МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение
высшего образования
**«Южно-Уральский государственный университет
(национальный исследовательский университет)»**
Высшая школа электроники и компьютерных наук
Кафедра системного программирования

ДОПУСТИТЬ К ЗАЩИТЕ

Заведующий кафедрой, д.ф.-м.н.,
профессор

_____ Л.Б. Соколинский

«___»_____ 2024 г.

Разработка веб-приложения «Помощник автора»

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА
ЮУрГУ – 09.03.04.2024.308-340.ВКР

Научный руководитель,
доцент кафедры СП, к.ф.-м.н.
_____ Б.А. Марков

Автор работы,
студент группы КЭ-403
_____ А.В. Еремяшева

Ученый секретарь
(нормоконтролер)
_____ И.Д. Володченко
«___»_____ 2024 г.

Челябинск, 2024 г.

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение
высшего образования
«Южно-Уральский государственный университет
(национальный исследовательский университет)»
Высшая школа электроники и компьютерных наук
Кафедра системного программирования

УТВЕРЖДАЮ
Зав. кафедрой СП
_____ Л.Б. Соколинский
29.01.2024 г.

ЗАДАНИЕ
на выполнение выпускной квалификационной работы бакалавра
студентке группы КЭ-403
Еремяшевой Алёне Вячеславовне,
обучающейся по направлению
09.03.04 «Программная инженерия»

1. **Тема работы** (утверждена приказом ректора от 22.04.2024 г. № 764-13/12)

Разработка веб-приложения «Помощник автора».

2. **Срок сдачи студентом законченной работы:** 03.06.2024 г.

3. **Исходные данные к работе**

3.1. Руководство по NodeJS. [Электронный ресурс] URL: https://metanit.com/web/nodejs/ (дата обращения: 31.01.2024 г.).

3.2. VueJS для начинающих. [Электронный ресурс] URL: https://habr.com/ru/company/ru vds/blog/509700/ (дата обращения: 01.02.2024 г.).

3.3. Прохоренок Н.А. JavaScript и Node.js для веб-разработчиков. / Н.А. Прохоренок, В.А. Дорнов // СПб.: БХВ-Петербург, 2022. – 767 с.

3.4. Документация Express. [Электронный ресурс] URL: https://expressjs.com/ru/ (дата обращения: 01.03.2024 г.).

3.5. Построение сюжета – основы. [Электронный ресурс] URL: https://m-evildoer.ru/syuzhet/postroenie-syuzheta-osnovy/ (дата обращения: 31.01.2024 г.).

3.6. Десять типов структуры сценария, которые можно использовать в любом жанре. [Электронный ресурс] URL: https://snegiri-studio.ru/blog/389910 (дата обращения: 02.02.2024 г.).

4. Перечень подлежащих разработке вопросов

4.1. Выполнить анализ предметной области и произвести обзор существующих решений.

4.2. Спроектировать архитектуру системы.

4.3. Реализовать веб-приложение для проработки идей и сюжета книги.

4.4. Провести тестирование веб-приложения.

5. Дата выдачи задания: 29.01.2024 г.

Научный руководитель,
доцент кафедры СП, к.ф.-м.н. Б.А. Марков

Задание принял к исполнению А.В. Еремяшева

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Актуальность

В современном мире, где информационная перегрузка и конкуренция в сфере литературы и контента достигают высоких уровней, приложение, которое помогает авторам прорабатывать идеи, сюжеты и персонажей, является особенно актуальным.

С развитием технологий и доступностью цифровых инструментов, использование информационных технологий для работы над литературными проектами становится неотъемлемой частью современного писательского процесса. Использование современных технологий облегчает процесс творчества, позволяя писателям организовать и хранить свои идеи в удобном формате. Это способствует повышению эффективности и продуктивности в создании литературного произведения и позволяет ускорить и упростить процесс работы над произведением.

Приложение, которое помогает авторам прорабатывать идеи, сюжеты и персонажей, позволяет писателям пройти путь от наброска идеи, отдельного события или одного не до конца проработанного персонажа до полноценного сюжета, расписанного во всех подробностях и тщательно продуманных персонажей, каждый из которых занимает в истории свое место. Благодаря такому приложению, авторы могут более осознанно реализовывать свои идеи, делая свои произведения более интересными и запоминающимися для читателей. Такие инструменты содействуют развитию литературного мастерства и способствуют созданию качественной литературы.

Постановка задачи

Целью выпускной квалификационной работы является разработка веб-приложения «Помощник автора». Для достижения поставленной цели необходимо решить следующие задачи:

1) выполнить анализ предметной области и произвести обзор существующих решений;

2) спроектировать архитектуру системы;

3) реализовать программное обеспечение для проработки идей;
4) провести тестирование веб-приложения.

Структура и содержание работы

Работа состоит из введения, четырех глав, заключения и списка литературы. Объем работы составляет 46 страниц, объем списка литературы – 19 источников.

Первая глава «Анализ предметной области» содержит постановку задачи, описание предметной области, обзор решений и анализ аналогов со сравнением их функционала для дальнейшего проектирования интерфейса и функционала веб-приложения.

Вторая глава «Проектирование веб-приложения» содержит описание и анализ требований к веб-приложению, рассмотрение вариантов использования веб-приложения, общее описание архитектуры веб-приложения, проектирование базы данных и проектирование пользовательского интерфейса с макетами.

Третья глава «Реализация системы» описывает использованные средства реализации и подробности реализации клиентской и серверной частей веб-приложения.

Четвертая глава «Тестирование системы» включает описание процесса тестирования и результаты тестирования веб-приложения.

1. АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

1.1. Описание предметной области

Автор – это создатель какого-либо произведения. Литературное произведение – это творческое произведение в письменной форме, которое может включать в себя различные жанры и формы литературы, такие как романы, рассказы, стихи, пьесы и др. Чтобы произведение было целостным, оно должно иметь четкую проработанную структуру. Структура [1] – это способ связи составляющих элементов. Несмотря на оригинальность конкретных произведений, структурная модель любого литературного произведения состоит из оболочки и ядра. Оболочкой является текст. Ядро включает в себя тему и идею и имеет собственное внутреннее строение. Внутреннюю форму ядра составляют персонажи и их взаимодействие, то есть сюжет, а внешнюю – авторский стиль.

Для написания полноценного произведения автор должен проработать все элементы структуры, чтобы все они гармонично перекликались между собой, дополняя и раскрывая друг друга. Это все возможно сделать без дополнительных инструментов, но это трудоемкий процесс, который значительно усложняется по мере увеличения масштабов планируемого произведения.

Современные технологии позволяют облегчить работу автора. Появляются приложения, призванные помочь авторам грамотно организовать элементы произведения, подробно расписать каждый из них, и иметь возможность в любой момент легко исправить уже написанное.

1.2. Обзор решений

Рассмотрим, какие приложения предоставляют функционал, который может помочь автору при работе с текстом, персонажами, сюжетными линиями и другими важными частями будущего произведения.

Текстовые редакторы [2] – основной инструмент любого автора. Самый популярный текстовый редактор – Microsoft Word. Он предоставляет

широкий функционал для работы с текстом, позволяет настроить параметры страниц и форматировать текст по необходимости. Интерфейс приложения представлен на рисунке 1.

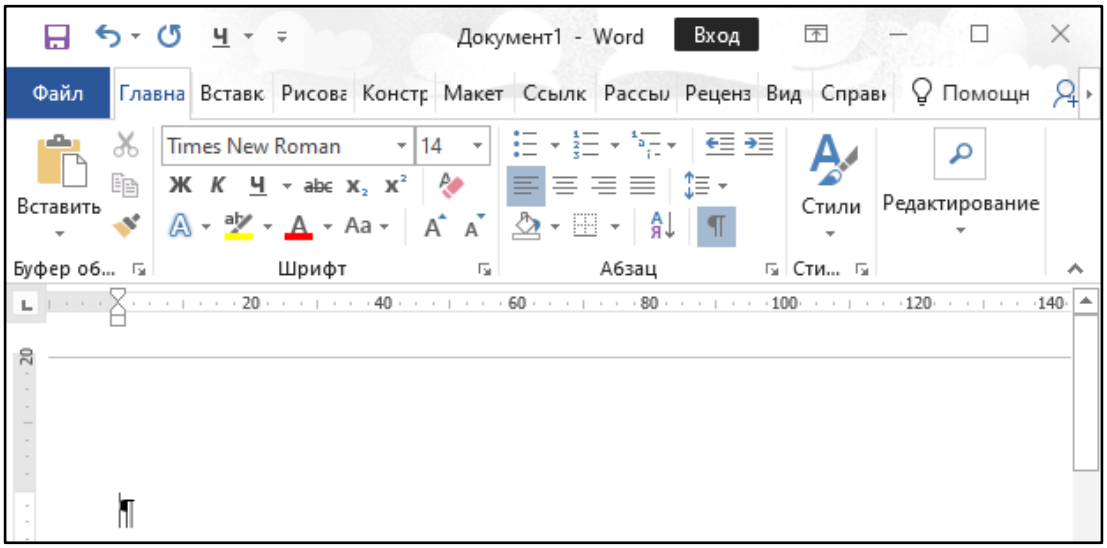

Рисунок 1 – Скриншот приложения Microsoft Word

Для записи каких-либо обрывочных идей можно использовать приложения для заметок. Evernote [3] – одно из приложений, предоставляющих подобный функционал. Помимо создания заметок Evernote дает пользователю возможность создавать списки дел и добавлять эти списки в календарь. Интерфейс приложения представлен на рисунке 2.

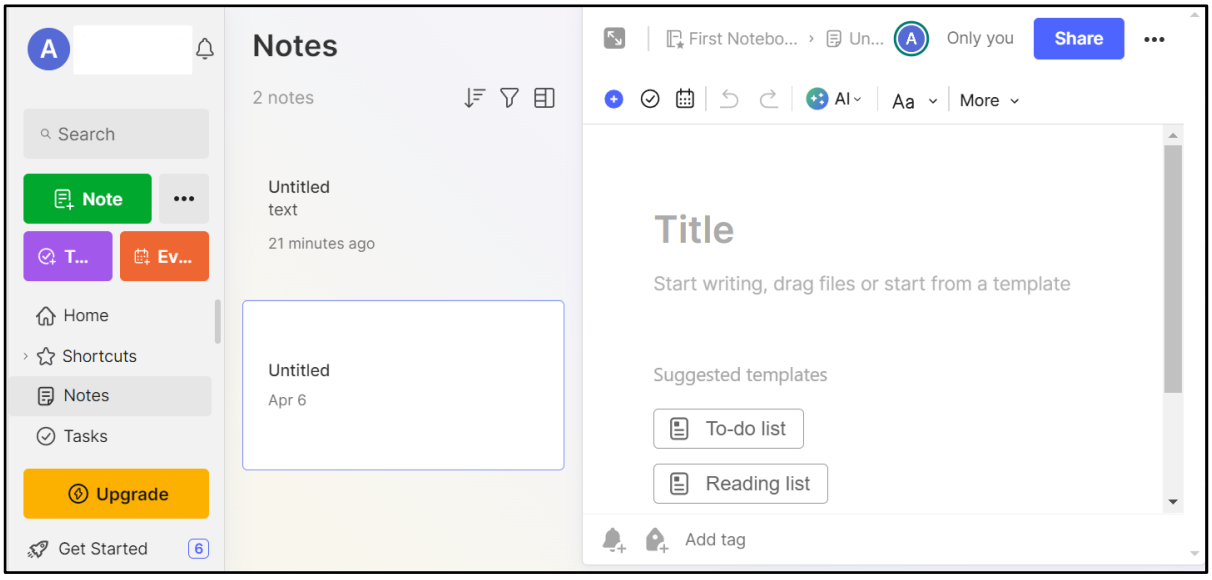

Рисунок 2 – Скриншот приложения Evernote

Карты мыслей [4] – способ фиксации идей с сохранением логики их развития. Подобный подход можно удобно использовать в планировании и проектировании будущего произведения. Карта мыслей может помочь как в развитии общего сюжета – отображении его разветвлений, так и в работе с отдельными сюжетными линиями персонажей – демонстрации их личностного роста по ходу сюжета. На рисунке 3 представлен интерфейс одного из приложений для создания карт мыслей – Mind Meister [5].

