



JEPPIAAR
ENGINEERING COLLEGE

SHOPEZ: E-COMMERCE APPLICATION MERN

A PROJECT REPORT

SUBMITTED BY

MADHAN KUMAR.S - 310821104051
MONISH KRISHNAN.S - 310821104024
MOHAMED RAAID.T - 310821104058
ARUN JEGASH M - 310821104303
MITHUN.K - 310821104056

IN PARTIAL FULFILLMENT FOR THE AWARD OF THE DEGREE

OF

BACHELOR OF ENGINEERING

IN

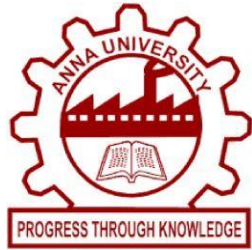
COMPUTER SCIENCE ENGINEERING

JEPPIAAR ENGINEERING COLLEGE

ANNA UNIVERSITY

CHENNAI – 600025

NOVEMBER 2024



ANNA UNIVERSITY: CHENNAI 600 025

BONAFIDE CERTIFICATE

Certified that this project report **“SHOPEZ:E-COMMERCE APPLICATION MERN”** is the bonafide work of **“MADHAN KUMAR.S(310821104051)”**, **“MONISH KRISHNAN.S(310821104059)”**, **“MITHUN.K(310821104056)”**, **“ARUN JEGASH M - 310821104303”**, **“MOHAMED RAAID.T(310821104058)”** who carried out the project under supervision.

SUPERVISOR

JEEVITHA.D

HEAD OF THE DEPARTMENT

DR.J.ANITHA GNANASELVI

TABLE OF CONTENTS

S NO	CONTENTS	PAGE NO
1	INTRODUCTION	4
2	PROJECT OVERVIEW	5
3	ARCHITECTURE	8
4	SETUP INSTRUCTIONS	12
5	FOLDER STRUCTURE	18
6	RUNNING THE APPLICATION	22
7	API DOCUMENTATION	25
8	AUTHENTICATION	30
9	USER INTERFACE	31
10	TESTING	33
11	SCREENSHOTS	35
12	RESULT	45

1. Introduction

Project Title: Shopez: E-Commerce Application with MERN Stack

This project delivers a robust online e-commerce platform where buyers and sellers can interact seamlessly. Buyers can browse products, add them to their cart, and make secure payments, while sellers can list products, manage inventory, and track orders. The platform also features real-time updates, user authentication, and efficient order management. Built using the MERN stack, the goal is to create a scalable and user-friendly application ensuring a smooth shopping experience for all users.

Team Members:

- **S. Madhan Kumar** (Frontend Developer): Responsible for building the user interface using React and ensuring the frontend's responsiveness and user experience.
- **Monish Krishnan.S** (Full Stack Developer): Works on both the frontend and backend to ensure seamless integration and overall functionality of the application.
- **Mohamed Raaid.T** (Backend Developer): Handles the backend API development with Node.js, Express.js, and MongoDB.
- **Mithun. K** (UI & UX Design) : Focused on designing the platform's interface and creating intuitive and user-friendly experiences.
- **Arun Jegash.M** (UI & UX Design) : Focused on designing the platform's interface and creating intuitive and user-friendly experiences.

The project is developed with the **MERN** (MongoDB, Express.js, React, Node.js) stack, utilizing Socket.io for real-time communication and React's context for state management on the frontend.

2. Project Overview

Purpose

The Shoppez: E-Commerce Application with MERN Stack is designed to provide a seamless platform for buyers and sellers, enabling efficient and secure online transactions. Buyers face challenges in finding a diverse range of products with trustable reviews, while sellers often struggle with reaching the right audience and managing inventory effectively. Shop addresses these challenges, offering a comprehensive solution for product listing, real-time updates, and secure payments.

The platform prioritizes simplicity and efficiency, integrating features that enhance the online shopping and selling experience. By providing advanced product management, secure payment gateways, and interactive features, ShopZ ensures a dynamic e-commerce environment for users.

The core goals of the platform include:

- **Expanding Market Accessibility:** ShopZ aims to connect buyers with a wide range of sellers across various categories, offering diverse product options. Sellers can easily reach potential customers and showcase their inventory, fostering a thriving marketplace.
- **Real-Time Updates:** Built with real-time technologies like Socket.io, ShopZ keeps buyers and sellers informed about order statuses, product availability, and notifications. This feature enhances interaction and provides a seamless shopping experience.
- **Secure Payment Gateway:** A key focus of ShopZ is secure payment processing. By integrating trusted payment gateways, the platform ensures that transactions are protected and efficient, building trust among users.
- **Efficient Product Management:** Sellers can manage inventory, update product details, and track sales seamlessly, while buyers can browse, filter,

and purchase products with ease. The centralized management system simplifies operations for both parties.

- **User-Friendly Design:** With a focus on intuitive navigation and responsive design, ShopZ ensures that users of all technical levels can engage effortlessly. The platform is optimized to deliver a smooth shopping experience across devices.

Features

1. User Authentication:

- The platform employs JSON Web Tokens (JWT) for secure user authentication, ensuring that buyers and sellers have access only to their respective accounts.
- Bcrypt is utilized to hash user passwords securely, providing robust protection against potential security breaches. This guarantees that user credentials remain safeguarded, even in the unlikely event of a data breach.

2. Product Listing and Management:

- Sellers can list products with detailed descriptions, pricing, and high-quality images. Advanced inventory management tools enable sellers to track stock levels and update product information in real-time.
- Bulk upload features allow sellers to manage large inventories efficiently, while dynamic pricing options enable promotions and discounts.

3. Real-Time Updates:

- The platform integrates Socket.io for real-time notifications, including order status updates, new product alerts, and seller responses to buyer inquiries.
- Buyers are notified of flash sales, discounts, and limited-time offers, ensuring they never miss out on deals.

