МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет Информационных Технологий

Кафедра Информационных систем и технологий

Специальность 1-40 05 01 «Информационные системы и технологии»

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

**К КУРСОВОМУ ПРОЕКТУ НА ТЕМУ:**

Веб-приложение для управления библиотекой книг

Выполнил студент Халалеенко Андрей Николаевич

(Ф.И.О.)

Руководитель проекта ассистент Кантарович В.С.

(учен. степень, звание, должность, подпись, Ф.И.О.)

Заведующий кафедрой

(учен. степень, звание, должность, подпись, Ф.И.О.)

Курсовой проект защищен с оценкой

Минск 2024

Содержание

[Введение 4](#_Toc185278399)

[1. Постановка задачи веб-приложения 5](#_Toc185278400)

[1.1 Обзор аналогов веб приложения 5](#_Toc185278401)

[1.2 Постановка задач веб-приложения 7](#_Toc185278402)

[1.3 Вывод по разделу 7](#_Toc185278403)

[2. Проектирование веб-приложения 8](#_Toc185278404)

[2.1 Архитектура веб-приложения 8](#_Toc185278405)

[2.2 Описание работы веб-приложения 9](#_Toc185278406)

[2.3 Описание диаграммы использования 10](#_Toc185278407)

[2.4 Проектирование базы данных веб-приложения 12](#_Toc185278409)

[2.5 Проектирование основных разделов 16](#_Toc185278410)

[2.5.1 Алгоритм смены роли пользователя на автора 16](#_Toc185278411)

[2.5.2 Алгоритм работы страницы закладки книг 18](#_Toc185278412)

[2.6 Вывод по разделу 18](#_Toc185278413)

[3. Разработка веб-приложения 19](#_Toc185278414)

[3.1 Разработка серверной части 20](#_Toc185278415)

[3.2 Разработка клиентской части 23](#_Toc185278416)

[3.3 Вывод по разделу 25](#_Toc185278417)

[4. Тестирование веб-приложения 26](#_Toc185278418)

[4.1 Тестирование под видом клиента 26](#_Toc185278419)

[4.2 Вывод по разделу 28](#_Toc185278420)

[5. Руководство пользователя 28](#_Toc185278421)

[5.1 Регистрация пользователя 28](#_Toc185278422)

[5.2 Аутентификация и авторизация пользователя 29](#_Toc185278423)

[5.3 Главная страница 30](#_Toc185278424)

[5.4 Страница книги 30](#_Toc185278425)

[5.5 Личная страница пользователя 31](#_Toc185278426)

[5.6 Панель администратора 31](#_Toc185278427)

[5.7 Удаление комментариев 33](#_Toc185278428)

[5.8 Вывод по разделу 33](#_Toc185278429)

[Заключение 34](#_Toc185278430)

[Список используемых источников 35](#_Toc185278431)

[ПРИЛОЖЕНИЕ А 36](#_Toc185278432)

[ПРИЛОЖЕНИЕ Б 39](#_Toc185278433)

Введение

Современное информационное общество требует доступа к высококачественной литературе, и управление большими библиотеками книг может представлять сложность без эффективных инструментов и ресурсов. В свете этой проблемы становится актуальным и востребованным разработка функционального веб-приложения для управления библиотекой книг.

Целью данного курсового проекта является создание такого веб-приложения, которое позволит пользователям эффективно управлять библиотекой книг. Главная цель заключается в разработке полнофункционального приложения, обеспечивающего возможность регистрации и авторизации пользователей, поддержку ролей администратора и пользователя, добавление новых книг и информации о них, чтение и скачивание книг, редактирование информации, поиск книг по различным критериям, просмотр информации о книгах, оставление отзывов и оценок, а также создание личных списков прочитанных книг, списка «хочу прочитать» и списка «избранное», а также «прочитанное» и «читаю сейчас». Для достижения этих целей были выполнены следующие задачи:

* постановка задачи;
* проектирование веб-приложения;
* разработка веб-приложения;
* тестирование веб-приложения;
* составление руководства пользователя.

Для разработки веб-приложения были выбраны современные технологии и инструменты, которые позволили скорость разработки и удобство использования приложения.

В данном проекте на стороне сервера были использованы два популярных инструмента: Express и Socket.IO. Для хранения данных была использована база данных MySQL, которая является одной из наиболее распространенных реляционных баз данных. Для работы с базой данных MySQL в проекте был использован ORM-фреймворк Prisma. Таким образом, Express обеспечивает обработку HTTP-запросов и маршрутизацию на стороне сервера, Socket.IO обеспечивает двунаправленную связь в режиме реального времени между клиентом и сервером, MySQL служит в качестве хранилища данных, а Prisma упрощает взаимодействие с базой данных и выполнение запросов. Эти инструменты совместно позволяют разрабатывать функциональное и эффективное веб-приложение для управления библиотекой книг.

На стороне клиента для разработки пользовательского интерфейса был использован React Bootstrap. Bootstrap — это открытый и бесплатный HTML, CSS и JS фреймворк, который используется веб-разработчиками для быстрой вёрстки адаптивных дизайнов сайтов и веб-приложений.

Вся инфраструктура приложения законтейнеризирована в Docker, благодаря чему приложения с БД могут изолированно взаимодействовать между собой [5].

1. Постановка задачи веб-приложения

Изучая, аналогичные веб-приложения, были найдены схожие по теме и функционалу приложения. Большая часть веб-приложений похожи друг на друга, и функционалом почти не отличаются. Так же были найдены статьи, которые помогли реализовать некоторые функции. Результат анализа представлен ниже.

## **Обзор аналогов веб приложения**

Goodreads.com – это популярная онлайн-платформа для любителей чтения книг. Она предоставляет возможность пользователям искать, открывать и оценивать книги. Goodreads.com также позволяет пользователям составлять личные списки прочитанных книг, добавлять отзывы и рекомендации, а также общаться с другими читателями через сообщества и группы обсуждения. Сайт предлагает разнообразные функции для удобного взаимодействия с книжным контентом и обмена литературными впечатлениями.

Интерфейс этого сайта можно увидеть на рисунке 1.1.

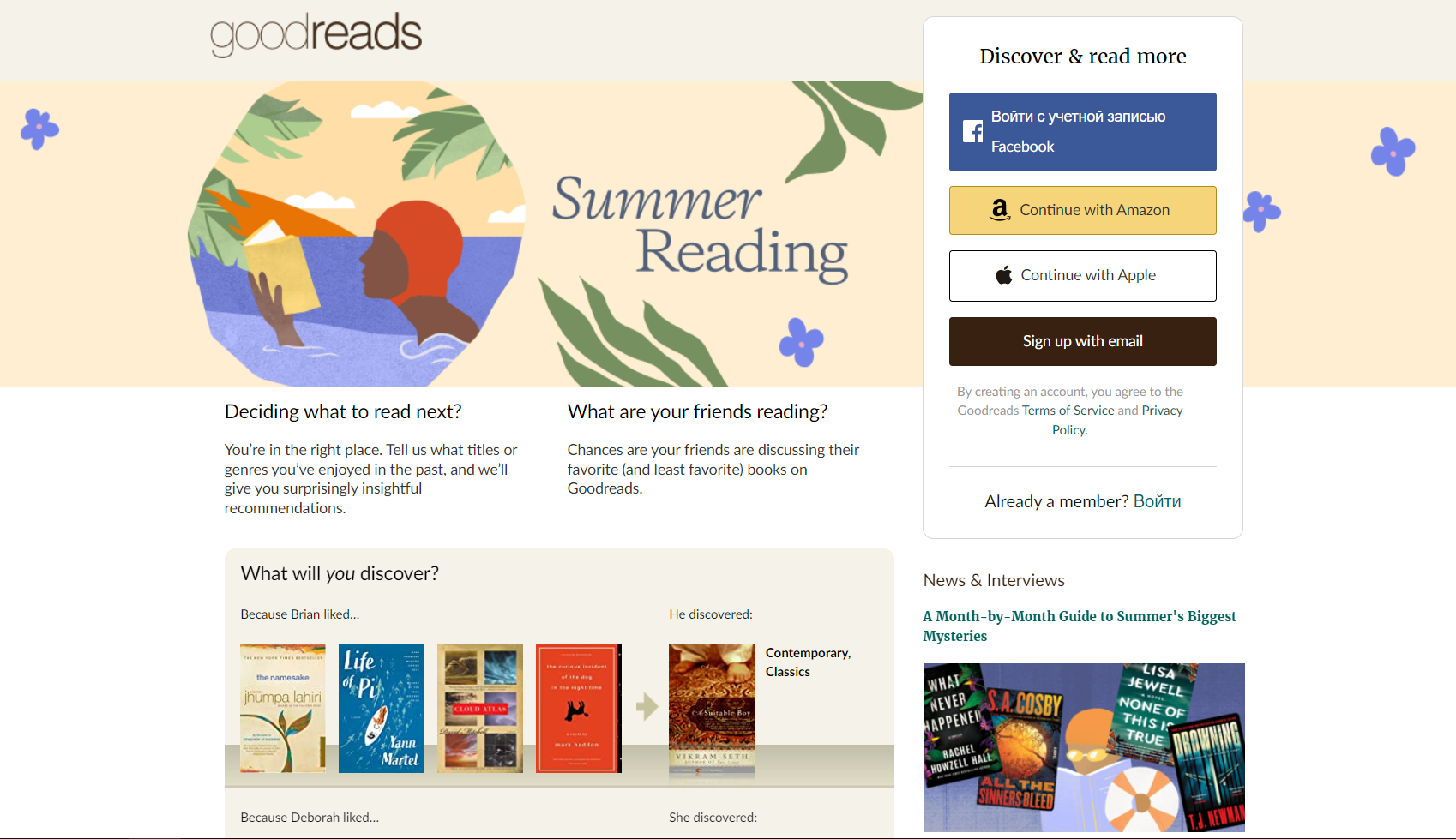


Рисунок 1.1 – Домашняя страница Goodreads

Преимущества данного сайта представлены ниже:

* большой каталог книг;
* сообщество чтения: Goodreads предоставляет возможность вступить в сообщество чтения, где пользователи могут обмениваться рекомендациями, отзывами и обсуждать книги с другими читателями;
* личные списки и отслеживание прочитанного. Пользователи могут создавать личные списки прочитанных книг, отмечать книги, которые они хотят прочитать, и отслеживать свой прогресс чтения. Это помогает пользователям организовать свою литературную активность и следить за своими литературными интересами.

Project Gutenberg - это онлайн-библиотека, которая предоставляет бесплатный доступ к большой коллекции электронных книг. Сайт B характеризуется огромным выбором литературных произведений разных жанров, включая классику, научно-популярные и художественные книги. Пользователи могут бесплатно скачивать книги в различных форматах, таких как EPUB, MOBI, PDF и др. Project Gutenberg сосредоточен на предоставлении доступа к общественному достоянию литературы и является важным ресурсом для чтения книг в электронном формате.

Интерфейс этого сервиса можно увидеть на рисунке 1.2.

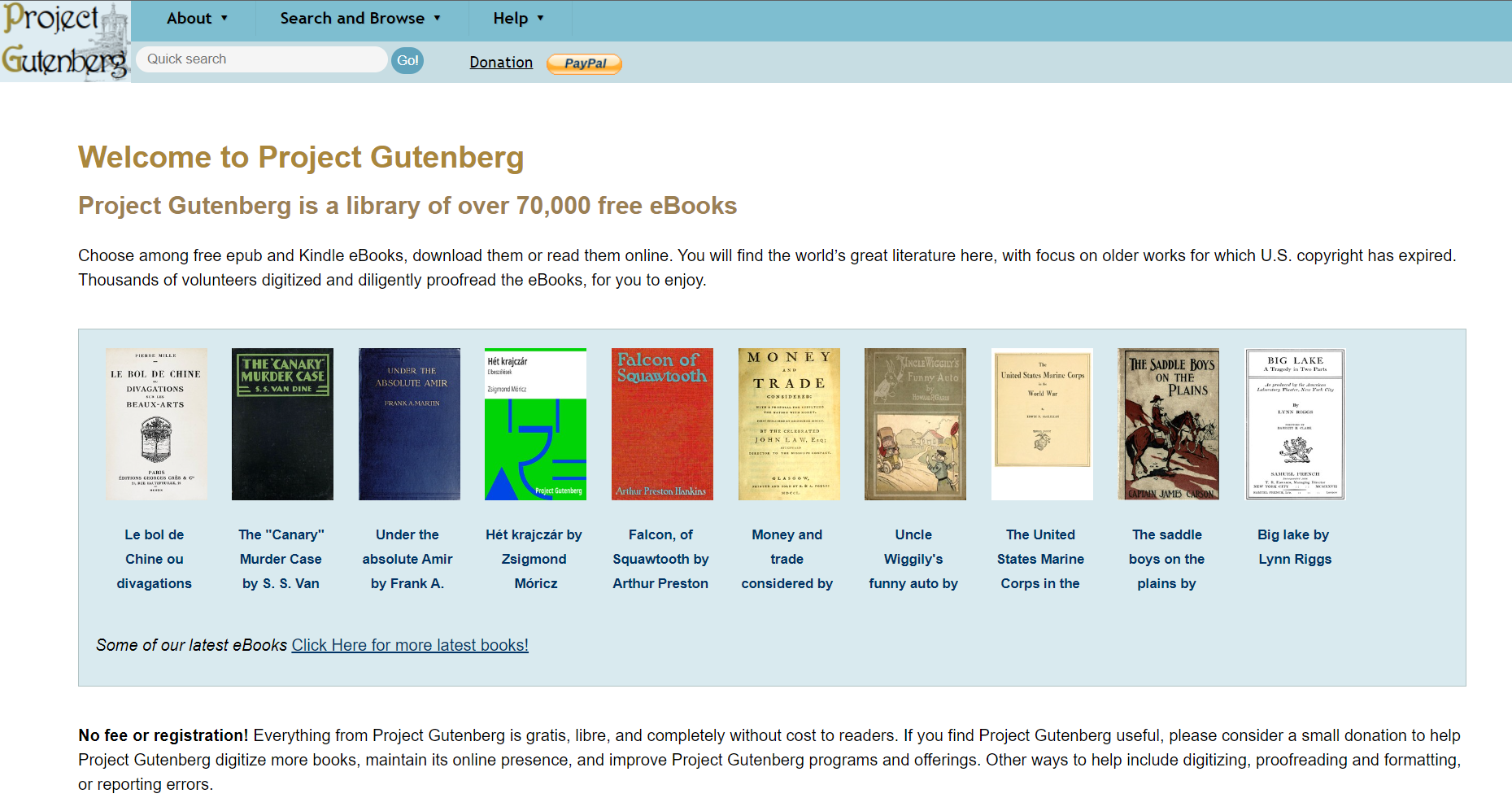


Рисунок 1.2 – Домашняя страница Project Gutenberg

В целом, Project Gutenberg предоставляет бесплатный и удобный доступ к большой коллекции книг в общественном достоянии, что делает его ценным ресурсом для чтения и распространения литературного наследия. Рассмотрим некоторые приемущества.

