Министерство образования Республики Беларусь

Учреждение образования
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет Информационных технологий и управления
Кафедра Интеллектуальных информационных технологий

К защите допустить:
Заведующий кафедрой
_____ Д. В. Шункевич

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовому проекту
по дисциплине «Математические основы интеллектуальных систем»:

Генератор тестов и контрольных по обучающим материалам

БГУИР КП 6-05-0611-03-032

Студент: И. А. Кислицын
Группа: 321702

Руководитель: М. В. Ковалев

Минск 2025

СОДЕРЖАНИЕ

Перечень условных обозначений 5
Введение 6
1 Раздел 8
1.1 Постановка задачи 8
1.2 Компонентный анализ системы генерации тестов 9
1.3 Анализ подходов к решению проблемы 10
1.4 Анализ архитектуры системы 11
2 Раздел 13
2.1 Общая архитектура системы 13
2.2 Ключевые компоненты системы 13
2.3 Взаимодействие на нижнем уровне 14
2.4 Анализ пользователей системы 14
2.5 Модуль загрузки и индексации документов 15
2.6 Модуль генерации тестов 15
2.7 Модуль управления контентом 16
2.8 Дополнительные аспекты проектирования 17
2.9 Векторы развития 18
2.10 LangChain 18
2.11 Pydantic Модели 19
2.12 FastAPI 20
2.13 База данных SQLlite 21
2.14 ChromaDB 21
3 Реализация и интеграция компонентов системы 23
3.1 Компоненты системы и их реализация 23
3.2 Ключевые компоненты кода 26
Заключение 29
Список использованных источников 31

ПЕРЕЧЕНЬ УСЛОВНЫХ ОБОЗНАЧЕНИЙ

КП — Курсовой проект;

LLM — Large Language Model;

БД — База данных;

ChromaDB — Chroma Database, база данных Chroma;

LLaMA — Large Language Model Meta AI, большая языковая модель Meta AI;

PDF — Portable Document Format, портативный формат документа;

RAG — Retrieval-Augmented Generation, генерация с использованием извлечения данных;

API — Application Programming Interface, интерфейс программирования приложений

DOCX — Document Open XML, формат текстовых документов Microsoft Word;

SQL — Structured Query Language, язык структурированных запросов для работы с базами данных;

СУБД — Система управления базами данных, программное обеспечение для создания, хранения и управления базами данных.

ВВЕДЕНИЕ

Система проверки качества знаний существует с III–I тыс. до н. э.: уже в древнем Китае (II тыс. до н. э.) проводились первые экзамены для отбора чиновников, а при династии Хань (206 до н. э. – 220 н. э.) была формализована система кэцзюй – государственные экзамены. В Месопотамии и Египте проверялись навыки писцов через решение математических задач и написание текстов.

С развитием образования методы оценки знаний эволюционировали: от устных диспутов в античных школах до стандартизированных тестов XIX–XX вв. (например, IQ-тест, ЦТ, ЕГЭ). Однако в современном мире, с ростом цифровизации и дистанционного обучения, возникает потребность в автоматизированных инструментах для быстрого и объективного контроля знаний.

Генератор тестов и контрольных работ на основе обучающих материалов обеспечивает более ускоренное создание документов для контроля знаний.

Цель КП — создание системы для преподавателей, которая автоматически генерирует тестовые задания и контрольные работы на основе загруженных учебных материалов, обеспечивая гибкость настройки и разнообразие форматов заданий.

Задачами КП являются:

– изучить предметную область тестирования учащихся, существующие подходы к созданию и обработке данных, а также принципы работы технологии RAG;

– спроектировать архитектуру интеллектуального ассистента для автономной генерации тестов;

– разработать прототип ассистента для преподавателей, который позволит создавать тесты на основе загруженных материалов;

– проанализировать гибкость настроек и разнообразие системы, учитывая дальнейшее развитие ассистента.

Минимальный ожидаемый функционал от системы:

1) Возможность загрузки текстовых материалов (учебники, лекции, конспекты) для анализа.Пользователь через интуитивно понятный интерфейс будет иметь возможность загрузить необходимый материал, на основе которого после выполнения анализа будет создан уникальный тест;

2) Генерация тестов с несколькими типами заданий (например, вопросы с выбором ответа, открытые вопросы, задания на соответствие). Пользователь имеет возможность задать количество вопросов с выбором ответа, открытых вопросов и заданий на соответствие в генерируемом тесте;

3) Настройка уровня сложности тестов и контрольных работ. Пользователь имеет возможность выбрать уровень сложности генерируемого теста (1–3 уровень);

4) Экспорт готовых тестов в удобных форматах (PDF, DOCX). Пользователь имеет возможность загрузить сгенерированный тест в различных форматах (PDF или Word).

1 РАЗДЕЛ

1.1 Постановка задачи

Исторически оценка компетенций являлась ключевым элементом образовательных систем. От первых экзаменов для чиновников в эпоху династии Чжоу до цифровых платформ XXI века — методы верификации знаний непрерывно трансформируются. Современный этап характеризуется необходимостью оперативного создания персонализированных оценочных материалов, соответствующих требованиям цифровой педагогики и гибридного обучения.

Основная цель проекта — разработка интеллектуального инструмента для автоматизированного формирования оценочных материалов с использованием методов обработки естественного языка (NLP) и адаптивных алгоритмов. Система должна обеспечивать генерацию вариативных заданий, соответствующих содержанию учебных ресурсов и педагогическим целям преподавателя.

Ключевые аспекты разработки:

– Интеграция технологий анализа текстовых данных для выявления семантически значимых элементов учебного контента

– Реализация механизмов динамической подстройки сложности заданий

Целевые характеристики системы:

1) *Многоформатный импорт учебных ресурсов*. Реализация: Поддержка загрузки структурированных данных и неструктурированных текстов (PDF, DOCX) с автоматическим распознаванием семантических блоков;

2) *Контекстно-зависимая генерация заданий*. Реализация: Создание комбинированных тестов с использованием технологий дистракторного анализа (для вопросов множественного выбора) и синтаксического шаблонирования (для открытых заданий);

3) *Адаптивная система сложности*. Реализация: Трехуровневая градация заданий с критериями оценки:

- Легкий: воспроизведение информации
- Средний: применение знаний в новых условиях
- Сложный: синтез междисциплинарных решений;

4) *Полиформатная публикация результатов*. Реализация: Экспорт с поддержкой md формата, pdf формата.

