

МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

учреждение образования

«Гродненский государственный университет имени Янки Купалы»

Факультет математики и информатики

Кафедра современных технологий программирования

КАЗЕЛЛО АННА ВИТАЛЬЕВНА

**Разработка вэб-приложения «Система заказа книг
в библиотеке»**

Курсовая работа

студентки 3 курса специальности

1-40 01 02 «Программное обеспечение информационных технологий»
дневной формы получения образования

Научный руководитель

Скращук В.С.,

старший преподаватель

кафедры современных

технологий программирования

Гродно 2016

СОДЕРЖАНИЕ

РЕЗЮМЕ	3
ВВЕДЕНИЕ.....	4
ГЛАВА 1. ОБЗОР СУЩЕСТВУЮЩИХ РЕШЕНИЙ	5
ГЛАВА 2. ПОСТАНОВКА ЗАДАЧИ	10
ГЛАВА 3. ПРОЕКТИРОВАНИЕ ПРИЛОЖЕНИЯ.....	12
3.1 ОПИСАНИЯ ПРЕДМЕТНОЙ ОБЛАСТИ БАЗЫ ДАННЫХ	14
3.2 ПРОЕКТИРОВАНИЕ ПОЛЬЗОВАТЕЛЬСКОГО ИНТЕРФЕЙСА	16
ГЛАВА 4. ПРОГРАММНАЯ РЕАЛИЗАЦИЯ ПРИЛОЖЕНИЯ	19
4.1 ПРЕДСТАВЛЕНИЕ ДАННЫХ В ПРОЕКТЕ.....	21
4.2 РЕАЛИЗАЦИЯ ПРОГРАММНОГО ИНТЕРФЕЙСА ПРИЛОЖЕНИЯ:	23
4.3 АУТЕНТИФИКАЦИЯ ПОЛЬЗОВАТЕЛЕЙ	24
4.4 ЗАКАЗ КНИГИ.....	26
4.5 РЕАЛИЗАЦИЯ КЛИЕНТСКОЙ ЧАСТИ	29
4.5.1 Компонент App.....	30
4.5.2 Компонент Header.....	32
4.5.3 Отображение списка книг	32
4.5.4 Заказы	34
ЗАКЛЮЧЕНИЕ	35
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ	36

РЕЗЮМЕ

Казелло Анна Витальевна Курсовая работа – «Система заказа книг в библиотеке», 36 страниц, 12 рисунков, 11 использованных источников.

Ключевые слова: библиотека, заказ книги, книга, автор, жанр.

Цель исследования: реализация Web приложения «Система заказа книг в библиотеке» с возможностью создания, редактирования, удаления и сохранения объектов в документо-ориентированную базу данных MongoDB, а так же автоматической постановки в очередь на книгу с помощью Javascript, Node.js, Express.js, React.js.

ВВЕДЕНИЕ

Еще не так давно, за неимением аналогов, обычные бумажные книги были очень популярны. Потом появились электронные книги. Они были всегда с собой, занимали место только на устройстве, с которого читались. С появлением электронных книг появились библиотеки и для них.

Однако, несмотря на все удобство таких книг, бумажные носители все еще активно используются. Многие книги все еще не оцифрованы по разным причинам. Помимо всего, есть круг пользователей, которые просто предпочитают бумажные носители из-за тактильных ощущений или привычки.

Таким образом, самые обычные библиотеки, все еще остаются актуальными, хоть и теряют свою популярность из-за отсутствия представления в сети. Но для того, чтобы взять необходимую книгу, нужно идти в библиотеку, а ведь еще не известно, имеется ли книга в наличии вообще, и останется ли она там, пока вы придете, что очень отталкивает.

Для того, чтобы люди не теряли время зря на напрасные походы в библиотеку было решено разработать приложение, которое позволило бы пользователям просматривать информацию об имеющихся в наличии книгах и заказывать их, а в случае их отсутствия позволит стать в очередь на книгу.

Для написания данного приложения необходимо было решить следующие задачи:

- Изучить технологии для работы с документно-ориентированными базами данных
- Продумать логику и смоделировать интерфейс приложения
- Изучить принципы REST-архитектуры
- Изучить способы написания одностраничных приложений с помощью библиотеки React
- Изучить способы аутентификации пользователей на основе JSON Web Token

ГЛАВА 1. ОБЗОР СУЩЕСТВУЮЩИХ РЕШЕНИЙ

Далее представлен обзор существующих решений систем заказа книг в библиотеке

1. Официальный сайт российской государственной библиотеки

Официальный сайт российской библиотеки предоставляет своим читателям электронную систему заказа книг. На часть изданий из Центрального основного фонда библиотеки распространяется система электронного заказа. Это значит, что требование на него можно оформить через Интернет, а затем получить в одном из читальных залов. Для заказа книг пользователю предлагается авторизоваться. Для авторизации необходимо иметь читательский билет РГБ нового образца (пластиковая карта).

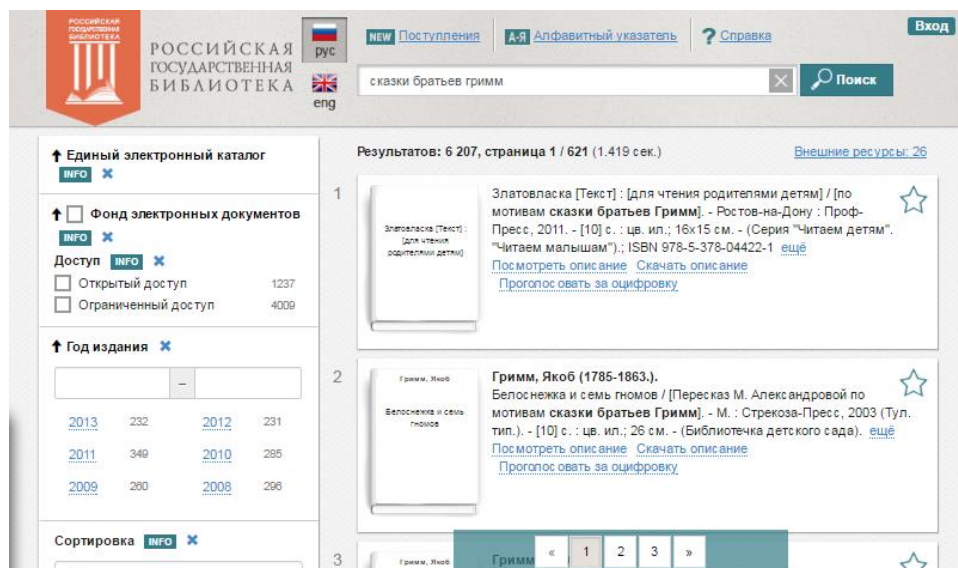


Рисунок 1. официальный сайт российской государственной библиотеки

Достоинства:

- Имеется расширенный поиск
- Можно установить фильтры для поиска
- Есть возможность добавлять документ в избранное
- Имеется возможность сортировки
- Имеется возможность поиска по ключевым словам

Недостатки:

- Невозможно встать в очередь на заказ книги
- Не предоставляются аннотации для книг

2. Сайт центральной библиотечной системы приморского района Санкт-Петербурга

Официальный сайт центральной библиотечной системы приморского района Санкт-Петербурга предоставляет своим читателям возможность предварительного заказа книг. Для этого пользователю необходимо иметь номер читательского билета и указать имя, фамилию, автора, название книги и свою электронную почту. На сайте расположена различная информация о библиотеке и районе города. А так же возможно скачать список новых поступлений.

The screenshot shows a web interface for a library system. The main form is titled 'Автор *' (Author) and includes fields for 'Название книги *' (Book Title), 'Ваше имя *' (Your Name), 'Номер читательского билета *' (Reader Card Number), 'Библиотека:' (Library) with a dropdown menu showing 'Центральная районная библиотека им. М. Е. Салтыкова-Щедрина', and 'Email *'. A blue 'Бронировать' (Book) button is at the bottom right of the form. The left sidebar contains buttons 'СОВЕТУЕМ ПОЧИТАТЬ' and 'ФОТОГАЛЕРЕЯ', social media icons, and a 'Настроить...' (Configure...) button. The right sidebar features a 'НАВЕРХ' (Back to Top) button and a logo for 'ДОСТУПНАЯ СРЕДА' (Accessible Environment) with the text 'Санкт-Петербург госуслуги gu.spb.ru'.

Рисунок 2. сайт центральной библиотечной системы приморского района Санкт-Петербурга

Достоинства:

- Мгновенная обработка заказа
- Имеется расширенный поиск
- Можно установить фильтры для поиска
- Результаты можно сортировать
- Имеется возможность поиска по ключевым словам

Недостатки:

- Невозможно встать в очередь на заказ книги
- Для продления заказа необходимо написать электронное письмо в библиотеку

- На электронное письмо о продлении могут не ответить, соответственно продление необходимо контролировать самому пользователю
- Для заказа книги необходимо произвести поиск на одном ресурсе, а бронировать ее на другом
- Для просмотра каталога необходимо переходить на другой ресурс

3. Сайт национальной библиотеки Беларуси

Национальная библиотека Беларуси, возглавляя систему библиотек страны, является хранительницей мощного информационного ресурса. Она активно генерирует собственные, приобретает электронные информационные ресурсы крупнейших мировых производителей и предоставляет свободный доступ к национальным и мировым информационным ресурсам. Оснащенная современным инженерным и технологическим оборудованием, библиотека использует новейшие информационные технологии, на качественно новом уровне удовлетворяет образовательные, научные, культурные запросы общества, формирует документальную память нации.

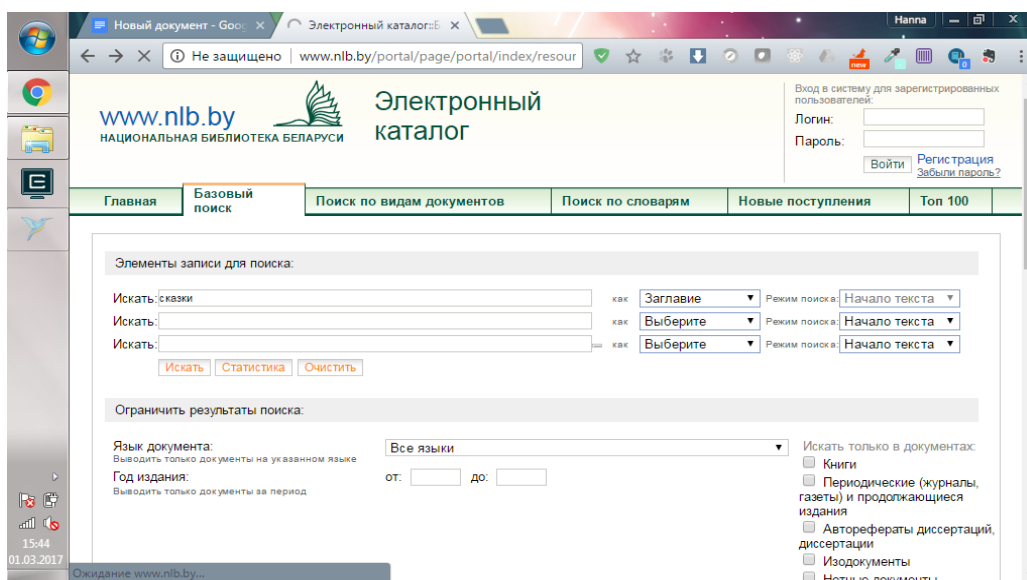


Рисунок 3. Сайт национальной библиотеки Беларуси

Достоинства:

- Имеется расширенный поиск
- Можно установить фильтры для поиска
- Можно заказывать издание на сегодня и на последующие дни

- Имеется возможность поиска по ключевым словам
- У пользователя уточняют его предпочтения

Недостатки:

- Не предоставляются аннотации для книг
- Невозможно встать в очередь на заказ книги

4. Российская национальная библиотека

Официальный сайт Национальной российской библиотеке предоставляет своим читателям возможность поиска и онлайн заказа книг. Для этого необходимо быть зарегистрированным в библиотеке. По номеру читательского билета и фамилии предоставляется допуск в систему онлайн заказа книг. Все коллекции РНБ, кроме документов, защищенных авторским правом, можно просматривать, не выходя из дома. Доступ к закрытым документам можно получить в читальных залах РНБ или в одном из Виртуальных читальных залов, организованных в Российской государственной библиотеке.

