SWIFTBUY

A PROJECT REPORT

Submitted by

ADITHYA VINOD (SNG22CS015)

ADWAID MANOJ (SNG22CS016)

AJAY DAS (SNG22CS021)

ANAMIKA S (SNG22CS033)

TO

The APJ Abdul Kalam Technological University

in partial fulfillment of the requirements for the award of the Degree of

Bachelor of Technology

In

Computer Science and Engineering

Department of Computer Science and Engineering

Sree Narayana Gurukulam College of Engineering,

Kadayiruppu, 682311

APRIL 2025

DECLARATION

We undersigned hereby declare that the project report “SWIFTBUY” submitted for partial fulfillment of the requirements for the award of degree of Bachelor of Technology of the APJ Abdul Kalam Technological University, Kerala is a bonafide work done by us under supervision of Asst. Prof. Vinila V . This submission represents our ideas in our own words and where ideas or words of others have been included. We have adequately and accurately cited and referenced the original sources. We also declare that we have adhered to ethics of academic honesty and integrity and have not misrepresented or fabricated any data or idea or fact or source in our submission. We understand that any violation of the above will be a cause for disciplinary action by the institute and/or the University and can also evoke penal action from the sources which have thus not been properly cited or from whom proper permission has not been obtained. This report has not been previously formed the basis for the award of any degree, diploma or similar title of any other University.

Place: Kadayiruppu

Date: 03/04/2025

Adithya Vinod

Adwaid Manoj

Ajay Das

Anamika S

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING, SREE NARAYANA GURUKULAM COLLEGE OF ENGINEERING, KADAYIRUPPU,682311

(Affilated to APJ Abdul Kalam Technological University & Approved by A.I.C.T.E)

CERTIFICATE

This is to certified that the project report,"**SwiftBuy**",Submitted by Adithya Vinod, Adwaid Manoj, Ajay Das, Anamika S to the APJ Abdul Kalam Technological University in partial fulfillment of the requirements for the award of the Degree of Bachelor of Technology in Computer Science and Engineering is a bona fide record of the project work carried out by them under our guidance and supervision .This report in any form has not been submitted to any other University or Institute for any purpose.

Prof. (Dr.) Smitha Suresh	**Asst. Prof. Vinila V**	**Assoc. Prof. Saini Jacob Soman**
HEAD OF THE DEPARTMENT	**PROJECT GUIDE**	**PROJECT COORDINATOR**

Submitted for the University Evaluation on …….

University Register No ……..……

INTERNAL EXAMINER EXTERNAL EXAMINER

ACKNOWLEDGEMENT

Dedicating this project to the Almighty God whose abundant grace and mercy enabled its successful completion, we would like to express our profound gratitude to all the people who had inspired and motivated us to undertake this project.

We wish to express our sincere thanks to our Head of the Department, **Prof. (Dr.) Smitha Suresh**, for providing us the opportunity to undertake this project. We are deeply indebted to our project coordinator **Assoc. Prof**. **Saini Jacob Soman** and project guide **Asst. Prof**. **VinilaV** in the Department of Computer Science and Engineering for providing us with valuable advice and guidance during the course of the project.

Finally, we would like to express my gratitude to Sree Narayana Gurukulam College of Engineering for providing me with all the required facilities without which the successful completion of the project work would not have been possible.

COURSE OUTCOME AND PROGRAM OUTCOME

Course Outcome	
CO1	Identify technically and economically feasible problems (Cognitive Knowledge Level: Apply)
CO2	Identify and survey the relevant literature for getting exposed to related solutions and get familiarized with software development processes (Cognitive Knowledge Level: Apply)
CO3	Perform requirement analysis, identify design methodologies and develop adaptable & reusable solutions of minimal complexity by using modern tools &advanced programming techniques (Cognitive Knowledge Level: Apply)
CO4	Prepare technical report and deliver presentation (Cognitive Knowledge Level: Apply)
CO5	Apply engineering and management principles to achieve the goal of the project (Cognitive Knowledge Level: Apply)

Program outcomes	
Engineering Graduates will be able to:	
PO1.	Engineering knowledge: Apply the knowledge of mathematics, science, engineeringfundamentals, and an engineering specialization to the solution of complex engineeringproblems.
PO2.	Problem analysis: Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.
PO3.	Design/development of solutions: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.
PO4.	Conduct investigations of complex problems: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.
PO5.	Modern tool usage: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.
PO6.	The engineer and society: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice
PO7.	Environment and sustainability: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.
PO8.	Ethics: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
PO9.	Individual and team work: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.

PO10.	Communication: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.

PROGRAM SPECIFIC OUTCOMES (PSO's)	
PSO1.	Shall enhance the employability skills by finding innovative solutions for challenges and problems in various domains of CS.
PSO2.	Shall apply the acquired knowledge to develop software solutions and innovative mobile applications for various problems.

CO PO PSO MAPPING

	PO1 (Engineering Knowledge)	PO2 (Problem Analysis)	PO3(Design/Development of Solution)	PO4 (Investigation of complex problem)	PO5 (Modern tool usage)	PO6 (The Engineer and Society)	PO7 (Environment and Sustainability)	PO8 (Ethics)	PO9 (Individual and team work)	PO10 (Communication)	PO11(Management and Finance)	PO12 (Life long learning)	PSO1 finding innovative solution	PSO2 Software envelopment
CO1	2	2	2	2		2	2	2	2	2	2	2	2	2
CO2	2	2	2	2	2	2		2	2	2	2	2	2	2
CO3	2	3	3	3	3	2	2	2	3	3	2	2	3	3
CO4	2	2	2	2	2			2	2	2	2	2		2
CO5	3	3	3	3	3	2	2	2	2		3	3	3	3
AVERAGE	2.2	2.4	2.4	2.4	2.5	2.0	2.0	2.0	2.2	2.25	2.2	2.2	2.5	2.4

PO PSO Attainment		
PO	Attained Point (0/1/2/3)	Justification
PO1	3	The system applies engineering knowledge in software development and cybersecurity to improve e-commerce. It ensures secure payments and scalable architecture.
PO2	3	User preferences are analyzed to optimize search algorithms. Large-scale data analysis refines recommendations and enhances user interactions.
PO3	3	Advanced filtering and secure transactions improve online shopping. The system maintains efficiency and user friendliness.
PO4	3	Real-time tracking and data analytics optimize operations and decision-making. Customer behavior analysis enhances satisfaction.
PO5	3	Modern web technologies and encrypted transactions enhance security and reliability. The system ensures seamless user engagement.
PO6	2	Secure transactions and data protection uphold ethical shopping. Customer privacy is safeguarded while preventing cyber threats.
PO7	1	Sustainability is not a core focus but can be promoted through eco-friendly filters and responsible brand recommendations.
PO8	3	Ethical considerations are maintained through data privacy policies and compliance with security standards in e-commerce.
PO9	3	Engineers, designers, and security experts collaborate to create an efficient and user-friendly shopping experience.
PO10	3	An intuitive UI, real-time notifications, and order tracking ensure clear communication and improve user engagement.
PO11	3	Financial transactions and project management sustain feature enhancements and long-term scalability.

PO12	3	The system evolves with new technologies, integrating innovations to meet e-commerce demands.
PSO1	3	The project enhances employability by showcasing expertise in data analytics and secure web development.
PSO2	3	Engineering principles are applied to create an integrated platform that ensures a smooth shopping experience.

ABSTRACT

SwiftBuy is designed to offer a simple, user-friendly approach to online shopping, making it easy for users to browse and buy products across a variety of categories, including electronics, fashion, and groceries. Rather than overwhelming users with cluttered interfaces or complex tools, SwiftBuy focuses on clarity, speed, and ease of navigation.

With efficient search and filtering options, users can quickly narrow down their choices and find what they're looking for. The platform supports key features like wishlists and product reviews, helping shoppers stay organized and make informed decisions based on community feedback.

While currently supporting only cash on delivery, SwiftBuy ensures a smooth and secure checkout process. Its clean design and straightforward functionality create a stress-free shopping experience, especially for users who value simplicity and reliability.

In essence, SwiftBuy brings together essential e-commerce elements in a streamlined format—prioritizing usability, trust, and a hassle-free experience for every shopper.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATION

Abbreviation	**Full Form**
AI	Artificial Intelligence
NLP	Natural Language Processing
ML	Machine Learning
ANN	Artificial Neural Network
CNN	Convolutional Neural Network
LSTM	Long Short-Term Memory
RNN	Recurrent Neural Network
SVM	Support Vector Machine
NB	Naive Bayes
LR	Logistic Regression
DT	Decision Tree
RF	Random Forest
KNN	K-Nearest Neighbors
POS	Part Of Speech
BoW	Bag of Words
SA	Sentiment Analysis
E-commerce	Electronic Commerce
IoT	Internet of Things

CHAPTER :1
INTRODUCTION

SwiftBuy is a user-centric e-commerce platform developed to simplify and enhance the online shopping experience. Designed with accessibility and ease of use in mind, the platform allows users to browse, filter, and purchase products across various categories, including electronics, fashion, and groceries. SwiftBuy emphasizes straightforward functionality, offering essential features such as wishlists and customer reviews to support informed decision-making. With a clean, intuitive interface and a secure checkout process supporting cash on delivery, SwiftBuy caters to a wide range of users, providing a reliable and convenient solution for everyday online shopping needs.

1.1 Overview:

SwiftBuy is a simplified e-commerce web application developed to offer a smooth and efficient shopping experience. The platform enables users to explore a wide range of products in categories like electronics, fashion, and groceries. It incorporates essential features such as product filtering, wishlists, and customer reviews to help users find and manage their desired items easily. SwiftBuy emphasizes a clean and intuitive user interface, ensuring easy navigation for both new and experienced users. With a focus on security and convenience, the platform currently supports cash on delivery, offering a trusted and hassle-free checkout process. Overall, SwiftBuy is designed to deliver a reliable, accessible, and user-friendly online shopping environment.

1.2 Scope:

The scope of SwiftBuy focuses on delivering a simplified and efficient online shopping experience by implementing core e-commerce functionalities without unnecessary complexity. Key functionalities include:

- **Product Management**: Browsing and filtering a wide range of products across categories like electronics, fashion, and groceries.
- **User Interaction**: Enabling users to create wishlists, view customer reviews, and manage their selected items.
- **Order Handling**: Facilitating a straightforward checkout process with support for cash on delivery as the current payment method.
- **Interface Design**: Providing a clean, responsive design

1.3 FEATURES :

SwiftBuy makes online shopping easy, convenient, and user-friendly. Instead of adding unnecessary features, it focuses on what users need for a smooth and reliable shopping experience. Here are some key features that make SwiftBuy stand out:

- **Simple Shopping Interface** – SwiftBuy offers an intuitive layout that allows users to easily browse, search, and explore products across various categories.
- **Essential Product Filtering** – Users can quickly narrow down product choices using basic filters, helping them find exactly what they need without hassle.
- **Wishlist and Reviews** – Shoppers can save favorite items to wishlists and refer to customer reviews to make better-informed decisions.
- **Cash on Delivery** – The platform supports a secure and straightforward checkout process with cash on delivery as the current payment method.
- **Lightweight and Responsive Design** – SwiftBuy is optimized for fast performance and consistent functionality across devices, even with limited resources.
- **User-Friendly Experience** – Designed for everyday users, SwiftBuy requires no learning curve, making online shopping accessible to all.

