**Khas**

|  |
| --- |
| **Riphah** |

**By:**

**Syed Mushahid hussain**

**13629**

**Usama shah**

**13784**

**Talha khizar**

**13676**

**Supervised by:**

**Mr. Osamah ahmad**

**Faculty of Computing**

**Riphah International University, Islamabad**

**Spring 2023**

**A Dissertation Submitted To**

**Faculty of Computing,**

**Riphah International University, Islamabad**

**As a Partial Fulfillment of the Requirement for the Award of the Degree of**

**Bachelors of Science in Software Engineering**

**Faculty of Computing**

**Riphah International University, Islamabad**

Date: 29th May 2023

**Final Approval**

This is to certify that we have read the report submitted by ***Syed Mushahid Hussain(13629), Usama shah(13784), Talha Khizar(13676)***, for the partial fulfillment of the requirements for the degree of the Bachelors of Science in Software Engineering (BSSE). It is our judgment that this report is of sufficient standard to warrant its acceptance by Riphah International University, Islamabad for the degree of Bachelors of Science in Software Engineering (BSSE).

**Committee:**

|  |  |
| --- | --- |
| **1** | Mr. Osamah Ahmad  (Supervisor) |
|  |  |
| **2** | Dr. Musharraf Ahmad  (Head of Department/chairman) |

**Declaration**

We hereby declare that this document “**Khas**” neither as a whole nor as a part has been copied out from any source. It is further declared that we have done this project with the accompanied report entirely on the basis of our personal efforts, under the proficient guidance of our teachers especially our supervisor **Mr. Osamah Ahmad**. If any part of the system is proved to be copied out from any source or found to be reproduction of any project from anywhere else, we shall stand by the consequences.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Syed Mushahid Hussain**

**13629**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Usama shah**

**13784**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Talha khizar**

**13676**

**Dedication**

We dedicate our work to our family, friends and teachers. The unrivalled encouragement from our parents and outstanding support from teachers is what lead to success of this project. We also dedicate our work to our supervisor Mr. Osamah Ahmad and the faculty members.

**Acknowledgement**

First of all we are obliged to Allah Almighty the Merciful, the Beneficent and the source of all Knowledge, for granting us the courage and knowledge to complete this Project.

We are grateful to our parents for the love and encouragement they provided us throughout the project. To our supervisor Mr. Osamah Ahmad, we are highly obliged for his considerations, the highly refined and sophisticated atmosphere he extended to us. Also, a special thanks to the rest of the other faculty members for their unconditional support.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Syed Mushahid Hussain**

**13629**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Usama shah**

**13784**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Talha khizar**

**13676**

**Abstract**

Introducing Khas, an online marketplace for buying and selling used and new mobile phones. Our goal is to build trust in online shopping by providing a safe and secure platform for users to buy and sell mobile phones. We offer users a platform where they can post ads about their phones with full specifications, and other users can purchase these phones safely through our secure payment system. This makes it easy for them to sell their phones online. As far as payment is concerned, we will hold the payment until our team has verified the phone and sent it to the purchaser. Khas makes buying and selling mobile phones easy and worry-free. With Khas, users can find great deals on mobile phones while also having peace of mind that your transaction is safe and secure.

**Table of Contents**

[**Chapter 1: Introduction** 14](#_Toc140786689)

[**1.1** **Opportunity and Stakeholders** 15](#_Toc140786690)

[**1.1.1** **Opportunity** 15](#_Toc140786691)

[**1.1.2** **Stakeholders** 16](#_Toc140786692)

[**1.2** **Motivations and Challenges** 16](#_Toc140786693)

[**1.2.1** **Motivations** 16](#_Toc140786694)

[**1.2.2** **Challenges** 17](#_Toc140786695)

[**1.3** **Goals and Objectives** 17](#_Toc140786696)

[**1.4** **Solution Overview** 17](#_Toc140786697)

[**1.5** **Report Outline** 18](#_Toc140786698)

[**Chapter 2: Literature/Market Survey** 20](#_Toc140786699)

[**2.1 Introduction** 21](#_Toc140786700)

[**2.2 Literature Review** 21](#_Toc140786701)

[**2.2 Summary** 22](#_Toc140786702)

[**Chapter 3: Requirement Engineering** 23](#_Toc140786703)

[**3.1 Introduction** 24](#_Toc140786704)

[**3.2 Problem Scenarios** 24](#_Toc140786705)

[**3.3 Functional Requirements** 25](#_Toc140786706)

[**3.4 Non-Functional Requirements** 43](#_Toc140786707)

[**Chapter 4: System Design** 44](#_Toc140786708)

[**4.1 Introduction** 45](#_Toc140786709)

[**4.2 Architectural Design** 46](#_Toc140786710)

[**4.2.1 MVC Architectural Diagram** 46](#_Toc140786711)

[**4.3 Detailed Design** 49](#_Toc140786717)

[**4.3.1** **Use Case Diagrams** 49](#_Toc140786718)

[**4.3.2** **Detailed Use Cases** 57](#_Toc140786719)

[**4.3.3 Activity Diagrams** 90](#_Toc140786720)

[**4.3.4 Test Case Design** 122](#_Toc140786721)

[**Chapter 5: Implementation** 158](#_Toc140786722)

[**5.1 Endeavour (Team + Work + Way of Working)** 159](#_Toc140786723)

[**5.2 Flow Control/Pseudo codes** 159](#_Toc140786724)

[**5.3 Components, Libraries, Web Services and stubs** 163](#_Toc140786725)

[**5.4 IDE, Tools and Technologies** 163](#_Toc140786726)

[**5.5 Best Practices / Coding Standards** 163](#_Toc140786727)

[**5.5.1 Software Engineering Practices** 163](#_Toc140786728)

[**5.5.2 Development Practices & Standards** 163](#_Toc140786729)

[**5.6 Deployment Environment** 164](#_Toc140786730)

[**5.8 Summary** 164](#_Toc140786733)

[**Chapter 6: Conclusion and Outlook** 165](#_Toc140786734)

[**6.1 Introduction** 166](#_Toc140786735)

[**6.2 Achievements and Improvements** 166](#_Toc140786736)

[**6.3 Critical Review** 166](#_Toc140786737)

[**6.4 Future Recommendations/Outlook** 167](#_Toc140786738)

[**6.5 Summary** 167](#_Toc140786739)

**List of Figures**

|  |  |
| --- | --- |
| 1.1 Caption of first figure of first chapter | 6 |
| 1.2 Caption of second figure of first chapter | 7 |
| 2.1 Caption of first figure of second chapter | 14 |
| 2.2 Caption of second figure of second chapter | 22 |
| 2.3 Caption of third figure of second chapter | 26 |
| 5.1 Caption of first figure of fifth chapter | 49 |
| 5.2 Caption of second figure of fifth chapter | 49 |

**List of Tables**

|  |  |
| --- | --- |
| 1.1 label of first table of first chapter | 6 |
| 1.2 label of second table of first chapter | 7 |
| 2.1 label of first table of second chapter | 14 |
| 2.2 label of second table of second chapter | 22 |
| 2.3 label of third table of second chapter | 26 |
| 5.1 label of first table of fifth chapter | 49 |
| 5.2 label of second table of fifth chapter | 49 |

# **Chapter 1: Introduction**

**Chapter 1:**

**Introduction**

This chapter focuses on background of Khas. It also discusses the challenges handled, goals and objectives of this project.

The first chapter will give a deep knowledge of our project by giving additional analyses of the idea. We'll start by talking about the opportunity and stakeholders. The motivational and challenges aspects of this project will next be explored, along with some of the problems we may face as we put that into practice. The rest of this chapter will then focus on the project's goals and objectives. The report's content outline will then be explained, followed by a discussion of the suggested solution.

## **Opportunity and Stakeholders**

### **Opportunity**

1. **Large market:** The market for buying and selling mobile phones is huge, with a high demand for both used and new phones. Khas can tap into this market by providing a platform for users to buy and sell mobile phones easily and safely.
2. **Convenience:** Khas can provide a convenient solution for people looking to buy or sell mobile phones. By providing an online platform, Khas makes it easy for users to find the phone they want and make a purchase without having to leave their homes.
3. **Trust and safety:** By providing a platform that is safe and secure, Khas can build trust in online shopping and remove the risk of scams.
4. **Increased Efficiency:** Khas can help increase efficiency by providing a centralized platform for buying and selling mobile phones. This will save time and money for both buyers and sellers.
5. **Cost savings:** Users can find great deals on mobile phones as they can compare prices and features from multiple sellers on Khas.

### **Stakeholders**

1. **Users:** Users of Khas will be both buyers and sellers of mobile phones. They will benefit from the convenience and safety provided by the platform.
2. **Team:** The team behind Khas will be responsible for the development and maintenance of the platform. They will also verify the phones before releasing the payment.
3. **Investors:** Investors will be interested in Khas as it has the potential to tap into a large market and provide a valuable service to users.
4. **Payment providers:** Payment providers will be stakeholders in Khas as they will be responsible for processing the payments made on the platform.
5. **Shipping Companies:** They will be the stakeholders as they will be responsible for shipping the phone after the verification process.

## **Motivations and Challenges**

### **Motivations**

1. Our mission is to provide an online marketplace where consumers can buy and sell mobile phones safely and securely.
2. Making buying and selling mobile phones easy and convenient for users.
3. To help users find great deals on mobile phones.
4. To build trust in online shopping.
5. To eliminate the risk of scams when buying and selling mobile phones.
6. To provide a central platform for buying and selling mobile phones.
7. To increase efficiency and save time and money for both buyers and sellers.
8. To expand the market nationally.
9. To diversify its services to increase revenue.
10. To provide personalization options to make search results more relevant for users.

### **Challenges**

1. Establishing user credibility and trust
2. Maintaining the security of user financial and personal information
3. Staying up-to-date with the latest technologies and trends in the mobile phone industry
4. Before releasing payment, verifying the phone's authenticity and condition
5. Maintaining a large amount of data and information on the platform
6. Customer support and platform maintenance
7. Maintaining an easy-to-navigate, user-friendly platform
8. Handling the logistics of shipping and handling the phones
9. Dealing with disputes and issues that may arise during the sale and purchase process
10. Offering unique services and features to keep the platform competitive.

## **Goals and Objectives**

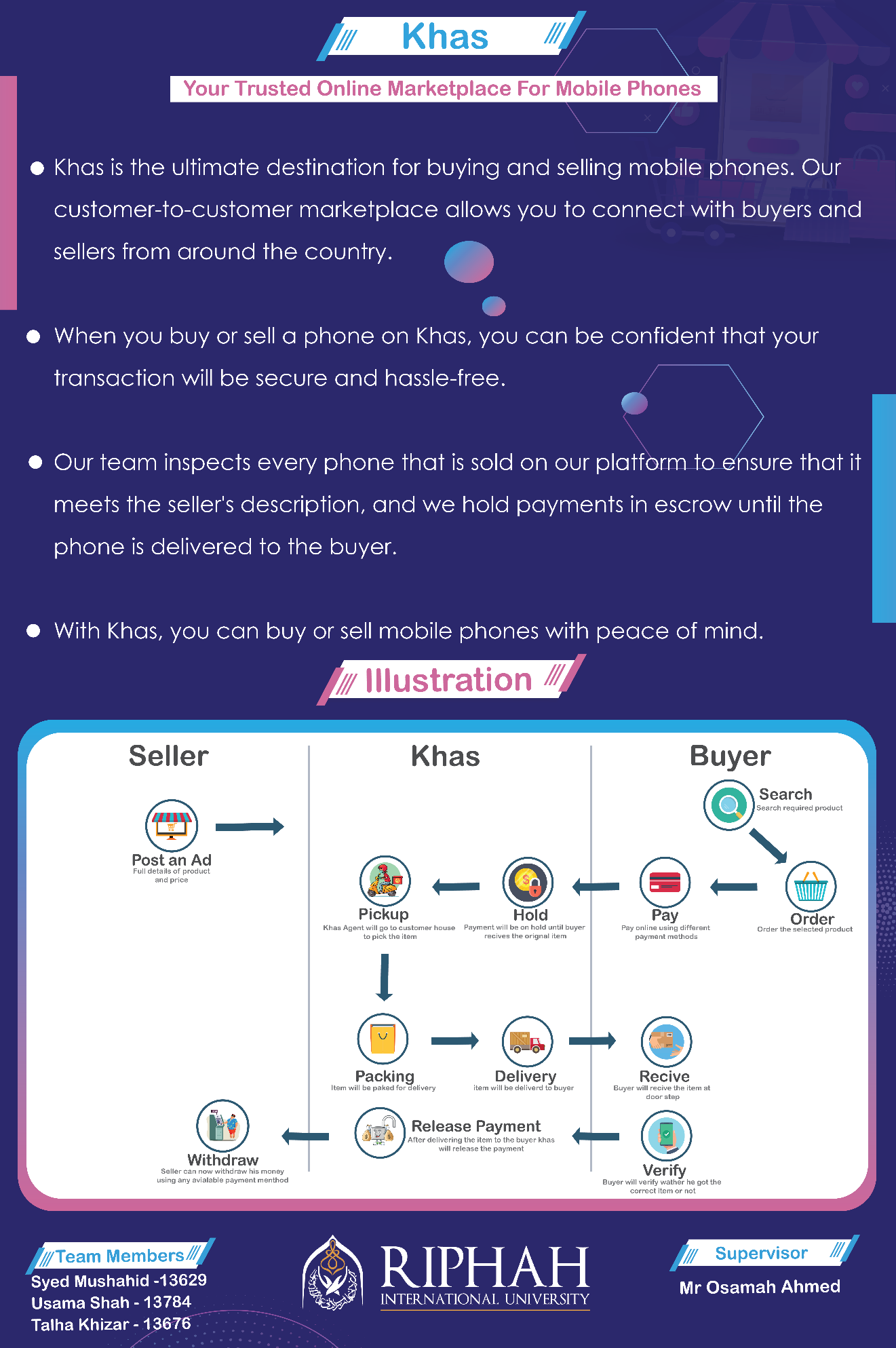
The main goal of developing this application is to make it easier for people to buy or sell products online in the C2C market. And easily buy or sell verified products to each other.

The following list includes the main aims and objectives:

1. To establish Khas as a leading online marketplace for buying and selling mobile phones.
2. To establish Khas as a brand known for trust, safety and reliability in the online mobile phone market.
3. To provide a safe and secure platform for users to buy and sell mobile phones.
4. To build trust and credibility among users and remove the risk of scams.
5. To make it easy and convenient for users to buy and sell mobile phones.
6. To expand the market to national level.

## **Solution Overview**

A seller can post an ad about their product on the website (Khas) for which he or she must provide full details and specifications. Buyers can search and select products. He will pay online on our website (Khas) to order the product. Once the buyer places an order the seller will have to drop the product off at our nearest office. The inspector will check the product specifications and condition and compare them with the details which were posted online by the seller. The payment which was paid by the customer will not be released until the buyer receives and verifies the product. Once the buyer receives the product and confirms it, we will release the payment to the seller and he can withdraw that money. The buyer will be given a refund if the product specification is not the same as the details posted online, and the product will be returned to the seller. This will reduce the chances of scams and frauds and save users' time by allowing them to shop online while at home.



## **Report Outline**

In this chapter, we have introduced the main idea of our project, clarified its levels and ideas, and discussed the opportunities and user, stakeholders. We have also further dwelled on the purposes for our motivation in creating this project, as well as the challenges we will faced and how we plan to overcome them. Finally, we have reviewed and clarified the solution to enable this platform. This report will cover a literature review and market surveys in the upcoming chapters, where a detailed analysis of the currently available platform will be made. We will then attempt to establish the case that (Khas) is, in many ways, superior to the market's current solutions. Additionally, we will elaborate on the functional and non-functional postulated requirements of the suggested solution. The discussion of various technical diagrams and artefacts will then shift to system design. All of the difficulties to the execution of this project will be covered in the implementation chapter.

# **Chapter 2: Literature/Market Survey**

## **2.1 Introduction**

The best C2C systems currently on the market will be examined in this chapter. Following that, we will compare our suggested solution to the systems currently on the market to determine how well it addresses all the gaps and flaws in those systems. Additionally, by examining various existing systems, this section offers a deep insight into the market survey conducted for the Khas development team.

## **2.2 Literature Review**

With the help of the literature review and market analysis, we were able to identify some operational C2C systems.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | **Olx** | **Facebook** | **PakWheel** | **Zameen.pk** | **Baichday.com.pk** | **eBay** |
| **Delivery Service** | **No** | **No** | **No** | **No** | **No** | **Yes** |
| **Payment Gateway** | **No** | **No** | **No** | **No** | **No** | **Yes** |
| **Refund/Return** | **No** | **No** | **No** | **No** | **No** | **Yes** |
| **Chat** | **Yes** | **Yes** | **Yes** | **Yes(external)** | **No** | **Yes** |
| **Premium Ads** | **Yes** | **Yes** | **Yes** | **Yes** | **No** | **Yes** |
| **Product Categories** | **All** | **All** | **Vehicles** | **Real estate** | **All** | **Multiple** |
| **Review/Comment** | **No** | **Yes** | **Yes(own rating)** | **No** | **Yes** | **Yes** |
| **Location Based Products** | **Yes** | **Yes** | **Yes** | **Yes** | **Yes** | **Yes** |
| **Verification/ Inspection** | **No** | **No** | **Yes(on request)** | **No** | **No** | **No** |
| **Working in Pakistan** | **Yes** | **Yes** | **Yes** | **Yes** | **Yes** | **No** |
| **Search/Filter Products** | **Yes** | **Yes** | **Yes** | **Yes** | **Yes** | **Yes** |
| **Cart** | **No** | **No** | **No** | **No** | **No** | **No** |
| **Add to Favorite** | **Yes** | **Yes** | **Yes** | **No** | **No** | **Yes** |
| **Payment Method** | **None** | **None** | **Bank Plans** | **Multiple Pans** | **None** | **Paypal**  **Cards**  **Google/Apple pay** |
| **Detail Specification Gathering** | **No** | **No** | **Yes** | **Yes** | **No** | **Yes** |
| **Offers** | **No** | **Yes** | **No** | **No** | **No** | **Yes** |

## **2.2 Summary**

In summary, Khas is an online marketplace for buying and selling used and new mobile phones. It provides a safe and secure platform for users to buy and sell mobile phones easily and safely. Users can post ads about their phones with full specifications, and other users can purchase these phones online through our secure payment system. Khas addresses the problem of trust and safety in online shopping by providing a platform that is safe and secure, it also addresses the problem of scams by verifying the authenticity and condition of the phones before releasing the payment. Additionally, Khas addresses the problem of convenience by providing an online platform for buying and selling mobile phones which eliminates the need for users to leave their homes. The platform is user-friendly, easy to navigate and provides personalization options.

