



МИНОБРАЗОВАНИЯ РОССИИ

Федеральное государственное бюджетное образовательное учреждение
высшего образования

«МИРЭА – Российский технологический университет»

РТУ МИРЭА

Институт информационных технологий (ИТ)

Кафедра инструментального и прикладного программного обеспечения (ИиПО)

КУРСОВАЯ РАБОТА

по дисциплине: Разработка клиент-серверных приложений

по профилю: Разработка программных продуктов и проектирование информационных систем

направления профессиональной подготовки: 09.03.04 «Программная инженерия»

Тема: «Разработка клиент-серверного фуллстек-приложения для создания виртуальной книжной полки пользователей»

Студент: Мухаметшин Александр Ринатович

Группа: ИКБО-20-21

Работа представлена к защите 5.6.24 (дата) [подпись] / Мухаметшин А. Р. /
(подпись и ф.и.о. студента)

Руководитель: ст. п. Коваленко Михаил Андреевич

Работа допущена к защите 20.05.24 (дата) [подпись] / Коваленко М. А. /
(подпись и ф.и.о. рук-ля)

Оценка по итогам защиты: отл.

[подпись] 05.06.24 / Алпатов А.Н., доцент, доцент, к.т.н. /
[подпись] 05.06.24 / Коваленко М.А., ст.преп. /

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

М. РТУ МИРЭА. 2024 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: Разработка клиент-серверных приложений

по профилю: Разработка программных продуктов и проектирование информационных систем

направления профессиональной подготовки: Программная инженерия (09.03.04)

Студент: Мухаметшин Александр Ринатович

Группа: ИКБО-20-21

Срок представления к защите: 20.05.2024

Руководитель: ст.п., Коваленко Михаил Андреевич

Тема: «Разработка клиент-серверного фуллстек-приложения для создания виртуальной книжной полки пользователей»

Исходные данные: React, Redux, Go, PostgreSQL, Git.

Перечень вопросов, подлежащих разработке, и обязательного графического материала: 1. Провести анализ предметной области для выбранной темы, обосновать выбор клиент-серверной архитектуры и дать её детальное описание с помощью UML для разрабатываемого приложения; 2. Выбрать программный стек для реализации фуллстек разработки, ориентируясь на мировой опыт и стандарты в данной области; 3. Реализовать фронтенд и бекенд части клиент-серверного приложения, обеспечивать авторизацию и аутентификацию пользователя; 4. Разместить готовое клиент-серверное приложение в репозитории GitHub с указанием ссылки в тексте отчёта; 5. Развернуть клиент-серверное приложение в облаке. 6. Провести пользовательское тестирование функционирования минимально жизнеспособного продукта. 7. Разработать презентацию с графическими материалами.

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

Зав. кафедрой ИиППО: [подпись] / Болбаков Р.Г. /, « 26 » 02 2024 г.

Задание на КР выдал: [подпись] / Коваленко М.А. /, « 26 » 02 2024 г.

Задание на КР получил: [подпись] / Мухаметшин А. Р. /, « 26 » 02 2024 г.

АННОТАЦИЯ

Руководитель курсовой работы: ст. п. Коваленко Михаил Андреевич.

Мухаметшин А. Р., Курсовая работа направления подготовки «Программная инженерия» на тему «**Разработка клиент-серверного фуллстек-приложения для создания виртуальной книжной полки пользователей**»: М. 2024 г., МИРЭА – Российский технологический университет (РТУ МИРЭА), Институт информационных технологий (ИИТ), кафедра инструментального и прикладного программного обеспечения (ИиППО) – 42 стр., 20 рис., 1 табл., 20 источн.

Ключевые слова: GOLANG, REACT, приложение, развертывание, API, REST

Целью работы является разработка клиент-серверного фуллстек-приложения для создания виртуальной книжной полки пользователей. В ходе работы был проведён анализ предметной области и обзор сайтов с аналогичной тематикой. Рассмотрены процесс разработки архитектуры и её детальное описание с помощью UML, выбор программного стека для реализации фуллстек разработки, реализация фронтенд и бекенд частей приложения с обеспечением авторизации и аутентификации пользователей. Готовое клиент-серверное приложение размещено в репозитории GitHub и развернуто в облаке. Результатом является приложение, соответствующее стандартам RESTful и позволяющее пользователям добавлять книги, собирать их в коллекции, делиться этими коллекциями, а также оставлять комментарии к книгам.

Mukhametshin A. R., Coursework of the direction of training "Software Engineering" on the topic "Development of client-server fullstack application for creating a virtual bookshelf of users": M. 2024, MIREA - Russian Technological University (RTU MIREA), Institute of Information Technologies (IIT), Department of Instrumental and Applied Software (I&APS) - 42 pp, 20 figs, 1 tab, 20 sources.

Keywords: GOLANG, REACT, application, deployment, API, REST

The purpose of the work is to develop a client-server fullstack application to create a virtual bookshelf for users. The work involved analyzing the subject area and reviewing websites with similar topics. The process of architecture development and its detailed description with the help of UML, selection of software stack for implementation of fullstack development, implementation of frontend and backend parts of the application with provision of authorization and authentication of users are considered. The finished client-server application is hosted in a GitHub repository and deployed in the cloud. The result is a RESTful standards-compliant application that allows users to add books, collect them into collections, share those collections, and leave comments on the books.

РТУ МИРЭА: 119454, Москва, пр-т Вернадского, д. 78

кафедра инструментального и прикладного программного обеспечения (ИиППО)

Тираж: 1 экз. (на правах рукописи)

Файл: «ПЗ_РКСП_ИКБО-20-21_МухаметшинАР.pdf», исполнитель Мухаметшин А. Р

© Мухаметшин А. Р

СОДЕРЖАНИЕ

ГЛОССАРИЙ	7
1 ОБЩИЕ СВЕДЕНИЯ	9
1.1 Обозначение и наименование интернет-ресурса	9
1.2 Функциональное назначение	9
Вывод к главе 1	9
2 АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ	10
Вывод к главе 2	12
3 ВЫБОР И ОБОСНОВАНИЕ ТЕХНОЛОГИЙ	13
3.1 Выбор архитектуры	13
3.2 Выбор основных технологий	14
Вывод к главе 3	15
4 РАЗРАБОТКА БАЗЫ ДАННЫХ.....	17
4.1 Проектирование базы данных.....	17
4.2 Разработка скрипта базы данных.....	18
Вывод к главе 4	19
5 РАЗРАБОТКА СЕРВЕРНОЙ ЧАСТИ ИНТЕРНЕТ-РЕСУРСА.....	20
5.1 Создание слоя моделей	20
5.2 Создание слоя репозитория	20
5.3 Создание слоя сервисов	21
5.4 Создание слоя контроллеров	22
5.5 Создание конфигурации JWT и аутентификация.	23
Вывод к главе 5	25
6 РАЗРАБОТКА КЛИЕНТСКОЙ ЧАСТИ ИНТЕРНЕТ-РЕСУРСА	26
6.1 Создание клиентской части	26
6.2 Настройка Redux и Axios	26
6.3 Страницы приложения	27
Вывод к главе 6	30
7 ПРОВЕДЕНИЕ ТЕСТИРОВАНИЯ РАЗРАБОТАННОГО ПРИЛОЖЕНИЯ	31
Вывод к главе 7	35

8 РАЗВЕРТЫВАНИЕ ИНТЕРНЕТ-РЕСУРСА	36
8.1 Выбор платформы для развертывания	36
8.2 Процесс развертывания.....	36
8.3 Развертывание сервисной и клиентской частей	36
Вывод к главе 8	38
ЗАКЛЮЧЕНИЕ.....	40

ПЕРЕЧЕНЬ СОКРАЩЕНИЙ И ОБОЗНАЧЕНИЙ

RESTful API	—	Representational State Transfer API
СУБД	—	Система Управления Базами Данных
JWT	—	JSON Web Token
SQL	—	Structured Query Language
URL	—	Uniform Resource Locator

ГЛОССАРИЙ

1. **Серверная часть** — часть приложения, которая занимается обработкой сложных данных, взаимодействует с базой данных и может лишь передавать клиенту данные в специальном установленном формате.

2. **Клиентская часть** - часть приложения, отвечающая за представление данных, полученных от сервера, пользователю. Также клиентская часть, иначе — «визуальная часть», позволяет взаимодействовать с пользователем, принимая от него какую-то информацию и передавая её на сервер.

3. **Архитектура приложения** - правила разработки приложения, которые часто позволяют упростить разработку, поддержку и расширение приложения.

4. **База данных** - хранилище с данными, которые разработчик предпочел хранить отдельно от клиентской части (пользователи, сотрудники и т.п.). Такое хранилище управляется посредством серверной части и СУБД.

5. **СУБД** - система, которая позволяет взаимодействовать с базой данных: добавлять, изменять, получать и удалять данные.

6. **Фуллстек-приложение** — это программное приложение, которое разрабатывается с использованием одной технологической стека для всех его компонентов, как на стороне клиента (front-end), так и на стороне сервера (back-end).

ВВЕДЕНИЕ

В современном обществе информационные технологии тесно переплетены с сферами государства, общественной жизни и бизнеса, играя важнейшую роль в организации различных процессов. Эффективное функционирование в наши дни становится невозможным без применения передовых технологий, которые значительно упрощают, систематизируют и оптимизируют рабочие процессы. Разработка веб-приложений в данном контексте представляет собой сложную и многогранную задачу, требующую от разработчика глубокого понимания архитектуры, технологий и инструментов.

Цель настоящего исследования заключается в создании клиент-серверного фулстек-приложения для создания виртуальной книжной полки пользователей и развертывании его в облаке, применяя современные технологии на базе языка Golang, библиотеки React и сервиса для облачного развертывания Render.

Для выполнения поставленной цели курсовой работы необходимо выполнить следующие пункты:

- провести анализ предметной области разрабатываемого веб-приложения;
- обосновать выбор клиент-серверной архитектуры и дать её детальное описание с помощью UML;
- выбрать программный стек для реализации фулстек разработки, ориентируясь на мировой опыт и стандарты в данной области;
- реализовать фронтенд и бекенд части клиент-серверного приложения, обеспечивая авторизацию и аутентификацию пользователя;
- разместить готовое клиент-серверное приложение в репозитории GitHub с указанием ссылки в тексте отчёта;
- развернуть клиент-серверное приложение в облаке;
- провести пользовательское тестирование функционирования минимально жизнеспособного продукта.

1 ОБЩИЕ СВЕДЕНИЯ

1.1 Обозначение и наименование интернет-ресурса

Темой разработанного веб-приложения является «Виртуальная книжная полка». Разработанное приложение получило название – «Bookshelf».

1.2 Функциональное назначение

Функциональное назначение веб-приложения заключается в предоставлении пользователям возможности создавать и управлять своей виртуальной книжной полкой. Приложение обеспечивает возможность поиска и добавления книг из различных источников, создания коллекций книг, делиться этими коллекциями с другими пользователями, а также оставлять комментарии к книгам. Пользователи могут систематизировать свои книжные коллекции, получать рекомендации и взаимодействовать с сообществом любителей книг.

Вывод к главе 1

В результате рассмотрения общей информации о разработанном веб-приложении «Bookshelf» были определены основные функциональные возможности и цели приложения. Приложение предоставляет пользователям инструменты для создания и управления виртуальными книжными полками, что позволяет улучшить организацию личной библиотеки и взаимодействие с другими пользователями. Эти функциональные возможности делают «Bookshelf» удобным и полезным инструментом для любителей книг.

