Full Stack Development with MERN

1.INTRODUCTION:

PROJECT TITLE : **SHOPEZ:E-COMMERCE APPLICATION**

TEAM MEMBERS: 1. Kowsalya M

2.Kowsalya B

3.Swetha M

4.Swetha V

2.PROJECT OVERVIEW:

PURPOSE: The purpose of **Shopez**—an **e-commerce application**—is to create a seamless, user-friendly online shopping platform that connects buyers and sellers, offering a wide range of products in a secure and convenient way. The project aims to provide a comprehensive ecommerce solution for customers, enabling them to browse, purchase, and track products from multiple categories (e.g., electronics, fashion, groceries, etc.).

FEATURES: The **Shopez E-Commerce Application** is designed to provide a seamless and userfriendly shopping experience for customers, while offering powerful tools for sellers to manage their products and sales. The platform will feature a comprehensive **product catalog** with detailed descriptions, images, and advanced search filters, allowing users to easily find and purchase items across multiple categories. Customers can create personalized **accounts**, manage their **order history**, and track shipments, while benefiting from **secure payment options** like credit/debit cards, digital wallets, and cash on delivery. The app will also include a **shopping cart**, **wishlist**, and **discount/coupon support**, ensuring users can shop conveniently and take advantage of ongoing promotions.

3.ARCHITECTURE:

FRONTEND:

The **frontend architecture** of the **Shopez E-Commerce Application** is built with **React.js** to provide a dynamic and responsive user experience. The project structure is modular, with dedicated folders for components, pages, services, and state management, making it scalable and maintainable. Performance optimization techniques like **code splitting**, **image compression**, and **service workers** are employed to ensure fast load times and offline functionality. The architecture also includes **unit testing** with **Jest** and **React Testing Library**, and **end-to-end testing** with **Cypress** to ensure reliability.

BACKEND:

The **backend architecture** of the **Shopez E-Commerce Application** is built using **Node.js** and **Express.js**, providing a scalable and efficient environment to handle various aspects of the platform, such as user authentication, product management, order processing, and payment integration

DATABASE: The **database** for the **Shopez E-Commerce Application** is built using **MongoDB**, a NoSQL database that offers flexibility and scalability to handle the dynamic nature of e-commerce data. MongoDB's document-oriented model is well-suited for storing varied product information, user profiles, orders, and transactions in a way that can easily accommodate future growth. The database schema includes collections such as **Users**, **Products**, and **Orders**.

4.SETUP INSTRUCTIONS:

PREREQUISTIES: The **prerequisites** for developing the **Shopez E-Commerce Application** include a system with **Node.js**, **npm**, **MongoDB**, **Git**, a text editor (like **VS Code**), knowledge of **JavaScript**, **React.js**, **Express.js**, **MongoDB**, and access to services like **Stripe** (for payments) and cloud hosting (e.g., **AWS** or **Heroku**).

INSTALLATION :

To install the **Shopez E-Commerce Application**, start by cloning the project repository from GitHub to your local machine using the command `git clone https://github.com/yourrepository/shopez-ecommerce.git`

5.FOLDER STRUCTURE:

CLIENT:

```
/client
├── /public              # Static files like index.html, images, and assets
│   ├── index.html       # Main HTML file │   └──
/assets          # Images, logos, fonts, etc.
│
├── /src                 # Main source code
│   ├── /components      # Reusable UI components (Button, Navbar, ProductCard)
│   ├── /pages           # Page components (Home, ProductList, Checkout)
│   ├── /redux           # Redux store, actions, reducers for state management
│   │   ├── actions.js   # Redux action creators
│   │   ├── reducers.js  # Redux reducers for handling state updates │
│   └── store.js         # Configuring Redux store
│   ├── /services        # API calls to the backend (Axios requests)
│   ├── /styles          # Global CSS or SCSS files
│   ├── /utils           # Utility functions (helpers, constants, etc.)
│   ├── App.js           # Main App component that includes routes and layout
```

```
│   ├── index.js          # Entry point, renders the App component
│   └── routes.js         # Route configuration for React Router
├── .env                  # Environment variables (e.g., API URL)
├── package.json          # Project dependencies, scripts
└── README.md             # Project documentation
```

