项目文档

# Functional Requirement

1. \*\*Functional Requirements\*\*  
  
1.1 \*\*Customer Account Registration\*\*   
Function ID: FR-01   
Description: Customers can register a new account by providing valid personal information. The system validates the email format and password strength and calls the Plugin API to ensure the email is not already registered.   
Input: Customer personal information (FirstName, LastName, Email, Password).   
Output: A new Customer record stored in the database with a "Verified" status.  
  
1.2 \*\*Customer Login\*\*   
Function ID: FR-02   
Description: Customers can log in using their registered email and password. The system validates the credentials and calls the Plugin API for authentication.   
Input: Email and password.   
Output: A valid Session record created and associated with the Customer.  
  
1.3 \*\*Customer Logout\*\*   
Function ID: FR-03   
Description: Customers can log out of the system, which invalidates their current session.   
Input: User session token.   
Output: Session invalidated and system logs the event.  
  
1.4 \*\*Customer Profile Management\*\*   
Function ID: FR-04   
Description: Customers can view, update, or delete their own profile information, such as name, email, and billing address.   
Input: Customer profile data (FirstName, LastName, Email, BillingAddress, etc.).   
Output: Updated or deleted Customer record in the database.  
  
1.5 \*\*Customer Purchase History View\*\*   
Function ID: FR-05   
Description: Customers can view a list of their completed orders, including order date, items, and total cost.   
Input: CustomerID.   
Output: A list of Order records with associated OrderDetail and Product data.  
  
1.6 \*\*Shopping Cart Management\*\*   
Function ID: FR-06   
Description: Customers can manage the contents of their shopping cart, including adding, removing, or modifying items.   
Input: ProductID, Quantity, CartID, ActionType (Add, Remove, Modify).   
Output: Updated CartItem records in the database with current cart state and recalculated total.  
  
1.7 \*\*Shopping Cart Checkout\*\*   
Function ID: FR-07   
Description: Customers can proceed to checkout by selecting a valid payment method, confirming the order, and converting the shopping cart to an order.   
Input: CartID, PaymentID, ShippingAddress.   
Output: A new Order record created with associated OrderDetails, and the CartItem records deleted.  
  
1.8 \*\*Order Status View\*\*   
Function ID: FR-08   
Description: Customers can view the current status of their orders, such as Processing, Shipped, Delivered, or Cancelled.   
Input: OrderID.   
Output: Order status and related details retrieved and displayed.  
  
1.9 \*\*Order Cancellation\*\*   
Function ID: FR-09   
Description: Customers can cancel an order if it has not yet been shipped. The system updates the order status and logs the event.   
Input: OrderID.   
Output: Order status updated to "Canceled" and the event is logged.  
  
1.10 \*\*Payment Information Management\*\*   
Function ID: FR-10   
Description: Customers can add, update, or delete their payment information.   
Input: CardNumberHash, ExpiryDate, HolderName, BillingAddress, ActionType (Add, Update, Delete).   
Output: Updated or deleted PaymentInfo records in the database.  
  
1.11 \*\*Inventory Management\*\*   
Function ID: FR-12   
Description: Administrators can manage product inventory by adding, updating, or deleting product records.   
Input: ProductID, Name, Description, Price, StockQuantity, ImageURL, CategoryID, ActionType (Add, Update, Delete).   
Output: Updated or deleted Product records in the database.  
  
1.12 \*\*Product Category Management\*\*   
Function ID: FR-13   
Description: Administrators can manage product categories by creating, updating, or deleting Category records.   
Input: CategoryID, Name, Description, ActionType (Create, Update, Delete).   
Output: Updated or deleted Category records in the database.  
  
1.13 \*\*Plugin Installation and Management\*\*   
Function ID: FR-14   
Description: Administrators can install, configure, or uninstall plugins to extend system functionality.   
Input: PluginID, ConfigurationSettings, ActionType (Install, Configure, Uninstall).   
Output: Updated PluginInstance records in the database, and system configuration adjusted accordingly.  
  
1.14 \*\*Order Confirmation Email Notification\*\*   
Function ID: FR-15   
Description: The system sends an email confirmation to the customer after order creation.   
Input: OrderID, CustomerID, Email template data (e.g., items, total price, shipping details).   
Output: EmailNotification record created and Plugin API called to send the email.  
  
1.15 \*\*Discount Code Application\*\*   
Function ID: FR-16   
Description: Customers can apply a discount code during the checkout process to reduce the total cost.   
Input: DiscountCode, CartID.   
Output: Updated CartItem records with adjusted total price and a log of the applied discount.  
  
1.16 \*\*Plugin API Integration\*\*   
Function ID: FR-17   
Description: The system integrates with the Plugin API for payment processing, inventory checks, email notifications, and other external functions.   
Input: PluginID, API request parameters (varies per function).   
Output: Plugin API response confirmation and corresponding system action (e.g., payment processed, inventory updated).  
  
1.17 \*\*Session Management\*\*   
Function ID: FR-18   
Description: The system manages user sessions to ensure secure login and logout operations.   
Input: CustomerID, Session token, IP address, UserAgent.   
Output: Session record created, updated, or deleted based on user activity.  
  
1.18 \*\*Order Detail Management\*\*   
Function ID: FR-19   
Description: Administrators can view and manage the details of each order, such as quantity and unit price per item.   
Input: OrderID, ProductID, Quantity, UnitPrice.   
Output: Updated OrderDetail records in the database and recalculated order total.  
  
1.19 \*\*Shopping Cart to Order Conversion\*\*   
Function ID: FR-20   
Description: The system converts the contents of a Shopping Cart into an Order when the customer completes the checkout process.   
Input: CartID, CustomerID, PaymentID, ShippingAddress.   
Output: New Order and OrderDetail records created, CartItem records deleted, and inventory updated.  
  
1.20 \*\*Email Notification Sending\*\*   
Function ID: FR-21   
Description: The system sends email notifications to customers for events such as order confirmation or account registration.   
Input: CustomerID, Email content (Subject, Body), OrderID (optional).   
Output: EmailNotification record created and Plugin API called for delivery.  
  
1.21 \*\*Administrator Account Management\*\*   
Function ID: FR-22   
Description: Administrators can manage other administrator accounts by creating, updating, or deleting them.   
Input: AdminID, Username, PasswordHash, Role, Permissions, ActionType.   
Output: Updated or deleted Administrator records in the database.  
  
1.22 \*\*System Event Logging\*\*   
Function ID: FR-23   
Description: The system logs all user actions and events for auditing and troubleshooting purposes.   
Input: Event type (e.g., login, checkout, plugin installation), ActorID (CustomerID or AdminID), Timestamp.   
Output: Event log entry stored in the system.  
  
1.23 \*\*Shopping Cart Creation and Access\*\*   
Function ID: FR-24   
Description: A Shopping Cart is created when a customer logs in and can be accessed to view or modify items.   
Input: CustomerID.   
Output: New or existing Shopping Cart record retrieved and displayed.  
  
1.24 \*\*Product Availability Check\*\*   
Function ID: FR-25   
Description: The system checks the availability of products in the inventory via the Plugin API before adding them to the cart or proceeding to checkout.   
Input: ProductID.   
Output: Availability status returned from the Plugin API and reflected in the system.  
  
1.25 \*\*Shopping Cart Total Calculation\*\*   
Function ID: FR-26   
Description: The system calculates the total price of the Shopping Cart based on the quantity and price of the selected items.   
Input: CartItem records (ProductID, Quantity, UnitPrice).   
Output: Updated cart total price and display.  
  
1.26 \*\*Inventory Update on Purchase\*\*   
Function ID: FR-27   
Description: The system updates the stock quantity of products in the inventory after an order is successfully placed.   
Input: OrderDetail records (ProductID, Quantity).   
Output: Updated Product records with reduced StockQuantity.

# External Description

# Chapter 2: External Interfaces  
  
This chapter defines the external interfaces that the system interacts with. These interfaces are categorized into four types: \*\*User Interface\*\*, \*\*Hardware Interface\*\*, \*\*Software Interface\*\*, and \*\*Communication Interface\*\*. Each interface is described in terms of its purpose, interaction method, and relevant data flows to ensure clarity for developers and stakeholders.  
  
---  
  
## 2.1 User Interface  
  
### 2.1.1 Web Application UI   
\*\*Description:\*\* The primary user interface through which customers and administrators interact with the system. It includes forms for registration, login, profile management, product browsing, shopping cart viewing/editing, checkout process, order history viewing, and inventory/category/order management for administrators.   
\*\*Interaction Method:\*\* Users navigate web pages via browsers and interact with HTML forms, buttons, tables, and modal dialogs.   
\*\*Data Flows:\*\*  
- Input from users (e.g., personal information, payment details, product selections).  
- Output to users (e.g., shopping cart summary, order status, error messages).  
  
---  
  
## 2.2 Hardware Interface  
  
### 2.2.1 Customer Devices   
\*\*Description:\*\* The system supports access from standard customer devices such as desktops, laptops, tablets, and smartphones.   
\*\*Interaction Method:\*\* Standard HTTP(S) communication over wired or wireless internet connections.   
\*\*Supported Features:\*\*  
- Responsive design for different screen sizes.  
- Compatibility with major operating systems (Windows, macOS, iOS, Android).  
  
### 2.2.2 Administrator Workstations   
\*\*Description:\*\* Administrators use standard computing devices to manage products, categories, orders, plugins, and other administrative functions.   
\*\*Interaction Method:\*\* Same as above, but restricted to authenticated users with admin roles.   
\*\*Security Requirements:\*\* Secure login and session management required.  
  