Project Gutenberg имеет несколько преимуществ:

* бесплатный доступ;
* общественное достояние;
* разнообразие книг. Коллекция Project Gutenberg сосредоточена на произведениях, находящихся в общественном достоянии. Это включает классические литературные работы, которые больше не защищены авторским правом. Пользователи могут получить доступ к произведениям, которые иначе могли бы быть недоступны или требовали бы платы;
* различные форматы. Книги в Project Gutenberg доступны в различных форматах, включая EPUB, MOBI, PDF и другие. Это позволяет пользователям выбирать формат, который наиболее удобен для чтения на их устройствах, будь то электронные книги, смартфоны или планшеты.

## **Постановка задач веб-приложения**

На основе проведенного анализа и учета требований, указанных в задании на курсовое проектирование, для разработки функционального веб-приложения по управлению библиотекой книг могут быть поставлены следующие задачи:

* разработка системы регистрации и авторизации: задача состоит в создании функционала, позволяющего пользователям регистрироваться на сайте, авторизовываться и сохранять их учетные данные в безопасном виде;
* создание системы ролей: задача заключается в реализации механизма, позволяющего выделять роли администратора и пользователя. Администратору предоставляются дополнительные привилегии для управления книгами и данными библиотеки;
* разработка функционала добавления и редактирования книг: задача состоит в создании возможности добавления новых книг в библиотеку с указанием соответствующей информации (название, автор, жанр и др.). Также необходимо реализовать функционал редактирования информации о существующих книгах;
* реализация функционала чтения и скачивания книг: задача заключается в создании механизма, позволяющего пользователям читать книги онлайн и скачивать их в удобном формате;
* разработка системы поиска книг: задача состоит в создании возможности поиска книг в библиотеке по различным критериям, таким как название, автор, жанр и т.д. Это позволит пользователям быстро находить интересующие их книги;
* реализация функционала отзывов и оценок: задача заключается в создании механизма, позволяющего пользователям оставлять отзывы на книги и ставить им оценки. Также необходимо предусмотреть возможность просмотра отзывов и оценок других пользователей;
* разработка функционала личных списков: задача состоит в создании функционала, позволяющего пользователям создавать личные списки прочитанных книг, списки «хочу прочитать» и «избранное», а также «читаю сейчас» и «прочитанные». Это позволит пользователям организовать и управлять своими литературными предпочтениями;

## **Вывод по разделу**

В результате анализа аналогичных веб-приложений и постановки задач были определены ключевые требования для разработки системы управления библиотекой книг. Изучение существующих решений позволило выделить функции, повышающие удобство и эффективность работы с приложением: регистрацию и авторизацию, поддержку ролей, добавление и редактирование книг, поиск, а также чтение и скачивание книг. Реализация личных списков, отзывов и оценок дополнительно позволяет пользователям персонализировать взаимодействие с библиотекой. На основе собранных данных определены задачи для создания полнофункционального приложения, отвечающего современным требованиям библиотечных систем.

1. Проектирование веб-приложения

Проект, разработанный в рамках данной работы, представляет собой клиент-серверную архитектуру, включающую два основополагающие части: клиентскую и серверную. Клиентская часть отвечает за взаимодействие с пользователем, предоставляет интерфейс для ввода данных и отображения результатов, в то время как серверный компонент обеспечивает обработку запросов, взаимодействие с базой данных и логику бизнес-процессов. Такое разделение функциональности позволяет эффективно управлять ресурсами и обеспечить надежную работу приложения.

## **Архитектура веб-приложения**

Общая структура проекта клиентская часть и серверная.

Проект состоит из клиентской и серверной частей. Клиент представляет собой веб-интерфейс, построенный с использованием библиотеки React [4]. Он позволяет клиенту отправлять запросы на сервер и отображать полученные ответы. На рисунке 2.1 предложена схема-развертывания приложения.

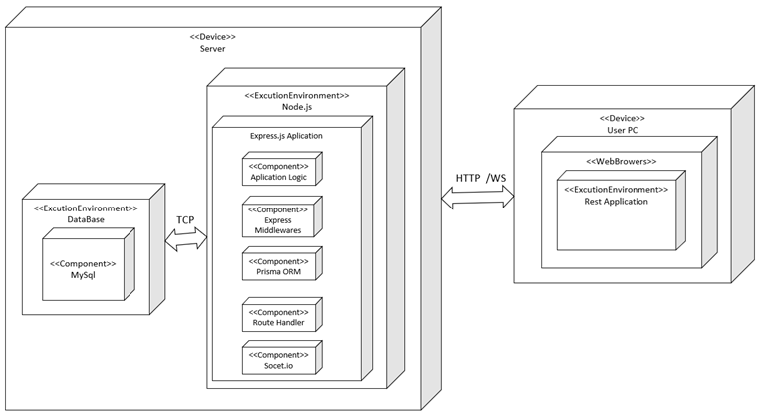
****

Рисунок 2.1 – Структурная схема приложения

Серверная часть приложения построена на базе фреймворка Express, который обеспечивает обработку входящих запросов от клиентской стороны [1]. Этот легковесный и гибкий инструмент позволяет разрабатывать серверную логику, определять маршруты и управлять API. Для взаимодействия с базой данных используется ORM (объектно-реляционное отображение) Prisma, что упрощает выполнение операций с базой данных, таких как чтение, запись, обновление и удаление данных [2]. Prisma позволяет безопасно и эффективно работать с различными реляционными базами данных, предоставляя гибкий и удобный API.

Взаимодействие между клиентской и серверной частью приложения осуществляется через протокол HTTPS, что обеспечивает безопасную передачу данных между пользователем и сервером. Клиентская часть отправляет запросы на заранее определённые серверные маршруты, а серверная часть обрабатывает эти запросы, выполняет необходимую бизнес-логику, взаимодействует с базой данных и возвращает результаты в формате JSON.

Для разработки пользовательского интерфейса используется библиотека React Bootstrap, которая предоставляет множество готовых компонентов, стилей и сеток. Это ускоряет процесс разработки, делая создание современного и отзывчивого дизайна интуитивным и удобным. Использование этой библиотеки обеспечивает визуальную консистентность, улучшая пользовательский опыт.

На стороне сервера подключены и настроены различные промежуточные компоненты (middlewares), которые расширяют функциональность и делают обработку запросов более надежной. Например:

* CORS используется для настройки политик доступа, позволяя серверу принимать запросы с разных источников (например, из других доменов).
* ErrorHandlerMiddleware служит для централизованной обработки ошибок, упрощая отладку и улучшая стабильность работы сервера.
* express-fileupload добавляет поддержку загрузки файлов, что позволяет пользователям загружать различные данные, такие как текстовые файлы, изображения для обложек книг или аватары пользователей. После загрузки файлы могут быть использованы клиентской частью для скачивания, отображения или чтения.

Для реализации функций в режиме реального времени в приложении интегрирована библиотека Socket.IO. Она позволяет устанавливать двустороннее соединение между клиентом и сервером, что обеспечивает мгновенное обновление данных на клиентской стороне без необходимости периодической отправки запросов. Это особенно важно для сценариев, где требуется высокая интерактивность, например, для отображения новых отзывов или обновлений данных без перезагрузки страницы.

В результате такая архитектура приложения обеспечивает надежную и отзывчивую работу веб-продукта. Пользователи могут эффективно управлять отзывами, работать с базой данных и загружать/скачивать файлы, наслаждаясь современным интерфейсом и бесшовным взаимодействием с серверной частью.

## **Описание работы веб-приложения**

При запуске программы, сервер настраивается для прослушивания входящих запросов на определенном порту. Клиентское приложение загружается в веб-браузере пользователя, и пользователь может взаимодействовать с интерфейсом, предоставленным React Bootstrap [6].

Когда пользователь выполняет определенные действия, например, отправляет запрос на добавление закладки книги, клиентское приложение формирует соответствующий HTTPS-запрос и отправляет его на сервер.

Сервер, используя фреймворк Express. Express представляет собой популярный веб-фреймворк, написанный на JavaScript и работающий внутри среды исполнения node.js. Этот модуль освещает некоторые ключевые преимущества этого фреймворка, установку среды разработки и выполнение основных задач веб-разработки и развёртывания. принимает запрос и маршрутизирует его к соответствующему обработчику маршрута. Обработчик маршрута выполняет необходимые операции, например, создает новую задачу в базе данных с использованием Prisma или рабаты с файлами. При успешном выполнении операции, сервер формирует ответ и отправляет его обратно клиентскому приложению.

Клиентское приложение получает ответ от сервера и обновляет свой интерфейс, отображая актуальные данные или информацию об успешном выполнении операции. В случае возникновения ошибок, сервер использует middleware для обработки и отправки соответствующего сообщения об ошибке обратно клиентскому приложению.

Кроме того, взаимодействие между клиентом и сервером в режиме реального времени осуществляется с использованием Socket.IO. Например, при появлении новой задачи, сервер отправляет уведомление о ней клиентскому приложению через WebSocket-соединение, и клиентское приложение мгновенно отображает это уведомление пользователю.

Данная диаграмма показывает, что клиентское приложение взаимодействует с сервером через протокол HTTP, а также осуществляет обмен данными с БД. Такая структура программного продукта позволяет пользователям взаимодействовать с сервером, отправлять запросы и получать ответы, а также обновлять данные в режиме реального времени с помощью сокетов. Это создает эффективное и отзывчивое веб-приложение для управления задачами и обработки данных из базы данных.

## **Описание диаграммы использования**

Диаграмма использования разработана для визуального представления функционала веб-приложения для управления библиотекой книг. На диаграмме представлены основные роли пользователей системы и соответствующие им функции, что позволяет наглядно продемонстрировать, какие действия доступны каждой роли и как эти действия взаимосвязаны.

Диаграмма использования включает следующие основные роли:

* гость – роль, представляющая пользователей, не прошедших авторизацию. Гости имеют доступ к базовым функциям, таким как регистрация и авторизация в приложении;
* пользователь – авторизованный участник системы, который наследует функционал роли гостя и обладает расширенными возможностями, такими как чтение и скачивание книг, управление личными списками, оставление отзывов и оценок, а также использование закладок;
* администратор – привилегированная роль, наследующая возможности пользователя, но также обладающая функциями для управления книгами, жанрами, тегами и авторами;
* автор – роль, похожая на администратора, наследующая функции пользователя и позволяющая управлять книгами, но не имеющая доступа к управлению жанрами и тегами.

Основной функционал приложения можно наглядно увидеть на рисунке 2.2.

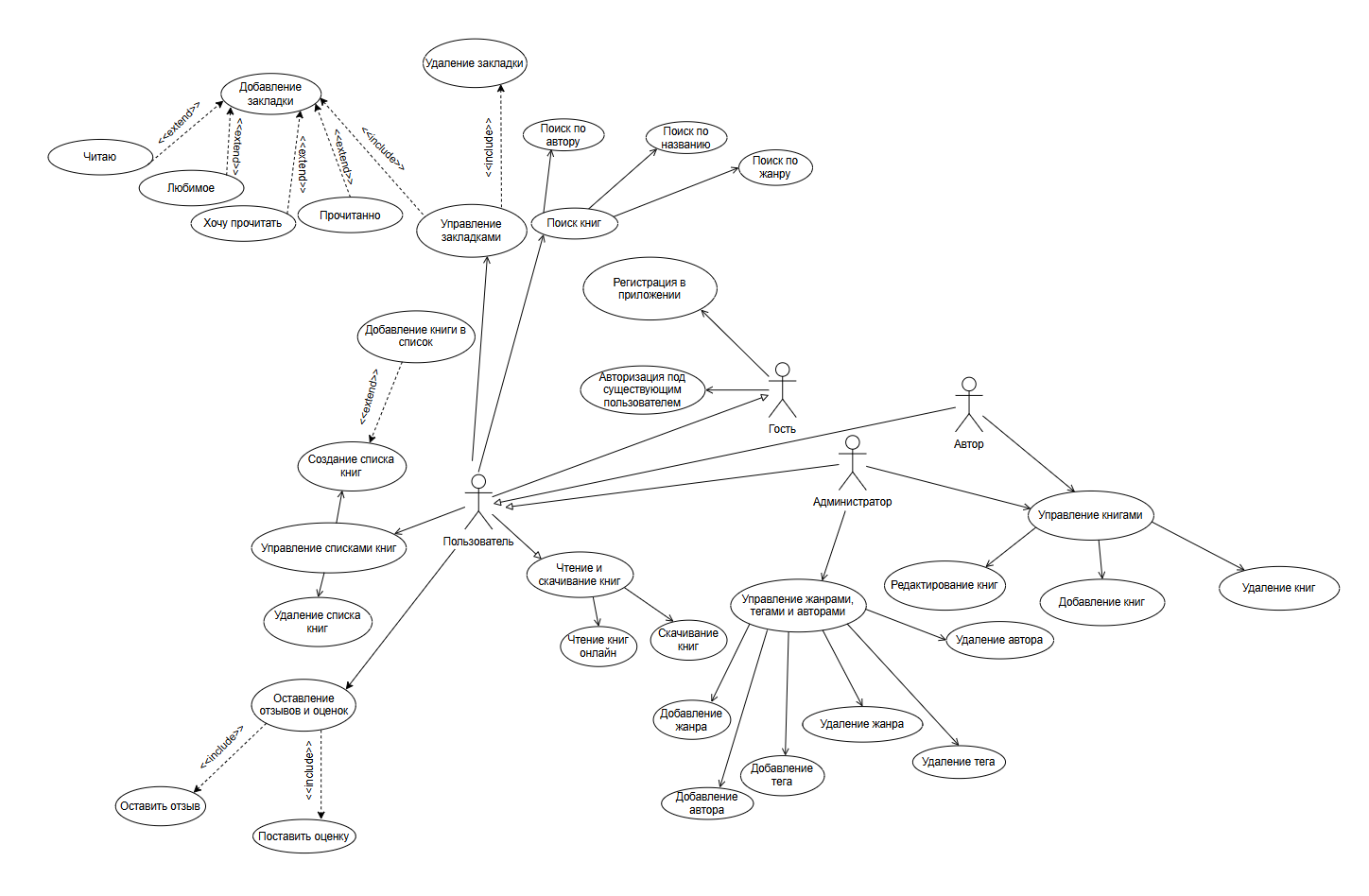


Рисунок 2.2 – Use-case диаграмма

Диаграмма подробно описывает следующие функции и взаимосвязи между ролями и действиями:

