Use Case Document: AI-Powered Chatbot Web Application
Position: Full-Stack Python Developer
Time Limit: 5 Days

Objective

Develop a **minimum viable product (MVP)** for a web application where users interact with an AI-powered chatbot designed to assist with product/service inquiries (e.g., tech support, e-commerce FAQs). The MVP must include a functional chat interface, real-time interaction, and AI integration. Candidate is encouraged to add bonus features (e.g., GCP services, chat history, advanced UI/UX) if time permits.

You are open to choose your technology as per your skills and experience. Your focus should be on demonstrating your coding and integration ability with understanding on latest tools and technology. Below tasks can be taken as reference guidelines to achieve MVP requirements.

1. Frontend

- **Chat Interface**:
    - Real-time display of messages between the user and chatbot.
    - Input field for users to type questions.
    - Visual feedback (e.g., "Bot is typing..." animation) during AI processing.
    - Responsive design (works on both desktop and mobile).
- **Session Management**:
    - Maintain conversation context **within a single session** (no login required).
    - Messages persist until the user refreshes or closes the browser.

2. Backend

- **AI Integration**:
    - Use can use RASA, OpenAI's GPT-3.5/4 or Google's Vertex AI/ Dialogflow or any open-source tools or packages to generate responses.
    - Ensure the bot understands context (e.g., references prior messages in the same session).

- **API/WebSocket Design**:
    - Create an endpoint to send user messages and receive AI responses (e.g., /chat).
    - Use **REST API** or **WebSocket** for real-time communication.
- **Basic Error Handling**:
    - Gracefully handle cases where the AI service is unavailable (e.g., display error messages).

Bonus Points (Optional Enhancements)

Below tasks are not mandatory, but it will give an added advantage if you can demonstrate your skills beyond MVP implementation.

- **Database Integration**:
    - Store chat history (e.g., using Firestore, SQLite, or PostgreSQL).
- **GCP Services**:
    - Deploy the app on GCP App Engine or Cloud Run.
    - Use Dialogflow ES/CX for intent-based chatbot logic.
    - Implement caching with Memorystore (Redis).
- **Advanced Features**:
    - Allow users to download/export chat history.
    - Add multilingual support (e.g., detect language and respond accordingly).
    - Implement rate limiting to prevent API abuse.
- **UI/UX Enhancements**:
    - Add emoji reactions, file uploads, or voice-to-text input.
    - Use a frontend framework like React or Vue.js for a polished interface.

Technolgy Specifications

Frontend

- Use HTML/CSS/JS or a modern framework (React/Vue.js/Angular).
- Avoid page reloads during chat interactions (use AJAX or Websockets).

Backend

- Use Python with a framework like Flask or FastAPI or Django (optional).
- Use environment variables for sensitive data (e.g., API keys).

Deliverables

1. **Source Code**:
    a. A GitHub/GitLab repository with:
        i. Frontend and backend code in separate directories.
        ii. requirements.txt or for dependencies.
        iii. Basic unit tests (e.g., testing API endpoints(optional).
2. **Documentation**:
    a. README.md with:
        i. Setup instructions (installation, API key configuration).
        ii. Explanation of technical choices (e.g., why Flask over Fast API?).
        iii. Notes on limitations and future improvements.
3. **Demo or Slides**:
    a. A demo **OR** a screen recording showing:
        i. User input → AI response flow.
        ii. Handling of edge cases (e.g., slow AI response).

Evaluation Criteria

MVP (80% Weight)

1. **Functional Chat Interface**:
    a. Smooth real-time interaction with no page reloads.
    b. AI responses are relevant and context-aware.
2. **Code Quality**:
    a. Clean, modular code with proper separation of concerns (e.g., routes, services).
    b. Error handling for API failures or invalid inputs.
3. **Basic Security**:
    a. Sanitized inputs and secure handling of API keys.

Bonus (20% Weight)

1. **GCP Integration**:
    a. Effective use of at least **one GCP service** (e.g., Firestore, App Engine).
2. **Advanced Features**:
    a. Implementation of chat history, rate limiting, or multilingual support.
3. **UI/UX Polish**:

a. Professional design with animations or mobile responsiveness.

Example User Scenarios

1. **User asks a question**:
    a. **User**: “How do I reset my password?”
    b. **Bot**: Provides step-by-step instructions (generated by AI).
2. **Follow-up question**:
    a. **User**: “What if I don’t receive the reset email?”
    b. **Bot**: References the prior question and suggests checking spam folders.

Timeline Suggestions

- **Day 1-2**: Setup backend (API/Websocket), integrate AI model.
- **Day 3**: Build frontend interface and connect to backend.
- **Day 4**: Implement error handling, tests, and basic documentation.
- **Day 5**: Add bonus features (if time permits) and finalize deployment.

Notes for Candidates

- **Focus on the MVP first**. A simple but functional app is better than an incomplete "feature-rich" one.
- **Document your assumptions** (e.g., “Session data is stored in memory, not a database”).
- **Use open-source libraries** (e.g., python-dotenv for environment variables, Flask-CORS for cross-origin requests).
- **Reach out** if requirements are unclear.