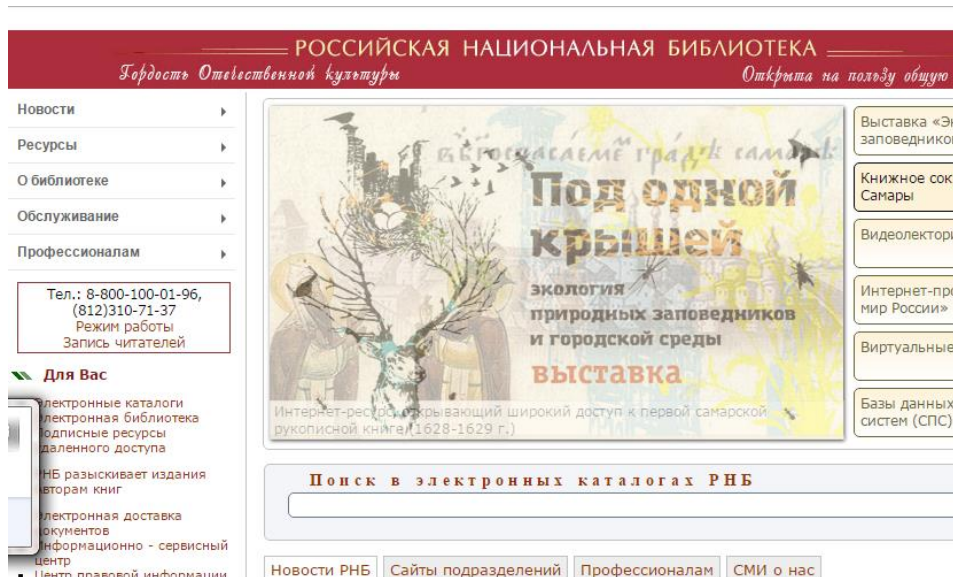


Рисунок 4. Российская национальная библиотека

Достоинства:

- Имеется возможность онлайн продления заказа
- Наличие расширенного поиска
- Наличие поиска по ключевым словам
- Возможность тематических запросов

Недостатки:

- Нет аннотаций к книгам
- Нет возможности постановки на очередь заказа

Таким образом, можно сделать вывод, что в целом такие системы заказа книг в библиотеке обычно обладают такими достоинствами как расширенный поиск, возможность добавлять заказа в избранное, а так же сортировка и фильтр результатов поиска. Но в то же время не предоставляют аннотации к книгам, что затрудняет выбор читателя, и не обладают возможностью постановки в очередь на книгу. Так же, возможность продления заказа онлайн скорее исключение, чем правило.

ГЛАВА 2. ПОСТАНОВКА ЗАДАЧИ

Так как целью является разработка системы заказов книг в библиотеке, пользователь должен получать актуальную информацию о книгах, имеющихся в наличии, и иметь возможность заказывать книгу. Будем считать это основными задачами данного приложения.

Для учета хранящихся в библиотеке книг в первую очередь необходимо хранилище данных, куда будут помещаться сведения об имеющихся книгах.

Для заполнения и работы с базой данных нужно предусмотреть такой функционал как просмотр, изменение и удаление информации о книгах. Так же, для более удобной работы, стоит реализовать функцию поиска по книгам.

Помимо непосредственно реализации функции заказа книги, необходимо предоставить инструмент обслуживания этих заказов библиотекарем. Под обслуживанием заказов будем понимать изменение статуса заказа.

Исходя из анализа, проведенного в предыдущей главе, так же следует дать пользователю возможность продлевать заказ онлайн, а так же постараться решить вопрос с постановкой пользователей в очередь на книгу.

Ну и естественно, необходимо предусмотреть регистрацию пользователей, для разграничения их прав и идентификации пользователя, который делает заказ.

Таким образом, к функционалу разрабатываемого приложения предъявляются следующие требования:

Роль «администратор»:

- Управление центральным хранилищем данных (добавление, редактирование, удаление информации).
- Управление пользовательскими заказами (изменять статус заказа).

Роль «пользователь»:

- Поиск книги по названию, описанию, ключевым словам или альтернативным именам

- Просмотр подробной информации о выбранной книге.
- Заказ книги.
- Продление книги.
- Просмотр истории своих заказов

Общие требования:

- Процесс работы с приложением доступен только при наличии Интернет-соединения.
- При изменении статуса заказа - оповещение пользователя по электронной почте
- Удаление заказа возможно, только если он не актуален

ГЛАВА 3. ПРОЕКТИРОВАНИЕ ПРИЛОЖЕНИЯ

Согласно Википедии, веб-приложение — клиент-серверное приложение, в котором клиентом выступает браузер, а сервером — веб-сервер. Логика веб-приложения распределена между сервером и клиентом, хранение данных осуществляется преимущественно на сервере, обмен информацией происходит по сети. Одним из преимуществ такого подхода является тот факт, что клиенты не зависят от конкретной операционной системы пользователя, поэтому веб-приложения являются кроссплатформенными сервисами.[1]

Серверная часть данного приложения представлена веб-службой, написанная в стиле REST, которая предоставляет данные и модифицирует их в соответствии с запросами через программный интерфейс приложения.

REST (Representational state transfer) – это стиль архитектуры программного обеспечения для распределенных систем, таких как World Wide Web, который, как правило, используется для построения веб-служб. Термин REST был введен в 2000 году Роем Филдингом, одним из авторов HTTP-протокола. Системы, поддерживающие REST, называются RESTful-системами.

В общем случае REST является простым интерфейсом управления информацией без использования каких-то дополнительных внутренних прослоек. Каждая единица информации однозначно определяется глобальным идентификатором, таким как URL. Каждая URL в свою очередь имеет строго заданный формат[2]

В роли клиентской части выступает одностраничное приложение. Одностраничное приложение использует единственный HTML-документ как оболочку для всех веб-страниц и организующий взаимодействие с пользователем через динамически подгружаемые HTML, CSS, JavaScript, обычно посредством AJAX.