---  
  
## 2.3 Software Interface  
  
### 2.3.1 Plugin API Integration   
\*\*Description:\*\* A third-party Plugin API used to perform critical functions including email verification during registration, payment processing, email notifications, inventory checks, and authentication.   
\*\*Interaction Method:\*\* RESTful API calls using HTTPS protocol.   
\*\*Key Functions Supported:\*\*  
- Email validation (used in FR-01, FR-15)  
- Payment processing (used in FR-07, FR-10)  
- Inventory availability check (used in FR-25)  
- Authentication (used in FR-02)  
- Email notification delivery (used in FR-15, FR-21)  
  
\*\*Sample Inputs/Outputs:\*\*  
- \*\*Input:\*\* PluginID, request parameters (e.g., email, payment token, template data)  
- \*\*Output:\*\* JSON response indicating success/failure, along with any necessary data (e.g., payment confirmation, email delivery status)  
  
### 2.3.2 Database Management System (DBMS)   
\*\*Description:\*\* Backend relational database system responsible for storing and managing all persistent data including customer accounts, orders, products, sessions, payments, logs, and plugin configurations.   
\*\*Interaction Method:\*\* SQL queries executed by the application server to read/write data.   
\*\*Key Tables Involved:\*\*  
- `Customer`, `Administrator`, `Product`, `Category`, `Order`, `OrderDetail`, `PaymentInfo`, `ShoppingCart`, `CartItem`, `PluginInstance`, `EmailNotification`, `EventLog`  
  
\*\*Data Integrity Requirements:\*\* ACID-compliant transactions for operations involving order creation, payment updates, and inventory changes (FR-07, FR-20, FR-27).  
  
### 2.3.3 Session Management Module   
\*\*Description:\*\* A software module that manages user sessions securely, handling login, logout, and session expiration.   
\*\*Interaction Method:\*\* Token-based authentication using session tokens stored in cookies or local storage.   
\*\*Security Mechanisms:\*\*  
- Encrypted session tokens  
- IP address and User-Agent validation  
- Expiration after inactivity or explicit logout  
  
\*\*Used By:\*\* FR-02, FR-03, FR-18  
  
---  
  
## 2.4 Communication Interface  
  
### 2.4.1 Email Notification Service   
\*\*Description:\*\* External email service used to send automated emails to customers for events like account registration, order confirmation, and other notifications.   
\*\*Interaction Method:\*\* Triggered via Plugin API (see Section 2.3.1), using SMTP or similar protocols.   
\*\*Sample Emails Sent:\*\*  
- Account registration confirmation (FR-01)  
- Order confirmation (FR-15, FR-21)  
  
\*\*Inputs Required:\*\*  
- Recipient email address  
- Email subject and body content  
- Optional attachments or template variables  
  
\*\*Output:\*\* Email sent confirmation or failure status recorded in `EmailNotification` table.  
  
### 2.4.2 Web Browsing (HTTP/HTTPS)   
\*\*Description:\*\* Primary communication protocol used between client browsers and the server.   
\*\*Interaction Method:\*\* RESTful endpoints exposed over HTTPS for secure data transmission.   
\*\*Endpoints Used For:\*\*  
- Registration, login/logout (FR-01–03)  
- Shopping cart operations (FR-06, FR-24, FR-26)  
- Order processing (FR-07–09, FR-20, FR-27)  
- Profile and payment management (FR-04, FR-10)  
- Admin operations (FR-11–14, FR-19, FR-22)  
  
\*\*Security Requirements:\*\*  
- TLS encryption  
- Input validation and sanitization  
- Protection against CSRF/XSS attacks  
  
---  
  
## Cross-Check Summary  
  
| Functional Requirement | Referenced External Data Source | Covered in This Chapter |  
|------------------------|----------------------------------|----------------------------|  
| FR-01 (Registration) | Plugin API (email check), DBMS (customer data) | Yes (2.3.1, 2.3.2) |  
| FR-02 (Login) | Plugin API (auth), DBMS, Session Module | Yes (2.3.1, 2.3.2, 2.3.3) |  
| FR-03 (Logout) | Session Module | Yes (2.3.3) |  
| FR-04 (Profile) | DBMS | Yes (2.3.2) |  
| FR-05 (Purchase History) | DBMS | Yes (2.3.2) |  
| FR-06–07, 20, 26–27 (Cart & Checkout) | DBMS, Plugin API (inventory), Session | Yes (2.3.1, 2.3.2, 2.3.3) |  
| FR-08–09 (Order Status/Cancellation) | DBMS | Yes (2.3.2) |  
| FR-10 (Payment Info) | DBMS | Yes (2.3.2) |  
| FR-11–12, 19 (Admin Inventory/Products) | DBMS | Yes (2.3.2) |  
| FR-13 (Categories) | DBMS | Yes (2.3.2) |  
| FR-14 (Plugins) | DBMS, Plugin API | Yes (2.3.1, 2.3.2) |  
| FR-15, 21 (Email Notif) | Plugin API, Email Service | Yes (2.3.1, 2.4.1) |  
| FR-16 (Discount Code) | DBMS (optional) | Implied under DBMS (2.3.2) |  
| FR-17 (Plugin API) | Plugin API | Yes (2.3.1) |  
| FR-18 (Session) | Session Module | Yes (2.3.3) |  
| FR-22 (Admin Accounts) | DBMS | Yes (2.3.2) |  
| FR-23 (Event Logging) | DBMS | Yes (2.3.2) |  
| FR-24–25 (Cart & Availability) | DBMS, Plugin API | Yes (2.3.1, 2.3.2) |  
  
All external data sources referenced in the functional requirements have been clearly defined and categorized in this chapter.

# Use Case

Use Case Name: Register Customer Account   
Use Case ID: UC-01   
Actors: Customer, Administrator, Plugin API   
Preconditions:   
1. The system is accessible.   
2. The customer has not previously registered an account.   
3. The plugin API is available for integration and validation.   
  
Postconditions:   
1. A new customer account is created successfully.   
2. The customer receives a confirmation message.   
3. The system logs the registration event.   
  
Main Flow:   
1. The customer navigates to the registration page.   
2. The customer enters their personal information (name, email, password, etc.).   
3. The system validates the email format and password strength.   
4. The system calls the Plugin API to verify if the email is already registered.   
5. The Plugin API returns a confirmation that the email is not registered.   
6. The system saves the customer's information in the Customer data entity.   
7. The system sends a confirmation message to the customer's email.   
8. The customer verifies their email via the confirmation link.   
9. The system updates the customer's status to "verified."   
10. The customer is redirected to the login page.   
  
Alternative Flow:   
1. If the email format is invalid, the system displays an error message and prompts the customer to re-enter a valid email.   
2. If the password does not meet the strength requirements, the system displays an error message and prompts the customer to re-enter a stronger password.   
3. If the Plugin API returns a result that the email is already registered, the system displays an error message and prompts the customer to use a different email.   
4. If the Plugin API is unavailable, the system displays a system error message and logs the issue for the Administrator to review.  
  
Use Case Name: Manage Customer Profile   
Use Case ID: UC-02   
Actors: Customer, Administrator   
Preconditions:   
1. The customer is authenticated and logged in.   
2. The system has access to the Customer data entity.   
3. The customer has an existing account.   
  
Postconditions:   
1. The customer profile is updated or deleted based on the action performed.   
2. The system logs the modification or deletion event.   
3. The Administrator is notified if a profile is deleted.   
  
Main Flow:   
1. The customer navigates to the "Manage Profile" section.   
2. The system displays the current customer profile information.   
3. The customer selects an option to edit or delete their profile.   
4. If editing, the customer modifies the desired information (e.g., address, phone number, email).   
5. The system validates the updated information.   
6. The system saves the updated information to the Customer data entity.   
7. The system displays a success message confirming the profile update.   
8. If deletion, the customer confirms the deletion request.   
9. The system removes the customer profile from the Customer data entity.   
10. The system logs the deletion and notifies the Administrator.   
  
Alternative Flow:   
1. If the customer enters invalid information during editing, the system displays an error message and prompts for correction.   
2. If the system fails to save the updated profile, it displays an error message and logs the issue.   
3. If the customer cancels the deletion request, the system returns to the profile view without making changes.   
4. If the system is unable to notify the Administrator of a deletion, it logs the failure for review.  
  
Use Case Name: View Purchase History   
Use Case ID: UC-03   
Actors: Customer, Administrator   
Preconditions:   
1. The system is accessible.   
2. The customer is authenticated and logged in.   
3. The customer has at least one completed order.   
4. The system has access to the Order data entity.   
  
Postconditions:   
1. The customer views their purchase history.   
2. The system logs the access to the purchase history.   
3. If unauthorized access is detected, the system notifies the Administrator.   
  
Main Flow:   
1. The customer navigates to the "Purchase History" section.   
2. The system retrieves the customer's order data from the Order data entity.   
3. The system displays a list of completed orders with details (order date, product names, prices, order status).   
4. The customer can select an order to view more detailed information.   
5. The system logs the customer's access to the purchase history.   
6. The customer is returned to the main account page or shopping cart.   
  
Alternative Flow:   
1. If the customer has no completed orders, the system displays a message indicating that there is no purchase history available.   
2. If the system cannot retrieve the order data, it displays an error message and logs the issue for the Administrator.   
3. If the customer attempts to access another user's purchase history, the system denies the request and logs the unauthorized access.   
4. If the system fails to display the order details, it shows an error message and logs the issue for troubleshooting.  
  
Use Case Name: Add Payment Information   
Use Case ID: UC-04   
Actors: Customer, Administrator, Plugin API   
Preconditions:   
1. The system is accessible.   
2. The customer is authenticated and logged in.   
3. The system has access to the Payment Info data entity.   
4. The Plugin API is available for payment validation and processing.   
  
Postconditions:   
1. The customer's payment information is added or updated successfully.   
2. The system logs the addition or modification of payment information.   
3. The Administrator is notified if there is a system error during the process.   
  