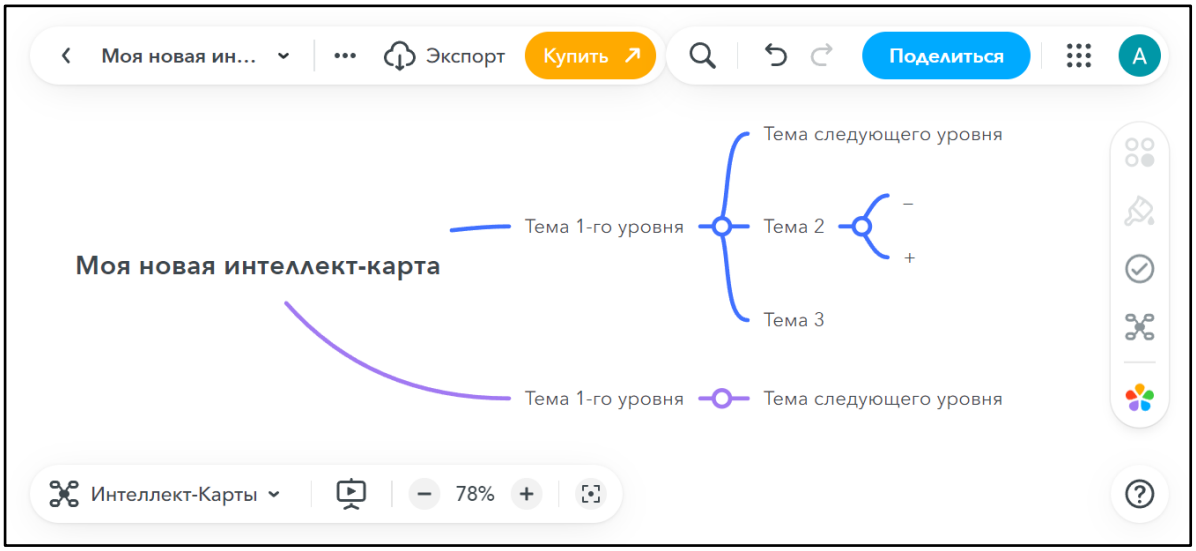

Рисунок 3 – Скриншот приложения MindMeister

1.3. Анализ аналогов

Помимо стандартных текстовых редакторов, приложений с заметками или редакторов диаграмм, предоставляющих требуемый функционал по отдельности, существуют приложения, разработанные специально для авторов. Они объединяют необходимые функции для помощи автору в написании произведения и позволяют хранить информацию обо всех элементах произведения в едином пространстве с делением по разделам для навигации. Рассмотрим несколько наиболее популярных приложений: Novel Factory, bibsco и Plottr.

Novel Factory [6]

Приложение Novel Factory, интерфейс которого представлен на рисунке 4, это веб-приложение, созданное для помощи автору при написании произведения.

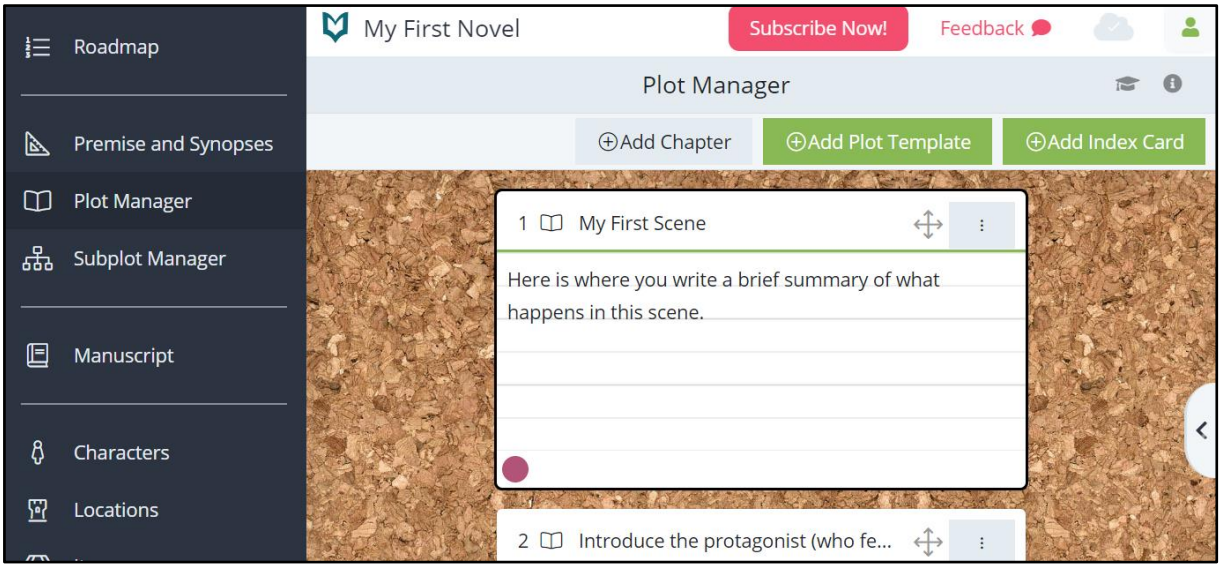

Рисунок 4 – Скриншот приложения Novel Factory

Функционал приложения включает в себя два инструмента для проработки последовательности событий. Первый позволяет распланировать основную сюжетную линию с ключевыми точками, а второй помогает сопоставить события основной линии с второстепенными сюжетными линиями. Также это приложение позволяет написать краткий обзор будущего произведения, составить список персонажей, локаций и каких-либо сюжетно важных предметов. Для каждого предмета, персонажа или локации в приложении создается отдельная карточка, в которой можно подробно расписать всю необходимую информацию.

Кроме того, в Novel Factory есть встроенный текстовый редактор с привязкой написанного текста к конкретному событию из последовательности. Помимо этого, приложение позволяет делать заметки, определять цели написания и отслеживать их выполнение. Novel Factory также позволяет пользователю использовать шаблоны сюжетов, основанные на популярных

жанрах и предоставляет подробное руководство по написанию произведения. Приложение автоматически сохраняет рукопись и имеет историю версий, что позволяет получить доступ к предыдущим черновикам.

Приложение bibisco [7]

Приложение bibisco – настольное приложение-помощник для авторов, представленное на рисунке 5.

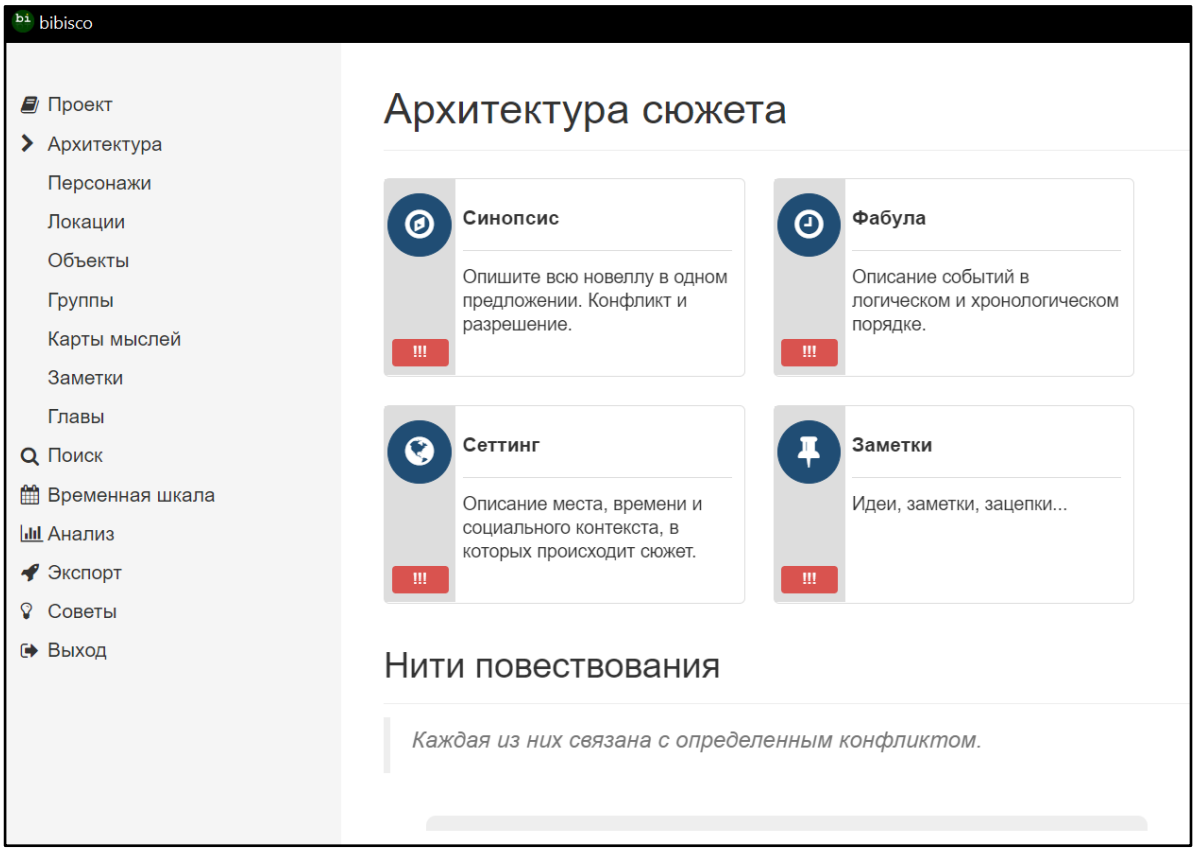

Рисунок 5 – Скриншот приложения bibisco

Это приложение похоже на Novel Factory, но в нем нет шаблонов для сюжета, а вместо руководства для написания добавлен раздел с советами для автора. Другая отличительная особенность – наличие карт мыслей, позволяющих визуально отобразить связи между любыми элементами сюжета, будь то персонажи, локации или события, приложение не ставит никаких ограничений и звеном на карте мыслей может являться любая часть будущей истории на усмотрение пользователя. Также приложение предлагает делить персонажей на основных и второстепенных. Основные персонажи

имеют подробную анкету, тогда как второстепенные – только одно текстовое поле, позволяющее описать, какую роль они играют в сюжете. Раздел с архитектурой позволяет переключаться между 4 подразделами: синопсисом, фабулой, сеттингом и заметками.

Все подразделы представляют собой многострочное поле ввода с инструментами форматирования текста. Отличаются они заголовками и описаниями. В разделе с архитектурой также есть подраздел с нитями повествования. Каждая созданная пользователем нить повествования оформляется так же, как остальные подразделы архитектуры. Кроме всего вышеперечисленного в приложении есть раздел с главами. Каждая глава состоит из сцен, заметок и причины. Сцены содержат описание происходящих событий, а причины описывают значимость сцены в сюжете. Также bibisco предоставляет статистику по написанным главам, персонажам и прочим элементам произведения. В приложении пользователь может расположить события на временной шкале, а также имеет возможность экспортировать написанное в форматы PDF, EPUB, TXT и DOCX.

Plottr [8]

Plottr – еще одно настольное приложение, призванное помочь автору в написании произведения. Скриншот приложения представлен на рисунке 6.

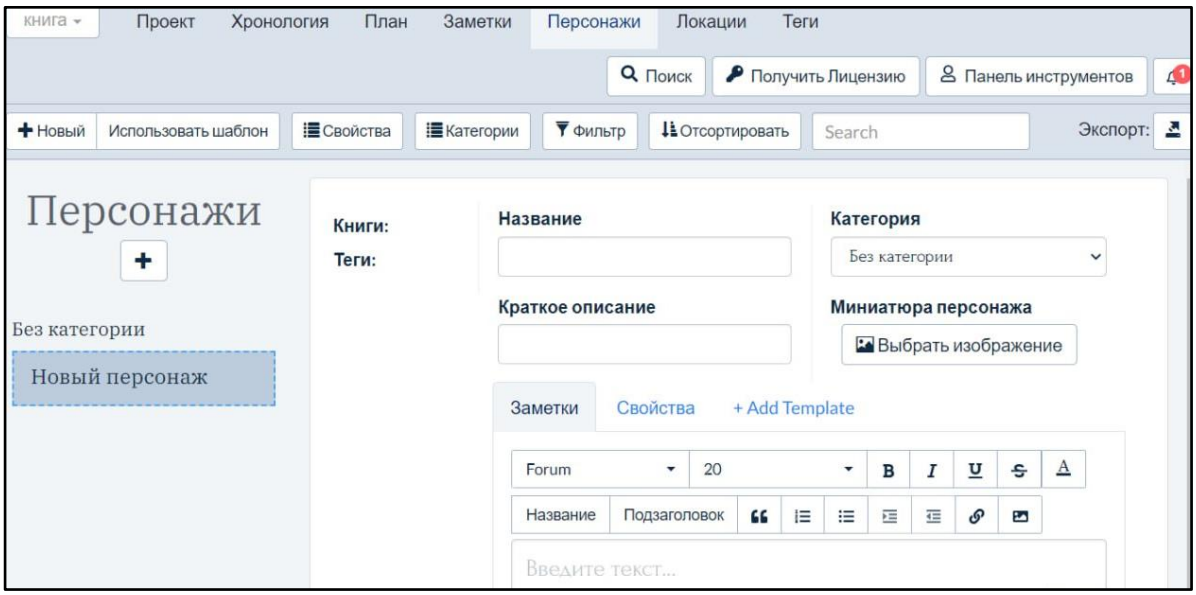

Рисунок 6 – Скриншот приложения Plottr

Оно также, как Novels Factory и bibsco содержит инструменты для планирования последовательности событий с возможностью добавления нескольких сюжетных линий, характеристик персонажей и локаций, составления заметок по произведению. В этом приложении нет шаблонов и все элементы структуры произведения оформлены одинаково и требуют обязательного заполнения полей, отвечающих за связь между элементами сюжета, персонажами и прочими элементами произведения. Приложение предоставляет возможность создания шаблонов и копий уже описанных элементов.

