МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение
высшего образования

«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет	*Кафедра*
вычислительной техники	«Вычислительная техника»

Направление подготовки: 09.03.01 «Информатика и вычислительная техника»
Профиль подготовки: «Программное обеспечение средств вычислительной техники и автоматизированных систем»

БАКАЛАВРСКАЯ РАБОТА

на тему

СИСТЕМА ПОЛНОТЕКСТОВОГО ПОИСКА ПО PDF-ДОКУМЕНТАМ С РАСПОЗНАВАНИЕМ ТЕКСТА НА ИЗОБРАЖЕНИЯХ

Студент 30.05.25 Столяров Артем Алексеевич
(подпись, дата) (ФИО полностью)

Руководитель 30.05.25 Сорокин А.А.
(подпись, дата) (фамилия, инициалы)

Нормоконтролёр 30.05.25 Бычков А.С.
(подпись, дата) (фамилия, инициалы)

Работа допущена к защите (протокол зас. кафедры от 28.05.2025 г. № 16)

Заведующий кафедрой _____ М.А. Митрохин
(подпись) (фамилия, инициалы)

Работа защищена с отметкой отлично (протокол заседания ГЭК от 2 июня 2025 г. № 7)

Секретарь ГЭК _____ Е.И. Гурин
(подпись) (фамилия, инициалы)

Пенза 2025

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет вычислительной техники

Кафедра «Вычислительная техника»

«Утверждаю»
Заведующий кафедрой
_____М.А. Митрохин
«6» мая 2025 г.

ЗАДАНИЕ
НА БАКАЛАВРСКУЮ РАБОТУ

1. Студент Столяров А.А. гр. 21ВВП2 факультета ВТ

Столяров Артем Алексеевич

2.Тема работы Система полнотекстового поиска по PDF-документам с распознаванием текста на изображениях

Тема утверждена приказом ПГУ № 601/о от " 6 " Мая 2025 г.

3. Руководитель бакалаврской работы Сорокин А.А.

4. Задание на работу (назначение разработки, исходные данные и т.п.)
Разработать систему полнотекстового поиска по PDF - документам с распознаванием текста на изображениях. Программа должна корректно обрабатывать PDF файлы и выполнять поиск по ним. Должна иметь интуитивно понятный интерфейс.

5. Перечень подлежащих разработке вопросов

Требуется провести анализ предметных областей: проанализировать актуальность разработки, изучить существующие аналоги, выделить их достоинства и недостатки. Выбрать язык программирования, а также среду разработки. Описать архитектуру проекта, выбрать и настроить инструменты, которые будут использоваться для написания программы. Разработать программу, описать реализованные функции, проанализировать результаты.

6. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1	Изучение предметной области	24	06.05.2025 – 08.05.2025	
2	Анализ аналогов	32	09.05.2025 – 12.05.2025	
3	Требования к программе	24	13.05.2025 – 15.05.2025	
4	Выбор инструментов разработки	16	16.05.2025 – 17.05.2025	
5	Разработка программы	64	18.05.2025 – 25.05.2025	
6	Тестирование программы	16	26.05.2025 – 27.05.2025	
7	Оформление пояснительной записки	24	28.05.2025– 30.05.2025	

Дата выдачи задания " 6 " мая 2025 г.

Руководитель бакалаврской работы 06.05.25 Сорокин А.А.
(подпись, дата)

Задание к исполнению принял студент 06.05.25 Столяров А.А.
(подпись, дата)

Работу к защите допустить
Декан факультета 30.05.2025 Фионова Л.Р.
(подпись, дата)

Реферат

Бакалаврская работа содержит 105 листов, 56 рисунков, 3 таблицы, 13 источников, 3 приложения.

ПОЛНОТЕКСТОВЫЙ ПОИСК, ИЗВЛЕЧЕНИЕ ТЕКСТА, ИЗВЛЕЧЕНИЕ ТЕКСТА ИЗ ИЗОБРАЖЕНИЙ, СУММАРИЗАЦИЯ ТЕКСТА, TESSERACT, MANTICORE SEARCH, SUMY, VISUAL STUDIO

Цель работы – разработать систему полнотекстового поиска по PDF-документам с распознаванием текста на изображениях.

Одним из ключевых решений для реализации поставленной задачи стало создание программы, которая обеспечивает извлечение текста, а также извлечение текста с изображений, создание краткого содержания, индексирование и полнотекстовый поиск по индексированным данным, имеющая интуитивно-понятный интерфейс для просмотра результатов поиска.

В результате разработана программа, осуществляющая полнотекстовый поиск по PDF-документам. Для достижения результата применены технологии: C#, python, Tesseract OCR, Manticore Search, Sumy.

Содержание

Перечень сокращений и обозначений 7

Введение 8

1 Исследование предметной области 9

1.1 Анализ темы 9

1.2 Обзор аналогов 11

1.3 Постановка задачи 16

2 Проектирование системы 18

2.1 Архитектура программы 18

2.2 Выбор инструментов разработки 21

2.2.1 Модуль парсинга PDF 22

2.2.2 Модуль распознавания текста OCR 23

2.2.3 Модуль суммаризации текста 24

2.2.4 Модуль индексирования 24

2.2.5 Модуль поиска 25

3 Разработка системы 26

3.1 Сравнение и выбор библиотеки для парсинга PDF и OCR 26

3.2 Разработка модуля суммаризации текста 45

3.4 Разработка модуля индексирования 48

3.5 Разработка интерфейса 55

4 Тестирование программы 59

5 Безопасность жизнедеятельности 64

5.1 Правила безопасности при использовании ПК 64

Заключение 65

Список использованных источников 66

Приложение А – UML-диаграммы 68

Приложение Б – Листинг программы 73

Приложение В – Презентация 101

Перечень сокращений и обозначений

ИИ – искусственный интеллект

ОС – операционная система

Парсинг – извлечение информации

ПО – программное обеспечение

Суммаризация – создание краткого содержания исходного текста

ЦП – центральный процессор

ID (Identifier) – идентификатор

IDE (Integrated Development Environment) – интегрированная среда разработки

RAM (Random Access Memory) – оперативная память

UI (User Interface) – пользовательский интерфейс

UML (Unified Modeling Language) – унифицированный язык моделирования

URI (Uniform Resource Identifier) – унифицированный идентификатор ресурса

Введение

С ростом объема цифровых документов возникает потребность в эффективных средствах для их обработки и поиска. Существует большое множество типов документов, которые содержат различные виды информации, такие как научные документы, техническая документация, юридические и финансовые отчеты, а также сканированные книги и архивы. Однако большая часть данных хранится и передается в формате PDF файлов. Современные организации и пользователи ежедневно сталкиваются с большим количеством текстовой информации, представленной в вышеописанном формате, которую необходимо обрабатывать и достаточно быстро находить нужные файлы для более эффективной работы. Ежегодно только в Outlook открывают более двух миллиардов PDF файлов, а в электронной почте и Google Drive ежедневно сохраняют более 70 миллионов новый файлов.

1 Исследование предметной области

1.1 Анализ темы

PDF (Portable Document Format) изначально разрабатывался как формат для неизменяемого представления документов, а сами файлы имеют различные типы, такие как программно-генерируемые, которые создаются на компьютере при помощи ПО, которые могут содержать изображения, текст и ссылки. Так же есть тип отсканированных документов, такие PDF получаются благодаря сканера или же мобильного приложения. Эти файлы являются обычным набором изображений, которые помещены вместе в один PDF файл, поэтому на них нельзя выбрать текст, ссылку и выполнить по ним поиск. Последний тип, который используется так же наиболее часто – это отсканированные документы с OCR. В этом случае после сканирования применяется специальное ПО для распознавания символов. Файл так же представлен в формате набора изображений, но ПО добавляет слой текста, благодаря чему при просмотре файла можно выбирать его отдельные компоненты.

В результате возникает проблема работы с большим количеством PDF файлов, когда человеку необходимо достаточно быстро найти нужные файлы из всех что представлены у него на ПК. Это может возникнуть как у обычного пользователя, так и у крупной или небольшой компании, где количество файлов может быть колоссальным, поэтому для сокращения времени и упрощения этой задачи и возникает необходимость использования системы полнотекстового поиска по PDF – документам. Собственно, конечным пользователем такой системы может являться и простой человек, и компании.

Трудности могут возникнуть на моменте извлечения текста, ведь как было описано выше, PDF может содержать текст в виде встроенного текста, в виде изображений. Для этого применяется парсинг PDF – документов. Это автоматизированный процесс программной обработки документов для

извлечения необходимой информации. С помощью него можно вытаскивать из файлов сам текст, а также изображения из него. Из подходов к парсингу можно выделить парсинг текстового слоя, когда текст хранится в виде Unicode – символов с координатами. Так же некоторые файлы содержат логическую разметку, что позволяет извлекать абзацы, заголовки, списки и тд. Но не все файлы могут иметь разметку и даже при наличии тегов структура может быть нарушена.

Если же PDF файл будет представлен в виде изображений, тут тоже используется парсинг для извлечения изображений, а после применяется оптическое распознавание символов (OCR) – это преобразование бумажных печатных или же рукописных документов в машиночитаемый текст. Качество самого распознавания может зависеть от множества факторов, таких как качество самого скана, на которое может иметь плохое разрешение, шумы или же наклон самой страны, могут встречаться сложные шрифты, а наличие таблиц и формул так же могут дать сложности в распознавании. В этих случаях необходимо применять методы улучшения качества самих изображений, таких как увеличение контраста что бы текст больше выделялся на общем фоне, билинейная фильтрация для удаления шумов а так же выравнивание текста.

Так же весьма удобным было бы автоматическое обобщение текста, ведь человеку достаточно сложно быстро оценить содержание большого PDF документа. Здесь применяется суммаризация текста – это извлечение ключевых идей и предложений. Данная технология позволит значительно сократить время для понимая того, потребуется ли данный файл в дальнейшей работе.

Количество самих PDF файлов может быть огромным, и открывать каждый из них что бы найти нужные задача достаточно трудоемкая. В этом случае полнотекстовый поиск по всем файлам значительно облегчит поставленную задачу, ведь с помощью данного метода можно значительно

сократить временные затраты. Полнотекстовый поиск – это метод поиска информации в базе данных, который позволяет находить нужный текст внутри больших объемов текста. Преимуществом данного метода будет являться то, что анализирует и индексирует весь текст в документе.

1.2 Обзор аналогов

Существуют аналоги для полнотекстового поиска по PDF файлам, которые решают вышеописанные проблемы. Данные аналоги можно разделить на 3 категории:

– облачные сервисы обработки документов;

– коммерческие проекты;

– open-source проекты для корпоративного пользования.

К облачным решениям можно отнести Google Drive, который представляет собой облачное хранилище с системой поиска [1]. Он использует прямое извлечение текстового слоя через PDF.js и имеет индексацию с учетом простой морфологии, но только для английского языка. Для отсканированных документов применяет модифицированный Tesseract OCR. Имеет задержку между загрузкой документа и доступностью для его поиска в 3-20 минут. Максимальный размер файла, который он может обработать ограничивается 100 МБ. Так же не поддерживаются рукописные тексты. Является платным решением – 6$ в месяц, что позволяет загружать до 100 ГБ файлов. Отличная интеграция с другими сервисами Google. Не имеет пакетной обработки, что означает что каждый файл индексируется отдельно. Могут возникать проблемы с кодировками в старых документах. Отсутствует русская морфология. Точность обработки 82-87%. В итоге можно сказать, что Google Drive хорошо подходит для базовых потребностей, так как не требует технических навыков, а также доступен на любом устройстве, ведь является облачным сервисом. Серьезным недостатком же будет являться отсутствие сложного поиска по файлам. Аналог показан на рисунке 1.

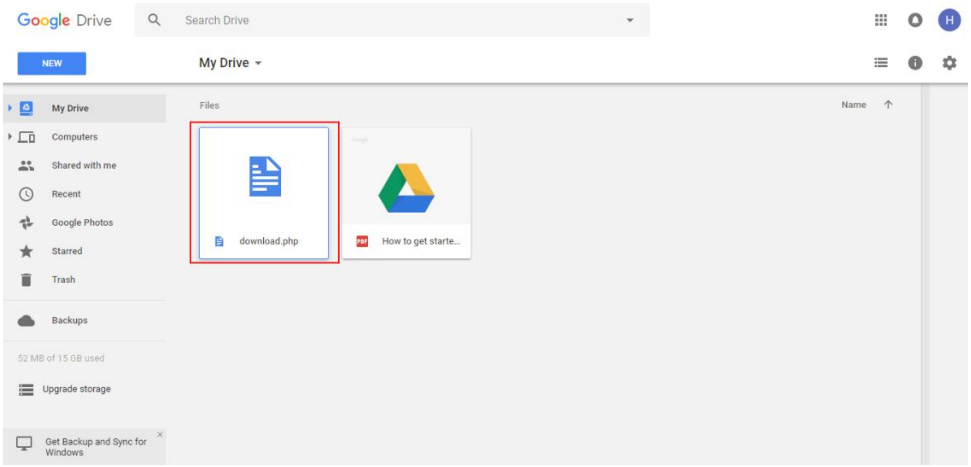


Рисунок 1 – Google Drive

Так же к облачным решениям можно отнести Amazon Textract – сервис, специализирующийся на автоматическом извлечении текста и данных из документов [2]. Данный проект основан на глубоком обучении, используя собственные модели CNN + LSTM, а также поддерживает механизмы для дообучения. Поиск осуществляет с помощью Elastic search. Хорошо обрабатывает текст и анализирует сложные таблицы. Обрабатывает PDF до 500 страниц. По скорости обработки страниц достигает 10 страниц в секунду, а точность достигает 94.3% английских текстов и 89.1 русского текста. Является платным решением со стоимостью 1.5$ за 1000 страниц. Максимальный размер файла – 500 Мб. Показан на рисунке 2.

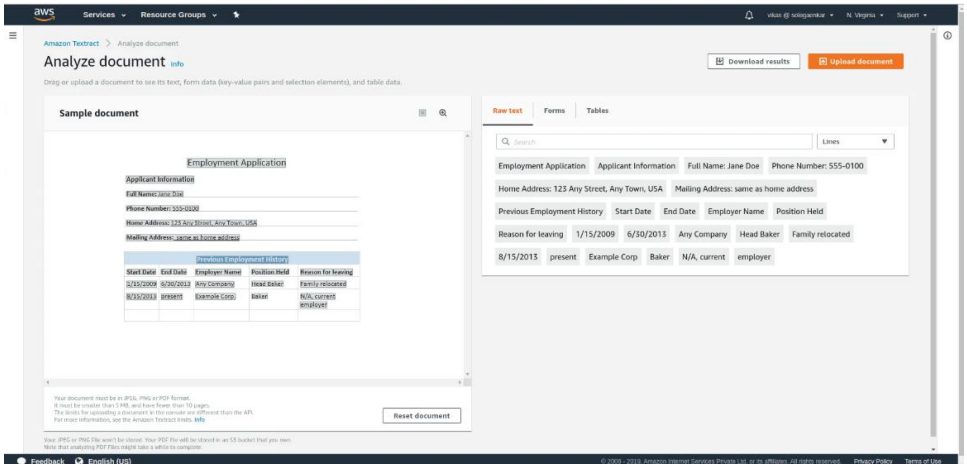


Рисунок 2 - Amazon Textract

К коммерческим проектам можно отнести Adobe Acrobat Pro DC – это уже более профессиональное решение для работы с PDF документами [3]. Он включает в себя функции редактирования документов, расширенный поиск и OCR. Может осуществлять поиск как по одному документу, так и сразу по 500 документам за раз. Использует OCR собственной разработки Adobe Scan. Поддерживает более 40 языков включая русский. Максимальный размер документа, который можно загрузить составляет 2 Гб. Отлично работает с распознаванием таблиц. Скорость обработки составляет 20 страниц в минуту на процессоре i7-11800H. Точность распознавания же 98% для качественных сканов и 85-90% для старых документов с плохим качеством. Является платным решением со стоимостью 179.88$ в год. Изображен на рисунке 3.

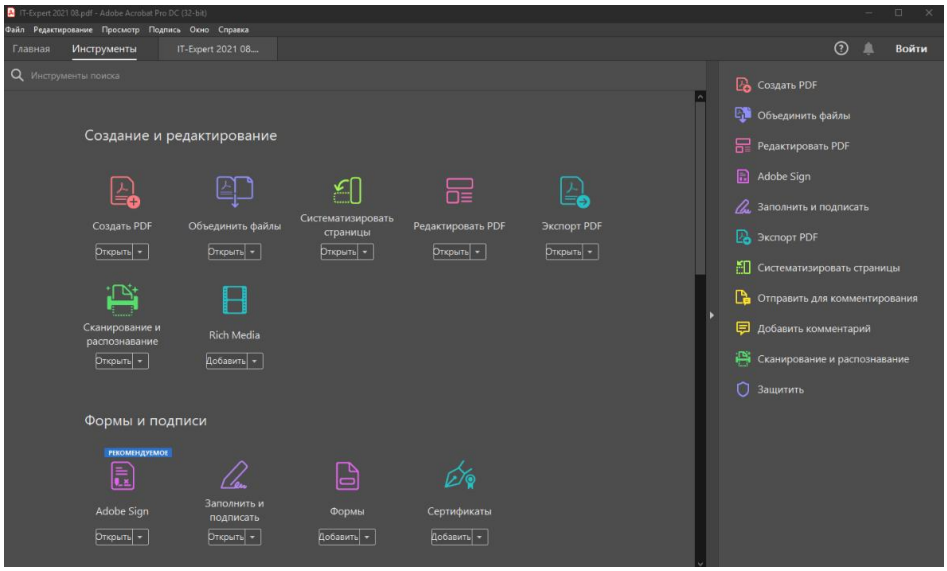


Рисунок 3 - *Adobe Acrobat Pro DC*

Так же к коммерческим проектам относится ABBY FineReader 16 – профессиональное решение для работы с PDF документами [4]. Включает в себя многоязычное распознавание текста (OCR), имеет инструменты для работы с файлами и хорошо сохраняет структуру документов. OCR используется ADRT 3.0, которая поддерживает 192 языка включая русский с точностью 99%. Хорошо работает со сканами с низким качеством. Функционал поиска может осуществляться как в пределах одного документа,

так и в наборе до 1000 файлов, но находит нужный файл только по точному совпадению, так как отсутствует морфология для русского языка. Скорость обработки составляет 30 страниц в минуту на процессоре i7-11800H, а максимальный размер документа 1000 страниц. Максимальный размер загружаемого документа составляет 500 Мб. Является платным решением 299$. Изображен на рисунке 4.

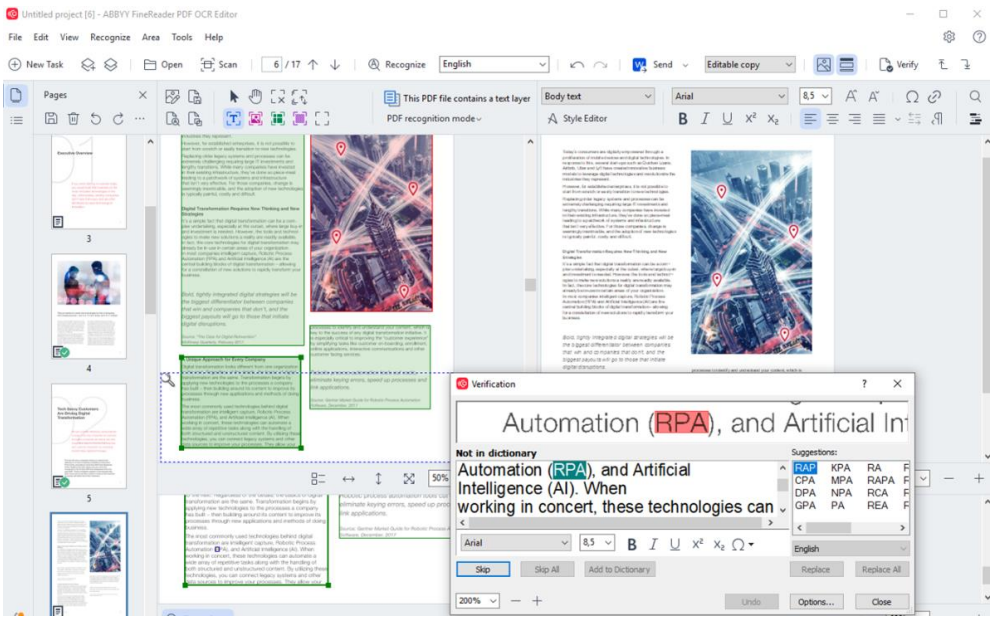


Рисунок 4 - ABBY FineReader 16

К Open-source проектам можно отнести Apache Tika 2.7 +Solr9 – эта связка из двух компонентов, первый обеспечивает извлечение текста и метаданных, второй отвечает за индексацию и поиск [5]. В свою очередь поддерживает не только PDF форматы, но и другие. За OCR здесь используется Tesseract, что обеспечивает точность 88-92%. Полнотекстовый поиск поддерживается с морфологией, нечетким соответствием, а также синонимами. Можно осуществлять поиск по метаданным, датам создания и языкам документа. Что бы работала русская морфология – требуется ручная настройка. Поддерживает миллионы документов в одном индексе. По скорости обработки, предполагая, что файл содержит текст и изображения с текстом, скорость составит 100 страниц в минуту. Максимальный размер

файла около 2 Гб, но рекомендуемый максимум 500 Мб и 10000 страниц. Преимуществами данного решения будет являться то, что оно имеет открытый исходный код, достаточно гибкую схему индексации и хорошо подходит для корпоративных поисковых систем, где большое количество файлов. Недостатками будут являться сложность настройки, высокая требовательность к ресурсам – 8 Гб RAM. Однако данное решение больше подходит для крупных организаций, которые имеют соответствующие технические ресурсы. Изображен на рисунке 4.

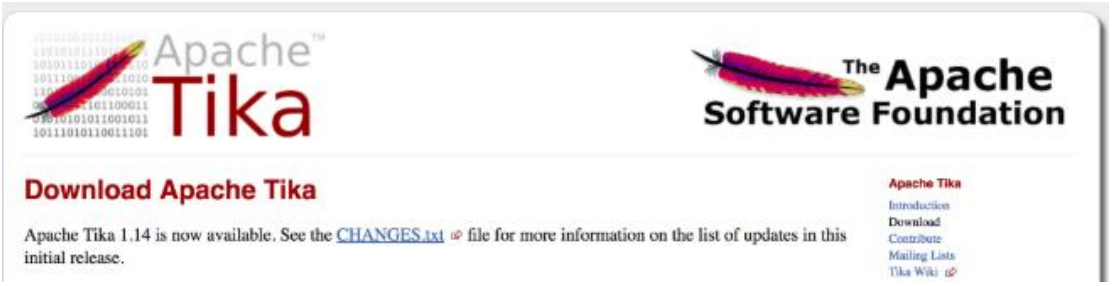


Рисунок 5 - Apache Tika

Так же к Open-source проектам можно отнести Paperless – ngx 2.7 [6]. Это решение делает акцент на автоматическую классификацию файлов, так же использует OCR Tesseract, работает с полнотекстовым поиском и имеет веб-интерфейс для работы. Для полнотекстового поиска используется Elasticsearh, который сам по себе является сильным инструментом для этой задачи, но достаточно сложным в настройке и больше подходит для более серьезных решений так как требует больше ресурсов системы чем тот же Manticore search. Поиск можно выполнять по содержимому документов, названиям файлов, а также тегам и метаданным. Не имеет морфологического поиска для русского языка, а максимальный размер индекса 500000 документов. Скорость обработки достигает 20-30 страниц в минуту. Поддерживает более 50 языков, в том числе русский. Максимально рекомендованный размер документа составляет 200 Мб. Обрабатывает 50-70 документов в час. Из минусов, требует минимум 4 Гб RAM. Так же требует ручной настройки русского языка.

Присутствует задержка в обновлении индекса до 5 минут. Точность распознавания составляет 85-90%. В результате данное решение больше подходит для небольших предприятий, где не требуется обработка большого количества файлов. Показан на рисунке 6.

