

**PES University, Bengaluru**

**Department of Computer Science and Engineering**

**Object Oriented Analysis and Design PROJECT**

**UE22CS352B: OOAD PROJECT**

**Section : H**

**NAME: RITHVIK KASHYAP B S SRN: PES2UG22CS450**

**NAME: ROHAN MULLAY SRN: PES2UG22CS455**

**NAME: RAYAAN SATAAR SRN: PES2UG22CS437**

**Problem Statement (Synopsis)**

With the growing demand for online shopping, there is a need for a robust, scalable, and maintainable eCommerce platform. This project aims to develop a simplified eCommerce system that allows users to browse products, add them to a cart, place orders, and track their status. Administrators or backend logic can manage inventory and stock levels, and users can view their order history.

The system incorporates essential business logic for product management, order processing, and inventory control. Additionally, it uses software design principles and patterns to ensure scalability, maintainability, and ease of extension.

**🔹 Models**

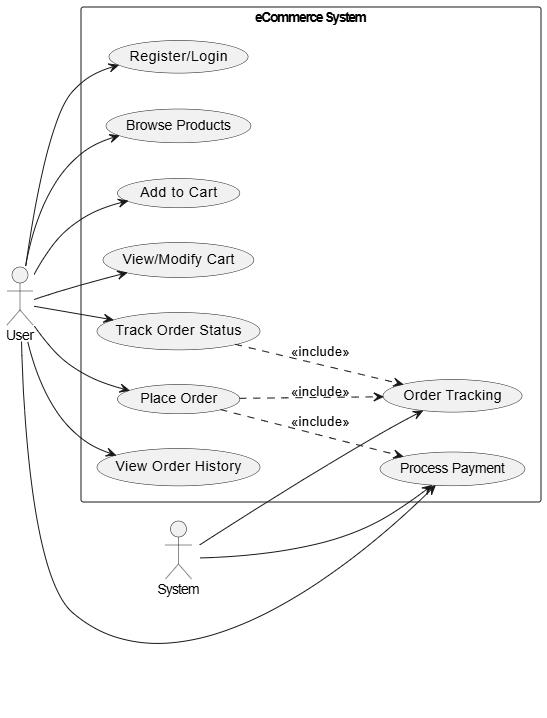
**➤ Use Case Model**

**Actors:**

* **User (Customer):** Can register, log in, browse products, manage cart, place orders, and view order history.
* **System:** Handles backend tasks like order processing, inventory updates, and sending order notifications.

**Use Cases:**

* **Register/Login**
* **Browse Products**
* **Add to Cart**
* **View/Modify Cart**
* **Place Order**
* **Track Order Status**
* **View Order History**



**➤ Class Model (High-level Overview)**

**User**

* id, username, password, email, address, phone
* Relationships: Has many Orders, has a Cart

**Product**

* id, name, description, price, stock\_quantity, category

**Cart**

* Contains multiple CartItems, associated with one User

**Order**

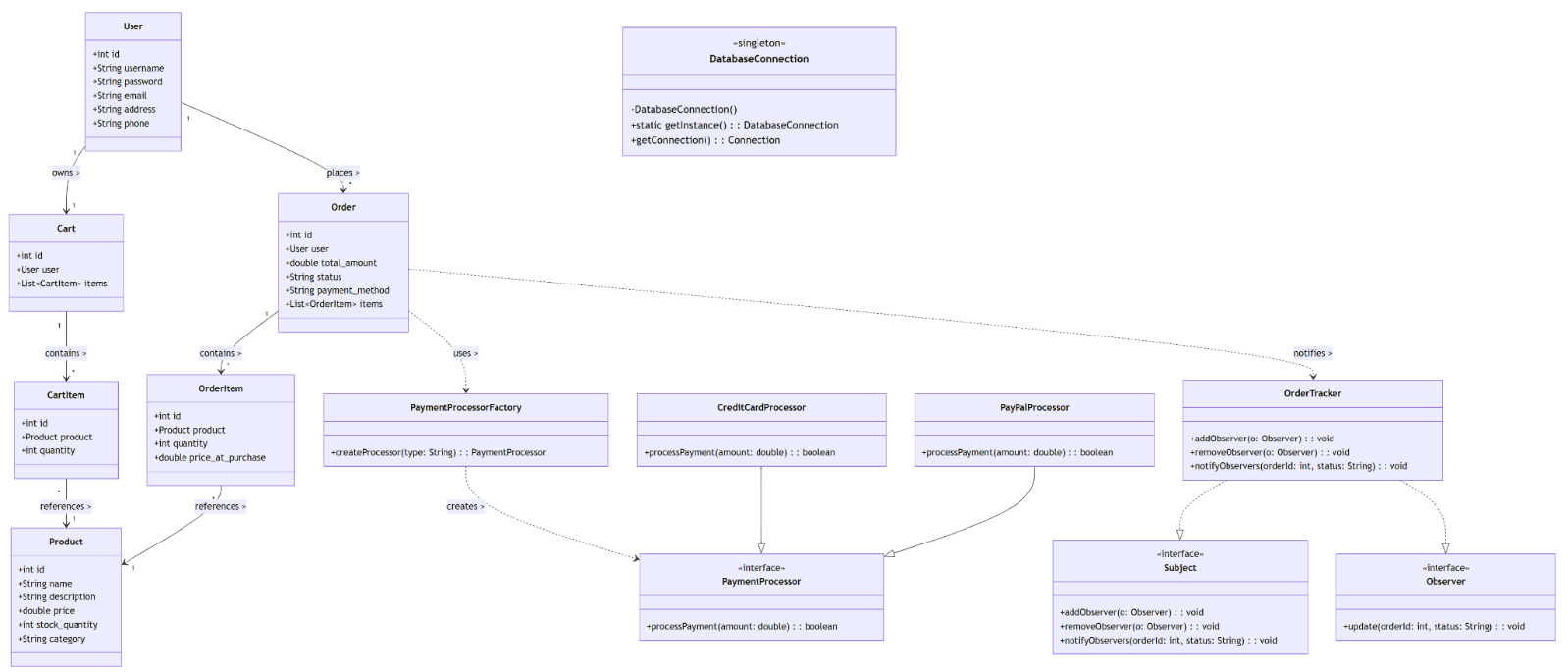
* id, user\_id, total\_amount, status, payment\_method
* Relationships: Contains many OrderItems

