и другие утилиты.

В качестве объектно-реляционного отображения (ORM)[4] использовался фреймворк Hibernate, который предоставляет программистам возможность работать с базами данных в объектно-ориентированной манере. Hibernate упрощает работу с базами данных, скрывая от разработчика детали взаимодействия с БД, такие как создание SQL-запросов и управление транзакциями.

В качестве языка программирования выбран Java, так как он является одним из наиболее популярных языков программирования, имеет обширную документацию и сообщество разработчиков, а также обладает хорошей производительностью и безопасностью.

Для реализации функционала приложения использован фреймворк Spring, который является одним из наиболее популярных фреймворков для разработки серверных приложений на языке Java. Spring предоставляет широкие возможности для управления жизненным циклом[5] объектов, интеграции с другими технологиями и фреймворками, а также обеспечивает высокую производительность и безопасность приложения.

В частности, для работы с базой данных использован модуль Spring Data Jpa, который позволяет упростить работу с базой данных и снизить количество кода. Spring Data Jpa обеспечивает поддержку различных баз данных и предоставляет удобные методы для выполнения запросов к базе данных.

Серверная часть приложения реализуется с помощью фреймворка Spring Boot, который предоставляет готовую инфраструктуру для создания веб-приложений на Java. Фреймворк обладает рядом преимуществ, таких как упрощение создания сложных приложений, повышение безопасности и производительности приложения, использование Dependency Injection (DI)[6] / Inversion of Control (IoC)[7] для лучшей модульности и тестирования, широкий выбор инструментов для работы с базами данных и активное сообщество разработчиков, которое предоставляет обширную документацию и учебные материалы для пользователей фреймворка.

Для обеспечения безопасности приложения использован модуль Spring Security, который предоставляет возможности для аутентификации и авторизации пользователей. Spring Security позволяет защитить приложение от различных видов атак и обеспечить безопасность передачи данных.

Для взаимодействия между серверной и клиентской частями приложения был использован протокол HTTP, а также формат передачи данных JSON. JSON является универсальным форматом, который поддерживается большинством

языков программирования и позволяет передавать данные в удобном для обработки виде.

Среди возможного множества редакторов кода выбран редактор исходного кода IntelliJ IDEA. Интегрированная среда разработки программного обеспечения для многих языков программирования, в частности Java, JavaScript, Python, разработанная компанией JetBrains.

Таким образом, использование фреймворка Spring и его модулей позволило значительно упростить и ускорить разработку серверного программного приложения для, а также обеспечить его высокую производительность и безопасность.

### **3 РЕАЛИЗАЦИЯ ПРИЛОЖЕНИЯ**

#### **3.1 Использование библиотеки Lombok**

Для облегчения разработки и поддержки кода использована библиотека Lombok. Lombok — это библиотека для Java, которая помогает уменьшить объем написания шаблонного кода. Она предоставляет аннотации, которые автоматически генерируют конструкторы, методы доступа и т.д. для классов, что упрощает разработку и поддержку кода. Кроме этого, Lombok предоставляет аннотацию @Builder для создания объектов с помощью паттерна Builder.

#### **3.2 Использование технологий Spring Boot**

В разработанном приложении использовался паттерн слоистой архитектуры. В Spring слоистая архитектура обычно реализуется следующим образом.

Представление (Presentation) – в Spring этот слой реализован с помощью контроллеров (Controllers). Они обрабатывают запросы от клиента и формируют ответы. Контроллеры используются для взаимодействия с пользователем и получения/обработки данных.

Бизнес-логика (Business Logic) – в Spring этот слой реализован с помощью сервисов (Services). Сервисы содержат бизнес-логику, которая выполняет операции с данными и производит вычисления.

Доступ к данным (Data Access) – в Spring этот слой реализован с помощью репозиторий (Repositories). Репозитории обеспечивают доступ к базам данных и другим источникам данных. Они реализуют CRUD-операции (Create, Read, Update, Delete) и обеспечивают хранение и извлечение данных.

Инфраструктура (Infrastructure) – в Spring этот слой отвечает за настройку и конфигурирование приложения. Он может содержать компоненты для обработки исключений, логирования, аутентификации и т.д.

Spring также предоставляет много инструментов для упрощения реализации слоистой архитектуры, таких как Dependency Injection (DI), Aspect-Oriented Programming (AOP)[8], Spring Data JPA и другие. Эти инструменты позволяют создавать чистый и модульный код, который легко поддерживать и расширять в дальнейшем. Для абстрагирования от сложностей взаимодействия с базой данных напрямую и работать с объектами на более высоком уровне абстракции использованы компоненты сущности, такие компоненты помечаются аннотацией «Entity». В процессе разработки созданы следующие сущности: Customer, Order, OrderItem, OrderStatus, Product, ProductImage.

### **3.3 Создание моделей данных**

#### **3.3.1 Класс «Customer»**

Класс Customer является сущностью, описывающей таблицу с информацией о покупателях. Данная сущность связана с сущностью Image связью один ко многим и с сущностью CartItem связью один ко многим – см. рисунок 3.3.1.1.

Аннотация @Entity указывает, что класс Customer является сущностью, которая будет отображена на таблицу в базе данных. Аннотация @Table указывает, что таблица в базе данных будет называться "customer".

```

@Entity
@RequiredArgsConstructor
@AllArgsConstructor
@FieldDefaults(level = AccessLevel.PRIVATE)
public class Customer {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    long id;

    String name;
    String email;
    String password;

    String role;

    3 usages
    @ManyToMany(mappedBy = "customerList")
    private List<Product> cart;

    3 usages
    @OneToMany(mappedBy = "owner")
    private List<Order> orders;

```

Рисунок 3.3.1.1 – Java код класса Customer

Аннотация `@Data` является Lombok аннотацией, которая генерирует методы геттеров, сеттеров, `toString`, `equals`, `hashCode`, что уменьшает количество шаблонного кода.

Аннотация `@Id` указывает, что поле "id" является первичным ключом таблицы. Аннотация `@GeneratedValue` указывает, что значение поля будет генерироваться автоматически.

### 3.3.2 Класс «Order»

Класс Order представляет собой корзину в интернет-магазине, в которую клиент может добавлять различные товары. Корзина имеет следующие свойства: адрес доставки (`address`), дата создания (`creationDate`), контактный телефон (`phone`), статус заказа (`status`) и общую сумму заказа (`total`). Корзина также содержит список элементов заказа (`orderItems`), которые связаны с данным

заказом, и владельца заказа (owner), который является клиентом. Клиент может добавлять и удалять товары из корзины. Код сущности представлен на рисунке 3.3.2.1.

```
@FieldDefaults(level = AccessLevel.PRIVATE)
@Table(name = "order")
public class Order {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    long id;

    String address;
    LocalDate creationDate;
    String phone;
    OrderStatus status;
    Float total;

    @OneToMany(mappedBy = "order")
    private List<OrderItem> orderItems;

    @ManyToOne
    @JoinColumn(name = "customer_id", referencedColumnName = "id")
    private Customer owner;
}
```

Рисунок 3.3.2.1 – Java код класса Order

### 3.3.3 Класс «OrderItem»

Класс OrderItem предназначен для хранения информации о товарах в корзине. Когда товар добавляется в корзину, он становится объектом класса OrderItem - рисунок 3.3.3.1.

Аннотация @Builder генерирует паттерн Builder для класса OrderItem, что позволяет создавать объекты с более удобным синтаксисом.

Аннотация @NoArgsConstructor генерирует конструктор без аргументов для класса OrderItem.

Аннотация @AllArgsConstructor генерирует конструктор, принимающий все поля класса OrderItem.



```

@Entity
@Data
@Builder
@NoArgsConstructor
@AllArgsConstructor
public class OrderItem {

    @Id
    @GeneratedValue(strategy = GenerationType.SEQUENCE)
    private long id;

    private Short quantity;

    @ManyToOne
    @JoinColumn(name = "order_id")
    private Order order;

    @OneToOne
    @JoinColumn(name = "product_id")
    private Product product;

}

```

Рисунок 3.3.3.1 – Java код класса CartItem

### 3.3.4 Класс «Product»

Класс Product представляет сущность товара в интернет-магазине. Каждый товар имеет уникальный идентификатор (id), название (name), описание (description), категорию (category) и цену (price) - рисунок 3.3.4.1.

Аннотация `@FieldDefaults(level = AccessLevel.PRIVATE)` от Lombok устанавливает уровень доступа private для всех полей класса по умолчанию.

