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»

Специализация Программирование интернет-приложений

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

**К КУРСОВОМУ ПРОЕКТУ НА ТЕМУ:**

WEB-приложение «Библиотека»

Выполнил студент Пицуха Ярослав Анатольевич

(Ф.И.О.)

Руководитель проекта ассистент Бурмакова А.В.

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

Заведующий кафедрой к.т.н., доц. Смелов В.В.

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

Консультант: ассистент Бурмакова А.В.

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

Нормоконтролер: ассистент Бурмакова А.В.

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

Курсовой проект защищен с оценкой

Минск 2020

**Реферат**

Пояснительная записка курсового проекта содержит 34 страниц пояснительной записки, 16 иллюстрации, 7 источников литературы, 4 приложения.

JAVASCRIPT, NODE.JS, EXPRESS.JS, MONGODB, MONGOOSE, VUE.JS, VUE ROUTER, MATERIALIZE CSS, MVC, REST API

Целью курсового проекта является создание веб-приложения, которое предназначено для контроля и ведения дел библиотекарями и удобного просмотра информации читателями.

В первой главе проводится аналитический обзор прототипов по тематике курсового проекта и содержится описание технологий, использованных во время выполнения проекта.

Вторая глава посвящена процессу проектирования системы.

В третьей главе описывается процесс разработки и принципы функционирования компонент проекта.

Четвёртая глава посвящена тестированию приложения, которое позволяет более подробно понять интерфейс программного средства.

В пятой главе описывается руководство пользователя

Содержание

[Введение 6](#_Toc41639910)

[1 Постановка задачи 7](#_Toc41639911)

[1.1 Обзор прототипов 7](#_Toc41639912)

[1.2 Анализ прототипов 9](#_Toc41639913)

[1.3 Описание используемых технологий 9](#_Toc41639914)

[2 Разработка архитектуры проекта 11](#_Toc41639915)

[2.1 Взаимосвязь всех компонентов 11](#_Toc41639916)

[2.2 Модель базы данных 12](#_Toc41639917)

[3 Работка программного средства 15](#_Toc41639918)

[3.1 Проектирование структуры сервера 15](#_Toc41639919)

[3.2 Проектирование приложения клиента 17](#_Toc41639920)

[3.3 Реализация технологии Fetch запросов 20](#_Toc41639921)

[4 Тестирование 21](#_Toc41639922)

[5 Руководство пользователя 23](#_Toc41639923)

[Заключение 26](#_Toc41639924)

[Список литературных источников 27](#_Toc41639925)

[Приложение А 28](#_Toc41639926)

[Приложение Б 30](#_Toc41639927)

[Приложение В 31](#_Toc41639928)

[Приложение Г 33](#_Toc41639929)

# Введение

Библиотека всё еще остаются популярным местом у людей, несмотря на развитие электронных носителей информации. С каждым годом информационные технологии расширяют свой диапазон применения. В этой ситуации важно помогать развивать сферы, придумывать новые решения и адаптировать их к новой реальности. Такой целью задался и я при выборе задач проекта.

Целью моего курсового проекта является разработка веб-приложения «Библиотека», которое поможет читателю следить за взятыми им книгами, а также просмотреть список имеющихся в наличии книг. Библиотекарь же сможет вести полный учёт взятых книг, пользователей и всех книг библиотеки, имеющихся в наличии.

Основными задачами курсовой работы являются:

* провести аналитический обзор литературы;
* разработать модель базы данных;
* научиться разрабатывать приложение с использованием многоуровневой ахитектуры;
* провести тестирование;
* написать руководство пользователя.

Node.js – программная платформа для разработки серверных web-приложений на языке JS/V8.

В соответствии с заданием курсового проекта для проектирования базы данных используется система управления базами данных MongoDB с открытым исходным кодом, не требующая описания схемы таблиц. Классифицирована как NoSQL, использует JSON-подобные документы и схему базы данных.

Mongoose представляет специальную ODM-библиотеку (Object Data Modelling) для работы с MongoDB, которая позволяет определять объекты со строго-типизированной схемой, соответствующей документу MongoDB. Это библиотека, часто используемая в приложении Node.js с базой данных MongoDB.

Vue.js это [JavaScript](https://ru.wikipedia.org/wiki/JavaScript)-[фреймворк](https://ru.wikipedia.org/wiki/%D0%A4%D1%80%D0%B5%D0%B9%D0%BC%D0%B2%D0%BE%D1%80%D0%BA) с [открытым исходным кодом](https://ru.wikipedia.org/wiki/%D0%9E%D1%82%D0%BA%D1%80%D1%8B%D1%82%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B5_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D0%B5) для создания [пользовательских интерфейсов](https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D1%82%D0%B5%D1%80%D1%84%D0%B5%D0%B9%D1%81_%D0%BF%D0%BE%D0%BB%D1%8C%D0%B7%D0%BE%D0%B2%D0%B0%D1%82%D0%B5%D0%BB%D1%8F). Легко интегрируется в проекты с использованием других [JavaScript-библиотек](https://ru.wikipedia.org/wiki/%D0%91%D0%B8%D0%B1%D0%BB%D0%B8%D0%BE%D1%82%D0%B5%D0%BA%D0%B0_JavaScript" \o "Библиотека JavaScript). Может функционировать как [веб-фреймворк](https://ru.wikipedia.org/wiki/%D0%9A%D0%B0%D1%80%D0%BA%D0%B0%D1%81_%D0%B2%D0%B5%D0%B1-%D0%BF%D1%80%D0%B8%D0%BB%D0%BE%D0%B6%D0%B5%D0%BD%D0%B8%D0%B9) для разработки [одностраничных приложений](https://ru.wikipedia.org/wiki/%D0%9E%D0%B4%D0%BD%D0%BE%D1%81%D1%82%D1%80%D0%B0%D0%BD%D0%B8%D1%87%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%B8%D0%BB%D0%BE%D0%B6%D0%B5%D0%BD%D0%B8%D0%B5) в [реактивном](https://ru.wikipedia.org/wiki/%D0%A0%D0%B5%D0%B0%D0%BA%D1%82%D0%B8%D0%B2%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5) стиле.

Material Design, созданный Google, является языком дизайна, который сочетает в себе классические принципы успешного дизайна, а также инновации и технологии. Цель Google состоит в том, чтобы разработать систему дизайна, которая учитывает унифицированный пользовательский опыт для всех их продуктов на любой платформе.

# Постановка задачи

В соответствии с заданием курсового проекта следует разработать web-приложение, которое должно реализовывать свои функции, а именно позволять библиотекарю вести учёт взятых читателями книг. Для того чтобы сформировать окончательные требования к проектируемому программному средству сначала рассмотрим прототипы программных средств того же направления.

## 1.1 Обзор прототипов

На сегодняшний день существует не так много современных приложений для библиотек. Рассмотрим примеры web-приложений подобных сайтов.

1С:Библиотека [1]

Одним из найденных мною сайтов является 1С:Библиотека. На этом сайте можно посмотреть архив книг, прочитать информацию об определённом авторе и импортировать список книг из файла.

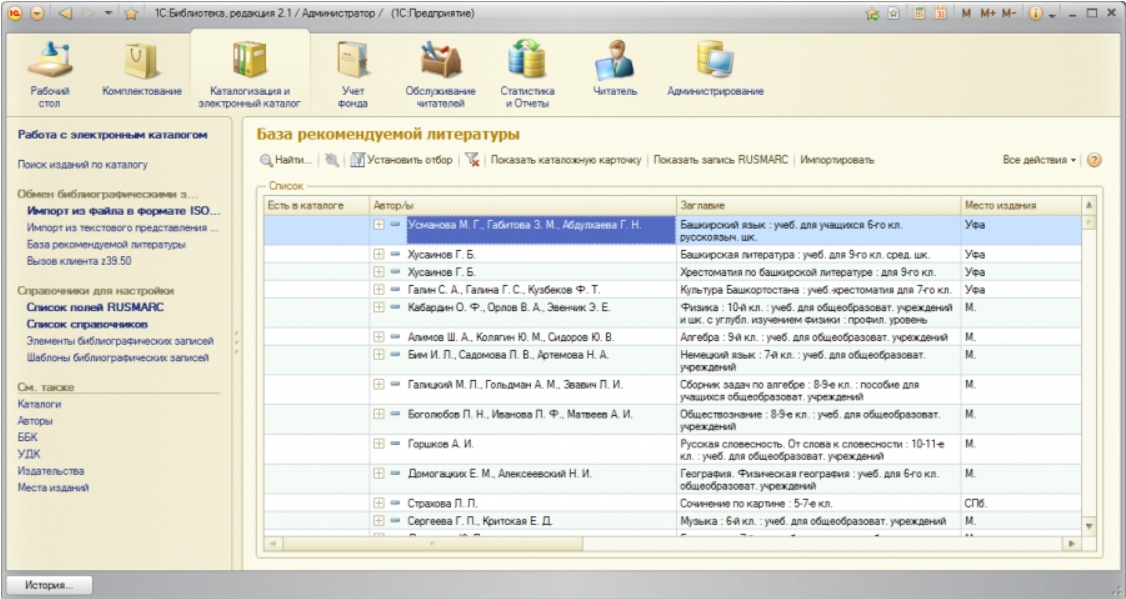


Рисунок 1.1 – Рейтинг сериалов сайта 1С:Библиотека

Также можно получить различные отчёты о деятельности библиотеки. Имеется страница читателя. Можно составлять базы рекомендованной литературы.

В целом неплохая программа.