SERVER:

```
├── /config               # Configuration files for database, environment variables
│   ├── db.js             # MongoDB connection setup
│   ├── config.js         # Other configuration settings (e.g., JWT secrets)
│   └── constants.js      # Constants used across the server (e.g., error messages)
├── /controllers          # Logic to handle API requests and business logic
│   ├── authController.js     # Authentication (login, registration)
│   ├── productController.js  # CRUD operations for products
│   ├── orderController.js    # Order processing logic
│   └── userController.js     # User profile management
│
├── /models               # Mongoose models for MongoDB collections
│   ├── userModel.js          # User schema and model
│   ├── productModel.js       # Product schema and model
│   ├── orderModel.js         # Order schema and model
│   └── cartModel.js          # Shopping cart schema and model
│
├── /routes               # Express routes (API endpoints)
│   ├── authRoutes.js         # Routes related to user authentication
│   ├── productRoutes.js      # Routes for product management
│   ├── orderRoutes.js        # Routes for handling orders
│   └── userRoutes.js         # Routes for user management (profile, settings)
├── /middleware           # Middleware functions (authentication, validation)
│   ├── authMiddleware.js     # JWT authentication middleware
│   ├── errorMiddleware.js    # Custom error handling middleware
│   └── validationMiddleware.js # Input validation middleware
│
├── /services             # Business logic and external API integrations
│   ├── paymentService.js     # Payment processing (e.g., Stripe integration)
│   └── emailService.js       # Email notifications (e.g., order confirmation)
├── /utils                # Utility functions and helpers
│   ├── generateToken.js      # Utility to generate JWT tokens
│   └── hashPassword.js       # Utility to hash passwords using bcrypt
```

```
├── server.js            # Main entry point for the backend server
├── .env                 # Environment variables (e.g., database URI, JWT secret)
├── package.json         # Project dependencies, scripts
└── README.md            # Project documentation
```

6.RUNNING THE APPLICATION:

Frontend Setup (React):

1. Navigate to the `client` folder and install dependencies using `npm install`.
2. Start the React development server with `npm start`, which will run the frontend on `http://localhost:3000`

Backend Setup (Node.js/Express):

1. Navigate to the `server` folder and install dependencies using `npm install`.
2. Ensure you have a `.env` file with necessary environment variables (e.g., MongoDB URI, JWT secret).
3. Start the backend server with `npm start`, which will run on http://localhost:5000.

7.API DOCUMENTATION:

The **Shopez E-Commerce Application** provides a RESTful API for handling essential operations like user authentication, product management, order processing, and cart management. Below is a brief overview of the key API endpoints:

1. User Authentication

- **Register (`POST /api/auth/register`)**: Allows new users to register by providing an email, password, and optional role (e.g., customer, admin).
- **Login (`POST /api/auth/login`)**: Authenticates users and generates a JWT token for secure access to protected routes.
- **Current User (`GET /api/auth/me`)**: Retrieves information about the currently authenticated user.

2. Products

- **Get All Products (`GET /api/products`)**: Fetches a list of all products, with optional filters like category and search query.
- **Add Product (`POST /api/products`)**: Allows an admin to add a new product with details like name, description, price, and stock.
- **Get Product Details (`GET /api/products/:id`)**: Fetches detailed information about a specific product.
- **Update Product (`PUT /api/products/:id`)**: Allows admins to update product details.
- **Delete Product (`DELETE /api/products/:id`)**: Removes a product from the database (Admin only).

3. Orders

- **Place Order (`POST /api/orders`)**: Allows a user to place an order by providing items, shipping address, and payment method.
- **Get Order Details (`GET /api/orders/:id`)**: Retrieves details about a specific order.
- **Get All Orders (`GET /api/orders`)**: Allows admins to view all orders placed by customers.

4. Cart

- **Get Cart (`GET /api/cart`)**: Retrieves the current user's shopping cart, including items and total price.
- **Add Item to Cart (`POST /api/cart`)**: Adds a specific product and quantity to the cart.
- **Remove Item from Cart (`DELETE /api/cart/:id`)**: Removes a specific item from the cart.

