**SAVIFY**

|  |
| --- |
|  |

**By:**

**Muzammil Arif**

**35747**

**Farhan Ahmed**

**32621**

**Supervised by:**

**Mr. Syed Hassaan Ali Shah**

**Faculty of Computing**

**Riphah International University, Islamabad**

**Spring 2025**

**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 Computer Science**

**Faculty of Computing**

**Riphah International University, Islamabad**

Date: December, 2024

**Final Approval**

This is to certify that we have read the report submitted by ***Muzammil Arif (35747)***, ***Farhan Ahmed (32621)*** for the partial fulfillment of the requirements for the degree of the Bachelors of Science in Computer Science (BSCS). 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 Computer Science (BSCS).

**Committee:**

|  |  |
| --- | --- |
| **1** | Syed Hassaan Ali Shah  (Supervisor) |
|  |  |
| **2** | Dr. Musharraf Ahmed  (Head of Department/chairman) |

**Declaration**

We hereby declare that this document “**Savify**” 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 **Syed** **Hassaan Ali Shah**. 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.

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

**Muzammil Arif**

**35747**

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

**Farhan Ahmed**

**32621**

**Dedication**

Our project is dedicated to our parents, teachers, friends, and our supervisor " Syed Hassaan Ali Shah" who has been our mentor and inspiration throughout out educational journey. We are pleased to dedicate our project to such motivational and inspiring people.**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 greatly indebted to our project supervisor “Syed Hassaan Ali Shah”. Without their personal supervision, advice and valuable guidance, completion of this project would have been doubtful. We are deeply indebted to them for their encouragement and continual help during this work.

And we are also thankful to our parents and family who have been a constant source of encouragement for us and brought us the values of honesty & hard work.

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

**Muzammil Arif**

**35747**

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

**Farhan Ahmed**

**32621**

**Abstract**

In today’s digital age, online shopping has become an integral part of consumer lifestyle, yet many existing platforms fall short of delivering an engaging and intuitive experience. Shoppers often face challenges like limitedinteraction, and the inabilitytobargain, which diminishes the appeal of online shopping compared to traditional markets. Moreover, searching for products can be time**-**consuming, especially for users who want to browse visually rather than text search. In response to these limitations, there is a need for an e-commerce platform that combines the convenience of online shopping with features that closely mimic the traditional shopping experience. This approach would provide consumers with greater flexibility,personalizedinteraction, and a more efficient way to find products tailored to their preferences.

Table of Contents

[Table of Contents i](#_Toc197613443)

[List of Tables iv](#_Toc197613444)

[List of Figures v](#_Toc197613445)

[1.2 Goals and Objectives 7](#_Toc197613446)

[1.3 Scope of the Project 8](#_Toc197613447)

[Chapter 2: Literature Review 10](#_Toc197613448)

[2.1 Introduction 10](#_Toc197613449)

[2.2 Background and Problem Elaboration 10](#_Toc197613450)

[2.3 Detailed Literature Review 11](#_Toc197613451)

[2.3.1 Definitions 11](#_Toc197613452)

[2.3.2 Related Research Work 1 11](#_Toc197613453)

[2.3.3 Related Research Work 2 12](#_Toc197613454)

[2.4 Literature Review Summary Table 12](#_Toc197613455)

[2.5 Research Gap 13](#_Toc197613456)

[2.6 Problem Statement 13](#_Toc197613457)

[3.2 Requirements 16](#_Toc197613458)

[3.2.1 Functional Requirements 16](#_Toc197613459)

[3.2.2 Non-Functional Requirements 17](#_Toc197613460)

[3.2.3 Hardware and Software Requirements 17](#_Toc197613461)

[3.3 Proposed Methodology 18](#_Toc197613462)

[3.4 System Architecture 19](#_Toc197613463)

[3.5 Use Cases: 19](#_Toc197613464)

Fully Dressed Use Cases: 23

[3.5.1: Login 24](#_Toc197613465)

[3.5.2 Sign Up 24](#_Toc197613466)

[3.5.3 Edit Profile Information 25](#_Toc197613467)

[3.5.4 View Sellers 25](#_Toc197613468)

[3.5.5 Delete Sellers 26](#_Toc197613469)

[3.5.6 Delete Buyers 27](#_Toc197613470)

[3.5.7 View Buyer 28](#_Toc197613471)

[3.5.8 Recover Password 28](#_Toc197613472)

[3.5.9 Add Products to cart 29](#_Toc197613473)

[3.5.10 Remove product from cart 30](#_Toc197613474)

[3.5.11 Add Review to Product 30](#_Toc197613475)

[3.5.12 View Products Page 31](#_Toc197613476)

[3.5.13 View Cart 32](#_Toc197613477)

[3.5.14 Search for Product 32](#_Toc197613478)

[3.5.15 Image Search 33](#_Toc197613479)

[3.5.16 AI Bargain by Buyer 33](#_Toc197613480)

[3.5.17 Complete Checkout 34](#_Toc197613481)

[3.5.18 Add New Product 35](#_Toc197613482)

[3.5.19 View Product 35](#_Toc197613483)

[3.5.20 Delete Product 36](#_Toc197613484)

[3.5.21 Reply to Reviews 37](#_Toc197613485)

[3.5.22 View and Manage Buyer Orders 37](#_Toc197613486)

[3.5.23 Edit Products 38](#_Toc197613487)

[3.5.25 Database Schema Design 39](#_Toc197613488)

[4.1 Introduction 41](#_Toc197613489)

[4.2 Implementation 41](#_Toc197613491)

[4.2.1 Implementation of the First Component/Algorithm 41](#_Toc197613494)

[4.2.2 Implementation of the Second Component/Algorithm 41](#_Toc197613495)

[4.2.3 Implementation of the Third Component/Algorithm 42](#_Toc197613497)

[4.3.1 Test Case 1: 42](#_Toc197613500)

[4.3.2 Test Case 2: 43](#_Toc197613501)

[4.3.3 Test Case 3: 44](#_Toc197613502)

[4.3.4 Test Case 4: 44](#_Toc197613503)

[4.3.5 Test Case 5: 45](#_Toc197613504)

[4.3.6 Test Case 6: 45](#_Toc197613505)

[4.3.7 Test Case 7: 46](#_Toc197613506)

[4.3.8 Test Case 8: 47](#_Toc197613507)

[4.3.9 Test Case 9: 47](#_Toc197613508)

[4.3.10 Test Case 10: 48](#_Toc197613509)

[4.3.11 Test Case 11: 49](#_Toc197613510)

[4.3.12 Test Case 12: 50](#_Toc197613511)

[4.5 Test Metrics 62](#_Toc197613512)

[Sample Test case Matric.No.1 63](#_Toc197613513)

[4.6 Conclusion 63](#_Toc197613514)

[5.1 Introduction 65](#_Toc197613515)

[5.2 Model Performance Analysis 65](#_Toc197613516)

[5.2.1 Training History and RMSE Analysis 66](#_Toc197613517)

[5.2.2 Final RMSE Values 66](#_Toc197613518)

[5.3 Model Evaluation Metrics 67](#_Toc197613519)

[5.4 Visual Evaluation: Actual vs Predicted 67](#_Toc197613520)

[5.5 Integration into Savify 67](#_Toc197613521)

[5.6 Future Enhancements 67](#_Toc197613522)

[5.7 Conclusion 67](#_Toc197613523)

[References 71](#_Toc197613524)

List of Tables

[Table 2.1 Research Work 12](#_Toc55559466)

[Table 3.1: Functional Requirement Buyer 13](#_Toc55559467)

[Table 3.2: Functional Requirement Seller 14](#_Toc55559467)

[Table 3.3: Functional Requirement Admin 14](#_Toc55559467)

[Table 3.4: Fully Dressed UseCase Login 21](#_Toc55559467)

[Table 3.5: Fully Dressed UseCase SignUp 21](#_Toc55559467)

[Table 3.6: Fully Dressed UseCase Edit Profile Information 22](#_Toc55559467)

[Table 3.7: Fully Dressed UseCase View Sellers 22](#_Toc55559467)

[Table 3.8: Fully Dressed UseCase Delete Sellers 23](#_Toc55559467)

[Table 3.9: Fully Dressed UseCase Delete Buyer 24](#_Toc55559467)

[Table 3.10: Fully Dressed UseCase View Buyer 25](#_Toc55559467)

[Table 3.11: Fully Dressed UseCase Recover Password 25](#_Toc55559467)

[Table 3.12: Fully Dressed UseCase Add Product to cart 26](#_Toc55559467)

[Table 3.13: Fully Dressed UseCase Remove Product from Cart 27](#_Toc55559467)

[Table 3.14: Fully Dressed UseCase Review to Product 27](#_Toc55559467)

[Table 3.15: Fully Dressed UseCase view product page 28](#_Toc55559467)

[Table 3.16: Fully Dressed UseCase view Cart 29](#_Toc55559467)

[Table 3.17: Fully Dressed UseCase Search for Product 29](#_Toc55559467)

[Table 3.18: Fully Dressed UseCase Image Search 30](#_Toc55559467)

[Table 3.19: Fully Dressed UseCase AI Bargain 30](#_Toc55559467)

[Table 3.20: Fully Dressed UseCase Complete Checkout 31](#_Toc55559467)

[Table 3.21: Fully Dressed UseCase Add new product 32](#_Toc55559467)

[Table 3.22: Fully Dressed UseCase view product 32](#_Toc55559467)

[Table 3.23: Fully Dressed UseCase Delete Product 33](#_Toc55559467)

[Table 3.21: Fully Dressed UseCase Reply to Reviews 34](#_Toc55559467)

[Table 3.22: Fully Dressed UseCase view and manage Buyer orders 34](#_Toc55559467)

[Table 3.23: Fully Dressed UseCase Edit Products 35](#_Toc55559467)

[Table 4:1 Test Case 1 39](#_Toc55559467)

[Table 4:2 Test Case 2 40](#_Toc55559467)

[Table 4:3 Test Case 3 41](#_Toc55559467)

[Table 4:4 Test Case 4 41](#_Toc55559467)

[Table 4:5 Test Case 5 42](#_Toc55559467)

[Table 4:6 Test Case 6 42](#_Toc55559467)

[Table 4:7 Test Case 7 43](#_Toc55559467)

[Table 4:8 Test Case 8 44](#_Toc55559467)

[Table 4:9 Test Case 9 44](#_Toc55559467)

[Table 4:10 Test Case 10 45](#_Toc55559467)

[Table 4:11 Test Case 11 46](#_Toc55559467)

[Table 4:12 Test Case 12 47](#_Toc55559467)

[Table 4.13 Test Data Buyer Signup 48](#_Toc55559467)

[Table 4.14 Test Data Buyer Signup 48](#_Toc55559467)

[Table 4.15 Test Data Buyer Signup 49](#_Toc55559467)

[Table 4.16 Test Data Buyer Signup 49](#_Toc55559467)

[Table 4.17 Test Data Buyer Signup 50](#_Toc55559467)

[Table 4.18 Test Data Buyer Signup 50](#_Toc55559467)

[Table 4.19 Test Data Seller Signup 51](#_Toc55559467)

[Table 4.20 Test Data Seller Signup 51](#_Toc55559467)

[Table 4.21 Test Data Seller Signup 52](#_Toc55559467)

[Table 4.22 Test Data Seller Signup 53](#_Toc55559467)

[Table 4.23 Test Data Seller Signup 54](#_Toc55559467)

[Table 4.24 Test Data Seller Signup 54](#_Toc55559467)

[Table 4.25 Test Data Seller Signup 55](#_Toc55559467)

[Table 4.26 Test Data Add Product 55](#_Toc55559467)

[Table 4.27 Test Data Add Product 56](#_Toc55559467)

[Table 4.28 Test Data Add Product 56](#_Toc55559467)

[Table 4.29 Test Data Add Product 57](#_Toc55559467)

[Table 4.30 Test Data Add Product 57](#_Toc55559467)

[Table 4.31 Test Data Add Product 57](#_Toc55559467)

[Table 4.32 Test Data Add Product 58](#_Toc55559467)

[Table 4.33 Test Data Add Product 59](#_Toc55559467)

[Table 4.34 Test Data Add Product 59](#_Toc55559467)

[Table 4.35 Test Case Matric 64](#_Toc55559467)

List of Figures

Figure 1: System Architecture Diagram 13

Figure 2:YOLO V11 Model Training 37

**Chapter 1: Introduction**

**1.1 Introduction**

**Savify** is a web-based multi-vendor platform designed to create a more interactive and personalized online shopping experience. The platform allows sellers to upload product information, while buyers can explore, negotiate, and search for items using advanced search functionalities. As current e-commerce platforms often lack personalization and effective engagement features, customers frequently experience impersonal transactions, limited communication options, and challenges in finding relevant products.

