AI Customer Support Bot - Project Report

Author: Samaksh Sethiya
Registration Number: 22BCE10187

---

1. What This Project Is About

1.1. The Problem

In today's fast-paced world, customers expect quick, almost instant, answers to their questions. However, many businesses find their support teams spending a significant portion of their day answering the same basic questions over and over. This includes common queries like, "How do I reset my password?", "What are the shipping options?", or "Where can I find my order status?". This repetitive work not only takes up valuable time that skilled support agents could be spending on more complex, high-priority problems, but it can also lead to frustrating delays for customers who are left waiting for simple information.

1.2. The Goal

The main goal of this project was to directly solve this problem by building a smart, automated chatbot that can act as the first line of customer support. The idea was to create an intelligent assistant that could instantly handle these common questions 24/7, freeing up human agents to focus their expertise on more difficult and unique customer issues.
The key things this project successfully accomplished are:

- **Answering Questions:** The bot is designed to understand what a user is asking and provide an accurate and helpful answer by drawing from a pre-written list of Frequently Asked Questions (FAQs).
- **Remembering the Conversation:** The bot keeps track of the current conversation, which allows it to understand context and handle follow-up questions naturally.

- **Knowing When to Ask for Help:** A crucial feature for building user trust is that the bot knows its own limits. If a user asks a question that isn't in its knowledge base, it politely informs the user that it will pass the question to a human team member.
- **Easy to Use:** The project includes a simple, clean, and modern chat window that is intuitive and easy for anyone to use without instructions.

---

2. Project Links

2.1. Demo Video

A full demonstration of the chatbot's features and functionality can be viewed at the following link: Watch the Demo Video

2.2. GitHub Repository

The complete source code for this project is available on GitHub. You can access it here: View on GitHub

---

3. How It's Built (The Big Picture)

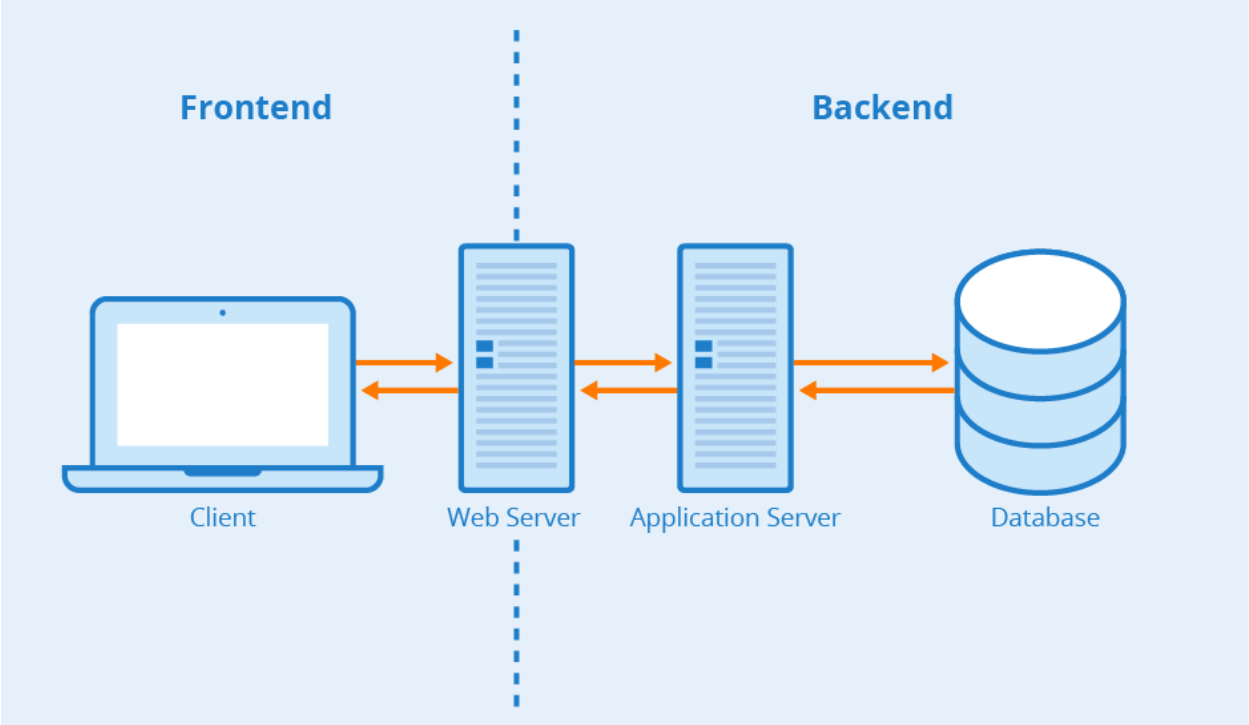
The project is split into two main parts that work together seamlessly: a **Frontend** (what you see and interact with on your screen) and a **Backend** (the hidden "brain" that does all the thinking).

