ShopEZ: E-commerce Web Application

Project Documentation

Submitted by:

Harshit Ambwani (22BCE10076)

Harleen Kaur Pama (22BCE10214)

Jiya (22BCE10490)

Sushant Kumar (22BCE11381)

Mentored by:

SmartBridge

Department of Computer Science and Engineering

VIT Bhopal University

Academic Year: 2024–2025

Acknowledgement

I would like to take this opportunity to acknowledge and commend the exceptional hard work, commitment, and teamwork displayed by everyone involved in the development of the ***ShopEZ E-commerce Application****. This project has been a meaningful journey, laying the groundwork for a modern, scalable, and user-friendly online shopping experience. I am incredibly proud of the progress that has been made and the collaborative spirit that has driven it forward.*

The success of this application is the result of effective collaboration, creative problem-solving, and strong technical expertise. Every challenge was met with resilience, and key milestones were achieved through consistent effort and determination. The ability of the team to communicate openly and work cohesively was instrumental in transforming ideas into a functioning digital product.

I would also like to extend special thanks to each individual who contributed to this project. Your dedication and expertise were vital to bringing ShopEZ to life. Additionally, I express my sincere gratitude to my mentors and guides for their constant support, insights, and encouragement throughout this process. Their involvement helped shape the vision and ensure that the project remained aligned with its broader goals.

As we move forward, I am excited about the potential of ShopEZ to grow further and deliver real value to both users and sellers. This experience has been deeply enriching, and I look forward to the continued evolution of this platform.

ABSTRACT

ShopEZ is a full-stack e-commerce web application developed using the MERN stack (MongoDB, Express.js, React.js, and Node.js) to facilitate seamless online shopping experiences for users while empowering sellers to manage and showcase their products efficiently. The platform supports three primary user roles: Admin, Seller, and Buyer, each with tailored functionalities.

The frontend, built with React and styled using Bootstrap and React-Bootstrap, offers a responsive and user-friendly interface. Redux is used for efficient state management, especially in handling user sessions and dynamic UI components. The backend, developed with Node.js and Express.js, handles routing, authentication, and secure API communication, while MongoDB serves as the primary database.

Key features include user registration and login with token-based authentication (JWT), product listing by sellers, search and filter options for buyers, and order management for sellers and admins. RESTful APIs ensure smooth client-server interaction, and the system has been tested using Postman.

Deployed on Render with separate live servers for frontend and backend, the application, despite minor limitations like incomplete cart functionality and profile rendering issues, showcases real-world scalability and commercial potential. This project highlights the power of modern web technologies and reflects strong team collaboration in building a real-time, dynamic e-commerce solution.

CONTENTS

LIST OF FIGURES, CODE SNIPPET AND TABLES

LIST OF ABBREVIATIONS

DOM	*Document Object Model*
NPM	*Node Package Manager*
API	*Application Program Interface*
HTML	*Hypertext Markup Language*
HTTP	*Hypertext Transfer Protocol*
URL	*Uniform Resource Locator*
JSON	*Javascript Object Notation*
JS	*Javascript*

1. INTRODUCTION

The ShopEZ E-commerce Application is developed as part of a MERN stack-based project, aimed at creating a modern, secure, and highly user-friendly platform for both online shoppers and small-scale sellers. This project was built under academic supervision and reflects the practical implementation of full-stack development concepts, focusing on real-world usability and technical efficiency.

ShopEZ envisions a centralized and seamless shopping experience, with special consideration for sellers aiming to manage their products and grow their businesses through a digital medium. Inspired by platforms that serve both buyers and independent sellers, this project brings forward a clean and optimized system that handles user authentication, product discovery, cart functionality, payment processes, and seller order management with ease.

The MERN (MongoDB, Express.js, React.js, Node.js) technology stack was selected for its powerful ecosystem, fast development capability, and vast community support. React ensures a dynamic and responsive user interface, while Node and Express handle robust server-side operations. MongoDB serves as the NoSQL database solution for scalable data storage.

This documentation is organized into multiple parts, each addressing key elements of the development lifecycle—starting from requirements, architectural design, and implementation, to results, testing, and analysis. The technical architecture section provides a high-level overview of the system components:

- *Frontend: Developed using React, it includes modules for user interaction like product browsing, cart handling, checkout, profile management, and the seller/admin dashboard.*

- *Backend: Implemented using Node.js and Express.js, this layer provides RESTful API endpoints for user management, product CRUD operations, order processing, and admin authentication.*

- *Database: MongoDB is used to store and manage collections related to users, products, carts, and orders.*

This system is not only designed for performance and scalability but also provides a foundation for future enhancements like machine learning-based product recommendations, real-time chat support, and integrations with third-party payment and logistics providers.

2. <u>REQUIREMENTS</u>

2.1 Client-Specified Requirements

Before initiating development, the client defined several specific requirements tailored for the student-centric nature of ShopEZ:

1. Student Seller Eligibility: Only students above the age of 18 can register as sellers. Sellers must verify their student status.

2. User-Friendly UI: A responsive and modern interface is required to attract and retain student users.

3. High-Performance Backend: The system should include a fast and reliable server.

4. Scalable Database Architecture: The platform must accommodate increasing numbers of users and transactions.

2.2 Requirement Analysis

2.2.1 Registration System

Given the dual nature of users (students as sellers and general users as buyers), the registration system required additional complexity.

Initial ideas like uploading student ID cards with QR codes were considered. However, limitations such as image readability and institutions' reluctance to share data made this unfeasible. Instead, the project implemented email verification through academic domains as a practical and secure method of validating studentship.

2.2.2 Easy and Modern Responsive UI

The front-end was developed using React.js, a popular JavaScript library supported by a vast global community. It allows the creation of highly interactive and component-based interfaces. The UI is complemented with Bootstrap, a CSS framework offering customizable layouts and responsive design out-of-the-box, ensuring smooth *operation across desktop, tablet, and mobile devices.*

2.2.3 Fast Responding Server Side

The backend was built using Node.js with Express.js. Express offers lightweight, robust features for building fast web applications, while Node.js ensures scalable and real-time interactions. This stack was chosen over alternatives like Laravel and Django due to its performance advantages and JavaScript continuity across the entire stack.

2.2.4 Scalable Database

The database uses MongoDB, a NoSQL solution known for flexibility and horizontal scalability. Unlike relational databases such as MySQL, MongoDB supports rapid data insertion (e.g., InsertMany API), and is especially suitable for applications where user-generated content (e.g., images, files) needs to be stored efficiently.

2.3 Pre-requisites

To build and run the ShopEZ project, the following prerequisites are required:

- *Node.js and npm: For running the server and managing packages.*
- *MongoDB: For storing user, product, and transaction data.*
- *Express.js: Node.js framework for backend routing and API handling.*
- *React.js: Frontend library for building the user interface.*
- *HTML, CSS, JavaScript: Foundational web development knowledge.*
- *Database Connectivity: Use of Mongoose ODM for seamless MongoDB interaction.*
- *Version Control: Git for version management and GitHub for hosting the repository.*
- *Development Environment: Recommended editors include Visual Studio Code, Sublime Text, and WebStorm.*

- *Run ShopEZ Locally:*
    - *- Clone the repository: git clone [REPO_LINK]*
    - *- Navigate to the directory: cd ShopEZ-e-commerce-App-MERN*
    - *- Install dependencies: npm install*
    - *- Start the server: npm run dev or npm start*
    - *- Visit: http://localhost:3000*

2.4 Methodology

After evaluating all client requirements and technological alternatives, the team selected the MERN stack for development. This stack supports fast development cycles, JavaScript uniformity across front-end and back-end, high scalability, and extensive community support.

Technologies Used:

- *React.js: Front-end user interface*
- *Node.js + Express.js: Backend server and APIs*
- *MongoDB + Mongoose: Database and ODM*
- *Bootstrap: Styling and responsive design*
- *Git & GitHub: Version control and collaboration*
- *Visual Studio Code: Development environment*

3. <u>DESIGN</u>

This chapter provides a detailed overview of the entire development process of the ShopEZ e-commerce platform. It covers the planning and design phases, selection of tools and IDEs, version control strategy, and the implementation logic covering both frontend and backend aspects.

3.1 Project Planning

Before beginning the development of ShopEZ, it was essential to plan the application's structure and functionality. The application features three types of users:

- *Admin – Oversees the entire platform.*
- *Seller – Uploads and manages their products.*
- *Buyer – Browses, purchases, and manages orders.*

Each user role has unique functionalities and UI views customized to their specific access level.

3.1.1 Project Design Planning

The design of ShopEZ was carefully structured to provide a smooth and responsive user experience for all user types:

- *Admin Dashboard:*
  - *Manage products (add, edit, delete).*
  - *Monitor orders and track sales.*
  - *View users (sellers and buyers), remove users if needed.*
  - *Check messages via the integrated chat feature.*
  - *Access analytics dashboard to review monthly sales performance.*
  - *Log out functionality.*

- *Seller Dashboard:*
  - *- Manage their profile.*
  - *- Upload new products with images, prices, and descriptions.*
  - *- View sales history and order information related to their products.*
- *Buyer Interface:*
  - *- Personalized dashboard upon login.*
  - *- Browse all products, add items to cart, place orders.*
  - *- Track order history and manage profile.*
  - *- View specific product pages and details.*

Figures representing admin, seller, and buyer UI designs can be found in the following sections.

