

CHAPTER 1

INTRODUCTION

1.1 OVERVIEW OF THE PROJECT

The MANIYAN STORES E-commerce Platform is designed to modernize the shopping experience for departmental goods by providing a seamless and efficient online shopping solution. Developed using the MERN stack (MongoDB, Express.js, React, and Node.js), this platform ensures a dynamic user interface and a robust backend for smooth performance. Key features include Real-Time Inventory Management, Personalized Product Recommendations, Customer Reviews, and Advanced Search and Filtering options, allowing users to find products quickly. Secure authentication is implemented using Firebase and JWT, ensuring safe access and data protection. The integration of Stripe enables secure and hassle-free transactions, while MongoDB efficiently manages product listings and customer data. With responsive design, order tracking, and dedicated customer support, the platform enhances user convenience, builds trust, and supports scalability for future growth.

1.2 PROBLEM DEFINITION

In the traditional departmental stores often face challenges due to manual operations and disconnected systems, leading to inefficiencies in inventory management, customer service, and order processing. Without an integrated e-commerce platform, businesses struggle to provide a seamless shopping experience, resulting in limited customer reach, delayed order fulfillment, and difficulties in tracking product availability. Additionally, the absence of secure online payment options and real-time inventory updates restricts the ability to offer a smooth and reliable purchasing process.

The goal of this project is to develop the MANIYAN STORES E-commerce Platform using the MERN stack. This platform will address these challenges by providing a scalable, secure, and user-friendly online shopping solution. Key features include Real-Time Inventory Management, Personalized Product Recommendations, Secure Payment Integration with Stripe, and Advanced Search and Filtering. By ensuring seamless order

tracking, secure authentication using Firebase and JWT, and an intuitive user interface, the platform will enhance customer satisfaction, streamline business operations, and position MANIYAN STORES for long-term growth in the competitive e-commerce market.

1.3 OBJECTIVE OF THE PROJECT

The objective of this project is to develop the MANIYAN STORES E-commerce Platform, designed to enhance the shopping experience for departmental goods while improving operational efficiency and scalability. Built using the MERN stack (MongoDB, Express, React, Node.js), the platform ensures high performance, security, and seamless user interactions. Key features include Real-Time Inventory Management, Secure Authentication with Firebase and JWT, Personalized Product Recommendations, Customer Reviews, Advanced Search and Filtering, and Secure Payment Integration via Stripe. These features will provide users with a smooth and intuitive shopping experience, enabling them to browse, compare, and purchase products effortlessly.

Additionally, the platform will enhance customer engagement by offering easy order tracking, responsive customer support, and personalized shopping suggestions. By ensuring a user-friendly and feature-rich online presence, MANIYAN STORES will expand its customer reach, streamline business operations, and establish itself as a competitive and modern player in the e-commerce industry.

CHAPTER 2

SYSTEM ANALYSIS

2.1 EXISTING SYSTEM

The current system at MANIYAN STORES relies on traditional in-store shopping and manual processes for inventory, billing, and transactions, causing inefficiencies. Customers must visit the store physically, limiting accessibility and reach. Without an online catalog, the store struggles to adapt to modern shopping trends. Manual inventory updates lead to stock mismatches, while payments rely on cash or basic POS systems, lacking secure digital options. The absence of a customer engagement system results in no personalized recommendations or targeted promotions. Without an integrated e-commerce platform, MANIYAN STORES faces challenges in scaling operations, streamlining processes, and enhancing customer satisfaction, emphasizing the need for a digital solution.

2.1.1 DISADVANTAGES OF EXISTING SYSTEM

The existing system has following disadvantages:

- Inefficient order management, causing crowding and delays at the billing counter.
- Inconvenient for customers needing to visit the store.
- No centralized system for managing inventory, orders, or customer data.
- Limited customer reach due to the absence of an online presence.
- Higher risk of stock mismatches and sales errors due to manual record-keeping.

2.2 PROPOSED SYSTEM

The proposed MANIYAN STORES E-commerce Platform will modernize the business by enabling customers to browse products, check prices, and make purchases online, enhancing the shopping experience.

With features like real-time inventory management, secure payments, and personalized recommendations, the platform will streamline operations, expand reach, and support future growth.

With an intuitive, user-friendly interface powered by the MERN stack (MongoDB, Express, React, Node), customers will easily browse products, check prices, and make purchases online. The system will include features such as Product Management, Customer Orders, Billing, and User Authentication. Administrators will have real-time control over inventory, orders, and customer data, enabling efficient management and informed decision-making.

2.2.1 ADVANTAGES OF PROPOSED SYSTEM

The proposed system offers the following advantages:

- Streamlined product browsing and purchasing process, allowing customers to order online conveniently.
- User-friendly interface for easy navigation and browsing of products.
- Real-time inventory management and updates for administrators.
- Centralized platform for managing product listings, customer orders, and data.
- Enhanced customer reach through a scalable online presence, supporting potential growth opportunities.

2.3 FEASIBILITY STUDY

Feasibility studies are crucial for evaluating the proposed MANIYAN STORES E-commerce Platform, assessing its functionality, potential impact on the business, ability to meet customer needs, and resource efficiency. While any project may seem feasible with unlimited resources and time. The proposed system will undergo a proof-of-concept phase to determine its viability before proceeding with full development. Understanding the relationship between feasibility and risk is essential, especially when project risks are significant and the feasibility of development is uncertain.

- Technical Feasibility
- Operational Feasibility
- Economic Feasibility

2.3.1 TECHNICAL FEASIBILITY

Technical feasibility assesses whether the necessary technology and resources are available for the successful development and implementation of the MANIYAN STORES E-commerce Platform. This evaluation is critical, as it involves the analysis of both technology availability and resource requirements. A key consideration is whether the organization has sufficient resources for both development and implementation. Given that the proposed system utilizes existing technologies (such as the MERN stack) and requires minimal additional resources, it is classified as technically feasible.

2.3.2 OPERATIONAL FEASIBILITY

Operational feasibility evaluates how effectively the MANIYAN STORES E-commerce Platform meets the needs of both customers and administrators. The platform is designed to provide robust support to users while enhancing overall operational performance. If it successfully fulfills these criteria, it can be regarded as operationally feasible. The system aims to deliver a convenient, user-friendly experience accessible to customers globally, expanding its reach. Additionally, by creating better market opportunities for the business, the proposed system further reinforces its operational viability.

2.3.3 ECONOMIC FEASIBILITY

Economic feasibility examines the project's development costs in relation to the potential revenue and benefits it will generate. The proposed MANIYAN STORES E-commerce Platform is economically advantageous as it does not require additional hardware or software investments, ensuring its financial viability. By carefully analyzing costs against expected returns, the project demonstrates a strong economic rationale, ensuring that the investment will yield positive financial outcomes for the business. This thorough assessment of economic feasibility highlights the project's potential to deliver value and sustainability.

CHAPTER 3

SYSTEM SPECIFICATION

3.1 HARDWARE SPECIFICATION

- Processor : Intel® Core™ i3- 2.00GHz
- Ram : 8GB
- System type : 64-bit operating system
- Hard disk : 512GB
- Keyboard : Standard 102

3.2 SOFTWARE SPECIFICATION

- Operating System : Windows
- Front End : HTML, CSS, REACT JS
- Back End : NODEJS, Mongo DB
- Environment : Visual Studio, Chrome

3.2.1 FRONT END

3.2.1.1 HTML

Hypertext Markup Language (HTML) forms the foundation of web development and is crucial for creating the structure of a website. For the MANIYAN STORES E-commerce Platform, HTML will be used to organize the layout, including sections for product listings, customer information, and order forms. HTML defines elements like headings, paragraphs, images, and links, ensuring that the website is easy to navigate and visually appealing. Semantic HTML will be employed to ensure well-structured content that is search engine-

friendly, improving the platform's online visibility. HTML also integrates with Cascading Style Sheets (CSS) to style the site and JavaScript to add interactive features, enhancing the user experience. Additionally, HTML attributes provide extra information about elements, supporting the manipulation of content through CSS and JavaScript for a more engaging and dynamic platform.

3.2.1.2 CSS

Cascading Style Sheets (CSS) are essential for defining the visual presentation of HTML elements on the MANIYAN STORES E-commerce Platform. CSS allows developers to control design aspects such as fonts, colors, margins, borders, background images, and the overall layout. By separating content (HTML) from presentation (CSS), the code becomes cleaner and easier to maintain. For this platform, CSS ensures a visually appealing and user-friendly interface, enabling easy navigation for customers to explore products and place orders. CSS also supports responsive design, ensuring the website is accessible and functional across various devices, including desktops, tablets, and smartphones. The use of external, internal, and inline style sheets offers flexibility in managing styles, with external style sheets ensuring consistency and reducing development time by applying the same styles across multiple pages.

3.2.1.3 JAVASCRIPT

JavaScript is a client-side scripting language that enhances the interactivity and functionality of web applications. It allows developers to create dynamic content that responds to user actions, such as form validation, animations, and real-time updates. JavaScript also enables asynchronous communication with the server through AJAX, allowing data to be fetched and updated without reloading the page. For the MANIYAN STORES E-commerce Platform, JavaScript plays a vital role in creating interactive features like product search, order placement, and real-time notifications. It improves the user experience by providing instant feedback and smooth interactions. JavaScript can be embedded directly into HTML using internal scripts or written in external files for better organization and maintainability. External JavaScript files help keep the code structured and easier to debug as the website grows in complexity. Its versatility enables real-time interaction, ensuring users can access up-to-date product details, product bookings, and enjoy a seamless shopping experience.

3.2.1.4 REACT JS

React JS is a powerful JavaScript library used for building user interfaces, especially for single-page applications. React allows developers to create reusable components, which are self-contained pieces of code that can be combined to build dynamic and complex web applications. For the MANIYAN STORES E-commerce Platform, React will be used to create a highly interactive and responsive user interface that can handle and display dynamic data such as product listings, user profiles, and order details without compromising performance. React's use of the virtual DOM (Document Object Model) ensures efficient updates and rendering, enhancing the site's performance by reducing direct manipulations of the real DOM. This ensures that even when new products are added or orders are updated, the site remains fast and responsive, providing a seamless user experience. Additionally, React's component-based structure promotes code reuse, making the development process faster and more organized. This modularity makes it easier to maintain and scale the website as the platform grows. By managing the application state efficiently, React allows smooth handling of user interactions such as product selection, order management, and personalized recommendations, offering an engaging and dynamic shopping experience for customers.

3.2.2 BACKEND

3.2.2.1 MONGO DB

MongoDB is a highly flexible and scalable NoSQL database system, ideal for managing large and complex datasets. Unlike traditional relational databases, MongoDB stores data in JSON-like documents, providing a more dynamic and adaptable structure. This flexibility is essential for the MANIYAN STORES E-commerce Platform, where the system needs to manage various types of data, such as product listings, customer profiles, and order details. MongoDB efficiently handles unstructured or semi-structured data, making it easy to store and retrieve product and customer information. Additionally, MongoDB's horizontal scaling capabilities ensure that as the platform grows, it can manage increased traffic and data volumes without sacrificing performance. This makes MongoDB a perfect choice for supporting the scalability and flexibility required for the online shopping platform.

