System Design Document

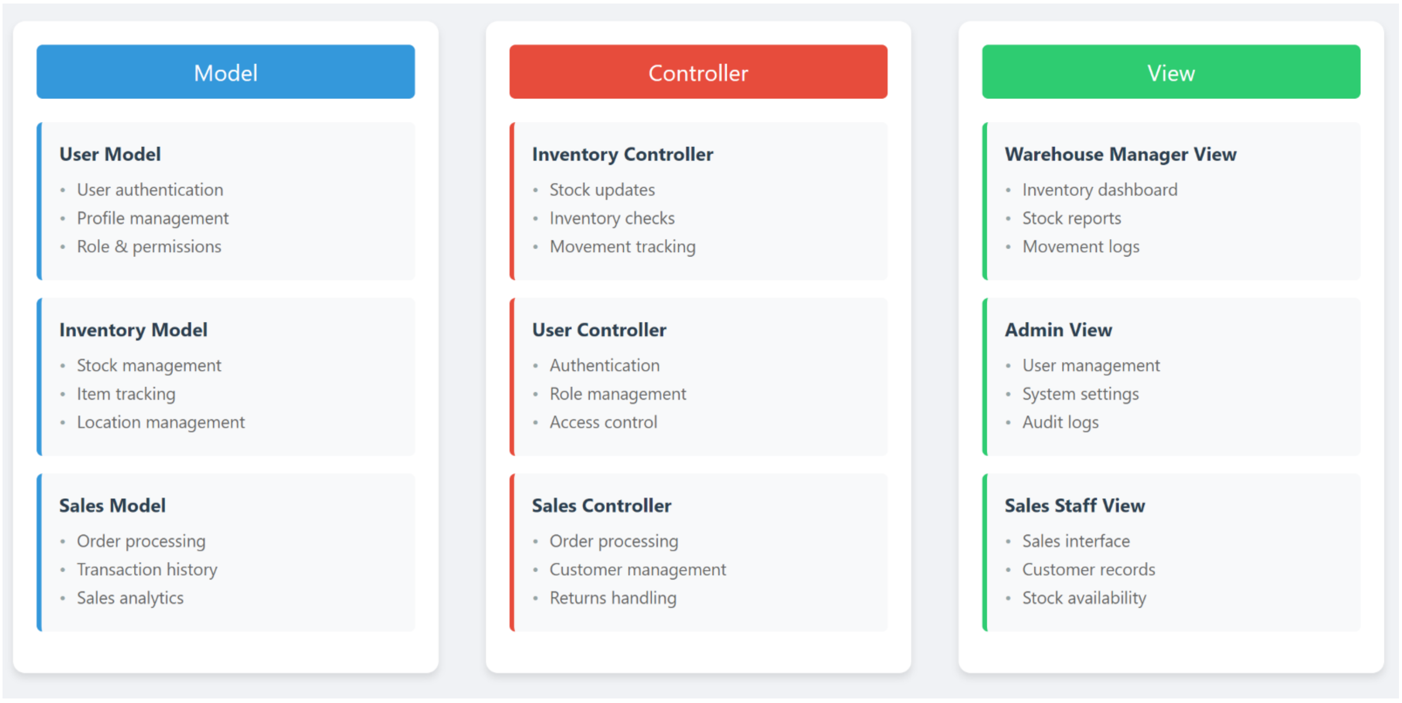
1. ARCHITECTURE DESIGN

1.1 Design Goals

- Simplicity: Keep the system easy to use and understand for all users (system admin, warehouse manager, sales staff).
- Reliability: Ensure the system works correctly without errors, especially for inventory and sales updates.
- Efficiency: Make the system fast, so users can quickly view and update data.
- Scalability: Design the system to handle more data and users if the business grows.
- Security: Protect sensitive data, like user accounts and sales records, from unauthorized access.

1.2 Overall Architecture

The Homantin Furniture Nexus System (hereinafter referred to as "the system") is based on an MVC pattern.



1.3 Technology Selection

1.3.1 Frontend Technology Selection

HTML+CSS

Responsibilities:

- Provide user interface for inventory and sales management.
- Interact with backend services via RESTful APIs

