UML Diagram Report

Clothing Shop

This project involves the development of an online clothing shop platform. The system is designed to provide functionalities for both customers and administrators. Customers can browse clothing items, filter, search, and make purchases, while administrators can manage products, users, and support tickets

1. **Time and Cost Estimation**
    - Analysis and Requirement Gathering
    - Designing UML Diagrams
    - UI/UX Design
    - Frontend Development
    - Backend Development
    - Testing and Debugging
    - Deployment
    - Maintenance and Updates
2. **UML Diagrams**
    - Use Case Diagram
    - Activity Diagram
    - Class Diagram
    - Object Diagram
    - Sequence Diagram
3. **Explanations of Diagrams**
    - Use Case Diagram
    - Activity Diagram
    - Class Diagram
    - Object Diagram
    - Sequence Diagram
4. **Conclusion**
    - Summary of Workflows
    - Future Improvements

Time and Cost Estimation for Designing and Developing the Online Clothing Shop

Below is a detailed breakdown of the estimated **time** and **cost** required to design and build the online clothing shop, including all the requested features and diagrams:

1. Analysis and Requirement Gathering

- **Tasks**:
  - Understanding the project scope and goals.
  - Writing functional and non-functional requirements.
  - Finalizing features (e.g., sign-up/login, product management, cart, payment integration).
  - Creating initial mockups and diagrams.
- **Time**: 1 week (40 hours).
- **Cost**: $500
  (Based on a business analyst rate of $50/hour).

2. Designing UML Diagrams

- **Tasks**:
  - **Use Case Diagram**: Designing scenarios for all actors (Customers, Admin, Bank).
  - **Class Diagram**: Creating relationships between entities (e.g., Customer, Product, Order).
  - **Activity Diagram**: Representing workflows like login, purchase, etc.
  - **Sequence Diagram**: Detailing interactions between components.
  - **Object Diagram**: Showing object instances during a sample runtime.
- **Time**: 1 week (40 hours).
- **Cost**: $600
  (Based on a software architect rate of $50/hour).

3. UI/UX Design

- **Tasks**:
  - Designing user-friendly interfaces for:
    - Home Page.
    - Product Listing Page with filters.
    - Cart and Payment Page.
    - Admin Dashboard for managing users, products, and tickets.
  - Creating responsive layouts for desktop and mobile devices.
- **Time**: 2 weeks (80 hours).
- **Cost**: $1,200
  (Based on a UI/UX designer rate of $40/hour).

4. Frontend Development

- **Tasks**:
  - Implementing the UI using **HTML, CSS, JavaScript** frameworks (e.g., React or Angular).

    - Integrating dynamic product filters, cart management, and real-time UI updates.
- **Time**: 3 weeks (120 hours).
- **Cost**: $2,400
  (Based on a frontend developer rate of $40/hour).

5. Backend Development

- **Tasks**:
    - Setting up the server and database:
        - Using **Django** or **Node.js** for backend logic.
        - Using **SQLite/MySQL** for the database.
    - Developing key functionalities:
        - Login/Sign-up system (with validations).
        - Product catalog and search filters.
        - Cart and order processing logic.
        - Payment gateway integration (e.g., Stripe or PayPal).
        - Admin panel for managing products, users, and support tickets.
    - Ensuring secure data handling and API development.
- **Time**: 4 weeks (160 hours).
- **Cost**: $4,800
  (Based on a backend developer rate of $50/hour).

6. Testing and Debugging

- **Tasks**:
    - Writing unit tests for backend and frontend components.
    - Functional testing of features (e.g., login, payment, admin panel).
    - Performance testing under different load conditions.
    - Fixing any bugs or issues found during testing.
- **Time**: 2 weeks (80 hours).
- **Cost**: $2,000
  (Based on a QA engineer rate of $25/hour).

