# E-Commerce Platform with ASP.NET Core

## Project Overview:

The E-Commerce Website is a modern and interactive web platform developed using ASP.NET Core, tailored to offer users a smooth and intuitive online shopping journey. It allows customers to effortlessly explore available products, add items to their cart, complete secure payments, and monitor the status of their orders.

On the administrative side, a dedicated dashboard empowers admins to manage product listings, categories, and order records efficiently.
To enhance performance, scalability, and code maintainability, the system adopts an N-Tier architectural approach and leverages both the Repository and Unit of Work patterns to streamline data access and business logic separation. With ASP.NET Identity securing user authentication and payment services like Stripe or PayPal integrated, the platform ensures a secure, scalable, and reliable e-commerce experience.

# Project Guidelines:

# 1. Project Planning & Management

### a) Project Proposal

- **Overview**

  With the increasing reliance on digital solutions in modern commerce, the demand for reliable and accessible online shopping platforms is greater than ever. This project is focused on building a feature-rich e-commerce website that allows users to browse products, make secure transactions, and track their purchases with ease. An administrative backend will also be developed, allowing seamless management of inventory, orders, and user interactions. The solution will leverage contemporary web development tools to ensure it meets high standards of performance, security, and scalability—providing a solid foundation for both consumers and administrators.

- **Objectives**
  - Design and develop a customer-centric e-commerce platform with a clean and user-friendly interface.
  - Integrate robust user authentication for both customers and administrators.
  - Build a responsive product catalog with efficient search and filter capabilities.
  - Enabling a smooth **cart and checkout** process with multiple payment options.
  - Enabling order tracking and efficient handling of delivery operations.
  - Equipping admins with tools to oversee inventory, manage categories, and process customer orders.

- **Scope**
  - User account creation and secure login mechanisms.
  - Product listing with support for search, filtering, and categorization.

- Functionality for adding items to both cart and Wishlist.
- Secure checkout and payment processing.
- Order history and real-time tracking.

## b) Project Plan

### Week 1: Initial Setup & Database Design

- Establish N-Tier Architecture (Presentation, Business Logic, Data Access).
- Integrate ASP.NET Identity for authentication (registration and login).
- Design the SQL Server database schema.
- Configure Entity Framework Core using Database-First approach.
- Set up version control using GitHub.

### Week 2: Product & Category Management

- Develop logic for managing products and categories within the MVC structure.
- Apply Repository and Unit of Work patterns for clean data handling.
- Build Admin views for product and category CRUD operations.
- Add product search and filter functionality for customers.

### Week 3: Shopping Cart & Role-Based Access Control

- Implement shopping cart system using session storage.
- Create role-based access control for Admins and Customers.
- Develop UI for cart management and initiate checkout process.

### Week 4: Order Processing & Payment Integration

- Implement order placement, order history, and tracking mechanisms.
- Integrate Stripe for secure online payments.
- Ensure transactional integrity with Unit of Work pattern.
- Display order status and confirmation to users.

### Week 5: Frontend Development – Customer UI Enhancements

- Build responsive customer-facing UI (Product pages, Checkout, Profile).
- Use JavaScript/jQuery for interactive features.
- Enhance Admin panel with order management tools and DataTables.
- Add notifications and improve user feedback using Bootstrap and Toastr.

### Week 6: Admin Panel Enhancements & DataTables Integration

- Apply server-side and client-side validation.
- Sanitize inputs and protect sensitive operations.
- Optimize database performance and MVC routing.
- Enforce security best practices (HTTPS, role checks).

### Week 7: Deployment & Final Review

- Conduct testing: manual, functional, and cross-browser.
- Conduct functional and integration testing to ensure modules interact correctly.
- Fix UI bugs and ensure consistent user experience.
- Deploy the application to Microsoft Azure or similar hosting.
- Perform User Acceptance Testing (UAT) and finalize documentation.

## c) Task Assignment & Roles – Defined responsibilities for team members

## d) Risk Assessment & Mitigation Plan

### 1. Technical Risks

| Risk | Potential Impact | Mitigation Strategy |
|---|---|---|
| **Logic Errors in MVC Layers** | Faults in Controller, View, or Model logic could break key workflows or display wrong data. | Apply unit and integration testing across MVC components; conduct peer reviews and step-by-step debugging. |
| **Payment Gateway Integration Failures** | Failed transactions or incomplete order records due to incorrect integration. | Validate Stripe integration thoroughly; simulate various payment scenarios before production; maintain logs for issue tracking. |
| **Slow Page Rendering / Performance Lag** | Reduced responsiveness, especially on product listing or checkout pages | Optimize Razor Views, apply server-side pagination and caching for repeated queries, and monitor response time |
| **Data Loss or Corruption** | Potential loss of order or user data due to improper transaction handling. | Use transaction scopes for critical operations, apply regular SQL backups, and test rollback functionality. |

## 2. Security Risks

| Risk | Potential Impact | Mitigation Strategy |
|---|---|---|
| **Unauthorized Access (Hacking, SQL Injection, XSS Attacks)** | User data compromise, system breaches. | Use **JWT authentication**, enforce **input validation & sanitization**, and apply **OWASP security best practices**. |
| **Data Breaches** | Exposure of customer payment and personal details. | Encrypt sensitive data, **implement HTTPS**, and limit **admin access permissions**. |
| **Weak Passwords & Account Takeovers** | Unauthorized users gaining control of accounts. | Enforce **strong password policies**, **multi-factor authentication (MFA)**, and **lockout mechanisms** for failed attempts. |

## 3. Deployment & Operational Risks

| Risk | Potential Impact | Mitigation Strategy |
|---|---|---|
| **Server Downtime or Deployment Issues** | Website becomes inaccessible to users. | Use **CI/CD pipelines** to automate deployments and test before pushing live. Deploy on **Azure with auto-scaling**. |
| **Post-Deployment Bugs** | Users may encounter unexpected issues. | Conduct **User Acceptance Testing (UAT)** before going live and set up **hotfix deployment plans**. |

## e) KPIs (Key Performance Indicator)

### 1. System Performance KPIs:

- **Page Load Speed:** Measures how fast web pages load for users.
- **System Uptime:** Tracks system availability and downtime incidents.
- **Database Query Efficiency:** Evaluates how quickly queries return results.
- **Controller Response Time:** Checks how fast MVC controllers handle requests.

### 2. Security KPIs:

- **Failed Login Attempts:** Monitors repeated login failures for suspicious
- **Protection Against SQL Injection & XSS:** Ensures the system is secure from common attacks and Security vulnerabilities.
- **SSL Certificate & Data Encryption:** Verifies secure data transmission and storage practices.

### 3. User Experience KPIs:

- **User Adoption Rate:** Measures how many users actively engage with the platform after registration.
- **Cart Abandonment Rate:** Tracks the percentage of users who leave without completing a purchase.
- **Customer Satisfaction Score (CSAT):** Evaluates user feedback on overall platform experience.

# 2. Literature Review

- **Feedback & Evaluation** – Lecturer's assessment of the project.
- **Suggested Improvements** – Areas where the project can be enhanced.
- **Final Grading Criteria** – Breakdown of marks based on documentation, implementation, testing, and presentation.

# 3. Requirements Gathering

***Stakeholders:***

1. **Customers**: End-users who will browse, purchase, and interact with the platform.
2. **Admin**: Responsible for managing products, categories, orders, and user accounts.

**Stakeholder Needs:**

- **Customers**:
- Easy registration and login process.
- Ability to browse and search products. o Secure payment and order processing.
- Notifications for offers, orders, and shipments.
- **Admin**:
- Manage products, categories, and orders efficiently.
- Block/unblock user accounts.
- Generate and manage discount coupons.
- **System**:
- Validate user credentials securely.
- Send notifications and emails (e.g., reset password, order updates).
- Ensure system reliability and performance.

### *User Stories:*

1. **As a customer**, I want to register and log in so that I can access my account and make purchases.
2. **As a customer**, I want to browse products by category and search for specific items so that I can find what I need.
3. **As a customer**, I want to add products to my cart and proceed to checkout so that I can complete my purchase.
4. **As a customer**, I want to receive notifications about my order status so that I can stay informed.
5. **As an admin**, I want to manage products and categories so that I can keep the inventory up-to-date.
6. **As an admin**, I want to block/unblock user accounts so that I can manage user access.

## *User Cases:*

1. **USER – Login**

   **Goal:** Authenticate users so they can access their accounts.
   **Summary:** Users log in by entering valid credentials. If incorrect, the system prompts them to retry. Users can access password recovery or registration options if needed.
   **Preconditions:** User has an account or has completed registration/recovery.
   **Postconditions:** Successful login grants access; failed login keeps them on the login page.
   **Alternate Scenarios:** Includes account creation and password reset flows. Security checks may appear after multiple failed attempts.

2. **USER – Logout**

   **Goal:** Allow users to securely end their session.
   **Summary:** Users log out to prevent unauthorized access. The system terminates the session and returns the user to the login page.
   **Preconditions:** User is logged in.
   **Postconditions:** User is logged out and must re-authenticate to access protected features.
   **Alternate Scenarios:** System may auto-logout after a period of inactivity.

3. **USER– Update Account**

   **Goal**: Allow users to edit their account information.
   **Summary**: Users can modify their name, email, profile picture, etc. Changes are validated and saved.
   **Preconditions**: User is logged in and authorized to edit their info.
   **Postconditions**: Updated account details are reflected in the system.
   **Alternate Scenarios**: Account deletion may require a different use case.

4. **USER – View Account Details**
   **Goal:** Let users review their account information.
   **Summary:** Users can view details like name, email, and profile photo. Authorized users may also view others' details.
   **Preconditions:** User is authenticated and has permission to view the data.
   **Postconditions:** User sees the data; changes (if any) are synced system-wide.

5. **USER – Change Password**

**Goal:** Improve account security by letting users update their password.
**Summary:** Users provide their current password and set a new one. The system verifies and updates the password.
**Preconditions:** User is logged in and knows their current password.
**Postconditions:** Password is updated; the old one is no longer valid.
**Alternate Scenarios:** If the user forgets their password, they should use the "Forgot Password" flow instead.

**USER: Choose Category**

- **Initiator**: Customer
- **Goal**: Select a product category to filter or browse content.
- **Preconditions**:
  - Customer is on a page requiring category selection.
  - Customer has permission to view categories.
- **Postconditions**:
  - System filters and displays content based on selected category.
- **Basic Flow**:
  - Customer opens the page.
  - System shows list of categories.
  - Customer selects a category.
  - System displays relevant content.

---

**USER: Display Bill**

- **Initiator**: Customer
- **Goal**: View and manage bill information.
- **Preconditions**:
  - Customer has an account and an associated bill.
  - Customer has permission to view billing details.
- **Postconditions**:
  - Customer views their bill and related details.
- **Basic Flow**:
  - Navigate to Billing page.
  - System shows bill list.
  - Customer selects a bill.
  - System displays bill details (amount, due date, history).
- **Alternate Flows**:
  - Filtering/searching for multiple bills.
  - Showing paid bills with confirmation and history.

---

**USER: Make Payment**

- **Initiator**: Customer
- **Goal**: Complete a payment for a bill/order.
- **Preconditions**:
  - Valid payment method available.
  - Amount and reason for payment are known.