Main Flow:   
1. The customer navigates to the "Payment Information" section in their account.   
2. The customer selects an option to add new payment information.   
3. The customer enters their payment details (e.g., card number, expiration date, CVV, billing address).   
4. The system validates the format and completeness of the entered information.   
5. The system calls the Plugin API to verify and process the payment information.   
6. The Plugin API confirms the validity of the payment details.   
7. The system stores the verified payment information in the Payment Info data entity.   
8. The system displays a confirmation message that the payment information was added successfully.   
  
Alternative Flow:   
1. If the customer enters incomplete or invalid payment details, the system displays an error message and prompts for correction.   
2. If the Plugin API returns an error (e.g., invalid card number), the system informs the customer and suggests re-entering the payment information.   
3. If the Plugin API is unavailable, the system displays a system error message and logs the issue for the Administrator.   
4. If the system fails to store the payment information, it displays an error message and logs the failure for review.  
  
Use Case Name: Update Payment Information   
Use Case ID: UC-05   
Actors: Customer, Administrator, Plugin API   
Preconditions:   
1. The system is accessible.   
2. The customer is authenticated and logged in.   
3. The customer has existing payment information in the Payment Info data entity.   
4. The Plugin API is available for payment validation and processing.   
  
Postconditions:   
1. The customer's payment information is updated successfully.   
2. The system logs the update event.   
3. The Administrator is notified if there is an error during the update process.   
  
Main Flow:   
1. The customer navigates to the "Payment Information" section in their account.   
2. The system displays the customer's current payment information.   
3. The customer selects an option to edit their payment details.   
4. The customer updates the payment information (e.g., card number, expiration date, CVV, billing address).   
5. The system validates the format and completeness of the updated information.   
6. The system calls the Plugin API to verify and process the updated payment details.   
7. The Plugin API confirms the validity of the updated payment information.   
8. The system updates the Payment Info data entity with the new details.   
9. The system displays a confirmation message that the payment information was updated successfully.   
  
Alternative Flow:   
1. If the customer enters incomplete or invalid payment details, the system displays an error message and prompts for correction.   
2. If the Plugin API returns an error (e.g., invalid card number), the system informs the customer and suggests re-entering the payment information.   
3. If the Plugin API is unavailable, the system displays a system error message and logs the issue for the Administrator.   
4. If the system fails to update the Payment Info data entity, it displays an error message and logs the failure for review.  
  
Use Case Name: Delete Payment Information   
Use Case ID: UC-06   
Actors: Customer, Administrator, Plugin API   
Preconditions:   
1. The system is accessible.   
2. The customer is authenticated and logged in.   
3. The customer has existing payment information in the Payment Info data entity.   
4. The Plugin API is available for payment validation and processing.   
  
Postconditions:   
1. The selected payment information is deleted successfully.   
2. The system logs the deletion event.   
3. The Administrator is notified if there is an error during the deletion process.   
  
Main Flow:   
1. The customer navigates to the "Payment Information" section in their account.   
2. The system displays a list of the customer's saved payment methods.   
3. The customer selects a payment method to delete and confirms the action.   
4. The system validates the selection and confirms the deletion.   
5. The system removes the selected payment information from the Payment Info data entity.   
6. The system displays a confirmation message that the payment information was deleted successfully.   
  
Alternative Flow:   
1. If the customer cancels the deletion request, the system returns to the payment information view without making changes.   
2. If the system fails to validate the selection, it displays an error message and logs the issue.   
3. If the Plugin API is unavailable, the system displays a system error message and logs the issue for the Administrator.   
4. If the system fails to delete the payment information, it displays an error message and logs the failure for review.  
  
Use Case Name: Login to System   
Use Case ID: UC-07   
Actors: Customer, Administrator, Plugin API   
Preconditions:   
1. The system is accessible.   
2. The customer or administrator has a registered and verified account.   
3. The Plugin API is available for authentication and validation.   
  
Postconditions:   
1. The customer or administrator is successfully logged into the system.   
2. The system logs the login event.   
3. If the login fails, the system displays an error message and logs the attempt.   
  
Main Flow:   
1. The user navigates to the login page.   
2. The user enters their registered email and password.   
3. The system validates the email and password format.   
4. The system calls the Plugin API to authenticate the user.   
5. The Plugin API confirms the user's credentials are valid.   
6. The system logs the successful login.   
7. The user is redirected to their respective dashboard (Customer or Administrator).   
  
Alternative Flow:   
1. If the email or password format is invalid, the system displays an error message and prompts the user to re-enter valid information.   
2. If the Plugin API returns an invalid authentication result, the system displays an error message indicating incorrect login details.   
3. If the Plugin API is unavailable, the system displays a system error message and logs the issue for the Administrator.   
4. If the login attempt is unauthorized or suspicious, the system logs the attempt and notifies the Administrator.  
  
Use Case Name: Logout from System   
Use Case ID: UC-08   
Actors: Customer, Administrator   
Preconditions:   
1. The system is accessible.   
2. The customer or administrator is authenticated and logged in.   
  
Postconditions:   
1. The customer or administrator is successfully logged out.   
2. The system invalidates the session and clears session data.   
3. The system logs the logout event.   
  
Main Flow:   
1. The customer or administrator clicks on the "Logout" button in the system.   
2. The system verifies the user's session.   
3. The system invalidates the session and clears session data.   
4. The user is redirected to the login page.   
5. The system logs the logout event.   
  
Alternative Flow:   
1. If the system cannot invalidate the session, it displays an error message and logs the issue for the Administrator.   
2. If the user attempts to access the system after logging out, they are redirected to the login page.   
3. If the system fails to log the logout event, it logs the failure for review.  
  
Use Case Name: Manage Inventory   
Use Case ID: UC-09   
Actors: Administrator, Product, Category   
Preconditions:   
1. The system is accessible.   
2. The administrator is authenticated and logged in.   
3. The system has access to the Product and Category data entities.   
  
Postconditions:   
1. Inventory items are added, updated, or removed as per the administrator’s request.   
2. The system logs the inventory modification event.   
3. Product and Category data entities are updated accordingly.   
  
Main Flow:   
1. The administrator navigates to the "Inventory Management" section.   
2. The system displays the current list of products and their associated categories.   
3. The administrator selects an option to add, update, or delete a product.   
4. If adding, the administrator enters product details (name, price, quantity, category, description).   
5. If updating, the administrator selects a product and modifies its information.   
6. If deleting, the administrator selects a product and confirms the deletion.   
7. The system validates the input and processes the request.   
8. The system updates the Product and Category data entities accordingly.   
9. The system displays a confirmation message for the inventory change.   
  
Alternative Flow:   
1. If the administrator enters invalid or incomplete product information, the system displays an error message and prompts for correction.   
2. If the system fails to update the Product or Category data entities, it displays an error message and logs the failure.   
3. If the administrator cancels the action, the system returns to the inventory view without making changes.   
4. If the selected product has existing orders, the system prevents deletion and displays a warning message.  
  
Use Case Name: Add New Product   
Use Case ID: UC-10   
Actors: Administrator, Product, Category, Plugin API   
Preconditions:   
1. The system is accessible.   
2. The administrator is authenticated and logged in.   
3. The system has access to the Product and Category data entities.   
4. The Plugin API is available for product validation and integration.   
  
Postconditions:   
1. A new product is successfully added to the system.   
2. The product is associated with the correct category.   
3. The system logs the addition event.   
4. The Plugin API confirms the successful integration of the new product.   
  
Main Flow:   
1. The administrator navigates to the "Add New Product" section.   
2. The system displays a form for entering product details.   
3. The administrator enters the product information (name, price, description, quantity, image, and category selection).   
4. The system validates the input data (e.g., price is numeric, quantity is positive, image is uploaded).   
5. The system calls the Plugin API to verify product details (e.g., image format, duplicate product name).   
6. The Plugin API returns confirmation that the product details are valid.   
7. The system saves the new product to the Product data entity and updates the associated Category.   
8. The system displays a confirmation message that the product was added successfully.   
9. The Plugin API is called to integrate the product into external systems if necessary.   
10. The administrator is redirected to the product management page.   
  
Alternative Flow:   
1. If the administrator enters invalid or incomplete product information, the system displays an error message and prompts for correction.   
2. If the Plugin API returns an error (e.g., duplicate product name, invalid image format), the system displays the error and prevents the product from being added.   
3. If the Plugin API is unavailable, the system displays a system error message and logs the issue for the Administrator to review.   
4. If the system fails to save the product to the Product data entity, it displays an error message and logs the failure for troubleshooting.  
  
Use Case Name: Update Product Details   
Use Case ID: UC-11   
Actors: Administrator, Product, Category, Plugin API   
Preconditions:   
1. The system is accessible.   
2. The administrator is authenticated and logged in.   
3. The system has access to the Product and Category data entities.   
4. The product to be updated exists in the Product data entity.   
5. The Plugin API is available for product validation and integration.   
  
Postconditions:   
1. The product details are successfully updated in the system.   
2. The product is associated with the updated or correct category.   
3. The system logs the update event.   
4. The Plugin API confirms the successful integration of the updated product.   
  
Main Flow:   
1. The administrator navigates to the "Product Management" section.   
2. The system displays a list of existing products.   
3. The administrator selects a product and chooses the "Edit" option.   
4. The system retrieves the current product details and displays them in an editable form.   
5. The administrator modifies the product information (e.g., name, price, description, quantity, category, or image).   
6. The system validates the updated input data (e.g., price is numeric, quantity is positive, image is uploaded).   
7. The system calls the Plugin API to verify the updated product details (e.g., image format, duplicate product name).   
8. The Plugin API returns confirmation that the product details are valid.   
9. The system updates the Product data entity with the new information and adjusts the associated Category if needed.   
10. The system displays a confirmation message that the product details were updated successfully.   
11. The Plugin API is called to integrate the updated product into external systems if necessary.   
12. The administrator is redirected to the product management page.   
  