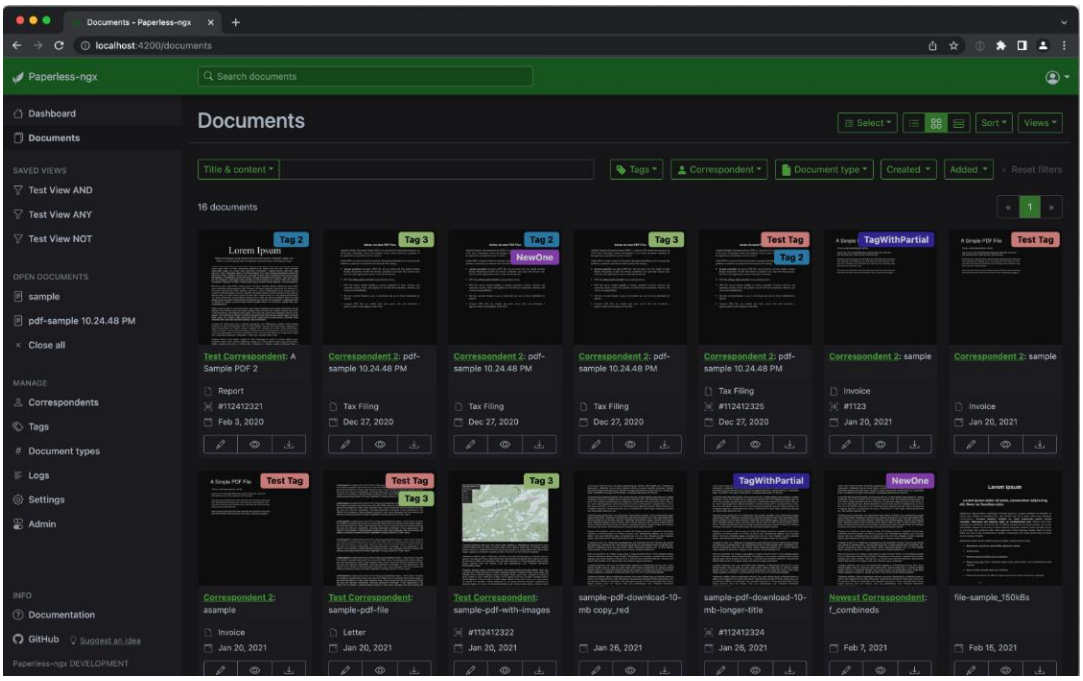


Рисунок 6 - Paperless – ngx

1.3 Постановка задачи

Целью настоящей работы является программная реализация системы полнотекстового поиска по PDF-документам с извлечением текста и распознаванием текста с изображений.

Исходя из сказанного, для достижения поставленной цели необходимо решить ряд основных задач, таких как:

– программа должна корректно и быстро извлекать текст из PDF файлов;

– необходимо использовать OCR для распознавания текста с изображений;

– приложение должно выполнять поиск по обработанным файлам;

– должна быть возможность загрузки как одного файла, так и директории файлов для последующей обработки;

– должна быть возможность прервать процесс обработки файлов;

– приложение должно иметь автоматическую функцию обобщения текста;

– должен быть реализован просмотр PDF файла в самом приложении;

– приложение должно иметь интуитивно понятный интерфейс.

UML-диаграмма вариантов использования представлена на рисунке 7.

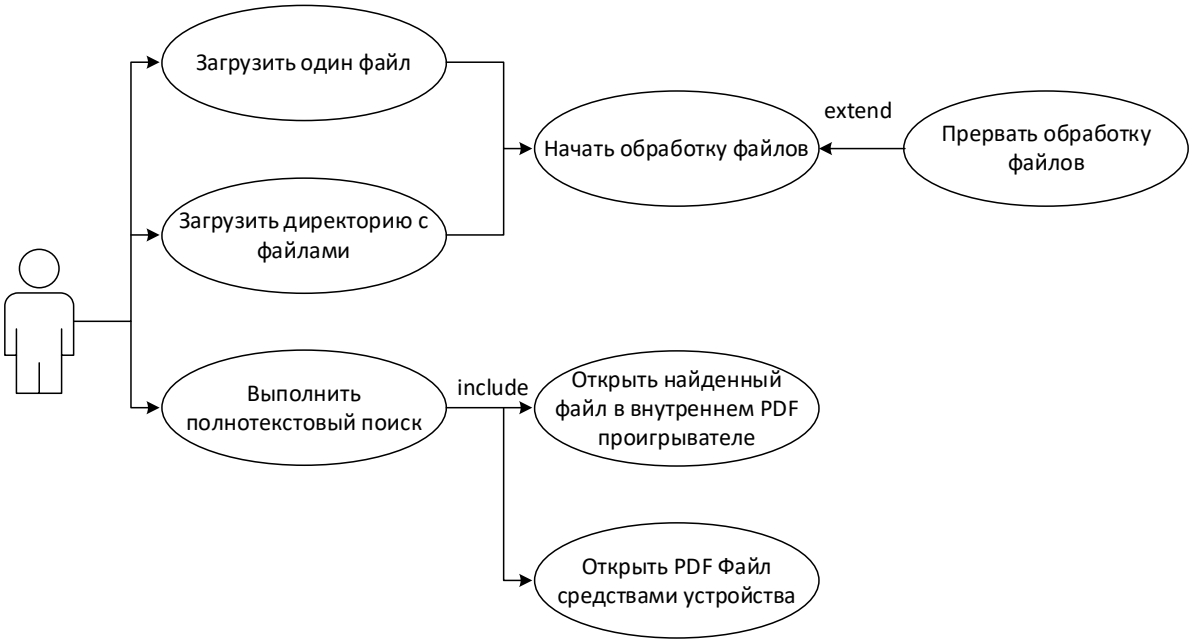


Рисунок 7 – UML-диаграмма вариантов использования

Минимальные системные требования:

– ОС Windows 10 (x64);

– процессор Intel i3 / AMD Ryzen 3 (2+ядра);

– память: 4 Гб ОЗУ;

– место на диске: 500 Мб без учета документов;

– дополнительное ПО: .NET 6.0 Runtime, Python 3.10.

2 Проектирование системы

2.1 Архитектура программы

Выбор архитектуры является критически важным этапом разработки любого приложения для ОС Windows. Она определяет, каким образом компоненты системы будут взаимодействовать между собой, как будут обрабатываться данные. Выбор неправильной архитектуры может создать значительные проблемы: ухудшится читаемость кода, а также общее понимание самой логики программы. Все это неизбежно приведет к росту числа ошибок, а также некорректной работе приложения в целом. Правильная архитектура же, в свою очередь, позволит обеспечить удобство поддержки самого решения, оптимально использовать ресурсы, а также повысит надежность самого приложения.

Исходя из требований, которые были описаны выше, было принято решение использовать за архитектуру монолит с логически выделенными слоями – когда все компоненты развертываются как единое целое, хотя сама система разделена на слои без жесткой изоляции. Так же в нашем случае будут использоваться внешние сервисы, что делает архитектуру монолита куда сложнее классического, можно сказать, получится гибридный монолит. Преимуществами такого выбора будет являться сочетанием плюсов монолитной архитектуры и использования внешних сервисов. Монолитная часть позволит быстро взаимодействовать между модулями, и будет иметь минимальные накладные расходы на межмодульное взаимодействие. Что касается внешних сервисов, то их использование в проекте позволит пользоваться уже проверенными инструментами, которые специализированы в своей области и имеют мощные функции для решения конкретных задач. Также немало важным фактом будет являться то, что при необходимости всегда будет возможность заменить какой-либо отдельный компонент без переписывания всей системы. Монолитную часть так же можно тестировать

изолированно, допустим, проводить тесты для PDF – парсера, что значительно упростит разработку самого приложения.

Основными модулями системы будут являться:

– модуль парсинга PDF;

– модуль распознавания текста OCR;

– модуль суммаризации текста;

– модуль индексирования;

– модуль поиска.

Детализация самих модулей описана ниже.

Модуль парсинга PDF предназначен для извлечения текста из PDF файлов. Извлечение должно происходить с сохранением структуры самого текста, то есть заголовки, абзацы, списки должны идти в той же последовательности, что и в оригинальном PDF файле, иначе будет нарушена логика самого текста, что в дальнейшем скажется на суммаризации текста, а также последующем полнотекстовом поиске. Также данный модуль должен выделять изображения с их координатами в документе – чтобы связать текст с визуальным контекстом, в этом случае также имеется необходимость в последующем вставлять текст с изображения в нужное место в итоговом результате парсинга. В этом модуле должна обеспечиваться потоковая обработка, что в свою очередь позволит работать с файлами размер которых более 1 Гб без загрузки в память.

Модуль распознавания текста OCR предназначен для распознавания текста из изображений, которые находятся в PDF документе. Распознанный текст должен вставляться в нужной последовательности, как и в оригинальном документе. Необходимо предусмотреть сохранение координат текстовых блоков – это как раз нужно для того, чтобы понимать, где текст находился в исходном файле. Должно быть автоматическое определение языка текста, поддержка русского и английского текста. Запуск должен производится как

Nuget-библиотека, которая позволяет работать с самой OCR. Требуются предустановленные языковые данные, как раз для работы с языками.

Модуль суммаризации реализует функционал сжимания полученного текста, сохраняя саму суть. Необходимо использовать extractive-подход, то есть выделять готовые предложения, а не генерировать новые. Так же необходима конфигурируемая степень сжатия самого текста.

Модуль индексирования предназначен для обработки данных после парсинга. Должна поддерживаться индексация текста с морфологией русского языка.

Модуль поиска предназначен для и выполнения полнотекстового поиска по индексированным документам. Необходимо быстрое время ответа по поиску.

UML-диаграмма взаимодействия модулей представлена на рисунке 8.

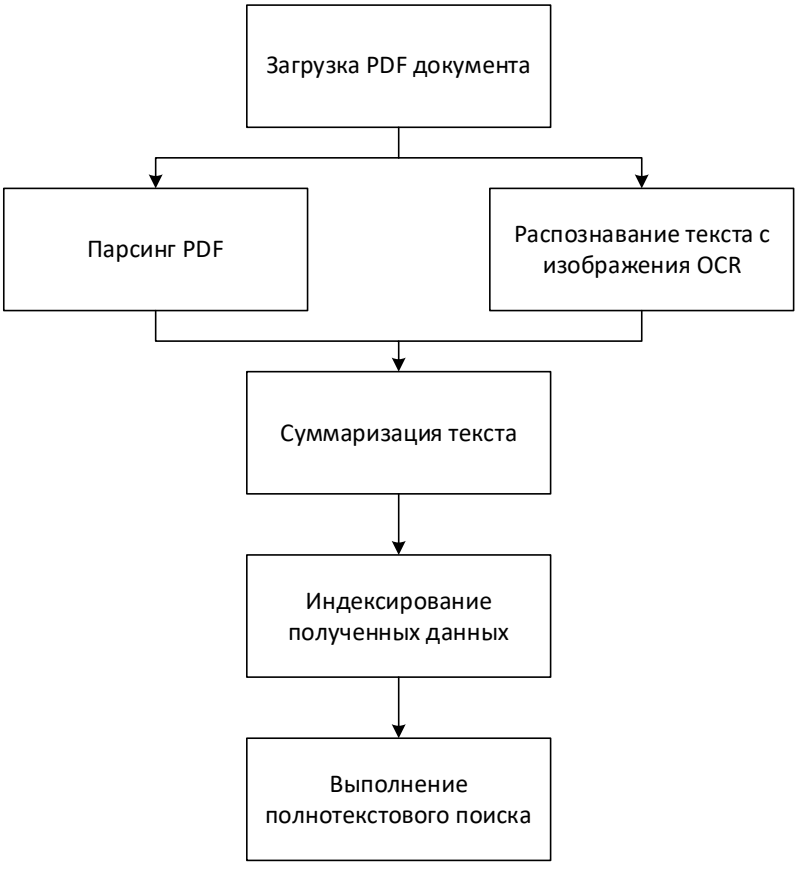


Рисунок 8 – UML - диаграмма взаимодействия модулей

Модули взаимодействуют друг с другом. Их взаимодействие представляется в последовательной работе каждого модуля. Сначала PDF парсер извлекает текст и изображения. Изображения передаются в OCR из которого возвращается результат в виде распознанного текста. Далее текст склеивается и передается в суммаризатор, который сжимает весь текст и создает его краткую версию. После всех действий все данные объединяются и отправляются в модуль индексирования и поиска.

2.2 Выбор инструментов разработки

Теперь, когда модули были спроектированы, необходимо выбрать инструменты разработки, которые будут соответствовать всем указанным требованиям. Язык программирования выбран C# - это кросс-платформенный объектно-ориентированный язык программирования, разработанный компанией Microsoft в начале 2000х годов как часть платформы .NET [7]. Данный язык отлично подойдет под поставленные задачи. Отличным выбором среды разработки для написания программы на C# будет являться Visual Studio – это интегрированная среда разработки, которая также была создана компанией Microsoft. В этой IDE имеется полный набор инструментов и функций для улучшения и усовершенствования каждого этапа разработки программного обеспечения - например отладка и тестирование кода, что делает процесс разработки более эффективным и удобным. Редактор исходного кода поддерживает технологию IntelliSence и возможность простейшего рефакторинга кода. Также отладчик может работать в двух режимах: как отладчик уровня исходного кода и как отладчик машинного уровня [8]. Visual Studio позволяет разрабатывать как консольные программы, так и программы с графическим интерфейсом GUI. Графический интерфейс может быть реализован с помощью Windows Forms. Данная среда имеет возможность достаточно просто интегрировать работу сторонних библиотек,

что значительно упростит работу с ними [9]. Как раз за счет использования сторонних библиотек будут реализованы вышеописанные модули.

2.2.1 Модуль парсинга PDF

Для реализации модуля извлечения текста из документа PDF существует ряд библиотек для языка C#, которые позволяют получить текст.

Чтобы сравнить инструменты для извлечения текста будет применяться ряд критериев, благодаря которым и будет осуществлен выбор.

Критерии:

– скорость обработки;

– работа со сложной разметкой;

– работа с большими файлами;

– сложность интегрирования в проект.

Подходящие библиотеки и их краткие характеристики представлены в таблице 1.

Таблица 1 – библиотеки для парсинга PDF

Название	Версия	Размер (Мб)	Примечание
Itext7	9.1.0v	20-30	Имеет высокую производительность без особых настроек. Хорошо справляется со сложной разметкой. Необходима покупка лицензии для закрытых проектов.
PDFSharp	1.5v	3-5	Имеет ограниченную поддержку извлечения текста, подходит для простых документов. Лицензия не нужна.
PDFPig	1.0.0v	2-3	Хорошо извлекает текст, а также работает со сложной разметкой. Лицензия не нужна.
PdfiumViewer	1.0v	15-20	Хорошо извлекает простые и средние по сложности документы. Перестала обновляться. Проблемы с извлечением текста с нестандартной разметкой. Лицензия не нужна.

Название	Версия	Размер (Мб)	Примечание
PDFBox	2.0.15v	20-30	Не очень хорошо работает со сложной разметкой, но нормально с извлечением изображений. Лицензия не нужна.

Как можно видеть из сравнения библиотек, наиболее хорошо выделяются такие как Itext7 и PDFPig. Хоть Itext7 и требует лицензии для использования в закрытых проектах, но его можно бесплатно использовать в Open-source проектах. Остановим выбор на этих двух библиотеках и в дальнейшем сравним, какая из них лучше справляется с поставленными задачами.

2.2.2 Модуль распознавания текста OCR

Для реализации модуля распознавания текста OCR существует не так много программ и библиотек для работы с ними. Из всех можно выделить Tesseract и EasyOCR, остальные же являются либо платными, либо зависят от Tesseract.

Tesseract хорошо справляется с распознаванием текста на изображениях среднего и высокого качества. Сам по себе это программный движок, который имеет открытый исходный код. Разработан компанией Google. Имеет поддержку широкого спектра языков для распознавания – более 130 [10]. Для русского языка необходимо дополнительно установить пакет с русскими словами, буквами. Можно применить множество настроек, благодаря чему можно улучшить качество самого распознавания. Сам он предназначен для языка C++, но на языке C# можно работать с Tesseract через обертку Tesseract.Net.SDK без особых трудностей.

EasyOCR – это библиотека с открытым исходным кодом, которая предназначена для извлечения текста из изображений, делая это просто и эффективно. В ней применяются методы глубокого обучения, а также она поддерживает более 80 языков. Она представляет собой хорошее решение для

проектов на Python, но для интеграции в решение C# проекта не самая удобная, что может значительно усложнить разработку приложения.

В результате, лучшим выбором для решения поставленных задач будет являться использование Tesseract OCR.

2.2.3 Модуль суммаризации текста

Для реализации модуля суммаризации текста отлично подойдет библиотека Sumy – это инструмент, который позволяет автоматически обобщать текста [11]. Данная библиотека разработана для языка Python, но ее можно использовать в проекте C# с помощью вызова Python скрипта. В основе работы данного решения лежит задача резюмирования текста, которая позволяет получать краткие и четкие версии больших объемов информации. Особенностями Sumy будет являться то, что она поддерживает такие методы обобщения, как TextRank, LexRang и LSA. Это решение можно использовать с минимальным количеством кода, благодаря чему можно начать обобщение довольно быстро. Поддерживает русский, английский и другие языки, языки необходимо будет настроить в самом Python скрипте.

2.2.4 Модуль индексирования

Для реализации модуля индексирования существует такая система как Manticore Search. Это достаточно высокопроизводительная поисковая система, которая имеет открытый исходный код. Она отлично справляется с обработкой больших объемов данных и предоставляет быстрый и релевантный поиск по ним. Сама она была создана в 2017 году на основе поисковой системы Sphinx. Было взято все самое лучшее от Sphinx, переработано и улучшено. Является достаточно легковесной и более простой в настройке, а также требующая меньше ресурсов чем аналог Elastic Search. Так же само время полнотекстового поиска значительно превосходит конкурента. Manticore Search имеет полную поддержку SQL, что значительно упрощает интеграцию существующими системами и облегчает разработку программ, которые будут использовать данную систему. Соответственно,

чтобы использовать Manticore в проекте, необходимо делать SQL запросы к ней [12]. Они имеют не совсем стандартный вид, как у обычных запросов, но синтаксис очень похож, и для работы, необходимо так же установить MySQL Client.

Сама индексация отвечает за подготовку данных для их дальнейшего эффективного поиска. Само индексирование подразделяется на сбор данных – когда происходит получение извлеченного текста, его краткое содержание, и создание индекса – запись в базу данных.

2.2.5 Модуль поиска

Для реализации модуля поиска будем использовать ранее описанную систему Manticore Search. Данный модуль отвечает за обработку запросов пользователя и выдачу результатов на основе данных, которые были проиндексированы в Manticore Search.

Сам модуль состоит из следующих компонентов:

– прием и предобработка запроса – полученный запрос приводится к нижнему регистру, удаляются стоп слова;

– формирование SQL запроса – используя обработанный запрос создается SQL-подобный запрос к Manticore Search;

– выполнение поиска – запрос отправляется в Manticore Search, который возвращает результаты поиска.

3 Разработка системы

3.1 Сравнение и выбор библиотеки для парсинга PDF и OCR

Для разработки системы полнотекстового поиска по PDF документам с извлечением текста и распознаванием текста с изображений будем использовать технологии, которые были рассмотрены в процессе выбора инструментов разработки.

Ранее были рассмотрены и выбраны две библиотеки для парсинга PDF файлов, теперь же необходимо сравнить, какая из них справляется лучше и какую из библиотек использовать в дальнейшем при разработке системы.

Для более лучшего сравнения будем проводить тестирование уже совместно с Tesseract, который является OCR для распознавания текста с изображения. Для этого скачаем установщик Tesseract с официального сайта и установим его на компьютер что показано на рисунке 9.

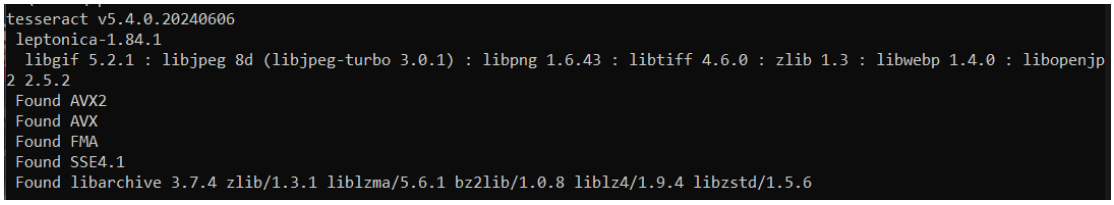


Рисунок 9 – проверка установки Tesseract

Начнем с тестирования PdfPig. Для этого был создан проект в Visual Studio, как консольное приложение, в который с помощью управления пакетами NuGet были установлены необходимые для дальнейшей работы библиотеки что показано на рисунке 10.

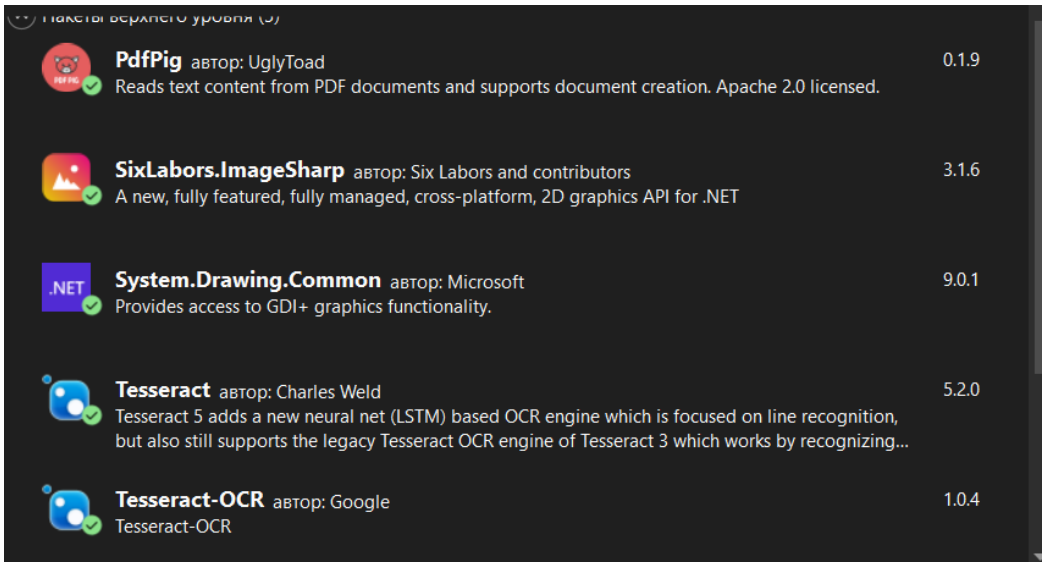


Рисунок 10 – установленные библиотеки

PdfPig – позволит извлекать текст и изображения из PDF документов.

System.Drawing.Common – используется в данном проекте для преобразования набора байт в изображение, так как PdfPig позволяет извлекать изображения в виде байтов, которые затем необходимо преобразовать в само изображение для дальнейшей передачи в Tesseract.

Библиотеки Tesseract здесь же являются оберткой .NET, которые позволяют использовать саму программу при помощи языка C#.

Далее были подготовлены 104 PDF файла для тестирования библиотеки PdfPig, что бы определить, насколько хорошо она справляется с извлечением текста и изображений.

После, была написана программа, которая обрабатывает PDF документы и сохраняет результаты в .txt формате.

Листинг 1 – обработка с помощью PdfPig

```
static void ExtractTextAndImagesFromPdf(string pdfFilePath, TesseractEngine ocrEngine)
    {
        using (var document = PdfDocument.Open(pdfFilePath))
        {
```

```
            for (int pageNumber = 0; pageNumber < document.NumberOfPages; pageNumber++)
            {
                var page = document.GetPage(pageNumber + 1);
                Console.WriteLine($"Page {pageNumber + 1}:");

                var elements = new List<(float yPosition, string content)>();

                // Извлекаем изображения
                foreach (var image in page.GetImages())
                {
                    try
                    {
                        string extractedText = ExtractTextFromImage(image, ocrEngine);
                        if (!string.IsNullOrWhiteSpace(extractedText))
                        {
                            elements.Add((yPosition: (float)image.Bounds.Top, content: $"[Изображение]:\n{extractedText}\n"));
                        }
                    }
                    catch (Exception ex)
                    {
                        Console.WriteLine($"Ошибка обработки изображения на странице {pageNumber + 1}: {ex.Message}");
                    }
                }

                // Извлекаем текст
                var pageText = page.Text;
                if (!string.IsNullOrWhiteSpace(pageText))
                {
```

```
                    elements.Add((yPosition:     0,     content:
pageText));
                }

                elements.Sort((x,              y)              =>
y.yPosition.CompareTo(x.yPosition));

                foreach (var element in elements)
                {
                    Console.WriteLine(element.content);
                }
            }
        }
    }
```

Данная часть кода извлекает изображения и текст из PDF файла.

