**РЕФЕРАТ**

WEB-ПРИЛОЖЕНИЕ, ДИАГРАММА ВАРИАНТОВ ИСПОЛЬЗОВАНИЯ, КОНЦЕПТУАЛЬНАЯ МОДЕЛЬ, ФИЗИЧЕСКАЯ МОДЕЛЬ БАЗЫ ДАННЫХ, ЛОГИЧЕСКАЯ МОДЕЛЬ БАЗЫ ДАННЫХ

Объектом исследования является работа кафедры.

Цель проекта – разработать WEB-приложение предназначенное для чтения книг «Электронная книга».

В процессе работы (проектирования) выполнены следующие исследования (разработки):

1. построена диаграмма вариантов использования;
2. разработаны концептуальная, логическая и физическая модели БД;
3. спроектирован пользовательский интерфейс;
4. разработано приложение;
5. разработаны тестовые случаи для функционального и углубленного типов тестирования.

Элементами практической значимости полученных результатов являются:

1. надежное и эффективное web-приложение;
2. быстрота web-приложения;
3. современный дизайн.

Областью возможного практического применения является использование в высших учебных заведениях.

Студент-дипломник подтверждает, что приведенный в дипломном проекте расчетно-аналитический материал объективно отражает состояние исследуемого процесса (разрабатываемого объекта), все заимствованные из литературных и других источников теоретические и методологические положения и концепции сопровождаются ссылками на их авторов.

Дипломный проект: 80 с., 40 рис., 16 табл., 15 источников, 2 прил.

ОГЛАВЛЕНИЕ

ВЕДОМОСТЬ ОБЪЕМА ДИПЛОМНОГО ПРОЕКТА

[ПЕРЕЧЕНЬ УСЛОВНЫХ ОБОЗНАЧЕНИЙ И ТЕРМИНОВ 7](#_Toc73133706)

[ВВЕДЕНИЕ 8](#_Toc73133707)

[1 ОБЗОР СОСТОЯНИЯ ВОПРОСА 10](#_Toc73133708)

[1.1 Обзор существующих аналогов 11](#_Toc73133709)

[1.2 React 11](#_Toc73133710)

[1.3 Webpack 13](#_Toc73133711)

[1.4 Express 14](#_Toc73133712)

[1.5 MongoDB 14](#_Toc73133713)

[1.6 Краткий обзор используемых источников 16](#_Toc73133714)

[2 ЛОГИЧЕСКОЕ МОДЕЛИРОВАНИЕ 18](#_Toc73133715)

[2.1 Модель вариантов использования 18](#_Toc73133716)

[2.1.1 Список действующих лиц 18](#_Toc73133717)

[2.1.2 Список вариантов использования 18](#_Toc73133718)

[2.1.2 Диаграммы вариантов использования 19](#_Toc73133719)

[3 ФИЗИЧЕСКОЕ МОДЕЛИРОВАНИЕ 20](#_Toc73133720)

[3.1 Взаимодействие компонентов распределенного приложения 20](#_Toc73133721)

[3.2 Модель данных 21](#_Toc73133722)

[4 РЕАЛИЗАЦИЯ И ТЕСТИРОВАНИЕ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ 24](#_Toc73133723)

[4.1 Архитектура клиентской части проекта 24](#_Toc73133724)

[4.1.1 Компоненты 26](#_Toc73133725)

[4.2 Архитектура серверной части проекта 28](#_Toc73133726)

[4.3 Библиотеки 29](#_Toc73133727)

[4.4 Тестирование программного обеспечения 30](#_Toc73133728)

[4.4.1 Тестирование программного обеспечения 30](#_Toc73133729)

[4.4.2 Критическое тестирование 30](#_Toc73133730)

[4.4.3 Углубленное тестирование 32](#_Toc73133731)

[4.4.4 Результаты тестирования 35](#_Toc73133732)

[5 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ 36](#_Toc73133733)

[6 ТЕХНИКО-ЭКОНОМИЧЕСКОЕ ОБОСНОВАНИЕ ВЕБ-ПРИЛОЖЕНИЯ 45](#_Toc73133734)

[6.1 Определение единовременных затрат на создание продукта 45](#_Toc73133735)

[6.1.1 Определение трудоемкости разработки программного продукта 45](#_Toc73133736)

[6.1.2 Определение себестоимости создания программного продукта 48](#_Toc73133737)

[6.1.3 Определение оптовой и отпускной цены программного продукта 49](#_Toc73133738)

[6.1.4. Определение стоимости машино–часа работы персонального компьютера 50](#_Toc73133739)

[6.2 Определение ожидаемого прироста прибыли в результате внедрения программного продукта 53](#_Toc73133740)

[6.2.1 Определение готовых эксплуатационных расходов при ручном решении задачи 54](#_Toc73133741)

[6.2.2 Определение годовых затрат, связанных с эксплуатацией задачи 54](#_Toc73133742)

[6.2.3 Определение ожидаемого прироста прибыли в результате внедрения программного продукта 56](#_Toc73133743)

[6.3 Расчёт показателей эффективности использования программного продукта 56](#_Toc73133744)

[7 ОХРАНА ТРУДА 59](#_Toc73133745)

[7.1 Производственная санитария, техника безопасности и пожарная профилактика 59](#_Toc73133746)

[7.1.1 Метеоусловия 59](#_Toc73133747)

[7.1.2 Вентиляция и отопление 60](#_Toc73133748)

[7.1.3 Освещение 62](#_Toc73133749)

[7.1.4 Шум 62](#_Toc73133750)

[7.1.5 Электробезопасность 63](#_Toc73133751)

[7.1.6 Излучение 63](#_Toc73133752)

[7.1.7 Пожарная безопасность 64](#_Toc73133753)

[7.2. Требования к организации режима труда и отдыха пользователей при работе с ВДТ, ЭВМ и ПЭВМ. 65](#_Toc73133754)

[ЗАКЛЮЧЕНИЕ 69](#_Toc73133755)

[СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ 70](#_Toc73133756)

[ПРИЛОЖЕНИЕ A 81](#_Toc73133757)

[ПРИЛОЖЕНИЕ Б 87](#_Toc73133758)

# ПЕРЕЧЕНЬ УСЛОВНЫХ ОБОЗНАЧЕНИЙ И ТЕРМИНОВ

В данной пояснительной записке используются следующие условные обозначения и термины:

1. JSON – текстовый формат обмена данными, основанный на JavaScript;
2. CSS – каскадные таблицы стилей;
3. DOM – объектная модель документа;
4. HTML – язык разметки гипертекста;
5. БД – база данных;
6. NPM – пакетный менеджер;
7. API – программный интерфейс приложения, интерфейс прикладного программирования.

# ВВЕДЕНИЕ

Web-технология полностью перевернула представления о работе с информацией, да и с компьютером вообще. Оказалось, что традиционные параметры развития вычислительной техники – производительность, пропускная способность, емкость запоминающих устройств – не учитывали главного «узкого места» системы – интерфейса с человеком.

Устаревший механизм взаимодействия человека с информационной системой сдерживал внедрение новых технологий и уменьшал выгоду от их применения. И только когда интерфейс между человеком и компьютером был упрощен до естественности восприятия обычным человеком, последовал беспрецедентный взрыв интереса к возможностям вычислительной техники.

Разработка устанавливаемых на операционную систему приложений позволяет добиться хорошей производительности благодаря многопоточности и при помощи использования компилируемых языков программирования. Но в таком подходе есть минус после переустановки системы к примеру придется заново скачивать и устанавливать все приложения и системные администраторы тратят много времени на установку и настройку приложений в рамках коммерческих компаний.

Скорость разработки программного обеспечения играет важную роль в процессе разработки. Технологии развиваются очень стремительно, начиная писать программу на одном стандарте, могут выходить новые улучшенные стандарты в конце концов процесс поддержки последнего стандарта может быть невозможен из за сложного и долгово процесса миграции. В связи с этим удобно использовать браузер для разработки.

Создание веб-сайтов является одной из важнейших технологий разработки ресурсов Internet. Хороший сайт, вбирая в себя всю полезную информацию, является лучшей визитной карточкой и коммерческой фирмы и образовательного учреждения, работая на них в любое время суток.

Спрос на веб-приложения растет с каждым днем. Благодаря ядру KHML, процесс создания интерфейсов еще никогда не был так прост и быстр. А Java Script позволяет добавить логику на сайт: обработку событий, вычисление данных, асинхронные запросы и так делее.

Создание веб-приложений позволяет оптимизировать скорость загрузки, и убрать ненужные перерисовки страницы. На данный момент существует 3 основных библиотеки для создания Web-приложений: React, Vue, Angular.

Для данного проекта была выбрана библиотека React так как эта библиотека обеспечивает лучшую производительность, при условии правильного построения компонентов. Для создания React приложения, используется компонентно ориетированный подход, Компоненты внутри себя могут содеражать другие компоненты. В итоге получается дерево React компонентов схожее на DOM дерево элементов, построение дерева необходимо для сравнения компонентов, так библиотека React понимает какие компоненты были изменены и что следует перерисовать.

В настоящее время популярны социальные сети. Они занимают большой обьем рынка в сфере развлечений, статистика показала что люди тратят много свободного времени на эти приложения. К сожалению социальные сети несут минимум полезности, а найти достоверную информацию и убедится в ее подлинности практически невозможно. Читая книги человек развивается и получает ту истинную информауцию, которую хотел донести автор, иными словами первоисточник.

Бумажные издания стали просто не практичны, а мобильный телефон всегда с собой. Люди будут пользоваться приложением и читать книги если оно будет иметь интуитивно понятный дизайн, и будет просто в использовании.

«Электронная книга» – это клиент-серверное веб-приложение, которое предоставляет удобной интерфейс для взаимодействия с книгами в формате PDF. В отличии от своих аналагов, данное приложение предоставляет возможность загружать и публиковать свои собственные книги. Также еще одна особенность, это кросплатформенность. Данное приложение можно открыть с любого устройства на котором установлен браузер, дизайн на разных экранах выглядит по разному, это сделано для удобства чтения и восприятия контента. Приложение сохраняет загруженные книги на облаке, поэтому есть возможность загрузить книгу с одного устройства, и открыть эту же книгу с другого устройства.

# 1 ОБЗОР СОСТОЯНИЯ ВОПРОСА

Веб-приложения стали довольно популярны в последнее время. Для того чтобы открыть веб-приложение не нужно ничего устанавливать, достаточно открыть страницу в браузере. Веб-приложение не нужно обновлять, оно получит самую последнию версию просто при повторном открытии вкладки.

Браузер позволяет решать множество различных проблем. Веб-приложения это действительно кроссплатформенные приложения так как для их запуска на устройстве не нужно устанавливать ничего лишнего. Можно открыть один тот же сайт на разных устройствах с разными типоми экранов. Благодаря высокой популярности постоянно разрабатываются новые библиотеки, фреймворки, средства разработки и прочее.

В настоящее время существует множество электронных книг, причем каждая из них имеет как очевидные преимущества, так и недостатки.

Приложение «Электронная книга» – это приложение для чтения книг. Оно должно позволять пользователям : загружать книги и читать их. Одним из главных отличий от аналагов является то что можно разместить в публичный доступ свою книгу, чтобы ее смогли загружать и читать другие пользователи. Сам процесс чтения книги должен быть удобен.

Основные требования к новому веб-приложению:

* оно должно быть одностраничным, то есть при выборе другого пункта меню страница в браузере не должна меняться и перезагружаться, при этом заголовок сайта и меню должны быть статичными, и заменяться должен компонент страницы, содержащий элементы в соответствии с выбранным пунктом меню;
* интерфейс приложения должен быть интуитивно понятным и дружелюбным, а также адаптивным под различные размеры экранов;
* возможность загружать собственные книги на серве;
* поддерживать формат PDF для книг;
* скорость загрузки должна быть приемлема и не превышать среднее время ожидания пользователей в интернете;
* возможность выкладывать книгу в общественный доступ, с возможностью ее загружать;
* возможность отмечать понравившиеся страницы выбранной книги;
* возможность выделять текст и отмечать понравившиеся цитаты выбранной книги;
* возможность изменять масштаб текста в выбранной книге.
* возможность цвет текста в выбранной книге.
* возможность изменять формат отображения книги.

## 1.1 Обзор существующих аналогов

Анализ наиболее распространенных приложений для чтения книг (bookash.pro, ЛитРес, Calibre, Amazon Kindle, Apple Books, Cool Reader, FBReader, Adobe Digital Edition, Sumatra PDF, Moon+ Reader, Read Era и др.) позволяет сделать вывод о многообразии предлагаемых функциональных возможностей, о разных подходах к реализации, о возможности индивидуальной компоновк, о разных схемах лицензирования и стоимости. Все это свидетельствует о том, что классификацию таких систем необходимо проводить по определенным признакам, в рамках которых будут рассматриваться однотипные системы. Набор требуемых для конкретного пользователя признаков сформирует критерии выбора наилучшей для него системы.

Bookash.pro – это сайт, имеющий ряд недостатков начиная от семантики, дизайна и заканчивая функциональностью. Данный сайт использует встраивоемое приложение ЛитРес для предоставления возможности открытия и чтения книги. Использование на сайте встраиваемого приложения не является хорошей практикой, такое приложение сложно тестировать, а при обновлении встраивомго приложения поведения системы становится нестабильным.

ЛитРес – как было упомянуто ранее приложение может встраиваться в другие, однако не главная его особенность. Приложение имеет приятный и понятный дизайн. Информация хорошо структурированна. А также стоит отметить хороший обьем доступных книг. Имеются рекомендации а также сортировка книг по категориям. Любую книгу можно скачать либо читать онлайн после покупки. Минусом является то что все книги являются платными, нет возможности загружать собственные книги.

Приложения Calibre, Amazon Kindle, Cool Reader, FBReader, Adobe Digital Edition, Sumatra PDF, Moon+ Reader, Read Era – приложения которые необходимо устанавливать, некоторые из них имеют собственные библиотеки, а также есть возможность загружать собственные книги.

Apple Books – приложение достаточно хорошее и удовлетворяет почти всем требованиям по дизайну, интерфейсу и функциональности. Однако стоит отметить тот факт что Apple – компания монополист, и данное приложение невозможно открыть не на устройстве Apple. Устройства Apple занимают не более чем 30% рынка всех устойств. Как результат большая половина устройств не имеет достойных аналагов приложени Apple Books.

## 1.2 React

React – это JavaScript библиотека для построения пользовательских интерфейсов, разработанный программистами Facebook. Он предназначен для создания интерфейсов. Здесь важно учитывать, что React – это только система представления, в отличие от полноценного фреймворка, например, Angular.

React предоставляет разработчикам языковые шаблоны и функции обратной связи, позволяющие отрисовать HTML. Связки HTML-JS называются компонентами, которые запоминают свое внутреннее состояние. В React используется компонентный подход, что позволяет обходится без шаблонов, моделей, контроллеров и т.д.

Результат работы в React всегда будет зависеть от составляющих его компонентов. Особенностью этого фреймворка является JSX (сочетание Javascript и HTML). Что позволяет видеть происходящие изменения в одном месте. После обработки всех процессов в результате мы получаем чистый JS.

React может использоваться практически в любом проекте и подойдет как для крупного, так и малого бизнеса. Среди разработчиков существует немало споров по поводу использования этого продукта. Как и любой инструмент React имеет свои преимущества и недостатки.

Преимущества:

1) легко представить конечный результат. Посмотрев на исходный код, можно представить, как будет отрисован нужный компонент. Зная состояние кода, можно с точностью определить результат отрисовки. Это важно при работе в команде, так как не нужно отслеживать весь процесс разработки проекта;

2. простота использования. На React просто писать исходный код, и на нем же легко тестировать продукт;

3. cвязка JavaScript и HTML в JSX позволяет быстрее и проще работать с компонентами. Следить за многоуровневым потоком JS-HTML-JS не доставляет большого удовольствия программистам. Объединение функциональной части с разметкой и их упаковка в отдельный компонент сделает код лучше;

4. рендеринг на сервере. React можно визуализировать на сервере ещё до знакомства сайта с его потенциальными клиентами.

5. cоздание интерфейса (UI) происходит на основе отдельных компонентов. Это будущее веб-разработки.

Недостатки:

1. использование глобальных состояний. За счет компонентного подхода иногда глобальные состояния приходится пробрасывать сквозь 10 компонентов, а это трата времени на всего лишь одну константу;

2. некоторые классические приложения не поддерживают React. Чтобы избежать этой ситуации применяют дополнительные плагины и скачивают расширения. Однако, из-за большого веса, в React их лучше использовать по минимуму;

3. react занимает достаточно места из-за большого веса по сравнению с подобным ему VueJS.

Почему стоит использовать React?

1. прекрасно подходит для командной разработки, где осуществляется точное следование шаблону и соблюдение пользовательского интерфейса (UI);

2. ui код прост в сопровождении и легко читаем;

3. разработка интерфейса происходит на основе отдельных компонентов – это прогрессивная тенденция в современном программировании.

## 1.3 Webpack

Приложения на JavaScript постоянно меняются и усложняются. Для наиболее удобной работы используется «сборщик» (бандлер) front-end проектов - Webpack. Он имеет открытый исходный код и позволяет решать большой спектр задач. С помощью этого инструмента программисты могут собрать требуемые ресурсы для проекта и провести упаковку и компиляцию необходимых элементов.

С помощью Webpack используются как файлы своей системы, так и иные библиотеки. Такая модульная система лучше организует проект, который разбит на отдельные, маленькие части.

Достоинства:

1. благодаря Webpack в браузере используются различные модули http, events и т.д. Однако, если модуль все-таки несовместим с браузером, его можно заменить другим или оставить поле пустым – вариантов масса;

2. этот инструмент имеет большое разнообразие плагинов для обработки кода. Позволяет сгенерировать сразу несколько файлов, в которых содержится определенное число плагинов;

3. с помощью Вебпака может выполнятся дополнительная обработка первоначальных файлов, благодаря сторонним загрузчикам и плагинам. - Webpack легко вызывается из любого диспетчера задач;

4. возможность управлять и выполнять предварительную обработку данных проекта: файлы JS, таблицы стилей, изображений, шрифтов и так далее;

5. инструмент позволяет продвинуто разделять код на составные части, загрузка которых осуществляется тогда, когда это необходимо браузеру;

6. вебпак выбирают при работе с одностраничными приложениями.

Недостатки:

1. сложен для новичков. Для тех, кто уже давно знаком с Webpack, разобраться достаточно просто;

2. из-за того, что система постоянно меняется и дорабатывается, часть документации устаревает;

3. специфика работы с CSS-файлами на начальном этапе кажется очень запутанной.

Гибкость настройки Webpack и его потенциал в программировании стали очень привлекательными для разработчиков. Он используется также и для управления кодом, предназначенного для работы на стороне заказчика. Многие клиентские библиотеки поддерживают различные модули и плагины, что приоткрывает новые перспективы в разработке.

Webpack – гибко настраиваемая система, которая постоянно совершенствуется и дорабатывается. Этот модульный упаковщик позволяет упаковывать модули во едино для контроля над файлами.