A significant issue addressed by Savify is the limited interaction between buyers and sellers, as well as the lack of real-time communication tools, which often results in a detached shopping experience to address these gaps, Savify incorporates mechanisms to improve buyer-seller interaction, enhance search capabilities, and maintain a respectful online environment through AI-driven moderation.

Savify aims to provide a transparent, secure, and efficient shopping experience, enhancing customer satisfaction and promoting innovation in online retail. Ultimately, Savify supports a more dynamic e-commerce environment, contributing to the growth and engagement of digital marketplaces.

**.**

## 1.2 Goals and Objectives

The primary objective of *Savify* is to develop an interactive e-commerce platform where customers can browse, search, and negotiate confidently, creating an experience that feels as engaging as traditional shopping.

**1.2.1 Goals:**

**1.2.1.1** To provide a web-based platform enabling shoppers to negotiate on product and get in their desired price.

**1.2.1.2** To enhance the online shopping experience by enabling visual base search which makes search process easier.

**1.2.2 Objectives:**

**1.2.2.1** Create a platform that allows multiple vendors to list and sell products seamlessly, providing a user-friendly experience for both sellers and buyers.

**1.2.2.2** Develop an AI-driven feature that enables real-time price negotiations between customers and vendors, offering a personalized and interactive shopping experience.

**1.2.2.3** Incorporate image and voice search capabilities to enhance product discovery and improve user convenience.

*Savify* aims to set a new standard in e-commerce by addressing gaps in personalization, interaction, and accessibility that are prevalent in current platforms.

## 1.3 Scope of the Project

**1.3.1** Our Website will be developed on MERN.

**1.3.2** Create a web-based system that allows multiple vendors to register, list, and manage their products, including inventory, pricing, and order fulfillment.

**1.3.3** Implement secure user registration and login functionalities for both customers and vendors, along with profile management features.

**1.3.3** Integrate image search functionality enabling users to search for products using images.

**1.3.4** Develop an AI-driven bargaining feature that allows customers to negotiate prices in real-time, providing a dynamic and personalized shopping experience.

**1.3.5** Include features for customer reviews, ratings.

**1.3.6**  Problem will be solved with Machine Learning.

**1.3.7** The platform would be accessible and user-friendly, and simplified for both buyer and seller.

**Chapter 2: Literature Review**

# Chapter 2: Literature Review

## 2.1 Introduction

**Savify** is a web-based multi-vendor e-commerce platform developed to bring a traditional shopping experience into the digital space. While e-commerce has seen rapid growth globally, many platforms lack interactive and personalized features that could improve user experience and buyer-seller engagement. Savify is tailored specifically to provide customers with a more dynamic and connected shopping journey. The platform enables sellers to easily share product details and interact with buyers, addressing common issues such as limited negotiation options, impersonal interactions, and difficulty in finding products intuitively.

The primary challenges addressed by Savify include the lack of personalized interaction and buyer engagement, which often reduces customer satisfaction on existing platforms. By integrating features like real-time bargaining, Savify has taken steps to bridge this gap, making online shopping both interactive and efficient.

## 2.2 Background and Problem Elaboration

While multi-vendor e-commerce platforms have broadened online selling opportunities, they often lack features that enhance personalization and user interaction. Traditional platforms miss the personalized negotiations and interactive experiences of physical stores, leading to the less customer engagement.

Text-based search functionalities can be limiting due to language barriers or vague descriptions, making product discovery frustrating. By integrating an AI Bargaining System, image search, the platform can simulate in-store experiences and improve accessibility.

This project aims to create a web-based multi-vendor e-commerce platform that addresses these shortcomings by incorporating advanced AI features to enhance user satisfaction and streamline the shopping experience.

## 2.3 Detailed Literature Review

### 2.3.1 Definitions

These definitions are the foundation for understanding key concepts related to our project and provide clarity for readers who may not be familiar with specific terms in the context of Ecommerce and Savify.

**2.3.1.1 AI-Powered Bargaining**  
The use of artificial intelligence to automate price negotiations between buyers and sellers in e-commerce platforms, aiming to optimize outcomes for both the parties.

**2.3.1.2** **E-Commerce Platforms**  
Online systems that facilitate the buying and selling of goods and services, providing a digital marketplace for transactions.

**2.3.1.3 Chatbots in E-Commerce**  
AI-driven tools that simulate human conversation to assist customers in navigating e-commerce platforms and answering queries.

### 2.3.2 Related Research Work 1

**2.3.2.1 Title:** E-COMMERCE NEGOTIATION BASED ON ARTIFICIAL INTELLIGENCE

**2.3.2.2 Authors:** Dinesh Kumar Singh, R.K. Srivastava

**2.3.2.3 Publication Year:** 2021

**2.3.2.4 Summary:**  
This paper provides a comprehensive review of various AI-based negotiation mechanisms employed in e-commerce models. It discusses the effectiveness of these mechanisms in automating bargaining processes and enhancing the transaction efficiency.

**2.3.2.5 Analysis:**  
The paper focuses on the functionalities of AI negotiation tools like Nibble, which utilize artificial intelligence to interact with online retailers and potentially lower prices for consumers. These tools automate the price negotiation process by initiating communication with a retailer's chat interface or virtual assistant, offering a price below the listed amount and adjusting offer based on the retailer's response. They also leverage data-driven decision-making by analyzing historical pricing trends and competitor offerings to identify opportunities for better deals. For consumers who generally dislike manual negotiation or lack experience in bargaining, these AI tools offer a convenient and potentially cost-saving solution. By eliminating the need for direct interaction with retailers, these tools streamline the negotiation process and are particularly effective for standardized products with established pricing structures.

**2.3.2.6 Connections to Savify:**  
The insights from this paper inform the development of Savify's AI bargaining system, particularly in understanding different negotiation models and their applicability in e-commerce platforms.

**2.3.2.7 Significance to the Project:**  
This work serves as a foundational reference for integrating AI-driven negotiation mechanisms into Savify, aiding in the design of an efficient and user-friendly bargaining system.

### 2.3.3 Related Research Work 2

**2.3.3.1 Title:** AI Negotiation Emerges as a New Frontier in E-Commerce

**2.3.3.2 Author:** Luís Rijo

**2.3.3.3 Publication Year:** 2024

**2.3.3.4 Summary:**  
This article discusses the rise of AI negotiation tools in e-commerce, emphasizing their role in automating price negotiations and enhancing customer experience through personalized interactions.

**2.3.3.5 Analysis:**  
The research highlights several AI techniques utilized in e-commerce negotiations. Knowledge-Based Systems (KBS) use predefined rules and expert knowledge to simulate human decision-making in making price less. Case-Based Reasoning (CBR) involves solving new problems based on the solutions of the similar past problems, adapting previous experiences to current negotiations. Artificial Neural Networks (ANN) employ machine learning to model complex relationships and patterns in negotiation data, enabling adaptive and predictive negotiation strategies. Genetic Algorithms (GA) apply evolutionary principles to optimize negotiation strategies over successive generations, thereby enhancing the efficiency and effectiveness of negotiations. Multi-Agent Systems (MAS) represent buyers and sellers as the autonomous agents that negotiate on behalf of their users, facilitating dynamic and scalable negotiation processes. These models aim to duplicate human negotiation behaviors, enabling automated and efficient bargaining in e-commerce platforms.

**2.3.3.6 Connections to Savify:**  
The findings align with Savify's objectives of integrating AI negotiation to offer personalized pricing and enhance user engagement.

**2.3.3.7 Significance to the Project:**  
This research underscores the importance of AI negotiation in modern e-commerce, providing valuable insights for refining Savify's bargaining features.

## 2.4 Literature Review Summary Table

Table 2.1 Research Work

| **No.** | **Title** | **Authors** | **Year** | **Focus Area** | **Relevance to Savify** |
| --- | --- | --- | --- | --- | --- |
| 1 | E-COMMERCE NEGOTIATION BASED ON ARTIFICIAL INTELLIGENCE | Dinesh Kumar Singh, R.K. Srivastava | 2021 | AI negotiation mechanisms in e-commerce | Provides foundational understanding of AI negotiation models applicable to Savify. |
| 2 | AI Negotiation Emerges as a New Frontier in E-Commerce | Luís Rijo | 2024 | Implementation and challenges of AI negotiation in e-commerce | Offers insights into the practical application and challenges of AI negotiation. |

## 2.5 Research Gap

Despite advancements in e-commerce technologies, there is a noticible gap in integrating advanced AI features into multi-vendor platforms to enhance user interaction and personalization. Current multi-vendor e-commerce platforms often lack the incorporation of AI-driven bargaining systems that allow for dynamic price negotiations, a feature that could simulate the personalized experience of physical store shopping. While some platforms have implemented image search independently to improve product discovery, there is limited research and practical application combining these functionalities within a single platform.

The research gap lies in developing a comprehensive, web-based multi-vendor e-commerce platform that seamlessly incorporates an AI bargaining system alongside advanced image search capabilities. Such an integration remains underexplored in academic research and commercial applications. Addressing this gap can lead to the more engaging and accessible shopping experience, meeting modern consumer expectations and providing vendors with the innovative tools to enhance customer satisfaction and loyalty.

## 2.6 Problem Statement

Existing multi-vendor e-commerce platforms, many lack advanced features that provide personalized and interactive shopping experiences akin to physical stores. Traditional platforms often miss opportunities for real-time price negotiations, leading to reduced customer engagement and satisfaction. Additionally, reliance on text-based search functionalities presents challenges for users facing language barriers or when product descriptions are insufficient, making product discovery cumbersome. There is a pressing need for an innovative e-commerce solution that integrates an AI-driven bargaining system, image search, capabilities to enhance user interaction, accessibility, and overall satisfaction in the online shopping experience.

**Chapter 3: Requirements and Design**

**3.1 Introduction**

In this chapter, we have developed the functional requirements for our actors, i.e., Buyer**,** Seller**,** andAdmin. The requirements are specifically designed for the Savify platform.

Savify is a web-based e-commerce platform designed to provide an interactive and efficient way for customers and sellers to connect and engage with each other. The platform is user-friendly, easy to navigate and search, and offers features such as AI bargaining system, image search, AI assistant, and speech-to-text support. These functionalities ensure a convenient and seamless experience for all users.

We created system use cases based on each functional requirement and developed corresponding use case diagrams. Additionally, we prepared fully dressed use cases for the main actors, i.e., Buyer, Seller, and Admin, ensuring that each role's interactions and responsibilities are clearly outlined within the Savify system.

## 3.2 Requirements

### 3.2.1 Functional Requirements

**Buyer:**

Table 3.1 Functional Requirement Buyer

|  |  |
| --- | --- |
| **ID** | **Requirements** |
| FR-1.1 | Buyer shall be able to sign up on website. |
| FR-1.2 | Buyer shall be able to login to the website. |
| FR-1.3 | Buyer shall be able to edit their profile. |
| FR-1.4 | Buyer shall be able to recover passwords. |
| FR-1.5 | Buyer shall be able to add product to cart. |
| FR-1.6 | Buyer shall be able to delete product from cart. |
| FR-1.7 | Buyer shall be able to buy product. |
| FR-1.8 | Buyer shall be able to add review to product. |
| FR-1.9 | Buyer shall be able to bargain from seller. |
| FR-1.10 | Buyer shall be able to view products |
| FR-1.11 | Buyer shall be able to view orders. |
| FR-1.12 | Buyer shall be able to view cart. |
| FR-1.13 | Buyer shall be chat with savify. |

**Seller:**

Table 3.2 Functional Requirement Seller

|  |  |
| --- | --- |
| **ID** | **Requirements** |
| FR-2.1 | Seller shall be able to register their account. |
| FR-2.2 | Seller shall be able to login to their account. |
| FR-2.3 | Seller shall be able to edit their profile. |
| FR-2.4 | Seller shall be able to recover passwords. |
| FR-2.5 | Seller shall be able to add products. |
| FR-2.6 | Seller shall be able to view products. |
| FR-2.7 | Seller shall be able to delete products. |
| FR-2.8 | Seller shall be able to edit products. |
| FR-2.9 | Seller shall be able to view orders. |
| FR-2.10 | Seller shall be able to manage orders. |
| FR-2.11 | Seller shall be able to reply to customers reviews. |

**Admin:**

Table 3.3 Functional Requirement Admin

|  |  |
| --- | --- |
| **ID** | **Requirements** |
| FR-3.1 | Admin shall be able to login to account. |
| FR-3.2 | Admin shall be able to view sellers. |
| FR-3.3 | Admin shall be able to view products |
| FR-3.4 | Admin shall be able to delete sellers. |
| FR-3.5 | Admin shall be able to delete products. |
| FR-3.6 | Admin shall be able to view buyers. |
| FR-3.7 | Admin shall be able to delete buyers. |

### 3.2.2 Non-Functional Requirements

### 3.2.3 Hardware and Software Requirements

**3.2.3.1 Hardware Requirements:**

**3.2.3.1.1 Server:** Server should run windows 10-11 for the latest requirements.

**3.2.3.1.2 Storage:** Moderate Storage to save all the data during and after project completion.

**3.2.3.1.3 Processors:** High performance Processors such as GPUs to efficiently compute the projects.

**3.2.3.1.4 Camera:** Webcam for product Search through image detection.

**3.2.3.2 Software Requirements:**

**3.2.3.2.1 Operating System:** Operating system such as Windows, Linux or MacOS.

**3.2.3.2.2 Database:** We used MongoDB as our Database for storage purpose.

**3.2.3.2.3 Programming Languages:**

* The website can be built using the MERN stack, which includes:

➢ JavaScript: for server-side and client-side scripting.

