Министерство образования Республики Беларусь
Учреждение образования «Белорусский государственный университет информатики и радиоэлектроники»

Факультет компьютерных систем и сетей
Кафедра электронных вычислительных машин
Дисциплина: Программирование на языках высокого уровня

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовому проекту

на тему

Система контроля процесса чтения

БГУИР КП 1–40 02 01 208 ПЗ

Студент: группы 250502,
Грибовская А. А.

Руководитель: ассистент каф. ЭВМ
Богдан Е. В.

Минск 2023

Учреждение образования
«Белорусский Учреждение образования
«Белорусский государственный университет информатики
и радиоэлектроники»

Факультет компьютерных систем и сетей

УТВЕРЖДАЮ
Заведующий кафедрой

_____
(подпись)
_____2023 г.

ЗАДАНИЕ
по курсовому проектированию

Студенту _____*Грибовской Александре Александровне*_____

Тема проекта_____*Система контроля процесса чтения*_____

2. Срок сдачи студентом законченного проекта____*15 декабря 2023 г.*

3. Исходные данные к проекту *Язык программирования – C++, среда разработки – Qt-Creator*

4. Содержание расчетно-пояснительной записки (перечень вопросов, которые подлежат разработке)

1. Лист задания.
2. Введение.
3. Обзор литературы.
   3.1. Обзор методов и алгоритмов решения поставленной задачи.
4. Функциональное проектирование.
   4.1. Структура входных и выходных данных.
   4.2. Разработка диаграммы классов.
   4.3. Описание классов.

5. Разработка программных модулей.

5.1. Разработка схем алгоритмов (два наиболее важных метода).

5.2. Разработка алгоритмов (описание алгоритмов по шагам, для двух методов).

6. Результаты работы.

7. Заключение

8. Литература

9. Приложения

5. Перечень графического материала (с точным обозначением обязательных чертежей и графиков)

1. Диаграмма классов.

2. Схема алгоритма конвертора из FB2 в HTML.

3. Схема алгоритма отображения списка книг на экране

4. Схема структурная

6. Консультант по проекту (с обозначением разделов проекта) Е. В. Богдан

7. Дата выдачи задания *15.09.2023г.*

8. Календарный график работы над проектом на весь период проектирования (с обозначением сроков выполнения и трудоемкости отдельных этапов):

1. Выбор задания. Разработка содержания пояснительной записки. Перечень графического материала – 15 %;

разделы 2, 3 – 10 %;

разделы 4 к –20 %;

разделы 5 к – 35 %;

раздел 6,7,8 – 5 %;

раздел 9 к – 5%;

оформление пояснительной записки и графического материала к 15.12.22 – 10 %

Защита курсового проекта с 21.12 по 28.12.23г.

РУКОВОДИТЕЛЬ Е.В. Богдан

(подпись)

Задание принял к исполнению_____

(дата и подпись студента)

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Система контроля процесса чтения – это приложение на языке C++, которое предназначено для чтения электронных книг формата FB2.

Приложение для чтения книг – это инновационный способ получить доступ к огромному количеству литературных произведений в удобном и доступном формате. Оно позволяет читать в любом месте и в любое время, достаточно иметь доступ к компьютеру. Цифровые книги не занимают место на полках или в сумках. Они сохраняются в электронном виде, что позволяет иметь доступ к огромной коллекции, не перегружая свое пространство. Такие приложения предлагают различные настройки для улучшения комфорта чтения, например, изменение размера шрифта и использование различных режимов чтения (ночной режим, режим защиты глаз и др.). Чтение электронных книг помогает экономить бумагу и другие ресурсы, что способствует уменьшению экологического следа.

Приложения для чтения книг дают свободу выбора и комфортное чтение, открывая мир знаний и развлечений в удобной и доступной форме.

Язык программирования C++ с Qt представляют собой мощную комбинацию инструментов для разработки программного обеспечения, обладающую высокой производительностью и гибкостью. C++ является языком программирования общего назначения, известным своей эффективностью и возможностью создания высокопроизводительных приложений. Qt, в свою очередь, предоставляет богатые возможности для разработки графического интерфейса и работы с различными аспектами приложений.

Qt также предлагает обширный набор библиотек и инструментов, включая графические элементы управления, инструменты работы с сетью, базами данных и мультимедиа.

Qt Creator, интегрированная среда разработки, предоставляет удобный набор инструментов для написания, отладки и развертывания приложений, что упрощает процесс работы разработчиков. Благодаря активным сообществам пользователей и обширной документации, разработчики получают поддержку, информацию и ресурсы для изучения и использования C++ с Qt в своих проектах.

SQLite3 – это компактная и легкая в использовании реляционная база данных, которая не требует отдельного сервера для работы. Она реализует SQL–стандарт и обладает высокой производительностью, позволяя хранить данные в одном файле базы данных. SQLite3 поддерживает множество операций с данными, включая создание таблиц, запросы, обновление и удаление записей. Эта база данных широко используется в мобильных приложениях, десктопных приложениях, а также в различных веб–приложениях благодаря своей простоте в интеграции и гибкости в использовании.

1 ПОСТАНОВКА ЗАДАЧИ

Программный модуль «Система контроля процесса чтения» представляет собой инструмент с удобным пользовательским интерфейсом для эффективного взаимодействия с содержимым электронных книг. Он заботится о хранении информации о каждой книге в базе данных, предоставляя возможность добавлять и удалять книги из базы данных. Также реализована функция поиска книг по их названию внутри приложения. Кроме того, модуль позволяет создавать тематические полки, в которых можно группировать книги по определенным тематикам или категориям.

Для реализации этой системы используется иерархия классов с применением наследования. Классы-контейнеры используются для эффективной организации данных и управления ими. При разработке уделяется внимание обработке исключительных ситуаций, что обеспечивает более стабильную и безопасную работу приложения.

Важной особенностью данной системы являются различные режимы чтения, предоставляющие пользователям возможность выбирать настройки, соответствующие их предпочтениям и комфорту чтения.

Этот модуль предоставляет универсальный инструмент для работы с книгами, обеспечивая удобное управление содержимым, удобный поиск и доступ к электронным книгам в соответствии с предпочтениями и потребностями пользователей.

Парсинг книг представляет собой процесс анализа и извлечения информации из электронных текстовых документов, чтобы обработать их или использовать для определенных целей. В случае электронных книг, парсинг может включать в себя извлечение текстового содержания, метаданных (например, название, автор, год издания) или даже изображений обложек. Обычно это выполняется с помощью специальных программных инструментов или скриптов, которые анализируют структуру и форматы электронных книг для извлечения нужной информации. После парсинга эта информация может быть обработана, использована для построения баз данных, а также отображена в приложениях для чтения электронных книг или библиотечных приложениях для удобства пользователей.

2 ОБЗОР ЛИТЕРАТУРЫ

2.1 Анализ существующих аналогов

Тема курсового проекта была выбрана в первую очередь для получения знаний в области разработки десктопных приложений с использованием ООП. В процессе создания приложения будут затронуты главные принципы ООП, работа с базой данных, интерфейс QT, использован источник [1]. Было выбрано приложение, связанное с взаимодействием с текстовыми материалами, которое является актуальным в процессе обучения. Для того, чтобы создать корректно работающее приложение, необходимо иметь представление об уже реализованных аналогах и изучить содержание источника [2].

2.1.1 Приложение eBook

eBook – это приложение для чтения форматов fb2, epub, pdf, doc, docx, mobi, prc, txt, rtf, odt, html, cbr, cbz, zip– и rar–архивы. Удобная загрузка книг в ридер из любых папок телефона и SD–карты, из облака и браузеров. Имеет еще ряд преимуществ: оптимальные настройки под себя для комфортного чтения, высокая оптимизация, простая и удобная загрузка книг с карты памяти и интернета, удобный интерфейс, синхронизация файлов для Android–устройств. На рисунке 2.1 показана часть интерфейса данного приложения.

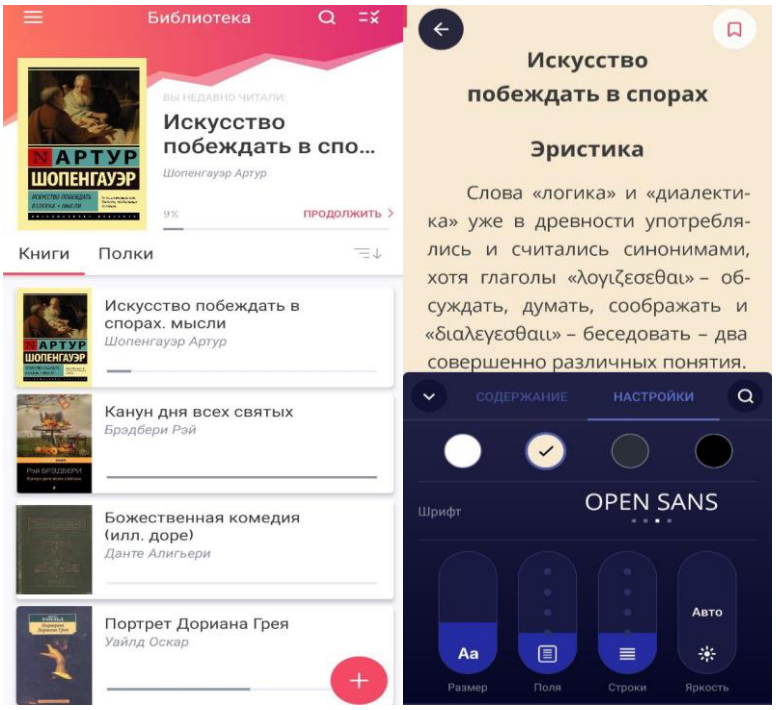

Рисунок 2.1 – Скриншоты eBook

2.1.2 Приложение LitRes

LitRes – наиболее популярный аналог с самым большим рускоязычным каталогом электронных книг, аудиокниг и подкастов. Большинство новинок книжного рынка появляются в каталоге ЛитРес одновременно или даже до выхода бумажной книги. Можно искать книги по категориям, жанрам и оставлять отзывы на них. Также можно менять яркость, размер шрифта и цвет фона, следить за прогрессом: сколько страниц прочитано, сколько осталось, использовать ночной режим для комфортного чтения. Рисунок 2.2 демонстрирует базовый интерфейс приложения LitRes.

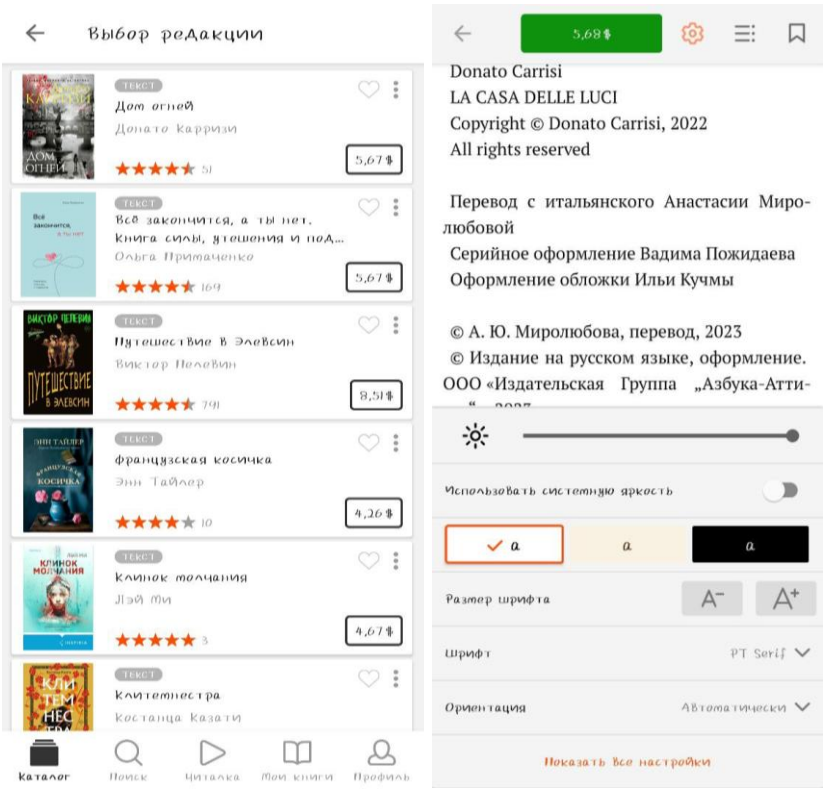

Рисунок 2.2 – Скриншот LitRes

2.1.3 Приложение ReadEra

ReadEra – приложение для чтения книг, позволяет читать книги бесплатно, без интернета в форматах FB2, PDF, EPUB, Microsoft Word (DOC, DOCX, RTF), DJVU, Kindle (MOBI, AZW3), TXT, ODT и CHM. Быстрый доступ к настройкам чтения, оглавлению, закладкам, цитатам, заметкам, выделению текста цветом, истории просмотра страниц и другим параметрам электронной книги. Приложение ReadEra позволяет одновременно читать несколько книг и документов. Например, можно одновременно читать Fb2 книгу и PDF журнал, разместив их на экране устройства в режиме разделенного экрана (двух окон). Автоматическое сохранение текущей

страницы чтения. Комфортные темы при чтении книг: день, ночь, сепия, консоль. Настройка ориентации, яркости экрана и полей страниц, включая PDF и DjVu. Отдельные настройки для текстовых и графических форматов. Рисунок 2.3 демонстрирует интерфейс для работы в данном приложении.

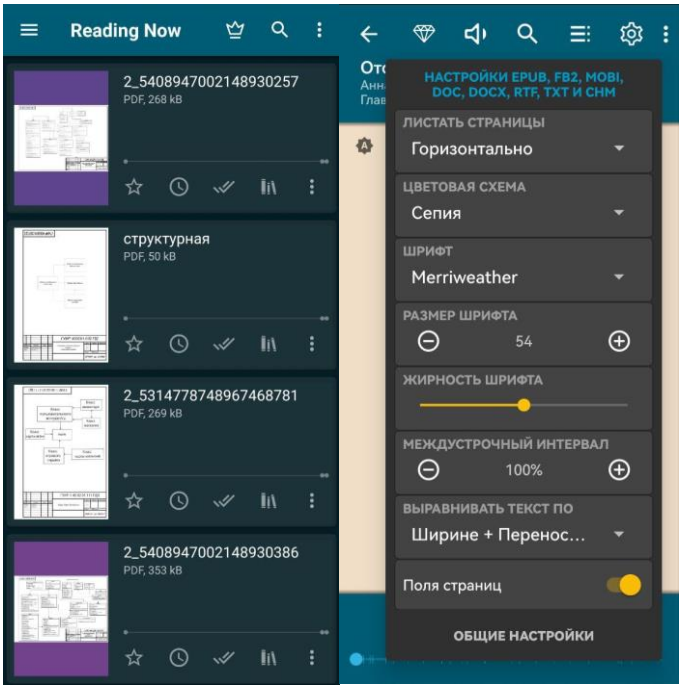

Рисунок 2.3 – Скриншоты ReadEra

2.2 Требования к работе программы

После рассмотрения аналогов данного курсового проекта, становится понятно, что подобного рода приложения, в основном, включают в себя следующие основные функции:

- демонстрация содержимого пользовательской библиотеки;
- обработка определенного формата файла;
- комбинирование файлов в тематические группы;
- корректное отображение содержимого электронной книги;
- разнообразие режимов;
- сохранение добавленного объекта, при выходе из приложения;
- удаление выбранного объекта;
- поиск нужного объекта из списка по определенным параметрам.