- **Backend (The Brain):** The backend is the powerful engine of the project, built with the **Python** programming language and a modern, high-performance framework called **FastAPI**.
- **Frontend (The Interface):** This is the visual part of the project that you see and use in your web browser, built with **HTML**, **CSS**, and **JavaScript**.
- **Database (The Memory):** To remember conversations, the project uses a lightweight

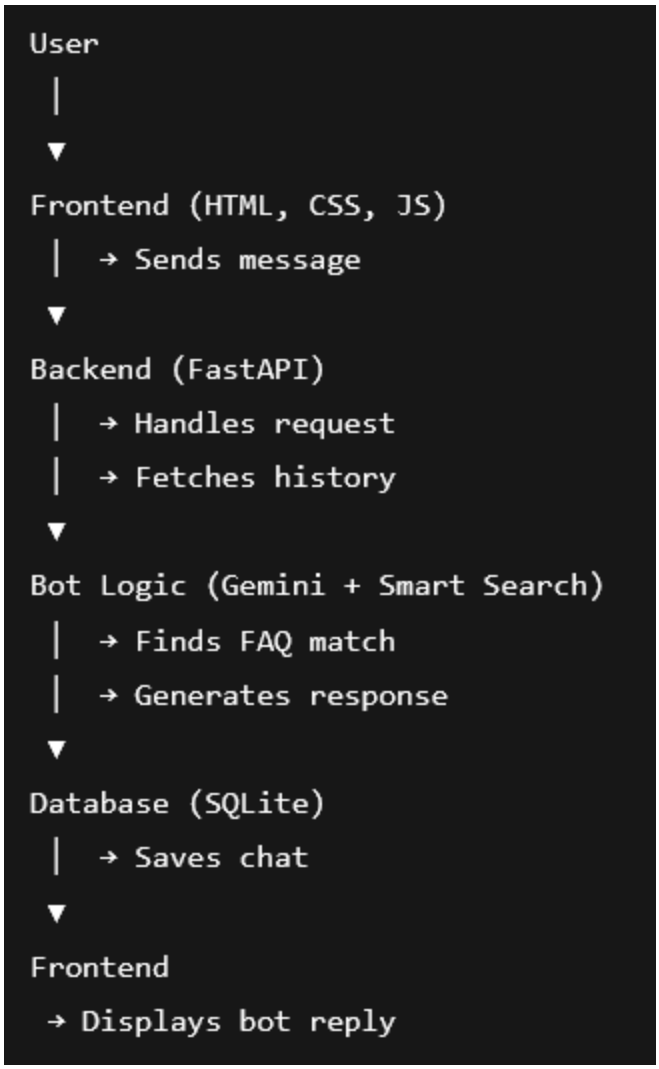
and simple file-based database called **SQLite** with **SQLAlchemy**.

- **Artificial Intelligence (AI):** The bot's ability to understand and generate human-like text comes from **Google's Gemini** model, connected via the **LangChain** library.

- **Smart Search:** This project uses **SentenceTransformers** to understand the *meaning* and *intent* behind the words, allowing it to find the correct FAQ even if the question is phrased differently.