Поле `images` представляет собой список изображений товара (`ProductImage`), связанных с данным товаром. Оно объявлено с аннотацией `@OneToMany`, которая указывает на связь "один-ко-многим" между классом Product и ProductImage. `mappedBy = "product"` указывает, что связь управляется полем `product` в классе ProductImage.

Поля `customerList` и `orders` представляют списки клиентов (`Customer`) и заказов (`Order`), связанных с данным товаром. Они объявлены с аннотацией `@ManyToMany`, что указывает на связь "многие-ко-многим" между классами. Аннотация `@JoinTable` указывает на промежуточную таблицу, которая связывает товары с клиентами (`cart`) и заказами (`ordered_product`). С помощью аннотаций `@JoinColumn` и `inverseJoinColumns` указываются столбцы, которые представляют связи между таблицами.

```
@FieldDefaults(level = AccessLevel.PRIVATE)
public class Product {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    long id;

    String name;
    String description;
    String category;

    int price;

    @OneToMany(mappedBy = "product")
    List<ProductImage> images;

    3 usages
    @ManyToMany
    @JoinTable(name = "cart",
        joinColumns = @JoinColumn(name = "product_id"),
        inverseJoinColumns = @JoinColumn(name = "customer_id"))
    List<Customer> customerList;

    2 usages
    @ManyToMany
    @JoinTable(name = "ordered_product",
        joinColumns = @JoinColumn(name = "product_id"),
        inverseJoinColumns = @JoinColumn(name = "order_id"))
    private List<Order> orders;
```

Рисунок 3.3.4.1 – Java код класса `Product`

### 3.3.5 Класс «`ProductImage`»

Класс `ProductImage` представляет изображение товара в интернет-магазине. Каждое изображение имеет уникальный идентификатор и URL-адрес,

указывающий на расположение изображения – рисунок 3.3.5.1. Поле product представляет связь "многие-к-одному" с классом Product. Оно объявлено с аннотацией @ManyToOne, что указывает на связь, где много изображений (ProductImage) могут быть связаны с одним товаром (Product). Аннотация @JoinColumn указывает на столбец, который представляет связь между таблицами. В данном случае, связь устанавливается через столбец product\_id.

```
@Entity
@Data
@Builder
@NoArgsConstructor
@AllArgsConstructor
@FieldDefaults(level = AccessLevel.PRIVATE)
public class ProductImage {
    @Id
    @GeneratedValue(strategy = GenerationType.SEQUENCE)
    Long id;
    String url;

    @ManyToOne
    @JoinColumn(name = "product_id")
    Product product;
}
```

Рисунок 3.3.5.1 – Java код сущности Image

### 3.4 Создание JPA репозитория для работы с базой данных

Репозитории в Java Spring являются частью слоя доступа к данным (Data Access Layer)[9] и предназначены для управления объектами, хранимыми в базе данных. Они предоставляют высокоуровневый API для выполнения операций CRUD и запросов к базе данных.

В данном примере представлен класс репозитория CustomerRepository, который расширяет интерфейс JpaRepository и работает с сущностями класса Customer. Аннотация @Repository указывает, что данный интерфейс является компонентом Spring, отвечающим за доступ к данным.

Метод findByEmail(String email) объявлен в интерфейсе CustomerRepository для поиска объекта Customer по электронной почте. Он

возвращает `Optional<Customer>`, что позволяет избежать возможной `NullPointerException` при отсутствии совпадений.

Использование интерфейса `JpaRepository`[\[10\]](#) позволяет автоматически генерировать SQL-запросы на основе сигнатур методов. Это значительно упрощает кодирование запросов и сокращает время разработки.

Класс `CustomerRepository` имеет типовой параметр `<Customer, Long>`, где `Customer` - это тип сущности, а `Long` - тип первичного ключа сущности - рисунок 3.4.1.

```
@Repository
public interface CustomerRepository extends JpaRepository<Customer, Long> {
    2 usages
    Optional<Customer> findByEmail(String email);
}
```

Рисунок 3.4.1 – Java код компонента `CustomerRepository`

Класс `ProductImageRepository` является репозиторием для работы с объектами класса `ProductImage` - рисунок 3.4.2. Он расширяет интерфейс `JpaRepository`, предоставляющий методы для выполнения операций CRUD и запросов к базе данных. Данный репозиторий использует тип `Long` в качестве первичного ключа и обеспечивает абстракцию от конкретных технологий хранения данных.

```
public interface ProductImageRepository extends JpaRepository<ProductImage, Long> {
}
```

Рисунок 3.4.2 – Java код компонента `ProductImageRepository`

Класс `ProductRepository` также является репозиторием, но для работы с объектами класса `Product` - рисунок 3.4.3. Он также расширяет интерфейс `JpaRepository` и предоставляет методы для выполнения операций CRUD и запросов к базе данных для объектов `Product`. Подобно `ProductImageRepository`, он использует тип `Long` в качестве первичного ключа и обладает преимуществами абстракции от конкретных технологий хранения данных.

```
@Repository
public interface ImageRepo extends JpaRepository<Image, Long>{
}
```

Рисунок 3.4.3 – Java код компонента ProductRepository

Оба репозитория обрабатывают транзакции и автоматически генерируют SQL-запросы на основе сигнатур методов, что упрощает разработку приложений, снижает затраты на поддержку и обновление кода, и улучшает производительность.

### 3.5 Создание сервисов для реализации логики работы приложения

#### 3.5.1 Класс «CartService»

Класс CartService представляет сервисный компонент, отвечающий за управление корзиной покупок. Он обеспечивает операции по поиску, добавлению и удалению товаров в корзине, а также проверку наличия товара в корзине - рисунок 3.5.1.1.

```
@Service
@RequiredArgsConstructor
public class CartService {

    4 usages
    private final CustomerRepository customerRepository;

    2 usages
    private final ProductRepository productRepository;

    1 usage
    public List<Product> findCartByPerson(long customerId) {
        Customer customer = customerRepository.findById(customerId).orElse( other: null);
        Hibernate.initialize(customer.getCart());
        return customer.getCart();
    }

    1 usage
    @Transactional
    public void addItemToCart(long customerId, long productId) {
        Customer customer = customerRepository.findById(customerId).orElse( other: null);
        Product product = productRepository.findById(productId).orElse( other: null);

        customer.addToCart(product);
        product.addToCart(customer);
    }
}
```

Рисунок 3.5.1.1 – Java код класса CartService

В конструкторе класса CartService происходит внедрение зависимостей CustomerRepository и ProductRepository, которые необходимы для доступа к данным о клиентах и продуктах.

Метод findCartByPerson используется для поиска товаров в корзине по идентификатору клиента. Он загружает данные о клиенте и инициализирует его корзину, чтобы обеспечить доступ к товарам.

В целом, класс CartService предоставляет удобный интерфейс для работы с корзиной покупок, используя функциональность репозитория для доступа к данным и обеспечивая транзакционность операций.

### **3.5.2 Класс «ProductImageService»**

Класс ProductImageService является сервисным компонентом, отвечающим за управление изображениями товаров. Он предоставляет методы для сохранения изображений товаров, связанных с определенным продуктом - рисунок 3.5.2.1.

В конструкторе класса ProductImageService происходит внедрение зависимостей ProductImageRepository, ProductRepository и ImageNameService, которые необходимы для доступа к данным о изображениях товаров, продуктах и генерации имени файла изображения.

Метод save выполняет сохранение изображения товара. Он получает идентификатор продукта и объект MultipartFile, содержащий информацию о загруженном файле изображения. Сначала происходит поиск продукта по его идентификатору с использованием ProductRepository. Если продукт не найден, выбрасывается исключение ProductNotFoundException. Затем генерируется уникальное имя файла изображения с помощью ImageNameService на основе типа содержимого файла. Создается путь для сохранения файла изображения на основе заданного UPLOAD\_DIRECTORY и сгенерированного имени файла. Далее происходит запись байтов изображения в указанный файл. Создается экземпляр класса ProductImage, устанавливается связь с продуктом и задается URL для доступа к изображению. Наконец, сохраненный объект ProductImage сохраняется в базе данных с использованием ProductImageRepository.

```

public class ProductImageService {

    1 usage
    private final ProductImageRepository productImageRepository;
    1 usage
    private final ProductRepository productRepository;
    1 usage
    private final ImageNameService nameService;
    1 usage
    @Value("src/main/resources/static")
    private String UPLOAD_DIRECTORY;

    1 usage
    @Transactional
    public void save(long productID, MultipartFile file) throws IOException, ProductNotFoundException {
        Optional<Product> optionalProduct = productRepository.findById(productID);
        if (optionalProduct.isEmpty())
            throw new ProductNotFoundException("Product not found");
        Product product = optionalProduct.get();

        String productName = nameService.generate(file.getContentType());
        Path filenameAndPath = Paths.get( first: UPLOAD_DIRECTORY + "/products", productName);
        Files.write(filenameAndPath, file.getBytes());

        ProductImage productImage = new ProductImage();
        productImage.setProduct(product);
        productImage.setUrl("/products/" + productName);

        productImageRepository.save(productImage);
    }
}

```

Рисунок 3.5.2.1 – Java код класса ProductImageService

Класс ProductImageService предоставляет удобный интерфейс для сохранения изображений товаров, обеспечивая генерацию уникальных имен файлов, сохранение файлов на диске и связывание изображений с соответствующими продуктами в базе данных.

