Week 8: React Router and Single Page Application (SPA) Navigation

Day 1: Routing Fundamentals and the SPA Concept

Theoretical Overview

Traditional web applications reload the entire page every time a user clicks a link. In a Single Page Application (SPA), the browser stays on a single HTML page, and React dynamically updates the content by swapping components. This provides a desktop-like experience with no page flickering.

The react-router-dom Library

React does not have a built-in router. We use the react-router-dom library to manage navigation. This library synchronizes the UI with the URL in the address bar.

- **Router vs. Browser Router**: BrowserRouter uses the HTML5 history API to keep your UI in sync with the URL.
- **The Route Component**: This is the most basic component; it renders a specific UI when its path matches the current URL.
- **Switch/Routes**: This component wraps multiple routes and ensures that only the first matching route is rendered at a time.

Day 2: Declarative Navigation with Links

Navigating without Page Reloads

To navigate between pages in React, we cannot use standard HTML <a> tags, as they trigger a full browser refresh.

- **The <Link> Component**: This allows users to navigate to different parts of the application while keeping the state intact.
- **The <NavLink> Component**: A special version of the <Link> that knows whether it is "active," allowing you to style the current page link in the navigation bar automatically.
- **Redirects**: Used to send users to a specific page (like a Login page) if they try to access a protected route without authorization.

Day 3: Dynamic Routing and Route Parameters

Handling Variable Data in URLs

In applications like the **GadgetShop**, you don't create a separate route for every product. Instead, you use dynamic routing.

- **Route Parameters**: Using a colon syntax (e.g., path="/product/:id") allows the route to match any ID.
- **Retrieving Parameters**: Parameters can be accessed using the useParams hook. This allows the component to fetch the correct data from the **Gadget API** based on the ID in the URL.

Query Strings

Sometimes data needs to be passed via query parameters (e.g., ?search=phone).

- **useLocation Hook**: Used to access the current URL location object.
- **useSearchParams**: A modern hook used to read and modify query strings efficiently.

Day 4: Nested Routes and Layout Management

Hierarchical Navigation

Complex applications require nested routes. For example, a "Course Dashboard" might have its own sub-navigation for "Lessons," "Assignments," and "Grades."

- **Outlet**: A special component from React Router that marks the spot where child routes should be rendered within a parent layout.
- **Layout Components**: We create a wrapper component containing a Navbar and Sidebar that remains constant while the central content changes based on the nested route.

Day 5: Programmatic Navigation and Search

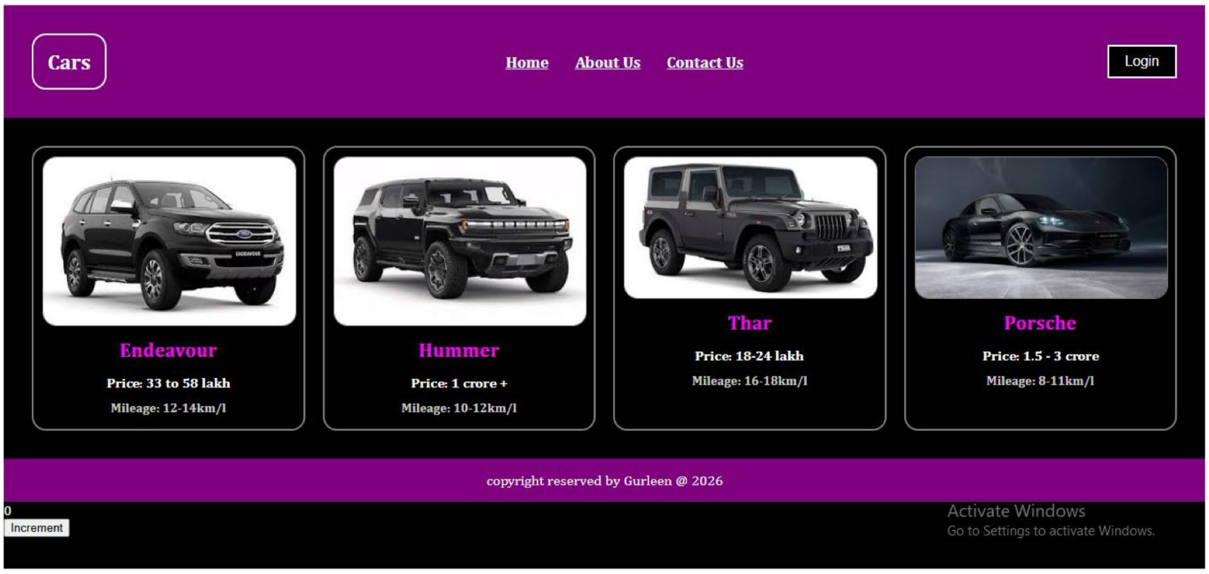
Navigating via Code

Sometimes navigation must happen automatically after a logic block completes (e.g., redirecting to the "Success" page after a user saves a contact in the **Phone Book** assignment).

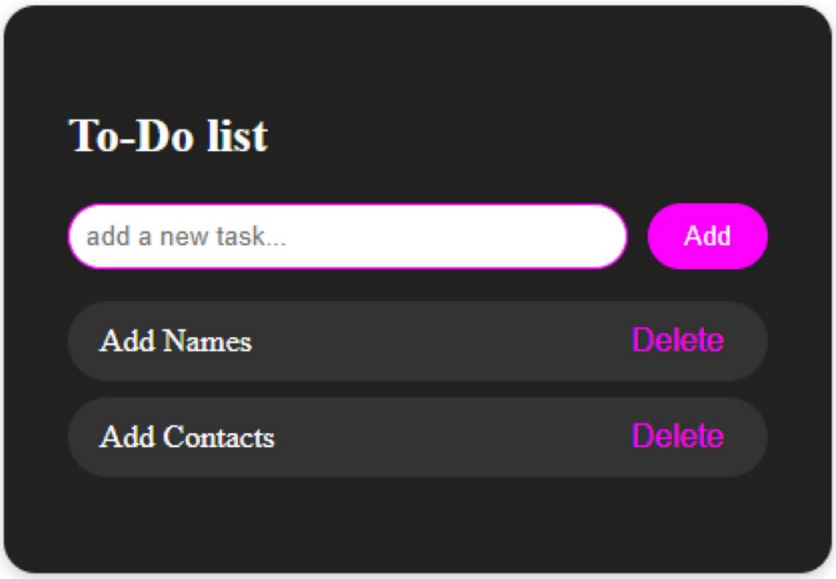
- **useNavigate Hook**: This hook returns a function that allows you to trigger navigation programmatically (e.g., Maps('/home')).
- **Replace vs. Push**: You can choose whether to add the new page to the browser history or replace the current entry to prevent the user from going back to a finished form.

Integration with Redux

In advanced MERN applications, the router is often synced with a global state library like Redux to ensure that the navigation state is consistent with the user's authentication and data status.



Task 1 Cards



Task 2 TodoList