AJAX, Ajax (от англ. Asynchronous Javascript and XML — «асинхронный JavaScript и XML») — подход к построению интерактивных

пользовательских интерфейсов веб-приложений, заключающийся в «фоновом» обмене данными браузера с веб-сервером. В результате, при обновлении данных веб-страница не перезагружается полностью, и веб-приложения становятся быстрее и удобнее. [3]

В требованиях к приложению указано, что необходимо хранилище данных. Таким хранилищем может выступать документно-ориентированная база данных. Документно-ориентированная СУБД — СУБД, специально предназначенная для хранения иерархических структур данных (документов) и обычно реализуемая с помощью подхода NoSQL. В основе документно-ориентированных СУБД лежат документные хранилища (англ. document store), имеющие структуру дерева (иногда леса).[4]

3.1 ОПИСАНИЯ ПРЕДМЕТНОЙ ОБЛАСТИ БАЗЫ ДАННЫХ

Каждая книга определяется следующими параметрами:

- Название книги
- Авторы книги
- Описание
- Количество имеющихся копий
- Жанры книги
- Обложка
- Язык
- Количество страниц
- ISBN код
- Год публикации
- Город публикации
- Издательство
- Список альтернативных названий
- Список ключевых слов

Название книг могут совпадать, однако они характеризуются уникальным идентификатором, однозначно характеризующим каждую книгу. Также, книга может иметь несколько авторов и несколько жанров, список альтернативных названий и список ключевых слов. Количество страниц и количество копий книги не может быть меньше нуля. И может равняться нулю, в случае, когда все имеющиеся в библиотеке копии находятся на руках. Также хранится не само изображение обложки книги, а ссылка на изображение. ISBN код хранится в виде строки.

Следует хранить следующую информацию о пользователе:

- Пароль
- Адрес электронной почты
- Имя
- Фамилия
- Номер телефона

- Адрес
- Дата рождения
- Роль

Адрес электронной почты пользователя должен быть уникальным. Имя и Фамилия могут повторяться, однако они характеризуются уникальным идентификатором, однозначно определяющим пользователя. Роль пользователя хранится в виде булева значения, означающего, является ли пользователь администратором

Для пользователя следует предусмотреть 2 роли: администратор и читатель.

- Администратор должен иметь права на просмотр, изменение, добавление и удаление информации о пользователе, книге, авторе, жанре. А так же на изменение статуса заказа.
- Читатель должен иметь права на просмотр информации о себе, книгах, авторах, жанрах. А также на создание и удаление заказа.

У заказа есть следующие данные:

- Идентификатор книги
- Идентификатор пользователя, сделавшего заказ
- Дата взятия книги
- Дата заказа книги
- Дата окончания заказа
- Статус заказа

Для статуса заказа доступны только 3 значения: «Заказано», «Взято», «Возвращено». Дата окончания заказа назначается автоматически: к дате взятия книги прибавляются 14 календарных дней.

Так же в базе данных хранится очередь на книгу в виде:

- Идентификатор книги
- Список пользователей

3.2 ПРОЕКТИРОВАНИЕ ПОЛЬЗОВАТЕЛЬСКОГО ИНТЕРФЕЙСА

Изначально пользователю будет предлагаться войти в систему с помощью стандартной формы для входа. В случае ошибки ввода под полями будут выводиться сообщения для пользователя. В случае ошибки входа, сообщения об ошибке выводится под кнопкой «Sign in».

Рисунок 5. Форма входа в систему

Для неавторизованного пользователя все же доступен просмотр списка книг, авторов и жанров. Так же здесь находится поле для поиска по книгам.

Рисунок 6. Список книг

Рисунок 7. Списки авторов и жанров

Так же доступен просмотр подробной информации о книге. Кнопки «Edit» и «Delete» доступны только пользователям с правами «Администратор» и ведут к редактированию и удалению книги соответственно. Для зарегистрированного пользователя эти две кнопки заменяются на «Order», по нажатию на которую можно сделать заказ книги, а для незарегистрированного пользователя недоступны данные элементы.

Book Title

Authors

Edit Delete

Genres : genres, genres, genres Language : RU

Keywords : keywords, keywords Publication year : 2015

Alternative names : City of publishing : Minsk

Copies : 5 Publishing house : house

Pages : 320

Description : Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Рисунок 8. Подробная информация о книге

Для пользователя с правами «Администратор» добавляется 2 дополнительных пункта в меню: «Orders» и «Add book».

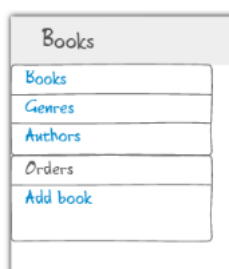


Рисунок 9. Меню для администратора

Пункт «Orders» ведет к просмотру списка заказов. Список заказов отображается в виде таблицы с колонками:

- «User» - имя и фамилия пользователя, заказавшего книгу
- «Book» - название книги
- «Authors» - авторы книги
- «Order date» - дата заказа

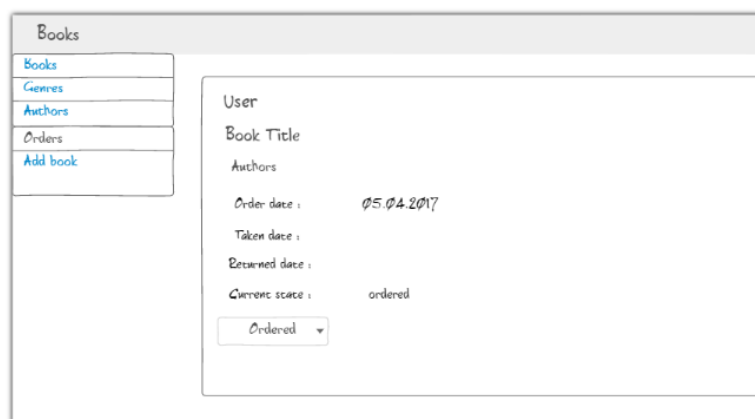
- «Taking date» - дата взятия книги
- «Status» - статус книги



User	Book	Authors	Order Date	Taking Date	Status
Hilky Mouse	Title	Author, Author2	05.04.2017	07.04.2017	returned
Babs Bunny	Title 2	Author 2	06.04.2017	06.04.2017	taken
Bugs Bunny	Title	Author	06.04.2017		ordered

Рисунок 10. Список заказов

При нажатии на статус заказа можно просмотреть полную информацию о заказе и изменить статус заказа путем выбора значения элемента «select».