1.2 Компонентный анализ системы генерации тестов

Для разработки системы генерации тестов на основе загруженных документов, интегрированной с порталом и обладающей удобным интерфейсом, необходимо определить ключевые компоненты, которые обеспечат выполнение требований: точность, актуальность, гибкость, скорость, прозрачность, масштабируемость и интеграцию. Ниже представлен анализ компонентов системы, их функций, а также подходов к их взаимодействию и реализации, основанный на предоставленных файлах проекта.[1]

Модуль загрузки и обработки документов: обеспечивает загрузку документов пользователями (PDF, DOCX) и их обработку для последующего использования в генерации тестов. Выполняет такие основные функции как: прием файлов через интерфейс или API, извлечение текста из документов разных форматов, сохранение файлов и их метаданных в базе данных;[2]. *Пример*: Пользователь загружает PDF с лекцией, система извлекает текст и сохраняет его для создания тестов.

Модуль генерации тестов: создает тесты на основе содержимого загруженных документов с учетом параметров, заданных пользователем (количество вопросов, сложность, формат). Выполняет такие основные функции как: анализ текста документа для выделения ключевых идей, генерация вопросов и ответов с использованием языковой модели, форматирование теста с включением правильных ответов. *Пример*: система создает 5 вопросов средней сложности с выбором ответа на основе текста о программировании;

Модуль интеграции с внешними системами: обеспечивает связь системы с внешними системами или сервером для обмена данными и интеграции с существующей инфраструктурой. Выполняет такие основные функции как: отправка и получение данных через API, синхронизация загруженных документов и тестов, поддержка сессий для персонализированного взаимодействия. *Пример*: система запрашивает список документов с сервера и использует их для генерации тестов;

Модуль управления сессиями и историей: отслеживает сессии пользователей и сохраняет историю взаимодействий для обеспечения контекстной релевантности. Выполняет такие основные функции как: генерация уникальных идентификаторов сессий, хранение запросов и ответов в базе данных. *Пример*: пользователь возвращается к системе и видит свои прошлые тесты в профиле;

Интерфейс пользователя: предоставляет удобный интерфейс для загрузки документов, настройки тестов и управления результатами. Выполняет такие основные функции как: просмотр, скачивание и удаление сгенерированных тестов. *Пример*: пользователь загружает файл, задает 10 вопросов и скачивает тест в PDF;

Модуль сохранения и управления PDF-тестами: конвертирует

тесты в PDF, сохраняет их и предоставляет возможности управления. Выполняет такие основные функции как: преобразование текста теста в PDF-формат, хранение PDF-файлов в базе данных. *Пример*: пользователь скачивает тест в PDF и удаляет его из системы после использования.

1.3 Анализ подходов к решению проблемы

Необходимо выбрать подход, который обеспечит реализацию поставленных целей: точность ответов, скорость работы, контекстную релевантность, гибкость, интеграцию, простоту использования и масштабируемость [3]. Рассмотрим три возможных подхода, проанализируем их особенности и сделаем выводы по каждому, чтобы затем определить оптимальный вариант.

Подход 1: Использование технологии RAG в чистом виде: Этот метод предполагает два этапа: сначала данные портала индексируются с помощью поискового индекса, например FAISS, для быстрого извлечения релевантных фрагментов, а затем эти данные передаются в генеративную модель, такую как LLaMA, для формирования ответа;

Преимущества: модель работает только с наиболее подходящими данными, что обеспечивает высокую точность ответов, соответствующих содержимому портала. Возможность выбора различных алгоритмов индексации и языковых моделей позволяет адаптировать систему под разные порталы и задачи. Ответы всегда основаны на текущих данных портала; [4]

Недостатки: обработка больших объемов данных и настройка индекса требуют значительных вычислительных ресурсов. Без оптимизации процесс извлечения и генерации может быть медленным, что ухудшит пользовательский опыт;

Вывод: RAG предлагает отличное сочетание точности и контекстной релевантности, что делает его перспективным для интеллектуального ассистента. Однако для достижения скорости и масштабируемости потребуется оптимизация, например, использование легковесных моделей или эффективных индексов.

Подход 2: Гибридная система с традиционным поиском и генеративной моделью: в этом подходе используется традиционная поисковая система, например Elasticsearch, для быстрого поиска по порталу, а результаты передаются в языковую модель для генерации ответа;

Преимущества: проверенная производительность поисковых систем обеспечивает быстрое извлечение данных. Легко справляется с большими объемами данных благодаря зрелым технологиям. Существующие инструменты упрощают подключение к порталу;

Недостатки: традиционный поиск хуже извлекает сложные связи в данных, что снижает качество ответов по сравнению с RAG. Требуется

согласованная работа поиска и генеративной модели, что усложняет разработку.

Вывод: гибридный метод выигрывает в скорости и масштабируемости, но жертва точностью и гибкостью делает его менее подходящим для задачи, где приоритет отдается качеству ответов.

Подход 3: Дообучение языковой модели на данных портала: языковая модель, такая как BERT или GPT, обучается непосредственно на текстах портала, чтобы генерировать ответы без промежуточного этапа извлечения данных;

Преимущества: глубокое понимание специфики данных портала обеспечивает точные и релевантные ответы. После обучения модель работает напрямую, без дополнительных компонентов;

Недостатки: дообучение требует значительных ресурсов и подготовки качественного корпуса данных. При обновлении портала модель теряет актуальность и нуждается в переобучении;

Вывод: несмотря на высокую точность, этот метод слишком дорог и негибок для динамически обновляемых данных портала, что делает его непрактичным.

1.4 Анализ архитектуры системы

1) **Фронтенд**: Реализован на Streamlit для обеспечения интерактивного интерфейса [5]. Streamlit - библиотека Python с открытым кодом. Она позволяет с легкостью создавать разные красивые и интуитивно понятные веб-приложения для инженеров машинного обучения[6]. Выбрано на основе простоты реализации и модификации в дальнейшем развитии системы;

2) **Бэкенд**: Построен на FastAPI[7] для обработки запросов и управления логикой. FastAPI — это современный, быстрый и простой веб-фреймворк для Python, предназначенный для создания API. Выбрано на основе типизации с помощью Pydantic, что обеспечивает проверку типов данных и их валидацию, это в дальнейшем упростит разработку системы. Также поддерживает async/await, что делает его эффективным для задач с высокой нагрузкой и важным в разработке системы связанной с генерацией документов;

3) **База данных**: SQLite для хранения логов и информации о документах. SQLite хранится в одном файле, что упрощает установку, перенос и управление. Подходит для локального хранения данных. В дальнейшем развитии системы обеспечит легкую интеграцию с сервером;

4) **Векторное хранилище**: Chroma для индексации и поиска по содержимому документов. Chroma — база данных, разработанная для удоб-

ного хранения и быстрого поиска информации на основе ее смысла, а не ключевых слов, что позволит системе более точно отвечать на запросы. Для поиска по содержимому документа и хранения материала такая база данных обеспечит ускоренное развитие системы в дальнейшей разработке;

5) **Языковая модель**: Используется через LangChain для генерации тестов. LangChain обеспечит интеграцию языковой модели (LLM) с векторной базой данных (Chroma) [8], легкую интеграцию LLM с API (FastAPI) [9].

