**CONTACT MANAGER**

# MINI PROJECT WORK I

**Submitted by**

# DEEPAK D

**22ADR014**

# LAVANYA J

**21CSR138**

# LAVANYA G

**21CSR144**

***in partial fulfilment of the requirements for the award of the degree***

***of***

**BACHELOR OF TECHNOLOGY**

**IN**

**ARTIFICIAL INTELLIGENCE AND DATASCIENCE**

# DEPARTMENT OF ARTIFICIAL INTELLIGENCE AND DATA SECIENCE



**KONGU ENGINEERING COLLEGE**

(**Autonomous)**

**PERUNDURAI ERODE – 638 060**

**MAY 2024**

**DEPARTMENT OF ARTIFICIAL INTELLIGENCE AND DATASCIENCE KONGU ENGINEERING COLLEGE**

**(Autonomous) PERUNDURAI ERODE – 638 060**

**MAY 2024**

# BONAFIDE CERTIFICATE

This is to certify that the Project report entitled **CONTACT MANAGER** is the bonafide record of the project work done by **DEEPAK D (Register No: 22ADR014) , LAVANYA J (Register No: 21CSR138), LAVANYA G (Register**

**No:21CSR144),** in the partial fulfillment of the requirements for the award of the Degree of Bachelor of Technology in **ARTIFICIAL INTELLIGENCE AND DATA SCIENCE** of Anna University Chennai during the year 2023-2024.

**SUPERVISOR HEAD OF THE DEPARTMENT**

**Date: (Signature with seal)**

# ABSTRACT

The project "Contact Manager" focuses on the development of a centralized platform for managing company contacts using the MERN stack, including MongoDB, Express, React, and Node.js. This application enables administrators to efficiently add, update, delete, and retrieve contact details, streamlining internal contact management. Designed with a user-friendly interface, the platform offers an organized view of the company's key contacts, structured across multiple pages: Home, Dashboard, Users List, Employees, Employee Categories (such as Frontend Developer, Backend Developer, Cloud Engineer), and Admin Profile.

With React handling the frontend and MongoDB as the database, the project ensures scalability and real-time data access. Node.js and Express power the backend to deliver secure and responsive interactions. The "Contact Manager" application aims to enhance the efficiency of internal communication, improve data accessibility, and support the company in maintaining well-structured contact records for better administrative and operational flow. Through effective utilization of modern web technologies, the project is poised to deliver a reliable, easy-to-use solution for streamlined contact management, meeting the needs of the organization for long-term data handling and growth.

By implementing authentication and authorization features, the application ensures data security and restricts access to sensitive information based on user roles. Additionally, the platform’s modular design supports future scalability, enabling the seamless addition of features as the company’s contact management needs evolve. Real-time search and filtering functions further enhance usability, allowing administrators to quickly locate and organize contact data. The "Contact Manager" project ultimately seeks to empower organizations with a powerful tool for maintaining and accessing essential contact information effortlessly.

**KONGU ENGINEERING COLLEGE**

(**Autonomous**)

**PERUNDURAI ERODE – 638 060**

**MAY 2024**

**DECLARATION**

We affirm that the Project Report titled **CONTACT MANAGER** being submitted in partial fulfillment of the requirements for the award of Bachelor of Technology is the original work carried out by us. It has not formed the part of any other project report or dissertation on the basis of which a degree or award was conferred on an earlier occasion this or any other candidate.

**Date: DEEPAK D (22ADR014)**

**LAVANYA J**

(**21CSR138)**

**LAVANYAG**

**(21CSR144)**

I certify that the declaration made by the above candidate is true to the best of my knowledge.

Date: Name and Signature of the Supervisor

# ACKNOWLEDGEMENT

We express our sincere thanks and gratitude to **Thiru. A. K. ILANGO B.Com., M.B.A., LLB.,** our beloved Correspondent, and all other philanthropic trust members of Kongu Vellalar Institute of Technology Trust who have always encouraged us in academic and co-curricular activities.

We are extremely thankful with no words of a formal nature to the dynamic Principal **Dr. V. BALUSAMY, M.Tech., Ph.D**., for providing the necessary facilities to complete our work.

We would like to express our sincere gratitude to our respected Head of the Department **Dr. S. MALLIGA, M.E., Ph.D.**, for providing the necessary facilities.

We extend our thanks to **Mr. M. MUTHURAJA, M.E.,** Assistant Professor, Computer Science Engineering, Project Coordinator for his encouragement and valuable advice that made us carry out the project work successfully.

We extend our gratitude to our Supervisor **Mr. M. MUTHURAJA, M.E.,** Assistant Professor of Computer Science Engineering, for his valuable ideas and suggestions, which have been very helpful in the project. We are grateful to all the faculty members of the Computer Science and Engineering Department, for their support.

**TABLE OF CONTENTS**

|  |  |  |
| --- | --- | --- |
| **CHAPTER No.** | **TITLE** | **PAGE No.** |
|  | **ABSTRACT** | **iii** |
|  | **LIST OF FIGURES** | **viii** |
| **1.** | **INTRODUCTION** | **1** |
|  | 1.1 EXISTING SYSTEM | 1 |
|  | 1.2 SYSTEM STUDY | 2 |
|  | 1.3 OBJECTIVE | 2 |
|  | 1.4 SCOPE | 3 |
| **2.** | **GENERAL DESCRIPTION** | **4** |
|  | 2.1 PROJECT PERSPECTIVE | 4 |
|  | 2.2 USER CHARACTERISTICS | 4 |
|  | 2.3 DESIGN AND IMPLEMENTATION |  |
|  | CONSTRAINTS |  |
| **3.** | **REQUIREMENTS** | **6** |
|  | 3.1 FUNCTIONAL REQUIREMENTS | 6 |
|  | 3.2 NON-FUNCTIONAL REQIREMENTS | 7 |
|  | 3.3 USER INTERFACES | 8 |
| **4.** | **DETAILED DESIGN** | **14** |
|  | 4.1 ARCHITECTURAL DESIGN | 14 |
|  | 4.1.1 MODULE CHARACTERISTICS | 14 |
|  | 4.1.1.1 USER MODULE | 14 |
|  | 4.1.1.2 ADMIN MODULE | 14 |

|  |  |  |
| --- | --- | --- |
|  | 4.1.1.2 USE CASE DIAGRAM | 15 |
|  | 4.1.1.3 SEQUENCE DIAGRAM | 16 |
|  | 4.2 INTERFACE DESIGN | 17 |
|  | 4.3 DATABSE DESIGN | 18 |
|  | 4.4 OUTPUT DESIGN | 18 |
| **5.** | **TESTING** | 19 |
|  | 5.1 UNIT TESTING | 19 |
|  | 5.2 REGRESSION TESTING | 19 |
|  | 5.3 VALIDATION TESTING | 19 |
|  | 5.4 VERIFICATION TESTING | 20 |
|  | 5.5 INTEGRATION TESTING | 20 |
| **6.** | **CONCLUSION AND FUTURE SCOPE** | 21 |
|  | APPENDIX 1 | 22 |
|  | APPENDIX 2 | 50 |
|  | REFERENCES | 57 |

**LIST OF FIGURES**

|  |  |  |
| --- | --- | --- |
| **FIGURE NO.** | **TITLE** | **PAGE NO.** |
| 4.1 | FLOW DIAGRAM | 15 |
| 4.2 | USE CASE DIAGRAM | 16 |
| 4.3 | SEQUENCE DIAGRAM | 17 |
| 4.4 | DATABASE DESIGN DIAGRAM | 18 |

# CHAPTER 1 INTRODUCTION

Welcome to "Contact Manager," where organization meets innovation, transforming the way you manage company contacts with ease and efficiency. Step into a digital space designed for seamless interaction, where each feature enhances productivity and brings order to your network. Explore our intuitive platform, crafted to simplify and elevate the contact management experience, enabling you to focus on what truly matters. Immerse yourself in a system that seamlessly organizes, updates, and categorizes contacts, empowering you with quick access and real-time collaboration. Embrace a tool that not only strengthens professional connections but also drives efficiency with every interaction. "Contact Manager" redefines internal communication, offering a reliable and stylish solution that inspires confidence and productivity. Join a community of forward-thinking professionals who value streamlined operations and efficient data handling. With "Contact Manager," transform the way you connect, manage, and grow—creating a network that reflects the future of organized success.

# EXISTING SYSTEM

In the current system, companies often rely on basic methods like spreadsheets or simple CRM tools to manage contact information. These methods can be time-consuming, prone to errors, and lack advanced features for quick access, organization, and security. Without real-time updates or easy ways to categorize and retrieve contacts, managing information becomes inefficient and less reliable. This highlights the need for a more streamlined, user-friendly solution that can improve productivity and maintain accurate, up-to-date contact records for the company.

# SYSTEM STUDY

## Understanding the Business Requirements

The first step in the system study is to understand the business requirements of the "Contact Manager" project. This involves identifying key objectives, understanding the needs of the administrators who will use the system, and defining the essential features for efficient contact management. These requirements include functionalities such as adding, updating, deleting, and organizing contact information securely and effectively.

## Analyzing the Existing Systems and Processes

The next step is to analyze the existing systems and processes currently used for contact management, such as spreadsheets, basic CRM tools, or any other software solutions. This analysis helps to identify limitations, such as inefficiency, lack of real-time updates, or limited data security, which impact productivity and data accuracy. By understanding these challenges, we can pinpoint areas for improvement in the new "Contact Manager" system to ensure it addresses these gaps and offers a more streamlined, user-friendly experience.

## Identifying User Needs

The system study should also involve identifying the needs of the users who will be accessing the "Contact Manager" system. This can be done through surveys, focus groups, or other user research methods. Understanding their expectations and pain points will help tailor the system to meet their specific needs, ensuring a more efficient and user-friendly experience.

## Developing Use Cases

Use cases can be developed to illustrate how the "Contact Manager" system will function and how users will interact with it. These use cases will detail specific scenarios, such as adding a new contact, updating contact information, or categorizing contacts based on roles. By outlining these interactions, we can identify any potential issues or areas for improvement in the design and functionality, ensuring the system meets user needs and operates smoothly.

# OBJECTIVE

The objective of **Contact Manager** is to streamline the process of managing company contacts through a user-friendly, efficient, and secure platform. The system aims to provide administrators with the ability to easily add, update, delete, and categorize contact information, improving data accuracy and accessibility. By offering role-based access, real-time updates, and a centralized contact database, the system will enhance internal communication and support smoother operational workflows. The **Contact Manager** aims to become an essential tool for businesses, helping them manage and maintain a well-organized contact database while boosting productivity and collaboration across teams.

# SCOPE

The scope of **Contact Manager** focuses on providing a centralized platform for managing company contacts efficiently. The system allows administrators to add, update, delete, and categorize contact information securely, ensuring that all relevant data is easily accessible and well-organized. It is designed to support different user roles, ensuring that only authorized personnel can access or modify sensitive data. The platform is built to handle both small and large datasets, offering scalability as the company grows. It supports real-time updates, search functionality, and categorization of contacts based on departments, roles, or any other customizable attributes.

Additionally, the system provides role-based access controls to protect confidential information, allowing different levels of access depending on the user’s role. This ensures that sensitive contact details are only accessible to authorized users. The \*\*Contact Manager\*\* is designed to enhance internal communication, simplify contact retrieval, and enable better collaboration among teams by providing accurate and up-to-date contact information in a central repository. The platform's customizable nature allows businesses to tailor it to their specific needs, with features that can be expanded or adjusted as required.

Furthermore, the system can be accessed globally, offering remote access to company contact details anytime, anywhere. It provides an intuitive user interface that allows quick searches, easy updates, and efficient organization of contacts. The system also ensures data consistency and minimizes the risk of duplication or errors in contact records. The **Contact Manager** is suitable for businesses of all sizes, from small startups to large enterprises, helping to streamline their contact management process and improve productivity. With future scalability in mind, the platform is designed to accommodate additional features as business requirements evolve, ensuring it remains a valuable asset for businesses

in managing their contacts information effectively.

# CHAPTER 2 GENERAL DESCRIPTION

* 1. **PROJECT PERSPECTIVE**

• The need for efficient contact management drives the adoption of a digital solution to improve accessibility and organization.

• The **Contact Manager** system centralizes contacts, overcoming challenges in manual record-keeping and enhancing operational efficiency.

• It provides convenient access to contact details anytime, improving communication and boosting productivity.

# USER CHARACTERISTICS

For the **Contact Manager** system, understanding the needs of both users and administrators is key to designing a solution that meets their requirements. Users, typically company employees or staff, need an easy-to-navigate interface that allows quick access to contact information. They prioritize simple search functionality, easy updates, and secure access to relevant contacts, enabling efficient communication within teams.

Administrators, who manage the system, focus on efficient contact organization, real-time updates, and data security. They require tools for managing user roles, tracking changes, and ensuring the accuracy of contact details. Additionally, administrators need customizable dashboards, user management capabilities, and the ability to organize contacts based on specific criteria, like department or role. These user and admin characteristics help shape a functional, intuitive contact management system that supports both daily use and administrative tasks.

# DESIGN AND IMPLEMENTATION CONSTRAINTS

## Time

## The Contact Manager project must be completed within a strict timeline to ensure timely deployment and avoid disruptions in company operations. Delays could impact the efficiency of contact management and communication processes.

## Budget Limitations

Budget constraints may limit the resources available for development, technology procurement, and integrations. This could influence the selection of features, tools, and third-party services for the **Contact Manager** system.

## Security

The **Contact Manager** system must comply with industry standards for data security and privacy, ensuring safe handling of sensitive contact information. Any specific security requirements or compliance mandates could pose implementation constraints and impact system design.

## Scalability

The **Contact Manager** system must be designed to scale efficiently as the company grows, accommodating increased users and contact data. Scalability constraints could arise if the system architecture cannot handle higher volumes of data or users.