## 1.4 Express

Express – это модульная веб-структура для Node.js, имеет открытый исход код. Он спроектирован для создания веб-приложений и API. Он используется для упрощения создания веб-приложений и сервисов.

Express используется на сервере. Конечно можно не использовать этот фреймворк вовсе, можно легко написать тоже самое на обычном Node JS, но потратить на это больше времени.

Этот фреймворк отлично подходит для небольших невысоконагруженных серверов на Node JS.

Redis – это система баз данных в памяти, известная своей быстрой представление. Вы можете использовать его с Express.js, используя redis клиент.

С использованием Express кодовая база на сервере выглядит структурированной и понятной.

Основные плюсы Express:

1. управляет маршрутами, обрабатывает запросы и представления;
2. библиотека легковесная;
3. ускоряет разработку;
4. хорошо подходит для работы с базой данных mongo DB.

Минусы использования Express:

1. нет серверного рендеринга;
2. использует функциональный подход;
3. малый набор встроеных функций.

## 1.5 MongoDB

MongoDB реализует новый подход к построению баз данных, где нет таблиц, схем, запросов SQL, внешних ключей и многих других вещей, которые присущи объектно-реляционным базам данных.

Ранее было обычным делом хранить все данные в реляционных базах данных (MS SQL, MySQL, Oracle, PostgresSQL). При этом было не столь важно, а подходят ли реляционные базы данных для хранения данного типа данных или нет.

В отличие от реляционных баз данных MongoDB предлагает документо-ориентированную модель данных, благодаря чему MongoDB работает быстрее, обладает лучшей масштабируемостью, ее легче использовать.

Но, даже учитывая все недостатки традиционных баз данных и достоинства MongoDB, важно понимать, что задачи бывают разные и методы их решения бывают разные. В какой-то ситуации MongoDB действительно улучшит производительность вашего приложения, например, если надо хранить сложные по структуре данные. В другой же ситуации лучше будет использовать традиционные реляционные базы данных. Кроме того, можно использовать смешенный подход: хранить один тип данных в MongoDB, а другой тип данных - в традиционных БД.

Вся система MongoDB может представлять не только одну базу данных, находящуюся на одном физическом сервере. Функциональность MongoDB позволяет расположить несколько баз данных на нескольких физических серверах, и эти базы данных смогут легко обмениваться данными и сохранять целостность.

Одним из популярных стандартов обмена данными и их хранения является JSON (JavaScript Object Notation). JSON эффективно описывает сложные по структуре данные. Способ хранения данных в MongoDB в этом плане похож на JSON, хотя формально JSON не используется. Для хранения в MongoDB применяется формат, который называется BSON (БиСон) или сокращение от binary JSON.

BSON позволяет работать с данными быстрее: быстрее выполняется поиск и обработка. Хотя надо отметить, что BSON в отличие от хранения данных в формате JSON имеет небольшой недостаток: в целом данные в JSON-формате занимают меньше места, чем в формате BSON, с другой стороны, данный недостаток с лихвой окупается скоростью.

MongoDB написана на C++, поэтому ее легко портировать на самые разные платформы. MongoDB может быть развернута на платформах Windows, Linux, MacOS, Solaris. Можно также загрузить исходный код и самому скомпилировать MongoDB, но рекомендуется использовать библиотеки с офсайта.

Если реляционные базы данных хранят строки, то MongoDB хранит документы. В отличие от строк документы могут хранить сложную по структуре информацию. Документ можно представить как хранилище ключей и значений.

Ключ представляет простую метку, с которым ассоциировано определенный кусок данных.

Однако при всех различиях есть одна особенность, которая сближает MongoDB и реляционные базы данных. В реляционных СУБД встречается такое понятие как первичный ключ. Это понятие описывает некий столбец, который имеет уникальные значения. В MongoDB для каждого документа имеется уникальный идентификатор, который называется \_id. И если явным образом не указать его значение, то MongoDB автоматически сгенерирует для него значение.

Каждому ключу сопоставляется определенное значение. Но здесь также надо учитывать одну особенность: если в реляционных базах есть четко очерченная структура, где есть поля, и если какое-то поле не имеет значение, ему (в зависимости от настроек конкретной бд) можно присвоить значение NULL. В MongoDB все иначе. Если какому-то ключу не сопоставлено значение, то этот ключ просто опускается в документе и не употребляется.

Отсутствие жесткой схемы базы данных и в связи с этим потребности при малейшем изменении концепции хранения данных пересоздавать эту схему значительно облегчают работу с базами данных MongoDB и дальнейшим их масштабированием. Кроме того, экономится время разработчиков. Им больше не надо думать о пересоздании базы данных и тратить время на построение сложных запросов.

## 1.6 Краткий обзор используемых источников

Главным источником является официальная документация фреймворка React на английском языке [14]. Здесь содержится вся необходимая для разработчика информация: инструкция по установке и запуску, описание доступных интерфейсов, примеры использования различных интерфейсов и компонентов представлены на рисунке 1.1:



Рисунок 1.1 – Веб-страница официальной документации React

Главным источником информации для библиотеки Redux является официальная документация на английском языке [15]. Здесь содержится вся необходимая для разработчика информация: инструкция по установке и запуску, примеры использования в связке с React. Также официальная документация описывает рекомендуемый стиль написания кода и предлагает использовать полезные библиотеки для ускорения процесса разработки и повышения качества и стабильности кода.

Для библиотеки Redux основным источником является официальная документация на английском языке. Здесь содержится вся необходимая для разработчика информация: инструкция по установке и запуску, примеры использования, подробная инструкция по настройке конфигурации. Единтсвенный минус данного источника, много избыточной информации, новичку легко запутаться и потерять много времени на изучение документации. А также данная документация полезна только для тех кто использует последнию версию WebPack, в противном случае при возникновении ошибок совместимости модулей, информацию в данной документации не найти.

# 2 ЛОГИЧЕСКОЕ МОДЕЛИРОВАНИЕ

## 2.1 Модель вариантов использования

На диаграммах вариантов использования отображается взаимодействие между вариантами использования, представляющими функции системы, и действующими лицами, представляющими людей или системы, получающие или передающие информацию в данную систему. Из диаграмм вариантов использования можно получить довольно много информации о системе. Этот тип диаграмм описывает общую функциональность системы.

Пользователи, менеджеры проектов, аналитики, разработчики, специалисты по контролю качества и все, кого интересует система в целом, могут, изучая диаграммы вариантов использования, понять, что система должна делать.

Достоинствами модели вариантов использования являются:

* определение пользователей и границы системы;
* эффективность общения заказчика и разработчика;
* использование при написании тестов;
* использование при написании пользовательской документации.

### 2.1.1 Список действующих лиц

В диаграммах вариантов использования пользователи, которые взаимодействуют с системой, называются акторами.

При анализе работы с курсовыми проектами были выделены следующие действующие лица:

* авторизованный пользователь;
* неавторизованный пользователь.

### 2.1.2 Список вариантов использования

Сценарий использования (вариант использования, прецедент использования) определяет взаимодействия между внешними агентами и системой, направленные на достижение цели. Актор представляет собой роль, которую играет человек или вещь, взаимодействуя с системой.

Сценарий использования должен:

* описывать, что именно нужно сделать, чтобы актор достиг своей цели;
* не затрагивать деталей реализации;
* иметь достаточный уровень детализации.

В сценариях будут также определены требования к данным, поступающим извне.

Сценарии использования приложения:

1. загрузка книги на сервер;
2. заполнение данных загружаемой книги;
3. сохранение последней прочитанной странице при входе на страницу книги;
4. загрузка книги в магазин книг;
5. загрузка книги из магазина книг;
6. удаление книги.

### 2.1.2 Диаграммы вариантов использования

Информация в графическом виде воспринимается гораздо проще, чем в текстовом, поэтому для сложных вариантов использования, связанных с размещением новых книг, разработаем соответствующую диаграмму, диаграмма представлена на рисунок 2.1:



Рисунок 2.1 – Диаграмма вариантов использования

Процесс использования приложения начинается с загрузки новой книги, прикрепления PDF файла данной книги а также заполнение полей «Название книги», «Автор». После того как книга загрузится, ее можно использовать, есть возможность удалить загруженную книгу.

Альтернативный вариант использования это загрузка книги из магазина, для этого необходимо выбрать книгу и нажать кнопку «Get» для загрузки данных книги.

# 3 ФИЗИЧЕСКОЕ МОДЕЛИРОВАНИЕ

## 3.1 Взаимодействие компонентов распределенного приложения

Приложение-сервер разработано с использованием технологии Node JS и использует базу данных MongoDB. Оно предоставляет нам внешний интерфейс в виде веб-сервисов, использующих протокол HTTP. Разработанное нами приложение-клиент будет посылать запросы этим сервисам. Такая архитектура взаимодействия компонентов распределенного приложения называется REST (от англ. Representational State Transfer – «передача состояния представления»). Схема архитектуры представлена на рисунке 3.1.

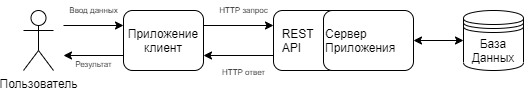


Рисунок 3.1 – REST архитектура

REST используется для создания легковесных, поддерживаемых и масштабируемых веб-сервисов. Сервис, построенный на REST архитектуре, называется RESTful-сервисом.

− rest – это способ получить доступ к ресурсам, которые находятся в определённой среде. Например, у вас может быть сервер с важными документами или фотографиями. Всё это – ресурсы. Если клиенту, скажем, веб-браузеру, нужны какието из этих ресурсов, ему необходимо отправить запрос на сервер для получения доступа к ним. REST определяет, как может осуществляться доступ к этим ресурсам. Ключевые составляющие реализации RESTful;

− ресурсы. Допустим, у нас есть сервер с записями о сотрудниках, а адрес вебприложения – http://server.com. Чтобы получить доступ к записи сотрудника, мы можем выполнить команду http://server.com/employee/1, которая говорит серверу предоставить запись сотрудника под номером 1;

− методы запросов. Они говорят, что вы хотите сделать с ресурсом. Браузер использует метод GET, чтобы проинформировать удалённую сторону о том, что он хочет получить данные. Кроме GET есть много других методов вроде POST, PUT и DELETE. В примере с http://server.com/employee/1 выше браузер на самом деле использует метод GET, поскольку он хочет получить данные о сотруднике;

− заголовки запроса. Это дополнительные инструкции, посылаемые вместе с 28 запросом;

− тело запроса. Это данные, отправляемые вместе с запросом. Данные обычно отправляются, когда выполняется POST-запрос к REST веб-сервису. Зачастую в POSTзапросе клиент говорит серверу, что он хочет добавить на него ресурс. Следовательно, тело запроса содержит подробную информацию о ресурсе, который необходимо добавить на сервер;

− тело ответа. Это основная часть ответа. В нашем примере на запрос http://server.com/employee/1 сервер мог бы прислать XML-документ с данными о сотруднике в теле ответа;

− коды ответа. Эти коды возвращаются сервером вместе с ответом. Например, код 200 обычно означает, что при отправке ответа не произошло никакой ошибки.

## 3.2 Модель данных

Рассмотрим модель данных. Так как база данных Mongo DB база даннх нереляционная то и отношения на диаграмме отсутствуют. Физическая модель данных представлена в виде диаграммы таблиц БД на рисунке 3.2.

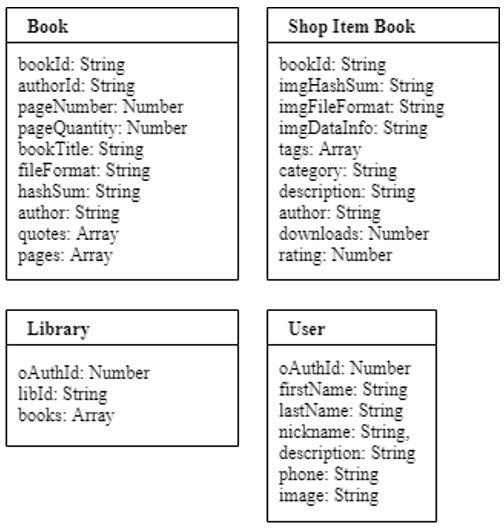


Рисунок 3.2 – Диаграмма таблиц базы данных

Таблица Book отражает сущность «Книга» и имеет столбцы bookId (Уникальный идентификатор книги), authorId (Уникальный идентификатор автора), pageNumber (Номер страницы), pageQuantity (Колличество страниц), bookTitle (Название книги), fileFormat (Формат файла), hashSum (Хеш-сумма), author (Автор), quotes (Циаты), pages (Страницы).

Таблица Library отражает сущность «Библиотека» и имеет столбцы oAuthId (Уникальный идентификатор пользователя), libId (Уникальный идентификатор библиотеки), books (Книги).

Таблица Shop Item Book отражает сущность «Элемент книги магазина» и имеет столбцы bookId (Уникальный идентификатор книги), imgHashSum (Хеш-сумма изображения), imgFileFormat (Формат файла изображения), imgDataInfo (Информация о изображении), tags (Теги), category (Категория), description (Описание), author (Автор), downloads (Скачивание), rating (Рейтинг).

Таблица User отражает сущность «Пользователь» и имеет столбцы oAuthId (Уникальный идентификатор пользователя), firstName (Имя), lastName (Фамилия), nickname (Краткое имя), description (Описание), phone (Телефон), image (Изображение).

Данные из БД наше React-приложение будет получать от приложения-сервера в структурированном виде в формате JSON.

На рисунке 3.3 показан ответ сервера на запрос на получение списка книг библиотеки.

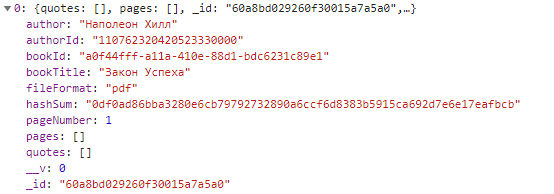


Рисунок 3.3 – Ответ сервера

Для хранения состояния на клиенте используется Redux. Данные хранятся в нормализованном виде.

Нарисунке 3.4 показан пример как выглядит Redux хранилище после загрузки одной книги.

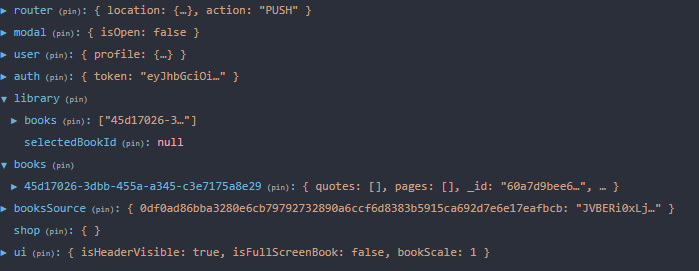


Рисунок 3.4 – Redux хранилище

В отличии от базы данных на сервере Redux, хранилище создается новое с каждым открытием страницы, и зполняется постепенно. Поэтому пользователь сразу увидит шаблон страницы, затем формы в которых будут данные, и только потом все нужные данные.

Для изменения состояния на клиентской части описывается «Reducer», это обычная JavaScript функция которая принимает измененные данные а также тип события, результатом является создание нового обьекта с обновленными данными. В последствии Redux фиксирует изменения и React запускает перерисовку всех компонентов, однако обновлены будут только компоненты, которые связаны с ранее изменной частью хранилища.

# 4 РЕАЛИЗАЦИЯ И ТЕСТИРОВАНИЕ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

## 4.1 Архитектура клиентской части проекта

Архитектура приложения обусловлена использованием React и Redux и является стандартной для программ, разработанных на базе данного стека. Она состоит из таких частей как Компонент (React Component), Css (Cтили), Action (Действие), Middleware (Функиция для промежуточных вычислений), Thunk (Функция позволяющая создавать новые действия), Reducer (Функция обработчик дейсвтий), State (Применение нового состояния), Connect (Функция, устанавливающая связь хранилища с компонентом). Архитектура приложения представлена представлена на рисунке 4.2:

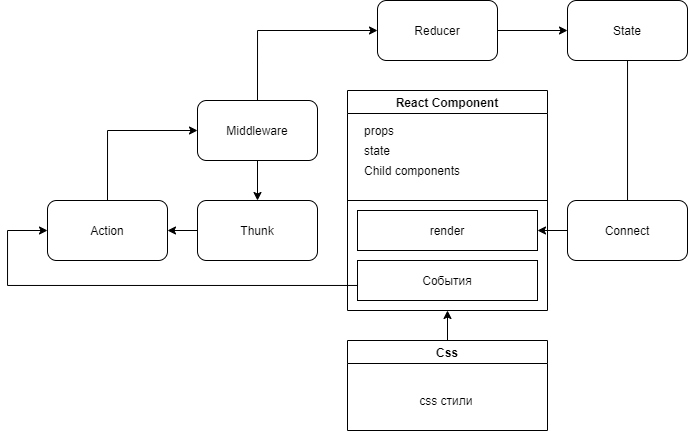


Рисунок 4.1 – Архитектура React Redux приложения

React использует компонентно ориентированный подход. React строит виртуальное DOM дерево из компонентов, после перерисовки одного компонента все дочерние компоненты будут так же обновляться в виртуальном DOM. Чтобы не перерисовывать лишнего React сравнивает виртуальный DOM и реальный, в итоге перерисовано будет только то что реально было изменено.

После того как происходит какое либо действие над компонентом, обработчик компонента может вызвать обновление глобального состояния посредством вызова функции «dispatch» и передав туда функию action. Благодаря функции «thunk», можно в функции action вызвать другие события.

Middleware в основном используется для обработки каких либо асинхронных действий или чтобы сделать промежуточные вычисления.

Всё хранилище разделено на разделы. Reducer отвечает за свой раздел. Для реализации необходимо описать обработчики событий по их типу. После любых изменений в хранилище, данные перезаписываются с обновлением ссылок на адреса памяти, это создано для того чтобы не происходило мутации данных и чтобы обезопасить от неявных повидений системы.

После обработки действия и обновления хранилища, срабатывает connect для всех компонентов. Чтобы не вызывать перерисовку всего приложения connect смотрит входные параметры: «mapStateToProps» (свойства локального состояния) и «mapDispatchToProps» (свойства обработчики хранилища). Для сравнения используется «Shallow Compare» (сравнение по значению или по ссылки для непримитивных типов данных).

Если компонент подписывается на обьект или массив, то при любом обновлении этого обьекта будет вызвана перерисовка компонента. Чтобы избавится от этого, сложные обьекты кэшируются таким образом при одинаковых входных данных кэш-функция возвращает ссылку на тот же обьект а не создает новое представление.

В React при использовании Component перерисовка будет всегда в не зависимости от измененных данных, поэтому в проекте используются только PureComponent которые обеспечивают «Shallow Compare», что было упомянуто выше.

Для управления модулями и настройками запуска используется NPM. Формат файла JSON, в нем описывается необходимые скрипты для запуска, сборки, тестов, а также указываются зависимостей используемых модулей.

