**Requirements Specifications**

**<P14>:<Shop Savvy>**

|  |  |
| --- | --- |
| **Student ID** | **Name** |
| **25100002** | **Ahmad Kashif Jabbar** |
| **25100105** | **Husnain Ali** |
| **25100179** | **Musa Aftab Ahmad** |
| **25100229** | **Syed Messam Ali** |
| **25100208** | **Zainab Fatima** |

|  |  |  |
| --- | --- | --- |
| **Content** | **Totals** | **Obtained** |
| Introduction & system actors | 5 | 5 |
| Use case diagram | 10 | 7 |
| Use case descriptions | 20 | 14 |
| Class diagram | 20 | 18 |
| Sequence diagram | 20 | 19 |
| State diagram | 5 | 5 |
| Non-functional requirements | 5 | 5 |
| Who did what | 5 | 5 |
| Review checklist | 5 | 5 |
| Overall formatting/template | 5 | 5 |
| Late submission penalty | -20 | - |
| GitHub Folder structure penalty | -5 | - |
| **Grand Total** | **100** | **88** |
| **General Comments/Individual Grading:** | | |

**Table of Contents**

[1.](#_gjdgxs) Introduction 3

[2.](#_30j0zll) System Actors 4

[3.](#_1fob9te) Use Cases 5

[3.1 Use Case Diagrams 5](#_3znysh7)

[3.2 Description of Use Cases 5](#_2et92p0)

[3.2.1 Withdraw cash 5](#_tyjcwt)

[3.2.2 Transfer funds 6](#_3dy6vkm)

[4.](#_1t3h5sf) Class Diagram 7

[4.1 Diagram 7](#_4d34og8)

[4.2 Description 7](#_2s8eyo1)

[5.](#_17dp8vu) Sequence Diagrams 8

[5.1 Use case Name e.g., Withdraw cash 8](#_3rdcrjn)

[5.2 Use case Name e.g., Transfer funds 8](#_26in1rg)

[6.](#_lnxbz9) State Diagrams 9

[6.1 Diagram details 9](#_35nkun2)

[6.2 Diagram 9](#_1ksv4uv)

[7.](#_44sinio) Non-functional Requirements / Quality Attributes 10

[8.](#_2jxsxqh) Who Did What? 11

[9.](#_z337ya) Review checklist 11

# Introduction

The AI-driven Personalized Clothing Recommendation Platform is designed to enhance the online shopping experience for Pakistani consumers by bringing together a curated selection of local clothing brands on a single platform. The platform will feature at least 10 prominent Pakistani clothing brands, allowing users to explore and shop from a variety of options conveniently in one place. The main objective is to simplify the shopping process by offering personalized recommendations tailored to individual user preferences, all while showcasing local fashion.

The platform’s primary users are Pakistani consumers who are looking for a seamless, convenient, and personalized shopping experience. By aggregating clothing options from multiple brands, the platform eliminates the need for shoppers to visit multiple websites. The AI-powered recommendation engine will leverage user data—such as browsing history, past purchases, and personal preferences—to suggest relevant products, making the experience more engaging and efficient.

In the long term, the platform has the potential to generate revenue through affiliate marketing by partnering with local brands, earning commissions on purchases made via the platform. This creates a win-win situation, providing visibility for the brands and a tailored shopping experience for consumers.

The platform aims to provide a personalized, convenient, and enjoyable online shopping experience, helping Pakistani consumers discover and purchase clothing from a range of local brands in a simple, AI-enhanced environment.

