**Mini Agent Chat Project**

**Idea I Chose and Why**

The idea I decided to build was a **chat application with a simple AI agent**. The backend provides an API where a user can send a message, and the system returns both the stored interaction ID and the agent’s reply. The frontend then displays the conversation history, showing the user input alongside the model response.

I chose this idea because I wanted something practical, easy to test, and visually clear. A chat interface is a natural way to interact with an AI model, and it shows the full pipeline: backend logic, API integration, data storage, and a frontend user interface. I also liked that it ties together multiple skills (FastAPI, databases, Next.js frontend) into one cohesive mini project.

**Tools I Used (and Why)**

1. **FastAPI** – I picked FastAPI for the backend because it is lightweight, easy to set up, and has automatic documentation (Swagger UI). It allowed me to quickly define endpoints like /chat and /interactions and handle JSON input/output without extra boilerplate.
2. **PostgreSQL (with SQLAlchemy)** – I used Postgres to persist interactions (user inputs and model responses). SQLAlchemy made it simple to define the interactions table and query it in Python code.
3. **Next.js (React framework)** – For the frontend, I chose Next.js because it supports modern React patterns and is straightforward to set up. It let me create a minimal chat UI that could call the backend API and update dynamically.
4. **Axios** – I used Axios in the frontend for HTTP requests. It’s easier to work with than the built-in fetch, especially for handling JSON and errors.
5. **Environment Variables (.env.local)** – To avoid hardcoding the API URL in multiple places, I used environment variables. This way, I can point the frontend to different backends (local or deployed) without changing the code.

**Time Spent and What I’d Do With More Time**

* **Time spent**: I spent a few hours setting up the backend API (defining routes, database models, and testing with Postman), and another few hours creating the frontend chat UI with Next.js. In total, it took me roughly **8–10 hours** spread across a couple of days.
* **With more time, I’d improve**:
  + Add **real LLM integration** (instead of a stubbed reply when no model is available).
  + Support **multiple users and sessions**, with authentication.
  + Deploy the backend (FastAPI + Postgres) and frontend (Next.js) to cloud hosting, so it’s accessible outside localhost.