CipherSchools

Class 8 ReactJS

React Router

What is React Router?

React Router is a standard library for routing in React. It enables navigation among views of various components, allows changing the browser URL, and keeps the UI in sync with the URL.

Why React Router?

- Single Page Applications (SPAs) don't reload the page.
- We need a way to change views/components based on URL changes without refreshing the whole page.
- React Router lets us manage routing declaratively.

Installation

```
npm install react-router
```

Use version 7+ as it's the latest and comes with major improvements.

Key Concepts

<BrowserRouter>

- Wraps your entire app.
- Uses HTML5 history API for cleaner URLs (example.com/page) instead of hash (example.com/#/page).

```
import { BrowserRouter } from "react-router-dom";
```

```
<BrowserRouter>
  <App />
</BrowserRouter>
```

<Routes> and <Route>

- Routes is a container for Route.
- Each Route maps a path to a component.

```
import { Routes, Route } from "react-router-dom";

<Routes>
  <Route path="/" element={<Home />} />
  <Route path="/about" element={<About />} />
</Routes>
```

Nested Routes

```
<Routes>
  <Route path="/dashboard">
    <Route path="profile" element={<Profile />} />
    <Route path="settings" element={<Settings />} />
  </Route>
</Routes>
```

NavLink (for Navigation)

```
import { NavLink } from "react-router-dom";

<NavLink to="/" style={({ isActive }) => ({ color: isActive ? "red" : "black" })}>
  Home
</NavLink>
```

NavLink is like Link but adds an active state by default.

useNavigate (Programmatic Navigation)

```
import { useNavigate } from "react-router-dom";
const navigate = useNavigate();

const goToProfile = () ⇒ {
  navigate("/profile");
};
```

Dynamic Routing (Route Params)

```
<Route path="/users/:id" element={<User />} />
```

To get id:

```
import { useParams } from "react-router-dom";
const { id } = useParams();
```

Example App Structure

```
<BrowserRouter>
  <Routes>
    <Route path="/" element={<Home />} />
    <Route path="/about" element={<About />} />
    <Route path="/profile/:id" element={<Profile />} />
    <Route path="/dashboard" element={<Dashboard />}>
      <Route path="settings" element={<Settings />} />
    </Route>
  </Routes>
</BrowserRouter>
```

Diagram (Textual Version)

```
<BrowserRouter>
  └── <Routes>
      ├── Route path="/" → <Home />
      ├── Route path="/about" → <About />
      ├── Route path="/profile/:id" → <Profile />
```

└── Route path="/dashboard" → <Dashboard />
├── Nested Route path="settings" → <Settings />

What is a Layout in React Router?

In web apps, you often want to define a common layout (like a header, sidebar, or footer) that wraps all your pages. React Router enables this using nested routes and layouts.

Layout is a React component that:

- Defines shared UI (e.g., navbar, sidebar).
- Renders children routes via <Outlet />.

What is <Outlet>?

- <Outlet /> is a placeholder used inside a layout component.
- It renders the nested route's element.
- Think of it like {children} but for nested routing.

Why use Layout and <Outlet>?

- To avoid repeating UI code (header/sidebar).
- Helps structure large apps better.
- Encourages component reusability and cleaner nesting.

Examples

```
import React from "react";
import { Outlet } from "react-router";
import Navbar from "../components/navbar";
import Footer from "../components/footer";

function Dashboard() {
  return (
    <main style={{ height: "100vh", padding: 0, margin: 0 }}>
      {/* Navbar */}
      <Navbar />
      {/* Nested Routes will render here */}
      <Outlet />
      {/* Footer */}
      <Footer />
    </main>
  );
}

export default Dashboard;
```

Task:

Create a Login & Signup Flow

1. Login Page
   Fields:
   - Email
   - Password

2. Signup Page
   Fields:
   - Name
   - Email
   - Password
   - Confirm Password

Data Handling

1. On Successful Signup:
   - Store the user in a users array in localStorage.
   - User object structure:
     ```
     {
       "id": Number,
       "name": String,
       "email": String,
       "password": String,
       "createdAt": Timestamp
     }
     ```
2. On Login:

    - Search for the user in the users array from localStorage.
    - Authenticate based on the email and password.
    - On success, store the user object in localStorage as loggedInUser.

Post Login Behavior

- Display the logged-in user's name in the navbar.
- On Logout, remove the loggedInUser entry from localStorage.