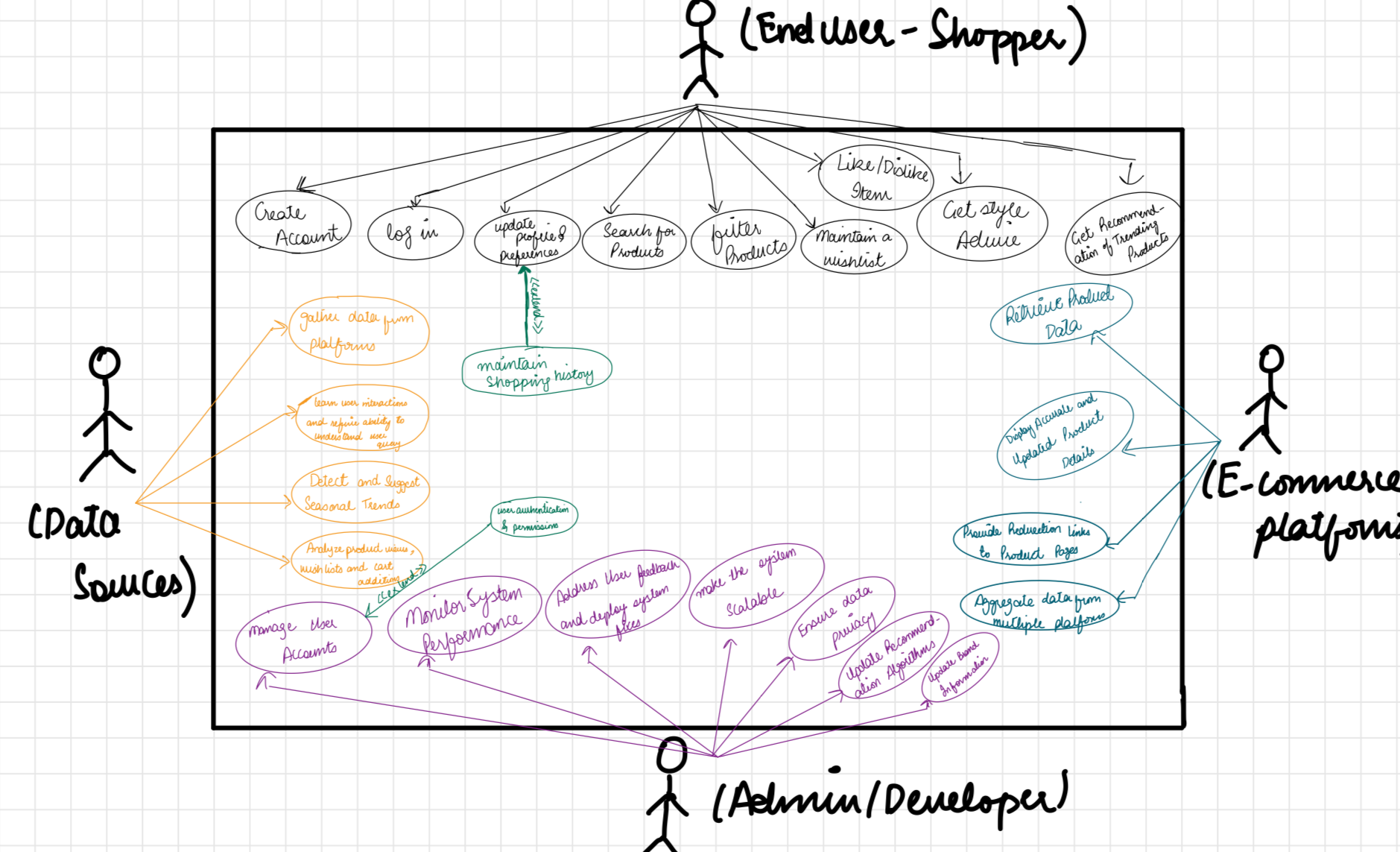
# System Actors

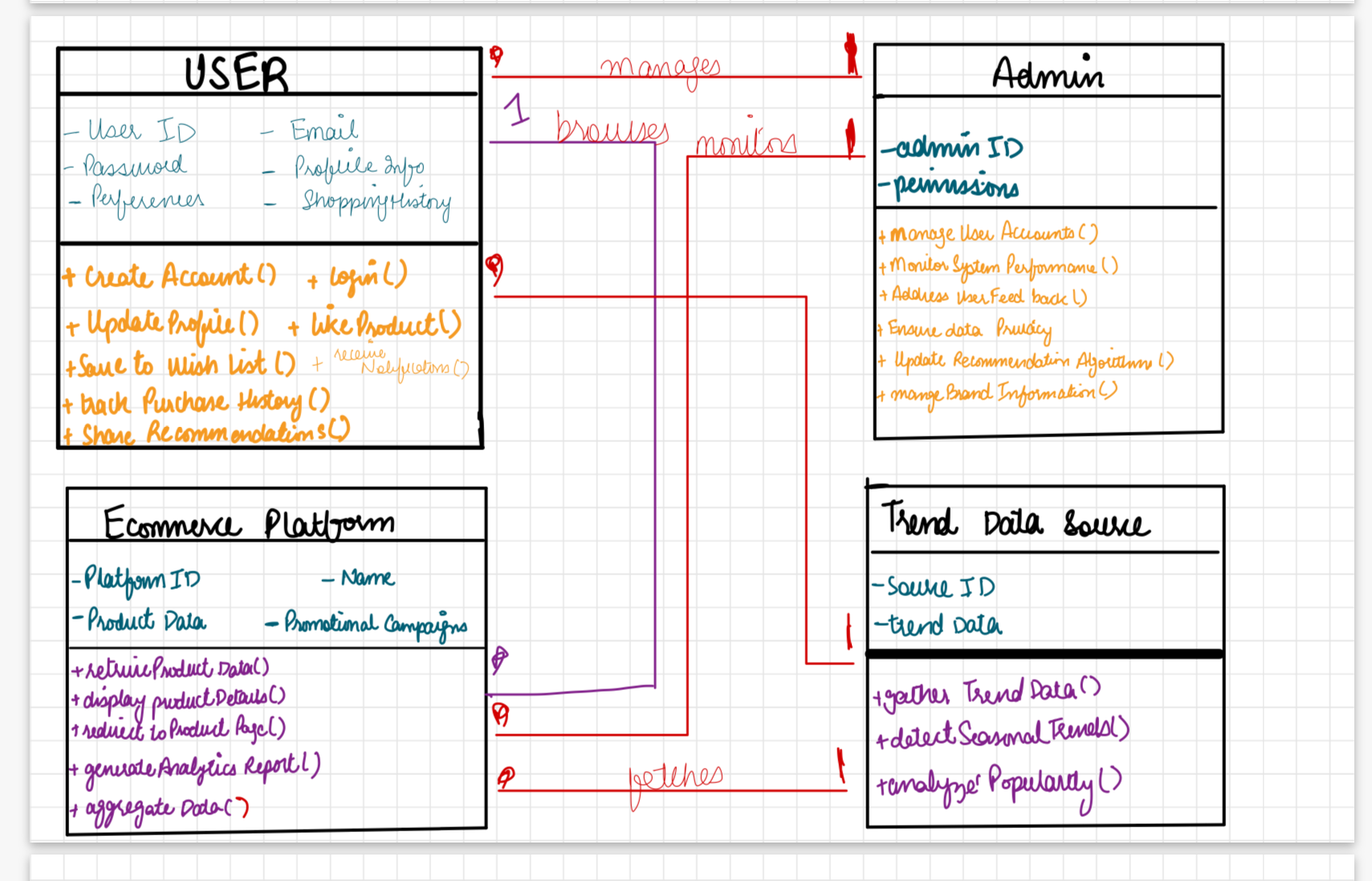
|  |  |
| --- | --- |
| **Actor Name** | **Description** |
| End User (Shopper) | The primary actor, the shopper, interacts with the virtual shopping assistant to browse, search, and receive personalized product recommendations. They provide inputs through queries, wish lists, and purchase history, enabling the system to tailor suggestions based on their preferences. |
| E-commerce Platforms | These represent local and international shopping brands that the assistant connects with. The platforms provide links to their product pages, along with details like descriptions, prices, and availability. |
| Admin/Developer | The system administrator or developer manages platform operations, monitors system performance, ensures API integrations, and updates recommendation algorithms. They also address user feedback, maintain data privacy, and scale the platform for growing user and brand integration. |
| Data Sources | The data sources combine real-time information on fashion trends from sources like social media, blogs, and external platforms with natural language processing capabilities. It helps the system not only suggest trending products based on current styles and consumer preferences but also understand and process user queries in conversational language. By learning from user interactions, it refines recommendations and enhances search accuracy, delivering personalized and trend-aware suggestions. |

# Use Cases

## Use Case Diagrams

[Please draw in some tool using UML notations, are data sources external to your system and using your system as a entity?].





## Description of Use Cases

[**Select 20 most important use cases of your project and create their comprehensive descriptions.**]

<Write description of each use case separately using the template below.>

### Manage user accounts

|  |  |  |
| --- | --- | --- |
| **Identifier** | | UC-001 |
| **Purpose** | | Admin can manage user accounts including registration, authentication, and permissions. |
| **Pre-conditions** | | Admin is logged in |
| **Post-conditions** | | Accounts are registered, authenticated, or updated. [Format the post-condition correctly]. |
|  | | |
| **Step #** | **Typical Course of Action** | |
| 1. **1** | Admin logs in | |
|  | Admin selects account management option. | |
|  | Admin performs actions (e.g., add, delete, modify). | |
|  | System updates data. | |
|  | | |
| **Step #** | **Alternate Courses of Action** | |
|  | Admin can choose to skip updates and go back to the dashboard. | |
| **Step #** | **Exception Paths** | |
| **1.** | If user input is invalid, an error message is displayed, and admin is prompted to correct the input. [at which step?] | |

### Monitor system performance

|  |  |  |
| --- | --- | --- |
| **Identifier** | | UC-002 |
| **Purpose** | | Admin monitors system performance and integrations with e-commerce platforms. |
| **Pre-conditions** | | System is running and admin has access |
| **Post-conditions** | | Admin views the status of system performance and integrations. |
|  | | |
| **Step #** | **Typical Course of Action** | |
| **1.** | Admin logs in | |
| **2.** | Admin checks performance metrics | |
| **3/** | Asmin identifies performance issues | |
| **4.** | Admin takes actions as necessary | |
|  | | |
| **Step #** | **Alternate Courses of Action** | |
| 1 | Admin can skip detailed checks and directly apply general fixes. | |
| **Step #** | **Exception Paths** | |
| 2. | If system integration is broken, an alert is generated, and further diagnostic steps are required. | |

3.3.3 Address user feedback and deploy fixes

|  |  |  |
| --- | --- | --- |
| **Identifier** | | UC - 003 |
| **Purpose** | | Admin addresses feedback including bug reports and deploys fixes rapidly. |
| **Pre-conditions** | | User feedback is received |
| **Post-conditions** | | Bug is fixed or feedback is addressed, the system is updated. |
|  | | |
| **Step #** | **Typical Course of Action** | |
| **1.** | Admin logs in | |
| **2.** | Admin reviews feedback | |
| **3.** | Admin resolves issues or fixes bug | |
| **4.** | System is updated | |
|  |  | |
|  | | |
| **Step #** | **Alternate Courses of Action** | |
| 1. | Admin can choose to mark non-critical issues for later resolution. | |
| **Step #** | **Exception Paths** | |
|  | If the issue persists, admin is prompted to escalate or rollback changes. | |

3.3.4 Gather trend-related data

|  |  |  |
| --- | --- | --- |
| **Identifier** | | UC - 004 |
| **Purpose** | | The system gathers data from social media and fashion blogs to suggest trending products. |
| **Pre-conditions** | | Trend Data is available |
| **Post-conditions** | | System learns from new trends and suggests trending items. |
|  | | |
| **Step #** | **Typical Course of Action** | |
| **1.** | System scrapes social media/blogs. | |
| **2.** | Data is processed. | |
| **3.** | Trends are identified and applied to suggestions. | |
|  | | |
| **Step #** | **Alternate Courses of Action** | |
| 1. | System can prioritize data from specific platforms or blogs based on admin configuration. | |
| **Step #** | **Exception Paths** | |
| 1. | If data is incomplete, the system retries data gathering from backup sources. | |

3.3.5 Refine NLP model with user interaction

|  |  |  |
| --- | --- | --- |
| **Identifier** | | UC-005 |
| **Purpose** | | The system refines its NLP capabilities based on user queries and interactions over time |
| **Pre-conditions** | | The system has collected user query data. |
| **Post-conditions** | | Improved understanding of user queries and intents. |
|  | | |
| **Step #** | **Typical Course of Action** | |
| **1.** | System processes user inputs. | |
| **2.** | User interactions are analyzed | |
| **3.** | NLP model is refined based on insights | |
| **4.** | Recommendations improve | |
|  | | |
| **Step #** | **Alternate Courses of Action** | |
| 1. | System can apply existing data if interactions are too limited for meaningful refinement. | |
| **Step #** | **Exception Paths** | |
| 1. | If data is insufficient for model improvement, refinement is postponed until more data is available. | |

3.3.6 Detect seasonal trends

|  |  |  |
| --- | --- | --- |
| **Identifier** | | UC - 006 |
| **Purpose** | | The system detects seasonal clothing trends, such as festive or weather-based preferences. |
| **Pre-conditions** | | Seasonal or festive data is available. |
| **Post-conditions** | | Seasonal trends are identified and suggested to users. |
|  | | |
| **Step #** | **Typical Course of Action** | |
| **1.** | System gathers data on seasonal trends. | |
| **2.** | Season trends are identified | |
| **3.** | System updates recommendations accordingly | |
|  | | |
| **Step #** | **Alternate Courses of Action** | |
| 1. | System can apply last year's seasonal data as a fallback if current data is lacking. | |
| **Step #** | **Exception Paths** | |
| 1. | If no trends are detected, the system flags the issue for admin review and skips seasonal recommendations | |