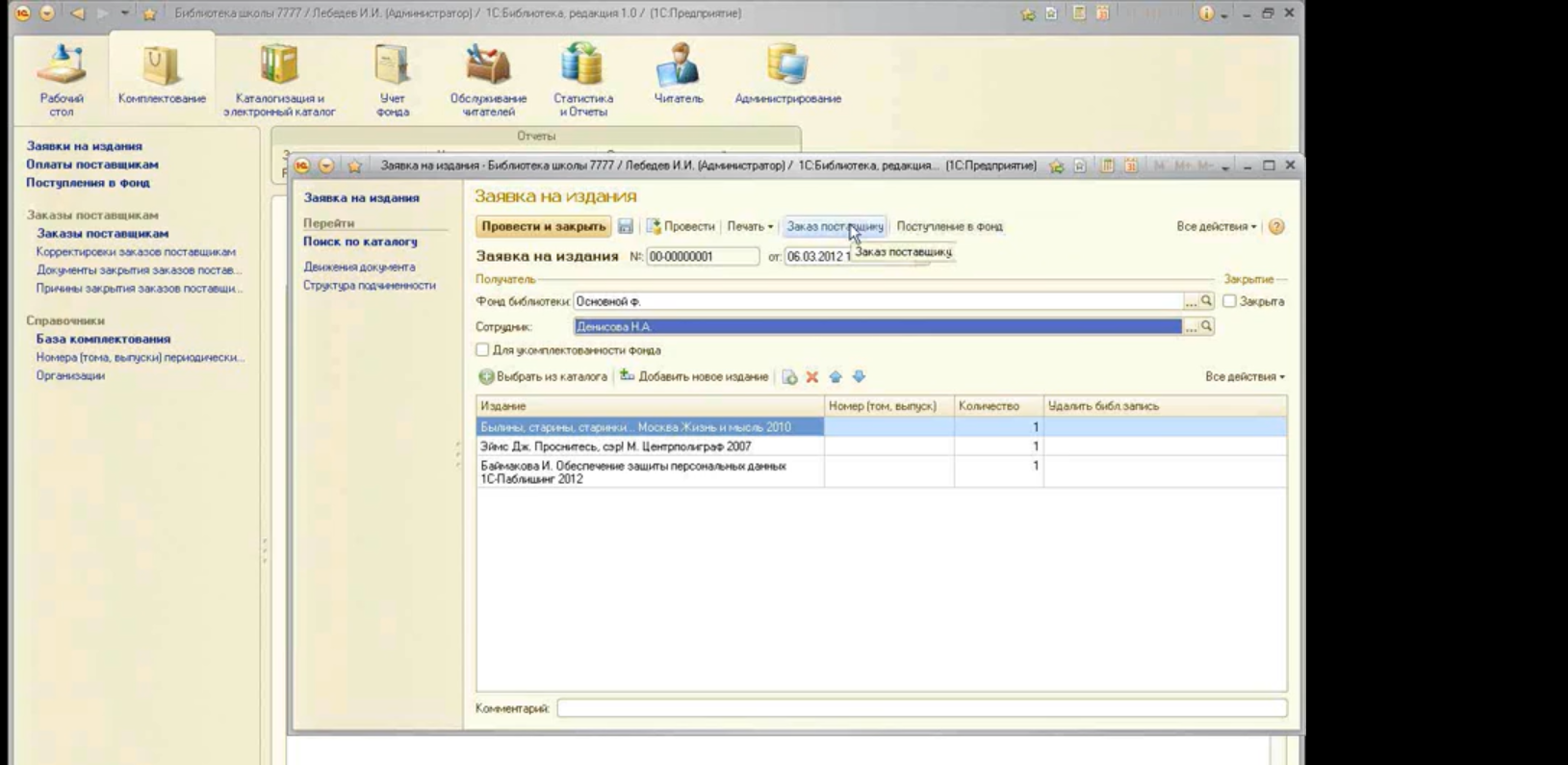


Рисунок 1.2 – Заказ на издание приложения 1С:Библиотека

МАРК Сloud [2]

Данный сайт является достаточно сложным, но многофункциональным. Как можно увидеть регистрация и авторизация хорошо продуманы, хотя не всё интуитивно понятно с первого взгляда.

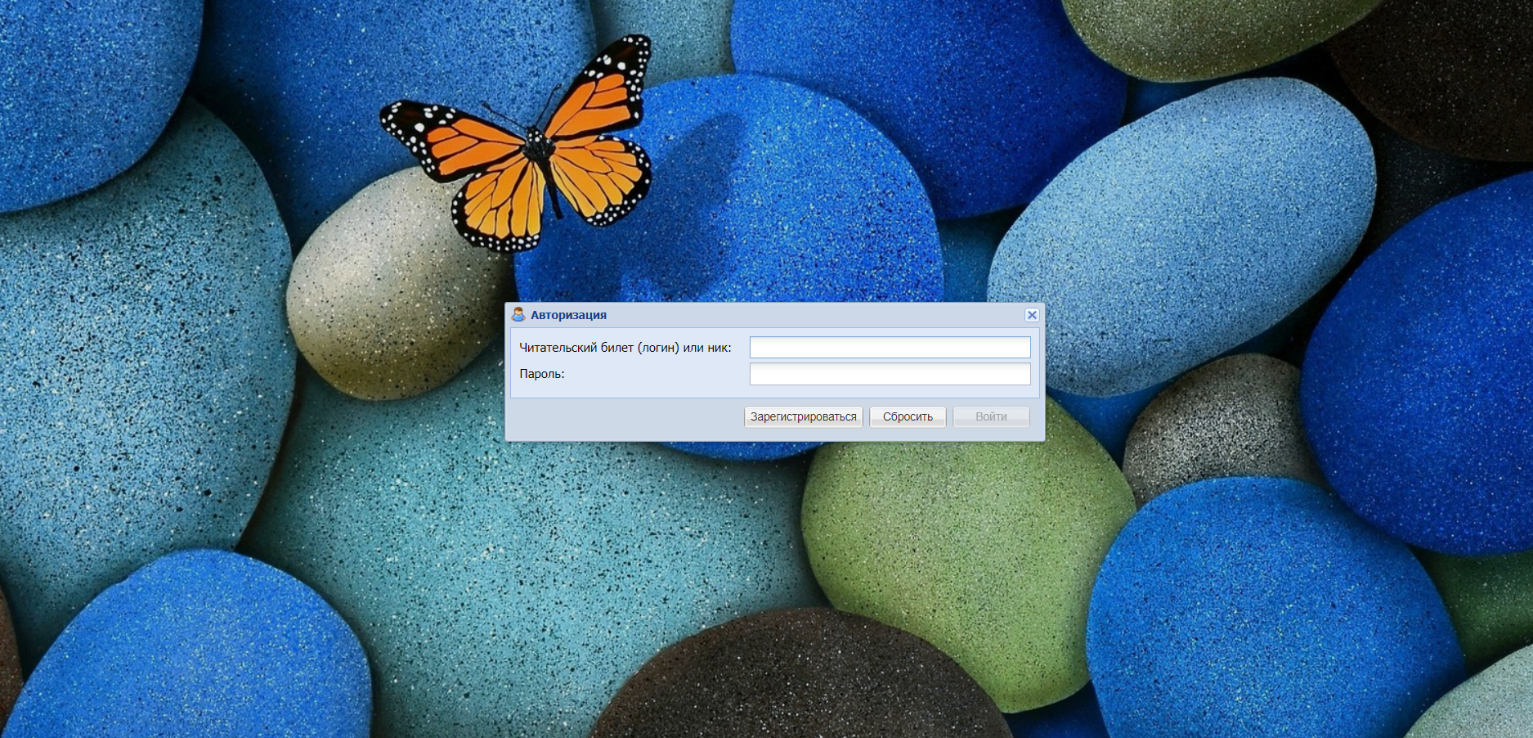


Рисунок 1.3 – Окно авторизации на сайте МАРК Сloud

Также можно, как и в ранее представленном приложении посмотреть информацию о книгах и сделать заказ.

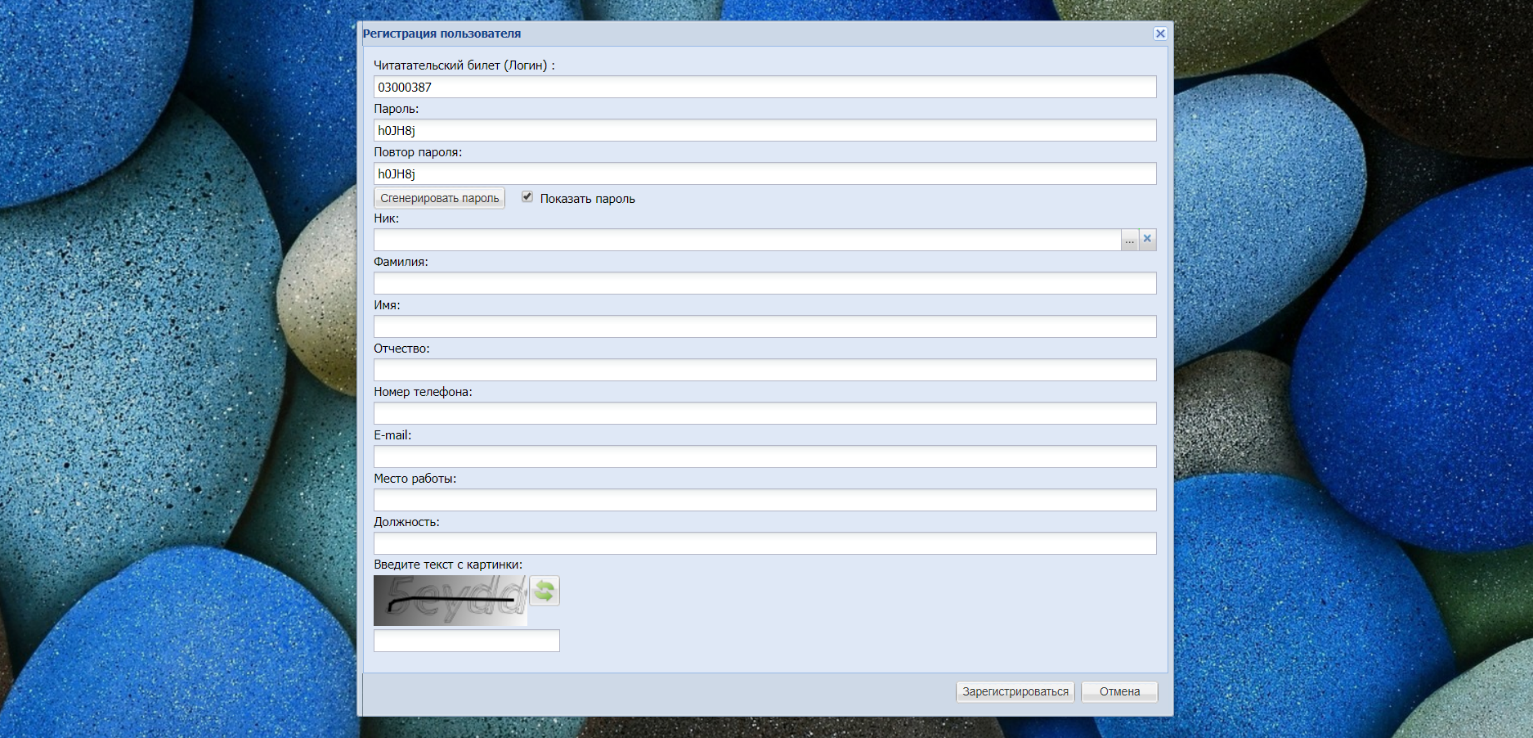


Рисунок 1.4 – Окно регистрации на сайте МАРК Сloud

**1.2 Анализ прототипов**

Проанализировав прототипы сайтов, можно отметить ожидаемую схожесть в их функциональности и интерфейсе.