Листинг 2 – преобразование в PdfPig

```
static    string    ExtractTextFromImage(IPdfImage    pdfImage,
TesseractEngine ocrEngine)
    {
        byte[] imageBytes;

        if (pdfImage.TryGetPng(out imageBytes))
        {
            Console.WriteLine("Изображение   получено   в   PNG-
формате.");
        }
        else
        {
            Console.WriteLine("Не удалось получить PNG. Пробуем
RawBytes.");
            imageBytes   =   pdfImage.RawBytes.Length   >   0   ?
pdfImage.RawBytes.ToArray() : null;

            if (imageBytes == null || imageBytes.Length == 0)
            {
```

```
                Console.WriteLine("Ошибка: изображение пустое.");
                return string.Empty;
            }
        }

        using (var ms = new MemoryStream(imageBytes))
        {
            try
            {
                using (Image<Rgba32> image = Image.Load<Rgba32>(ms))
                {
                    // Улучшение изображения перед OCR
                    image.Mutate(ctx => ctx
                        .Grayscale()
                        .Contrast(1.2f)
                            .BinaryThreshold(0.5f));
                    using (var pix = ConvertToPix(image))
                    {
                        using (var page = ocrEngine.Process(pix))
                        {
                            return page.GetText();
                        }
                    }
                }
            }
            catch (Exception ex)
            {
                Console.WriteLine($"Ошибка загрузки изображения: {ex.Message}");
                return string.Empty;
            }
        }
```

```
        }
    static Pix ConvertToPix(Image<Rgba32> image)
    {
        using (var ms = new MemoryStream())
        {
            image.Save(ms, new PngEncoder());
            return Pix.LoadFromMemory(ms.ToArray());
        }
    }
}
```

В данной части кода происходит преобразование набора байт в изображение, а также предварительное улучшение качества картинки перед отправкой в Tesseract путем перевода в оттенки серого, повышения контрастности и бинаризации.

В итоге получаем следующий результат, представленный на рисунках 11 и 12.

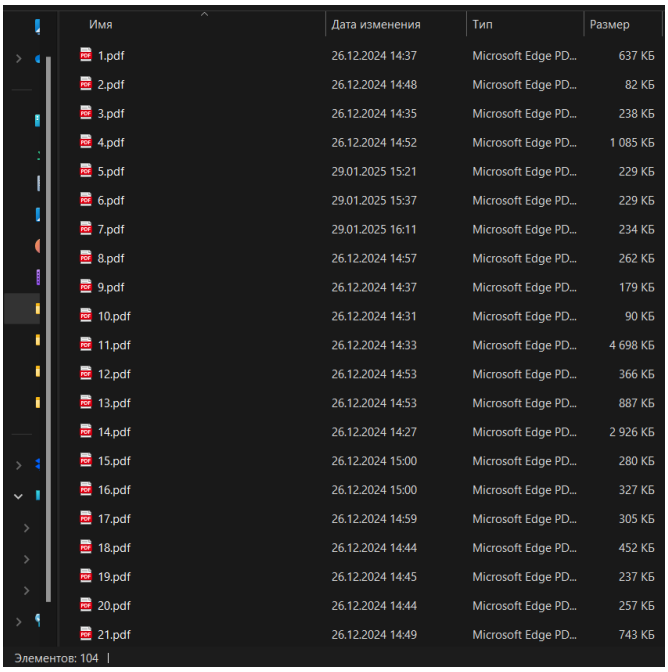


Рисунок 11 – исходные PDF документы

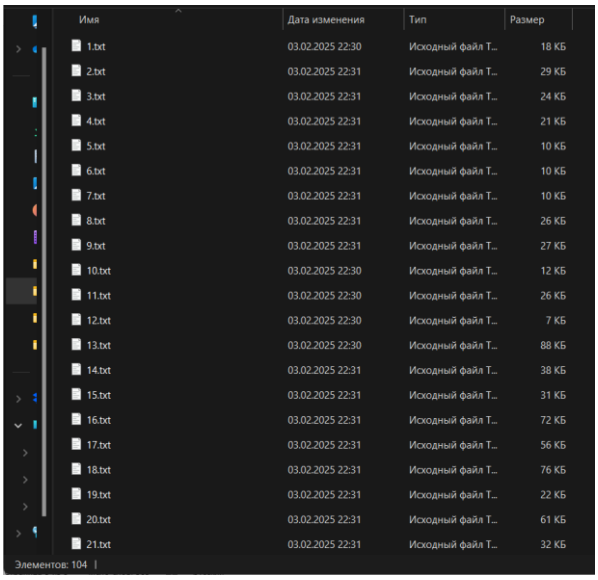


Рисунок 12 – результат обработки

Время обработки 104 файлов составило примерно 67 секунд, что изображено на рисунке 13.

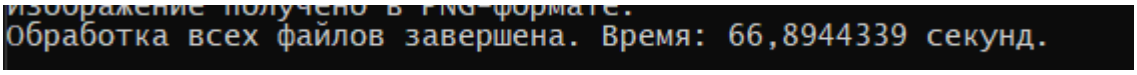
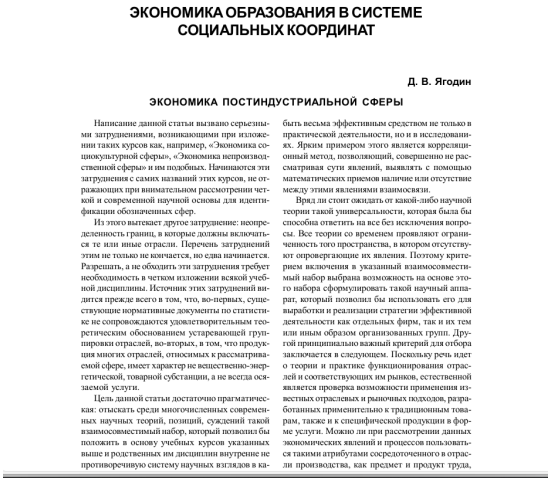


Рисунок 13 – время обработки

Сравним содержание PDF файла с полученным текстом в результате работы программы, что показано на рисунке 14.



ЭКОНОМИКА ОБРАЗОВАНИЯ В СИСТЕМЕ СОЦИАЛЬНЫХ КООРДИНАТ

Д. В. Ягодин

ЭКОНОМИКА ПОСТИНДУСТРИАЛЬНОЙ СФЕРЫ

Написание данной статьи вызвано серьезными затруднениями, возникающими при изложении таких курсов как, например, «Экономика социокультурной сферы», «Экономика непроизводственной сферы» и им подобных. Начинаются эти затруднения с самих названий этих курсов, не отражающих при внимательном рассмотрении четкой и современной научной основы для идентификации обозначенных сфер.

Из этого вытекает другое затруднение: неопределенность границ, в которые должны включаться те или иные отрасли. Перечень затруднений этим не только не кончается, но едва начинается. Разрешать, а не обходить эти затруднения требует необходимость в четком изложении всякой учебной дисциплины. Источник этих затруднений видится прежде всего в том, что, во-первых, существующие нормативные документы по статистике не сопровождаются удовлетворительным теоретическим обоснованием устаревающей группировки отраслей, во-вторых, в том, что продукция многих отраслей, относимых к рассматриваемой сфере, имеет характер не вещественно-энергетической, товарной субстанции, а не всегда осязаемой услуги.

Цель данной статьи достаточно прагматическая: отыскать среди многочисленных современных научных теорий, позиций, суждений такой взаимосовместимый набор, который позволил бы положить в основу учебных курсов указанных выше и родственных им дисциплин внутренне не противоречивую систему научных взглядов в качестве весьма эффективным средством не только в практической деятельности, но и в исследовании. Ярким примером этого является корреляционный метод, позволяющий, совершенно не рассматривая сути явлений, выявлять с помощью математических приемов наличие или отсутствие между этими явлениями взаимосвязи.

Вряд ли стоит ожидать от какой-либо научной теории такой универсальности, которая была бы способна ответить на все без исключения вопросы. Все теории со временем проявляют ограниченность того пространства, в котором отсутствуют опровергающие их явления. Поэтому критерием включения в указанный взаимосовместимый набор выбрана возможность на основе этого набора сформулировать такой научный аппарат, который позволил бы использовать его для выработки и реализации стратегии эффективной деятельности как отдельных фирм, так и их тем или иным образом организованных групп. Другой принципиально важный критерий для отбора заключается в следующем. Поскольку речь идет о теории и практике функционирования отраслей и соответствующих им рынков, естественной является проверка возможности применения известных отраслевых и рыночных подходов, разработанных применительно к традиционным товарам, также и к специфической продукции в форме услуги. Можно ли при рассмотрении данных экономических явлений и процессов пользоваться такими атрибутами сосредоточенного в отрасли производства, как предмет и продукт труда,

Рисунок 14 – исходный PDF файл

Как можно видеть, данный документ имеет сложную разметку, где текст представлен в 2 колонки что показано на рисунке 15.

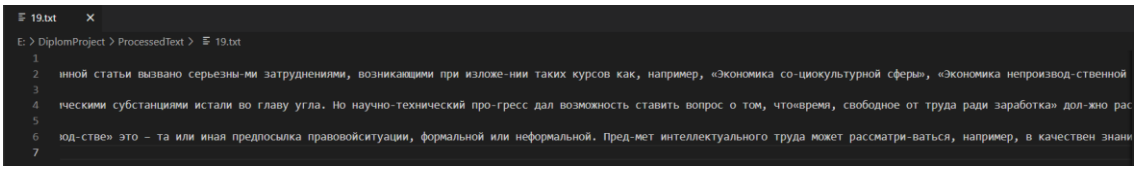


Рисунок 15 – обработанный PDF файл

В файле .txt, который получился в результате обработки, текст разделяется по страницам, и содержание каждой страницы записано в отдельную строку.

Теперь рассмотрим пример парсинга PDF документа, который содержит изображение с текстом что показано на рисунке 16.

Рисунок 16 – PDF файл с изображением

Получается следующий результат, который показан на рисунке 17.

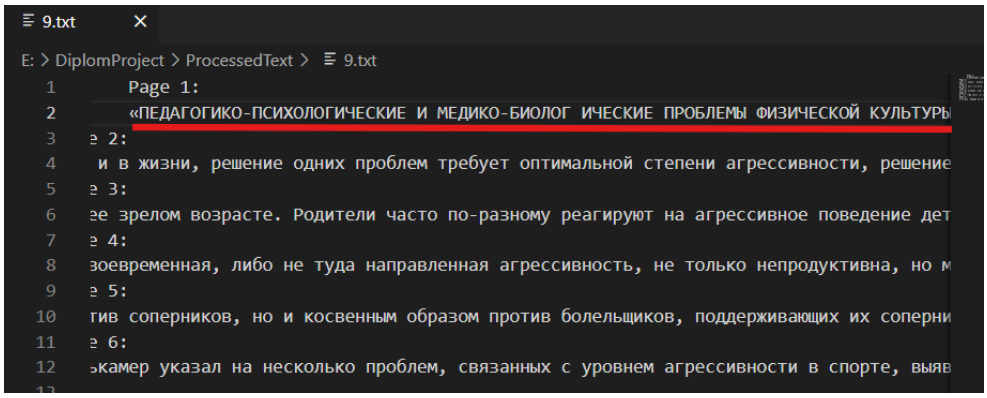


Рисунок 17 – результат обработки

Как можно видеть, текст с изображения был получен и вставлен в основной поток текста.

При попытке разбивать текст на абзацы, как это представлено в самом PDF файле, получившийся результат начинал считываться построчно, из-за чего текст начал «склеиваться», если в файле он был представлен в две и более колонок, последствием этого является то, что текст теряет смысл.

Так же извлечение изображений данная библиотека предоставляет неудобным способом - получением в виде набора байт, которые затем необходимо заново преобразовывать в изображение с использованием другой библиотеки.

Из-за того, что текста в PDF документах имеют разную разметку, библиотека PdfPig не всегда хорошо определяет, что извлекать первым делом, из-за этого текст и текст с изображений могут меняться местами в пределах одной страницы, что может нарушить логический смысл самого текста.

Далее был создан еще один консольный проект Visual Studio, где была протестирована библиотека iText7 (ITextSharp).

По аналогии с прошлым проектом, с помощью управления пакетами NuGet, были подключены необходимые библиотеки, что показано на рисунке 18.

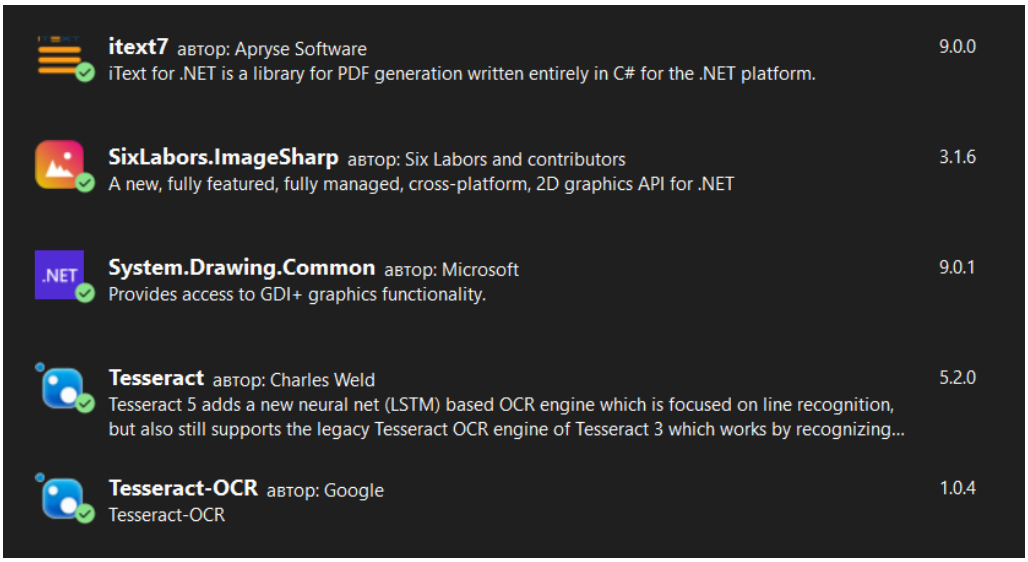


Рисунок 18 – установленные библиотеки

Так же была написана программа, которая выполняла тот же функционал, что и в первом проекте с PdfPig.

Листинг 3 – обработка с помощью iText7

```
public static List<string> ExtractTextAndImagesFromPDF(string filePath)
    {
        List<string> extractedText = new List<string>();

        using (PdfReader reader = new PdfReader(filePath))
        using (PdfDocument pdfDoc = new PdfDocument(reader))
        {
            for (int page = 1; page <= pdfDoc.GetNumberOfPages(); page++)
            {
                var pdfPage = pdfDoc.GetPage(page);

                // Извлекаем текст с каждой страницы
                ITextExtractionStrategy strategy = new SimpleTextExtractionStrategy();
                string pageText = PdfTextExtractor.GetTextFromPage(pdfPage, strategy);
```

```
                // Разделяем текст на блоки
                List<string>      pageTextBlocks      =      new
List<string>(pageText.Split('\n'));

                List<string>      imageTextBlocks      =      new
List<string>();

                // Проверяем, есть ли изображения на странице
перед их обработкой
                if
(pdfPage.GetResources().GetResource(PdfName.XObject) != null)
                {
                    var pageImages = GetImagesFromPage(pdfPage);

                    foreach (var image in pageImages)
                    {
                        string           textFromImage           =
RecognizeTextFromImage(image);
                        if (!string.IsNullOrEmpty(textFromImage))
                        {
                            imageTextBlocks.Add(textFromImage);
                        }
                    }
                }

                // Интегрируем текст с изображений в текст
страницы, сохраняя порядок
                int imageIndex = 0;
                foreach (var block in pageTextBlocks)
                {
                    extractedText.Add(block);     //     Добавляем
обычный текст
                    if (imageIndex < imageTextBlocks.Count)
                    {
```

```
extractedText.Add(imageTextBlocks[imageIndex]);    //    Добавляем
текст с изображений
                        imageIndex++;
                    }
                }
            }
        }

        return extractedText;
    }
```

В данной части кода происходит извлечение текста из PDF документа с учетом изображений, текст разделяется на блоки, а также происходит вставка текста с изображения сохраняя порядок как в оригинальном PDF файле.

Листинг 4 – преобразование в iText7

```
public static List<byte[]>
GetImagesFromPage(iText.Kernel.Pdf.PdfPage page)
    {
        List<byte[]> images = new List<byte[]>();
        try
        {
            PdfCanvasProcessor processor = new
PdfCanvasProcessor(new ImageExtractionEventListener(images));
            PdfResources resources = page.GetResources();
            processor.ProcessContent(page.GetContentBytes(),
resources);
        }
        catch (Exception ex)
        {
            Console.WriteLine($"Ошибка при извлечении
изображений: {ex.Message}");
        }
        return images;
```

```
    }
    // Слушатель для извлечения изображений
    public class ImageExtractionEventListener : IEventListener
    {
        private List<byte[]> _images;
        public ImageExtractionEventListener(List<byte[]> images)
        {
            _images = images;
        }
        public void EventOccurred(IEventData data, EventType
type)
        {
            // Проверяем, является ли событие изображением
            if (type == EventType.RENDER_IMAGE)
            {
                var renderInfo = data as ImageRenderInfo;
                if (renderInfo != null)
                {
                    var img = renderInfo.GetImage();
                    if (img != null)
                    {
                        byte[] imageBytes = img.GetImageBytes();
                        _images.Add(imageBytes);
                        // Сохраняем изображение в файл для
отладки

File.WriteAllBytes($"extracted_image_{Guid.NewGuid()}.png",
imageBytes);
                    }
                }
            }
        }
        public ICollection<EventType> GetSupportedEvents()
        {
```

```
            // Мы интересуемся только событием RENDER_IMAGE
            return new List<EventType> { EventType.RENDER_IMAGE
};
        }
    }
    public static Image<Rgba32> EnhanceImage(byte[] imageBytes)
    {
        using (var stream = new MemoryStream(imageBytes))
        {
            var image = Image.Load<Rgba32>(stream);
            // Проверяем, что глубина цвета не 4 бита (например,
8 бит и выше)
            if (image.PixelType.BitsPerPixel < 8)
            {
                Console.WriteLine($"Пропущено изображение с
низкой глубиной цвета ({image.PixelType.BitsPerPixel} бит)");
                return null;
            }
            image.Metadata.ExifProfile = null;
            image.Metadata.IccProfile = null;
            // Улучшение изображения
            image.Mutate(ctx => ctx
                .Grayscale()
                .Contrast(1.2f)
                .BinaryThreshold(0.5f));
            return image;
        }
    }
    public static string RecognizeTextFromImage(byte[]
imageBytes)
    {
        string result = string.Empty;

        if (imageBytes == null || imageBytes.Length == 0)
```

```
            return result;
        try
        {
            var enhancedImage = EnhanceImage(imageBytes);

            if (enhancedImage == null) // Если изображение не
поддерживается
                return result;
            using (var enhancedStream = new MemoryStream())
            {
                enhancedImage.Save(enhancedStream, new
PngEncoder());
                byte[] enhancedImageBytes =
enhancedStream.ToArray();
                string tessdataPath =
@"E:\tesseractorc\tessdata";
                using (var engine = new
TesseractEngine(tessdataPath, "rus+eng", EngineMode.Default))
                {
                    using (var pix =
Pix.LoadFromMemory(enhancedImageBytes))
                    {
                        var ocrResult = engine.Process(pix);
                        result = ocrResult.GetText();
                    }
                }
            }
        }
        catch (Exception ex)
        {
            Console.WriteLine($"Ошибка OCR: {ex.Message}");
        }
        return result;
    }
```

В данной части кода происходит распознавание текста с помощью Tesseract, так же запускается слушатель для извлечения изображений и улучшается сама картинка перед отправкой в Tesseract теми же способами, что и в предыдущем проекте.

Как можно видеть, код получился сложнее.

Время обработки тех же 104 PDF файлов составило примерно 106 секунд, что показано на рисунке 19.

```
Обработка завершена. Время выполнения: 105,7601698 секунд.
```

Рисунок 19 – время обработки

Рассмотрим результаты обработки тех же файлов, которые ранее были обработаны с помощью другой библиотеки, что показано на рисунках 20 и 21.

ЭКОНОМИКА ОБРАЗОВАНИЯ В СИСТЕМЕ СОЦИАЛЬНЫХ КООРДИНАТ

Д. В. Ягодин

ЭКОНОМИКА ПОСТИНДУСТРИАЛЬНОЙ СФЕРЫ

Написание данной статьи вызвано серьезными затруднениями, возникающими при изложении таких курсов как, например, «Экономика социокультурной сферы», «Экономика непроизводственной сферы» и им подобных. Начинаются эти затруднения с самих названий этих курсов, не отражающих при внимательном рассмотрении четкой и современной научной основы для идентификации обозначенных сфер.

Из этого вытекает другое затруднение: неопределенность границ, в которые должны включаться те или иные отрасли. Перечень затруднений этим не только не кончается, но едва начинается. Разрешать, а не обходить эти затруднения требует необходимость в четком изложении всякой учебной дисциплины. Источник этих затруднений видится прежде всего в том, что, во-первых, существующие нормативные документы по статистике не сопровождаются удовлетворительным теоретическим обоснованием устаревающей группировки отраслей, во-вторых, в том, что продукция многих отраслей, относимых к рассматриваемой сфере, имеет характер не вещественно-энергетической, товарной субстанции, а не всегда осязаемой услуги.

Цель данной статьи достаточно прагматическая: отыскать среди многочисленных современных научных теорий, позиций, суждений такой взаимосовместимый набор, который позволил бы положить в основу учебных курсов указанных выше и родственных им дисциплин внутренне не противоречивую систему научных взглядов в качестве весьма эффективным средством не только в практической деятельности, но и в исследованиях. Ярким примером этого является корреляционный метод, позволяющий, совершенно не рассматривая сути явлений, выявлять с помощью математических приемов наличие или отсутствие между этими явлениями взаимосвязи.

Вряд ли стоит ожидать от какой-либо научной теории такой универсальности, которая была бы способна ответить на все без исключения вопросы. Все теории со временем проявляют ограниченность того пространства, в котором отсутствуют опровергающие их явления. Поэтому критерием включения в указанный взаимосовместимый набор выбрана возможность на основе этого набора сформулировать такой научный аппарат, который позволил бы использовать его для выработки и реализации стратегии эффективной деятельности как отдельных фирм, так и их тем или иным образом организованных групп. Другой принципиально важный критерий для отбора заключается в следующем. Поскольку речь идет о теории и практике функционирования отраслей и соответствующих им рынков, естественной является проверка возможности применения известных отраслевых и рыночных подходов, разработанных применительно к традиционным товарам, также и к специфической продукции в форме услуги. Можно ли при рассмотрении данных экономических явлений и процессов пользоваться такими атрибутами сосредоточенного в отрасли производства, как предмет и продукт труда,

Рисунок 20 – исходный PDF файл

19.txt

E: > DiplomProject > Results > 19.txt

```
З Экономика образования № 3, 2007
ЭКОНОМИКА ОБРАЗОВАНИЯ В СИСТЕМЕ
СОЦИАЛЬНЫХ КООРДИНАТ
Д. В. Ягодин
ЭКОНОМИКА ПОСТИНДУСТРИАЛЬНОЙ СФЕРЫ
© Д. В. Ягодин, 2007
Написание данной статьи вызвано серьезны-
ми затруднениями, возникающими при изложе-
нии таких курсов как, например, «Экономика со-
циокультурной сферы», «Экономика непроизвод-
ственной сферы» и им подобных. Начинаются эти
затруднения с самих названий этих курсов, не от-
ражающих при внимательном рассмотрении чет-
кой и современной научной основы для иденти-
фикации обозначенных сфер.
Из этого вытекает другое затруднение: неопре-
деленность границ, в которые должны включать-
ся те или иные отрасли. Перечень затруднений
этим не только не кончается, но едва начинается.
Разрешать, а не обходить эти затру днения требует
необходимость в четком изложении всякой учеб-
ной дисциплины. Источник этих затруднений ви-
дится прежде всего в том, что, во-первых, суще-
ствующие нормативные документы по статисти-
ке не сопровождаются удовлетворительным тео-
ретическим обоснованием устаревающей груп-
пировки отраслей, во-вторых, в том, что продук-
ция многих отраслей, относимых к рассматрива-
емой сфере, имеет характер не вещественно-энер-
гетической, товарной субстанции, а не всегда ося-
заемой услуги.
Цель данной статьи достаточно прагматичес-
кая: отыскать среди многочисленных современ-
ных научных теорий, позиций, суждений такой
взаимосовместимый набор, который позволил бы
```

Рисунок 21 – результат обработки

Как можно видеть, текст сразу разделяется на абзацы и не происходит «склеивание» текста.

Возьмем ранее рассмотренный файл с изображением, что показано на рисунках 22 и 23.