### 3.5.3 Класс «ProductService»

Данный класс представляет сервисный слой приложения, который отвечает за работу с сущностями типа Product. Он имеет зависимость от репозитория productRepository, который в свою очередь позволяет осуществлять доступ к данным, связанным с цветами, хранящимися в базе данных – рисунок 3.5.3.1.

```

@Service
public class FlowerService {
    @Autowired
    private FlowerRepo flowerRepository;

    public List<Flower> getFlowers(){
        return flowerRepository.findAll();
    }

    public List<Flower> findById(Long id){
        Optional<Flower> flower = flowerRepository.findById(id);
        return flower.map(Collections::singletonList).orElse(Collections.emptyList());
    }
}

```

Рисунок 3.5.3.1 – Java код класса ProductService

Данный класс предоставляет набор методов, которые могут быть использованы другими компонентами приложения, например, контроллерами, чтобы получить доступ к информации о товарах и осуществлять с ними операции.

### 3.6 Использование Spring MVC для реализации веб-интерфейса

RESTful контроллеры в Spring[11] используются для создания веб-сервисов, которые предоставляют клиентам API для доступа к данным и функциональности на сервере. Они позволяют обрабатывать запросы через HTTP-протокол и возвращать данные в формате JSON/XML[12].

#### 3.6.1 Класс «AdminController»

Класс AdminController является контроллером, отвечающим за обработку запросов, связанных с администрированием товаров в системе. Он использует сервисы ProductService и ProductImageService для выполнения операций над товарами и изображениями товаров соответственно - рисунок 3.6.1.1.



```

@RestController
@RequiredArgsConstructor
@RequestMapping("/admin")
public class AdminController {

    3 usages
    private final ProductService productService;

    2 usages
    private final ProductMapper productMapper = ProductMapper.INSTANCE;

    1 usage
    private final ProductImageService productImageService;

    @GetMapping("/products")
    public List<ProductDTO> getAll() {
        return productService.findAll().stream().
            map(productMapper::convertProductToProductDTO).collect(Collectors.toList());
    }

    @PostMapping("/products/add")
    public void addProduct(@RequestBody NewProductDTO newProductDTO) {
        productService.save(productMapper.convertNewProductDTOToProduct(newProductDTO));
    }
}

```

Рисунок 3.6.1.1 – Java код класса AdminController

Аннотация `@RestController` указывает, что данный класс является контроллером REST API, который обрабатывает HTTP-запросы[13] и возвращает данные в формате JSON.

Аннотация `@RequestMapping("/admin")` определяет базовый URL-адрес для всех методов контроллера, начинающихся с `"/admin"`.

В конструкторе класса `AdminController` происходит внедрение зависимостей `ProductService` и `ProductImageService`, которые необходимы для выполнения операций над товарами и изображениями товаров.

Метод `getAll` обрабатывает GET-запросы по пути `"/admin/products"` и возвращает список всех товаров в виде списка объектов `ProductDTO`. Он использует метод `findAll` предоставляемый `ProductService` для получения всех товаров, а затем с помощью объекта `ProductMapper` преобразует каждый товар в соответствующий объект `ProductDTO`.

Метод `addProduct` обрабатывает POST-запросы по пути `"/admin/products/add"` и принимает объект `NewProductDTO` в теле запроса. Он преобразует объект `NewProductDTO` в объект `Product` с помощью объекта

ProductMapper и сохраняет товар с использованием метода save предоставляемого ProductService.

Метод delProduct обрабатывает POST-запросы по пути "/admin/products/del/{id}" и принимает идентификатор товара для удаления в виде пути переменной "id". Он вызывает метод delete предоставляемый ProductService для удаления товара с заданным идентификатором.

Метод loadImage обрабатывает POST-запросы по пути "/admin/load\_image/{id}" и принимает идентификатор товара и файл изображения в теле запроса. Он вызывает метод save предоставляемый ProductImageService для сохранения изображения товара, связанного с заданным идентификатором.

Класс AdminController обеспечивает обработку запросов, связанных с администрированием товаров, включая получение всех товаров, добавление нового товара, удаление товара и загрузку изображения товара.

### **3.6.2 Класс «AuthenticationController»**

Класс AuthenticationController является контроллером, отвечающим за обработку запросов, связанных с аутентификацией и регистрацией пользователей - рисунок 3.6.2.1.

Аннотация @RestController указывает, что данный класс является контроллером REST API[14], который обрабатывает HTTP-запросы и возвращает данные в формате JSON.

Аннотация @CrossOrigin(origins = "http://localhost:3000") позволяет выполнение запросов с указанного источника (http://localhost:3000) для обеспечения междоменных запросов.

В конструкторе класса AuthenticationController происходит внедрение зависимости AuthenticationService, которая используется для обработки аутентификации и регистрации пользователей.

Метод register обрабатывает POST-запросы по пути "/register" и принимает объект RegisterRequest в теле запроса. Он передает этот объект в метод register предоставляемый AuthenticationService для регистрации нового пользователя. В

результате возвращается объект `AuthenticationResponse`, содержащий JWT-токен.

```
@RestController
@CrossOrigin(origins = "http://localhost:3000")
@RequiredArgsConstructor
public class AuthenticationController {

    2 usages
    private final AuthenticationService service;

    @PostMapping("/register")
    public AuthenticationResponse register(@RequestBody RegisterRequest request) { return service.register(request); }

    @PostMapping("/login")
    public AuthenticationResponse authenticate(@RequestBody AuthenticationRequest request) {
        return service.authenticate(request);
    }
}
```

Рисунок 3.6.2.1 – Java код класса `CartController`

Метод `authenticate` обрабатывает POST-запросы по пути `"/login"` и принимает объект `AuthenticationRequest` в теле запроса. Он передает этот объект в метод `authenticate` предоставляемый `AuthenticationService` для аутентификации пользователя. В результате возвращается объект `AuthenticationResponse`, содержащий JWT-токен.

Класс `AuthenticationController` обеспечивает обработку запросов, связанных с аутентификацией и регистрацией пользователей, и предоставляет эндпоинты для выполнения этих операций.

### 3.6.3 Класс «`CartController`»

Класс `CartController` является контроллером, отвечающим за обработку запросов, связанных с корзиной покупок - рисунок 3.6.3.1.

Аннотация `@RestController` указывает, что данный класс является контроллером REST API, который обрабатывает HTTP-запросы и возвращает данные в формате JSON.

В конструкторе класса `CartController` происходит внедрение зависимостей `AuthenticatedCustomerService`, `ProductMapper` и `CartService`.

```

@RestController
@RequiredArgsConstructor
@RequestMapping("/cart")
public class CartController {

    4 usages
    private final AuthenticatedCustomerService authenticatedCustomerService;

    1 usage
    private final ProductMapper productMapper = ProductMapper.INSTANCE;

    4 usages
    private final CartService cartService;

    @GetMapping
    public List<ProductDTO> getCart() {
        Customer customer = authenticatedCustomerService.getAuthenticatedCustomer();
        return cartService.findCartByPerson(customer.getId()).stream()
            .map(productMapper::convertProductToProductDTO).collect(Collectors.toList());
    }

    1 usage
    @GetMapping("/in_cart/{id}")
    public Boolean isItemInCart(@PathVariable("id") long productId) {
        Customer customer = authenticatedCustomerService.getAuthenticatedCustomer();
        return cartService.isInCart(customer.getId(), productId) != null;
    }
}

```

Рисунок 3.6.3.1 – Java код класса CartController

Метод `getCart` обрабатывает GET-запросы по пути `"/cart"` и возвращает список продуктов, находящихся в корзине покупок текущего аутентифицированного пользователя. Для получения текущего пользователя используется `AuthenticatedCustomerService`. Затем вызывается метод `findCartByPerson` из `CartService` для поиска продуктов в корзине и преобразования их в список объектов `ProductDTO` с помощью `ProductMapper`.

