Karthik Project Documentation

Project Overview

This project is a Flask-based web application that utilizes Google Gemini's Generative Language API (gemini-2.0-flash) to generate text based on user-provided prompts. The application handles API calls, error management (including retries), and returns the generated text to the user. The application also incorporates CORS to enable requests from different origins. The Gemini API key is securely managed using environment variables.

Installation/Setup

1. **Clone the repository:** `git clone <repository_url>` (replace `<repository_url>` with the actual repository URL)
2. **Create a `.env` file:** In the root directory, create a file named `.env` and add your Gemini API key:

   ```
   GEMINI_API_KEY=YOUR_GEMINI_API_KEY
   ```

   Replace `YOUR_GEMINI_API_KEY` with your actual Google Gemini API key. **Do not commit this file to version control.**
3. **Install dependencies:**

   ```
   pip install Flask flask-cors requests python-dotenv
   ```

4. **Run the application:**

   ```
   python app.py
   ```

   The application will run on a default port (usually 5000).

Tech Stack

- **Python:** The primary programming language.
- **Flask:** A lightweight web framework for building the API.
- **Flask-CORS:** Enables Cross-Origin Resource Sharing, allowing requests from different domains.
- **Requests:** A library for making HTTP requests to the Gemini API.
- **python-dotenv:** For loading environment variables from a `.env` file.
- **Google Gemini API (gemini-2.0-flash):** The large language model used for text generation.

Feature List

- **Text Generation:** Accepts text prompts and generates text using the Google Gemini API.
- **Error Handling:** Includes retry logic for failed API calls.
- **Environment Variable Management:** Securely stores the Gemini API key in a `.env` file.
- **CORS Support:** Allows requests from different origins.
- **JSON Response:** Returns the generated text in a structured JSON format (although the provided code only returns the text string directly).

Code Architecture

The application is structured around a single Flask application (`app.py`). The core logic resides in the `call_gemini` function, which handles the interaction with the Gemini API. This function incorporates a retry mechanism to improve the robustness of the API calls. The Flask application handles routing and response formatting. The `GEMINI_API_KEY` and `GEMINI_API_URL` are defined using environment variables for security and maintainability.

Usage Examples

To use the application, you would send a POST request to the appropriate endpoint (likely `/generate`) with a JSON payload containing the prompt:

```
{
  "prompt": "Write a short story about a robot learning to love."
}
```

The response will contain the generated text. The exact endpoint and response format would need to be defined within the Flask application's routing. The currently provided code lacks explicit routing and return formatting beyond the text generated.

APIs or Functions Explained

`call_gemini(prompt, max_retries=5):`

This function is the core of the application. It takes a `prompt` (the text input for the Gemini API) and an optional `max_retries` parameter (defaulting to 5). It constructs a JSON payload for the API request, sends the request, and handles potential errors. The function uses a loop to retry the request up to `max_retries` times with an exponential backoff (delay increases with each failed attempt). If successful, it extracts and returns the generated text from the API's response. If all attempts fail, it implicitly returns None (error handling should be improved to explicitly handle this case).

Missing from the provided code:

- **Flask routing:** The code lacks the Flask routing necessary to define an endpoint for receiving user prompts and returning the generated text. This needs to be added to make the application functional.
- **Error handling:** While the `call_gemini` function handles retries, it does not explicitly handle potential errors in the API response (beyond checking `response.ok`). More robust error handling (e.g., checking for specific error codes) should be included.
- **Input validation:** No input validation is present. The application should sanitize user inputs to prevent vulnerabilities.
- **Response formatting:** Currently, the function returns only the generated text. It would be beneficial to return a JSON response with status codes and potentially additional metadata.

The provided code is a basic foundation; significant additions are required to create a fully functional and robust application.