Books

User

Book Title

Authors

Order date : 05.04.2017

Taken date :

Returned date :

Current state : ordered

Ordered ▼

Рисунок 11. Полная информация о заказе

ГЛАВА 4. ПРОГРАММНАЯ РЕАЛИЗАЦИЯ ПРИЛОЖЕНИЯ

Для разработки логической части приложения был выбран язык программирования Javascript.

JavaScript— прототипно-ориентированный сценарный язык программирования. Является реализацией языка ECMAScript (стандарт ECMA-262). Приложения, написанные на JavaScript, могут исполняться на серверах, использующих Java 6 и более поздних версий. Это обстоятельство используется для построения серверных приложений, позволяющих обрабатывать JavaScript на стороне сервера. Помимо Java 6, существует ряд платформ, использующих существующие движки (интерпретаторы) JavaScript для исполнения серверных приложений.[5]

Одним из таких движков является Node.js

Node или Node.js — программная платформа, основанная на движке V8 (транслирующем JavaScript в машинный код), превращающая JavaScript из узкоспециализированного языка в язык общего назначения. Node.js добавляет возможность JavaScript взаимодействовать с устройствами ввода-вывода через свой API (написанный на C++), подключать другие внешние библиотеки, написанные на разных языках, обеспечивая вызовы к ним из JavaScript-кода.[6]

Express.js, или просто Express, каркас web-приложений для Node.js, реализованный как свободное и открытое программное обеспечение под лицензией MIT. Он спроектирован для создания веб-приложений и API. Де-факто является стандартным каркасом для Node.js.[7]

Express.js минималистичен и включает большое число подключаемых плагинов, что во многом упрощает разработку приложения.

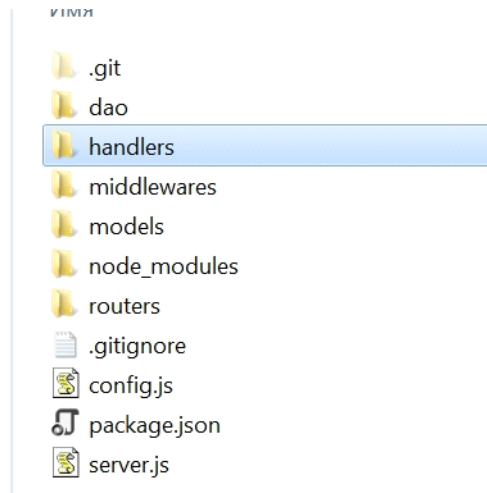


Рисунок 12. Структура приложения

На рисунке 6 представлена структура серверной части данного приложения.

Любой проект состоит из некоторого количества файлов, по которым разносят код. Это дает возможность структурировать проект, вынести независимые части, которые можно будет использовать в других проектах и вообще сделать код нагляднее.

Так вот, в Node.js каждый такой файл и представляет собой модуль, который можно подключить. Подключение происходит с помощью вызова функции `require`, которой нужно передать путь к файлу. [8]

- В папке `node_modules` находятся все модули, которые добавлены с помощью NPM. NPM — это менеджер пакетов для Node.js, который упрощает поиск и подключение сторонних модулей.
- В папке `routers` находятся файлы, отвечающие за маршрутизацию в приложении
- В папке `models` лежат схемы документов для базы данных
- В папке `handlers` находяся обработчики для маршрутизатора
- В папке `middlewares` находится файл промежуточного обработчика для маршрутизатора, определяющего авторизован ли пользователь
- В файле `package.json` перечислены все зависимости проекта. Файл `server.js` - главный файл, в котором и запускается сервер, а так же

производятся все настройки приложения. Сами настройки приложения хранятся в config.js .

4.1 ПРЕДСТАВЛЕНИЕ ДАННЫХ В ПРОЕКТЕ

Ранее было определено, что для хранения данных будет использована документо-ориентированная база данных. Такой базой данных была выбрана Mongo. Mongo — документоориентированная система управления базами данных (СУБД) с открытым исходным кодом, не требующая описания схемы таблиц. Написана на языке C++. Здесь используется документоориентированное хранение (JSON-подобная схема данных), а язык для формирования запросов Javascript, что очень удобно в данном случае.[9] Mongoose — это ODM(object-document mapper) для Mongo сделанная под node.js, необходимая для отображения документов из баз данных NoSQL.[10]

Все в Mongoose начинается со схемы. Каждая схема отображается на коллекцию в Mongo и определяет форму документов в коллекции.

Для того, чтобы создать модель данных, сначала необходимо создать схему данных, которая помещается в конструктор модели.

```
var UserSchema = new mongoose.Schema({
  ...
});
var User = mongoose.model('user', UserSchema);
```

При создании любой записи в базе данных ей присваивается уникальный идентификатор ObjectId, который генерируется автоматически.

Любое поле в документе может ссылаться на идентификатор документа в другой коллекции в пределах базы данных и быть дополненным при запросе.

К примеру, отрывок схемы документа для заказа:

```
book:{
  type: mongoose.Schema.Types.ObjectId,
  ref: 'book'
},
user:{
  type: mongoose.Schema.Types.ObjectId,
  ref: 'user'
}
```

},

Поле book хранит идентификатор книги и ссылается на документ book, поле user также. Благодаря этому, при запросе данного документа мы можем выполнить функцию populate, которая прикрепит все или только требуемые поля документа, на который ссылаются.

К примеру, реализация функции запроса заказа по идентификатору

```
Order.findOne({
  _id: new ObjectId(order_id)
}).populate('book', ['name', 'authors']).populate('user',
['firstName', 'lastName'])
.exec(function(err, result) {
  if (err) throw err;
  callback(result);
});
```

При поиске заказа мы выполняем функцию populate, в которую в качестве параметров отправляем название коллекции, и массив с названиями полей, которые хотим прикрепить. Если второй параметр не отправлять в функцию, то к документу прикрепятся все поля.