«ПЕДАГОГИКО-ПСИХОЛОГИЧЕСКИЕ И МЕДИКО-БИОЛОГИЧЕСКИЕ ПРОБЛЕМЫ ФИЗИЧЕСКОЙ КУЛЬТУРЫ И СПОРТА»
Электронный журнал Камской государственной академии физической культуры, спорта и туризма
Рег. № Эл №ФС77-27659 от 26 марта 2007 г.
№8 (3/2008)

УДК 796:159.9

АГРЕССИВНОСТЬ В СПОРТЕ

Аспирант А.В. Сычугов
Камская государственная академия физической культуры, спорта и туризма
Набережные Челны

AGGRESSIVENESS IN SPORT

A.V. Sychugov, post-graduate student
Kama State Academy of Physical Culture, Sports and Tourism
Naberezhnye Chelny

e-mail: sychugov.aleksej@rambler.ru

Ключевые слова: *агрессивность, психологическая подготовка спортсмена, конфликты в спорте, педагогические исследования агрессии.*

Аннотация. *У современной молодежи изменились культурные и духовные ценности, изменились примеры для подражания. Притягательными для подростков зачастую становятся те, кого они видят на экранах телевизоров, а именно бандиты, насильники и убийцы. Все это способствует повышению уровня агрессивности. Учёные придерживаются разных точек зрения по поводу того, является ли агрессия врожденным или приобретается в течение всей жизни. У человека в результате соответствующего воспитания в раннем детстве могут формироваться*

Рисунок 22 – PDF файл с изображением

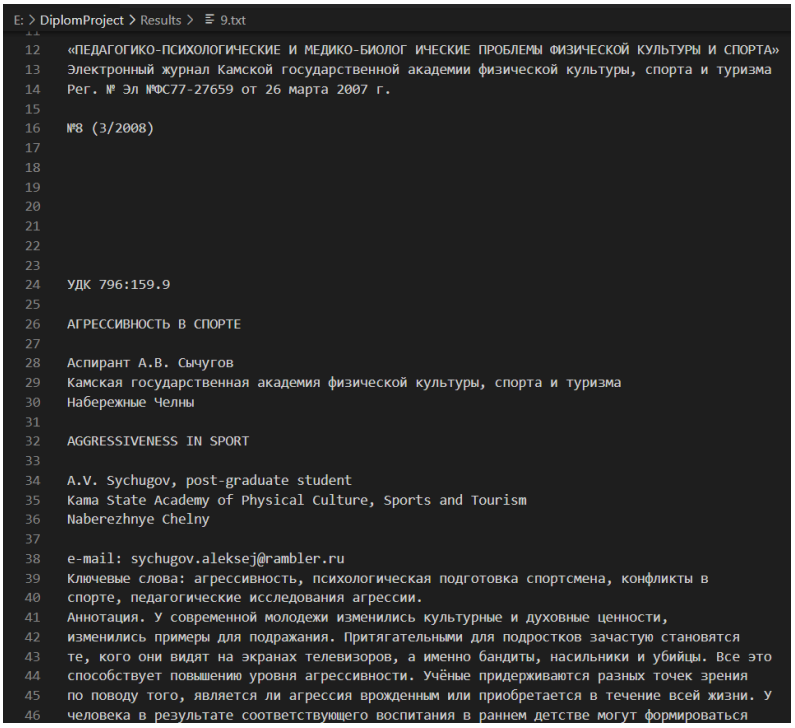


Рисунок 23 – результат обработки

В библиотеке iText7 (ITextSharp) изображения извлекаются так же в виде набора байт, из-за чего приходится снова преобразовывать байты в изображение перед отправкой в Tesseract.

Текст же извлекается лучше, чем с помощью библиотеки PdfPig. Здесь он сразу разделяется на абзацы, и полностью повторяет вид как в самом PDF документе.

Вывод в работе библиотек представлен в виде таблицы 2.

Таблица 2 – сравнение библиотек

Библиотека	Плюсы	Минусы
PdfPig	1. Более быстрая обработка 2. Работа со сложной разметкой	1. Может перемешивать порядок текста и текста с изображения 2. Не делит текст на абзацы 3. Сложное извлечение изображений

Библиотека	Плюсы	Минусы
iText7 (iTextSharp)	1. Работа со сложной разметкой 2. Деление текста на абзацы 3. Оптимизирован для работы с большими файлами	1. Более долгая обработка 2. Может перемешивать порядок текста и текста с изображения, но делает это реже за счет улучшенной работы с разметкой 3. Более сложный код для реализации функционала

Исходя из данных, которые были описаны выше, было принято решение использовать в проекте библиотеку iText7 (iTextSharp). Теперь программа включает в себя ранее описанную библиотеку для парсинга текста из PDF документов, а также библиотеку Tesseract.

Для более подробного анализа того, насколько точно программа справляется с извлечением текста из PDF файлов, были вручную составлены 10 файлов Word, которые позже были экспортированы в формат PDF. Данные файлы содержат текст, обычные изображения и изображения с текстом. Составление их вручную позволяет видеть, сколько слов было в оригинальном файле Word. Необходимо учесть, что Word не учитывает слова, которые находятся на изображениях, поэтому будет вестись подсчет данных слов, а после полученное значение будет добавляться к общему количеству слов.

Данные о проведенном тестировании представлены в таблице 3.

Таблица 3 – тестирование программы

№	Кол-во слов в оригинале		Кол-во распознанных слов	
	Слов в файле	Слов в изобр.	Слов в файле	Слов в изобр.
1	412	37	413	27
2	661	4	663	8
3	476	-	480	-
4	2886	6	2894	6
5	1923	57	1930	56

№	Кол-во слов в оригинале		Кол-во распознанных слов	
	Слов в файле	Слов в изобр.	Слов в файле	Слов в изобр.
6	672	-	689	-
7	914	-	937	-
8	3844	29	3882	22
9	1358	10	1437	8
10	1985	-	1997	-

В процессе проверки было замечено, что слов в извлеченном тексте оказывается больше, чем слов в оригинальном документе, это получается из-за следующих исключений:

– некоторые знаки препинания могут считываться отдельно, выделяясь запятыми, из-за чего Word показывает большее количество слов;

– в одном из файлов были древнегреческие символы, которые были считаны набором символов, превышающим изначальное количество слов;

– если есть значения в степени, например «10^5», то такие значения считываются как «10 5», в этом случае вместо одного слова Word покажет два слова.

Что касается извлеченного текста с изображений, то не всегда весь текст извлекается удачно. Tesseract хорошо извлекает текст, если изображение имеет достаточно высокое разрешение, в противном случае слова могут быть либо не извлечены, либо могут иметь некорректный вид.

В итоге, получаем готовые модули для парсинга текста из PDF файлов, а также распознавания текста с изображений.

3.2 Разработка модуля суммаризации текста

Для разработки модуля суммаризации текста используем библиотеку Sumy. Так как она разработана для языка Python, в C# проекте ее функционал будет использоваться посредством вызова Python скрипта.

Для начала скачаем и установим на компьютер сам Python версии 3.10.11, что показано на рисунке 24.

```
C:\Users\Артем>python --version
Python 3.10.11
```

Рисунок 24 – проверка версии Python

После установки Python, переходим к установке Sumy, что показано на рисунке 25

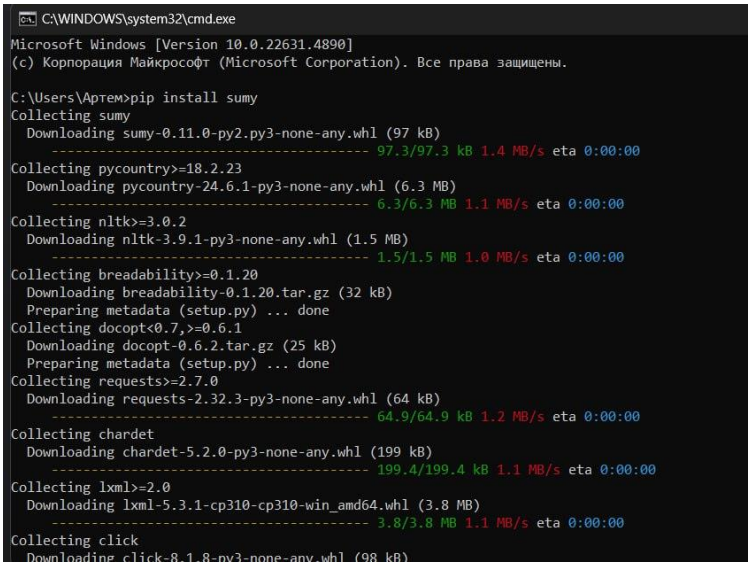


Рисунок 25 – установка Sumy

Так же для использования данной библиотеки потребуются дополнительные инструменты, такие как NumPy и Nltk.

Sumy использует NumPy для выполнения математических операций, которые необходимы для алгоритма LSA, используемом Sumy. LSA – представляет собой алгоритм обработки естественного языка, который используется для выявления скрытых семантических связей между словами и документами. Он работает по принципу построения матрицы «термины - документы», где каждая строка соответствует слову, а столбец – документу. То есть каждое предложение рассматривается как «документ». Элементами матрицы же являются веса слов. Так же применяется SVD – здесь выделяются скрытые темы, определяющие важность предложений. Установка показана на рисунке 26.

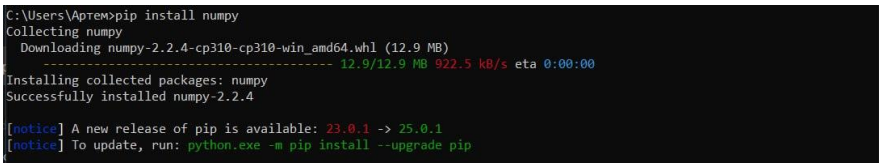


Рисунок 26 – установка NumPy

Sumy опирается на Nltk для разбиения текста на предложения, а так же разбиения предложений на слова, поэтому данную библиотеку тоже необходимо установить, что показано на рисунке 27.

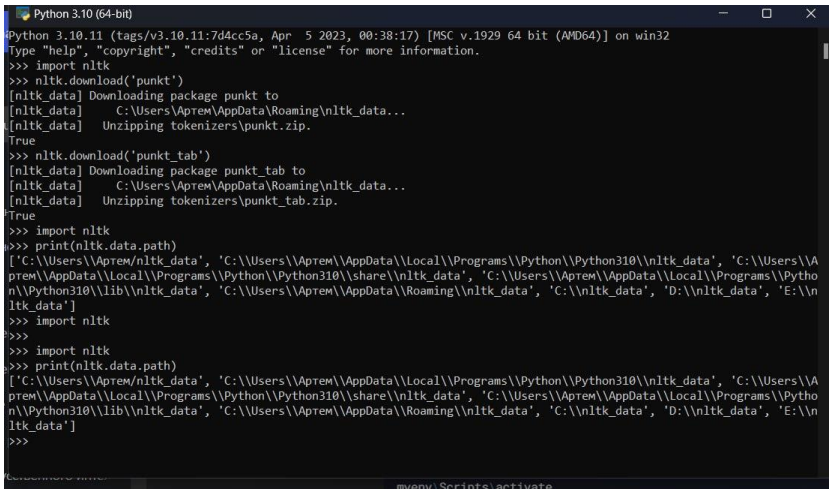


Рисунок 27 – установка Nltk

После в проекте C# была написана функция, которая сохраняет полученный текст во временный файл, а после вызывает Python скрипт передавая ему путь этого файла. Далее происходит ожидание получения пути временного файла с результатом суммаризации.

Далее был написан сам Python скрипт для суммаризации извлеченного текста.

Листинг 5 – настройка *Sumy*

```
    def summarize(text, sentences_count=5):
        parser = PlaintextParser.from_string(text, Tokenizer("russian"))
        summarizer = LsaSummarizer()
        summary = summarizer(parser.document, sentences_count)
        return " ".join(str(sentence) for sentence in summary)
```

В данной части кода мы производим настройку. Так как Sumy не способен обработать слишком большой текст, было принято решение ограничить суммаризацию по первым пяти предложениям. Так же здесь задаем токенизатор для русского языка и выбираем алгоритм обработки LSA.

Листинг 6 – обработка текста с помощью Sumy

```
if __name__ == "__main__":
    input_file_path = sys.argv[1]
    with open(input_file_path, 'r', encoding='utf-8') as input_file:
        text = input_file.read()
    summary = summarize(text)
    with tempfile.NamedTemporaryFile(mode='w', delete=False, encoding='utf-8') as temp_file:
        temp_file.write(summary)
        temp_file_path = temp_file.name
    print(temp_file_path)
```

В данной части кода происходит чтение пути к входному файлу из аргументов командной строки, после чего текст суммаризируется и результат сохраняется во временный файл, путь которого передается обратно C# коду.

В итоге получаем следующий результат, который показан на рисунке 28.

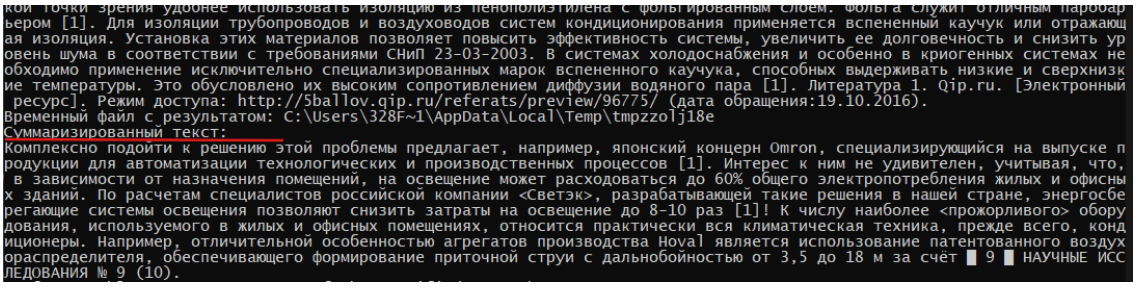


Рисунок 28 – результат суммаризации текста

3.4 Разработка модуля индексирования

Для разработки модуля индексирования и поиска будем использовать выбранную ранее систему Manticore Search. Для начала ее необходимо установить на сам компьютер, для этого необходимо скачать установщик с официального сайта, что показано на рисунке 29.

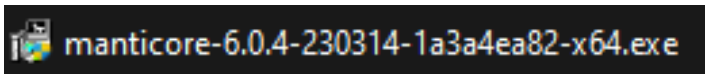


Рисунок 29 – скаченный установщик

Была скачена версия 6.0.4 для 64х разрядных систем. Теперь можно перейти к самой установке, что показано на рисунках 30, 31,32 и 33.

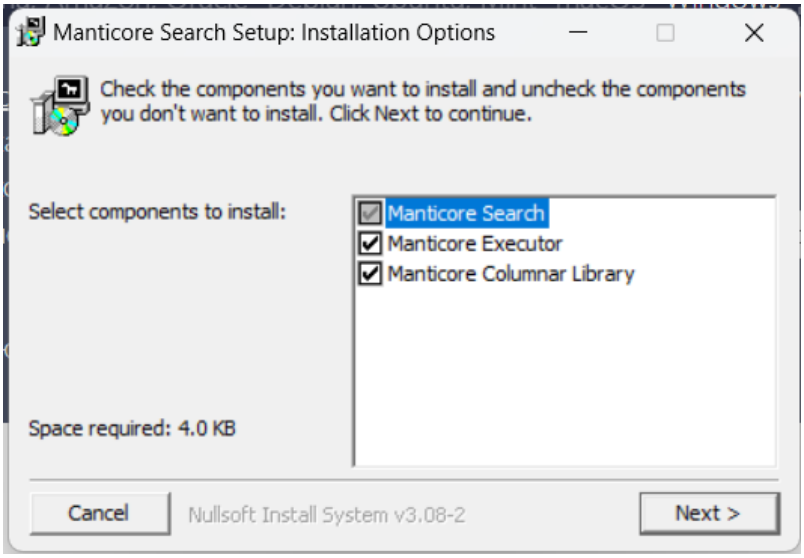


Рисунок 30 – выбор установки компонентов

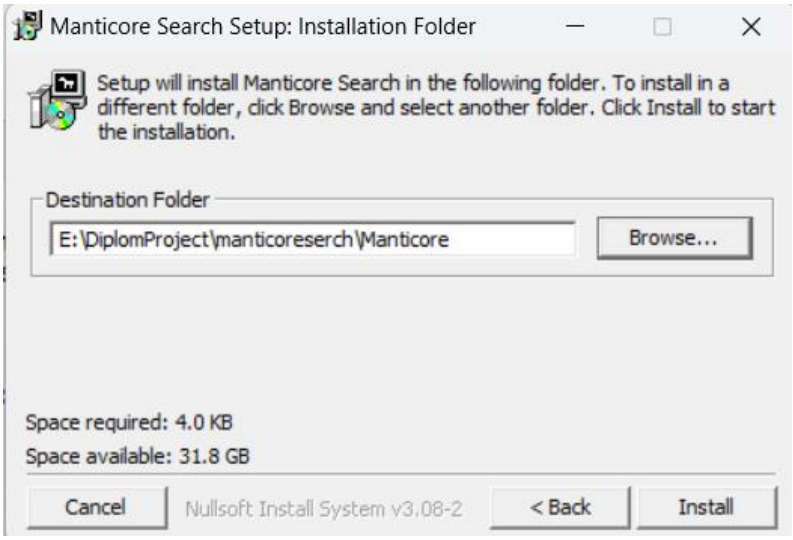


Рисунок 31 – выбор места установки

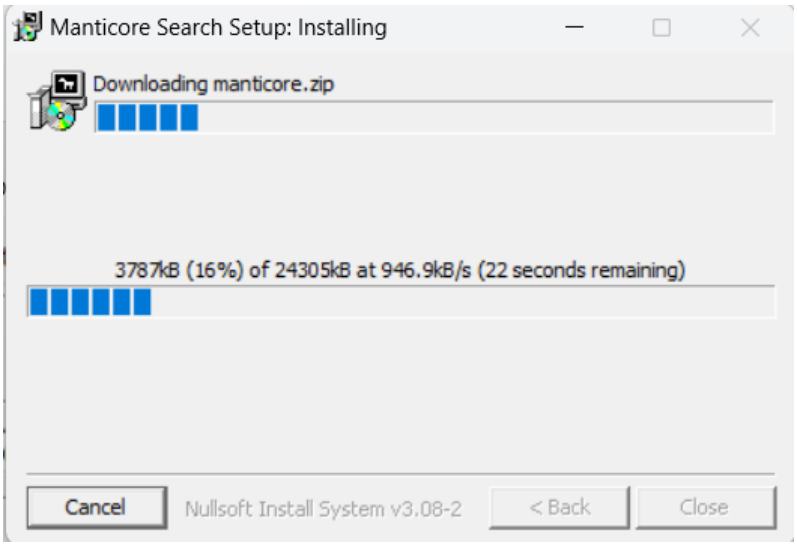


Рисунок 32 – процесс установки

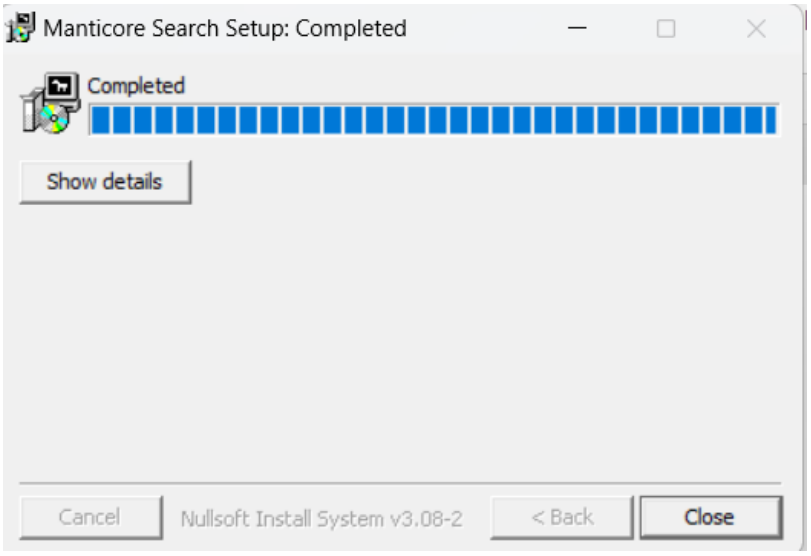


Рисунок 33 – завершение установки

После того, как программа была полностью установлена, необходимо проверить, имеется ли файл с русскими stop words. Для этого перейдем к файлам установленной программы и найдем файл ru. Путь: Manticore\share\stopwords\ru. При открытии данного файла он должен иметь следующий вид, что показано на рисунке 34.

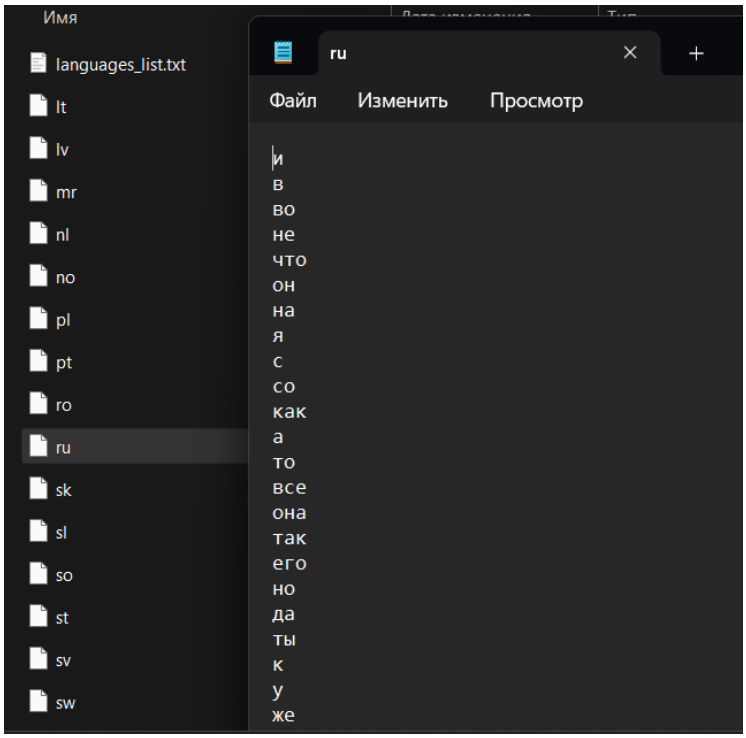


Рисунок 34 – файл с русскими stop words

Если данный файл отсутствует, то его необходимо загрузить отдельно с официального сайта Manticore и после добавить в папку stop words.

Теперь проверим, запускается ли Manticore, для этого откроем командную строку, перейдем в директорию, где было установлено приложение, зайдем в папку bin и выполним команду, которая показана на рисунке 35.

```
e:\DiplomProject\manticoreserch\Manticore\bin>searchd --config manticore.conf
```

Рисунок 35 – запуск Manticore

Здесь файл manticore.conf является файлом конфигурации запуска Manticore Search. После нажатия Enter, мы должны увидеть то, что показано на рисунке 36.

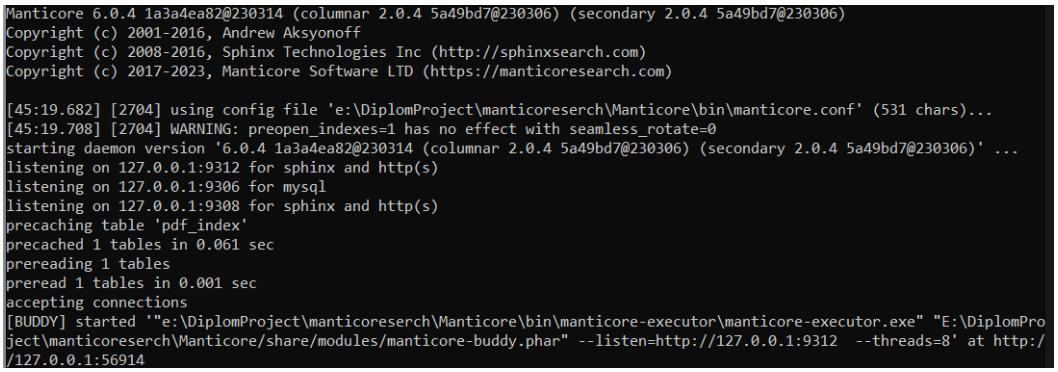


Рисунок 36 – запущенный Manticore

Это означает что Manticore Search успешно запустилась и работает. Для того что бы завершить работу, можно нажать сочетание клавиш Ctrl + C, или прописать команду searchd –stop, что показано на рисунке 37.

Рисунок 37 – завершение работы *Manticore*

Для взаимодействия с Manticore используется SQL подобный язык, для чего необходимо дополнительно установить на компьютер MySQL. Для работы потребуется только MySQL Client, который показан на рисунке 38.

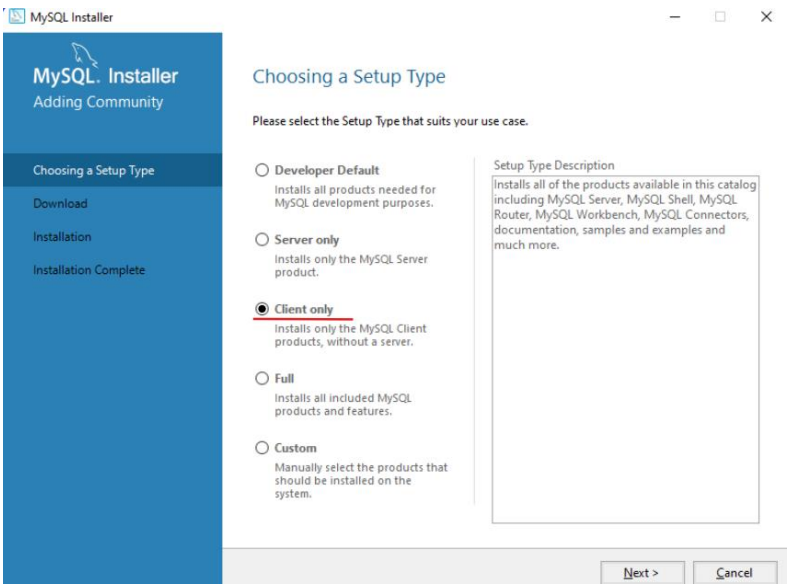


Рисунок 38 – установка MySQL Client

После установки и настройки MySQL Client переходим в проект Visual Studio, где с помощью управления пакетами NuGet устанавливаем необходимую библиотеку для работы с MySQL, что показано на рисунке 39.