3.3.7 Analyze product views and wish lists

|  |  |  |
| --- | --- | --- |
| **Identifier** | | UC--007 |
| **Purpose** | | The system analyzes product views, wish lists, and cart additions to identify popular products. |
| **Pre-conditions** | | User data on product views, wish lists is available. |
| **Post-conditions** | | Popular products are identified and suggested to more users. |
|  | | |
| **Step #** | **Typical Course of Action** | |
| **1** | System tracks product views, wish lists, and cart additions. | |
| **2** | Analysis is conducted | |
| **3.** | Popular products are flagged and recommended. | |
|  | | |
| **Step #** | **Alternate Courses of Action** | |
| 1. | |  | | --- | | Admin can prioritize specific metrics (e.g., views over wish lists) for analysis. | | |
| **Step #** | **Exception Paths** | |
| 1. | If data is inconclusive, the system defers recommendations for manual review. | |

3.3.8 Create Account

|  |  |  |
| --- | --- | --- |
| **Identifier** | | UC-008 |
| **Purpose** | | End user creates an account using email or social media profiles. |
| **Pre-conditions** | | User has email or social media account. |
| **Post-conditions** | | Account is created, user is authenticated. |
|  | | |
| **Step #** | **Typical Course of Action** | |
| **1.** | User signs up. | |
| **2.** | User inputs details | |
| **3.** | System verifies and creates account | |
| **4.** | User is authenticated | |
|  |  | |
|  | | |
| **Step #** | **Alternate Courses of Action** | |
| 1. | User can skip email verification and use social media login for quicker sign-up. | |
| **Step #** | **Exception Paths** | |
| 1. | If email/social media validation fails, the user is prompted to retry or choose another method. | |

3.3.9 Update profile

|  |  |  |
| --- | --- | --- |
| **Identifier** | | UC - 009 |
| **Purpose** | | End user updates their personal information, preferences, and shopping history. |
| **Pre-conditions** | | User has an account and is logged in. |
| **Post-conditions** | | Profile is updated with new information |
|  | | |
| **Step #** | **Typical Course of Action** | |
| **1.** | User logs in. | |
| **2.** | User navigates to profile settings. | |
| **3.** | User updates information. | |
| **4.** | System saves the updates. | |
|  | | |
| **Step #** | **Alternate Courses of Action** | |
| 1. | User can choose to revert changes before saving. | |
| **Step #** | **Exception Paths** | |
| 1. | If profile update fails, the user is notified, and the system retains the previous profile information. | |

3.3.10 Receive personalized product recommendations

|  |  |  |
| --- | --- | --- |
| **Identifier** | | UC - 010 |
| **Purpose** | | End user receives personalized recommendations based on preferences and past purchases. |
| **Pre-conditions** | | User profile and purchase history are available. |
| **Post-conditions** | | Tailored product recommendations are shown to the user. |
|  | | |
| **Step #** | **Typical Course of Action** | |
| **1.** | User logs in | |
| **2.** | System analyzes preferences and past purchases. | |
| **3.** | Recommendations are displayed | |
|  | | |
| **Step #** | **Alternate Courses of Action** | |
| 1. | System can prioritize current trends over past purchases for recommendations. | |
| **Step #** | **Exception Paths** | |
| 1. | If recommendations cannot be generated, the system defaults to general popular products. | |

3.3.11 Search for products using natural language queries

|  |  |  |
| --- | --- | --- |
| **Identifier** | | UC-011 |
| **Purpose** | | End user searches for products using natural language queries (e.g., “Show me affordable jeans”). |
| **Pre-conditions** | | User is logged in and inputs a query. |
| **Post-conditions** | | Search results match user’s natural language query. |
|  | | |
| **Step #** | **Typical Course of Action** | |
| **1.** | User inputs query | |
| **2.** | System interprets query using NLP | |
| **3.** | System retrieves matching results. | |
| **4.** | Results are shown to user | |
|  | | |
| **Step #** | **Alternate Courses of Action** | |
| 1. | User can choose to refine the query for more specific results. | |
| **Step #** | **Exception Paths** | |
| 1. | If no products match the query, the system suggests related alternatives or prompts the user to refine. | |

3.3.12 Like or dislike products

|  |  |  |
| --- | --- | --- |
| **Identifier** | | UC-012 |
| **Purpose** | | End user likes or dislikes products, refining future recommendations |
| **Pre-conditions** | | User is logged in and has viewed products. |
| **Post-conditions** | | Future product recommendations are refined based on user feedback. |
|  | | |
| **Step #** | **Typical Course of Action** | |
|  | User views products. | |
|  | User likes/dislikes a product. | |
|  | System records user feedback | |
| **8.** | Recommendations improve based on input [how? Any dependent use case?] | |
|  | | |
| **Step #** | **Alternate Courses of Action** | |
| 1. | User can undo a like/dislike action if done accidentally | |
| **Step #** | **Exception Paths** | |
| 1. | If feedback is not recorded, the user is notified, and the action is retried. | |

3.3.13 Save products to a wishlist

|  |  |  |
| --- | --- | --- |
| **Identifier** | | UC - 013 |
| **Purpose** | | End user saves products to a wishlist for future reference. |
| **Pre-conditions** | | User is logged in. |
| **Post-conditions** | | Products are added to user’s wishlist |
|  | | |
| **Step #** | **Typical Course of Action** | |
| 1. **1.** | User selects a product. | |
| 1. **2.** | User saves the product to wishlist. | |
| 1. **3.** | System updates user’s wishlist. | |
|  | | |
| **Step #** | **Alternate Courses of Action** | |
| 1. | User can remove an item from the wishlist at any time | |
| **Step #** | **Exception Paths** | |
|  | If the product cannot be saved, an error is displayed, and the user is prompted to retry. | |

3.3.14 Filter products based on price, brand, and category

|  |  |  |
| --- | --- | --- |
| **Identifier** | | UC-014 |
| **Purpose** | | End user filters products based on specific criteria (e.g., price, brand, or category). |
| **Pre-conditions** | | User has performed a search or is browsing |
| **Post-conditions** | | Filtered product results are displayed. |
|  | | |
| **Step #** | **Typical Course of Action** | |
| **1.** | User inputs filter criteria. | |
| **2.** | System applies filters | |
| **3.** | Filtered results are displayed to user | |
|  | | |
| **Step #** | **Alternate Courses of Action** | |
| 1. | User can change filter criteria at any time for a different view. | |
| **Step #** | **Exception Paths** | |
|  | If filtering fails, all products are displayed by default. | |

3.3.15 Track purchase history

|  |  |  |
| --- | --- | --- |
| **Identifier** | | UC-015 |
| **Purpose** | | End user tracks their purchase history through the platform. |
| **Pre-conditions** | | User has made purchases through the platform. |
| **Post-conditions** | | Purchase history is available to the user. |
|  | | |
| **Step #** | **Typical Course of Action** | |
| **1.** | User logs in. | |
| **2.** | User navigates to their profile | |
| **3.** | User selects ‘Purchase History’ | |
| **4.** | System retrieves past orders and displays them | |
|  | | |
| **Step #** | **Alternate Courses of Action** | |
| 1. | User can filter purchase history by date, price, or category. | |
| **Step #** | **Exception Paths** | |
| 2. | If no purchase history is found, a message is displayed to the user | |

3.3.16 Retrieve product data

|  |  |  |
| --- | --- | --- |
| **Identifier** | | UC-016 |
| **Purpose** | | The system retrieves product data like descriptions, reviews, prices, and availability from e-commerce APIs or web scraping. |
| **Pre-conditions** | | APIs or scraping tools are available and functioning |
| **Post-conditions** | | Product data is retrieved and displayed to users. |
|  | | |
| **Step #** | **Typical Course of Action** | |
| **1.** | System sends request to e-commerce API or scrapes website. | |
| **2.** | Product data is fetched. | |
| **3.** | System displays data to users | |
|  | | |
| **Step #** | **Alternate Courses of Action** | |
| 1. | System can fallback to previously cached data if API or scraping fails. | |
| **Step #** | **Exception Paths** | |
| 2. | If the product data cannot be fetched, a message informs the user and no data is displayed. | |

3.3.17 Display real-time product details

|  |  |  |
| --- | --- | --- |
| **Identifier** | | UC-017 |
| **Purpose** | | The system displays accurate and up-to-date product details to users in real-time. |
| **Pre-conditions** | | APIs or scraping tools are available and functioning. |
| **Post-conditions** | | Real-time product details are visible to users. |
|  | | |
| **Step #** | **Typical Course of Action** | |
| **1.** | System sends request for product details | |
| **2.** | Product details are displayed in real-time. | |
|  | | |
| **Step #** | **Alternate Courses of Action** | |
| 1. | If the system detects outdated data, it can request an update. | |
| **Step #** | **Exception Paths** | |
| 2. | If real-time details are unavailable, a message informs users of the issue. | |

3.3.18 Seamlessly integrate promotional campaigns

|  |  |  |
| --- | --- | --- |
| **Identifier** | | UC-018 |
| **Purpose** | | The system integrates promotional campaigns and deals for products |
| **Pre-conditions** | | Promotional campaigns and deals are available. |
| **Post-conditions** | | Promotional deals are visible and integrated into product listings. |
|  | | |
| **Step #** | **Typical Course of Action** | |
| **1.** | System receives promotions from e-commerce platforms | |
| **2.** | Promotions are applied to relevant products. | |
| **3.** | System displays updated pricing | |
|  | | |
| **Step #** | **Alternate Courses of Action** | |
| 1. | System can temporarily disable promotions for debugging purposes. | |
| **Step #** | **Exception Paths** | |
| 2. | If promotions fail to load, users are informed, and full prices are shown. | |