# CHAPTER 3 REQUIREMENTS

* 1. **FUNCTIONAL REQUIREMENTS**

## User Authentication

Users should be able to register, log in, and log out securely to access their accounts and manage contact information. They should have the ability to update their details, search for specific contacts, and categorize contacts based on roles or departments.

## Contact Catalog

The system should display a comprehensive catalog of company contacts, including names, roles, departments, and contact details. Users should be able to search, filter, and sort contacts based on various criteria such as department, role, or contact type.

## Contact management

Users should be able to add new contacts, view contact details, update information, and categorize contacts. The system should retain the added or updated contacts even if the user navigates away from the page, ensuring seamless access and management.

## Contact Importing

The system should provide an easy and secure process for importing contact data. Users should be able to add, update, and export contact information in bulk, ensuring that data management is efficient and secure.

## Access Control

The system should integrate role-based access control to ensure that only authorized users can access or modify certain contact details. Admins should have full access, while regular users may have limited access depending on their role or permissions..

# NON-FUNCTIONAL REQUIREMENTS

## Performance

The **Contact Manager** system should perform efficiently, loading quickly and responding promptly to user actions, even when handling large volumes of contact data. The system should be optimized for fast search, updates, and data retrieval to ensure a smooth user experience.

## Usability

The **Contact Manager** system should have an intuitive and user-friendly interface, with clear navigation for easy access to contacts, updates, and search functionality. The design should be simple yet effective, allowing users to manage and retrieve contact information with minimal effort and confusion.

## Reliability

The **Contact Manager** system should be reliable and stable, with strong error handling and backup mechanisms to prevent data loss or corruption. The system should ensure that contact information remains intact and accessible, even in the event of system failures or unexpected errors.

## Availability

The Contact Manager system should be available at all times with minimal downtime. It should have failover mechanisms in place to ensure users can always access contact information without interruptions.

# USER INTERFACES

**HOME PAGE:**

The home page links to various pages like Sign Up and Login Pages.



FIGURE 3.3.0

**LOGIN PAGE:**

A login page typically contains fields for entering a username, password, a login button, and links for password recovery or account registration.

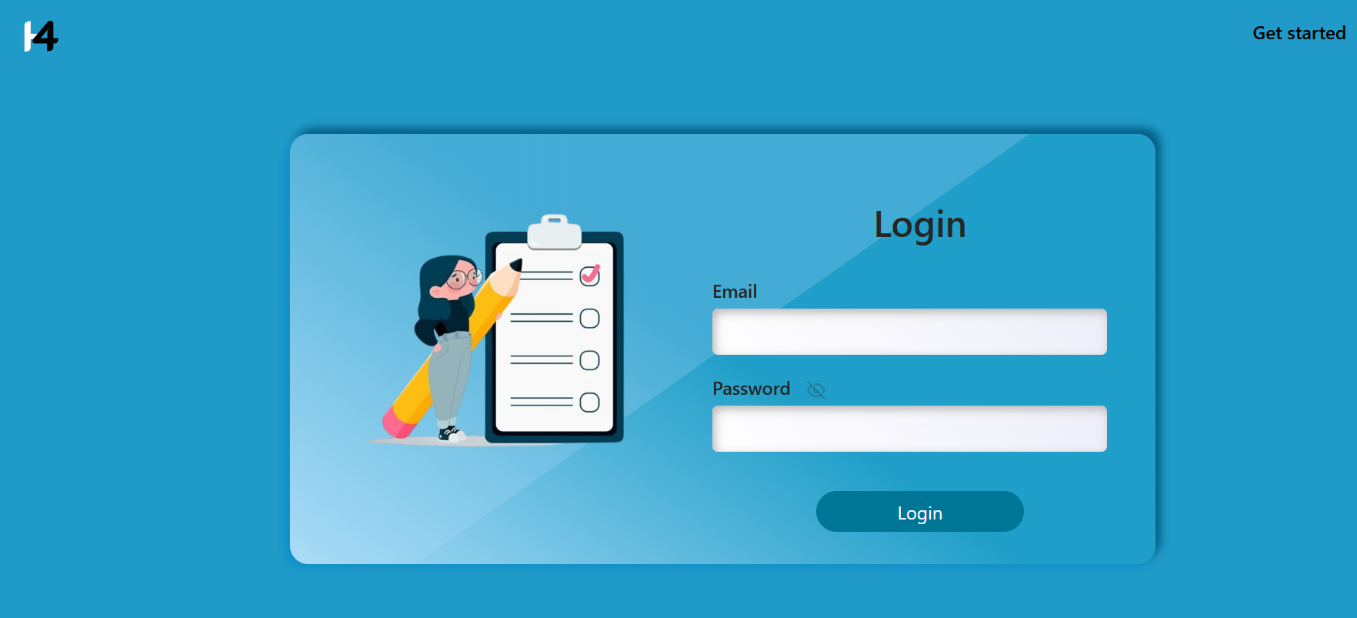


FIGURE 3.3.1

**SIGN UP PAGE:**

A sign-up page typically contains fields for entering an email, username, password, and confirm password, along with a submit button for registration.

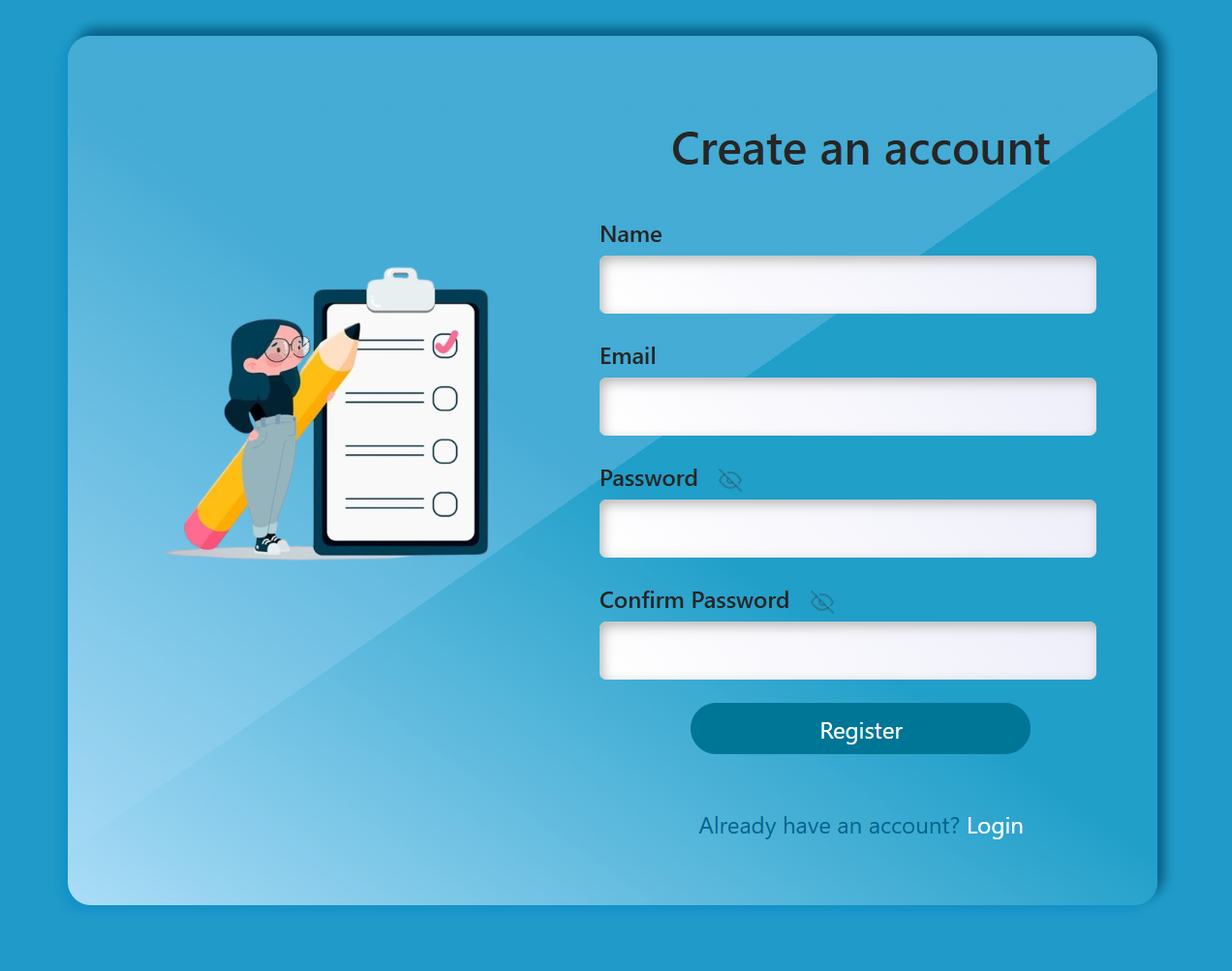


FIGURE 3.3.2

**CONTACT LIST PAGE**:

In Contact Page it contains the list of contacts with their email ,mobile, address, category containing the edit and delete options.

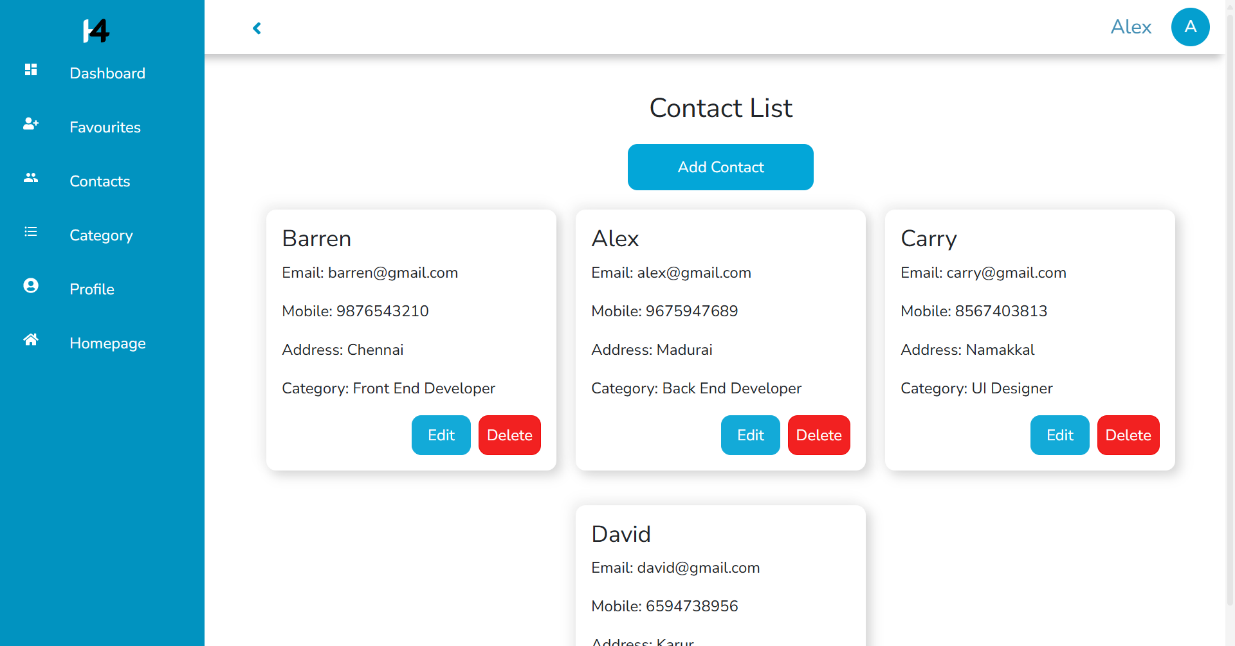


FIGURE 3.3.3

**CATEGORY LIST PAGE**:

The category list page shows the type / category of the contact .

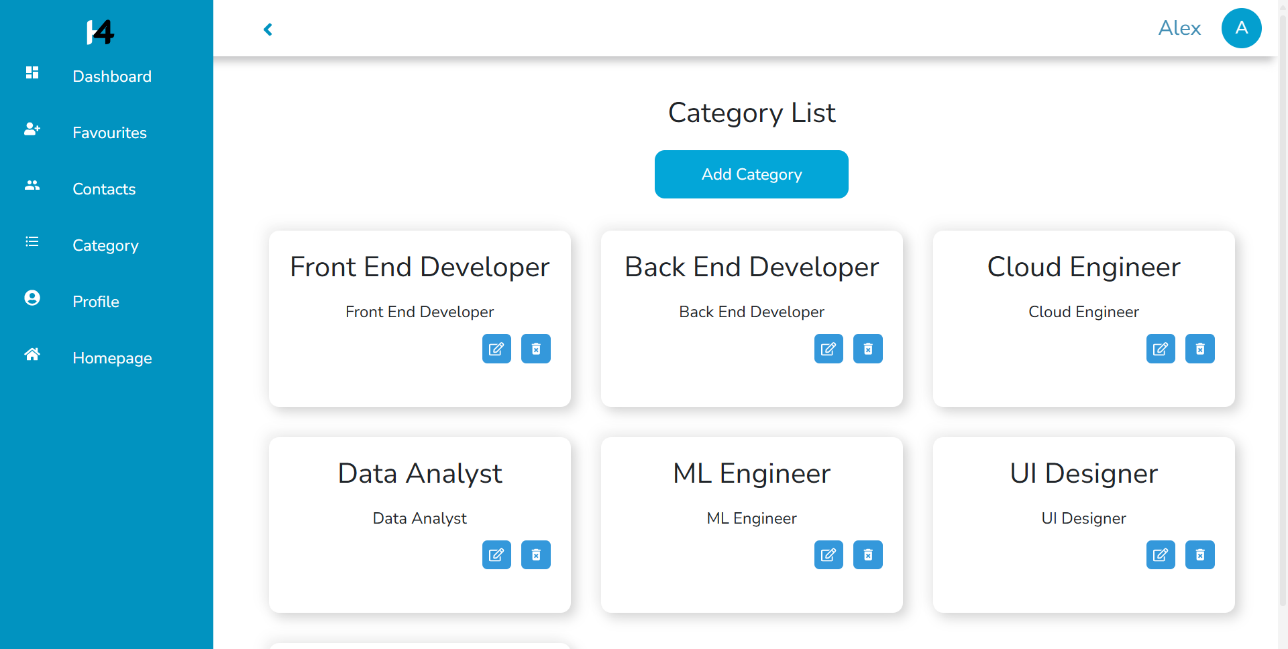


FIGURE 3.3.4

**SIDE NAVBAR PAGE**:

The side navbar container the Dashboard , Favourites , Contacts , Category , Profile , HomePage.

