Министерство науки и высшего образования Российской Федерации

Федеральное государственное автономное образовательное учреждение

высшего образования

«Казанский (Приволжский) федеральный университет»

*Институт вычислительной математики и информационных технологий*

**ОТЧЁТ**

**по производственной технологической (проектно-технологической) практике**

|  |  |
| --- | --- |
| Обучающийся Шанин С.А., 09-051  *(ФИО, группа)* | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  *(подпись)* |

Руководитель практики от КФУ

ст. преподаватель кафедры анализа данных

и технологий программирования Жажнева И.В. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*(должность, ФИО) (подпись)*

Оценка за практику \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись руководителя)

Дата сдачи отчета 23.05.2024

Казань, 2024

Содержание

[АННОТАЦИЯ 3](#_Toc166489014)

[Введение 4](#_Toc166489015)

[1. Разработка серверной части системы 5](#_Toc166489016)

[1.1. Описание системы хранения отчетов и ее интеграция с бд 5](#_Toc166489016)

[1.2. Реализация API с использованием Flask 8](#_Toc166489016)

[1.3. Взаимодействие с базой данных через SQLAlchemy 10](#_Toc166489016)

[2. Разработка клиентской части системы 12](#_Toc166489016)

[3. Анализ технической документации 21](#_Toc166489016)

[4. Тестирование функциональной части 31](#_Toc166489021)

[Заключение 4](#_Toc166489022)8

[Список использованных источников 49](#_Toc166489023)

[ПРИЛОЖЕНИЕ 50](#_Toc166489024)

АННОТАЦИЯ

Цель настоящей работы заключается в разработке десктопной рекомендательной системы, которая позволяет автоматизировать процесс проверки технической документации на соответствие стандартам ГОСТ. В рамках данной работы была исследована предметная область, проведен сравнительный анализ готовых решений, составлен ряд функциональных и нефункциональных требований к системе, спроектирована схема базы данных. Была затронута проблема несоответствия документации стандартам, для решения которой была разработана система с возможностью динамического обновления правил проверки.

Серверная часть системы была разработана с применением архитектурных принципов, подходящих для десктопных приложений. Создание сервера производилось с использованием программной платформы PostgreSQL и ряда дополнительных библиотек, таких как SQLAlchemy для взаимодействия с базой данных.

В ходе разработки клиентской части системы были подобраны подходящие решения для дизайна интерфейса. Для создания пользовательского интерфейса были применены библиотеки PyQt, а также ряд дополнительных библиотек для улучшения взаимодействия пользователя с системой.

Ключевые слова: рекомендательная система, ГОСТ, техническая документация, автоматизация, десктопное приложение, PostgreSQL, SQLAlchemy, PyQt, дизайн интерфейса.

# Введение

Целью производственной технологической (проектно-технологической) практики направления подготовки 09.03.03 «Прикладная информатика» согласно программе практики является закрепление теоретических знаний, практических умений и навыков, полученных в процессе освоения дисциплин основной образовательной программы приобретение практического опыта решения профессиональных задач. При прохождении данной практики обучающийся опирается на материалы ранее освоенных дисциплин (модулей) и/или практик: Информатика, Информационные технологии, Архитектура информационных систем, Управление данными, Технологии программирования, Ознакомительная практика.

Освоение данной практики способствует эффективному выполнению и защите выпускной квалификационной работы.

Основными задачами производственной технологической (проектно-технологической) практики являются:

- овладеть навыками использования отечественных и международных стандартов, норм и правил; навыками разработки технической документации на всех стадиях жизненного цикла информационной системы;

- овладеть практическими навыками разработки программного обеспечения;

- овладеть навыками выполнения работ на всех стадиях жизненного цикла создания информационных систем; навыками использования инструментальных средств управления проектами в области разработки информационных систем.

Технологической (проектно-технологической) практика проходила на базе Института вычислительной математики и информационных технологий Казанского (Приволжского) федерального университета.

Сроки прохождения практики: с 08.04.2024 по 23.05.2024.

## 1. Разработка серверной части системы

## 1.1. Описание системы хранения отчетов и её интеграции с БД

В этом разделе представлено подробное описание структуры базы данных, которая служит фундаментом для работы рекомендательной системы (рисунок 1). Особое внимание уделяется атрибутам таблиц, ранее введенным во второй главе, и механизму хранения отчетов, который обеспечивает эффективный доступ и управление результатами анализа. Эта информация критически важна для понимания того, как система обрабатывает и хранит данные, что является ключевым аспектом её функциональности.

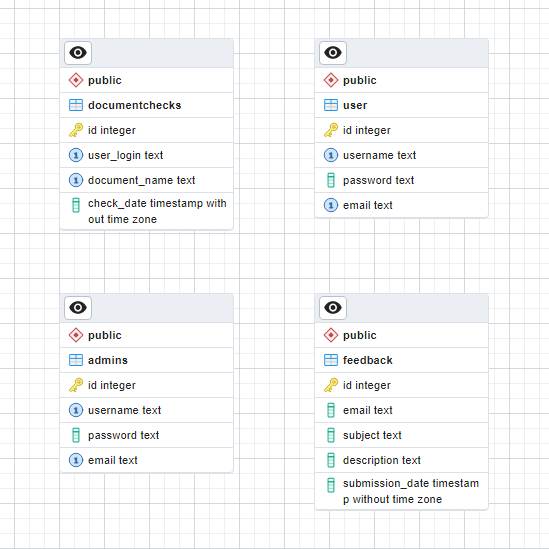


Рисунок 1. Схема базы данных

Каждая таблица базы данных спроектирована с целью оптимизации хранения и обработки данных. Ниже представлены ключевые атрибуты таблиц:

Таблица User:

* id: Integer, первичный ключ;
* username: Text, уникальное, не может быть NULL;
* password: Text, не может быть NULL;
* email: Text, уникальное, не может быть NULL.

Таблица Admins:

* id: Integer, первичный ключ;
* username: Text, уникальное, не может быть NULL;
* password: Text, не может быть NULL;
* email: Text, уникальное, не может быть NULL.

Таблица Feedback:

* id: Integer, первичный ключ;
* email: Text, не может быть NULL;
* subject: Text, не может быть NULL;
* description: Text, не может быть NULL.

Таблица DocumentChecks:

* id: Integer, первичный ключ;
* user\_login: Text, не может быть NULL;
* document\_name: Text, не может быть NULL;
* check\_date: TIMESTAMP.

Каждая таблица спроектирована таким образом, чтобы обеспечить эффективное хранение и быстрый доступ к данным, что является неотъемлемой частью функционирования рекомендательной системы. Уникальные ограничения и индексы используются для оптимизации запросов и обеспечения целостности данных.

Отчеты, генерируемые системой, представляют собой важный элемент, обеспечивающий пользователей информацией о результатах анализа документов. Отчеты сохраняются в файловой системе сервера и связываются с соответствующими записями в таблицах Users и Documents. Это позволяет пользователям легко находить и получать доступ к своим отчетам. Имена файлов отчетов формируются на основе уникального идентификатора пользователя и документа, что обеспечивает их уникальность и предотвращает конфликты имен. Отчеты организованы в иерархическую структуру каталогов, что упрощает навигацию и поиск необходимых файлов. Доступ к отчетам контролируется через систему прав пользователей, что гарантирует, что только авторизованные пользователи могут получить доступ к своим отчетам (рисунок 2).

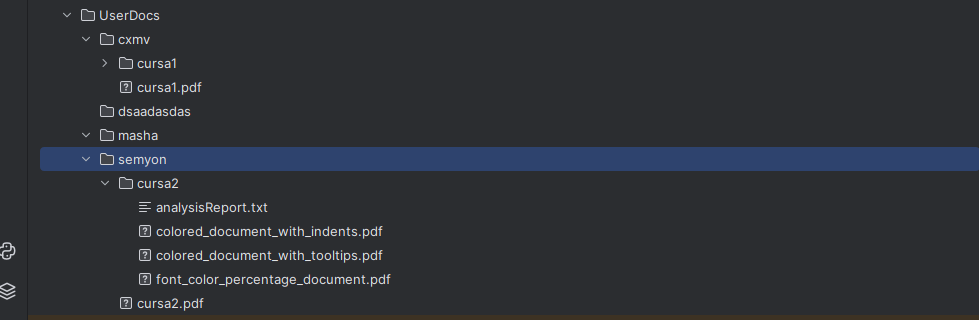


Рисунок 2. Структура каталога отчетов

Структура хранения отчетов организована следующим образом:

* Путь к отчетам: Отчеты располагаются по адресу UserDocs/username/, где username - это уникальное имя пользователя в системе.
* Организация файлов: Внутри папки пользователя, каждый документ и связанные с ним отчеты находятся в отдельной папке, названной в соответствии с именем документа.
* Именование отчетов: Имена файлов отчетов формируются таким образом, чтобы отражать уникальный идентификатор пользователя и документа, что исключает возможность дублирования и конфликтов имен.
* Доступ к отчетам: Система прав пользователей регулирует доступ к отчетам, гарантируя, что только авторизованные пользователи имеют возможность просматривать или скачивать свои отчеты.

Такая организация не только упрощает управление данными, но и повышает безопасность, предотвращая несанкционированный доступ к чувствительной информации. Это обеспечивает надежную и эффективную работу рекомендательной системы, делая процесс анализа документов максимально прозрачным и доступным для пользователей.

## 1.2. Реализация API с использованием Flask

Flask был выбран в качестве основы для создания API, так как он предоставляет гибкость и удобство в разработке, что особенно важно при работе с внешними сервисами. В случае данного проекта, Flask позволяет создавать как веб-интерфейсы, так и бэкенд для десктопных приложений, обеспечивая единообразие и универсальность взаимодействия с системой. Это делает его идеальным выбором для разработки масштабируемых приложений, которые могут поддерживать как веб-браузеры, так и десктопные клиенты.

В этом разделе описывается процесс создания API с использованием микрофреймворка Flask, который предоставляет интерфейс для регистрации и аутентификации пользователей, обработки обратной связи и проверки документов.

Основные точки доступа API:

* Регистрация пользователя (/register): Эндпоинт принимает данные пользователя и регистрирует новый аккаунт, если такой ещё не существует.
* Вход в систему (/login): Эндпоинт проверяет учетные данные пользователя и предоставляет доступ к системе.
* Обратная связь (/feedback): Эндпоинт позволяет пользователям отправлять отзывы или сообщения об ошибках.
* Смена пароля (/change\_password): Эндпоинт позволяет пользователю изменить свой текущий пароль.
* Получение списка документов (/documents): Возвращает список всех документов, доступных в системе.
* Работа с документами пользователя (/documents/<user\_login>): Позволяет добавлять новые документы для проверки или получать список документов конкретного пользователя.
* Получение отзывов (/reviews): Возвращает список всех отзывов, оставленных пользователями.
* Управление пользователями (/users): Предоставляет информацию о пользователях системы.
* Удаление отзыва (/reviews/<int:review\_id>): Удаляет отзыв по его идентификатору.

Каждый эндпоинт реализован с использованием декоратора @app.route, которые определяют маршруты и типы запросов. Внутри функций эндпоинтов происходит валидация данных, взаимодействие с базой данных через SQLAlchemy и отправка соответствующих ответов клиенту в формате JSON (рисунок 3).

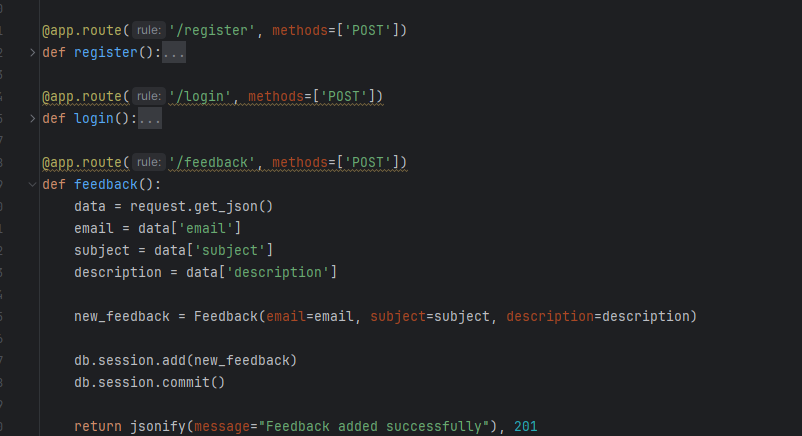


Рисунок 3. Пример эндпоинтов Flask

Этот подход позволяет создать чистый и модульный код, который легко поддерживать и расширять. Важно отметить, что для обеспечения безопасности передачи данных все запросы должны быть защищены с использованием HTTPS, а пароли пользователей хранятся в базе данных в зашифрованном виде.

В продолжение разработки серверной части системы, API, реализованный с помощью Flask, предоставляет функциональность для работы с документами и отзывами. Это включает в себя получение списка документов, добавление новых записей проверок, управление отзывами и получение списка пользователей.

Каждая функция API тщательно разработана для обеспечения корректной обработки запросов и ответов, а также для взаимодействия с базой данных через ORM SQLAlchemy.

