# E-Commerce Platform with ASP.NET Core

## Project Overview:

The **E-Commerce Website Project** is a dynamic online shopping platform designed to provide a seamless The **E-Commerce Website Project** is a full-featured online platform built using **ASP.NET Core** that enables users to browse and purchase products seamlessly. The system provides a **customer-friendly shopping experience** with features such as product search, shopping cart, secure checkout, and order tracking. Additionally, an **admin panel** allows efficient management of products, categories, and orders.

To ensure scalability and maintainability, the project follows the **NTier Architecture**, incorporating **Repository and Unit of Work patterns** for better data handling and business logic separation. With a **secure authentication system (ASP.NET Identity)** and **integrated payment gateway (Stripe/PayPal)**, this platform aims to deliver a robust and scalable e-commerce experience.

# Project Guidelines:

# 1. Project Planning & Management

### a) Project Proposal

- **Overview**

  In today's digital economy, online shopping has become an essential part of daily life, driving the need for robust and user-friendly e-commerce platforms. This project aims to develop a **comprehensive e-commerce website** that enables customers to explore a wide range of products, make secure purchases, and track their orders effortlessly. Additionally, an intuitive **admin panel** will be implemented to streamline product management, order processing, and customer interactions. By leveraging modern web technologies, this platform will ensure high performance, security, and scalability, catering to both businesses and consumers.

- **Objectives**

  The key objectives of this project include:

  - Developing a user-friendly e-commerce platform with an intuitive interface.
  - Implementing **secure authentication** for customers and admins.
  - Providing an **efficient product catalog** with search and filter functionalities.
  - Enabling a smooth **cart and checkout** process with multiple payment options.
  - Incorporating **order tracking and shipping management**.
  - Allowing admins to **manage products, categories, and customer orders**.

- **Scope**
  - Customer registration and authentication.
  - Product browsing, searching, and filtering.

- Adding products to the cart and Wishlist.
- Secure checkout and payment processing.
- Order history and real-time tracking.

## b) Project Plan

### Week 1: Initial Setup & Database Design

- Set up **NTier Architecture** (Presentation, Business Logic, Data Access).
- Implement **ASP.NET Identity** for authentication (login, register).
- Design **SQL Server database schema** (Products, Orders, Users, etc.).
- Configure **Entity Framework Core (Code First Approach)**.
- Set up **GitHub repository** for version control.

### Week 2: Product & Category Management

- Develop **Product & Category APIs** in **ASP.NET Core Web API**.
- Implement **Repository & Unit of Work Patterns** for data management.
- Build **Admin Panel UI (ASP.NET Core MVC) for managing products**.
- Implement **search and filter functionality**.

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

- Develop **Cart APIs** and implement **session-based cart management**.
- Implement **role-based access control (RBAC) for Admin & Customers**.
- Design **UI components for the shopping cart and checkout flow**.

### Week 4: Order Processing & Payment Integration

- Develop **Order placement, tracking, and history APIs**.
- Implement **Stripe Payment Gateway** for secure transactions.
- Use **Unit of Work Pattern** to ensure transaction consistency.
- Integrate **order status updates and notifications**.

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

- Build **responsive customer-facing UI (Product Listings, Checkout, Profile)**.
- Implement **JavaScript & AJAX for interactive features**.
- Improve **user experience using Bootstrap & Toaster JS**.
- Test frontend API integrations with backend.

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

- Improve **Admin Panel UI** with **DataTables for product management**.
- Implement **order management features (approve, cancel orders, refunds)**.
- Work on **frontend validation and error handling**.

### Week 7: Security & Optimization

- Implement **data validation & input sanitization**.
- Optimize **SQL queries and database indexing**.
- Enhance **API security with JWT Authentication & Authorization**.
- Work on **performance tuning for API responses**.

### Week 8: Testing & Debugging

- Conduct **unit testing (xUnit, NUnit) for backend APIs**.
- Perform **API testing with Postman & integration testing)**.
- Fix **UI/UX bugs and ensure cross-browser compatibility**.

### Week 9: Deployment & Final Review

- Set up **Docker containers for API & Database**.
- Deploy on **Microsoft Azure** and ensure a live environment.
- Conduct **User Acceptance Testing (UAT) & final optimizations**.
- Complete **project documentation** (system architecture, setup guide).

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

### 1. Team Lead & Full Stack Developer (Alaa Hassan)

**Key Responsibilities:**

- Manages **project architecture and planning**.
- Develops core **.NET Core Web API** for authentication & user management).
- Implements **NTier architecture** and enforces best practices.
- Develop **.NET Core Web API** for authentication & user management.
- Leads **GitHub repository** and version control strategies.
- Oversee **deployment on Microsoft Azure**.

### 2. Backend Developer (Haytham Hossam) – Product & Category Management

**Key Responsibilities:**

- Develops **Product & Category APIs** using **.NET Core Web API**.
- Implements **Repository Pattern** for efficient database interactions.
- Works on **Entity Framework Core for SQL Server** integration.
- Assists with **Docker containerization for backend services**

### 3. Backend Developer (Mariam Mahmoud) – Order & Payment Processing

**Key Responsibilities:**

- Implements **order placement, tracking, and management APIs**.
- Integrates **Stripe payment gateway** for secure transactions.
- Ensures **Unit of Work Pattern** for data consistency.
- Assists in **unit testing using xUnit/NUnit**.

### 4. Frontend Developer (Youssef Abdelazeem) – ASP.NET MVC & Admin Panel

**Key Responsibilities:**

- Develops **Admin Dashboard using ASP.NET Core MVC**.
- Implements **role-based access control (Admin & Customer roles)**.
- Works on **data tables for product & order management**.
- Ensures **MVC pattern implementation & API integration**.

### 5. Frontend Developer (Sara Alaa) – UI & User Experience

**Key Responsibilities:**

- Designs **responsive UI with HTML5, JavaScript, CSS3, and Bootstrap**.
- Implements **interactive UI components** and search/filter functionality.
- Works on **customer-facing pages (homepage, product listing, checkout)**.
- Conducts **frontend testing and bug fixes**.

### 6. DevOps & Testing Engineer (Mai Ezz Eldin) – CI/CD & Quality Assurance

**Key Responsibilities:**

- Manages **GitHub repository, version control, and CI/CD pipelines**.
- Works on **Docker containerization for backend and database**.
- Conducts **unit testing (xUnit, NUnit) & API testing (Postman)**.
- Assists with **final deployment and performance optimization.**

## d) Risk Assessment & Mitigation Plan

### 1. Technical Risks

| Risk | Potential Impact | Mitigation Strategy |
|---|---|---|
| **Bugs & Errors in Code** | Can cause system crashes, incorrect transactions, or security vulnerabilities. | Implement **unit testing (xUnit, NUnit)**, **code reviews**, and continuous debugging. |
| **Integration Issues (Payment Gateway, APIs)** | Payment failures or data mismatches may occur. | Perform **extensive API testing (Postman, Swagger)** before deployment. Keep logs for debugging. |
| **Performance Bottlenecks** | Slow website response times, affecting user experience. | Optimize **database queries**, use **caching**, and monitor API response times. |
| **Data Loss or Corruption** | Risk of losing important transaction or user data. | Implement **regular database backups** and **transaction rollbacks** for critical operations. |

## 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. Project Management Risks

| Risk | Potential Impact | Mitigation Strategy |
|---|---|---|
| **Scope Creep (Uncontrolled feature expansion)** | Delays in project completion, exceeding deadlines. | Define **clear requirements** and follow **Agile Sprints** with regular reviews. |
| **Missed Deadlines** | Project delivery is delayed, affecting stakeholders. | Use **weekly sprint planning**, **progress tracking (JIRA, Trello, or Azure DevOps)**, and set realistic timelines. |
| **Poor Communication Among Team Members** | Misaligned development efforts and duplicated work. | Conduct **daily standup meetings**, use **Slack/Teams for discussions**, and maintain a **shared project documentation repository**. |
| **Lack of Team Expertise in Some Technologies** | Slower development due to learning curves. | Provide **training sessions**, use **pair programming**, and encourage knowledge sharing. |

## 4. 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**. |

| | | |
|---|---|---|
| **High Traffic Handling Issues** | Website crashes during peak usage. | Implement **load balancing, caching mechanisms (Redis), and scalable cloud infrastructure (Azure/AWS)**. |

### 5. Compliance & Legal Risks

| Risk | Potential Impact | Mitigation Strategy |
|---|---|---|
| **Failure to Comply with Data Protection Laws (GDPR, CCPA, etc.)** | Legal consequences and penalties. | Ensure **proper data encryption, privacy policies, and user consent handling**. |
| **Unclear Terms & Conditions** | Disputes with users regarding transactions. | Draft a **clear terms of service, return policy, and privacy policy**. |

## e) KPIs (Key Performance Indicator)

### 1. System Performance KPIs:

- **API Response Time:** Measures how quickly backend APIs respond to requests.
- **Page Load Time:** Evaluates the time taken for pages to fully load in the browser.
- **System Uptime:** Tracks the availability of the system to ensure continuous service.
- **Database Query Performance:** Assesses the efficiency of database operations.

### 2. Security KPIs:

- **Failed Login Attempts:** Monitors repeated unsuccessful login attempts to detect potential threats.
- **SQL Injection & XSS Attack Prevention:** Ensures protection against common web 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.

### 4. Business Success KPIs:

- **Total Orders Processed:** Monitors the number of successful transactions completed on the platform.
- **Repeat Customer Rate:** Tracks the percentage of customers who return for additional purchases.
- **Payment Success Rate:** Measures the number of successful payments compared to failed transactions.

### 5. Development & Maintenance KPIs:

- **Code Quality & Bug Rate:** Assesses the number of critical bugs found in each release.
- **Testing Coverage:** Measures the extent of unit and integration test implementation.
- **Deployment Time:** Tracks the efficiency of feature updates and system fixes.

# 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

### 1. Stakeholder

**Analysis *Key***

***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.
3. **System**: Handles notifications, validations, and backend processes.

**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:*

# Login

| | |
|---|---|
| **Identifier and name:** | US1 Login |
| **Initiator:** | User |
| **Goal:** | The user wants to access their account and use the system's features. |
| **Description**: | The user wants to access their account on the application that requires authentication. To do so, they must provide their login credentials. The system will then verify the credentials and allow the user to access their account. If the user forgets their password, they can use the Forgot Password use case to recover it. If the user is new and doesn't have an account, they can use the Register use case to create a new account. If the user enters incorrect login credentials, the system will show a wrong password message and prompt the user to retry. |
| **Preconditions**: | • The user has a valid account on the application, or they have completed the registration process.<br>• The user knows their login credentials or has completed the forgot password process. |
| **Postconditions:** | • If the login is successful, the user can access their account and perform actions that require authentication.<br>• If the login fails, the user can't access their account and must retry the login process, use the Forgot Password use case to recover their password, or use the Register use case to create a new account. |
| **Basic flow (**Main success scenario**):** | 1. The user navigates to the login page.<br>2. The system presents a form for the user to enter their login credentials.<br>3. The user enters their username and password.<br>4. The system verifies the credentials using the Verify Login use case.<br>5. If the credentials are valid, the system logs the user in and redirects them to their account homepage.<br>6. If the credentials are invalid, the system shows a wrong password message and prompts the user to retry. |

| | |
|---|---|
| **Alternate flows:** | - If the user enters an incorrect username, the system may prompt them to try again or provide additional assistance, such as suggesting they create a new account.<br>- If the system detects suspicious activity, such as repeated failed login attempts or login attempts from an unfamiliar device or location, it may prompt the user to perform additional security checks, such as entering a verification code or answering security questions. |
| | **Included Verify Login use case:**<br>1. The system receives the user's login credentials from the Login use case.<br>2. The system verifies the credentials by comparing them to the user's account information in the database.<br>3. If the credentials are valid, the system returns a success message to the Login use case.<br>4. If the credentials are invalid, the system returns an error message to the Login use case. |
| **Extended flows:** | **Extends Register use case:**<br>1. If the user doesn't have an account, they can click the "Register" link.<br>2. The system redirects the user to the registration page.<br>3. The user fills out the registration form and submits it.<br>4. The system creates a new account and redirects the user to the login page.<br>**Extends Forgot Password use case:**<br>1. If the user forgets their password, they can click the "Forgot Password" link.<br>2. The system prompts the user to enter their email address or username.<br>3. The user enters their email address or username.<br>4. The system sends a password reset link to the user's email address.<br>5. The user clicks the password reset link and follows the instructions to reset their password.<br>6. The system prompts the user to enter a new password.<br>7. The user enters a new password.<br>8. The system updates the user's password and redirects them to the login page. |

# Logout

| | |
|---|---|
| **Identifier and name:** | US2 Logout |
| **Initiator:** | User |
| **Goal:** | The user wants to safely end their session and prevent unauthorized access to their account. |
| **Description**: | The user wants to end their current session on the application. To do so, they must log out of their account. The system will then terminate the session and return the user to the login page. |
| **Preconditions**: | - The user is currently logged in to their account. |
| **Postconditions:** | - The user is no longer logged in to their account and cannot access actions that require authentication until they log in again. |
| **Basic flow (**Main success scenario**):** | 1. The user clicks the "Logout" button.<br>2. The system terminates the user's session and logs them out of their account.<br>3. The system returns the user to the login page or displays a message confirming that they have been logged out. |
| **Alternate flows:** | - If the user is inactive for a certain period, the system may automatically log them out to prevent unauthorized access. |

# Update Account

| | |
|---|---|
| **Identifier and name:** | US6 Update Account |
| **Initiator:** | User |
| **Goal:** | The user wants to modify their account information to keep it up-to-date and accurate. |
| **Description**: | The user wants to update their account information on an application. This may include changing their name, email address, profile picture, or other details. The system will then update the user's account information with the new data. |
| **Preconditions**: | - The user is currently logged in to their account.<br>- The user has permission to update their account information. |
| **Postconditions:** | -The user's account information is updated with the new data.<br>-The user can log in to the system using their updated account information. |
| **Basic flow (**Main success scenario**):** | 1. The user navigates to the "Account Settings" page.<br>2. The system presents a form for the user to update their account information.<br>3. The user makes the desired changes to their account information.<br>4. The system verifies that the changes are valid.<br>5. If the verification is successful, the system updates the user's account information and displays a message confirming that the changes have been saved.<br>6. If the verification is unsuccessful, the system displays an error message and prompts the user to retry. |
| **Alternate flows:** | - If the user wants to delete their account, they may need to use the Delete Account use case instead. |

# View Account Details

| | |
|---|---|
| **Identifier and name:** | US5 View Account Details |
| **Initiator:** | User |
| **Goal:** | The user wants to view their account information and details. |
| **Description**: | The user wants to view their account details on the application. This may include details such as their name, email address, profile picture, or other information. The system will then display the user's account details to the user and any other users who have permission to view them. |
| **Preconditions**: | - The user is currently logged in to their account.<br>- The user has permission to view their own account details.<br>- Other users have permission to view the user's account details. |
| **Postconditions:** | - The user can view and/or update their account details as needed.<br>- Other users who have permission to view the user's account details can see the information as well.<br>- Any changes made to the user's account details are reflected in the system and any features or actions that rely on the updated information. |
| **Basic flow (**Main success scenario**):** | 1. The user navigates to the "Account Details" page.<br>2. The system displays the user's account details, including their name, email address, profile picture, or other information.<br>3. The user reviews their account details and/or updates them using the Update Account use case. |

