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

ПРОГРАММА ДЛЯ ЧТЕНИЯ ЭЛЕКТОННЫХ КНИГ

БГУИР КП 1–400201.519 ПЗ

Студент: группы 250505,   
Панас А. А.

Руководитель: ассистент

Каф. ЭВМ, Марзалюк А. В.

Минск 2023

Учреждение образования

«Белорусский государственный университет информатики   
и радиоэлектроники»

Факультет компьютерных систем и сетей

УТВЕРЖДАЮ

Заведующий кафедрой ЭВМ

Б. В. Никульшин

(подпись)

–––––––––––––––––––– 2023 г.

ЗАДАНИЕ

по курсовому проектированию

Студенту Панас Анне Алексеевне  *––––––––––––––*

Тема проекта Программа для чтения электронных книг

2. Срок сдачи студентом законченного проекта––––*11 декабря 2023 г. –*

3. Исходные данные к проекту *save.txt (данные для начального заполнения таблицы электронных книг).*

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

1. Лист задания.

2. Введение.

3. Постановка задачи

4. Обзор литературы.

4.1. Обзор методов и алгоритмов решения поставленной задачи.

5. Функциональное проектирование.

5.1. Структура входных и выходных данных.

5.2. Разработка диаграммы классов.

5.3. Описание классов.

6. Разработка программных модулей.

6.1. Разработка схем алгоритмов (loadDocument() и paintEvent ()).

6.2. Разработка алгоритмов.

6.2.1. Разработка алгоритма loadDocument().

6.2.2. Разработка алгоритма paintEvent ().

7. Результаты работы.

8. Заключение

9. Литература

10. Приложения

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

*1. Диаграмма классов. ––––––––––––––––––––––––––––––*

*2. Схема алгоритма метода* loadDocument*()*

*3. Схема алгоритма метода paintEvent ()––––– ––––––––––––––––––––––*

*––––––––––––––––––––––––––––––––––––––––––––– –––––––––––––––––––––*

6. Консультант по проекту (с обозначением разделов проекта)

А. В. Марзалюк

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):

*Выбор задания. Разработка содержания пояснительной записки. Перечень графического материала – к 01.10.2023 5 %;*

*разделы 2, 3 – к 01.10.2023 10 %;*

*раздел 4, 5 – к 01.11.2023 25 %;*

*раздел 6 – к 01.12.2023 35 %;*

*разделы 7,8,9 – к 01.12.2023 10 %;*

*раздел 10 – к 11.12.2023 5%;*

*оформление пояснительной записки и   
графического материала – к 11.12.2023 10 %;*

*Защита курсового проекта с 21.12 по 28.12.2023г.*

РУКОВОДИТЕЛЬ А.В. Марзалюк

(подпись)

Задание принял к исполнению *А. А. Панас*

(дата и подпись студента)

**СОДЕРЖАНИЕ**

[ВВЕДЕНИЕ 5](#_Toc153938963)

[1 ПОСТАНОВКА ЗАДАЧИ 6](#_Toc153938964)

[2 ОБЗОР ЛИТЕРАТУРЫ 7](#_Toc153938965)

[2.1. ОБЗОР МЕТОДОВ И АЛГОРИТМОВ РЕШЕНИЯ ПОСТАВЛЕННОЙ ЗАДАЧИ 9](#_Toc153938966)

[3 ФУНКЦИОНАЛЬНОЕ ПРОЕКТИРОВАНИЕ 11](#_Toc153938967)

[3.1. СТРУКТУРА ВХОДНЫХ И ВЫХОДНЫХ ДАННЫХ 11](#_Toc153938968)

[3.2 . РАЗРАБОТКА ДИАГРАММЫ КЛАССОВ 11](#_Toc153938969)

[3.3 . ОПИСАНИЕ КЛАССОВ 11](#_Toc153938970)

[4 РАЗРАБОТКА ПРОГРАММНЫХ МОДУЛЕЙ 17](#_Toc153938971)

[4.1. РАЗРАБОТКА СХЕМ АЛГОРИТМОВ 17](#_Toc153938972)

[4.2. РАЗРАБОТКА АЛГОРИТМОВ (описание алгоритмов по шагам, для двух методов) 17](#_Toc153938973)

[4.2.1. РАЗРАБОТКА АЛГОРИТМА loadDocument() 17](#_Toc153938974)

[4.2.2. РАЗРАБОТКА АЛГОРИТМА paintEvent () 18](#_Toc153938975)

[5 РЕЗУЛЬТАТ РАБОТЫ ПРОГРАММЫ 19](#_Toc153938976)

[ЗАКЛЮЧЕНИЕ 22](#_Toc153938977)

[СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ 23](#_Toc153938978)

[ПРИЛОЖЕНИЕ А 24](#_Toc153938979)

[ПРИЛОЖЕНИЕ Б 49](#_Toc153938980)

[ПРИЛОЖЕНИЕ В 50](#_Toc153938981)

[ПРИЛОЖЕНИЕ Г 51](#_Toc153938982)

# ВВЕДЕНИЕ

В современном информационном обществе, где технологии занимают центральное место в повседневной жизни, электронные книги стали неотъемлемой частью литературного опыта. С появлением многочисленных электронных форматов книг и разнообразных устройств для их чтения, разработка эффективных программ для удобного взаимодействия с этими ресурсами стала важным направлением в области программной инженерии.

Данная курсовая работа посвящена исследованию и разработке программы для чтения электронных книг, реализованной с использованием языка программирования C++ и фреймворка Qt. Электронные книги представляют собой не только современный формат чтения, но и пространство для инноваций в области программного обеспечения. Программа, созданная на основе языка C++ и Qt, предоставляет уникальные возможности по оптимизации процесса восприятия литературного материала, а также обеспечивает высокий уровень пользовательского комфорта.

В работе будут рассмотрены основные аспекты разработки программы, включая архитектуру, основные функциональные возможности, а также технические детали, связанные с использованием языка C++ и фреймворка Qt. Кроме того, рассматривается влияние разработанной программы на улучшение процесса взаимодействия пользователя с электронными книгами и современными технологиями чтения.

Целью данной работы является не только создание эффективной программы для чтения электронных книг, но и изучение принципов проектирования и программирования, а также их применение в контексте разработки приложений, ориентированных на современные потребности читателей.

# 1 ПОСТАНОВКА ЗАДАЧИ

Необходим разработать программу, в которой будет реализован функционал взаимодействия с электронными книгами. Были выбраны возможности, которые будут выполнены в рамках данной курсового проекта:

- программа должна иметь удобный пользовательский интерфейс с необходимыми пунктами меню.

- программа должна уметь загружать электронные книги в цифровом формате EPUB (Electronic PUBlication).

- в программе должна быть предусмотрена возможность хранения информации о недавно загруженных из памяти устройства книгах.

- в программе должна быть реализована возможность создавать разделы для организации ранее загруженных в программу пользователем книг.

- в программе должны быть реализованы сохранение информации со списком добавленных книг в текстовый файл, и загрузка из существующего текстового файла нового списка с книгами.

Должна быть реализована проверка на ввод ошибочных значений.

# 2 ОБЗОР ЛИТЕРАТУРЫ

В эпоху цифровых технологий, где доступ к информации становится все более мгновенным и удобным, электронные книги стали неотъемлемой частью современного чтения. С появлением первых электронных компьютеров в 1970-х годах зародилась идея создания программ для чтения электронных книг, что открыло двери в новую эру литературного восприятия.

Первые программы для чтения электронных книг впервые появились во времена проекта Гутенберг в начале 1970-х, предлагая электронные версии классических произведений литературы. Однако, технологические ограничения того времени существенно ограничивали их функциональность. С развитием технологий, в середине 1990-х годов, на сцену вышли первые устройства для чтения электронных книг, сопровождаемые программами, предоставляющими базовые функции.

С началом 2000-х годов Microsoft Reader и Palm Reader внесли свой вклад, предлагая дополнительные возможности, такие как поддержка звука и многоуровневая масштабируемость. С появлением устройств Kindle и Nook в середине 2000-х годов произошел взрывной рост популярности электронных книг, требующий разработки более универсальных и функциональных программ для их чтения.

Это вступление подготавливает почву для более подробного рассмотрения эволюции программ для чтения электронных книг, технологий, используемых в этой области, и современных тенденций, которые формируют будущее цифрового чтения.

С развитием технологий форматы электронных книг также эволюционировали. В начале 2000-х годов форматы, такие как LIT и PDB, были предназначены для программ чтения на операционных системах Windows и Palm OS. Однако, они оказались неудовлетворительными из-за ограниченной поддержки и совместимости.

С середины 2000-х годов формат EPUB, предложенный Adobe Digital Editions, стал широко распространенным и открытым стандартом для электронных книг. EPUB обеспечивает адаптивную вёрстку, поддерживает разнообразные устройства и открывает новые возможности для создания интерактивных книг.

Основные технологии в программных средствах для чтения включают в себя работу с форматами файлов (EPUB, MOBI, PDF), поддержку DRM для защиты авторских прав, а также функциональности, такие как настраиваемые интерфейсы, подсветка текста, поддержка изображений и звука, адаптивная вёрстка и синхронизация между устройствами.

EPUB (Electronic Publication) стал стандартом для электронных книг благодаря своей открытой природе и поддержке разнообразных устройств. Формат позволяет создавать книги с динамическим контентом, обеспечивает удобство чтения на различных экранах и поддерживает структурированный контент.

Рассмотрим несколько известных программ, проведя сравнительный анализ их функционала, интерфейсов и возможностей, выделяя как положительные, так и негативные аспекты каждой из них.

**Adobe Digital Editions:** Положительные черты Adobe Digital Editions включают богатый функционал, поддержку формата EPUB и возможность работы с DRM-защитой. Однако, программу могут отличать относительно высокие требования к ресурсам системы, а также не всегда стабильная работа, особенно при больших библиотеках.

**Amazon Kindle:** Программа Amazon Kindle славится своим удобством использования, синхронизацией и широким выбором книг. Однако, жесткие ограничения в форматах файлов и зависимость от экосистемы Amazon могут быть недостатками для тех, кто предпочитает большую свободу выбора.