# **Chapter 3: Requirement Engineering**

## **3.1 Introduction**

|  |
| --- |
| Khas is an online marketplace for buying and selling used and new mobile phones. It aims to provide a safe and secure platform for users to buy and sell mobile phones easily and safely. As a requirement engineering process, Khas team will be responsible for gathering, analyzing, specifying and validating the needs of the stakeholders and end-users. The goal of the requirement engineering process is to ensure that the final product meets the needs of the users and stakeholders while also being technically feasible and economically viable. This process is crucial for the success of the project as it sets the foundation for the design, development, and testing of the platform. Throughout the development process, the requirements will be continuously evaluated and refined to ensure they are aligned with the project goals and objectives. |
| **3.2 Problem Scenarios** |
| Ali is a college student and he is looking to buy a used mobile phone as he is on a tight budget. He found a phone on a classified website that meets his requirements and budget but he is worried about the authenticity of the phone and the risk of scams. He doesn't want to spend his money on a phone that might turn out to be a fake or in bad condition. He also doesn't want to meet the seller in person as he is concerned about his safety.  Ali faces the following problems:   1. Lack of trust and safety in buying a used mobile phone from a classified website. 2. Risk of scams and frauds. 3. Inconvenience of meeting the seller in person. 4. Lack of assurance about the authenticity and condition of the phone.   Khas can solve these problems by providing a safe and secure platform for buying and selling mobile phones online. Users can post ads about their phones with full specifications and other users can purchase these phones online through our secure payment system. We hold the payment until the phone is verified by our team and sent to the buyer. Khas also provides an option to filter the search results according to the user's requirements, such price range, brand, model and condition of the phone. With Khas, Ali can buy a used mobile phone with peace of mind knowing that he is protected from scams and frauds, and that the phone he is buying is authentic and in good condition. **3.3 Functional Requirements** **For Users Panel** |
| **3.3.1 Registration**  **3.3.1.1 Login**   * 1. The user should be able to enter their username/email and password to log in to the system.   2. The user's credentials should be verified against a database to ensure that they are valid.   3. The user should be able to request a password reset if they have forgotten their password.   4. The login process should support multiple authentication methods, such as password-based login or login using a third-party service such as Google or Facebook.   5. The login process should support secure authentication methods, such as two-factor authentication.   6. The login process should support session management, allowing the user to remain logged in until they log out or their session expires.   7. The login process should block or limit the number of login attempts by a user to prevent brute-force attacks.   8. The login process should provide the user with feedback if their login attempt was unsuccessful, such as an error message indicating an incorrect username or password.   **3.3.1.2 Signup**   * 1. The user should be able to enter their personal and login information to create a new account.   2. The user's information should be validated to ensure that it is complete and accurate.   3. The user's password should be encrypted and stored securely in a database.   4. The user should be able to choose a unique username for their account.   5. The user should be able to enter their email address and receive a verification email to confirm their account.   6. The signup process should enforce password strength requirements to ensure that user passwords are secure.   7. The signup process should provide the user with feedback if their account creation was unsuccessful, such as an error message indicating that their chosen username is already in use.   **3.3.1.3 Forgot Password**   * 1. The user should be able to enter their email address to initiate a password reset process.   2. The user's email address should be verified against a database to ensure that it is registered with the system.   3. The system should send a password reset email to the user's email address, containing a link or code to reset their password.   4. The user should be able to enter a new password and confirm it to complete the password reset process.   5. The user's new password should be encrypted and stored securely in a database.   6. The password reset process should enforce password strength requirements while setting new password to ensure that user passwords are secure.   7. The password reset process should provide the user with feedback if their request was unsuccessful, such as an error message indicating that their email address is not registered with the system.   8. The password reset process should allow the user to choose a new password that is different from their previous password.   9. The password reset process should expire after a certain amount of time to prevent unauthorized access to the user's account.      + 1. **Email Verification**  1. The user should be able to enter their email address during the account registration process. 2. The system should send a verification email to the user's email address, containing a link or code to verify their email address. 3. The user should be able to click the link or enter the code in the verification page to complete the email verification process. 4. The system should verify the link or code to ensure that it is valid and matches the user's email address. 5. If the link or code is invalid or does not match the user's email address, the system should display an error message indicating that the email verification has failed. 6. If the email verification is successful, the system should mark the user's email address as verified in the database. 7. The email verification process should expire after a certain amount of time to prevent unauthorized access to the user's account. 8. The email verification process should allow the user to request a new verification email if they did not receive the original email or if the original email has expired.    1. **Home**       * + 1. The home page should display featured products or categories to highlight popular or new items.           2. The home page should provide a search bar for users to easily find specific products.           3. The home page should display information about the website, such as the company's mission, values, or policies.           4. The home page should display customer service information, such as contact details or links to frequently asked questions.           5. The home page should display links to the website's social media pages or other external resources.           6. The home page should display a navigation menu to allow users to browse different categories of products.           7. The home page should display user account options, such as login, sign up, or profile           8. The home page should display a shopping cart summary, including the number of items and total cost.           9. The home page should display promotional offers or discounts.       1. **Other Pages**   These includes pages like about us, terms and conditions, privacy policy user guides and other informational pages.   * + 1. **Product**  1. The product page should display the product's name, image, price, and availability. 2. The product page should display a detailed description of the product, including features, specifications, conditions and instructions. 3. The product page should display customer reviews and ratings for the seller. 4. The product page should provide a button for users to add the product to their shopping cart. 5. The product page should display related or recommended products. 6. The product page should provide a way for users to share the product on social media or email. 7. The product page should display the shipping and return policies for the product. 8. The product page should provide a way for users to ask questions or request more information about the product. 9. Buyers should be able to send custom price offers to sellers.    * 1. **Product Compare** 10. The module should allow users to select multiple products and add them to a comparison list. 11. The module should display a side-by-side comparison of the selected products, including their price, features, and specifications.     * 1. **Search Filter** 12. The module should allow users to enter a search query and retrieve relevant products from the database. 13. The module should allow users to filter the search results by various criteria, such as price range, brand, color, size, conditions or product category. 14. The module should display the search results in a clear and organized way, with relevant information about each product such as price, ratings, and reviews. 15. Products should have an option to add in cart. 16. User should be able to view the details of the product. 17. The module should allow users to sort the search results by various criteria, such as price, relevance, or popularity.     * 1. **Categories/Brands Page** 18. In this module user will see all products under specific category or brand. 19. It should display al products in list. 20. Show thumbnail and title of the product. 21. Show small description, category , seller ratings ,price and tags of the product. 22. Should have an option to add product in cart. 23. Should have an option to view the details of the product.     * 1. **Cart** 24. The cart should be able to store items that are added by the user. 25. The user should be able to view the items in their cart at any time. 26. The user should be able to remove items from their cart. 27. The user should be able to update the quantity of an item in their cart. 28. The user should be able to clear their entire cart. 29. The cart should display the total cost of all items in the cart, including any applicable discounts or promotions. 30. The user should be able to proceed to checkout and place an order for the items in their cart.     * 1. **Checkout** 31. The checkout process should allow the user to review the items in their cart and the total cost of their order. 32. The user should be able to select a shipping address for their order. 33. The user should be able to select a shipping method (company) for their order. 34. The user should be able to select a payment method for their order. 35. The checkout process should validate the user's payment information to ensure that it is valid before allowing the user to place an order. 36. The user should be able to review and confirm the details of their order before placing it. 37. The checkout process should provide the user with confirmation of their order, including a receipt or invoice. 38. The checkout process should store the details of the order in a database. 39. The checkout process should send an email confirmation to the user's email address. 40. The checkout process should update the inventory levels or product status in the system to reflect the items that were purchased.     * 1. **User Dashboard**   The customer dashboard module should display overview of all other modules and links to those modules.  **3.3.11 User Profile**  It should display the information of user such as name ,email profile picture.  **3.3.11.1 Profile Settings**   1. The user should be able to update their account information, including their personal information, contact details, shipping information, and payment preferences. 2. The user should be able to update their password.   **3.3.11.2 Logout**   1. Upon logging out, the website should clear any session data associated with the user's account to prevent unauthorized access. 2. After logging out, the user should be redirected to the login page or a relevant landing page. 3. The logout process should be persistent, meaning that the user will stay logged out even if they close the browser or navigate to a different page.   **3.3.12 Wallet**   1. The user should be able to view their current wallet balance and transaction history. 2. The user should be able to use their wallet balance to make payments on the website. 3. The user should be able to add funds to their wallet using given payment methods. 4. The user should be able to withdraw funds from their wallet to their linked bank account or other designated payment method. 5. The user should receive notifications for wallet transactions, including balance updates and payment confirmations. 6. The user wallet should be secured with appropriate measures to protect against unauthorized access and fraud.   **3.3.13 Products Management**   1. The module should allow users to create new product listings, including the ability to upload product images, set pricing, and add product descriptions and other specifications. 2. The module should allow users to control the visibility of their products on the website, including the ability to make products visible or hidden to users. 3. The module should allow users to set the availability of their products, including the ability to mark products as out of stock or unavailable for purchase. 4. The module should allow users to update existing product listings, including the ability to modify product details, images, and pricing and specification. But if the product is sold and customer had made the payment then user should not be able to update the information. 5. Keep the record of old data of product when user updates it. 6. The module should allow users to delete product listings that are no longer needed or relevant. 7. The module should allow users to search for products using various criteria, such as keywords, categories, and price range.   **3.3.14 Sales**   1. The sales module should display list of all products which was sold by the user. 2. User can only view the product. 3. User should not be able to modify it or delete the products from this list. 4. This module should display product title and price and sale status i.e pending ,completed ,rejected e.t.c. in list 5. When user click on the product it should display all details about the product like product specifications, images, price, buyer, seller details purchase recipte details. 6. Recent Purchases should be displayed on top. 7. Product Search filter should be available to filter the data in the list.   **3.3.15 Purchases**   1. The sales module should display list of all products which was purchased by the user. 2. User can only view the product. 3. User should not be able to modify it or delete the products from this list. 4. This module should display product title and price and sale status i.e pending ,completed ,rejected e.t.c. in list 5. When user click on the product it should display all details about the product like product specifications, images, price, buyer, seller details purchase recipte details. 6. Recent Purchases should be displayed on top. 7. Product Search filter should be available to filter the data in the list.   **3.3.16 Sent Offers**   1. The offers modules should display all offers sent buy the user or received by the users. 2. Recent offers should be displayed on top. 3. When user gets new offer he should get notification and email about the offer. 4. When user click on the offer it should redirect to product page. 5. User should be able to accept or reject the offer. 6. Sender should get a notification and email about the offer status on rejection or approval. 7. If seller accepts the offer then buyer should be able to purchase the product with that price.   **3.3.17 Notifications**   1. The notification module should display all new notifications on top. 2. Unread notification counts should be displayed on top. 3. Notification status i.e read or unread should be displayed with notification. 4. New notification should be displayed in alert box on side of screen if user is on website. 5. Notification bar should display a title of notification and time.   **3.3.18 Inspection Report**   1. When inspector will fill the form of inspection from admin panel result of that inspection should be shown in this module to both seller and buyer. 2. User should not be allowed to modify or delete this report.   **3.3.19 Messages**   1. All users should be able to send messages to each other. 2. Messages with different users should be kept as different chat. 3. All previous chats should be available. 4. User can delete any message and chat.   **3.3.20 Customer Support**   1. The customer support module should provide a comprehensive knowledge base or FAQ section to help customers resolve common issues on their own. 2. The customer support module should allow customers to submit support requests. 3. Customer support should also keep previous chats and messages in database.   **3.3.21 Riders Module**  **3.3.21.1 Orders**   1. The order module should allow delivery riders to view and accept new delivery orders. 2. The order module should provide delivery riders with real-time updates on the status of their orders, such as pickup and delivery locations, as well as any special instructions. 3. The order module should provide delivery riders with the ability to track their delivery routes and estimate the time of arrival at the destination. 4. The order module should allow delivery riders to enter and confirm the delivery of orders 5. The order module should have the ability to track and report on key metrics, such as delivery times and completion rates. 6. Riders should be able to accept or reject any new delivery. 7. When rider complete any delivery his commission and earning should be updated in database. 8. Riders should be able to see their total earnings and the history of orders done by them and their status.   **For Admin**  **Login**  1. Get email and password from user.  2. Validate the email and password if it is wrong display a error message.  **Dashboard**   1. The admin panel dashboard should allow the administrator to view a summary of key metrics and statistics, such as the number of registered users, active users, and sales. 2. The dashboard should provide the administrator with the ability to manage and update user accounts, including the ability to add, delete, or modify user permissions. 3. The dashboard should allow the administrator to view and manage orders, including the ability to view order details, update order status, and generate reports on order data. 4. The dashboard should provide the administrator with the ability to manage and update the company's products and inventory, including the ability to add, delete, or modify product information. 5. The dashboard should allow the administrator to view and manage customer support requests, including the ability to view request details, update request status, and generate reports on customer support data. 6. The dashboard should provide the administrator with the ability to manage and update the company's financial data, including the ability to view financial reports, update billing information, and process payments. 7. The dashboard should provide the administrator with the ability to customize the appearance and functionality of the admin panel, including the ability to add or remove features and change the layout and design.   **Users Management**  **Add user**   1. The system should allow the administrator to input the user's personal information, including name, email, and password. 2. The system should validate the email address to ensure it is in the correct format. 3. The system should enforce password complexity requirements, such as a minimum length and the inclusion of at least one special character. 4. The system should store the user's personal information in a secure database. 5. The system should auto assign a role user to this new member. 6. The system should send a confirmation email to the user's email address with login details. 7. The system should ensure that only authorized administrators have the ability to add new users to the system.   **User List**   1. The system should allow the administrator to view a list of all users in the system. 2. The system should display each user's personal information, such as name and email address. 3. The system should allow the administrator to filter the list of users by various criteria, such as role or status. 4. The system should allow the administrator to sort the list of users by different fields, such as name or email address. 5. The system should allow the administrator to view the details of a specific user, including their personal information and role within the system. 6. The system should allow the administrator to edit a user's personal information. 7. The system should allow the administrator to delete a user from the system.   **Products**  **Product List**   1. The system should allow the administrator to view a list of all products in the system. 2. The system should display each product's information, such as name, description, and price. 3. The system should allow the administrator to filter the list of products by various criteria, such as category or price range. 4. The system should allow the administrator to sort the list of products by different fields, such as name or price. 5. The system should allow the administrator to view the details of a specific product, including its name, description, price, and any relevant images or documents. 6. The system should allow the administrator to edit a product's information or delete it from the system.   **Removed Products**   1. The system should allow the administrator to view a list of all products that have been removed from the system. 2. The system should display each removed product's information, such as name, description, and price. 3. The system should allow the administrator to filter the list of removed products by various criteria, such as category or price range. 4. The system should allow the administrator to sort the list of removed products by different fields, such as name or price. 5. The system should allow the administrator to view the details of a specific removed product, including its name, description, price, and any relevant images or documents. 6. The system should allow the administrator to restore a removed product back to the active product list.   **Manage Advertisement**  **Boost Products**   1. Admin should be able to add new boost products to the system by providing necessary details such as Product name, Boost from, Boost to, and price. 2. Admin should be able to edit existing boost products by modifying the details of the product. 3. Admin should be able to deactivate a boost product if it is no longer available for purchase or if it needs to be removed from the system for any other reason. 4. Admin should be able to set up promotions or discounts for boost products by specifying the terms of the promotion and the applicable time period. 5. User Should get a notification when his product is boosted.   **Current ads**   1. Admin should be able to view a list of all current ads in the system and filter or sort the list by various criteria such as Product name, charges, and time period. 2. Admin should be able to edit existing ad campaigns by modifying the details of the campaign or updating the assets associated with the campaign. 3. Users and Admin should be able to generate reports on ad campaign performance and use these reports to inform future marketing strategy and decision-making.   **Manage Categories**   1. The administrator should be able to enter a new category name and any relevant details, such as a description or image. 2. The system should validate the category name to ensure it is unique and meets any prescribed formatting requirements. 3. The system should allow the administrator to specify any parent categories or subcategories for the new category. 4. The system should store the new category in the database and update any relevant listings or menus to reflect the new category. 5. Display confirmation that the category has been successfully added when category is added in database. 6. The system should provide an option for the administrator to view, edit, or delete the newly added category.   **Manage Brands**   1. The administrator should be able to enter a new brand name and any relevant details. 2. The administrator should be able to view, edit, or delete any of the listed brands.   **Manage Inspection**   1. The administrator should be able to select the product that the inspection is being performed on from a list of options. 2. The administrator should fill the inspection form according to the condition of a product. 3. The inspection report must be able to accurately reflect the condition of the inspected product. 4. The report must be stored securely and made available for future reference if needed. 5. The report should be send to both buyer and seller. 6. The system should also save the name of inspector in database with the inspection report.   **Sales**   1. The sales report should allow the admin to view a list of all sales in the system. 2. The admin should be able to filter the report by sale date, customer name, or product name. 3. The report should display the total number of sales in the list and the total value of all sales in the list. 4. The system should allow the admin to view a list of all pending sales in the system. 5. The system should allow the admin to view a list of all completed sales, including details such as the date of the sale, the customer's name, and the total amount of the sale.   **Finance**  **Transaction Ledger**   1. The system should allow the admin to view a list of all transactions, including details such as the date of the transaction, the type of transaction (e.g. sale, refund, payment), and the amount of the transaction. 2. The system should allow the admin to view detailed information about each transaction, including any associated customer or seller information, as well as date and amount   **Wallets**   1. The system should allow the admin to view a list of all wallets, including details such as the username, amount and date of last transaction. 2. The administrator should be able to create, edit, and delete user wallets.   **Charges and Discounts**   1. Admin should be able to add any additional charges like txt e.t.c from here. 2. Admin can also add any amount of discounts from here. 3. The amount added by admin from here should be reflect in the total cost of the products when the user will checkout.   **Withdraw Requests, Withdraw History**   1. The system should allow the admin to search for withdraw requests based on various criteria, such as the date of the request, amount and username. 2. The system should allow the admin to view a list of all completed withdraws, including details such as the date of withdraw and the amount. 3. The administrator should be able accept the withdraw request if it is a valid request. 4. When accepts the withdraw request the admin or the system should send the withdraw amount to the users given bank account. 5. On successful withdraw the system should update the users wallet and send notification and email to the user informing about the withdraw.   **Pending Refunds, Approved Refunds**   1. The system should allow the admin to view a list of all pending refund requests, including details such as the date refunded, the username, amount and the product against which the refund is requested. 2. The system should allow the admin to view a list of all approved refund requests, including details such as the date refunded, the username, and the amount. 3. The administrator should be able approve or reject the pending refund requests. 4. The system should send email and notification to user about the request result depending on the admins selection. 5. If the admin selects the refund he should make sure that he has received the product against which the refund is requested. 6. System should also inform the seller about this refund. 7. If the system has released the payment to the seller then refund is not possible. 8. Update the refund amount in users wallet.   **Deposits**   1. The admin should be able to select user from the list of all users. 2. The system should enter the amount to deposit into the user’s account.   **Roles**  **Add role**   1. The system should allow the admin to create new roles, with the ability to specify the name and description of the role, as well as the permissions and privileges associated with the role.   **Roles List**   1. The system should allow the admin to view a list of all roles, with their names. 2. The system should allow the admin to view, edit or delete existing roles.   **Assign Role**   1. The system should allow the admin to assign a role to a user, with the option to specify the name and description of the role, as well as the associated permissions and privileges. 2. The admin should be able to save and reset the user role.  **3.4 Non-Functional Requirements** |

**Performance:** The website should load quickly, even with heavy traffic, to provide an optimal user experience.

**Scalability:** The website should be able to handle increased traffic as the business grows.

**Security:** The website should be secure to protect user data and transactions.

**Usability:** The website should be easy to use, intuitive, and accessible for all users, including those with disabilities.

**Responsiveness:** The website should be designed to work on a variety of devices, including desktops, laptops, tablets, and smartphones.

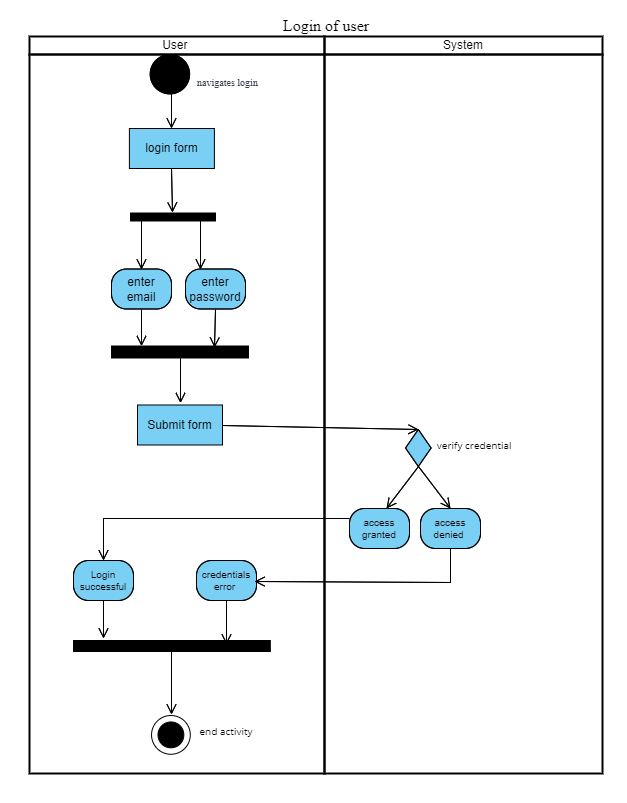
**Customizability:** The website should provide options for customizing the user experience, such as personalizing the homepage or product recommendations.

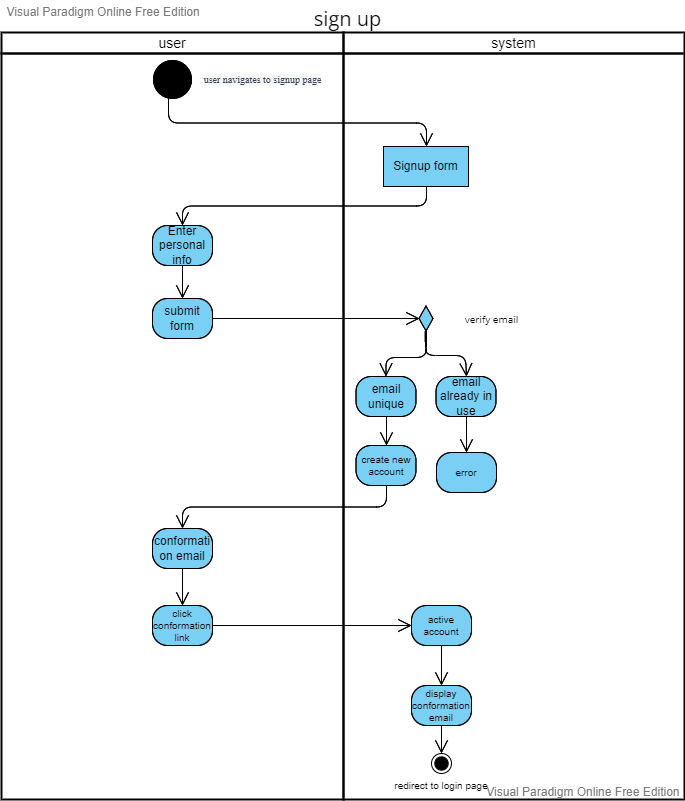
**Search Engine Optimization (SEO):** The website should be optimized for search engines to improve visibility and attract more traffic.

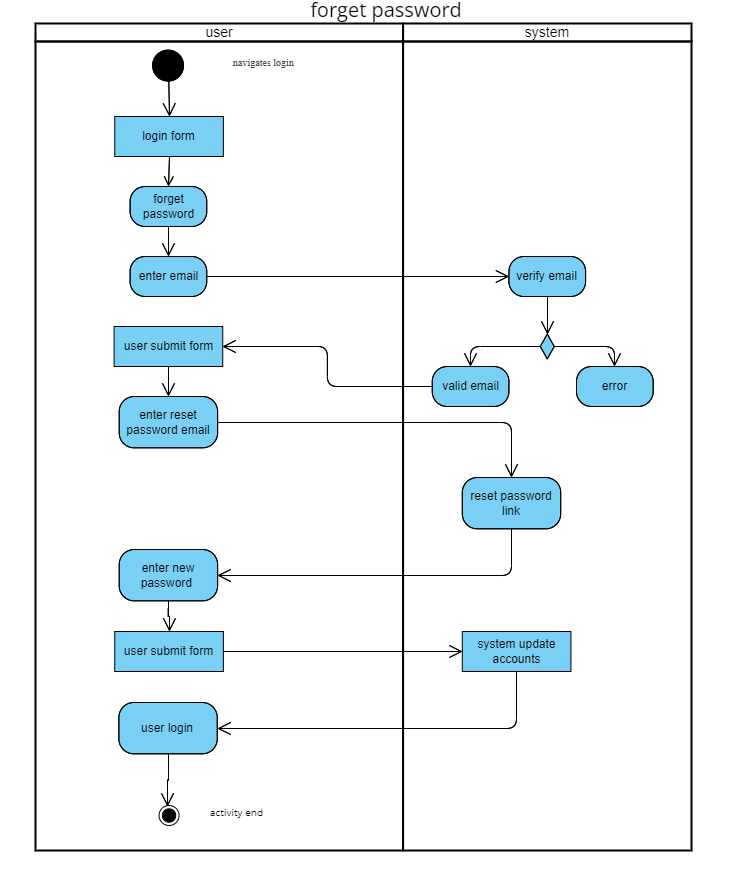
**Availability:** The website should be available 24/7, with minimal downtime for maintenance.