3.2.2.2 NODE JS

Node.js is a powerful runtime environment that enables JavaScript to be executed on the server side, making it an ideal choice for building scalable and high-performance web

applications. For the MANIYAN STORES E-commerce Platform, Node.js will serve as the server-side engine that processes incoming customer requests, such as product inquiries, bookings, and purchases, while efficiently communicating with the database to provide real-time responses. Its non-blocking, event-driven architecture allows Node.js to handle multiple customer interactions simultaneously, ensuring smooth performance even during peak traffic times. This makes Node.js an excellent choice for the platform, where multiple users may be browsing products, placing orders, or making purchases at the same time.

CHAPTER 4

SYSTEM DESCRIPTION

4.1 MODULE DESCRIPTION

The project contains the following modules such as:

- Login/Register
- Home
- Product
- Search and Filter
- Cart
- Order
- Payment
- Contact us
- Edit Profile
- Admin

4.1.1 LOGIN/REGISTER

The Login module allows users to securely log in by entering their username and password, with credentials validated and stored in an encrypted format. The Register module enables users to create an account by providing a username, email, password, and a confirmation password (cpassword). Both modules are designed to ensure secure user access, safeguarding personal information from unauthorized access.

4.1.2 HOME

The home page acts as the main entry point for users, offering an intuitive and visually engaging interface for easy navigation through sections like products, about us, contact us, and help. A dynamic top bar displays the logo, user profile options, and a

shopping cart icon with real-time item updates, improving user engagement. If users try to access product features without logging in, a prompt appears to ensure secure access. The page also uses animations and effects to provide a lively, enjoyable user experience.

4.1.3 PRODUCT

The product module provides a user-friendly interface for browsing a variety of grocery items, such as fresh produce, packaged goods, dairy products, beverages, and more. Users can easily filter products by categories, brands, and types using dropdown menus. Each product is displayed with high-quality images, detailed specifications (including product name, quantity, brand, and price), and options to add items to the cart. While direct online purchasing may not be available in this module, users can add products for future consideration or use the "Buy Now" feature for quick checkout, enhancing the overall shopping experience.

4.1.4 SEARCH AND FILTER

The search and filter module on the product page allows users to efficiently browse and find the grocery items they need. The search bar lets users enter keywords to quickly locate specific products, making the shopping experience faster and more convenient. Alongside the search function, a comprehensive filtering system enables users to narrow down products based on category, type, price range, and brand. This filter system ensures that customers can easily explore products that meet their specific preferences. Each filter is designed to be intuitive, offering clear dropdown options, and the selected filters are prominently displayed for users to review. Additionally, users can remove or reset individual filters with ease, providing full control over their search process. The module enhances overall user experience by offering both flexibility and precision, helping customers find exactly what they're looking for in a timely manner.

4.1.5 CART

The cart module manages the shopping cart functionality for an online grocery shopping platform. It uses React hooks to manage the cart's state, including item quantities, pricing, and payment options. Upon component mount, it fetches cart items from the backend API, initializing their quantities and prices. Users can update item quantities, remove products, and initiate the checkout process for selected items. The component

displays loading animations while fetching data and a notification if the cart is empty. Visual effects are enhanced with Lottie animations, and Axios is used for API requests, ensuring a seamless shopping experience with options for order and payment processing.

4.1.6 ORDER

The order module is an essential component of the online grocery shopping platform, streamlining the process from cart to final purchase. Once users are ready to check out, they can view a detailed summary of their selected items, including product names, quantities, prices, and total cost. The module enables users to enter their shipping information and choose from available payment options, ensuring a smooth and secure transaction. After the order is placed, users receive an order confirmation, and the backend processes the transaction, updating inventory and notifying users about order status. This module enhances the overall shopping experience by offering an intuitive and efficient flow, allowing users to easily track and complete their grocery orders with confidence.

4.1.7 PAYMENT

The payment module facilitates seamless transactions for users purchasing grocery products through various methods, including UPI QR codes and cash. Users can conveniently scan a QR code with their mobile device to complete the payment or opt for cash payment for offline orders. The module exclusively supports UPI payments and cash transactions, ensuring a straightforward payment process without the inclusion of credit or debit card options. Upon successful transactions, the system generates payment confirmations and receipts, which are promptly sent to users. This approach guarantees an efficient payment experience, allowing users to finalize their purchases with ease.

4.1.8 CONTACT US

The contact us module allows users to easily get in touch for assistance regarding their grocery shopping experience. Users can fill out a form to submit inquiries, feedback, or requests, which will be forwarded to the relevant support team for quick resolution. Additionally, key contact details such as a dedicated support phone number and email address are provided to enable direct communication. This module ensures that user inquiries and concerns are addressed promptly, contributing to improved customer satisfaction and overall support.

4.1.9 EDIT PROFILE

This module allows users to update their personal details such as username, email, and password. Users can modify their profile information at any time, including uploading a profile picture. The module also provides an option to change security settings like the password, ensuring users can maintain the security of their accounts. Profile changes are subject to authentication checks for user safety.

4.1.10 ADMIN

The Admin module provides administrators with full control over the platform's operations. Admins can view, manage, and update user profiles and product listings, ensuring the platform remains organized and up-to-date. They also have the ability to track payment records, including transactions made via UPI or cash, ensuring accurate financial logs. In addition, admins can edit or remove product content as necessary, maintaining the accuracy and relevance of the product database. This module streamlines the management and monitoring of platform activities, enhancing overall efficiency and ensuring the system runs smoothly.