Исходя из полученных результатов, можно сформулировать основные функциональные требования курсовой работы:

* добавление новых книг в каталог;
* возможность оформить выдачу книг библиотекарем;
* удаление книг;
* добавление новых читателей;
* редактирование информации о выдаче книги;
* возможность читателям самим создавать себе аккаунты;

Данные требования будут реализованы в веб-приложении.

**1.3 Описание используемых технологий**

Node.js – программная платформа для разработки серверных web-приложений на основе движка JavaScript Chrome V8. Является средой исполнения приложений на JavaScript. Также ориентирована на события, поддерживает асинхронность и является однопоточной. Также node.js является открытым проектом. [3]

Express, который является популярным веб-фреймворком, написанным на JavaScript и работающий внутри среды исполнения node.js. Express использует модуль http, а также в него включены готовые абстракции для упрощения создания сервера и серверной логики. Express не мешает общей производительности приложения т.к. представляет собой тонкий слой основных функций веб-приложений. Также в нём удобно настраивать маршруты приложения, поскольку он использует уже давно известные нам методы http.

Также приложение будет построено таким образом, чтобы соответствовать архитектуре MVC. Оно будет разделено на три части Model, View, Controller. Такая архитектура используется для того, чтобы упростить большой по объёму код. MVC применима к разным видам приложений: серверным веб-приложениям, десктопным приложениям.

Для проектирования базы данных будет использоваться система управления базами данных MongoDB. Она имеет открытый исходный код, не требует описания схемы таблиц. Также MongoDB является NoSQL базой данных, т.е. не реляционной базой данных.[4]

Mongoose представляет специальную ODM-библиотеку (Object Data Modelling) для работы с MongoDB, которая позволяет определять объекты со строго-типизированной схемой, соответствующей документу MongoDB. Это библиотека, часто используемая в приложении Node.js с базой данных MongoDB.

Для представлений в приложении будет использоваться веб-фреймворк Vue.js, он предоставляет удобный и понятный архитектурный стиль для создания клиентской части приложения.

Для получения данных с сервера на клиенте в приложении будет использоваться метод Fetch. Он является подходом для интерфейсов веб-приложений при котором осуществляется фоновый обмен данными между сервером и клиентом. При использовании асинхронного Javascript страница не перезагружается при обновлении данных.

Также в приложении будет использовано расширение протокола HTTP – HTTPS. Оно будет применено для повышения безопасности передачи данных. Передача данных будет происходить поверх криптографического протокола SSL. Он использует асимметричную криптографию для аутентификации ключей обмена, симметричное шифрование для сохранения конфиденциальности, коды аутентификации сообщений для целостности сообщений.

Для красивого вида страницы будет использован CSS-фреймвор Materialize CSS.

Протокол WebSocket тоже будет реализован в приложении. WebSocket является абстракцией над протоколом TCP и используется для обмена данными между клиентом и сервером в режиме реального времени.

# Разработка архитектуры проекта

Разработка архитектуры проекта –важная задач в процессе работы над приложением, потому что в зависимости от неё определяется уровень зависимости компонентами приложения, и насколько легко можно будет его расширить.

## Взаимосвязь всех компонентов

При разработке курсового проекта весь процесс был разбит на 3 следующие составляющие:

- база данных;

- сервер;

- клиент.

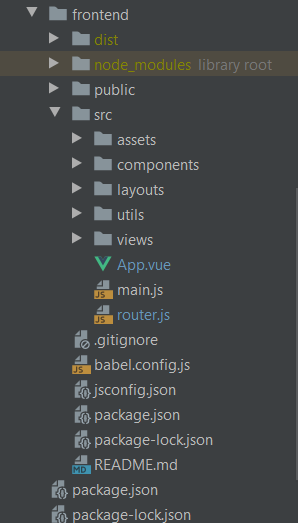
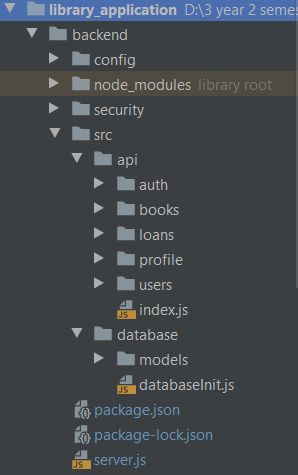


Рисунок 2.1 – Структура программного средства

Поскольку в проекте мы используем архитектуру MVC, то проект будет разделён на три части: Model, View, Controller. Структура программного средства представлена на рисунке 2.1.

Как мы можем увидеть в проекте имеется две главных директории, это Frontend и Backend.

Директорий database в backend содержит в себе файлы, которые описывают данные, а также логику управления этими данными.

Директорий frontend предназначен для хранения файлов представлений, там находится вся визуальная часть приложения.

Директорий api содержит файлы, которые отвечают за обработку логики запроса пользователя.

Также есть много дополнительных директорий, в котором содержится дополнительная информация необходимая для работы основных компонентов программы. Например, стили на клиентской части приложения отделены от разметки и обработку событий страницы.

Одной из целей разработки архитектуры было разделить как можно на большие отдельные и независимые части нашу программу, при этом что бы структура оставалась интуитивно понятной.

Директорий с сертификатами, которые нужны для реализации HTTPS.

## Модель базы данных

Для работы с базой данных было принято решение использовать ODM-библиотеку Mongoose.[5] Object data model представляет собой модель данных на основе объектно-ориентированного программирования, связывая методы (процедуры) с объектами, которые могут находиться в иерархическом порядке. Таким образом, объекты – это уровни абстракции, которые включают атрибуты и поведение.

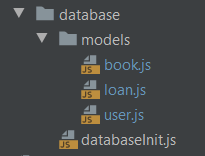


Рисунок 2.2 – Директорий с файлами модели данных

Коллекция Book содержит в себе следующие ключи:

id – идентификатор книги, тип ObjectId;

isbn – международный код книги, тип String;

author – автор книги, тип String;

numberOfCopies – количество копий книги в наличии, тип Number;

status – статус книги, тип String.

Коллекция User содержит в себе следующие ключи:

id – идентификатор пользователя, тип ObjectId;

firstName – Имя пользователя, тип String;

lastName – фамилия пользователя, тип String;

phoneNumber – номер телефона пользователя, тип String;

email – электронная почта пользователя, тип String;

password – пароль пользователя, тип String;

disabled – отключён ли пользователь, тип Bollean;

role – роль пользователя, тип String;

loans – массив ссылок на взятые книги пользователем, тип ObjectId.

Коллекция Loan содержит в себе следующие ключи:

id – идентификатор выдачи, тип ObjectId;

book – ссылка на выданную книгу, тип ObjectId;

member – ссылка на читателя, тип ObjectId;

issueDate – дата выдачи книги, тип Date;

dueDate – срок сдачи книги, тип Date;

returnDate – реальный срок сдачи книги, тип Date;

status – статус книги, тип String.

Чтобы работать с данными необходимо было разработать модель данных [4]. Листинг создания всех составляющих модели данных представлен в приложении А.

Схема модели базы данных находится в приложении Б.

В процессе создания коллекций нужно было также их связать. В листинге 2.3 мы добавляем в коллекцию Loan ссылки на читателя и книгу.

**Листинг 2.3 – Ссылки в схеме колекций**

**book**: { **type**: Schema.**Types**.*ObjectId*, **ref**: **'Book'**, **required**: **true** },  
**member**: { **type**: Schema.**Types**.*ObjectId*, **ref**: **'User'**, **required**: **true** },

База данных к приложению подключается так, как показано в листинге 2.4. Также здесь представлено подключение модуля mongoose и обработка ошибки подключения к базе данных.

**Листинг 2.4 – Подключение базы данных**

**const mongoose** = *require*(**'mongoose'**);  
**const** config = *require*(**'../../config'**)();  
  
***module***.*exports* = **async** (req, res, next) => {  
 **try**{  
 **if** (!**mongoose**.**connection**.**readyState**) {  
 **await mongoose**.connect(config.database.**connection**,  
 {  
 **useNewUrlParser**: **true**,  
 **useFindAndModify**: **false**,  
 **useUnifiedTopology**: **true**,  
 **useCreateIndex**: **true** }).then(() => ***console***.log(**'MongoDB connected'**))  
 }  
 } **catch** (error) {  
 ***console***.error(**'Server Error'**, error.**message**);  
 res.sendStatus(500);  
 **return**;  
 }  
 **return** next();  
};

# Работка программного средства

При проектировании программного средства в качестве программной платформы для сервера был выбран Node.js. Также на стороне клиента имеется код на JavaScript и Vue.js.

## Проектирование структуры сервера

Маршрутизация в приложении организована с помощью роутеров. Для этого используется Express [6]. Ниже в листинге 3.1 представлен листинг добавления роутеров с учётом их расположения в директориях проекта. Также мы настраиваем приложение на использование роутеров по определённому пути.