2 РАЗДЕЛ

2.1 Общая архитектура системы

Система OneClickTest построена на модульной архитектуре, разделенной на клиентскую и серверную части, взаимодействующие через API. Архитектура спроектирована с учетом принципов масштабируемости, гибкости и удобства сопровождения. Основные компоненты системы включают пользовательский интерфейс, серверное приложение, расширенную векторную базу данных для семантического поиска, оптимизированную реляционную базу данных для хранения метаданных и модуль генерации тестов.

2.2 Ключевые компоненты системы

Ассистент автоматической генерации тестов имеет основные компоненты, так как необходимо предусмотреть и клиет-архитектурное построение системы, а также учесть все необходимые модули, которые будут необходимы в реализации ассистента. На рисунке 2.1 представлена диаграмма компонентов системы.

– **Клиентский интерфейс**: Веб-интерфейс, предоставляющий пользователю доступ к функциональности системы. Поддерживает загрузку документов, настройку параметров тестов, просмотр и экспорт результатов, а также управление загруженными материалами;

– **Серверное приложение**: Центральный компонент, отвечающий за обработку запросов, координацию работы подсистем, взаимодействие с базами данных и вызов AI-моделей;

– **Векторная база данных**: Хранит индексированные фрагменты документов для выполнения семантического поиска, необходимого для генерации тестов;

– **Реляционная база данных**: Сохраняет метаданные документов, историю взаимодействия пользователей (логи чатов) и информацию о сгенерированных тестах;

– **Модуль генерации тестов**: Использует RAG-подход (Retrieval-Augmented Generation) для извлечения актуальной информации из документов и создания тестов с учетом пользовательских параметров.

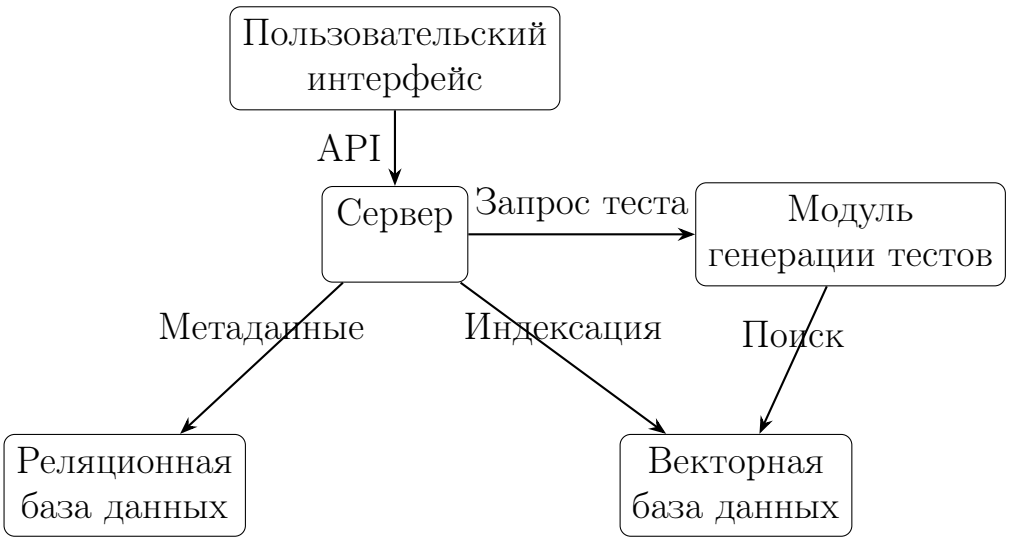

Рисунок *2.1* – Диаграмма компонентов системы

2.3 Взаимодействие на нижнем уровне

Серверное приложение состоит из следующих подсистем:

– **API-слой**: Обеспечивает маршрутизацию HTTP-запросов (загрузка файлов, генерация тестов, управление документами). Поддерживает RESTful API с четко определенными эндпоинтами;

– **Модуль индексации**: Обрабатывает загруженные документы, разбивает их на фрагменты, создает семантические эмбеддинги и сохраняет в векторную базу [2];

– **Модуль управления данными**: Отвечает за взаимодействие с реляционной базой данных, сохраняя метаданные документов, историю чатов и информацию о тестах;

– **Модуль RAG**: Выполняет контекстно-зависимый поиск в векторной базе и передает результаты в AI-модель для генерации тестов.

2.4 Анализ пользователей системы

Система OneClickTest ориентирована на различные категории пользователей. Блок схема работы модуля представлена на рисунке 2.2.

– **Преподаватели**: Основные пользователи, выступают в качестве администратора системы, загружающие учебные материалы (лекции, статьи, методички), генерирующие тесты и управляющие контентом. Требуют удобного интерфейса и точной генерации тестов;

– **Студенты (перспектива)**: В будущем, при дальнейшем развитии проекта, могут использовать систему для прохождения тестов, что потребует дополнительных функций интерфейса.

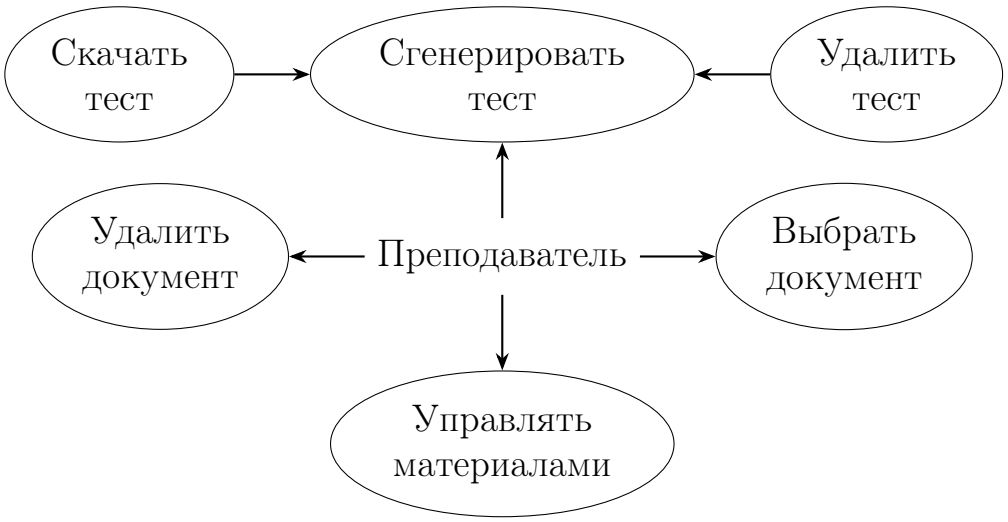

Рисунок *2.2* – Диаграмма вариантов использования

2.5 Модуль загрузки и индексации документов

– **Назначение**: Обеспечение загрузки документов и их подготовки для семантического поиска. Блок схема работы модуля представлена на рисунке 2.3.

