МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ

ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет Информационных технологий

Кафедра Информационные системы и технологии

Специальность 1–40 01 01 Программное обеспечение информационных технологий

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

**К КУРСОВОЙ РАБОТЕ НА ТЕМУ:**

«Web-приложение «Библиотека»

Выполнил студент Подобед Владислав Георгиевич

(Ф.И.О.)

Руководитель работы асс. Нистюк Ольга Александровна

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

И.о. зав. кафедрой ст. преп. Блинова Евгения Александровна

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

Курсовая работа защищена с оценкой

Минск 2024

**Содержание**

[Введение 3](#_Toc166169594)

[1 Постановка задачи 4](#_Toc166169595)

[1.1 Обзор аналогов 4](#_Toc166169596)

[2 Проектирование web-приложения 7](#_Toc166169597)

[2.1 Диаграмма использования 7](#_Toc166169598)

[2.2 Структурная схема приложения 8](#_Toc166169599)

[2.3 Проектирование базы данных 8](#_Toc166169600)

[3 Разработка web-приложения 13](#_Toc166169601)

[3.1 Разработка серверной части 13](#_Toc166169602)

[3.2 Разработка клиентской части 16](#_Toc166169603)

[4 Тестирование web-приложения 19](#_Toc166169604)

[5 Руководство пользователя 22](#_Toc166169605)

[5.1 Регистрация пользователя 22](#_Toc166169606)

[5.2 Аутентификация и авторизация пользователя 23](#_Toc166169607)

[5.3 Главная страница 24](#_Toc166169608)

[5.4 Страница книги 24](#_Toc166169609)

[5.5 Личная страница пользователя 25](#_Toc166169610)

[5.6 Панель администратора 26](#_Toc166169611)

[5.7 Удаление комментарием 27](#_Toc166169612)

[Заключение 29](#_Toc166169613)

[Список используемых источников 30](#_Toc166169614)

[Приложение А 31](#_Toc166169615)

[Приложение Б 34](#_Toc166169616)

[Приложение Г 35](#_Toc166169617)

# Введение

С течением времени библиотеки стали неотъемлемой частью общественной и культурной жизни, играя ключевую роль в сохранении, распространении и передаче знаний. Исторически служащие хранилищами книг, библиотеки превратились в динамичные центры, предоставляющие доступ к разнообразным информационным ресурсам и предоставляющие место для обучения, исследований и культурного обмена. является создание такого web-приложения, которое позволит пользователям эффективно управлять библиотекой книг.

Основываясь на описанных выше задачах, данная работа направлена на разработку и реализацию веб-приложения, которое будет предоставлять широкий спектр функций, включая регистрацию и аутентификацию пользователей, а также управление содержимым библиотеки книг.

Целью данного курсового проекта является не только создание веб-приложения, но и исследование возможностей его применения в контексте современных библиотечных и информационных технологий, а также оценка его эффективности и перспективы дальнейшего развития.

Для достижения этой цели были определены следующие задачи:

* обеспечивать возможность регистрации и авторизации;
* поддерживать роли администратора и пользователя;
* позволять добавление новых книг и информацию о них в библиотеку;
* предоставлять редактировать информацию о книгах;
* возможность чтения и скачивания книг;
* предоставлять возможность поиска книг по критериям;
* предоставить возможность просматривать информацию о книгах;
* предоставлять пользователю оставлять отзывы о книгах.

Важным аспектом проекта является не только техническая реализация функционала, но и учет потребностей и предпочтений пользователей, обеспечивая им максимальное удобство и функциональность при работе с приложением.

1 Постановка задачи

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.

## 1.1 Обзор аналогов

litres.ru - это популярная онлайн-платформа для любителей чтения книг. Она предоставляет возможность пользователям искать, открывать и оценивать книги. litres.ru 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.

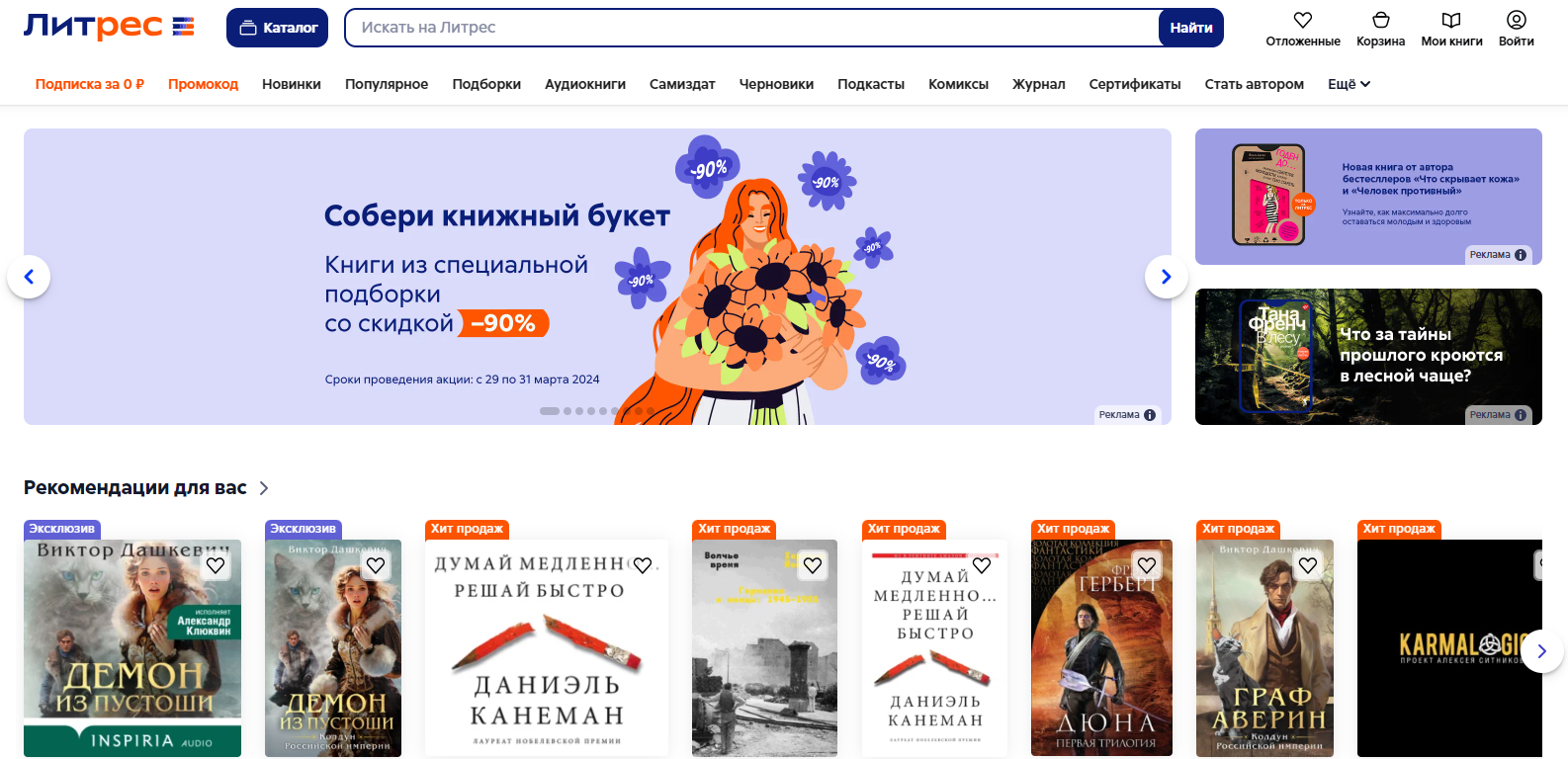


Рисунок 1.1 – Домашняя страница Litres

Преимущества данного сайта представлены ниже:

* Большой каталог книг;
* Сообщество чтения: Litres 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

Bookmate.ru - 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.

