INTRODUCTION

Project Title: ShopEZ: E-commerce Application
Team Members:

Jaya Aishwaryaa J (Project Coordinator),

Nandhini J (Frontend developer),

Roopini P V (Backend developer),

Subhashinee G K (Backend Manager)

PROJECT OVERVIEW

Purpose

The *ShopEZ* project is a comprehensive e-commerce platform designed to enhance online shopping by making it effortless and enjoyable for users while supporting efficient management for sellers. Key goals of ShopEZ include:

- **Effortless Product Discovery:** Users can quickly find items that match their interests with intuitive navigation, categories, and filtering options, as well as personalized recommendations based on their browsing history.
- **Seamless Checkout Process:** A secure and streamlined checkout enables users to place orders with ease, ensuring they have a positive experience from start to finish. ● **Personalized Shopping Experience:** The application provides tailored product recommendations, enhancing user engagement and increasing the likelihood of finding desired items.
- **Efficient Order Management for Sellers:** A dedicated seller dashboard offers tools for order processing and inventory tracking, supporting sellers in managing their business effectively.
- **Insightful Analytics:** By offering analytics, sellers can gain insights into customer preferences and sales trends, empowering them to make data-driven decisions for growth.

Features

The *ShopEZ* e-commerce platform is packed with features and functionalities designed to create a smooth, secure, and personalized online shopping experience. Here are the key features:

For Shoppers

- **Effortless Product Discovery:** Intuitive navigation and filtering options make it easy to browse through categories and find specific products. Users can quickly locate items that match their needs, preferences, and budget.
- **Personalized Recommendations:** ShopEZ uses user behavior and preferences to provide tailored product recommendations, enhancing engagement and helping users discover items they may not have considered.
- **Detailed Product Descriptions & Reviews:** Each product page provides comprehensive descriptions, customer reviews, and ratings, helping users make informed decisions.
- **Seamless Checkout Process:** A streamlined and secure checkout experience allows users to complete purchases with minimal steps, including multiple payment options

and easy address entry.
- **Instant Order Confirmation:** Shoppers receive immediate confirmation and status updates on their orders, giving them confidence and peace of mind.
- **User Profile Management:** Users can manage their personal information, track order history, and save items to a wishlist for future purchases.

For Sellers

- **Efficient Order Management:** ShopEZ provides a robust seller dashboard where sellers can manage and track orders, update stock levels, and streamline fulfillment.
- **Inventory Management:** Sellers can easily add, edit, and organize their products within the platform, with options to adjust pricing and availability as needed.
- **Insightful Analytics:** The seller dashboard includes analytics tools, enabling sellers to monitor sales trends, view customer behavior, and make informed decisions for growth.
- **Customer Feedback & Ratings:** Sellers receive customer reviews and ratings for their products, helping them gauge customer satisfaction and make improvements.

ARCHITECTURE

<u>Frontend</u>

The frontend is designed with **React.js**, providing a modular and responsive user interface. Key components and architecture details include:

- Component-Based Design:
  - The frontend consists of reusable React components such as ProductList, ProductDetail, Cart, Checkout, UserProfile, and SellerDashboard.
  - Each component is responsible for a specific UI element and maintains its own state, ensuring a clean and scalable structure.
- State Management:
  - Redux (or Context API) is used for state management, handling global states like user authentication, product data, and shopping cart contents.
  - Middleware such as Redux Thunk or Redux Saga is used for managing asynchronous actions like API calls.
- Routing:
  - React Router is used for client-side routing, enabling seamless navigation between different sections of the app (e.g., Home, Product Details, Checkout, Profile).
  - Protected Routes are implemented to restrict access to certain pages (e.g., Admin Dashboard) based on user roles.
- API Integration:
  - Axios or Fetch API is used for HTTP requests to the backend, facilitating data fetching for products, orders, and user profiles.
  - Loading states and error handling are implemented to ensure responsive feedback to users.
- Responsive Design:
  - CSS frameworks (like Bootstrap or Tailwind) or CSS-in-JS libraries (like Styled Components) are used for responsive design, ensuring compatibility across devices.

<u>Backend</u>

The backend uses **Node.js** and **Express.js** to create a RESTful API and handle the business logic. Key components include:

- RESTful API Endpoints:

    - Express is used to create a modular and RESTful API, with endpoints for managing resources like users, products, orders, and reviews.
    - Routes are organized by functionality and grouped into modules, such as userRoutes, productRoutes, and orderRoutes.
- Middleware and Authentication:
    - Middleware handles cross-cutting concerns, such as user authentication, input validation, error handling, and logging.
    - JWT (JSON Web Token) is used for secure user authentication and authorization, allowing users to log in and access protected routes.
- Business Logic Services:
    - Business logic is separated into service modules (e.g., OrderService, ProductService, UserService) to keep the codebase modular and maintainable. ○ Each service manages specific functionality, such as order processing, inventory updates, and user role verification.
- Error Handling and Logging:
    - Custom error-handling middleware captures errors and provides informative responses.
    - Logging frameworks (like Winston) are used for tracking application events and assisting in debugging.
- Security Features:
    - Input validation using libraries like Joi or Express-validator prevents SQL injection and cross-site scripting attacks.
    - Helmet and CORS middleware enforce security headers and handle cross-origin resource sharing policies.