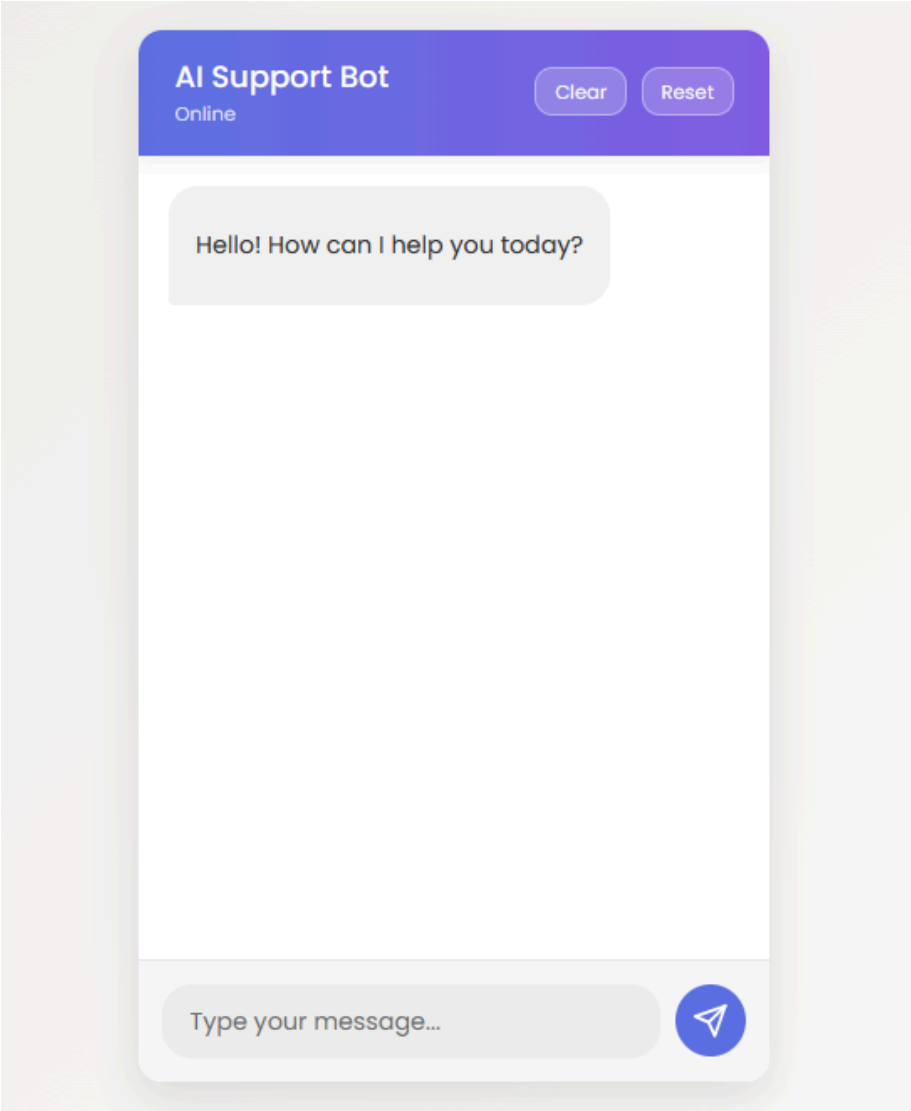
3.1. Flow of the Project



---

4. The User Interface (What You See)

The frontend creates the visual chat experience using three files.



4.1. index.html (The Skeleton)

This file lays out the structure of the chat window, including the header, the message box, and the input form.

```
<div class="chat-container">
    <div class="chat-header">...</div>
    <div class="chat-box" id="chat-box"></div>
    <form class="chat-input-form" id="chat-form">
```

```
        <input type="text" id="user-input" ... >
        <button type="submit" id="send-btn">...</button>
    </form>
</div>
```

4.2. style.css (The Designer)

This file contains the CSS code that controls all the visual styling, from colors and fonts to the layout of the chat bubbles.

4.3. script.js (The Worker)

This JavaScript file makes the chat interactive. Its most important job is to send the user's message to the backend using a fetch request and display the response.

```
// Send the message to the backend API
const response = await fetch(CHAT_API_URL, {
    method: 'POST',
    headers: { 'Content-Type': 'application/json' },
    body: JSON.stringify({
        session_id: sessionId,
        message: userMessage,
    }),
});
```

---

5. The Brain of the Project (The Backend)

The backend is where all the important decisions are made.

5.1. main.py (The Control Center)

This file uses FastAPI to create the web server and the API "endpoints" (URLs) that the frontend communicates with. The main endpoint is /chat.

```
@app.post("/chat", tags=["Chat"])
def chat_endpoint(request: ChatMessage):
    # 1. Save the user's message
    save_message(...)
    # 2. Get a response from the bot
    bot_response = chatbot_instance.get_response(...)
    # 3. Save the bot's response
    save_message(...)
    # 4. Return the response to the frontend
    return {"response": bot_response}
```

5.2. database.py (The Librarian)

This file manages all interactions with the SQLite database, providing simple functions like save_message and get_conversation_history.

5.3. models.py (The Blueprints)

This file defines the data structures using Pydantic and SQLAlchemy to ensure data is consistent throughout the application.

5.4. bot.py (The Core AI Logic)

This is the smartest part of the project. The get_response function orchestrates the bot's thinking process.

1. **Simple Talk Check:** It first checks for simple greetings to give a quick, predefined answer.