# Change Password

| | |
|---|---|
| **Identifier and name:** | US3 Change Password |
| **Initiator:** | User |
| **Goal:** | The user wants to update their password to ensure the security of their account. |
| **Description**: | The user wants to change their password for their account on the application. To do so, they must provide their current password and choose a new password. The system will then update their account information with the new password. |
| **Preconditions**: | - The user is currently logged in to their account.<br>- The user knows their current password. |
| **Postconditions:** | - The user's account information is updated with the new password.<br>- The user can log in to their account using the new password.<br>- The user's old password is no longer valid and cannot be used to access their account. |
| **Basic flow (**Main success scenario**):** | 1. The user navigates to the "Change Password" page.<br>2. The system presents a form for the user to enter their current password and choose a new password.<br>3. The user enters their current password and chooses a new password.<br>4. The system verifies that the current password is correct and that the new password meets the system's password requirements (such as length, complexity, etc.).<br>5. If the verification is successful, the system updates the user's account information with the new password and displays a message confirming that the password has been changed.<br>6. If the verification is unsuccessful, the system displays an error message and prompts the user to retry. |
| **Alternate flows:** | - If the user forgets their current password, they may need to use the Forgot Password use case to recover it before they can change their password. |

# Manage Categories

| | |
|---|---|
| **Identifier and name:** | US2 Manage Categories |
| **Initiator:** | Admin |
| **Goal:** | The admin wants to manage product categories. |
| **Description**: | The admin can view, add, edit, and delete categories. The system validates the category information. |
| **Preconditions**: | - The admin has access to the category management functionality. |
| **Postconditions:** | - The categories are up-to-date and accurate. |
| **Basic flow (**Main success scenario**):** | 1. The admin navigates to the "Category Management" page.<br>2. The admin can add new categories by entering their information manually or importing a list of categories from a file.<br>3. The admin can edit existing category records by updating their information, such as name, description, or image.<br>4. The admin can delete category records from the hierarchy.<br>7. The system validates the category information.<br>8. The system displays a message confirming that the changes have been saved. |

# Manage Customers Database

| | |
|---|---|
| **Identifier and name:** | US3 Manage Customers Database |
| **Initiator:** | Admin |
| **Goal:** | The admin wants to manage the customers database, including adding, editing, and deleting customer records |

| | |
|---|---|
| **Description**: | The admin wants to manage the customers database on the application. This may be necessary to add, edit, or delete customer information, or to perform other administrative tasks related to customer data. |
| **Preconditions**: | - The admin has the necessary permissions and access to manage the customers database.<br>- The application has a customer's database available for management. |
| **Postconditions:** | - The customers database is updated based on the admin's actions.<br>- Any actions or features that rely on the updated customer information will reflect the updated database.<br>- The admin can continue to manage the customers database as needed. |
| **Basic flow (**Main success scenario**):** | 1. The admin navigates to the "Customers Database" page.<br>2. The system presents a list or menu of available tasks for managing the customers database, such as adding a new customer, editing an existing customer, or deleting a customer.<br>3. The admin selects the desired task from the list or menu.<br>4. The system presents a form for entering or editing the customer information or prompts the admin to confirm the deletion.<br>5. The admin enters or edits the required information or confirms the deletion.<br>6. The system updates the customers database based on the admin's actions and displays a message confirming the successful completion of the task. |

# Manage Products

| | |
|---|---|
| **Identifier and name:** | US1 Manage Products |
| **Initiator:** | Admin |
| **Goal:** | The admin wants to manage products in the product catalog. |

| | |
|---|---|
| **Description**: | The admin can view, search, add, edit, and delete product records in the product catalog. The system validates the product information and updates the catalog accordingly. |
| **Preconditions**: | - The admin has access to the product management functionality.<br>- The product catalog is accessible. |
| **Postconditions:** | - The product catalog is up-to-date and accurate. |
| **Basic flow (**Main success scenario**):** | 1. The admin navigates to the "Product Management" page.<br>2. The admin can view a list of all products in the catalog, including their basic information such as name, description, price, and stock level.<br>3. The admin can search for specific products using various search criteria, such as name or category.<br>4. The admin can add new products to the catalog by entering their information manually or importing a list of products from a file.<br>5. The admin can edit existing product records by updating their information, such as name, description, price, or stock level.<br>6. The admin can delete product records from the catalog.<br>7. The admin can manage product categories, including creating, editing, and deleting categories.<br>8. The user can manage product images, including uploading, editing, and deleting images for products.<br>9. The system validates the product information and updates the catalog accordingly.<br>10. The system displays a message confirming that the changes have been saved. |

# Block & Unblock Customers

| | |
|---|---|
| **Identifier and name:** | US4 Block & Unblock Customers |
| **Initiator:** | Admin |
| **Goal:** | The admin wants to block or unblock a customer from accessing the system. |
| **Description**: | The admin wants to block or unblock customers on the application. This may be necessary if a customer violates the application's policies or terms of service, or if the admin needs to prevent a customer from accessing certain features or content. |
| **Preconditions**: | - The admin has the necessary permissions and access to block or unblock customers.<br>- The application has customers that can be blocked or unblocked. |
| **Postconditions:** | - The selected customer's account is updated based on the admin's actions.<br>- Any actions or features that rely on the blocked or unblocked customer status will reflect the updated account status.<br>- The admin can continue to manage the customers on the application as needed. |
| **Basic flow (**Main success scenario**):** | 1. The admin navigates to the "Customers" page.<br>2. The system presents a list or menu of available customers.<br>3. The admin selects the desired customer from the list or menu.<br>4. The system presents options for blocking or unblocking the selected customer.<br>5. If the admin chooses to block the customer, the system updates the customer's account to prevent access to certain features or content.<br>6. If the admin chooses to unblock the customer, the system updates the customer's account to restore access to previously blocked features or content.<br>7. The system displays a message confirming the successful completion of the blocking or unblocking action. |

# Cancel Order

| | |
|---|---|
| **Identifier and name:** | US16 Cancel Order |
| **Initiator:** | Customer |
| **Goal:** | The Customer wants to cancel an order in the application. |
| **Description**: | The Customer wants to cancel their order on the application. This may be necessary if the Customer changes their mind, if the product is no longer needed, or if there is a problem with the order. The customer service representative will guide the user through the process of cancelling the order. |
| **Preconditions**: | - The Customer has placed an order on the application.<br>- The Customer has identified a reason for cancelling the order.<br>- The Customer has contacted customer service to request a cancellation. |
| **Postconditions:** | - The customer's order is cancelled and any associated products are returned or refunded (if applicable).<br>- Any actions or features that rely on the cancelled order will reflect the cancellation.<br>- The Customer can continue to use the application to place new orders as desired. |
| **Basic flow (**Main success scenario**):** | 1. The customer service representative verifies the user's account and order information.<br>2. The customer service representative confirms the user's reason for cancelling the order.<br>3. The customer service representative provides the user with instructions on how to return any received products (if applicable). 4. The user follows the instructions to return any received products (if applicable).<br>5. The customer service representative processes the order cancellation and confirms the estimated refund date (if applicable).<br>6. The customer receives a confirmation of the order cancellation and any associated refund information (if applicable). |

| | |
|---|---|
| **Alternate flows:** | - If the customer has not received any products yet, the customer service representative may simply cancel the order and provide a confirmation.<br>- If the user needs assistance with returning products or obtaining a refund, the customer service representative may provide additional support or guidance. |

# Choose Category

| | |
|---|---|
| **Identifier and name:** | US6 Choose Category |
| **Initiator:** | Customer |
| **Goal:** | The customer wants to select a category for a product. |
| **Description**: | The user wants to select a category on the application. This may be necessary for various reasons, such as browsing products, searching for information, or filtering content. The system will then display the relevant information or content based on the selected category. |
| **Preconditions**: | - The customer is currently on a page or interface that requires a category selection.<br>- The customer has permission to view or access the selected category. |
| **Postconditions:** | - The customer can view the relevant information or content based on the selected category.<br>- Any actions or features that rely on the selected category will reflect the chosen category. |
| **Basic flow (**Main success scenario**):** | 1. The customer navigates to the page or interface where the category selection is required.<br>2. The system presents a list or menu of available categories.<br>3. The customer selects the desired category from the list or menu.<br>4. The system displays the relevant information or content based on the selected category. |

# Display Bill

| | |
|---|---|
| **Identifier and name:** | US12 Display Bill |
| **Initiator:** | Customer |
| **Goal:** | The customer wants to view and manage bills in the application. |
| **Description**: | The customer wants to view their bill on the application. This may include details such as the amount owed, due date, payment history, or other information. The system will then display the user's bill and any relevant information associated with it. |
| **Preconditions**: | - The Customer has an account on the application.<br>- The Customer has a bill associated with their account.<br>- The Customer has permission to view their bill. |
| **Postconditions:** | - The Customer can view their bill and any relevant information associated with it.<br>- Any actions or features that rely on the bill information will reflect the details displayed.<br>- The Customer can continue to view or modify their bill information as needed. |
| **Basic flow (**Main success scenario**):** | 1. The Customer navigates to the "Billing" page or interface.<br>2. The system presents a list or menu of available bills associated with the customer's account.<br>3. The customer selects the desired bill from the list or menu.<br>4. The system displays the bill and any relevant information associated with it, such as the amount owed, due date, payment history, or other details. |
| **Alternate flows:** | - If the customer has multiple bills associated with their account, they may need to use additional search or filter features to find the desired bill.<br>- If the user has completed payment on a bill, the system may display a message confirming that the bill has been paid and any associated payment history. |

# Make Payment

| | |
|---|---|
| **Identifier and name:** | US20 Make Payment |
| **Initiator:** | Customer |
| **Goal:** | The Customer wants to make a payment for an order or bill in the application. |
| **Description**: | The Customer wants to make a payment on the application. This may be necessary if the Customer owes a balance, needs to pay a bill, or wants to purchase products or services. The system will then guide the Customer through the process of making a payment. |
| **Preconditions**: | - The Customer has a valid payment method on file with the application.<br>- The Customer has identified the amount to be paid and the reason for the payment. |
| **Postconditions:** | - The customer's payment is processed and applied to the associated balance, bill, or order.<br>- Any actions or features that rely on the completed payment will reflect the updated payment status.<br>- The Customer can continue to use the application with the payment successfully processed. |
| **Basic flow (**Main success scenario**):** | 1. The Customer navigates to the "Make Payment" page or interface.<br>2. The system presents a form or interface for entering payment details, including the payment amount and any relevant information such as a billing address or payment method.<br>3. The Customer enters the required payment details and confirms the payment.<br>4. The system processes the payment and displays a message confirming that the payment is complete. |

| | |
|---|---|
| **Alternate flows:** | - If the Customer needs to modify their payment method or information, they may need to use additional forms or interfaces to update their payment information before making the payment.<br>- If the payment is for a specific product or service, the system may need to verify that the payment matches the associated order or invoice before processing the payment. |

# Place Order

| | |
|---|---|
| **Identifier and name:** | US10 Place Order |
| **Initiator:** | Customer |
| **Goal:** | The customer wants to place an order for the selected products and complete the purchase process. |
| **Description**: | The customer wants to place an order and complete the purchase process on the application. The customer selects the products they want to purchase, enters the shipping address, billing information, and payment details, and confirms the order and payment details. The application processes the payment, updates the order status, and sends a confirmation email to the customer with the order details and confirmation number. The app also updates the inventory levels and shipping status of the ordered products. |
| **Preconditions**: | - The customer has selected the products they want to purchase.<br>- The customer has access to the shopping cart or checkout functionality.<br>- The customer is logged in to the app. |
| **Postconditions:** | - The customer has placed an order and completed the purchase process.<br>- The customer receives a confirmation email with the order details and confirmation number.<br>- The app updates the inventory levels and shipping status of the ordered products. |

| | |
|---|---|
| **Basic flow (**Main success scenario**):** | 1. The customer navigates to the shopping cart or checkout page.<br>2. The app displays the selected products, along with their quantities and prices.<br>3. The customer reviews the product details, quantities, and prices, and can update the cart contents as needed.<br>4. The customer enters the shipping address, billing information, and payment details.<br>5. The customer confirms the order and payment details.<br>6. The app processes the payment and updates the order status.<br>7. The app sends a confirmation email to the customer with the order details and confirmation number.<br>8. The app updates the inventory levels and shipping status of the ordered products. |
| **Alternate flows:** | Invalid Payment Information |
| | 1. The customer enters the shipping address, billing information, and payment details.<br>2. The app validates the payment information and detects an error or issue, such as an expired or invalid credit card, insufficient funds, or a declined transaction.<br>3. The app displays an error message to the customer, informing them of the issue and prompting them to correct the payment information.<br>4. The customer updates the payment information and confirms the order and payment details.<br>5. The app processes the payment and updates the order status.<br>6. The app sends a confirmation email to the user with the order details and confirmation number.<br>7. The platform updates the inventory levels and shipping status of the ordered products. |

# Proceed to Checkout

| | |
|---|---|
| **Identifier and name:** | US13 Proceed to Checkout |
| **Initiator:** | Customer |
| **Goal:** | The Customer wants to finalize the order and proceed to the checkout process in the application. |
| **Description**: | The Customer wants to proceed to check out on the application. This may be necessary if the user wants to purchase products or services. The system will then guide the Customer through the checkout process to complete the transaction. |
| **Preconditions**: | - The Customer has added products or services to their cart.<br>- The Customer is currently on the "Cart" page or interface.<br>- The Customer has permission to proceed to checkout. |
| **Postconditions:** | - The customer's transaction is complete and any purchased products or services are processed according to the system's policies.<br>- The Customer can view their order history or receipt for the transaction.<br>- Any actions or features that rely on the completed transaction will reflect the purchased products or services. |
| **Basic flow (**Main success scenario**):** | 1. The Customer navigates to the "Checkout" page or interface.<br>2. The system presents a summary of the customer's cart contents and total cost.<br>3. The Customer enters their shipping information, billing information, and payment details as required.<br>4. The system verifies that the information is valid and meets any applicable requirements or restrictions.<br>5. If the verification is successful, the system processes the payment and displays a message confirming that the transaction is complete. 6. If the verification is unsuccessful, the system displays an error message and prompts the user to retry. |

| | |
|---|---|
| **Alternate flows:** | - If the user wants to modify their cart contents, they may need to use the View Cart use case before proceeding to checkout.<br>- If the user needs to use a coupon or discount code, they may need to use a separate interface or feature to apply the discount before proceeding to checkout. |

# Replace product

| | |
|---|---|
| **Identifier and name:** | US15 Replace a Product |
| **Initiator:** | Customer |
| **Goal:** | The Customer wants to replace a product in the application. |
| **Description**: | The Customer wants to replace a product they have received from the application. This may be necessary if the product is damaged, defective, or not as described. The customer service representative will guide the user through the process of replacing the product. |
| **Preconditions**: | - The Customer has received the product from the application.<br>- The Customer has identified a problem with the product that requires a replacement.<br>- The Customer has contacted customer service to request a replacement. |
| **Postconditions:** | - The customer's replacement order is processed and confirmed by the customer service representative.<br>- The Customer can track the replacement order and delivery information. |

| | |
|---|---|
| **Basic flow (**Main success scenario**):** | 1. The customer service representative verifies the user's account and order information.<br>2. The customer service representative confirms the user's reason for requesting a replacement.<br>3. The customer service representative provides the user with instructions on how to return the product.<br>4. The user returns the product to the designated address or location as instructed.<br>5. The customer service representative verifies that the returned product matches the original order and is in the expected condition.<br>6. If the verification is successful, the customer service representative processes the replacement order and confirms the estimated delivery date.<br>7. If the verification is unsuccessful, the customer service representative may investigate further or provide alternative solutions. |
| **Alternate flows:** | - If the user needs assistance with returning the product, the customer service representative may provide additional support or arrange for a pickup.<br>- If the user requires a refund instead of a replacement, the customer service representative may guide the user through the refund process instead. |