1.4. Сравнение функционала

В ходе анализа были рассмотрены три приложения, объединяющие в себе сразу несколько функций. Приложение Novel Factory объединяет в себе работу с текстом, заметками, карточками персонажей и сюжетной линией. Его основным плюсом является предоставление шаблонов для структуры сюжета, позволяющих автору не прописывать последовательность основных сюжетных точек самостоятельно. Такие элементы как персонажи, локации или значимые для истории объекты представлены в виде карточек с полями для заполнения и возможностью добавлять свои поля.

Второе рассмотренное приложение – bibisco. Заполнение информации об элементах сюжета делится на несколько этапов, каждый из которых выносится на отдельном экране без возможности увидеть предыдущий этап, что создает нежелательные паузы и не дает возможности оценить общий объем информации об элементе. В приложении есть советы для автора, возможность указывать порядок событий и хранить информацию о книге локально в выбранном пользователем формате.

Последнее приложение, упомянутое в обзоре – Plottr. В данном приложении все макеты для заполнения информации об элементах сюжета ничем не отличаются друг от друга. Нажимая на разделы, подразумевающие разные элементы историй, пользователь видит одинаковый шаблон без отличительных признаков. С одной стороны, таким образом пользователю не

ставят никаких ограничений, но с другой, однообразие затрудняет определение функционала текущего раздела. Также в этом приложении нет подсказок и какой-либо помощи в использовании для создания произведений.

Существенным недостатком всех рассмотренных аналогов является то, что доступ к их функционалу можно получить только оформив пробную, или полноценную подписку, ни одно из приложений не является полностью доступным.

Из трех рассмотренных в обзоре приложений наиболее удобным, простым и понятным для пользователя является Novel Factory, оно предоставляет инструкции по написанию, шаблоны для структуры сюжета и отличающиеся макеты для заполнения информации об элементах структуры произведения.

Выводы по первой главе

В ходе первой главы был проведен сравнительный анализ аналогов и инструментов. После анализа приложений-инструментов, выполняющих одну функцию и сравнительного анализа приложений-помощников для авторов, можно сделать вывод о нюансах, на которые стоит обратить внимание при разработке подобного приложения. Приоритетом являются простота и удобство использования приложения. Стоит исключить однообразные макеты для заполнения в разных разделах приложения, а также избегать пошагового заполнения какого-либо раздела. Необходимо добавить разделы с карточками персонажей, заметками для записи идей и непосредственно рукописью. Помимо этого, стоит включить карты мыслей как способ отображения линии повествования.

Необходимо добавить возможность выстраивания связей между элементами разных разделов (например, персонажей и ключевых моментов линии повествования), но сделать это опциональным, чтобы пользователь мог пропустить этот шаг и вернуться к нему в любой момент по необходимости. Для удобства навигации по рукописи можно так же сделать привязку ее частей к элементам линии повествования.

2. ПРОЕКТИРОВАНИЕ ВЕБ-ПРИЛОЖЕНИЯ

2.1. Требования к веб-приложению

Функциональные требования

При разработке программного обеспечения функциональные требования определяют функции, которые должно выполнять все приложение или только один из его компонентов, то есть функциональные требования описывают, что должна делать система.

В результате анализа предметной области и обзора существующих решений были сформированы следующие требования.

1. Система должна предоставлять возможность создания сущности книги для дальнейшего взаимодействия с ее элементами.
2. Система должна предоставлять возможность создания и редактирования заметок для планируемой книги.
3. Система должна предоставлять возможность составления и редактирования карты мыслей для планируемой книги.
4. Система должна предоставлять возможность создания и редактирования карточек персонажей для планируемой книги.
5. Система должна предоставлять возможность редактирования текста планируемой книги.

Нефункциональные требования

Нефункциональные требования определяют стандарты производительности и атрибуты качества программного обеспечения, например, удобство использования системы, эффективность, безопасность, масштабируемость. То есть функциональные требования определяют, что система делает, а нефункциональные – как система это делает.

Рассмотрим нефункциональные требования системы.

1. Система должна быть разработана на языке программирования JavaScript.
2. Система должна представлять собой веб-приложение.

3. Система должна быть разработана с использованием Vue JS для клиентской части приложения.

4. Система должна быть разработана с использованием Node JS для серверной части приложения.

5. Система должна использовать PostgreSQL в качестве системы управления базой данных.

2.2. Варианты использования

На основе требований, предъявляемых к разрабатываемому приложению, были разработаны варианты его использования, которые изображены на диаграмме (рисунок 7).

В ходе проектирования в системе веб-приложения «Помощник автора» был выделен один актер: пользователь.

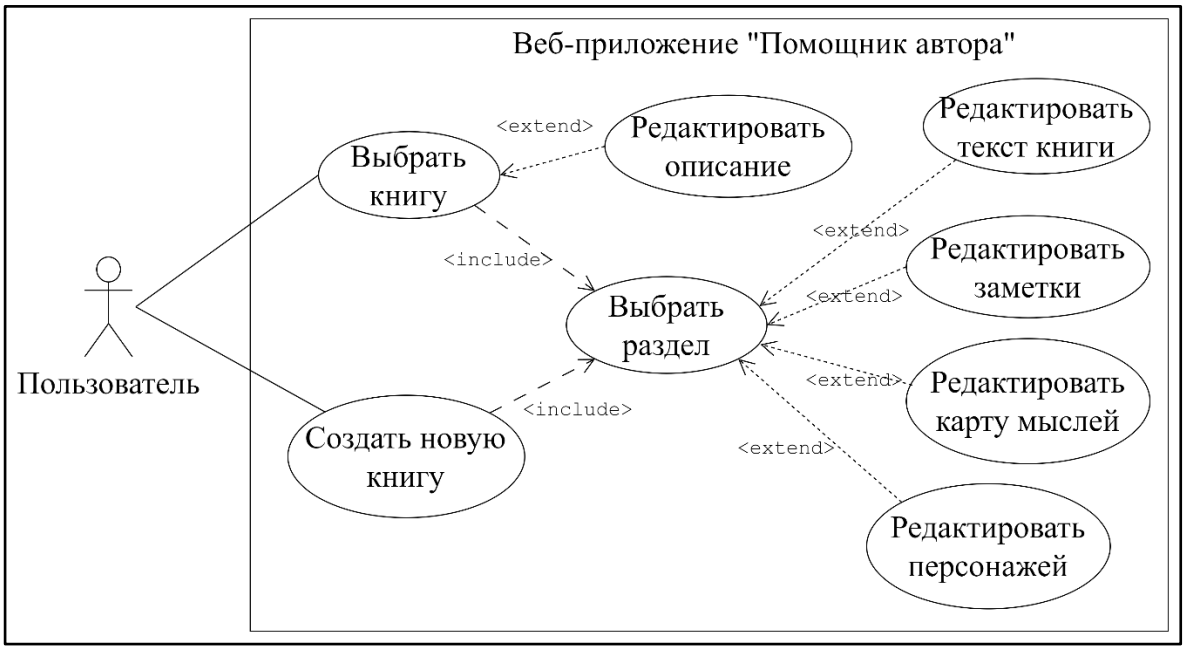

Рисунок 7 – Варианты использования веб-приложения

Пользователь создает новую книгу или выбирает существующую. Создав новую книгу, пользователь может редактировать описание, а может открыть один из четырех разделов, содержащих информацию об элементе структуры произведения, и записать в него необходимую информацию. Вы-

брав одну из существующих книг, пользователь получает возможность редактировать написанное ранее описание, либо также выбрать один из четырех разделов и редактировать или удалить информацию, записанную в нем, или записать туда что-либо, если раздел ранее не заполнялся. После внесения любых изменений в содержимое разделов пользователь может сохранить их.

2.3. Общее описание архитектуры веб-приложения

Для реализации веб-приложения будет применена клиент-серверная архитектура с трехуровневой архитектурой [9]. Приложения, созданные на основе данной архитектуры, состоят из клиента, сервера и базы данных. Клиенты запускают приложения, которые инициируют запросы к серверам для выполнения каких-либо операций или получения определенных ресурсов. Серверы в свою очередь обращаются к базе данных для получения или сохранения данных. Клиент и сервер физически представляют собой две программы. Базы данных: это хранилища информации, которые используются на серверной стороне для хранения и управления данными. Базы данных позволяют серверу эффективно хранить, организовывать и извлекать информацию по запросу клиента.

Веб-приложение будет разрабатываться на языке программирования JavaScript. Для клиентской части было принято решение использовать Vue.js [10] – фреймворк с открытым кодом для создания пользовательских интерфейсов. Основными его преимуществами являются быстрая разработка и доступность, что позволяет легче находить решения для проблем, возникающих при разработке. Широкое сообщество энтузиастов влечет за собой обилие существующих шаблонов, библиотек и других инструментов для упрощения работы над проектом. Для серверной части были выбраны платформа Node.js [11] и фреймворк Express [12]. Использование фреймворка обусловлено тем, что он предоставляет ряд готовых абстракций, которые упрощают создание сервера и серверной логики.

2.4. Проектирование базы данных

Для хранения данных о книгах и их элементов была спроектирована база данных (БД). На рисунке 8 изображена схема базы данных приложения.

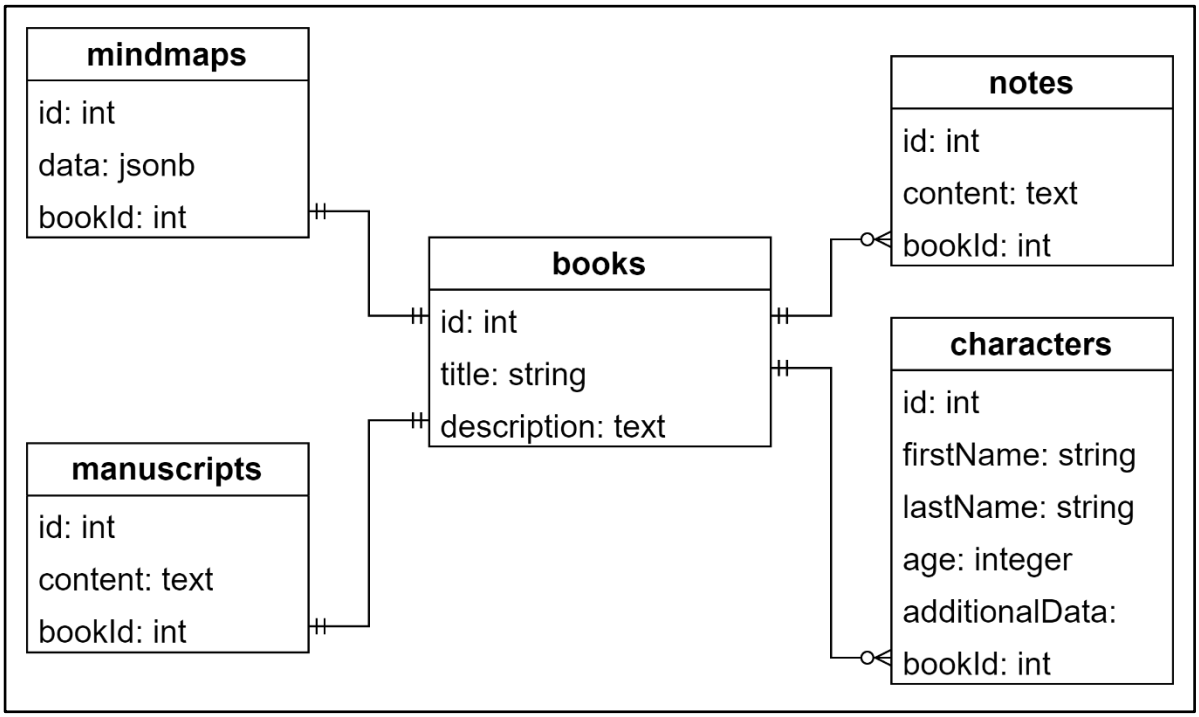

Рисунок 8 – Схема базы данных

Таблица Books содержит основные данные о созданных книгах: название и описание. Название хранится в виде строки, а описание в формате json.