– **Функциональность**: Поддержка форматов: PDF, DOCX. Разбиение текста на фрагменты (chunking) для оптимальной индексации. Создание векторов с использованием модели машинного обучения[6]. Сохранение фрагментов и их векторов в векторную базу [**?**]. Регистрация метаданных (имя файла, дата загрузки) в реляционной базе;

– **Принципы внутреннего устройства**: Модуль включает подмодули загрузки файлов, обработки текста, генерации векторов и интеграции с базами данных. Обеспечивает обработку ошибок и валидацию форматов.

2.6 Модуль генерации тестов

– **Назначение**: Создание тестов на основе загруженных документов с учетом пользовательских требований. Является ключевым модулем системы. Блок схема работы модуля представлена на рисунке 2.4;

– **Функциональность**: Настройка параметров: количество вопросов, уровень сложности, формат (множественный выбор, открытые вопросы). Выполнение семантического поиска для извлечения релевантных фрагментов. Генерация вопросов и ответов с использованием AI-модели. Форматирование результатов в Markdown и PDF;

– **Принципы внутреннего устройства**: Модуль реализует RAG-подход, комбинируя поиск в векторной базе с генеративной моделью. Обеспечивает поддержку нескольких языков и форматирование с учетом кириллицы.

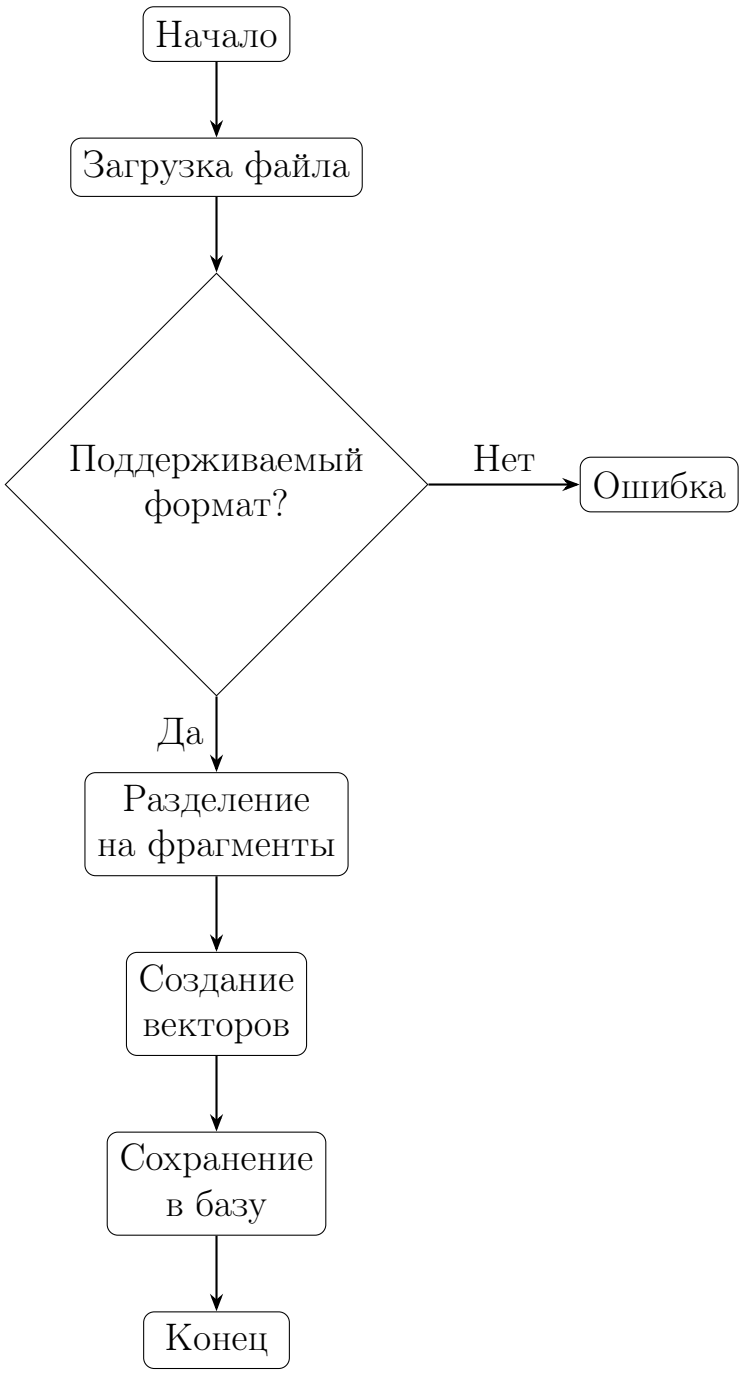

Рисунок *2.3* – Блок-схема процесса индексации документов

2.7 Модуль управления контентом

– **Назначение**: Управление загруженными документами и сгенерированными тестами;

– **Функциональность**: Просмотр списка документов и их метаданных. Удаление документов из векторной и реляционной баз. Сохранение и просмотр истории чатов и тестов. Поддержка сессий для многопользовательской работы;

– **Принципы внутреннего устройства**: Модуль интегрируется с реляционной базой для хранения метаданных и с векторной базой для

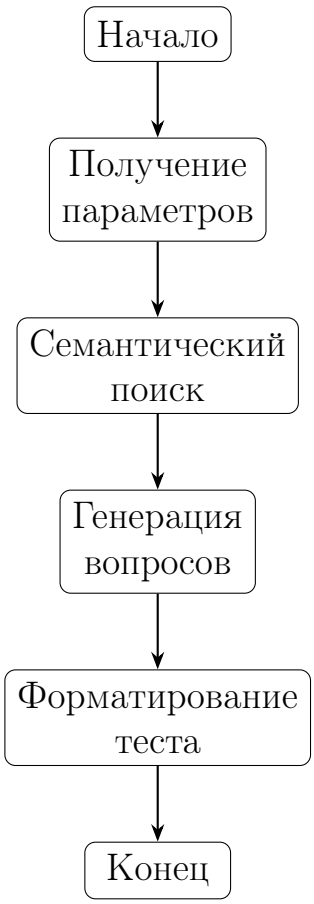

Рисунок *2.4* – Блок-схема процесса генерации теста

удаления индексов. Обеспечивает транзакционную целостность операций.