1.3.2 Backend Technology Selection

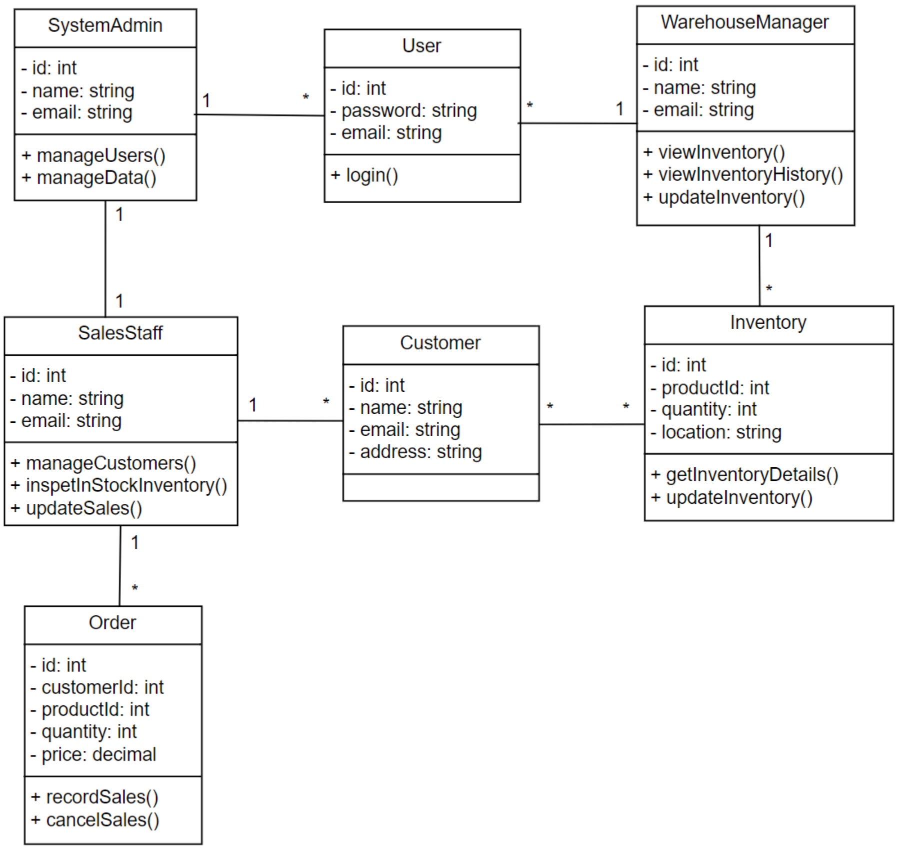
PHP + XAMPP: PHP serves as the server-side programming language, responsible for core business logic, data processing, and database interaction.

1.3.3 Database Technology Selection

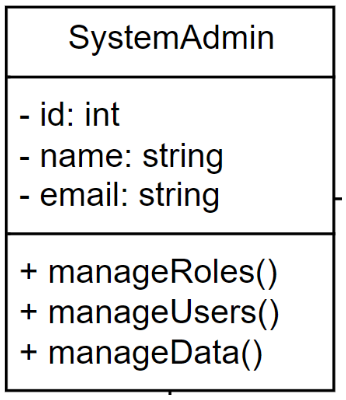
- **Type**: Relational
- **PostgreSQL**: Suitable for open-source requirements, supporting various data types and complex queries, ideal for projects with strict cost controls.

2. DETAILED DESIGN

2.1. CLASS DIAGRAMS



2.1.1 System admin



Attributes:

1. id: int
2. name: string
3. email: string

Operations:

1. manageUsers()

a) Aim: Manage user accounts

b) Precondition: Admin must be authenticated

c) Postcondition: User account changes are saved

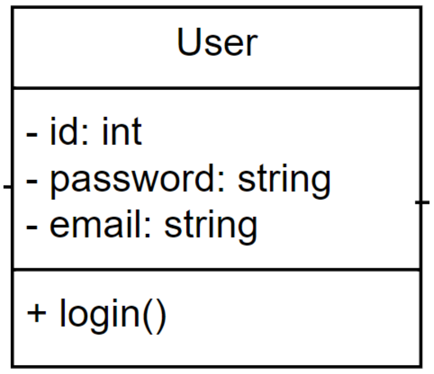
2. manageData()

a) Aim: Manage system data

b) Precondition: Admin must be authenticated

c) Postcondition: Data changes are saved

2.1.2 User



Attributes:

1. id: int
2. password: string
3. email: string

Operations:

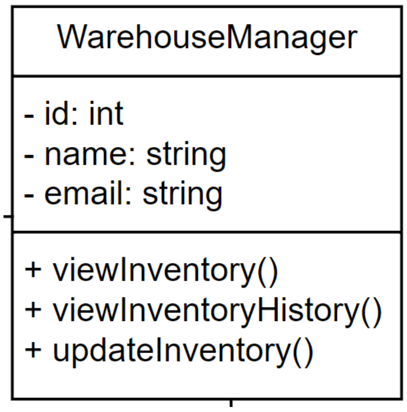
1. login()

a) Aim: Authenticate user

b) Precondition: User exists in system

c) Postcondition: User is authenticated and session created

2.1.3 WarehouseManager



Attributes:

1. id: int 2. name: string 3. email: string

Operations:

1. viewInventory()

a) Aim: Display current inventory

b) Precondition: Manager is authenticated

c) Postcondition: Inventory data is displayed

2. viewInventoryHistory()

a) Aim: Display inventory history

b) Precondition: Manager is authenticated

c) Postcondition: Inventory history is displayed

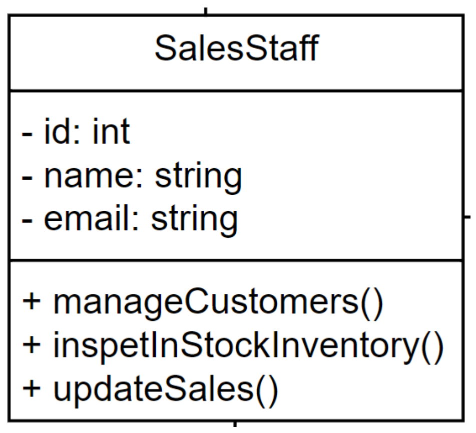
3. updateInventory()

a) Aim: Modify inventory levels

b) Precondition: Manager is authenticated and inventory exists

c) Postcondition: Inventory is updated

2.1.4 SalesStaff



Attributes:

1. id: int
2. name: string
3. email: string

Operations:

1. manageCustomers()

a) Aim: Manage customer information

b) Precondition: Staff is authenticated

c) Postcondition: Customer data is updated

2. inspectInStockInventory()

a) Aim: Check available inventory

b) Precondition: Staff is authenticated

c) Postcondition: Current inventory status displayed

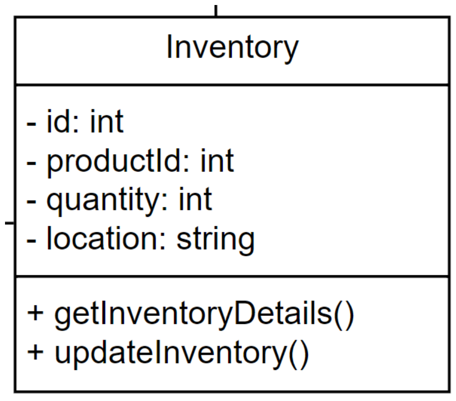
3. updateSales()

a) Aim: Update sales records

b) Precondition: Valid sale transaction exists

c) Postcondition: Sales record is updated

2.1.5 Inventory



Attributes:

1. id: int
2. productId: int
3. quantity: int
4. location: string

Operations:

1. getInventoryDetails()

a) Aim: Retrieve inventory information

b) Precondition: Inventory record exists

c) Postcondition: Inventory details returned

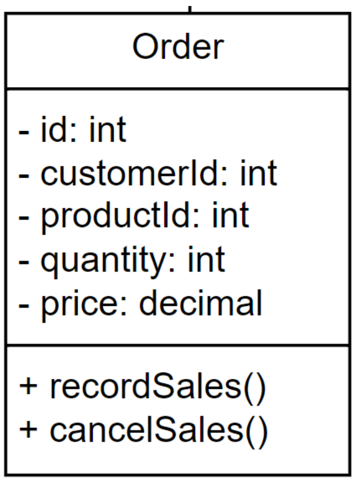
2. updateInventory()

a) Aim: Modify inventory records

b) Precondition: Valid inventory exists

c) Postcondition: Inventory record updated

2.1.6 Order



Attributes:

1. id: int
2. customerId: int
3. productId: int
4. quantity: int
5. price: decimal

Operations:

1. recordSales()

a) Aim: Record new sales transaction

b) Precondition: Valid order details exist

c) Postcondition: Sale is recorded in system

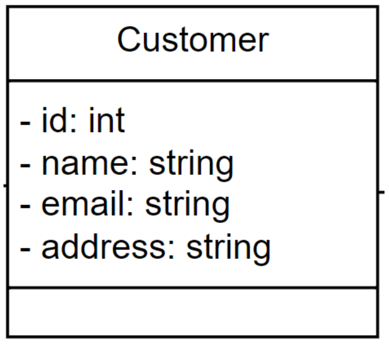
2. cancelSales()

a) Aim: Cancel existing sale

b) Precondition: Valid sale record exists

c) Postcondition: Sale is cancelled and inventory updated

2.1.7 Customer



Attributes:

1. id: int

2. name: string

3. email: string

4. address: string

2.2. DATA FLOW

- User Actions: Initiate requests from the frontend.
- HTTP Requests: Frontend sends requests to backend.
- Data Processing: Backend processes requests and interacts with the database.
- Response: Backend sends data back to the frontend for display.

2.2.1. User authentication process

- All users authenticate through the login() method
- Verify user type (administrator/warehouse/sales) and permissions

2.2.2. System administrator operation flow

- Manage all user accounts through manageUsers()
- Manage system data through manageData()
- Access all data with the highest authority

2.2.3. Warehouse administrator operation flow

- View inventory (viewInventory)
- View inventory history (viewInventoryHistory)
- Update inventory (updateInventory)
- All operations record history

2.2.4. Salesperson operation flow

- Customer management (manageCustomers)
- View in-store inventory (inspetInStockInventory)
- Update sales information (updateSales)
- Manage order data

2.2.5. Order processing flow

- Record sales (recordSales)
- Cancel sales (cancelSales)
- Automatically update related inventory

2.2.6. Data Interaction Rules

- Each user type has restricted access to specific data flows
- All inventory changes are tracked and logged
- Sales operations automatically trigger inventory updates
- System maintains data consistency across all operations