4. Secure Payment Processing:

- ShopZ integrates with multiple payment gateways, allowing users to choose their preferred payment methods while ensuring transaction security.

- Features such as escrow services protect both buyers and sellers, holding payments until the order is confirmed as delivered.
- Automated invoicing and transaction history help users track their purchases and payments effortlessly.

5. Advanced Search and Filtering:

- Users can search products using filters like price range, categories, ratings, and availability. Personalized recommendations based on browsing history enhance the shopping experience.
- Sellers can filter orders by status (e.g., pending, shipped, delivered), streamlining order management.

6. Shopping Cart and Wishlist:

- Buyers can add products to a shopping cart for quick checkout or save them to a wishlist for future reference.
- The cart includes features like estimated tax calculation and multiple delivery options for convenience.

7. Admin Panel:

- Administrators can manage user accounts, oversee transactions, handle disputes, and monitor platform activity.
- Admin tools allow for the moderation of reviews, approval of product listings, and removal of fraudulent content, ensuring a secure and trusted platform for all users.

8. Product Reviews and Ratings:

- Buyers can leave detailed reviews and ratings for purchased products, helping other users make informed decisions.
- Sellers receive insights from reviews, enabling them to improve their offerings and customer service.

9. Order Tracking and Management:

- Buyers can track orders in real-time, with updates on shipping and delivery status.
- Sellers have access to dashboards for managing orders, generating sales reports, and monitoring customer feedback.

10. Multi-Language and Multi-Currency Support:

- The platform supports multiple languages and currencies, ensuring accessibility for a global audience.
- Automatic currency conversion based on the buyer's location simplifies international transactions.

11. Loyalty Programs and Discounts:

- ShopZ includes loyalty programs, offering reward points for purchases that can be redeemed for discounts.
- Seasonal and bulk purchase discounts attract buyers, fostering customer retention.

3. Architecture

Frontend Architecture

The frontend of the Shopez: E-Commerce Application with MERN Stack is built using React.js, leveraging a component-based architecture to ensure modularity and scalability. This design approach simplifies code maintenance and enhances the overall performance of the application. The key aspects of the frontend architecture include:

- Component-Based Design:

- Each major feature, such as product listings, shopping cart, checkout process, and user profiles, is developed as an independent React component.
- Reusable components like buttons, form inputs, and modals are implemented across the application, reducing redundancy and improving consistency.
- The modular structure ensures that features can be easily added or updated without impacting other parts of the application.

- State Management:

- The application uses the Context API for state management, ensuring a lightweight and efficient solution for managing global state.
- Context is utilized to handle states such as user authentication, shopping cart items, and product filters, ensuring seamless synchronization across components.
- The combination of Context API with useReducer provides an elegant and scalable way to manage complex state updates, while avoiding unnecessary re-renders.

- Routing:

- React Router is used to implement dynamic routing, enabling smooth navigation between pages like Home, Product Details, Categories, and Order History.
- Lazy loading is employed for route-based code splitting, ensuring that components are loaded on demand, which improves the application's performance.
- React Router Hash Link is integrated to enable smooth scrolling for anchor links within pages, enhancing user experience for features like FAQs and product details.

- UI/UX Design:

The platform is designed with CSS/SCSS, focusing on a clean, responsive layout. A mobile-first approach ensures that the platform works well on mobile devices before enhancing it for larger screens, offering an intuitive and accessible experience across all devices.

Backend Architecture :

The backend is developed using Node.js and Express.js, which together create a RESTful API. The backend handles the core operations of the platform, such as user authentication, Product management, user's Cart, and. Key features of the backend architecture include:

- API Routes:

The backend exposes various API routes for interacting with the platform's data. Using Express.js, these routes handle CRUD operations (Create, Read, Update, Delete) for users, updating price, and products.

- Security:

Bcrypt is used for hashing passwords before storing them in the database, ensuring that user credentials remain secure. JWT (JSON Web Tokens) are used for authentication, ensuring that only authorized users can access sensitive routes and data, such as user profiles and job listings.

-Database Architecture :

The platform uses MongoDB, a NoSQL database, to store data. MongoDB is flexible and scalable, making it suitable for the dynamic nature of this platform. The database is organized into collections, each serving a specific purpose. The key collections include:

- Users:

- This collection stores information about buyers and sellers, including authentication details, contact information, and profiles.
- Sellers can manage their product listings and track sales, while buyers have access to their order history, saved addresses, and wishlist.

- Products:

- The Products collection contains details about each listed item, such as title, description, price, category, stock availability, and images.
- Sellers can update product details and inventory, while buyers access these listings to browse and purchase items.

- Cart:

- The Cart collection manages items that buyers add to their shopping carts. Each cart entry links to the user and product, with details like quantity and subtotal.
- Persistent cart functionality ensures that users can revisit and complete purchases even after logging out.

-Payments:

- Payment details, including transaction IDs, payment status, and timestamps, are stored in this collection.
- It ensures transparency and easy reconciliation for both buyers and sellers while maintaining data security.

4. Setup Instructions

Prerequisites:

Before setting up the **Shopez: E-Commerce with MERN Stack** locally, ensure you have the following software installed on your machine:

1. Node.js:

- Node.js is a JavaScript runtime environment required to run both the frontend and backend.
- You can download and install Node.js from [here](#).

2. MongoDB:

- MongoDB is a NoSQL database used to store application data such as user profiles, products, orders, and messages.
- You can install MongoDB locally or use MongoDB Atlas for a cloud-based solution.

3. npm (Node Package Manager):

- npm is bundled with Node.js and is used to install the project dependencies for both the frontend and backend.