2 АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

Для анализа предметной области было проведено исследование существующих аналогичных приложений и интернет-ресурсов, выявлены их преимущества и недостатки, которые представлены в таблице 2.1. Для сравнения были выбраны интернет-ресурсы Goodreads (Рисунок 2.1), LibraryThing (Рисунок 2.2) и BookLikes (Рисунок 2.3).

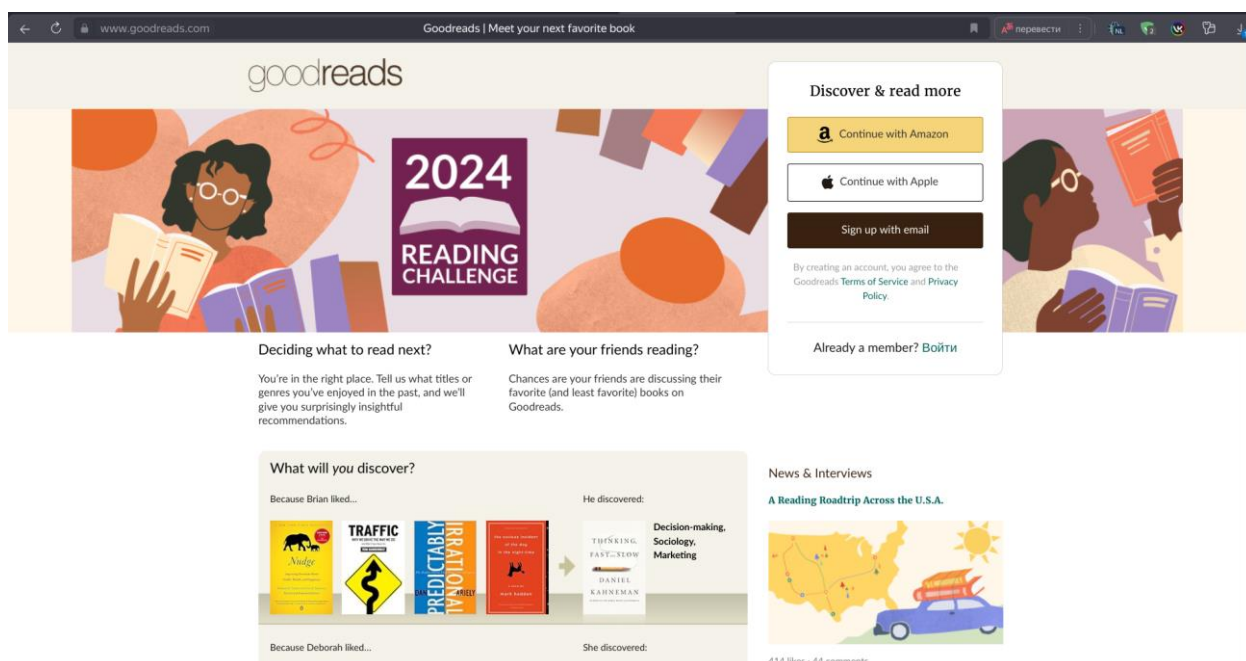


Рисунок 2.1 - Интернет-ресурс Goodreads

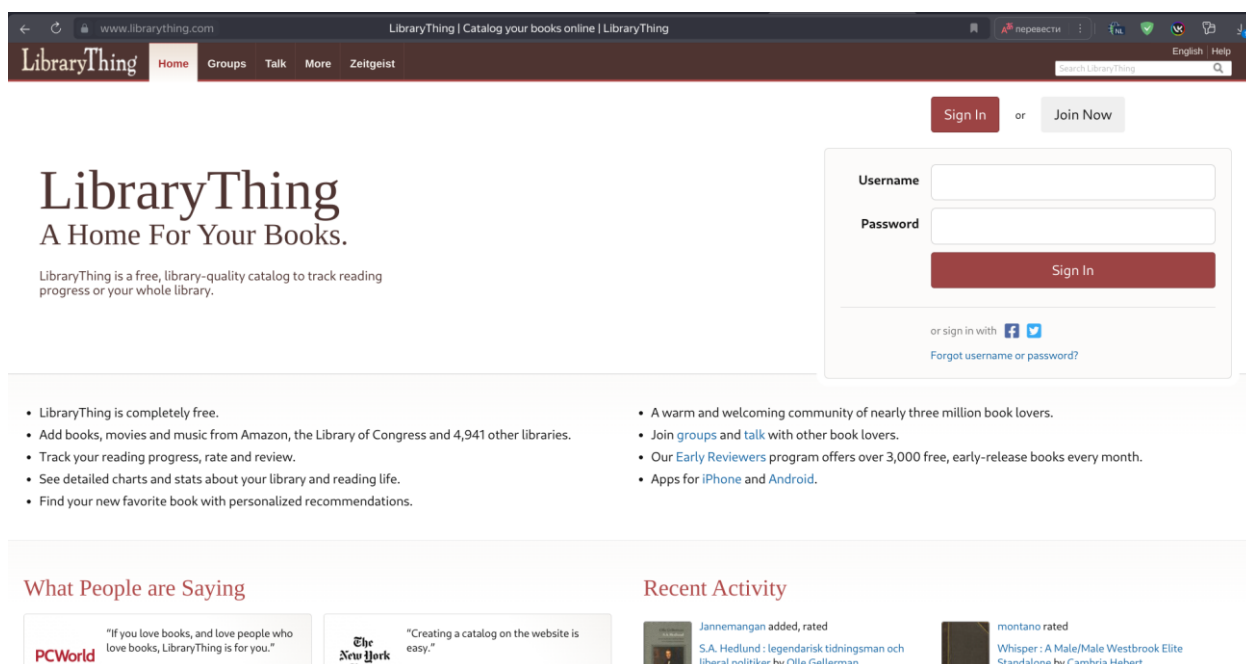


Рисунок 2.2 - Интернет-ресурс LibraryThing

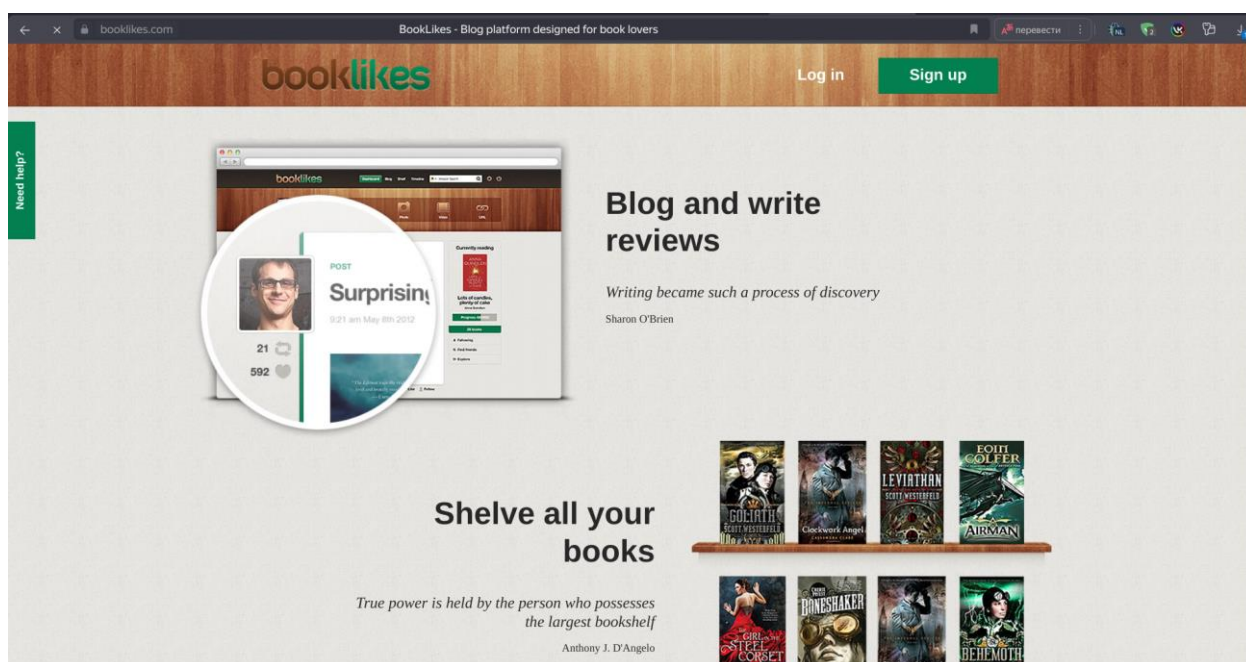


Рисунок 2.3 - Интернет-ресурс BookLikes

Таблица 2.1 – Сравнение аналогичных приложений

Название веб-сервиса	Goodreads	LibraryThing	BookLikes
Возможность добавления книг в коллекции	+	+	+
Возможность создания аккаунта для сохранения списка книг и их статуса	+	+	+
Наличие фильтров для поиска книг по категориям, авторам, рейтингу и т.д.	+	-	-
Возможность оценки и комментирования различных книг	+	+	+
Возможность создания и управления личными коллекциями	-	-	+
Наличие рекомендаций на основе прочитанных книг	+	+	+

Таким образом, разрабатываемое приложение должно быть функционально насыщенным, иметь интуитивно понятный интерфейс и

предоставлять высококачественные услуги пользователям.

При разработке «Bookshelf» были учтены достоинства и недостатки существующих аналогичных приложений.

Вывод к главе 2

В процессе анализа предметной области были определены преимущества и недостатки различных веб-сервисов для управления книжными коллекциями. На основании этого были сформулированы требования к будущему приложению «Bookshelf»:

- возможность поиска и добавления книг в коллекции,
- создание личного кабинета для пользователей,
- управление книжными коллекциями,
- возможность оставления комментариев и оценок к книгам,
- возможность добавления книг со сторонних ресурсов.

3 ВЫБОР И ОБОСНОВАНИЕ ТЕХНОЛОГИЙ

3.1 Выбор архитектуры

При создании приложения был выбран архитектурный подход Model-View-Controller (MVC), который обеспечивает четкое разделение между представлением, логикой и данными (рисунок 3.1.1). Такой подход помогает упростить разработку, тестирование и поддержку кода.

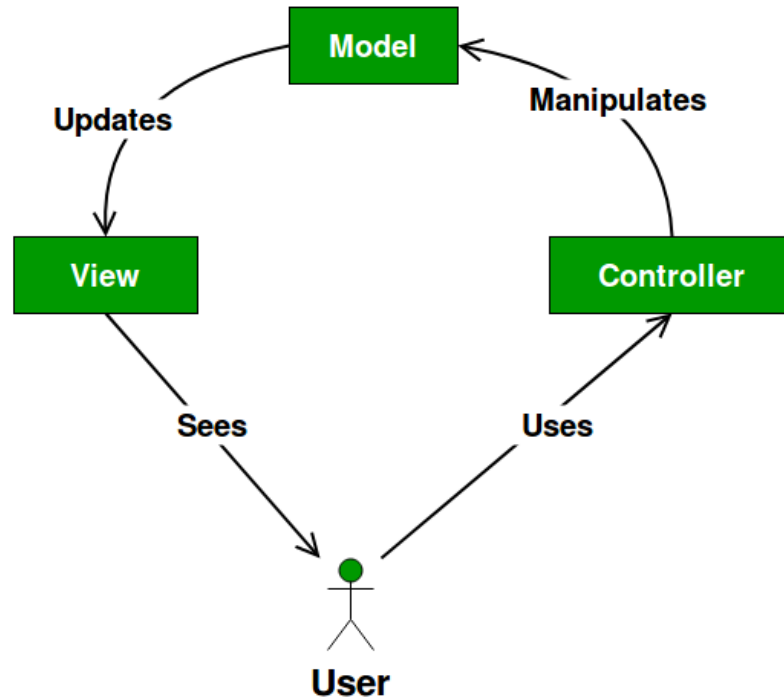


Рисунок 3.1.1 – Архитектура MVC

В приложении «Bookshelf» MVC реализован следующим образом:

Модели: Определяют структуру данных и их связи в базе данных PostgreSQL. Модели служат для представления данных и их атрибутов, а также для взаимодействия с базой данных.

