DOCIFY: AI-Powered Code Documentor - Documentation

Overview

DOCIFY is a Streamlit application that leverages Google's Gemini AI model to automatically generate detailed documentation for Python code. Users paste their code into a text area, and the application produces a comprehensive documentation in natural language, which can then be downloaded as a PDF or listened to as an audio file. The application incorporates error handling for empty code inputs and utilizes custom styling for improved user experience.

Code Structure

The code is organized into several key functions and uses the Streamlit framework for building the user interface.

1. **Import Statements:** Imports necessary libraries including `streamlit`, `os`, `google.generativeai`, `dotenv`, `gtts`, `io`, and `fpdf`.

2. **Environment Variable Loading:** Loads the Google Cloud API key from a `.env` file using `load_dotenv()`. This key is essential for interacting with the Gemini AI model.

3. **Page Configuration:** Configures the Streamlit page title, layout, and initial display using `st.set_page_config()`.

4. **UI Elements:** Creates the main UI elements using Streamlit widgets:
    - `st.title()` displays the application title.
    - `st.markdown()` provides introductory text and features.

- `st.text_area()` creates a text input area for the user to paste their code.
- `st.button()` creates a button to trigger the documentation generation process.

5. **`generate_documentation(code_input)` Function:**
   - Takes the user's code as input.
   - Constructs a prompt for the Gemini AI model, which includes instructions to generate detailed documentation (overview, function descriptions, code structure explanation, and usage examples).
   - Uses the `google.generativeai` library to send the prompt to the Gemini model and receive the generated documentation.
   - Returns the generated documentation as a string.

6. **`create_pdf(documentation)` Function:**
   - Takes the generated documentation text as input.
   - Creates a PDF document using the `fpdf` library.
   - Cleans the documentation text to remove emojis and unsupported characters during PDF creation. Handles potential encoding issues by encoding to `ascii` and then to `latin1` before creating the PDF.
   - Returns the PDF document as a `BytesIO` object.

7. **`text_to_speech(text)` Function:**
   - Takes the generated documentation text as input.
   - Uses the `gTTS` library to convert the text to speech.
   - Returns the audio data as a `BytesIO` object.

8. **Main Execution Block:**
   - Contains the main logic of the application.
   - Checks if the user has entered code.

- If code is present, it calls `generate_documentation()` to get the documentation, then displays the documentation.
- It calls `create_pdf()` to generate a PDF and provides a download button.
- It calls `text_to_speech()` to generate an audio file and provides a download and play button.
- Handles empty code input gracefully with a warning message.

9. **Custom Styling:** Applies custom CSS styling to the Streamlit components for enhanced visual appeal.

Function Descriptions

`generate_documentation(code_input)`: This function acts as the core of the documentation generation process. It takes the user's Python code as input and uses Google's Gemini AI to generate a detailed documentation. The prompt explicitly instructs the AI to provide an overview, function descriptions, code structure explanation, and usage examples.

`create_pdf(documentation)`: This function converts the generated documentation text into a PDF file. It uses the `fpdf` library for PDF creation and handles potential encoding issues by cleaning the text to remove emojis and unsupported characters. It returns the PDF in a memory-friendly `BytesIO` stream for downloading.

`text_to_speech(text)`: This function converts the generated documentation text into an audio file using `gTTS`. The function returns the audio data in a `BytesIO` object.

Usage Examples

1. **Run the application:** Make sure you have all required libraries installed (`streamlit`, `google-generativeai`, `python-dotenv`, `gTTS`, `fpdf`). Create a `.env` file with your Google Cloud API key (named `GOOGLE_API_KEY`). Then run the script using `streamlit run your_script_name.py`.

2. **Paste Code:** Paste your Python code into the text area.

3. **Generate Documentation:** Click the "Generate Documentation" button.

4. **Download PDF & Audio:** Download the generated documentation as a PDF and/or an MP3 audio file.

Error Handling

The application includes a check for empty code input. If the user clicks the "Generate Documentation" button without entering any code, a warning message is displayed. The `create_pdf` function handles potential encoding issues by removing unsupported characters.

Dependencies

- `streamlit`
- `google-generativeai`
- `python-dotenv`
- `gTTS`

- `fpdf`

Setup Instructions

1. **Install Libraries:**

   ```bash
   pip install streamlit google-generativeai python-dotenv gTTS fpdf
   ```

2. **Obtain Google Cloud API Key:** Follow the instructions on the Google Cloud Platform to create a project, enable the Generative AI API, and obtain an API key.

3. **Create .env file:** Create a `.env` file in the same directory as your script and add your API key:

   ```
   GOOGLE_API_KEY=your_api_key_here
   ```

4. **Run the application:** Run the script using `streamlit run your_script_name.py`.