Alternative Flow:   
1. If the administrator enters invalid or incomplete product information, the system displays an error message and prompts for correction.   
2. If the Plugin API returns an error (e.g., duplicate product name, invalid image format), the system displays the error and prevents the product from being updated.   
3. If the Plugin API is unavailable, the system displays a system error message and logs the issue for the Administrator to review.   
4. If the system fails to update the Product data entity, it displays an error message and logs the failure for troubleshooting.  
  
Use Case Name: Delete Product   
Use Case ID: UC-12   
Actors: Administrator, Product, Category, Plugin API   
Preconditions:   
1. The system is accessible.   
2. The administrator is authenticated and logged in.   
3. The system has access to the Product and Category data entities.   
4. The product to be deleted exists in the Product data entity.   
5. The Plugin API is available for product validation and integration.   
  
Postconditions:   
1. The selected product is deleted successfully from the system.   
2. The associated category is updated if necessary.   
3. The system logs the deletion event.   
4. The Plugin API confirms the successful removal of the product from external systems.   
  
Main Flow:   
1. The administrator navigates to the "Product Management" section.   
2. The system displays a list of existing products.   
3. The administrator selects a product and chooses the "Delete" option.   
4. The system confirms the selection and prompts the administrator to confirm the deletion.   
5. The administrator confirms the deletion request.   
6. The system checks if the product is linked to any active orders.   
7. If no active orders are found, the system proceeds to remove the product from the Product data entity.   
8. The system updates the associated Category to reflect the removal of the product.   
9. The system logs the deletion event.   
10. The system calls the Plugin API to synchronize the deletion with external systems.   
11. The Plugin API confirms the successful removal.   
12. The system displays a confirmation message that the product was deleted successfully.   
13. The administrator is redirected to the product management page.   
  
Alternative Flow:   
1. If the administrator cancels the deletion request, the system returns to the product management view without making changes.   
2. If the product is linked to active orders, the system prevents deletion and displays a warning message.   
3. If the Plugin API is unavailable, the system displays a system error message and logs the issue for the Administrator to review.   
4. If the system fails to delete the product or update the Category data entity, it displays an error message and logs the failure for troubleshooting.  
  
Use Case Name: Assign Product to Category   
Use Case ID: UC-13   
Actors: Administrator, Product, Category, Plugin API   
Preconditions:   
1. The system is accessible.   
2. The administrator is authenticated and logged in.   
3. The system has access to the Product and Category data entities.   
4. The product to be assigned exists in the Product data entity.   
5. The Plugin API is available for validation and integration.   
  
Postconditions:   
1. The product is successfully assigned to the selected category.   
2. The Category data entity is updated to reflect the new product association.   
3. The system logs the assignment event.   
4. The Plugin API confirms the successful integration of the product-category relationship.   
  
Main Flow:   
1. The administrator navigates to the "Product Management" section.   
2. The system displays a list of products along with their current category assignments.   
3. The administrator selects a product and chooses the "Assign to Category" option.   
4. The system presents a list of available categories for selection.   
5. The administrator selects the appropriate category for the product.   
6. The system validates the category selection and confirms it is valid.   
7. The system updates the Product data entity with the new category assignment.   
8. The system updates the Category data entity to include the product.   
9. The system logs the assignment event.   
10. The system calls the Plugin API to synchronize the category assignment with external systems.   
11. The Plugin API confirms the successful integration.   
12. The system displays a confirmation message that the product was assigned to the category.   
13. The administrator is redirected to the product management page.   
  
Alternative Flow:   
1. If the selected category is invalid, the system displays an error message and prompts the administrator to choose a valid category.   
2. If the Plugin API returns an error during synchronization, the system displays an error message and prevents the assignment from being finalized.   
3. If the Plugin API is unavailable, the system displays a system error message and logs the issue for the Administrator to review.   
4. If the system fails to update the Product or Category data entities, it displays an error message and logs the failure for troubleshooting.  
  
Use Case Name: Create Product Category   
Use Case ID: UC-14   
Actors: Administrator, Category, Plugin API   
Preconditions:   
1. The system is accessible.   
2. The administrator is authenticated and logged in.   
3. The system has access to the Category data entity.   
4. The Plugin API is available for validation and integration.   
  
Postconditions:   
1. A new product category is successfully created in the system.   
2. The Category data entity is updated with the new category information.   
3. The system logs the category creation event.   
4. The Plugin API confirms the successful integration of the new category.   
  
Main Flow:   
1. The administrator navigates to the "Category Management" section.   
2. The system displays a form for creating a new product category.   
3. The administrator enters the category name and description.   
4. The system validates the input (e.g., name is not empty, format is correct).   
5. The system calls the Plugin API to verify if the category name is already in use.   
6. The Plugin API returns confirmation that the category name is available.   
7. The system saves the new category to the Category data entity.   
8. The system displays a confirmation message that the category was created successfully.   
9. The Plugin API is called to synchronize the new category with external systems if needed.   
10. The administrator is redirected to the category management page.   
  
Alternative Flow:   
1. If the administrator enters an empty or invalid category name, the system displays an error message and prompts for correction.   
2. If the Plugin API returns an error (e.g., duplicate category name), the system informs the administrator and prevents the category from being created.   
3. If the Plugin API is unavailable, the system displays a system error message and logs the issue for the Administrator to review.   
4. If the system fails to save the new category to the Category data entity, it displays an error message and logs the failure for troubleshooting.  
  
Use Case Name: Update Product Category   
Use Case ID: UC-15   
Actors: Administrator, Product, Category, Plugin API   
Preconditions:   
1. The system is accessible.   
2. The administrator is authenticated and logged in.   
3. The system has access to the Product and Category data entities.   
4. The product exists in the Product data entity and is currently assigned to a category.   
5. The Plugin API is available for validation and integration.   
  
Postconditions:   
1. The product is successfully reassigned to a new category.   
2. The Category data entity is updated to reflect the category change.   
3. The system logs the category update event.   
4. The Plugin API confirms the successful integration of the updated category assignment.   
  
Main Flow:   
1. The administrator navigates to the "Category Management" section.   
2. The system displays a list of products and their current category assignments.   
3. The administrator selects a product and chooses the "Update Category" option.   
4. The system presents a list of available categories for reassignment.   
5. The administrator selects the new category for the product.   
6. The system validates the new category selection and confirms it is valid.   
7. The system updates the Product data entity with the new category assignment.   
8. The system updates the Category data entity to reflect the change.   
9. The system logs the category update event.   
10. The system calls the Plugin API to synchronize the updated category with external systems.   
11. The Plugin API confirms the successful integration.   
12. The system displays a confirmation message that the product's category was updated.   
13. The administrator is redirected to the category management page.   
  
Alternative Flow:   
1. If the selected category is invalid, the system displays an error message and prompts the administrator to choose a valid category.   
2. If the Plugin API returns an error during synchronization, the system displays an error message and prevents the category update from being finalized.   
3. If the Plugin API is unavailable, the system displays a system error message and logs the issue for the Administrator to review.   
4. If the system fails to update the Product or Category data entities, it displays an error message and logs the failure for troubleshooting.  
  
Use Case Name: Delete Product Category   
Use Case ID: UC-16   
Actors: Administrator, Category, Plugin API   
Preconditions:   
1. The system is accessible.   
2. The administrator is authenticated and logged in.   
3. The system has access to the Category data entity.   
4. The category to be deleted exists in the Category data entity.   
5. The Plugin API is available for validation and integration.   
  
Postconditions:   
1. The selected product category is deleted successfully from the system.   
2. The Category data entity is updated to remove the category.   
3. The system logs the deletion event.   
4. The Plugin API confirms the successful removal of the category from external systems.   
  
Main Flow:   
1. The administrator navigates to the "Category Management" section.   
2. The system displays a list of existing product categories.   
3. The administrator selects a category and chooses the "Delete" option.   
4. The system confirms the selection and prompts the administrator to confirm the deletion.   
5. The administrator confirms the deletion request.   
6. The system checks if the category is associated with any active products.   
7. If no active products are linked to the category, the system proceeds to remove the category from the Category data entity.   
8. The system logs the deletion event.   
9. The system calls the Plugin API to synchronize the deletion with external systems.   
10. The Plugin API confirms the successful removal.   
11. The system displays a confirmation message that the category was deleted successfully.   
12. The administrator is redirected to the category management page.   
  
Alternative Flow:   
1. If the administrator cancels the deletion request, the system returns to the category management view without making changes.   
2. If the category is linked to active products, the system prevents deletion and displays a warning message.   
3. If the Plugin API is unavailable, the system displays a system error message and logs the issue for the Administrator to review.   
4. If the system fails to delete the category or update the Category data entity, it displays an error message and logs the failure for troubleshooting.  
  
Use Case Name: Add Product to Shopping Cart   
Use Case ID: UC-17   
Actors: Customer, Product, Shopping Cart, Plugin API   
Preconditions:   
1. The system is accessible.   
2. The customer is authenticated and logged in.   
3. The system has access to the Product and Shopping Cart data entities.   
4. The product to be added exists in the Product data entity.   
5. The Plugin API is available for inventory and availability checks.   
  
Postconditions:   
1. The selected product is successfully added to the customer's Shopping Cart.   
2. The product quantity in the Shopping Cart is updated or incremented.   
3. The system logs the addition of the product to the Shopping Cart.   
4. The Plugin API confirms the product is available and in stock.   
  
Main Flow:   
1. The customer browses the product catalog or searches for a product.   
2. The customer selects a product and views its details.   
3. The system calls the Plugin API to check the product's availability and inventory.   
4. The Plugin API confirms the product is available for purchase.   
5. The customer clicks the "Add to Cart" button.   
6. The system retrieves the product information from the Product data entity.   
7. The system adds the product to the Shopping Cart data entity with the default quantity of 1.   
8. The system displays a confirmation message that the product has been added to the cart.   
9. The customer is redirected back to the product page or shopping cart summary.   
  