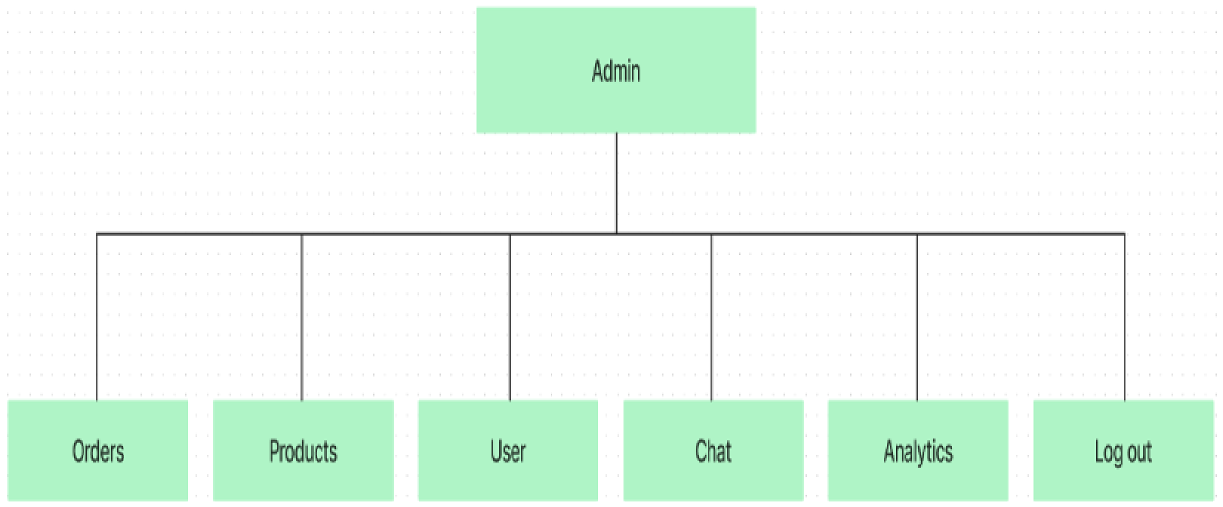


Figure 1: Admin page design

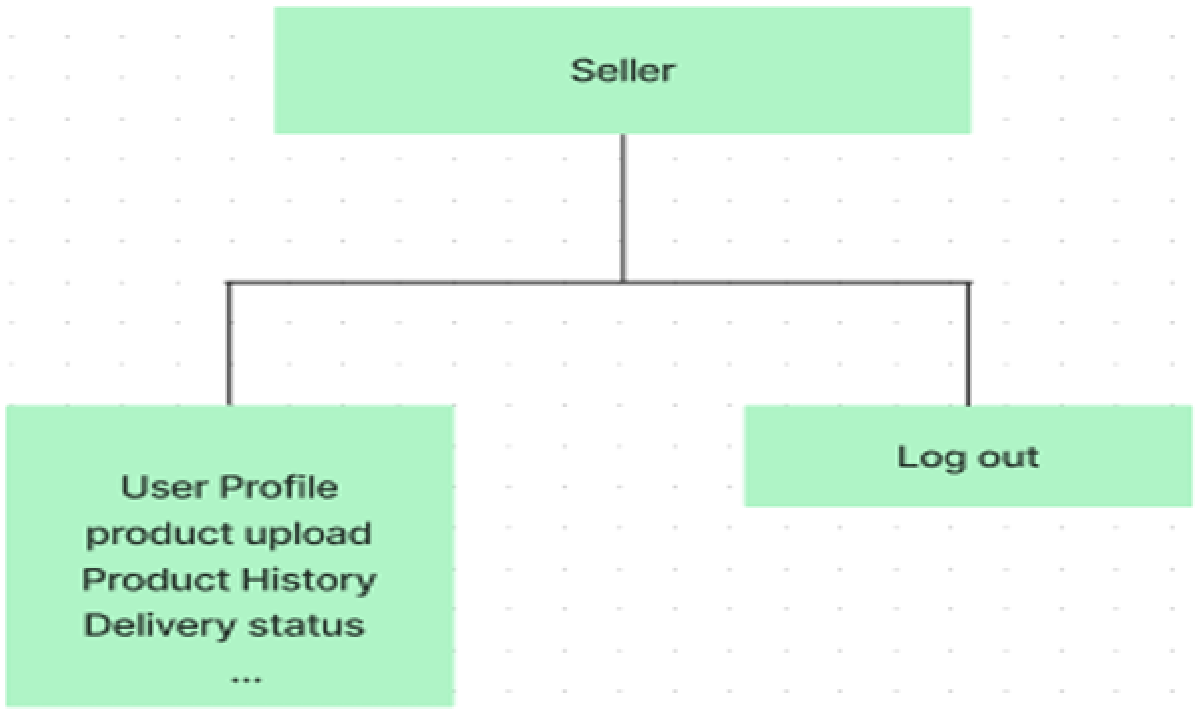


Figure 2: Seller page design

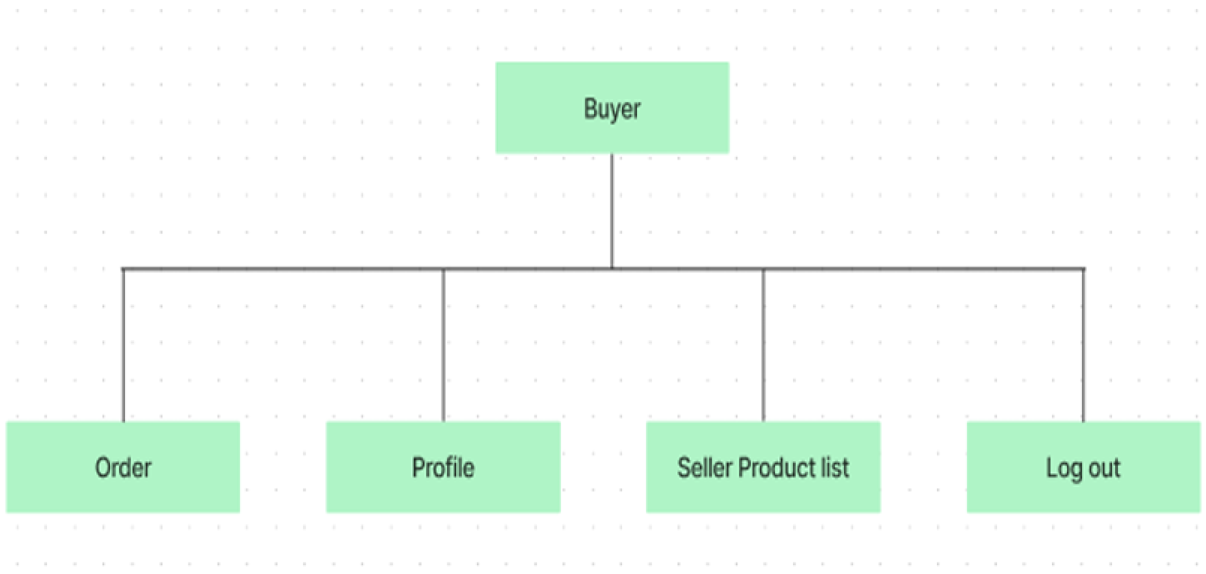


Figure 3: Buyer page design

3.1.2 Developer Environment

The development environment significantly contributes to efficient coding and project management. For ShopEZ, Visual Studio Code was selected due to its lightweight nature, built-in terminal, rich extension support, and integration capabilities.

Extensions Used:

1. *ES7+ React/Redux/React-Native Snippets – Speed up React development.*
2. *Babel JavaScript – Syntax highlighting for modern JS.*
3. *Live Preview – For viewing HTML/JS files in real-time.*
4. *vscode-icons – Better visualization of the file system.*

3.1.3 Git Version Control

To manage the source code efficiently and allow collaboration, Git was used as the version control system. Commands like git init, git status, git add, git commit, and git push were regularly used to manage commits and updates. The codebase was hosted on GitHub, making collaboration, backup, and version history management seamless.

3.1.4 List of Packages and Frameworks

The following tools and libraries were used in the development of ShopEZ:

Backend (Node.js + Express):

- *Node.js v16.17.1 – JavaScript runtime.*
- *Express.js – Backend web framework.*
- *Mongoose v6.2.8 – MongoDB object modeling.*
- *JWT (jsonwebtoken v8.5.1) – Authentication and session management.*

- *bcryptjs v2.4.3 – Password hashing for secure authentication.*
- *multer v1.4.5 – Handle image and file uploads.*
- *nodemailer v6.9.1 – Email OTP verification.*
- *helmet v5.1.1 – Securing HTTP headers.*
- *nodemon v2.0.15 – Automatic server restarts during development.*

Frontend (React.js):

- *React v17.0.2 – UI library for building components.*
- *React-DOM v17.0.2 – For DOM rendering.*
- *React-Router-DOM v6.2.2 – SPA navigation.*
- *Axios v0.26.1 – Promise-based HTTP requests.*
- *Bootstrap v5.1.3 – Responsive design.*
- *React-Bootstrap v2.2.1 – React components for Bootstrap.*

Other dependencies and tools can be found in the package.json file in the project root directory.

3.2 Database Design

The structure of the database for the ShopEZ application is based on a NoSQL schema using MongoDB. It is designed to store and manage different collections such as Users, Products, Orders, and Cart. Each collection is optimized for scalability and ease of retrieval.

The ER diagram and schema breakdown are shown below for a better understanding of the relationships between entities and their attributes.

3.2.1 Schema Structure

The ShopEZ database consists of the following collections:

1. *Users Collection:*
    - *Stores information about registered users.*
    - *Key Fields:*
        - *_id: Unique identifier for the user.*
        - *username: User’s login name.*
        - *email: User’s email address.*
        - *password: Hashed password for authentication.*
        - *cart: List of products added to the user's cart.*
2. *Admin Collection:*
    - *Contains information managed by admins, including banners and product categories.*
    - *Key Fields:*
        - *_id: Unique identifier for the admin.*
        - *bannerImage: URL of the promotional banner.*
        - *categories: List of product categories.*
3. *Products Collection:*
    - *Stores product listings available on the platform.*
    - *Key Fields:*
        - *_id: Unique product identifier.*
        - *name: Name of the product.*
        - *description: Detailed product description.*
        - *price: Product price.*
        - *category: The category the product belongs to.*
4. *Cart Collection:*
    - *Contains products added to a user’s cart.*
    - *Key Fields:*
        - *_id: Unique identifier for the cart.*
        - *userId: Reference to the user who owns the cart.*
        - *items: List of products (with quantity) in the cart.*

5. *Orders Collection:*

- Stores order details for completed transactions.

- Key Fields:

_id: Unique order identifier.

userId: Reference to the user who placed the order.

products: List of products in the order.

totalAmount: Total price for the order.

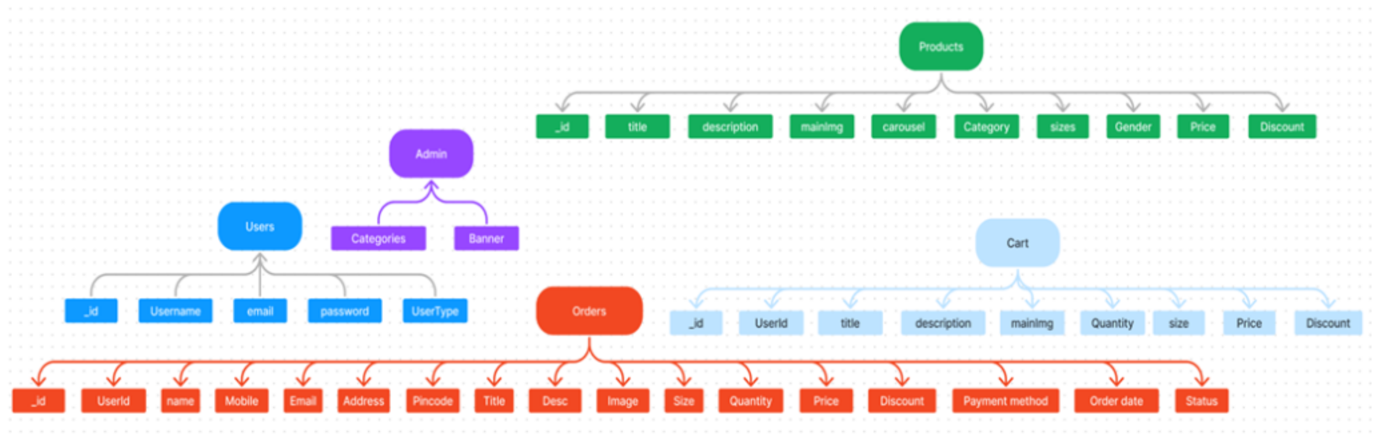


Figure 4: Schema Structure

3.2.2 ER Diagram

The ER Diagram below illustrates the relationships between different entities in the ShopEZ platform:

- *User Entity: Represents customers who use the platform.*

  - A user can have multiple orders (one-to-many relationship).

  - A user can add multiple products to their cart (many-to-many relationship).

- *Admin Entity: Represents the platform's admin who manages products and categories.*

  - An admin can manage multiple products and categories (one-to-many relationship).

- *Product Entity: Represents products listed on the platform.*

  - A product can appear in multiple orders (many-to-many

relationship).

- A product can be added to multiple carts (many-to-many relationship).

- *Cart Entity: Represents the shopping cart, where users add products before purchasing.*

  - A cart is owned by one user (one-to-one relationship).

  - A cart can contain multiple products (many-to-many relationship).

- *Order Entity: Represents an order placed by a user.*

  - An order contains multiple products (many-to-many relationship).

  - An order is placed by one user (many-to-one relationship).

Here is the ER Diagram showing these relationships:

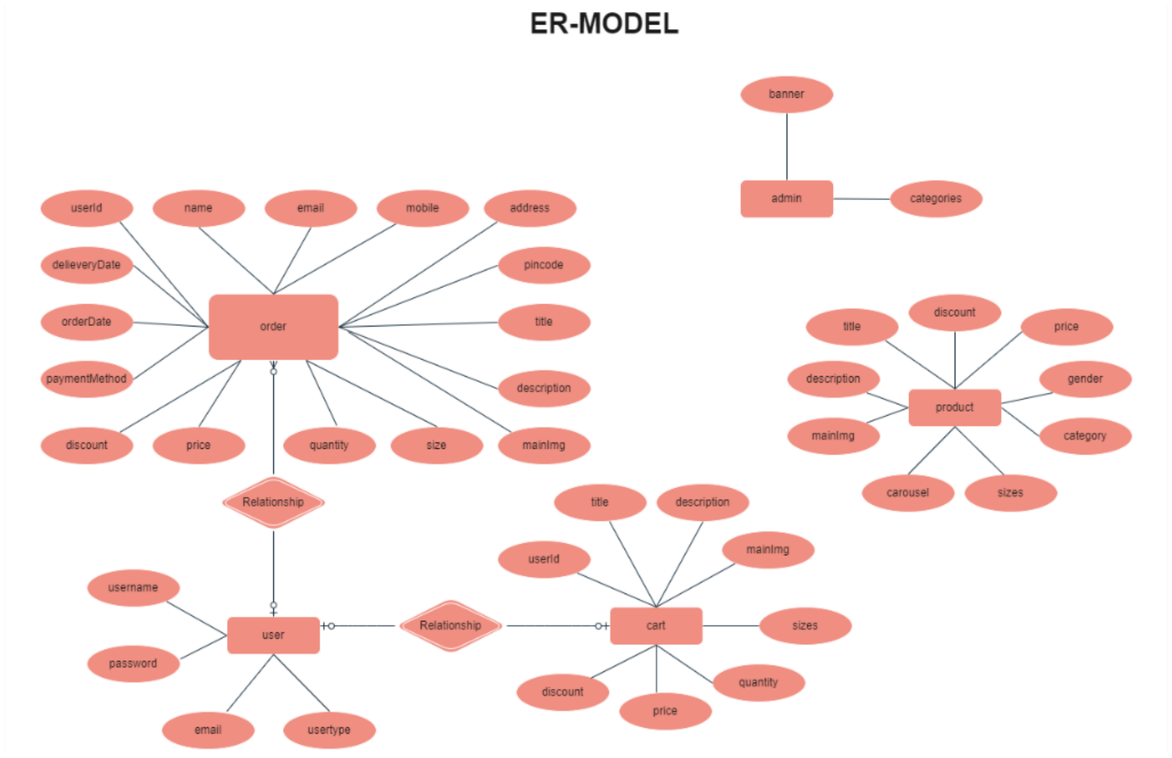


Figure 5: ER Diagram representing entities and their relationships.

