AI Chat Agent – Local Setup Guide

This guide will help you run the AI-powered Customer Service Chat Agent locally using a **FastAPI** backend and a **React** frontend.

Project Structure

Nullaxis/

```
│   ├── main.py            # FastAPI app
│   ├── classify.py        # Intent classification using Gemini
│   ├── handlers.py        # Handlers for various intents
│   ├── rag.py             # RAG module using LangChain + ChromaDB
│   ├── utils.py           # Utility functions for logging, etc.
│   ├── requirements.txt   # Backend Python dependencies
│   └── .env               # Contains GOOGLE_API_KEY
├── frontend/
│   ├── App.js             # React frontend app
│   ├── App.css            # Styles
│   ├── index.js           # React entry point
│   └── package.json       # Frontend dependencies
└── rosy-hope-*.json       # (Optional) GCP service account for Gemini
```

1. Backend Setup (FastAPI)

Requirements

- Python 3.8+
- pip

- `fastapi`, `uvicorn`

Environment Variable

Create a `.env` file inside the `Nullaxis/` directory:

GOOGLE_API_KEY=your_google_gemini_api_key

Replace `your_google_gemini_api_key` with your actual Google Gemini API key.

Install Dependencies

cd Nullaxis

pip install -r requirements.txt

Run the Backend Server

uvicorn main:app --reload

The backend will be available at: `http://localhost:8000/chat`

2. Frontend Setup (React)

Requirements

- Node.js (v16+ recommended)
- npm or yarn

Install and Start Frontend

cd frontend

npm install

npm start

The React app will run at: `http://localhost:3000`
It connects to the backend at: `http://localhost:8000/chat`

Component Descriptions

File/Module	Description
`main.py`	FastAPI app with `/chat` POST route
`classify.py`	Uses Gemini API to classify user intent via `classify_intent(message)`
`handlers.py`	Contains intent handler functions like `handle_technical()`
`rag.py`	LangChain-powered Retrieval-Augmented Generation with ChromaDB
`utils.py`	Utility functions such as `log_interaction(message, intent, response)`
`App.js`	React-based chat UI, handles user input and connects to the backend
`rosy-hope-*.json`	*(Optional)* GCP service account credentials for LangChain integrations

Notes

- Ensure all the key files (`classify.py`, `handlers.py`, `utils.py`) are created with the respective function stubs.
- The RAG setup uses **LangChain**, **ChromaDB**, and **sentence-transformers**.
- `chat_log.json` will store logs of chat interactions.

Troubleshooting

- Double-check that `.env` is in the `backend/` directory.
- If CORS errors occur, enable CORS in `main.py`.
- If using RAG, ensure the service account `.json` file is correctly configured and accessible.