# Search Product

| | |
|---|---|
| **Identifier and name:** | US8 Search Product |
| **Initiator:** | Customer |
| **Goal:** | The customer wants to search for a product in the application. |
| **Description**: | The Search Product use case involves users searching for specific products or items on the application. The customer enters a search query into a search bar or field, and the platform returns a list of relevant products that match the user's query. |
| **Preconditions**: | - User is looking for a specific product or item.<br>- The application has a search feature. |

| | |
|---|---|
| **Postconditions:** | - Customer has found the desired product or item<br>- The application has provided accurate search results to the user |
| **Basic flow (**Main success scenario**):** | 1. Customer navigates to the application search bar or field.<br>2. Customer enters a search query for the desired product or item<br>3. The application search algorithm processes the query and returns a list of relevant products<br>4. Customer reviews the list of products and selects the desired item<br>5. Customer proceeds to view the selected product's details and may choose to purchase it |
| **Alternate flows:** | - If the Customer 's search query returns no results, the application may suggest alternative search terms or provide other product recommendations<br>- If the Customer finds the search results to be inaccurate or irrelevant, they may refine their search query or adjust their search filters to narrow down the results |

# See Today's Offers

| | |
|---|---|
| **Identifier and name:** | US18 View Today's Offers |
| **Initiator:** | Customer |
| **Goal:** | The Customer wants to view the current offers and promotions in the application. |
| **Description**: | The Customer wants to view today's offers on the application. This may include discounts, promotions, or other special deals that are only available for a limited time. The system will then display the available offers and any relevant information associated with them. |
| **Preconditions**: | - The Customer is currently using the application.<br>- The application has today's offers available. |

| | |
|---|---|
| **Postconditions:** | - The Customer can view today's offers and any relevant information associated with them.<br>- Any actions or features that rely on the offer details will reflect the displayed information.<br>- The Customer can continue to view or take advantage of the available offers until they expire or are no longer available. |
| **Basic flow (**Main success scenario**):** | 1. The Customer navigates to the "Today's Offers" page or interface.<br>2. The system presents a list or menu of available offers for the day.<br>3. The Customer selects the desired offer from the list or menu.<br>4. The system displays the offer and any relevant information associated with it, such as the discount amount, promotion details, or other relevant information. |
| **Alternate flows:** | - If the user does not see the desired offer in the list or menu, they may need to use additional search or filter features to find it.<br>- If the offer is time-sensitive, the system may display a countdown or other indicator of the remaining time to take advantage of the offer. |

# Submit Feedback

| | |
|---|---|
| **Identifier and name:** | US11 Submit Feedback |
| **Initiator:** | Customer |
| **Goal:** | The customer wants to provide feedback or suggestions to the application. |
| **Description**: | The customer wants to provide feedback on the application. This may include suggestions for improvement, bug reports, or other comments. The system will then use this feedback to improve the customer experience or address any issues. |
| **Preconditions**: | - The customer is currently using the application.<br>- The customer has permission to submit feedback. |

| | |
|---|---|
| **Postconditions:** | - The customer's feedback is saved and can be used by the system to improve the customer experience or address any issues.<br>- Any actions or features that rely on the user feedback will reflect the submitted feedback. |
| **Basic flow (**Main success scenario**):** | 1. The customer navigates to the "Feedback" page or interface.<br>2. The system presents a form or interface for the customer to enter their feedback.<br>3. The customer enters their feedback in the provided text field or interface.<br>4. The system verifies that the feedback is valid and meets any applicable requirements or restrictions.<br>5. If the verification is successful, the system saves the feedback and displays a message confirming that the feedback has been submitted. 6. If the verification is unsuccessful, the system displays an error message and prompts the user to retry. |
| **Alternate flows:** | - If the user wants to provide feedback on a specific feature or aspect of the application, they may need to use a more specific feedback form or interface. |

# View Cart

| | |
|---|---|
| **Identifier and name:** | US9 View Cart |
| **Initiator:** | Customer |
| **Goal:** | The Customer wants to view the items they have added to their shopping cart and manage their cart contents. |
| **Description**: | The View Cart use case involves users viewing the items they have added to their shopping cart on application. The Customer navigates to the cart and can view the items they have added to their cart, along with their quantities and prices. The Customer may also have the option to edit or remove items from their cart. |
| **Preconditions**: | - Customer has added items to their shopping cart.<br>- The application has a cart or checkout page |

| | |
|---|---|
| **Postconditions:** | - Customer has reviewed the items in their shopping cart<br>- Customer has the option to edit or remove items from their cart<br>- Customer may proceed to the checkout page to complete their purchase |
| **Basic flow (**Main success scenario**):** | 1. Customer navigates to the application's cart or checkout page<br>2. The application displays the items the Customer has added to their cart, along with their quantities and prices.<br>3. Customer reviews the items in their cart and may choose to edit or remove items as needed<br>4. Customer proceeds to the checkout page to complete their purchase |
| **Alternate flows:** | - If the customer has not added any items to their cart, the platform may display a message indicating that the cart is empty<br>- If the customer encounters any issues or errors while viewing their cart, the platform may provide error messages or support options to help the user resolve the issue |

# Add product to cart

| | |
|---|---|
| **Identifier and name:** | US14 Add product to cart |
| **Initiator:** | Customer |
| **Goal:** | The Customer wants to add products to their cart for future purchase on the application. |
| **Description**: | The Customer wants to add products to their cart. The user selects the desired product and adds it to their cart by entering the quantity and clicking the "Add to Cart" button. The platform adds the selected product to the customer's cart and displays a confirmation message. The customer can then view the cart contents and proceed to checkout when ready to purchase. |
| **Preconditions**: | - The Customer is logged in to the app.<br>- The Customer has accessed the product page on the app. |

| | |
|---|---|
| **Postconditions:** | - The Customer has added the product to their cart for future purchase.<br>- The app updates the cart contents and displays the updated cart summary. |
| **Basic flow (**Main success scenario**):** | 1. The Customer selects the product they want to add to their cart.<br>2. The app displays the product details, including the price and quantity available.<br>3. The Customer enters the desired quantity of the product.<br>4. The Customer clicks the "Add to Cart" button.<br>5. The app adds the selected product to the user's cart and displays a confirmation message. |
| **Alternate flows:** | - If the Customer wants to remove a product from their cart, they can select the "Remove from Cart" button from View Cart use case. |

# Add Shipping Info

| | |
|---|---|
| **Identifier and name:** | US7 Add Shipping Info |
| **Initiator:** | Customer |
| **Goal:** | The Customer wants to add shipping information for an order. |
| **Description**: | The Customer wants to add shipping information to their order on the application. This may include details such as their name, address, phone number, or other information. The system will then use this information to deliver the order to the Customer 's desired location. |
| **Preconditions**: | - The Customer has placed an order on the application.<br>- The Customer is required to provide shipping information to complete the order.<br>- The Customer has permission to add shipping information to their order. |

| | |
|---|---|
| **Postconditions:** | - The Customer 's order is updated with the new shipping information.<br>- The system will use the shipping information to deliver the order to the Customer 's desired location.<br>- Any actions or features that rely on the shipping information will reflect the new information. |
| **Basic flow (**Main success scenario**):** | 1. The Customer navigates to the "Shipping Information" page or interface.<br>2. The system presents a form or interface for the user to enter their shipping information.<br>3. The Customer enters their name, address, phone number, or other required information.<br>4. The system verifies that the information is valid and meets any applicable requirements or restrictions.<br>5. If the verification is successful, the system saves the shipping information to the customer's order and displays a message confirming that the information has been added.<br>6. If the verification is unsuccessful, the system displays an error message and prompts the user to retry. |
| **Alternate flows:** | - If the user has already provided shipping information for their order, they may need to use the Update Shipping Info use case to make changes or updates.<br>- If the user wants to cancel their order, they may need to use the Cancel Order use case instead. |

# Call Customer Service

| | |
|---|---|
| **Identifier and name:** | US17 Call Customer Service |
| **Initiator:** | Customer |
| **Goal:** | The Customer wants to contact customer service for assistance with an issue related to the application. |

| | |
|---|---|
| **Description**: | The customer wants to call customer service on the application. This may be necessary if the Customer needs assistance with an issue, has a question, or needs to report a problem. The customer service representative will guide the user through the process of resolving their issue or addressing their concern. |
| **Preconditions**: | - The Customer has a phone or device capable of making a phone call.<br>- The Customer has identified a reason for calling customer service. |
| **Postconditions:** | - The customer's issue is resolved or concern is addressed to their satisfaction.<br>- Any actions or features that rely on the resolved issue or addressed concern will reflect the resolution.<br>- The Customer can continue to use the application with the issue resolved or concern addressed. |
| **Basic flow (**Main success scenario**):** | 1. The Customer dials the customer service phone number provided on the application.<br>2. The call is answered by a customer service representative.<br>3. The customer service representative verifies the user's account and reason for calling.<br>4. The customer service representative helps or guidance to the user to resolve their issue or address their concern.<br>5. If the issue is resolved or the concern is addressed, the customer service representative concludes the call.<br>6. If the issue or concern cannot be resolved on the call, the customer service representative may provide alternative solutions or escalate the issue to a higher level of support. |
| **Alternate flows:** | - If the user is unable to reach a customer service representative on the call, they may need to try again later or use an alternate contact method (such as email or chat).<br>- If the user needs assistance in a different language, the customer service representative may transfer the call to a representative who speaks the desired language. |

# Send Offers Notification

| | |
|---|---|
| **Identifier and name:** | US11 Send Offers Notification |
| **Initiator:** | Customers, system |
| **Goal:** | The system wants to send promotional offers notifications to customers. |
| **Description**: | The application wants to send an offers notification to a user. This may be necessary to provide information on new products, services, or promotions that may be of interest to the user. The system will then generate and send the offers notification to the user. |
| **Preconditions**: | - The application has offers or promotions available to send.<br>- The user has provided contact information or other information necessary to receive the offers notification. |
| **Postconditions:** | - The user receives and takes advantage of the offers or promotions, resulting in the applicable discount or promotion.<br>- Any actions or features that rely on the offers or promotions will reflect the updated information.<br>- The user can continue to receive and take advantage of offers or promotions as available. |
| **Basic flow (**Main success scenario**):** | 1. The system generates an offers notification based on the available offers or promotions.<br>2. The system sends the offers notification to the user via the user's preferred notification method, such as email or SMS.<br>3. The user receives the offers notification and can view the available offers or promotions.<br>4. The user can then choose to take advantage of the offers or promotions as desired. |
| **Alternate flows:** | - If the user doesn't receive the offers notification or has trouble taking advantage of the offers or promotions, the system may send additional notifications or provide support to assist with the issue.<br>- If the offers or promotions are time-sensitive, the system may display a countdown or other indicator of the remaining time to take advantage of the offers or promotions. |

# Send Order Notification

| | |
|---|---|
| **Identifier and name:** | US10 Send Order Notification |
| **Initiator:** | Customers, system |
| **Goal:** | The system wants to send order notifications to customers. |
| **Description**: | The application wants to send an order notification to a user. This may be necessary to provide information about the order. The system will then generate and send the order notification to the user. |
| **Preconditions**: | - The user has placed an order on the application.<br>- The application has order notifications available to send. |
| **Postconditions:** | - The user receives the order notification and can view the updated order status.<br>- Any actions or features that rely on the updated order status will reflect the updated information.<br>- The user can continue to use the application and receive order notifications as necessary. |
| **Basic flow (**Main success scenario**):** | 1. The system receives an update on the status of the order.<br>2. The system generates an order notification based on the status update.<br>3. The system sends the order notification to the user via the user's preferred notification method, such as email or SMS.<br>4. The user receives the order notification and can view the updated order status. |
| **Alternate flows:** | - If the user has not opted in to receive order notifications, the system may prompt the user to opt in or may send the notification via a different method (such as displaying the update on the application).<br>- If the order encounters any issues or delays, the system may send additional notifications to the user to provide updates or information on the resolution of the issue. |

# Send Shipment Update Notification

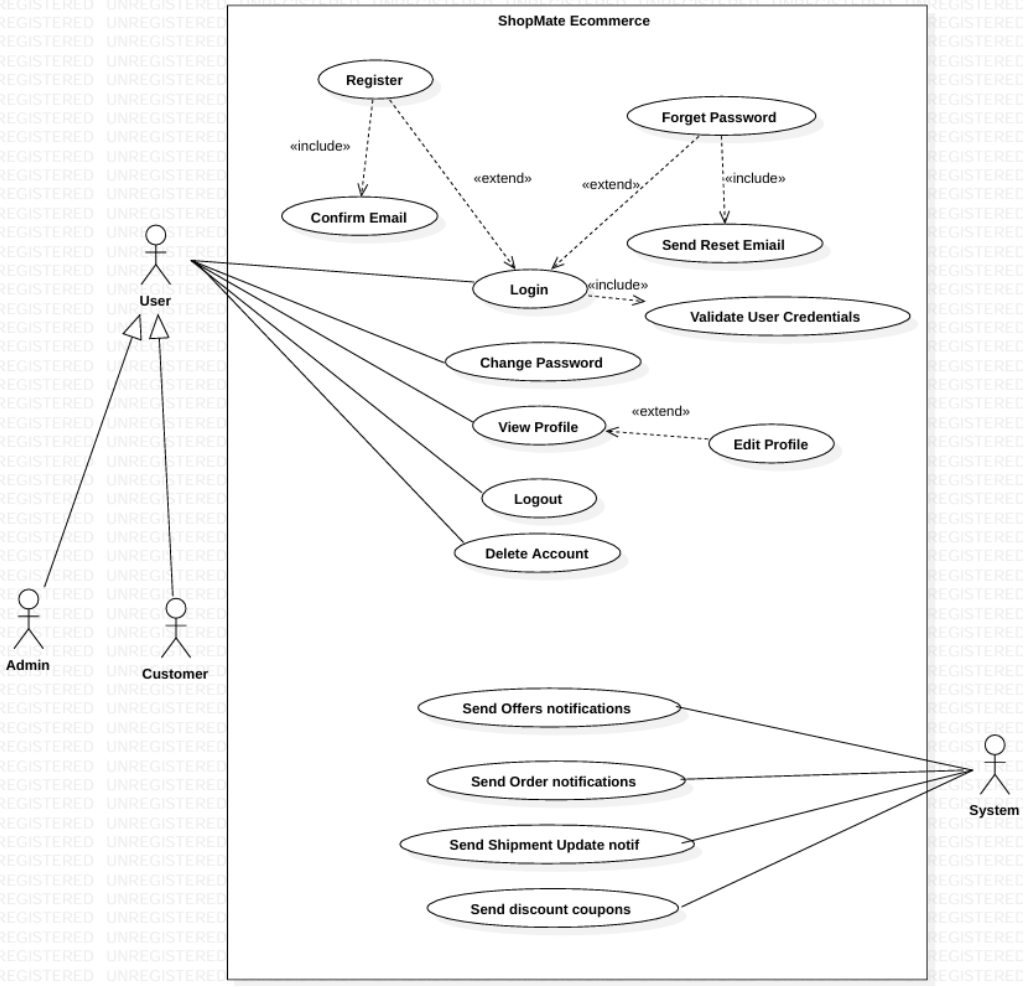
| | |
|---|---|
| **Identifier and name:** | US8 Send Shipment Update Notification |
| **Initiator:** | Customers, System |
| **Goal:** | The system wants to send a shipment update notification to a customer. |
| **Description**: | The application wants to send a shipment update notification to a user. This may be necessary to provide information on the status of a shipment, including tracking information or delivery updates. The system will then generate and send the shipment update notification to the user. |
| **Preconditions**: | - The user has placed an order on the application that includes a shipment.<br>- The application has shipment update notifications available to send. |
| **Postconditions:** | - The user receives the shipment update notification and can view the updated shipment status.<br>- Any actions or features that rely on the updated shipment status will reflect the updated information.<br>- The user can continue to use the application and receive shipment update notifications as necessary. |
| **Basic flow (**Main success scenario**):** | 1. The system receives an update on the status of the shipment.<br>2. The system generates a shipment update notification based on the status update.<br>3. The system sends the shipment update notification to the user via the user's preferred notification method, such as email or SMS.<br>4. The user receives the shipment update notification and can view the updated shipment status. |
| **Alternate flows:** | - If the user has not opted in to receive shipment update notifications, the system may prompt the user to opt in or may send the notification via a different method (such as displaying the update on the application).<br>- If the shipment encounters any issues or delays, the system may send additional notifications to the user to provide updates or information on the resolution of the issue. |