CHAPTER :2

LITERATURE SURVEY

Research in the field of online product decision-making has led to the development of sophisticated models that leverage sentiment analysis and fuzzy logic for improved recommendations. One notable study presents a novel approach that integrates BiLSTM-CRF, sentiment analysis, and K-means clustering to mine product attributes from cross-platform reviews. The authors propose a multi-platform online decision framework based on the q-Rung Orthopair Fuzzy Cloud (q-ROFC) model, which enhances sentiment error analysis and product selection. This method significantly improves sentiment mining accuracy and decision-making effectiveness. However, challenges such as real-time scalability, fake reviews, and uneven review distribution remain. Future work envisions expanding applications beyond e-commerce and creating real-time systems for live consumer behavior analysis.[1]

Another research effort evaluates the usability and security of e-commerce websites using a multi-criteria decision-making framework. This approach employs the Analytic Hierarchy Process (AHP), VIKOR, and TOPSIS to assess performance based on user survey data and security scans from platforms like SUCURI and Qualys. The study effectively combines quantitative and qualitative metrics to offer insights into e-commerce usability and cybersecurity, particularly in rural settings. However, the study is constrained by a limited sample size and geographic scope, with no real-time adaptability. Future research aims to integrate dynamic data and expand the framework for global application. [2]

A comprehensive review explores current sentiment analysis techniques used in e-commerce platforms. By analyzing 54 experimental studies, this paper categorizes sentiment analysis methods into machine learning (e.g., SVM, Naïve Bayes) and deep learning (e.g., LSTM, BERT) techniques. The study highlights research gaps, such as sarcasm detection and implicit aspect extraction, while also identifying a lack of focus on non English datasets. Although the review provides a valuable taxonomy and comparison, it lacks experimental validation. Future directions include developing universal, multi-language sentiment models and refining aspect-level sentiment analysis. [3]

A systematic literature review of hybrid recommendation systems offers insights into their evolution and effectiveness in e-commerce. This study analyzes 48 papers and identifies trends in accuracy, scalability, and personalization through hybrid approaches. It addresses key challenges such as data sparsity and the cold start problem, suggesting that AI integration, explainable models, and multimodal data fusion are promising future directions. Despite improvements, the complexity of adapting to dynamic user behavior remains an open research issue. [4]

Another study investigates the impact of computer applications on enhancing cross-border e-commerce performance. The researchers employ deep learning techniques (CNNs) and Genetic Algorithms (GAs) to optimize data analytics and improve customer satisfaction. The proposed approach enhances operational efficiency and predictive accuracy. However, the method faces hurdles related to computational complexity, sensitivity to parameters, and regulatory compliance. Future research is expected to explore blockchain for secure transactions and AI personalization techniques while addressing ethical concerns. [5]

A hybrid recommendation model combining ontology-based knowledge with sequential pattern mining has been proposed to boost e-commerce personalization. The OntoCommerce system uses domain-specific ontologies and pattern discovery to address data sparsity and cold-start problems. While it provides highly personalized recommendations, the model is computationally expensive and reliant on accurate ontology construction. The authors recommend expanding this approach to other domains and focusing on improving system scalability. [6]

Advancements in AI-based customer service in e-commerce are the focus of another study that introduces an end-cloud collaboration framework. The system combines cloud-based large models with on-device AI, enabling privacy-preserving, adaptive, and real-time customer support. Benefits include computational efficiency and personalization, but the framework demands high initial setup costs and expensive training. Future research should explore industrial applications and more efficient optimization techniques using large datasets. [7]

Computer vision techniques have also been used to improve product image quality in e-commerce platforms. One such method uses Fast-SAM instance segmentation, background replacement, and shadow generation to enhance visual appeal and recognition accuracy. While the technique improves image aesthetics and maintains efficiency, its effectiveness is limited in diverse product environments and real-time scenarios. Future work should focus on expanding dataset diversity and exploring applications in other visual domains like healthcare and digital art. [8]

CHAPTER :3

PROBLEM STATEMENT AND OBJECTIVE

3.1 Problem Statement:

Modern e-commerce platforms often prioritize feature-rich designs and complex personalization, which can lead to cluttered interfaces, slower performance, and a frustrating experience for users—especially those who prefer simplicity and speed. For many shoppers, basic needs such as easy product browsing, efficient filtering, and a straightforward checkout are buried under layers of unnecessary functionality or require high-speed connections and advanced devices to function smoothly.

Additionally, small businesses or new users looking for streamlined platforms face challenges when adapting to systems designed for large-scale operations. Existing platforms may also demand integration with multiple payment gateways and third-party services, which adds to the technical and financial burden.

SwiftBuy addresses these challenges by offering a lightweight, easy-to-use e-commerce solution focused on core shopping functionalities. With a clean interface, basic filters, wishlist support, and a simple checkout system (currently supporting cash on delivery), SwiftBuy is built for accessibility, speed, and ease of use—making online shopping convenient and efficient for a wide range of users.

3.2 Objective:

SwiftBuy was built with a clear mission: to simplify the online shopping experience by focusing on usability, speed, and accessibility. It achieves this by:

- **Delivering a Clean and Efficient Shopping Interface**: SwiftBuy eliminates unnecessary clutter, allowing users to browse and shop with ease across essential product categories like fashion, electronics, and groceries.
- **Simplifying Product Discovery with Essential Filters:** Instead of overwhelming users with complex search tools, SwiftBuy offers straightforward filtering options that help shoppers find what they need quickly.

- **Supporting Seamless and Secure Checkout:** With cash on delivery as its primary payment method, the platform ensures a simple, secure, and familiar transaction process for users of all backgrounds.
- **Enhancing Usability Through Lightweight Design:** SwiftBuy is built to perform smoothly on a wide range of devices, offering fast load times and a responsive experience even in low-bandwidth environments.
- **Making E-commerce More Accessible:** With features like wishlists, customer reviews, and intuitive navigation, SwiftBuy lowers the barrier to entry for online shopping, making it easy for anyone—from first-time buyers to regular users.

By combining these core elements, SwiftBuy empowers users with a fast, reliable, and user-friendly platform—redefining how simple and efficient online shopping can be.

CHAPTER :4
REQUIREMENT SPECIFICATION

4.1 Functional Requirements:

- **Product Browsing and Filtering:** Allow users to view a wide range of products and apply basic filters such as category and price range to refine their search
- **User Registration and Login:** Enable users to create accounts, log in securely and manage their profiles.
- **Cart and Wishlist Management:** Provide features to add, remove, and update items in a shopping cart or wishlist for future purchases.
- **Order Placement and Tracking:** Support placing orders with confirmation and basic tracking updates from order placement to delivery.
- **Cash on Delivery Checkout:** Implement a secure and simple checkout process using cash on delivery as the payment method.
- **Product Review Display:** Show customer reviews and ratings for products to help users make informed buying decisions.
- **Responsive User Interface:** Ensure a lightweight, intuitive interface that works smoothly across various devices and screen sizes.

4.2 Non-Functional Requirements:

- **Usability:** The platform must provide an intuitive and user-friendly interface that supports effortless navigation for users of all experience levels.

- **Performance:** SwiftBuy should deliver fast page loads and responsive interactions, even when handling a large number of products or user sessions.
- **Reliability:** he system must function consistently across different devices and browsers, ensuring that users can complete their shopping process without errors or interruptions..
- **Maintainability:** The codebase should be modular and well-documented, allowing for easy updates, scalability, and integration of future enhancements.

4.3 System Requirements :

- **Software:**

  To run Tracescape, you'll need Python 3 (version 3.7 or later is best). The project uses a few handy libraries:

  - **Node.js and npm:** or managing frontend dependencies and running the development server.
  - **MongoDB:** A NoSQL database used to store product listings, user information, and order details.
  - **Express.js:** A lightweight backend framework for handling API routes and server-side logic.
  - **React.js (with Vite):** Powers the frontend, providing a fast and interactive user interface.
  - **Mongoose**: Connects and manages interactions between the backend and MongoDB.

- **Hardware:**

  You don’t need a high-end machine to run **SwiftBuy**—a standard modern computer will do just fine. An Intel i3, i5, or i7 processor provides enough power to run both the backend and frontend smoothly. With at least 8GB of RAM, you’ll have a comfortable experience, though 16GB is ideal for multitasking or handling heavier workloads during development. A 256GB hard drive will cover your storage needs, but using an SSD can significantly speed things up, especially when dealing with larger datasets or dependencies. Lastly, make sure you have a reliable high-speed internet connection for installing packages, loading assets, and testing online features without delays.

CHAPTER :5

SYSTEM DESIGN

5.1 ER-Diagram

An Entity-Relationship (ER) diagram is a conceptual blueprint that visually represents the structure of a database. It illustrates the entities involved in a system, the attributes of those entities, and the relationships between them.

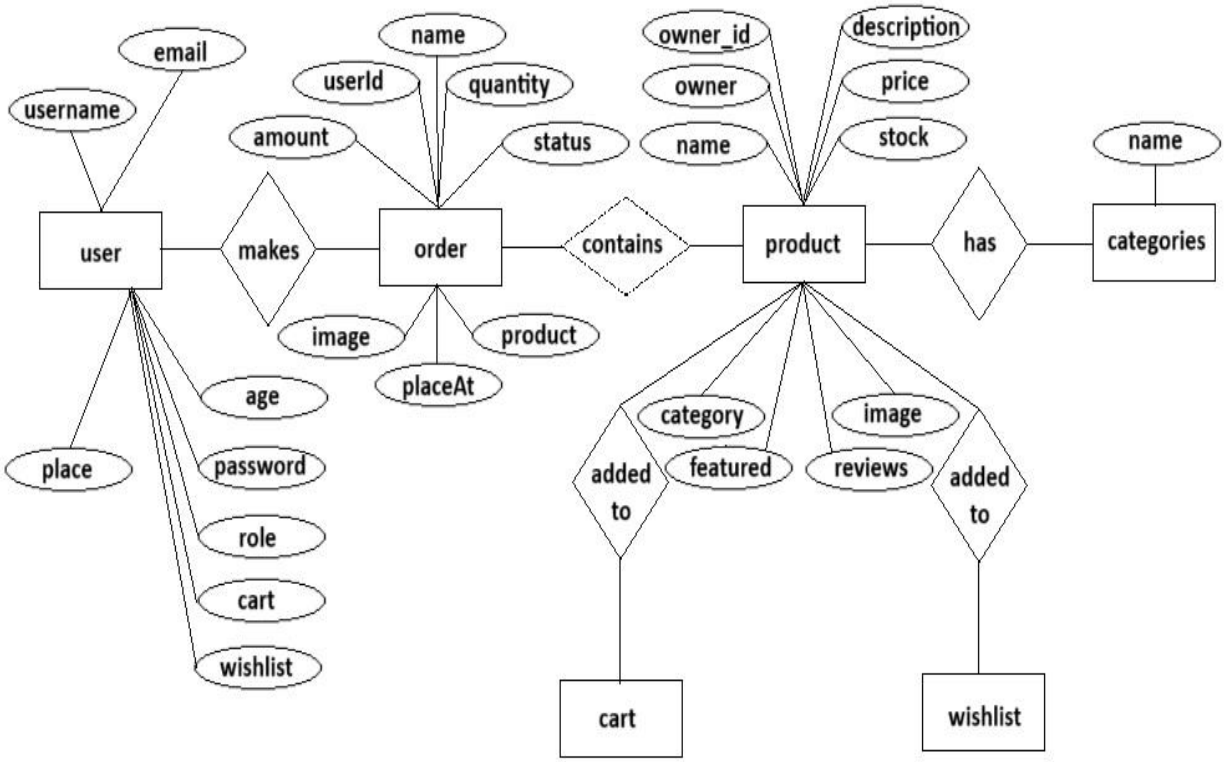

Figure 5.1: ER-Diagram of SwiftBuy

This Entity-Relationship (ER) diagram illustrates the structural framework of a typical e-commerce application, highlighting how various entities interact with each other. At the core of the system are

five primary entities: user, order, product, cart, and wishlist. Each entity is represented as a rectangle, while their attributes are shown as ovals connected by straight lines. Relationships between entities are depicted as diamonds, describing how data flows and is interconnected within the system.