➢ Node.js: A JavaScript runtime environment for server-side development.

➢ Express.js: a web application framework for building the server-side application.

➢ React.js: A JavaScript library for building the client-side user interface.

* Python: Trained YOLOv11 latest model on product images.

**3.2.3.3 Development Tools:** Development tools such as Google Colab, Visual Studio code to run and debug codes. Furthermore, we used Roboflow to annotate images of dataset.

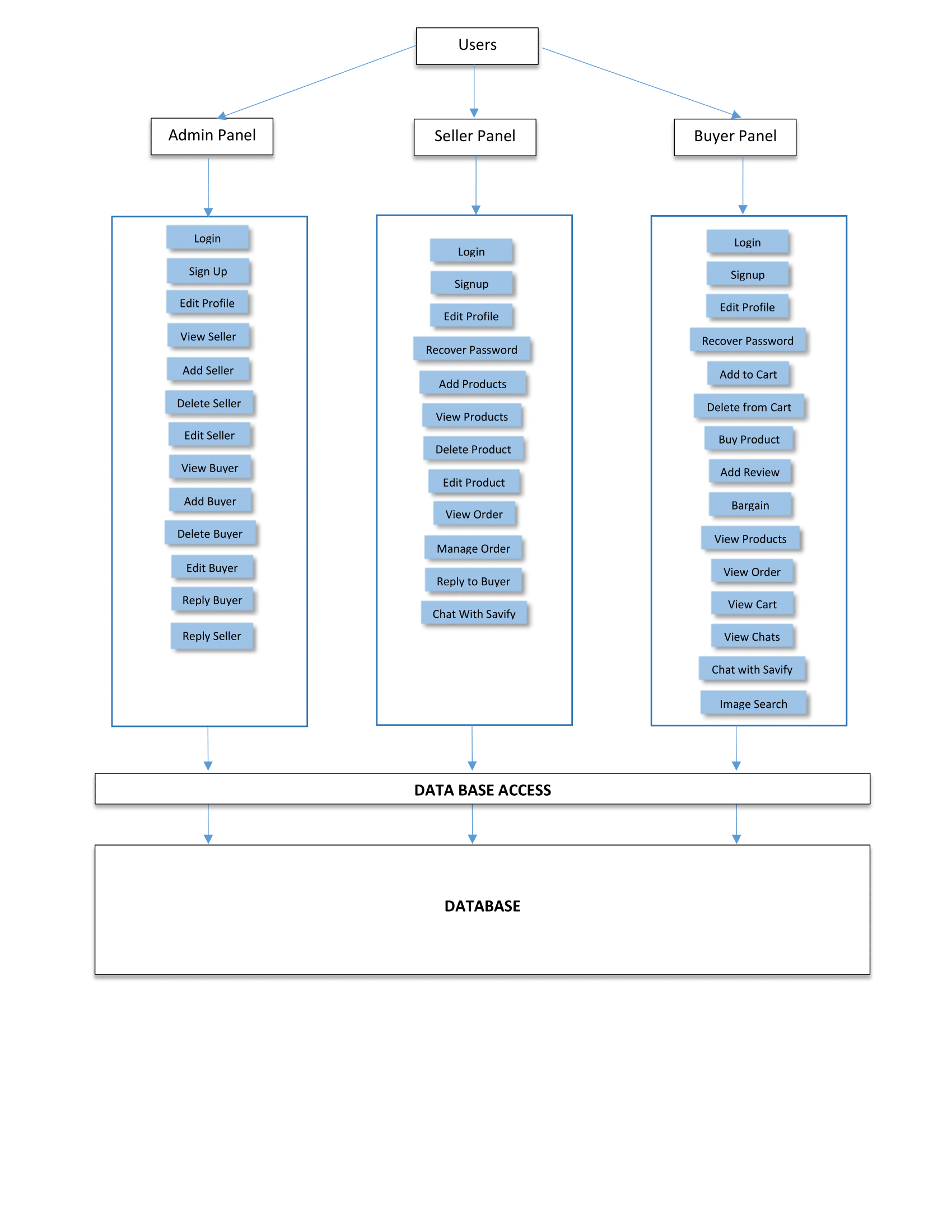
**3.2.3.4 Version Control:** A version control system like Git to manage source code and collaborate with multiple developers.

## 3.3 Proposed Methodology

Savify is a web-based platform designed to connect customers and sellers easily. The platform is tailored specifically for e-commerce, providing features that facilitate seamless interaction between both parties. Sellers can upload their products, and customers can browse, search, and purchase items conveniently.

As current e-commerce platforms in Pakistan often lack features like real-time bargaining and AI-assisted shopping, Savify addresses these gaps by creating a user-friendly and innovative experience. The platform allows sellers to showcase their products with images and descriptions, while customers can negotiate prices using the built-in bargaining system.

## 3.4 System Architecture



**Fig.1.** **System Architecture Diagram**

## 3.5 Use Cases:

* Admin:



Figure3.2:Use-Case Diagram of Admin

* Seller:



Figure 3.3:Use-Case of Seller

* Buyer:



Figure 3.4:Use-Case of Buyer

**Complete Use Case:**

Figure 3.5:Complete Use-case

**Fully-Dressed Use Cases:**

### 3.5.1: Login

Table 3.4:Fully Dressed UseCase of User Login

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | | Login to System | | |
| Actors | | Admin, Seller, Buyer | | |
| Summary | | The user provides their login credentials. If valid, they are granted access to the system. | | |
| Pre-Conditions | |  The user must be registered in the system database.   The user must not already be logged in. | | |
| Post-Conditions | |  The user’s session is initiated.   The user is redirected to their respective dashboard. | | |
| Special Requirements | |  Ensure encryption of passwords during verification.   Provide feedback for invalid credentials. | | |
| Basic Flow | | | | |
| Actor Action | | | **System Response** | |
| 1 | The user opens the login page. | | 2 | The login page is displayed asking for email and password. |
| 3 | The user enters valid email and password. | | 4 | The system verifies the credentials, starts the session, and redirects to the appropriate dashboard. |
| **Alternative Flow** | | | | |
| 3 | The user enters invalid email or password. | | 4-A | The system displays an error message: "Invalid email or password." |

### 3.5.2 Sign Up

Table 3.5:Fully Dressed UseCase of SignUp

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | | Sign up | | |
| Actors | | Seller, Buyer | | |
| Summary | | The user registers for an account by providing necessary details. | | |
| Pre-Conditions | |  The user must not already have an account.   All required fields in the sign-up form must be valid. | | |
| Post-Conditions | |  A new account is created.   The user is redirected to the login page for authentication. | | |
| Special Requirements | |  Validate email format.   Check for duplicate email addresses during registration. | | |
| Basic Flow | | | | |
| Actor Action | | | **System Response** | |
| 1 | The user navigates to the sign-up page. | | 1 | The user navigates to the sign-up page. |
| 3 | The user fills out the form and submits it. | | 3 | The user fills out the form and submits it. |
| **Alternative Flow** | | | | |
| 4.1 | The user provides invalid or duplicate information. | | 4.2 | The user provides invalid or duplicate information. |

### 3.5.3 Edit Profile Information

Table 3.6:Fully Dressed UseCase of Edit Profile Information

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | | Edit Profile Information | | |
| Actors | | Seller, Buyer | | |
| Summary | | A logged-in user updates their profile details. | | |
| Pre-Conditions | |  The user must be logged in.   The user must have access to the "Edit Profile" section. | | |
| Post-Conditions | |  The user’s updated details are stored in the database.   Changes are reflected in the user’s account. | | |
| Special Requirements | |  Validation for email format, phone numbers, etc.   Real-time feedback for successful updates. | | |
| Basic Flow | | | | |
| Actor Action | | | **System Response** | |
| 1 | The user navigates to "Edit Profile." | | 2 | The system displays the user’s current profile information. |
| 3 | The user modifies the desired fields and submits the form. | | 4 | The system validates inputs and updates the user’s profile in the database, displaying a success message. |
| **Alternative Flow** | | | | |
| 4.1 | The user enters invalid data (e.g., invalid email format). | | 4.2 | The system highlights errors in the form and prompts the user to fix them. |

### 3.5.4 View Sellers

Table 3.7:Fully Dressed UseCase of View Sellers

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | | View Seller | | |
| Actors | | Admin | | |
| Summary | | The admin views the details of a registered seller, including their profile information, products, sales, and other relevant data. | | |
| Pre-Conditions | |  The admin must be logged into the system with appropriate permissions.   There must be at least one seller registered in the system. | | |
| Post-Conditions | |  The admin successfully views the details of the selected seller.   No data is modified during this process. | | |
| Special Requirements | |  The system should load seller details quickly.   The interface should provide comprehensive details, including seller profile, products, and performance metrics.   Only authorized admins should have access to seller details. | | |
| Basic Flow | | | | |
| Actor Action | | | **System Response** | |
| 1 | The admin navigates to the "Manage Sellers" section in the admin dashboard. | | 2 | The system displays a list of all registered sellers. |
| 3 | The admin selects a specific seller to view. | | 4 | The system retrieves and displays the seller’s details, including profile information, a list of products, sales statistics, and feedback received from buyers. |
| **Alternative Flow** | | | | |
| 3 | If there are no sellers in the system, the system displays a message indicating that no sellers are available to view. | | 4-A | If the selected seller's account has been deleted or is inaccessible, the system displays an error message. |

### 3.5.5 Delete Sellers

Table 3.8:Fully Dressed UseCase of Delete Sellers

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | | Delete Seller | | |
| Actors | | Admin | | |
| Summary | | Admin Deletes Sellers. | | |
| Pre-Conditions | |  Admin must be logged in.   The seller must be registered in the system. | | |
| Post-Conditions | |  The seller’s account is deleted from the database.   The seller's products and associated data are either archived or removed based on platform policies. | | |
| Special Requirements | |  Admin must confirm deletion to prevent accidental removals.   The system should notify the seller of their account deletion via email or notification.   Compliant with data retention and privacy laws (e.g., GDPR). | | |
| Basic Flow | | | | |
| Actor Action | | | **System Response** | |
| 1 | The admin navigates to "Manage Sellers." | | 2 | The system displays a list of registered Sellers. |
| 3 | The admin selects a buyer to Delete. | | 4 | The system performs the action and confirms the changes. |
| **Alternative Flow** | | | | |
| 3 | The admin attempts to delete a non-existent Seller. | | 4-A | The system displays an error message: "Seller not found." |