Installation

Follow these steps to set up the project locally:

1. Clone the Repository:

Start by cloning the project repository to your local machine using the following command:

```
bash
Copy code
git clone <repository-url>
```

Replace <repository-url> with the actual URL of the project repository.

2. Install Dependencies:

Navigate to each project directory and install the required dependencies.

- **Frontend (Client):**

The frontend, built with React, is located in the frontend directory.

Run the following commands:

```
bash
Copy code
cd frontend/
npm install
```

This will install all dependencies specified in package.json, such as react, react-router-dom, and bootstrap.

- **Backend (API):**

The backend, developed with Node.js and Express, is located in the backend directory. Run the following commands:

```
bash
Copy code
cd backend/
npm install
```

This will install required packages like express, mongoose, jsonwebtoken, and multer.

3. Setup Environment Variables:

Create .env files in both the frontend and backend directories to configure environment-specific variables.

- **Frontend (frontend/.env):**

Add the following variables:

```
env
Copy code
REACT_APP_API_URL=http://localhost:5000
```

This URL points to the backend server.

- **Backend (backend/.env):**

Add the following variables:

```
env
```

Copy code

```
MONGODB_URI=<your-mongodb-uri>
```

```
JWT_SECRET=<your-jwt-secret>
```

- MONGODB_URI: MongoDB connection string. Use your local database URI or the one from MongoDB Atlas.
- JWT_SECRET: A random string used to sign and verify JWT tokens for secure authentication.

4. **Run the Application:**

Use separate terminals to start the frontend and backend servers:

- **Start the Backend Server:**

Navigate to the backend directory and run:

```
bash
```

Copy code

```
cd backend/
```

```
npm start
```

The backend will run on `http://localhost:5000` by default.

- **Start the Frontend Server:**

Navigate to the frontend directory and run:

```
bash
```

Copy code

```
cd frontend/
```

```
npm start
```

The frontend will run on `http://localhost:3000` by default.

Project Directory Structure:

- **Frontend:** Contains the React-based user interface, including components for product listings, shopping cart, and user authentication.
- **Backend:** Houses the Node.js server, including APIs for user authentication, product management, order processing, and database interactions.

By following these steps, you can successfully set up the Shopeez: E-Commerce Application on your local machine for development and testing.

Access the Application

Once all the servers are running, you can access the application in your web browser:

- **Frontend:** Open your browser and go to http://localhost:3000 to interact with the user interface of the application.

- **Backend:** The backend API can be accessed at http://localhost:3001, though it is primarily used by the frontend.

5. Folder Structure

Server Folder

The server folder contains backend-related files organized to handle API routes, business logic, and database interactions.

```
Server/
|
|— Controller/           # Contains business logic for handling user operations.
|   |— userController.js # Handles user-related business logic (e.g.,
authentication, profile updates).
```

```

├── Models/           # Contains database schemas for MongoDB collections.
│   └── user.js       # Mongoose schema for user-related data (e.g., email,
password, profile details).
│
├── Route/           # Defines API endpoints for user-related actions.
│   └── userRoute.js  # Routes for user-related operations (e.g., login,
registration).
│
├── node_modules/    # Contains the installed backend dependencies.
├── .gitignore        # Specifies files and directories to ignore in version
control.
├── index.js         # Entry point for the server; sets up Express.js and API
routes.
├── package.json      # Lists project dependencies and scripts for the
backend.
└── package-lock.json # Automatically generated file for locking
dependency versions.

```

- **Controller/**: Includes userController.js, which contains the functions implementing the logic for routes such as user registration, login, and updates.
- **Models/**: Contains the user.js file, defining the Mongoose schema for storing and validating user data in MongoDB.
- **Route/**: Defines API routes. The userRoute.js file maps user-related requests (e.g., login, signup) to their respective controller functions.

Client Folder

The client folder contains the frontend files developed using React.js. These files are organized into reusable components and pages to provide the user interface.

```

Client/
│
├── src/             # Contains all source code for the React frontend.
│   ├── components/  # Reusable React components for various
functionalities.
│   │   └── Assets/  # Static assets like images or icons used in the
components.

```

- | | | — BreadCrums/ # Components for breadcrumb navigation (includes CSS and JSX files).
- | | | — CartItems/ # Components for displaying cart items in the shopping cart.
- | | | | — CartItems.css # Styling specific to cart items.
- | | | | — CartItems.jsx # React component for rendering cart item details.
- | | | — DescriptionBox/ # Component for displaying detailed product descriptions.
- | | | — HomeComp/ # Component for rendering the homepage layout.
- | | | — ProductDisplay/ # Components for displaying product information.
- | | | | — ProductDisplay.css # Styling for the product display component.
- | | | | — ProductDisplay.jsx # React component for showing product details.
- | | | — RelatedProducts/ # Components for showing related products on product pages.
- | | | — RelatedProducts.css # Styling for the related products section.
- | | | — RelatedProducts.jsx # Component for rendering related product items.
- | | |
- | | — Context/ # Context API setup for managing application-wide state.
- | | — Pages/ # Contains individual pages like Home, Products, Cart, etc.
- | | — App.css # Global CSS file for the application.
- | | — App.js # Main component that sets up routing and layout.
- | | — index.css # Styling for the application entry point.
- | | — index.js # Entry point for rendering the React app into the DOM.
- | | — reportWebVitals.js # Utility for measuring performance.
- | | — setupTests.js # Configuration file for running tests.
- |
- | — .gitignore # Specifies files and directories to ignore in version control.
- | — package.json # Lists project dependencies and scripts for the frontend.
- | — package-lock.json # Automatically generated file for locking dependency versions.
- | — node_modules/ # Contains the installed frontend dependencies.