**OrderItem**

* Contains product\_id, order\_id, quantity, price\_at\_purchase

**CartItem**

* Contains user\_id, product\_id, quantity



**🔹 Architecture Patterns, Design Principles, and Design Patterns Used**

**1. MVC (Model-View-Controller) Architecture**

* **Model Layer (model/)**: Represents core business entities like User, Product, Order, and data access via repositories.
* **View Layer (view/)**: Handles the presentation logic, interacting with the user interface.
* **Controller Layer (controller/)**: Orchestrates interactions between the view and the model.

This separation ensures high cohesion within layers and loose coupling between them, promoting scalability and maintainability.

**2. DAO Pattern (Data Access Object)**

* Found in the Repository/ folder.
* Abstracts database operations (e.g., insert, update, delete, query) for each entity.
* Promotes Single Responsibility Principle (SRP) by separating business logic from database access logic.

**3. Factory Pattern**

* Implemented in util/factory/PaymentProcessorFactory.java.
* Used to create objects of different payment processors (e.g., CreditCardProcessor, PayPalProcessor).
* Facilitates Open/Closed Principle — easy to add new payment methods without changing client code.

**4. Singleton Pattern**

* DatabaseConnection.java ensures a single shared connection object to manage database interactions.
* Controls access to shared resources and avoids redundant connections.

**5. Observer Pattern**

* Found in util/observer/.
* OrderTracker notifies observers (e.g., user UI or notification service) when an order's status changes.
* Ensures decoupled communication between subject (order status) and observers.

**6. Design Principles Applied**

| **Principle** | **How It's Applied** |
| --- | --- |
| **Single Responsibility** | Repositories handle DB operations, Controllers handle logic, Views handle UI. |
| **Open/Closed Principle** | Factory and Observer patterns allow new types/extensions without modifying existing code. |
| **Dependency Inversion** | Controllers depend on abstractions (e.g., repository interfaces), not implementations. |
| **Don't Repeat Yourself (DRY)** | Reusable methods in repositories and shared procedures/triggers in the DB. |
| **Separation of Concerns** | Clear distinction between UI, logic, data, and utilities. |

**🔹 Individual Contributions of Team Members**

**👤 Member 1: RITHVIK KASHYAP B S SRN: PES2UG22CS450**

**Design Pattern Implemented:**  
🧱 **Singleton Pattern**

* Implemented DatabaseConnection as a Singleton to ensure a single point of database access across the application.

**Design Principle Applied:**  
✅ **Single Responsibility Principle (SRP)**

* Each class (especially in the DAO layer) is responsible for only one functionality: either handling database interaction or business logic.

**Responsibilities:**

* Designed and created the entire database schema (tables, views, procedures, triggers).
* Developed the DatabaseConnection.java Singleton utility class.
* Implemented UserRepository and ProductRepository.

**👤 Member 2: ROHAN MULLAY SRN: PES2UG22CS455**

**Design Pattern Implemented:**  
🏭 **Factory Pattern**

* Developed PaymentProcessorFactory to dynamically create different payment processor objects like CreditCard, PayPal, etc.

**Design Principle Applied:**  
✅ **Open/Closed Principle (OCP)**

* The system is open for extension (e.g., adding new payment methods) but closed for modification of existing code.

**Responsibilities:**

* Created and maintained the PaymentProcessorFactory.java.
* Developed the OrderController.java to integrate payment logic using the factory.
* Handled logic for placing orders and calculating totals.

**👤 Member 3: RAYAAN SATAAR SRN: PES2UG22CS437**

**Design Pattern Implemented:**  
🔔 **Observer Pattern**

* Built Subject, Observer, and OrderTracker to notify components when the order status is updated.

**Design Principle Applied:**  
✅ **Dependency Inversion Principle (DIP)**

* High-level modules (like controllers) depend on abstractions (Observer interfaces), not on concrete implementations.

**Responsibilities:**

* Implemented OrderTracker and observer registration/notification logic.
* Designed ProductView, CartView, and OrderView.
* Connected views to the controllers and tracked updates using the observer pattern.