На рисунке 4.2 приведена структура клиентской части проекта.

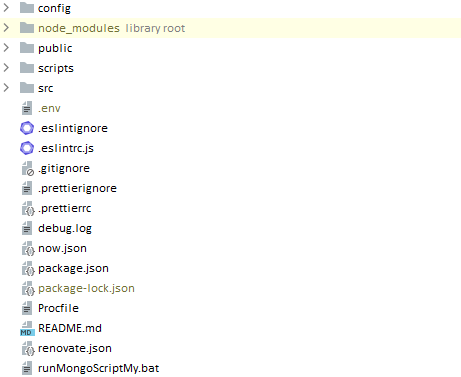


Рисунок 4.2 – структура клиентской части проекта

Основная логика и компненты находятся в папке src. В scripts указаны скрипты запуска и сборки. В config находятся webpack файл настройки. В public находится html файл шаблон.

### 4.1.1 Компоненты

Компонент является основным строительным блоком пользовательского интерфейса (UI). Каждый компонент сопоставлен с шаблоном (рисунок 4.3).

React-компонент содержит свойства, методы, конструктор, а также входные события, выходные события и методы жизненного цикла, такие как «componentDidMount», «componentWillUnmount» и т. д. React создает, обновляет и уничтожает компоненты при перемещении пользователя по приложению.

Реализация компонента Library представлена на рисунке 4.3



Рисунок 4.3 – Реализация компонента Libarary

Компонент Library – функциональный, поэтому он обернут в функцию React.memo, которая обеспечивает Shallow Compare (сравнение свойств по ссылке).

На рисунке видно что компоненты всегда возвращают другие компоненты и это могут быть как html-подобные компоненты так и собственные компоненты.

Компонентам можно передавать свойства. К примеру можно передать обработчик действия при помощи callback (функция обратного вызова) для того чтобы в контексте данного компонента произвести какое-либо действие.

Для того чтобы задать стили можно использовать атрибут style и передать обьект стилей, но это считается плохим способом для статичных стилей. Лучший способ задать стили в React использовать класс и через атрибут className передать css класс.

Современный стандарт React рекомендует использовать hooks (хуки). Это механизм реализующие жизненный цикл функционального компонента. К примеру хук useEffect вызывается после первой отрисовки компонента, когда компонент прикрепился а также можно передать массив вторым аргументом в котором указать переменные при изменении которых этот хук должен будет вызваться.

Также для связки компонента с redux используются функции useDispatch и useSelector. useDispatch – возвращает функцию dispatch при помощи которой можно вызвать обновление state. useSelector – функция хук, она принимает функцию collback с аргументом state.

Перечень всех компонентов приведен на рисунке 4.4

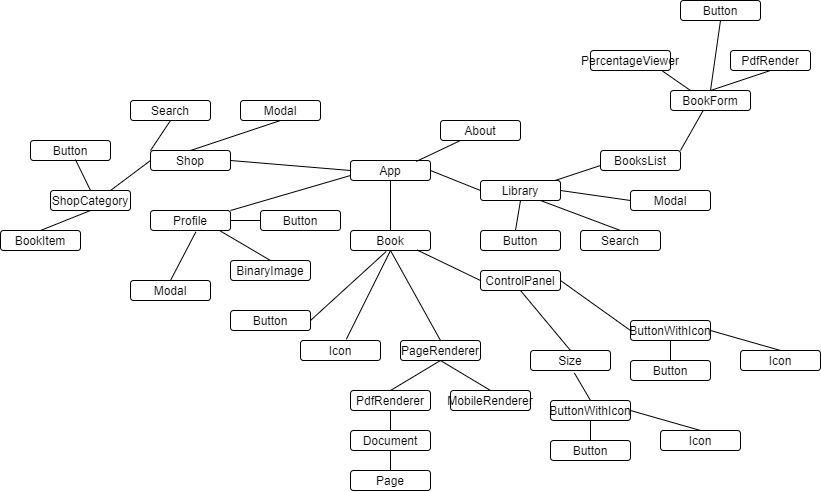


Рисунок 4.4 – Перечень компонентов приложения

Так как данное приложение является одностраничным, обмен данными с сервером не требует обновления страницы в браузере. Поэтому необходимо каким-то образом уведомлять пользователя, когда вызванные им операции выполняются. Для этого в момент загрузки данных будет показываться компонент Loader.

Чтобы не дублировать логику были разработаны общие компоненты, такие как: Button, Icon, Search, Input, PdfRender и т. д.

## 4.2 Архитектура серверной части проекта

Архитектура серверной части обусловлена использованием библиотеки express и базы данных Mongo DB.

Сервер содержит модули в которых обрабатываются конкретные route (маршруты). Каждый route характеризуется HTTP методом GET, POST, PUT, DELETE и т.д.

MongoDB содержит описанные модели, там указываются исходные данные а так же описываются поля их типы и ставится флаг обязательности поля.

На рисунке 4.5 приведена структура клиентской части проекта.

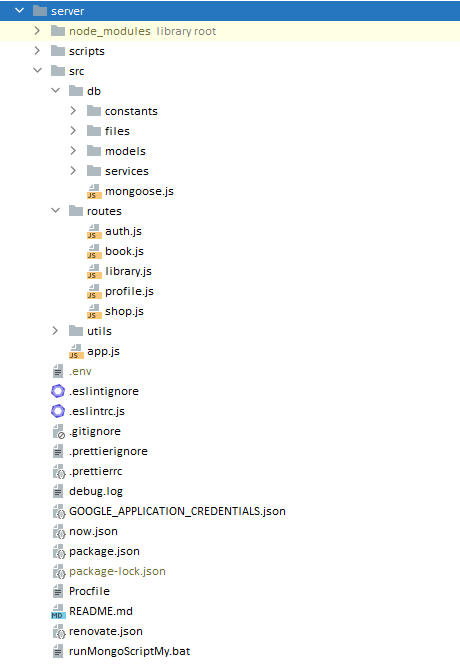


Рисунок 4.5 – Структура серверной части

Основная логика находятся в папке src. В scripts указаны скрипты запуска и сборки. В файле «package.json» размещается список использованных библиотек а также скрипты запуска.

## 4.3 Библиотеки

В данном приложение используется множество библиотек для улучшения качества кода а так же ускорения процесса написания кода.

Для взаимодействия с API используется библиотека Axios. Axios − это JavaScript-библиотека для выполнения либо HTTP-запросов в Node.js, либо XMLHttpRequests в браузере. Она поддерживает промисы (новинку стандарта ES6) и автоматически преобразует JSON-данные. Эта библиотека определяет ряд методов для отправки различного рода запросов: GET, POST, PUT, DELETE.

На рисункке 4.6 приведен пример использования библиотеки Axios.



Рисунок 4.6 – Пример использования библиотеки Axios

Axios умеет следить за ходом выгрузки данных. Fetch это не поддерживает. Это может стать решающим фактором выбора технологии для тех, кто разрабатывает приложение, позволяющее пользователям выгружать на сервер фотографии или видеофайлы.

С помощью Axios можно обойтись без передачи результатов HTTP-запроса в метод json(), как это делается в стадартной функции fetch. Axios возвращает именно тот объект с данными, который мы ожидаем.

На рисунке 4.7 приведен пример результата запроса.



Рисунок 4.7 – Пример результата запроса

Для утилитарных вещей таких как работа со списками или обьектами используется библиотека Ramda. В отличии от своего аналога Lodash, это библиотека имеет ряд особенностей. Библиотека использует функциональный «Data Last» подход и механизм создание сложной функциональной логики реализуется с помощью композиции. В Ramda все её функции автоматически каррированы. Если такой функции передать не все ожидамые аргументы, то она вернёт новую функцию, которая будет ожидать оставшиеся аргументы.

Автокаррирование функций Ramda сочетается с принципом «Вначале функции, потом данные». Передача данных в последнюю очередь делает Ramda очень простой библиотекой для работы в стиле функциональной композиции.

## 4.4 Тестирование программного обеспечения

Функциональное тестирование проводится для определения, насколько компонент или система соответствуют заданным функциональным требованиям, описанным в спецификациях.

Целью тестирования является поиск ошибок. Ошибка – это несоответствие ожидаемых результатов фактически полученным.

В рамках данного дипломного проекта будет проведено критическое и углубленное тестирование системы.

### 4.4.1 Тестирование программного обеспечения

Тестирование проводится на персональном компьютере с параметрами, указанными в таблице 4.1

Таблица 4.1 – Аппаратные средства, используемые при тестировании

|  |  |  |
| --- | --- | --- |
| Роль | Аппаратная конфигрурация | Программная конфигурация |
| Рабочая станция | AMD Ryzen 7 4800H with Radeon Graphics 2.90 GHz, RAM 16 GB, SSD 512 GB | Windows 10 Pro версия 1909 |

### 4.4.2 Критическое тестирование

Критическое тестирование – это процесс поиска ошибок в программе при стандартной ее работе (при правильной последовательности действий, при верном заполнении полей и т. д.). Этот вид тестирования проводят всегда над каждой версией продукта, по заранее подготовленным тестовым случаям. Если в процессе тестирования находятся ошибки непредусмотренные тестовыми случаями, то необходимо добавить новый тестовый случай, а также составить отчет об ошибке.

В таблице 4.2 приведен перечень граничных и эквивалентных значений для критического тестирования.

Таблица 4.2 – Граничные и эквивалентные значения для критического тестирования

|  |  |  |  |
| --- | --- | --- | --- |
| Поле | Формат данных | Граничные значения | Эквивалентные значения |
| 1 | 2 | 3 | 4 |
| Название книги | Обязательное поле, от 3 до 35 символов | Строки длинной:  3 символа 35 символов | Строки длинной:  4 символа 34 символов 15 символов |
| Автор книги | от 0 до 20 символов | Строки длинной:  0 символов 20 символов | Строки длинной:  1 символ 12 символов 19 символов |
| Файл книги | PDF файл, размером до 250mb | PDF файл, размером от 0 до 250mb | PDF файл, размером 10 mb 100mb 222mb |
| Описание книги | Обязательное поле, от 1 до 400 символов | Строки длинной:  1 символ 400 символов | Строки длинной:  7 символов 22 символов 365 символов |
| Поисковые теги | #имя#имя2 и т.д. | Начало строки с символа # | Начало строки с символа # |
| Файл изображение обложки | PNG, JPG, JPEG файл, размером до 250mb | PNG ,или JPG или JPEG файл, размером от 0 до 250mb | PNG файл размером10 mb JPG файл размером 110mb JPEG файл размером 212mb |
| Имя пользователя | Обязательное поле, от 3 до 35 символов | Строки длинной:  3 символа 35 символов | Строки длинной:  4 символа 34 символов 10 символов |
| Фамилия пользователя | Обязательное поле, от 3 до 35 символов | Строки длинной:  3 символа 35 символов | Строки длинной:  4 символа 32 символов 15 символов |
| Краткое имя пользователя | 3 до 35 символов | Строки длинной:  3 символа 35 символов | Строки длинной:  4 символа 32 символов 15 символов |
| Категория книги | список возможных вариантов | одна из категорий доступная в списке | одна из категорий доступная в списке |
| Цитата | 3 до 200 символов | Строки длинной:  3 символа 200 символов | Строки длинной:  4 символа 189 символов 132 символов |

В таблице 4.3 приведены примеры тестовых случаев для критического тестирования.

Таблица 4.3 – Тестовые случаи для критического тестирования

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № | Название тестового случая | Описание тестового случая | Ожидаемые результаты | Тест пройден? |
| 1 | 2 | 3 | 4 | 5 |
| 1 | Загрузка книги | 1. Перейти на стрницу "library". 2. Нажать на кнопку "add book". 3. В открывшемся модальном окне заолнить все поля допустимыми значениями. 4. Нажать на кнопку "apply". | Добавленная книга отображается в списке книг | Да |
| 2 | Обновление данных профиля | 1. Перейти на стрницу "profile". 2. Нажать на кнопку "edit profile". 3. В открывшемся модальном окне заолнить все поля допустимыми значениями. 4. Нажать на кнопку "apply". | Данные профиля обновлены и тображаются на странице | Да |
| 3 | Загрузка книги в магазин | 1. Перейти на стрницу "library". 2. Нажать на кнопку для публикации книги. 3. В открывшемся модальном окне заолнить все поля допустимыми значениями. 4. Нажать на кнопку "apply". | Книга загружена в магазин и отображается в списке доступных книг в своей категории на странице "book" | Да |

### 4.4.3 Углубленное тестирование

Углубленное тестирование – это процесс поиска ошибок в программе в нестандартных ситуациях, например, когда пользователь вводит данные, не соответствующие формату. В таких ситуациях система должна адекватно реагировать и выводить пользователю сообщение о том, что он неправильно ввел данные.

В таблице 4.4 приведен перечень граничных и эквивалентных значений для углубленного тестирования.

Таблица 4.4 – Граничные и эквивалентные значения для углубленного тестирования

|  |  |  |  |
| --- | --- | --- | --- |
| Поле | Формат данных | Граничные значения | Эквивалентные значения |
| 1 | 2 | 3 | 4 |
| Название книги | Обязательное поле, от 3 до 35 символов | Строки длинной:  3 символа 35 символов | Строки длинной:  4 символа 34 символов 15 символов |
| Автор книги | от 0 до 20 символов | Строки длинной:  0 символов 20 символов | Строки длинной:  1 символ 12 символов 19 символов |
| Файл книги | PDF файл, размером до 250mb | PDF файл, размером от 0 до 250mb | PDF файл, размером 10 mb 100mb 222mb |
| Описание книги | Обязательное поле, от 1 до 400 символов | Строки длинной:  1 символ 400 символов | Строки длинной:  7 символов 22 символов 365 символов |
| Поисковые теги | #имя#имя2 и т.д. | Начало строки с символа # | Начало строки с символа # |
| Файл изображение обложки | PNG, JPG, JPEG файл, размером до 250mb | PNG ,или JPG или JPEG файл, размером от 0 до 250mb | PNG файл размером10 mb JPG файл размером 110mb JPEG файл размером 212mb |
| Имя пользователя | Обязательное поле, от 3 до 35 символов | Строки длинной:  3 символа 35 символов | Строки длинной:  4 символа 34 символов 10 символов |
| Фамилия пользователя | Обязательное поле, от 3 до 35 символов | Строки длинной:  3 символа 35 символов | Строки длинной:  4 символа 32 символов 15 символов |
| Краткое имя пользователя | 3 до 35 символов | Строки длинной:  3 символа 35 символов | Строки длинной:  4 символа 32 символов 15 символов |

Продолжение таблицы 4.4

|  |  |  |  |
| --- | --- | --- | --- |
| 1 | 2 | 3 | 4 |
| Категория книги | список возможных вариантов | одна из категорий доступная в списке | одна из категорий доступная в списке |
| Цитата | 3 до 200 символов | Строки длинной:  3 символа 200 символов | Строки длинной:  4 символа 189 символов 132 символов |

В таблице 4.5 приведены примеры тестовых случаев для углубленного тестирования.

Таблица 4.5 – Пример тестовых случаев углубленного тестирования

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № | Название тестового случая | Описание тестового случая | Ожидаемые результаты | Тест пройден? |
| 1 | Загрузка книги | 1. Перейти на стрницу "library". 2. Нажать на кнопку "add book". 3. В открывшемся модальном окне заолнить все поля допустимыми значениями. | Добавленная книга отображается в списке книг | Да |
| 2 | Обновление данных профиля | 1. Перейти на стрницу "profile". 2. Нажать на кнопку "edit profile". 3. В открывшемся модальном окне заолнить все поля допустимыми значениями. | Данные профиля обновлены и тображаются на странице | Да |
| 3 | Загрузка книги в магазин | 1. Перейти на стрницу "library". 2. Нажать на кнопку для публикации книги. 3. В открывшемся модальном окне заолнить все поля допустимыми значениями. 4. Нажать на кнопку "apply". | Книга загружена в магазин и отображается в списке доступных книг в своей категории на странице "book" | Да |

### 4.4.4 Результаты тестирования

В процессе тестирования были выявлены ошибки. Список ошибок представлен в таблице 4.6.

Таблица 4.6 – Анализ ошибок

|  |  |  |  |
| --- | --- | --- | --- |
| Номер ошибки | Описание ошибки | Гипотеза местонахождения | Гипотеза прошла? Да/Нет |
| 1 | 2 | 3 | 4 |
| 1 | Возможно загрузить данные с некорректными данными в поле «Цитата» | Отсутствие валидации данных для поля «Цитата» | Да |
| 2 | Кнопка применить является активной при загрузки файла с неверным формматом | Отсутствие валидации данных для поля «PDF file» | Да |

Каждой ошибке присваивался свой уровень важности, в зависимости от влияния на работу приложения. Ошибки с более высоким приоритетом исправлялись в первую очередь.

Результаты тестирования приложения, статистика найденных и исправленных ошибок представлена в таблице 4.7.

Таблица 4.7 – Результаты тестирования приложения

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Критических | Важных | Средних | Незначительных | Всего |
| Найдено ошибок | 0 | 2 | 0 | 0 | 2 |
| Исправлено ошибок | 0 | 2 | 0 | 0 | 2 |

Тестирование приложения позволило выявить ряд ошибок, допущенных во время реализации. Критических ошибок, влияющих на работоспособность системы в целом, обнаружено не было. Все описанные выше ошибки были устранены.

# 5 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

Для доступа к приложению необходимо ввести в адресную строку веб-браузера адрес сайта и войти в систему. Неавторизованный пользователь попадет на страницу «about», на которой находится небольшое описание данного проекта. Для входа в систему необходимо кликнуть по ссылке «Click here to login with google» на рисунке 5.1 представлена приветственная страница.

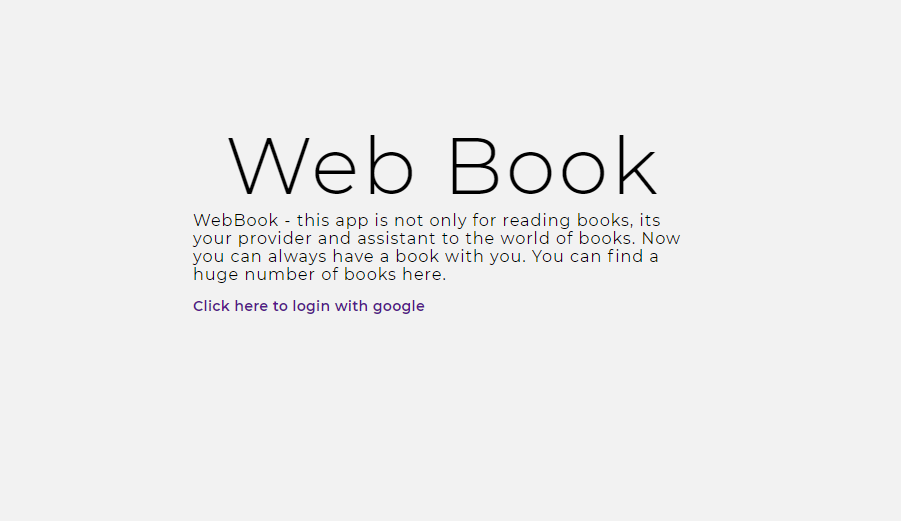


Рисунок 5.1 – Приветственная страница

Вход в систуему входит при помощи Google авторизации. Если у пользователя нет Google аккаунта, нужно его создать и авторизоваться в системе.

