ОГЛАВЛЕНИЕ

[ПЕРЕЧЕНЬ УСЛОВНЫХ ОБОЗНАЧЕНИЙ И ТЕРМИНОВ 4](#_Toc72613912)

[ВВЕДЕНИЕ 5](#_Toc72613913)

[1 ОБЗОР СОСТОЯНИЯ ВОПРОСА 7](#_Toc72613914)

[1.1 Обзор существующих аналогов 7](#_Toc72613915)

[1.2 React 8](#_Toc72613916)

[1.3 Webpack 10](#_Toc72613917)

[1.4 Express 11](#_Toc72613918)

[1.5 MongoDB 11](#_Toc72613919)

[1.6 Краткий обзор используемых источников 13](#_Toc72613920)

[2 ЛОГИЧЕСКОЕ МОДЕЛИРОВАНИЕ 15](#_Toc72613921)

[2.1 Модель вариантов использования 15](#_Toc72613922)

[2.1.1 Список действующих лиц 15](#_Toc72613923)

[2.1.2 Список вариантов использования 15](#_Toc72613924)

[2.1.2 Диаграммы вариантов использования 24](#_Toc72613925)

[3 ФИЗИЧЕСКОЕ МОДЕЛИРОВАНИЕ 25](#_Toc72613926)

[3.1 Взаимодействие компонентов распределенного приложения 25](#_Toc72613927)

[3.2 Модель данных 26](#_Toc72613928)

[4 РЕАЛИЗАЦИЯ И ТЕСТИРОВАНИЕ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ 29](#_Toc72613929)

[4.1 Архитектура клиентской части проекта 29](#_Toc72613930)

[4.1.1 Компоненты 31](#_Toc72613931)

[4.2 Архитектура серверной части проекта 34](#_Toc72613932)

ВЕДОМОСТЬ ОБЪЕМА ДИПЛОМНОГО ПРОЕКТА

ОГЛАВЛЕНИЕ

ПЕРЕЧЕНЬ УСЛОВНЫХ ОБОЗНАЧЕНИЙ И ТЕРМИНОВ

ВВЕДЕНИЕ

ОБЗОР СОСТОЯНИЯ ВОПРОСА

Обзор существующих аналогов

1.1.1 React Js

1.1.2 Webpack

Express

MongoDb

1.5 Краткий обзор используемых источников

ЛОГИЧЕСКОЕ МОДЕЛИРОВАНИЕ

2.1 Модель вариантов использования

2.1.1 Список действующих лиц

2.1.2 Список вариантов использования

2.1.3 Диаграммы вариантов использования

3 ФИЗИЧЕСКОЕ МОДЕЛИРОВАНИЕ

3.1 Взаимодействие компонентов распределенного приложения

3.2 Модель данных

4 РЕАЛИЗАЦИЯ И ТЕСТИРОВАНИЕ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

4.1 Архитектура проекта

4.2 Модули

4.3 Компоненты

4.4 Шаблоны

4.5 Сервисы

4.6 Тестирование программного обеспечения

4.6.1 Используемые аппаратные средства

4.6.2 Критическое тестирование

4.6.3 Углубленное тестирование

4.6.4 Результаты тестирования

РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

6 ТЕХНИКО-ЭКОНОМИЧЕСКОЕ ОБОСНОВАНИЕ

ВЕБ-ПРИЛОЖЕНИЯ

6.1 Определение единовременных затрат на создание программного продукта

6.1.1 Определение трудоемкости разработки программного продукта

6.1.2 Определение себестоимости создания программного продукта

6.1.3 Определение оптовой и отпускной цены программного продукта

6.1.4 Определение стоимости машино-часа работы ЭВМ

6.1.5 Определение ожидаемого прироста прибыли в результате внедрения

программного продукта

6.2.1 Определение годовых эксплуатационных расходов при ручном решении

поставленной задачи

6.2.2 Определение годовых затрат, связанных с эксплуатацией задачи

6.2.3 Определение ожидаемого прироста прибыли в результате внедрения

программного продукта

6.3 Расчёт показателей эффективности использования программного продукта

ОХРАНА ТРУДА

7.1 Производственная санитария, техника безопасности и пожарная профилактика

7.1.1 Метеоусловия

7.1.2 Вентиляция и отопление

7.1.3 Освещение

7.1.4 Шум

7.1.5 Электробезопасность

7.1.6 Излучение

7.1.7 Пожарная безопасность

7.2 Организация рабочего места пользователя ПЭВМ

ЗАКЛЮЧЕНИЕ

СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ

ГРАФИЧЕСКАЯ ЧАСТЬ

ПРИЛОЖЕНИЕ

# 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. 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.

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 такие как: Instagram, Vkonakte, TikTok, YouTube. 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.