Для запуска сервера используется стандартный метод app.run(), который запускает локальный сервер для разработки с включенным режимом отладки. Это позволяет разработчикам тестировать API в реальном времени и быстро вносить изменения в код.

## 1.3. Взаимодействие с базой данных через SQLAlchemy

Для взаимодействия с базой данных в системе используется библиотека SQLAlchemy. SQLAlchemy предоставляет ORM (Object-Relational Mapping) для работы с базами данных, что позволяет обращаться к данным в виде объектов Python, а не напрямую через SQL-запросы.

Подключение к базе данных PostgreSQL происходит с помощью следующих параметров (рисунок 4):

* **SQLALCHEMY\_DATABASE\_URI**: Строка подключения к базе данных. В данном случае, мы используем PostgreSQL, но SQLAlchemy также поддерживает другие СУБД.
* **db = SQLAlchemy(app):** Создаем объект SQLAlchemy, который будет использоваться для взаимодействия с базой данных.



Рисунок 4. Подключение к базе данных

В дополнение к настройкам подключения, в системе используются модели SQLAlchemy для представления таблиц базы данных. Эти модели определяют структуру таблиц и отношения между ними, что позволяет разработчикам взаимодействовать с базой данных на более высоком уровне абстракции. Каждая модель соответствует таблице в базе данных и включает в себя атрибуты, которые представляют поля таблицы (рисунок 5).

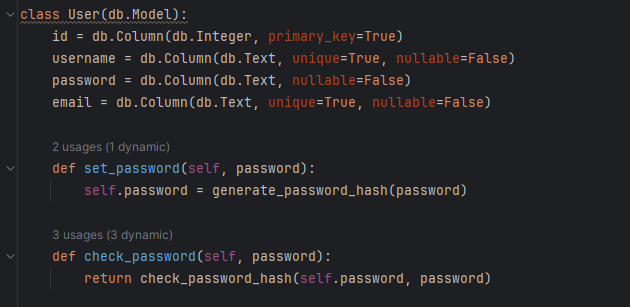


Рисунок 5. Модель таблицы БД в формате SQLAlchemy

Модели облегчают процесс создания, запроса, обновления и удаления записей в базе данных, делая код читаемым и удобным для поддержки. Они также способствуют лучшей организации кода и повышают эффективность разработки за счет предоставления четкой структуры данных и встроенных методов для работы с этими данными.

## 2. Разработка клиентской части системы

Процесс взаимодействия пользователя с программой начинается с окна авторизации, которое является ключевым элементом обеспечения безопасности и персонализации пользовательского опыта . В этом окне представлены поля для ввода имени пользователя и пароля, а также кнопки для выполнения входа в систему и перехода к регистрации нового пользователя (рисунок 6).

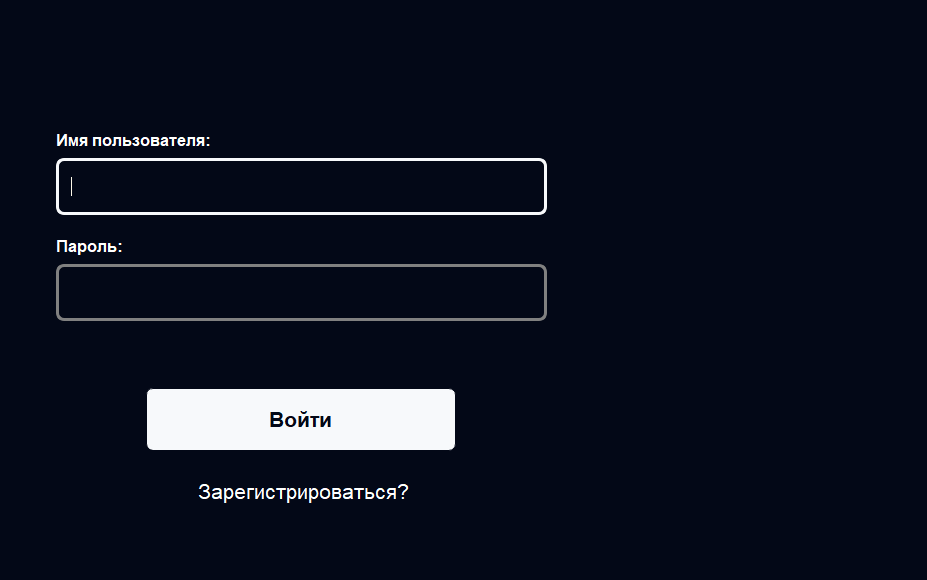


Рисунок 6. Окно авторизации

Если данные верны, то система на Flask отправляет положительный ответ, и пользователь получает доступ к своему личному кабинету. В случае, если данные введены неверно, Flask возвращает сообщение об ошибке, и пользователю предлагается повторить попытку входа. Этот процесс не только упрощает вход в систему, но и поддерживает высокий уровень безопасности за счет проверки данных на сервере. Ниже продемонстрировано как выглядят запросы от интерфейса к API (рисунок 7) и от API к базе данных (рисунок 8). Код остальных запросов будет представлен в разделе диплома “Приложение”. Это позволит подробно ознакомиться с техническими аспектами реализации системы и обеспечит полное понимание процессов обработки данных и коммуникации внутри приложения.

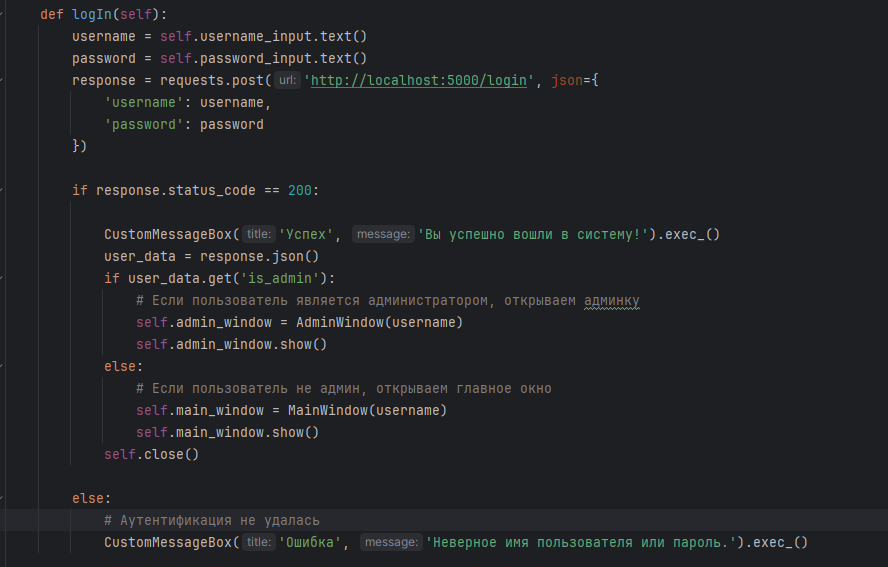


Рисунок 7. Пример функции запроса от интерфейса к бэкенду

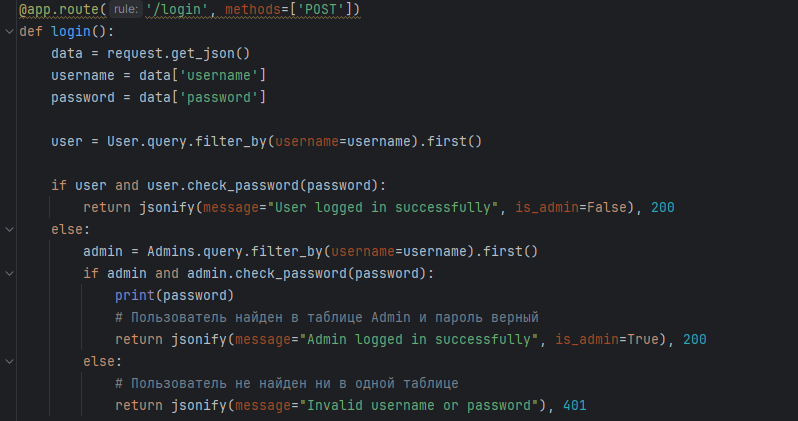


Рисунок 8. Пример запроса к БД

Если пользователь выбирает регистрацию, открывается новое окно, где необходимо ввести имя пользователя, пароль и адрес электронной почты (рисунок 9). После успешной регистрации данные о пользователе добавляются в БД. Подробный анализ взаимодействия интерфейса с базой данных, например регистрация нового пользователя, представлен в разделе диплома “Тестирование”.

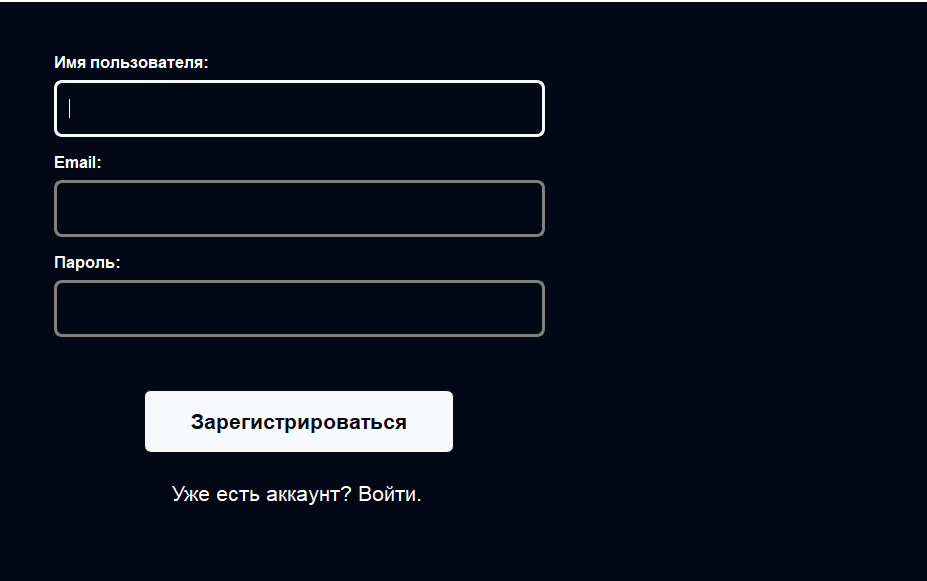


Рисунок 9. Окно регистрации

После успешной авторизации пользователь попадает на главную страницу, которая служит центральным узлом для навигации по системе (рисунок 10). На главной странице расположена панель навигации, которая включает следующие разделы:

* Личный кабинет: Раздел, где пользователь может просмотреть и редактировать свои личные данные.
* Настройки: Секция для изменения настроек системы в соответствии с предпочтениями пользователя.
* Анализ: Интерфейс для загрузки и анализа документов на соответствие стандартам.
* Отчет: Раздел для просмотра отчетов о проверках документов.
* Выйти: Опция для выхода из учетной записи и завершения сеанса работы с системой.

Каждый из этих разделов представлен в виде кнопки, облегчая пользователю доступ к необходимым функциям. Дизайн интерфейса разработан таким образом, чтобы обеспечить интуитивно понятную навигацию и быстрый доступ к основным функциям программы.

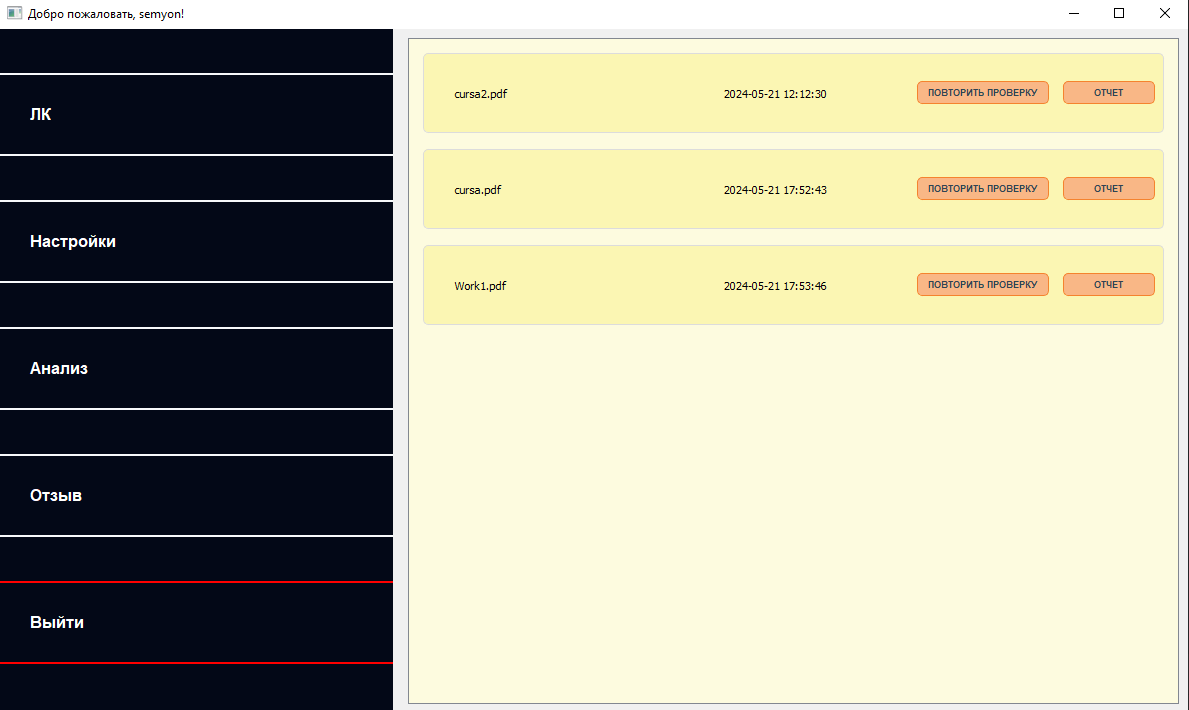


Рисунок 10. Главное меню

Рассмотрим интерфейс при нажатии на кнопку “Настройки” (рисунок 11). Левая часть интерфейса содержит форму для обновления почты и пароля аккаунта. Пользователи могут ввести новые данные и подтвердить изменения, нажав кнопку “Изменить”. Правая часть интерфейса предназначена для изменения пароля. Здесь расположены поля для ввода текущего пароля, нового пароля и его подтверждения. Стилизация кнопок выполнена с помощью QSS, что обеспечивает приятный визуальный вид и улучшает пользовательский опыт.

