# Repository Documentation

# Karthik's Gemini-powered Application Documentation
## Project Overview
This project is a simple Flask application that leverages Google's Gemini 2.0 Flash large language model (LLM) via its API to generate text based on user-provided prompts. The application handles potential API errors with retry mechanisms and utilizes environment variables for secure API key management. The primary function is to provide a convenient interface for interacting with the Gemini API.
## Installation/Setup
1. \*\*Clone the Repository:\*\* Clone this GitHub repository to your local machine.
2. \*\*Create a `.env` file:\*\* Create a file named `.env` in the root directory of the project. Add your Gemini API key to this file as follows:
```
GEMINI\_API\_KEY=YOUR\_ACTUAL\_GEMINI\_API\_KEY
```
Replace `YOUR\_ACTUAL\_GEMINI\_API\_KEY` with your actual Google Gemini API key. This is crucial for the application to function correctly.
3. \*\*Install Dependencies:\*\* Install the required Python packages using pip:
```bash
pip install Flask flask\_cors requests python-dotenv
```
4. \*\*Run the Application:\*\* Execute the application using:
```bash
python app.py
```
The application will typically run on `http://127.0.0.1:5000/` (or similar).
## Tech Stack
\* \*\*Python:\*\* The primary programming language.
\* \*\*Flask:\*\* A lightweight web framework for creating 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 secure management of environment variables.
\* \*\*Google Gemini 2.0 Flash:\*\* The Large Language Model used for text generation.
## Feature List
\* \*\*Text Generation:\*\* Accepts a text prompt as input and returns text generated by the Gemini 2.0 Flash model.
\* \*\*API Key Management:\*\* Uses environment variables for secure storage of the Gemini API key.
\* \*\*Error Handling and Retries:\*\* Includes retry logic to handle potential errors during API calls.
\* \*\*JSON Response:\*\* Returns the generated text as a JSON response.
## Code Architecture
The application is a simple Flask application with a single endpoint. The `app.py` file contains the entire codebase. The core logic resides within the `call\_gemini` function:
\* \*\*`call\_gemini(prompt, max\_retries=5)`:\*\* This function takes a prompt as input and makes a POST request to the Gemini API. It includes a retry mechanism with exponential backoff to handle transient network issues or API rate limits. The function returns the generated text from Gemini.
The Flask application sets up CORS to allow cross-origin requests and handles routing to the function.
## Usage Examples
While the provided code doesn't include a user interface, you can test it using tools like `curl` or Postman. Here's an example using `curl`:
```bash
curl -X POST -H "Content-Type: application/json" -d '{"prompt": "Write a short poem about a cat."}' http://127.0.0.1:5000/
```
Replace `http://127.0.0.1:5000/` with the actual URL where your Flask application is running. The response will be a JSON object containing the generated text.
## APIs or Functions Explained
The core function is `call\_gemini`.
\*\*`call\_gemini(prompt, max\_retries=5)`:\*\*
\* \*\*`prompt` (str):\*\* The input text prompt to send to the Gemini API.
\* \*\*`max\_retries` (int, optional):\*\* The maximum number of times to retry the API call in case of errors. Defaults to 5.
The function constructs a JSON payload containing the prompt, sends a POST request to the Gemini API, and handles the response. It includes a retry mechanism with an exponential backoff strategy to improve the robustness of the API interaction. In case of failure after multiple retries, it implicitly returns an error (no text). More sophisticated error handling could be added to provide informative error messages to the user.