3.3.19 Provide redirection links

|  |  |  |
| --- | --- | --- |
| **Identifier** | | UC-019 |
| **Purpose** | | The system provides redirection links to product pages where users can complete purchases |
| **Pre-conditions** | | Products have valid e-commerce links. |
| **Post-conditions** | | Users are redirected to external platforms to complete their purchases. |
|  | | |
| **Step #** | **Typical Course of Action** | |
| **1.** | User clicks on a product link. | |
| **2.** | System generates and displays a redirection link. | |
| **3.** | User is redirected to the external platform | |
|  | | |
| **Step #** | **Alternate Courses of Action** | |
| 1. | System can offer users a "confirm" option before redirection occurs. | |
| **Step #** | **Exception Paths** | |
| 2. | If redirection fails, users are shown a message and remain on the platform. | |

3.3.20 Generate user behavior analytics reports

|  |  |  |
| --- | --- | --- |
| **Identifier** | | UC-020 |
| **Purpose** | | The system generates analytics reports to help understand purchasing patterns and optimize offerings. |
| **Pre-conditions** | | The system has collected sufficient user data. |
| **Post-conditions** | | E-commerce platforms receive detailed reports on user behavior. |
|  | | |
| **Step #** | **Typical Course of Action** | |
| **1.** | System collects user data (views, purchases, etc.). | |
| **2.** | Data is processed and analyzed. | |
| **3.** | Reports are generated and provided to platforms. | |
|  | | |
| **Step #** | **Alternate Courses of Action** | |
| 1. | System can allow custom report generation options for platform admins. | |
| **Step #** | **Exception Paths** | |
| 2. | If data processing fails, the system retries or sends a failure notification to admins. | |

# Class Diagram

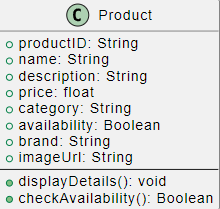
## Diagram

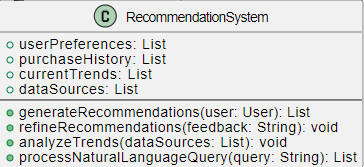
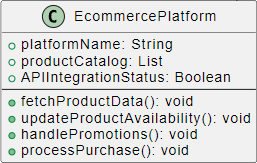
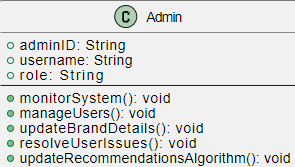
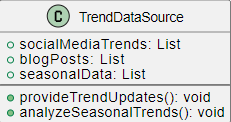
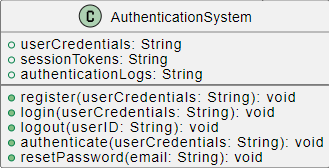
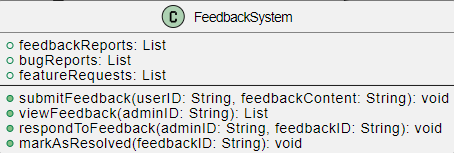
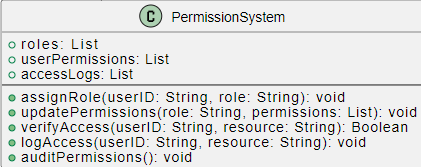
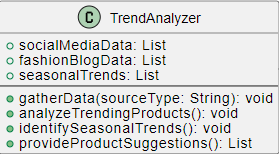
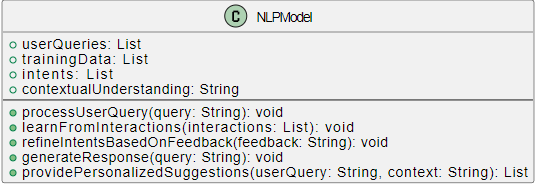
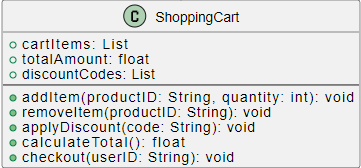
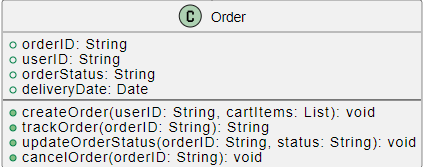
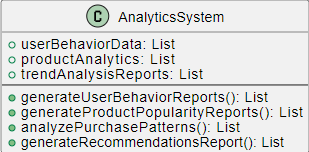
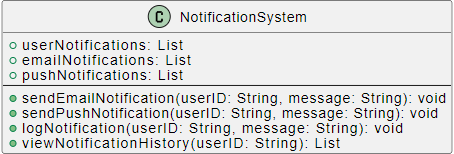
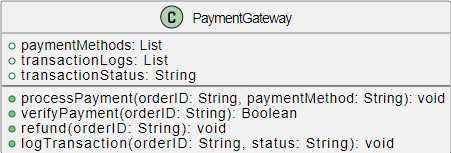
## [how trends and analytics getting data? These don’t have any relationship within the class diagram]. Link: [*https://drive.google.com/file/d/1EwOA7rFcPIuWlEnzRbDAlaj85BcdJf09/view?usp=sharing*](https://drive.google.com/file/d/1EwOA7rFcPIuWlEnzRbDAlaj85BcdJf09/view?usp=sharing)

## Description

* + 1. **User (Shopper)**
* Represents the end-user of the platform who signs up, browses, and purchases products. The class handles user details, their shopping activity, preferences, and interactions with other systems like the recommendation system and shopping cart.
* **Key Attributes**: userID, username, email, preferences, purchaseHistory, shoppingCart.
* **Key Methods**: createAccount(), login(), checkout(), receiveRecommendations().

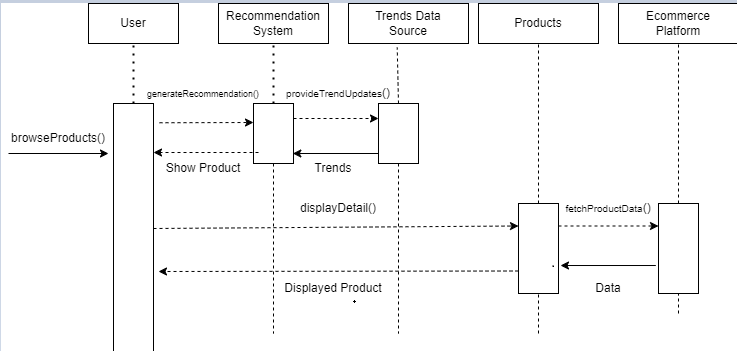


* + 1. **Product**
* Represents the products (fashion items) available on the platform. Each product has attributes like price, availability, and category. The system displays product details to the user.
* **Key Attributes**: productID, name, price, category, availability, brand, imageUrl.
* **Key Methods**: displayDetails(), checkAvailability().  
    
  

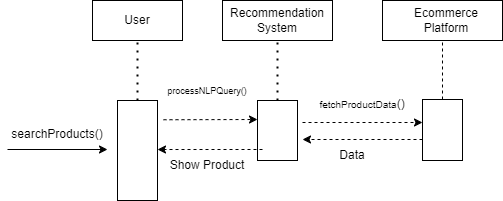
* + 1. **RecommendationSystem (AI Model)**
* Analyzes user preferences, purchase history, and current trends to provide personalized recommendations. It refines recommendations based on user feedback.
* **Key Attributes**: userPreferences, purchaseHistory, currentTrends, dataSources.
* **Key Methods**: generateRecommendations(user), refineRecommendations(feedback), analyzeTrends().  
    
  
  + 1. **EcommercePlatform**
* The central platform that integrates the product catalog, manages product data, and handles key functions like promotions and purchases. It interacts with users to fetch product data and process orders.
* **Key Attributes**: platformName, productCatalog, APIIntegrationStatus.
* **Key Methods**: fetchProductData(), processPurchase().  
    
    
  + 1. **Admin**
* Manages the platform operations, including user accounts, brand details, and resolving user issues. Admins also have control over the recommendation system algorithms and system monitoring.
* **Key Attributes**: adminID, username, role.
* **Key Methods**: monitorSystem(), manageUsers(), resolveUserIssues(), updateRecommendationsAlgorithm().  
    
    
  + 1. **TrendDataSource**
* Gathers fashion trends from external sources such as social media, blogs, and seasonal data. This data is fed into the system to influence recommendations and trend analysis.
* **Key Attributes**: socialMediaTrends, blogPosts, seasonalData
* **Key Methods**: provideTrendUpdates(), analyzeSeasonalTrends().  
    