Database

The database uses **MongoDB**, with **Mongoose** as the Object Data Modeling (ODM) library to facilitate data interactions. Key schema and interaction details include:

a.Collections and Schemas:

Users Collection:

- Fields: userId, name, email, passwordHash, address, role, orderHistory, wishlist.
- Stores user data, including roles to differentiate between buyer, seller, and admin users.

Products Collection:

- Fields: productId, sellerId, name, description, price, category, inventoryCount, rating, reviews.
- Stores product information with sellerId for tracking the seller and inventoryCount for stock management.

Orders Collection:

- Fields: orderId, userId, productIds, totalPrice, status, shippingAddress, createdAt.
- Contains order data, linking each order to the user who placed it and the products involved.

Reviews Collection:

- Fields: reviewId, productId, userId, rating, comment, createdAt.
- Allows users to leave product reviews, linked by userId and productId.

b.Database Interactions:

- Mongoose models define the schema structure for each collection, supporting CRUD operations and data validation.
- Indexed fields (e.g., productId, category, userId) optimize query performance, especially for frequent searches like product lookups and order history.

c.Data Relationships:

- Relationships are established through embedded documents or references (Object IDs), improving data retrieval speed.
- Commonly accessed fields, such as user and product details within an order, are embedded when appropriate for faster performance.

SETUP INSTRUCTIONS

<u>Prerequisites</u>.

To build and run the *ShopEZ* app, make sure the following software dependencies are installed on your development machine:

- Node.js and npm:
  - Node.js is required for running the server-side code.
  - npm (Node Package Manager), bundled with Node.js, is used to manage packages.
- MongoDB:
  - MongoDB is used as the primary database for storing user, product, and order data.
  - You can install MongoDB locally or use a cloud-based MongoDB service (e.g., MongoDB Atlas).
- Express.js:
  - Express.js is a lightweight web application framework for Node.js, used to create API routes and handle server-side requests.
  - Install via npm: npm install express
- React.js:
  - React.js is used for building the frontend user interface.
- Mongoose:
  - Mongoose is an ODM (Object Data Modeling) library that helps with MongoDB interactions in a Node.js environment.
  - Install via npm: npm install mongoose
- Git:
  - Git is required for version control, allowing you to clone the repository and track changes.
- Development Environment:
  - A code editor or IDE is recommended for development. Popular choices include:
    - **Visual Studio Code**
    - **Sublime Text**
    - **WebStorm**

<u>Installation</u>

To set up the *ShopEZ* app on your local machine, follow these steps:

a.Clone the Repository:

Open your terminal or command prompt.

Navigate to the directory where you want to store the project.

Clone the repository:
git clone <repository-url>

Replace <repository-url> with the actual URL of the *ShopEZ* GitHub repository.

b.Navigate to the Project Directory:
cd ShopEZ-e-commerce-App-MERN

c.Install Dependencies:

Ensure you're in the project's root directory.

Run the following command to install all necessary dependencies:
npm install

This will install both frontend and backend dependencies if they are specified in a single package.json file.
If frontend and backend are in separate folders (e.g., /client for React and /server for Node.js), navigate into each folder and run npm install:
# For backend

```
    cd server

    npm install

    # For frontend

    cd ../client

    npm install
```

d.Set Up Environment Variables:

Create a .env file in the root directory of the backend project (e.g., server).

Add the following environment variables to the .env file:
env

```
PORT=5000

    MONGO_URI=<your-mongodb-connection-string>

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

Replace <your-mongodb-connection-string> with the MongoDB URI for your database (either local or cloud-based).

Replace <your-jwt-secret> with a secure string for JSON Web Token (JWT) authentication.

You may need additional variables depending on your app configuration, such as payment gateway keys or email service credentials.

e.Start the Development Server:

For a combined project:

```
npm run dev
```

Start the backend server:

```
cd server
```

```
npm run start
```

Start the frontend development server:

```
cd ../client
```

```
npm run start
```

f.Access the Application:

By default, the app should be accessible at http://localhost:3000 for the frontend.

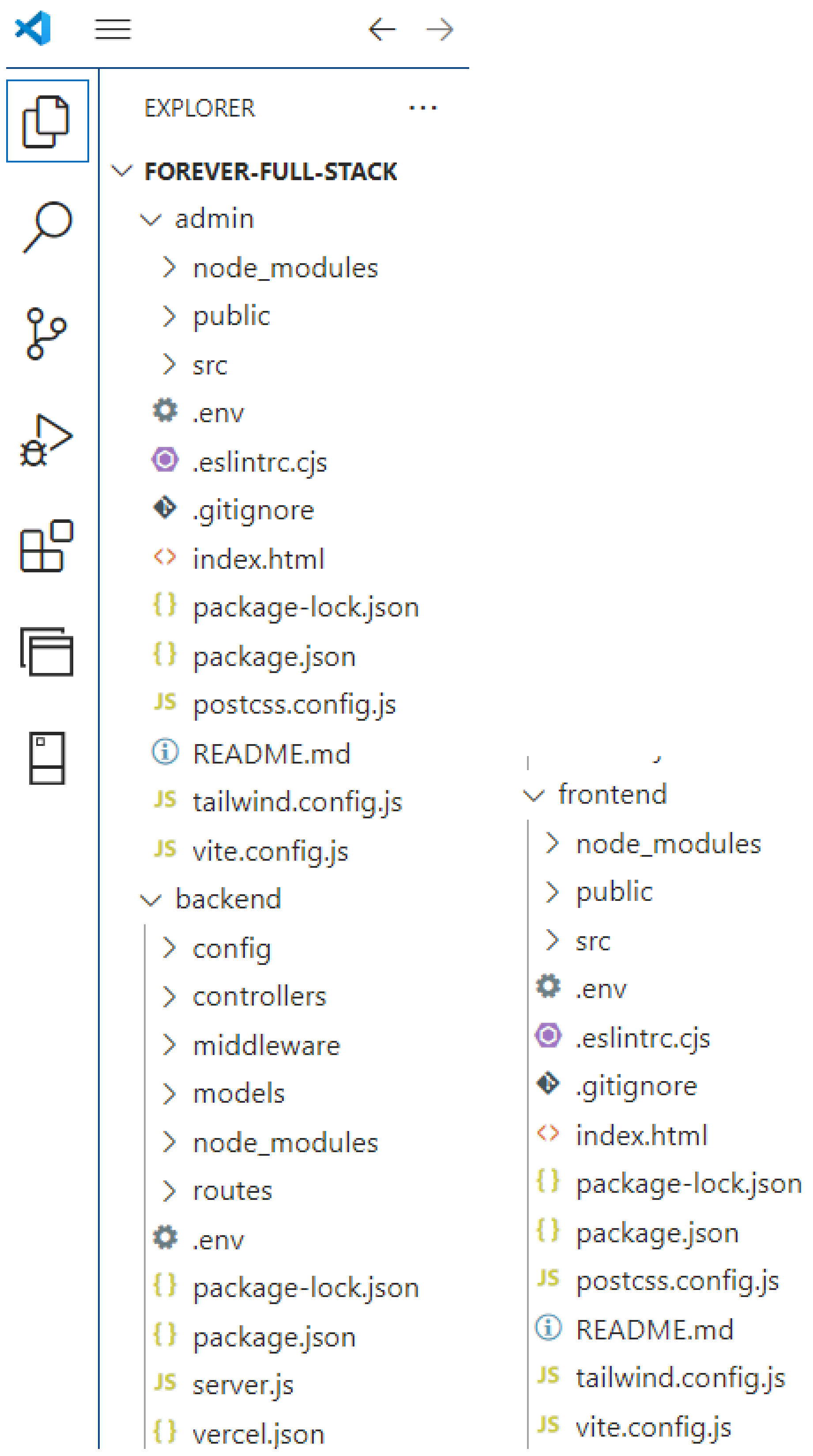
The backend server, if separate, should run on http://localhost:5000 or the port specified in the .env file.

g.Verification:

Open a web browser and go to http://localhost:3000.

You should see the homepage of the *ShopEZ* app, indicating successful setup.

FOLDER STRUCTURE



```
FOREVER-FULL-STACK
  admin
    node_modules
    public
    src
    .env
    .eslintrc.cjs
    .gitignore
    index.html
    package-lock.json
    package.json
    postcss.config.js
    README.md
    tailwind.config.js
    vite.config.js
  backend
    config
    controllers
    middleware
    models
    node_modules
    routes
    .env
    package-lock.json
    package.json
    server.js
    vercel.json
  frontend
    node_modules
    public
    src
    .env
    .eslintrc.cjs
    .gitignore
    index.html
    package-lock.json
    package.json
    postcss.config.js
    README.md
    tailwind.config.js
    vite.config.js
```

RUNNING THE APPLICATION

Frontend

Navigate to the frontend directory (typically named `client`):

```
cd client
```

Install dependencies:

```
npm install
```

Start the React development server:

```
npm start
```

The frontend will run by default at:

```
http://localhost:5173
```

Backend

Navigate to the backend directory (typically named `server`):

```
cd server
```

Install dependencies:

```
npm install
```

Start the Node.js server:

```
npm start
```

The backend will run by default at:

```
http://localhost:5174
```

(Ensure the port is correctly set in the `.env` file.)

API DOCUMENTATION

Base URL

`http://localhost:5000/api`

Authentication

1. User Registration

   - POST `/auth/register`

   - Registers a new user.

   - Body: `name`, `email`, `password`.

   - Response: Success message, user details.

2. User Login

   - POST `/auth/login`

   - Logs in a user.
   - Body: `email`, `password`.

   - Response: JWT token, user info.

3. Admin Login

   - POST `/auth/admin-login`

   - Logs in an admin.

   - Body: `email`, `password`.

   - Response: Admin JWT token.

Products

1. Get All Products

- GET `/products`

- Fetches all products.

2. Get Product Details

- GET `/products/:id`

- Fetches product by ID.

3. Add Product (Admin)

- POST `/products`

- Adds a product.

- Body: `name`, `price`, `description`, `category`.

4. Delete Product (Admin)

- DELETE `/products/:id`

- Deletes a product by ID.

Orders

1. Place an Order

- POST `/orders`

- Places an order.

- Body: `products`, `quantity`.

2. View User Orders

- GET `/orders/user`

- Fetches orders of the logged-in user.

3. View All Orders (Admin)

- GET `/orders`

- Fetches all orders.

AUTHENTICATION

- User Authentication:
  - Implemented via **JWT (JSON Web Tokens)**, which are issued upon successful login.
  - The user provides their credentials (email and password) to the /api/auth/login

endpoint.
  - Upon verification, the server generates a JWT token that is sent back to the client. This token must be included in the headers of subsequent requests (e.g., Authorization: Bearer <token>).
- Password Security:
  - User passwords are hashed using secure hashing algorithms (e.g., bcrypt) before being stored in the database, ensuring that raw passwords are never saved.
- Token Expiry:
  - JWT tokens include an expiry time, after which users must log in again to obtain a new token.

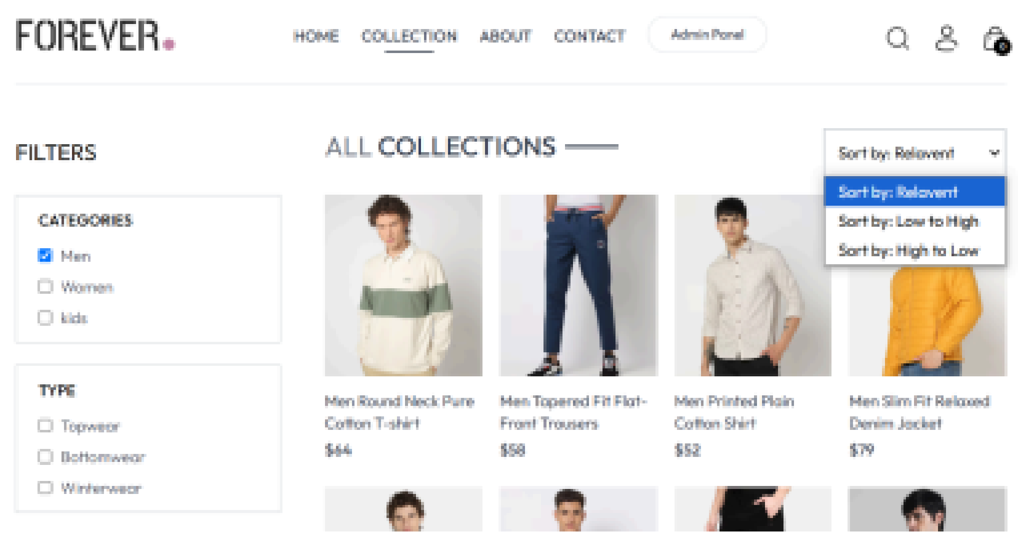
Authorization

- Role-Based Access:
  - The project implements role-based authorization, which uses the user's role (e.g., admin, customer) encoded in the JWT to restrict access to specific endpoints.
  - For example:
    - Admin users may access routes for product management (/api/products for creation or deletion).
    - Regular users can access endpoints like viewing products or placing orders.
- Middleware:
  - Authorization is enforced using middleware functions:
    - Middleware verifies the presence and validity of the JWT.
