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 Джава
по профилю: Разработка программных продуктов и проектирование информационных систем
направления профессиональной подготовки: 09.03.04 «Программная инженерия»

Тема: Приложение «Мир Формулы-1»

Студент: Прокопчук Роман Олегович
Группа: ИКБО-01-22
Работа представлена к защите_____(дата)_____/Прокопчук Р.О./
(подпись и ф.и.о. студента)

Руководитель: старший преподаватель Рачков Андрей Владимирович

Работа допущена к защите_____(дата)_____/Рачков А.В./
(подпись и ф.и.о. рук-ля)

Оценка по итогам защиты: _____
_____ / _____ /
_____ / _____ /
(подписи, дата, ф.и.о., должность, звание, уч. степень двух преподавателей, принявших защиту)

М. РТУ МИРЭА. 2024

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 систе
направления профессиональной подготовки: Программная инженерия (09.03.04)

Студент: Прокопчук Роман Олегович
Группа: ИКБО-01-22
Срок представления к защите: 13.05.2024 г.
Руководитель: старший преподаватель Рачков Андрей Владимирович

Тема: Приложение «Мир Формулы-1»

Исходные данные: индивидуальное задание на разработку; документация по Sprin 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 п 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 приложени 2. Обосновать выбор средств ведения разработки. 3. Разработать приложение с использовани фреймворка Spring, выбранной технологии и инструментария. 4.Провести тестировани приложения. 5. Оформить пояснительную записку по курсовой работе в соответствии с ГОС 7.32-2017. 6. Провести анализ текста на антиплагиат 7. Создать презентацию по выполненно курсовой работе.

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

Зав. кафедрой ИиППО: _____/Р. Г. Болбаков/, «28» февраля 2024 г.

Задание на КР выдал: _____/ А.В. Рачков /, «28» февраля 2024 г.

Задание на КР получил: _____/ Р.О. Прокопчук /, «28» февраля 2024 г.

РЕФЕРАТ

Отчёт 37 с., 31 рис., 14 источн.

ФОМУЛА-1, HTTP-ЗАПРОС, SPRING BOOT, SPRING DATA JPA, SPRING SECURITY, ВЕБ-ПРИЛОЖЕНИЕ, GRADLE, JAVA, SQL, JWT, ТОВАР.

Объект исследования – приложение, отвечающее за продажу фанатской атрибутики Формулы-1.

Цель работы: создание приложения, предоставляющего функционал для поиска и выбора необходимой фанатской атрибутики и оформления заказов.

В ходе работы с помощью метода сравнительного анализа были выявлены ключевые решения, которые определяют функциональность приложения.

Изучены технологии создания серверных частей интернет ресурсов.

Результатом работы является приложение «Мир Фомулы-1», отвечающее всем поставленным требованиям.

Report 37 p., 31 fig., 14 sources.

FOMULA-1, HTTP REQUEST, SPRING BOLT, SPRING DATA JPA, SPRING SECURITY, WEB APPLICATION, GRADLE, JAVA, SQL, JWT, PRODUCT.

The object of the study is an application responsible for the sale of Formula 1 fan paraphernalia.

The purpose of the work: to create an application that provides functionality for search and selecting the necessary fan attributes and placing orders.

During the work, using the method of comparative analysis, key solutions were identified that determine the functionality of the application.

Technologies for creating server parts of Internet resources have been studied.

The result of the work is the application "World of Fomula-1", which meets all the requirements.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Формула-1 является одним из самых захватывающих и, несомненно, самым популярным видом автоспорта в мире. Ее влияние выходит далеко за пределы гоночных трасс, проникая в сердца поклонников и становясь неотъемлемой частью их стиля жизни. В современном цифровом мире возможность выразить свою любовь к этому захватывающему виду спорта через приобретение качественной атрибутики становится все более актуальной.

Благодаря сезону 2021 года, который стал одним из самых захватывающих в истории, а также сериалу «Drive to Survive», Формула-1 привлекла огромное количество новых фанатов, из-за чего все больше и больше новых автопроизводителей и гоночных команд стремятся попасть в этот спорт. Среди них: Audi, Porsche, Ford, Geely, Cadillac, Andretti.

Таким образом, для фанатов, число которых растет с каждым годом, возможность выразить свою любовь к этому захватывающему виду спорта через приобретение качественной атрибутики становится все более актуальной.

Целью данной работы является исследование, проектирование и создание полнофункционального приложения, представляющего собой интернет-магазин фанатской атрибутики для Формулы-1.

Данная работа включает в себя создание веб-интерфейса с использованием React, а также создание бэкенд-части приложения с использованием стека фреймворков Spring.

В итоге разработанное приложение позволит пользователям получать информацию о доступных товарах, сохранять товары в список желаемого для последующей покупки и оформлять заказы на понравившиеся товары.

1 ОПИСАНИЕ ЛОГИЧЕСКОЙ СТРУКТУРЫ

1.1 Анализ предметной области

Перед началом разработки необходимо провести тщательный анализ предметной области, чтобы понять особенности изучаемой сферы и удовлетворить требования пользователей.

В процессе анализа изучены различные источники и похожие приложения.

Изучив и сравнив несколько решений, можно определить оптимальную концепцию. Пусть каждое из исследуемых приложений имеет свой порядковый номер: «www.fuelforfans.com[1]» – 1 (рисунок 1), «merchf1.ru[2]» – 2 (рисунок 2), «fueler.store[3]» – 3 (рисунок 3) , «formulasport.pro[4]» – 4 (рисунок 4). Для проведения анализа определены следующие критерии сравнения функциональности предоставления данных: поиск товаров, фильтрация товаров, информация о конкретном товаре, избранные товары, корзина товаров.

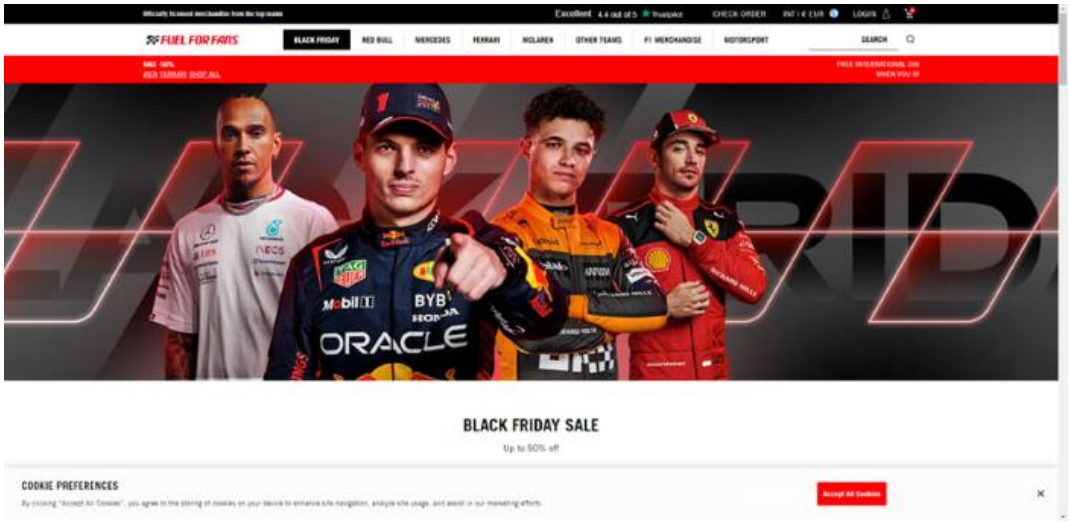

Рисунок 1 – Сайт www.fuelforfans.com

Сравнительный анализ уже существующих решений представлен в таблице 1.