Для возможности поиска по коллекции можно задавать индексы. В данном случае для полей, по которым необходимо производить поиск определяются тип данных и вес.

```
BookSchema.index({name: 'text', genres: text, authors: 'text', description: 'text',
alternative_names: 'text', keywords: 'text' }, {name: 'Book search index', weights: {name:
10, alternative_names: 8, keywords: 7, description: 6, genres: 4, authors: 4 }});
```

На основе описания предметной области базы данных описано 3 схемы для документов:

- Book
- Order
- User

Все поля для данных схем описаны в соответствии с писанием предметной области.

Так же добавлена четвертая схема Queue. Документы, описанные по ней, хранят идентификатор книги и массив идентификаторов пользователей.

4.2 РЕАЛИЗАЦИЯ ПРОГРАММНОГО ИНТЕРФЕЙСА ПРИЛОЖЕНИЯ:

В REST архитектуре клиентская часть обращается к серверной через API, где каждая единица информации однозначно определяется URL.

Наиболее распространенный протокол передачи данных в таком случае - HTTP. Для HTTP действие над данными задается с помощью методов:

- GET (получить),
- PUT (добавить, заменить),
- POST (добавить, изменить, удалить),
- DELETE (удалить).

В данном приложении был определен следующий формат URL для получения и манипуляции различными данными:

Данные о книгах:

GET /books/ – получить данные всех книг

POST /books/ – добавить новую книгу

GET /books/:isbn – получить информацию о книге

PUT /books/:isbn – редактировать данные о книге

DELETE /books/:isbn – удалить данные о книге

GET /books/search/:searchString – поиск заданной строки во всех документах

GET /books/genres – получить список всех жанров

GET /books/genres/:genre – получить все книги с заданным жанром

GET /books/authors/ – получить список всех авторов

GET /books/authors/:author – получить все книги с заданным автором

Данные о пользователях:

/users/

Данные о заказах:

GET /orders/ – получить информацию о всех заказах

POST /orders/ – добавить новый заказ

GET /orders/state/:state — получить данные всех заказов в заданном состоянии

GET /orders/user/:user_id — получить все заказы заданного пользователя

GET /orders/book/:book_id — получить все заказы заданной книги

GET /orders/:order_id — получить всю информацию о заказе

DELETE /orders/:order_id — удалить заказ

PUT /orders/:order_id/:state — изменить статус заказа

Данные о пользователе:

GET /users/ — получить данные о всех пользователях

POST /users/ — добавить нового пользователя

GET /users/:user_id — получить информацию о пользователе

DELETE /users/:user_id — удалить пользователя

PUT /users/:user_id — изменить информацию о пользователе

Данные об очередях:

GET /queue/ — получить данные о всех очередях

GET /queue/:book — получить информацию об очереди за конкретной книгой

4.3 АУТЕНТИФИКАЦИЯ ПОЛЬЗОВАТЕЛЕЙ

В данном приложении было решено проводить аутентификацию пользователей на основе JSON Web Token. JSON Web Token (JWT) — маркер, который содержит в зашифрованном виде всю минимально необходимую информацию для аутентификации.

По запросу /users/authenticate/ методом POST в теле запроса отправляется электронный адрес и пароль. Обработчик запроса ищет запрашиваемого пользователя и проверяет правильность пароля. Если все хорошо, сторонний модуль «jsonwebtoken» создает маркер доступа на основе данных о пользователе и ключа, который хранится в config.json. После чего обработчик отправляет ответ с сообщением о результате аутентификации и токен, если аутентификация прошла успешно.


```

User.findOne({
  email: req.body.email
}, function(err, user) {
  if (err) throw err;
  if (!user) {
    res.json({
      success: false,
      message: 'Authentication failed. User not found.'
    });
  } else if (user) {
    // check if password matches
    if (user.password !== req.body.password) {
      res.json({
        success: false,
        message: 'Authentication failed. Wrong password.'
      });
    } else {
      // if user is found and password is right
      // create a token
      var token = jwt.sign(user, localConfig.secret, {
        //expiresInMinutes: 1440 // expires in 24 hours
      });
      // return the information including token as JSON
      res.json({
        success: true,
        message: 'Enjoy your token!',
        token: token,
        user: user
      });
    }
  }
});
});

```

Так же, некоторые запросы доступны только зарегистрированным пользователям. Например, таким как просмотр информации о пользователе. Для этого необходимо определять, аутентифицирован ли пользователь.

```

var token = req.body.token || req.query.token || req.headers['x-access-token'];

// decode token

if (token) {

  // verifies secret and checks exp

  jwt.verify(token, localConfig.secret, function(err, decoded) {

    if (err) {

      return res.json({

```

```

        success: false,

        message: 'Failed to authenticate token.'

    });

    } else {

        // if everything is good, save to request for use in other routes

        req.decoded = decoded;

        next();

    }

    });

} else {

    // if there is no token

    // return an error

    return res.status(403).send({

        success: false,

        message: 'No token provided.'

    });

}

```

Разделение прав доступа реализовано простым булевым «admin» полем в схеме пользователя «user», которое показывает, является ли пользователь администратором.

4.4 ЗАКАЗ КНИГИ

Для заказа книги необходимо отправить запрос методом POST на адрес /orders, где в теле запроса указывается идентификатор пользователя и идентификатор книги. Маршрутизатор вызывает функцию обработчика для создания нового заказа. Здесь определяется, есть ли свободная копия. В случае, если результат положительный, то копия отнимается и создается новый заказ. Если нет, пользователь ставится в очередь.

```

module.exports.addNewOrder = function(body, callback){

    bookDao.ifThereACopy(body.book, function(err, result){

        console.log('ifThereACopy '+ result);

        if (err) return callback(err);
    });
}

```

```

    else if(result==true) {
        bookDao.decrement(body.book, function(err, result){
            console.log('ifThereACopy ' + result);
            if (err) callback(err);
            else orderDao.addNewOrder(body, callback);
        })
    }
    else queueDao.editQueue(body, 'push', callback);
});
}

```

Если нет свободных копий, пользователь ставится в очередь. Для этого сначала вызывается функция редактирования очереди, в которой ищется очередь для данной книги. И если очередь не находится, то создается новая.

```

module.exports.editQueue = function(body, type, callback) {
    Queue.findOne({book: body.book}, function(err, result) {
        if (err) callback(err);
        if (result===null) {
            return addNewQueue(body, function(err, result) {
                if (!err) callback(null, result);
            })
        }
        var users = result.users;
        var user;
        if (type == 'push') {
            users.push(body.user);
            result.update({users: users}, function(err, result) {
                if (err) return callback(err);
                callback(null, {
                    message: "Successfully add user",
                    book: result
                });
            });
        }
    });
}

```

```
});}
```

При создании заказа ему автоматически присваивается статус «ordered» и заполняется дата заказа.

