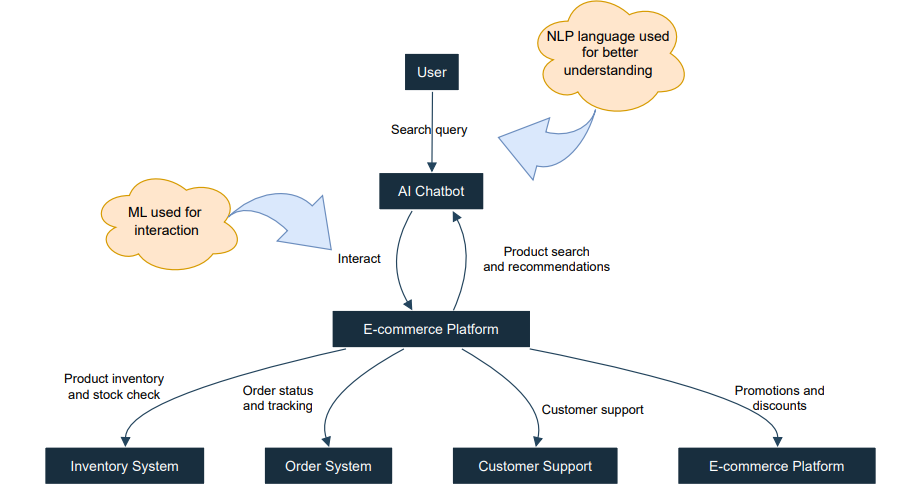
**Title:** Chat Bot

**Problem Statement:**

E-commerce platforms frequently struggle to provide an efficient and satisfying search experience for their users due to several key limitations:

* **Keyword Search Inefficiencies**: Traditional search algorithms rely heavily on exact keyword matching, which often results in irrelevant or incomplete search results when user queries are not precise or are worded differently from the product listings.
* **Customer Dissatisfaction**: The inability to find relevant products quickly frustrates users, leading to higher bounce rates, abandoned carts, and decreased customer satisfaction, which directly impacts conversion rates.
* **Handling Complex Queries**: Users may search using complex or nuanced language, which conventional search engines find difficult to interpret, leaving customers unable to find specific products or details they are looking for.
* **Lack of Personalization**: Standard search systems fail to consider individual preferences, browsing history, or purchasing behaviour, resulting in generic search results that do not cater to the user's unique needs or preferences, reducing the chances of conversion.

**Architecture:**



The diagram represents the architecture of an AI-powered chatbot integrated into an e-commerce platform.

It illustrates the interaction flow between the user and various components of the system.

Here's a breakdown of each section:

1. **User Interaction**:
   * The user communicates with the chatbot by sending a *search query* or asking questions related to products, orders, or promotions.
2. **AI Chatbot**:
   * The chatbot is the interface between the user and the e-commerce platform, employing Machine Learning (ML) techniques to facilitate better interaction.
   * It uses **Natural Language Processing (NLP)** to understand the user's language more effectively, allowing it to interpret complex or nuanced queries.
3. **E-commerce Platform**:
   * The chatbot interacts with different subsystems within the platform, which include:
     + **Inventory System**: To check product availability and stock levels.
     + **Order System**: For providing updates on order status and tracking.
     + **Customer Support**: Handling customer inquiries, support requests, and resolving issues.
     + **Promotions and Discounts**: Informing users about active promotions and offers.
4. **Interaction Flow**:
   * Once the user sends a query, the chatbot performs a **product search** and provides **recommendations** based on the user’s input.
   * It communicates with the e-commerce platform’s internal systems for checking **inventory**, **order status**, or providing **customer support**.
5. **Machine Learning Integration**:
   * The chatbot uses ML to continuously improve its responses and recommendations, adapting to user preferences over time.

**Dependencies**

### **1. Pandas (pandas==1.3.5)**

* **Purpose**: Pandas is a powerful library for data manipulation and analysis. It provides data structures like DataFrames and Series, which are essential for handling and analyzing structured data (like CSV files, Excel sheets, and SQL databases).
* **Usage in Your Project**: You use Pandas to load, manipulate, and analyze product data stored in a CSV file. The data is then used to match user queries to product descriptions.

### **2. Flask (Flask==2.0.3)**

* **Purpose**: Flask is a lightweight web application framework for Python. It’s designed to be simple, flexible, and easy to use, making it popular for building web applications and APIs.
* **Usage in Your Project**: Flask serves as the backend framework for your chatbot. It handles HTTP requests from the frontend (PHP part of your project), processes them (using the product data and model), and sends responses back to the frontend.

### **3. Transformers (transformers==4.18.0)**

* **Purpose**: The transformers library, developed by Hugging Face, provides pre-trained models for various natural language processing (NLP) tasks. These models can perform tasks such as text classification, translation, summarization, and more.
* **Usage in Your Project**: You use the transformers library to load a pre-trained BERT tokenizer, which is used to tokenize user queries into tokens (words or subwords) that the model can process for understanding and matching products.