Виды: Представляют собой данные, которые сервер возвращает клиенту в ответ на запросы. В данном приложении виды представлены в формате JSON, что позволяет легко интегрироваться с фронтендом, построенным на React.

Контроллеры: Обрабатывают входящие запросы, взаимодействуют с моделями и определяют, какие виды должны быть возвращены в ответ на запросы. Контроллеры выступают связующим звеном между моделями и

видами.

Контроллеры в приложении «Bookshelf» подразделяются на три основных компонента:

REST-контроллеры: Обрабатывают HTTP-запросы от клиента, определяют параметры запросов и управляют взаимодействием между клиентом и сервером.

Сервисы: Реализуют бизнес-логику приложения. Они обрабатывают данные, полученные от REST-контроллеров, взаимодействуют с репозиториями и вызывают другие вспомогательные сервисы для выполнения необходимых операций.

Репозитории: Отвечают за взаимодействие с базой данных PostgreSQL. Они выполняют SQL-запросы для управления данными и предоставляют сервисам доступ к данным.

Этот архитектурный подход обеспечивает модульность и гибкость, позволяя легко масштабировать и поддерживать приложение.

3.2 Выбор основных технологий

В рамках данного исследования были определены ключевые технологии, которые обеспечат эффективную разработку веб-приложения для создания виртуальной книжной полки. Использование этих технологий позволит реализовать все необходимые требования и создать надежное и функциональное приложение:

- **GoLand:** Интегрированная среда разработки, которая предоставляет множество инструментов и расширений для комфортной разработки приложений на языке Go.

- **Golang:** Основной язык программирования для серверной части приложения. Golang выбран благодаря своей высокой производительности и поддержке параллельного выполнения задач.

- **React:** JavaScript-библиотека для разработки пользовательского интерфейса. React позволяет создавать динамичные и отзывчивые веб-приложения с высокой интерактивностью.

- PostgreSQL: Реляционная база данных, выбранная для хранения данных. PostgreSQL обеспечивает надежность и гибкость при работе с данными, что важно для масштабируемых приложений.
- Gorilla Mux: Библиотека маршрутизации для Golang, которая упрощает создание RESTful API и поддерживает различные HTTP-методы и маршруты.
- Redux: Библиотека для управления состоянием приложения на клиентской стороне, особенно полезная для крупных приложений, разработанных с использованием React.
- GitHub: Платформа для хостинга кода и система контроля версий, что позволяет эффективно управлять разработкой приложений.
- JSON: Формат для обмена данными между серверной и клиентской частями приложения, обеспечивающий удобочитаемость и легкость передачи данных.
- Insomnia: Многофункциональная платформа для тестирования и разработки API. Insomnia предоставляет интуитивно понятный интерфейс для создания, отправки и отладки запросов к API, а также для автоматизации тестирования.

Эти технологии широко используются в индустрии и совместимы друг с другом, что позволит создать функциональное и надежное веб-приложение для управления виртуальными книжными полками. Их применение обеспечит эффективную реализацию всех требований, поставленных в данной курсовой работе.

Вывод к главе 3

В данной главе были рассмотрен архитектурный подход MVC, который был выбран для разработки веб-сервиса «BookShelf».

Кроме того, был сделан выбор основных технологий для реализации проекта. Эти технологии включают в себя интегрированную среду разработки GoLand, язык программирования Golang, библиотеку React, базу данных PostgreSQL, библиотеку маршрутизации Gorilla Mux, Redux для управления

состоянием приложения, GitHub для хостинга и контроля версий, формат JSON для обмена данными и Insomnia для тестирования API. Эти инструменты были выбраны из-за их широкого использования в индустрии, совместимости и способности обеспечить создание гибкого и эффективного веб-приложения.

Таким образом, выбор архитектуры в сочетании с указанными технологиями закладывает основу для разработки функционального, масштабируемого и легко поддерживаемого веб-приложения для создания виртуальной книжной полки. В следующих главах будут рассмотрены более детальные аспекты реализации, включая модели данных, серверную и клиентскую логику, а также взаимодействие с базой данных.

4 РАЗРАБОТКА БАЗЫ ДАННЫХ

4.1 Проектирование базы данных

При проектировании базы данных для приложения "Bookshelf" была проведена детальная работа над моделью данных. Эта модель включает несколько ключевых сущностей, отражающих основные функциональные аспекты приложения.

- Пользователь (Users): Представляет пользователей приложения. Включает такие атрибуты, как уникальный идентификатор, имя пользователя, роль пользователя и хэш пароля.

- Коллекция (Collections): Описывает коллекции книг, созданные пользователями. Включает идентификатор коллекции, идентификатор пользователя, который создал коллекцию, название коллекции, её описание, рейтинг и статус публичности.

- Книга (Books): Представляет книги, добавленные пользователями. Содержит идентификатор книги, идентификатор пользователя, который добавил книгу, название книги, автора, жанр, изображение, описание и содержание.

- Комментарий (Comments): Описывает комментарии к книгам. Включает идентификатор комментария, идентификатор пользователя, оставившего комментарий, идентификатор книги, к которой оставлен комментарий, рейтинг и текст комментария.

- Связь книги с коллекцией (Collections_Books): Промежуточная таблица для реализации связи многие ко многим между книгами и коллекциями. Включает идентификатор коллекции и идентификатор книги.

- Рейтинг коллекции (Collection_Ratings): Описывает рейтинги, присвоенные коллекциям. Включает идентификатор рейтинга, идентификатор коллекции и значение рейтинга.

Эти сущности были тщательно спроектированы и связаны между собой для обеспечения эффективного хранения информации, поддержки функционала приложения и обеспечения целостности данных. Итоги

проектирования базы данных представлены в ER-диаграмме на рисунке 4.1.1.

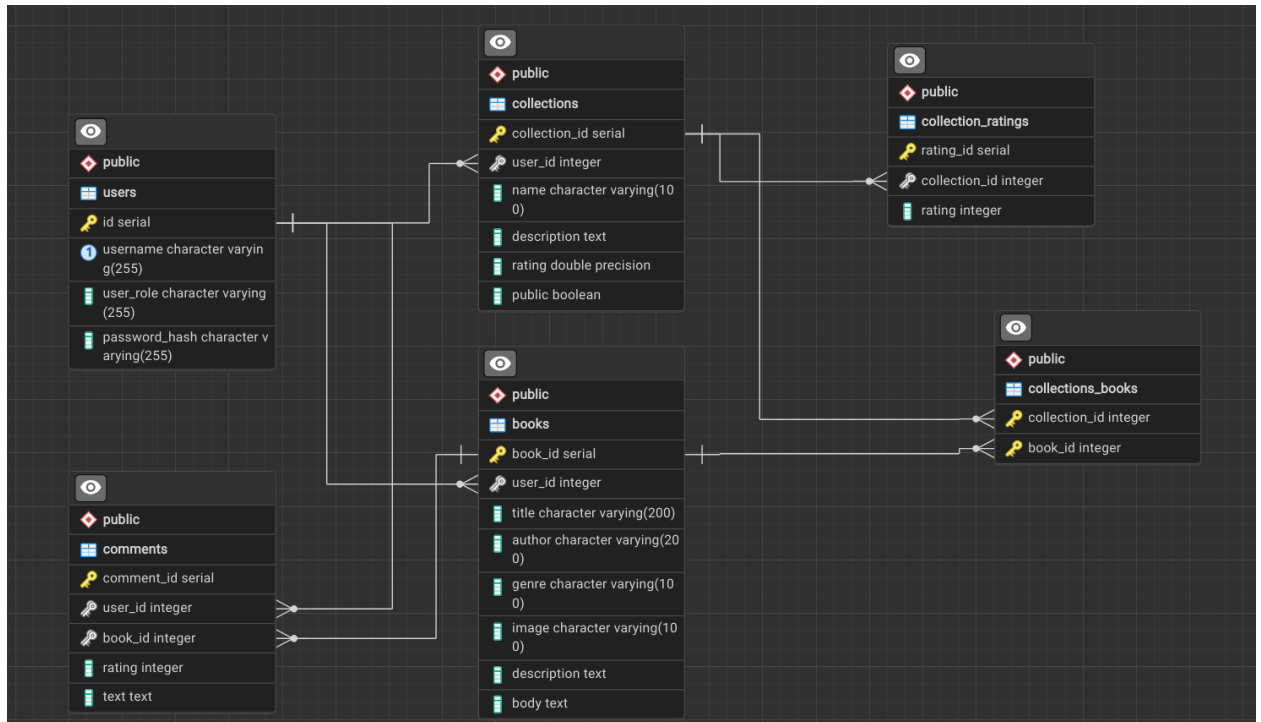


Рисунок 4.1.1 - ER-диаграмма базы данных

4.2 Разработка скрипта базы данных

При разработке скрипта базы данных использовались SQL запросы для создания таблиц и их связей для автоматического обновления связанных данных.

Были созданы разработанные таблицы. Для примера, рассмотрим создание таблицы comment, которая содержит информацию о комментариях (листинг 4.2.1).

Листинг 4.2.1 – пример создания таблицы

```

CREATE TABLE IF NOT EXISTS comments (
    comment_id SERIAL PRIMARY KEY,
    user_id INT REFERENCES users(id),
    book_id INT REFERENCES books(book_id),
    rating INT CHECK (rating >= 0 AND rating <= 5),
    text TEXT
);
    
```

Для реализации связей многие-с-многим использовались промежуточные таблицы. Например, таблица collection_book связывает

таблицы collection и book, позволяя отслеживать книги, добавленные в коллекции. Пример скрипта представлен на листинге 4.2.2.

Листинг 4.2.2 – пример реализации связи многие-к-многим

```
CREATE TABLE IF NOT EXISTS collections_books (  
    collection_id INT REFERENCES collections(collection_id),  
    book_id INT REFERENCES books(book_id),  
    PRIMARY KEY (collection_id, book_id)  
);
```

Вывод к главе 4

При проектировании базы данных для приложения "Bookshelf" была создана детальная модель, включающая ключевые сущности: пользователи, коллекции, книги, комментарии и рейтинги коллекций. Основное внимание было уделено разработке связей между этими сущностями для обеспечения целостности данных и поддержки всех функциональных возможностей приложения. Схема данных и связи между сущностями наглядно отображены в ER-диаграмме.

Для создания таблиц и установления связей между ними были использованы SQL-запросы, что позволило эффективно организовать хранение и управление данными. Также были предусмотрены механизмы для поддержания целостности данных, такие как ограничения и ссылки между таблицами.

Проектирование базы данных обеспечило создание надежной основы для реализации всех функций приложения, включая создание и управление коллекциями книг, добавление и просмотр книг, оставление комментариев и оценок. Это позволило сформировать функциональную и гибкую структуру данных, отвечающую требованиям курсовой работы.

5 РАЗРАБОТКА СЕРВЕРНОЙ ЧАСТИ ИНТЕРНЕТ-РЕСУРСА

5.1 Создание слоя моделей

В процессе разработки интернет-проекта были созданы модели данных, которые описывают основные сущности предметной области проекта. Каждая из этих моделей имеет свои поля, которые описывают ее поведение в системе. В приложении представлены следующие модели: Book, User, Collection и Comment.