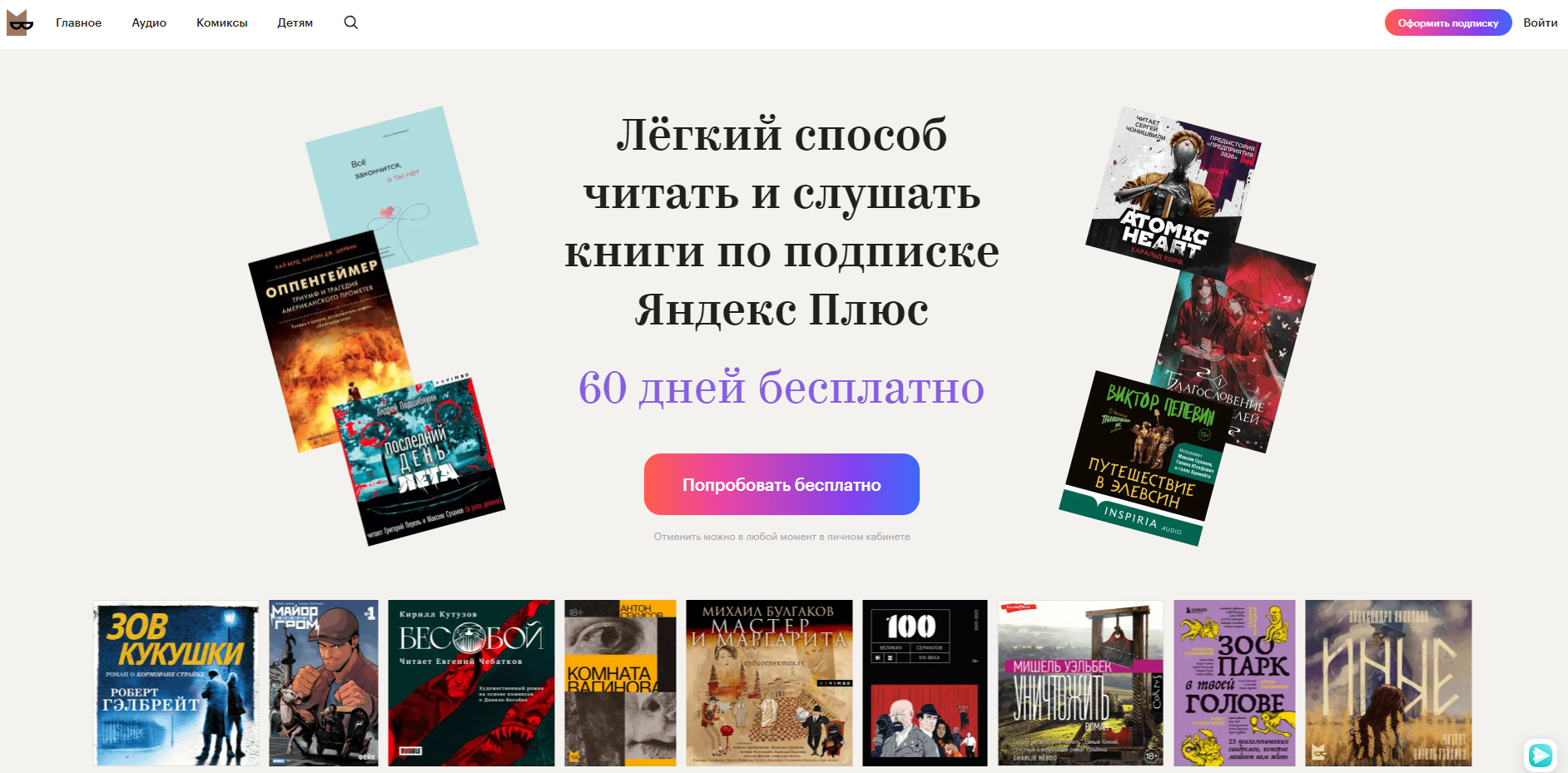


Рисунок 1.2 – Домашняя страница Bookmate

В целом, Bookmate 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.

Bookmate имеет несколько преимуществ:

* Бесплатный доступ;
* Общественное достояние;
* Разнообразие книг. Коллекция Bookmate 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 Bookmate 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.

2 Проектирование web-приложения

## 2.1 Диаграмма использования

Диаграмма использования для разрабатываемого приложения представлена на рисунке 2.1.

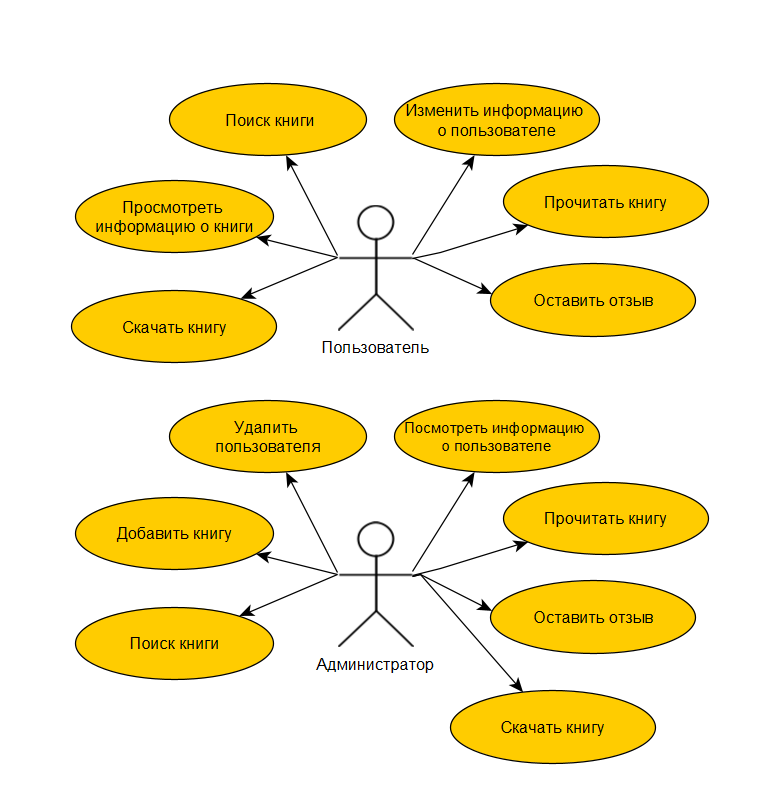


Рисунок 2.1 – Диаграмма использования

Диаграмма использования включает в себя две роли: администратор, пользователь.

Администратор может удалить пользователя, добавить книгу, скачать книгу, посмотреть информацию о пользователе, поиск книги, оставить отзыв, прочитать книгу.

Пользователь имеет возможность поиск книг, просмотреть информацию о книги, скачать книгу, изменить информацию о пользователе, прочитать книгу, оставить отзыв.

## 2.2 Структурная схема приложения

Структурная схема приложения представлена на рисунке 2.2.