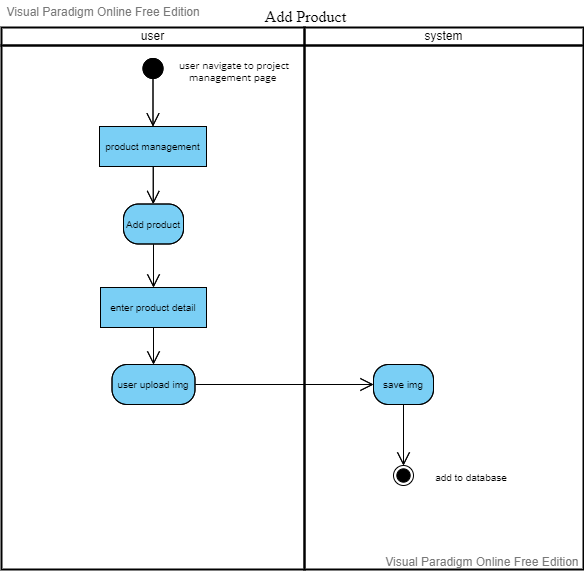
# **Chapter 4: System Design**

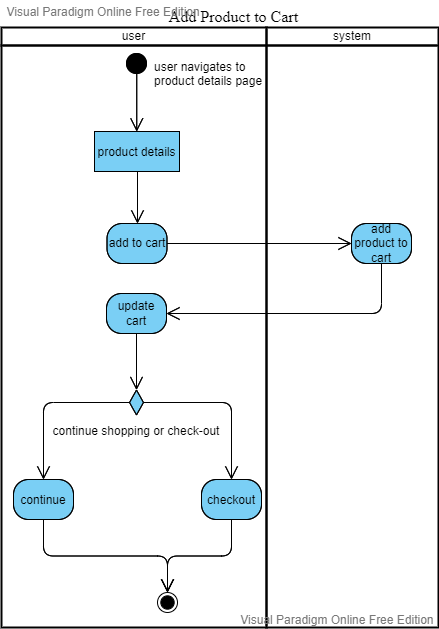
|  |
| --- |
| **4.1 Introduction** Architectural design refers to the process of creating and defining the high-level structure of a software system. It involves defining the components, modules, and interfaces that make up the system, as well as their relationships, constraints, and data flows. The architectural design is a critical step in the software development process, as it provides the foundation for all subsequent design and implementation work. It is concerned with making sure that the software system meets the requirements and constraints of the stakeholders, as well as making sure that the system is scalable, maintainable, and efficient. The architectural design should be based on well-established design patterns and best practices, and should consider factors such as performance, security, and modularity. |
| **4.2 Architectural Design****4.2.1 MVC Architectural Diagram** **General Architecture Diagram**  **Checkout Architecture Diagram**    **Manage Product Architecture Diagram**  **Inspection Architecture Diagram**    **Deployment Diagram at Development level**  **Deployment Diagram at Production level** **ERD Diagram**  **4.3 Detailed Design****Use Case Diagrams****4.3.1.1 Rider Use case Diagram**  **4.3.1.2 Admin Use case Diagram**  **4.3.1.3 User Use case Diagram**                            **Detailed Use Cases****4.3.2.1 Detailed Use case for user module**   **Detailed use case of Login for user module**   |  |  | | --- | --- | | **Use Case #** | 1 | | **Title** | Login | | **Actor** | User | | **Precondition** | The user should have an account.  The user is not already logged in to their account. | | **Flow of Events** | The user navigates to the login page of the website.  The website displays a login form, which asks the user to enter their email address and password.  The user enters their email address and password into the form and submits it.  The website verifies the user's credentials and logs the user in to their account if they are correct.  The website displays a welcome message to the user and redirects them to the homepage. | | **Post Conditions** | The user is logged in to their account and can access their account-specific features and functionality.  The website displays the user's name or email address in the top right corner to indicate that the user is logged in. | | **Exceptions** | If the user's credentials are incorrect, the website displays an error message and prompts the user to try again.  If the user's account has been disabled or suspended, the website displays an error message and does not allow the user to log in. |   **Detailed use case of Signup for user module**   |  |  | | --- | --- | | **Use Case #** | 2 | | **Title** | Signup | | **Actor** | User | | **Precondition** | The user does not have an account. | | **Flow of Events** | 1. The user navigates to the signup page of the khas 2. The website displays a signup form, which asks the user to enter their personal information, such as their name, email address, and password. 3. The user enters their personal information into the form and submits it. 4. The website verifies that the user's email address is unique and not already in use. 5. The website creates a new account for the user and sends a confirmation email to the user's email address. 6. The user clicks the confirmation link in the email to activate their account. 7. The website displays a confirmation message to the user and redirects them to the login page. | | **Post Conditions** | The user has a new account on the khas website and can log in to their account using their email address and password. | | **Exceptions** | 1. If the user's email address is already in use, displays an error message and prompts the user to enter a different email address. 2. If the user's personal information is incomplete or invalid, Website displays an error message and prompts the user to correct the error. |   **Detailed use case of Forgot Password for user module**   |  |  | | --- | --- | | **Use Case #** | 3 | | **Title** | Forgot Password | | **Actor** | User | | **Precondition** | 1. The user has an account on the website. 2. The user is not logged in to their account. | | **Flow of Events** | 1. The user navigates to the login page. 2. The user clicks the "Forgot password?" link on the login form. 3. Displays a form that asks the user to enter their email address. 4. The user enters their email address and submits the form. 5. Send a password reset email to the user's email address. 6. The user clicks the password reset link in the email. 7. The website displays a form that allows the user to enter a new password. 8. The user enters a new password and submits the form. 9. Website Updates the user's account with the new password and logs the user in to their account. | | **Post Conditions** | The user's password is reset and they can log in to their account using the new password. | | **Exceptions** | 1. If the user's email address is not recognized, the website displays an error message and prompts the user to enter a different email address. 2. If the password reset link has expired or been used, the website displays an error message and prompts the user to request a new password reset email. |   **Detailed use case of Add Product for user module**   |  |  | | --- | --- | | **Use Case #** | 4 | | **Title** | Add Product | | **Actor** | User, Admin | | **Precondition** | 1. The user has an account on the website. 2. The user is logged in to their account. 3. The user has the necessary permissions to add products to the website. | | **Flow of Events** | 1. The user navigates to the product management page in their account. 2. The user clicks the "Add Product" button. 3. The user enters the product details, such as the name, description, price, and specifications,conditions into the form provided. 4. The user uploads images of the product to be displayed on the website. 5. The user clicks the "Save" button to add the product to the database. | | **Post Conditions** | The product is added to the database and is visible to users on the website. | | **Exceptions** | 1. If the user does not have the necessary permissions to add products, display an error message and does not allow the user to add the product. 2. If the product details are incomplete or invalid, display an error message and prompts the user to correct the errors before saving the product. |   **Detailed use case of Add Product to Cart for user module**   |  |  | | --- | --- | | **Use Case #** | 5 | | **Title** | Add Product to Cart | | **Actor** | User | | **Precondition** | The user has found a product they want to purchase.  The user should have an account on website. | | **Flow of Events** | 1. The user navigates to the product details page of the desired product. 2. The user clicks the "Add to Cart" button. 3. The website adds the product to the user's shopping cart and updates the cart count to reflect the number of items in the cart. 4. The user can continue shopping or navigate to the shopping cart to review and checkout their items. | | **Post Conditions** | The product is added to the user's shopping cart and is available for checkout. | | **Exceptions** | 1. If the product is out of stock or no longer available, the display an error message and does not allow the user to add the product to their cart. 2. If same Product is already present in cart website shows message to user that the product is already present in cart. |   **Detailed use case of Send Price Offer for user module**   |  |  | | --- | --- | | **Use Case #** | 6 | | **Title** | Send Price Offer | | **Actor** | User | | **Precondition** | The user has found a product that is listed for sale by a seller.  The user have an account and is logged in.  The product it still available to sell. | | **Flow of Events** | 1. The user navigates to the product details page of the desired product. 2. The user clicks the "Send Offer" button. 3. The website displays a form that allows the user to enter their desired price. 4. The user enters their desired price submits the form. 5. The website sends the price offer to the seller and displays a confirmation message to the user. | | **Post Conditions** | The user's price offer is sent to the seller and the seller can choose to accept, reject | | **Exceptions** | 1. If the user does not have the necessary permissions to send price offers, the website displays an error message and does not allow the user to send the offer. 2. If the user's price offer is invalid, Website displays an error message and prompts the user to enter a different price. |     **Detailed use case of Add to Compare List for user module**   |  |  | | --- | --- | | **Use Case #** | 7 | | **Title** | Add to Compare List | | **Actor** | User | | **Precondition** | 1. The user has found at least one product to add in compare list. 2. The user is already have an account and logged in. | | **Flow of Events** | 1. The user navigates to the product page of the first product they want to compare. 2. The user clicks the "Add to Compare" button on the product page. 3. The website adds the product to the user's compare list and displays a confirmation message. 4. The user navigates to the product page of the second product they want to compare. 5. The user clicks the "Add to Compare" button on the product page. 6. Add the product to the user's compare list and displays a confirmation message. 7. The user clicks the "Compare" button in the header the website to view the compare list. 8. Display a comparison page with the selected products side by side, allowing the user to easily compare their features and specifications. | | **Post Conditions** | The selected products are added to the user's compare list and are available for comparison on the compare page. | | **Exceptions** | 1. If the user tries to add more than the maximum allowed number of products to the compare list, display an error message and does not allow the user to add the product. 2. If the user is not logged in ask him to login first by displaying a message. |   **Detailed use case of Search Product for user module**   |  |  | | --- | --- | | **Use Case #** | 8 | | **Title** | Search Product | | **Actor** | User | | **Precondition** | 1. The user is looking for a specific product or type of product. 2. More than one product is available on website. | | **Flow of Events** | 1. The user enters a search query into the search bar in the header of the website. 2. The user clicks the "Search" button or hits the "Enter" key on their keyboard. 3. The website searches its database for products matching the user's search query. 4. Website displays a list of search results, including the product name, image, and price and other tags for each product. 5. The user can browse the search results and click on a product to view its product details page. | | **Post Conditions** | The user is presented with a list of search results matching their search query. | | **Exceptions** | 1. If the user's search query does not match any products in the database, Website displays a message indicating that no results were found. |     **Detailed use case of Search Filter for user module**   |  |  | | --- | --- | | **Use Case #** | 9 | | **Title** | Search Filter | | **Actor** | User | | **Precondition** | The user has performed a search and is presented with a list of search results. | | **Flow of Events** | 1. The user navigates to the search results page. 2. The user selects one or more filters from the filter options provided, such as price range, brand, or specifications. 3. The user clicks the "Apply" button to apply the selected filters. 4. The website updates the search results to only include products that match the selected filters. 5. The user can continue to refine their search by applying additional filters or modifying the existing filters. | | **Post Conditions** | The search results are filtered to only include products that match the selected filters. | | **Exceptions** | If the user selects filters that do not match any products in the search results, then website displays a message indicating that no results were found. |   **Detailed use case of Edit Cart for user module**   |  |  | | --- | --- | | **Use Case #** | 10 | | **Title** | Edit Cart | | **Actor** | User | | **Precondition** | 1. The user is logged in to their account on the website. 2. The user has added items to their shopping cart. | | **Flow of Events** | 1. The user navigates to the shopping cart page. 2. The user modifies the quantity of an item in their cart by entering a new value in the quantity field and clicking the "Update" button. 3. The website updates the item's quantity and displays the updated total for the item. 4. The user can continue to modify the quantities of other items in their cart. 5. The user can remove an item from their cart by clicking the "Remove" button next to the item. 6. The website removes the item from the cart and updates the cart total. 7. User can also click on the remove all item button to remove all item at once. | | **Post Conditions** | The items in the user's shopping cart are modified according to the user's changes. | | **Exceptions** | If the user tries to set the quantity of an item to a value that is not available or exceeds the maximum allowed quantity, the website displays an error message and does not allow the user to update the quantity. |   **Detailed use case of Apply Coupon for user module**   |  |  | | --- | --- | | **Use Case #** | 11 | | **Title** | Apply Coupon | | **Actor** | User | | **Precondition** | 1. The user is logged in to their account on the website. 2. The user has added items to their shopping cart. 3. The user has a valid coupon code. | | **Flow of Events** | 1. The user navigates to the shopping cart page. 2. The user enters the coupon code into the coupon code field. 3. The user clicks the "Apply" button. 4. The website verifies the coupon code and checks if it is valid and can be applied to the purchase. 5. If the coupon is valid, the website applies the discount to the total purchase amount and displays the updated total. 6. If the coupon is invalid or cannot be applied, the website displays an error message. | | **Post Conditions** | If the coupon is valid, the discount is applied to the total purchase amount. | | **Exceptions** | 1. If the coupon code is invalid or has already been used, the website displays an error message and does not apply the discount. 2. If the coupon has expired or is not applicable to the items in the shopping cart, the website displays an error message and does not apply the discount. |   **Detailed use case of Checkout for user module**   |  |  | | --- | --- | | **Use Case #** | 12 | | **Title** | Checkout | | **Actor** | User | | **Precondition** | 1. The user is logged in to their account. 2. The user has added items to their shopping cart. 3. The user has selected a shipping address for their order. | | **Flow of Events** | 1. The user navigates to the checkout page. 2. The website displays the items in the user's shopping cart, the shipping address, and the total purchase amount. 3. The user selects a payment method from the options provided. 4. The user enters their payment details into the form provided. 5. The user reviews their order and clicks the "Place Order" button. 6. The website processes the payment and creates the order. 7. The website displays a confirmation message and provides the user with a copy of their order details, including the order number. | | **Post Conditions** | 1. The order is placed and the payment is processed. 2. The user receives a confirmation of their order in website notification and via email. | | **Exceptions** | 1. If the payment is declined or fails for any reason, the website displays an error message and does not place the order. 2. If the user has not selected a valid shipping address, the website prompts the user to select a shipping address before proceeding with the checkout. |  **4.3.2.2 Detailed Use case for rider modules** **Detailed Use case of Accept Order for rider for rider modules**   |  |  | | --- | --- | | **Use Case #** | 1 | | **Title** | Accept Order for rider | | **Actor** | Delivery Boy | | **Precondition** | 1. The delivery boy must be logged in to the delivery system. 2. The delivery boy must have a valid account with the delivery company. 3. The delivery boy must be on duty and available to take on new orders. | | **Flow of Events** | 1. The delivery boy logs in to the delivery system using their username and password. 2. The delivery system displays a list of available orders in the delivery area. 3. The delivery boy selects an order from the list and clicks the "Accept Order" button. 4. The delivery system verifies that the delivery boy is eligible to take on the selected order. 5. If the delivery boy is eligible, the delivery system assigns the order to the delivery boy and displays the details of the order, including the delivery address, the items to be delivered, and any special instructions. 6. The delivery boy reviews the order details and confirms that they understand and will fulfill the requirements of the order. 7. The delivery system marks the order as "Accepted" and updates the delivery boy's pending orders to reflect the new assignment. 8. The delivery boy begins the delivery process, following the instructions provided by the delivery system and any additional instructions provided by the customer or the khas company. | | **Post Conditions** | 1. The delivery boy has accepted the order and is responsible for fulfilling the delivery. 2. The delivery system has updated the order status to "Accepted" and the delivery boy's pending orders to reflect the new assignment. | | **Exceptions** | 1. If the delivery boy is not eligible to take on the selected order (e.g. due to location restrictions or availability), the delivery system will display an error message and the delivery boy will need to select a different order. 2. If the delivery boy is unable to confirm their understanding and acceptance of the order, the delivery system will not mark the order as "Accepted" and the delivery boy will need to review and confirm the order details before proceeding. |  **4.3.2.3 Detailed Use case for Admin modules** **Detailed Use case of Add User for Admin modules**  |  |  | | --- | --- | | **Use Case #** | 1 | | **Title** | Add User | | **Actor** | Admin | | **Precondition** | 1. The admin has the necessary permissions to add a new user to the system. 2. The new user has provided the necessary information, such as a username, password, and any required personal or contact details. | | **Flow of Events** | 1. The admin navigates to the "Add User" page in the system's admin panel. 2. The admin enters the new user's information, including the username, password, and any required personal or contact details. 3. The admin clicks the "Add User" button to submit the new user's information. 4. Website will automatically assign a user role to this new user. 5. The website successfully adds the new user and displays a confirmation message. 6. The new user receives an email notification with login information and instructions on how to log in to the system. | | **Post Conditions** | 1. The new user has been added to the system and can access the resources and privileges associated with their role. 2. The admin has a record of the new user in the database. | | **Exceptions** | 1. If the new user's information is incomplete or invalid, the system may display an error message and prompt the admin to correct the information before adding the user. 2. If the system is unable to add the new user if the user already exists or due to a technical issue, it may display an error message and prompt the admin to try again later. |   **Detailed Use case of Manage Product for Admin modules**  |  |  | | --- | --- | | **Use Case #** | 2 | | **Title** | Manage Product | | **Actor** | Admin | | **Precondition** | 1. The admin has the necessary permissions to manage products in the system 2. The admin has access to the necessary product information, such as a product name, description, price, and any relevant details | | **Flow of Events** | 1. The admin navigates to the "Manage Product" page in the system's admin panel. 2. The admin selects the appropriate action: add a new product, update an existing product, or delete a product. 3. If adding a new product, the admin enters the product information, including the name, description, price, and any relevant details. If updating or deleting a product, the admin selects the appropriate product from a list or search results. 4. The admin clicks the "Submit" button to save the changes. 5. If the admin deletes the product the system will ask for confirmation and if admin selects yes then delete the product and if admin selects no then don’t delete the product. 6. If the admin clicks on update button the system should open form with all existing data of product and admin should be able to change them. 7. When the system successfully adds, updates, or deletes the product and displays a confirmation message. | | **Post Conditions** | 1. The product has been added, updated, or deleted in the system as specified by the admin. 2. The database reflects the changes made by the admin. | | **Exceptions** | 1. If the product information is incomplete or invalid, the system may display an error message and prompt the admin to correct the information before saving the changes. 2. If the system is unable to add, update, or delete the product due to a technical issue, it may display an error message and prompt the admin to try again later. |  **Detailed Use case of Boost Product for Admin modules**  |  |  | | --- | --- | | **Use Case #** | 3 | | **Title** | Boost Product | | **Actor** | Admin | | **Precondition** | 1. The admin has the necessary permissions to boost products in the system 2. The admin has selected the products to be boosted and has determined the appropriate boost duration and method | | **Flow of Events** | 1. The admin navigates to the "Boost Product" page in the system's admin panel. 2. The admin selects the products to be boosted from a list or search results. 3. The admin specifies the duration of the boost, such as one week or one month. 4. The admin selects the boost method, such as featuring the product on the homepage or sending a promotional email to all users. 5. The admin clicks the "Boost" button to submit the boost request. 6. The system successfully boosts the selected products and displays a confirmation message. 7. The boosted products are prominently displayed to users according to the boost method chosen by the admin. | | **Post Conditions** | 1. The selected products have been boosted in the system according to the duration and method specified by the admin. 2. Users are more likely to discover and interact with the boosted products. | | **Exceptions** | 1. If the admin selects an invalid boost duration or method, the system may display an error message and prompt the admin to correct the selection. 2. If the system is unable to boost the products due to a technical issue, it may display an error message and prompt the admin to try again later. |   **Detailed Use case of Add Category for Admin modules**   |  |  | | --- | --- | | **Use Case #** | 4 | | **Title** | Add Category | | **Actor** | Admin | | **Precondition** | 1. The admin has the necessary permissions to add categories in the system 2. The admin has determined the name and any relevant details for the new category | | **Flow of Events** | 1. The admin navigates to the "Add Category" page in the system's admin panel. 2. The admin enters the name the new category. 3. The admin clicks the "Add Category" button to submit the new category. 4. The system successfully adds the new category and displays a confirmation message. | | **Post Conditions** | 1. The new category has been added to the system and can be used to organize products. 2. The system's category database reflects the addition of the new category. | | **Exceptions** | 1. If the category name is invalid or already in use, the system may display an error message and prompt the admin to enter a different name. 2. If the system is unable to add the new category due to a technical issue, it may display an error message and prompt the admin to try again later. |   **Detailed Use case of Add Brand for Admin modules**   |  |  | | --- | --- | | **Use Case #** | 5 | | **Title** | Add Brand | | **Actor** | Admin | | **Precondition** | 1. The admin has the necessary permissions to add brands in the system 2. The admin has determined the name and any relevant details for the new brand | | **Flow of Events** | 1. The admin navigates to the "Add Brand" page in the system's admin panel. 2. The admin enters the name and any relevant details for the new brand. 3. The admin clicks the "Add Brand" button to submit the new brand. 4. The system successfully adds the new brand and displays a confirmation message. | | **Post Conditions** | 1. The new brand has been added to the system and can be associated with products. 2. The system's brand database reflects the addition of the new brand. | | **Exceptions** | 1. If the brand name is invalid or already in use, the system may display an error message and prompt the admin to enter a different name. 2. If the system is unable to add the new brand due to a technical issue, it may display an error message and prompt the admin to try again later. |   **Detailed Use case of Add Inspection for Admin modules**   |  |  | | --- | --- | | **Use Case #** | 6 | | **Title** | Add Inspection | | **Actor** | Admin | | **Precondition** | 1. The admin has the necessary permissions to add inspections in the system 2. The admin has determined the necessary details for the inspection, such as the product to be inspected, the inspection criteria. | | **Flow of Events** | 1. The admin navigates to the "Add Inspection" page in the system's admin panel. 2. The admin selects the product to be inspected from a list or search results. 3. The admin enters all required information in inspection form. 4. The admin clicks the "Save Inspection" button to submit the new inspection. 5. The system successfully adds the new inspection and displays a confirmation message. 6. The system will also save the details of the inspector along with the inspection details. | | **Post Conditions** | 1. The new inspection has been added to the system and can be accessed by seller, buyer and admin. 2. The system's inspection database reflects the addition of the new inspection. | | **Exceptions** | 1. If the admin selects an invalid asset, or fails to specify the necessary inspection criteria, the system may display an error message and prompt the admin to correct the information. 2. If the system is unable to add the new inspection due to a technical issue, it may display an error message and prompt the admin to try again later. |     **Detailed Use case of Edit Sale for Admin modules**   |  |  | | --- | --- | | **Use Case #** | 7 | | **Title** | Edit Sale | | **Actor** | Admin | | **Precondition** | 1. The admin has the necessary permissions to edit sales in the system 2. The admin has located the sale to be edited and has determined the necessary changes to be made | | **Flow of Events** | 1. The admin navigates to the "Edit Sale" page in the system's admin panel. 2. The admin searches for and selects the sale to be edited from a list or search results. 3. The admin makes the necessary changes to the sale information, such as adjusting the quantity or price of a product, or adding or removing informations from the sale. 4. The admin clicks the "Save Changes" button to submit the updated sale information. 5. The system successfully updates the sale and displays a confirmation message. | | **Post Conditions** | 1. The sale has been updated in the system according to the changes made by the admin. 2. System should notify by email to both seller and buyer about this new change. 3. The system's sale database reflects the updated sale information. | | **Exceptions** | 1. If the sale information is invalid or incomplete, the system may display an error message and prompt the admin to correct the information before saving the changes. 2. If the system is unable to edit the sale due to a technical issue, it may display an error message and prompt the admin to try again later. |   **Detailed Use case of Approve Withdrawal for Admin modules**   |  |  | | --- | --- | | **Use Case #** | 8 | | **Title** | Approve Withdrawal | | **Actor** | Admin | | **Precondition** | 1. The admin has the necessary permissions to review and approve withdrawal requests 2. The users have made valid withdrawal requests and provided the necessary information, such as the amount and method of withdrawal | | **Flow of Events** | 1. The admin navigates to the "Approve Withdrawal" page in the system's admin panel. 2. The admin selects the withdrawal request to be reviewed from a list or search results. 3. The admin reviews the details of the withdrawal request, including the user's account balance and any relevant terms and conditions. 4. The admin approves the withdrawal request if it is valid and meets the necessary criteria. 5. The system processes the approved withdrawal request and transfers the funds or assets to the user's account or designated method of withdrawal. 6. The system sends a notification to the user confirming the approval of the withdrawal request. | | **Post Conditions** | 1. The withdrawal request has been approved and the funds or assets have been transferred to the user as specified in the request. 2. The system's database reflects the approval of the request and updates the balance of users wallet. | | **Exceptions** | 1. If the withdrawal request is invalid or does not meet the necessary criteria, the admin may deny the request and send a notification to the user explaining the reason for the denial. 2. If the system is unable to process the withdrawal request due to a technical issue, it may display an error message and prompt the admin to try again later. |   **Detailed Use case of Approve Refund for Admin modules**   |  |  | | --- | --- | | **Use Case #** | 9 | | **Title** | Approve Refund | | **Actor** | Admin | | **Precondition** | 1. The admin has the necessary permissions to review and approve refund requests. 2. The users have made valid refund requests and provided the necessary information, such as the reason for the refund and the original purchase information. 3. Khas has received the product from the seller against which the refund is requested. | | **Flow of Events** | 1. The admin navigates to the "Approve Refund" page in the system's admin panel. 2. The admin selects the refund request to be reviewed from a list or search results. 3. The admin reviews the details of the refund request, including the user's account balance and the original purchase information. 4. The admin approves the refund request if it is valid and meets the necessary criteria. 5. The system processes the approved refund request and credits the user's account wallet 6. The system sends a notification to the user confirming the approval of the refund request. 7. And Khas will send the product to the seller. | | **Post Conditions** | 1. The refund request has been approved and the user has received a refund as specified in the request. 2. The system's database reflects the approval of the request. | | **Exceptions** | 1. If the refund request is invalid or does not meet the necessary criteria, the admin may deny the request and send a notification to the user explaining the reason for the denial. 2. If the system is unable to process the refund request due to a technical issue, it may display an error message and prompt the admin to try again later. |   **Detailed Use case of Add Deposit for Admin modules**   |  |  | | --- | --- | | **Use Case #** | 10 | | **Title** | Add Deposit | | **Actor** | Admin | | **Precondition** | 1. The admin has the necessary permissions to add deposits to user accounts 2. The admin has determined the amount and method of the deposit | | **Flow of Events** | 1. The admin navigates to the "Add Deposit" page in the system's admin panel. 2. The admin selects the user's account from a list or search results. 3. The admin enters the amount and method of the deposit. 4. The admin clicks the "Add Deposit" button to submit the deposit request. 5. The system processes the deposit and credits the user's account wallet. 6. The system sends a notification to the user confirming the deposit. | | **Post Conditions** | 1. The deposit has been added to the user's account as specified by the admin. 2. The system's transaction database reflects the deposit. | | **Exceptions** | 1. If the deposit amount or method is invalid, the system may display an error message and prompt the admin to correct the information before submitting the request. 2. If the system is unable to process the deposit due to a technical issue, it may display an error message and prompt the admin to try again later. |   **Detailed Use case of Add Role for Admin modules**   |  |  | | --- | --- | | **Use Case #** | 11 | | **Title** | Add Role | | **Actor** | Admin | | **Precondition** | 1. The admin has the necessary permissions to add roles in the system 2. The admin has determined the name and permissions for the new role | | **Flow of Events** | 1. The admin navigates to the "Add Role" page in the system's admin panel. 2. The admin enters the name and any relevant details for the new role. 3. The admin selects the permissions for the new role from a list of available options. 4. The admin clicks the "Add Role" button to submit the new role. 5. The system successfully adds the new role and displays a confirmation message. | | **Post Conditions** | 1. The new role has been added to the system and can be assigned to users. 2. The system's role database reflects the addition of the new role. | | **Exceptions** | 1. If the role name is invalid or already in use, the system may display an error message and prompt the admin to enter a different name. 2. If the system is unable to add the new role due to a technical issue, it may display an error message and prompt the admin to try again later. |   **Detailed Use case of Assign Role for Admin modules**   |  |  | | --- | --- | | **Use Case #** | 12 | | **Title** | Assign Role | | **Actor** | Admin | | **Precondition** | 1. The admin has the necessary permissions to assign roles in the system 2. The admin has determined the appropriate role for the user based on their responsibilities and tasks | | **Flow of Events** | 1. The admin navigates to the "Assign Role" page in the system's admin panel. 2. The admin selects the employee to be assigned a role from a list or search results. 3. The admin selects the appropriate role from a list of available options. 4. The admin clicks the "Assign Role" button to submit the role assignment. 5. The system successfully assigns the role to the user and displays a confirmation message. | | **Post Conditions** | 1. The user has been assigned the selected role and can access the necessary permissions and responsibilities. 2. The system's database reflects the assignment of the role to the user. | | **Exceptions** | 1. If the selected role is invalid or unavailable for the user, the system may display an error message and prompt the admin to select a different role. 2. If the system is unable to assign the role due to a technical issue, it may display an error message and prompt the admin to try again later. |   **Detailed Use case of Add Employee for Admin modules**   |  |  | | --- | --- | | **Use Case #** | 13 | | **Title** | Add Employee | | **Actor** | Admin | | **Precondition** | 1. The admin has the necessary permissions to add employees to the system. 2. The admin has obtained the necessary personal and professional information for the new employee | | **Flow of Events** | 1. The admin navigates to the "Add Employee" page in the system's admin panel. 2. The admin enters the personal and professional information for the new employee, including their name, contact information, designation , and any relevant details. 3. The admin clicks the "Add Employee" button to submit the new employee information. 4. The system successfully adds the new employee and displays a confirmation message. | | **Post Conditions** | 1. The new employee has been added to the system and can access the necessary resources and tools for their role. 2. The system's database reflects the addition of the new employee.\ 3. Employee can login and use admin panel according to their role and permissions. | | **Exceptions** | 1. If the employee information is invalid or incomplete, the system may display an error message and prompt the admin to correct the information before adding the employee. 2. If the system is unable to add the new employee due to a technical issue, it may display an error message and prompt the admin to try again later. |   **Detailed Use case of Add Designation for Admin modules**   |  |  | | --- | --- | | **Use Case #** | 14 | | **Title** | Add Designation | | **Actor** | Admin | | **Precondition** | 1. The admin has the necessary permissions to add designations to the system 2. The admin has determined the name and relevant details for the new designation | | **Flow of Events** | 1. The admin navigates to the "Add Designation" page in the system's admin panel. 2. The admin enters the for the new designation. 3. The admin clicks the "Add Designation" button to submit the new designation. 4. The system successfully adds the new designation and displays a confirmation message. | | **Post Conditions** | 1. The new designation has been added to the system and can be assigned to employees. | | **Exceptions** | 1. If the designation name is invalid or already in use, the system may display an error message and prompt the admin to enter a different name. 2. If the system is unable to add the new designation due to a technical issue, it may display an error message and prompt the admin to try again later. |   **Detailed Use case of Add Attendance for Admin modules**   |  |  | | --- | --- | | **Use Case #** | 15 | | **Title** | Add Attendance | | **Actor** | Admin | | **Precondition** | 1. The admin has the necessary permissions to add attendance records to the system 2. The admin has obtained the necessary attendance information for the employee, including the dates and times of attendance | | **Flow of Events** | 1. The admin navigates to the "Add Attendance" page in the system's admin panel. 2. The admin selects the employee whose attendance is being recorded from a list or search results. 3. The admin enters the attendance information for the employee, including the dates and times of attendance. 4. The admin clicks the "Add Attendance" button to submit the records. 5. The system successfully adds the attendance records and displays a confirmation message. | | **Post Conditions** | 1. The attendance records have been added to the system and can be used to calculate the employee's pay and benefits. | | **Exceptions** | 1. If the attendance information is invalid or incomplete, the system may display an error message and prompt the admin to correct the information before adding the attendance records. 2. If the system is unable to add the attendance records due to a technical issue, it may display an error message and prompt the admin to try again later. |   **Detailed Use case of Issue Leave for Admin modules**   |  |  | | --- | --- | | **Use Case #** | 16 | | **Title** | Issue Leave | | **Actor** | Admin | | **Precondition** | 1. The admin has the necessary permissions to issue leave requests in the system 2. The employee has provided the necessary information for the leave request, including the start and end dates and the reason for the leave | | **Flow of Events** | 1. The admin receives the leave request from the employee and navigates to the "Issue Leave" page in the system's admin panel. 2. The admin selects the employee's account from a list or search results. 3. The admin enters the start and end dates and the reason for the leave request. 4. The admin clicks the "Issue Leave" button to submit the request. 5. The system issues the leave request and sends a notification to the employee confirming the request. | | **Post Conditions** | 1. The leave request has been issued and is being processed according to the necessary actions. 2. Employees absents on these days will not be count as absent. | | **Exceptions** | 1. If the leave request is invalid or does not meet the necessary criteria, or employee have no more leave left the admin may deny the request. 2. If the system is unable to issue the leave request due to a technical issue, it may display an error message and prompt the admin to try again later. |   **Detailed Use case of Set Salary for Admin modules**   |  |  | | --- | --- | | **Use Case #** | 17 | | **Title** | Set Salary | | **Actor** | Admin | | **Precondition** | 1. The admin has the necessary permissions to set salaries in the system 2. The admin has determined the appropriate salary for the employee based on their role, responsibilities, and performance | | **Flow of Events** | 1. The admin navigates to the "Set Salary" page in the system's admin panel. 2. The admin selects the employee whose salary is being set from a list or search results. 3. The admin enters the salary amount and any relevant details for the employee. 4. Admin enters the values for all benefits and deductions for the employee. 5. The admin clicks the "Set Salary" button to submit the salary information. 6. The system successfully sets the salary and displays a confirmation message. | | **Post Conditions** | 1. The salary has been set for the employee and can be used to calculate their pay and benefits. | | **Exceptions** | 1. If the salary information is invalid or incomplete, the system may display an error message and prompt the admin to correct the information before setting the salary. 2. If the system is unable to set the salary due to a technical issue, it may display an error message and prompt the admin to try again later. |   **Detailed Use case of Generate Payroll for Admin modules**   |  |  | | --- | --- | | **Use Case #** | 18 | | **Title** | Generate Payroll | | **Actor** | Admin | | **Precondition** | 1. The admin has the necessary permissions to generate payroll in the system 2. The necessary attendance and salary information has been entered and is up to date for all employees | | **Flow of Events** | 1. The admin navigates to the "Generate Payroll" page in the system's admin panel. 2. The admin selects the period for which the payroll is being generated. 3. The admin clicks the "Generate Payroll" button to initiate the process. 4. The system generates the payroll and displays a summary of the total pay for each employee. 5. The admin reviews the payroll summary and confirms that the pay is accurate. 6. The admin clicks the "Submit Payroll" button to finalize the process. 7. The system successfully generates the payroll and sends notifications to the employees with their pay details. | | **Post Conditions** | 1. The payroll has been generated and employees have received their pay and pay details. | | **Exceptions** | 1. If there are errors or discrepancies in the attendance or salary information, the system may display an error message and prompt the admin to correct the information before generating the payroll. 2. If the system is unable to generate the payroll due to a technical issue, it may display an error message and prompt the admin to try again later. |   **Detailed Use case of Add Bonus and Detection for Admin modules**   |  |  | | --- | --- | | **Use Case #** | 19 | | **Title** | Add Bonus and Detection | | **Actor** | Admin | | **Precondition** | 1. The admin has the necessary permissions to add bonuses and detections in the system 2. The admin has determined the appropriate bonus or detection for the employee based on their performance and contributions | | **Flow of Events** | 1. The admin navigates to the "Add Bonus and Detection" page in the system's admin panel. 2. The admin selects "Bonus" or "Detection" from a dropdown menu or enter value and enters the amount and any relevant details for the bonus or detection. 3. The admin clicks the "Add Bonus/Detection" button to submit the information. 4. The system successfully adds the bonus or detection and displays a confirmation message. | | **Post Conditions** | 1. The bonus or detection has been added can be used to set and calculate employees pay and benefits. | | **Exceptions** | 1. If the bonus or detection information is invalid or incomplete, the system may display an error message and prompt the admin to correct the information before adding the bonus or detection. 2. If the system is unable to add the bonus or detection due to a technical issue, it may display an error message and prompt the admin to try again later. |  **4.3.3 Activity Diagrams****4.3.3.1 Activity Diagrams for user** |