Для создания моделей данных в Golang использованы соответствующие структуры и теги для работы с базой данных и удобным преобразованием в JSON, такие как “json”.

Пример реализации модели книги с использованием указанных тегов можно ознакомиться на листинге 5.1.1.

Листинг 5.1.1 – создание модели книги

```
package models

type Book struct {
    BookId      int    `json:"book_id"`
    UserId      int    `json:"user_id"`
    Title       string `json:"title"`
    validate:"required,min=1,max=200"
    Author      string `json:"author"`
    validate:"required,min=1,max=200"
    Genre       string `json:"genre,omitempty"`
    Description  string `json:"description,omitempty"`
    Image       string `json:"image,omitempty"`
    Body        string `json:"body,omitempty"`
}
```

5.2 Создание слоя репозитория

В разработанном приложении этот слой реализован с помощью библиотеки PostgreSQL. Репозитории обеспечивают доступ к базам данных и другим источникам данных. Они реализуют CRUD-операции (Create, Read, Update, Delete) и обеспечивают хранение и извлечение данных.

Репозитории позволяют абстрагироваться от конкретных технологий хранения данных и скрыть сложность работы с базой данных за простым

интерфейсом. Это позволяет существенно упростить разработку приложений и снизить затраты на поддержку и обновление кода.

Так были созданы следующие репозитории: `BookRepository`, `UserRepository`, `CollectionRepository` и `CommentRepository`.

На листинге 5.2.1 представлена реализация репозитория для книг.

Листинг 5.2.1 – часть функций репозитория книги

```
package repository

type BookPostgres struct {
    db *sql.DB
}

func NewBookPostgres(db *sql.DB) *BookPostgres {
    return &BookPostgres{db: db}
}

func (r *BookPostgres) Create(book models.Book) (int, error) {
    var id int
    query := fmt.Sprintf("INSERT INTO %s (user_id, title,
author, genre, description, image, body) VALUES ($1, $2, $3, $4, $5,
$6, $7) RETURNING book_id", db.BOOKS)
    row := r.db.QueryRow(query, book.UserId, book.Title,
book.Author, book.Genre, book.Description, book.Image, book.Body)
    if err := row.Scan(&id); err != nil {
        log.Panic(err)
        return 0, err
    }
    return id, nil
}
```

5.3 Создание слоя сервисов

В разработанном приложении сервисы обеспечивают бизнес-логику и взаимодействие между слоями контроллеров, моделей и репозиториев. В каждом сервисе реализуются методы, которые выполняют конкретные операции с данными и содержат логику, специфичную для определенных сущностей.

Для каждой сущности (`Book`, `User`, `Collection`, `Comment`) был создан соответствующий сервис: `BookService`, `UserService`, `CollectionService` и

CommentService. Эти сервисы инкапсулируют основную бизнес-логику и обеспечивают её выполнение в ответ на запросы от контроллеров.

На листинге 5.3.1 представлена реализация сервиса для книг.

Листинг 5.3.1 – часть функций сервиса для книг

```
package service
type BookService struct {
    repo Book
}
func NewBookService(repo Book) *BookService {
    return &BookService{repo: repo}
}
func (s *BookService) GetAll() ([]DTO.BookDTO, error) {
    return s.repo.FindAll()
}
```

5.4 Создание слоя контроллеров

Следующим шагом является создание контроллеров, которые отвечают за обработку запросов и взаимодействие с клиентской частью приложения. Создаются контроллеры, которые определяют конечные точки (эндпоинты) REST API и обрабатывают входящие запросы.

Для каждой сущности в проекте создаются соответствующие контроллеры, которые обеспечивают CRUD-операции (создание, чтение, обновление и удаление) и другие действия с данными. Контроллеры используют репозитории для обработки запросов и доступа к данным.

Для назначения эндпоинтов используется функции библиотеки Mux. HandleFunc назначает в качестве обработчика определенного эндпоинта соответствующую функцию, также для корректной работы требуется указать тип запроса (POST, GET, OPTIONS и т.д.). Назначением эндпоинтов занимается функция SetupRoutes, в которой также происходит назначение настроек CORS. В приложении А на листинге 3 представлена реализация функции SetupRoutes.

С реализацией контроллера на примере контроллера книг можно ознакомиться на листинге 5.4.1.

Листинг 5.4.1 – часть контроллера книги

```
package handler

func (h *Handler) GetAllBooks(w http.ResponseWriter, r
*http.Request) {
    w.Header().Set("Content-Type", "application/json")

    books, err := h.service.Book.GetAll()
    if err != nil {
        newErrorResponse(w, http.StatusInternalServerError,
err.Error())
        return
    }

    booksByte, err := json.Marshal(books)
    if err != nil {
        newErrorResponse(w, http.StatusInternalServerError,
err.Error())
        return
    }

    log.Println("GetAllBooks is ok")
    w.WriteHeader(http.StatusOK)
    _, _ = w.Write(booksByte)
}

books := api.PathPrefix("/books").Subrouter()
books.HandleFunc("", h.GetAllBooks).Methods("GET", "OPTIONS")
```

5.5 Создание конфигурации JWT и аутентификация.

Для реализации аутентификации и авторизации пользователей в приложении на Go используются следующие функции и методы, обеспечивающие создание пользователей, генерацию и парсинг JWT-токенов, а также хэширование паролей. С реализацией данной функциональности можно ознакомиться в на листинге 5.5.1.

Непосредственно бизнес-логика данного компонента происходит в функции Authorize (Приложение А Листинг 6), которая реализует проверку токена на валидность для авторизованных пользователей и извлечение

нужных данных. В случае же регистрации происходит генерация токена и создание нового пользователя.

Листинг 5.5.1 – реализация JWT авторизации

```
func (s *AuthService) GenerateToken(username, password string)
(string, int, error) {
    user, err := s.repo.GetOne(username,
generatePasswordHash(password))
    if err != nil {
        return "", 0, err
    }
    token := jwt.NewWithClaims(jwt.SigningMethodHS256, &tokenClaims{
        jwt.StandardClaims{
            IssuedAt:  time.Now().Unix(),
            ExpiresAt: time.Now().Add(tokenTTL).Unix(),
        },
        user.Id,
    })
    str, err := token.SignedString([]byte(signingKey))
    return str, user.Id, err
}

func (s *AuthService) ParseToken(accessToken string) (int, error) {
    token, err := jwt.ParseWithClaims(accessToken,
&tokenClaims{}, func(token *jwt.Token) (interface{}, error) {
        if _, ok := token.Method.(*jwt.SigningMethodHMAC);
!ok {
            return nil, errors.New("invalid signing method")
        }
        return []byte(signingKey), nil
    })
    if err != nil {
        return 0, err
    }
    claims, ok := token.Claims.(*tokenClaims)
    if !ok {
return 0, errors.New("token claims aare not of type *tokenClaims")
    }
    return claims.UserId, nil
}
```

Метод `CreateUser` принимает на вход объект пользователя, хэширует его пароль, проверяет наличие пользователя с таким же именем в базе данных и создает нового пользователя, если имя свободно.

Метод `GetUser` возвращает пользователя по его ID.

Метод `GenerateToken` принимает имя пользователя и пароль, находит пользователя в базе данных, и, если учетные данные верны, генерирует JWT-токен с помощью `jwt.StandardClaims` и возвращает его.

Метод `ParseToken` парсит JWT-токен, извлекает информацию о пользователе и возвращает его ID.

Метод `HashPassword` хэширует пароли.

Вся бизнес-логика для работы с пользователями и токенами реализована в методах `CreateUser`, `GetUser`, `GenerateToken`, `ParseToken` и `HashPassword`. Эти методы обеспечивают регистрацию новых пользователей, проверку учетных данных, генерацию и валидацию JWT-токенов, а также безопасное хранение паролей.

Вывод к главе 5

В данной главе были рассмотрены ключевые аспекты разработки веб-приложения для управления виртуальной книжной полкой. В ходе работы были применены современные подходы и технологии, такие как микросервисная архитектура, REST API, Golang и соответствующие библиотеки.

Модели данных были разработаны с использованием тегов, обеспечивающих их интеграцию с базой данных. Слой репозитория обеспечил удобство взаимодействия с данными. Контроллеры были созданы для обработки запросов и управления взаимодействием с клиентской частью. Настройка JWT обеспечила безопасность приложения через аутентификацию и авторизацию. Сервисы инкапсулировали бизнес-логику и обеспечили её выполнение.

6 РАЗРАБОТКА КЛИЕНТСКОЙ ЧАСТИ ИНТЕРНЕТ-РЕСУРСА

6.1 Создание клиентской части

Для создания клиентской части и обеспечения лучшего пользовательского опыта был выбран React фреймворк, который позволяет разрабатывать динамические и отзывчивые веб-приложения. В результате получился клиентский интерфейс с удобной межстраничной навигацией и интуитивно понятными элементами, обеспечивающий приятное взаимодействие пользователей с приложением.

6.2 Настройка Redux и Axios

С помощью библиотеки Axios была реализована обработка запросов пользователя. Важно отметить, что выбор React и связанных с ним библиотек был обусловлен их популярностью, широкой поддержкой сообщества разработчиков и возможностью легкой интеграции с другими инструментами и библиотеками. Реализация настройки Axios представлена на листинге 6.2.1.

Листинг 6.2.1 – реализация Axios

```
// axiosInstance.js
import axios from 'axios';
import Cookies from 'js-cookie';

const axiosInstance = axios.create({
  baseURL: 'https://bookshelf-cq3i.onrender.com/api',
  // baseURL: 'http://0.0.0.0:3001/api',
});

axiosInstance.interceptors.request.use((config) => {
  const token = Cookies.get('token');
  if (token) {
    config.headers.Authorization = `Bearer ${token}`;
  }
  return config;
}, (error) => {
  return Promise.reject(error);
});

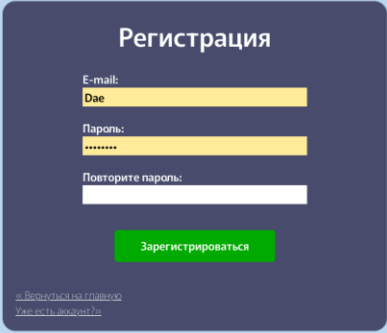
export default axiosInstance;
```


Для управления состоянием приложения был использован Redux, который предоставляет удобные для этого методы и функции. Reducer хранит состояние и позволяет изменять его или получать. В приложении А на листинге 9 представлена реализация Cart Reducer в качестве примера.

6.3 Страницы приложения

Ниже приведен список страниц, реализованных в приложении для обычного пользователя:

- 1) страница регистрации (Рисунок 6.3.1) - здесь пользователи могут зарегистрировать свои аккаунты, предоставив необходимую информацию;
- 2) страница входа (Рисунок 6.3.2) - пользователи могут войти в свои аккаунты, предоставив соответствующие учетные данные;
- 3) главная страница (Рисунок 6.3.3) - эта страница является основным интерфейсом приложения, где пользователи найти интересные коллекции и просмотреть книги добавленные другими пользователями;
- 4) страница «МояПолка» (Рисунок 6.3.4) - здесь пользователи добавлять свои книги и просматривать свои коллекции.
- 5) Так же на рисунках 6.3.5 и 6.3.6 показаны карточки коллекций и книг



The image shows a registration form titled "Регистрация" (Registration) on a dark blue background. It contains three input fields: "E-mail:" with the placeholder "Dae", "Пароль:" (Password) with masked characters "*****", and "Повторите пароль:" (Repeat password). Below the fields is a green button labeled "Зарегистрироваться" (Register). At the bottom, there are two links: "или вернуться на главную" (or return to home) and "или восстановить пароль" (or recover password).

