**Requirement Analysis of E-commerce Application**

**1. Break Down Requirements:** This step involves understanding each part of the application in detail and dividing it into **smaller tasks or features** to develop. It helps developers and the team know **exactly what needs to be built**.

**Functional Requirements :**

| **Module** | **Feature** | | **Description** | |
| --- | --- | --- | --- | --- |
| **Customer Management** | Register/Login | | Users should be able to sign up and log in using email & password. | |
|  | Profile Management | | Users can update personal details (name, email, password). | |
| **Product Management** | Add/View Products | | Admin can add products with name, price, description, stock. Customers can view them. | |
|  | Edit/Delete Product | | Admin can update or remove product details. | |
| **Cart System** | Add to Cart | | Customers can add multiple products to their cart. | |
|  | Update Cart | | Users can change quantity or remove items from the cart. | |
| **Order System** | Place Order | | Users can place orders by checking out the cart. | |
|  | View Orders | | Users can see their past orders and order status. | |
| **Admin Panel** | | Manage Products & Orders | | Admin can view all orders and manage inventory. |
| **Search/Filter** | | Product Search | | Customers can search or filter products by name, price, or category. |

### Flow chart:

### 

### 2. Functional Requirement Breakdown (With Technical Perspective)

#### 2.1 ****User Management Module****

This module forms the entry point of the system. It manages user identity, authentication, and profile details.

* **Technical Implementation:**
  + Python Classes: User, AuthenticationService
  + Database Tables: customers
  + SQL Keywords: INSERT, SELECT, UPDATE, DELETE
  + Features:
    - Register new users with hashed passwords (security layer)
    - Authenticate with SELECT and credential checks
    - Use UPDATE to modify user details

#### 2.2 ****Product Management Module****

Responsible for storing and retrieving inventory data like product details, stock status, and pricing.

* **Technical Implementation:**
  + Python Classes: Product, InventoryManager
  + Database Tables: products
  + SQL Keywords: CREATE TABLE, INSERT INTO, UPDATE, SELECT, DELETE
  + Features:
    - Add new items to catalog using INSERT
    - Update stock/price using UPDATE
    - Remove obsolete entries using DELETE

#### 2.3 ****Shopping Cart Module****

Temporarily stores products added by the user before purchase.

* **Technical Implementation:**
  + Python Classes: Cart, CartService
  + Database Tables: cart
  + SQL Concepts: Foreign Keys, Aggregate Functions
  + Features:
    - Add/update quantity of products (INSERT, UPDATE)
    - Remove items from cart (DELETE)
    - Fetch all items for a user via JOIN on products

#### 2.4 ****Order Processing Module****

Final stage where a transaction is confirmed. Combines cart data into order records.

* **Technical Implementation:**
  + Python Classes: Order, OrderProcessor
  + Database Tables: orders, order\_items
  + SQL Features: Transactions (ACID properties), Joins
  + Features:
    - Create an order using data from cart
    - Auto-calculate total and save to orders
    - Store detailed items in order\_items using batch INSERT
    - Implement rollback if insert fails (transaction control)

#### 2.5 ****Administrative Operations****

Manages users, inventory, and orders for monitoring and data correction.

* **Technical Implementation:**
  + Python Classes: AdminPanel, UserManager, ReportService
  + SQL Concepts: JOIN, GROUP BY, HAVING, COUNT()
  + Features:
    - View user activity
    - Monitor top-selling products
    - Delete suspicious users or fake reviews

### 3. Non-Functional Requirements (NFRs) with Technical Elaboration

#### 3.1 ****Performance****

* Use of indexed columns in products, orders for faster search
* Optimize queries with conditions (WHERE, LIMIT, LIKE)
* Efficient data retrieval using proper JOIN strategies

#### 3.2 ****Security****

* Passwords stored with hashing (e.g., bcrypt or hashlib)
* Parameterized SQL queries to avoid SQL injection
* Input validation using Python exceptions and regex