- **Postconditions**:
  - Payment is processed and confirmed.
  - **Basic Flow**:
    - Navigate to "Make Payment" page.
    - Enter payment details.
    - Confirm payment.
    - System processes and confirms transaction.
  - **Alternate Flows**:
    - Update or modify payment method.
    - Verify order/invoice before payment.

---

**User: Place Order**

- **Initiator**: Customer
- **Goal**: Complete a product purchase.
- **Preconditions**:
  - Products selected in the cart.
  - Logged in and accessing checkout.
- **Postconditions**:
  - Order is placed, payment processed, confirmation sent.
  - Inventory and shipping status updated.

- **Basic Flow**:
- Open cart/checkout page.
  1. Review and update products.
  2. Enter shipping, billing, payment info.
  3. Confirm order.
  4. Payment processed.
  5. Confirmation email sent.
  6. Inventory and shipping updated.
- **Alternate Flows**:
  - Handle invalid payment.
  - Prompt for payment correction and retry.

---

**User: Proceed to Checkout**

- **Initiator**: Customer
- **Goal**: Finalize purchase and proceed with transaction.
- **Preconditions**:
  - Products are added to cart.
  - User is on Cart page.
- **Postconditions**:
  - Transaction completed and order confirmed.
- **Basic Flow**:
  1. Go to Checkout page.
  2. Review cart and total cost.

3. Enter shipping, billing, and payment info.
4. System verifies and processes payment.
5. Confirmation displayed if successful; error shown if not.

- **Alternate Flows**:
  - Modify cart before checkout.
  - Apply coupon/discount.

---

**User: Replace a Product**

- **Initiator**: Customer
- **Goal**: Replace a defective or incorrect product.
- **Preconditions**:
  - Product received and issue identified.
  - Contact made with customer service.
- **Postconditions**:
  - Replacement order processed.
  - Customer can track replacement delivery.
- **Basic Flow**:
  1. CSR verifies customer and order info.
  2. CSR confirms reason for replacement.
  3. CSR gives return instructions.
  4. Customer returns product.
  5. CSR verifies returned product.
  6. If verified, replacement is processed.
- **Alternate Flows**:
  - CSR helps with return or pickup.
  - Refund process guided if requested instead.

**US8: Search Product**
**Initiator:** Customer
**Goal:** Search for a specific product in the application.
**Preconditions:**
- Customer is looking for a specific product or item.
- The application has a search feature.

**Postconditions:**
- Customer has found the desired product or item.
- Application has provided accurate search results.

**Basic Flow:**
1. Customer navigates to the search bar.
2. Customer enters a search query.
3. Application processes the query and returns relevant results.
4. Customer reviews results and selects a product.
5. Customer views product details and may proceed to purchase.

**Alternate Flows:**
- If no results are found, the system suggests alternatives or shows recommendations.
- If results are irrelevant, customer can refine query or apply filters.

---

**User: View Today's Offers**

**Initiator:** Customer
**Goal:** View current offers and promotions.
**Preconditions:**
- Customer is using the application.
- The application has current offers available.

**Postconditions:**
- Customer can view available offers and associated details.
- Offers remain viewable until they expire or are removed.

**Basic Flow:**
1. Customer navigates to “Today’s Offers” page.
2. System displays list of current offers.
3. Customer selects an offer.
4. System displays offer details (e.g., discount, conditions).

**Alternate Flows:**
- If desired offer isn’t visible, customer may use filters/search.
- If offer is time-limited, system displays countdown timer.

---

**User: Submit Feedback**
**Initiator:** Customer
**Goal:** Provide feedback or suggestions.
**Preconditions:**
- Customer is using the application.
- Customer has permission to submit feedback.

**Postconditions:**
- Feedback is saved for future improvements.
- Features relying on feedback reflect submitted suggestions.

**Basic Flow:**
1. Customer opens the feedback page.
2. System displays a feedback form.
3. Customer enters feedback.
4. System validates the input.
5. If valid, feedback is saved, and confirmation is shown.
6. If invalid, an error is shown, and user is prompted to retry.

**Alternate Flows:**
- Customer may choose a specific feedback form related to certain features.

---

**User: View Cart**
**Initiator:** Customer
**Goal:** View and manage items in the shopping cart.
**Preconditions:**
- Items have been added to the cart.
- Application has a cart or checkout page.

**Postconditions:**
- Customer can review, edit, or remove items.
- Customer can proceed to checkout.

**Basic Flow:**

1. Customer opens the cart page.
2. Application displays items, quantities, and prices.
3. Customer reviews and optionally edits or removes items.
4. Customer proceeds to checkout.

**Alternate Flows:**

- If cart is empty, system shows a message.
- If there's an error, system displays a message and support options.

---

**User: Add Product to Cart**
**Initiator:** Customer
**Goal:** Add product(s) to the shopping cart.
**Preconditions:**

- Customer is logged in.
- Customer has accessed a product page.

**Postconditions:**

- Product is added to cart.
- Cart summary is updated and displayed.

**Basic Flow:**

1. Customer selects a product.
2. Product details are shown.
3. Customer enters quantity.
4. Customer clicks "Add to Cart."
5. System confirms addition with a message.

**Alternate Flows:**

- If customer wants to remove product, they use the "Remove from Cart" option in the cart.

---

**User: Add Shipping Info**
**Initiator:** Customer
**Goal:** Add shipping details for order delivery.
**Preconditions:**

- Order has been placed.
- Customer must provide shipping info.
- Customer has permission to enter shipping info.

**Postconditions:**

- Shipping info is saved.
- Order will be delivered to the provided address.

**Basic Flow:**

1. Customer navigates to the shipping info page.
2. System displays input fields (e.g., name, address, phone).
3. Customer enters and submits info.
4. System validates and saves the information.
5. Confirmation message is shown.

**Alternate Flows:**

- If info is invalid, system shows error and prompts retry.
- If customer uses saved addresses, system pre-fills fields.

**User: Manage Products**
**Initiator:** Admin
**Goal:** Manage products in the product catalog.
**Preconditions:**
• Admin has access to product management functionality.
• Product catalog is accessible.
**Postconditions:**
• Product catalog is up-to-date and accurate.
**Basic Flow:**

1. Admin navigates to the "Product Management" page.
2. System displays a list of all products with basic details (name, description, price, stock level).
3. Admin searches for specific products by criteria (e.g., name or category).
4. Admin adds new products manually or imports them from a file.
5. Admin edits existing product details (e.g., name, price, stock level).
6. Admin deletes product records from the catalog.
7. Admin manages product categories (create/edit/delete).
8. Admin manages product images (upload/edit/delete).
9. System validates product information and updates the catalog.
10. System displays confirmation of successful changes.

**Alternate Flows:**
• If product data is invalid, system shows error messages and prompts correction.

---

**User: Manage Categories**
**Initiator:** Admin
**Goal:** Manage product categories.
**Preconditions:**
• Admin has access to category management functionality.
**Postconditions:**
• Categories are accurate and updated.
**Basic Flow:**

1. Admin navigates to the "Category Management" page.
2. Admin adds new categories manually or imports from a file.
3. Admin edits existing categories (name, description, image).
4. Admin deletes category records.
5. System validates category data.
6. System displays confirmation of saved changes.

**Alternate Flows:**
• If data is invalid, system shows error and allows retry.

**User: Manage Customers Database**
**Initiator:** Admin
**Goal:** Manage the customers database.
**Preconditions:**
- Admin has permission to manage customers.
- The application has an accessible customers database.

**Postconditions:**
- Customers database is updated based on admin actions.

**Basic Flow:**
1. Admin navigates to the "Customers Database" page.
2. System presents options: add, edit, or delete customers.
3. Admin selects a task (e.g., edit a customer).
4. System displays a form or prompt to confirm deletion.
5. Admin enters data or confirms the action.
6. System updates the database and shows a confirmation message.

   **Alternate Flows:**
   - If input is missing or incorrect, system displays an error.

---

**User: Block & Unblock Customers**
**Initiator:** Admin
**Goal:** Block or unblock customer access.
**Preconditions:**
- Admin has permissions to block/unblock customers.
- Customer accounts exist in the system.

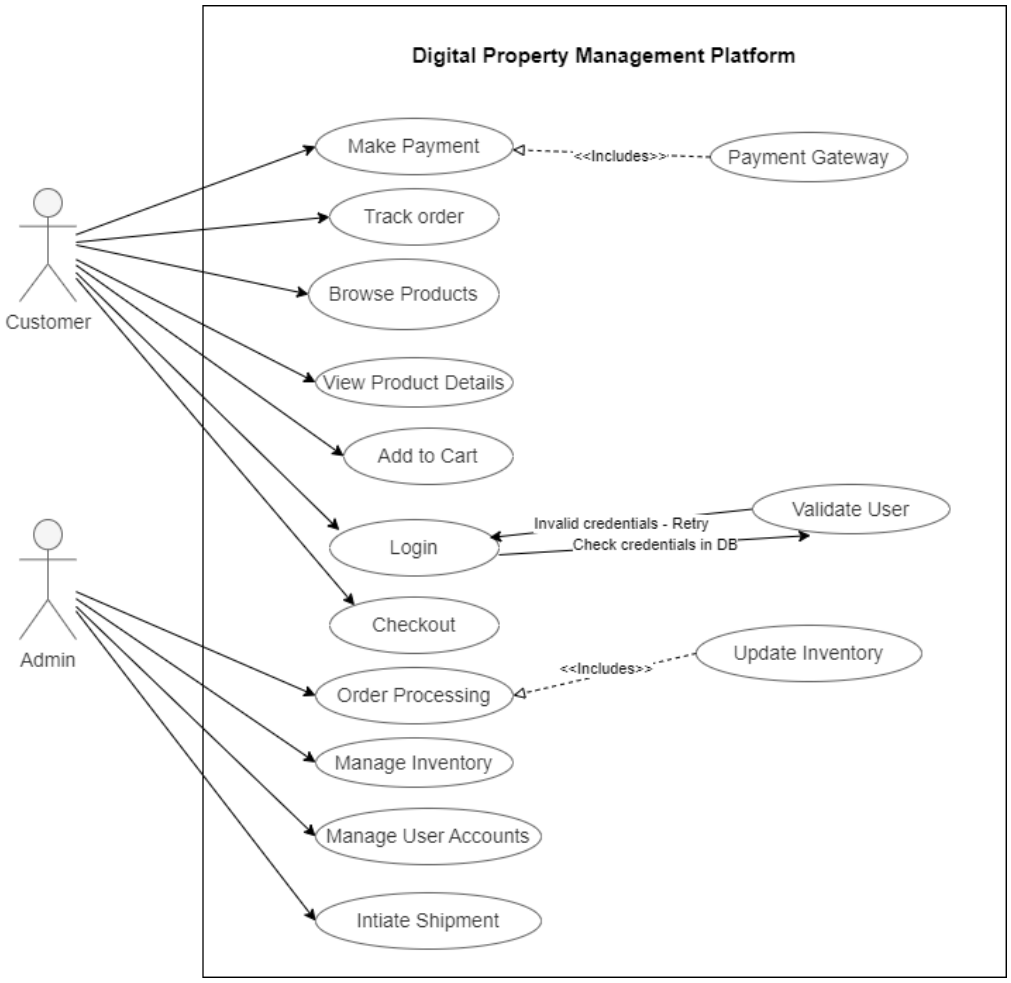
**Postconditions:**
- Customer’s account status is updated accordingly.