**FBReader:** FBReader привлекает своей легкостью и простотой. Недостатком может быть ограниченная функциональность в сравнении с более мощными приложениями. Также, программа может не предоставлять такого уровня поддержки как более крупные конкуренты.

**Calibre:** Calibre обладает мощным функционалом для управления библиотекой, но интерфейс программы может быть сложен для новичков. Некоторые пользователи также могут столкнуться с проблемами при конвертации некоторых форматов файлов.

**Sumatra PDF:** Sumatra PDF выделяется своей легкостью, но она не предоставляет таких расширенных функций, как многие другие программы. Она ориентирована в первую очередь на чтение PDF, и для электронных книг может быть слишком упрощенной.

**Moon+ Reader:** Позитивные черты Moon+ Reader включают обширные возможности настройки, но приложение ограничено платформой Android, что может быть недостатком для пользователей других операционных систем.

Каждая из этих программ имеет свои преимущества и недостатки, и выбор зависит от индивидуальных предпочтений и потребностей пользователя.

В настоящее время, современные программы для чтения электронных книг активно интегрируются с облачными сервисами, предоставляют персонализированные рекомендации, обеспечивают социальные функции, такие как обсуждение книг и обмен рецензиями. Многие из них также предлагают расширенные возможности для настройки внешнего вида страниц и взаимодействия с контентом.

В будущем можно ожидать углубление интеграции и взаимодействия с другими медийными форматами, расширение возможностей интерактивности (возможность встраивания видео, аудио, 3D-графики), а также улучшение технологий адаптивной вёрстки для создания более комфортного и персонализированного опыта чтения. Возможно, развитие искусственного интеллекта также будет направлено на создание более интеллектуальных систем рекомендаций и аналитики предпочтений читателей.

## 2.1. ОБЗОР МЕТОДОВ И АЛГОРИТМОВ РЕШЕНИЯ ПОСТАВЛЕННОЙ ЗАДАЧИ

Задачу написания программы для чтения книг можно разбить на 4 составляющие части: реализация графического интерфейса, реализация функций сохранения и загрузки данных, организация добавленных пользователем книг и категорий в виде таблицы и отображение материала электронных книг в оконном виде.

Сохранение и загрузка данных из памяти устройства в память программы осуществлены путём записи информации в текстовый файл.

При нажатии пользователем кнопки «Save» на главном экране программы происходит создание текстового файла, в который записываются текущие значения полей таблицы. Поля включают в себя название книги, категорию, к которой принадлежит книга и путь к оригинальному файлу. Чтение из файла реализовано схожим образом. При нажатии кнопки «Upload table» пользователь может выбрать текстовый файл, содержащий данные о ранее созданной таблице книг. Далее информация о названии, категории и пути к файлу считывается в таблицу программы и может быть использована дальше. Поскольку программа имеет возможность создавать и присваивать книгам различные категории, то при чтении файла из памяти устройства названия категорий заносятся в поле со списком для возможности редактирования категорий книг.

Добавление информации о книгах пользователя в таблицу для удобного просмотра и редактирования данных. Пользователь загружает файл с помощью кнопки «Upload table», вводит желаемое название книги в строку «Book name» и загружает данные в таблицу в главном окне программы. В таблице пользователю предоставляется возможность сортировки данных в алфавитном порядке. По умолчанию программа устанавливает новой книге категорию «undecided». Для дальнейшей смены категории пользователь вводит в строку «Shelve name» требуемое имя категории и, если категория была создана раньше, присваивает ее книге, название которой прописано в строке «Book name». Если категория новая для программы, то она записывается в поле списка «Shelves» для дальнейшей возможности ее использования программой.

Для того, чтобы отобразить электронную книгу в оконном виде пользователь должен ввести название книги в строку «Book name» и добавить ее в таблицу. Далее программа считывает путь к файлу и отображает ее на экран.

Для обработки структуры zip-архива файлов EPUB в данном проекте была подключена сторонняя библиотека KArchive. Она играет решающую роль в этом проекте для управления структурой zip-архива файлов EPUB, позволяя программе эффективно получать доступ к содержимому книг EPUB и обрабатывать его.

Файл EPUB содержит различные компоненты, такие как файлы метаданных, файлы содержимого, изображения и т. д. Библиотека позволяет приложению читать эти компоненты из архива.

Также библиотека KArchive включает механизмы обработки ошибок во время операций архивирования. Это позволяет приложению адекватно реагировать в случае таких проблем, как не найденный файл, повреждение и др.

Для анализа и извлечения контента из EPUB файлов используются классы EPubContainer, EPubDocument. Для отображения и организации в удобном пользовательском интерфейсе используется класс Widget, наследуемый от стандартного класса QDialog.

# 3 ФУНКЦИОНАЛЬНОЕ ПРОЕКТИРОВАНИЕ

## 3.1. СТРУКТУРА ВХОДНЫХ И ВЫХОДНЫХ ДАННЫХ

В программе имеется возможность загружать сохраненный список электронных книг путем загрузки сгенерированного ранее программой файла формата .txt.

Структура входных данных, записываемых в файл, представлена в таблице 3.1.1

Таблица 3.1.1 – Пример входных данных

|  |  |  |
| --- | --- | --- |
| Название книги | Категория | Путь к файлу |
| Atomic habits | Psychology | /Desktop/books/Atomic\_Habits.pdf |
| Norsk grammatikk | Norsk | /Desktop/books/grammatikk-naring-begynner.epub |
| Pa vei | Norsk | /Desktop/books/p-vei-tekstbok-a1-og-a2.epub |
| Harry Potter Stone | Harry Potter | /Desktop/books/harry-potter-book-1.epub |
| Harry Potter Half-blood Prince | Harry Potter | /Desktop/books/HP-half-blood-prince.epub |
| 13 reasons | Novel | /Desktop/books/13-reasons-why.epub |
| The power of habit | Psychology | /Desktop/books/the-power-of-habit.epub |

Также присутствует возможность сохранять новый или модифицированный список электронных книг. Выходные данные сохраняются в текстовом формате .txt со структурой, представленной в таблице 3.1.1

## . РАЗРАБОТКА ДИАГРАММЫ КЛАССОВ

Одним из важнейших пунктов в ООП является разработка логической модели системы в виде диаграммы классов. Диаграмма классов для представления статической структуры модели в терминологии классов ООП. Она отражает связи между классами и структурами.

Диаграмма классов программы представлена в приложении Б.

## . ОПИСАНИЕ КЛАССОВ

* + 1. **MainWindow**

Класс наследуется от стандартного библиотечного класса QMainWindow. Данный класс является графическим представлением программы, он представляет собой окно, в котором происходит работа в приложении.

class MainWindow : public QMainWindow

{

Q\_OBJECT — макрос, который объявляет свои собственные сигналы и слоты или использует другие сервисы, предоставляемые метаобъектной системой Qt

public:

MainWindow(QWidget \*parent = nullptr); — конструктор главного окна

~MainWindow(); — деструктор

private slots:

void on\_createShelveBtn\_clicked(); — добавление новой категории в таблицу.

void on\_deleteShelveBtn\_clicked(); — удаления категории из таблицы.

void on\_uploadFileBtn\_clicked(); — получение пути к файлу через файловый менеджер.

void on\_addBookBtn\_clicked(); — добавление книги в строку таблицы.

void on\_deleteBookBtn\_clicked(); — удаление книги из списка

void on\_saveTableBtn\_clicked(); — сохранение информации о таблице в текстовом файле.

void on\_uploadTableBtn\_clicked(); — загрузка файла сохранения в таблицу.

void on\_setShelveBtn\_clicked(); — установка категории книги.

void on\_openBookBtn\_clicked(); — открытие книги в графическом виде.

private:

Ui::MainWindow \*ui; — переменная-член для взаимодействия с компонентами пользовательского интерфейса.

};

* + 1. **EPubContainer**

Класс наследуется от стандартного библиотечного класса QObject.

Он инкапсулирует логику, необходимую для навигации, извлечения информации и содержимого из файлов EPUB, что упрощает интеграцию поддержки EPUB в приложение.

class EPubContainer : public QObject

{

Q\_OBJECT

public:

explicit EPubContainer(QObject \*parent); — конструктор класса EPubContainer.

~EPubContainer(); — деструктор класса EPubContainer.

bool openFile(const QString path); — открывает файл EPUB, указанный по заданному пути.

EpubItem getEpubItem(const QString &id) const ; — возвращает значение элемента по указаному идентификатору.

QSharedPointer<QIODevice> getIoDevice(const QString &path); — извлекает QIODevice по указанному пути к файлу в архиве EPUB.

QImage getImage(const QString &id); — извлекает изображение из файла EPUB, используя указанный идентификатор.

QString getMetadata(const QString &key); — извлекает информацию метаданных на основе предоставленного ключа.

QStringList getItems();— возвращает значение переменной m\_orderedItems.

QString getStandardPage(EpubPageReference::StandardType type) — возвращает значение переменной m\_standardReferences.

signals:

void errorHappened(const QString &error); — сигнал для выдачи сообщений об ошибках.

private:

bool parseMimetype();— анализирует файл mimetype, чтобы убедиться, что это файл EPUB.

bool parseContainer();—анализирует файл контейнера, чтобы получить путь к корневому файлу.

bool parseContentFile(const QString filepath); — анализирует файл содержимого для извлечения метаданных, элементов, порядка spine и справочной информации.

bool parseMetadataItem(const QDomNode &metadataNode); — анализирует элементы метаданных из раздела метаданных файла содержимого.

bool parseManifestItem(const QDomNode &manifestNodes, const QString currentFolder); — анализирует элементы из раздела манифеста файла содержимого.

bool parseSpineItem(const QDomNode &spineNode); — анализирует элементы из раздела spine файла содержимого.

bool parseGuideItem(const QDomNode &guideItem); — анализирует элементы руководства, чтобы получить ссылки на стандартные страницы EPUB.

const KArchiveFile \*getFile(const QString &path); — получает указатель на KArchiveFile для указанного пути к файлу.