3.3 Implementation

Here's a phase-wise breakdown of the implementation details of **ShopEZ**, the MERN stack-based e-commerce platform:

Phase 1: Project Setup

- **Frontend Setup:**
    - Initialize React.js project using `create-react-app`.
    - Setup folder structure:
        - `src/` for components, pages, and route logic.
        - `public/` for static assets like logos and uploaded images.
    - Install necessary dependencies including `redux`, `react-router`, etc.
- **Backend Setup:**
    - Initialize Node.js project using `npm init`.
    - Install and configure `Express.js`, `Mongoose`, and other dependencies.
    - Setup folder structure for modular backend:
        - `config/`, `controllers/`, `models/`, `routes/`, `middleware/`, `utils/`, and `seeder/`.

Phase 2: Frontend Development

- **UI Implementation:**
    - Design and build UI components for three roles:
        - **Admin** – Dashboard, user/product management.
        - **Seller** – Product listing, order tracking.
        - **Buyer** – Browsing, cart, checkout.
- **Routing and Navigation:**
    - Implement navigation using `React Router`.
- **State Management:**
    - Integrate `Redux Toolkit` for global state across components (auth, cart, etc.).

Phase 3: Backend Development

- **Database and Configuration:**
    - Use `config/` to set up MongoDB connection and manage environment variables.
- **API Development:**
    - Define data models using `Mongoose` in `models/`.
    - Create route handlers and controllers for:
        - User registration/login
        - Product CRUD operations

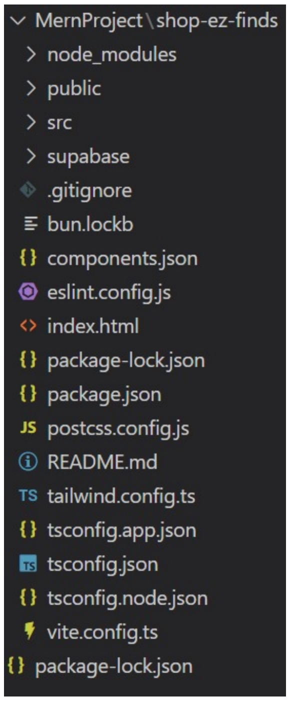
        - Order processing
- **Authentication & Security:**
    - Use `middleware/` to implement:
        - JWT-based authentication.
        - Error handling and authorization checks.
- **Utility Functions:**
    - Implement helper functions in `utils/` (e.g., token generation, file uploads).

Phase 4: Testing & Seeding

- **Database Seeding:**
    - Use `seeder/` to populate the database with sample data for development and testing.
- **Component and API Testing:**
    - Test frontend components using sample roles.
    - Test API endpoints using tools like Postman.

Phase 5: Integration & Final Touches

- **Frontend-Backend Integration:**
    - Connect frontend to backend APIs using `Axios` or `fetch`.
    - Handle data fetching, error states, and loading indicators.
- **Deployment Preparation:**
    - Finalize environment variables.
    - Optimize performance and code cleanup.
    - Prepare for deployment on platforms like Heroku, Vercel, or AWS.



```
MernProject\shop-ez-finds
  node_modules
  public
  src
  supabase
  .gitignore
  bun.lockb
  components.json
  eslint.config.js
  index.html
  package-lock.json
  package.json
  postcss.config.js
  README.md
  tailwind.config.ts
  tsconfig.app.json
  tsconfig.json
  tsconfig.node.json
  vite.config.ts
package-lock.json
```

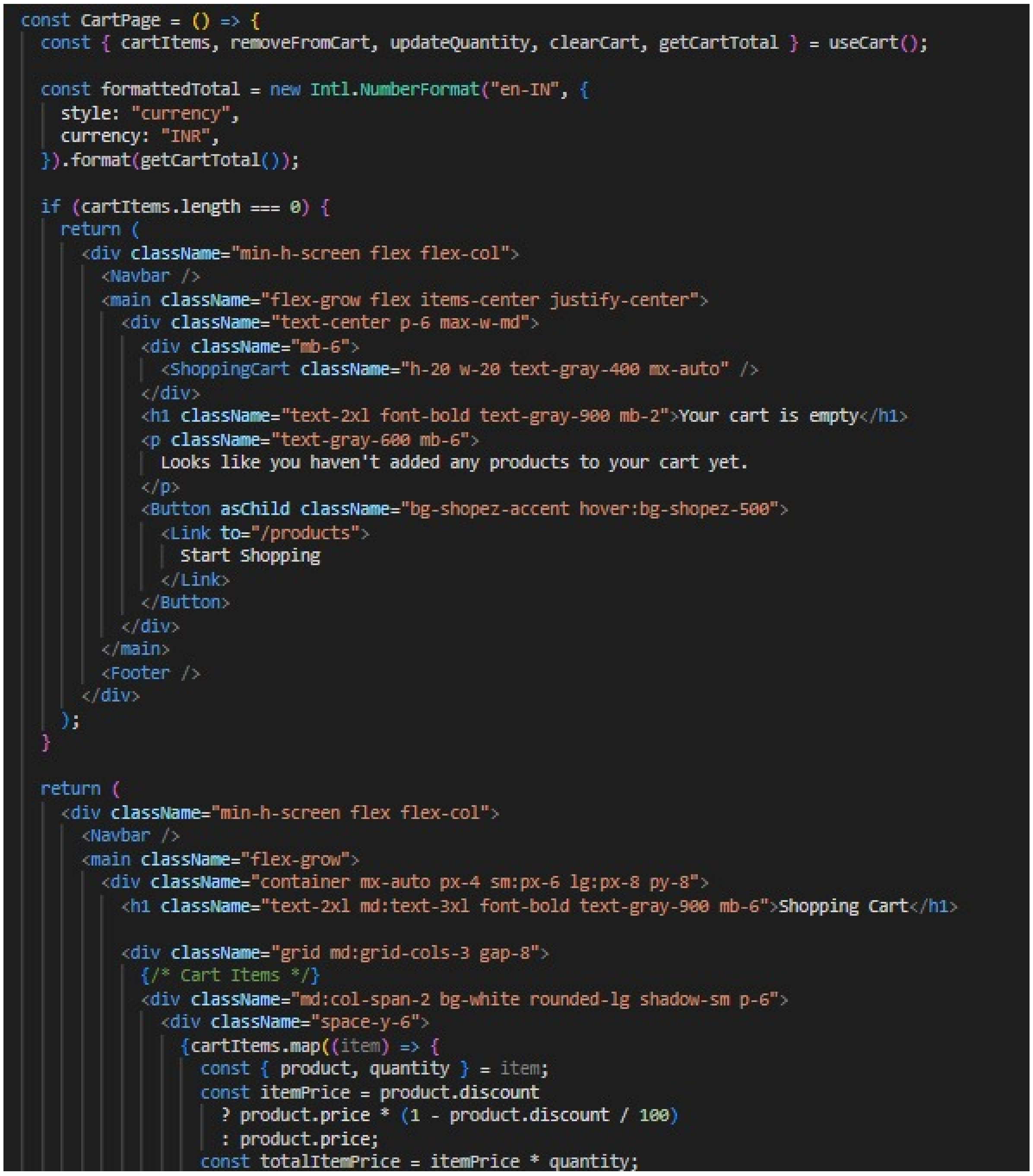
Figure 6: ShopEZ Project file structure

3.3.1 Frontend Implementation

In ShopEZ, the frontend is designed to serve three types of users—Admin, Seller, and Buyer—each with access to dedicated components and routes to streamline their interactions with the platform. All frontend code is structured under the src/ folder. Each user role has a dedicated layout and navigation structure, simplifying user experience and code maintenance.

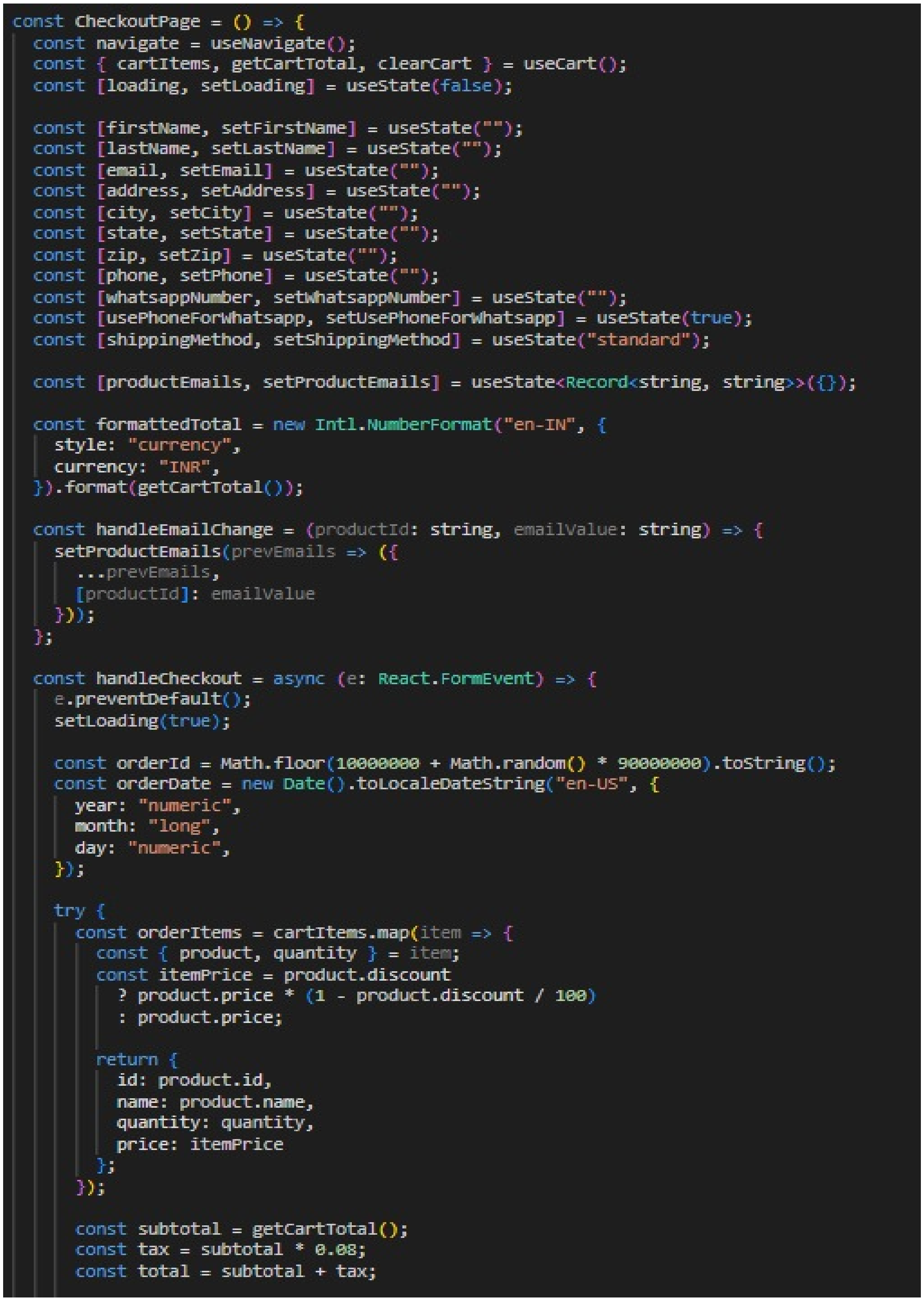
To manage application-wide state, Redux Toolkit is used. It enables efficient data flow between components such as user login status, shopping cart management, and order tracking.
The UI is styled using React-Bootstrap, which provides responsive components like Container, Row, Col, Navbar, and Carousel. These were used to build common UI elements like the homepage banner, product

cards, navigation bar, and footers.



```
const CartPage = () => {
  const { cartItems, removeFromCart, updateQuantity, clearCart, getCartTotal } = useCart();

  const formattedTotal = new Intl.NumberFormat("en-IN", {
    style: "currency",
    currency: "INR",
  }).format(getCartTotal());

  if (cartItems.length === 0) {
    return (
      <div className="min-h-screen flex flex-col">
        <Navbar />
        <main className="flex-grow flex items-center justify-center">
          <div className="text-center p-6 max-w-md">
            <div className="mb-6">
              <ShoppingCart className="h-20 w-20 text-gray-400 mx-auto" />
            </div>
            <h1 className="text-2xl font-bold text-gray-900 mb-2">Your cart is empty</h1>
            <p className="text-gray-600 mb-6">
              Looks like you haven't added any products to your cart yet.
            </p>
            <Button asChild className="bg-shopez-accent hover:bg-shopez-500">
              <Link to="/products">
                Start Shopping
              </Link>
            </Button>
          </div>
        </main>
        <Footer />
      </div>
    );
  }

  return (
    <div className="min-h-screen flex flex-col">
      <Navbar />
      <main className="flex-grow">
        <div className="container mx-auto px-4 sm:px-6 lg:px-8 py-8">
          <h1 className="text-2xl md:text-3xl font-bold text-gray-900 mb-6">Shopping Cart</h1>

          <div className="grid md:grid-cols-3 gap-8">
            {/* Cart Items */}
            <div className="md:col-span-2 bg-white rounded-lg shadow-sm p-6">
              <div className="space-y-6">
                {cartItems.map((item) => {
                  const { product, quantity } = item;
                  const itemPrice = product.discount
                    ? product.price * (1 - product.discount / 100)
                    : product.price;
                  const totalItemPrice = itemPrice * quantity;
```