**Basic Flow:**
1. Admin navigates to the "Customers" page.
2. System displays a list of customers.
3. Admin selects the customer.
4. System provides options to block or unblock.
5. Admin selects action (block or unblock).
6. System updates account access status.
7. System displays confirmation of the action.

   **Alternate Flows:**
   - If an error occurs during update, system shows a message.

# 4. System Analysis & Design

## Use Case Diagram



- Functional & Non-Functional Requirements.

**1. Functional Requirements**

1.1 User Management

- User Registration
- User Login
- Forgot Password
- Edit Profile
- Delete Account
- View Profile

1.2 Product Management

- Search Products
- View Product Details
- Add Review
- Show Today's Offers

1.3 Cart Management

- Add to Cart
- Modify Cart
- View Cart

1.4 Order Management

- Checkout Process.
- Add Shipping Information.
- Choose Payment Method.
- Apply Coupon.
- Place Order.
- Cancel Order.
- Confirm Order Page.

1.5 Admin Management

- Manage Products.
- Manage Categories.
- Manage Orders.
- Block/Unblock Accounts.
- Display Bill.

1.6 Notifications

- Send Offers Notifications.
- Send Order Notifications.
- Send Shipment Updates.
- Send Reset Email.

## 2.Non-Functional Requirements

1.7 Performance

- Response Time
- Load Handling

1.8 Security

- Data Encryption
- Authentication
- Authorization

1.9 Usability

- User Interface
- Accessibility

1.10 Scalability

- Scalability

1.11 Reliability

- Uptime
- Error Handling

1.12 Compatibility

- Cross-Browser Compatibility
- Mobile Compatibility

## Software Architecture

### High-Level Design

The architecture of the clothing eCommerce platform is designed to ensure scalability, maintainability, and performance. The system follows a layered architecture that separates concerns across different components such as presentation, business logic, and data access. This structure supports core functionalities like product browsing, cart management, order processing, and secure transactions.

---

## System Components

### 1. Frontend

The frontend is the user interface of the web application, where customers can:

- Browse and search for clothing products.
- View product details, images, and prices.
- Add products to the cart or wishlist.
- Register, log in, and manage their profile.
- Place and track orders.

Developed using HTML, CSS, JavaScript, and integrated with modern frameworks to ensure responsiveness and accessibility across all devices.

### 2. Backend

The backend handles the core logic of the application. It:

- Processes user requests from the frontend.
- Manages authentication, authorization, and user sessions.
- Handles product and inventory management.
- Processes orders and payments.
- Facilitates admin functionalities (e.g., managing products, categories, users).

Built using .NET technologies, ensuring reliability and security.

### 3. Database

The database stores all persistent data, including:

- Product details (name, price, category, stock, images).
- User information (account details, addresses).
- Order history and transactions.
- Customer feedback and reviews.

A relational database system like SQL Server is used to maintain structured and consistent data.

### 4. External Services

The system integrates with third-party services such as:

- Payment Gateways (e.g., PayPal, Stripe) for secure online transactions.
- Email/SMS Services for sending order confirmations, shipping updates, and promotional messages.

---

## Interactions Between Components

- **Frontend ↔ Backend:**
  The frontend sends HTTP requests (e.g., via REST APIs) to the backend when users interact with the system (e.g., logging in, placing orders). The backend processes these and returns appropriate responses.
- **Backend ↔ Database:**
  The backend queries the database to retrieve or update data. For example, fetching product listings or storing new orders.
- **Backend ↔ External Services:**
  The backend communicates with payment gateways for transaction processing and with messaging services for notifications.

---

## Architecture Style

Model-View-Controller (MVC)

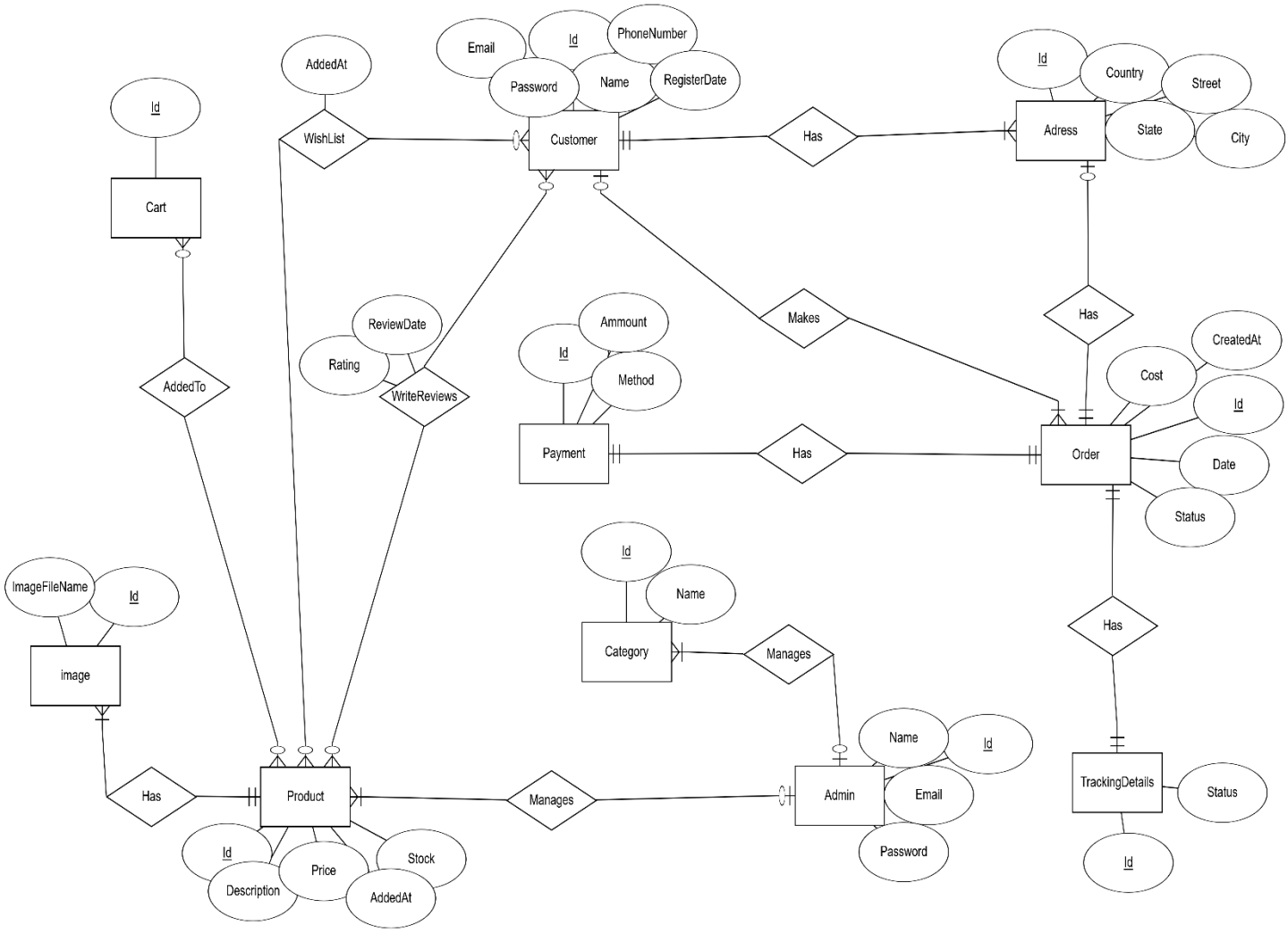
The project follows the MVC architecture pattern:

- Model: Represents the data structure and business logic (e.g., Product, Order, User models).
- View: The user interface rendered to the customer.
- Controller: Handles user input, processes requests, and updates the model or view accordingly.

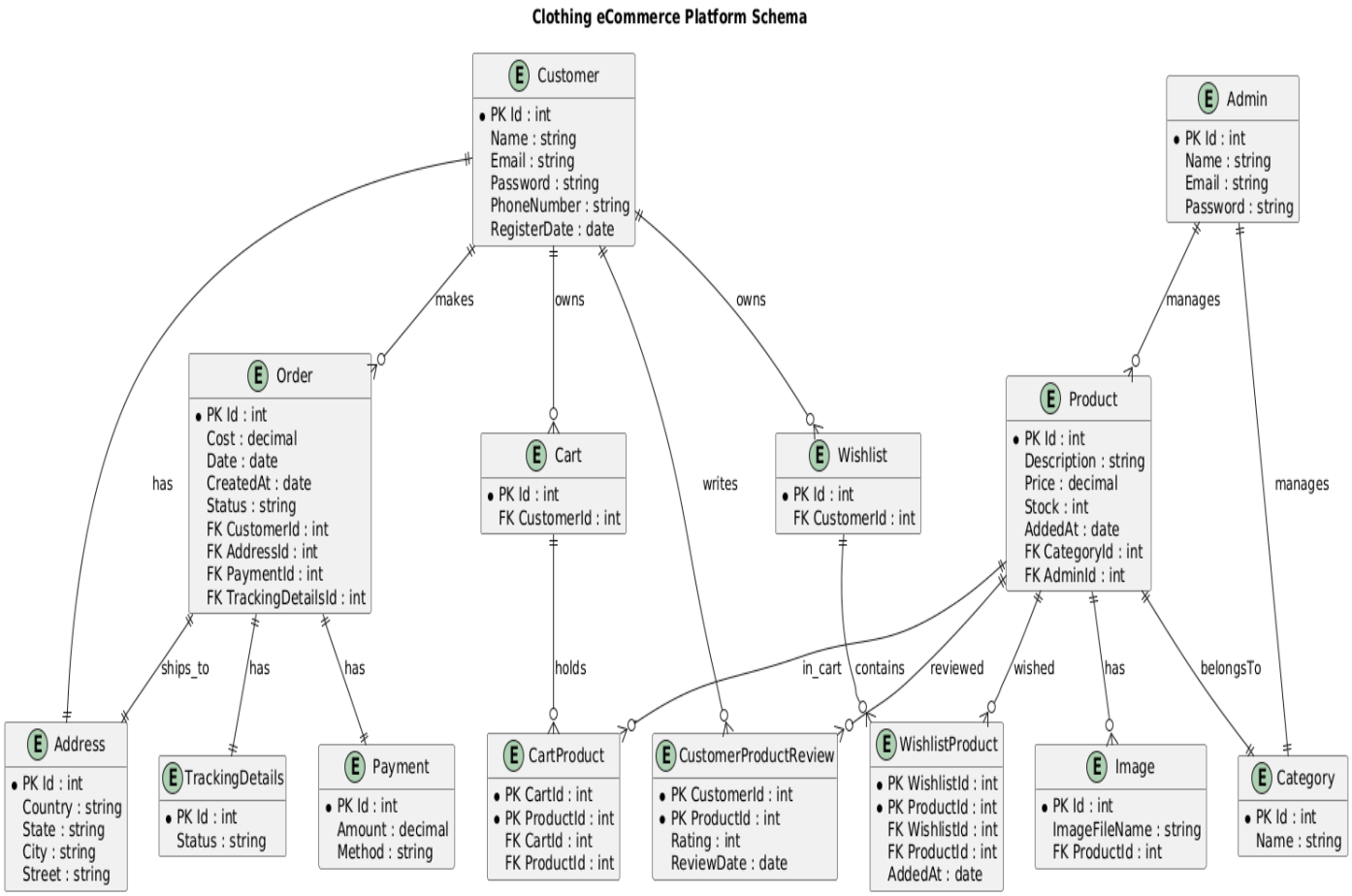
This pattern helps keep the application modular, organized, and easy to maintain.