* регистрация и авторизация: гость имеет доступ к функциям регистрация в приложении и авторизация под существующим пользователем. Это позволяет новым пользователям зарегистрироваться в системе и существующим пользователям – авторизоваться;
* управление книгами: администратор и автор имеют доступ к блоку управление книгами, в который входят подфункции добавление книги, редактирование книги и удаление книги. Это позволяет администратору и автору добавлять, редактировать и удалять книги в библиотеке;
* чтение и скачивание книг: пользователь может использовать функции чтение книг онлайн и скачивание книг. Это дает возможность авторизованным пользователям читать книги прямо в приложении или загружать их для офлайн-просмотра;
* поиск книг: пользователь имеет доступ к блоку поиск книг, который включает подфункции поиск по названию, поиск по автору и поиск по жанру. Эти функции позволяют быстро находить книги в библиотеке, используя различные критерии;
* управление жанрами, тегами и авторами: администратор имеет доступ к блоку управление жанрами, тегами и авторами, включающему функции добавление жанра, удаление жанра, добавление тега, удаление тега, добавление автора и удаление автора. Это предоставляет администратору возможность настраивать категории и атрибуты для книг, облегчая пользователям поиск и сортировку;
* оставление отзывов и оценок: пользователь может использовать функции оставить отзыв и поставить оценку для книг, доступных через блок оставление отзывов и оценок. Эти действия включены с помощью <<include>>, поскольку для оставления полного отзыва и оценки оба действия должны быть выполнены;
* управление списками книг: пользователь имеет доступ к блоку управление списками книг, который включает подфункции для создания различных списков: прочитанные, хочу прочитать, избранное и читаю сейчас. Каждый из этих списков расширен функцией добавление книги в список с помощью <<extend>>, указывая, что книги могут быть добавлены в каждый список по выбору пользователя.

## **Проектирование базы данных веб-приложения**

После проведения аналогов данного проекта, была спроектирована база данных, для реализации всех поставленных задач. Данная СУБД выбрана в качестве преимущественной из-за своей простоты создания таблиц и широкой поддержки веб-приложений. MySql обладает простым и понятным языком запросов (SQL) и хорошо интегрируется с фреймворком Prisma, который используется в серверной части проекта [3].

С использованием MySql, серверная часть может эффективно управлять данными, хранить информацию о пользователях, задачах и других сущностях, необходимых для работы приложения. Создание таблиц и определение связей между ними происходит в соответствии с моделью данных, разработанной для проекта.

Схема созданного БД для данного проекта представлена на рисунке 2.3.

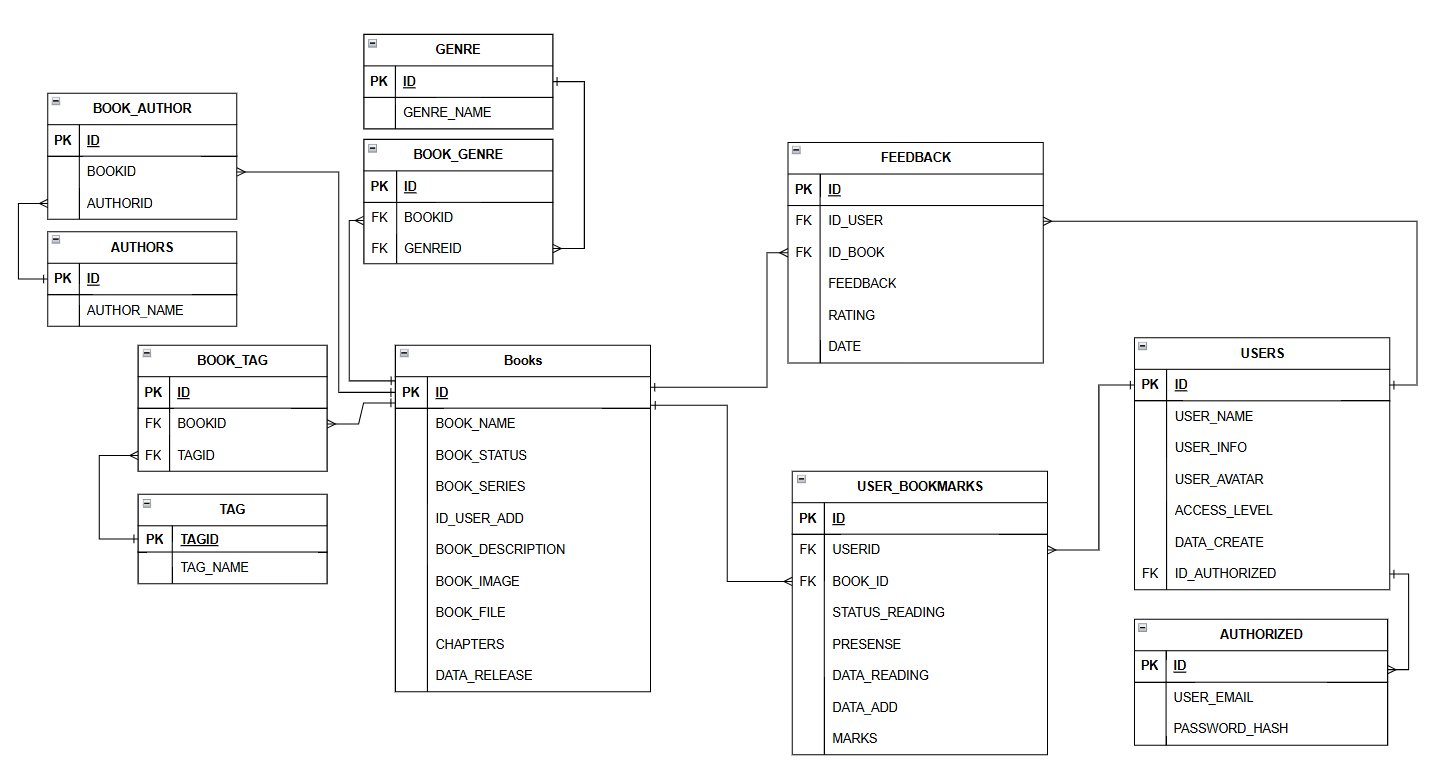


Рисунок 2.3 – Модель базы данных

Основой инфраструктуры базы данных является грамотно спроектированная модель, которая отображает связь пользовательских таблиц. Правильное и корректное взаимодействие их друг с другом как раз и заключается в схеме базы данных со связями, верно отображающими их положение.

Далее будет описана каждая таблица базы данных.

Описание структуры таблицы «users» представлено в таблице 2.1.

Таблица 2.1 – Структура таблицы «users»

|  |  |  |
| --- | --- | --- |
| Название столбца | Тип, ограничение целостности | Описание столбца |
| id | int, primary key | Идентификатор пользователя |
| EMAIL | varchar(50),unique | Электронная почта пользователя |
| pass | varchar(500) | Хеш пароля пользователя |
| ACCESS\_LEVEL | varchar(20) | Роль пользователя |
| USER\_NAME | varchar(20) | Имя пользователя |
| DATA\_CREATE | DateTime | Время создания профиля |
| USER\_INFO | Text | Описание профиля |

Таблица «users» содержит расширенную информацию о пользователях, включая их личные данные и настройки профиля. Здесь хранятся данные об уровне доступа, что позволяет системе разграничивать права пользователей и администраторов. Дополнительные столбцы, такие как аватар и описание профиля, создают основу для индивидуализации профиля, улучшая взаимодействие с пользователем. Связь с таблицей «authorized» позволяет объединить данные профиля с механизмом авторизации, что повышает удобство и безопасность.

Описание структуры таблицы «books» представлено в таблице 2.2.

Таблица 2.2 – Структура таблицы «books»

|  |  |  |
| --- | --- | --- |
| Название столбца | Тип, ограничение целостности | Описание столбца |
| ID | int, primary key | Идентификатор книги |
| BOOK\_NAME | Text | Название книги |
| BOOK\_SERIES | Text | Цикл книги |
| BOOK\_DESCRIPTION | Text | Описание книги |
| CHAPTERS | int | Количество глав |
| DATA\_RELEASE | DateTime | Дата выпуска |
| DATA\_ADD | DateTime | Дата добавления в бд |

Таблица «books» является основным хранилищем данных о каждой книге, объединяя информацию о названии, серии, статусе, описании и медиафайлах. Она также содержит информацию о количестве глав, дате выпуска и других атрибутах, что делает систему универсальным средством для управления литературными произведениями. Эта таблица является центральной частью функционала, отвечающего за добавление, редактирование и удаление книг, а также отображение их детальной информации пользователям.

Структура таблицы «authors» представлена в таблице 2.3.

Таблица 2.3 – Структура таблицы «authors»

|  |  |  |
| --- | --- | --- |
| Название столбца | Тип, ограничение целостности | Описание столбца |
| ID | Int, primary key | Идентификатор автора |
| AUTHOR\_NAME | varchar(25), unique | Имя автора |

Таблица «authors» выполняет важную роль в классификации литературных произведений, представляя уникальный список всех авторов, чьи книги хранятся в библиотеке. Эта таблица позволяет системе связывать каждую книгу с соответствующим автором, а также реализует функционал фильтрации по автору, что помогает пользователям находить книги определенного автора и углубляться в его творчество.

Описание структуры таблицы «genre» представлено в таблице 2.4.

Таблица 2.4 – Структура таблицы «genre»

|  |  |  |
| --- | --- | --- |
| Название столбца | Тип, ограничение целостности | Описание столбца |
| ID | Int, primary key | Идентификатор жанра |
| GENRE\_NAME | varchar(25), unique | Название жанра |

Таблица «genre» используется для организации книг по жанрам, создавая систему классификации для удобного поиска и сортировки. Жанры помогают структурировать библиотеку и предоставляют пользователям возможность быстро ориентироваться в ее содержимом. Эта таблица обеспечивает поддержку фильтров и категорий, что важно для улучшения пользовательского опыта и облегчения навигации.

Описание структуры таблицы «tag» представлено в таблице 2.5.

Таблица 2.5 – Структура таблицы «tag»

|  |  |  |
| --- | --- | --- |
| Название столбца | Тип, ограничение целостности | Описание столбца |
| ID | Int, primary key | Идентификатор тега |
| TAG\_NAME | varchar(25), unique | Название тега |

Таблица «tag» обеспечивает гибкость в организации библиотеки, добавляя к каждой книге ключевые слова или метки, которые описывают ее особенности. Теги предоставляют дополнительный способ для навигации.

Описание структуры таблицы представлено в таблице 2.6.

Таблица 2.6 – Структура таблицы «book\_author»

|  |  |  |
| --- | --- | --- |
| Название столбца | Тип, ограничение целостности | Описание столбца |
| ID | int, primary key | Идентификатор записи таблицы |
| BOOKID | int, foreign key | Идентификатор книги |
| AUTHORID | int, foreign key | Идентификатор автора |

Таблица «book\_author» представляет собой связь между книгами и их авторами, позволяя системе поддерживать многие-к-одному и многие-ко-многим отношения. Это особенно важно в случаях, когда одна книга может быть написана несколькими авторами, а один автор, в свою очередь, может быть причастен к написанию нескольких книг.

Описание структуры таблицы представлено в таблице 2.7.

Таблица 2.7 – Структура таблицы «book\_genre»

|  |  |  |
| --- | --- | --- |
| Название столбца | Тип, ограничение целостности | Описание столбца |
| ID | int, primary key | Идентификатор записи таблицы |
| BOOKID | int, foreign key | Идентификатор книги |
| GENREID | int, foreign key | Идентификатор жанра |

Таблица «book\_genre» является связующей таблицей между книгами и их жанрами, создавая структуру для фильтрации книг по определенным категориям. Это важно для организации библиотеки и помогает пользователям быстрее находить книги по интересующим их жанрам. Кроме того, таблица «book\_genre» помогает реализовать гибкие запросы для отображения всех книг конкретного жанра или определения всех жанров, к которым принадлежит выбранная книга.

Описание структуры таблицы представлено в таблице 2.8.

Таблица 2.8 – Структура таблицы «book\_tag»

|  |  |  |
| --- | --- | --- |
| Название столбца | Тип, ограничение целостности | Описание столбца |
| ID | int, primary key | Идентификатор записи таблицы |
| BOOKID | int, foreign key | Идентификатор книги |
| TAGID | int, foreign key | Идентификатор тега |

Таблица «book\_tag» связывает книги с тегами, что позволяет более подробно описывать каждое произведение. Это особенно полезно для тем, которые могут не попадать в стандартные жанровые категории, такие как темы книг или аудитория. Возможность добавления тегов улучшает фильтрацию и поиск, делая навигацию по библиотеке более интуитивной и удобной.