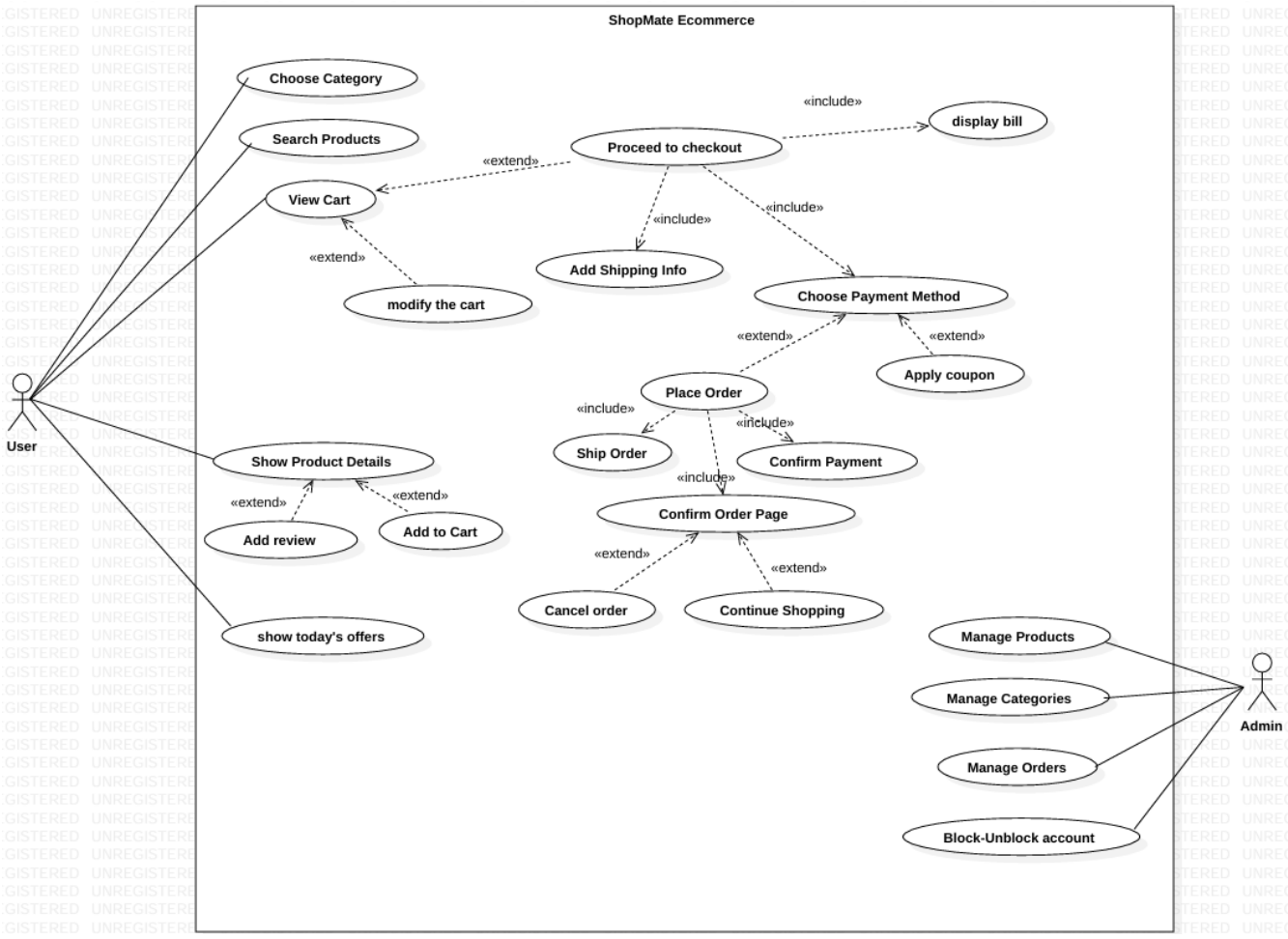
# Delete Account

| | |
|---|---|
| **Identifier and name:** | US7 Delete Account |
| **Initiator:** | User |
| **Goal:** | The user wants to permanently remove their account and all associated data from the system |
| **Description**: | The user wants to delete their account information on the application. The system will then delete the user's account information. |
| **Preconditions**: | - The user is logged in to the system.<br>- The user has the necessary permissions to delete their account. |
| **Postconditions:** | - The user's account and all associated data are permanently deleted from the system.<br>- The user cannot log in to the system using their deleted account information. |
| **Basic flow (**Main success scenario**):** | 1. The user navigates to the "Account Settings" page.<br>2. The user clicks on the "Delete Account" button.<br>3. The system displays a confirmation message warning the user that the action is irreversible.<br>4. The user confirms the deletion by entering their password and clicking on the "Delete Account" button.<br>5. The system permanently removes the user's account and all associated data from the system.<br>6. The system displays a message confirming that the account has been deleted. |

# 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**

2.1 Performance
- Response Time
- Load Handling

2.2 Security
- Data Encryption
- Authentication
- Authorization

2.3 Usability
- User Interface
- Accessibility

2.4 Scalability
- Scalability

2.5 Reliability
- Uptime
- Error Handling

2.6 Compatibility
- Cross-Browser Compatibility
- Mobile Compatibility

## Software Architecture:

**High-Level Design:**
This part outlines the major components of the system, the roles they play, and how they work together. It gives you an overview of the entire system's structure.
**System Components:**
These are the major pieces of the software. They could be:

- **Frontend:** The part of the system that users interact with. It could be a website or mobile app where customers browse products, place orders, and track shipments.
- **Backend:** This handles the logic behind the system. It processes requests from the frontend, manages data, and handles business rules. It might include services like authentication, payment processing, and inventory management.
- **Database:** The system's data storage. It keeps records like product information, customer details, orders, and transaction histories.
- **External Services:** These might be external APIs or services that the system integrates with, such as payment gateways (PayPal, Stripe) or email/SMS notification services.

**Interactions:**
This describes how the components communicate with each other. For example:

- **Frontend to Backend:** When a user submits a form (e.g., login or order), the frontend sends a request to the backend to process the action. The backend responds with data or an action, such as confirming the order or sending an error message.
- **Backend to Database:** The backend queries the database for product details or updates the database when a customer places an order.
- **Backend to External Services:** The system may interact with external services for tasks like sending email notifications or processing payments.

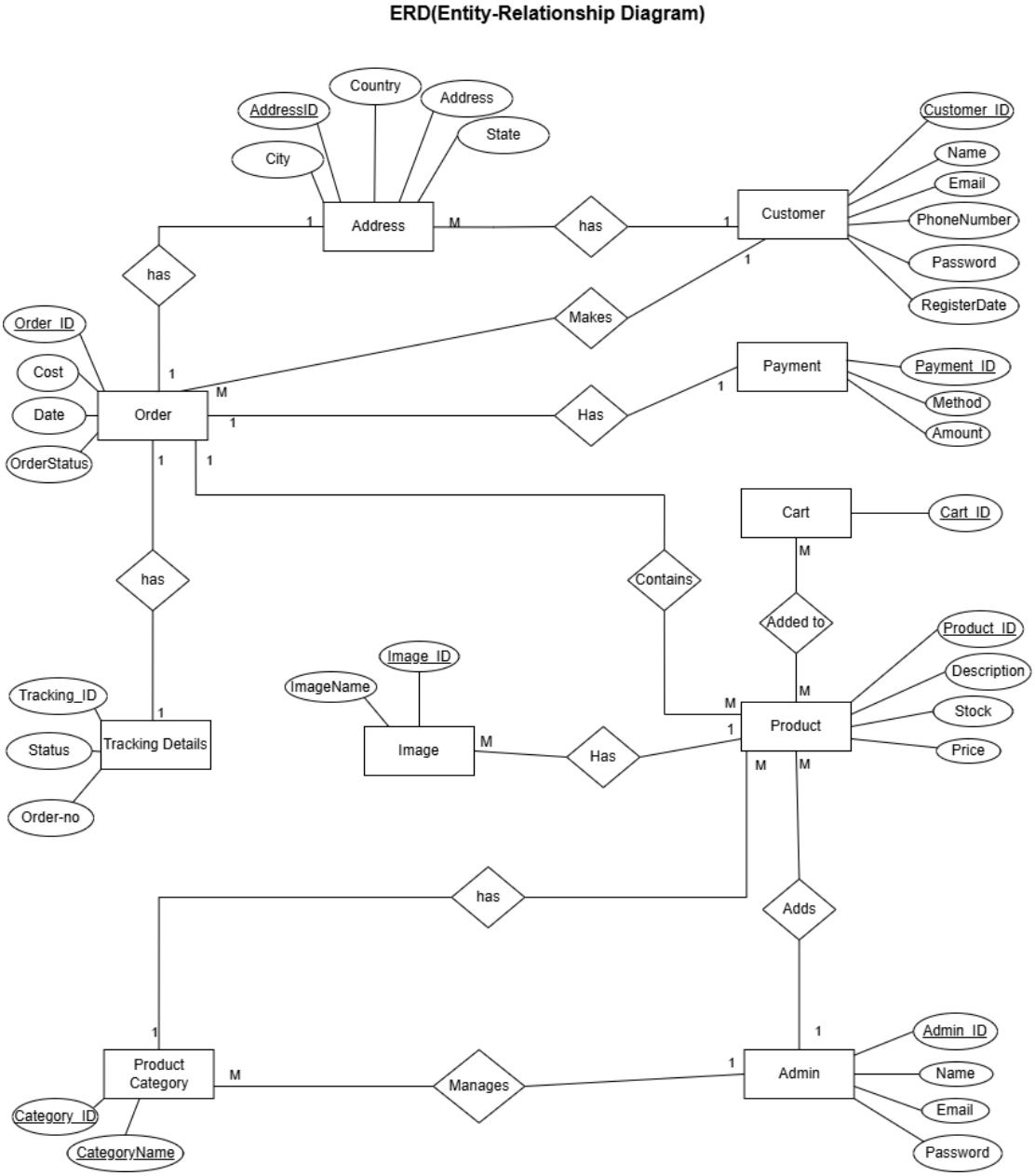
**Architecture Style:**
This refers to the overall design approach or pattern that guides how the system is structured. Common architecture styles include:

- **MVC (Model-View-Controller):** This is a design pattern where the system is divided into three components:
  - **Model:** Handles data and business logic.
  - **View:** The user interface (UI) that the user interacts with.
  - **Controller:** Handles user input and updates the Model and View accordingly.

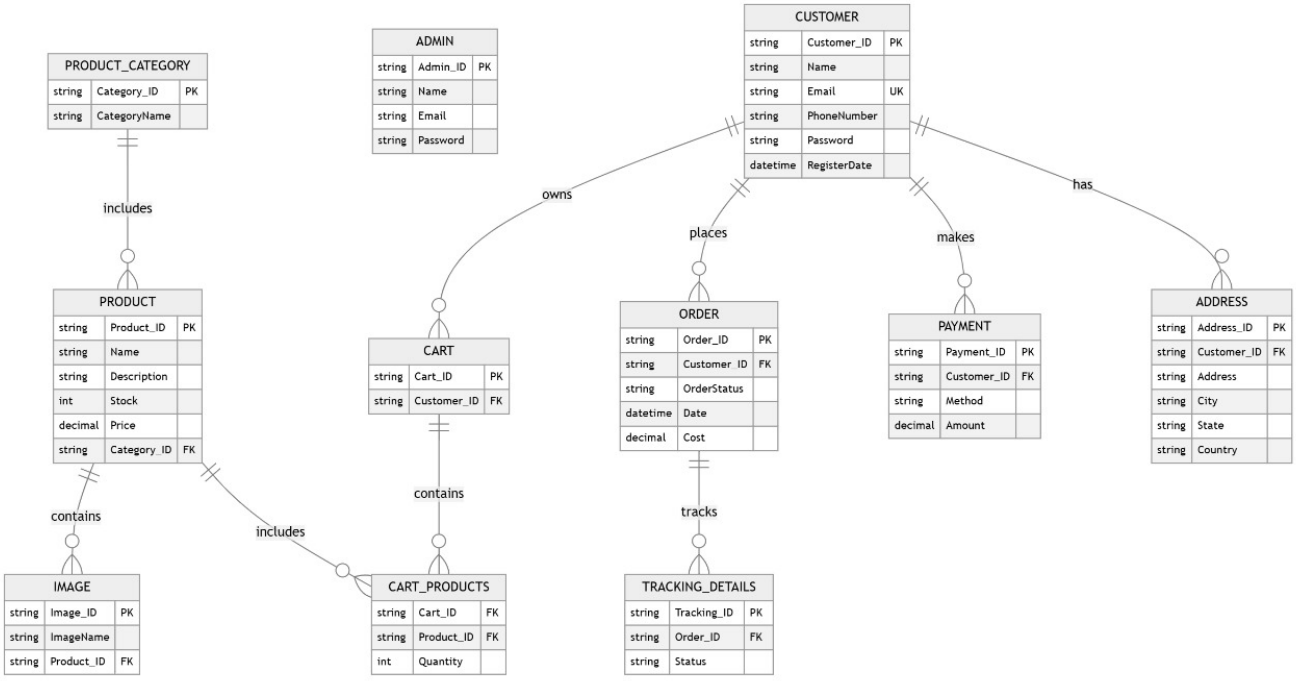
  This separation helps keep the code organized and maintainable.
- **Microservices:** This is a more modern approach where the system is broken down into smaller, independent services. Each microservice is responsible for a specific task, like managing orders or handling payments. These services communicate with each other but can be developed, deployed, and scaled independently.