Таблица Mindmaps содержит информацию о картах мыслей, каждая из которых обращается к книге, с которой связана по внешнему ключу bookId связью один к одному. Расположение узлов карты мыслей и их наполнение хранится в поле data в формате json.

Таблица Manuscripts хранит данные о рукописях для книг. Рукописи и книги так же соединены связью один к одному. У рукописей есть внешний ключ bookId, указывающий с какой книгой связана рукопись. Содержимое рукописи хранится в поле content в виде текста в формате json.

Таблица Characters хранит данные о персонажах. В данной таблице хранятся имя, фамилия и возраст персонажа. Кроме этого, есть поле additionalData, хранящее информацию о пользовательских характеристиках персонажа в формате json. Таблица персонажей имеет первичный ключ bookId,

указывающий на связь между персонажами и книгой. Книги и персонажи имеют связь один ко многим.

Таблица Notes содержит информацию о заметках. Текст заметок хранится так же в формате json. В таблице заметок хранится первичный ключ bookId, отражающий, к какой книге обращаются заметки. Книги и заметки имеют связь один ко многим.

2.5. Проектирование пользовательского интерфейса

Приложение представляет собой многостраничное веб-приложение. Оно состоит из 5 страниц и панели навигации, позволяющей переключаться между ними.

При открытии приложения открывается главная страница, которая отображает список книг, созданных пользователем либо информацию о том, что еще не было создано ни одной книги. На рисунке 9 представлен макет главной страницы веб-приложения.

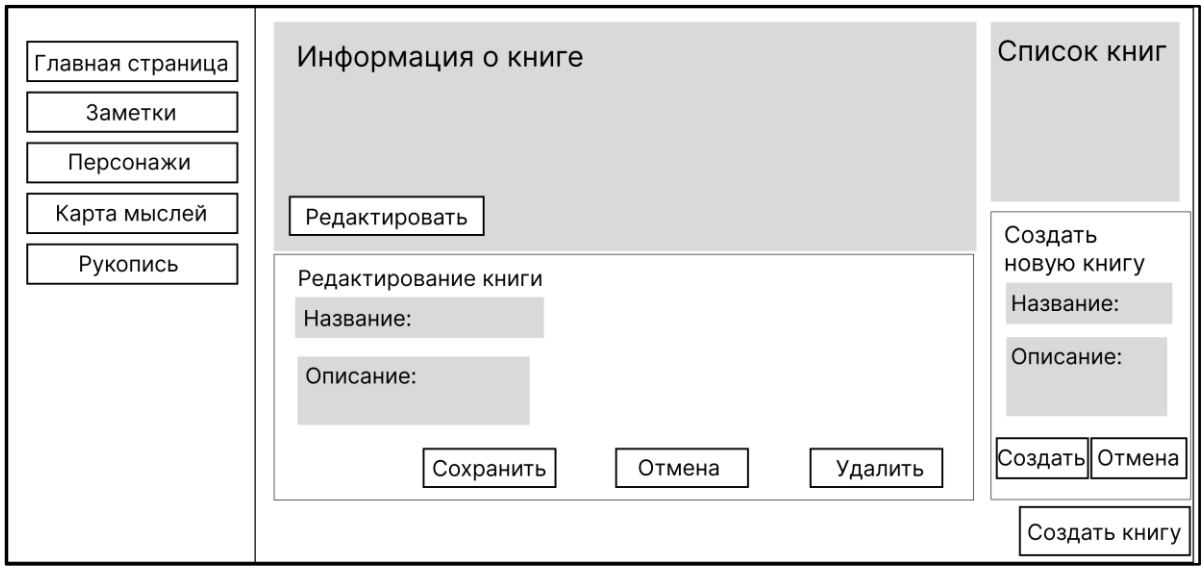

Рисунок 9 – Макет главной страницы

При нажатии на кнопку «Создать книгу» отображается окно создания книги, которое предлагает заполнить название и описание книги. После нажатия на кнопку «Создать» в списке появляется новая книга. При нажатии

на кнопку «Редактировать» при просмотре информации о книге пользователь получает возможность редактировать информацию о книге или удалить книгу. Кнопка «Сохранить» позволяет сохранить результат редактирования. Кнопка «Удалить» удаляет существующую книгу. Кнопка «Отмена» закрывает окна редактирования и создания книги.

При нажатии на кнопку «Заметки» на панели навигации, открывается страница с заметками для книги. На рисунке 10 представлен макет для страницы заметок.

Рисунок 10 – Макет страницы с заметками

При нажатии на кнопку «Создать заметку» текст из поля выше отображается в одной из плиток списка заметок. В этом списке заметки можно редактировать. Нажатие на кнопку «Удалить заметку» удаляет заметку из списка. Сохранение изменения заметок происходит автоматически сразу после того, как пользователь заканчивает вносить изменения в текст заметки.

При нажатии на кнопку «Персонажи» на панели навигации, открывается страница с персонажами. Макет страницы с персонажами представлен на рисунке 11.

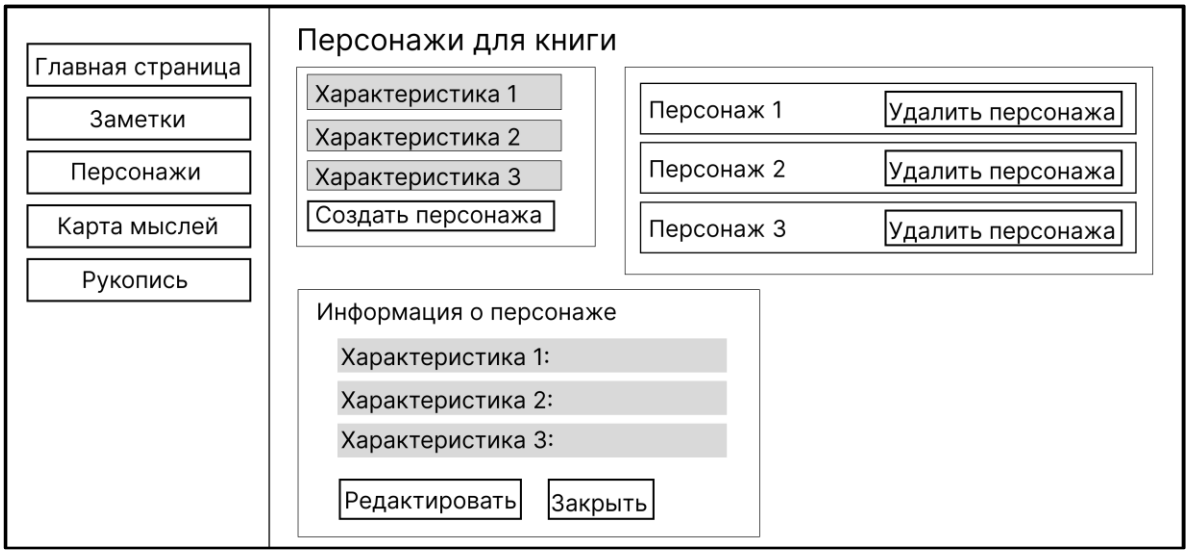

Рисунок 11 – Макет страницы с персонажами

На странице с персонажами слева расположена форма с полями для создания персонажа. Справа располагается список персонажей, в котором можно открывать карточки персонажей для просмотра и удалять персонажей. При выборе персонажа из списка под элементами создания и редактирования открывается карточка персонажа с подробной информацией и возможностью редактирования персонажа.

При нажатии на кнопку «Редактировать» в карточке персонажа, открывается окно редактирования персонажа, представленное на рисунке 12.

Рисунок 12 – Макет окна редактирования персонажа

Кнопка «Добавить поле» позволяет создать пользовательское поле, у которого можно задавать не только содержимое, но и название самого поля. После сохранения пользовательского поля оно отображается в карточке персонажа наравне с другими полями.

При нажатии на кнопку «Карта мыслей» на панели навигации, открывается страница с картой мыслей для книги, макет которой представлен на рисунке 13

Рисунок 13 – Макет страницы с картой мыслей

На странице с картой мыслей открывается редактор диаграмм, позволяющий создавать, настраивать и удалять узлы и связи между ними. При нажатии на кнопку «Создать узел» на холсте редактора рисуется узел с заданными по умолчанию текстом и размерами. Узел можно свободно перемещать по карте, связать с другими узлами или удалить. При нажатии на кнопку «Создать связь» можно выбрать два узла, которые должны быть связаны между собой, и между ними будет создана связь. Связи между узлами удаляются при нажатии на кнопку «Удалить связь».

Последняя страница – это страница с рукописью, которая открывается на нажатии на кнопку «Рукопись». Макет страницы представлен на рисунке 14.

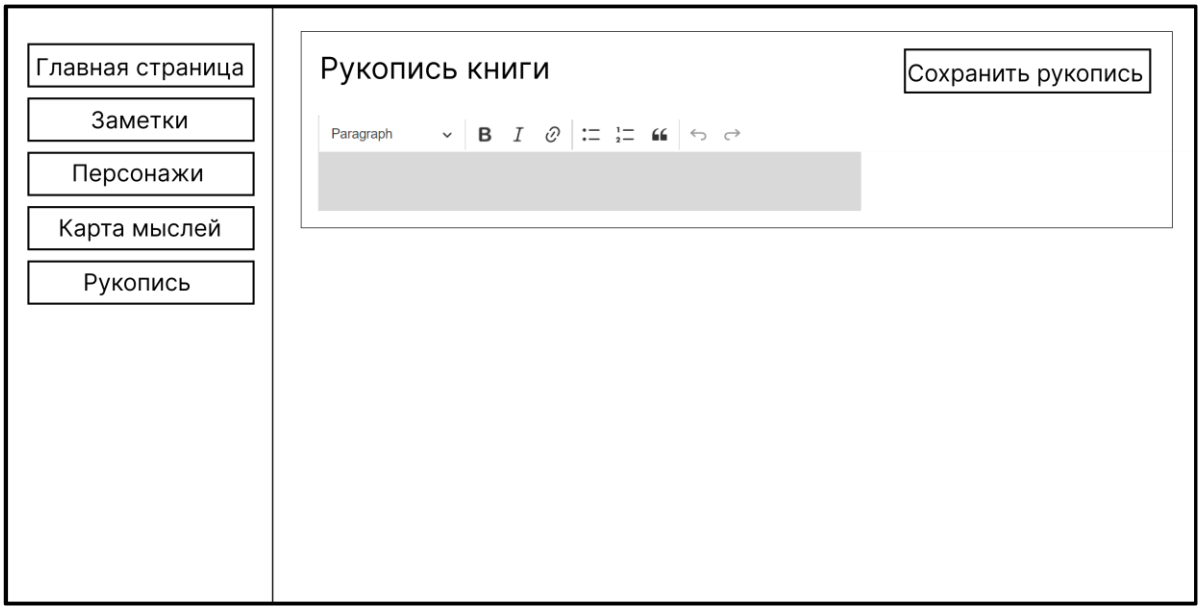

Рисунок 14 – Макет страницы с рукописью

На странице с рукописью отображается текст рукописи с возможностью форматирования. Панель форматирования позволяет менять форму и размер шрифта, создавать списки и добавлять цитирование. Кнопка «Сохранить рукопись» сохраняет изменения, внесенные в поле ввода текстового редактора.

Выводы по второй главе

В ходе проектирования были выявлены функциональные и нефункциональные требования к приложению, спроектирована диаграмма вариантов использования и описана общая архитектура веб-приложения и используемые средства реализации. Кроме того, была спроектирована база данных и разработаны макеты пользовательского интерфейса.

3. РЕАЛИЗАЦИЯ СИСТЕМЫ

3.1. Средства реализации

В ходе проектирования были выбраны средства реализации клиентской и серверной частей приложения. Серверная часть приложения реализована с использованием платформы Node.js и фреймворка Express. Для реализации клиентской части используется фреймворк Vue.js.

