Assignment: React Authentication with JWT (Access + Refresh)

Overview

Build a React single-page application that implements secure authentication using **JWT access tokens** and **refresh tokens**. The app should use **Axios** for HTTP requests, **React Query** for managing server state, and **React Hook Form** for handling form input and validation. You will design a complete client-side authentication flow that includes login, token storage, token refresh, logout, and public hosting.

Objectives

- Understand and differentiate access tokens and refresh tokens.
- Configure Axios to attach access tokens and refresh them when expired.
- Use React Query to manage authentication and data fetching.
- Use React Hook Form for login and form input validation.
- Implement protected routes that require valid authentication.
- Handle logout and token invalidation correctly.
- Host the finished application on a public hosting platform.

Requirements

1. **Authentication Flow**
    - The system must include a login and logout mechanism.
    - Upon successful login, the server will return an access token and refresh token.
    - The access token should be used for all authorized API requests.
    - When the access token expires, the refresh token must be used to obtain a new one automatically.
2. **Token Management**
    - The access token should be stored in memory during the session.
    - The refresh token should be stored in persistent storage (e.g., `localStorage`).
    - On logout, all tokens must be cleared.
3. **Axios Configuration**
    - Create an Axios instance for API communication.

    - Attach the access token to every request's `Authorization` header.
    - Handle `401 Unauthorized` responses by refreshing the access token using the refresh token.
    - If the refresh fails (e.g., expired refresh token), log the user out and redirect to the login page.
4. **React Query Integration**
    - Use React Query to manage API calls and authentication mutations.
    - Use `useMutation` for login and logout actions.
    - Use `useQuery` for fetching user data from protected endpoints.
    - Invalidate or refetch queries as needed when the authentication state changes.
5. **React Hook Form Integration**
    - Use React Hook Form to manage the login form.
    - Validate user input fields such as email and password before submission.
    - Display error messages for missing or invalid input.
    - Integrate form submission with the login mutation from React Query.
6. **Protected Routes**
    - Implement protected routes that only allow access if a valid access token exists.
    - Redirect unauthenticated users to the login page.
7. **User Interface**
    - Create a login page with email and password fields managed by React Hook Form.
    - Display user information on a protected dashboard after successful login.
    - Include a logout button that clears tokens and redirects to the login page.
8. **Public Hosting**
    - Deploy the application to a public hosting platform (e.g., Netlify, Vercel, GitHub Pages, or Firebase Hosting).
    - Ensure that all protected routes and API calls function correctly in the hosted version.
    - Provide the public URL in the README file.
9. **Error Handling**
    - Display meaningful error messages for failed login, expired tokens, or network issues.
    - Handle refresh token expiration gracefully by logging out automatically.

Deliverables

- A React application implementing the described authentication flow.
- The app hosted publicly and accessible via a shared URL.
- A short README file describing how to set up, run, and access the project.
- Optional: Include a simple backend or mock API to simulate the login, refresh, and protected data endpoints.

Evaluation Criteria

- **Authentication logic and correctness (30%)**
  Access and refresh token handling is implemented correctly.
- **Axios interceptor setup (20%)**
  Proper request and response interception with automatic token refresh.
- **React Query integration (15%)**
  Authentication and data fetching use React Query appropriately.
- **React Hook Form integration (10%)**
  Login form is implemented using React Hook Form with proper validation.
- **Public hosting and deployment (10%)**
  Application is deployed and accessible on a public hosting platform.
- **UI and UX (10%)**
  Functional and clear user interface for login, logout, and dashboard.
- **Error handling and code organization (5%)**
  Robust error management and clean, modular code structure.

Stretch Goals (Optional)

- Implement silent token refresh before expiration.
- Use cookies for refresh token storage instead of localStorage.
- Add support for multi-tab synchronization (logout reflects across tabs).
- Include role-based access control for specific routes.

Goal: Demonstrate a secure, maintainable, and deployable pattern for JWT-based authentication in React applications using Axios, React Query, and React Hook Form.