**1. Название проекта:** AIHub

**2. Авторы:** Аксютин Олег Романович

**3. Описание идеи**

Проект представляет собой Telegram-бота, предоставляющего пользователям удобный интерфейс для взаимодействия с различными моделями искусственного интеллекта (**Gemini**, **Llama**, **Midjourney**, **FLUX**). Бот позволяет пользователям выбирать конкретные AI-модели для решения своих задач или сравнивать результаты работы разных моделей в режиме "Арены".

Основные возможности:

* **Единая точка доступа:** Пользователи могут взаимодействовать с множеством AI-моделей через одного бота, не переключаясь между разными сервисами.
* **Поддержка различных модальностей:** Бот поддерживает работу с текстовыми запросами, голосовыми сообщениями (с автоматической транскрипцией) и изображениями (как для генерации, так и для анализа, в зависимости от возможностей выбранной модели).
* **Режим "Single":** Пользователь выбирает конкретную модель, после чего напрямую взаимодействует с ней, отправляя запросы и получая ответы.
* **Режим "Arena":** Пользователь отправляет запрос, а бот автоматически подбирает две случайные модели, способные обработать этот тип запроса (например, две текстовые модели или две модели для генерации изображений). Бот отправляет результаты от обеих моделей и предлагает пользователю проголосовать за лучший ответ. Это позволяет сравнивать производительность и качество разных AI по мнению пользователя в общую статистику.
* **Гибкая настройка:** Пользователь может легко переключаться между режимами и выбирать модели через меню настроек.
* **Модульность:** Система регистрации моделей позволяет легко добавлять поддержку новых AI-моделей в будущем.

**4. Описание реализации**

В основе реализации проекта лежит асинхронный фреймворк **aiogram** для создания Telegram-ботов на Python. Архитектура построена на разделении логики по модулям, что обеспечивает гибкость и расширяемость.

### 4.1. Ядро системы и управление моделями:

* **Центральный реестр (registry.py):** Ключевым компонентом является **AIRegistry** – синглтон-класс для регистрации и управления доступными AI-моделями. Модели (классы, наследующие **BaseAIModel**) автоматически регистрируются при старте бота с помощью декоратора ***@register\_model***. Реестр позволяет получать модели по провайдеру, версии или типу возможностей (**TextToTextModel, TextToImgModel, AudioToTextModel** и т.д.), используя метаданные, хранящиеся в **ModelInfo**.
* **Управление диалогами (FSM):** Взаимодействие с пользователем и управление контекстом диалога реализовано с помощью машины состояний фреймворка **aiogram** (**states.py**). Это позволяет четко отслеживать текущее состояние пользователя (например, выбор режима, ожидание запроса, голосование в арене) и хранить связанные данные (FSMContext).