«WebBook» – 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 «Web Book» – 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 сервер.
* 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.

## 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.

## 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. Связка 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. Создание 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.

## 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 т.д.

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. На сегодняшний момент Вебпак – один из самых востребованных инструментов JS-разработки.

## 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.

## 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.

## Краткий обзор используемых источников

Главным источником является официальная документация фреймворка React на английском языке [5]. Здесь содержится вся необходимая для разработчика информация: инструкция по установке и запуску, описание доступных интерфейсов, примеры использования различных интерфейсов и компонентов (рисунок 1.1).



Рисунок 1.1 – Веб-страница официальной документации React

Главным источником информации для библиотеки Redux является официальная документация на английском языке [5]. 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 [5]. 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

На этапе логического моделирования необходимо определить роли пользователей системы и основные сценарии её использования.

## 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.

В таблице 2.1 перечислены варианты использования для роли «Неавторизованный пользователь».

Таблица 2.1 – Варианты использования для роли «Неавторизованный пользователь»

|  |  |
| --- | --- |
| **UC1** | |
| Название | Авторизация |
| Роль | Неваторизованный пользователь |
| Цель | Авторизоваться в сиситеме |
| Результат | Пользователь авторизован |
| Триггер | Пользователь зашел в систему и нажал на кнопку "Click here to login with google" |
| Главный поток | 1. Пользователь входит в систему через систему Google Auth. 2. Выбор пользователся для авторизации. 3. Логин и пароль верны, пользователь авторизован. |
| Альтернативный поток | 3.1 Логин или пароль неверны, пользователь не авторизован. 3.2 Появляется сообщение о том, что пароль или логин. неправильные. Пользователь снова вводит логин и пароль. |
| **UC2** | |
| Название | Выход из системы |
| Роль | Авторизованный пользователь |
| Цель | Выйти из системы, попасть на страницу "about" |
| Результат | Пользователь переадресован на страницу "about", доступ к другим страницам закрыт |
| Триггер | Пользователь зашел в систему и нажал на кнопку "Logout" |
| Главный поток | 1. Пользователь входит на страницу профиля. 2. Нажатие на кнопку "Logout". |
| Альтернативный поток |  |
| **UC3** | |
| Название | Редактиование профиля |
| Роль | Авторизованный пользователь |
| Цель | Обновить данные профиля |
| Результат | Данные профиля обновлены |

Продолжение таблицы 2.1

|  |  |
| --- | --- |
| Триггер | Нажатие на кнопку "edit profile", на странице профиля "im" |
| Главный поток | 1. Нажатие на кнопку "edit profile". 2. Открывается модальное окно редактиования профиля. 3. Заполнение полей name, female, nickName новыми данными, а так же загрузку фото профиля. 4. Нажатие на кнопку "apply". 5. Данные пользоывателя обновлены, модальное окно закрыто. |
| Альтернативный поток | 4.1 Пользователь нажимает на кнопку закрытия. 5.1 Данные профиля не обновляются. |
| **UC4** | |
| Название | Загрузка новой книги |
| Роль | Авторизованный пользователь |
| Цель | Загрузить книгу в библиотеку пользователя |
| Результат | Книга успешно загружена, и доступна к использованию |
| Триггер | Нажатие на кнопку "add book", на странице профиля "library" |
| Главный поток | 1. Нажатие на кнопку "add book". 2. Открывается модальное окно добавления книги. 3. Заполнение полей Book Title, Author, qoute новыми данными, а так же загрузка книги в формате PDF. 4. Нажатие на кнопку "apply". 5. Данные пользоывателя обновлены, модальное окно закрыто. 6. Книга загружается на страницу библиотеки "library". |
| Альтернативный поток | 4.1 Пользователь нажимает на кнопку закрытия 5.1 Модальное окно закрывается, книга не загружается |
| **UC5** | |
| Название | Поиск книги в библиотеке |
| Роль | Авторизованный пользователь |
| Цель | Совершить поиск книги по названию |
| Результат | Книга найдена |
| Триггер | Ввод названия книги в поле поиска книги на странице "library" |
| Главный поток | 1. Ввод названия книги в поле поиска книги на странице библиотеке. 2. Индикатор отображает колличество найденных книг. 3. книга найдена, книги не подходящие по названию не показываются. |