**Листинг 3.1 – Обработка всех маршрутов связанных с книгами**

**const** {***check***} = *require*(**'express-validator'**);  
  
***module***.*exports* = (app) => {  
 app.get(  
 **'/books/get'**,  
 *require*(**'./booksGet'**)  
 );  
  
 app.post(  
 **'/books/filter'**,  
 *require*(**'./booksFilter'**)  
 );  
  
 app.post(  
 **'/books/add'**,  
 [  
 ***check***(**'isbn'**, **'Введите isbn'**).exists(),  
 ***check***(**'title'**, **'Введите название'**).exists(),  
 ***check***(**'author'**, **'Введите автора'**).exists(),  
 ***check***(**'numberOfCopies'**, **'Введите количество копий'**).exists()  
 ],  
 *require*(**'./booksAdd'**)  
 );  
  
 app.delete(  
 **'/books/delete'**,  
 [  
 ***check***(**'isbn'**, **'Введите isbn'**).exists()  
 ],  
 *require*(**'./booksDelete'**)  
 );  
};

В следующем листинге, представленном в 3.2 можно увидеть реализацию метода get роутера book/delete/.

Изначально запрос просто ищет все сериалы, но если в поисковой строке хотя бы частично введено название сериала или одна из границ даты публикации сериала, то запрос строит список сериалов по введённым критериям. После этого происходит перенаправление на страницу serials/index с заданными параметрами.

Как только сервер получает данный запрос, он прогружает в браузере страницу с использованием файла index.ejs. Где динамически заполняет её содержимое.

**Листинг 3.2 – Реализация роутера удаления книги**

**const Book** = *require*(**'../../database/models/book'**);  
**const** {***validationResult***} = *require*(**'express-validator'**);  
  
***module***.*exports* = **async** (req, res) => {  
 **try**{  
 **const** errors = ***validationResult***(req);  
  
 **if** (!errors.isEmpty()) {  
 **return** res.status(400).json({  
 **errors**: errors.array(),  
 **message**: **'Некорректные данные при удалении книги'** })  
 }  
  
 **const** {isbn} = req.**body**;  
  
 **await Book**.deleteOne({isbn});  
  
 res.status(200).json({**message**: **'Книга удалена'**})  
  
 }**catch** (e) {  
 res.status(500).json({**message**: **'Что-то пошло не так, попробуйте снова'**})  
 }  
};

Листинг файла loansAdd.js приведён в приложении Б. Остальные роутеры построены по аналогии.

Кроме того, к приложению нужно было подключить HTTPS. Для этого сначала были созданы сертификаты с использованием криптографической библиотеки OpenSSL, затем данный сертификат был дополнительно зарегистрирован. После этого был загружен модуль https и с помощью его и сертификатов был создан сервер, использующий HTTPS.

**Листинг 3.3 – Подключение HTTPS**

**var** options = {  
 **key**: fs.*readFileSync*(**'./security/key.pem'**),  
 **cert**: fs.*readFileSync*(**'./security/cert.pem'**)  
};  
  
***https***.*createServer*(options, ***api***).listen(PORT);

Веб-сокеты также являются одним из пунктов реализации приложения. Веб-сокеты — это протокол, который реализует более тесное взаимодействие между клиентом и сервером.

В листинге 3.4 показано подключение модуля ws для работы с веб сокетами. И прослушивание 8080 порта для организации взаимодействия клиента и сервера по данной технологии.

**Листинг 3.4 – Подключение веб-сокетов**

**const** wss = **new** WebSocket.**Server**({  
 **port**: 8080  
});  
  
wss.on(**'connection'**, ws => {  
 ws.on(**'message'**, message => {  
 ws.send(message);  
 });  
});

Следующий код обрабатывает метод delete. Происходит поиск жанра по идентификатору, который передаётся как параметр в строке запроса, и его удаление из базы данных. После этого сервер отправляет сообщение об успешном удалении клиенту через веб-сокеты. Если удаление жанра не прошло, то происходит перенаправление на главную страничку жанров.

Представлено в листинге 3.5.

## Проектирование приложения клиента

На стороне клиента используются представления, которые динамически конфигурируются с помощью движка Vue.js, который также является шаблонизатором.

**Листинг 3.5 – Создание экземпляра Vue**

**import** Vue **from 'vue'  
import** Vuelidate **from 'vuelidate'  
import** App **from './App.vue'  
import *router* from './router'  
import** messagePlugin **from '@/utils/message.plugin'  
import 'materialize-css/dist/js/materialize.min'**Vue.**config**.**productionTip** =  
 ***process***.**env**.**NODE\_ENV** === **'production'**;  
  
  
Vue.use(messagePlugin);  
Vue.use(Vuelidate);  
  
**new** Vue({  
 ***router***,  
 render: h => h(App),  
}).$mount(**'#app'**);

В приложении есть отдельные директории с представлениями для каждой страницы. Также представлены отдельно частичные представления, которые вставляются в другие и представление. Также имеются представления для задних фонов страниц.

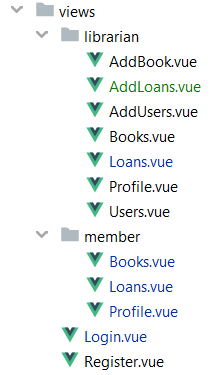


Рисунок 3.6 – Содержимое директории views

Также есть директорий public, который хранит в себе дополнительный функционал для клиентской части. В нём расположены некоторые стили.

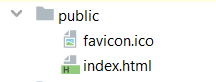


Рисунок 3.7 – Содержимое директория public

В листинге 3.8 представлен пример динамического формирования содержимого страницы.

**Листинг 3.8 – Пример динамического формирования содержимого страницы**

<**template**>  
 <**div id="app"**>  
 <**component :is="layout"**>  
  
 </**component**>  
 </**div**>  
</**template**>  
  
<**script**>  
 **import** EmptyLayout **from './layouts/EmptyLayout'**;  
 **import** LibrarianLayout **from "./layouts/LibrarianLayout"**;  
 **import** MemberLayout **from "./layouts/MemberLayout"**;  
  
 **export default** {  
 **computed**: {  
 layout() {  
 **return** (**this**.**$route**.**meta**.**layout** || **'empty'**) + **'-layout'** }  
 },  
 **components**: {  
 EmptyLayout, LibrarianLayout, MemberLayout  
 }  
 }  
</**script**>  
  
<**style lang="scss"**>  
 **@import '~materialize-css/dist/css/materialize.min.css'**;  
 **@import 'assets/index.css'**;  
</**style**>

В нижеприведённом листинге приведён код страницы заднего фона для других страниц.

**Листинг 3.9 – Пример кода заднего фона для страниц**

<**template**>  
 <**div class="app-main-layout"**>  
 <**Navbar @click="isOpen** = !**isOpen"**/>  
 <**Sidebar v-model="isOpen"**/>  
 <**main class="app-content" :class="**{**full**: !**isOpen**}**"**>  
 <**div class="app-page"**>  
 <**router-view**/>  
 </**div**>  
 </**main**>  
 </**div**>  
</**template**>  
  
<**script**>  
 **import** Navbar **from "@/components/app/librarian/Navbar"**;  
 **import** Sidebar **from "@/components/app/librarian/Sidebar"**;  
 **export default** {  
 **name**: **"LibrarianLayout"**,  
 data:() => ({  
 **isOpen**: **true** }),  
 **components**: {  
 Navbar, Sidebar  
 }  
 }  
</**script**>

## Реализация технологии запросов

Клиент и сервер обмениваются информацией по REST web API.

С помощью Fetch клиент обращается к серверу для получения каких либо данных. С помощью данной функции можно удобно получать данные с сервера на клиент.

На стороне сервера происходит проверка данных, отправленных клиентом, как и на клиента.

**Листинг 3.10 – Обработка fetch-запроса на стороне сервера**

app.post(  
 **'/loans/get\_by\_user'**,  
 [  
 ***check***(**'\_id'**, **'укажите \_id'**).exists()  
 ],  
 *require*(**'./loansGetByUser'**)  
);

Если клиенту приходит не успешный статус, то он должен вывести сообщение об ошибке, сформировав ещё дополнительный блок.

**Листинг 3.11 – Обработка fetch-запроса на стороне клиент**

**async** submitHandler() {  
 **if** (**this**.$v.$invalid) {  
 **this**.$v.$touch();  
 **return** }  
 **if** (**this**.**status** === **'нет в наличии'**) {  
 **this**.**status** = **'unavailable'** }  
 **else if** (**this**.**status** === **'есть в наличии'**) {  
 **this**.**status** = **'available'** }  
 **const** formData = {  
 **isbn**: **this**.**isbn**,  
 **title**: **this**.**title**,  
 **author**: **this**.**author**,  
 **status**: **this**.**status** };  
 **try** {  
 **const** responce = **await** requests.request(**'/api/books/filter'**, **'POST'**, formData);  
 **this**.**books** = responce.**books**;  
 **this**.**$message**(responce.**message**);  
 } **catch** (e) {  
 ***console***.log(e.**message**)  
 }  
}

# Тестирование

Тестирование приложения предназначено для выявления возможных ошибок при разработке и для того, чтобы исключить действия пользователя, которые могут повлечь за собой аварийное завершение программы.

Рассмотрим добавление библиотекарем новой книги с существующим ISBN.

