Disclaimer

This is a research project. Please do not use it commercially and use it responsibly.

<hr>

WebAI-to-API

WebAI-to-API is a modular web server built with FastAPI, designed to manage requests across AI services like Gemini. It features a clean, extendable architecture that simplifies configuration, integration, and maintenance.

> **Note:** Currently, **Gemini** is the primary supported AI service.

Features

- **Endpoints Management**:
  - `/v1/chat/completions`
  - `/gemini`
  - `/gemini-chat`
  - `/translate`
- **Service Switching**: Easily configure and switch between AI providers via `config.conf`.
- **Modular Architecture**: Organized into clearly defined modules for API routes, services, configurations, and utilities, making development and maintenance straightforward.

Installation

1. **Clone the repository:**

   ```bash
   git clone https://github.com/Amm1rr/WebAI-to-API.git
   cd WebAI-to-API
   ```

2. **Install dependencies using Poetry:**

   ```bash
   poetry install
   ```

3. **Create and update the configuration file:**

   ```bash
   cp config.conf.example config.conf
   ```

   Then, edit `config.conf` to adjust service settings and other options.

4. **Run the server:**

   ```bash
   poetry run python src/run.py
   ```

Usage

Send a POST request to `/v1/chat/completions` (or any other available endpoint) with the required

payload.

Example Request

```json
{
  "model": "gemini-2.0-flash",
  "messages": [{ "role": "user", "content": "Hello!" }]
}
```

Example Response

```json
{
  "id": "chatcmpl-12345",
  "object": "chat.completion",
  "created": 1693417200,
  "model": "gemini-2.0-flash",
  "choices": [
    {
      "message": {
        "role": "assistant",
        "content": "Hi there!"
      },
      "finish_reason": "stop",
      "index": 0
    }
  ],
  "usage": {
    "prompt_tokens": 0,
    "completion_tokens": 0,
    "total_tokens": 0
  }
}
```

Roadmap

- Gemini Support: Implemented
- ~~Claude, ChatGPT Development~~: Discontinued

<details>
  <summary>
    <h2>Configuration </h2>
  </summary>

Key Configuration Options

Section	Option	Description	Example Value
[AI]	default_ai	Default service for `/v1/chat/completions`	`gemini`
[EnabledAI]	gemini	Enable/disable Gemini service	`true`
[Browser]	name	Browser for cookie-based authentication	`firefox`

The complete configuration template is available in [`WebAI-to-API/config.conf.example`](WebAI-to-API/config.conf.example).__
If the cookies are left empty, the application will automatically retrieve them using the default browser specified.

Sample `config.conf`

```ini
[AI]
# Default AI service.
default_ai = gemini

# Default model for Gemini.
default_model_gemini = gemini-2.0-flash

# Gemini cookies (leave empty to use browser_cookies3 for automatic authentication).
gemini_cookie_1psid =
gemini_cookie_1psidts =

[EnabledAI]
# Enable or disable AI services.
gemini = true

[Browser]
# Default browser options: firefox, brave, chrome, edge, safari.
name = firefox
```

</details>

Project Structure

The project now follows a modular layout that separates configuration, business logic, API endpoints, and utilities:

```plaintext
src/
├── app/
│   ├── __init__.py
│   ├── main.py               # FastAPI app creation, configuration, and lifespan management.
│   ├── config.py             # Global configuration loader/updater.
│   ├── logger.py             # Centralized logging configuration.
│   ├── endpoints/            # API endpoint routers.
│   │   ├── __init__.py
│   │   ├── gemini.py         # Endpoints for Gemini (e.g., /gemini, /gemini-chat).
│   │   └── chat.py           # Endpoints for translation and OpenAI-compatible requests.
│   ├── services/             # Business logic and service wrappers.
│   │   ├── __init__.py
│   │   ├── gemini_client.py  # Gemini client initialization, content generation, and cleanup.
│   │   └── session_manager.py # Session management for chat and translation.
│   └── utils/                # Helper functions.
│       ├── __init__.py
│       └── browser.py        # Browser-based cookie retrieval.
├── models/                   # Models and wrappers (e.g., MyGeminiClient).
│   └── gemini.py
├── schemas/                  # Pydantic schemas for request/response validation.
│   └── request.py
├── config.conf               # Application configuration file.
└── run.py                    # Entry point to run the server.
```

Developer Documentation

Overview

The project is built on a modular architecture designed for scalability and ease of maintenance. Its primary components are:

- **app/main.py:** Initializes the FastAPI application, configures middleware, and manages

application lifespan (startup and shutdown routines).
- **app/config.py:** Handles the loading and updating of configuration settings from `config.conf`.
- **app/logger.py:** Sets up a centralized logging system.
- **app/endpoints/:** Contains separate modules for handling API endpoints. Each module (e.g., `gemini.py` and `chat.py`) manages routes specific to their functionality.
- **app/services/:** Encapsulates business logic, including the Gemini client wrapper (`gemini_client.py`) and session management (`session_manager.py`).
- **app/utils/browser.py:** Provides helper functions, such as retrieving cookies from the browser for authentication.
- **models/:** Holds model definitions like `MyGeminiClient` for interfacing with the Gemini Web API.
- **schemas/:** Defines Pydantic models for validating API requests.

How It Works

1. **Application Initialization:**
   On startup, the application loads configurations and initializes the Gemini client and session managers. This is managed via the `lifespan` context in `app/main.py`.

2. **Routing:**
   The API endpoints are organized into dedicated routers under `app/endpoints/`, which are then included in the main FastAPI application.

3. **Service Layer:**
   The `app/services/` directory contains the logic for interacting with the Gemini API and managing user sessions, ensuring that the API routes remain clean and focused on request handling.

4. **Utilities and Configurations:**
   Helper functions and configuration logic are kept separate to maintain clarity and ease of updates.

Docker Deployment Guide

For Docker setup and deployment instructions, please refer to the Docker.md documentation.

Star History

[![Star History Chart](https://api.star-history.com/svg?repos=Amm1rr/WebAI-to-API&type=Date)](https://www.star-history.com/#Amm1rr/WebAI-to-API&Date)

License

This project is open source under the [MIT License](LICENSE).

> **Note:** This is a research project. Please use it responsibly, and be aware that additional security measures and error handling are necessary for production deployments.

[](https://github.com/Amm1rr/)