# Mohammad Aosaf

[mohammadaosaf@gmail.com](mailto:mohammadaosaf@gmail.com)

03235943577

## **Overview**

The system is designed to provide API services using FastAPI, manage customer data with SQLite, and process natural language queries through machine learning models. It is hosted with ngrok for public accessibility and packaged in a Docker container using NVIDIA CUDA for optimized performance.

## **Components**

* **FastAPI**: Manages API endpoints to register and authenticate users, handle customer data, and process questions.
* **SQLite and aiosqlite**: Store and manage customer data asynchronously.
* **SentenceTransformer and BART**: Used for embedding generation and answering questions, respectively. Smaller models like BART are employed due to resource constraints of Google Colab, although larger models like Falcon7B or Llama could enhance performance if available.

## **API Endpoints**

* **/register**: Registers a new customer and generates a unique account number.
* **/chat**: Authenticates a user and processes their question to provide relevant answers.
* **/customer/{customer\_id}**: Retrieves and updates customer information.

## **Docker Configuration**

* **Base Image**: Uses NVIDIA CUDA with cuDNN support to leverage GPU acceleration, ensuring efficient processing of machine learning tasks.
* **Environment Setup**: The container is set up with Python, pip, and necessary utilities like wget and unzip. This environment supports the system's dependencies and runtime.
* **Dependencies**: Python dependencies are managed through pip, ensuring the latest packages are used.
* **ngrok Installation**: ngrok is installed directly in the container. It's used to create a secure tunnel to the FastAPI server, making the chatbot accessible over the internet from anywhere.
* **Port Exposing**: Port 8000 is exposed for FastAPI, making it accessible externally.
* **Running the Application**: The application is executed with **main.py**, which starts the FastAPI server.

## **Functionalities and Task Execution**

**Queries the Customer Database**: The code queries the SQLite database asynchronously through aiosqlite to fetch customer records. This allows the system to access data about customers such as their names, emails, and purchase histories.

**Fetches and Updates Rows in the Customer Database**:

* The **fetch\_customer\_record** function asynchronously retrieves a customer's data from the SQLite database using their ID.
* The **update\_customer\_record** function asynchronously updates customer information, such as name and email, in the database.

**Answers Queries Regarding Company Policies Using Their Custom Documents (Hint: R.I.D - Retrieve, Interpret, Disseminate)**:

* The system uses SentenceTransformer to create embeddings from customer queries and uses ChromaDB to store and retrieve related documents.
* When a query is received, it's processed to find the most relevant document in ChromaDB, which is then used as context for the BART model to generate a relevant response.

**Performs General Chatting and Question-Answering**:

* The BART model, pre-trained on diverse datasets, is employed to generate human-like responses to general queries. This model uses the context provided by ChromaDB to ensure that responses are relevant and informed.

**Records Conversations with Users, and Logs Which Component Was Accessed at Each Bot-Customer Interaction**:

* Each interaction, including the user input, system response, and which component was accessed (e.g., database or RAG), is logged into a log file using Python’s logging library. This helps in maintaining a transparent and traceable interaction history.

## **Development Steps for API and System Scaling**

**Step 1: API Development**

* Develop RESTful APIs using FastAPI, allowing for efficient request handling with asynchronous support.

**Step 2: Containerization and DevOps**

* Dockerize the application to ensure consistent environments and easy deployment. The Dockerfile provided sets up a Python environment with all necessary dependencies and configures ngrok to expose the FastAPI application.
* Use NVIDIA CUDA Docker base images to ensure that the GPU resources are properly utilized, especially for the machine learning models.

**Step 3: Testing and Scalability**

**Testing**:

* Conduct **stress tests** to evaluate the system’s performance with multiple simultaneous users, especially testing API endpoints like **/register**, **/chat**, and customer data management for concurrency and error handling.
* Focus on optimizing SQL queries and database interactions to improve response times and throughput.

**Optimization**:

* Enhance database operations using asynchronous processing to reduce latency.
* Optimize the BART model’s inference by leveraging CUDA capabilities effectively, adjusting inference parameters for better performance.

**Step 4: Documentation and Evaluation**

* Document the API endpoints, system architecture (with diagrams), and the testing/evaluation strategies.
* Include insights on system performance and potential areas for improvement, such as upgrading machine learning models or scaling the hardware resources.

## **Resource Constraints**

Due to the limitations of Google Colab, smaller models like BART were used instead of more powerful options that require greater resources. Upgrading to more capable hardware could allow the use of advanced models, potentially improving the system's performance and capabilities.