МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ

ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет Информационных технологий

Кафедра Программной инженерии

Специальность 1-40 01 01 Программное обеспечение информационных технологий

Специализация 1-40 01 01 10 Программное обеспечение информационных

технологий (программирование интернет-изданий)

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

**К КУРСОВОМУ ПРОЕКТУ НА ТЕМУ:**

«WEB-приложение «Книжный магазин»»

Выполнил студент Пунько Алина Андреевна

(Ф.И.О.)

Руководитель проекта преп.-стаж. Дубовик М.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: преп.-стаж. Дубовик М.В.

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

Нормоконтролер: преп.-стаж. Дубовик М.В.

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

Курсовой проект защищен с оценкой

Минск 2020

**Оглавление**

[Введение 3](#_Toc41756177)

[1 Постановка задачи 4](#_Toc41756178)

[2 Проектирование программного средства 7](#_Toc41756179)

[3 Разработка программного средства 10](#_Toc41756180)

[4 Руководство пользователя 18](#_Toc41756181)

[5 Тестирование 23](#_Toc41756182)

[Заключение 25](#_Toc41756183)

[Список используемых источников 26](#_Toc41756184)

[Приложение А 27](#_Toc41756185)

[Приложение Б 28](#_Toc41756186)

# Введение

Целью курсового проекта является создание веб-приложения для книжного магазина. Оно должно реализовать работу с данными о книгах, авторах, позволять пользователям аутентифицироваться и регистрироваться в системе, выбирать книги для покупки, оставлять комментарии, добавлять книги в избранное.

При построении архитектура приложения использовалась модель MVC – Model-View-Controller. Она позволяет создать легко расширяемый и поддерживаемый код, обеспечивает чёткую структуру программы.

Для серверной части приложения использовался фреймворк Express, для работы с базой данных использовалась ORM Sequelize. Express - это минималистичный и гибкий веб-фреймворк для приложений Node.js, предоставляющий обширный набор функций для мобильных и веб-приложений. Имея в своем распоряжении множество служебных методов HTTP и промежуточных обработчиков, создать надежный API можно быстро и легко. Sequelize - это ORM-библиотека для приложений на Node.js, которая осуществляет сопоставление таблиц в БД и отношений между ними с классами. При использовании Sequelize мы можем не писать SQL-запросы, а работать с данными как с обычными объектами. Причем Sequelize может работать с рядом СУБД - MySQL, Postgres, MariaDB, SQLite, MS SQL Server.

Кроме того, для создания клиентской части приложения использовались библиотеки React и Redux. React (иногда React.js или ReactJS) — [JavaScript](https://ru.wikipedia.org/wiki/JavaScript)-[библиотека](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)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 разработки [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). Она может использоваться для разработки [одностраничных](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) и мобильных приложений. Её цель — предоставить высокую скорость, простоту и масштабируемость. Redux — библиотека для [JavaScript](https://ru.wikipedia.org/wiki/JavaScript) с открытым исходным кодом, предназначенная для управления состоянием приложения. Чаще всего используется в связке с [React](https://ru.wikipedia.org/wiki/React) или [Angular](https://ru.wikipedia.org/wiki/Angular_(%D1%84%D1%80%D0%B5%D0%B9%D0%BC%D0%B2%D0%BE%D1%80%D0%BA)) для разработки клиентской части. Содержит ряд инструментов, позволяющих значительно упростить передачу данных хранилища через контекст.

# Постановка задачи

В настоящие время абсолютно все книжные магазины, имеющие огромный ассортимент различные книги, журналы, газеты. Такие магазины книг используют современные автоматизированные системы управления и накопления базы данных. Кроме того, пользователи используют сайт для просмотра ассортимента продукции и выполнения заказов, что экономит время и клиенту, и магазину. В качестве примера рассмотрим сайты существующих интернет-магазинов.

Рассмотрим интернет-магазин biblio.by. Его функционал позволяет пользователям регистрироваться и авторизовываться, просматривать книги, добавлять их в избранное, формировать заказы, оставлять комментарии. Интерфейс списка книг представлен на рисунке 1.1.

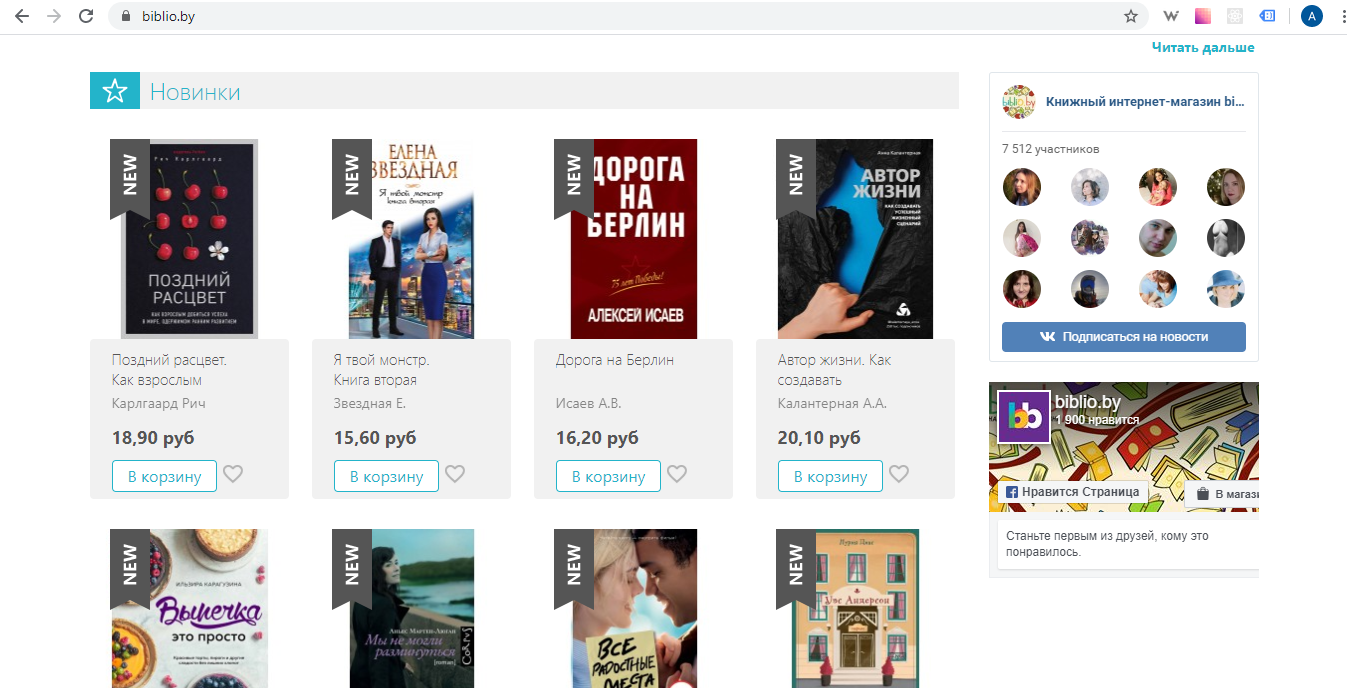


Рисунок 1.1 – Интерфейс списка книг

Интерфейс страницы детальной информации о книге представлен на рисунке 1.2.

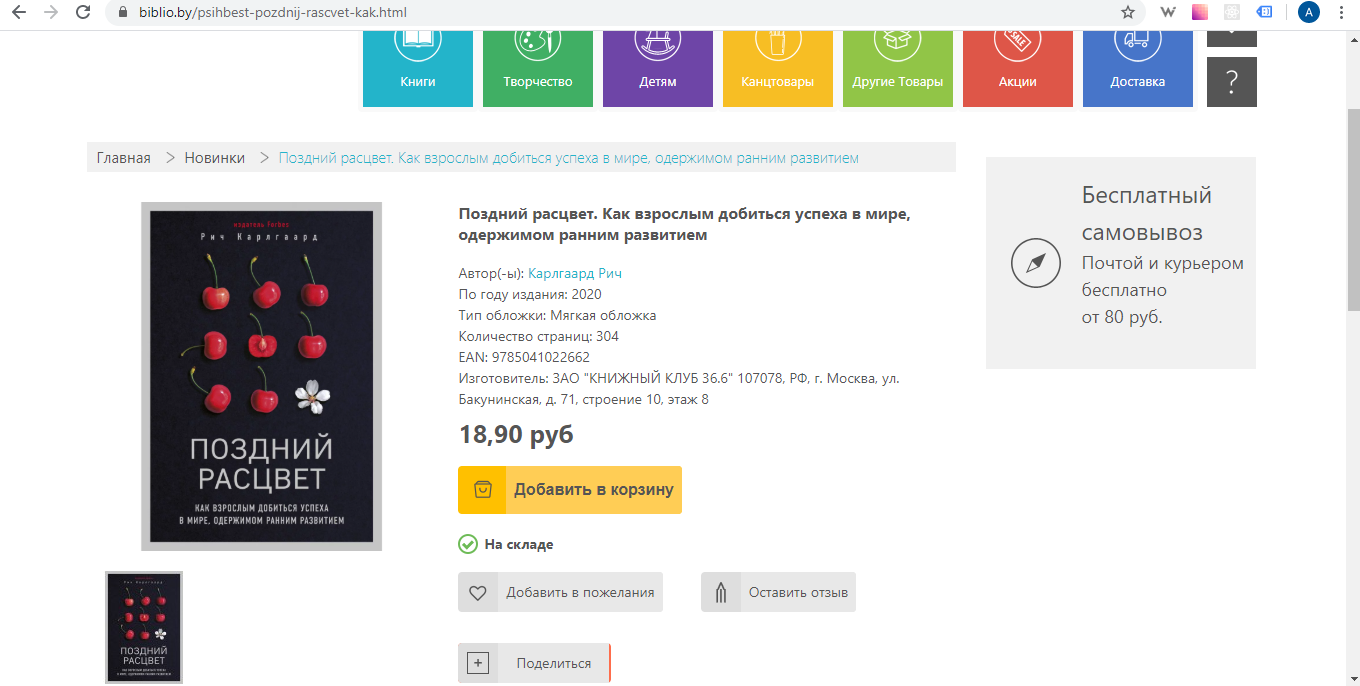


Рисунок 1.2 – Интерфейс страницы детальной информации о книге

Возможность добавлять и просматривать комментарии продемонстрирована на рисунке 1.3.

