**E-COMMERCE APPLICATION MERN STACK**

**1.Introduction:**

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

1. Nivetha S - Frontend developer
2. Sandhiya - Backend developer
3. Sangeetha - Database
4. Tamilselvan

**2. Project Overview**

The E-commerce MERN Stack Application is a web-based platform that enables users to browse, search, and purchase products online. Built with MongoDB, Express.js, React, and Node.js, it provides a seamless shopping experience with a scalable and modern architecture.

Scope: The platform is designed for both customers and admins. Customers can browse products, manage their cart, and place orders, while admins can manage product inventories and monitor sales. The project aims to deliver a responsive, secure, and user-friendly e-commerce solution.

**Features:**

* User Authentication: Secure login and registration.
* Product Management: View, search, and filter products.
* Shopping Cart: Add, update, and remove items.
* Order Management: Place orders and track history.
* Admin Dashboard: Manage products and view analytics.

**3. Architecture**

In the architecture section, you describe the overall structure of the system, explaining how the frontend, backend, and database interact:

* Frontend (React): This part of the application handles the user interface and communicates with the backend via HTTP requests *(using Axios or Fetch API).*
* Backend (Node.js/Express): The backend manages business logic, API requests, and user data. It acts as the intermediary between the frontend and the database.
* Database (MongoDB): MongoDB stores user data, job posts, messages, and other relevant information. The database is queried by the backend to retrieve and save data.

Additionally, mention the client-server model, where the frontend makes requests to the backend, which interacts with the database.

**4. Setup Instructions**

This section provides a step-by-step guide on how to set up the project locally. The goal is to help users get the app running in their own development environment.

* Clone the repository: The first step is to clone the project repository from a version control system (e.g., GitHub).
* Install dependencies: Use `npm install` to install necessary packages for both the frontend (React) and backend (Node.js/Express).
* Set up environment variables: Set up any required configuration files (e.g., MongoDB URI, JWT secret, etc.).
* Run the application: Run the frontend using `npm start` and the backend using `npm start` or `nodemon`.

**5. Folder Structure**

Here, you outline the organization of your codebase to give readers a clear understanding of where different components are located. Typical folder structure might look like this:

**Frontend:**

* ‘src/`: Contains React components, pages, and hooks.
* ‘public/`: Stores static assets like images and fonts.
* `App.js`: Main React entry point.

**Backend**:

* `models/`: Contains Mongoose schemas for MongoDB collections (e.g., `User`, `Job`, `Message`).
* `controllers/`: Holds business logic that interacts with the database (CRUD operations).
* `routes/`: API routes to handle client requests.
* `middleware/`: Functions that run before or after the main route handler (e.g., authentication check).
* `config/`: Database connection and environment settings.

**6. Running the Application**

In this section, explain how to run the application locally or deploy it to a production environment.

**Development:**

* Run the backend with `npm start` or `nodemon` for live-reloading.
* Start the React app with `npm start` to launch the frontend in development mode.

**Production:**

* For the production environment, build the React app using `npm run build` and serve it with a web server like “Nginx”.
* Host the backend on cloud platforms like “Heroku” or ”AWS”.
* Ensure the database (MongoDB) is either self-hosted or uses a cloud-based service like MongoDB Atlas.

**7.API Documentation:**

* **User Authentication**
  + POST /api/auth/register: Register a new user (requires name, email, and password).
  + POST /api/auth/login: Login user (requires email and password, returns a token).
* **Products**
  + GET /api/products: Fetch all products.
  + GET /api/products/

Get details of a specific product by id:

* + POST /api/products: Add a new product (admin-only, requires authentication).
* **Cart**
  + GET /api/cart: Get user's cart details (requires authentication).
  + POST /api/cart: Add items to the cart (requires product id and quantity).
* **Orders**
  + GET /api/orders: Fetch all user orders (requires authentication).
  + POST /api/orders: Place an order (requires cart id).