Помимо уже определенных в процессе проектирования средств реализации, необходимы система управления базой данных (СУБД) и инструменты для организации взаимодействия частей приложения между собой.

В качестве СУБД была выбрана PostgreSQL [13]. Преимуществами данной СУБД являются доступность, кроссплатформенность, возможность работать с разными типами данных и хранить большие объемы данных.

Во избежание прямого взаимодействия с SQL-запросами был использован ORM (Object Relational Mapping) фреймворк Sequelize [14], который позволяет работать с базами данных с помощью объектно-ориентированного подхода.

Для взаимодействия с клиентской частью через стандартные HTTP-запросы в серверной части приложения используется REST API.

Для обеспечения взаимодействия клиентской части приложения с серверной используется библиотека Axios [15]. Axios поддерживает стандартные методы для отправки HTTP-запросов на сервер и обработки ответов.

На рисунке 15 изображена архитектура веб-приложения с учетом выбранных инструментов.

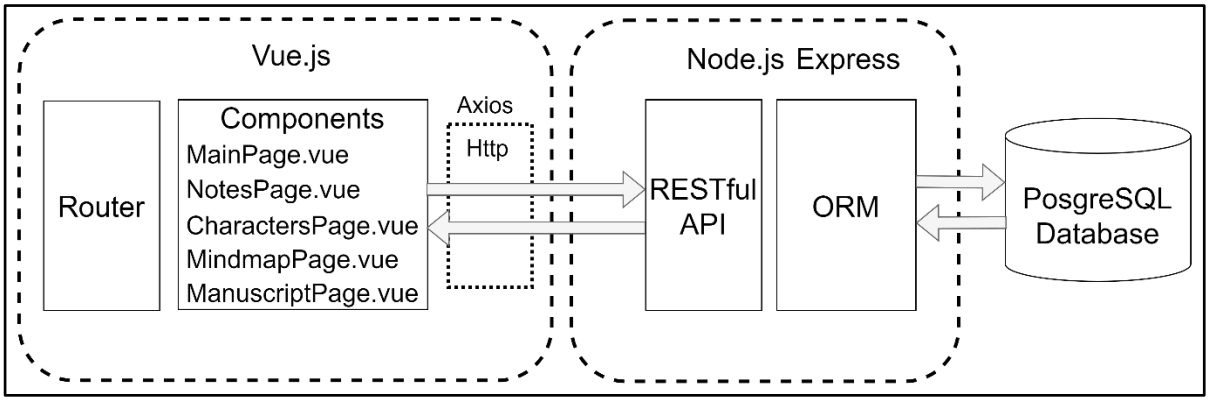

Рисунок 15 – Архитектура веб-приложения

3.2. Реализация серверной части приложения

Настройка сервера

Первый шаг в настройке сервера – инициализация Node.js приложения. Она выполняется с помощью менеджера пакетов npm, команды `npm init` и последующего указания параметров приложения, которые в дальнейшем отображаются в файле «package.json». В листинге 1 представлено содержимое файла «package.json».

Листинг 1 – Код файла «package.json»

```
{
  "name": "back",
  "version": "1.0.0",
  "description": "",
  "main": "server.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "AlyonaEremyasheva",
  "license": "ISC",
  "dependencies": {
    "body-parser": "^1.20.2",
    "cors": "^2.8.5",
    "express": "^4.19.2",
    "pg": "^8.11.5",
    "pg-hstore": "^2.3.4",
    "sequelize": "^6.37.3"
  }
}
```

Файл «package.json» хранит список пакетов, необходимых для проекта с нужными версиями. В этом файле видно, что помимо описанных библиотек также установлены пакеты cors и body-parser. Механизм cors [16] используется для того, чтобы обеспечить странице доступ к сторонним ресурсам. Он необходим чтобы обеспечить клиентской и серверной частям приложения доступ друг к другу, так как они запускаются на разных портах.

Следующий этап – непосредственно настройка сервера в файле «server.js». Код из файла «server.js» представлен в листинге 2.

Листинг 2 – Код настройки сервера

```
const express = require("express");
const bodyParser = require("body-parser");
const cors = require("cors");
const app = express();
var corsOptions = {
  origin: "http://localhost:8081"};
```

```
app.use(cors(corsOptions));
app.use(express.json());
app.use(express.urlencoded({ extended: true }));
const db = require("./app/models");
db.sequelize.sync()
    .then(() => {
        console.log("Synced db.");
    })
    .catch((err) => {
        console.log("Failed to sync db: " + err.message);
    });
const bookRoutes = require("./app/routes/book.routes");
const characterRoutes = require("./app/routes/character.routes");
const noteRoutes = require("./app/routes/note.routes");
const manuscriptRoutes = require("./app/routes/manuscript.routes");
const mindmapRoutes = require("./app/routes/mindmap.routes");
app.use("/api/books", bookRoutes);
app.use("/api/character", characterRoutes);
app.use("/api/notes", noteRoutes);
app.use("/api/manuscript", manuscriptRoutes);
app.use("/api/mindmap", mindmapRoutes);
const PORT = process.env.PORT || 8080;
app.listen(PORT, () => {
  console.log(`Server is running on port ${PORT}.`);
});
```

В данном файле импортируются модули, необходимые для работы сервера, и монтируются с помощью `app.use`. Источником серверного приложения становится порт 8081. Далее прописываются пути к файлам, где содержатся маршруты для основных сущностей приложения, и так же монтируются. Сервер настраивается на прослушивание порта 8080, на котором располагается клиентское приложение.

Настройка базы данных PostgreSQL и Sequelize

После настройки сервера необходимо настроить базу данных. Это делается с помощью создания файла «db.config.js». Листинг 3 демонстрирует содержимое этого файла.

Листинг 3 – Код настройки конфигурации БД

```
module.exports = {
    HOST: "localhost",
    USER: "postgres",
    PASSWORD: "547816",
    DB: "helper_db",
    dialect: "postgres"
  };
```

В файле «db.config.js» задаются параметры для подключения к базе данных helper_db в PostgreSQL.

Следующий этап – инициализация Sequelize. Это действие выполняется в файле «index.js», который находится в директории models, которая хранит описание моделей. Код файла «index.js» представлен в листинге 4.

Листинг 4 – Инициализация Sequelize

```
const dbConfig = require("../config/db.config.js");
const Sequelize = require("sequelize");
const sequelize = new Sequelize(dbConfig.DB, dbConfig.USER, dbConfig.PASS-
WORD, {
  host: dbConfig.HOST,
  dialect: dbConfig.dialect,
  operatorsAliases: false
});
const db = {};
db.Sequelize = Sequelize;
db.sequelize = sequelize;
db.book = require("./book.model.js")(sequelize, Sequelize);
db.character = require("./character.model.js")(sequelize, Sequelize);
db.manuscript = require("./manuscript.model.js")(sequelize, Sequelize);
db.note = require("./note.model.js")(sequelize, Sequelize);
db.mindmap = require("./mindmap.model.js")(sequelize, Sequelize);
db.book.hasOne(db.manuscript);
db.manuscript.belongsTo(db.book, {
  foreignKey: "bookId"
});
db.book.hasOne(db.mindmap);
db.mindmap.belongsTo(db.book, {
  foreignKey: "bookId"
});
db.book.hasMany(db.note);
db.note.belongsTo(db.book, {
  foreignKey: "bookId"
});
db.book.hasMany(db.character);
db.character.belongsTo(db.book, {
  foreignKey: "bookId"
});
module.exports = db;
```

В процессе инициализации Sequelize сначала импортируется конфигурация базы данных, потом создается объект `sequelize` с настройками подключения к БД. Затем создается объект `db`, куда затем добавляются инициализированные модели для приложения. Между моделями настраиваются ассоциации, которые установят отношения между таблицами в БД. Ассоциация `hasOne` определяет, что книга имеет одну карту мыслей и одну рукопись. Ассоциация `hasMany` определяет, что книга имеет несколько заметок и персонажей. Последним действием объект `db` экспортируется для дальнейшего использования в других частях приложения.

Определение моделей

Для того, чтобы в БД создались нужные таблицы, необходимо каждую сущность определить как модель Sequelize. Обычно в моделях определяют связи между таблицами, но в данном приложении в этом нет необходимости, так как все связи уже были определены ранее в файле «index.js». Рассмотрим определение модели на примере модели персонажа, остальные модели создаются аналогичным способом. Код определения модели персонажа представлено в листинге 5.

Листинг 5 – Определение модели персонажа

```
module.exports = (sequelize, Sequelize) => {
    const Character = sequelize.define("character", {
      firstName: {
        type: Sequelize.STRING,
        allowNull: false
      },
      lastName: {
        type: Sequelize.STRING,
        allowNull: false
      },
      age: {
        type: Sequelize.INTEGER
      },
      additionalData: {
        type: Sequelize.JSONB
      }
    });
    return Character;
  };
```

В модели персонажа мы определяем имя и фамилию персонажа как поля типа `STRING`. Возраст персонажа записывается в поле типа `INTEGER`, а для хранения дополнительных полей, создаваемых пользователем, используется поле `additionalData` типа `JSONB`. При этом у полей имени и фамилии флаг `allowNull` указан как ложь, это означает, что невозможно сохранить в базу персонажа без имени и фамилии.

Создание контроллеров и маршрутов

Контроллеры являются основными компонентами, отвечающими за обработку запросов, присылаемых с клиентской части, взаимодействие с моделями для чтения, редактирования, записи или удаления данных в БД,

формирование и возврат соответствующих ответов клиенту. Рассмотрим работу контроллера на примере метода добавления нового персонажа, который использует стандартные методы Sequelize. Код метода для работы с персонажами представлен в листинге 6

Листинг 6 – Метод добавления нового персонажа

```
exports.create = (req, res) => {
  if (!req.body.firstName || !req.body.lastName) {
    res.status(400).send({ message: "Имя и фамилия персонажа не могут быть
пустыми!" });
    return;
  }
  const character = {
    firstName: req.body.firstName,
    lastName: req.body.lastName,
    age: req.body.age,
    additionalData: req.body.additionalData || {},
    bookId: req.params.bookId
  };
  Character.create(character)
    .then(data => {
      res.send(data);
    })
    .catch(err => {
      res.status(500).send({
        message: err.message || "Произошла ошибка при создании персонажа."
      });
    });
};
```

Данный метод проверяет, все ли поля в запросе не являются пустыми. Если в запросе есть пустые поля, метод возвращает ошибку. Затем создается объект character с данными, полученными из тела запроса. Поле `additionalData` может быть пустым так как не является обязательным для заполнения. Далее метод Character.create пытается сохранить данные о персонаже в базу. Этот метод возвращает промис [17] – объект, представляющий результат асинхронной операции, который может быть получен в будущем. Если промис разрешается, это значит, что данные сохранены успешно, тогда метод отправляет клиенту созданный объект в ответе, иначе полученная ошибка обрабатывается и информация о ней отправляется клиенту.

На рисунке 16 представлена последовательность обращений компонентов серверного приложения при обработке запроса с клиентского приложения.

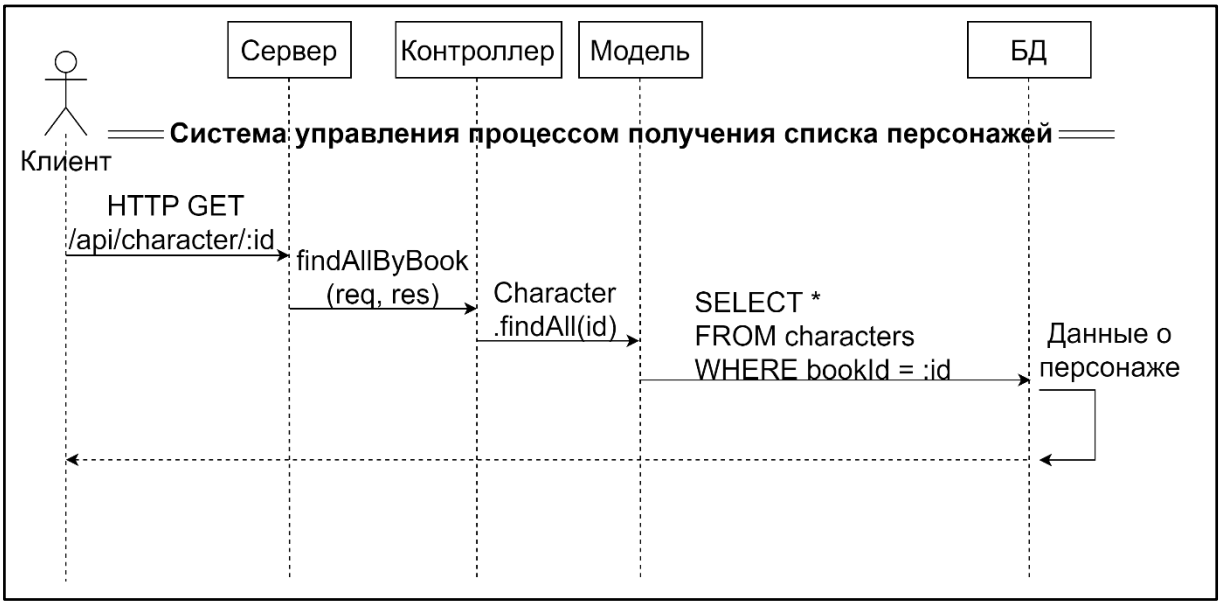

Рисунок 16 – Алгоритм получения списка персонажей

Для того, чтобы сервер понимал, как и какой метод контроллера должен обработать тот или иной запрос необходимо прописать маршруты. Они связывают HTTP-запросы с соответствующими методами контроллера. Для каждой модели маршруты прописываются в отдельном файле. Рассмотрим для примера маршруты для персонажей. Код маршрутов для персонажей определен в листинге 7.

