Shoppers Style
AI-powered e-commerce website

A PROJECT REPORT
for
Project (KCA451)
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

(APRIL 2025)

CERTIFICATE

Certified that **Saurav Kumar Sinha (2300290140165)** has carried out the project work having "**Shoppers Style (AI-powered e-commerce website)**" (**Project-KCA451**) 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.

This is to certify that the above statement made by the candidate is correct to the best of my knowledge.

Date:

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

Dr. Akash Rajak
Dean
Department of Computer Applications
KIET Group of Institutions, Ghaziabad

Shoppers Style
AI-powered e-commerce website

ABSTRACT

This project is about building an online shopping platform using the MERN stack (MongoDB, Express.js, React.js, Node.js) with AI features to improve the shopping experience. It will include all the basic e-commerce functions like user login, browsing products, managing shopping carts, checkout, and tracking orders. These features will allow users to shop easily and securely.

Along with these basic features, the platform will also have smart AI tools. The AI will automatically generate product descriptions, suggest products based on what users like, and provide chatbots to help customers answer questions. Users can also search for products by uploading images and prices will change based on demand and market trends.

The frontend will be built with React.js, which will provide a seamless and interactive experience, while the backend will use Node.js and Express.js to manage server-side functions. AI features will be added using serverless technologies for better scalability. This project shows how AI can improve online shopping by making it smarter, more personalized, and easier to use.

Keywords: E-commerce, MERN stack, AI, Machine learning, OpenAI, MongoDB, React.js, Node.js, Chatbot.

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. Akash Rajak**, Professor and Dean, 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

LIST OF FIGURES

Chapter 1: Introduction

1.1 Overview

Shoppers Style is an AI-powered e-commerce website developed using the MERN stack. It aims to deliver a seamless online shopping experience by combining modern web technologies with intelligent features. The platform supports core functionalities like user authentication, product browsing, cart operations, checkout, and order tracking, enhanced with AI capabilities such as smart recommendations, chatbot assistance, and dynamic pricing.

The website includes important features like:

- User login and sign-up
- Looking at products
- Adding items to the cart
- Buying products and tracking orders

It also has smart AI features like:

- Showing product suggestions based on what users like
- A chatbot to help answer questions
- Changing prices based on demand and stock

1.2 Background Study

Traditional e-commerce platforms rely heavily on manual product updates and user navigation. By leveraging AI, Shoppers Style introduces automation in content generation, behavior-based recommendations, and intelligent customer support. This innovation improves user satisfaction, increases conversion rates, and streamlines backend operations.

Shoppers Style uses AI to make things smarter and easier. It can:

- Write product details on its own,
- Suggest items based on what users like or bought before, and
- Help customers with a chatbot.

1.3 Project Planning

The project was divided into four main steps:

1. **Understanding Requirements**: - We first found out what users and admins need from the website.
2. **Design**: - We planned how the website will look and how the information will be saved and shared.
3. **Development**: - We wrote the code for the frontend (what users see), the backend (server), and the database (where data is stored).
4. **Testing and Deployment**: - We checked if everything works properly, fixed any problems, and then put the website online so people can use it.

1.4 Purpose and Objectives

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

1. Let users easily browse, search, and buy clothing items.
2. Offer safe and secure payment options, including online payment and Cash on Delivery.
3. Allow admins to manage products, track orders, and update stock smoothly.
4. Be reliable, fast, and able to handle more users and products as the business grows.
5. Use Artificial Intelligence (AI) to make shopping better by:
6. Showing personalized product recommendations based on what users like.
7. Generating automatic product descriptions using AI.
8. Offering a chatbot that answers customer questions quickly.
9. Allowing users to search for products by uploading a picture.

Chapter 2: System Design

2.1 Design

The platform adopts a modular and component-based design using React.js on the frontend and REST API on the backend. This makes it easy to update, fix, and enhance the website in the future. It ensures reusability, maintainability, and scalability.

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:

1. Frontend: Built with React.js for an interactive and responsive user interface.
2. Backend: Developed using Node.js and Express.js for handling APIs and business logic.
3. Database: MongoDB stores data securely, including user profiles, products, orders, and payment records.
4. Payment Gateway: Razorpay integration for secure online transactions.
5. AI Integration: Artificial Intelligence (AI) is used to improve the shopping experience by:
   - Automatically generating product descriptions.
   - Suggesting products based on user behavior.
   - Offering chatbot support for customer queries.
   - Adjusting prices based on demand and trends.

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 and AI APIs.
- 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 part explains how different people and systems interact with the **Shoppers Style** website. These interactions are usually shown in a **Context Diagram**, which gives a big-picture view of how everything connects.

Customers use the website to:

- Browse products
- Add items to the cart
- Make secure payments
- Get help from AI:
    - Shows products they might like (based on past behavior)
    - Changes prices based on demand and trends (smart pricing)

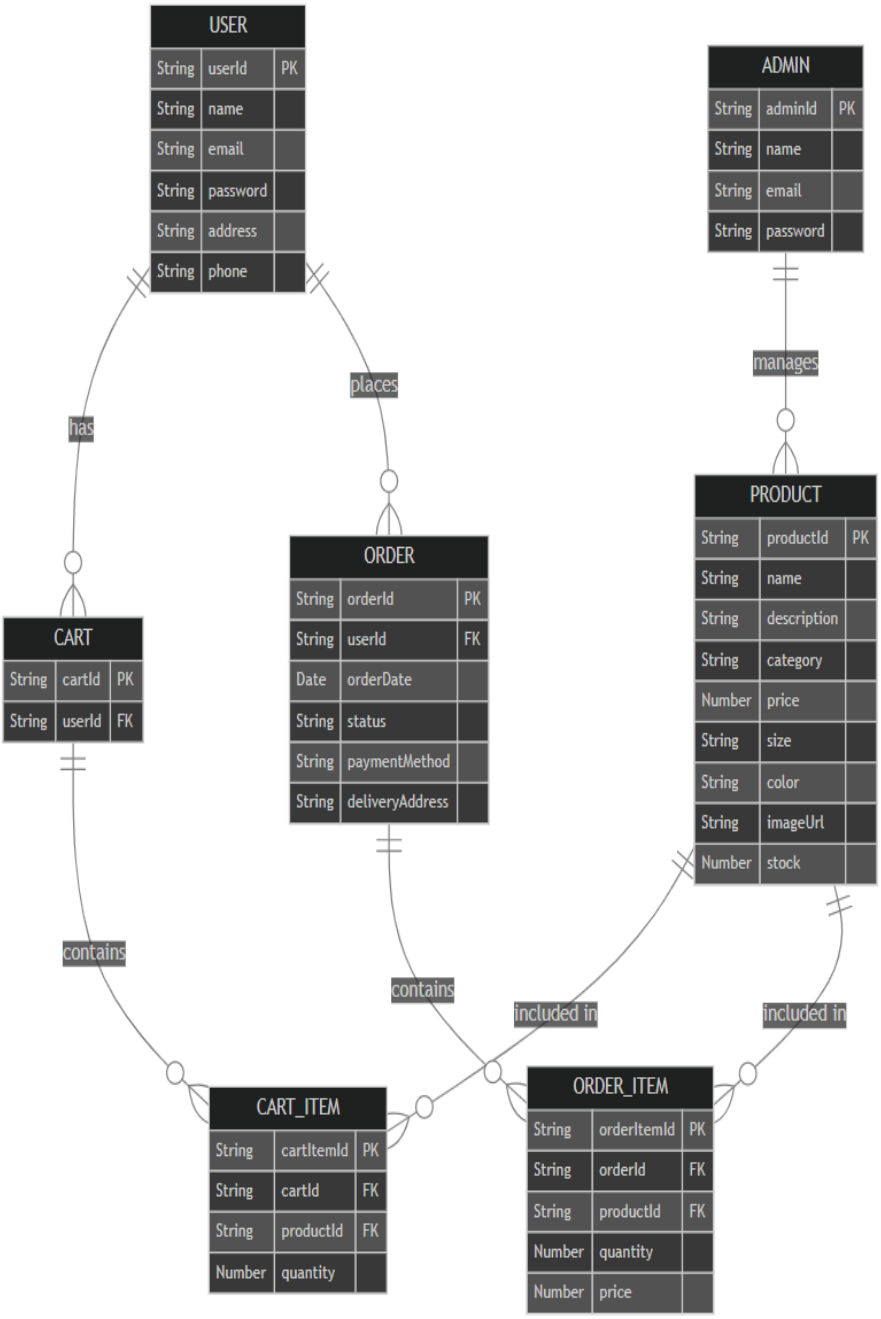
Administrators use the admin panel to:

- Add, update, or remove products
- Track and manage customer orders
- Use AI tools to:
    - Change prices automatically
    - See which products are popular

Payment Gateway (Razorpay and Cash on Delivery**)** is used to:

- Make sure payments are safe
- Accept cards, UPI, Cash on Delivery and wallets
- Connect smoothly with the website

2.7 Design Diagram:



2.7 Design Diagram: - The design diagram shows how all parts of the system connect and work together:

- **Users**: There are two main users:
  - **Customers** can browse products, get AI-based recommendations, add items to the cart, and make payments.
  - **Admins** can manage products, view orders, and use AI tools to adjust prices or track trends.

- **Website**: Built with **React.js**, it provides an interactive user interface where users perform actions like browsing or managing products.

- **Backend**: Handled by **Node.js and Express.js**, it connects the website to the database and processes all business logic.

- **Database**: **MongoDB** stores user details, product information, orders, and payment records securely.

- **AI Features**:
  - **Smart Recommendations**: Suggests products based on user behavior.
  - **Chatbot**: Assists users with questions or order tracking.
  - **Dynamic Pricing**: Adjusts prices based on demand and trends.

- **Payment Gateway**: **Razorpay** ensures safe and smooth online payments.