4.2 USE CASE DIAGRAM

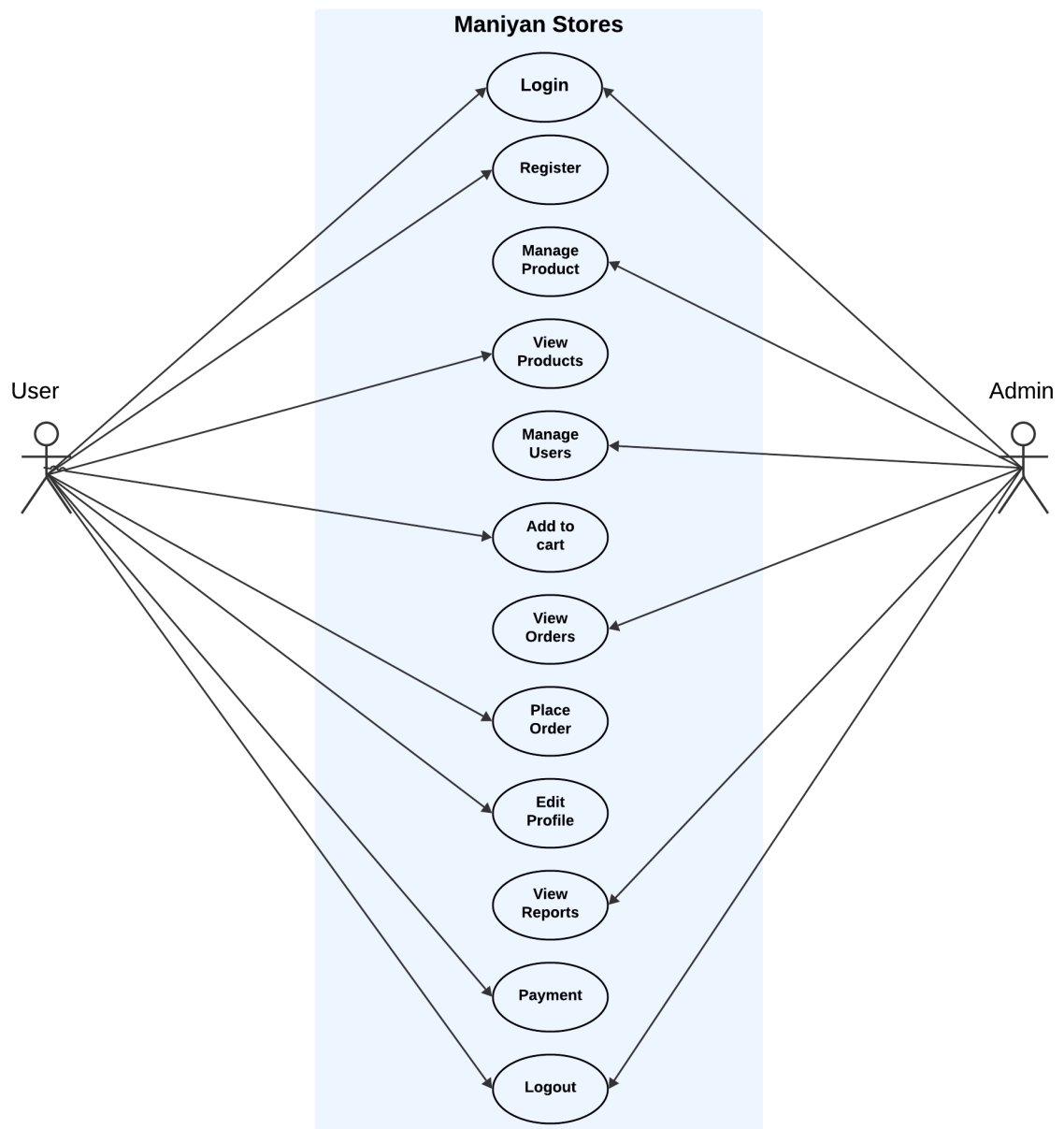


Figure 4.1 Use Case Diagram

4.3 SYSTEM FLOW DIAGRAM

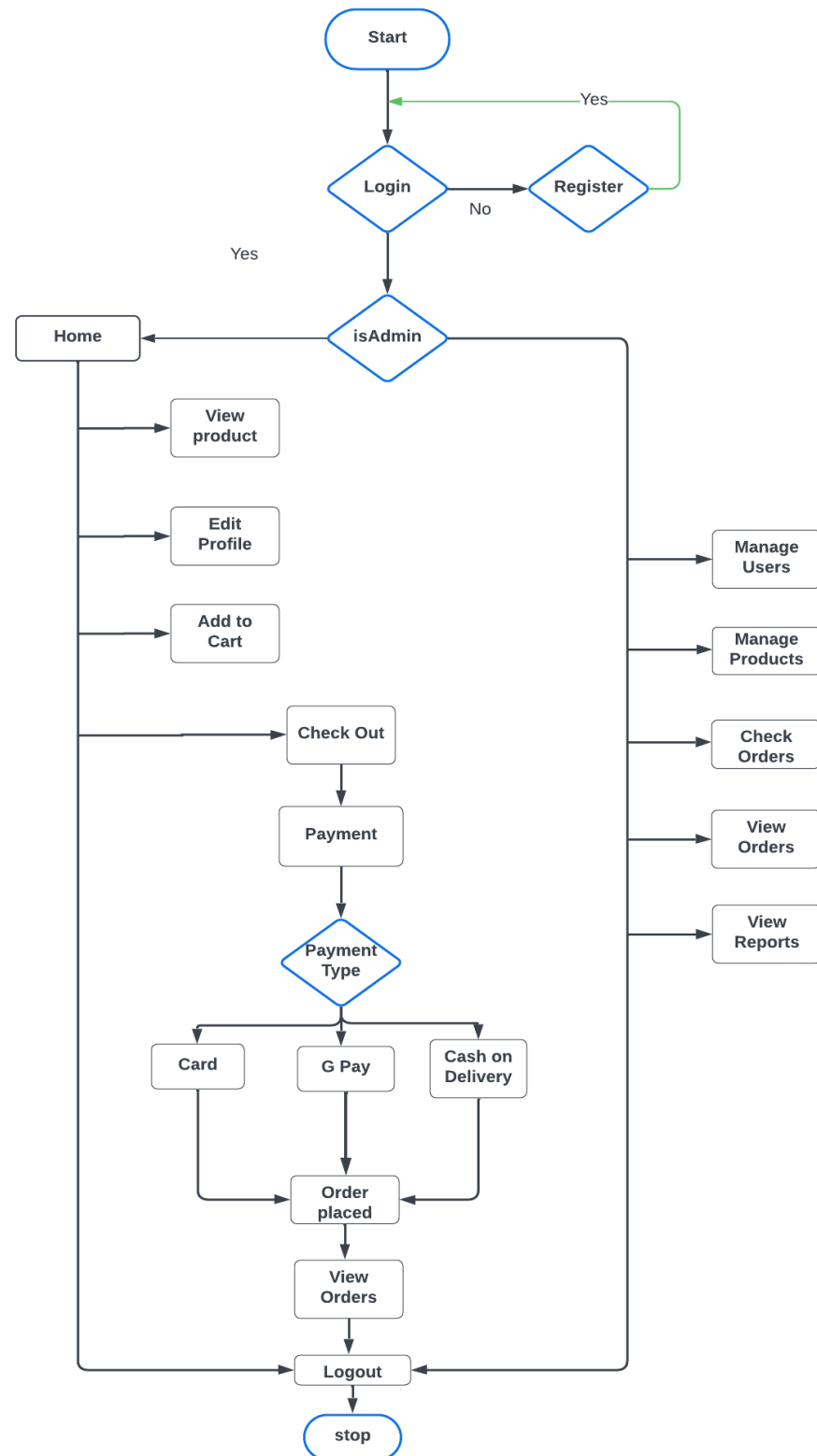


Figure 4.2 System Flow Diagram

4.4 DATA FLOW DIAGRAM

4.4.1 DATA FLOW DIAGRAM (LEVEL 0)

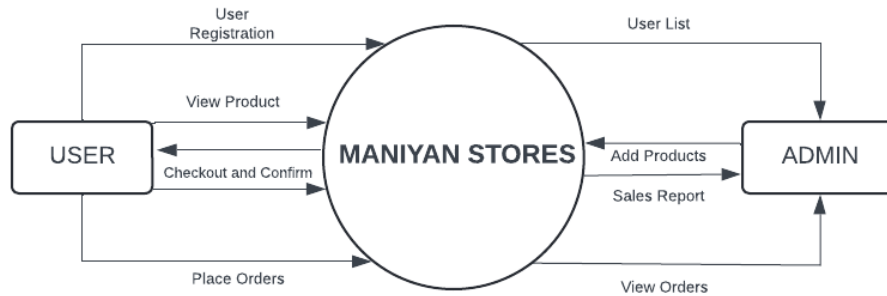


Figure 4.3 Data Flow Diagram (Level-0)

4.4.2 DATA FLOW DIAGRAM (LEVEL 1)

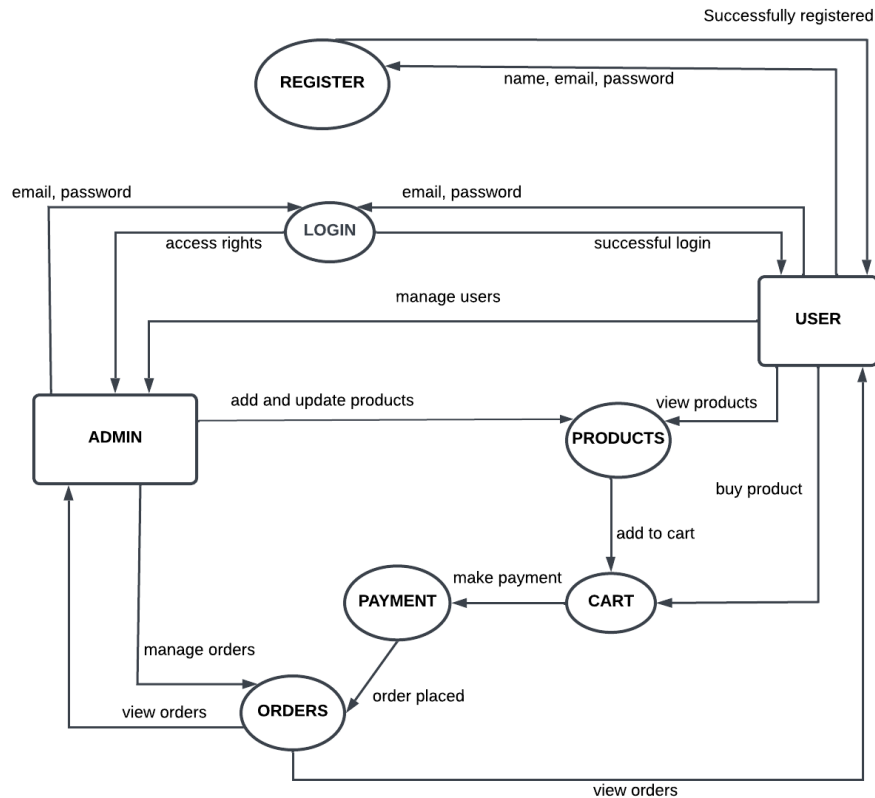


Figure 4.4 Data Flow Diagram (Level-1)

4.5 DATABASE DESIGN

TABLE NO : 4.5.1

TABLE NAME : users

DESCRIPTION : The table is used to store the login details of admin and the user.

FIELD NAME	DATA TYPE	DESCRIPTION
_id	Object	ObjectId
name	String	Username
email	String	Email
isAdmin	Boolean	User / Admin Role
isGoogleUser	Boolean	User Login Method
image	String	Profile Image
phonenumber	Number	Phone Number
address	String	User Address
password	String	Password
createdAt	Date	Account Creation Date
updatedAt	Date	Account Updation Date

TABLE NO : 4.5.2

TABLE NAME : products

DESCRIPTION : The table is used to store the product details.

FIELD NAME	DATA TYPE	DESCRIPTION
_id	Object	ObjectId
productImage	String	Product image
productCategory	String	Product Category
productType	String	Product Type
productBrand	String	Product Brand
productName	String	Product Name
productUnit	String	Product Unit
productContainerType	String	Product Container Type
productExpirationPeriod	String	Product Expiration Period
productPrice	Number	Product Price
productPreviousPrice	Number	Product Previous Price
productStock	Number	Product Stock
createdAt	Date	Product Creation Date
updatedAt	Date	Product Updation Date

TABLE NO : 4.5.3

TABLE NAME : carts

DESCRIPTION : The table is used to store the cart details.

FIELD NAME	DATA TYPE	DESCRIPTION
_id	Object	ObjectId
userId	String	User id
products	Array	Product items

TABLE NO : 4.5.4

TABLE NAME : orders

DESCRIPTION : The table is used to store the order details.

FIELD NAME	DATA TYPE	DESCRIPTION
_id	Object	ObjectId
productDetails	Array	Product Details
email	String	Email
userId	String	User id
address	String	User Address
paymentDetails	Array	Product Details
shipping_options	Array	Shipping Options
totalAmount	Number	Total Amount
orderStatus	String	Order Status
deliveryDate	Date	Delivery Date

4.6 INPUT DESIGN

In the proposed work, the input forms designed include the Customer Registration Form. The Customer Registration Form collects essential details such as Email ID and Password. Validation is implemented for the password to ensure it meets the required length criteria. If the input is invalid or incomplete, appropriate error messages are displayed to guide users. Following is the Input design:

- Login form
- Registration form
- Edit profile form

LOGIN FORM

The Login Form is the gateway for users to access their accounts, requiring an email ID and password for authentication. It ensures a seamless and secure login process with validation checks, guiding users through clear error messages if credentials are incorrect or missing. The form also includes a forgot password option for account recovery and supports Google Sign-In for convenience. With a responsive design, it offers a smooth experience across different devices, encouraging effortless access to the platform.

REGISTER FORM

The Register Form in Maniyan Stores enables users to create an account by providing their name, email, password, and confirmation password. It ensures secure registration with email validation, password length checks, and a mismatch warning to prevent errors. Clear messages guide users through corrections, and a success notification confirms registration. The form maintains a user-friendly design, featuring a responsive layout and engaging animations, ensuring accessibility across devices. Additionally, users can quickly navigate to the sign-in page if they already have an account, enhancing the overall experience.

EDIT PROFILE FORM

The edit profile form allows users to update their personal information easily and securely. It should pre-fill existing data to simplify the editing process while including validation messages to ensure accurate input. Requiring password re-entry for significant

changes enhances security and prevents unauthorized modifications. A straightforward design and clear instructions contribute to a positive user experience during the profile update.

4.7 OUTPUT DESIGN

The source of information to be generated through this work is referred to as the output. The systems should be improved through efficient and understandable output design.

Following is the output design:

- Generated Bill

GENERATING BILL

The Generating Bill section systematically compiles product details, including product name, brand, category, unit, container type, expiration period, and pricing, ensuring a transparent and organized billing process. It highlights essential transaction information such as stock availability and previous vs. current pricing, allowing users to make informed purchase decisions. The bill further includes timestamps for tracking and supports downloading a detailed invoice in PDF format, making it convenient for users to maintain a record of their grocery purchases.

CHAPTER 5

SYSTEM TESTING

System testing is a type of testing that evaluates the overall functionality and performance of a complete and fully integrated software. System testing is performed on a complete integrated system to evaluate the compliance of the system with the corresponding requirements.

5.1 UNIT DESIGN

In unit testing, we must test the programs making up the system. By giving improper inputs, the errors occurred are noted and eliminated. This enables, to detection of errors in coding and logic that are contained within the module alone. The testing was carried out during the programming. In this system each form is considered as a separate unit and tested for errors. Every user input is unit tested for a valid accepted range.

Test Case 1

Module	: Admin Login
Login Type	: Loading of the appropriate form for the administrator
Input	: Username and Password
Expected Output	: Display admin dashboard

Sample Test Case

Output	: Redirect to Main Page and display the admin menus.
Analysis	: In this form, the admin's username and password are verified. If correct, the Admin Dashboard will be displayed.

