Shop Smart AI Recommender

A conversational AI-powered product recommendation system built with a modern LLMOps stack, containerized with Docker, and deployed on Kubernetes.

Final Application

✨ Features

```
Conversational RAG Chain: Utilizes a Retrieval-Augmented Generation (RAG)
architecture to answer user questions based on a knowledge base of product
reviews.

Context-Aware Memory: Remembers the context of the conversation to answer
follow-up questions intelligently.

Cloud-Native Deployment: Fully containerized with Docker and orchestrated with
Kubernetes for scalability and resilience.

Real-time Monitoring: Integrated with Prometheus and Grafana for observing
application health and performance.

Secure Configuration: Manages all API keys and secrets securely using
environment variables and Kubernetes Secrets.
```

🛠️ Technology Stack

Category	Technology
AI/ML	LangChain, Groq, Hugging Face sentence-transformers
Web & Backend	Flask
Database	Astra DB (Vector Store)
Cloud & Infra	GCP, Docker, Kubernetes (Minikube)
Monitoring	Prometheus, Grafana

🚀 Local Setup and Usage

Follow these steps to set up and run the project on your local machine.
Prerequisites

```
Python 3.10+

An account with Astra DB

API keys for Groq and Hugging Face
```

1. Clone the Repository

git clone https://github.com/Najam0786/Smart-Shop-AI-Recommender.git
cd Smart-Shop-AI-Recommender

2. Create and Activate a Virtual Environment

For Windows

python -m venv env
.\env\Scripts\activate

For macOS/Linux

python3 -m venv env
source env/bin/activate

3. Set Up Environment Variables

   Create a file named .env in the root of the project.

   Add the following keys with your credentials:

GROQ_API_KEY="your_groq_api_key"
HUGGINGFACEHUB_API_TOKEN="your_huggingface_api_token"
ASTRA_DB_API_ENDPOINT="your_astra_db_endpoint"
ASTRA_DB_APPLICATION_TOKEN="your_astra_db_token"
ASTRA_DB_KEYSPACE="your_keyspace_name"
FLASK_SECRET_KEY="run_python_-c_import_secrets;_print(secrets.token_hex())_to_generate"

4. Install Dependencies

Install the project and all its dependencies in editable mode.

pip install -e .

5. Ingest Data into Astra DB

Run the data ingestion script. This only needs to be done once.

python utils/data_ingestion.py

6. Run the Flask Application

python app.py

The application will be available at http://127.0.0.1:5000.

☁️ Deployment on GCP with Kubernetes

This project is designed for cloud deployment. The high-level steps are:

```
Create a GCP VM: Set up an E2-Standard Ubuntu VM with at least 16GB RAM and
256GB storage.

Configure VM: Install Docker, Minikube, and kubectl on the VM.

Sync Code: Clone the repository onto the VM.

Build Image: Run eval $(minikube docker-env) and then docker build -t
shopsmart-ai-app:latest . to build the image inside Minikube.

Create Secrets: Manually create a .env file on the VM and run kubectl create
secret generic ... to create the Kubernetes secret.

Deploy: Apply all the .yaml files using kubectl apply -f [filename].

Expose Service: Use kubectl port-forward to access the application and
monitoring dashboards.
```

For detailed commands, please refer to the project_document.md.

🗁 Project Structure

```
/
├── assets/ # Project images and screenshots
├── chain/ # Core RAG chain logic
├── config/ # Application configuration
├── data/ # Raw dataset
├── grafana/ # Grafana Kubernetes manifests
├── prometheus/ # Prometheus Kubernetes manifests
├── static/ # CSS and other static assets
├── templates/ # HTML templates
├── utils/ # Reusable helper modules
├── .env # (Local Only) Secret keys and APIs
├── .gitignore # Files to be ignored by Git
├── app.py # Main Flask application entry point
├── Dockerfile # Instructions to build the container image
```

```
├── flask-deployment.yaml # Kubernetes manifest for the Flask app
├── requirements.txt # Python dependencies
└── setup.py # Project packaging script
```

License

This project is licensed under the MIT License. See the LICENSE file for details.