**8. Authentication**

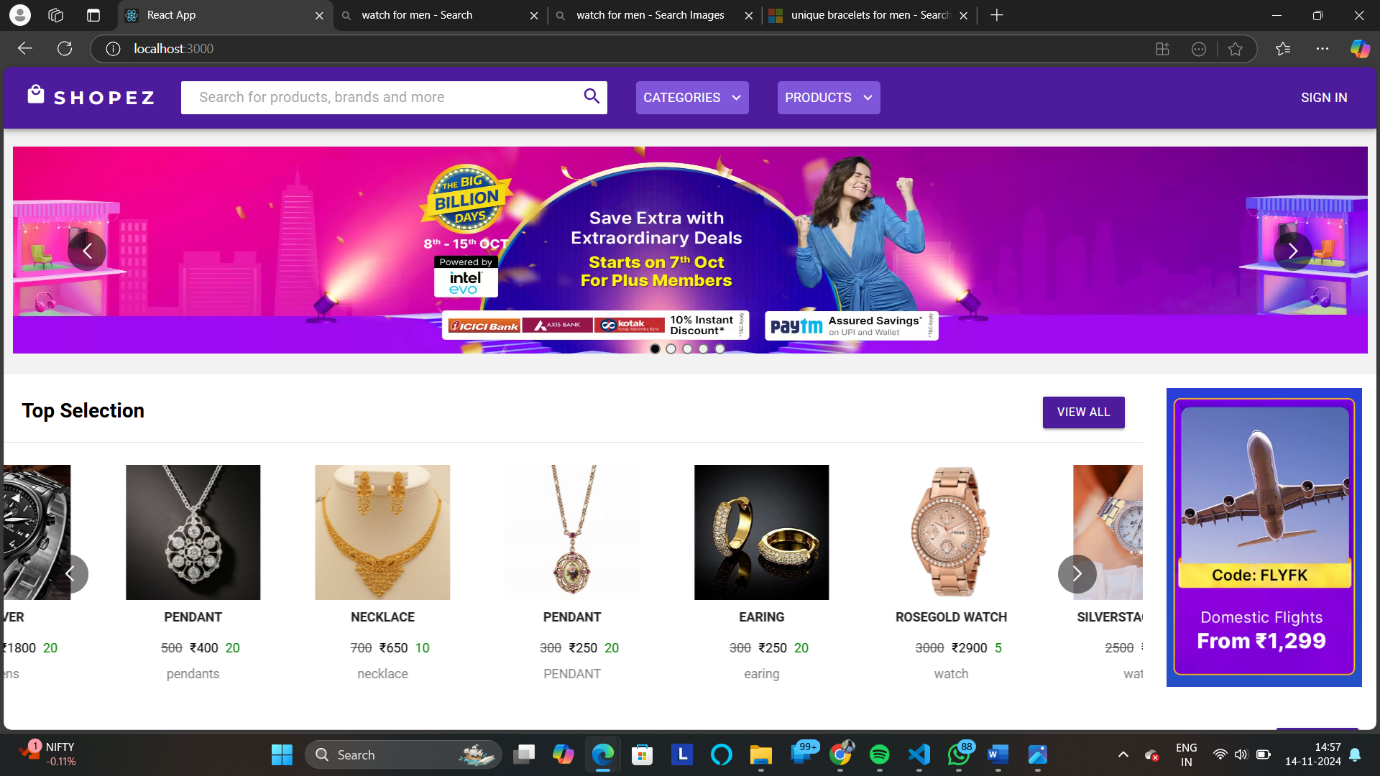
In this section, you explain how “authentication” works in the app:

* JWT Authentication: Users sign up and log in using their credentials. On login, they receive a JSON Web Token (JWT) that is stored on the client side (typically in local storage or cookies).
* Authorization: The JWT is passed in the header of each API request to verify that the user is authenticated.
* Role-based Access Control: The app may have different roles (e.g., freelancer, client, admin), and users are only allowed to access certain resources based on their role.

**9.User Interface:**

The E-commerce MERN App **UI** is designed for simplicity and usability, ensuring a smooth experience for both customers and admins.

* Home Page: Displays featured products, categories, and a search bar.
* Product Page: Lists products with sorting, filtering, and quick view options.
* Cart Page: Shows selected items with options to update quantities or remove items.
* Checkout Page: Secure order summary and payment process.
* Admin Dashboard: Intuitive interface for managing products, orders, and analytics.