Описание структуры таблицы представлено в таблице 2.9.

Таблица 2.9 – Структура таблицы «feeadback»

|  |  |  |
| --- | --- | --- |
| Название столбца | Тип, ограничение целостности | Описание столбца |
| ID | Int, primary key | Идентификатор отзыва |
| ID\_USER | Int, foreign key | Идентификатор пользователя |
| ID\_BOOK | Int, foreign key | Идентификатор книги |
| FEEDBACK | text | Текст отзыва |
| RATING | float | Оценка |
| DATE | DateTime | Дата создания |

Таблица «feedback» хранит отзывы и оценки пользователей для каждой книги, поддерживая социальный функционал системы. Отзывы позволяют другим пользователям узнать мнение о книгах и принимать более обоснованные решения, а оценки влияют на общий рейтинг книги, что может быть полезно для рекомендаций. Это также стимулирует пользователей оставлять свои впечатления и участвовать в жизни сообщества.

Описание структуры таблицы представлено в таблице 2.10.

Таблица 2.10 – Структура таблицы «user\_bookmarks»

|  |  |  |
| --- | --- | --- |
| Название столбца | Тип, ограничение целостности | Описание столбца |
| ID | Int, primary key | Идентификатор отзыва |
| ID\_USER | Int, foreign key | Идентификатор пользователя |
| ID\_BOOK | Int, foreign key | Идентификатор книги |
| STATUS\_READING | varchar(25) | Статус чтения книги |
| DATA\_READING | DateTime | Дата последнего чтения |
| DATA\_ADD | DateTime | Дата добавления |
| MARKS | Varchar(1000) | Закладка, в книге |

Таблица «user\_bookmarks» содержит данные о закладках, созданных пользователями, и фиксирует статус прочтения книги. Это позволяет пользователям сохранять прогресс чтения и возвращаться к нужной главе. Закладки улучшают пользовательский опыт, предлагая персонализацию и напоминая пользователям, где они остановились, что делает чтение более удобным и приятным.

Таким образом при проектировании базы данных используется 11 таблиц, присутствует связь один-ко-многим и многие-ко-многим.

## **Проектирование основных разделов**

## **Алгоритм смены роли пользователя на автора**

Процесс назначения роли "Автор" пользователю в веб-приложении включает несколько последовательных этапов, обеспечивающих корректное взаимодействие между клиентской частью, сервером и базой данных.

На клиентской стороне администратор вводит адрес электронной почты пользователя через специальный интерфейс. Введенные данные проходят начальную проверку на корректность формата (валидацию). Если email не соответствует стандартному формату, клиент выводит сообщение об ошибке и процесс завершается.

После успешной валидации клиент отправляет запрос на сервер, содержащий email пользователя. Серверная часть, реализованная с использованием Express.js, обрабатывает запрос и выполняет поиск пользователя в базе данных по указанному email. База данных возвращает информацию о пользователе, если он существует. Если пользователь не найден, сервер отвечает сообщением об отсутствии пользователя, и дальнейшая обработка прекращается.

В случае успешного нахождения пользователя сервер проверяет его текущую роль. Если роль пользователя не равна "USER", сервер возвращает сообщение, что назначение роли "Автор" невозможно. Если роль соответствует требуемой, сервер изменяет запись в базе данных, обновляя роль пользователя на "AUTHOR".

После успешного обновления данных сервер отправляет ответ клиенту о завершении операции. Клиентская часть отображает соответствующее сообщение, подтверждающее, что роль пользователя была успешно изменена. Обновленные данные сохраняются в базе данных, что обеспечивает возможность дальнейшей работы пользователя с новыми правами доступа.

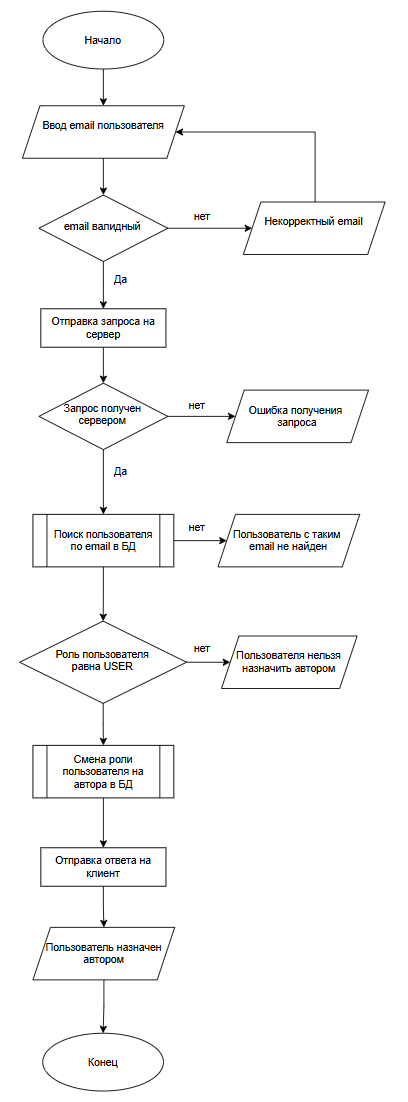


Рисунок 2.4 – Диаграмма смены роли

На рисунке 2.4 подробно показан алгоритм смены роли пользователя на автора.

## **Алгоритм работы страницы закладки книг**

Процесс работы страницы закладок в приложении начинается с того, что пользователь открывает страницу конкретной закладки. Приложение отправляет запрос на сервер с токеном аутентификации, чтобы проверить валидность токена и идентифицировать пользователя. Сервер проверяет валидность токена, после чего обращается к базе данных, чтобы определить, есть ли у этого пользователя конкретная закладка. После успешной обработки запроса сервер отправляет ответ обратно в приложение с необходимыми данными, отображая нужную закладку. В случае возникновения ошибок на любом этапе, приложение уведомляет пользователя.

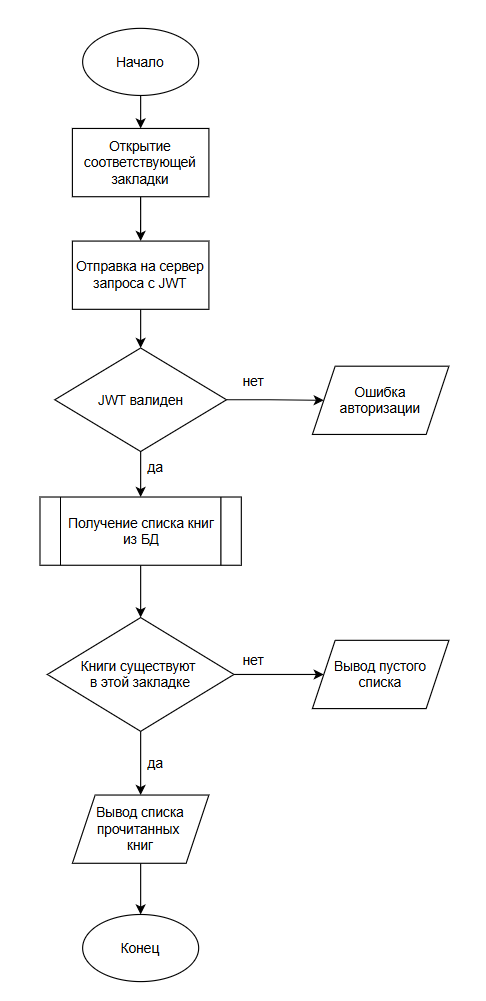


Рисунок 2.5 – Диаграмма страницы закладок

Наглядная работа алгоритма отображена на рисунке 2.5.

## **Вывод по разделу**

В результате данного проекта были разработаны основные компоненты, которые выполняют следующие функции:

Клиентское приложение: Клиентское приложение представляет собой веб-интерфейс на React Bootstrap, который обеспечивает пользовательскую интеракцию. Оно позволяет пользователям отправлять запросы на сервер, визуализировать ответы и взаимодействовать с функциональностью приложения. Клиентское приложение отображает данные, полученные от сервера, и обновляет интерфейс в соответствии с действиями пользователя.

Серверная часть: Серверная часть основана на фреймворке Express и обрабатывает запросы, поступающие от клиентского приложения. Она обеспечивает маршрутизацию запросов и взаимодействие с базой данных. Серверная часть выполняет следующие функции:

* принимает запросы от клиентского приложения через HTTPS протокол;
* маршрутизирует запросы к соответствующим обработчикам маршрутов;
* обрабатывает запросы, взаимодействуя с базой данных для создания, чтения, обновления и удаления данных;
* формирует ответы и отправляет их обратно клиентскому приложению.
* обеспечивает Web-Socket соединение.

База данных: База данных хранит информацию о пользователях, задачах и других сущностях, необходимых для работы приложения. Она обеспечивает сохранение и извлечение данных, которые используются серверной частью приложения. База данных выполняет следующие функции:

* хранит данные о пользователях, задачах и других сущностях;
* обеспечивает доступ к данным для чтения, записи, обновления и удаления;
* поддерживает целостность данных и обеспечивает безопасность доступа к данным.

Концептуально, основными компонентами проекта являются клиентское приложение, серверная часть и база данных. Клиентское приложение предоставляет пользовательский интерфейс и взаимодействует с сервером, отправляя запросы и получая ответы. Серверная часть обрабатывает запросы от клиента, взаимодействует с базой данных и возвращает результаты обратно клиентскому приложению. База данных хранит и обеспечивает доступ к данным, необходимым для функционирования приложения.

1. Разработка веб-приложения

В процессе выполнения курсового проекта было разработано веб-приложение с использованием архитектурного стиля REST API.

REST расшифровывается как Rеpresentational State Transfer. Это был термин, первоначально введен Роем Филдингом (Roy Fielding), который также был одним из создателей протокола HTTP. Отличительной особенностью сервисов REST является то, что они позволяют наилучшим образом использовать протокол HTTP.

Проект включает два отдельных компонента: server и client, ответственных соответственно за серверную и клиентскую части программного средства.

## **Разработка серверной части**

Серверная часть делиться на 3 основных блока: Controller, Model, Route и другие сопутствующие им блоки по типу static, public, prisma. А так же блоки обработки ошибок error и middleware. Более наглядно структура показана на рисунке 3.1.

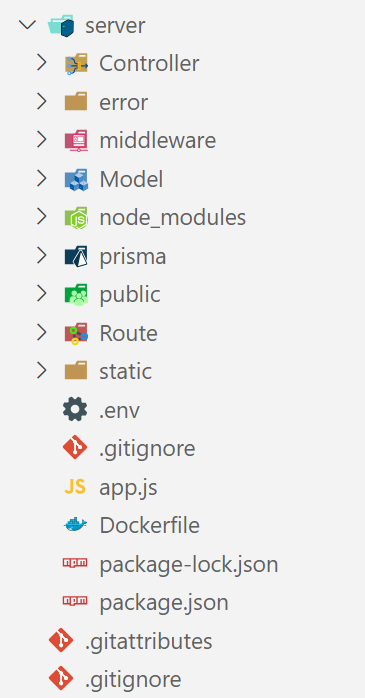


Рисунок 3.1 – Структура серверной части проекта

Входной точкой в приложение является файл app.js, в котором осуществляется запуск сервера для прослушивания входящих запросов, поступающих на определенный порт. В этом файле происходит подключение необходимых модулей для работы сервера, а также выполнение начальной инициализации приложения и регистрация маршрутизаторов.

app.js является основным файлом, который инициирует работу веб-сервера. В нем происходит создание экземпляра сервера с использованием фреймворка Express.js. Затем осуществляется настройка сервера, включая установку порта для прослушивания входящих запросов.

Далее происходит подключение необходимых модулей, которые используются в процессе работы сервера. Это может включать модули для работы с базой данных, обработки маршрутов, аутентификации и авторизации пользователей, обработки ошибок и другие.

После подключения модулей происходит инициализация приложения, включая настройку и конфигурацию необходимых компонентов и настройка маршрутов.

В результате выполнения кода в app.js сервер становится готовым к принятию входящих запросов и обработке их в соответствии с заданными маршрутами и логикой приложения.

Для удобной реализации серверной части приложения был выбран фреймворк Express.js.

Модели представляют собой компоненты, отвечающие за взаимодействие с базой данных и представление данных в виде объектов. Они содержат определения схемы данных и методы для выполнения операций чтения, записи и обновления данных в базе. Рассмотрим один из методов модели book, getBooks – представляет собой метод принимающий номер страницы, параметры фильтрации и поиска. Данный метод связывается с prisma.client обращаясь к таблице books. В результате запроса к бд возвращает указанно страницу из списка отфильтрованный данных. Такой запрос можно увидеть в листинге 3.1.

|  |
| --- |
| const books = await prisma.books.findMany({              where,              skip: skip(page),              take: take,              include: {                  book\_tag: {include: { tag: true },},                  book\_genre: {include: { genre: true },},                  book\_author: {include: { authors: true },},              },          }); |

Листинг 3.1 – Запрос метода getBooks

Контроллеры (controllers) являются частями Express.js, где реализуются функции для формирования ответов на запросы клиента. Они обрабатывают входящие запросы, взаимодействуют с моделями и формируют нужные данные для отправки клиенту. Контроллеры являются промежуточными слоями между маршрутами и моделями, отвечая за логику обработки запросов.