Для создания приложения был выбран язык C++, так как он поддерживает использование объектно–ориентированной парадигмы, что делает его удобным для создания больших и сложных систем, разбитых на модули и классы. C++ остается одним из самых популярных и востребованных языков программирования. Полная информация об языке приведена в источниках [3] и [4].

3 СИСТЕМНОЕ ПРОЕКТИРОВАНИЕ

После определения требований к функционалу разрабатываемого приложения его следует разбить на функциональные блоки. Такой подход упростит понимание создания приложения, а также позволит легко оптимизацию.

3.1 Модуль базы данных

Этот модуль отвечает за хранение и управление информацией о книгах и полках внутри приложения. Его задачи включают:

- Хранение информации о книгах: содержит данные о книгах, включая их названия, авторов, обложки, содержимое и другие сведения;
- Сохранение списка полок: хранит информацию о полках, к которым относятся книги;
- Сохранение связей между книгами и полками: сохраняет информацию о том, какие книги принадлежат определенной полке.

3.2 Модуль отображения списка книг

Модуль отображения списка книг является одним из основных элементов приложения, предназначенным для представления и управления информацией о книгах. Его функционал включает:

- Отображение информации о книгах. Каждая книга в списке представлена следующими элементами: обложка, название, автор, кнопка удаления книги из списка;
- Пользователь может осуществлять поиск книг из списка по их названию для быстрого доступа к конкретной книге;
- Кнопка добавления книги позволяет добавлять новые книги в список. При ее нажатии приложение обращается к файлам в памяти компьютера для добавления выбранной книги;
- Кнопка перехода к списку полок предоставляет возможность перехода к списку полок, где пользователь может просмотреть содержимое конкретной полки;
- Реализована возможность перехода к окну, где отображается содержимое определенной книги при нажатии на соответствующее поле. Это позволяет пользователю читать или просматривать содержимое выбранной книги более подробно.

3.3 Модуль отображения текста книги

Модуль отображения текста книги является частью приложения, позволяющей пользователям просматривать текстовое содержимое книги из базы данных и изменять его внешний вид для более удобного чтения.

Пользователи имеют возможность изменять стиль шрифта текста, такой как жирный, курсив, обычный, для настройки внешнего вида текста под свои предпочтения. Модуль предоставляет возможность изменения размера шрифта текста в книге. Пользователи могут настраивать цвет фона текста в книге для создания более комфортного и приятного визуального восприятия.

Модуль отображает как текстовое содержимое книги, так и ее обложку, обеспечивая полноценное чтение и визуальное представление книги в пользовательском интерфейсе.

3.4 Модуль управления полками

Модуль управления полками представляет собой функционал приложения, который отвечает за отображение списка полок, где каждая полка имеет свое название и кнопку удаления книги. Он также содержит строку поиска, позволяющую осуществлять поиск полки по ее названию. В этом модуле реализована возможность добавления новой полки. При нажатии на кнопку добавления полки открывается новое окно, в котором происходит проверка уникальности названия полки перед ее добавлением.

Данный модуль позволяет перейти к определенной полке и отобразить список книг, принадлежащих этой полке. Список книг, содержащийся в каждой полке, обладает тем же функционалом, что и модуль отображения списка книг. То есть в каждом элементе списка книг имеется обложка, название, автор, а также кнопка для удаления конкретной книги.

Структурная схема модулей приложения приведена в приложении А.

4 ФУНКЦИОНАЛЬНОЕ ПРОЕКТИРОВАНИЕ

В данном разделе описывается функционирование и структура разрабатываемого приложения.

Диаграмма классов представлена в Приложении Б.

4.1 Входные и выходные данные

Входными данными в приложении являются файлы формата fb2, которые мы загружаем из памяти компьютера (Рисунок 4.1).

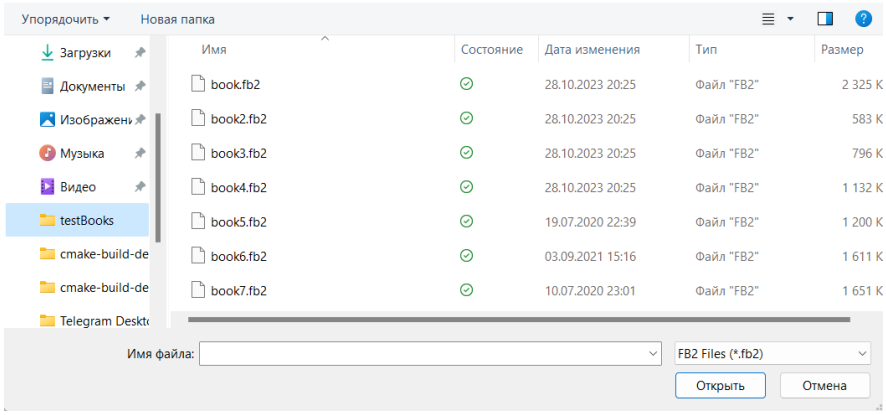

Рисунок 4.1 – Входные файлы формата fb2.

Дальше данные, содержащиеся в файле, записываются в SQLite в таком виде:

Таблица 4.1 – структура информации о книге в базе данных.

id	title	author	image	html
3	Канун для всех святых	Рэй Брэдбери	Формат Base64	Формат string

В конце открывается окно, где на экран выводится вся информация из SQLite (Рисунок 4.2).

Рэй Брэдбери

Канун дня всех святых

Ray Bradbury

THE HALLOWEEN TREE

Перевод с английского Арама Оганяна

Рисунок 4.2 – Выходные данные файла формата fb2.

4.2 Модуль базы данных

4.2.1 Класс BaseEntity

BaseEntity представляет собой базовую сущность (или базовый класс) для объектов, которые могут быть идентифицированы уникальным идентификатором `id`.

Наследуясь от этого базового класса, производные классы могут расширить его функциональность и добавить свои собственные уникальные атрибуты и методы, сохраняя при этом базовую функциональность работы с идентификатором. Поля:

- `int id` – уникальным идентификатором;

Методы:

- `BaseEntity(int id)` – конструктор для создания объекта с заданным идентификатором;
- `BaseEntity()` – конструктор используется для создания объектов без начального идентификатора;
- `int getId() const` – возвращает текущий идентификатор объекта;
- `void setId(int newId);` – устанавливает новый идентификатор объекта.

4.2.2 Класс Database

Этот класс представляет собой реализацию функциональности базы данных для управления книгами (`Book`) и полками (`Shelf`) в приложении.

Методы:

- `database()` – конструктор класса `Database`. Внутри него устанавливается имя базы данных и осуществляется подключение к базе данных SQLite или MySQL (в зависимости от того, какая база данных используется). Если подключение прошло успешно, выводится сообщение об успешном подключении;
- `~database()` – деструктор класса `Database`. Он закрывает соединение с базой данных при уничтожении объекта класса;
- `addBook(Book book)` – метод для добавления новой книги в базу данных. Использует SQL–запрос для вставки данных о книге (название, автор, изображение, содержание) в таблицу `Book`. В случае успешной вставки выводит сообщение об успешном добавлении, в противном случае выводит сообщение об ошибке;
- `getAllBooks()` – метод для получения всех книг из базы данных. Выполняет запрос к базе данных для извлечения всех книг из таблицы `Book` и сохраняет их в векторе книг. Затем возвращается этот вектор;
- `deleteBook(int id)` – метод для удаления книги из базы данных по заданному идентификатору. Происходит удаление книги из таблицы `Book` и связанных с ней полок `(BooksOnShelfes)`. Используется транзакция для обеспечения целостности данных;
- `addShelf(QString shelfName)` – метод для добавления новой полки в базу данных. Вставляет данные о полке (название) в таблицу `Shelf`;
- `getAllShelves()` – метод для получения всех полок из базы данных. Выполняет запрос к базе данных для извлечения всех полок из таблицы `Shelf` и сохраняет их в векторе полок;
- `addBookOnShelf(int shelfId, int bookId)` – метод для добавления книги на определенную полку. Вставляет запись о связи между книгой (`bookId`) и полкой (`shelfId`) в таблицу `BooksOnShelfes`;
- `deleteBookFromShelf(int shelfId, int bookId)` – метод для удаления книги с определенной полки. Удаляет запись о связи между книгой (`bookId`) и полкой (`shelfId`) из таблицы `BooksOnShelfes`;
- `deleteShelf(int shelfId)` – метод для удаления полки из базы данных по заданному идентификатору. Происходит удаление полки из таблицы `Shelf` и всех книг, связанных с этой полкой, из таблицы `BooksOnShelfes`;
- `getBooksFromShelf(int shelfId)` – метод для получения всех книг с определенной полки. Выполняет запрос к базе данных для извлечения всех книг, связанных с заданной полкой (`shelfId`) из таблицы `Book` и возвращает их в виде вектора книг.

4.3 Модуль отображения списка книг

4.3.1 Класс Fb2Helper

Данный класс используется для конвертации файлов в формате FB2 (`FictionBook`) в HTML и для парсинга метаданных книги из XML.

Методы:

- `fb2helper()` – это конструктор класса fb2helper, который на данный момент не содержит какой–либо логики и выполняет только инициализацию объекта;
- `convertFb2ToHTML(QString filePath)` – метод для преобразования файла FB2 в HTML. Он открывает файл, читает его содержимое в XML–поток (`QXmlStreamReader`) и на основе разбора XML формирует HTML–страницу книги. Метод выполняет обработку тегов и содержания файла FB2, формируя HTML–разметку. Также, на основе содержимого файла, извлекается информация о книге (например, заголовок, автор) для последующего создания объекта `Book`;
- `parseFictionBook(QString filePath)` – этот метод также используется для парсинга файлов FB2, но в данном случае, для извлечения метаданных книги, таких как имя автора и название книги. Он также использует `QXmlStreamReader` для чтения XML–структуры файла и извлечения соответствующей информации.

4.3.2 Класс CustomApplication

Класс является наследником класса `QApplication`. Он используется для создания кастомного приложения на основе Qt и выполняет установку иконки приложения и его имени. Этот класс обеспечивает кастомизацию приложения Qt, позволяя установить иконку и имя приложения, а также перехватывать определенные события, например, событие открытия файла, для выполнения дополнительных действий

Методы:

- `CustomApplication(int &argc, char **argv) : QApplication(argc, argv)` – конструктор класса `CustomApplication`, который инициализирует объект `QApplication`. Здесь вызываются методы `setAppIcon()` для установки иконки приложения и `setApplicationName()` для установки имени приложения;
- `notify(QObject *receiver, QEvent *event)`: Метод `notify`, переопределенный из `QApplication`. Он перехватывает события, которые могут возникнуть в приложении. В данном случае, при перехвате события `QEvent::FileOpen`, выполняется установка иконки приложения снова с помощью метода `setAppIcon()`. После этого, событие передается дальше на обработку базовому классу `QApplication`;

- `setAppIcon()` – метод `setAppIcon()` устанавливает иконку окна приложения с помощью `setWindowIcon()`, используя путь к файлу `icon.png`.

4.3.3 Класс Book

Этот класс служит для представления информации о книгах. Он имеет ряд методов для установки и получения значений заголовка книги (`title`), имени автора (`author`), изображения (`image`), HTML–контента (`html`), а также конструкторы для создания объектов класса `Book` с определенным идентификатором и без идентификатора.

Поля:

- `QString title`;
- `QString author`;
- `QString image`;
- `QString html`.

Методы:

- `setTitle(QString title)` – метод для установки заголовка книги;
- `getTitle()` – метод для получения заголовка книги;
- `setAuthor(QString author)` – метод для установки имени автора;
- `getAuthor()` – метод для получения имени автора;
- `setImage(QString image)` – метод для установки изображения книги;
- `getImage()` – метод для получения изображения книги;
- `setHtml(QString html)` – метод для установки HTML–контента книги;
- `getHtml()` – метод для получения HTML–контента книги;
- `Book(int id): BaseEntity(id)` – конструктор класса `Book`, который вызывает конструктор базового класса `BaseEntity` с передачей идентификатора;
- `Book():BaseEntity()` – конструктор класса `Book` без параметров, который вызывает конструктор базового класса `BaseEntity` без передачи идентификатора.

4.3.4 Класс Mainwindow

Класс является частью пользовательского интерфейса приложения, созданного с использованием библиотеки Qt. В нем содержатся методы для отображения списка книг, обработки событий нажатия на кнопки, открытия файлов, удаления книг из списка и поиска книг по их заголовкам.

Методы:

- `MainWindow(QWidget *parent)` – конструктор класса `MainWindow`, который инициализирует пользовательский интерфейс и устанавливает начальные значения;
- `~MainWindow()` – деструктор класса `MainWindow`, который освобождает ресурсы;
- `showBooks(std::vector<Book> books, QString searchTitle)` – метод для отображения списка книг в виджете `QListWidget`. Он создает виджеты для каждой книги, содержащие изображение, заголовок и имя автора, и связывает кнопку удаления с соответствующим слотом;
- `onListItemClicked(int id, QString html)` – метод, который обрабатывает событие щелчка на элементе списка книг. Он создает новое окно `Reading` для отображения содержимого выбранной книги;
- `deleteBookClicked(int id)` – метод для удаления книги из базы данных и списка отображаемых книг;
- `on_pushButton_clicked()` – обработчик события нажатия кнопки, который открывает диалоговое окно для выбора файла формата .fb2, конвертирует выбранный файл в HTML–контент и добавляет его в список отображаемых книг;
- `on_openShelves_clicked()` – обработчик события нажатия кнопки «Открыть полки», который отображает окно со списком полок (`shelves`);
- `isBookTitleMatchQString bookTitle, QString searchTitle)` – метод, который проверяет соответствие заголовка книги поисковому запросу.

4.4 Модуль отображения текста книги

4.4.1 Класс uiHelper

Класс предназначен для управления виджетом `QTextBrowser`, отображающим HTML–контент.