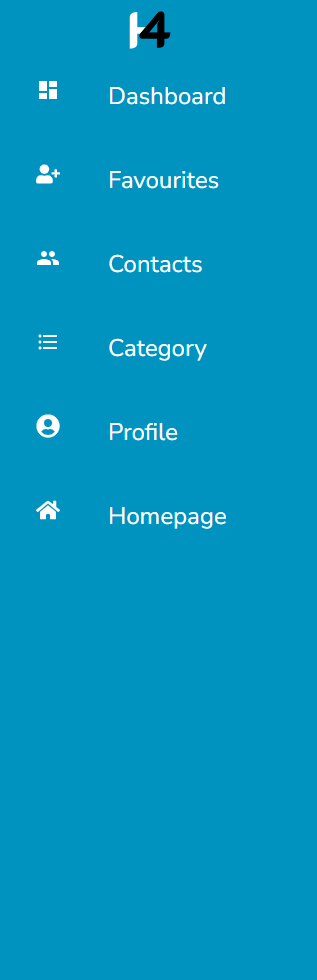


FIGURE 3.3.5

**DASHBOARD :**

The Dashboard contains the Admin Name , Total Number of Contacts , List of the Contacts with their Name , Email , Mobile , Address , About.

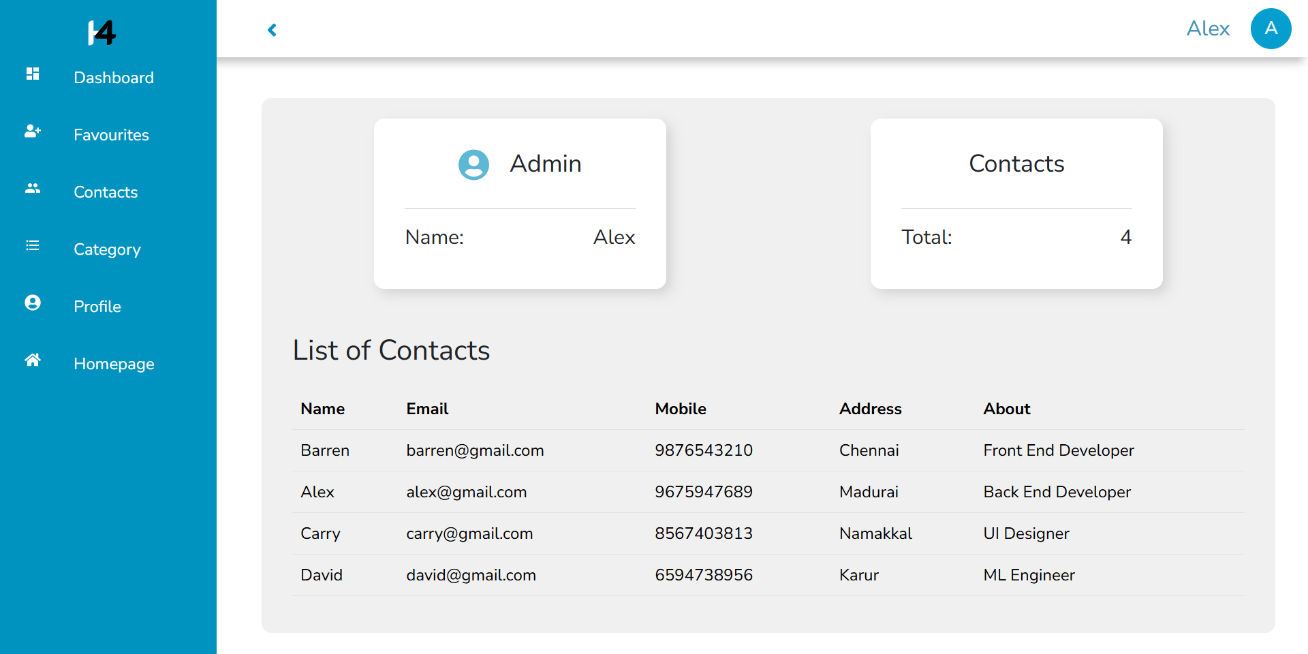


FIGURE 3.3.6

**PROFILE PAGE:**

The profile page shows the user name , email .

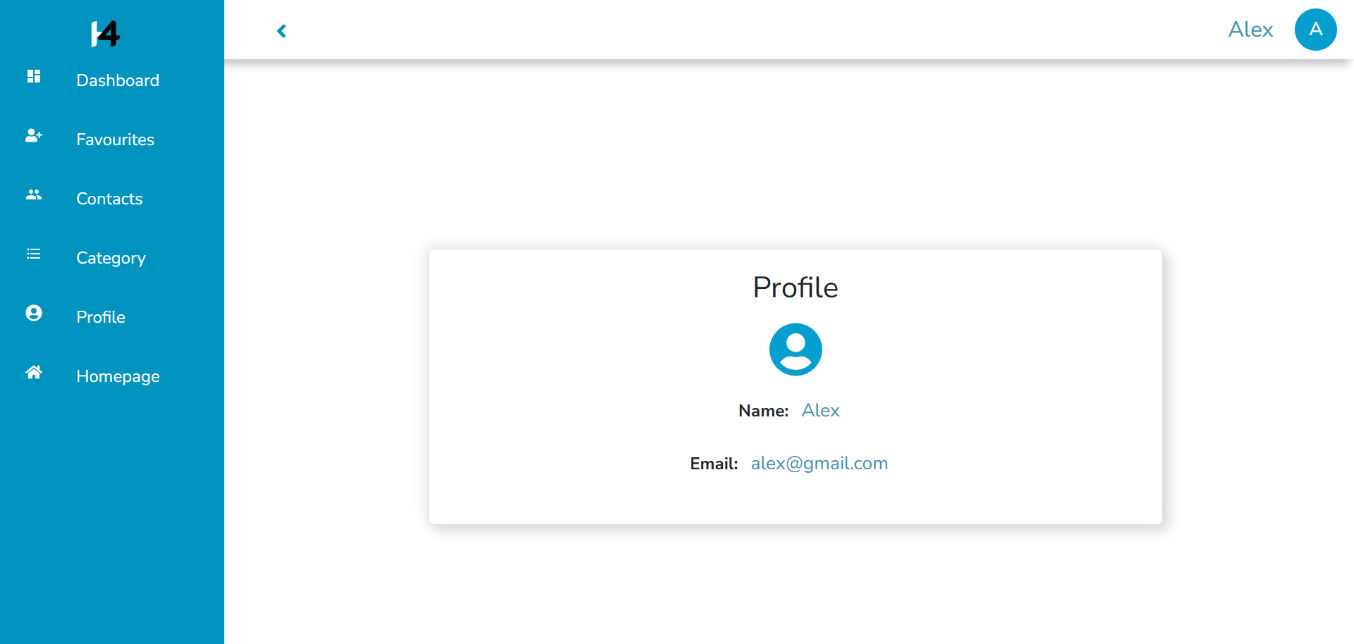


FIGURE 3.3.7

**FAVOURITES PAGE:**

In Favourites , list page there the user can add the their favourite contacts with their mail only.

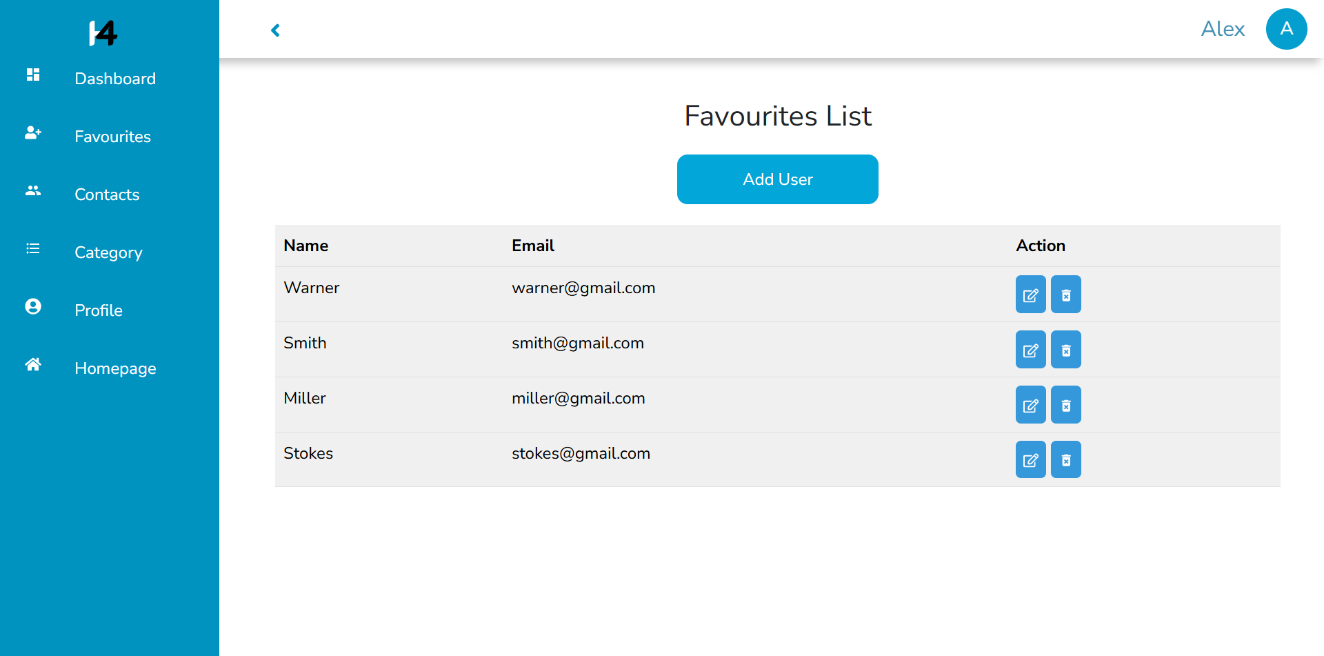


FIGURE 3.3.8

# FOOTER:

The footer contains the some details .



FIGURE 3.3.9

# CHAPTER 4 DETAILED DESIGN

* 1. **ARCHITECTURAL DESIGN**

The architectural design of the MERN Contact Manager includes use case, sequence, and activity diagrams to illustrate user and admin interactions, data flow, and system processes. These diagrams define how users manage contacts and interact with the system for CRUD operations.

# MODULE CHARACTERISTICS

The project contains one modules as follows and their description is given

below.

* User module
* Admin module

## User Module

The User Module in the Contact Manager includes functionalities for viewing and managing company contacts. Users can add, update, delete, and search for contacts, ensuring effective management of employee details within the application. FIGURE 4.1

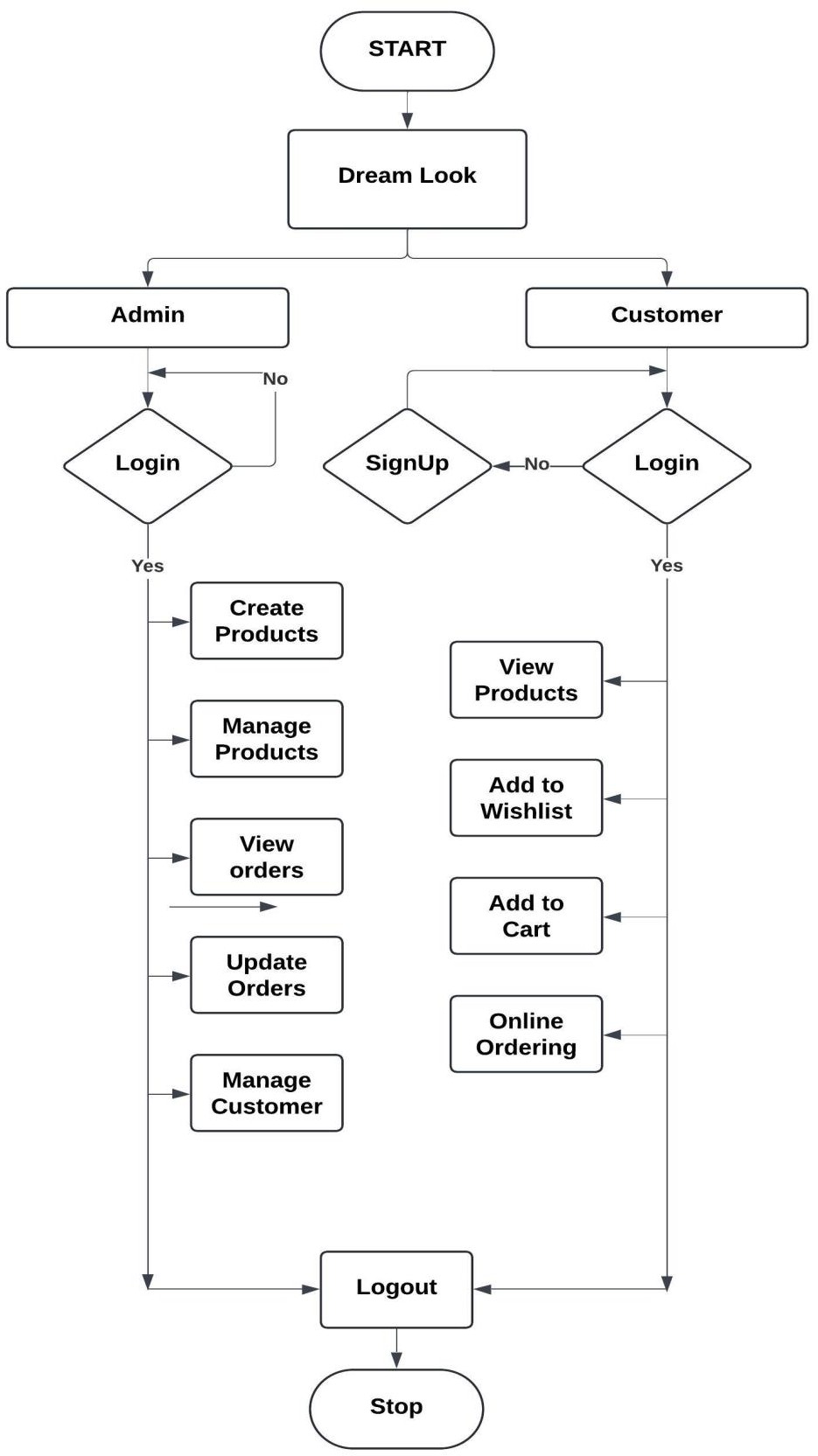


FIGURE 4.1 FLOWCHART

* + - 1. **USE CASE DIAGRAM**

A use case diagram is a dynamic or behaviour diagram in UML. Use case diagrams model the functionality of a system using actors and use cases. Use cases are set of actions, services, and functions that the system needs to perform. A use case diagram at its simplest is a representation of a user's interaction with the system that shows the relationship between the user and the different use cases in which the user is involved. The use cases are represented by either circles or ellipses. Due to their simplistic nature, use case diagrams can be a good communication tool for stakeholders. FIGURE 4.2