7. Deployment

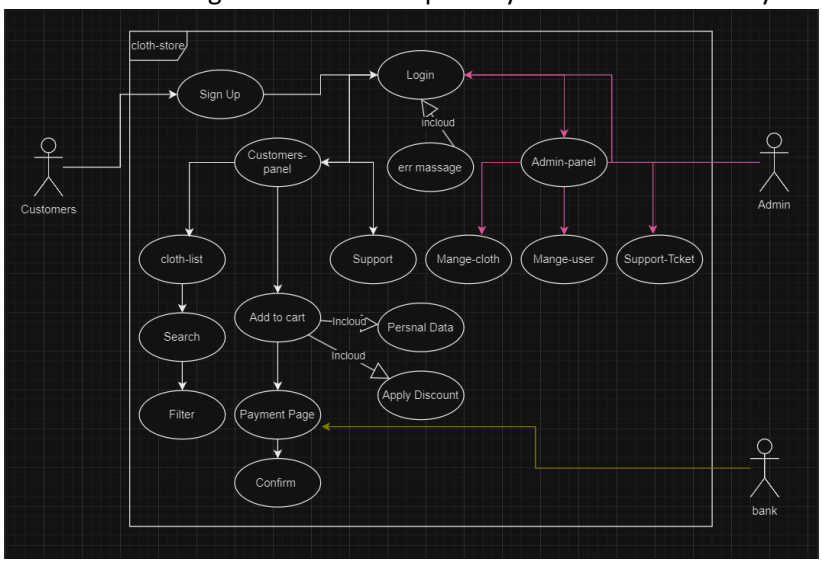
- **Tasks**:
    - Setting up hosting (e.g., AWS, Google Cloud, or Azure).
    - Deploying the application on a production server.
    - Configuring SSL certificates and domain settings.
    - Setting up monitoring and logging tools.
- **Time**: 1 week (40 hours).
- **Cost**: $1,000
  (Based on a DevOps engineer rate of $25/hour)

8. Maintenance and Updates

- **Tasks**:
  - Ongoing support for fixing bugs and adding new features.
  - Optimizing performance and updating libraries.
- **Time**: Monthly (20 hours).
- **Cost**: $500/month
  (Based on an average maintenance cost of $25/hour).

1. Use Case Diagram

The Use Case Diagram outlines the primary interactions in the system



Use Case Diagram Description for the Online Clothing Shop

The Use Case Diagram for the online clothing shop includes two main actors and their respective use cases. Below is a detailed explanation of the actors, use cases, and their interactions:

Actors:

1. **Customer:**
   - Represents regular users of the shop.
   - Key actions include signing up, logging in, browsing products, adding items to the cart, applying discounts, and making payments.
2. **Admin:**
   - Represents the administrator of the system with full control over managing users, products, and support tickets.
   - Key actions include managing clothes, managing users, and addressing support tickets.
3. **Bank:**
   - Represents the payment gateway that processes customer payments.

Use Cases:

1. **Sign Up:**
    - Allows customers to create a new account by providing personal details such as name, email, and password.
    - Ensures secure registration for accessing shop features.
2. **Login:**
    - Enables customers and administrators to log into the system to access their respective panels.
3. **Customers-Panel:**
    - Provides customers with access to their personal dashboard for managing orders and support tickets.
4. **Cloth-List:**
    - Displays a list of available clothing items for customers to browse.
    - Supports search and filter functionality for ease of navigation.
5. **Add to Cart:**
    - Allows customers to select items and add them to their shopping cart.
    - Ensures smooth integration with the checkout process.
6. **Apply Discount:**
    - Lets customers apply promotional codes or discounts during checkout.
7. **Payment Page:**
    - Handles the payment process, interacting with the bank system to confirm transactions.
8. **Support:**
    - Provides customers a way to submit support tickets and seek assistance for any issues.
9. **Admin-Panel:**
    - Grants administrators access to tools for managing products, users, and support tickets.
10. **Manage-Cloth:**
    - Allows the admin to add, update, or delete products in the shop's catalog.
11. **Manage-User:**
    - Enables the admin to manage customer accounts, including blocking or activating users.
12. **Support-Ticket:**
    - Lets the admin respond to customer queries and resolve their issues