## 2. Database Design & Data Modeling

### ER Diagram (Entity-Relationship Diagram)



## Logical & Physical Schema



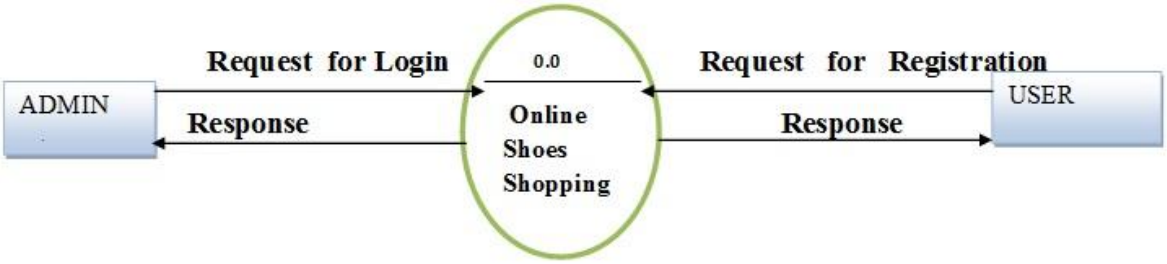

- **Normalization:**
  - The database is normalized to **3NF (Third Normal Form)** to eliminate redundancy.
  - Separate tables for Product, Product_Category, Image, and Cart ensure **data integrity**.
- **Indexes:**
  - **Primary Indexes** on PKs (e.g., `Customer_ID`, `Order_ID`, `Product_ID`).
  - **Foreign Key Indexes** for performance optimization.
- **Data Types:**
  - **VARCHAR** for textual data (e.g., names, addresses).
  - **DECIMAL** for price & cost to maintain precision.
  - **DATETIME** for timestamps (e.g., orders, registrations).

- **Relationships & Constraints:**

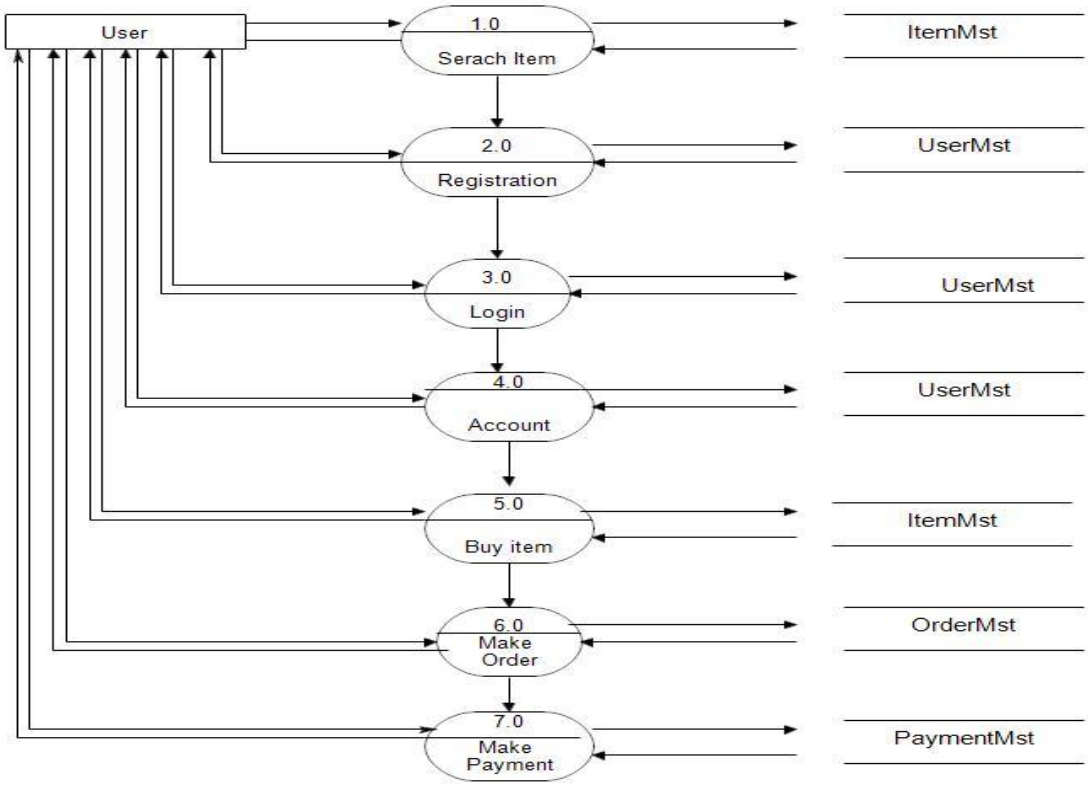
  - **One-to-Many** (Customer → Orders, Product → Category, Order → Tracking).
  - **Many-to-Many** (Cart ↔ Product using `Cart_Products`).

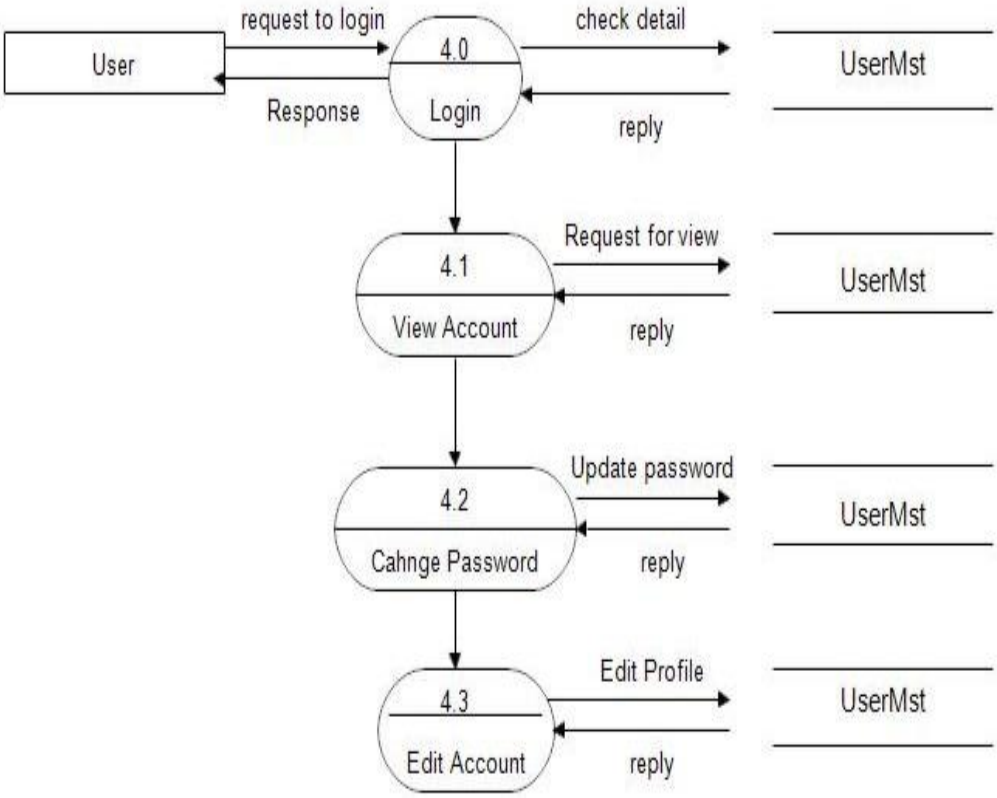
## 3. Data Flow & System Behavior

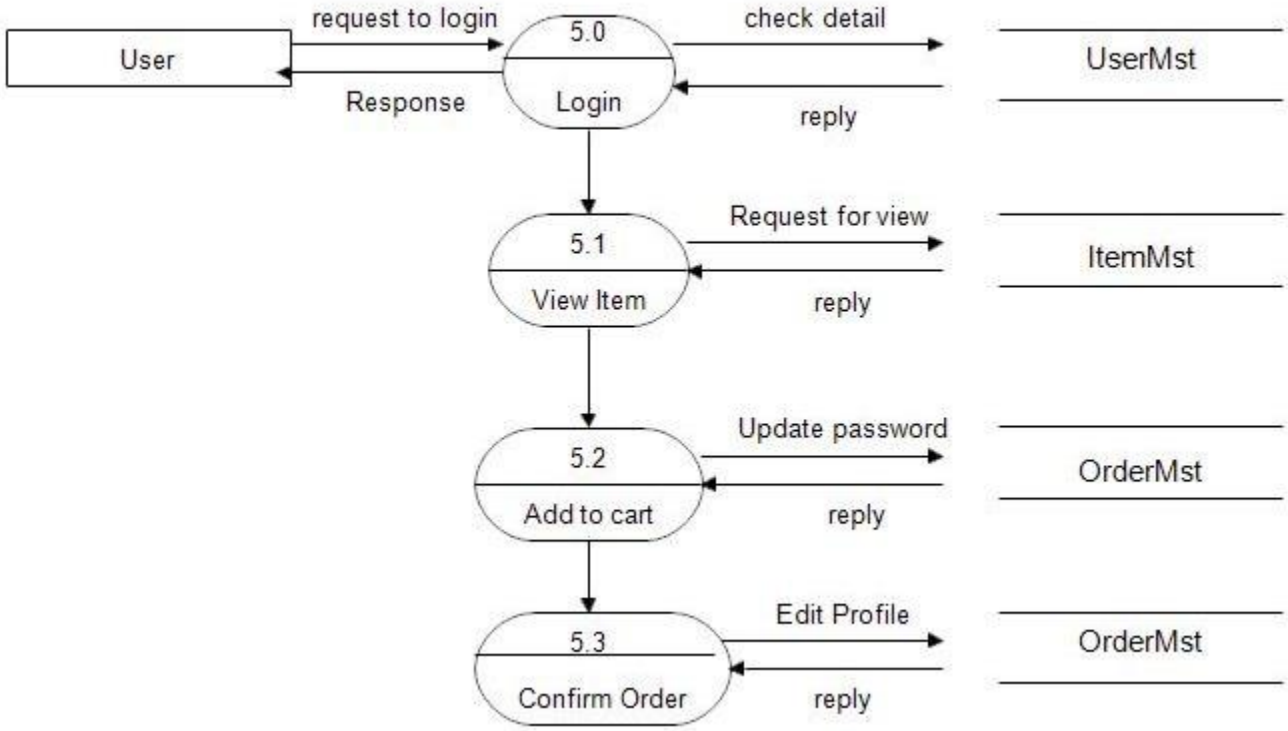
### DFD (Data Flow Diagram)