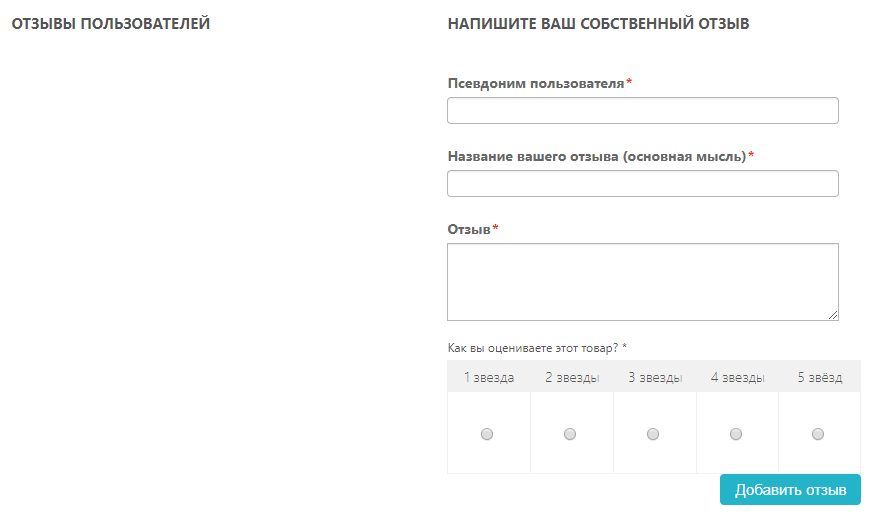


Рисунок 1.3 – Интерфейс секции комментариев

Окно регистрации представлено на рисунке 1.4.

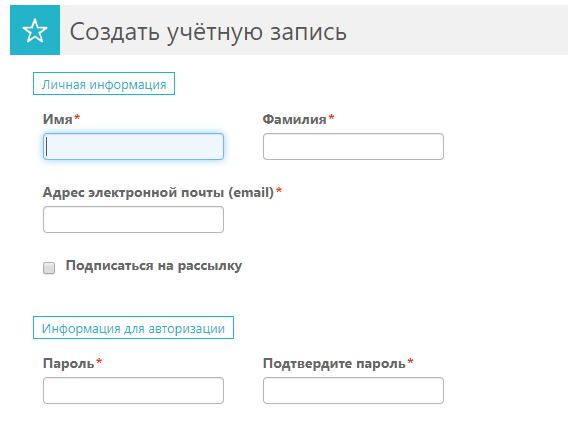


Рисунок 1.4 – Окно регистрации

Окно авторизации представлено на рисунке 1.5.

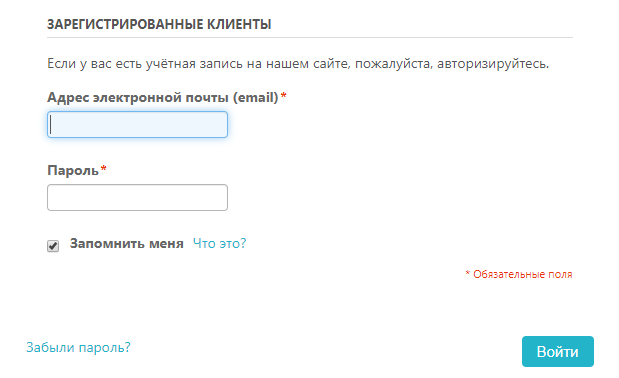


Рисунок 1.5 – Окно авторизации.

Исходя из основного функционала уже существующих программных средств, были выделены основные требования к курсовому проекту:

* выполнение регистрации и авторизации пользователей;
* разделение пользователей по ролям
* хранение и добавление товаров;
* покупка товаров;
* возможность оставлять комментарии к товарам;
* хранение избранных товаров.

# Проектирование программного средства

Обобщенная схема приложения показана на рисунке 2.1.

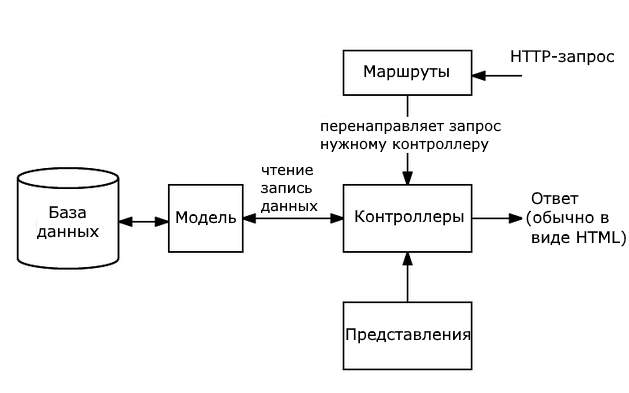


Рисунок 2.1 – Обобщенная схема приложения

В соответствии с этой схемой, приходящие HTTP запросы проходят через маршрутизатор, далее обрабатываются соответствующим контроллером, При необходимости происходит изменение базы данных с помощью моделей, ответ клиенту возвращается в виде JSON или HTML файла.

Рассмотрим подробнее паттерн MVC. Его схема приведена на рисунке 2.2.

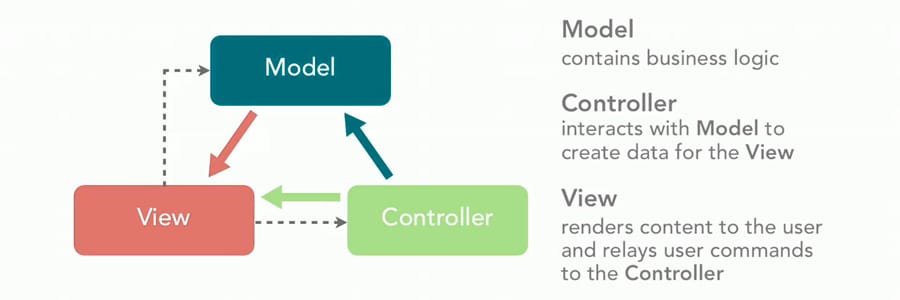


Рисунок 2.2 – Схема паттерна MVC

Model-View-Controller (MVC, «Модель-Представление-Контроллер», «Модель-Вид-Контроллер») — схема разделения данных приложения, [пользовательского интерфейса](https://ru.wikipedia.org/wiki/%D0%9F%D0%BE%D0%BB%D1%8C%D0%B7%D0%BE%D0%B2%D0%B0%D1%82%D0%B5%D0%BB%D1%8C%D1%81%D0%BA%D0%B8%D0%B9_%D0%B8%D0%BD%D1%82%D0%B5%D1%80%D1%84%D0%B5%D0%B9%D1%81) и управляющей логики на три отдельных компонента: модель, представление и контроллер — таким образом, что модификация каждого компонента может осуществляться независимо.

Модель предоставляет данные и методы работы с ними: запросы в базу данных, проверка на корректность. Модель не зависит от представления (не знает как данные визуализировать) и контроллера (не имеет точек взаимодействия с пользователем), просто предоставляя доступ к данным и управлению ими.

Модель строится таким образом, чтобы отвечать на запросы, изменяя своё состояние, при этом может быть встроено уведомление «[наблюдателей](https://ru.wikipedia.org/wiki/%D0%9D%D0%B0%D0%B1%D0%BB%D1%8E%D0%B4%D0%B0%D1%82%D0%B5%D0%BB%D1%8C_(%D1%88%D0%B0%D0%B1%D0%BB%D0%BE%D0%BD_%D0%BF%D1%80%D0%BE%D0%B5%D0%BA%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F))».

Модель, за счёт независимости от визуального представления, может иметь несколько различных представлений для одной «модели».

Представление отвечает за получение необходимых данных из модели и отправляет их пользователю. Представление не обрабатывает введённые данные пользователя.

Контроллер обеспечивает «связь» между пользователем и системой. Контролирует и направляет данные от пользователя к системе и наоборот. Использует модель и представление для реализации необходимого действия.

Для организации аутентификации и авторизации будет использоваться стандарт JSON Web Token. Это открытый стандарт ([RFC 7519](https://tools.ietf.org/html/rfc7519)) для создания токенов доступа, основанный на формате [JSON](https://ru.wikipedia.org/wiki/JSON). Как правило, используется для передачи данных для [аутентификации](https://ru.wikipedia.org/wiki/%D0%90%D1%83%D1%82%D0%B5%D0%BD%D1%82%D0%B8%D1%84%D0%B8%D0%BA%D0%B0%D1%86%D0%B8%D1%8F) в клиент-серверных приложениях. Токены создаются сервером, подписываются секретным ключом и передаются клиенту, который в дальнейшем использует данный токен для подтверждения своей личности. Схема его работы представлена на рисунке 2.3.