## 2. Database Design & Data Modeling

### ER Diagram (Entity-Relationship Diagram)



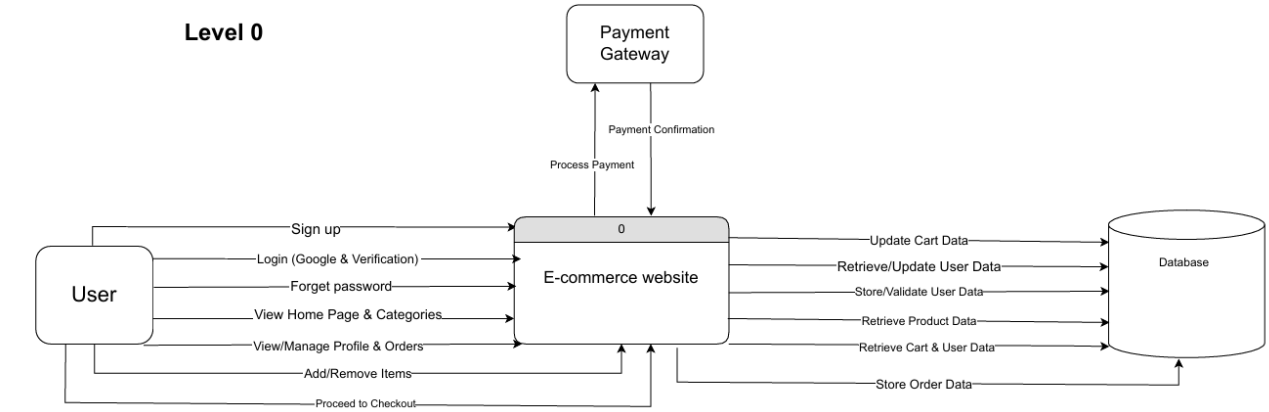
## Logical & Physical Schema

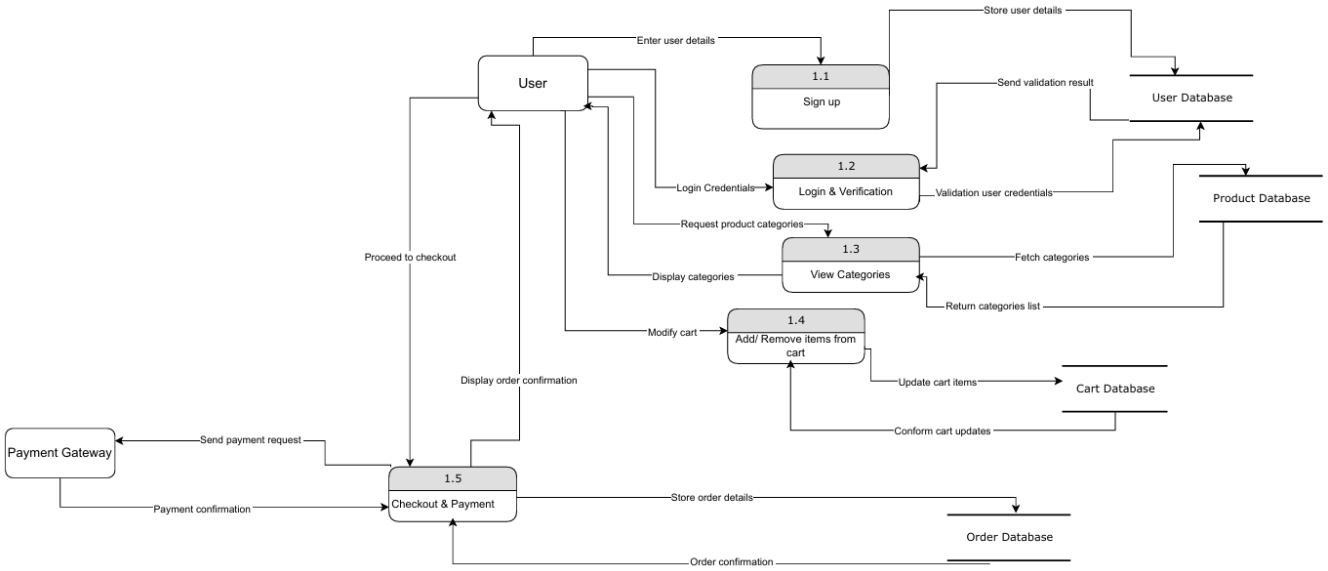



- **Normalization:**
  - The database is normalized to **3NF (Third Normal Form)** to eliminate redundancy.
  - Separate tables for Product, Product_Category, Image, and Cart ensure **data integrity**.
- **Indexes:**
  - **Primary Indexes** on PKs (e.g., `Customer_ID`, `Order_ID`, `Product_ID`).
  - **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`).

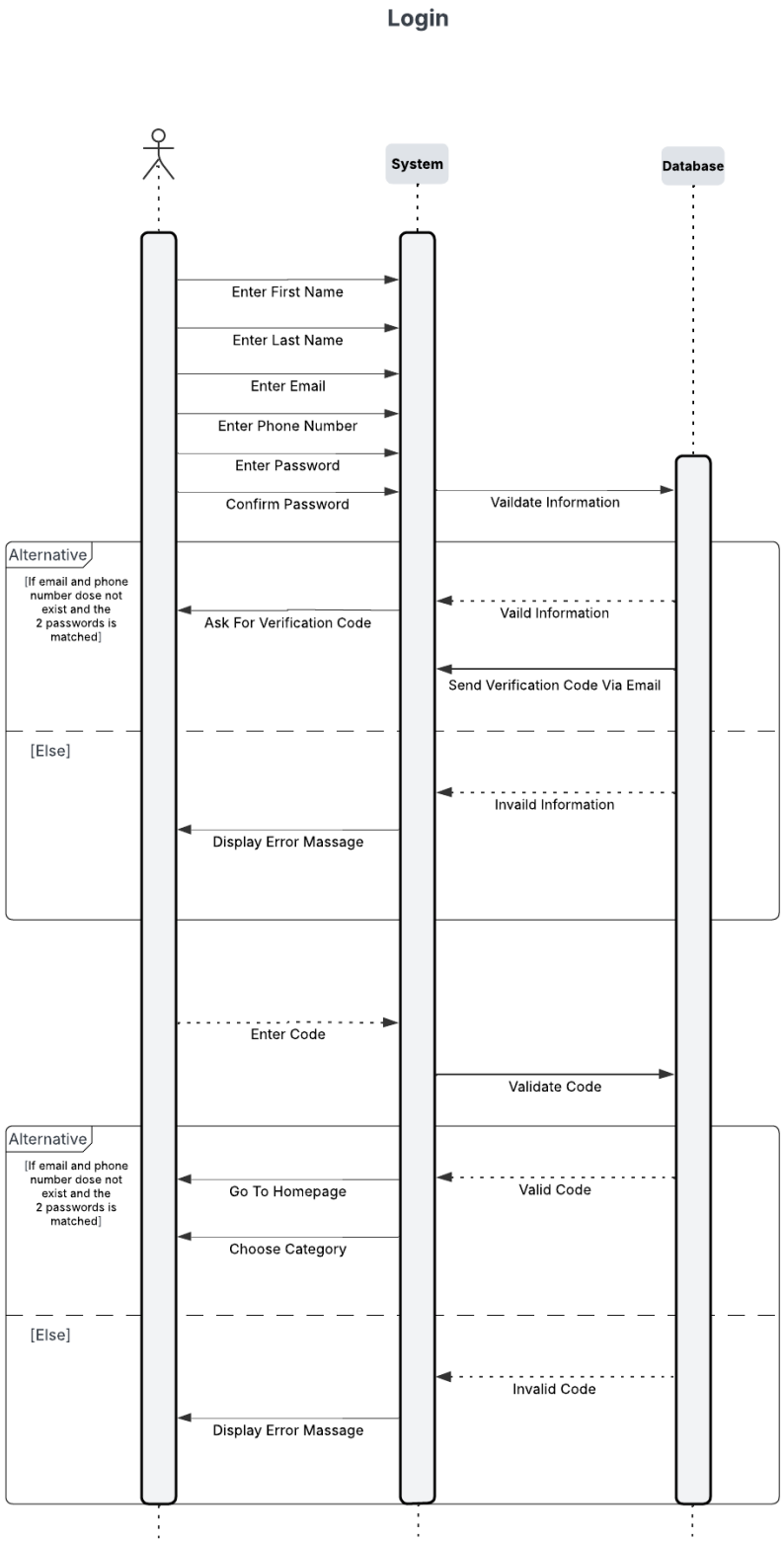
## 3. Data Flow & System Behavior

### DFD (Data Flow Diagram)

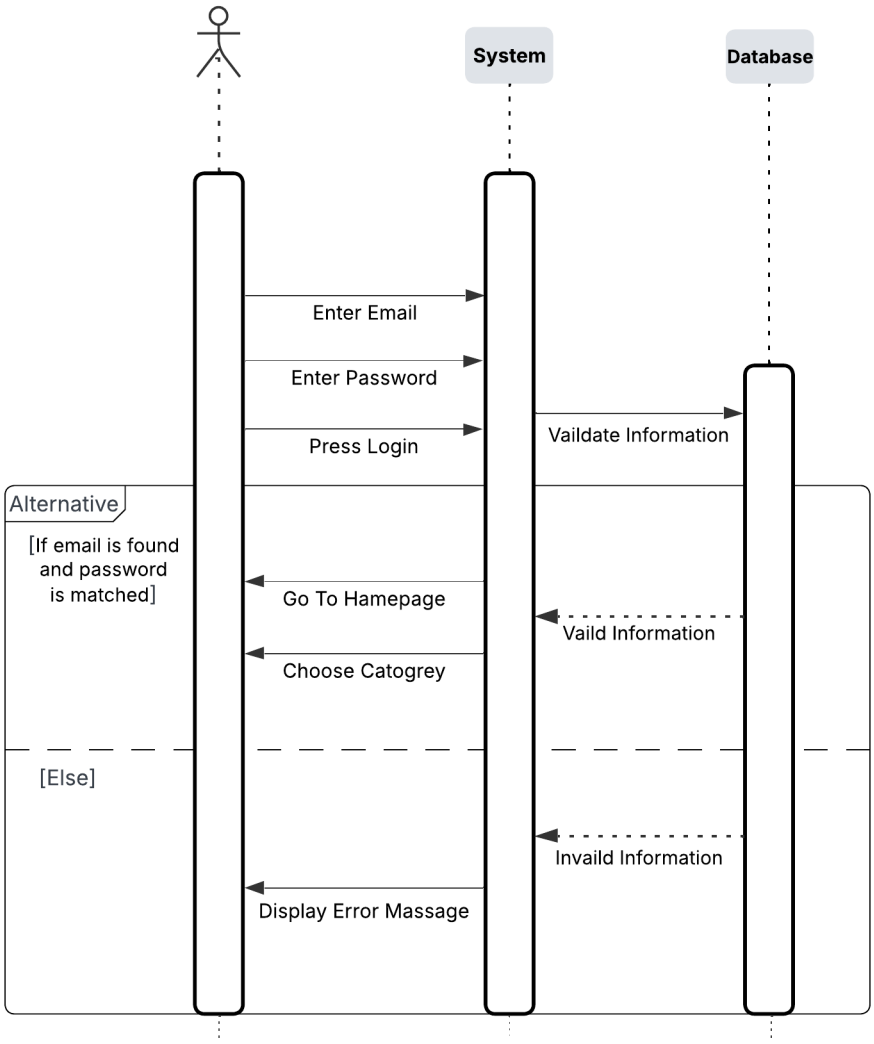


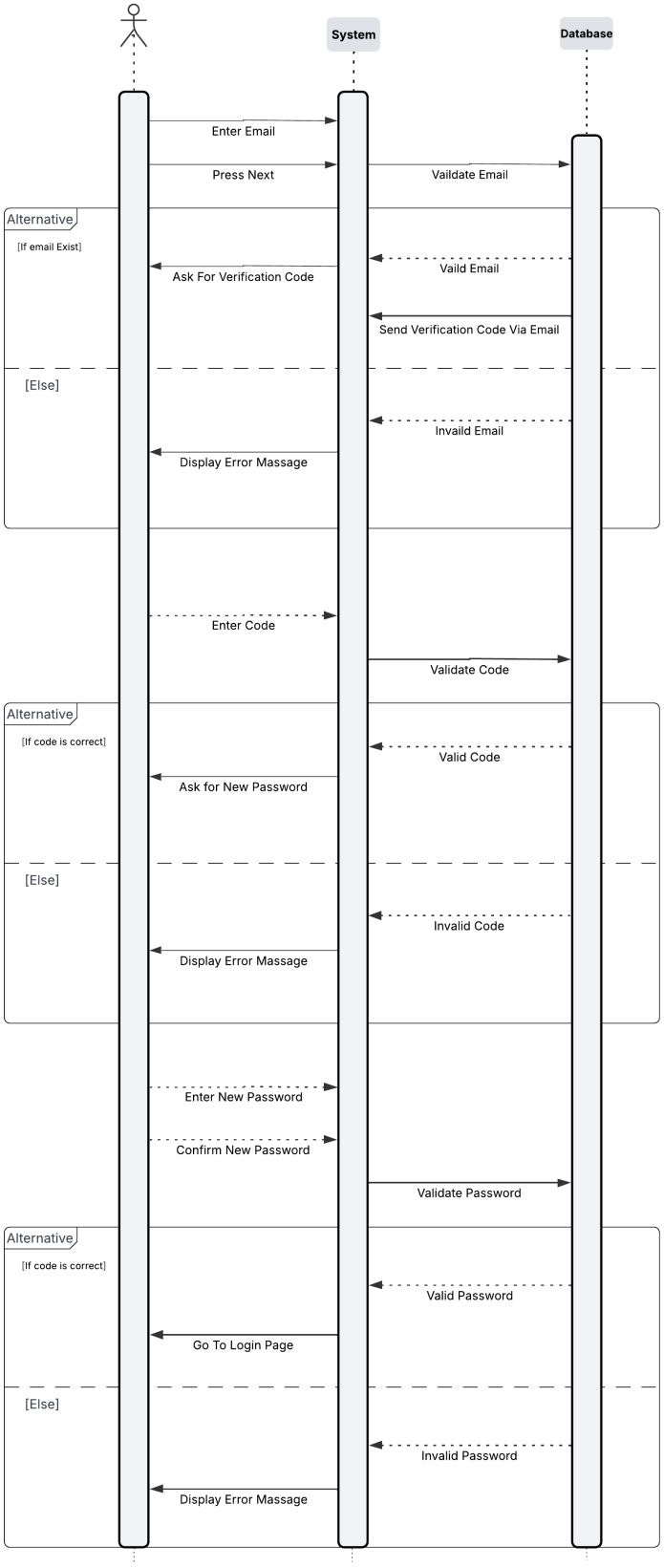
## Sequence Diagrams



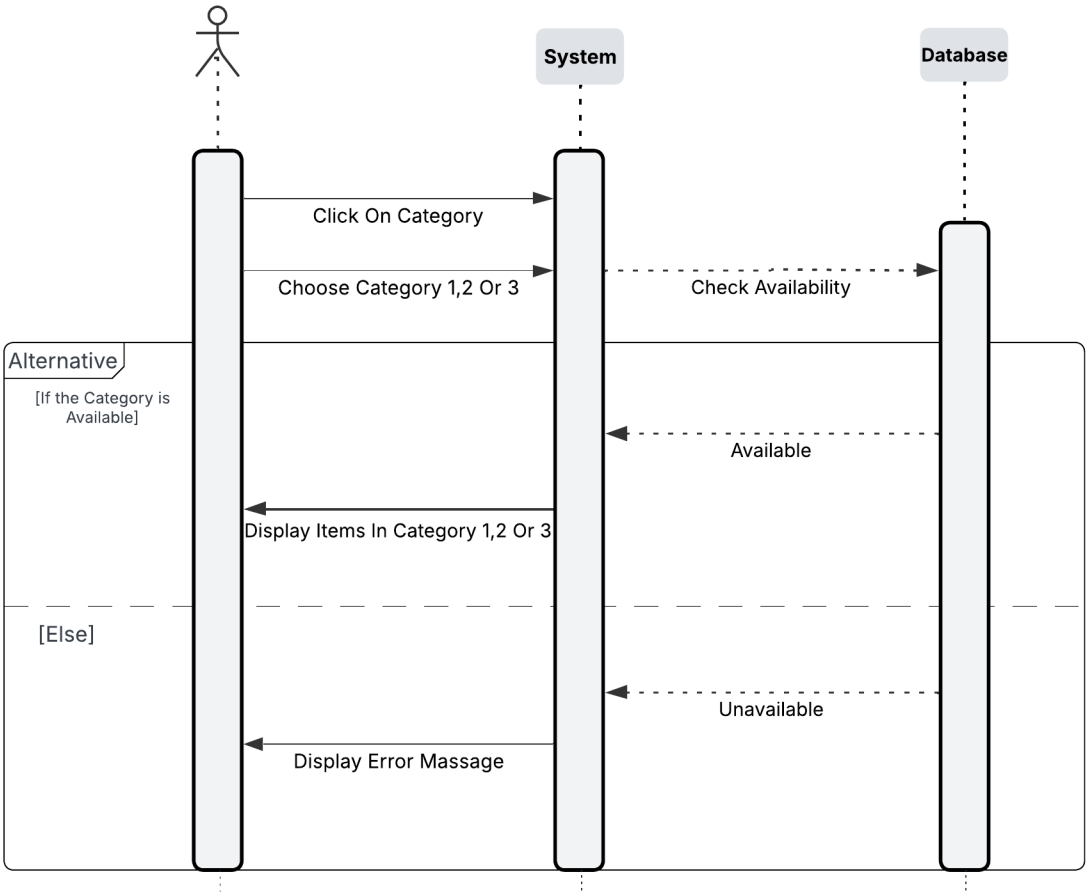
## Sign In


System
Database
Enter Email
Enter Password
Press Login
Vaildate Information
Alternative
[If email is found and password is matched]
Go To Hamepage
Vaild Information
Choose Catogrey
[Else]
Invaild Information
Display Error Massage