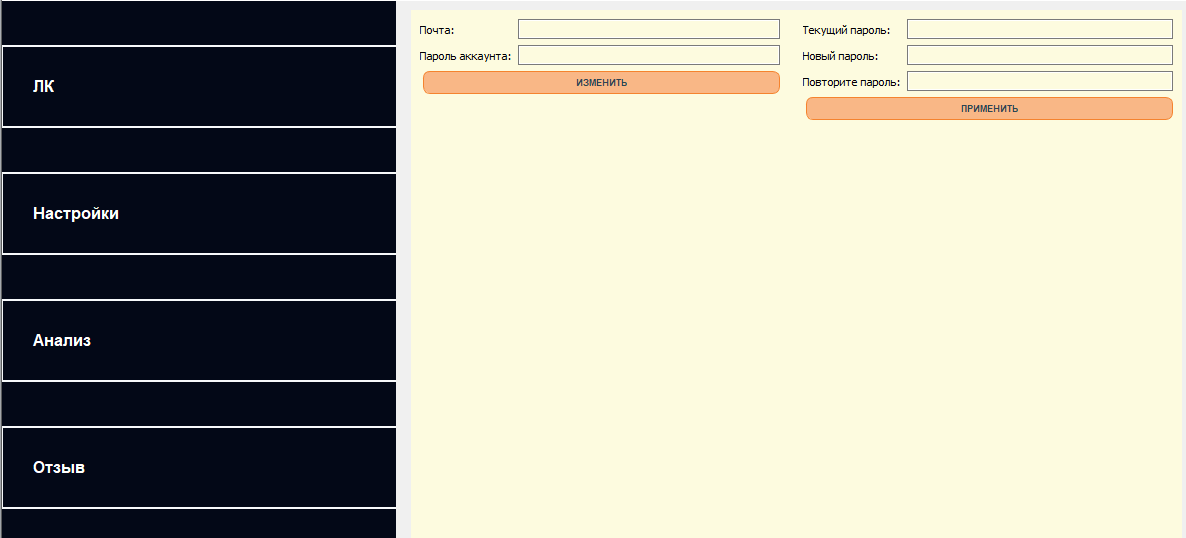


Рисунок 11. Раздел “Настройки”

Далее рассматривается раздел “Отзыв”. Этот раздел предоставляет пользователям возможность делиться своими впечатлениями о работе системы, высказывать предложения или сообщать о проблемах. Такой обратной связи придается большое значение, так как она служит ключевым инструментом для улучшения качества и функциональности приложения.

В окне “Отзывы” пользователи могут оставлять свои комментарии, которые затем отправляются на сервер и обрабатываются администрацией системы (рисунок 12). Это не только способствует двустороннему общению между пользователями и разработчиками, но и создает основу для постоянного совершенствования системы на основе реального пользовательского опыта.

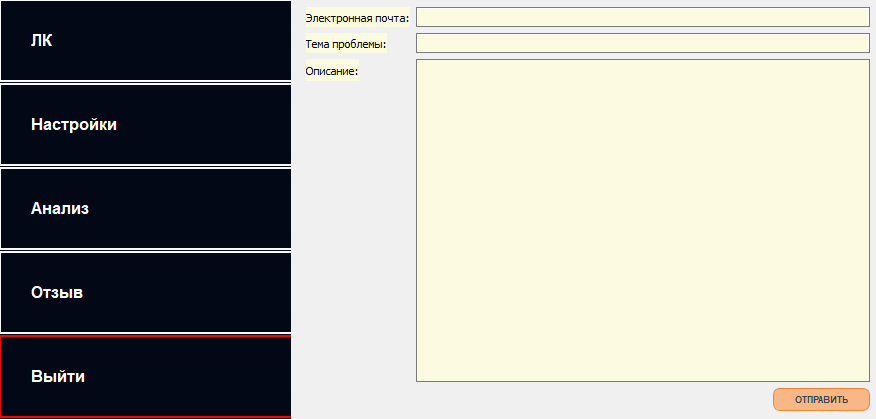


Рисунок 12. “Отзыв”

Раздел “Анализ” предназначен для загрузки и анализа документации пользователями (рисунки 13 и 14). Он включает в себя следующие элементы управления:

* Кнопка “Загрузить документацию”: Позволяет пользователям выбрать и загрузить файлы документации для анализа. Файлы могут быть в форматах PDF, DOCX или TXT.
* Кнопка “Отмена”: Дает возможность отменить текущую загрузку или очистить выбранные файлы перед началом анализа. Эта кнопка активируется только после того, как файл был выбран для загрузки.
* Кнопка “Проверить”: Инициирует процесс анализа загруженной документации. После нажатия кнопки система начинает обработку данных, и пользователь может наблюдать за прогрессом анализа через индикатор выполнения. Кнопка становится активной только после успешной загрузки файла.

После завершения анализа система формирует отчет, который включает в себя обобщенные данные и выявленные проблемы или аномалии в документации. Отчет может быть сохранен в учетной записи пользователя для дальнейшего изучения или отправлен по электронной почте.

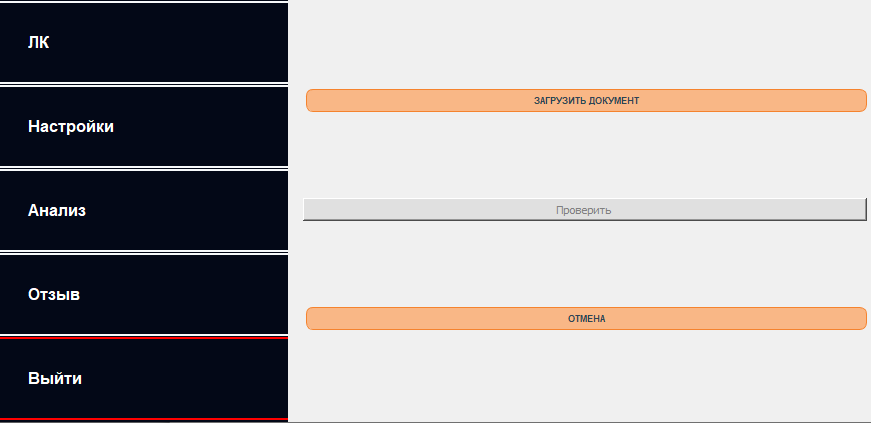


Рисунок 13. Раздел “Анализ”

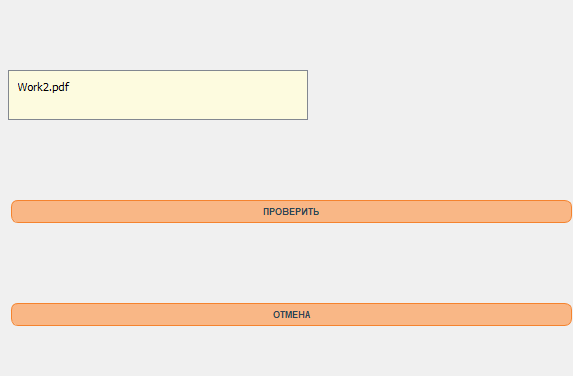


Рисунок 14. Окно после загрузки документа

В разделе “ЛК” представлен удобный список в виде карточек документов, который отображает информацию о документах, прошедших через процесс проверки пользователем. Каждая карточка включает в себя наименование документа, время проверки документа, кнопку “Повторить проверку” и кнопку “Отчет” (рисунок 15).

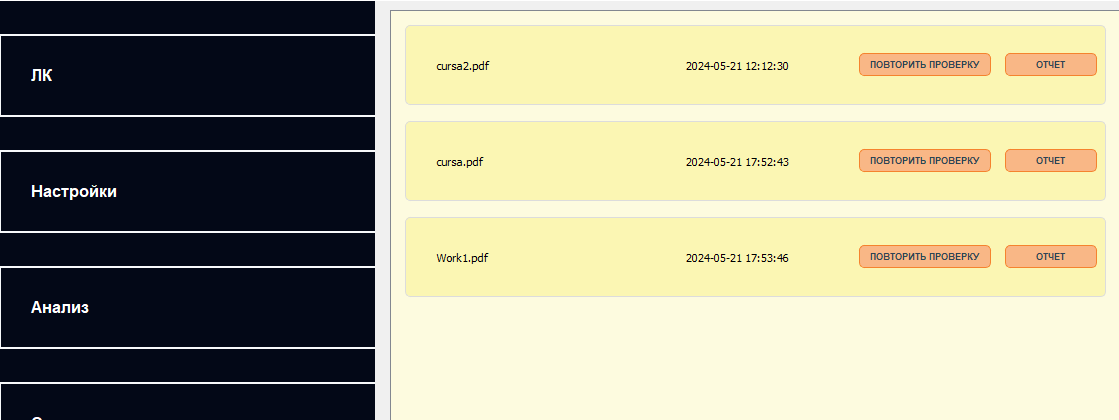


Рисунок 15. Раздел “ЛК”

При нажатии на кнопку “Отчет” открывается окно с четырьмя файлами отчета по результатам проверки (рисунок 16). Пользователь может выбрать, скачать файлы в формате ZIP или экспортировать их по электронной почте для дальнейшего анализа или архивации.

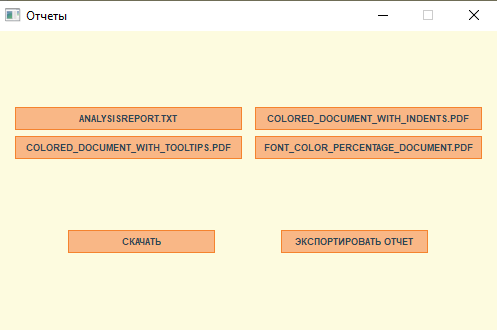


Рисунок 16. Окно просмотра отчета

Карточки документов облегчают навигацию и управление проверенными документами, предоставляя централизованный доступ ко всем результатам проверок.

Кнопка “Выйти” является важным элементом интерфейса, который обеспечивает безопасность и конфиденциальность пользовательских данных. При её нажатии сессия пользователя завершается, и открывается окно авторизации, предлагая пользователю войти в систему снова при необходимости.

Когда пользователь входит в систему с правами администратора, ему открывается специализированная панель управления, которая позволяет осуществлять мониторинг и управление ключевыми аспектами системы. Эта панель включает в себя разделы “Пользователи”, “Отзывы”, “Документы” (рисунок 17).

Раздел пользователи отображает всех пользователей системы с возможностью их удаления.



Рисунок 17. Панель администратора

Раздел “Отзывы” позволяет просматривать запросы пользователей (рисунки 18 и 19).

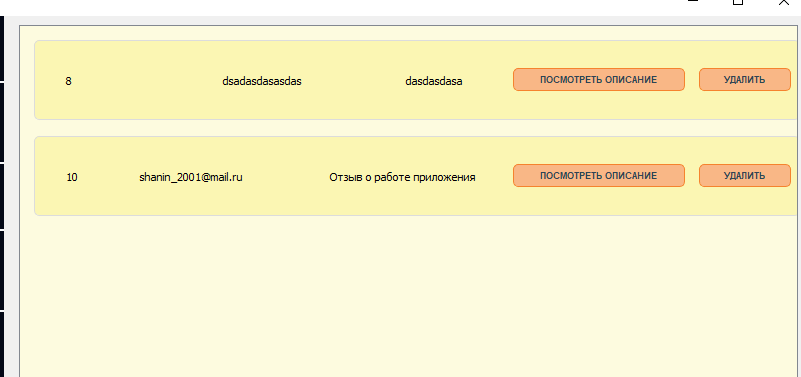


Рисунок 18. Раздел “Отзывы” в панели администратора

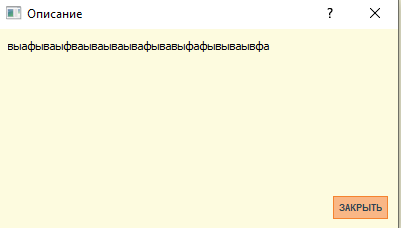


Рисунок 19. Окно просмотра отзыва

Раздел документы позволяет просматривать все документы проверенными любым из пользователей (рисунок 20). Администратор также может просмотреть отчет о проверке.

