SHOP (MERN)

TEAM MEMBERS:

1. **KAVITHA B**	- Frontend
2. **ARCHANA V**	- Backend
3. **SHREENITHI RP**	- Admin
4. **ROONESHA U**	- Admin
5. **RISHEKA V S**	- Testing

1.PROJECT OVERVIEW:

The Shop eCommerce Application is a comprehensive platform built using the MERN stack, designed to provide an efficient and enjoyable online shopping experience. It caters to both customers and administrators, offering user-friendly features, secure payment options, and advanced management tools. The project aims to streamline the eCommerce process with scalability, responsiveness, and robust functionality.

- **Frontend (React.js):** Provides a responsive and dynamic interface for product browsing, cart management, secure checkout, and order tracking, ensuring usability across all devices.
- **Backend (Node.js & Express.js):** Powers API endpoints, secure transactions, and real-time updates for inventory, orders, and payments, ensuring high performance and data security.
- **Admin Panel:** Simplifies management with tools for inventory, orders, analytics, and customer engagement through promotions and notifications.
- **Database (MongoDB):** A scalable solution for managing products, users, and orders.

I. Purpose of the Project

- Provide a seamless shopping experience with easy product browsing, advanced search filters, efficient cart management, and secure checkout.
- Enable customers to manage their profiles, track their orders, and view their purchase history conveniently.
- Empower administrators to efficiently manage product inventory, process orders, and generate detailed sales reports for improved business decision-making.
- Facilitate secure and diverse payment options through integrated gateways, ensuring safe and reliable transactions for customers.
- Support business growth by providing global reach, marketing tools, and customer engagement features like discounts, promotions, and personalized campaigns.

II. Goals of the Project

- Create a user-friendly, responsive platform that ensures an enjoyable and intuitive shopping experience across all devices.
- Develop an efficient, secure backend system capable of handling high volumes of transactions, orders, and user data.
- Provide administrators with powerful tools to streamline product management, inventory control, and order processing.
- Integrate secure payment gateways to ensure safe and smooth transactions for all customers.
- Build a scalable platform that can grow with the business, offering new features, supporting marketing strategies, and handling increased traffic as the user base expands.

2.ARCHITECTURE:

I. Frontend Architecture

The frontend of the Shop eCommerce application is built with React.js to ensure a dynamic and responsive user experience. The architecture follows key React principles, focusing on components, state management, and routing to provide a smooth user interface.

Key Concepts:

- **Component-Based Architecture:** The application is made of reusable UI components (e.g., buttons, product cards) that are either functional (using React Hooks) or class-based.
- **JSX:** React uses JSX to write HTML-like code inside JavaScript, which gets compiled into UI components (e.g., <button onClick={handleClick}>Add to Cart</button>).
- **State & Props Management:** State manages dynamic data like user info and shopping cart items, while props pass data between components.
- **React Router:** Used for navigating between pages like homepage, product details, and cart, without full-page reloads.
- **State Management (React Context/Redux):** Global state (e.g., user authentication, cart items) is managed using React Context API or Redux.

Breakdown of Key Folders and Files:

- **Components:** Reusable UI elements like Navbar, CartItems, Footer, Hero, and Payment components are included.
- **Pages:** Page components such as HomePage, ProductPage, CartPage, CheckoutPage, etc., are routed through React Router.
- **Context:** Manages global state like cart data and product details using ShopContext.js.
- **Hooks:** Custom hooks like useAuth.js for authentication and useCart.js for managing cart state.
- **Styles:** Contains global styles and component-specific styles using CSS or styled-components.
- **Routes:** Configures routes for different pages, including dynamic categories and product details.

II. Backend Architecture

The backend handles business logic, user authentication, product management, and payment processing using **Node.js** and **Express.js**. Node.js is ideal for scalable, I/O-heavy applications, while Express.js simplifies routing and API creation.

Key Components:

- **Node.js**: JavaScript runtime for building scalable, asynchronous applications.
- **Express.js**: Framework for routing, middleware, and RESTful API creation.
- **RESTful API**: Exposes endpoints for managing products, users, orders, and payments using HTTP methods (GET, POST, PUT, DELETE).
- **Authentication**: Handled via **JWT** for secure login and session management.
- **MongoDB**: Database for storing product, user, and order data, with **Mongoose** used for interactions.

Key Folders and Files:

- **config**: Contains DB connection logic, environment variables, and server configurations.
- **controllers**: Manages request handling, interacting with models and sending responses.
- **models**: Defines Mongoose schemas for entities like Product, User, and Order.
- **routes**: Maps API endpoints to controller functions.
- **middleware**: Validates input, handles authentication, and manages errors.

The backend is initialized in index.js, where the server is set up, the database is connected, and routes and middleware are configured.

III. Database Schema and Interaction with MongoDB

In the Shop eCommerce application, **MongoDB** will store essential data like user profiles, product details, and orders. MongoDB's flexible document-based schema suits the evolving data structure of an eCommerce platform.

Key MongoDB Collections:

- **User**: Stores user profiles and authentication data.
- **Product**: Stores product details.
- **Order**: Contains order information.
- **Cart**: Stores items in the shopping cart.
- **Payment**: Manages payment details.

Mongoose Schema Definitions:

- **User Schema**: Stores user details such as name, email, hashed password, role (admin/customer), and optional fields like address and profile picture.
- **Product Schema**: Stores product info including name, price, description, category, stock, and ratings.
- **Order Schema**: Tracks orders with details like products, total amount, shipping address, payment status, and order status.