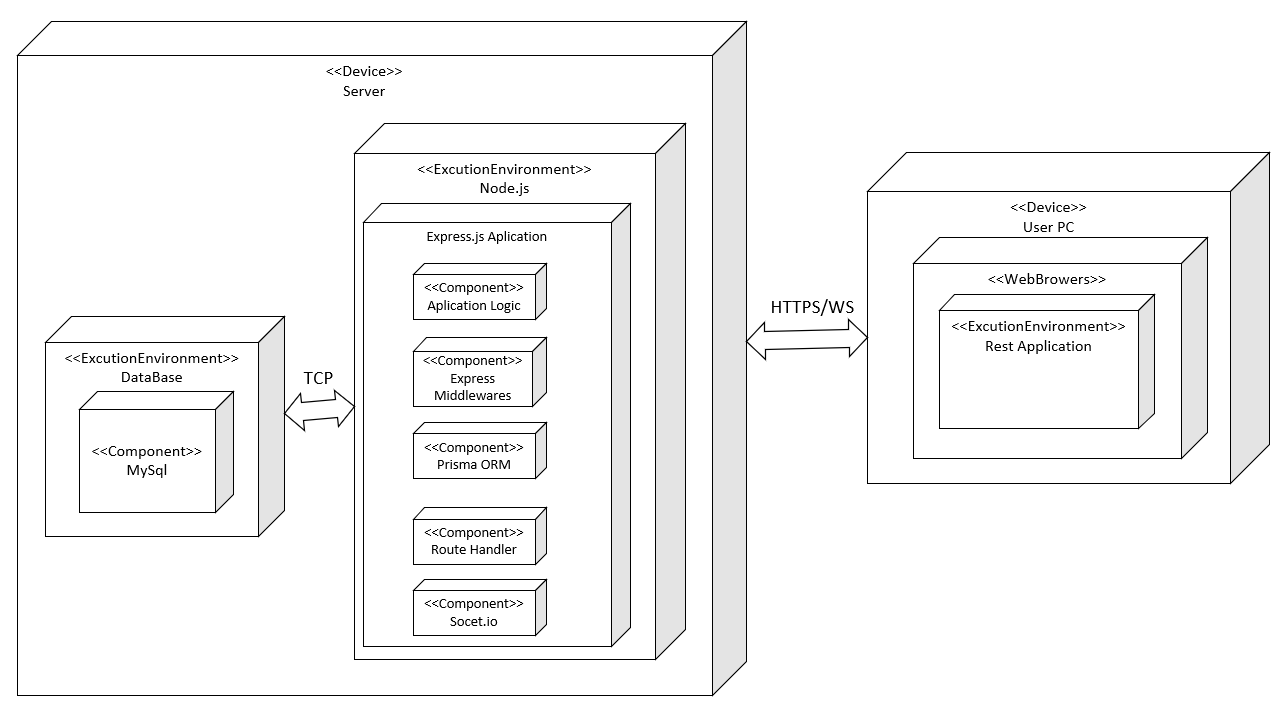


Рисунок 2.2 – Структурная схема приложения

На клиентской стороне используются технологии React для создания пользовательского интерфейса, Socket.IO для обмена данными в реальном времени с сервером.

Серверная часть реализована с использованием фреймворка Express. Он обрабатывает запросы, поступающие от клиента, и использует Prisma для взаимодействия с базой данных.

## 2.3 Проектирование базы данных

Модель данных для данного проекта должна быть тщательно спроектирована с учетом всех функциональных требований приложения. Это включает определение структуры таблиц, их атрибутов и связей между ними. Например, для хранения информации о пользователях могут быть созданы таблицы с данными о пользователе, их учетных записях, ролях и прочих сопутствующих данных.

Кроме того, для эффективного управления библиотекой необходимо разработать структуру таблиц, которая позволит хранить информацию о книгах, их авторах, жанрах, издательствах и других атрибутах. Для обеспечения возможности оставления отзывов о книгах и работы с задачами пользователей, также потребуется определить соответствующие таблицы и связи между ними.

Cерверная 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.

Модель базы данных для приложения представлена на рисунке 2.3.

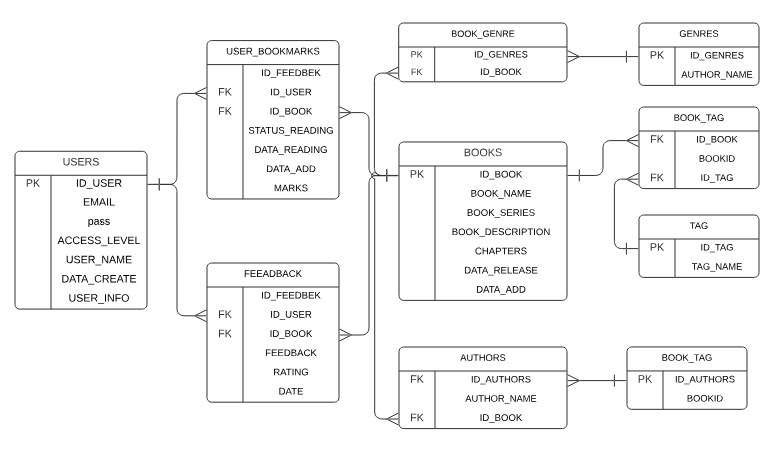


Рисунок 2.3 – Логическая схема базы данных

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.

В таблице «Users» хранятся все зарегистрированные пользователи. Описание структуры таблицы 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 «Books» хранятся книги. Её структура представлена 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 |

В таблице «Authors» содержит названия авторов книг. Описание структуры таблицы представлено 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 |

В таблице «Genres» содержит названия жанров. Описание структуры таблицы представлено в таблице 2.4.

Таблица 2.4 – Структура таблицы «Genres»

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

В таблице «Tag содержит название тегов используемый в книжках. Описание таблицы «Tag» представлено в таблице 2.5.

Таблица 2.5 – Структура таблицы «Subject»

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

В таблице «Book\_author» описывается какие авторы присвоены книге. Описание таблицы «Book\_author» представлено в таблице 2.6.

Таблица 2.6 – Структура таблицы «StudentSubject»

|  |  |  |
| --- | --- | --- |
| 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 |

В таблице «Book\_genre» содержатся информация о созданных тестах. Описание таблицы «Book\_genre» 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 |

В таблице «Book\_tag» описывается какие теги присвоены книге. Описание таблицы «Book\_tag» 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 |

В таблице «Feeadback» описывается какие теги присвоены книге. Описание таблицы «Feeadback» представлено в таблице 2.9.

Таблица 2.9 – Структура таблицы «Feeadback»

|  |  |  |
| --- | --- | --- |
| 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 |

В таблице «User\_bookmarks» описывается какие теги присвоены книге. Позволяет пользователям управлять своими закладками и организовывать книги по различным категориям или темам с помощью тегов. Это обеспечивает удобство навигации и поиска книг в приложении. Описание таблицы «User\_bookmarks» 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 |

При проектировании базы данных приложения с использованием 10 таблиц, присутствует связь один-ко-многим и многие-ко-многим.

Общая архитектура базы данных обеспечивает гибкость в управлении данными, высокую производительность и безопасность. Позволяет эффективно управлять административными и учебными аспектами приложения. Позволяет оптимизировать использование ресурсов системы и обеспечить эффективное взаимодействие с данными. Такая архитектура базы данных является надежным фундаментом для приложения, обеспечивая надежность и эффективность в управлении информацией.

Исходя из вышеизложенного, использование СУБД MySQL является рациональным выбором для данного приложения. MySQL обеспечивает стандартизированный синтаксис, широкий функционал, высокую производительность, безопасность, расширяемость и совместимость с другими системами. Все эти преимущества делают SQL-серверы надежным и эффективным решением для управления базой данных приложения.

