Shopper Style

A PROJECT REPORT
for
Mini Project (KCA353)
Session (2024-25)

Submitted by

Saurav Kumar Sinha
University Roll No 2300290140165

Submitted in partial fulfilment of the
Requirements for the Degree of

MASTER OF COMPUTER APPLICATION

Under the Supervision of
Mr. Apoorv Jain

Assistant Professor

Submitted to

DEPARTMENT OF COMPUTER APPLICATIONS
KIET Group of Institutions, Ghaziabad
Uttar Pradesh-201206

(DECEMBER 2024)

CERTIFICATE

Certified that **Saurav Kumar Sinha (2300290140165)** has carried out the project work having "**Shopper Style**" (**Mini-Project-KCA353**) for **Master of Computer Application** from Dr. A.P.J. Abdul Kalam Technical University (AKTU) (formerly UPTU), Lucknow under my supervision. The project report embodies original work, and studies are carried out by the student himself and the contents of the project report do not form the basis for the award of any other degree to the candidate or to anybody else from this or any other University/Institution.

Mr. Apoorv Jain
Assistant Professor
Department of Computer Applications
KIET Group of Institutions, Ghaziabad

Dr. Arun Tripathi
Head
Department of Computer Applications
KIET Group of Institutions, Ghaziabad

Shopper Style
(Saurav Kumar Sinha)

ABSTRACT

This project focuses on developing and deploying a full-stack eCommerce website using the MERN stack (MongoDB, Express.js, React.js, and Node.js) for selling clothing. The website allows users to browse various clothing products, filter and sort them by size, style, and price, and choose product options like size and color. Users can add items to their cart, enter their delivery details, and complete purchases through Cash on Delivery or online payments using Razorpay.

The website also includes an admin panel that lets administrators manage products adding, editing, or deleting items and keep track of orders and customer activity. The backend is built with Node.js and Express.js, while MongoDB stores all the product, order, and user data securely.

The frontend is deployed on Vercel to ensure a fast and responsive user experience. With its easy payment options, product management features, and secure data handling, this clothing eCommerce website provides a smooth shopping experience for customers and a user-friendly management tool for administrators.

Keywords: E-Commerce, MERN Stack, React.js, Node.js, MongoDB, Express.js

ACKNOWLEDGEMENTS

Success in life is never attained single-handedly. My deepest gratitude goes to my project supervisor, **Mr. Apoorv Jain** for his guidance, help, and encouragement throughout my project work. Their enlightening ideas, comments, and suggestions.

Words are not enough to express my gratitude to Dr. Arun Kumar Tripathi, Professor and Head, Department of Computer Applications, for his insightful comments and administrative help on various occasions.

Fortunately, I have many understanding friends, who have helped me a lot on many critical conditions.

Finally, my sincere thanks go to my family members and all those who have directly and indirectly provided me with moral support and other kind of help. Without their support, completion of this work would not have been possible in time. They keep my life filled with enjoyment and happiness.

Saurav Kumar Sinha

TABLE OF CONTENT

	Certificate	i
	Abstract	ii
	Acknowledgments	iii
	Table of Contents	iv
	List of Tables	v
	List of Figures	vi
1	**Introduction**	**01**
	1.1 Overview	01
	1.2 Background Study	01
	1.3 Project Planning	01
	1.4 Purpose and Objective	01
2	**System Design**	**02-04**
	2.1 Design	02
	2.2 User Characteristics	02
	2.3 System Information	02
	2.4 System Analysis	03
	2.5 Feasibility Analysis	03
	2.6 Context Design	03-04
3	**Hardware and Software Requirements**	**05-06**
	3.1 Hardware Requirements	05
	3.2 Software Requirements	06

4	**Implementing Tools for the Project**	**07-11**
4.1	Introduction to Tools	07
4.2	Frontend Tools	07-08
4.3	Backend Tools	08
4.4	Database Tools	09
4.5	Payment Integration Tools	09
4.6	Version Control and Collaboration Tools	09
4.7	Deployment Tools	10
4.8	Other Supporting Tools	10
4.9	Data Flow Diagram	11
5	**Entity Relationship Diagram**	**12**
6	**Project Database & Screenshot**	**13-23**
7	**Coding**	**24-46**
8	**Conclusion**	**47**
9	**Reference**	**48**

LIST OF FIGURES

Figure No.	Name of Figure	Page No.
2.6.	Context diagram	04
4.9	Data flow diagram	11
4.9	Level 0 DFD Diagram	11
4.9	Level 1 DFD Diagram	11
5.0	Entity Relationship Diagram	12
6.0	Order Database	14
6.1	User Database	15
6.2	Login/ Sign up Page	16
6.3	Home Page	17
6.4	Collection/ Best-Seller Page	18
6.5	About/ Contact Page	19
6.6	Cart/ Checkout Page	20
6.7	Payment Page	21
6.8	User Order Page	22
6.9	Admin Order Page	23

Chapter 1: Introduction

1.1 Overview

Online shopping has become very popular because it is convenient and easy to use. This project is about creating an eCommerce website for selling clothing using the MERN stack (MongoDB, Express.js, React.js, and Node.js). The website allows users to browse and filter products, choose options like size and color, add items to their cart, and pay using Razorpay or Cash on Delivery. It also includes an admin panel for managing products and orders, making it simple for administrators to handle the platform.

1.2 Background Study

Many people now prefer shopping online, especially for clothes, but some websites are not easy to use or lack good management tools. This project uses the MERN stack to build a modern website that's user-friendly, fast, and efficient for both customers and administrators.

1.3 Project Planning

The project was divided into four main steps:

1. Understanding Requirements: Figuring out what features users and admins need.
2. Design: Planning how the website will look and how data will be stored.
3. Development: Writing the frontend, backend, and database code.
4. Testing and Deployment: Making sure everything works and putting the site online.

1.4 Purpose and Objectives

The goal of this project is to make a website that:

- Let's users easily browse and buy clothes.
- Provides secure payment options.
- Helps admins manage products and orders efficiently.
- Is reliable and can grow as the business expands.

Chapter 2: System Design

2.1 Design

This section describes the overall design of the eCommerce website, including the frontend, backend, and database structure. It explains how each part interacts to provide a seamless user experience and efficient data management.

2.2 User Characteristics

This section identifies the target users of the system:

- Customers: Individuals looking to browse, filter, and purchase clothing.
  - Characteristics: Diverse age groups, tech-savvy or first-time online shoppers.
  - Needs: Easy navigation, secure payments, and fast checkout.
- Administrators: Business owners or staff managing the website.
  - Characteristics: Basic technical knowledge for managing products and orders.
  - Needs: Tools for adding, editing, or deleting products and tracking orders.

2.3 System Information

This section outlines the core system components:

- Frontend: Built with React.js for an interactive and responsive user interface.
- Backend: Developed using Node.js and Express.js for handling APIs and business logic.
- Database: MongoDB stores data securely, including user profiles, products, orders, and payment records.
- Payment Gateway: Razorpay integration for secure online transactions.

2.4 System Analysis

This section examines the functional and non-functional requirements of the system:

- Functional Requirements:
  - Allow users to browse, filter, and sort products.
  - Enable customers to add items to the cart and checkout.
  - Provide admin tools for managing products and orders.
- Non-Functional Requirements:
  - Ensure fast load times and responsiveness.
  - Provide secure data handling and payment processing.

2.5 Feasibility Analysis

This section evaluates the feasibility of the project in different areas:

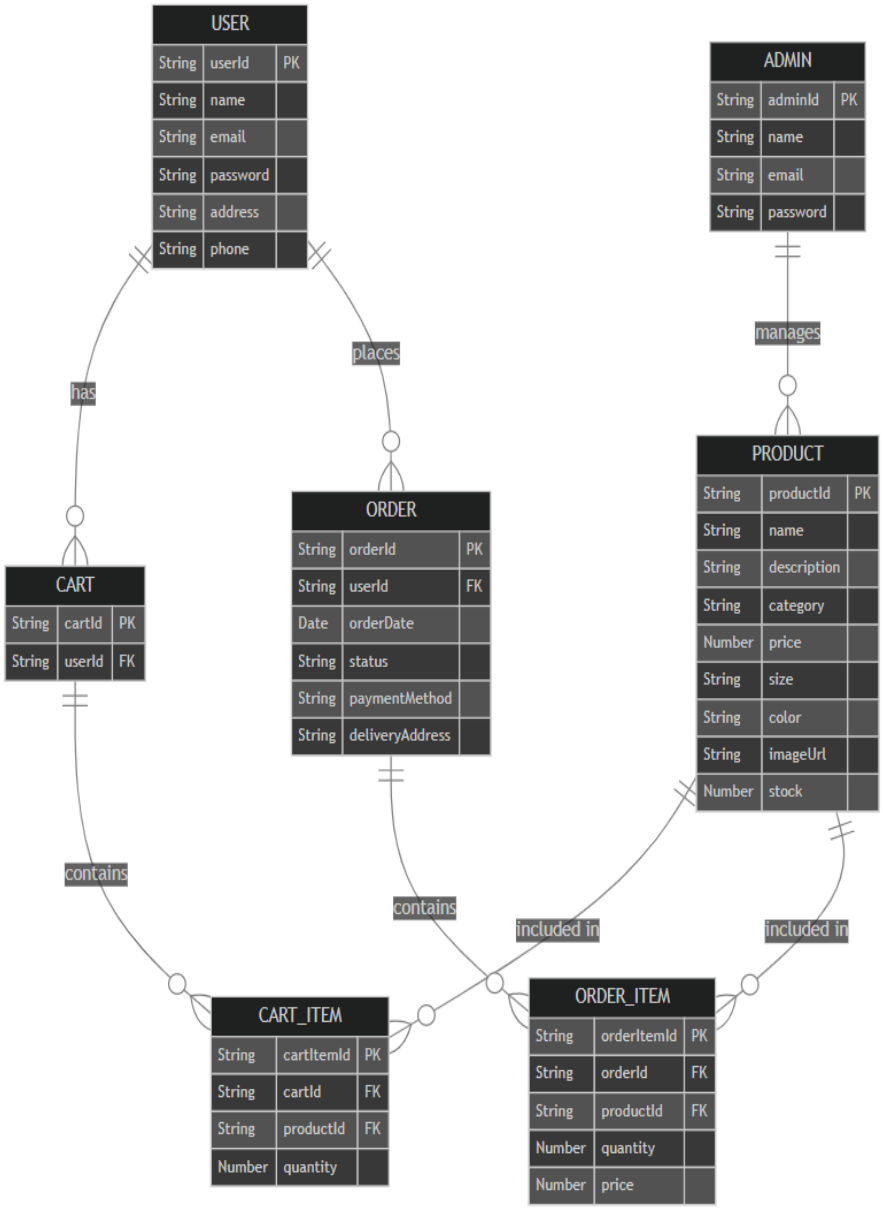
- Technical Feasibility: The MERN stack offers proven tools to build scalable and reliable applications.
- Economic Feasibility: Minimal cost due to open-source technologies like React.js and MongoDB.
- Operational Feasibility: Easy-to-use interfaces for both customers and administrators.