Продолжение таблицы 2.1

|  |  |
| --- | --- |
| **UC6** | |
| Название | Поиск по всем книгам в библиотеке |
| Роль | Авторизованный пользователь |
| Цель | Совершить поиск по всем книгам в библиотеке |
| Результат | Список всех книг отображен |
| Триггер | Ввод названия книги в поле поиска книги на странице библиотеке |
| Главный поток | 1. Ввод названия книги в поле поиска книги на странице "library". 2. Индикатор отображает колличество найденных книг. 3. все достопные книги отображаются. |
| Альтернативный поток |  |
| **UC7** | |
| Название | Удаление книги из библиотеки |
| Роль | Авторизованный пользователь |
| Цель | Удаление книги из библиотеки |
| Результат | Книга удалена из библиотеки |
| Триггер | Нажатие на кнопку "delete" у выбранной книги |
| Главный поток | 1. Нажатие на кнопку "delete" у выбранной книги находясь на странице "library". 2. Книга удалена, список книг в библиотеке обновлен. |
| Альтернативный поток |  |
| **UC8** | |
| Название | Просмотр цитата по выбранной книге |
| Роль | Авторизованный пользователь |
| Цель | Открыть список цитат |
| Результат | Список цитат открыт |
| Триггер | Нажатие на кнопку "citation" |
| Главный поток | 1. Нажатие на кнопку "citation" у выбранной книги находясь на странице "library". 2. Модальное окно со списком цитат открыто. |
| Альтернативный поток |  |
| **UC9** | |
| Название | Удаление цитаты по выбранной книге |
| Роль | Авторизованный пользователь |
| Цель | Удалить цитату |
| Результат | Выбранная цитата удалена, список цитат обновлен |

Продолжение таблицы 2.1

|  |  |
| --- | --- |
| Результат | Выбранная цитата удалена, список цитат обновлен |
| Триггер | Нажатие на иконку удаления у выбранной цитаты |
| Главный поток | 1. Нажатие на иконку удаления у выбранной цитаты. 2. Цитата удалена, список цитат обновлен. |
| Альтернативный поток |  |
| **UC10** | |
| Название | Просмотр отмеченных страниц по выбранной книге |
| Роль | Авторизованный пользователь |
| Цель | Открыть список отмеченных страниц |
| Результат | Список отмеченных страниц открыт |
| Триггер | Нажатие на кнопку "pages" |
| Главный поток | 1. Нажатие на кнопку "pages" у выбранной книги находясь на странице "library". 2. Модальное окно со списком отмеченных страниц открыто. |
| Альтернативный поток |  |
| **UC11** | |
| Название | Удаление отмеченной страницы по выбранной книге |
| Роль | Авторизованный пользователь |
| Цель | Удалить цитату |
| Результат | Выбранная страница удалена, список отмеченных страниц обновлен |
| Триггер | Нажатие на иконку удаления у выбранной цитаты |
| Главный поток | 1. Нажатие на иконку удаления у выбранной отмеченной страницы. 2. Отмеченная страница удалена, список цитат обновлен. |
| Альтернативный поток |  |
| **UC12** | |
| Название | Открытие отмеченной страницы по выбранной книге |
| Роль | Авторизованный пользователь |
| Цель | Удалить цитату |
| Результат | Открыта страница "book" на выбранной странице |
| Триггер | Нажатие на отмеченную страницу |
| Главный поток | 1. Нажатие на отмеченную страницу. 2. Отмеченная страница открыта. |

