MultiAgent Shopping Assistant: Comprehensive Documentation

1. Introduction

The MultiAgent Shopping Assistant is an advanced, RAG-powered shopping application built with Streamlit. It is designed to help users efficiently find, compare, and analyze products from various online stores. The application leverages web data, link analysis, and intelligent multi-agent recommendations to provide a comprehensive shopping experience. A key feature is its support for Hindi language queries, making it accessible to a broader user base.

Key Features:

- **Multi-Agent Shopping System**: Utilizes specialized AI agents for different aspects of the shopping process.
- **Agentic RAG Product Search**: Employs Retrieval-Augmented Generation (RAG) with Google Gemini and Serper for web data-powered product searches.
- **Link Analysis**: Capable of extracting and analyzing product information directly from provided URLs.
- **Product Comparison**: Offers side-by-side comparison of multiple products.
- **Hindi Language Support**: Facilitates searching in both Hindi and English, with automatic translation capabilities.
- **AI Shopping Assistant**: Provides intelligent insights and personalized recommendations.
- **Price Comparison**: Compares product prices across various retailers.
- **Direct Purchase Links**: Enables direct navigation to product pages for purchase.
- **Responsive Design**: Ensures a seamless user experience across desktop and mobile devices.

- **Smart Suggestions**: Offers search suggestions as the user types.
- **Demo Mode**: Allows basic functionality without requiring API keys, ideal for testing and demonstration.

2. Multi-Agent System Components

The application's core intelligence is driven by a sophisticated multi-agent system, where each agent specializes in a particular task to collectively enhance the shopping experience. While the initial design considered Google ADK, the current implementation leverages structured Gemini prompts for multi-agent functionality, ensuring robust and efficient operation.

Agents and Their Roles:

- **Translation Agent**: Handles the translation of Hindi queries into English, enabling seamless search functionality for Hindi-speaking users.
- **Search Agent**: Responsible for extracting and analyzing product information from web search results, primarily utilizing Serper for web data and Gemini for RAG capabilities.
- **Price Analysis Agent**: Focuses on comparing prices across different online retailers and providing value analysis to identify the best deals.
- **Delivery Analysis Agent**: Analyzes various delivery options, speeds, and associated costs for products.
- **Recommendation Agent**: Provides final, intelligent recommendations and personalized shopping tips based on the aggregated information.
- **Link Analysis Agent**: Specifically designed to extract detailed product data from direct product URLs provided by the user.
- **Comparison Agent**: Intelligently compares multiple products side-by-side, highlighting key differences and similarities to aid user decision-making.

3. Technical Architecture

The MultiAgent Shopping Assistant is built primarily using Python and the Streamlit framework for its user interface. It integrates with external APIs for enhanced search and AI capabilities. The project follows a modular structure, separating concerns into different directories and files for maintainability and scalability.

Project Structure:

Plain Text

```
multiagent-shopping-assistant/
├── streamlit_app.py          # Main Streamlit application
├── run_streamlit.py          # Streamlit runner script
├── requirements.txt          # Python dependencies
├── README.md                # Project README file
├── .env.example             # Environment variables template
└── src/
    ├── services/            # Contains core business logic and API
integrations
    │   ├── hindi_shopping_search.py  # Hindi-English search translation
logic
    │   ├── multi_agent_shopping.py   # Orchestrates multi-agent interactions
and Gemini API calls
    │   ├── product_comparison.py     # Handles product comparison logic
    │   ├── rag_shopping_search.py    # RAG implementation with Gemini and
Serper
    │   └── serper_shopping_search.py # Serper Google search API integration
    └── static/              # Static assets (e.g., favicon)
        └── favicon.ico
```

Core Technologies and Libraries:

- **Streamlit**: Used for building the interactive web application interface. It simplifies the creation of data apps with Python.
- **Python**: The primary programming language for the entire application.
- **Requests**: A popular HTTP library for making API calls to external services like Serper.
- **Pandas**: Utilized for data manipulation and analysis, especially for product comparison and formatting search results.

- **Google Generative AI (Gemini)**: Integrated for advanced AI capabilities, including query analysis, intelligent recommendations, and powering the RAG system.
- **BeautifulSoup4** and **lxml**: Libraries for web scraping and parsing HTML content, crucial for extracting product information from various online sources.
- **python-dotenv**: Manages environment variables, allowing for secure handling of API keys.
- **sentence-transformers**, **faiss-cpu**, **numpy**, **scikit-learn**: These libraries are listed in `requirements.txt` and suggest potential use cases for advanced text processing, embedding generation, and similarity search, which are common in RAG implementations, although their direct usage might be abstracted within the `src/services` modules.
- **langdetect**: Used for detecting the language of user queries, specifically to identify Hindi input for translation.

4. Installation and Setup

To set up and run the MultiAgent Shopping Assistant locally, follow these steps:

Prerequisites:

- Python 3.8 or higher
- `pip` package manager

Step-by-Step Installation:

1. **Clone the Repository**:
   Open your terminal or command prompt and clone the GitHub repository:
2. **Install Dependencies**:
   Navigate to the cloned directory and install the required Python packages using `pip`:
3. **Set up Environment Variables (Optional but Recommended)**:
   For full functionality, you will need API keys for Google Gemini (for AI capabilities) and

Serper (for web search). The application can run in a limited "Demo Mode" without these keys.