Code Snippet 1: Cart page design



```
const CheckoutPage = () => {
  const navigate = useNavigate();
  const { cartItems, getCartTotal, clearCart } = useCart();
  const [loading, setLoading] = useState(false);

  const [firstName, setFirstName] = useState("");
  const [lastName, setLastName] = useState("");
  const [email, setEmail] = useState("");
  const [address, setAddress] = useState("");
  const [city, setCity] = useState("");
  const [state, setState] = useState("");
  const [zip, setZip] = useState("");
  const [phone, setPhone] = useState("");
  const [whatsappNumber, setWhatsappNumber] = useState("");
  const [usePhoneForWhatsapp, setUsePhoneForWhatsapp] = useState(true);
  const [shippingMethod, setShippingMethod] = useState("standard");

  const [productEmails, setProductEmails] = useState<Record<string, string>>({});

  const formattedTotal = new Intl.NumberFormat("en-IN", {
    style: "currency",
    currency: "INR",
  }).format(getCartTotal());

  const handleEmailChange = (productId: string, emailValue: string) => {
    setProductEmails(prevEmails => ({
      ...prevEmails,
      [productId]: emailValue
    }));
  };

  const handleCheckout = async (e: React.FormEvent) => {
    e.preventDefault();
    setLoading(true);

    const orderId = Math.floor(10000000 + Math.random() * 90000000).toString();
    const orderDate = new Date().toLocaleDateString("en-US", {
      year: "numeric",
      month: "long",
      day: "numeric",
    });

    try {
      const orderItems = cartItems.map(item => {
        const { product, quantity } = item;
        const itemPrice = product.discount
          ? product.price * (1 - product.discount / 100)
          : product.price;

        return {
          id: product.id,
          name: product.name,
          quantity: quantity,
          price: itemPrice
        };
      });

      const subtotal = getCartTotal();
      const tax = subtotal * 0.08;
      const total = subtotal + tax;
```

Code Snippet 2: Checkout page

3.2 Backend Implementation

The backend of the ShopEZ project is built using Node.js and the Express.js framework. It facilitates communication between the frontend client and the MongoDB database using standard HTTP protocols. Four primary HTTP methods—GET, POST, PUT, and DELETE—are used to handle client requests and server responses.
Each route is responsible for a specific operation and has access control depending on user roles (Admin, Seller, Buyer). A summary of some implemented routes is shown in Table 1 below.

Table 1: Route Command Table

Route	**Purpose**	**Access**
`router.get("/", getUsers)`	Get user information	Admin
`router.delete('/:id', deleteUser)`	Delete user	Admin
`router.get("/search/:searchQuery", getProducts)`	Search product	Public
`router.post("/seller/product", createProduct)`	Product creation	Seller
`router.patch("/seller/setSold", setSold)`	Mark product as sold	Seller
`router.put("/delivered/:id", updateOrderToDelivered)`	Update delivery	Admin
`router.get("/admin",`		

`getOrders)`	View all orders	Admin

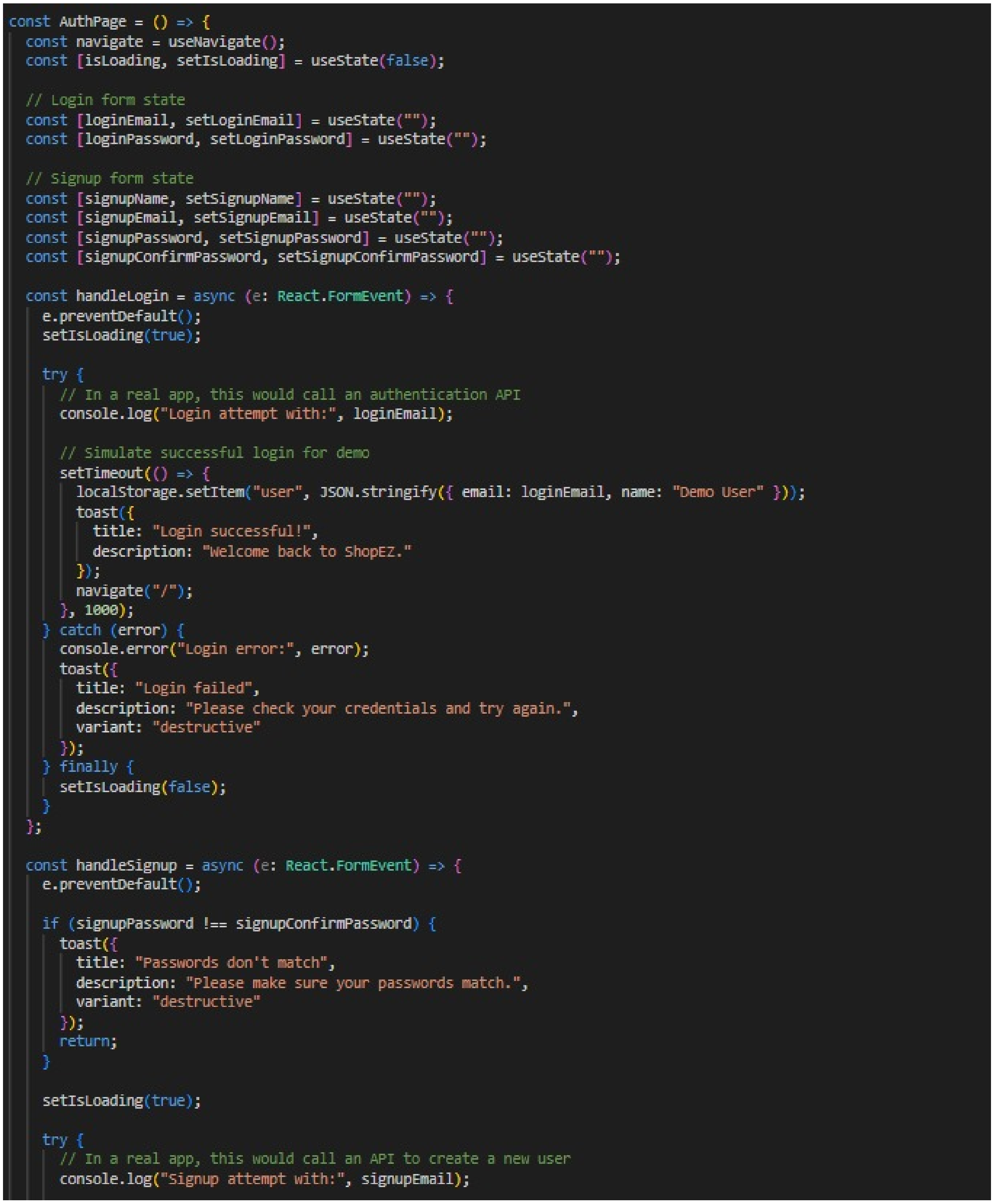
Each route maps to a controller that contains logic for handling request data and generating responses. Depending on the outcome of the operation, a relevant HTTP status code is returned

Table 2: HTTP Status Table

Status	*Function*
res.status(400)	*Bad Request*
res.status(201)	*Created*
res.status(500)	*Internal Server Error*

To manage secure access for different user roles, JWT (JSON Web Tokens) is used for authentication. Upon login, a token is generated and sent to the client, which is then used to authorize protected API requests. The token is validated by matching it with the secret key defined in the environment variables.
API functionality was tested using Postman. For example, sending a GET request to http://localhost:5000/api/categories returns a list of product categories in JSON format.

For simplicity a login page and login page component were created separately. In the log in page, the component contains all the Bootstrap elements, objects, and condition and from the login page when the user signs up and logs into their ac count and "loginUserApiRequest" function is created and POST the data to 18 /api/user/login. By this backend folder the function gets the email and password and if the validation is successful the token with data is stored into "userinfo" and *if the process fails, then an error massage is passed in payload and takes the user to a corresponding route which is the log in page and shows an error message in there*.



```
const AuthPage = () => {
  const navigate = useNavigate();
  const [isLoading, setIsLoading] = useState(false);

  // Login form state
  const [loginEmail, setLoginEmail] = useState("");
  const [loginPassword, setLoginPassword] = useState("");

  // Signup form state
  const [signupName, setSignupName] = useState("");
  const [signupEmail, setSignupEmail] = useState("");
  const [signupPassword, setSignupPassword] = useState("");
  const [signupConfirmPassword, setSignupConfirmPassword] = useState("");

  const handleLogin = async (e: React.FormEvent) => {
    e.preventDefault();
    setIsLoading(true);

    try {
      // In a real app, this would call an authentication API
      console.log("Login attempt with:", loginEmail);

      // Simulate successful login for demo
      setTimeout(() => {
        localStorage.setItem("user", JSON.stringify({ email: loginEmail, name: "Demo User" }));
        toast({
          title: "Login successful!",
          description: "Welcome back to ShopEZ."
        });
        navigate("/");
      }, 1000);
    } catch (error) {
      console.error("Login error:", error);
      toast({
        title: "Login failed",
        description: "Please check your credentials and try again.",
        variant: "destructive"
      });
    } finally {
      setIsLoading(false);
    }
  };

  const handleSignup = async (e: React.FormEvent) => {
    e.preventDefault();

    if (signupPassword !== signupConfirmPassword) {
      toast({
        title: "Passwords don't match",
        description: "Please make sure your passwords match.",
        variant: "destructive"
      });
      return;
    }

    setIsLoading(true);

    try {
      // In a real app, this would call an API to create a new user
      console.log("Signup attempt with:", signupEmail);
```