### 3.5.6 Delete Buyers

Table 3.9:Fully Dressed UseCase of Delete Buyers

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | | Delete Buyers | | |
| Actors | | Admin | | |
| Summary | | Admin Deletes Buyers. | | |
| Pre-Conditions | |  Admin must be logged in.   The buyer must be registered in the system. | | |
| Post-Conditions | |  The buyer’s account is deleted from the database.   The buyer’s products and associated data are either archived or removed based on platform policies. | | |
| Special Requirements | |  Admin must confirm deletion to prevent accidental removals.   The system should notify the seller of their account deletion via email or notification.   Compliant with data retention and privacy laws (e.g., GDPR). | | |
| Basic Flow | | | | |
| Actor Action | | | **System Response** | |
| 1 | The admin navigates to "Manage Buyers." | | 2 | The system displays a list of registered Buyers. |
| 3 | The admin selects a buyer to Delete. | | 4 | The system performs the action and confirms the changes. |
| **Alternative Flow** | | | | |
| 3 | The admin attempts to delete a non-existent Buyer. | | 4-A | The system displays an error message: "Buyer not found." |

### 3.5.7 View Buyer

Table 3.10:Fully Dressed UseCase of View Buyer

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | | View Buyer | | |
| Actors | | Admin | | |
| Summary | | The admin views the details of a registered buyer, including their profile information, order history, and other relevant data. | | |
| Pre-Conditions | |  The admin must be logged into the system with appropriate permissions.   There must be at least one buyer registered in the system. | | |
| Post-Conditions | |  The admin successfully views the details of the selected Buyer.   No data is modified during this process. | | |
| Special Requirements | |  The system should load Buyer details quickly.   The interface should provide comprehensive details, including buyer profile, products, and performance metrics.   Only authorized admins should have access to buyer details. | | |
| Basic Flow | | | | |
| Actor Action | | | **System Response** | |
| 1 | The admin navigates to the "Manage Buyers" section in the admin dashboard. | | 2 | The system displays a list of all registered Buyers. |
| 3 | The admin selects a specific buyer to view. | | 4 | The system retrieves and displays the buyer’s details, including profile information. |
| **Alternative Flow** | | | | |
| 3 | If there are no buyer in the system, the system displays a message indicating that no buyer are available to view. | | 4-A | If the selected buyer account has been deleted or is inaccessible, the system displays an error message. |

### 3.5.8 Recover Password

Table 3.11:Fully Dressed UseCase of Recover Password

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | | Recover Password | | |
| Actors | | Seller, Buyer | | |
| Summary | | A seller or buyer can recover their password by initiating a password reset process through the platform. | | |
| Pre-Conditions | |  The seller or buyer must have an active account on the platform.   The user must have access to the registered email or phone number associated with their account. | | |
| Post-Conditions | |  The system sends a password reset link or code to the registered email or phone number.   The user successfully resets their password and regains access to their account. | | |
| Special Requirements | |  The reset process must be secure to prevent unauthorized access.   Password reset links or codes must have an expiration time (e.g., 15 minutes). | | |
| Basic Flow | | | | |
| Actor Action | | | **System Response** | |
| 1 | The user navigates to the login page and clicks "Forgot Password." | | 2 | The system prompts the user to enter their registered email. |
| 3 | The user enters their registered email and submits the request. | | 4 | The system validates the input and sends a password reset link or code to the provided email. |
| 5 | The user clicks the link or enters the code and navigates to the password reset page. | | 6 | The system prompts the user to enter a new password. |
| 7 | The user enters and confirms the new password, then submits the form. | | 8 | The system validates the new password, updates the user's credentials in the database, and confirms the reset. |
| **Alternative Flow** | | | | |
| 3 | If the user enters an unregistered email. | | 4-A | The system displays an error message and prompts for a valid input. |
|  | If the user attempts to use an expired reset link or code | |  | The system notifies the user and prompts them to request a new one |

### 3.5.9 Add Products to cart

Table 3.12:Fully Dressed UseCase of Add Products to cart

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | | Add Products to Cart | | |
| Actors | | Buyer | | |
| Summary | | The buyer adds selected products to their shopping cart. | | |
| Pre-Conditions | | The buyer must be logged in. | | |
| Post-Conditions | |  The selected product is added to the cart.   The buyer’s cart is updated in real time. | | |
| Special Requirements | | Allow the buyer to specify quantity before adding to the cart. | | |
| Basic Flow | | | | |
| Actor Action | | | **System Response** | |
| 1 | The buyer clicks "Add to Cart" for a product. | | 2 | The system adds the product to the cart and updates the cart’s total. |
| **Alternative Flow** | | | | |
| 3 | The buyer tries to add an out-of-stock product | | 4-A | The system displays a message: "This product is currently out of stock." |

### 3.5.10 Remove product from cart

Table 3.13:Fully Dressed UseCase of Remove Product from cart

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | | Remove product from cart | | |
| Actors | | Buyer | | |
| Summary | | The buyer can remove a product from their shopping cart. | | |
| Pre-Conditions | |  The buyer must be logged in.   The buyer must have at least one product in the cart. | | |
| Post-Conditions | | The selected product is removed from the cart. | | |
| Special Requirements | |  | | |
| Basic Flow | | | | |
| Actor Action | | | **System Response** | |
| 1 | The buyer navigates to the cart page. | | 2 | The system displays all products in the buyer's cart. |
| 3 | The buyer clicks the "Remove" button next to a product. | | 4 | The system removes the product from the cart and updates the total price. |
| **Alternative Flow** | | | | |
| 3 |  | | 4-A |  |

### 3.5.11 Add Review to Product

Table 3.14:Fully Dressed UseCase of Add Review to Product

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | | Add Review to Product | | |
| Actors | | Buyer | | |
| Summary | | The buyer can add a review to a product. | | |
| Pre-Conditions | | The buyer must be logged in. | | |
| Post-Conditions | | The review is saved and associated with the product | | |
| Special Requirements | |  | | |
| Basic Flow | | | | |
| Actor Action | | | **System Response** | |
| 1 | The buyer navigates to the product page. | | 2 | The system displays the product details and a "Write a Review" section. |
| 3 | The buyer writes a review and submits it. | | 4 | The system validates the review and saves it to the database. |
|  |  | | 5 | The system displays a success message: "Your review has been submitted." |
| **Alternative Flow** | | | | |
| 3 |  | | 4-A |  |

### 3.5.12 View Products Page

Table 3.15:Fully Dressed UseCase of View Product Page

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | | View Products Page | | |
| Actors | | Buyer | | |
| Summary | | The buyer can browse and view details of products. | | |
| Pre-Conditions | | The buyer must be logged in. | | |
| Post-Conditions | | The buyer can view product details. | | |
| Special Requirements | |  | | |
| Basic Flow | | | | |
| Actor Action | | | **System Response** | |
| 1 | The buyer navigates to the "Shop" page. | | 2 | The system displays a list of products |
| 3 | The buyer clicks on a product. | | 4 | The system displays the product details, including images, price, and description. |
| **Alternative Flow** | | | | |
| 3 |  | | 4-A |  |

### 3.5.13 View Cart

Table 3.16:Fully Dressed UseCase of View Cart

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | | View Cart | | |
| Actors | | Buyer | | |
| Summary | | The buyer can view their shopping cart and the products it contains. | | |
| Pre-Conditions | |  The buyer must be logged in.   The buyer must have added at least one product to the cart. | | |
| Post-Conditions | | The cart content is displayed | | |
| Special Requirements | |  | | |
| Basic Flow | | | | |
| Actor Action | | | **System Response** | |
| 1 | The buyer clicks the "Cart" icon or link. | | 2 | The system displays the cart page with all added products, their quantities, and the total price. |
| **Alternative Flow** | | | | |
| 3 |  | | 4-A |  |

### 3.5.14 Search for Product

Table 3.17:Fully Dressed UseCase of Search for product

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | | Search for Product | | |
| Actors | | Buyer | | |
| Summary | | The buyer can search for a product using a keyword, such as the product name, category, or brand. | | |
| Pre-Conditions | | The buyer must be on the platform's main page or a search bar must be accessible. | | |
| Post-Conditions | | The system displays a list of products that match the search query. | | |
| Special Requirements | |  | | |
| Basic Flow | | | | |
| Actor Action | | | **System Response** | |
| 1 | The buyer enters a keyword into the search bar. | | 2 | The system processes the query and searches the database for relevant products. |
| 3 | The buyer clicks the "Search" button or presses Enter. | | 4 | The system displays a list of matching products with images, names, prices, and availability. |
| **Alternative Flow** | | | | |
| 3 | If no products match the search query | | 4-A | The system displays a message: "No products found. Try a different keyword. |

### 3.5.15 Image Search

Table 3.18:Fully Dressed UseCase of Image Search

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | | Image Search | | |
| Actors | | Buyer | | |
| Summary | | The buyer can upload or live detect an image to search for similar products using image recognition technology. | | |
| Pre-Conditions | | The buyer must have access to webcam access or access to files. | | |
| Post-Conditions | | The system displays a list of products that match the uploaded image. | | |
| Special Requirements | |  | | |
| Basic Flow | | | | |
| Actor Action | | | **System Response** | |
| 1 | The buyer clicks the "Image Search" button. | | 2 | The system opens a file uploader or camera option. |
| 3 | The buyer uploads an image or takes a photo. | | 4 | The system processes the image using image recognition technology. |
|  |  | | 5 | The system displays a list of similar products with images, names, and prices. |
| **Alternative Flow** | | | | |
| 3 | If no products match the uploaded image. | | 4-A | The system displays a message: "No matching products found. Please try another image. |

### 3.5.16 AI Bargain by Buyer

Table 3.19:Fully Dressed UseCase of AI Bargain

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | | AI Bargain By Buyer | | |
| Actors | | Buyer, AI System | | |
| Summary | | The buyer uses the AI system to negotiate the price of a product. | | |
| Pre-Conditions | |  The buyer must be logged in.   The product must be eligible for AI Bargain. | | |
| Post-Conditions | | The AI suggests a negotiated price, which the buyer can accept or reject. | | |
| Special Requirements | |  | | |
| Basic Flow | | | | |
| Actor Action | | | **System Response** | |
| 1 | The buyer clicks the "Bargain Now" button on the product page. | | 2 | The system opens an AI chat interface. |
| 3 | The buyer provides their offer to the AI. | | 4 | The AI evaluates the offer and responds with a counteroffer. |
| 5 | The buyer accepts or rejects the counteroffer. | | 6 | If accepted, the AI applies the negotiated price to the product. |
| **Alternative Flow** | | | | |
| 3 |  | | 4-A |  |

### 3.5.17 Complete Checkout

Table 3.20:Fully Dressed UseCase of Complete Checkout

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | | Complete Checkout | | |
| Actors | | Buyer | | |
| Summary | | The buyer finalizes their order by providing payment and shipping details. | | |
| Pre-Conditions | |  The buyer must be logged in.   The buyer must have items in their cart. | | |
| Post-Conditions | |  The order is placed successfully.   Payment is processed and confirmation is sent to the buyer. | | |
| Special Requirements | | Allow buyers to review and modify their order before confirming. | | |
| Basic Flow | | | | |
| Actor Action | | | **System Response** | |
| 1 | The buyer clicks "Checkout" in the cart. | | 2 | The system displays a summary of the cart items and the total price. |
| 3 | The buyer provides payment and shipping details. | | 4 | The system validates the input, processes the payment, and displays an order confirmation. |
| **Alternative Flow** | | | | |
| 3 | The buyer provides invalid shipping details | | 4-A | The system highlights the errors and prompts the buyer to correct them. |

### 3.5.18 Add New Product

Table 3.21:Fully Dressed UseCase of Add new Product

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | | Add New Products | | |
| Actors | | Seller | | |
| Summary | | The seller adds new products to the platform by providing details such as name, price, description, and images. | | |
| Pre-Conditions | |  The seller must be logged in.   Required product details must be available. | | |
| Post-Conditions | |  The product is successfully added to the system.   The product becomes available for buyers to view and purchase. | | |
| Special Requirements | |  Validate input fields (e.g., price, name, and description length).   Allow multiple images to be uploaded. | | |
| Basic Flow | | | | |
| Actor Action | | | **System Response** | |
| 1 | The seller navigates to the "Add Products" page. | | 2 | The system displays a form for entering product details (name, price, description, and images) |
| 3 | The seller fills out the form and uploads images. | | 4 | The system validates the input, saves the product, and displays a success message. |
| **Alternative Flow** | | | | |
| 3 | The seller provides incomplete or invalid details. | | 4-A | The system highlights the errors and prompts the seller to correct them. |