Chapter 3: Hardware and Software Requirements

3.1 Hardware Requirements

To build and run the AI-powered Shoppers Style eCommerce website, the following hardware is needed:

Development (Creating the website)

- **Processor**: Intel Core i5 or better
- **RAM**: At least 8 GB (for smooth working)
- **Storage**: 256 GB SSD or more (to store files fast)
- **Monitor**: Screen with 1920x1080 resolution (for better view while designing)

Deployment Server (Running the live website)

- **Processor**: Quad-core CPU (like Intel Xeon or similar)
- **RAM**: 16 GB or more (to handle many users at once)
- **Storage**: 500 GB SSD or higher (to save data fast)
- **Internet**: High-speed connection (to support real-time AI features and user requests)

3.2 Software Requirements

To build, test, and run the AI-powered Shoppers Style eCommerce website, we use the following software tools:

Development---Tools

- **Code Editor**: Visual Studio Code (for writing code)
- **Version Control**: Git and GitHub (to save and track code changes)

Frontend---(What users see)

- **Language**: JavaScript
- **Framework**: React.js (to build the website design)
- **Styling**: CSS and Bootstrap (to make the website look good)

Backend (The part that works in the background)

- **Runtime**: Node.js
- **Framework**: Express.js (to manage how data flows on the site)

Database

- **MongoDB** – stores all user info, products, orders, and payment records

AI Tools

- **Recommendation Engine** – suggests products based on user behavior
- **Dynamic Pricing Tool** – changes prices based on trends and demand
- **Chatbot Tool** – helps customers with smart, instant support

Testing Tool

- **Postman** – used to test if APIs (connections between frontend and backend) are working

Deployment

- **Frontend Hosting**: Vercel (to put the website live on the internet)

Other Tools

- **Payment Gateway**: Razorpay (to make secure online payments)
- **Browser**: Google Chrome or similar (for testing and using the site)

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.

- **Frontend** – How the website looks and works for users
- **Backend** – Handles the logic and connects everything
- **Database** – Stores user, product, and order data
- **Deployment** – Makes the website live on the internet
- **AI Tools** – Adds smart features like recommendations and dynamic pricing

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 AI-Integrated Features :

1. Smart Product Recommendations

- AI suggests products to users based on their browsing history, preferences, and popular trends.

2. Dynamic Pricing

- Prices of products automatically adjust based on demand, user interest, and seasonal trends using AI algorithms.

3. AI Chatbot Assistance

- An AI-powered chatbot answers customer queries instantly, helps with product search, and supports order tracking.

4. AI in Admin Dashboard

- AI tools help admins:
  - Analyze product popularity.
  - Adjust prices based on market trends.

5. Personalized User Experience

- Each user sees tailored content, banners, and deals based on their behavior and interests.

6. Automated Inventory Insights

- AI predicts stock needs and alerts admins about popular products and low inventory levels.

7. Visual Search (Optional Future Scope)

- Users can upload a photo to search for similar-looking clothing items using AI-powered image recognition.

Chapter :5 Methodology

The project was developed using the Agile method, focusing on teamwork and delivering features step by step. Each part of the project was done in short stages (sprints) with feedback at the end of each stage.

5.1 Requirement Gathering and Research

In the beginning, we researched what users want and checked what similar platforms offer. We focused on AI-powered features like product recommendations and chatbots.

5.2 System and UI Design

The system was designed to be modular, and UI designs were created using tools to ensure the platform is easy to use and looks good.

5.3 Development of Core Features

The main features, like user login, product catalog, and shopping cart, were built using APIs. User authentication was done securely with JWT.

5.4 AI Integration and Training

AI features were added, like:

- Product descriptions generated using OpenAI's GPT model.
- Product recommendations based on what users view or buy.
- Chatbot powered by AI to answer user questions.
- Dynamic pricing that changes based on demand and stock levels.

5.5 Testing

The system was tested to make sure everything works correctly. Both manual and automated tests were used to check the platform's functionality, performance, and AI features.

5.6 Deployment and User Feedback

After deployment, real users tested the platform, and feedback was collected to make improvements. The development was iterative, meaning updates were made quickly based on user feedback.

Chapter :6 Data Flow Diagram

The **Data Flow Diagram (DFD)** shows how data moves through the **Shoppers Style** system. This diagram is essential during the design phase and helps visualize how users, the system, and AI features interact with one another.

- **Customers:**
  - Browse products.
  - Add to cart and checkout.
  - Receive AI-based product recommendations.
  - View dynamic pricing adjusted by AI based on trends and demand.

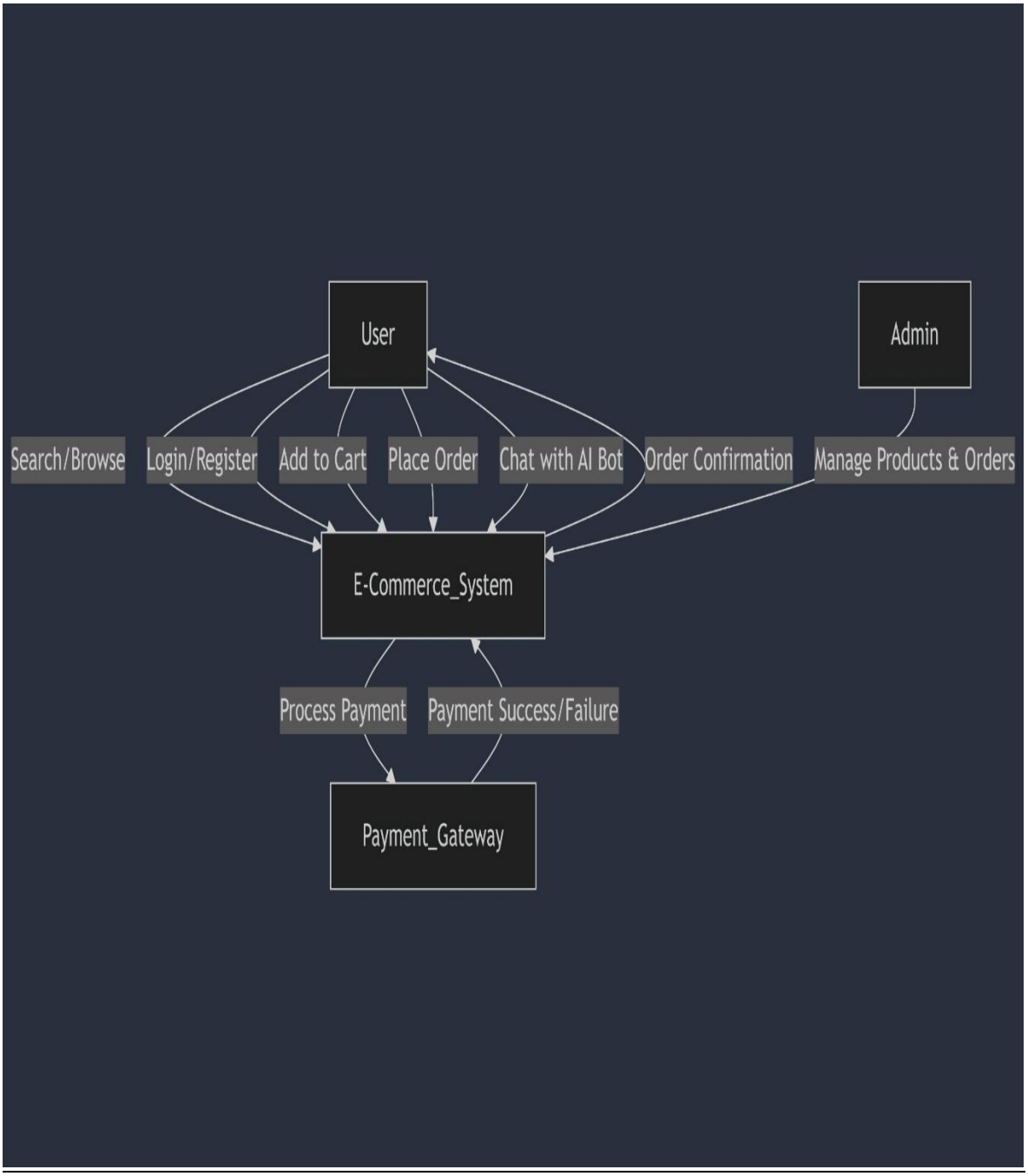
- **AI Module (New Component):**
  - Collects and analyzes user behavior.
  - Suggests personalized products.
  - Adjusts product prices dynamically.
  - Helps predict trends for admin insights.

- **Administrator:**
  - Manages products and orders.
  - Uses AI analytics dashboard to track top products and forecast demand.
  - Applies AI-suggested pricing strategies.