Продолжение таблицы 2.1

|  |  |
| --- | --- |
| **UC13** | |
| Название | Открытие выбранной книги |
| Роль | Авторизованный пользователь |
| Цель | Открыть страницу на последней прочитанной странице |
| Результат | Открытие страницы на последней прочитанной странице |
| Триггер | клик на изображение книги на странице "library" |
| Главный поток | 1. Клик на изображение книги на странице "library". 2. Переход на страницу "book", открывается выбранная книга на последней прочитанной странице. |
| Альтернативный поток |  |
| **UC14** | |
| Название | Переключить страницу для текужей кники |
| Роль | Авторизованный пользователь |
| Цель | Переключить страницу |
| Результат | Страница сменилась и отрисовалась, индекс страницы обновлен |
| Триггер | Стрелки по бокам экрана на странице "book" |
| Главный поток | 1. Нажатие на стрелку переключения страницы. 2. Страницы переключена, индекс страницы обновлен. |
| Альтернативный поток | 2.1 Страница не переключена, при условии что выбранной страницы не существует. |
| **UC15** | |
| Название | Отметить страницу |
| Роль | Авторизованный пользователь |
| Цель | Отметить страницу, обновить список отмеченных страниц |
| Результат | Страница отмечена |
| Триггер | Нажаие на иконку "star" на странице "book" |
| Главный поток | 1. Нажаие на иконку "star" на странице "book". 2. Страница отмечена, иконка обновилась, выбранная страница добавилась в список отмеченных страниц. |
| Альтернативный поток |  |
| **UC16** | |
| Название | Выделение цитаты |
| Роль | Авторизованный пользователь |

Продолжение таблицы 2.1

|  |  |
| --- | --- |
| Цель | Выделить цитату, и добавить ее в список цитат |
| Результат | Цитата выделена и добавлена в список цитата |
| Триггер | Выделение текста на странице книги |
| Главный поток | 1. Открытие книги. 2. Выделение текста. 3. Открывается модальное окно. 4. Нажатие на кнопку "Add citation". 5. Цитата добавлена в список цитат. 6. Модальное окно закрывается. 7. Убирается выделение текста. |
| Альтернативный поток | 4.1 Клик в не зоны модального окна. 5.1 Модальное окно закрывается. 6.1 Убирается выделение текста. |
| **UC17** | |
| Название | Использование Горячих клавиш |
| Роль | Авторизованный пользователь |
| Цель | Управление страницами книги, и позицией скрола на странице через гояриче клавиши |
| Результат | Страницы переключаются, позиция на странице меняется |
| Триггер | Клавиши "стрелки" |
| Главный поток | 1. Нажатие кнопки "стрелка влево". 2. Страница книги переключается на предыдущую. 3. Индикатор страницы обновлен. 4. Нажатие кнопки "стрелка вниз". 5. Положение страницы сместилось по вертикали. |
| Альтернативный поток | 2.1 Стрница не переключается на предыдущую, при условии что текущая страница является первой. |
| **UC18** | |
| Название | Увеличение масштаба текста в книге |
| Роль | Авторизованный пользователь |
| Цель | Увеличить масштаб книги |
| Результат | Масштаб увеличен, потери качества текста нет |
| Триггер | Нажатие на кнопку "+" на странице "book" |
| Главный поток | 1. Нажатие на кнопку "+" на странице "book". 2. Масштаб страницы увеличен. 3. Индикатор масштаба обновил значение. |
| Альтернативный поток | 2.1. Масштаб страницы не изменен, если значение масштаба уже максимальное (3). 3.1 Индикатор масштаба не изменен. |

Продолжение таблицы 2.1

|  |  |
| --- | --- |
| **UC19** | |
| Название | Уменьшение масштаба текста в книге |
| Роль | Авторизованный пользователь |
| Цель | Уменьшить масштаб книги |
| Результат | Масштаб уменьшен, потери качества текста нет |
| Триггер | Нажатие на кнопку "-" на странице "book" |
| Главный поток | 1. Нажатие на кнопку "-" на странице "book". 2. Масштаб страницы уменьшен. 3. Индикатор масштаба обновил значение. |
| Альтернативный поток | 2.1. Масштаб страницы не изменен, если значение масштаба уже максимальное (3). 3.1 Индикатор масштаба не изменен. |
| **UC20** | |
| Название | Полноэкранный режим чтения |
| Роль | Авторизованный пользователь |
| Цель | Включить полноэкранный режим для чтения книги |
| Результат | Полноэкранный режим активирован |
| Триггер | Нажатие на кнопку полноэкранного режима на странице книги |
| Главный поток | 1. Нажатие на кнопку полноэкранного режима на странице книги. 2. Меню, название книги скрыто |
| Альтернативный поток |  |
| **UC21** | |
| Название | Публикация книги |
| Роль | Авторизованный пользователь |
| Цель | Разместить книгу в магазине |
| Результат | Книга успешно размещена и домтупно к загрузке |
| Триггер | Нажатия на кнопку для публикации на книге |