Использование СУБД MySQL также обеспечивает простоту в установке и настройке, что делает его привлекательным выбором для разработчиков и администраторов баз данных. Благодаря открытому исходному коду и активному сообществу разработчиков, MySQL постоянно обновляется и совершенствуется, что позволяет быстро реагировать на изменения в требованиях приложения и использовать новые функциональные возможности. Кроме того, MySQL предоставляет широкий спектр инструментов для мониторинга и управления базой данных, что облегчает администрирование и обеспечивает высокую доступность и отказоустойчивость системы.

3 Разработка web-приложения

Разработка web-приложения проекта была осуществлена с использованием современных технологий и подходов, направленных на обеспечение эффективной работы и удобства пользователей. React для фронтенд-части приложения, чтобы создать интерактивный и отзывчивый пользовательский интерфейс. Для управления состоянием приложения и обмена данными в реальном времени мы использовали Socket.IO. На серверной стороне использует Node.js, а с помощью контроллеров, middleware и моделей, обеспечивая структурированную и безопасную работу с данными. Это позволило создать мощное и надежное веб-приложение, способное эффективно управлять образовательным процессом и обеспечить удобство как преподавателям, так и студентам.

## 3.1 Разработка серверной части

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

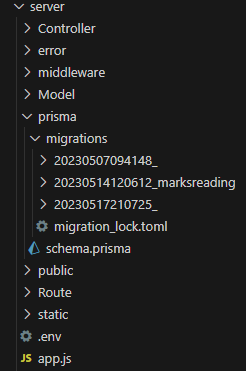


Рисунок 3.1 – Структура серверной части

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.

В директории «modules» содержится модули для каждой предметной области приложений. В каждой такой папке есть файл роутера, сервиса и репозитория.

Для удобной реализации серверной части приложения был выбран фреймворк Express.js. Этот фреймворк позволяет разделить сервер на модели, контроллеры и роуты, обеспечивая более структурированный подход к разработке.

Рассмотрим одну из моделей feedback – представляет собой модель принимающий отзыв от пользователя или администратора. Данный метод связывается с prisma.client обращаясь к таблице feedback. В результате запроса к бд возвращает указанный отзыв. Такой запрос можно увидеть в листинге 3.1

|  |
| --- |
| const prisma = new client.PrismaClient();  async function getFeedback(page)  { page = parseInt(page);  return await prisma.feedback.findMany({ skip: skip(page), take: take }); }  async function getFeedbackById(id)  { id = parseInt(id);  return await prisma.feedback.findUnique({ where: { id: id, },  }); } |

Листинг 3.1 – Содержимое файла «feedback»

Код в «app.js» отвечает за создание и настройку серверной части приложения с использованием Express.js. Содержимое представлено в листинге 3.2.

|  |
| --- |
| const app = express();  const options = {  key: fs.readFileSync('LAB.key'),  cert: fs.readFileSync('LAB.crt')  };  const http = require('https').createServer(options, app);  const io = require("socket.io")( http, {  cors: {  origin: "http://localhost:3000",  methods: ["GET", "POST"]  }}); |

Листинг 3.2 – Содержимое файла «app»

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 – Содержимое файла «indexRout»

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 – Содержимое файла «ApiError»

Данный код достает из заголовка запроса токе 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 аунтефикации используется jwt – авторизация. Это довольно простой и защищенный способ авторизации пользователей.

Так же в серверной части используются статические файлы, такие как статические функции вычисляющие skip значения по номеру страницы.

Обработка статический файлов происходит за счет express.static он позволяет без проблем отображать файлы на клиентской части.

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

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

## 3.2 Разработка клиентской части

Разработка клиентской части приложения включает в себя создание интерфейса пользователя (UI) и реализацию взаимодействия с сервером для получения и отправки данных. На рисунке 3.2 изображена структура клиентской части разрабатываемого приложения.

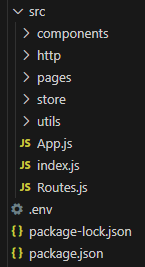


Рисунок 3.2 – Структура клиентской части

Клиентская часть приложения написана на React, JavaScript-библиотеке для создания пользовательских интерфейсов. React является одной из самых популярных библиотек для создания веб-приложений. React используется для создания одностраничных приложений (SPA), где все действия происходят на одной странице, без перезагрузки. Он основан на концепции компонентов, которые могут быть повторно использованы и являются основными строительными блоками приложения. ReactJS также использует виртуальный DOM (Document Object Model), что позволяет обновлять только те части страницы, которые изменились, вместо того чтобы перерисовывать всю страницу целиком. Это делает приложения, созданные с помощью ReactJS, более быстрыми и отзывчивыми. В целом, ReactJS - это мощный инструмент для создания высокопроизводительных и масштабируемых пользовательских интерфейсов для веб-приложений.

React также предоставляет множество инструментов и библиотек для управления состоянием приложения, маршрутизации, обработки событий и взаимодействия с сервером. С помощью библиотеки React Router можно легко управлять навигацией в одностраничном приложении, определяя маршруты и компоненты, которые должны отображаться при переходе по определенным адресам URL.

Директория «src» содержит исходный код клиентской части приложения. Основной точкой входа, для клиента является файл «index», cодержимое представлено в листинге 3.4.

|  |
| --- |
| export const Context = React.createContext(null);  console.log(process.env.REACT\_APP\_API\_URL);  const root = ReactDOM.createRoot(document.getElementById('root'));  root.render(  <Context.Provider value={{  user: new UserStore(),  book: new BookStore(),  currentBook: new Book()  }}>  <App />  </Context.Provider>  ); |

Листинг 3.4 – Содержимое файла «index»

Сначала создается корневой элемент приложения с помощью метода ReactDOM.createRoot(). Этот метод принимает элемент DOM, к которому будет привязан корневой компонент приложения. В данном случае элемент с идентификатором "root" извлекается из DOM с помощью document.getElementById('root').

После вызова метода root.render(), который выполняет рендеринг корневого компонента приложения, происходит следующий процесс. Внутри метода render() компонент <App /> передается в качестве основного компонента приложения.

Обертка компонента <App /> активирует строгий режим React. Этот режим помогает выявлять потенциальные проблемы и несовершенства в коде, обеспечивая более чистую и стабильную работу приложения.

В результате этого кода корневой компонент <App /> рендерится в указанном элементе DOM, и React начинает управлять интерфейсом приложения. Он будет обновлять интерфейс при изменениях состояния и данных, обеспечивая реактивность и отзывчивость приложения.

Таким образом, вызов метода root.render() с компонентом <App /> и активация строгого режима React позволяют запустить и отображать основной компонент приложения, а React обеспечивает эффективное управление его состоянием и интерфейсом.

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.

React Bootstrap предлагает множество других компонентов.

Для перемещения между страницами используется библиотека router, применение которой представлено в листинге 3.5.

|  |
| --- |
| export const authRoutes = [    {  path: BOOKMARKS\_ROUTE,  element: BookMarks,  },    {  path: USER\_ROUTE + '/:id',  element: UserPage,  } |

Листинг 3.5 – Содержимое файла «router»

App.js определяет основную структуру и маршрутизацию приложения, включая вложенные компоненты и их расположение представлено в листинге 3.6.

|  |
| --- |
| const App = observer(() => {  const { user } = useContext(Context);  const [loading, setLoading] = useState(true);  const [alertshow, setAlertshow] = useState(false);  const [variant, setVariant] = useState("success");  const [alertText, setAlertText] = useState("Успешно");  useEffect(() => {  console.log(process.env.REACT\_APP\_API\_URL);  setAlertshow(false);  check().then(data => {  user.setUser(true);  user.setIsAuth(true);  }).catch((e) => {  if (!e.response) {  setAlertshow(true);  setVariant("danger");  localStorage.removeItem('token');  localStorage.removeItem('id');  setAlertText("Cервер вернул ошибку: " + e.message);  console.log(e);  }  }).finally(() => setLoading(false)) |

Листинг 3.6. – Содержимое файла «api»

4 Тестирование web-приложения

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

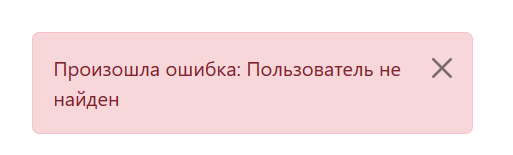


Рисунок 4.1 – Ошибка некорректных данных

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 корректной. Случай такой ошибки представлен на рисунке 4.3.