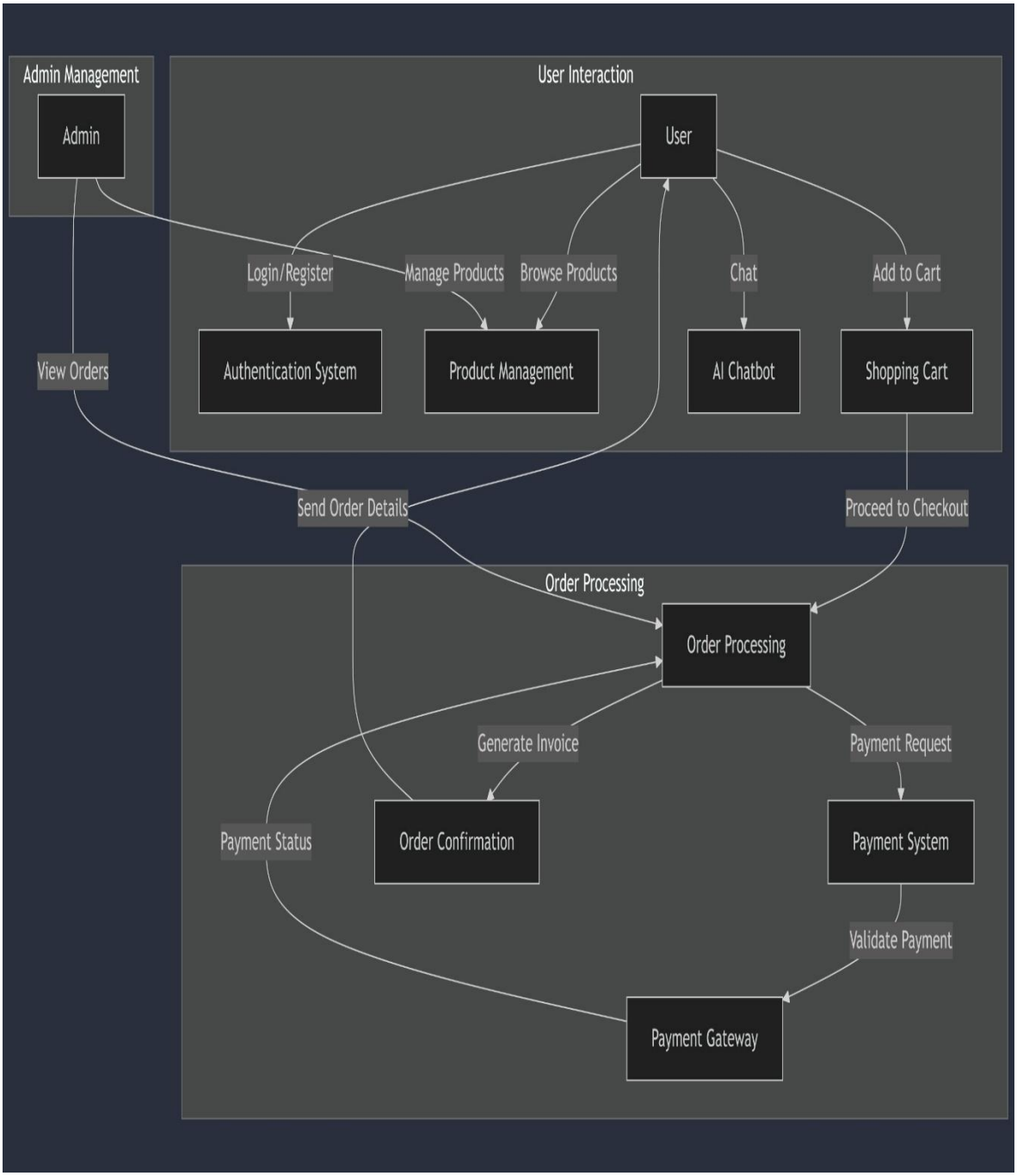
- **Database:**
  - Stores user info, product details, AI logs, and order records.

- **Payment Gateway (Razorpay):**
  - Processes secure payments.

6.1 Level 0 DFD: -



6.2 Level 1 DFD: -



Chapter 7: Entity Relationship Diagram

The Entity-Relationship Diagram (ERD) provides a high-level, conceptual view of the data used in the Shoppers Style eCommerce platform. It showcases entities, their attributes, and relationships, helping in the logical structuring of data storage and flow across the system.

Key Entities and Relationships: -

1. User

- Attributes: UserID (PK), Name, Email, Password, Address
- Relationships:
    - Can place multiple orders
    - Viewed by AI to generate personalized recommendations

2. Product

- Attributes: ProductID (PK), Name, Description, Price, Category, Stock
- Relationships:
    - Added by Admin
    - Linked with Order Items
    - Analyzed by AI for price adjustments and trends

3. Order

- Attributes: OrderID (PK), UserID (FK), Date, TotalAmount, PaymentStatus
- Relationships:
    - Placed by User
    - Contains multiple Order Items

4. OrderItem

- Attributes: ItemID (PK), OrderID (FK), ProductID (FK), Quantity, Price
- Relationships:
    - Links Orders to Products

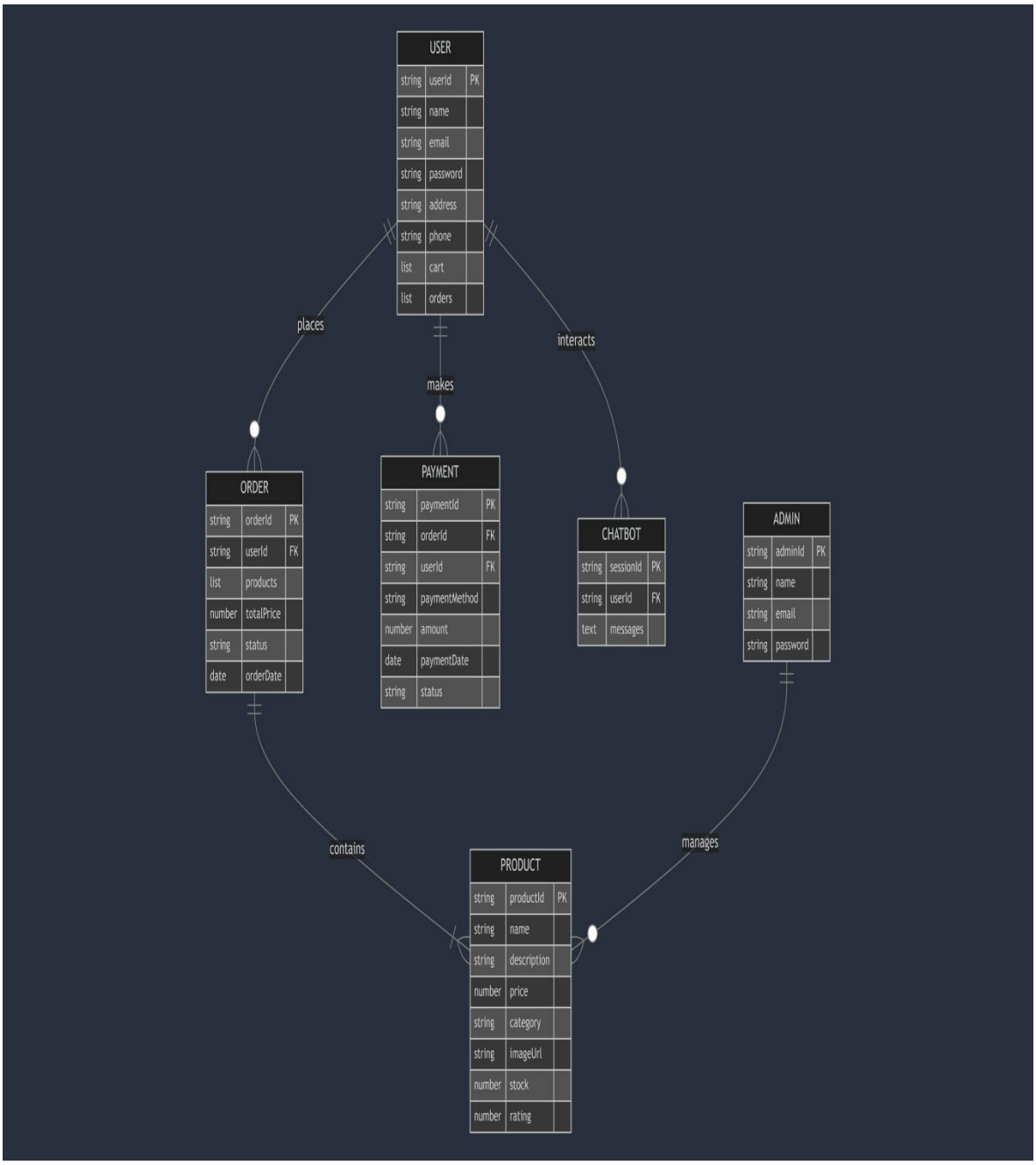
5. Admin

- Attributes: AdminID (PK), Name, Email, Password
- Relationships:
    - Manages Products
    - Views Reports generated by AI tools

6. AI Recommendation

- Attributes: RecID (PK), UserID (FK), ProductID (FK), Score
- Purpose:
    - Stores AI-generated suggestions for users

7.0 Entity Relationship Diagram



Chapter 8: Project Database & Screenshots

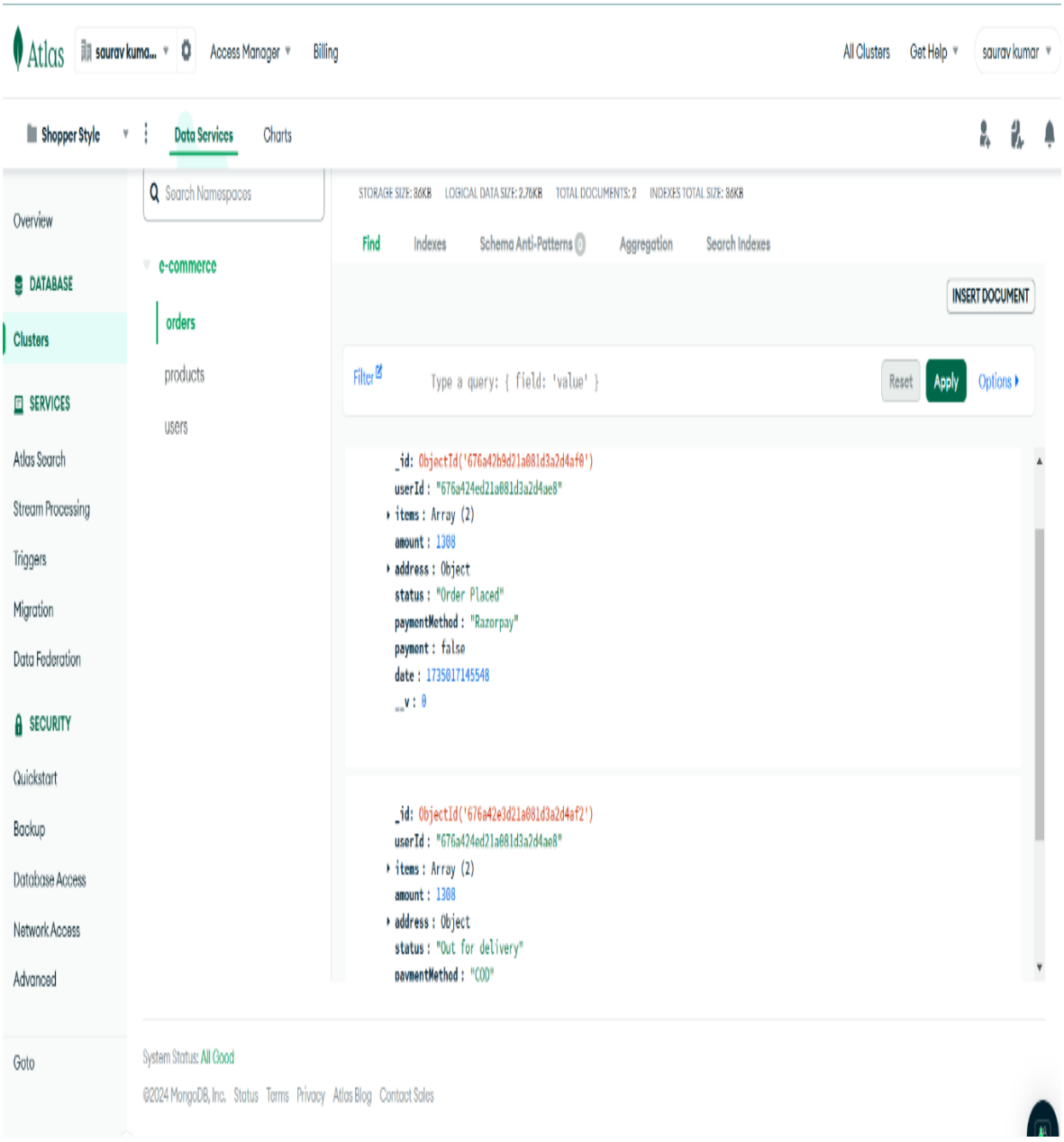
This chapter describes the structure and design of the database used for the eCommerce project, outlining all the tables (or MongoDB collections) and their respective fields. The database is central to managing data for users, products, categories, orders, and other key aspects of the project.

