МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ

УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

**«МОСКОВСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»**

**(МОСКОВСКИЙ ПОЛИТЕХ)**

Факультет информационных технологий

Кафедра «Инфокогнитивные технологии»

**КУРСОВОЙ ПРОЕКТ**

на тему: *«Инструмент исследования данных из файлов с книгами».*

Направление подготовки 09.03.03 «Прикладная информатика»

Профиль «Корпоративные информационные системы»

**Выполнил:**

студент группы 211-362

Назаренко Глеб Максимович

|  |  |
| --- | --- |
| 26.06.2023 |  |
|  | (подпись) |

Москва 2023

**СОДЕРЖАНИЕ**

[1 Основные сведения 4](#_Toc138759263)

[1.1 Цель проекта 4](#_Toc138759264)

[1.2 Обоснование актуальности задачи 4](#_Toc138759265)

[1.3 Описание функциональности приложения 4](#_Toc138759266)

[2 Общие принципы работы приложения 6](#_Toc138759267)

[2.1 Создание базы данных 6](#_Toc138759268)

[2.2 Обработка файлов и заполнение базы данных 6](#_Toc138759269)

[2.3 Извлечение данных из базы данных 6](#_Toc138759270)

[2.4 Пользовательский интерфейс 7](#_Toc138759271)

[3 Создание базы данных 8](#_Toc138759272)

[3.1 Выбор подходящей системы управления базами данных 8](#_Toc138759273)

[3.2 Проектирование структуры базы данных 8](#_Toc138759274)

[3.3 Описание процесса инициализации базы данных 8](#_Toc138759275)

[4 Заполнение базы данных 10](#_Toc138759276)

[4.1 Описание процесса чтения данных из файлов 10](#_Toc138759277)

[4.2 Обработка и структурирование данных перед внесением  
 в базу данных 10](#_Toc138759278)

[4.3 Процесс записи данных в базу данных 10](#_Toc138759279)

[5 Извлечение данных из базы данных 11](#_Toc138759280)

[5.1 Описание запросов к базе данных для извлечения данных 11](#_Toc138759281)

[5.2 Преобразование извлеченных данных для дальнейшего использования 12](#_Toc138759282)

[6 Расширенные функции управления 14](#_Toc138759283)

[6.1 Функционал фильтрации и отображения метаданных 14](#_Toc138759284)

[6.2 Управление списком избранных книг 15](#_Toc138759285)

[7 Обработка директорий и файлов 16](#_Toc138759286)

[7.1 Общий принцип работы обработки директорий 16](#_Toc138759287)

[7.2 Специфические механизмы обработки файлов различных типов 16](#_Toc138759288)

[Заключение 18](#_Toc138759289)

# Основные сведения

## Цель проекта

Целью данного проекта является создание многопоточного приложения на языке Python, которое способно обрабатывать и анализировать файлы различных типов (PDF, ODT, DOCX, EPUB), хранящиеся в директории. Обработка включает чтение содержимого файла, структурирование полученных данных и запись их в базу данных. Приложение также предоставляет удобный интерфейс для взаимодействия с пользователем, позволяющий выбрать директорию для обработки, задать параметры обработки, а также просматривать и фильтровать данные, хранящиеся в базе данных.

## Обоснование актуальности задачи

В современном мире информация часто хранится в форме файлов различных форматов, и их анализ может быть весьма трудоемким процессом. Особенно это становится сложным, когда файлы хранятся в неструктурированной манере в различных директориях. Наше приложение упрощает эту задачу, автоматизируя процесс чтения, обработки и записи данных. Также, за счет расширенной функциональности взаимодействия с базой данных, оно позволяет удобно просматривать и анализировать собранные данные, что может быть полезно для различных целей, таких как анализ данных, машинное обучение и другие задачи, требующие структурированного набора данных.