Методы:

- `nextPage(Ui::Reading *ui)` – метод для перехода к следующей странице в `QTextBrowser`. Он использует вертикальный ползунок (вертикальную полосу прокрутки) для увеличения значения на высоту видимой области `QTextBrowser`. Это приводит к прокрутке содержимого на одну страницу вниз;
- `prevPage(Ui::Reading *ui)` – метод для перехода к предыдущей странице в `QTextBrowser`. Он использует вертикальный ползунок для уменьшения значения на высоту видимой области `QTextBrowser`. Это приводит к прокрутке содержимого на одну страницу вверх;
- `initTextbrowser(Ui::Reading *ui, QString html, int currentPosition)` – метод для инициализации `QTextBrowser`. Он очищает

содержимое виджета, устанавливает новый HTML–контент, отключает вертикальную полосу прокрутки, включает режим переноса слов и устанавливает текущую позицию вертикальной полосы прокрутки.

4.4.2 Класс Reading

Этот класс представляет диалоговое окно для чтения содержимого книги и управления её отображением.

Методы:

- `Reading(QString content,int id,QWidget *parentProxy, QWidget *parent) : QDialog(parent)` – принимает содержимое книги (`content`), ID книги (`id`), указатель на родительское окно (`parentProxy`) и указатель на родительский виджет (`parent`). Инициализирует пользовательский интерфейс, группы кнопок (`group1, group2, group3`) для выбора стилей текста. Устанавливает соединения (`connect`) между различными сигналами и слотами для управления отображением содержимого книги. Использует метод `generateStyledHTML` для генерации стилизованного HTML и отображения содержимого книги;
- `on_exitButton_clicked()` – обрабатывает событие нажатия кнопки «Выход» для закрытия окна чтения книги;
- `on_prevButton_clicked()` – обрабатывает событие нажатия кнопки «Предыдущая страница»;
- `on_nextButton_clicked()` – обрабатывает событие нажатия кнопки «Следующая страница»;
- `on_fontStyleChanged(bool isChecked)` – обрабатывает событие изменения стиля текста;
- `on_backgroundColorChanged(bool isChecked)` – обрабатывает событие изменения цвета фона текста;
- `on_spinBox_valueChanged(int newValue)` – обрабатывает событие изменения значения размера шрифта;
- `on_fontChanged(bool isChecked)` – обрабатывает событие изменения шрифта текста;
- `displayBook(QString html)` – отображает содержимое книги в текстовом браузере;
- `generateStyledHTML(const QString& styleSheet, int fontSize, QString fontFamily)` – генерирует стилизованный HTML с заданным размером шрифта и типом шрифта.

4.5 Модуль управления полками

4.5.1 Класс AddShelfDialog

Класс представляет диалоговое окно для добавления новой полки книг.

Методы:

- `AddShelfDialog(std::vector<shelf> shelves, QWidget *parentProxy, QWidget *parent)` – закрывает родительское окно и устанавливает заголовок диалогового окна в название приложения;
- `on_addShelf_clicked()` – обрабатывает событие нажатия кнопки «Добавить полку». Получает введенное название полки из элемента lineEdit. Проверяет, свободно ли введенное название, перебирая уже существующие полки. Если название свободно, добавляет новую полку в базу данных, закрывает диалоговое окно и открывает родительское окно. Если название уже используется, выводит предупреждение;
- `on_cancelButton_clicked()` – обрабатывает событие нажатия кнопки «Отмена». Закрывает диалоговое окно и открывает родительское окно.

4.5.2 Класс SelectBooksToShelf

Класс предназначен для выбора книги для добавления на полку. Этот диалог позволяет пользователю выбирать книги из общего списка доступных книг, чтобы добавить их на выбранную полку.

Методы класса `SelectBooksToShelf` включают:

- `SelectBooksToShelf(int shelfId, std::vector<Book> booksOnShelf, QWidget *parentProxy, QWidget *parent)` – конструктор класса, инициализирующий окно диалога, принимающий идентификатор полки, список книг на этой полке и указатели на родительские виджеты. Здесь окно настраивается, отображаются доступные книги для выбора;
- `displayAllAvailableBooks` – метод для отображения всех доступных книг, которые можно добавить на полку. В этом методе создаются виджеты для каждой книги, отображаются изображения, названия и авторы книг;
- `checkBookNotOnShelf(int bookId)` – метод для проверки, находится ли книга уже на выбранной полке или нет. Он проверяет, есть ли книга с определенным идентификатором в списке книг, которые уже находятся на этой полке;
- `onListItemClicked(int bookId)` – метод, вызываемый при клике на элемент списка книг. Он добавляет выбранную книгу на полку с указанным идентификатором полки;
- `~SelectBooksToShelf()` – отвечает за освобождение памяти, занимаемой объектами интерфейса, созданными во время выполнения программы. Это важно для предотвращения утечек памяти;
- `on_cancelButton_clicked()` – вызывается при нажатии на кнопку «`Cancel`» в интерфейсе и закрывает текущее диалоговое окно

SelectBooksToShelf, затем отображает родительское окно, которое было скрыто при открытии этого диалогового окна.

4.5.3 Класс Shelf

Этот класс Shelf обеспечивает возможность установки и получения названия для объекта полки.

Методы:

- setNaming(QtString naming) – используется для установки названия;
- getNaming() – используется для получения значения названия этой полки.

4.5.4 Класс ShelfBooks

Класс ShelfBooks представляет диалоговое окно, которое отображает список книг на полке.

Методы:

- ShelfBooks(int shelfId, QString shelfName, QWidget *parentProxy, QWidget *parent) – конструктор класса ShelfBooks инициализирует объект класса. Он принимает shelfId – идентификатор полки, shelfName – название полки, parentProxy – родительское окно (диалоговое окно, из которого было вызвано текущее) и parent – родительское окно, в котором будет отображаться текущее. Внутри конструктора инициализируется пользовательский интерфейс и устанавливаются настройки окна;
- ~ShelfBooks() – деструктор освобождает выделенную память, уничтожая объект пользовательского интерфейса ui;
- on_shelvesButton_clicked() – вызывается при нажатии кнопки shelvesButton. Закрывает текущее окно ShelfBooks и отображает родительское окно;
- displayAllBooksForShelf(QString searchTitle) – отображает список всех книг на полке, фильтруемых по названию (если указан параметр поиска);
- displayAllBooksForShelf() – позволяет отобразить все книги на полке без фильтрации по названию;
- onListItemClicked(int id, QString html) – вызывается при клике на элементе списка книг и отображает содержимое выбранной книги;
- isBookTitleMatch(QString bookTitle, QString searchTitle) – проверяет соответствие названия книги критериям поиска;
- on_addToShelf_clicked() – открывает окно для добавления книги на полку при нажатии кнопки addToShelf;

- `on_lineEdit_textChanged(const QString &arg1)` – фильтрует книги на полке по тексту, введенному в строку поиска.

4.5.5 Класс ShelvesList

Этот класс обеспечивает удобную навигацию и управление полками с книгами, позволяя пользователю просматривать содержимое каждой полки и управлять ее составом.

Методы:

- `ShelvesList(QWidget *parentProxy, QWidget *parent) : QDialog(parent)` – создает объект класса `ShelvesList`. Инициализирует графический интерфейс с помощью `setupUi`. Закрывает родительское окно (если оно передано). Загружает список полок с помощью метода `showAllShelves`. Устанавливает заголовок окна приложения;
- `~ShelvesList()` – освобождает ресурсы, освобождает память, выделенную под объект `ui`;

`showAllShelves()` – загружает все полки из базы данных и отображает их в виде элементов `QListWidget` в графическом интерфейсе. Создает кнопку `DELETE` для каждой полки. Устанавливает стили и выравнивание для названий полок. Реализует удаление полки при нажатии на соответствующую кнопку;

- `onListItemClicked(int id, QString name)` – открывает окно ShelfBooks для выбранной полки. Создает новый экземпляр `ShelfBooks` с переданным идентификатором и названием полки и отображает его;
- `on_exitButton_clicked()` – отображает родительское окно (если оно было скрыто) и закрывает текущее окно;
- `on_addShelfBtn_clicked()` – открывает диалоговое окно для добавления новой полки `AddShelfDialog`. Загружает заново список полок после добавления новой полки.

5 РАЗРАБОТКА ПРОГРАММНЫХ МОДУЛЕЙ

В данном разделе рассмотрены описания алгоритмов, используемых в программе.

5.1 Алгоритм конвертора из FB2 в HTML

Для алгоритма по шагам рассмотрены методы класса `Fb2Helper`.

Шаг 1. Открытие файла FB2 для чтения;

Шаг 2. Создание XML-парсера для разбора содержимого файла (например, `QXmlStreamReader`);

Шаг 3. Инициализация переменных для хранения информации о книге (например, `firstName`, `lastName`, `bookTitle`);

Шаг 4. Инициализация переменной для хранения HTML-кода содержания книги (`bookHtml`);

Шаг 5. Инициализация переменной для хранения информации о книге (`Book`);

Шаг 6. Итерация по содержимому файла FB2;

Шаг 7. Чтение следующего элемента XML с помощью `readNext()`;

Шаг 8. Проверка типа элемента для дальнейшей обработки (начальный тег, конечный тег, текст и т. д.);

Шаг 9. Обработка начальных тегов;

Шаг 10. При обнаружении начального тега, содержащего информацию о книге, извлечение данных об авторе, названии книги и других сведениях;

Шаг 11. Обработка конечных тегов;

Шаг 12. При обнаружении конечного тега, завершение соответствующей операции обработки;

Шаг 13. Формирование HTML-кода;

Шаг 14. При обработке содержимого книги формирование HTML-кода с использованием соответствующих HTML-тегов для каждого элемента (параграфы, изображения, таблицы и т. д.);

Шаг 15. Сохранение изображений;

Шаг 16. При обнаружении изображений извлечение ссылок для дальнейшего использования;

Шаг 17. Обработка текста и других содержательных элементов книги;

Шаг 18. Обработка текста, форматирования, стилей и других содержательных элементов для включения в HTML-код книги;

Шаг 19. Управление структурой книги;

Шаг 20. Обработка информации о структуре книги (например, разделы, заголовки, содержание) для корректной организации и форматирования HTML-кода;

Шаг 21. Извлечение дополнительных данных;

Шаг 22. Извлечение дополнительной информации, такой как аннотация, описание, метаданные и другие сведения о книге;

Шаг 23. Обработка ссылок;

Шаг 24. Обработка ссылок и гиперссылок, включая создание внутренних ссылок в тексте книги;

Шаг 25. Заполнение объекта `Book`;

Шаг 26. Использование извлеченных данных об авторе, названии книги и других данных для заполнения объекта `Book`;

Шаг 27. Проверка условия завершения итерации;

Шаг 28. Проверка наличия дальнейших элементов в содержимом файла FB2 для обработки. В случае отсутствия оставшихся элементов или достижения конца файла завершение итерации;

Шаг 29. Проверка наличия ошибок при разборе XML. В случае возникновения ошибки прекращение итерации и обработка ошибки для корректной работы алгоритма;

Шаг 30. В случае успешного завершения обработки всех элементов файла FB2 и отсутствия ошибок выполнение завершающих операций;

Шаг 31. Завершение формирования HTML-кода с учетом всех обработанных элементов книги;

Шаг 32. Подготовка объекта Book с учтенной информацией об авторе, названии книги и других данных, извлеченных в процессе итерации;

Шаг 33. Окончание итерации и завершение алгоритма, возвращение объекта Book с готовыми данными о книге и сформированным HTML-кодом содержания для использования в дальнейших процессах приложения или программы.

Схема алгоритма представлена в Приложении В.

5.2 Алгоритм отображения списка книг на экране

Для алгоритма по шагам рассмотрен метод `void showBooks(std::vector<Book> books, QString searchTitle)` класса `MainWindow`.

Шаг 1. Определение максимальных размеров изображения для отображения в списке книг (`maxImageWidth = 100, maxImageHeight = 150`);

Шаг 2. Отключение существующих соединений с элементом `QListWidget` и очистка его содержимого;

Шаг 3. Итерация по списку книг, переданному в функцию;

Шаг 4. Проверка, содержит ли заголовок книги (`searchTitle`) текст поиска, и соответствует ли заголовок книги критериям поиска;

Шаг 5. Создание нового элемента `QListWidgetItem` для каждой книги;

Шаг 6. Установка данных книги (`ID`) в `UserRole` элемента `QListWidgetItem`;

Шаг 7. Создание пользовательского виджета для отображения информации о книге (название, автор, изображение, кнопка удаления);

Шаг 8. Создание кнопки удаления с определенными характеристиками;

Шаг 9. Создание горизонтального слоя для пользовательского виджета;

Шаг 10. Создание меток для изображения, заголовка и автора книги;

Шаг 11. Загрузка изображения книги из HTML-тега, изменение его размеров и установка в `QLabel` для отображения;

Шаг 12. Настройка стилей для меток заголовка и автора;

Шаг 13. Установка стилей для пользовательского виджета;

Шаг 14. Установка фиксированной ширины для метки изображения;

Шаг 15. Размещение меток заголовка и автора в вертикальном слое;

Шаг 16. Размещение элементов (изображения, текста, кнопки) в горизонтальном слое пользовательского виджета;

Шаг 17. Установка размера виджета в элементе `QListWidgetItem`;

Шаг 18. Установка пользовательского виджета в элемент списка;

Шаг 19. Соединение нажатия кнопки удаления с соответствующим действием;

Шаг 20. Соединение нажатия на элемент списка с действием отображения содержимого книги;

Шаг 21. Открытие окна чтения для выбранной книги при щелчке на элементе списка;

Шаг 22. Очистка поля ввода поиска после открытия книги;

Шаг 23. Показ книг без применения фильтрации по заголовку;

Шаг 24. Вызов функции `showBooks(books, QString(""))` для отображения всех книг без поиска;

Шаг 25. Отображение выбранной книги в окне для чтения;

Шаг 26. Создание нового окна `Reading` с HTML-содержимым выбранной книги и отображение этого окна;

Шаг 27. Очистка поля ввода поиска после открытия книги;

Шаг 28. Завершение функции `onListItemClicked`.

Схема алгоритма представлена в Приложении В.

5.3 Алгоритм отображения окна с текстом книги

Для алгоритма по шагам рассмотрены методы класса `Reading`.