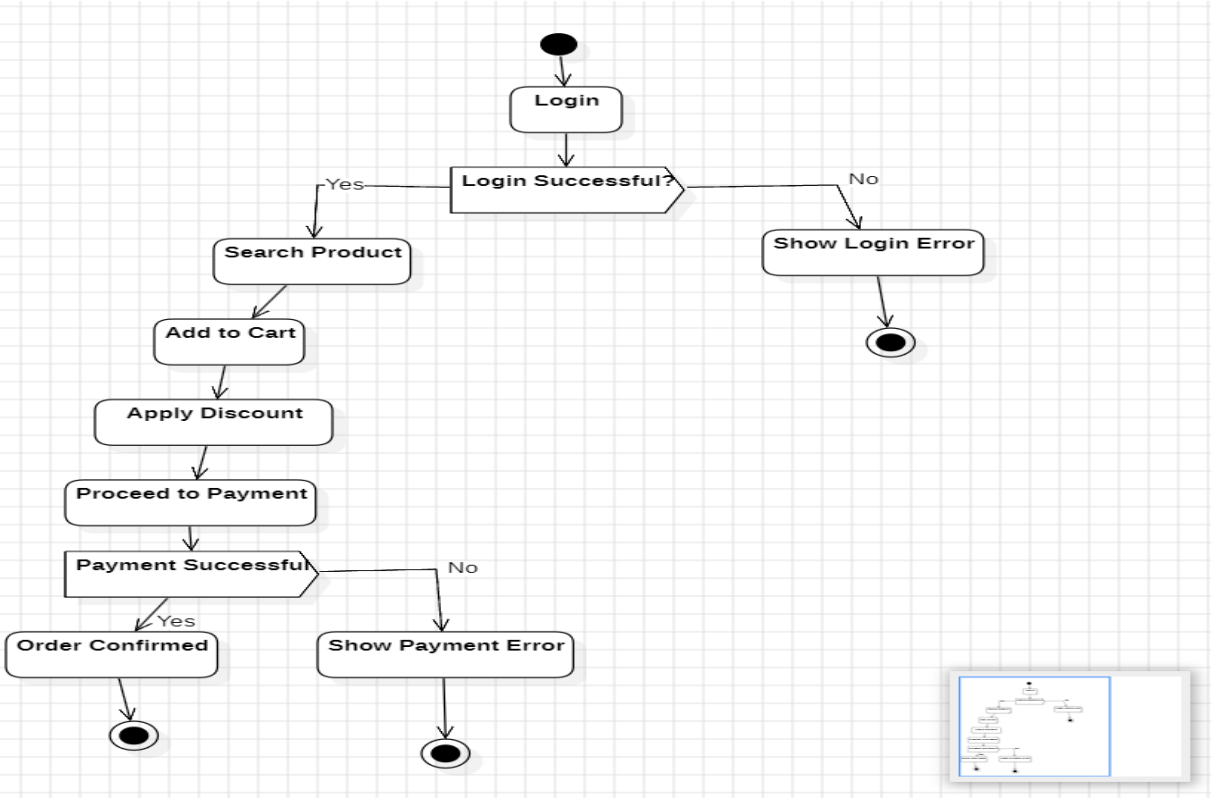
Relationships:

- **Include Relationships:**
    - The *Login* use case is included in the *Customers-Panel* and *Admin-Panel* as users need to log in first to access their respective dashboards.
    - *Add to Cart* includes *Apply Discount* and *Payment Page* for integrating discounts and initiating payments.
- **Extend Relationships:**

    - The *Error Message* use case extends *Login* to display an error if authentication fails.
- **Interaction with External Systems:**
    - The *Payment Page* interacts with the *Bank* to process payments securely.