Code Snippet 3: Login/Signup Page

3.2.1 Database Implementation

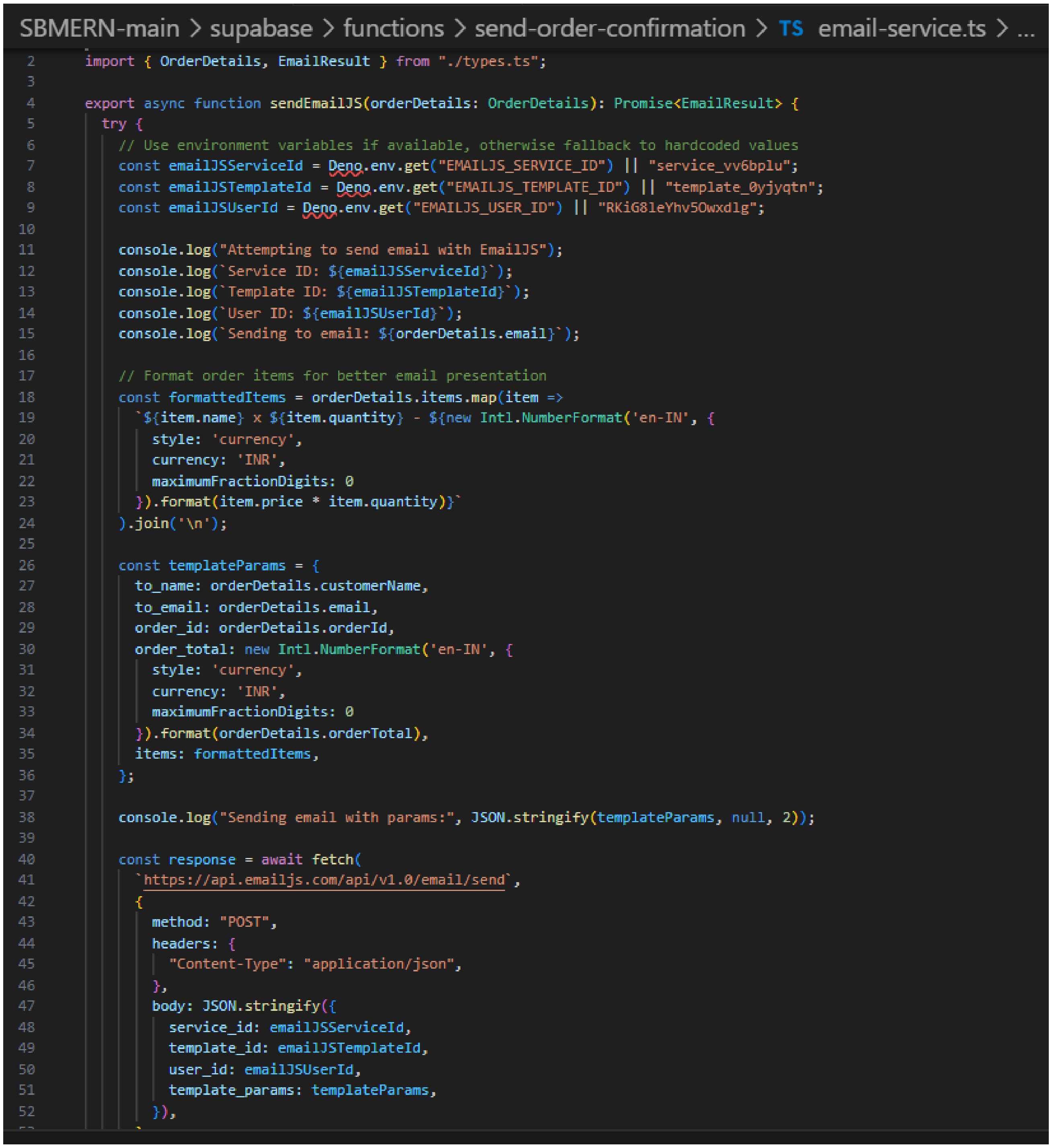
The database used for this project is MongoDB, and Mongoose is used as the ODM (Object Data Modeling) library to define schemas and interact with MongoDB. Each collection (e.g., users, products, orders) is represented by a Mongoose model.

To assist in schema generation, generate-schema was used to convert example JSON data structures into MySQL-style table schemas, which helped visualize the database layout.

The MongoDB URI and credentials are securely stored in a .env file and accessed using environment variables. When a user or product is added, a unique ObjectId along with timestamps are stored in the database for reference and tracking.

Sensitive information such as passwords is encrypted using bcrypt. This ensures that even if the database is compromised, original passwords are protected using hashing.

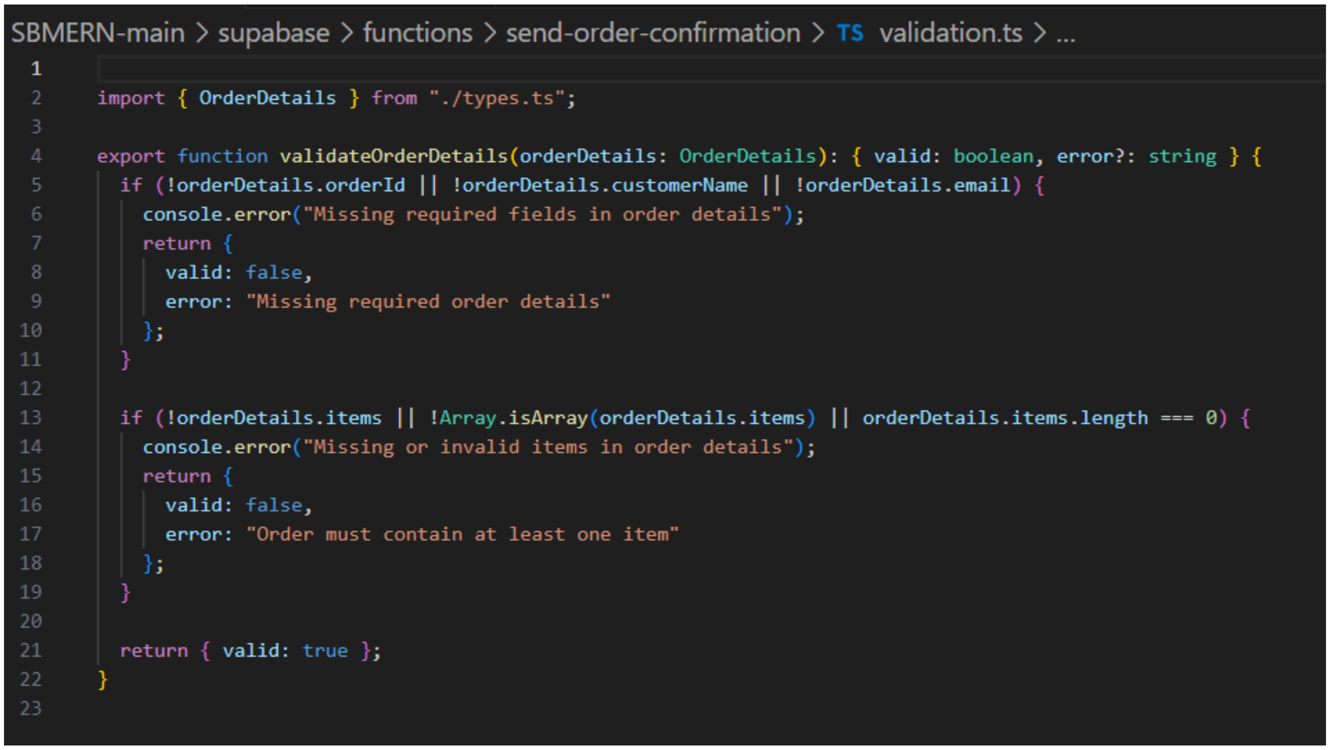
A config file manages the MongoDB connection status using asynchronous functions to avoid blocking the main thread during connection attempts.



SBMERN-main > supabase > functions > send-order-confirmation > TS email-service.ts > ...

```typescript
import { OrderDetails, EmailResult } from "./types.ts";

export async function sendEmailJS(orderDetails: OrderDetails): Promise<EmailResult> {
  try {
    // Use environment variables if available, otherwise fallback to hardcoded values
    const emailJSServiceId = Deno.env.get("EMAILJS_SERVICE_ID") || "service_vv6bplu";
    const emailJSTemplateId = Deno.env.get("EMAILJS_TEMPLATE_ID") || "template_0yjyqtn";
    const emailJSUserId = Deno.env.get("EMAILJS_USER_ID") || "RKiG8leYhv5Owxdlg";

    console.log("Attempting to send email with EmailJS");
    console.log(`Service ID: ${emailJSServiceId}`);
    console.log(`Template ID: ${emailJSTemplateId}`);
    console.log(`User ID: ${emailJSUserId}`);
    console.log(`Sending to email: ${orderDetails.email}`);

    // Format order items for better email presentation
    const formattedItems = orderDetails.items.map(item =>
      `${item.name} x ${item.quantity} - ${new Intl.NumberFormat('en-IN', {
        style: 'currency',
        currency: 'INR',
        maximumFractionDigits: 0
      }).format(item.price * item.quantity)}`
    ).join('\n');

    const templateParams = {
      to_name: orderDetails.customerName,
      to_email: orderDetails.email,
      order_id: orderDetails.orderId,
      order_total: new Intl.NumberFormat('en-IN', {
        style: 'currency',
        currency: 'INR',
        maximumFractionDigits: 0
      }).format(orderDetails.orderTotal),
      items: formattedItems,
    };

    console.log("Sending email with params:", JSON.stringify(templateParams, null, 2));

    const response = await fetch(
      `https://api.emailjs.com/api/v1.0/email/send`,
      {
        method: "POST",
        headers: {
          "Content-Type": "application/json",
        },
        body: JSON.stringify({
          service_id: emailJSServiceId,
          template_id: emailJSTemplateId,
          user_id: emailJSUserId,
          template_params: templateParams,
        }),
```

Code Snippet 4: Email service of ShopEZ



SBMERN-main > supabase > functions > send-order-confirmation > TS validation.ts > ...

```typescript

import { OrderDetails } from "./types.ts";

export function validateOrderDetails(orderDetails: OrderDetails): { valid: boolean, error?: string } {
  if (!orderDetails.orderId || !orderDetails.customerName || !orderDetails.email) {
    console.error("Missing required fields in order details");
    return {
      valid: false,
      error: "Missing required order details"
    };
  }

  if (!orderDetails.items || !Array.isArray(orderDetails.items) || orderDetails.items.length === 0) {
    console.error("Missing or invalid items in order details");
    return {
      valid: false,
      error: "Order must contain at least one item"
    };
  }

  return { valid: true };
}

```

Code Snippet 5: Validation of Database

3.2.2 Deployment

Deployment for ShopEZ was done using Render, a cloud platform that simplifies deployment of full-stack applications.

Steps for deployment:

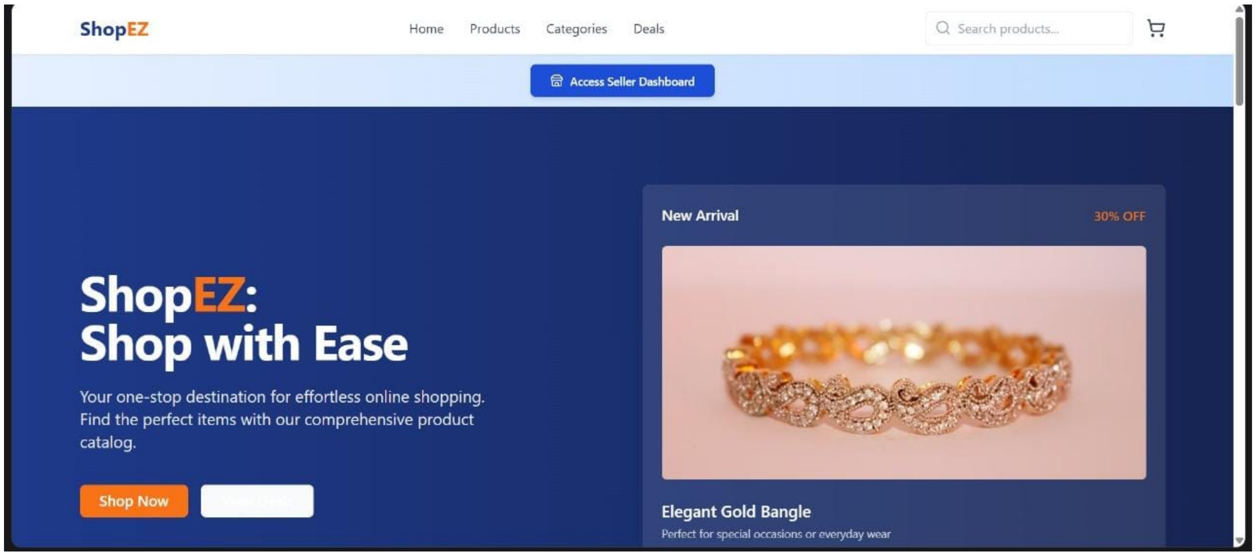
1. *A .gitignore file was configured to exclude the .env file.*
2. *The complete project was pushed to a GitHub repository using a personal access token.*
3. *The GitHub repository was connected to Render, and services were set up for both frontend and backend applications.*
4. *After providing necessary credentials and environment variables, the applications were successfully deployed.*

4. RESULT

In this chapter the project is demonstrated and described.

4.1 Home Page

The homepage of the ShopEZ e-commerce website offers a clean and user-friendly design, highlighting easy and convenient online shopping. It features a top navigation bar with links to Home, Products, Categories, Deals, a search bar, and a shopping cart. A prominent button allows sellers to access their dashboard. The main section showcases the brand tagline "ShopEZ: Shop with Ease," along with a clear call-to-action to start shopping. A featured product section highlights a new arrival—an elegant gold bangle—with a 30% discount, attracting customer attention. The overall layout is modern, visually appealing, and focused on a smooth shopping experience



ShopEZ:
Shop with Ease

Your one-stop destination for effortless online shopping. Find the perfect items with our comprehensive product catalog.