Когда заказ возвращается, идет проверка, есть ли очередь на данную книгу. И если есть очередь, то из очереди извлекается пользователь и для него создается заказ.

```
if (state == 'returned') {
    Order.findOne({
        _id: new ObjectId(id)
    }, function(err, result) {
        if (!err) {
            bookDao.increment(result.book, function(err, result) {
                if (err) throw err;
            })
            queueDao.popUserFromQueue(result.book, function(err, user){
                var body = {
                    book: result.book,
                    user: user
                }
                addNewOrder(body,function(result){
                });
            });
        }
    })
}
```

4.5 РЕАЛИЗАЦИЯ КЛИЕНТСКОЙ ЧАСТИ

Для реализации клиентской части данного приложения было решено использовать React.js. Это javascript-библиотека которая использует виртуальный DOM для отрисовки компонентов. Virtual DOM — это дерево React элементов на JavaScript. React отрисовывает Virtual DOM в браузере, чтоб сделать интерфейс видимым. React следит за изменениями в Virtual DOM и автоматически изменяет DOM в браузере так, чтоб он соответствовал виртуальному.

Для создания данного одностраничного приложения использовался глобальный модуль npm «create-react-app». Данный модуль создает основу для приложения, автоматически добавляя Webpack (сборщик приложений), Babel (транспилятор кода для возможности использовать новые функции языка) и другие необходимые вещи, избавляя от рутинной настройки.

Так как это одностраничное приложение, то здесь есть только одна HTML-страница, в теле которой расположен элемент, куда отрисовывается все приложение. В данном случае это элемент с идентификатором «root».

```
<body>
  <div id="root"></div>
</body>
```

Точкой входа приложения является файл index.js.

```
ReactDOM.render(
  <MuiThemeProvider>
    <Router>
      <App />
    </Router>
  </MuiThemeProvider>
, document.getElementById('root'))
```

Функция render первым параметром принимает компонент, который будет отрисован, а вторым — элемент, в который помещается компонент.

Если необходимо поместить несколько компонентов, они обязаны быть обернуты в один родительский компонент.

В данном случае, компонент `MuiThemeProvider` поставляется библиотекой компонентов «Material-ui». Компонент `Router` поставляется библиотекой `ReactRouter`. `React Router` – это набор навигационных компонентов. Компонент `App` – это основной компонент данного приложения.

4.5.1 КОМПОНЕНТ APP

Данный компонент, как и все компоненты, представляет собой класс расширяющий класс `Component`, предоставляемый `React`.

У этого класса есть конструктор и методы. В конструкторе определяется начальное состояние компонента и свойства, переданные ему.

```
constructor(props) {
  super(props);
  this.state = {
    isAuthenticated: false,
    user: {},
    isAdmin: false
  };
}
```

Метод `ComponentWillMount` – метод, определяющий одну из стадий жизненного цикла компонентов `React`, а именно когда компонент будет примонтирован.

```
componentWillMount(){
  var token=localStorage.getItem('cks_token');
  var user=localStorage.getItem('user');
  if(token===''| token===null|| user===''| user===null) this.changeAuth(false);
  else this.changeAuth(true, JSON.parse(user));
}
```

Токен для аутентификации хранится в локальном хранилище браузера. Поэтому при монтировании главного компонента приложения, необходимо проверить есть ли токен и информация о пользователе, чтобы проверить, аутентифицирован ли данный пользователь.

Но главным методом является `render`. Данный метод обязан возвращать компонент или функцию, возвращающую компонент.

```
<div>

  <Header isAuthenticated={this.state.isAuthenticated} search={this.search}/>

  <div className="Body">

    <Menu isAdmin={this.state.isAdmin}/>

    <Route path="/home" component={Home}/>

    <Route path="/signIn" render={()=><SignIn changeAuth={this.changeAuth}/>}/>

    <Route exact path="/books" render={({match})=><Books
      isAdmin={this.state.isAdmin} match={match}/>}/>

    <Route exact path="/authors" render={({match})=><Authors match={match}/>}/>

    <Route exact path="/genres" render={({match})=><Genres match={match}/>}/>

    <Route exact path="/search/:searchString" render={({match})=><Books
      isAdmin={this.state.isAdmin} match={match}
      searchString={this.state.searchString}/>}/>

    <Route path="/authors/:author" render={({match})=><Books
      isAdmin={this.state.isAdmin} match={match}/>}/>

    <Route path="/genres/:genre" render={({match})=><Books
      isAdmin={this.state.isAdmin} match={match}/>}/>

    <Route path="/book/:ISBN_code" render={({match})=><Book
      isAdmin={this.state.isAdmin} match={match}/>}/>

    <Route path="/signOut" render={()=><SignOut
      changeAuth={this.changeAuth}/>}/>

    <Route path="/admin" component={this.checkAuth(()=><Admin/>)}>

    <Route exact path="/orders" component={this.checkAuth(()=><Orders/>)}>

    <Route exact path="/orders/:id"
      component={this.checkAuth(({match})=><OrderDetails match={match}/>)}>

  </div>

</div>
```

Таким образом, есть заголовок страницы, компонент `Header`, который отображается всегда, и тело страницы, в котором отображаются различные элементы в зависимости от адреса.

В теле находится компонент Menu, который меняется в зависимости от того, какими правами обладает пользователь. Тут же находятся компоненты Route, предоставленные ReactRouter. Данные компоненты обеспечивают маршрутизацию в приложении. В зависимости от того, какая ссылка открыта в определенный момент времени, они возвращают компонент, переданный им в свойство component или render.