This Use Case Diagram provides a clear overview of the system's core functionalities and how different actors interact with the system to achieve their goals.

3. Activity Diagram (Purchase Process)



The given **Activity Diagram** represents the **purchase process** for a customer in the online clothing shop. It highlights the steps and decision points a customer goes through while interacting with the system. Below is a detailed explanation:

Activities

1. **Login:**
    - The process begins with the customer logging into the system.
    - This is an essential step for accessing personalized features like browsing the catalog, adding items to the cart, and purchasing.
2. **Login Successful?**
    - A decision point to verify if the customer provided valid login credentials:
        - **Yes:** If the login is successful, the process continues to the next activity.

        - **No:** If the login fails, the system shows a login error, and the process ends here.
3. **Search Product:**
    - After successful login, the customer can search for the desired product(s) using the system's search functionality.
4. **Add to Cart:**
    - The customer selects products and adds them to the shopping cart for later purchase.
5. **Apply Discount (Optional):**
    - The customer may choose to apply a discount code or coupon to reduce the price of their purchase.
6. **Proceed to Payment:**
    - The customer initiates the payment process for the items in the cart.
7. **Payment Successful?**
    - A decision point to confirm if the payment was processed successfully:
        - **Yes:** If the payment is successful, the system confirms the order.
        - **No:** If the payment fails, the system shows a payment error, and the process ends.
8. **Order Confirmed:**
    - The final activity where the system confirms the order and provides feedback to the customer.
9. **Show Login/Payment Error:**
    - These activities handle unsuccessful attempts (login or payment) by displaying appropriate error messages to the customer.

Decision Points (Yes/No Paths)

1. **Login Successful?**
    - Ensures only authorized customers can proceed further.
    - Handles invalid login attempts gracefully.
2. **Payment Successful?**
    - Verifies if the transaction was completed securely.
    - Redirects the customer to the respective paths (success or error).

Relation to Use Case

This activity diagram aligns closely with the following **use cases**:

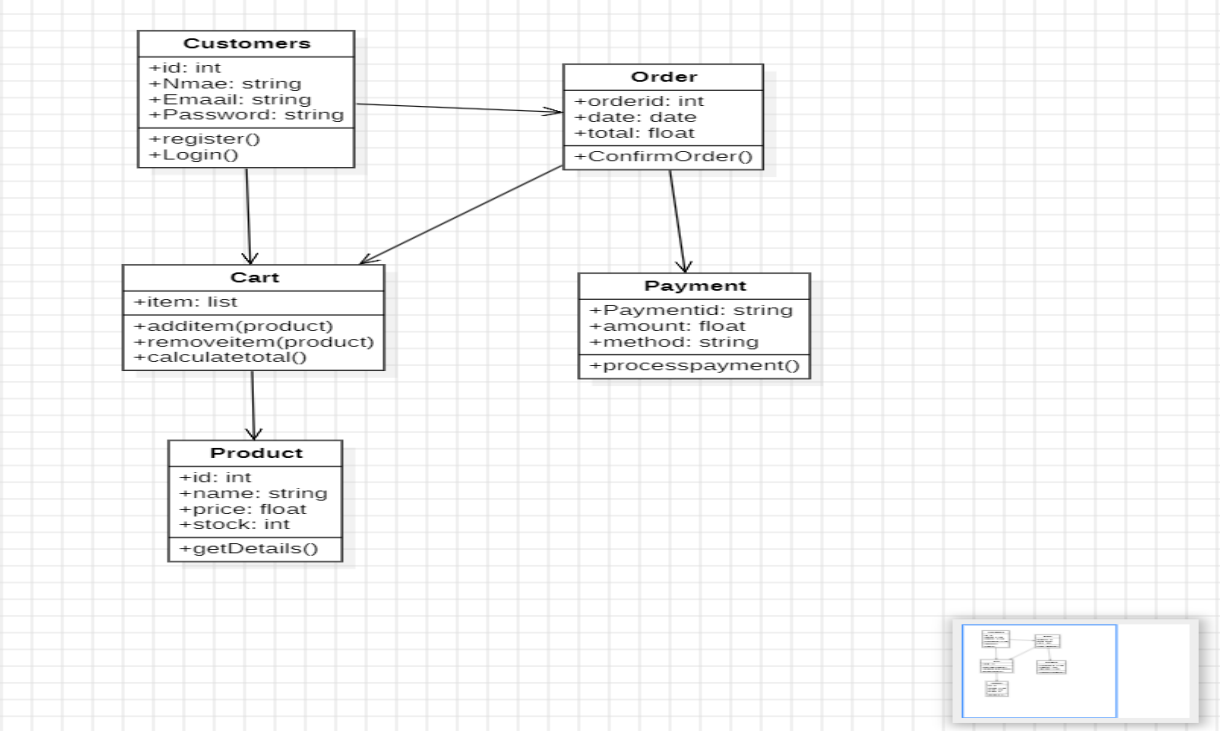
- **Login:** Represents the steps required for authentication.
- **Search Product:** Allows customers to find specific items using filters or keywords.
- **Add to Cart:** Facilitates adding selected products for checkout.
- **Apply Discount:** Optionally applies promotional codes.
- **Payment Page:** Processes the customer's payment securely through an integrated payment gateway.
- **Order Confirmation:** Notifies the customer that their order has been placed successfully.