Interactions with MongoDB:

Mongoose provides an easy-to-use API for performing CRUD operations on collections.

3. SETUP INSTRUCTIONS :

I. Prerequisites

Before you begin setting up the application, make sure you have the following software installed on your machine:

- **Node.js** (version 14 or higher)
  - Download Node.js
- **MongoDB** (version 4.4 or higher)
  - Download MongoDB

Additionally, you'll need a code editor such as Visual Studio Code to view and edit the project files.

II. Installation

Clone Repository:

- Clone the repo: git clone <repository-url>
- Navigate to project directory: cd <project-directory>

Install Dependencies:

- Frontend (React):
  ```
  cd frontend
  npm install
  ```
- Backend (Node.js):
  ```
  cd backend
  npm install
  ```

Set Up Environment Variables:

- Frontend (.env):
  REACT_APP_API_URL=http://localhost:3000
- Backend (.env):
  DB_URI=<mongodb+srv://kavithabalaji:ecommerceshophere@cluster0.yl4pl.mongodb.net/e-commerce>
- PORT=3000
  JWT_SECRET=mySecretKey

MongoDB Setup:

- Ensure MongoDB is running locally or update .env for MongoDB Atlas.

Run Database Migrations:

- Follow backend README or script files for migrations.

4. FOLDER STRUCTURE :

Frontend (Client):

- **node_modules:** npm packages for the frontend.
- **public/:** Static files like index.html and assets.
- **Components:** Reusable UI components (e.g., Navbar, Footer).
- **Pages:** Specific pages (e.g., Cart, ShopCategory).
- **Context:** Manages state (e.g., ShopContext.jsx).
- **Css:** Styling files (e.g., App.css).
- **.gitignore:** Specifies files to ignore by Git.
- **package.json:** Lists dependencies and scripts.
- **package-lock.json:** Ensures consistent dependency versions.

Backend (Server):

- **node_modules:** npm packages for the backend.
- **upload/:** Stores uploaded files (e.g., product images).
- **index.js:** Server entry point with config, middleware, and routing.
- **package.json:** Lists backend dependencies and scripts.

- **package-lock.json**: Ensures consistent backend dependency installation.

5. RUNNING THE APPLICATION

To run the application locally, you'll need to start both the frontend and backend servers. Follow the commands below to launch each part of the application:

Frontend

1. Navigate to the client directory:

```
cd frontend
```

2. Start the React development server:

```
npm start
```

This will run the frontend application on http://localhost:3000

Backend (Node.js)

1. Navigate to the server directory:

```
cd backend
```

2. Start the Node.js server:

```
node .\index.js
```

This will run the backend server on http://localhost:4000

ADMIN

1. Navigate to the server directory:

```
cd admin
```

2. Start the Node.js server:

```
npm run dev
```

This will run the backend server on http://localhost:5173

6. API DOCUMENTATION

Below is an outline of the backend API endpoints:

i.GET /

Description: Verifies if the backend server is running.
Request Method: GET
Parameters: None

---

ii.POST /upload

Description: Uploads an image file and provides its publicly accessible URL.
Request Method: POST
Parameters:

Response Example (Success):

json

```
{ "success": 1,"image_url": "http://localhost:4000/images/<uploaded_filename>"

}
```

Response Example (Failure):

json

```
{

  "success": 0,"message": "File upload failed"

}
```

---

iii. POST /addproduct

Description: Adds a new product to the database.
Request Method: POST
Request Body:

json

{

  "name": "Product Name","image": "Image URL","category": "Category Name","new_price": 100,"old_price": 120

}

Response Example:

json

{

"success": true, "name": "Product Name"

}

---

iv. POST /removeproduct

Description: Removes a product from the database based on its ID.
Request Method: POST
Request Body:

json

{

  "id": 1

}

Response Example:

json

{

  "success": true,"name": "Product Name"

}

---

v. GET /allproducts

Description: Retrieves all products from the database.
Request Method: GET
Parameters: None
Response Example:

json

[

  {

    "id": 1,"name": "Product Name","image": "Image URL", "category": "Category Name", "new_price": 100,"old_price": 120, "date": "2024-11-16T10:15:30Z", "available": true

  }

]

---

vi. POST /signup

Description: Registers a new user.
Request Method: POST
Request Body:

json

{

  "username": "User Name","email": "user@example.com","password": "password123"

}

Response Example (Success):

json

{

  "success": true,"token": "<JWT Token>"

}

Response Example (Failure):

json

{

"success": false,"errors": "Existing user found with same email address"

}

---

vii. POST /login

Description: Logs in an existing user and returns a token.
Request Method: POST
Request Body:

json

{

"email": "user@example.com","password": "password123"

}

Response Example (Success):

json

{

"success": true, "token": "<JWT Token>"

}

Response Example (Failure):

json

{

"success": false,"errors": "Wrong Email ID"

}

---

viii. GET /newcollections

Description: Retrieves the latest 8 products added to the database.
Request Method: GET
Parameters: None
Response Example:

json

[

  {

    "id": 8,"name": "New Product","image": "Image URL","category": "Category Name","new_price": 150,"old_price": 170

}

]

---

ix. GET /popularinwomen

Description: Retrieves 4 popular products in the women's category.
Request Method: GET
Parameters: None
Response Example:

json

[

  {

    "id": 1,"name": "Product Name","image": "Image URL","category": "women","new_price": 100,"old_price": 120

  }

]

---

x. POST /addtocart

Description: Adds a product to the authenticated user's cart.
Request Method: POST
Headers:

Request Body:

json

{

"itemId": 1

}

Response Example:

json

"Added"

---

xi. POST /removefromcart

Description: Removes a product from the authenticated user's cart.
Request Method: POST

Request Body:

json

{

"itemId": 1

}

Response Example:

json

"Removed"

---

xii. POST /getcart

Description: Fetches the authenticated user's cart data.
Request Method: POST
Response Example:

json

{

"1": 2,

"2": 0,

"3": 1 }

7. AUTHENTICATION :

Authentication and authorization JWT (JSON Web Token) is used in this project.