2.6 Context Design

This section explains the system's context using a Context Diagram to show interactions between users and the system.

- Customers interact with the website to browse, add to the cart, and make payments.
- Administrators use the admin panel to manage products and orders.
- Payment Gateway (Razorpay) handles secure payment transactions.

Design Diagram: - The design diagram shows how different parts of the system work together. It connects users (customers and admins) with the website, the database, and the payment gateway. Customers browse, order, and pay for products, while admins manage products and orders. The database stores all data, and the payment gateway handles secure payments. This diagram helps explain the system's overall structure and flow.



Chapter 3: Hardware and Software Requirements

3.1 Hardware Requirements

The following hardware components are required to develop and run the eCommerce website:

- Development System:
  - Processor: Intel Core i5 or higher
  - RAM: 8 GB or higher
  - Storage: 256 GB SSD or higher
  - Monitor: Minimum resolution of 1920x1080 pixels
- Deployment Server:
  - Processor: Quad-core CPU (e.g., Intel Xeon or AMD equivalent)
  - RAM: 16 GB or higher
  - Storage: 500 GB SSD or higher
  - Network: High-speed internet connection for handling user requests

3.2 Software Requirements

The software required for development, testing, and deployment includes:

- Development Tools:
  - Code Editor: Visual Studio Code
  - Version Control: Git and GitHub
- Frontend:
  - Language: JavaScript
  - Library/Framework: React.js
  - Styling: CSS, Bootstrap
- Backend:
  - Runtime Environment: Node.js
  - Framework: Express.js
- Database:
  - MongoDB
- Testing Tools:
  - Postman (for API testing)
- Deployment:
  - Hosting Platform: Vercel (for the frontend)
- Other Tools:
  - Payment Gateway: Razorpay
  - Browser: Google Chrome or equivalent

Chapter 4: Implementing Tools for the Project

This chapter details the tools and technologies used to implement the eCommerce project successfully. The tools were selected based on their reliability, ease of use, and ability to meet the project's requirements.

4.1 Introduction to Tools

Developing a full-stack eCommerce website requires a combination of tools for frontend design, backend logic, database management, and deployment. These tools work together to create a seamless shopping experience for users and efficient management capabilities for administrators.

4.2 Frontend Tools

Frontend tools focus on creating a visually appealing and responsive user interface:

1. **React.js**:
   - React.js was used to build the dynamic user interface of the website.
   - Features like reusable components and state management through React Hooks were utilized to streamline development and enhance functionality.
   - React Router was used for navigation between different pages.
2. **Bootstrap**:
   - Bootstrap provided pre-designed CSS classes and components for faster and more consistent UI development.
   - It ensured the website was responsive across different devices, including mobiles, tablets, and desktops.

3. **HTML5 and CSS3**:
    - HTML5 was used for structuring the website, defining its content, and ensuring semantic accuracy.
    - CSS3 was employed for custom styling, animations, and visual effects to enhance the overall appearance.
4. **Vercel**:
    - The frontend was deployed on Vercel, a platform that provides fast deployment and global content delivery, ensuring the website loads quickly for users worldwide.

4.3 Backend Tools

Backend tools handle the server-side logic and communication with the database:

1. **Node.js**:
    - Node.js served as the runtime environment for building the backend server.
    - It handled asynchronous requests, allowing multiple users to interact with the website simultaneously without lag.
2. **Express.js**:
    - Express.js, a framework built on Node.js, simplified the creation of RESTful APIs for communication between the frontend and backend.
    - Middleware features were used for error handling, logging, and validating user input.

4.4 Database Tools

Database tools were essential for managing and securing application data:

1. **MongoDB**:
   - MongoDB, a NoSQL database, was chosen for its scalability and flexibility.
   - Collections were created for storing products, user details, orders, and payment data in a structured format.

4.5 Payment Integration Tools

Secure and seamless payment processing is critical for any eCommerce platform:

1. **Razorpay**:
   - Razorpay was integrated into the project to handle online payments securely.
   - APIs provided by Razorpay were used to process transactions, send notifications for payment confirmations, and manage refunds if necessary.

4.6 Version Control and Collaboration Tools

Version control ensures smooth collaboration and prevents code conflicts:

1. **Git**:
   - Git was used to track changes to the codebase, allowing developers to roll back to previous versions if needed.
   - Branching and merging features helped manage different features or bug fixes without affecting the main codebase.
2. **GitHub**:
   - The project repository was hosted on GitHub, where team members could push their changes and review each other's code.

4.7 Deployment Tools

Deployment tools made the project accessible to end-users:

1. **Vercel**:
   - Vercel was used to deploy the frontend application, providing a simple interface and efficient deployment pipelines.
   - It ensured fast load times by leveraging CDN (Content Delivery Network) for static assets.

4.8 Other Supporting Tools

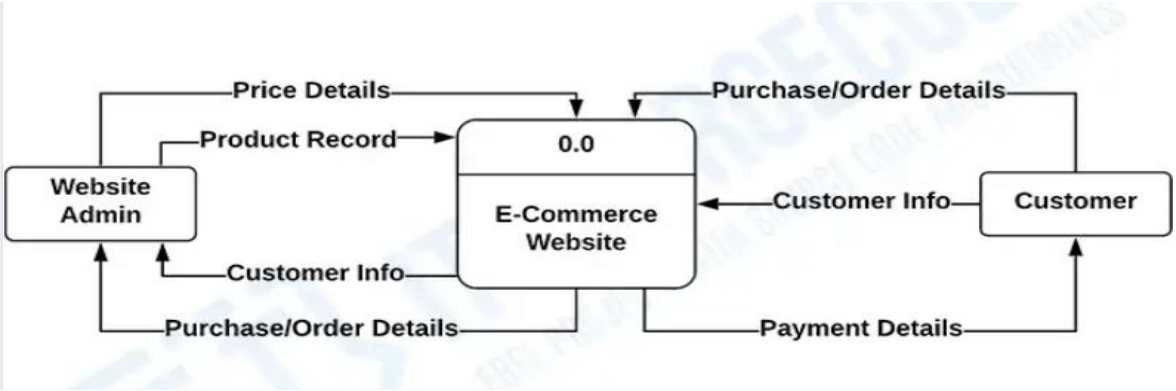
Additional tools enhanced development and testing:

1. **Visual Studio Code**:
   - The project was developed using Visual Studio Code, a powerful code editor with support for JavaScript and Node.js.
   - Extensions such as ESLint and Prettier helped maintain code quality and consistency.
2. **Browser Developer Tools**:
   - Built-in browser tools were used for debugging frontend issues and testing responsiveness.

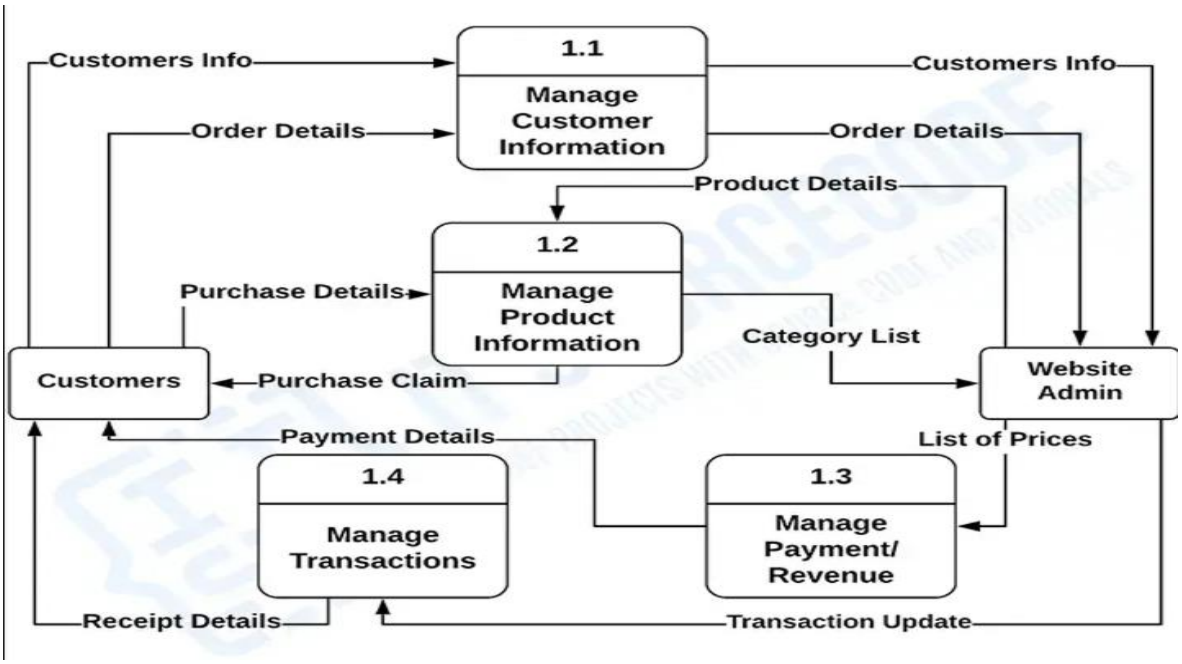
4.9 Data Flow Diagram

The data flow diagram shows the flow of data within any system. It is an important tool for designing phase of software engineering. Larry Constantine first developed it. It represents graphical view of flow of data. It's also known as BUBBLE CHART.

