**MERN eCommerce Grocery WebApp**

**1. Introduction**

* **Project Title**: MERN eCommerce Grocery WebApp
* **Team Members**:
  + **Harishma Sabu**
    - Register Number: 2021503018
    - NMID: 43501E637C3BC1090354E888D86CB5C4
  + **Vidhya R**
    - Register Number: 2021503516
    - NMID: B3DAF0D50AFC002987E41E63041A5630
  + **Kiruthika Y B**
    - Register Number: 2021503516
    - NMID: 3A28E9BBADE2266A03A43751A5B5BE47
  + **Ashwin Muthu Chandru**
    - Register Number: 2021503004
    - NMID: C03D92B4B1ADD94A9F9404529279B96A

**2. Project Overview**

**Purpose**

The MERN eCommerce Shopping Cart is a modern web application that allows users to browse products online. It includes essential features such as user authentication, cart management (add, remove, update products), and a responsive user interface.

**Features**

* **User Authentication**:
  + Signup, login, and logout functionality.
  + Passwords are securely hashed with **bcrypt**.
  + Authentication is handled with **JWT (JSON Web Tokens)**.
* **Form Validation**:
  + **Formik** for handling forms.
  + **Yup** for schema validation of form inputs.
* **Cart Management**:
  + Users can add products to their cart, update product quantities, and remove items.
* **Product Detail Page**:
  + Each product has its own detail page with a description and pricing information.
* **Responsive UI**:
  + The application is fully responsive and designed with **Tailwind CSS** for a modern look.

**3. Architecture**

**Frontend**

The frontend is built with **React**. State management is handled using **Zustand**, while **Axios** is used to make API requests to the backend. The user interface is styled with **Tailwind CSS** to provide a modern, responsive design.

**Backend**

The backend is built with **Node.js** and **Express.js**. **MongoDB** is used to store user and product data. User authentication is handled with **JWT** for secure login, and **bcrypt** is used for password hashing.

**Database**

MongoDB is used for storing product and user data. The database is designed with collections for users, products, and orders. The backend interacts with MongoDB using **Mongoose** to manage and query the data.

**4. Setup Instructions**

**Prerequisites**

* Install **Node.js**: [Download Node.js](https://nodejs.org/)
* Install **MongoDB**: [Download MongoDB](https://www.mongodb.com/try/download/community)

**Installation Steps**

1. Clone the repository from GitHub:

bash

Copy code

git clone [repository-url]

1. Navigate to the project directory and install the dependencies:

bash

Copy code

npm install

1. Configure the MongoDB connection string in the .env file:  
   Example:

makefile

Copy code

MONGO\_URI=your-mongodb-connection-string

JWT\_SECRET=your-jwt-secret

1. Start the development server:

bash

Copy code

npm run dev

**5. Folder Structure**

**Client (Frontend)**

* src/components/: Contains React components like Header, Cart, ProductList, etc.
* src/store/: Contains Zustand state management files.
* src/utils/: Contains utility functions like Axios API calls.

**Server (Backend)**

* models/: Contains MongoDB models for User and Product.
* routes/: Contains routes for handling authentication, products, and cart functionality.
* controllers/: Handles the logic for each route (e.g., adding a product to the cart).
* middleware/: Contains middleware for authentication and authorization.

**6. Running the Application**

**Frontend**

To run the frontend, navigate to the client directory and use the following command:

bash

Copy code

npm start

**Backend**

To run the backend, navigate to the server directory and use the following command:

bash

Copy code

npm run dev

**7. API Documentation**

**POST /api/auth/signup**

* **Description**: Registers a new user.
* **Body**:
  + email: User's email (string)
  + password: User's password (string)
* **Response**:
  + Success: Returns a JWT token.
  + Failure: Error message.

**POST /api/auth/login**

* **Description**: Logs in an existing user.
* **Body**:
  + email: User's email (string)
  + password: User's password (string)
* **Response**:
  + Success: Returns a JWT token.
  + Failure: Error message.

**GET /api/products**

* **Description**: Fetches all products.
* **Response**:
  + Returns an array of products.