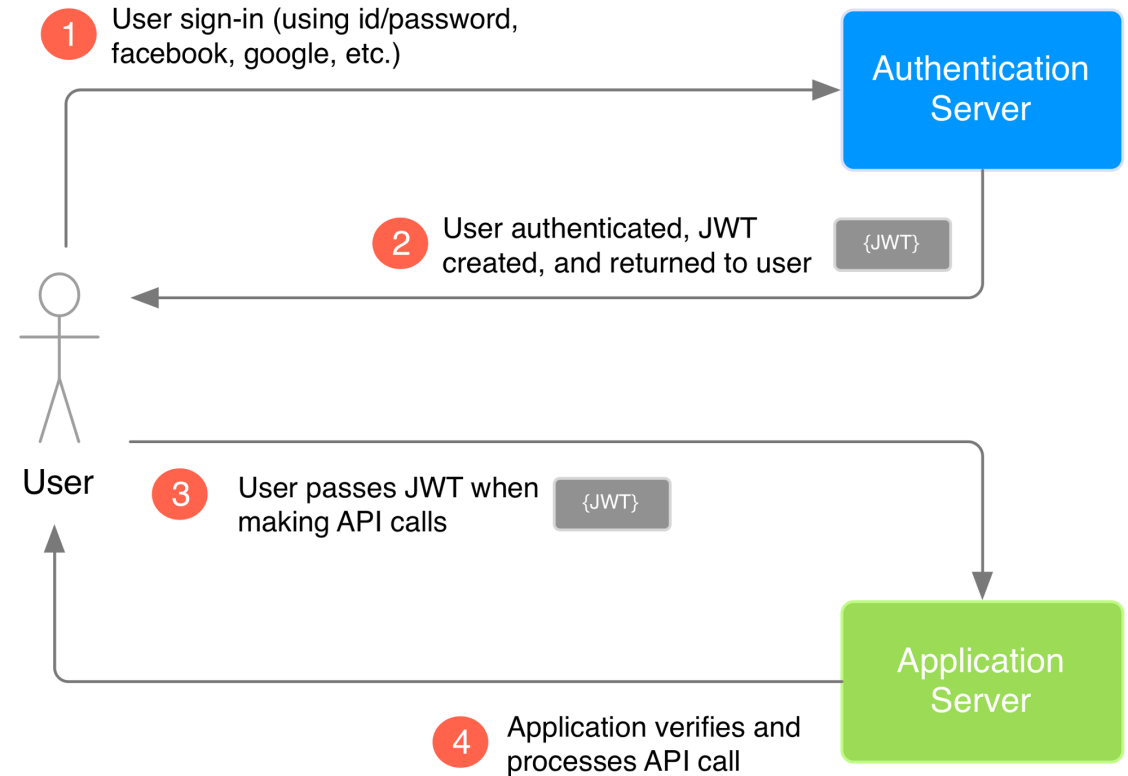


Рисунок 2.3 – Схема работы JWT

В соответствии с требованиями к программному средству были выделены следующие необходимые контроллеры:

* AuthorController – обрабатывает запросы на добавление и изменение авторов;
* BookController – обрабатывает запросы на добавление и изменение книг;
* CategoryController – обрабатывает запросы на получение данных о жанрах книг;
* CommentController – обрабатывает WebSocket-запросы на добавление комментариев;
* FavoriteController – обрабатывает запросы на получение избранных книг для пользователей
* HomeController – обрабатывает стартовый запрос;
* OrderController – обрабатывает запрос на создание заказа;
* UserController – обрабатывает запросы на создание, аутентификацию и изменение данных пользователя.

Кроме того, были выделены основные сущности базы данных:

* Author – хранит информацию об авторе;
* Book – хранит информацию о книге;
* Category – хранит информацию о жанре;
* Comment – хранит информацию о комментарии;
* Order – хранит информацию о заказе;
* FavoriteBook – хранит информацию о избранной пользователем книге;
* User – хранит информацию о пользователе.

Структура базы данных приведена в приложении А.

База данных содержит 10 таблиц:

* Authors – хранит информацию об авторах;
* Books – хранит информацию о книгах;
* Categories – хранит информацию о жанрах;
* Comments – хранит информацию о комментариях;
* Orders – хранит информацию о заказах;
* Users – хранит информацию о пользователях.
* FavoriteBooks – хранит информацию о избранных пользователями книгах;
* BooksToAuthors – содержит информацию о соответствии авторов книгам, вспомогательная таблица для организации связи многие-ко-многим между таблицами Books и Authors;
* BooksToCategories – содержит информацию о соответствии жанров книгам, вспомогательная таблица для организации связи многие-ко-многим между таблицами Books и Categories;
* OrdersToBooks – содержит информацию о соответствии заказов книгам, вспомогательная таблица для организации связи многие-ко-многим между таблицами Books и Orders.

# Разработка программного средства

Рассмотрим отдельно front-end и back-end части программы.

Структура front-end части представлена на рисунке 3.1.

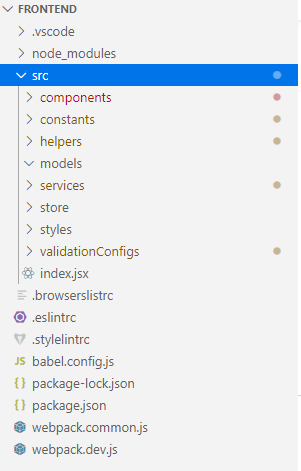


Рисунок 3.1 – Структура клиентской части приложения

Папка components содержит файлы с React-компонентами приложения и соответствующими им стилями, сгруппированные по страницам. Кратко об особенностях React будет рассказано позже. В папке constants хранятся файлы с различными константами: типы фильтров, адреса внутреннего роутинга, адреса для запросов к серверу, значения по умолчанию для компонентов. В папке helpers лежат различные вспомогательные модули: модуль для работы с датой, для формирования URL, для отправки запросов серверу, работы с local storage и др. Папка store в свою очередь содержит папки actions, где содержатся типы и генераторы действий в Redux, context, где хранятся значения по умолчанию для контекста React, и reducers, которая хранит редукторы для Redux. Кратко об особенностях Redux будет рассказано позже. В папке styles лежат некоторые общие для всего проекта стили и иконки. Папка validationConfigs хранит файлы с правилами клиентской валидации для некоторых форм.

Файл .eslintrc – это конфигурация для плагина Eslint. Это инструмент статического анализа кода для выявления проблемных шаблонов, обнаруженных в коде JavaScript.

Среди его достоинств стоит выделить:

* большое количество проверок;
* автоматическое исправление кода;
* легкую расширяемость;
* хорошую интеграцию с редакторами кода, системами сборки и не только;
* активное развитие;
* поддержку свежих стандартов не только в парсере;
* отличную [документацию](https://eslint.org/).

Babel.config.js содержит конфигурация для Babel. Это бесплатный JavaScript-транскомпилятор с открытым исходным кодом, который в основном используется для преобразования кода ECMAScript 2015+ в обратно совместимую версию JavaScript, которая может быть запущена старыми механизмами JavaScript. Он необходим т.к. для создания React-компонентов используются классы, стрелочные функции, асинхронные операции и другие функции последних стандартом ECMAScript, которые могут не поддерживаться в старых браузерах.

Файлы webpack.common.js и webpack.dev.js содержат соответственно общую и dev-конфигурацию для Webpack Это система сборки, которая предоставляет не только бандлинг (компоновку) модулей, но и может выполнять задачи, которыми занимаются Gulp/Grunt. К тому же, Webpack не ограничивается JavsScript-файлами, он может работать с другой статикой вроде CSS, картинок, html-компонентов и др. Webpack также поддерживает очень полезную функцию — code splitting (разбиение кода). Большое приложение можно разбить на куски, которые загружаются по мере необходимости.

Рассмотрим некоторые особенности библиотеки React. Как уже говорилось, React – это [JavaScript](https://ru.wikipedia.org/wiki/JavaScript)-[библиотека](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)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 разработки [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). React-компоненты — это маленькие, повторно используемые части кода, которые возвращают React-элементы для отображения на странице. Props (пропсы) — это входные данные React-компонентов, передаваемые от родительского компонента дочернему компоненту. Компонент нуждается в state, когда данные в нём со временем изменяются. Например, компоненту Checkbox может понадобиться isChecked, а компоненту NewsFeed необходимо отслеживать посты при помощи состояния fetchedPosts. Самая большая разница между state и props состоит в том, что props передаются от родителя потомку, а state управляется самим компонентом. Компонент не может изменять props, но может изменять state. Кроме того, в приложение используется контекст. Контекст позволяет передавать данные через дерево компонентов без необходимости передавать пропсы на промежуточных уровнях.

В типичном React-приложении данные передаются сверху вниз (от родителя к дочернему компоненту) с помощью пропсов. Однако, этот способ может быть чересчур громоздким для некоторых типов пропсов (например, выбранный язык, UI-тема), которые необходимо передавать во многие компоненты в приложении. Контекст предоставляет способ делиться такими данными между компонентами без необходимости явно передавать пропсы через каждый уровень дерева.

Рассмотрим один из созданных компонентов. Его код приведен ниже.

import React from 'react';

import './input.scss';

export default class Input extends React.PureComponent {

    render() {

        const { label,  name,  type,  value,  onChange } = this.props;

        return (

            <div className="input">

                <label className="input\_\_label">{label}</label>

                <input name={name} type={type} value={value} className="input\_\_value" onChange={onChange}

                />

            </div>

        );

    }

}

Листинг 3.1 – Пример структуры React-компонента

Сначала мы подключаем необходимые библиотеки: React и PropTypes (она используется для проверки типов параметров) и файл со стилями. Далее объявляем и экспортируем класс Input, наследуемый от класса React.PureComponent. Внутри мы определяем типы для пропсов (это не обязательно, но облегчает отладку и тестирование), и реализуем метод render. Внутри него мы деструктуризируем пропсы и описываем возвращаемый элемент так, как хотим его видеть в браузере. При этом используется так называемый JSX-синтаксис, он позволяет использовать HTML-разметку прямо внутри JS кода. Для задания стиля используется свойство className, прописываем необходимые свойства для элементов, заключая JS-переменные в фигурные скобки. Данный элемент не имеет state, но имеет props: label, name, type, value, onChange.Когда мы захотим использовать этот компонент в другом месте, нам будет достаточно его импортировать и подключить так, как показано в листинге 3.2.

<Input name="name"

type="text"

label="Name:"

onChange={this.changeName}

value={this.state.name}

/>

Листинг 3.2 – Пример использования React-компонента

Кратко рассмотрим основные особенности библиотеки Redux. Redux — это способ управления состоянием приложения. Он основан на нескольких концепциях, изучив которые, можно с лёгкостью решать проблемы с состоянием. Redux идеально подходит для средних и крупных приложений. Им стоит пользоваться только в случаях, когда невозможно управлять состоянием приложения с помощью стандартного менеджера состояний в React или любой другой библиотеке.

В Redux общее состояние приложения представлено одним объектом JavaScript — state (состояние) или state tree (дерево состояний). Неизменяемое дерево состояний доступно только для чтения, изменить ничего напрямую нельзя.

Изменения возможны только при отправке action (действия). Действие (action) — это JavaScript-объект, который лаконично описывает суть изменения. В простом приложении тип действия задаётся строкой. По мере разрастания функциональности приложения лучше переходить на константы. Пример действия представлен в листинге 3.3.

export const addBooks = (books) => ({

    type: actionTypes.ADD\_BOOKS,

    books

});

Листинг 3.3 – Пример Redux-действия

Генераторы действий (actions creators) — это функции, создающие действия. При запуске действия обязательно что-то происходит и состояние приложения изменяется. Это работа редукторов. В нашем случае генераторы передаются в компоненты в качестве пропсов.