Level 0: -

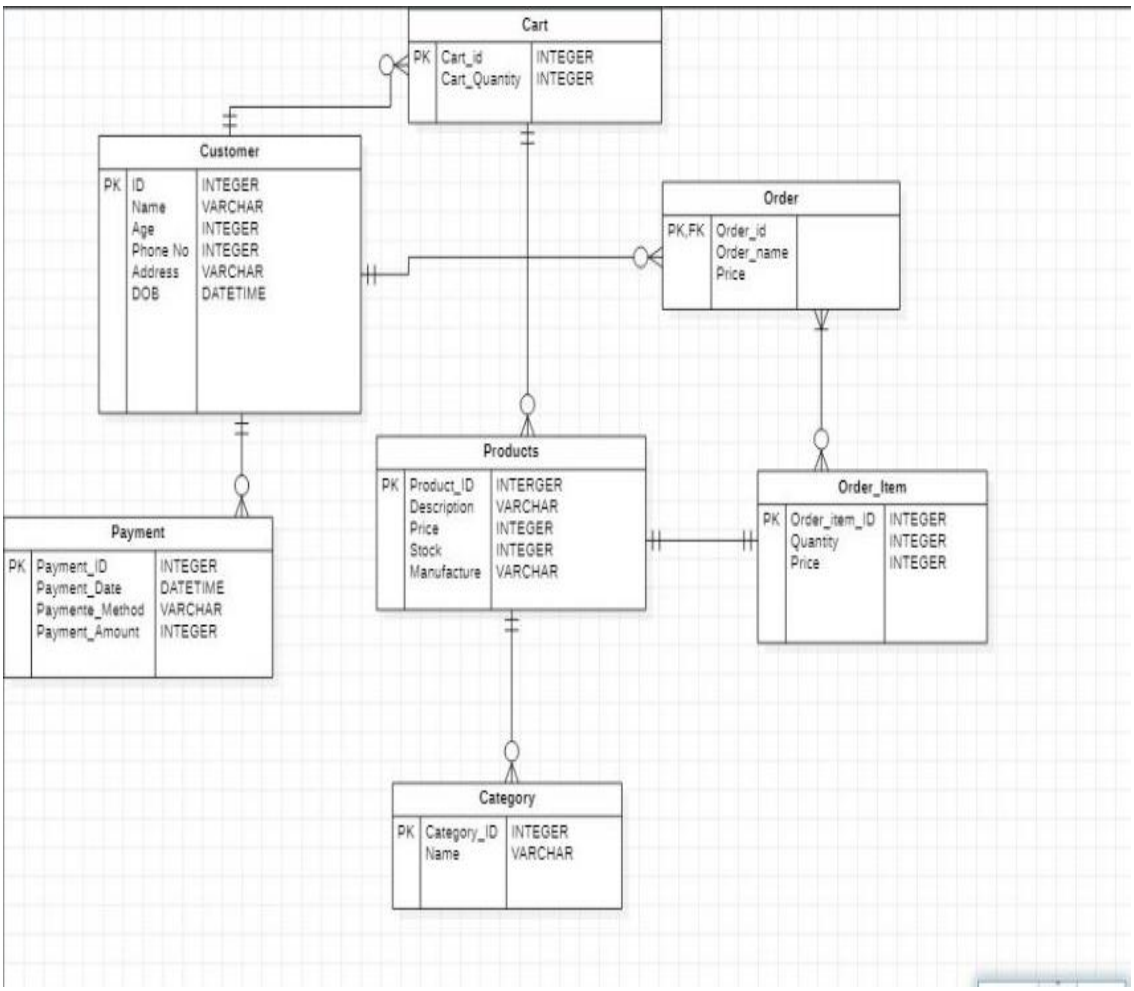


Level 1: -



Chapter 5: Entity Relationship Diagram

The entity-relationship model or entity-relationship diagram (ERD) is a data model or diagram for high-level descriptions of conceptual data models, and it provides a graphical notation for representing such data models in the form of entity-relationship diagrams.



Chapter 6: Project Database & Screenshots

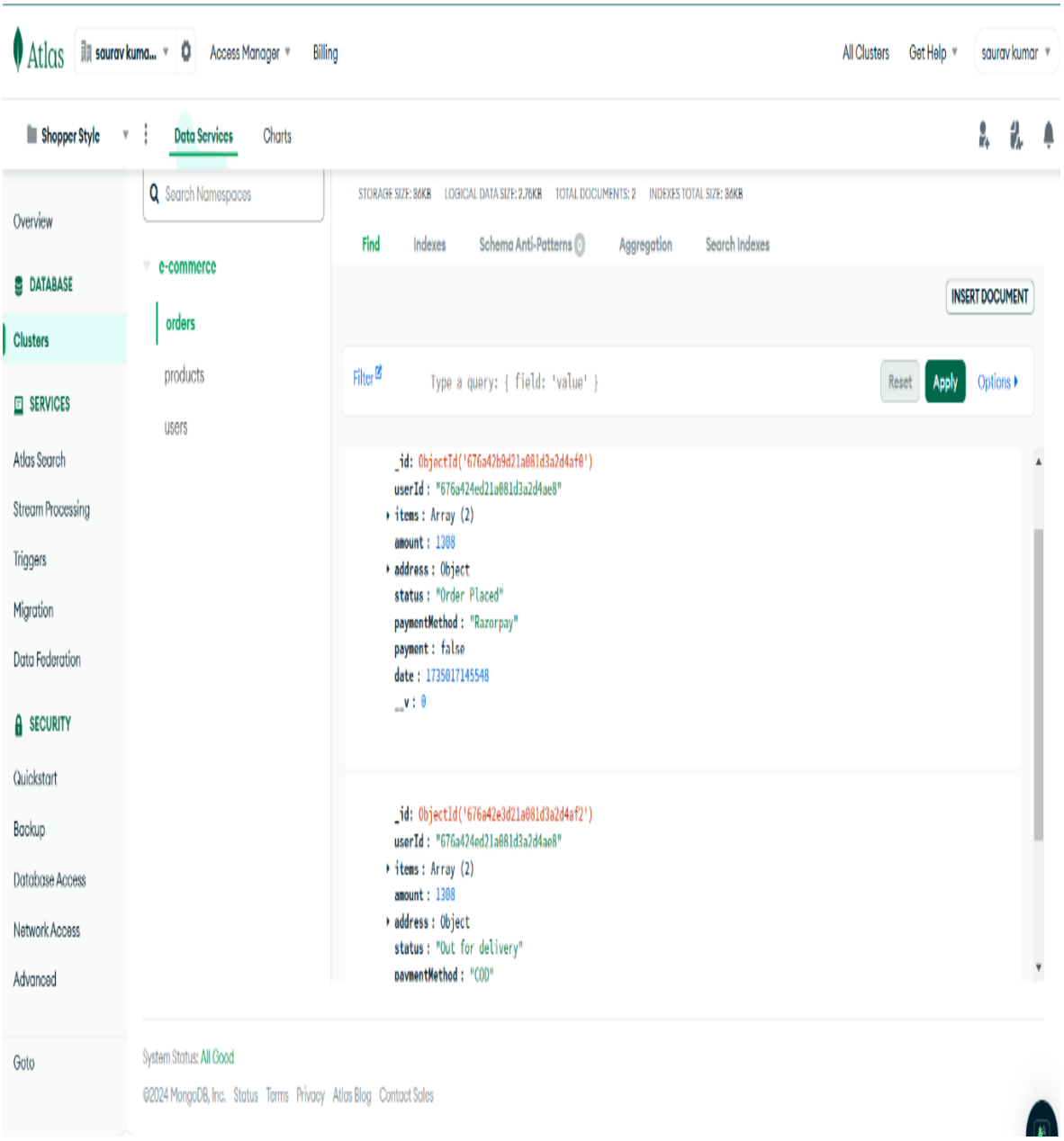
This chapter describes the structure and design of the database used for the eCommerce project, outlining all the tables (or MongoDB collections) and their respective fields. The database is central to managing data for users, products, categories, orders, and other key aspects of the project.

6.1 Database Design

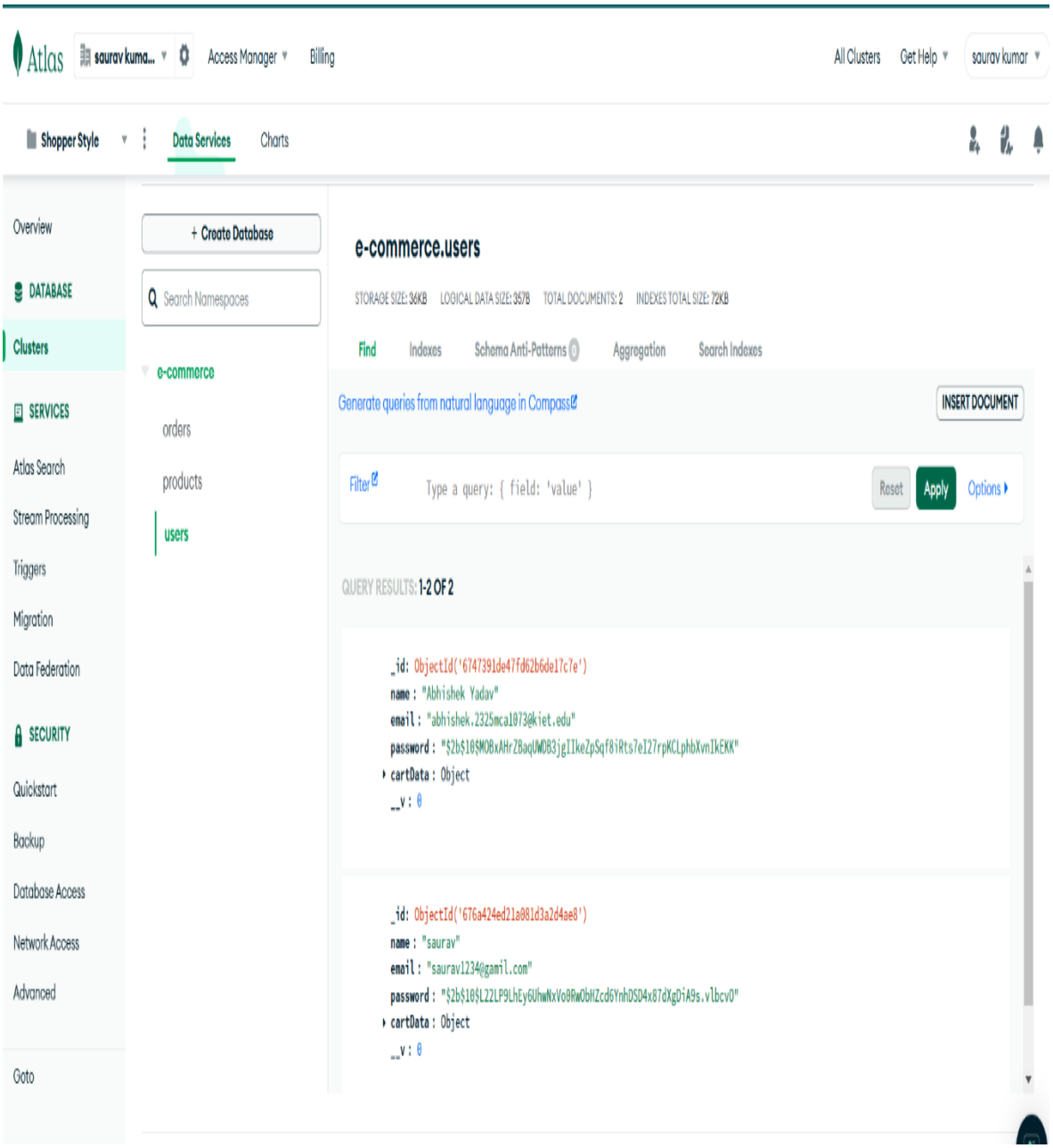
The eCommerce database is designed using **MongoDB**, a NoSQL database that provides flexibility and scalability. The database schema ensures efficient data storage, quick retrieval, and proper relationships between various entities like users, products, and orders. Key design considerations include:

- **Normalization** to avoid redundancy.
- **Secure data handling**, especially for sensitive information like passwords and payment details.