    
    
  + 1. **AuthenticationSystem**
* Manages user login, registration, and authentication. It tracks session tokens and ensures secure access to the platform. It is responsible for password resets and validating credentials.
* **Key Attributes**: userCredentials, sessionTokens, authenticationLogs.
* **Key Methods**: register(), login(), authenticate(), resetPassword().  
    
  
  + 1. **FeedbackSystem**
* Allows users to submit feedback, bug reports, and feature requests. Admins can view, respond to, and resolve feedback items, creating a feedback loop between users and the platform.
* **Key Attributes**: feedbackReports, bugReports, featureRequests.
* **Key Methods**: submitFeedback(), viewFeedback(), respondToFeedback().  
    
  
  + 1. **PermissionSystem**
* Manages roles and permissions within the platform. Primarily used by admins to assign roles to users, update access permissions, and audit system access logs.
* **Key Attributes**: roles, userPermissions, accessLogs.
* **Key Methods**: assignRole(), updatePermissions(), verifyAccess().  
    
  
  + 1. **TrendAnalyzer**
* Processes data from the TrendDataSource to analyze and identify trending products. It helps in providing suggestions based on seasonal fashion trends and user preferences.
* **Key Attributes**: socialMediaData, fashionBlogData, seasonalTrends.
* **Key Methods**: gatherData(), analyzeTrendingProducts(), identifySeasonalTrends().  
    
    
  + 1. **NLPModel**
* A smart search engine that interprets user input beyond basic keyword matching. It processes user queries and attempts to match intent with actual product data, refining its understanding over time.
* **Key Attributes**: userQueries, trainingData, intents, contextualUnderstanding.
* **Key Methods**: processUserQuery(), learnFromInteractions(), generateResponse().  
    
  
  + 1. **ShoppingCart**
* Manages the products selected by the user for purchase. It calculates the total cost, applies discounts, and supports the checkout process.
* **Key Attributes**: cartItems, totalAmount, discountCodes.
* **Key Methods**: addItem(), removeItem(), applyDiscount(), checkout().  
    
    
  + 1. **Order**
* Handles order-related operations. This class tracks user orders, monitors their status, and allows the user to cancel or track their order.
* **Key Attributes**: orderID, userID, orderStatus, deliveryDate.
* **Key Methods**: createOrder(), trackOrder(), updateOrderStatus(), cancelOrder().  
    
    
  + 1. **AnalyticsSystem**
* Analyzes user behavior, product performance, and purchasing patterns. It generates reports that help optimize product offerings and recommendations.
* **Key Attributes**: userBehaviorData, productAnalytics, trendAnalysisReports.
* **Key Methods**: generateUserBehaviorReports(), analyzePurchasePatterns(), generateRecommendationsReport().  
    
    
  + 1. **NotificationSystem**
* Sends notifications to users via email or push notifications about their order status, promotions, or recommendations. It also logs notification history.
* **Key Attributes**: userNotifications, emailNotifications, pushNotifications.
* **Key Methods**: sendEmailNotification(), sendPushNotification(), logNotification().  
    
  
  + 1. **PaymentGateway**
* Manages payment transactions for user orders. It verifies payments, handles refunds, and logs transaction details.
* **Key Attributes**: paymentMethods, transactionLogs, transactionStatus.
* **Key Methods**: processPayment(), verifyPayment(), refund().  
    
  

**5. Sequence Diagrams**

## 5.1 Browse Products on Feed



## 5.2 Search For Products

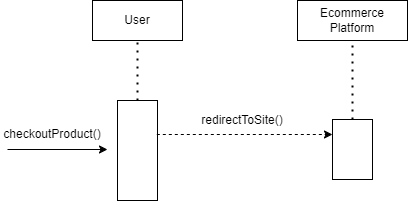


## 5.3 Like a product

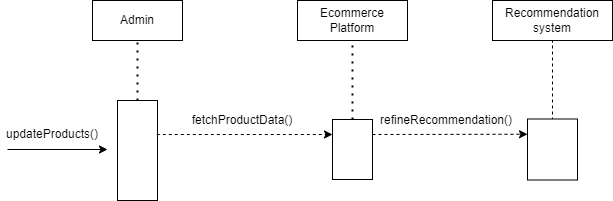
## 

## 

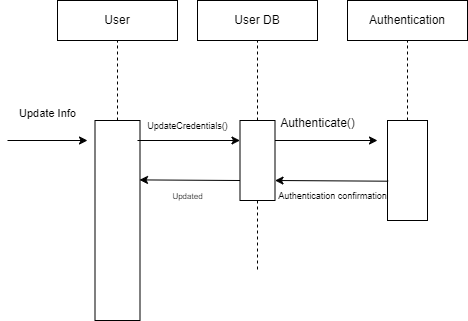
## 5.4 CheckOut Products / Redirecting to E commerce Website



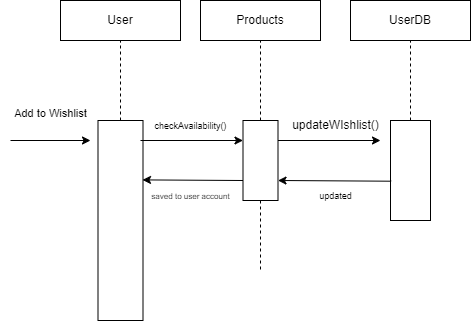
## 5.5 Update Products On Feed As Admin



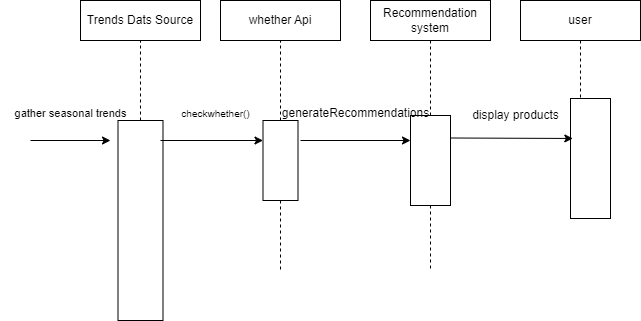
## 5.6 Update Personal Information



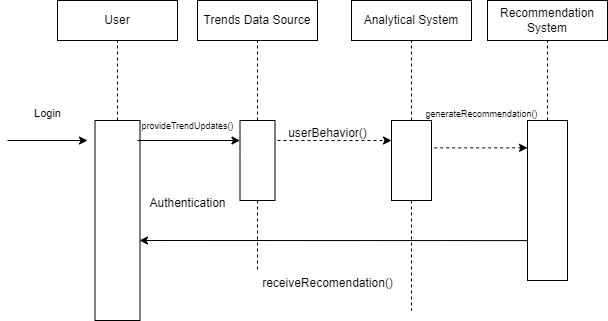
## 5.7 Update User WishList



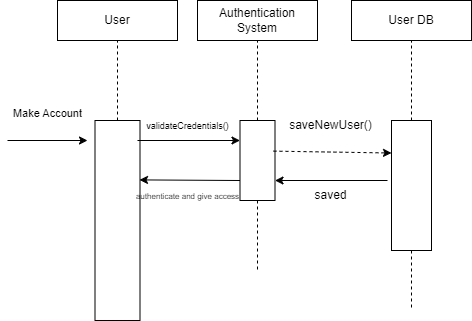
## 5.8 Analyze Seasonal Trends For Recommendation



## 5.9 Analyzing User Behavior For Recommendations



## 5.10 Create An Account as a User



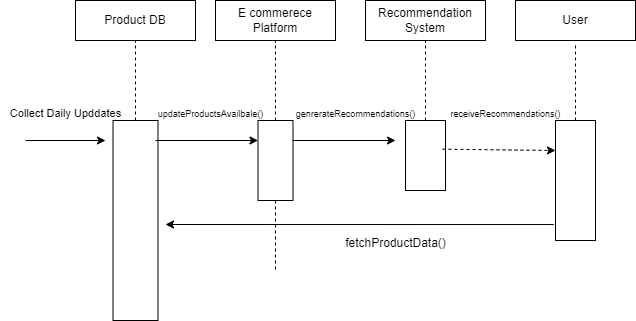
## 5.11 Feedback integration in recommendations

## 5.12 Filter Search Products

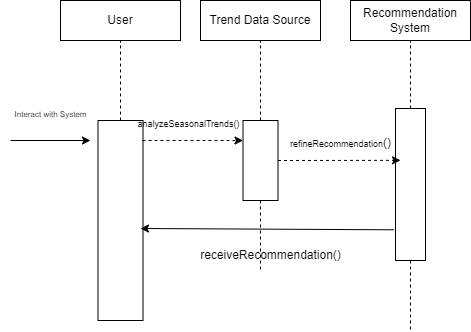
## 5.13 Gather trends using

## 5.14 Generate user Behavior Reports

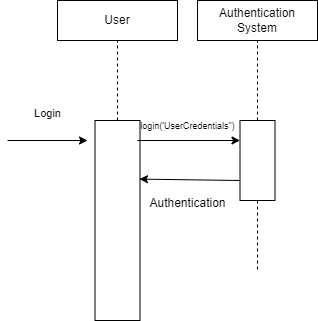
## 5.15 Gather real time data form e commerce website for updated recommendations