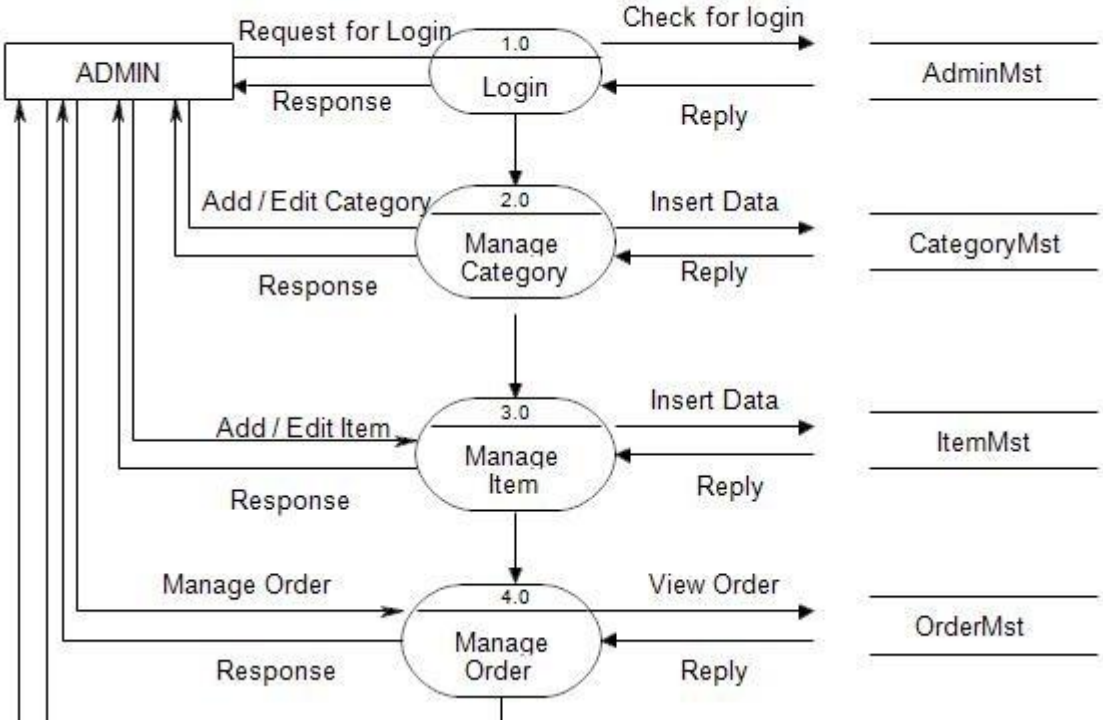

**1st Level User side DFD**



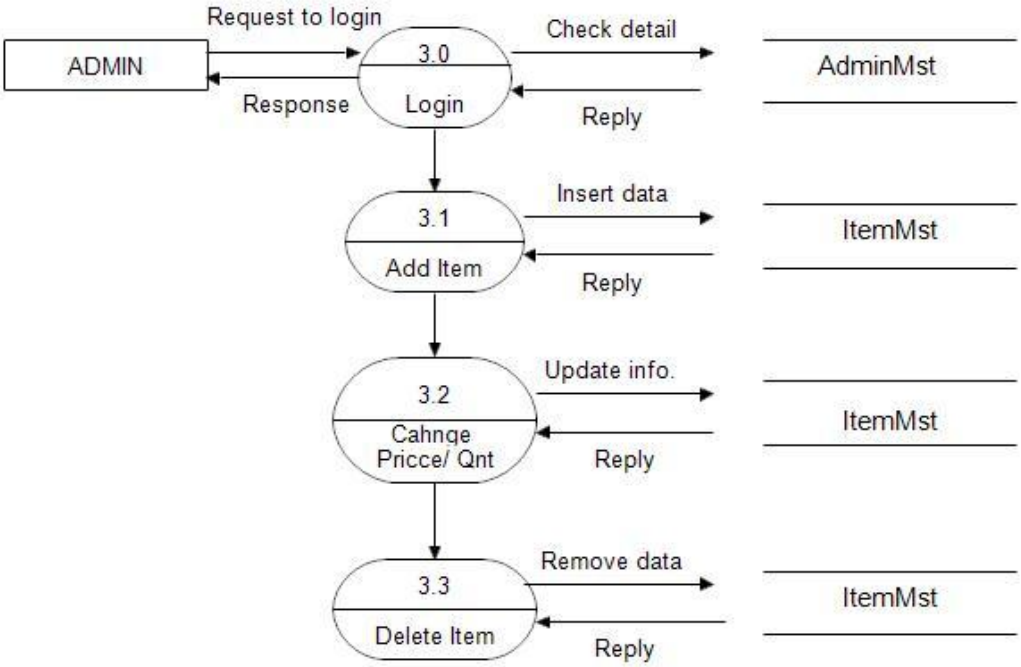

User
request to login
Response
4.0
Login
check detail
reply
UserMst
4.1
View Account
Request for view
reply
UserMst
4.2
Cahnge Password
Update password
reply
UserMst
4.3
Edit Account
Edit Profile
reply
UserMst


User
request to login
Response
5.0
Login
check detail
reply
UserMst
5.1
View Item
Request for view
reply
ItemMst
5.2
Add to cart
Update password
reply
OrderMst
5.3
Confirm Order
Edit Profile
reply
OrderMst

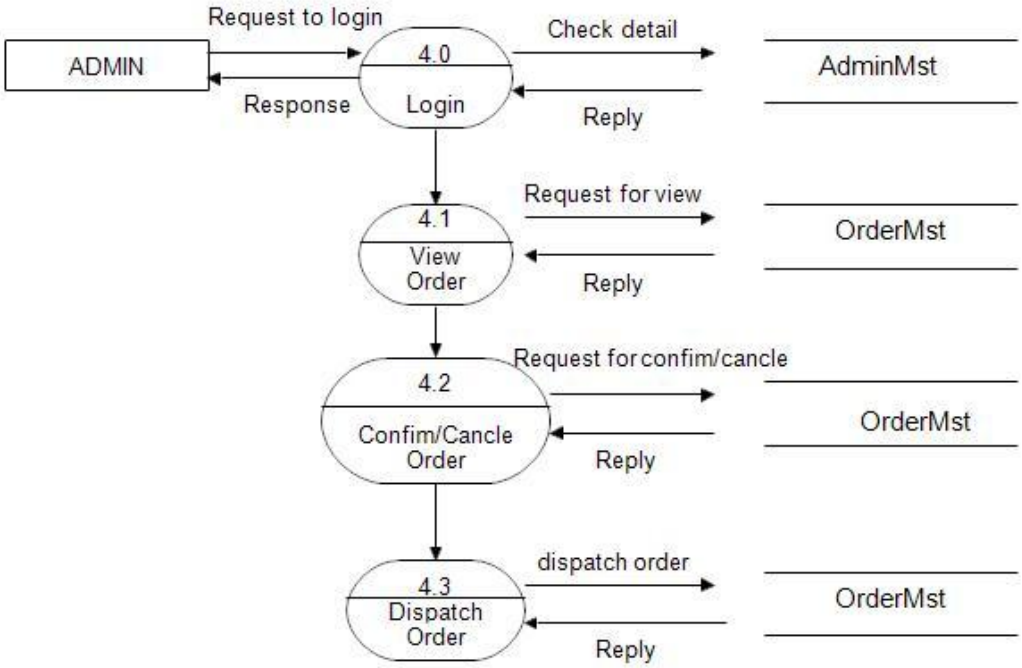
## Admin Side DFD - 1st Level



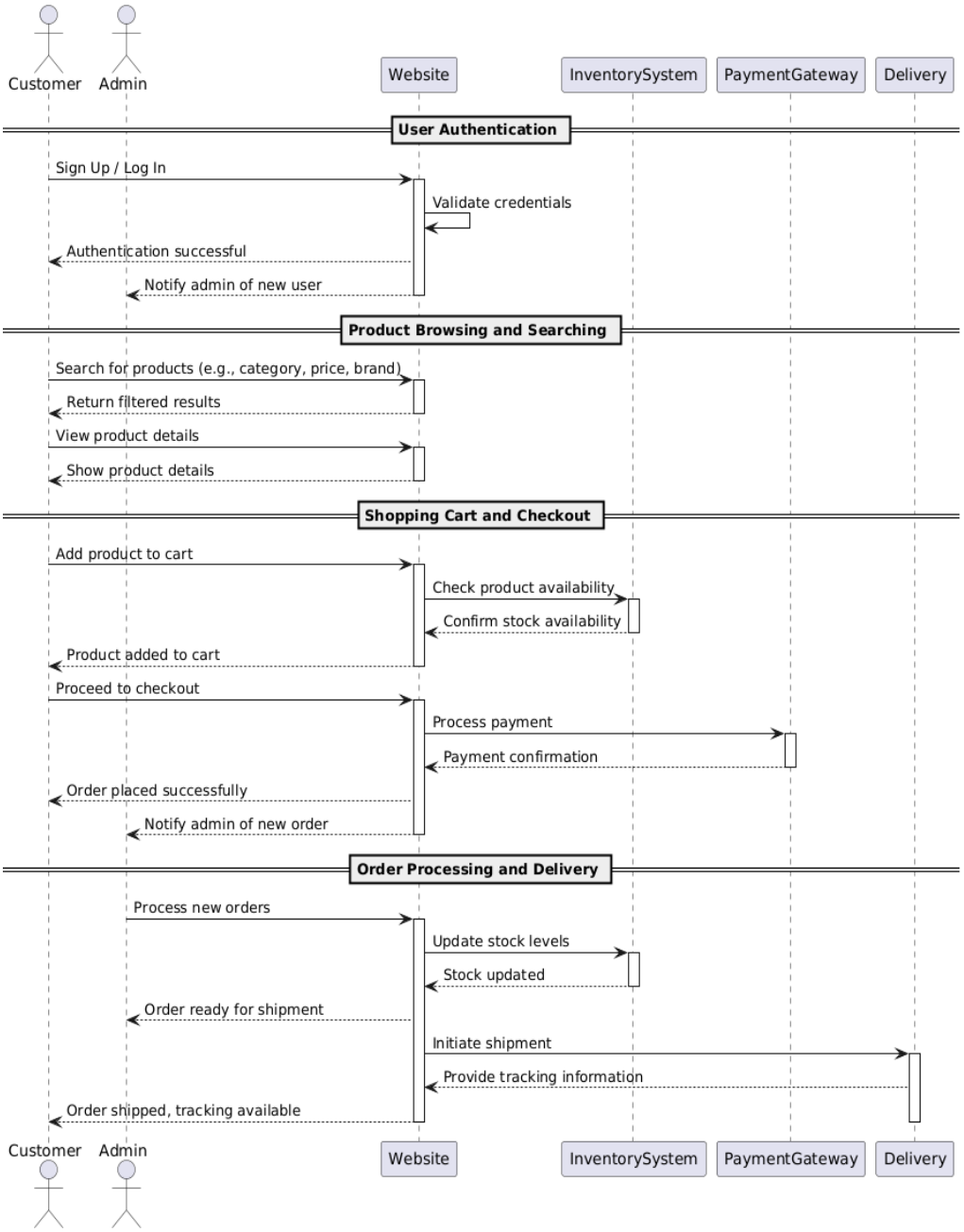
## 2nd Level Admin DFD - (3.0)