### **4. Torch (torch==1.10.2)**

* **Purpose**: PyTorch (imported as torch) is a popular deep learning framework. It provides tensor computations (similar to NumPy) with strong GPU acceleration and a dynamic computational graph that makes building neural networks more flexible and easier.
* **Usage in Your Project**: While not explicitly used in the code you provided, torch would be required if you're using deep learning models for tasks like NLP (Natural Language Processing), especially if you’re leveraging the models from the transformers library. PyTorch is the underlying framework that many of these models are built upon.

### **5. FuzzyWuzzy (fuzzywuzzy==0.18.0)**

* **Purpose**: FuzzyWuzzy is a library for fuzzy string matching. It allows you to find approximate matches between strings by computing a similarity score between them. This is useful when you want to match user input with existing strings that may not exactly match but are close enough.
* **Usage in Your Project**: You use FuzzyWuzzy to match the tokenized user query against product descriptions in your data. This helps in finding the most relevant products based on the user's input, even if the input isn’t an exact match.

### **6. Werkzeug (werkzeug==2.0.3)**

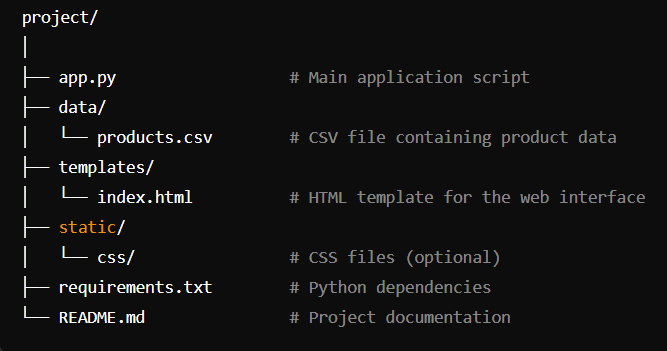
* **Purpose**: Werkzeug is a WSGI (Web Server Gateway Interface) utility library for Python. It's used by Flask to handle the underlying request and response objects. It also provides utilities for routing, debugging, and error handling.
* **Usage in Your Project**: Werkzeug is used under the hood by Flask to manage HTTP requests and responses, as well as various other web-related tasks. You typically don’t interact with Werkzeug directly, but it is critical for Flask’s operation.

### **7. Scikit-learn (scikit-learn==0.24.2)**

* **Purpose**: Scikit-learn is a widely-used machine learning library in Python. It provides simple and efficient tools for data mining, data analysis, and machine learning. It includes algorithms for classification, regression, clustering, and dimensionality reduction, among others.
* **Usage in Your Project**: While scikit-learn is often used for building machine learning models, in your project, it may be used for tasks like tokenizing and transforming data, possibly for pre-processing before using it with models from the transformers library or for matching with FuzzyWuzzy.

**Project Structure**

The following is the recommended structure for your Flask application:



### **1. Project Root (project/)**

* The root directory of your Flask application. This is where your main application script and other key files are located. This directory contains everything needed to run your project.

### **2. app.py**

* **Purpose**: This is the main application script for your Flask project.
* **Role**: It contains the core logic of your application, including routes that handle incoming requests, interactions with data, and rendering templates. When you start your Flask server, this script is executed, and it runs the application.
* **Example Content**: Defines routes (URLs) and connects them to functions that process requests and return responses. For example, it might have routes for displaying the home page, handling user queries, and returning JSON responses for chatbot interactions.

### **3. data/**

* **Purpose**: This directory is used to store data files that your application needs to access and use.
* **Role**: In your case, it contains products.csv, which holds the product information that your application will search and display based on user queries.
* **Example Content**: The products.csv file might contain columns such as product names, descriptions, prices, and availability, which your app reads and uses to respond to user queries.

### **4. templates/**

* **Purpose**: This directory contains HTML templates for your web application's front-end.
* **Role**: Flask uses Jinja2 as its template engine, which allows you to create dynamic HTML pages. The index.html file here is a template that serves as the main interface for your application, allowing users to interact with the chatbot.
* **Example Content**: The index.html might include the structure of your webpage, placeholders for dynamic content (e.g., chatbot responses), and integration with CSS and JavaScript for styling and functionality.

### **5. static/**

* **Purpose**: This directory is where you store static files like CSS, JavaScript, and images.
* **Role**: Static files are served directly to the browser without any server-side processing. The css/ subdirectory would contain any stylesheets you use to style your HTML templates.
* **Example Content**: The css/ directory might include files like styles.css, which could define the look and feel of your application, including layouts, colors, fonts, etc.

### **6. requirements.txt**

* **Purpose**: This file lists all the Python libraries that your project depends on, along with their specific versions.
* **Role**: It allows others (or automated systems) to install all necessary dependencies for your project in one step using pip. This ensures that your project has a consistent environment across different setups.
* **Example Content**: It includes lines like Flask==2.0.3, which tells pip to install Flask version 2.0.3.

