Karthik Project Documentation

Project Overview

This project is a simple Flask application that acts as a wrapper for the Gemini 2.0 Flash generative language model provided by Google's Generative Language API. It allows users to send prompts to the Gemini API and receive generated text responses. The application handles potential API errors with retry mechanisms and incorporates CORS for cross-origin resource sharing. The project emphasizes secure API key management using environment variables.

Installation/Setup

1. **Clone the repository:**

   ```
   git clone <repository_url>
   ```

2. **Install dependencies:**

   ```
   pip install -r requirements.txt
   ```

   (Note: A `requirements.txt` file is assumed to exist and contain `Flask`, `flask_cors`, `requests`, and `python-dotenv`.)

3. **Create a `.env` file:** Create a file named `.env` in the project's root directory. Add your Gemini API key as follows:

   ```
   GEMINI_API_KEY=YOUR_GEMINI_API_KEY
   ```

   Replace `YOUR_GEMINI_API_KEY` with your actual API key from the Google Cloud Console.

4. **Run the application:**

   ```
   python app.py
   ```

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:** Loads environment variables from a `.env` file, improving security by keeping API keys out of the main code.
- **Google Gemini 2.0 Flash API:** The core large language model used for text generation.

Feature List

- **Prompt Submission:** Allows users to submit text prompts to the Gemini API.
- **Response Handling:** Receives and returns the generated text response from Gemini.
- **Error Handling:** Includes retry logic to handle potential API errors (e.g., network issues, rate limiting).
- **Secure API Key Management:** Uses environment variables to store the Gemini API key, preventing hardcoding sensitive information.
- **CORS Support:** Enables cross-origin requests, making the API accessible from different websites or applications.

Code Architecture

The application uses a simple Flask structure. The `app.py` file contains all the application logic:

- **Initialization:** The Flask application is initialized, CORS is enabled, and the Gemini API key is loaded from the environment variables.
- **`call_gemini()` function:** This function makes the request to the Gemini API, including retry logic. It handles the JSON response and extracts the generated text.
- **Route Handling (Implicit):** While not explicitly shown, a Flask route (likely `/generate` or similar) would be defined to accept POST requests containing the user's prompt and call `call_gemini()` to get a response. This response would then be sent back to the client as a JSON object.

Usage Examples

(Assuming a `/generate` endpoint exists that accepts POST requests with a JSON payload containing the prompt):

Request (using curl):

```
curl -X POST -H "Content-Type: application/json" -d '{"prompt": "Write a short story about a talking dog."}'
http://localhost:5000/generate
```

Expected Response (JSON):

```
{
  "generated_text": "Once upon a time, in a small town nestled beside a sparkling river, lived a scruffy terrier named Pip.  Pip
wasn't just any terrier; he could talk!..."
}
```

APIs or Functions Explained

`call_gemini(prompt, max_retries=5)`:

This function is the core of the application. It takes a `prompt` string as input and an optional `max_retries` parameter (defaulting to 5). It attempts to send a POST request to the Gemini API using the `requests` library. The `try-except` block could be included (not present in given code) for more robust error handling. If the request fails, it waits before retrying up to `max_retries` times. Upon success, it parses the JSON response and returns the generated text. The exponential backoff strategy (delay increasing with each retry) isn't implemented but would be a beneficial addition. A more sophisticated error-handling mechanism might also log errors or provide more informative error messages to the user.