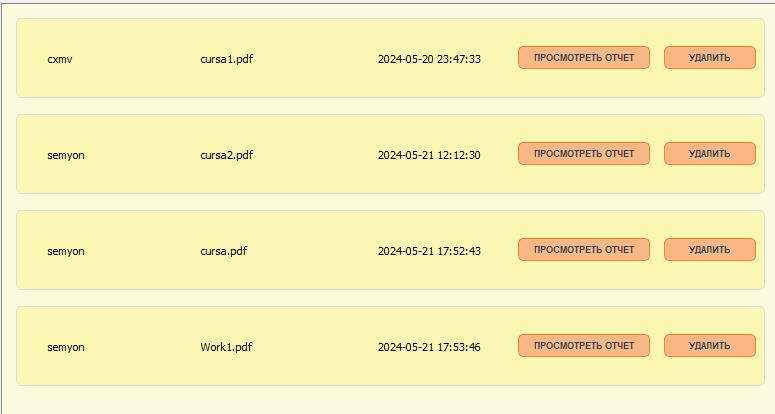


Рисунок 20. Раздел “Документы” в панели администратора

Таким образом, разработка интерфейса становится фундаментом для взаимодействия пользователей с системой, и его качество напрямую влияет на успешность работы всего приложения.

## 3. Анализ технической документации

В данной главе осуществляется детальное исследование процессов и методов, применяемых для анализа технической документации. Цель данного анализа — обеспечить точность и соответствие документов установленным стандартам и требованиям. Для достижения этой цели используется ряд скриптов, каждый из которых выполняет определенную функцию:

* check\_text\_centering\_2 отвечает за проверку центрирования текста в документе.
* find\_sources\_in\_pdf\_3 используется для поиска источников информации внутри PDF-файла.
* font\_and\_size\_color\_4 анализирует шрифты, размер текста и цвет, чтобы убедиться в их единообразии.
* indent\_5 проверяет правильность отступов в тексте.
* numeration\_7 контролирует корректность нумерации разделов и подразделов.
* heading\_finder\_8 помогает определить и проверить заголовки на соответствие стилистическим нормам.
* drawing\_and\_mention\_9 отслеживает упоминания и соответствие рисунков в тексте.
* cosine\_similarity\_10 применяется для оценки степени схожести текстовых фрагментов.

Каждый скрипт руководствуется логикой, направленной на выявление конкретных аспектов документа, и в совокупности они формируют комплексный подход к анализу документации. Код этих скриптов и подробное описание их работы представлены в приложении к данной работе, что позволяет заинтересованным лицам ознакомиться с техническими деталями реализации алгоритмов.

**Скрипт check\_text\_centering\_2** представляет собой инструмент, разработанный для анализа центрирования текста на первой странице PDF-документа с использованием библиотеки PyMuPDF (рисунок 21). Основная логика скрипта заключается в следующем:

* Сначала скрипт открывает указанный PDF-документ и получает доступ к его первой странице.
* Затем он извлекает информацию о всех текстовых блоках на странице в формате словаря. Каждый блок содержит координаты своего расположения на странице.
* Для определения центрирования текста устанавливается порог отклонения. В данном случае порог составляет 50 единиц, что примерно соответствует 1% от ширины стандартной страницы формата A4.
* Скрипт анализирует каждый текстовый блок, вычисляя центр блока и сравнивая его с центром страницы.
* Если расстояние между центром текстового блока и центром страницы меньше установленного порога, текст считается отцентрованным. В противном случае — неотцентрованным.

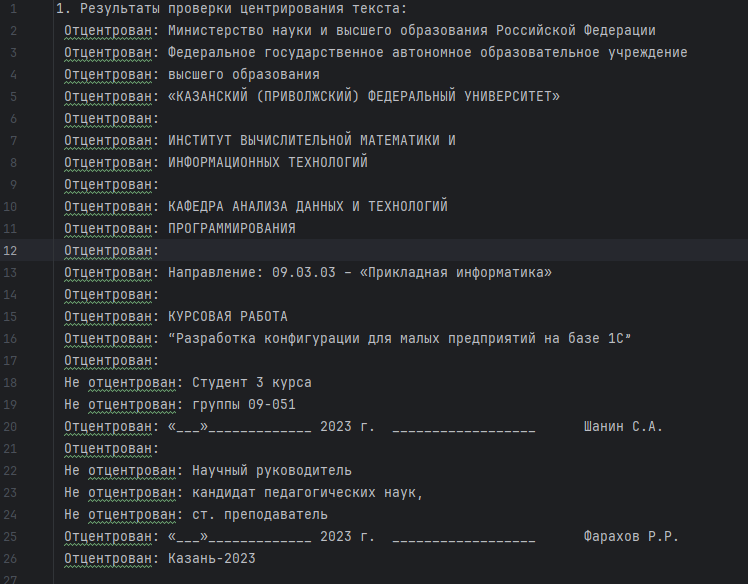


Рисунок 21. Отчет по скрипту check\_text\_centering\_2

**Скрипт find\_sources\_in\_pdf\_3** разработан для проверки оформления электронных ресурсов в списке литературы PDF-документа. Он использует библиотеку PyMuPDF для чтения текста документа и регулярные выражения для поиска соответствия установленным стандартам оформления. Вот как работает скрипт:

* Скрипт открывает PDF-документ и последовательно просматривает каждую страницу в поисках заголовка списка литературы.
* Как только находится строка, начинающаяся с одного из возможных вариантов заголовка списка литературы, скрипт фиксирует этот момент и начинает анализ с этой точки.
* С помощью регулярного выражения скрипт ищет записи об электронных ресурсах, которые должны соответствовать определенному формату: номер источника, описание, указание на электронный ресурс, URL и дата обращения.
* Все найденные совпадения записываются в файл отчета, что позволяет пользователю или администратору системы проверить правильность оформления электронных ресурсов (рисунок 22).

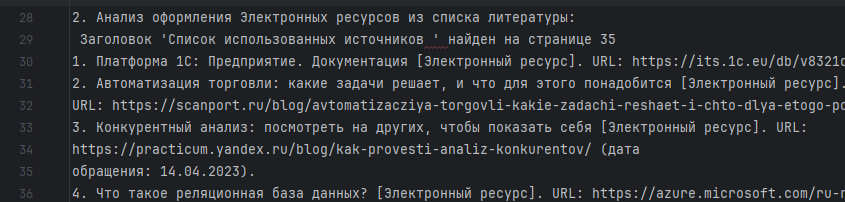


Рисунок 22. Отчет по скрипту find\_sources\_in\_pdf\_3

**Скрипт font\_and\_size\_color\_4** предназначен для анализа и визуализации использования различных шрифтов и размеров текста в PDF-документе. Он выполняет следующие функции:

* Открытие документа: С помощью PyMuPDF скрипт открывает PDF-документ для анализа.
* Сбор данных: Проходит по всем страницам документа, собирая информацию о шрифтах и размерах текста, используемых в документе.
* Уникальные комбинации: Создает словарь уникальных комбинаций шрифтов и размеров, подсчитывая частоту их использования.
* Генерация цветов: Для каждой уникальной комбинации генерируется уникальный цвет.
* Визуализация: Скрипт проходит по страницам документа второй раз, раскрашивая текст в соответствии с ранее сгенерированными цветами и добавляя подсказки с информацией о шрифте и размере.
* Сохранение изменений: Измененный документ сохраняется с раскрашенным текстом и подсказками (рисунок 23).



Рисунок 23. Отчет по скрипту font\_and\_size\_color\_4

* Анализ использования: Рассчитывает процентное соотношение использования каждой комбинации шрифта и размера от общего количества.
* Создание отчета: Генерирует новый документ, который визуально отображает процентное соотношение использования шрифтов и размеров с помощью цветных прямоугольников и текста (рисунок 24).



Рисунок 24. Отчет по скрипту font\_and\_size\_color\_4, в процентах

Этот скрипт полезен для оценки стилистического разнообразия документа и может быть использован для обеспечения единообразия оформления текста. Результаты работы скрипта помогают визуально оценить, какие шрифты и размеры преобладают в документе, что может быть важно при подготовке официальных документов, научных работ или публикаций.

**Скрипт indent\_5** предназначен для анализа отступов в тексте PDF-документа. Он выполняет несколько ключевых задач:

* Скрипт начинает с открытия документа и создания словаря для хранения уникальных отступов и их количества.
* Затем он проходит по страницам документа, собирая данные об отступах каждого текстового блока и подсчитывая их частоту.
* После сбора данных скрипт генерирует уникальные цвета для каждого отступа, чтобы визуализировать их в документе.
* Скрипт также определяет общий цвет для отступов, превышающих определенную величину, что помогает выделить стандартные отступы, используемые в документе.
* В процессе второго прохода по страницам документа скрипт раскрашивает текст в соответствии с ранее определенными цветами и добавляет подсказки с информацией об отступах.
* Измененный документ сохраняется с раскрашенным текстом и подсказками, что позволяет пользователю легко идентифицировать и оценить использование отступов в документе (рисунок 25).

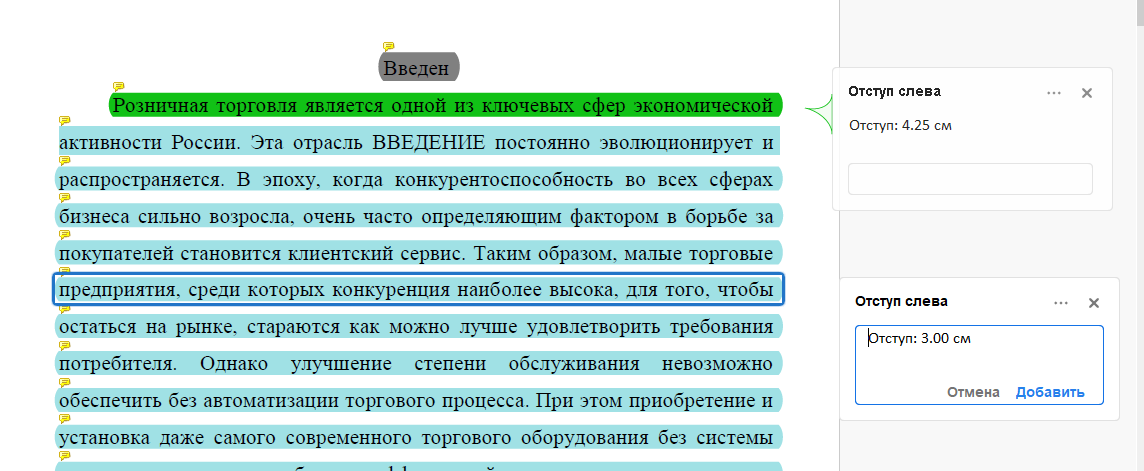


Рисунок 25. Отчет по скрипту indent\_5

**Скрипт numeration\_7** выполняет анализ нумерации страниц в PDF-документе. Вот как он работает:

* Сначала скрипт открывает указанный PDF-документ.
* Затем он проверяет, что номер каждой страницы находится в нижней части страницы.
* Если номер страницы не находится в нижней части или указан буквенными символами, скрипт фиксирует это и добавляет страницу в список неверно пронумерованных.
* По завершении анализа скрипт сохраняет результаты в отчете (рисунок 26).

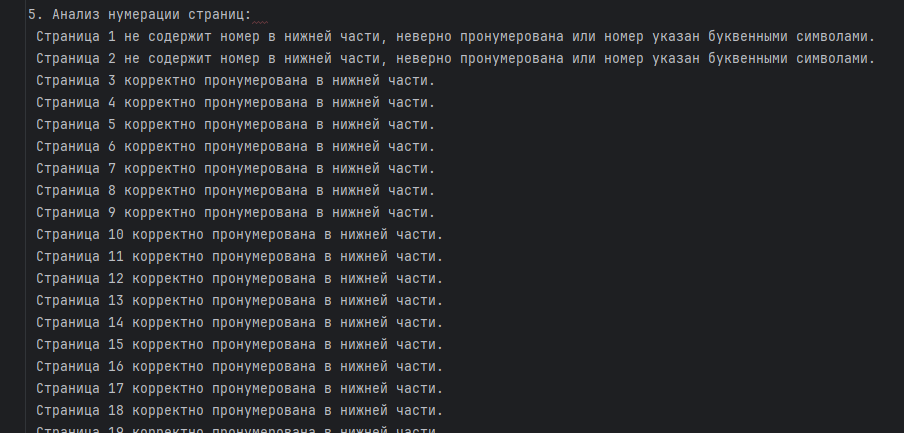


Рисунок 26. Отчет по скрипту numeration\_7

**Скрипт heading\_finder\_8** выполняет анализ соответствия заголовков разделов на страницах содержания (обычно это вторая страница) и номеров самих страниц в PDF-документе. Вот как он работает:

* Сначала скрипт открывает указанный PDF-документ.
* Затем он извлекает текст со страницы содержания, где обычно перечислены заголовки разделов и соответствующие им номера страниц.
* С помощью регулярных выражений скрипт извлекает разделы и номера страниц из текста содержания.
* Далее скрипт проверяет, что каждый заголовок раздела действительно присутствует на соответствующей странице, и записывает результаты в отчет (рисунок 27).