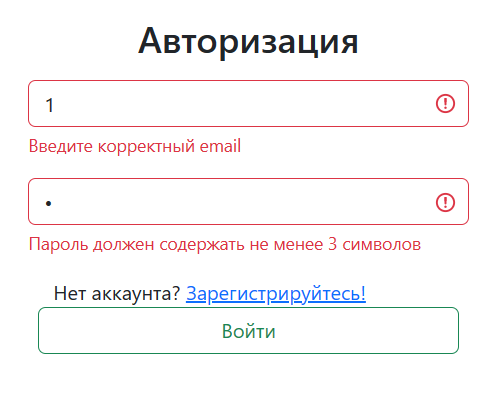


Рисунок 4.2 – Ошибка заполнения полей password и email

Валидация выполняется на стороне сервера, и в случае, если она не пройдена, возвращается ошибка, которая выводится на стороне клиента.

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 тега

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.

В результате тестирования приложения было обнаружено, что компоненты корректно рендерятся и взаимодействуют с пользовательскими действиями. Также было успешно протестировано сохранение и обновление прогресса чтения книги, а также отображение информации о текущей странице. В целом, приложение прошло основные тестовые сценарии и работает стабильно.

В случае если сервер или бд будут не доступны, то возникнет ошибка. Это можно увидеть на рисунке 4.4

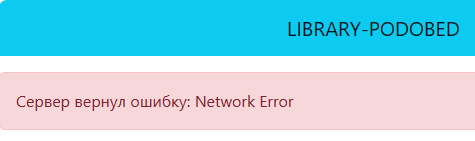


Рисунок 4.4 – Ошибка при неработающем сервере или бд

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.

В результате тестирования приложения было обнаружено, что компоненты корректно рендерятся и взаимодействуют с пользовательскими действиями. Также было успешно протестировано сохранение и обновление прогресса чтения книги, а также отображение информации о текущей странице. В целом, приложение прошло основные тестовые сценарии и работает стабильно.

В результате тестирования приложения было также выявлено и успешно обработано несколько потенциальных проблемных сценариев, таких как некорректный ввод данных пользователем, ошибки сервера, и прочие непредвиденные ситуации. Благодаря грамотной обработке и отображению информации об ошибках, пользователи могут легко идентифицировать проблемы и предпринять соответствующие действия для их решения.

Тестирование также подтвердило эффективность механизмов безопасности, реализованных в приложении, таких как аутентификация и авторизация пользователей, защита от межсайтового скриптинга (XSS) и инъекций SQL. Это обеспечивает надежную защиту данных пользователей и предотвращает возможные атаки на приложение.

В результате проведенного тестирования можно с уверенностью сказать, что приложение готово к использованию и соответствует высоким стандартам качества и безопасности.

В целом, результаты тестирования подтверждают, что приложение обладает высоким уровнем качества, производительности и безопасности, и готово к использованию в реальных условиях эксплуатации.

5 Руководство пользователя

## 5.1 Регистрация пользователя

Для регистрации пользователя, необходимо ввести электронную почту и пароль, а также дополнительные детали, в зависимости от роли создаваемого пользователя. Форма регистрации представлена на рисунке 5.1.

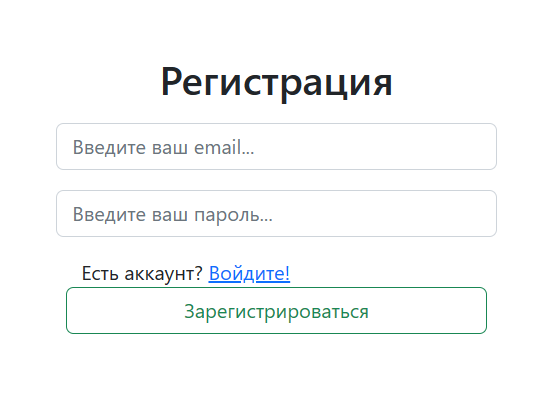


Рисунок 5.1 – Окно регистрации

Для создания аккаунта нужно следовать следующим шагам:

* Зайдите на главную страницу приложения.
* Нажмите на ссылку "Регистрация" или "Создать аккаунт".
* Заполните необходимые поля в форме регистрации, включая имя пользователя, адрес электронной почты и пароль.
* Подтвердите правильность введенной информации.
* Нажмите кнопку "Зарегистрироваться" для создания аккаунта.
* Если все данные введены правильно, вы будете перенаправлены на страницу входа в систему.

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

## 5.2 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.

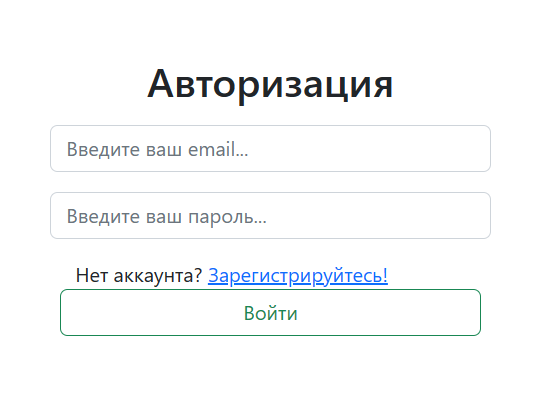


Рисунок 5.2 – Форма авторизации

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.

## 5.3 Главная страница

При успешном логине 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 рейтиг книг, который вычисляется по отзывам других пользователей.

Данная страница предоставлена на рисунке 5.3.