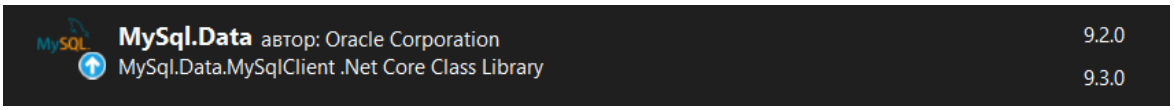


Рисунок 39 – установка библиотеки

Далее создается таблица, в которую будут помещаться обработанные данные.

Листинг 6 – Таблица данных

```
CREATE TABLE pdf_index (
    id          UINT,
    file_path   TEXT,
    content     TEXT,
    summary     TEXT,
    date_added TIMESTAMP
);
```

– поле id в данной таблице отвечает за уникальный идентификатор, который будет присваиваться каждой загруженной записи таблицы;

– поле file_path содержит в себе путь к исходному PDF документу;

– поле content содержит извлеченный текст из PDF файла;

– поле summary содержит суммаризированный текст;

– поле date_added содержит дату добавления в индекс.

После, в проекте Visual Studio была написана функция, благодаря которой происходит запись обработанных PDF документов. Параметры подключения к Manticore задаются в строке, что указано в листинге 7.

Листинг 7 – Параметры подключения к Manticore

```
String connectionString = Server=127.0.0.1;Port=9306;Database=default;User ID=root;Charset=utf8;";
```

А сами данные передаются с помощью кода, указанном в листинге 8.

Листинг 8 – передача данных в Manticore

```
INSERT INTO pdf_index (id, file_path, content, summary, date_added)
```

```
VALUES (@id, @file_path, @content, @summary, @date_added);";
using (MySqlCommand cmd = new MySqlCommand(query, conn))
{
    cmd.Parameters.AddWithValue("@id", GetUniqueId());
    cmd.Parameters.AddWithValue("@file_path", filePath);
    cmd.Parameters.AddWithValue("@content", content);
    cmd.Parameters.AddWithValue("@summary", summary);
    cmd.Parameters.AddWithValue("@date_added", currentDate);
    int rowsAffected = cmd.ExecuteNonQuery();
    Console.WriteLine($"Добавлено {rowsAffected} записей.");}
```

Далее необходимо проверить работоспособность описанного функционала. Для этого запускаем Manticore Search и подключаемся к нему через MySQL Client для того, чтобы можно было увидеть записанный результат. Подключение происходит командой, которая указана в листинге 9.

Листинг 9 – Подключение к Manticore

```
mysql --default-character-set=utf8mb4 -h 127.0.0.1 -P 9306 -u root
```

Запускаем программу и сравниваем результат, который показан на рисунке 40.

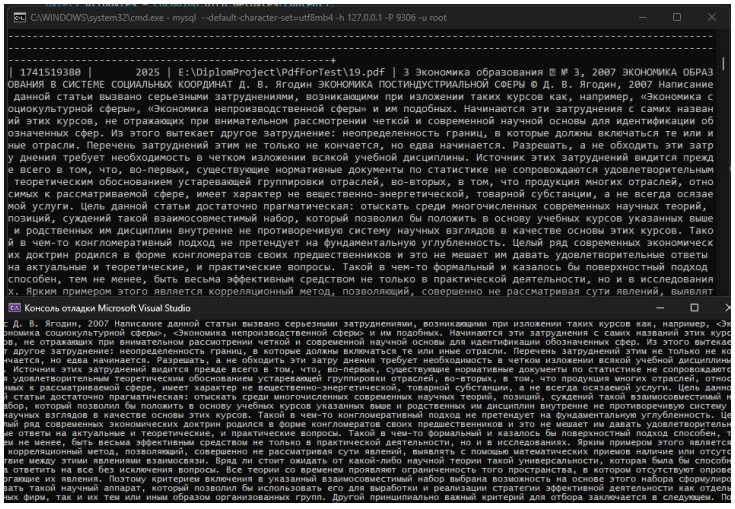


Рисунок 40 – запись в Manticore

Как можно видеть (рисунок 40), результат был полностью записан в базу данных Manticore Search. Сначала идет уникальный идентификатор, затем дата добавления, путь к файлу, извлеченный текст и суммаризированный текст.

3.5 Разработка интерфейса

Теперь, когда основные модули программы разработаны, необходимо приступить к разработке интерфейса приложения. Интерфейс должен придерживаться концепции user friendly interface (UFI) – это интерфейс, который удобен и понятен пользователю, даже если у него нет специальных знаний.

Основными характеристиками данной концепции будут являться:

– простота – минимум сложных элементов;

– удобство – логичное расположение кнопок, меню и функций;

– эстетика – приятный дизайн без лишней визуальной нагрузки;

– обратная связь – система подсказок, которая будет сообщать об ошибках и успешных действиях.

Для разработки интерфейса будем использовать Windows Forms – технология пользовательского интерфейса для .NET, которая представляет собой набор управляемых библиотек, что упрощают выполнение стандартных задач. Поддерживает широкий набор функций для разработки приложения, включая графику, привязку данных и другие. Функциональность, такая как размещение визуальных элементов управления методом перетаскивания упрощает создание приложений.

Сначала создадим проект в Visual Studio приложение Windows Forms, после чего мы увидим саму форму. Слева будет располагаться панель, где находятся все необходимые элементы для дальнейшей разработки, что показано на рисунке 42.

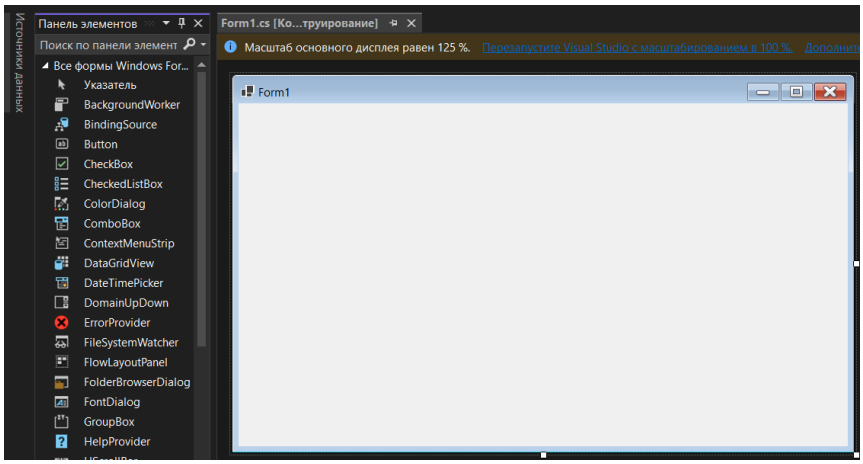


Рисунок 42 – начальное окно Forms

Далее с помощью элемента TubControl создадим две вкладки, в одной расположим функционал загрузки и обработки PDF документов, а в другой расположим функционал полнотекстового поиска.

На первую вкладку добавим все необходимые элементы, что показано на рисунке 43.

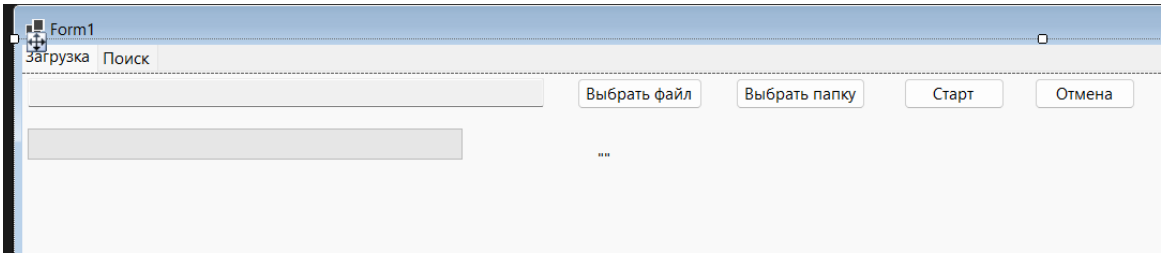


Рисунок 43 – вкладка загрузки

Здесь располагается поле, в котором отображается путь к выбранному файлу или выбранной папке с файлами, чтобы обработать не один документ, а сразу все документы в выбранной директории. Так же здесь располагаются элементы управления в виде кнопок, которые позволяют выбрать файл или папку, а также начать процесс обработки или остановить его. Для удобства наблюдения за процессом был добавлен Progress Bar и Label, которые отображают какой файл обрабатывается.

Перейдем на вторую вкладку, что показано на рисунке 44.

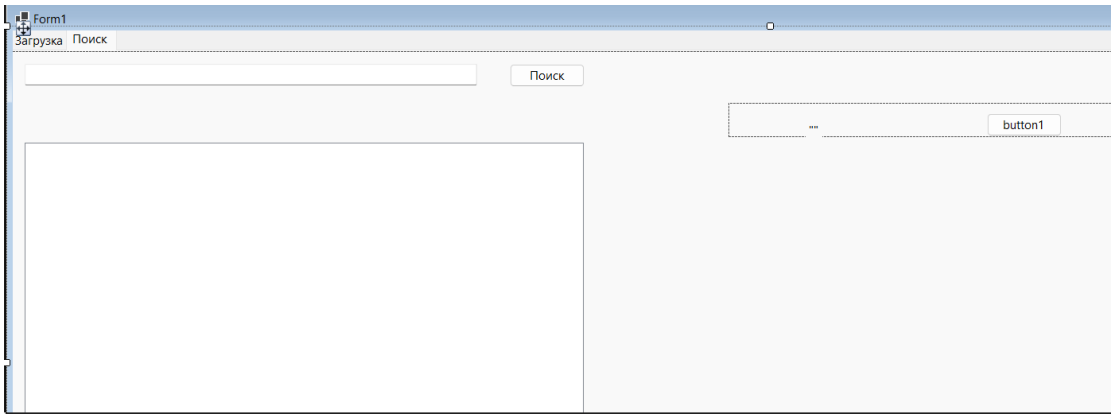


Рисунок 44 – вкладка поиска

Здесь располагается текстовое поле, куда будет вводиться текст для дальнейшего полнотекстового поиска, а также кнопка, запускающая сам поиск.

Ниже расположен ListBox, в котором будут отображаться результаты поиска, в которых содержится имя файла, дата, его расположения и краткое содержание. Левее будет располагаться внутренний PDF проигрыватель, а также его элементы управления, чтобы можно было просмотреть найденный файл в самой программе. Для этого необходимо дополнительно доустановить библиотеку PdfiumViewer, которая и позволит отображать PDF документы прямо в приложении, что показано на рисунке 45.

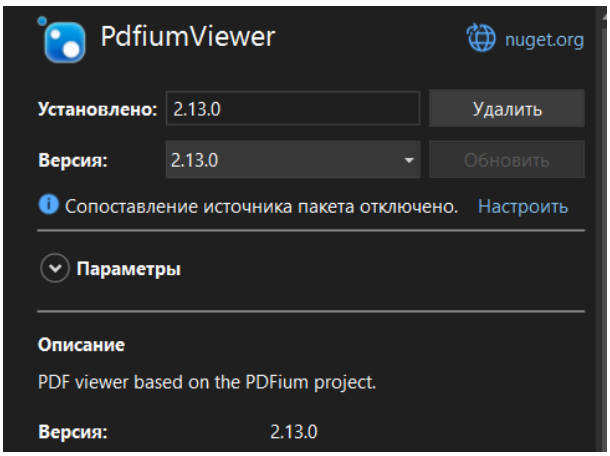


Рисунок 45 – библиотека PdfiumViewer

После установки, в панели элементов появится новый элемент, который так же перенесем на форму, что показано на рисунке 46.

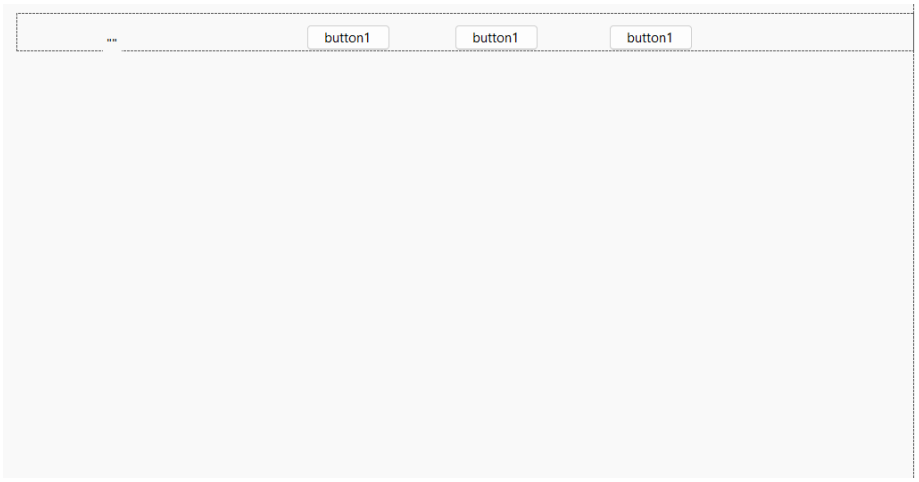


Рисунок 46 – инструменты отображения PDF

Здесь имеется Label, который будет меняться в зависимости от того, какая страница открыта, и три кнопки, две из них отвечают за переход между страницами, а третья позволяет закрыть проигрыватель PDF документа.

После расположения всех элементов, связываем их код с функционалом проекта и получаем конечное приложение для системы полнотекстового поиска по PDF документам с извлечением текста и распознаванием текста с изображений.

4 Тестирование программы

Для проведения тестирования разработанной системы полнотекстового поиска по PDF документам был взят компьютер со следующими характеристиками:

– ОС Windows 11;

– разрядность системы x64;

– процессор Intel Core i5 9300h;

– 8 Гб оперативной памяти;

Запускаем программу, выбираем для обработки один файл и нажимаем кнопку старт, что показано на рисунке 47.

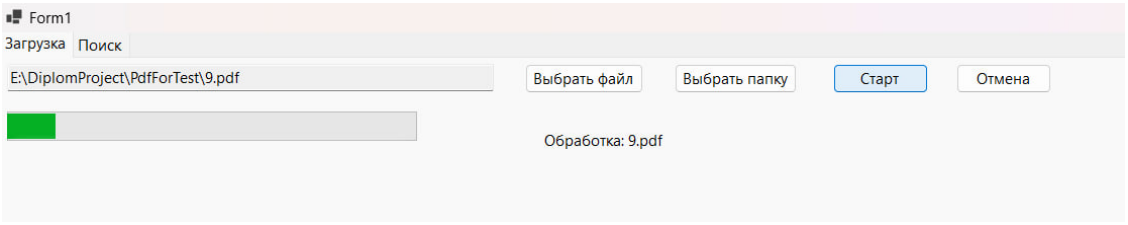


Рисунок 47 – процесс обработки одного файла

После обработки осуществляется автоматический переход на страницу поиска, где в списке файлов сразу отображается только что обработанный файл. Так же появляется Message Box, который оповещает пользователя о том, что файл успешно обработался, что показано на рисунке 48.

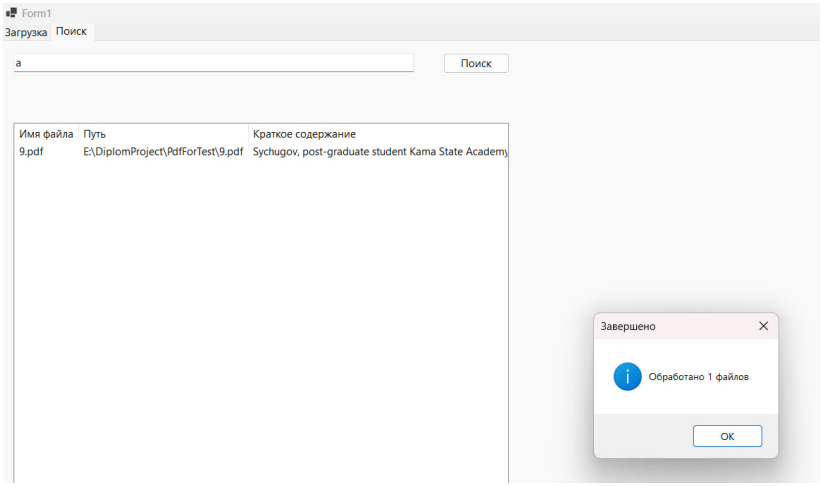


Рисунок 48 – обработанный файл

Далее снова переходим на вкладку загрузка и выбираем папку с PDF документами, после нажимаем старт, что показано на рисунке 49.

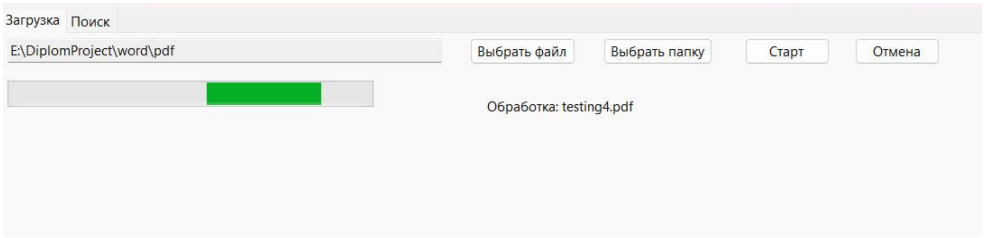


Рисунок 49 – процесс обработки файлов в папке

Далее все обработанные файлы так же отобразятся в списке, что показано на рисунке 50.

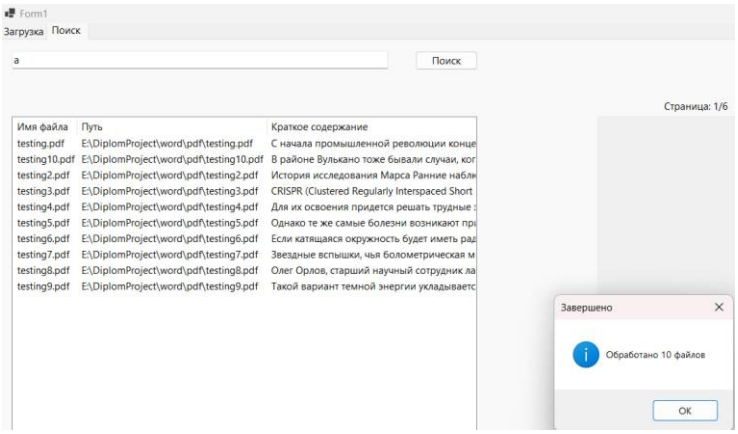


Рисунок 50 – обработанные файлы

Процесс обработки также можно отменить, для этого, пока выполняется обработка одного или нескольких файлов, необходимо нажать на кнопку отмены. В этом случае обработка остановится на том файле, на котором была прервана, что показано на рисунке 51.

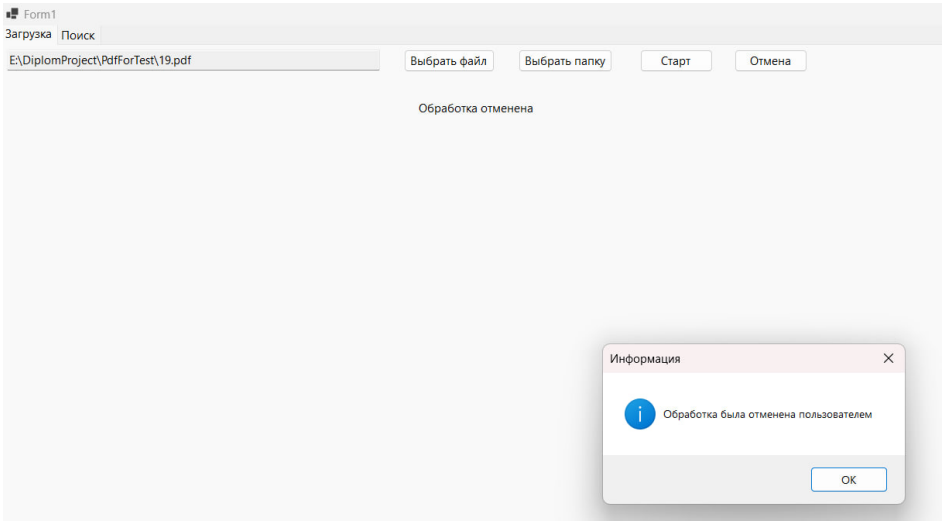


Рисунок 51 – отмена обработки

Далее были обработаны так же 104 файла. При их обработке потребление оперативной памяти приложением было в среднем около 220 Мб, а процессор загружался в среднем на 9%, что показано на рисунке 52.

Рисунок 52 – потребление оп и загрузка ЦП

Для того, чтобы выполнить полнотекстовый поиск по документам, необходимо перейти на вкладку поиска и ввести запрос в поле, после чего нажать на кнопку поиск. Можно вводить как одно слово, так и целое предложение, это можно видеть на рисунках 53 и 54.

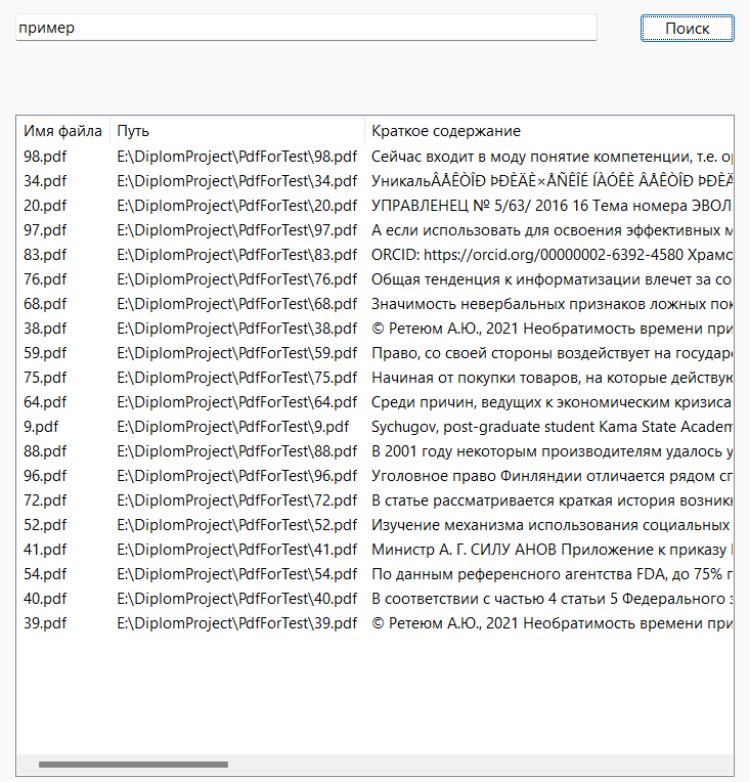


Рисунок 53 – поиск по слову

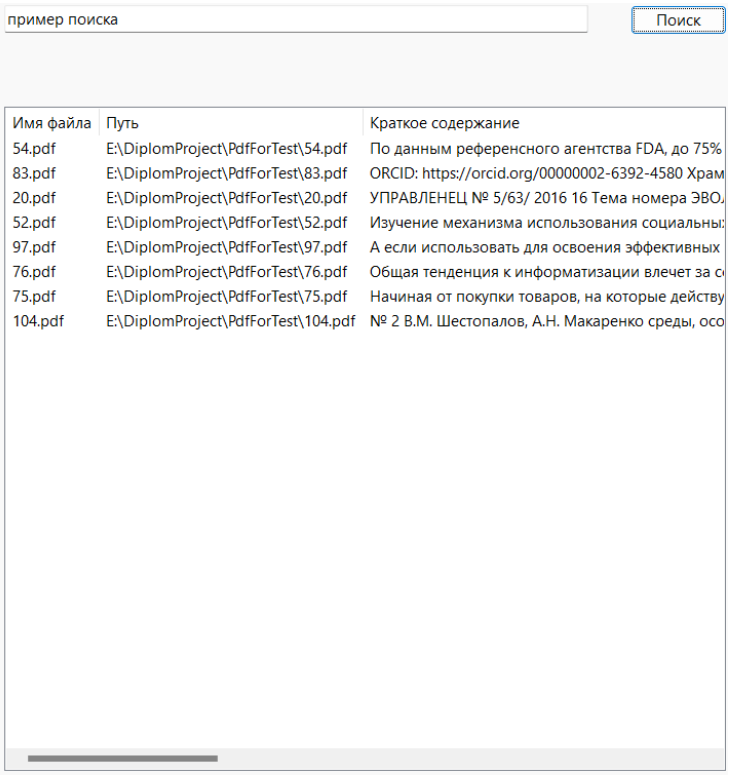


Рисунок 54 – поиск по предложению

Так же при однократном нажатии на выбранный из списка файл откроется внутренний PDF проигрыватель, в котором можно листать страницы, приближать или отдалять сам файл, а также закрыть проигрыватель, что показано на рисунке 55.