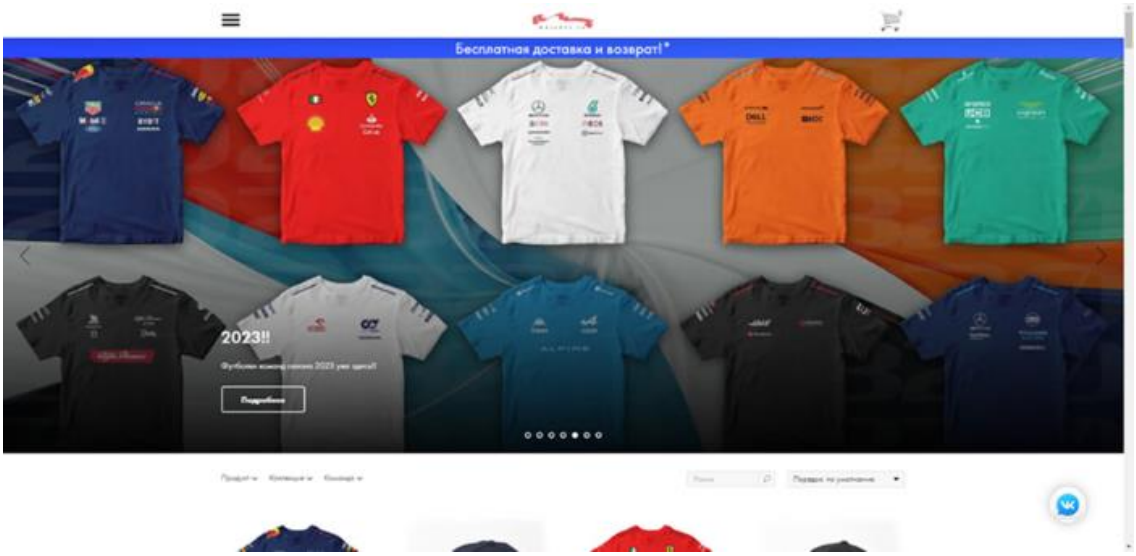

Рисунок 2 – Сайт merchf1.ru

Рисунок 3 – Сайт fueler.store

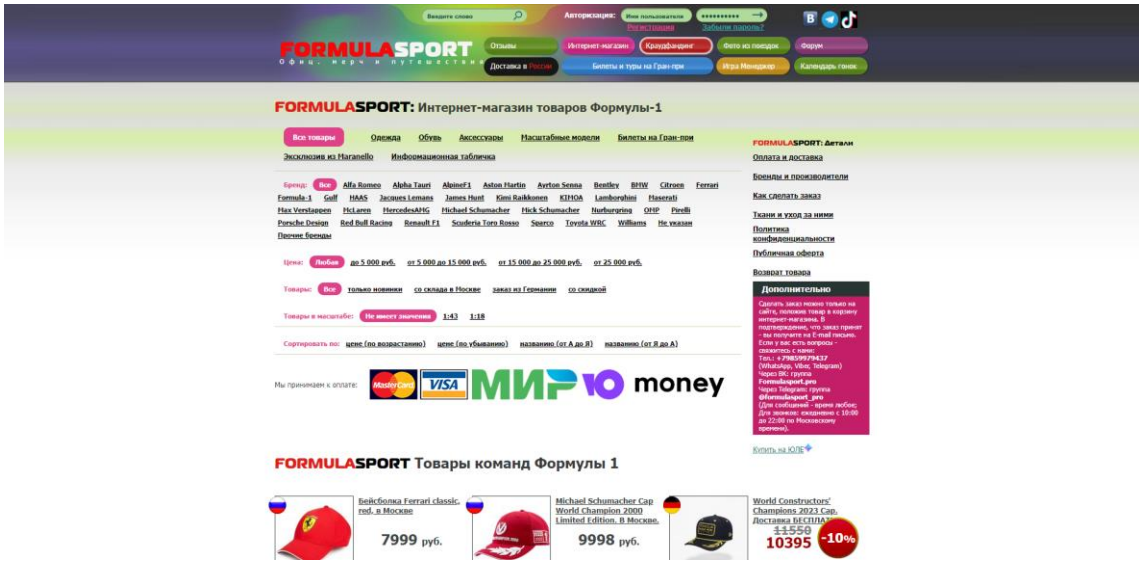

Рисунок 4 – Сайт formulasport.pro

Таблица 1 – Сравнительный анализ приложений для продажи фанаской атрибутики для Формулы-1

Сайт	Критерии				
	Поиск товаров	Фильтрация товаров	Информация о товаре	Избранные товары	Корзина товаров
1	Удобный и быстрый поиск, учитывающий тематику товаров.	Быстрая, функциональная фильтрация.	Информативная быстрая выборка данных.	Отсутствует возможность добавлять товары в избранное.	Многофункциональная, быстрая корзина.
2	Удобный и быстрый поиск, не учитывающий тематику товаров.	Быстрая, но мало функциональная фильтрация.	Мало информативная быстрая выборка данных.	Отсутствует возможность добавлять товары в избранное.	Многофункциональная, быстрая корзина.
3	Удобный и быстрый поиск, учитывающий тематику товаров.	Быстрая, функциональная фильтрация, иногда возникают ошибки.	Информативная быстрая выборка данных.	Отсутствует возможность добавлять товары в избранное.	Отсутствует возможность быстрого удаления многочисленного товара. Высокая скорость работы.
4	Неудобный, медленный поиск, периодические ошибки.	Мало функциональная фильтрация, ошибки при выборе некоторых фильтров.	Мало информативная быстрая выборка данных, ошибки при получении некоторых товаров.	Отсутствует возможность добавлять товары в избранное.	Отсутствует возможность изменения числа товаров в корзине, нет быстрого перехода после добавления товара. Низкая скорость работы.

По результатам сравнительного анализа существующих приложений видно, что самыми востребованными функциями являются:

– поиск товаров,

– фильтрация товаров,

– просмотр подробной информации о конкретном товаре,

– многофункциональная корзина товаров.

Также стоит отметить функцию добавления товаров в список желаемого (избранное), которая отсутствует в представленных решениях, но является востребованной для пользователей.

Приложением, объединяющим в себе весь этот функционал, должна стать разрабатываемая система «Мир Формулы-1».

1.2 Структура базы данных

Для проектирования системы необходимо определить основные модели данных приложения.

Таблица users базы данных содержит следующие поля: email, password (зашифрованный пароль), id – уникальный идентификатор. Она используется для регистрации и авторизации пользователей и привязки данных других таблиц к определенному аккаунту.

Таблица products базы данных содержит следующие поля: id – уникальный идентификатор, name – название товара, description – описание товара, image_link – ссылка на изображение, category – категория товара, price – цена товара, amount – кол-во доступных товаров. Эта таблица используется для хранения информации о товарах и их связи с другими сущностями в БД.

Таблица wishlist_products содержит следующие столбцы: user_id – идентификатор пользователя, product_id – идентификатор товара. Она хранит данные о связи пользователя с избранными им товарами.

Таблица cart_products содержит следующие столбцы: user_id – идентификатор пользователя, product_id – идентификатор товара, amount – кол-во товаров. Она хранит данные о связи пользователя с товаровами, добавленными им в корзину, а также о кол-ве каждого товара.

Таблица orders базы данных содержит следующие поля: id – уникальный идентификатор, user_id – идентификатор пользователя, совершившего заказ, а также данные, которые он указывает при оформлении заказа. Она используется для хранения информации о заказах и их связи с пользователями.

Таблица order_products базы данных содержит следующие поля: order_id – идентификатор заказа, к которому относится запись, product_id – идентификатор товара, price – цена, по которой был заказан товар, amount – в каком количестве был заказан товар. Она используется для хранения товаров, заказанных в рамках определенного заказа.

Схема базы данных приложения представлена на рисунке 5.

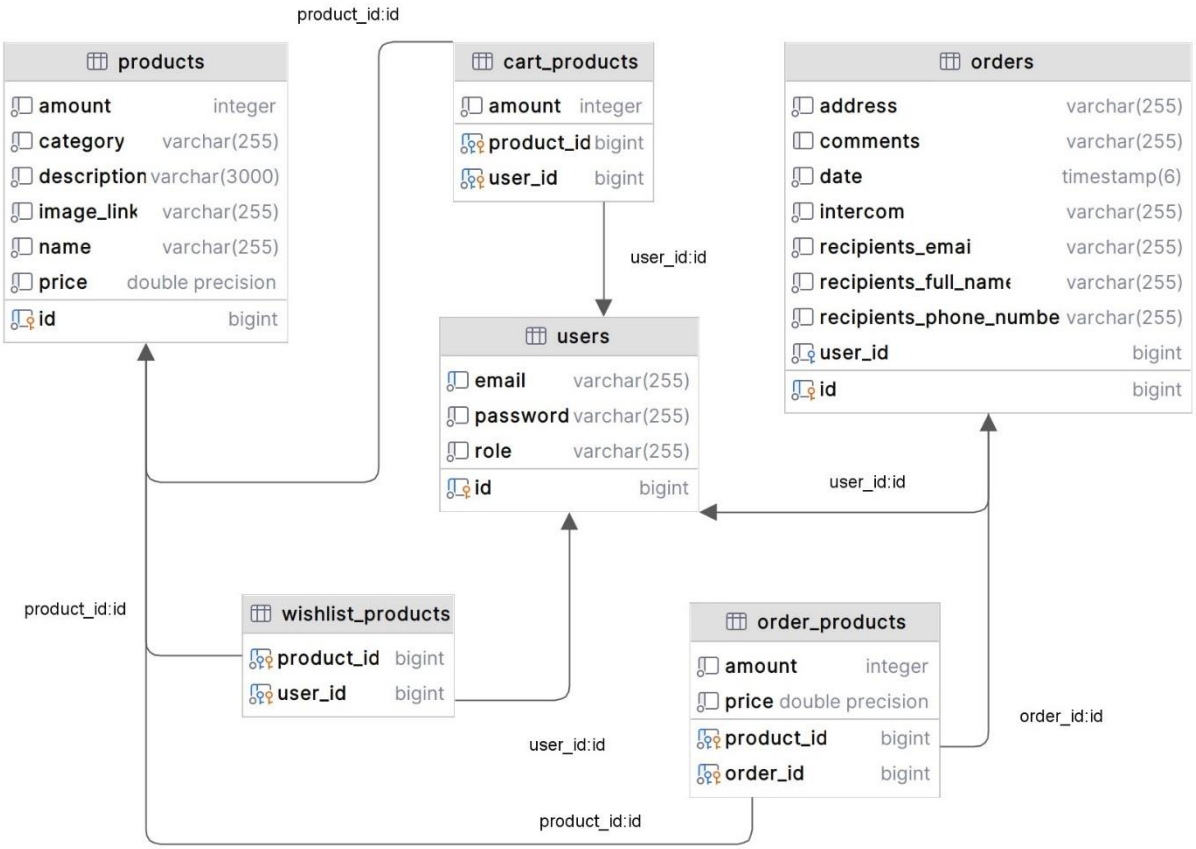

Рисунок 5 – Схема базы данных приложения

2 ОБОСНОВАНИЕ ВЫБОРА ТЕХНОЛОГИЙ

2.1 Используемое прикладное программное обеспечение

Для работы над курсовой работой использовалось следующее прикладное программное обеспечение: IntelliJIDEA, WebStorm, Postman, Yandex Browser, Git, Docker.

Использование IntelliJ IDEA в курсовой работе обосновано ее глубокой интеграцией с Java-экосистемой, наличием встроенных инструментов для работы с БД [5]. Выбор WebStorm обусловлен его оптимизацией для работы с JavaScript, CSS, HTML и обширным функционалом для работы с веб-приложениями [6]. Обе IDE обладают расширенными возможности рефакторинга и отладки, предоставляют умную поддержку кодирования, используя ИИ для автодополнения кода, анализа качества и поиска возможной оптимизации, обладают широким набором плагинов для расширения функциональности для различных технологий и фреймворков.

Использование Postman обусловлено его обширными возможностями тестирования API в интуитивно понятном интерфейсе, а также возможностью автоматизации тестирования [7].

Yandex Browser был выбран для работы из-за поддержки современных веб-стандартов, высокой скорости и безопасности, мощными инструментами для разработчика, а также встроенным ИИ-инструментам для перевода документации.

Использование Git обусловлено удобством контроля версий приложения, возможностью работать на нескольких устройствах, а также его глубокой интеграцией в IntelliJIDEA и WebStorm.

Docker был выбран для упрощения процесса настройки и деплоя приложения [8] и изоляции его и его зависимостей от остальной системы, глубокой поддержкой со стороны IDE.

Также стоит отметить, что все выбранное ПО может быть установлено бесплатно.