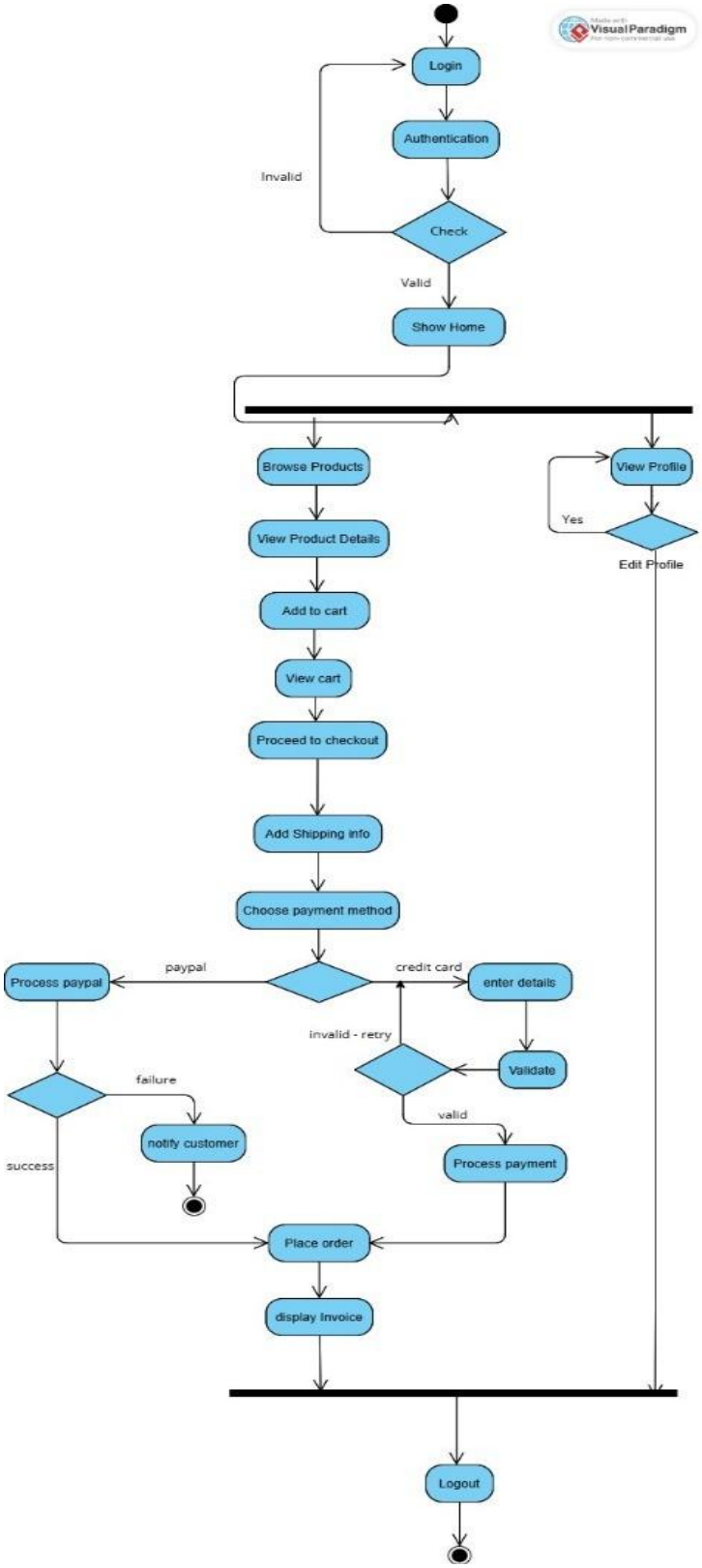
## 2nd Level Admin DFD - (4.0)