Alternative Flow:   
1. If the product is out of stock, the system displays a message and prevents it from being added to the cart.   
2. If the Plugin API is unavailable, the system displays a system error message and logs the issue for the Administrator to review.   
3. If the customer adds the same product again, the system increments the quantity in the Shopping Cart instead of creating a new entry.   
4. If the system fails to update the Shopping Cart, it displays an error message and logs the failure for troubleshooting.  
  
Use Case Name: View Shopping Cart Contents   
Use Case ID: UC-18   
Actors: Customer, Shopping Cart, Product, Plugin API   
Preconditions:   
1. The system is accessible.   
2. The customer is authenticated and logged in.   
3. The system has access to the Shopping Cart and Product data entities.   
4. The Plugin API is available for inventory and availability checks.   
  
Postconditions:   
1. The customer views the contents of their Shopping Cart.   
2. The system logs the access to the Shopping Cart.   
3. The Plugin API confirms the availability of all items in the cart.   
4. If the Plugin API is unavailable, the system logs the issue for the Administrator.   
  
Main Flow:   
1. The customer navigates to the "Shopping Cart" section.   
2. The system retrieves the customer's Shopping Cart data from the Shopping Cart data entity.   
3. The system displays the list of products in the cart, including product name, quantity, price, and total.   
4. The system calls the Plugin API to verify the current inventory status of each product in the cart.   
5. The Plugin API confirms that all products are available and in stock.   
6. The system updates the cart display with the latest availability information.   
7. The system logs the customer's access to the Shopping Cart.   
8. The customer can continue to the checkout process or return to shopping.   
  
Alternative Flow:   
1. If the customer's Shopping Cart is empty, the system displays a message indicating that no items are currently in the cart.   
2. If the Plugin API returns an error indicating that a product is out of stock, the system displays a warning message for that product and removes it from the cart or prompts the customer to adjust the quantity.   
3. If the Plugin API is unavailable, the system displays a system error message and logs the issue for the Administrator to review.   
4. If the system fails to retrieve the Shopping Cart data, it displays an error message and logs the failure for troubleshooting.  
  
Use Case Name: Modify Cart Items   
Use Case ID: UC-19   
Actors: Customer, Shopping Cart, Product, Plugin API   
Preconditions:   
1. The system is accessible.   
2. The customer is authenticated and logged in.   
3. The system has access to the Shopping Cart and Product data entities.   
4. The customer's Shopping Cart contains at least one item.   
5. The Plugin API is available for inventory and availability checks.   
  
Postconditions:   
1. The customer's Shopping Cart is updated with modified items (quantity, removal, or replacement).   
2. The system logs the modification event.   
3. The Plugin API confirms the updated inventory status of affected products.   
4. If the Plugin API is unavailable, the system logs the issue for the Administrator to review.   
  
Main Flow:   
1. The customer navigates to the "Shopping Cart" section.   
2. The system retrieves and displays the current contents of the customer's Shopping Cart.   
3. The customer selects an item to modify (e.g., change quantity, remove item, or replace with another product).   
4. If changing quantity, the system updates the quantity in the Shopping Cart data entity.   
5. If removing an item, the system confirms the removal and deletes the item from the cart.   
6. If replacing an item, the system removes the old item and adds the new one.   
7. The system calls the Plugin API to verify the updated inventory status of the affected products.   
8. The Plugin API returns confirmation of the inventory status.   
9. The system updates the cart display with the new or modified items and recalculates the total price.   
10. The system logs the modification of the Shopping Cart.   
11. The customer is redirected back to the cart summary or to the product page.   
  
Alternative Flow:   
1. If the customer attempts to set a quantity higher than available inventory, the system displays an error message and prevents the update.   
2. If the Plugin API is unavailable during inventory verification, the system displays a system error and logs the issue for the Administrator.   
3. If the customer cancels the modification, the system returns to the cart summary without making changes.   
4. If the system fails to update the Shopping Cart data entity, it displays an error message and logs the failure for troubleshooting.  
  
Use Case Name: Remove Product from Cart   
Use Case ID: UC-20   
Actors: Customer, Shopping Cart, Product, Plugin API   
Preconditions:   
1. The system is accessible.   
2. The customer is authenticated and logged in.   
3. The system has access to the Shopping Cart and Product data entities.   
4. The customer's Shopping Cart contains at least one item.   
5. The Plugin API is available for inventory and availability checks.   
  
Postconditions:   
1. The selected product is successfully removed from the customer's Shopping Cart.   
2. The system logs the removal event.   
3. The Plugin API confirms the updated inventory status of the affected product.   
4. If the Plugin API is unavailable, the system logs the issue for the Administrator to review.   
  
Main Flow:   
1. The customer navigates to the "Shopping Cart" section.   
2. The system retrieves and displays the current contents of the customer's Shopping Cart.   
3. The customer selects a product and chooses the "Remove" option.   
4. The system confirms the selection and prompts the customer to confirm the removal.   
5. The customer confirms the removal request.   
6. The system deletes the selected product from the Shopping Cart data entity.   
7. The system recalculates the cart total and updates the display.   
8. The system logs the removal event.   
9. The system calls the Plugin API to update the inventory status of the removed product.   
10. The Plugin API confirms the updated inventory status.   
11. The customer is redirected back to the cart summary or to the product page.   
  
Alternative Flow:   
1. If the customer cancels the removal request, the system returns to the cart summary without making changes.   
2. If the Plugin API is unavailable during inventory update, the system displays a system error message and logs the issue for the Administrator.   
3. If the system fails to remove the product or update the cart, it displays an error message and logs the failure for troubleshooting.  
  
Use Case Name: Checkout Products   
Use Case ID: UC-21   
Actors: Customer, Shopping Cart, Product, Order, Payment Info, Plugin API   
Preconditions:   
1. The system is accessible.   
2. The customer is authenticated and logged in.   
3. The customer's Shopping Cart contains at least one item.   
4. The system has access to the Shopping Cart, Product, Order, and Payment Info data entities.   
5. The customer has valid payment information stored in the Payment Info data entity.   
6. The Plugin API is available for payment processing and inventory updates.   
  
Postconditions:   
1. A new Order is created and saved in the Order data entity.   
2. The selected products are removed from the Shopping Cart.   
3. The inventory quantity for the purchased products is updated.   
4. The system logs the checkout event.   
5. The Plugin API processes the payment and confirms transaction success.   
6. The customer receives a confirmation message for the completed purchase.   
  
Main Flow:   
1. The customer navigates to the "Checkout" section.   
2. The system displays the items in the Shopping Cart along with the total price.   
3. The customer selects a payment method from their Payment Info data.   
4. The system validates the selected payment information and confirms its availability.   
5. The system calls the Plugin API to process the payment.   
6. The Plugin API confirms the payment is successful.   
7. The system creates a new Order in the Order data entity, including the purchased products, total price, and payment method.   
8. The system updates the inventory quantities in the Product data entity.   
9. The system removes the purchased items from the Shopping Cart data entity.   
10. The system logs the checkout and payment processing event.   
11. The system displays a confirmation message to the customer with the order details.   
12. The customer is redirected to the order confirmation page.   
  
Alternative Flow:   
1. If the customer has no payment methods saved, the system redirects them to add payment information before proceeding.   
2. If the Plugin API returns an error (e.g., payment declined), the system displays an error message and prevents the order from being created.   
3. If the Plugin API is unavailable, the system displays a system error message and logs the issue for the Administrator to review.   
4. If the inventory update fails (e.g., product quantity is insufficient), the system displays an error message and prevents the checkout process.   
5. If the system fails to create the Order or update the Shopping Cart, it displays an error message and logs the failure for troubleshooting.  
  
Use Case Name: Confirm Order Placement   
Use Case ID: UC-22   
Actors: Customer, Shopping Cart, Order, Payment Info, Plugin API   
Preconditions:   
1. The system is accessible.   
2. The customer is authenticated and logged in.   
3. The Shopping Cart contains at least one item.   
4. The customer has valid payment information stored in the Payment Info data entity.   
5. The Plugin API is available for payment processing and inventory validation.   
  
Postconditions:   
1. The order is confirmed and saved in the Order data entity.   
2. The Shopping Cart is cleared of the purchased items.   
3. The inventory for the purchased products is updated.   
4. The system logs the order confirmation event.   
5. The Plugin API confirms the successful processing of the payment and inventory update.   
6. The customer receives a confirmation message for the order.   
  
Main Flow:   
1. The customer initiates the checkout process.   
2. The system verifies the items in the Shopping Cart and their availability via the Plugin API.   
3. The system confirms that all items are available and in stock.   
4. The customer selects a valid payment method from their Payment Info data.   
5. The system processes the payment through the Plugin API.   
6. The Plugin API confirms the payment is approved.   
7. The system creates a new Order in the Order data entity with the purchased items and payment details.   
8. The system updates the inventory quantities of the purchased products in the Product data entity.   
9. The system clears the Shopping Cart.   
10. The system displays a confirmation message to the customer with the order summary.   
11. The customer is redirected to the order confirmation page.   
  
Alternative Flow:   
1. If an item in the Shopping Cart is no longer available, the system displays a warning and prompts the customer to remove or replace the item.   
2. If the Plugin API is unavailable during inventory or payment validation, the system displays an error and logs the issue for the Administrator.   
3. If the payment is declined by the Plugin API, the system displays an error message and does not proceed with the order creation.   
4. If the system fails to create the Order or update the inventory, it displays an error message and logs the failure for review.  
  