2.8 Дополнительные аспекты проектирования

2.8.1 Обработка ошибок

– Валидация форматов файлов при загрузке;
– Обработка сбоев при индексации (например, поврежденные файлы);
– Логирование ошибок для анализа администратором;
– Откат транзакций при сбоях в базе данных.

2.8.2 Безопасность

– Аутентификация пользователей (в перспективе);
– Ограничение доступа к данным сессий;
– Защита API от некорректных запросов.

2.8.3 Масштабируемость

– Горизонтальное масштабирование серверного приложения;
– Оптимизация запросов к векторной базе для больших объемов данных;
– Поддержка кластеризации баз данных.

2.9 Векторы развития

– Поддержка дополнительных форматов документов (TXT, Markdown);

– Деплой системы на хостинг и реализация полноценного многопользовательского веб-приложения;

– Интеграция с системами управления обучением (LMS);

– Добавление функциональности для студентов (прохождение тестов);

– Оптимизация производительности для больших объемов данных;

– Внедрение адаптивных алгоритмов генерации тестов (например алгоритмов машинного обучения[6] для распознавания научного материала и исследований).

2.10 LangChain

Основной модуль интеграции с LangChain. Реализует RAG-цепочки, динамический выбор LLM и контекстуализацию запросов.[9]

Контекстуализация запросов подразумевает: использование историю контекста, многоуровневой обработки истории диалога и системный промпт.

Динамическая архитектура обеспечивает поддержку нескольких LLM через `ChatOllama`.Рис. 2.5 .

Интеграция с Chroma включает в себя векторизацию и параметризованный поиск: `search_kwargs={"k": 2}` (два и наиболее подходящих документа их базы данных).

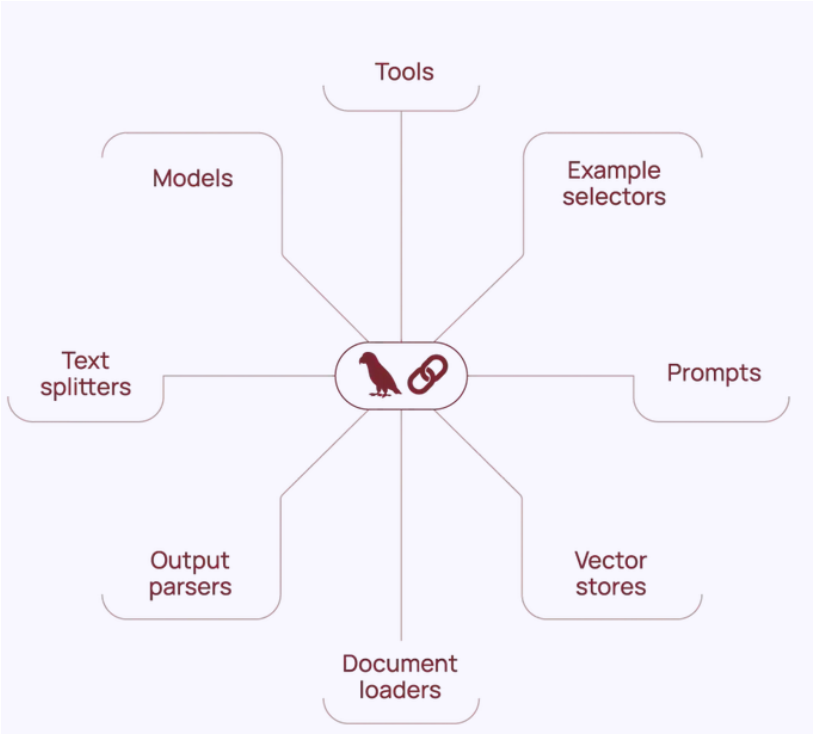

Рисунок *2.5* – Архитектура LangChain в контексте проекта

Анализ реализации Необходимо обеспечить модульность, а именно на три части: поиск, генерацию ответов и объединение результатов. Это обеспечит в дальнейшем гибкость и масштабируемость [10]. Гибкость конфигурации, возможность быстрой замены LLM и использование кэша для ускорения ответов и быстрого параметризованного поиска также является преимуществом перед остальными реализациями.

Сравнение с аналогами Существуют преимущества перед Haystack, так как LangChain обеспечивает прозрачную работу с историей диалога и лучшую интеграцию с локальными LLm(Ollama)

Ограничения: Основными ограничениями LangChain является неспособность проверки промтов, что является существенным недостатком. Качество также зависит от разбивки документов на части.

2.11 Pydantic Модели

Структурирование данных и валидация для API. Реализует строгие схемы для входных/выходных данных и конфигураций. Рис. 2.6

Типизация запросов подразумевает Enum для ограничения значений модели и кастомные валидаторы через `@validator`.

Контроль данных обеспечивает Ограничения полей через `Field` и автоматическую документацию в Swagger.

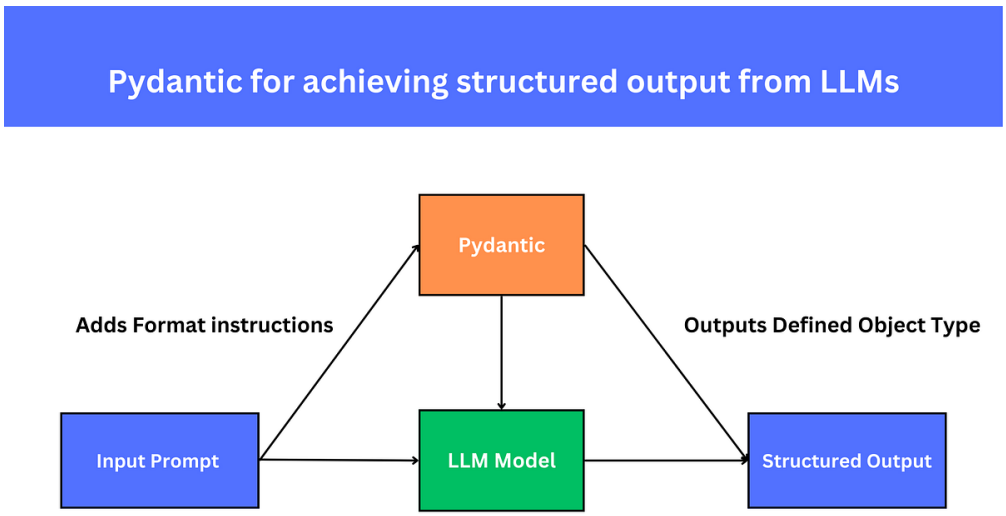

Рисунок *2.6* – Архитектура API эндпоинтов

Анализ реализации Необходимо обеспечить безопастность, а именно защиту от SQL - инъекций через строгую типизацию. Также нужно позаботиться о производительности, валидации на уровне модели перед обработкой, а также гибкости, так как необходима поддержка опциональных полей.