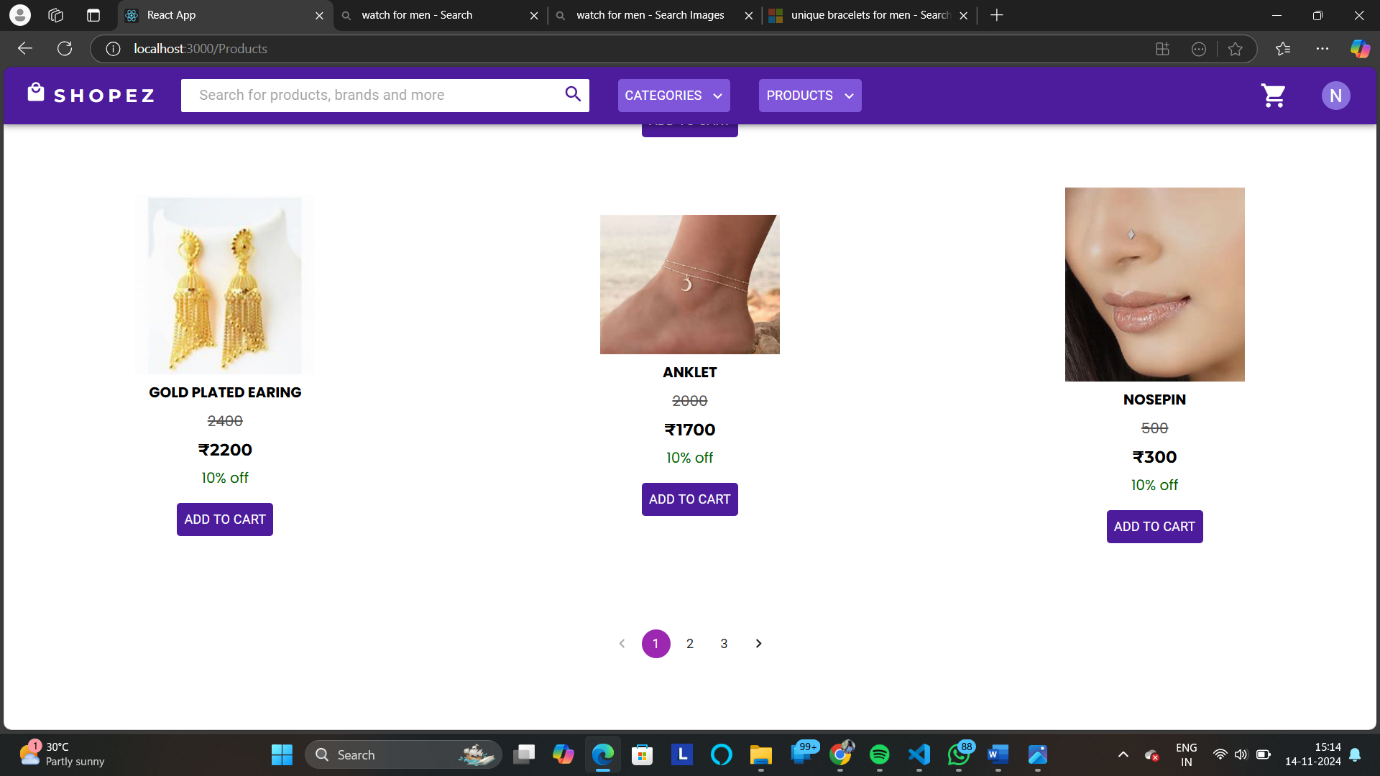
**10.Testing:**

In the “**testing**\*” section, describe how you ensure the application works as expected:

* “Unit Testing”: Testing individual components or backend logic using tools like “*Jest* “ or “*Mocha*”
* Integration Testing: Testing interactions between different parts of the system, such as the backend and frontend, to ensure data is being passed correctly.
* End-to-End Testing: Testing the entire user journey, using tools like “Cypress” or “Puppeteer.

**11.Screenshots/Demo:**

* Home Page: Displays featured products, a search bar, and categories for browsing.
* Product Details Page: Shows product images, descriptions, pricing, and an "Add to Cart" button.
* Cart Page: Summarizes selected items with quantity and price details, plus checkout options.
* Checkout Page: Secure form for entering payment and shipping details, with order confirmation.
* Admin Dashboard: Features product management, order tracking, and sales analytics.



**12. Known Issues**

List any “known issues” that users might encounter. These could include:

* Bugs that haven’t been resolved yet.
* Incomplete features or limited functionality.
* UI issues (e.g., responsiveness problems, broken elements).
* Performance issues under heavy load.

Be transparent about what still needs attention.

**13.FUTURE ENHANCEMENTS:**

* Real-Time Notifications:Implement push notifications for order updates, promotions, and important events using WebSocket or Firebase.
* Enhanced Search Functionality:Add advanced filters, sorting, and full-text search capabilities to improve product discovery.
* Multi-Language Support:Introduce localization to cater to a diverse, global user base.
* AI-Powered Recommendations:Use machine learning to provide personalized product recommendations based on user behavior.
* Progressive Web App (PWA):Transform the application into a PWA for better performance, offline capabilities, and mobile accessibility.