    - Decodes the token to extract user information, such as role and ID, and attaches it to the request object for further use in the application.
    - Blocks access to unauthorized routes if the user lacks the required role or permissions.

Flow Example

1. User logs in via /api/auth/login.
2. Receives a JWT token and includes it in the headers for protected endpoints. 3. Middleware validates the token, checks user roles, and grants or denies access based on the route's requirements.

USER INTERFACE

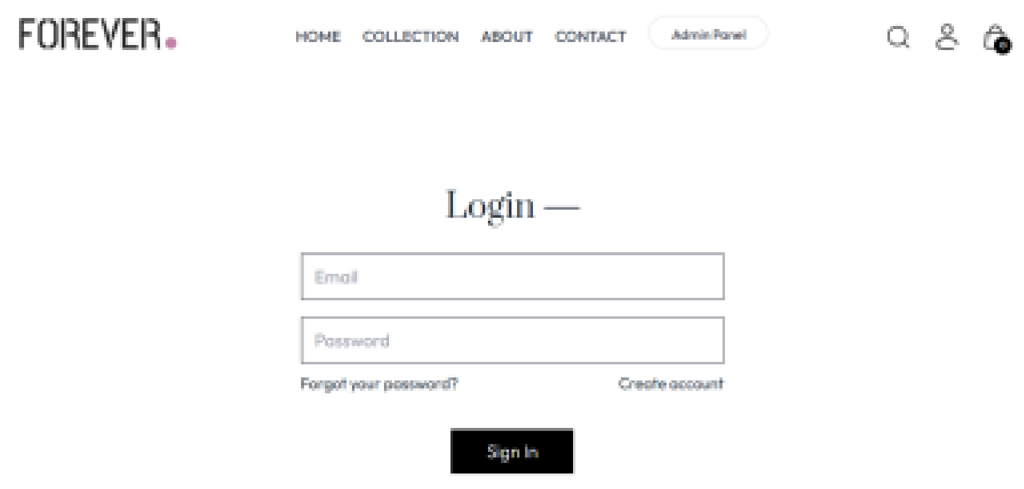


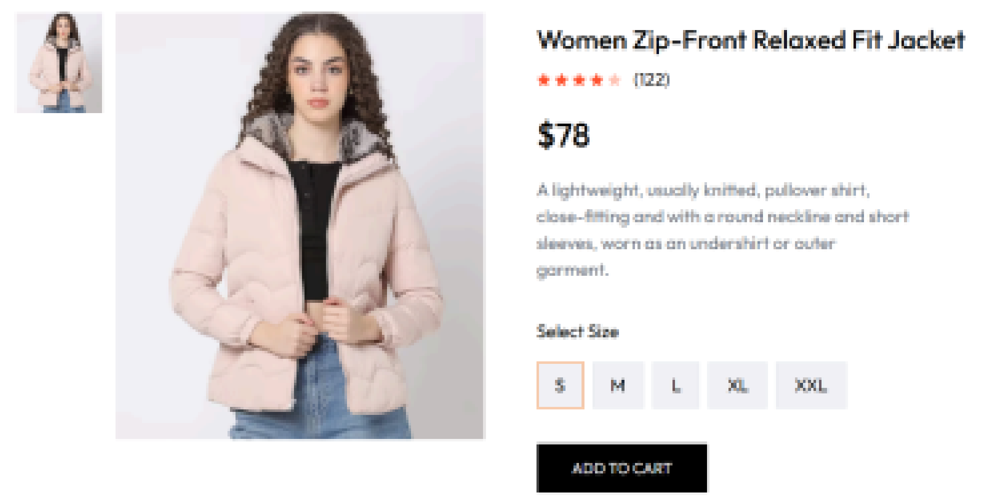
Men Round Neck Pure Cotton T-shirt
$64

Men Tapered Fit Flat-Front Trousers
$58

Men Printed Plain Cotton Shirt
$52

Men Slim Fit Relaxed Denim Jacket
$79





Women Zip-Front Relaxed Fit Jacket

(122)

$78

A lightweight, usually knitted, pullover shirt, close-fitting and with a round neckline and short sleeves, worn as an undershirt or outer garment.

TESTING

a.Unit Testing

○ **Objective**: Validate individual components and modules in isolation. ○ **Frontend**: Test React components, especially those involving dynamic states or API interactions.

○ **Backend**: Validate API routes, controllers, and utility functions.

○ **Tools**:

■ **Jest**: For testing React components and JavaScript code.

■ **Mocha/Chai**: For backend testing in Node.js.

b.Integration Testing

- **Objective**: Ensure the proper interaction between frontend and backend components.
- **Scope**: Test data flow from the frontend to the backend and database.
- **Tools**:
  - **Supertest**: For HTTP endpoint testing.
  - **Postman/Newman**: For API testing.

c.End-to-End (E2E) Testing

- **Objective**: Simulate user workflows, such as browsing products, adding to cart, and completing checkout.
- **Tools**:
  - **Cypress**: Ideal for automating and testing complete user flows.

d.Performance Testing

- **Objective**: Assess application speed, API response time, and scalability under load.
- **Tools**:
  - **JMeter**: For backend load testing.
  - **Lighthouse**: For evaluating frontend performance.

e.Security Testing

- **Objective**: Identify vulnerabilities in user authentication and sensitive data handling.
- **Tools**:
  - **OWASP ZAP**: For scanning and identifying security issues.

f.Regression Testing

- **Objective**: Ensure new updates do not break existing functionality.
- **Tools**:
  - Use automated test suites built with **Jest** or **Selenium** for comprehensive coverage.

Tools and Practices

a.Continuous Integration (CI):

