SPA & Routing

CS568 – Web Application Development I

Computer Science Department

Maharishi International University

Maharishi International University - Fairfield, Iowa

Content

- SPA
- React Router

SPA (Single-page application)

A single-page application (SPA) is a web application or website that interacts with the user by **dynamically rewriting** the current web page with new data from the web server, instead of the default method of a web browser loading entire new pages. The goal is **faster transitions**.

Some of the most popular SPA frameworks include:

- React
- Angular
- Vue.JS

SPA (Single-page application)

In a SPA, a page refresh never occurs; instead, all necessary HTML, JavaScript, and CSS code is either

- retrieved by the browser with a single page load
- or the appropriate resources are dynamically loaded and added to the page as necessary, usually in response to user actions.

The HTML5 History API can be used to provide the perception and navigability of separate logical pages in the application. React Router is under the hood, making it possible by using these browser history API.

Web History API

The DOM Window object provides access to the browser's session through the history object. It exposes useful methods and properties that let you navigate back and forth through the user's history, and manipulate the contents of the history stack.

Some useful methods:

- window.history.back() or window.history.go(-1)
- window.history.forward() or window.history.go(1)
- window.history.go(0) or window.history.go() = refreshes the page

Routing & SPA

- Being able to show different pages to the user.
- SPA has a single HTML file. We do not have multiple HTML files.
- SPA apps still need URL or Route for loading different components for different path. **For example, you want to bookmark a url**.

React Router

- React Router is a collection of navigational components.
- Used in mobile and web apps.
- The newest version v6 embrace functional components and hooks.
- It is a third-party library, you need to install it: *npm install react-router-dom*
- Homepage: https://reactrouter.com/

BrowserRouter

- A <BrowserRouter> stores the current location in the browser's address bar using clean URLs and navigates using the browser's built-in history stack.
- It should be a top-level component of your application. Therefore, it can save the history of navigations. It contains a list of routes.

```
const router = createBrowserRouter([
{
path: "/",
element: <Home/>
}
])
function App() {
return (
<RouterProvider router={router}></RouterProvider>
);
}
```

Route

- A route is a combination of URL segment, Component, and data
- Two common ways to define routes
- Use objects to declare a route

```
const router = createBrowserRouter([
{
path: "/",
element: <Home/>
},
{
path: "/add",
element: <AddProduct/>
}
])
```

Route

- Use Route

```
const router = createBrowserRouter(
createRoutesFromElements([
<Route path="/" element={<Home />} />,
<Route path="/add" element={<AddProduct />} />
])
);
```

Route - Children

- Route can contain other routes as its children

```
const router = createBrowserRouter([
{
path: "/",
element: <Home/>,
children:[
{
path: "add",
element: <AddProduct/>
}
]
}
])
```

Route – useOutlet

- Outlet is used to load the child routes

```
export default function Home() {
let outlet = useOutlet();
return (
<div>
<p>Home</p>
<hr/>
{outlet}
</div>
);
}
```

Conventional Link: <a>

- To link to another page by refreshing the entire page

```
<div>
<p>Home</p>
<a href="/add">Add Product</a>
<hr></hr>
{outlet}
</div>
```

React Router Link: <Link>

- When we use <a> tag, it reloads the page. That is not what we want. We will use ‘Link’ to prevent that behavior.

- **<Link>** Provides declarative, accessible navigation around your single page application. When users click on the link, it will change the URL and the app will reflex on this change.

React Router Link: <Link>

```
<div>
<p>Home</p>
<Link to="/add">Add Product</Link>
<hr></hr>
{outlet}
</div>
```

useNavigate

- Use this hook to navigate programmatically

```
let outlet = useOutlet();
let navigate = useNavigate();
const navigateToAdd = () => {
navigate('/add');
}
return (
<div>
<p>Home</p>
<button onClick={navigateToAdd}>Add  Product</button>
<hr></hr>
{outlet}
</div>
);
```

Passing data between routes

- Location
- Params
- Search Params

Passing data between routes using location

- Create URL path

```
const router = createBrowserRouter([
{
path: "/",
element: <Home/>,
children:[
{
path: "add",
element: <AddProduct/>
}
]
}
])
```

Passing data between routes using location

- Produce the params

```
export default function Home() {
…
const navigateToAdd = () => {
navigate('/add', {state: {code: 2}});
}
return (
<div>
<p>Home</p>
<button onClick={navigateToAdd}>Add Product</button>
{outlet}
</div>
);
}
```

Passing data between routes using location

- Consume the search params: useSearchParams

```
export default function AddProduct(){
const location= useLocation();
return(
<p>Add Product {location.state.code}</p>
)
}
```

Passing data between routes using params

- Create URL path

```
const router = createBrowserRouter([
{
path: "/",
element: <Home/>,
children:[
{
path: "add/:code",
element: <AddProduct/>
}
]
}
])
```

Passing data between routes using param

- Produce the params

```
export default function Home() {
…
const navigateToAdd = () => {
navigate('/add/2');
}
return (
<div>
<p>Home</p>
<button onClick={navigateToAdd}>Add Product</button>
{outlet}
</div>
);
}
```

Passing data between routes using params

- Consume the params: useParams

```
export default function AddProduct(){
const params = useParams();
return(
<p>Add Product {params.code}</p>
)
}
```

Passing data between routes using Search Param

- Create URL path

```
const router = createBrowserRouter([
{
path: "/",
element: <Home/>,
children:[
{
path: "add",
element: <AddProduct/>
}
]
}
])
```

Passing data between routes using Search params

- Produce the params

```
export default function Home() {
…
const navigateToAdd = () => {
navigate('/add?code=2');
}
return (
<div>
<p>Home</p>
<button onClick={navigateToAdd}>Add Product</button>
{outlet}
</div>
);
}
```

Passing data between routes using params

- Consume the params: useParams

```
export default function AddProduct(){
const params = useSearchParams();
return(
<p>Add Product {params[0].get('code')}</p>
)
}
```

Error handling

- To handle unexpected URL segments, you can use errorElement

```
const router = createBrowserRouter([
{
path: "/",
element: <Home/>,
errorElement: <p>Sorry. Page Not Found</p>
}
])
```

Loader

- Will run before the rendering

```
createBrowserRouter([
  {
    element: <ProductList/>,
    path: "/list",
    loader: async () => {
      return db.getAllProducts();
    }
}])
```

useLoaderData

- To use the data returning from the loader function

```
export default function ProductList() {
  const albums = useLoaderData();
  // ...
}
```

Redirect

- To redirect to another link

```
const loader = async () => {
  const user = await getUser();
  if (!user) {
    return redirect("/login");
  }
  return null;
};
```

Summary

- Route: path, element, errorElement, children
- Hooks: useOutlet, useNavigate, useLocation, useParams, useSearchParams
- Error Handling
- Loader
- Redirect