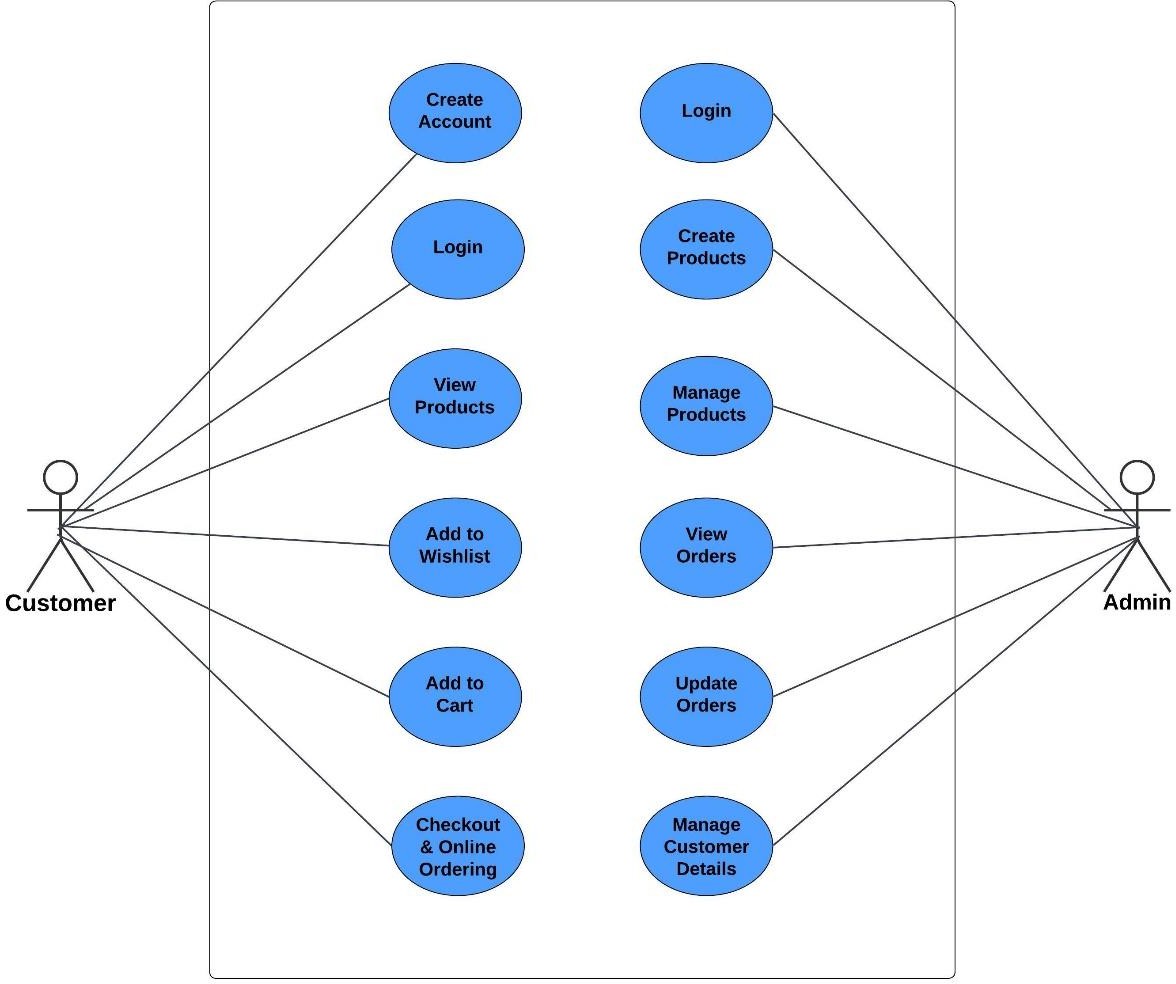


FIGURE 4.2 USE CASE DIAGRAM

* + - 1. **SEQUENCE DIAGRAM**

Sequence chart diagram is one of the five UML diagrams used to model the dynamic nature of a system. The admin defines the sequence of an object during its lifetime. FIGURE 4.3

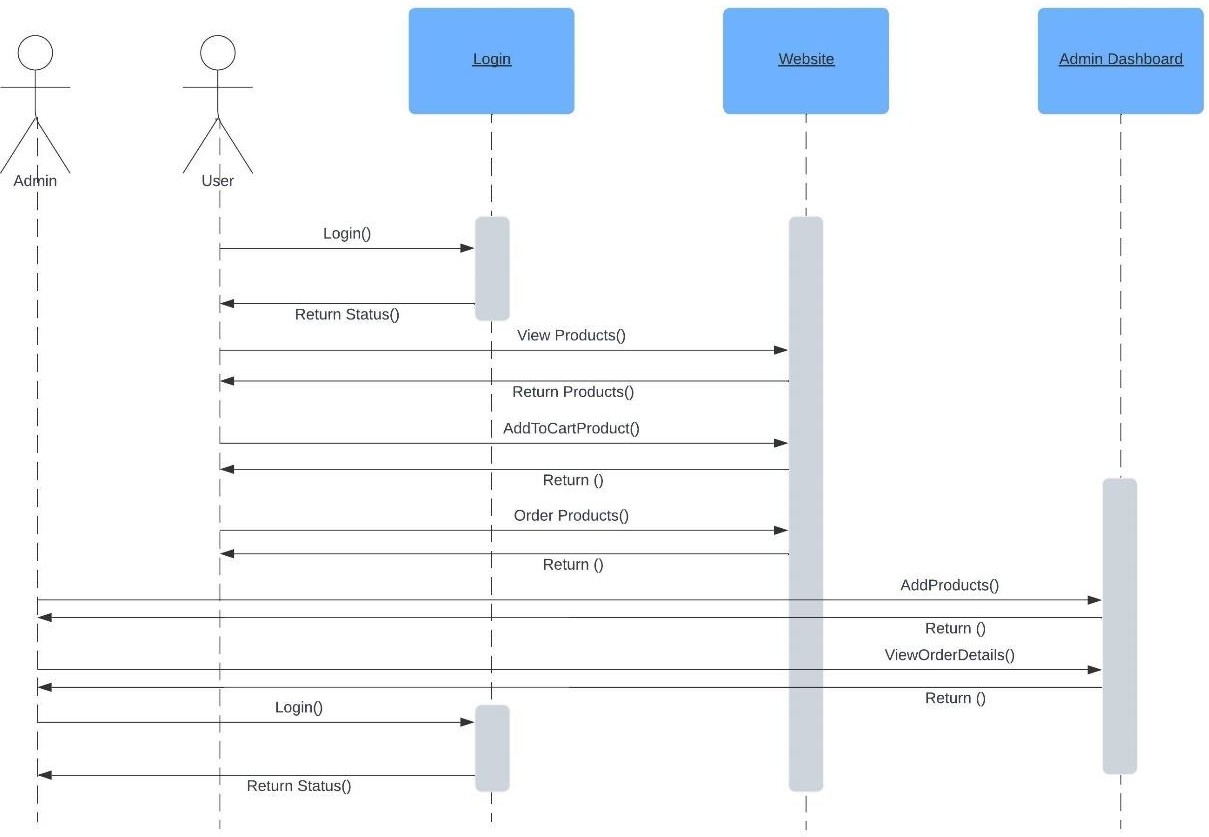


FIGURE 4.3 SEQUENCE DIAGRAM

* 1. **INTERFACE DESIGN**

The "Contact Manager" application features a user-friendly and responsive interface, providing easy access to essential pages such as Home, Dashboard, Users List, Employees, and Admin Profile. The interface ensures clarity and consistency across all screens, allowing admins to efficiently manage contacts and employee categories. It is designed to be visually appealing, simple to use, and responsive across devices.

* 1. **DATABASE DESIGN**

The database used in the "Contact Manager" system is MongoDB, where data is stored in collections and documents. Collections include *contacts* to manage company contacts, *users* for storing admin and user information, and *feedback* to store customer feedback. Customer details such as name, email, and phone number are stored in MongoDB, allowing efficient retrieval, update, and management of contact information.

FIGURE 4.4

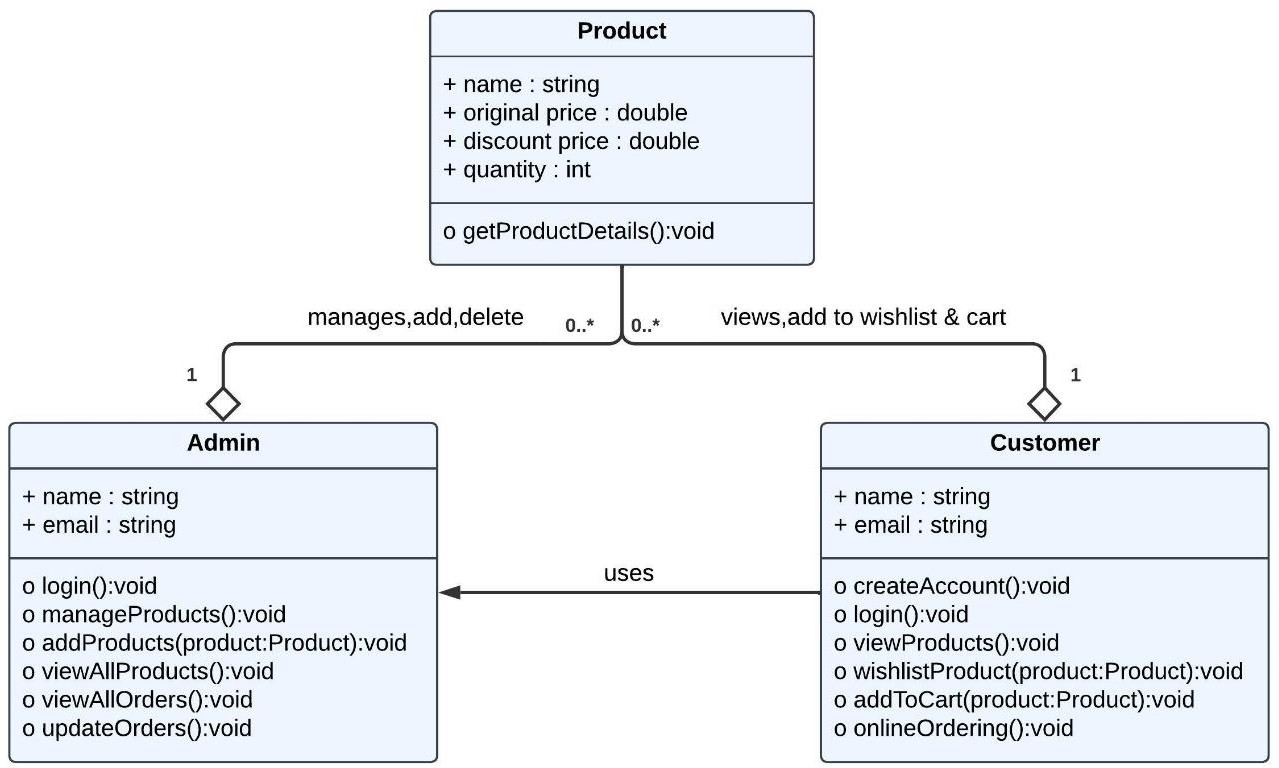


FIGURE 4.4

* 1. **OUTPUT DESIGN**

Output design generally refers to the results and information that are generated by the system for many end-users; it should be understandable with the enhanced format. Previewing the output reports by the user is extremely important because the user is the ultimate judge of the quality of the output and, in turn, the success of the system. The output is designed in such a way that it is attractive, convenient and informative.

# CHAPTER 5 TESTING

* 1. **UNIT TESTING**

In the "Contact Manager" application, unit testing involves testing individual components such as contact management, user authentication, and admin functionalities to ensure each part operates as expected. Automated unit tests are conducted on modules like adding, updating, deleting contacts, managing user roles, and handling feedback. Once each module meets the required functionality, integration testing is performed to ensure seamless interaction between multiple modules, ensuring the system works as a whole.