KZip \*m\_archive; — представляет архив KZip файла EPUB.

const KArchiveDirectory \*m\_rootFolder; — представляет корневую папку в архиве EPUB.

QHash<QString, QString> m\_metadata; — хранит пары ключ-значение метаданных, извлеченные из файла EPUB.

QHash<QString, EpubItem> m\_items; — сохраняет информацию об файлах EPUB в архиве.

QStringList m\_orderedItems; — представляет упорядоченный список элементов согласно порядку spine.

QSet<QString> m\_unorderedItems; — хранит идентификаторы элементов, не перечисленных в spine.

QHash<EpubPageReference::StandardType, EpubPageReference> m\_standardReferences; — хранит ссылки на стандартные страницы EPUB.

QHash<QString, EpubPageReference> m\_otherReferences; — хранит ссылки на нестандартные страницы EPUB.

QMimeDatabase m\_mimeDatabase; — предоставляет информацию о типе MIME.

};

* + 1. **EPubDocument**

Класс наследуется от стандартного библиотечного класса QTextDocument.

Предназначен для управления представлением и рендерингом документов EPUB, предоставляющий функциональные возможности для загрузки, обработки и отображения содержимого файлов EPUB.

class EPubDocument : public QTextDocument

{

Q\_OBJECT

public:

explicit EPubDocument(QObject \*parent); — конструктор класса EPubDocument.

virtual ~EPubDocument();— деструктор класса EPubDocument.

bool loaded() — проверка, загружен ли документ.

void openDocument(const QString &path); — открытие документа EPUB по его пути.

void clearCache(); — очистка кеша отрендереных SVG файлов.

signals:

void loadCompleted();— сигнал, подаваемый при завершении загрузки документа.

protected:

virtual QVariant loadResource(int type, const QUrl &url) override; — загрузка ресурса по указанному URL-адресу.

private slots:

void loadDocument(); — слот для загрузки документов.

private:

void fixImages(QDomDocument &newDocument); — вспомогательная функция для исправления изображений в документе DOM.

const QImage &getSvgImage(const QString &id); — получить визуализированное изображение для данного идентификатора SVG.

QHash<QString, QByteArray> m\_svgs; — хэш-таблицы для хранения данных SVG.

QHash<QString, QImage> m\_renderedSvgs; — хэш-таблицы для хранения визуализированных изображений.

QString m\_documentPath; — путь к документу EPUB

EPubContainer \*m\_container; — указатель на EPubContainer, обрабатывающий операции с файлами EPUB.

EpubItem m\_currentItem; — текущий обрабатываемый элемент EPUB.

QList<int> m\_loadedFonts; — cписок загруженных шрифтов.

QSizeF m\_docSize; — размер документа.

bool m\_loaded; — флаг, указывающий, загружен ли документ.

};

* + 1. **Widget**

Класс наследуется от стандартного библиотечного класса QDialog.

Предназначен для просмотра документов EPUB, предоставляя функциональные возможности для загрузки и отображения содержимого EPUB, прокрутки страниц и обработки ключевых событий для взаимодействия с пользователем.

class Widget : public QDialog

{

Q\_OBJECT

public:

Widget(QWidget \*parent = 0); — конструктор класса Widget.

~Widget(); — деструктор класса Widget.

void scroll(int amount); — прокручивание контент на указанную величину.

void scrollPage(int amount); — прокручивание контент на указанное количество страниц.

bool loadFile(const QString &path, const QString bookName); — загрузка файла EPUB и установка названия книги.

protected:

void paintEvent(QPaintEvent \*event) override; — управление рисованием в виджете.

void keyPressEvent(QKeyEvent \*event) override; — обработка нажатий клавиш.

void resizeEvent(QResizeEvent \*event) override; — обработка событий изменения размера виджета

private:

QImage m\_cover; — изображение, используемое для обложки.

EPubDocument \*m\_document; — указатель на EPubDocument, обрабатывающий содержимое EPUB.

int m\_currentChapter; — индекс текущей главы.

int m\_yOffset; — вертикальное смещение для прокрутки.

};

# 4 РАЗРАБОТКА ПРОГРАММНЫХ МОДУЛЕЙ

## 4.1. РАЗРАБОТКА СХЕМ АЛГОРИТМОВ

Схемы методов loadDocument() и paintEvent () представлены в приложениях В и Г соответственно.

## 4.2. РАЗРАБОТКА АЛГОРИТМОВ

### 4.2.1. РАЗРАБОТКА АЛГОРИТМА loadDocument()

Описание алгоритма loadDocument () по шагам:

1. Начало
2. Создаем и начинаем таймер timer для измерения времени выполнения функции.
3. Экземпляр класса EPubContainer m\_container создается для обработки операций, специфичных для EPUB.
4. Открываем файл EPUB из m\_container с помощью EPubContainer.
5. Если файл не открывается, то переходим к пункту 28, иначе переходим к пункту 6.
6. Переменной QStringList присваиваем список элементов (глав) в документе m\_container.
7. Переменной cover присваиваем значение полученной методом getStandardPage () титульной страницы.
8. Если переменная cover пустая, то переходим к пункту 28, иначе переходим в пункту 9.
9. Добавляем cover в список элементов items.
10. Создаем QTextCursor для управления содержимым QTextDocument.
11. Настраиваем формат разрыва страницы функцией setPageBreakPolicy (), чтобы каждая глава начиналась с новой страницы.
12. Начало цикла. Цикл работает, пока список элементов items не станет пустым.
13. Для каждого элемента связанный EPubItem извлекается с помощью m\_container->getEpubItem(chapter).
14. Если текущая глава m\_currentItem пустая, то переходим к пункту 13, иначе переходим к пункту 15.
15. Получаем QSharedPointer для устройства ввода-вывода для текущей главы m\_currentItem.
16. Если устройство ввода-вывода ioDevice не действительно, то переходим к пункту 13, иначе к пункту17.
17. Разбераем содержимое главы с помощью QDomDocument.
18. Устанавливаем базовый URL-адрес для разрешения относительных URL-адресов для текущей главы m\_currentItem.
19. Исправляем изображения в главе с помощью метода fixImages.
20. Если цикл не закончился, переходим к пункту 13, иначе к пункту 21.
21. Издаем сигнал loadCompleted(), указывающий, что загрузка документа завершена.
22. Устанавливаем ширину текста в зависимости от макета документа. mw = fm.horizontalAdvance(QLatin1Char('x')) \* 80;
23. Отрегулируем ширину текста в зависимости от размера документа.
24. Если size.width() != 0,тогда присваиваем переменной w значение qSqrt((uint)(5 \* size.height() \* size.width() / 3)), иначе рассчитываем идеальную ширину текста в переменную w.
25. Устанавливаем идеальную ширину функцией setTextWidth(w);
26. Завершаем блок редактирования QTextCursor функцией endEditBlock ().
27. Отмечаем документ m\_loaded как загруженный.
28. Конец.

### 4.2.2. РАЗРАБОТКА АЛГОРИТМА paintEvent ()

Описание алгоритма parseContainer () по шагам:

1. Начало.
2. Создаем объект класса QPainter painter для рисования на виджете.
3. Заполняем весь виджет белым фоном, используя fillRect метод.
4. Если m\_document не загрузился, то переходим к пункту 13, иначе переходим к пункту 5.
5. Устанавливаем перевод paintContext на основе вертикального смещения, paintContext.clip.translate(0, m\_yOffset);
6. Устанавливаем палитру для рисования paintContext.palette = palette();
7. Начало цикла по int group = 0. Цикл работает пока group < 3. Шаг равен 1.
8. Перебирайте цветовые группы (обычные, отключенные и неактивные) и задавайте цвета текста и фона для каждой группы. Это сделано для обработки изменений цвета в зависимости от состояния виджета.
9. Если цикл не закончился, переходим к пункту 8, иначе к пункту 10.
10. Переводим рисовальщик painter на основе вертикального смещения translate(0, -m\_yOffset);
11. Устанавливаем прямоугольную рамку для ограничения области видимости setClipRect(paintContext.clip);
12. Рисуем макет документа m\_document с помощью функции рисования макета документа documentLayout () ->draw(&painter, paintContext);.
13. Конец.

# 5 РЕЗУЛЬТАТ РАБОТЫ ПРОГРАММЫ

При первом запуске программы перед вами появится главное окно программы (см. рисунок 5.1)