Рисунок 6.3.1 - Страница регистрации

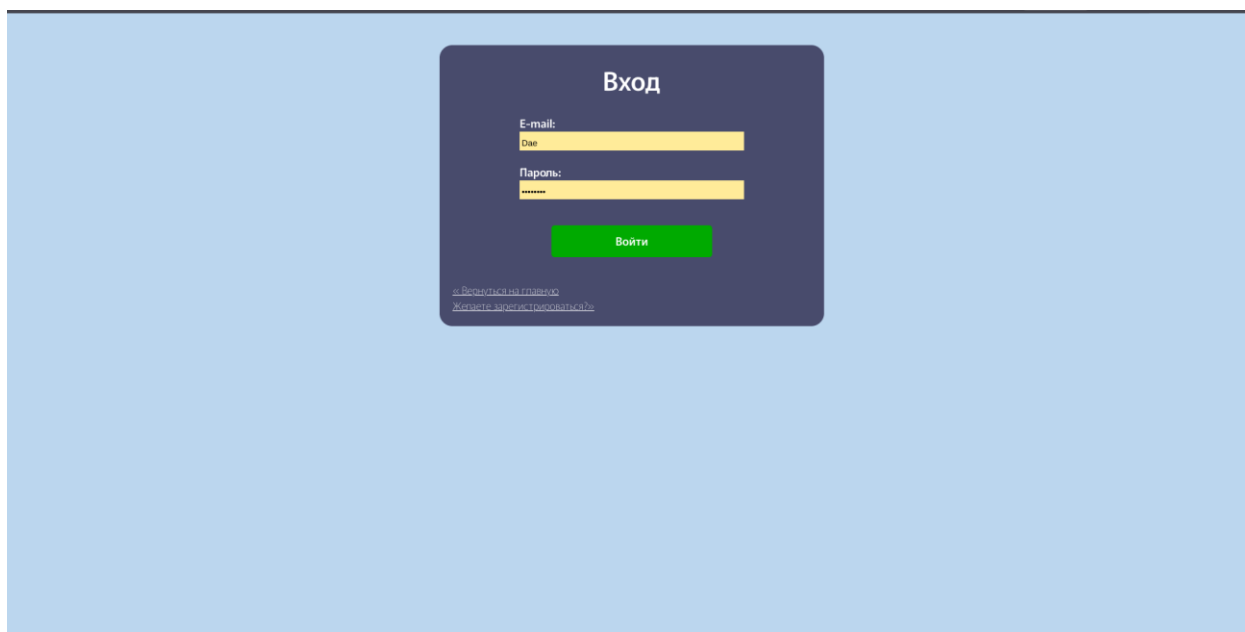


Рисунок 6.3.2 - Страница входа

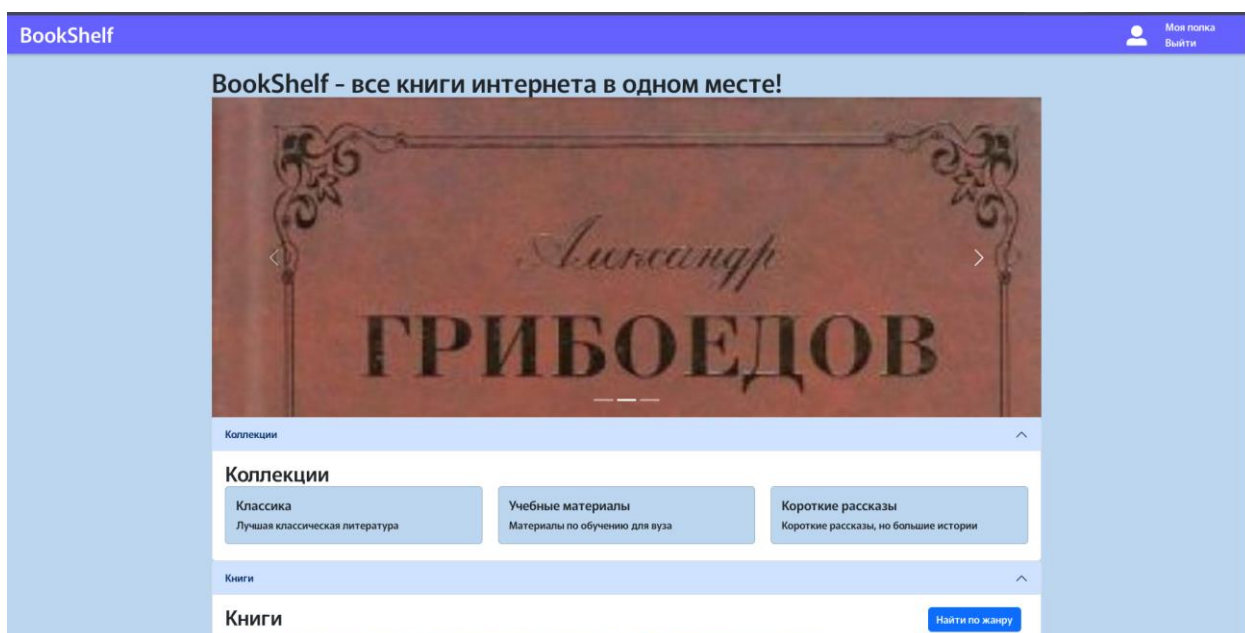


Рисунок 6.3.3 - Главная страница

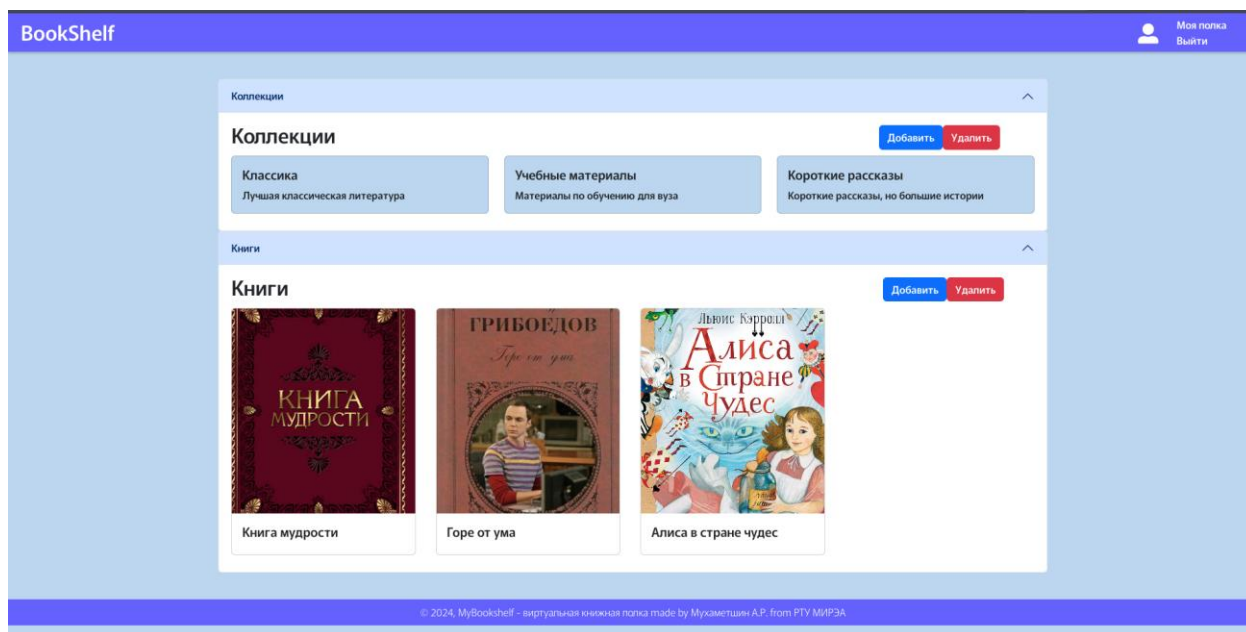


Рисунок 6.3.4 - Страница «МояПолка»

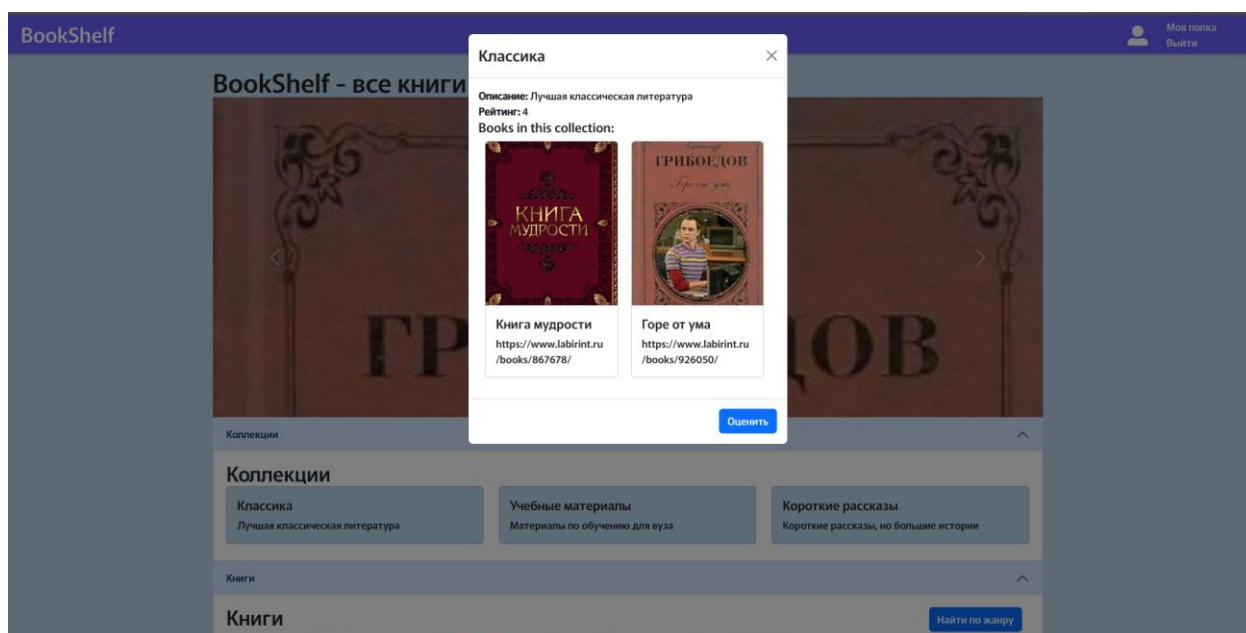


Рисунок 6.3.5 - Карточка коллекции

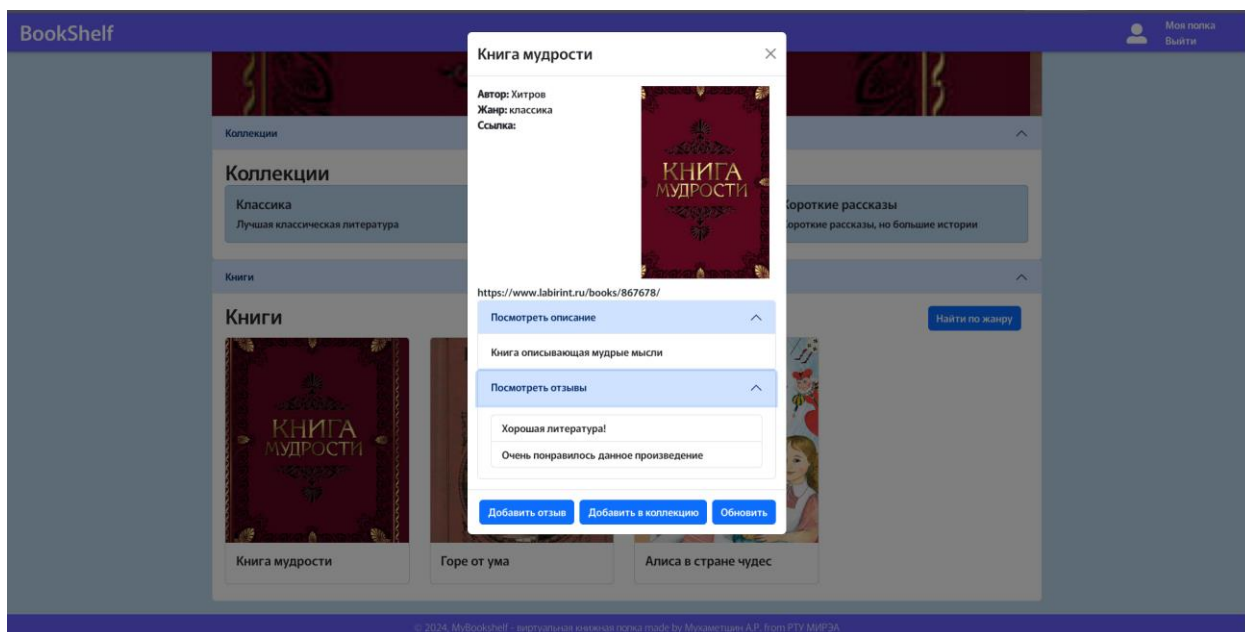


Рисунок 6.3.6 – Карточка книги

Также был реализован админский функционал в котором в у каждой сущности появляется кнопки редактирования и удаления.