Elegant Gold Bangle

Perfect for special occasions or everyday wear

Figure 7: Home page

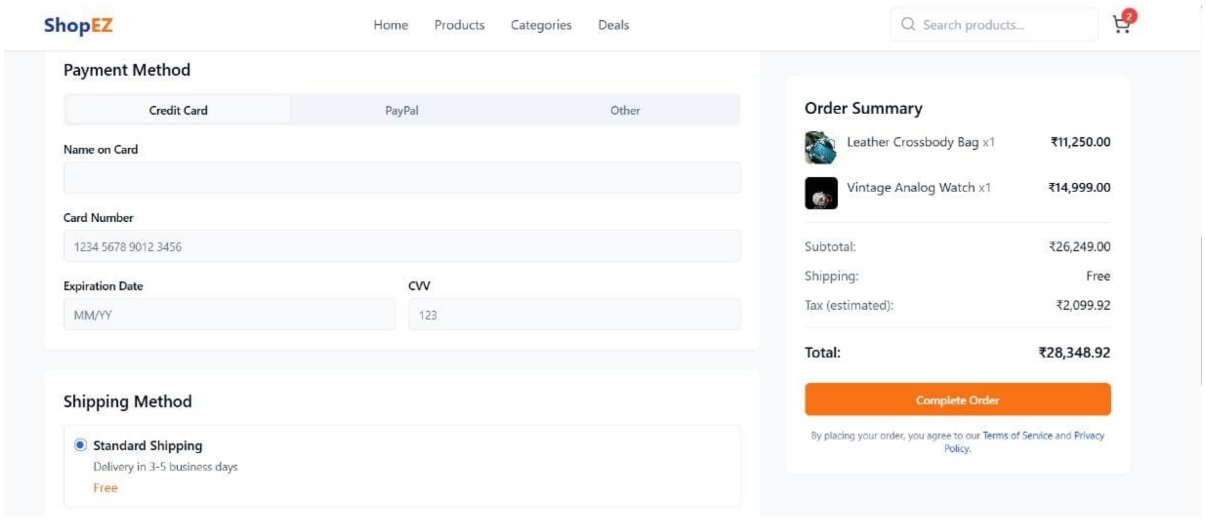
4.2 User Dashboard

The User Dashboard in ShopEZ serves as a centralized hub for customers to manage their shopping activities and account-related *information. It enhances user experience by offering quick access to key features in a visually intuitive layout.*

4.2.1 Payment Page

The ShopEZ payment page provides a clean and streamlined checkout experience for users. On the left side, customers can choose their preferred payment method—Credit Card, PayPal, or other options—and securely enter their card details including name, card number, expiration date, and CVV. Below that, users can select a shipping method, with the current option offering free standard shipping (delivery in 3–5 business days).

On the right, there's a clear order summary showing the items being purchased—Leather Crossbody Bag and Vintage Analog Watch—along with their prices. The subtotal, tax, and final total amount are calculated and displayed transparently. A bright "Complete Order" button prompts users to finalize the purchase. The page also includes a note that by completing the order, the user agrees to the site's Terms of Service and Privacy Policy, ensuring compliance and trust.



Payment Method

Name on Card

Card Number
1234 5678 9012 3456

Expiration Date
MM/YY

CVV
123

Shipping Method

Standard Shipping
Delivery in 3-5 business days
Free

Order Summary

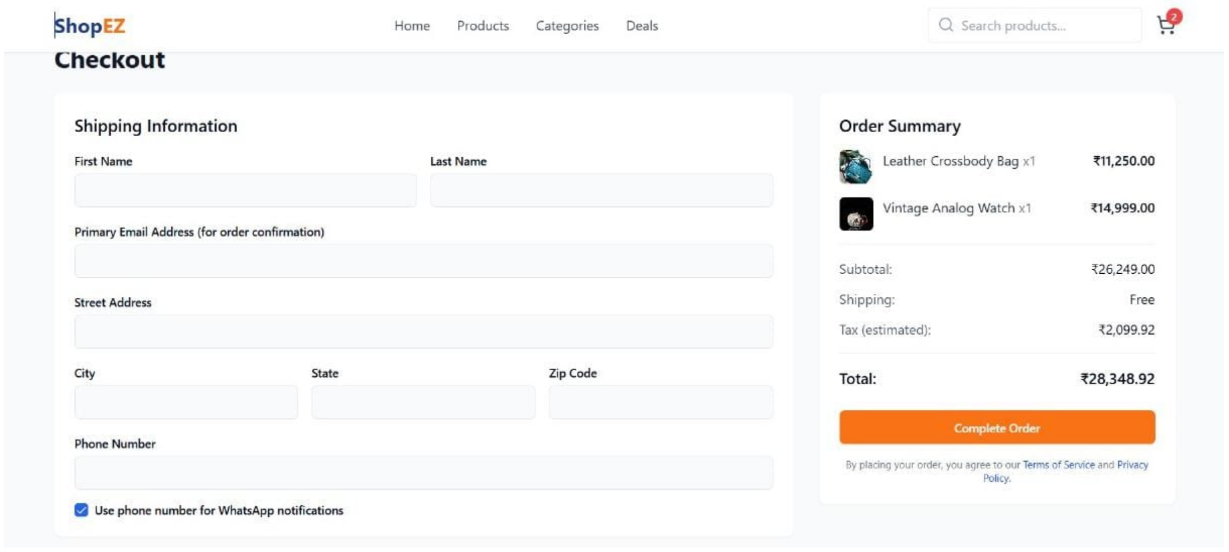
Leather Crossbody Bag x1	₹11,250.00
Vintage Analog Watch x1	₹14,999.00
Subtotal:	₹26,249.00
Shipping:	Free
Tax (estimated):	₹2,099.92
Total:	₹28,348.92

By placing your order, you agree to our Terms of Service and Privacy Policy.

Figure 8: Payment page

4.2.2 Checkout Page

The ShopEZ checkout page is designed to capture essential shipping information in a simple and organized layout. Users are prompted to fill in their first and last names, email address (for order confirmation), street address, city, state, zip code, and phone number. There's also an option to use the phone number for WhatsApp notifications, enhancing communication.


ShopEZ

Home Products Categories Deals

Search products...

Checkout

Shipping Information

First Name

Last Name

Primary Email Address (for order confirmation)

Street Address

City

State

Zip Code

Phone Number

☑ Use phone number for WhatsApp notifications

Order Summary

Leather Crossbody Bag x1	₹11,250.00
Vintage Analog Watch x1	₹14,999.00
Subtotal:	₹26,249.00
Shipping:	Free
Tax (estimated):	₹2,099.92
Total:	₹28,348.92

Complete Order

By placing your order, you agree to our Terms of Service and Privacy Policy.

Figure 9: Checkout page

4.2.3 Product Categories

The Products page of ShopEZ displays a wide range of high-quality items, organized into categories like Accessories, Beauty, Clothing, Electronics, Home, and Jewelry. Users can filter products by category, search using keywords, or set a preferred price range using the slider. Featured products such as the Elegant Gold Bangle, Silver Chain Bracelet, Leather Crossbody Bag, and Vintage Analog Watch are showcased with images, prices, star ratings, and stock availability. A sorting option allows users to arrange products based on preferences like "Featured." The layout is clean and user-friendly, making browsing easy and engaging.

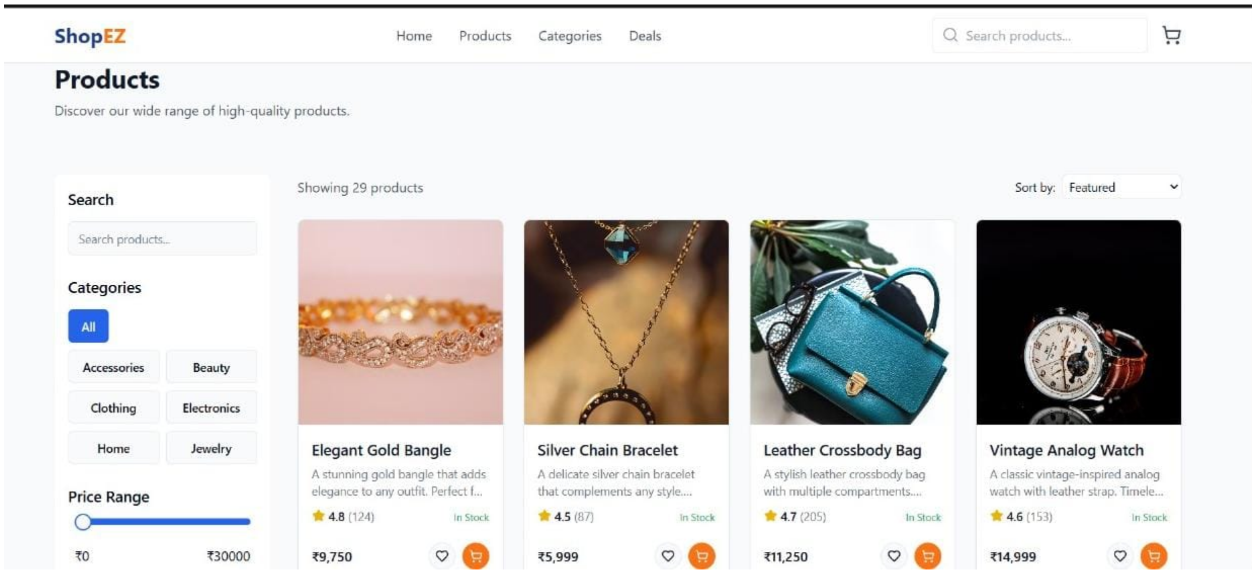
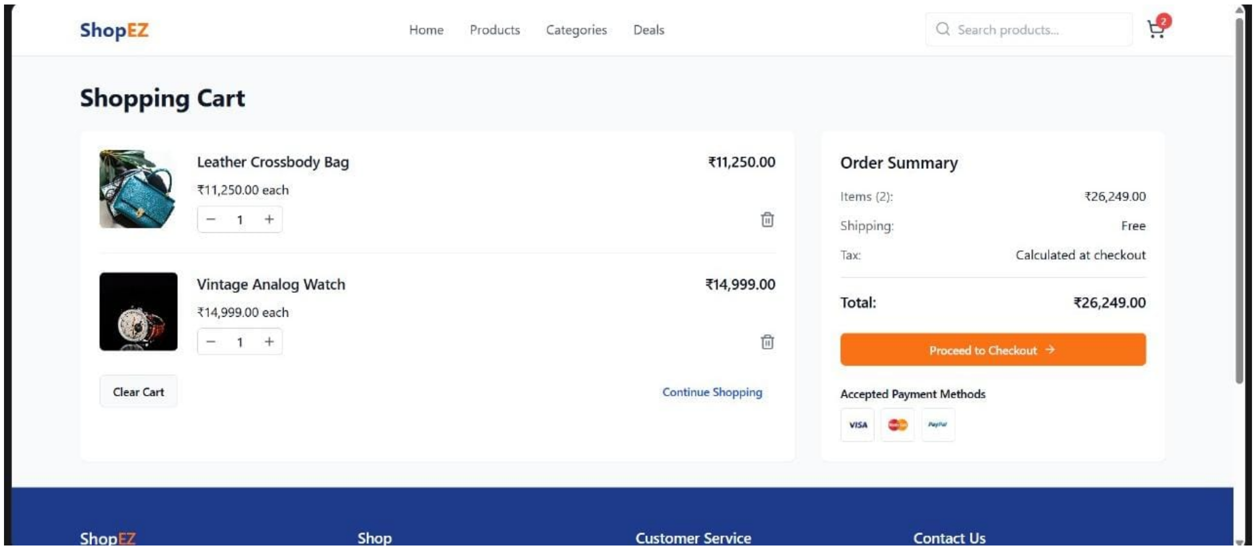


Figure 10: Product Categories

4.2.4 Cart Page

The ShopEZ cart page offers a user-friendly and visually appealing interface that allows customers to easily review the items they intend to purchase. It displays product details such as the name, image, price, and quantity for each item, along with options to increase, decrease, or remove items from the cart. The order summary section on the right provides a clear breakdown of the total cost, including the number of items, shipping (which is free), and a note that taxes will be calculated at checkout. A prominent "Proceed to Checkout" button encourages users to move forward with their purchase. Additionally, the page shows accepted payment methods like Visa, MasterCard, and PayPal, reassuring users of secure and flexible payment options.



Shopping Cart

Leather Crossbody Bag ₹11,250.00
₹11,250.00 each
1

Vintage Analog Watch ₹14,999.00
₹14,999.00 each
1

Order Summary

Items (2):	₹26,249.00
Shipping:	Free
Tax:	Calculated at checkout
Total:	₹26,249.00

Figure 11: Cart page

4.2.5 Wishlist Page

The My Wishlist page allows users to save and manage products they're interested in purchasing later. Each item card displays a product image, name, price, and two action buttons—Add to Cart and Remove. This feature enhances user convenience and supports decision-making by storing preferred products for future access.