Use Case Name: Receive Order Confirmation Email   
Use Case ID: UC-23   
Actors: Customer, Order, Plugin API   
Preconditions:   
1. The system is accessible.   
2. The customer has completed the checkout process and the order is saved in the Order data entity.   
3. The customer has a valid and verified email address.   
4. The Plugin API is available for email sending and tracking.   
  
Postconditions:   
1. The customer receives an order confirmation email.   
2. The Plugin API logs the email delivery attempt.   
3. The system logs the confirmation email event.   
4. If the Plugin API fails, the system logs the failure for the Administrator to review.   
  
Main Flow:   
1. The system generates an order confirmation message after the checkout is completed.   
2. The system retrieves the customer’s email address from the Customer data entity.   
3. The system prepares the email content, including order details (order ID, items, total amount, delivery status).   
4. The system calls the Plugin API to send the confirmation email to the customer.   
5. The Plugin API confirms the email was sent successfully.   
6. The system logs the successful email delivery event.   
7. The customer receives the confirmation email and can review their order.   
  
Alternative Flow:   
1. If the customer’s email address is invalid or unverified, the system displays a warning and logs the issue for the Administrator.   
2. If the Plugin API is unavailable, the system displays an error message and logs the issue for troubleshooting.   
3. If the Plugin API returns an error (e.g., failed to send the email), the system displays an error message to the customer and logs the failure.   
4. If the system fails to prepare or retrieve the email content, it displays an error message and logs the failure for review.  
  
Use Case Name: Process Payment   
Use Case ID: UC-24   
Actors: Customer, Shopping Cart, Order, Payment Info, Plugin API   
Preconditions:   
1. The system is accessible.   
2. The customer is authenticated and logged in.   
3. The customer's Shopping Cart contains at least one item.   
4. The system has access to the Order, Shopping Cart, and Payment Info data entities.   
5. The customer has valid payment information stored in the Payment Info data entity.   
6. The Plugin API is available for payment processing and transaction confirmation.   
  
Postconditions:   
1. The payment is successfully processed and confirmed.   
2. A new Order is created and saved in the Order data entity.   
3. The purchased items are removed from the Shopping Cart data entity.   
4. The inventory for the purchased products is updated in the Product data entity.   
5. The system logs the payment and order creation event.   
6. The Plugin API confirms the transaction and logs the event.   
7. The customer receives a confirmation message for the completed payment and order.   
  
Main Flow:   
1. The customer proceeds to the checkout page.   
2. The system displays the items in the Shopping Cart along with the total amount.   
3. The system verifies the customer's selected payment method from the Payment Info data entity.   
4. The system calls the Plugin API to initiate the payment process with the selected method.   
5. The Plugin API processes the payment and returns a confirmation of success.   
6. The system creates a new Order in the Order data entity with the purchased items and payment details.   
7. The system reduces the inventory quantity of the purchased products in the Product data entity.   
8. The system clears the Shopping Cart of the purchased items.   
9. The system logs the payment and order processing event.   
10. The system displays a confirmation message to the customer with the order summary.   
11. The customer is redirected to the order confirmation page.   
  
Alternative Flow:   
1. If the customer does not have a valid payment method saved, the system redirects them to the "Add Payment Information" use case.   
2. If the Plugin API returns a payment error (e.g., declined transaction), the system displays an error message and does not proceed with the order creation.   
3. If the Plugin API is unavailable, the system displays a system error message and logs the issue for the Administrator to review.   
4. If the inventory update fails (e.g., insufficient stock), the system displays an error message and halts the payment and order process.   
5. If the system fails to create the Order or update the Shopping Cart, it displays an error message and logs the failure for troubleshooting.  
  
Use Case Name: View Order Status   
Use Case ID: UC-25   
Actors: Customer, Administrator, Order, Plugin API   
Preconditions:   
1. The system is accessible.   
2. The customer is authenticated and logged in.   
3. The system has access to the Order data entity.   
4. The order to be viewed exists in the Order data entity.   
5. The Plugin API is available for status updates and notifications.   
  
Postconditions:   
1. The customer views the current status of their selected order.   
2. The system logs the access to the order status.   
3. If the Plugin API is used for status updates, the system logs the event.   
4. The Administrator is notified if there is an unauthorized access attempt.   
  
Main Flow:   
1. The customer navigates to the "Order Status" section in their account.   
2. The system retrieves a list of the customer's orders from the Order data entity.   
3. The customer selects a specific order to view its status.   
4. The system calls the Plugin API to fetch the latest status of the selected order.   
5. The Plugin API returns the current status (e.g., Processing, Shipped, Delivered, Cancelled).   
6. The system displays the order status and relevant details (e.g., order date, items, total price).   
7. The system logs the customer's access to the order status.   
8. The customer is returned to the order list or account dashboard.   
  
Alternative Flow:   
1. If the customer has no orders, the system displays a message indicating no orders are available.   
2. If the Plugin API is unavailable, the system displays a system error message and logs the issue for the Administrator.   
3. If the system fails to retrieve the order data, it displays an error message and logs the failure.   
4. If the customer attempts to access another user's order status, the system denies the request and logs the unauthorized access.  
  
Use Case Name: Cancel Order   
Use Case ID: UC-26   
Actors: Customer, Administrator, Order, Plugin API   
Preconditions:   
1. The system is accessible.   
2. The customer is authenticated and logged in.   
3. The customer has an existing order in the Order data entity.   
4. The Plugin API is available for order status updates and notifications.   
  
Postconditions:   
1. The selected order is successfully canceled.   
2. The system updates the order status in the Order data entity.   
3. The system logs the cancellation event.   
4. The Plugin API is notified of the cancellation and confirms the update.   
5. The Administrator is alerted if the cancellation involves specific conditions or issues.   
  
Main Flow:   
1. The customer navigates to the "Order Status" or "My Orders" section.   
2. The system displays a list of the customer's active orders.   
3. The customer selects an order and chooses the "Cancel Order" option.   
4. The system prompts the customer to confirm the cancellation.   
5. The customer confirms the cancellation.   
6. The system validates that the order is eligible for cancellation (e.g., not yet shipped).   
7. The system updates the order status in the Order data entity to "Canceled."   
8. The system logs the cancellation event.   
9. The system calls the Plugin API to notify external systems of the cancellation.   
10. The Plugin API confirms the status update.   
11. The system displays a confirmation message to the customer.   
12. The customer is redirected to the order list or account dashboard.   
  
Alternative Flow:   
1. If the customer cancels the confirmation request, the system returns to the order view without making changes.   
2. If the order is not eligible for cancellation (e.g., already shipped), the system displays a warning and prevents the cancellation.   
3. If the Plugin API is unavailable, the system displays a system error message and logs the issue for the Administrator to review.   
4. If the system fails to update the Order data entity, it displays an error message and logs the failure for troubleshooting.  
  
Use Case Name: Install Plugin   
Use Case ID: UC-27   
  
Actors: Administrator, Plugin API   
  
Preconditions:   
1. The system is accessible.   
2. The administrator is authenticated and logged in.   
3. The Plugin API is available for integration and validation.   
4. The plugin to be installed is available and compatible with the system.   
  
Postconditions:   
1. The plugin is successfully installed and integrated into the system.   
2. The system logs the plugin installation event.   
3. The Plugin API confirms the successful integration.   
4. The administrator receives a confirmation message.   
  
Main Flow:   
1. The administrator navigates to the "Plugin Management" section.   
2. The system displays a list of available plugins for installation.   
3. The administrator selects a plugin and clicks the "Install" button.   
4. The system validates the plugin compatibility and configuration requirements.   
5. The system calls the Plugin API to initiate the installation process.   
6. The Plugin API confirms the plugin is ready for installation.   
7. The system installs the plugin and updates the system configuration.   
8. The system logs the installation event.   
9. The system displays a confirmation message that the plugin was installed successfully.   
10. The administrator is redirected to the plugin management page.   
  
Alternative Flow:   
1. If the selected plugin is incompatible with the current system, the system displays an error message and prevents installation.   
2. If the Plugin API returns an error during installation, the system displays the error and logs the failure for review.   
3. If the Plugin API is unavailable, the system displays a system error message and logs the issue for the Administrator to review.   
4. If the system fails to update the configuration after installation, it displays an error message and logs the failure for troubleshooting.  
  
Use Case Name: Configure Plugin Settings   
Use Case ID: UC-28   
Actors: Administrator, Plugin API   
Preconditions:   
1. The system is accessible.   
2. The administrator is authenticated and logged in.   
3. The Plugin API is available for configuration and validation.   
4. The plugin to be configured is already installed in the system.   
  
Postconditions:   
1. The plugin settings are updated or configured successfully.   
2. The system logs the configuration event.   
3. The Plugin API confirms the successful update of settings.   
4. The administrator receives a confirmation message.   
  
Main Flow:   
1. The administrator navigates to the "Plugin Management" section.   
2. The system displays a list of installed plugins along with their current configuration status.   
3. The administrator selects a plugin and chooses the "Configure" option.   
4. The system retrieves the current settings of the selected plugin from the Plugin API.   
5. The system displays the plugin configuration form with the current settings.   
6. The administrator modifies the plugin settings as needed (e.g., API keys, timeout limits, enabled/disabled features).   
7. The system validates the new configuration data (e.g., correct format, required fields are filled).   
8. The system calls the Plugin API to update the plugin settings.   
9. The Plugin API confirms the updated settings.   
10. The system logs the configuration change and displays a confirmation message to the administrator.   
  
Alternative Flow:   
1. If the administrator enters invalid or incomplete configuration data, the system displays an error message and prompts for correction.   
2. If the Plugin API returns an error during configuration update, the system displays the error and prevents the changes from being saved.   
3. If the Plugin API is unavailable, the system displays a system error message and logs the issue for the Administrator to review.   
4. If the system fails to save the configuration or log the event, it displays an error message and logs the failure for troubleshooting.  
  