8.0 Database Design

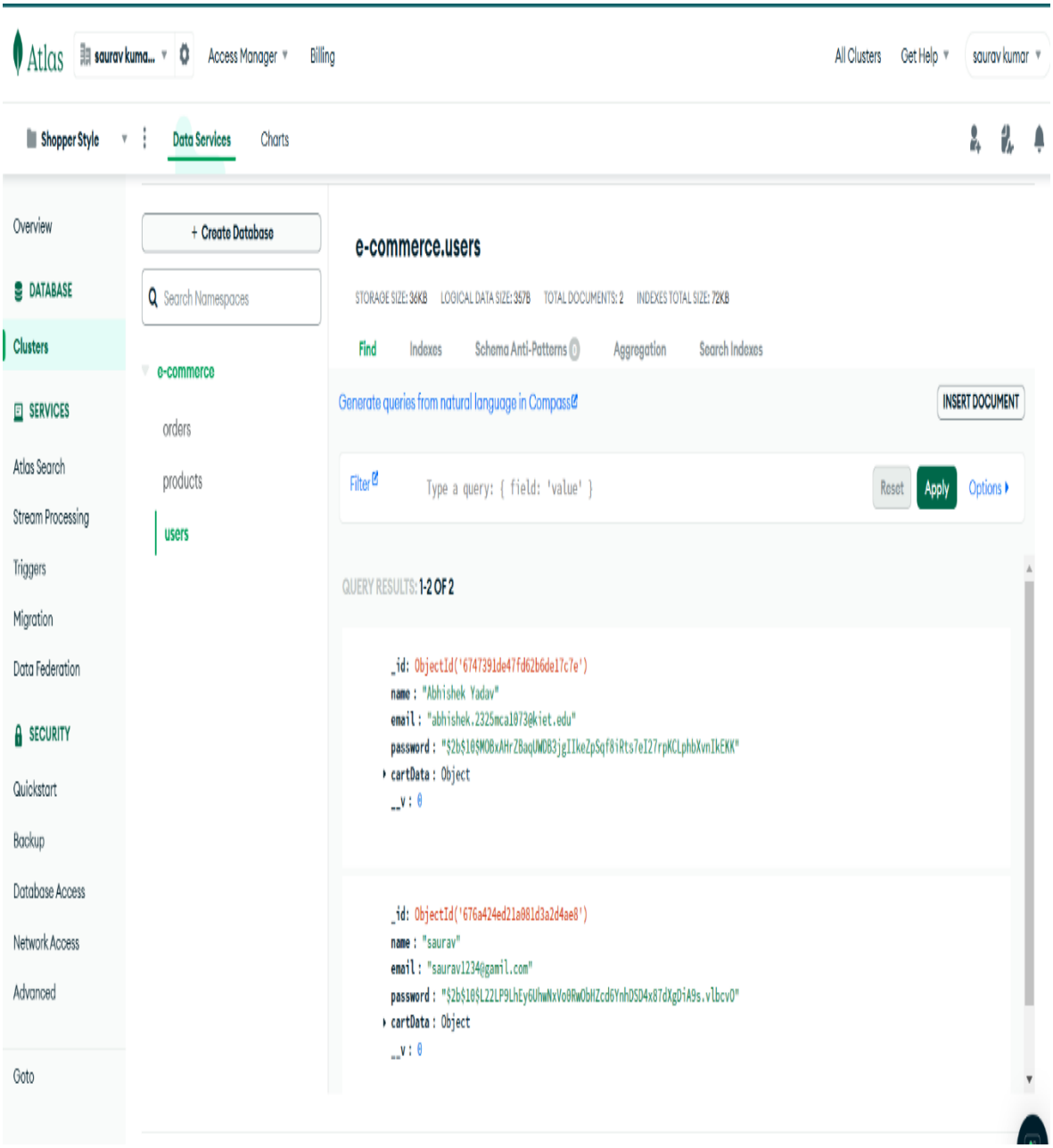
The eCommerce database is designed using **MongoDB**, a NoSQL database that provides flexibility and scalability. The database schema ensures efficient data storage, quick retrieval, and proper relationships between various entities like users, products, and orders. Key design considerations include:

- **Document-Based Structure**: Collections instead of traditional tables (like Users, Products, Orders, etc.).
- **Normalization:** Data is organized to reduce duplication and ensure consistency.
- **Security Measures:** Passwords are encrypted, and payment details are handled securely through third-party services.
- **AI-Ready Schema**: The structure supports AI features like storing recommendation scores and tracking dynamic pricing.

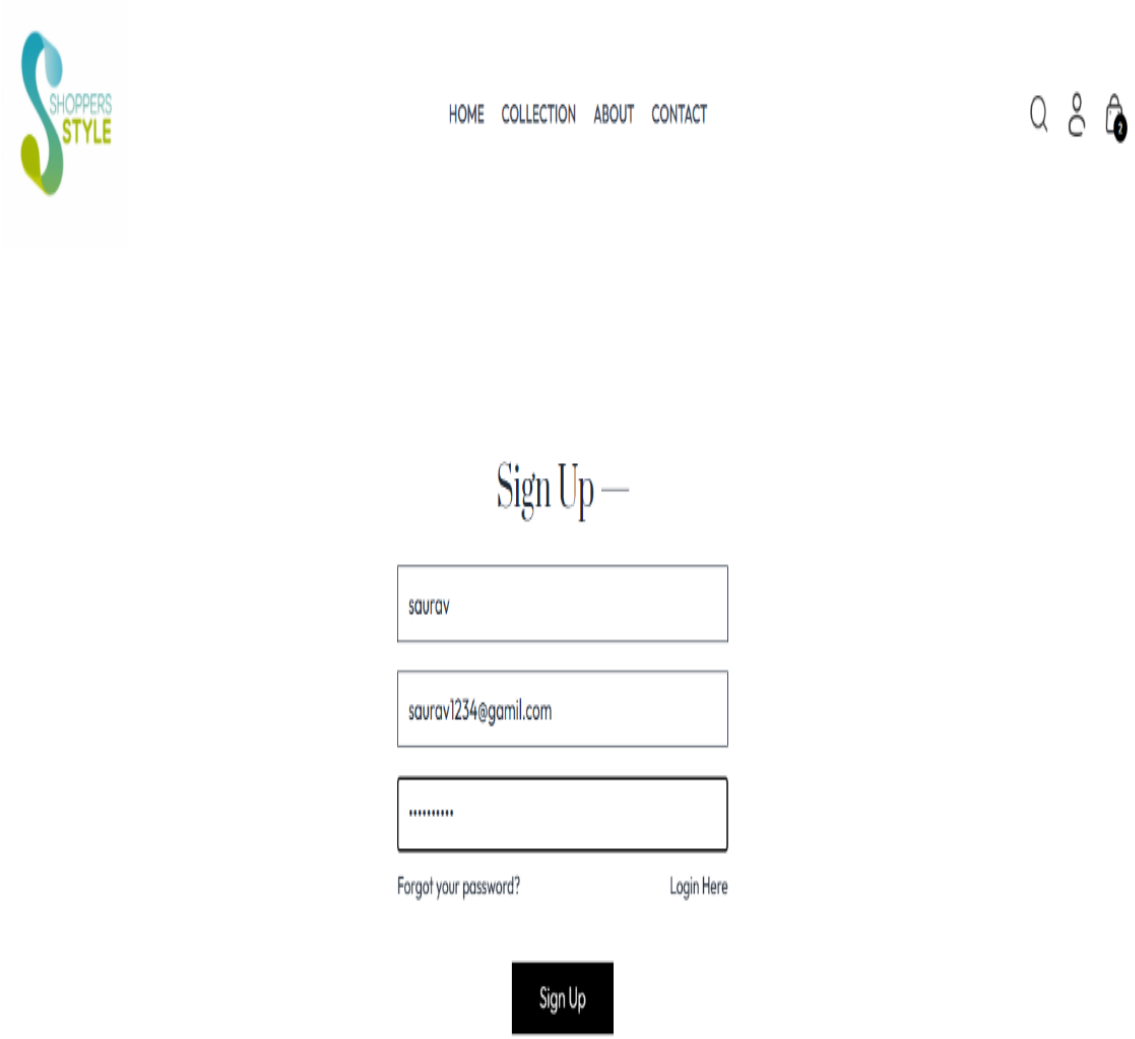
8.0.1 <u>Order Database: -</u> The Order database keeps track of customer orders, including product details, quantities, sizes, payment status, and delivery updates. It helps the system follow an order from start to finish, ensuring smooth management. Administrators can update orders and track their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