- **components/**: Includes reusable components grouped into folders for specific functionalities like CartItems, ProductDisplay, and RelatedProducts. Each folder contains both the CSS and JSX files.
- **Context/**: Contains React Context API files to manage state across the application.
- **Pages/**: Holds page-specific components such as Home, Cart, or ProductDetails to structure different sections of the app.
- **App.js**: Serves as the root component, defining routes and setting up the app structure

6. Running the Application

To run the Freelancing Application locally, follow the steps below:

Frontend:

1. Navigate to the client directory:

```
``bash
cd client/
``
```

2. Install the required dependencies:

```
``bash
npm install
``
```

3. Start the frontend server:

```
``bash
npm start
``
```

4. The frontend will be available at http://localhost:3000.

Backend:

1. Navigate to the server directory:

```
``bash
cd server/
``
```

2. Install the required dependencies:

```
``bash
npm install
``
```

3. Start the backend server:

```
``bash
npm start
``
```

4. The backend will be available at http://localhost:3001.

By following these steps, you'll have the application running locally, with the frontend accessible on port `3000`, the backend on port `3001` server enabling real-time communication.

7.API Documentation

User Authentication

POST /api/auth/register

- **Description:** Register a new user.
- **Request Body:**

```
json
Copy code
{
  "name": "John Doe",
  "email": "johndoe@example.com",
  "password": "password123",
  "role": "customer" // or "admin"
}
```

- **Response:**

```
json
Copy code
{
  "message": "User registered successfully"
}
```

POST /api/auth/login

- **Description:** Login a user and return a JWT token.
- **Request Body:**

```
json
Copy code
{
  "email": "johndoe@example.com",
  "password": "password123"
}
```

- **Response:**

```
json
Copy code
{
  "token": "JWT_Token_Here",
  "user": {
    "id": "user_id_here",
    "name": "John Doe",
    "email": "johndoe@example.com",
    "role": "customer"
  }
}
```

Product Management

POST /api/products

- **Description:** Add a new product (admin only).
- **Request Body:**

```
json
Copy code
{
  "name": "Wireless Earbuds",
  "description": "Noise-cancelling wireless earbuds.",
  "price": 59.99,
  "category": "Electronics",
  "stock": 100,
  "image": "image_url_here"
}
```

- **Response:**

```
json
Copy code
{
  "message": "Product added successfully"
}
```

```
}
```

GET /api/products

- **Description:** Retrieve all products.
- **Response:**

json

Copy code

```
[  
  {  
    "id": "product_id_1",  
    "name": "Wireless Earbuds",  
    "description": "Noise-cancelling wireless earbuds.",  
    "price": 59.99,  
    "category": "Electronics",  
    "stock": 100,  
    "image": "image_url_here"  
  },  
  {  
    "id": "product_id_2",  
    "name": "Gaming Laptop",  
    "description": "High-performance gaming laptop with RGB keyboard.",  
    "price": 1199.99,  
    "category": "Electronics",  
    "stock": 20,  
    "image": "image_url_here"  
  }  
]
```

GET /api/products/:id

- **Description:** Retrieve details of a specific product.
- **Response:**

json

Copy code

```
{
  "id": "product_id_here",
  "name": "Wireless Earbuds",
  "description": "Noise-cancelling wireless earbuds.",
  "price": 59.99,
  "category": "Electronics",
  "stock": 100,
  "image": "image_url_here"
}
```

Cart Management

POST /api/cart

- **Description:** Add a product to the cart.
- **Request Body:**

json

Copy code

```
{
  "productId": "product_id_here",
  "quantity": 2
}
```

- **Response:**

json

Copy code

```
{
  "message": "Product added to cart successfully"
}
```

GET /api/cart

- **Description:** Retrieve the user's cart.
- **Response:**

json

Copy code

```
{
  "items": [
    {
      "product": {
        "id": "product_id_here",
        "name": "Wireless Earbuds",
        "price": 59.99,
        "image": "image_url_here"
      },
      "quantity": 2,
      "totalPrice": 119.98
    }
  ],
  "cartTotal": 119.98
}
```

DELETE /api/cart/:itemId

- **Description:** Remove an item from the cart.
- **Response:**

json

Copy code

```
{
  "message": "Item removed from cart successfully"
}
```

Order Management

POST /api/orders

- **Description:** Place an order.
- **Request Body:**

json

Copy code

```
{
  "cartItems": [
    {
      "productId": "product_id_here",
      "quantity": 2
    }
  ],
  "shippingAddress": {
    "street": "123 Main St",
    "city": "New York",
    "state": "NY",
    "zip": "10001",
    "country": "USA"
  },
  "paymentMethod": "Credit Card"
}
```

- **Response:**

json

Copy code

```
{
  "message": "Order placed successfully",
  "order": {
    "id": "order_id_here",
    "totalAmount": 119.98,
    "status": "Processing"
  }
}
```

GET /api/orders

- **Description:** Retrieve all orders of the logged-in user.
- **Response:**

json

Copy code

```
[
  {
    "id": "order_id_1",
    "items": [
      {
        "product": "Wireless Earbuds",
        "quantity": 2,
        "price": 59.99
      }
    ],
    "totalAmount": 119.98,
    "status": "Shipped"
  },
  {
    "id": "order_id_2",
    "items": [
      {
        "product": "Gaming Laptop",
        "quantity": 1,
        "price": 1199.99
      }
    ],
    "totalAmount": 1199.99,
    "status": "Processing"
  }
]
```

GET /api/orders/:id

- **Description:** Retrieve details of a specific order.
- **Response:**

json

Copy code

```
{
  "id": "order_id_here",
```

```
"items": [  
  {  
    "product": "Wireless Earbuds",  
    "quantity": 2,  
    "price": 59.99  
  }  
],  
"shippingAddress": {  
  "street": "123 Main St",  
  "city": "New York",  
  "state": "NY",  
  "zip": "10001",  
  "country": "USA"  
},  
"totalAmount": 119.98,  
"status": "Shipped"  
}
```