## Описание функциональности приложения

Приложение предоставляет следующую функциональность:

1. Возможность выбора директории для обработки.
2. Указание параметров обработки, включая типы файлов для обработки, исключенные директории, максимальную глубину поиска и конвертацию файлов в PDF формат для получения дополнительных данных.
3. Чтение, структурирование и запись данных из обработанных файлов в базу данных.
4. Интерфейс для взаимодействия с базой данных, включая извлечение, просмотр и фильтрацию данных.
5. Предоставление дополнительной информации о файлах, такой как размер файла, количество страниц, метаданные и т.д.
6. Возможность фильтрации файлов по различным параметрам, включая название, автора, размер файла, количество страниц и метаданные.

# Общие принципы работы приложения

## Создание базы данных

В основе работы приложения лежит база данных SQLite. Создание базы данных происходит при инициализации экземпляра класса BookAnalyzer, используя встроенные возможности Python для работы с SQLite. При этом создаются необходимые таблицы и индексы для эффективной работы с данными.

## Обработка файлов и заполнение базы данных

Процесс обработки файлов запускается из пользовательского интерфейса приложения. Пользователь может выбрать директорию для обработки и задать параметры обработки, включая типы файлов для обработки, исключенные директории, максимальную глубину поиска и конвертацию файлов в PDF формат, для получения большего количества данных из файла.

Для обработки файлов используется класс BookAnalyzer. Он рекурсивно просматривает все файлы в указанной директории и поддиректориях (до указанной глубины), анализирует файлы указанных типов и обновляет данные в базе данных.

## Извлечение данных из базы данных

Данные из базы данных могут быть извлечены и отфильтрованы с помощью методов класса BookAnalyzer. Эти методы позволяют выполнить SQL-запросы к базе данных и получить результаты в форме структурированных данных. Функциональность фильтрации данных включает возможность сортировки по различным параметрам, таким как название, автор, размер файла, количество страниц и метаданные. Таким образом, данные, изначально содержащиеся в неструктурированной форме в файлах различных форматов, преобразуются в удобную для анализа форму.

## Пользовательский интерфейс

Пользовательский интерфейс приложения создается с использованием библиотеки tkinter. Он предоставляет пользователю удобные инструменты для выбора директории и параметров обработки, а также взаимодействия с базой данных. Интерфейс позволяет пользователям просматривать и фильтровать данные, хранящиеся в базе данных, и отображает информацию в удобном для чтения формате. Кроме того, он обеспечивает взаимодействие с базой данных, позволяя пользователю выполнять запросы к базе данных и просматривать результаты.

# Создание базы данных

## Выбор подходящей системы управления базами данных

В основе приложения лежит система управления базами данных (СУБД) SQLite. SQLite – это встраиваемая СУБД, которая хранит всю базу данных (включая определения, таблицы, индексы и актуальные данные) как один стандартный файл дисковой файловой системы. Это обеспечивает простоту взаимодействия с базой данных, поскольку не требуется установка сервера базы данных.

## Проектирование структуры базы данных

Данные о книгах хранятся в одной таблице books. Таблица включает следующие поля:

* id – уникальный идентификатор записи (целое число);
* title – название книги (текст);
* author – автор книги (текст);
* file\_ext – расширение файла книги (текст);
* file\_path – путь к файлу книги на диске (текст);
* file\_size – размер файла книги в байтах (целое число);
* num\_pages – количество страниц в книге (целое число);
* preview – превью книги (BLOB);
* metadata – метаданные книги (текст);
* favorite – индикатор избранности книги (целое число, где 0 означает, что книга не добавлена в избранное, и 1 означает, что книга добавлена в избранное).

## Описание процесса инициализации базы данных

Процесс инициализации базы данных происходит в методе init\_database() класса BookAnalyzer. В этом методе создается подключение к базе данных SQLite по заданному пути. Если файл базы данных не существует, он создается.

При помощи курсора выполняются SQL-запросы. Сначала, если параметр reset установлен в True, выполняется запрос на удаление таблицы books, если она существует. Затем выполняется запрос на создание таблицы books, если она не существует.

После выполнения запросов соединение с базой данных закрывается. Таким образом, после инициализации базы данных готова к использованию.