2.2 Используемые технологии

Использование Java в курсовой работе обусловлено ее надежностью, возможностью запуска на любой системе, поддерживающей JVM, большим количеством технологий и фреймворков созданных для написания кода на ней. Также Java – один из самых распространенных языков программирования и поддерживается огромным сообществом разработчиков, что ускоряет процесс решения возникающих проблем быстрым нахождением необходимой информации.

Выбор Gradle в качестве системы автоматической сборки обусловлен тем, что она предоставляет гибкое декларативное программирование и позволяет исполнять код на Groovy или Kotlin в своих конфигурационных скриптах. Это дает большую степень контроля и адаптируемости. Gradle позволяет создавать многомодульные проекты с легко конфигурируемыми зависимостями и простыми управлением жизненным циклом проекта и его сборкой.

Spring Boot — фреймворк, заметно ускоряющий процесс разработки за счет предоставления готовых конфигураций, которые внедряются по умолчанию, упрощая начало работы с приложением без траты времени на подключение всех необходимых зависимостей [9]. Наличие встроенных серверов приложений, таких как Tomcat, означает, что приложение можно запускать без необходимости развертывания на внешние серверы. Это упрощает разработку и тестирование, и в то же время повышает эффективность процесса разработки.

Spring Data JPA — ключевой компонент экосистемы Spring, который упрощает реализацию слоя доступа к данным, автоматизируя рутинные задачи, связанные с CRUD-операциями и другими шаблонными запросами, такими как поиск записи по значению, тем самым позволяя сосредоточиться на более сложной бизнес-логике [10]. С его помощью удается избежать кропотливой работы по написанию и поддержке большого количества SQL-запросов, что существенно повышает скорость разработки. Также Spring Data

JPA поддерживает язык запросов JPQL, который позволяет писать сложные запросы, упрощая связи между таблицами, что также сильно упрощает работу со связанными сущностями в базах данных.

Использование PostgreSQL оправдано его мощными возможностями управления транзакциями, расширенным SQL-синтаксисом и поддержкой JSON, что обеспечивает высокую производительность в обработке сложных запросов и гибкость в работе с данными различных форматов. Также стоит отметить удобные настройки транзакций по умолчанию для Spring Data JPA, которые понижают вероятность возникновения ошибок при работе с данными.

Spring Validation обеспечивает строгую проверку данных на стороне сервера с помощью декларативного подхода и аннотаций, способствующих сокращению количества кода, необходимого для проверки целостности и правильности входящих данных, что увеличивает надежность приложения и уменьшает вероятность возникновения ошибок в результате некорректного ввода, позволяя не тратить время на написание простых правил валидации DTO-объектов [11].

Spring Security вводит механизмы защиты на уровне инфраструктуры приложения, реализуя комплексный подход к аутентификации и авторизации пользователей, а также предоставляя стандартные средства для обеспечения безопасности веб-приложений, такие как CSRF-защита и управление сеансами, шифрование данных, что повышает степень защищенности системы в целом.

JSON Web Tokens (JWT) являются ключевым компонентом стратегии защиты, так как позволяют безопасно передавать информацию о состоянии аутентификации и авторизации между клиентом и сервером, что упрощает распределение систем и уменьшает количество состояний, которые необходимо управлять на сервере, результатом чего является повышение расширяемости и масштабируемости системы [12]. Также JWT для существуют удобные настройки в Spring Security, что упрощает их конфигурацию.

Выбор Lombok обусловлен стремлением минимизировать рутинный и шаблонный код, который зачастую несет в себе высокий потенциал для ошибок из-за повторяемости операций. Lombok, используя аннотации, автоматизирует генерацию стандартных методов, таких как getters, setters и других, уменьшая объем исходного кода и повышая его читабельность без ущерба для функциональности.

JUnit, представляющий собой фреймворк для модульного тестирования, позволяет обеспечить высокое качество кода за счет раннего выявления проблем и ошибок. С его помощью разработчики могут не только тестировать отдельные блоки кода в изоляции от внешних зависимостей, но и создавать автоматические тесты, повышая тем самым стабильность и надежность программного обеспечения.

Выбор React JS как инструментария для разработки пользовательских интерфейсов обосновывается его способностью создавать высокопроизводительные, реактивные и модульные веб-приложения. Благодаря виртуальному DOM, React обеспечивает эффективное обновление и рендеринг компонентов, что особенно важно в сценариях, требующих быстрого отклика пользовательского интерфейса и динамического взаимодействия с пользователем.

Использование библиотеки React Router DOM целесообразно для реализации маршрутизации в одностраничных приложениях (SPA), созданных с использованием React. Она обеспечивает клиентскую маршрутизацию, которая позволяет приложению реагировать на изменения URL без перезагрузки страницы, предоставляя механизмы навигации и группировки компонентов в соответствии с различными сетевыми маршрутами.

Axios является популярной HTTP-клиентской библиотекой, которая специализируется на выполнении XMLHttpRequests из веб-браузера. Это инструмент, оптимизированный для работы с обещаниями (promises) в JavaScript, позволяя разработчикам осуществлять асинхронные HTTP-запросы для взаимодействия с REST API. Axios отличается удобным API, предоставляя

возможность глобальной настройки запросов, перехвата и трансформации данных запросов и ответов, что очень важно при взаимодействии с серверной частью приложения [13].

3 РАЗРАБОТКА

3.1 Структура проекта

Структура проекта с указанным стеком технологий предполагает четкое разделение ответственности между фронтендом и бэкендом (рисунок 6), а также внутри каждого из этих уровней.

На стороне клиента используется React, что делает интерфейс приложения реактивным и пользовательски ориентированным, легко адаптируясь к изменениям данных в реальном времени.

Серверная часть разработана на основе Spring Boot, что позволяет быстро и эффективно создать структурированное серверное приложение с минимальной конфигурацией. Для работы с данными используется PostgreSQL.

Интеграция и развертывание всех компонентов приложения упрощаются с использованием Docker-compose, который позволяет описать и запустить многоконтейнерные приложения Docker.

Также в общем слое проекта описан файл .gitignore для конфигурации отслеживаемых файлов git.

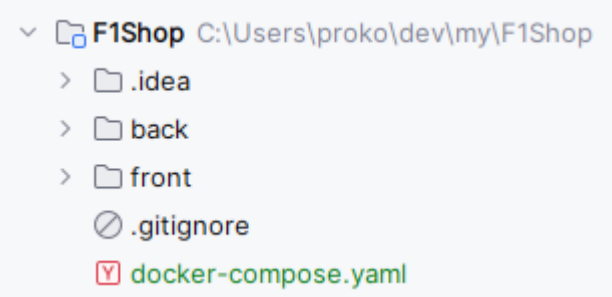

Рисунок 6 – Общая структура приложения

3.2 Разработка серверной части приложения

3.2.1 Структура серверной части приложения

Структура бэкенда в проекте, использующем Spring Boot, представляет собою многоуровневую архитектуру, которая разделяет приложение на четко определенные уровни ответственности, обеспечивая тем самым порядок в кодовой базе и упрощение тестирования и поддержки.

Слой контроллеров (controller) служит в качестве внешнего интерфейса между пользователем и системой, обрабатывая входящие HTTP запросы, действуя как первая точка контакта для клиентских операций. В данном слое применяется Spring Validation для проверки корректности входящих данных, что позволяет отсеивать невалидные запросы до того, как они достигнут более глубоких слоёв бизнес-логики, сэкономив тем самым ресурсы и время на обработку.

Слой сервисов (service) отвечает за бизнес-логику приложения. Здесь сосредоточены классы и методы, ответственные за выполнение задач, связанных с логикой предметной области. Этот слой отвечает за принятие решений, обработку данных, полученных от контроллеров, и последующее взаимодействие со слоем репозиториев для сохранения или изменения данных в базе данных.

Слой репозиториев (repository) предоставляет абстракцию доступа к данным, позволяя сервисному слою взаимодействовать с базой данных без необходимости знания о конкретной реализации. С использованием Spring Data JPA этот уровень упрощает создание DAO слоя и написание запросов в базу данных, автоматически реализуя многие стандартные операции и превращая доступ к данным в простой и чистый процесс. PostgreSQL используется в качестве СУБД для управления базами данных.

В проекте также применяются сущности (entity), которые являются отражением таблиц базы данных на уровне объектов и определяют структуру хранения данных в приложении.

Data Transfer Objects (DTO) помогают в организации передачи данных между клиентом и сервером [14]. Они идеально подходят для инкапсуляции и передачи данных, особенно когда необходимо ограничить объем или формат данных, подаваемых на вход или вывод из API.

Мапперы (mapper) используются для преобразования данных между объектами сущностей и DTO.

Также в проекте есть файлы конфигураций (configuration) для настройки взаимодействия бинов между собой.

В целом классы серверной части приложения разбиты по пакетам относительно функционала, за который они отвечают, а уже внутри них на слои. Это упрощает поиск файлов внутри проекта. Когда появляется новый функционал, классы для работы с ним выносятся в отдельный пакет, не загромождая при этом уже существующие директории.

Структура серверной части приложения представлена на рисунке 7.