Редуктор (reducer) — это чистая функция, которая вычисляет следующее состояние дерева на основании его предыдущего состояния и применяемого действия. Пример редуктора приведен в листинге 3.4.

const books = (state = [], action) => {

    switch (action.type) {

        case actionTypes.ADD\_BOOKS:

            return [...state, ...action.books];

        default:

            return state;

    }

};

Листинг 3.4 – Пример Redux-редуктора

Хранилище (store) — это объект, который:

* содержит состояние приложения;
* отображает состояние через getState();
* может обновлять состояние через dispatch();
* позволяет регистрироваться (или удаляться) в качестве слушателя изменения состояния через subscribe().

Хранилище в приложении всегда уникально.

Запросы к серверу в приложении осуществляются с помощью fetch API. [Fetch API](https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API) предоставляет интерфейс JavaScript для работы с запросами и ответами HTTP. Он также предоставляет глобальный метод [fetch()](https://developer.mozilla.org/ru/docs/Web/API/GlobalFetch/fetch), который позволяет легко и логично получать ресурсы по сети асинхронно.

Подобная функциональность ранее достигалась с помощью [XMLHttpRequest](https://developer.mozilla.org/ru/docs/Web/API/XMLHttpRequest). Fetch представляет собой лучшую альтернативу, которая может быть легко использована другими технологиями, такими как [Service Workers](https://developer.mozilla.org/ru/docs/Web/API/ServiceWorker_API). Fetch также обеспечивает единое логическое место для определения других связанных с HTTP понятий, такие как CORS и расширения для HTTP. Клиент и сервер обмениваются данными с помощью формата JSON. Полученные от сервера данные передаются компонентам.

Стоит отметить, что почти вся серверная часть использует протокол HTTPS. HTTPS – это расширение протокола HTTP, поддерживающее шифрование посредством криптографических протоколов SSL и TLS. Для реализации передачи данных посредством HTTPS на веб-сервере, обрабатывающем запросы от клиентов, должен быть установлен специальный SSL-сертификат. Данный сертификат был сгенерирован с помощью криптографической библиотеки OpenSSL и в последствии был установлен на разработанном сервере.

Однако, не вся серверная часть основана на протоколе HTTPS. Функционал, связанный с добавлением отзывов книге, реализован на отдельном сервере, использующем протокол WebSocket. WebSocket представляет собой альтернативу HTTP, его можно применять для организации обмена данными в веб-приложениях. Этот протокол позволяет создавать долгоживущие двунаправленные каналы связи между клиентом и сервером. После установления соединения канал связи остаётся открытым, что даёт в распоряжение приложения очень быстрое соединение, характеризующееся низкими задержками и небольшой дополнительной нагрузкой на систему. Протокол WebSocket поддерживают все современные браузеры. Реализация функционала отзывов с помощью данного протокола позволяет добавлять новый отзыв в режиме реального времени, т.е. все клиенты, подключенные к серверу в данный момент, увидят этот добавленный отзыв без принудительной перезагрузки страницы.

Далее рассмотрим подробнее серверную часть приложения. Для обработки запросов использовался фреймворк Express, он позволяет маршрутизировать и обрабатывать запросы клиента. Маршрутизация запросов представлена в листинге 3.5.

app.use('/', homeRoute);

app.use('/Books', booksRoute);

app.use('/Users', usersRoute);

app.use('/FavoriteBooks', favoritesRoute);

app.use('/Authors', authorsRoute);

app.use('/Categories', categoriesRoute);

app.use('/Orders', ordersRoute);

expressApp.ws('/', function (ws) {

    ws.onmessage = function (msg) {

        commentsRoute(msg, ws, aWss);

    };

})

Листинг 3.5 – Маршрутизация запросов

После получения запроса он попадает в соответствующий метод контроллера, а при необходимости перед этим обрабатывается с помощью middleware, который отвечает за авторизацию пользователя. Функция, реализующая проверку корректности JWT-токена, показана в листинге 3.6.

let checkToken = (req, res, next) => {

    let token = req.headers['x-access-token'] || req.headers['authorization'];

    if (token) {

        jwt.verify(token, config, (err, decoded) => {

            if (err) {

                res.status = 401;

                return res.json({

                    success: false,

                    message: 'Token is not valid'

                });

            } else {

                req.decoded = decoded;

                next();

            }

        });

    } else {

        res.status = 401;

        return res.json({

            success: false,

            message: 'Auth token is not supplied'

        });

    }

};

Листинг 3.6 – Функция проверки корректности токена

Общая структура контроллера показана в листинге 3.7.

const express = require('express');

const authorRepository = require('../api/author')

const authorRoute = express.Router();

const middleware = require('../middlewares/checkJwt');

authorRoute.get('/', async (request, response) =>

{

…

}

);

authorRoute.get('/Get', async (request, response) => {…});

authorRoute.post('/Add', middleware.checkToken, express.json({ type: '\*/\*' }), async (request, response) => {…});

authorRoute.put('/Update', middleware.checkToken, express.json({ type: '\*/\*' }), async (request, response) => {…});

module.exports = authorRoute;

Листинг 3.7 – Структура контроллера

Контроллер реализует необходимую логику и обращается к базе данных. Он обеспечивает связь между пользователем и системой, использует модель и представление для реализации необходимой реакции на действия пользователя. Как правило, на уровне контроллера осуществляется фильтрация полученных данных и авторизация — проверяются права пользователя на выполнение действий или получение информации. В данном приложении, как уже отмечалось выше, используется ORM Sequelize. Пример модели для базы данных приведен в листинге 3.8.

module.exports = (Sequelize, sequelize) => {

    return sequelize.define('Author', {

        id: {

            type: Sequelize.INTEGER,

            primaryKey: true,

            allowNull: false,

            autoIncrement: true

        },

        name: {

            type: Sequelize.STRING

        },

        country: {

            type: Sequelize.STRING

        },

        birthdate: {

            type: Sequelize.DATE

        }

    });

};

Листинг 3.8 – Модель для сущности автора

Также был написан слой DAO – слой доступа к данным каждой из таблиц. Этот [шаблон проектирования](https://ru.wikipedia.org/wiki/%D0%A8%D0%B0%D0%B1%D0%BB%D0%BE%D0%BD_%D0%BF%D1%80%D0%BE%D0%B5%D0%BA%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F) применим ко множеству языков программирования, большинству программного обеспечения, нуждающемуся в хранении информации и к большей части баз данных, но традиционно этот шаблон связывают с приложениями на платформе [Java EE](https://ru.wikipedia.org/wiki/Java_EE), При этом были реализованы CRUD операции. CRUD — [акроним](https://ru.wikipedia.org/wiki/%D0%90%D0%BA%D1%80%D0%BE%D0%BD%D0%B8%D0%BC), обозначающий четыре базовые функции, используемые при работе с [базами данных](https://ru.wikipedia.org/wiki/%D0%91%D0%B0%D0%B7%D0%B0_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85): создание ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) create), чтение (read), модификация (update), удаление (delete). В [SQL](https://ru.wikipedia.org/wiki/SQL) этим функциям операциям соответствуют операторы [Insert](https://ru.wikipedia.org/wiki/Insert_(SQL)) (создание записей), [Select](https://ru.wikipedia.org/wiki/Select_(SQL)) (чтение записей), [Update](https://ru.wikipedia.org/wiki/Update_(SQL)) (редактирование записей), [Delete](https://ru.wikipedia.org/wiki/Delete_(SQL)) (удаление записей). В системах, реализующих доступ к базе данных через [API](https://ru.wikipedia.org/wiki/API) в стиле [REST](https://ru.wikipedia.org/wiki/REST), эти функции реализуются зачастую (но не обязательно) через HTTP-методы [POST](https://ru.wikipedia.org/wiki/POST_(HTTP)), [GET](https://ru.wikipedia.org/w/index.php?title=GET_(HTTP)&action=edit&redlink=1), [PUT](https://ru.wikipedia.org/w/index.php?title=PUT_(HTTP)&action=edit&redlink=1) и [DELETE](https://ru.wikipedia.org/w/index.php?title=DELETE_(HTTP)&action=edit&redlink=1) соответственно.

Хотя традиционно оперирование в стиле CRUD применяется к базам данных, такой подход может быть распространён на любые хранимые вычислительные сущности ([файлы](https://ru.wikipedia.org/wiki/%D0%A4%D0%B0%D0%B9%D0%BB), структуры в памяти, объекты). Шаблон проектирования [ActiveRecord](https://ru.wikipedia.org/wiki/ActiveRecord) обеспечивает соответствие функций CRUD [объектно-ориентированному подходу](https://ru.wikipedia.org/wiki/%D0%9E%D0%B1%D1%8A%D0%B5%D0%BA%D1%82%D0%BD%D0%BE-%D0%BE%D1%80%D0%B8%D0%B5%D0%BD%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B5%D0%BA%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5), и широко используется в различных [фреймворках](https://ru.wikipedia.org/wiki/%D0%A4%D1%80%D0%B5%D0%B9%D0%BC%D0%B2%D0%BE%D1%80%D0%BA) для доступа к базам данных из [объектно-ориентированных языков программирования](https://ru.wikipedia.org/wiki/%D0%9E%D0%B1%D1%8A%D0%B5%D0%BA%D1%82%D0%BD%D0%BE-%D0%BE%D1%80%D0%B8%D0%B5%D0%BD%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%BD%D1%8B%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA_%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%D1%8F).

Пример такого модуля приведен в листинге 3.9.

const db = require('../../db').Order;

module.exports = {

    getAll: () => {

        return db.findAll({ raw: true })

            .then(orders => { return orders })

            .catch(err => {

                console.log(err.toString());

            });

    },

    add: (order) => {

        return db.create(order)

            .catch(err => {

                console.log(err.toString());

            });

    },

    update: (order) => {

        db.update(order, {

            where: { id: order.id }

        }).then(isUpdatedArray => {

            if (!isUpdatedArray[0]) {

                return JSON.stringify({ error: 'No such records have been found' });

            }

        }).catch(err => {

            return JSON.stringify({ error: err.toString() })

        });

    },

    delete: (order) => {

        db.destroy({

            where: { id: order.id }

        }).then(isDeleted => {

            if (!isDeleted) {

                return (JSON.stringify({ error: 'No such records have been found' }));

            }

        }).catch(err => {

            return (JSON.stringify({ error: err.toString() }));

        });

    }

};

Листинг 3.9 – Модуль доступа к данным таблицы Orders.