5.2 INTEGRATION TESTING

Testing is done for each module. After testing all the modules, the modules are integrated and testing of the final system is done with the test data, specially designed to show that the system will operate successfully in all its aspects conditions.

Test Case 1

Module	: Admin
Test Type	: Product Management and Order Management.
Input	: Navigation between Admin options.
Expected Output	: Navigation between modules is completed successfully.

Sample Test Case

Output	: On Clicking Login and other Admin modules the respective pages will open correctly.
Analysis	: Ensures that all admin functionalities navigate to the correct pages without errors.

5.3 VALIDATION TESTING

Verification and validation testing are two important tests, which are carried out before the product has been handed over to the customer. It determines whether the software function as the user expected.

Test Case 1

Module	: User Registration.
Login Type	: Register new user.
Input	: Input to all required fields.
Expected Output	: Required field should not be empty.

Sample Test Case

Input : Input for a required field is not provided.

Output : Prompt user to provide all the required fields.

Analysis : Navigation to the next page is blocked until all required fields are correctly.

CHAPTER 6

SYSTEM IMPLEMENTATION

System implementation is the stage of the project where the theoretical design is turned into a working system. If the implementation stage is not properly planned and controlled, it can cause errors. Thus, it can be the most crucial stage in achieving a successful new system and in giving the user confidence that the new system will work and be effective.

Implementation is the process of converting a new or revised system design into an operational one when the initial design was done by the system, a demonstration was given to the end user about the working system.

This process is used to verify and identify any logical mess working in the system by feeding various combinations of test data. After the approval of the system by both the end user and management the system was implemented. System implementation is made up of many activities. The six major activities are as follows.

CODING

Coding is the process in which the physical design specifications, created by the analysis team, are transformed into working computer code by the programming team. This stage involves selecting appropriate programming languages and tools that align with the system's requirements. The quality of the code is vital, as it lays the foundation for the system's functionality and performance.

TESTING

Testing begins once the coding process starts and continues in parallel. Each program module is subjected to rigorous testing to ensure it operates as intended. This process includes unit testing, integration testing, and system testing to validate that all components work together harmoniously. Effective testing ensures that a high-quality product is delivered to the customers, minimizing the risk of defects and enhancing user satisfaction.

INSTALLATION

Installation is the process of replacing the current system with the new one. This includes converting existing data, updating software, and aligning documentation and work procedures with the new system. Careful planning is essential during installation to ensure a seamless transition, minimizing disruptions to daily operations and ensuring that users can adapt to the new system effectively.

DOCUMENTATION

Documentation emerges from the installation process and includes user guides that provide comprehensive information on how to operate the system and its flow. This documentation serves as a vital resource for users, helping them understand the system's functionalities, troubleshooting common issues, and facilitating smoother onboarding for new users.

TRAINING AND SUPPORT

A well-developed training plan is essential to equip users with the knowledge and skills needed to navigate the new system effectively. The training strategy likely begins in the earlier stages of the project, ensuring that users can quickly acclimate to the system. Ongoing support is also critical, providing users with assistance and resources as they adapt to the changes, ultimately contributing to the system's overall success and user satisfaction.

CHAPTER 7

CONCLUSION AND FUTURE ENHANCEMENTS

7.1 CONCLUSION

The website for Maniyan Stores has been successfully developed, utilizing React JS for the frontend and Node JS with Express JS for the backend. This combination provides a seamless, user-friendly experience while ensuring robust performance and scalability. The design prioritizes intuitive navigation, responsive layouts for all devices, and optimized loading speeds. Secure transactions are integrated through trusted payment gateways, safeguarding customer data. Additionally, comprehensive error handling and logging systems are in place to ensure smooth operations and quick issue resolution.

On the backend, the infrastructure is built to handle data efficiently, enabling fast retrieval and secure storage of product details, customer information, and orders. A dedicated admin dashboard empowers store managers to update products, track orders, and oversee customer interactions with ease. The system's architecture is designed for scalability, allowing future expansion as the business grows. SEO optimization strategies are also embedded to improve visibility and attract organic traffic, enhancing the store's online presence.

Extensive testing and validation ensure reliability, performance, and user satisfaction. Unit testing, integration testing, and user acceptance testing were conducted to minimize errors and ensure a robust user experience. The platform also includes analytics and performance tracking, supporting continuous improvements based on real-world usage data. This successful implementation not only boosts operational efficiency but also lays a strong foundation for future upgrades, contributing to the long-term growth and success of Maniyan Stores.

APPENDIX 1 - SAMPLE CODING

HOME

```
import React, { useEffect, useState } from 'react';
import Navbar from './Navbar/Navbar';
import { useNavigate } from "react-router-dom";
import { FontAwesomeIcon } from "@fortawesome/react-fontawesome";
import Loading_Animation from "../Pages/Animation/Loading_Animation";
import { faTruckFast, faLeaf, faCreditCard } from "@fortawesome/free-solid-svg-icons";
import './Home.css';
import Slider from './Slider/Slider';
function Home() {
  const navigate = useNavigate();
  const [products, setProducts] = useState([]);
  const [loading, setLoading] = useState(true);
  const [pageLoading, setPageLoading] = useState(true);
  useEffect(() => {
    setTimeout(() => setPageLoading(false), 500);
  }, []);
  const handleProductClick = (product) => {
    navigate(`/product/${product._id}`, { state: product });
  };
  useEffect(() => {
    const fetchProducts = async () => {
      try {
        setLoading(true);
        const response = await fetch("http://localhost:8080/products/display-product-
data");
        const data = await response.json();
        const allProducts = data.products || data;
        const randomProducts = allProducts.sort(() => 0.5 - Math.random()).slice(0, 5);
```

```

        setProducts(randomProducts);
    } catch (error) {
        console.error("Error fetching products:", error);
    } finally {
        setTimeout(() => setLoading(false), 300);
    }
};

fetchProducts();
}, []);

const calculateOffer = (previousPrice, currentPrice) => {
    if (previousPrice && previousPrice > currentPrice) {
        const discount = Math.round(((previousPrice - currentPrice) / previousPrice) *
100);
        return `${discount}% off`;
    }
    return null;
};

if (pageLoading) {
    return (
        <div className="loading-wrapper">
            <Loading_Animation />
        </div>
    );
}

return (
    <div>
        <Navbar />
        <Slider />
        <div className="random-products">
            <h2>Featured Products</h2>
            {loading ? (
                <p>Loading products...</p>

```

```

): (
  <div className="product-grid">
    {products.map((product) => (
      <div className="product-card" key={product._id}>
        <img src={product.productImage} alt={product.productName}
className="product-image" />
        <div className="product-details">
          <h3>{product.productName}</h3>
          <p>Units: {product.productUnit}</p>
          <div className="product-details-a">
            <p className="product-price">Price:
₹ {product.productPrice}</p>
            {product.productPreviousPrice && (
              <p className="product-previous-
price"><strike>{product.productPreviousPrice}</strike></p>
            )}
            {product.productPreviousPrice && product.productPreviousPrice
> product.productPrice && (
              <p className="product-offer">
                {calculateOffer(product.productPreviousPrice,
product.productPrice)}
              </p>
            )}
          </div>
          <div className="product-buttons">
            <button onClick={() => handleProductClick(product)}>VIEW
PRODUCT</button>
          </div>
        </div>
      </div>
    )})
  </div>
)
</div>

```

```

    <div className="features-section">
      <div className="feature">
        <i className="feature-icon"><FontAwesomeIcon icon={faTruckFast} /></i>
        <h3>Fast Delivery</h3>
        <p>Get fresh groceries delivered to your home quickly and safely.</p>
      </div>
      <div className="feature">
        <i className="feature-icon"><FontAwesomeIcon icon={faLeaf} /></i>
        <h3>Fresh Produce</h3>
        <p>We offer only the freshest fruits, vegetables, and herbs.</p>
      </div>
      <div className="feature">
        <i className="feature-icon"><FontAwesomeIcon icon={faCreditCard}
/></i>
        <h3>Secure Payment</h3>
        <p>Shop with confidence with our secure payment options.</p>
      </div>
    </div>
  </div>
);
}
export default Home;

```

PRODUCT

```

import React, { useState, useEffect, useCallback } from "react";
import { useLocation, useNavigate } from "react-router-dom";
import Navbar from "../Navbar/Navbar";
import { FontAwesomeIcon } from "@fortawesome/react-fontawesome";
import { faFilter, faTimes } from "@fortawesome/free-solid-svg-icons";
import Loading_Animation from "../Animation/Loading_Animation";
import "../Product.css";

```

```

function Product() {
  const [products, setProducts] = useState([]);
  const [searchQuery, setSearchQuery] = useState("");
  const [loading, setLoading] = useState(true);
  const [searchLoading, setSearchLoading] = useState(false);
  const [filteredProducts, setFilteredProducts] = useState([]);
  const location = useLocation();
  const navigate = useNavigate();
  const [selectedCategory, setSelectedCategory] = useState(
    localStorage.getItem("selectedCategory") || "All"
  );
  const [selectedType, setSelectedType] = useState(
    localStorage.getItem("selectedType") || "All"
  );
  const [selectedPriceRange, setSelectedPriceRange] = useState(
    localStorage.getItem("selectedPriceRange") || "All"
  );
  const [selectedBrand, setSelectedBrand] = useState(
    localStorage.getItem("selectedBrand") || "All"
  );
  const [categoryOptions] = useState([
    "All",
    "Food Products",
    "Dairy Products",
    "Beverages",
    "Fruits and Vegetables",
    "Bread and Desserts",
    "Cleaning Supplies",
  ]);
  const [typeOptions, setTypeOptions] = useState(["All"]);
  const [priceRangeOptions] = useState(["All", "0 - 100", "101 - 500", "501 - 1000", "1000+"]);

```



```

const [brandOptions, setBrandOptions] = useState(["All"]);
useEffect(() => {
  localStorage.setItem("selectedCategory", selectedCategory);
  localStorage.setItem("selectedType", selectedType);
  localStorage.setItem("selectedPriceRange", selectedPriceRange);
  localStorage.setItem("selectedBrand", selectedBrand);
}, [selectedCategory, selectedType, selectedPriceRange, selectedBrand]);
useEffect(() => {
  const fetchProducts = async () => {
    setLoading(true);
    try {
      const response = await fetch("http://localhost:8080/products/display-product-data");
      const data = await response.json();
      setProducts(data.products);
      const uniqueTypes = [...new Set(data.products.map((p) => p.productType))];
      const uniqueBrands = [...new Set(data.products.map((p) => p.productBrand))];
      setTypeOptions(["All", ...uniqueTypes]);
      setBrandOptions(["All", ...uniqueBrands]);
      filterProducts(data.products);
    } catch (error) {
      console.error("Error fetching products:", error);
    } finally {
      setTimeout(() => setLoading(false), 200);
    }
  };
  fetchProducts();
}, []);
const filterProducts = useCallback(
  (productsToFilter = products) => {
    setSearchLoading(true);
    const filtered = productsToFilter.filter((product) => {