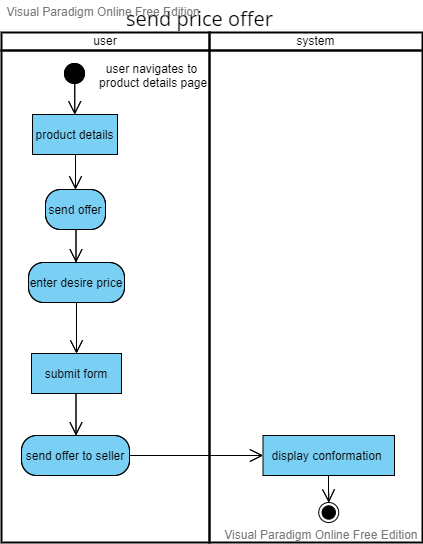


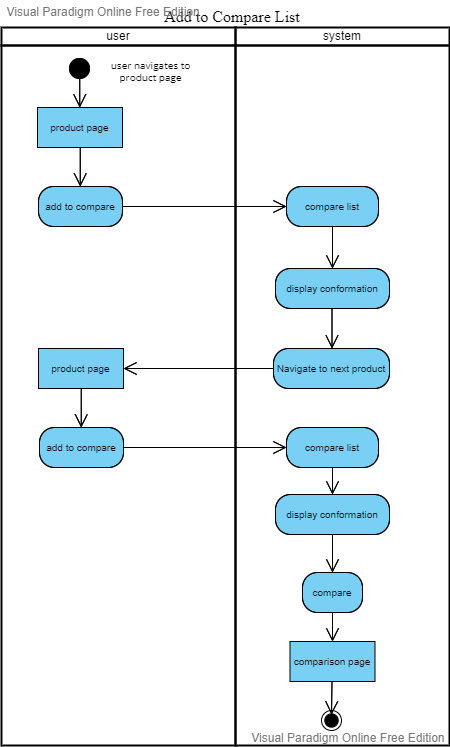


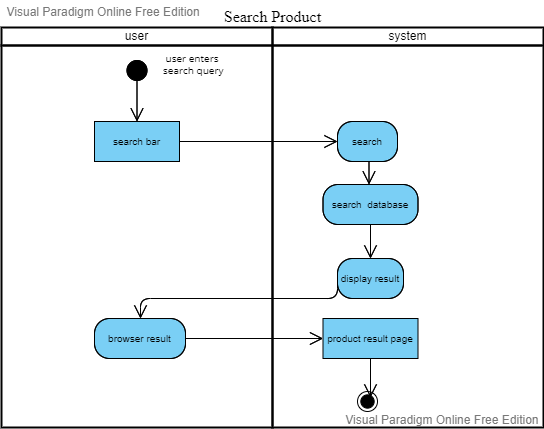
****

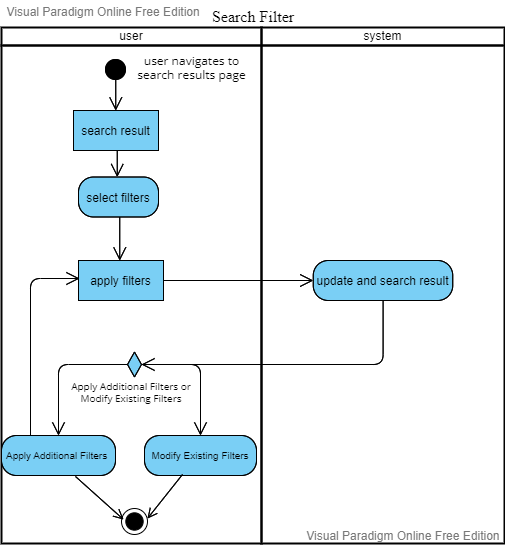
****

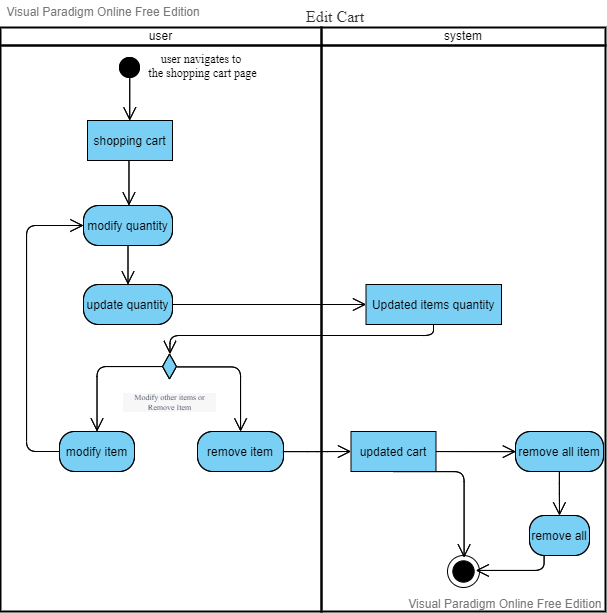
****

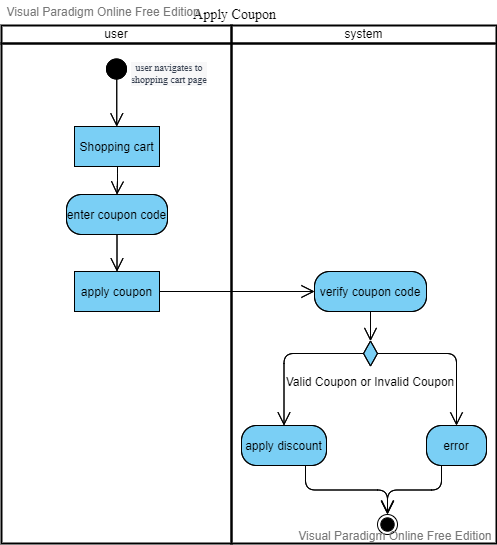
****

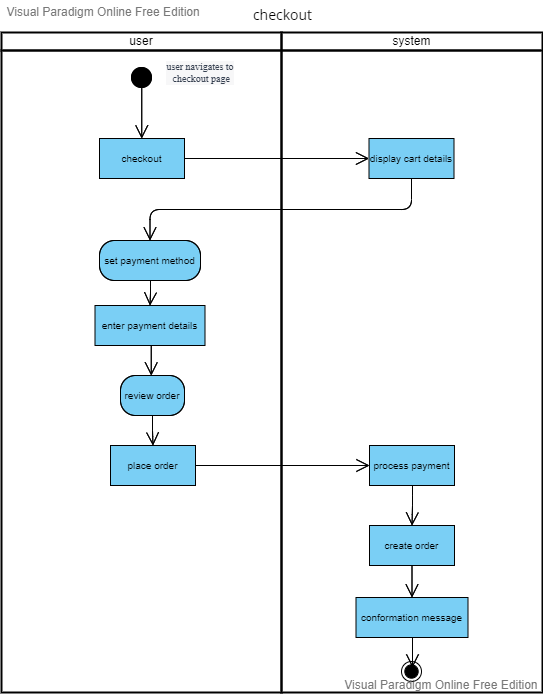
****

****

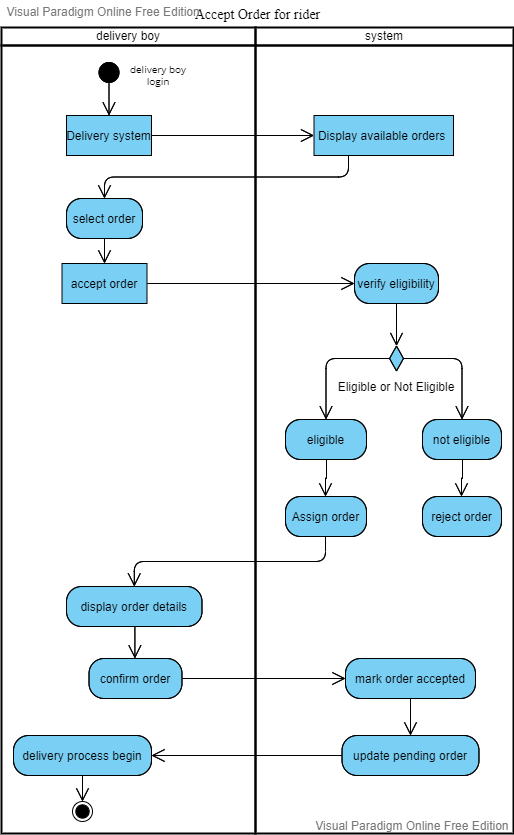
****

****

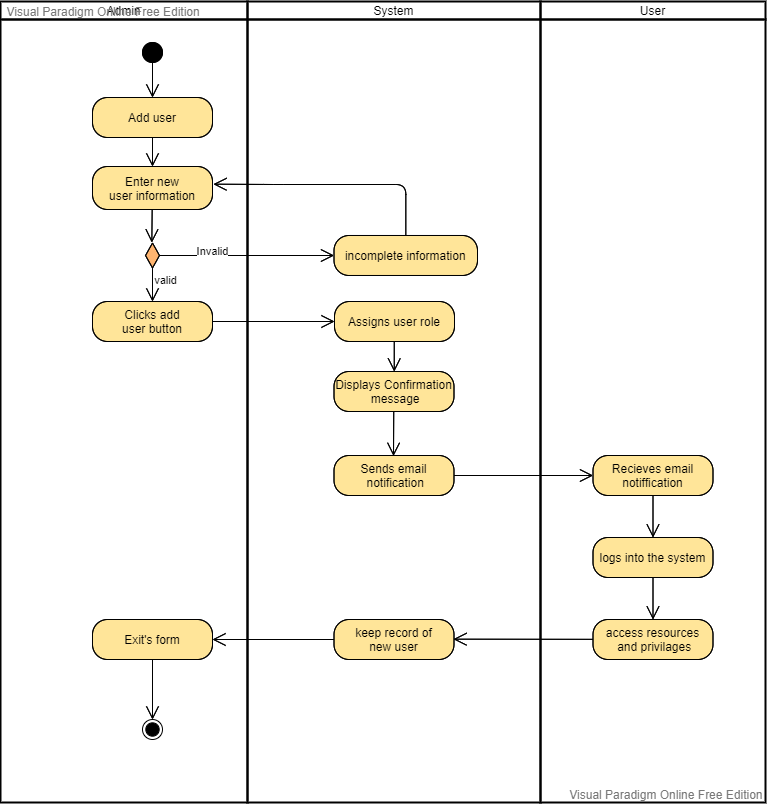
****

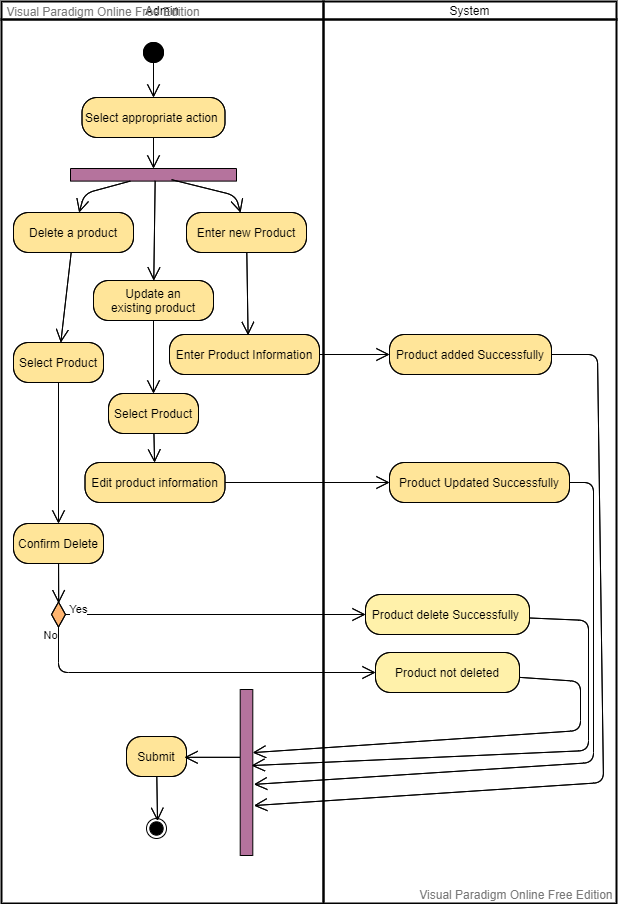
****

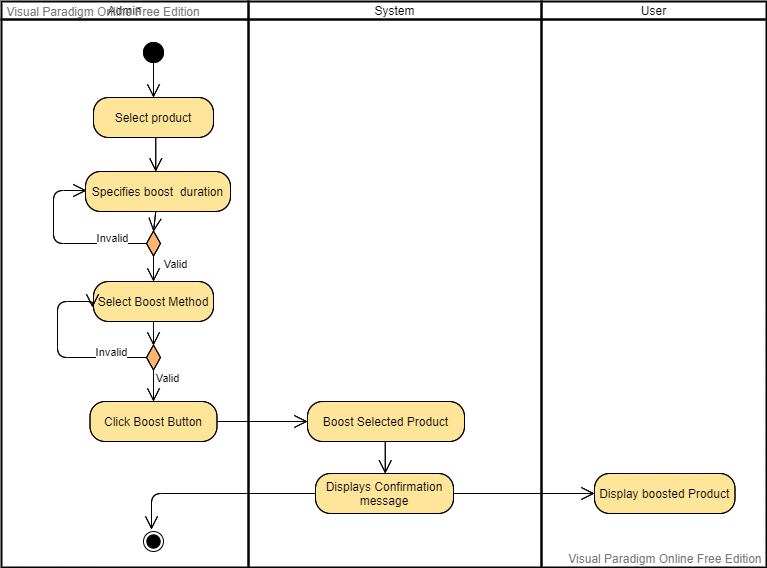
#### **4.3.3.2 Activity Diagrams for rider**