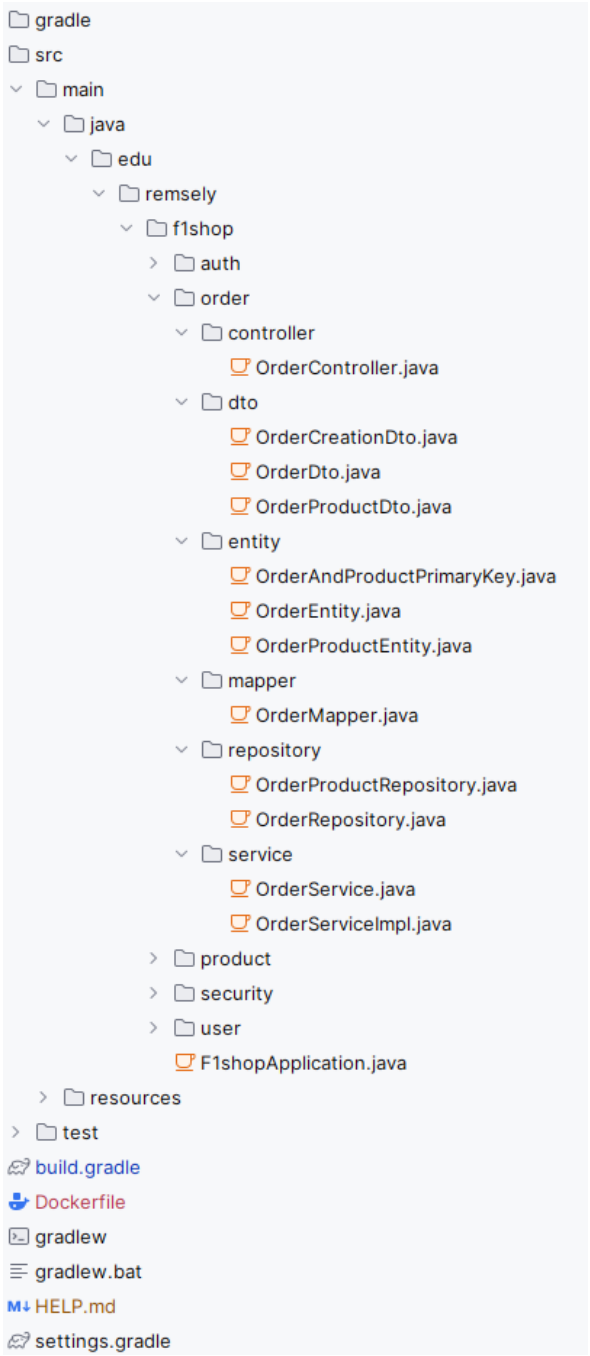

Рисунок 7 – Структура серверной части приложения

3.2.2 Файл application.yml

В файле application.yaml в проектах на Spring Boot определяются глобальные настройки приложения. Он используется для удобной организации параметров подключения к БД, безопасности, JPA и других важных аспектов работы приложения. Файл представлен на рисунке 8.

```yaml
security:
  jwt:
    secret-sey: verysecretkeydonttellanybodyimserious

spring:
  datasource:
    driver-class-name: org.postgresql.Driver
    url: jdbc:postgresql://localhost:5432/f1shop
    username: test
    password: test

  jpa:
    hibernate:
      ddl-auto: update
    show-sql: false
    open-in-view: false
```

Рисунок 8 – Файл application.yaml

3.2.3 Класс WebSecurityConfig

Класс WebSecurityConfig отвечает за настройку параметров безопасности веб-приложения на Spring, активируя фильтры для аутентификации, управление CORS, сессиями и шифрованием паролей. Он определяет правила обработки входящих HTTP запросов, разрешает кросс-доменные запросы и устанавливает политику безопасности сессий, подходящую для REST API. Этот класс является необходимым для обеспечения безопасной работы веб-приложения. Класс представлен на рисунке 9.

```java
@Configuration  Remsely
@EnableWebSecurity
@EnableWebMvc
@RequiredArgsConstructor
public class WebSecurityConfig implements WebMvcConfigurer {
    @Bean  Remsely
    public SecurityFilterChain applicationSecurity(HttpSecurity http,
                                                   JwtAuthenticationFilter jwtAuthenticationFilter) throws Exception {
        http.addFilterBefore(jwtAuthenticationFilter, UsernamePasswordAuthenticationFilter.class);
        http.cors(Customizer.withDefaults())
                .csrf(AbstractHttpConfigurer::disable)
                .sessionManagement(sessionManagement ->
                        sessionManagement.sessionCreationPolicy(SessionCreationPolicy.STATELESS)
                ).formLogin(AbstractHttpConfigurer::disable)
                .securityMatcher("/**")
                .authorizeHttpRequests(registry -> registry
                        .requestMatchers("auth/login").permitAll()
                        .requestMatchers("auth/register").permitAll()
                        .anyRequest().authenticated()
                );
        return http.build();
    }

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

    @Bean  Remsely
    public AuthenticationManager authenticationManager(CustomUserDetailService userDetailService) {
        DaoAuthenticationProvider daoAuthenticationProvider = new DaoAuthenticationProvider();
        daoAuthenticationProvider.setUserDetailsService(userDetailService);
        daoAuthenticationProvider.setPasswordEncoder(passwordEncoder());
        return new ProviderManager(daoAuthenticationProvider);
    }

    @Bean  Remsely
    public CorsConfigurationSource corsConfigurationSource() {
        CorsConfiguration configuration = new CorsConfiguration();
        configuration.setAllowedOrigins(List.of( e1: "*"));
        configuration.setAllowedMethods(List.of( e1: "*"));
        configuration.setAllowedHeaders(List.of( e1: "*"));
        UrlBasedCorsConfigurationSource source = new UrlBasedCorsConfigurationSource();
        source.registerCorsConfiguration( pattern: "/**", configuration);
        return source;
    }
}
```

Рисунок 9 – Реализация класса WebSecurityConfig

3.2.4 Класс сервиса AuthServiceImpl

Класс AuthSrviceImpl служит элементом системы, отвечающим за регистрацию и авторизацию пользователей и выдачу им JWT токенов при помощи AuthenticationManager и JwtIssuer для авторизации в запросах остальных серверов. Он осуществляет работу с данными пользователей через userService. Функционал AuthSrviceImpl представлен на рисунке 10.

```java
@Slf4j  Remsely
@Service
@RequiredArgsConstructor
public class AuthServiceImpl implements AuthService {
    private final JwtIssuer jwtIssuer;
    private final AuthenticationManager authenticationManager;
    private final UserService userService;

    @Override  1 usage  Remsely
    public LoginResponse login(LoginRequest request) {
        final String email = request.getEmail();
        if (!userService.userExistByEmail(email)) {
            log.info("Login failed. No user found with email {}!", email);
            return LoginResponse.builder()
                    .accessToken(null)
                    .build();
        }
        return getToken(request);
    }

    @Override  1 usage  Remsely
    public LoginResponse register(LoginRequest request) {
        userService.addUser(User.builder()
                .email(request.getEmail())
                .password(request.getPassword())
                .build());
        return getToken(request);
    }

    private LoginResponse getToken(LoginRequest request) {  2 usages  Remsely
        var authentication = authenticationManager.authenticate(
                new UsernamePasswordAuthenticationToken(request.getEmail(), request.getPassword())
        );
        SecurityContextHolder.getContext().setAuthentication(authentication);
        UserPrincipal principal = (UserPrincipal) authentication.getPrincipal();

        List<String> roles = principal.getAuthorities().stream()  Stream<capture of extends GrantedAuthority>
                .map(GrantedAuthority::getAuthority)  Stream<String>
                .toList();

        var token = jwtIssuer.issue(principal.getUserId(), principal.getEmail(), roles);

        log.info("Login successful. User email : {}.", request.getEmail());
        return LoginResponse.builder()
                .accessToken(token)
                .build();
    }
}
```

Рисунок 10 – Реализация AuthServiceImpl

3.2.5 Класс сервиса UserServiceImpl

Класс UserServiceImpl используется в системе для сохранения и поиска пользователей в базе данных. Он осуществляет работу с данными пользователей через интерфейс userStorage унаследованный от JpaRepository. Также необходимо отметить, что на UserServiceImpl также лежит ответственность за хеширование паролей пользователей. Функционал Этого сервиса представлен на рисунке 11.

```java
@Slf4j  Remsely
@Service
@RequiredArgsConstructor
public class UserServiceImpl implements UserService {
    private final UserRepository userRepository;
    private final PasswordEncoder passwordEncoder;

    @Transactional(readOnly = true)  1 usage  Remsely
    @Override
    public User findUserByEmail(String email) {
        User user = userRepository.findByEmailIgnoreCase(email)
                .orElseThrow(() -> new RuntimeException("User not found"));
        log.info("User found by email successful. User : {} ", user);
        return user;
    }

    @Transactional  1 usage  Remsely
    @Override
    public User addUser(User user) {
        user.setRole(UserRole.USER);
        user.setPassword(passwordEncoder.encode(user.getPassword()));
        User savedUser = userRepository.save(user);
        log.info("User saved successful. User {}", savedUser);
        return savedUser;
    }

    @Transactional(readOnly = true)  1 usage  Remsely
    @Override
    public boolean userExistByEmail(String email) {
        boolean exists = userRepository.existsByEmailIgnoreCase(email);
        log.info("User existence by email {} has been checked. existence : {} ", email, exists);
        return exists;
    }
}
```

Рисунок 11 – Реализация UserServiceImpl

3.2.6 Класс сервиса ProductServiceImpl

Класс ProductServiceImpl служит в качестве сервисного компонента для управления данными о товарах, добавления их в список избранного пользователем в приложении, используя интерфейсы-репозитории, унаследованные от JpaReposotory. Класс ProductServiceImpl инкапсулирует логику и обеспечивает абстракцию для работы с базой данных для контроллера, что повышает модульность и расширяемость кода. Часть логики класса представлена на рисунках 12 и 13.

```java
@Slf4j  Remsely
@Service
@RequiredArgsConstructor
public class ProductServiceImpl implements ProductService {
    private final ProductRepository productRepository;
    private final WishlistRepository wishlistRepository;
    private final UserRepository userRepository;
    private final CartRepository cartRepository;
    private final ProductMapper productMapper;

    @Transactional  1 usage  Remsely
    @Override
    public Product addProduct(Product product) {
        Product newProduct = productRepository.save(product);
        log.info("New product added. Product : {}", newProduct);
        return newProduct;
    }

    @Transactional  1 usage  Remsely
    @Override
    public Product updateProduct(Product product, long productId) {
        Product productToUpdate = findProduct(productId);

        updateNonNullProperties(productToUpdate, product);
        productRepository.save(productToUpdate);

        log.info("Product updated. Product : {}", productToUpdate);
        return productToUpdate;
    }

    @Transactional(readOnly = true)  3 usages  Remsely
    @Override
    public ProductDto getProduct(long productId, long userId) {
        User user = findUser(userId);
        Product product = findProduct(productId);

        boolean inWishlist = isProductInWishlist(product, user);
        boolean inCart = isProductInCart(product, user);

        log.info("Product found. Product : {}", product);
        return productMapper.toDto(product, inWishlist, inCart);
    }
```

Рисунок 12 – Реализация CRUD-операций над товаром в ProductServiceImpl

```java
@Transactional(readOnly = true)  1 usage  Remsely
@Override
public List<CartEntity> getProductsFromCart(long userId) {
    User user = findUser(userId);
    List<CartEntity> cartEntities = cartRepository.findByUser(user);
    log.info("User's with id {} cart found. List length : {}", userId, cartEntities.size());
    return cartEntities;
}
```