```

```

    const matchesSearchQuery =
      !searchQuery ||
      product.productName.toLowerCase().includes(searchQuery.toLowerCase());

    const matchesCategory = selectedCategory === "All" || product.productCategory ===
      selectedCategory;

    const matchesType = selectedType === "All" || product.productType ===
      selectedType;

    const matchesPriceRange =
      selectedPriceRange === "All" ||
      (selectedPriceRange === "0 - 100" && product.productPrice <= 100) ||
      (selectedPriceRange === "101 - 500" && product.productPrice > 100 &&
        product.productPrice <= 500) ||
      (selectedPriceRange === "501 - 1000" && product.productPrice > 500 &&
        product.productPrice <= 1000) ||
      (selectedPriceRange === "1000+" && product.productPrice > 1000);

    const matchesBrand = selectedBrand === "All" || product.productBrand ===
      selectedBrand;

    return matchesSearchQuery && matchesCategory && matchesType &&
      matchesPriceRange && matchesBrand;

  });

  setTimeout(() => {
    setFilteredProducts(filtered);
    setSearchLoading(false);
  }, 200);
},
[searchQuery, selectedCategory, selectedType, selectedPriceRange, selectedBrand,
products]
);

useEffect(() => {
  filterProducts();
}, [searchQuery, selectedCategory, selectedType, selectedPriceRange, selectedBrand,
filterProducts]);

useEffect(() => {
  const params = new URLSearchParams(location.search);
  const query = params.get("search") || "";

```

```

    setSearchQuery(query);
  }, [location.search]);
const handleCategoryChange = (event) => setSelectedCategory(event.target.value);
const handleTypeChange = (event) => setSelectedType(event.target.value);
const handlePriceRangeChange = (event) => setSelectedPriceRange(event.target.value);
const handleBrandChange = (event) => setSelectedBrand(event.target.value);
const clearAllFilters = () => {
  setSelectedCategory("All");
  setSelectedType("All");
  setSelectedPriceRange("All");
  setSelectedBrand("All");
};
const removeFilter = (filterType) => {
  switch (filterType) {
    case "category":
      setSelectedCategory("All");
      break;
    case "type":
      setSelectedType("All");
      break;
    case "priceRange":
      setSelectedPriceRange("All");
      break;
    case "brand":
      setSelectedBrand("All");
      break;
    default:
      break;
  }
};
const handleCardClick = (product) => {

```

```

    navigate(`/product/${product._id}`, { state: product });
  };
  return (
    <div>
      <Navbar />
      {loading ? (
        <div className="loading-wrapper">
          <Loading_Animation />
        </div>
      ) : (
        <div className="product-container">
          <div className="filter-container">
            <div className="filter">
              <FontAwesomeIcon icon={faFilter} className="filter-icon" />
              <p>Filters</p>
            </div>
            <hr />
            <div className="selected-filters">
              {selectedCategory !== "All" && (
                <span className="filter-tag">
                  {selectedCategory} <FontAwesomeIcon icon={faTimes} onClick={() =>
removeFilter("category")} className="filter-tag-icon" />
                </span>
              )}
              {selectedType !== "All" && (
                <span className="filter-tag">
                  {selectedType} <FontAwesomeIcon icon={faTimes} onClick={() =>
removeFilter("type")} className="filter-tag-icon" />
                </span>
              )}
              {selectedPriceRange !== "All" && (
                <span className="filter-tag">

```

```

        {selectedPriceRange} <FontAwesomeIcon icon={faTimes} onClick={() =>
removeFilter("priceRange")} className="filter-tag-icon"/>
      </span>
    )}
    {selectedBrand !== "All" && (
      <span className="filter-tag">
        {selectedBrand} <FontAwesomeIcon icon={faTimes} onClick={() =>
removeFilter("brand")} className="filter-tag-icon"/>
      </span>
    )}
  </div>

  {([selectedCategory, selectedType, selectedPriceRange, selectedBrand].some(filter
=> filter !== "All")) && (
    <button className="clear-all-btn" onClick={clearAllFilters}>Clear All</button>
  )}
</div>

<label className="filter-select-label" htmlFor="category-
select">CATEGORIES</label>

<select id="category-select" value={selectedCategory}
onChange={handleCategoryChange} className="filter-select">
  {categoryOptions.map((option) => (
    <option key={option} value={option}>{option}</option>
  ))}
</select>
</div>

<div>
  <label className="filter-select-label" htmlFor="type-select">TYPE</label>

  <select id="type-select" value={selectedType} onChange={handleTypeChange}
className="filter-select">
    {typeOptions.map((option) => (
      <option key={option} value={option}>{option}</option>
    ))}
  </select>
</div>

```

```

<div>
  <label className="filter-select-label" htmlFor="price-range-select">PRICE
  RANGE</label>
  <select id="price-range-select" value={selectedPriceRange}
  onChange={handlePriceRangeChange} className="filter-select">
    {priceRangeOptions.map((option) => (
      <option key={option} value={option}>{option}</option>
    ))}
  </select>
</div>
<div>
  <label className="filter-select-label" htmlFor="brand-select">BRAND</label>
  <select id="brand-select" value={selectedBrand}
  onChange={handleBrandChange} className="filter-select">
    {brandOptions.map((option) => (
      <option key={option} value={option}>{option}</option>
    ))}
  </select>
</div>
</div>
<div className="product-grid">
  {searchLoading ? (
    <div className="loading-wrapper">
      <Loading_Animation />
    </div>
  ) : filteredProducts.length > 0 ? (
    filteredProducts.map((product) => {
      const offerPercentage = product.productPreviousPrice &&
      product.productPreviousPrice > product.productPrice
      ? Math.round(((product.productPreviousPrice - product.productPrice) /
      product.productPreviousPrice) * 100)
      : null;
      return (

```

```

<div
  className="product-card"
  key={product._id}
  onClick={() => handleCardClick(product)}
>
  <img src={product.productImage} alt={product.productName}
  className="product-image" />
  <div className="product-details">
    <h3>{product.productName}</h3>
    <p>Units: {product.productUnit}</p>
    <div className="product-details-a">
      <p className="product-price">Price: ₹ {product.productPrice}</p>
      {product.productPreviousPrice && (
        <p className="product-previous-
price"><strike>{product.productPreviousPrice}</strike></p>
      )}
      {offerPercentage && (
        <p className="product-offer">{offerPercentage}% off</p>
      )}
    </div>
    <p className={`product-stock ${product.productStock === 0 ||
product.productStock < 10 ? "low-stock" : ""}`}>
      {product.productStock === 0
        ? "Out of Stock!"
        : product.productStock < 10
          ? `Only ${product.productStock} left!`
          : `In Stock: ${product.productStock}`}
    </p>
  </div>
</div>
);
})

```

```

    ): (
      !loading &&
      <div className="no-product">
        
      </div>
    )}
  </div>
</div>
)}
</div>
);
}
export default Product;

```

CART

```

import React, { useEffect, useState } from "react";
import { useNavigate } from "react-router-dom";
import Navbar from "../Navbar/Navbar";
import axios from "axios";
import { FontAwesomeIcon } from "@fortawesome/react-fontawesome";
import { faArrowLeft, faCreditCard, faMobileAlt, faMoneyBillAlt } from
"@fortawesome/free-solid-svg-icons";
import Loading_Animation from "../Animation/Loading_Animation";
import Lottie from 'lottie-react';
import "./Cart.css";
import Empty from "../empty.json";
function Cart() {
  const [cartItems, setCartItems] = useState([]);
  const [loading, setLoading] = useState(true);
  const [showPaymentModal, setShowPaymentModal] = useState(false);

```



```

const [selectedItem, setSelectedItem] = useState(null);
const navigate = useNavigate();
const handleBackClick = () => navigate(-1);
const fetchCartItems = async () => {
  const userData = JSON.parse(localStorage.getItem("userdata"));
  if (!userData || !userData._id) {
    alert("Please log in first");
    setLoading(false);
    return;
  }
  try {
    setLoading(true);
    const { data } = await axios.get(`http://localhost:8080/cart?userId=${userData._id}`);
    setCartItems(data.map(item => ({
      ...item,
      quantity: item.quantity || 1,
      originalStock: item.product.productStock,
      product: {
        ...item.product,
        productStock: item.product.productStock - (item.quantity || 1)
      }
    })));
  } catch (error) {
    console.error("Error fetching cart items:", error);
  } finally {
    setTimeout(() => setLoading(false), 300);
  }
};

useEffect(() => {
  fetchCartItems();
}, []);

```

```

const handleRemoveFromCart = async (itemId) => {
  const userData = JSON.parse(localStorage.getItem("userdata"));
  if (!userData || !userData._id) {
    alert("Please log in first");
    return;
  }
  try {
    console.log("Removing item:", itemId);
    await axios.delete(`http://localhost:8080/cart/remove/${itemId}`, {
      data: { userId: userData._id },
      headers: { "Content-Type": "application/json" },
    });
    alert("Item removed from cart");
    fetchCartItems();
  } catch (error) {
    alert("Failed to remove item");
    console.error("Remove error:", error);
  }
};

const calculateOffer = (previousPrice, currentPrice, quantity) => {
  if (previousPrice && previousPrice > currentPrice) {
    return `${Math.round(((previousPrice - currentPrice) / previousPrice) * 100)}% off`;
  }
  return null;
};

const handleQuantityChange = (itemId, delta) => {
  setCartItems(prevItems => prevItems.map(item => {
    if (item._id === itemId) {
      const newQuantity = item.quantity + delta;
      if (newQuantity > 10) {
        alert("Sorry! Only 10 units allowed in each order");
      }
    }
  }));
};

```

```

    return item;
  }
  if (newQuantity < 1) return item;
  const updatedQuantity = newQuantity > 0 ? newQuantity : 1;
  const updatedStock = item.product.productStock - delta;
  if (updatedStock > item.originalStock) {
    return item;
  }
  if (updatedStock < 0) {
    alert("Not enough stock available");
    return item;
  }
  return {
    ...item,
    quantity: updatedQuantity,
    product: {
      ...item.product,
      productStock: updatedStock
    }
  };
}
return item;
}));
};

const calculateTotalSaved = () => {
  const subtotalSaved = cartItems.reduce((totalSaved, item) => {
    const itemPriceSaved = item.product.productPreviousPrice -
item.product.productPrice;
    if (itemPriceSaved > 0) {
      return totalSaved + itemPriceSaved * item.quantity;
    }
  }, 0);
  return subtotalSaved;
};