Шаг 1. Конструктор `Reading` принимает содержание книги (`content`) в формате HTML, идентификатор книги (`id`), указатель на родительский виджет (`parentProxy`) и родительский объект (`parent`). Создает экземпляр класса `Ui::Reading` и настраивает пользовательский интерфейс;

Шаг 2. `fontSize` устанавливает значение шрифта для виджета `spinBox`;

Шаг 3. Присваивание родительское окно для текущего виджета из `parentProxy`;

Шаг 4. Создание и настраивает группы радиокнопок для выбора стилей шрифта и фона;

Шаг 5. Установка начальных значений выбранных радиокнопок в каждой группе;

Шаг 6. Установка соединений (`connect`). Связываются сигналы изменения состояния радиокнопок со слотами для обновления стилей шрифта, фона и размера шрифта, а также обновления содержимого книги;

Шаг 7. Отображение содержимого книги. Вызывается метод `displayBook` для отображения содержимого книги на основе стиля, размера шрифта и типа шрифта;

Шаг 8. Закрытие родительского окна `parentProxy`;

Шаг 9. Деструктор `Reading` освобождает память, выделенную для `ui`;

Шаг 10. Слот `on_exitButton_clicked()`;

Шаг 11. Слоты `on_prevButton_clicked()` и `on_nextButton_clicked()` - обработчики для кнопок `Previous` и `Next`, позволяющие перемещаться по страницам текста;

Шаг 12. Слот `on_fontStyleChanged(bool isChecked)`. Обработчик изменений стиля шрифта на основе выбранных радиокнопок;

Шаг 13. Слот `on_backgroundColorChanged(bool isChecked)`. Обработчик изменений фона: устанавливает новую цветовую палитру на основе выбранных радиокнопок и отображает обновленное содержимое книги;

Шаг 14. Слот `on_spinBox_valueChanged(int newValue)`. Обработчик изменения размера шрифта: обновляет переменную `fontSize`, сохраняет текущую позицию прокрутки и отображает обновленное содержимое книги;

Шаг 15. Метод `generateStyledHTML(const QString& styleSheet, int fontSize, QString fontFamily)` генерирует HTML-содержимое книги с заданными стилем, размером шрифта и типом шрифта;

Шаг 16. Слот `on_fontChanged(bool isChecked)`.Обработчик изменения типа шрифта: устанавливает новый тип шрифта, сохраняет текущую позицию прокрутки и отображает обновленное содержимое книги.

6 ПРОГРАММА И МЕТОДИКА ИСПЫТАНИЙ

В данном разделе описывается функциональное тестирование программы.

6.1 Тестирование выбора файла неподдерживаемого формата

При попытке загрузить файл не формата FB2, в проводнике не будут отображаться файлы, которые выбрать нельзя (Рисунок 6.1).

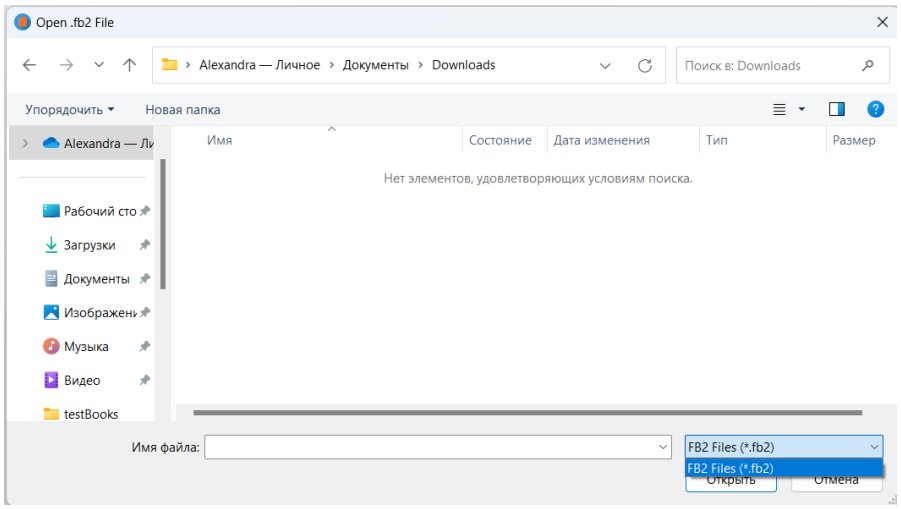

Рисунок 6.1 – Проводник.

6.2 Тестирование уникальности названия полки

При попытке ввести уже существующие название полки, выводится сообщение об ошибке и название запрашивается повторно (Рисунок 6.2).

Рисунок 6.2 – Создание новой полки с существующим названием.

7 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

7.1 Системные требования

Данное приложение разработано в QT Creator на 64–разрядной операционной системе Windows 11. Процессор Intel Core i7–10510U, размер оперативной памяти 16 Гб. Для нормальной работы данного приложения требуется не менее 100 Мб свободной оперативной памяти.

7.2 Использование приложения

Для запуска программы необходим открыть файлы исходного кода в Qt Creator и собрать проект. Информация по созданию проекта находится в источнике [5]. Когда программа запустится, необходимо добавить файлы формата FB2 для дальнейшей работы с ними (Рисунок 7.1).

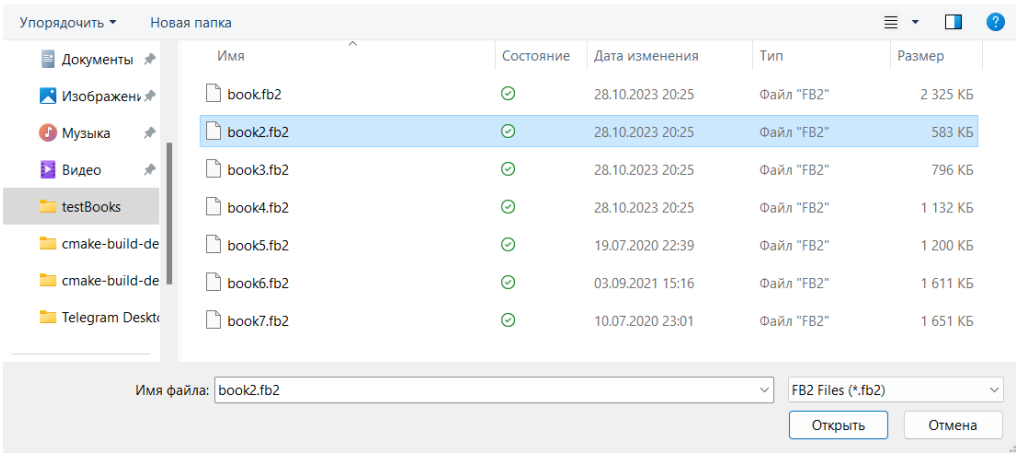

Рисунок 7.1 – Загрузка файлов.

После этого откроется окно программы с главным меню, где будет отображаться список добавленных книг (Рисунок 7.2).

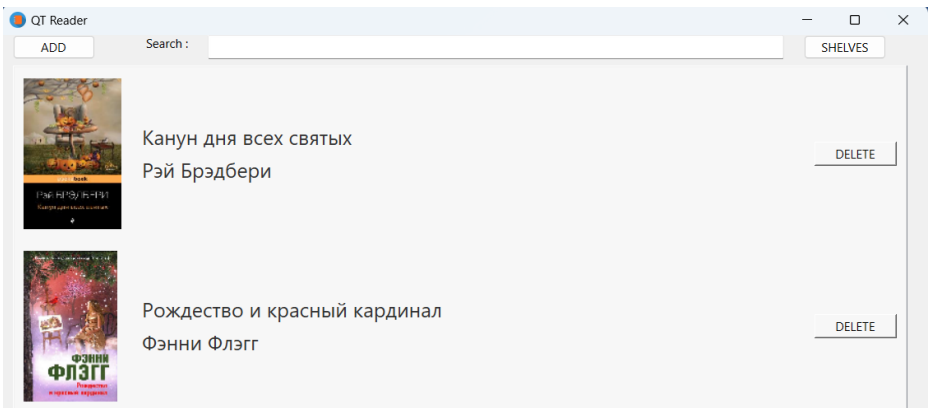

Рисунок 7.2 – Главное меню.

При нажатии на поле книги откроется окно, где будет отображаться текст и характеристики, которые можно изменять (Рисунок 7.3 и Рисунок 7.4).

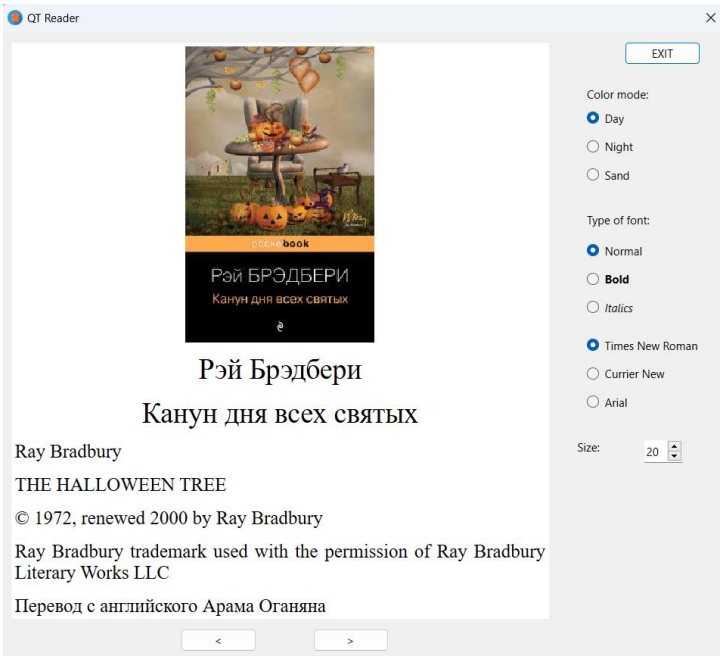

Рисунок 7.3 – Отображение содержимого книги с характеристиками по умолчанию.

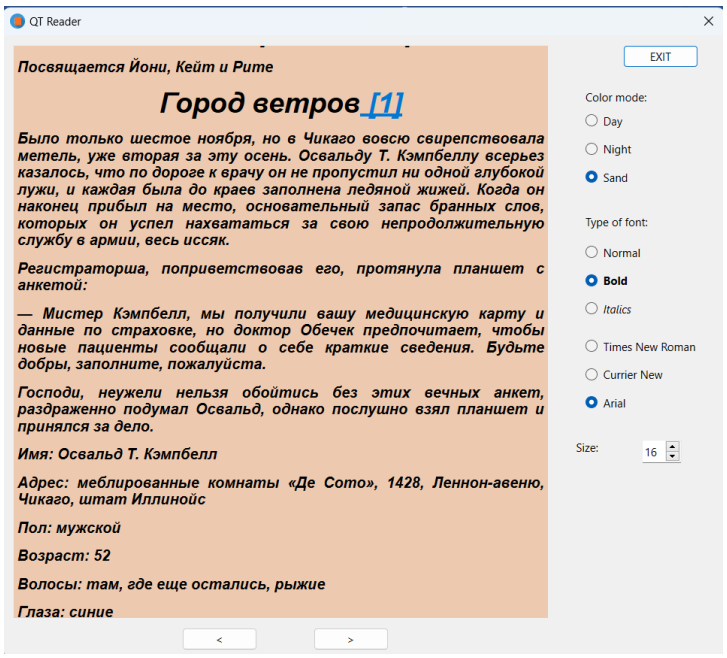

Рисунок 7.4 – Отображение содержимого книги с измененными характеристиками.

После того как книга была прочитана, можно ее добавить в полку «READ» (Рисунок 7.5 и Рисунок 7.6).

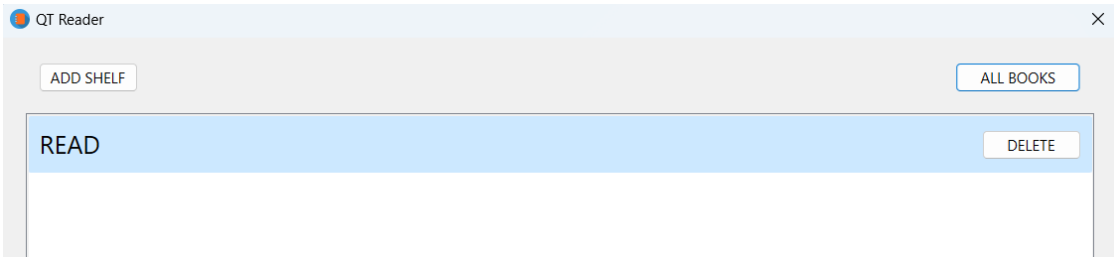

Рисунок 7.5 – Список полок.

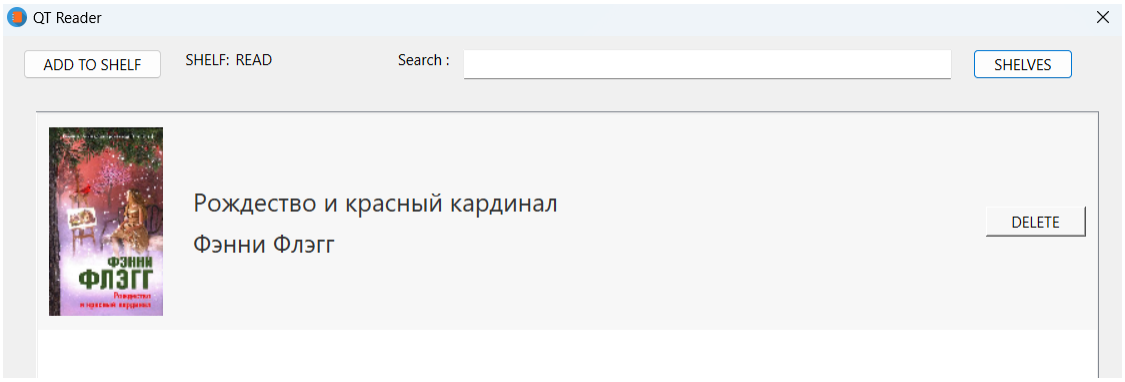

Рисунок 7.6 – Книги в полке.

Чтобы найти нужную книгу из списка, можно воспользоваться строкой поиска по названию (Рисунок 7.7).

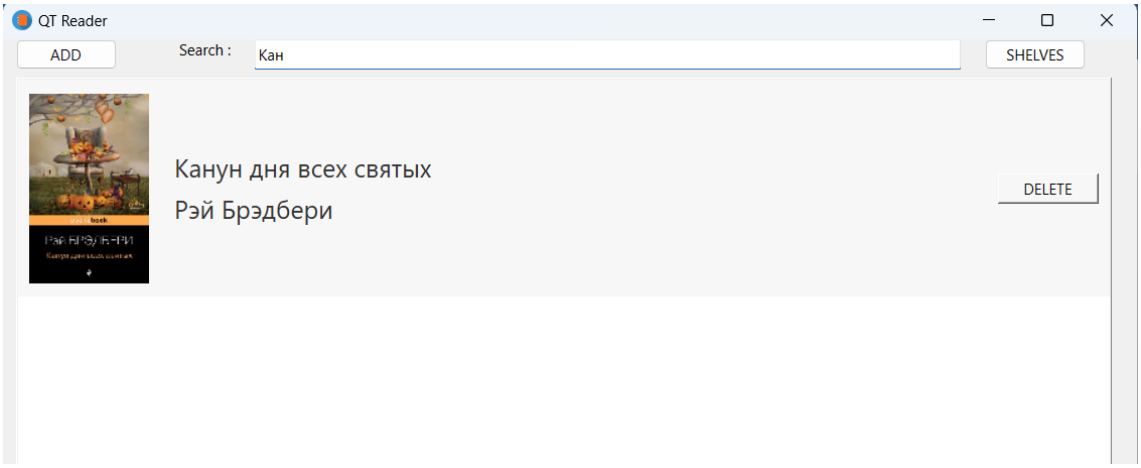

Рисунок 7.7 – Поиск книги по названию

ЗАКЛЮЧЕНИЕ

Данный курсовой проект представляет собой разработанное приложение, которое обладает широким функционалом, интуитивно понятным пользовательским интерфейсом и собственной базой данных. В ходе его создания были успешно достигнуты поставленные цели, и весь запланированный функционал был реализован полностью.