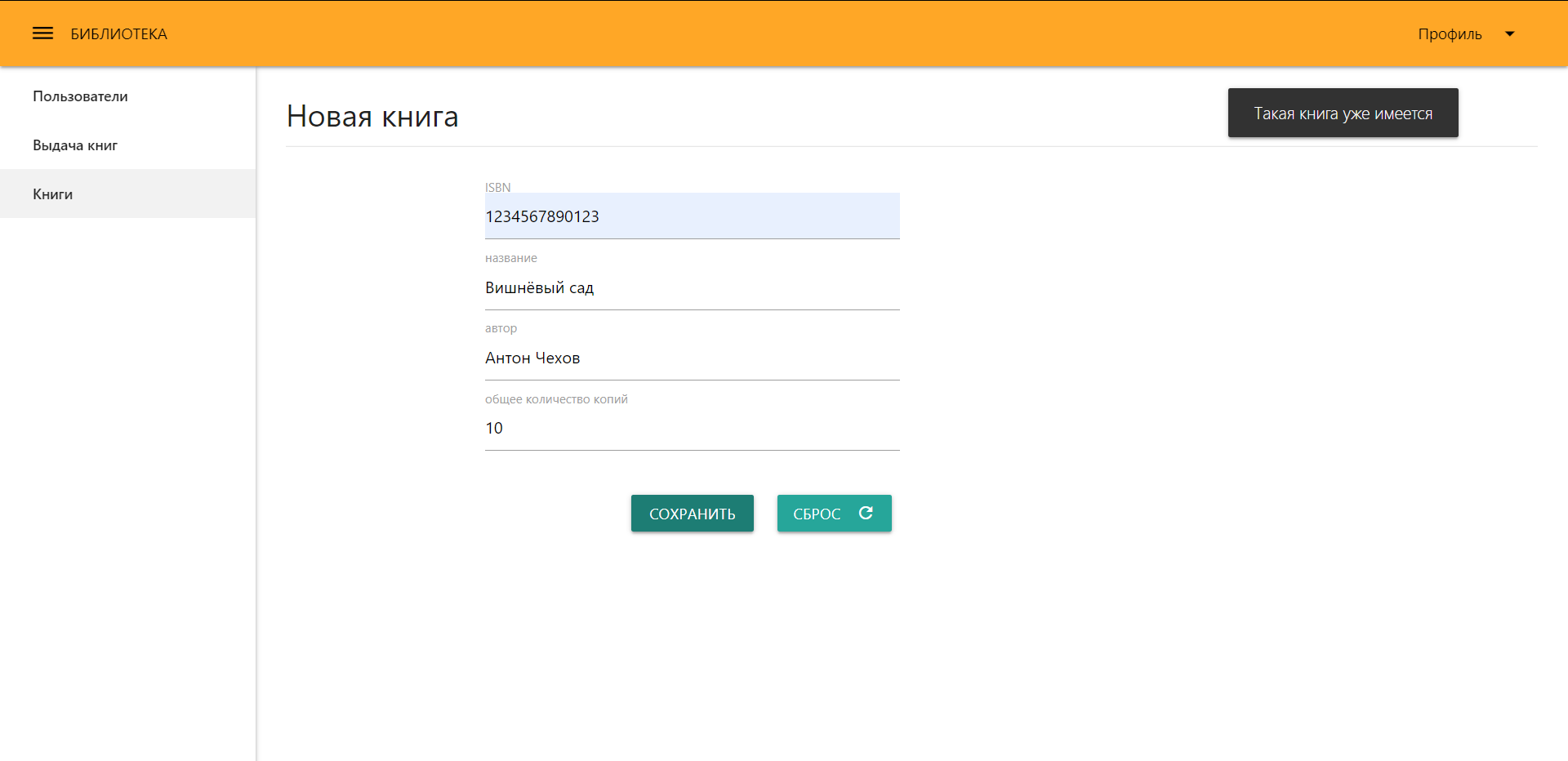


Рисунок 4.1 – Ошибка добавления книги с существующим ISBN

Кроме этого все выбранные поля также проверяются на корректное заполнение при фильтрации таблицы.

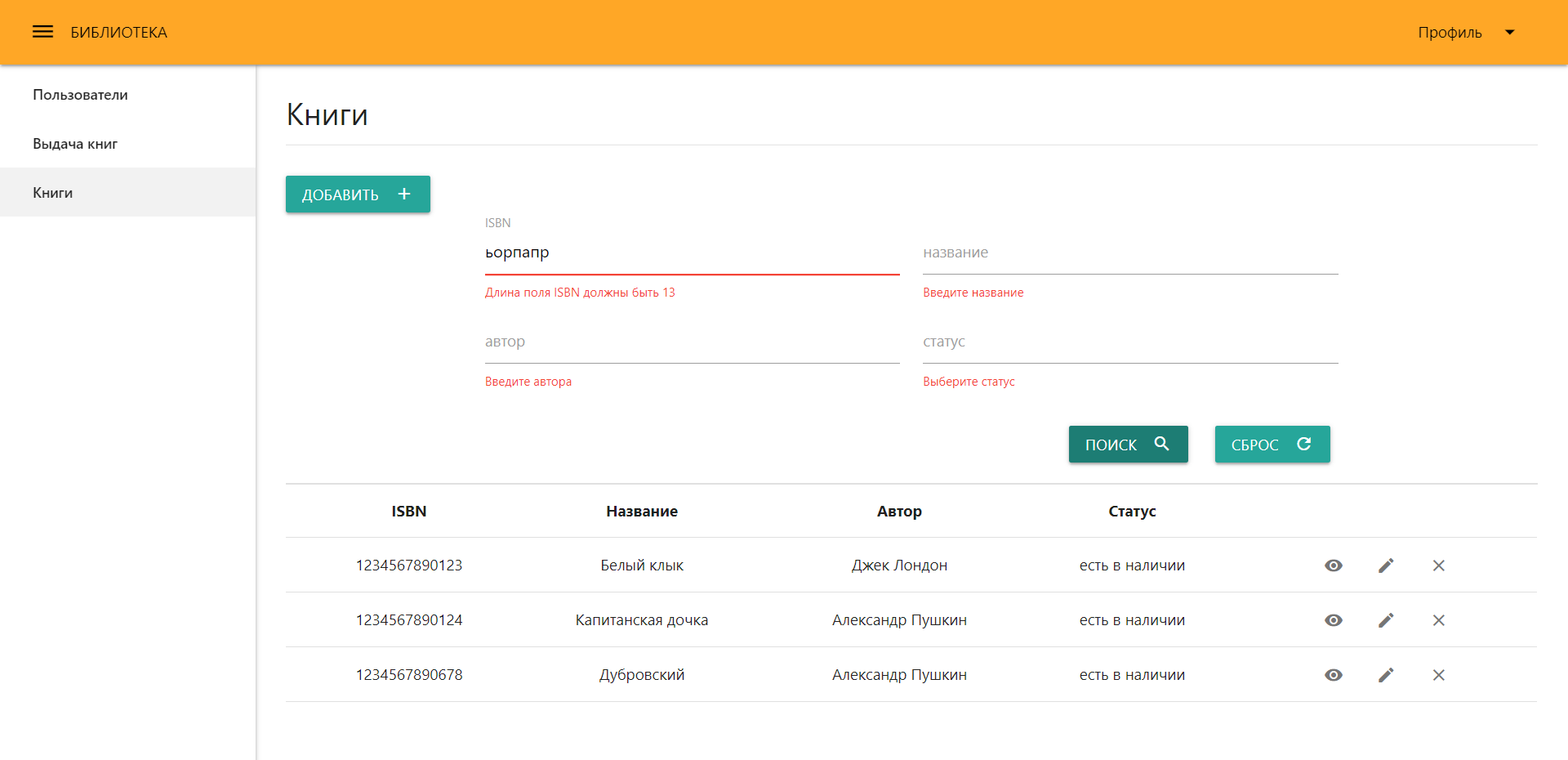


Рисунок 4.2 – Проверка корректности введённых данных при фильтрации

В добавлении пользователей проверяется корректность введённого email.

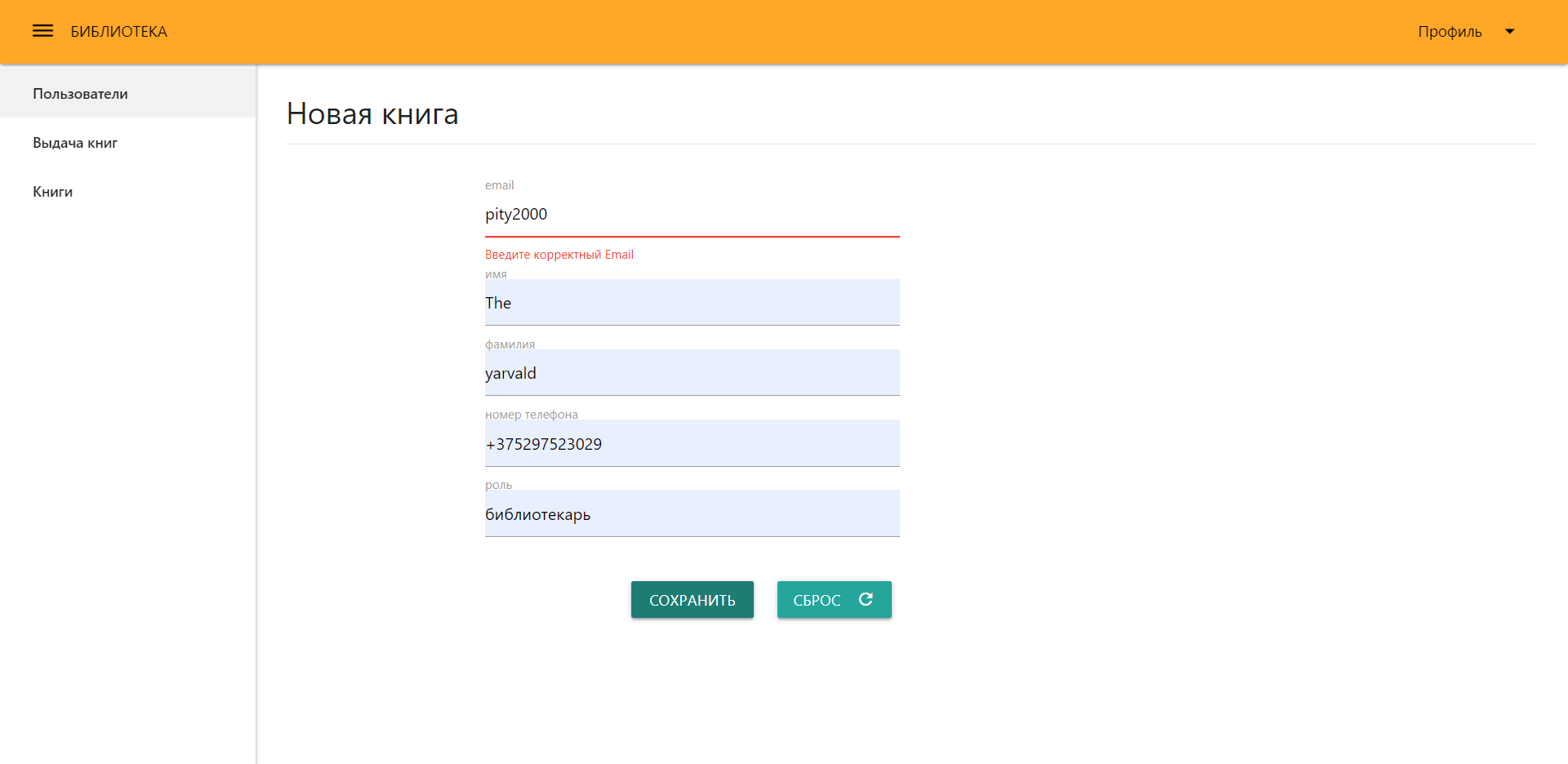


Рисунок 4.3 – Проверка email на корректность

# Руководство пользователя

Первое, что может сделать пользователь, когда он открывает сайт – это войти или зарегистрироваться в системе.