How it works:

1. Registration:

Whenever a user signs up, the password is hashed against bcrypt and saved to the database for secure storage.

2. Login:

Once the user logs in successfully, it generates a JWT having the ID of the user and signing it with some secret key.

3. Token Usage:

In this case, each protected request sends the token in the Authorization header (Bearer <token>).

4.Authentication Middleware:

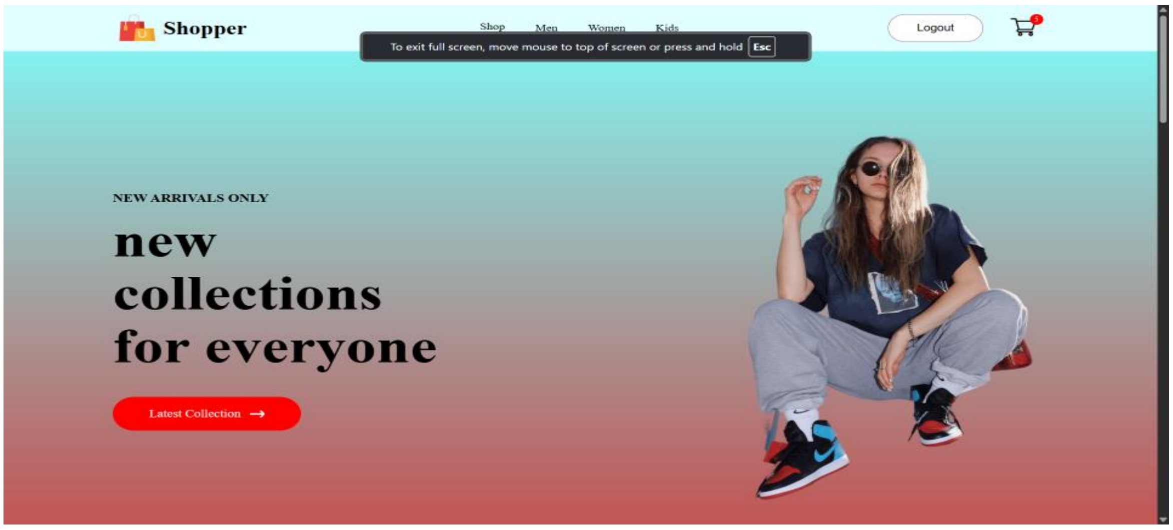
This middle ware function validates a token and allows only authenticated users to access the protected routes.

8. User Interface:

Screenshots of UI:

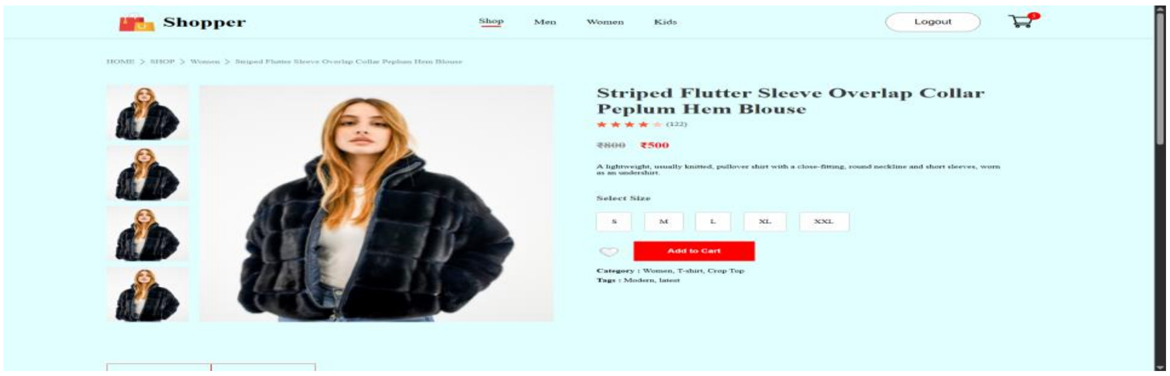
Home Page

Displays all available products with search and category filter options.


Shopper
Shop
Men
Women
Kids
Logout
To exit full screen, move mouse to top of screen or press and hold Esc
NEW ARRIVALS ONLY
new collections for everyone
Latest Collection

Product Details Page

Detailed view of the product with an "Add to Cart" button.