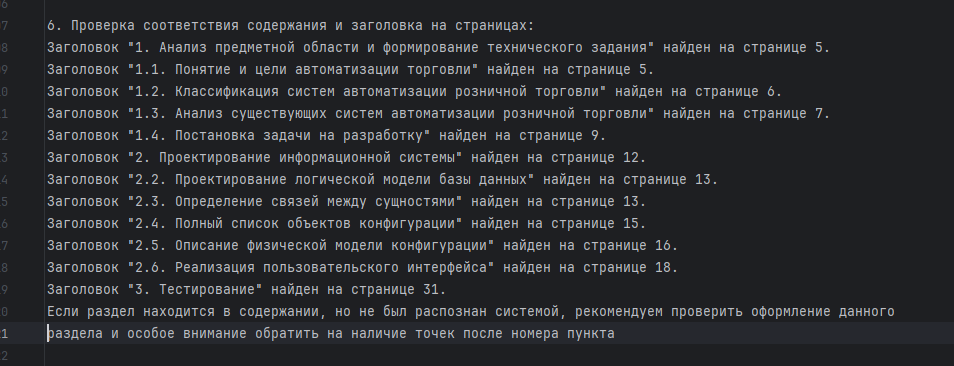


Рисунок 27. Отчет по скрипту heading\_finder\_8

Этот скрипт полезен для обеспечения правильной нумерации разделов и соответствия заголовков на страницах содержания.

**Скрипт drawing\_and\_mention\_9** предназначен для проверки наличия и корректности упоминаний рисунков в PDF-документе. Он выполняет следующие действия:

* Использует регулярные выражения для поиска упоминаний рисунков и соответствующих подписей на страницах документа.
* Создает словарь для отслеживания упоминаний каждого рисунка и страниц, на которых они встречаются.
* Проверяет, что каждое упоминание рисунка сопровождается подписью, соответствующей номеру рисунка.
* Записывает результаты в отчет, указывая, где рисунки корректно упомянуты и описаны, а также выявляет случаи, когда упоминание рисунка не сопровождается подписью (рисунок 28).

Этот скрипт помогает обеспечить, что все рисунки в документе правильно упомянуты и описаны, что является важным аспектом при подготовке научных и технических документов.

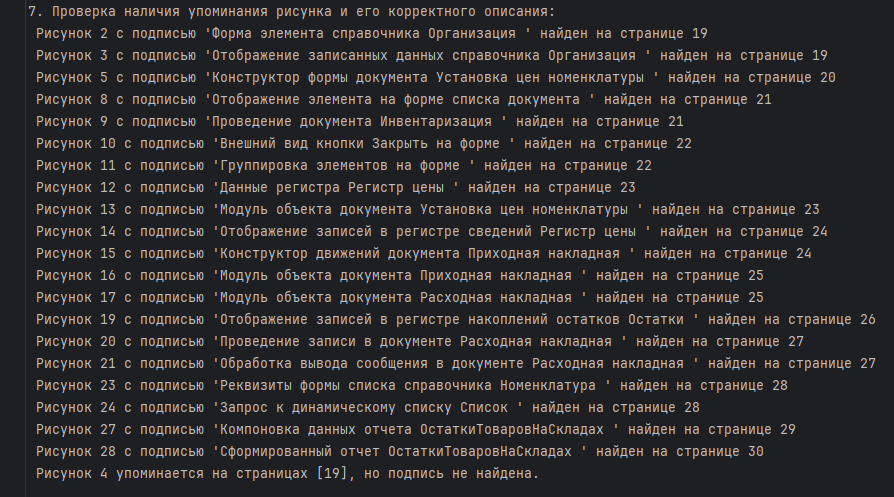


Рисунок 28. Отчет по скрипту drawing\_and\_mention\_9

**Скрипт cosine\_similarity\_10** используется для анализа текстов на предмет схожести, что может указывать на заимствования или повторное использование текстового материала. Он основан на методе косинусного сходства, который является популярным способом сравнения документов в области обработки естественного языка и информационного поиска. Вот более подробное описание работы скрипта:

* Чтение текста: Скрипт начинает с чтения текста из основного PDF-документа, который будет использоваться для сравнения с другими текстами.
* Сбор текстов для сравнения: Затем скрипт читает тексты из всех PDF-файлов в заданной папке, добавляя их в список для последующего сравнения.
* Векторизация текстов: С помощью TfidfVectorizer, который преобразует тексты в векторное пространство, где каждое слово представлено в виде вектора, скрипт подготавливает тексты к сравнению. Это позволяет учесть важность каждого слова в контексте всего корпуса документов.
* Расчет косинусного сходства: Скрипт вычисляет косинусное сходство между вектором основного документа и векторами других текстов. Результатом является числовое значение от -1 до 1, где 1 означает полное сходство, а -1 — полное несходство.
* Анализ результатов: Скрипт анализирует полученные значения сходства, чтобы определить, насколько близок каждый из текстов к основному документу. Это позволяет выявить тексты с высокой степенью схожести, что может указывать на потенциальные заимствования.
* Запись в отчет: Результаты сравнения записываются в отчет, который может быть использован для дальнейшего анализа и принятия решений относительно оригинальности текста (рисунок 29).

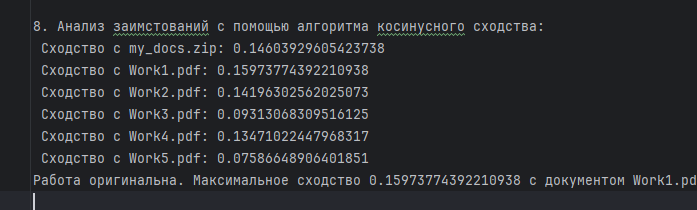


Рисунок 29. Отчет по скрипту cosine\_similarity\_10

Этот скрипт может быть особенно полезен в академической среде для проверки научных работ на предмет плагиата, а также в любой области, где важно поддерживать оригинальность контента. Он предоставляет объективный и количественный способ оценки схожести текстов, что является ценным инструментом в процессе редактирования и проверки документов.

Код всех скриптов доступен в приложении к работе.

## 4. Тестирование функциональной части

Цель тестирования программного обеспечения – обнаружить и исправить проблемы дефекты и неполадки в коде. Это также процесс измерения функциональности и правильности ПО с помощью анализа. Глава, посвященная тестированию, описывает систематический подход к проверке функциональности интерфейса взаимодействия с базами данных. Это включает в себя не только проверку соответствия спецификациям, но и оценку удобства использования, производительности и безопасности системы. В данной главе будут рассмотрены ключевые аспекты , которые необходимо тестировать, чтобы конечный продукт не только работал, как задумано, но и предоставлял пользователю оптимальный опыт.

Исходя из ключевых функций приложения, был разработан следующий план тестирования:

* Регистрация пользователя;
* Изменение пароля и проверка авторизации с новым паролем;
* Отображение документов, связанных с пользователем;
* Возможность скачивания отчета;
* Возможность экспортирования отчета;
* Удаление пользователя;
* Повторная проверка документа;
* Удаление отзыва;
* Тест, что при нажатии на кнопку “проверить” создается директория с файлом.

Этот план тестирования охватывает основные аспекты работы приложения и предназначен для обеспечения его надежности и удобства использования. Каждый пункт плана направлен на проверку конкретного функционала, что позволяет выявить потенциальные проблемы и убедиться в корректности работы всех элементов системы.

1. Регистрация пользователя

При открытии окна регистрации пользователь видит перед собой форму с полями для заполнения. Эти поля могут включать информацию, такую как имя пользователя, электронная почта, пароль и другие необходимые данные (рисунок 30).

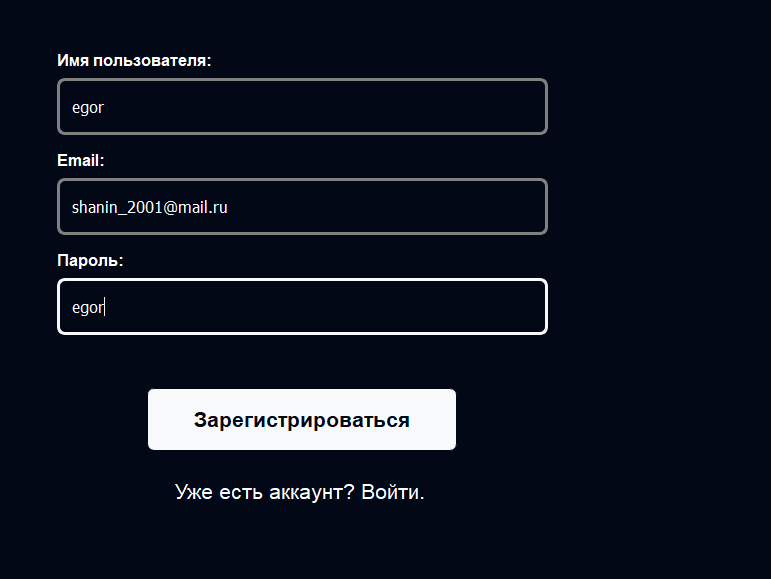


Рисунок 30. Заполнение данных при регистрации

Пользователь заполняет все необходимые поля и нажимает кнопку “Регистрация”. После этого происходит проверка введенных данных, и если все в порядке, информация о новом пользователе передается в базу данных (рисунок 31).

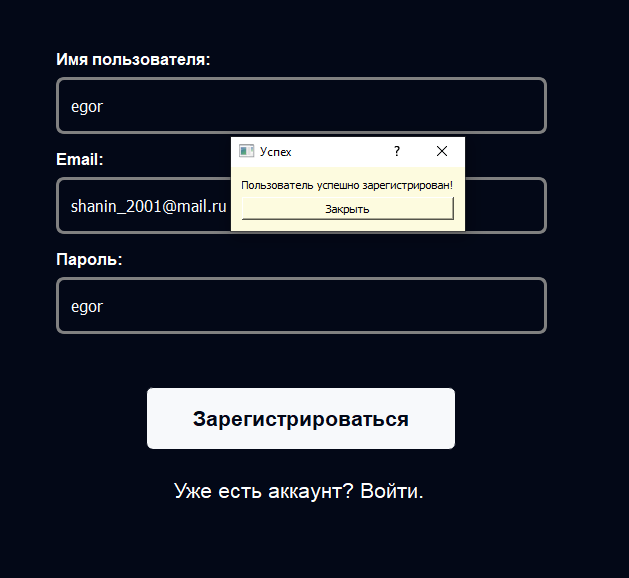


Рисунок 31. Успешная регистрация пользователя

В базе данных создается новая запись, соответствующая зарегистрированному пользователю. Это подтверждает успешное добавление пользователя в систему (рисунок 32).

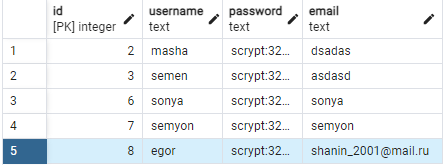


Рисунок 32. Добавление пользователя в БД

2. Изменение пароля

В разделе настроек главного окна программы пользователь начинает процесс изменения пароля. Первым шагом является ввод старого пароля, затем пользователь должен ввести новый пароль дважды для подтверждения.

Если введенные новые пароли не совпадают, система выведет уведомление об ошибке, предлагая пользователю повторить попытку (рисунок 33).

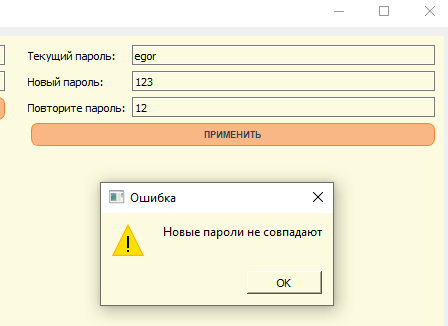


Рисунок 33. Новые пароли не совпадают

После успешного изменения пароля, в базе данных происходит обновление соответствующей записи. Хотя визуально пароль остается неизменным, его хэш-код изменится, что подтверждает изменение пароля в системе (рисунок 34).

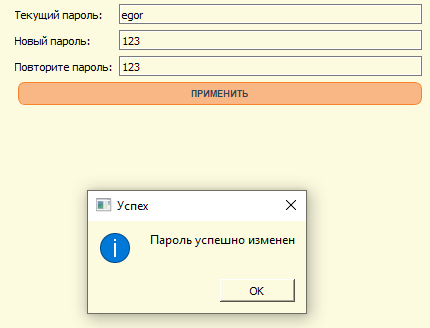


Рисунок 34. Успешное изменение пароля

После изменения пароля пользователь пытается авторизоваться в системе, чтобы убедиться, что изменения вступили в силу.

При попытке входа со старым паролем система должна выдать ошибку, указывая на то, что пароль больше не действителен (рисунок 35).