# REGRESSION TESTING

In the "Contact Manager" application, regression testing is performed to identify any unintended issues or side effects resulting from code updates, dependency changes, or modifications in the infrastructure. By rerunning previously executed test cases, the development team ensures that core functionalities, such as contact management, user roles, and feedback handling, remain unaffected by recent changes. The main goal of regression testing is to ensure the system remains reliable, functional, and user-friendly, maintaining its integrity and performance throughout the development lifecycle and meeting the needs of both users and stakeholders.

# VALIDATION TESTING

In the "Contact Manager" application, validation testing is carried out to ensure that the software meets the specified business requirements and satisfies the needs of the client. This process involves evaluating the system during or at the end of the development phase to confirm that it fulfills its intended purpose, such as managing contacts, handling feedback, and managing admin roles. Validation testing ensures that the application performs correctly in the target environment and aligns with the expectations of both users and stakeholders.

# VERIFICATION TESTING

In the "Contact Manager" application, verification testing is conducted to evaluate the work and service of each development phase to ensure they meet the specified requirements. Each module, such as contact management, user authentication, and feedback handling, is checked during its development stage to confirm that it aligns with the predefined requirements. Verification ensures that individual modules function as expected before moving on to subsequent stages, maintaining the overall integrity of the application.

# INTEGRATION TESTING

In the "Contact Manager" application, integration testing is performed to verify that the integrated software components work together as expected. It focuses on identifying issues that may arise when combining different modules, such as the user module, contact management, and admin functionalities. After integrating these modules with the core service, integration testing ensures that the system operates smoothly as a whole, with data flowing correctly between components and no unintended conflicts or errors occurring.

# CHAPTER 6 CONCLUSION AND FUTURE WORK

The development of the "Contact Manager" application marks a significant step towards improving contact management and streamlining communication within a company. By utilizing the MERN stack (MongoDB, Express, React, and Node.js), the project provides an efficient platform for managing contacts, employee categories, and user roles with ease. The user-friendly interface, secure data handling, and seamless admin functionality ensure a smooth experience for both admins and users.

Looking ahead, further improvements will focus on enhancing the application's scalability, security, and user experience. Future work includes integrating advanced search and filtering capabilities, implementing role-based access control for more granular permissions, and improving data analytics to support better decision-making. Additionally, efforts will be made to optimize the system for mobile devices, explore integration with other services, and introduce new features to maintain competitiveness in the evolving digital landscape.

.

# APPENDIX 1

**CODING:**

## App.js:

import React, { useEffect, useState } from "react"; import "./App.css";

import { BrowserRouter, Routes, Route } from "react-router-dom"; import Feedback from "./pages/Feedback.jsx";

import ProtectedRoute from "./routes/ProtectedRoute"; useEffect(() => {

Store.dispatch(loadUser()); Store.dispatch(loadSeller()); Store.dispatch(getAllProducts()); Store.dispatch(getAllEvents()); getStripeApikey();

}, []);

return (

<BrowserRouter>

{stripeApikey && (

<Elements stripe={loadStripe(stripeApikey)}>

<Routes>

<Route path="/payment" element={

<ProtectedRoute>

<PaymentPage />

</ProtectedRoute>

}

/>

</Routes>

</Elements>

)}

<Routes>

<Route path="/" element={<HomePage />} />

<Route path="/login" element={<LoginPage />} />

<Route path="/sign-up" element={<SignupPage />} />

<Route path="/activation/:activation\_token" element={<ActivationPage />}

/>

<Route path="/seller/activation/:activation\_token" element={<SellerActivationPage />}

/>

<Route path="/products" element={<ProductsPage />} />

<Route path="/product/:id" element={<ProductDetailsPage />} />

<Route path="/checkout" element={

<ProtectedRoute>

<CheckoutPage />

</ProtectedRoute>

}

/>

<Route path="/shop/preview/:id" element={<ShopPreviewPage />} />

{/\* shop Routes \*/}

<Route path="/shop-create" element={<ShopCreatePage />} />

<Route path="/admin" element={<ShopLoginPage />} />

export default App;;

## Header:

import React, { useState } from "react"; import { Link } from "react-router-dom"; import styles from "../../styles/styles";

import { categoriesData, productData } from "../../static/data"; import {

AiOutlineHeart, AiOutlineSearch, AiOutlineShoppingCart,

} from "react-icons/ai";

import { IoIosArrowDown, IoIosArrowForward } from "react-icons/io"; import { BiMenuAltLeft } from "react-icons/bi";

import { CgProfile } from "react-icons/cg"; import DropDown from "./DropDown"; import Navbar from "./Navbar";

import { useSelector } from "react-redux"; import Cart from "../cart/Cart";

import Wishlist from "../Wishlist/Wishlist"; import { RxCross1 } from "react-icons/rx";

return (

<>

<div className={`${styles.section}`}>

<div className="hidden 1000px:h-[40px] 800px:my-[20px] 800px:flex items-center justify-between">

<div>

</div>

</div>

</div>

<div className={`${

active === true ? "shadow-sm fixed top-0 left-0 z-30" : null

} transition hidden 800px:flex items-center justify-between w-full bg-[#023e8a] h- [70px]`}

>

</div>

export default Header;

## Navbar:

import React from "react";

import { Link } from "react-router-dom"; import { navItems } from "../../static/data"; import styles from "../../styles/styles";

const Navbar = ({ active }) => { return (

<div className={`block 800px:${styles.noramlFlex}`}>

{navItems && navItems.map((i, index) => (

<div className="flex">

<Link

to={i.url} className={`${ active === index + 1

? "text-[#fff]"

: "text-black 800px:text-[#fff]"

} pb-[30px] 800px:pb-0 font-[500] px-6 cursor-pointer}`}

>

{i.title}

</Link>

</div>

))}

</div>

);

};

export default Navbar;

## Footer:

import React from "react";

import { useHistory } from "react-router-dom"; import {

AiFillFacebook, AiFillInstagram, AiFillYoutube, AiOutlineTwitter,

} from "react-icons/ai";

import { Link } from "react-router-dom"; import {

footercompanyLinks, footerProductLinks, footerSupportLinks,

} from "../../static/data";

const Footer = () => { return (

<ul className="text-center sm:text-start">

<h1 className="mb-1 font-semibold">Shop</h1>

{footercompanyLinks.map((link, index) => (

<li key={index}>

<Link

className="text-gray-400 hover:text-teal-400 duration-300 text-sm cursor-pointer leading-6"

to={link.link}

>

{link.name}

</Link>

</li>

))}

</ul>

</div>

<div

className="grid grid-cols-1 sm:grid-cols-2 gap-20 text-center pt-2 text-gray-400 text-sm pb-8"

>

<span>© 2024 . DreamLook . All rights reserved.</span>

<span>Terms · Privacy Policy</span>

</div>

</div>

);

};

export default Footer;

## Contact:

import React, { useState } from "react";

import Footer from "../components/Layout/Footer"; import Header from "../components/Layout/Header"; import styles from "../styles/styles";

import { Link } from "react-router-dom";

const ContactPage = () => { return (

<div>

<Header />

<Contact />

<Footer />

</div>

);

};

const Contact = () => {

const [activeTab, setActiveTab] = useState(0);

const toggleTab = (tab) => {

if (activeTab === tab) { setActiveTab(0);

} else { setActiveTab(tab);

}

};

return (

<div class="mt-0 px-4">

<h1 class="text-2xl font-bold mb-4 ml-20">Phone Number : </h1>

<p class="mb-4 ml-24">+91 99650-88654</p>

<br />

<h1 class="text-2xl font-bold mb-4 ml-20">Email -ID : </h1>

<p class="mb-4 ml-2[4">kuma](mailto:kumardreamlook1@gmail.com)rdr[eamlook1@gmail.com<](mailto:kumardreamlook1@gmail.com)/p>

<br />

<h1 class="text-2xl font-bold mb-4 ml-20">Address : </h1>

<p class="mb-4 ml-24">

204,Uzhavu Vanika Valagam,Anna Nagar,Arachalur,Erode - 638101

</p>

<br />

<br />

<br />

</div>

);

};

export default ContactPage;

## Database:

const mongoose = require("mongoose"); const connectDatabase = () => { mongoose

.connect(process.env.DB\_URL, { useNewUrlParser: true, useUnifiedTopology: true,

})

.then((data) => {

console.log(`mongod connected with server: ${data.connection.host}`);

});

};

module.exports = connectDatabase;

## AdminDashboard:

import React, { useEffect, useState } from "react"; import styles from "../../styles/styles";

const AdminDashboardMain = () => { const dispatch = useDispatch();

const { adminOrders,adminOrderLoading } = useSelector((state) => state.order); const { sellers } = useSelector((state) => state.seller);

useEffect(() => { dispatch(getAllOrdersOfAdmin()); dispatch(getAllSellers());

}, []);

const adminEarning = adminOrders && adminOrders.reduce((acc,item) => acc + item.totalPrice \* .10, 0);

const adminBalance = adminEarning?.toFixed(2); const row = [];

return (

<>

{

adminOrderLoading ? (

<Loader />

) :

<h5 className="pt-2 pl-[36px] text-[22px] font-[500]">$ {adminBalance}</h5>

</div>

<div className="w-full mb-4 800px:w-[30%] min-h-[20vh] bg-white shadow rounded px-2 py-5">

<div className="flex items-center">

<MdBorderClear size={30} className="mr-2" fill="#00000085" />

<h3

className={`${styles.productTitle} !text-[18px] leading-5 !font-[400] text- [#00000085]`}

>

All Sellers

</h3>

</div>

</Link>

</div>

<div className="w-full mb-4 800px:w-[30%] min-h-[20vh] bg-white shadow rounded px-2 py-5">

<div className="flex items-center">

<AiOutlineMoneyCollect size={30} className="mr-2" fill="#00000085"

/>

<h3

className={`${styles.productTitle} !text-[18px] leading-5 !font-[400] text- [#00000085]`}

>

All Orders

</h3>

</div>

<h5 className="pt-2 pl-[36px] text-[22px] font-[500]">{adminOrders && adminOrders.length}</h5>

<Link to="/admin-orders">

<h5 className="pt-4 pl-2 text-[#077f9c]">View Orders</h5>

</Link>

</div>

</div>

<br />

export default AdminDashboardMain;

## Checkout:

import React, { useState } from "react"; import styles from "../../styles/styles";

import { Country, State } from "country-state-city"; import { useNavigate } from "react-router-dom"; import { useSelector } from "react-redux";

import { useEffect } from "react"; import axios from "axios";

import { server } from "../../server"; import { toast } from "react-toastify";

const paymentSubmit = () => { if (

address1 === "" || address2 === "" || zipCode === null || country === "" || city === ""

) {

toast.error("Please choose your delivery address!");

} else {

const shippingAddress = { address1,

address2, zipCode, country, city,

};

const orderData = { cart,

totalPrice,

subTotalPrice, shipping, discountPrice, shippingAddress, user,

};

// update local storage with the updated orders array localStorage.setItem("latestOrder", JSON.stringify(orderData)); navigate("/payment");

}

};

const subTotalPrice = cart.reduce(

(acc, item) => acc + item.qty \* item.discountPrice, 0

);

// this is shipping cost variable

const shipping = subTotalPrice \* 0.1; const handleSubmit = async (e) => { e.preventDefault();

const name = couponCode;

await axios.get(`${server}/coupon/get-coupon-value/${name}`).then((res) => { const shopId = res.data.couponCode?.shopId;

const couponCodeValue = res.data.couponCode?.value; if (res.data.couponCode !== null) {

const isCouponValid =

cart && cart.filter((item) => item.shopId === shopId); return (

<div className="w-full flex flex-col items-center py-8">

<div className="w-[90%] 1000px:w-[70%] block 800px:flex">

<div className="w-full 800px:w-[65%]">

<ShippingInfo user={user} country={country} setCountry={setCountry} city={city}

/>

</div>

</div>

<div

className={`${styles.button} w-[150px] 800px:w-[280px] mt-10`} onClick={paymentSubmit}

>

<h5 className="text-white">Go to Payment</h5>

</div>

</div>

);

};

return (

<div className="w-full 800px:w-[95%] bg-white rounded-md p-5 pb-8">

<h5 className="text-[18px] font-[500]">Shipping Address</h5>

<br />

<div className="w-full flex pb-3">

<div className="w-[50%]">

<label className="block pb-2">Country</label>

<select

className="w-[95%] border h-[40px] rounded-[5px]"

value={country}

onChange={(e) => setCountry(e.target.value)}

>

))}

</select>

</div>

<div className="w-[50%]">

<label className="block pb-2">City</label>

<select

className="w-[95%] border h-[40px] rounded-[5px]" value={city}

onChange={(e) => setCity(e.target.value)}

>

<option className="block pb-2" value=""> Choose your City

</option>

{State && State.getStatesOfCountry(country).map((item) => (

<option key={item.isoCode} value={item.isoCode}>

{item.name}

</option>

))}

</select>

</div>

</div>

<div className="w-full flex pb-3">

<div className="w-[50%]">

<label className="block pb-2">Address1</label>

<input type="address" required value={address1}

onChange={(e) => setAddress1(e.target.value)} className={`${styles.input} !w-[95%]`}

/>

</div>

</div>

<div></div>

</form>

<h5

className="text-[18px] cursor-pointer inline-block" onClick={() => setUserInfo(!userInfo)}

>

Choose From saved address

</h5>

</div>

);

};

return (

<div className="w-full bg-[#fff] rounded-md p-5 pb-8">

<div className="flex justify-between">

<h3 className="text-[16px] font-[400] text-[#000000a4]">subtotal:</h3>

<h5 className="text-[18px] font-[600]">Rs.{subTotalPrice}</h5>

</div>

<br />

<div className="flex justify-between">

<h3 className="text-[16px] font-[400] text-[#000000a4]">shipping:</h3>

<h5 className="text-[18px] font-[600]">Rs.{shipping.toFixed(2)}</h5>

</div>

</div>

);