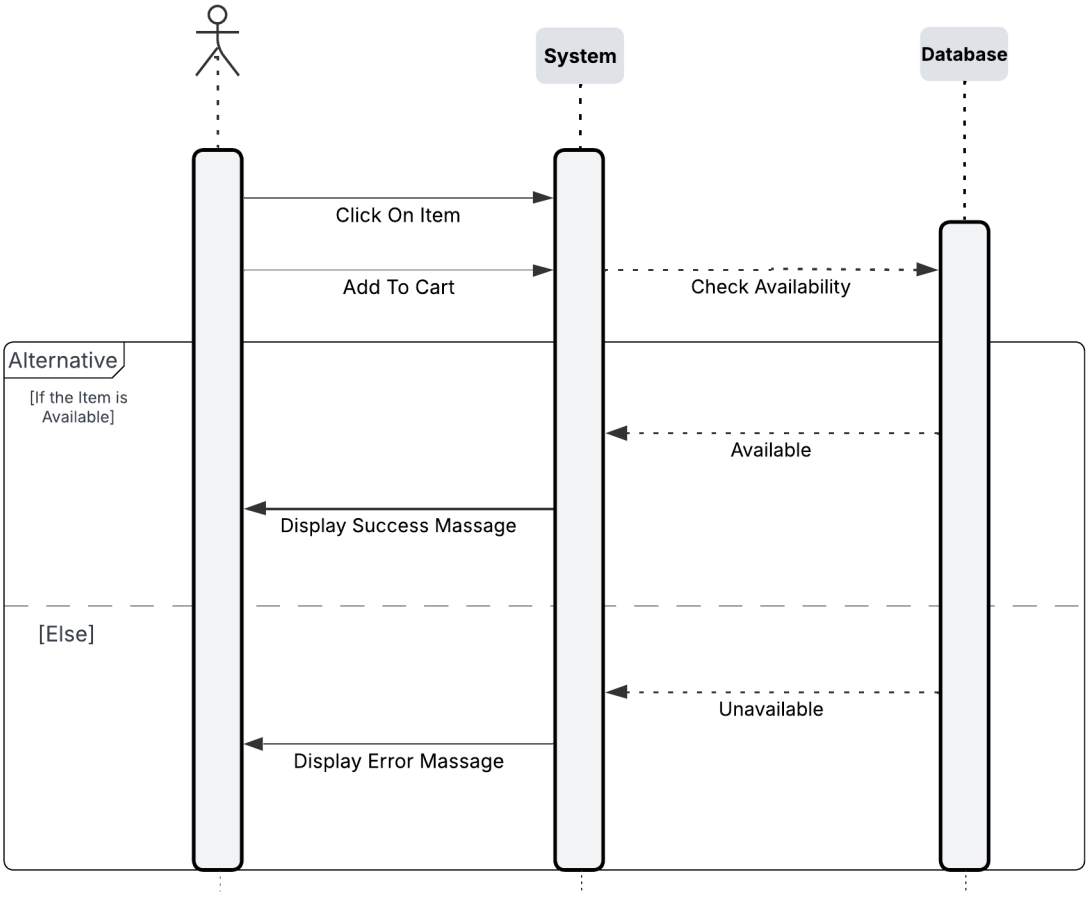
# Forget Password



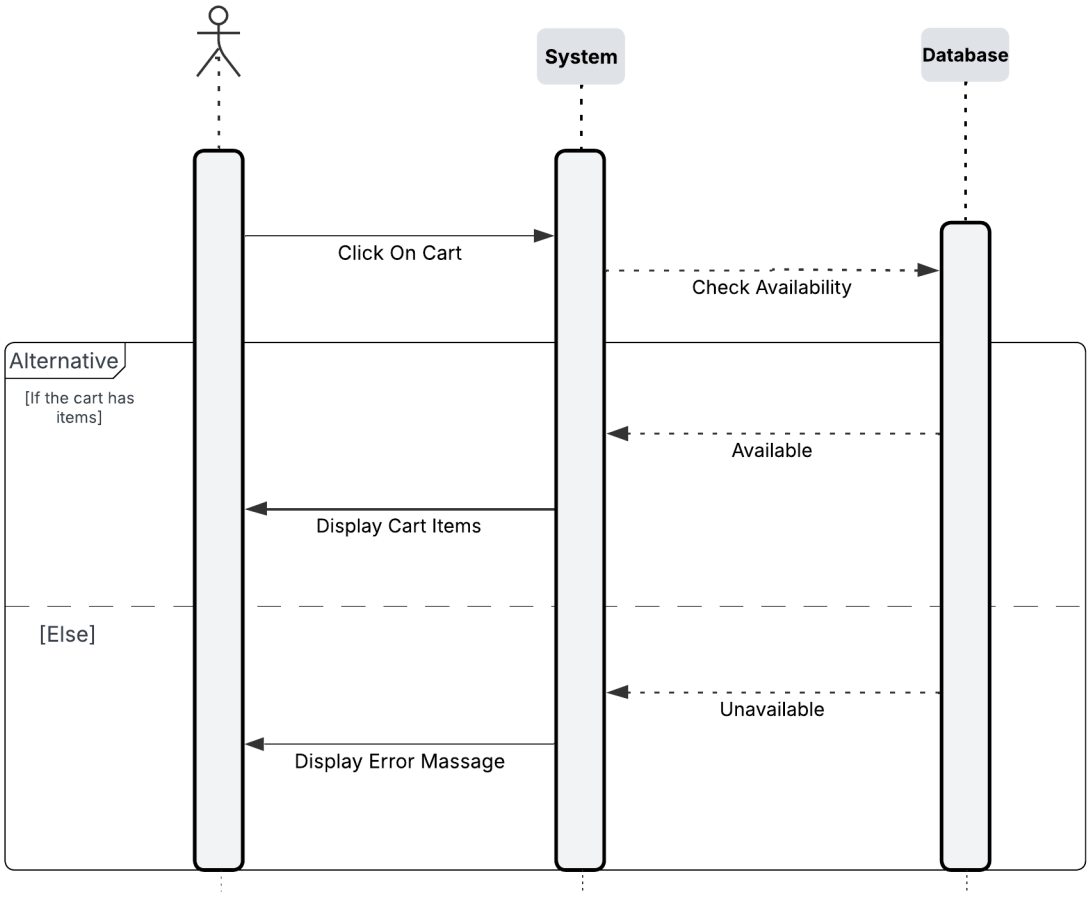
## Choose Category



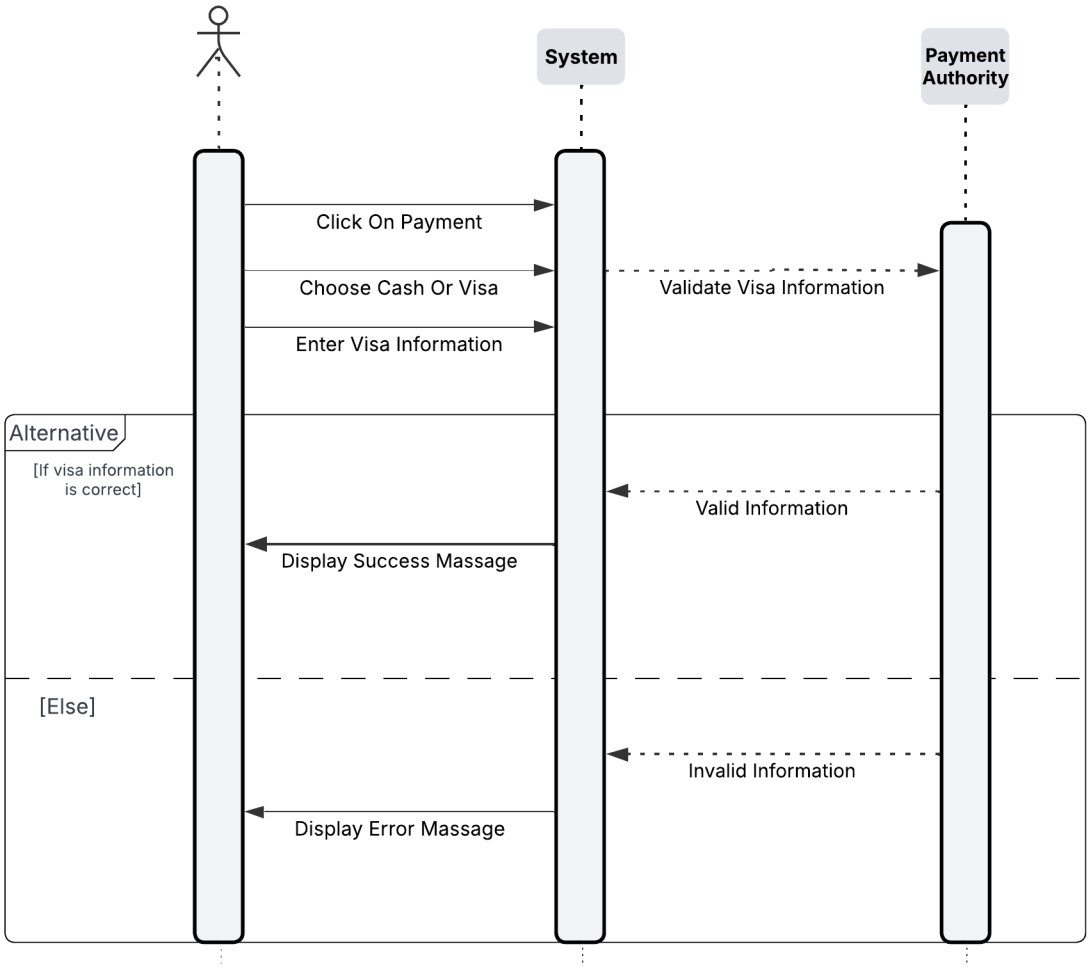
## Add Items


System
Database
Click On Item
Add To Cart
Check Availability
Alternative
[If the Item is Available]
Available
Display Success Massage
[Else]
Unavailable
Display Error Massage