Для создания программного продукта была проведено детальное изучение среды разработки Qt Creator, а также основ базы данных SQLite и структуры файлов в формате FB2. Исследования помогли получить глубокое понимание особенностей каждого элемента, что сыграло ключевую роль в успешной реализации проекта.

Основное внимание уделено языку программирования C++, где были усвоены основы объектно-ориентированного программирования (ООП), что позволило эффективно использовать его возможности при создании различных модулей приложения. Опыт написания функции парсера открыл новые возможности по извлечению и обработке данных из файлов.

Работа над проектом была разбита на этапы: анализ аналогов и литературных источников, постановка требований, проектирование, конструирование, разработка модулей и тестирование. Благодаря последовательности выполнения каждого этапа был достигнут результат – функциональный и стабильно работающий программный модуль «Система контроля процесса чтения».

В будущем планируется улучшение текущего функционала. Планируется добавить новые возможности, такие как выделение участков текста маркером, графическое отображение прогресса прочитанных страниц и возможность создания закладок для удобства пользователей.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. C++ GUI Programming with Qt 4, – Jasmin Blanchette, Mark Summerfield, 2015
2. Книжные магазины и приложения для чтения книг [Электронный ресурс]. -Электронные данные. -Режим доступа: https://habr.com/ru/companies/maccentre/articles/463987/ - Дата доступа: 27.11.2023
3. The C++ Programming Language, – Bjarne Stroustrup, 1985
4. The C Programming Language. 2nd Edition, –Dennis Ritchie, Brian Kernighan, 1978
5. Qt Documentation [Электронный ресурс]. -Электронные данные. -Режим доступа: https://doc.qt.io/ - Дата доступа: 27.11.2023

ПРИЛОЖЕНИЕ А
(обязательное)

Схема структурная

ПРИЛОЖЕНИЕ Б
(обязательное)

Диаграмма классов

ПРИЛОЖЕНИЕ В

(обязательное)

Схема алгоритма конвертора из FB2 в HTML

ПРИЛОЖЕНИЕ Г

(обязательное)

Схема алгоритма отображения списка книг на экране

ПРИЛОЖЕНИЕ Д
(обязательное)

Листинг кода

Файл addshelfdialog.cpp:
```
#include "addshelfdialog.h"
#include "customapplication.h"
#include "shelveslist.h"
#include "ui_addshelfdialog.h"
AddShelfDialog::AddShelfDialog(std::vector<shelf> shelves, QWidget
*parentProxy, QWidget *parent) :
  QDialog(parent),
  ui(new Ui::AddShelfDialog) {
  this->shelves = shelves;
  ui->setupUi(this);
  parentWindow = parentProxy;
  parentProxy->close();
  setWindowTitle(CustomApplication::applicationName()); {
AddShelfDialog::~AddShelfDialog() {
  delete ui; {
void AddShelfDialog::on_addShelf_clicked() {
  try {
    QString shelfName = ui->lineEdit->text();
    bool nameIsFree = true;
    for (shelf shelf : shelves) {
      if (shelf.getNaming() == shelfName) {
        nameIsFree = false;
        break; { {
    if (!nameIsFree) {
      throw std::runtime_error("Name is already in use"); {
    db->addShelf(shelfName);
    parentWindow->show();
    this->close(); {
  catch (const std::runtime_error &e) {
    ui->nameInUseWarning->setText(e.what()); { {
void AddShelfDialog::on_cancelButton_clicked() {
  parentWindow->show();
  this->close(); {
```

Файл addshelfdialog.h:
```
#ifndef ADDSHELFDIALOG_H
#define ADDSHELFDIALOG_H
#include <QWidget>
#include <QDialog>
#include "database.h"
#include <vector>
#include "shelf.h"
namespace Ui {
class AddShelfDialog; {
class AddShelfDialog : public QDialog {
```

```
  Q_OBJECT
public:
  explicit AddShelfDialog(std::vector<shelf> shelves,QWidget
*parentProxy, QWidget *parent = nullptr);
  ~AddShelfDialog();
private slots:
  void on_addShelf_clicked();
  void on_cancelButton_clicked();
private:
  Ui::AddShelfDialog *ui;
  QWidget *parentWindow;
  database* db;
  std::vector<shelf> shelves;
};
#endif // ADDSHELFDIALOG_H
```

Файл baseentity.cpp:

```
#include "baseentity.h"
BaseEntity::BaseEntity(int id) {
  this->id=id; {
BaseEntity::BaseEntity(){}
int BaseEntity::getId() const {
  return id; {
void BaseEntity::setId(int newId) {
  id = newId; {
```

Файл baseentity.h:

```
#ifndef BASEENTITY_H
#define BASEENTITY_H
class BaseEntity {
private:
  int id;
public:
  BaseEntity(int id);
  BaseEntity();
  int getId() const;
  void setId(int newId);
};
#endif // BASEENTITY_H
```

Файл book.cpp:

```
#include "book.h"
void Book::setTitle(QString title) {
   this->title=title; {
QString Book::getTitle() {
    return title; {
void Book::setAuthor(QString author) {
   this->author=author; {
QString Book::getAuthor() {
    return author; {
void Book::setImage(QString image) {
   this->image=image; {
```

```
QString Book::getImage() {
    return image; {
void Book::setHtml(QString html) {
  this->html=html; {
QString Book::getHtml() {
  return html; {
Book::Book(int id): BaseEntity(id){}
Book::Book():BaseEntity(){}
```

Файл book.h:

```
#ifndef BOOK_H
#define BOOK_H
#include "baseentity.h"
#include <qstring.h>
class Book: public BaseEntity {
private:
  QString title;
  QString author;
  QString image;
  QString html;
public:
  void setTitle(QString title);
  QString getTitle();
  void setAuthor(QString author);
  QString getAuthor();
  void setImage(QString image);
  QString getImage();
  void setHtml(QString html);
  QString getHtml();
  Book(int id);
  Book();
};
#endif // BOOK_H
```

Файл customapplication.cpp:

```
#include "customapplication.h"
CustomApplication::CustomApplication(int &argc, char **argv) :
QApplication(argc, argv) {
  setAppIcon();
  setApplicationName("QT Reader"); {
bool CustomApplication::notify(QObject *receiver, QEvent *event) {
  if (event->type() == QEvent::FileOpen) {
    // Set the application icon here
    setAppIcon(); {
  return QApplication::notify(receiver, event); {
void CustomApplication::setAppIcon() {
  setWindowIcon(QIcon("путь к icon.png")); {
```

Файл customapplication.h:

```
#ifndef CUSTOMAPPLICATION_H
#define CUSTOMAPPLICATION_H
#include "QApplication"
```

```
#include <QIcon>
class CustomApplication : public QApplication {
  Q_OBJECT
public:
  CustomApplication(int &argc, char **argv);
  bool notify(QObject *receiver, QEvent *event) override;
private:
  void setAppIcon();
};
#endif // CUSTOMAPPLICATION_H
```

Файл database.cpp:

```
#include "Book.h"
#include "database.h"
#include <vector>
database::database() {
db.setDatabaseName("путь к qtreader.db");
  if (db.open()) {
    qDebug() << "Успешное подключение к серверу MySQL!";
    QSqlQuery query;
    query.exec("PRAGMA foreign_keys = ON;"); { {
database::~database() {
  if (db.open()) {
    qDebug() << "Успешное отключение к серверу MySQL!";
    db.close();
  } else {
    qDebug() << "сервер SQLite не запущен: " <<
db.lastError().text(); { {
void database::addBook(Book book) {
  QSqlQuery query;
  QString insertQuery = "INSERT INTO Book (title, author, img,
content) VALUES (?, ?, ?, ?)";
  query.prepare(insertQuery);
  query.bindValue(0, book.getTitle());
  query.bindValue(1, book.getAuthor());
  query.bindValue(2, book.getImage());
  query.bindValue(3, book.getHtml());
  if (query.exec()) {
    qDebug() << "Insert successful.";
  } else {
    qDebug() << "Insert failed: " << query.lastError().text(); { {
std::vector<Book> database::getAllBooks() {
  std::vector<Book> books;
  QString sqlQuery = "SELECT * FROM Book;";
  QSqlQuery query;
  if (query.exec(sqlQuery)) {
    while (query.next()) {
      Book book;
      int id = query.value(0).toInt();
      QString title = query.value(1).toString();
      QString author = query.value(2).toString();
       QString image = query.value(3).toString();
```

```
        QString html = query.value(4).toString();
        book.setId(id);
        book.setTitle(title);
        book.setAuthor(author);
        book.setImage(image);
        book.setHtml(html);
        books.push_back(book); {
  } else {
    qDebug() << "Query failed: " << query.lastError().text(); {
  return books; {
void database::deleteBook(int id) {
  QSqlQuery query;
  QString deleteBookFromAllShelvesQuery = "DELETE FROM
BooksOnShelfes WHERE book_id=?";
  QString deleteBookQuery = "DELETE from Book where id=?";
  if (db.transaction()) {
    query.prepare(deleteBookFromAllShelvesQuery);
    query.bindValue(0,id);
    if (query.exec()) {
       qDebug() << "Book deleted from all shelves.";
    } else {
       qDebug() << "Delete book from all shelves failed: " <<
query.lastError().text(); {
    query.prepare(deleteBookQuery);
    query.bindValue(0,id);
    if (query.exec()) {
       qDebug() << "Book deleted.";
    } else {
       qDebug() << "Delete failed: " << query.lastError().text(); {
    if (db.commit()) {
       qDebug() << "Transation completed."; {
    else {
       db.rollback(); { {
  else {
    qDebug() << "Unable to start transaction"; { {
void database::addShelf(QString shelfName) {
  QSqlQuery query;
  QString insertQuery = "INSERT INTO Shelf (naming) VALUES (?);";
  query.prepare(insertQuery);
  query.bindValue(0, shelfName);
  if (query.exec()) {
    qDebug() << "Insert successful.";
  } else {
    qDebug() << "Insert failed: " << query.lastError().text(); { {
std::vector<shelf> database::getAllShelves() {
  std::vector<shelf> shelves;
  QString sqlQuery = "SELECT * FROM Shelf;";
  QSqlQuery query;
  if (query.exec(sqlQuery)) {
    while (query.next()) {
       shelf shelf;
       int id = query.value(0).toInt();
```

```
        QString naming = query.value(1).toString();
        shelf.setId(id);
        shelf.setNaming(naming);
        shelves.push_back(shelf); {
    } else {
      qDebug() << "Query failed: " << query.lastError().text(); {
    return shelves; {
void database::addBookOnShelf(int shelfId, int bookId) {
    QSqlQuery query;
    QString insertQuery = "INSERT INTO BooksOnShelfes (shelf_id,
book_id) VALUES (?,?)";
    query.prepare(insertQuery);
    query.bindValue(0, shelfId);
    query.bindValue(1, bookId);
    if (query.exec()) {
      qDebug() << "Insert successful.";
    } else {
      qDebug() << "Insert failed: " << query.lastError().text(); { {
void database::deleteBookFromShelf(int shelfId, int bookId) {
    QSqlQuery query;
    QString insertQuery = "DELETE FROM BooksOnShelfes WHERE book_id =
? AND shelf_id = ?";
    query.prepare(insertQuery);
    query.bindValue(0, bookId);
    query.bindValue(1, shelfId);
    if (query.exec()) {
      qDebug() << "Insert successful.";
    } else {
      qDebug() << "Insert failed: " << query.lastError().text(); { {
void database::deleteShelf(int shelfId) {
    QSqlQuery query;
    QString deleteAllBooksFromShelfQuery = "DELETE FROM
BooksOnShelfes WHERE shelf_id=?";
    QString deleteShelfQuery = "DELETE FROM Shelf WHERE id=?";
//  if (db.transaction())
//  {
      query.prepare(deleteAllBooksFromShelfQuery);
      query.bindValue(0, shelfId);
      if (query.exec()) {
         qDebug() << "All books deleted from shelf.";
      } else {
         qDebug() << "Delete all books from shelf failed: " <<
query.lastError().text(); {
      query.prepare(deleteShelfQuery);
      query.bindValue(0, shelfId);
      if (query.exec()) {
         qDebug() << "Shelf deleted.";
      } else {
         qDebug() << "Delete failed: " << query.lastError().text();
std::vector<Book> database::getBooksFromShelf(int shelfId) {
    std::vector<Book> books;
```

```
  QString sqlQuery = "SELECT Book.id, Book.title, Book.author,
Book.img, Book.content FROM Book INNER JOIN BooksOnShelfes ON
BooksOnShelfes.book_id = Book.id INNER JOIN Shelf ON Shelf.id =
BooksOnShelfes.shelf_id WHERE Shelf.id = ?";
  QSqlQuery query;
  query.prepare(sqlQuery);
  query.bindValue(0, shelfId);
  if (query.exec()) {
    while (query.next()) {
       Book book;
       int id = query.value(0).toInt();
       QString title = query.value(1).toString();
       QString author = query.value(2).toString();
       QString image = query.value(3).toString();
       QString html = query.value(4).toString();
       book.setId(id);
       book.setTitle(title);
       book.setAuthor(author);
       book.setImage(image);
       book.setHtml(html);
       books.push_back(book); {
  } else {
    qDebug() << "Query failed: " << query.lastError().text(); {
  return books; {
```

Файл database.h:

```
#ifndef DATABASE_H
#define DATABASE_H
#include "Book.h"
#include <QCoreApplication>
#include <QtSql>
#include "shelf.h"
class database {
private:
  QSqlDatabase db = QSqlDatabase::addDatabase("QSQLITE");
public:
  database();
  ~database();
  void addBook(Book book);
  std::vector<Book>  getAllBooks();
  void deleteBook(int id);
  void addShelf(QString shelfName);
  std::vector<shelf> getAllShelves();
  void addBookOnShelf(int shelfId, int bookId);
  void deleteBookFromShelf(int shelfId, int bookId);
  void deleteShelf(int shelfId);
  std::vector<Book> getBooksFromShelf(int shelfId);
};
#endif // DATABASE_H
```