Роуты (routes) определяют пути URL и связанные с ними обработчики контроллеров. Они определяют, какой контроллер будет обрабатывать определенный тип запроса (GET, POST, PUT, DELETE) на определенном пути. Роуты могут также содержать параметры и динамические сегменты URL, чтобы обрабатывать различные варианты запросов. Основной роут indexRout.js который распределяет запросы по разным роутам представлен на листинге 3.2.

|  |
| --- |
| const Route = require('express').Router();  Route.use('/users', UserRoute);  Route.use('/books', BookRoute);  Route.use('/feedback', FeedbackRoute);  Route.use('/bookmark', BookmarkRoute);  Route.use('/files', fileRoute);  Route.use('/tag', tagRoute);  Route.use('/genre', genreRoute);  Route.use('/author', authorRoute);  module.exports = Route; |

Листинг 3.2 – Файл indexRoute.js

Файлы для обработки ошибок представляет из себя пользовательский класс ApiError, код которого можно увидеть на листинге 3.3.

|  |
| --- |
| class ApiError extends Error {    constructor(status, message) {      super();  this.status = status;  this.message = message;    }  static badRequest(message) {      return new ApiError(404, message);    }  static unauthorized(message) {    return new ApiError(401, message);    }    static forbidden(message) {      return new ApiError(403, message);    }  static internal(message) {      return new ApiError(500, message);    }}  module.exports = ApiError; |

Листинг 3.3 – Файл ApiError.js

Блок middleware представляет собой обработку ошибок и так же реализованы проверка авторизован ли пользователь и проверка на соответствие роли.

|  |
| --- |
| module.exports = function (req, res, next) {      if (req.method === 'OPTIONS') {         return next();}      try {          const token = req.headers.authorization.split(' ')[1];          if (!token) {              return next(ApiError.unauthorized('Нe авторизован(Не передан токен)'));          }          const decoded = verify(token, process.env.SECRET\_KEY);          req.user = decoded;          return next();      }      catch (e) {          return next(ApiError.unauthorized('Не авторизован(Непредвиденная ошибка)'+e.message));      }  } |

Листинг 3.3 – файл authMiddleware.js

Данный код достает из заголовка запроса токен и декодирует его, используя секретный ключ. Если же токен не был получен или не смог декодироваться, то возникает ошибка.

В данном примере видно, что для регистрации, авторизации и аутентификации используется jwt-авторизация. Это довольно простой и защищенный способ авторизации пользователей.

Так же на сервере реализован web-socket. Он используется для мгновенного отображение отзывов на странице книги. Это возможно с помощью модуля socket.io.

Код реализации web-socket предоставлен на листинге 3.4.

|  |
| --- |
| const io = require("socket.io")(http, {      cors: {origin: "http://localhost:3000",methods: ["GET", "POST"]}    });  io.on('connection', (socket) => {      socket.on('disconnect', () => {console.log('user disconnected');      });      socket.on('message', (msg) => {          feedback.addFeedback(msg.id, msg.id\_Book, msg.raiting, msg.text).then((res) => {              io.emit('message', res);          });      });  }); |

Листинг 3.4 – реализация web-socket на сервере

В данном коде сокет прослушивает два состояния «message» и «disconnect». Сервер принимает сообщения и отправляет его всем клиентам в сети.

## **Разработка клиентской части**

Клиентская часть разработана на react, что имеет собственную структура для реализации react api. Поэтому мы будем рассматривать лишь структуру папки «srс». Его структура предоставлена на рисунке 3.2.

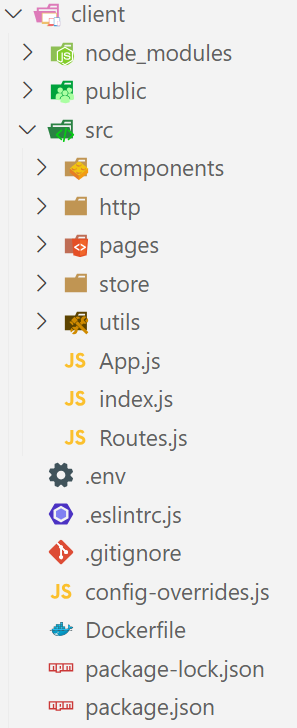


Рисунок 3.2 – Структура клиентской части проекта

Директория «src» содержит исходный код клиентской части приложения. Внутри нее располагаются следующие файлы и папки:

* components: эта папка содержит компоненты, которые используются для создания пользовательского интерфейса. Компоненты представляют собой переиспользуемые блоки кода, которые описывают отдельные элементы интерфейса или части страницы. Примеры компонентов могут включать формы, таблицы, кнопки и другие интерактивные элементы.
* pages: здесь находятся компоненты, которые представляют отдельные страницы или экраны приложения. Каждая страница обычно состоит из нескольких компонентов и определяет структуру и содержимое конкретной части интерфейса;
* http: В этой папке содержатся службы (services), которые обеспечивают взаимодействие клиентской части приложения с сервером. Сервисы выполняют функции отправки HTTP-запросов на сервер, обработки ответов и управления данными. Например, здесь может быть сервис для работы с API сервера, сервис для работы с авторизацией и другие;
* utils: в данной папке содержатся вспомогательные утилиты и функции, которые могут использоваться в разных частях клиентского приложения. Это могут быть функции для обработки данных, форматирования или других задач, которые упрощают разработку и повторное использование кода;
* store: здесь находятся файлы реализующие некоторые классы. По типу UserStore и BookStore. Которые служат для передачи данных между компонентами в Context;
* App.js и index.js: эти файлы являются основной точкой входа в клиентское приложение. App.js определяет основную структуру и маршрутизацию приложения, включая вложенные компоненты и их расположение. index.js отвечает за инициализацию приложения и его рендеринг в корневой элемент HTML-страницы.

В клиентской части приложения используется библиотека React Bootstrap, которая предоставляет набор предопределенных компонентов для создания пользовательского интерфейса на основе React. React Bootstrap комбинирует возможности React и стилей.

React Bootstrap предлагает множество других компонентов, таких как навигационное меню (Navbar), формы (Form), модальные окна (Modal) и многое другое. Вы можете ознакомиться с полным набором доступных компонентов и их возможностями в документации на официальном сайте React Bootstrap. Пример кода для импорта и использования компонента Navbar из React предоставлен в приложении А. В этом примере мы импортируем компоненты Navbar и Nav из библиотеки React Bootstrap и используем их для создания навигационного меню.

Так же для чтения файлов «.epub» был использован модуль ReactReader. React Reader — это реактивная оболочка для epub.js , использующая ветку v.03. epub.js — отличная библиотека, и это оболочка для нее. Эта оболочка упрощает использование в React-приложении.

Также стоит обратить внимание, что EpubJS — это браузерный epub-reader и он работает, отображая текущую epub-главу в iframe, а затем с помощью css-колонок отображает текущую страницу.

Пример части кода компонента для чтения книг представлен на листинге 3.5.

|  |
| --- |
| <Card>        {alertshow && <Alert variant={variant} className="mt-3" onClose={() => setAlertshow(false)} dismissible>{alertText}</Alert>}        <div style={{ height: '100vh' }}>          <ReactReader locationChanged={locationChanged}            {...location && { location }}            url={process.env.REACT\_APP\_API\_URL + 'Book' + data + '.epub'}            getRendition={rendition => (renditionRef.current = rendition)}            tocChanged={toc => (tocRef.current = toc)}          />        </div>      </Card> |

Листинг 3.5 – Реализация ReactReader

В данном примере представлен код вывода на экран книги с возможностью перелистывать страницы и выбора глав.

Так же и на клиенте, как на сервере, реализован web-socket. Часть реализации представлен на листинге 3.6.

|  |
| --- |
| const socket = io(process.env.REACT\_APP\_API\_URL);   useEffect(() => {          try {              socket.on('message', (msg) => {                  console.log(msg);                  setMessages([...messages, msg]);                  console.log(messages);              });          } catch (e) {              setAlertshow(true);              setVariant('danger');              setAlertText('Произошла ошибка: ' + e.response.data.message);          }      }, [messages]); |

Листинг 3.6 – Реализация web-socket на клиенте

В данном коде клиент отправляет отзыв на сервер и получает сообщения от других пользователей.

## **Вывод по разделу**

В данном разделе описаны этапы разработки веб-приложения с использованием REST API, фреймворка Express.js для серверной части и React для клиентской.

Серверная часть включает модели, контроллеры, маршруты, middleware и обработчики ошибок, что обеспечивает безопасное взаимодействие с базой данных через Prisma. Реализованы функции загрузки файлов, JWT-авторизация и поддержка web-socket для обновления данных в реальном времени.

Клиентская часть построена на React с использованием компонентов, страниц, сервисов и хранилища состояния. Для интерфейса применена библиотека React Bootstrap, а для чтения файлов .epub – ReactReader. Также реализована поддержка web-socket для синхронизации отзывов. Веб-приложение обладает современной архитектурой, удобной для использования и дальнейшего развития.

1. Тестирование веб-приложения

## **Тестирование под видом клиента**

Тестирование проекта является важной частью разработки программного обеспечения, поскольку оно помогает обнаруживать ошибки и уверенно утверждать, что система работает корректно. Для тестирования проекта можно использовать различные подходы и инструменты.

Первым на что может наткнуться пользователь окно входа и регистрации. В данном окне могут возникнуть ошибку входа пример такой ошибки представлено на рисунке 4.1.

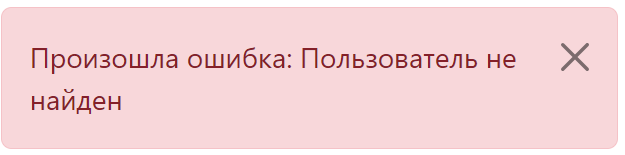


Рисунок 4.1 – Ошибка поиска пользователя в системе

Кроме этого поля также проверяются на корректное заполнение. Например, при входе, почта и пароль должны быть длиной как минимум 3 символа. А почта должна быть валидной. Случай такой ошибки представлен на рисунке 4.3.

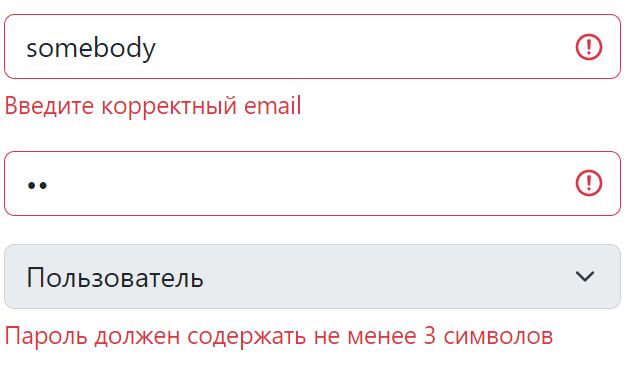


Рисунок 4.2 – Валидация полей ввода данных для входа

В данном случае, валидация осуществляется с использованием простых проверок в функциях validateEmail и validatePassword, которые вызываются при потере фокуса (событие onBlur) полей ввода.

Ошибки могут возникать так же и в панели администратора, например при добавлении уже существующего тега. Пример представлен на рисунке 4.3.

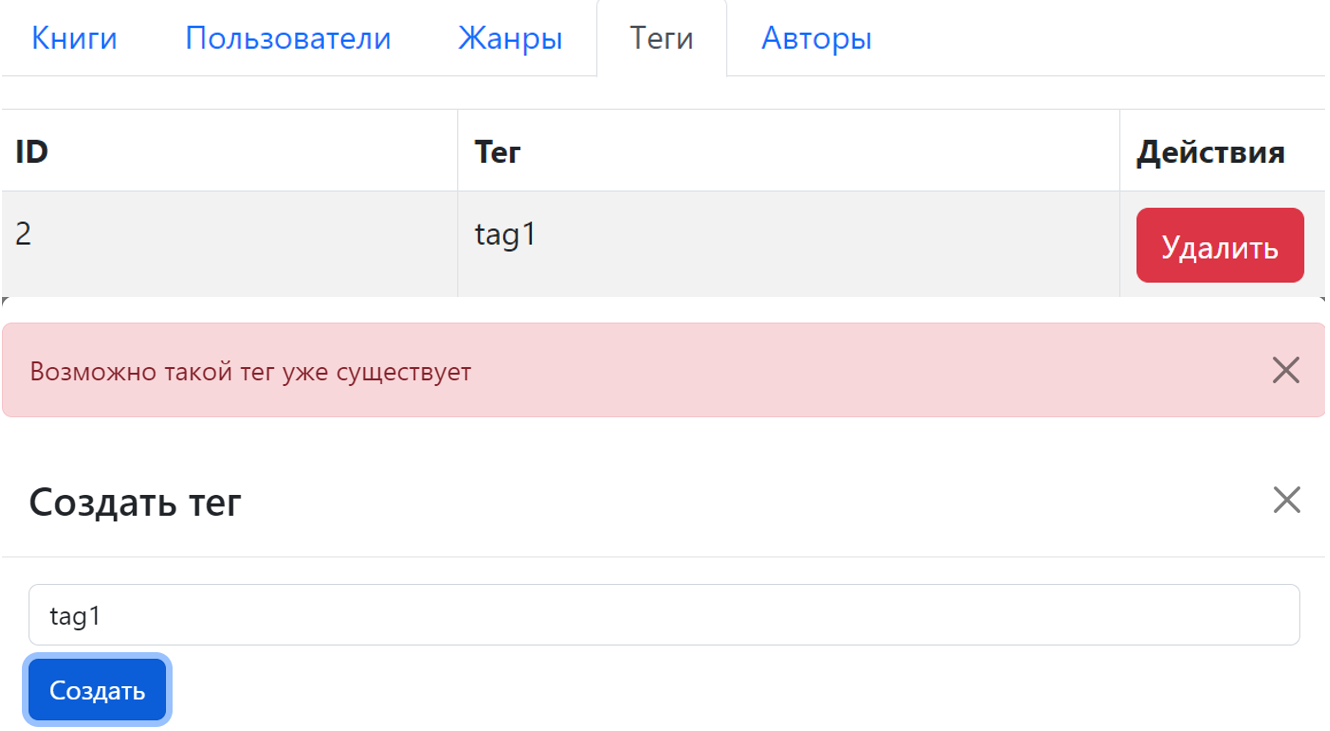


Рисунок 4.3 – Ошибка создания повторяющегося тега

Таким образом, не позволяется заполнить базу данных одинаковыми данными.

На клиенте так же могут возникнуть и другие непредвиденные ошибки. Но если пользователь попытается попасть на неразрешенный ему ресурс то, клиент просто вернет его на главную.

Демонстрация мобильного вида страницы книги предоставлена на рисунке 4.4.