HOME > SHOP > Women > Striped Flutter Sleeve Overlap Collar Peplum Hem Blouse

Striped Flutter Sleeve Overlap Collar Peplum Hem Blouse

★★★★☆ (122)

~~₹800~~ ₹500

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

Select Size

S M L XL XXL

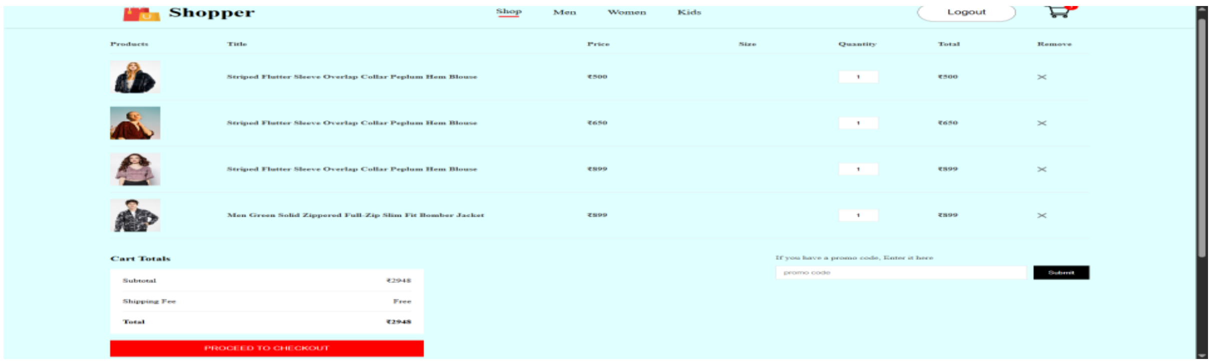
Add to Cart

Category : Women, T-shirt, Crop Top

Tags : Modern, latest

Shopping Cart

It would display the products under cart with quantity and price breakdown.



Products	Title	Price	Size	Quantity	Total	Remove
	Striped Flutter Sleeve Overlap Collar Peplum Hem Blouse	₹500		1	₹500	✕
	Striped Flutter Sleeve Overlap Collar Peplum Hem Blouse	₹650		1	₹650	✕
	Striped Flutter Sleeve Overlap Collar Peplum Hem Blouse	₹899		1	₹899	✕
	Men Green Solid Zippered Full-Zip Slim Fit Bomber Jacket	₹899		1	₹899	✕

Cart Totals

Subtotal	₹2948
Shipping Fee	Free
Total	**₹2948**

PROCEED TO CHECKOUT

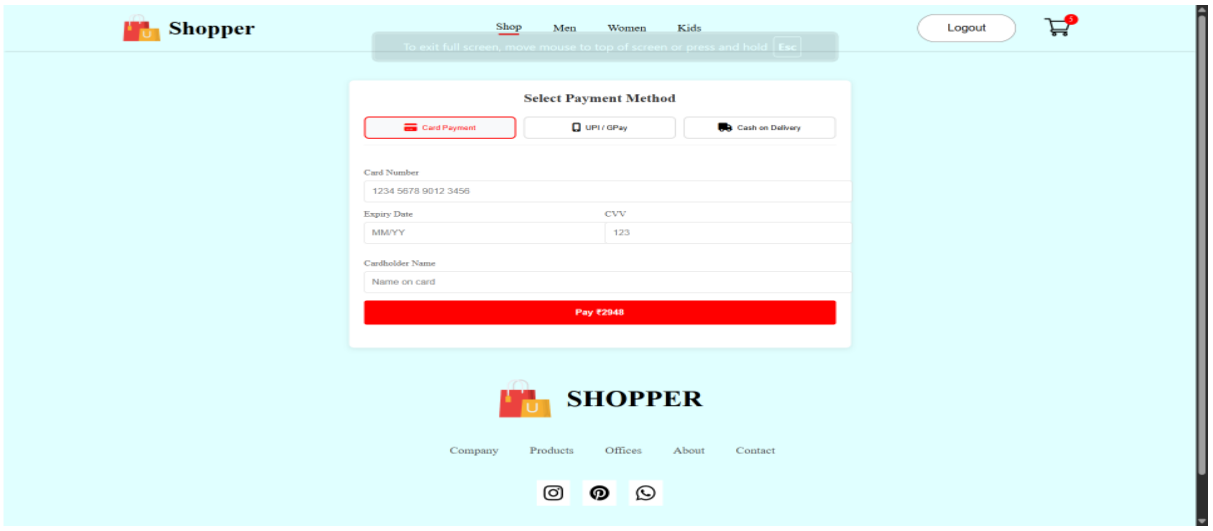
If you have a promo code, Enter it here

promo code

Submit

Payment Process

Displays the list of purchased items, quantity, price, and total amount.


Shopper
Shop
Men
Women
Kids
Logout
To exit full screen, move mouse to top of screen or press and hold Esc

