Project Report — Hyper-Personalized Customer Support AI

1. Introduction

Modern customers expect instant, context-aware support. Traditional chatbots fail because they:

- Give generic responses
- Cannot personalize based on user history
- Cannot understand live store data
- Cannot integrate real-world context such as location

This project solves that gap by building a **Hyper-Personalized Customer Support AI** designed for retail and café environments.

The system provides:

- Personalized responses based on user profile
- Live store information (inventory, offers, opening status)
- Location-aware suggestions
- Secure PII-masked LLM integration
- Retrieval-Augmented Generation (RAG) for deeper personalization

The platform serves as an intelligent store assistant capable of elevating customer experience while reducing human support load.

---

2. Directory Structure

```
hackathon_project/
│
├── __pycache__/              # Cached Python bytecode
│
├── CE/                       # Core AI Engine (Modular Backend Logic)
│   ├── __init__.py
│   ├── llm.py                # LLM inference (Gemini)
│   ├── loaders.py            # User/store data loaders
│   ├── main.py               # Core pipeline orchestration
│   ├── masking.py            # PII masking utilities
│   ├── rag.py                # RAG pipeline (chunking, embeddings)
│   ├── vectorstore.py        # FAISS vector DB for retrieval
│   └── app/                  # Auxiliary modules (optional)
```

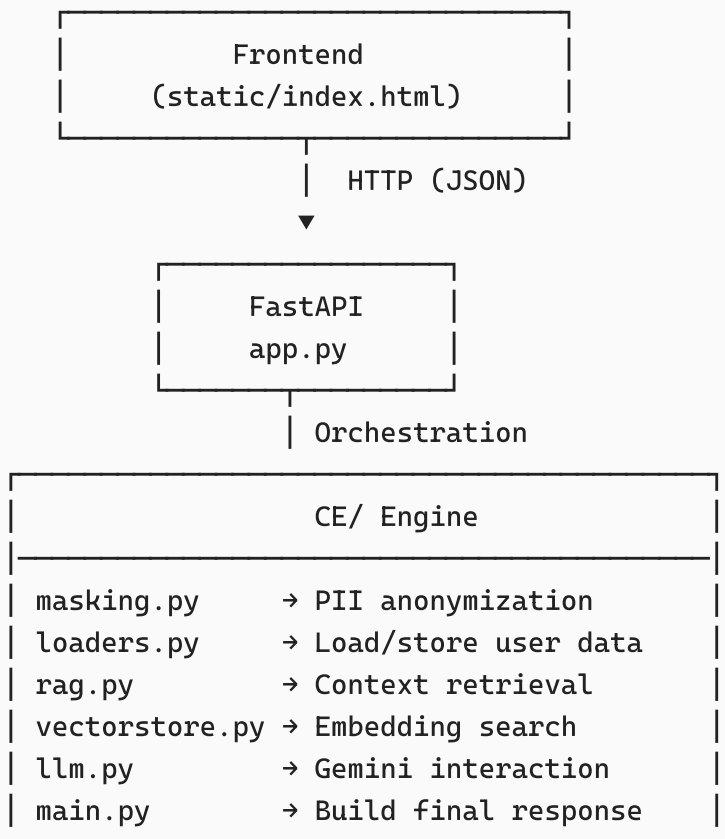
```
│
├── data/                       # Application datasets
│   ├── users.md                # User profiles (markdown)
│   ├── store.md                # Store metadata (markdown)
│   ├── user_history/           # Conversation memory files
│
├── models/                     # Pretrained models (if any)
│
├── static/                     # Frontend UI (HTML/CSS/JS)
│   └── index.html              # Chat interface
│
├── venv/                       # Python virtual environment
│
├── .env                        # Environment variables (Gemini API key)
├── .gitignore                  # Ignore unneeded files
├── app.py                      # FastAPI server (main backend entry point)
├── README.md                   # Project overview
└── requirements.txt            # Dependencies
```