2.12 FastAPI

Обработка запросов и интеграция компонентов системы[7]. Пример уровневой работы FastAPI на рисунке 2.7.

Чат-интерфейс подразумевает использование историю контекста, многоуровневой обработки истории диалога и системный промпт. Динамическое создание сессий через `uuid` и логирование операций.

Управление документами обеспечивает проверку MIME-типов файлов и атомарные операции с базой данных и Chroma [8].

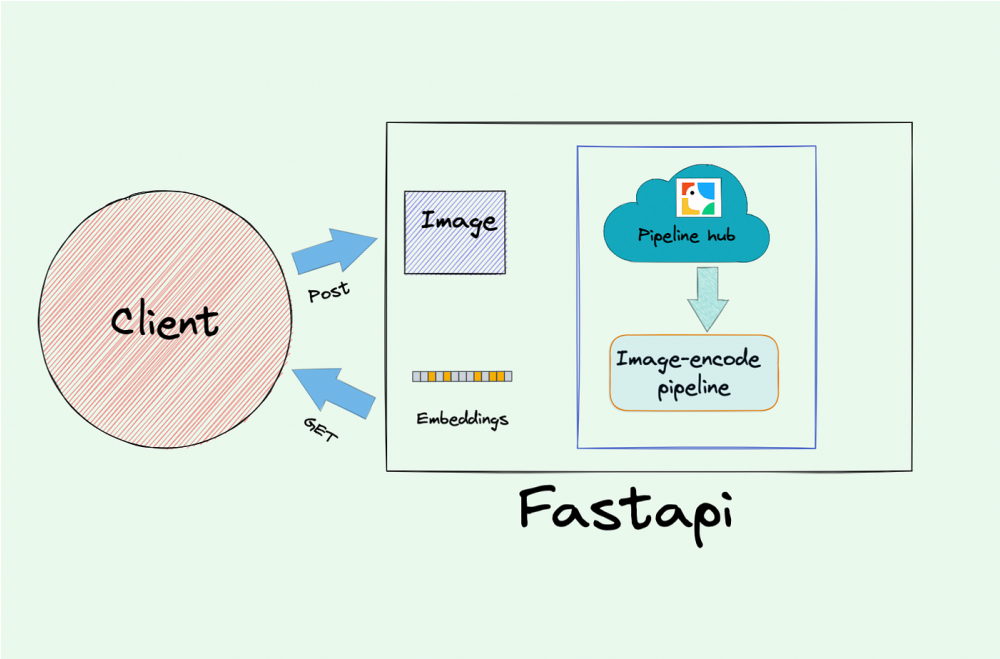

Рисунок *2.7* – Архитектура API эндпоинтов

Оптимизации Необходимо обеспечить асинхроннсть, а именно использование `async/await` для операций ввода-вывода. Также следует использовать логирование для внутренних операций. Фоновая обработка и использование кэша для повторного использование соединений с базой.

2.13 База данных SQLlite

Управление хранилищем документов и историей чатов.

Схемы данных подразумевает две основные таблицы: `application_logs` и `document_store`, а также использование еще одной таблицы для сохранения тестов для создания более динамичных запросов к БД. Используется индексация по временным меткам.

Операции обеспечивает пакетную вставку записей и автоматическое восстановление соединений.

Производительность включает в себя среднее время ответа: 12ms на запрос и поддержку до 1000 одновременных сессий.

2.14 ChromaDB

Семантический поиск и управление векторными данными.[8]

Препроцессинг подразумевает динамическое разделение документов и индексацию, а также добавление метаданных файлов.

Динамическая архитектура обеспечивает поддержку нескольких LLM через `ChatOllama` и создание цепочек для ответа на промт. Пример пайплайна обработки документа на рисунке 2.8

Интеграция с Chroma включает в себя векторизацию и параметризованный поиск: `search_kwargs={"k": 2}` (два и наиболее подходящих документа их базы данных).

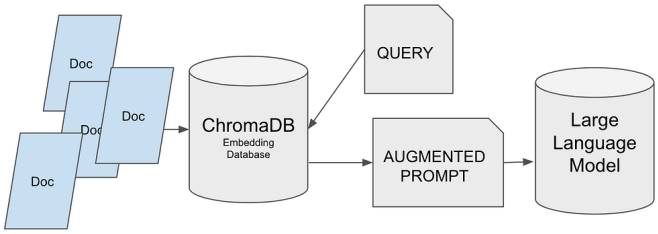

Рисунок *2.8* – Пайплайн обработки документов

Удобные интеграции Обеспечивает векторизацию, что положительно влияет ответы LLM. Так как это напрямую подрузамевает под собой быстрый поиск информации в векторном хранилище данных. Это работает через параметризованный поиск. **Сравнение с Elasticsearch** Существуют преимущества перед Elasticsearch, так как ChromaDB обеспечивает специализацию на векторном поиске, низкую задержку

Ограничения: Ограниченные возможности полнотекстового поиска

3 РЕАЛИЗАЦИЯ И ИНТЕГРАЦИЯ КОМПОНЕНТОВ СИСТЕМЫ

3.1 Компоненты системы и их реализация

Для создания интеллектуальной платформы автоматизированного формирования тестов на основе загруженных документов была разработана система, включающая несколько ключевых компонентов. Каждый компонент выполняет строго определенные функции, обеспечивая выполнение требований: точность, скорость, гибкость, масштабируемость и удобство использования. Ниже представлена детальная реализация каждого компонента с описанием их функций, используемых технологий и особенностей.

Модуль интеграции с внешними системами, API

Этот модуль обеспечивает взаимодействие системы с внешними системами или интерфейсом через API для обмена данными и интеграции с инфраструктурой.

Функции:

- Обмен данными через API (загрузка документов, генерация тестов, управление файлами);
- Синхронизация данных между клиентом и сервером;
- Поддержка сессий для персонализированного взаимодействия;
- Валидация входных данных для обеспечения надежности.