Select Payment Method

Card Payment | UPI / GPay | Cash on Delivery

Card Number: 1234 5678 9012 3456

Expiry Date: MM/YY

CVV: 123

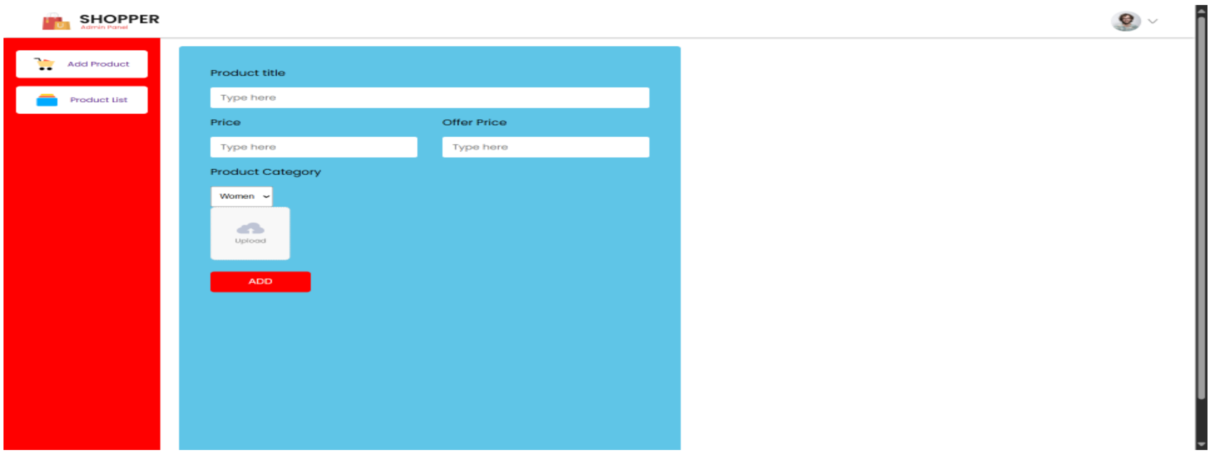
Cardholder Name: Name on card

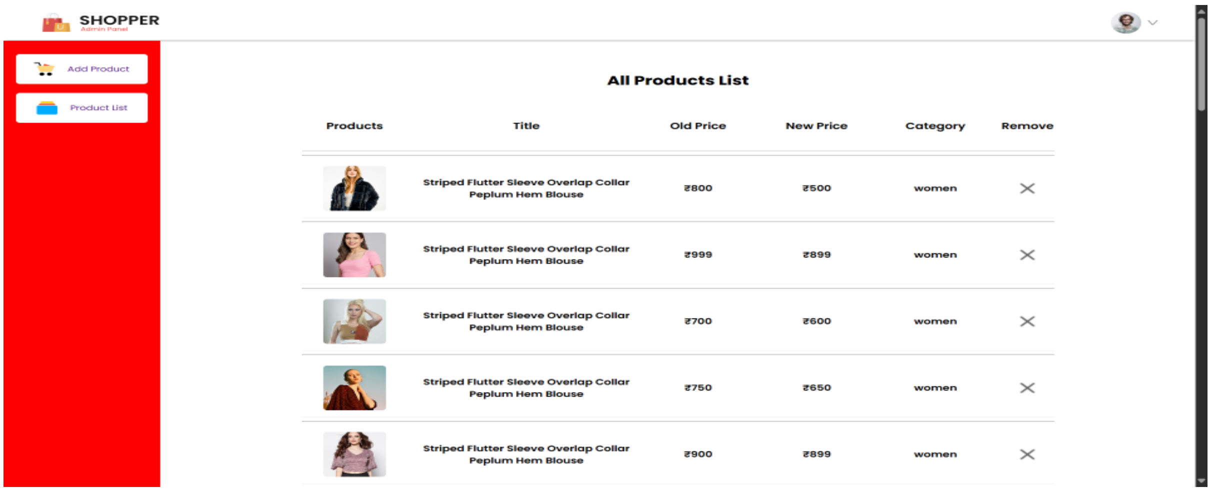
Pay ₹2948

SHOPPER
Company
Products
Offices
About
Contact

Admin Panel

It must provide an interface for administering products, users, and orders.





All Products List

Products	Title	Old Price	New Price	Category	Remove
	Striped Flutter Sleeve Overlap Collar Peplum Hem Blouse	₹800	₹500	women	✕
	Striped Flutter Sleeve Overlap Collar Peplum Hem Blouse	₹999	₹899	women	✕
	Striped Flutter Sleeve Overlap Collar Peplum Hem Blouse	₹700	₹600	women	✕
	Striped Flutter Sleeve Overlap Collar Peplum Hem Blouse	₹750	₹650	women	✕
	Striped Flutter Sleeve Overlap Collar Peplum Hem Blouse	₹900	₹899	women	✕

9. TESTING

The testing strategy for the E-Commerce Shop includes the following components:

1. **Unit Testing:**
   - Mocha and Chai are used to verify individual modules/functions.
2. **Static Analysis:**
   - ESLint ensures code adheres to style and quality guidelines.
3. **Code Coverage:**
   - NYC tracks code coverage during testing to measure test coverage.

4. **Script-Based Testing Workflow:**
   - **"test":** Runs test cases (via Mocha).
   - **"coverage":** Generates code coverage reports (via NYC).
   - **"lint":** Runs ESLint for code linting.

Tools Used:

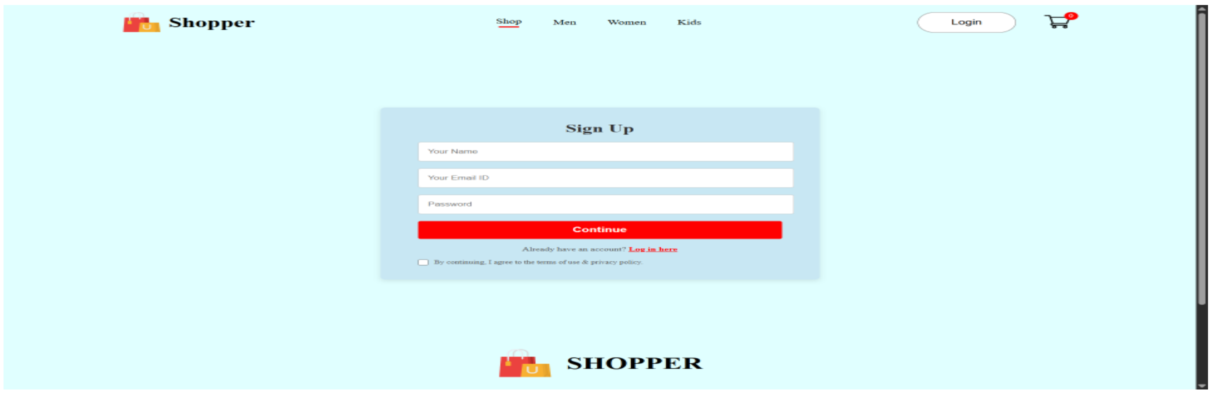
- **Mocha:** JavaScript test framework for asynchronous testing.
- **Chai:** Assertion library for tests.
- **ESLint:** Identifies and fixes JavaScript code issues.
- **nyc:** Tracks code coverage.