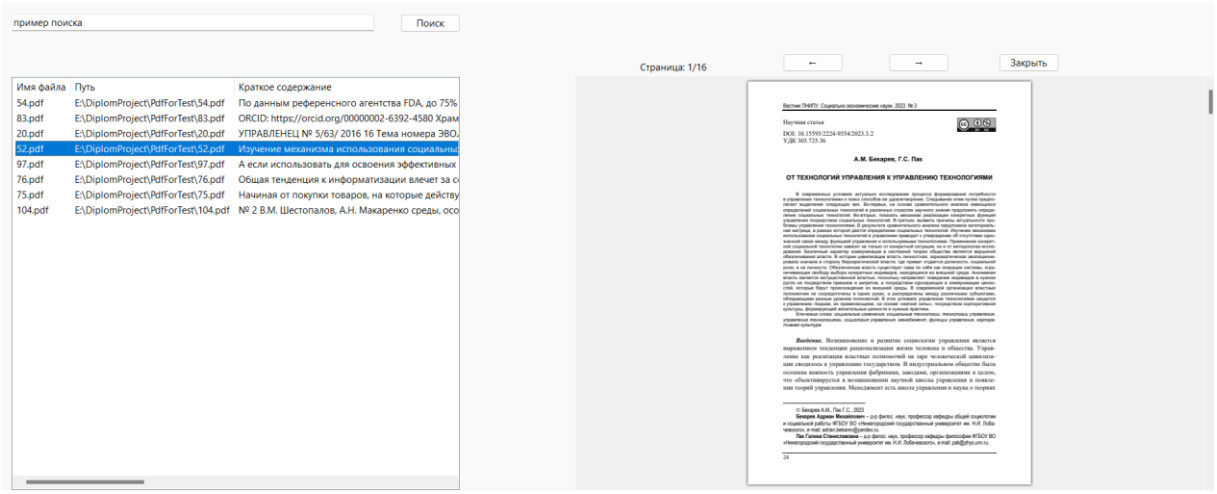


Рисунок 55 – открытие проигрывателя

При двухкратном нажатии на выбранный файл, он откроется средствами компьютера, в данном случае при помощи браузера Microsoft Edge, что показано на рисунке 56

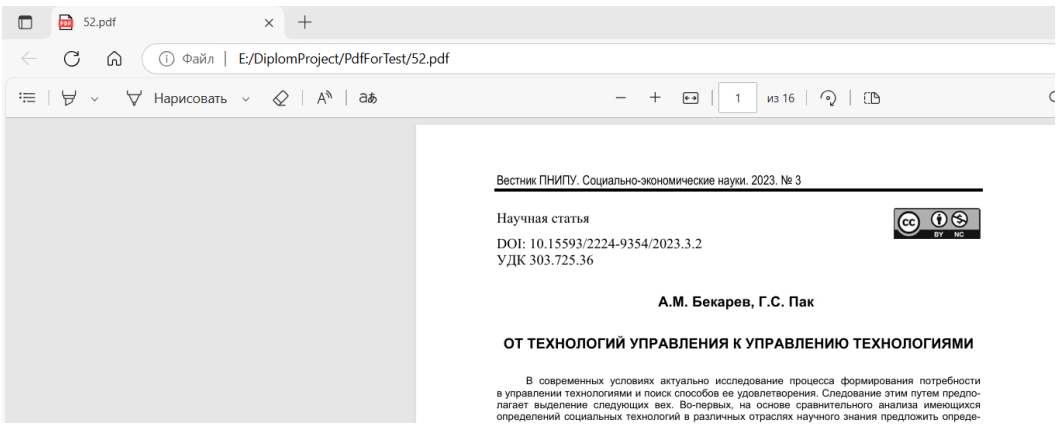


Рисунок 56 – открытие в браузере

5 Безопасность жизнедеятельности

5.1 Правила безопасности при использовании ПК

Правила выполнения работ с ПК обязаны обеспечивать минимизацию основных рисков, перечисленных в ГОСТ 12.0.003-2015. В первую очередь к ним относятся:

– повышенная температура отдельных элементов персонального компьютера;

– монотонность трудового процесса;

– высокий уровень зрительных нагрузок;

– вероятность поражения статическим электричеством, генерируемым техникой;

– недостаточная освещенность в зоне выполнения операций у компьютера;

– высокий уровень напряженности электрического и магнитного полей в зоне установки компьютера;

– другие факторы.

Полностью избавиться от их действия, конечно, не получится, однако работодатель обязан стремиться к тому, чтобы сократить воздействие этих факторов на сотрудников, обеспечивая возможность восстановления нормального самочувствия и работоспособности уже к началу следующей рабочей смены без применения дополнительных средств, например, массажа или медикаментозной терапии [13].

Заключение

В ходе выполнения дипломной работы мною был проведен детальный анализ современных подходов к обработке PDF-документов, выявлены их преимущества и недостатки. Выбраны оптимальные инструменты для разработки на основе тестирования различных библиотек для обработки PDF. Разработана архитектура системы, включающая:

– модуль парсинга PDF;

– модуль распознавания текста OCR;

– модуль суммаризации текста;

– модуль индексирования;

– модуль поиска.

Реализована система полнотекстового поиска по PDF-документам с поддержкой извлечения текста и распознаванием текстовой информации с изображений. Разработанное решение позволяет эффективно обрабатывать различные PDF-файлы и обеспечивает удобный механизм поиска по их содержимому. Полученные результаты открывают возможности для дальнейшего совершенствования системы и ее внедрения в реальные производственные процессы.

Список использованных источников

1. Google Document AI. [Электронный ресурс]. URL: https://cloud.google.com/document-ai (дата обращения 09.05.25)

2. Amazon Textract. [Электронный ресурс]. URL: https://aws.amazon.com/ru/textract/ (дата обращения 09.05.25)

3. Adobe Acrobat Pro DC. [Электронный ресурс]. URL: https://www.adobe.com/ru/acrobat.html (дата обращения 11.05.25)

4. ABBYY FineReader. [Электронный ресурс].URL:https://www.abbyy.com/finereader/ (дата обращения 11.05.25)

5. Apache Tika + Solr. [Электронный ресурс]. URL: https://tika.apache.org/ (дата обращения 12.05.25)

6. Paperless - ngx. [Электронный ресурс]. URL: https://github.com/paperless-ngx/paperless-ngx (дата обращения 12.05.25)

7. Руководство C#. [Электронный ресурс]. URL: https://learn.microsoft.com/ru-ru/dotnet/csharp/ (дата обращения 16.05.25)

8. Microsoft Visual Studio. [Электронный ресурс]. URL: https://visualstudio.microsoft.com/ru/ (дата обращения 16.05.25)

9. Windows Forms. [Электронный ресурс]. URL:https://learn.microsoft.com/ru-ru/dotnet/desktop/winforms/overview/?view=netdesktop-9.0 (дата обращения 17.05.25)

10. Tesseract documentation. [Электронный ресурс]. URL: https://tesseract-ocr.github.io/ (дата обращения 18.05.25)

11. Manual Sumy. [Электронный ресурс]. URL: https://pypi.org/project/sumy/ (дата обращения 20.05.25)

12. Manual Manticore search. [Электронный ресурс]. URL: https://manual.manticoresearch.com/Intr oduction (дата обращения 21.05.25)

13. Информация о технике безопасности. [Электронный ресурс]. URL: https://www.centrattek.ru/info/tekhnika-bezopasnosti-pri-rabote-s-kompyuterom-razyasneniya/ (дата обращения 28.05.25)

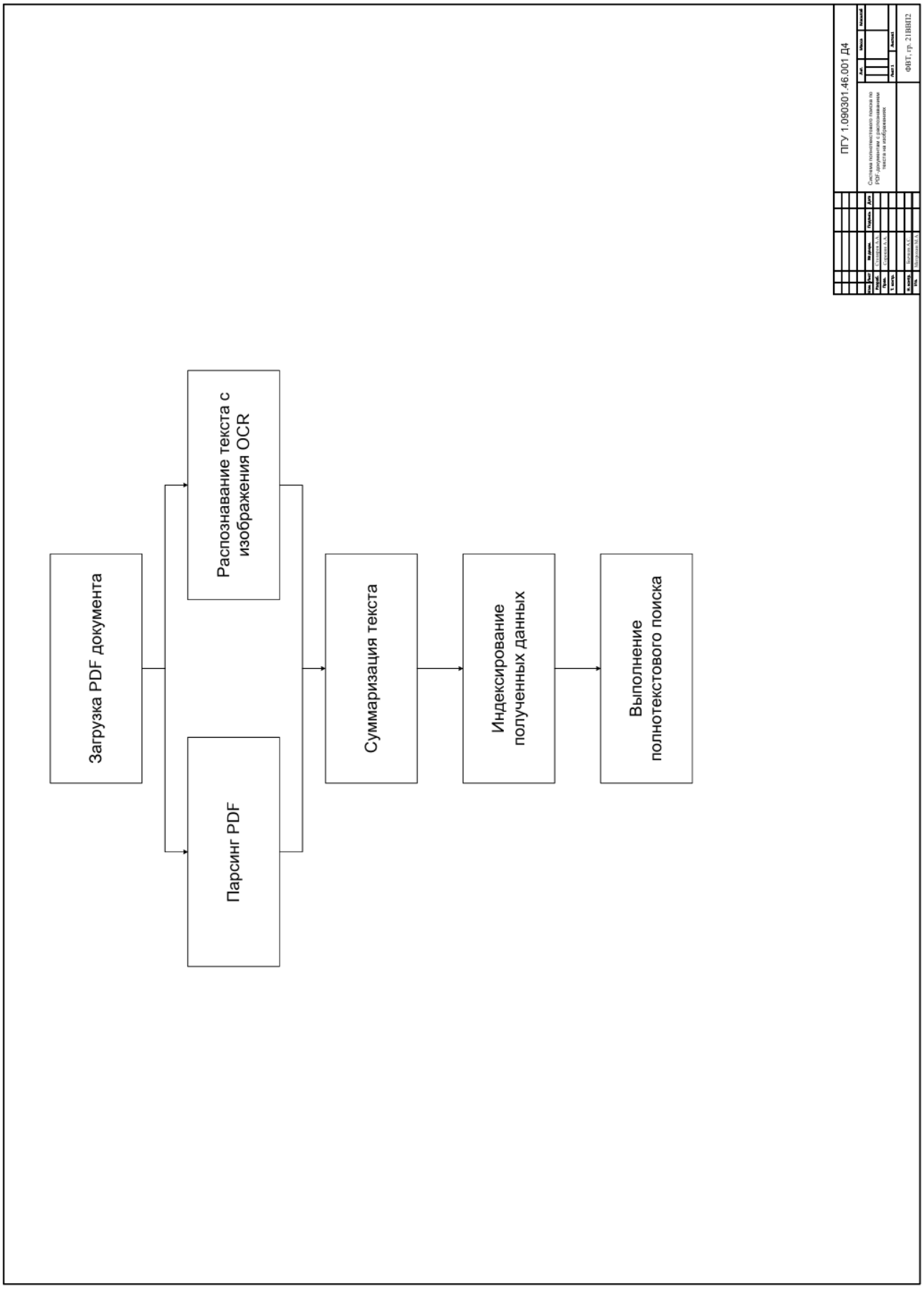
Приложение А

(Обязательное)

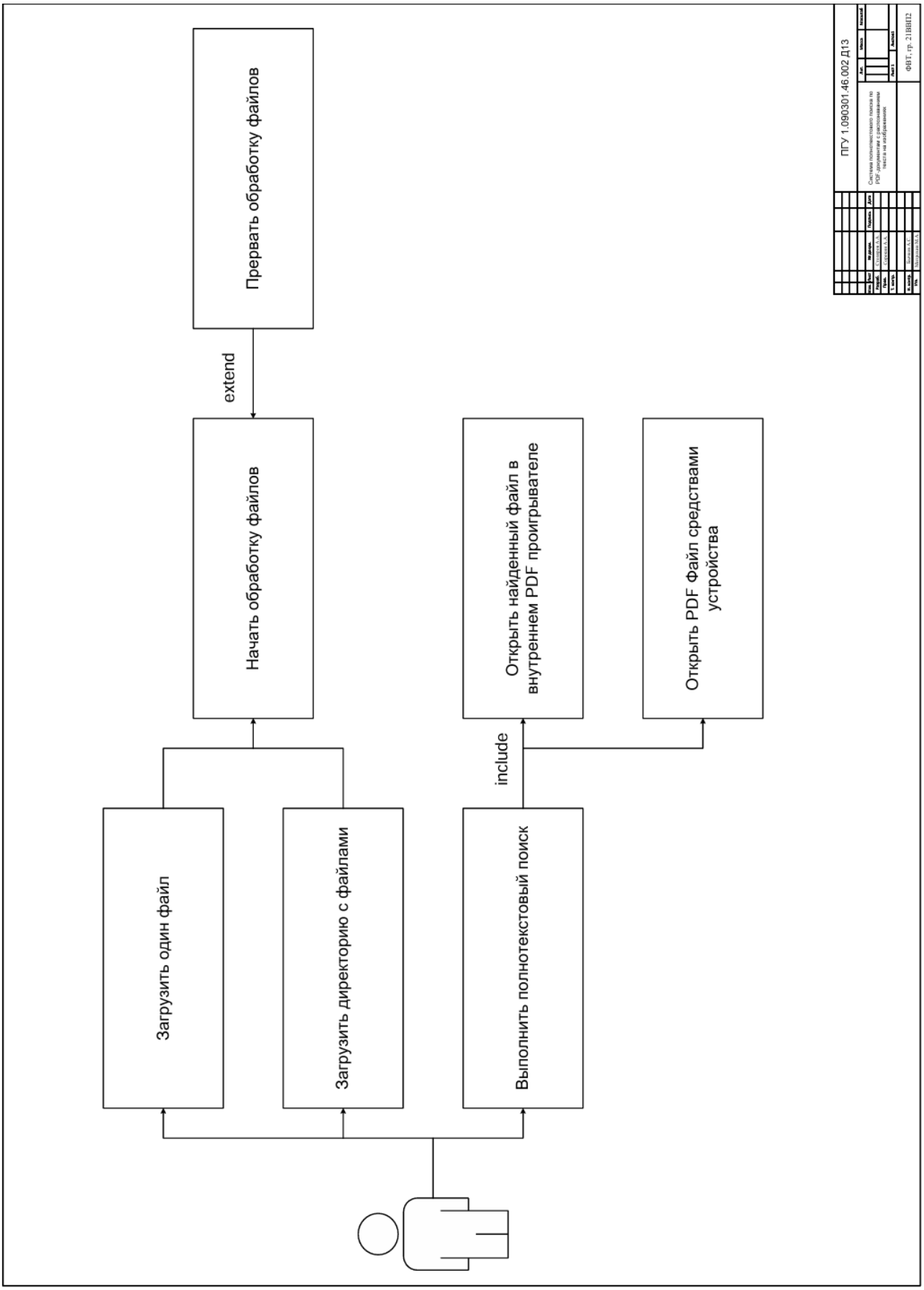
Система полнотекстового поиска по PDF-документам с распознаванием текста на изображениях

UML-диаграммы

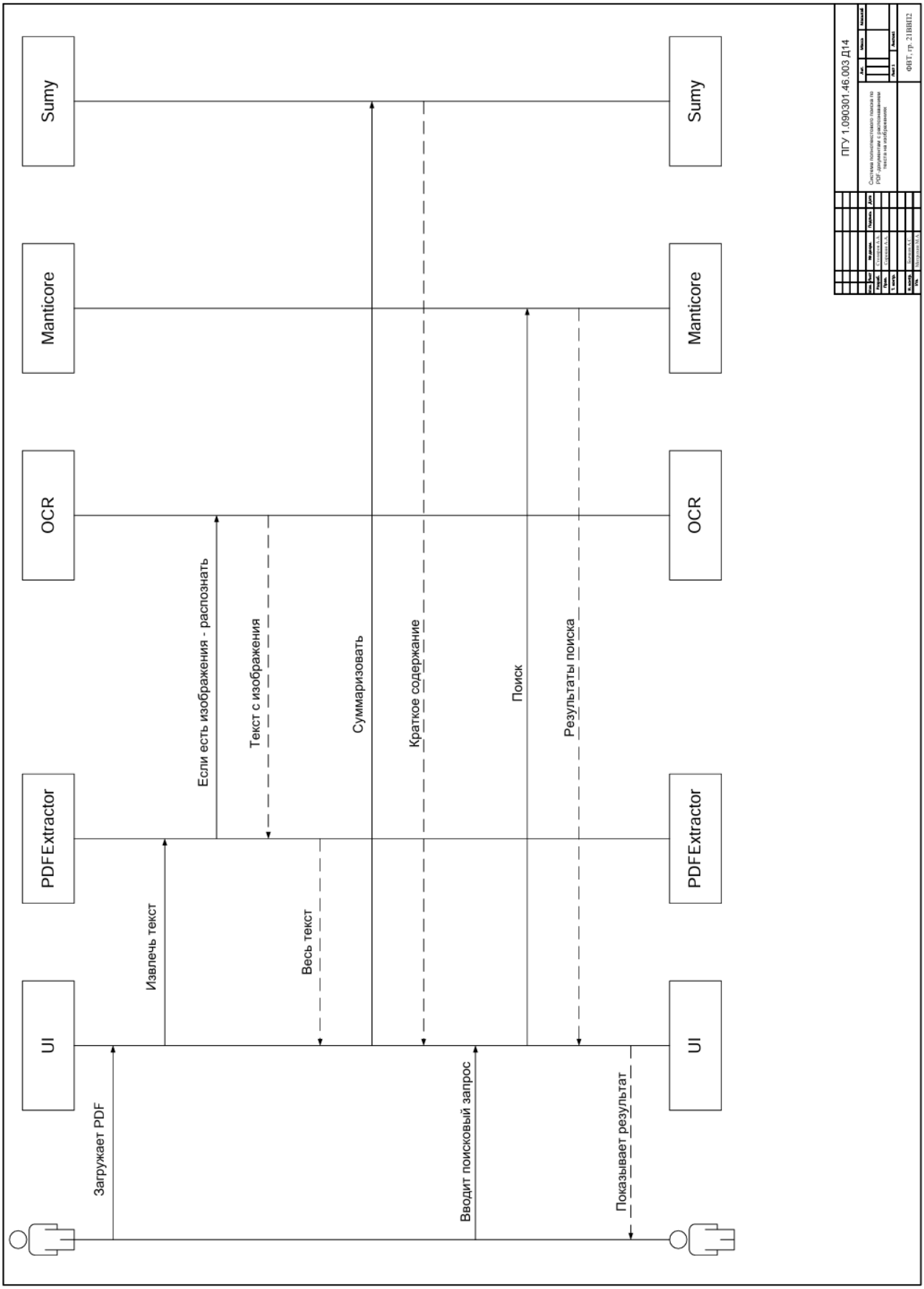
Приложение А.1 – UML-диаграмма взаимодействия модулей



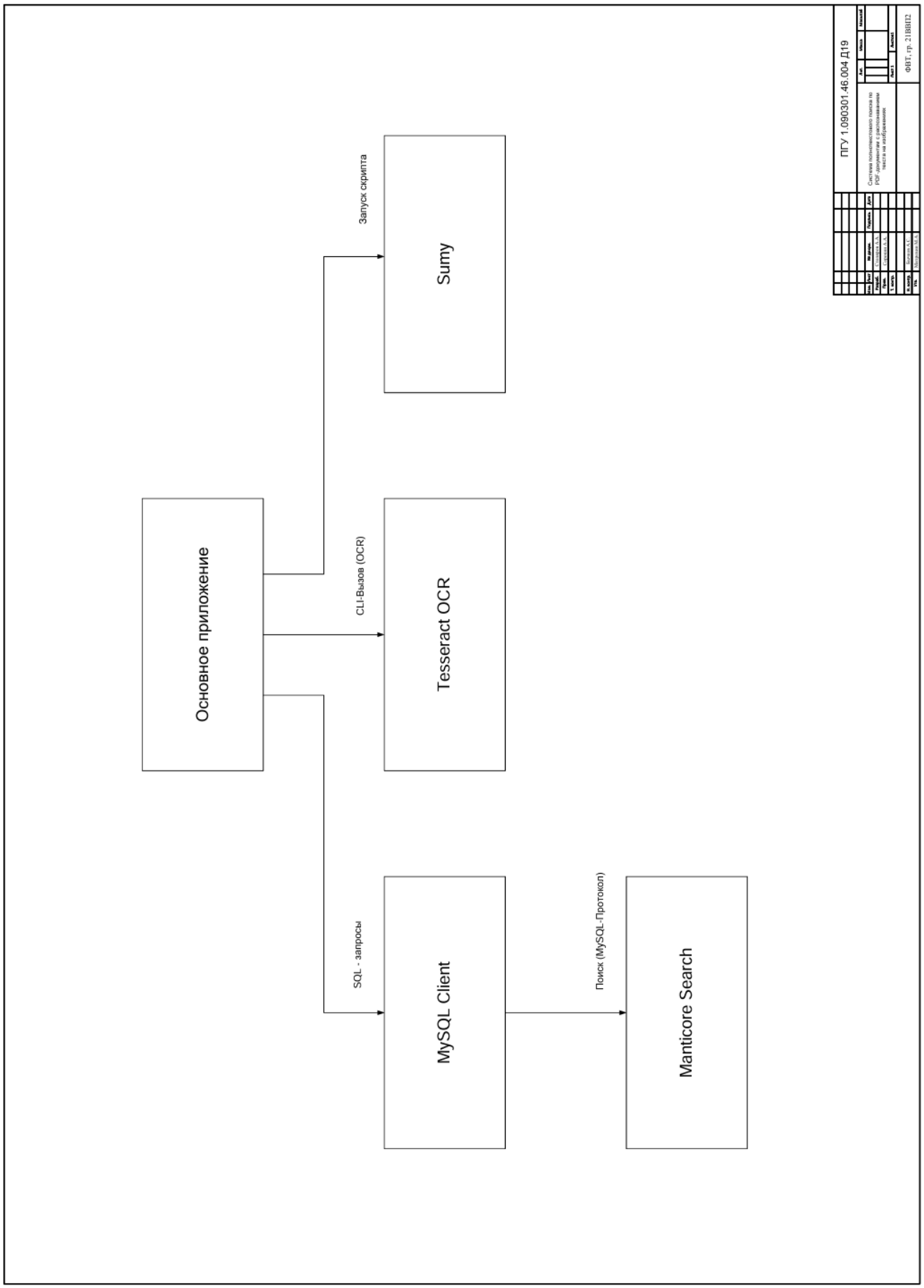
Приложение А.2 – UML-диаграмма вариантов использования



Приложение А.3 – UML-диаграмма последовательностей



Приложение А.4 – UML-диаграмма развертывания



Приложение Б

(Обязательное)

Система полнотекстового поиска по PDF-документам с распознаванием текста на изображениях

Листинг программы

Файл «Program.cs»

```
namespace WinFormsApp1
{
    internal static class Program
    {
        /// <summary>
        ///  The main entry point for the application.
        /// </summary>
        [STAThread]
        static void Main()
        {
            // To customize application configuration such as
set high DPI settings or default font,
            // see https://aka.ms/applicationconfiguration.
            ApplicationConfiguration.Initialize();
            Application.Run(new Form1());
        }
    }
}
```