The diagram starts with the user entity, which holds attributes such as username, email, password, age, place, role, and links to both cart and wishlist.A user can make an order, which records details like userId, amount, image, name, quantity, status, and placeAt. Each order is associated with one or more product entries through a contains relationship.

The product entity is rich in attributes, including owner_id, owner, name, description, price, stock, category, featured, image, and reviews. Products are linked to categories via a has relationship and can also be added to either a cart or a wishlist.

These relationships provide a comprehensive structure for organizing user interactions with products and managing transactions in the system. By representing key data elements and their associations, the ER diagram serves as a blueprint for database design, supporting data integrity, scalability, and future system enhancements.

5.2 Use-Case Diagram

A use case diagram is a high-level visual representation used in software engineering to depict how different users, known as actors, interact with a system to accomplish specific tasks or goals, referred to as use cases. It is a component of the Unified Modeling Language (UML) and is typically utilized in the early stages of system design and requirements gathering to clearly convey the intended functionalities of a system.

The diagram focuses on what the system should do rather than how it does it. This makes it an effective communication tool for both technical teams and non-technical stakeholders, helping to ensure a shared understanding of system functionality and user expectations.

In a standard use case diagram, actors are illustrated as stick figures that represent users or external systems. Use cases are displayed as ovals and represent the different services or functions the system provides. These elements are placed within a rectangular system boundary, which outlines the scope of the system.

The relationships in a use case diagram are key to its usefulness. Associations link actors to the use cases they interact with. Additional relationships, such as include and extend, show how use cases can

be reused or optionally extended with additional behavior. Generalization can also be used to indicate hierarchical relationships between actors or use cases.

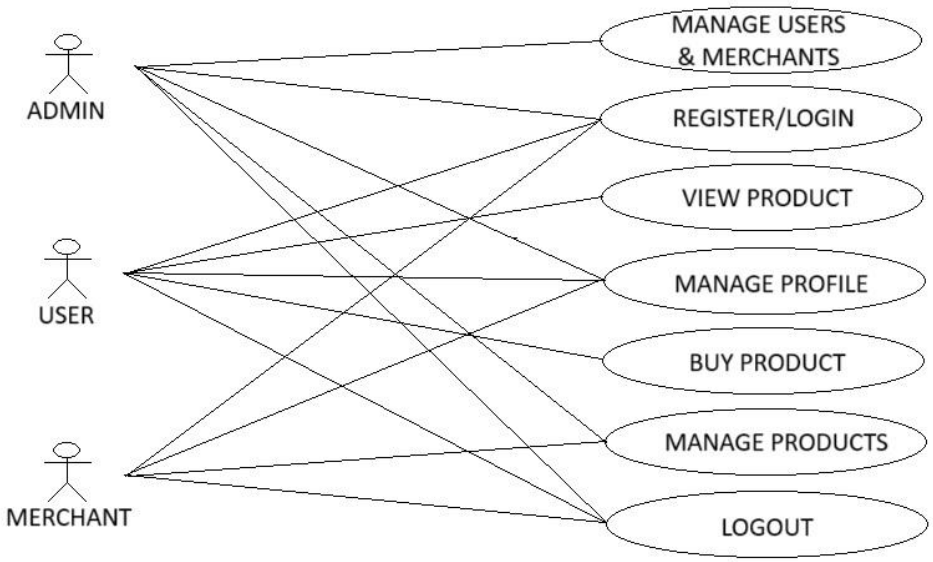

Figure 5.2 : Use Case Diagram of SwiftBuy

This use case diagram illustrates the various interactions between the key actors—Admin, User, and Merchant—and the system functionalities. Each actor has distinct responsibilities and access levels, which are mapped to specific use cases that define how they interact with the system.

The **Admin** has the ability to manage both users and merchants, allowing full administrative control over the platform. In addition, the admin can register or log in to the system, view products listed on the platform, manage their own profile settings, and log out securely.

The **User** role is central to the e-commerce experience. Users can register or log in, browse and view available products, manage their profile settings, and purchase items through the platform. Logout functionality is also available to ensure session security.

The **Merchant**, responsible for supplying products, can register or log in, view products, manage their personal profiles, and most importantly, manage products. This includes adding, updating, or removing products from the catalog. Like other users, they also have access to secure logout functionality.

This diagram effectively captures how different roles access various features of the system, providing a high-level overview of the user interactions within the e-commerce platform. It serves as a visual blueprint for understanding functionality allocation, streamlining both development and future updates to the system.

5.3 Working Architecture

Working architecture refers to the structured design of a system that defines how different components interact to achieve a specific function. It includes the logical and physical structure of software or hardware systems, ensuring efficient data flow, processing, and output generation. A well-defined architecture enhances scalability, maintainability, and performance.

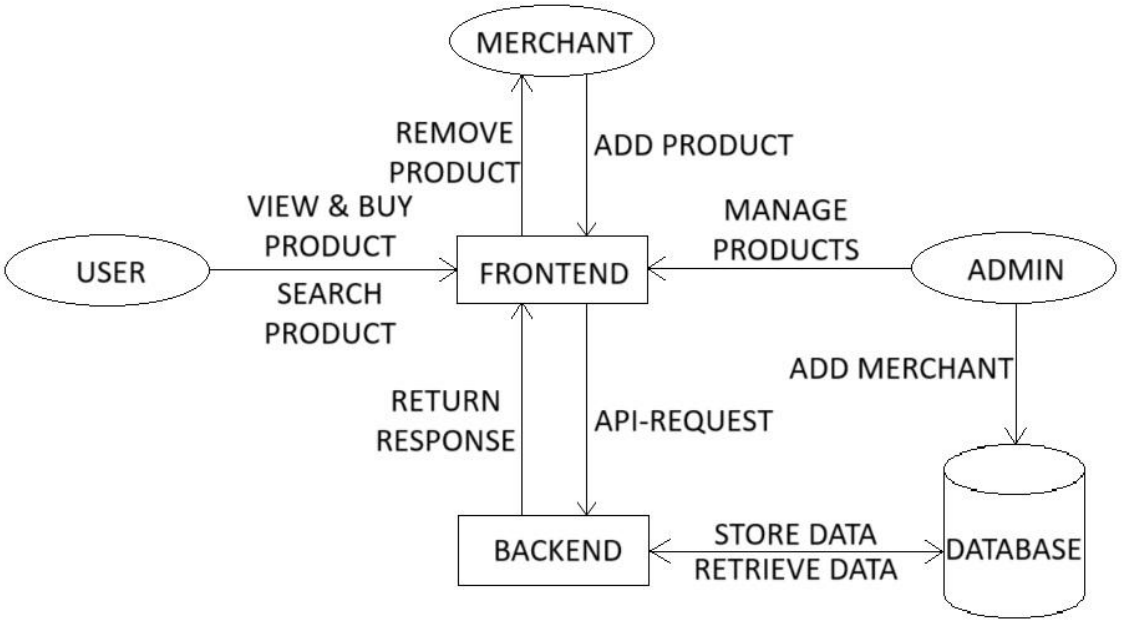

Figure **5.3: Working architecture of SwiftBuy**

This working architecture illustrates how the SwiftBuy system handles user interaction, product management, and data processing across various roles. It begins with the **Frontend**, which serves as the central interface for Users, Merchants, and Admins. Users can search for products, view details, and complete purchases through the frontend interface.

Merchants interact with the frontend to add or remove products from the platform. These actions trigger API requests to the **Backend**, which processes the operations and updates the **Database** accordingly. Similarly, Admins manage the system by adding merchants and overseeing product management. Admin-level operations also communicate with the backend to ensure proper data handling and security.

The **Backend** acts as a middleware that handles API requests from the frontend and performs operations like data validation and business logic processing. It then interacts with the **Database** to store or retrieve necessary data, such as user information, product details, and merchant records. The responses are then returned to the frontend, completing the cycle of interaction.

This layered approach ensures a clear separation of roles and responsibilities, enabling a smooth and scalable e-commerce experience across all user types.

CHAPTER :6

SYSTEM IMPLEMENTATION

6.1 PROGRAMMING LANGUAGE

- **JAVASCRIPT :**

SwiftBuy is built using JavaScript, leveraging both frontend and backend technologies to deliver a seamless e-commerce experience. JavaScript was chosen for its flexibility, large ecosystem, and ability to create dynamic, responsive applications.

Several key technologies and libraries make SwiftBuy work efficiently:

- Node.js – Powers the backend server, handles API requests, and manages data flow between the client and the database.
- Express.js – A minimal and flexible Node.js framework used to create robust RESTful APIs for handling products, users, and orders.
- React.js – Manages the frontend interface, enabling fast and interactive user experiences with features like product listing, filtering, and cart updates.
- MongoDB and Mongoose Store and manage data such as product details, user information, and orders with a schema-based NoSQL database.

6.2 IDE PLATFORM

- **VISUAL STUDIO CODE :**

SwiftBuy was developed entirely using Visual Studio Code (VS Code), chosen for its lightweight yet powerful features. Its intuitive interface, built-in terminal, and excellent JavaScript and React support made coding, debugging, and testing an efficient process. VS Code provided everything needed in one place—smart autocompletion, easy-to-use extensions, and an integrated environment for smooth development. This allowed us to focus on refining SwiftBuy's functionality without unnecessary complexity, ensuring a streamlined workflow from start to finish.