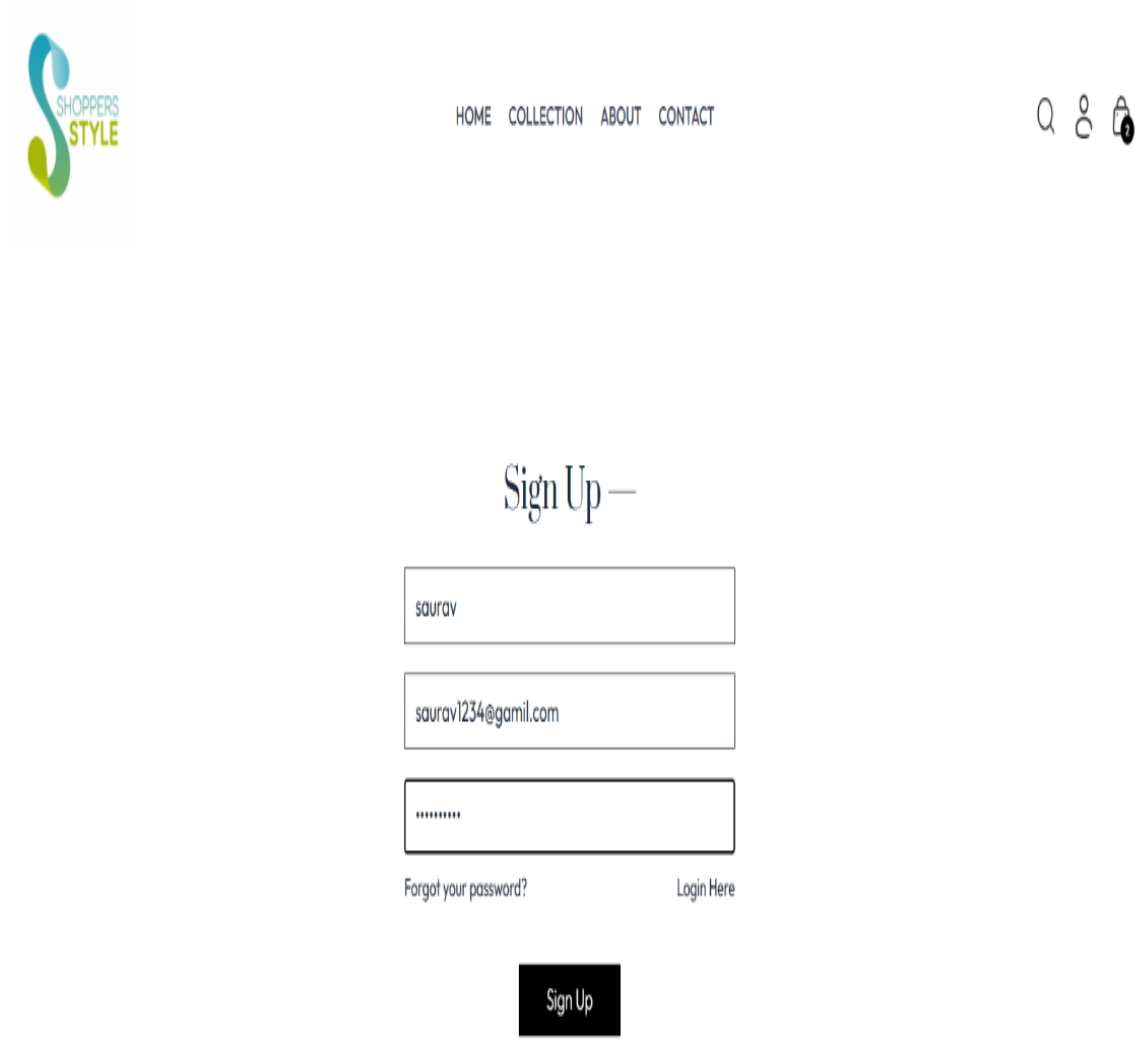
Order Database: - The Order database keeps track of customer orders, including product details, quantities, sizes, payment status, and delivery updates. It helps the system follow an order from start to finish, ensuring smooth management. Administrators can update orders and track their progress. The secure design supports future growth, making it reliable for handling many orders efficiently.



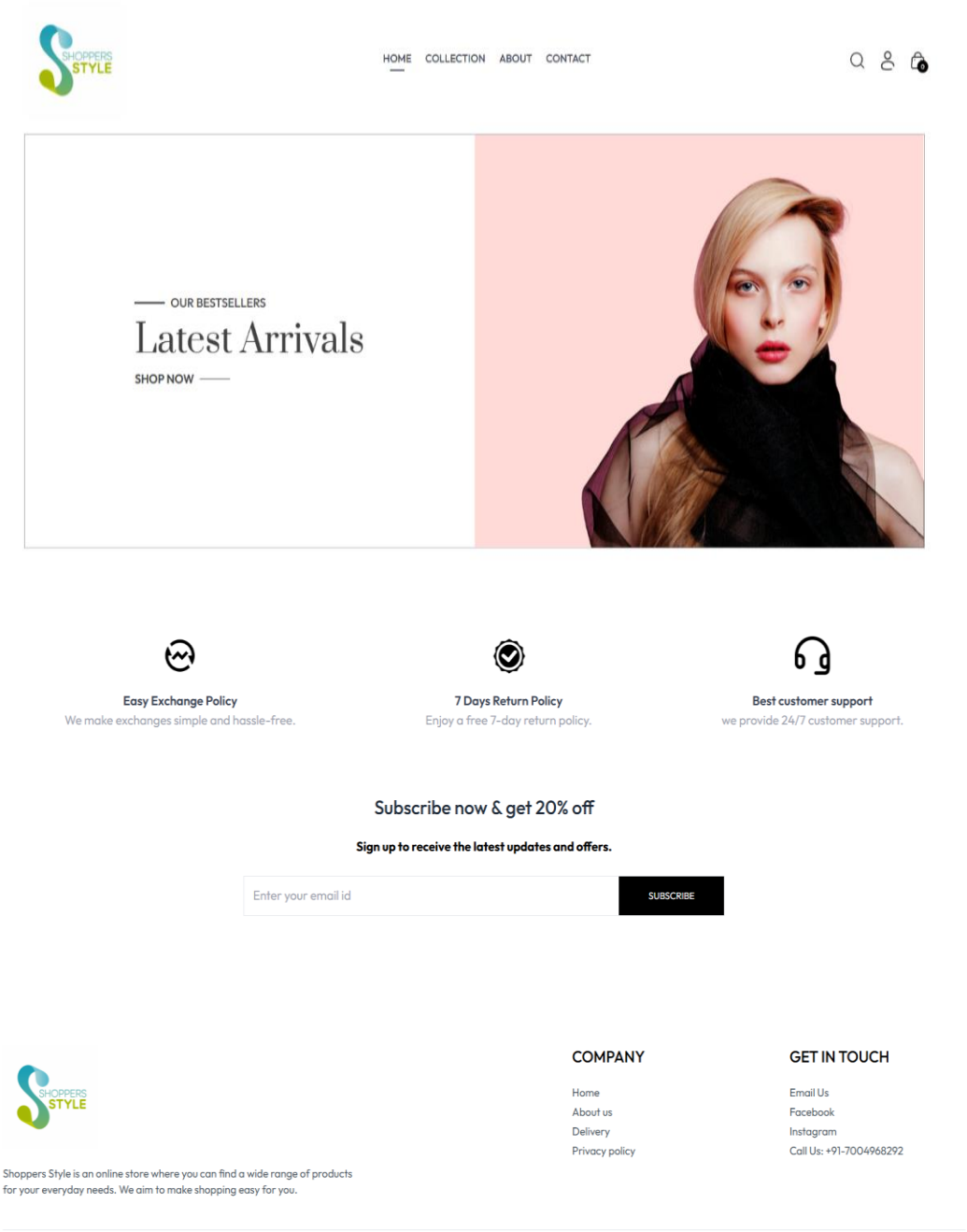
User Database: - The User database stores user information, such as their name, email, and securely hashed passwords. It also keeps track of their shopping carts. This database is essential for logging in users, personalizing their experience, and securely connecting them to other platform parts like orders and payments.



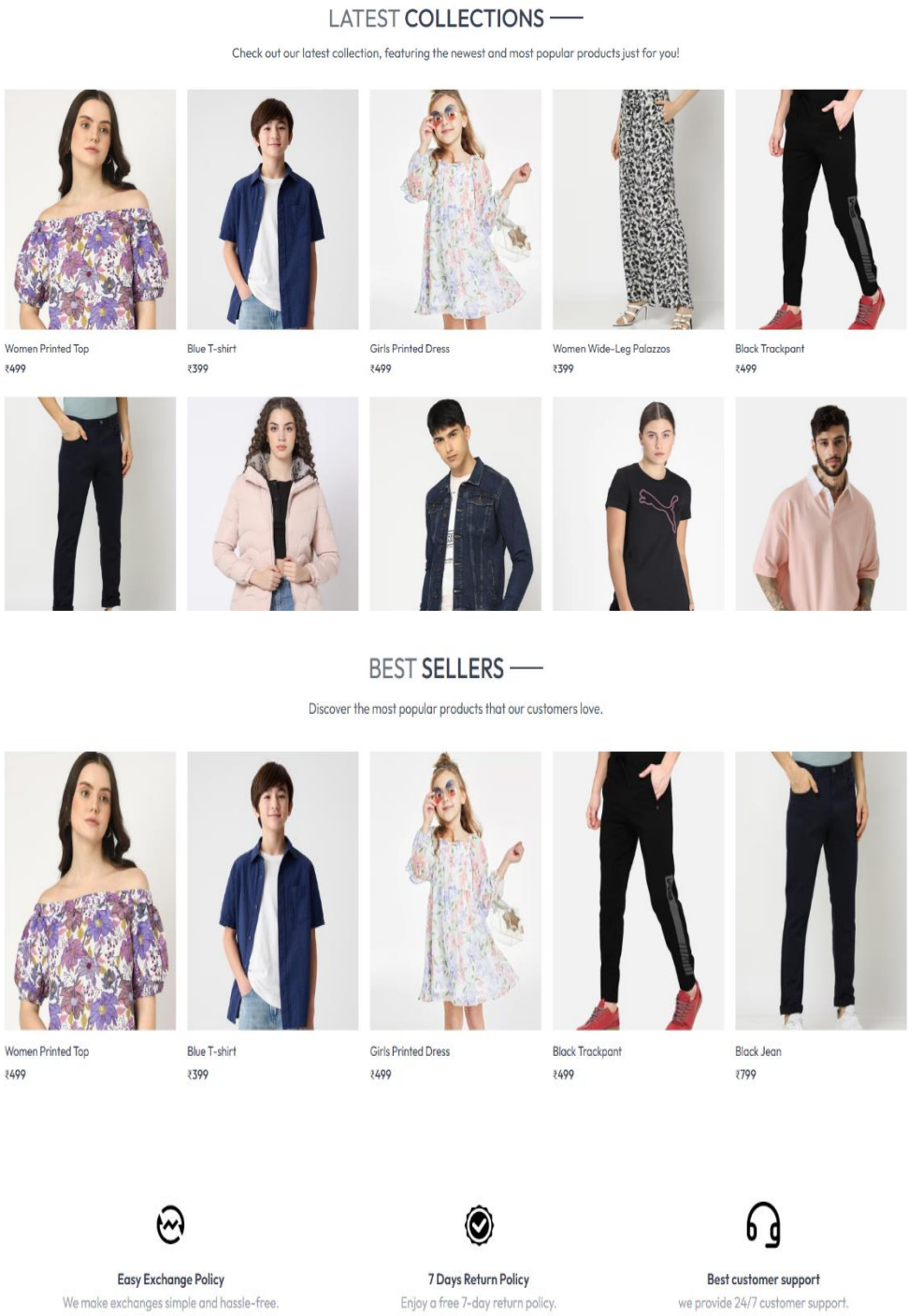
Login / Signup Page: - The Login and Sign-Up pages let users register or log in securely. The login page collects email and password, while the sign-up page adds fields like name. They ensure secure access to the platform using modern methods for password storage and user authentication, creating a smooth and safe user experience.