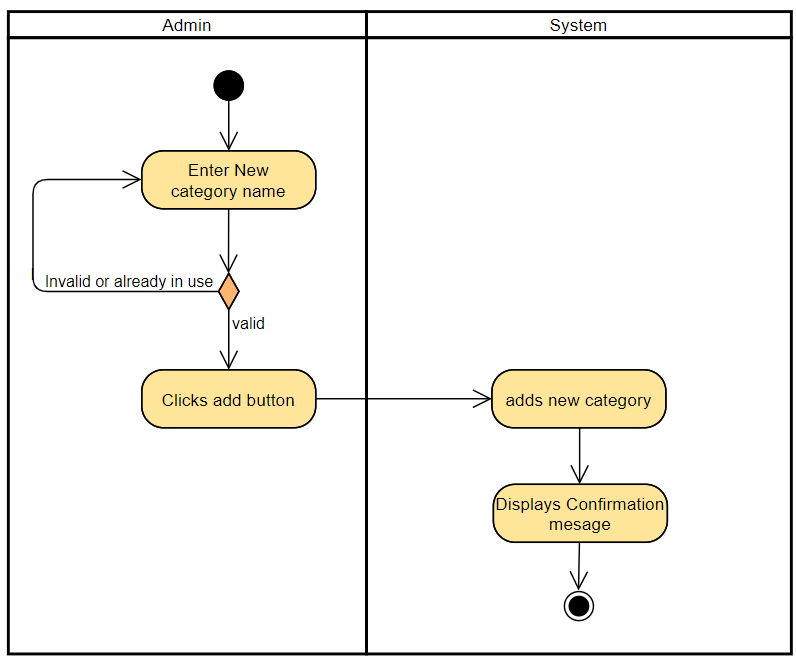


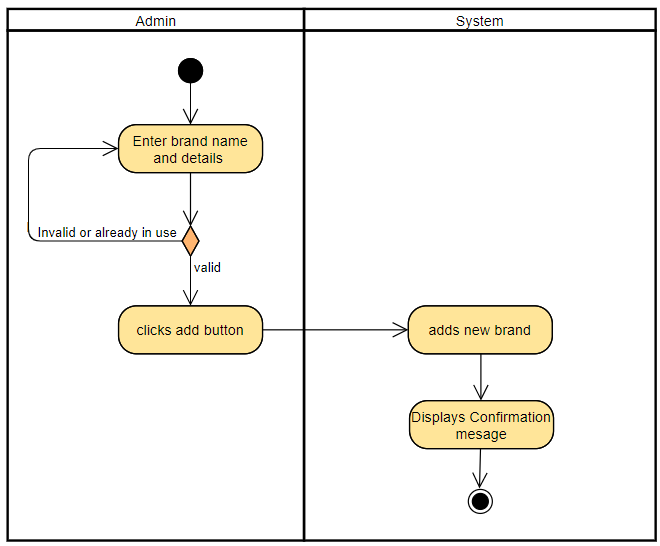
#### **4.3.3.3 Activity Diagrams for admin**

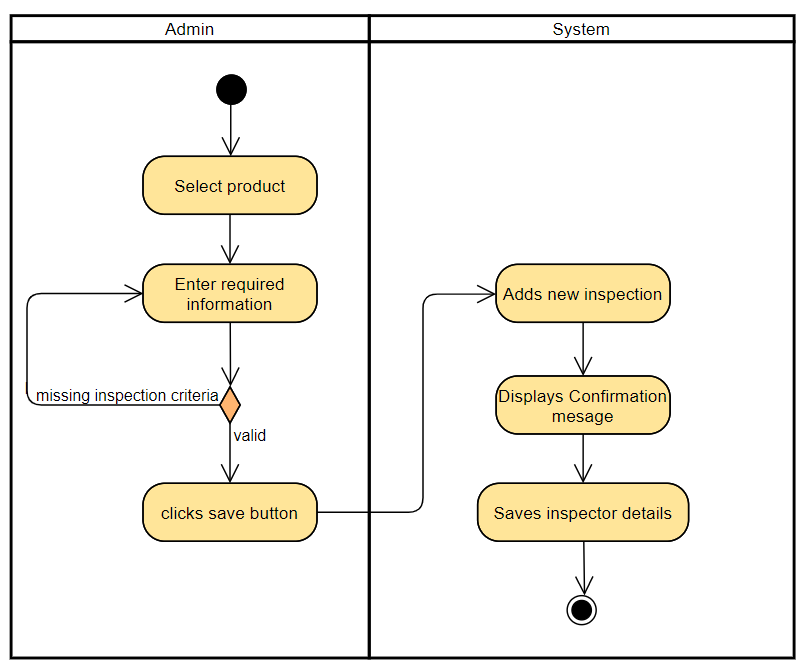
****

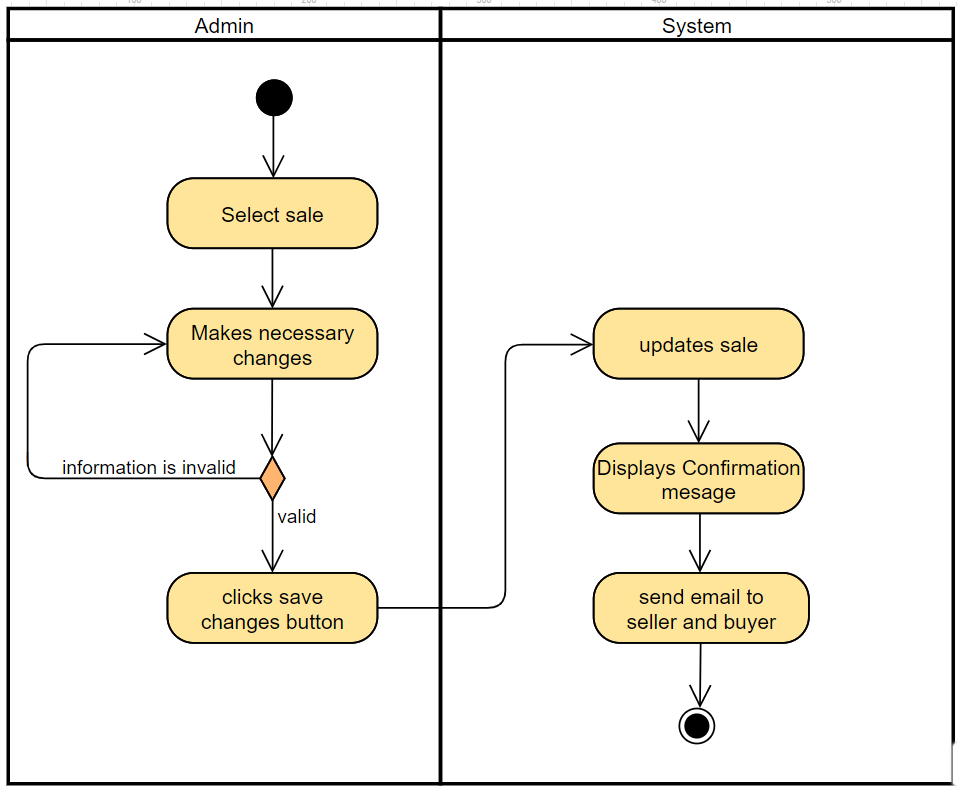
****

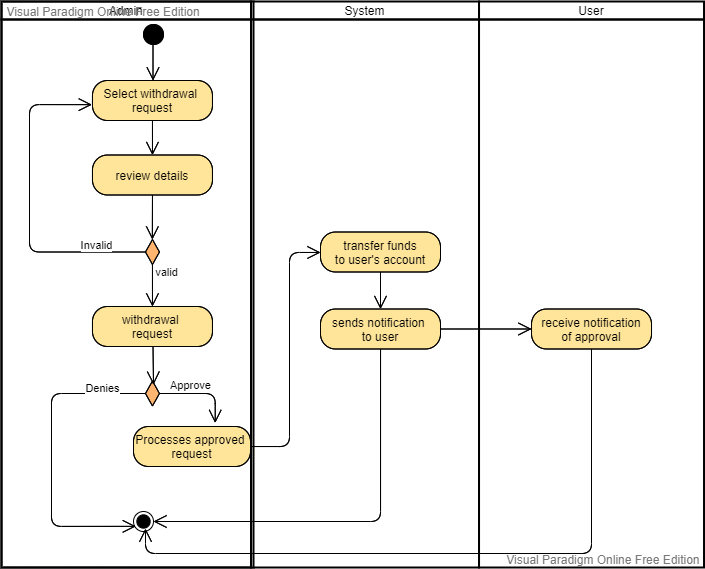
****

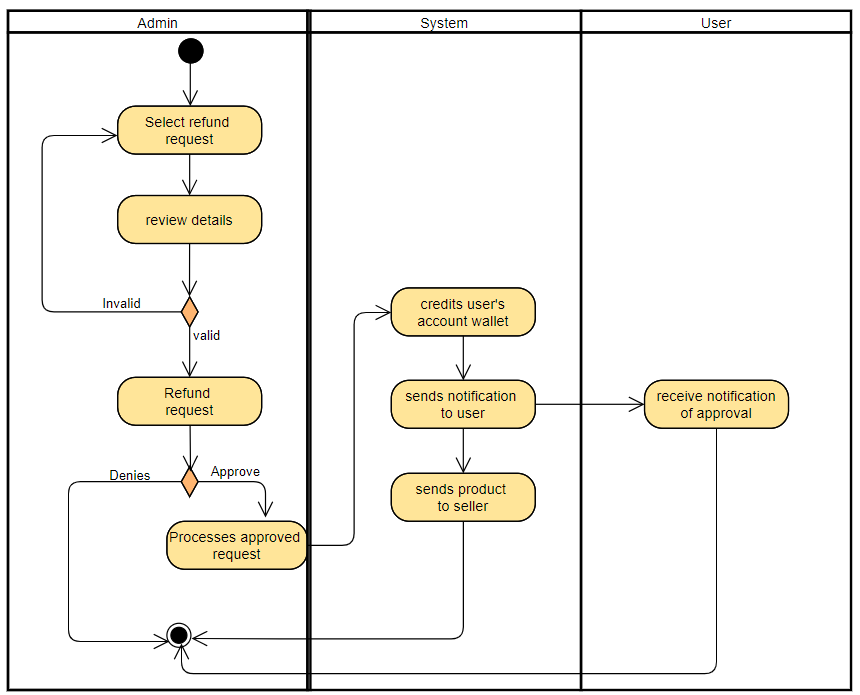
****

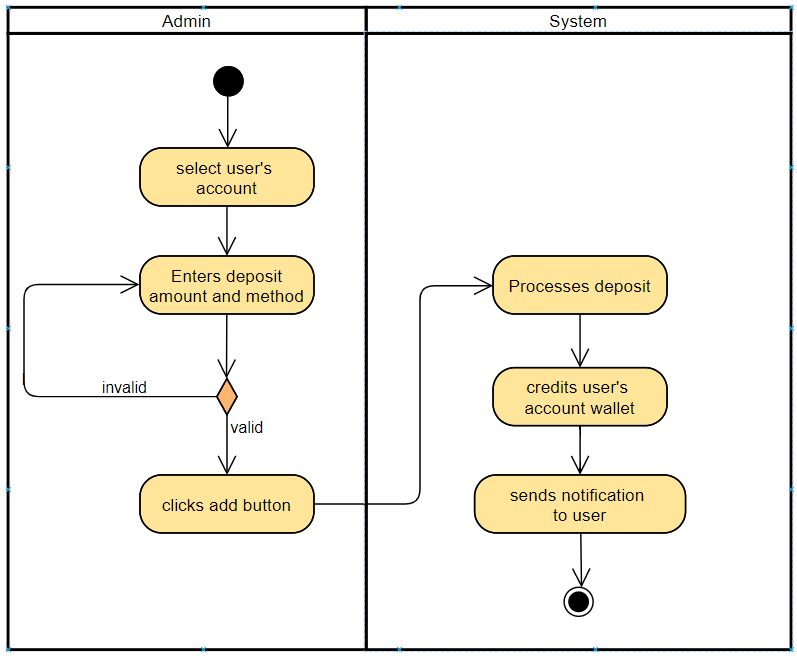
****

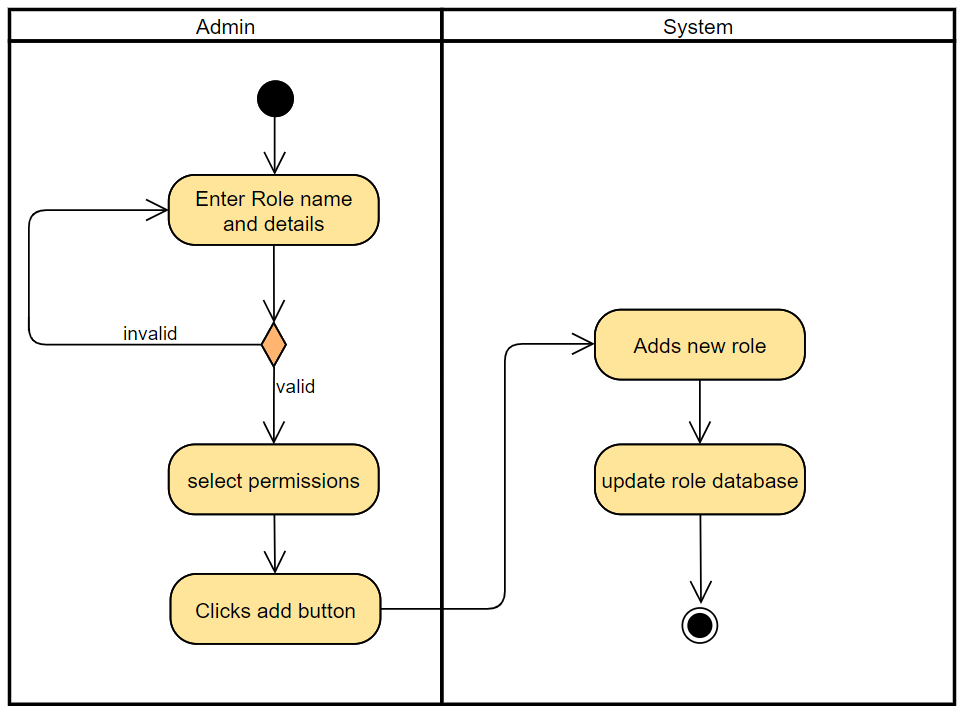
****

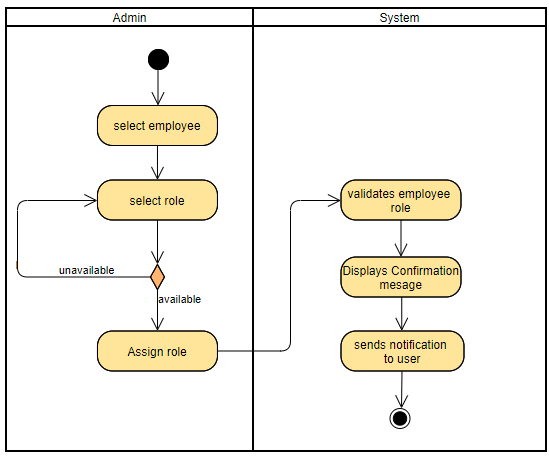
****

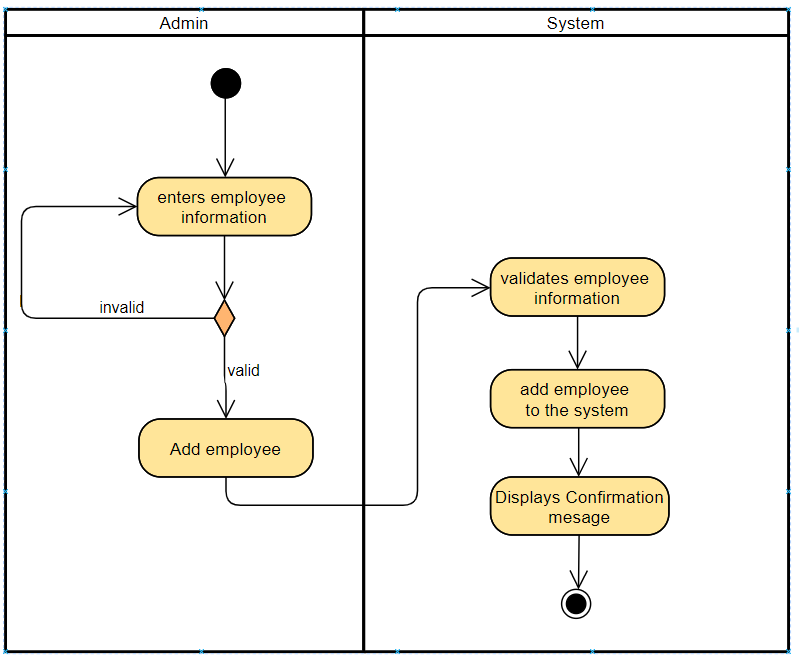
****

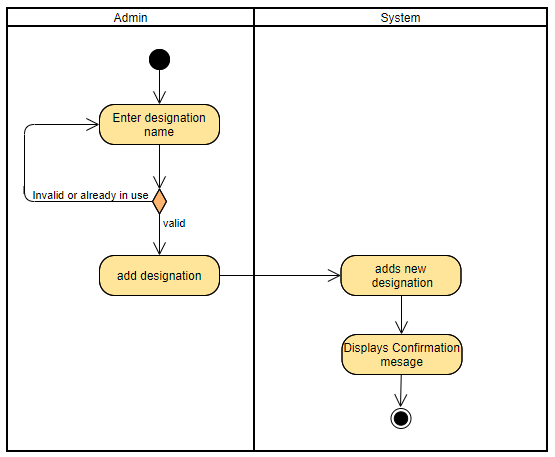
****

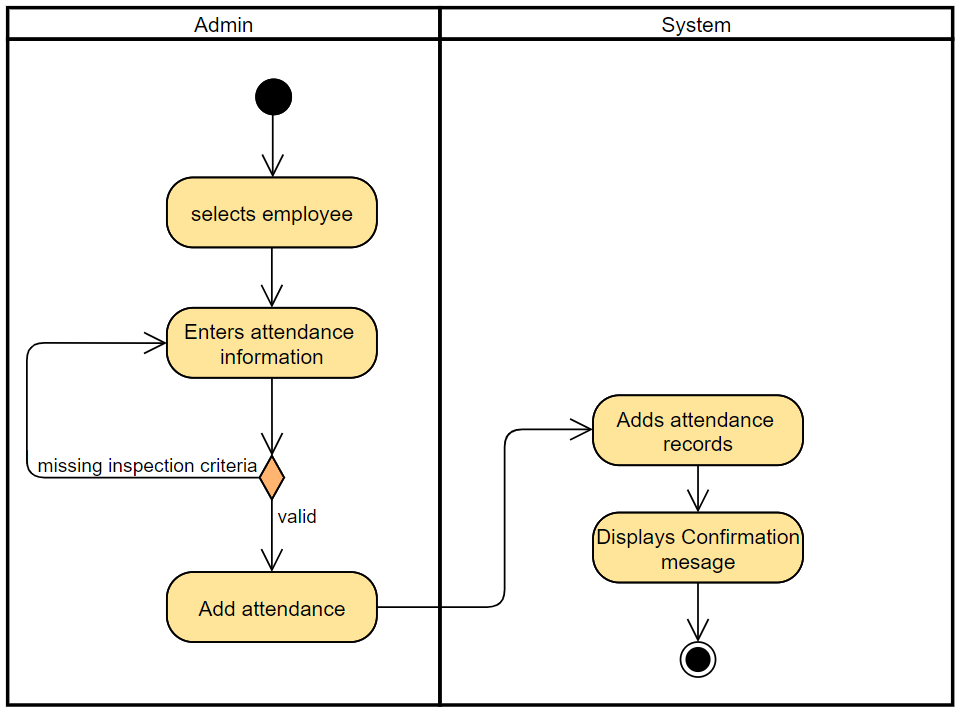
****

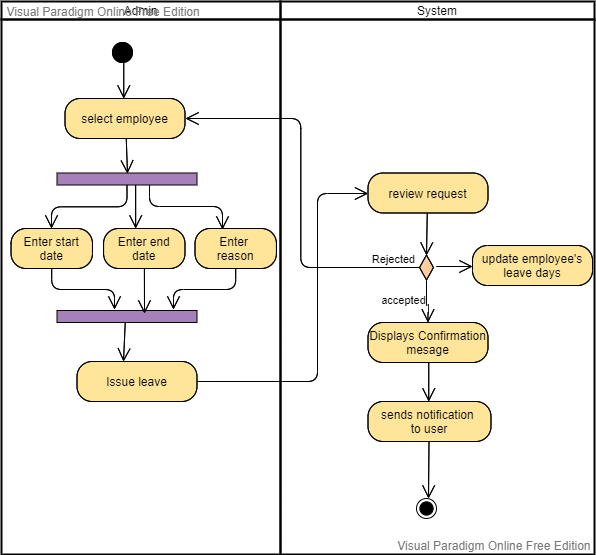
****

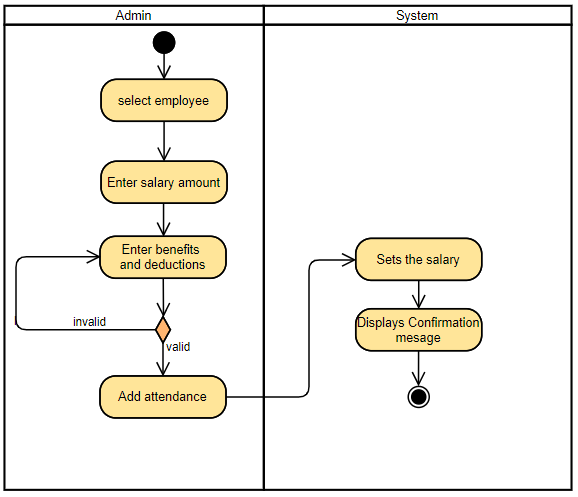
****

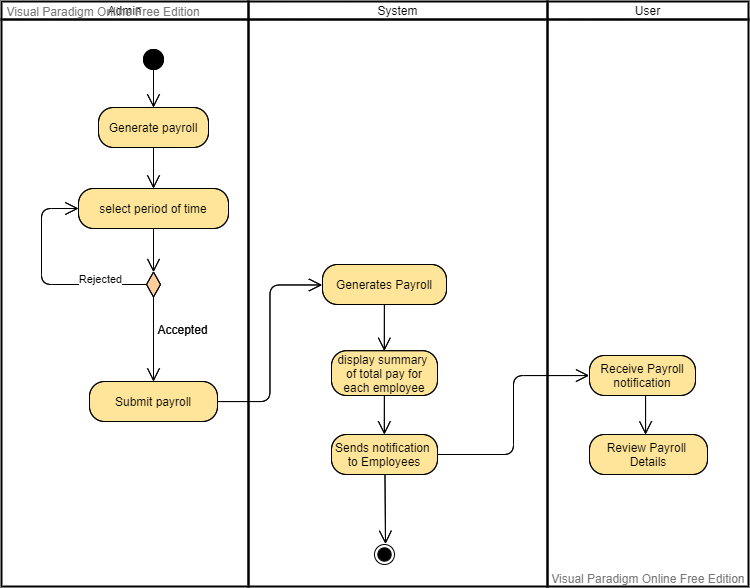
****

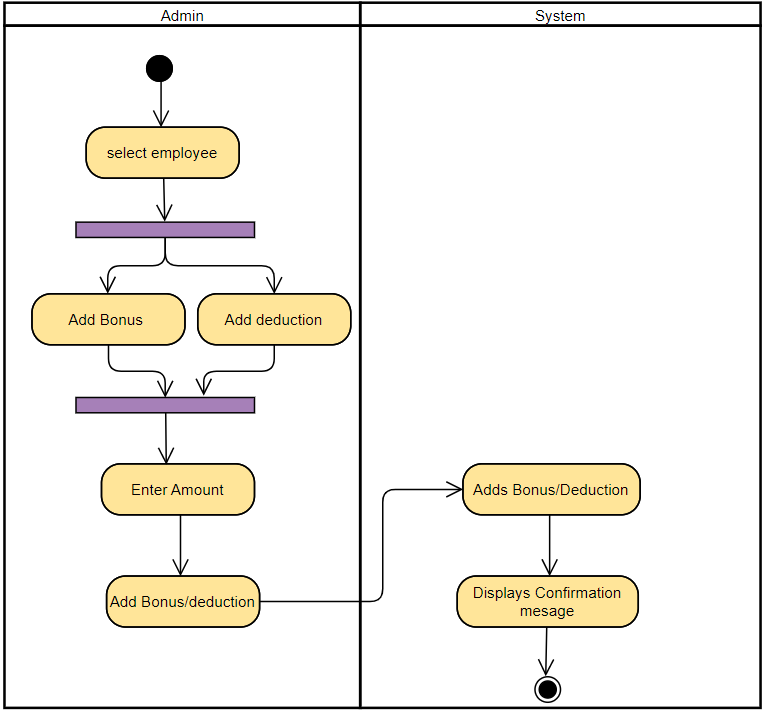
****

****

****

****

****



### **4.3.4 Test Case Design**

**For User**

1. **User login:**

|  |  |
| --- | --- |
| ID | ULD-1 |
| Use case | 1.1 |
| Form | User Login |
| Stakeholder | User |
| Field | username, password |
| Technique | Equivalence Class Partitioning (ECP) |
| Valid | {a,b,c, …..,z}  {0,1,2,3,…..} |
| Invalid | {!,@,#,$,%,^,&,\*,…} {<,>,=,….} |