Продолжение таблицы 2.1

|  |  |
| --- | --- |
| Главный поток | 1. Нажать на кнопку публикации на данной книге на странице "library". 2. Открывается модальное окно для заполнения данных для публикации книги. 3. Заполнение поле "Название книги", "Описание" (минимальная длинна 1 символ и максимальная 4000 символов), "Категория", "Теги" (формат заполнения тегов: #Название). Необходимо так же загрузить изображение обложки, поставить галочку для того чтобы убедиться в то что пользователь даете отчет о то что имеете авторские права и можете загружать в публичный доступ данную книгу. 4. После успешного заполнения всех необходимых полей, кнопка "publish" становится активна. 5. Нажатие на кноку "publish", модальное окно закрывается. 6. На странице "Shop", появляется загруженная книга, она также доступна к загрузке. |
| Альтернативный поток | 4.1 При неправильно заполненных полях, кнопка "publish" не становится активной. |
| **UC22** | |
| Название | Загрузка книги из магазина |
| Роль | Авторизованный пользователь |
| Цель | Загрузить книгу в библиотеку пользователя из магазина |
| Результат | Книга загружена и отображена в списке книг на странице "library" |
| Триггер | Нажатие на кнопку "Get" на странице "shop" |
| Главный поток | 1. Нажатие на кнопку "Get" на странице "shop" у выбранной книги. 2. Модальное окно закрывается. 3. Открывается страница "library", в список книг, добавлена выбранная книга и доступна для использования. |
| Альтернативный поток |  |

### 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 (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 Mongo DB. 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.

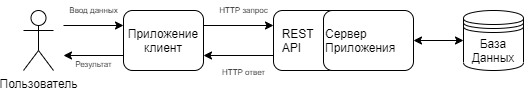


Рисунок 3.1 – REST архитектура

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. Они могут определять тип необходимого ресурса или подробности авторизации.

− 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.

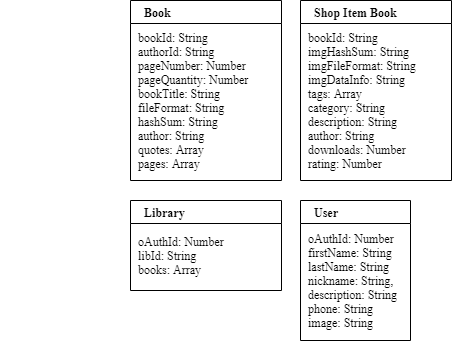


Рисунок 3.2 – Диаграмма таблиц базы данных

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.

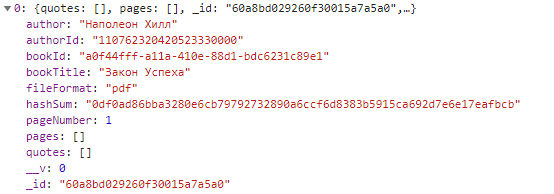


Рисунок 3.3 – Ответ сервера

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.

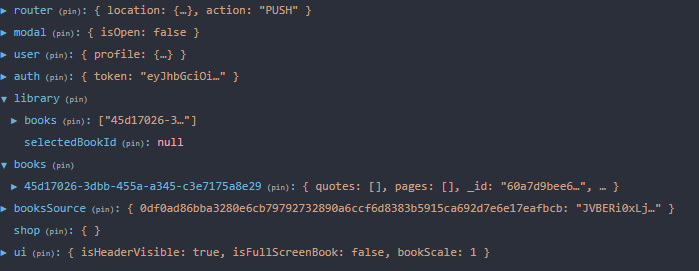


Рисунок 3.4 – Redux хранилище

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

Хорошо проработанная архитектура нужна любому ПО, и сложному, и шаблонному. С ее помощью экономится время, усилия и деньги.

Прежде чем приступить к разработке приложения, надо получить представление о назначении каждой его части. Это позволит сделать приложение более плавным и быстрым, а код более читабельным и пригодным для поддержки.