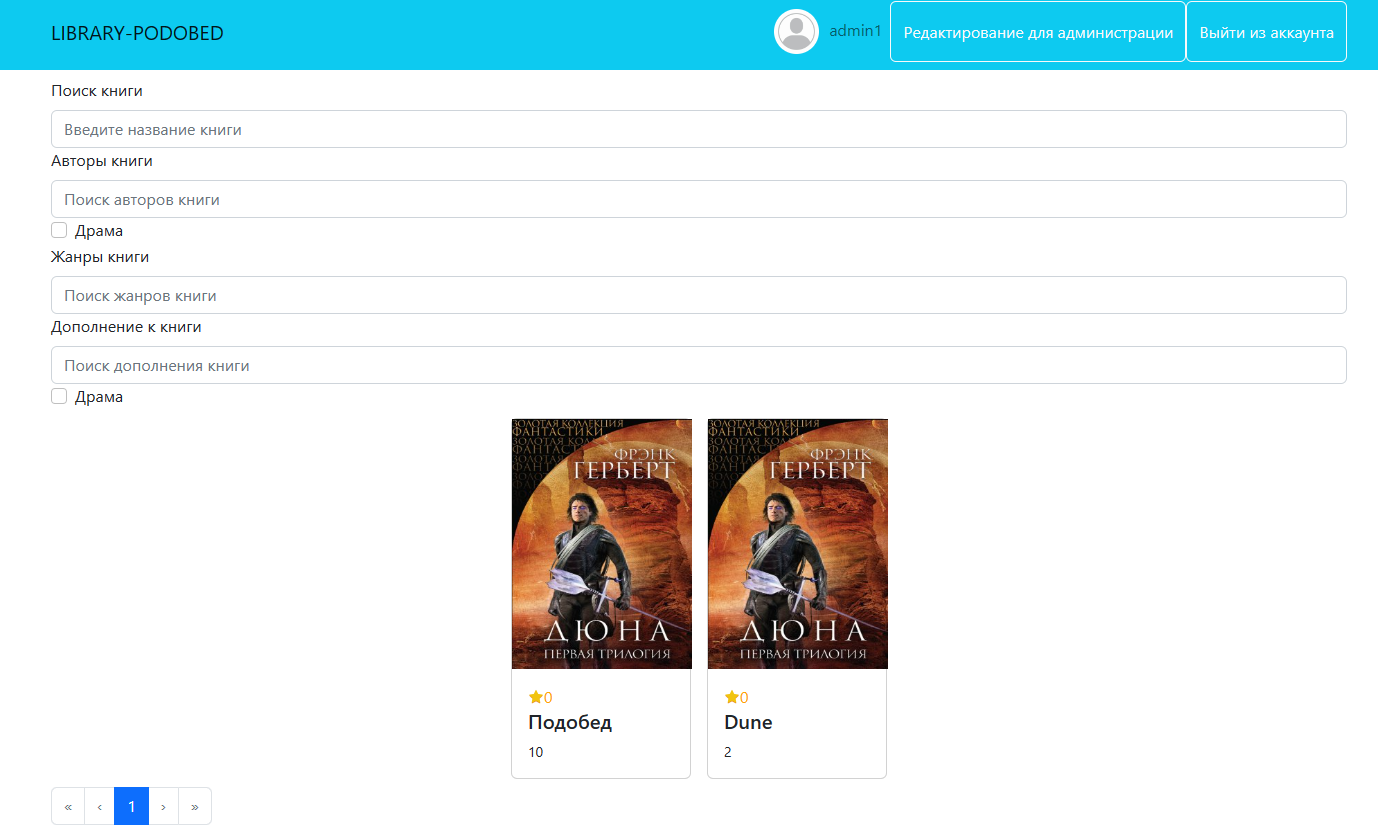


Рисунок 5.3 – Страница поиска

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

## 5.4 Страница книги

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

Это позволяет пользователям более детально ознакомиться с содержанием книги, её автором, жанром и другими характеристиками перед началом чтения.

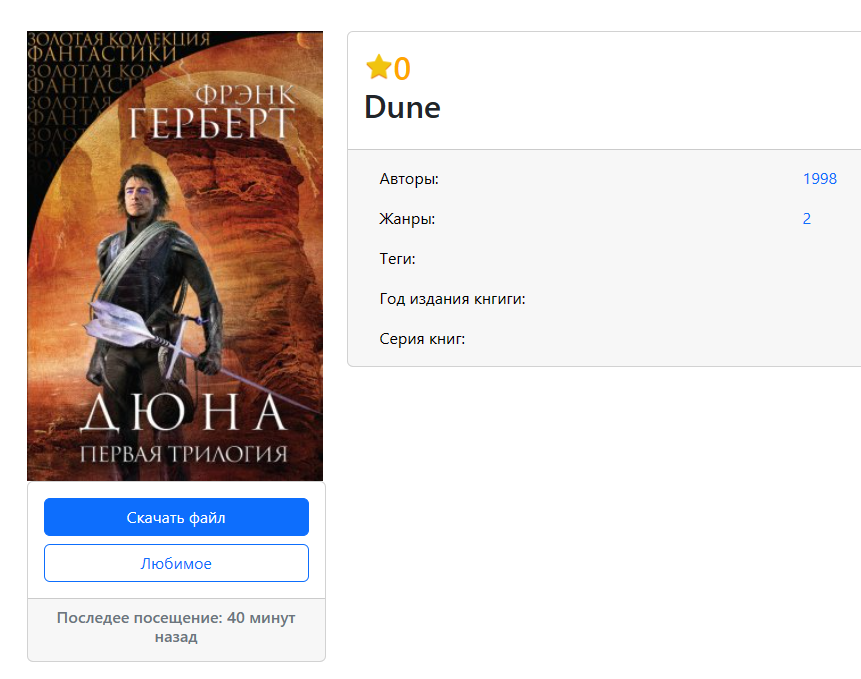


Рисунок 5.3 – Страница поиска

Так же на странице можно добавить закладку и оставить отзыв.

## 5.5 Личная страница пользователя

Нажав на аватарку или имя можно попасть на страницу пользователя. Эта страница представлена на рисунке 5.5.

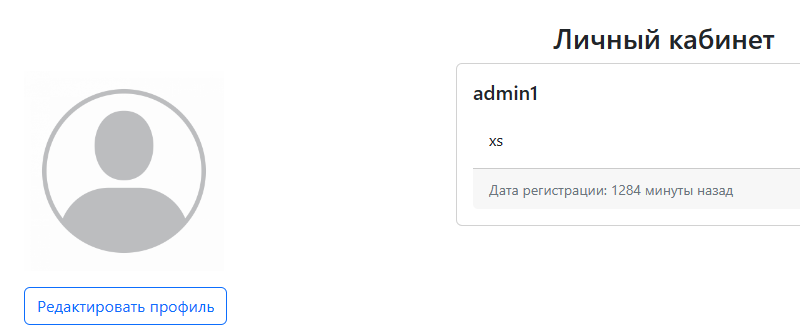


Рисунок 5.5 – Личная страница пользователя

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

## 5.6 Панель администратора

В данном курсовом проекте была разработана возможность администрирования. Страница панели администратора представлена на рисунке 5.6.

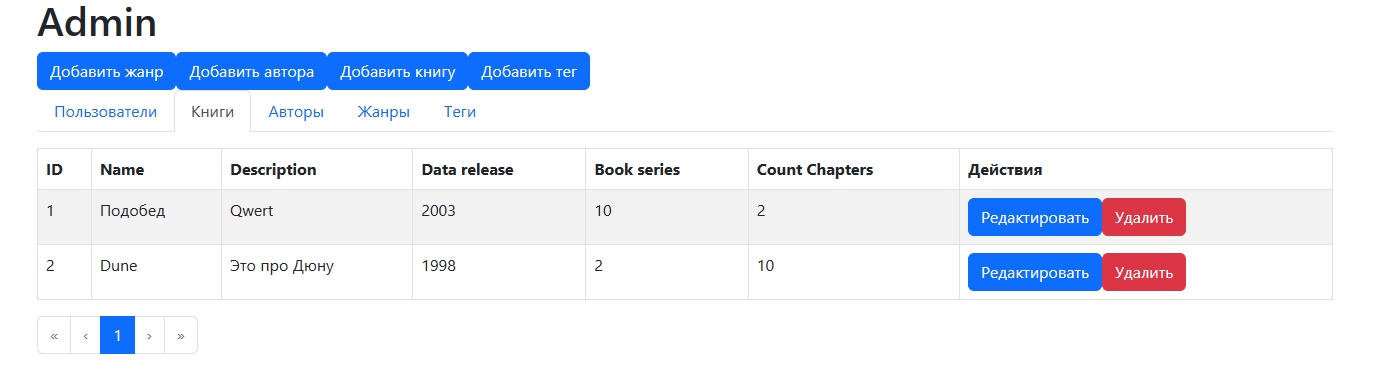


Рисунок 5.6 – Панель администратора

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 создания автора.