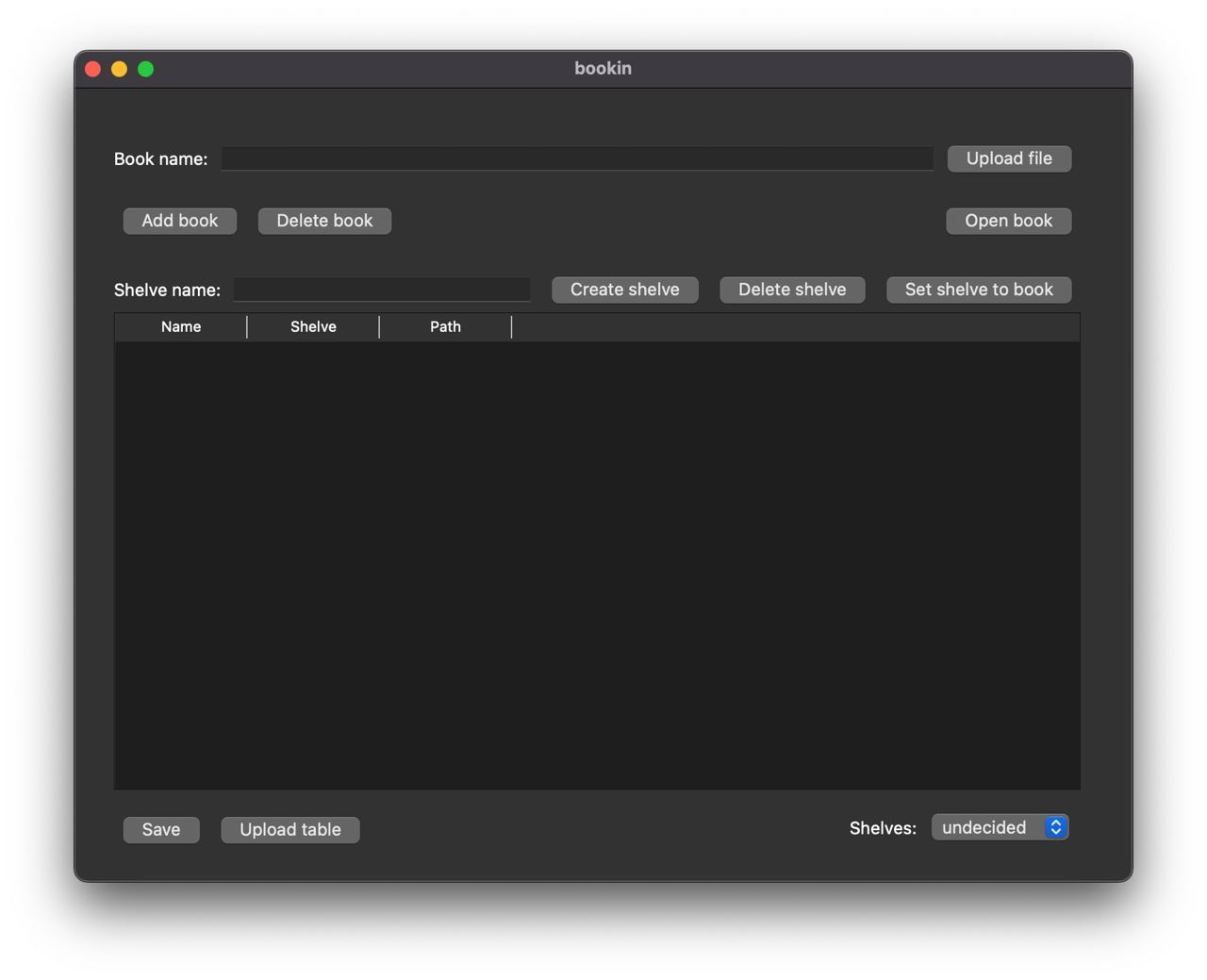


Рисунок 5.1 – Главное окно приложения

Для того, чтобы открыть электронную книгу, сначала нужно воспользоваться кнопкой Upload file и выбрать желаемый документ из памяти устройства. Затем, в текстовом поле Book name требуется указать имя для выбранной книги (данное имя будет использоваться для дальнейшего взаимодействия с документом в программе). Последним шагом является нажатие на кнопку Add book. В итоге выбранная книга, установленная по умолчанию категория undecided и абсолютный путь к документу будут указаны в таблице в нижней части программы (см. рисунок 5.2)

В программе возможно присваивание занесенным в таблицу книгам пользовательских категорий. Для этого в текстовом поле Shelve name нужно указать название желаемой категории и нажать на кнопку Create shelve. Затем требуется указать имя книги в текстовом поле Book name и нажать на кнопку Set shelve to book. Если категория была использована ранее, то шаг с использованием кнопки Create shelve следует пропустить.  
 Список всех существующих категорий находится в нижней правой части программы в динамическом списке Shelves.  
 Если требуется удалить категорию, то для этого в текстовом поле Shelve name нужно указать название желаемой категории, затем нажать на кнопку Delete shelve. Если категория была присвоена какому-либо элементу из таблицы, то после удаления категория книги изменится на undecided.

Для удаления поля из таблицы, требуется указать имя книги в текстовом поле Book name и нажать на кнопку Delete book.

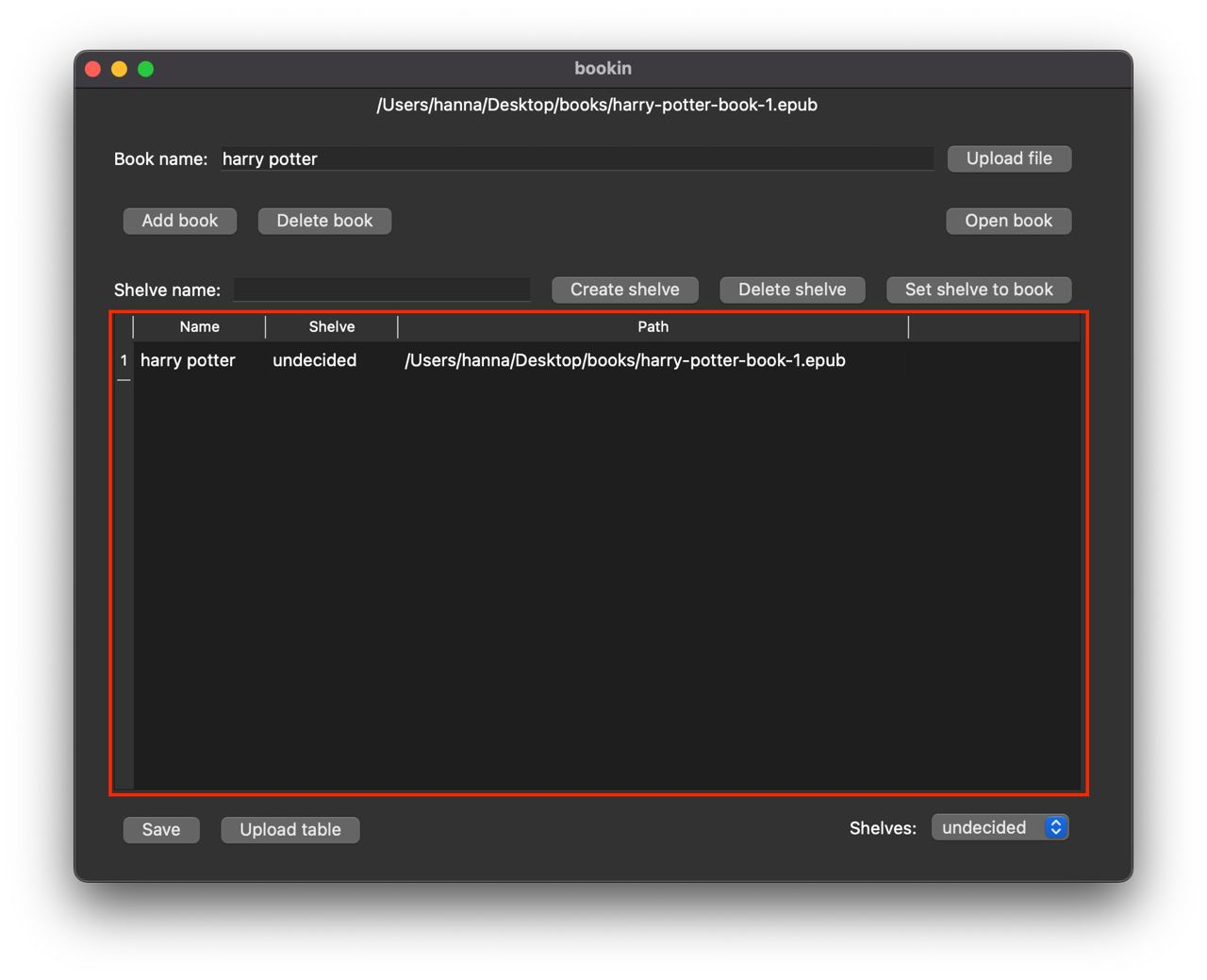


Рисунок 5.2 – Добавление книги в список

В программе реализовано сохранение текущего состояния таблицы в виде текстового документа в корневой папке проекта “Saves”. Для сохранения нужно нажать на кнопку Save. Таким образом будет создан файл сохранения, в названии включающий дату и время сохранения.  
 Для загрузки данных из файла сохранения следует нажать кнопку Upload table и выбрать путь к файлу сохранения. После чего в программе обновится список книг и загрузятся все присвоенные книгам категории.

Для того, чтобы открыть документ для чтения, необходимо указать в текстовом поле Book name имя книги, предварительно занесенной в список, и нажать на кнопку Open book. В результате на экране появится графическое отображение электронной книги (см. рисунок 5.3).

Навигация по файлу осуществляется с помощью стрелок на клавиатуре.  
 Клавиши вверх и вниз позволяют перемещаться по книге на 50 пикселей вверх (стрелка вверх) или вниз (стрелка вниз) по содержимому книги.

Клавиши влево и вправо дают доступ на перемещение по содержимому книги на целую страницу. Где стрелка влево означает переход на предыдущую страницу, а стрелка вправо переходит на следующую страницу.

Для выхода из просмотра книги можно нажать на крестик в углу окна или на клавишу Esc на клавиатуре.