Вывод к главе 6

В данной главе были рассмотрены ключевые аспекты создания клиентской части приложения, настройки инструментов для работы с API и управления состоянием, а также реализация пользовательских и административных интерфейсов.

Итогом данной главы является создание функционального и удобного клиентского интерфейса с использованием современных инструментов и библиотек, что значительно улучшило взаимодействие пользователей с приложением и упростило его управление для администраторов.

7 ПРОВЕДЕНИЕ ТЕСТИРОВАНИЯ РАЗРАБОТАННОГО ПРИЛОЖЕНИЯ

Тестирование проводилось с помощью программы Insomnia, которая предоставляет все необходимые функции для проверки работоспособности запросов.

На рисунке 7.1 изображено тестирование регистрации. Отправляется запрос с именем пользователя и паролем, при этом возвращается JWT токен и id пользователя.

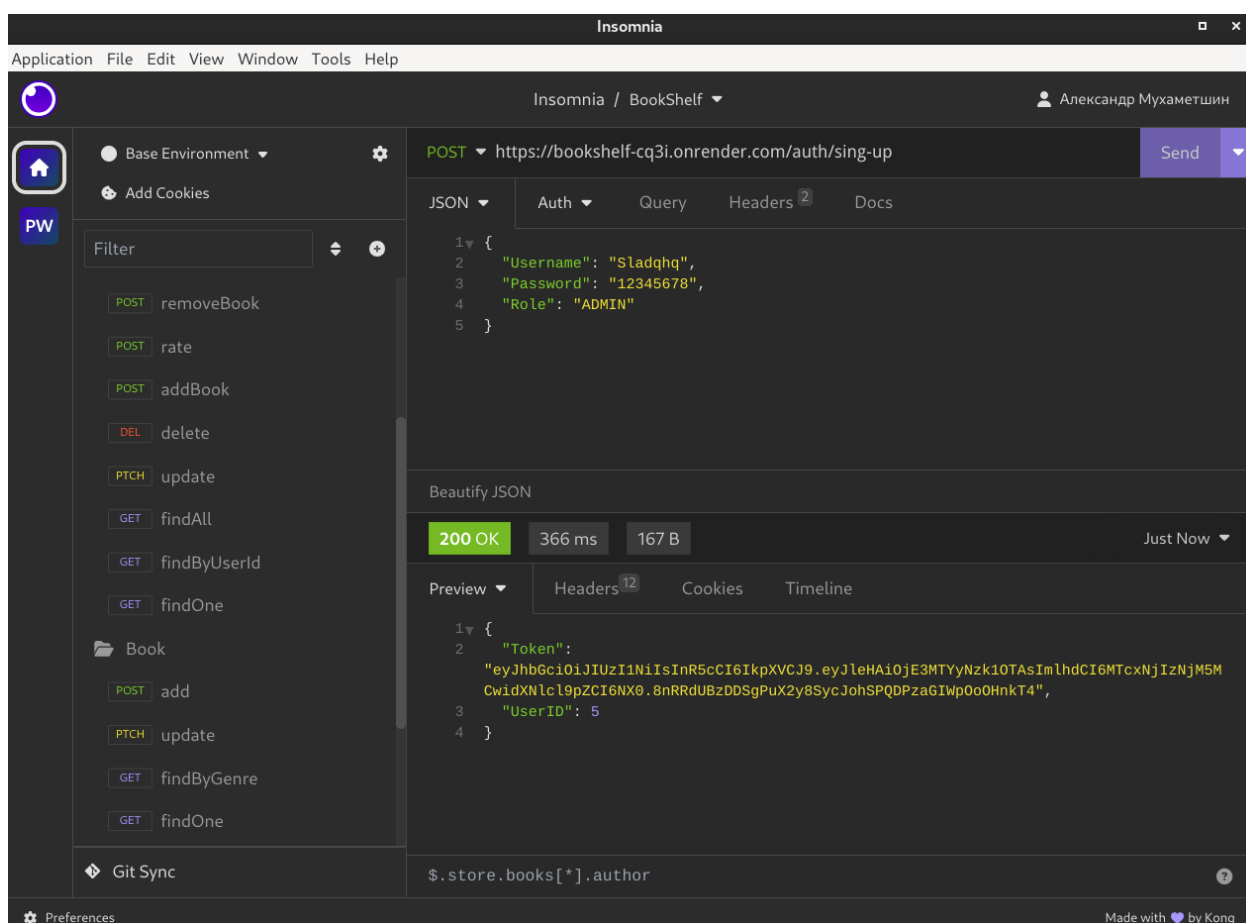


Рисунок 7.1 - Тестирование регистрации

На рисунке 7.2 изображено тестирование получения списка всех книг. При этом возвращается детальная информация о книгах и статус 200.

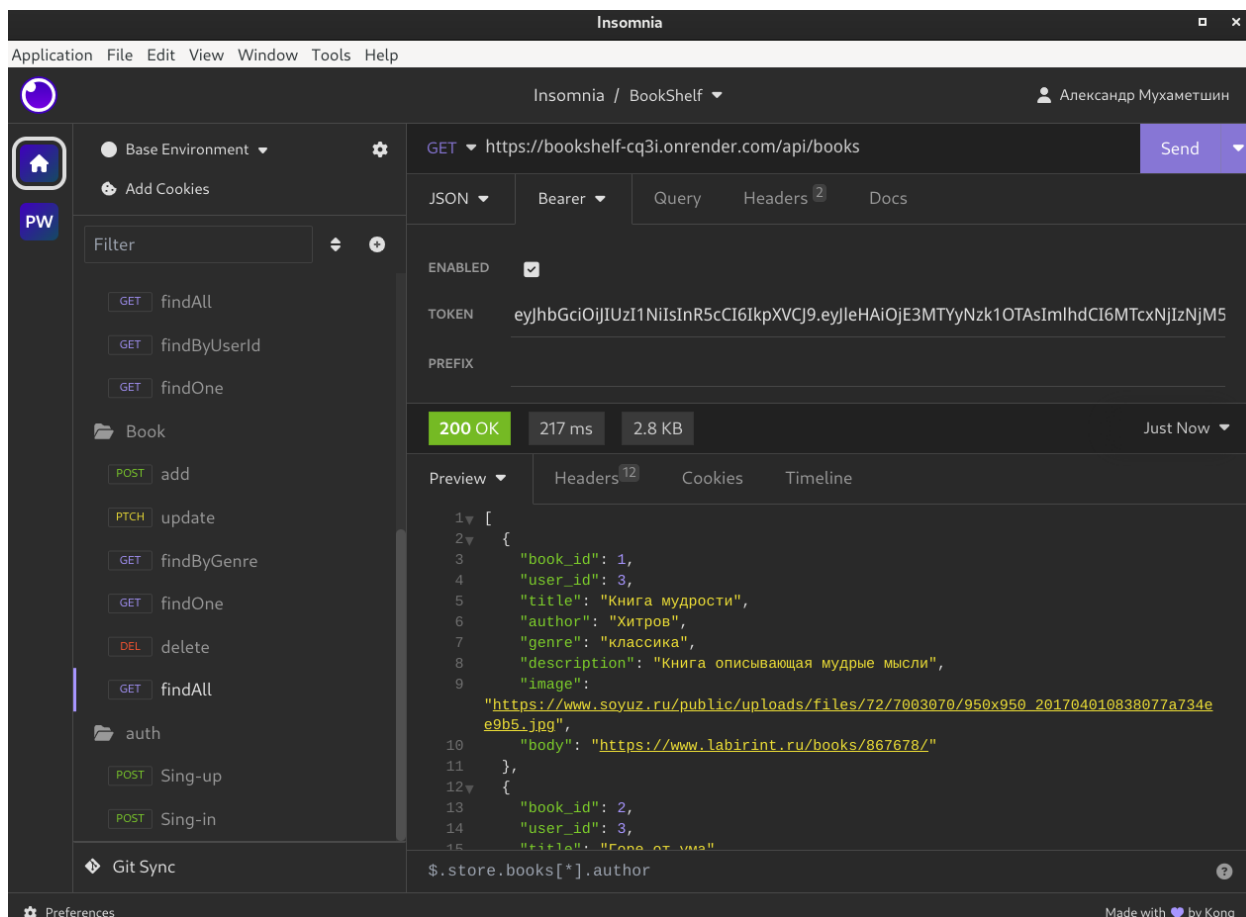


Рисунок 7.2 - Тестирование получения списка коллекций

На рисунке 7.3 изображено тестирование добавления книги. Отправляется запрос с данными книги в теле запроса и токен в авторизации, при этом возвращается статус 200 и идентификатор новой книги.