   ```
   normalized_message = user_message.lower().strip("?!., ")
   ```

```
greetings = ["hello", "hi", "hii", "hey"]
if any(greeting in normalized_message for greeting in greetings):
     return "Hello! How can I assist you today?"
```

2. **Smart Search:** If it's a real question, it uses the _find_relevant_faq function to find the most similar FAQ based on meaning.
3. **Preparing Instructions for the AI:** It gathers the chat history and the best FAQ and inserts them into a detailed PROMPT_TEMPLATE.
4. **Asking the AI:** It sends this complete prompt to the Gemini API to get a well-written, relevant, and natural-sounding answer.
5. **Final Check:** It checks the AI's response for an escalation phrase. If found, it provides the standard escalation message; otherwise, it returns the helpful answer.

---

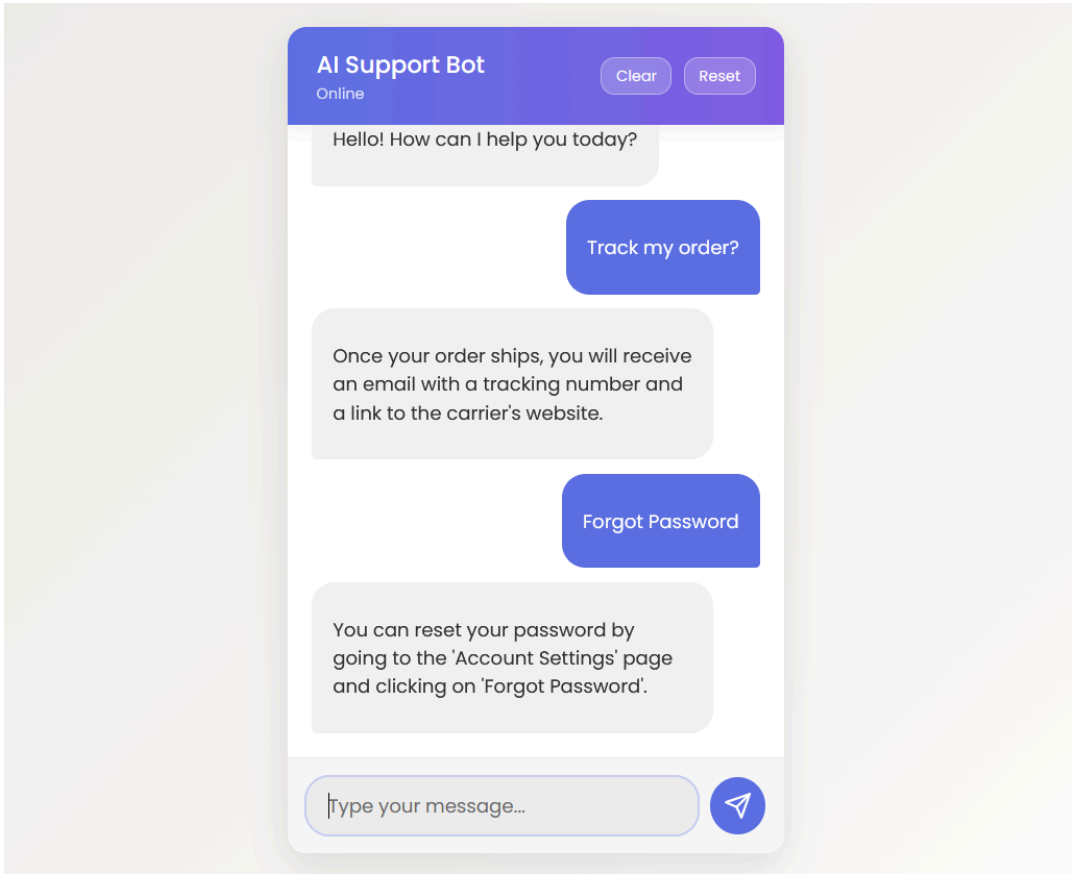
6. Final Thoughts and What's Next

6.1. In Conclusion

This project successfully created a smart, useful, and user-friendly AI Customer Support Bot. It provides a strong and practical starting point for what could easily be developed into a real-world customer service tool.

6.2. Ideas for the Future

- **Smarter Escalation:** Use the AI to summarize the chat for human agents during escalation.
- **Bigger Brain:** Connect the bot to a larger knowledge base like user manuals or articles.
- **User Accounts:** Add a login system for users to see their chat history across devices.
- **Go Live (Deployment):** Deploy the application to a cloud server to make it publicly available.



Thank you

You're welcome! Is there anything else I can help with?