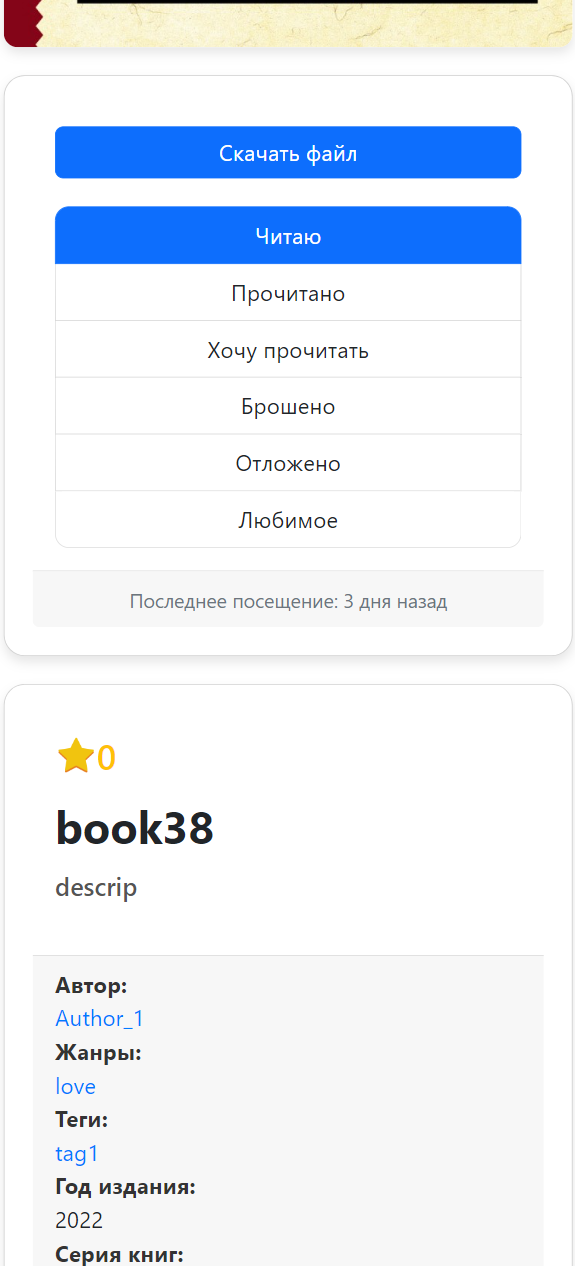


Рисунок 4.4 – Страница книги

При просмотре информации о книге также есть возможность увидеть другие книги с такими тегами или жанрами, а также с таким автором.

## **Вывод по разделу**

В результате тестирования приложения было обнаружено, что компоненты корректно отрисованы и взаимодействуют с пользовательскими действиями. Также было успешно протестировано создание закладок чтения книги и отображение информации о текущей странице. В целом, приложение прошло основные тестовые сценарии и работает стабильно.

1. Руководство пользователя

## **Регистрация пользователя**

Для создания аккаунта нужно следовать следующим шагам:

* перейти на главную страницу приложения.
* нажать на кнопку "зарегистрироваться".
* заполнить необходимые поля в форме регистрации, включая, адрес электронной почты и пароль.
* подтвердить правильность введенной информации.
* нажать кнопку "зарегистрироваться" для создания аккаунта.
* если все данные введены правильно, пользователь будет перенаправлен на страницу входа в систему.

Форма для регистрации предоставлена на рисунке 5.1.

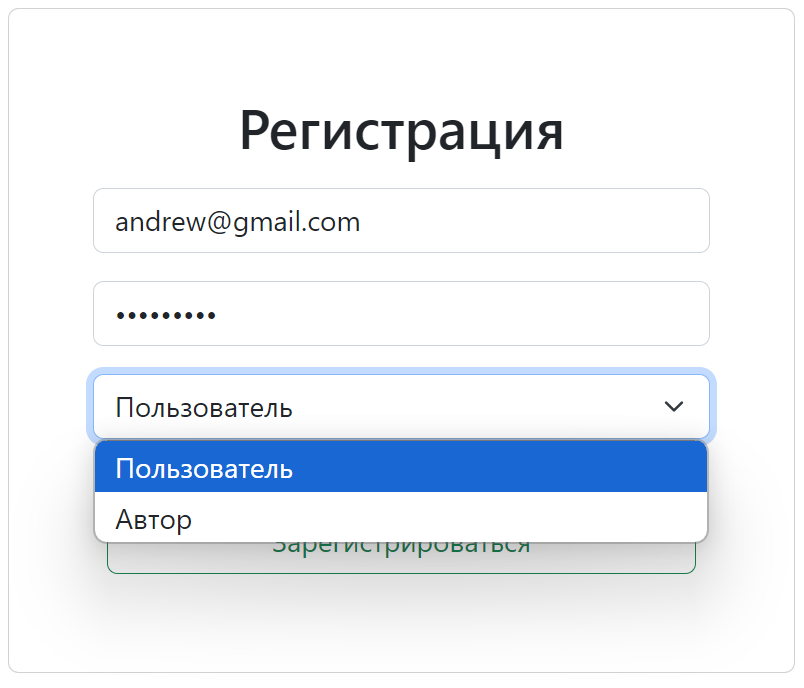


Рисунок 5.1 – Окно регистрации пользователя

Форма имеет валидацию, поэтому нужно соблюдать пару правил: почта, должна быть корректной, а пароль – не менее 3 символов. Если не соблюдать валидацию, то зарегистрироваться не получиться. Так же надо помнить, что на одну почту может быть создан лишь один аккаунт. При регистрации, в localStorage сохраняются данные зарегистрированного пользователя.

## **Аутентификация и авторизация пользователя**

Для входа в систему необходимо ввести свою электронную почту, пароль и нажать на кнопку "Войти". Это позволит системе проверить наличие пользователя с указанными данными. Все поля обязательны для заполнения.

Если аутентификация прошла успешно и были соблюдены все вышеуказанные правила, пользователь будет перенаправлен на главную страницу, что означает успешную авторизацию. На рисунке 5.2 представлен скриншот формы авторизации.



Рисунок 5.2 – Окно авторизации пользователя

После успешного входа пользователь получит доступ к большей части функций, и так же к редактированию собственного профиля.

Для использования некоторых функций приложения необходимо пройти процесс авторизации. Если пользователь не авторизован, он не сможет получить доступ к определенным страницам и функциям приложения. Для получения доступа к полному функционалу необходимо пройти процесс авторизации.

## **Главная страница**

При успешном входе в аккаунт пользователь попадает на главную страницу. На этой странице можно совершить поиск и фильтрацию по различным параметрам, таким как жанры, теги, авторы, серии, названия. Так же на странице виден рейтинг книг, который вычисляется по отзывам других пользователей.

Данная страница предоставлена на рисунке 5.3.

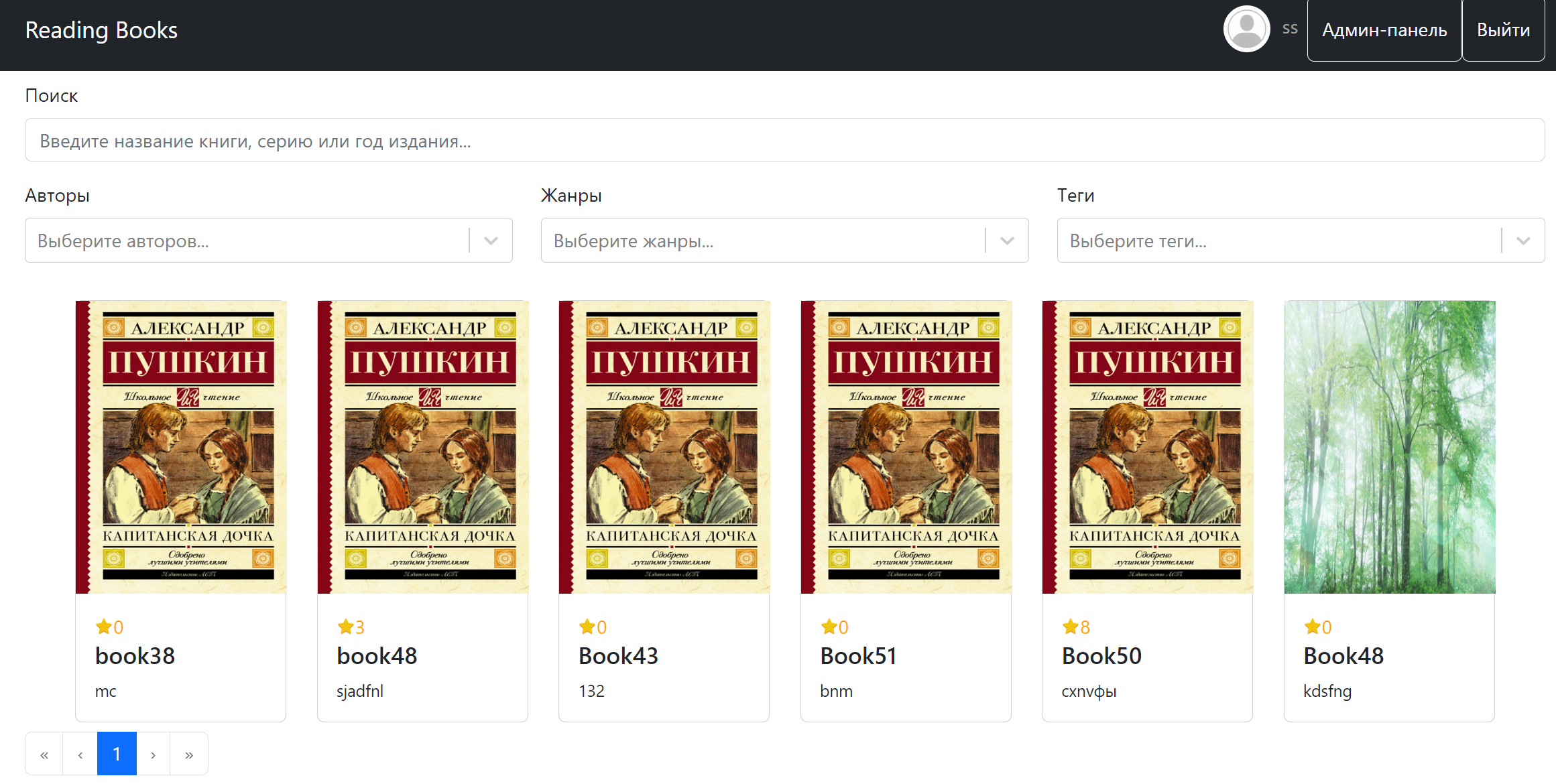


Рисунок 5.3 – Главная страница приложения

На данной странице можно перейти к чтению книг или просмотреть собственные списки, перейдя в профиль пользователя.

## **Страница книги**

На данной странице можно узнать более подробно информацию о книге, прочитать её. Страница представлена на рисунке 5.4

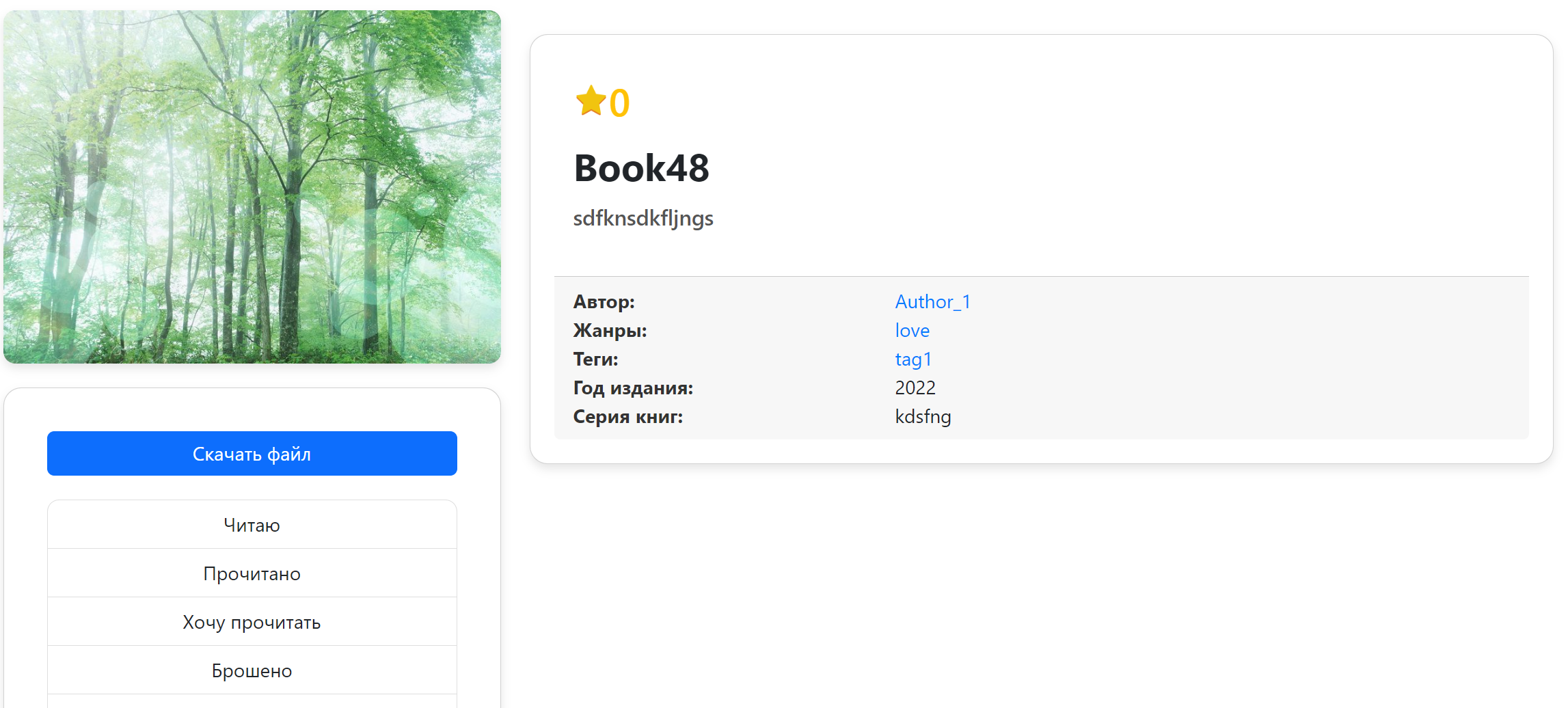


Рисунок 5.4 – Страница книги

Кроме того, на странице можно оставить отзыв с оценкой на книгу.