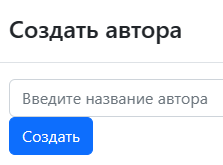


Рисунок 5.7 – Форма создания автора

При изменении книги мы перейдем на страницу изменения книги. Это представлено на странице 5.8 в книге, где описывается процесс редактирования информации о книге, включая ее название, авторов, описание и другие характеристики. На этой странице пользователь может вносить изменения в информацию о книге и сохранять их для последующего использования.

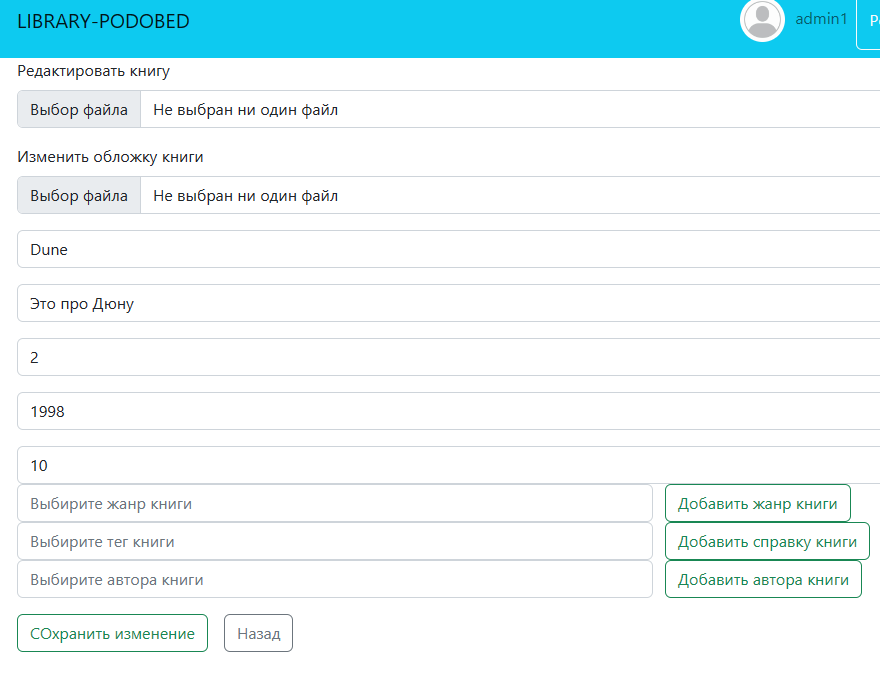


Рисунок 5.8 – Форма изменения книги

Тут можно добавить книге жанров, тегов и авторов, а также обновить обложку и файл.

## 5.7 Удаление комментарием

Администратор, может удалять комментарии у книги. Кнопка удаление представлена на рисунке 5.9.

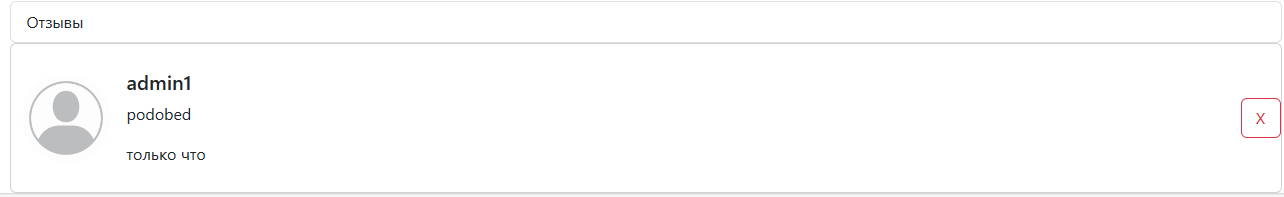


Рисунок 5.9 – Кнопка удаление

Администратор, как и пользователь может модерировать собственные списки книг. Пример таких списков представлен на рисунке 5.10

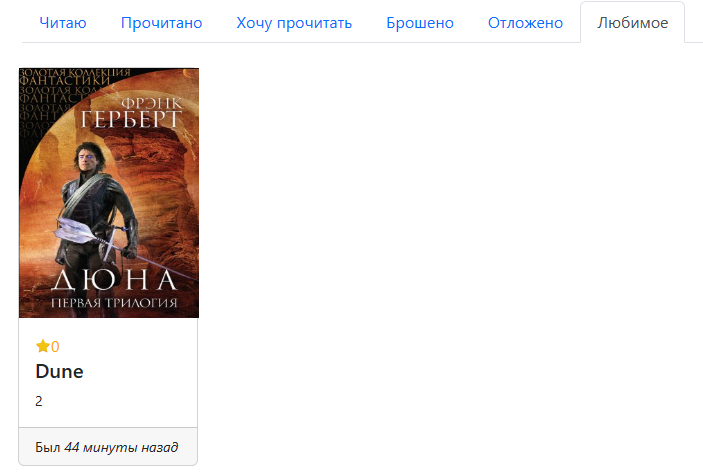


Рисунок 5.10 – Личные списки книг

Тут представленные различные списки книг, так же видно, когда последний обновлялась закладка.

# Заключение

В ходе данного курсового проекта было разработано удобное, безопасное web-приложение.

Перед началом разработки был произведен аналитический обзор прототипов приложений подобной тематики и определение функциональных возможностей разрабатываемого приложения.

Был разработан удобный и понятный интерфейс, в котором могут разобраться даже те пользователи, которые ни разу не посещали подобные сайты.

Приложение предоставляет возможность ведения учета студентов на занятии.

В процессе разработки использовались современные и популярные библиотеки, модули и фреймворки, которые позволили полностью реализовать задуманный функционал.

В процессе решения поставленной задачи была достигнута поставленная цель по Функциональное web-приложение, разработанное с использованием технологий React, Express, и MySql, представляет собой полноценную библиотеку для чтения и управления информацией о книгах.

Важным этапом разработки было тестирование программного продукта, которое позволило выявить и исправить ошибки и недостатки в работе приложения.

Тестирование позволило убедиться, что программа работает верно и соответствует требованиям технического задания.

# Список используемых источников

1. React.js документация [Электронный ресурс]. – Режим доступа: <https://www.reactjs.com> – Дата доступа: 25.02.2024.
2. Socket.io документация [Электронный ресурс]. – Режим доступа: <https://socket.io>– Дата доступа: 09.03.2024
3. Axios документация [Электронный ресурс]. – Режим доступа: [https://axios-http.com](https://axios-http.com/) – Дата доступа: 30.02.2024.

4 React Context API документация [Электронный ресурс]. – Режим доступа: https://reactjs.org/docs/context.html – Дата доступа: 25.02.2024.