4.5.2 КОМПОНЕНТ HEADER

Компонент Header возвращает Toolbar – готовый компонент от «Material-ui», в который помещена строка поиска, и ссылки для перехода на форму входа и главную страницу.

```
<Toolbar style={{background: '#90CAF9'}}>
  <ToolbarTitle text="Books" />
  <ToolbarGroup>
    <TextField hintText="Search" value={this.state.searchString}
      onChange={this.handleSearchStringChange} />
    <FloatingActionButton mini={true} >
      <Link to={'/search/'+this.state.searchString}><SearchIcon /></Link>
    </FloatingActionButton>
    <FlatButton><Link to={signInBtnTxt}>
      {signInBtnTxt}</Link></FlatButton>
    <FlatButton className={this.props.isAuthenticated ? '':'none'}><Link
      to='/home'>Home</Link></FlatButton>
  </ToolbarGroup>
</Toolbar>
```

Значение signInBtnTxt меняется в зависимости от того, аутентифицирован ли пользователь. Если да, то ссылка меняется на значение «Sign Out», если же нет, то текст принимает значение «Sign In» и данная ссылка ведет на форму входа. Кнопка Home отображается только если пользователь вошел в систему.

4.5.3 ОТОБРАЖЕНИЕ СПИСКА КНИГ

За отображение списка книг отвечает компонент Books. В своем состоянии он хранит массив книг, для отображения. Когда компонент монтируется, то делает запрос на сервер в соответствии со ссылкой, по которой отображается.

```
tryFetch=(callback)=>{
    var address='http://localhost:8080/books';
    if(this.props.match.url===''/books') address='http://localhost:8080';

    fetch(address+this.props.match.url).then(function(response){
        return response.json();
    })
    .then(function(myJson){
        callback(null, myJson);
    }).catch(function(err){
        // callback(err);
    })
}
```

Затем результат запроса устанавливается в состояние компонента и отображается массив карточек, на которых указана каткая информация о книге.

```
BooksTemplate=this.state.books.map(function(item,index){
    return (
        <div className='Book' key={'Book'+index}>
            <Card>
                <CardHeader title={item.name} subtitle={item.authors.join(', ')}/>
                <CardActions>
                    <FlatButton><Link to={'/book/'+item.ISBN_code}>Show
more</Link></FlatButton>
                </CardActions>
                <CardText >
                    <p>ISBN code: {item.ISBN_code}</p>
                    <p>Genres: {item.genres.join(', ')}</p>
                    <p>Keywords: {item.keywords? item.keywords.join(', '): ''}</p>
                </CardText>
            </Card>
        </div>
    )
})
```

```

    </Card>
  </div>)

```

4.5.4 ЗАКАЗЫ

За отображение списка заказов отвечает контейнер Orders и компонент OrdersList. У контейнера Orders есть 2 поля состояния: orders – массив полученных заказов, и filter – фильтр для заказов.

Так же, как и компонент Books, при монтировании данного контейнера происходит запрос к серверу в зависимости от установленного фильтра, который задается значением элементов «Radio Button».

```

render() {
  var ordersTemplate = <p>No orders</p>;
  if(this.state.orders){
    ordersTemplate=<OrderList orders={this.state.orders}
      updateList={this.updateList}/>
  }
  return(
    <div className='Orders' >
      <RadioButtonGroup className='RB' name="shipSpeed" defaultSelected="/"
        onChange={this.handleChange}>
        <RadioButton value="/" label="All" />
        <RadioButton value="ordered" label="Ordered" />
        <RadioButton value="returned" label="Returned" />
        <RadioButton value="taken" label="Taken"/>
      </RadioButtonGroup>
      {ordersTemplate}
    </div>)}

```

Если есть заказы, то вызывается компонент OrderList, который отвечает за непосредственное отображение заказов.

ЗАКЛЮЧЕНИЕ

В рамках данной курсовой работы было реализовано приложение «Система заказа книг в библиотеке». Смоделированы логика и интерфейс приложения. Изучены способы разработки web-приложений, и особенности их разработки на Node.js. Получены навыки работы с документо-ориентированными базами данных, а именно с MongoDB. Изучены принципы проектирования архитектуры приложений в стиле REST и написания одностраничных приложений с помощью библиотеки React. А так же рассмотрены способы аутентификации пользователей на основе JSON Web Token

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Веб-приложение//*Википедия*. [Электронный ресурс]. URL: <https://ru.wikipedia.org/wiki/Вэб-приложение>. (дата обращения: 6.5.2017 г.)
2. Архитектура REST//*Habrahabr*. [Электронный ресурс]. URL: <https://habrahabr.ru/post/38730/>. (дата обращения: 6.5.2017 г.)
3. AJAX//*Википедия*. [Электронный ресурс]. URL: <https://ru.wikipedia.org/wiki/AJAX>. (дата обращения: 6.5.2017 г.)
4. Документоориентированная_СУБД//*Википедия*. [Электронный ресурс]. URL: https://ru.wikipedia.org/wiki/Документоориентированная_СУБД. (дата обращения: 3.5.2017 г.)
5. JavaScript//*Википедия*. [Электронный ресурс]. URL: <https://ru.wikipedia.org/wiki/JavaScript>. (дата обращения: 3.5.2017 г.)
6. Node.js// *Википедия*. [Электронный ресурс]. URL: <https://ru.wikipedia.org/wiki/Node.js>. (дата обращения: 3.5.2017 г.)
7. Express.js//*Википедия*. [Электронный ресурс]. URL: <https://ru.wikipedia.org/wiki/Express.js>. (дата обращения: 2.5.2017 г.)
8. Основы работы с модулями в Node.js// *Habrahabr*. [Электронный ресурс]. URL: <https://habrahabr.ru/post/233827/>. (дата обращения: 2.5.2017 г.)
9. MongoDB//*MongoDB*. [Электронный ресурс]. URL: <https://www.mongodb.com/what-is-mongodb>. (дата обращения: 3.5.2017 г.)
10. Mongoose//*Mongoose.js*. [Электронный ресурс]. URL: <http://mongoosejs.com/>. (дата обращения: 3.5.2017 г.)