- Tools like **GitHub Actions** could be configured to run automated tests for every code commit.

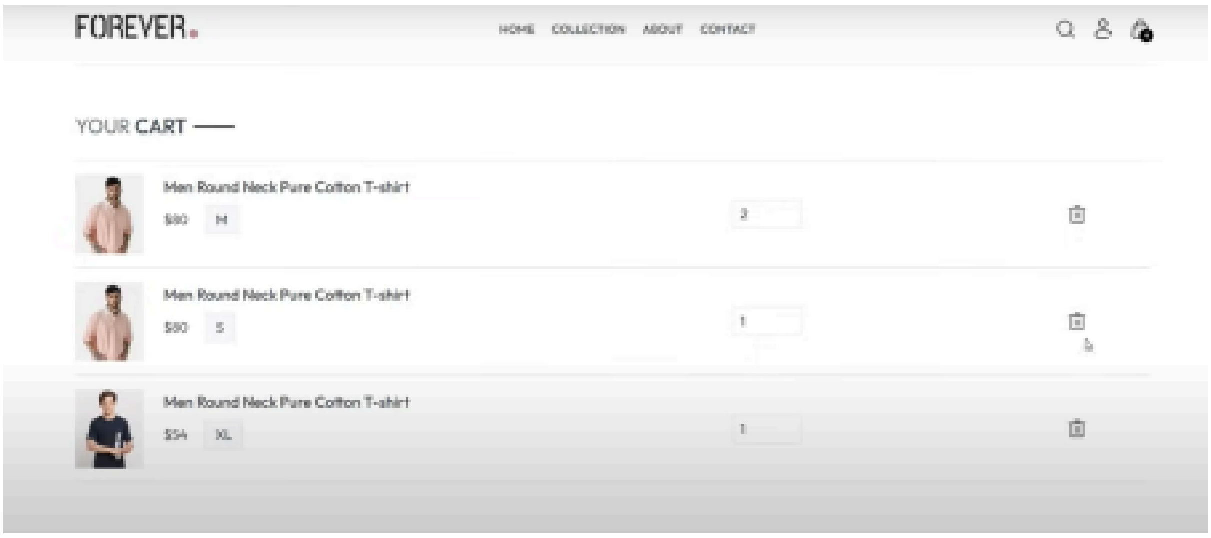
b.Mocking Frameworks:

- Use libraries like **Mock Service Worker (MSW)** to test frontend components that rely on API data.

c.Code Coverage:

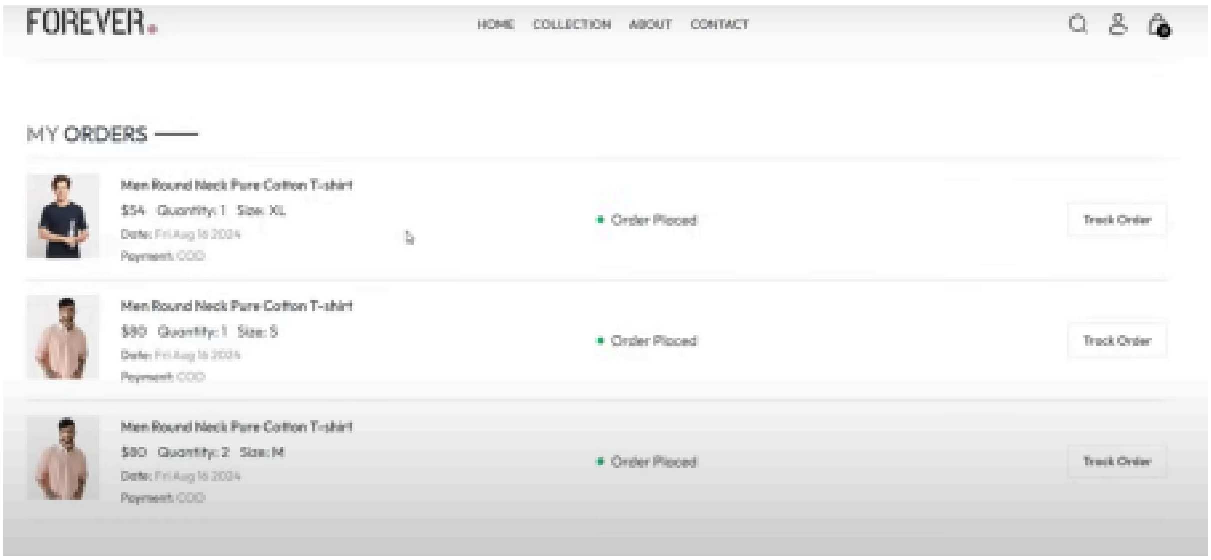
- Tools like **Istanbul** (integrated with Jest) provide insights into test coverage.

SCREENSHOTS OR DEMO



YOUR CART

Men Round Neck Pure Cotton T-shirt	$80	M	2
Men Round Neck Pure Cotton T-shirt	$80	S	1
Men Round Neck Pure Cotton T-shirt	$54	XL	1

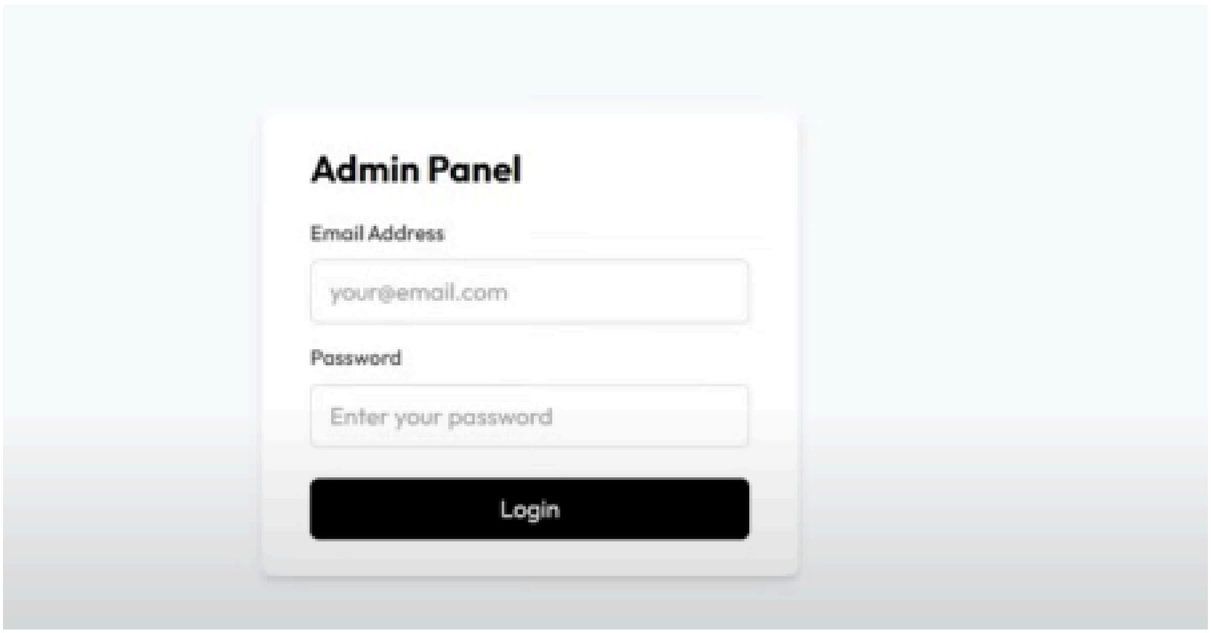


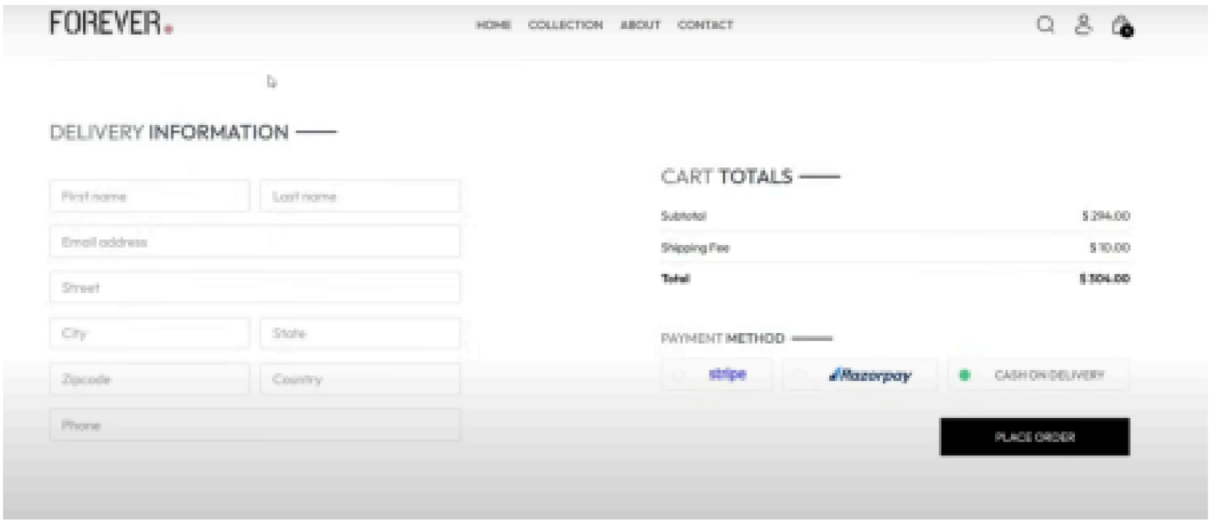
MY ORDERS

Men Round Neck Pure Cotton T-shirt
$54 Quantity: 1 Size: XL
Date: Fri Aug 16 2024
Payment: COD
Order Placed
Track Order

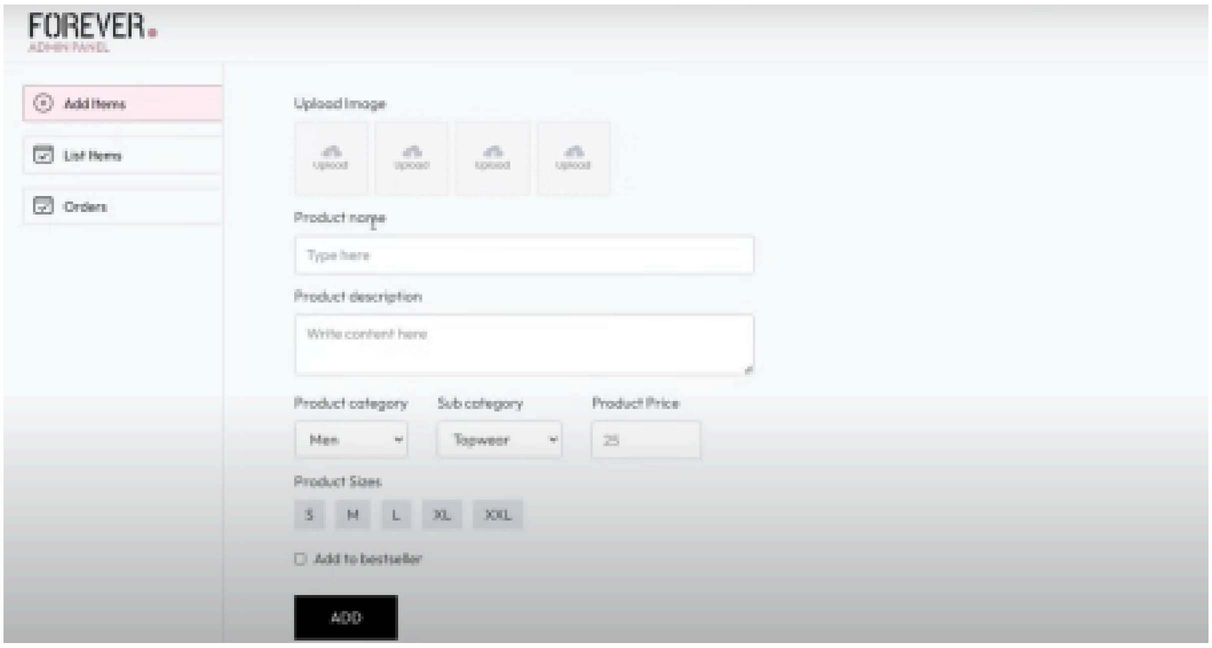
Men Round Neck Pure Cotton T-shirt
$80 Quantity: 1 Size: S
Date: Fri Aug 16 2024
Payment: COD
Order Placed
Track Order

Men Round Neck Pure Cotton T-shirt
$80 Quantity: 2 Size: M
Date: Fri Aug 16 2024
Payment: COD
Order Placed
Track Order





Subtotal	$ 294.00
Shipping Fee	$ 10.00
Total	**$ 304.00**



KNOWN ISSUES

a. Slow Page Load Times on Product Pages

- **Issue**: Some users experience delays when navigating to product pages, particularly when filtering or sorting items.
- **Cause**: This is likely due to inefficient queries to the backend, especially with large datasets, or lack of caching mechanisms.
- **Temporary Workaround**: Reload the page if loading seems to stall. Developers may consider adding pagination or query optimization in the backend.

b. Cart Items Not Updating Properly

- **Issue**: Occasionally, the cart does not update when items are added or removed, requiring users to refresh the page to see the latest cart state.
- **Cause**: This appears to be related to inconsistent state management in the frontend, likely due to asynchronous issues with Redux.
- **Temporary Workaround**: Users can refresh the page to see the correct cart contents. Developers may need to review the Redux flow and asynchronous actions.

c. Payment Gateway Timeout

- **Issue**: Some users encounter a timeout error during checkout when using certain payment methods.
- **Cause**: The server may not be able to handle multiple payment requests

simultaneously, or there may be a misconfiguration in payment gateway settings. ● **Temporary Workaround**: Retry the payment after a few minutes. Developers should consider increasing server timeout settings and reviewing payment API configurations.