Key Features

- **Error Handling:** The diagram clearly demonstrates how login and payment errors are managed to improve user experience.
- **Sequential Flow:** Activities follow a logical progression from login to order confirmation.
- **Optional Discount:** The inclusion of applying a discount code is well-highlighted as a non-mandatory step.

This activity diagram effectively visualizes the purchase workflow, providing a clear understanding of system functionality and user interactions.

4. Class Diagram



The **Class Diagram** showcases the structure of the online clothing shop by detailing the key system classes, their attributes, methods, and relationships. Here's an explanation of the diagram, connecting it to the **use cases** in the system:

Classes and Their Roles:

1. Customer Class:

- **Attributes:**
  - `customerID`: A unique identifier for each customer.
  - `name`, `email`, `username`, `password`: Information required for registration and login.
- **Methods:**
  - `signUp()`: Implements the **Sign Up** use case, allowing customers to register on the platform.

- `login()`: Implements the **Login** use case, validating user credentials.
- `searchProduct()`: Implements the **Search Product** use case, enabling product search.
- `addToCart()`: Implements the **Add to Cart** use case.
- `applyDiscount()`: Implements the **Apply Discount** use case.
- `proceedToPayment()`: Implements part of the **Payment Page** use case.

Relation to Use Cases:

- The **Customer** class is central to most use cases, as customers perform nearly all major actions, including browsing, cart management, and making payments.

2. Admin Class:

- **Attributes:**
  - `adminID`: A unique identifier for each administrator.
  - `username`, `password`: Used for authentication.
- **Methods:**
  - `manageUsers()`: Implements the **Manage User** use case, enabling administrators to update, delete, or deactivate customer accounts.
  - `manageClothes()`: Implements the **Manage Cloth** use case, allowing administrators to add, update, or remove products.
  - `respondToSupportTickets()`: Implements the **Support-Ticket** use case, enabling administrators to address customer inquiries.

Relation to Use Cases:

- The **Admin** class interacts directly with administrative use cases, providing tools to manage the system.

3. Product Class:

- **Attributes:**
  - `productID`: A unique identifier for each product.
  - `name`, `price`, `category`, `description`: Essential details about a product.
  - `quantity`: Tracks stock availability.
- **Methods:**
  - `filterByCategory()`: Implements part of the **Search Product** use case by allowing customers to narrow results.
  - `updateQuantity()`: Supports the **Manage Cloth** use case by letting admins manage inventory.