Для навигации по приложению пользователь может использовать меню, которое расположено сверху экрана пример приведен на рисунке 5.2.



Рисунок 5.2 – Меню, панель навигации приложения

После успешной авторизации в системе пользователь попадает на страницу профиля, на рисунке 5.3 представлена страница профиля. На странице профиля размещена информация об авторизованном пользователи имя, фамилия, имя в системе а так же фото профиля.

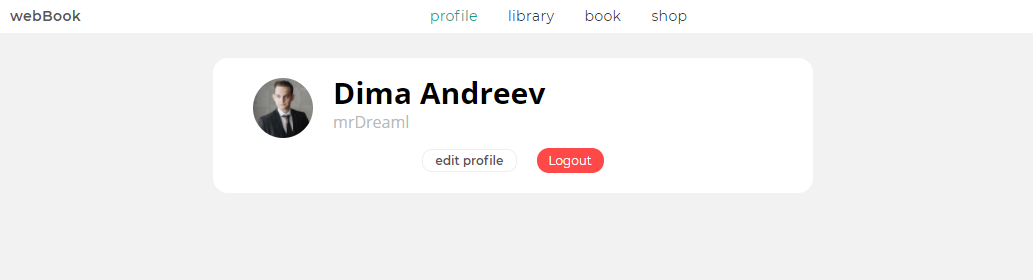


Рисунок 5.3 – Страница профиля

Для того чтобы выйти из системы нужно нажать на кнопку «Logout», после этого страница профиля и прочие не будут доступны и пользователь попадет на страницу «about».

Чтобы отредактировать информацию профиля необходимо нажать на кнопку «edit profile». После этого будет открыто модальное окно пример представлен на рисунке 5.4 в котором можно будет указать новые данные полей «Имя», «Фамилия», «Имя в системе» а также загрузить новое фото профиля. После заполнения всех полей нужно нажать на кнопу «apply» для применения обновлений.

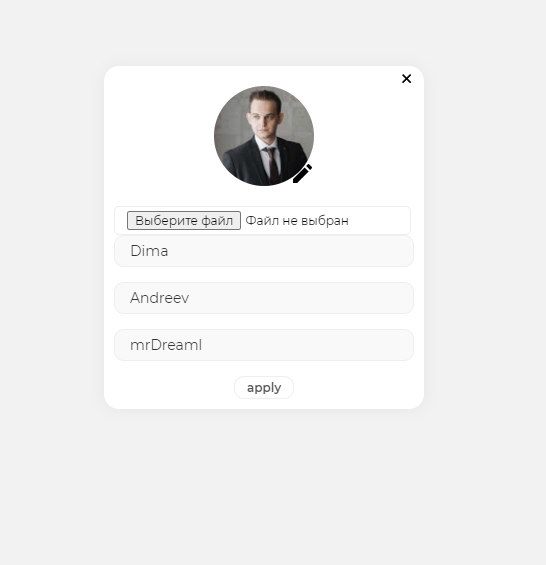


Рисунок 5.4 – Модальное окно редактирования профиля

Для того чтобы закрыть модальное окно можно кликнуть на крестик в правом верхнем углу либо кликнуть вне модальной формы.

После открытия страницы «library» пользователь увидит пустой список книг, форму поиска и кнопку «add book». Страница «library» представлена на рисунке 5.5.

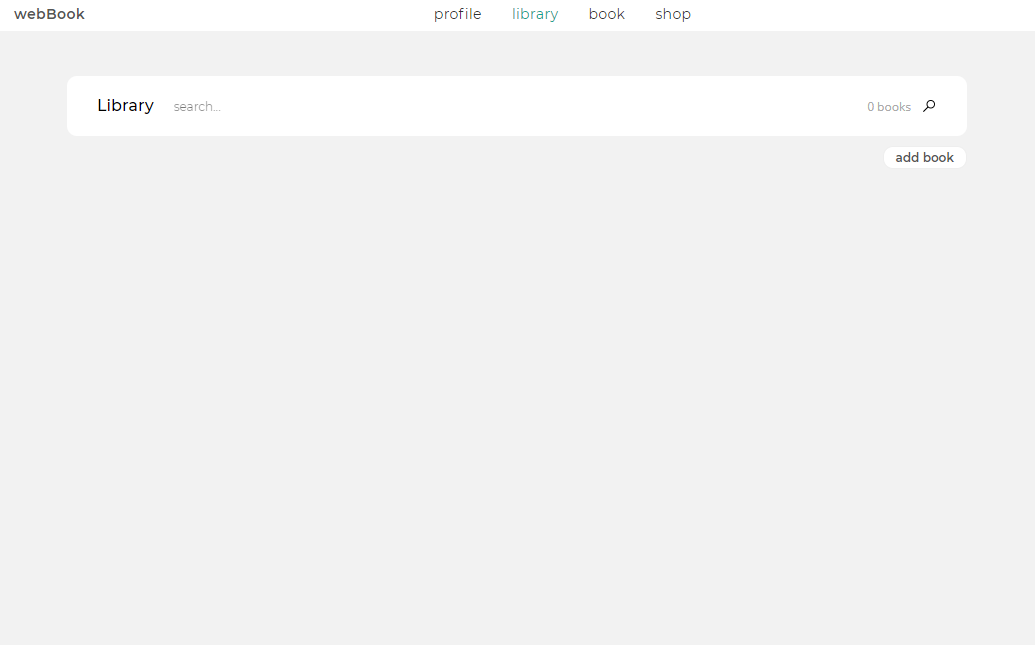


Рисунок 5.5 – Страница «library»

После нажатия на «add book» кнопку для добавления новой книги будет открыто модальное окно пример на рисунек 5.6. После заполнения всех полей и загрузки PDF-файла, будет активна кнопка «apply» после нажатия на которую будет произведена загрузка книги.

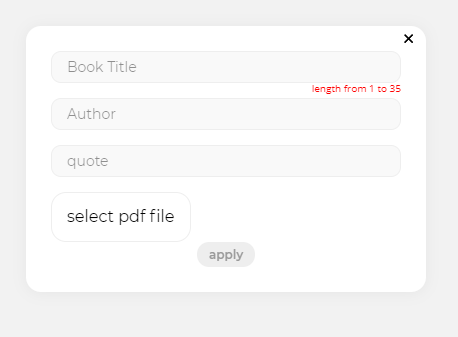


Рисунок 5.6 – Модальное окно для добавления новой книги

После успешной загрузки книги, она будет отображена (рисунок 5.7) располагаются все загруженные книги. Сверху экрана располагается форма пойска, поиск совершается по имени книги или автора. Результат пойска отображается на странице и показывыется колличесвто книг. Для того чтобы сбросить поиск и показать все доступные книги, необходимо стереть текст в поиске.

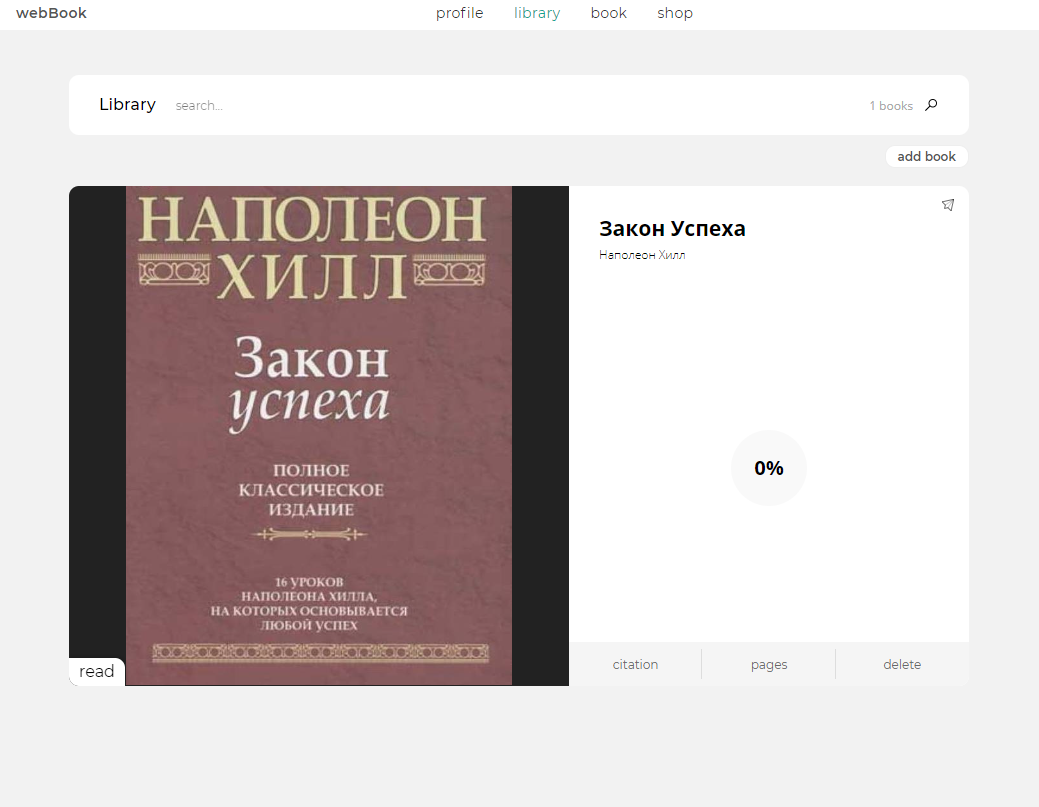


Рисунок 5.7 – Страница «library» после загрузки книги

Для удаления книги из библиотеки, нужно нажать на кнопку «delete». После нажатия на кнопку будет открыто модальное окно для подтверждения действия. Для удаления нужно нажать на кнопку «Yes». Для отмены удаления нужно кликнуть на крестик либо вне зоны модального окна. На рисунке 5.8 представлено модальное окно для удаления книги.

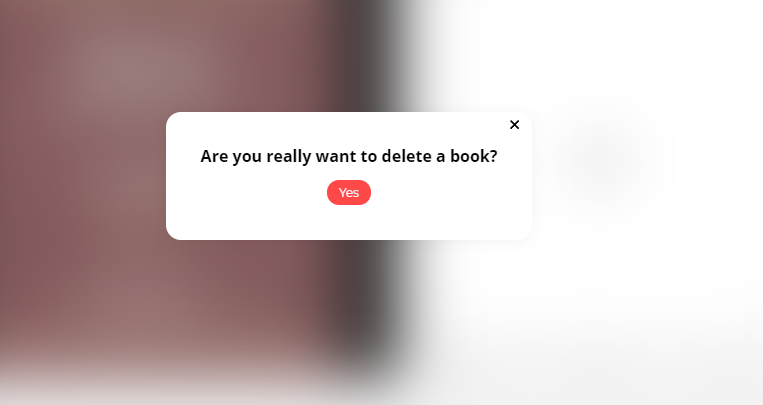


Рисунок 5.8 – Модальное окно удаления книги

Пользователю также доступен список цитат к данной книги. Чтобы открыть список цитат, необходимо кликнуть на кнопку «citation». Если цитаты отсутствуют будет показано соответствующее сообщение. На рисунке 5.9 представлено модальное окно цитат с пустым списком.

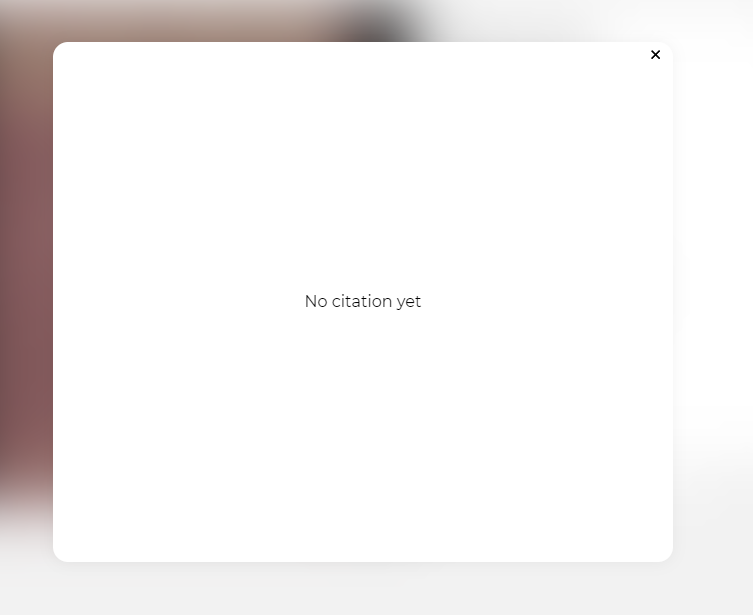


Рисунок 5.9 – Модальное окно цитат с пустым списоком

Если список цитат не пустой, он будет отображен пример представлен на рисунке 5.10. Для удаления цитаты, нужно кликнуть на крестик около соответствующей цитаты.

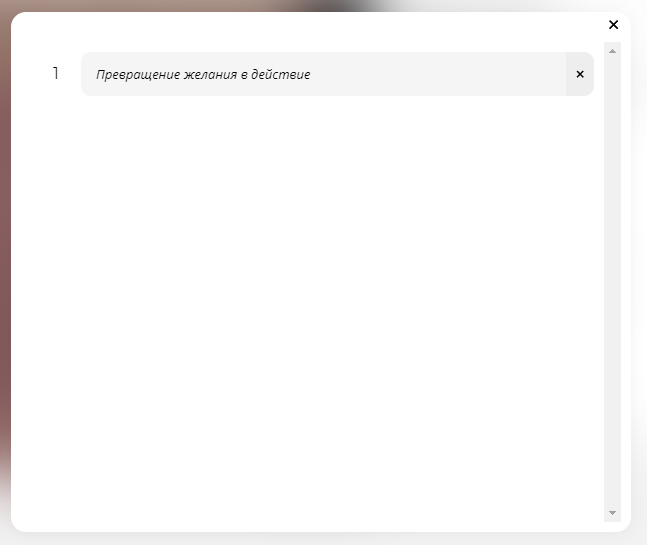


Рисунок 5.10 – Модальное окно цитат

Для того чтобы выбрать список сохраненных страниц необходимо кликнуть на кнопку «pages». После клика откроется модальное окно, при отсутствующих сохраненных страницах, список будет пустой и будет отображаться соответствующее сообщение. На рисунке 5.11 представлено модальное окно сохраненных страниц с пустым списком.

Для закртия модального окна нужно как во всех модальных окнах данного приложения кликнуть на крестик или в не зоны модального окна.

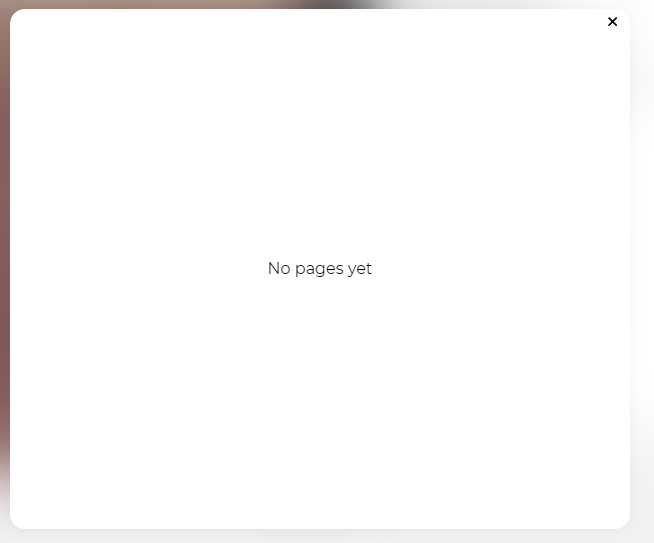


Рисунок 5.11 – Модальное окно сохраненных страниц с пустым списком

Если список сохраненных страниц не путой то они так же как и цитаты будет отображаться. На рисунке 5.12 представлено модальное окно сохраненных страниц. Можно удалить сохраненную страницу кликнув на крестик около конкретной сохраненной страницы. Имеется также возможность открыть книгу на этой странице, для этого нужно просто кликнуть на этот элемент.

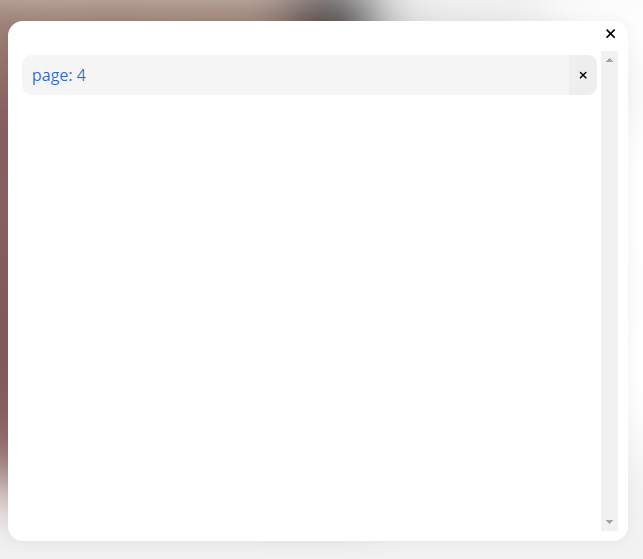


Рисунок 5.12 – Модальное окно сохраненных страниц

На странице библиотеки отображается вся необходимая информация о книге: название книги, автор книги, последняя сохраненная цитата, процент прочитанных книги а так же кол-во прочитанных страниц.

Для того чтобы разместить книгу в магазине необходимо кликнуть на иконку в верхнем правом углу книги. После клика будет открыто модальное окно, пример представлен на рисунке 5.13 в котором будут поля для заполнения такие как: название книги, изображение обложки, автор, описание, поисковые теги и категория.

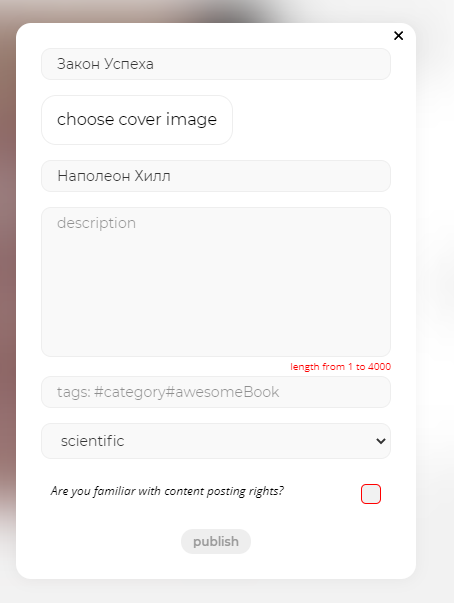


Рисунок 5.13 – Модальное окно загрузки книги в магазин

После заполнения всех полей кнопка «publish» станет активной, после клика на нее, модальное окно будет закрыто и книга добавится в магазин. Одно из обязательных условий для публикации книги, пользователь загружающий книгу должен являтся автором этой книги, либо должен иметь права на размещения данной книги.