Файл fb2helper.cpp:

```
#include "fb2helper.h"
```

```
#include "mainwindow.h"
#include <qfile.h>
#include <qxmlstream.h>
fb2helper::fb2helper(){}
Book fb2helper::convertFb2ToHTML(QString filePath) {
  QString book;
  QStringList content;
  QFile f(filePath);
  if (!f.open(QIODevice::ReadOnly | QIODevice::Text)) {
    // qDebug() << "файл не открыт"; {
  bool ok = true;
  QString special;
  QString description; // описание
  //  настройки отображения
  int fontSize = 20;
  QXmlStreamReader sr(&f);
  QString rId;
  QString rType;
  QStringList titles;
  QStringList authors;
  QString opt;
  QStringList images;
  QStringList thisToken;
  bool insideTitleInfo = false;
  QString author;
  QString title;
  while( !sr.atEnd() ) {
    switch( sr.readNext() ) {
    case QXmlStreamReader::NoToken:
      break;
    case QXmlStreamReader::StartDocument:
      book = "<!DOCTYPE HTML><html><body style=\"font-size:%1px;
font-family:%2;\">";
      break;
    case QXmlStreamReader::EndDocument:
      book.append("</body></html>");
      break;
    case QXmlStreamReader::StartElement:
      thisToken.append( sr.name().toString() );
      if(thisToken.contains("description")) // ОПИСАНИЕ КНИГИ {
        if( thisToken.back() != "image" ) //пропускаем всё кроме
обложки
          break; // не выводим {
      if (sr.name().toString() == "title-info") {
        insideTitleInfo = true; {
      else if (insideTitleInfo && sr.name().toString() == "author")
{
        while (sr.readNextStartElement()) {
          if (sr.name().toString() == "first-name" ||
sr.name().toString() == "last-name") {
            author += sr.readElementText() + " "; { { {
```

```
      else if (insideTitleInfo && sr.name().toString() == "book-
title") {
        title = sr.readElementText(); {
      if(sr.name().toString() == "title") {
        content.append(""); // добавляем пункт содержания
        break; {
      if( sr.name().toString() == "body" )
        if( !sr.attributes().isEmpty()
          && sr.attributes().first().value().toString() == "notes")
          special = "notes";  // режим примечаний
      if(special == "notes") {
        if( sr.name().toString() == "section" ) {
          if( sr.attributes().count() > 0 ) {
            rId = sr.attributes().at(0).value().toString(); //
ссылка на текст
            rType = ""; { { {
      opt = " align=\"justify\"";
      if(thisToken.contains("title") ) {
        opt = " align=\"center\" style=\"font-size:"
+QString::number(int(fontSize * 1.5)) + "px\" ";
        if(special == "notes") {
          opt += (" id=\"" + rId + "\""); { {
      if(thisToken.contains("subtitle") ) {
        opt = " align=\"center\" style=\"font-size:"
+QString::number(int(fontSize * 1.2)) + "px\" "; {
      if(thisToken.contains("annotation") ) {
        opt = " align=\"left\" "; {
      if(sr.name().toString() == "p"
        || sr.name().toString() == "subtitle") {
        book.append("<p"+opt +" >");
        break; {
      if( sr.name().toString() == "table" ) {
        QString text;
        for(int i = 0; i < sr.attributes().count(); i++) {
          if(sr.attributes().at(i).name() == QString("id"))
            //     qDebug() <<
sr.attributes().at(i).value().toString();
            if(sr.attributes().at(i).name() == QString("style"))
              text.append( "style=\""
+sr.attributes().at(i).value().toString()+ ";\"" ); {
        book.append("<table border=1 align=\"center\"
style=\"border:solid;\" " + text + ">");
        break; {
      if( sr.name().toString() == "tr" ) {
        QString text;
        if(!thisToken.contains("table"))
          //    qDebug() << "ошибка в таблице";
          for(int i = 0; i < sr.attributes().count(); i++) {
            if(sr.attributes().at(i).name() == QString("aling"))
              text.append( "aling=\""
+sr.attributes().at(i).value().toString()+ "\"" );
            else
```

```
              qDebug() << "<tr>" << sr.attributes().at(i).name() <<
sr.attributes().at(i).value().toString(); {
        book.append("<tr " + text + ">");
        break;
      }
      if( sr.name().toString() == "td"
        || sr.name().toString() == "th" ) {
        QString text;
        for(int i = 0; i < sr.attributes().count(); i++) {
          if(sr.attributes().at(i).name() == QString("aling"))
            text.append( "aling=\""
+sr.attributes().at(i).value().toString()+ "\" " );
          else if(sr.attributes().at(i).name() ==
QString("valing"))
            text.append( "valing=\""
+sr.attributes().at(i).value().toString()+ "\" " );
          else if(sr.attributes().at(i).name() ==
QString("colspan"))
            text.append( "colspan="
+sr.attributes().at(i).value().toString()+ " " );
          else if(sr.attributes().at(i).name() ==
QString("rowspan"))
            text.append( "rowspan="
+sr.attributes().at(i).value().toString()+ " " );
          else
            qDebug() << "<td th>" << sr.attributes().at(i).name()
<< sr.attributes().at(i).value().toString(); {
        book.append( "<"+sr.name().toString()+ " " + text +">" );
        break; {
      if( sr.name().toString() == "empty-line" ) {
        book.append("<br/>");
        break; {
      if(sr.name().toString() == "strong"
        || sr.name().toString() == "sup"
        || sr.name().toString() == "sub"
        || sr.name().toString() == "code"
        || sr.name().toString() == "cite") {
        book.append( "<" + sr.name().toString() + ">");
        break; {
      if(sr.name().toString() == "emphasis") {
        book.append( "<i>" );
        break; {
      if( sr.name().toString() == "v" ) {
        book.append("<p align=\"left\" style=\"margin-
left:25px;\">");
        break; {
      if(sr.name().toString() == "strikethrough") {
        book.append( "<strike>" );
        break; {
      if( sr.name().toString() == "a" ) {
        rId = "";
        for(int i = 0; i < sr.attributes().count(); i++) {
```

```
          if(sr.attributes().at(i).name() == QString("type") ) {
            //rType = sr.attributes().at(i).value().toString(); {
          if(sr.attributes().at(i).name() == QString("href")) {
            rId = sr.attributes().at(i).value().toString(); { {
        book.append("<a href=\"" + rId + "\"> "); {
      if(sr.name().toString() == "poem"
        || sr.name().toString() == "stanza"
        || sr.name().toString() == "epigraph") {
        break; {
      if(sr.name().toString() == "text-author" ) // автор текстта {
        book.append( "<p align=\"justify\" style=\"margin-
left:45px;\">" );
        break; {
      if(sr.name().toString() == "date" ) // автор текстта {
        book.append( "<p align=\"justify\" style=\"margin-
left:45px;\">" );
        break; {
      if( sr.name().toString() == "image" ) // расположение
рисунков {
        if(sr.attributes().count() > 0)
          book.append("<p
align=\"center\">"+sr.attributes().at(0).value().toString() + "#" +
"</p>"); {
      if(sr.name() == QString("binary")) // хранилище рисунков {
        if(sr.attributes().at(0).name() == QString("id")) {
          rId = sr.attributes().at(0).value().toString();
          rType = sr.attributes().at(1).value().toString(); {
        if(sr.attributes().at(1).name() == QString("id")) {
          rId = sr.attributes().at(1).value().toString();
          rType = sr.attributes().at(0).value().toString(); { {
      break;
    case QXmlStreamReader::EndElement:
      if( thisToken.last() == sr.name().toString() ) {
        thisToken.removeLast(); {
      else
        //  qDebug() << "error token";
        if(thisToken.contains("description")) // ОПИСАНИЕ КНИГИ {
          break; // не выводим {
      if (sr.name().toString() == "title-info") {
        insideTitleInfo = true; {
      else if (insideTitleInfo && sr.name().toString() == "author")
{
        while (sr.readNextStartElement()) {
          if (sr.name().toString() == "first-name" ||
sr.name().toString() == "last-name") {
            author += sr.readElementText() + " "; { { {
      else if (insideTitleInfo && sr.name().toString() == "book-
title") {
        title = sr.readElementText(); {
      if( sr.name().toString() == "p"
        || sr.name().toString() == "subtitle"
        || sr.name().toString() == "v"
```

```
        || sr.name().toString() == "date"
        || sr.name().toString() == "text-author") {
        book.append("</p>");
        break; {
      if(sr.name().toString() == "text-autor")
        authors.append(sr.name().toString());
      if(sr.name().toString() == "td"
        || sr.name().toString() == "th"
        || sr.name().toString() == "tr"
        || sr.name().toString() == "table"
        || sr.name().toString() == "sup"
        || sr.name().toString() == "sub"
        || sr.name().toString() == "strong"
        || sr.name().toString() == "code"
        || sr.name().toString() == "cite") {
        book.append( "</"+sr.name().toString()+">" );
        break; {
      if( sr.name().toString() == "a" ) {
        rId.remove("#");
        book.append( "</a><span id=\"" + rId + "___" + "\"></span>"
);
        //  qDebug() << "id" << rId + "___";
        break; {
      if(sr.name().toString() == "emphasis") {
        book.append( "</i>" );
        break; {
      if(sr.name().toString() == "strikethrough") {
        book.append( "</strike>" );
        break; {
      if(sr.name().toString() == "stanza") // конец строфы {
        break; {
      if(sr.name().toString() == "epigraph"
        || sr.name().toString() == "poem") {
        break; {
      if(special == "notes") // режим извлечения примечаний {
        if( sr.name().toString() == "body" ) {
          special = ""; {
        if( sr.name().toString() == "section" ) {
          book.insert(book.lastIndexOf("<"), "<a href=\"#" + rId +
"___" + "\"> назад</a>"); { {
      break;
    case QXmlStreamReader::Characters:
      if( sr.text().toString() == "" ) {
        break; {
      if( sr.text().toString() == "\n" ) {
        break; {
      if(thisToken.contains("description")) // ОПИСАНИЕ КНИГИ {
        description.append(sr.text().toString() + " "); // не
выводим
        break; {
      if (sr.name().toString() == "title-info") {
        insideTitleInfo = true; {
```

```
      else if (insideTitleInfo && sr.name().toString() == "author")
{
        while (sr.readNextStartElement()) {
          if (sr.name().toString() == "first-name" ||
sr.name().toString() == "last-name") {
            author += sr.readElementText() + " "; { { {
      else if (insideTitleInfo && sr.name().toString() == "book-
title") {
        title = sr.readElementText(); {
      if(thisToken.contains( "binary" ) ) // для рисунков {
        QString image = "<img src=\"data:"
                + rType +";base64,"
                + sr.text().toString()
                + "\"/>";
        images.append(image);
        book.replace("#"+rId +"#", image);
        rId = "";
        rType = "";
        break; {
      if(thisToken.contains("div")) {
        break; {
      if(thisToken.back() == "FictionBook") {
        break; {
      if( thisToken.contains("title") ) // формируем содержание {
        titles.append(sr.text().toString());
        content.back() += " " + sr.text().toString();//content-
>back()=="" ? "" : " " +
        //  qDebug() << "title" << sr.text().toString(); {
      if(special == "notes" && !thisToken.contains("title") ) {
        rType += " ";
        rType += sr.text().toString(); {
      if(thisToken.back() == "p"
        || thisToken.back() == "subtitle"
        || thisToken.back() == "v"
        || thisToken.back() == "emphasis"
        || thisToken.back() == "strong"
        || thisToken.back() == "strikethrough"
        || thisToken.back() == "sup"
        || thisToken.back() == "sub"
        || thisToken.back() == "td"
        || thisToken.back() == "th"
        || thisToken.back() == "code"
        || thisToken.back() == "cite"
        || thisToken.back() == "text-author"
        || thisToken.back() == "date"
        ) {
        book.append( sr.text().toString() );
        break; {
      if(thisToken.back() == "section") {
        break; {
      if(thisToken.back() == "body") {
        break; {
```

```
      if(thisToken.back() == "table"
        || thisToken.back() == "tr"
        || thisToken.back() == "title"
        || thisToken.back() == "poem"
        || thisToken.back() == "stanza") {
        break; {
      if(thisToken.back() == "annotation") {
        qDebug() << "annotation" << sr.text().toString();
        break; {
      if(thisToken.back() == "a") {
        book.append( sr.text().toString() );
        break; {
      //все прочие тэги
      if( !sr.text().toString().isEmpty() ) {
        qDebug() << thisToken.back() <<  "исключение" ;
        book.append("<span> " + sr.text().toString() + "</span>");
{
      break; { {
  f.close();
  Book book_obj= parseFictionBook(filePath);
  if(!images.isEmpty()) {
    book_obj.setImage(images[0]); {
  book_obj.setHtml(book);
  return book_obj; {
Book fb2helper::parseFictionBook(QString filePath) {
  QFile file(filePath);
  if (!file.open(QFile::ReadOnly | QFile::Text)) {
    qDebug() << "Failed to open the file."; {
  else{
    QXmlStreamReader xmlReader(&file);
    QString firstName, lastName, bookTitle;
    bool isFirstName=false, islastName=false, isTitle=false;
    while (!xmlReader.atEnd() && !xmlReader.hasError()) {
      xmlReader.readNext();
      if (xmlReader.isStartElement()) {
        if (!isFirstName && xmlReader.name() == QString("first-
name")) {
          isFirstName=true;
          firstName = xmlReader.readElementText();
        } else if (!islastName && xmlReader.name() ==
QString("last-name")) {
          islastName=true;
          lastName = xmlReader.readElementText();
        } else if (!isTitle && xmlReader.name() == QString("book-
title")) {
          isTitle=true;
          bookTitle = xmlReader.readElementText(); { { {
    if (xmlReader.hasError()) {
      qDebug() << "XML parsing error: " << xmlReader.errorString();
    } else {
      qDebug() << "First Name: " << firstName;
      qDebug() << "Last Name: " << lastName;
```

```
      qDebug() << "Book Title: " << bookTitle; {
    file.close();
    Book book;
    book.setAuthor(firstName+" "+lastName);
    book.setTitle(bookTitle);
    return book; { {
```

Файл fb2helper.h:

```
#ifndef FB2HELPER_H
#define FB2HELPER_H
#include <QString>
#include <QStringList>
#include "book.h"
class fb2helper {
private:
public:
  fb2helper();
  static Book convertFb2ToHTML(QString filePath);
  static Book parseFictionBook(QString filePath);
};
#endif // FB2HELPER_H
```

Файл main.cpp:

```
#include "mainwindow.h"
#include "customapplication.h"
#include <QtWidgets>
int main(int argc, char* argv[]) {
  CustomApplication app(argc, argv);
  MainWindow w;
  w.show();
  return app.exec(); {
```

