ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ ОБРАЗОВАТЕЛЬНОЕ

УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ

(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)

**Отчет**

**По проекту « ОКНО ЗНАНИЙ »**

**Хакатона “T1”**

**ПО ДИСЦИПЛИНЕ «РАЗРАБОТКА ИТ-ПРОЕКТОВ 2 семестр»**

**Команда:**

Осташевская Алёна М80-206Б-2023

Погожев Максим М80-201Б-2023

**Москва 2024**

**Описание задачи:**

Создать платформу, которая позволяет компаниям:

- быстро интегрировать существующие базы знаний с цифровыми ассистентами.

-Кастомизировать окна взаимодействия для обеспечения удобного доступа к информации.

- Выбрать модель для обучения базы знаний из публичных и доступных сервисом.

- Вставить созданное окно взаимодействия через прямую ссылку собственного сервиса, API или в существующий сайт компании, как чат бота.

**Решение:**

Проект представляет собой сервер для обработки вопросов и ответов (QA) на основе загруженных файлов.

Он объединяет несколько модулей для обработки файлов, анализа текста, создания эмбеддингов и выполнения QA-цепочки с использованием моделей машинного обучения. Проект также интегрирует Kafka и Spark для распределенной обработки данных и управления знаниями.

Основные компоненты и их назначение

1. \*\*core/\*\*:

- \*\*api\_key\_handler.py\*\*: Управляет API-ключами для доступа к внешним сервисам (например, OpenAI).

- \*\*file\_handlers.py\*\*: Обрабатывает загруженные файлы (например, PDF, TXT) и извлекает из них текст.

- \*\*text\_processor.py\*\*: Разделяет текст на чанки и создает эмбеддинги для дальнейшегqa\_chain.py \*\*qa\_chain.py\*\*: Запускает QA-цепочку, используя модель машинного обучения для ответа на вопросы.

- \*\*knowledge\_base.py\*\*: Интегрирует Kafka и Spark для обработки и хранения знаний (например,orchestrator.py \*\*orchestrator.py\*\*: Координирует все этапы обработки, от загрузки файлов до полmodel\_launcher.py\*model\_launcher.py\*\*: Инициализирует и управляserver/кой.

2. \*\*server/\*\*:

- \*\*server\_core.py\*\*: Реализует сервер на FastAPI, который предоставляет API для загрузкиconfig/опросов.

3. \*\*config/\*\*:

- \*\*config.yaml\*\*: Содержит конфигурации для всех модулей (например, параметры Karequirements.txtи).

4. \*\*requirements.txt\*\*:

- Список зависимостей Python, необходиmain.pyоты проекта.

5. \*\*main.py\*\*:

- Точка входа для запуска сервера.

Загрузка файлов и вопросов

1.Загрузка файлов и вопросов\*\*:

- Пользователь загружает файлы (например, документы) и задает воОбработка файловра.

2. \*\*Обработка файлов\*\*:

- Файлы обрабатываются модулем file\_handlers.py, Создание чанков и эмбеддинговдание чанков и эмбеддингов\*\*:

- Текст разделяется на чанки с помощью text\_processor.py, и для каждого чанкИнтеграция с Kafka и SparkИнтеграция с Kafka и Spark\*\*:

- Чанки и эмбеддинги отправляются в Kafka для потоковой обработки. Spark используется для распределенной обработки данных (например, фПоиск релевантных чанков\*\*Поиск релевантных чанков\*\*:

- Модуль qa\_chain.py находит чанки, наиболее релевантныЗапуск QA-цепочкия.

6. \*\*Запуск QA-цепочки\*\*:

- На основе релевантных чанков и вопроса пользователя модель машинного обуВозврат результата.

7. \*\*Возврат результата\*\*:

- Сервер возвращает ответ пользователю через API.

---FastAPIизмы проекта

1. \*\*FastAPI\*\*:

- Используется для создания REST API, который принимает файлы и вопросы, а тKafkaращает ответы.

2. \*\*Kafka\*\*:

- Обеспечивает потоковую передачу данных (например, чанков и эмбедSparkежду модулями.

3. \*\*Spark\*\*:

- Используется для распределенной обработки данных, что позволяет эффективно работать с большиМодели машинного обучения\*Модели машинного обучения\*\*:

- QA-цепочка использует модели (например, OpenAI GPT) для генераЛогирование и обработка ошибокрование и обработка ошибок\*\*:

- Все этапы работы логируются, а ошибки обрабатываются для обеспечения стабильности системы.

**Основная цель проекта** — предоставить удобный и масштабируемый инструмент для анализа текстовых данных и ответов на вопросы. Проект может быть использован в различных сценариях, таких как:

- Автоматизация обработки документов.

- Построение систем вопросов и ответов на основе корпоративных знаний.

- Интеграция с чат-ботами или другими приложениями, требующими анализа текста.

**Структура проекта:**

│

├── backend\_utils/ # модуль для backend utilities

│ ├── \_\_init\_\_.py

│ ├── api\_key\_handler.py # ApiKeyHandler класс

│ ├── file\_handlers.py # FileHandlerFactory класс

│ ├── text\_processor.py # DefaultTextProcessor класс

│ └── qa\_chain.py # QAChainRunner класс

│

├── knowledge\_base/ # модуль для интеграции базы знаний

│ ├── \_\_init\_\_.py

│ └── knowledge\_base\_plug.py # KnowledgeBasePlug класс

│

├── model/ # модуль для модели orchestration

│ ├── \_\_init\_\_.py

│ ├── orchestrator.py # Orchestrator класс

│ └── model\_launcher.py # ModelLauncher класс

│

├── server/ # модуль ядра сервера

│ ├── \_\_init\_\_.py

│ └── server\_core.py # ServerCore класс

│

├── config/ # файлы конфигурации

│ └── config.yaml # файл конфигурации ParameterController

│

├── tests/ # модульный и интеграционный тесты

│ ├── \_\_init\_\_.py

│ ├── test\_api\_key\_handler.py

│ ├── test\_file\_handlers.py

│ ├── test\_text\_processor.py

│ ├── test\_qa\_chain.py

│ ├── test\_knowledge\_base\_plug.py

│ ├── test\_orchestrator.py

│ ├── test\_model\_launcher.py

│ └── test\_server\_core.py

│

├── scripts/ # модульный скрипты

│ ├── \_\_init\_\_.py

│ └── start\_server.py # скрипт запуска сервера

│

├── requirements.txt # Python-зависимости

├── README.md # документация проекта

└── main.py # точка входа подачи заявки

**Выводы:**

- Модульность: Каждый компонент может быть легко заменен или расширен.

- Масштабируемость: Использование Kafka и Spark позволяет обрабатывать большие объемы данных.

- Удобство: Простой API для интеграции с другими приложениями.

- Гибкость: Поддержка различных типов файлов и моделей машинного обучения.

Этот проект идеально подходит для команд, которые хотят автоматизировать анализ текстовых данных и построить интеллектуальные системы вопросов и ответов.