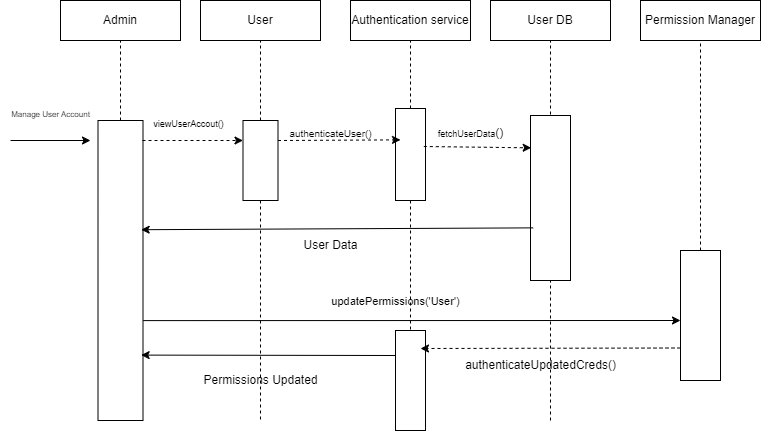
## 5.16 Learning User Behaviors and Refining Recommendations



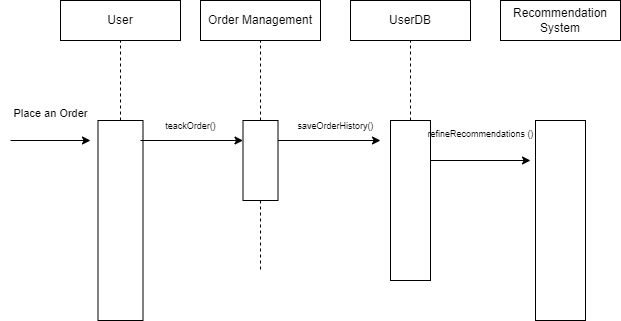
## 5.17 Login to A User Account



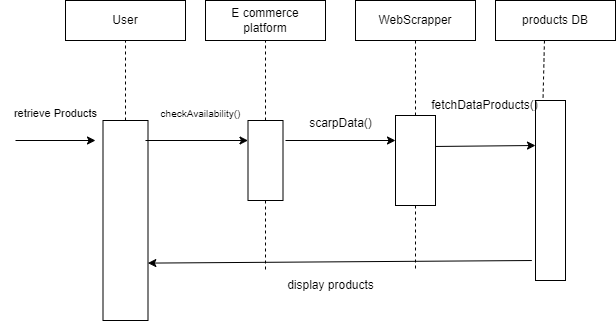
## 5.18 Manage User Account permissions as an Admin



## 5.19 Tracking purchase history for improved personalisation



## 5.20 Scrapping Fashion data from E commerce Websites



# 6. State Diagrams

**6.1 User Account State Diagram Details**

|  |  |
| --- | --- |
| **State** | **Description** |
| Guest | The user is not registered or logged in. Can browse products but cannot access additional features. |
| Registered | The user has created an account and can access more features but is not logged in. |
| Logged In | The user is logged into their account and has full access to the account dashboard and its features. |
| Profile Updated | The user has successfully updated their profile information. |
| Logged Out | The user has logged out of their account. The system displays the login screen. |

**Transitions:**

● Guest → Registered: Account creation

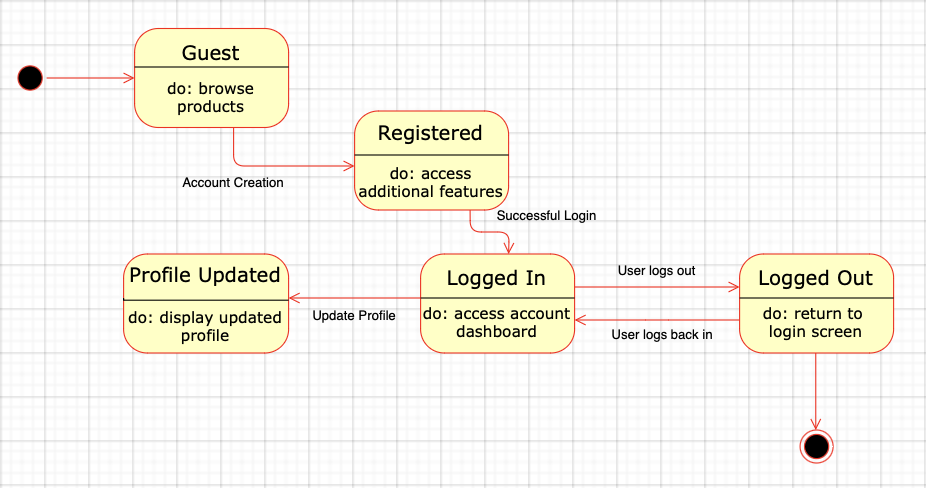
● Registered → Logged In: Successful login

● Logged In → Profile Updated: Update profile

● Logged In → Logged Out: User logs out

● Logged Out → Logged In: User logs back in

**State Diagram**



**6.2 Product Recommendation State Diagram Details**

|  |  |
| --- | --- |
| **State** | **Description** |
| Idle | The system is waiting for user action, such as submitting a search query for product recommendations. |
| Query Received | The system has received the user’s input (query) and is ready to process it. |
| Fetching Recommendations | The system is querying the product database to retrieve personalized recommendations. |
| Displaying Recommendations | The system displays the personalized product recommendations to the user. |
| Refining Recommendations | The system refines recommendations based on user feedback, showing updated suggestions. |

**Transitions:**

● Idle → Query Received: User submits query

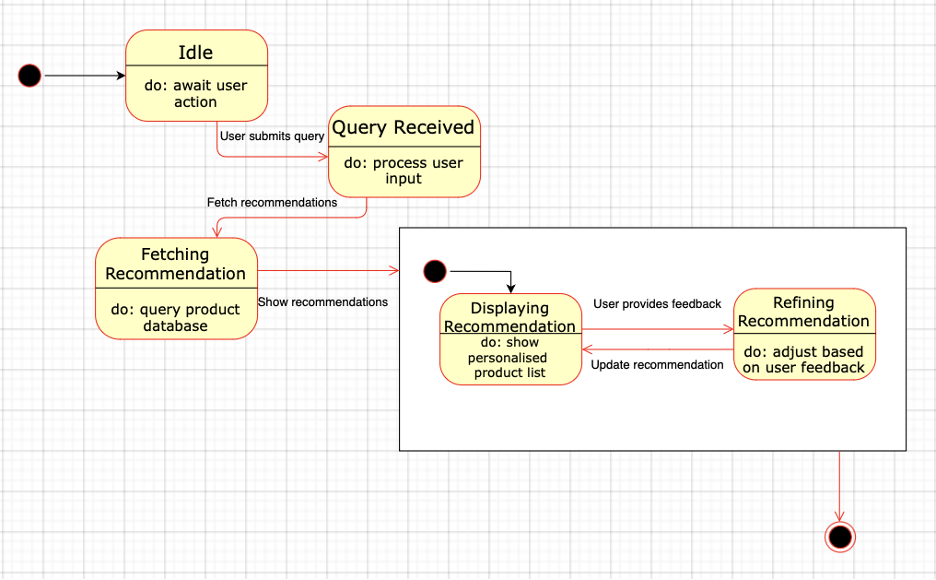
● Query Received → Fetching Recommendations: Fetch recommendations

● Fetching Recommendations → Displaying Recommendations: Show recommendations

● Displaying Recommendations → Refining Recommendations: User provides feedback

● Refining Recommendations → Displaying Recommendations: Update recommendation

**State Diagram**



**6.3 Admin Dashboard State Diagram Details**

|  |  |
| --- | --- |
| **State** | **Description** |
| Idle | The admin is idle, and no specific task is selected. |
| Managing Users | The admin is managing user accounts, which may include creating, updating, or deleting users. |
| Monitoring System | The admin is viewing and tracking system performance and metrics. |
| Handling Feedback | The admin is resolving user feedback or issues reported by users. |
| Updating Algorithms | The admin is making modifications or updates to the recommendation algorithm. |