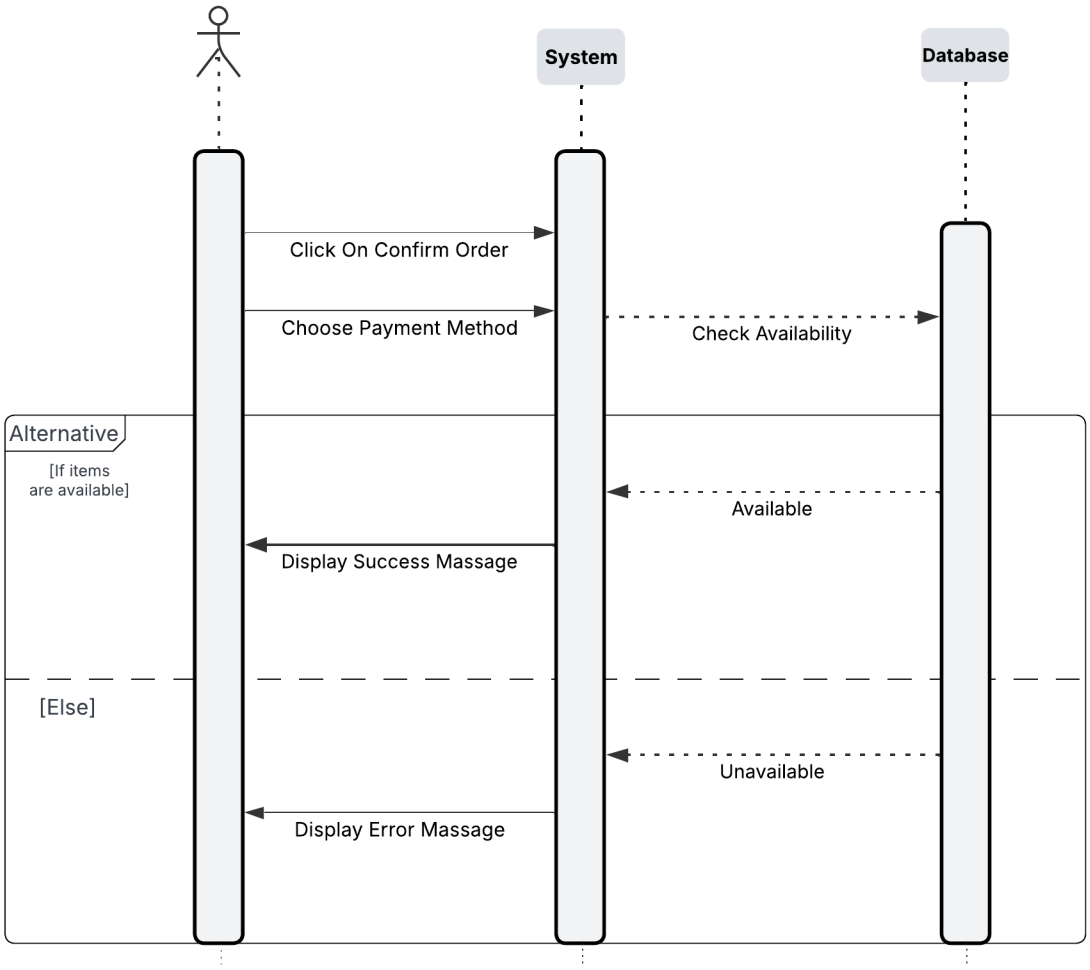
## View Cart Items



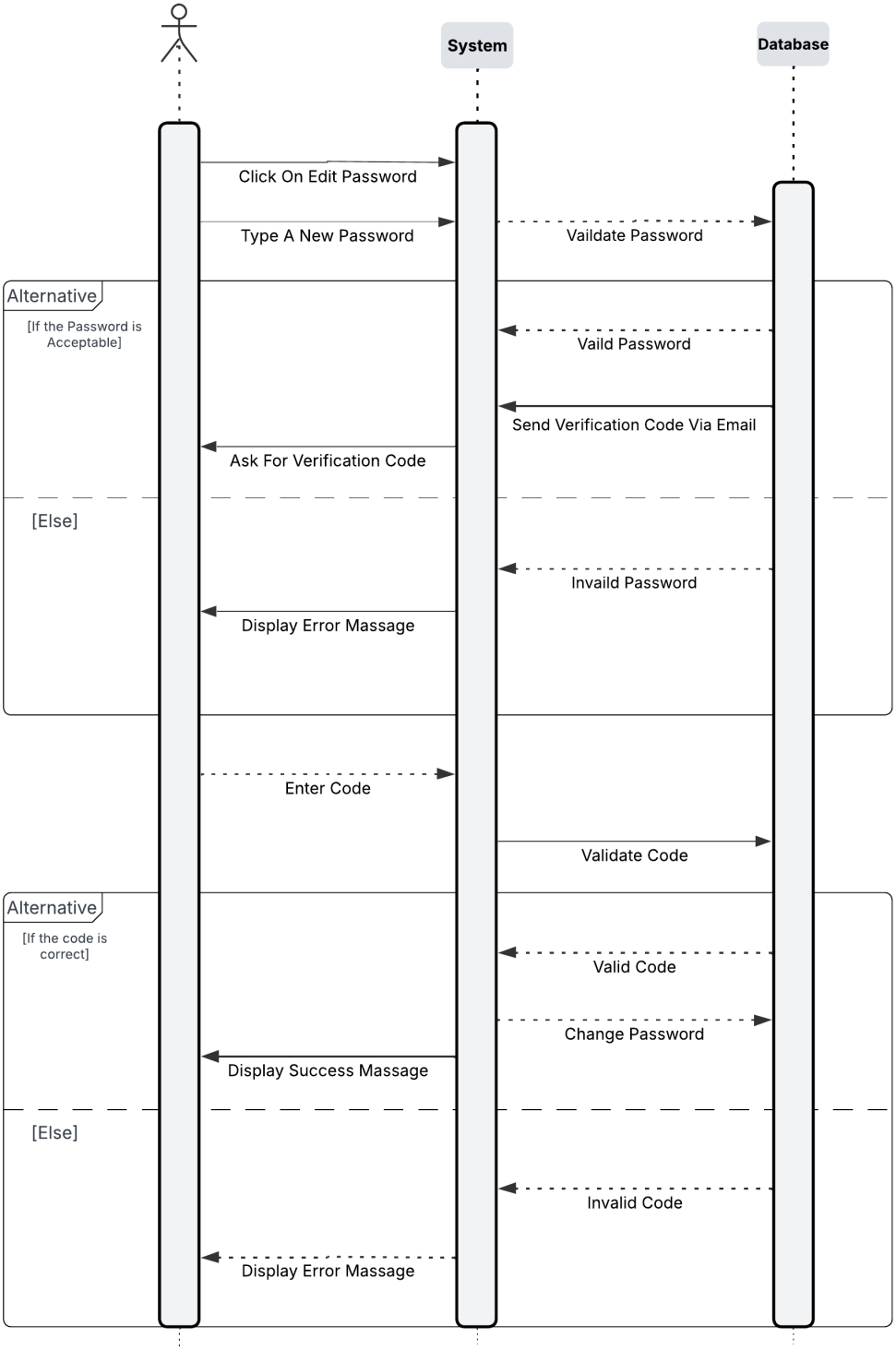
## Add Payment Methode



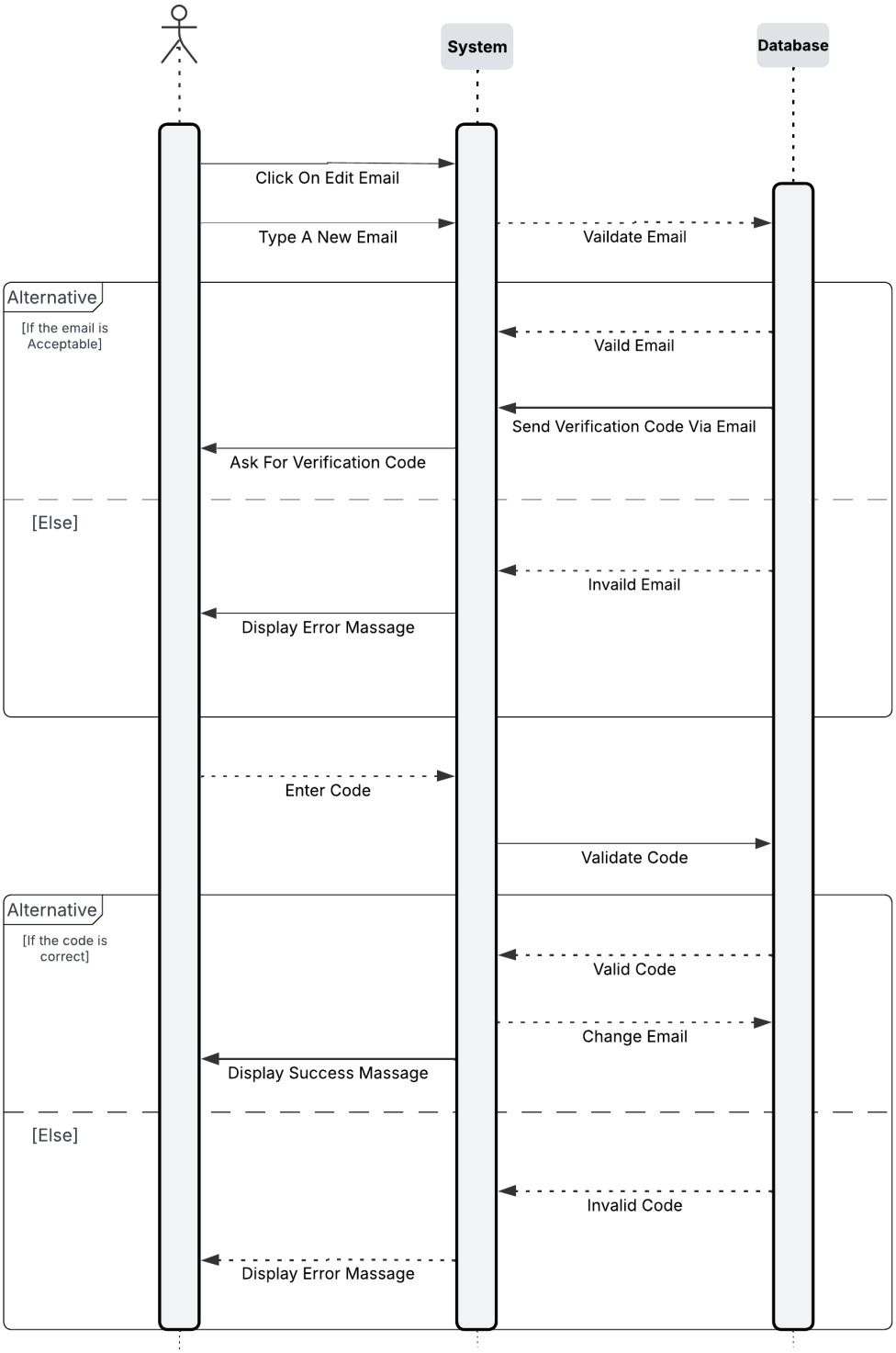
## Confirm Order



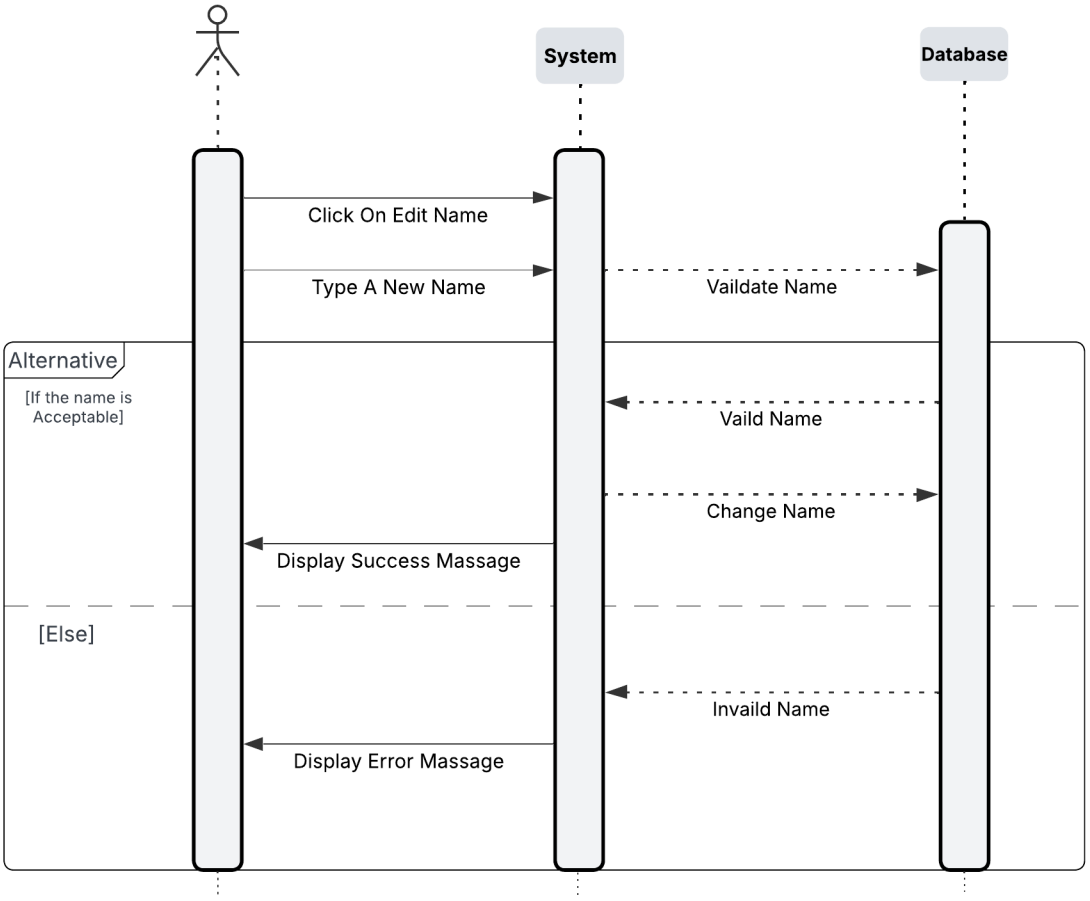
# Change Password



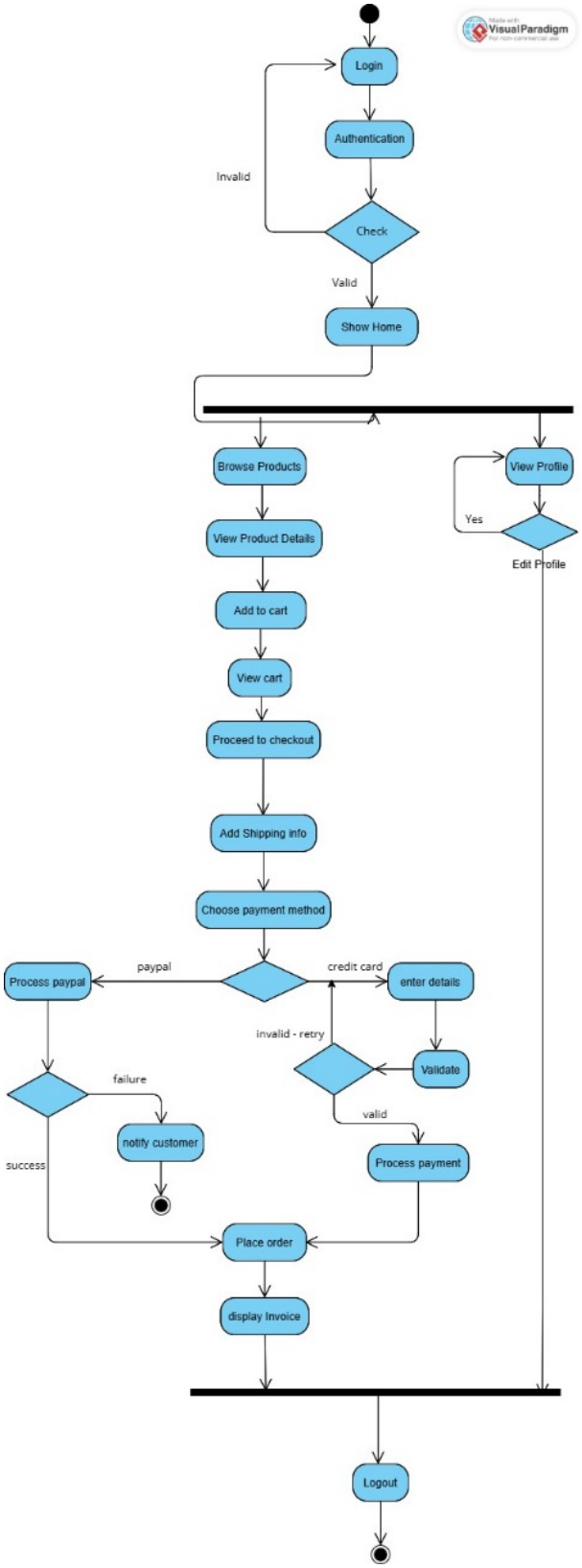
# Change Email



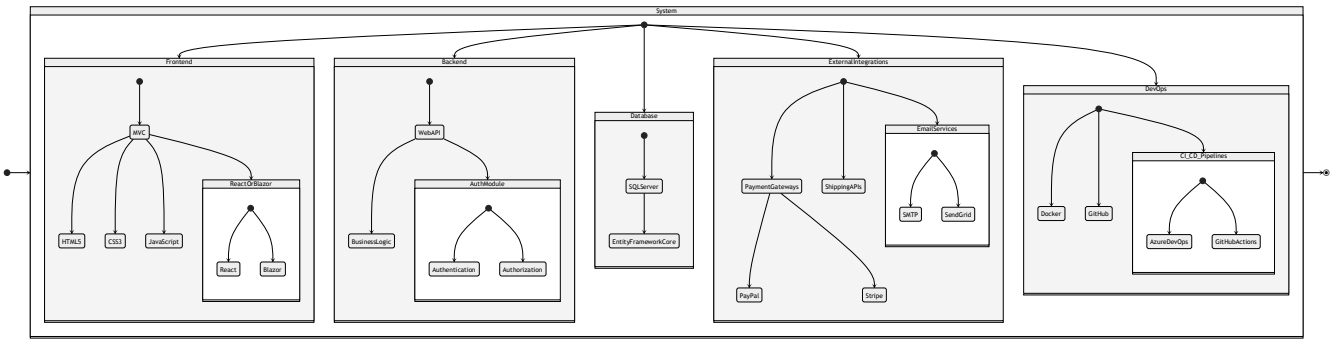
## Change Name


System
Database
Click On Edit Name
Type A New Name
Vaildate Name
Alternative
[If the name is Acceptable]
Vaild Name
Change Name
Display Success Massage
[Else]
Invaild Name
Display Error Massage

