Below is a simplified project documentation for the **Yash AI Chatbot**, tailored for interview preparation. This version focuses on key aspects that highlight your understanding of the project, its purpose, and your contributions, while keeping it concise and easy to present during an interview. It avoids overly technical details but still showcases the project's structure, deployment, and potential talking points for an interview setting.

**Yash AI Chatbot - Simplified Project Documentation for Interview**

**1. Project Overview**

**1.1 What is Yash AI Chatbot?**

The **Yash AI Chatbot** is a web-based conversational AI application I developed to provide real-time responses to user queries. It uses **Google Gemini API** for natural language processing and **Streamlit** for the user interface. The chatbot supports interactive features like chat history and streaming responses, making it user-friendly and dynamic. I deployed it on **AWS** and it's currently running at http://3.83.87.37:8502/.

**1.2 Purpose and Goals**

* Build a conversational AI tool for real-time interaction.
* Create an intuitive UI with features like chat history and dark mode.
* Deploy the app on AWS to demonstrate cloud deployment skills.
* Learn and apply technologies like Streamlit, Google Gemini API, and Docker.

**1.3 My Role**

I was responsible for the end-to-end development of the project, including:

* Designing the architecture and selecting technologies.
* Writing the backend logic to integrate with the Gemini API.
* Developing the frontend UI with Streamlit and custom CSS/JavaScript.
* Containerizing the app with Docker and deploying it on AWS.

**2. Key Features**

* **Conversational AI**: Powered by Google Gemini (gemini-1.5-pro) for answering queries.
* **Interactive UI**: Built with Streamlit, with custom styling and JavaScript for a better user experience.
* **Chat History**: Users can view past conversations within a session.
* **Streaming Responses**: Responses are delivered in real-time for a smooth experience.
* **Responsive Design**: Works on both desktop and mobile, with dark mode support.
* **Secure Deployment**: Hosted on AWS with Docker for scalability.

**3. Technologies Used**

* **Frontend**: Streamlit, HTML/CSS (custom styling in styles.css), JavaScript (script.js for interactivity).
* **Backend**: Python, Google Gemini API for AI responses.
* **Deployment**: Docker, AWS EC2 (deployed at http://3.83.87.37:8502/).
* **Other Tools**: python-dotenv for secure API key handling, logging for debugging.

**4. Project Structure**

text

CollapseWrapCopy

Yash\_AI\_Chatbot/

├── app.py # Main application file (Streamlit app)

├── chatbot.py # Handles Gemini API integration

├── config.py # Manages environment variables (e.g., API key)

├── static/

│ ├── script.js # Adds client-side interactivity (e.g., sending messages)

│ └── styles.css # Custom styles for UI (responsive design, dark mode)

├── templates/

│ └── components.py # Reusable UI components (e.g., chat header, history box)

├── Dockerfile # Docker configuration for deployment

├── requirements.txt # Lists dependencies (Streamlit, Google Gemini, etc.)

├── .gitignore # Ignores sensitive files (e.g., .env)

└── LICENSE # GNU General Public License v3.0

**5. How It Works**

1. **User Interaction**: Users type a query (e.g., "program for factorial java") in the chat interface.
2. **Backend Processing**: The query is sent to the Google Gemini API via chatbot.py, and the response is streamed back.
3. **Frontend Display**: The response is shown in the UI, and the conversation is saved in the chat history.
4. **Deployment**: The app runs on AWS EC2, accessible via a public IP on port 8502.

**6. Deployment on AWS**

* I containerized the app using Docker (Dockerfile) to ensure consistent deployment.
* Pushed the Docker image to **Amazon ECR** (Elastic Container Registry).
* Launched an **AWS EC2** instance, installed Docker, and ran the container on port 8502.
* Configured security groups to allow public access on port 8502.
* The app is live at http://3.83.87.37:8502/.

**7. Challenges Faced**

* **API Integration**: Setting up the Google Gemini API and handling streaming responses was tricky. I used logging to debug issues.
* **AWS Deployment**: Configuring security groups and port mapping on EC2 took some trial and error.
* **UI Customization**: Streamlit has limited styling options, so I added custom CSS and JavaScript to improve the design.
* **License Conflict**: The README mentioned MIT License, but the LICENSE file was GPL-3.0. I standardized it to GPL-3.0.

**8. What I Learned**

* How to integrate AI APIs (Google Gemini) with a web application.
* Building and styling a web app with Streamlit, CSS, and JavaScript.
* Containerizing an app with Docker and deploying it on AWS.
* Managing environment variables securely and debugging with logs.
* Importance of consistent licensing in open-source projects.

**9. Future Improvements**

* Add a database (e.g., AWS RDS) to store chat history permanently.
* Implement the dark mode toggle feature (currently styled but not functional).
* Add support for images or voice input using Gemini Vision models.
* Secure the app with HTTPS using AWS Certificate Manager.
* Scale the app with AWS Auto Scaling for higher traffic.

**10. Why This Project Matters**

This project showcases my ability to:

* Build a full-stack AI application from scratch.
* Work with modern technologies like Streamlit, Docker, and AWS.
* Solve real-world problems like API integration and cloud deployment.
* Create a user-friendly and scalable product.

**Interview Preparation Tips**

1. **Key Points to Highlight**:
   * Explain your role in the project and the technologies you used.
   * Mention the AWS deployment process to show cloud experience.
   * Discuss challenges (e.g., API integration, UI customization) and how you solved them.
   * Emphasize what you learned (e.g., Docker, secure deployment, debugging).
2. **Potential Questions and Answers**:
   * **Q: Why did you choose Streamlit?**  
     A: Streamlit is beginner-friendly for building web apps quickly, especially for Python developers. It allowed me to focus on the AI integration while providing a decent UI out of the box.
   * **Q: How did you deploy on AWS?**  
     A: I used Docker to containerize the app, pushed the image to Amazon ECR, and ran it on an EC2 instance. I configured the security group to allow traffic on port 8502.
   * **Q: What challenges did you face?**  
     A: One challenge was integrating the Gemini API with streaming responses. I used logging to debug issues and ensured the response was displayed smoothly in the UI.
   * **Q: How would you improve the project?**  
     A: I’d add persistent storage for chat history, implement the dark mode toggle, and secure the app with HTTPS using AWS Certificate Manager.
3. **Demo Preparation**:
   * If possible, show the live app at http://3.83.87.37:8502/ during the interview.
   * Demonstrate a simple query (e.g., "program for factorial java") and explain the response.
   * Point out the chat history feature and the responsive design.