Use Case Name: Access Plugin Documentation   
Use Case ID: UC-29   
Actors: Administrator, Plugin API   
Preconditions:   
1. The system is accessible.   
2. The administrator is authenticated and logged in.   
3. The plugin to be accessed is installed and available in the system.   
4. The Plugin API is available for retrieving documentation.   
  
Postconditions:   
1. The administrator views the plugin’s documentation.   
2. The system logs the access to the plugin documentation.   
3. The Plugin API confirms the successful retrieval of the documentation.   
4. If the Plugin API is unavailable, the system logs the issue for the Administrator.   
  
Main Flow:   
1. The administrator navigates to the "Plugin Management" section.   
2. The system displays a list of installed plugins.   
3. The administrator selects a plugin and chooses the "View Documentation" option.   
4. The system calls the Plugin API to retrieve the documentation for the selected plugin.   
5. The Plugin API returns the documentation content.   
6. The system displays the documentation in a readable format.   
7. The system logs the access event.   
8. The administrator is returned to the plugin management page or can navigate further within the documentation.   
  
Alternative Flow:   
1. If the selected plugin does not have documentation available, the system displays a message indicating that no documentation is found.   
2. If the Plugin API is unavailable, the system displays a system error message and logs the issue for the Administrator to review.   
3. If the system fails to retrieve or display the documentation, it shows an error message and logs the failure for troubleshooting.  
  
Use Case Name: Uninstall Plugin   
Use Case ID: UC-28   
Actors: Administrator, Plugin API   
Preconditions:   
1. The system is accessible.   
2. The administrator is authenticated and logged in.   
3. The Plugin API is available for integration and validation.   
4. The plugin to be uninstalled is already installed in the system.   
  
Postconditions:   
1. The plugin is successfully uninstalled from the system.   
2. The system logs the uninstallation event.   
3. The Plugin API confirms the successful removal of the plugin.   
4. The administrator receives a confirmation message.   
  
Main Flow:   
1. The administrator navigates to the "Plugin Management" section.   
2. The system displays a list of installed plugins.   
3. The administrator selects a plugin and clicks the "Uninstall" button.   
4. The system validates the plugin status and confirms it is installed.   
5. The system calls the Plugin API to initiate the uninstallation process.   
6. The Plugin API confirms the plugin can be uninstalled.   
7. The system uninstalls the plugin and updates the system configuration.   
8. The system logs the uninstallation event.   
9. The system displays a confirmation message that the plugin was uninstalled successfully.   
10. The administrator is redirected to the plugin management page.   
  
Alternative Flow:   
1. If the selected plugin is in use or dependencies exist, the system displays a warning message and prevents uninstallation.   
2. If the Plugin API returns an error during uninstallation, the system displays the error and logs the failure for review.   
3. If the Plugin API is unavailable, the system displays a system error message and logs the issue for the Administrator to review.   
4. If the system fails to update the configuration after uninstallation, it displays an error message and logs the failure for troubleshooting.  
  
Use Case Name: Convert Shopping Cart to Order   
Use Case ID: UC-30   
Actors: Customer, Shopping Cart, Order, Payment Info, Plugin API   
  
Preconditions:   
1. The system is accessible.   
2. The customer is authenticated and logged in.   
3. The customer's Shopping Cart contains at least one item.   
4. The system has access to the Shopping Cart, Product, Order, and Payment Info data entities.   
5. The customer has at least one valid payment method stored in the Payment Info data entity.   
6. The Plugin API is available for payment processing and inventory updates.   
  
Postconditions:   
1. The items in the Shopping Cart are successfully converted into an Order.   
2. The Shopping Cart is cleared of the converted items.   
3. The inventory for the purchased products is updated in the Product data entity.   
4. The selected payment method is processed through the Plugin API.   
5. The system logs the conversion and order creation event.   
6. The Plugin API confirms the successful transaction and inventory update.   
7. The customer receives a confirmation message for the new order.   
  
Main Flow:   
1. The customer navigates to the "Checkout" section.   
2. The system displays the contents of the Shopping Cart and the total amount.   
3. The customer selects a valid payment method from their Payment Info data.   
4. The system validates the payment method and confirms its availability.   
5. The system calls the Plugin API to process the payment.   
6. The Plugin API confirms the payment is successful.   
7. The system creates a new Order in the Order data entity, including the items from the Shopping Cart, total price, and payment details.   
8. The system reduces the inventory quantity of the purchased products in the Product data entity.   
9. The system removes the items from the Shopping Cart data entity.   
10. The system logs the conversion of the Shopping Cart to an Order and the payment processing event.   
11. The system displays a confirmation message to the customer with the order summary.   
12. The customer is redirected to the order confirmation page.   
  
Alternative Flow:   
1. If the customer has no valid payment method saved, the system redirects them to the "Add Payment Information" use case.   
2. If the Plugin API returns an error during payment processing (e.g., declined transaction), the system displays an error message and does not proceed with the order creation.   
3. If the Plugin API is unavailable, the system displays a system error message and logs the issue for the Administrator to review.   
4. If the inventory update fails (e.g., product quantity is insufficient), the system displays an error message and halts the conversion process.   
5. If the system fails to create the Order or clear the Shopping Cart, it displays an error message and logs the failure for troubleshooting.  
  
Use Case Name: Manage Order Detail   
Use Case ID: UC-31   
Actors: Administrator, Customer, Order, Order Detail, Product, Plugin API   
Preconditions:   
1. The system is accessible.   
2. The user (Customer or Administrator) is authenticated and logged in.   
3. The system has access to the Order and Order Detail data entities.   
4. The order to be managed exists in the Order data entity.   
5. The Plugin API is available for validation and external system synchronization.   
  
Postconditions:   
1. The order detail is updated, added, or removed based on the action performed.   
2. The system logs the modification of order details.   
3. The Plugin API confirms the synchronization of updated order details.   
4. If unauthorized access is attempted, the system logs it and notifies the Administrator.   
  
Main Flow:   
1. The user navigates to the "Order Details" section of a specific order.   
2. The system retrieves the order and its associated Order Detail records from the Order and Order Detail data entities.   
3. The system displays the order details, including product names, quantities, prices, and total.   
4. The user selects an action to manage the details (e.g., update quantity, remove product, add a new product to the order).   
5. If updating, the user modifies the quantity or price of a product in the order.   
6. If removing, the system confirms the removal and deletes the selected Order Detail entry.   
7. If adding, the user selects a product and specifies the quantity, and the system adds a new Order Detail entry.   
8. The system validates the updated or new order detail data (e.g., valid product ID, non-negative quantity).   
9. The system updates the Order Detail data entity with the new or modified information.   
10. The system recalculates the order total based on the updated details.   
11. The system logs the modification of the order details.   
12. The system calls the Plugin API to synchronize the updated order details with external systems (e.g., inventory or accounting).   
13. The Plugin API confirms the synchronization.   
14. The system displays a confirmation message to the user.   
15. The user is redirected to the order summary or back to the order list.   
  
Alternative Flow:   
1. If the user enters invalid or incomplete data during modification or addition, the system displays an error message and prompts for correction.   
2. If the Plugin API is unavailable during synchronization, the system displays a system error message and logs the issue for the Administrator to review.   
3. If the system fails to update the Order Detail data entity or recalculate the order total, it displays an error message and logs the failure for troubleshooting.   
4. If the user (Customer) attempts to modify an order detail that is not editable (e.g., order has been shipped), the system displays a warning and prevents the modification.   
5. If the user (Customer) tries to access an order that does not belong to them, the system denies the request and logs the unauthorized access.  
  
Use Case Name: Manage Cart Item   
Use Case ID: UC-32   
Actors: Customer, Shopping Cart, Product, Plugin API   
Preconditions:   
1. The system is accessible.   
2. The customer is authenticated and logged in.   
3. The system has access to the Shopping Cart and Product data entities.   
4. The customer's Shopping Cart contains at least one item.   
5. The Plugin API is available for inventory and availability checks.   
  
Postconditions:   
1. The customer's Shopping Cart is updated with modified cart items (e.g., quantity adjustments, item removal, or item replacement).   
2. The system logs the cart modification event.   
3. The Plugin API confirms the updated inventory status of affected products.   
4. If the Plugin API is unavailable, the system logs the issue for the Administrator to review.   
  
Main Flow:   
1. The customer navigates to the "Shopping Cart" section.   
2. The system retrieves and displays the current list of cart items from the Shopping Cart data entity.   
3. The customer selects an item and chooses an action (e.g., update quantity, remove item, or replace item).   
4. If updating quantity, the customer enters the new quantity value.   
5. If replacing an item, the customer selects a new product and specifies the quantity.   
6. The system validates the requested action and input data (e.g., quantity is within inventory limits, product is available for replacement).   
7. The system updates the Shopping Cart data entity with the changes (e.g., adjusts quantity, removes the item, or replaces it with a new product).   
8. The system calls the Plugin API to verify the updated inventory status for the affected product(s).   
9. The Plugin API confirms the inventory status for the product(s).   
10. The system recalculates the total price of the cart and updates the display.   
11. The system logs the modification of the Shopping Cart.   
12. The customer is returned to the cart summary or redirected to the product page for replacement.   
  
Alternative Flow:   
1. If the customer enters a quantity higher than what is available in the inventory, the system displays an error message and prevents the update.   
2. If the Plugin API is unavailable during inventory verification, the system displays a system error message and logs the issue for the Administrator to review.   
3. If the customer cancels the modification request, the system returns to the cart summary without making changes.   
4. If the system fails to update the Shopping Cart data entity, it displays an error message and logs the failure for troubleshooting.  
  
Use Case Name: Manage Administrator   
Use Case ID: UC-33   
Actors: Administrator   
Preconditions:   
1. The system is accessible.   
2. The user is an authenticated Administrator.   
3. The system has access to the Administrator data entity.   
4. The user has the necessary permissions to manage other Administrator accounts.   
  