### 3.5.19 View Product

Table 3.22:Fully Dressed UseCase of View Product

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | | View Products | | |
| Actors | | Seller | | |
| Summary | | A seller views the detailed information of their own products, including product status, inventory, price, and product performance (e.g., views and orders). | | |
| Pre-Conditions | | The seller must be logged into their account.  - The product must exist in the seller's inventory. | | |
| Post-Conditions | | - The product details are displayed successfully.  - The seller can make further actions like editing or managing inventory. | | |
| Special Requirements | | The system must ensure that the seller can only view products they own. | | |
| Basic Flow | | | | |
| Actor Action | | | **System Response** | |
| 1 | The seller logs in and navigates to the "Manage Products" section in their dashboard. | | 2 | The system displays a list of all products the seller has listed, including summary information such as name, price, and stock status. |
| 3 | The seller selects a specific product to view more details. | | 4 | The system retrieves and displays detailed information about the selected product, such as product description, price, stock levels, product status (e.g., active, inactive), |
| **Alternative Flow** | | | | |
| 3 | If the seller has not listed any products | | 4-A | The system displays a message indicating that no products are available and suggests adding a new product. |

### 3.5.20 Delete Product

Table 3.23:Fully Dressed UseCase of Delete Product

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | | Delete Products | | |
| Actors | | Seller | | |
| Summary | | The seller can delete a product listed in their store. Upon confirmation, the product is removed from the store, and the database is updated. | | |
| Pre-Conditions | |  The seller must be authenticated and logged into their account.   The seller must have at least one product listed in their store. | | |
| Post-Conditions | | The product is successfully removed from the database and is no longer visible to buyers. | | |
| Special Requirements | |  The system should validate that the seller can only delete their own products.   There must be a confirmation step before the deletion to avoid accidental actions. | | |
| Basic Flow | | | | |
| Actor Action | | | **System Response** | |
| 1 | The seller logs in and navigates to the "Manage Products" section. | | 2 | The system displays a list of products added by the seller. |
| 3 | The seller clicks the "Delete" button for a specific product. | | 4 | The system displays a confirmation prompt: "Are you sure you want to delete this product? |
| 5 | The seller confirms the deletion. | | 6 | The system deletes the product from the database and updates the product list.  The system displays a success message: "Product deleted successfully." |
| **Alternative Flow** | | | | |
| 3 | The seller clicks "Cancel" on the confirmation prompt. | | 4-A | The seller clicks "Cancel" on the confirmation prompt. |

### 3.5.21 Reply to Reviews

Table 3.24:Fully Dressed UseCase of Reply to Reviews

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | | Reply to Reviews | | |
| Actors | | Seller | | |
| Summary | | The seller can respond to customer reviews regarding their products through the system. Sellers can view reviews and provide replies. | | |
| Pre-Conditions | |  The seller must be authenticated and logged into their account.   The seller must have received at least one review from customers. | | |
| Post-Conditions | |  The customer receives the reply through the system. | | |
| Special Requirements | |  | | |
| Basic Flow | | | | |
| Actor Action | | | **System Response** | |
| 1 | The seller logs into their account. | | 2 | The system displays the seller's dashboard with a “Reviews" section. |
| 3 | The seller navigates to the " Reviews" section. | | 4 | The system displays a list of customer reviews related to the seller's products. |
| 5 | The seller clicks on a specific review. | | 6 | The system displays the detailed review along with a text box for the reply. |
| **Alternative Flow** | | | | |
| 3 |  | | 4-A |  |

### 3.5.22 View and Manage Buyer Orders

Table 3.25:Fully Dressed UseCase of View and Manage Buyer Orders

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | | View and Manage Buyer Orders | | |
| Actors | | Seller | | |
| Summary | | The seller reviews and manages orders placed by buyers for their products. | | |
| Pre-Conditions | |  The seller must be logged in.   Orders for the seller's products must exist in the system. | | |
| Post-Conditions | |  The seller updates the status of orders (e.g., confirmed, shipped).   Buyers are notified of order status changes. | | |
| Special Requirements | |  Display order details clearly, including buyer information.   Allow filtering orders by name | | |
| Basic Flow | | | | |
| Actor Action | | | **System Response** | |
| 1 | The seller navigates to the "Manage Orders" page. | | 2 | The system displays a list of orders for the seller's products. |
| 3 | The seller selects an order and updates its status. | | 4 | The system updates the order status and notifies the buyer of the changes. |
| **Alternative Flow** | | | | |
| 3 | The seller attempts to update an invalid order. | | 4-A | The system displays an error message: "Unable to update order. Please try again." |

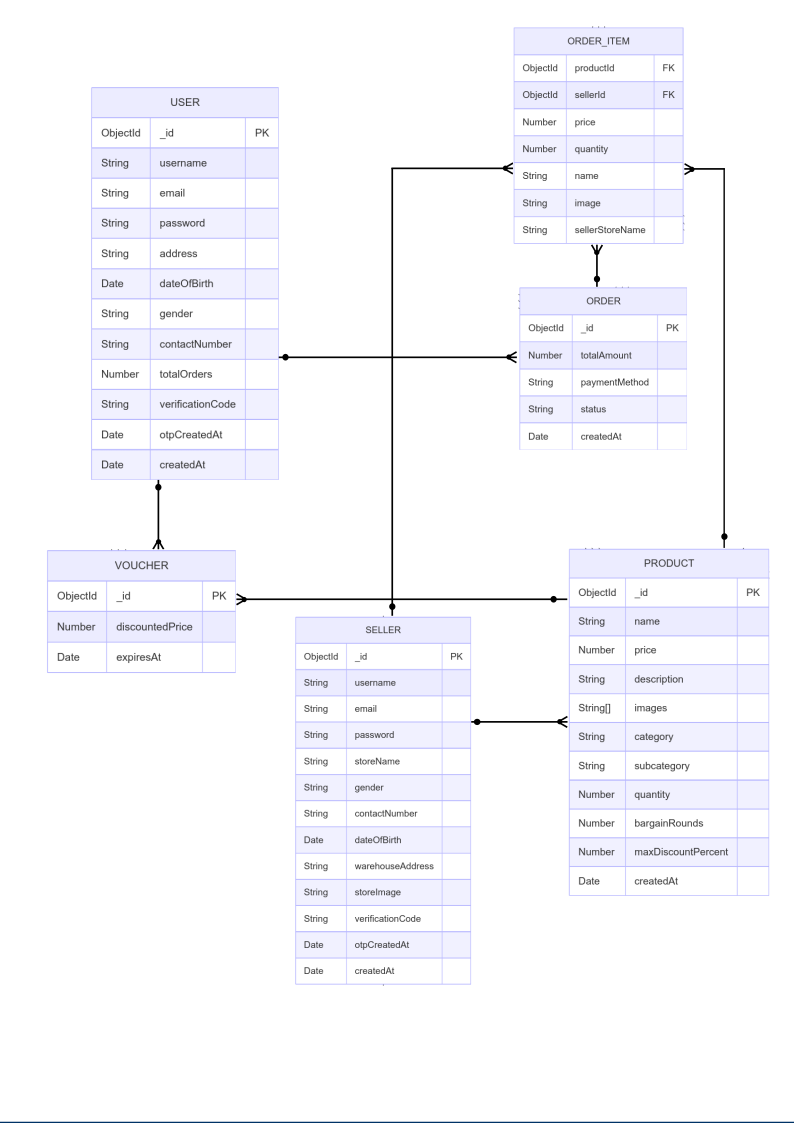
### 3.5.23 Edit Products

Table 3.26:Fully Dressed UseCase of Edit Products

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | | Edit Products | | |
| Actors | | Seller | | |
| Summary | | The seller modifies details of their existing products. | | |
| Pre-Conditions | |  The seller must be logged in.   The product to be edited must exist in the system and belong to the seller. | | |
| Post-Conditions | |  The updated product details are saved in the system.   Buyers see the updated product details immediately. | | |
| Special Requirements | |  Validate all updated fields (e.g., price must be numeric, description must meet length criteria).   Ensure no duplicate product names within the seller's product list. | | |
| Basic Flow | | | | |
| Actor Action | | | **System Response** | |
| 1 | The seller navigates to the "Edit Products" page. | | 2 | The system displays a list of products added by the seller. |
| 3 | The seller selects a product to edit. | | 4 | The system displays the product details in an editable form. |
| 5 | The seller updates the details and submits the form. | | 6 | The system validates the input, saves the changes, and displays a success message. |
| **Alternative Flow** | | | | |
| 5 | The seller enters invalid or duplicate product details. | | 6-A | 6-A. The system highlights errors and prompts the seller to fix them. |

## 3.5.25 Database Schema Design

## 



**Chapter 4: Implementation and Test Cases**

## 4.1 Introduction

## This section focuses on the practical steps taken to bring the Savify platform to life—a feature-rich, multi-vendor e-commerce solution that incorporates cutting-edge tools like AI-driven bargaining and image search. Here, we walk through how initial ideas were turned into a fully functional system, covering both the frontend and backend development. The implementation process included building essential features such as user authentication, product management, dynamic price negotiation, and seller dashboards—each tailored to the needs of buyers, sellers, and administrators. We also detail the tools and methods used for testing, including integration and system-level tests, to ensure everything runs smoothly, efficiently, and securely. By diving into the codebase, system architecture, and testing processes, this chapter offers a complete picture of how Savify delivers a reliable and user-friendly shopping experience.

## 4.2 Implementation

### At the heart of Savify’s implementation is its AI-powered bargaining system, which enables real-time price negotiations between buyers and sellers. This system uses an XGBoost model to analyze buyer behavior and market trends, helping to suggest optimal prices. Sellers can configure negotiation settings, like how many rounds to allow and the maximum discount they’re willing to offer. Meanwhile, the AI adjusts offers dynamically during the negotiation process based on the interaction.

### The platform was developed using the MERN stack, which supports secure user authentication, integration of negotiation features, and dynamic cart functionality. Rigorous testing, including integration tests, ensured the platform is stable and performs well under real-world conditions. This AI bargaining system adds a layer of personalization and efficiency that sets Savify apart from traditional online marketplaces.

### 4.2.1 Implementation of the First Component/Algorithm

**Development began with the thoughtful design of Savify’s user interface using Illustrator. The aim was to create a sleek, modern, and intuitive e-commerce experience for all user types—buyers, sellers, and admins. Figma prototypes guided the development of the frontend, making it easier to convert design ideas into functional components. These prototypes helped maintain a consistent look and feel across key areas like product listings, shopping carts, and admin dashboards, which in turn sped up development and made it easier to gather and act on user feedback.**

### ****4.2.2 Implementation of the Second Component/Algorithm****

### **Savify’s frontend was built using React.js, chosen for its modular structure and dynamic capabilities, which together create a smooth and responsive user experience. Key modules include product displays, cart management, checkout processes, and role-specific dashboards for buyers, sellers, and administrators. React Router was used for efficient navigation within the platform’s multi-role system, and localStorage helped maintain session states such as login status, cart data, and ongoing bargaining sessions. Special attention was given to responsive design to ensure that the platform works just as well on mobile devices as it does on desktops, increasing usability and engagement across all devices.**

### ****4.2.3 Implementation of the Third Component/Algorithm****

### **The backend of Savify was developed using Node.js and Express.js, creating a scalable and modular architecture for building RESTful APIs. The backend manages everything from user authentication and product listings to order processing, reviews, and admin functions. MongoDB was used as the database of choice due to its flexibility and smooth integration with Mongoose ORM.**

### **A standout feature in this phase was the integration of the AI bargaining system using a custom XGBoost model developed in Python. This model predicts the best possible discounts by analyzing user behavior, including factors like account age and order history, along with product ratings. A Python script (predict.py) communicates with the Express API using child processes, runs the model, and returns a calculated discount in real time. The model was trained on a mix of synthetic and realistic data, and evaluated using performance metrics such as RMSE, R², and MAE (as discussed in Chapter 5). The system supports multiple rounds of negotiation, with gradually decreasing discounts, allowing for smart, adaptable pricing conversations between buyers and sellers.**

**4.3 Test Case Design and Description**

**This section outlines the main features of the test cases created for the Savify platform, especially focusing on the AI-powered bargaining system. The tests are designed to cover a wide range of scenarios to ensure consistent performance. Input constraints—such as valid user profiles and accurate product details—are enforced to make sure the system behaves reliably.**

**The tests are run under shared conditions like a stable internet connection, properly configured systems, and access to backend services such as the database and authentication servers. There are also specific testing procedures in place to verify that negotiation rules, discount thresholds, and user-specific data are correctly handled across different scenarios.**