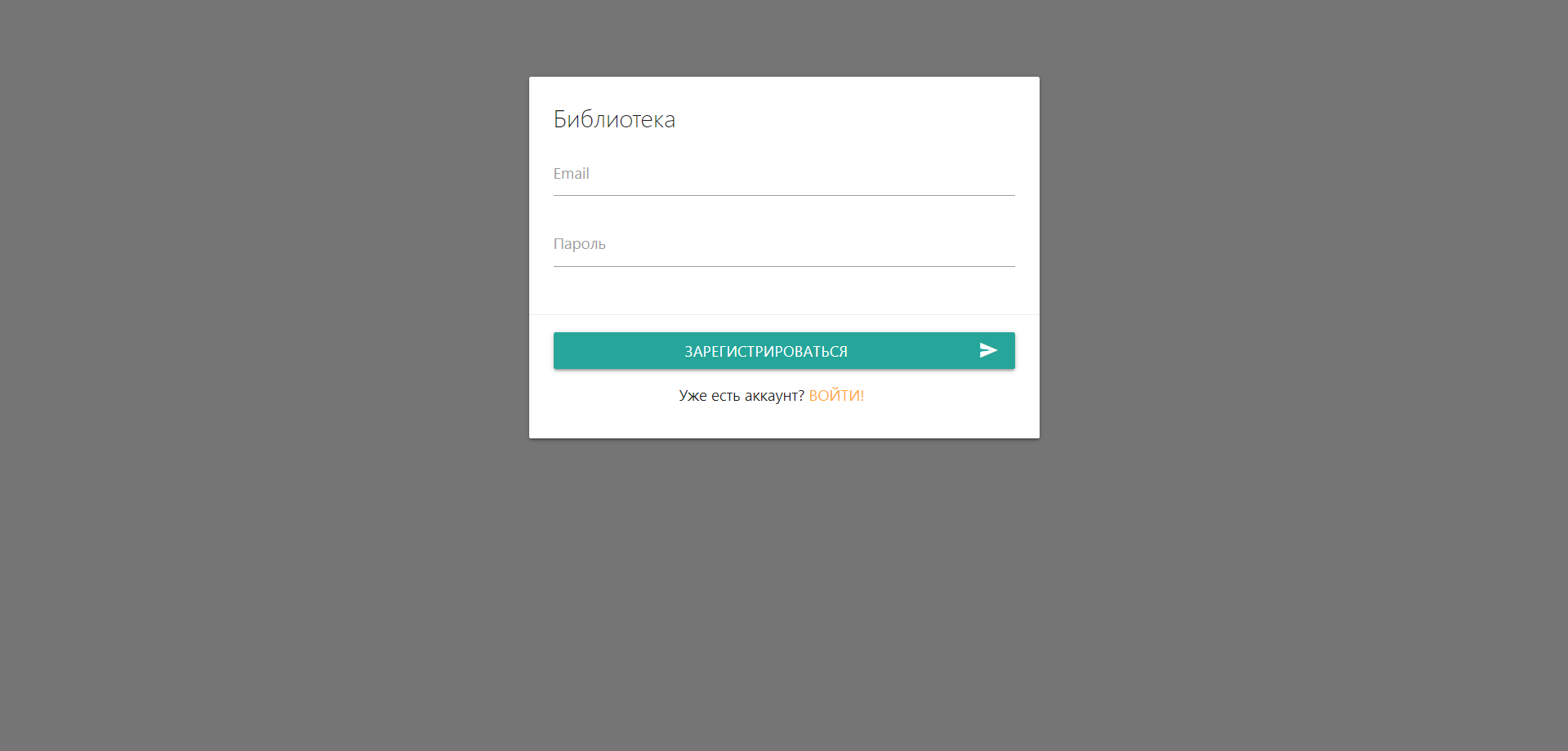


Рисунок 5.1 – Регистрация нового читателя

Далее в зависисости от роли ему предоставляются различные возможности. Основные конечно у библиотекаря. Читатель может только зайти к мебе в профиль, посмотреть смписок книг библиотеки и книги которые он взял.

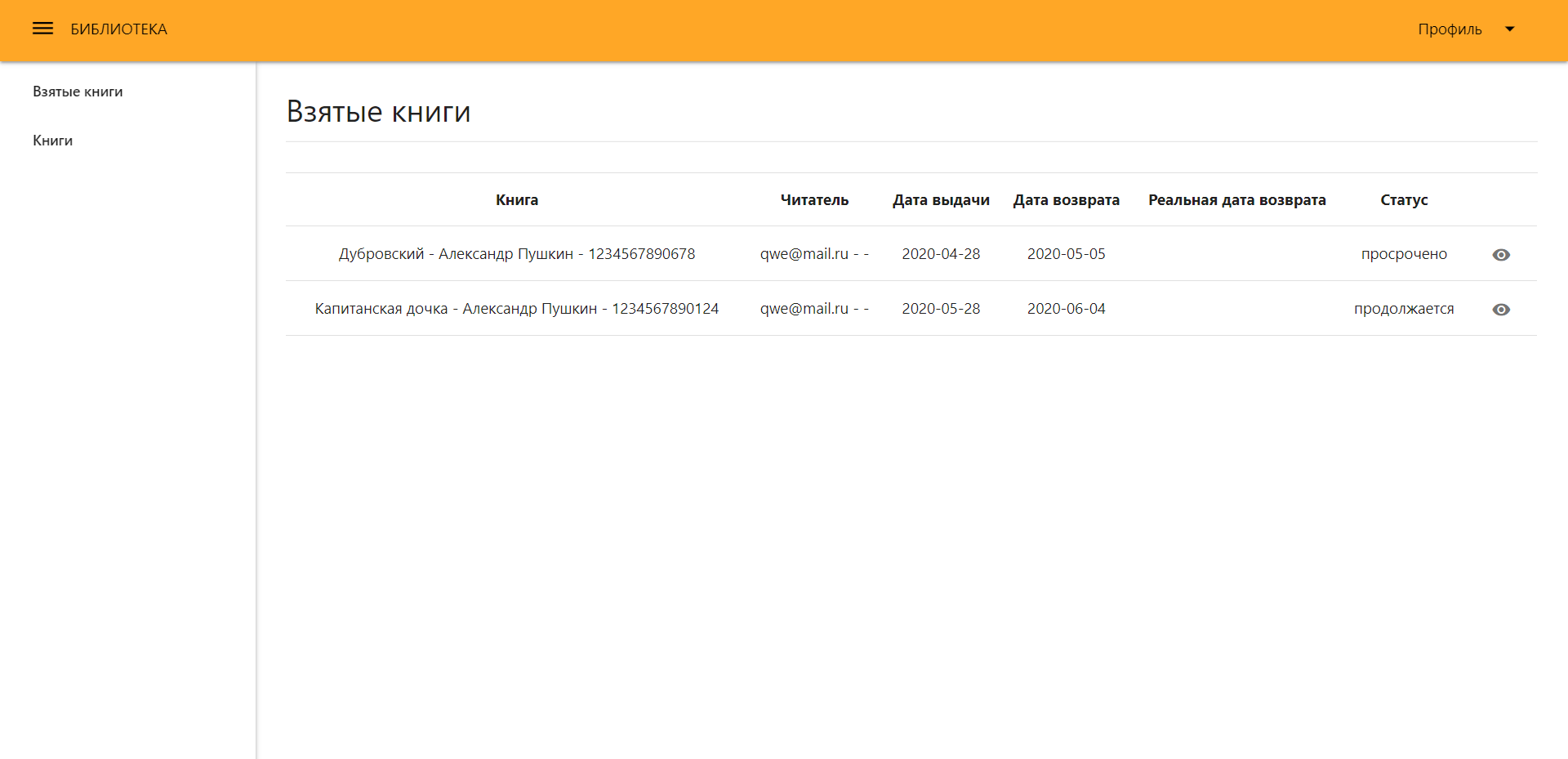


Рисунок 5.2 – Страница взятых книг читателя

Библиотекарь же может добавлять и удалять новых пользователей, а также увидеть весь список читателей.