Class: username  
Valid:  
 {a,b,c,….,z}  
 {0,1,2,……}

Invalid:  
 {!,@,#,$,%,^,&,\*,…}  
 {<,>,=,….}

|  |  |  |  |
| --- | --- | --- | --- |
| Scenario | username | ECP | Expected Output |
| Unsuccessfully login | johndoe! | Invalid | error! |
| Successfully login | johndoe | Valid | Successful login! |

Class: password  
Valid:  
 {a,b,c,….,z}  
 {0,1,2,……}

{!,@,#,$,%,^,&,\*,…}

Invalid:  
 {<,>,=,….}

|  |  |  |  |
| --- | --- | --- | --- |
| Scenario | password | ECP | Expected Output |
| Unsuccessfully login | password123= | Invalid | Error! |
| Successfully login | password@123 | Valid | Login Successful! |

**2. User signup:**

|  |  |
| --- | --- |
| ID | SU-2 |
| Use case | 1.2 |
| Form | Signup |
| Stakeholder | User |
| Field | username, password, email |
| Technique | Equivalence Class Partitioning (ECP) |
| Valid | {a,b,c, …..,z}  {0,1,2,3,…..} |
| Invalid | {!,@,#,$,%,^,&,\*,…} {<,>,=,….} |

Class: username

Valid:

{a,b,c,….,z}

{0,1,2,……}

Invalid:

{!,@,#,$,%,^,&,\*,…}

{<,>,=,….}

|  |  |  |  |
| --- | --- | --- | --- |
| Scenario | username | ECP | Expected Output |
| Unsuccessfully signup | User@ | Invalid | Error! |
| Successfully signup | JohnDoe | Valid | Signup successful! |
| Successfully signup | JohnDoe | Valid | Signup successful! |

Class: email

Valid:

{a,b,c,….,z}

{0,1,2,……}

{@, .}

Invalid:

{!,#,$,%,^,&,\*,…}

{<,>,=,….}

|  |  |  |  |
| --- | --- | --- | --- |
| Scenario | email | ECP | Expected Output |
| Unsuccessfully signup | user#gmail.com | Invalid | Error! |
| Successfully signup | john.doe@gmail.com | Valid | Signup successful! |
| Successfully signup | jane.smith@yahoo.com | Valid | Signup successful! |

Class: password

Valid:

{0,1,2,……}

{a,b,c,….,z}

{!,@,#,$,%,^,&,\*,…}

Invalid:

{<,>,=,….}

|  |  |  |  |
| --- | --- | --- | --- |
| Scenario | password | ECP | Expected Output |
| Unsuccessfully signup | pass@word= | Invalid | Error! |
| Successfully signup | password123 | Valid | Signup successful! |
| Successfully signup | p@ssword456 | Valid | Signup successful! |

**3. Forget password:**

|  |  |
| --- | --- |
| ID | FP-3 |
| Use case | 1.3 |
| Form | Forget Password |
| Stakeholder | User |
| Field | email |
| Technique | Equivalence Class Partitioning (ECP) |
| Valid | {a,b,c, …..,z}  {0,1,2,3,…..}  {@,.,\_} |
| Invalid | {!,#,$,%,^,&,\*,…} {<,>,=,….} |

Class: Email

Valid:

{a,b,c,….,z}

{0,1,2,……}

{@,.,\_}

Invalid:

{!,#,$,%,^,&,\*,…}

{<,>,=,….}

|  |  |  |  |
| --- | --- | --- | --- |
| Scenario | Email | ECP | Expected Output |
| Successfully email sent | abc@gmail.com | Valid | Password reset email sent |
| Successfully email sent | ab.c@gmail.com | Valid | Password reset email sent |
| Unsuccessfully email sent | ab\*c@gmail.com | Invalid | Invalid email address |

**4. Add product:**

|  |  |
| --- | --- |
| ID | APD-4 |
| Use case | 1.4 |
| Form | Add Product |
| Stakeholder | User |
| Field | product\_name, product\_category, product\_price, product\_quantity, product\_description |
| Technique | Equivalence Class Partitioning (ECP) |
| Valid | {a,b,c, …..,z}  {0,1,2,3,…..} |
| Invalid | {!,@,#,$,%,^,&,\*,…} {<,>,=,….} |

Class: product\_name

Valid:  
 {a,b,c,….,z}  
 {0,1,2,……}

Invalid:  
 {!,@,#,$,%,^,&,\*,…}  
 {<,>,=,….}

|  |  |  |  |
| --- | --- | --- | --- |
| Scenario | product\_name | ECP | Expected Output |
| Successfully product added | Laptop | Valid | Added! |
| Successfully product added | Tablet | Valid | Added! |
| Unsuccessfully product added | !pad | InValid | Error! |

Class: product\_category

Valid:

{a,b,c,….,z}

{0,1,2,……}

Invalid:

{!,@,#,$,%,^,&,\*,…}

{<,>,=,….}

|  |  |  |  |
| --- | --- | --- | --- |
| Scenario | product\_category | ECP | Expected Output |
| Successfully product added | Electronics | Valid | Product Added! |
| Successfully product added | Fashion | Valid | Product Added! |
| Unsuccessfully product added | !Electronics | Invalid | Error! |

Class: product\_price

Valid:

{0,1,2,……}

Invalid:

{a,b,c,….,z}

{!,@,#,$,%,^,&,\*,…}

{<,>,=,….}

|  |  |  |  |
| --- | --- | --- | --- |
| Scenario | product\_price | ECP | Expected Output |
| Successfully product added | 499 | Valid | Product Added! |
| Successfully product added | 299 | Valid | Product Added! |
| Unsuccessfully product added | $99 | Invalid | Error! |

Class: product\_quantity

Valid:

{0,1,2,……}

Invalid:

{a,b,c,….,z}

{!,@,#,$,%,^,&,\*,…}

{<,>,=,….}

|  |  |  |  |
| --- | --- | --- | --- |
| Scenario | product\_quantity | ECP | Expected Output |
| Successfully product added | 10 | Valid | Product Added! |
| Successfully product added | 20 | Valid | Product Added! |
| Unsuccessfully product added | one | Invalid | Error! |

Class: product\_description

Valid:

{a,b,c,….,z}

{0,1,2,……}

Invalid:

{!,@,#,$,%,^,&,\*,…}

{<,>,=,….}

|  |  |  |  |
| --- | --- | --- | --- |
| Scenario | product\_description | ECP | Expected Output |
| Successfully product added | 14 inch Full HD display | Valid | Product Added! |
| Successfully product added | Stylish and comfortable clothing | Valid | Product Added! |
| Unsuccessfully product added | 14 inch Full HD display! | Invalid | Error! |

**5. Add product to cart:**

|  |  |
| --- | --- |
| ID | APC-5 |
| Use case | 1.5 |
| Form | Add Product to cart |
| Stakeholder | User |
| Field | product\_name, product\_price, product\_quantity |
| Technique | Equivalence Class Partitioning (ECP) |
| Valid | {a,b,c, …..,z}  {0,1,2,3,…..} |
| Invalid | {!,@,#,$,%,^,&,\*,…} {<,>,=,….} |

Class: product\_name

Valid:

{a,b,c,….,z}

{0,1,2,……}

Invalid:

{!,@,#,$,%,^,&,,…}

{<,>,=,….}

|  |  |  |  |
| --- | --- | --- | --- |
| Scenario | product\_name | ECP | Expected Output |
| Successfully product added | iPhone 12 | Valid | Product Added to cart! |
| Successfully product added | Samsung s20 | Valid | Product Added to cart! |
| Unsuccessfully product added | iPD | Invalid | Error! |

Class: product\_price

Valid:

{0,1,2,……}

Invalid:

{a,b,c,….,z}

{!,@,#,$,%,^,&,\*,…}

{<,>,=,….}

|  |  |  |  |
| --- | --- | --- | --- |
| Scenario | product\_price | ECP | Expected Output |
| Successfully product added | 799 | Valid | Product Added to cart! |
| Successfully product added | 899 | Valid | Product Added to cart! |
| Unsuccessfully product added | $200 | Invalid | Error! |

Class: product\_quantity

Valid:

{0,1,2,……}

Invalid:

{a,b,c,….,z}

{!,@,#,$,%,^,&,\*,…}

{<,>,=,….}

|  |  |  |  |
| --- | --- | --- | --- |
| Scenario | product\_quantity | ECP | Expected Output |
| Successfully product added | 2 | Valid | Product Added to cart! |
| Successfully product added | 3 | Valid | Product Added to cart! |
| Unsuccessfully product added | one | Invalid | Error! |

**6. Send Price Offer:**

|  |  |
| --- | --- |
| ID | SPO-6 |
| Use case | 1.6 |
| Form | Send Price Offer |
| Stakeholder | User |
| Field | receiver\_username, mobile\_name, offered\_price |
| Technique | Equivalence Class Partitioning (ECP) |
| Valid | {a,b,c, …..,z}  {0,1,2,3,…..} |
| Invalid | {!,@,#,$,%,^,&,\*,…} {<,>,=,….} |

Class: receiver\_username

Valid:

{a,b,c,….,z}

{0,1,2,……}

Invalid:

{!,@,#,$,%,^,&,\*,…}

{<,>,=,….}

|  |  |  |  |
| --- | --- | --- | --- |
| Scenario | receiver\_username | ECP | Expected Output |
| Unsuccessfully send offer | Us#er@123 | Invalid | Error! |
| Successfully send offer | User123 | Valid | Offer Sent! |
| Successfully send offer | AnotherUser | Valid | Offer Sent! |

Class: mobile\_name

Valid:

{a,b,c,….,z}

{0,1,2,……}

Invalid:

{!,@,#,$,%,^,&,,…}

{<,>,=,….}

|  |  |  |  |
| --- | --- | --- | --- |
| Scenario | mobile\_name | ECP | Expected Output |
| Successfully send offer | iPhone 12 Pro Max | Valid | Offer Sent! |
| Successfully send offer | Samsung Galaxy S21 | Valid | Offer Sent! |
| Unsuccessfully send offer | !@#$%^& | Invalid | Error! |

Class: offered\_price

Valid:

{0,1,2,……}

Invalid:

{a,b,c,….,z}

{!,@,#,$,%,^,&,\*,…}

{<,>,=,….}

|  |  |  |  |
| --- | --- | --- | --- |
| Scenario | offered\_price | ECP | Expected Output |
| Successfully send offer | 1000 | Valid | Offer Sent! |
| Successfully send offer | 999 | Valid | Offer Sent! |
| Unsuccessfully send offer | 1abc | Invalid | Error! |

**7. Add to Compare List:**

|  |  |
| --- | --- |
| ID | ACL-7 |
| Use case | 1.7 |
| Form | Add to Compare List |
| Stakeholder | User |
| Field | product\_name, product\_brand, product\_specification |
| Technique | Equivalence Class Partitioning (ECP) |
| Valid | {a,b,c, …..,z}  {0,1,2,3,…..} |
| Invalid | {!,@,#,$,%,^,&,\*,…} {<,>,=,….} |

Class: product\_name

Valid:

{a,b,c,….,z}

{0,1,2,……}

Invalid:

{!,@,#,$,%,^,&,,…}

{<,>,=,….}

|  |  |  |  |
| --- | --- | --- | --- |
| Scenario | product\_name | ECP | Expected Output |
| Successfully product added | iPhone 12 | Valid | Product Added to compare list! |
| Successfully product added | Samsung s12 | Valid | Product Added to compare list! |
| Unsuccessfully product added | iPD | Invalid | Error! |

Class: product\_brand

Valid:

{a,b,c,….,z}

{0,1,2,……}

Invalid:

{!,@,#,$,%,^,&,\*,…}

{<,>,=,….}

|  |  |  |  |
| --- | --- | --- | --- |
| Scenario | product\_brand | ECP | Expected Output |
| Successfully product added | Apple | Valid | Product Added to compare list! |
| Successfully product added | Samsung | Valid | Product Added to compare list! |
| Unsuccessfully product added | App!e | Invalid | Error! |

Class: product\_specification

Valid:

{a,b,c,….,z}

{0,1,2,……}

Invalid:

{!,@,#,$,%,^,&,\*,…}

{<,>,=,….}

|  |  |  |  |
| --- | --- | --- | --- |
| Scenario | product\_specification | ECP | Expected Output |
| Successfully product added | 64gb | Valid | Product Added to compare list! |
| Successfully product added | 128gb | Valid | Product Added to compare list! |
| Unsuccessfully product added | 6#gb | Invalid | Error! |

1. **Search product:**

|  |  |
| --- | --- |
| ID | SPD-8 |
| Use case | 1.8 |
| Form | Search for Mobile |
| Stakeholder | User |
| Field | search\_query |
| Technique | Equivalence Class Partitioning (ECP) |
| Valid | {a,b,c, …..,z}  {0,1,2,3,…..} |
| Invalid | {!,@,#,$,%,^,&,\*,…} {<,>,=,….} |

Class: search\_query

Valid:

{a,b,c,….,z}

{0,1,2,……}

Invalid:

{!,@,#,$,%,^,&,\*,…}

{<,>,=,….}

|  |  |  |  |
| --- | --- | --- | --- |
| Scenario | search\_query | ECP | Expected Output |
| Successfully search | iPhone X | Valid | Results Found |
| Successfully search | Samsung S20 | Valid | Results Found |
| Unsuccessfully search | Iph#ne X | Invalid | Error! |
| Unsuccessfully search | Sam$ung S20 | Invalid | Error! |

1. **Search Filter:**

|  |  |
| --- | --- |
| ID | SF-9 |
| Use case | 1.9 |
| Form | Search Filter |
| Stakeholder | User |
| Field | product\_category, product\_price, product\_brand |
| Technique | Equivalence Class Partitioning (ECP) |
| Valid | {a,b,c, …..,z}  {0,1,2,3,…..} |
| Invalid | {!,@,#,$,%,^,&,\*,…} {<,>,=,….} |

Class: product\_category

Valid:

{a,b,c,….,z}

{0,1,2,……}

Invalid:

{!,@,#,$,%,^,&,\*,…}

{<,>,=,….}

|  |  |  |  |
| --- | --- | --- | --- |
| Scenario | product\_category | ECP | Expected Output |
| Successfully product added | Smart phone | Valid | Products filtered by category "Smartphones" |
| Successfully product added | Tablets | Valid | Products filtered by category " Tablets " |
| Unsuccessfully product added | !Electronics | Invalid | Error! |

Class: product\_price

Valid:

{0,1,2,……}

Invalid:

{a,b,c,….,z}

{!,@,#,$,%,^,&,\*,…}

{<,>,=,….}

|  |  |  |  |
| --- | --- | --- | --- |
| Scenario | product\_price | ECP | Expected Output |
| Successfully product added | 799 | Valid | Products filtered by price "799" |
| Successfully product added | 899 | Valid | Products filtered by price "899" |
| Unsuccessfully product added | $200 | Invalid | Error! |

Class: product\_brand

Valid:

{a,b,c,….,z}

{0,1,2,……}

Invalid:

{!,@,#,$,%,^,&,\*,…}

{<,>,=,….}

|  |  |  |  |
| --- | --- | --- | --- |
| Scenario | product\_brand | ECP | Expected Output |
| Successfully product added | Apple | Valid | Products filtered by brand "Apple" |
| Successfully product added | Samsung | Valid | Products filtered by brand "Samsung" |
| Unsuccessfully product added | App!e | Invalid | Error! |

1. **Edit cart:**

|  |  |
| --- | --- |
| ID | EC |
| Use case | 1.10 |
| Form | Edit cart |
| Stakeholder | User |
| Field | product\_id, product\_quantity |
| Technique | Equivalence Class Partitioning (ECP) |
| Valid | {a,b,c, …..,z}  {0,1,2,3,…..} |
| Invalid | {!,@,#,$,%,^,&,\*,…} {<,>,=,….} |

Class: product\_id

Valid:

{0,1,2,…,9}

Invalid:

{a,b,c,….,z}

{!,@,#,$,%,^,&,\*,…}

{<,>,=,….}

|  |  |  |  |
| --- | --- | --- | --- |
| Scenario | product\_id | ECP | Expected Output |
| Successfully edited | 12345 | Valid | Product with id "12345" successfully edited in the cart |
| Successfully edited | 67890 | Valid | Product with id "67890" successfully edited in the cart |
| Unsuccessfully edited | one | Invalid | Error! |

Class: product\_quantity

Valid:

{0,1,2,……}

Invalid:

{a,b,c,….,z}

{!,@,#,$,%,^,&,\*,…}

{<,>,=,….}

|  |  |  |  |
| --- | --- | --- | --- |
| Scenario | product\_quantity | ECP | Expected Output |
| Successfully edited | 2 | Valid | Product quantity successfully edited to "2" in the cart |
| Successfully edited | 5 | Valid | Product quantity successfully edited to "5" in the cart |
| Unsuccessfully edited | one | Invalid | Error! |

1. **Apply Coupon:**

|  |  |
| --- | --- |
| ID | AC-11 |
| Use case | 1.11 |
| Form | Apply Coupon |
| Stakeholder | User |
| Field | coupon\_code |
| Technique | Equivalence Class Partitioning (ECP) |
| Valid | {a,b,c,….,z}  {0,1,2,…,9} |
| Invalid | {!,@,#,$,%,^,&,\*,…}  {<,>,=,….} |

Class: coupon\_code

Valid:

{a,b,c,….,z}

{0,1,2,…,9}

Invalid:

{!,@,#,$,%,^,&,\*,…}

{<,>,=,….}

|  |  |  |  |
| --- | --- | --- | --- |
| Scenario | coupon\_code | ECP | Expected Output |
| Successfully applied | COUPON 123 | Valid Coupon | "COUPON123" successfully applied |
| Successfully applied COUPON | SUMMER2022 | Valid Coupon | "SUMMER2022" successfully applied |
| Unsuccessfully applied COUPON | @#$$ | Invalid | Error! |

1. **Checkout:**

|  |  |
| --- | --- |
| ID | COP-12 |
| Use case | 1.12 |
| Form | Checkout |
| Stakeholder | User |
| Field | shipping\_address, payment\_method, item\_quantity |
| Technique | Equivalence Class Partitioning (ECP) |
| Valid | {a-z, A-Z, 0-9, #, -, ., /} |
| Invalid | {!, @, $, %, ^, &, \*} |

Class: shipping\_address

Valid:

{a-z, A-Z, 0-9, #, -, ., /}

Invalid:

{!, @, $, %, ^, &, \*}

|  |  |  |  |
| --- | --- | --- | --- |
| Scenario | shipping\_address | ECP | Expected Output |
| Successfully checkout | 1234 Main St. | Valid | Checkout Successful |
| Unsuccessfully checkout | 1234 Main St! | Invalid | Error: Invalid Address |

Class: payment\_method

Valid:

{Credit Card, Debit Card, PayPal, Bank Transfer}

Invalid:

{other than the valid options}

|  |  |  |  |
| --- | --- | --- | --- |
| Scenario | payment\_method | ECP | Expected Output |
| Successfully checkout | Credit | Valid | Checkout Successful |
| Unsuccessfully checkout | Cash | Invalid | Error: Invalid Payment Method |

Class: item\_quantity

Valid:

{positive integer}

Invalid:

{negative number, 0, non-integer}

|  |  |  |  |
| --- | --- | --- | --- |
| Scenario | payment\_method | ECP | Expected Output |
| Successfully checkout | Checkout 2 | Valid | Checkout Successful |
| Unsuccessfully checkout | Checkout -2 | Invalid | Error: Invalid Quantity |
| Unsuccessfully checkout | Checkout 0 | Invalid | Error: Invalid Quantity |
| Unsuccessfully checkout | Checkout two | Invalid | Error: Invalid Quantity |

**For Rider**

1. **Accept Order for rider:**

|  |  |
| --- | --- |
| ID | AOR-1 |
| Use case | 1.1 |
| Form | Accept Order |
| Stakeholder | Rider |
| Field | Order\_id, Rider\_id, Order\_status |
| Technique | Equivalence Class Partitioning (ECP) |
| Valid | {a,b,c, …..,z}  {0,1,2,3,…..} |
| Invalid | {!,@,#,$,%,^,&,\*,…} {<,>,=,….} |

Class: Order\_id

Valid:

{0,1,2,……}

Invalid:

{a,b,c,….,z}

{!,@,#,$,%,^,&,\*,…}

{<,>,=,….}

|  |  |  |  |
| --- | --- | --- | --- |
| Scenario | Order\_id | ECP | Expected Output |
| Successfully Order accepted | 235 | Valid | Accepted! |
| Unsuccessfully Order accepted | abc | Invalid | Error! |

Class: Rider\_id

Valid:

{0,1,2,……}

Invalid:

{a,b,c,….,z}

{!,@,#,$,%,^,&,\*,…}

{<,>,=,….}

|  |  |  |  |
| --- | --- | --- | --- |
| Scenario | Rider\_id | ECP | Expected Output |
| Successfully Order accepted | 12 | Valid | Accepted! |
| Unsuccessfully Order accepted | xyz | Invalid | Error! |

Class: Order\_status

Valid:

{Accepted, Declined}

Invalid:

{a,b,c,….,z}

{0,1,2,……}

{!,@,#,$,%,^,&,\*,…}

{<,>,=,….}

|  |  |  |  |
| --- | --- | --- | --- |
| Scenario | Order\_status | ECP | Expected Output |
| Successfully Order accepted | Accepted | Valid | Accepted! |
| Unsuccessfully Order accepted | Delivered | Invalid | Error! |

**For Admin**

1. **Add user:**

|  |  |
| --- | --- |
| ID | ATC-1 |
| Use case | 3.1 |
| Form | Add User |
| Stakeholder | Admin |
| Field | username, password, email, role |
| Technique | Equivalence Class Partitioning (ECP) |
| Valid | {a,b,c, …..,z}  {0,1,2,3,…..} |
| Invalid | {!,@,#,$,%,^,&,\*,…} {<,>,=,….} |

Class: username

**Valid:**

{a,b,c,….,z}

{0,1,2,……}

**Invalid:**

{!,@,#,$,%,^,&,\*,…}

{<,>,=,….}

|  |  |  |  |
| --- | --- | --- | --- |
| Scenario | username | ECP | Expected Output |
| Unsuccessfully | johndoe! | Invalid | Error! |
| Successfully login | johndoe | Valid | User added successfully! |

Class: password  
Valid:  
 {a,b,c,….,z}  
 {0,1,2,……}

{!,@,#,$,%,^,&,\*,…}

Invalid:  
 {<,>,=,….}

|  |  |  |  |
| --- | --- | --- | --- |
| Scenario | password | ECP | Expected Output |
| Unsuccessfully login | password123= | Invalid | Error! |
| Successfully login | password@123 | Valid | User added successfully! |

Class: email

**Valid:**

{a,b,c,…,z}

{0,1,2,3,….}

{@, .}

**Invalid:**

{!,#,$,%,^,&,\*,…}

{<,>,=,….}

|  |  |  |  |
| --- | --- | --- | --- |
| Scenario | email | ECP | Expected Output |
| Unsuccessfully login | johndoe#gmail.com | Invalid | Error! |
| Successfully login | johndoe@gmail.com | Valid | User added successfully! |

1. **Manage Product:**

|  |  |
| --- | --- |
| ID | MPD-2 |
| Use case | 3.2 |
| Form | Manage Product |
| Stakeholder | Admin |
| Field | product\_name, product\_brand, product\_price |
| Technique | Equivalence Class Partitioning (ECP) |
| Valid | {a,b,c, …..,z}  {0,1,2,3,…..} |
| Invalid | {!,@,#,$,%,^,&,\*,…} {<,>,=,….} |

Class: product\_name

Valid:

{a,b,c,….,z}

{0,1,2,……}

Invalid:

{!,@,#,$,%,^,&,\*,…}

{<,>,=,….}

|  |  |  |  |
| --- | --- | --- | --- |
| Scenario | Product\_name | ECP | Expected Output |
| Unsuccessfully product added | iPhon@ | Invalid | Error! |
| Successfully product added | iPhone XR | Valid | Product Added! |
| Successfully product added | Samsung S21 | Valid | Product Added! |

Class: product\_brand

Valid:

{a,b,c,….,z}

{0,1,2,……}

Invalid:

{!,@,#,$,%,^,&,\*,…}

{<,>,=,….}

|  |  |  |  |
| --- | --- | --- | --- |
| Scenario | product\_brand | ECP | Expected Output |
| Unsuccessfully product added | #XIAOMI | Invalid | Error! |
| Successfully product added | iPhone XR | Valid | Product Added! |
| Successfully product added | Samsung S21 | Valid | Product Added! |

Class: product\_price

Valid:

{0,1,2,……}

Invalid:

{a,b,c,….,z}

{!,@,#,$,%,^,&,\*,…}

{<,>,=,….}

|  |  |  |  |
| --- | --- | --- | --- |
| Scenario | product\_price | ECP | Expected Output |
| Unsuccessfully product added | seven-hundred | Invalid | Error! |
| Successfully product added | 799 | Valid | Product Added! |
| Successfully product added | 900 | Valid | Product Added! |

1. **Boost Product:**

|  |  |
| --- | --- |
| ID | BPD-3 |
| Use case | 3.3 |
| Form | Boost Product |
| Stakeholder | Admin |
| Field | product\_id, boost\_duration, boost\_amount |
| Technique | Equivalence Class Partitioning (ECP) |
| Valid | {0,1,2,3,…..} |
| Invalid | {!,@,#,$,%,^,&,\*,…} {<,>,=,….}  {a,b,c, …..,z} |

Class: product\_id

Valid:

{0,1,2,……}

Invalid:

{a,b,c,….,z}

{!,@,#,$,%,^,&,\*,…}

{<,>,=,….}

|  |  |  |  |
| --- | --- | --- | --- |
| Scenario | product\_id | ECP | Expected Output |
| Successfully product boosted | 111 | Valid | Product Boosted! |
| Successfully product boosted | 555 | Valid | Product Boosted! |
| Unsuccessfully product boosted | One hunderd | Invalid | Error! |

Class: boost\_duration

Valid:

{0,1,2,……}

Invalid:

{a,b,c,….,z}

{!,@,#,$,%,^,&,\*,…}

{<,>,=,….}

|  |  |  |  |
| --- | --- | --- | --- |
| Scenario | boost\_duration | ECP | Expected Output |
| Successfully product boosted | 3 | Valid | Product Boosted! |
| Successfully product boosted | 7 | Valid | Product Boosted! |
| Unsuccessfully product boosted | Two weeks | Invalid | Error! |

Class: boost\_amount

Valid:

{0,1,2,……}

Invalid:

{a,b,c,….,z}

{!,@,#,$,%,^,&,\*,…}

{<,>,=,….}

|  |  |  |  |
| --- | --- | --- | --- |
| Scenario | boost\_amount | ECP | Expected Output |
| Successfully product boosted | 300 | Valid | Product Boosted! |
| Successfully product boosted | 700 | Valid | Product Boosted! |
| Unsuccessfully product boosted | Two hundred | Invalid | Error! |

1. **Add Category:**

|  |  |
| --- | --- |
| ID | ADC-4 |
| Use case | 3.4 |
| Form | Add Category |
| Stakeholder | Admin |
| Field | category\_name |
| Technique | Equivalence Class Partitioning (ECP) |
| Valid | {a,b,c, …..,z} |
| Invalid | {!,@,#,$,%,^,&,\*,…} {<,>,=,….}  {0,1,2,3,…..} |

Class: category\_name

Valid:

{a,b,c,….,z}

Invalid:

{0,1,2,3,…}

{!,@,#,$,%,^,&,\*,…}

{<,>,=,….}

|  |  |  |  |
| --- | --- | --- | --- |
| Scenario | category\_name | ECP | Expected Output |
| Successfully category added | Smartphones | Valid | Category Added! |
| Successfully category added | Laptops | Valid | Category Added! |
| Unsuccessfully category added | 123abc | Invalid | Error! |

1. **Add Brand:**

|  |  |
| --- | --- |
| ID | ADB-5 |
| Use case | 3.5 |
| Form | Add Brand |
| Stakeholder | Admin |
| Field | brand\_name |
| Technique | Equivalence Class Partitioning (ECP) |
| Valid | {a,b,c, …..,z} |
| Invalid | {!,@,#,$,%,^,&,\*,…} {<,>,=,….}  {0,1,2,3,…..} |

Class: brand\_name

Valid:

{a,b,c,….,z}

Invalid:

{0,1,2,3,…}

{!,@,#,$,%,^,&,\*,…}

{<,>,=,….}

|  |  |  |  |
| --- | --- | --- | --- |
| Scenario | brand\_name | ECP | Expected Output |
| Successfully brand added | Apple | Valid | Brand Added! |
| Successfully brand added | Samsung | Valid | Brand Added! |
| Unsuccessfully brand added | 123abc | Invalid | Error! |

1. **Inspect Product:**

|  |  |
| --- | --- |
| ID | INP-6 |
| Use case | 3.6 |
| Form | Inspect product |
| Stakeholder | Admin |
| Field | Product\_ID, product\_name, product\_description |
| Technique | Equivalence Class Partitioning (ECP) |

Class: product\_ID

Valid:

{0,1,2,3,……}

Invalid:

{!,@,#,$,%,^,&,\*,…}

{<,>,=,….}

{a,b,c,…,z}

|  |  |  |  |
| --- | --- | --- | --- |
| Scenario | Product\_ID | ECP | Expected Output |
| Unsuccessfully | product1@ | Invalid | Error! |
| Successfully inspect product | product 123 | Valid | Product information displayed successfully! |

Class: product\_name

Valid:

{a,b,c,….,z}

{0,1,2,……}

Invalid:

{!,@,#,$,%,^,&,\*,…}

{<,>,=,….}

|  |  |  |  |
| --- | --- | --- | --- |
| Scenario | Product\_name | ECP | Expected Output |
| Unsuccessfully inspect product | product!1 | Invalid | Error! |
| Successfully inspect product | product1 | Valid | Product information displayed successfully! |

Class: product\_description

Valid:

{a,b,c,…,z}

{0,1,2,3,….}

Invalid:

{!,@,#,$,%,^,&,\*,…}

{<,>,=,….}

|  |  |  |  |
| --- | --- | --- | --- |
| Scenario | Product\_description | ECP | Expected Output |
| Unsuccessfully inspect product | description! | Invalid | Error! |
| Successfully inspect product | description1 | Valid | Product information displayed successfully! |

1. **Edit Sale:**

|  |  |
| --- | --- |
| ID | ES-7 |
| Use case | 3.7 |
| Form | Edit Sale |
| Stakeholder | Admin |
| Field | product\_name, original\_price, sale\_price |
| Technique | Equivalence Class Partitioning (ECP) |
| Valid | {a,b,c, …..,z}  {0,1,2,3,…..} |
| Invalid | {!,@,#,$,%,^,&,\*,…} {<,>,=,….} |

Class: product\_name

Valid:

{a,b,c,….,z}

{0,1,2,……}

Invalid:

{!,@,#,$,%,^,&,\*,…}

{<,>,=,….}

|  |  |  |  |
| --- | --- | --- | --- |
| Scenario | product\_name | ECP | Expected Output |
| Successfully sale edited | Iphone | Valid | Sale Edited! |
| Successfully sale edited | Samsung | Valid | Sale Edited! |
| Unsuccessfully sale edited | 1#plus | Invalid | Error! |

Class: original\_price

Valid:

{0,1,2,……}

Invalid:

{a,b,c,….,z}

{!,@,#,$,%,^,&,\*,…}

{<,>,=,….}

|  |  |  |  |
| --- | --- | --- | --- |
| Scenario | original\_price | ECP | Expected Output |
| Successfully sale edited | 799 | Valid | Sale Edited! |
| Successfully sale edited | 299 | Valid | Sale Edited! |
| Unsuccessfully sale edited | Two hundred | Invalid | Error! |

Class: sale\_price

Valid:

{0,1,2,……}

Invalid:

{a,b,c,….,z}

{!,@,#,$,%,^,&,\*,…}

{<,>,=,….}

|  |  |  |  |
| --- | --- | --- | --- |
| Scenario | sale\_price | ECP | Expected Output |
| Successfully sale edited | 799 | Valid | Sale Edited! |
| Successfully sale edited | 299 | Valid | Sale Edited! |
| Unsuccessfully sale edited | Two hundred | Invalid | Error! |

1. **Approve Withdrawal:**

|  |  |
| --- | --- |
| ID | AWD-8 |
| Use case | 3.8 |
| Form | Approve Withdrawal |
| Stakeholder | Admin |
| Field | user\_id, amount, date |
| Technique | Equivalence Class Partitioning (ECP) |
| Valid | {a,b,c, …..,z}  {0,1,2,3,…..} |
| Invalid | {!,@,#,$,%,^,&,\*,…} {<,>,=,….} |

Class: user\_id

Valid:

{a,b,c,….,z}

{0,1,2,……}

Invalid:

{!,@,#,$,%,^,&,\*,…}

{<,>,=,….}

|  |  |  |  |
| --- | --- | --- | --- |
| Scenario | user\_id | ECP | Expected Output |
| Successfully withdrawal approved | User\_456 | Valid | withdrawal approved! |
| Successfully withdrawal approved | User\_299 | Valid | withdrawal approved! |
| Unsuccessfully withdrawal approved | User\_#1 | Invalid | Error! |

Class: amount

Valid:

{0,1,2,……}

Invalid:

{a,b,c,….,z}

{!,@,#,$,%,^,&,\*,…}

{<,>,=,….}

|  |  |  |  |
| --- | --- | --- | --- |
| Scenario | amount | ECP | Expected Output |
| Successfully withdrawal approved | 456 | Valid | withdrawal approved! |
| Successfully withdrawal approved | 299 | Valid | withdrawal approved! |
| Unsuccessfully withdrawal approved | Five hundred | Invalid | Error! |

Class: date

Valid:

{0,1,2,……}

Invalid:

{a,b,c,….,z}

{!,@,#,$,%,^,&,\*,…}

{<,>,=,….}

|  |  |  |  |
| --- | --- | --- | --- |
| Scenario | date | ECP | Expected Output |
| Successfully withdrawal approved | 06-02-2023 | Valid | withdrawal approved! |
| Successfully withdrawal approved | 05-02-2023 | Valid | withdrawal approved! |
| Unsuccessfully withdrawal approved | Feb 6 2023 | Invalid | Error! |

1. **Approve Refund:**

|  |  |
| --- | --- |
| ID | AR-8 |
| Use case | 3.8 |
| Form | Approve Refund |
| Stakeholder | Admin |
| Field | refund\_amount , customer\_id |
| Technique | Equivalence Class Partitioning (ECP) |
| Valid | {a,b,c, …..,z} |
| Invalid | {!,@,#,$,%,^,&,\*,…} {<,>,=,….}  {0,1,2,3,…..} |

Class: refund\_amount

Valid:

{0,1,2,……}

Invalid:

{a,b,c,….,z}

{!,@,#,$,%,^,&,\*,…}

{<,>,=,….}

|  |  |  |  |
| --- | --- | --- | --- |
| Scenario | refund\_amount | ECP | Expected Output |
| Successfully approved refund | 500 | Valid | approved refund! |
| Unsuccessfully approved refund | $556 | Invalid | Error! |

Class: customer\_id

Valid:

{0,1,2,……}

Invalid:

{a,b,c,….,z}

{!,@,#,$,%,^,&,\*,…}

{<,>,=,….}

|  |  |  |  |
| --- | --- | --- | --- |
| Scenario | customer\_id | ECP | Expected Output |
| Successfully approved refund | 12345 | Valid | approved refund! |
| Unsuccessfully approved refund | dfrsf | Invalid | Error! |

1. **Add Deposit:**

|  |  |
| --- | --- |
| ID | AD-10 |
| Use case | 3.10 |
| Form | Add deposit |
| Stakeholder | Admin |
| Field | Amount, Date, Description |
| Technique | Equivalence Class Partitioning (ECP) |

Class: Amount

Valid:

{0,1,2,3,……}

Invalid:

{a,b,c,….,z}

{!,@,#,$,%,^,&,\*,…}

{<,>,=,….}

{-,+}

|  |  |  |  |
| --- | --- | --- | --- |
| Scenario | Amount | ECP | Expected Output |
| Invalid amount | abc | Invalid | error! |
| Valid amount | 100 | Valid | Deposit Added Successfully! |

Class: Date

**Valid:**

dd/mm/yyyy

**Invalid:**

{a,b,c,….,z}

{!,@,#,$,%,^,&,\*,…}

{<,>,=,….}

dd-mm-yyyy

dd.mm.yyyy

|  |  |  |  |
| --- | --- | --- | --- |
| Scenario | Date | ECP | Expected Output |
| Invalid date | 12-02-2022 | Invalid | Error! |
| Valid date | 12/02/2022 | Valid | Deposit Added Successfully! |

Class: Description

**Valid:**

{a,b,c,….,z}

{0,1,2,……}

{!,@,#,$,%,^,&,\*,…}

{<,>,=,….}

**Invalid:**

{NULL}

|  |  |  |  |
| --- | --- | --- | --- |
| Scenario | Description | ECP | Expected Output |
| Invalid description | NULL | Invalid | Error! |
| Valid description | Deposit for purchase | Valid | Deposit Added Successfully! |

1. **Add role:**

|  |  |
| --- | --- |
| ID | AR-11 |
| Use case | 3.11 |
| Form | Add role |
| Stakeholder | Admin |
| Field | role\_name , role\_description |
| Technique | Equivalence Class Partitioning (ECP) |
| Valid | {a,b,c, …..,z} |
| Invalid | {!,@,#,$,%,^,&,\*,…} {<,>,=,….}  {0,1,2,3,…..} |

Class: role\_name

Valid:

{a,b,c,….,z}

Invalid:

{0,1,2,3,…}

{!,@,#,$,%,^,&,\*,…}

{<,>,=,….}

|  |  |  |  |
| --- | --- | --- | --- |
| Scenario | role\_name | ECP | Expected Output |
| Successfully added role | Manager | Valid | added role! |
| Unsuccessfully added role | 12Manage | Invalid | Error! |

Class: role\_description

Valid:

{a,b,c,….,z}

Invalid:

{0,1,2,3,…}

{!,@,#,$,%,^,&,\*,…}

{<,>,=,….}

|  |  |  |  |
| --- | --- | --- | --- |
| Scenario | role\_description | ECP | Expected Output |
| Successfully added role | Manages operations | Valid | added role! |
| Unsuccessfully added role | Manage operation# | Invalid | Error! |

1. **Assign role:**

|  |  |
| --- | --- |
| ID | AR-12 |
| Use case | 3.12 |
| Form | Assign role |
| Stakeholder | Admin |
| Field | user\_id, role\_id |
| Technique | Equivalence Class Partitioning (ECP) |
| Valid | {0,1,2,3,…..} |
| Invalid | {!,@,#,$,%,^,&,\*,…} {<,>,=,….}  {a,b,c, …..,z} |

Class: user\_id

Valid:

{0,1,2,3,…}

Invalid:

{a,b,c,….,z}

{!,@,#,$,%,^,&,\*,…}

{<,>,=,….}

|  |  |  |  |
| --- | --- | --- | --- |
| Scenario | user\_id | ECP | Expected Output |
| Successfully assigned role | 1334 | Valid | assigned role! |
| Unsuccessfully assigned role | gfdf | Invalid | Error! |

Class: role\_id

Valid:

{0,1,2,3,…}

Invalid:

{a,b,c,….,z}

{!,@,#,$,%,^,&,\*,…}

{<,>,=,….}

|  |  |  |  |
| --- | --- | --- | --- |
| Scenario | role\_id | ECP | Expected Output |
| Successfully assigned role | 534 | Valid | assigned role! |
| Unsuccessfully assigned role | efg | Invalid | Error! |

1. **Add employee:**

|  |  |
| --- | --- |
| ID | AE-13 |
| Use case | 3.13 |
| Form | Add Employee |
| Stakeholder | Admin |
| Field | employee\_name, employee\_email, employee\_phone, employee\_designation |
| Technique | Equivalence Class Partitioning (ECP) |
| Valid | {a,b,c,….,z}  {A,B,C,….,Z} |
| Invalid | {!,@,#,$,%,^,&,\*,…} {<,>,=,….} |

Class: employee\_name

Valid:

{a,b,c,….,z}

{A,B,C,….,Z}

Invalid:

{!,@,#,$,%,^,&,\*,…}

{<,>,=,….}

|  |  |  |  |
| --- | --- | --- | --- |
| Scenario | reason | ECP | Expected output |
| Unsuccessfully added employee | Joh!n | Invalid | Error |
| Successfully added employee | John | Valid | Employee Added! |
| Successfully added employee | Jane | Valid | Employee Added! |

Class: employee\_email

Valid:

{a,b,c,….,z}

{A,B,C,….,Z}

{0,1,2,……}

{@, .}

Invalid:

{!,#,$,%,^,&,\*,…}

{<,>,=,….}

|  |  |  |  |
| --- | --- | --- | --- |
| Scenario | employee\_email | ECP | Expected output |
| Unsuccessfully added employee | john#email.com | Invalid | Error |
| Successfully added employee | john@email.com | Valid | Employee Added! |
| Successfully added employee | jane@email.com | Valid | Employee Added! |

Class: employee\_phone

Valid:

{0,1,2,……}

Invalid:

{a,b,c,….,z}

{!,@,#,$,%,^,&,\*,…}

{<,>,=,….}

|  |  |  |  |
| --- | --- | --- | --- |
| Scenario | employee\_phone | ECP | Expected output |
| Unsuccessfully added employee | \*+9658kjhh | Invalid | Error |
| Successfully added employee | 1234567890 | Valid | Employee Added! |
| Successfully added employee | 12365478922 | Valid | Employee Added! |

Class: employee\_designation

Valid:

{a,b,c,….,z}

{A,B,C,….,Z}

Invalid:

{!,@,#,$,%,^,&,\*,…}

{<,>,=,….}

|  |  |  |  |
| --- | --- | --- | --- |
| Scenario | employee\_designation | ECP | Expected output |
| Unsuccessfully added employee | Mana#ger | Invalid | Error |
| Successfully added employee | Manager | Valid | Employee Added! |
| Successfully added employee | Developer | Valid | Employee Added! |

1. **Add Designation:**

|  |  |
| --- | --- |
| ID | AD-14 |
| Use case | 3.14 |
| Form | Add Designation |
| Stakeholder | Admin |
| Field | designation\_name |
| Technique | Equivalence Class Partitioning (ECP) |
| Valid | {a,b,c,….,z} |
| Invalid | {!,@,#,$,%,^,&,\*,…} {<,>,=,….}  {0,1,2,3,….} |

Class: designation\_name

Valid:

{a,b,c,….,z}

Invalid:

{0,1,2,3,…}

{!,@,#,$,%,^,&,\*,…}

{<,>,=,….}

|  |  |  |  |
| --- | --- | --- | --- |
| Scenario | designation\_name | ECP | Expected Output |
| Successfully added designation | Manager | Valid | Designation Added! |
| Unsuccessfully added designation | 123 | Invalid | Error! |

1. **Add Attendance:**

|  |  |
| --- | --- |
| ID | AA-15 |
| Use case | 3.15 |
| Form | Add Attendance |
| Stakeholder | Admin |
| Field | employee\_id, date, status (present/absent) |
| Technique | Equivalence Class Partitioning (ECP) |
| Valid | {0,1,2,3,…} (for employee\_id)  {YYYY-MM-DD} (for date)  {present, absent} (for status) |
| Invalid | {a,b,c,….,z} (for employee\_id)  {DD-MM-YYYY, MM-DD-YYYY} (for date)  {p,a,Present,Absent} (for status) |

Class: employee\_id

Valid:

{0,1,2,3,…}

Invalid:

{a,b,c,….,z}

|  |  |  |  |
| --- | --- | --- | --- |
| Scenario | Employee\_id | ECP | Expected Output |
| Successfully added attendance | 1234 | Valid | Attendance Added! |
| Unsuccessfully added attendance | ffgh | Invalid | Error! |