Файл «Form1.cs»

```
using System.Text.RegularExpressions;
using iText.Kernel.Pdf;
using iText.Kernel.Pdf.Canvas.Parser;
using iText.Kernel.Pdf.Canvas.Parser.Listener;
using Tesseract;
using SixLabors.ImageSharp;
using SixLabors.ImageSharp.Processing;
using SixLabors.ImageSharp.Formats.Png;
using SixLabors.ImageSharp.PixelFormats;
using iText.Kernel.Pdf.Canvas.Parser.Data;
using System.Diagnostics;
using MySql.Data.MySqlClient;
using System.Text;
using SixLabors.ImageSharp.Formats.Gif;
using System.ComponentModel;
namespace WinFormsApp1
{
    public partial class Form1 : Form
    {
        private BackgroundWorker backgroundWorker;

        public static List<string>
ExtractTextAndImagesFromPDF(string filePath)
        {
            List<string> extractedText = new List<string>();
            List<byte[]> images = new List<byte[]>();
```

```
            var eventListener = new
ImageExtractionEventListener(images); // Передаём список
изображений

            using (PdfReader reader = new PdfReader(filePath))
            using (PdfDocument pdfDoc = new PdfDocument(reader))
            {
                for (int page = 1; page <=
pdfDoc.GetNumberOfPages(); page++)
                {
                    var pdfPage = pdfDoc.GetPage(page);
                    ITextExtractionStrategy strategy = new
SimpleTextExtractionStrategy();
                    string pageText =
PdfTextExtractor.GetTextFromPage(pdfPage, strategy);
                    List<string> pageTextBlocks = new
List<string>(pageText.Split('\n'));
                    List<string> imageTextBlocks = new
List<string>();

                    if
(pdfPage.GetResources().GetResource(PdfName.XObject) != null)
                    {
                        var pageImages =
GetImagesFromPage(pdfPage, eventListener); // Передаём слушатель
                        foreach (var image in pageImages)
                        {
                            string textFromImage =
RecognizeTextFromImage(image);
                            if
(!string.IsNullOrEmpty(textFromImage))
                            {

imageTextBlocks.Add(textFromImage);
                            }
                        }
                    }

                    int imageIndex = 0;
                    foreach (var block in pageTextBlocks)
                    {
                        string cleanedBlock =
CleanLineBreaks(block);
                        extractedText.Add(cleanedBlock);
                        if (imageIndex < imageTextBlocks.Count)
                        {

extractedText.Add(imageTextBlocks[imageIndex]);
                            imageIndex++;
                        }
                    }
```

```
                }
            }

            List<string> mergedText =
MergeTextBlocks(extractedText);
            List<string> finalText =
MergeHyphenatedWords(mergedText);

            // Удаляем временные файлы после завершения
обработки
            eventListener.CleanupTempFiles();

            return finalText;
        }

        // Метод для очистки переносов слов
        private static string CleanLineBreaks(string text)
        {
            // Удаляем дефисы в конце строк и объединяем строки
            if (text.EndsWith("- "))
            {
                text = text.TrimEnd('-').TrimEnd(); // Удаляем
дефис и лишние пробелы
            }
            return text;
        }

        // Метод для объединения текстовых блоков в один список
без лишних пробелов
        private static List<string> MergeTextBlocks(List<string>
textBlocks)
        {
            List<string> mergedText = new List<string>();
            string currentBlock = string.Empty;
            foreach (var block in textBlocks)
            {
                if (string.IsNullOrWhiteSpace(block))
                {
                    // Если текущий блок не пустой, добавляем
его в список
                    if
(!string.IsNullOrWhiteSpace(currentBlock))
                    {
                        mergedText.Add(currentBlock.Trim());
                        currentBlock = string.Empty;
                    }
                }
                else
                {
                    // Если текущий блок пустой, начинаем новый
блок
```

```
                    if (string.IsNullOrWhiteSpace(currentBlock))
                    {
                        currentBlock = block.Trim();
                    }
                    else
                    {
                        // Объединяем блоки с учётом пробелов
                        currentBlock += " " + block.Trim();
                    }
                }
            }
            // Добавляем последний блок, если он не пустой
            if (!string.IsNullOrWhiteSpace(currentBlock))
            {
                mergedText.Add(currentBlock.Trim());
            }
            return mergedText;
        }

        // Метод для объединения перенесённых слов
        private static List<string>
MergeHyphenatedWords(List<string> textBlocks)
        {
            // Объединяем все блоки в один текстовый блок
            string fullText = string.Join(" ", textBlocks);
            // Используем регулярное выражение для объединения
перенесённых слов
            string cleanedText = Regex.Replace(fullText,
@"\b(\w+)- (\w+)\b", "$1$2");
            // Удаляем лишние пробелы между словами
            cleanedText = Regex.Replace(cleanedText, @"\s+", "
").Trim();
            // Разделяем текст на предложения
            List<string> sentences =
SplitIntoSentences(cleanedText);
            return sentences;
        }

        // Метод для разделения текста на предложения
        private static List<string> SplitIntoSentences(string
text)
        {
            List<string> sentences = new List<string>();
            // Регулярное выражение для разделения текста на
предложения
            Regex sentenceRegex = new
Regex(@"(?<=[.!?])\s+(?=[A-ZA-Я])");
            string[] parts = sentenceRegex.Split(text);
            foreach (var part in parts)
            {
                if (string.IsNullOrWhiteSpace(part))
```

```
                {
                    continue;
                }
                sentences.Add(part.Trim());
            }
            return sentences;
        }

        // Метод для извлечения изображений из PDF и
распознавания текста на них с помощью Tesseract
        public static List<byte[]>
GetImagesFromPage(iText.Kernel.Pdf.PdfPage page,
ImageExtractionEventListener eventListener)
        {
            PdfCanvasProcessor processor = new
PdfCanvasProcessor(eventListener);
            PdfResources resources = page.GetResources();
            processor.ProcessContent(page.GetContentBytes(),
resources);
            return eventListener.GetImages(); // Возвращаем
список изображений
        }

        // Слушатель для извлечения изображений
        public class ImageExtractionEventListener :
IEventListener
        {
            private List<byte[]> _images; // Список для хранения
извлечённых изображений
            private List<string> _tempFilePaths; // Список для
хранения путей к временным файлам

            public ImageExtractionEventListener(List<byte[]>
images)
            {
                _images = images;
                _tempFilePaths = new List<string>();
            }

            public void EventOccurred(IEventData data, EventType
type)
            {
                if (type == EventType.RENDER_IMAGE)
                {
                    var renderInfo = data as ImageRenderInfo;
                    if (renderInfo != null)
                    {
                        var img = renderInfo.GetImage();
                        if (img != null)
                        {
                            try
```

```
                {
                    // Извлекаем изображение в виде
массива байтов
                    byte[] imageBytes =
img.GetImageBytes();

                    // Загружаем изображение с
помощью ImageSharp
                    using (var image =
SixLabors.ImageSharp.Image.Load<Rgba32>(imageBytes))
                    {
                        // Если изображение
использует индексированную палитру, преобразуем его в RGB
                        if
(image.Frames.RootFrame.Metadata.GetFormatMetadata(PngFormat.Ins
tance) != null ||

image.Frames.RootFrame.Metadata.GetFormatMetadata(GifFormat.Inst
ance) != null)
                        {
                            var rgbImage =
image.CloneAs<Rgba32>();
                        }

                        // Сохраняем изображение в
формате PNG
                        using (var outputStream =
new MemoryStream())
                        {
                            image.Save(outputStream,
new PngEncoder());
                            byte[] rgbImageBytes =
outputStream.ToArray();

                            // Добавляем
преобразованное изображение в список

_images.Add(rgbImageBytes);

                            // Сохраняем изображение
в файл для отладки
                            string tempFilePath =
$"extracted_image_{Guid.NewGuid()}.png";

File.WriteAllBytes(tempFilePath, rgbImageBytes);

_tempFilePaths.Add(tempFilePath);
                        }
                    }
                }
                catch (Exception ex)
```

```
                            {
                                Console.WriteLine($"Ошибка при
обработке изображения: {ex.Message}");
                            }
                        }
                    }
                }

                public ICollection<EventType> GetSupportedEvents()
                {
                    return new List<EventType> {
EventType.RENDER_IMAGE };
                }

                // Метод для удаления временных файлов
                public void CleanupTempFiles()
                {
                    foreach (var filePath in _tempFilePaths)
                    {
                        try
                        {
                            if (File.Exists(filePath))
                            {
                                File.Delete(filePath);
                                Console.WriteLine($"Удалён временный
файл: {filePath}");
                            }
                        }
                        catch (Exception ex)
                        {
                            Console.WriteLine($"Ошибка при удалении
файла {filePath}: {ex.Message}");
                        }
                    }
                }

                // Метод для получения списка изображений
                public List<byte[]> GetImages()
                {
                    return _images; // Возвращаем список изображений
                }
            }

            // Метод для улучшения изображения перед его
распознаванием
            public static Image<Rgba32> EnhanceImage(byte[]
imageBytes)
            {
                using (var stream = new MemoryStream(imageBytes))
                {
```

```
                var image =
SixLabors.ImageSharp.Image.Load<Rgba32>(stream);
                image.Metadata.ExifProfile = null;
                image.Metadata.IccProfile = null;
                image.Mutate(ctx => ctx
                    .Grayscale()
                    .Contrast(1.2f)
                    .BinaryThreshold(0.5f));
                return image;
            }
        }

        // Метод для распознавания текста с изображения с
использованием Tesseract
        public static string RecognizeTextFromImage(byte[]
imageBytes)
        {
            string result = string.Empty;
            if (imageBytes != null && imageBytes.Length > 0)
            {
                try
                {
                    // Загружаем изображение с помощью
ImageSharp
                    using (var image =
SixLabors.ImageSharp.Image.Load<Rgba32>(imageBytes))
                    {
                        // Преобразуем изображение в RGB (если
оно в другом формате)
                        var rgbImage = image.CloneAs<Rgba32>();

                        // Сохраняем изображение в поток для
передачи в EnhanceImage
                        using (var stream = new MemoryStream())
                        {
                            rgbImage.Save(stream, new
PngEncoder());
                            byte[] enhancedImageBytes =
stream.ToArray();

                            // Улучшаем изображение
                            var enhancedImage =
EnhanceImage(enhancedImageBytes);

                            // Распознаем текст с помощью
Tesseract
                            string tessdataPath =
@"E:\tesseractorc\tessdata"; // Укажите путь к tessdata
                            using (var engine = new
TesseractEngine(tessdataPath, "rus+eng", EngineMode.Default))
                            {
```

```
                                using (var pix =
Pix.LoadFromMemory(enhancedImageBytes))
                                {
                                    var ocrResult =
engine.Process(pix);
                                    result =
ocrResult.GetText();
                                }
                            }
                        }
                    }
                }
                catch (Exception ex)
                {
                    Console.WriteLine($"Ошибка распознавания
текста: {ex.Message}");
                }
            }
            return result;
        }
        public static void InsertDataToManticore(string
filePath, string content, string summary)
        {
            byte[] utf8Bytes = Encoding.UTF8.GetBytes(content);
            string utf8Content =
Encoding.UTF8.GetString(utf8Bytes);
            string connectionString =
"Server=127.0.0.1;Port=9306;Database=default;User
ID=root;Charset=utf8;";

            using (MySqlConnection conn = new
MySqlConnection(connectionString))
            {
                try
                {
                    conn.Open();

                    string currentDate =
DateTime.Now.ToString("yyyy-MM-dd HH:mm:ss");

                    string query = @"
                INSERT INTO pdf_index (id, file_path, content,
summary, date_added)
                VALUES (@id, @file_path, @content, @summary,
@date_added);";

                    using (MySqlCommand cmd = new
MySqlCommand(query, conn))
                    {
                        cmd.Parameters.AddWithValue("@id",
GetUniqueId());
```

```
cmd.Parameters.AddWithValue("@file_path", filePath);
                        cmd.Parameters.AddWithValue("@content",
content);
                        cmd.Parameters.AddWithValue("@summary",
summary);

cmd.Parameters.AddWithValue("@date_added", currentDate);

                        int rowsAffected =
cmd.ExecuteNonQuery();
                        Console.WriteLine($"Добавлено
{rowsAffected} записей.");
                    }
                }
                catch (Exception ex)
                {
                    Console.WriteLine("Ошибка: " + ex.Message);
                }
            }
        }
        public static List<(string FilePath, string Summary)>
SearchInManticore(string searchQuery)
        {
            var results = new List<(string, string)>();
            string connectionString =
"Server=127.0.0.1;Port=9306;Database=default;User
ID=root;Charset=utf8;";

            try
            {
                using (MySqlConnection conn = new
MySqlConnection(connectionString))
                {
                    conn.Open();

                    string query = @"
                SELECT file_path, summary
                FROM pdf_index
                WHERE MATCH(@searchQuery);
            ";

                    using (MySqlCommand cmd = new
MySqlCommand(query, conn))
                    {

cmd.Parameters.AddWithValue("@searchQuery", searchQuery);

                        using (MySqlDataReader reader =
cmd.ExecuteReader())
                        {
```

```
                            while (reader.Read())
                            {
                                string filePath =
reader.GetString("file_path");
                                string summary =
reader.GetString("summary");
                                results.Add((filePath,
summary));
                            }
                        }
                    }
                }
            }
            catch (Exception ex)
            {
                Console.WriteLine("Ошибка при поиске в
Manticore: " + ex.Message);
            }

            return results;
        }
        private static int GetUniqueId()
        {
            return
(int)DateTimeOffset.UtcNow.ToUnixTimeSeconds(); // Уникальный ID
на основе времени
        }
        public static string SummarizeText(string text, int
maxSentences = 50)
        {
            // Разделяем текст на предложения
            List<string> sentences = SplitIntoSentences(text);

            // Ограничиваем количество предложений
            string limitedText = string.Join(" ",
sentences.Take(maxSentences));

            string pythonScriptPath =
@"E:\DiplomProject\ConsoleApp2\ConsoleApp2\summarize_text.py";
// Укажите путь к скрипту
            string pythonExecutable = @"python"; // Или полный
путь к python.exe

            // Сохраняем текст во временный файл
            string tempInputFilePath = Path.GetTempFileName();
            File.WriteAllText(tempInputFilePath, limitedText,
Encoding.UTF8);

            ProcessStartInfo start = new ProcessStartInfo
            {
                FileName = pythonExecutable,
```

```
                Arguments = $"{pythonScriptPath}
\"{tempInputFilePath}\"", // Передаём путь к файлу
                UseShellExecute = false,
                RedirectStandardOutput = true,
                RedirectStandardError = true,
                StandardOutputEncoding = Encoding.UTF8,
                CreateNoWindow = true
            };

            using (Process process = Process.Start(start))
            {
                using (StreamReader reader =
process.StandardOutput)
                {
                    // Читаем путь к временному файлу с
результатом
                    string tempOutputFilePath =
reader.ReadLine();
                    Console.WriteLine("Временный файл с
результатом: " + tempOutputFilePath);

                    // Читаем результат из временного файла
                    string summary =
File.ReadAllText(tempOutputFilePath, Encoding.UTF8);

                    // Удаляем временные файлы
                    File.Delete(tempInputFilePath);
                    File.Delete(tempOutputFilePath);

                    return summary;
                }
            }
        }
        private void backgroundWorker1_DoWork(object sender,
DoWorkEventArgs e)
        {
            for (int i = 0; i < selectedFiles.Count; i++)
            {
                if (backgroundWorker.CancellationPending)
                {
                    e.Cancel = true;
                    return;
                }

                string filePath = selectedFiles[i];
                backgroundWorker.ReportProgress((i * 100) /
selectedFiles.Count, $"Обработка:
{Path.GetFileName(filePath)}");

                try
                {
```

```
                    // Добавляем частые проверки отмены
                    var text =
ExtractTextAndImagesFromPDF(filePath);
                    if (backgroundWorker.CancellationPending) {
e.Cancel = true; return; }

                    string fullText = string.Join(" ", text);
                    if (backgroundWorker.CancellationPending) {
e.Cancel = true; return; }

                    string summary = SummarizeText(fullText);
                    if (backgroundWorker.CancellationPending) {
e.Cancel = true; return; }

                    InsertDataToManticore(filePath, fullText,
summary);
                    if (backgroundWorker.CancellationPending) {
e.Cancel = true; return; }

                    this.Invoke((MethodInvoker)delegate
                    {
                        var item = new
ListViewItem(Path.GetFileName(filePath));
                        item.SubItems.Add(filePath);
                        item.SubItems.Add(summary);
                        item.Tag = filePath;
                        resultsListView.Items.Add(item);


resultsListView.AutoResizeColumns(ColumnHeaderAutoResizeStyle.Co
lumnContent);

resultsListView.AutoResizeColumns(ColumnHeaderAutoResizeStyle.He
aderSize);

                        tabControl1.SelectedTab = tabPage2;
                    });
                }
                catch (Exception ex)
                {
                    this.Invoke((MethodInvoker)delegate
                    {
                        MessageBox.Show($"Ошибка обработки
{Path.GetFileName(filePath)}: {ex.Message}",
                                        "Ошибка",
                                        MessageBoxButtons.OK,
                                        MessageBoxIcon.Error);
                    });
                }
            }
        }
```

```
        private void backgroundWorker_ProgressChanged(object
sender, ProgressChangedEventArgs e)
        {
            processingProgressBar.Value = e.ProgressPercentage;
            lblStatus.Text = e.UserState.ToString();
        }

        private void backgroundWorker_RunWorkerCompleted(object
sender, RunWorkerCompletedEventArgs e)
        {
            processingProgressBar.Visible = false;
            btnCancelProcessing.Enabled = true; // Вновь
активируем кнопку отмены

            if (e.Cancelled)
            {
                lblStatus.Text = "Обработка отменена";
                MessageBox.Show("Обработка была отменена
пользователем", "Информация",
                                MessageBoxButtons.OK,
MessageBoxIcon.Information);
            }
            else if (e.Error != null)
            {
                lblStatus.Text = "Ошибка";
                MessageBox.Show($"Произошла ошибка:
{e.Error.Message}", "Ошибка",
                                MessageBoxButtons.OK,
MessageBoxIcon.Error);
            }
            else
            {
                lblStatus.Text = "Готово";
                MessageBox.Show($"Обработано
{selectedFiles.Count} файлов", "Завершено",
                                MessageBoxButtons.OK,
MessageBoxIcon.Information);
            }
        }
        private List<string> selectedFiles = new List<string>();
        public Form1()
        {
            this.WindowState = FormWindowState.Maximized;
            InitializeComponent();

            InitializePdfViewer();
            // Инициализация BackgroundWorker
            backgroundWorker = new BackgroundWorker
            {
                WorkerReportsProgress = true,
```

```
                WorkerSupportsCancellation = true
            };
            backgroundWorker.DoWork += backgroundWorker1_DoWork;
            backgroundWorker.ProgressChanged +=
backgroundWorker_ProgressChanged;
            backgroundWorker.RunWorkerCompleted +=
backgroundWorker_RunWorkerCompleted;
        }

        private void txtFilePath_TextChanged(object sender,
EventArgs e)
        {

        }

        private void Form1_Load(object sender, EventArgs e)
        {
            resultsListView.Columns.Add("Имя файла", 200);
            resultsListView.Columns.Add("Путь", 300);
            resultsListView.Columns.Add("Краткое содержание",
400);

resultsListView.AutoResizeColumns(ColumnHeaderAutoResizeStyle.Co
lumnContent);

resultsListView.AutoResizeColumns(ColumnHeaderAutoResizeStyle.He
aderSize);
        }

        private void btnSelectFile_Click(object sender,
EventArgs e)
        {
            using (OpenFileDialog openFileDialog = new
OpenFileDialog())
            {
                openFileDialog.Filter = "PDF files
(*.pdf)|*.pdf";
                if (openFileDialog.ShowDialog() ==
DialogResult.OK)
                {
                    txtFilePath.Text = openFileDialog.FileName;
                    selectedFiles.Clear();
                    selectedFiles.Add(openFileDialog.FileName);
                }
            }
        }

        private void btnSelectFolder_Click(object sender,
EventArgs e)
        {
```

```
            using (FolderBrowserDialog folderDialog = new
FolderBrowserDialog())
            {
                if (folderDialog.ShowDialog() ==
DialogResult.OK)
                {
                    string[] pdfFiles =
Directory.GetFiles(folderDialog.SelectedPath, "*.pdf");
                    if (pdfFiles.Length > 0)
                    {
                        txtFilePath.Text =
folderDialog.SelectedPath;
                        selectedFiles.Clear();
                        selectedFiles.AddRange(pdfFiles);
                    }
                    else
                    {
                        MessageBox.Show("В выбранной папке нет
PDF файлов.");
                    }
                }
            }

        }

        private void btnStartProcessing_Click(object sender,
EventArgs e)
        {
            if (selectedFiles.Count == 0)
            {
                MessageBox.Show("Выберите файл(ы) для
обработки.");
                return;
            }

            // Очищаем предыдущие результаты
            resultsListView.Items.Clear();

            // Настраиваем интерфейс
            processingProgressBar.Visible = true;
            processingProgressBar.Style =
ProgressBarStyle.Marquee;
            btnCancelProcessing.Enabled = true;
            lblStatus.Text = "Обработка...";

            // Запускаем обработку
            backgroundWorker.RunWorkerAsync();

        }

        private void btnSearch_Click(object sender, EventArgs e)
```

```
        {
            if (string.IsNullOrWhiteSpace(txtSearchQuery.Text))
            {
                MessageBox.Show("Введите текст для поиска");
                return;
            }

            resultsListView.Items.Clear();
            var foundFiles =
SearchInManticore(txtSearchQuery.Text);

            foreach (var (filePath, summary) in foundFiles)
            {
                var item = new
ListViewItem(Path.GetFileName(filePath));
                item.SubItems.Add(filePath);
                item.SubItems.Add(summary); // Не обрезаем текст
                item.Tag = filePath;
                resultsListView.Items.Add(item);
            }

            // Автоматически подгоняем ширину колонок под
содержимое

resultsListView.AutoResizeColumns(ColumnHeaderAutoResizeStyle.Co
lumnContent);

resultsListView.AutoResizeColumns(ColumnHeaderAutoResizeStyle.He
aderSize);

            if (resultsListView.Items.Count == 0)
            {
                MessageBox.Show("Ничего не найдено");
            }
        }

        private void resultsListView_DoubleClick(object sender,
EventArgs e)
        {
            if (resultsListView.SelectedItems.Count > 0)
            {
                string filePath =
resultsListView.SelectedItems[0].Tag?.ToString();

                if (string.IsNullOrEmpty(filePath))
                {
                    MessageBox.Show("Путь к файлу не указан.");
                    return;
                }

                if (!File.Exists(filePath))
```

```
                {
                    MessageBox.Show("Файл не найден по
указанному пути.");
                    return;
                }

                try
                {
                    Process.Start(new ProcessStartInfo(filePath)
{ UseShellExecute = true });
                }
                catch (Exception ex)
                {
                    MessageBox.Show($"Ошибка открытия файла:
{ex.Message}");
                }
            }
        }

        private void InitializePdfViewer()
        {
            // Настройка обработчиков событий
            btnPrevPage.Click += (s, e) =>
            {
                NavigatePdfPage(-1);
                UpdatePdfToolbar(); // Обновляем после изменения
страницы
            };

            btnNextPage.Click += (s, e) =>
            {
                NavigatePdfPage(1);
                UpdatePdfToolbar(); // Обновляем после изменения
страницы
            };

            btnClosePdf.Click += (s, e) =>
TogglePdfViewer(false);

            // Начальные настройки
            btnPrevPage.Text = "←";
            btnNextPage.Text = "→";
            btnClosePdf.Text = "Закрыть";
            pdfRenderer.Visible = false;
            pdfToolPanel.Visible = false;
        }

        private void NavigatePdfPage(int delta)
        {
            if (pdfRenderer.Document == null) return;
```

```
            int newPage = pdfRenderer.Page + delta;
            if (newPage >= 0 && newPage <
pdfRenderer.Document.PageCount)
            {
                pdfRenderer.Page = newPage;
                // Не вызываем UpdatePdfToolbar() здесь, так как
это делается в обработчиках кнопок
            }
        }

        private void TogglePdfViewer(bool show, string filePath
= null)
        {
            if (show && !string.IsNullOrEmpty(filePath))
            {
                try
                {
                    var document =
PdfiumViewer.PdfDocument.Load(filePath);
                    pdfRenderer.Load(document);

                    pdfRenderer.Visible = true;
                    pdfToolPanel.Visible = true;
                    UpdatePdfToolbar(); // Добавьте этот вызов
                }
                catch (Exception ex)
                {
                    MessageBox.Show($"Ошибка загрузки PDF:
{ex.Message}");
                    TogglePdfViewer(false);
                }
            }
            else
            {
                pdfRenderer.Visible = false;
                pdfToolPanel.Visible = false;
                pdfRenderer.Document?.Dispose();
            }
        }

        private void UpdatePdfToolbar()
        {
            if (pdfRenderer.Document != null && lblPageInfo !=
null)
            {
                // Обратите внимание: Page + 1, так как
нумерация страниц обычно начинается с 1
                lblPageInfo.Text = $"Страница: {pdfRenderer.Page
+ 1}/{pdfRenderer.Document.PageCount}";
            }
        }
```

```
        private void resultsListView_SelectedIndexChanged(object
sender, EventArgs e)
        {
            if (resultsListView.SelectedItems.Count == 0)
return;

            string filePath =
resultsListView.SelectedItems[0].Tag?.ToString();

            if (!string.IsNullOrEmpty(filePath) &&
File.Exists(filePath))
            {
                if (Path.GetExtension(filePath).Equals(".pdf",
StringComparison.OrdinalIgnoreCase))
                {
                    TogglePdfViewer(true, filePath);
                }
                else
                {
                    TogglePdfViewer(false);
                }
            }
        }
        protected override void
OnFormClosing(FormClosingEventArgs e)
        {
            pdfRenderer.Document?.Dispose();
            base.OnFormClosing(e);
        }


        private void btnCancel_Click(object sender, EventArgs e)
        {
            if (backgroundWorker.IsBusy)
            {
                backgroundWorker.CancelAsync();
                btnCancelProcessing.Enabled = false;
                lblStatus.Text = "Отмена...";
            }
        }

        private void pdfToolPanel_Paint(object sender,
PaintEventArgs e)
        {

        }
    }
}
```