Листинг 7 – Файл маршрутов для персонажей

```
const express = require("express");
const router = express.Router();
const character = require("../controllers/character.controller.js");
router.get("/:bookId", character.findAllByBook);
router.post("/:bookId", character.create);
router.put("/:id", character.update);
router.delete("/:id", character.delete);
router.post("/:id/fields", character.createCustomField);
module.exports = router;
```

В этом файле прописано, что, если от клиента придет запрос «GET http://localhost:8080/api/character/3», сервер вызовет из контроллера персонажей метод `findAllByBook` и передаст ему идентификатор книги для поиска всех персонажей, у которых `bookId` равен идентификатору, пришедшему в запросе.

3.4. Реализация клиентской части приложения

На основе спроектированных макетов реализована клиентская часть приложения, которая состоит из компонентов Vue.js [18]. Каждая страница приложения представлена в виде отдельного компонента. Каждый файл компонента состоит из трех основных частей: шаблона, скрипта и стиля. Шаблон содержит в себе HTML-разметку компонента. Скрипт – описание логики работы компонента на JavaScript. Стили включают в себя CSS-стили, используемые для улучшения внешнего вида компонента.

Основным компонентом клиентской части приложения является компонент «App.vue». Верстка компонента «App.vue» представлена в листинге 8

Листинг 8 – Верстка компонента «App.vue»

```
<template>
  <div id="app">
    <div id="sidebar">
      <router-link to="/">
        <button @click="resetSelectedBook">Home</button>
      </router-link>
      <router-link to="/notes-page">
        <button :disabled="!selectedBook">Notes</button>
      </router-link>
      <router-link to="/mind-map-page">
        <button :disabled="!selectedBook">Mind Map</button>
      </router-link>
      <router-link to="/characters-page">
        <button :disabled="!selectedBook">Characters</button>
      </router-link>
      <router-link to="/manuscript-page">
        <button :disabled="!selectedBook">Manuscript</button>
      </router-link>
      <div v-if="selectedBook" class="book-id-display">
        Book ID: {{ selectedBook.id }}
      </div>
    </div>
    <div id="content">
      <router-view @book-selected="setSelectedBook"></router-view>
    </div>
  </div>
</template>
```

Этот компонент содержит панель навигации, кнопки которой управляют состоянием компонента `router-view`. Пока поле `selectedBook` пусто, остальные кнопки навигации, кроме главной страницы, недоступны.

Поле selectedBook заполняется, когда на главной странице пользователь выбирает книгу из списка. Часть верстки главной страницы, которая отображает список книг для выбора представлена в листинге 9.

Листинг 9 – Верстка списка книг для главной страницы

```
    <div class="book-list">
      <div class="list-container">
        <h2>Список книг</h2>
        <ul>
          <li v-for="book in books" :key="book.id"
@click="selectBook(book)">
            {{ book.title }}
          </li>
        </ul>
        <div v-if="books.length === 0">
          <p>Вы еще не создали ни одной книги</p>
        </div>
      </div>
    </div>
```

Обработчик события передает книгу, на которую было произведено нажатие в метод selectBook, который в свою очередь записывает выбранную книгу в локальное хранилище, для дальнейшего взаимодействия с ней на других страницах. Код метода selectBook представлен в листинге 10.

Листинг 10 – Метод selectBook

```
selectBook(book) {
      if (!this.isEditing && !this.showCreateBookModal) {
        this.selectedBook = book;
        this.$emit('book-selected', book);
        localStorage.setItem('selectedBook', JSON.stringify(book));
      }
    },
```

Этот метод сохраняет выбранную книгу в локальное хранилище, чтобы в дальнейшем использовать ее для подгрузки данных на другие страницы, потому что логика приложения подразумевает, что на всех остальных страницах содержатся элементы только выбранной книги, с которой в данный момент работает пользователь.

На рисунке 17 представлен скриншот главной страницы, когда поле selectedBook заполнено.

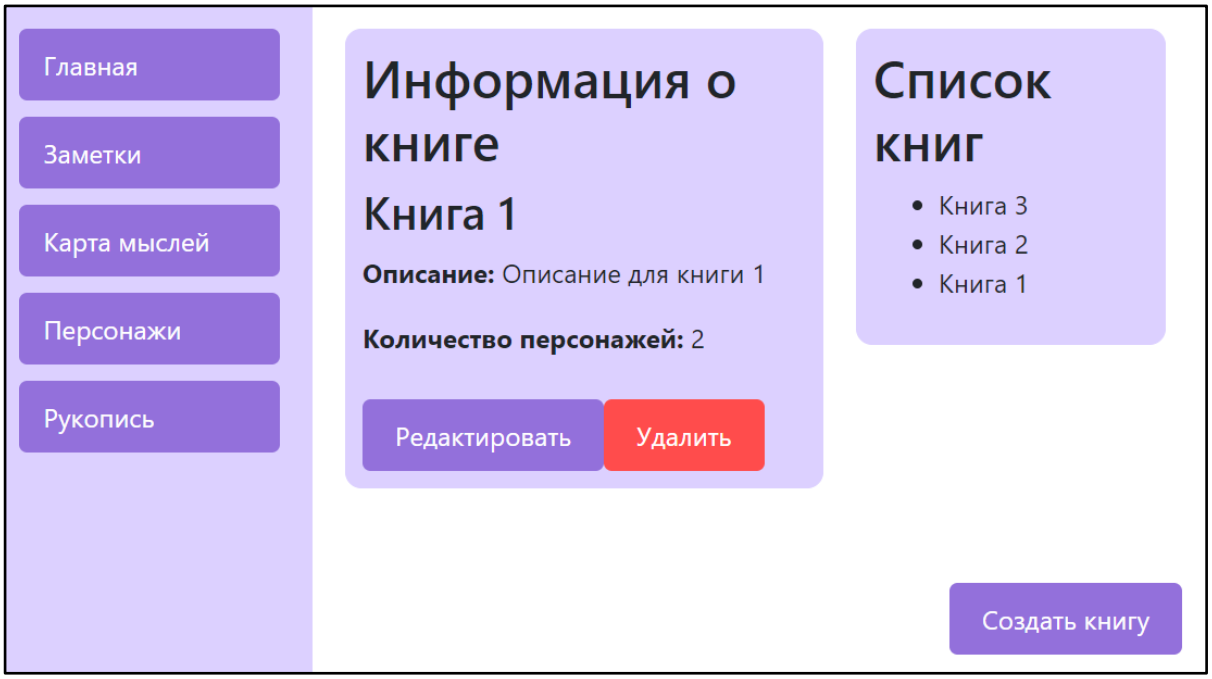

Рисунок 17 – Главная страница приложения

Рассмотрим реализацию других компонентов на примере компонента страницы с персонажами. Он отображает список персонажей, созданных для выбранной книги. Код метода страницы персонажей, вызываемого при ее отображении представлен в листинге 11.

Листинг 11 – Метод `loadCharacters` страницы персонажей

```
async loadCharacters() {
      try {
        const response = await axios.get(`http://localhost:8080/api/char-
acter/${this.selectedBook.id}`);
        this.characters = response.data;
      } catch (error) {
        console.error('Error fetching characters:', error);
        this.errorMessage = 'Ошибка при загрузке персонажей';
      }
    }
```

При создании страницы вызывается метод загрузки персонажей, который с помощью библиотеки Axios отправляет на сервер GET-запрос с идентификатором выбранной книги и обрабатывает полученный ответ для отображения результатов. На рисунке 18 показана работа всей системы при получении списка персонажей по идентификатору книги.

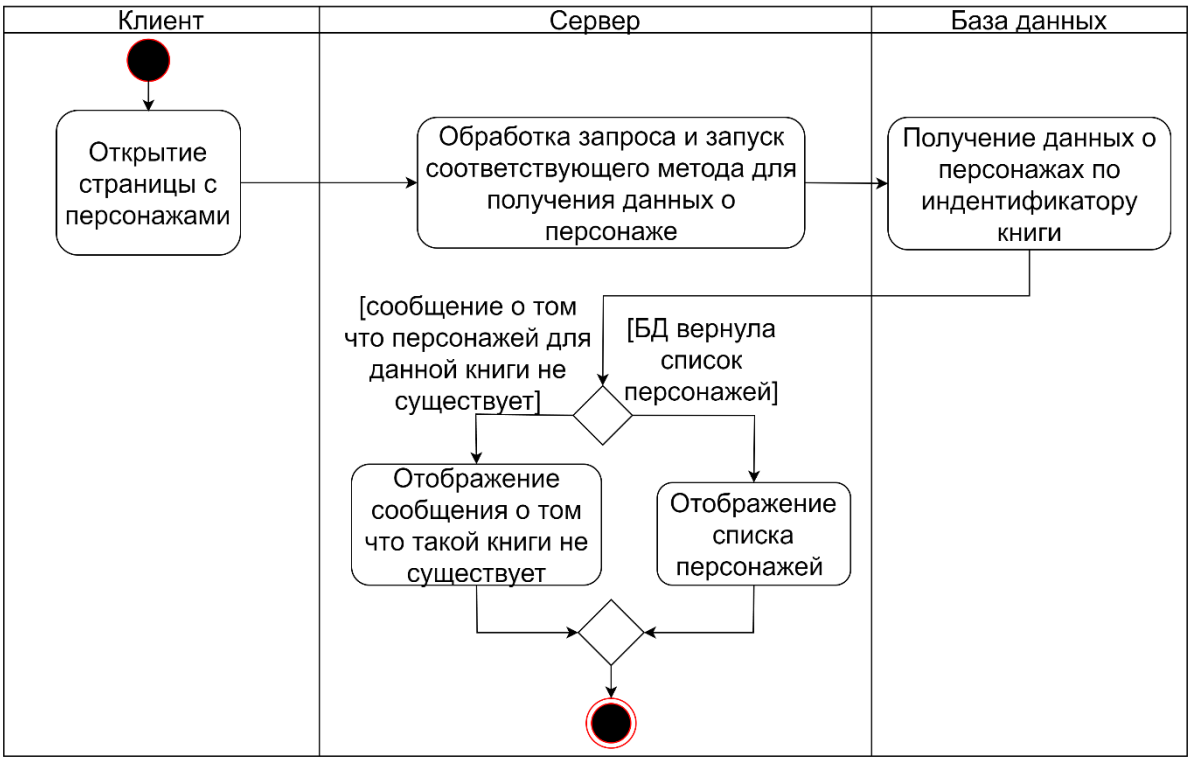

Рисунок 18 – Алгоритм получения списка персонажей

Процесс начинается при отправке с клиентского приложения GET-запроса с идентификатором книги после того, как пользователь нажимает на кнопку, открывающую страницу с персонажами. Сервер вызывает соответствующий маршруту метод в контроллере, который получает данные из БД. В зависимости от того, что вернула БД, данные или сообщение об их отсутствии, полученная информация обрабатывается сервером и отправляется клиенту для отображения.

Скриншот страницы с персонажами представлен на рисунке 19.