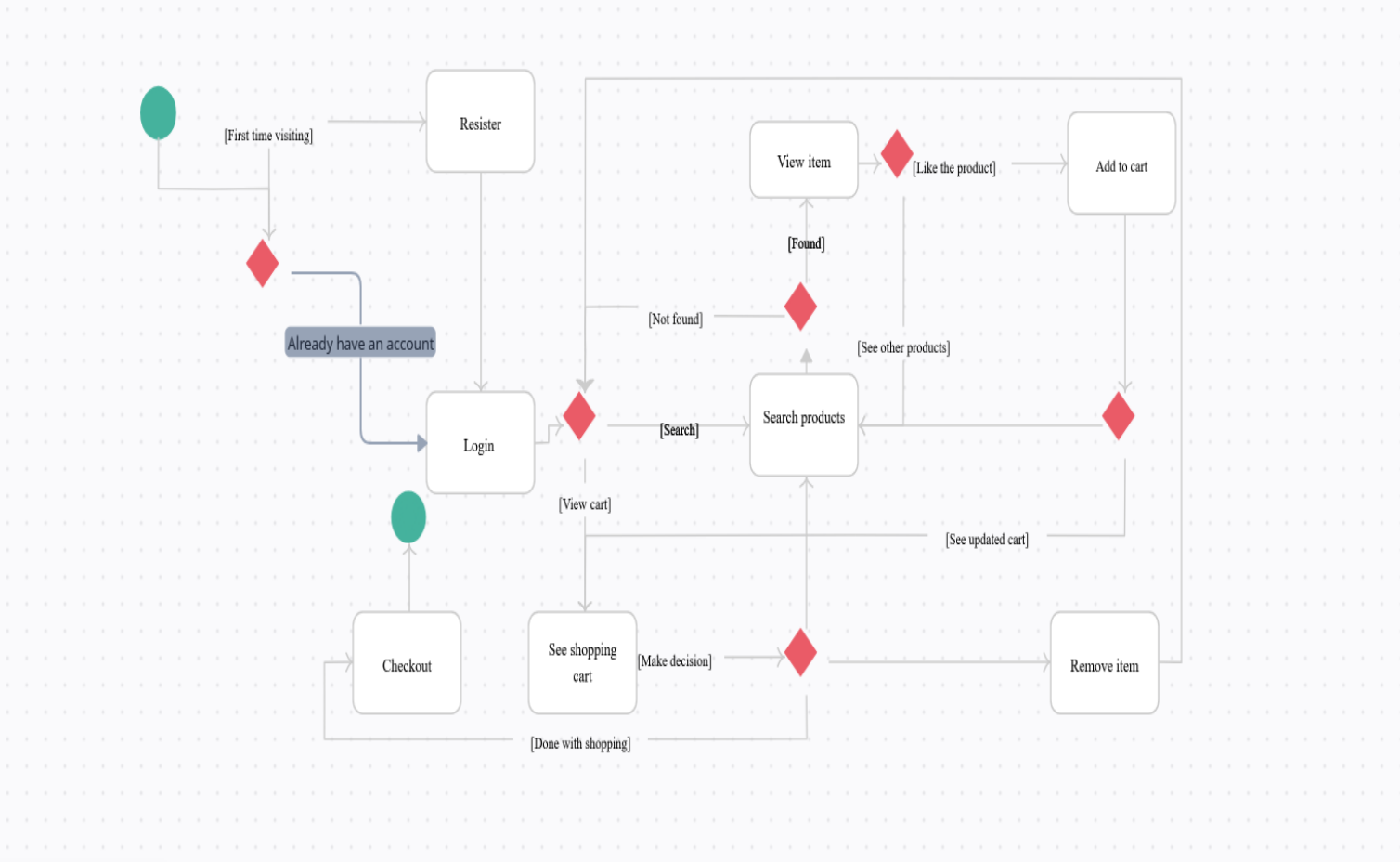
## Sequence Diagram



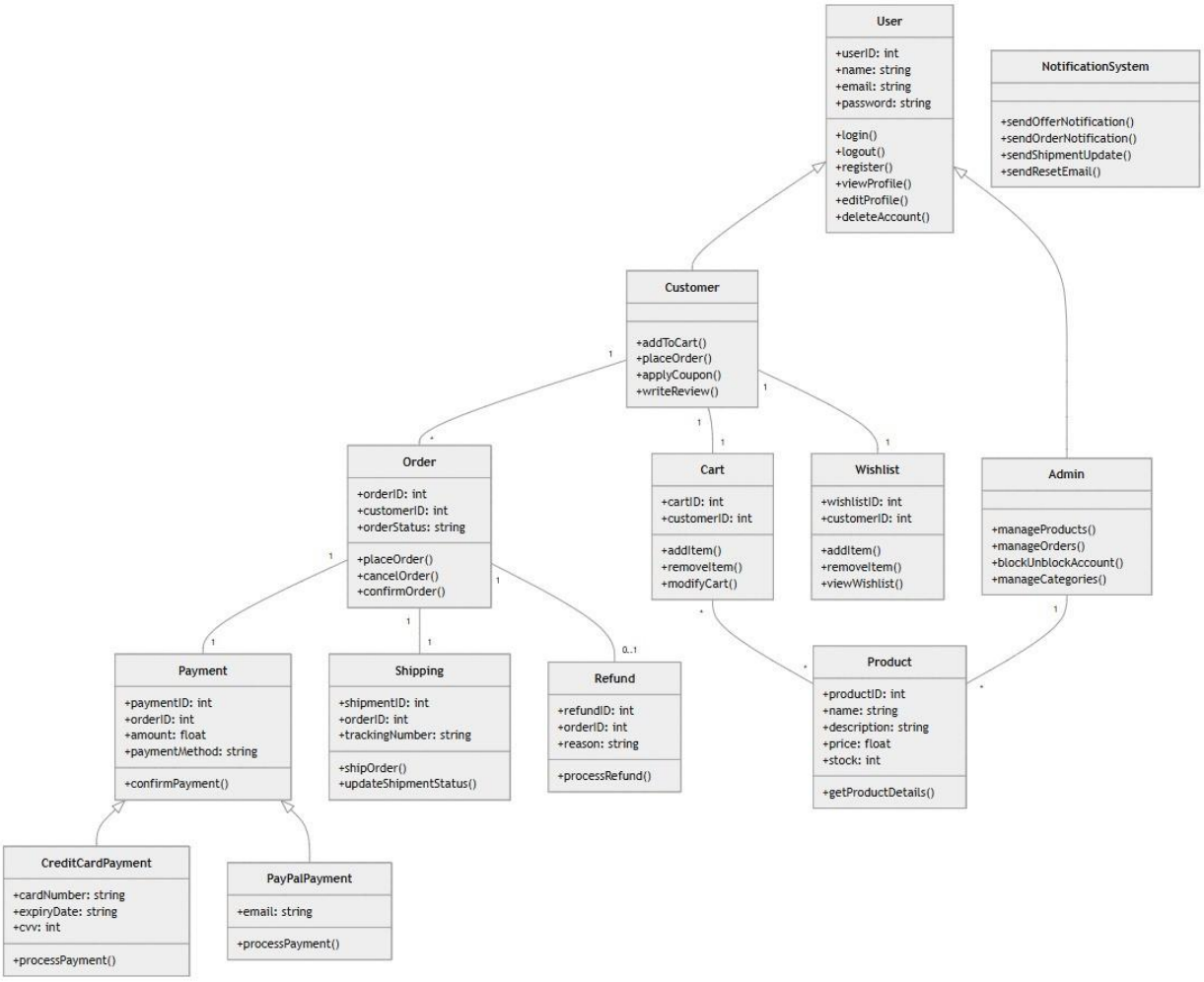
## Activity Diagram



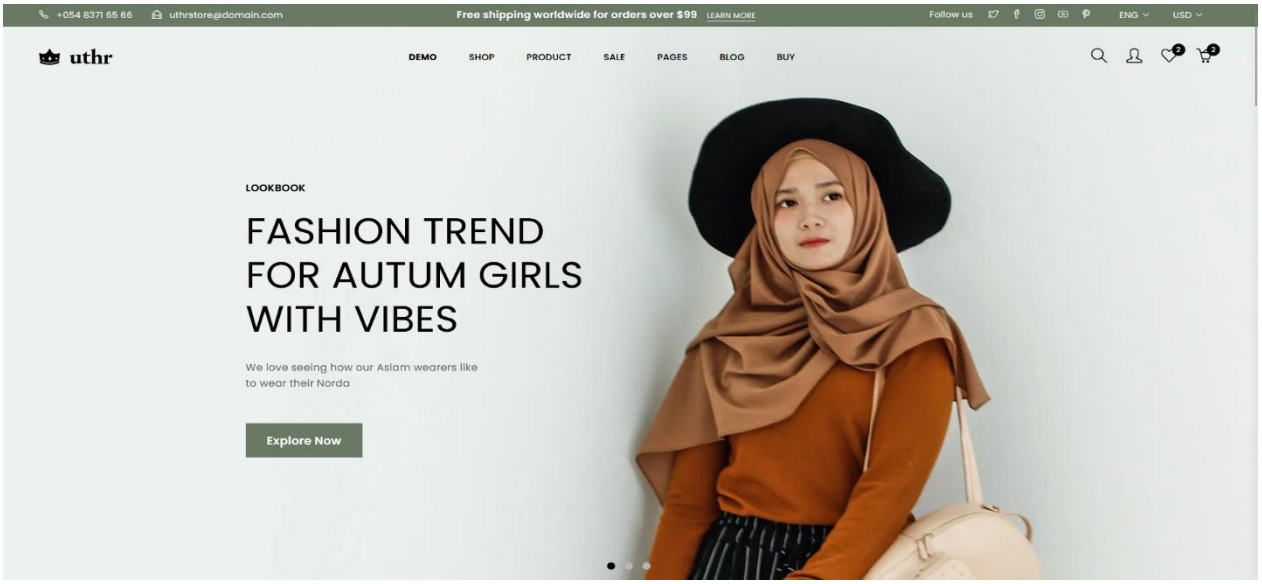
## State Diagram

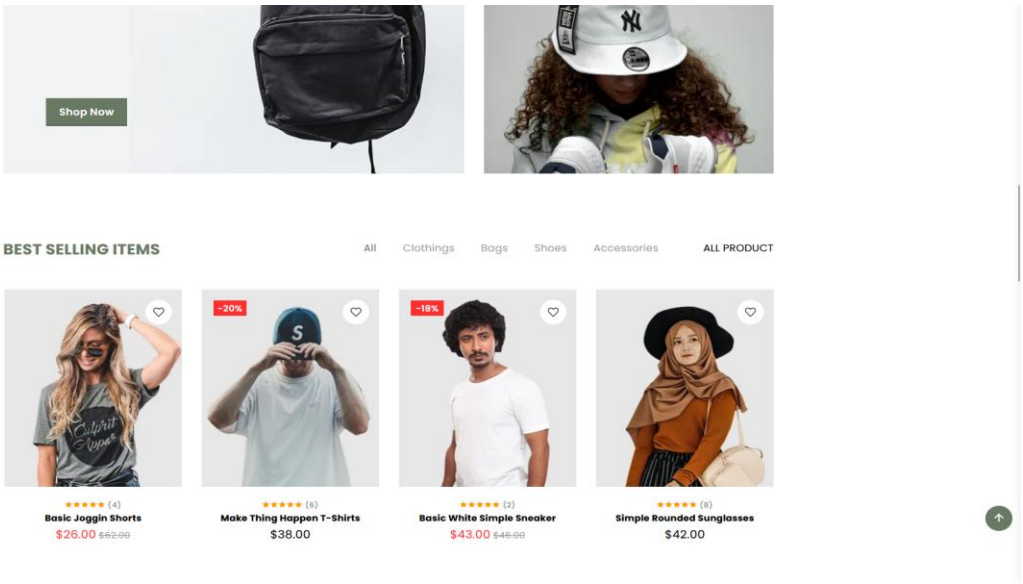


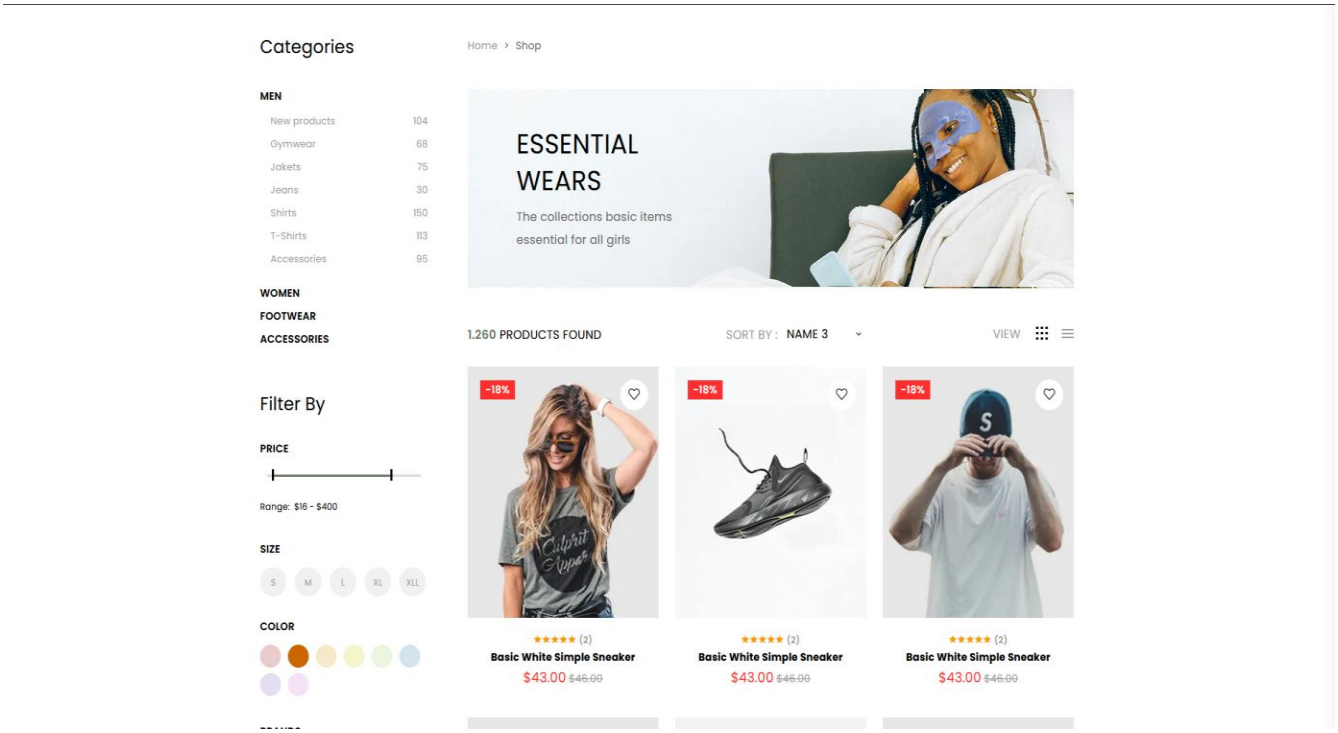

## Class Diagram

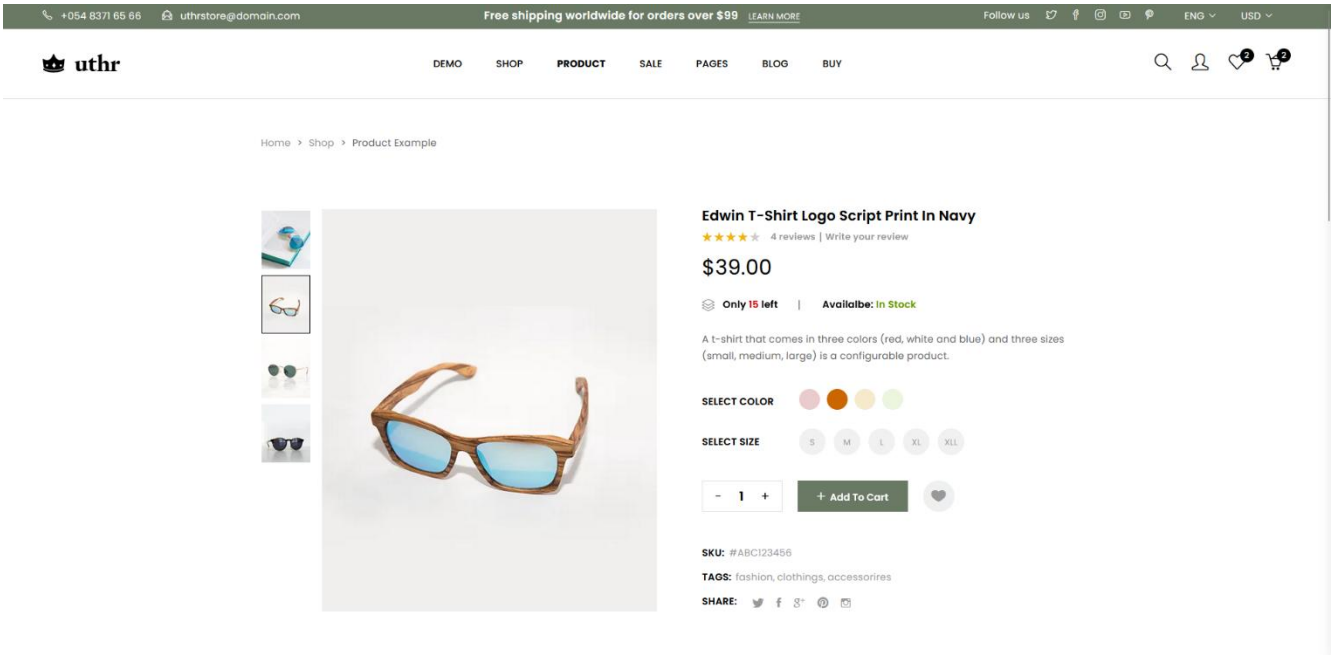

User
+userID: int
+name: string
+email: string
+password: string
+login()
+logout()
+register()
+viewProfile()
+editProfile()
+deleteAccount()
NotificationSystem
+sendOfferNotification()
+sendOrderNotification()
+sendShipmentUpdate()
+sendResetEmail()
Customer
+addToCart()
+placeOrder()
+applyCoupon()
+writeReview()
Order
+orderID: int
+customerID: int
+orderStatus: string
+placeOrder()
+cancelOrder()
+confirmOrder()
Cart
+cartID: int
+customerID: int
+addItem()
+removeItem()
+modifyCart()
Wishlist
+wishlistID: int
+customerID: int
+addItem()
+removeItem()
+viewWishlist()
Admin
+manageProducts()
+manageOrders()
+blockUnblockAccount()
+manageCategories()
Payment
+paymentID: int
+orderID: int
+amount: float
+paymentMethod: string
+confirmPayment()
Shipping
+shipmentID: int
+orderID: int
+trackingNumber: string
+shipOrder()
+updateShipmentStatus()
Refund
+refundID: int
+orderID: int
+reason: string
+processRefund()
Product
+productID: int
+name: string
+description: string
+price: float
+stock: int
+getProductDetails()
CreditCardPayment
+cardNumber: string
+expiryDate: string
+cvv: int
+processPayment()
PayPalPayment
+email: string
+processPayment()
1
*
0..1

## 4. UI/UX Design & Prototyping




Categories
Home > Shop
MEN
New products 104
Gymwear 68
Jakets 75
Jeans 30
Shirts 150
T-Shirts 113
Accessories 95
WOMEN
FOOTWEAR
ACCESSORIES
ESSENTIAL WEARS
The collections basic items essential for all girls
1.260 PRODUCTS FOUND
SORT BY : NAME 3
VIEW
Filter By
PRICE
Range: $16 - $400
SIZE
S M L XL XLL
COLOR
-18%
Basic White Simple Sneaker
$43.00 $46.00


+054 8371 65 66
uthrstore@domain.com
Free shipping worldwide for orders over $99
LEARN MORE
Follow us
ENG
USD
uthr
DEMO SHOP PRODUCT SALE PAGES BLOG BUY
Home > Shop > Product Example
Edwin T-Shirt Logo Script Print In Navy
4 reviews | Write your review
$39.00
Only 15 left
Availalbe: In Stock
A t-shirt that comes in three colors (red, white and blue) and three sizes (small, medium, large) is a configurable product.
SELECT COLOR
SELECT SIZE
S M L XL XLL
+ Add To Cart
SKU: #ABC123456
TAGS: fashion, clothings, accessorires
SHARE:

### Design Principles:

1. Consistency: Maintain uniform layouts, fonts, and colors across all pages.
2. Simplicity: Keep the design clean with a clear visual hierarchy.
3. User-Centric Approach: Prioritize ease of navigation and accessibility.
4. Responsiveness: Ensure the design adapts well to different screen sizes.