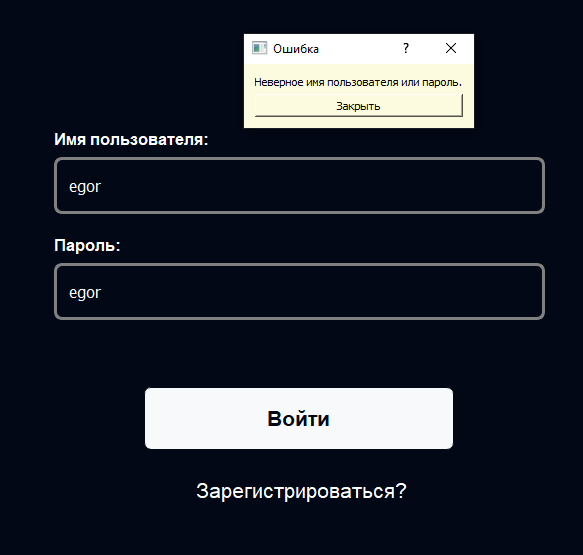


Рисунок 35. Попытка авторизации со старым паролем

При входе с новым паролем система должна разрешить доступ и вывести уведомление об успешной авторизации (рисунок 36).

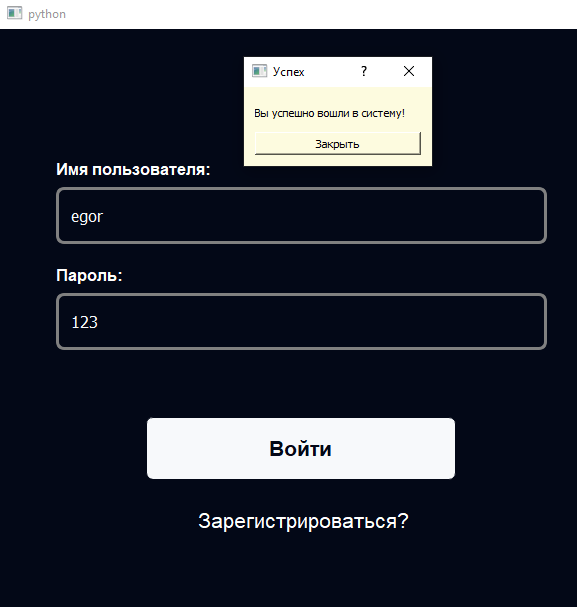


Рисунок 36. Авторизация с новым паролем

3. Отображение документов, связанных с пользователем

Пользователь, находясь в разделе личного кабинета (ЛК) главного окна приложения, имеет доступ к документам, которые привязаны непосредственно к его учетной записи. Это обеспечивает персонализированный и безопасный доступ к данным.

В интерфейсе ЛК отображается список документов, которые связаны с логином пользователя (рисунок 37). Каждый документ имеет уникальный идентификатор, состоящий из комбинации логина пользователя и имени документа.

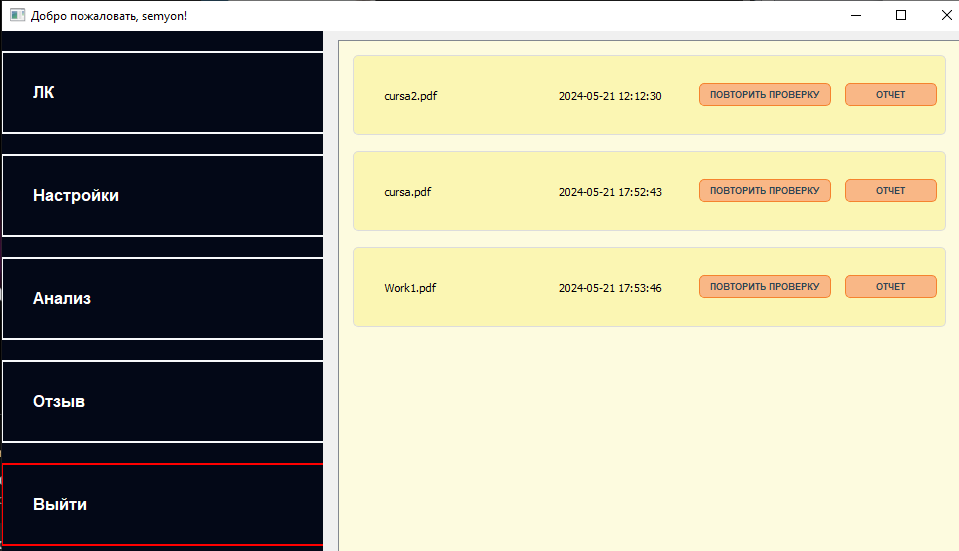


Рисунок 37. Отображение документов, связанных с пользователем, в интерфейсе

В БД проводится проверка, что для данного пользователя существуют только те документы, которые отображаются в его ЛК (рисунок 38). Это подтверждает корректность связи документов с учетной записью пользователя .

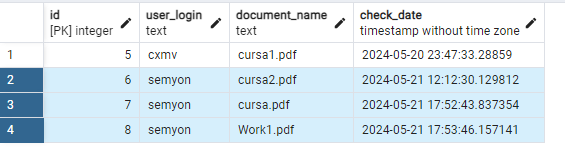


Рисунок 38. Отображение документов в БД

В директории проекта под путем UserDocs/username/filename находится директория, содержащая документы пользователя и связанные с ними отчеты. Это обеспечивает легкий доступ к файлам и подтверждает их наличие в системе (рисунок 39).

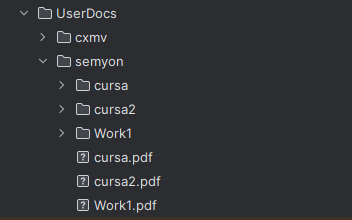


Рисунок 39. Директория с отчетами пользователя

5. Возможность скачивания отчета

Пользователь, находясь в своем личном кабинете, имеет возможность скачать отчеты, связанные с его аккаунтом.

Пользователь выбирает интересующий его отчет и нажимает на кнопку “Отчет”. Открывается окно с четырьмя доступными отчетами (рисунок 40).

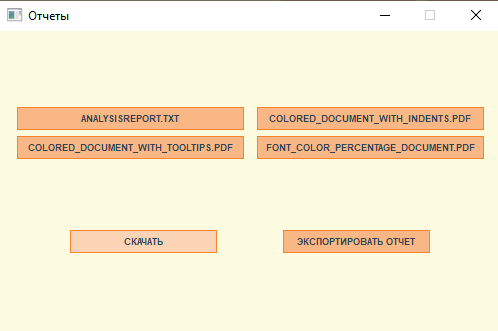


Рисунок 40. Реализация скачивания отчета

После нажатия на кнопку “Скачать” появляется диалоговое окно, где пользователь может выбрать путь для сохранения файла. Отчет сохраняется в формате ZIP (рисунок 41).

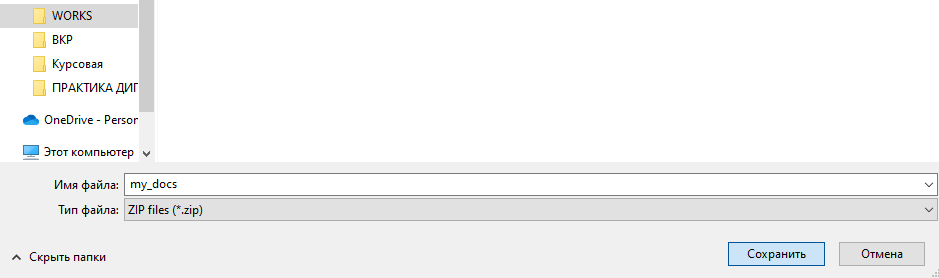


Рисунок 41. Открытие диалогового окна для выбора места загрузки

В указанной пользователем директории появляется ZIP-файл с отчетом, что подтверждает успешное скачивание (рисунок 42).

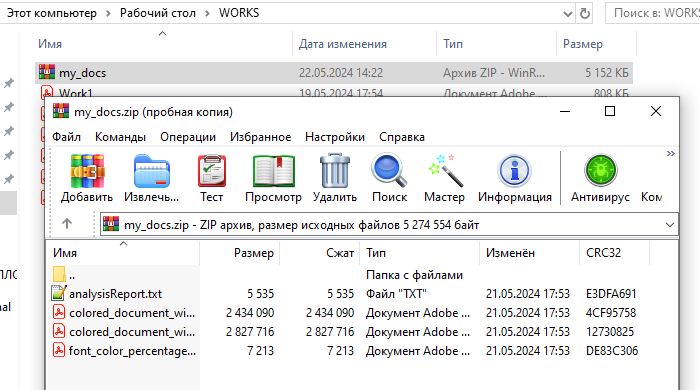


Рисунок 42. Успешное скачивание архива отчетов

6. Возможность экспортирования отчета

Пользователь может экспортировать отчеты, связанные с его аккаунтом, непосредственно на свою электронную почту.

Пользователь выбирает отчет в своем личном кабинете и нажимает на кнопку “Экспортировать” в окне отчета.

В появившемся диалоговом окне пользователь вводит адрес своей электронной почты, на который будет отправлен отчет (рисунок 43).

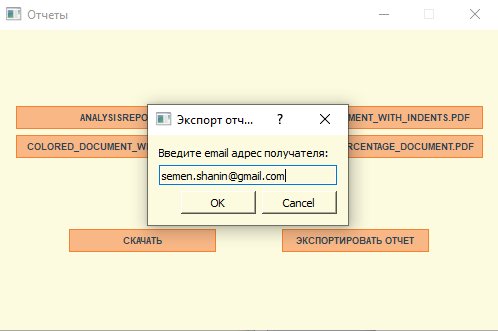


Рисунок 43. Ввод почты для отправки отчета

После подтверждения экспорта, отчет отправляется на указанную электронную почту. Пользователь может проверить свой почтовый ящик, чтобы убедиться, что отчет был успешно доставлен (рисунок 44).

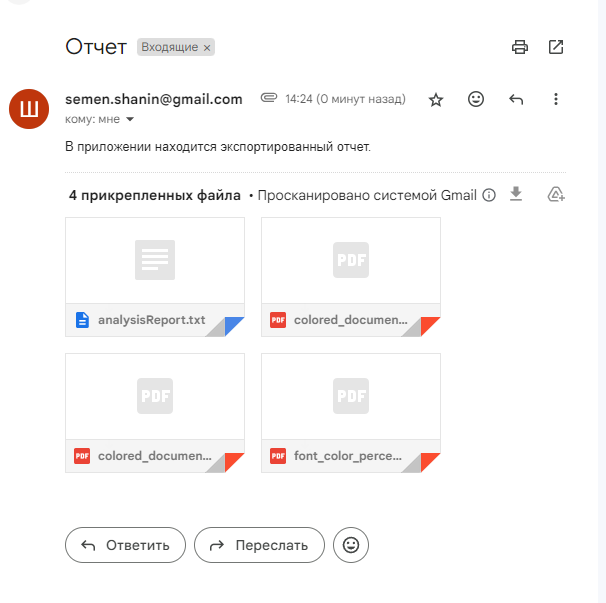


Рисунок 44. Входящее сообщение с отчетом на почте

7. Удаление пользователя

Администратор имеет доступ к панели управления, где отображается список всех пользователей, зарегистрированных в системе. Этот список синхронизирован с базой данных (БД), что позволяет администратору видеть актуальное состояние учетных записей.

В панели администратора (рисунок 45) отображается список пользователей с их основной информацией. Этот список соответствует записям в БД.



Рисунок 45. Старый список пользователей в административной панели

Список пользователей в панели администратора точно соответствует данным, хранящимся в БД (рисунок 46).

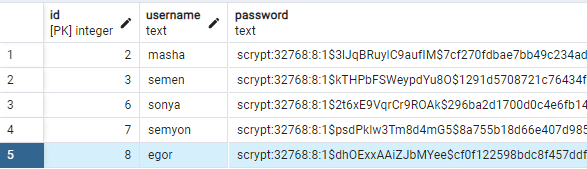


Рисунок 46. Старый список пользователей в базе данных

Администратор выбирает пользователя для удаления и нажимает кнопку “Удалить” (рисунок 47). Учетная запись пользователя удаляется из системы и соответственно из БД.

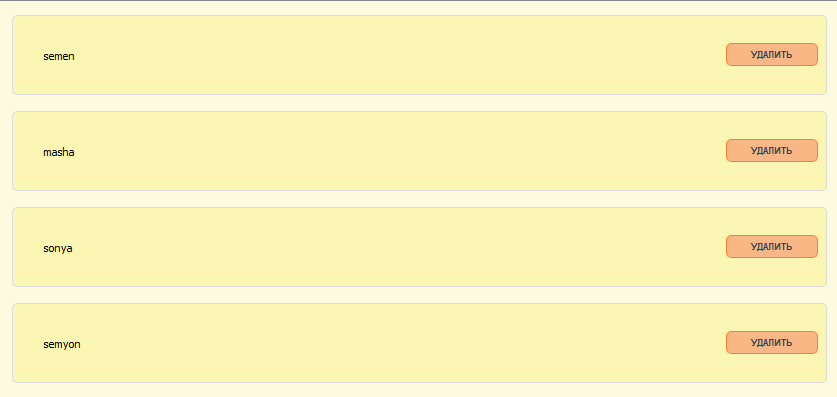


Рисунок 47. Обновленный список пользователей

После удаления пользователя его запись исчезает из списка в панели администратора и из БД, что подтверждает успешное выполнение операции (рисунок 48).