Метод `isItemInCart` обрабатывает GET-запросы по пути `"/cart/in_cart/{id}"` и проверяет, находится ли товар с указанным идентификатором в корзине текущего пользователя. Он использует `AuthenticatedCustomerService` для получения текущего пользователя и вызывает метод `isInCart` из `CartService` для проверки наличия товара в корзине.

Метод `add` обрабатывает POST-запросы по пути `"/cart/add/{id}"` и добавляет товар с указанным идентификатором в корзину покупок текущего пользователя. Сначала он проверяет, отсутствует ли товар уже в корзине,

используя метод `isItemInCart`. Если товар отсутствует, вызывается метод `addItemToCart` из `CartService` для добавления товара в корзину пользователя. Если товар уже присутствует в корзине, генерируется исключение `AlreadyInCartException`.

Метод `remove` обрабатывает POST-запросы по пути `"/cart/delete/{id}"` и удаляет товар с указанным идентификатором из корзины покупок текущего пользователя. Он использует `AuthenticatedCustomerService` для получения текущего пользователя и вызывает метод `removeItemFromCart` из `CartService` для удаления товара из корзины.

Класс `CartController` предоставляет эндпоинты для работы с корзиной покупок, позволяя пользователям просматривать содержимое корзины, добавлять товары в корзину и удалять товары из корзины.

### **3.7 Использование паттерна DTO**

DTO (Data Transfer Object)[\[15\]](#) - это классы, которые используются для передачи данных между различными слоями или компонентами системы. Они служат для удобной и эффективной передачи данных, облегчая обработку и обмен информацией между различными частями системы.

Одной из функций данных классов является передача данных: DTO предоставляют структурированный способ передачи данных между различными компонентами системы. Они определяют поля, которые должны быть переданы или получены из других компонентов.

Также благодаря им происходит упрощение взаимодействия: DTO упрощают взаимодействие между компонентами, предоставляя четко определенные объекты данных, которые могут быть легко созданы, заполнены и переданы.

К тому же DTO позволяют изолировать данные от внутренней реализации компонентов. Они определяют только необходимые данные для передачи, скрывая сложность внутренней структуры и повышая безопасность.

Пример класса `CustomerDTO`, представленного ниже, показывает DTO для представления информации о клиенте - рисунок 3.7.1.

```
@Getter
@Setter
@NoArgsConstructor
@FieldDefaults(level = AccessLevel.PRIVATE)
public class CustomerDTO {
    String name;
    String email;
}
```

Рисунок 3.7.1 – Java код класса CustomerDTO

Этот класс CustomerDTO имеет два приватных поля: name (имя клиента) и email (электронная почта клиента). Он также предоставляет геттеры и сеттеры для этих полей. Аннотации Lombok, такие как @Getter, @Setter, @NoArgsConstructor и @FieldDefaults, используются для автоматической генерации геттеров, сеттеров, конструктора без аргументов и настройки уровня доступа по умолчанию для полей класса.

CustomerDTO служит для передачи информации о клиенте между различными компонентами системы. Например, он может использоваться при передаче данных о клиенте между клиентским интерфейсом и серверной частью системы или между различными слоями приложения. Предоставление DTO позволяет явно определить данные, которые должны быть переданы или получены для работы с клиентом, и упрощает обработку этих данных в различных компонентах системы.

### 3.8 Использование библиотеки-маппера

Mapper классы (также известные как преобразователи) используются для преобразования данных между объектами различных типов или классов. Они облегчают процесс преобразования данных, особенно при работе с объектами DTO (Data Transfer Object) и сущностями (entities) в приложении. Mapper классы позволяют автоматически выполнять преобразование полей и свойств одного объекта в другой.

Mapper классы позволяют преобразовывать объекты одного типа в объекты другого типа. Это особенно полезно при работе с DTO и сущностями, где объекты имеют разные структуры, но содержат схожие данные.

А также Mapper классы автоматически преобразуют поля одного объекта в поля другого объекта, исходя из их имени и типа. Это позволяет избежать ручного написания повторяющегося кода для преобразования каждого поля.

Некоторые Mapper классы поддерживают обратное преобразование, позволяя выполнять обратную операцию преобразования, то есть преобразовывать объекты обратно в исходный тип.

Пример класса CustomerMapper, представленного ниже, демонстрирует Mapper класс для преобразования объектов Customer в объекты CustomerDTO - рисунок 3.8.1.

```
@Mapper
public interface CustomerMapper {
    1 usage
    CustomerMapper INSTANCE = Mappers.getMapper(CustomerMapper.class);
    1 usage 1 implementation
    CustomerDTO convertCustomerToCustomerDTO(Customer customer);
}
```

Рисунок 3.8.1 – Java код класса CustomerMapper

В этом примере CustomerMapper объявлен как интерфейс с аннотацией @Mapper, что указывает на то, что это Mapper класс. Интерфейс также объявляет статическую переменную INSTANCE, которая используется для получения экземпляра CustomerMapper через Mappers.getMapper(). Метод convertCustomerToCustomerDTO() определен в интерфейсе и используется для преобразования объекта Customer в объект CustomerDTO.

CustomerMapper служит для автоматического преобразования объектов Customer в объекты CustomerDTO. Он позволяет сконфигурировать правила преобразования и выполнять преобразование полей между этими двумя типами объектов. Такой Mapper класс упрощает преобразование данных между слоями

приложения и повышает читаемость кода. Он избавляет разработчиков от необходимости вручную преобразовывать каждое поле объекта при создании DTO из сущности и наоборот.

### **3.9 Использование Spring Security**

Spring Security представляет собой мощный фреймворк для обеспечения безопасности в приложениях на платформе Spring. Его основная цель - защита приложения путем управления аутентификацией (проверкой подлинности) и авторизацией (контролем доступа).

В разрабатываемом приложении используется JSON Web Token (JWT) для аутентификации пользователей. Конфигурационный класс `JWTConfig` определяет бины, необходимые для работы с JWT - рисунок 3.9.1. В нем определен бин `userDetailsService()`, который осуществляет поиск пользователя в репозитории на основе его электронной почты и возвращает объект `CustomerDetails`, содержащий информацию о пользователе. Также определен бин `authenticationProvider()`, который использует `userDetailsService` для проверки подлинности пользователей при аутентификации. Здесь также определен бин `passwordEncoder()`, который используется для хеширования паролей пользователей.



```

@RequiredArgsConstructor
public class JWTConfig {

    1 usage
    private final CustomerRepository repository;

    1 usage
    @Bean
    public UserDetailsService userDetailsService() {
        return username -> {
            Customer customer = repository.findByEmail(username)
                .orElseThrow(() -> new UsernameNotFoundException("User not found"));
            return new CustomerDetails(customer);
        };
    }

    @Bean
    public AuthenticationProvider authenticationProvider() {
        DaoAuthenticationProvider authProvider = new DaoAuthenticationProvider();
        authProvider.setUserDetailsService(userDetailsService());
        authProvider.setPasswordEncoder(passwordEncoder());
        return authProvider;
    }

    @Bean
    public AuthenticationManager authenticationManager(AuthenticationConfiguration config) throws Exception {
        return config.getAuthenticationManager();
    }
}

```

Рисунок 3.9.1 – Java код класса JWTConfig

Дальше идет класс SecurityConfiguration определяет настройки безопасности веб-приложения, включая правила доступа, провайдер аутентификации и фильтры для обработки аутентификации и авторизации - рисунок 3.9.2.

```

public class SecurityConfiguration {

    1 usage
    private final JwtAuthenticationFilter jwtAuthFilter;
    1 usage
    private final AuthenticationProvider authenticationProvider;

    @Bean
    public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {
        http
            .csrf() CsrfConfigurer<HttpSecurity>
            .disable() HttpSecurity
            .cors() CorsConfigurer<HttpSecurity>
            .and() HttpSecurity
            .authorizeHttpRequests() AuthorizeHttpRequestsConfigurer<...>.AuthorizationManagerRequestMatcherRegistry
            .requestMatchers( ...patterns: "/register", "/login") AuthorizeHttpRequestsConfigurer<...>.AuthorizedUrl
            .permitAll() AuthorizeHttpRequestsConfigurer<...>.AuthorizationManagerRequestMatcherRegistry
            .requestMatchers(HttpMethod.GET, ...patterns: "/test", "/people", "/image/**", "/cart/**") Authc
            .authenticated() AuthorizeHttpRequestsConfigurer<...>.AuthorizationManagerRequestMatcherRegistry
            .requestMatchers(HttpMethod.POST, ...patterns: "/cart/**") AuthorizeHttpRequestsConfigurer<...>.AuthorizedUrl
            .authenticated() AuthorizeHttpRequestsConfigurer<...>.AuthorizationManagerRequestMatcherRegistry
            .requestMatchers( ...patterns: "/admin/**") AuthorizeHttpRequestsConfigurer<...>.AuthorizedUrl
            .hasAuthority("ADMIN") AuthorizeHttpRequestsConfigurer<...>.AuthorizationManagerRequestMatcherRegistry
            .and() HttpSecurity |
            .authenticationProvider(authenticationProvider)
            .addFilterBefore(jwtAuthFilter, UsernamePasswordAuthenticationFilter.class)
            .logout() LogoutConfigurer<HttpSecurity>
            .logoutUrl("/api/v1/auth/logout");

        return http.build();
    }
}