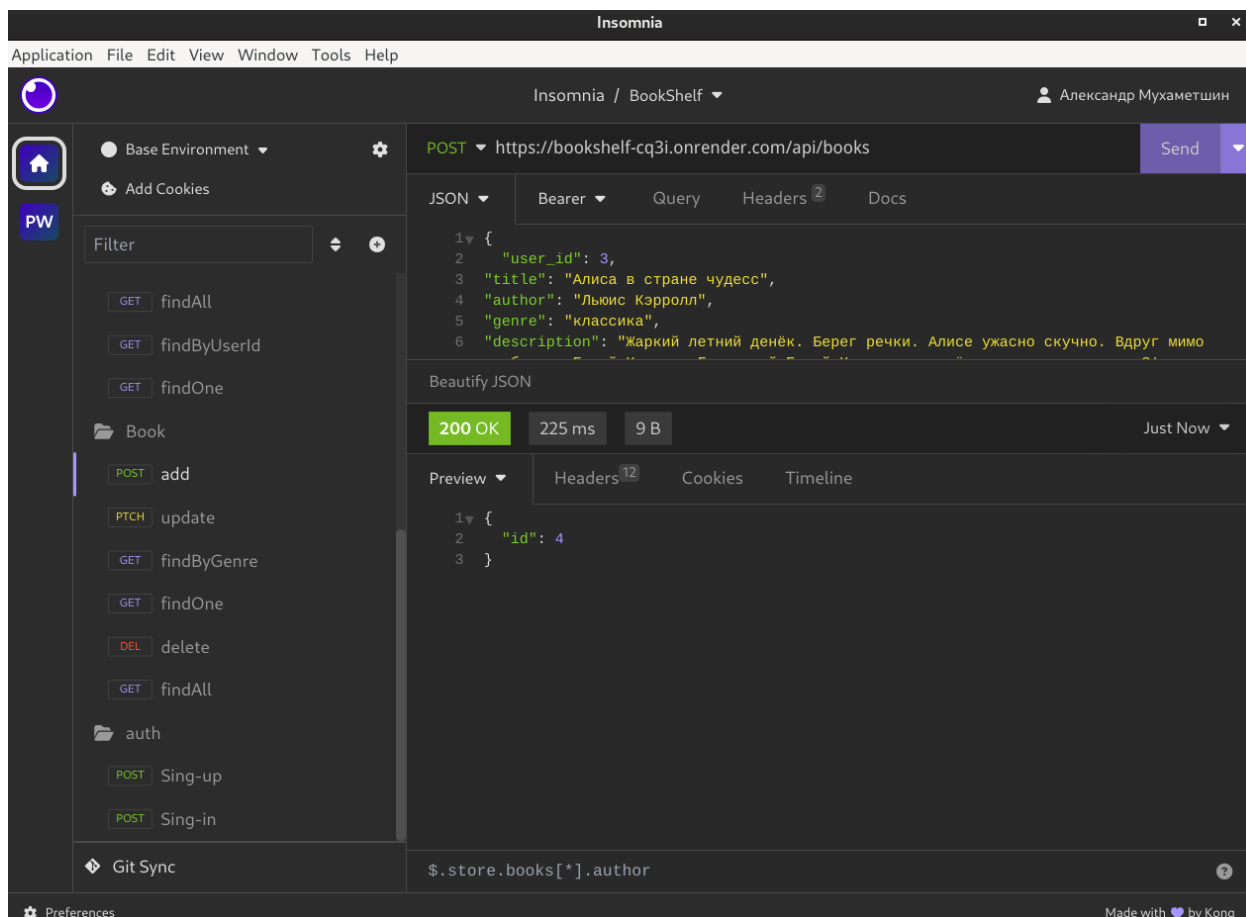


Рисунок 7.3 - Тестирование добавления книги

На рисунке 7.4 изображено тестирование удаления книги. Отправляется запрос, при этом возвращается статус 200.

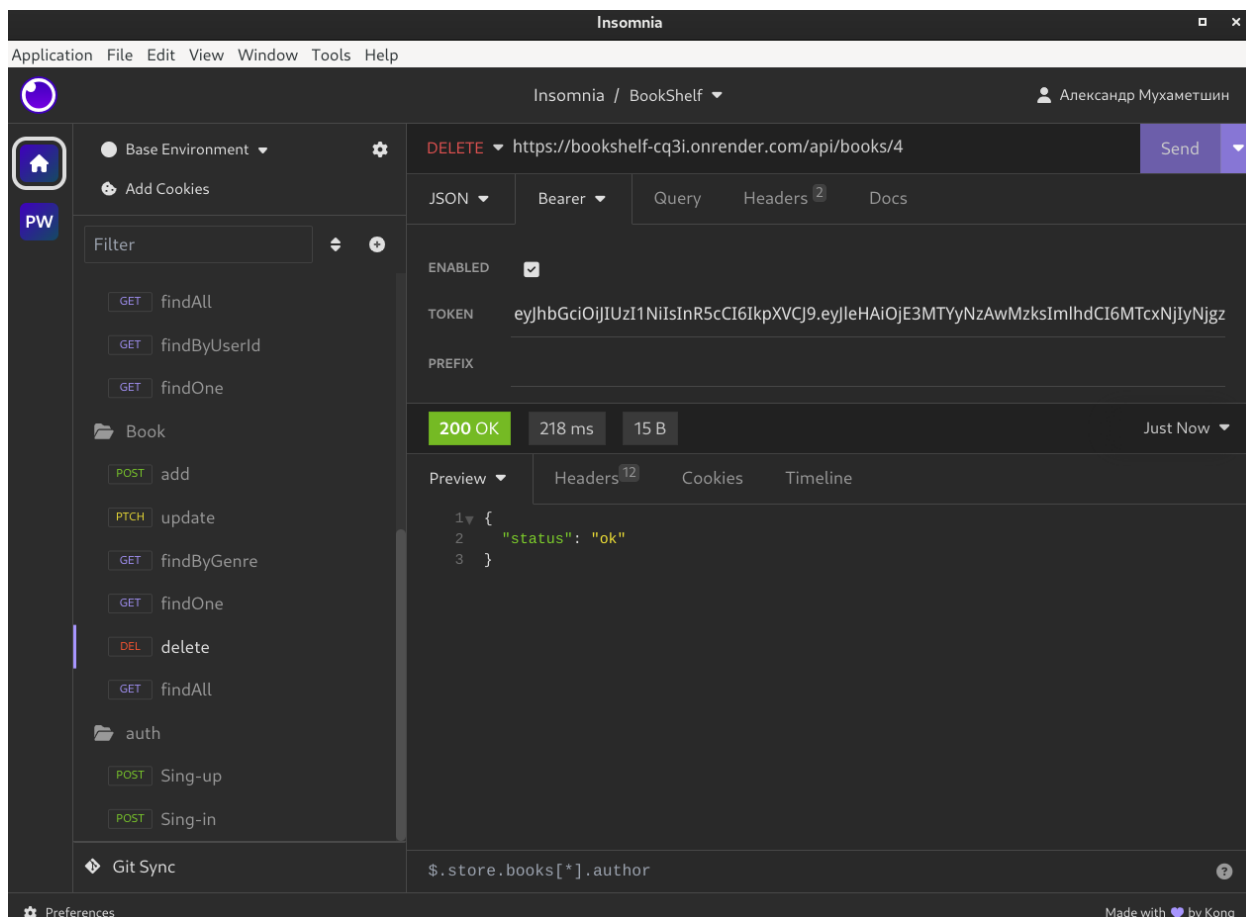


Рисунок 7.4 - Тестирование удаления книги

На рисунке 7.5 изображено тестирование получения одной коллекции. Отправляется запрос, при этом возвращается детальная информация о коллекции и статус 200.

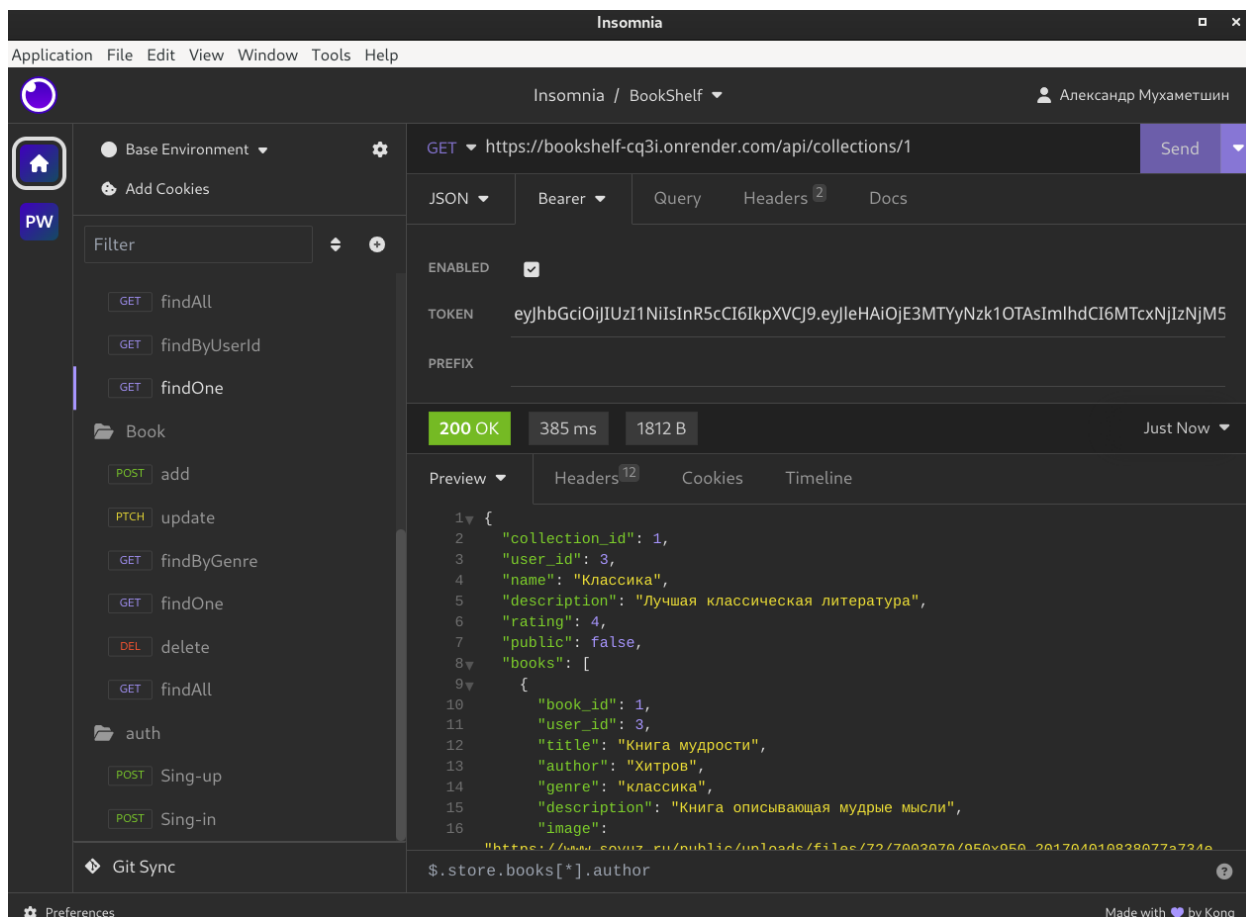


Рисунок 1 – Тестирование получения коллекции

Аналогичным способом был протестирован весь остальной функционал – создание, обновление, вывод и удаление всех сущностей, а так же специальные функции, такие как добавление книги в коллекцию.

Вывод к главе 7

Было успешно проведено тестирование разработанного веб-приложения с помощью Insomnia. Все тесты прошли успешно.

8 РАЗВЕРТЫВАНИЕ ИНТЕРНЕТ-РЕСУРСА

Развертывание интернет-ресурса является важным этапом, обеспечивающим доступность и стабильную работу приложения в производственной среде. Для развертывания данного проекта была выбрана платформа Render, которая предоставляет удобные и мощные инструменты для деплоя современных веб-приложений.

8.1 Выбор платформы для развертывания

Выбор Render для развертывания проекта был обусловлен следующими преимуществами:

- Простота использования: Render предлагает интуитивно понятный интерфейс и понятные инструкции для настройки и развертывания приложений.

- Интеграция с GitHub: Возможность автоматического развертывания из репозитория на GitHub, что упрощает процесс обновления и деплоя.

- Поддержка различных технологий: Render поддерживает широкий спектр технологий и стэков, что делает его универсальным решением для различных типов приложений.

- Автоматическое масштабирование: Платформа обеспечивает автоматическое масштабирование в зависимости от нагрузки, что позволяет поддерживать стабильную работу приложения при увеличении числа пользователей.

8.2 Процесс развертывания

Процесс развертывания на Render включает следующие основные шаги:

- 1) Создание аккаунта и настройка проекта:
- 2) Настройка окружения,
- 3) Настройка и запуск деплоя,
- 4) Мониторинг и управление.