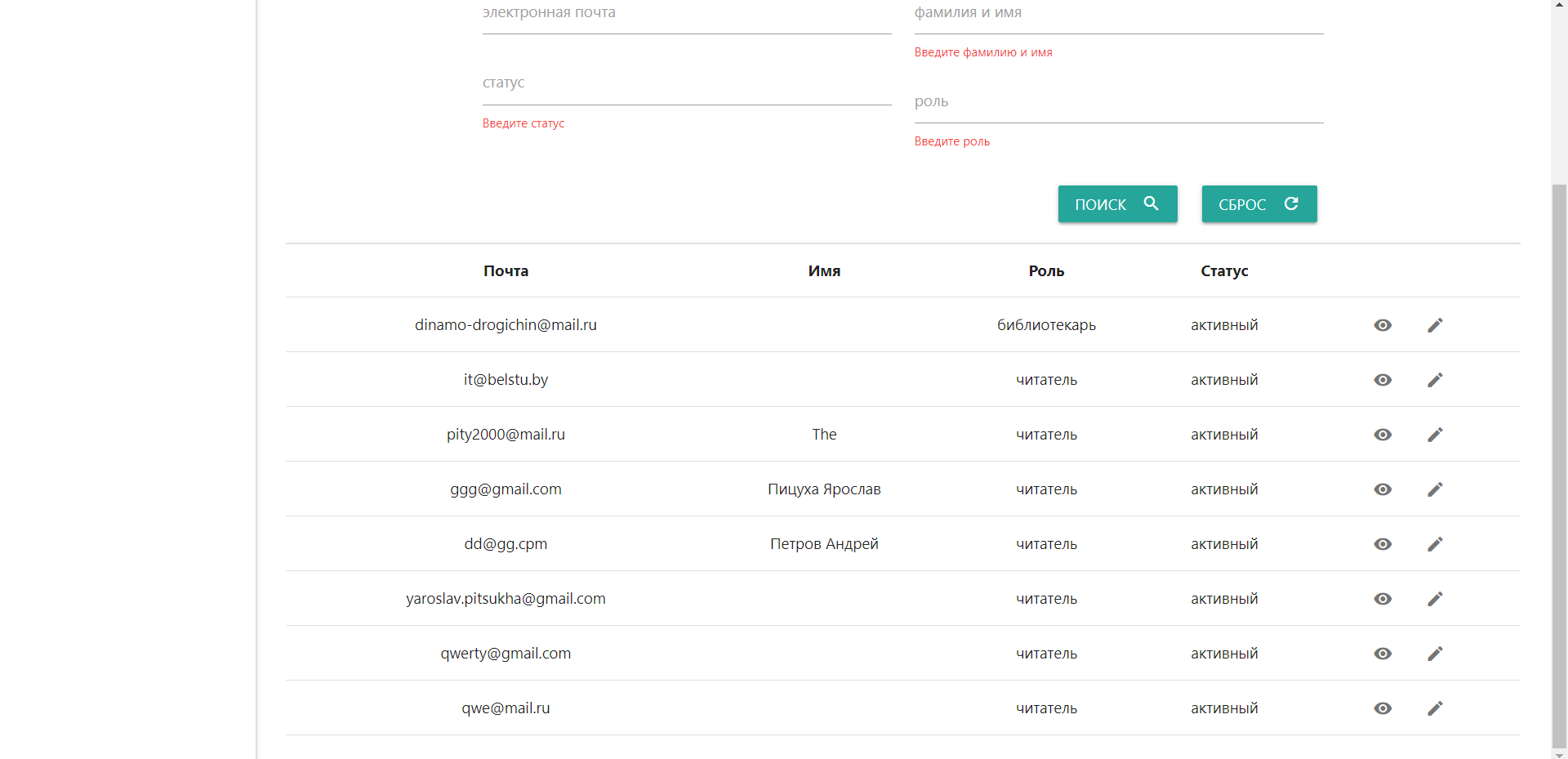


Рисунок 5.3 – Просмотр таблицы со всеми пользователями

Библиотекарь таже может добавлять, удалять и просматривать книги библиотеки.

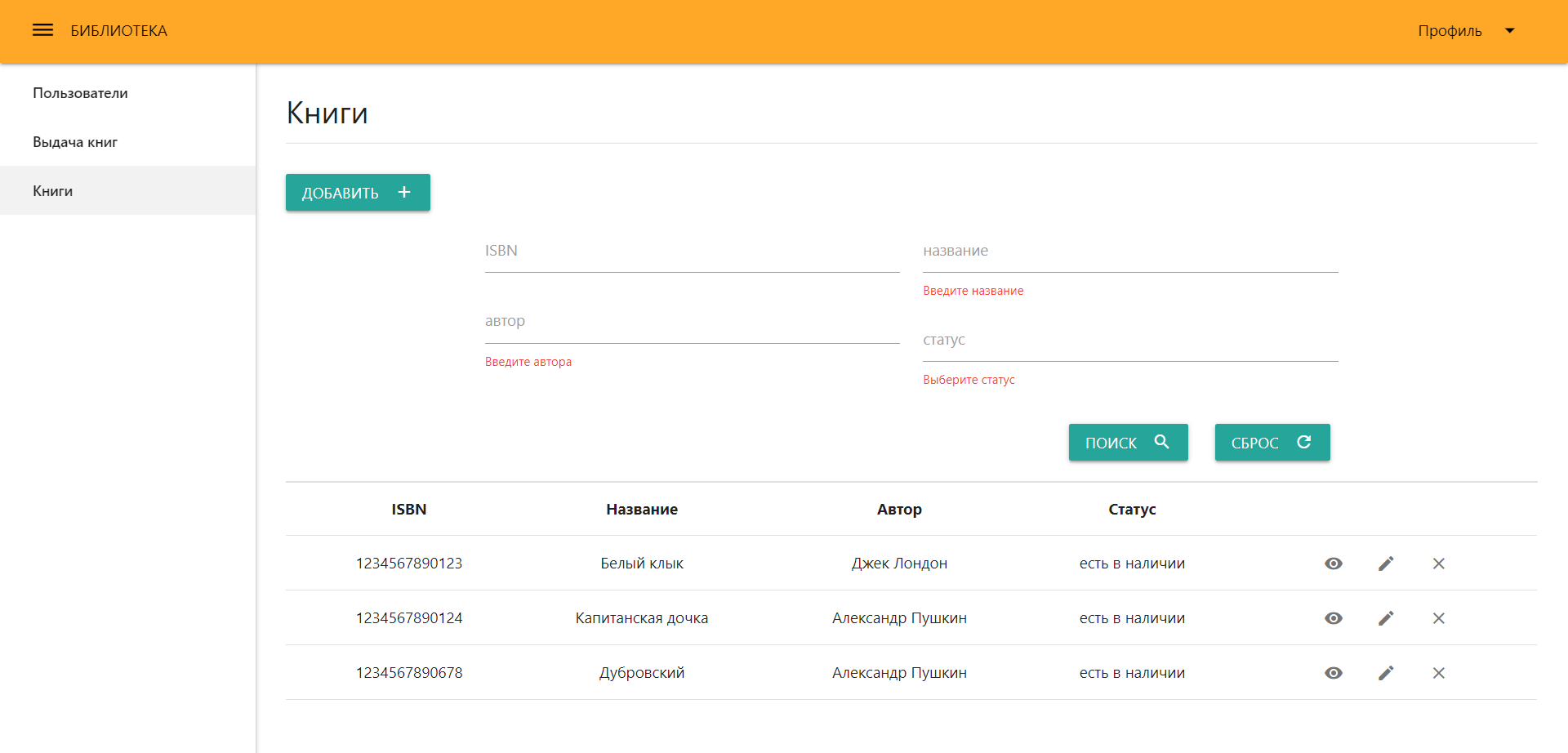


Рисунок 5.4 – Таблица с информацией о книгах библиотеки

И одна из самых важных функций которой может воспользоваться библиотекарь, это добавлять и удалять новые выдачи книг. Вся информация представлена в понятных таблицах. Также эти таблицы имеют форму для фильтрации.

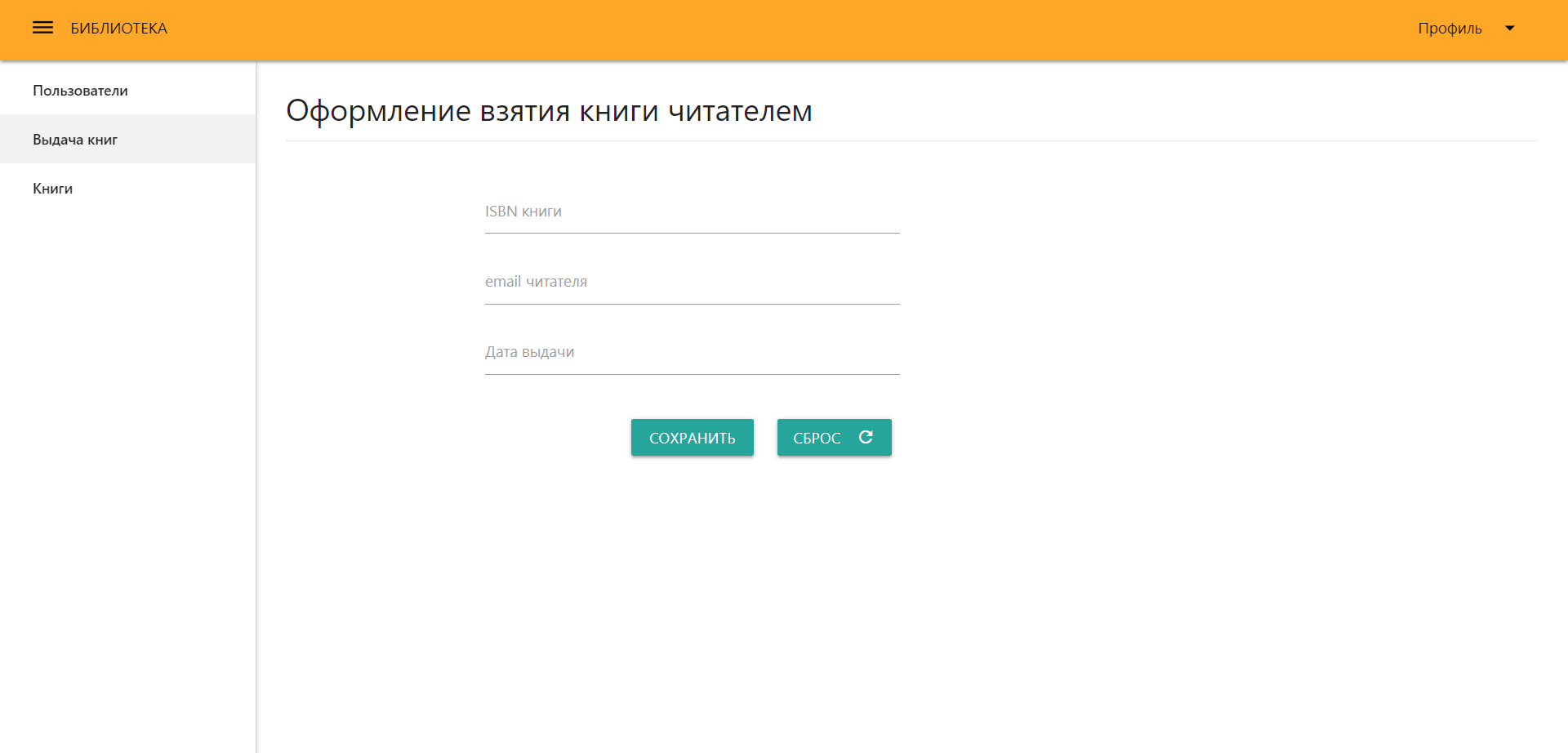


Рисунок 5.5 – Страница добавления новой выдачи книги

# Заключение

В данной курсовой работе было создано веб-приложение «Библиотека».