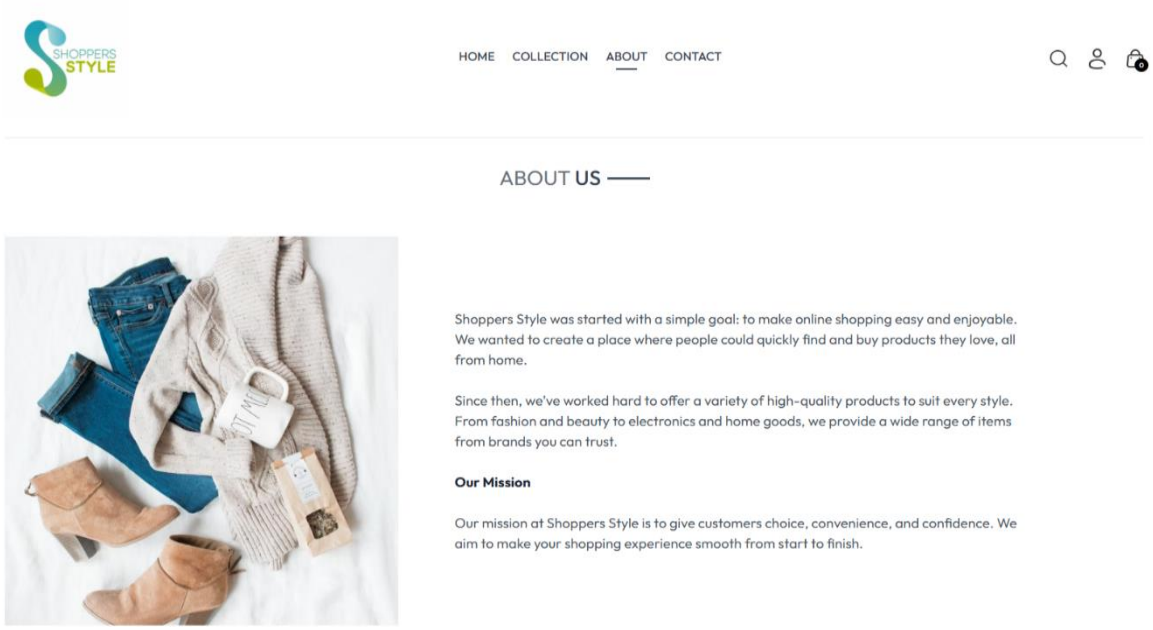
Home Page: - The Home page is the starting point for users. It showcases featured products, new collections, and links to key categories. The page is designed to work well on all devices, making navigation easy and engaging. It updates in real-time to show users the latest products and promotions.

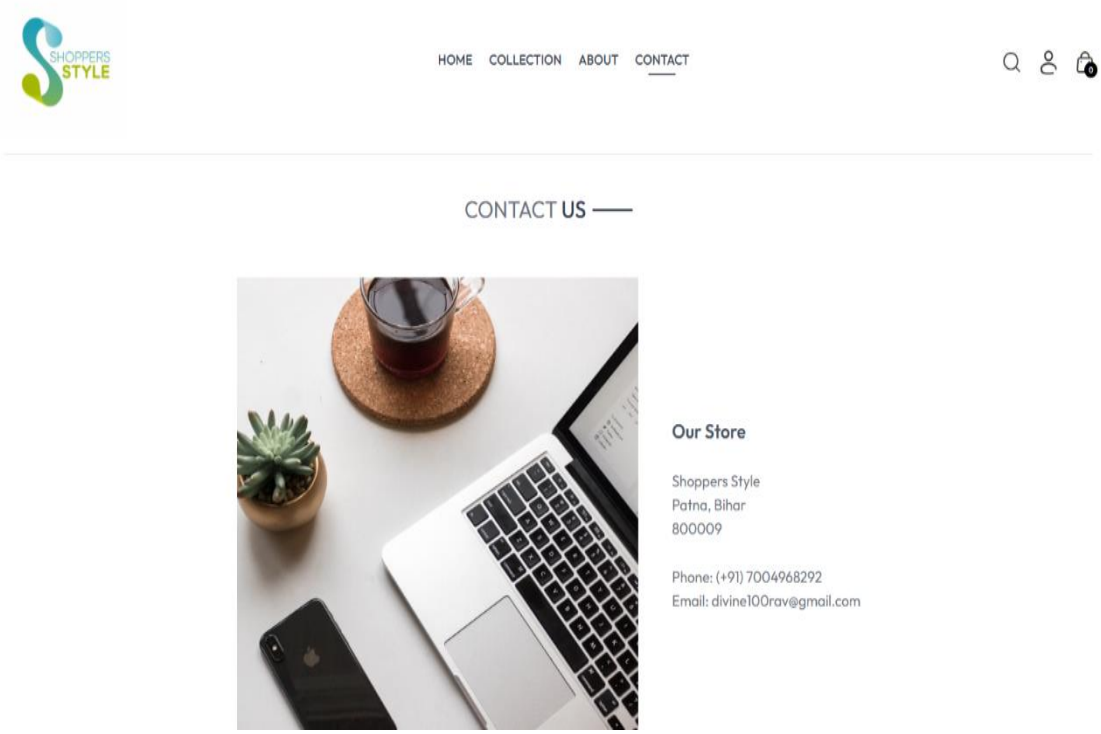


Collection/ Best-Seller Page: - This page highlights popular or specific collections of products. It lets users filter and sort items based on size, price, or style. The responsive design ensures it works on all devices, while live updates make sure the displayed products are always relevant, helping users find what they want quickly.

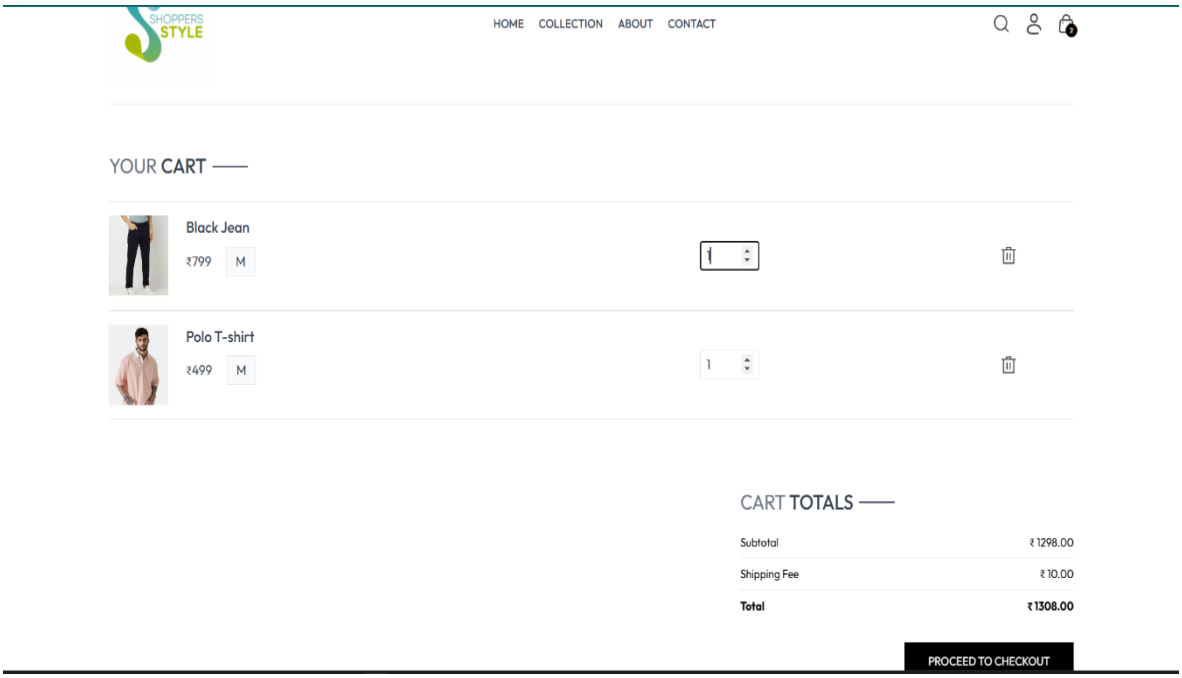


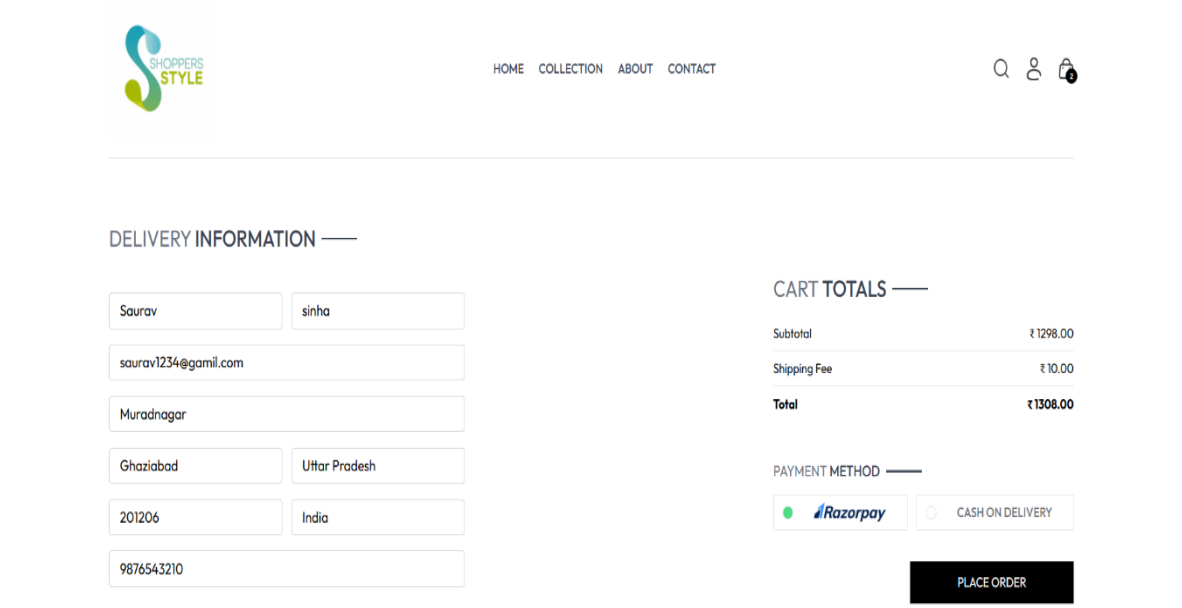
About/ Contact Page: - The About page explains the platform's purpose and values, while the Contact page provides ways to reach out for support, like email or forms. It’s designed to be accessible on all devices, helping build trust by showing professionalism and offering help when users need assistance.