```

Рисунок 3.9.2 – Java код класса SecurityConfiguration

Класс `AuthenticationController` является контроллером, отвечающим за обработку запросов аутентификации и регистрации. В методе `register()` происходит регистрация нового пользователя, создание JWT-токена и его возврат клиенту. В методе `authenticate()` выполняется аутентификация пользователя на основе предоставленных учетных данных, создается JWT-токен и возвращается клиенту.

Класс `AuthenticationService` предоставляет функциональность для регистрации и аутентификации пользователей. В методе `register()` создается новый объект `Person` на основе данных из запроса, пароль хешируется с использованием `passwordEncoder`, пользователь сохраняется в репозитории, и создается JWT-токен для нового пользователя. В методе `authenticate()` выполняется аутентификация пользователя с помощью `authenticationManager` на

основе предоставленных учетных данных, создается JWT-токен и возвращается клиенту.

Класс `JwtAuthenticationFilter` является фильтром, который выполняет аутентификацию на основе JWT. В этом фильтре проверяется наличие заголовка `Authorization` с токеном JWT. Если токен присутствует и валиден, то пользователь аутентифицируется и устанавливается контекст безопасности.

Класс `JwtService` предоставляет функциональность для работы с JWT. Он содержит методы для генерации токена, извлечения информации из токена, проверки валидности токена и т.д. Также в этом классе определен секретный ключ для подписи токена.

Класс `CustomerDetails` реализует интерфейс `UserDetails` и представляет информацию о пользователе, необходимую для аутентификации и авторизации. Он содержит методы для получения имени пользователя, пароля, роли и других атрибутов.

Таким образом, описанные выше классы и их взаимодействие реализовывает аутентификацию и авторизацию с использованием Spring Security и JWT в приложении.

## **4 ТЕСТИРОВАНИЕ ПРИЛОЖЕНИЯ**

### **4.1 Тестирование приложения со стороны пользователя**

Проведем тестирование разработанного приложения. Для начала проверим корректность работы страниц регистрации и входа. Зарегистрируем нового пользователя приложения – рисунок 4.1.1.

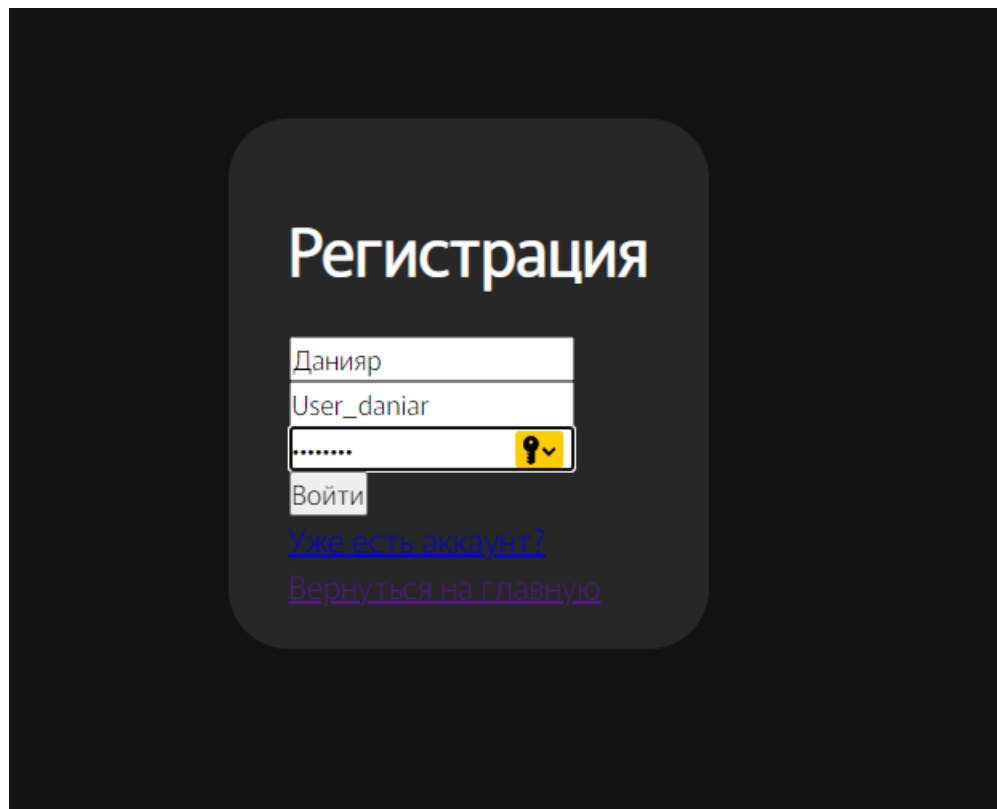


Рисунок 4.1.1 – Тестирование работы регистрации

Пользователь успешно создан, данные о нем были добавлены в базу данных – рисунок 4.1.2.

WHERE		ORDER BY			
	id	email	name	password	role
1	1	fc_mirea.edu@mail.ru	Leo Messi	\$2a\$10\$fngavgR3MrmJiByXw.Vg4eb7w...	ADMIN
2	2	egor.belosludtsev@mail.ru	Егор	\$2a\$10\$K0gtyg2JgPF/2LXvxIqqEeAfH...	USER
3	3	User_daniar	Данияр	\$2a\$10\$PGMoodezvYH2i20VWc72keyJp...	USER

Рисунок 4.1.2 – Таблица «users» базы данных

После чего войдем, используя данные созданного ранее аккаунта – рисунок 4.1.3

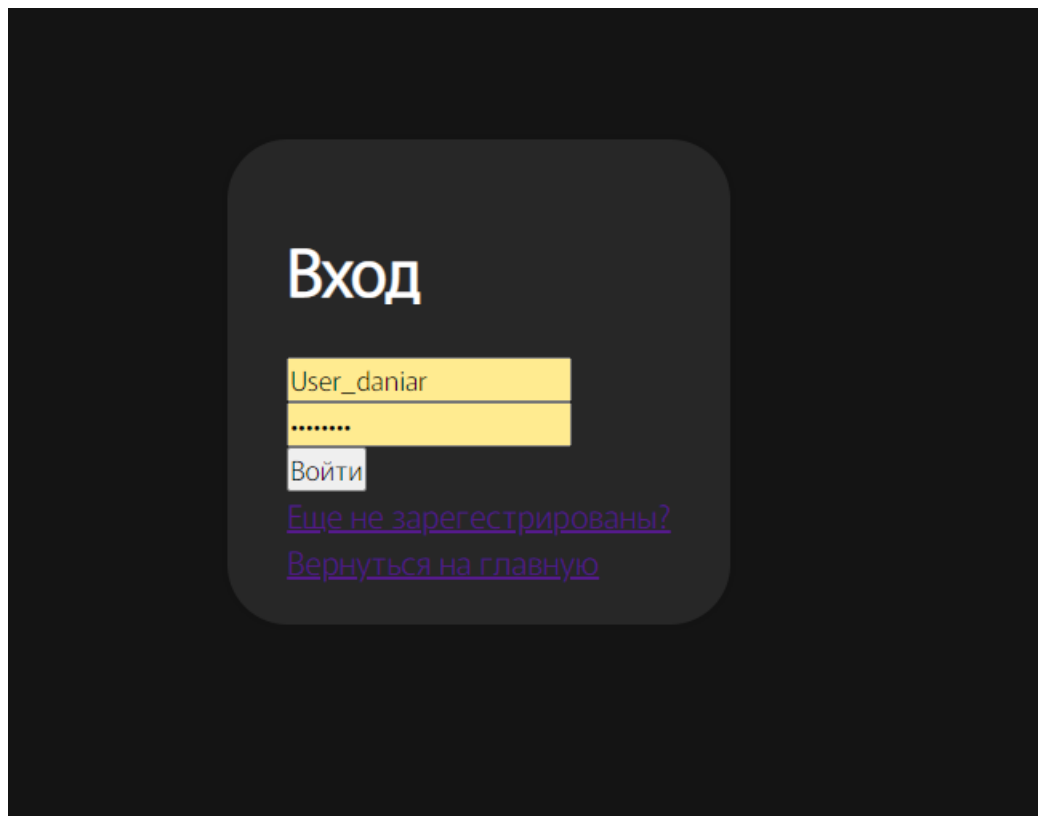


Рисунок 4.1.3 – Тестирование работы авторизации

Авторизация прошла успешно – рисунок 4.1.4. Теперь добавим товар в корзину, после его добавления итоговая сумма товаров в корзине изменилась – рисунки 4.1.5 – 4.1.7.