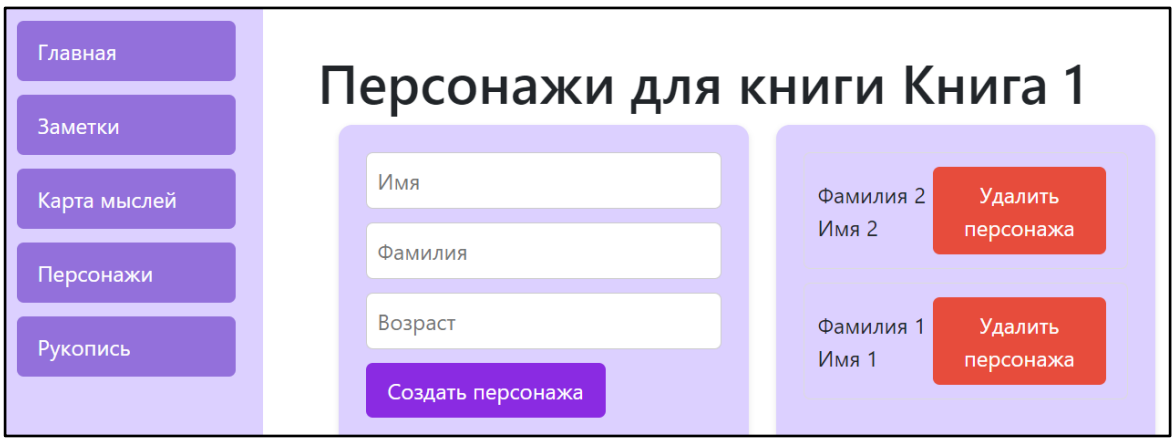

Рисунок 19 – Страница с персонажами

Как и на главной странице, на странице с персонажами есть элементы, которые отображаются не сразу. В данном случае это модальные окна [19]. На странице с персонажами модальное окно карточки персонажа отображается после того, как пользователь нажал на имя этого персонажа в списке.

Верстка и метод, используемые для отображения карточки персонажа и перевода карточки в режим редактирования представлена на листинге 12.

Листинг 12 – Управление состоянием карточки персонажа

```
<div class="character-list">
        <ul>
          <li v-for="character in characters" :key="character.id"
@click="viewCharacter(character)">
            {{ character.lastName }} {{ character.firstName }}
            <button @click.stop="deleteCharacter(character.id)"
class="button red">Удалить персонажа</button>
          </li>
        </ul>
      </div>
    </div>
 viewCharacter(character) {
       this.selectedCharacter = character;
       this.showCharacterCard = true;
       this.showEditCharacter = false;
    },
```

Клик по имени персонажа в компоненте списка вызывает метод `viewCharacter`, который переключает режим видимости модального окна с карточкой персонажа. Внешний вид модального окна представлен на рисунке 20.

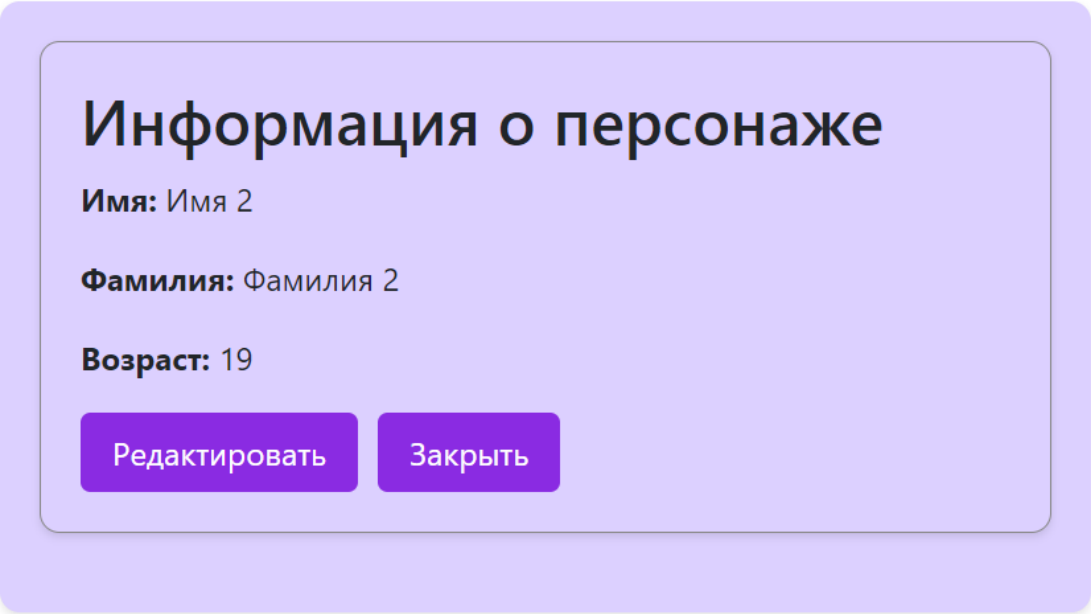

Рисунок 20 – Карточка персонажа

При нажатии на кнопку редактировать открывается окно редактирования информации о персонаже, представленное на рисунке 21.

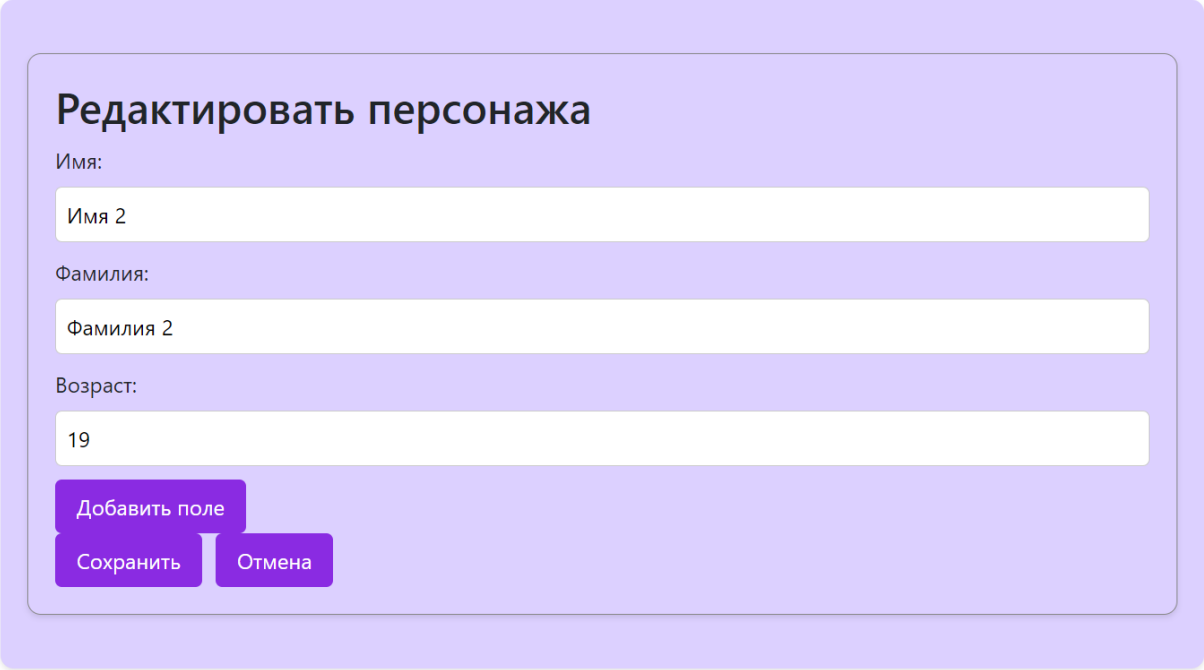

Рисунок 21 – Окно редактирования информации о персонаже

Во время работы над книгой, помимо работы с персонажами, пользователь может также работать с заметками, картами мыслей и рукописью.

Скриншот страницы с заметками представлен на рисунке 22. Для добавления заметки необходимо написать текст в поле и нажать кнопку «Создать заметку»

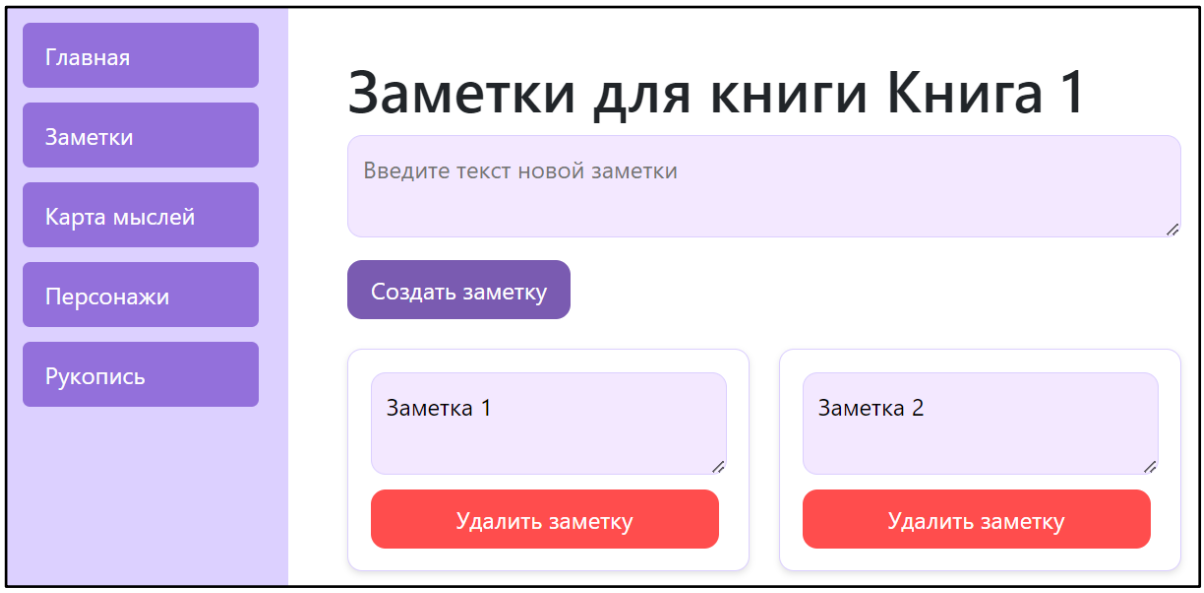

Рисунок 22 – Страница с заметками

Редактирование заметок производится без открытия отдельных модальных окон: при нажатии на текст сохраненной заметки, он автоматически становится редактируемым. Как только пользователь перестает взаимодействовать с заметкой и кликает по другой заметке, либо иной части экрана, система обновляет содержимое этой заметки.

Реализация подобного взаимодействия представлена в листинге 13.

Листинг 13 – Редактирование заметки

```
<div class="notes-grid" v-if="notes.length > 0">
      <div v-for="note in notes" :key="note.id" class="note-card">
        <textarea v-model="note.content" @change="editNote(note.id)" @in-
put="markAsEdited(note.id)"></textarea>
        <button @click="deleteNote(note.id)" class="button red">Удалить 
заметку</button>
      </div>
    </div>
markAsEdited(noteId) {
      this.editedNotes.add(noteId);
    },
    async editNote(noteId) {
      if (this.editedNotes.has(noteId)) {
        try {
          const note = this.notes.find(n => n.id === noteId);
          await axios.put(`http://localhost:8080/api/notes/${noteId}`, {
            content: note.content
          });
          this.editedNotes.delete(noteId);
          this.showNotification('Заметка обновлена', 'success');
        } catch (error) {
          console.error('Error updating note:', error);
          this.showNotification('Ошибка при обновлении заметки', 'error');
        }
      }
    },
```

Когда пользователь изменяет текст в текстовом поле с содержимым заметки, вызывается событие `@input`, которое передает идентификатор заметки в метод `markAsEdited`. Этот метод в свою очередь помечает заметку измененной, добавляя ее идентификатор в множество редактируемых заметок.

Когда фокус с текстового поля смещается на другие элементы, система распознает это как завершение изменения, вызывает событие `@change`, которое запускает метод `editNote`. Этот метод проверяет, находится ли идентификатор заметки в множестве редактируемых заметок. Если

находится, метод находит заметку в общем списке заметок по идентификатору и затем отправляет PUT-запрос на сервер, чтобы обновить содержимое заметки в БД. Если запрос выполнился успешно, идентификатор заметки удаляется из множества редактируемых заметок и отображается уведомление об успешном редактировании, иначе отображается уведомление об ошибке.

На рисунке 23 изображен скриншот страницы с картой мыслей для выбранной книги.

Рисунок 23 – Страница карты мыслей

Эта страница позволяет пользователю взаимодействовать с редактором диаграмм для создания карты мыслей. При открытии страницы клиент отправляет на сервер запрос, чтобы узнать, существует ли карта мыслей для выбранной книги. Если карта мыслей существует, ее данные передаются из базы данных и компонент, получив информацию о расположении, размерах и содержимом узлов, отрисовывает их на том же месте.

