E-Commerce Chatbot Documentation

Project Overview

The **E-Commerce Chatbot** is a smart assistant designed to enhance the shopping experience by providing detailed information about products and order history. Leveraging **Retrieval-Augmented Generation (RAG)** techniques, the chatbot integrates with product and order datasets to deliver accurate and contextual responses to user queries.

Key Features

- **Product Queries**: Retrieve detailed information about products, including descriptions, features, ratings, and pricing.
- **Order Queries**: Fetch and provide order details based on customer ID.
- **RAG Integration**: Enhances response accuracy by retrieving relevant information from datasets.
- **Cost-Efficiency**: Balances model quality with performance and hosting costs.
- **Interactive Interfaces**: Utilizes FastAPI for backend services and Streamlit for user-facing interfaces.

Project Structure

The project follows a modular and organized structure to ensure scalability and maintainability.

```
ecommerce_chatbot/
├── app/
│   ├── __init__.py
│   ├── main.py
│   ├── routers/
│   │   ├── __init__.py
│   │   ├── products.py
│   │   ├── orders.py
│   │   ├── chatbot.py
│   │   └── healthcheck.py
│   ├── services/
│   │   ├── __init__.py
```

```
|   |   ├── rag_service.py
|   |   ├── gemini_service.py
|   |   ├── product_service.py
|   |   ├── chatbot_service.py
|   |   └── order_service.py
|   ├── utils/
|   |   ├── __init__.py
|   |   ├── preprocess.py
|   |   ├── logger.py
|   |   └── config.py
|   └── models/
|       ├── __init__.py
|       ├── product_model.py
|       └── order_model.py
├── datasets/
|   ├── orders.csv
|   └── products.csv
├── tests/
|   ├── test_products.py
|   ├── test_orders.py
|   └── test_rag_service.py
├── requirements.txt
├── Dockerfile
├── docker-compose.yml
├── .env
├── README.md
├── streamlit_orders.py
├── streamlit_products.py
└── run.py
```

Directory Breakdown

- **app/**: Contains the main application modules.
    - **routers/**: API route handlers for different endpoints.
    - **services/**: Business logic and service layer implementations.
    - **utils/**: Utility modules such as preprocessing, logging, and configuration.
    - **models/**: Data models representing the structure of product and order data.
- **datasets/**: Raw datasets used by the chatbot.
- **tests/**: Unit and integration tests ensuring code reliability.

- **streamlit_*.py**: Streamlit interfaces for interacting with the chatbot.
- **run.py**: Main script to launch the application.

Setup and Installation

Prerequisites

- **Python 3.11**
- **Git** (for version control)

Installation Steps

Clone the Repository

```
git clone https://github.com/yourusername/ecommerce_chatbot.git
cd ecommerce_chatbot
```

1.

Create a Virtual Environment

```
python3 -m venv venv
source venv/bin/activate
```

2.

Install Dependencies

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

3.

Set Up Environment Variables

Create a `.env` file in the root directory and add the following:
env

```
GEMINI_API_KEY=your_gemini_api_key
```

4.

Run the Application

```
python run.py
```

5. The API will be accessible at `http://localhost:8000`.
6. **Access Streamlit Interfaces**
    - **Orders Chatbot**: `streamlit_orders.py`
    - **Products Chatbot**: `streamlit_products.py`

Run them using:

```
streamlit run streamlit_orders.py
streamlit run streamlit_products.py
```

7.

Detailed Instructions for Use

This section provides step-by-step instructions on how to interact with the **E-Commerce Chatbot** using both the API and the Streamlit interfaces. It also explains the underlying processes based on the provided code.

Running the API

The backend of the chatbot is powered by **FastAPI**, which serves various endpoints to handle product and order queries.

1. Start the Backend Server

Ensure your virtual environment is activated and run:

```
python run.py
```

The FastAPI server will start at `http://0.0.0.0:8000`.

Using the Streamlit Interfaces

Streamlit provides user-friendly web interfaces for interacting with the chatbot without needing to use API endpoints directly. Two separate interfaces are available: one for order-related queries and another for product-related queries.

1. Orders Chatbot Interface

- **File**: `streamlit_orders.py`

Running the Orders Chatbot

1. **Ensure the Backend Server is Running**
   Start the FastAPI server as described above.

Run the Streamlit Application

```
streamlit run streamlit_orders.py
```

2.
3. **Interact with the Chatbot**
    - **Customer ID**: Enter the customer's unique identifier.
    - **Your Query**: Type your question related to orders (e.g., "What are my recent orders?").
    - **Send Query**: Click the button to submit your query.
    - **Conversation History**: View the ongoing dialogue between the user and the chatbot.

2. Products Chatbot Interface

- **File**: `streamlit_products.py`

Running the Products Chatbot

1. **Ensure the Backend Server is Running**
   Start the FastAPI server as described above.

Run the Streamlit Application

```
streamlit run streamlit_products.py
```

2.
3. **Interact with the Chatbot**
    - **Your Query**: Type your question related to products (e.g., "Tell me about the latest acoustic guitars.").
    - **Send Query**: Click the button to submit your query.
    - **Conversation History**: View the ongoing dialogue between the user and the chatbot.