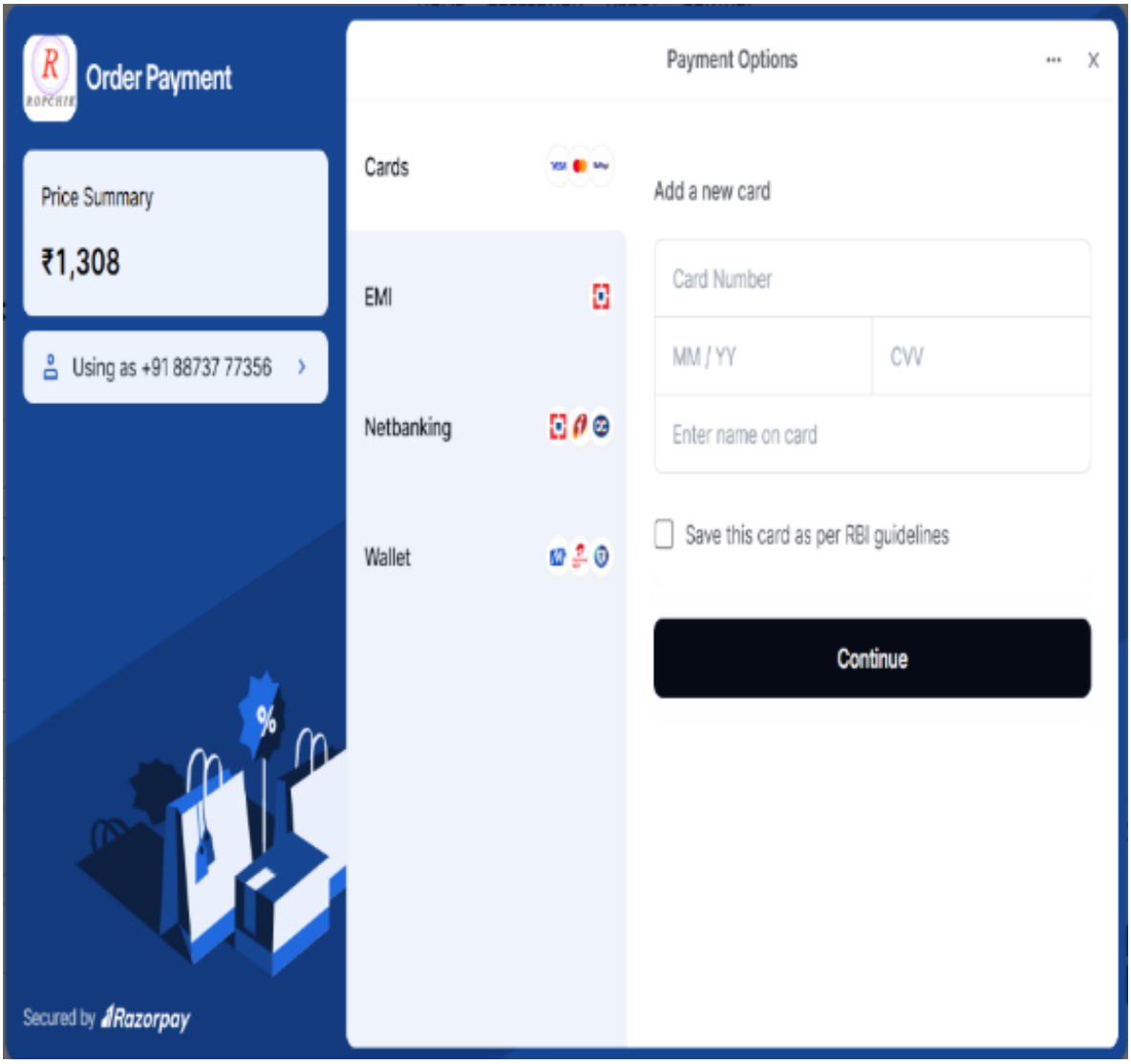


Cart/ Checkout Page: - The Cart page shows the items users want to buy, allowing them to change quantities or remove products. The Checkout page collects delivery details and payment preferences. It integrates securely with payment gateways like Razorpay, making the buying process simple and smooth, encouraging users to complete their purchases.