#### 3.3 ****Scalability & Modularity****

* Class-based design (Python OOP) allows component reusability
* Use of DAO (Data Access Object) pattern for DB access
* Easy to switch DB engine with configuration abstraction

#### 3.4 ****Reliability & Maintainability****

* Custom exceptions for better debugging (e.g., ProductNotFoundError)
* Centralized error logging
* Modular file structure: services, data access, models

#### 3.5 ****Testability****

* Unit tests with unittest framework
* Test scripts for login, cart, order modules
* Mock data used to simulate DB operations

**Simple Table Representation**

| **Type** | **Requirement** |
| --- | --- |
| **Performance** | Pages should load within 2–3 seconds for a smooth experience. |
| **Usability** | Interface should be easy to understand for both admins and customers. |
| **Security** | Passwords should be hashed and user inputs validated. |
| **Scalability** | The system should support more users/products in the future. |
| **Maintainability** | Code should follow OOP and be modular for easier updates. |

**4. Prioritize Requirements**

Not all features need to be developed at once. Prioritizing helps us **focus on the most important tasks first**, especially when working within a time limit or small team.

We use the **MoSCoW method** here:

* **M – Must Have**: Critical features
* **S – Should Have**: Important, but not urgent
* **C – Could Have**: Nice to have, only if time permits
* **W – Won’t Have Now**: Can be added in future



**Prioritized Feature Table:**

| **Feature** | **Priority** | **Reason** |
| --- | --- | --- |
| User Registration/Login | **Must Have** | Basic for all operations |
| Add/View Products | **Must Have** | Core feature of e-commerce |
| Add to Cart & Checkout | **Must Have** | Enables the purchase process |
| Place/View Orders | **Must Have** | Shows order history |
| Search/Filter Products | **Should Have** | Improves user experience |
| Admin – Add/Edit/Delete Products | **Must Have** | Required to manage inventory |
| Admin – View Orders | **Should Have** | Helps monitor sales |
| Profile Update | **Could Have** | Useful but not urgent |
| Product Reviews/Ratings | **Could Have** | Enhances shopping experience |
| Wishlist or Save for Later | **Could Have** | Optional feature |
| Dark Mode UI or Themes | **Won’t Have Now** | Not needed in first version |

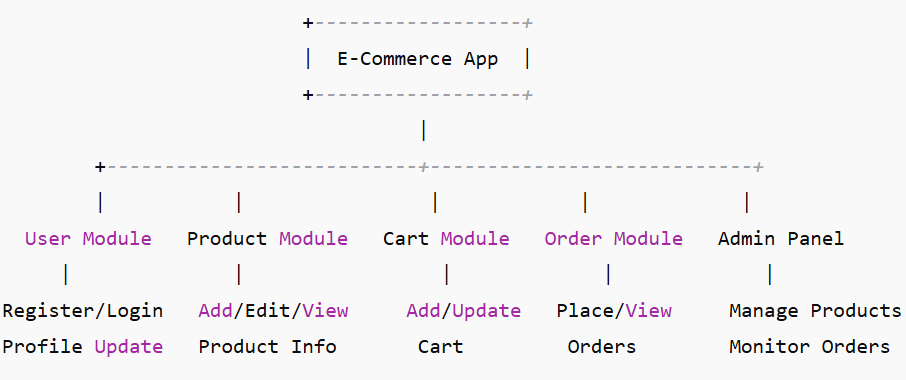
### 5. SQL Keywords/Concepts Applied in the Project

* **DDL:** CREATE TABLE, ALTER, DROP
* **DML:** INSERT, SELECT, UPDATE, DELETE
* **Constraints:** PRIMARY KEY, FOREIGN KEY, NOT NULL, AUTO\_INCREMENT
* **Joins:** INNER JOIN, LEFT JOIN for fetching related records
* **Transactions:** START TRANSACTION, COMMIT, ROLLBACK