Relation to Use Cases:

- The **Product** class supports customers during browsing and searching and administrators during product management.

4. Cart Class:

- **Attributes:**
  - `cartID`: A unique identifier for the cart.
  - `products[]`: A collection of products added to the cart.
  - `totalPrice`: The total cost of items in the cart.
- **Methods:**
  - `addProduct()`: Implements the **Add to Cart** use case by allowing products to be added.
  - `removeProduct()`: Allows customers to remove items from their cart.
  - `calculateTotal()`: Ensures accurate billing for the **Payment Page** use case.

Relation to Use Cases:

- The **Cart** class directly handles the **Add to Cart** and checkout-related use cases, ensuring a smooth purchasing process.

5. Payment Class:

- **Attributes:**
  - `paymentID`: A unique identifier for each payment transaction.
  - `paymentStatus`: Tracks whether the payment was successful or failed.
  - `amount`: The total cost to be paid.
- **Methods:**
  - `processPayment()`: Implements the **Payment Page** use case by interacting with the bank system.
  - `applyDiscount()`: Calculates discounts during the checkout process.

Relation to Use Cases:

- The **Payment** class is essential for completing the **Payment Page** use case and processing transactions with the **Bank**.

6. SupportTicket Class:

- **Attributes:**
  - `ticketID`: A unique identifier for each support query.
  - `issueDescription`: Details of the customer's issue.
  - `status`: Tracks whether the issue is resolved or pending.
- **Methods:**
  - `createTicket()`: Implements the **Support** use case by allowing customers to submit their issues.
  - `updateStatus()`: Enables administrators to manage the **Support-Ticket** use case.

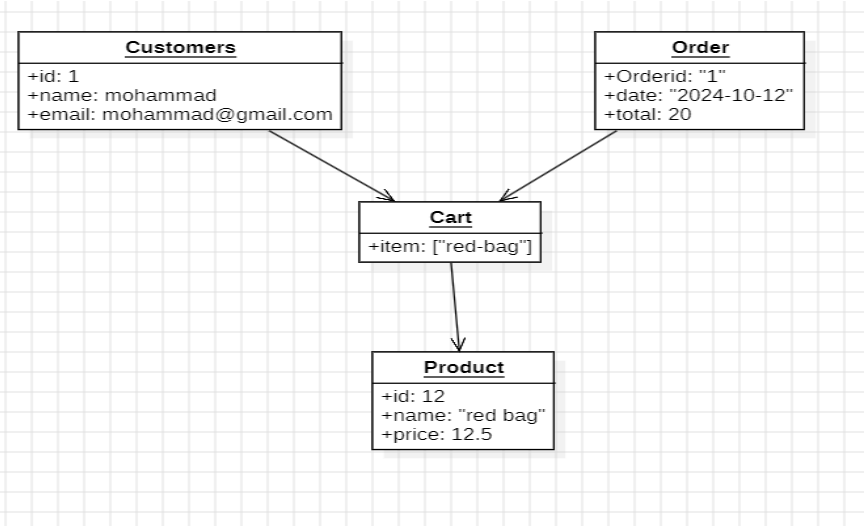
Relation to Use Cases:

- The **SupportTicket** class ensures smooth communication between customers and administrators for resolving issues.

Relationships:

1. **Customer ↔ Cart:**
   - A customer is associated with a single cart, representing the items they wish to purchase. This supports use cases like **Add to Cart** and **Proceed to Payment**.
2. **Customer ↔ SupportTicket:**
   - Customers can submit multiple support tickets, ensuring issues related to orders or payments are resolved.
3. **Admin ↔ Product:**
   - The admin has control over the product catalog, which is central to the **Manage Cloth** use case.
4. **Payment ↔ Cart:**
   - Payments are linked to a cart to calculate the total cost and process discounts, implementing the **Payment Page** use case.
5. **Customer ↔ Payment:**
   - Customers initiate payments for their purchases, completing the checkout process.