Рисунок 13 – Метод получения товаров в корзине в ProductServiceImpl

3.2.7 Класс сервиса OrderServiceImpl

Класс OrderServiceImpl служит элементом системы, отвечающим за обработку информации по поступающим со стороны пользователей заказам. Он осуществляет связь с базой данных через наследуемые от JpaRepository интерфейсы для доступа к данным. OrderServiceImpl скрывает комплексные операции с данными за простым интерфейсом, что способствует легкости интеграции и возможности масштабирования приложения. Отдельные аспекты его функционала продемонстрированы на рисунке 14.

```
@Service  Remsely
@RequiredArgsConstructor
public class OrderServiceImpl implements OrderService {
    private static final Logger log = LoggerFactory.getLogger(OrderServiceImpl.class);  3 usages
    private final OrderRepository orderRepository;
    private final UserRepository userRepository;
    private final CartRepository cartRepository;
    private final OrderProductRepository orderProductRepository;
    private final ProductRepository productRepository;
    private final OrderMapper orderMapper;

    @Transactional  1 usage  Remsely
    @Override
    public OrderDto createOrder(OrderEntity order, long userId) {
        User user = findUser(userId);
        order.setUser(user);

        List<CartEntity> cartEntities = cartRepository.findByUser(user);

        if (cartEntities.isEmpty())
            throw new RuntimeException("Cart Is Empty!");

        OrderEntity savedOrder = orderRepository.save(order);
        Set<OrderProductEntity> orderProductEntities = getCartProductsIds(cartEntities, savedOrder);

        List<Product> productsToUpdateCount = orderProductEntities.stream()
                .map(orderProductEntity -> {
                    Product product = orderProductEntity.getProduct();
                    int newAmount = product.getAmount() - orderProductEntity.getAmount();

                    if (newAmount < 0)
                        throw new RuntimeException("No enough product amount!");

                    product.setAmount(newAmount);
                    return product;
                }).toList();

        productRepository.saveAll(productsToUpdateCount);
        List<OrderProductEntity> savedOrderProducts = orderProductRepository.saveAll(orderProductEntities);
        cartRepository.deleteAll(cartEntities);

        log.info("User's with id {} order added. Order length : {}", userId, savedOrderProducts.size());
        return orderMapper.toDto(savedOrder, orderMapper.toDtoList(savedOrderProducts));
    }
```

Рисунок 14 – Реализация оформления заказа пользователем в OrderServiceImpl

3.3 Разработка клиентской части приложения

Фронтенд-часть приложения, реализованная с использованием библиотеки React.js, обеспечивает интерактивность пользовательского интерфейса в соответствии с концепциями современных одностраничных приложений.

Для организации маршрутизации внутри приложения используется библиотека React Router DOM, которая представляет собой стандартный набор компонентов для имплементации декларативной маршрутизации. Это позволяет не только определять переходы между различными представлениями и их адресацией в контексте одностраничного приложения, но и обеспечивает динамическую загрузку соответствующих компонентов без необходимости перезагрузки страницы, что способствует повышению общей производительности и улучшению пользовательского опыта.

Ключевую роль в организации взаимодействия фронтенда с внешним API играют сервисы, выполненные с применением axios — популярного HTTP-клиента. Axios выделяется лёгкостью интеграции, возможностью глобальной конфигурации и промис-основанным API, что в комплексе позволяет разработчикам создавать асинхронные HTTP-запросы к серверу для отправки и получения данных.

Фронтенд-архитектура написанная на React.js с использованием React Router DOM, и axios представляет собой мощную и гибкую структуру для создания масштабируемых и легко поддерживаемых веб-приложений.

Ниже представлено описание основных страниц приложения.

3.3.1 Страницы авторизации и регистрации

На данных страницах представлены классические формы регистрации и авторизации, позволяющие пользователю войти в систему или зарегистрироваться в ней. Страницы представлены на рисунках 15-16.

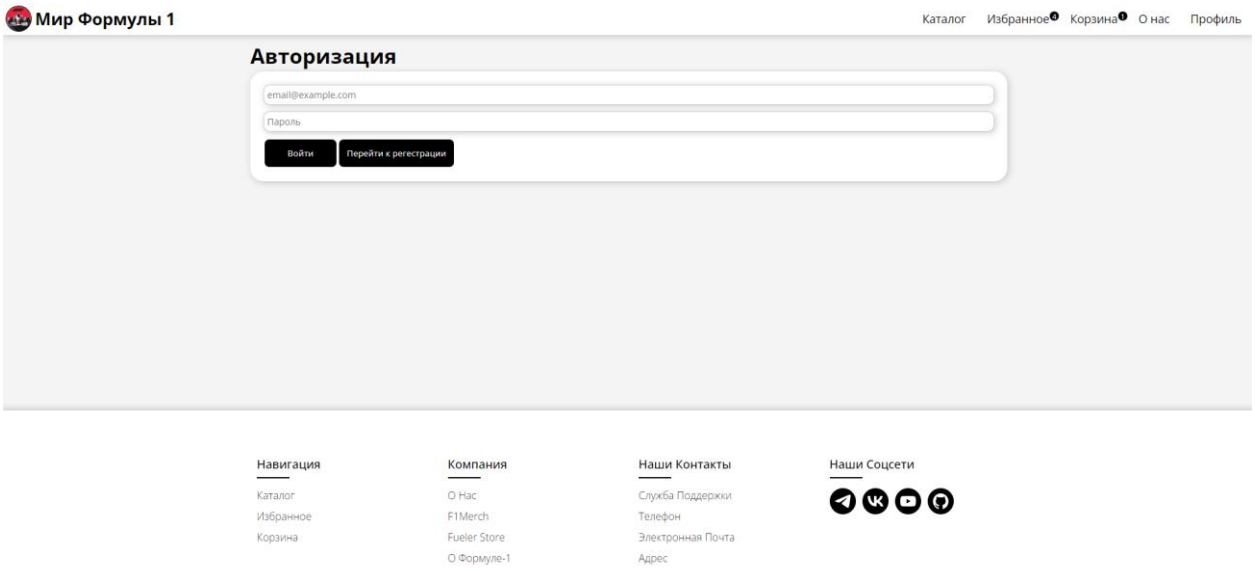

Рисунок 15 – Страница авторизации

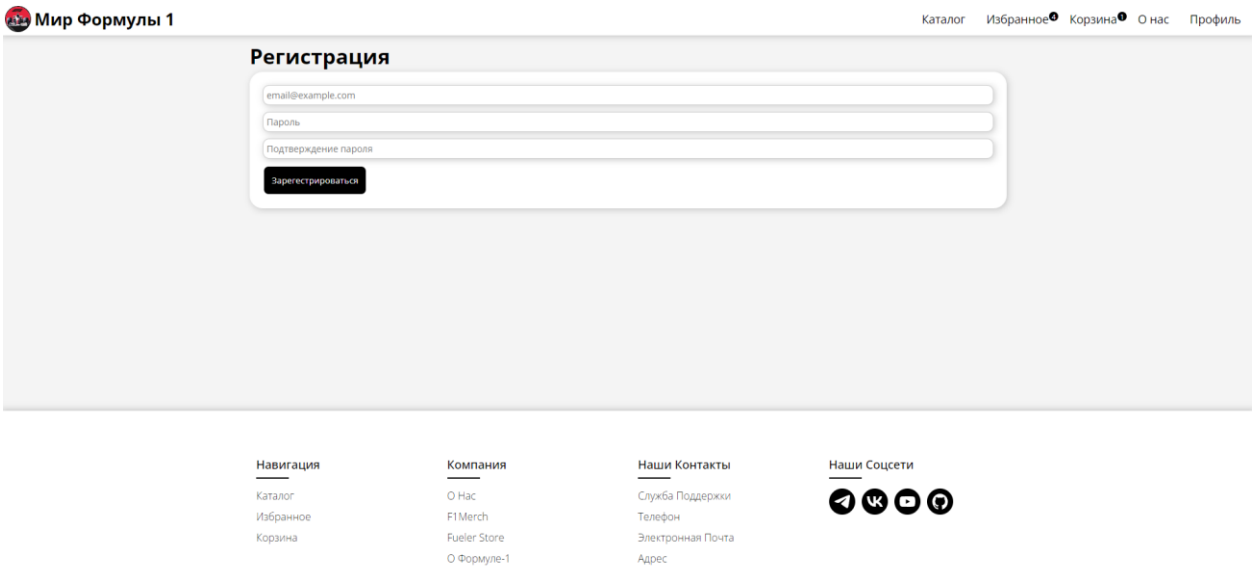

Рисунок 16 – Страница регистрации

3.3.2 Страница каталога

На рисунке 17 изображена страница разработанного веб-ресурса, на которой можно увидеть каталог товаров, который состоит из списка карточек товаров, и блока поиска, фильтрации и сортировки.