d. Inaccurate Stock Availability

- **Issue**: Occasionally, products marked as "In Stock" are not actually available, leading to order cancellations after checkout.
- **Cause**: Stock levels are not updating in real-time across multiple sessions, potentially due to lack of synchronization with the database.
- **Temporary Workaround**: Users can contact support to confirm stock status before ordering. Developers may want to add more frequent stock checks or database locks during high-traffic periods.

e. Poor Mobile Responsiveness on Certain Pages

- **Issue**: Some pages, particularly the checkout and product pages, may appear distorted or not fully responsive on mobile devices.
- **Cause**: CSS media queries and layout adjustments have not been fully optimized for various screen sizes.
- **Temporary Workaround**: Rotate the device or use a desktop to access the site. Developers should prioritize CSS improvements and testing on a wider range of mobile devices.

f. Search Function Limitations

- **Issue**: The search function returns limited results or sometimes irrelevant products, frustrating users attempting to locate specific items.
- **Cause**: The search algorithm currently lacks advanced filters and keyword matching, leading to suboptimal search results.
- **Temporary Workaround**: Use precise keywords or browse through categories manually. Developers should enhance search algorithms to improve relevancy.

g. Login Session Expiration Without Notification

- **Issue**: Users are logged out after a session expiration but are not notified, leading to potential data loss if they were in the middle of an action.
- **Cause**: The session management lacks notification or warning mechanisms for session timeouts.
- **Temporary Workaround**: Refresh the page if unexpected logout occurs and re-login. Developers should implement a session timeout warning.

h. Inconsistent Display of Order History

- **Issue**: Users report that some orders do not appear in their order history or appear multiple times.
- **Cause**: Possible race conditions or issues with how the database fetches and renders order history on the frontend.
- **Temporary Workaround**: Refresh the order history page if discrepancies are noticed. Developers may need to review database queries and consider adding caching for consistency.

i. Unreliable Notification System

- **Issue**: Notifications for order updates, promotions, or discounts do not consistently reach users.