Admin-Specific Endpoints

GET /api/admin/orders

- **Description:** Retrieve all orders (admin only).
- **Response:**

```
json  
Copy code  
[  
  {  
    "id": "order_id_1",  
    "customer": "John Doe",  
    "items": [  
      {  
        "product": "Wireless Earbuds",  
        "quantity": 2,  
        "price": 59.99  
      }  
    ]  
  }  
]
```

```
    ],  
    "totalAmount": 119.98,  
    "status": "Shipped"  
  },  
  {  
    "id": "order_id_2",  
    "customer": "Jane Smith",  
    "items": [  
      {  
        "product": "Gaming Laptop",  
        "quantity": 1,  
        "price": 1199.99  
      }  
    ],  
    "totalAmount": 1199.99,  
    "status": "Processing"  
  }  
]
```

8. Authentication

Authentication Strategy

The authentication system for the freelancing platform is based on JWT (JSON Web Tokens). The platform employs a token-based authentication system to manage user sessions.

- User Registration:

- When a user registers (either a freelancer or client), the system hashes the password using Bcrypt.

- The user details are saved in the MongoDB database, and a JWT token is returned for authentication.

- Login:

- Upon login, the system checks the credentials against the database.
 - If the credentials are valid, the user receives a JWT token, which must be included in the Authorization header of subsequent requests.

- Authorization:

- The JWT token is used to authenticate and authorize users for protected routes.
 - The backend verifies the token using a secret key stored in the environment variables (JWT_SECRET).

- Session Management:

- The frontend stores the JWT token in the browser's localStorage or sessionStorage, which is sent with every request to the backend to secure access to protected routes.

By utilizing this token-based authentication strategy, the platform ensures that only authorized users can access certain features and manage their account securely.

9. User Interface

The E-Commerce website is designed with a modern and intuitive interface, ensuring a seamless shopping experience for users. Below are the key components of the user interface:

Homepage

- **Features:**
 - Displays a banner with promotions, discounts, or featured products.
 - Categories section to help users explore different product groups.
 - Highlights best-selling and newly launched products.
-

Product Listings Page

- **Features:**
 - Displays a grid or list of products with images, prices, and short descriptions.
 - Includes filters for categories, price range, brands, and ratings.
 - Search functionality to quickly find desired products.
-

Product Details Page

- **Features:**
 - Detailed information about a specific product, including high-quality images, price, description, specifications, and reviews.
 - Add to Cart and Buy Now buttons for quick action.
 - Recommendations for similar or related products.
-

Shopping Cart

- **Features:**
 - Displays items added by the user, including product name, quantity, price, and total cost.

- Option to update quantities or remove items.
 - Provides a clear CTA to proceed to checkout.
-

Checkout Page

- **Features:**
 - Allows users to enter shipping and billing information.
 - Provides a summary of items in the cart and the total cost, including taxes and shipping charges.
 - Payment gateway integration for completing the order securely.
-

User Dashboard

- **Features:**
 - **For Customers:**
 - View past orders with details such as order status, items purchased, and total cost.
 - Manage saved addresses and payment methods.
 - Option to update account details (e.g., name, email, and password).
 - **For Admins:**
 - Manage products (add, edit, delete) and stock levels.
 - View and process customer orders.
-

Profile Management

- **Features:**
 - Customers can update their profile information, including name, email, phone number, and saved addresses.
 - Admins can manage their account details and access admin-specific tools.
-

Screenshots

1. **Homepage of the E-Commerce Website:**
 - A visually engaging homepage showcasing the latest promotions, featured products, and quick navigation links to categories.
2. **Product Listings Page:**
 - Displays a grid of products with options to filter and sort by price, category, or rating.
3. **Shopping Cart:**
 - A user-friendly interface showing all cart items, allowing modifications, and providing a total cost summary.
4. **Checkout Page:**
 - A clean form to collect shipping and payment details, with a summary of the order.
5. **User Dashboard:**
 - Displays past orders and provides account management options for customers.

10. Testing

To ensure a high-quality, secure, and smooth shopping experience, a comprehensive testing strategy was implemented for the E-Commerce MERN stack website. Below are the key testing approaches used:

Unit Testing

Frontend (React)

- **Tools Used:** Jest, React Testing Library
- **Focus Areas:**

- Components such as ProductDisplay, CartItems, and CheckoutForm were tested to ensure they render correctly and handle user interactions as expected.
- Tests covered:
 - Button clicks (e.g., "Add to Cart").
 - Proper rendering of product information (title, price, ratings).
 - Input validation for checkout forms.
- Example Test Case:
 - Verify that adding a product to the cart updates the cart count.

Backend (Node.js/Express)

- **Tools Used:** Mocha, Chai
 - **Focus Areas:**
 - Tested individual API endpoints to ensure they handle requests and responses correctly.
 - Key API tests:
 - **Product Retrieval:** Ensures products are fetched correctly from the database.
 - **User Authentication:** Validates token generation and user session handling.
 - **Order Placement:** Tests that orders are stored properly in the database and trigger the appropriate success messages.
-

API Testing

- **Tools Used:** Postman, Newman
- **Focus Areas:**
 - Tested all key REST API endpoints for functionality, including:
 - **GET /api/products:** Ensures all products are retrieved with correct details.
 - **POST /api/cart:** Verifies products can be added to the cart.
 - **POST /api/orders:** Confirms successful order placement.
 - **POST /api/users/register and /login:** Ensures valid registration and login functionality.