# Руководство пользователя

При заходе на сайт пользователь попадает на страницу поиска книг, она показана на рисунке 4.1. В данный момент пользователь не авторизован. Он также может просматривать информацию о книгах и смотреть список авторов.

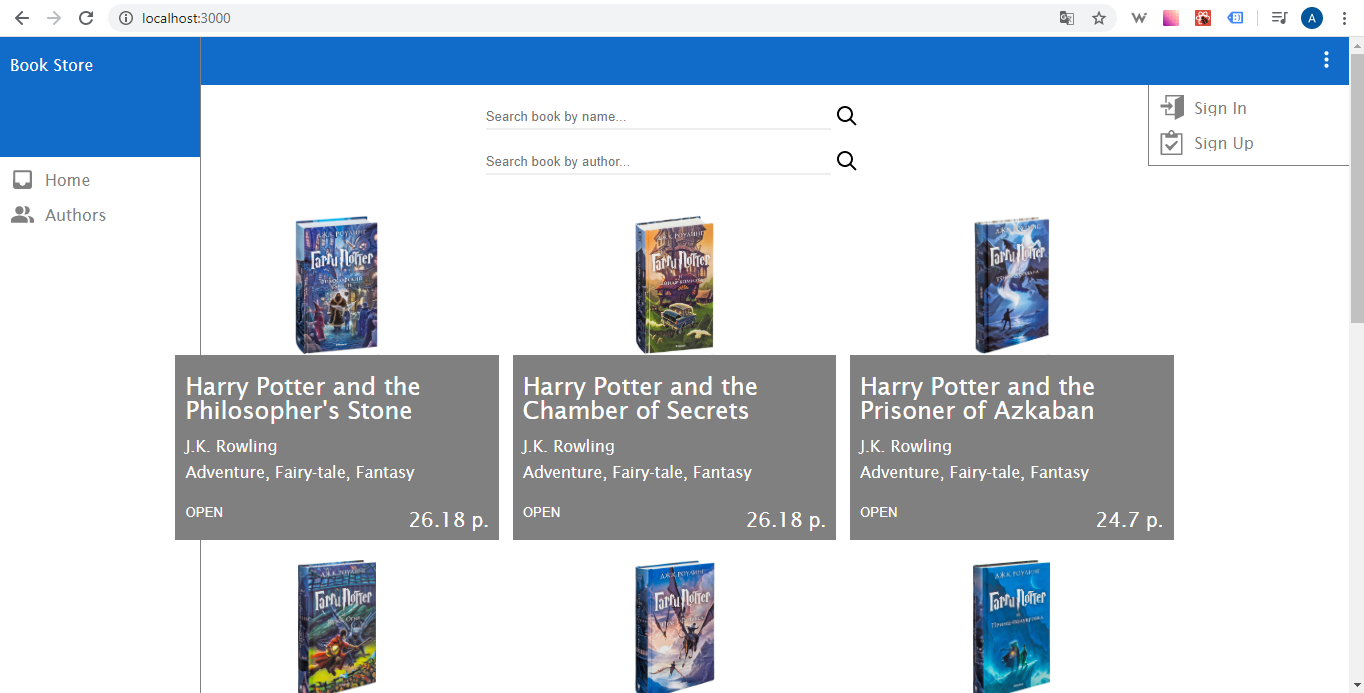


Рисунок 4.1 – Страница поиска книг

Пользователь имеет возможность зарегистрироваться, эта страница представлена на рисунке 4.2.

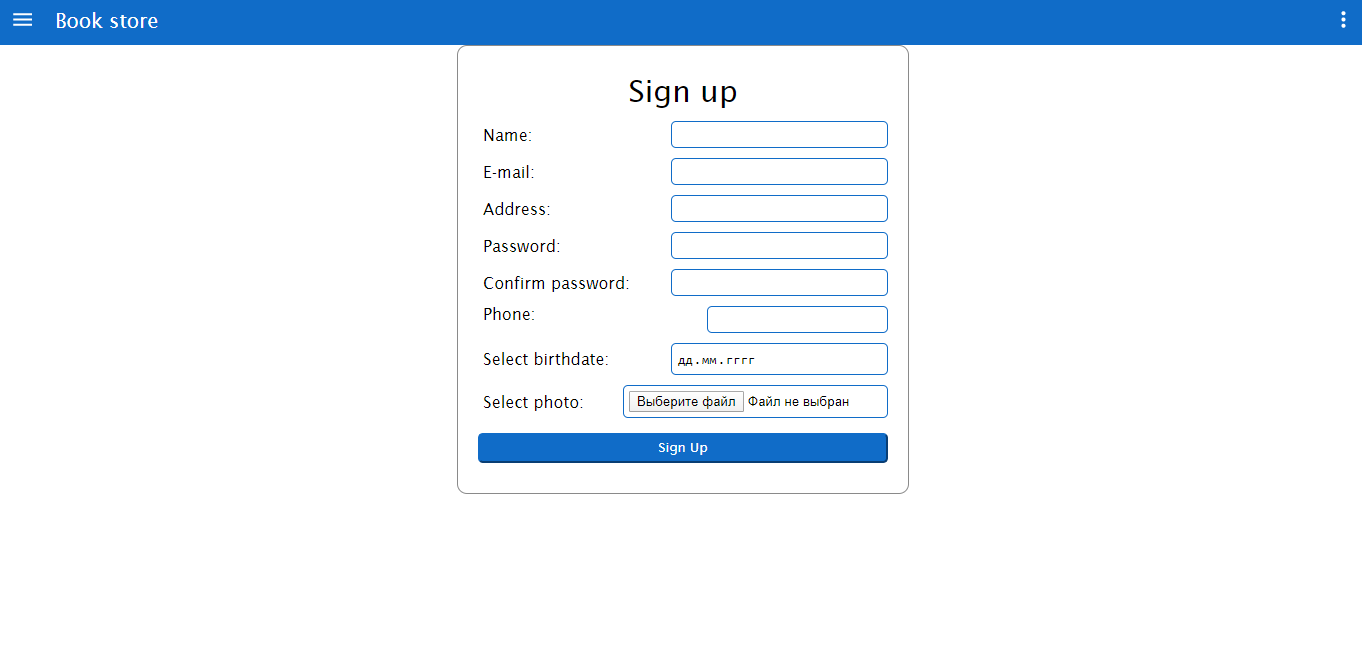


Рисунок 4.2 – Страница регистрации

Также пользователь имеет возможность зарегистрироваться, эта страница представлена на рисунке 4.3.

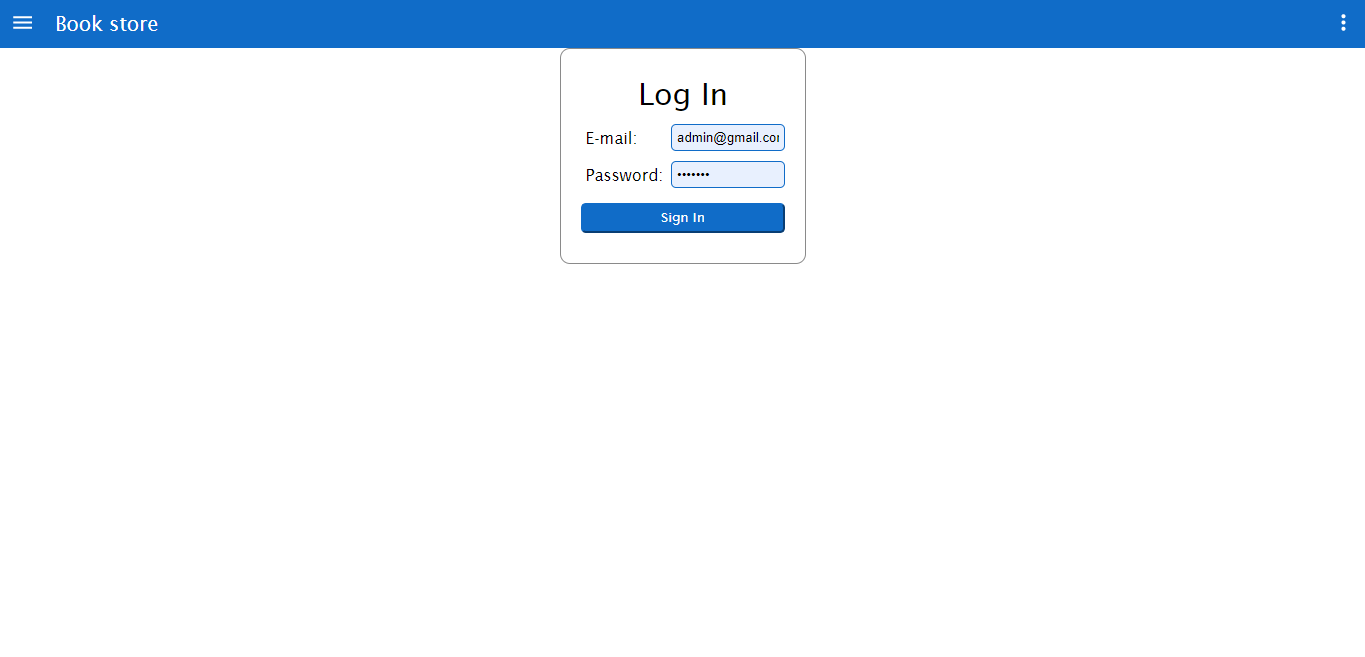


Рисунок 4.3 – Страница аутентификации

То, как выглядит стартовая страница для авторизованного обычного пользователя показано на рисунке 4.4.