# Заполнение базы данных

## Описание процесса чтения данных из файлов

Метод update\_book\_data(self, file\_path) класса BookAnalyzer предназначен для чтения данных из файлов различных форматов и их последующего занесения в базу данных. Данные из файлов считываются с использованием специализированных библиотек, зависящих от формата файла (PdfReader для формата .pdf, epub для формата .epub, Document для формата .docx).

## Обработка и структурирование данных перед внесением в базу данных

После чтения данных из файла они обрабатываются и структурируются для занесения в базу данных. Например, извлекается расширение файла, и в зависимости от этого формата происходит соответствующая обработка данных. Для каждого формата производится извлечение метаданных, количества страниц и превью книги. Если в метаданных отсутствуют название книги, это поле заполняется из имени файла. Для каждого файла также извлекается его размер.

## Процесс записи данных в базу данных

После обработки и структурирования данных происходит их запись в базу данных. Для этого создается соединение с базой данных и открывается курсор для выполнения SQL-запросов. Производится проверка, существует ли уже в базе данных запись для данного файла. Если записи нет, выполняется запрос на вставку новой записи с полученными данными. Если запись уже существует, выполняется запрос на обновление этой записи. Затем сохраняются изменения и закрывается соединение с базой данных.

# Извлечение данных из базы данных

## Описание запросов к базе данных для извлечения данных

В классе BookAnalyzer реализованы различные методы для извлечения данных из базы данных:

* get\_all\_books() используется для получения данных обо всех книгах из базы данных. Метод поддерживает фильтрацию по избранным книгам, а также позволяет задать лимит и смещение для запроса. Результат запроса преобразуется в человекочитаемый формат перед возвратом;
* get\_book\_preview() и get\_book\_preview\_path() используются для извлечения превью конкретной книги по ее идентификатору или пути к файлу соответственно;
* search\_books\_by\_title() позволяет искать книги по названию, search\_books\_by\_author() - по автору, а search\_books\_by\_extension() - по расширению файла. Все эти методы поддерживают фильтрацию по избранным книгам;
* get\_largest\_books() возвращает самые большие книги, а метод get\_books\_with\_most\_pages() - книги с наибольшим количеством страниц, get\_recently\_added\_books() возвращает недавно добавленные книги. Эти методы поддерживают фильтрацию по избранным книгам и позволяют задать лимит и смещение для запроса;
* get\_books\_without\_author() и get\_books\_without\_metadata() позволяют получить книги без указанного автора или метаданных соответственно;
* get\_file\_extension\_statistics() возвращает статистику по расширениям файлов, а метод search\_books\_by\_metadata() позволяет искать книги по части метаданных;
* plot\_books\_pages() строит график распределения количества страниц по книгам.

## Преобразование извлеченных данных для дальнейшего использования

После извлечения данных из базы данных они преобразуются для дальнейшего использования или отображения в пользовательском интерфейсе. Например, размер файла преобразуется в человеко-читаемый формат с использованием функции pretty\_size(file\_size). Индикатор избранности книги преобразуется из цифрового значения в текстовое ("Да" или "Нет").

В методе plot\_books\_pages(self) извлеченные данные используются для создания графика, который отображает количество страниц для каждой книги. Данные разделяются на названия книг и количество страниц, которые затем используются для построения графика.

Метод display\_file\_extension\_statistics(self) класса App обрабатывает и отображает статистику расширений файлов, полученную из get\_file\_extension\_statistics(self).

1. Сначала метод получает статистику по расширениям файлов с помощью self.analyzer.get\_file\_extension\_statistics(). Полученные данные представляют собой список кортежей, где каждый кортеж содержит расширение файла и количество книг с этим расширением.
2. Данные затем преобразуются в словарь, где ключи – это расширения файлов, а значения - количество книг с соответствующим расширением.
3. После этого вычисляется общее количество файлов, что необходимо для вычисления процентного соотношения каждого расширения.
4. Затем создаются метки для диаграммы, которые включают расширение файла и его процентное соотношение от общего числа файлов.
5. Размеры (количество книг с каждым расширением) используются для создания круговой диаграммы с помощью matplotlib.
6. Для каждого сектора на диаграмме создается аннотация, которая показывает соответствующую метку. Аннотация располагается таким образом, чтобы она не перекрывала другие элементы на диаграмме.
7. Если в процессе обработки или отображения данных возникает исключение, оно перехватывается и пользователю показывается сообщение об ошибке.
8. В конце диаграмма отображается с помощью plt.show().