5. Object Diagram



Explanation of the Object Diagram

This **Object Diagram** represents a snapshot of the system's state at a specific point in time, showcasing the relationships and interactions between instances of the classes in the online clothing shop system. Here's a breakdown of the components and how they interact:

Objects and Their States

1. Customer Object:

- **Attributes:**
  - `id: 1`: This is the unique identifier for the customer.

    - `name: Mohammad`: Represents the name of the customer.
    - `email: mohammad@gmail.com`: The customer's email address, used for login and communication.
- **Role:**
    - The `Customer` object represents a user interacting with the system. In this snapshot, "Mohammad" is the customer associated with this transaction.

2. Product Object:

- **Attributes:**
    - `id: 12`: The unique identifier for the product.
    - `name: red bag`: The name of the product selected by the customer.
    - `price: 12.5`: The price of the product.
- **Role:**
    - The `Product` object is an item available in the shop that the customer can add to their cart. In this instance, "red bag" is the product being purchased.

3. Cart Object:

- **Attributes:**
    - `item: ["red-bag"]`: A list of products added to the cart. Currently, it contains the "red bag."
- **Role:**
    - The `Cart` object serves as a temporary storage for the products that the customer intends to purchase. It links the `Customer` and `Product` objects, enabling the checkout process.

4. Order Object:

- **Attributes:**
    - `OrderId: 1`: A unique identifier for the order.
    - `date: "2024-10-12"`: The date when the order was placed.
    - `total: 20`: The total cost of the order (in this case, including any additional charges or discounts).
- **Role:**
    - The `Order` object represents the finalized purchase. It is associated with the customer and reflects the cart's contents at the time of confirmation.

Relationships Between Objects

1. **Customer ↔ Cart:**
    - The customer (Mohammad) is linked to the cart, which holds the products they have selected. This represents the **Add to Cart** process from the use case.
2. **Cart ↔ Product:**
    - The cart contains a reference to the "red bag" product, showing that the customer has added it for potential purchase. This relationship supports the **Search Product** and **Add to Cart** use cases.

3. **Cart ↔ Order:**
    - Once the customer confirms their purchase, the cart is transformed into an order, linking the selected items to the finalized transaction. This interaction supports the **Confirm Order** step in the **Payment Page** use case.
4. **Customer ↔ Order:**
    - The customer is directly linked to their order, showing ownership of the transaction. This reflects the **Order Confirmation** process.

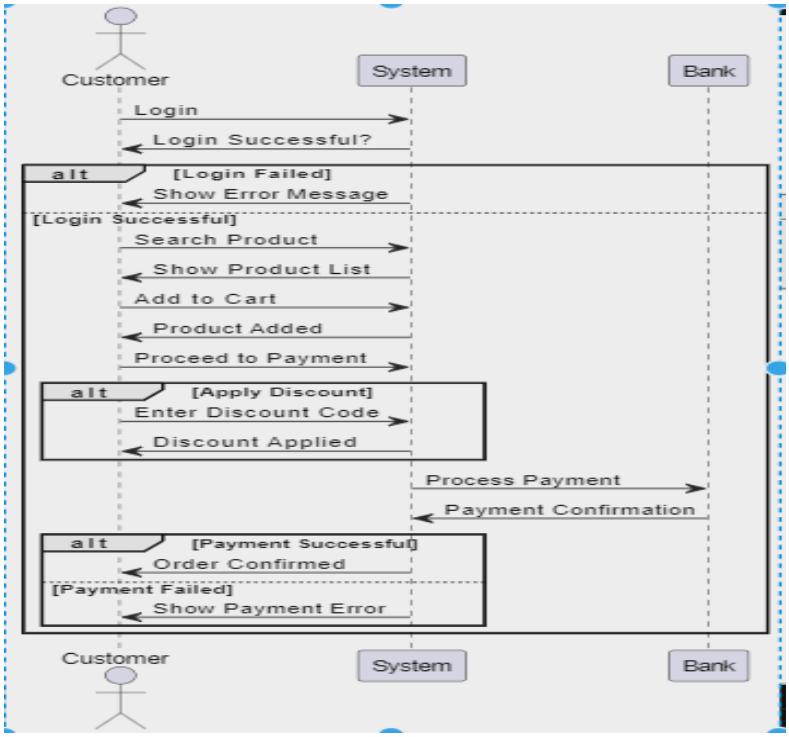
Summary

This object diagram illustrates a real-world example of the system's functionality:

1. Mohammad (customer) has logged in, searched for products, and added the "red bag" to their cart.
2. The cart contains this product and has been finalized into an order with ID `1`, placed on `"2024-10-12"`, with a total cost of `20`.

This snapshot effectively connects the **Customer**, **Product**, **Cart**, and **Order** objects, demonstrating the system's workflow from product selection to order confirmation.

6. Sequence Diagram (Purchase Process)



The **Sequence Diagram** provided outlines the step-by-step interactions and message exchanges between various actors and system components involved in the **purchase process** of the online

clothing shop. This diagram visually represents the **temporal flow of events** from the initiation of the purchase to the final order confirmation. Below is a detailed explanation:

Actors and Components

1. **Customer**:
    - The end-user (customer) interacting with the system to browse, add products to their cart, and complete the purchase.
2. **System Components**:
    - **Login Service**: Validates the customer's credentials for authentication.
    - **Catalog Service**: Provides the list of products, facilitates product searching, and filtering.
    - **Cart Service**: Manages the addition of items to the cart and handles discounts.
    - **Payment Gateway**: Processes payment transactions securely.
    - **Order Service**: Generates and confirms the final order.

Process Flow and Interactions

1. Login Process:

- The **Customer** sends their login credentials (username and password) to the **Login Service**.
- **Login Service** validates the credentials:
    - **If successful**, it allows the customer to proceed to the next step.
    - **If failed**, it returns an error message, halting further actions.

2. Browsing and Adding Products to the Cart:

- The **Customer** sends a request to the **Catalog Service** to browse or search for products. This might include:
    - **Search Queries**: Keywords or filters (e.g., price, category, brand).
- The **Catalog Service** returns the product details (e.g., name, price, availability).
- The **Customer** selects the desired product(s) and sends an **"Add to Cart"** request to the **Cart Service**.
- **Cart Service** updates the cart with the selected product(s) and confirms the addition.

3. Applying Discounts (Optional):

- The **Customer** can choose to apply a discount code or coupon to the cart.
- The **Cart Service** validates the discount and updates the cart's total if the discount is applicable.

4. Proceeding to Payment:

- Once the cart is finalized, the **Customer** sends a payment request to the **Payment Gateway**.
- The **Payment Gateway** processes the payment:
    - **If successful**, it confirms the transaction and forwards the confirmation to the system.
    - **If failed**, it returns an error message, prompting the customer to retry or use an alternate method.

5. Order Confirmation:

- After payment is confirmed, the **Order Service** is notified to create the final order.
- The **Order Service** generates an order ID, stores the order details, and sends the confirmation back to the customer.
- The **Customer** receives an order confirmation, completing the purchase process.

Key Points of Interaction:

1. **Authentication**: The system ensures that only authenticated users can proceed with the purchase.
2. **Seamless Integration**: Each component (Catalog, Cart, Payment Gateway, Order Service) works together to provide a smooth shopping experience.
3. **Error Handling**: Login failure, invalid discounts, or payment errors are appropriately handled with feedback to the customer.

Summary:

This sequence diagram provides a comprehensive view of the interactions and events required for completing a purchase:

- From **login** to **product browsing**,
- Adding items to the **cart**,
- Applying **discounts**,
- Finalizing the **payment**,
- And generating the **order confirmation**.

It ensures that every actor and system component is accounted for, showcasing a robust and user-friendly shopping experience for the online clothing store.