- Checked response codes, data validation, and error handling for invalid requests.
 - **Example Scenarios:**
 - Placing an order without authentication should return a 401 error.
 - Adding out-of-stock items to the cart should return an appropriate error.
-

Integration Testing

- **Focus Areas:**
 - Tested the integration of the frontend with backend APIs to ensure seamless data flow.
 - Mock Data Testing:
 - Simulated various scenarios, such as:
 - Users browsing products, adding items to the cart, and placing orders.
 - Admins adding or updating products.
 - Tested end-to-end processes like:
 - A customer searching for a product, adding it to the cart, and successfully completing a purchase.
 - Admins uploading new products and viewing them on the frontend.
-

Performance Testing

- **Tools Used:** Lighthouse, JMeter
- **Focus Areas:**
 - **Page Load Speed:** Ensured that the homepage, product listing, and checkout pages load within optimal timeframes.
 - **API Performance:**
 - Measured response times for key API endpoints under different loads.
 - Simulated multiple users browsing products and placing orders simultaneously to check server scalability.

- **Stress Testing:** Tested the website's behavior under high traffic, including scenarios like flash sales or heavy seasonal traffic.
-

User Acceptance Testing (UAT)

- Conducted manual testing with real users to simulate real-world scenarios.
 - Gathered feedback from testers for improvements in:
 - Navigability of the UI.
 - Simplicity of checkout and payment processes.
 - Responsiveness across devices (mobile, tablet, desktop).
-

Security Testing

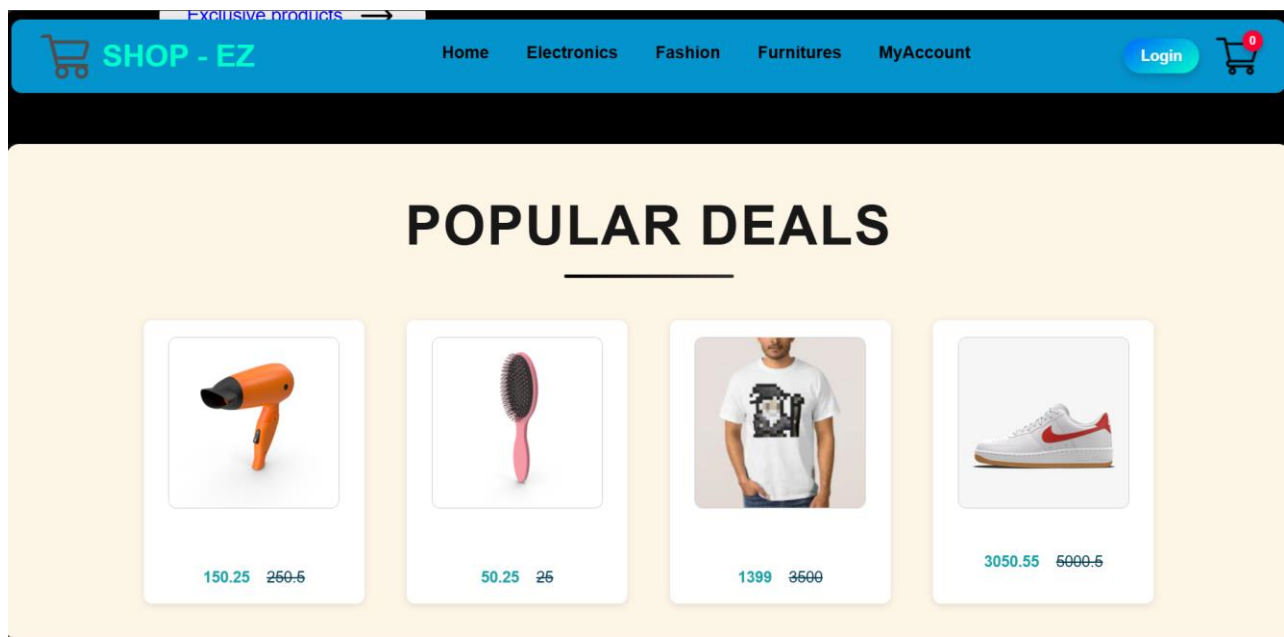
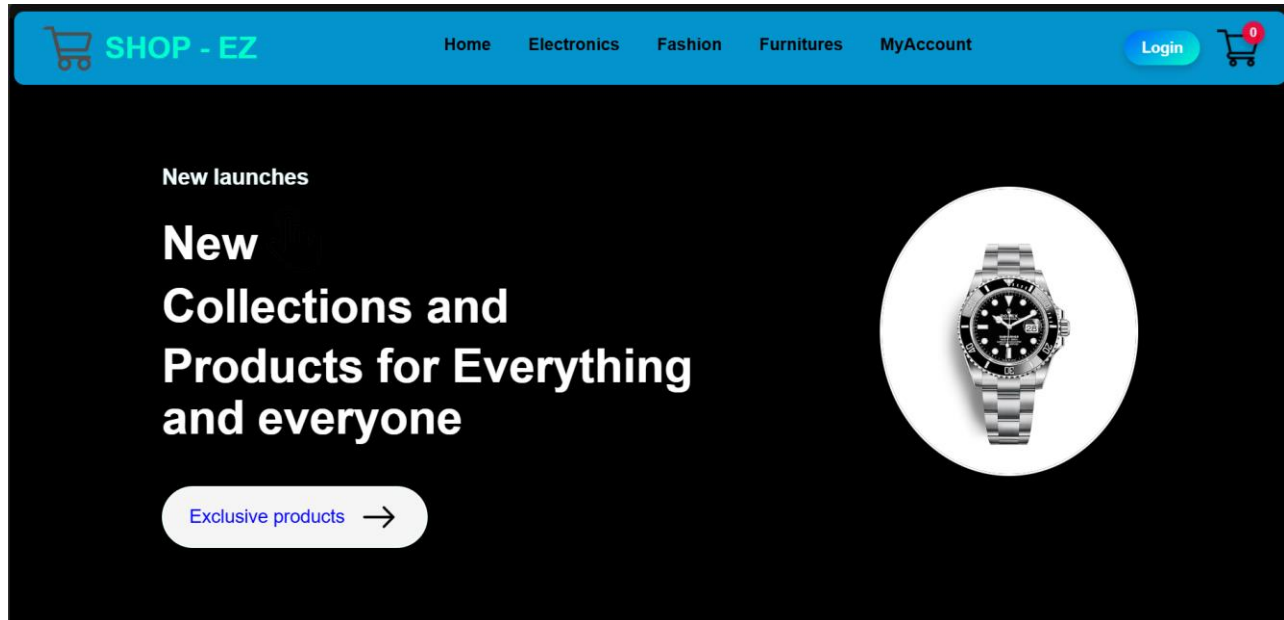
- **Focus Areas:**
 - Validated secure handling of sensitive user data (e.g., passwords, payment details).
 - Tested for common vulnerabilities such as:
 - SQL injection (database security).
 - XSS (cross-site scripting) attacks.
 - CSRF (cross-site request forgery).
 - Ensured HTTPS and encryption of sensitive API endpoints.
-