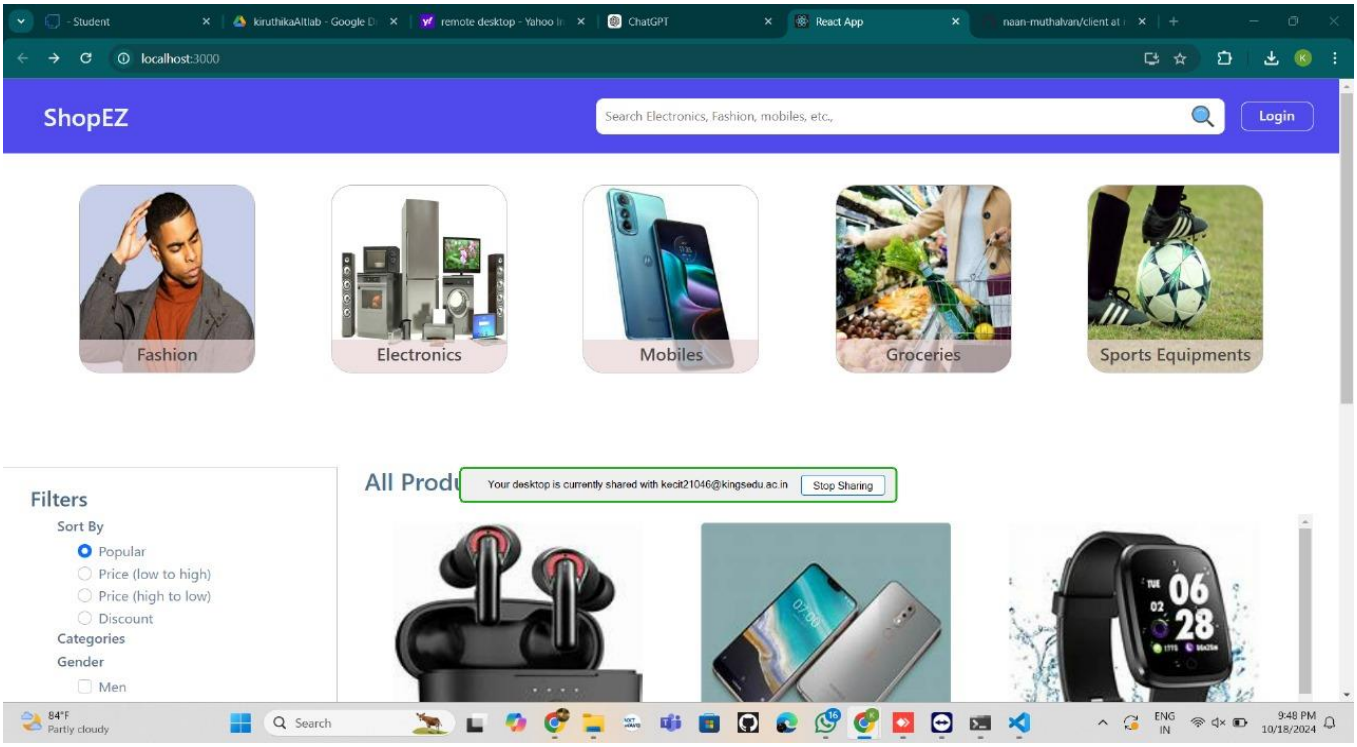
.

8.AUTHENTICATION :

- **User Registration**:
  Users sign up by providing their email, password, and optional role (e.g., customer or admin). The password is hashed for security.
  Upon successful registration, the user is created in the database.
- **User Login**:
  Existing users log in with their email and password.
  If the credentials are correct, a **JWT token** is generated and returned. This token is used for subsequent requests to authenticate the user.

- **Accessing Protected Routes**:
  To access protected routes (e.g., viewing the user's profile), the client includes the JWT token in the **Authorization** header.
  The server verifies the token, and if valid, grants access to the requested resource.
  Role-Based Access Control:
  Admins have special permissions (e.g., adding products, managing orders). A middleware checks the user's role to restrict access to certain routes based on the role.

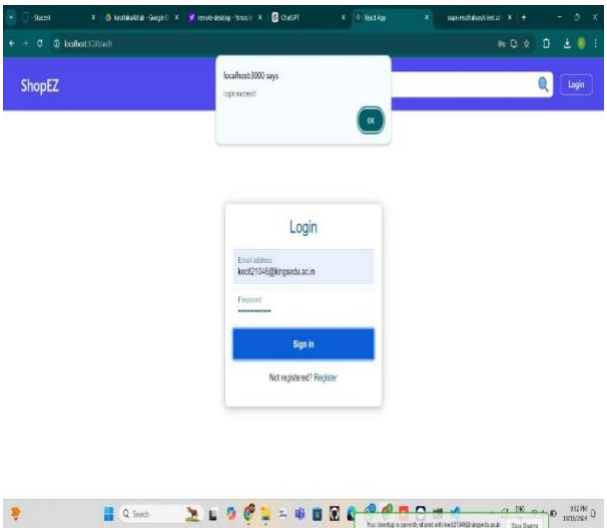
9.USER INTERFACE:

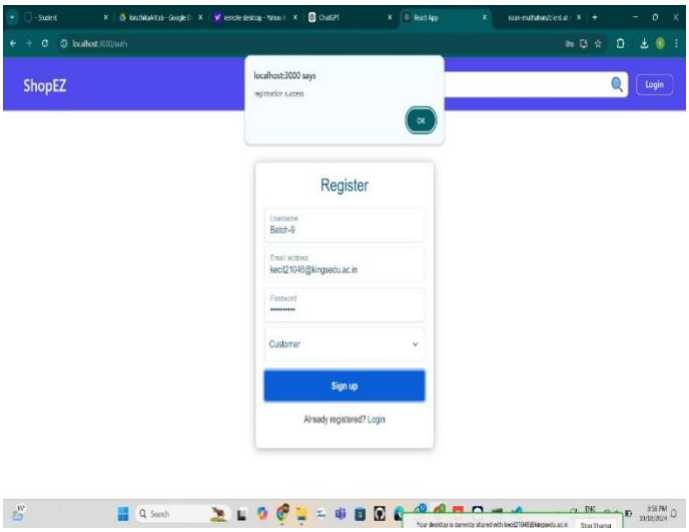


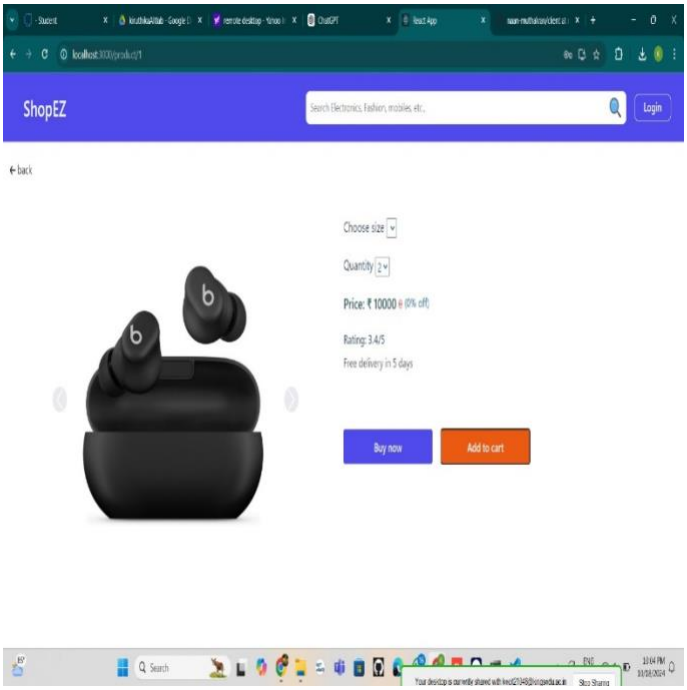
10.TESTING:

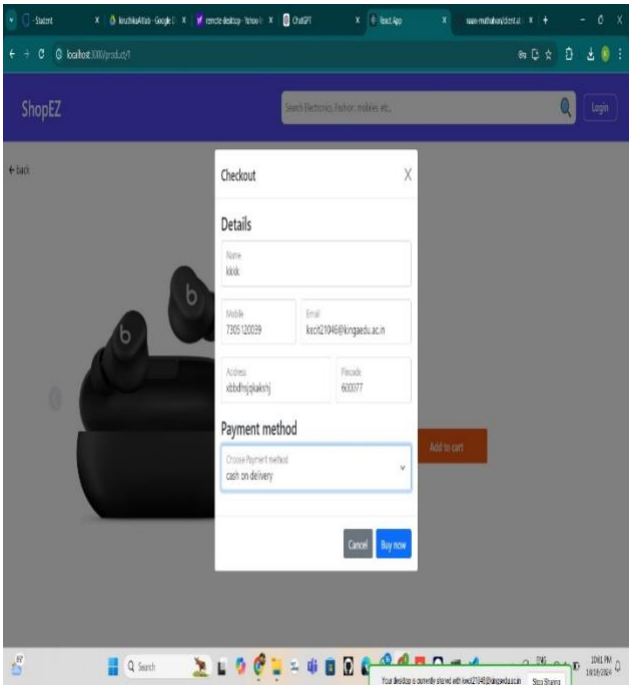
- **Unit testing** for individual components.
- **Integration testing** to ensure components interact as expected.
- **End-to-end testing** to simulate real-world user interactions.
- **API testing** to validate backend functionality.
- **Performance testing** to ensure scalability.
- **Security testing** to protect against common vulnerabilities.

11.SCREENSHOTS OR DEMO:









12.KNOWN ISSUES:

- **Payment Gateway Delays**: Occasional delays or failures during checkout due to thirdparty API issues.
- **Mobile Responsiveness**: Some pages, particularly product details and checkout, have layout issues on smaller screens.
- **Performance**: Slow page load times when displaying large product catalogs (over 100 items).
- **Email Delays**: Order confirmation emails are sometimes delayed due to integration issues with the email service.

13.FUTURE ENHANCEMENT:

- **Advanced Search and Filters**: Implement more sophisticated search functionality with advanced filters like price range, ratings, and specific product attributes to improve user experience.
- **Multi-Currency and Multi-Language Support**: Add support for multiple currencies and languages to cater to a global audience, allowing users to switch between different regions easily.
- **Personalized Recommendations**: Introduce AI-driven product recommendations based on user behavior and preferences to enhance shopping experience and increase sales.