**EIEI Generic Modular AI Response API Documentation**

**Introduction**

This documentation outlines the architecture and workflow of the EIEI Generic Modular AI Response API. This API is designed to facilitate communication between various messaging platforms and an AI service. The system is capable of processing different types of input (text, audio, image) and is easily adaptable to changes in external service providers.

**Workflow Description**

**Overview**

The application workflow encompasses several steps, from message reception to response dispatch. It utilizes a webhook for message reception and employs scripts for processing and database interaction.

**Workflow Steps**

1. Message Reception

* **Bot Receives Message via Webhook**: Messages from the messaging platform are received through a webhook, which forwards the data to the application's API.

2. Message Processing

* **Conversion to Generic Format**: Received messages are converted from the platform-specific format to a generic format suitable for AI processing.

3. Input Type Handling

* **Text**: Directly stored in the database.
* **Audio**: Transcribed to text and stored.
* **Image**: Converted to a descriptive text and stored.

4. Database Operations

* **Storage of Text and Metadata**: Both the text and the type of input are stored in the database, along with relevant metadata.

5. Outgoing Message Processing

* **Regular Checks for New Messages**: A background process checks the database every 5 seconds for new messages.
* **Creation of JSON Payload**: New messages trigger the creation of a JSON payload, containing all necessary information for AI processing.

6. Response Generation and Dispatch

* **Communication with AI Service**: The payload is sent to an AI service, which generates a response.
* **Delivery of Response**: The AI-generated response is sent back to the user through the messaging platform.

**Database Schema**

**Message Table (tbl\_200\_messages)**

* **pk\_messages\_ID**: Primary key (sequential)
* **channel**: Source of the message (e.g., "TELEGRAM", "WHATSAPP")
* **bot\_id**, **chat\_id**: Identifiers from the channel system
* **type**: Type of the original message (AUDIO/TEXT/IMAGE)
* **role**: Creator of the record (HUMAN/BOT)
* **content\_text**: The message, transcription, or image description
* **file\_id**: File identifier from the channel system
* **message\_timestamp**: Timestamp of message reception or sending
* **update\_id**, **message\_id**: Identifiers related to Telegram
* **is\_processed**: Status of message processing
* **created\_by**, **created\_on**, **updated\_by**, **updated\_on**: Audit fields

**Technology Stack**

* **Web Framework**: FASTAPI
* **Database**: POSTGRESQL
* **Hosting**: Render.com

**Adaptability and Modularity**

The system is designed for easy adaptation to different AI services and messaging platforms. Changes to service providers or AI models can be made with minimal alterations to the global environment file, ensuring flexibility and ease of maintenance.

**1. Project Structure**

The project can be structured into several directories and files:

* **/main.py**: The entry point of the application.
* **/app**: Contains the application logic.
  + **/controllers**: Handles the routing of requests.
  + **/services**: Business logic for different functionalities.
  + **/models**: Data models and schemas.
  + **/utils**: Helper functions and utilities.
  + **/integrations**: Contains modules for integrating with different services (e.g., AI APIs, messaging platforms).
  + **/config**: Configuration settings and environment variables.
* **/tests**: Unit and integration tests for the application.

**2. Main Components**

**main.py**

* Initializes the FastAPI application.
* Includes routes that connect to the controllers in **/app/controllers**.

**/app/controllers**

* Define routes for receiving and sending messages.
* Call services from **/app/services** to process the data.

**/app/services**

* Business logic for handling different types of inputs (text, audio, image).
* Modules for database interaction.
* Service for constructing JSON payloads and communicating with AI APIs.

**/app/models**

* Define database models and schemas (e.g., for the **tbl\_200\_messages** table).

**/app/utils**

* Helper functions (e.g., format conversions, timestamp handling).

**/app/integrations**

* Modules for interacting with external services (e.g., Telegram API, AI service providers).
* Each module in this directory should have a common interface so that swapping services requires minimal changes in other parts of the application.

**/app/config**

* Configuration settings, API keys, and environment variables.

NOTES:

Local initialization steps:

start PostgreSQL service:

brew services start postgresql

psql -d postgres

Activate the Virtual Environment:

source venv/bin/activate

Running the Application

uvicorn main:app --reload

---

CREATE DATABASE eiei\_api;

psql -d eiei\_api

CREATE USER admin\_eiei\_user2 WITH PASSWORD '12345678';

GRANT ALL PRIVILEGES ON DATABASE eiei\_api TO admin\_eiei\_user2;

GRANT ALL PRIVILEGES ON ALL TABLES IN SCHEMA public TO admin\_eiei\_user2;

GRANT ALL PRIVILEGES ON ALL SEQUENCES IN SCHEMA public TO admin\_eiei\_user2;

Databasename: eiei\_api

Admin user: admin\_eiei\_user2

Password: 12345678

CREATE TABLE tbl\_200\_messages ( pk\_messages SERIAL PRIMARY KEY, channel VARCHAR(100), bot\_id INTEGER NOT NULL, user\_id INTEGER, chat\_id INTEGER, type VARCHAR(100), role VARCHAR(100), content\_text VARCHAR(4000), file\_id VARCHAR(4000), message\_date TIMESTAMP, update\_id INTEGER, message\_id INTEGER, created\_by VARCHAR(1000), created\_on TIMESTAMP DEFAULT CURRENT\_TIMESTAMP, updated\_by VARCHAR(1000), updated\_on TIMESTAMP DEFAULT CURRENT\_TIMESTAMP );

Workflow inmage:

A diagram of a software program

Description automatically generated