8.3 Развертывание сервисной и клиентской частей

Далее необходимо было задеплоить сервисную и клиентскую части, для этого был указан репозиторий с исходным кодом и написаны команды для

сборки и запуска, что можно видеть на рисунках 8.3.1 и 8.3.2 соответственно.

Build Command

This command runs in the root directory of your repository when a new version of your code is pushed, or when you deploy manually. It is typically a script that installs libraries, runs migrations, or compiles resources needed by your app.

```
$ go build -o ./app cmd/main.go
```

Edit

Pre-Deploy Command Optional

This command runs before starting your service. It is typically used for tasks like running a database migration or uploading assets to a CDN.

```
$
```



Edit

Start Command

This command runs in the root directory of your app and is responsible for starting its processes. It is typically used to start a webserver for your app. It can access environment variables defined by you in Render.

```
$ ./app
```

Edit

Рисунок 8.3.1 - Команды для сборки и запуска серверной части

Build Command

This command runs in the root directory of your repository when a new version of your code is pushed, or when you deploy manually. It is typically a script that installs libraries, runs migrations, or compiles resources needed by your app.

```
$ npm run build
```

Edit

Рисунок 8.3.2 - Команда для сборки и запуска клиентской части

На рисунках 8.3.3 – 8.3.4 изображена успешность развертывания.

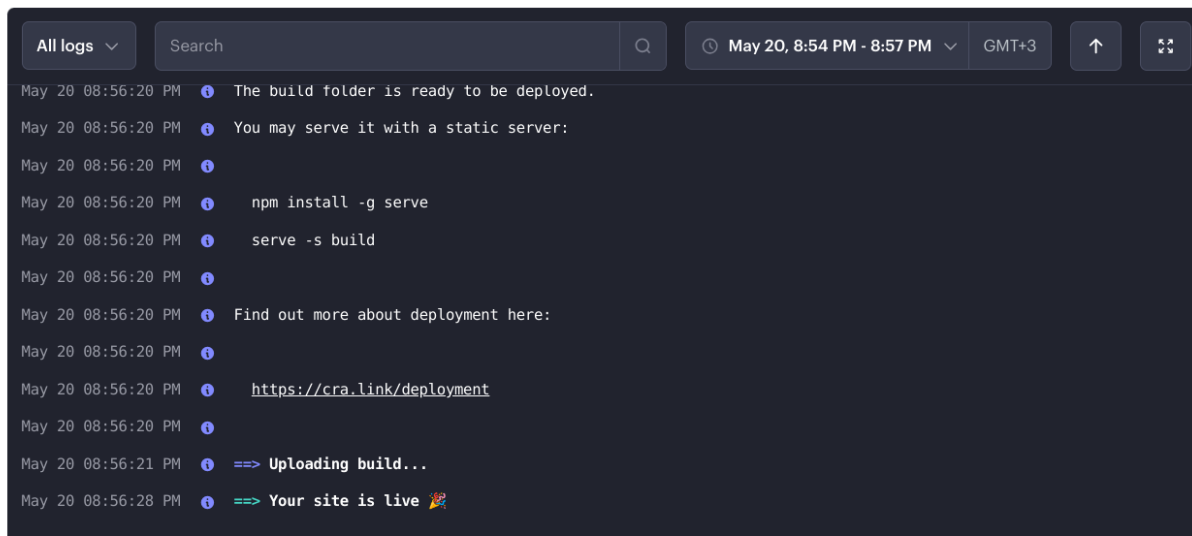
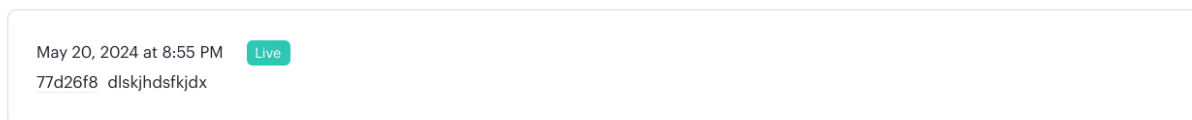


Рисунок 8.3.3 - Логи клиентской части

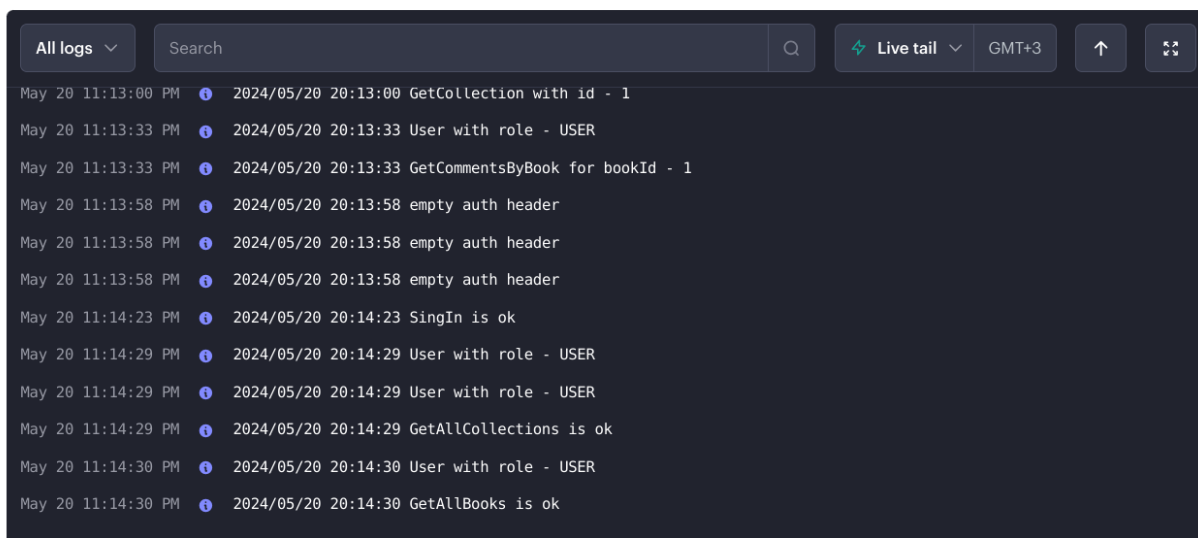
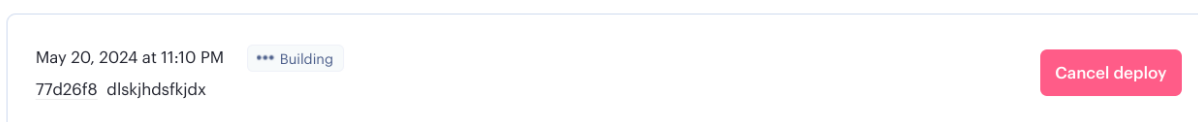


Рисунок 8.3.4 - Логи серверной части

Вывод к главе 8

В данной главе были рассмотрены ключевые аспекты развертывания интернет-ресурса, обеспечивающие его доступность и стабильную работу в производственной среде. Платформа Render была выбрана для деплоя проекта благодаря ряду преимуществ, таких как простота использования, интеграция

с GitHub, поддержка различных технологий и автоматическое масштабирование. Эти особенности позволили упростить процесс настройки и развертывания приложения, обеспечив его надежную и эффективную работу в условиях реального использования. Выбор Render для развертывания данного проекта способствовал сокращению времени на внедрение и улучшению общей производительности и устойчивости веб-приложения.

ЗАКЛЮЧЕНИЕ

В ходе выполнения курсовой работы была освоена компетенция «Разработка клиент-серверных приложений» в степени, соответствующей рабочей программе дисциплины. Результатом выполнения курсовой работы на тему «Разработка клиент-серверного фуллстек-приложения для создания виртуальной книжной полки пользователей» стало разработанное серверное программное приложение, которое написано с использованием языка Go и React. Для создания серверного приложения требовались навыки отличного владения и знания всех нюансов среды разработки, в которой написана работа, поэтому в ходе её выполнения приобретены навыки работы в Goland. Также изучена библиотека JavaScript – React. Она позволяет создавать компоненты, которые описывают структуру и внешний вид элементов интерфейса, и управлять ими динамически.

Благодаря глубокому анализу предметной области, учтены и проработаны все недочёты 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.

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 организации и управления виртуальной книжной полкой.

Исходный код Интернет-ресурса по курсовой работе доступен по ссылке: <https://github.com/SlasherSDCaT/BookShelf>

Приложение было развернуто на сервисе Render и доступно по ссылке: <https://bookshelf-1-ph0o.onrender.com/>

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Раджпут Д. Все паттерны проектирования. – СПб.: Питер, 2019.
2. Создание микросервисов. 2-е изд. — СПб.: Питер, 2023. — 624 с.: ил. — (Серия «Бестселлеры O'Reilly») ISBN 978-5-9775-6723-7.
3. Ричардсон, С. Шаблоны микросервисов: - 1-е издание. - Manning Publications, 2020. - 526 с. - ISBN 978-1617294549.
4. Тузовский, А. Ф. Проектирование и разработка web-приложений: учебное пособие для вузов / А. Ф. Тузовский. — Москва: Издательство Юрайт, 2021. — 218 с. — (Высшее образование). — ISBN 978-5-534-00515-8. — Текст : электронный // Образовательная платформа Юрайт [Электронный ресурс]. — URL: <https://urait.ru/bcode/469982> (дата обращения: 1.04.2024).
5. Сьерра Кэти, Бэйтс Берт Head First / Сьерра Кэти, Бэйтс Берт — СПб. : Эксмо, 2023. — 720 с. — ISBN 978-5-699-54574-2.
6. Мартин, Р. Чистая архитектура. Искусство разработки программного обеспечения / Р. Мартин. — СПб. : Питер, 2021. — 352 с.
7. Goland [Электронный ресурс] – URL: <https://www.jetbrains.com/go/> (дата обращения 17.01.2024).
8. Golang [Электронный ресурс] – URL: <https://go.dev/> (дата обращения 17.04.2024).
9. React [Электронный ресурс] – URL: <https://legacy.reactjs.org/docs/gettingstarted.html?url=https%3A%2F%2Freactjs.org%2Fdocs%2Fgetting-started.html> (дата обращения 20.04.2024).
10. MongoDB [Электронный ресурс] – URL: <https://www.mongodb.com/> (дата обращения 20.04.2024).
11. Gorilla Mux [Электронный ресурс] – URL: <https://github.com/gorilla/mux> (дата обращения 21.04.2024).
12. Redux [Электронный ресурс] – URL: <https://redux.js.org/> (дата обращения 21.04.2024).
13. GitHub [Электронный ресурс] – URL: <https://docs.github.com/ru> (дата обращения 20.04.2024).

14. JSON [Электронный ресурс] – URL: <https://www.json.org/json-en.html> (дата обращения 2.05.2024).
15. Swagger [Электронный ресурс] – URL: <https://editor.swagger.io/> (дата обращения 2.05.2024).
16. Postman [Электронный ресурс] – URL: <https://www.postman.com/> (дата обращения 2.05.2024).
17. MongoDB Go Driver [Электронный ресурс] – URL: <https://www.mongodb.com/docs/drivers/go/current/> (дата обращения 2.05.2024).
18. Render [Электронный ресурс] – URL: <https://render.com/> (дата обращения 2.05.2024).
19. Hands-On RESTful API Design Patterns and Best Practices / Harihara Subramanian, Pethuru. — Raj Packt Publishing Ltd, 2019. — 378 с. — ISBN 978-1788992664.
20. Хоффман Эндрю X85 Безопасность веб-приложений. — СПб.: Питер, 2021. — 336 с.: ил. — (Серия «Бестселлеры O'Reilly»).