## **Личная страница пользователя**

Нажав на фото или имя профиля, можно попасть на страницу пользователя. Эта страница представлена на рисунке 5.5.

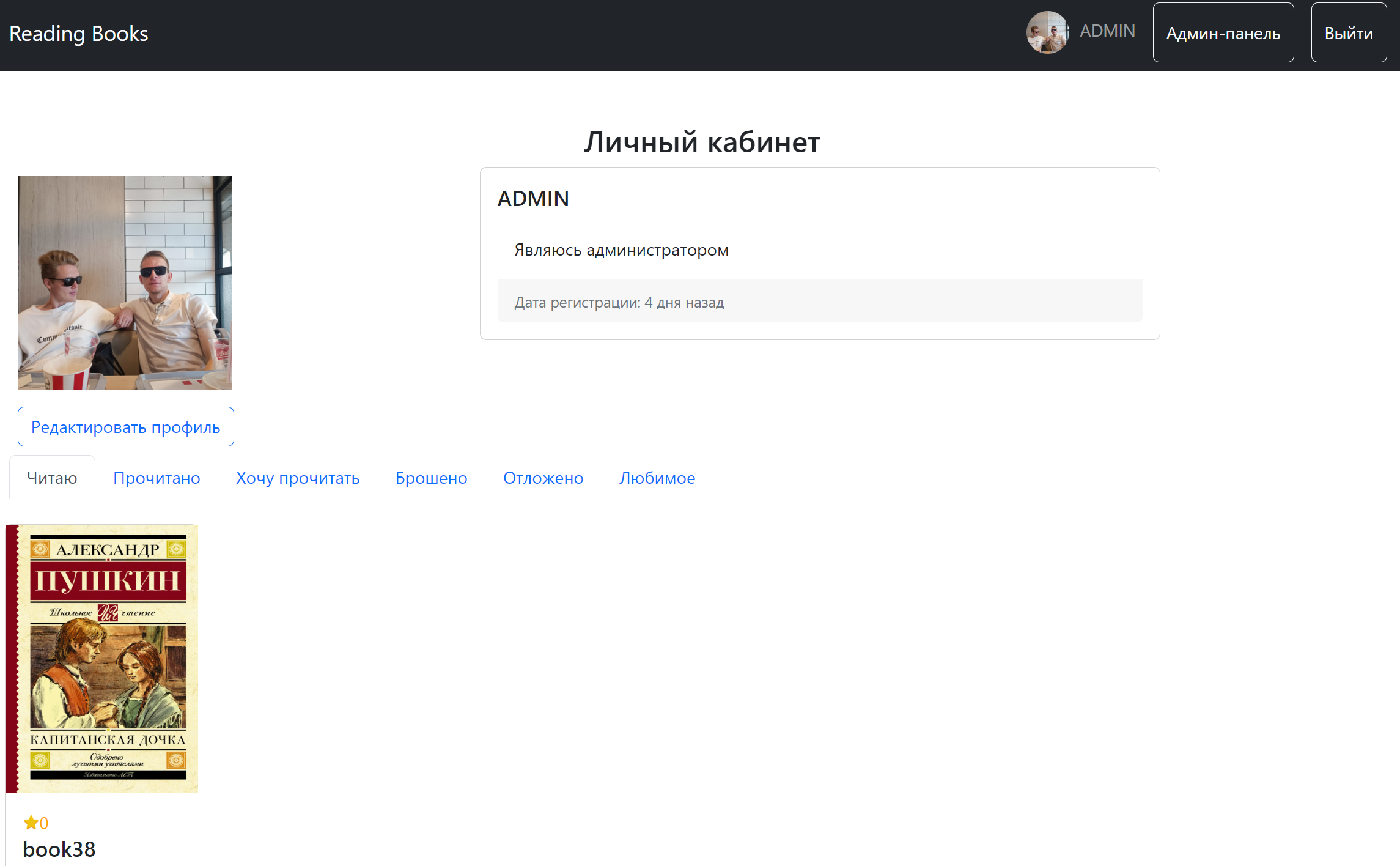


Рисунок 5.5 – Страница пользователя

На этой странице представлена основная информация о пользователе, а также можно изменить информацию о пользователе, или загрузить картинку.

## **Панель администратора**

Конечно же была разработана и возможность администрирования. Страница панели администратора представлена на рисунке 5.6.

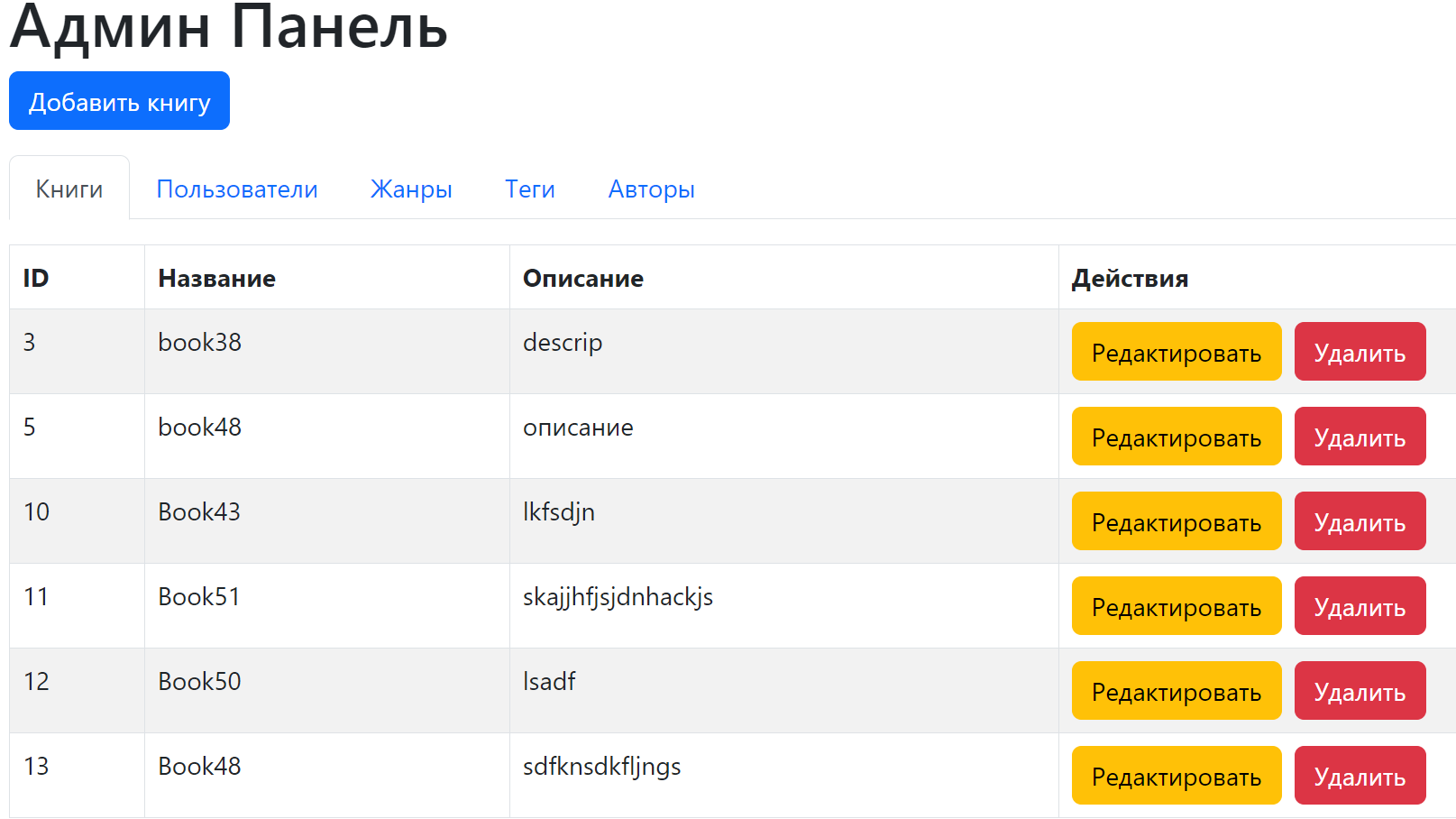


Рисунок 5.6 – Админ-панель

Администратор может добавлять и удалять жанры, авторов, теги, книги пользователей. Так же можно изменить книги. Страница имеет переключения по старицам таблиц. На рисунке 5.7 представлена форма назначения пользователя автором.

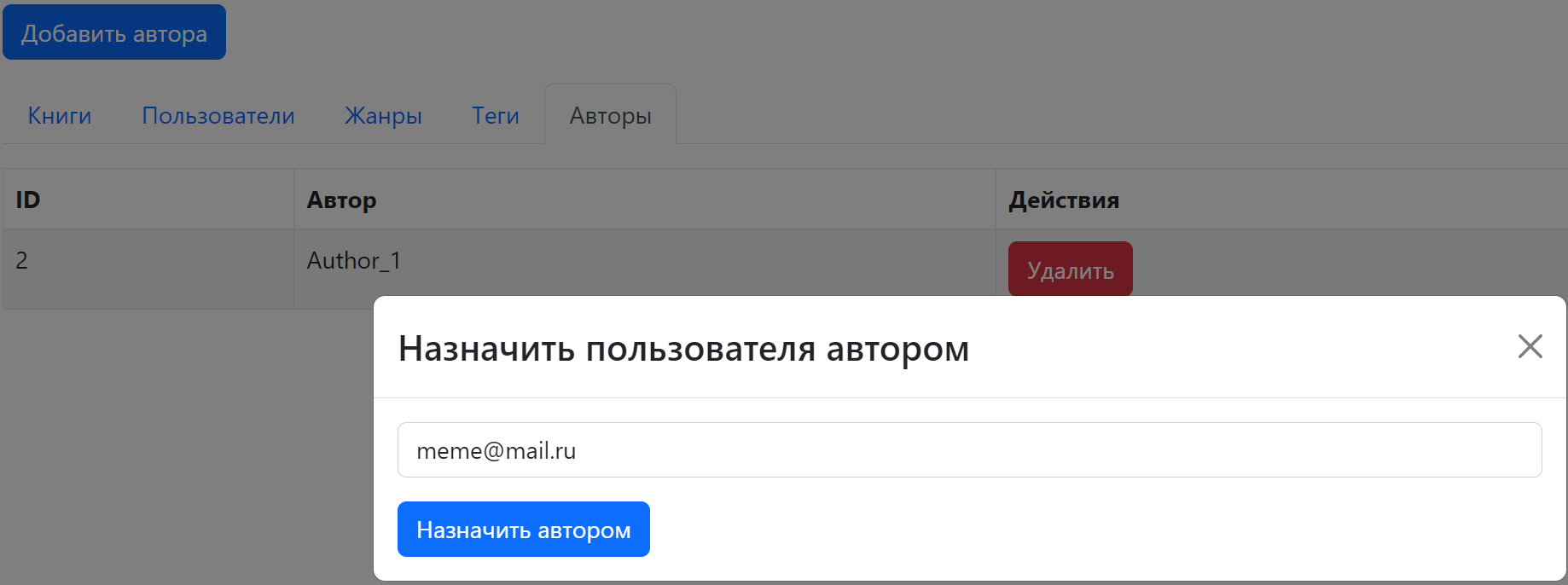


Рисунок 5.7 – Назначение автора

Для изменения книги осуществляется переход в форму редактирования книги. Этот функционал показан на странице 5.8

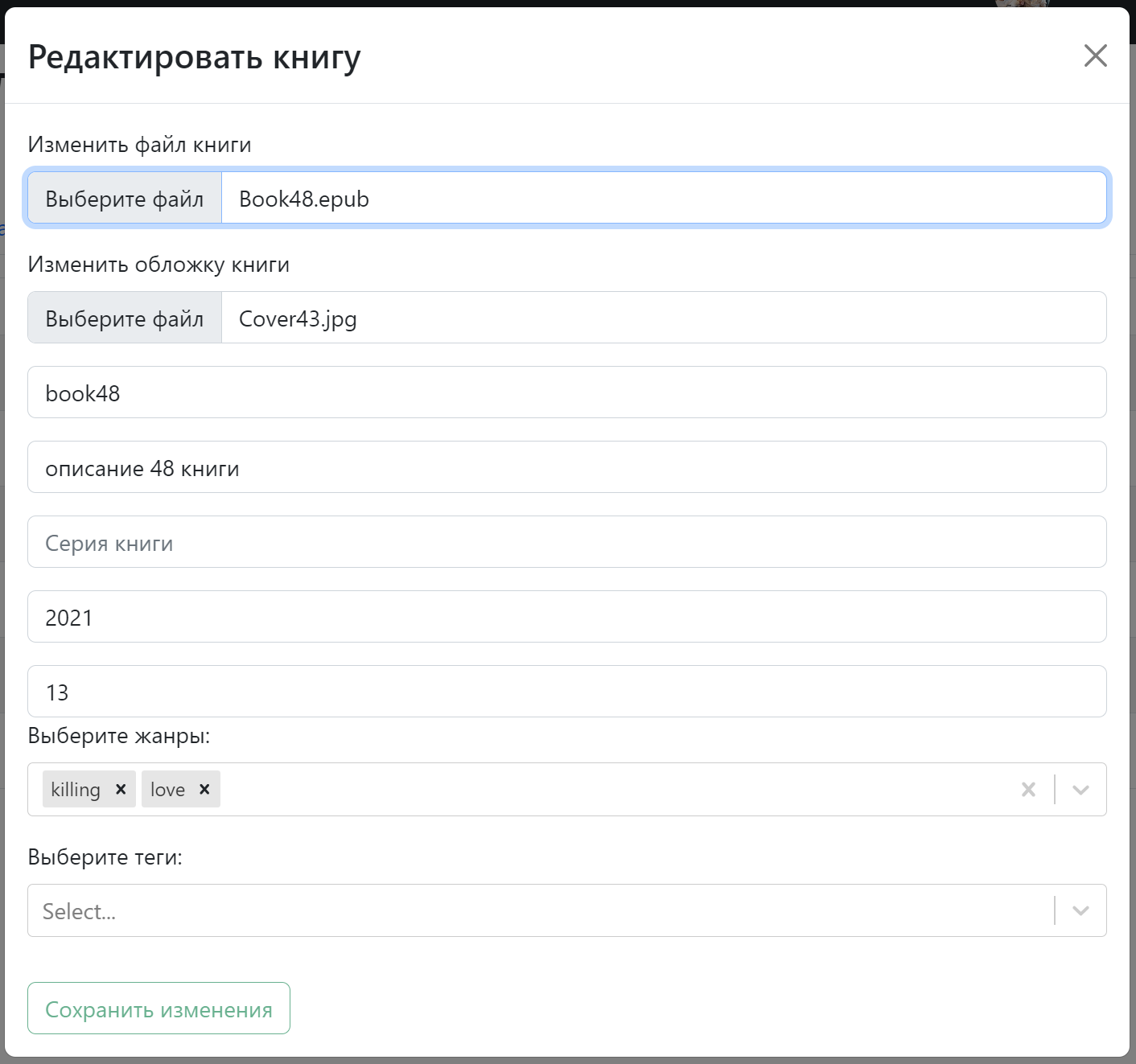


Рисунок 5.8 – Редактирования информации о книге

## **Удаление комментариев**

Конкретно администратору доступно удаление комментариев книги. Кнопка удаление представлена на рисунке 5.9.