# Расширенные функции управления

## Функционал фильтрации и отображения метаданных

В дополнение к уже существующим возможностям приложения были добавлены функции фильтрации списка книг и отображения метаданных в отдельном окне.

Отображение только избранных книг. Пользователи могут отфильтровать отображаемые книги, чтобы видеть только те, которые они добавили в избранное. Это может быть особенно полезно при работе с большими каталогами. Для включения фильтрации пользователю нужно активировать чекбокс "Показать только избранные" на главном экране приложения. Команда self.favorites\_checkbox = tk.Checkbutton(self.root, text="Показать только избранные", variable=self.favorites\_var, command=self.update\_table) создает этот чекбокс, и когда его состояние меняется, вызывается функция self.update\_table, которая обновляет таблицу, показывая только избранные книги.

Сортировка по столбцам. Кроме того пользователи могут сортировать отображаемые книги по любому столбцу в таблице, просто нажав на название столбца. Это может быть полезно для быстрого нахождения книги с наибольшим количеством страниц, наибольшим размером файла и т.д. Функция сортировки реализована с помощью функции self.treeview\_sort\_column, которая принимает в качестве параметров виджет таблицы, имя столбца и флаг направления сортировки. Эта функция сначала собирает все данные из столбца, затем сортирует их и, наконец, перемещает строки таблицы в соответствии с отсортированным списком данных.

Отображение метаданных. Приложение позволяет пользователю просмотреть метаданные каждой книги в отдельном окне. Для этого пользователь может выделить книгу и нажать комбинацию клавиш Ctrl+m (должен быть выбран английский язык и нажимать прописную букву). Команда self.tree.bind("<Control-m>", show\_metadata) привязывает эту комбинацию к функции show\_metadata, которая открывает новое окно и заполняет его метаданными выбранной книги. Метаданные выводятся в формате ключ-значение для удобного просмотра.

## Управление списком избранных книг

В приложении предусмотрена возможность добавления книг в список избранных, а также удаления их из этого списка. Эта функция позволяет пользователю формировать персонализированный список книг, которые он считает наиболее интересными или полезными.

Добавление книги в избранное. Для добавления книги в список избранных пользователь может выделить книгу в общем списке и нажать комбинацию клавиш Ctrl+f (должен быть выбран английский язык и нажимать прописную букву). Команда self.tree.bind("<Control-f>", change\_favorite) привязывает эту комбинацию к функции change\_favorite, которая проверяет текущий статус книги ("Избранное" или нет) и изменяет его. Если статус был "Нет", он меняется на "Да", и книга добавляется в список избранных.

Удаление книги из избранного. Для удаления книги из списка избранных процесс такой же, как и при добавлении: пользователь выделяет книгу и нажимает комбинацию клавиш Ctrl+f (должен быть выбран английский язык и нажимать прописную букву). Если статус книги был "Да", он меняется на "Нет", и книга удаляется из списка избранных.

Важно отметить, что информация о статусе "Избранное" для каждой книги хранится в базе данных приложения и обновляется в реальном времени. Это обеспечивает сохранность информации о списке избранных даже при закрытии и повторном открытии приложения.

# Обработка директорий и файлов

## Общий принцип работы обработки директорий

Метод process\_directory() принимает следующие параметры: директорию для обработки, типы файлов для обработки, список исключений (директорий, которые не следует обрабатывать), максимальную глубину обработки директориев и текущую глубину обработки.

