Full-Stack Technical Assignment: Chatbot Widget with Optional RAG

1. Description

We are looking for a full-stack developer to create a functional chatbot widget that sits at the bottom-right of a webpage. The chatbot should be able to engage with users in a conversational manner. The core functionality involves integrating Google's Gemini API to generate responses, while the ability to retrieve and reference a predefined database of books is a bonus task.

The task will test the candidate's ability to:

- Develop a clean, responsive frontend.
- Implement a simple backend to integrate the Google Gemini API (free-tier).
- Design a chatbot interface that provides useful responses based on user inputs.

Bonus: The candidate may also implement Retrieval-Augmented Generation (RAG) to enhance the chatbot's responses by retrieving relevant data from a sample OWASP Q&A database and incorporating that data into its answers.

---

2. Evaluation Criteria

The evaluation will focus on the following areas:

1. **Frontend Development:**
    - **UI/UX:** Design a simple, intuitive UI for the chatbot that fits seamlessly into the webpage.
    - **Interactivity:** The chatbot widget should open, close, and allow users to interact with it in a responsive manner.
    - **UI Components:** Display messages in a chat-bubble format, handle user inputs effectively, and show chatbot responses.
2. **Backend Development:**
    - **API Integration:** Proper use of Google's Gemini API (free-tier) to generate responses.
    - **Session Management:** Store user inputs and maintain a simple context for interactions during the session.
3. **Full-Stack Integration:**
    - Ensure smooth interaction between frontend and backend, leveraging a clean API architecture.

- Implement the chatbot logic that makes API calls to Gemini for generating responses.

4. **Prompt Engineering:**
    - Craft effective prompts for the Gemini API to generate meaningful and coherent responses.
    - Provide helpful, informative answers to general queries (e.g., weather, trivia, basic Q&A).
5. **Optional RAG Component (Bonus):**
    - Implement a mechanism to retrieve relevant information from a fixed OWASP Q&A database (CSV file or database) and augment the chatbot's responses based on user queries related to OWASP answers.
6. **Performance & Reliability:**
    - Ensure the chatbot is responsive and performs well even with multiple users interacting with it.
7. **Security:**
    - Secure the API key (e.g., environment variables) and validate/sanitize user input.

---

3. Task

1. **Frontend:**
    - Create a floating chatbot widget that appears at the bottom-right of the webpage.
    - The widget should have a "Chat" button to open and close the chat interface.
    - The chatbot UI should include a message display area, a text input field, and a send button.
    - The chatbot should be able to handle multiple user inputs and display them in a conversational format.
    - Use basic animations (e.g., smooth opening/closing of the widget, sending/receiving messages).
2. **Backend:**

- Set up a server (using a backend framework such as Node.js, Django, Flask, etc.) to handle API requests.
- Integrate Google's Gemini API (free-tier) for generating chatbot responses.
- The backend should accept user input, send the query to the Gemini API, and return the response to the frontend.
- Implement session management to store user interactions and maintain a simple conversational context.

3. **Optional: RAG Implementation (Bonus):**
   - Implement a search mechanism to query a fixed OWASP Q&A database (CSV file or a small database).
   - Use this data to retrieve relevant security questions related to OWASP and generate a meaningful and accurate answer.
   - Augment the chatbot's responses with the OWASP data by sending relevant information retrieved from the database alongside the API-generated response from Google's Gemini API.
   - Craft prompts that incorporate data from the OWASP Q&A database into the responses (e.g., "Can you recommend 3 measures to prevent against XSS-attacks?" or "Tell me more about 'Broken Object Level Authorization (BOLA)'").
4. **Deployment:**
   - Ensure the application is production-ready with proper error handling and logging.

     Option 1
   - Deploy your application on any cloud service (AWS, Heroku, Google Cloud, etc.) so that it's publicly accessible.

     Option 2
   - Deploy your application as an Virtual Machine in the form of VMDK format.

     Option 3
   - Deploy your application as a Docker/Container.

---

4. Requirements

Frontend:

- **Languages/Frameworks:** HTML, CSS, JavaScript (or a frontend framework such as React, Vue, etc.)
- **Features:** Responsive design (works well on both desktop and mobile), smooth animations for chatbot interactions.
- **UI:** Simple chat interface with chat bubbles displaying both user and chatbot messages.
- **Interactivity:** Open/close functionality for the widget, input handling, and message display.

Backend:

- **Server:** Node.js/Express, Python Flask, Django, or any other backend framework.
- **Gemini API:** Integration with Google's Gemini API (free-tier). OpenAI API is also acceptable.
- **Session Management:** Store user inputs and session data (e.g., using in-memory storage or a simple database).

RAG Component (Bonus):

- **Database:** Use the fixed csv of OWASP question and answers.
- **Search:** Implement basic search logic that retrieves relevant question and answer data when users ask about certain OWASP topics.
- **Augmented Responses:** Integrate the question and answer data with Gemini-generated responses to provide more detailed, context-specific answers.

Security:

- **API Keys:** Store Gemini API keys securely (e.g., using environment variables).

Deployment:

- **Deployment Platform:** AWS, Heroku, Google Cloud, or another cloud provider. Docker or VMDK is also acceptable.
- **Production-Ready:** Ensure the chatbot is robust, secure, and can handle user interactions smoothly.

---

5. Submission Guidelines

1. **Code:**

- Provide all source code in a public GitHub repository or any other code-sharing platform.
- Organize the code into appropriate folders for frontend and backend.
- Include a README.md file with instructions on how to run the application locally, how to deploy it, and any other relevant information (e.g., how to set up the Gemini API).

2. **Deployment:**
   - Provide a live link to your deployed chatbot application.
   - Ensure the application is publicly accessible and functions as expected.
3. **Documentation:**
   - Write brief documentation explaining your approach to solving the task.
   - If you implemented the RAG component, provide details on how the search mechanism works and how the book data is integrated into the chatbot's responses.
4. **Testing:**
   - Include unit tests for both frontend and backend code (optional but encouraged).
   - Optionally, provide basic performance benchmarks (e.g., response times, load handling).

Bonus Task: RAG Implementation (Optional)

- **Objective:** Implement Retrieval-Augmented Generation (RAG) by integrating a fixed OWASP question and answer database with the chatbot's response mechanism. The RAG feature is optional and will not be required for completing the core assignment but will provide an opportunity to showcase advanced skills in combining retrieval systems with language models like Google's Gemini API.

Timeline:

- **Time Allocation:** The task should take approximately 6-8 hours to complete, depending on the complexity of the RAG implementation and the polish of the user interface. The RAG component is optional but will add value to the final submission.

Good luck with the assignment! We're excited to see how you implement a full stack chatbot with or without the RAG enhancement.