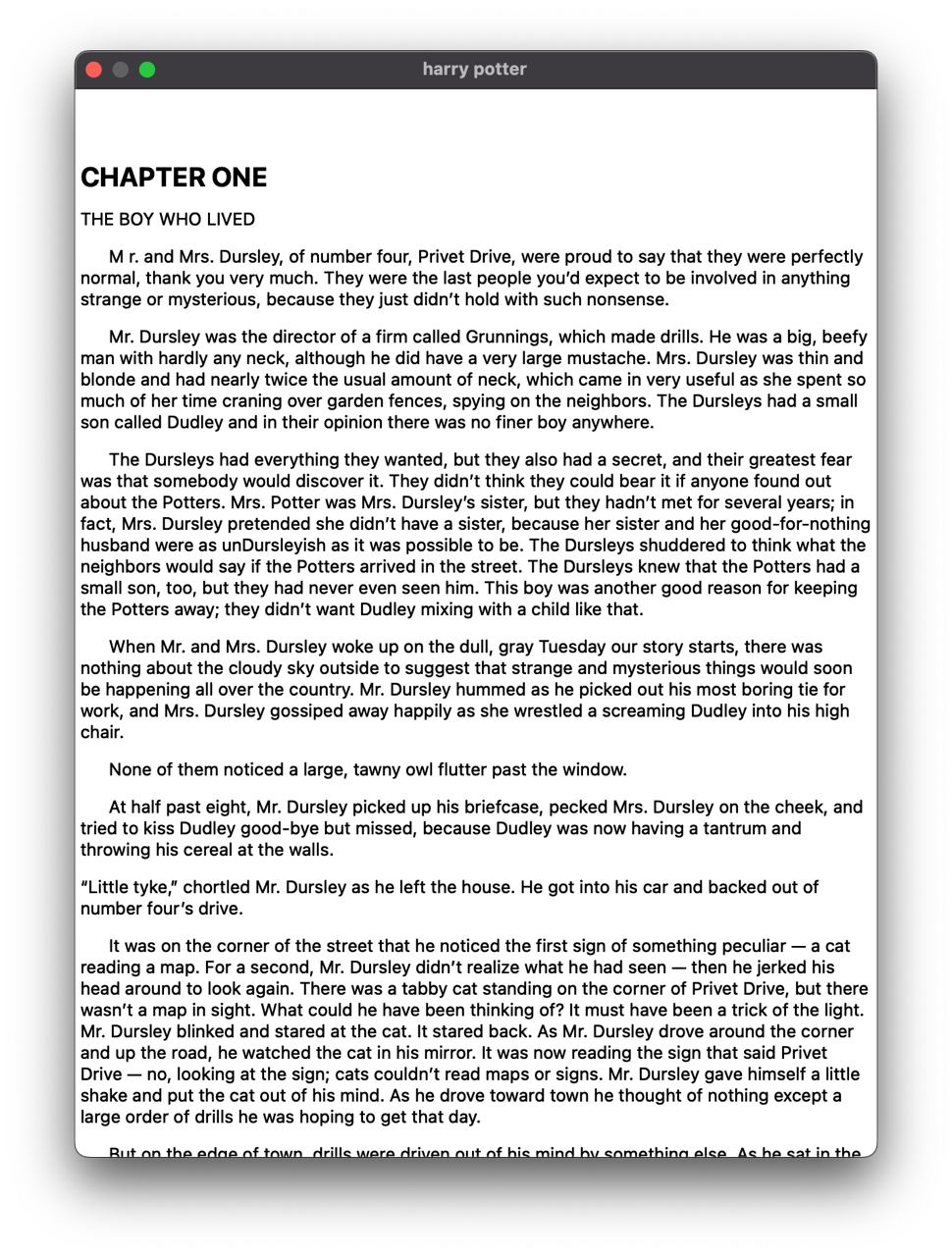


Рисунок 5.3 – Отображение электронной книги

# ЗАКЛЮЧЕНИЕ

В результате проделанной работы была разработана программа для чтения электронных книг, удовлетворяющая условиям, описанным в разделе 1. Проект был отлажен и задокументирован.

Полученный графический редактор является приложением, которое можно переносить на устройства, работающие на macOS версии 10.12 и выше.

Были получены знания о том, какие существуют подходы к созданию ПО, и как заниматься разработкой последнего максимально эффективно.

В процессе разработке также были углублены знания языка С++ и его расширения для фреймворка Qt Creator, что является очень важным и существенным опытом, на который можно будет опираться в дальнейшем.

Благодаря соблюдению стандартов написания кода и хорошо построенной логике программы ее дальнейшее расширение не должно вызывать особых проблем.

Итоговый продукт получился нетребовательным, однако скорость генерации визуализации электронной книги напрямую зависит от количества изображений в документе и аппаратного обеспечения пользователя.

В перспективе, планируется реализовать поддержку иных форматов электронных книг, добавить возможность изменять размер шрифта, имплементировать пролистывание книги с помощью колеса мыши и трекпада, также планируется улучшение пользовательского интерфейса.

# **СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ**

1. Б. Страуструп, Программирование: принципы и практики с использованием С++ / Б. Страуструп, Москва:  2016. – 1329 c.
2. В.Т. Фисенко, Т.Ю. Фисенко Компьютерная обработка и распознавание изображений / В.Т. Фисенко, Т.Ю. Фисенко, Санкт-Петербург: 2008. – **195** c.
3. А. В. Бушкевич, А. М. Ковальчук, И. В. Лукьянова. Конструированиепрограмм и языки программирования: метод. указания по курсовому проектированию для студ. спец. I-40 02 01 «Вычислительные машины, системы и сети» всех форм обуч. / сост. А. В. Бушкевич, А. М. Ковальчук, И. В. Лукьянова. – Минск: БГУИР, 2009. – 30 с.: ил.
4. Макс Шелле, Qt 5.3 Профессиональное программирование на С++ / М. Шелле, Санкт-Петербург: 2015 – 929с.
5. Рожнова Н.Г. , Вычислительные машины системы и сети. Дипломное проектирование: учебно-методическое пособие / Н.Г. Рожнова, Н.А. Искра, И.И. Глецевич. – Минск: БГУИР, 2014 – 96с.
6. Qt Documentaion [Электронный ресурс]. – Электронные данные. – Режим доступа: http://doc.qt.io/

# ПРИЛОЖЕНИЕ А

*(обязательное)*

Код программы

Файл data.cpp

#include "data.h"

void saveTableContents(QTableWidget\* tableWidget, const QString& fileName){

QFile file(fileName);

if (!file.open(QIODevice::WriteOnly | QIODevice::Text))

return;

QTextStream out(&file);

for (int row = 0; row < tableWidget->rowCount(); ++row)

{

for (int column = 0; column < tableWidget->columnCount(); ++column)

{

QTableWidgetItem\* item = tableWidget->item(row, column);

if (item)

out << item->text();

out << "\t";

}

out << "\n";

}

file.close();

}

void loadTableContents(QTableWidget\* tableWidget, QComboBox\* comboWidget, const QString& fileName){

QFile file(fileName);

if (!file.open(QIODevice::ReadOnly | QIODevice::Text))

return;

tableWidget->setRowCount(0) ;

tableWidget->clear() ;

QTextStream in(&file);

while (!in.atEnd())

{

QString line = in.readLine();

QStringList fields = line.split("\t");

int row = tableWidget->rowCount();

tableWidget->insertRow(row);

for (int column = 0; column < fields.size(); ++column)

{

QTableWidgetItem\* item = new QTableWidgetItem(fields[column]);

tableWidget->setItem(row, column, item);

}

}

for (int i = 0; i < tableWidget->rowCount(); i++) {

QString givenShelve = tableWidget->item(i, 1)->text() ;

if (comboWidget->findText(givenShelve) == -1) {

comboWidget->addItem(givenShelve);

}

}

file.close();

}

Файл epubcontainer.cpp

#include "epubcontainer.h"

#include <KZip>

#include <KArchiveDirectory>

#include <KArchiveFile>

#include <QDebug>

#include <QScopedPointer>

#include <QDomDocument>

#include <QDir>

#include <QImage>

#include <QImageReader>

#define METADATA\_FOLDER "META-INF"

#define MIMETYPE\_FILE "mimetype"

#define CONTAINER\_FILE "META-INF/container.xml"

EPubContainer::EPubContainer(QObject \*parent) : QObject(parent),

m\_archive(nullptr),

m\_rootFolder(nullptr)

{

}

EPubContainer::~EPubContainer()

{

delete m\_archive;

}

bool EPubContainer::openFile(const QString path)

{

delete m\_archive;

m\_archive = new KZip(path);

if (!m\_archive->open(QIODevice::ReadOnly)) {

emit errorHappened(tr("Failed to open(readonly) %1").arg(path));

return false;

}

m\_rootFolder = m\_archive->directory();

if (!m\_rootFolder) {

emit errorHappened(tr("Failed to read %1").arg(path));

return false;

}

if (!parseMimetype()) {

return false;

}

if (!parseContainer()) {

return false;

}

return true;

}

QSharedPointer<QIODevice> EPubContainer::getIoDevice(const QString &path)

{

const KArchiveFile \*file = getFile(path);

if (!file) {

emit errorHappened(tr("Unable to open file %1").arg(path.left(100)));

return QSharedPointer<QIODevice>();

}

return QSharedPointer<QIODevice>(file->createDevice());

}

QImage EPubContainer::getImage(const QString &id)

{

if (!m\_items.contains(id)) {

qWarning() << "Asked for unknown item" << id;

return QImage();

}

const EpubItem &item = m\_items.value(id);

if (!QImageReader::supportedMimeTypes().contains(item.mimetype)) {

qWarning() << "Asked for unsupported type" << item.mimetype;

return QImage();

}

QSharedPointer<QIODevice> ioDevice = getIoDevice(item.path);

if (!ioDevice) {

return QImage();

}

return QImage::fromData(ioDevice->readAll());

}

QString EPubContainer::getMetadata(const QString &key)

{

return m\_metadata.value(key);

}

bool EPubContainer::parseMimetype()

{

Q\_ASSERT(m\_rootFolder);

const KArchiveFile \*mimetypeFile = m\_rootFolder->file(MIMETYPE\_FILE);

if (!mimetypeFile) {

emit errorHappened(tr("Unable to find mimetype in file"));

return false;

}

QScopedPointer<QIODevice> ioDevice(mimetypeFile->createDevice());

QByteArray mimetype = ioDevice->readAll();

if (mimetype != "application/epub+zip") {

qWarning() << "Unexpected mimetype" << mimetype;

}

return true;

}

bool EPubContainer::parseContainer()

{

Q\_ASSERT(m\_rootFolder);

const KArchiveFile \*containerFile = getFile(CONTAINER\_FILE);

if (!containerFile) {

qWarning() << "no container file";

emit errorHappened(tr("Unable to find container information"));

return false;

}

QScopedPointer<QIODevice> ioDevice(containerFile->createDevice());

Q\_ASSERT(ioDevice);

// The only thing we need from this file is the path to the root file

QDomDocument document;

document.setContent(ioDevice.data());

QDomNodeList rootNodes = document.elementsByTagName("rootfile");

for (int i=0; i<rootNodes.count(); i++) {

QDomElement rootElement = rootNodes.at(i).toElement();