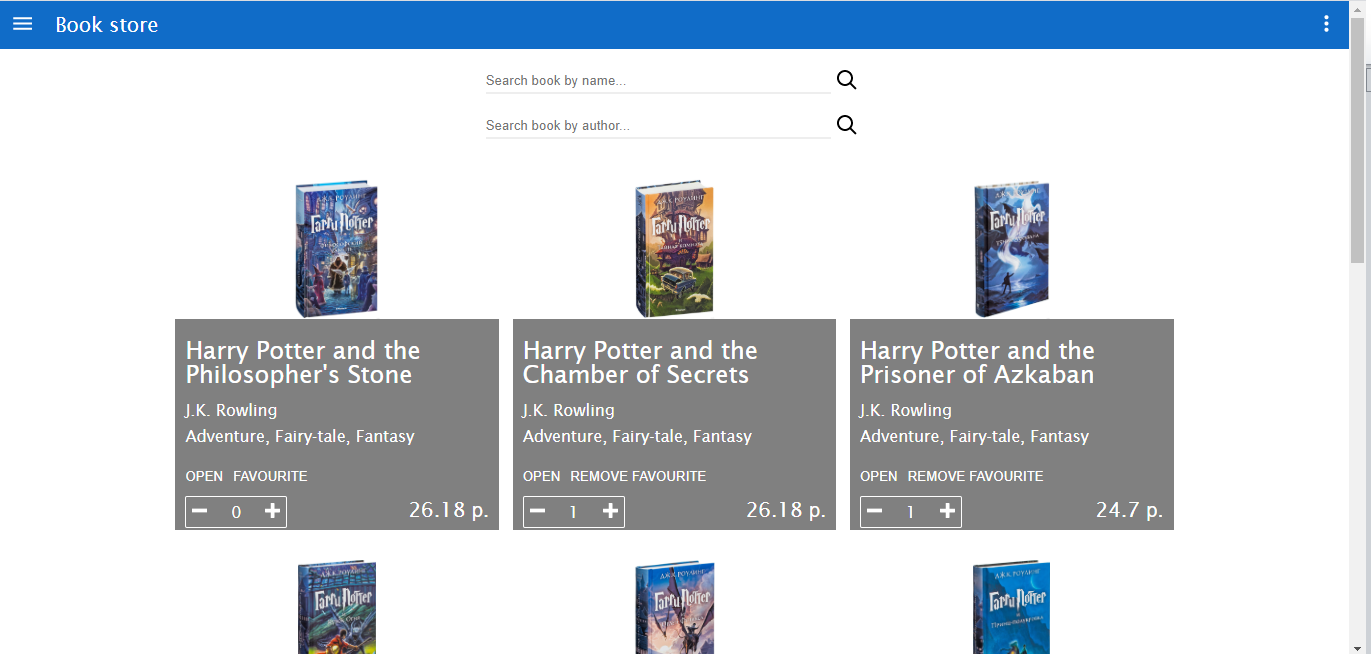


Рисунок 4.4 – Стартовая страница авторизованного пользователя

Авторизованный пользователь может изменять информацию профиля, комментировать книги, добавлять их в избранное, смотреть свой список избранных книг, добавлять книги в корзину и оформлять заказы. Также изменяется левое и правое меню, появляется возможность перейти на указанные страницы и выйти из аккаунта.

Страница детальной информации о книге представлена на рисунке 4.5.

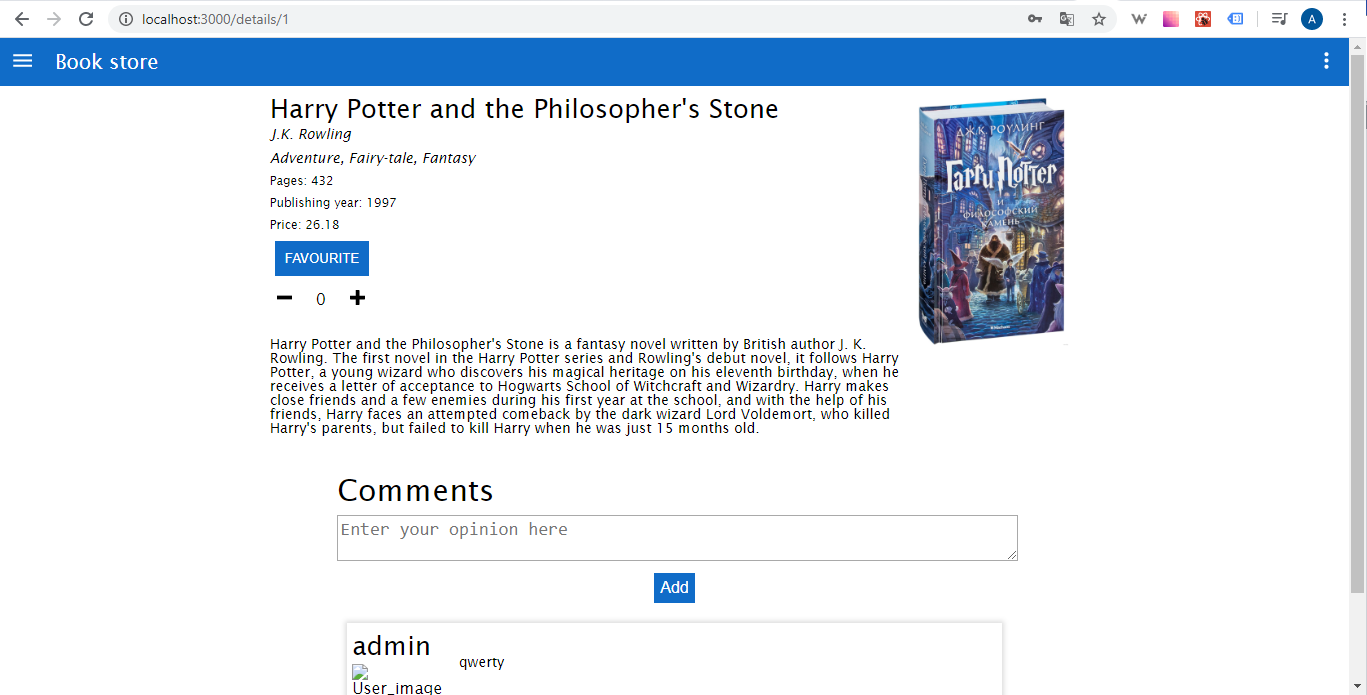


Рисунок 4.5 – Страница детальной информации о книге

Список избранных книг показан на рисунке 4.6.

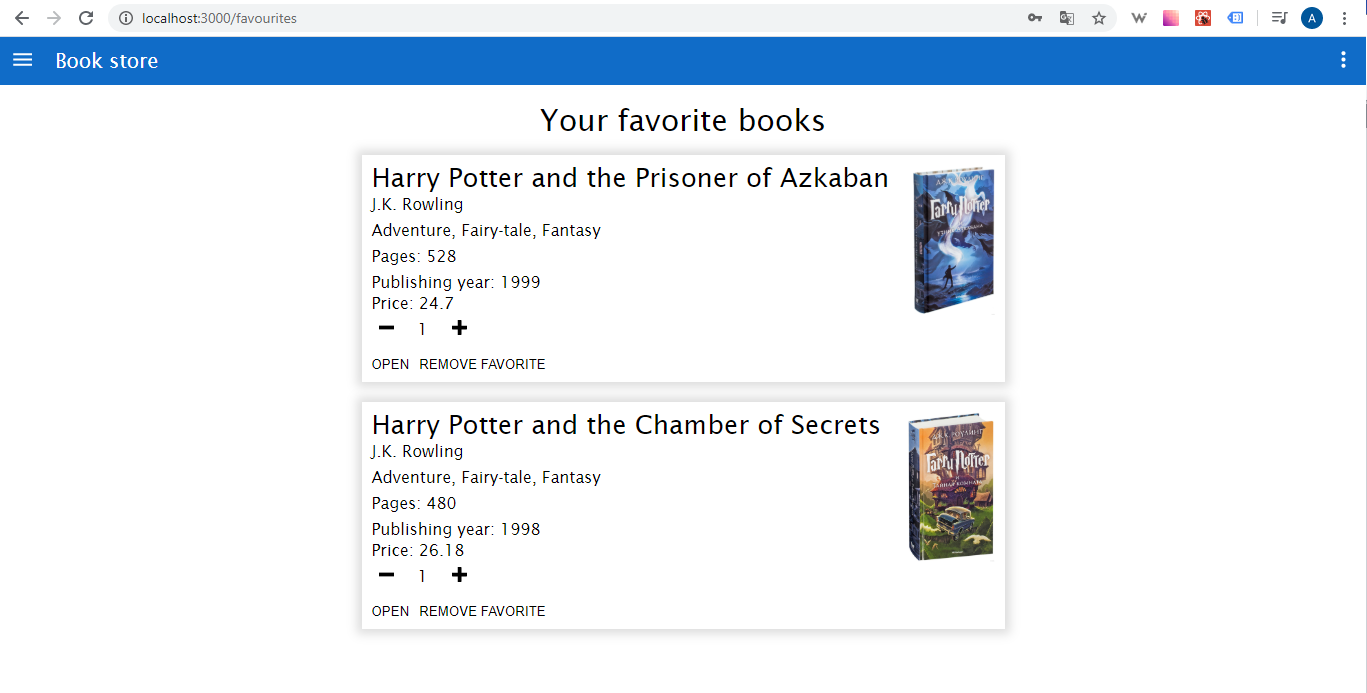


Рисунок 4.6 – Список избранных книг

Пользователь может добавлять и удалять товары из корзины, они хранятся в local storage, и поэтому данные не теряются между сессиями.

Страница сведений о заказе представлена на рисунке 4.7.

