Senior React Developer Test

Objective: Develop a performance-optimized dashboard application with real-time chat functionality and role-based authentication.

Project Requirements

1. Project Setup and Initial Configuration:

- Use **Create React App** (or Vite for faster builds) to set up the project.
- Structure the application using a modular approach with separate directories for components, services, and utilities.
- Use **Material-UI (MUI)** or **Ant Design** for UI components to ensure a professional and responsive design.

2. Authentication and Authorization:

- Implement a login system using **JWT** for authentication.
- Create a `Login` component with form validation using **React Hook Form** or **Formik**.
- Build an authentication service to manage login, logout, and token storage using `localStorage` or `sessionStorage`.
- Implement route protection using **React Router**'s `ProtectedRoute` based on authentication status and user roles (e.g., admin, user).
- Mock different user roles and permissions for testing.

3. Real-Time Chat:

- Integrate a **WebSocket** service for real-time chat functionality. Use libraries like **socket.io-client**.
- Create a `Chat` component that allows users to send and receive messages in real-time.
- Implement group chats and individual (private) chats.
- Add the ability to send images or file attachments in the chat.
- Connect to a mock WebSocket server to simulate real-time chat.

4. Performance Optimization:

- Analyze the initial performance using **React Profiler** or **Lighthouse**.
- Implement **code-splitting** using **React.lazy** and **Suspense** for lazy loading components.
- Optimize **React's re-rendering** behavior using the **useMemo**, **useCallback**, and `React.PureComponent` where necessary.
- Ensure the application uses **tree shaking** to eliminate unused code.
- Optimize images and other assets using tools like **webpack** or **Vite**'s built-in optimization tools.

5. Error Handling and User Feedback:

- Implement global HTTP error handling using **Axios interceptors** or a similar method.
- Provide user feedback using **Snackbars** or **Toast notifications** for actions like sending messages, updates, and errors.

6. Testing:

- Write **unit tests** for the authentication service, WebSocket service, and key components using **Jest** and **React Testing Library**.
- Perform **end-to-end (e2e) testing** using **Cypress** to ensure the application flows work correctly.
- Test the application's performance before and after optimization to confirm improvements.

Detailed Task Breakdown

1. Authentication Service:

- Create an `AuthService` to handle authentication logic (login, logout, token management).
- Use **JWT** for authentication and securely store the token in `localStorage` or `sessionStorage`.
- Implement methods to check authentication status and user roles.

2. Login Component:

- Create a `Login` component with **React Hook Form** or **Formik** for user input and validation.
- Handle authentication errors and provide user feedback using notifications.

3. Route Guards:

- Implement **ProtectedRoute** components to protect routes based on authentication status.
- Apply role-based protection using mock data for different user roles (e.g., admin, user).

4. WebSocket Service for Chat:

- Create a `ChatService` to manage WebSocket connections.
- Implement methods to send and receive chat messages.
- Use **React Context API** or **Redux** to manage chat state globally.

5. Chat Component:

- Create a `Chat` component to display and send real-time chat messages.
- Support group chats and private chats, with notifications for new messages.
- Implement image or file attachment functionality within the chat.

6. Performance Optimization:

- Implement **lazy loading** and **code-splitting** using **React.lazy** and **Suspense**.
- Optimize **React rendering** with **memoization** (`useMemo`, `useCallback`).
- Use **webpack** or **Vite** to optimize images and other assets.
- Verify performance improvements with **React Profiler** and **Lighthouse**.

7. Error Handling and User Feedback:

- Implement global HTTP error handling with **Axios interceptors**.
- Provide feedback for chat messages and errors using **Snackbars** or **Toast notifications**.

8. Testing and Documentation:

- Write comprehensive **unit tests** using **Jest** and **React Testing Library**.
- Perform **e2e testing** using **Cypress** for complete application functionality.
- Document the code and provide instructions for running the application and tests.

Expected Deliverables

1. **Source Code** in a Git repository with clear commit history.
2. A **README** file with setup and run instructions.
3. **Documentation** for the authentication and WebSocket services.
4. **Test results** from unit and e2e tests.
5. **Performance report** comparing initial and optimized states.

Estimated Time to Complete

1. **Project Setup and Initial Configuration:** 0.5 day
2. **Authentication and Authorization:** 1 days
3. **Real-Time Chat:** 1 days
4. **Performance Optimization:** 1 day
5. **Error Handling and User Feedback:** 0.5 day
6. **Testing and Documentation:** 0.5–1 day

Total Estimated Time: 4–5 days

Tips to Complete in 3-4 Days

- **Parallel Development:** Work on authentication, real-time chat, and performance optimization in parallel.
- **Prioritize Core Features:** Focus on core functionality first (authentication, chat, performance), and add enhancements later if time permits.
- **Use Well-Documented Libraries:** Leverage **React libraries** like **Material-UI**, **socket.io-client**, and **React Hook Form** to accelerate development.
- **Daily Milestones:** Set clear objectives and milestones for each day to track progress.