6.3 SOURCE CODE

6.3.1 APP.JSX

```
import { BrowserRouter, Route, Routes } from "react-router-dom";
import Login from "./components/Login";
import Signin from "./components/Signin";
import colors from "./colors";
import { AppContext } from "./AppContext";
import { useState } from "react";
import Profile from "./components/Profile";
import ProductAdd from "./components/ProductAdd";
import MyProduct from "./components/MyProduct";
import SearchProduct from "./components/SearchProduct";
import Users from "./components/Users";
import Products from "./components/Products";
import Dashboard from "./components/Dashboard";
import DetailedProduct from "./components/DetailedProduct";
import MyCart from "./components/MyCart";
import MyOrder from "./components/MyOrder";
import Wishlist from "./components/Wishlist";
import AdminTools from "./components/AdminTools";

function App() {
  const [data, setData] = useState({
    username: "",
    place: "",
    age: "",
    email: "",
    password: "",
    role: "",
```

```
  });
  document.body.style.backgroundColor = colors.backgroundcolor;

  return (
    <>
      <BrowserRouter>
        <AppContext.Provider value={{ data, setData }}>
          <Routes>
            <Route path="/" element={<Login />} />
            <Route path="/signin" element={<Signin />} />
            <Route path="/dashboard" element={<Dashboard />} />
            <Route path="/profile" element={<Profile />} />
            <Route path="/product/add" element={<ProductAdd />} />
            <Route path="/merchant/products" element={<MyProduct />} />
            <Route path="/search/products" element={<SearchProduct />} />
            <Route path="/admin/users" element={<Users />} />
            <Route path="/admin/products" element={<Products />} />
            <Route path="/detproduct" element={<DetailedProduct />} />
            <Route path="/mycart" element={<MyCart />} />
            <Route path="/user/orders" element={<MyOrder />} />
            <Route path="/user/wishlist" element={<Wishlist />} />
            <Route path="/admintools" element={<AdminTools />} />
          </Routes>
        </AppContext.Provider>
      </BrowserRouter>
    </>
  );
}

Export default App;
```

6.3.2 LOGIN.JSX

```
import React, { useContext, useState } from "react";
import { Box, Button, TextField, Typography } from "@mui/material";
import { Link, useNavigate } from "react-router-dom";
import axios from "axios";
import styles from "../styles";
import { AppContext } from "../AppContext";

const Login = () => {
  const { setData } = useContext(AppContext);
  const [user, setUser] = useState({ email: "", password: "" });
  const [errors, setErrors] = useState({ email: false, password: false });
  const [generalError, setGeneralError] = useState("");
  const navigate = useNavigate();
  const api_url = import.meta.env.VITE_API_URL;

  const inputHandler = (e) => {
    setUser({ ...user, [e.target.name]: e.target.value });
    setErrors({ ...errors, [e.target.name]: false });
    setGeneralError("");
  };

  const validateFields = () => {
    const newErrors = {
      email: user.email === "",
      password: user.password === "",
    };
    setErrors(newErrors);
    return !newErrors.email && !newErrors.password;
  };
```

```
const submitHandler = async () => {
  if (validateFields()) {
    try {
      const login = await axios.get(
        `${api_url}/user/get/${user.email}/${user.password}`
      );
      const userData = {
        username: login.data.username,
        email: login.data.email,
        place: login.data.place,
        age: login.data.age,
        password: login.data.password,
        role: login.data.role,
        _id: login.data._id,
      };
      setData(userData);
      localStorage.setItem("userData", JSON.stringify(userData));
      navigate("/dashboard", { state: login.data });
    } catch (error) {
      console.error(error);
      setGeneralError("Invalid Email or Password");
    }
  }
};

return (
  <div>
    <Box
      sx={{
        display: "flex",
```

```
    justifyContent: "center",
    alignItems: "center",
    height: "97vh",
  }}
>
  <Box sx={styles.box_style}>
    <img
      src="/logo1.png"
      alt="Login Icon"
      style={{
        width: "180px",
        marginBottom: "-1rem",
        marginTop: "-2.5rem",
      }}
    />
    <Typography
      fontFamily={"fantasy"}
      variant="h3"
      color="white"
      style={{
        marginBottom: "0px",
      }}
      gutterBottom
    >
      LOG-IN
    </Typography>
    <TextField
      required
      fullWidth
      name="email"
      label="Email"
```

```
  variant="outlined"
  margin="normal"
  value={user.email}
  onChange={inputHandler}
  error={errors.email}
  helperText={errors.email ? "Email is required" : generalError}
  FormHelperTextProps={{
    sx: { color: errors.email ? "red" : "red" },
  }}
  InputLabelProps={{ style: { color: "white" } }}
  InputProps={styles.textfield}
 />
 <TextField
  required
  fullWidth
  name="password"
  type="password"
  label="Password"
  variant="outlined"
  margin="normal"
  value={user.password}
  onChange={inputHandler}
  error={errors.password}
  helperText={errors.password ? "Password is required" : ""}
  InputLabelProps={{ style: { color: "white" } }}
  InputProps={styles.textfield}
 />
 <Button
  variant="contained"
  sx={{
    mt: 2,
```

```
            backgroundColor: "orange",
            "&:hover": { backgroundColor: "orange" },
           }}
           onClick={submitHandler}
          >
           Log-in
          </Button>
          <Box mt={2}>
           <Typography style={{ color: "darkgray" }}>
            Don't have an Account? 
            <Link style={styles.link_style} to={"/signin"}>
             SignUp
            </Link>
           </Typography>
          </Box>
        </Box>
      </Box>
    </div>
  );
};

export default Login;
```

6.3.3 MYCART.JSX

```
import React, { useEffect, useState } from "react";
import { Link, useLocation, useNavigate } from "react-router-dom";
import axios from "axios";
import {
  Box,
```

```
  Button,
  Container,
  Grid,
  Input,
  List,
  ListItem,
  Paper,
  Rating,
  Table,
  TableBody,
  TableCell,
  TableContainer,
  TableHead,
  TableRow,
  Typography,
} from "@mui/material";
import Navbar from "./Navbar";

const MyCart = () => {
  const navigate = useNavigate();
  const [loading, setLoading] = useState(true);
  const [products, setProducts] = useState([]);
  const [empty, setEmpty] = useState(true);
  const api_url = import.meta.env.VITE_API_URL;
  const data = JSON.parse(localStorage.getItem("userData"));
  const [checkout, setCheckout] = useState(true);
  var Price = 0;
  var Index = 1;

  useEffect(() => {
    const apiUrl = `${api_url}/user/getcart/${data._id}`;
```

```
  axios
    .get(apiUrl)
    .then((response) => {
      setProducts(response.data);
      console.log(response.data);
      setEmpty(response.data.length === 0);
    })
    .catch((error) => {
      console.error("Error fetching data:", error);
    })
    .finally(() => {
      setLoading(false);
    });
  // }
}, []);

useEffect(() => {
  products.forEach((product) => {
    if (product.product.stock < product.quantity) {
      setCheckout(false);
    }
  });
}, [products]);

return (
  <div>
    <Navbar />
    {loading ? (
      <center>
        <br />
        <br />
```

```
    <br />
    <br />
    loading...
  </center>
) : empty ? (
  <center>
    <Typography style={{ fontSize: 17, marginTop: "50vh" }}>
      Your cart is empty
    </Typography>
  </center>
) : (
  <Box
    sx={{
      display: "flex",
      alignItems: "flex-start",
      marginTop: "9vh",
      width: "90%",
      justifyContent: "space-between",
    }}
  >
    <TableContainer>
      <Table>
        <TableHead>
          <TableRow>
            <TableCell
              sx={{
                fontFamily: "fantasy",
                color: "white",
                fontWeight: "bold",
                fontSize: "3vh",
                width: "600px",
```

```
  }}
>
  ITEM
</TableCell>
<TableCell
  sx={{
    fontFamily: "fantasy",
    color: "white",
    fontWeight: "bold",
    fontSize: "3vh",
  }}
>
  PRICE
</TableCell>
<TableCell
  sx={{
    fontFamily: "fantasy",
    color: "white",
    fontWeight: "bold",
    fontSize: "3vh",
  }}
>
  QUANTITY
</TableCell>
<TableCell
  sx={{
    fontFamily: "fantasy",
    color: "white",
    fontWeight: "bold",
    fontSize: "3vh",
  }}
```

```
    >
      TOTAL
    </TableCell>
  </TableRow>
</TableHead>
<TableBody>
  {products.map((product, index) => (
    <TableRow key={index}>
      <TableCell
        sx={{
          fontFamily: "cursive",
          color: "white",
          display: "flex",
          alignItems: "flex-start",
        }}
      >
        <Box>
          <img
            src={`${api_url}/${product.product.image}`}
            alt={product.product.name}
            style={{
              width: 150,
              height: "auto",
              cursor: "pointer",
            }}
            onClick={() => {
              navigate("/detproduct", { state: product.product });
            }}
          />
        </Box>
        <Box sx={{ mt: 11.8, ml: 6 }}>
```

```
<Typography
  sx={{
    fontFamily: "fantasy",
    color: "white",
    fontSize: 32,
    mt: -10,
  }}
>
  {product.product.name}
</Typography>
<Box sx={{ display: "flex", alignItems: "center" }}>
  <Box
    sx={{
      mt: 1,
      display: "flex",
      alignItems: "center",
      backgroundColor: "#222",
      color: "white",
      borderRadius: "8px",
      padding: "4px 8px",
      fontSize: "14px",
      fontWeight: "bold",
      width: "fit-content",
    }}
  >
    <Typography sx={{ mr: 0.5, color: "#FFAD18" }}>
      {parseFloat(product.product.rating.toFixed(1)) ||
        0}{" "}
      ⭐
    </Typography>
  </Box>
```

```
<Typography
  sx={{
    color: "white",
    fontFamily: "cursive",
    ml: 1,
    mt: 0.8,
  }}
>
  {product.product.reviews.length === 0
    ? "No Rating"
    : product.product.reviews.length === 1
    ? "1 Rating"
    : `${product.product.reviews.length} Ratings`}
</Typography>
</Box>
<Button
 onClick={async () => {
   try {
     await axios.delete(
       `${api_url}/user/cart/delitem/${data._id}/${product.product._id}`
     );
     window.location.reload(true);
   } catch (error) {
     console.error(error);
   }
 }}
 variant=""
 sx={{
   ml: -2,
   mt: 1,
   color: "transparent",
```

```
        "&:hover .remove_cart": {
          color: "red",
        },
      }}
    >
      <Typography
        className="remove_cart"
        color="white"
        sx={{
          fontFamily: "fantasy",
          textTransform: "none",
        }}
      >
        Remove
      </Typography>
    </Button>
  </Box>
</TableCell>
<TableCell>
  <Typography
    sx={{
      mt: 1.5,
      fontFamily: "cursive",
      color: "yellow",
    }}
  >
    ₹{product.product.price}
  </Typography>
</TableCell>
<TableCell>
  <Box
```

```
  sx={{
    display: "flex",
    alignItems: "center",
    border: "2px solid white",
    borderRadius: "8px",
    width: "100px",
    height: "25px",
    overflow: "hidden",
  }}
>
  <Button
    sx={{
      minWidth: "30px",
      fontSize: "20px",
      borderRight: "2px solid white",
      borderRadius: "0",
      color: "white",
    }}
    onClick={async () => {
      if (product.quantity === 1) {
        try {
          await axios.delete(
            `${api_url}/user/cart/delitem/${data._id}/${product.product._id}`
          );
          window.location.reload(true);
        } catch (error) {
          console.error(error);
        }
      } else {
        try {
          setProducts((prevProducts) =>
```

```
                prevProducts.map((p) =>
                  p.product._id === product.product._id
                    ? { ...p, quantity: p.quantity - 1 }
                    : p
                )
              );
              product.quantity -= 1;
              await axios.post(
`${api_url}/user/cart/updateitemquantity/${data._id}/${product.product._id}/${product.quantity}`
              );
              if (checkout === false) {
                window.location.reload(true);
              }
            } catch (error) {
              console.error(error);
            }
          }
        }}
      >
        -
      </Button>

      <Typography
        type="text"
        sx={{
          width: "40px",
          textAlign: "center",
          fontSize: "16px",
          background: "transparent",
          color: "white",
```

```
  }}
>
  {product.quantity}
</Typography>

<Button
  sx={{
    minWidth: "30px",
    fontSize: "20px",
    borderLeft: "2px solid white",
    borderRadius: "0",
    color: "white",
  }}
  onClick={async () => {
    setProducts((prevProducts) =>
      prevProducts.map((p) =>
        p.product._id === product.product._id
          ? p.quantity >= p.product.stock
            ? { ...p, available: "Not available" }
            : {
                ...p,
                quantity: p.quantity + 1,
                available: "",
              }
          : p
      )
    );

    if (product.quantity < product.product.stock) {
      try {
        await axios.post(
```

```
            `${api_url}/user/cart/updateitemquantity/${
              data._id
            }/${product.product._id}/${
              product.quantity + 1
            }`
          );
        } catch (error) {
          console.error(error);
        }
      }
    }}
  >
    +
  </Button>
</Box>
<Typography
  sx={{
    color: "red",
    minHeight: "25px",
    mb: -4.5,
    mt: 0.5,
    ml: 0.5,
  }}
>
  {product.available}
</Typography>
{product.product.stock < product.quantity && (
  <Typography
    sx={{
      color: "red",
      minHeight: "25px",
```

```
                mb: -4.5,
                mt: 1,
                ml: 0.5,
              }}
            >
              Not available
            </Typography>
          )}
        </TableCell>
        <TableCell>
          <Typography
            sx={{
              mt: 1.5,
              fontFamily: "cursive",
              color: "yellow",
            }}
          >
            ₹{`${product.product.price * product.quantity}`}
          </Typography>
        </TableCell>
      </TableRow>
    ))}
  </TableBody>
 </Table>
</TableContainer>
<Box
 sx={{
   width: "18%",
   height: "89vh",
   mt: "1vh",
   position: "sticky",
```

```
    top: "10vh",
    borderLeft: "1px solid white",
   }}
  >
   <Box
    sx={{
     borderBottom: "1px solid white",
     width: "25.65vw",
    }}
   >
    <Typography
     sx={{
      ml: 3,
      mb: 1.4,
      mt: 0.25,
      fontFamily: "fantasy",
      color: "white",
      fontWeight: "bold",
      fontSize: "3vh",
      width: "300px",
     }}
    >
     SUMMARY
    </Typography>
   </Box>

   {products.forEach((product, index) => {
    Price += product.product.price * product.quantity;
    Index += index;
    console.log(product);
   })}
```

```
<Box
 sx={{
  minWidth: 350,
  mt: 2,
  ml: 3,
 }}
>
 <Typography
  sx={{ fontFamily: "cursive", fontWeight: "bold", fontSize: 23 }}
 >
  Total Items :<span style={{ color: "orange" }}> {Index}</span>
  <br />
  <br />
  Delivery : Free
  <br />
  <br />
  Payment : Cash On Delivery
  <br />
  <br />
  Tax Payable :<span style={{ color: "orange" }}> ₹0</span>
  <br />
  <br />
  <br />
   Estimated Total :
  <span style={{ color: "darkorange" }}> ₹{Price}</span>
  <br />
  {checkout ? (
   <Button
    variant="contained"
    sx={{
     mt: 3,
```

```
      ml: 1,
      fontSize: "20px",
      padding: "15px 30px",
      minWidth: "350px",
      backgroundColor: "orangered",
    }}
    onClick={async () => {
      try {
        for (const product of products) {
          await axios.post(
            `${api_url}/product/buy/${data._id}/${product.product._id}/${product.quantity}`
          );
        }

        for (const product of products) {
          await axios.delete(
            `${api_url}/user/cart/delitem/${data._id}/${product.product._id}`
          );
        }

        window.location.reload(true);
        console.log("Order confirmed");
      } catch (error) {
        console.error("Error ordering product:", error);
      }
    }}
  >
    CheckOut
  </Button>
) : (
  <Button
```

```
                variant="contained"
                sx={{
                  mt: 3,
                  ml: 1,
                  fontSize: "20px",
                  padding: "15px 30px",
                  minWidth: "350px",
                  backgroundColor: "gray",
                  cursor: "not-allowed",
                }}
              >
                CheckOut
              </Button>
            )}
          </Typography>
        </Box>
      </Box>
    </Box>
  )}
 </div>
 );
};

export default MyCart;
```