Рисунок 4.1.4 – Результат успешной авторизации



1. Футбольный мяч 1

Описание: обыкновенный мячик

Цена: 2000

Картинка:

Выберите файл


мяч1.png

В корзину

Удалить

Рисунок 4.1.5 – Тестирование работы добавления в корзину товара

MainPage



1. Футбольный мяч 1

Описание: обыкновенный мячик

Цена: 2000

+

1

-

Удалить из корзины

Рисунок 4.1.6 – Товар добавлен в корзину

WHERE	ORDER BY
product_id	customer_id
1	35
	3

Рисунок 4.1.7 – Измененные данные в таблице «cart»

Также проверим корректность удаления и увеличения количества товара в корзине – рисунки 4.1.8 – 4.1.9.

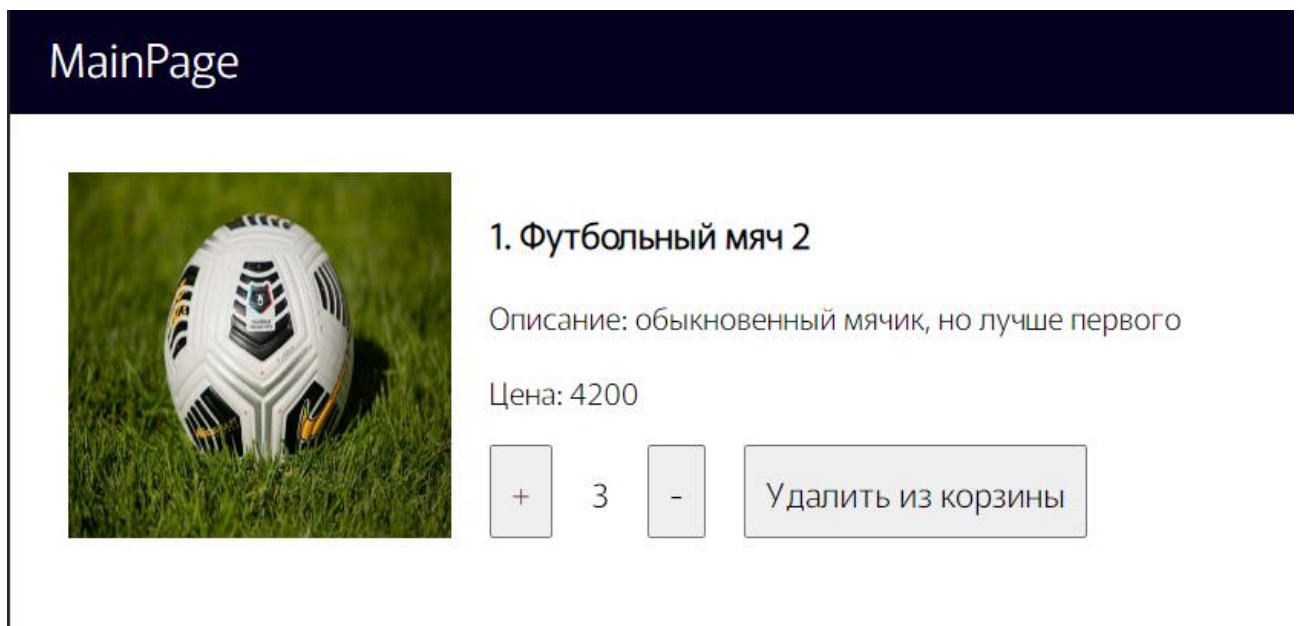


Рисунок 4.1.8 – Тестирование увеличения кол-ва товара в корзине



Рисунок 4.1.9 – Тестирование удаления товара из корзины

Проверим корректность работы функции добавления нового товара в данный магазин – рисунки 4.1.10 - 4.1.12. Данной функцией обладаешь лишь пользователь, которые отмечены как «admin».

Название товара:

Перчатки Nike

Описание товара:

-----

Категория товара:

Перчатки

Цена:


1200

Добавить

## Перчатки

Рисунок 4.1.10 – Заполнение данных нового товара





### 1. Перчатки Nike

Описание: ----

Цена: 1200

Картинка:

перчатки1.jpg

Рисунок 4.1.11 – Выбор изображения для нового товара

	🔍 id ↕	📁 category ↕	📄 description ↕	📄 name ↕	💰 price ↕
1	35	Мячи	обыкновенный мячик	Футбольный ...	2000
2	36	Мячи	обыкновенный мячик, ...	Футбольный ...	4200
3	37	Перчатки	----	Перчатки Ni...	1200

Рисунок 4.1.12 – Измененные данные в таблице «product»

Также проверим корректность выполнения удаления данного товара из каталога от лица админа – рисунки 4.1.13 – 4.1.14.

Название товара:

Перчатки Nike

Описание товара:

-----

Категория товара:

Перчатки

Цена:

1200

Добавить

## Перчатки

Рисунок 4.1.13 – Удаление товара из каталога

WHERE		ORDER BY			
	id	category	description	name	price
1	35	Мячи	обыкновенный мячик	Футбольный ...	2000
2	36	Мячи	обыкновенный мячик, ...	Футбольный ...	4200

Рисунок 4.1.14 – Измененные данные в таблице «product»

## 4.2 Тестирование приложения с помощью Postman

Проверим работу обработчика POST запроса на регистрацию пользователя, тело POST запроса показано на рисунке 4.2.1.

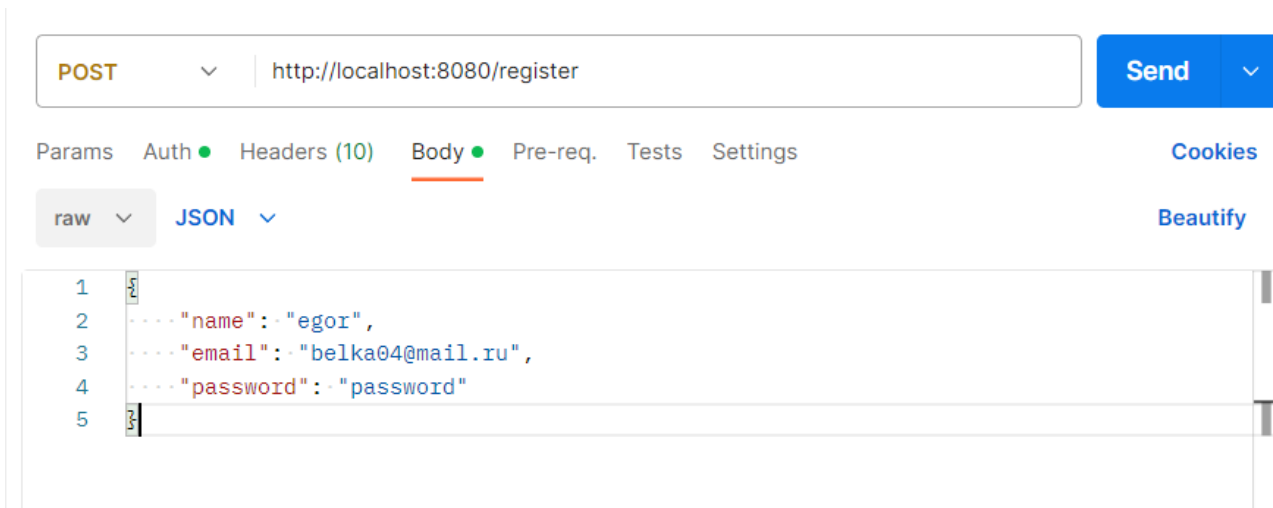


Рисунок 4.2.1 – Конфигурация POST запроса для регистрации

Ответом на запрос получает JWT токен – рисунок 4.2.2.