**Test cases also account for dependencies between features. For example, the system ensures that a negotiation must successfully conclude before an order can be placed. This structured testing strategy helps verify that all parts of the platform—from AI bargaining to secure transactions—work seamlessly together. As a result, Savify can deliver a secure, functional, and resilient e-commerce experience for all its users.**

### 

### 4.3.1 Test Case 1:

Table 4:1 Test Case 1

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **User Login** | | | | | |
| Test Case ID: | | *TC-01* | Test Date: | | *Date* |
| Test case Version: | | *1.1* | Use Case Reference(s): | | *3.5.1* |
| Revision History: | | N/A | | | |
| Objective | | Objective of this test case is to login the user through email, password, and OTP | | | |
| Product/Ver/Module: | | Savify 1.0 | | | |
| Environment: | | DEV environment with internet and browser support | | | |
| Assumptions: | | Assumes server is up, user has valid credentials and OTP | | | |
| Pre-Requisite: | | The user must be signed up and must have received a login OTP | | | |
| Step No. | Execution description | | | Procedure result | |
| 1 | Enter valid email, password and OTP | | | System verifies and logs in user, returns JWT token. | |
| Comments: During this test case the application successfully verified the user’s credentials and logged user in. | | | | | |
| *Test case Passed* | | | | | |

### 4.3.2 Test Case 2:

Table 4:2 Test Case 2

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **User Signup** | | | | | |
| Test Case ID: | | *TC-02* | Test Date: | | *Date* |
| Test case Version: | | *1.2* | Use Case Reference(s): | | *3.5.2* |
| Revision History: | | N/A | | | |
| Objective | | Objective of this test case is to register a new user with all mandatory fields and validations | | | |
| Product/Ver/Module: | | Savify 1.0 | | | |
| Environment: | | DEV environment with internet and browser support | | | |
| Assumptions: | | Server is up, and user provides valid information | | | |
| Pre-Requisite: | | Must not have an existing account with the same email address | | | |
| Step No. | Execution description | | | Procedure result | |
| 1 | Fill the form with all required fields: username, email, password, address, DOB, gender, contact number | | | System verifies fields, hashes password | |
| 2 | Click Signup button | | | System creates a new user in the database and returns success message | |
| Comments: During this test case the application successfully saved user data in database. | | | | | |
| *Test case Passed* | | | | | |

### 4.3.3 Test Case 3:

Table 4:3 Test Case 3

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Admin login** | | | | | |
| **<Reference>** | | | | | |
| Test Case ID: | | *TC-03* | Test Date: | | *Date* |
| Test case Version: | | *1.3* | Use Case Reference(s): | | *3.5.1* |
| Revision History: | | *Nil* | | | |
| Objective | | To validate login for Admin using credentials and OTP | | | |
| Product/Ver/Module: | | Savify 1.0 | | | |
| Environment: | | DEV environment with SMTP enabled | | | |
| Assumptions: | | Admin credentials are correct and the OTP is valid | | | |
| Pre-Requisite: | | Admin must request an OTP first before login | | | |
| Step No. | Execution description | | | Procedure result | |
|  | Admin must request an OTP first before login | | | OTP is generated and sent to the admin email | |
|  | Admin enters email, password, and OTP | | | System verifies password and OTP validity | |
|  | Clicks Login | | | JWT token is generated and admin is authenticated | |
| Comments: | | | | | |
| *Test case Passed* | | | | | |

### 4.3.4 Test Case 4:

Table 4:4 Test Case 4

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Seller Signup** | | | | | |
| **<Reference>** | | | | | |
| Test Case ID: | | *TC-04* | Test Date: | | *Date* |
| Test case Version: | | *1.4* | Use Case Reference(s): | | *3.5.2* |
| Revision History: | | *Nil* | | | |
| Objective | | To validate seller registration with all required information and a store image upload | | | |
| Product/Ver/Module: | | Savify 1.0 | | | |
| Environment: | | DEV environment with file upload capability and backend running | | | |
| Assumptions: | | Server is running, and all fields are filled with valid data | | | |
| Pre-Requisite: | | Seller must not be already registered with the same email address | | | |
| Step No. | Execution description | | | Procedure result | |
|  | Seller fills all fields: username, email, password, storeName, gender, contactNumber, DOB, warehouseAddress, and uploads store image | | | System validates inputs and uploads image to uploads/ folder | |
|  | Clicks "Signup" | | | Password is hashed and seller data is saved to database with success message | |
| Comments: | | | | | |
| *Test case Passed* | | | | | |

### 4.3.5 Test Case 5:

Table 4:5 Test Case 5

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Seller login** | | | | | |
| **<Reference>** | | | | | |
| Test Case ID: | | *TC-05* | Test Date: | | *Date* |
| Test case Version: | | *1.5* | Use Case Reference(s): | | *3.5.1* |
| Revision History: | | *Nil* | | | |
| Objective | | To verify login of a registered seller using credentials and OTP | | | |
| Product/Ver/Module: | | Savify 1.0 | | | |
| Environment: | | DEV environment with internet and SMTP support | | | |
| Assumptions: | | The email and password are correct, and the OTP is valid and unexpired | | | |
| Pre-Requisite: | | Seller must be registered and have requested a login OTP | | | |
| Step No. | Execution description | | | Procedure result | |
|  | Seller enters email, password, and received OTP | | | System validates credentials and OTP | |
|  | Clicks Login | | | JWT token is returned and seller is authenticated successfully | |
| Comments: | | | | | |
| *Test case Passed* | | | | | |

.

### 4.3.6 Test Case 6:

Table 4:6 Test Case 6

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Buyer Password Reset** | | | | | |
| **<Reference>** | | | | | |
| Test Case ID: | | *TC-06* | Test Date: | | *Date* |
| Test case Version: | | *1.6* | Use Case Reference(s): | | *3.5.8* |
| Revision History: | | *Nil* | | | |
| Objective | | To verify the functionality of password reset for a buyer using OTP verification. | | | |
| Product/Ver/Module: | | Savify 1.0 | | | |
| Environment: | | DEV environment with working email server and internet | | | |
| Assumptions: | | Internet is working and registered email exists in database | | | |
| Pre-Requisite: | | The buyer must request an OTP through their registered email | | | |
| Step No. | Execution description | | | Procedure result | |
| 1 | Enter registered email and request OTP | | | OTP is sent to the email | |
| 2 | Enter email and received OTP | | | System verifies the OTP | |
| 3 | Enter new password | | | Password is successfully updated | |
| Comments: | | | | | |
| *Test case Passed* | | | | | |

.

### 4.3.7 Test Case 7:

Table 4:7 Test Case 7

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Password Reset – Seller** | | | | | |
| Test Case ID: | | *TC-07* | Test Date: | | *Date* |
| Test case Version: | | *1.7* | Use Case Reference(s): | | *3.5.8* |
| Revision History: | | N/A | | | |
| Objective | | To ensure sellers can reset their password using email-based OTP | | | |
| Product/Ver/Module: | | Savify 1.0 | | | |
| Environment: | | DEV environment with functioning Nodemailer and Gmail integration | | | |
| Assumptions: | | Seller email is valid and OTP is used within 5-minute window | | | |
| Pre-Requisite: | | Seller must initiate password reset via email | | | |
| Step No. | Execution description | | | Procedure result | |
| 1 | Enter registered seller email to request OTP | | | OTP email is sent | |
| 2 | Enter correct OTP and email for verification | | | System accepts and confirms | |
| 3 | Submit new password | | | System updates seller password | |
| Comments: | | | | | |
| *Test case Passed* | | | | | |

### 4.3.8 Test Case 8:

Table 4:8 Test Case 8

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Buyer Update Profile** | | | | | |
| **<Reference>** | | | | | |
| Test Case ID: | | *TC-08* | Test Date: | | *Date* |
| Test case Version: | | *1.8* | Use Case Reference(s): | | *3.5.3* |
| Revision History: | | *Nil* | | | |
| Objective | | To verify that a buyer can successfully update their profile fields such as name, email, address, contact number, etc. | | | |
| Product/Ver/Module: | | Savify 1.0 | | | |
| Environment: | | DEV environment with MongoDB and Express API running | | | |
| Assumptions: | | User is logged in and provides valid input values | | | |
| Pre-Requisite: | | Existing buyer account with valid token and userId available in the request body | | | |
| Step No. | Execution description | | | Procedure result | |
| 1 | Submit updated profile fields using PUT request | | | Server validates and updates the data | |
| 2 | Submit invalid contact number (not +92 format) | | | Server returns validation error | |
| 3 | Submit missing required fields (e.g., username) | | | Server returns missing field error | |
| Comments: | | | | | |
| *Test case Passed* | | | | | |

.

### 4.3.9 Test Case 9:

Table 4:9 Test Case 9

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Seller Update Profile** | | | | | |
| **<Reference>** | | | | | |
| Test Case ID: | | *TC-09* | Test Date: | | *Date* |
| Test case Version: | | *1.9* | Use Case Reference(s): | | *3.5.3* |
| Revision History: | | *Nil* | | | |
| Objective | | To ensure a seller can update profile fields like store name, warehouse address, gender, contact number, etc. | | | |
| Product/Ver/Module: | | Savify 1.0 | | | |
| Environment: | | DEV environment with backend APIs active | | | |
| Assumptions: | | Seller is authenticated and has access to valid sellerId | | | |
| Pre-Requisite: | | Existing seller account; valid token or session | | | |
| Step No. | Execution description | | | Procedure result | |
| 1 | Submit updated seller fields (e.g., storeName, warehouseAddress) | | | Server updates the seller record | |
| 2 | Provide incorrect sellerId | | | Server returns 404 – Seller not found | |
| 3 | Submit empty required field (e.g., contact number) | | | Server returns validation error | |
| Comments: | | | | | |
| *Test case Passed* | | | | | |

.

### 4.3.10 Test Case 10:

Table 4:10 Test Case 10

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Place Order** | | | | | |
| **<Reference>** | | | | | |
| Test Case ID: | | TC-10 | Test Date: | | *Date* |
| Test case Version: | | *1.10* | Use Case Reference(s): | | *3.5.17* |
| Revision History: | | *Nil* | | | |
| Objective | | To verify that the buyer can place an order with products from multiple sellers, and the system splits them into separate orders correctly. | | | |
| Product/Ver/Module: | | Savify 1.0 | | | |
| Environment: | | DEV environment with Node.js, Express.js, MongoDB, Postman for testing | | | |
| Assumptions: | | Buyer must be registered and logged in, products should be available in inventory | | | |
| Pre-Requisite: | | Buyer account exists, valid product IDs in stock, proper buyerId, and request body format used | | | |
| Step No. | Execution description | | | Procedure result | |
| 1 | Send POST request to /api/orders/place-order with buyerId, items[] | | | Server processes and groups items by seller | |
| 2 | Items include multiple sellers' products | | | Server creates separate orders per seller | |
| 3 | Include cartVouchers object in request body (if any discount applied) | | | Discounted total is reflected in respective order | |
| 4 | Inventory quantity is checked and reduced after order placement | | | Product quantity is updated; stock validated correctly | |
| 5 | TotalOrders for the buyer is incremented | | | Buyer's totalOrders updated in the database | |
| Comments: | | | | | |
| *Test case Passed* | | | | | |

.

### 4.3.11 Test Case 11:

Table 4:11 Test Case 11

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Update Product Details** | | | | | |
| **<Reference>** | | | | | |
| Test Case ID: | | TC-11 | Test Date: | | *Date* |
| Test case Version: | | *1.11* | Use Case Reference(s): | | *3.5.23* |
| Revision History: | | *Nil* | | | |
| Objective | | To verify that a seller can update the product’s name, price, description, category, subcategory, and quantity successfully. | | | |
| Product/Ver/Module: | | Savify 1.0 | | | |
| Environment: | | DEV environment using Express.js API, MongoDB, and Postman/Frontend | | | |
| Assumptions: | | Product ID exists and is valid; user is authenticated and authorized as seller | | | |
| Pre-Requisite: | | A valid product already exists in the database with correct seller linkage | | | |
| Step No. | Execution description | | | Procedure result | |
| 1 | Send PUT request to /api/products/:id with updated fields in request body | | | Backend validates product existence by ID | |
| 2 | Include updated values for name, price, description, category, etc. | | | Product document is updated and saved in MongoDB | |
| 3 | Use Postman or UI to verify response status and returned updated product JSON | | | Updated product fields are reflected in response | |
| 4 | Validate changes by fetching the updated product via GET /api/products/:id | | | Updated product fields are reflected in response | |
| Comments: | | | | | |
| *Test case Passed* | | | | | |

.