```

```

    }, 0);

    const deliveryCharge = 40;

    return subtotalSaved - deliveryCharge;
  };

  const calculateTotalAmount = () => {
    const subtotal = cartItems.reduce((acc, item) => acc + (item.product.productPrice *
item.quantity), 0);

    const deliveryCharge = 40;

    return subtotal + deliveryCharge;
  };

  const handleOrderAll = async () => {
    setShowPaymentModal(true);
    setSelectedItem(null);
  };

  const handleBuyNow = async (item) => {
    setSelectedItem(item);
    setShowPaymentModal(true);
  };

  const handlePaymentMethodSelect = (method) => {
    setShowPaymentModal(false);
    if (method === "card") {
      handleCardPayment(selectedItem);
    } else if (method === "upi") {
      if (selectedItem) {
        navigate("/google-payment", {
          state: {
            product: selectedItem.product,
            quantity: selectedItem.quantity
          }
        });
      } else if (cartItems.length > 0) {
        navigate("/google-payment", { state: { cartItems } });
      }
    }
  };

```

```

    } else {
      alert("No items in the cart!");
    }
  } else if (method === "cod") {
    handleCashOnDelivery(selectedItem);
  }
};

const handleCardPayment = async (selectedItem) => {
  const userData = JSON.parse(localStorage.getItem("userdata"));
  if (!userData || !userData._id) {
    alert("Please log in first");
    return;
  }
  const payload = {
    cartItems: selectedItem
      ? [{
          product: selectedItem.product._id,
          name: selectedItem.product.productName,
          images: [selectedItem.product.productImage],
          price: selectedItem.product.productPrice,
          quantity: selectedItem.quantity,
        }]
      : cartItems.map(item => ({
          product: item.product._id,
          name: item.product.productName,
          images: [item.product.productImage],
          price: item.product.productPrice,
          quantity: item.quantity,
        })),
    email: userData.email,
  };

```

```

try {
  const response = await axios.post("http://localhost:8080/payment/checkout", payload);
  if (response.data.url) {
    window.location.href = response.data.url;
  }
} catch (error) {
  console.error("Error during checkout:", error);
  alert("Failed to proceed to checkout!");
}
};

const handleCashOnDelivery = (selectedItem) => {
  alert("Cash on Delivery selected. Your order will be confirmed shortly!");
};

return (
  <
    <Navbar />
    {loading ? (
      <div className="loading-wrapper">
        <Loading_Animation />
      </div>
    ) : (
      <div className="cart-container">
        <div className="cart-left">
          {cartItems.length === 0 ? (
            <div className="cart-item-empty">
              <h2>Your Cart is Empty!</h2>
              <Lottie className="empty-img" animationData={Empty} loop={true} />
            </div>
          ) : (
            cartItems.map(item => (
              <

```

```

<div key={item._id} className="cart-item">
  <button className="back-button" onClick={handleBackClick}>
    <FontAwesomeIcon icon={faArrowLeft} className="faArrowLeft" />
  </button>
  <img src={item.product.productImage} alt={item.product.productName} />
  <div className="cart-item-details">
    <h2>{item.product.productName}</h2>
    <p>Units: {item.product.productUnit}</p>
    <div className="cart-item-details-a">
      <p className="price">₹ {item.product.productPrice * item.quantity}</p>
      {item.product.productPreviousPrice && (
        <p className="previous-
price"><strike>{item.product.productPreviousPrice * item.quantity}</strike></p>
      )}
      {item.product.productPreviousPrice && item.product.productPreviousPrice
> item.product.productPrice && (
        <p className="offer">{calculateOffer(item.product.productPreviousPrice,
item.product.productPrice, item.quantity)}</p>
      )}
    </div>
    <p className={`stock ${item.product.productStock === 0 ||
item.product.productStock < 10 ? "low-stock" : ""}`}>
      {item.product.productStock === 0
        ? "Out of Stock!"
        : item.product.productStock < 10
          ? `Only ${item.product.productStock} left!`
          : `In Stock: ${item.product.productStock}`}
    </p>
    <div className="quantity-controls">
      <button onClick={() => handleQuantityChange(item._id, -1)}>-</button>
      <span>{item.quantity}</span>
      <button onClick={() => handleQuantityChange(item._id, 1)}>+</button>
    </div>
  </div>

```

```

    </div>

    <div className="class-item-button">
      <button onClick={() => handleBuyNow(item)}>BUY NOW</button>
      <button className="remove-button" onClick={() =>
handleRemoveFromCart(item._id)}>REMOVE</button>
    </div>
  </div>

  <div className="class-item-button-mobile">
    <button onClick={() => handleBuyNow(item)}>BUY NOW</button>
    <button className="remove-button" onClick={() =>
handleRemoveFromCart(item._id)}>REMOVE</button>
  </div>
</hr>
</>
))
)}
</div>
{cartItems.length > 0 && (
  <div className="cart-right">
    <table className="cart-summary-table">
      <thead>
        <tr>
          <th colspan="2" className="cart-summary-heading">Cart Summary</th>
        </tr>
      </thead>
      <tbody>
        {cartItems.map((item, index) => (
          <tr key={index}>
            <td className="item-row">{item.product.productName} <b>(Quantity
{item.quantity})</b></td>
            <td>₹ {item.product.productPrice * item.quantity}</td>
          </tr>

```



```

    ))}
    <tr>
      <td className="total-row"><b>Delivery Charge</b></td>
      <td><b>₹40</b></td>
    </tr>
    <tr>
      <td className="total-row"><b>Total</b></td>
      <td><b>₹{calculateTotalAmount()}</b></td>
    </tr>
    {calculateTotalSaved() > 0 && (
    <tr>
      <td className="total-row"><b>You Saved</b></td>
      <td><b>₹{calculateTotalSaved()}</b></td>
    </tr>
    )}
    <tr>
      <td colspan="2">
        <button onClick={handleOrderAll}>ORDER ALL</button>
      </td>
    </tr>
  </tbody>
</table>
</div>
)}
</div>
)}
{showPaymentModal && (
  <div className="payment-modal-overlay">
    <div className="payment-modal">
      <h2>Select Payment Method</h2>
      <button onClick={() => handlePaymentMethodSelect("card")}>

```

```

    <FontAwesomeIcon icon={faCreditCard} className="credit-icon" />
    Debit/Credit Card
  </button>
  <button onClick={() => handlePaymentMethodSelect("upi")}>
    <FontAwesomeIcon icon={faMobileAlt} className="upi-icon" />
    UPI Payment
  </button>
  <button onClick={() => handlePaymentMethodSelect("cod")}>
    <FontAwesomeIcon icon={faMoneyBillAlt} className="cash-icon" />
    Cash on Delivery
  </button>
  <button onClick={() => setShowPaymentModal(false)}>Close</button>
</div>
</div>
)}
</>
);
}
export default Cart;

```

APPENDIX 2 - SAMPLE SCREENSHOTS

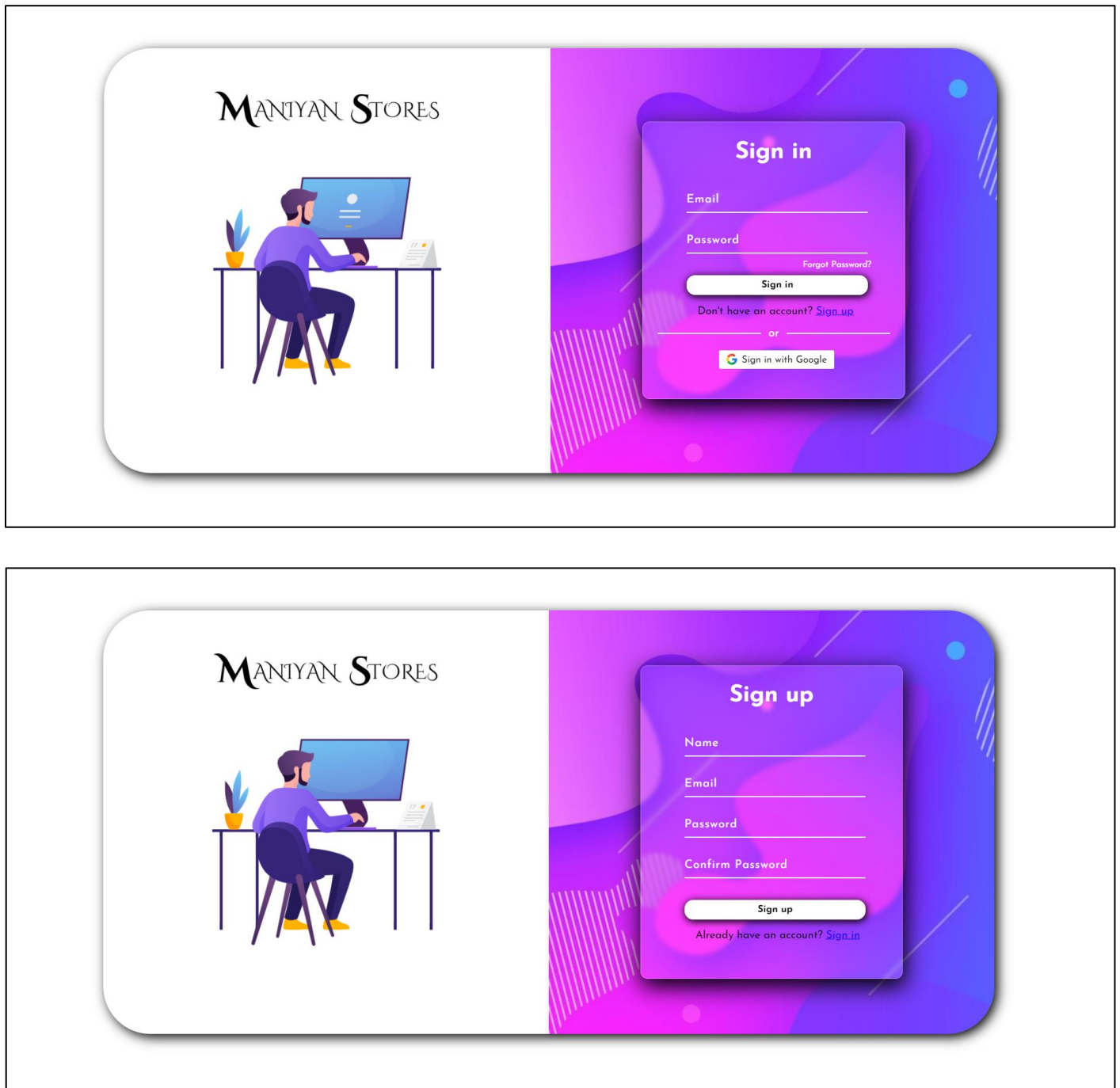


Figure A.2.1 Login and Register Page

MANTHAN STORES
Home Products Orders Contact
Search for Products...
Gowtham Prasath
Edit Profile
Admin Dashboard
Help & Support
Sign out

DAIRY DELIGHTS

HEALTHY DAIRY

Discover the goodness of fresh and healthy dairy products. Packed with essential nutrients, our dairy range ensures quality and taste in every bite. From creamy milk to delicious cheese, indulge in the best dairy has to offer!