4. **Run the Application**:
   You can run the Streamlit application using either `python` or `streamlit` command:

5. **Access the Application**:
   Once the application starts, open your web browser and navigate to `http://localhost:8501`.

Demo Mode:

The application supports a "Demo Mode" which allows it to run without any API keys. In this mode, it provides:

- Immediate functionality without configuration.
- Sample data for testing and demonstration purposes.
- Hindi translation capabilities.
- Shopping list functionality and UI/UX features.

Limitations of Demo Mode:

- No real web search results.
- Limited AI-powered analysis.
- Restricted product data.

To unlock full functionality, it is highly recommended to configure the API keys in the `.env` file.

5. Usage Guide

The MultiAgent Shopping Assistant provides an intuitive interface for various shopping-related tasks. Here’s how to use its main features:

5.1. Product Search

Use the main search bar to find products online. The assistant supports queries in both Hindi and English, providing AI-powered insights.

- **Input**: Type your product query into the search bar. Examples include "tamatar" (tomato), "doodh" (milk), "mobile", or "laptop".
- **Suggestions**: As you type, the assistant may provide smart search suggestions to refine your query.
- **Results**: The search results will display relevant products with their titles, descriptions, prices, sources, and images. An AI-generated response will also provide comprehensive shopping recommendations based on the search results.

5.2. Link Analysis

If you have a direct product link, you can paste it into the application to get a detailed analysis of the product.

- **Supported Sites**: The link analysis feature supports major e-commerce sites like Amazon, Flipkart, and others.
- **Details**: The assistant will extract and present detailed information about the product from the provided URL.

5.3. Product Comparison

Compare multiple products side-by-side to make informed purchasing decisions.

- **Input**: Enter 2-4 product links into the designated comparison section.
- **Intelligent Comparison**: The Comparison Agent will intelligently analyze and present a side-by-side comparison, highlighting key attributes and differences.

5.4. AI Insights

The AI assistant provides intelligent recommendations and price analysis to help you shop smarter.

- **Recommendations**: Get personalized shopping tips and product suggestions.
- **Price Analysis**: Understand price ranges, best value options, and budget considerations.
- **Delivery Analysis**: Gain insights into delivery options, speed, and costs.

5.5. Language Help

For Hindi-speaking users, a dedicated "Language Help" tab provides Hindi-English translations to assist with queries.

6. Deployment

This section outlines various methods for deploying the MultiAgent Shopping Assistant.

6.1. Local Development

For local development and testing, you can run the Streamlit application directly:

```bash
streamlit run streamlit_app.py
```

6.2. Streamlit Cloud Deployment

Streamlit Cloud offers a straightforward way to deploy Streamlit applications. Follow these steps:

1. **Push your code to GitHub**: Ensure your project is pushed to a GitHub repository.
2. **Connect to Streamlit Cloud**: Go to Streamlit Cloud and connect your GitHub repository.
3. **Set Environment Variables**: Configure your `SERPER_API_KEY` and `GOOGLE_API_KEY` in the Streamlit Cloud dashboard secrets management.
4. **Deploy!**

6.3. Heroku Deployment

To deploy the application on Heroku, you will need a `Procfile` to specify how your application should be run. Heroku's standard Python deployment process can then be followed.

1. **Create a `Procfile`**: In the root directory of your project, create a file named `Procfile` (no file extension) with the following content:
2. **Deploy to Heroku**: Follow Heroku's standard Python deployment guide to deploy your application.

7. Dependencies

The project relies on the following key Python libraries, as specified in `requirements.txt`:

- **streamlit**: The web application framework.
- **requests**: For making HTTP requests to external APIs.
- **pandas**: For data manipulation and analysis.
- **python-dotenv**: For loading environment variables from a `.env` file.
- **google-generativeai**: The official Google Generative AI client library for interacting with Gemini models.
- **beautifulsoup4**: For parsing HTML and XML documents.
- **lxml**: A fast and powerful XML and HTML processing library.
- **langdetect**: For language detection.
- **sentence-transformers**: For generating sentence embeddings (likely used for RAG or similarity search).
- **faiss-cpu**: A library for efficient similarity search and clustering of dense vectors (likely used in conjunction with `sentence-transformers`).
- **numpy**: Fundamental package for numerical computing in Python.

- **scikit-learn**: Provides simple and efficient tools for data mining and data analysis.
- **urllib3**, **certifi**, **charset-normalizer**, **idna**: Utility dependencies for network operations and character encoding.

8. Contributing

Contributions to the MultiAgent Shopping Assistant project are welcome! If you wish to contribute, please follow these general guidelines:

1. **Fork the repository**.
2. **Create a feature branch** (`git checkout -b feature/YourFeatureName`).
3. **Make your changes** and ensure they adhere to the project's coding standards.
4. **Test the application** thoroughly to ensure new features work as expected and existing functionality is not broken.
5. **Submit a pull request** with a clear description of your changes.

9. License

This project is licensed under the MIT License. See the `LICENSE` file in the repository for full details.