### 4.3.12 Test Case 12:

Table 4.12 Test Case 12

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Add Product Review** | | | | | |
| **<Reference>** | | | | | |
| Test Case ID: | | TC-12 | Test Date: | | *Date* |
| Test case Version: | | *1.12* | Use Case Reference(s): | | *3.5.11* |
| Revision History: | | *Nil* | | | |
| Objective | | To verify that a logged-in buyer can add a review (rating + comment) to a product. | | | |
| Product/Ver/Module: | | Savify 1.0 | | | |
| Environment: | | DEV environment using Express.js API and MongoDB | | | |
| Assumptions: | | Product exists; user is authenticated; valid review data is provided | | | |
| Pre-Requisite: | | A product should exist, and the user must be logged in and provide review content | | | |
| Step No. | Execution description | | | Procedure result | |
| 1 | Send POST request to /api/products/:id/reviews with fields user, rating, comment | | | Backend finds product by ID and appends the review | |
| 2 | Ensure rating is a number and comment is a non-empty string | | | Product review array is updated and saved | |
| 3 | Validate that response contains a success message and the new review | | | Review object is returned in the response | |
| 4 | Fetch product again via /api/products/:id to verify review is present in its review list | | | Review is visible in product details | |
| Comments: | | | | | |
| *Test case Passed* | | | | | |

**4.4 Test Data:**

Table 4.13 Test Data Buyer Sign Up

|  |  |
| --- | --- |
| **Test Data ID:** | **TD-1** |
| **Form:** | Signup Form |
| **Stakeholder:** | User |
| **Field:** | Username |
| **Technique:** | Equivalence Class Partitioning (ECP) |
| **Test Data:** | Valid username, invalid username (starts with number, contains special characters, etc.) |
| **Valid Data:** | JohnDoe\_123, alice\_smith, bob25\_ |
| **Invalid Data:** | 123johnDoe, !invalidUser, username@123 |

Table 4.14 Test Data Buyer Sign Up

|  |  |
| --- | --- |
| **Test Data ID:** | **TD-2** |
| **Form:** | Signup Form |
| **Stakeholder:** | User |
| **Field:** | Email |
| **Technique:** | Equivalence Class Partitioning (ECP) |
| **Test Data:** | Valid email, invalid email (non-Gmail or non-"riphah.edu.pk" email) |
| **Valid Data:** | user@gmail.com, student@students.riphah.edu.pk |
| **Invalid Data:** | user@invalid.com, invalid@domain.edu |

Table 4.15 Test Data Buyer Sign Up

|  |  |
| --- | --- |
| **Test Data ID:** | **TD-3** |
| **Form:** | Signup Form |
| **Stakeholder:** | User |
| **Field:** | Password |
| **Technique:** | Equivalence Class Partitioning (ECP) |
| **Test Data:** | Valid password, invalid password (doesn't meet complexity requirements) |
| **Valid Data:** | Password123!, Secure@2025 |
| **Invalid Data:** | password123, 123456, simplepassword |

Table 4.16 Test Data Buyer Sign Up

|  |  |
| --- | --- |
| **Test Data ID:** | **TD-5** |
| **Form:** | Signup Form |
| **Stakeholder:** | User |
| **Field:** | Date of Birth |
| **Technique:** | Equivalence Class Partitioning (ECP) |
| **Test Data:** | Valid DOB, invalid DOB (future date, incomplete format) |
| **Valid Data:** | 1990-12-31, 2000-05-15 |
| **Invalid Data:** | 2025-01-01, 1990-02-30, abcd |

Table 4.17 Test Data Buyer Sign Up

|  |  |
| --- | --- |
| **Test Data ID:** | **TD-6** |
| **Form:** | Signup Form |
| **Stakeholder:** | User |
| **Field:** | Gender |
| **Technique:** | Equivalence Class Partitioning (ECP) |
| **Test Data:** | Valid gender, invalid gender (empty, unsupported option) |
| **Valid Data:** | Male, Female, Other |
| **Invalid Data:** | ``, Non-Binary, undefined |

Table 4.18 Test Data Buyer Sign Up

|  |  |
| --- | --- |
| **Test Data ID:** | **TD-7** |
| **Form:** | Signup Form |
| **Stakeholder:** | User |
| **Field:** | Contact Number |
| **Technique:** | Equivalence Class Partitioning (ECP) |
| **Test Data:** | Valid contact number, invalid contact number (wrong format) |
| **Valid Data:** | +923001234567, +923456789012 |
| **Invalid Data:** | 12345, +91234567890, 987654321 |

Table 4.19 Test Data Seller Sign Up

|  |  |
| --- | --- |
| **Test Data ID** | **TD-8** |
| **Test Data** | Username |
| **Stakeholder** | Seller |
| **Field** | Username |
| **Technique** | Equivalence Class Partitioning (ECP) |
| **Test Description** | Test the username format to ensure it follows the required pattern |
| **Test Data** | Username must not start with a number and must not contain special characters. |
| **Input Examples** | john\_doe, john123, john#doe, 123john, john@doe |
| **Expected Result** | Valid usernames must start with a letter and contain only letters, numbers, and underscores (e.g., john\_doe). Invalid usernames must be rejected. |
| **Comments** | Only alphanumeric usernames or with underscores are allowed. Special characters are not permitted. |

Table 4.20 Test Data Seller Sign Up

|  |  |
| --- | --- |
| **Test Data ID** | **TD-9** |
| **Test Data** | Email |
| **Stakeholder** | Seller |
| **Field** | Email |
| **Technique** | Equivalence Class Partitioning (ECP) |
| **Test Description** | Test for allowed email formats with specific domain restrictions |
| **Test Data** | Only Gmail or Students email addresses from "riphah.edu.pk" are allowed. |
| **Input Examples** | john.doe@gmail.com, johndoe@students.riphah.edu.pk, john.doe@outlook.com, john#123@gmail.com |
| **Expected Result** | Valid emails should end with either @gmail.com or @students.riphah.edu.pk. |
| **Comments** | Emails from other domains or invalid emails should be rejected. |

Table 4.21 Test Data Seller Sign Up

|  |  |
| --- | --- |
| **Test Data ID** | **TD-10** |
| **Test Data** | Password |
| **Stakeholder** | Seller |
| **Field** | Password |
| **Technique** | Equivalence Class Partitioning (ECP) |
| **Test Description** | Test that the password meets security requirements |
| **Test Data** | Password must be at least 6 characters long and contain at least one special character, one capital letter, and one number. |
| **Input Examples** | P@ssw0rd, admin123!, password, 12345, @123 |
| **Expected Result** | Password should be at least 6 characters long and contain at least one uppercase letter, one number, and one special character. |
| **Comments** | Passwords not fulfilling the conditions should be rejected. |

Table 4.22 Test Data Seller Sign Up

|  |  |
| --- | --- |
| **Test Data ID** | **TD-11** |
| **Test Data** | Contact Number |
| **Stakeholder** | Seller |
| **Field** | Contact Number |
| **Technique** | Equivalence Class Partitioning (ECP) |
| **Test Description** | Test that the contact number matches the specified format (+92 followed by exactly 10 digits). |
| **Test Data** | Contact number must start with +92 and contain exactly 10 digits. |
| **Input Examples** | +923001234567, +921234567890, +929876543210 |
| **Expected Result** | Valid contact numbers should start with +92 and have 10 digits. |
| **Comments** | Invalid numbers or those without the proper format should be rejected. |

Table 4.23 Test Data Seller Sign Up

|  |  |
| --- | --- |
| **Test Data ID** | **TD-12** |
| **Test Data** | Store Name |
| **Stakeholder** | Seller |
| **Field** | Store Name |
| **Technique** | Equivalence Class Partitioning (ECP) |
| **Test Description** | Test that the store name is properly provided and is not empty. |
| **Test Data** | Store name is required and should not be empty. |
| **Input Examples** | John's Electronics, Tech Store, SuperMart, `` |
| **Expected Result** | Store name must be provided and cannot be empty. |
| **Comments** | Empty store names should be rejected. |

Table 4.24 Test Data Seller Sign Up

|  |  |
| --- | --- |
| **Test Data ID** | **TD-13** |
| **Test Data** | Warehouse Address |
| **Stakeholder** | Seller |
| **Field** | Warehouse Address |
| **Technique** | Equivalence Class Partitioning (ECP) |
| **Test Description** | Test that the warehouse address is provided and properly formatted. |
| **Test Data** | Warehouse address is required. |
| **Input Examples** | 123 Warehouse St, Tech Park Warehouse, Warehouse at Sector 5 |
| **Expected Result** | Warehouse address should be provided. |
| **Comments** | Invalid or missing addresses should be rejected. |

Table 4.25 Test Data Seller Sign Up

|  |  |
| --- | --- |
| **Test Data ID** | **TD-14** |
| **Test Data** | Gender |
| **Stakeholder** | Seller |
| **Field** | Gender |
| **Technique** | Equivalence Class Partitioning (ECP) |
| **Test Description** | Test that the gender field is provided and is one of the valid options. |
| **Test Data** | Valid genders must be provided as male, female, or other. |
| **Input Examples** | Male, Female, Other, `` |
| **Expected Result** | Gender must be one of the predefined options. |
| **Comments** | Missing or invalid gender options should be rejected. |

Table 4.26 Test Data Add Product

|  |  |
| --- | --- |
| **Test Data ID** | **TD-15** |
| **Test Data** | Product Name |
| **Stakeholder** | Seller |
| **Field** | Name |
| **Technique** | Equivalence Class Partitioning (ECP) |
| **Test Description** | Test that the product name is provided and is valid |
| **Test Data** | Product name is required. |
| **Input Examples** | Wireless Headphones, Smartphone, Laptop |
| **Expected Result** | The name must be provided and be a non-empty string. |
| **Comments** | Empty product names should be rejected. |

Table 4.27 Test Data Add Product

|  |  |
| --- | --- |
| **Test Data ID** | **TD-16** |
| **Test Data** | Product Price |
| **Stakeholder** | Seller |
| **Field** | Price |
| **Technique** | Equivalence Class Partitioning (ECP) |
| **Test Description** | Test that the product price is a valid numeric value |
| **Test Data** | Price must be a positive number. |
| **Input Examples** | 150, 299.99, 5000, -100 |
| **Expected Result** | Price must be a positive number greater than 0. |
| **Comments** | Negative values or non-numeric inputs should be rejected. |

Table 4.28 Test Data Add Product

|  |  |
| --- | --- |
| **Test Data ID** | **TD-17** |
| **Test Data** | Product Description |
| **Stakeholder** | Seller |
| **Field** | Description |
| **Technique** | Equivalence Class Partitioning (ECP) |
| **Test Description** | Test that the product description is provided and meets the required length |
| **Test Data** | Description is required. |
| **Input Examples** | Wireless headphones with noise-cancellation, 4K UHD smart TV |
| **Expected Result** | Description must be a string and cannot be empty. |
| **Comments** | Empty or excessively short descriptions should be rejected. |

Table 4.29 Test Data Add Product

|  |  |
| --- | --- |
| **Test Data ID** | **TD-18** |
| **Test Data** | Product Category |
| **Stakeholder** | Seller |
| **Field** | Category |
| **Technique** | Equivalence Class Partitioning (ECP) |
| **Test Description** | Test that the product category is provided and valid |
| **Test Data** | Valid categories must be selected. |
| **Input Examples** | Electronics, Home Appliances, Sports |
| **Expected Result** | The category must be provided and match predefined categories. |
| **Comments** | Invalid categories or missing values should be rejected. |

Table 4.30 Test Data Add Product

|  |  |
| --- | --- |
| **Test Data ID** | **TD-19** |
| **Test Data** | Product Subcategory |
| **Stakeholder** | Seller |
| **Field** | Subcategory |
| **Technique** | Equivalence Class Partitioning (ECP) |
| **Test Description** | Test that the product subcategory is provided and valid |
| **Test Data** | Subcategory must be provided. |
| **Input Examples** | Headphones, Smartphones, Laptops |
| **Expected Result** | Subcategory must be provided and match predefined options. |
| **Comments** | Invalid or missing subcategories should be rejected. |