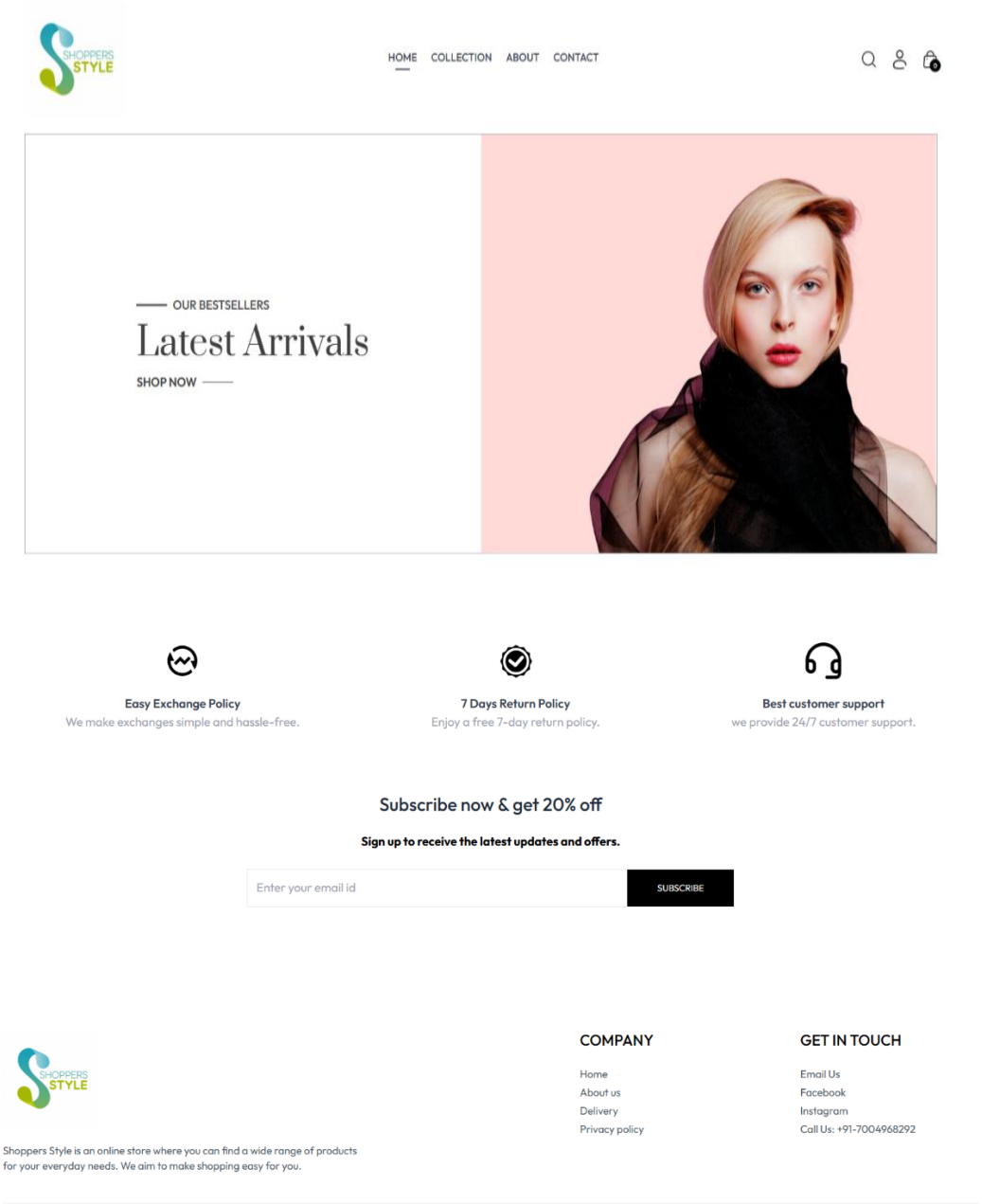
8.0.1 <u>User Database: -</u> The User database stores user information, such as their name, email, and securely hashed passwords. It also keeps track of their shopping carts. This database is essential for logging in users, personalizing their experience, and securely connecting them to other platform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



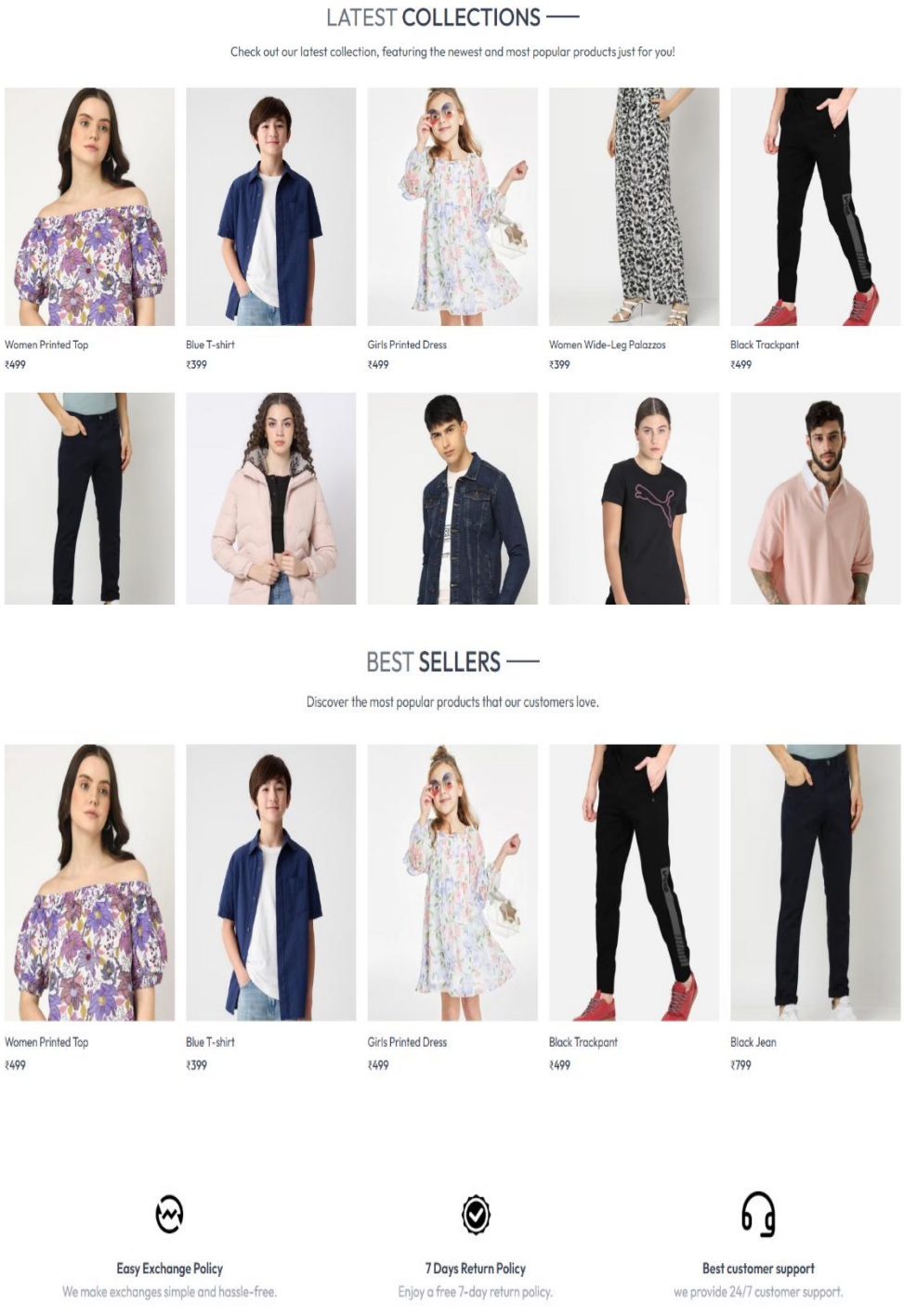
8.1 <u>Login / Signup Page: -</u> The Login and Sign-Up pages let users register or log in securely. The login page collects email and password, while the sign-up page adds fields like name. They ensure secure access to the platform using modern methods for password storage and user authentication, creating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



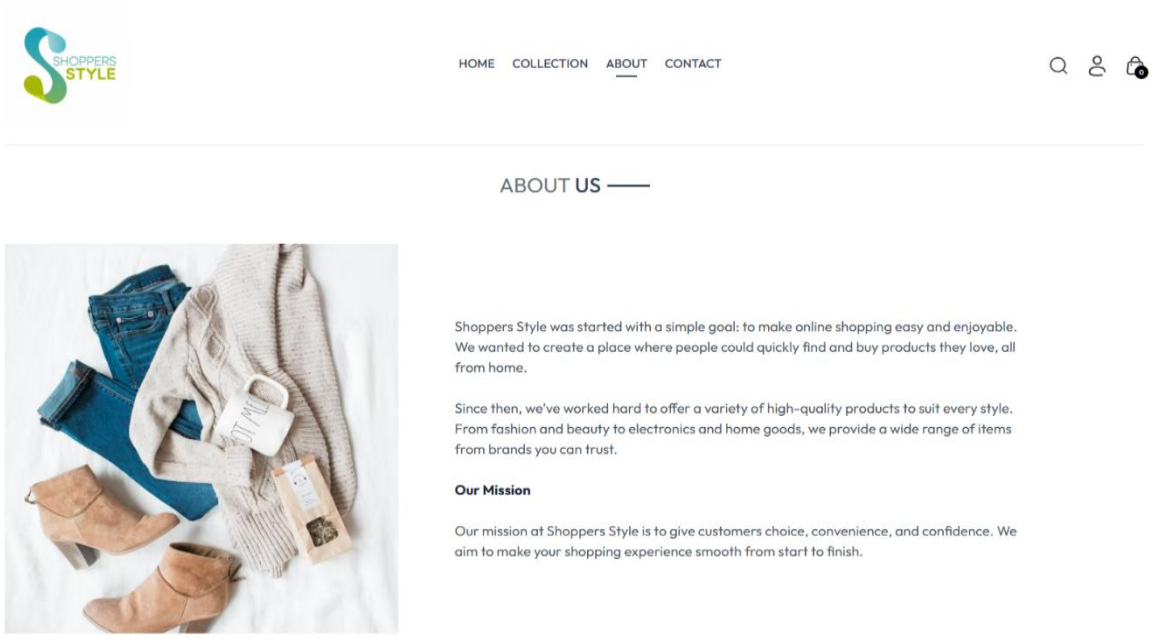
8.2 <u>Home Page: -</u> The Home page is the starting point for users. It showcases featured products, new collections, and links to key categories. The page is designed to work well on all devices, making navigation easy and engaging. It updates in real-time to show users the latest products and promotions.