Файл «summarize_text.py»

```
from sumy.parsers.plaintext import PlaintextParser
from sumy.nlp.tokenizers import Tokenizer
from sumy.summarizers.lsa import LsaSummarizer
import sys
import tempfile

def summarize(text, sentences_count=5):
    # Используем токенизатор для русского языка
    parser = PlaintextParser.from_string(text,
Tokenizer("russian"))
    summarizer = LsaSummarizer()
    summary = summarizer(parser.document, sentences_count)
    return " ".join(str(sentence) for sentence in summary)

if __name__ == "__main__":
    # Чтение пути к входному файлу из аргументов командной
строки
    input_file_path = sys.argv[1]
    with open(input_file_path, 'r', encoding='utf-8') as
input_file:
        text = input_file.read()

    # Суммируем текст
    summary = summarize(text)

    # Сохраняем результат во временный файл
    with tempfile.NamedTemporaryFile(mode='w', delete=False,
encoding='utf-8') as temp_file:
        temp_file.write(summary)
        temp_file_path = temp_file.name

    # Выводим путь к временному файлу с результатом
    print(temp_file_path)
```

Файл «Form1.Designer.cs»

```
namespace WinFormsApp1
{
    partial class Form1
    {
        /// <summary>
        ///  Required designer variable.
        /// </summary>
        private System.ComponentModel.IContainer components =
null;

        /// <summary>
        ///  Clean up any resources being used.
```

```
        /// </summary>
        /// <param name="disposing">true if managed resources
should be disposed; otherwise, false.</param>
        protected override void Dispose(bool disposing)
        {
            if (disposing && (components != null))
            {
                components.Dispose();
            }
            base.Dispose(disposing);
        }

        #region Windows Form Designer generated code

        /// <summary>
        ///  Required method for Designer support - do not
modify
        ///  the contents of this method with the code editor.
        /// </summary>
        ///

        private void InitializeComponent()
        {
            txtFilePath = new TextBox();
            txtSearchQuery = new TextBox();
            btnSelectFile = new Button();
            btnSelectFolder = new Button();
            btnStartProcessing = new Button();
            btnSearch = new Button();
            resultsListView = new ListView();
            processingProgressBar = new ProgressBar();
            lblStatus = new Label();
            backgroundWorker1 = new
System.ComponentModel.BackgroundWorker();
            tabControl1 = new TabControl();
            tabPage1 = new TabPage();
            btnCancelProcessing = new Button();
            tabPage2 = new TabPage();
            pdfToolPanel = new Panel();
            lblPageInfo = new Label();
            btnClosePdf = new Button();
            btnNextPage = new Button();
            btnPrevPage = new Button();
            pdfRenderer = new PdfiumViewer.PdfRenderer();
            tabControl1.SuspendLayout();
            tabPage1.SuspendLayout();
            tabPage2.SuspendLayout();
            pdfToolPanel.SuspendLayout();
            SuspendLayout();
            //
            // txtFilePath
```

```
            //
            txtFilePath.Location = new Point(6, 5);
            txtFilePath.Name = "txtFilePath";
            txtFilePath.ReadOnly = true;
            txtFilePath.Size = new Size(482, 27);
            txtFilePath.TabIndex = 0;
            txtFilePath.TextChanged += txtFilePath_TextChanged;
            //
            // txtSearchQuery
            //
            txtSearchQuery.Location = new Point(16, 16);
            txtSearchQuery.Name = "txtSearchQuery";
            txtSearchQuery.Size = new Size(569, 27);
            txtSearchQuery.TabIndex = 1;
            //
            // btnSelectFile
            //
            btnSelectFile.Location = new Point(519, 5);
            btnSelectFile.Name = "btnSelectFile";
            btnSelectFile.Size = new Size(117, 29);
            btnSelectFile.TabIndex = 2;
            btnSelectFile.Text = "Выбрать файл";
            btnSelectFile.UseVisualStyleBackColor = true;
            btnSelectFile.Click += btnSelectFile_Click;
            //
            // btnSelectFolder
            //
            btnSelectFolder.Location = new Point(667, 5);
            btnSelectFolder.Name = "btnSelectFolder";
            btnSelectFolder.Size = new Size(121, 29);
            btnSelectFolder.TabIndex = 3;
            btnSelectFolder.Text = "Выбрать папку";
            btnSelectFolder.UseVisualStyleBackColor = true;
            btnSelectFolder.Click += btnSelectFolder_Click;
            //
            // btnStartProcessing
            //
            btnStartProcessing.Location = new Point(824, 5);
            btnStartProcessing.Name = "btnStartProcessing";
            btnStartProcessing.Size = new Size(94, 29);
            btnStartProcessing.TabIndex = 4;
            btnStartProcessing.Text = "Старт";
            btnStartProcessing.UseVisualStyleBackColor = true;
            btnStartProcessing.Click +=
btnStartProcessing_Click;
            //
            // btnSearch
            //
            btnSearch.Location = new Point(626, 16);
            btnSearch.Name = "btnSearch";
            btnSearch.Size = new Size(94, 29);
```

```
            btnSearch.TabIndex = 5;
            btnSearch.Text = "Поиск";
            btnSearch.UseVisualStyleBackColor = true;
            btnSearch.Click += btnSearch_Click;
            //
            // resultsListView
            //
            resultsListView.FullRowSelect = true;
            resultsListView.Location = new Point(16, 115);
            resultsListView.Name = "resultsListView";
            resultsListView.Size = new Size(703, 647);
            resultsListView.TabIndex = 6;
            resultsListView.UseCompatibleStateImageBehavior =
false;
            resultsListView.View = View.Details;
            resultsListView.SelectedIndexChanged +=
resultsListView_SelectedIndexChanged;
            resultsListView.DoubleClick +=
resultsListView_DoubleClick;
            //
            // processingProgressBar
            //
            processingProgressBar.Location = new Point(6, 52);
            processingProgressBar.Name =
"processingProgressBar";
            processingProgressBar.Size = new Size(406, 29);
            processingProgressBar.TabIndex = 7;
            processingProgressBar.Visible = false;
            //
            // lblStatus
            //
            lblStatus.AutoSize = true;
            lblStatus.Location = new Point(534, 70);
            lblStatus.Name = "lblStatus";
            lblStatus.Size = new Size(21, 20);
            lblStatus.TabIndex = 8;
            lblStatus.Text = "\"\"";
            //
            // backgroundWorker1
            //
            backgroundWorker1.WorkerReportsProgress = true;
            backgroundWorker1.WorkerSupportsCancellation = true;
            backgroundWorker1.DoWork +=
backgroundWorker1_DoWork;
            //
            // tabControl1
            //
            tabControl1.Controls.Add(tabPage1);
            tabControl1.Controls.Add(tabPage2);
            tabControl1.Location = new Point(-5, -3);
            tabControl1.Name = "tabControl1";
```

```
tabControl1.SelectedIndex = 0;
tabControl1.Size = new Size(1915, 841);
tabControl1.TabIndex = 9;
//
// tabPage1
//
tabPage1.Controls.Add(btnCancelProcessing);
tabPage1.Controls.Add(txtFilePath);
tabPage1.Controls.Add(lblStatus);
tabPage1.Controls.Add(btnSelectFile);
tabPage1.Controls.Add(processingProgressBar);
tabPage1.Controls.Add(btnSelectFolder);
tabPage1.Controls.Add(btnStartProcessing);
tabPage1.Location = new Point(4, 29);
tabPage1.Name = "tabPage1";
tabPage1.Padding = new Padding(3);
tabPage1.Size = new Size(1907, 808);
tabPage1.TabIndex = 0;
tabPage1.Text = "Загрузка";
tabPage1.UseVisualStyleBackColor = true;
//
// btnCancelProcessing
//
btnCancelProcessing.Location = new Point(946, 5);
btnCancelProcessing.Name = "btnCancelProcessing";
btnCancelProcessing.Size = new Size(94, 29);
btnCancelProcessing.TabIndex = 9;
btnCancelProcessing.Text = "Отмена";
btnCancelProcessing.UseVisualStyleBackColor = true;
btnCancelProcessing.Click += btnCancel_Click;
//
// tabPage2
//
tabPage2.Controls.Add(pdfRenderer);
tabPage2.Controls.Add(pdfToolPanel);
tabPage2.Controls.Add(txtSearchQuery);
tabPage2.Controls.Add(resultsListView);
tabPage2.Controls.Add(btnSearch);
tabPage2.Location = new Point(4, 29);
tabPage2.Name = "tabPage2";
tabPage2.Padding = new Padding(3);
tabPage2.Size = new Size(1907, 808);
tabPage2.TabIndex = 1;
tabPage2.Text = "Поиск";
tabPage2.UseVisualStyleBackColor = true;
//
// pdfToolPanel
//
pdfToolPanel.Controls.Add(lblPageInfo);
pdfToolPanel.Controls.Add(btnClosePdf);
pdfToolPanel.Controls.Add(btnNextPage);
```

```
            pdfToolPanel.Controls.Add(btnPrevPage);
            pdfToolPanel.Location = new Point(901, 65);
            pdfToolPanel.Name = "pdfToolPanel";
            pdfToolPanel.Size = new Size(1006, 43);
            pdfToolPanel.TabIndex = 8;
            pdfToolPanel.Paint += pdfToolPanel_Paint;
            //
            // lblPageInfo
            //
            lblPageInfo.AutoSize = true;
            lblPageInfo.Location = new Point(97, 24);
            lblPageInfo.Name = "lblPageInfo";
            lblPageInfo.Size = new Size(21, 20);
            lblPageInfo.TabIndex = 3;
            lblPageInfo.Text = "\"\"";
            //
            // btnClosePdf
            //
            btnClosePdf.Location = new Point(664, 13);
            btnClosePdf.Name = "btnClosePdf";
            btnClosePdf.Size = new Size(94, 29);
            btnClosePdf.TabIndex = 2;
            btnClosePdf.Text = "button1";
            btnClosePdf.UseVisualStyleBackColor = true;
            //
            // btnNextPage
            //
            btnNextPage.Location = new Point(491, 13);
            btnNextPage.Name = "btnNextPage";
            btnNextPage.Size = new Size(94, 29);
            btnNextPage.TabIndex = 1;
            btnNextPage.Text = "button1";
            btnNextPage.UseVisualStyleBackColor = true;
            //
            // btnPrevPage
            //
            btnPrevPage.Location = new Point(325, 13);
            btnPrevPage.Name = "btnPrevPage";
            btnPrevPage.Size = new Size(94, 29);
            btnPrevPage.TabIndex = 0;
            btnPrevPage.Text = "button1";
            btnPrevPage.UseVisualStyleBackColor = true;
            //
            // pdfRenderer
            //
            pdfRenderer.Location = new Point(901, 114);
            pdfRenderer.Name = "pdfRenderer";
            pdfRenderer.Page = 0;
            pdfRenderer.Rotation =
PdfiumViewer.PdfRotation.Rotate0;
            pdfRenderer.Size = new Size(1006, 648);
```

```
            pdfRenderer.TabIndex = 9;
            pdfRenderer.Text = "pdfRenderer1";
            pdfRenderer.ZoomMode =
PdfiumViewer.PdfViewerZoomMode.FitHeight;
            //
            // Form1
            //
            AutoScaleDimensions = new SizeF(8F, 20F);
            AutoScaleMode = AutoScaleMode.Font;
            ClientSize = new Size(1952, 985);
            Controls.Add(tabControl1);
            Name = "Form1";
            Text = "Form1";
            Load += Form1_Load;
            tabControl1.ResumeLayout(false);
            tabPage1.ResumeLayout(false);
            tabPage1.PerformLayout();
            tabPage2.ResumeLayout(false);
            tabPage2.PerformLayout();
            pdfToolPanel.ResumeLayout(false);
            pdfToolPanel.PerformLayout();
            ResumeLayout(false);
        }

        #endregion

        private TextBox txtFilePath;
        private TextBox txtSearchQuery;
        private Button btnSelectFile;
        private Button btnSelectFolder;
        private Button btnStartProcessing;
        private Button btnSearch;
        private ListView resultsListView;
        private ProgressBar processingProgressBar;
        private Label lblStatus;
        private System.ComponentModel.BackgroundWorker
backgroundWorker1;
        private TabControl tabControl1;
        private TabPage tabPage1;
        private TabPage tabPage2;
        private Button btnCancelProcessing;
        private PdfiumViewer.PdfRenderer pdfRenderer;
        private Panel pdfToolPanel;
        private Label lblPageInfo;
        private Button btnClosePdf;
        private Button btnNextPage;
        private Button btnPrevPage;
    }
}
```

Приложение В

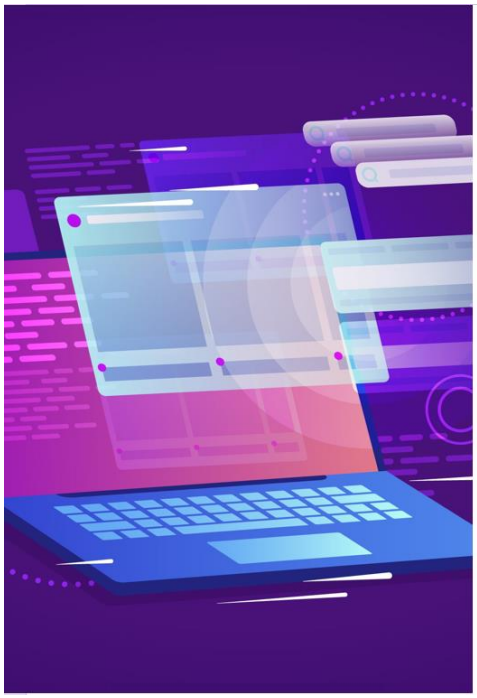
(Обязательное)

Система полнотекстового поиска по PDF-документам с распознаванием текста на изображениях

Презентация

Система полнотекстового поиска по PDF-документам с распознаванием текста на изображениях

Автор работы: студент группы 21ВВП2
Столяров Артем Алексеевич
Научный руководитель: доцент кафедры ВТ
Сорокин Алексей Александрович



Актуальность

PDF
Основной формат хранения

Проблемы с поиском PDF
41% PDF – сканы без текстового слоя

Рост объема цифровых документов
Возникает потребность в эффективной обработке и поиске информации

Цель работы

Разработка программы

Локальная система поиска на ПК с ОС Windows

Использование OCR

Технология оптического распознавания символов

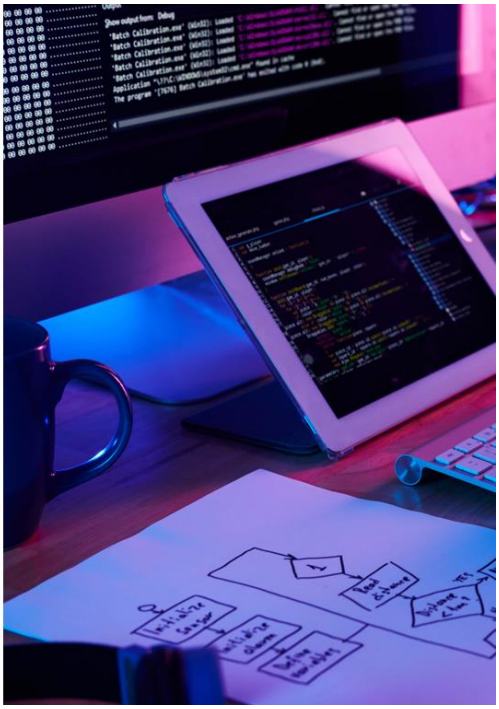
Автономность

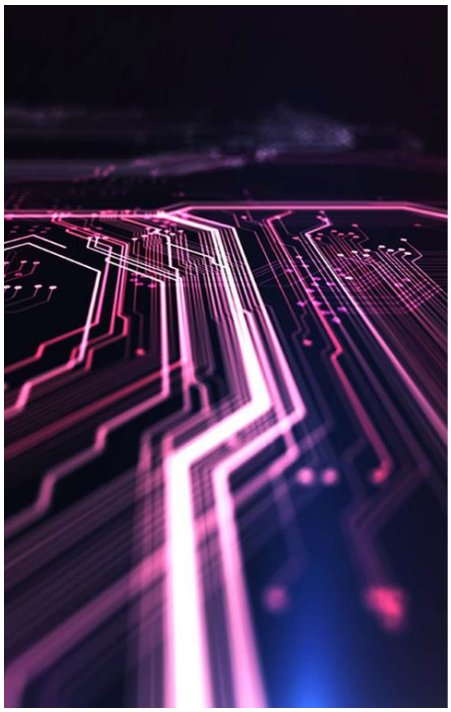
Работа без подключения к интернету

Функциональные требования

Основные функции

- Извлечение текста и изображений из PDF
- Распознавание текста на изображениях
- Индексирование данных
- Поиск по документам
- Реферирование текста





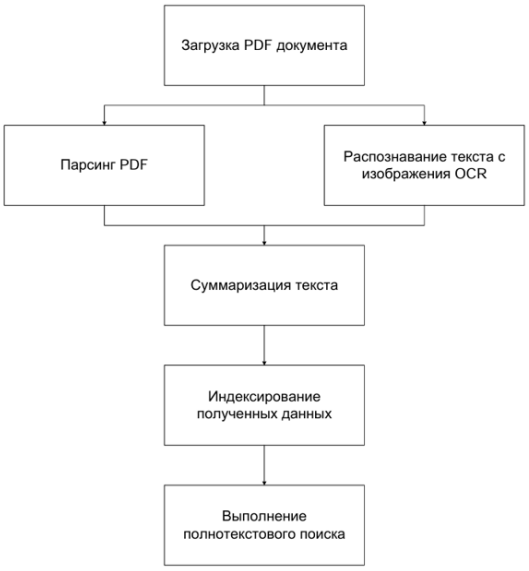
Анализ аналогов

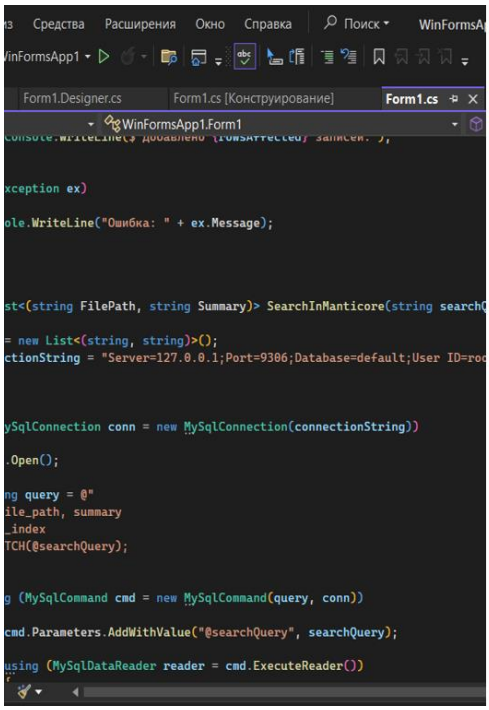
Adobe Acrobat Pro
ABBYY FineReader
Google Drive
Amazon Textract
Parless-ngx

Вывод: большинство решений являются платными или не поддерживают OCR полностью

Архитектура системы

- Импорт PDF
- Извлечение текста и изображений
- OCR на изображениях
- Реферирование текста
- Индексация данных
- Поиск и отображение





Технологии

C#
Основной язык разработки

Visual Studio
Среда разработки

Manticore Search
Поисковая система

Tesseract OCR
Оптическое распознавание символов

iText7
Извлечение текста

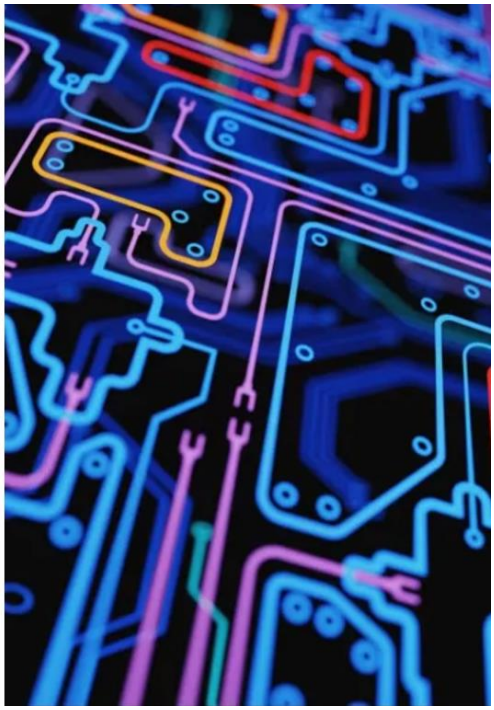
Sumy
Реферирование текста

Реализация программы

Загрузка PDF
Выбор как одного PDF файла, так и папки с файлами

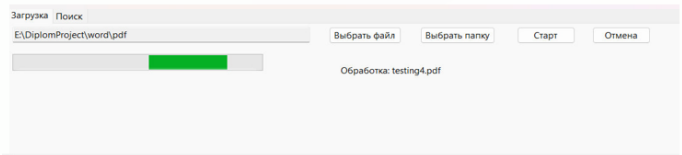
Обработка
Извлечение, реферирование, индексация

Поиск
Вывод результатов поиска и открытие PDF в проигрывателе

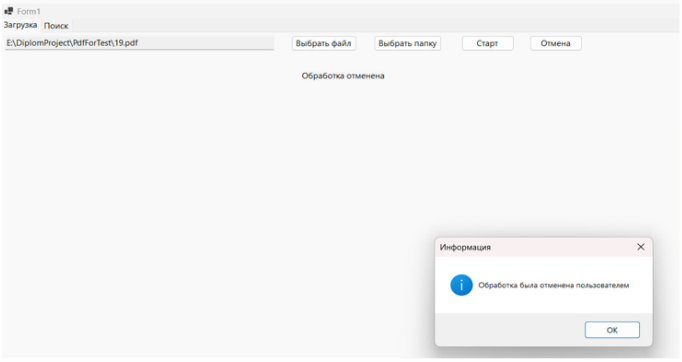


Демонстрация функционала

Запуск обработки файлов

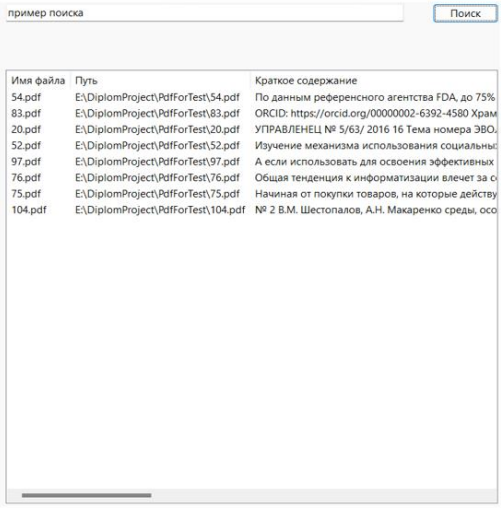


Прерывание обработки



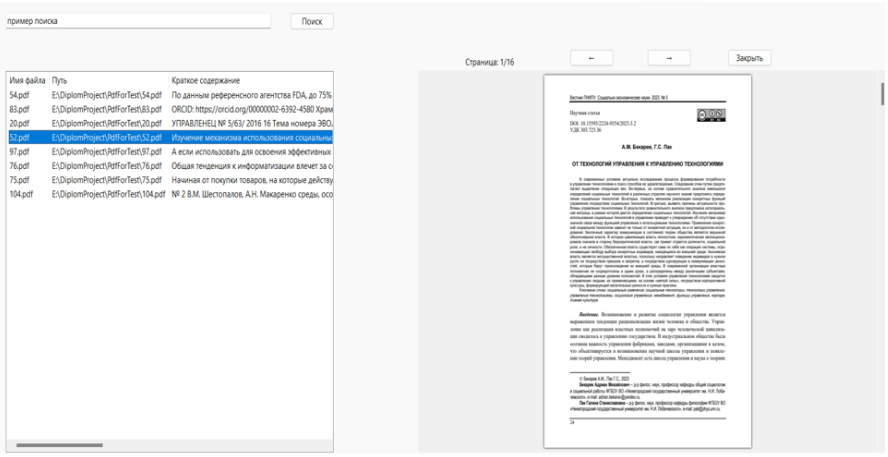
Демонстрация функционала

Полнотекстовый поиск по файлам



Демонстрация функционала

Открытие файла в проигрывателе



Спасибо за внимание!