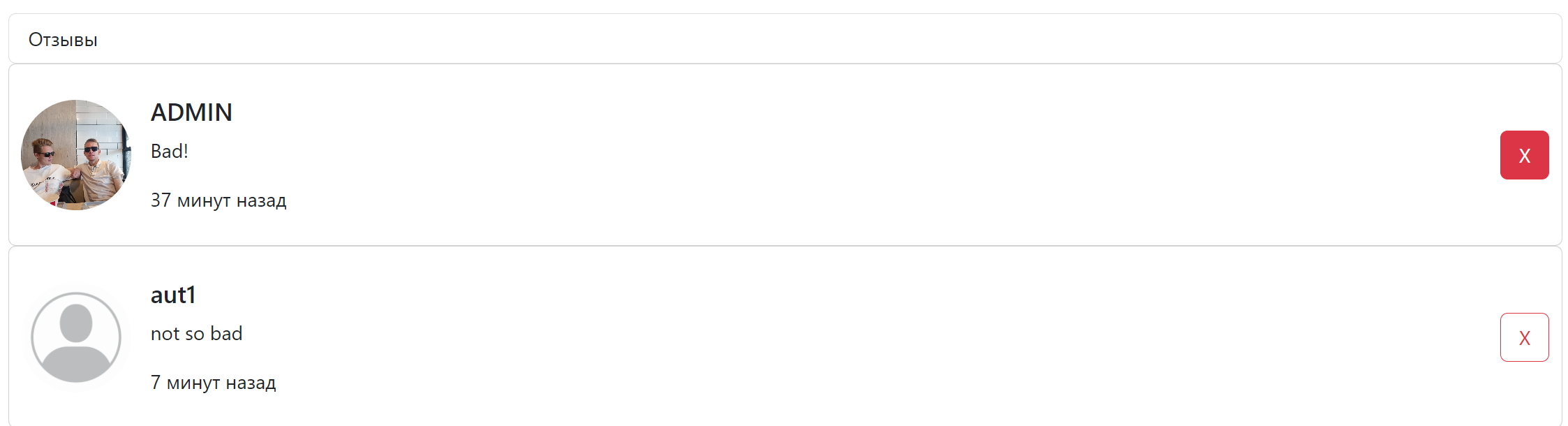


Рисунок 5.9 – Удаление комментариев

## **Вывод по разделу**

В результате разработки веб-приложения были реализованы основные функции для работы с книгами, включая регистрацию и авторизацию пользователей, доступ к функционалу через главную страницу с возможностью поиска и фильтрации, просмотр информации о книгах, чтение и оставление отзывов. Пользователи могут управлять своими профилями, изменяя данные или загружая изображения. Для администраторов предусмотрена панель управления с возможностью добавления, изменения и удаления жанров, авторов, тегов, книг и комментариев, а также назначения пользователей авторами. Все страницы интерфейса имеют удобную навигацию и встроенную валидацию, обеспечивая корректность ввода данных и доступ к функционалу в соответствии с ролью пользователя.

Заключение

В процессе решения поставленной задачи была достигнута поставленная цель. Функциональное веб-приложение, разработанное с использованием технологий React, Bootstrap, Express, и Mysql представляет собой полноценную библиотеку для чтения и управления информацией о книгах. Оно обладает следующими основными возможностями:

* регистрация и авторизация: пользователи могут создавать учетные записи и входить в систему для доступа к функциональности приложения. Это обеспечивает безопасность и персонализированный опыт использования;
* роли администратора, автора и пользователя: приложение поддерживает три роли: администратора, автора и пользователя. Администратор имеет расширенные привилегии, такие как добавление и редактирование книг, управление пользователями и отзывами. Пользователи могут просматривать информацию о книгах, оставлять отзывы и устанавливать оценки. Авторы имеют функционал пользователей и могут добавлять свои книги;
* добавление и редактирование книг: администратор и автор могут добавлять новые книги в библиотеку, предоставляя информацию о них, такую как название, автор, жанр и другие характеристики. Также имеется возможность редактировать информацию о существующих книгах;
* чтение и скачивание книг: пользователи могут читать книги онлайн и скачивать их для офлайн-чтения. Приложение обеспечивает удобный интерфейс для чтения и предоставляет возможность загрузки книг на устройства пользователей;
* поиск книг: в приложении реализован поиск книг по различным критериям, включая название, автор, жанр и другие параметры. Это позволяет пользователям быстро находить интересующие их книги;
* информация о книгах: пользователи могут просматривать подробную информацию о каждой книге, включая обложку, описание, рейтинги, отзывы и другие характеристики. Это помогает им принимать решение о прочтении книги;
* отзывы и оценки: пользователи могут оставлять отзывы на книги и ставить им оценки. Это позволяет пользователям делиться своим мнением о книгах и помогает другим пользователям принять решение о прочтении;
* личные списки: пользователи могут создавать личные списки прочитанных книг, книг, которые они хотят прочитать, а также список своих любимых книг. Это помогает пользователям организовать свою библиотеку и делиться рекомендациями с другими.

Таким образом, функциональное веб-приложение предоставляет все необходимые возможности для удобного управления библиотекой книг. Оно сочетает в себе современные технологии разработки с понятным и интуитивным пользовательским интерфейсом, что делает его привлекательным и удобным в использовании для пользователей.

# Список используемых источников

1. Документация на фреймворк Express [Электронный ресурс] / Режим доступа: https://expressjs.com – Дата доступа: 05.11.2024.
2. Документация на ORM Prisma [Электронный ресурс] / Режим доступа: https://prisma.io/docs.com – Дата доступа: 02.11.2024.
3. Документация на БД MySQL [Электронный ресурс] / Режим доступа: <https://dev.mysql.com/doc> – Дата доступа: 07.10.2024.
4. Документация на библиотеку React [Электронный ресурс] / Режим доступа: https://react.dev – Дата доступа: 02.11.2024.
5. Документация на платформу контейнеризации Docker [Электронный ресурс] / Режим доступа: https://docs.docker.com – Дата доступа: 15.11.2024.
6. Документация на библиотеку React Bootstrap [Электронный ресурс] / Режим доступа: https://react-bootstrap.netlify.app – Дата доступа: 03.11.2024.

# ПРИЛОЖЕНИЕ А

Листинг prisma-схемы базы данных

|  |
| --- |
| generator client {    provider = "prisma-client-js"    binaryTargets = ["native", "debian-openssl-1.1.x"]  }  datasource db {    provider = "mysql"    url      = env("DATABASE\_URL")  }  model authors {    ID          Int           @id @default(autoincrement())    AUTHOR\_NAME String        @unique(map: "AUTHOR\_NAME") @db.VarChar(25)    ID\_USER     Int           @unique    user        users         @relation(fields: [ID\_USER], references: [ID])    book\_author book\_author[]  }  model book\_author {    BOOKID   Int    AUTHORID Int    ID       Int     @id @default(autoincrement())    books    books   @relation(fields: [BOOKID], references: [ID], onDelete: NoAction, onUpdate: NoAction, map: "book\_author\_ibfk\_1")    authors  authors @relation(fields: [AUTHORID], references: [ID], onDelete: NoAction, onUpdate: NoAction, map: "book\_author\_ibfk\_2")    @@index([AUTHORID], map: "AUTHORID")    @@index([BOOKID], map: "BOOKID")  }  model book\_genre {    BOOKID  Int    GENREID Int    ID      Int   @id @default(autoincrement())    books   books @relation(fields: [BOOKID], references: [ID], onDelete: NoAction, onUpdate: NoAction, map: "book\_genre\_ibfk\_1")    genre   genre @relation(fields: [GENREID], references: [ID], onDelete: NoAction, onUpdate: NoAction, map: "book\_genre\_ibfk\_2")    @@index([BOOKID], map: "BOOKID")    @@index([GENREID], map: "GENREID")  }  model book\_tag {    BOOKID Int    TAGID  Int    ID     Int   @id @default(autoincrement())    books  books @relation(fields: [BOOKID], references: [ID], onDelete: NoAction, onUpdate: NoAction, map: "book\_tag\_ibfk\_1")    tag    tag   @relation(fields: [TAGID], references: [ID], onDelete: NoAction, onUpdate: NoAction, map: "book\_tag\_ibfk\_2")    @@index([BOOKID], map: "BOOKID")    @@index([TAGID], map: "TAGID")  }  model books {    ID               Int              @id @default(autoincrement())    BOOK\_NAME        String           @db.Text    BOOK\_SERIES      String?          @db.Text    BOOK\_DESCRIPTION String?          @db.Text    CHAPTERS         Int?    DATA\_RELEASE     Int?    DATA\_ADD         DateTime         @default(now()) @db.DateTime(0)    book\_author      book\_author[]    book\_genre       book\_genre[]    book\_tag         book\_tag[]    feedback         feedback[]    user\_bookmarks   user\_bookmarks[]  }  model feedback {    ID       Int      @id @default(autoincrement())    ID\_USER  Int    ID\_BOOK  Int    FEEDBACK String?  @db.Text    RATING   Float    @db.Float    DATE     DateTime @default(now()) @db.DateTime(0)    users    users    @relation(fields: [ID\_USER], references: [ID], onDelete: NoAction, onUpdate: NoAction, map: "feedback\_ibfk\_1")    books    books    @relation(fields: [ID\_BOOK], references: [ID], onDelete: NoAction, onUpdate: NoAction, map: "feedback\_ibfk\_2")    @@index([ID\_BOOK], map: "ID\_BOOK")    @@index([ID\_USER], map: "ID\_USER")  }  model genre {    ID         Int          @id @default(autoincrement())    GENRE\_NAME String       @unique(map: "GENRE\_NAME") @db.VarChar(50)    book\_genre book\_genre[]  }  model tag {    ID       Int        @id @default(autoincrement())    TAG\_NAME String     @unique(map: "TAG\_NAME") @db.VarChar(50)    book\_tag book\_tag[]  }  model user\_bookmarks {    USERID         Int    BOOKID         Int    STATUS\_READING String    @db.VarChar(25)    DATA\_READING   DateTime? @db.DateTime(0)    DATA\_ADD       DateTime  @default(now()) @db.Timestamp(0)    MARKS          String?   @db.VarChar(1000)    ID             Int       @id @default(autoincrement())    users          users     @relation(fields: [USERID], references: [ID], onDelete: NoAction, onUpdate: NoAction, map: "user\_bookmarks\_ibfk\_1")    books          books     @relation(fields: [BOOKID], references: [ID], onDelete: NoAction, onUpdate: NoAction, map: "user\_bookmarks\_ibfk\_2")    @@index([BOOKID], map: "BOOKID")    @@index([USERID], map: "USERID")  }  model users {    ID             Int              @id @default(autoincrement())    EMAIL          String           @unique(map: "EMAIL") @db.VarChar(50)    pass           String           @db.VarChar(500)    USER\_NAME      String           @db.VarChar(20)    USER\_INFO      String?          @db.Text    ACCESS\_LEVEL   String           @db.VarChar(20)    DATA\_CREATE    DateTime         @default(now()) @db.DateTime(0)    feedback       feedback[]    user\_bookmarks user\_bookmarks[]    authors        authors[]  } |

Листинг 1 – Схема Prisma базы данных

ПРИЛОЖЕНИЕ Б

Листинг файла docker-compose.yaml

|  |
| --- |
| services:    mysql:      image: mysql:8.0      container\_name: mysql\_container      environment:        MYSQL\_ROOT\_PASSWORD: root\_password        MYSQL\_DATABASE: my\_database        MYSQL\_USER: user        MYSQL\_PASSWORD: user\_password      ports:        - "3306:3306"      volumes:        - mysql\_data:/var/lib/mysql    server:      build:        context: ./server        dockerfile: Dockerfile      container\_name: server      ports:        - "3001:3001"      env\_file:        - ./server/.env      restart: always      depends\_on:        - mysql      command: >        sh -c "npx prisma migrate deploy &&               node app.js"    client:      build:        context: ./client        dockerfile: Dockerfile      container\_name: client      ports:        - "3000:3000"      env\_file:        - ./client/.env      restart: always      depends\_on:        - server  volumes:    mysql\_data: |

Листинг 2 – код yaml-файла для контейнеризации приложения