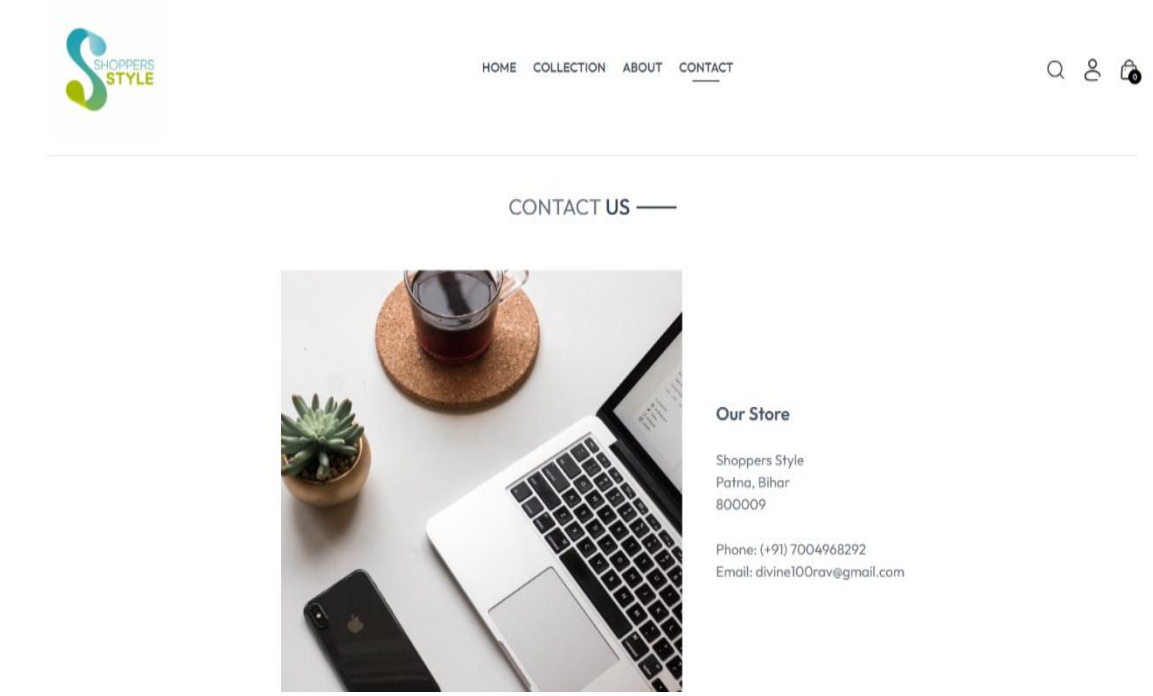


8.3 <u>Collection/ Best-Seller Page: -</u> This page highlights popular or specific collections of products. It lets users filter and sort items based on size, price, or style. The responsive design ensures it works on all devices, while live updates make sure the displayed products are always relevant,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

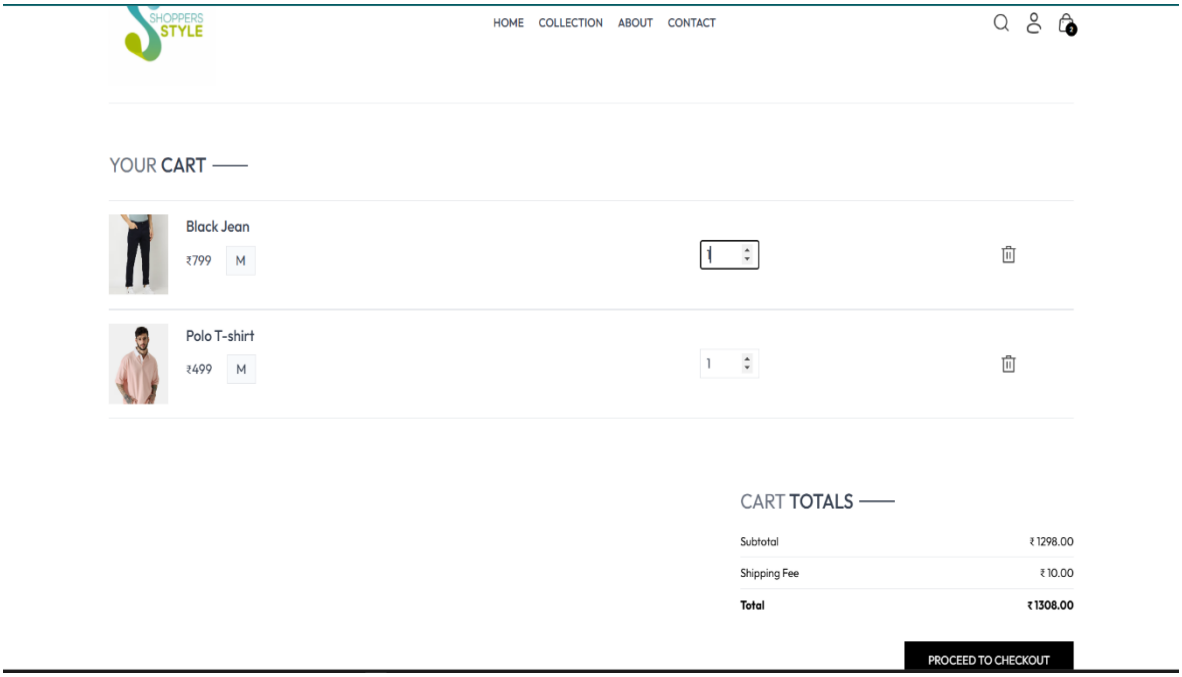


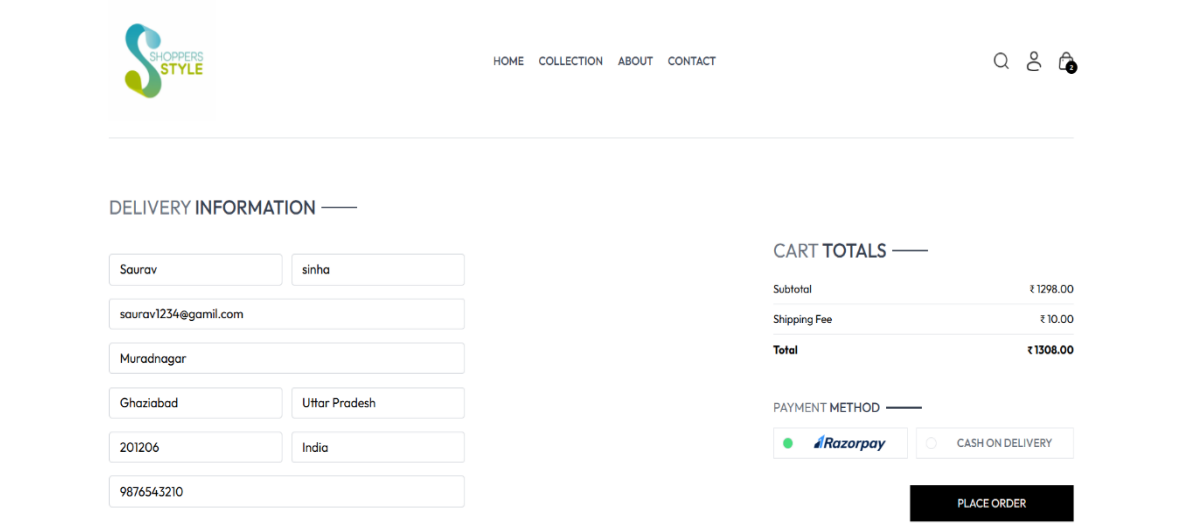
8.4 <u>About/ Contact Page: -</u> The About page explains the platform's purpose and values, while the Contact page provides ways to reach out for support, like email or forms. It's designed to be accessible on all devices, helping build trust by showing professionalism and offering help when users need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