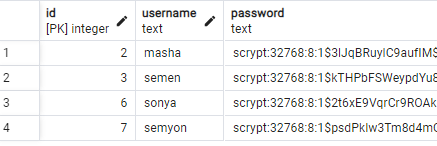


Рисунок 48. Обновленный список пользователей в БД

8. Повторная проверка документа

Пользователь имеет возможность провести повторную проверку документа прямо из своего личного кабинета (ЛК). В ЛК пользователь находит нужный документ и нажимает кнопку “Повторная проверка”. Это действие запускает процесс анализа документа (рисунок 49).

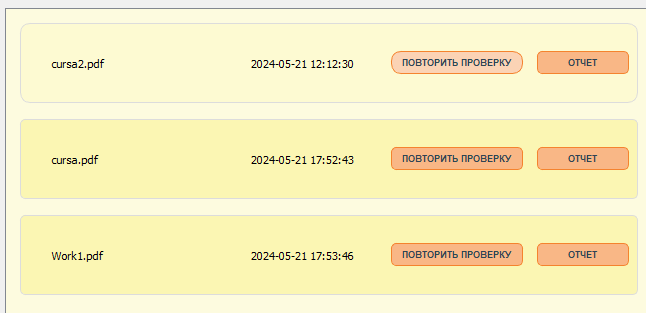


Рисунок 49. Повторная проверка отчета

После завершения проверки в консоли появляется сообщение о том, что отчет сохранен. Это подтверждает, что процесс проверки был выполнен и результаты обработаны.

При тестировании данной функции возникла ошибка, которая заключается в том, что файлы обновлялись в неправильной директории (рисунок 50). При проверке документа **cursa2.pdf** документация должна была сохраниться по адресу **UserDocs\semyon\cursa2**.

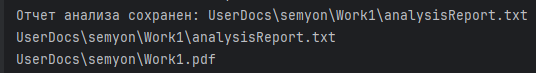


Рисунок 50. Ошибочный адрес формирования отчета

Ошибка возникла вследствие того, что в лямбда-функции использовалась функция из внешнего контекста. В данном случаем в функцию повторной проверки **recheckDocument()** передается параметр названия документа. В данном случае лямбда-функция брала значение имени документа с последней отображенной карточки документа в интерфейсе пользователя (рисунок 51).



Рисунок 51. Исправление функции, вызывающей ошибку

Для решения этой проблемы был изменен код вызова лямбда-функции таким образом, что значение передавалось на момент создания соответствующей кнопки в виде парамтра **doc = document[‘document\_name’]** (рисунок 52).



Рисунок 52. Обновленный вид лямбда-функции

В результате информация в консоли при проверке соответствующего документа стала отображаться верно (рисунок 53).

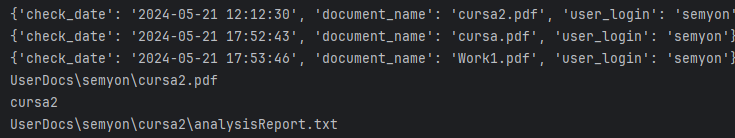


Рисунок 53. Правильный адрес загрузки отчета

В результате повторной проверки в файлах происходят изменения: цвет документа, разметка отступов и шрифты могут быть обновлены. Это демонстрирует, что документ был перепроверен и обновлен согласно новым данным (рисунки 54 и 55).



Рисунок 54. Старый набор цветов отчета по шрифтам



Рисунок 55. Новый набор цветов отчета по шрифтам

9. Удаление отзыва

Администратор может удалять отзывы пользователей через административную панель. Этот процесс обеспечивает управление контентом на платформе и поддержание ее актуальности. В административной панели администратор находит отзыв, который необходимо удалить (рисунок 56). В данном случае рассматривается удаление отзыва с id = 8.

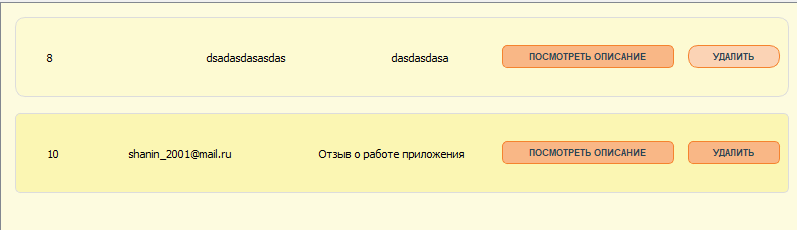


Рисунок 56. Удаление отзыва администратором

После выбора отзыва администратор нажимает кнопку “Удалить”. Страница обновляется, и отзыв исчезает из списка (рисунки 57 и 58).

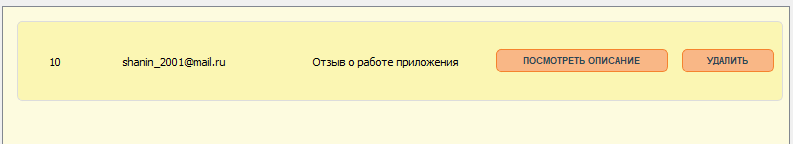


Рисунок 57. Обновленный интерфейс после удаления отзыва

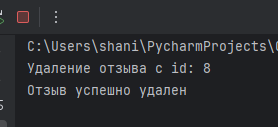


Рисунок 58. Информация в консоли об удалении отзыва

Подтверждение удаления из БД В базе данных также происходит удаление записи об отзыве, что гарантирует, что отзыв был полностью удален из системы (рисунок 59).

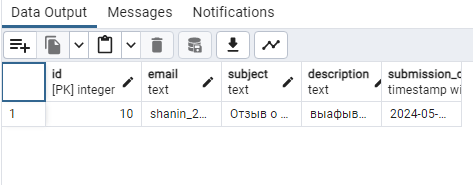


Рисунок 59. База данных после удаления отзыва

10. Проверка создания директории и файла

В данном пункте рассматривается авторизация пользователя у которого еще нет проверенных документов и соответственно отсутствует директория в папке с отчетами (рисунки 60 и 61).

Пользователь может инициировать процесс проверки документа, нажав на кнопку “Проверить” в интерфейсе приложения.

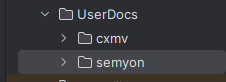


Рисунок 60. Директория с отчетами пользователей

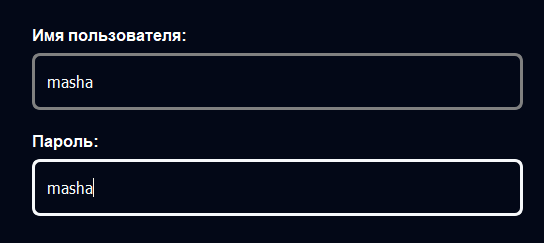


Рисунок 61. Авторизация пользователя, не имеющего отчетов

При нажатии на кнопку “Проверить” система начинает обработку документа (рисунок 62).

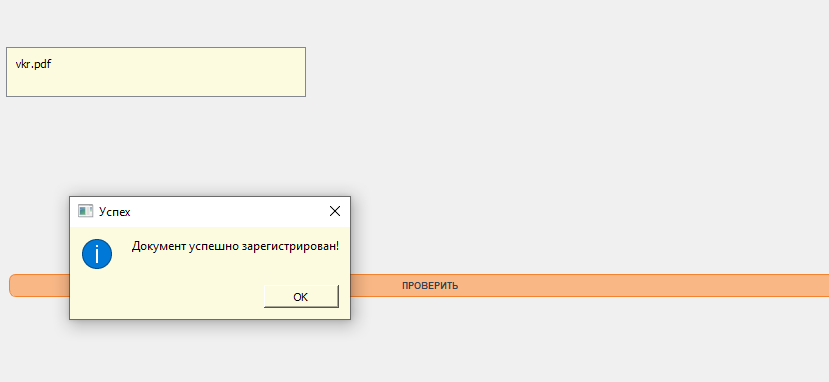


Рисунок 62. Запуск проверки документа

По завершении проверки в системе создается директория с файлом, который содержит результаты проверки. Пользователь может проверить наличие новой директории и файла, чтобы убедиться в успешном завершении процесса (рисунки 63 и 64).

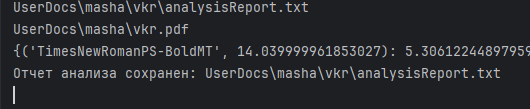


Рисунок 63. Информация о завершении проверки в консоли

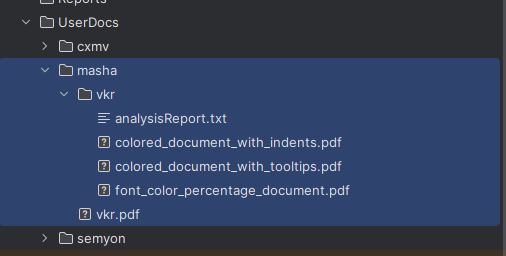


Рисунок 64. Обновленная директория с добавленными отчетами

В процессе тестирования были систематически проверены ключевые функции приложения. В ходе тестирования, на этапе 8, была обнаружена ошибка, которая была своевременно исправлена. Это подчеркивает важность тестирования не только для подтверждения работоспособности ключевых функций приложения, но и для обеспечения их оптимальной производительности и удобства использования.

# Заключение

За время прохождения производственной технологической (проектно-технологической) практики мной приобретены следующие компетенции:

| **Шифр компетенции** | **Расшифровка приобретаемой компетенции** |
| --- | --- |
| ОПК-4 | Способен участвовать в разработке стандартов, норм и правил, а также технической документации, связанной с профессиональной деятельностью |
| ОПК-7 | Способен разрабатывать алгоритмы и программы, пригодные для практического применения |
| ОПК-8 | Способен принимать участие в управлении проектами создания информационных систем на стадиях жизненного цикла |

# Список использованных источников

1. PyQT Tutorial [Электронный ресурс]. URL: https://www.pythontutorial.net/pyqt/ (дата обращения: 12.04.2024).
2. SQLAlchemy – Useful Resources[Электронный ресурс]. URL: https://www.tutorialspoint.com/sqlalchemy/sqlalchemy\_useful\_resources.htm (дата обращения 15.04.2024).
3. Quickstart – Flask Documentation [Электронный ресурс]. URL: https://ghostscript.com/~julian/docs-PyMuPDF/resources.html (дата обращения: 17.04.2020).
4. Resources – PyMuPDF 1.24.4 documentation [Электронный ресурс]. URL: https://ghostscript.com/~julian/docs-PyMuPDF/resources.html (дата обращения 17.04.2024).
5. Python for you and me[Электронный ресурс]. URL: https://pymbook.readthedocs.io/en/latest/ (дата обращения: 19.04.2024).

# ПРИЛОЖЕНИЕ

def check\_text\_centering\_2(pdf\_path, report\_file):

# Открыть PDF-документ

doc = fitz.open(pdf\_path)

# Получить первую страницу

page = doc[0]

# Получить информацию о тексте в формате словаря

blocks = page.get\_text("dict")["blocks"]

# Установить порог для определения центрирования

tolerance = 50

report\_file.write(f"1. Результаты проверки центрирования текста: \n")

# Проверка центрирования текста

for block in blocks:

if block["type"] == 0: # Тип 0 означает блок текста

x0, y0, x1, y1 = block["bbox"]

text\_center = (x0 + x1) / 2

page\_center = page.rect.width / 2

# Извлекаем текст из блока

text\_content = " ".join([line["spans"][0]["text"] for line in block["lines"]]).strip()

if abs(text\_center - page\_center) < tolerance:

report\_file.write(f" Отцентрован: {text\_content} \n")

else:

report\_file.write(f" Не отцентрован: {text\_content} \n")

def find\_sources\_in\_pdf\_3(pdf\_path, report\_file):

pdf = fitz.open(pdf\_path)

report\_file.write(f"\n")

report\_file.write(f"2. Анализ оформления Электронных ресурсов из списка литературы: \n")

# Читаем текст со всех страниц и ищем заголовок

for page\_number in range(len(pdf)):

page = pdf[page\_number]

text = page.get\_text("text")

# Разделяем текст на строки

lines = text.split('\n')

# Перебираем все строки на странице

for line in lines:

# Проверяем, начинается ли строка с заголовка списка источников

if (line.startswith('Список использованных источников') or line.startswith(

'Список использованной литературы') or

line.startswith('Список литературы') or line.upper().startswith('Список использованных источников') or

line.upper().startswith('Список использованной литературы') or line.upper().startswith(

'Список литературы')) and lines.index(line) < 3:

found\_title = line # Сохраняем найденный заголовок

report\_file.write(f" Заголовок '{found\_title}' найден на странице {page\_number + 1} \n")

# Создаем паттерн для поиска источников после заголовка

source\_pattern = re.compile(

r'\d+\.\s+.+\[Электронный ресурс\]\.\s+URL:\s+.+\s+\(дата обращения:\s+.+\)\.',

re.DOTALL

)

# Объединяем строки обратно в текст, начиная с найденного заголовка

text\_from\_header = '\n'.join(lines[lines.index(line):])