- **Cause**: Notifications are sometimes blocked by browsers, or there may be issues with

the push notification setup.

- **Temporary Workaround**: Manually check the app for order updates. Developers should review notification settings and add redundancy to ensure notifications are delivered.

j. Profile Update Issues

- **Issue**: Some users are unable to update their profiles, with changes not saving consistently.
- **Cause**: Potential issues with form validation or improper API handling during profile updates.
- **Temporary Workaround**: Try updating the profile again or clearing the browser cache. Developers should validate API handling and ensure form data is correctly processed.

FUTURE ENHANCEMENTS

a. Enhanced Personalization and Recommendations

- Implement machine learning algorithms to provide smarter, more personalized product recommendations based on user behavior, purchase history, and browsing patterns.
- Develop dynamic profiles to store user preferences, allowing for tailored product suggestions, targeted promotions, and personalized shopping experiences.

b. Advanced Search and Filter Options

- Enable users to search for products using voice commands or by uploading images to find visually similar items.
- Add more advanced filtering options like price range sliders, brand selections, and rating-based filters to improve product discovery.

c. Real-Time Order Tracking and Notifications

- Send real-time notifications to users about their order status, including shipping updates and estimated delivery times.
- Implement notifications within the app and push notifications on mobile to keep users informed about discounts, new arrivals, and cart abandonment reminders.

d. Multi-Language and Multi-Currency Support

- Translate the app into multiple languages to support international users, and offer currency conversions based on the user's location or preference.
- Display prices in the local currency of the user with an option to manually switch between currencies for better accessibility.

e. Enhanced Security Features

● Add 2FA for users to add an extra layer of security to their accounts. ● Integrate a fraud detection module that flags suspicious transactions and automatically triggers additional verification steps.

f. Wishlist and Social Sharing Options

- Allow users to save items in a wishlist for future purchases, with notifications for price drops or stock availability.

- Enable users to share products or wishlists on social media platforms for collaborative shopping experiences.