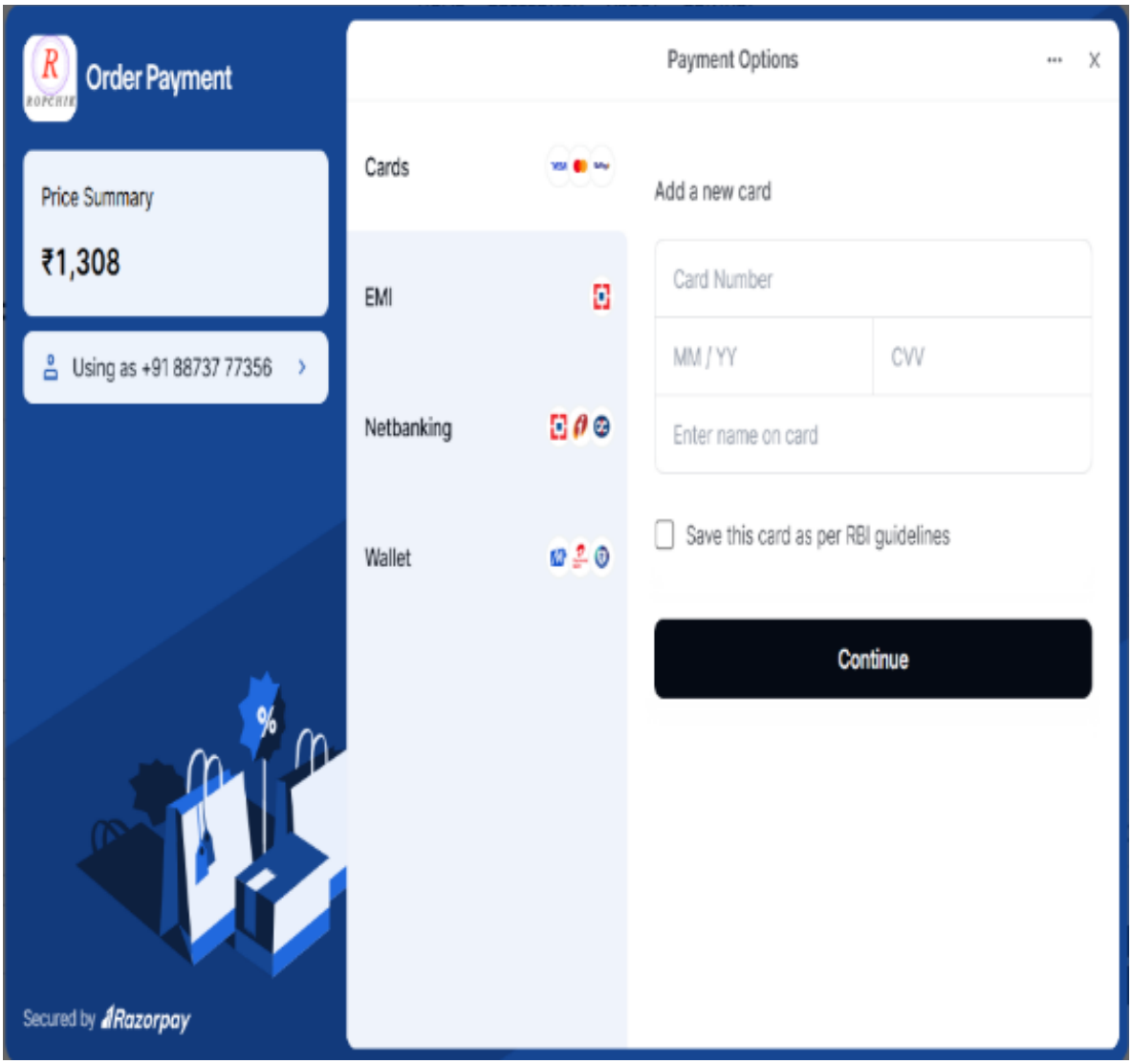


8.5 <u>Cart/ Checkout Page: -</u> The Cart page shows the items users want to buy, allowing them to change quantities or remove products. The Checkout page collects delivery details and payment preferences. It integrates securely with payment gateways like Razorpay, making the buying process simple and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

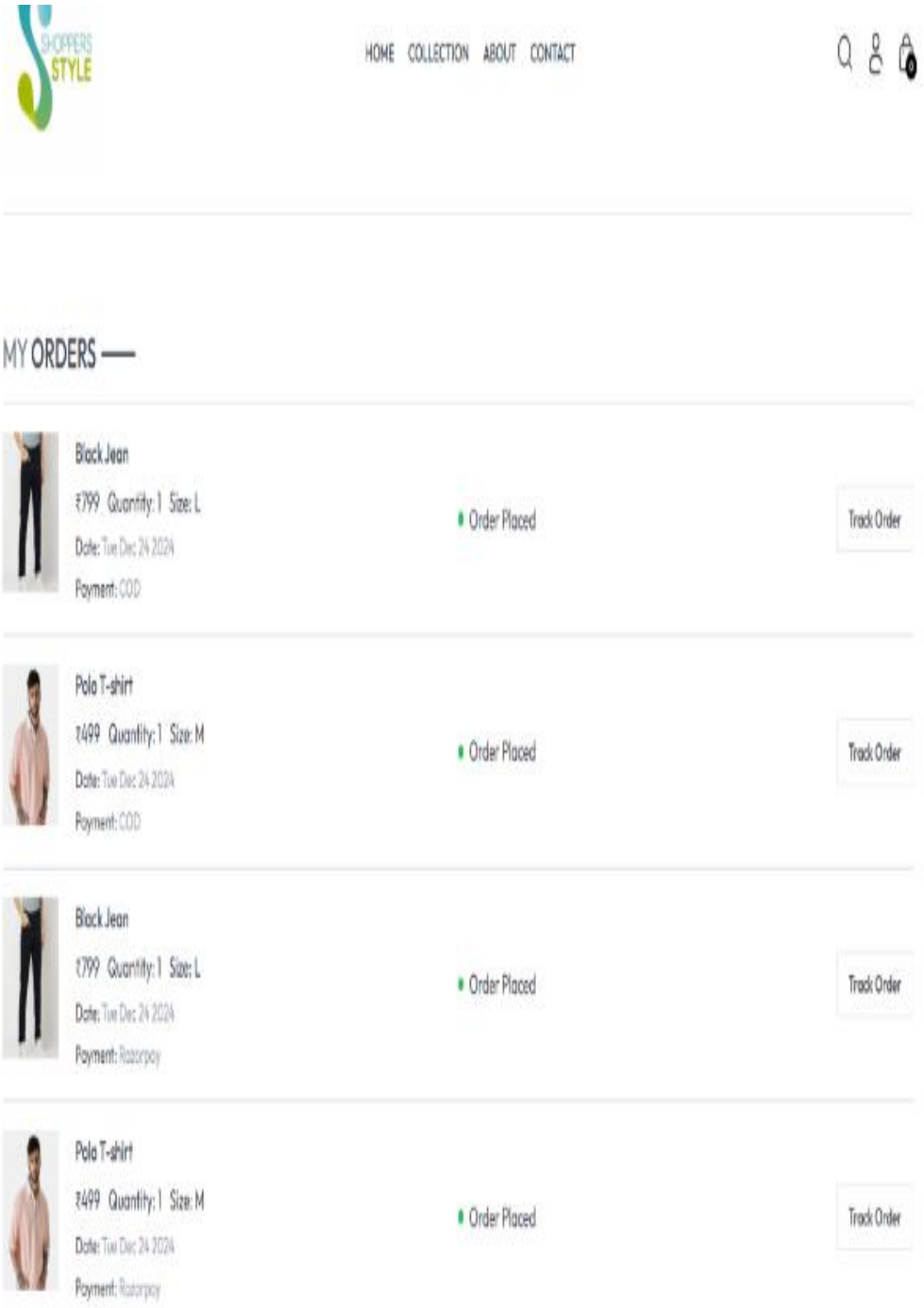




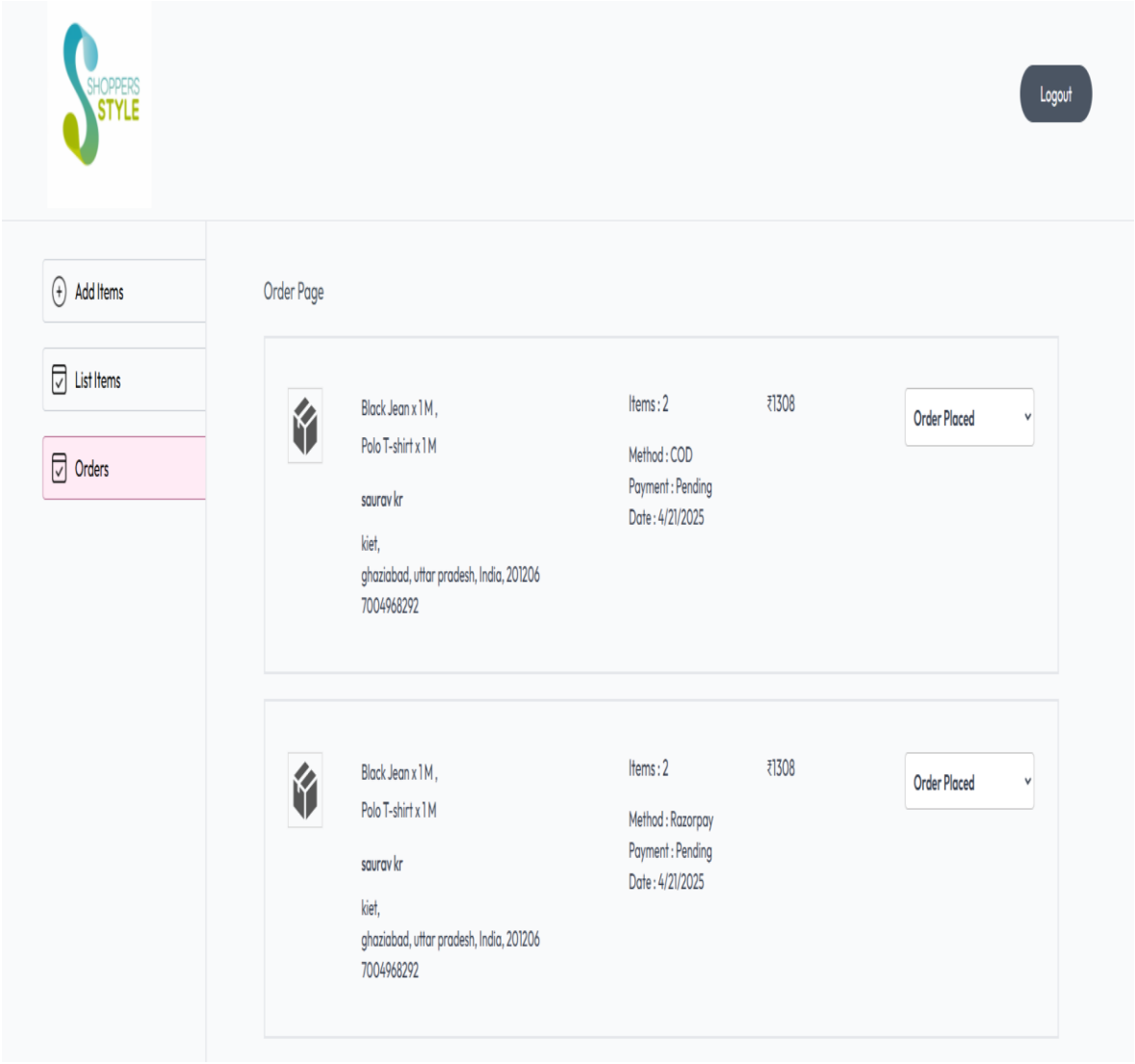
8.6 Payment Page: - The Payment page allows users to pay for their orders securely. It supports methods like Razorpay and Cash on Delivery. Transactions are processed safely, and users get updates on their payment status. The page is designed to make the process quick and easy while ensuring data security.