This approach ensures comprehensive testing, code quality, and maintainability.

10. SCREENSHOTS OR DEMO

FRONTEND SCREENSHOTS

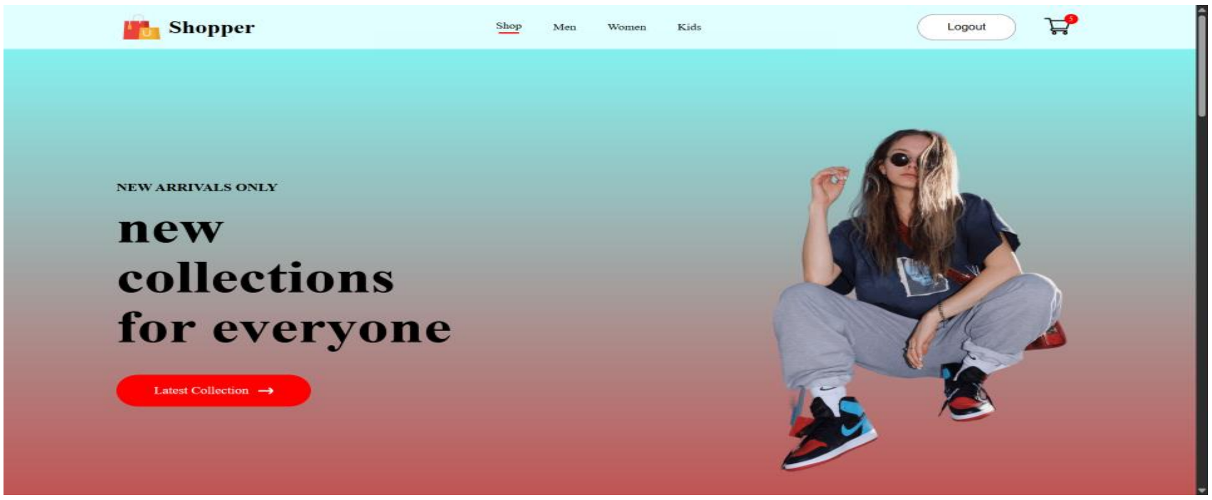
LOGIN/SIGNUP



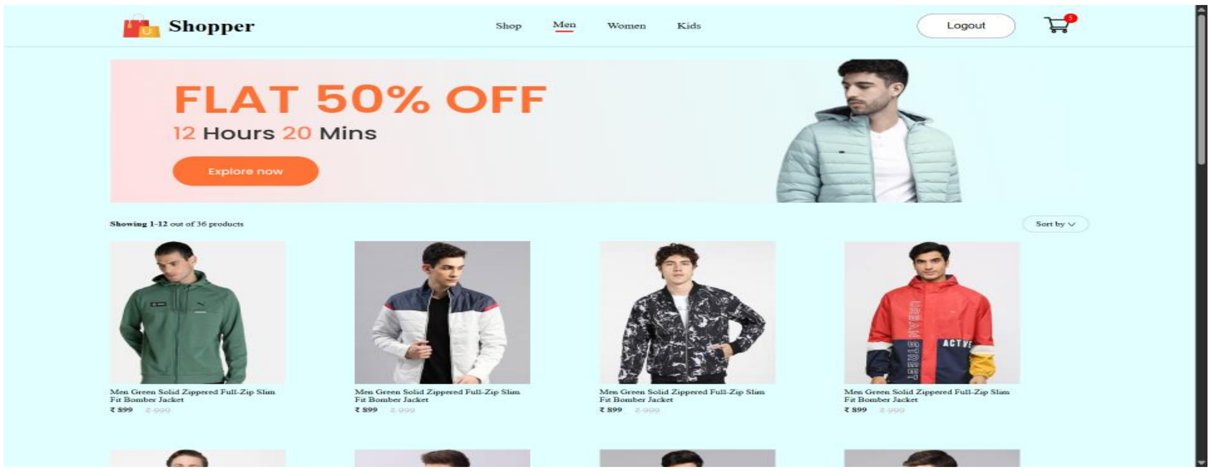
Already have an account? Log in here

By continuing, I agree to the terms of use & privacy policy.