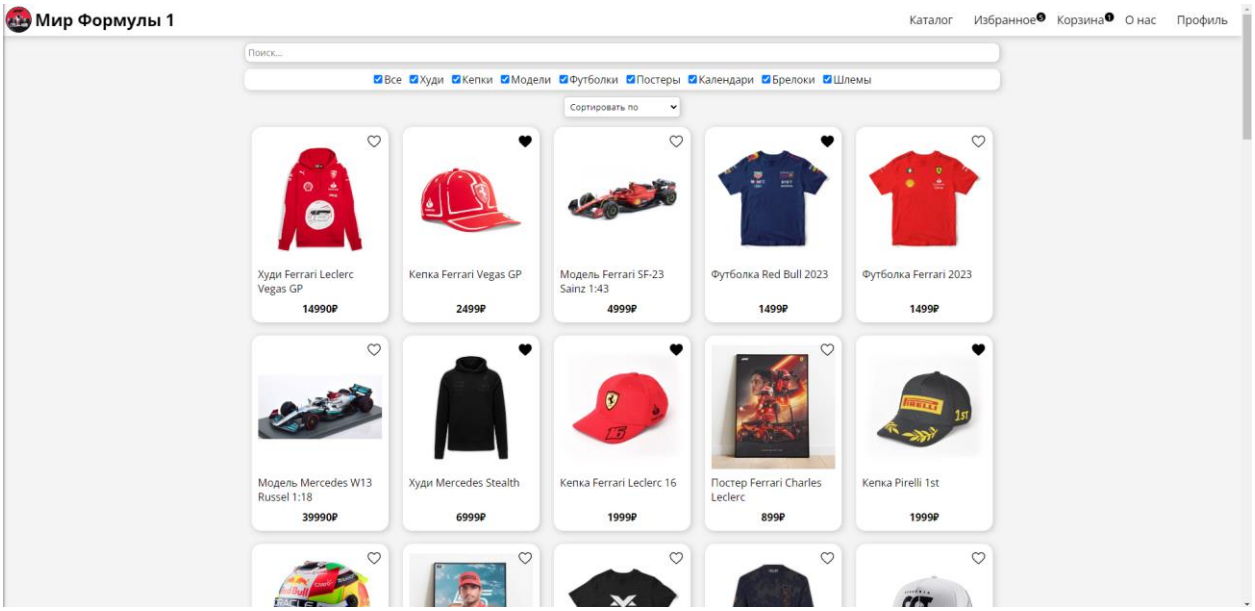

Рисунок 17 – Страница каталога

3.3.3 Страница корзины товаров

На рисунке 18 представлена страница избранных товаров, которые пользователь может сохранить, если они ему понравились.

Страница состоит из списка карточек избранных товаров. Товары можно удалять нажатием на иконку сердца.

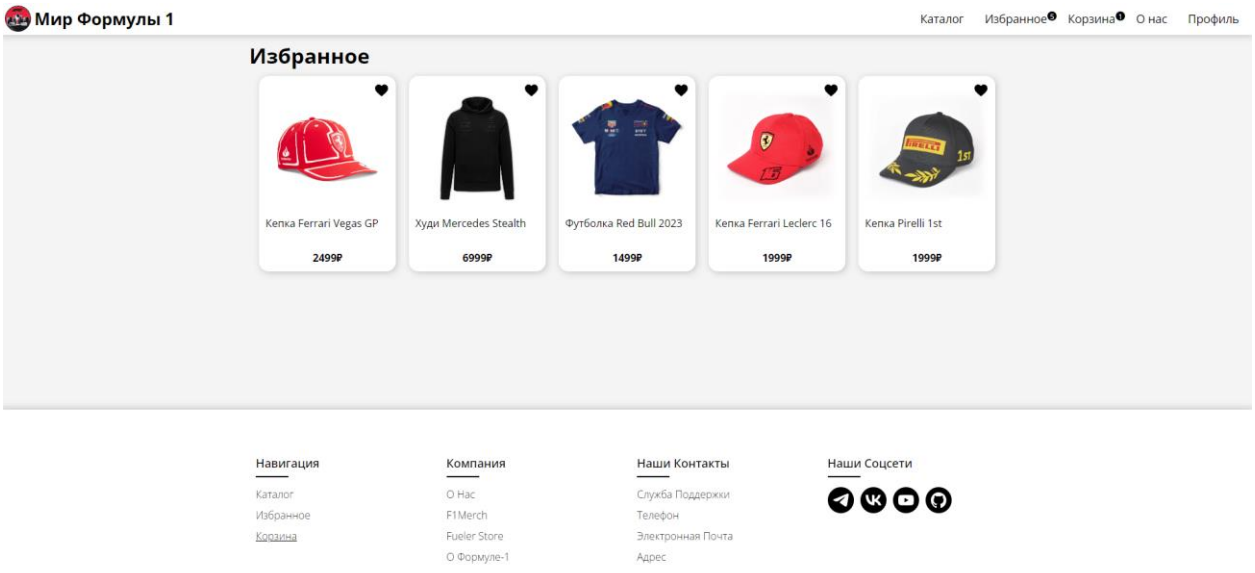

Рисунок 18 – Страница избранных товаров

3.3.4 Страница поиска туристических мест

На рисунках 19-21 представлена страница корзины товаров, на которой представлен список товаров в корзине с возможностью удаления, увеличения

и уменьшения их числа. Также в списке отображается информация о стоимости товаров и полная стоимость заказа, которая автоматически пересчитывается при любой манипуляции со списком.

Также на данной странице представлена форма, которую пользователь должен заполнить для оформления заказа.

После нажатия на кнопку «оформить заказ» пользователь получит сообщение об успешном оформлении заказа, которое проинформирует его о дальнейших действиях.