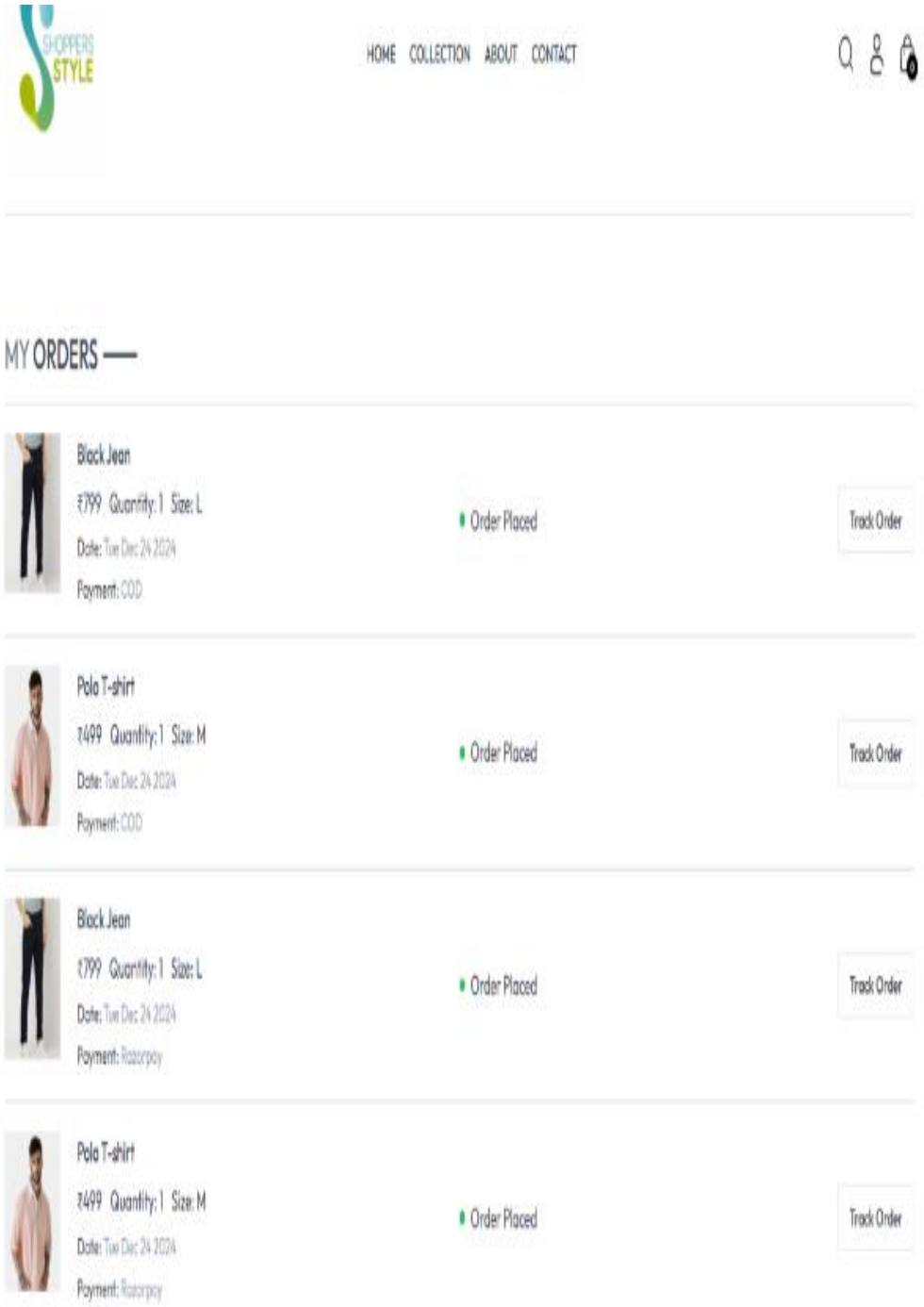




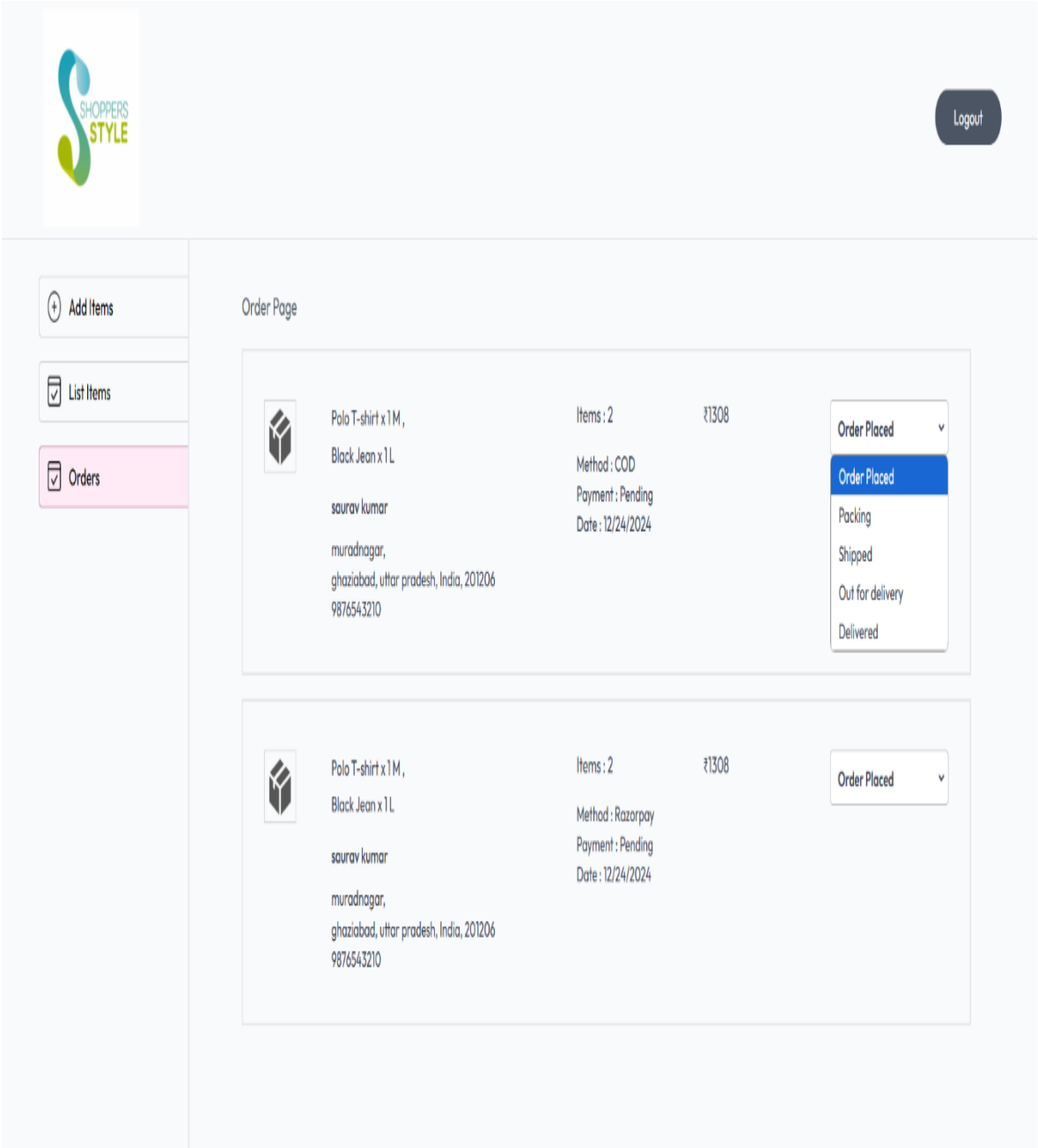
Payment Page: - The Payment page allows users to pay for their orders securely. It supports methods like Razorpay and Cash on Delivery. Transactions are processed safely, and users get updates on their payment status. The page is designed to make the process quick and easy while ensuring data security.



User Order Page: - The User Order page displays all the orders a user has placed. It shows details like order status, payment method, and delivery updates. Users can track active orders and check past purchases. It's easy to use and works well on all devices, helping users stay informed about their orders.



Admin Order Page:- The Admin Order page helps manage customer orders. It shows details about each order and lets administrators update the status, such as "Shipped" or "Delivered." This page makes it easy to monitor and handle orders efficiently, ensuring customers receive their items on time and without issues.



Chapter 7 : Coding

Login.jsx :-

```
import React, { useContext, useEffect, useState } from 'react'
import { ShopContext } from '../context/ShopContext';
import axios from 'axios';
import { toast } from 'react-toastify';
const Login = () => {
  const [currentState, setCurrentState] = useState('Login');
  const { token, setToken, navigate, backendUrl } = useContext(ShopContext)
  const [name,setName] = useState('')
  const [password,setPasword] = useState('')
  const [email,setEmail] = useState('')
  const onSubmitHandler = async (event) => {
      event.preventDefault();
      try {
        if (currentState === 'Sign Up') {
          const response = await axios.post(backendUrl +
'/api/user/register',{name,email,password})
          if (response.data.success) {
            setToken(response.data.token)
            localStorage.setItem('token',response.data.token)
          } else {
            toast.error(response.data.message)
```

```
        }
      } else {
        const response = await axios.post(backendUrl + '/api/user/login',
{email,password})
        if (response.data.success) {
          setToken(response.data.token)
          localStorage.setItem('token',response.data.token)
        } else {
          toast.error(response.data.message)
        }
      }
    } catch (error) {
      console.log(error)
      toast.error(error.message)
    }
  }
  useEffect(()=>{
    if (token) {
      navigate('/')
    }
  },[token])
  return (
    <form onSubmit={onSubmitHandler} className='flex flex-col items-center w-
[90%] sm:max-w-96 m-auto mt-14 gap-4 text-gray-800'>
      <div className='inline-flex items-center gap-2 mb-2 mt-10'>
```

```
            <p className='prata-regular text-3xl'>{currentState}</p>
            <hr className='border-none h-[1.5px] w-8 bg-gray-800' />
        </div>
        {currentState === 'Login' ? '' : <input
onChange={(e)=>setName(e.target.value)} value={name} type="text"
className='w-full px-3 py-2 border border-gray-800' placeholder='Name'
required/>}
        <input onChange={(e)=>setEmail(e.target.value)} value={email} type="email"
className='w-full px-3 py-2 border border-gray-800' placeholder='Email' required/>
        <input onChange={(e)=>setPasword(e.target.value)} value={password}
type="password" className='w-full px-3 py-2 border border-gray-800'
placeholder='Password' required/>
        <div className='w-full flex justify-between text-sm mt-[-8px]'>
            <p className=' cursor-pointer'>Forgot your password?</p>
            {
              currentState === 'Login'
              ? <p onClick={()=>setCurrentState('Sign Up')} className=' cursor-
pointer'>Create account</p>
              : <p onClick={()=>setCurrentState('Login')} className=' cursor-
pointer'>Login Here</p>
            }
        </div>
        <button className='bg-black text-white font-light px-8 py-2 mt-
4'>{currentState === 'Login' ? 'Sign In' : 'Sign Up'}</button>
    </form>
```

```
  )
}
export default Login
```

Home.jsx :-

```
import React from 'react'
import Hero from '../components/Hero'
import LatestCollection from '../components/LatestCollection'
import BestSeller from '../components/BestSeller'
import OurPolicy from '../components/OurPolicy'
import NewsletterBox from '../components/NewsletterBox'
const Home = () => {
  return (
    <div>
      <Hero />
      <LatestCollection/>
      <BestSeller/>
      <OurPolicy/>
      <NewsletterBox/>
    </div>
  )
}
```

Add.jsx :-

```
  export default Home
import React, { useState } from 'react'
import {assets} from '../assets/assets'
import axios from 'axios'
import { backendUrl } from '../App'
import { toast } from 'react-toastify'
const Add = ({token}) => {
  const [image1,setImage1] = useState(false)
  const [image2,setImage2] = useState(false)
  const [image3,setImage3] = useState(false)
  const [image4,setImage4] = useState(false)
   const [name, setName] = useState("");
   const [description, setDescription] = useState("");
   const [price, setPrice] = useState("");
   const [category, setCategory] = useState("Men");
   const [subCategory, setSubCategory] = useState("Topwear");
   const [bestseller, setBestseller] = useState(false);
   const [sizes, setSizes] = useState([]);
   const onSubmitHandler = async (e) => {
    e.preventDefault();
    try {
      const formData = new FormData()
      formData.append("name",name)
```

```
      formData.append("description",description)
      formData.append("price",price)
      formData.append("category",category)
      formData.append("subCategory",subCategory)
      formData.append("bestseller",bestseller)
      formData.append("sizes",JSON.stringify(sizes))
      image1 && formData.append("image1",image1)
      image2 && formData.append("image2",image2)
      image3 && formData.append("image3",image3)
      image4 && formData.append("image4",image4)
      const response = await axios.post(backendUrl +
"/api/product/add",formData,{headers:{token}})
      if (response.data.success) {
        toast.success(response.data.message)
        setName('')
        setDescription('')
        setImage1(false)
        setImage2(false)
        setImage3(false)
        setImage4(false)
        setPrice('')
      } else {
        toast.error(response.data.message)
      }
    } catch (error) {
```

```
      console.log(error);
      toast.error(error.message)
    }
  }
  return (
    <form onSubmit={onSubmitHandler} className='flex flex-col w-full items-start
gap-3'>
        <div>
          <p className='mb-2'>Upload Image</p>
          <div className='flex gap-2'>
            <label htmlFor="image1">
              <img className='w-20' src={!image1 ? assets.upload_area :
URL.createObjectURL(image1)} alt="" />
              <input onChange={(e)=>setImage1(e.target.files[0])} type="file"
id="image1" hidden/>
            </label>
            <label htmlFor="image2">
              <img className='w-20' src={!image2 ? assets.upload_area :
URL.createObjectURL(image2)} alt="" />
              <input onChange={(e)=>setImage2(e.target.files[0])} type="file"
id="image2" hidden/>
            </label>
            <label htmlFor="image3">
              <img className='w-20' src={!image3 ? assets.upload_area :
URL.createObjectURL(image3)} alt="" />
```

```
            <input onChange={(e)=>setImage3(e.target.files[0])} type="file"
id="image3" hidden/>
          </label>
          <label htmlFor="image4">
            <img className='w-20' src={!image4 ? assets.upload_area :
URL.createObjectURL(image4)} alt="" />
            <input onChange={(e)=>setImage4(e.target.files[0])} type="file"
id="image4" hidden/>
          </label>
        </div>
      </div>
      <div className='w-full'>
        <p className='mb-2'>Product name</p>
        <input onChange={(e)=>setName(e.target.value)} value={name}
className='w-full max-w-[500px] px-3 py-2' type="text" placeholder='Type here'
required/>
      </div>
      <div className='w-full'>
        <p className='mb-2'>Product description</p>
        <textarea onChange={(e)=>setDescription(e.target.value)} value={description}
className='w-full max-w-[500px] px-3 py-2' type="text" placeholder='Write content
here' required/>
      </div>
      <div className='flex flex-col sm:flex-row gap-2 w-full sm:gap-8'>
          <div>
```

```
            <p className='mb-2'>Product category</p>
            <select onChange={(e) => setCategory(e.target.value)} className='w-full
px-3 py-2'>
                <option value="Men">Men</option>
                <option value="Women">Women</option>
                <option value="Kids">Kids</option>
            </select>
          </div>
          <div>
            <p className='mb-2'>Sub category</p>
            <select onChange={(e) => setSubCategory(e.target.value)} className='w-
full px-3 py-2'>
                <option value="Topwear">Topwear</option>
                <option value="Bottomwear">Bottomwear</option>
                <option value="Winterwear">Winterwear</option>
            </select>
          </div>
          <div>
            <p className='mb-2'>Product Price</p>
            <input onChange={(e) => setPrice(e.target.value)} value={price}
className='w-full px-3 py-2 sm:w-[120px]' type="Number" placeholder='25' />
          </div>
        </div>
        <div>
          <p className='mb-2'>Product Sizes</p>
```

```
<div className='flex gap-3'>
  <div onClick={()=>setSizes(prev => prev.includes("S") ? prev.filter( item =>
item !== "S") : [...prev,"S"])}>
    <p className={`${sizes.includes("S") ? "bg-pink-100" : "bg-slate-200" } px-
3 py-1 cursor-pointer`}>S</p>
  </div>
  <div onClick={()=>setSizes(prev => prev.includes("M") ? prev.filter( item =>
item !== "M") : [...prev,"M"])}>
    <p className={`${sizes.includes("M") ? "bg-pink-100" : "bg-slate-200" }
px-3 py-1 cursor-pointer`}>M</p>
  </div>
  <div onClick={()=>setSizes(prev => prev.includes("L") ? prev.filter( item =>
item !== "L") : [...prev,"L"])}>
    <p className={`${sizes.includes("L") ? "bg-pink-100" : "bg-slate-200" } px-
3 py-1 cursor-pointer`}>L</p>
  </div>
  <div onClick={()=>setSizes(prev => prev.includes("XL") ? prev.filter( item =>
item !== "XL") : [...prev,"XL"])}>
    <p className={`${sizes.includes("XL") ? "bg-pink-100" : "bg-slate-200" }
px-3 py-1 cursor-pointer`}>XL</p>
  </div>
  <div onClick={()=>setSizes(prev => prev.includes("XXL") ? prev.filter( item
=> item !== "XXL") : [...prev,"XXL"])}>
    <p className={`${sizes.includes("XXL") ? "bg-pink-100" : "bg-slate-200" }
px-3 py-1 cursor-pointer`}>XXL</p>
```

```
            </div>
          </div>
        </div>
        <div className='flex gap-2 mt-2'>
          <input onChange={() => setBestseller(prev => !prev)} checked={bestseller}
type="checkbox" id='bestseller' />
          <label className='cursor-pointer' htmlFor="bestseller">Add to
bestseller</label>
        </div>
        <button type="submit" className='w-28 py-3 mt-4 bg-black text-
white'>ADD</button>
    </form>
  )
}
export default Add
```