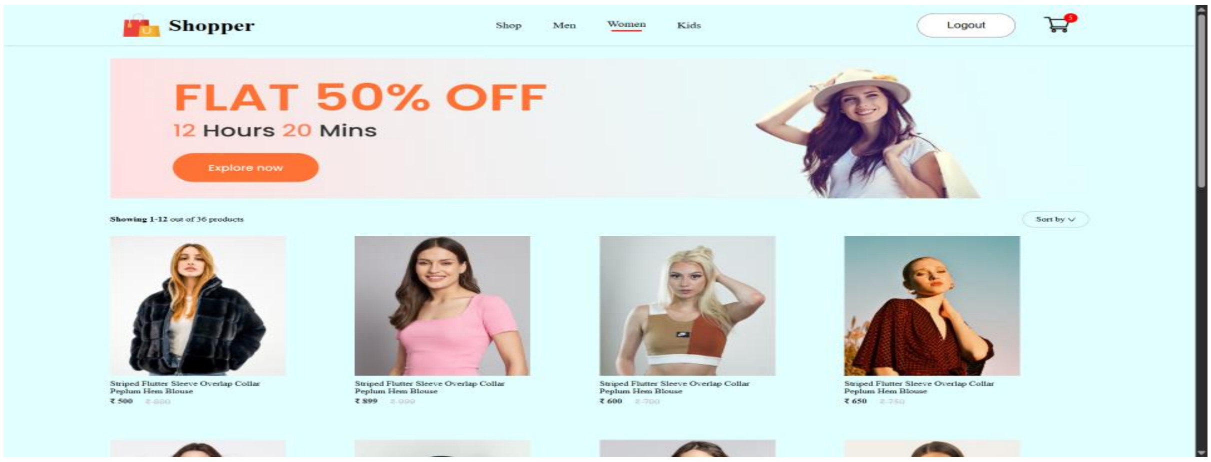
HOME PAGE (SHOP)



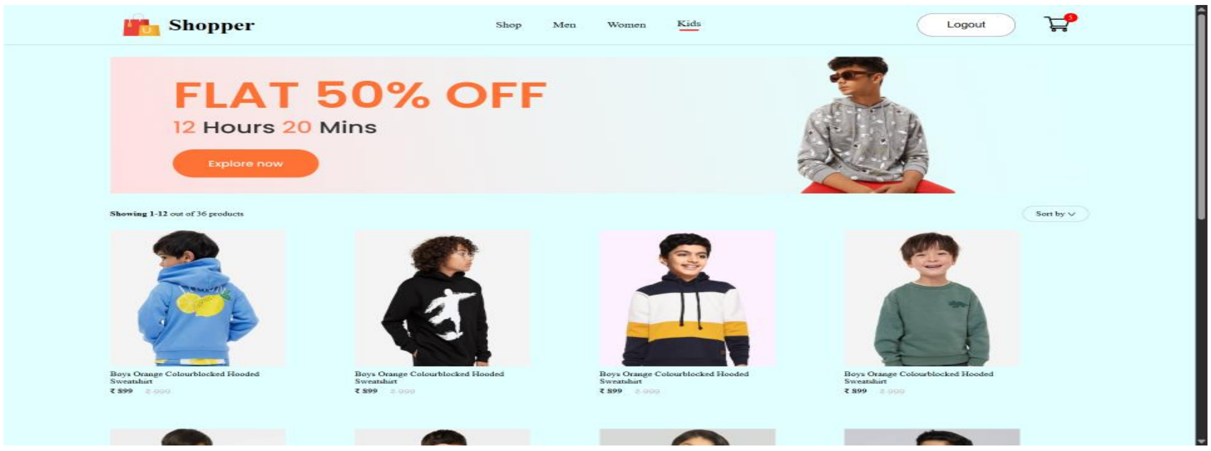
MEN PAGE


Shopper
Shop
Men
Women
Kids
Logout
FLAT 50% OFF
12 Hours 20 Mins
Explore now
Showing 1-12 out of 36 products
Sort by
Men Green Solid Zippered Full-Zip Slim Fit Bomber Jacket
₹ 899 ₹ 999
Men Green Solid Zippered Full-Zip Slim Fit Bomber Jacket
₹ 899 ₹ 999
Men Green Solid Zippered Full-Zip Slim Fit Bomber Jacket
₹ 899 ₹ 999
Men Green Solid Zippered Full-Zip Slim Fit Bomber Jacket
₹ 899 ₹ 999