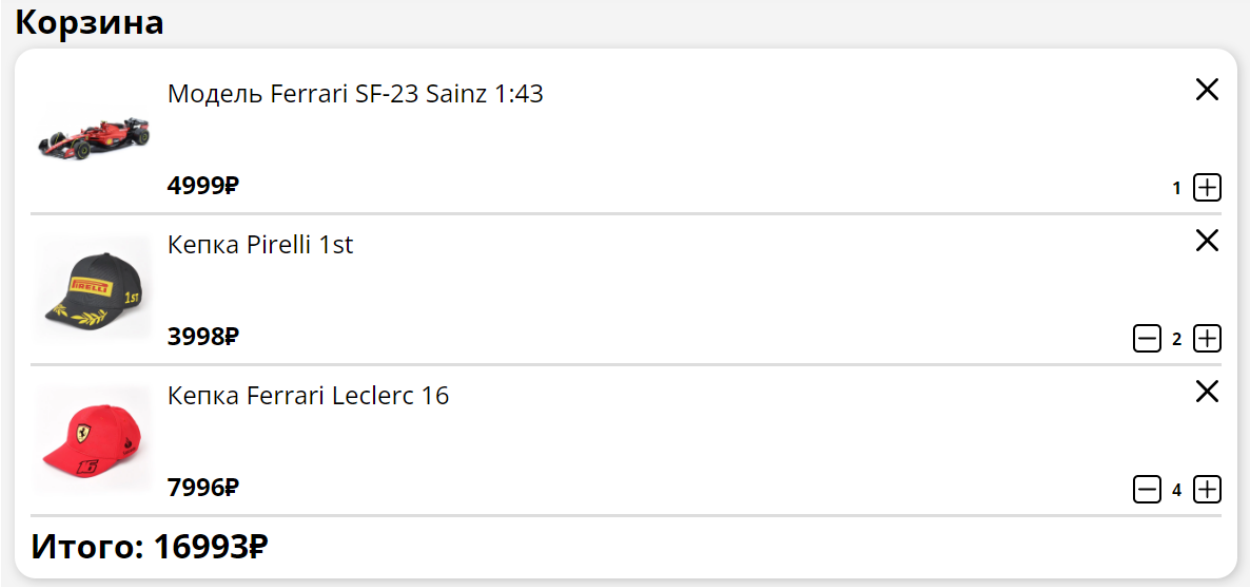

Рисунок 19 – Список элементов корзины

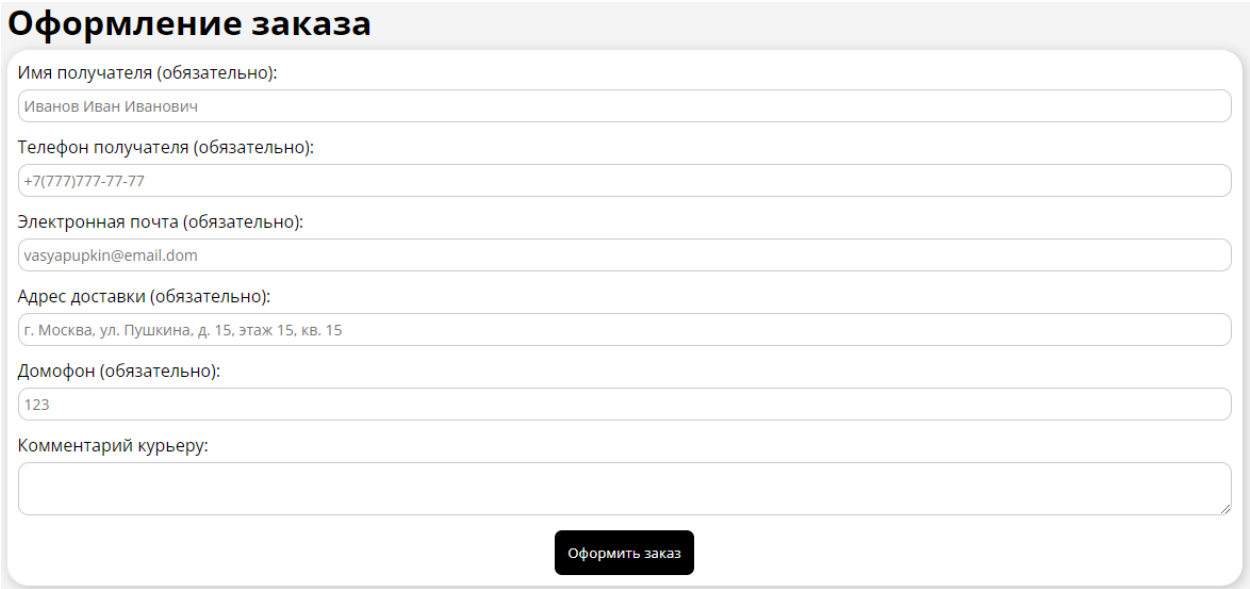

Рисунок 20 – Форма оформления заказа

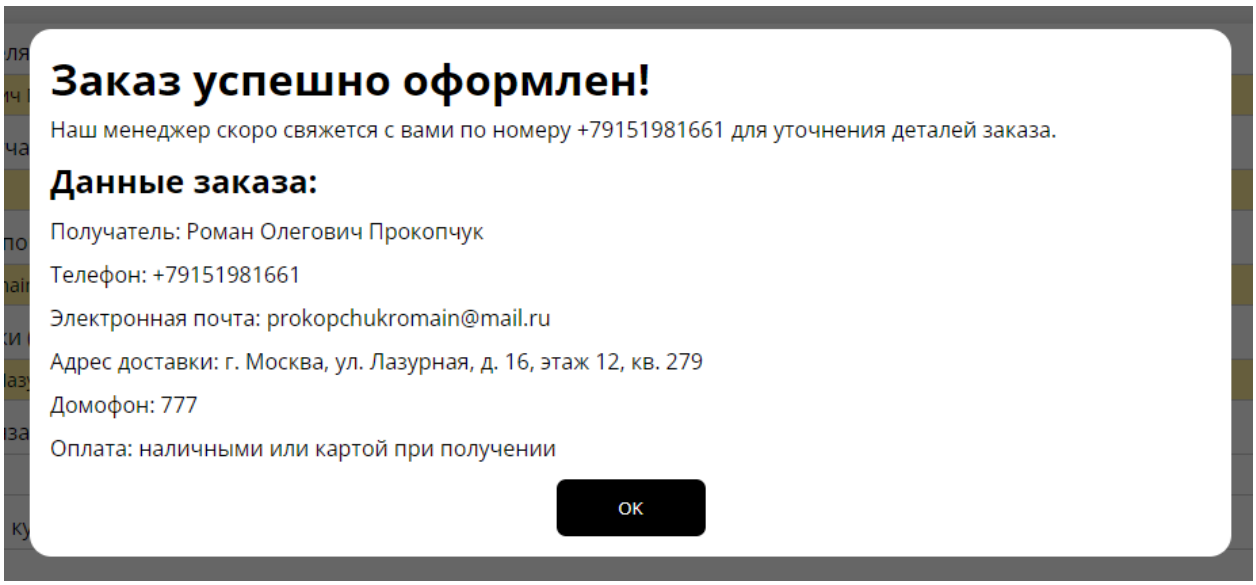

Рисунок 21 – Сообщение об успешном оформлении заказа

3.3.5 Страница информации о товаре

На рисунке 22 представлена страница, на которой пользователь может ознакомиться с информацией о товаре. На ней представлено фото товара, его описание, цена, кнопки добавления в избранное и в корзину, а также кнопка закрытия страницы, которая отправит пользователя на предыдущую страницу.

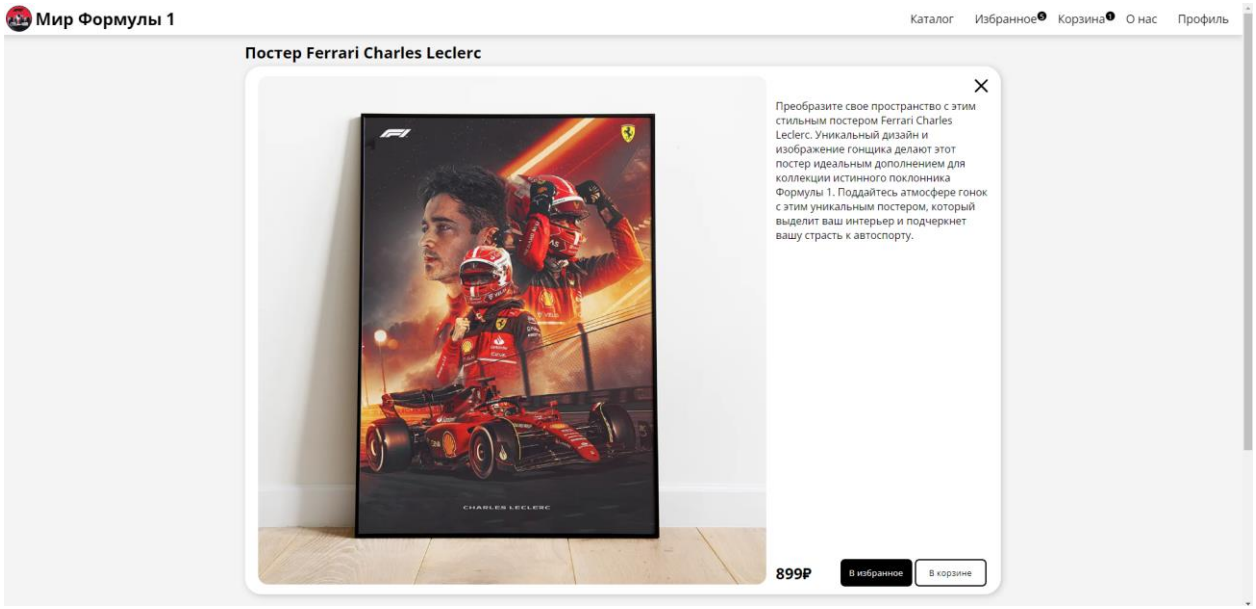

Рисунок 22 – Страница информации о товаре

3.4 Тестирование приложения

Для обеспечения корректности работы приложения и его компонентов разработаны unit-тесты с использованием фреймворка JUnit, которые позволяют проверить функциональность в изолированной среде. В

дополнение к стандартному юнит-тестированию, имитация взаимодействия с внешними зависимостями (такими как базы данных или веб-сервисы) выполнена с использованием mock-объектов с помощью библиотеки Mockito. Пример тестов представлен на рисунке 23, результат их работы – на рисунке 24.

```java
public class UserServiceTest {
    @Mock 10 usages
    private UserRepository userRepository;

    @Mock 2 usages
    private PasswordEncoder passwordEncoder;

    @InjectMocks 5 usages
    private UserServiceImpl userService;

    @BeforeEach
    public void init() {
        MockitoAnnotations.openMocks( testClass: this);
    }

    @Test
    public void testFindUserByEmailUserExists() {
        String email = "test@example.com";
        User mockUser = new User();
        mockUser.setEmail(email);

        when(userRepository.findByEmailIgnoreCase(email)).thenReturn(Optional.of(mockUser));

        User result = userService.findUserByEmail(email);
        assertEquals(mockUser, result);
        verify(userRepository).findByEmailIgnoreCase(email);
    }

    @Test
    public void testFindUserByEmailUserDoesNotExist() {
        String email = "nonexistent@example.com";

        when(userRepository.findByEmailIgnoreCase(email)).thenReturn( t: Optional.empty());

        assertThrows(RuntimeException.class, () -> userService.findUserByEmail(email));
        verify(userRepository).findByEmailIgnoreCase(email);
    }

    @Test
    public void testAddUserValidUser() {
        User newUser = new User();
        newUser.setEmail("newuser@example.com");
        newUser.setPassword("password123");
        newUser.setRole(UserRole.USER);

        when(passwordEncoder.encode(anyString())).then(returnsFirstArg());
        when(userRepository.save(any(User.class))).thenReturn(newUser);

        User savedUser = userService.addUser(newUser);

        assertNotNull(savedUser);
        assertEquals(newUser.getEmail(), savedUser.getEmail());
        verify(userRepository).save(newUser);
        verify(passwordEncoder).encode( rawPassword: "password123");
    }
```

Рисунок 23 – Тестирование UserService

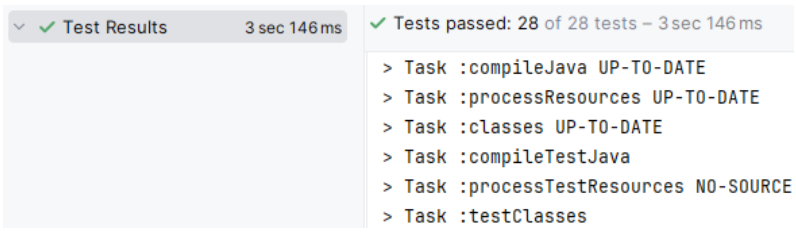

Рисунок 24 – Результат тестирования

При помощи Postman проведено API-тестирование приложения. Этот инструмент позволяет отправлять запросы, анализировать ответы и выполнять тестовые сценарии для проверки интерфейсов приложения. Примеры Postman-запросов приведены на рисунках 25-26.