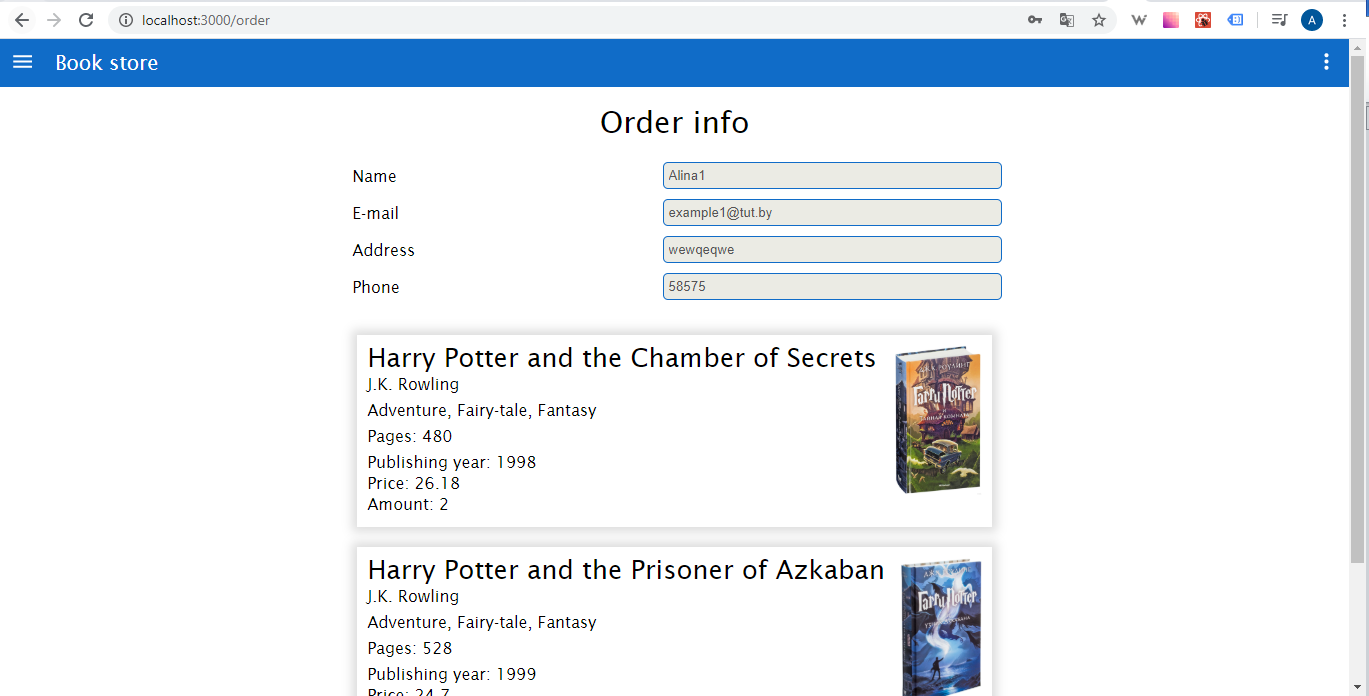


Рисунок 4.7 – Страница заказа

Страница с информацией о пользователе, предоставляющая возможность её изменить, представлена на рисунке 4.8.

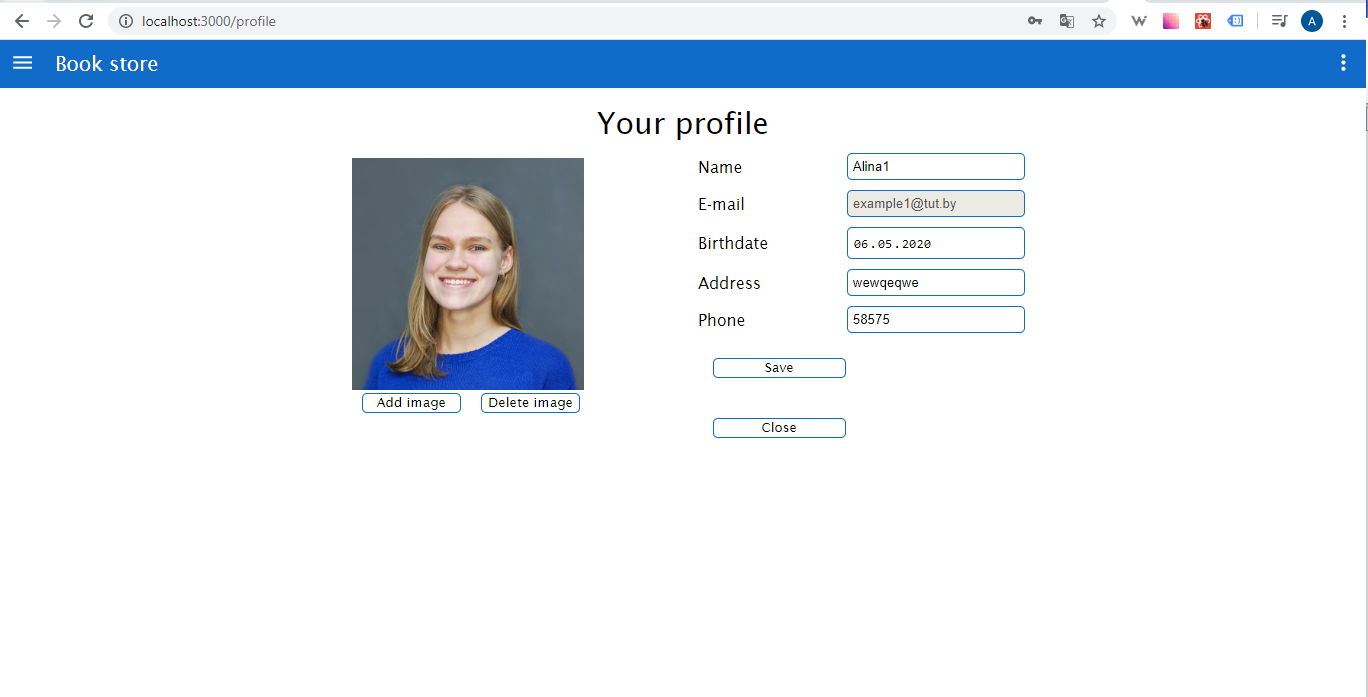


Рисунок 4.8 – Профиль пользователя

Авторизованный администратор может добавлять и изменять книги, добавлять и изменять авторов. Профиль администратора предопределен при создании базы данных, поэтому создать его явно невозможно.

Страница для добавления и обновления книги представлена на рисунке 4.9.

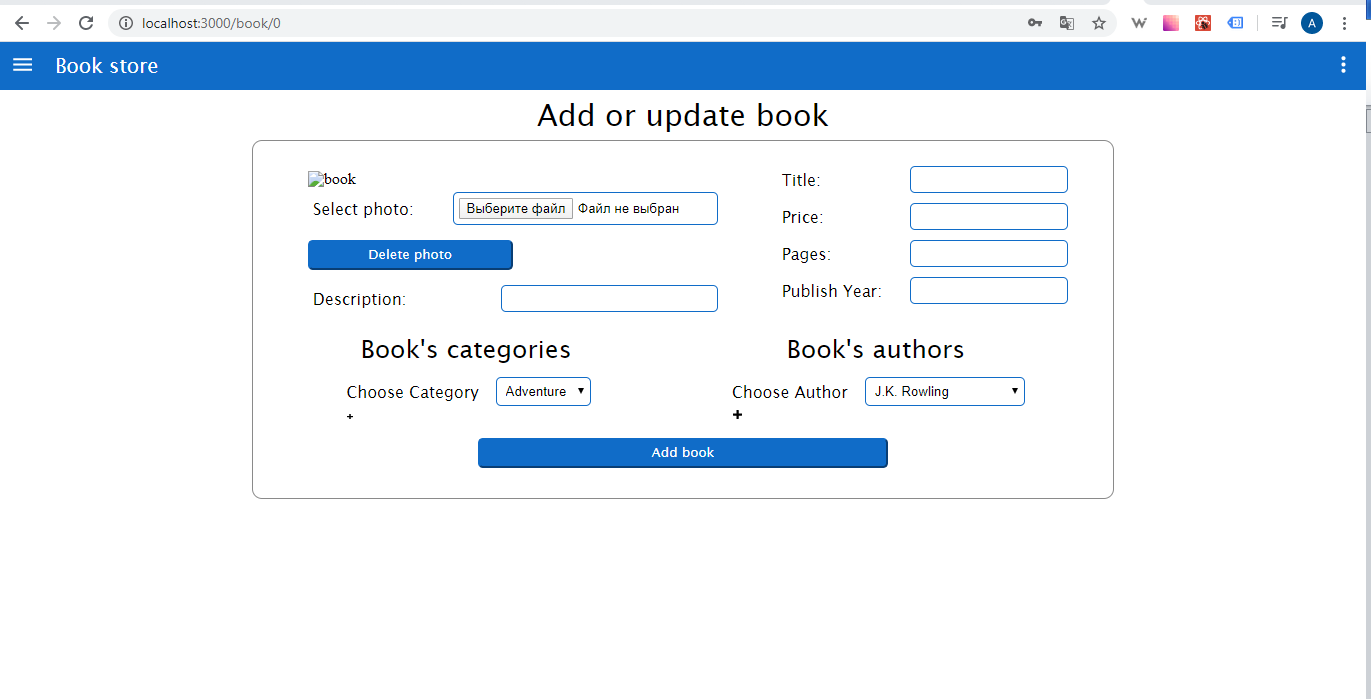


Рисунок 4.9 – Страница для добавления или обновления книги

Страница для добавления и обновления автора представлена на рисунке 4.10.

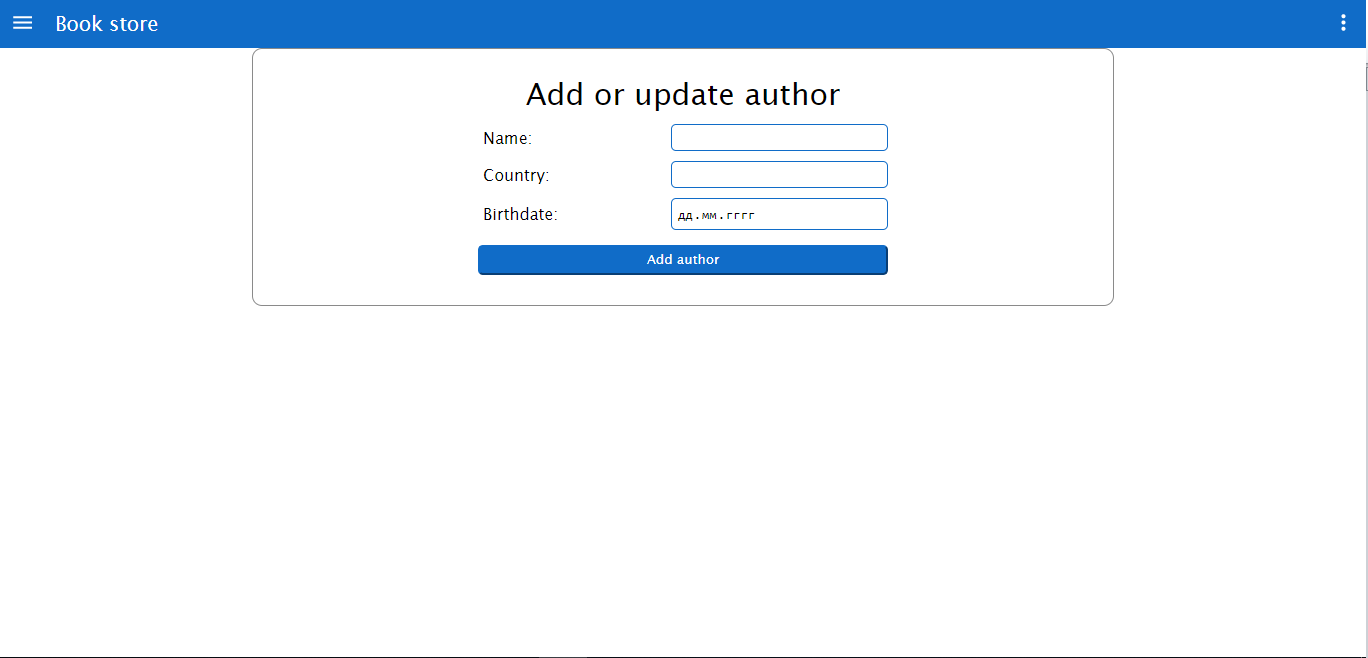


Рисунок 4.10 – Страница для добавления или обновления автора

Диаграмма использования веб-сайта приведена в приложении Б.