6.3.4 CONNECTION.JS

```
var mongoose = require("mongoose");
```

```
mongoose.connect("mongodb+srv://test:test@cluster0.cguda.mongodb.net/new1?retryWrites=true&w=majority&appName=Cluster0").then(()=>{
    console.log("Connected!");
}).catch((error) => {
    console.log(error)
})
```

6.3.5 INDEX.JS

```
var express = require("express");
var cors = require("cors");
var path = require('path');
var multer = require("multer");
var crypto = require('crypto')
var app = express();
var fs = require("fs");
require("./connection.js");

const URL = "http://localhost";
const PORT = 3000;

var CryptoJS = require('crypto-js');
var userModel = require("./models/user");
var productModel = require("./models/product");
var orderModel = require("./models/order")
var miscModel = require("./models/misc")

const storage = multer.diskStorage({
    destination: (req, file, cb) => {
        cb(null, 'images/products');
    },
```

```
    filename: (req, file, cb) => {
        cb(null, crypto.randomBytes(8).toString('hex').slice(0, 8) + path.extname(file.originalname));
    },
});
const upload = multer({storage: storage});

app.use(express.json());
app.use(cors());

app.get("/user/viewall", async (req, res) => {
    try {
        var data = await userModel.find();
        res.send(data)

    } catch (error) {
        console.log(error);
    }
});

app.post("/user/add", async (req, res) => {
    try {
        await userModel(req.body).save();
        res.send({message: "Data Added"});
    } catch (error) {
        console.log(error);
    }
});

app.get("/user/get/:email/:password", async (req, res) => {
    try {
        var email = req.params.email;
```

```
        var password = CryptoJS.SHA256(req.params.password).toString(CryptoJS.enc.Hex)
        var user = await userModel.findOne({email: email, password: password});
        if (user) {
            res.send(user);
        } else {
            res.status(404);
            res.send({message: "Invalid Email or Password"});
        }

    } catch (error) {
        console.log(error);
    }
});

app.post("/user/register/", async (req, res) => {
    try {
        var user = req.body;
        user.admin = false;
        user.merchant = false;
        user.cart = [];
        user.wishlist = [];
        user.password = CryptoJS.SHA256(user.password).toString(CryptoJS.enc.Hex)
        var existing_user = await userModel.findOne({email: user.email});
        if (!existing_user) {

            await userModel(user).save();
            res.send({message: "Account Registered"});

        } else {
            res.status(409);
            res.send({message: "Email Already Exists"});
```

```
        }

    } catch (error) {
        console.log(error);
    }
});

app.put("/user/edit/", async (req, res) => {
    try {
        var id = req.body._id;
        var user = req.body;
        var existing_user = await userModel.findOne({email: user.email});
        if (!existing_user) {
            await userModel.findByIdAndUpdate(id, user);
            res.send({message: "Profile Updated"})
        } else if (existing_user._id == user._id) {
            await userModel.findByIdAndUpdate(id, user);
            res.send({message: "Profile Updated"})
        } else {
            res.status(409);
            res.send({message: "Email Already Exists."})
        }

    } catch (error) {
        console.log(error);
    }
})

app.delete("/user/delete/", async (req, res) => {
    try {
        var id = req.body._id;
```

```
        var del = await userModel.findByIdAndDelete(id);
        if (del != null) {
            var delproducts = await productModel.find({owner: id})
            await productModel.deleteMany({owner: id});
            for (let i = 0; i < delproducts.length; i++) {
                fs.unlink(delproducts[i].image, () => {
                });
            }
            res.send({message: "Account Deleted"});
        } else {
            res.status(404);
            res.send({message: "Failed To Delete Account"});
        }

    } catch (error) {
        res.status(404);
        res.send({message: "Failed To Delete Account"});
    }
})

app.post("/product/add/", upload.single('file'), async (req, res) => {
    try {
        var product = req.body;
        product.reviews = [];
        product.rating = 0;
        product.featured = false;
        req.body.image = ""
        req.body.keywords = req.body.keywords.split(",")
        product = await productModel(product).save();
        var img_path = `${req.file.destination}/${product._id}${path.extname(req.file.filename)}`;
        fs.rename(req.file.path, img_path, () => {
```

```
        })
        product.image = `${img_path}`;
        product.save();
        res.send({message: "Product Added"})

    } catch (error) {
        console.log(error);
    }
});

app.get("/product/viewall", async (req, res) => {
    try {
        var data = await productModel.find();
        res.send(data)

    } catch (error) {
        console.log(error);
    }
});

app.get("/product/view/:pid", async (req, res) => {
    try {
        var id = req.params.pid
        var data = await productModel.findOne({_id: id});
        res.send(data)

    } catch (error) {
        console.log(error);
    }
});
```

```
app.get("/merchant/products/:id", async (req, res) => {
    try {
        var id = req.params.id;
        var data = await productModel.find({merchant_id: id});
        res.send(data)

    } catch (error) {
        console.log(error);
    }
});

app.delete("/product/delete/:id", async (req, res) => {
    try {
        var id = req.params.id;
        var del = await productModel.findByIdAndDelete(id);
        if (del != null) {
            fs.unlink(del.image, () => {
            });
            await userModel.updateMany(
                { "cart.product": id },
                { $pull: { cart: { product: id } } }
            );
            await userModel.updateMany(
                { "wishlist.product": id },
                { $pull: { wishlist: { product: id } } }
            );
            res.send({message: "Product Deleted"});
        } else {
            res.status(404);
            res.send({message: "Failed To Delete Product"});
        }
```

```
    } catch (error) {
        res.status(404);
        res.send({message: "Failed To Delete Product"});
    }
})
app.get("/product/search/:word?", async (req, res) => {
    try {
        var word="";
        word = req.params.word;
        var query = word && word.trim() !== "" ? { name: new RegExp(".*" + word + ".*", "i") } : {};
        var data = await productModel.find(query);

        res.send(data)

    } catch (error) {
        console.log(error);
    }
});

app.post("/product/addreview/:productId", async (req, res) => {
    try {
        var id = req.params.productId;
        var review = req.body;
        var product = await productModel.findById(id);
        product.reviews.unshift(review)
        let total = 0
        for (let i=0;i<product.reviews.length;i++){
            total += product.reviews[i].rating;
        }
        product.rating = total / product.reviews.length;
```

```
        await product.save();
        res.send({ message: "Review Added" })

    } catch (error) {
        console.log(error);
    }
});

app.delete("/product/delreview/:productId/:userId", async (req, res) => {
    try {
        var recid = req.params.productId;
        var userid = req.params.userId;
        var product = await productModel.findById(recid);
        product.reviews = product.reviews.filter(review => review.userId != userid)
        let total = 0
        for (let i=0;i<product.reviews.length;i++){
            total += product.reviews[i].rating;
        }
        product.rating = total / product.reviews.length;
        await product.save();
        res.send({ message: "Review Deleted" })

    } catch (error) {
        console.log(error);
    }
});

app.get("/product/getreviews/:productId", async (req, res) => {
    try {
        var id = req.params.productId;
        var product = await productModel.findById(id);
```

```
        res.send(product.reviews);

    } catch (error) {
        console.log(error);
    }
});

app.post("/product/addtocart/:userId/:productId", async (req, res) => {
    try {
        var userId = req.params.userId;
        var productId = req.params.productId;
        var user = await userModel.findById(userId);
        var cartItem = user.cart.find(item => item.product == productId);
        if (cartItem == undefined){
            user.cart.unshift({"product": productId, "quantity": 1});
        }
        else{
            cartItem.quantity += 1
        }

        await user.save();
        res.send({ message: "Product Added To Cart" })

    } catch (error) {
        console.log(error);
    }
});
app.get("/user/getcart/:userId", async (req, res) => {
    try {
        var userId = req.params.userId;
        var user = await userModel.findById(userId);
```

```
        await user.populate("cart.product")

        res.send(user.cart)

    } catch (error) {
        console.log(error);
    }
});

app.delete("/user/cart/delitem/:userId/:productId", async (req, res) => {
    try {
        var userId = req.params.userId;
        var productId = req.params.productId;
        var user = await userModel.findById(userId);
        user.cart = user.cart.filter(entry => entry.product != productId);
        await user.save();
        res.send({ message: "Product Deleted" })

    } catch (error) {
        console.log(error);
    }
});

app.post("/user/cart/updateitemquantity/:userId/:productId/:quantity", async (req, res) => {
    try {
        var userId = req.params.userId;
        var productId = req.params.productId;
        var quantity = parseInt(req.params.quantity);
        var user = await userModel.findById(userId);
        var item = user.cart.find(entry => entry.product == productId);
        item.quantity = quantity;
```

```
        await user.save();
        res.send({ message: "Quantity Updated" })

    } catch (error) {
        console.log(error);
    }
});

app.get("/orders/viewall", async (req, res) => {
    try {
        var data = await orderModel.find();
        res.send(data)

    } catch (error) {
        console.log(error);
    }
});

app.post("/product/buy/:userId/:productId/:quantity", async (req, res) => {
    try {
        var userId = req.params.userId;
        var productId = req.params.productId;
        var productDetails = await productModel.findById(productId)
        var quantity = parseInt(req.params.quantity);
        var user = await userModel.findById(userId);
        var order = {
            "name": productDetails.name,
            "userId": userId,
            "amount": productDetails.price * quantity,
            "quantity": quantity,
            "status": "Processing",
```

```
            "image": productDetails.image,
            "product": productId
        }
        if (productDetails.stock < quantity){
            res.send("Requested Quantity Not In Stock")
        }
        else{
            productDetails.stock -= quantity
            await productDetails.save()
            order = await orderModel(order).save()
            res.send({ message: "Product Order Placed" })
        }

    } catch (error) {
        console.log(error);
    }
});

app.post("/user/cartcheckout/:userId", async (req, res) => {
    try {
        var userId = req.params.userId;
        var productId = req.params.productId;
        var productDetails = await productModel.findById(productId)
        var quantity = parseInt(req.params.quantity);
        var user = await userModel.findById(userId);
        var order = {
            "name": productDetails.name,
            "userId": userId,
            "amount": productDetails.price * quantity,
            "quantity": quantity,
            "status": "Processing",
```

```
            "image": productDetails.image,
            "product": productId
        }
        if (productDetails.stock < quantity){
            res.send("Requested Quantity Not In Stock")
        }
        else{
            productDetails.stock -= quantity
            await productDetails.save()
            order = await orderModel(order).save()
            res.send({ message: "Product Order Placed" })
        }

    } catch (error) {
        console.log(error);
    }
});

app.get("/user/orders/:userId", async (req, res) => {
    try {
        var userId = req.params.userId
        var data = await orderModel.find({userId: userId})
        res.send(data)

    } catch (error) {
        console.log(error);
    }
});

app.post("/product/makefeatured/:productId", async (req, res) => {
    try {
```

```
        var productId = req.params.productId;
        var product = await productModel.findById(productId);
        product.featured = true;
        product.save();

        res.send({ message: "Added Product to Featured" })

    } catch (error) {
        console.log(error);
    }
});

app.post("/product/removefeatured/:productId", async (req, res) => {
    try {
        var productId = req.params.productId;
        var product = await productModel.findById(productId);
        product.featured = false;
        product.save();

        res.send({ message: "Removed Product From Featured" })

    } catch (error) {
        console.log(error);
    }
});

app.get("/product/getfeatured", async (req, res) => {
    try {
        var data = await productModel.find({ featured: true });
        res.send(data)
```

```
    } catch (error) {
        console.log(error);
    }
});

app.delete("/user/wishlist/delitem/:userId/:productId", async (req, res) => {
    try {
        var userId = req.params.userId;
        var productId = req.params.productId;
        var user = await userModel.findById(userId);
        user.wishlist = user.wishlist.filter(entry => entry.product != productId);
        await user.save();
        res.send({ message: "Product Removed From Wishlist" })

    } catch (error) {
        console.log(error);
    }
});

app.post("/misc/categories/add/:name", async (req, res) => {
    try {
        var name = req.params.name;
        var misc = await miscModel.findOne();
        if (misc.categories.filter(item => item === name).length == 0){
            misc.categories.push(name);
            await misc.save()
            res.send({message: "Category Added"});
        }
        else {
            res.send({message:"Category Already Exists"})
        }
```

```
    } catch (error) {
        console.log(error);
    }
});

app.delete("/misc/categories/delete/:name", async (req, res) => {
    try {
        var name = req.params.name;
        var misc = await miscModel.findOne();
        misc.categories = misc.categories.filter(item => item !== name)
        await misc.save()
        res.send({message: "Category Deleted"})
    } catch (error) {
        console.log(error);
    }
});

app.get("/misc/categories/get", async (req, res) => {
    try {
        res.send((await miscModel.findOne()).categories)
    } catch (error) {
        console.log(error);
    }
});

app.get("/misc", async (req, res) => {
    try {
        res.send(await miscModel.findOne())
    } catch (error) {
        console.log(error);
    }
```

```
});

app.use('/images/products', express.static(path.join(__dirname, 'images/products')));

app.listen(PORT, () => {
    console.log("Port is Up");
});
```