Пользователь может добавить книгу из магазина. Для этого нужной перейти на страницу «shop». На рисунке 5.14 представлена страница «shop».

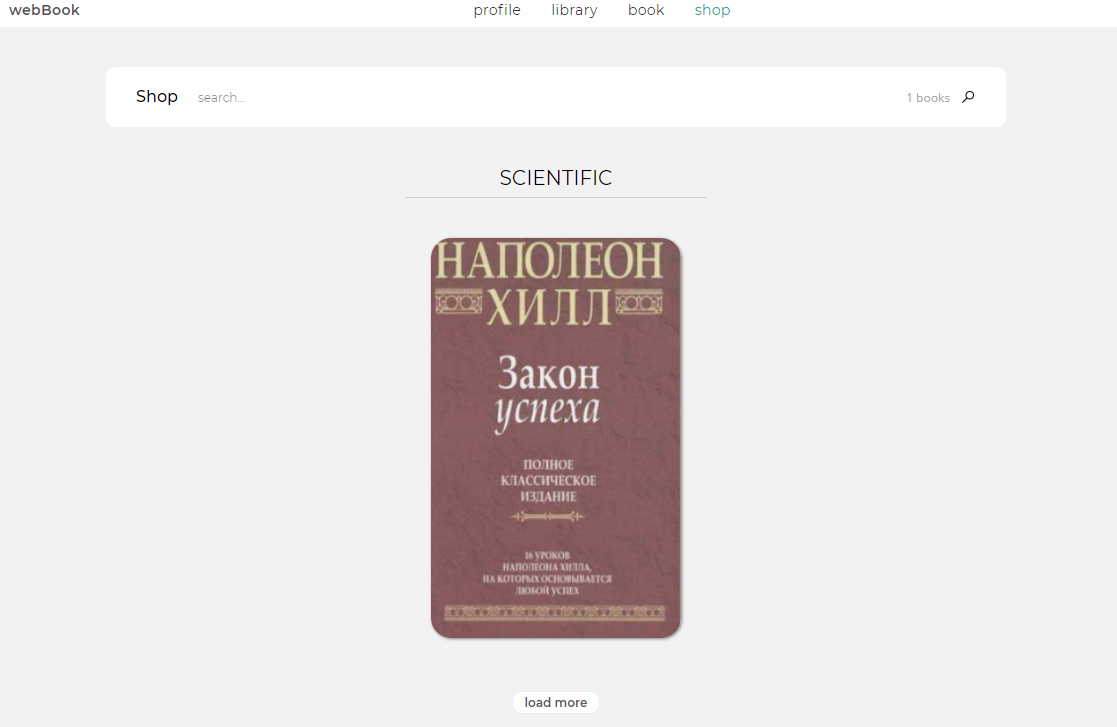


Рисунок 5.14 – Страница «shop»

На странице располгается форма поиска, которая осуществляет поиск по названию книге либо по пойсковым тегам. Книги отображаются в разделе своей категрии.

Для быстрого просмотра информации о книге можно навести на изображение обложки, пример изображен на рисунке 5.15. После наведения будет отображено название книги и описание, если оно присутствует.

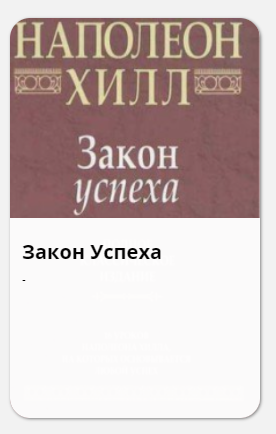


Рисунок 5.15 – Режим предпросмотра книги

Для получения книги и просмотра полной информации необходимо кликнуть на книгу, после клика будет открыто модальное окно, пример представлен на рисунке 5.16.

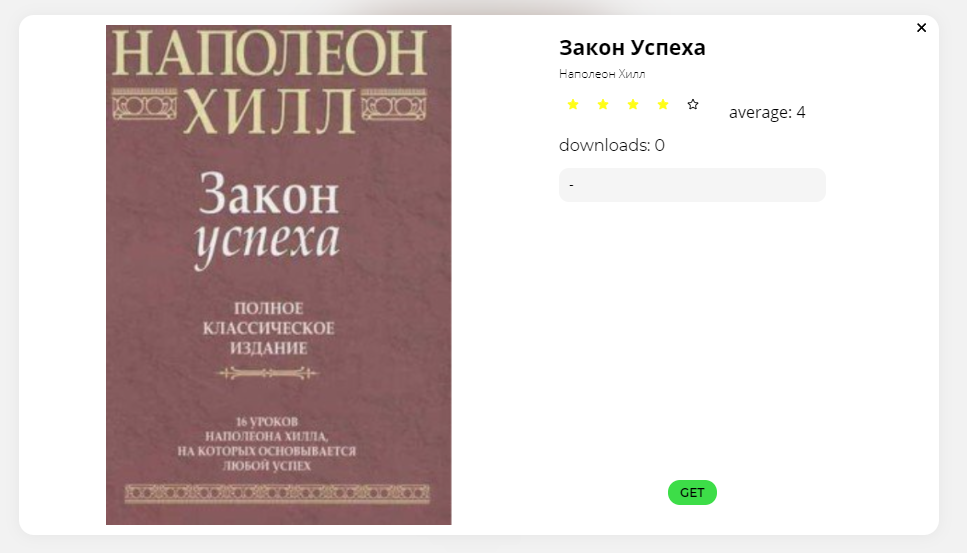


Рисунок 5.16 – Модальное окно просмотра книги в магазине

# 6 ТЕХНИКО-ЭКОНОМИЧЕСКОЕ ОБОСНОВАНИЕ ВЕБ-ПРИЛОЖЕНИЯ

## 6.1 Определение единовременных затрат на создание продукта

При расчете экономической эффективности разработки программного обеспечения необходимо сопоставить затраты на решение задачи при ручном методе ее решения с затратами, связанными с ее автоматизацией. В том случае, если разрабатываемая задача внедряется взамен уже функционирующей или она представляет собой модификацию существующей задачи, необходимо осуществить сравнение затрат на создание и функционирование старой и новой задачи.

Единовременные капитальные затраты представляют собой цену ПП (программного продукта) или модели. Различают оптовую и отпускную цены. Все расчеты между покупателем и продавцом продукции, к числу которой относят и программные продукты (модели), производятся на основе отпускных цен. В настоящее время в соответствии с законодательством Республики Беларусь в отпускную цену наряду с оптовой ценой включается налог на добавленную стоимость.

Определяющим фактором оптовой цены разработки является трудоёмкость создания ПП.

### 6.1.1 Определение трудоемкости разработки программного продукта

Для определения трудоёмкости разработки программного продукта необходимо определить степень новизны и сложность алгоритма.

Предусмотрено четыре степени новизны разрабатываемых задач.

1) A – разработка задач, предусматривающая применение принципиально новых методов разработки, проведение научно–исследовательских работ;

2) B – разработка типовых проектных решений, оригинальных задач и систем, не имеющих аналогов;

3) C – разработка проекта с использованием типовых проектных решений при условии их изменения; разработка проектов, имеющих аналогичные решения;

4) D – привязка типовых проектных решений.

Сложность алгоритма представлена тремя группами:

– алгоритмы оптимизации и моделирования систем и объектов;

– алгоритмы учета, отчетности, статистики и поиска;

– алгоритмы, реализующие стандартные методы решения, а также не предусматривающие применения сложных численных и логических методов.

При рассмотрении аналогов данного дипломного проекта стало понятно, что степенью новизны для данного проекта является В, а сложность алгоритма – 1.

Трудоемкость разработки ПП может быть определена на основе типовых норм времени для программирования задач на ЭВМ. Трудоемкость измеряется в чел.-часах, чел./днях. Она включает время на постановку задачи и время на программирование задачи и определяется по формуле:

|  |  |  |
| --- | --- | --- |
|  | Трз = Тоа + Тбс + Тп + Тотл Тдр + Тдо, | (6.1) |

где Тоа – трудоемкость подготовки описания задачи и исследования алгоритма решения;

Тбс – трудоемкость разработки блок–схемы алгоритма;

Тп – трудоемкость программирования по готовой блок–схеме;

Тотл – трудоемкость отладки программы на персональном компьютере;

Тдр – трудоемкость подготовки документации по задаче в рукописи;

Тдо – трудоемкость редактирования, печати и оформления документации по задаче.

Составляющие приведенной формулы определяются, в свою очередь, через условное число операторов *Q* в разрабатываемом ПП по формуле:

|  |  |  |
| --- | --- | --- |
|  | *Q* = *q* ∗ *C* ∗ (1 + *p*), | (6.2) |

где *q* – число операторов в программе, принимаем 1110;

*C* – коэффициент сложности программы (принят 1,26);

*p* – коэффициент коррекции программы (принят 0,3).

Коэффициент сложности программы *С* характеризует относительную сложность программ задачи по отношению к так называемой типовой задаче, сложность которой принята за единицу. Значение коэффициента определяется на базе экспертных оценок. Также выведена таблица для вычисления данного коэффициента.

Коэффициент коррекции программ *p* характеризует увеличение объема работ за счет внесения изменений в алгоритм и программу, изменения состава и структуры информации, а также уточнений, вносимых разработчиком программы для улучшения её качества без изменения постановки задачи. Значение *p* может быть принято равным 0,15 … 0,5 (принято 0,3).

*Q* = 1110 ∗ 1,26 ∗ (1 + 0,3) = 1818 (операторов).

Составляющие трудоемкости разработки программы определятся по формулам:

|  |  |  |
| --- | --- | --- |
|  | Тоа, | (6.3) |
|  |  |  |
|  | Тбс | (6.4) |
|  |  |  |
|  | Тп, | (6.5) |
|  |  |  |
|  | Тотл, | (6.6) |
|  | Тдр, | (6.7) |
|  |  |  |
|  | Тдо=0,75 ∗ Тдр, | (6.8) |

где *W* – коэффициент увеличения затрат труда вследствие недостаточного или некачественного описания задачи (*W* = 1,2...1,5);

*K* – коэффициент квалификации разработчика алгоритмов и программ (при стаже работы до двух лет *К* = 0,8, при стаже от двух до трех лет *К* = 1,0, при стаже от трех до пяти лет *К* = 1,1...1,2, при стаже от пяти до семи лет *К* = 1,3...1,4, при стаже свыше семи лет *К* = 1,6...1,6).

|  |  |  |
| --- | --- | --- |
|  | Тоа (чел.-час), |  |
|  | Тбс (чел.-час), |  |
|  | Тп(чел.-час), |  |
|  | Тотл(чел.-час), |  |
|  | Тдр (чел.-час), |  |
|  | Тдо = 0,75 ∗ 151,5 = 113,63 (чел.-час). |  |

Исходя из полученных данных можно найти трудоемкость разработки программного продукта, используя формулу (7.1):

Трз = + + + + + 113,63 = 958,25 (ч).

Трудоёмкость разработки программного продукта составляет 958,25 чел.-час.

### 6.1.2 Определение себестоимости создания программного продукта

Для определения себестоимости создания программного продукта необходимо определить затраты на заработную плату разработчика по формуле:

|  |  |  |
| --- | --- | --- |
|  | Зрз = Трз ∗ *t*чр ∗ (1 + *k*пр) ∗ (1 + *a*) ∗ (1 + *b*), | (6.9) |

где Трз – трудоемкость разработки программного продукта, чел–ч.;

*t*чр – среднечасовая ставка работника, осуществляющего разработку программного продукта;

*k*пр – коэффициент, учитывающий процент премий (0,4);

*а* – коэффициент, учитывающий дополнительную заработную плату (0,15);

*b* – коэффициент, учитывающий отчисления от фонда заработной платы (отчисления в фонд социальной защиты населения и отчисления на обязательное медицинское страхование от несчастных случаев) (*b* = 0,346).

Среднечасовая ставка работника определяется исходя из Единой тарифной системы оплаты труда в Республике Беларусь по следующей формуле:

|  |  |  |
| --- | --- | --- |
|  | *t*чр, | (6.10) |

где *TC*1р – среднемесячная заработная плата работника первого разряда (принято 41 руб.).

*k*т – тарифный коэффициент работника (принято 2,84).

170 – среднее нормативное количество рабочих часов в месяце для 2021 года.

|  |  |  |
| --- | --- | --- |
|  | *t*чр = 0,685 (руб.). |  |

Теперь можно определить затраты на заработную плату разработчиков программы:

Зрз = 958,25 ∗ 0,685 ∗ (1 + 0,4) ∗ (1 + 0,15) ∗ (1 + 0,346) = 1422,46 (руб.).

В себестоимость разработки ПП включаются также затраты на отладку ПП в процессе его создания. Для определения их величины необходимо рассчитать стоимость машино–часа работы персонального компьютера, на которой осуществлялась разработка и отладка. Данная величина соответствует величине арендной платы за один час работы персонального компьютера.

Затраты на отладку программы определяются по формуле:

|  |  |  |
| --- | --- | --- |
|  | Зот = Тотл ∗ Sмч, | (6.11) |

где Tотл – трудоемкость отладки программы ( час.).

*S*мч – стоимость машино–часа работы персонального компьютера (0,37 руб./час.).

|  |  |  |
| --- | --- | --- |
|  | Зот = 454,5∗ 0,37 = 168,165 (руб.). |  |

Себестоимость разработки ПП определяется по формуле:

|  |  |  |
| --- | --- | --- |
|  | Спр = Зрз ∗ *(1 + F)* + Зот, | (6.12) |

где *F* – коэффициент накладных расходов проектной организации без учета эксплуатации ЭВМ (принят 0,15).

Таким образом:

Спр = 1422,46 ∗ 1,15 + = 1803,99 (руб.).

### 6.1.3 Определение оптовой и отпускной цены программного продукта

Оптовая цена складывается из себестоимости создания программного продукта и плановой прибыли на программу.

Оптовая цена ПП определяется по формуле:

|  |  |  |
| --- | --- | --- |
|  | Цо =Спр + Пр, | (6.13) |

где Пр – плановая прибыль на программу, руб.

Плановая прибыль на программу определяется по формуле:

|  |  |  |
| --- | --- | --- |
|  | Пр =Спр ∗ Нп, | (6.14) |

где Спр – себестоимость программы;

Нп – норма прибыли проектной организации (принята Нп = 0,3).

Таким образом:

Пр =1803,99 ∗ 0,3 = 541,2 (руб.)

Тогда,

Цо =1803,99 + 541,2= 2345,19 (руб.)

Отпускная цена программы определяется по формуле:

|  |  |  |
| --- | --- | --- |
|  | Цпр =Цо +Цо ∗ Кндс, | (6.15) |

где Цо – оптовая цена программы, руб;

Кндс – ставка налога на добавленную стоимость (принято 20%).

Из этого следует, что

Цпр = 2345,19 +2345,19 ∗ 0,2 = 2814,22 (руб.).

### 6.1.4. Определение стоимости машино–часа работы персонального компьютера

Стоимость машино–часа работы ЭВМ определяется по формуле:

|  |  |  |
| --- | --- | --- |
|  | Sмч = Сэ + , | (6.16) |

где Сэ – расходы на электроэнергию за час работы персонального компьютера, руб;

Аэвм – годовая величина амортизационных отчислений на реновацию персонального компьютера;

Рэвм – годовые затраты на ремонт и техническое обслуживание персонального компьютера, руб;

Апл – годовая величина амортизационных отчислений на реновацию производственных площадей, занимаемых персонального компьютера, руб;

Рпл – годовые затраты на ремонт и содержание производственных площадей;

Рар – годовая величина арендных платежей за помещение, занимаемое персонального компьютера;

Фэвм – годовой фонд времени работы персонального компьютера, час.

Расходы на электроэнергию за час работы персонального компьютера определяются по формуле:

|  |  |  |
| --- | --- | --- |
|  | *С*э =*N*э ∗ *k*ис ∗ Цэ, | (6.17) |

где *N*э – установленная мощность электродвигателя ЭВМ (принято 0,6 кВт),

*k*ис – коэффициент использования энергоустановок по мощности (*k*ис = 0,9),

Цэ – стоимость 1 кВт–часа электроэнергии (принято 0,28666 руб. [6]),

Чэл – среднечасовое потребление электроэнергии перс. компьютера, кВт.

Сэ =0,6 ∗0,9 ∗ 0,28666 = 0,155 (руб.).

Годовая величина амортизационных отчислений на реновацию ЭВМ определяется по формуле:

|  |  |  |
| --- | --- | --- |
|  | Аэвм =Цэвм ∗*k*у ∗ *k*м ∗ = ∗ , | (6.18) |

где Цэвм – цена персонального компьютера на момент ее выпуска (850 руб.),

*k*у – коэффициент удорожания персонального компьютера (принят *k*у = 1),

*k*м– коэффициент, учитывающий затраты персонального компьютера (*k*м = 1,05);

– норма амортизационных отчислений на персонального компьютера, % ( = 10%);

– балансовая стоимость персонального компьютера, руб.

Из формулы 7.18 получим:

|  |  |  |
| --- | --- | --- |
|  | Аэвм =850 ∗ 1 ∗ 1,05 ∗ = 892,5 (руб.), |  |
|  | . |  |

Годовые затраты на ремонт и техническое обслуживание персонального компьютера укрупненно могут быть определены по формуле:

|  |  |  |
| --- | --- | --- |
|  | Рэвм = ∗ , | (6.19) |

где *k*ро – коэффициент, учитывающий затраты на ремонт и техническое обслуживание ЭВМ, в том числе затраты на запчасти, зарплату ремонтного персонала и др. (*k*ро = 0,13).

Тогда,

|  |  |  |
| --- | --- | --- |
|  | Рэвм = 892,5 ∗ . |  |

Годовая величина амортизационных отчислений на реновацию производственных площадей, занятых персонального компьютера определяется по формуле:

|  |  |  |
| --- | --- | --- |
|  | Апл = ∗ = *S*эвм ∗ *k*д ∗ Цпл ∗ , | (6.20) |

– норма амортизационных отчислений (= 1,2%);

*S*эвм – площадь, занимаемая персонального компьютера (1 кв.м.);

*k*д – коэффициент, учитывающий дополнительную площадь (*k*д = 3);

Цпл – цена квадратного метра производственной площади (400 руб.).

|  |  |  |
| --- | --- | --- |
|  | Апл = 1 ∗ 3 ∗ 400 ∗ = 14,4 (руб.). |  |

Балансовая стоимость площади:

|  |  |  |
| --- | --- | --- |
|  | = 1 ∗ 3 ∗ 400 = 1200 (руб.). |  |

Годовые затраты на ремонт и содержание производственных площадей укрупненно могут быть определены по формуле:

|  |  |  |
| --- | --- | --- |
|  | Рпл = ∗ , | (6.21) |

где *k*рэ – коэффициент, учитывающий затраты на ремонт и эксплуатацию производственных площадей (*k*рэ = 0,05).

|  |  |  |
| --- | --- | --- |
|  | Рпл =1200 ∗ . |  |

Годовая величина арендных платежей за помещение, занимаемое персонального компьютера, рассчитывается по формуле:

|  |  |  |
| --- | --- | --- |
|  | Рар = *S*эвм ∗ *k*д ∗ *k*ар ∗ *k*комф ∗ *k*пов ∗ 12, | (6.22) |

где *S*эвм – площадь, занимаемая персонального компьютера (1 кв.м);

*k*д – коэффициент, учитывающий дополнительную площадь (*k*д = 3);

*k*ар – ставка арендных платежей за помещение (базовая ставка арендной платы – 12 руб. за 1 кв. метр);

*k*df – коэффициент комфортности помещения (0,8);

*k*пов – повышающий коэффициент, учитывающий географическое размещение площади (0,81).