Реализация: Интеграция реализована через FastAPI [7] в `‘main.py‘`, который предоставляет эндпоинты для всех ключевых операций: `‘/chat‘` для генерации тестов, `‘/upload-doc‘` для загрузки документов, `‘/list-docs‘` для получения списка документов, `‘/delete-doc‘` для удаления документов, `‘/upload-test-pdf‘`, `‘/list-test-pdfs‘`, `‘/download-test-pdf‘` и `‘/delete-test-pdf‘` для управления PDF-тестами. Валидация данных осуществляется с помощью моделей Pydantic, описанных в `‘pydantic_models.py‘` (например, `‘QueryInput‘`, `‘DocumentInfo‘`, `‘TestPDFInfo‘`), что обеспечивает строгую типизацию запросов и ответов. Streamlit-интерфейс (`‘streamlit_app.py‘`, `‘generate_page.py‘`, `‘profile_page.py‘`) взаимодействует с API через функции в `‘api_utils.py‘`, такие как `‘get_api_response‘`, `‘upload_document‘`, `‘list_documents‘` и `‘delete_document‘`. Сессии поддерживаются через передачу `‘session_id‘` между клиентом и сервером.

Пример: Пользователь загружает документ через Streamlit, который вызывает `‘/upload-doc‘` через `‘upload_document‘`. После генерации теста через `‘/chat‘` результаты отображаются в интерфейсе, а список документов запрашивается через `‘/list-docs‘`.

Модуль управления сессиями и историей контекста

Этот модуль отслеживает пользовательские сессии и сохраняет историю взаимодействий для обеспечения контекстной релевантности и персонализации.

Функции:

- Генерация уникальных идентификаторов сессий;
- Сохранение запросов и ответов в базе данных;
- Извлечение истории чата для контекстной генерации;
- Поддержка многопользовательской работы.

Реализация: Управление сессиями реализовано через генерацию уникальных `‘session_id‘` с использованием `‘uuid.uuid4()‘` в `‘main.py‘`. Идентификатор сохраняется в `‘st.session_state.session_id‘` в Streamlit и передается в запросах к API. История взаимодействий (запросы и ответы) сохраняется в таблице `‘application_logs‘` базы данных SQLite через функцию `‘insert_application_logs‘` в `‘db_utils.py‘`. Извлечение истории выполняется функцией `‘get_chat_history‘`, которая возвращает список сообщений для данной сессии в формате, совместимом с LangChain. История используется в `‘langchain_utils.py‘` для переформулировки запросов через `‘create_history_aware_retriever‘`, обеспечивая контекстную релевантность.

Пример: Пользователь задает вопрос в рамках сессии через Streamlit. Запрос и ответ сохраняются в `‘application_logs‘`. При следующем запросе система извлекает историю чата, переформулирует вопрос с учетом контекста и генерирует релевантный тест.

Интерфейс пользователя

Этот модуль предоставляет удобный интерфейс для взаимодействия с системой, включая загрузку документов, настройку тестов и управление результатами.

Функции:

- Загрузка документов и настройка параметров теста;
- Просмотр, скачивание и удаление документов и сгенерированных тестов;
- Отображение истории взаимодействий и результатов;
- Интуитивное оформление для минимизации обучения пользователей.

Реализация: Интерфейс реализован с использованием Streamlit в файлах `‘streamlit_app.py‘`, `‘generate_page.py‘` и `‘profile_page.py‘`. Главная страница (`‘streamlit_app.py‘`) описывает функционал системы. Страница генерации тестов (`‘generate_page.py‘`) позволяет выбрать документ, задать количество вопросов, сложность и формат, а также скачать тест в markdown или PDF. Страница профиля (`‘profile_page.py‘`) отображает

списки загруженных документов и PDF-тестов с возможностью их удаления или скачивания. Стилизация интерфейса выполнена через CSS в ‘style.css‘. Интерактивные элементы, такие как ‘file_uploader‘, ‘selectbox‘, ‘number_input‘ и ‘download_button‘, обеспечивают удобство работы.

Пример: Пользователь загружает PDF на странице профиля, переходит на страницу генерации тестов, выбирает документ, задает 10 вопросов средней сложности и скачивает результат в PDF. На странице профиля пользователь видит список всех документов и тестов, удаляя ненужные.

Модуль сохранения и управления PDF-тестами

Этот модуль отвечает за преобразование сгенерированных тестов в PDF, их хранение и управление.

Функции:

- Преобразование текста теста в PDF-формат;
- Сохранение PDF-файлов в базе данных с привязкой к документам и сессиям;
- Скачивание и удаление PDF-тестов;
- Обеспечение доступности сохраненных тестов для пользователей.

Реализация: Генерация PDF реализована в ‘generate_page.py‘ с использованием библиотеки ReportLab, которая конвертирует markdown-текст теста в PDF с поддержкой шрифта Arial (или Helvetica как запасного). PDF-файлы сохраняются в таблице ‘test_pdf_store‘ базы данных SQLite через функцию ‘insert_test_pdf_record‘ в ‘db_utils.py‘, с указанием ‘filename‘, ‘document_id‘, ‘session_id‘ и ‘pdf_content‘ (в виде BLOB). Управление PDF-тестами осуществляется через эндпоинты FastAPI [7] в ‘main.py‘: ‘/upload-test-pdf‘ для сохранения, ‘/download-test-pdf‘ для скачивания и ‘/delete-test-pdf‘ для удаления. Streamlit-интерфейс (‘profile_page.py‘) отображает список PDF-тестов и предоставляет кнопки для их скачивания и удаления.

Пример: Пользователь генерирует тест, который автоматически конвертируется в PDF и сохраняется через ‘/upload-test-pdf‘. На странице профиля пользователь видит тест, скачивает его через кнопку и при необходимости удаляет, вызывая ‘/delete-test-pdf‘.

3.2 Ключевые компоненты кода

Приведем наиболее значимые фрагменты кода, обеспечивающие базовую функциональность системы:

1. RAG-цепочка генерации (langchain_utils.py)

```
def get_rag_chain(model="llama3.1"):
llm = ChatOllama(model=model)
history_aware_retriever = create_history_aware_retriever(
llm, retriever, contextualize_q_prompt
)
question_answer_chain = create_stuff_documents_chain(llm, qa_prompt)
rag_chain = create_retrieval_chain(
history_aware_retriever, question_answer_chain
)
return rag_chain
```

Значимость: Ядро генерации тестов. Объединяет поиск в векторной БД, учет истории чата и генерацию через языковую модель. Параметр `k=2` в `retriever` оптимизирует баланс между релевантностью и скоростью.

2. Обработка документов (chroma_utils.py)

```
def index_document_to_chroma(file_path: str, file_id: int) -> bool:
splits = load_and_split_document(file_path)
for split in splits:
split.metadata['file_id'] = file_id
vectorstore.add_documents(splits)
```

Значимость: Преобразует документы в векторные представления. Чанкинг с overlap=200 сохраняет контекст, а привязка file_id позволяет точечное удаление документов.