QString rootfilePath = rootElement.attribute("full-path");

if (rootfilePath.isEmpty()) {

qWarning() << "Invalid root file entry";

continue;

}

if (parseContentFile(rootfilePath)) {

return true;

}

}

// Limitations:

// - We only read one rootfile

// - We don't read the following from META-INF/

// - manifest.xml (unknown contents, just reserved)

// - metadata.xml (unused according to spec, just reserved)

// - rights.xml (reserved for DRM, not standardized)

// - signatures.xml (signatures for files, standardized)

emit errorHappened(tr("Unable to find and use any content files"));

return false;

}

bool EPubContainer::parseContentFile(const QString filepath)

{

const KArchiveFile \*rootFile = getFile(filepath);

if (!rootFile) {

emit errorHappened(tr("Malformed metadata, unable to get content metadata path"));

return false;

}

QScopedPointer<QIODevice> ioDevice(rootFile->createDevice());

QDomDocument document;

document.setContent(ioDevice.data(), true); // turn on namespace processing

QDomNodeList metadataNodeList = document.elementsByTagName("metadata");

for (int i=0; i<metadataNodeList.count(); i++) {

QDomNodeList metadataChildList = metadataNodeList.at(i).childNodes();

for (int j=0; j<metadataChildList.count(); j++) {

parseMetadataItem(metadataChildList.at(j));

}

}

// Extract current path, for resolving relative paths

QString contentFileFolder;

int separatorIndex = filepath.lastIndexOf('/');

if (separatorIndex > 0) {

contentFileFolder = filepath.left(separatorIndex + 1);

}

// Parse out all the components/items in the epub

QDomNodeList manifestNodeList = document.elementsByTagName("manifest");

for (int i=0; i<manifestNodeList.count(); i++) {

QDomElement manifestElement = manifestNodeList.at(i).toElement();

QDomNodeList manifestItemList = manifestElement.elementsByTagName("item");

for (int j=0; j<manifestItemList.count(); j++) {

parseManifestItem(manifestItemList.at(j), contentFileFolder);

}

}

// Parse out the document order

QDomNodeList spineNodeList = document.elementsByTagName("spine");

for (int i=0; i<spineNodeList.count(); i++) {

QDomElement spineElement = spineNodeList.at(i).toElement();

QString tocId = spineElement.attribute("toc");

if (!tocId.isEmpty() && m\_items.keys().contains(tocId)) {

EpubPageReference tocReference;

tocReference.title = tr("Table of Contents");

tocReference.target = tocId;

m\_standardReferences.insert(EpubPageReference::TableOfContents, tocReference);

}

QDomNodeList spineItemList = spineElement.elementsByTagName("itemref");

for (int j=0; j<spineItemList.count(); j++) {

parseSpineItem(spineItemList.at(j));

}

}

// Parse out standard items

QDomNodeList guideNodeList = document.elementsByTagName("guide");

for (int i=0; i<guideNodeList.count(); i++) {

QDomElement guideElement = guideNodeList.at(i).toElement();

QDomNodeList guideItemList = guideElement.elementsByTagName("reference");

for (int j=0; j<guideItemList.count(); j++) {

parseGuideItem(guideItemList.at(j));

}

}

return true;

}

bool EPubContainer::parseMetadataItem(const QDomNode &metadataNode)

{

QDomElement metadataElement = metadataNode.toElement();

QString tagName = metadataElement.tagName();

QString metaName;

QString metaValue;

if (tagName == "meta") {

metaName = metadataElement.attribute("name");

metaValue = metadataElement.attribute("content");

} else if (metadataElement.prefix() != "dc") {

qWarning() << "Unsupported metadata tag" << tagName;

return false;

} else if (tagName == "date") {

metaName = metadataElement.attribute("event");

metaValue = metadataElement.text();

} else {

metaName = tagName;

metaValue = metadataElement.text();

}

if (metaName.isEmpty() || metaValue.isEmpty()) {

return false;

}

m\_metadata[metaName] = metaValue;

return true;

}

bool EPubContainer::parseManifestItem(const QDomNode &manifestNode, const QString currentFolder)

{

QDomElement manifestElement = manifestNode.toElement();

QString id = manifestElement.attribute("id");

QString path = manifestElement.attribute("href");

QString type = manifestElement.attribute("media-type");

if (id.isEmpty() || path.isEmpty()) {

qWarning() << "Invalid item at line" << manifestElement.lineNumber();

return false;

}

// Resolve relative paths

path = QDir::cleanPath(currentFolder + path);

EpubItem item;

item.mimetype = type.toUtf8();

item.path = path;

m\_items[id] = item;

static QSet<QString> documentTypes({"text/x-oeb1-document", "application/x-dtbook+xml", "application/xhtml+xml"});

// All items not listed in the spine should be in this

if (documentTypes.contains(type)) {

m\_unorderedItems.insert(id);

}

return true;

}

bool EPubContainer::parseSpineItem(const QDomNode &spineNode)

{

QDomElement spineElement = spineNode.toElement();

// Ignore this for now

if (spineElement.attribute("linear") == "no") {

// return true;

}

QString referenceName = spineElement.attribute("idref");

if (referenceName.isEmpty()) {

qWarning() << "Invalid spine item at line" << spineNode.lineNumber();

return false;

}

if (!m\_items.keys().contains(referenceName)) {

qWarning() << "Unable to find" << referenceName << "in items";

return false;

}

m\_unorderedItems.remove(referenceName);

m\_orderedItems.append(referenceName);

return true;

}

bool EPubContainer::parseGuideItem(const QDomNode &guideItem)

{

QDomElement guideElement = guideItem.toElement();

QString target = guideElement.attribute("href");

QString title = guideElement.attribute("title");

QString type = guideElement.attribute("type");

if (target.isEmpty() || title.isEmpty() || type.isEmpty()) {

qWarning() << "Invalid guide item" << target << title << type;

return false;

}

EpubPageReference reference;

reference.target = target;

reference.title = title;

EpubPageReference::StandardType standardType = EpubPageReference::typeFromString(type);

if (standardType == EpubPageReference::Other) {

m\_otherReferences[type] = reference;

} else {

m\_standardReferences[standardType] = reference;

}

return true;

}

const KArchiveFile \*EPubContainer::getFile(const QString &path)

{

if (path.isEmpty()) {

return nullptr;

}

const KArchiveDirectory \*folder = m\_rootFolder;

// Try to walk down the correct path

QStringList pathParts = path.split('/', Qt::SkipEmptyParts);

for (int i=0; i<pathParts.count() - 1; i++) {

QString folderName = pathParts[i];

const KArchiveEntry \*entry = folder->entry(folderName);

if (!entry) {

qWarning() << "Unable to find folder name" << folderName << "in" << path.left(100);

const QStringList entries = folder->entries();

for (const QString &folderEntry : entries) {

if (folderEntry.compare(folderName, Qt::CaseInsensitive) == 0) {

entry = folder->entry(folderEntry);

break;

}

}

if (!entry) {

qWarning() << "Didn't even find with case-insensitive matching";

return nullptr;

}

}

if (!entry->isDirectory()) {

qWarning() << "Expected" << folderName << "to be a directory in path" << path;

return nullptr;

}

folder = dynamic\_cast<const KArchiveDirectory\*>(entry);

Q\_ASSERT(folder);

}

QString filename;

if (pathParts.isEmpty()) {

filename = path;

} else {

filename = pathParts.last();

}

const KArchiveFile \*file = folder->file(filename);

if (!file) {

qWarning() << "Unable to find file" << filename << "in" << folder->name();

const QStringList entries = folder->entries();

for (const QString &folderEntry : entries) {

if (folderEntry.compare(filename, Qt::CaseInsensitive) == 0) {

file = folder->file(folderEntry);

break;

}

}

if (!file) {

qWarning() << "Unable to find file" << filename << "in" << folder->name() << "with case-insensitive matching" << entries;

}

}

return file;

}

EpubPageReference::StandardType EpubPageReference::typeFromString(const QString &name) {

if (name == "cover") {

return CoverPage;

} else if (name == "title-page") {

return TitlePage;

} else if (name == "toc") {

return TableOfContents;

} else if (name == "index") {

return Index;

} else if (name == "glossary") {

return Glossary;

} else if (name == "acknowledgements") {

return Acknowledgements;

} else if (name == "bibliography") {

return Bibliography;

} else if (name == "colophon") {

return Colophon;

} else if (name == "copyright-page") {

return CopyrightPage;

} else if (name == "dedication") {

return Dedication;

} else if (name == "epigraph") {

return Epigraph;

} else if (name == "foreword") {

return Foreword;

} else if (name == "loi") {

return ListOfIllustrations;

} else if (name == "lot") {

return ListOfTables;

} else if (name == "notes") {

return Notes;

} else if (name == "preface") {

return Preface;

} else if (name == "text") {

return Text;

} else {

return Other;

}

}

Файл epubdocument.cpp

#include "epubdocument.h"

#include "epubcontainer.h"

#include <QIODevice>

#include <QDebug>

#include <QDir>

#include <QTextCursor>

#include <QThread>

#include <QElapsedTimer>

#include <QDomDocument>

#include <QSvgRenderer>

#include <QPainter>

#include <QTextBlock>

#include <QRegularExpression>

#include <QFontDatabase>

#include <QTextDocumentFragment>

#include <QImageReader>

#include <QAbstractTextDocumentLayout>

#include <qmath.h>

#ifdef DEBUG\_CSS

#include <private/qcssparser\_p.h>

#endif

EPubDocument::EPubDocument(QObject \*parent) : QTextDocument(parent),

m\_container(nullptr),

m\_loaded(false)

{

setUndoRedoEnabled(false);

connect(documentLayout(), &QAbstractTextDocumentLayout::documentSizeChanged, this, [=](const QSizeF &newSize) {

qDebug() << "doc size changed" << newSize;

m\_docSize = newSize;

});

}

EPubDocument::~EPubDocument()

{

for (const int fontId : m\_loadedFonts) {

QFontDatabase::removeApplicationFont(fontId);