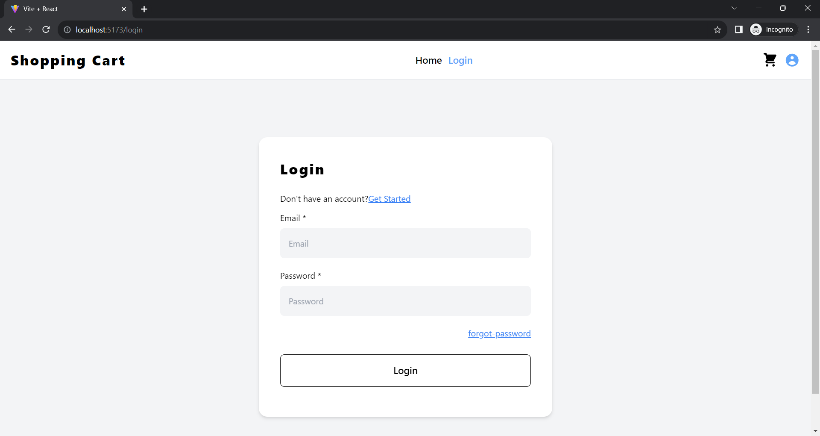
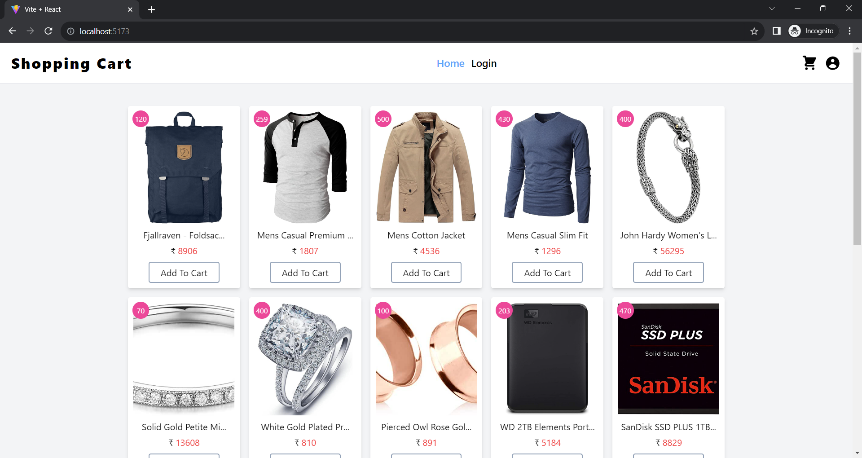
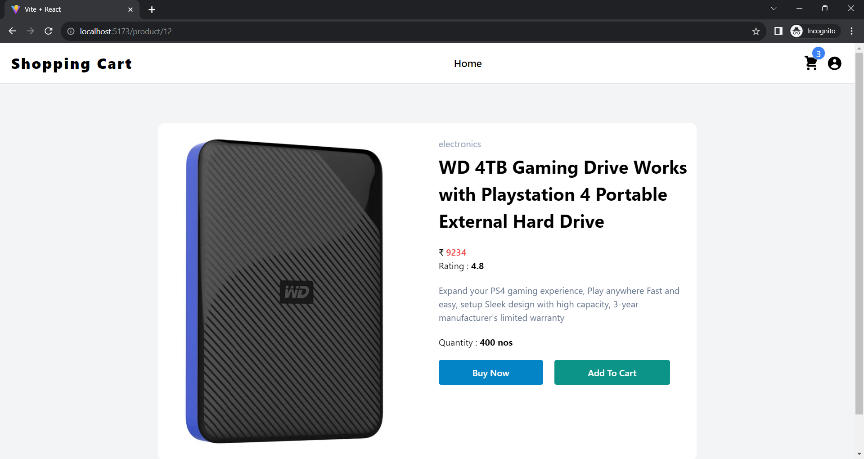
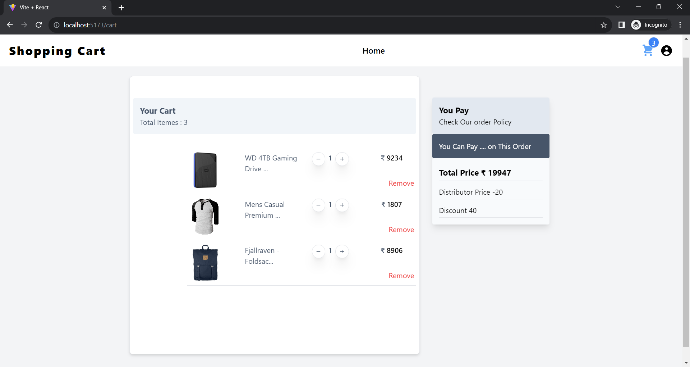
**POST /api/cart**

* **Description**: Adds a product to the cart.
* **Body**:
  + productId: Product ID (string)
  + quantity: Quantity to be added (number)
* **Response**:
  + Success: Returns updated cart.

**8. Authentication**

Authentication is handled using **JWT** (JSON Web Tokens). Users can sign up, log in, and log out. The JWT is used to authenticate user actions in the app, such as adding items to the cart and checking out. **bcrypt** is used to hash and store passwords securely.

**9. User Interface**

* **Login Screen**
* **Product Listing Page**
* **Product Detail Page**
* **Cart Page**

**10. Testing**

The project is tested using a combination of unit tests for individual components and API endpoints. The frontend tests are written using **Jest** and **React Testing Library**, while the backend is tested with **Mocha** and **Chai**.

**11. Demo**

You can view a live demo of the application here:

https://drive.google.com/file/d/1lz9qJ0PV\_ndrQPPB\_ZcTdvdRtufTxtCE/view

**12. Known Issues**

* **Login Issue**: Occasionally, JWT tokens may expire unexpectedly causing issues during user login.
* **Cart Update**: The cart update may not reflect immediately due to frontend state management issues with Zustand.

**13. Future Enhancements**

* **Payment Gateway Integration**: Integrate payment gateways like Stripe or PayPal for real transactions.
* **Product Search**: Implement search functionality to allow users to find products quickly.
* **User Profile**: Add a feature where users can view and update their profile information.