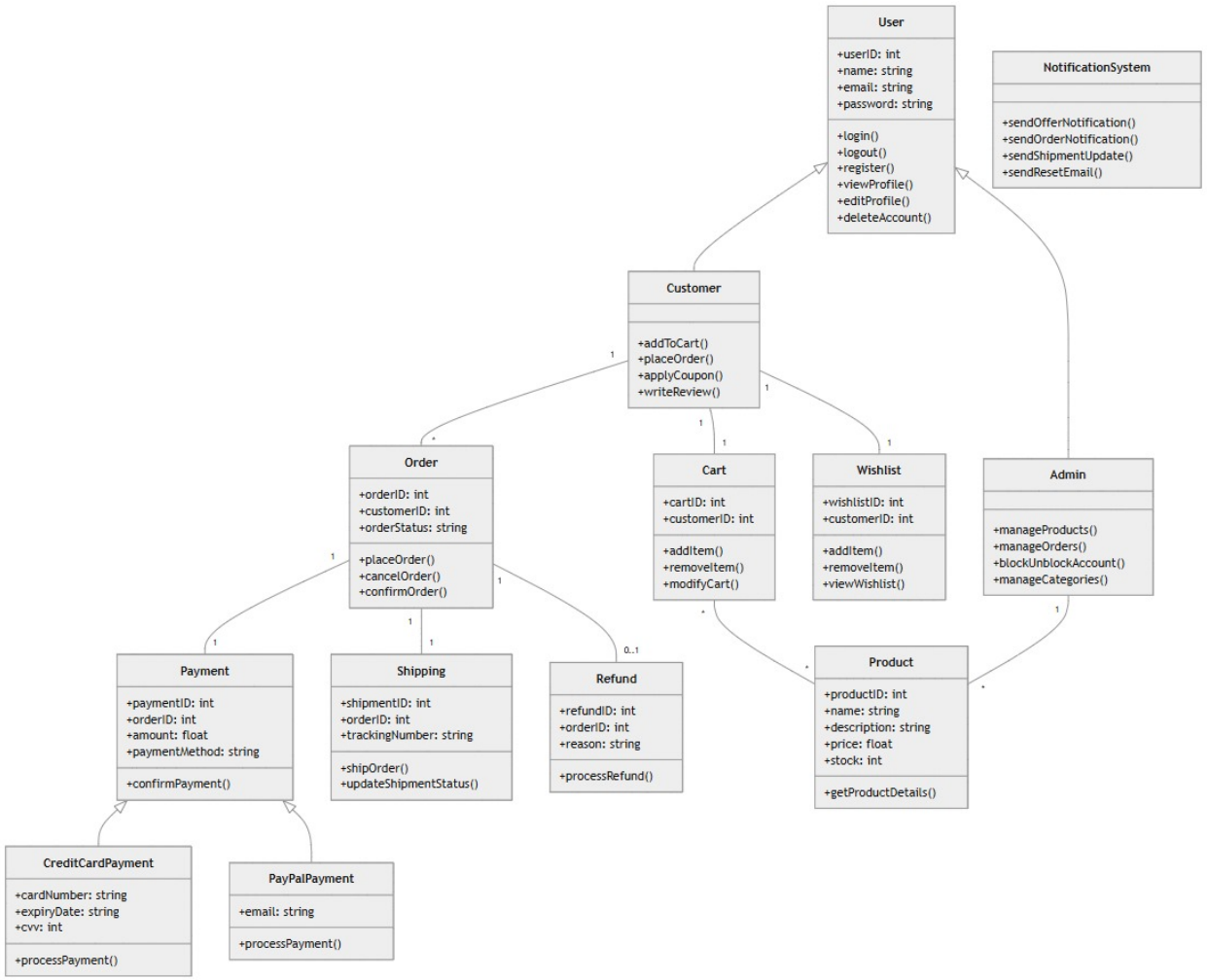
## Activity Diagram



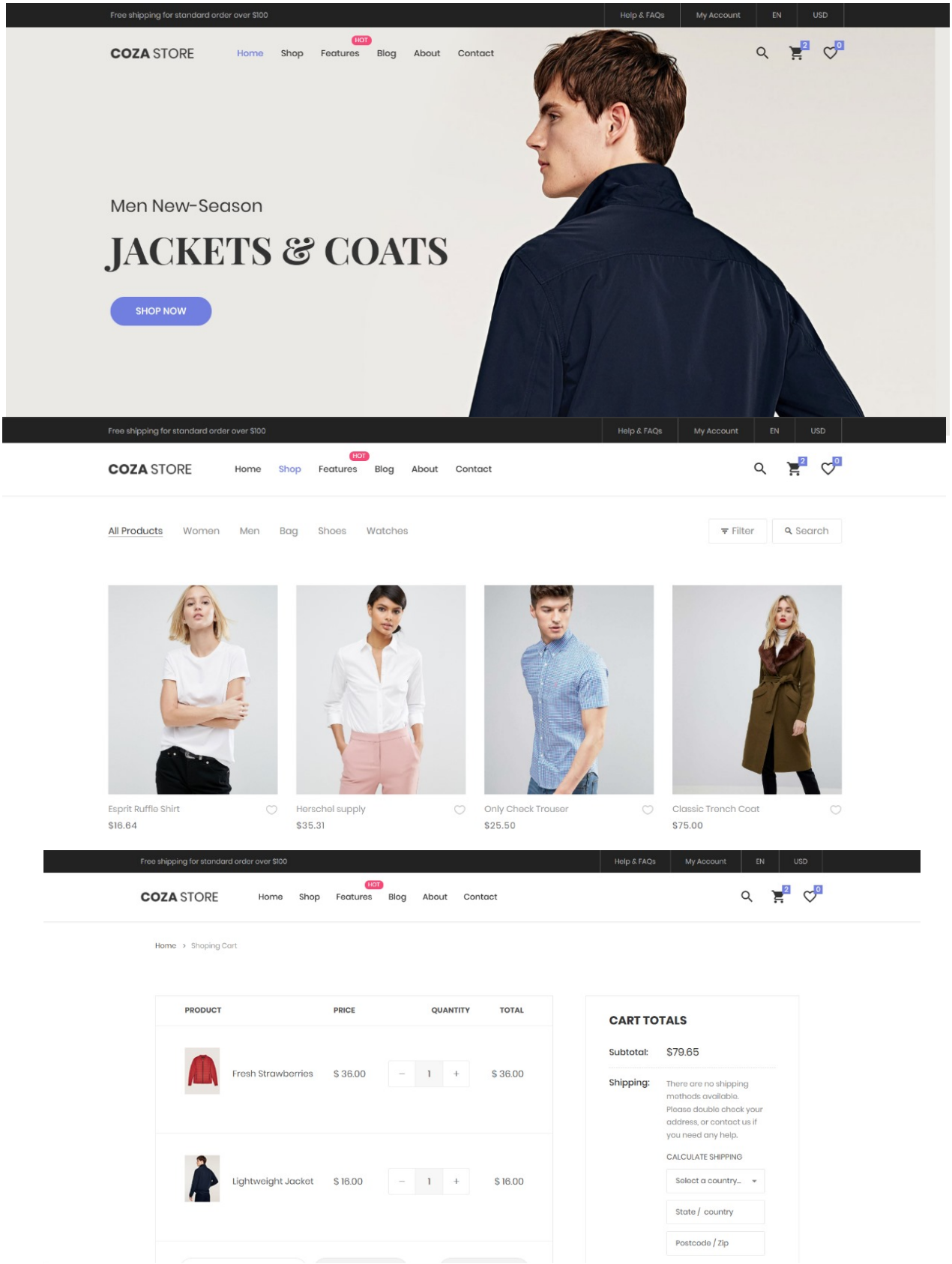
## State Diagram

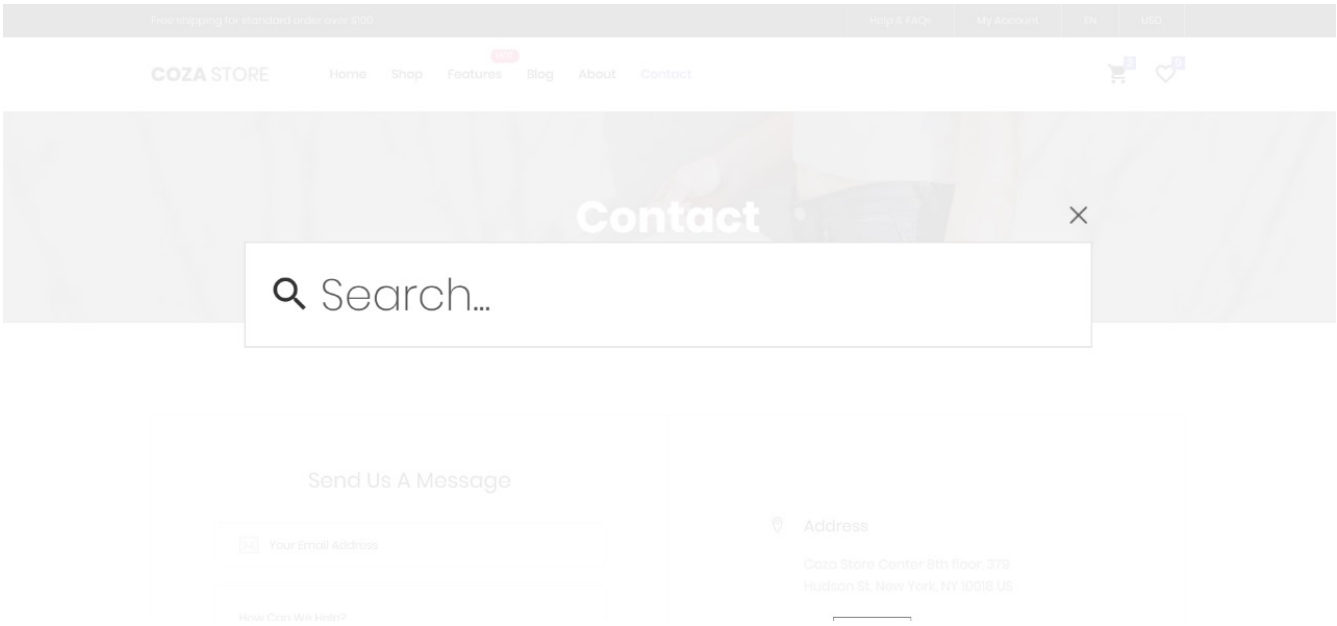



## Class Diagram



## 4. UI/UX Design & Prototyping






**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.

## 6. System Deployment & Integration

### Technology Stack:

**Frontend:**

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

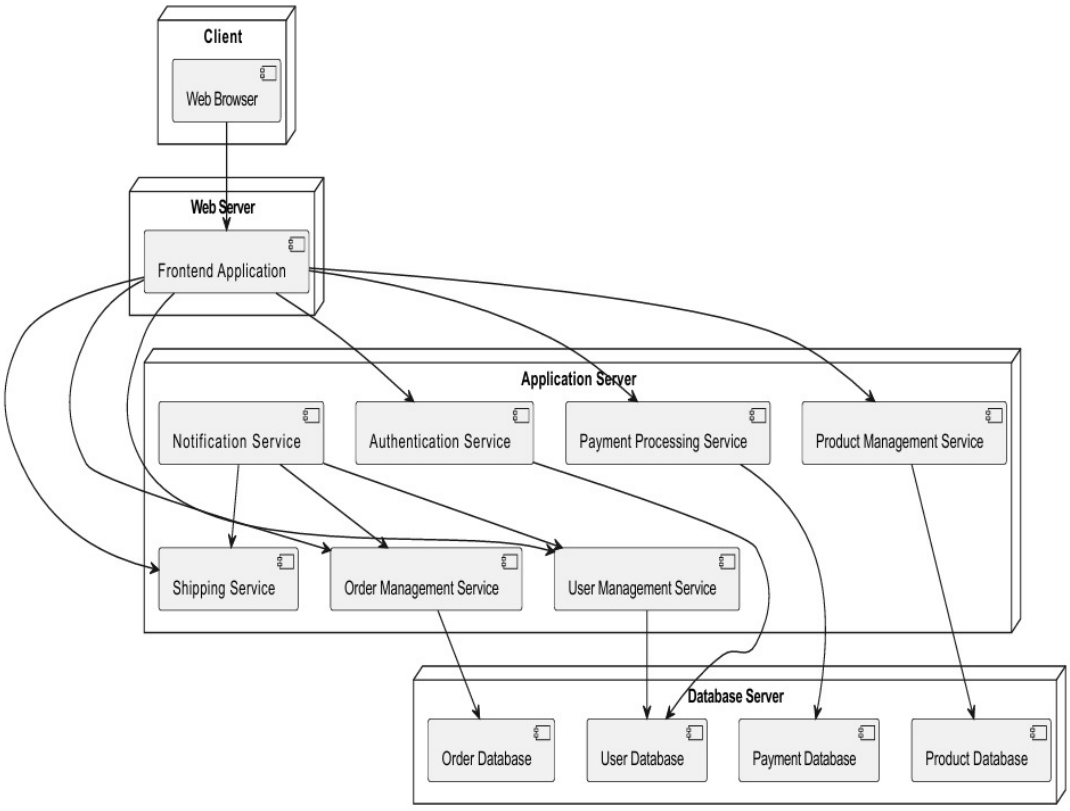
**Backend:**

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

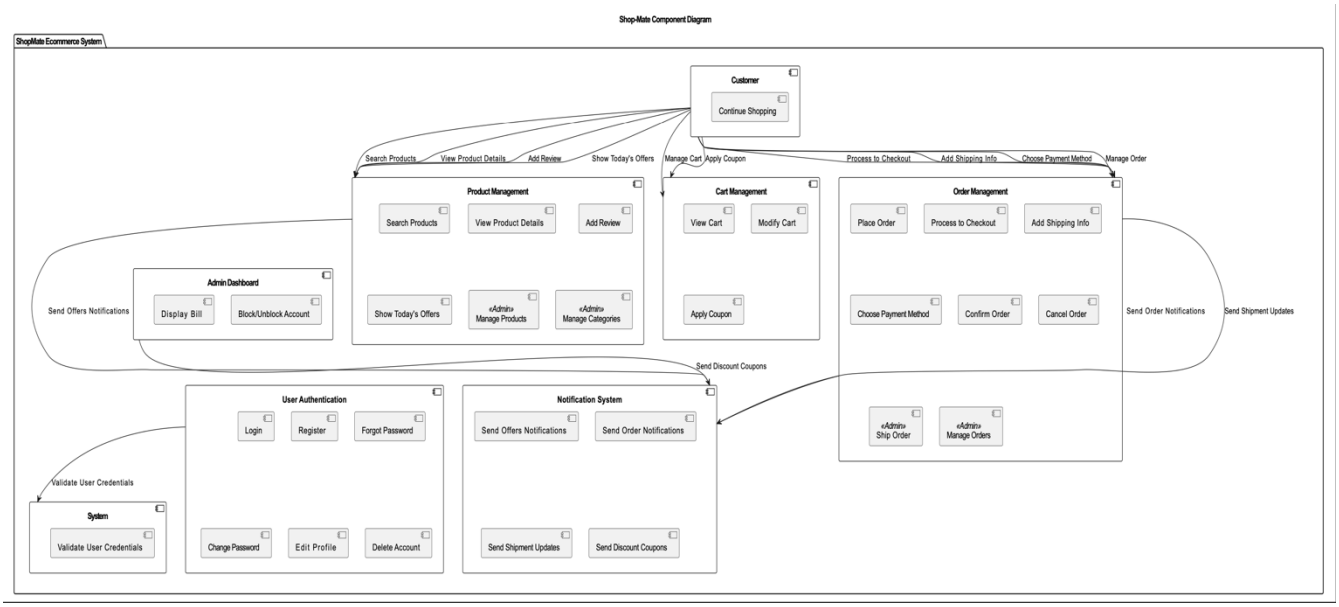
**Database:**

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

### Deployment Diagram



### Component Diagram




**7. 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.