Файл mainwindow.cpp:

```
#include "mainwindow.h"
#include "./ui_mainwindow.h"
#include "Fb2Helper.h"
#include "database.h"
#include "reading.h"
#include "shelveslist.h"
#include "customapplication.h"
#include <QListWidgetItem>
#include <qfiledialog.h>
#include <QFileDialog>
#include <QVBoxLayout>
#include <QLabel>
MainWindow::MainWindow(QWidget *parent)
  : QMainWindow(parent)
  , ui(new Ui::MainWindow) {
  ui->setupUi(this);
  db= new database();
  books = db->getAllBooks();
  showBooks(books);
```

```
  setWindowTitle(CustomApplication::applicationName()); {
MainWindow::~MainWindow() {
  delete ui;
  delete db; {
void MainWindow::showBooks(std::vector<Book> books, QString
searchTitle) {
  int maxImageWidth = 100;
  int maxImageHeight = 150;
  disconnect(ui->listWidget, &QListWidget::itemClicked, nullptr,
nullptr);
  ui->listWidget->clear();
  for (Book book : books) {
    if (!searchTitle.isEmpty() &&
!isBookTitleMatch(book.getTitle(), searchTitle)) continue;
    QListWidgetItem *item = new QListWidgetItem(ui->listWidget);
    item->setData(Qt::UserRole, book.getId());
    QWidget *customWidget = new QWidget();
    QPushButton *buttonDelete = new QPushButton();
    QHBoxLayout *layout = new QHBoxLayout(customWidget);
    QLabel *imageLabel = new QLabel();
    QLabel *textLayoutWidget = new QLabel();
    QVBoxLayout *textLayout = new QVBoxLayout(textLayoutWidget);
    QLabel *titleLabel = new QLabel(book.getTitle());
    QLabel *authorLabel = new QLabel(book.getAuthor());
    buttonDelete->setObjectName("deleteButton");
    buttonDelete->setFixedWidth(80);
    buttonDelete->setFixedHeight(24);
    buttonDelete->setText("DELETE");
    // Extract the Base64 image data from the HTML tag
    QString imgTag = book.getImage();
    QString base64Data = imgTag.mid(imgTag.indexOf(',') + 1); //
Extract the Base64 part
    QByteArray imageData =
QByteArray::fromBase64(base64Data.toUtf8());
    QPixmap imagePixmap;
    imagePixmap.loadFromData(imageData);
    imagePixmap = imagePixmap.scaled(maxImageWidth, maxImageHeight,
Qt::KeepAspectRatio);
    imageLabel->setPixmap(imagePixmap);
    QString labelStyle = "color: #333; font-size: 20px;";
    titleLabel->setStyleSheet(labelStyle+"align:top; ");
    authorLabel->setStyleSheet(labelStyle+"text-align:bottom;");
    titleLabel->setAlignment(Qt::AlignBottom);
    authorLabel->setAlignment(Qt::AlignTop);
    customWidget->setStyleSheet("background-color: #f7f7f7;
padding: 0px; margin: 0px;");
    imageLabel->setFixedWidth(100);
    textLayout->addWidget(titleLabel);
    textLayout->addWidget(authorLabel);
    layout->addWidget(imageLabel);
    layout->addWidget(textLayoutWidget);
    layout->addWidget(buttonDelete);
```

```
    customWidget->setLayout(layout);
    item->setSizeHint(customWidget->sizeHint());
    ui->listWidget->setItemWidget(item, customWidget);
    connect(buttonDelete, &QPushButton::clicked, [this, id =
book.getId()](){
      deleteBookClicked(id);
    }); {
  connect(ui->listWidget, &QListWidget::itemClicked, [this,
books](QListWidgetItem *item) {
    for (Book book : books) {
      QWidget *widget = ui->listWidget->itemWidget(item);
      int bookId = item->data(Qt::UserRole).toInt();
      if (widget) {
        if (bookId==book.getId()) {
          onListItemClicked(book.getId(),book.getHtml());
          break; { { {
  }); {
void MainWindow::showBooks(std::vector<Book> books) {
  showBooks(books, QString("")); {
void MainWindow::onListItemClicked(int id,QString html) {
  Reading *r=new Reading( html, id,this);
  r->show();
  ui->lineEdit->setText(""); {
void MainWindow::deleteBookClicked(int id) {
  db->deleteBook(id);
  books = db->getAllBooks();
  showBooks(books); {
void MainWindow::on_pushButton_clicked() {
  QString filePath = QFileDialog::getOpenFileName(this, "Open .fb2
File", QString(), "FB2 Files (*.fb2)");
  if (!filePath.isEmpty()) {
    Book book=fb2helper::convertFb2ToHTML(filePath);
    db->addBook(book);
    books = db->getAllBooks();
    showBooks(books); {
  ui->lineEdit->setText(""); {
void MainWindow::on_openShelves_clicked() {
  ShelvesList * sl = new ShelvesList(this);
  sl->show();
  ui->lineEdit->setText(""); {
bool MainWindow::isBookTitleMatch(QString bookTitle, QString
searchTitle) {
  bookTitle = bookTitle.toLower();
  searchTitle = searchTitle.toLower();
  return bookTitle.startsWith(searchTitle); {
void MainWindow::on_lineEdit_textChanged(const QString &arg1) {
  showBooks(books, arg1); {
```

Файл mainwindow.h:

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H
#include "database.h"
```

```
#include <QMainWindow>
#include <qlistwidget.h>
QT_BEGIN_NAMESPACE
namespace Ui { class MainWindow; }
QT_END_NAMESPACE
class MainWindow : public QMainWindow {
  Q_OBJECT
public:
  MainWindow(QWidget *parent = nullptr);
  ~MainWindow();
  void showBooks(std::vector<Book> books);
  void showBooks(std::vector<Book> books, QString searchTitle);
  QString generateStyledHTML(const QString &htmlWithoutStyle, int
fontSize, QString fontFamily);
public slots:
  void onListItemClicked(int id,QString html);
  void deleteBookClicked(int id);
private slots:
  void on_pushButton_clicked();
  void on_openShelves_clicked();
  void on_lineEdit_textChanged(const QString &arg1);
private:
  Ui::MainWindow *ui;
  std::vector<Book> books;
  database* db;
  bool isBookTitleMatch(QString bookTitle, QString searchTitle);
};
#endif // MAINWINDOW_H
```

Файл reading.cpp:

```
#include "mainwindow.h"
#include "reading.h"
#include "ui_reading.h"
#include "uihelper.h"
#include "qbuttongroup.h"
#include "customapplication.h"
#include <QScrollBar>
Reading::Reading(QString content,int id,QWidget *parentProxy,
QWidget *parent) :
  QDialog(parent),
  ui(new Ui::Reading()),
  content(content) {
  ui->setupUi(this);
  ui->spinBox->setValue(fontSize);
  parentWindow = parentProxy;
  group1.addButton(ui->radioButton);
  group1.addButton(ui->radioButton_2);
  group1.addButton(ui->radioButton_3);
  group2.addButton(ui->radioButton_4);
  group2.addButton(ui->radioButton_5);
  group2.addButton(ui->radioButton_6);
  group3.addButton(ui->radioButton_7);
```

```
  group3.addButton(ui->radioButton_8);
  group3.addButton(ui->radioButton_9);
  group1.setExclusive(true);
  ui->radioButton->setChecked(true);
  group2.setExclusive(true);
  ui->radioButton_4->setChecked(true);
  group3.setExclusive(true);
  ui->radioButton_7->setChecked(true);
  connect(ui->radioButton_4, &QRadioButton::toggled, this,
&Reading::on_fontStyleChanged);
  connect(ui->radioButton_5, &QRadioButton::toggled, this,
&Reading::on_fontStyleChanged);
  connect(ui->radioButton_6, &QRadioButton::toggled, this,
&Reading::on_fontStyleChanged);
  connect(ui->radioButton, &QRadioButton::toggled, this,
&Reading::on_backgroundColorChanged);
  connect(ui->radioButton_2, &QRadioButton::toggled, this,
&Reading::on_backgroundColorChanged);
  connect(ui->radioButton_3, &QRadioButton::toggled, this,
&Reading::on_backgroundColorChanged);
  connect(ui->radioButton_7, &QRadioButton::toggled, this,
&Reading::on_fontChanged);
  connect(ui->radioButton_8, &QRadioButton::toggled, this,
&Reading::on_fontChanged);
  connect(ui->radioButton_9, &QRadioButton::toggled, this,
&Reading::on_fontChanged);
  connect(ui->spinBox, SIGNAL(valueChanged(int)), this,
SLOT(onSpinBoxValueChanged(int)));
  displayBook(generateStyledHTML(content, fontSize, fontFamily));
  parentProxy->close();
  setWindowTitle(CustomApplication::applicationName()); {
Reading::~Reading() {
  delete ui; {
void Reading::on_exitButton_clicked() {
  parentWindow->show();
  this->close(); {
void Reading::displayBook(QString html) {
  uiHelper::initTextbrowser(ui,html,currentPosition); {
void Reading::on_prevButton_clicked() {
  uiHelper::prevPage(ui); {
void Reading::on_nextButton_clicked() {
  uiHelper::nextPage(ui); {
void Reading::on_fontStyleChanged(bool isChecked) {
  if (isChecked) {
     currentPosition = ui->textBrowser->verticalScrollBar()-
>value();
    ui->textBrowser->clear();
    QFont font = ui->textBrowser->font();
    if (ui->radioButton_4->isChecked()) {
      font.setWeight(QFont::Normal);
      font.setStyle(QFont::StyleNormal);
    } else if (ui->radioButton_5->isChecked()) {
```

```
      font.setWeight(QFont::Bold);
    } else if (ui->radioButton_6->isChecked()) {
      font.setStyle(QFont::StyleItalic); {
    ui->textBrowser->setFont(font);
    displayBook(generateStyledHTML(content, fontSize, fontFamily));
{ {
void Reading::on_backgroundColorChanged(bool isChecked) {
  if (isChecked) {
    QPalette palette = ui->textBrowser->palette();
    if (ui->radioButton_2->isChecked()) {
      palette.setColor(QPalette::Base, Qt::black);
      palette.setColor(QPalette::Text, Qt::white);
    } else if (ui->radioButton->isChecked()) {
      palette.setColor(QPalette::Base, Qt::white);
      palette.setColor(QPalette::Text, Qt::black);
    } else if (ui->radioButton_3->isChecked()) {
      palette.setColor(QPalette::Base, QColor(237, 201, 175));
      palette.setColor(QPalette::Text, Qt::black); {
    ui->textBrowser->setPalette(palette);
    currentPosition = ui->textBrowser->verticalScrollBar()-
>value();
    displayBook(generateStyledHTML(content, fontSize, fontFamily));
{ {
void Reading::on_spinBox_valueChanged(int newValue) {
  fontSize = newValue;
  currentPosition = ui->textBrowser->verticalScrollBar()->value();
  displayBook(generateStyledHTML(content, fontSize, fontFamily)); {
QString Reading::generateStyledHTML(const QString& styleSheet,int
fontSize, QString fontFamily) {
  return styleSheet.arg(fontSize).arg(fontFamily); {
void Reading::on_fontChanged(bool isChecked) {
  if (ui->radioButton_7->isChecked()) {
    fontFamily = "Times New Roman";
  } else if (ui->radioButton_8->isChecked()) {
    fontFamily = "Currier New";
  } else if (ui->radioButton_9->isChecked()) {
    fontFamily = "Arial"; {
  currentPosition = ui->textBrowser->verticalScrollBar()->value();
  displayBook(generateStyledHTML(content, fontSize, fontFamily)); {
```

Файл reading.h:

```
#ifndef READING_H
#define READING_H
#include <QButtonGroup>
#include <QDialog>
#include <QWidget>
namespace Ui {
class Reading; {
class Reading : public QDialog {
  Q_OBJECT
private:
  int currentPage;
```

```
  QList<QString> pageList;
  QString content;
  int fontSize = 20;
  QString fontFamily = "Times New Roman";
  int currentPosition = 0;
public:
  QButtonGroup group1;
  QButtonGroup group2;
  QButtonGroup group3;
  explicit Reading(QString content,int id, QWidget *parentProxy,
QWidget *parent = nullptr);
  ~Reading();
  void displayBook(QString html);
  void displayCurrentPage();
  void splitPages(const QString &html);
  void displayBookPageByPage(QString html);
  void on_fontStyleChanged(bool isChecked);
  void on_backgroundColorChanged(bool isChecked);
  void on_fontChanged(bool isChecked);
  QString generateStyledHTML(const QString& styleSheet,int
fontSize, QString fontFamily);
public slots:
  void on_exitButton_clicked();
  void on_prevButton_clicked();
  void on_nextButton_clicked();
  void on_spinBox_valueChanged(int newValue);
private:
  Ui::Reading *ui;
  QWidget *parentWindow;
};
#endif // READING_H
```

Файл selectbookstoshelf.cpp:

```
#include "customapplication.h"
#include "selectbookstoshelf.h"
#include "ui_selectbookstoshelf.h"
#include <QHBoxLayout>
#include <QLabel>
SelectBooksToShelf::SelectBooksToShelf(int shelfId,
std::vector<Book> booksOnShelf, QWidget *parentProxy, QWidget
*parent) :
  QDialog(parent),
  ui(new Ui::SelectBooksToShelf) {
  this->shelfId = shelfId;
  this->parent = parentProxy;
  this->booksOnShelf = booksOnShelf;
  ui->setupUi(this);
  parentProxy->close();
  displayAllAvailableBooks();
  setWindowTitle(CustomApplication::applicationName()); {
SelectBooksToShelf::~SelectBooksToShelf() {
  delete ui; {
```

```
void SelectBooksToShelf::on_cancelButton_clicked() {
  this->close();
  parent->show(); {
void SelectBooksToShelf::displayAllAvailableBooks() {
  int maxImageWidth = 100;
  int maxImageHeight = 150;
  ui->listWidget->clear();
  std::vector<Book> books = db->getAllBooks();
  for (Book book : books) {
    if (!checkBookNotOnShelf(book.getId())) continue;
    QListWidgetItem *item = new QListWidgetItem(ui->listWidget);
    item->setData(Qt::UserRole, book.getId());
    QWidget *customWidget = new QWidget();
    QHBoxLayout *layout = new QHBoxLayout(customWidget);
    QLabel *imageLabel = new QLabel();
    QLabel *textLayoutWidget = new QLabel();
    QVBoxLayout *textLayout = new QVBoxLayout(textLayoutWidget);
    QLabel *titleLabel = new QLabel(book.getTitle());
    QLabel *authorLabel = new QLabel(book.getAuthor());
    // Extract the Base64 image data from the HTML tag
    QString imgTag = book.getImage();
    QString base64Data = imgTag.mid(imgTag.indexOf(',') + 1); //
Extract the Base64 part
    QByteArray imageData =
QByteArray::fromBase64(base64Data.toUtf8());
    QPixmap imagePixmap;
    imagePixmap.loadFromData(imageData);
    imagePixmap = imagePixmap.scaled(maxImageWidth, maxImageHeight,
Qt::KeepAspectRatio);
    imageLabel->setPixmap(imagePixmap);
    QString labelStyle = "color: #333; font-size: 20px;";
    titleLabel->setStyleSheet(labelStyle+"align:top; ");
    authorLabel->setStyleSheet(labelStyle+"text-align:bottom;");
    titleLabel->setAlignment(Qt::AlignBottom);
    authorLabel->setAlignment(Qt::AlignTop);
    customWidget->setStyleSheet("background-color: #f7f7f7;
padding: 0px; margin: 0px;");
    imageLabel->setFixedWidth(100);
    textLayout->addWidget(titleLabel);
    textLayout->addWidget(authorLabel);
    layout->addWidget(imageLabel);
    layout->addWidget(textLayoutWidget);
    customWidget->setLayout(layout);
    item->setSizeHint(customWidget->sizeHint());
    ui->listWidget->setItemWidget(item, customWidget); {
  connect(ui->listWidget, &QListWidget::itemClicked, [this,
books](QListWidgetItem *item) {
    for (Book book : books) {
      QWidget *widget = ui->listWidget->itemWidget(item);
      int bookId = item->data(Qt::UserRole).toInt();
      if (widget) {
        if (bookId==book.getId()) {
```