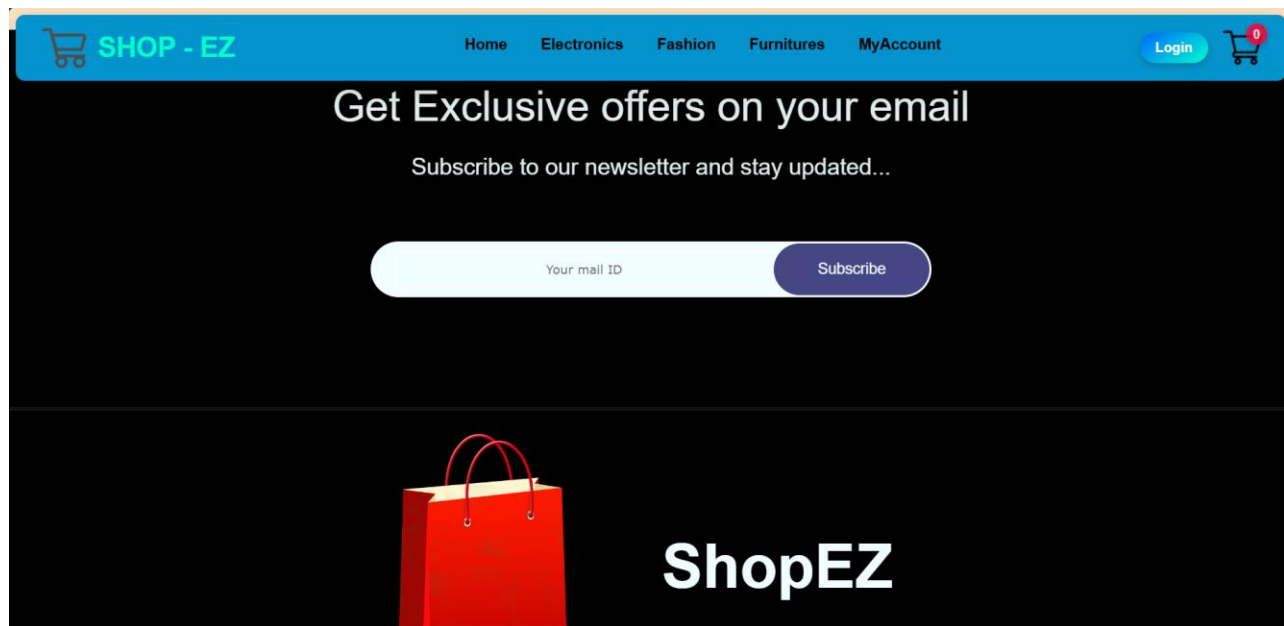
Regression Testing

- **Tools Used:** Selenium (for frontend automation)
- Performed automated regression tests to ensure that new updates do not break existing functionality.
- Automated workflows for:
 - Login and registration.
 - Adding items to the cart and completing checkout.
 - Product search and filter functionalities.

11. Screenshots

Screenshots:







SHOP - EZ

[Home](#)

[Electronics](#)

[Fashion](#)

[Furnitures](#)

[MyAccount](#)

[Login](#)



SHOP - EZ

[Home](#)

[Electronics](#)

[Fashion](#)

[Furnitures](#)

[MyAccount](#)