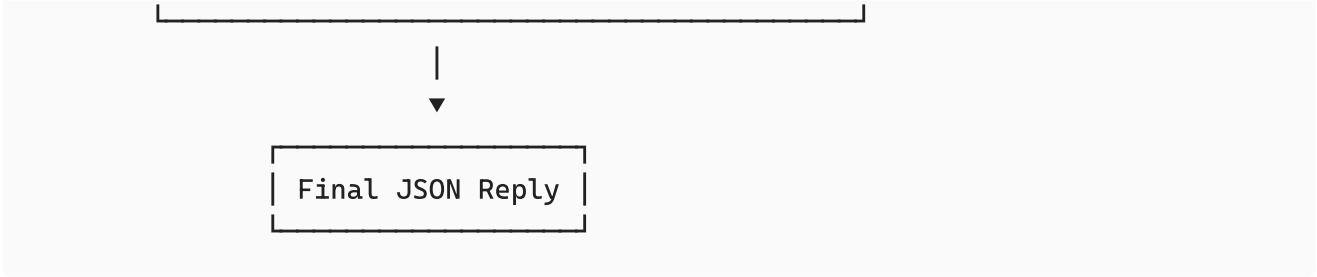
---

3. Core System Architecture

The system follows a **modular**, **production-friendly architecture**, similar to real microservices-based backend AI applications.

3.1 Architecture Diagram





---

4. Component Descriptions

4.1 CE/ — Core AI Engine

This folder contains all backend logic written in a clean, modular pattern.

llm.py

Centralized LLM module:

- Initializes Gemini API
- Builds prompts
- Fetches AI responses
- Handles errors & fallback messaging

masking.py

Responsible for masking:

- Phone numbers
- Email addresses
- Credit card numbers

Ensures **no raw PII** ever reaches the LLM.

rag.py

Handles RAG operations:

- PDF or markdown ingestion
- Chunk generation
- Embedding computation
- Context retrieval

vectorstore.py

Wrapper around FAISS for:

- Index creation
- KNN search
- Index persistence

loaders.py

Reads and structures:

- User profiles
- Store details
- Inventory files

main.py

This is the “brain” of the system:

- Merges user data + RAG + location + LLM
- Formats the final response
- Adds personalization into every message

---

5. Data Layer

users.md

Markdown representation of user profiles (name, preferences, loyalty status).

store.md

Markdown file describing the store location, offers, timings.

user_history/

Directory storing per-user conversation context to enable:

- Memory
- Preference tracking
- Better personalization

---

6. Frontend UI

The UI is built using:

- **TailwindCSS** (design system)

- **Vanilla JavaScript** (lightweight)
- **Smooth animations**
- **Starbucks-themed color palette**

Frontend features:

- Real chat UI
- Quick actions (e.g., “I’m cold”)
- Store info panel
- Live response rendering
- Typing indicator

---

7. Backend (FastAPI)

Features:

- `/api/chat` → Main conversational endpoint
- `/api/health` → Service status
- PII masking
- Conversational memory
- Personalized replies
- Location-aware suggestions

Technologies:

- FastAPI
- Gemini 2.5 Flash
- Geopy
- FAISS
- Sentence Transformers
- Dotenv

---

8. Running the Application

Step 1: Install dependencies

```
pip install -r requirements.txt
```

Step 2: Set your Gemini API key

```
echo GEMINI_API_KEY=your_key_here > .env
```

Step 3: Start the backend

```
uvicorn app.main:app --reload
```

Step 4: Open the frontend

Open:

```
static/index.html
```

---

9. Key Technical Highlights

Hyper-personalized responses

Based on:

- User identity
- Past chats
- Favorite drinks
- Loyalty tier
- Offers and inventory

Location awareness

Bot uses geodesic distance to estimate how far the user is from the store.

Real RAG pipeline

Uses chunking & embeddings to retrieve:

- Order history
- Preference notes
- Store policies

Production-level privacy

PII masking ensures compliance with enterprise rules.

Clean, modular architecture

Separates:

- LLM logic
- RAG logic
- Data loading
- Masking
- API layer

Lightweight & fast

Optimized for hackathon-style rapid deployment.

---

10. Conclusion

This project demonstrates a **real-world, production-ready** solution to customer experience automation.

It integrates:

- AI
- RAG
- Real-time store context
- Geolocation
- Secure data handling
- A modern UI