CHAPTER :7

SYSTEM TESTING

7.1 TEST CASE

Table 7.1.1 Test Case for SwiftBuy

Test Case	Module	Objective	Expected Result
TC01	Product Browsing	Verify that users can view products across all categories	Product listings load correctly with images and details
TC02	Filtering	Ensure that category and filter options work as intended	Products are filtered based on selected criteria
TC03	Wishlist	Check if users can add and manage items in their wishlist	Items are added/removed from wishlist and persist across sessions
TC04	Customer Reviews	Verify that users can view and submit product reviews	Reviews are displayed and new ones are submitted successfully
TC05	Search Functionality	Ensure the search bar returns relevant product results	Accurate product results are shown based on keywords
TC06	Checkout Process	Validate the cash on delivery checkout flow	Order is placed successfully with COD option selected
TC07	User Interface	Confirm that the UI is clean and intuitive for users	Users can navigate the site easily without confusion
TC08	Accessibility	Check that the platform is accessible to all users including those with disabilities	All accessibility features function correctly (e.g., keyboard navigation, screen readers)
TC09	Security	Ensure secure handling of user data and sessions	User data is protected and sessions are securely managed
TC10	Full Purchase Flow	Test end-to-end shopping from browsing to checkout	User can complete a full purchase with no errors

7.2 TEST PLAN

Table 7.2.1 Test Plan for SwiftBuy

Test Case ID	Test Description	Input	Expected Output	Actual Result
TC01	Browse product categories	User selects a category	Product list appears	Products displayed
TC02	Filter products by criteria	User applies filters (e.g., price, brand)	Filtered products shown	Filter works correctly
TC03	Add item to wishlist	User clicks 'Add to Wishlist'	Item saved to wishlist	Item added successfully

TC04	View/submit customer reviews	User visits product page or adds review	Reviews visible or new review saved	Reviews shown/submitted
TC05	Search for products	User enters keyword in search bar	Relevant products displayed	Correct results shown
TC06	Checkout using cash on delivery	User places order with COD option	Order confirmation	Order placed with COD
TC07	Access user interface	User loads SwiftBuy in browser	Homepage loads properly	UI loads and functions
TC08	Test platform accessibility	Navigation via keyboard/screen reader	Accessible controls and labels	Accessibility confirmed
TC09	Verify data security	User logs in and browses site	Session secured, data protected	No data/security issues
TC10	Full purchase workflow	User browses, selects, and orders	End-to-end flow completes	Purchase successful

7.3 TRACEABILITY MATRIX

Table 7.3.1 Traceability matrix for SwiftBuy

Requirement ID	Requirement Description	Linked Test Case ID(s)
R01	Product browsing functionality	TC01
R02	Filtering capability across products	TC02
R03	Wishlist management features	TC03
R04	Customer review system	TC04
R05	Search functionality	TC05
R06	Checkout process with COD	TC06
R07	User interface usability	TC07
R08	Accessibility features	TC08
R09	Data and session security	TC09
R10	Complete purchase workflow	TC10