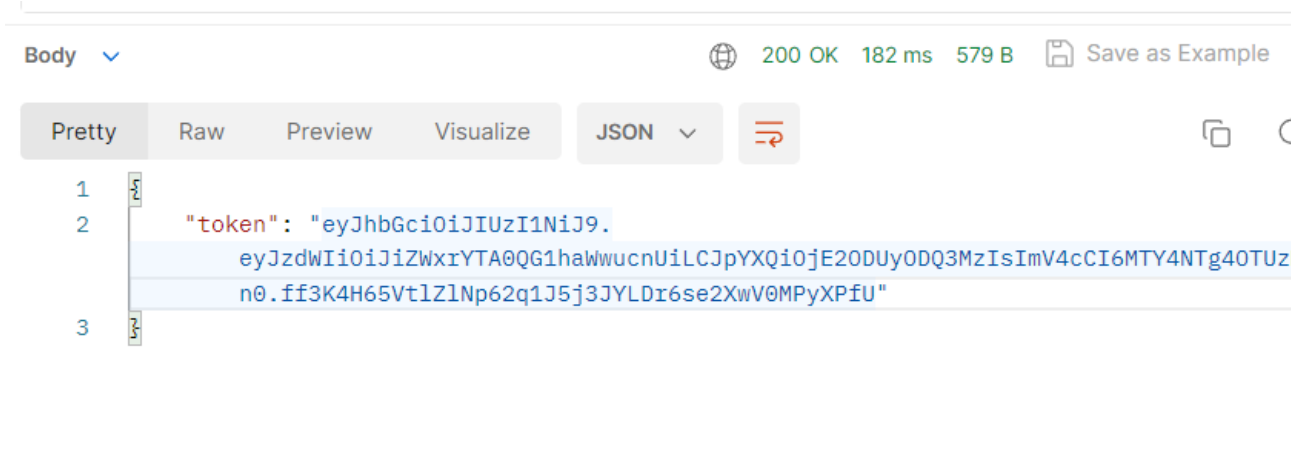


Рисунок 4.2.2 – Ответ на POST запрос регистрации

Проверим работу обработчика POST запроса на авторизацию пользователя, тело POST запроса показано на рисунке 4.2.3.

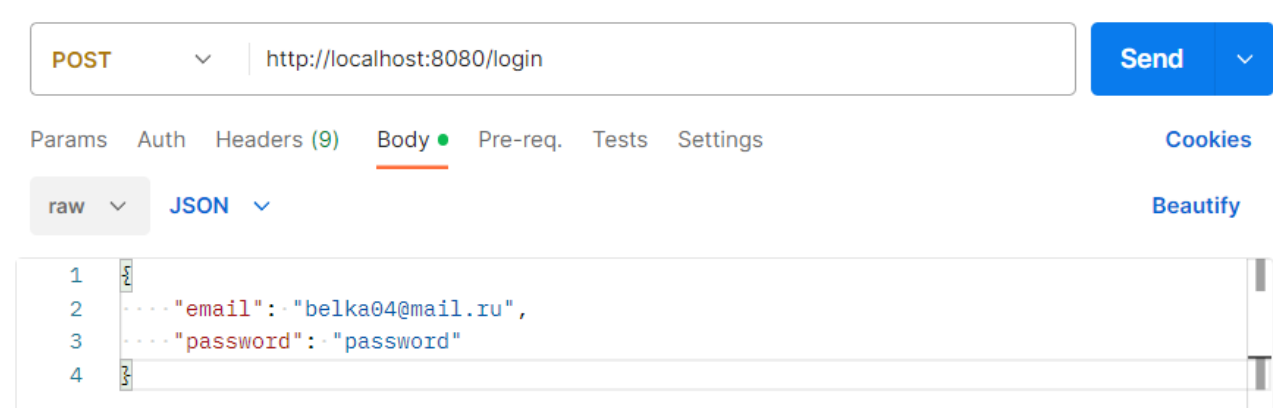


Рисунок 4.2.3 – Конфигурация POST запроса для авторизации

Ответом на запрос получает очередной JWT токен – рисунок 4.2.4.



Рисунок 4.2.4 – Ответ на POST запрос авторизации

Проверим работу обработчика POST запроса на добавление товара в корзину – рисунок 4.2.5.

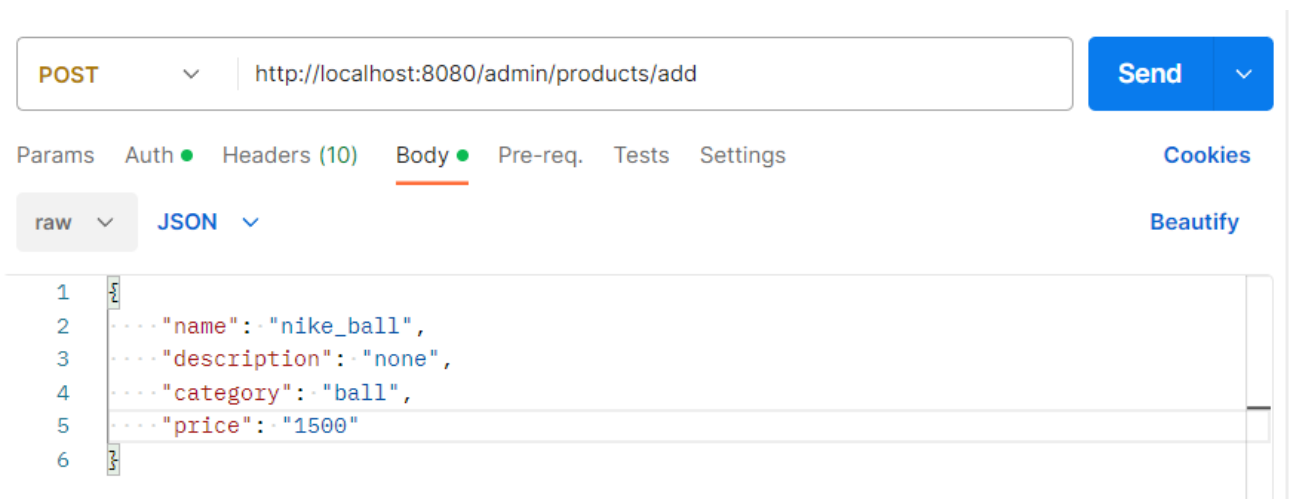


Рисунок 4.2.5 – Конфигурация POST запроса на добавление товара в корзину

Ответом на запрос является код успешного выполнения запроса 200 и детальная информация об данном товаре – рисунок 4.2.8.



Рисунок 4.2.8 – Ответ на POST запрос добавления товара в корзину

Создадим POST запрос удаления товара из корзины и проверим его работу – рисунки 4.2.9 – 4.2.11.