[SEE MORE](#)

Featured Products

Colgate MaxFresh Toothpaste, Blue Gel Paste with Menthol
Units: 150 g x 4
Price: ₹318 328 3% off

[VIEW PRODUCT](#)

AASHIRVAAD Crystal Iodized Salt
Units: 1 Kg
Price: ₹20 22 9% off

[VIEW PRODUCT](#)

NutroVally Green Tea with Tulsi for Weight Loss
Units: 25 Sachets
Price: ₹172 249 31% off

[VIEW PRODUCT](#)

3 Roses Tea Box
Units: 250 g
Price: ₹225

[VIEW PRODUCT](#)

Taj Mahal Tea Non South Black Tea Box
Units: 500 g
Price: ₹360 384 6% off

[VIEW PRODUCT](#)

Fast Delivery

Get fresh groceries delivered to your home quickly and safely.

Fresh Produce

We offer only the freshest fruits, vegetables, and herbs.

Secure Payment

Shop with confidence with our secure payment options.

Figure A.2.2 Home Page

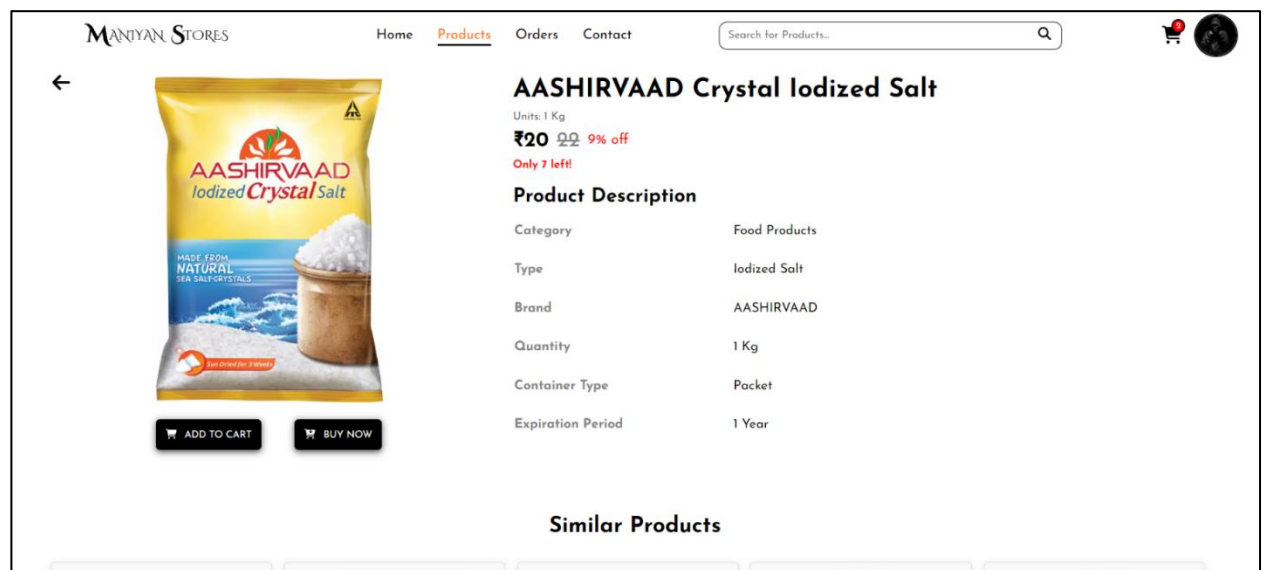
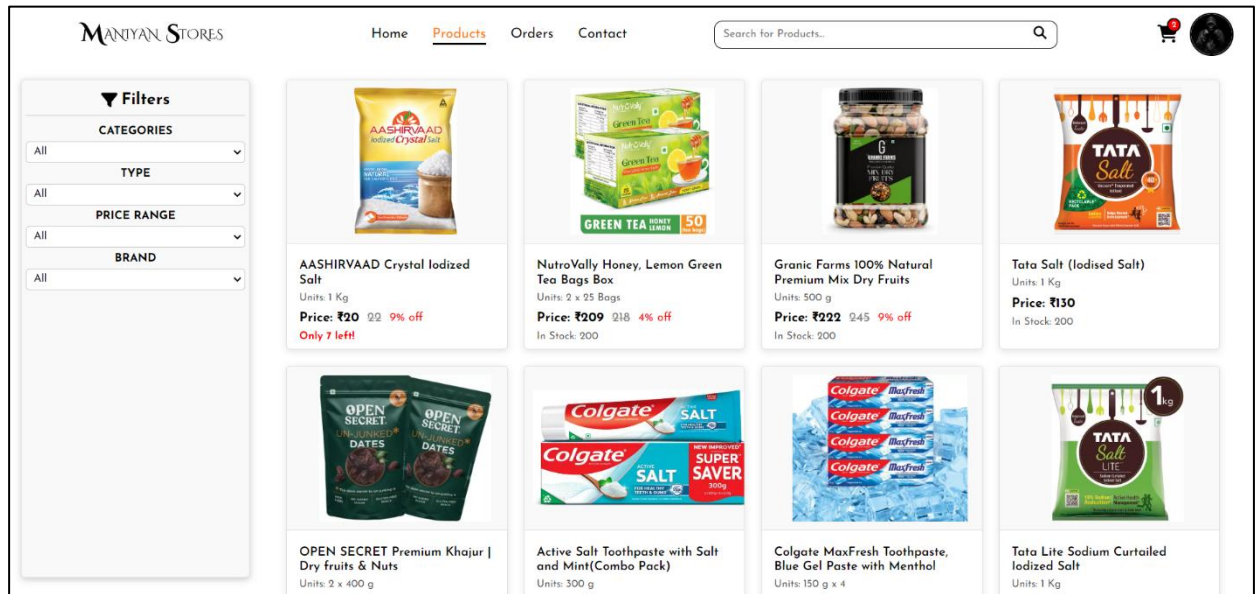


Figure A.2.3 Product Page

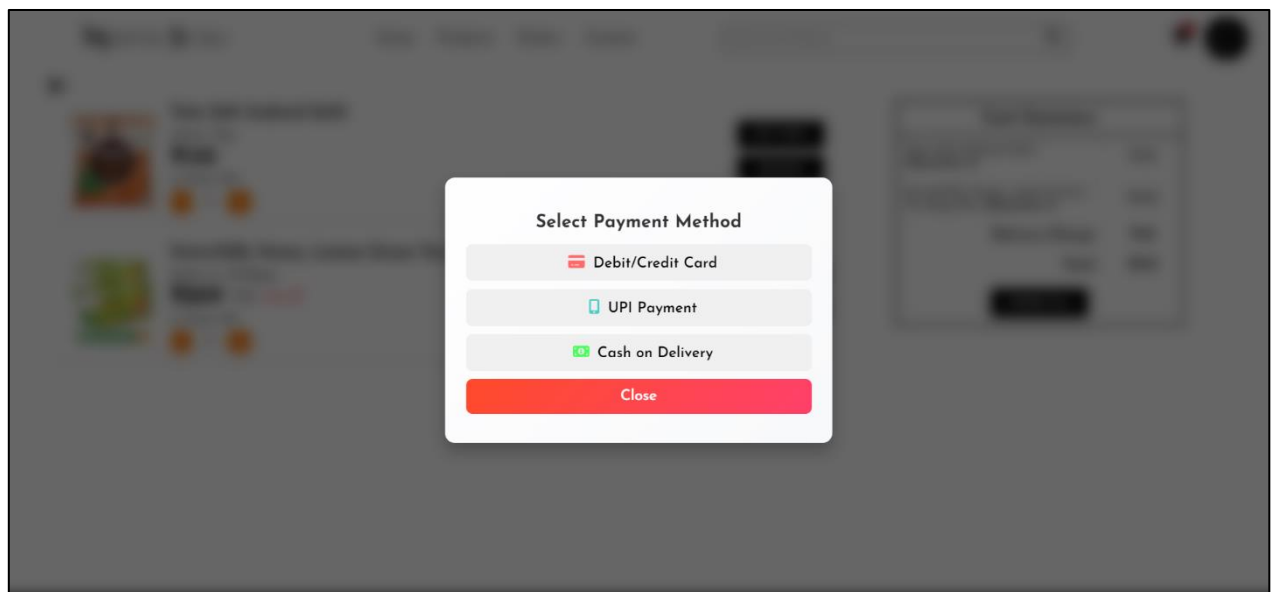
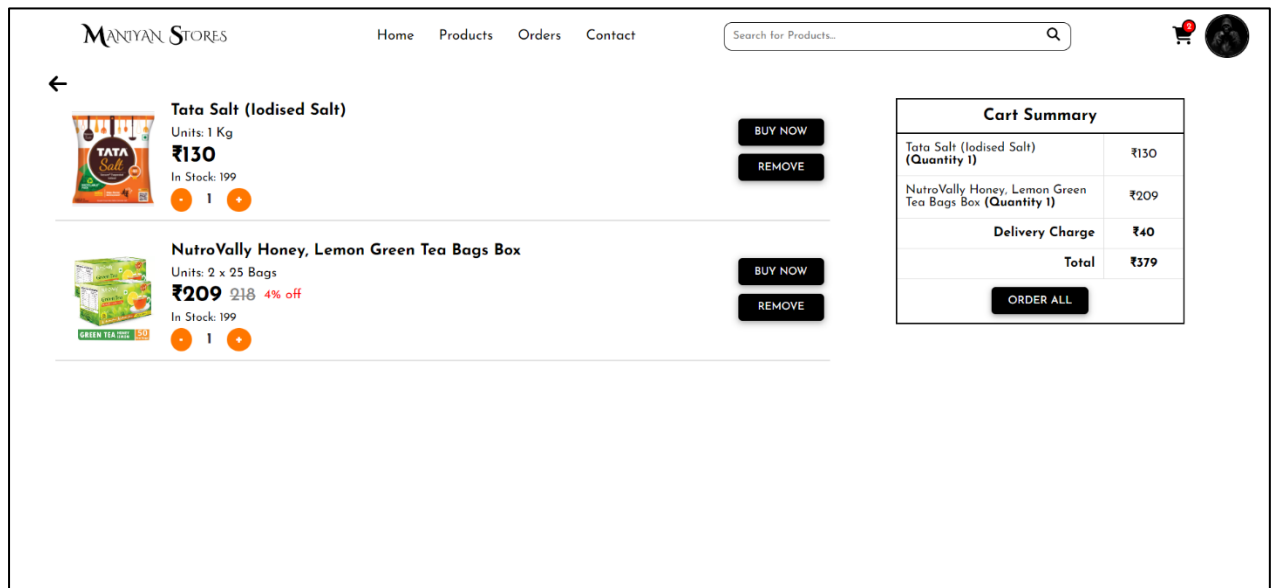



Figure A.2.4 Cart Page

← Payment TEST MODE


Pay Payment

₹639.00



Tata Salt (Iodised Salt)
Qty 3 ▾

₹390.00
₹130.00 each



NutroVally Honey, Lemon Green Tea
Bags Box
Qty 1 ▾

₹209.00

Subtotal



₹599.00

Shipping
Shipping Charge

₹40.00

Total due

₹639.00

 **link**  4242

Or pay with card





Email

gowthamprasath125@okaxis

Continue with Link


Card information

1234 1234 1234 1234

MM / YY

CVC




Cardholder name

Full name on card

Country or region

India ▾

Pay

Powered by  [Terms](#) [Privacy](#)