# Приложение А

|  |
| --- |
| generator client {  provider = "prisma-client-js"  }  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)  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[]  } |

Листинг «Модели базы данных»

# Приложение Б

{

"name": "client",

"version": "0.1.0",

"private": true,

"dependencies": {

"@testing-library/jest-dom": "^5.16.5",

"@testing-library/react": "^13.4.0",

"@testing-library/user-event": "^13.5.0",

"axios": "^1.3.6",

"bootstrap": "^5.2.3",

"epubjs": "^0.3.93",

"formik": "^2.2.9",

"jwt-decode": "^3.1.2",

"localforage": "^1.10.0",

"mdb-react-ui-kit": "^6.0.0",

"mdbreact": "^5.2.0",

"mobx": "^6.9.0",

"mobx-react-lite": "^3.4.3",

"pdfjs-dist": "^3.6.172",

"react": "^18.2.0",

"react-bootstrap": "^2.7.4",

"react-bootstrap-typeahead": "^6.1.2",

"react-dom": "^18.2.0",

"react-reader": "^1.2.1",

"react-router-dom": "^6.10.0",

"react-scripts": "^5.0.1",

"react-select": "^5.7.3",

"socket.io-client": "^4.6.1",

"web-vitals": "^2.1.4"

},

"scripts": {

"start": "react-scripts start",

"build": "react-scripts build",

"test": "react-scripts test",

"eject": "react-scripts eject"

},

"eslintConfig": {

"extends": [

"react-app",

"react-app/jest"

]

},

"browserslist": {

"production": [

">0.2%",

"not dead",

"not op\_mini all"

],

"development": [

"last 1 chrome version",

"last 1 firefox version",

"last 1 safari version"

]

}

} "typescript": "^5.1.3"

Файл package.json

# Приложение Г

|  |
| --- |
| const Route = require('express').Router();  const authorController = require('../Controller/authorController');  const checkRole = require('../middleware/checkRoleMiddleware');  Route.get('/:page', authorController.getAuthors);  Route.post('/book/:id', checkRole('ADMIN'), authorController.addAuthor);  Route.delete('/book/:id/:author', checkRole('ADMIN'), authorController.deleteAuthor);  Route.get('/book/:id', authorController.getBookAuthors);  //TODO: Неверный контроллер  Route.get('/book/:id/:book', authorController.getBookAuthors);  Route.post('/', checkRole('ADMIN'),authorController.createAuthor);  Route.put('/:id', checkRole('ADMIN'), authorController.updateAuthor);  Route.delete('/:id', checkRole('ADMIN'), authorController.delete);  module.exports = Route;  const Route = require('express').Router();  const bookmarkController = require('../Controller/user\_bookmarksController');  const authMiddleware = require('../middleware/authMiddleware');  const checkRole = require('../middleware/checkRoleMiddleware');  Route.get('/:page', authMiddleware, bookmarkController.getBookMarks);  Route.get('/:id', authMiddleware, bookmarkController.getBookMarksById);  Route.get('/user/:id', authMiddleware, bookmarkController.getBookMarksByUserId);  //Добавить закладку  Route.post('/', authMiddleware, bookmarkController.createBookMark);  //Получить закладку по id пользователя и id книги  Route.get('/user/:id\_user/:id\_book', authMiddleware, bookmarkController.getBookMarksByUserIdAndBookId);  //getStatusReading  Route.get('/status/:id/:reading', authMiddleware, bookmarkController.statusReading);  //updateStatusReading  Route.put('/status/', authMiddleware, bookmarkController.updateStatusReading);  //getBookMarksUser  Route.get('/user/:id/:sort/:page', authMiddleware, bookmarkController.getBookMarksByUserId);  module.exports = Route;  const Route = require('express').Router();  const bookController = require('../Controller/booksController');  const checkRole = require('../middleware/checkRoleMiddleware');  Route.get('/:page', bookController.getBooks);  Route.post('/search/:page', bookController.getBookSearch);  Route.get('/id/:id', bookController.getBookById);  Route.get('/raiting/:id', bookController.getBookAvarageRaiting);  // Route.get('/name/:name', bookController.getBooksByName);  // Route.get('/series/:series/:page', bookController.getBooksBySeries);  // Route.get('/author/:author/:page', bookController.getBooksByAuthor);  // Route.get('/genre/:genre/:page', bookController.getBooksByGenre);  // Route.get('/status/:status/:page', bookController.getBooksByStatus);  //Удаление, добавление, изменение данных  Route.post('/', checkRole('ADMIN'), bookController.create);  Route.put('/:id', checkRole('ADMIN'), bookController.update);  Route.delete('/:id', checkRole('ADMIN'), bookController.delete);  //Ganre  //Tags  module.exports = Route;  const Rote = require('express').Router();  const feedbackController = require('../Controller/feedbackController');  const authMiddleware = require('../middleware/authMiddleware');  Rote.get('/:page', feedbackController.getFeedback);  Rote.get('/:id', feedbackController.getFeedbackById);  Rote.get('/user/:id/:page', feedbackController.getFeedbackByUserId);  Rote.get('/book/:id/:page', feedbackController.getFeedbackByBookId);  //Добавить отзыв  Rote.post('/', authMiddleware, feedbackController.addFeedback);  //Удалить отзыв  Rote.delete('/:id', authMiddleware, feedbackController.deleteFeedback);  //Обновить отзыв  Rote.put('/:id', authMiddleware, feedbackController.updateFeedback);  module.exports = Rote;  const ApiError = require('../error/ApiError');  const Route = require('express').Router();  Route.post('/upload', (req, res) => {  const { name } = req.body;  if (!req.files || Object.keys(req.files).length === 0) {  return res.status(400).send('Файл не выбран');  }    const file = req.files.file;  const fileName = name;    file.mv(${\_\_dirname}/../public/static/${name}, err => {  if (err) {  console.log(err);  return res.status(500).send('Ошибка при загрузке файла');  }    res.json({ fileName: fileName, filePath: /static/${name} });  });  });  module.exports = Route;  const Route = require('express').Router();  const genreController = require('../Controller/genreController');  const checkRole = require('../middleware/checkRoleMiddleware');  Route.post('/', checkRole('ADMIN'), genreController.createGenre);  Route.delete('/:id', checkRole('ADMIN'), genreController.deleteGenreById);  Route.get('/:page', genreController.getGenres);  Route.get('/book/:id', genreController.getBookGenres);  Route.delete('/book/:id/:genre', checkRole('ADMIN'), genreController.deleteGenre);  Route.post('/book/:id', checkRole('ADMIN'), genreController.addGenre);  module.exports = Route;  const Route = require('express').Router();  const UserRoute = require('./usersRoute');  const BookRoute = require('./booksRoute');  const FeedbackRoute = require('./feedbackRoute');  const BookmarkRoute = require('./bookmarksRoute');  const fileRoute = require('./filesRoute');  const tagRoute = require('./tagRoute');  const genreRoute = require('./genreRoute');  const authorRoute = require('./authorRoute');  const path = require('path');  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);  const Route = require('express').Router();  const tagController = require('../Controller/tagController');  const checkRole = require('../middleware/checkRoleMiddleware');  Route.get('/:page', tagController.getTags)  Route.post('/book/:id', checkRole('ADMIN'), tagController.addTag);  Route.delete('/book/:id/:tag', checkRole('ADMIN'), tagController.deleteTag);  Route.get('/book/:id', tagController.getTagsByBook);  Route.post('/',checkRole('ADMIN'), tagController.create);  Route.delete('/:id', checkRole('ADMIN'), tagController.delete);  module.exports = Route;  module.exports = Route;  const Route = require('express').Router();  const userController = require('../Controller/usersController');  const authMiddleware = require('../middleware/authMiddleware');  Route.post('/registration', userController.registration);  Route.post('/login', userController.login);  Route.get('/auth', authMiddleware, userController.check);  Route.get('/:page', userController.getUsers);  Route.get('/id/:id', userController.getUsersById);  Route.get('/authorized/:id', userController.getUserAuthorized);  Route.get('/email/:email', userController.getUsersByEmail);  //Изменение данных пользователя  Route.put('/change/:id', authMiddleware, userController.update);  //Удаление пользователя  Route.delete('/:id', authMiddleware, userController.delete);  module.exports = Route; |

Файл controller.serivce.json