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

В процессе выполнения курсовой работы была спроектирована база данных для хранения в ней информации о сериалах. База данных была разработана с помощью системы управления базами данных «MongoDB». Также был создан сервер и клиент с пользовательским интерфейсом. Тестирование программного продукта было реализовано в том числе.

В результате написания курсового проекта, было разработано приложение, на основе архитектуры MVC. Приложение написано с помощью языка программирования Node.js. Связь между базой данных и приложением осуществляется с помощью специальной ODM-библиотеки, которая позволяет определять объекты со строго-типизированной схемой, соответствующей документу MongoDB. Также было написано руководство пользователя для созданного приложения.

В приложении была реализована технология веб-сокетов для обмена сообщениями между клиентом и сервером. Технологи vuelidate также добавлена в проект для дополнительной валидации. Для повышения безопасности передачи данных в приложении используется HTTPS.

При разработке выполнены следующие пункты:

* просмотр списка книг;
* просмотр списка пользователей;
* добавление выдач книг;
* добавление книг;
* удаление книг;
* регистрация пользователей;
* работа с фильтрами таблиц;

В соответствии с полученным результатом, можно сказать, что разработанное приложение функционирует верно, требования технического задания реализованы в полном объеме, поэтому цель курсового проекта можно считать достигнутой.

**Список литературных источников**

1. Сайт «1С:Библиотека» [Электронный ресурс] – Режим доступа: [<https://solutions.1c.ru/catalog/library>/](https://myshows.me/) – Дата доступа: 04.05.2020.
2. Официальный сайт «МАРК Сloud» [Электронный ресурс] – Режим доступа: [https://test.informsystema.ru](https://test.informsystema.ru/) – Дата доступа: 04.05.2020.
3. About Node.js [Электронный ресурс] – Режим доступа: <https://nodejs.org/en/about/> – Дата доступа: 27.04.2020.
4. MongoDB [Электронный ресурс] – Режим доступа: <https://www.mongodb.com/> – Дата доступа: 04.05.2020
5. Mogoose [Электронный ресурс] – Режим доступа: <https://mongoosejs.com/>– Дата доступа: 01.05.2020.
6. Express [Электронный ресурс] – Режим доступа: <https://expressjs.com/ru/>– Дата доступа: 30.04.2020.
7. Express и MongoDB [Электронный ресурс] <https://metanit.com/web/nodejs/6.5.php> – Дата доступа: 01.05.2020.

# Приложение А

const database = require('mongoose');

const Schema = database.Schema;

const BookSchema = new Schema(

{

isbn: { type: String, required: true, minlength: 13, maxlength: 13, unique: true },

title: { type: String, required: true, maxlength: 255 },

author: { type: String, required: true, maxlength: 255 },

numberOfCopies: { type: Number, required: true, min: 1 },

status: { type: String, required: true, enum: [ "available", "unavailable"] },

createdBy: { type: Schema.Types.ObjectId, ref: 'User' },

updatedBy: { type: Schema.Types.ObjectId, ref: 'User' }

},

{ timestamps: true },

);

BookSchema.virtual('id').get(function() {

return this.\_id.toHexString();

});

BookSchema.set('toJSON', {

getters: true,

});

BookSchema.set('toObject', {

getters: true,

});

module.exports = database.model('Book', BookSchema);

const database = require('mongoose');

const Schema = database.Schema;

const LoanSchema = new Schema(

{

book: { type: Schema.Types.ObjectId, ref: 'Book', required: true },

member: { type: Schema.Types.ObjectId, ref: 'User', required: true },

issueDate: { type: Date, required: true },

dueDate: { type: Date, required: true },

returnDate: { type: Date },

status: { type: String, enum: [ "inProgress", "overdue", "closed" ] },

createdBy: { type: Schema.Types.ObjectId, ref: 'User' },

updatedBy: { type: Schema.Types.ObjectId, ref: 'User' }

},

{ timestamps: true },

);

LoanSchema.virtual('id').get(function() {

return this.\_id.toHexString();

});

LoanSchema.set('toJSON', {

getters: true,

});

LoanSchema.set('toObject', {

getters: true,

});

module.exports = database.model('Loan', LoanSchema);

const database = require('mongoose');

const Schema = database.Schema;

const UserSchema = new Schema(

{

firstName: { type: String, maxlength: 80 },

lastName: { type: String, maxlength: 175 },

phoneNumber: { type: String, maxlength: 24},

email: { type: String, maxlength: 255, required: true, unique: true },

password: { type: String, maxlength: 255, required: true },

disabled: { type: Boolean, default: false },

role: { type: String, required: true },

loans: [{ type: Schema.Types.ObjectID, ref: 'Loans' }],

createdBy: { type: Schema.Types.ObjectId, ref: 'User' },

updatedBy: { type: Schema.Types.ObjectId, ref: 'User' }

},

{ timestamps: true },

);

UserSchema.virtual('id').get(function () {

return this.\_id.toHexString();

});

UserSchema.set('toJSON', {

getters: true,

});

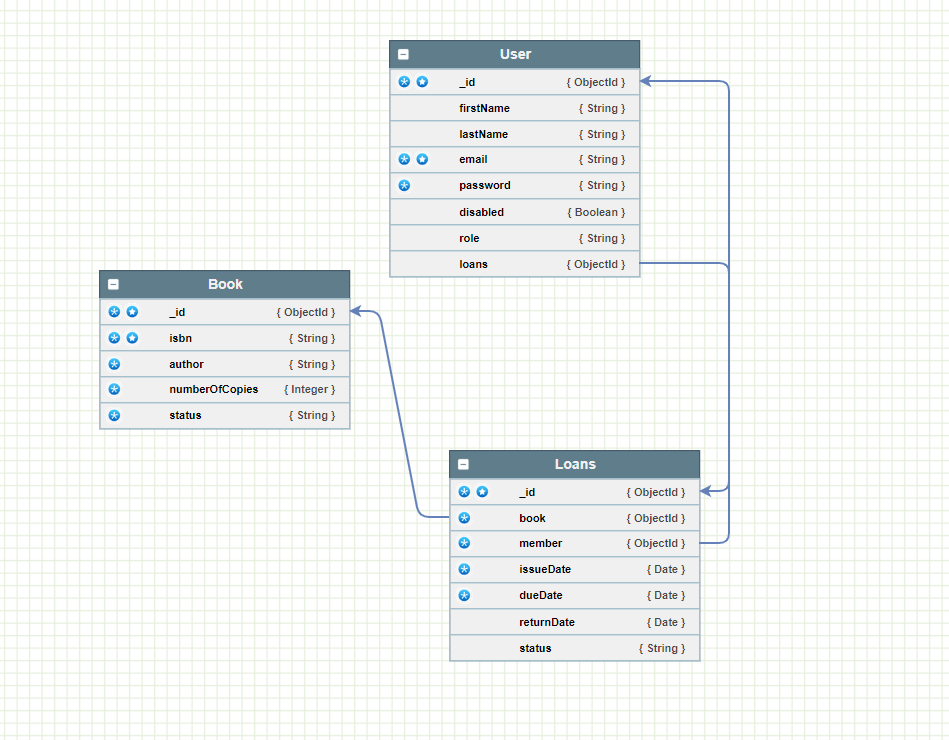
UserSchema.set('toObject', {

getters: true,

});

module.exports = database.model('User', UserSchema);

# Приложение Б



# Приложение В

const Book = require('../../database/models/book');

const User = require('../../database/models/user');

const Loan = require('../../database/models/loan');

const {validationResult} = require('express-validator');

module.exports = async (req, res) => {

try{

const errors = validationResult(req);

if (!errors.isEmpty()) {

return res.status(400).json({

errors: errors.array(),

message: 'Некорректные данные при добавлении книги'

})

}

let {isbn, email, issueDate} = req.body;

const dueDate = new Date(issueDate);

issueDate = new Date(issueDate);

const nowDate = new Date();

dueDate.setDate(dueDate.getDate() + 7);

let status = 'inProgress';

const book = await Book.findOne({isbn});

if (!book) {

return res.status(400).json({ message: 'Такой книги нет'})

}

if (book.numberOfCopies < 1 || book.status === "unavailable" ) {

return res.status(400).json({ message: 'Таких книг сейчас нет в наличии'})

}

const member = await User.findOne({email, role: 'member'});

if (!member) {

return res.status(400).json({ message: 'Такого читателя нет'})

}

const candidate = await Loan.findOne({book, member});

if (candidate) {

return res.status(400).json({ message: 'Этот читатель уже взял эту книгу'})

}

if ( nowDate >= issueDate && nowDate <= dueDate) {

status = 'inProgress';

}

else if (nowDate > dueDate) {

status = 'overdue';

}

loan = new Loan({book, member, issueDate, dueDate, status});

await loan.save();

const numberOfCopies = book.numberOfCopies;

const newNumberOfCopies = Number(numberOfCopies) - 1;

if (numberOfCopies === 1) {

book.numberOfCopies = newNumberOfCopies;

book.status = 'unavailable';

book.save();

}

else {

book.numberOfCopies = newNumberOfCopies;

book.save();

}

member.loans.push(loan);

member.save();

res.status(201).json({message: 'Данные о выдаче книги успешно занесены'})

}catch (e) {

res.status(500).json({message: 'Что-то пошло не так, попробуйте снова'})

}

};

const Loan = require('../../database/models/loan');

const {validationResult} = require('express-validator');

module.exports = async (req, res) => {

try{

const errors = validationResult(req);

if (!errors.isEmpty()) {

return res.status(400).json({

errors: errors.array(),

message: 'Некорректные данные при удалении книги'

})

}

const {\_id} = req.body;

await Loan.deleteOne({\_id});

res.status(200).json({message: 'Данные о выбранной выдаче книги удалены'})

}catch (e) {

res.status(500).json({message: 'Что-то пошло не так, попробуйте снова'})

}

};

**Приложение Г**