Table 4.31 Test Data Add Product

|  |  |
| --- | --- |
| **Test Data ID** | **TD-20** |
| **Test Data** | Product Quantity |
| **Stakeholder** | Seller |
| **Field** | Quantity |
| **Technique** | Equivalence Class Partitioning (ECP) |
| **Test Description** | Test that the quantity is a valid integer and greater than 0 |
| **Test Data** | Quantity must be a positive number. |
| **Input Examples** | 10, 50, 100 |
| **Expected Result** | Quantity must be a positive integer greater than 0. |
| **Comments** | Negative or non-integer values should be rejected. |

Table 4.32 Test Data Add Product

|  |  |
| --- | --- |
| **Test Data ID** | **TD-21** |
| **Test Data** | Seller ID |
| **Stakeholder** | Seller |
| **Field** | Seller ID |
| **Technique** | Equivalence Class Partitioning (ECP) |
| **Test Description** | Test that the seller ID is valid and exists in the database |
| **Test Data** | Valid seller ID is required. |
| **Input Examples** | 603cf92b217d5f55d8f1d285, UNKNOWN\_SELLER |
| **Expected Result** | Seller ID must be a valid MongoDB ObjectId and must correspond to an existing seller. |
| **Comments** | Invalid or non-existent seller IDs should be rejected. |
| **Test Data ID** | TD-18 |
| **Test Data** | Bargain Rounds |
| **Stakeholder** | Seller |
| **Field** | Bargain Rounds |
| **Technique** | Equivalence Class Partitioning (ECP) |
| **Test Description** | Test that the number of bargain rounds is a valid integer |
| **Test Data** | Bargain rounds must be a positive integer. |
| **Input Examples** | 0, 5, 10 |
| **Expected Result** | Bargain rounds must be a non-negative integer. |
| **Comments** | Invalid or negative numbers should be rejected. |

Table 4.33 Test Data Add Product

|  |  |
| --- | --- |
| **Test Data ID** | **TD-22** |
| **Test Data** | Max Discount Percent |
| **Stakeholder** | Seller |
| **Field** | Max Discount Percent |
| **Technique** | Equivalence Class Partitioning (ECP) |
| **Test Description** | Test that the discount percentage is a valid number and within the allowed range (0-100) |
| **Test Data** | Max discount percent must be between 0 and 100. |
| **Input Examples** | 10, 50, 100, 200, -10 |
| **Expected Result** | Discount percent must be a number between 0 and 100. |
| **Comments** | Values outside the range of 0-100 should be rejected. |

Table 4.34 Test Data Add Product

|  |  |
| --- | --- |
| **Test Data ID** | **TD-23** |
| **Test Data** | Product Images |
| **Stakeholder** | Seller |
| **Field** | Images |
| **Technique** | Equivalence Class Partitioning (ECP) |
| **Test Description** | Test that product images are provided and are in the correct format |
| **Test Data** | Only image files (e.g., .jpeg, .jpg) are allowed. |
| **Input Examples** | image1.jpg, image2.jpeg, image3.png |
| **Expected Result** | Only image files (JPEG, PNG, etc.) should be accepted. |
| **Comments** | Invalid file types should be rejected. |

## 4.5 Test Metrics

Test metrics play a crucial role in evaluating the effectiveness and progress of the testing phase, it will provide a comprehensive overview of the common attributes associated with test case metrics. It will encapsulate key parameters such as test coverage, defect density, and test execution efficiency. These metrics will be instrumental in gauging the thoroughness of testing, identifying potential areas for improvement, and ensuring the overall quality of the developed system. The focus will be on establishing a standardized set of metrics that align with project objectives, fostering a systematic approach to monitoring and enhancing the testing process.

### Sample Test case Matric.No.1

Table 4.35 Test Case Matric

|  |  |
| --- | --- |
| Metric: | Purpose |
| Number of Test Cases: | 34 |
| Number of Test Cases Passed: | 34 |
| Number of Test Cases Failed: | 0 |
| Test Case Defect Density: | 0 |
| Test Case Effectiveness: | 100% |
| Traceability Matrix: | Since all test cases passed, there are no failed cases to trace back to requirements. |

.

## 4.6 Conclusion

In conclusion, Chapter 4 explored the core aspects of implementation and testing phases for the Savify platform. The implementation phase involved the creation of the key components such as the AI-powered bargaining system, multi-vendor order placement functionality, and the backend APIs responsible for managing user interactions, product management, and checkout processing. These elements collectively build the foundation for Savify, offering a smooth and secure user experience while ensuring efficient backend operations.

The test case design and description outlined a structured approach to validating the system's functionality and reliability. The test cases were designed to address input constraints, environmental prerequisites, and procedural dependencies, ensuring the robustness of critical components like user login, AI negotiation and order placement. This comprehensive testing methodology aimed to confirm the platform's security, performance and operational efficiency.

Looking ahead, the test metrics discussed in this chapter will serve as a guiding framework for continuous improvement. Metrics such as test coverage, defect density, and execution efficiency will provide valuable insights into the quality of the testing process. By analyzing these metrics, the project team will be able to optimize Savify, ensuring that it meets high standards of performance and reliability. Chapter 4 thus lays a solid foundation for the next phases, emphasizing the importance of the strong implementation and thorough testing in the successful development of Savify.

**Chapter 5: Experimental Result and Analysis**

## 5.1 Introduction

This chapter presents the evaluation and analysis of the AI discount prediction module integrated into Savify—a multivendor e-commerce platform featuring AI-driven price negotiation and image-based product search. The focus of this chapter is to assess the model’s training progression, accuracy, and practical performance using standard regression evaluation metrics. The core model used is XGBoost, trained to predict personalized discount percentages based on buyer and product features.

## 5.2 Model Performance Analysis

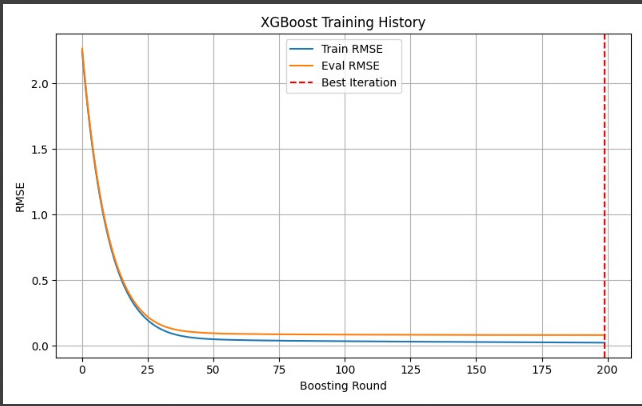


Figure 5.1 indicating the XGBoost model is learning effectively without overfitting

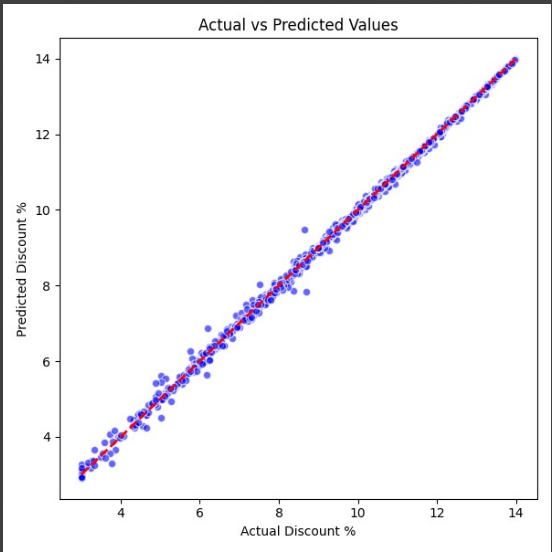


Figure 5.2 shows model's predicted discount percentages closely match the actual values

### 5.2.1 Training History and RMSE Analysis

The training process was tracked using Root Mean Squared Error (RMSE) for both training and evaluation datasets across 200 boosting rounds. The following trends were observed:  
  
- Rapid Decline: RMSE values for both training and evaluation sets dropped sharply within the initial iterations.  
- Convergence: Evaluation RMSE flattened out and converged with training RMSE, suggesting excellent generalization without overfitting.  
- Stable Learning: The model maintained stability and avoided variance across epochs.  
  
Training and Evaluation RMSE Over Iterations  
  
The graph illustrates a smooth decline in both training and validation RMSE. By the end of the training phase, both curves are nearly parallel, confirming that the model has reached optimal performance without excessive training.

### 5.2.2 Final RMSE Values

At the final iteration, the model achieved the following RMSE values:  
  
- Final Training RMSE: 0.025   
- Final Evaluation RMSE: 0.083  
  
These values indicate that the predicted discounts closely match the actual discounts during both training and validation, reflecting high predictive precision.

## 5.3 Model Evaluation Metrics

|  |  |
| --- | --- |
| Metric | Value |
| Mean Absolute Error (MAE) | 0.042 |
| Mean Squared Error (MSE) | 0.0027 |
| Root Mean Squared Error (RMSE) | 0.083 |
| R-squared (R²) | 0.977 |

- MAE confirms that on average, the prediction deviates by just 0.042 from the actual discount.  
- MSE and RMSE are both low, indicating minimal large errors.  
- R² Score of 0.977 shows that 97.7% of the variance in the actual discount values is explained by the model—demonstrating excellent fit and high reliability.

## 5.4 Visual Evaluation: Actual vs Predicted

A scatter plot comparing actual and predicted discount values shows a dense alignment along the diagonal. This strong linear trend indicates minimal deviation between predicted and real values across the full range of discounts. The model effectively learns discount patterns based on features such as buyer behavior and product ratings.

## 5.5 Integration into Savify

This AI-powered discount module is integrated into Savify’s buyer negotiation system, enabling:  
  
- Smart counter-offers based on buyer profile and product rating.  
- Real-time dynamic discount predictions during negotiations.  
- Enhanced user trust through consistent, personalized bargaining.  
  
Sellers predefine boundaries (max discount, negotiation rounds), and the model ensures the buyer cannot negotiate below the threshold, protecting seller interests while promoting transparency.

## 5.6 Future Enhancements

To further improve the system, the following directions are planned:  
  
- Online model retraining using real-time feedback from accepted/rejected offers.  
- Hybrid approaches combining rule-based filters and AI predictions.  
- Integration of behavioral data (e.g., session time, abandonment rate) into the discount model.

## 5.7 Conclusion

The experimental results demonstrate that the AI discount prediction model within Savify performs with high accuracy, excellent generalization, and minimal prediction error. With RMSE below 0.1 and an R² of 0.977, the model is highly effective and reliable for real-world bargaining scenarios. Its seamless integration into the platform enhances user engagement, automates negotiations, and aligns seller and buyer interests through intelligent pricing decisions.

**Chapter 6: Conclusion and Future Direction**

**6.1 Conclusion and Future Directions**

In this chapter, we wrap up the Savify project and explore what’s next. From thebeginning, our main goal was to create a smart, scalable online shopping platform that supports multiple sellers. We wanted to tackle the common problems in e-commerce by adding AI-powered bargaining and image search features. Savify was built to make buying and selling smoother and smarter, especially by using machine learning to personalize and automate price negotiations.

We made strong progress toward this goal. The platform now includes an automated price negotiation system that uses an XGBoost model to predict discounts. Sellers can set rules like how many times a buyer can negotiate and the maximum discount allowed. On the buyer’s side, an AI tool helps negotiate prices based on factors like past purchases, product ratings, and more. We also added an image search feature, so users can find products just by uploading a photo—making the platform more user-friendly and engaging.

Of course, we faced some challenges along the way. For example, keeping the model accurate for all types of buyer behavior was tricky. It was also complex to manage negotiations happening at the same time with many sellers. Integrating the AI features smoothly into our MERN-based platform took extra effort. Even though our system supports both product-level and cart-level bargaining, there are still improvements to be made—for example, handling cases where vouchers expire, items are removed from the cart, or the cart updates during negotiation.

Still, we achieved most of what we set out to do. Savify now stands as a promising platform that brings AI into the world of online shopping in a meaningful way. It automates the negotiation process, reduces the work sellers need to do manually, and gives buyers fair, personalized discount offers.

Looking to the future, we see many ways to improve Savify. One idea is to use reinforcement learning to make the bargaining system even smarter and more adaptable. We also want to add a system that updates the AI models in real-time, keeping them accurate and fresh. Using natural language processing could let the system adjust based on how the buyer is feeling during negotiation. We also aim to upgrade the image search feature using more advanced computer vision, and even explore voice commands and augmented reality previews for products.

To keep improving, it’s important that we work with experts and listen to feedback from users. By staying up to date with new technologies and what people want, Savify can become a truly next-generation shopping platform—smart, user-focused, and ready to change the way people buy and sell online.

# References