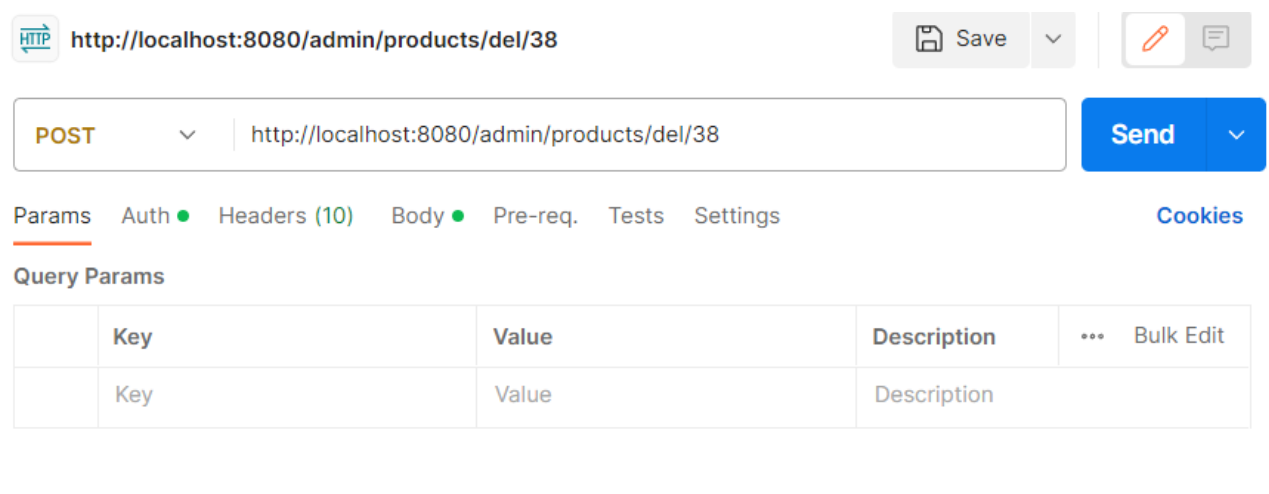


Рисунок 4.2.9 – Тело POST запроса на удаление товара из корзины

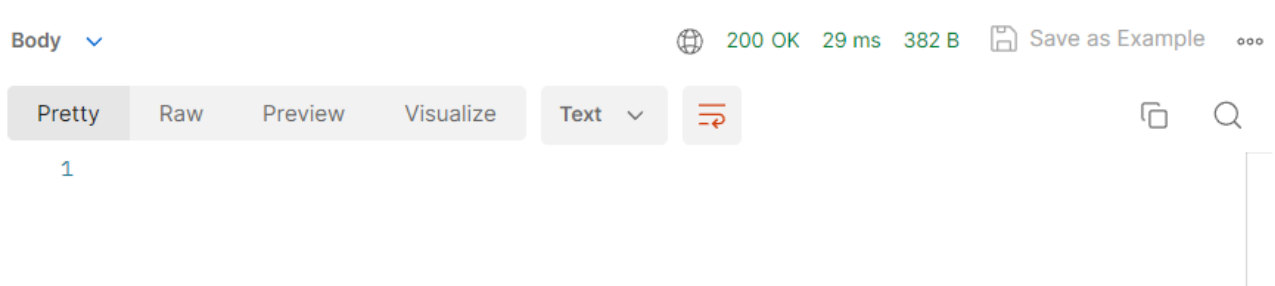


Рисунок 4.2.10 – Ответ сервера на POST запрос удаления товара из корзины

```
Body ▾ 200 OK 28 ms 861 B Save as Example

Pretty Raw Preview Visualize JSON ↺

1  [
2    {
3      "id": 35,
4      "name": "Футбольный мяч 1",
5      "description": "обыкновенный мячик",
6      "category": "Мячи",
7      "price": 2000,
8      "images": [
9        {
10         "url": "/products/UYFISe9KMu1Ma7DXlKalevC039DKPC.png"
11       }
12     ]
13   },
14   {
15     "id": 36,
16     "name": "Футбольный мяч 2",
17     "description": "обыкновенный мячик, но лучше первого",
18     "category": "Мячи",
19     "price": 4200,
20     "images": [
21       {
22         "url": "/products/vK6XgPUriDN2wTEcbJ2oDYKwCAlIyL.jpeg"
23       }
24     ]
25   }
26 ]
```

Рисунок 4.2.11 – Ответ сервера на GET запрос просмотра всех товаров

Как видно из ответа, отправленного сервером, в каталоге отсутствует удаленный товар.

В результате тестирования было установлено, что приложение соответствует всем требованиям, предъявленным к нему в ходе разработки. Все основные функциональные возможности работают корректно и без сбоев, а также обеспечивают быстрый и удобный доступ к необходимым данным и функциям.

## **ЗАКЛЮЧЕНИЕ**

Результатом выполнения курсовой работы на тему "Магазин футбольной атрибутики" стало разработанное серверное программное приложение, которое написано с использованием языка Java и фреймворка Spring, включая Spring Data JPA, Spring Security, Spring Core и Spring MVC. Для создания серверного приложения требовались навыки отличного владения и знания всех нюансов среды разработки, в которой написана работа, поэтому в ходе ее выполнения приобретены навыки работы в IntelliJ IDEA. Изучен фреймворк Spring, который предназначен для разработки приложений на языке Java. Он предоставляет широкий набор инструментов и библиотек для упрощения процесса разработки, повышения производительности и улучшения масштабируемости приложений.

Благодаря глубокому анализу предметной области, учтены и проработаны все недочеты веб-ресурсов с похожей тематикой. Разработанный веб-ресурс содержит необходимые элементы, такие как изображения, текст и визуальное оформление, которые совместимы между собой и обеспечивают единообразие

стилей и шрифтов на всех страницах, а также имеет всю функциональность, необходимую для интернет – магазина.

Учитывая все вышеперечисленные пункты, можно с уверенностью сказать, что все требования, поставленные в задании на курсовую работу, соблюдены и выполнены успешно, цель работы достигнута. Разработанное приложение является полноценным и функциональным инструментом для управления заказами в цветочном магазине и готово к использованию в реальных условиях.

Исходный код Интернет-ресурса по курсовой работе доступен по ссылке:  
<https://github.com/Belchonok31/java-term-work>



## СПИСОК ИНФОРМАЦИОННЫХ ИСТОЧНИКОВ

1. Кэролл М. Spring в действии. — М.: ДМК Пресс, 2020. — 832 с. — ISBN 978-5-97060-840-9. — Текст: печатная книга.
2. Шарма А. Spring Boot Cookbook / А. Шарма. — М.: Питер, 2021. — 800 с. — ISBN 978-5-496-03472-3. — Текст: печатная книга.
3. Gupta M. Spring Boot 2.0 Projects: Build production-grade reactive applications and microservices with Spring Boot / M. Gupta. — Birmingham, UK: Packt Publishing, 2018. — 556 p. — ISBN 978-1789136159. — Текст: электронная книга.
4. Javid S. Pro Spring Boot 2: An Authoritative Guide to Building Microservices, Web and Enterprise Applications, and Best Practices / S. Javid. — Berkeley, CA: Apress, 2019. — 484 p. — ISBN 978-1484247104. — Текст: электронная книга.
5. Sharma S. Hands-On Microservices with Spring Boot and Spring Cloud: Build and deploy Java microservices using Spring Cloud, Istio, and Kubernetes / S. Sharma. — Birmingham, UK: Packt Publishing, 2019. — 374 p. — ISBN 978-1789613476. — Текст: электронная книга.
6. Verma A. Spring Boot 2.0 Projects: Build production-grade reactive and conventional microservices with Spring Boot / A. Verma. — Birmingham, UK: Packt Publishing, 2018. — 404 p. — ISBN 978-1789136159. — Текст: электронная книга.
7. Уолла К. Spring Boot 2: для профессионалов / К. Уолла, К. Хеммингс, Р. Феррара. — М.: ДМК Пресс, 2021. — 528 с. — ISBN 978-5-97060-982-6. — Текст: печатная книга.
8. Sharma R. Learning Spring Boot 2.0: Second Edition / R. Sharma. — Birmingham, UK: Packt Publishing, 2018. — 432 p. — ISBN 978-1786463784. — Текст: электронная книга.
9. Бреславский Д. Spring Framework 5. Программирование с применением самого широко используемого Java-фреймворка. — СПб.: Питер, 2019. — 720 с. — ISBN 978-5-4461-0961-2. — Текст: печатная книга.

10. Gabriele P. Spring Boot 2: Up and Running: Building Cloud Native Applications / P. Gabriele. — Sebastopol, CA: O'Reilly Media, 2019. — 416 p. — ISBN 978-1491984134. — Текст: печатная книга.

11. Feliciano A. Spring Boot Cookbook: Configure, test, extend, deploy, and monitor your Spring Boot application both outside and inside Docker / A. Feliciano. — Birmingham, UK: Packt Publishing, 2021. — 484 p. — ISBN 978-1801074268. — Текст: электронная книга.

12. Goyvaerts M. Spring Boot: Up and Running: Building Cloud Native Java and Kotlin Applications / M. Goyvaerts. — Sebastopol, CA: O'Reilly Media, 2021. — 300 p. — ISBN 978-1492076975. — Текст: электронная книга.

13. Datta S. Spring Boot 2.0 Projects: Develop 10 end-to-end projects with clean code using Spring Boot 2.0 / S. Datta. — Birmingham, UK: Packt Publishing, 2018. — 410 p. — ISBN 978-1789136159. — Текст: электронная книга.

14. Verma S. Mastering Spring Boot 2.0: Build modern, cloud-native, and distributed systems using Spring Boot / S. Verma. — Birmingham, UK: Packt Publishing, 2018. — 434 p. — ISBN 978-1787127562. — Текст: электронная книга.

15. Kurniawan R. Building RESTful Web Services with Spring 5: Leverage the power of Spring 5.0, Java SE 9, and Spring Boot 2.0, Third Edition / R. Kurniawan. — Birmingham, UK: Packt Publishing, 2018. — 570 p. — ISBN 978-1788997571. — Текст: электронная книга.