## 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) (рисунок 4.1.1).

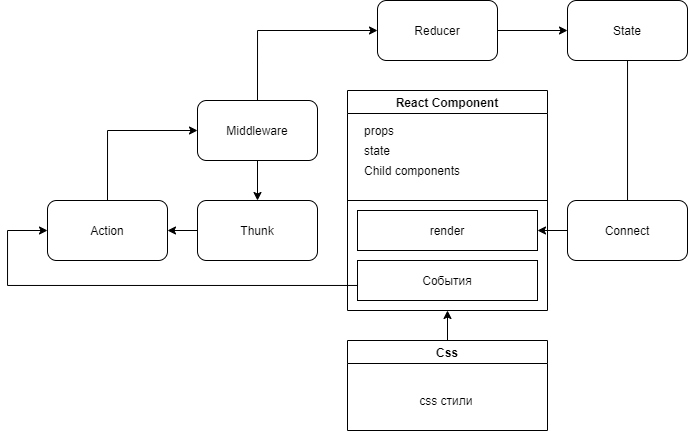


Рисунок 4.1.1 – Архитектура React Redux приложения

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 (node package manager). Формат файла JSON, в нем описывается необходимые скрипты для запуска, сборки, тестов, а также указываются зависимостей используемых модулей.

На рисункке 4.1.2 приведена структура клиентской части проекта.

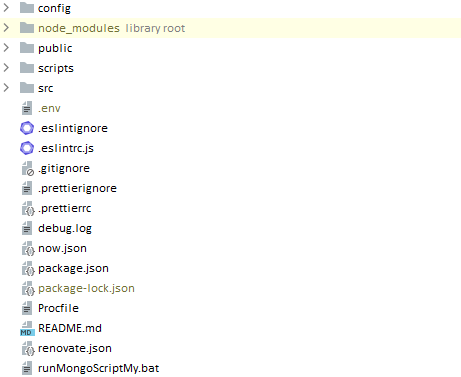


Рисунок 4.1.2 – структура клиентской части проекта

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 4.1).

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 4.1.1



Рисунок 4.1.1 – 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 4.1.2

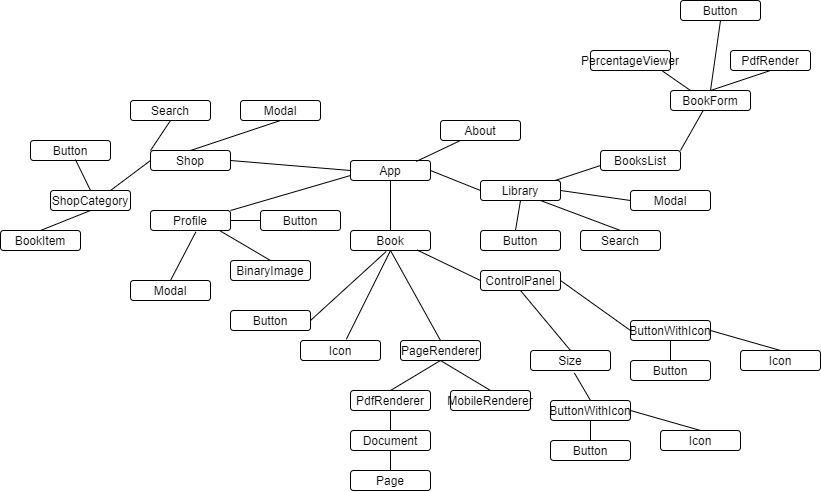


Рисунок 4.1.2 – Перечень компонентов приложения

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.

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

На рисункке 4.2.1 приведена структура клиентской части проекта.

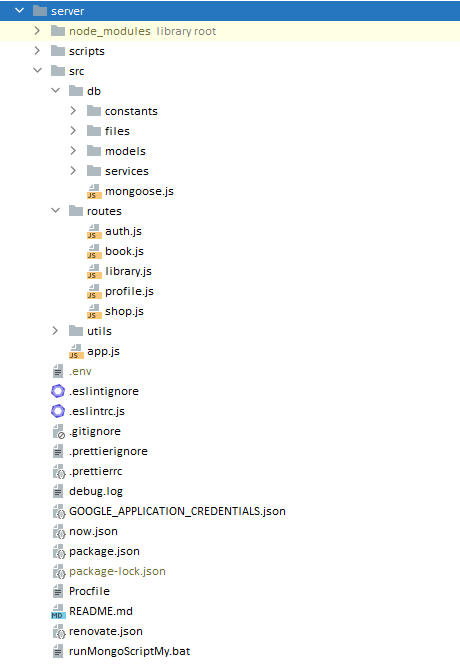


Рисунок 4.2.1 – Структура серверной части

Основная логика находятся в папке src. В scripts указаны скрипты запуска и сборки.