};

export default Checkout;

## Payment:

import React, { useState } from "react";

import { useNavigate } from "react-router-dom"; import styles from "../../styles/styles";

import { useEffect } from "react"; import {

CardNumberElement, CardCvcElement, CardExpiryElement, useStripe, useElements,

} from "@stripe/react-stripe-js";

import { PayPalScriptProvider, PayPalButtons } from "@paypal/react-paypal-js"; import { useSelector } from "react-redux";

import axios from "axios";

import { server } from "../../server"; import { toast } from "react-toastify";

import { RxCross1 } from "react-icons/rx";

const Payment = () => {

const [orderData, setOrderData] = useState([]); const [open, setOpen] = useState(false);

const { user } = useSelector((state) => state.user); const navigate = useNavigate();

const stripe = useStripe();

const elements = useElements();

useEffect(() => {

const orderData = JSON.parse(localStorage.getItem("latestOrder")); setOrderData(orderData);

}, []);

const onApprove = async (data, actions) => {

return actions.order.capture().then(function (details) { const { payer } = details;

let paymentInfo = payer;

if (paymentInfo !== undefined) { paypalPaymentHandler(paymentInfo);

}

});

};

const paypalPaymentHandler = async (paymentInfo) => { const config = {

headers: {

"Content-Type": "application/json",

},

};

order.paymentInfo = {

id: paymentInfo.payer\_id, status: "succeeded",

type: "Paypal",

};

};

const paymentData = {

amount: Math.round(orderData?.totalPrice \* 100),

};

const paymentHandler = async (e) => { e.preventDefault();

try {

const config = { headers: {

"Content-Type": "application/json",

},

};

const { data } = await axios.post(

`${server}/payment/process`, paymentData,

config

);

const client\_secret = data.client\_secret; if (!stripe || !elements) return;

const result = await stripe.confirmCardPayment(client\_secret, { payment\_method: {

card: elements.getElement(CardNumberElement),

},

});

if (result.error) { toast.error(result.error.message);

} else {

if (result.paymentIntent.status === "succeeded") { order.paymnentInfo = {

id: result.paymentIntent.id,

status: result.paymentIntent.status, type: "Credit Card",

};

const cashOnDeliveryHandler = async (e) => { e.preventDefault();

const config = { headers: {

"Content-Type": "application/json",

},

};

order.paymentInfo = {

type: "Cash On Delivery",

};

await axios

.post(`${server}/order/create-order`, order, config)

.then((res) => { setOpen(false); navigate("/order/success");

toast.success("Order successful!"); localStorage.setItem("cartItems", JSON.stringify([])); localStorage.setItem("latestOrder", JSON.stringify([])); window.location.reload();

});

};

return (

<div className="w-full flex flex-col items-center py-8">

<div className="w-[90%] 1000px:w-[70%] block 800px:flex">

<div className="w-full 800px:w-[35%] 800px:mt-0 mt-8">

<CartData orderData={orderData} />

</div>

<div className="w-full flex pb-3">

<div className="w-[50%]">

<label className="block pb-2">Card Number</label>

<CardNumberElement

className={`${styles.input} !h-[35px] !w-[95%]`}

</div>

<div className="w-[50%]">

<label className="block pb-2">CVV</label>

<CardCvcElement className={`${styles.input} !h-[35px]`} options={{

style: { base: {

fontSize: "19px", lineHeight: 1.5,

color: "#444",

},

</form>

</div>

) : null}

</div>

<br />

{/\* paypal payment \*/}

<div>

<div className="flex w-full pb-5 border-b mb-2">

<div

className="w-[25px] h-[25px] rounded-full bg-transparent border-[3px] border- [#1d1a1ab4] relative flex items-center justify-center"

onClick={() => setSelect(2)}

>

{select === 2 ? (

<div className="w-[13px] h-[13px] bg-[#1d1a1acb] rounded-full" />

) : null}

</div>

<h4 className="text-[18px] pl-2 font-[600] text-[#000000b1]"> Pay with Paypal

</h4>

</div>

{/\* pay with payement \*/}

{select === 2 ? (

<div className="w-full flex border-b">

<div

className={`${styles.button} !bg-[#f63b60] text-white h-[45px] rounded-[5px] cursor-pointer text-[18px] font-[600]`}

onClick={() => setOpen(true)}

>

Pay Now

</div>

{open && (

<PayPalScriptProvider

options={{ "client-id":

"Aczac4Ry9\_QA1t4c7TKH9UusH3RTe6onyICPoCToHG10kjlNdI- qwobbW9JAHzaRQwFMn2-k660853jn",

}}

>

<PayPalButtons

style={{ layout: "vertical" }} onApprove={onApprove} createOrder={createOrder}

/>

</PayPalScriptProvider>

</div>

</div>

)}

</div>

) : null}

</div>

<br />

{/\* cash on delivery \*/}

{select === 3 ? (

<div className="w-full flex">

<form className="w-full" onSubmit={cashOnDeliveryHandler}>

<input type="submit" value="Confirm"

className={`${styles.button} !bg-[#f63b60] text-[#fff] h-[45px] rounded-[5px] cursor-pointer text-[18px] font-[600]`}

/>

</form>

</div>

) : null}

</div>

</div>

);

};

const CartData = ({ orderData }) => {

const shipping = orderData?.shipping?.toFixed(2); return (

<div className="flex justify-between border-b pb-3">

<h3 className="text-[16px] font-[400] text-[#000000a4]">Discount:</h3>

<h5 className="text-[18px] font-[600]">

{orderData?.discountPrice ? "Rs." + orderData.discountPrice : "-"}

</h5>

);

};

export default Payment;

## Route.js:

import LoginPage from "../pages/Login"; import SignupPage from "../pages/SignupPage";

import ActivationPage from "../pages/ActivationPage"; import HomePage from "../pages/HomePage";

import ProductsPage from "../pages/ProductsPage"; import BestSellingPage from "../pages/BestSellingPage"; import ShopCreatePage from "../pages/ShopCreate";

import SellerActivationPage from "../pages/SellerActivationPage"; import ShopLoginPage from "../pages/ShopLoginPage";

import OrderDetailsPage from "../pages/OrderDetailsPage"; import TrackOrderPage from "../pages/TrackOrderPage"; import Feedback from "../pages/Feedback.jsx";

export { LoginPage, SignupPage, ActivationPage,

HomePage, ProductsPage, BestSellingPage, EventsPage, ContactPage, FAQPage, AboutPage,

};

## Data.js:

// navigation Data

export const navItems = [

{

title: "Home",

url: "/",

},

{

title: "About Us", url: "/about",

},

{

title: "Products", url: "/products",

},

{

title: "FAQ",

url: "/faq",

},

{

title: "Contact Us", url: "/contact",

},

];

// categories data

{

id: 8,

title: "Jeans", subTitle: "", image\_Url:

"https://images.lee.com/is/image/Lee/112343021-HERO?$KDP-LARGE2$",

},

];

export const footerSupportLinks = [

{

name: "FAQ",

link: "/faq",

},

{

name: "Contact Us", link: "/contact",

},

];

## Server.js:

import React from "react";

import ReactDOM from "react-dom"; import App from "./App";

import reportWebVitals from "./reportWebVitals"; import { Provider } from "react-redux";

import Store from "./redux/store";

ReactDOM.render(

<Provider store={Store}>

<App />

</Provider>, document.getElementById("root")

);

reportWebVitals();