**Transitions:**

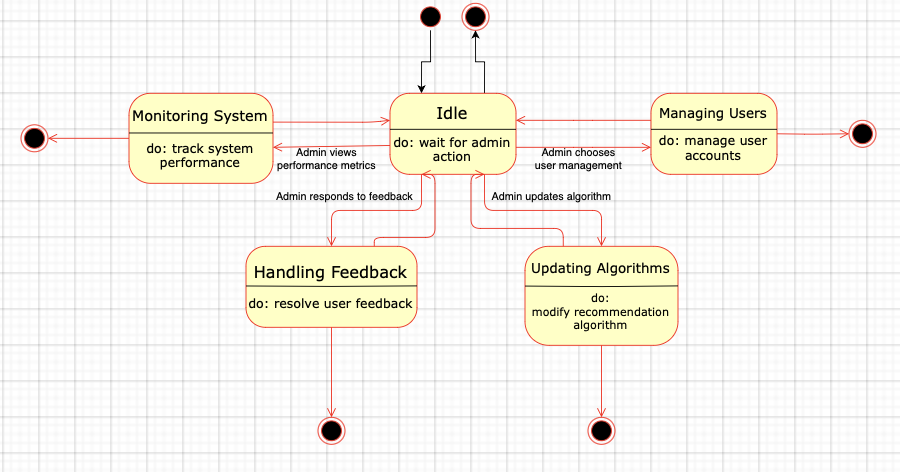
● Idle → Managing Users: Admin chooses user management

● Idle → Monitoring System: Admin views performance metrics

● Idle → Handling Feedback: Admin responds to feedback

● Idle → Updating Algorithms: Admin updates algorithms

**State Diagram**

**  
  
  
6.4 E-commerce Platform Integration State Diagram Details**

|  |  |
| --- | --- |
| **State** | **Description** |
| Idle | The system is waiting for an integration request to fetch product data from the e-commerce platform. |
| Retrieving Data | The system is fetching product data from the e-commerce platform based on the user's request. |
| Displaying Data | The system displays the retrieved product data (e.g., prices, descriptions) to the user. |
| Updating Data | The system is updating existing data with new information, possibly changing product details. |
| Error State | The system encountered an error during data retrieval or display and shows an error message to the user. |

**Transitions:**

● Idle → Retrieving Data: Initiate data retrieval

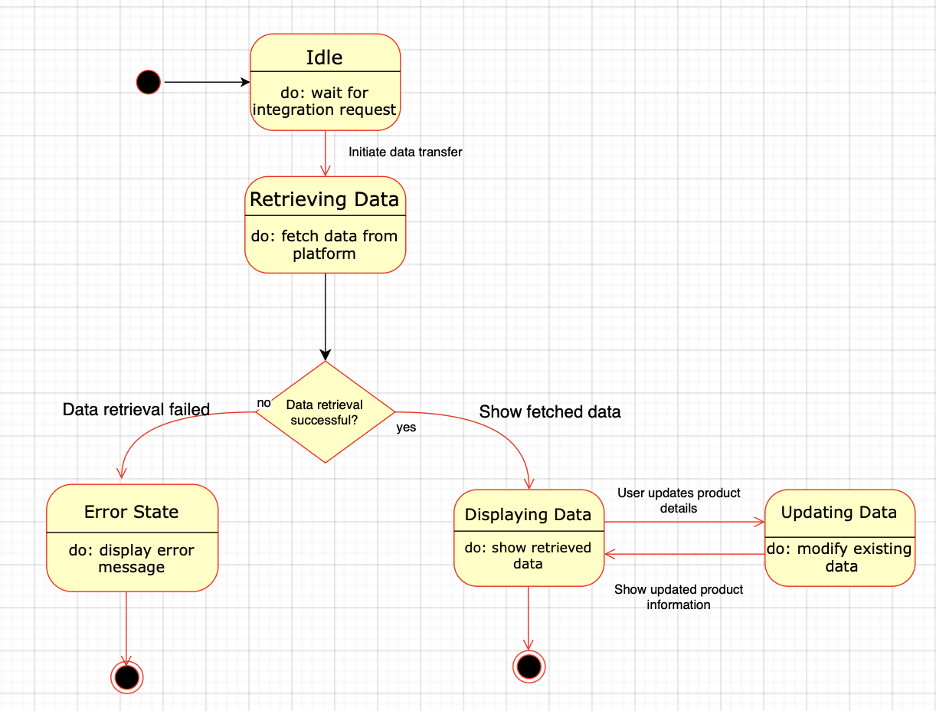
● Retrieving Data → Displaying Data: Show fetched data

● Retrieving Data → Error State: Data retrieval failed

● Displaying Data → Updating Data: User updates product details

● Updating Data → Displaying Data: Show updated product information

**State Diagram**



**6.5 User Interaction State Diagram Details**

|  |  |
| --- | --- |
| **State** | **Description** |
| Idle | The system is waiting for user input, such as browsing products or searching for specific items. |
| Browsing Products | The user is viewing the available product catalog, with various filters and categories. |
| Searching Products | The user has entered a search query, and the system is showing relevant results. |
| Adding to Wishlist | The user has added a product to their Wishlist for future reference. |
| Making a Purchase | The user has proceeded to checkout and is finalizing the payment process to complete their purchase. |

**Transitions:**

● Idle → Browsing Products: User begins browsing

● Idle → Searching Products: User searches for specific items

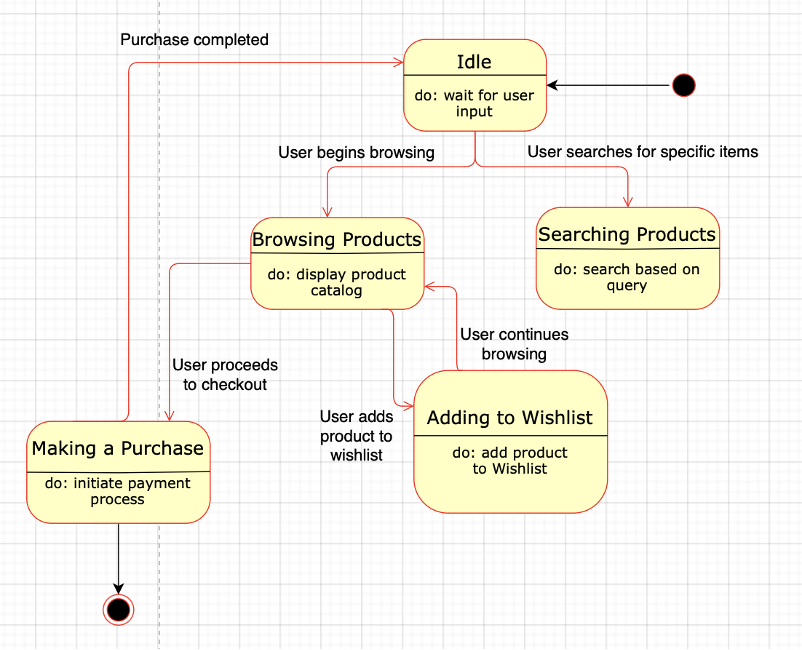
● Browsing Products → Adding to Wishlist: User adds product to wishlist

● Browsing Products → Making a Purchase: User proceeds to checkout

● Adding to Wishlist → Browsing Products: User continues browsing

● Making a Purchase → Idle: Purchase completed

**State Diagram**

****

**6.6 System Performance Monitoring and Error Handling**

|  |  |
| --- | --- |
| **State** | **Description** |
| Idle | System is functioning normally without issues. |
| Monitoring | System is tracking performance metrics such as CPU and memory. |
| Error Detected | System detects an issue such as slow response or failure. |
| Error Logging | System logs the error for diagnostics. |
| Alert Sent | System sends an alert to the admin about the detected error. |
| Issue Resolved | Issue is fixed, and the system resumes normal operation. |

**Transitions:**

● **Idle → Monitoring**: System starts tracking performance metrics.

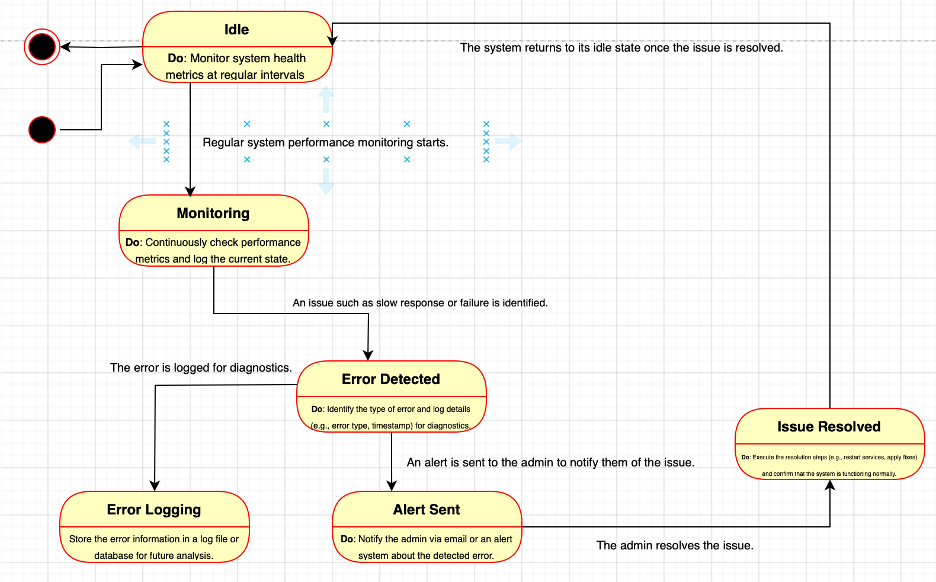
● **Monitoring → Error Detected**: An issue is identified during monitoring.