Метод реализует рекурсивный обход всех поддиректорий, начиная с указанной директории. Если текущая глубина превышает максимальную, обход прекращается. Для каждой поддиректории выполняется следующее:

1. С помощью os.scandir() перебираются все файлы и директории.
2. Если элемент является файлом и его расширение соответствует указанным типам файлов, метод update\_book\_data() вызывается для обновления данных о книге в базе данных.
3. Если элемент является директорией и его имя не включено в список исключений, метод process\_directory() вызывается рекурсивно для обработки этой поддиректории.

## Специфические механизмы обработки файлов различных типов

Метод update\_book\_data() принимает путь к файлу и обновляет информацию о книге в базе данных.

Для файлов разных типов используются различные методы извлечения данных:

1. Расширение «.pdf»: Извлекается метаданные и количество страниц с помощью PdfReader. Превью создается с помощью \_\_get\_preview().
2. Расширение «..epub»: Используется epub.read\_epub() для чтения книги, метаданные и количество страниц извлекаются из объекта книги. Превью создается с помощью \_\_get\_epub\_cover().
3. Расширение «..docx»: Если опция convert\_docx\_to\_pdf включена, файл преобразуется в PDF и извлекаются метаданные и количество страниц. В противном случае используется Document для чтения файла и извлечения метаданных. Количество страниц приближенно подсчитывается с помощью \_\_count\_pages\_docx(), а превью создается с помощью \_\_get\_docx\_preview().
4. Расширение «..odt»: Если опция convert\_odt\_to\_pdf включена, файл преобразуется в PDF и извлекаются метаданные и количество страниц. В противном случае из файла не извлекается метаданные, количество страниц и автор не определяется, а превью создается с помощью \_\_get\_odt\_preview().

После обработки файла проверяется наличие книги в базе данных. Если книги нет, то она добавляется, иначе обновляется информация о ней.

В случае возникновения ошибок при обработке файлов они перехватываются, и выводится сообщение об ошибке. После обработки всех файлов в директории соединение с базой данных закрывается.

# Заключение

Создано программное обеспечение для эффективного анализа и каталогизации электронных книг. В основе приложения лежит методика обработки различных типов файлов, включая PDF, EPUB, DOCX и ODT, что позволяет обеспечить широкую функциональность и универсальность использования.

Приложение предоставляет следующие возможности:

1. Сканирование директорий с книгами и автоматическая каталогизация файлов.
2. Просмотр списка книг с их ключевыми метаданными.
3. Возможность просмотра подробной информации о книге, включая метаданные, размер файла, количество страниц и предварительный просмотр.
4. Просмотр статистики по расширениям файлов.
5. Поиск книг по названию, автору, расширению файла и метаданным.
6. Фильтрация списка книг по различным критериям, включая отбор только избранных книг, поиск книг без автора или метаданных, выбор книг с наибольшим числом страниц или размером файла.
7. Возможность обновления информации о книгах в базе данных.
8. Управление списком избранных книг.
9. Просмотр подробных метаданных книги.

Возможности для дальнейшего развития и улучшения приложения:

Расширение поддерживаемых форматов. В настоящий момент поддерживаются только четыре типа файлов. Можно расширить список поддерживаемых форматов для анализа и каталогизации еще большего числа книг.

Улучшение пользовательского интерфейса. Пользовательский интерфейс может быть улучшен для обеспечения большего удобства использования, включая лучшую навигацию, возможность настройки вида и сортировки списков книг, и так далее.

Улучшение поиска. Система поиска может быть улучшена, чтобы предоставлять более точные и полезные результаты. Это может включать в себя поддержку поиска по нескольким критериям одновременно, использование фильтров и расширенного поиска.

Интеграция с облачными хранилищами. Можно добавить возможность сканирования и каталогизации книг, хранящихся в облачных хранилищах, таких как Google Drive или Dropbox.

Поддержка многопользовательской работы. Возможность использования приложения несколькими пользователями одновременно может быть полезной функцией, особенно для больших библиотек или групп пользователей.

Интеграция с внешними базами данных книг. Возможность связи с внешними базами данных, такими как Google Books или Open Library, может предоставить дополнительные возможности для получения и обогащения информации о книгах.