# APPENDIX 2

**SNAPSHOTS**

FIGURE A2.0 HOME PAGE

FIGURE A2.1 CATEGORIES PAGE

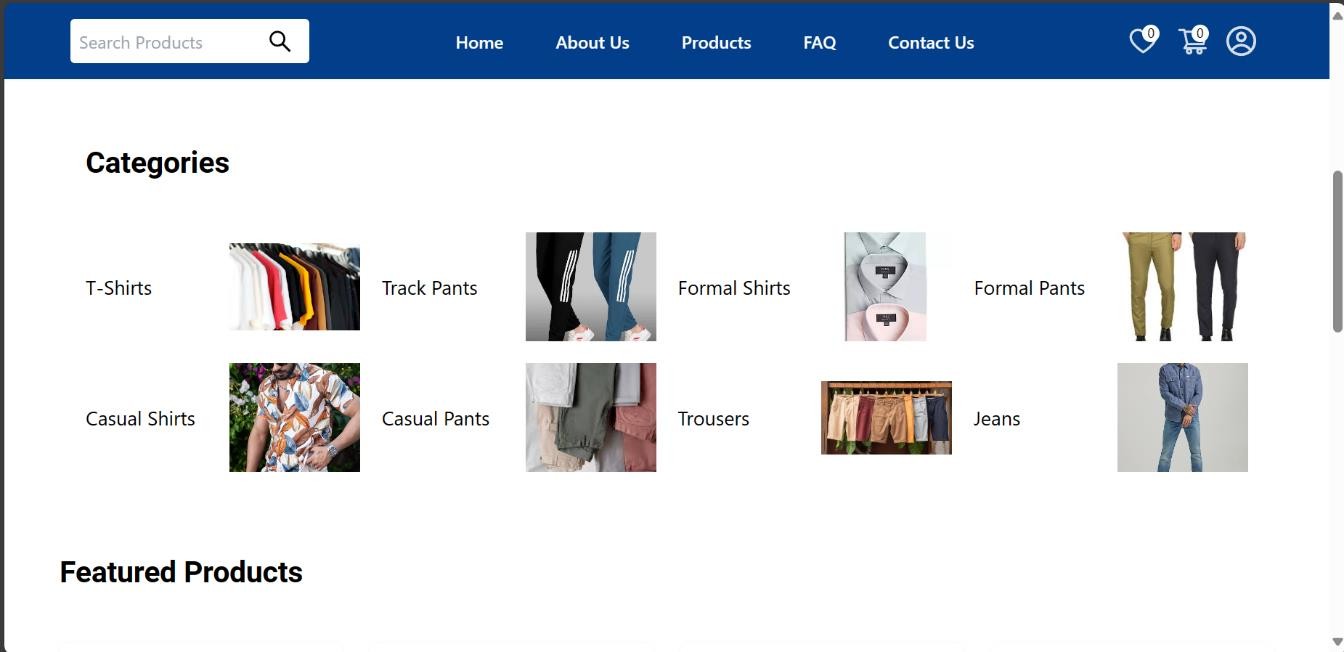


FIGURE A2.2 PRODUCTS

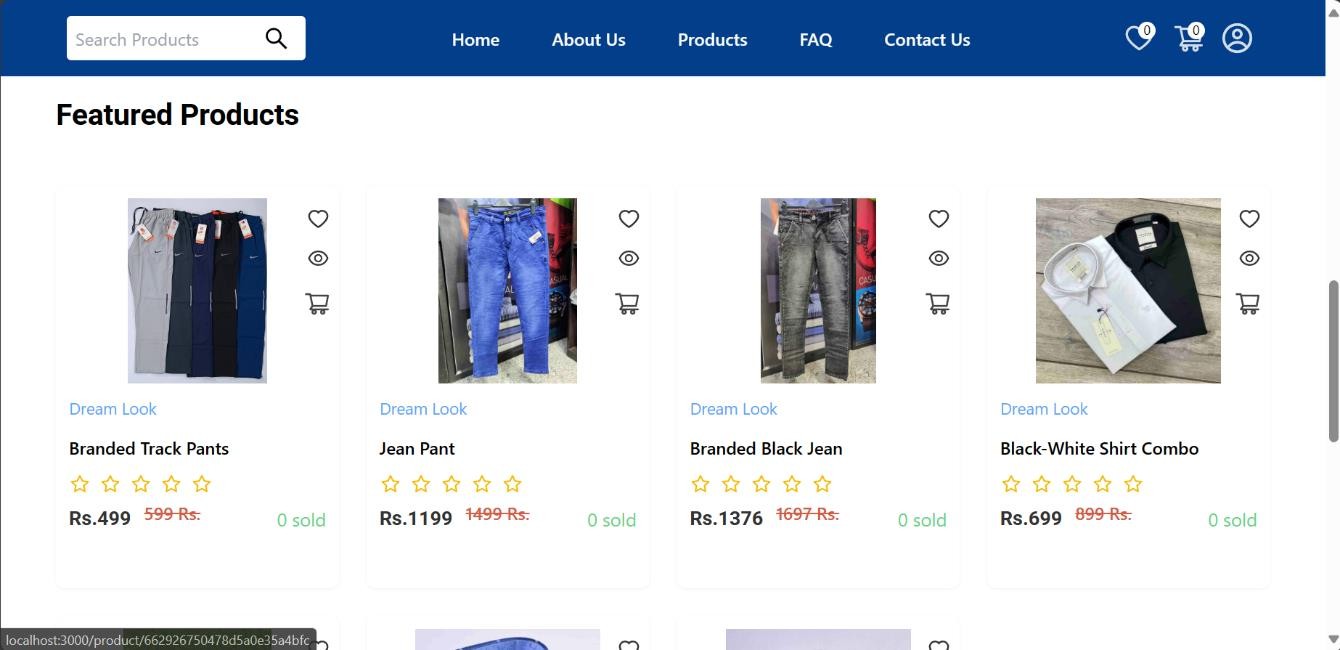


FIGURE A2.1 PROJECTS PAGE

FIGURE A2.3 ABOUT PAGE

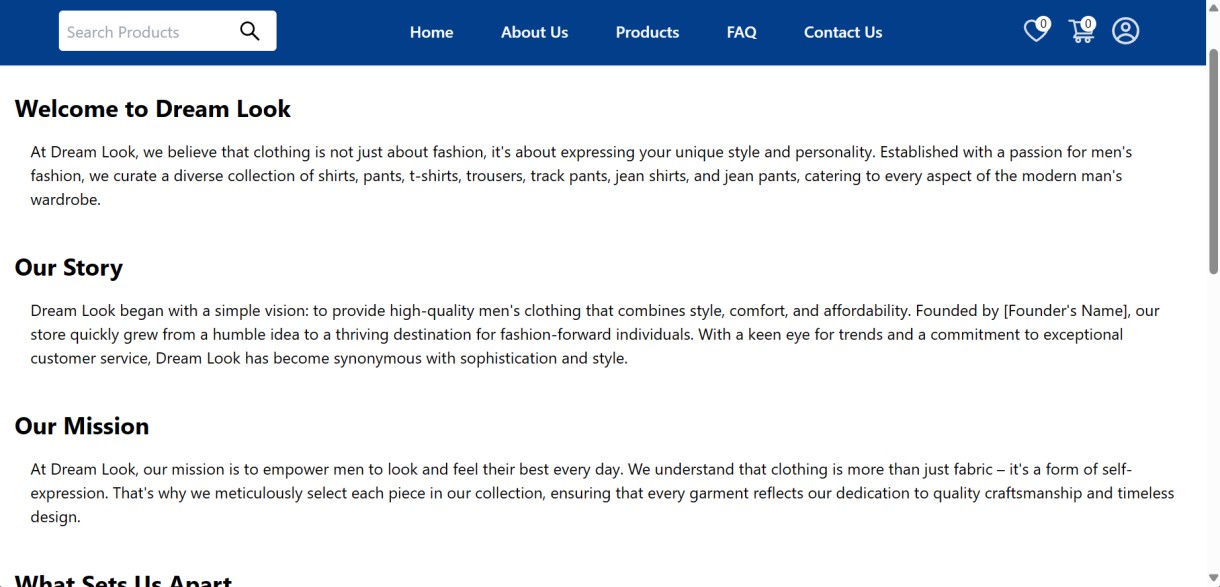


FIGURE A2.4 CONTACT PAGE

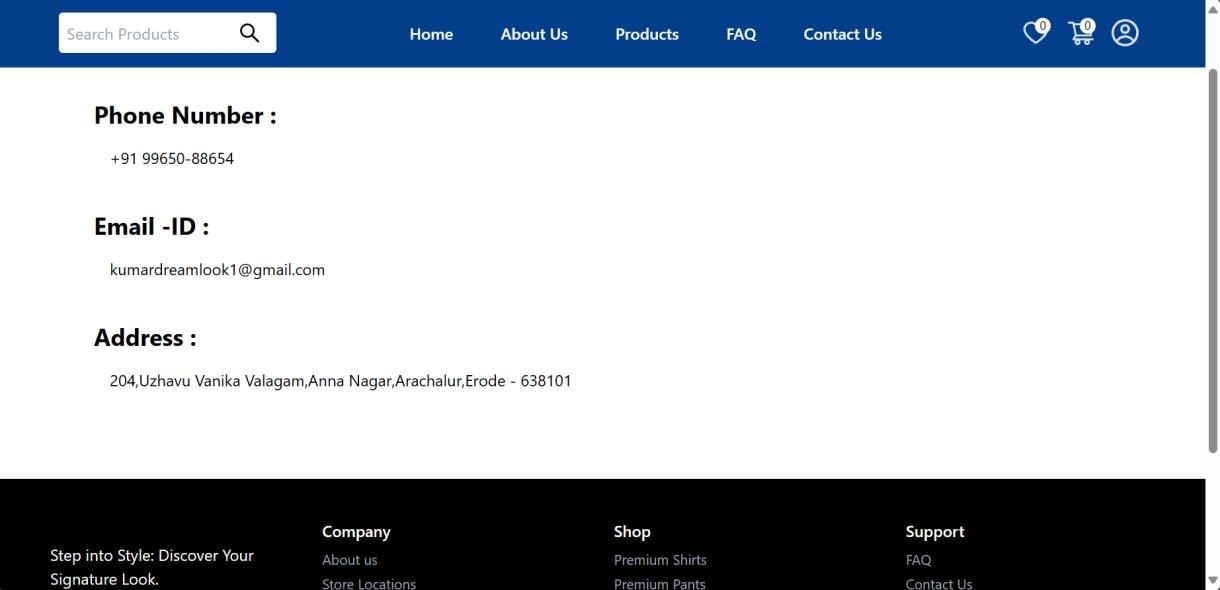


FIGURE A2.5 FOOTER PAGE

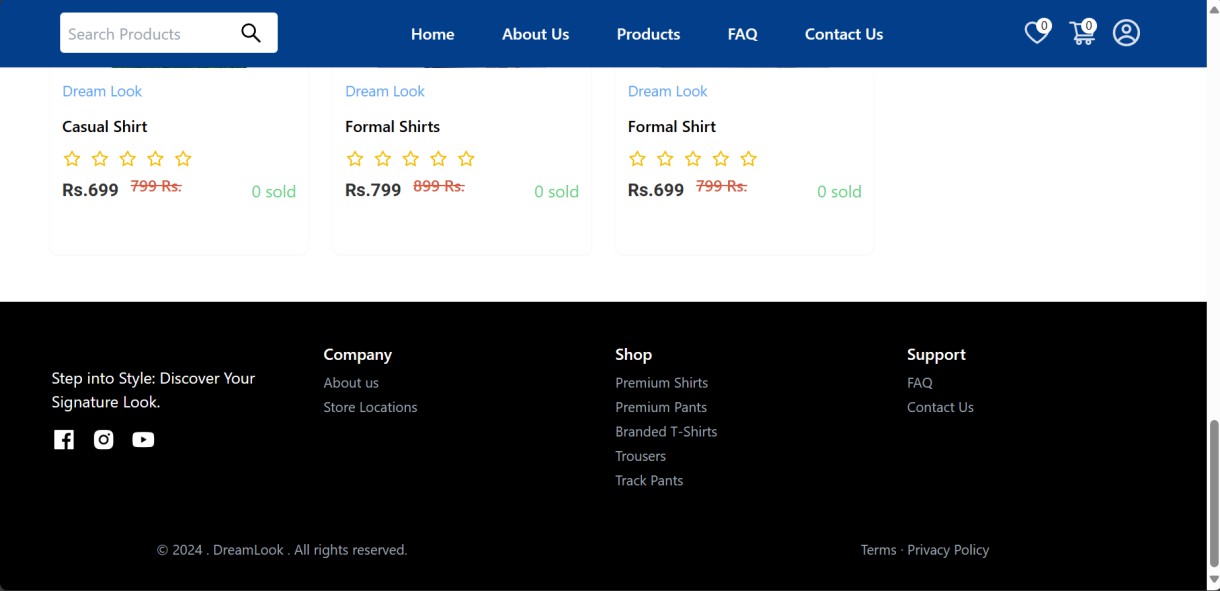


FIGURE A2.6 PROFILE PAGE

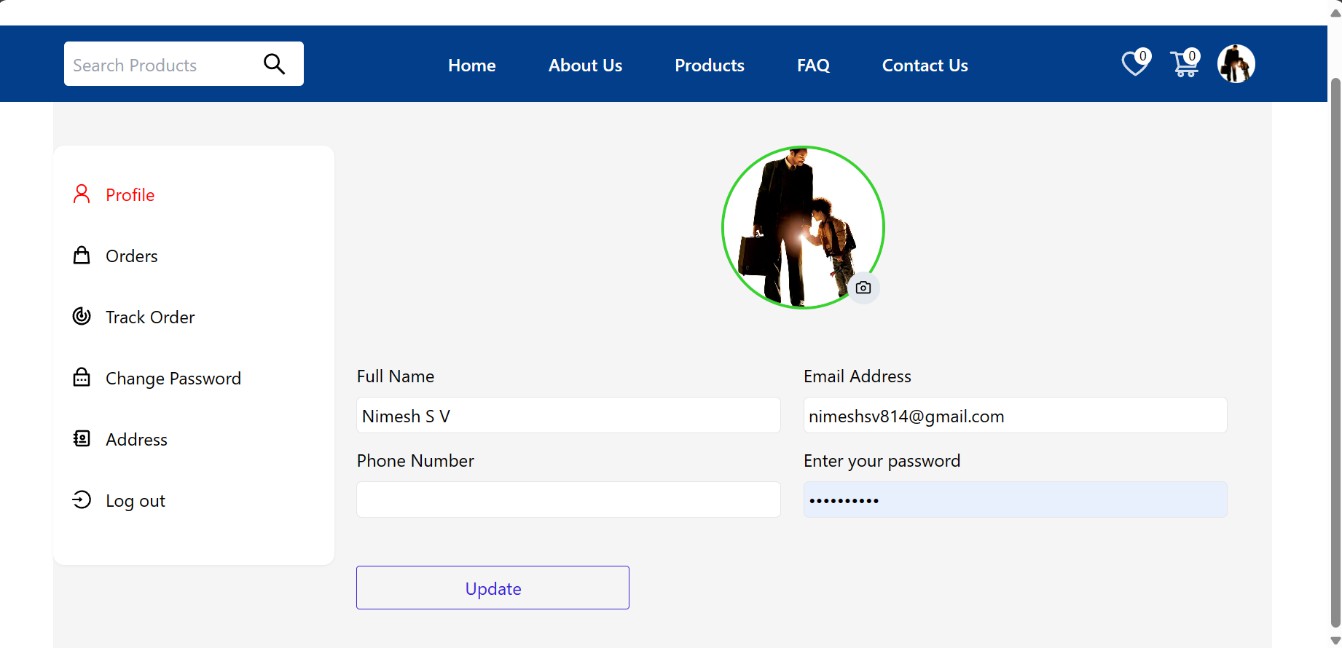


FIGURE A2.7 ADMIN LOGIN

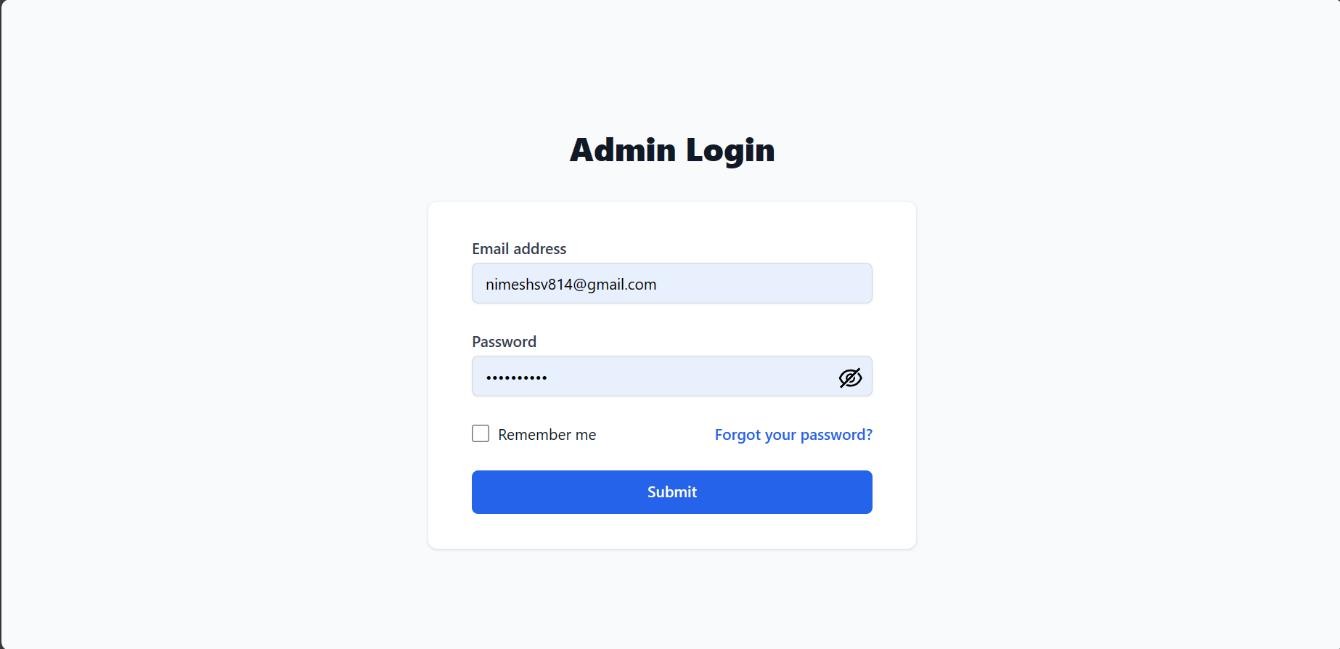


FIGURE A2.8 ADMIN DASHBOARD

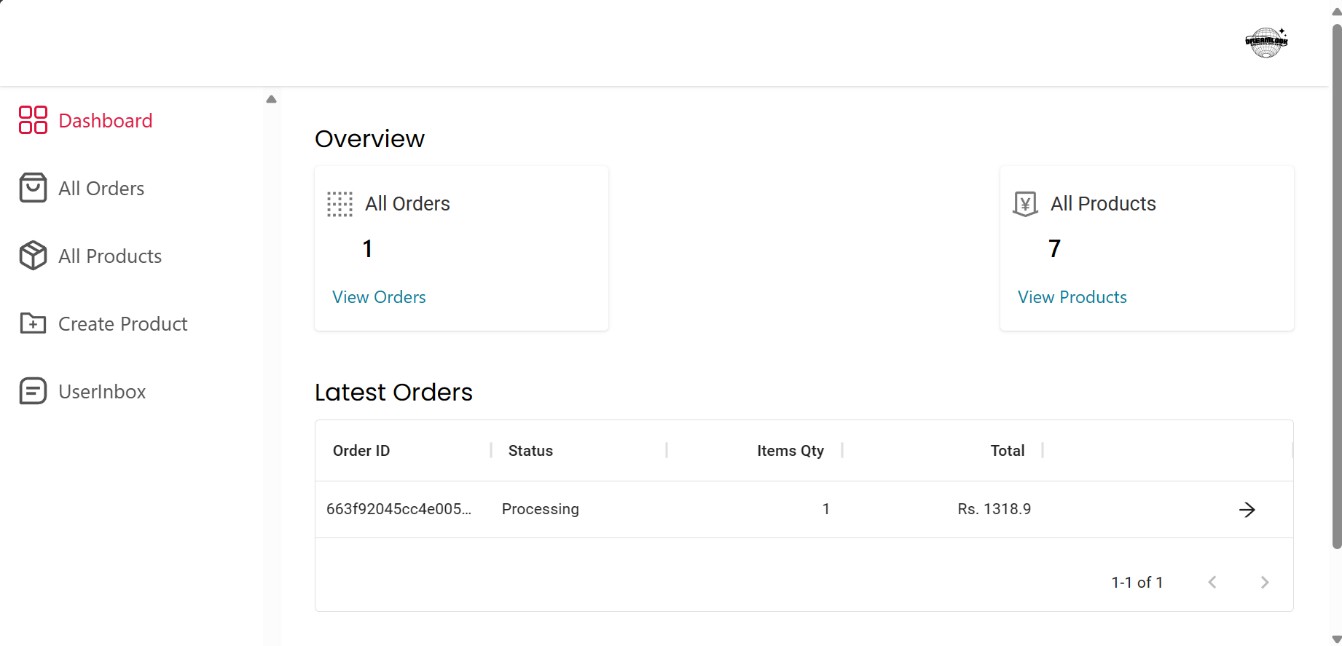


FIGURE A2.9 PRODUCT DESCRIPTION PAGE

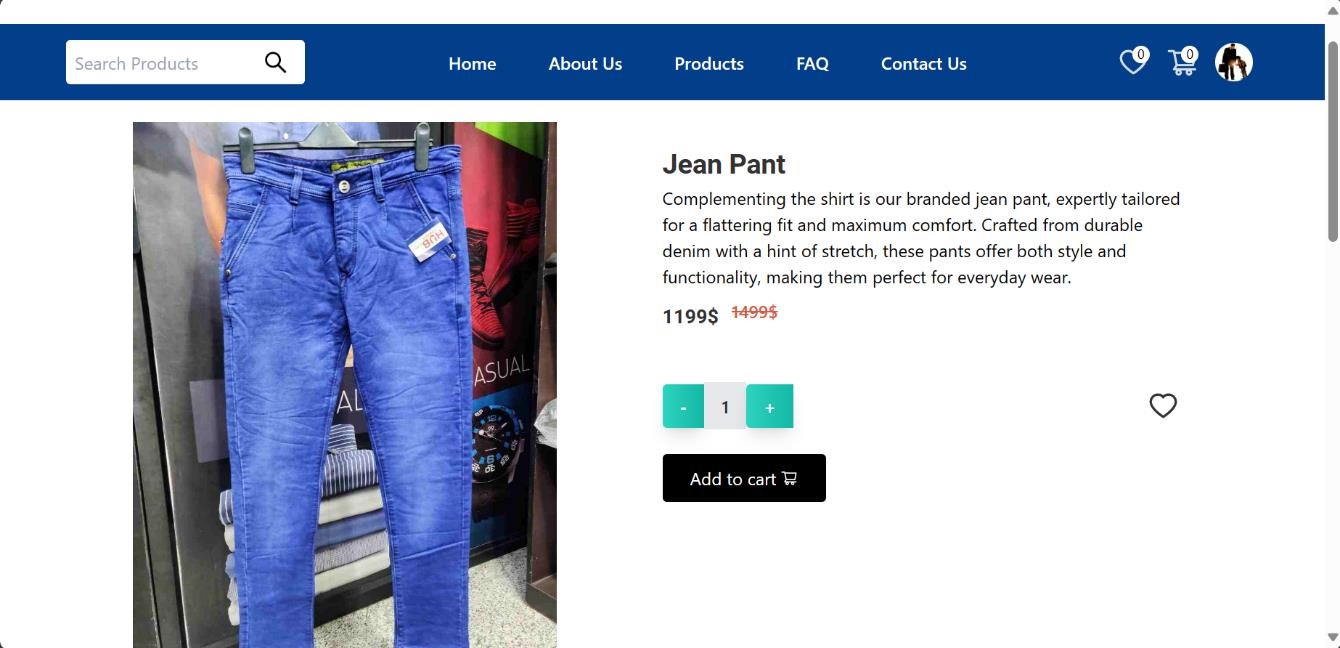


FIGURE A2.10 WISHLIST TAB