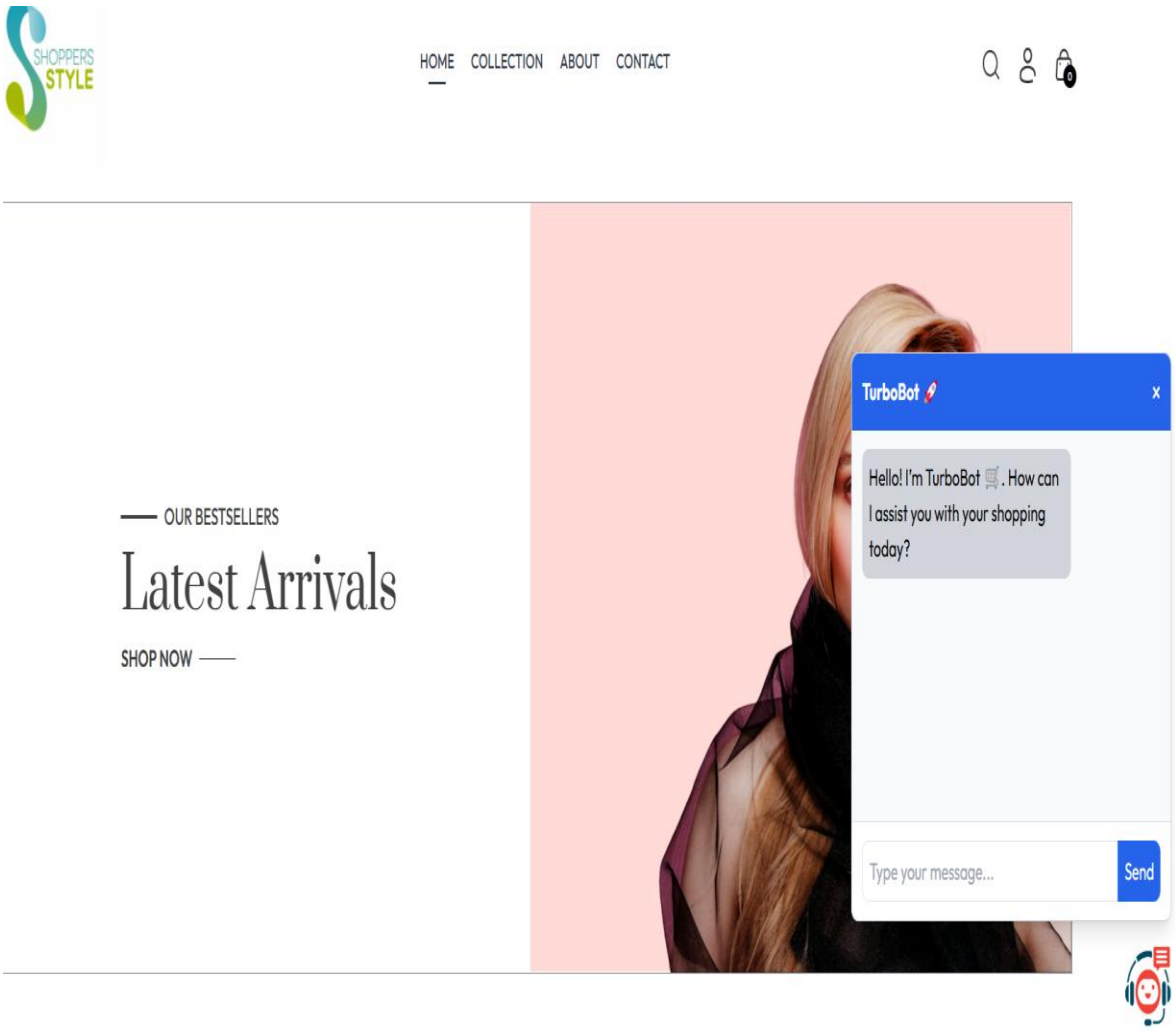
8.7 <u>User Order Page: -</u> The User Order page displays all the orders a user has placed. It shows details like order status, payment method, and delivery updates. Users can track active orders and check past purchases. It's easy to use and works well on all devices, helping users stay informed about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



8.8 <u>Admin Order Page: -</u> 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.



8.9 <u>ChatBot Page: -</u> The ChatBot page in the eCommerce website is designed to assist users by providing instant support using AI technology. It helps customers by answering their questions, guiding them through the shopping process, and suggesting products based on their preferences. The chatbot is available 24/7 and can also provide order details and support with common queries. It is integrated using AI tools like Dialogflow or OpenAI for smart responses and is built using React.js for the frontend, with Node.js and Express.js handling backend communication. MongoDB is used to store chat history and relevant data. This feature enhances the user experience by making the website more interactive and helpful.



Chapter 9 : 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 }
```

Chatbot.jsx:-

```
import React, { useState, useRef, useEffect } from 'react';


const ChatBot = ({ onClose }) => {
   const [messages, setMessages] = useState([
      { text: "Hello! I’m TurboBot 🛒. How can I assist you with your shopping today?", sender: "bot" }
   ]);
   const [input, setInput] = useState("");
   const messagesEndRef = useRef(null);


   useEffect(() => {
      messagesEndRef.current?.scrollIntoView({ behavior: "smooth" });
   }, [messages]);


   const handleSend = () => {
      if (input.trim() === "") return;
      const userMessage = input.trim();
      setMessages(prev => [...prev, { text: userMessage, sender: "user" }]);
      setInput("");


      // eCommerce Bot Response Logic
      setTimeout(() => {
         let botReply = "Sorry, I didn't understand. Can you please rephrase?";
         const lowerCaseMsg = userMessage.toLowerCase();


         if (lowerCaseMsg.includes("order status") || lowerCaseMsg.includes("track my order")) {
            botReply = "You can track your order in the 'My Orders' section or share your Order ID.";
```

```
        } else if (lowerCaseMsg.includes("return") || lowerCaseMsg.includes("refund")) {
            botReply = "Our return policy allows returns within 7 days of delivery. Do you want help initiating a return?";
        } else if (lowerCaseMsg.includes("shipping") || lowerCaseMsg.includes("delivery")) {
            botReply = "We offer free shipping on orders above ₹999. Delivery takes 3-5 business days.";
        } else if (lowerCaseMsg.includes("product") || lowerCaseMsg.includes("available")) {
            botReply = "Please share the product name or category. I'll check availability for you.";
        } else if (lowerCaseMsg.includes("payment") || lowerCaseMsg.includes("cod")) {
            botReply = "We accept UPI, Credit/Debit Cards, and Cash on Delivery (COD).";
        } else if (lowerCaseMsg.includes("offer") || lowerCaseMsg.includes("discount")) {
            botReply = "Check out our latest offers in the 'Deals' section! Flat 20% off on new arrivals.";
        } else if (lowerCaseMsg.includes("thank you") || lowerCaseMsg.includes("thanks")) {
            botReply = "You're welcome! 😊 Let me know if you need any more help.";
        }

        setMessages(prev => [...prev, { text: botReply, sender: "bot" }]);
    }, 1000);
  };

  const handleKeyPress = (e) => {
    if (e.key === 'Enter') {
      handleSend();
    }
  };

  return (
```

```
<div className='fixed bottom-24 right-6 w-full max-w-sm h-96 bg-white border border-gray-300 rounded-lg shadow-lg flex flex-col z-50 '>
        {/* Chat Header */}
        <div className='bg-blue-600 text-white p-3 rounded-t-lg flex justify-between items-center'>
          <h2 className='font-bold'>TurboBot 🚀 </h2>
          <button onClick={onClose} className='text-white text-xl'>×</button>
        </div>

        {/* Messages */}
        <div className='flex-1 overflow-y-auto p-3 space-y-2 bg-gray-50'>
          {messages.map((msg, index) => (
            <div
              key={index}
              className={`flex ${msg.sender === "user" ? "justify-end" : "justify-start"}`}
            >
              <div
                className={`p-2 rounded-lg max-w-[70%] ${msg.sender === "user"
                    ? "bg-blue-500 text-white text-right"
                    : "bg-gray-300 text-black text-left"
                  }`}
              >
                {msg.text}
              </div>
            </div>
          ))}
          <div ref={messagesEndRef} />
        </div>

        {/* Input */}
```

```
        <div className='p-3 border-t flex'>
          <input
            type="text"
            value={input}
            onChange={(e) => setInput(e.target.value)}
            onKeyDown={handleKeyPress}
            className='flex-1 border rounded-l-lg p-2 outline-none'
            placeholder='Type your message...'
          />
          <button
            onClick={handleSend}
            className='bg-blue-600 text-white p-2 rounded-r-lg hover:bg-blue-700'
          >
            Send
          </button>
        </div>
      </div>
    );
};

export default ChatBot;
```

Chapter 10 : Conclusion

Our project, a clothing e-commerce website developed using the MERN stack (MongoDB, Express.js, React.js, Node.js), incorporates AI chatbot features to enhance the overall shopping experience. Focused on meeting the evolving needs of modern online shoppers, the platform is designed to be intuitive, user-friendly, efficient, and reliable.

The inspiration for this project came from the growing demand for online clothing stores and the desire to create a solution that is not only functional but also offers a personalized, dynamic experience. To achieve this, we integrated AI technologies into various aspects of the platform. Features such as smart product recommendations, AI-powered trend predictions, dynamic pricing, and chatbot assistants provide users with a more interactive and customized shopping experience.

The main functionalities include browsing, filtering, and sorting products, choosing sizes, managing carts, and placing orders. AI tools provide intelligent product suggestions based on user behavior, while the dynamic pricing model adjusts prices based on trends. Additionally, the platform integrates secure user logins and payment gateways like Razorpay to ensure safe transactions.

During the development process, we followed a clear plan, from defining the project goals and designing the system to thoroughly testing every feature. This ensured that the website works smoothly and meets the expectations of its users.

In the future, we plan to expand the AI features by further personalizing recommendations, improving trend analysis, and introducing advanced analytics to optimize product offerings. Our goal is to continue innovating and enhancing the functionality of the platform, making it a preferred destination for online shoppers. This project exemplifies our commitment to developing practical, visionary solutions that integrate AI to improve the user experience in e-commerce.

Chapter 11 : 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/
- https://platform.openai.com/docs/overview
- https://razorpay.com/docs/
- https://www.npmjs.com/
- https://reactrouter.com/
- https://redux-toolkit.js.org/
- https://www.freecodecamp.org/news/learn-the-mern-stack/
- https://tailwindcss.com/
- https://getbootstrap.com/