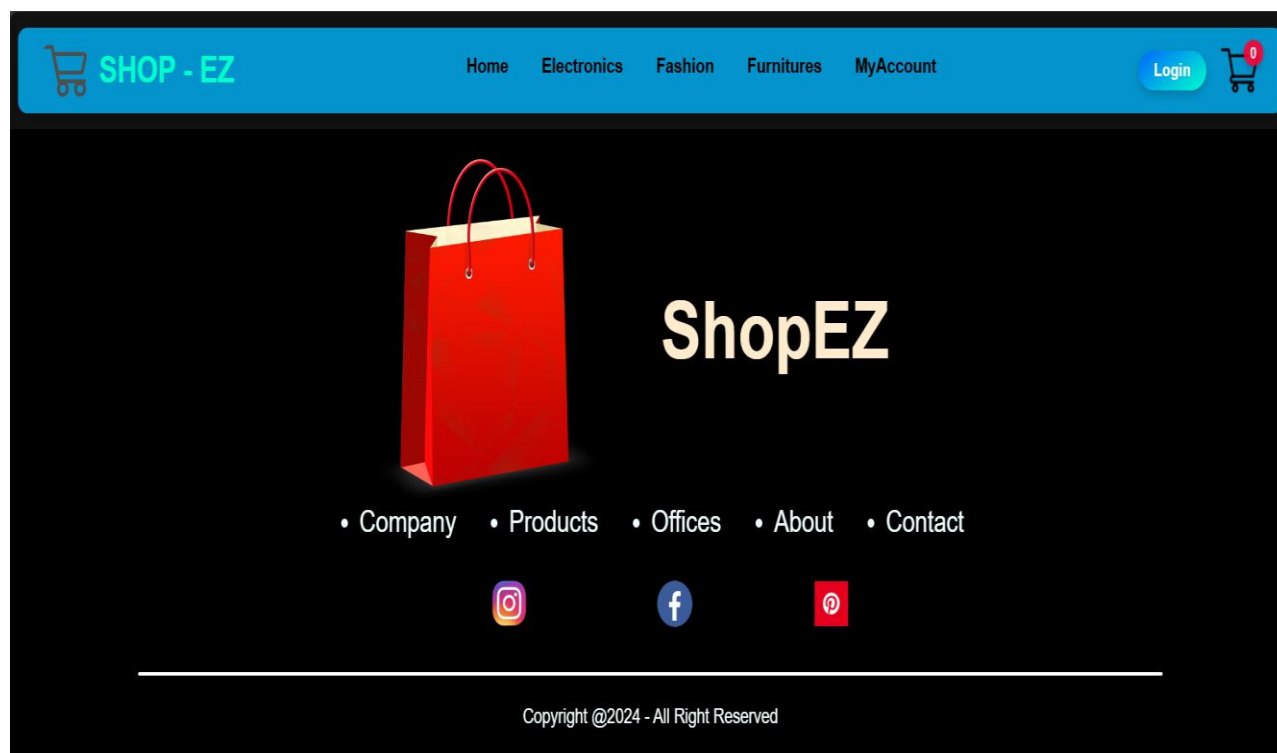
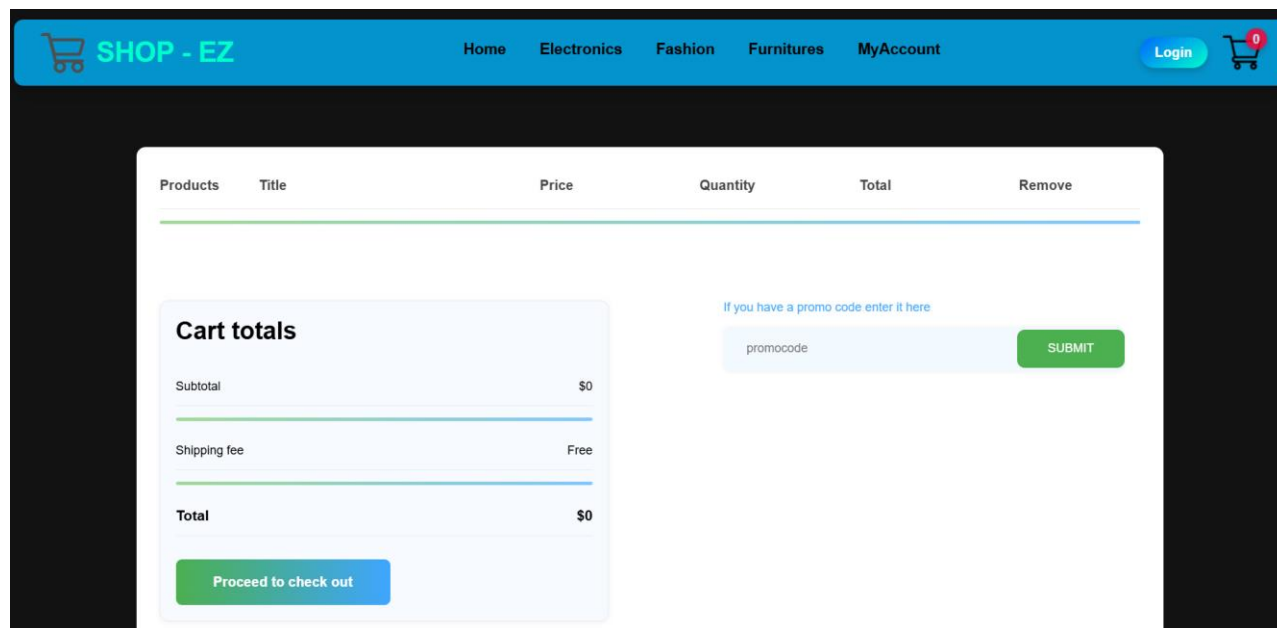
[Login](#)




s.madhan2364@gmail.com


Submit

Create an account? [Sign up here](#)



 **SHOP - EZ**

[Home](#) [Electronics](#) [Fashion](#) [Furnitures](#) [MyAccount](#)

[Login](#) 




HURRY! FOR @MIDNIGHTSALES


Only after midnight

For 1 Hour

[CHECK NOW](#)

 **SHOP - EZ**

[Home](#) [Electronics](#) [Fashion](#) [Furnitures](#) [MyAccount](#)

[Login](#) 

[HOME](#) [SHOP](#) [Electronics](#) [Microwave Oven](#)

Microwave Oven

★★★★★ (122)

~~\$430~~ **\$100**

Select size

[Add to cart](#)

Category: Women, tshirt, crop top

Tags: Modern, tshirt

Conclusion:

In conclusion, the development of this E-Commerce platform using the MERN stack highlights the robust capabilities of modern web technologies. By combining React, Node.js, Express.js, and MongoDB, the platform delivers a seamless, responsive, and scalable shopping experience for users. The integration of essential features such as product browsing, secure authentication, shopping cart management, order tracking, and an intuitive admin dashboard ensures an efficient and user-friendly experience for both customers and administrators. The scalable architecture of the MERN stack allows the platform to handle growing traffic and product catalogues, ensuring reliability and performance. This project demonstrates how the MERN stack can be effectively utilized to build dynamic, feature-rich E-Commerce solutions tailored to meet user needs.

Result:

The app has been successfully developed using the required software and technologies.