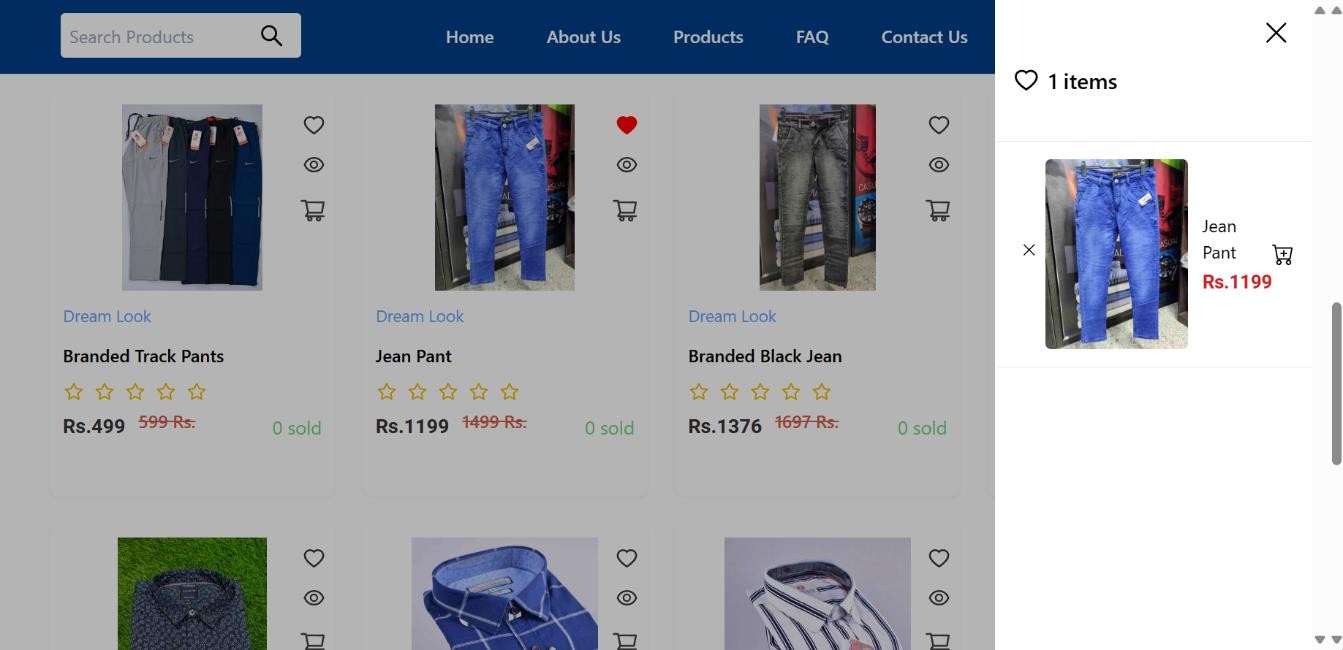


FIGURE A2.11 ADD TO CART TAB

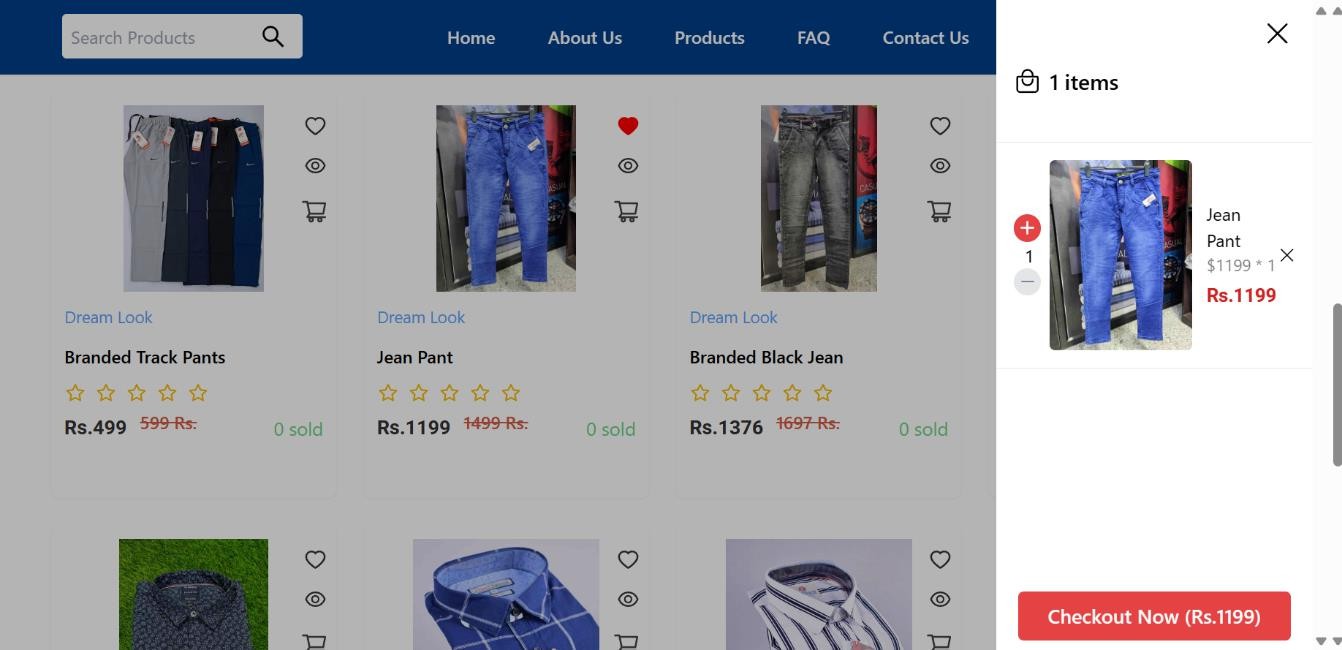
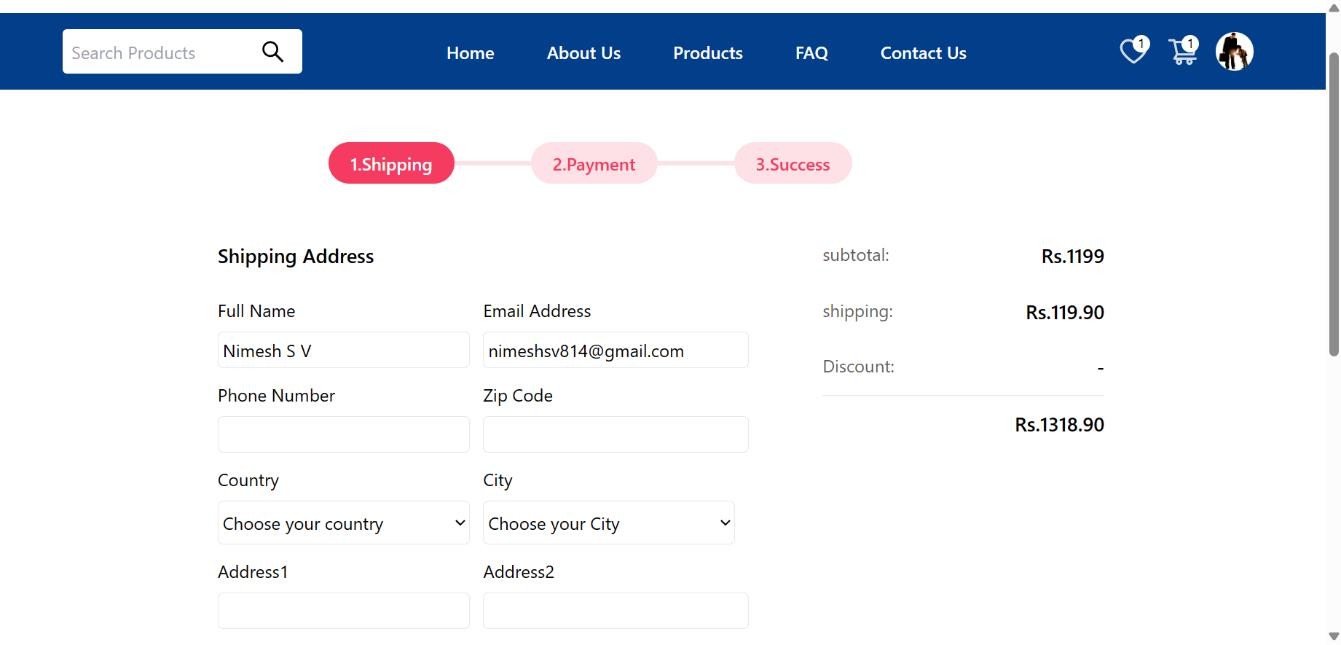


FIGURE A2.12 ONLINE ORDER PAGE



**REFERENCES**

|  |  |
| --- | --- |
| [1] | Mehra, Monika, Manish Kumar, Anjali Maurya, and Charu Sharma. "MERN stack Web Development." Annals of the Romanian Society for Cell Biology 25, no. 6 (2021): 11756-11761 |
| [2] | Yogesh Baiskar, Priyas Paulzagade, Krutik Koradia, Pramod Ingole, Dhiraj Shirbhate, "MERN: A Full-Stack Development", International Journal for Research in Applied Science & Engineering Technology (IJRASET) ISSN: 2321- 9653; IC Value: 45.98; SJ  Impact Factor: 7.538 Volume 10 Issue I Jan 2022. |