### 4.2. Модули и обработка запросов:

* **Инициализация (main.py):** Запускает бота, загружает переменные окружения (**dotenv**), инициализирует **AIRegistry** путем импорта модулей с реализациями моделей, настраивает диспетчер **aiogram** и подключает роутеры обработчиков.
* **Реализации AI-моделей (ai/):** Каждый модуль (например, **gemini.py**, **llama.py**, **flux.py**, **midjourney.py**, **whisper.py**) содержит классы, реализующие взаимодействие с конкретными AI. Они инкапсулируют логику вызовов API и обработку ответов.
  + **Интеграции:**
    - **Google Gemini API:** Используется для моделей ***Gemini*** (через библиотеку ***google-genai***), поддерживающих генерацию текста, анализ и генерацию изображений, транскрибацию аудио.
    - **Groq Cloud:** Предоставляет доступ к моделям ***Llama 3*** (через библиотеку ***groq***) для быстрой генерации текста.
    - **Gradio Spaces:** Интеграция с моделями, развернутыми на ***Hugging Face*** (через ***gradio\_client***), такими как ***Whisper*** (для транскрибации аудио), ***Midjourney*** и ***FLUX*** (для генерации изображений по текстовому описанию).
* **Обработчики (handlers/):** Модули (**settings\_handlers.py**, **single\_chat\_handlers.py**, **arena\_chat\_handlers.py**) содержат асинхронные функции для обработки команд (***/start***, ***/settings***), текстовых сообщений, голосовых сообщений, изображений и колбэков от inline-клавиатур. Они взаимодействуют с **FSM** и **AIRegistry** для определения логики ответа.
  + **Обработка типов запросов:**
    - **Текст:** Запросы обрабатываются моделями ***Llama*** (через **Groq**) или ***Gemini*** (в зависимости от выбора пользователя или режима арены).
    - **Изображения:** При получении изображения с текстом (*caption*) может выполняться анализ контента (**Gemini ImgToTextMode**l). При текстовом запросе и выборе соответствующей модели происходит генерация картинок (**Midjourney**, **FLUX**, **Gemini TextToImgModel**).
    - **Аудио:** Голосовые сообщения сначала транскрибируются с использованием **Whisper** (через **transcription.py** и ***gradio\_client***), а полученный текст затем обрабатывается выбранной текстовой моделью если она сама не может работать с аудио.
* **Режим "Арена" (arena\_chat\_handlers.py):**
  + При получении запроса пользователя выбираются две случайные модели из **AIRegistry**, соответствующие типу запроса (текст, изображение) и типу арены, выбранному пользователем.
  + Обе модели выполняются параллельно (по возможности).
  + Результаты от обеих моделей отправляются пользователю.
  + Пользователю предлагается проголосовать за лучший ответ с помощью inline-клавиатуры. Голосование обрабатывается соответствующим колбэк-хендлером.
* **Клавиатуры (keyboards/):** Генерируют **InlineKeyboardMarkup** и **ReplyKeyboardMarkup** для навигации по меню и взаимодействия с ботом.
* **Утилиты (utils/, response\_handler.py):** Вспомогательные функции для разделения текста (**split\_text**), транскрибации аудио (**transcribe\_voice\_message**), и унифицированной отправки ответов пользователю (**handle\_model\_response**), корректно обрабатывая текст (с **Markdown**) и изображения.
* **Управление данными и Квотами (database.py):**
  + Проект использует легковесную базу данных **SQLite** (database.py) для сохранения постоянных данных.
  + В базе хранятся:
    - **Список AI-моделей** с их уникальными ID, отображаемыми именами и, что важно для режима "Арена", их **рейтингами Elo**.
    - **Информация о пользователях**, включая их максимальный дневной лимит запросов (квоту), использованную квоту и дату последнего сброса квоты.
  + Реализован механизм **дневной квоты запросов** для каждого пользователя. Перед выполнением каждого запроса к модели (особенно в режимах Single и Arena) автоматически проверяется, достаточно ли у пользователя оставшейся квоты (quota\_check декоратор).
  + При успешном выполнении запроса (или в случае Арены, после ответа моделей) соответствующее количество квоты **списывается** с пользователя.
  + Сброс дневной квоты реализован по принципу 'ленивого сброса': использованная квота обнуляется и дата последнего сброса обновляется при первом обращении пользователя к боту в течение нового дня, независимо от типа запроса
  + Рейтинги моделей в режиме "Арена" **персистентно хранятся** в базе данных и обновляются после каждого голосования.

## 5. Описание технологий и необходимых библиотек

Проект разработан на языке программирования **Python** с использованием асинхронного подхода.

**Основные библиотеки и фреймворки:**

* **aiogram:** Асинхронный фреймворк для создания Telegram-ботов. Используется для обработки обновлений, маршрутизации запросов (роутеры), управления состоянием диалога (**FSM**), работы с API Telegram (отправка сообщений, клавиатуры).
* **google-genai:** Официальный клиент ***Google*** для взаимодействия с API моделей ***Gemini***.
* **groq:** Клиент для работы с ***API Groq Cloud***, предоставляющим доступ к моделям ***Llama***.
* **gradio\_client:** Библиотека для взаимодействия с моделями ИИ, развернутыми на платформе ***Hugging Face***.
* **Pillow (PIL Fork):** Используется для базовой обработки и подготовки изображений перед отправкой пользователю.
* **python-dotenv:** Для управления переменными окружения и безопасного хранения API-ключей.
* **requests:** Стандартная де-факто библиотека для выполнения HTTP-запросов (используется ***gradio\_client*** и для скачивания изображений с URL).

**Вспомогательные и стандартные библиотеки:**

* **asyncio:** Основа для асинхронного выполнения операций ввода-вывода (работа с сетью, API).
* **io, tempfile:** Для работы с потоками данных в памяти (при передаче изображений) и временными файлами (для транскрибации аудио).
* **os, typing, dataclasses, collections, abc, random:** Стандартные библиотеки Python для работы с ОС, типизации, создания структур данных, абстрактных классов и генерации случайных чисел.

**Обоснование выбора технологий:**

* **Python и Asyncio:** Предоставляют мощную и удобную среду для разработки I/O-bound приложений, таких как Telegram-боты, взаимодействующие с множеством внешних API.
* **Aiogram:** Выбран как современный, активно развивающийся и гибкий фреймворк для Telegram, отлично подходящий для реализации сложных сценариев взаимодействия с использованием FSM и роутеров.
* **Модульная архитектура:** Позволяет легко поддерживать, тестировать и расширять проект, добавляя новые AI-модели или функциональные возможности без значительной переработки существующего кода.
* **Централизованный реестр (AIRegistry):** Упрощает управление моделями и делает код более чистым и понятным.
* **Интеграция с Gradio:** Дает возможность легко подключать и тестировать множество открытых и экспериментальных AI-моделей, доступных на ***Hugging Face Spaces***, без необходимости их самостоятельного развертывания и поддержки.
* **Использование специализированных API-клиентов (google-genai, groq):** Упрощает и стандартизирует взаимодействие с соответствующими облачными платформами AI.

## 6. Скриншоты