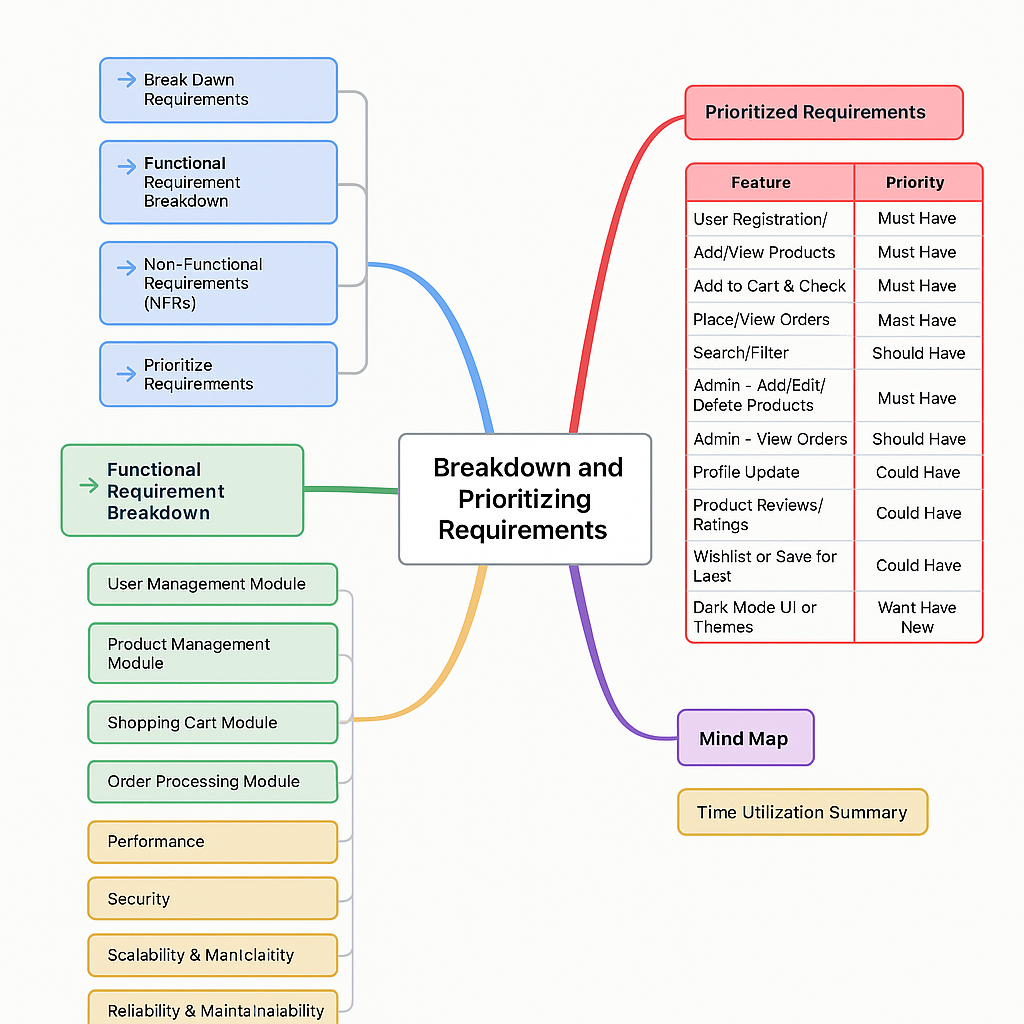
### 6. Python Techniques and Design Patterns

* **OOP Design:** Class-based approach for each entity (User, Product, Cart)
* **Database Integration:** mysql.connector for SQL communication
* **Exception Handling:** Try-except blocks with logging
* **Data Abstraction:** DAO pattern separates logic from DB access
* **CLI App Structure:** Menu-driven program using input() and loops

**7.How each Application work:**

****

**Conclusion:**



By breaking down and prioritizing the requirements, we make the development process **clear, focused, and manageable**. We build the **most important features first** and leave optional or complex features for future versions. The use of technical best practices from OOP, database normalization, and structured programming underpins the robustness and extensibility of the system.This method improves **project planning, teamwork, and successful delivery**.