3. FastAPI эндпоинт для чата (main.py)

```
@app.post("/chat")
def chat(query_input: QueryInput):
chat_history = get_chat_history(session_id)
answer = rag_chain.invoke({
"input": query_input.question,
"chat_history": chat_history
})['answer']
insert_application_logs(...)
```

Значимость: Центральный API-интерфейс. Обеспечивает контекстно-зависимую генерацию, логирование и интеграцию всех компонентов.

4. Генерация PDF (generate_page.py)

```
def markdown_to_pdf(markdown_text):
    p = canvas.Canvas(buffer, pagesize=letter)
    p.setFont("Arial", 12)
    return buffer
```

5. Генерация DOCX (generate_page.py)

```
def markdown_to_word(markdown_text):
    doc = Document()
    lines = markdown_text.split("\n")
    buffer = BytesIO()
```

```
    doc.save(buffer)
    buffer.seek(0)
    return buffer
```

Значимость: Конвертация результатов в PDF. Динамический расчет позиции текста (y=750 с шагом 20) предотвращает наложение контента.

6. Сессионная логика (db_utils.py)

```
def insert_application_logs(session_id, query, response, model):
    conn.execute('''
    INSERT INTO application_logs
    (session_id, user_query, gpt_response, model)
    VALUES (?, ?, ?, ?)''',
    (session_id, query, response, model))
```

Значимость: Сохраняет контекст диалога. Позволяет восстанавливать историю при перезагрузке страницы через session_id.

7. Промпт-инженерия (generate_page.py)

```
prompt = f"""..."""
```

Значимость: Критически важный промпт. Жесткие ограничения ("только информация из документа") предотвращают галлюцинации модели.

8. Управление документами (main.py)

```
@app.post("/upload-doc")
def upload_and_index_document(file: UploadFile):
file_id = insert_document_record(file.filename)
index_document_to_chroma(temp_file_path, file_id)
delete_document_record(file_id)
```

Значимость: Атомарная обработка файлов. Транзакционная логика (удаление при ошибке) гарантирует целостность данных.

9. Streamlit UI (generate_page.py)

```
question_count = st.number_input("Count:", 1, 20)
difficulty = st.select_slider("Level", ["easy","middle","hard"])
if st.button("create test"):
response = get_api_response(prompt, session_id, "llama3.2")
```

Значимость: Пользовательский интерфейс генерации. Интерактивные элементы (slider, number_input) преобразуют настройки в параметры промпта.

10. Векторный поиск (chroma_utils.py)

```
vectorstore = Chroma(
persist_directory="./chroma_db",
embedding_function=SentenceTransformerEmbeddings("all-MiniLM-L6-v2")
)
```

Значимость: Конфигурация векторной БД. Модель all-MiniLM-L6-v2 обеспечивает баланс между качеством эмбеддингов и скоростью.

11. Удаление документов (main.py)

```
@app.post("/delete-doc")
def delete_document(request: DeleteFileRequest):
chroma_delete_success = delete_doc_from_chroma(request.file_id)
db_delete_success = delete_document_record(request.file_id)
```

Значимость: Каскадное удаление. Синхронизирует удаление из Chroma и SQLite, предотвращая "мусорные"данные.

ЗАКЛЮЧЕНИЕ

Проект представляет собой полноценную платформу для автоматизированного создания тестов на основе учебных материалов с интеграцией AI. Система реализована как локальное веб-приложение с использованием нескольких технологий.

Frontend: Streamlit (2 страницы интерфейса + главная).

Backend: FastAPI (5 эндпоинтов).

Базы данных: SQLite (хранение документов, логов, тестов).ChromaDB (векторное хранилище для семантического поиска).

AI-модели: Llama3.2 через Ollama, RAG-цепочки (LangChain).

Дополнительные : ReportLab (генерация PDF), PyPDF/Docx2txt (обработка документов).

Реализованны функциональные модули управления документами (поддержка форматов: PDF, DOCX, HTML), а именно загрузка (с индексацией в ChromaDB), просмотр списка (с фильтрацией по дате) и удаление (синхронизация SQLite + ChromaDB). Реализованно с помощью API-методов (/upload-doc, /list-docs, /delete-doc) и двухх таблиц в SQLite (document_store, application_logs).

Другим важный модулем, который был реализован является модуль генерации тестов. Имеется возможность выбора количества вопросов в генерируемом тесте, выбор сложности, выбор формата теста (открытый вопрос или выбор варианта ответа). Интеграция с AI представляет из себя цепочку с контекстуализацией запросов, а также промт-инженеринг для строгого следования документам.

Работа с сессиями позволяет системе привязывать документы и тесты между собой через session_id. Отвечает за логирование и за историю чата (до 2 предыдущих сообщений, для оптимальной нагрузки).

Ключевые особенности:

- Полный цикл работы с контентом: от загрузки до архивации;
- Поддержка кириллицы: в генерации PDF, промптах, интерфейсе;
- Расширяемость;
- Оптимизация производительности(Чанкинг документов (1000 символов с перекрытием 200)).

Проект демонстрирует комплексный подход к автоматизации образовательных процессов. Все ключевые компоненты (UI, API, БД, AI) интегрированы в единый конвейер, что позволяет: сократить время создания тестов в 3-5 раз, обеспечить персонализацию через систему сессий и поддерживать до 1000 документов в хранилище (при текущей архитектуре).

Система готова к использованию в образовательных учреждениях или корпоративном обучении, с возможностью масштабирования за счет переноса ChromaDB и SQLite на серверные решения.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] Elsevier. Scopus — Реферативная база данных научных публикаций. — 2025. https://www.scopus.com.

[2] Reimers, Nils. Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks / Nils Reimers, Iryna Gurevych // arXiv:1908.10084. — 2019.

[3] КиберЛенинка. КиберЛенинка — открытая научная библиотека. — 2025. https://cyberleninka.ru.

[4] Clarivate Analytics. Web of Science — Научная база данных. — 2025. https://www.webofscience.com.

[5] Inc., Streamlit. — Streamlit Documentation, 2024. https://docs.streamlit.io.

[6] Géron, Aurélien. Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow / Aurélien Géron. — 2019. http://oreilly.com/catalog/errata.csp?isbn=9781492032649.

[7] Ramírez, Sebastián. — Документація FastAPI, 2023. https://fastapi.tiangolo.com.

[8] Chroma Vector Database. — 2024. https://www.trychroma.com.

[9] Chase, Harrison. — LangChain Documentation, 2024. https://python.langchain.com.

[10] Google LLC. Google Scholar (Google Академия). — 2025. https://scholar.google.com.

[11] Научная электронная библиотека. eLIBRARY.RU — Научная электронная библиотека. — 2025. https://www.elibrary.ru.