**Design Documentation: E-commerce Recommendation System**

**1. Introduction**

The goal of the **E-commerce Recommendation System** is to enhance user experience on an e-commerce platform by providing personalized product recommendations. This system will have two versions: one with machine learning and one without machine learning. The recommendation engine will suggest relevant products based on user behavior and product features.

**2. Low-Level Design (LLD)**

The low-level design focuses on detailed components, algorithms, database schema, and module-level details.

**2.1 Module Breakdown**

**Module 1: User Management**

* **Components**:
  + Registration
  + Authentication (login, logout)
  + Profile Management
  + Role Management (Admin, Customer)
* **Detailed Description**:
  + A user can register and log in.
  + User roles will determine who can add products (Admin) and who can browse/purchase (Customer).
  + Passwords should be stored securely (hashed).
  + JWT (JSON Web Token) or session-based authentication will be used for handling logged-in users.

**Module 2: Product Catalog**

* **Components**:
  + Add, Edit, Delete Products (Admin)
  + View Products (Customer)
  + Product Filtering and Search
* **Detailed Description**:
  + The Admin can manage the products through a dashboard.
  + Customers will be able to view product details, filter by category, and search for products.
  + A database table will store product details like name, category, price, and image.

**Module 3: Shopping Cart**

* **Components**:
  + Add to Cart
  + Remove from Cart
  + Cart Review
  + Checkout
* **Detailed Description**:
  + Users can add products to a shopping cart, review items, and proceed to checkout.
  + Cart information will be stored in a database session or cookies for persistence across visits.

**Module 4: Recommendation System**

* **Components**:
  + Rule-based recommendation engine (Non-ML)
  + Machine Learning-based recommendation engine
* **Detailed Description**:
  + **Non-ML Version**:
    - Use basic heuristics such as “users also bought,” “popular products,” and category-based recommendations.
    - Store rule-based recommendation algorithms in separate functions.
  + **ML Version**:
    - Collaborative filtering (User-User, Item-Item).
    - Use datasets to train and store models for dynamic product suggestions.
    - Product similarity can be implemented using cosine similarity or matrix factorization.

**Module 5: Order History and Recommendations**

* **Components**:
  + Order History Tracking
  + Personalized Recommendations
* **Detailed Description**:
  + The system will track all purchases and generate recommendations based on past user orders and interactions.
  + Recommendations will be displayed on the homepage/dashboard or product page.

**2.2 Database Design**

**Users Table**

| **Field** | **Type** | **Description** |
| --- | --- | --- |
| user\_id | INT | Primary Key, unique user identifier |
| username | VARCHAR | Unique username for login |
| email | VARCHAR | User email |
| password\_hash | VARCHAR | Hashed password |
| role | ENUM | Admin or Customer |

**Products Table**

| **Field** | **Type** | **Description** |
| --- | --- | --- |
| product\_id | INT | Primary Key, unique product identifier |
| name | VARCHAR | Name of the product |
| category | VARCHAR | Category of the product |
| description | TEXT | Product description |
| price | DECIMAL | Product price |
| image\_url | VARCHAR | URL to product image |

**Orders Table**

| **Field** | **Type** | **Description** |
| --- | --- | --- |
| order\_id | INT | Primary Key, unique order identifier |
| user\_id | INT | Foreign Key, refers to user |
| order\_date | DATETIME | Date and time when the order was placed |

**Cart Table**

| **Field** | **Type** | **Description** |
| --- | --- | --- |
| cart\_id | INT | Primary Key, unique cart identifier |
| user\_id | INT | Foreign Key, refers to user |
| product\_id | INT | Foreign Key, refers to product |
| quantity | INT | Quantity of the product added to cart |

**Recommendation History Table**

| **Field** | **Type** | **Description** |
| --- | --- | --- |
| recommendation\_id | INT | Primary Key, unique recommendation ID |
| user\_id | INT | Foreign Key, refers to user |
| product\_id | INT | Foreign Key, refers to product |
| recommended\_at | DATETIME | Timestamp of recommendation |

**3. High-Level Design (HLD)**

The high-level design provides an overview of the system architecture, main components, and data flow.

**3.1 System Architecture**

1. **Frontend**: The user interface will be built using HTML/CSS/JavaScript frameworks (like React or Vue.js). It will be responsive to ensure it works across devices. The frontend will communicate with the backend using RESTful APIs.
2. **Backend**: The backend server will be developed using Python (Flask or Django). It will handle user authentication, product management, and the recommendation engine.
   * **For the ML version**, a separate service could be responsible for training the recommendation model and serving the results.
3. **Database**: PostgreSQL or SQLite will be used to store user data, product information, order history, and recommendation history.
4. **API Layer**: RESTful APIs will be developed to handle user requests, product searches, and recommendation results. Endpoints could include:
   * /api/products - To retrieve a list of products.
   * /api/cart - To manage the user's shopping cart.
   * /api/recommendations - To get product recommendations.

**3.2 Data Flow Diagram**

* **Step 1**: The user browses or searches for products, and a request is made to the backend API.
* **Step 2**: The backend retrieves the products from the database and sends them back to the frontend.
* **Step 3**: The user adds products to the cart, and the cart details are stored in the database or session.
* **Step 4**: When the user views a product or checks out, the recommendation system is triggered to suggest similar products.
  + **Non-ML version**: The backend uses predefined rules to suggest similar or popular products.
  + **ML version**: The recommendation model (stored and served separately) provides personalized product suggestions.
* **Step 5**: The user completes the purchase, and order details are stored in the database for future recommendations.

**3.3 Component Diagram**

**Frontend**:

* User Registration & Login Page
* Product Listing Page
* Product Detail Page (with recommendations)
* Cart and Checkout Page
* Order History Page

**Backend (Flask/Django)**:

* User Authentication Service
* Product Management Service (CRUD for products)
* Recommendation Engine (ML/Non-ML)
* Cart Management
* Order History Tracking

**Database (PostgreSQL/SQLite)**:

* User Table
* Product Table
* Orders Table
* Cart Table
* Recommendation History Table

**Recommendation Engine**:

* **Non-ML version**: Rule-based service using Python functions.
* **ML version**: Separate microservice to train, store, and serve recommendation models (e.g., using scikit-learn or TensorFlow).

**3.4 Sequence Diagram (For Recommendation Process)**

1. **User Views Product Page**:
   * User → Frontend: Request product page.
   * Frontend → Backend: API request to fetch product details.
   * Backend → Database: Retrieve product info and user interaction data.
   * Backend → Recommendation Engine: Request recommendations based on product/user history.
   * Recommendation Engine → Backend: Return recommended product IDs.
   * Backend → Frontend: Send product info and recommendations.
   * Frontend → User: Display product details along with recommendations.

**4. Conclusion**

This document provides a complete low-level and high-level design for the **E-commerce Recommendation System**. The LLD offers a detailed breakdown of modules, database schema, and components, while the HLD covers the system architecture, data flow, and component relationships. The recommendation engine's modularity allows for both a rule-based and machine learning-based approach, making the system scalable and adaptable for future improvements.