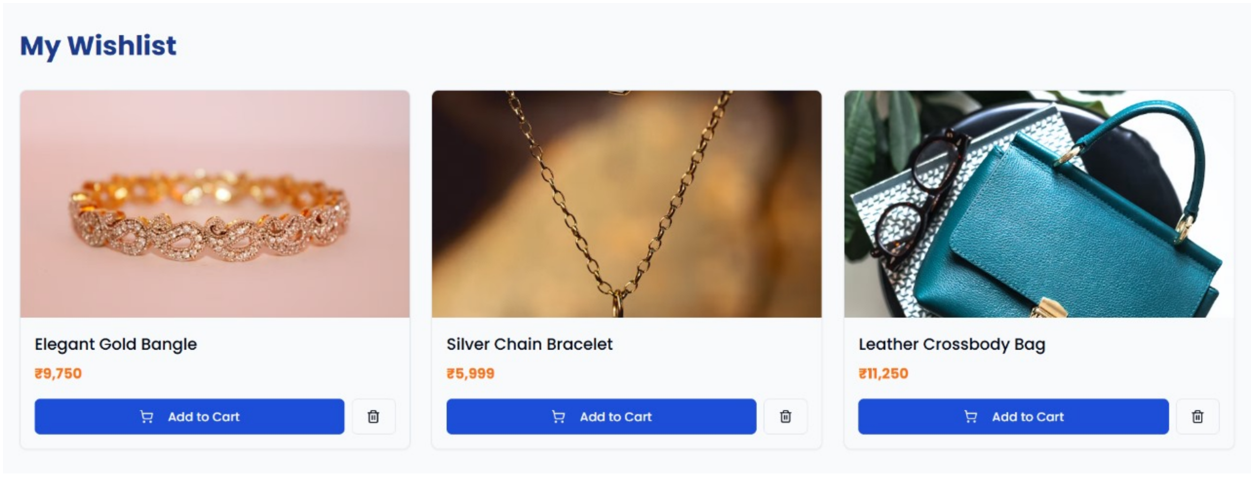
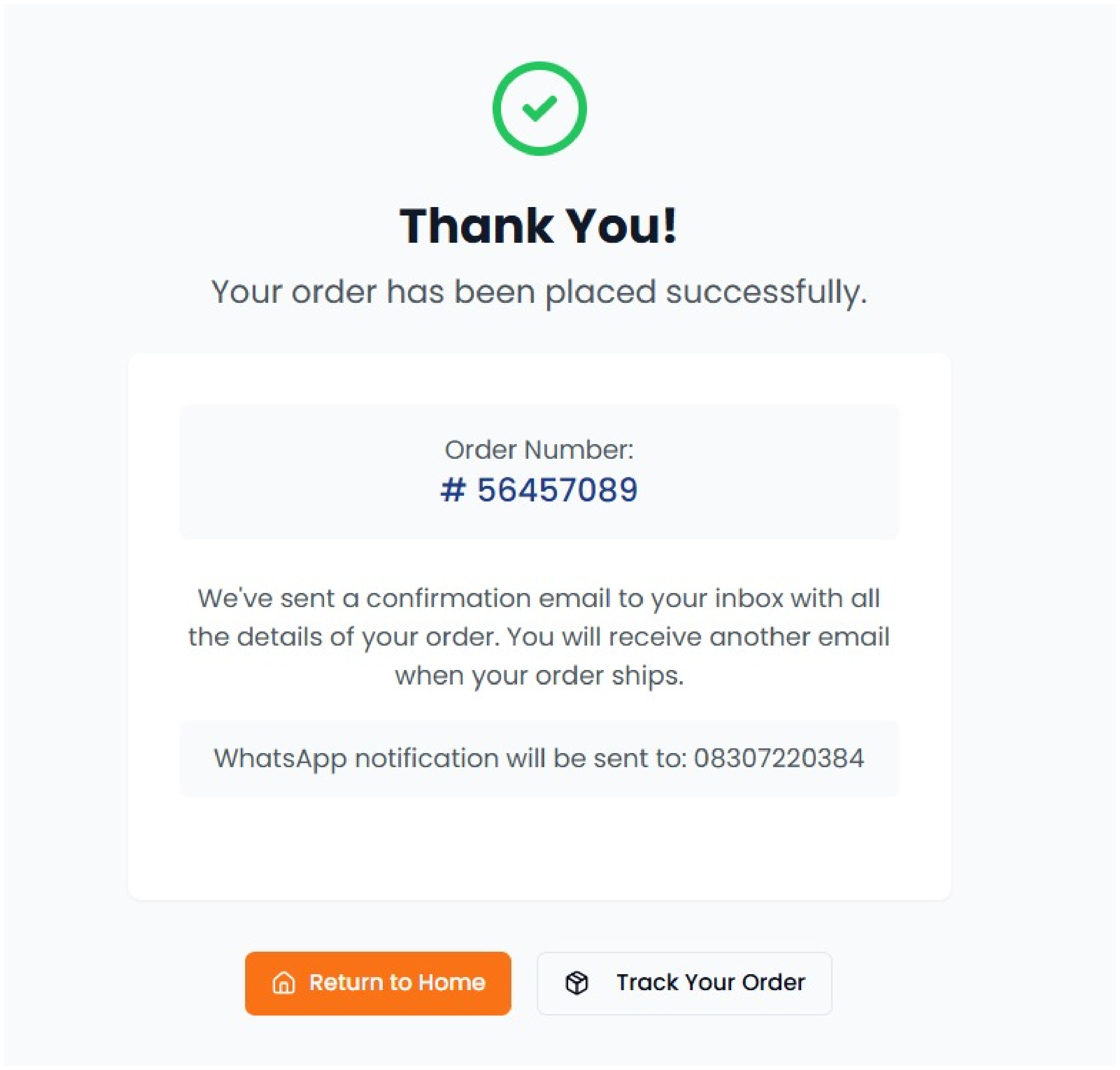


Figure 12: Wishlist page

4.2.6 Order Confirmation

The order confirmation feature in an e-commerce application plays a *crucial role in enhancing user trust and providing a seamless post-purchase experience. Upon successfully placing an order, users are presented with a confirmation message, including essential details such as the order number and communication updates. Notifications via email and messaging platforms like WhatsApp ensure users remain informed about their purchase status. Additionally, action buttons for returning to the homepage or tracking orders improve usability and streamline navigation. This functionality demonstrates the platform's commitment to transparency, convenience, and customer satisfaction.*



Thank You!

Your order has been placed successfully.

Order Number:
56457089

We've sent a confirmation email to your inbox with all the details of your order. You will receive another email when your order ships.

WhatsApp notification will be sent to: 08307220384

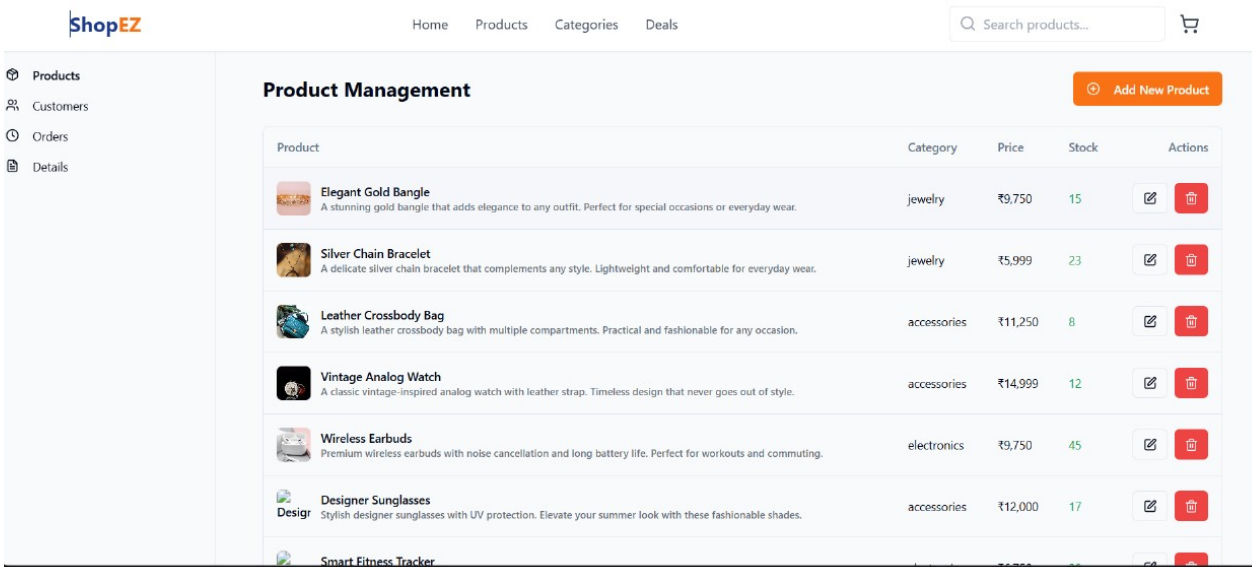
Figure 13: Order Confirmation Page

4.7 Seller Dashboard

The ShopEZ seller dashboard offers a streamlined interface for vendors to efficiently manage their products. Sellers can add, edit, or delete items, and monitor stock, price, and category details at a glance. With intuitive navigation and real-time inventory tracking, the dashboard—built using React.js and styled with Bootstrap—simplifies product management. It empowers sellers to keep their listings updated, enhancing the overall shopping experience on the platform.

4.7.1 Product Management Page

The ShopEZ seller product management page provides a clear and organized interface for managing product listings. Sellers can view product details such as name, category, price, stock, and description in a tabular format. The dashboard allows easy interaction with each product through edit and delete actions. With features like "Add New Product", it simplifies the process of updating inventory. This React-based component ensures a seamless user experience, enabling sellers to maintain their listings efficiently and keep their shop up to date.



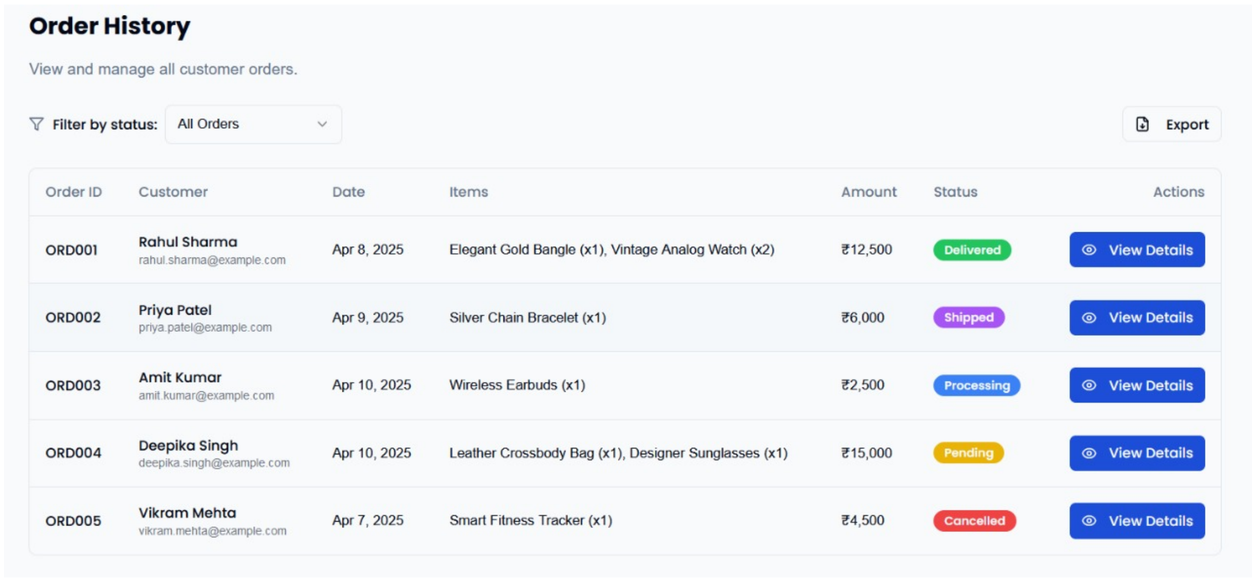
Product Management

Product	Category	Price	Stock	Actions
Elegant Gold Bangle A stunning gold bangle that adds elegance to any outfit. Perfect for special occasions or everyday wear.	jewelry	₹9,750	15	
Silver Chain Bracelet A delicate silver chain bracelet that complements any style. Lightweight and comfortable for everyday wear.	jewelry	₹5,999	23	
Leather Crossbody Bag A stylish leather crossbody bag with multiple compartments. Practical and fashionable for any occasion.	accessories	₹11,250	8	
Vintage Analog Watch A classic vintage-inspired analog watch with leather strap. Timeless design that never goes out of style.	accessories	₹14,999	12	
Wireless Earbuds Premium wireless earbuds with noise cancellation and long battery life. Perfect for workouts and commuting.	electronics	₹9,750	45	
Desigr Designer Sunglasses Stylish designer sunglasses with UV protection. Elevate your summer look with these fashionable shades.	accessories	₹12,000	17	
Smart Fitness Tracker				

Figure 14: Product Management Page

4.7.2 Order History Page

The Order History section allows sellers to efficiently monitor and manage all customer purchases. Each order entry includes key details such as Order ID, Customer Name, Email, Order Date, Items Purchased, Total Amount, and the Current Status (e.g., Delivered, Shipped, Processing, Pending, Cancelled). This helps sellers track fulfillment progress, provide customer support, and maintain transparency. A filter option and "View Details" button for each order enhance order tracking and management.