● **Error Detected → Error Logging**: The error is logged for diagnostics.

● **Error Detected → Alert Sent**: An alert is sent to the admin.

● **Alert Sent → Issue Resolved**: Admin resolves the detected issue.

● **Issue Resolved → Idle**: System returns to idle state after resolution

**State Diagram**  


**6.7 NLP and Machine Learning Interaction**

|  |  |
| --- | --- |
| **State** | **Description** |
| Idle | System is waiting for user input. |
| Query Received | User submits a query for processing. |
| Processing Query | System processes the natural language query using NLP techniques. |
| Fetching Data | System retrieves data relevant to the processed query. |
| Learning from Feedback | System refines recommendations based on user feedback (likes/dislikes). |
| Updating Model | System updates NLP models to improve future query responses. |

**Transitions:**

● **Idle → Query Received**: User submits a natural language query.

● **Query Received → Processing Query**: System begins processing the query.

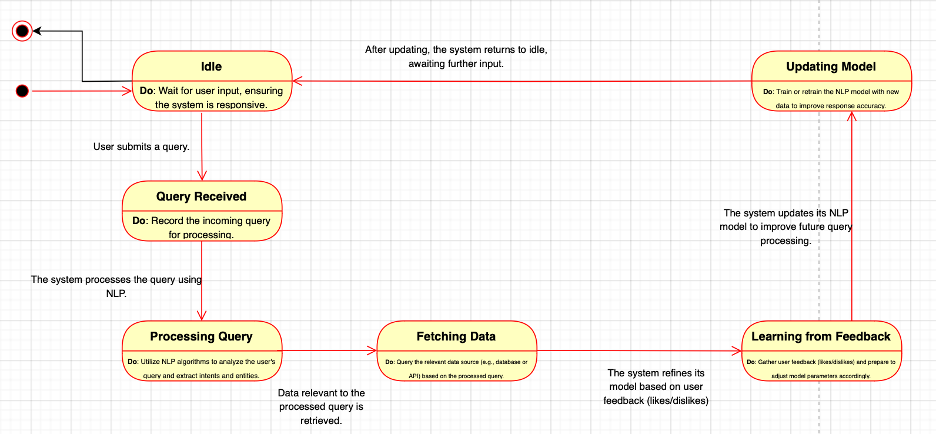
● **Processing Query → Fetching Data**: System retrieves relevant data based on the query.

● **Fetching Data → Learning from Feedback**: User provides feedback on the retrieved data.

● **Learning from Feedback → Updating Model**: System updates its model based on the feedback.

● **Updating Model → Idle**: System returns to idle state, ready for the next query.

**State Diagram**

**  
  
  
  
6.8** **Trend Data Gathering and Analysis**

|  |  |
| --- | --- |
| **State** | **Description** |
| Idle | System is waiting for data collection triggers. |
| Gathering Data | System collects data from sources like social media and user interactions. |
| Analyzing Data | System processes the data to identify patterns and trends. |
| Detecting Trends | System identifies specific trends, such as seasonal trends. |
| Suggesting Trends | System suggests trending products or preferences to users. |
| Learning from Engagement | System learns from user engagement with the suggested trends and updates its suggestions. |

**Transitions:**

● **Idle → Gathering Data**: System starts collecting trend-related data.

● **Gathering Data → Analyzing Data**: System processes the collected data.

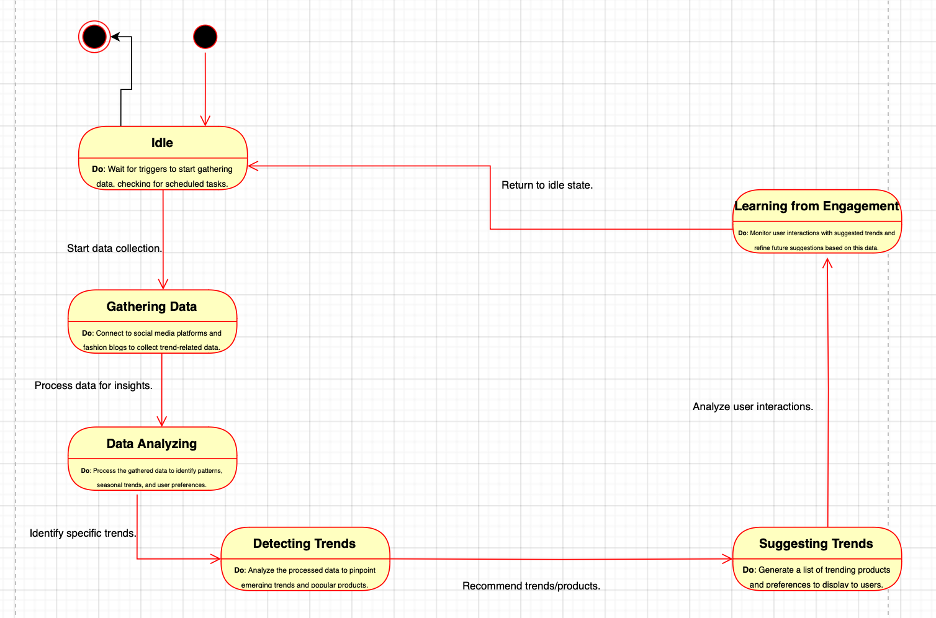
● **Analyzing Data → Detecting Trends**: System identifies specific trends in the data.

● **Detecting Trends → Suggesting Trends**: System recommends trends or products to users.

● **Suggesting Trends → Learning from Engagement**: System analyzes user interactions with suggestions.

● **Learning from Engagement → Idle**: System returns to idle state, ready for the next cycle.

**State Diagram**

****

# 

# 

# 7. Non-functional Requirements / Quality Attributes

|  |  |
| --- | --- |
| **Sr#** | **Requirements** |
| 1 | **Scalability:**  ● The system must be capable of handling at least 10,000 concurrent user interactions without degradation in performance. |
| 2 | **Response Time:**  ● The system should provide product recommendations within 2 seconds of receiving a user query. |
| 3 | **Data Storage:**  ● The system should be able to store and manage up to 1 TB of user and product data without performance impact. |
| 4 | **System Availability:**  ● The system must maintain 99.9% uptime, with scheduled maintenance windows not exceeding 1 hour per month. |
| 5 | **Error Handling:**  ● The system should have automated error logging and alerting mechanisms, and errors should be logged with a severity level and timestamp for analysis. |
| 6 | **Security:**  ● All user data must be encrypted using at least AES-256 encryption both at rest and in transit. |
| 7 | **Compliance:**  ● The system should comply with GDPR and CCPA regulations for data privacy and user consent. |
| 8 | **Integration Latency:**  ● The real-time integration with e-commerce platforms should have a latency of no more than 1 second for updating product availability and prices. |
| 9      10 | **Model Accuracy:**  ● The recommendation engine should achieve a precision score of at least 80% in product suggestions based on user interactions.      **NLP Accuracy:**  ● The NLP model should achieve an F1 score of at least 75% in understanding and processing user queries. |
| 11 | **User Interface Performance:**  ● The web application should load within 3 seconds on standard broadband connections. |
| 12 | **Data Update Frequency:**  ● The product data and trend information should be updated at least once every hour to reflect the latest information. |
| 13 | **Backup and Recovery:**  ● The system should perform automated backups every 24 hours and be able to restore to the last backup point within 30 minutes of a data loss incident. |
| 14       15 | **User Interaction Limits:**  ● The system should be capable of handling up to 1 million user queries and interactions per day.  **Resource Utilization:**  ● The application servers should not exceed 75% CPU utilization under peak load conditions. |
| 16 | **Compatibility:**  ● **Requirement:** The web application must be compatible with the latest versions of Chrome, Firefox, Safari, and Edge. |
| 17 | **Data Handling Capacity:**  ● Requirement: The system must be able to handle a minimum of 100,000 data transactions per hour without performance degradation. |
| 18 | **Load Distribution:**  ● Requirement: The load balancer should evenly distribute incoming requests across at least three application servers. |
| 19 | **High Availability:**  ● Requirement: The system must maintain high availability with no more than 5 minutes of downtime during a load balancer failure. |

# 8. Who Did What?

|  |  |
| --- | --- |
| **Name of the Team Member** | **Tasks done** |
| Ahmad Kashif Jabbar | Use cases and Diagrams |
| Musa Aftab | Class Diagrams |
| Zainab Fatima | Sequence Diagrams |
| Husnain Ali and Messam Ali | State Diagrams |
|  |  |

# 9. Review checklist

Before submission of this deliverable, the team must perform an internal review. Each team member will review one or more sections of the deliverable.

|  |  |
| --- | --- |
| **Section** **Title** | **Reviewer Name(s)** |
| Class Diagrams | Zainab Fatima |
| Sequence Diagrams | Ahmad Jabbar |
| Use cases and Diagrams | Husnain Ali and Messam |
| State Diagrams | Musa Aftab, Zainab Fatima |