Тогда:

|  |  |  |
| --- | --- | --- |
|  | Рар = 1 ∗ 3 ∗ 12 ∗ 0,8 ∗ 0,81 ∗ 12 = 279,936 (руб.). |  |

Годовой фонд времени работы персонального компьютера определяется исходя из режима ее работы и может быть рассчитан по формуле:

|  |  |  |
| --- | --- | --- |
|  | Фэвм = *t*cc ∗ Tсг, | (6.23) |

где *t*сс – среднесуточная фактическая загрузка персонального компьютера (принято 8 час.);

Тсг – среднее количество дней работы персонального компьютера в год (принято 250 дней).

|  |  |  |
| --- | --- | --- |
|  | Фэвм = 8 ∗ 250 = 2000 (час). |  |

Таким образом, получим стоимость машино–часа работы персонального компьютера:

|  |  |  |
| --- | --- | --- |
|  | *S*мч = 0,1548 + = 0,435 (руб.). |  |

## 6.2 Определение ожидаемого прироста прибыли в результате внедрения программного продукта

Внедрение программного продукта может обеспечить пользователю ожидаемый прирост прибыли за счет сокращения трудоемкости решения задачи, являющейся предметом автоматизации и, как результат, снижения текущих затрат, связанных с решением данной задачи.

При условии, если внедряемый программный продукт полностью заменяет ручной труд, то производится сопоставление текущих затрат, связанных с решением задачи в ручном режиме пользователем и автоматизированном (при использовании данного проекта). В том случае, если разрабатываемая задача внедряется взамен уже функционирующей или она представляет собой модификацию существующей задачи, необходимо осуществить сравнение затрат на создание и функционирование старой и новой задачи.

### 6.2.1 Определение готовых эксплуатационных расходов при ручном решении задачи

Годовые эксплуатационные расходы при ручной обработке информации либо ручном решении задачи определяются по формуле:

|  |  |  |
| --- | --- | --- |
|  | Зр = Tр ∗ к ∗ tчр ∗ (1 + *q*) ∗ (1 + *a*) ∗ (1 + *b*), | (6.24) |

где Тр – трудоемкость разового решения задачи вручную;

к – периодичность решения задачи в течение года (400);

*t*чр – среднечасовая ставка работника, осуществляющего ручной расчет задачи (0,685);

*q* – коэффициент, учитывающий процент премий (принят 0,4);

*а* – коэффициент, учитывающий дополнительную заработную плату (0,15);

*b* – коэффициент, учитывающий отчисления от фонда заработной платы (0,346).

Трудоемкость разового ручного решения задачи определяется по формуле:

|  |  |  |
| --- | --- | --- |
|  | Тр = , | (6.25) |
|  |  |  |

Для начала выполнения задачи необходимо включить компьютер и подключиться к интернету: Tp1 = 0,2 ч. Затем, пройти регистрацию или авторизоваться на портале: Tp2 = 0,2 ч. Ввести необходимые данные, для заполнения договора: Tp3 = 1 ч. Получить и прочитать необходимую информацию: Tp4 = 0,6 ч.

Периодичность решения задачи в течение года зависит непосредственно от установленного количества дней приемной кампании, количества. Можно говорить о том, что периодичность решения задачи в год составляет порядка 400 раз.

|  |  |  |
| --- | --- | --- |
|  | Тр = 0,2 + 0,2 + 1 + 0,6 = 2 (чел. – ч). |  |
|  | Зр = 2 ∗ 400 ∗ 0,685 ∗ (1 + 0,4) ∗ (1 + 0,15) ∗ (1 + 0,346) = 1187,44 (руб.). |  |

### 6.2.2 Определение годовых затрат, связанных с эксплуатацией задачи

Для расчета годовых текущих затрат, связанных с эксплуатацией ПП, необходимо определить время решения данной задачи на персональном компьютере.

Время решения задачи на персональном компьютере определяется по формуле:

|  |  |  |  |
| --- | --- | --- | --- |
|  | Тз = (Твв + Тр + Твыв) ∗ , | (6.26) | (7.26) |

где Твв – время ввода в ЭВМ исходных данных, необходимых для решения задачи, мин (принято 15 мин.);

Тр – время вычислений, мин (принято 1 мин.);

Твыв – время вывода результатов решения задачи, мин; (принято 0,1 мин.);

*d*пз – коэффициент, учитывающий подготовительно–заключительное время (0,1).

|  |  |  |
| --- | --- | --- |
|  | Тз = (15+ 1+ 0,1) ∗ = 0,295 (чел. – ч). |  |

На основе рассчитанного времени может быть определена заработная плата пользователя. Затраты на заработную плату пользователя определяются по формуле:

|  |  |  |
| --- | --- | --- |
|  | Зп = Tз ∗ к ∗ *t*чр ∗ (1 + *q*) ∗ (1 + *a*) ∗ (1 + *b*), | (6.27) |

где Тз – время решения задачи на персональном компьютере, час;

*t*чп – среднечасовая ставка пользователя программы, руб. (определяется аналогично ставке работника, осуществляющего ручной расчет).

|  |  |  |
| --- | --- | --- |
|  | Зп = 0,295 ∗ 400 ∗ 0,685 ∗ (1 + 0,4) ∗ (1 + 0,15) ∗ (1 + 0,346) = 175,25 (руб.). |  |

Состав затрат, связанных с решением задачи, включаются также затраты, связанные с эксплуатацией персонального компьютера.

Затраты на оплату аренды персонального компьютера для решения задачи определяются по следующей формуле:

|  |  |  |
| --- | --- | --- |
|  | За = Tз ∗ *k* ∗ *S*мч, | (6.28) |

где *S*мч – стоимость одного машино–часа работы персонального компьютера.

Отсюда следует:

|  |  |  |
| --- | --- | --- |
|  | За = 0,295 ∗ 400 ∗ 0,435 = 51,33 (руб.). |  |

Годовые текущие затраты, связанные с эксплуатацией задачи, определяются по формуле:

|  |  |  |
| --- | --- | --- |
|  | Зт = Зп + За, | (6.29) |

где Зп – затраты на заработную плату пользователя программы;

За – затраты на оплату аренды персонального компьютера при решении задачи.

|  |  |  |
| --- | --- | --- |
|  | Зт = 175,25 + 51,33 = 226,58 (руб.). |  |

### 6.2.3 Определение ожидаемого прироста прибыли в результате внедрения программного продукта

Ожидаемый прирост прибыли в результате внедрения задачи взамен ручного ее расчета укрупненно может быть определен по формуле:

|  |  |  |
| --- | --- | --- |
|  | Пу = (Зр – Зт) ∗ (1 – Снп), | (6.30) |

где Снп – ставка налога на прибыль (18%).

|  |  |  |
| --- | --- | --- |
|  | Пу = (1187,44 – 226.58) ∗ (1 – 0,18) = 787,91 (руб.). |  |

## 6.3 Расчёт показателей эффективности использования программного продукта

Для определения годового экономического эффекта от разработанной программы необходимо определить суммарные капитальные затраты на разработку и внедрения программы по формуле:

|  |  |  |
| --- | --- | --- |
|  | Ко = Кз + Цпр, | (6.31) |

где Кз – капитальные и приравненные к ним затраты;

Цпр – отпускная цена программы.

|  |  |  |
| --- | --- | --- |
|  |  | (6.32) |

где – балансовая стоимость комплекта вычислительной техники, необходимого для решения задачи (принято за 892,5 руб.).

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  | . |  |

Годовой экономический эффект от сокращения ручного труда при обработке информации определяется по формуле:

|  |  |  |
| --- | --- | --- |
|  | ЭФ = Пу – Е ∗ Ко = Пу – Е ∗ (Кз + Цпр), | (6.33) |

где Е – коэффициент эффективности, равный ставке за кредиты на рынке долгосрочных кредитов (Е = 0,2).

ЭФ = 787,91 – 0,2 ∗ = 214,534 (руб.).

Срок возврата инвестиций определяется по формуле:

|  |  |  |
| --- | --- | --- |
|  | , | (6.34) |
|  | = 3,64 (г.). |  |

В таблице 6.1 приведены технико–экономические показатели проекта.

Таким образом, разрабатываемый в дипломном проекте «Web-приложение «Электронная книга»» раздел автоматизации ввода данных с отпускной ценой в 2814,22 руб. будет иметь ожидаемый прирост прибыли в 787,91 руб. Годовой экономический эффект составит 214,534 руб., а срок возврата инвестиции составит 3,64 года.

Таблица 6.1 – Технико–экономические показатели проекта

|  |  |  |
| --- | --- | --- |
| Наименование показателя | Базовый | Проектный |
| Трудоемкость решения задачи, час | 958,25 | 0,295 |
| Периодичность решения задачи, раз/год | 400 | 400 |
| Годовые текущие затраты, связанные с решением задачи, руб. | 1187,44 | 175,25 |
| Отпускная цена программы, руб |  | 2814,22 |
| Степень новизны программы |  | В |
| Группа сложности алгоритма |  | 1 |
| Прирост условной прибыли, руб |  | 787,91 |
| Годовой экономический эффект, руб. |  | 214,534 |
| Срок возврата инвестиций, лет |  | 3,64 |

# 7 ОХРАНА ТРУДА

## 7.1 Производственная санитария, техника безопасности и пожарная профилактика

При работе с ПЭВМ пользователи могут подвергаться воздействию опасных и вредных производственных факторов, таких как повышенные уровни: электромагнитного, ультрафиолетового и инфракрасного излучения; статического электричества; запыленность воздуха рабочей зоны; повышенный или пониженный уровень освещенности рабочей зоны, содержание в воздухе рабочей зоны вредных веществ (оксид углерода, озон, аммиак, фенол, формальдегид); напряжение зрения, памяти, внимания; длительное статическое напряжение; монотонность труда; нерациональная организация рабочего места; эмоциональные перегрузки.

Работа пользователей ПЭВМ проводится в соответствии с Санитарными нормами и правилами «Требования при работе с видеодисплейными терминалами и электронно-вычислительными машинами» и Гигиеническим нормативом «Предельно-допустимые уровни нормируемых параметров при работе с видеодисплейными терминалами и электронно-вычислительными машинами», утвержденными постановлением Министерства здравоохранения от 28.06.2013 г. № 59 и Типовой инструкцией по охране труда при работе с персональными ЭВМ, утвержденной постановлением Министерства труда и социальной защиты от 24.12.2013 № 130.

### 7.1.1 Метеоусловия

В производственных помещениях, в которых работа с использованием ПЭВМ является основной обеспечиваются оптимальные параметры микроклимата для категории работ 1а и 1б согласно нормативным документам, изображенные в таблице 7.1.

Таблица 7.1 – Оптимальные параметры микроклимата для помещений с ВДТ, ЭВМ и ПЭВМ

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Период года | Категория работ | Температура воздуха, оС, не более | Относительная влажность воздуха, % | Скорость движения воздуха, м/с |
| Холодный | легкая-1а | 22-24 | 40-60 | 0,1 |
| легкая-1б | 21-23 | 40-60 | 0,1 |
| Теплый | легкая-la | 23-25 | 40-60 | 0,1 |
| легкая-1б | 22-24 | 40-60 | 0,2 |

Работа с компьютером относится к категории 1а (работы, производимые сидя и сопровождающиеся незначительным физическим напряжением, при которых расход энергии составляет до 120 ккал/ч, т.е. до 139 Вт).

Интенсивность теплового излучения работающих от нагретых поверхностей технологического оборудования, осветительных приборов, инсоляции на постоянных рабочих местах не превышает значений, указанных в таблице 8.2 согласно Санитарных норм и правил.

Таблица 7.2 – Предельно допустимые уровни интенсивности излучения в инфракрасном и видимом диапазоне излучения на расстоянии 0,5 м со стороны экрана ВДТ, ЭВМ и ПЭВМ

|  |  |  |  |
| --- | --- | --- | --- |
| Диапазоны длин волн | 400–760 нм | 760–1050 нм | свыше 1050 нм |
| Предельно допустимые уровни | 0,1 Вт/м2 | 0,05 Вт/м2 | 4,0 Вт/м2 |

Для создания нормальных метеорологических условий наиболее целесообразно уменьшить тепловыделения от самого источника – монитора, что предусматривается при разработке его конструкции.

В производственных помещениях для обеспечения необходимых показателей микроклимата предусмотрены системы отопления, вентиляции и кондиционирования воздуха.

### 8.1.2 Вентиляция и отопление

Воздух рабочей зоны помещения соответствует санитарно–гигиеническим требованиям по содержанию вредных веществ и частиц пыли, приведенным в Санитарных нормах и правилах «Требованию к контролю воздуха рабочей зоны», Гигиеническом нормативе «Предельно допустимые концентрации вредных веществ в воздухе рабочей зоны», утв. пост. Министерства здравоохранения от 10.10.2017 г. № 92.

В помещениях, проводится ежедневная влажная уборка и систематическое проветривание после каждого часа работы.

Одним из мероприятий по оздоровлению воздушной среды является устройство вентиляции и отопления. Задачей вентиляции является обеспечение чистоты воздуха и параметров метеорологических условий на рабочих местах. Чистота воздушной среды достигается удалением загрязненного или нагретого воздуха из помещения и подачей в него свежего воздуха. Для поддержания нормального микроклимата необходим достаточный объем вентиляции, для чего в вычислительном центре предусматривается кондиционирование воздуха, осуществляющее поддержание постоянных параметров микроклимата в помещении независимо от наружных условий.

Параметры микроклимата поддерживаются в холодный период года за счет системы водяного отопления с нагревом воды до 100°С, а в теплый – за счет кондиционирования, с параметрами, отвечающими требованиям СНБ 4.02.01–03.

Уровни положительных и отрицательных аэроионов, а также коэффициент униполярности в воздухе всех помещений, где расположены ПЭВМ, соответствуют значениям, указанным в таблице 7.3.

Таблица 7.3 – Уровни ионизации и коэффициент униполярности воздуха помещений при работе с ВДТ, ЭВМ и ПЭВМ

|  |  |  |  |
| --- | --- | --- | --- |
| Уровни | Число ионов в 1 см3 воздуха | | Коэффициент униполярности (У) |
| n+ | n- |
| Минимально допустимые | 400 | 600 | 0,4 ≤ У < 1,0 |
| Оптимальные | 1500-3000 | 3000-5000 |
| Максимально допустимые | 50000 | 50000 |

### 7.1.3 Производственное освещение

В помещении для эксплуатации ПЭВМ предусмотрены естественное и искусственное освещение. Естественное освещение на рабочих местах осуществляется через световые проемы, ориентированные преимущественно на север, северо–восток, восток, запад или северо–запад и обеспечивает коэффициент естественной освещенности не ниже 1,5 %. Оконные проемы оборудованы регулируемыми устройствами типа жалюзи, занавесей.

Искусственное освещение в помещениях осуществляется системой общего равномерного освещения. При работе с документами применяется система комбинированного освещения, а освещенность поверхности стола в зоне размещения рабочего документа должна составлять 300–500 люкс. Освещенность поверхности экрана не более 300 люкс. В качестве источников света применяем люминесцентные лампы типа ЛБ. Коэффициент запаса для осветительных установок общего освещения принимается равным 1,4, а коэффициент пульсации – не более 5 %.

### 7.1.4 Шум

Основными источниками шума в помещениях, оборудованных ЭВМ, являются

принтеры, множительная техника и оборудование для кондиционирования воздуха, в самих ЭВМ – вентиляторы систем охлаждения и трансформаторы.

В таблице 7.4 приведены нормированные уровни шума согласно Санитарных норм и правил «Требования при работе с видеодисплейными терминалами и электронно-вычислительными машинами» и Гигиенических нормативов «Предельно-допустимые уровни нормируемых параметров при работе с видеодисплейными терминалами и электронно-вычислительными машинами», которые обеспечиваются за счет использования малошумного оборудования, применения звукопоглощающих материалов для облицовки помещений, а также различных звукопоглощающих устройств (перегородки и т. п.).

Таблица 7.4 – Предельно-допустимые уровни звука, эквивалентные уровни звука и уровни звукового давления в октавных полосах частот при работе с ВДТ, ЭВМ и ПЭВМ и периферийными устройствами

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Категория нормы  шума | Уровни звукового давления, дБ, в октавных полосах  со среднегеометрическими частотами, Гц | | | | | | | | | Уровни звука и  эквивалентные уровни звука, дБА |
| 31.5 | 63 | 125 | 250 | 500 | 1000 | 2000 | 4000 | 8000 |
| I | 86 | 71 | 61 | 54 | 49 | 45 | 42 | 40 | 38 | 50 |
| II | 93 | 79 | 70 | 63 | 58 | 55 | 52 | 50 | 49 | 60 |
| III | 96 | 83 | 74 | 68 | 63 | 60 | 57 | 55 | 54 | 65 |
| IV | 103 | 91 | 83 | 77 | 73 | 70 | 68 | 66 | 64 | 75 |

### 7.1.5 Электробезопасность

Помещение вычислительного центра по степени опасности поражения электрическим током относится к помещениям без повышенной опасности.

Основные меры защиты от поражения током:

– изоляция и недоступность токоведущих частей;

– защитное заземление (R3 = 4 Ом ГОСТ 12.1.030 - 81).

Первая помощь при поражениях электрическим током состоит из двух этапов: освобождение пострадавшего от действия тока и оказание ему доврачебной медицинской помощи. После освобождения пострадавшего от действия электрического тока необходимо оценить его состояние. Во всех случаях поражения электрическим током необходимо вызвать врача независимо от состояния пострадавшего.

### 7.1.6 Излучение

При работе с дисплеем могут возникать следующие опасные факторы: электромагнитные и электростатические поля, ультрафиолетовое и инфракрасное излучение.

Уровни физических факторов на рабочих местах пользователей, создаваемые ПЭВМ и периферийными устройствами, не превышают предельно-допустимые уровни: электромагнитных и электростатических полей, указанных в таблицах 7.5 и 7.6, ультрафиолетового, указанного в таблице 7.7, установленных Гигиеническим нормативом «Предельно допустимые уровни нормируемых параметров при работе с видеодисплейными терминалами и электронно-вычислительными машинами».

