17. Authenticating React Frontend

What

Add user authentication to secure your React app. This involves protecting specific routes or features based on the user's login status.

Why

Authentication ensures that only authorized users can access certain parts of your app, safeguarding sensitive data and functionality.

How

1. **Install Dependencies**
   - Use react-router-dom for navigation and context to manage authentication status.

Run in terminal - npm install react-router-dom

2. **Create Authentication Context**

```
import React, { createContext, useContext, useState } from 'react';
const AuthContext = createContext();
export function AuthProvider({ children }) {
const [isAuthenticated, setIsAuthenticated] = useState(false);

const login = () => setIsAuthenticated(true);
const logout = () => setIsAuthenticated(false);
return (
<AuthContext.Provider value={{ isAuthenticated, login, logout }}>
{children}
</AuthContext.Provider>
);
}

export function useAuth() {
return useContext(AuthContext);
}
```

3.Protect Routes

```
import { Navigate } from 'react-router-dom';
import { useAuth } from './AuthProvider';

function ProtectedRoute({ children }) {
  const { isAuthenticated } = useAuth();

  return isAuthenticated ? children : <Navigate to="/login" />;
}

export default ProtectedRoute;
```

3.Integrate Authentication

Wrap your app in AuthProvider and use ProtectedRoute to secure pages.

```
import { BrowserRouter as Router, Route, Routes } from 'react-router-dom';
import { AuthProvider, useAuth } from './AuthProvider';
import ProtectedRoute from './ProtectedRoute';
function App() {
  const { login, logout } = useAuth();
  return (
    <AuthProvider>
      <Router>
        <Routes>
          <Route path="/login" element={<LoginPage />} />
          <Route
            path="/dashboard"
            element={
              <ProtectedRoute>
                <Dashboard />
              </ProtectedRoute>
            }
          />
        </Routes>
        <button onClick={login}>Login</button>
        <button onClick={logout}>Logout</button>
      </Router>
    </AuthProvider>
  );
}

export default A
```

---

18. Tailwind and Daisy UI Integration

What

Use **Tailwind CSS** and **Daisy UI** to style your React app efficiently. These tools provide pre-designed utility classes and components for responsive, clean, and professional UI designs.

Why

- Speeds up the design process with ready-to-use classes.
- Makes components more visually appealing without much effort.

How

1. **Install Tailwind CSS**

```
npm install -D tailwindcss postcss autoprefixer
npx tailwindcss init
```

2. **Configure Tailwind**
   - Update tailwind.config.js to include paths to your React files.

```
module.exports = {
  content: ['./src/**/*.{js,jsx,ts,tsx}'],
  theme: {
    extend: {},
  },
  plugins: [],
};
```

3. **Install Daisy UI**

```
npm install daisyui
```

4. **Add Daisy UI to Tailwind Config**

```
module.exports = {
  plugins: [require('daisyui')],
};
```

5. **Use Tailwind and Daisy UI Classes**

Example: A styled button.

```
export default function App() {
  return (
    <button className="btn btn-primary">
      Click Me
    </button>
  );
}
```

---

19. Fetch API with Daisy UI

What

Combine API data fetching with Daisy UI components to build visually appealing, functional UIs.

Why

Helps practice integrating real-world APIs while leveraging Daisy UI for better design.

How

1. **Fetch Data**
    - Use fetch or Axios to get API data.

```
import { useEffect, useState } from 'react';
function App() {
  const [data, setData] = useState([]);
  const [loading, setLoading] = useState(true);

  useEffect(() => {
    fetch('https://jsonplaceholder.typicode.com/posts')
      .then((response) => response.json())
      .then((data) => {
        setData(data);
        setLoading(false);
      });
  }, []);

  return (
    <div>
      {loading ? <Spinner /> : data.map((item) => <Card key={item.id} {...item} />)}
    </div>
  );
}
export default App;
```

2. **Use Daisy UI Components**
    - Example: Display fetched data in styled cards.

```
function Card({ title, body }) {
  return (
    <div className="card w-96 bg-base-100 shadow-xl">
      <div className="card-body">
        <h2 className="card-title">{title}</h2>
        <p>{body}</p>
      </div>
    </div>
  );
}
```

---

20. Deployment on Netlify

What

Deploy your React app to **Netlify**, a popular and beginner-friendly platform for hosting modern web applications.

Why

Hosting allows you to share your app with the world and test it in a real environment.

How

1. **Prepare Your Project**
    - Ensure your app is working correctly by running:

      ```
      npm run build
      ```
2. **Sign Up for Netlify**
    - Create a free account at Netlify.
3. **Deploy Your App**
    - Drag and drop the dist or build folder into Netlify's deployment area.
4. **Connect GitHub Repository**
    - Alternatively, link your GitHub repo to Netlify for automatic deployments on changes.
5. **Access Your Site**
    - Netlify will provide a live URL where your app is hosted.