# Тестирование

Проведем тестирование на пустые и некорректные значения полей форм.

Тестирование валидации формы авторизации представлено на рисунке 5.1.

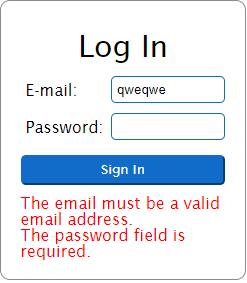


Рисунок 5.1 – Тестирование валидации формы авторизации

Тестирование валидации формы регистрации приведено на рисунке 5.2.

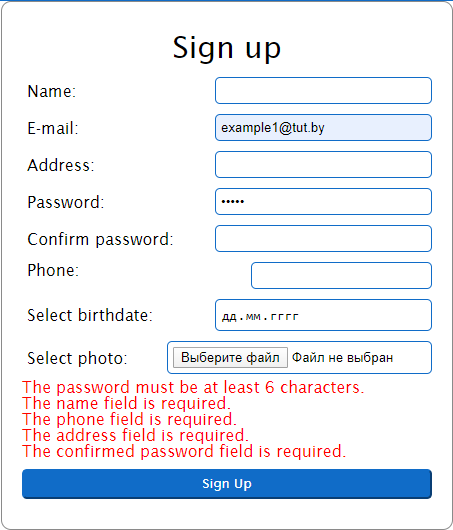


Рисунок 5.2 – Тестирование валидации формы регистрации

Тестирование формы добавления автора приведено на рисунке 5.3.



Рисунок 5.3 – Тестирование валидации формы добавления автора

Тестирование формы добавления книги представлено на рисунке 5.4.

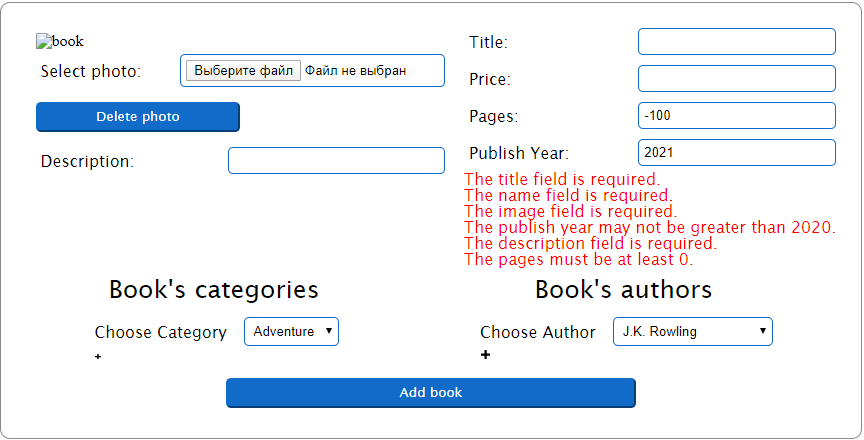


Рисунок 5.4 – Тестирование валидации формы добавления книги

Для реализации валидации использовалась библиотека simple-react-validator. Эта библиотека предлагает широкий набор встроенный правил, которые позволяют налету проводить проверку значения.

# Заключение

В результате выполнения данной курсовой работы было создано WEB-приложение «Книжный магазин». При разработке были выполнены все пункты из указанного списка предполагаемого основного функционала приложения.

В программном средстве реализованы следующие функции:

1. Анонимный пользователь может:

* зарегистрироваться;
* авторизоваться;
* смотреть список книг и осуществлять поиск и фильтрацию;
* смотреть список авторов.

1. Обычный пользователь может

* добавлять книги в избранное;
* изменять информацию профиля;
* добавлять книги в корзину
* оформлять заказы.

1. Администратор может:

* добавлять и редактировать книги;
* добавлять и редактировать авторов.

В соответствии с полученным результатом работы программы можно сделать вывод, что разработанная программа работает, верно, а требования технического задания выполнены в полном объеме.

# Список используемых источников

1. METANIT.COM Сайт о программировании [Электронный ресурс] / Режим доступа: <https://metanit.com>
2. ProfessorWeb .NET & Web Programming [Электронный ресурс] / Режим доступа: <https://professorweb.ru>
3. Stack Overflow [Электронный ресурс] / Режим доступа: <https://stackoverflow.com>
4. Хабр [Электронный ресурс] / Режим доступа: <https://habr.com>
5. Документация по ReactJS [Электронный ресурс] / Режим доступа <https://reactjs.org/>

# Приложение А

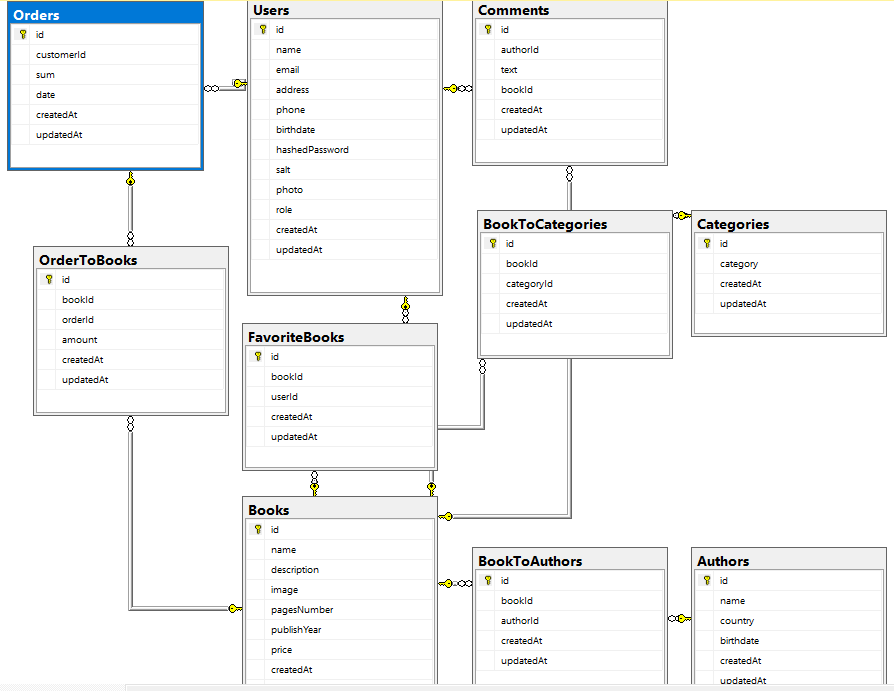


Рисунок 1 -Структура базы данных

# Приложение Б

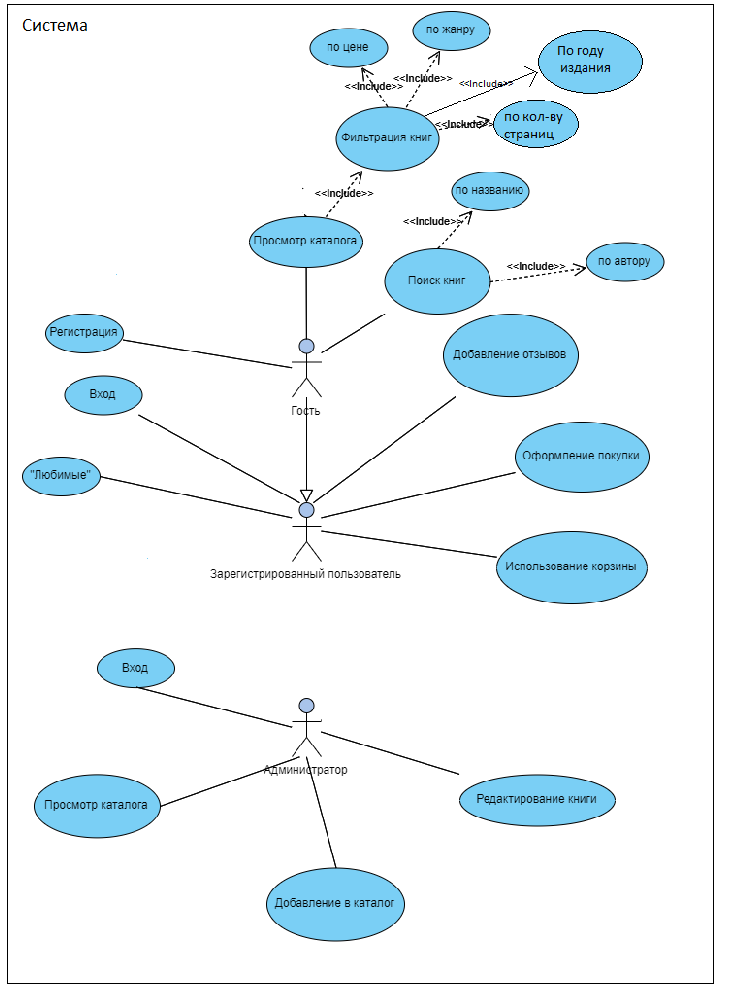


Рисунок 1 –Диаграмма вариантов использования