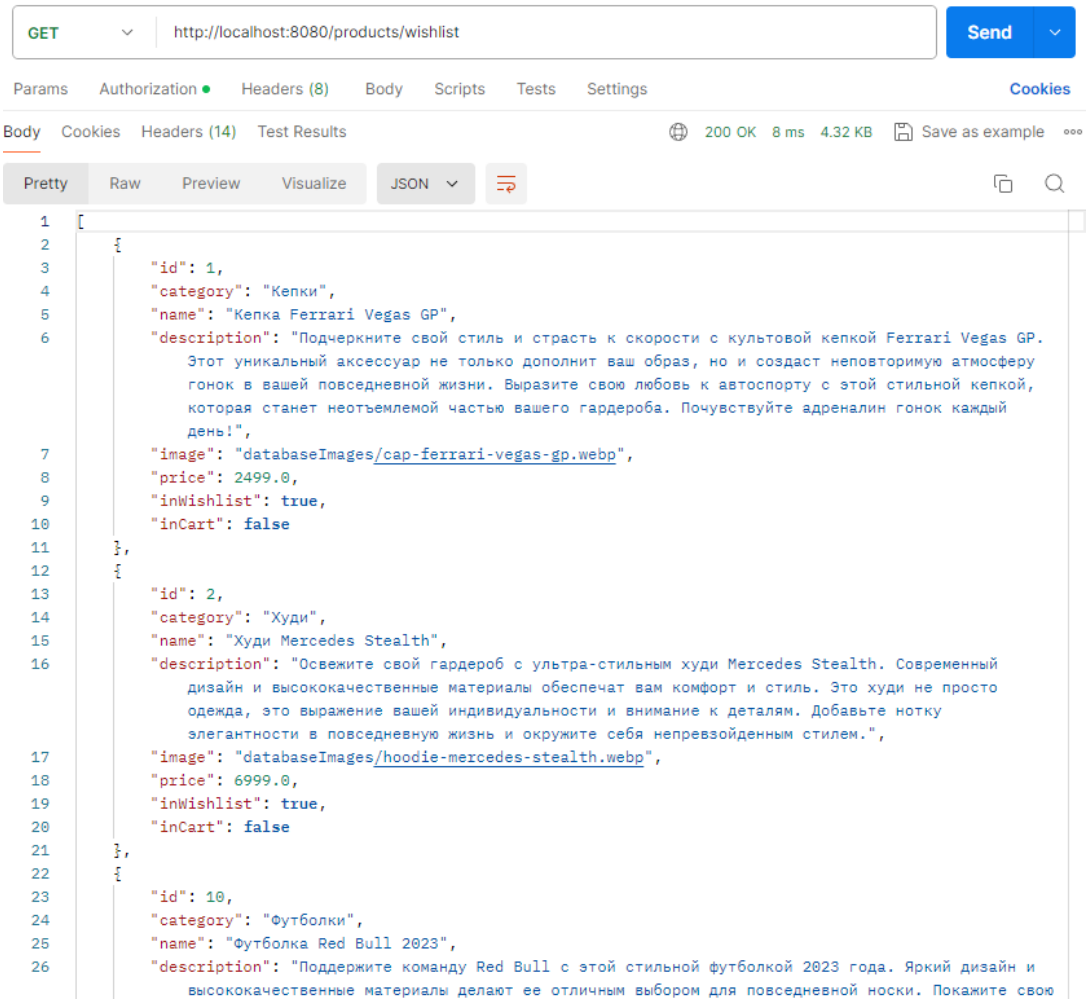

Рисунок 25 – Получение списка избранного при помощи Postman

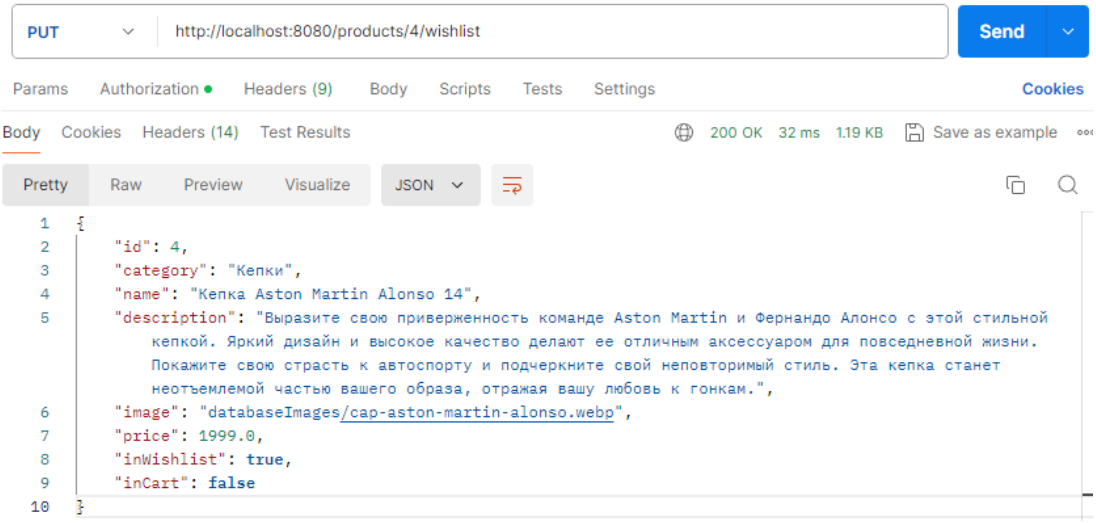

Рисунок 26 – Добавление товара в избранное при помощи Postman

4 КОНТЕЙНЕРИЗАЦИЯ

4.1 Работа с Docker

Docker облегчает разработку и развертывание приложений, обеспечивая изоляцию зависимостей в контейнерах. Создание образов с помощью Dockerfile для фронтенда и бэкенда устанавливает основу для унифицированного и легкого развертывания приложения. Dockerfile для бэкенда на Spring Boot и фронтенда на React.js содержат инструкции для сборки образов с необходимыми зависимостями (рисунки 27-28).

```
FROM amazoncorretto:17
WORKDIR /back
COPY build/libs/f1shop-0.0.1-SNAPSHOT.jar /back/back.jar
CMD ["java", "-jar", "back.jar"]
```

Рисунок 27 – Файл back/Dockerfile

```
FROM node:18-alpine

WORKDIR /front

EXPOSE 3000

COPY ["package.json", "package.json*", "./"]

RUN npm install

COPY . .

CMD ["npm", "start"]
```

Рисунок 28 – Файл front/Dockerfile

4.2 Работа с docker-compose

Docker Compose управляет многоконтейнерными приложениями при помощи docker-compose.yml. Файл YAML конфигурирует сервисы приложения, устанавливая важные параметры, такие как сети, тома и порты. Благодаря этому, поднятие инфраструктуры приложения выполняется одной командой, что делает процесс разработки более эффективным и предсказуемым. Написанный docker-compose.yml представлен на рисунке 29, запуск и работающие контейнеры – на рисунках 30-31.

```yaml
services:
  frontend:
    build: ./front/dockerfile
    ports:
      - "3000:3000"
    depends_on:
      - backend
    networks:
      - your-network

  backend:
    build: ./back/dockerfile
    ports:
      - "8080:8080"
    environment:
      - SPRING_DATASOURCE_URL=jdbc:postgresql://db:5432/f1shop
      - SPRING_DATASOURCE_USERNAME=test
      - SPRING_DATASOURCE_PASSWORD=test
    depends_on:
      - db
    networks:
      - your-network

  db:
    image: postgres:13.1-alpine
    volumes:
      - db-volume:/var/lib/postgresql/data
    environment:
      - POSTGRES_DB=f1shop
      - POSTGRES_USER=test
      - POSTGRES_PASSWORD=test
    networks:
      - your-network

networks:
  your-network:
    driver: bridge

volumes:
  db-volume:
```

Рисунок 29 – Файл docker-compose.yaml

```
PS C:\Users\proko\dev\my\F1Shop> docker compose up -d
[+] Running 3/3
 ✓ Container f1shop-db-1        Started
 ✓ Container f1shop-backend-1   Started
 ✓ Container f1shop-frontend-1  Started
```

Рисунок 30 – Запуск приложения с помощью docker-compose

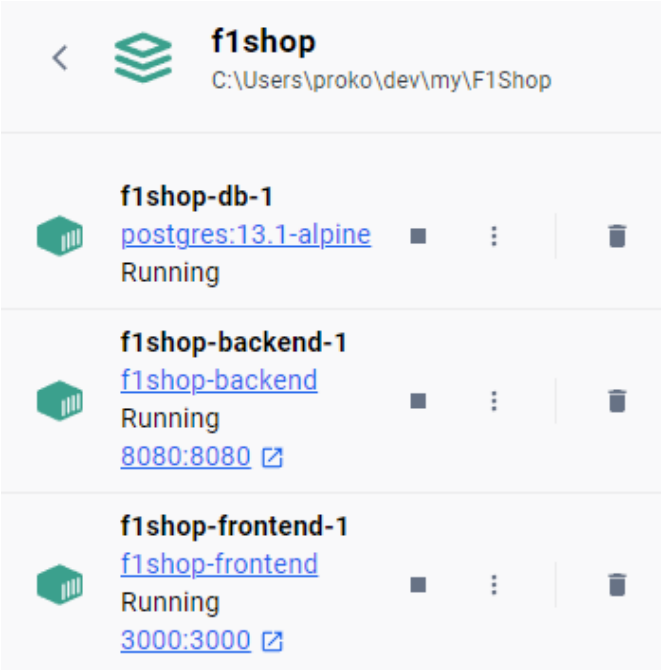

Рисунок 31 – Запущенные контейнеры в Docker Desktop

ЗАКЛЮЧЕНИЕ

В процессе выполнения работы проведен анализ предметной области и разработано приложение «Мир Формулы-1».

Анализ существующих решений в рассматриваемой предметной области позволил выделить ключевые преимущества и недостатки похожих проектов, что послужило основой для определения оптимальных путей развития создаваемого интернет-магазина.

На этапе проектирования разработана структура программной системы, учитывающая основные потребности пользователей и обеспечивающая эффективное взаимодействие с функционалом сайта. Этот этап включал в себя не только архитектурные решения, но и выбор современных технологий и фреймворков для обеспечения масштабируемости и упрощения дальнейшего поддерживания системы.

Особое внимание уделено тестированию приложения для обеспечения его безопасного использования в любых сценариях.

В завершении, с учетом проведенного анализа и успешно реализованных этапов, создана функциональная программная система, спроектированная с учетом современных стандартов и требований. Выполнены все пункты задания, достигнуты все поставленные цели.

Приложение доступно по ссылке: http://188.225.79.252:3000/F1Shop.

Исходный код приложения расположен по ссылке: https://github.com/Remsely/F1Shop.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Fuel For Fans [Электронный ресурс] – URL: https://www.fuelforfans.com/global/en/home/ (дата обращения 11.05.2024).
2. merchf1 – Мерч для фанатов автоспорта [Электронный ресурс] – URL: https://merchf1.ru/ (дата обращения 11.05.2023).
3. F1 2023 Collection – Fueler store [Электронный ресурс] – URL: https://fueler.store/collections/f1-2023-collection (дата обращения 11.05.2024).
4. formulasport [Электронный ресурс] – URL: https://formulasport.pro/shop/catalogue/ (дата обращения 11.05.2024).
5. Database tool window | IntelliJIDEA Documantation [Электронный ресурс] – URL: https://www.jetbrains.com/help/idea/database-tool-window.html#overview (дата обращения 11.05.2024).
6. WebStorm: Features [Электронный ресурс] – URL: https://www.jetbrains.com.cn/en-us/webstorm/features/ (дата обращения 11.05.2024).
7. API Tools | Postman API Platform [Электронный ресурс] – URL: https://www.jetbrains.com.cn/en-us/webstorm/features/ (дата обращения 11.05.2024).
8. Кочер П. С. Микросервисы и контейнеры Docker / пер. с анг. А. Н. Киселева. – М.: ДМК Пресс, 2019. – 240 с.
9. Spring Boot : Getting Started [Электронный ресурс] – URL: https://docs.spring.io/spring-boot/docs/current/reference/html/getting-started.html (дата обращения 11.05.2024).
10. JPA Query Methods :: Spring Data JPA [Электронный ресурс] – URL: https://docs.spring.io/spring-data/jpa/reference/jpa/query-methods.html (дата обращения 11.05.2024).
11. Java Bean Validation :: Spring Framework [Электронный ресурс] – URL: https://docs.spring.io/spring-framework/reference/core/validation/beanvalidation.html (дата обращения 11.05.2024).

12. JSON Web Token Introduction [Электронный ресурс] – URL: https://jwt.io/introduction (дата обращения 11.05.2024).

13. Config Defaults | Axios docs [Электронный ресурс] – URL: https://axios-http.com/docs/config_defaults (дата обращения 11.05.2024).

14. The DTO Pattern [Электронный ресурс] – URL: https://www.baeldung.com/java-dto-pattern (дата обращения 11.05.2024).