CHAPTER :8

RESULT

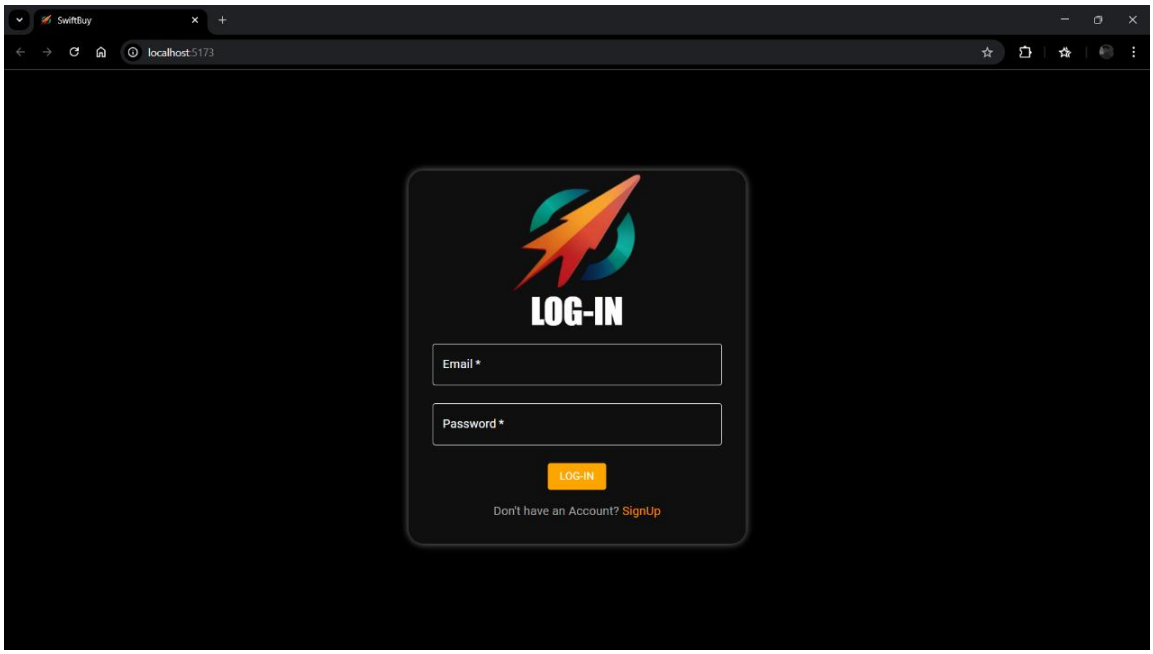

Figure 8.1 : Login Page

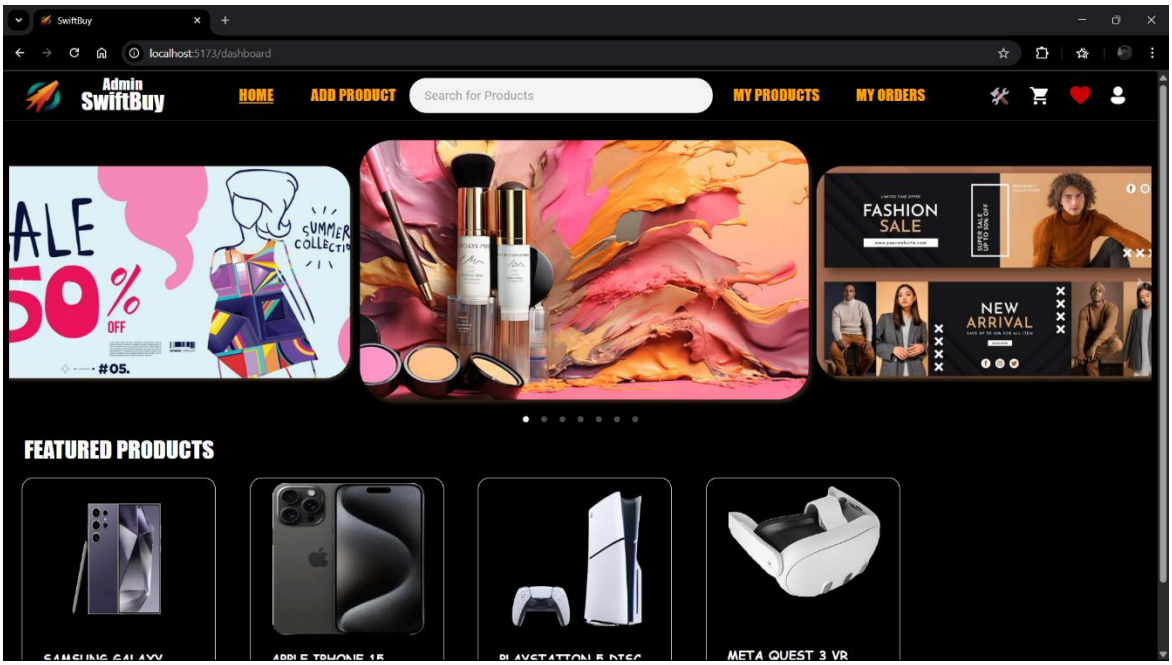

Figure 8.2 : Home Page

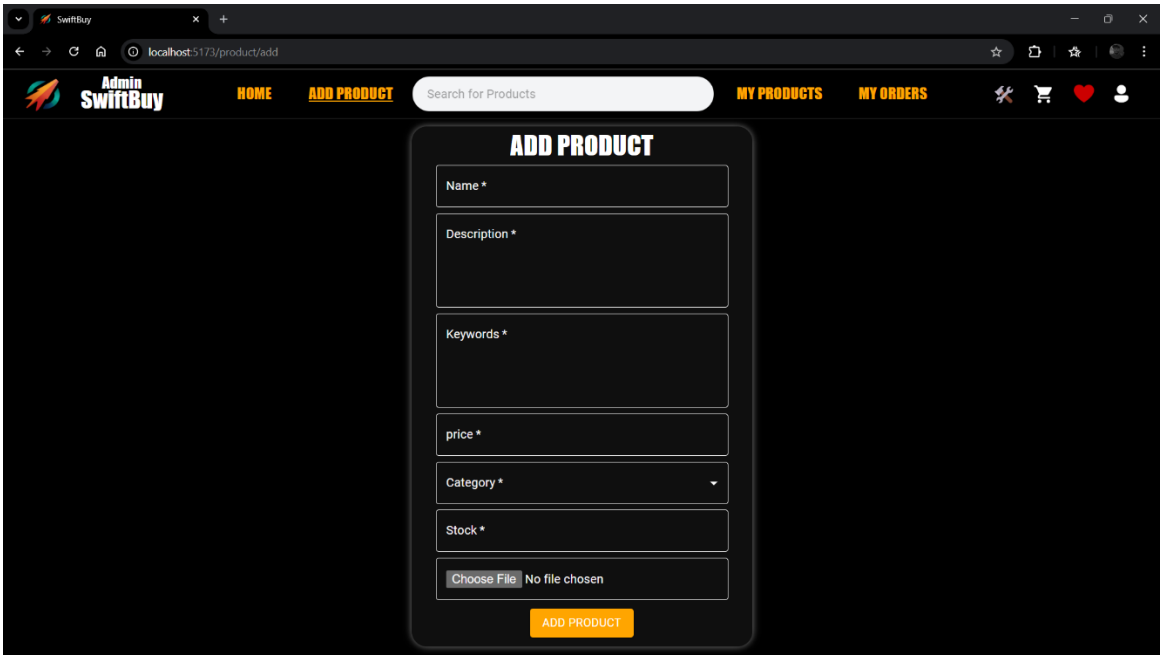

Figure 8.3 : Add Product Page

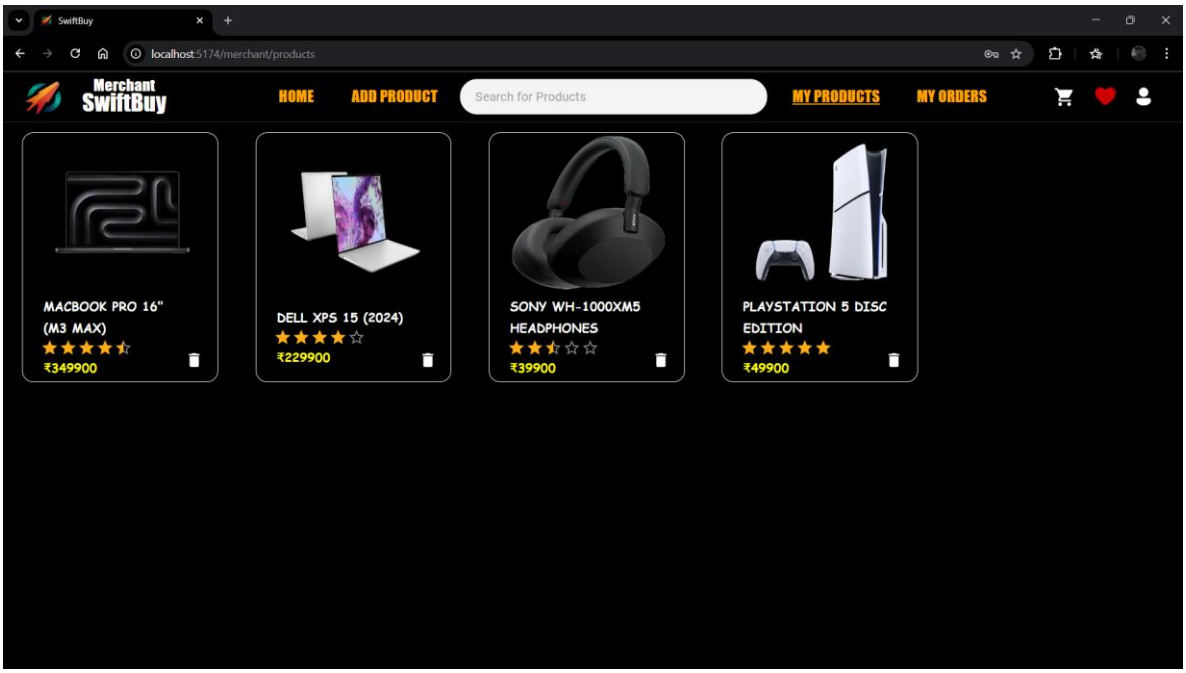

Figure 8.4 : My Products Page

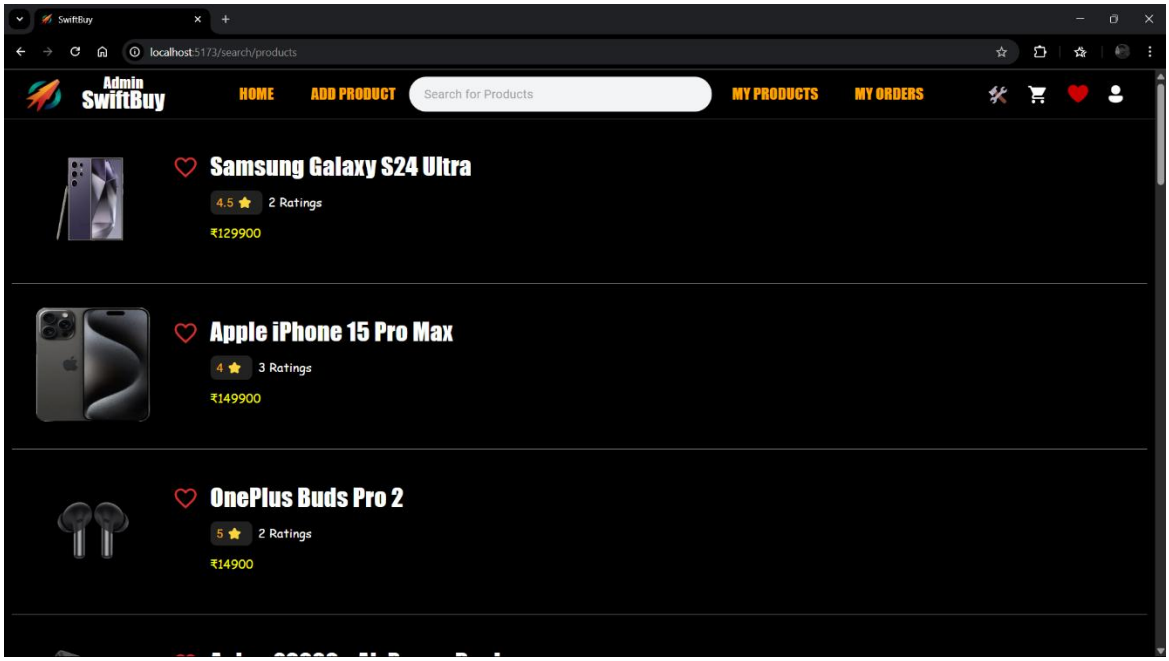

Figure 8.5 : Search Products

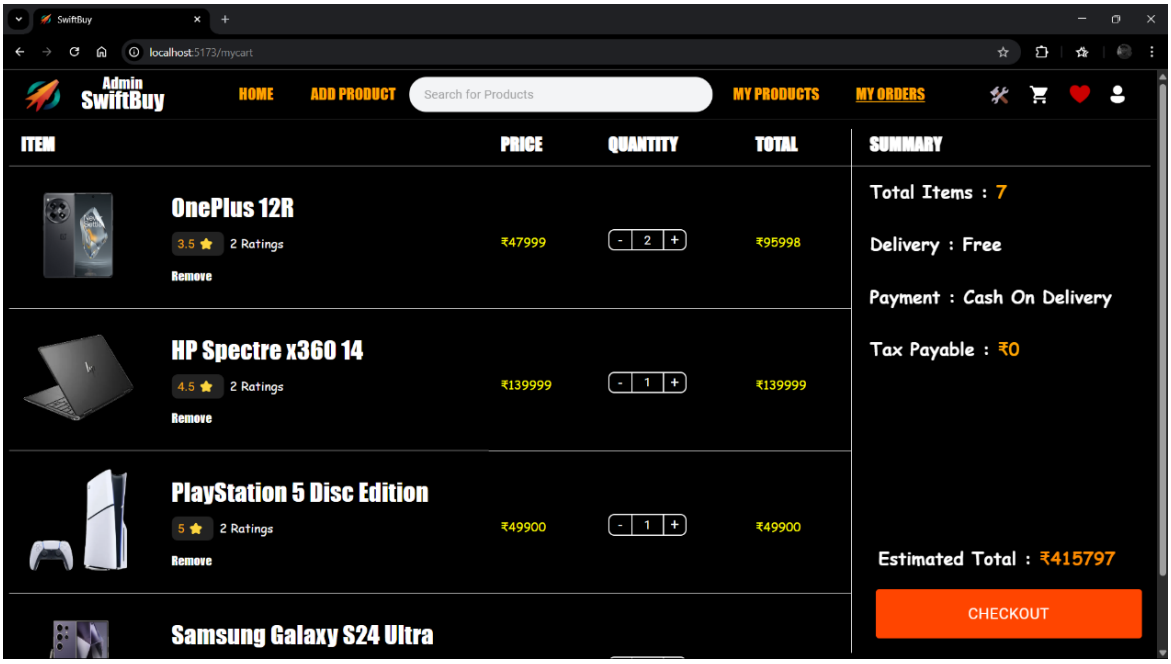

Figure 8.6 : My Cart Page

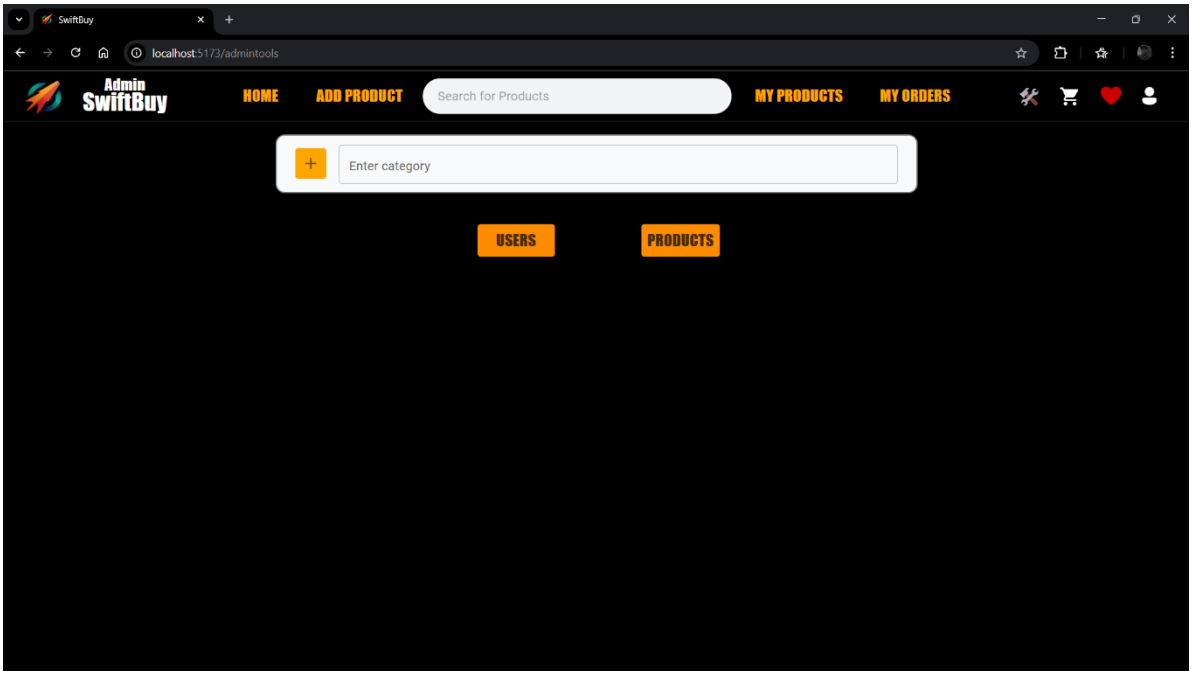

Figure 8.7 Admins Tools Page

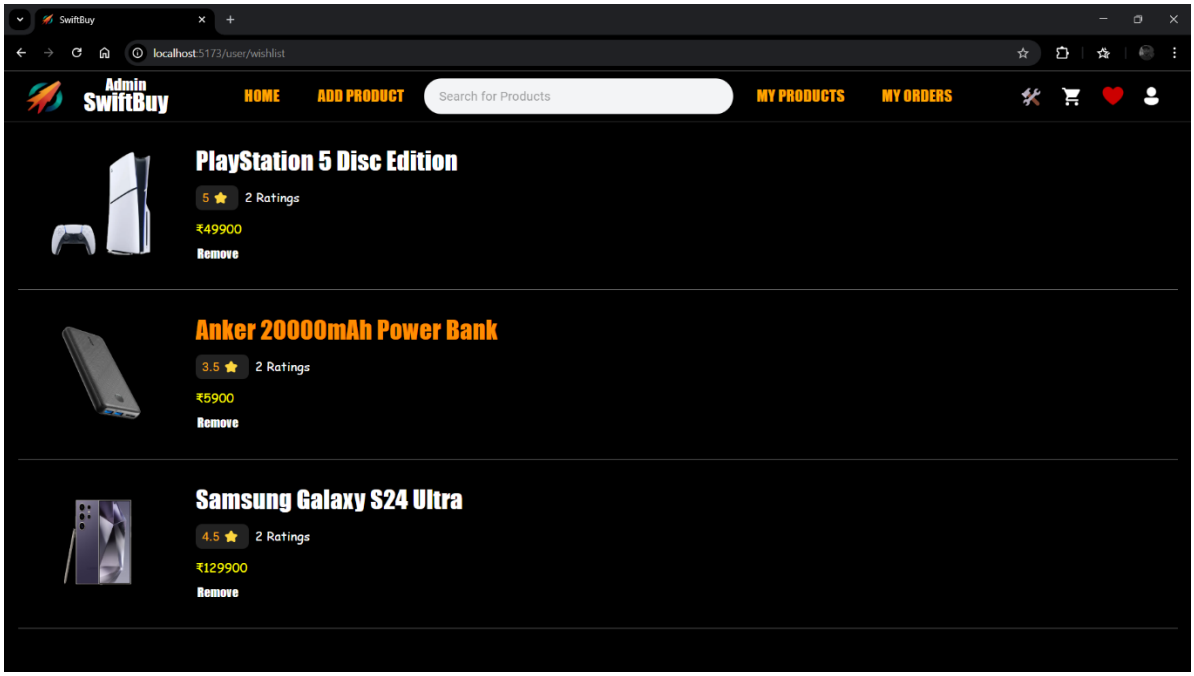

Figure 8.8 Wishlist Page

CHAPTER :9

CONCLUSION AND FUTURE SCOPE

9.1 Conclusion

SwiftBuy is a streamlined e-commerce platform created to deliver a smooth, accessible, and efficient shopping experience for users of all backgrounds. In an era where online retail is constantly evolving, SwiftBuy focuses on the core elements that truly matter to customers: easy product browsing, responsive search filtering, wishlists for future purchases, and real-time order tracking to ensure transparency from checkout to delivery. By simplifying these essential components, SwiftBuy eliminates the clutter and complexity often found in larger, bloated platforms.

While many e-commerce systems attempt to impress with excessive features, SwiftBuy emphasizes clarity, speed, and usability. It proves that effective design doesn't rely on overwhelming interfaces or flashy tools—instead, success lies in giving users what they need in a direct, dependable way. Whether a user is shopping for electronics, fashion, or groceries, SwiftBuy ensures a consistent and enjoyable experience with minimal barriers.

The development of SwiftBuy also demonstrates the power of focused software design. Even without advanced features like AI-based recommendations or diverse payment methods, the platform delivers real value through thoughtful structure and clean functionality. As a result, SwiftBuy stands as an example of how e-commerce can be both powerful and user-friendly, offering a reliable, satisfying journey from product search to purchase.

9.2 Future Scope

While SwiftBuy already offers a smooth and reliable online shopping experience, there are many opportunities to expand and improve the platform in the future. Some potential enhancements include:

- **AI Driven Recommendations:** Integrating machine learning to offer personalized product suggestions based on user preferences, browsing history, and trends.
- **Advanced Filtering Options:** Enabling more detailed and dynamic filters to help users find products faster and with greater precision.

- **Multiple Payment Integration:** Expanding beyond cash on delivery to include digital wallets, credit/debit card options, and UPI-based payments for convenience.
- **Personalized Notifications:** Sending tailored alerts for discounts, order updates, or restocks based on individual shopping behavior.
- **Mobile App Development:** Creating a dedicated mobile application to offer users a faster, on-the-go shopping experience with push notifications and offline capabilities.
- **Enhanced User Profiles**: Introducing features like shopping history, saved preferences, and account analytics to improve user engagement.

SwiftBuy lays a strong foundation for a functional and user-friendly e-commerce solution. As the platform continues to develop, these future features can greatly enhance both user experience and system efficiency, making SwiftBuy a comprehensive and competitive player in the digital marketplace.

REFERENCES

[1] "Online Product Decision Support Using Sentiment Analysis and Fuzzy Cloud-based Multi-Criteria Model through Multiple E-Commerce Platforms," Vincent C., et al., 2023.

[2] "A Static Machine Learning-Based Evaluation Method for Usability and Security Analysis in E-Commerce Websites," Kumar B., Roy S., Singh K. U., Pandey S. K., and Kumar A., 2023.

[3] "Sentiment Analysis in E-Commerce Platforms: A Review of Current Techniques and Future Directions," Huang H., Zavareh A. A., and Mustafa M. B., 2023.

[4] "Exploring the Landscape of Hybrid Recommendation Systems in E-Commerce: A Systematic Literature Review," Bodduluri K. C., Palma F., Kurti A., Jusufi I., and Löwenadler H., 2024.

[5] "Exploring the Impact of Computer Applications on Cross-Border E-Commerce Performance," Jin L. and Chen L., 2024.

[6] "OntoCommerce: Incorporating Ontology and Sequential Pattern Mining for Personalized E-Commerce Recommendations," Mustafa G., et al., 2024.

[7] "End-Cloud Collaboration Framework for Advanced AI Customer Service in E-Commerce," Teng L., Liu Y., Liu J., and Song L., 2024.

[8] "E-Commerce Image Enhancement Method Based on Instance Segmentation and Background Replacement," Gao Q., Hu H., and Liu W., 2024.