UPI PAYMENT

Total Amount : ₹379 (Incl. ₹40 Delivery Charge)

UPI ID:
Enter UPI ID (e.g., example@upi)

Pay via UPI

OR



UPI ID: gowthamprasath125@okaxis

Pay via QR

Figure A.2.5 Payment Page

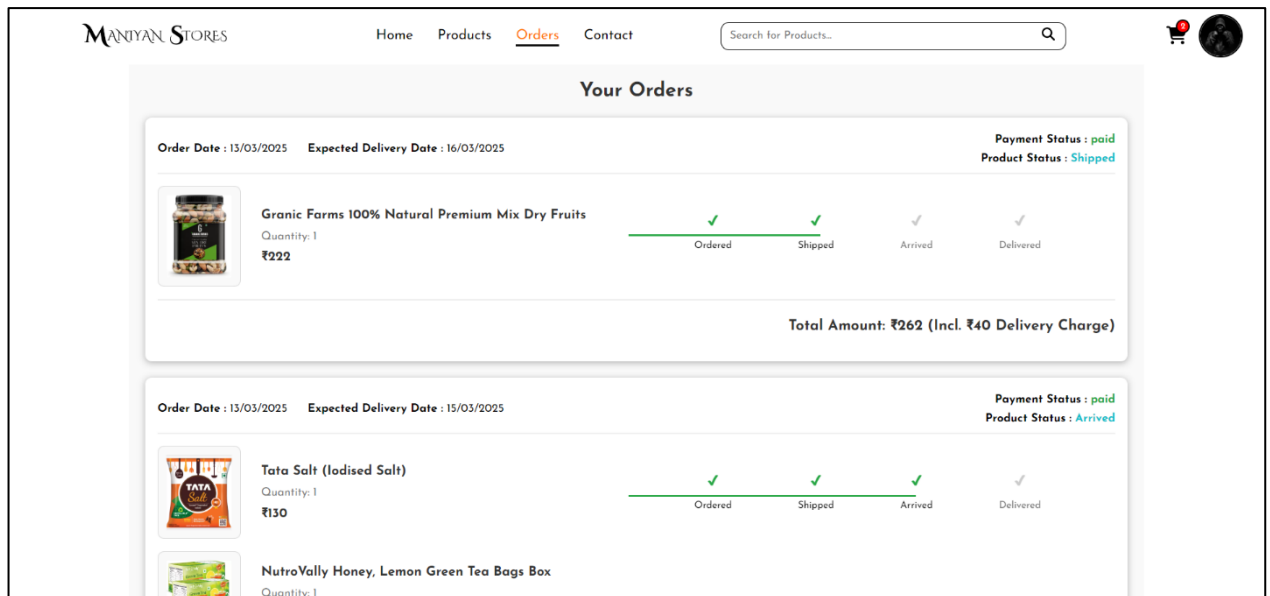


Figure A.2.6 Order Page

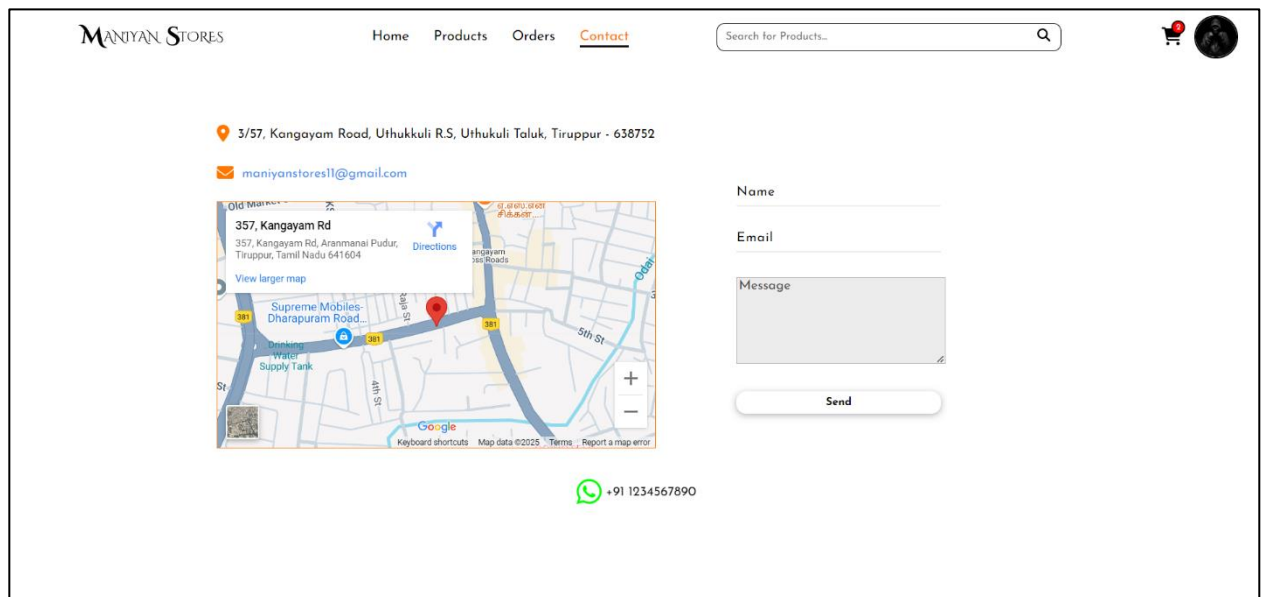


Figure A.2.7 Contact Page

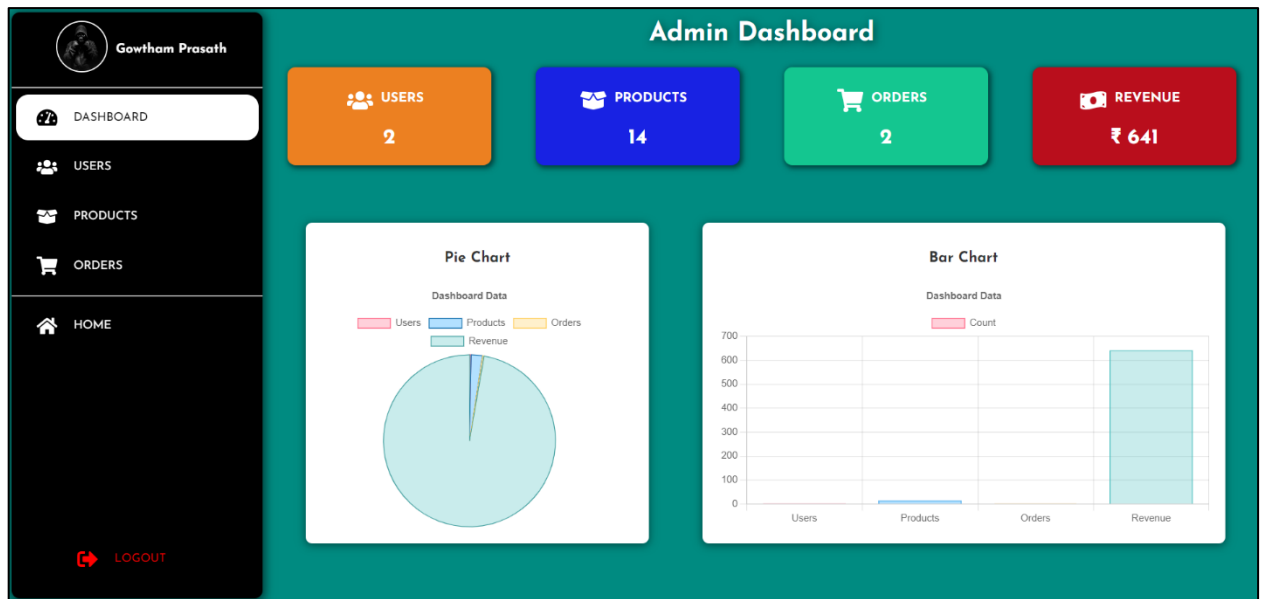



Figure A.2.8 Admin Dashboard

Users

Profile	Name	Email	Role	Phone	Address	Actions
	Gowtham Prasath	gowthamprasath122bir@kongu.edu	Admin	9344564546	Namakkal	
	Gowtham	gowthamprasath125@gmail.com	User	9344564546	DNC CSM Thetre road, B.Komarapalayam, Namakkal-638183	

Figure A.2.9 User Management


Gowtham Prasath


DASHBOARD

USERS

PRODUCTS

ORDERS

HOME

 LOGOUT

Products

ADD NEW PRODUCT

Add Product





Image	Category	Type	Brand	Name	Unit	Container Type	Expiration Period	Price	Previous Price	Stock	Actions
	Food Products	Iodized Salt	AASHIRVAAD	AASHIRVAAD Crystal Iodized Salt	1 Kg	Packet	1 Year	₹20	₹22	7	 
				NutraVally							

Figure A.2.10 Product Management


Gowtham Prasath


DASHBOARD

USERS

PRODUCTS

ORDERS

HOME

 LOGOUT



Order ID	User Email	Products	Total Amount	Payment Method	Payment Status	Order Status	Order Date	Delivery Date	Actions
67d2a8bf0ce96b070ec47d2	gowthamprasath122bir@kongu.edu	Tata Salt (Iodised Salt) (x1) NutraVally Honey, Lemon Green Tea Bags Box (x1)	₹379	UPI	paid	Arrived	13/03/2025	15/03/2025	
67d2a8faf0ce96b070ec49dc	gowthamprasath122bir@kongu.edu	Granic Farms 100% Natural Premium Mix Dry Fruits (x1)	₹262	card	paid	Shipped	13/03/2025	16/03/2025	

Figure A.2.11 Order Management

REFERENCES

- [1] **Ethan Brown**, "Web Development with Node and Express: Leveraging the JavaScript Stack," O'Reilly Media, First Edition, 2014.
- [2] **Alex Banks & Eve Porcello**, "Learning React: Modern Patterns for Developing React Apps." O'Reilly Media, Third Edition, 2023.
- [3] **Vasan Subramanian**, "Pro MERN Stack: Full Stack Web App Development with Mongo. Express. React, and Node," Apress, Second Edition, 2019.
- [4] **Robin Wieruch**, "The Road to React: Your Journey to Master Plain yet Pragmatic React.js." Independent Publishing, Fifth Edition, 2022.
- [5] **Andrew Mead & Rob Percival**, "The Complete Node.js Developer Course: Build RESTful APIs with Node, Express, Mongo DB." Udemy, First Edition, 2020.

WEBSITE LINKS

- [1] <https://stackoverflow.com/>
- [2] <https://Tutorialspoint.com/>
- [3] <https://github.com/>
- [4] <https://www.w3schools.com/>
- [5] <https://medium.com/>