### **7. README.md**

* **Purpose**: A markdown file that provides documentation for your project.
* **Role**: The README typically explains what the project is, how to set it up, and how to use it. It might also include information about the project structure, dependencies, and any other important details.
* **Example Content**: It could contain sections like "Introduction," "Installation," "Usage," and "Contributing," guiding new users or developers on how to work with your project.

### **Summary**

This structure organizes your Flask project in a way that separates different aspects of the application.

* **app.py** handles the application logic.
* **data/** stores the essential data files.
* **templates/** contains the HTML files that define the front-end interface.
* **static/** holds CSS and other assets that enhance the appearance and functionality of the front end.
* **requirements.txt** ensures that the environment is correctly set up with all necessary dependencies.
* **README.md** serves as a guide for users and developers.

**Usage**

This application is designed to help users search for products based on specific queries related to product features. For example, a user might enter a query like "high storage phone with good camera," and the application will return the most relevant product based on the available descriptions.

**API Endpoints**

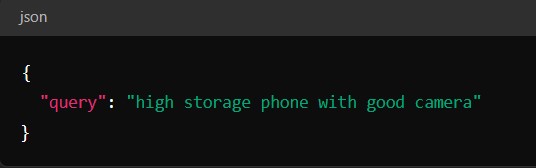
**/ [GET]**

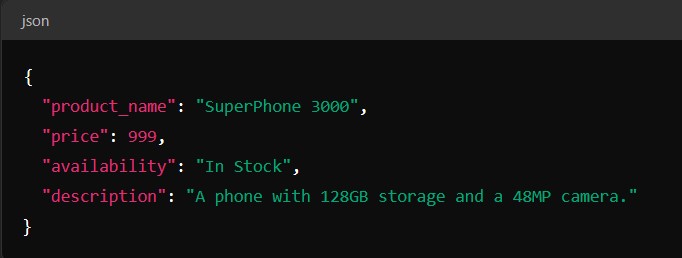
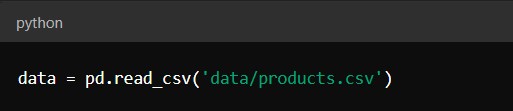
* **Description**: Serves the homepage of the application.
* **Response**: Renders the index.html template.

**/query [POST]**

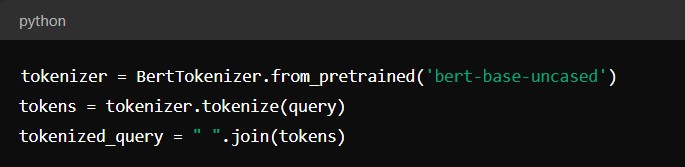
* **Description**: Processes a user query and returns the best matching product.
* **Request**: Expects a JSON payload with a query field. The query is a string that describes the features or characteristics the user is looking for in a product.
* **Response**: Returns JSON data containing details of the best-matching product, including:
  + product\_name: The name of the product.
  + price: The price of the product.
  + availability: Whether the product is in stock.
  + description: A brief description of the product.

Example Request:



Example Response:  
  
  
**KEY COMPONENTS**  
  
**Loading Data**  
  
**-> Purpose: Load the product data from a CSV file into a pandas DataFrame.**

**Tokenizing Queries**

  
**Purpose: Tokenizes the user query using a pre-trained BERT tokenizer.**

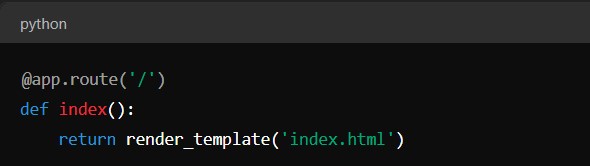
**Product Matching**



**Purpose: Finds the best matching product description using fuzzy string matching.**

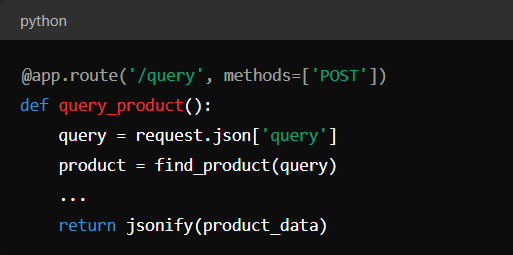
**Flask Routes**

* **index()**



**Purpose:** This route handles GET requests to the root URL (/). It renders and returns the index.html template, which serves as the homepage of the application.

* **query\_product()**



**Purpose**: This route handles POST requests to the /query endpoint. It processes the user's query, finds the most relevant product using the find\_product function, and returns the product details in a JSON format.

**Running the Application**

To run the Flask-based product search application, follow these steps:

1. **Navigate to Your Project Directory**:  
   cd E:\xampp\htdocs\ecommerce\project
2. **Start the Flask Development Server:**  
   python app.py
3. **Access the Application in Your Web Browser:**  
   http://127.0.0.1:5000

**OUTPUT**