Order.jsx :-

```
import React from 'react'
import { useEffect } from 'react'
import { useState } from 'react'
import axios from 'axios'
import { backendUrl, currency } from '../App'
import { toast } from 'react-toastify'
import { assets } from '../assets/assets'
const Orders = ({ token }) => {
  const [orders, setOrders] = useState([])
  const fetchAllOrders = async () => {
    if (!token) {
      return null;
    }
    try {
      const response = await axios.post(backendUrl + '/api/order/list', {}, { headers: {
token } })
      if (response.data.success) {
        setOrders(response.data.orders.reverse())
      } else {
        toast.error(response.data.message)
      }
```

```
    } catch (error) {
      toast.error(error.message)
    }
  }
  const statusHandler = async ( event, orderId ) => {
    try {
      const response = await axios.post(backendUrl + '/api/order/status' , {orderId,
status:event.target.value}, { headers: {token}})
      if (response.data.success) {
        await fetchAllOrders()
      }
    } catch (error) {
      console.log(error)
      toast.error(response.data.message)
    }
  }
  useEffect(() => {
    fetchAllOrders();
  }, [token])
  return (
    <div>
      <h3>Order Page</h3>
      <div>
        {
          orders.map((order, index) => (
```

```
<div className='grid grid-cols-1 sm:grid-cols-[0.5fr_2fr_1fr] lg:grid-cols-[0.5fr_2fr_1fr_1fr_1fr] gap-3 items-start border-2 border-gray-200 p-5 md:p-8 my-3 md:my-4 text-xs sm:text-sm text-gray-700' key={index}>
  <img className='w-12' src={assets.parcel_icon} alt="" />
  <div>
    <div>
      {order.items.map((item, index) => {
        if (index === order.items.length - 1) {
          return <p className='py-0.5' key={index}> {item.name} x {item.quantity} <span> {item.size} </span> </p>
        }
        else {
          return <p className='py-0.5' key={index}> {item.name} x {item.quantity} <span> {item.size} </span> ,</p>
        }
      })}
    </div>
    <p className='mt-3 mb-2 font-medium'>{order.address.firstName + " " + order.address.lastName}</p>
    <div>
      <p>{order.address.street + ","}</p>
      <p>{order.address.city + ", " + order.address.state + ", " + order.address.country + ", " + order.address.zipcode}</p>
    </div>
    <p>{order.address.phone}</p>
```

```
            </div>
            <div>
              <p className='text-sm sm:text-[15px]'>Items : {order.items.length}</p>
              <p className='mt-3'>Method : {order.paymentMethod}</p>
              <p>Payment : { order.payment ? 'Done' : 'Pending' }</p>
              <p>Date : {new Date(order.date).toLocaleDateString()}</p>
            </div>
            <p className='text-sm sm:text-[15px]'>{currency}{order.amount}</p>
            <select onChange={(event)=>statusHandler(event,order._id)}
value={order.status} className='p-2 font-semibold'>
              <option value="Order Placed">Order Placed</option>
              <option value="Packing">Packing</option>
              <option value="Shipped">Shipped</option>
              <option value="Out for delivery">Out for delivery</option>
              <option value="Delivered">Delivered</option>
            </select>
          </div>
        ))
      }
    </div>
  </div>
 )
}
export default Orders
```

CardController.js :-

```
import userModel from "../models/userModel.js"
// add products to user cart
const addToCart = async (req,res) => {
    try {
        const { userId, itemId, size } = req.body
        const userData = await userModel.findById(userId)
        let cartData = await userData.cartData;
        if (cartData[itemId]) {
            if (cartData[itemId][size]) {
                cartData[itemId][size] += 1
            }
            else {
                cartData[itemId][size] = 1
            }
        } else {
            cartData[itemId] = {}
            cartData[itemId][size] = 1
        }
        await userModel.findByIdAndUpdate(userId, {cartData})
        res.json({ success: true, message: "Added To Cart" })
    } catch (error) {
        console.log(error)
```

```
        res.json({ success: false, message: error.message })
    }
}
// update user cart
const updateCart = async (req,res) => {
    try {
        const { userId ,itemId, size, quantity } = req.body
        const userData = await userModel.findById(userId)
        let cartData = await userData.cartData;
        cartData[itemId][size] = quantity
        await userModel.findByIdAndUpdate(userId, {cartData})
        res.json({ success: true, message: "Cart Updated" })
    } catch (error) {
        console.log(error)
        res.json({ success: false, message: error.message })
    }
}
// get user cart data
const getUserCart = async (req,res) => {
    try {
        const { userId } = req.body
        const userData = await userModel.findById(userId)
        let cartData = await userData.cartData;
        res.json({ success: true, cartData })
    } catch (error) {
```

```
        console.log(error)
        res.json({ success: false, message: error.message })
    }
}
export { addToCart, updateCart, getUserCart }
```

UserController.js:-

```
import validator from "validator";
import bcrypt from "bcrypt"
import jwt from 'jsonwebtoken'
import userModel from "../models/userModel.js";
const createToken = (id) => {
    return jwt.sign({ id }, process.env.JWT_SECRET)
}
// Route for user login
const loginUser = async (req, res) => {
    try {
        const { email, password } = req.body;
        const user = await userModel.findOne({ email });
        if (!user) {
            return res.json({ success: false, message: "User doesn't exists" })
        }
        const isMatch = await bcrypt.compare(password, user.password);
        if (isMatch) {
            const token = createToken(user._id)
            res.json({ success: true, token })
        }
        else {
            res.json({ success: false, message: 'Invalid credentials' })
```

```
        }
    } catch (error) {
        console.log(error);
        res.json({ success: false, message: error.message })
    }
}
// Route for user register
const registerUser = async (req, res) => {
    try {
        const { name, email, password } = req.body;
        // checking user already exists or not
        const exists = await userModel.findOne({ email });
        if (exists) {
            return res.json({ success: false, message: "User already exists" })
        }
        // validating email format & strong password
        if (!validator.isEmail(email)) {
            return res.json({ success: false, message: "Please enter a valid email" })
        }
        if (password.length < 8) {
            return res.json({ success: false, message: "Please enter a strong password" })
        }
        // hashing user password
        const salt = await bcrypt.genSalt(10)
        const hashedPassword = await bcrypt.hash(password, salt)
```

```
        const newUser = new userModel({
            name,
            email,
            password: hashedPassword
        })
        const user = await newUser.save()
        const token = createToken(user._id)
        res.json({ success: true, token })
    } catch (error) {
        console.log(error);
        res.json({ success: false, message: error.message })
    }
}
// Route for admin login
const adminLogin = async (req, res) => {
    try {
        const {email,password} = req.body
        if (email === process.env.ADMIN_EMAIL && password ===
process.env.ADMIN_PASSWORD) {
            const token = jwt.sign(email+password,process.env.JWT_SECRET);
            res.json({success:true,token})
        } else {
            res.json({success:false,message:"Invalid credentials"})
        }
    } catch (error) {
```

```
        console.log(error);
        res.json({ success: false, message: error.message })
    }
}
export { loginUser, registerUser, adminLogin }
```

Chapter 8: Conclusion

Our project, a clothing e-commerce website built using the MERN stack (MongoDB, Express.js, React.js, Node.js), is designed to provide a smooth and user-friendly shopping experience. We focused on creating a platform that is efficient, reliable, and easy to use, keeping the needs of online shoppers in mind.

The idea for this project came from recognizing the growing demand for online clothing stores. Our goal was to build a solution that is not only functional but also visually appealing and tailored to modern users' needs. The platform includes essential features like browsing, filtering, and sorting products, selecting sizes, managing a cart, and placing orders. We also ensured secure user login and payment options by integrating gateways like Razorpay.

We followed a clear plan, from defining goals and designing the system to testing every feature thoroughly. This ensured that the website worked smoothly and met user expectations.

In the future, we aim to improve the platform by adding advanced features like personalized product recommendations and better analytics. This project reflects our dedication to creating practical and innovative solutions in e-commerce.

Chapter 9: Reference

- https://www.mongodb.com/
- https://legacy.reactjs.org/docs/getting-started.html
- https://nodejs.org/docs/latest/api/http://www.jdbc-tutorial.com/
- https://expressjs.com/
- https://www.npmjs.com/package/json-server
- https://www.npmjs.com/package/react-hook-form
- https://github.com/
- https://www.tutorialspoint.com/java/
- https://www.w3schools.com/
- https://dummyjson.com/
- https://rapidapi.com/utelly/api/utelly
- https://www.postman.com/