### Color Scheme:

- Primary Color: Neutral and elegant tones like white, black, and gray.
- Accent Colors: Use soft pastels or vibrant hues to highlight CTAs.
- Contrast: Ensure sufficient contrast for readability.

### Typography:

- Headings: Bold and modern fonts like "Playfair Display" or "Montserrat."
- Body Text: Clean and readable fonts like "Roboto" or "Lato."
- Font Sizes: Maintain a hierarchy (e.g., H1: 32px, H2: 24px, Body: 16px).

### Accessibility Considerations:

1. High Contrast: Use colors that are readable for visually impaired users.
2. Keyboard Navigation: Ensure the website is operable with a keyboard.
3. Alt Text for Images: Provide descriptive alt text for product images.
4. Readable Font Sizes: Use scalable font sizes for better readability.

## 5. System Deployment & Integration

### Technology Stack:

**Frontend:**

- Language: TypeScript, HTML, CSS
- Authentication: JWT Authentication

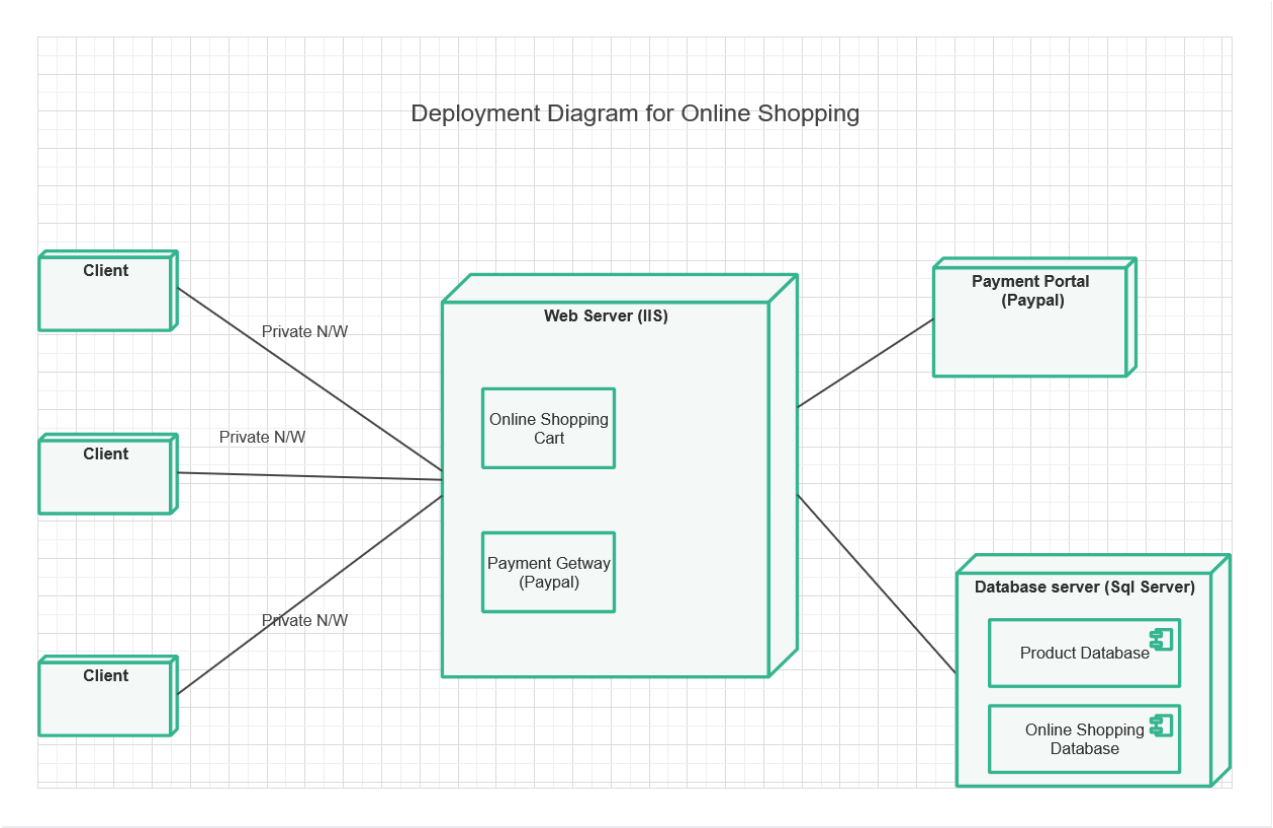
**Backend:**

- Framework: ASP.NET Core
- Language: C#
- Authentication & Authorization: Identity Server / JWT
- ORM (Object-Relational Mapping): Entity Framework Core

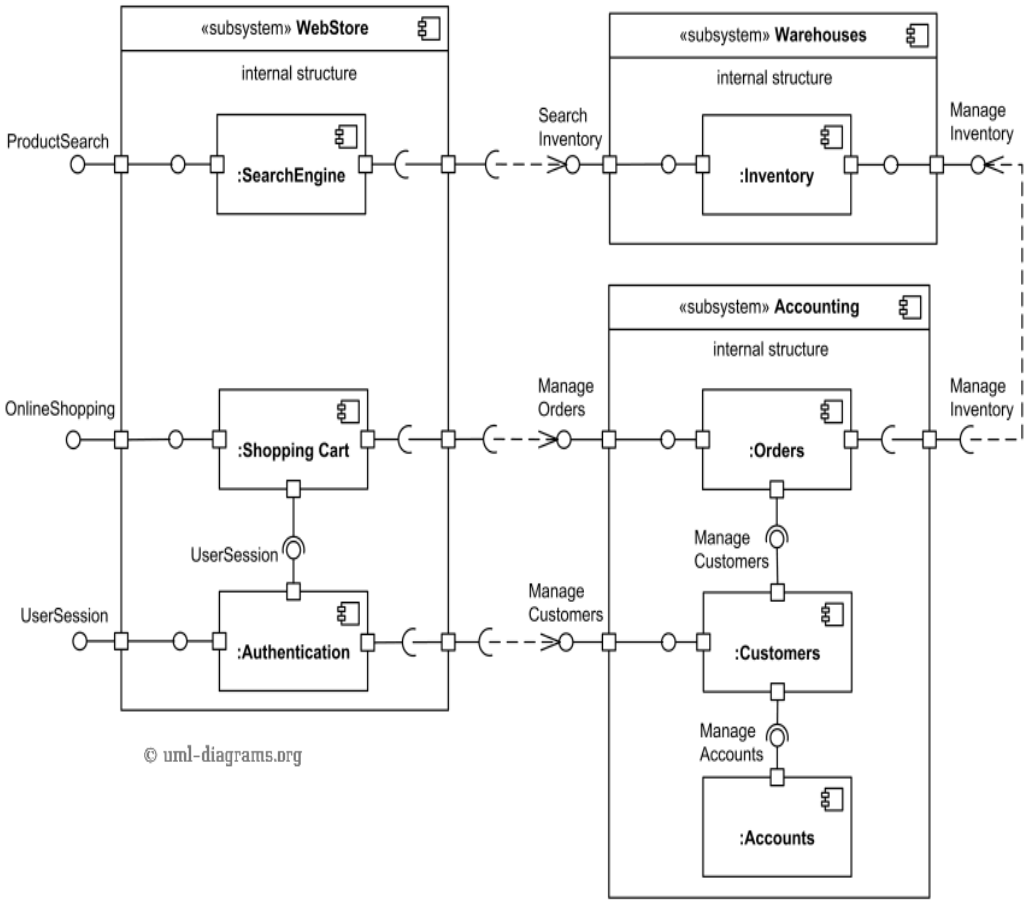
**Database:**

- Database Management System: SQL Server
- Query Language: T-SQL
- ORM Integration: Entity Framework Core
- Database Version Control: EF Migrations

### Deployment Diagram



### Component Diagram




**6. Additional Deliverables (if applicable)**

- API Documentation – If the system includes APIs, provide documentation for endpoints and usage.
- Testing & Validation – Unit tests, integration tests, and user acceptance testing plan.
- Deployment Strategy – Hosting environment, deployment pipelines, and scaling considerations.

# 5. Implementation (Source Code & Execution)

**1. Source Code**

- Structured & Well-Commented Code – Clean, maintainable, and properly documented code following best practices.

- Coding Standards & Naming Conventions – Consistent formatting, meaningful variable names, and adherence to industry standards.

- Modular Code & Reusability – Organized into reusable components, functions, and classes.

- Security & Error Handling – Secure coding practices, validation checks, and proper exception handling.

**2. Version Control & Collaboration**

- Version Control Repository – Hosted on GitHub, GitLab, or Bitbucket with a public/private repository link.

- Branching Strategy – Clear workflow (e.g., GitFlow, Feature Branching) for managing code updates.

- Commit History & Documentation – Meaningful commit messages and detailed pull request descriptions.

- CI/CD Integration (if applicable) – Automated builds, testing, and deployment pipelines.

**3. Deployment & Execution**

- **README File – Includes:**
- **Installation steps**
- **System requirements (hardware/software dependencies)**
- **Configuration instructions**
- **Execution guide (running the project locally or accessing a deployed version)**
- **API documentation (if applicable)**
- **Executable Files & Deployment Link –**

- **Compiled software or packaged application (e.g., .exe, .jar, .apk).**
- **Deployed web/mobile app**

---

# 6. Testing & Quality Assurance

- **Test Cases & Test Plan** – Document detailing test scenarios and expected outcomes.
- **Automated Testing (if applicable)** – Any automated test scripts used.

- **Bug Reports** – Issues identified and resolutions.

---

# 7. Final Presentation & Reports

- **User Manual** – Instructions for end users.
- **Technical Documentation** – System architecture, database schema, API documentation.
- **Project Presentation (PPT/PDF)** – Summary of the project, challenges, solutions, and outcomes.
- **Video Demonstration (Optional)** – Short demo showcasing the project's functionality.