Таблица 7.5 – Предельно допустимые уровни электромагнитных полей от экранов ВДТ, ЭВМ и ПЭВМ

|  |  |  |
| --- | --- | --- |
| Наименование параметра | | Предельно-допустимые уровни |
| Напряженность электрического поля в диапазоне частот: | |  |
|  | 5 Гц-2 кГц | не более 25,0 В/м |
|  | 2-400 кГц | не более 2,5 В/м |
| Плотность магнитного потока магнитного поля в диапазоне частот: | |  |
|  | 5 Гц-2 кГц | не более 250 нТл |
|  | 2-400 кГц | не более 25 нТл |
| Напряженность электростатического поля | | не более 15 кВ/м |

Таблица 7.6 – Предельно допустимые уровни электромагнитных полей при работе с ВДТ, ЭВМ, ПЭВМ от клавиатуры, системного блока, манипулятора «мышь», беспроводных системам передачи информации и иных периферийных устройств

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Диапазоны частот | 0,3-300  кГц | 0,3-3  МГц | 3-30  МГц | 30-300  МГц | 0,3-300  ГГц |
| Предельно допустимые уровни | 25 В/м | 15 В/м | 10 В/м | 3 В/м | 10 мкВт/см2 |

Таблица 7.7 – Предельно допустимые уровни интенсивности излучения в ультрафиолетовом диапазоне на расстоянии 0,5 м со стороны экрана ВДТ, ЭВМ и ПЭВМ

|  |  |  |  |
| --- | --- | --- | --- |
| Диапазоны длин волн | 200-280 нм | 280-315 нм | 315-400 нм |
| Предельно допустимые уровни | не допускается | 0,0001 Вт/м2 | 0,1 Вт/м2 |

Наиболее эффективным и часто применяемым методом защиты от электромагнитных излучений является установка экранов. Экранируют либо источник излучения, либо рабочее место. Часто экран устанавливают непосредственно на монитор.

При работе монитора на экране кинескопа накапливается электростатический заряд, создающий электростатическое поле. При этом персонал, работающий с монитором, приобретают электростатический потенциал. Заметный вклад в общее электростатическое поле вносят электризующиеся от трения поверхности клавиатуры и мыши.

### 7.1.7 Пожарная безопасность

По взрывопожарной и пожарной опасности помещения и здания для ЭВМ относятся к категории Д согласно ТКП 474-2013. Здания для ВЦ и части зданий другого назначения, в которых предусмотрено размещение ЭВМ, относятся к 2 степени огнестойкости согласно ТКП 45-2.02-315-2018.

Для предотвращения распространения огня во время пожара с одной части здания на другую устраивают противопожарные преграды в виде стен, перегородок, дверей, окон. Особое требование предъявляется к устройству и размещению кабельных коммуникаций.

Нормы первичных средств пожаротушения для вычислительных центров приведены в таблице 7.9.

Таблица 7.9 – Примерные нормы первичных средств пожаротушения для вычислительного центра

|  |  |  |  |
| --- | --- | --- | --- |
| Помещение | Площадь, м2 | Углекислотные огнетушители ручные | Порошковые огнетушители |
| Вычислительный центр | 100 | 1 | 1 |

Для ликвидации пожаров в начальной стадии применяются первичные средства пожаротушения: внутренние пожарные водопроводы, огнетушители типа ОВП-10, ОУ-2, асбестовые одеяла и др.

Эвакуация персонала вычислительного центра осуществляется через эвакуационные выходы. Количество и общая ширина эвакуационных выходов определяются в зависимости от максимального возможного числа эвакуирующихся через них людей и предельно допустимого расстояния от наиболее удаленного места возможного пребывания людей до ближайшего эвакуационного выхода согласно ТКП 45-2.02-315-2018.

Расчетное время эвакуации устанавливается по реальному расчету времени движения одного или нескольких потоков людей через эвакуационные выходы из наиболее удаленных мест размещения людей. Необходимое время эвакуации устанавливается на основе данных о критической продолжительности пожара с учетом степени огнестойкости здания, категории производства по взрывной и пожарной опасности. Для успешной эвакуации необходимо, чтобы расчетное время было меньше необходимого.

## 7.2 Требования к ВДТ, ЭВМ, ПЭВМ и периферийным устройствам

Уровни физических факторов (уровни электромагнитных и электростатических полей, уровни вибрации, уровни ультрафиолетового, инфракрасного, видимого и мягкого рентгеновского излучений), создаваемые ПЭВМ и периферийными устройствами, не превышают предельно-допустимые уровни, установленных Гигиеническим нормативом «Предельно-допустимые уровни нормируемых параметров при работе с видеодисплейными терминалами и электронно-вычислительными машинами».

Уровни шума, создаваемые ПЭВМ и периферийными устройствами, не превышают предельно допустимых уровней, предусмотренных Гигиеническим нормативом, и устанавливаются в зависимости от следующих категорий производимых работ, изображенных в таблице 8.4:

категория I – выполнение основной работы в залах вычислительной техники, а также в помещениях с ПЭВМ всех типов учреждений образования;

категория II – выполнение работы с ПЭВМ в помещениях, где работают инженерно-технические работники, осуществляющие лабораторный, аналитический или измерительный контроль;

категория III – выполнение работы в помещениях операторов ЭВМ (без дисплеев);

категория IV – выполнение работы с ПЭВМ в помещениях для размещения шумных агрегатов.

В производственных помещениях, в которых работа с ПЭВМ является вспомогательной, уровни шума на рабочих местах не превышают значений, установленных для видов трудовой деятельности, осуществляемых в этих помещениях, в соответствии с Санитарными нормами и правилами, устанавливающими ПДУ шума на рабочих местах, в помещениях жилых и общественных зданий.

Допустимые визуальные эргономические параметры устройств отображения ВДТ, ЭВМ и ПЭВМ соответствуют допустимым значениям, установленным Гигиеническим нормативом.

Конструкция ЭВМ и ПЭВМ, дизайн и совокупность эргономических параметров обеспечивают надежное и комфортное считывание отображаемой информации в условиях эксплуатации. Конструкция оборудования ЭВМ и ПЭВМ обеспечивает возможность поворота корпуса в горизонтальной и вертикальной плоскости с фиксацией в заданном положении для обеспечения фронтального наблюдения экрана.

Дизайн ВДТ, ЭВМ, ПЭВМ и периферийных устройств должен предусматривать окраску корпуса в спокойные мягкие тона с диффузным рассеиванием света. Корпус ВДТ, ЭВМ, ПЭВМ и периферийных устройств имеет матовую поверхность с коэффициентом отражения 0,4-0,6 и не имеет блестящих деталей, способных создавать блики.

# ЗАКЛЮЧЕНИЕ

В рамках дипломного проекта было спроектировано веб-приложение для чтения книг онлайн. Для этого был произведен анализ существующих аналогов, построена логическая и физическая модель данных.

Для реализации приложения была выбрана современная библиотека React, т.к. она реализует современные паттерны программирования, имеет подробную и понятную документацию, а также мощную поддержку и большое сообщество разработчиков.

Приложение позволяет загружать книги и читать их с любого устройства.

Приложение является частью распределённой системы и выступает в роли вебклиента. Оно будет взаимодействовать с сервером через REST-интерфейс. Таким образом, внешний интерфейс приложения-сервера можно будет использовать и в других приложениях, например в приложениях для смартфонов на базе Android или iOS.

Было проведено критическое и углубленное тестирование разработанного приложения. Это позволило выявить некоторые ошибки, которые затем были исправлены.

Разработка данного приложения оправдана тем, что с его помощью можно значительно упростить процесс поиска и чтения книг. Кроме того, результаты расчета экономических показателей говорят об экономической выгоде от данной разработки.

# СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ

1 Бейтс, Марк CoffeeScript. Второе дыхание JavaScript / Марк Бейтс. - М.: ДМК Пресс, 2016. - 310 c.

2 Бенедетти, Р. Фрейн – HTML5 CSS3 – разработка сайтов для любых браузеров и устройств/ Р. Бенедетти. - М.: Питер, 2012. - 737 c.

3 Вальтер, Штефен Создание приложений для Windows 8 с использованием HTML5 и JavaScript / Штефен Вальтер. - М.: ДМК Пресс, 2013. - 344 c.

4 JS web frameworks benchmark. [Электронный ресурс]. – Режим доступа: https://habr.com/company/netologyru/blog/327294/, свободный. – Загл. с экрана. Яз. Англ. Дата доступа: 12.04.2021.

5 Frontend-developer [Электронный ресурс]. – Режим доступа: https://itvdn.com/ru/specialities/frontend-developer, свободный. – Загл. с экрана. Яз. Англ. Дата доступа: 12.01.2021.

6 Кантелон, М. Node.js в действии / М. Кантелон. - М.: Питер, 2015.

7 Козловский, Павел Разработка веб-приложений с использованием AngularJS / Павел Козловский , Питер Бэкон Дарвин, Павел Козловский. - Москва: СИНТЕГ, 2014. - 394 c.

8 Крокфорд, Д. JavaScript. Сильные стороны / Д. Крокфорд. - М.: Питер, 2013. - 117 c.

9 Макфарланд, Дэвид JavaScript и jQuery. Исчерпывающее руководство (+ DVD-ROM) / Дэвид Макфарланд. - М.: Эксмо, 2012. - 688 c.

10 Сухов, Кирилл Node.js. Путеводитель по технологии / Кирилл Сухов. - М.: ДМК Пресс, 2015. - 416 c.

11 Флэнаган, Дэвид JavaScript. Подробное руководство / Дэвид Флэнаган. - М.: Символ-плюс, 2013. - 714 c.

12 Фримен, Адам jQuery для профессионалов / Адам Фримен. - М.: Вильямс, 2013. - 960 c.

13 Хэррон, Дэвид Node.js Разработка серверных веб-приложений на JavaScript / Дэвид Хэррон. - М.: ДМК Пресс, 2012. – 987

14 React [Электронный ресурс]. – Режим доступа: http://reactjs.org/docs/react-api.html, свободный. – Загл. с экрана. Яз. Англ. Дата доступа: 12.04.2021.

15 Redux [Электронный ресурс]. – Режим доступа: https://react-redux.js.org/, свободный. – Загл. с экрана. Яз. Англ. Дата доступа: 12.04.2021.

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 81  Диаграмма вариантов использования | | | | | | | | | | | |
|  |  |  |  |  |  | | | | | |
|  |  |  |  |  |
| Изм. | Лист | № документа | Подпись | Дата |
| Разраб. | |  |  |  |  | Лит | | | Лист | Листов |
| Руковод. | |  |  |  |  | У |  | 1 | 10 |
| Консульт. | |  |  |  |  | | | | | |
| Н.контр. | |  |  |  |
|  | |  |  |  |
| 82  REST архитектура | | | | | | | | | | | |
|  |  |  |  |  |  | | | | | |
|  |  |  |  |  |
| Изм. | Лист | № документа | Подпись | Дата |
| Разраб. | |  |  |  |  | Лит | | | Лист | Листов |
| Руковод. | |  |  |  |  | У |  | 2 | 10 |
| Консульт. | |  |  |  |  | | | | | |
| Н.контр. | |  |  |  |
|  | |  |  |  |
| 83  Диаграмма таблиц базы данных | | | | | | | | | | | |
|  |  |  |  |  |  | | | | | |
|  |  |  |  |  |
| Изм. | Лист | № документа | Подпись | Дата |
| Разраб. | |  |  |  |  | Лит | | | Лист | Листов |
| Руковод. | |  |  |  |  | У |  | 3 | 10 |
| Консульт. | |  |  |  |  | | | | | |
| Н.контр. | |  |  |  |
|  | |  |  |  |
| 84  Архитектура React Redux приложения | | | | | | | | | | | |
|  |  |  |  |  |  | | | | | |
|  |  |  |  |  |
| Изм. | Лист | № документа | Подпись | Дата |
| Разраб. | |  |  |  |  | Лит | | | Лист | Листов |
| Руковод. | |  |  |  |  | У |  | 4 | 10 |
| Консульт. | |  |  |  |  | | | | | |
| Н.контр. | |  |  |  |
|  | |  |  |  |
| 85  Реализация компонента Libarary | | | | | | | | | | | |
|  |  |  |  |  |  | | | | | |
|  |  |  |  |  |
| Изм. | Лист | № документа | Подпись | Дата |
| Разраб. | |  |  |  |  | Лит | | | Лист | Листов |
| Руковод. | |  |  |  |  | У |  | 5 | 10 |
| Консульт. | |  |  |  |  | | | | | |
| Н.контр. | |  |  |  |
|  | |  |  |  |
| 86  Перечень компонентов приложения | | | | | | | | | | | |
|  |  |  |  |  |  | | | | | |
|  |  |  |  |  |
| Изм. | Лист | № документа | Подпись | Дата |
| Разраб. | |  |  |  |  | Лит | | | Лист | Листов |
| Руковод. | |  |  |  |  | У |  | 6 | 10 |
| Консульт. | |  |  |  |  | | | | | |
| Н.контр. | |  |  |  |
|  | |  |  |  |
| 87  Страница профиля | | | | | | | | | | | |
|  |  |  |  |  |  | | | | | |
|  |  |  |  |  |
| Изм. | Лист | № документа | Подпись | Дата |
| Разраб. | |  |  |  |  | Лит | | | Лист | Листов |
| Руковод. | |  |  |  |  | У |  | 7 | 10 |
| Консульт. | |  |  |  |  | | | | | |
| Н.контр. | |  |  |  |
|  | |  |  |  |
| 88  Страница «library» | | | | | | | | | | | |
|  |  |  |  |  |  | | | | | |
|  |  |  |  |  |
| Изм. | Лист | № документа | Подпись | Дата |
| Разраб. | |  |  |  |  | Лит | | | Лист | Листов |
| Руковод. | |  |  |  |  | У |  | 8 | 10 |
| Консульт. | |  |  |  |  | | | | | |
| Н.контр. | |  |  |  |
|  | |  |  |  |
| 88  Страница «shop» | | | | | | | | | | | |
|  |  |  |  |  |  | | | | | |
|  |  |  |  |  |
| Изм. | Лист | № документа | Подпись | Дата |
| Разраб. | |  |  |  |  | Лит | | | Лист | Листов |
| Руковод. | |  |  |  |  | У |  | 9 | 10 |
| Консульт. | |  |  |  |  | | | | | |
| Н.контр. | |  |  |  |
|  | |  |  |  |
| 88  Модальное окно просмотра книги в магазине | | | | | | | | | | | |
|  |  |  |  |  |  | | | | | |
|  |  |  |  |  |
| Изм. | Лист | № документа | Подпись | Дата |
| Разраб. | |  |  |  |  | Лит | | | Лист | Листов |
| Руковод. | |  |  |  |  | У |  | 10 | 10 |
| Консульт. | |  |  |  |  | | | | | |
| Н.контр. | |  |  |  |
|  | |  |  |  |

# ПРИЛОЖЕНИЕ A

ЛИСТИНГ ИСХОДНЫХ КОДОВ СЕРВЕРНОЙ ЧАСТИ

Код файла app.js

const path = require('path')

const express = require('express')

const bodyParser = require('body-parser')

const passport = require('passport')

const cors = require('cors')

require('dotenv').config()

require('./db/mongoose')

process.env.GOOGLE\_APPLICATION\_CREDENTIALS = './GOOGLE\_APPLICATION\_CREDENTIALS.json'

console.log('Setup env from: ', process.env.GOOGLE\_APPLICATION\_CREDENTIALS)

const PORT = process.env.PORT

const app = express()

app.use(bodyParser.json({ limit: '500mb' }))

app.use(bodyParser.urlencoded({ extended: true }))

app.use(cors({ origin: true, credentials: true }))

app.use(passport.initialize())

passport.serializeUser((user, cb) => cb(null, user))

app.use('/api', require('./routes/auth'))

app.use('/profile', require('./routes/profile'))

app.use('/library', require('./routes/library'))

app.use('/book', require('./routes/book'))

app.use('/shop', require('./routes/shop'))

const dir = path.join(\_\_dirname, '/db/files/static')

app.use('/files', express.static(dir))

app.listen(PORT, () => console.log(`Listen on port ${PORT}`))

Код файла auth.js

const express = require('express')

const jwt = require('jsonwebtoken')

const passport = require('passport')

const GoogleStrategy = require('passport-google-oauth').OAuth2Strategy

const User = require('../db/services/user')

const JWT\_KEY = 'some\_secret\_key'

const router = express()

const { GOOGLE\_CONSUMER\_KEY, GOOGLE\_CONSUMER\_SECRET } = process.env

passport.use(new GoogleStrategy({

clientID: GOOGLE\_CONSUMER\_KEY,

clientSecret: GOOGLE\_CONSUMER\_SECRET,

callbackURL: `${process.env.SERVER\_PATH}/api/auth/google/callback`

},

(token, refreshToken, profile, done) => User.findOrCreate(profile, (err, user) => done(err, user))

));

router.get('/auth/google', passport.authenticate('google', { scope: ['https://www.googleapis.com/auth/plus.login'] }))

router.get(

'/auth/google/callback',

passport.authenticate('google', { failureRedirect: '/login' }),

(req, res) => {

const token = jwt.sign({ id: req.user.oAuthId }, JWT\_KEY, { expiresIn: 60 \* 60 \* 24 \* 1000 })

// res.cookie('authorization', token, { maxAge: 900000, httpOnly: true })

res.redirect(`${process.env.CLIENT\_PATH}?token=${token}`)

},

);

module.exports = router

Код файла book.js

const express = require('express')

const jwt = require('jsonwebtoken')

const Book = require('../db/services/book')

const fdb = require('../utils/fileDBManager')

const JWT\_KEY = 'some\_secret\_key'

const router = express()

router.use((req, res, next) => {

const token = req.headers.authorization

jwt.verify(token, JWT\_KEY, (err, data) => {

if (err) {

res.status(401).send({ error: 'NotAuthorized' })

} else {

req.user = data

next()

}

})

})

router.get('/getBookSourceData/:bookId', async (req, res) => {

const { bookId } = req.params

const { hashSum, fileFormat } = await Book.findById(bookId)

const data = await fdb.read({ path: 'Books', hashSum, fileFormat })

res.send(data.toString('base64'))

})

router.get('/getBookImgData/:hashSum/:fileFormat', async (req, res) => {

const { hashSum, fileFormat } = req.params

const data = await fdb.read({ path: 'CoverImages', hashSum, fileFormat })

res.send(data.toString('base64'))

})

router.patch('/', async (req, res) => {

try {

const { bookId, key, value } = req.body.params

await Book.patchBookData({ id: bookId, data: { [key]: value } })

res.send(true)

} catch (e) {

console.log(e)