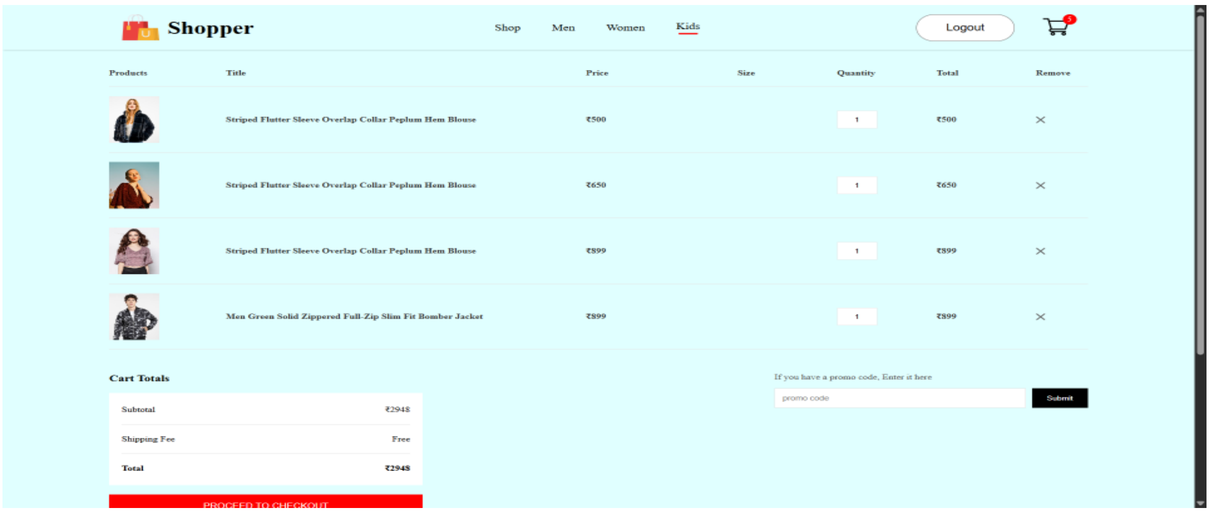
WOMEN PAGE


Shopper
Shop
Men
Women
Kids
Logout
FLAT 50% OFF
12 Hours 20 Mins
Explore now
Showing 1-12 out of 36 products
Sort by
Striped Flutter Sleeve Overlap Collar Peplum Hem Blouse
₹ 500 ₹ 600
Striped Flutter Sleeve Overlap Collar Peplum Hem Blouse
₹ 899 ₹ 999
Striped Flutter Sleeve Overlap Collar Peplum Hem Blouse
₹ 600 ₹ 700
Striped Flutter Sleeve Overlap Collar Peplum Hem Blouse
₹ 650 ₹ 740

KIDS PAGE


Shopper
Shop
Men
Women
Kids
Logout
FLAT 50% OFF
12 Hours 20 Mins
Explore now
Showing 1-12 out of 36 products
Sort by
Boys Orange Colourblocked Hooded Sweatshirt
₹ 899 ₹ 999
Boys Orange Colourblocked Hooded Sweatshirt
₹ 899 ₹ 999
Boys Orange Colourblocked Hooded Sweatshirt
₹ 899 ₹ 999
Boys Orange Colourblocked Hooded Sweatshirt
₹ 899 ₹ 999

CART PAGE



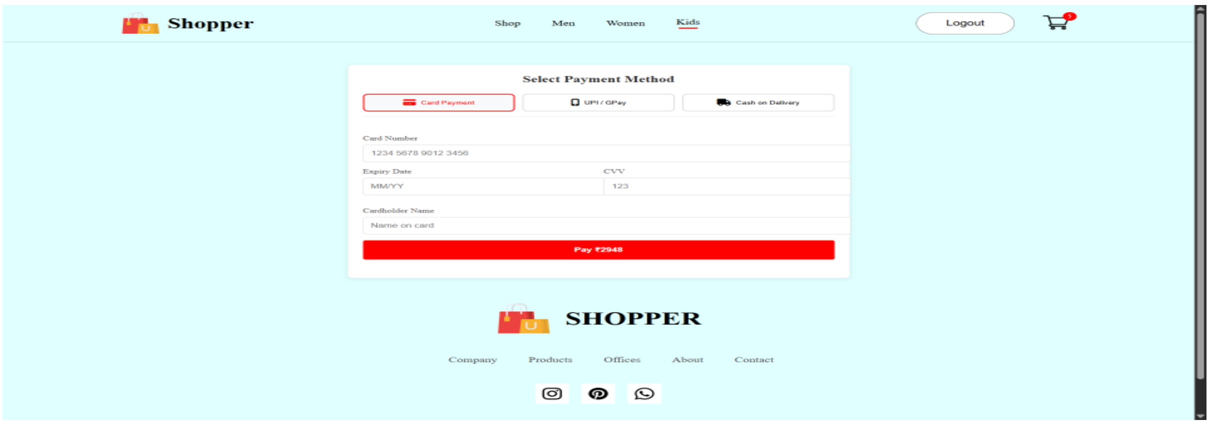
Products	Title	Price	Size	Quantity	Total	Remove
	Striped Flutter Sleeve Overlap Collar Peplum Hem Blouse	₹500		1	₹500	×
	Striped Flutter Sleeve Overlap Collar Peplum Hem Blouse	₹650		1	₹650	×
	Striped Flutter Sleeve Overlap Collar Peplum Hem Blouse	₹899		1	₹899	×
	Men Green Solid Zippered Full-Zip Slim Fit Bomber Jacket	₹899		1	₹899	×

Cart Totals

Subtotal	₹2948
Shipping Fee	Free
Total	**₹2948**

If you have a promo code, Enter it here

PAYMENT PROCESS



Select Payment Method

Card Number: 1234 5678 9012 3456

Expiry Date: MM/YY

CVV: 123

Cardholder Name: Name on card

11. Known Issues

- **Cart Synchronization Issue:** Cart contents not saved on logout, causing emptiness on login. **Impact:** Medium. **Workaround:** Complete checkout before logout; fix in progress.
- **Pagination Bug:** Pagination controls not displaying properly with fewer products. **Impact:** Low. **Workaround:** Manually adjust query parameters; UI patch in progress.
- **Slow Performance on High-Traffic Pages:** Slow page loads during peak traffic. **Impact:** High. **Workaround:** Implementing caching (Redis) and query optimizations.

- **JWT Token Expiry Handling:** Expired tokens not detected, causing authorization errors. **Impact:** High. **Workaround:** Manual refresh; auto-logout feature in development.
- **Mobile Responsiveness Issues:** UI problems with checkout and filters on mobile. **Impact:** Medium. **Workaround:** Use desktop; mobile fixes in progress.

12. FUTURE ENHANCEMENTS

- **User Experience:** AI-driven product recommendations, improved search filters, and multi-language/currency support.
- **Security:** Two-factor authentication (2FA), secure payment gateways, and rate-limiting for protection.
- **Scalability & Performance:** Server-side rendering (SSR), database optimization, and Redis caching for faster performance.
- **Advanced Features:** Wishlist, product reviews, live chat support, and abandoned cart recovery.
- **Mobile & Analytics:** PWA support, mobile app development, and real-time admin dashboards for insights.