Class: date

Valid:

{YYYY-MM-DD}

Invalid:

{DD-MM-YYYY, MM-DD-YYYY}

|  |  |  |  |
| --- | --- | --- | --- |
| Scenario | date | ECP | Expected Output |
| Successfully added attendance | 2022-02-01 | Valid | Attendance Added! |
| Unsuccessfully added attendance | 02-01-2022 | Invalid | Error! |

Class: status

Valid:

{present, absent}

Invalid:

{p,a,Present,Absent}

|  |  |  |  |
| --- | --- | --- | --- |
| Scenario | status | ECP | Expected Output |
| Successfully added attendance | present | Valid | Attendance Added! |
| Unsuccessfully added attendance | p | Invalid | Error! |

1. **Issue Leave:**

|  |  |
| --- | --- |
| ID | IL-16 |
| Use case | 3.16 |
| Form | Issue Leave |
| Stakeholder | Admin |
| Field | rider\_id, start\_date, end\_date, reason |
| Technique | Equivalence Class Partitioning (ECP) |
| Valid | {a,b,c, …..,z}  {0,1,2,3,…..} |
| Invalid | {!,@,#,$,%,^,&,\*,…} {<,>,=,….} |

Class: Employee\_id

Valid:

{0,1,2,……}

Invalid:

{a,b,c,….,z}

{!,@,#,$,%,^,&,\*,…}

{<,>,=,….}

|  |  |  |  |
| --- | --- | --- | --- |
| Scenario | Employee\_id | ECP | Expected output |
| Successfully issue leave | 123 | Valid | Leave issued |
| Unsuccessful issue leave | 123abc | Invalid | Error |

Class: start\_date

Valid:

dd/mm/yyyy format

Invalid:

{a,b,c,….,z}

{!,@,#,$,%,^,&,\*,…}

{<,>,=,….}

|  |  |  |  |
| --- | --- | --- | --- |
| Scenario | Start\_date | ECP | Expected output |
| Successfully issue leave | 01/01/2023 | Valid | Leave issued |
| Unsuccessful issue leave | 01-01-2023 | Invalid | Error |

Class: end\_date

Valid:

dd/mm/yyyy format

Invalid:

{a,b,c,….,z}

{!,@,#,$,%,^,&,\*,…}

{<,>,=,….}

|  |  |  |  |
| --- | --- | --- | --- |
| Scenario | End\_date | ECP | Expected output |
| Successfully issue leave | 01/01/2023 | Valid | Leave issued |
| Unsuccessful issue leave | 01-01-2023 | Invalid | Error |

Class: reason

Valid:

{a,b,c,….,z}

{0,1,2,……}

Invalid:

{!,@,#,$,%,^,&,\*,…}

{<,>,=,….}

|  |  |  |  |
| --- | --- | --- | --- |
| Scenario | reason | ECP | Expected output |
| Successfully issue leave | Personal | Valid | Leave issued |
| Unsuccessful issue leave | Per$onal | Invalid | Error |

1. **Set Salary:**

|  |  |
| --- | --- |
| ID | SS-17 |
| Use case | 3.17 |
| Form | Set Salary |
| Stakeholder | Admin |
| Field | employee\_id, salary |
| Technique | Equivalence Class Partitioning (ECP) |
| Valid | {0,1,2,3,…} (for employee\_id)  {0, 1, 2, 3,…} (for salary) |
| Invalid | {a,b,c,….,z} (for employee\_id)  {a, b, c,….,z} (for salary) |

Class: employee\_id

Valid:

{0,1,2,3,…}

Invalid:

{a,b,c,….,z}

|  |  |  |  |
| --- | --- | --- | --- |
| Scenario | employee\_id | ECP | Expected output |
| Successfully set salary | 1234 | Valid | Salary Set! |
| Unsuccessful set salary | abcd | Invalid | Error |

Class: salary

Valid:

{0, 1, 2, 3,…}

Invalid:

{a, b, c,….,z}

|  |  |  |  |
| --- | --- | --- | --- |
| Scenario | salary | ECP | Expected output |
| Successfully set salary | 5000 | Valid | Salary Set! |
| Unsuccessful set salary | Two thousand | Invalid | Error |

1. **Generate Payroll:**

|  |  |
| --- | --- |
| ID | GP-18 |
| Use case | 3.18 |
| Form | Generate Payroll |
| Stakeholder | Admin |
| Field | employee\_id, date\_range |
| Technique | Equivalence Class Partitioning (ECP) |
| Valid | {0,1,2,3,…} (for employee\_id)  {yyyy-mm-dd} (for date\_range) |
| Invalid | {a,b,c,….,z} (for employee\_id)  {dd-mm-yyyy, mm-dd-yyyy} (for date\_range) |

Class: employee\_id

Valid:

{0,1,2,3,…}

Invalid:

{a,b,c,….,z}

|  |  |  |  |
| --- | --- | --- | --- |
| Scenario | employee\_id | ECP | Expected output |
| Successfully generated payroll | 1234 | Valid | Payroll Generated! |
| Unsuccessful generated payroll | abcd | Invalid | Error |

Class: date

Valid:

{YYYY-MM-DD}

Invalid:

{DD-MM-YYYY, MM-DD-YYYY}

|  |  |  |  |
| --- | --- | --- | --- |
| Scenario | date | ECP | Expected Output |
| Successfully generated payroll | 2022-02-01 | Valid | Payroll Generated! |
| Unsuccessfully generated payroll | 02-01-2022 | Invalid | Error! |

1. **Add Bonus and Deduction:**

|  |  |
| --- | --- |
| ID | ABD-19 |
| Use case | 3.19 |
| Form | Add Bonus and Deduction |
| Stakeholder | Admin |
| Field | employee\_id, bonus/deduction\_amount, bonus/deduction\_reason |
| Technique | Equivalence Class Partitioning (ECP) |
| Valid | {0,1,2,3,…} (for employee\_id)  {0,1,2,3,…} (for bonus/deduction\_amount)  {a,b,c,….,z} (for bonus/deduction\_reason) |
| Invalid | {a,b,c,….,z} (for employee\_id)  {!,@,#,$,%,^,&,\*,…} (for bonus/deduction\_amount)  {0,1,2,3,…} (for bonus/deduction\_reason) |

Class: employee\_id

Valid:

{0,1,2,3,…}

Invalid:

{a,b,c,….,z}

|  |  |  |  |
| --- | --- | --- | --- |
| Scenario | employee\_id | ECP | Expected output |
| Successfully added bonus/deduction | 1234 | Valid | Bonus/Deduction Added! |
| Unsuccessful added bonus/deduction | abcd | Invalid | Error |

Class: bonus/deduction\_amount

Valid:

{0,1,2,3,…}

Invalid:

{!,@,#,$,%,^,&,\*,…}

|  |  |  |  |
| --- | --- | --- | --- |
| Scenario | bonus/deduction\_amount | ECP | Expected output |
| Successfully added bonus/deduction | 2000 | Valid | Bonus/Deduction Added! |
| Unsuccessful added bonus/deduction | $300 | Invalid | Error |

Class: bonus/deduction\_reason

Valid:

{a,b,c,….,z}

Invalid:

{0,1,2,3,…}

|  |  |  |  |
| --- | --- | --- | --- |
| Scenario | bonus/deduction\_reason | ECP | Expected output |
| Successfully added bonus/deduction | Performance | Valid | Bonus/Deduction Added! |
| Unsuccessful added bonus/deduction | 300 | Invalid | Error |

# **Chapter 5: Implementation**

## **5.1 Endeavour (Team + Work + Way of Working)**

**Team Members:**

• Syed Mushahid Hussain

• Usama Shah

• Talha Khizar

**Work and responsibilities**

All team members actively participates in all parts of a project.

Way of working

We are following agile methodologies for better collaboration and faster delivery.

## **5.2 Flow Control/Pseudo codes**

**Product Module**

**Display a list of all products**

ROUTE: /products

ACTION: index()

// Sep 1: Get the list of products from the database

$products = Product::all();

// Step 2: Render the view with the list of products

RETURN view('products.index', compact('products'));

**Create New Product**

// Step 1: Display the form to create a new product

ROUTE: /products/create

ACTION: create()

// Step 5: Render the view to create a new product

RETURN view('products.create');

// Step 2: Store the new product in the database

ROUTE: /products

METHOD: POST

ACTION: store(Request $request)

// Step 3: Validate the input

$validatedData = $request->validate([

'name' => 'required|string',

'price' => 'required|numeric',

]);

// Step 4: Create a new product

$product = Product::create($validatedData);

// Step 5: Redirect to the product details page

RETURN redirect()->route('products.show', ['product' => $product->id]);

**Display Product**

// Step 1: Display the details of a product

ROUTE: /products/{product}

ACTION: show(Product $product)

// Step 2: Render the view with the product details

RETURN view('products.show', compact('product'));

**Checkout Module**

**Display Checkout Form**

// Step 1: Show the checkout form

ROUTE: /checkout

ACTION: index()

// Step 2: Get the cart items from the database

$cart = Cart::where('user\_id', auth()->user()->id)->get();

// Step 3: Render the checkout form with the cart items

RETURN view('checkout.index', compact('cart'));

On Form Submit (Place order)

// Step 1: Store the order in the database

ROUTE: /checkout

METHOD: POST

ACTION: store(Request $request)

// Step 2: Validate the input

$validatedData = $request->validate([

'name' => 'required|string',

'email' => 'required|email',

'address' => 'required|string',

]);

// Step 3: Get the cart items from the database

$cart = Cart::where('user\_id', auth()->user()->id)->get();

// Step 4: Calculate the total amount of the order

$total = 0;

foreach ($cart as $item) {

$total += $item->product->price \* $item->quantity;

}

// Step 5: Create the order

$order = Order::create([

'name' => $validatedData['name'],

'email' => $validatedData['email'],

'address' => $validatedData['address'],

'total' => $total,

'user\_id' => auth()->user()->id,

]);

// Step 6: Add the order items to the database

foreach ($cart as $item) {

OrderItem::create([

'order\_id' => $order->id,

'product\_id' => $item->product\_id,

'quantity' => $item->quantity,

]);

}

// Step 7: Clear the cart

Cart::where('user\_id', auth()->user()->id)->delete();

// Step 8: Redirect to the order confirmation page

RETURN redirect()->route('checkout.confirm', ['order' => $order->id]);

**Order Confirmation Page**

// Step 1: Show the order confirmation page

ROUTE: /checkout/{order}/confirm

ACTION: confirm(Order $order)

// Step 2: Render the order confirmation page

RETURN view('checkout.confirm', compact('order'));

**Inspection Module**

**Display Inspection Form**

// Step 1: Show the inspection form for a new product

ROUTE: /inspection/create/{product}

ACTION: create(Product $product)

// Step 2: Render the inspection form

RETURN view('inspection.create', compact('product'));

**Save Inspection**

// Step 1: Store the inspection result

ROUTE: /inspection

METHOD: POST

ACTION: store(Request $request)

// Step 2: Validate the input

$validatedData = $request->validate([

'product\_id' => 'required|integer',

'status' => 'required|string|in:approved,rejected',

'comment' => 'nullable|string',

]);

// Step 3: Create the inspection

$inspection = Inspection::create([

'product\_id' => $validatedData['product\_id'],

'status' => $validatedData['status'],

'comment' => $validatedData['comment'],

'inspector\_id' => auth()->user()->id,

]);

// Step 4: Update the product status

$product = Product::find($validatedData['product\_id']);

$product->status = $validatedData['status'];

$product->save();

// Step 5: Send email report to sellers and buyers

Mail::send('emails.inspection\_report', [

'product' => $product,

'inspection' => $inspection,

], function ($message) use ($product) {

$message->to($product->user->email)->subject('Inspection Report');

});

// Step 6: Redirect to the product list page

RETURN redirect()->route('product.index');

**Profile Module:**

**Display Profile**

// Step 1: Show the user profile page

ROUTE: /profile

ACTION: index()

// Step 2: Get the authenticated user

$user = auth()->user();

// Step 3: Render the profile view

RETURN view('profile.index', compact('user'));

**Update Profile**

// Step 1: Update the user profile

ROUTE: /profile

METHOD: POST

ACTION: update(Request $request)

// Step 2: Validate the input

$validatedData = $request->validate([

'name' => 'required|string',

'email' => 'required|email|unique:users,email,' . auth()->id(),

'password' => 'nullable|confirmed',

]);

// Step 3: Update the user

$user = auth()->user();

$user->name = $validatedData['name'];

$user->email = $validatedData['email'];

if ($request->filled('password')) {

$user->password = bcrypt($validatedData['password']);

}

$user->save();

// Step 4: Redirect to the profile page

RETURN redirect()->route('profile.index')->with('success', 'Profile updated');

## **5.3 Components, Libraries, Web Services and stubs**

**Web Service:**

We are using namecheap Shared hosting for Web services.

**Libraries:**

• React js

• Jquery

• Datatable

• Stripe

**Frameworks:**

• Laravel

• Bootstrap

## **5.4 IDE, Tools and Technologies**

**IDE**

VS Code

**Other Tools and Technologies**

• Git

• Github

• Xampp

• Php

• MySQL

## **5.5 Best Practices / Coding Standards**

### **5.5.1 Software Engineering Practices**

We are following agile methodology

### **5.5.2 Development Practices & Standards**

* We are following MVC Pattern
* We write clean, readable, and well-documented code.

## **5.6 Deployment Environment**

* Shared Hosting

## **5.8 Summary**

The Endeavour team consists of Syed Mushahid Hussain, Usama Shah, and Talha Khizar who work collaboratively and are responsible for all aspects of the project. The team follows agile methodologies and uses tools and technologies such as VS Code, Git, Github, Xampp, PHP, MySQL, React JS, Jquery, Datatable, Stripe, Laravel, and Bootstrap. The team follows best practices such as MVC pattern and writes clean, readable, and well-documented code. The deployment environment is shared hosting and SQA activities include defect detection using Black box test case design.

# **Chapter 6: Conclusion and Outlook**

## **6.1 Introduction**

The development of "Khas," an online marketplace for buying and selling used and new mobile phones, marks the successful culmination of our final year project. This chapter presents the conclusion drawn from our project journey, highlights the achievements made, identifies areas of improvement, provides a critical review of our work, and offers recommendations for future enhancements.

## **6.2 Achievements and Improvements**

Throughout the development of Khas, we have achieved significant milestones in building a secure and reliable platform for users to engage in online mobile phone transactions. Our key accomplishments include:

* Successfully creating a user-friendly interface for posting ads with full specifications of mobile phones.
* Implementing a secure payment system to protect buyers and sellers from potential scams.
* Establishing trust and credibility among users by offering a safe platform for buying and selling mobile phones.
* Ensuring the verification of product authenticity and condition before releasing payments.
* Providing efficient customer support and maintaining a well-organized platform.

However, we recognize that there is room for improvement to enhance the overall user experience and platform performance. Some areas that can be addressed include:

* Fine-tuning user credibility and trust mechanisms to further enhance security.
* Implementing advanced technologies to stay up-to-date with the ever-changing mobile phone industry trends.
* Expanding the market nationwide to attract a broader user base.
* Enhancing personalized search results to offer users more relevant product listings.

## **6.3 Critical Review**

Throughout the development process, we encountered and resolved several challenges, including:

* Establishing initial user trust and credibility was crucial, but we successfully implemented verification measures to mitigate risks.
* Maintaining the security of user information and financial data required robust encryption and protection mechanisms, which were successfully implemented.
* Handling logistics related to shipping and handling mobile phones demanded efficient coordination, and we managed to streamline the process effectively.
* Addressing disputes and issues during sales and purchases was critical, and we developed protocols to handle them efficiently.

## **6.4 Future Recommendations/Outlook**

Looking ahead, there are several avenues for future improvement and expansion of Khas:

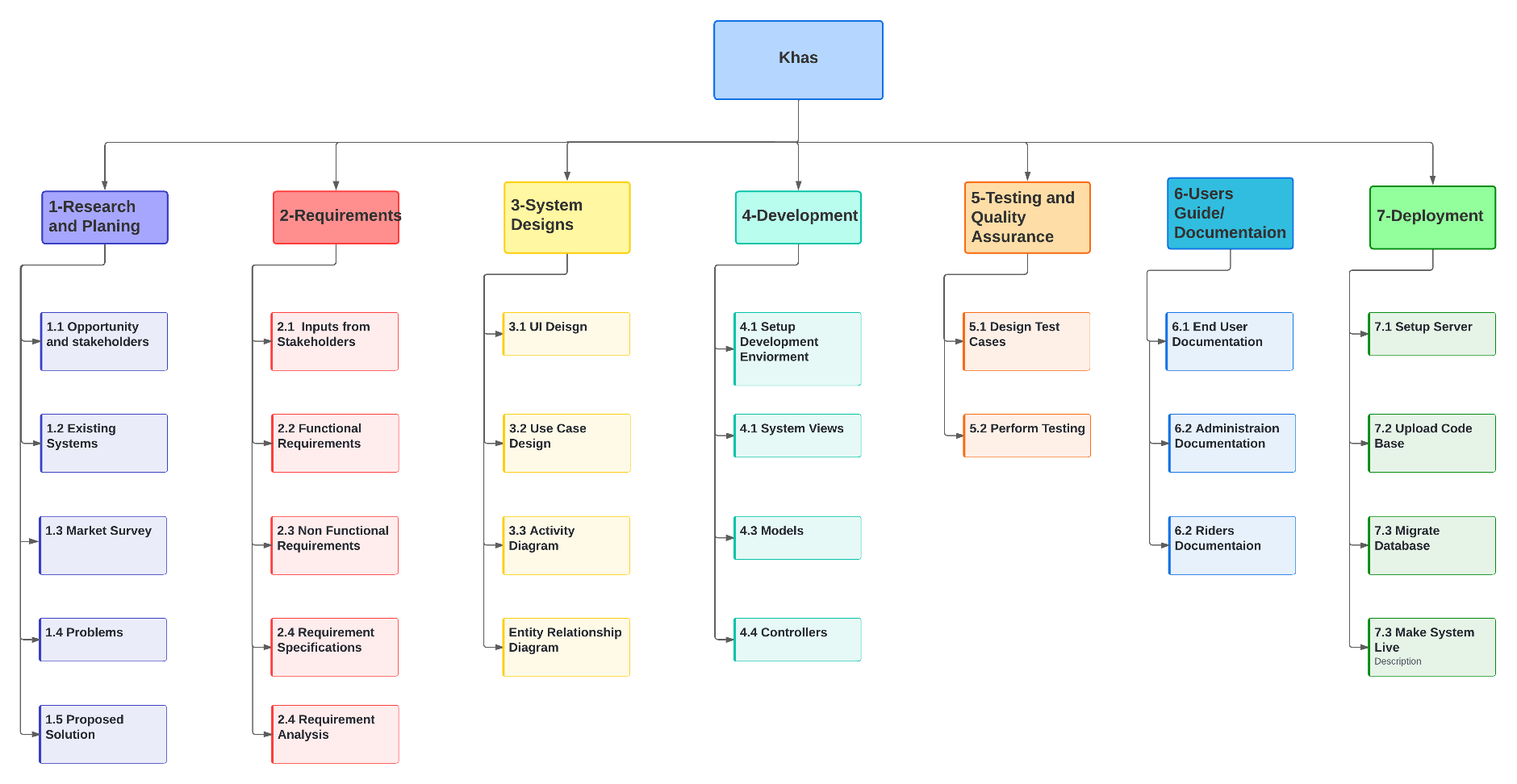
* Exploring opportunities to diversify services beyond mobile phones to attract a wider range of users and increase revenue streams.
* Continuous monitoring of user feedback to identify areas of enhancement and implement iterative updates to the platform.
* Collaborating with industry experts to stay at the forefront of technological advancements in the mobile phone market.
* Expanding Khas's reach internationally to tap into a global user base and establish it as a leading online marketplace.

## **6.5 Summary**

In conclusion, the successful development of Khas as an online marketplace for buying and selling mobile phones signifies our commitment to providing a secure and trustworthy platform for users. We have achieved our goals of building trust in online shopping, eliminating the risk of scams, and creating a central hub for mobile phone transactions. While we are proud of our accomplishments, we recognize the potential for further growth and improvements in the future. By addressing the challenges we faced and embracing future recommendations, Khas has the potential to become a prominent player in the C2C online marketplace, offering users an easy, convenient, and safe platform for their mobile phone transactions.

**Appendix A:**

**Work Breakdown Structure**



**Roles and Responsibility Matrix**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **WBS #** | **WBS Deliverable** | **Activity #** | **Activity to Complete the Deliverable** | **Duration**  **(# of Days)** | **Responsible Team Member(s) & Role(s)** |
| 1 | Research | 1.1 | Find opportunities and stake holders | 3 | All Members |
|  |  | 1.2 | Find existing systems and study them | 1 | All Members |
|  |  | 1.3 | Do Survey of niche market | 3 | All Members |
|  |  | 1.4 | Find all problems and challenges | 2 | All Members |
|  |  | 1.5 | Propose a solution | 3 | All Members |
|  |  |  |  |  |  |
| 2 | Requirements | 2.1 | Get Requirement Related inputs from stakeholders | 3 | Usama Shah |
|  |  | 2.2 | Enlist All Functional Requirements | 4 | Syed Mushahid |
|  |  | 2.3 | Enlist All Non – Functional Requirements | 2 | Talha Khizar |
|  |  | 2.3 | Detail Requirement Specifications | 6 | Usama Shah ,Syed Mushahid |
|  |  | 2.4 | Requirement Analysis | 4 | Talha Khizar |
|  |  |  |  |  |  |
| 3 | System Designs | 3.1.1 | Admin Panel UI Design | 4 | Talha Khizar |
|  |  | 3.1.2 | User Portal UI Design | 6 | Syed Mushahid |
|  |  | 3.1.3 | Riders Portal UI Design | 3 | Usama Shah |
|  |  | 3.2 | Use Case Diagram | 3 | Usama Shah |
|  |  | 3.3 | Activity Diagram | 3 | Talha Khizar |
|  |  | 3.4 | Entity Relationship Diagram | 3 | Syed Mushaid |
|  |  |  |  |  |  |
| 4 | Development | 4.1 | Setup Development Environment | 1 | Syed Mushahid |
|  |  | 4.2.1 | Admin Panel Views | 5 | Talha Khizar |
|  |  | 4.2.2 | User Portal Views | 6 | Syed Mushahid |
|  |  | 4.2.3 | Riders Portal Views | 4 | Usama Shah |
|  |  | 4.3.1 | Models | 14 | All Members |
|  |  | 4.3.2 | Controllers | 40 | All Members |
|  |  |  |  |  |  |
| 5 | Quality Assurance | 5.1 | Design Test Cases | 12 | Syed Mushahid |
|  |  | 5.2 | Perform Test Cases | 20 | Talha Khizar, Usama Shah |
|  |  |  |  |  |  |
| 6 | Users Guide Documentation | 6.1 | Admin Guide Documentation | 7 | Syed Mushahid,Talha Khizar |
|  |  | 6.2 | Users Guide Documentation | 7 | Usama Shah,Talha Khizar |
|  |  | 6.3 | Riders Guide Documentation | 4 | Usama Shah |
|  |  |  |  |  |  |
| 7 | Deployment | 7.1 | Setup Server | 2 | Syed Mushahid |
|  |  | 7.2 | Upload code Base | 1 | Usama Shah |
|  |  | 7.3 | Migrate Database | 1 | Talha Khizar |