Кнопка «Создать узел» отрисовывает новый узел по центру холста с параметрами по умолчанию. Кнопка «Создать связь» требует выбора двух

узлов, между которыми будет создана связь. Кнопки «Сохранить» и «Удалить» с помочью HTTP-запросов обращаются к серверу для внесения соответствующих изменений.

Последняя функция, которую пользователь может использовать для работы над книгой – создание манускрипта. Эта страница использует библиотеку CKEditor для отрисовки поля текстового редактора с возможностью форматирования текста. На рисунке 24 изображен скриншот страницы с рукописью.

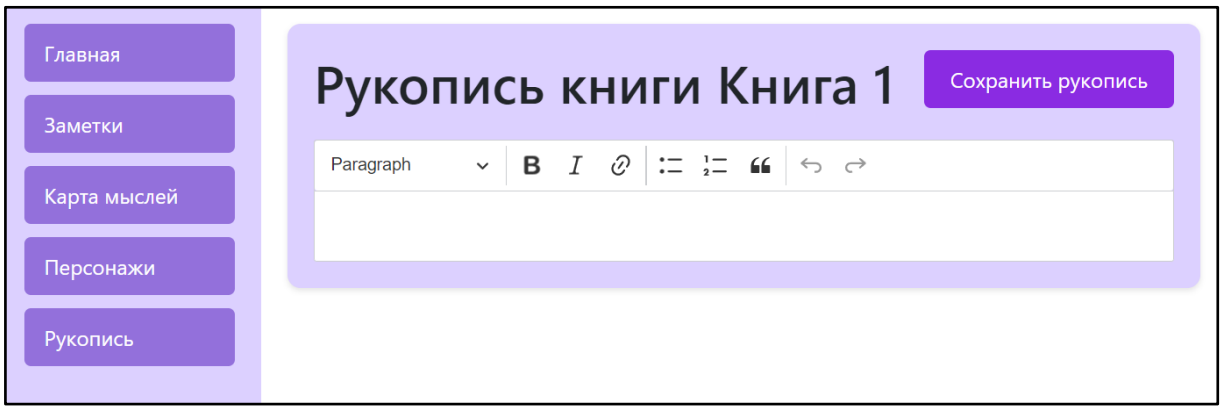

Рисунок 24 – Страница с рукописью

На данной странице пользователь может свободно писать и форматировать текст.

Выводы по третьей главе

В данной главе была описана реализация веб-приложения: средства реализации, реализация архитектуры веб-приложения, реализация серверной части приложения, а также реализация клиентской части приложения.

4. ТЕСТИРОВАНИЕ

Для тестирования веб-приложения было выбрано функциональное тестирование. Этот способ позволяет проверить реализуемость функциональных требований. Тестирование проводилось вручную.

В таблице 1 представлен протокол тестирования функционала веб-приложения.

Таблица 1 – Функциональное тестирование веб-приложения

№	Название теста	Действие	Ожидаемый результат	Тест пройден?
1	Проверка работы функции создания книг	Открыть веб-приложение, нажать кнопку создания книги, корректно заполнить все поля в модальном окне, нажать кнопку создания книги	Созданная книга отображается в списке книг	Да
2	Проверка работы функции просмотра и выбора книги	Открыть веб-приложение, в котором уже были созданы книги, просмотреть список появившихся книг, выбрать любую книгу из списка	При открытии приложения список книг загружается при условии, что книги уже были созданы, при выборе книги, ее идентификатор отображается на панели навигации и кнопки навигации становятся доступны	Да
3	Проверка работы функции редактирования книги	Открыть веб-приложение, выбрать книгу из списка, нажать на кнопку редактирования книги, внести изменения, нажать на кнопку сохранения	При последующем просмотре этой книги отображаются внесенные изменения	Да
4	Проверка работы функции добавления заметки	Открыть веб-приложение, выбрать книгу, открыть страницу с заметками, ввести текст в поле добавления заметки, нажать кнопку добавления заметки	Добавленная заметка отображается в общем списке заметок	Да

Продолжение таблицы 1

№	Название теста	Действие	Ожидаемый результат	Тест пройден?
5	Проверка работы функции редактирования заметки	Открыть веб-приложение, выбрать книгу, открыть страницу с заметками, нажать на текст уже существующей заметки, изменить текст, нажать куда угодно рядом с редактируемой заметкой	Изменения в тексте заметки сохранились	Да
6	Проверка работы функции удаления заметки	Открыть веб-приложение, выбрать книгу, открыть страницу с заметками, выбрать заметку, нажать на кнопку удаления заметки	Выбранная заметка исчезает из общего списка заметок	Да
7	Проверка работы функции создания карты мыслей	Открыть веб-приложение, выбрать книгу, открыть страницу с картой мыслей, создать произвольное количество узлов и связей между ними, нажать на кнопку сохранения, обновить страницу	Только что созданная карта после обновления страницы отображается такой какой ее только что создали	Да
8	Проверка работы функции удаления узлов	Открыть веб-приложение, выбрать книгу, открыть страницу с картой мыслей, выбрать произвольные узлы, удалить их	Выбранные узлы после нажатия на кнопку удаления исчезают	Да
9	Проверка работы функции удаления связей	Открыть веб-приложение, выбрать книгу, открыть страницу с картой мыслей, выбрать произвольные связи, нажат на них чтобы выделить, нажать на кнопку удаления	Выбранные связи после нажатия на кнопку удаления исчезают	Да

Продолжение таблицы 1

№	Название теста	Действие	Ожидаемый результат	Тест пройден?
10	Проверка работы функции перемещения узлов	Открыть веб-приложение, выбрать книгу, открыть страницу с картой мыслей, создать произвольную карту мыслей, выбрать узел, навести на него курсор, зажать левую кнопку мыши, переместить узел, отпустить левую кнопку мыши	Выбранный узел перемещается по холсту редактора карт мыслей вслед за курсором и после перемещения сохраняет свое местоположение	Да
11	Проверка работы функции редактирования параметров узла	Открыть веб-приложение, выбрать книгу, открыть страницу с картой мыслей, выбрать произвольный узел, изменить его размеры и содержимое	Выбранный узел меняет свой внешний вид в соответствии с внесенными изменениями	Да
12	Проверка работы функции удаления карты мыслей	Открыть веб-приложение, выбрать книгу, открыть страницу с картой мыслей, при условии, что карта мыслей уже была создана, нажать на кнопку удаления карты мыслей, обновить страницу	После обновления страницы окно редактора диаграмм остается пустым и высвечивается сообщение о том, что карты мыслей для данной книги не существует	Да
13	Проверка работы функции добавления персонажей	Открыть веб-приложение, выбрать книгу, открыть страницу с персонажами, заполнить поля добавления персонажа, нажать кнопку добавления	Персонаж с указанными характеристиками появляется в списке персонажей. При нажатии на него открывается его карточка с введенными данными	Да

Окончание таблицы 1

№	Название теста	Действие	Ожидаемый результат	Тест пройден?
14	Проверка работы функции редактирования персонажей	Открыть веб-приложение, выбрать книгу, открыть страницу с персонажами, выбрать персонажа, открыть его карточку, нажать на кнопку редактирования, изменить данные, сохранить изменения	Внесенные изменения отображаются при последующем открытии карточки персонажа	Да
15	Проверка работы функции добавления пользовательских полей у персонажа	Открыть веб-приложение, выбрать книгу, открыть страницу с персонажами, выбрать персонажа, открыть его карточку в режиме редактирования, нажать на кнопку добавления нового поля, ввести его название, сохранить	Новое поле теперь отображается в карточке персонажа и доступно для заполнения	Да
16	Проверка работы функции удаления персонажа	Открыть веб-приложение, выбрать книгу, открыть страницу с персонажами, нажать на кнопку удаления напротив любого персонажа	Персонаж больше не существует, он пропадает из списка	Да
17	Проверка работы функции редактирования рукописи	Открыть веб-приложение, выбрать книгу, открыть страницу с рукописью, внести изменения в рукопись, нажать на кнопку сохранения, обновить страницу	После обновления страницы внесенные в рукопись изменения сохраняются	Да

Выводы по четвертой главе

В четвертой главе было проведено функциональное тестирование веб-приложения. Все тесты из списка были пройдены успешно, из чего следует, что приложение соответствует функциональными требованиям и работает правильно.

ЗАКЛЮЧЕНИЕ

В рамках выпускной квалификационной работы было реализовано веб-приложение «Помощник автора».

Были достигнуты результаты, представленные ниже:

1. Произведен обзор предметной области и существующих решений.
2. Спроектирована архитектура веб-приложения.
3. Реализовано веб-приложение для проработки идей и сюжета книги.
4. Проведено тестирование веб-приложения.

В ходе выполнения работы был получен опыт работы с языком программирования JavaScript, платформой Node.js, фреймворками Vue.js и Express, также были получены навыки работы с СУБД PosgreSQL и библиотекой Axios.

Разработанное веб-приложение дает возможность фиксировать, структурировать и прорабатывать идеи, сюжет и персонажей для книги в виде заметок, карты мыслей, карточек персонажей и рукописи и хранить это все в едином пространстве.

Направления дальнейших исследований

Дальнейшие исследования и практические разработки будут направлены на расширение функционала приложения, а именно: добавление сущности «Серия» для возможности создавать связанные между собой книги, создание личного кабинета автора, расширение функционала карты мыслей, добавление возможности создания ссылок одних элементов книги на другие, добавление деления по главам в рукописи. Также планируется создание мобильной версии приложения и добавление возможности экспротировать содержимое книги в форматы DOCX и TXT.

ЛИТЕРАТУРА

1. Структура литературного произведения. [Электронный ресурс] URL: https://www.booksite.ru/fulltext/1/001/008/106/918.htm (дата обращения: 08.02.2024 г.).

2. Семь лучших текстовых редакторов в 2024 году. [Электронный ресурс] URL: https://kokoc.com/blog/luchshie-tekstovye-redaktory/ (дата обращения: 10.02.2024 г.).

3. Evernote. [Электронный ресурс] URL: https://evernote.com/ (дата обращения: 10.02.2024 г.).

4. Что такое карта мыслей и как с ней работать. [Электронный ресурс] URL: https://lifehacker.ru/chto-takoe-karta-myslej-i-kak-s-nej-rabotat/ (дата обращения: 11.02.2024 г.).

5. Mind Meister. [Электронный ресурс] URL: https://www.mindmeister.com/app/folders (дата обращения: 11.02.2024 г.).

6. Novel Factory – The Ultimate Novel Writing Software. [Электронный ресурс] URL: https://www.novel-software.com/ (дата обращения: 08.02.2024 г.).

7. Plottr – Plan Your Books Like a Pro. [Электронный ресурс] URL: https://plottr.com/ (дата обращения: 01.02.2024 г.).

8. Best novel writer software | bibisco. [Электронный ресурс] URL: https://bibisco.com/ (дата обращения: 01.02.2024 г.).

9. Клиент-серверная архитектура. [Электронный ресурс] URL: https://servergate.ru/articles/klient-servernaya-arkhitektura/ (дата обращения: 09.03.2024 г.).

10. Vue.js. [Электронный ресурс] URL: https://blog.skillfactory.ru/glossary/vue-js/ (дата обращения: 07.03.2024 г.).

11. Node.js. [Электронный ресурс] URL: https://nodejs.org/en/ (дата обращения: 01.03.2024 г.).

12. Начало работы с Express. [Электронный ресурс] URL: https://metanit.com/web/nodejs/4.1.php (дата обращения: 28.03.2024 г.).

13. PostgreSQL: Documentation. [Электронный ресурс] URL: https://www.postgresql.org/docs/ (дата обращения: 24.03.2024 г.).

14. Sequelize. [Электронный ресурс] URL: https://habr.com/ru/articles/565062/ (дата обращения: 24.03.2024 г.).

15. Axios Docs. [Электронный ресурс] URL: https://axios-http.com/ru/docs/intro (дата обращения: 31.03.2024 г.).

16. Что такое cors. [Электронный ресурс] URL: https://doka.guide/tools/cors/ (дата обращения: 16.03.2024 г.).

17. Promise. [Электронный ресурс] URL: https://learn.javascript.ru/promise (дата обращения: 16.03.2024 г.).

18. Основы компонентов. [Электронный ресурс] URL: https://v3.ru.vuejs.org/ru/guide/component-basics.html (дата обращения: 07.03.2024 г.).

19. Модальное окно на Vue js. [Электронный ресурс] URL: https://webdevnet.ru/modalnoe-okno-vue-js/ (дата обращения: 28.03.2024 г.).