}

}

void EPubDocument::openDocument(const QString &path)

{

m\_documentPath = path;

loadDocument();

}

void EPubDocument::loadDocument()

{

QElapsedTimer timer;

timer.start();

m\_container = new EPubContainer(this);

connect(m\_container, &EPubContainer::errorHappened, this, [](QString error) {

qWarning().noquote() << error;

});

if (!m\_container->openFile(m\_documentPath)) {

return;

}

qDebug() << "Opened in" << timer.restart() << "ms";

//QTextCursor cursor(this);

//cursor.movePosition(QTextCursor::End);

QStringList items = m\_container->getItems();

QString cover = m\_container->getStandardPage(EpubPageReference::CoverPage);

if (!cover.isEmpty()) {

items.prepend(cover);

qDebug() << cover;

}

QDomDocument domDoc;

QTextCursor textCursor(this);

textCursor.beginEditBlock();

textCursor.movePosition(QTextCursor::End);

QTextBlockFormat pageBreak;

pageBreak.setPageBreakPolicy(QTextFormat::PageBreak\_AlwaysBefore);

//for (const QString &chapter : items) {

while(!items.isEmpty()) {

const QString &chapter = items.takeFirst();

m\_currentItem = m\_container->getEpubItem(chapter);

if (m\_currentItem.path.isEmpty()) {

continue;

}

QSharedPointer<QIODevice> ioDevice = m\_container->getIoDevice(m\_currentItem.path);

if (!ioDevice) {

qWarning() << "Unable to get iodevice for chapter" << chapter;

continue;

}

domDoc.setContent(ioDevice.data());

setBaseUrl(QUrl(m\_currentItem.path));

fixImages(domDoc);

textCursor.insertFragment(QTextDocumentFragment::fromHtml(domDoc.toString()));

textCursor.insertBlock(pageBreak);

}

qDebug() << "Base url:" << baseUrl();

setBaseUrl(QUrl());

emit loadCompleted();

qDebug() << "Load done in" << timer.restart() << "ms";

{

QElapsedTimer timer;

timer.start();

QFont f = defaultFont();

QFontMetrics fm(f);

int mw = fm.horizontalAdvance(QLatin1Char('x')) \* 80;

int w = mw;

setTextWidth(w);

qDebug() << "Text width set in" << timer.restart() << "ms";

QSizeF size = m\_docSize;

if (size.width() != 0) {

w = qSqrt((uint)(5 \* size.height() \* size.width() / 3));

setTextWidth(qMin(w, mw));

size = m\_docSize;//documentLayout()->documentSize();

if (w\*3 < 5\*size.height()) {

w = qSqrt((uint)(2 \* size.height() \* size.width()));

setTextWidth(qMin(w, mw));

}

}

qDebug() << "Text width changed in" << timer.restart() << "ms";

w = idealWidth();

qDebug() << "Ideal width in" << timer.restart() << "ms";

setTextWidth(w);

qDebug() << "Final text width in" << timer.restart() << "ms";

}

qDebug() << "Adjust size done in" << timer.elapsed() << "ms";

textCursor.endEditBlock();

m\_loaded = true;

}

void EPubDocument::fixImages(QDomDocument &newDocument)

{

// TODO: FIXME: replace this with not smushing all HTML together in one document

{ // Fix relative URLs, images are lazily loaded so the base URL might not

// be correct when they are loaded

QDomNodeList imageNodes = newDocument.elementsByTagName("img");

for (int i=0; i<imageNodes.count(); i++) {

QDomElement image = imageNodes.at(i).toElement();

if (!image.hasAttribute("src")) {

continue;

}

QUrl href = QUrl(image.attribute("src"));

href = baseUrl().resolved(href);

image.setAttribute("src", href.toString());

}

}

{ // QImage which QtSvg uses isn't able to read files from inside the archive, so embed image data inline

QDomNodeList imageNodes = newDocument.elementsByTagName("image"); // SVG images

for (int i=0; i<imageNodes.count(); i++) {

QDomElement image = imageNodes.at(i).toElement();

if (!image.hasAttribute("xlink:href")) {

continue;

}

QString path = image.attribute("xlink:href");

QByteArray fileData = loadResource(0, QUrl(path)).toByteArray();

QByteArray data = "data:image/jpeg;base64," +fileData.toBase64();

image.setAttribute("xlink:href", QString::fromLatin1(data));

}

}

static int svgCounter = 0;

// QTextDocument isn't fond of SVGs, so rip them out and store them separately, and give it <img> instead

QDomNodeList svgNodes = newDocument.elementsByTagName("svg");

for (int i=0; i<svgNodes.count(); i++) {

QDomElement svgNode = svgNodes.at(i).toElement();

// Serialize out the old SVG, store it

QDomDocument tempDocument;

tempDocument.appendChild(tempDocument.importNode(svgNode, true));

QString svgId = QString::number(++svgCounter);

m\_svgs.insert(svgId, tempDocument.toByteArray());

// Create <img> node pointing to our SVG image

QDomElement imageElement = newDocument.createElement("img");

imageElement.setAttribute("src", "svgcache:" + svgId);

// Replace <svg> node with our <img> node

QDomNode parent = svgNodes.at(i).parentNode();

parent.replaceChild(imageElement, svgNode);

}

}

const QImage &EPubDocument::getSvgImage(const QString &id)

{

if (m\_renderedSvgs.contains(id)) {

return m\_renderedSvgs[id];

}

if (!m\_svgs.contains(id)) {

qWarning() << "Couldn't find SVG" << id;

static QImage nullImg;

return nullImg;

}

QSize imageSize(pageSize().width() - documentMargin() \* 4,

pageSize().height() - documentMargin() \* 4);

QSvgRenderer renderer(m\_svgs.value(id));

QSize svgSize(renderer.viewBox().size());

if (svgSize.isValid()) {

if (svgSize.scaled(imageSize, Qt::KeepAspectRatio).isValid()) {

svgSize.scale(imageSize, Qt::KeepAspectRatio);

}

} else {

svgSize = imageSize;

}

QImage rendered(svgSize, QImage::Format\_ARGB32);

QPainter painter(&rendered);

if (!painter.isActive()) {

qWarning() << "Unable to activate painter" << svgSize;

static const QImage dummy = QImage();

return dummy;

}

renderer.render(&painter);

painter.end();

m\_renderedSvgs.insert(id, rendered);

return m\_renderedSvgs[id];

}

QVariant EPubDocument::loadResource(int type, const QUrl &url)

{

Q\_UNUSED(type);

if (url.scheme() == "svgcache") {

return getSvgImage(url.path());

}

if (url.scheme() == "data") {

QByteArray data = url.path().toUtf8();

const int start = data.indexOf(';');

if (start == -1) {

qWarning() << "unable to decode data:, no ;" << data.left(100);

data = QByteArray();

addResource(type, url, data);

return data;

}

data = data.mid(start + 1);

if (data.startsWith("base64,")) {

data = QByteArray::fromBase64(data.mid(data.indexOf(',') + 1));

} else {

qWarning() << "unable to decode data:, unknown encoding" << data.left(100);

data = QByteArray();

}

addResource(type, url, data);

return data;

}

QSharedPointer<QIODevice> ioDevice = m\_container->getIoDevice(url.path());

if (!ioDevice) {

qWarning() << "Unable to get io device for" << url.toString().left(100);

qDebug() << url.scheme();

return QVariant();

}

QByteArray data = ioDevice->readAll();

if (type == QTextDocument::StyleSheetResource) {

const QString cssData = QString::fromUtf8(data);

// Extract embedded fonts

static const QRegularExpression fontfaceRegex("@font-face\\s\*{[^}]+}", QRegularExpression::MultilineOption);

QRegularExpressionMatchIterator fontfaceIterator = fontfaceRegex.globalMatch(cssData);

while (fontfaceIterator.hasNext()) {

QString fontface = fontfaceIterator.next().captured();

static const QRegularExpression urlExpression("url\\s\*\\(([^\\)]+)\\)");

QString fontPath = urlExpression.match(fontface).captured(1);

// Resolve relative and whatnot shit

fontPath = QDir::cleanPath(QFileInfo(baseUrl().path()).path() + '/' + fontPath);

QSharedPointer<QIODevice> ioDevice = m\_container->getIoDevice(fontPath);

if (ioDevice) {

m\_loadedFonts.append(QFontDatabase::addApplicationFontFromData(ioDevice->readAll()));

qDebug() << "Loaded font" << QFontDatabase::applicationFontFamilies(m\_loadedFonts.last());

} else {

qWarning() << "Failed to load font from" << fontPath << baseUrl();

}

}

data = cssData.toUtf8();

//#ifdef DEBUG\_CSS

QCss::Parser parser(cssData);

QCss::StyleSheet stylesheet;

qDebug() << "=====================";

qDebug() << "Parse success?" << parser.parse(&stylesheet);

qDebug().noquote() << parser.errorIndex << parser.errorSymbol().lexem();

//#endif

}

addResource(type, url, data);

return data;

}

Файл mainwindow.cpp

#include "ui\_mainwindow.h"

#include "widget.h"

#include "mainwindow.h"

#include "data.h"

#include <QSettings>

#include <QDateTime>

#include <QDebug>

#define SAVES\_PATH "/Users/hanna/Documents/qt/idgfa/idkwati/saves/"

#define PROJECT\_PATH "/Users/hanna/Documents/qt/idgfa/idkwati/"

MainWindow::MainWindow(QWidget \*parent)

: QMainWindow(parent)

, ui(new Ui::MainWindow)

{

ui->setupUi(this);

}

MainWindow::~MainWindow()

{

delete ui;

}

void MainWindow::on\_createShelveBtn\_clicked()

{

QString shelveName = ui->shelveNameLineEdit->text();

if(shelveName == ""){

QMessageBox::warning(this,"Shelve error", "You should enter a valid name. Try again.");

}else{

if(ui->typeOfShelve->findText(shelveName) == -1){ //not found in comboBox

ui->typeOfShelve->addItem(shelveName);