Postconditions:   
1. The Administrator account is successfully created, updated, or deleted.   
2. The system logs the action performed on the Administrator account.   
3. The changes are reflected in the Administrator data entity.   
4. If an account is created or updated, the system sends a notification to the affected Administrator.   
  
Main Flow:   
1. The Administrator navigates to the "Manage Administrators" section.   
2. The system displays a list of existing Administrator accounts.   
3. The Administrator selects an action: "Create New Administrator," "Edit Administrator," or "Delete Administrator."   
4. If creating, the Administrator enters the new user's information (name, email, password, role, and permissions).   
5. The system validates the email format and password strength for the new Administrator.   
6. The system saves the new Administrator account in the Administrator data entity.   
7. The system sends a confirmation message to the new Administrator's email.   
8. If editing, the Administrator selects an existing account and modifies the required information (e.g., role, permissions, or status).   
9. The system validates the updated information and applies the changes.   
10. The system updates the Administrator data entity with the new information.   
11. The system sends a notification to the affected Administrator.   
12. If deleting, the Administrator selects an existing account and confirms the deletion.   
13. The system removes the selected Administrator account from the Administrator data entity.   
14. The system logs the deletion and updates the list of Administrators.   
15. The system displays a confirmation message for the action performed.   
16. The Administrator is redirected to the Administrator management page.   
  
Alternative Flow:   
1. If the email or password format is invalid during creation or editing, the system displays an error message and prompts for correction.   
2. If the selected Administrator account cannot be deleted (e.g., due to being the last active administrator), the system displays a warning and prevents the deletion.   
3. If the system fails to save or update an Administrator account, it displays an error message and logs the failure for review.   
4. If the system fails to send the confirmation email to the new or updated Administrator, it logs the failure and displays a warning message.   
5. If the system fails to log the action, it logs the failure for troubleshooting.  
  
Use Case Name: Manage Category   
Use Case ID: UC-34   
Actors: Administrator, Category, Plugin API   
Preconditions:   
1. The system is accessible.   
2. The administrator is authenticated and logged in.   
3. The system has access to the Category data entity.   
4. The Plugin API is available for validation and external system synchronization.   
  
Postconditions:   
1. The category is successfully created, updated, or deleted as per the administrator’s request.   
2. The Category data entity is updated accordingly.   
3. The system logs the action performed on the category.   
4. The Plugin API confirms the synchronization of the category information with external systems.   
5. If the Plugin API is unavailable, the system logs the issue for the Administrator to review.   
  
Main Flow:   
1. The administrator navigates to the "Category Management" section.   
2. The system displays a list of existing categories along with their details (name, description, product count).   
3. The administrator selects an action: "Create New Category," "Edit Category," or "Delete Category."   
4. If creating, the administrator enters the category name and description.   
5. The system validates the input (e.g., name is not empty, format is correct).   
6. The system calls the Plugin API to check for duplicate category names or to confirm compatibility with external systems.   
7. The Plugin API confirms the category name is available.   
8. The system saves the new category to the Category data entity.   
9. The system updates the product associations if necessary.   
10. If editing, the administrator selects an existing category and modifies its name or description.   
11. The system validates the updated information.   
12. The system calls the Plugin API to confirm the changes are acceptable for external systems.   
13. The system updates the Category data entity with the new information.   
14. If deleting, the administrator selects a category and confirms the deletion.   
15. The system checks if the category is associated with any active products.   
16. If no active products are linked, the system proceeds to remove the category from the Category data entity.   
17. The system calls the Plugin API to synchronize the deletion with external systems.   
18. The Plugin API confirms the successful removal.   
19. The system logs the category management event.   
20. The system displays a confirmation message for the action performed.   
21. The administrator is redirected to the category management page.   
  
Alternative Flow:   
1. If the administrator enters an empty or invalid category name, the system displays an error message and prompts for correction.   
2. If the Plugin API returns an error (e.g., duplicate category name or failed validation), the system prevents the action and displays an error message.   
3. If the Plugin API is unavailable during the synchronization, the system displays a system error message and logs the issue for the Administrator to review.   
4. If the category is linked to active products, the system prevents deletion and displays a warning message.   
5. If the system fails to update or delete the Category data entity, it displays an error message and logs the failure for troubleshooting.  
  
Use Case Name: Manage Shopping Cart   
Use Case ID: UC-35   
Actors: Customer, Shopping Cart, Product, Plugin API   
Preconditions:   
1. The system is accessible.   
2. The customer is authenticated and logged in.   
3. The system has access to the Shopping Cart and Product data entities.   
4. The Plugin API is available for inventory and availability checks.   
5. The customer has at least one item in their Shopping Cart.   
  
Postconditions:   
1. The Shopping Cart is updated based on the customer's action (e.g., clear cart, apply discount, or view cart summary).   
2. The system logs the Shopping Cart management event.   
3. The Plugin API confirms the updated inventory status if applicable.   
4. If the Plugin API is unavailable, the system logs the issue for the Administrator to review.   
  
Main Flow:   
1. The customer navigates to the "Shopping Cart" section.   
2. The system displays the cart summary, including product details, quantities, and total price.   
3. The customer selects a management action (e.g., clear cart, apply discount, or adjust quantity for multiple items).   
4. If clearing the cart, the system removes all items from the Shopping Cart data entity.   
5. If applying a discount, the system checks for applicable promotions and adjusts the total price accordingly.   
6. The system calls the Plugin API to verify the updated inventory status for any modified or removed items.   
7. The Plugin API returns confirmation of the inventory status.   
8. The system updates the cart display with the new state and recalculates the total price.   
9. The system logs the Shopping Cart management event.   
10. The customer is returned to the cart summary or redirected to another section (e.g., product catalog or checkout).   
  
Alternative Flow:   
1. If the customer selects a discount that is no longer valid or unavailable, the system displays a message and does not apply the discount.   
2. If the Plugin API is unavailable during inventory verification, the system displays a system error and logs the issue for the Administrator.   
3. If the customer cancels the management action, the system returns to the cart summary without making changes.   
4. If the system fails to update the Shopping Cart data entity, it displays an error message and logs the failure for troubleshooting.  
  
Use Case Name: Manage Order   
Use Case ID: UC-36   
Actors: Administrator, Customer, Order, Product, Plugin API   
Preconditions:   
1. The system is accessible.   
2. The user (Customer or Administrator) is authenticated and logged in.   
3. The system has access to the Order and Product data entities.   
4. The order to be managed exists in the Order data entity.   
5. The Plugin API is available for status updates and notifications.   
  
Postconditions:   
1. The order is updated, modified, or canceled based on the action performed.   
2. The system logs the order management event.   
3. The Plugin API confirms the synchronization of the order status update with external systems.   
4. If unauthorized access is detected, the system logs the event and notifies the Administrator.   
  
Main Flow:   
1. The user navigates to the "Order Management" section.   
2. The system retrieves a list of orders associated with the user (Customer views their own orders; Administrator views all orders).   
3. The user selects an order for further action (e.g., update status, cancel, or modify).   
4. If the user is an Administrator and selects to update the status, they choose the new status (e.g., Processing, Shipped, Delivered, Cancelled).   
5. The system validates the selected action and the new status (if applicable).   
6. The system updates the Order data entity with the new status or modification.   
7. The system calls the Plugin API to synchronize the updated order status with external systems (e.g., logistics or inventory).   
8. The Plugin API confirms the synchronization.   
9. The system logs the order management event.   
10. The system displays a confirmation message to the user indicating the order has been successfully managed.   
11. The user is redirected to the order list or account dashboard.   
  
Alternative Flow:   
1. If the user is a Customer and attempts to modify an order that is not eligible for changes (e.g., already shipped), the system displays a warning message and prevents the modification.   
2. If the selected status is invalid or inconsistent with the order lifecycle, the system displays an error message and prompts for a valid status.   
3. If the Plugin API is unavailable during synchronization, the system displays a system error message and logs the issue for the Administrator to review.   
4. If the system fails to update the Order data entity, it displays an error message and logs the failure for troubleshooting.   
5. If the user attempts to manage an order that does not belong to them, the system denies the request and logs the unauthorized access.  
  
Use Case Name: Apply Discount Code   
Use Case ID: UC-37   
Actors: Customer, Shopping Cart, Plugin API   
Preconditions:   
1. The system is accessible.   
2. The customer is authenticated and logged in.   
3. The system has access to the Shopping Cart data entity.   
4. The customer's Shopping Cart contains at least one item.   
5. The Plugin API is available for discount validation and application.   
  
Postconditions:   
1. The discount code is successfully applied to the Shopping Cart.   
2. The system recalculates the total price based on the discount.   
3. The system logs the application of the discount code.   
4. If the Plugin API is unavailable, the system logs the issue for the Administrator to review.   
  
Main Flow:   
1. The customer navigates to the "Shopping Cart" section.   
2. The system displays the cart summary, including product details, quantities, and total price.   
3. The customer enters a discount code in the designated input field.   
4. The system calls the Plugin API to validate the discount code.   
5. The Plugin API confirms the discount is valid and applicable to the current cart items.   
6. The system applies the discount to the Shopping Cart, adjusting the total price accordingly.   
7. The system updates the cart display to reflect the new total price.   
8. The system logs the application of the discount code.   
9. The customer is returned to the cart summary or redirected to the checkout page.   
  
Alternative Flow:   
1. If the discount code is invalid or expired, the system displays an error message and prevents the discount from being applied.   
2. If the discount code is not applicable to the current cart items, the system displays a message and does not apply the discount.   
3. If the Plugin API is unavailable during validation, the system displays a system error message and logs the issue for the Administrator to review.   
4. If the system fails to update the cart total or apply the discount, it displays an error message and logs the failure for troubleshooting.