Design Document for Artisan Chatbot

Jackson Qi | January 30, 2024

This project implements a chatbot feature, "Ava," designed to simulate a conversation between users and a virtual assistant. The chatbot supports sending, editing, and deleting messages.

It uses PostgreSQL (via Supabase) for data persistence, FastAPI for the backend API, and React with TypeScript for the frontend. We will have a server side (backend) and a client side (frontend). This setup enhances security by mitigating the risk of exposing sensitive information, such as the database secret key and gemini key, on the client side.

React, TypeScript	- Strongly typed language ensures fewer runtime errors - Optimizes rendering, updating only parts of the UI that have changed instead of re-rendering the whole page
FastAPI, Python	- FastAPI for its speed of development - Python's simplicity allows quick prototype - Horizontal scaling will allow to handle more traffic
PostgreSQL (via Supabase)	- SQL > NoSQL because we are storing structured data (message content, timestamp, owner of message); every message object follows the same structure. - ACID compliant, important because we want to ensure reliability in chat history and data persistence. - Supports order by when we query so we can display messages in order of created_at timestamp.

Core Features

1. Send a message: Users can send messages to the chatbot.
2. Chatbot response: Ava replies using Google Generative AI (gemini-pro).
3. Edit/Delete messages: Users can modify or remove their own messages from the chat.

Additional Features

1. Typing indicator: Ava simulates typing with a random delay (3–6 seconds), so it doesn't feel automated because of instant replies. Ava mimics how a human might take time to type a response.
2. Persistent storage: Messages are stored in PostgreSQL via Supabase for retrieval when reopening after closing the chat app.

Database Schema

id (UUID)	content (TEXT)	is_user (BOOLEAN)	created_at (TIMESTAMP)

1. The **id** will be the unique identifier for each message, and will be helpful when we update and delete, as we could delete and update by the message **id**.
2. **content** will be the message text.
3. **is_user** will be a flag used to determine where to put the text. If the message object is flagged as a user, then it will be placed on the right side, and if it's not a user (Ava), then it will be placed on the left side of the chat in the UI.
4. **created_at** is when the message was sent and inserted into the database, and will be used as a timestamp of the message in the chat.

Backend API endpoints

Method	Endpoint	Notes
GET	/api/messages	Fetches all the messages from the database of messages
POST	/api/messages	Adds a new message to the database of messages
PUT	/api/messages/{id}	Edit the content of an existing message in the database by id
DELETE	/api/messages/{id}	Delete an existing message in the database by id

AI Model

Model	Pros	Cons
Gemini	- Free tier available (15 RPM limit).	- System Prompt requires Paid Developer Status
OpenAI	- GPT-4 excels in reasoning and creativity	- costly - No Free Tier
Claude	- strong at reasoning tasks	- costly

Gemini was chosen because of cost efficiency; the free tier (15 RPM) makes it ideal for early-stage projects without incurring costs. I also have experience using Gemini. So, Ava's responses are generated using Google Generative AI (Gemini model). The prompt includes context about Ava's role and the user's input.

The prompt we will be tuning Ava is the following:
- You are Ava, an assistant onboarding as a Business Development Representative for StockDash, which delivers fresh produce to restaurants in bulk.
- Your role is to engage restaurant owners, understand their needs, and show how the company can streamline their operations.

- In this chat, you are speaking with your boss (Jackson) during onboarding.
- Ask thoughtful questions about the company mission, goals, target customers, and your responsibilities.

- Be enthusiastic, professional, and concise.
- Remember, you are replying as Ava, so double check your response before sending just incase it doesn't make sense.
- Keep responses concise (2-3 sentences).
IMPORTANT: sound normal and as human as possible, just like how a person would talk. response without starting with 'Ava:'

Frontend Components

Chat.tsx	1. Manages the entire chat state (contains all the messages) ordered by timestamp 2. Handles API calls to fetch, send, edit, and delete messages.
Message.tsx	1. Displays individual messages with options to edit or delete.

4. Key Test Cases to Implement
Category Test Cases
Backend API - Empty message prevention
- Gemini API failure handling
- DB connection errors
Frontend UI - Message persistence after refresh
- Edit/delete validation
- Error toasts
Edge Cases - Long messages (5000+ chars)
- Special characters
- High concurrency

Future Areas of Improvements

1. **User Login**
    a. Allow users to log in so each user has their own chat history.
    b. Personalized chatbot experience
2. **Contextual System Prompt**
    a. Fine-tune the model by including previous messages in the system prompt.
    b. Improves conversational accuracy and yield better responses from Ava
3. **Reply Suggestions**
    a. Provide users with AI-generated reply suggestions based on Ava's response.
    b. Reduces typing effort and guides users toward meaningful interactions.
4. **Real-Time Updates**
    a. Use WebSockets for real-time message updates instead of polling.
    b. Enhances responsiveness and reduces server load by eliminating frequent API calls.