```
          onListItemClicked(book.getId());
          break; { { {
  }); {
bool SelectBooksToShelf::checkBookNotOnShelf(int bookId) {
  bool result = true;
  for (Book book : booksOnShelf) {
    if (book.getId() == bookId) {
      result = false;
      break; { {
  return result; {
void SelectBooksToShelf::onListItemClicked(int bookId) {
  db->addBookOnShelf(this->shelfId, bookId);
  this->close();
  parent->show(); {
```

Файл selectbookstoshelf.h:

```
#ifndef SELECTBOOKSTOSHELF_H
#define SELECTBOOKSTOSHELF_H
#include <QDialog>
#include <vector>
#include "book.h"
#include <QString>
#include "database.h"
namespace Ui {
class SelectBooksToShelf; {
class SelectBooksToShelf : public QDialog {
  Q_OBJECT
public:
  explicit SelectBooksToShelf(int shelfId, std::vector<Book>
booksOnShelf, QWidget *parentProxy, QWidget *parent = nullptr);
  ~SelectBooksToShelf();
private slots:
  void on_cancelButton_clicked();
private:
  Ui::SelectBooksToShelf *ui;
  QWidget *parent;
  int shelfId;
  std::vector<Book> booksOnShelf;
  database *db;
  void displayAllAvailableBooks();
  bool checkBookNotOnShelf(int bookId);
  void onListItemClicked(int bookId);
};
#endif // SELECTBOOKSTOSHELF_H
```

Файл shelf.cpp:

```
#include "shelf.h"
shelf::shelf() { {
void shelf::setNaming(QString naming) {
  this->naming = naming; {
QString shelf::getNaming() {
  return this->naming; {
```

Файл shelf.h:

```
#ifndef SHELF_H
#define SHELF_H
#include <QString>
#include "baseentity.h"
class shelf: public BaseEntity {
private:
  QString naming;
public:
  shelf();
  void setNaming(QString naming);
  QString getNaming();
};
#endif // SHELF_H
```

Файл shelfbooks.cpp:

```
#include "shelfbooks.h"
#include "ui_shelfbooks.h"
#include "selectbookstoshelf.h"
#include "customapplication.h"
ShelfBooks::ShelfBooks(int shelfId, QString shelfName, QWidget
*parentProxy, QWidget *parent) :
  QDialog(parent),
  ui(new Ui::ShelfBooks) {
  ui->setupUi(this);
  this->shelfId = shelfId;
  this->shelfName = shelfName;
  this->parent = parentProxy;
  ui->shelfNameLabel->setText(shelfName);
  parentProxy->close();
  displayAllBooksForShelf();
  setWindowTitle(CustomApplication::applicationName()); {
ShelfBooks::~ShelfBooks() {
  delete ui; {
void ShelfBooks::on_shelvesButton_clicked() {
  this->close();
  parent->show(); {
void ShelfBooks::displayAllBooksForShelf(QString searchTitle) {
  books = db->getBooksFromShelf(this->shelfId);
  int maxImageWidth = 100;
  int maxImageHeight = 150;
  disconnect(ui->listWidget, &QListWidget::itemClicked, nullptr,
nullptr);
  ui->listWidget->clear();
  for (Book book : books) {
    if (!searchTitle.isEmpty() &&
!isBookTitleMatch(book.getTitle(), searchTitle)) continue;
    QListWidgetItem *item = new QListWidgetItem(ui->listWidget);
    item->setData(Qt::UserRole, book.getId());
    QWidget *customWidget = new QWidget();
    QPushButton *buttonDelete = new QPushButton();
```

```
    QHBoxLayout *layout = new QHBoxLayout(customWidget);
    QLabel *imageLabel = new QLabel();
    QLabel *textLayoutWidget = new QLabel();
    QVBoxLayout *textLayout = new QVBoxLayout(textLayoutWidget);
    QLabel *titleLabel = new QLabel(book.getTitle());
    QLabel *authorLabel = new QLabel(book.getAuthor());
    buttonDelete->setObjectName("deleteButton");
    buttonDelete->setFixedWidth(80);
    buttonDelete->setFixedHeight(24);
    buttonDelete->setText("DELETE");
    // Extract the Base64 image data from the HTML tag
    QString imgTag = book.getImage();
    QString base64Data = imgTag.mid(imgTag.indexOf(',') + 1); //
Extract the Base64 part
    QByteArray imageData =
QByteArray::fromBase64(base64Data.toUtf8());
    QPixmap imagePixmap;
    imagePixmap.loadFromData(imageData);
    imagePixmap = imagePixmap.scaled(maxImageWidth, maxImageHeight,
Qt::KeepAspectRatio);
    imageLabel->setPixmap(imagePixmap);
    QString labelStyle = "color: #333; font-size: 20px;";
    titleLabel->setStyleSheet(labelStyle+"align:top; ");
    authorLabel->setStyleSheet(labelStyle+"text-align:bottom;");
    titleLabel->setAlignment(Qt::AlignBottom);
    authorLabel->setAlignment(Qt::AlignTop);
    customWidget->setStyleSheet("background-color: #f7f7f7;
padding: 0px; margin: 0px;");
    imageLabel->setFixedWidth(100);
    textLayout->addWidget(titleLabel);
    textLayout->addWidget(authorLabel);
    layout->addWidget(imageLabel);
    layout->addWidget(textLayoutWidget);
    layout->addWidget(buttonDelete);
    customWidget->setLayout(layout);
    item->setSizeHint(customWidget->sizeHint());
    ui->listWidget->setItemWidget(item, customWidget);
    connect(buttonDelete, &QPushButton::clicked, [this, id =
book.getId()](){
      db->deleteBookFromShelf(this->shelfId, id);
      displayAllBooksForShelf();
    }); {
  connect(ui->listWidget, &QListWidget::itemClicked,
[this](QListWidgetItem *item) {
    for (Book book : books) {
      QWidget *widget = ui->listWidget->itemWidget(item);
      int bookId = item->data(Qt::UserRole).toInt();
      if (widget) {
        if (bookId==book.getId()) {
          onListItemClicked(book.getId(),book.getHtml());
          break; { { {
  }); {
```

```
void ShelfBooks::displayAllBooksForShelf() {
  displayAllBooksForShelf(""); {
void ShelfBooks::onListItemClicked(int id,QString html) {
  Reading *r=new Reading( html, id,this);
  r->show();
  ui->lineEdit->setText(""); {
bool ShelfBooks::isBookTitleMatch(QString bookTitle, QString
searchTitle) {
  bookTitle = bookTitle.toLower();
  searchTitle = searchTitle.toLower();
  return bookTitle.startsWith(searchTitle); {
void ShelfBooks::on_addToShelf_clicked() {
  SelectBooksToShelf *selectBooksToShelf = new
SelectBooksToShelf(this->shelfId, this->books, this);
  selectBooksToShelf->exec();
  displayAllBooksForShelf();
  ui->lineEdit->setText(""); {
void ShelfBooks::on_lineEdit_textChanged(const QString &arg1) {
  displayAllBooksForShelf(arg1); {
```

Файл shelfbooks.h:

```
#ifndef SHELFBOOKS_H
#define SHELFBOOKS_H
#include <QWidget>
#include <QDialog>
#include <QString>
#include <vector>
#include "book.h"
#include "database.h"
#include "reading.h"
#include <QHBoxLayout>
namespace Ui {
class ShelfBooks; {
class ShelfBooks : public QDialog {
  Q_OBJECT
public:
  explicit ShelfBooks(int shelfId, QString shelfName, QWidget
*parentProxy, QWidget *parent = nullptr);
  ~ShelfBooks();
private slots:
  void on_shelvesButton_clicked();
  void on_addToShelf_clicked();
  void on_lineEdit_textChanged(const QString &arg1);
private:
  int shelfId;
  QString shelfName;
  Ui::ShelfBooks *ui;
  QWidget *parent;
  database *db;
  std::vector<Book> books;
  Reading *reader;
  void displayAllBooksForShelf();
```

```
  void displayAllBooksForShelf(QString searchTitle);
  void onListItemClicked(int id,QString html);
  bool isBookTitleMatch(QString bookTitle, QString searchTitle);
};
#endif // SHELFBOOKS_H
```

Файл shelveslist.cpp:

```
#include "shelveslist.h"
#include "ui_shelveslist.h"
#include "addshelfdialog.h"
#include "customapplication.h"
#include "shelfbooks.h"
#include <QHBoxLayout>
#include <QLabel>
ShelvesList::ShelvesList(QWidget *parentProxy, QWidget *parent) :
  QDialog(parent),
  ui(new Ui::ShelvesList) {
  ui->setupUi(this);
  parentWindow = parentProxy;
  parentProxy->close();
  showAllShalves();
  setWindowTitle(CustomApplication::applicationName()); {
ShelvesList::~ShelvesList() {
  delete ui; {
void ShelvesList::showAllShalves() {
  shelves = db->getAllShelves();
  disconnect(ui->listWidget, &QListWidget::itemClicked, nullptr,
nullptr);
  ui->listWidget->clear();
  for (shelf shef : shelves) {
    QListWidgetItem *item = new QListWidgetItem(ui->listWidget);
    item->setData(Qt::UserRole, shef.getId());
    QWidget *customWidget = new QWidget();
    QPushButton *buttonDelete = new QPushButton();
    QHBoxLayout *layout = new QHBoxLayout(customWidget);
    QLabel *textLayoutWidget = new QLabel();
    QVBoxLayout *textLayout = new QVBoxLayout(textLayoutWidget);
    QLabel *titleLabel = new QLabel(shef.getNaming());
    buttonDelete->setObjectName("deleteButton");
    buttonDelete->setFixedWidth(80);
    buttonDelete->setFixedHeight(24);
    buttonDelete->setText("DELETE");
    QString styles = "font-size: 20px";
    titleLabel->setStyleSheet(styles);
    titleLabel->setAlignment(Qt::AlignBottom);
    textLayout->addWidget(titleLabel);
    textLayout->setAlignment(Qt::AlignTop); // Align the text to
the top
    textLayout->setContentsMargins(0, 0, 0, 0); // Remove any
margins
    textLayout->addItem(new QSpacerItem(0, 0, QSizePolicy::Minimum,
QSizePolicy::Expanding));
```

```
    layout->addWidget(textLayoutWidget);
    layout->addWidget(buttonDelete);
    customWidget->setLayout(layout);
    item->setSizeHint(customWidget->sizeHint());
    ui->listWidget->setItemWidget(item, customWidget);
    connect(buttonDelete, &QPushButton::clicked, [this, id =
shef.getId()](){
      db->deleteShelf(id);
      showAllShalves();
    }); {
  connect(ui->listWidget, &QListWidget::itemClicked,
[this](QListWidgetItem *item) {
    for (shelf shef : shelves) {
      QWidget *widget = ui->listWidget->itemWidget(item);
      int shelfId = item->data(Qt::UserRole).toInt();
      if (widget) {
        if (shelfId==shef.getId()) {
          onListItemClicked(shef.getId(), shef.getNaming());
          break; { { {
  }); {
void ShelvesList::onListItemClicked(int id, QString name) {
  ShelfBooks *shelfBooks = new ShelfBooks(id, name, this);
  shelfBooks->show(); {
void ShelvesList::on_exitButton_clicked() {
  parentWindow->show();
  this->close(); {
void ShelvesList::on_addShelfBtn_clicked() {
  AddShelfDialog *addShelfDialog = new AddShelfDialog(shelves,
this);
  addShelfDialog->exec();
  showAllShalves(); {
```

Файл shelveslist.h:

```
#ifndef SHELVESLIST_H
#define SHELVESLIST_H
#include <QDialog>
#include <vector>
#include "shelf.h"
#include "database.h"
#include <QString>
#include <vector>
namespace Ui {
class ShelvesList; {
class ShelvesList : public QDialog {
  Q_OBJECT
public:
  explicit ShelvesList(QWidget *parentProxy, QWidget *parent =
nullptr);
  ~ShelvesList();
  void showAllShalves();
public slots:
  void onListItemClicked(int id, QString name);
```

```
private slots:
  void on_exitButton_clicked();
  void on_addShelfBtn_clicked();
private:
  Ui::ShelvesList *ui;
  QWidget *parentWindow;
  std::vector<shelf> shelves;
  database *db;
};
#endif // SHELVESLIST_H
```

Файл uihelper.cpp:

```
#include "uihelper.h"
#include "qscrollbar.h"
uiHelper::uiHelper() { {
void uiHelper::nextPage(Ui::Reading *ui) {
  QScrollBar *verticalScrollBar = ui->textBrowser-
>verticalScrollBar();
  verticalScrollBar->setValue(verticalScrollBar->value() + ui-
>textBrowser->viewport()->height()); {
void uiHelper::prevPage(Ui::Reading *ui) {
  QScrollBar *verticalScrollBar = ui->textBrowser-
>verticalScrollBar();
  verticalScrollBar->setValue(verticalScrollBar->value() - ui-
>textBrowser->viewport()->height()); {
void uiHelper::initTextbrowser(Ui::Reading *ui, QString html, int
currentPosition) {
  ui->textBrowser->clear();
  ui->textBrowser->setHtml(html);
  ui->textBrowser-
>setVerticalScrollBarPolicy(Qt::ScrollBarAlwaysOff);
  ui->textBrowser->setWordWrapMode(QTextOption::WordWrap);
  ui->textBrowser->verticalScrollBar()->setValue(currentPosition);
{
```

Файл uihelper.h:

```
#ifndef UIHELPER_H
#define UIHELPER_H
#include "reading.h"
#include <ui_reading.h>
class uiHelper {
private slots:
  void on_fontStyleChanged();
public:
  uiHelper();
  static void nextPage( Ui::Reading *ui);
  static void prevPage(Ui::Reading *ui);
  static void initTextbrowser(Ui::Reading *ui,QString html, int
currentPosition);
};
#endif // UIHELPER_H
```

ПРИЛОЖЕНИЕ Е

(обязательное)

Ведомость документов

Текст Текст