Order History

View and manage all customer orders.

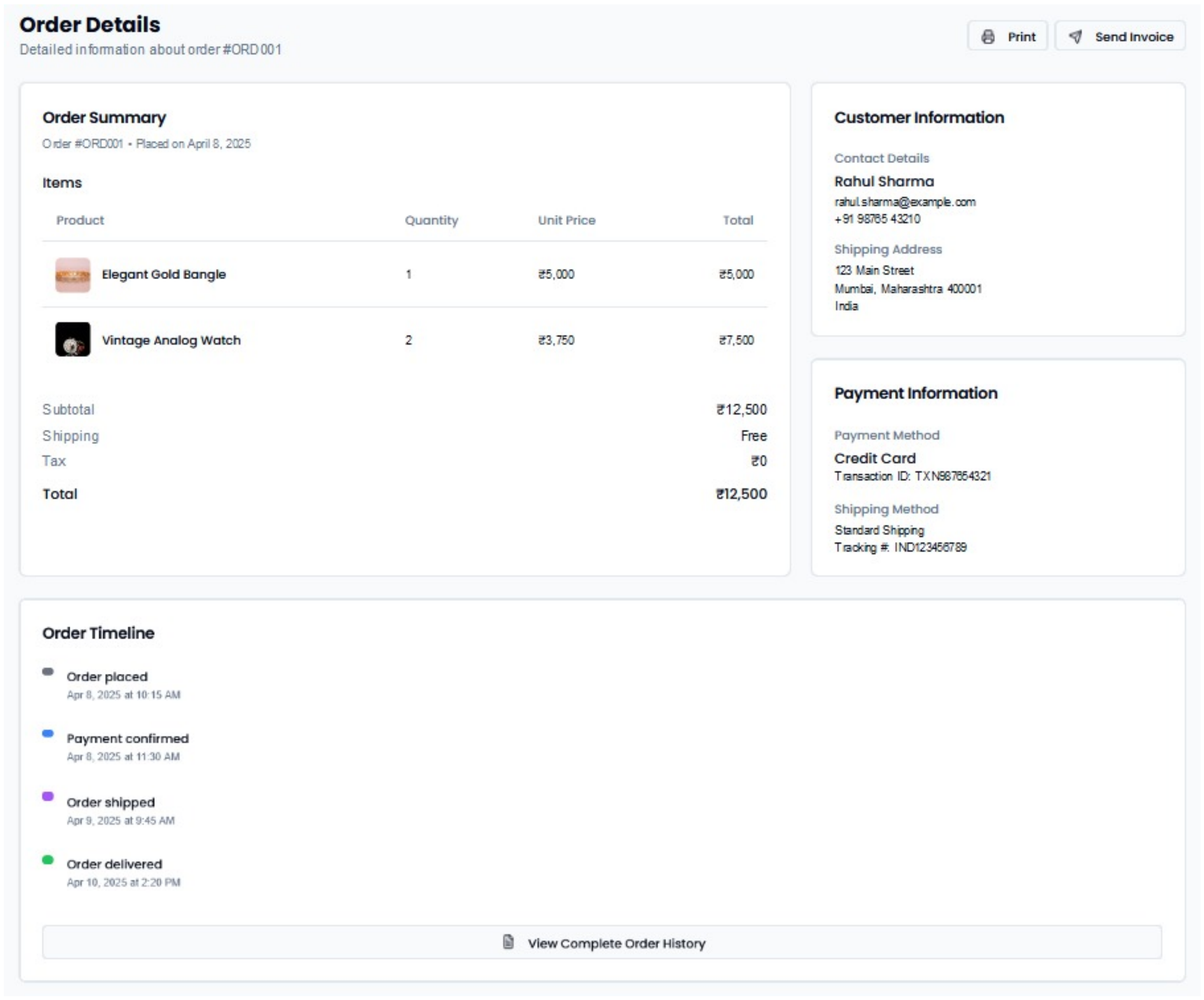
Filter by status: All Orders

Order ID	Customer	Date	Items	Amount	Status	Actions
ORD001	Rahul Sharma rahul.sharma@example.com	Apr 8, 2025	Elegant Gold Bangle (x1), Vintage Analog Watch (x2)	₹12,500	Delivered	View Details
ORD002	Priya Patel priya.patel@example.com	Apr 9, 2025	Silver Chain Bracelet (x1)	₹6,000	Shipped	View Details
ORD003	Amit Kumar amit.kumar@example.com	Apr 10, 2025	Wireless Earbuds (x1)	₹2,500	Processing	View Details
ORD004	Deepika Singh deepika.singh@example.com	Apr 10, 2025	Leather Crossbody Bag (x1), Designer Sunglasses (x1)	₹15,000	Pending	View Details
ORD005	Vikram Mehta vikram.mehta@example.com	Apr 7, 2025	Smart Fitness Tracker (x1)	₹4,500	Cancelled	View Details

Figure 15: Order History Page

4.7.3 Order Details

The Order Details page displays comprehensive information for individual orders. It includes a summary of items purchased with quantity and price, customer contact and shipping details, and payment method with transaction and tracking info. An order timeline visualizes each stage—placed, confirmed, shipped, and delivered—enabling transparent order tracking for both seller and customer.



Order Details

Detailed information about order #ORD001

Order Summary

Order #ORD001 • Placed on April 8, 2025

Items

Product	Quantity	Unit Price	Total
Elegant Gold Bangle	1	₹5,000	₹5,000
Vintage Analog Watch	2	₹3,750	₹7,500

Subtotal	₹12,500
Shipping	Free
Tax	₹0
Total	**₹12,500**

Customer Information

Contact Details
Rahul Sharma
rahul.sharma@example.com
+91 98765 43210

Shipping Address
123 Main Street
Mumbai, Maharashtra 400001
India

Payment Information

Payment Method
Credit Card
Transaction ID: TXN987654321

Shipping Method
Standard Shipping
Tracking #: IND123456789

Order Timeline

Order placed
Apr 8, 2025 at 10:15 AM

Payment confirmed
Apr 8, 2025 at 11:30 AM

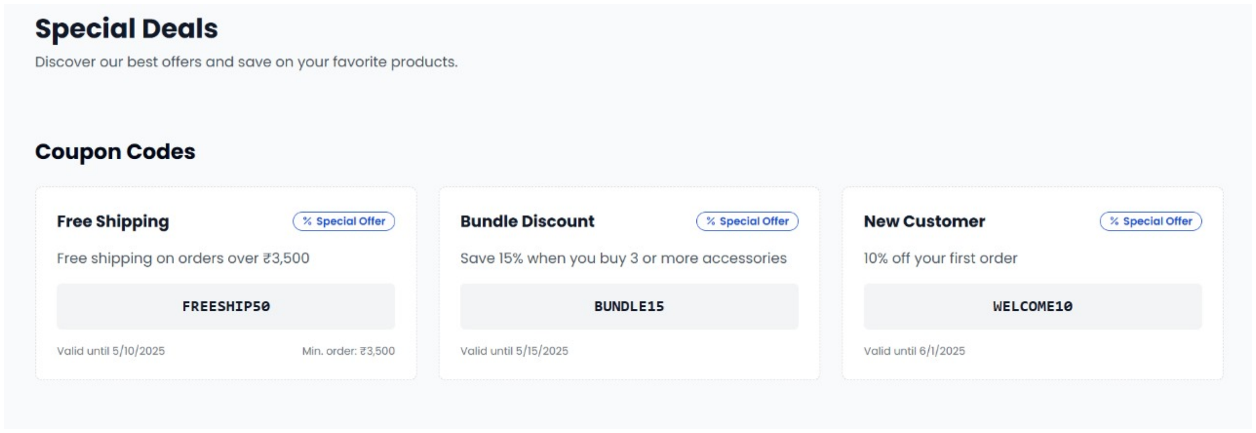
Order shipped
Apr 9, 2025 at 9:45 AM

Order delivered
Apr 10, 2025 at 2:20 PM

Figure 16: order details

4.8 Deals Page

The "Special Deals" section on Shopez offers three limited-time coupon codes: free shipping on orders over ₹3,500 with code FREESHIP50, a 15% discount on buying 3 or more accessories using BUNDLE15, and 10% off for new customers with WELCOME10. These special offers are valid through May and early June 2025, helping customers save more while shopping.



Special Deals

Discover our best offers and save on your favorite products.

Coupon Codes

Free Shipping — % Special Offer
Free shipping on orders over ₹3,500
`FREESHIP50`
Valid until 5/10/2025 — Min. order: ₹3,500

Bundle Discount — % Special Offer
Save 15% when you buy 3 or more accessories
`BUNDLE15`
Valid until 5/15/2025

New Customer — % Special Offer
10% off your first order
`WELCOME10`
Valid until 6/1/2025

Figure 17: Deals Page

5. ***DISCUSSION***

*This chapter discusses the key problems addressed by the **ShopEZ** project and the knowledge gained through its development process.*

***ShopEZ** is designed to simplify the buying and selling process by providing a structured platform that supports three types of users—**Admin, Seller, and Buyer**. The project streamlines product listing, order management, and user authentication within one unified platform. While the project's primary focus was on frontend development using **React**, significant backend features were also implemented using **Node.js**, **Express**, and **MongoDB**.*

*From the frontend perspective, UI components such as login, registration, dashboards, product listings, and order management interfaces were built using **React** and styled using **React-Bootstrap**. State management across components was handled with **Redux**, allowing seamless data flow and user state control throughout the application.*

*On the backend side, multiple routes and controllers were created for each user role. **JWT-based authentication** and **role-based access control** were implemented to ensure secure access to sensitive functionalities. The **MongoDB** database was integrated with the help of **Mongoose**, and APIs were tested using **Postman** before being connected to the frontend.*
From this project, several key takeaways include:

- *Practical understanding of **React**, **Redux**, and dynamic component-based UI design.*
- *Strong grasp of backend operations including **routing**, **middleware**, **controller logic**, and **API development**.*
- *Learning how to manage a NoSQL database using MongoDB.*

- *Real-world experience in deploying full-stack applications using platforms like* ***Render****.*
- *Improved debugging and testing skills using tools like* ***Postman*** *and browser dev tools.*

Limitations and Future Enhancements:

Due to time constraints, some features like the ***Cart Page*** *and* ***Buyer Profile Page*** *remain incomplete or non-functional on the live server. These will be fixed in future iterations. Additionally:*

- ***Authentication and user validation*** *can be made more secure and robust.*
- *The* ***UI/UX*** *can be enhanced to provide a more modern and responsive experience.*
- *Features such as* ***payment gateway integration****,* ***order tracking****, and* ***notifications*** *can be added to elevate the application to production-level readiness.*

6. CONCLUSIONS

ShopEZ is a modern e-commerce solution aimed at building a fast, user-friendly, and efficient platform for youth entrepreneurs and small businesses. The application provides a structured marketplace where sellers can manage products and buyers can place orders while admins oversee the entire system.

The project was successfully developed using the MERN stack (MongoDB, Express.js, React.js, Node.js), which ensured a smooth and scalable full-stack architecture. React was used to craft a dynamic UI, while Express and Node handled server-side logic. MongoDB served as the database solution for efficient and flexible data management.

Despite a few limitations in terms of features, this project demonstrates the potential of web development using the MERN stack and provides a solid foundation for future scalability and enhancements. With additional features, improved UI, and stronger authentication, ShopEZ has the potential to evolve into a commercially viable web application.
In conclusion, ShopEZ blends modern technology with practical use-case needs and sets the stage for further innovation in the e-commerce space.