}

})

module.exports = router

Код файла library.js

const express = require('express')

const jwt = require('jsonwebtoken')

const Library = require('../db/services/library')

const Book = require('../db/services/book')

const fdb = require('../utils/fileDBManager')

const JWT\_KEY = 'some\_secret\_key'

const router = express()

router.use((req, res, next) => {

const token = req.headers.authorization

jwt.verify(token, JWT\_KEY, (err, data) => {

if (err) {

res.status(401).send({ error: 'NotAuthorized' })

} else {

req.user = data

next()

}

})

})

router.get('/getBooksList', async (req, res) => {

const library = await Library.findOrCreate({ id: req.user.id })

const { books: booksIds } = library

const booksList = await Promise.all(booksIds.map(id => Book.findById(id)))

res.send(booksList)

})

router.patch('/', async (req, res) => {

try {

const { id } = req.user

const library = await Library.findOrCreate({ id })

const book = await Book.create({ ...req.body.params, authorId: id })

const { hashSum, fileFormat } = book

const BufferData = Buffer.from(req.body.params.uploadBook.data, 'base64')

await fdb.write({ path: 'Books', hashSum, fileFormat, data: BufferData })

await Library.updateBooks({ id, data: library, newBooks: [book.bookId] }) // TODO: possible to optimise

res.send(200)

} catch (e) {

console.log(e)

}

})

router.patch('/addFromShopById', async (req, res) => {

try {

const { id } = req.user

const targetBook = await Book.findById(req.body.params.bookId)

const { bookId } = await Book.clone(targetBook.toObject())

await Library.addBooks({ id, newBooks: [bookId] })

res.send(true)

} catch (e) {

console.log(e)

}

})

router.delete('/:bookId', async (req, res) => {

const { bookId } = req.params

const book = await Book.findById(bookId)

if (book) {

await Book.removeBook(bookId)

await Library.removeBook({ id: req.user.id, bookId })

const availableToDeleteSource = !(await Book.findByHashSum(book.hashSum))

if (availableToDeleteSource) {

const { hashSum, fileFormat } = book

await fdb.deleteFile({ dirType: 'dynamic', hashSum, fileFormat })

}

} else {

console.warn('Can`t delete book')

}

})

module.exports = router

Код файла profile.js

const express = require('express')

const jwt = require('jsonwebtoken')

const User = require('../db/services/user')

const JWT\_KEY = 'some\_secret\_key'

const router = express()

router.use((req, res, next) => {

const token = req.headers.authorization

jwt.verify(token, JWT\_KEY, (err, data) => {

if (err) {

res.status(401).send({ error: 'NotAuthorized' })

} else {

req.user = data

next()

}

})

})

router.get('/', async (req, res) => {

const user = await User.findById(req.user.id)

res.send(user)

})

router.patch('/', async (req, res) => {

try {

const user = await User.updateById({ id: req.user.id, data: req.body.params })

res.send(user)

} catch (e) {

console.log(e)

}

})

module.exports = router

Код файла shop.js

const express = require('express')

const jwt = require('jsonwebtoken')

const Shop = require('../db/services/shop')

const fdb = require('../utils/fileDBManager')

const JWT\_KEY = 'some\_secret\_key'

const router = express()

router.use((req, res, next) => {

const token = req.headers.authorization

jwt.verify(token, JWT\_KEY, (err, data) => {

if (err) {

res.status(401).send({ error: 'NotAuthorized' })

} else {

req.user = data

next()

}

})

})

router.get('/getByAllCategories', async (req, res) => {

const booksByCategories = await Shop.findByAllCategories()

res.send(booksByCategories)

})

router.post('/', async (req, res) => {

try {

const book = await Shop.create(req.body.params)

const { imgHashSum: hashSum, imgFileFormat: fileFormat } = book

const BufferData = Buffer.from(req.body.params.coverImage.data, 'base64')

await fdb.write({ path: 'CoverImages', hashSum, fileFormat, data: BufferData })

res.send(true)

} catch (e) {

console.log(e)

}

})

router.patch('/', async (req, res) => {

try {

const { id: oAuthId } = req.user

const { bookId, key, value } = req.body.params

if (key === 'rating') {

const book = await Shop.findById(bookId)

const rating = new Map(book.rating)

rating.set(oAuthId.toString(), value)

await Shop.patchBookData({ bookId, data: { rating } })

}

res.send(true)

} catch (e) {

console.log(e)

}

})

module.exports = router

# ПРИЛОЖЕНИЕ Б

ЛИСТИНГ ИСХОДНЫХ КОДОВ КЛИЕНТСКОЙ ЧАСТИ

Код index.js

import React from 'react'

import { render } from 'react-dom'

import { Provider } from 'react-redux'

import { ConnectedRouter } from 'connected-react-router'

import store, { history } from './redux/store'

import App from './pages/App'

import 'sanitize.css/sanitize.css'

import styles from './styles/index.scss'

render(

<div className={styles.root}>

<Provider store={store}>

<ConnectedRouter history={history}>

<App />

</ConnectedRouter>

</Provider>

</div>,

document.querySelector('#root')

)

Код services/localStorageMannager.js

import \* as R from 'ramda'

const appStorageName = 'webBookData'

export const getStorageData = () => {

const data = localStorage.getItem(appStorageName)

return data ? JSON.parse(data) : null

}

export const setToStorage = (path, data) => {

localStorage.setItem(appStorageName, JSON.stringify(R.assocPath(path, data, getStorageData())))

}

export const pathToStorage = data => {

localStorage.setItem(appStorageName, JSON.stringify(data))

}

Код redux/store.js

import { createStore, applyMiddleware, compose } from 'redux'

import { routerMiddleware } from 'connected-react-router'

import thunk from 'redux-thunk'

import \* as History from 'history'

import { withStorage } from 'reduxApp/midlewares/withStorage'

import rootReducer from './modules/reducers'

export const history = History.createBrowserHistory()

const initialState = {}

const enhancers = []

const middleware = [thunk, routerMiddleware(history), withStorage]

if (process.env.NODE\_ENV === 'development') {

const devToolsExtension = window.\_\_REDUX\_DEVTOOLS\_EXTENSION\_\_

if (typeof devToolsExtension === 'function') {

enhancers.push(devToolsExtension())

}

}

const composedEnhancers = compose(

applyMiddleware(...middleware),

...enhancers

)

export default createStore(

rootReducer(history),

initialState,

composedEnhancers

)

Код redux/modules/actions/library.js

import \* as api from 'api'

import { push } from 'connected-react-router'

import { INDEX\_DB\_STORES } from 'constants/indexDBStores'

import { getIndexedDbItem, openIndexedDB, updateIndexedDBStore } from 'utils/indexDBApi'

import { packDataToBlob } from 'utils/storingMedia'

import { getBookById, getSelectedBook, hasBookDataById } from 'reduxApp/selectors/book'

import { getBooksList, getSelectedBookId } from 'reduxApp/selectors/library'

import {

deleteBook,

markPage,

patchBooks,

removeLinkedPage,

removeQuote,

setPage,

setPageQuantity,

setQuote,

updateBookData,

} from 'reduxApp/modules/reducers/books'

import { addBooksToLib, setSelectedBook } from 'reduxApp/modules/reducers/library'

import { closeModal } from 'reduxApp/modules/reducers/modal'

const actionLoadBooks = () => dispatch => new Promise(res => {

api.getBooks().then(booksList => {

const booksObj = booksList.reduce((acc, book) => {

acc[book.bookId] = book

return acc

}, {})

dispatch(patchBooks(booksObj))

dispatch(addBooksToLib(booksObj))

res()

})

})

export const userActionLoadBooks = (shouldAlwaysLoad = false) => (dispatch, getState) => {

const state = getState()

const booksList = getBooksList(state)

if (shouldAlwaysLoad || !booksList.length) {

return dispatch(actionLoadBooks())

}

return Promise.resolve()

}

export const userActionLoadBookDataIfNoData = bookId => async (dispatch, getState) => {

const state = getState()

const hasContentAtRedux = hasBookDataById(state, { bookId })

if (!hasContentAtRedux) {

// TODO: mb add expiration date

const openDB = await openIndexedDB(INDEX\_DB\_STORES.BOOK\_CONTENT\_DATA)

const bookContentFromIndexDB = await getIndexedDbItem(openDB, INDEX\_DB\_STORES.BOOK\_CONTENT\_DATA, bookId)

let bookData

if (bookContentFromIndexDB) {

bookData = bookContentFromIndexDB.value

} else {

const bookDataBase64 = await api.getBookSourceData(bookId)

const { fileFormat } = getBookById(state, { bookId })

bookData = `data:application/${fileFormat};base64,${bookDataBase64}`

updateIndexedDBStore(openDB, INDEX\_DB\_STORES.BOOK\_CONTENT\_DATA, bookId, bookData) // no await because we do not need to wait it

}

const bookDataBlobLink = await packDataToBlob(bookData)

dispatch(updateBookData({ bookId, data: bookDataBlobLink }))

}

}

export const actionUpdateBookAtServer = ({ bookId, key }) => (\_, getState) => {

const book = getBookById(getState(), { bookId })

api.updateBook({ bookId, key, value: book[key] })

}

export const userActionPublishBook = formData => () => {

api.addBookToShop(formData)

}

export const userActionUploadBook = formData => dispatch => api.addBook(formData).then(\_res => {

dispatch(closeModal())

dispatch(actionLoadBooks())

})

export const userActionSelectBook = bookId => (dispatch, \_getState) => {

dispatch(setSelectedBook({ id: bookId }))

dispatch(userActionLoadBookDataIfNoData(bookId))

dispatch(push('/book'))

}

export const userActionAddQuote = quote => (dispatch, getState) => {

const bookId = getSelectedBookId(getState())

dispatch(setQuote({ bookId, quote }))

dispatch(actionUpdateBookAtServer({ bookId, key: 'quotes' }))

}

export const actionSetPage = pageNumber => (dispatch, getState) => {

const bookId = getSelectedBookId(getState())

dispatch(setPage({ bookId, page: pageNumber }))

dispatch(actionUpdateBookAtServer({ bookId, key: 'pageNumber' }))

}

export const userActionNextPage = () => (dispatch, getState) => {

const selectedBook = getSelectedBook(getState())

if (selectedBook) {

const { pageNumber, pageQuantity } = selectedBook

if (pageNumber < pageQuantity) {

dispatch(actionSetPage(pageNumber + 1))

}

}

}

export const userActionPrevPage = () => (dispatch, getState) => {

const selectedBook = getSelectedBook(getState())

if (selectedBook) {

const { pageNumber } = selectedBook

if (pageNumber > 1) {

dispatch(actionSetPage(pageNumber - 1))

}

}

}

export const userActionMarkPage = () => (dispatch, getState) => {

const selectedBook = getSelectedBook(getState())

if (selectedBook) {

const { bookId, pageNumber, pages } = selectedBook

if (!pages.includes(pageNumber)) {

dispatch(markPage({ bookId, page: pageNumber }))

dispatch(actionUpdateBookAtServer({ bookId, key: 'pages' }))

}

}

}

export const userActionSetPageQuantity = pageQuantity => (dispatch, getState) => {

const bookId = getSelectedBookId(getState())

dispatch(setPageQuantity({ bookId, pageQuantity }))

dispatch(actionUpdateBookAtServer({ bookId, key: 'pageQuantity' }))

}

export const userActionDeselectLinkedPage = ({ bookId, page }) => (dispatch, \_getState) => {

dispatch(removeLinkedPage({ bookId, page }))

dispatch(actionUpdateBookAtServer({ bookId, key: 'pages' }))

}

export const userActionDeleteQuote = ({ bookId, quote }) => dispatch => {

dispatch(removeQuote({ bookId, quote }))

dispatch(actionUpdateBookAtServer({ bookId, key: 'quotes' }))

}

export const userActionSelectBookAndSetPage = ({ bookId, pageNumber }) => (dispatch, \_getState) => {

dispatch(userActionSelectBook(bookId))

dispatch(actionSetPage(pageNumber))

}

export const userActionModalDeleteBook = bookId => dispatch => {

api.deleteBook(bookId)

dispatch(closeModal())

dispatch(deleteBook({ bookId }))

}

Код redux/modules/actions/shop.js

import \* as R from 'ramda'

import \* as api from 'api'

import { updateBooksAtShop } from 'reduxApp/modules/reducers/shop'

import { patchBooks, updateBookCoverImg, updateRating } from 'reduxApp/modules/reducers/books'

import { getBookById } from 'reduxApp/selectors/book'

import { getOAuthId } from 'reduxApp/selectors/profile'

import { userActionLoadBooks } from 'reduxApp/modules/actions/library'

import { closeModal } from 'reduxApp/modules/reducers/modal'

import { push } from 'connected-react-router'

import { packDataToBlob } from 'utils/storingMedia'

import { getBooksShopItemCoverImage } from 'api'

export const userActionShopLoadBooksByAllCategories = () => dispatch => {

api.getBooksShopByAllCategories().then(result => {

const shopData = R.map(R.map(R.prop('bookId')), result)

const booksData = Object.values(result).reduce((acc, books) => {

books.forEach(book => {

acc[book.bookId] = book

})

return acc

}, {})

dispatch(updateBooksAtShop(shopData))

dispatch(patchBooks(booksData))

})

}

export const userActionSetRatingById = ({ bookId, ratingMark }) => (dispatch, getState) => {

api.updateShopBook({ bookId, key: 'rating', value: ratingMark })

const oAuthId = getOAuthId(getState())

dispatch(updateRating({ oAuthId, bookId, ratingMark }))

}

export const userActionGetBookFromShop = bookId => async dispatch => {

await api.addBookToLibraryFromShop({ bookId })

dispatch(closeModal())

dispatch(push('/library'))

dispatch(userActionLoadBooks(true))

}

export const userActionUploadCoverImageIfNo = ({ bookId }) => (dispatch, getState) => {

const { bookCoverImgPath, imgHashSum, imgFileFormat, imgDataInfo } = getBookById(getState(), { bookId })

if (!bookCoverImgPath) {

getBooksShopItemCoverImage(imgHashSum, imgFileFormat).then(data => {

const Base64Data = `${imgDataInfo},${data}`

packDataToBlob(Base64Data).then(blobLink => dispatch(updateBookCoverImg({ bookId, data: blobLink })))

})

}

}

Код redux/modules/reducers/books.js

import \* as R from 'ramda'

import { createAction } from 'redux-actions'

import { getStorageData } from 'services/localStorageMannager'

export const ADD\_BOOK = 'ADD\_BOOK'

export const PATCH\_BOOKS = 'PATCH\_BOOKS'

export const UPDATE\_BOOKS = 'UPDATE\_BOOKS'

export const DELETE\_BOOK = 'DELETE\_BOOK'

export const MARK\_PAGE = 'MARK\_PAGE'

export const DOWNGRADE\_BOOK\_SCALE = 'DOWNGRADE\_BOOK\_SCALE'

export const RAISE\_BOOK\_SCALE = 'RAISE\_BOOK\_SCALE'

export const SET\_PAGE = 'SET\_PAGE'

export const REMOVE\_LINKED\_PAGE = 'REMOVE\_LINKED\_PAGE'

export const REMOVE\_QUOTE = 'REMOVE\_QUOTE'

export const SET\_PAGE\_QUANTITY = 'SET\_PAGE\_QUANTITY'

export const ADD\_QUOTE = 'ADD\_QUOTE'

export const UPDATE\_RATING = 'UPDATE\_RATING'

export const UPDATE\_BOOK\_COVER\_IMG = 'UPDATE\_BOOK\_COVER\_IMG'

export const UPDATE\_BOOK\_DATA = 'UPDATE\_BOOK\_DATA'

const bookReducer = (state, { payload, type }) => {

switch (type) {

case ADD\_BOOK:

return payload

case DELETE\_BOOK:

return null

case DOWNGRADE\_BOOK\_SCALE: {

const currentBookScale = state.bookScale || 1

const newScale = currentBookScale - 0.25

if (newScale >= 0.75) {

return { ...state, bookScale: newScale }

}

return state

}

case RAISE\_BOOK\_SCALE: {

const currentBookScale = state.bookScale || 1

const newScale = currentBookScale + 0.25

if (newScale <= 3) {

return { ...state, bookScale: newScale }

}

return state

}

case SET\_PAGE:

return { ...state, pageNumber: payload.page }

case SET\_PAGE\_QUANTITY:

return { ...state, pageQuantity: payload.pageQuantity }

case ADD\_QUOTE:

return { ...state, quotes: state.quotes ? state.quotes.concat(payload.quote) : [payload.quote] }

case MARK\_PAGE:

return { ...state, pages: state.pages ? state.pages.concat(payload.page) : [payload.page] }

case REMOVE\_LINKED\_PAGE:

return { ...state, pages: state.pages.filter(page => page !== payload.page) }

case REMOVE\_QUOTE:

return { ...state, quotes: state.quotes.filter(quote => quote !== payload.quote) }

case UPDATE\_RATING:

return R.assocPath(['rating', payload.oAuthId], payload.ratingMark, state)

case UPDATE\_BOOK\_COVER\_IMG:

return R.assoc('bookCoverImgPath', payload.data, state)

case UPDATE\_BOOK\_DATA:

return R.assoc('bookDataPath', payload.data, state)

default:

return state

}

}

export default (state = R.prop('books', getStorageData()) || {}, { payload, type }) => {

if (type === PATCH\_BOOKS) {

return R.mergeDeepRight(state, payload)

}

if (type === UPDATE\_BOOKS) {

return { ...state, ...payload }

}

if ([

ADD\_BOOK,

DELETE\_BOOK,

MARK\_PAGE,

RAISE\_BOOK\_SCALE,

DOWNGRADE\_BOOK\_SCALE,

SET\_PAGE,

REMOVE\_LINKED\_PAGE,

REMOVE\_LINKED\_PAGE,

REMOVE\_QUOTE,

SET\_PAGE\_QUANTITY,

ADD\_QUOTE,

UPDATE\_RATING,

UPDATE\_BOOK\_COVER\_IMG,

UPDATE\_BOOK\_DATA,

].includes(type)) {

return ({

...state,

[payload.bookId]: bookReducer(state[payload.bookId], { payload, type }),

})

}

return state

}

export const addBook = createAction(ADD\_BOOK)

export const patchBooks = createAction(PATCH\_BOOKS)

export const updateBooks = createAction(UPDATE\_BOOKS)

export const deleteBook = createAction(DELETE\_BOOK)

export const markPage = createAction(MARK\_PAGE)

export const raiseBookScale = createAction(RAISE\_BOOK\_SCALE)

export const downgradeBookScale = createAction(DOWNGRADE\_BOOK\_SCALE)

export const setPage = createAction(SET\_PAGE)

export const removeLinkedPage = createAction(REMOVE\_LINKED\_PAGE)

export const removeQuote = createAction(REMOVE\_QUOTE)

export const setPageQuantity = createAction(SET\_PAGE\_QUANTITY)

export const setQuote = createAction(ADD\_QUOTE)

export const updateRating = createAction(UPDATE\_RATING)

export const updateBookCoverImg = createAction(UPDATE\_BOOK\_COVER\_IMG)

export const updateBookData = createAction(UPDATE\_BOOK\_DATA)