# Ищем все совпадения с паттерном в тексте после заголовка

matches = source\_pattern.findall(text\_from\_header)

for match in matches:

report\_file.write(match)

break

report\_file.write(f"\n\nЕсли система не распознала какой-либо из электронных ресурсов, проверьте оформление: \n")

report\_file.write(f"Пример оформления: \n")

# Закрываем PDF-документ

pdf.close()

def font\_and\_size\_color\_4(pdf\_path, new\_directory\_path, report\_file):

def generate\_unique\_colors(n):

colors = set()

while len(colors) < n:

colors.add((random.random(), random.random(), random.random()))

return list(colors)

report\_file.write(f"\n\n")

report\_file.write(f"3. Определение стиля и размера шрифтов, используемых в документе: \n")

report\_file.write(f" Результат анализа в файлах colored\_document\_with\_tooltips.pdf и font\_color\_percentage\_document.pdf \n")

# Открытие документа

doc = fitz.open(pdf\_path)

print(pdf\_path)

# Словарь для хранения уникальных комбинаций шрифта и размера и их количества

unique\_combinations = {}

# Проход по страницам документа для сбора комбинаций и подсчета их количества

for page\_num in range(len(doc)):

page = doc[page\_num]

blocks = page.get\_text("dict")["blocks"]

for b in blocks:

if b['type'] == 0:

for line in b["lines"]:

for span in line["spans"]:

combo = (span['font'], span['size'])

unique\_combinations[combo] = unique\_combinations.get(combo, 0) + 1

# Генерация уникальных цветов

colors = generate\_unique\_colors(len(unique\_combinations))

# Словарь для соответствия комбинаций и цветов

font\_size\_color = dict(zip(unique\_combinations.keys(), colors))

# Проход по страницам документа для раскраски текста и добавления подсказок

for page\_num in range(len(doc)):

page = doc[page\_num]

blocks = page.get\_text("dict")["blocks"]

for b in blocks:

if b['type'] == 0:

for line in b["lines"]:

for span in line["spans"]:

combo = (span['font'], span['size'])

if combo in font\_size\_color:

color = font\_size\_color[combo]

# Создание аннотации с подсказкой

annot = page.add\_highlight\_annot(fitz.Rect(span['bbox']))

annot.set\_colors(stroke=color)

annot.set\_info(title="Шрифт и размер",

content=f"Шрифт: {span['font']}, Размер: {span['size']}")

annot.update()

# Сохранение изменений в новый файл

new\_file\_path = os.path.join(new\_directory\_path, 'colored\_document\_with\_tooltips.pdf')

doc.save(new\_file\_path)

doc.close()

# Расчет общего количества использований шрифтов

total\_usage = sum(unique\_combinations.values())

# Расчет процентного соотношения для каждой комбинации

percentage\_combinations = {combo: (count / total\_usage \* 100) for combo, count in unique\_combinations.items()}

# Вывод процентного соотношения для каждой комбинации

print(percentage\_combinations)

new\_doc = fitz.open()

new\_page = new\_doc.new\_page()

# Отображение информации на странице

text\_y\_position = 100

for combo, color in font\_size\_color.items():

# Процентное соотношение для комбинации

percentage = percentage\_combinations[combo]

# Текст для отображения

text = f"{combo[0]}, {combo[1]} - {percentage:.2f}%"

# Рисование цветного прямоугольника

rect = fitz.Rect(50, text\_y\_position, 100, text\_y\_position + 20)

new\_page.draw\_rect(rect, color=color, fill=color)

# Добавление текста рядом с прямоугольником

new\_page.insert\_text((110, text\_y\_position + 15), text, fontsize=12)

text\_y\_position += 50

# Сохранение нового документа

another\_new\_file\_path = os.path.join(new\_directory\_path, 'font\_color\_percentage\_document.pdf')

new\_doc.save(another\_new\_file\_path)

new\_doc.close()

def indent\_5(pdf\_path, new\_directory\_path, report\_file):

def generate\_unique\_colors(n):

colors = set()

while len(colors) < n:

colors.add((random.random(), random.random(), random.random()))

return list(colors)

# Открытие документа

report\_file.write(f"\n\n")

report\_file.write(f"4. Определение отступов, используемых в документе: \n")

report\_file.write(f" Результат анализа в файле colored\_document\_with\_indents.pdf \n")

doc = fitz.open(pdf\_path)

# Словарь для хранения уникальных отступов и их количества

unique\_indents = {}

# Проход по страницам документа для сбора отступов и подсчета их количества

for page\_num in range(2, len(doc)): # Индексация страниц начинается с 0

page = doc.load\_page(page\_num)

blocks = page.get\_text("dict")["blocks"]

for block in blocks:

if block["type"] == 0: # Тип 0 означает блок текста

left\_indent = block["bbox"][

0] \* 0.035278 # Переводим отступ в сантиметры

unique\_indents[left\_indent] = unique\_indents.get(left\_indent, 0) + 1

# Генерация уникальных цветов

colors = generate\_unique\_colors(len(unique\_indents))

# Словарь для соответствия отступов и цветов

indent\_color = dict(zip(unique\_indents.keys(), colors))

# Определение общего цвета для отступов больше 140 пунктов и для отступов 90 и 115

common\_color = (0.5, 0.5, 0.5) # Например, серый цвет

# Проход по отступам и присвоение общего цвета

for left\_indent in unique\_indents:

if left\_indent > 4.255 or (3.005 < left\_indent < 4.245):

indent\_color[left\_indent] = common\_color

# Проход по страницам документа для раскраски текста

for page\_num in range(2, len(doc)):

page = doc.load\_page(page\_num)

blocks = page.get\_text("dict")["blocks"]

for block in blocks:

if block["type"] == 0: # Тип 0 означает блок текста

left\_indent = block["bbox"][0] \* 0.035278 # Переводим отступ в сантиметры

if left\_indent in indent\_color:

color = indent\_color[left\_indent]

# Создание аннотации с подсказкой

for line in block["lines"]:

for span in line["spans"]:

annot = page.add\_highlight\_annot(fitz.Rect(span['bbox']))

annot.set\_colors(stroke=color)

annot.set\_info(title="Отступ слева", content=f"Отступ: {left\_indent:.2f} см")

annot.update()

# Сохранение изменений в новый файл

new\_file\_path = os.path.join(new\_directory\_path, 'colored\_document\_with\_indents.pdf')

doc.save(new\_file\_path)

doc.close()

def numeration\_7(pdf\_path, report\_file):

doc = fitz.open(pdf\_path)

incorrectly\_numbered\_pages = []

report\_file.write(f"\n5. Анализ нумерации страниц: \n")

for page\_num in range(len(doc)):

page = doc.load\_page(page\_num)

text\_instances = page.search\_for(str(page\_num + 1))

# Получаем размер страницы

page\_height = page.rect.height

# Определяем нижнюю границу для поиска номера страницы (например, последние 100 пикселей страницы)

lower\_bound = page\_height - 100

# Проверяем, находится ли номер страницы в нижней части страницы

if any([instance[1] > lower\_bound for instance in text\_instances]):

report\_file.write(f" Страница {page\_num + 1} корректно пронумерована в нижней части. \n")

else:

report\_file.write(

f" Страница {page\_num + 1} не содержит номер в нижней части, неверно пронумерована или номер указан буквенными символами. \n")

incorrectly\_numbered\_pages.append(page\_num + 1)

doc.close()

if incorrectly\_numbered\_pages:

report\_file.write("Рекомендуется проверить нумерацию на следующих страницах: \n")

for page\_num in incorrectly\_numbered\_pages:

report\_file.write(f" Страница {page\_num} \n")

else:

report\_file.write("Нумерация на всех страницах корректна. \n")

def heading\_finder\_8(pdf\_path, report\_file):

def extract\_sections(contents\_text):

pattern = re.compile(r'(\d+\.(?:\d+\.)?\s+.\*?)(\.{2,}\s+|\n\s\*\.{2,}\s\*)(\d+)')

matches = pattern.findall(contents\_text)

sections = {match[0].strip(): int(match[2]) for match in matches}

return sections

# Функция для проверки соответствия разделов и номеров страниц

def check\_sections(doc, sections):

for title, page\_number in sections.items():

# Получаем текст страницы

page\_text = doc[page\_number - 1].get\_text("text")

# Удаляем символы переноса строки для сравнения

page\_text = ' '.join(page\_text.split())

title = ' '.join(title.split())

# Создаем паттерн для поиска заголовка с учетом возможного переноса строки

title\_pattern = re.compile(re.escape(title), re.DOTALL)

if re.search(title\_pattern, page\_text):

report\_file.write(f'Заголовок "{title}" найден на странице {page\_number}. \n')

else:

report\_file.write(f'Заголовок "{title}" не найден на странице {page\_number}. \n')

report\_file.write(

'Если раздел находится в содержании, но не был распознан системой, рекомендуем проверить оформление данного раздела и особое внимание обратить на наличие точек после номера пункта \n')

doc = fitz.open(pdf\_path)

report\_file.write(f'\n\n6. Проверка соответствия содержания и заголовка на страницах: \n')

contents\_page\_text = doc[1].get\_text("text")

sections = extract\_sections(contents\_page\_text)

check\_sections(doc, sections)

doc.close()

def drawing\_and\_mention\_9(pdf\_path, report\_file):

doc = fitz.open(pdf\_path)

# Словарь для хранения упоминаний рисунков

figure\_mentions = {}

# Регулярное выражение для поиска упоминания рисунка

figure\_mention\_pattern = re.compile(r"рисунок\s+(\d+)")

report\_file.write(f"\n\n7. Проверка наличия упоминания рисунка и его корректного описания: \n" )

# Перебираем страницы документа и собираем упоминания рисунков

for page\_num in range(len(doc)):

page = doc.load\_page(page\_num)

text\_blocks = page.get\_text("blocks")

# Ищем упоминания рисунков в тексте

for block in text\_blocks:

match = figure\_mention\_pattern.search(block[4])

if match:

figure\_number = match.group(1)

# Добавляем номер страницы в словарь упоминаний

if figure\_number not in figure\_mentions:

figure\_mentions[figure\_number] = []

figure\_mentions[figure\_number].append(page\_num + 1)

# Регулярное выражение для поиска подписи к рисунку

figure\_caption\_pattern = re.compile(r"Рисунок\s+(\d+)\.\s+(.\*)")

# Перебираем страницы документа и ищем подписи к рисункам

for page\_num in range(len(doc)):

page = doc.load\_page(page\_num) # Загрузите страницу

text\_blocks = page.get\_text("blocks") # Получите блоки текста со страницы

# Ищем подписи к рисункам

for block in text\_blocks:

caption\_match = figure\_caption\_pattern.search(block[4])

if caption\_match:

figure\_number = caption\_match.group(1)

caption\_text = caption\_match.group(2)

# Проверяем, есть ли упоминание этого рисунка

if figure\_number in figure\_mentions:

report\_file.write(f" Рисунок {figure\_number} с подписью '{caption\_text}' найден на странице {page\_num + 1} \n")

# Удаляем номер рисунка из словаря упоминаний, так как подпись найдена

del figure\_mentions[figure\_number]

# Проверяем, остались ли рисунки без подписей

if figure\_mentions:

for figure\_number, pages in figure\_mentions.items():

report\_file.write(f" Рисунок {figure\_number} упоминается на страницах {pages}, но подпись не найдена. \n")

doc.close()

def cosine\_similarity\_10(pdf\_path, report\_file):

def calculate\_cosine\_similarity(text1, texts):

vectorizer = TfidfVectorizer()

tfidf = vectorizer.fit\_transform([text1] + texts)

cosine\_matrix = cosine\_similarity(tfidf[0:1], tfidf[1:])

return cosine\_matrix[0]

# Функция для чтения текста из PDF-файла

def read\_pdf\_file(file\_path):

with fitz.open(file\_path) as document:

text = ""

for page in document:

text += page.get\_text()

return text

# Путь к папке с работами

folder\_path = 'C:/Users/shani/Desktop/WORKS'

my\_document\_text = read\_pdf\_file(pdf\_path)

report\_file.write(f"\n\n8. Анализ заимстований с помощью алгоритма косинусного сходства: \n")

# Список для хранения текстов из папки

texts\_to\_compare = []

# Чтение всех PDF-файлов в папке и добавление их текстов в список

for filename in os.listdir(folder\_path):

if filename.endswith('.pdf'): # теперь мы работаем с PDF

file\_path = os.path.join(folder\_path, filename)

texts\_to\_compare.append(read\_pdf\_file(file\_path))

# Расчет косинусного сходства

similarities = calculate\_cosine\_similarity(my\_document\_text, texts\_to\_compare)

# Вывод результатов

for filename, similarity in zip(os.listdir(folder\_path), similarities):

report\_file.write(f" Сходство с {filename}: {similarity} \n")

max\_similarity\_index = np.argmax(similarities)

most\_similar\_document = os.listdir(folder\_path)[max\_similarity\_index]

if similarities[max\_similarity\_index] < 0.2:

report\_file.write(

f"Работа оригинальна. Максимальное сходство {similarities[max\_similarity\_index]} с документом {most\_similar\_document} \n")

else:

report\_file.write(

f"Работа не оригинальна. Максимальное сходство {similarities[max\_similarity\_index]} с документом {most\_similar\_document} \n")