}else{

QMessageBox::warning(this,"Shelve error", "Entered name already exists.");

}

}

}

void MainWindow::on\_deleteShelveBtn\_clicked()

{

QString shelveName = ui->shelveNameLineEdit->text();

if(shelveName == ""||shelveName == "undecided" ){

QMessageBox::warning(this,"Shelve error", "You should enter a valid name. Try again.");

}else{

if(ui->typeOfShelve->findText(shelveName) != -1 ){ //found in comboBox

ui->typeOfShelve->removeItem(ui->typeOfShelve->findText(shelveName,Qt::MatchContains));

for (int i = 0; i < ui->bookTable->rowCount(); i++) {

if (ui->bookTable->item(i, 1)->text() == shelveName) {

ui->bookTable->setItem(i, 1, new QTableWidgetItem("undecided"));

}

}

}else{

QMessageBox::warning(this,"Shelve error", "Entered shelve does not exist.");

}

}

}

void MainWindow::on\_uploadFileBtn\_clicked()

{

QString defaultPath ="/Users/hanna/Desktop/books/";

QString filter = "EPUB (\*.epub)";

QString fileName = QFileDialog::getOpenFileName(this, "Select a file", defaultPath, filter);

ui->pathLbl->setText(fileName);

}

void MainWindow::on\_addBookBtn\_clicked()

{

QString filePath = ui->pathLbl->text();

QString bookName = ui->bookNameLineEdit->text();

if(bookName == "" || filePath == ""){

QMessageBox::warning(this,"Name", "You should upload the book and enter the name.\nTry again.");

}else{

QList<QTableWidgetItem \*> NameList = ui->bookTable->findItems(bookName, Qt::MatchExactly);

QList<QTableWidgetItem \*> PathList = ui->bookTable->findItems(filePath, Qt::MatchExactly);

if(NameList.isEmpty()!=0 && PathList.isEmpty()!=0 ){

ui->bookTable->insertRow(0);

ui->bookTable->setItem(0, 0, new QTableWidgetItem(bookName));

ui->bookTable->setItem(0, 1, new QTableWidgetItem("undecided"));

ui->bookTable->setItem(0, 2, new QTableWidgetItem(filePath));

}else{

QMessageBox::warning(this,"Book in the table", "Book already exists in the table. Try again.");

}

}

}

void MainWindow::on\_deleteBookBtn\_clicked(){

QString bookName = ui->bookNameLineEdit->text();

int rowIndex = 0, flag = 0;

for (int i = 0; i < ui->bookTable->rowCount(); i++) {

if (ui->bookTable->item(i, 0)->text() == bookName) {

flag = 1;

rowIndex = i;

break;

}

}

if(flag == 1){

QString shelveName = ui->bookTable->item(rowIndex, 1)->text();

ui->typeOfShelve->removeItem(ui->typeOfShelve->findText(shelveName,Qt::MatchContains));

ui->bookTable->removeRow(rowIndex);

}else{

QMessageBox::warning(this,"Book in the table", "Book does not exist. Try again.");

}

}

void MainWindow::on\_saveTableBtn\_clicked()

{

QDateTime date = QDateTime::currentDateTime();

QString formattedTime = date.toString("dd-MM-yyyy\_hh:mm:ss");

QByteArray formattedTimeMsg = formattedTime.toLocal8Bit();

QString saveName = SAVES\_PATH + formattedTimeMsg + ".txt";

saveTableContents(ui->bookTable, saveName);

QMessageBox::warning(this,"Book in the table", "The book list has been saved succesfully in \"idkwati/saves\" folder!");

}

void MainWindow::on\_uploadTableBtn\_clicked()

{

QString filter = "TXT (\*.txt)";

QString fileName = QFileDialog::getOpenFileName(this, "Select a file", PROJECT\_PATH, filter);

ui->pathLbl->setText(fileName);

loadTableContents(ui->bookTable, ui->typeOfShelve, fileName);

}

void MainWindow::on\_setShelveBtn\_clicked()

{

int rowIndex = 0, flagTable = 0;

QString bookName = ui->bookNameLineEdit->text();

QString shelveName = ui->shelveNameLineEdit->text();

if(bookName == ""||shelveName == ""){

QMessageBox::warning(this,"Name", "You should enter the name of the book and the shelve. Try again.");

return;

}

if(ui->typeOfShelve->findText(shelveName) != -1 ){ //found in comboBox

for (int i = 0; i < ui->bookTable->rowCount(); i++) {

if (ui->bookTable->item(i, 0)->text() == bookName) {

flagTable = 1;

rowIndex = i;

break;

}

}

if(flagTable == 1){

ui->bookTable->setItem(rowIndex, 1, new QTableWidgetItem(shelveName));

}else{

QMessageBox::warning(this,"Book in the table", "Book does not exist. Try again.");

}

}else{

QMessageBox::warning(this,"Shelve error", "Given shelve does not exist. You should add it separatly and try again..");

}

}

void MainWindow::on\_openBookBtn\_clicked()

{

QString bookPath;

QString bookName = ui->bookNameLineEdit->text();

int rowIndex = 0, flag = 0;

for (int i = 0; i < ui->bookTable->rowCount(); i++) {

if (ui->bookTable->item(i, 0)->text() == bookName) {

flag = 1;

rowIndex = i;

break;

}

}

if(flag == 1){

bookPath = ui->bookTable->item(rowIndex, 2)->text();

}else{

QMessageBox::warning(this,"Book in the table", "Book does not exist. Try again.");

}

bool endsWith = false;

endsWith = bookPath.endsWith("epub",Qt::CaseInsensitive);

if(endsWith == true){

Widget \*w = new Widget;

w->setAttribute(Qt::WA\_DeleteOnClose);

QSettings settings;

settings.setValue("lastFile", bookPath);

if (!w->loadFile(bookPath, bookName)) {

qWarning() << "Failed to load" << bookPath;

return;

}

w->show();

}else{

QMessageBox::warning(this,"Format error", "Unsupported book format.");

}

}

Файл widget.cpp

#include "epubdocument.h"

#include "widget.h"

#include <QFileDialog>

#include <QSettings>

#include <QDebug>

#include <QPainter>

#include <QKeyEvent>

#include <QAbstractTextDocumentLayout>

#include <QApplication>

Widget::Widget(QWidget \*parent)

: QDialog(parent),

m\_document(new EPubDocument(this)),

m\_currentChapter(0)

{

setWindowFlags(Qt::Dialog);

resize(600, 800);

m\_document->setParent(this);

connect(m\_document, &EPubDocument::loadCompleted, this, [&]() {

update();

});

}

Widget::~Widget()

{

}

bool Widget::loadFile(const QString &path, const QString bookName)

{

if (path.isEmpty()) {

return false;

}

if (!QFile::exists(path)) {

qWarning() << path << "doesn't exist";

return false;

}

this->setWindowTitle(bookName);

m\_document->setPageSize(size());

m\_document->openDocument(path);

return true;

}

void Widget::scroll(int amount)

{

int offset = m\_yOffset + amount;

offset = qMin(int(m\_document->size().height() - m\_document->pageSize().height()), offset);

m\_yOffset = qMax(0, offset);

update();

}

void Widget::scrollPage(int amount)

{

int currentPage = m\_yOffset / m\_document->pageSize().height();

currentPage += amount;

int offset = currentPage \* m\_document->pageSize().height();

offset = qMin(int(m\_document->size().height() - m\_document->pageSize().height()), offset);

m\_yOffset = qMax(0, offset);

update();

}

void Widget::paintEvent(QPaintEvent\*)

{

QPainter painter(this);

painter.fillRect(rect(), Qt::white);

if (!m\_document->loaded()) {

painter.drawText(rect(), Qt::AlignCenter, "Loading...");

return;

}

QAbstractTextDocumentLayout::PaintContext paintContext;

paintContext.clip = rect();

paintContext.clip.translate(0, m\_yOffset);

paintContext.palette = palette();

for (int group = 0; group < 3; ++group) {

paintContext.palette.setColor(QPalette::ColorGroup(group), QPalette::WindowText, Qt::black);

paintContext.palette.setColor(QPalette::ColorGroup(group), QPalette::Light, Qt::black);

paintContext.palette.setColor(QPalette::ColorGroup(group), QPalette::Text, Qt::black);

paintContext.palette.setColor(QPalette::ColorGroup(group), QPalette::Base, Qt::black);

paintContext.palette.setColor(QPalette::ColorGroup(group), QPalette::Base, Qt::white);

paintContext.palette.setColor(QPalette::ColorGroup(group), QPalette::Window, Qt::white);

paintContext.palette.setColor(QPalette::ColorGroup(group), QPalette::Button, Qt::white);

}

painter.translate(0, -m\_yOffset);

painter.setClipRect(paintContext.clip);

m\_document->documentLayout()->draw(&painter, paintContext);

}

void Widget::keyPressEvent(QKeyEvent \*event)

{

if (event->key() == Qt::Key\_Up) {

scroll(-50);

} else if (event->key() == Qt::Key\_Down) {

scroll(50);

} else if (event->key() == Qt::Key\_Left) {

scrollPage(-1);

} else if (event->key() == Qt::Key\_Right) {

scrollPage(1);

} else if (event->key() == Qt::Key\_End) {

m\_yOffset = m\_document->size().height() - m\_document->pageSize().height();

update();

} else if (event->key() == Qt::Key\_Escape) {

close();

}

}

void Widget::resizeEvent(QResizeEvent \*)

{

m\_document->clearCache();

m\_document->setPageSize(size());

update();

}

Файл main.cpp

#include "mainwindow.h"

#include <QApplication>

int main(int argc, char \*argv[])

{

QApplication a(argc, argv);

MainWindow w;

w.show();

return a.exec();

}

# ПРИЛОЖЕНИЕ Б

(обязательное)

Диаграмма классов

# ПРИЛОЖЕНИЕ В

(обязательное)

Схема алгоритма loadDocument()

# ПРИЛОЖЕНИЕ Г

(обязательное)  
Схема алгоритма paintEvent()