React Router

Routing Mechanism

Routing in React is used to manage the URLs of the application and map them to different views or components that need to be displayed on the page.

In MPAs, each page has its own URL, and when the user navigates to a new page, the browser sends a request to the server, and the server responds with a new HTML page, which replaces the current page in the browser. The server determines which page to return based on the URL requested by the client. This process is known as server-side routing. This approach can be slower and less responsive, and it can lead to longer load times.

In contrast, in SPAs, the application is loaded once, and all the content is loaded dynamically without the need for page refreshes. Instead of loading new pages, the application updates the current view by manipulating the DOM. SPAs use client-side routing, which means that the routing is handled by the client-side JavaScript code. This process is known as client-side routing.This allows for faster and more responsive navigation, as the entire page does not need to be reloaded.

React Router

ReactJS Router is mainly used for developing Single Page Web Applications. React Router is used to define multiple routes in the application. A Route is used to define and render components based on the specified path. When the user navigates to a particular URL, React Router renders the component associated with that route.

For Example,

```
src/
    components/
      Header.js
      Footer.js
      Button.js
      Input.js
      ...
    pages/
      Home.js
      About.js
      Contact.js
      ...
    routes.js
    App.js
    index.js
```

In this example, the components folder contains reusable UI components that can be used across different pages. The pages folder contains the different views or pages of the application.

The routes.js file contains the route definitions for the application. This is where the <Route> components are defined, along with the path and the component to be rendered for each route.

Types of React Router

In React Router v6.4, there are different types of routers that can be used depending on the needs of the application:

- **<BrowserRouter> -** This is the most commonly used router and is designed for applications that will be hosted on a server that will serve all requests to the application. It uses HTML5 history API to keep the UI in sync with the URL.
- **<HashRouter> -** This router is designed for applications that will be hosted on a server that does not support HTML5 history API, such as GitHub Pages or static file servers. It uses the URL hash to keep the UI in sync with the URL.
- **<MemoryRouter> -** This router is designed for testing and non-visual use cases, such as server-side rendering or testing.

- **<NativeRouter> -** This router is designed for React Native applications and uses the native routing features of the platform.
- **<StaticRouter> -** This router is designed for server-side rendering and generates a set of static routes based on a given location.

In v6.4, new routers were introduced that support the new data APIs and to create custom routers:

- **createBrowserRouter:** This function creates a custom <BrowserRouter> router with a custom history object. The custom history object can be used to manipulate the browser's URL and handle navigation between pages without causing a full page refresh.
- **createMemoryRouter:** This function creates a custom <MemoryRouter> router with a custom history object. The custom history object can be used to manipulate the router's state and handle navigation between pages.
- **createHashRouter:** This function creates a custom <HashRouter> router with a custom history object. The custom history object can be used to manipulate the browser's URL hash and handle navigation between pages without causing a full page refresh.

createBrowserRouter

This is the recommended router for all React Router web projects. It uses the DOM History API to update the URL and manage the history stack.

For example, WAY-1

```
import { createBrowserRouter, RouterProvider } from "react-router-dom";
import Home from "./pages/Home";
import About from "./pages/About";

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

export default App;
```

In this example, the createBrowserRouter function is used to create a custom <BrowserRouter> router with two routes: one for the home page, and one for the about page. The RouterProvider component is used to wrap the app and provide access to the custom router.

1. Import the necessary modules, including createBrowserRouter, RouterProvider, Home, and About.
2. Use createBrowserRouter to create a custom <BrowserRouter> router with the two routes: Home and About.
3. Wrap the app with RouterProvider and pass in the custom router as a prop. The RouterProvider component ensures that the routing context is available to all child components of your app.
4. Render the App component.

For Example, WAY-2

```
import {
    createBrowserRouter,
    createRoutesFromElements,
    Route,
    RouterProvider,
  } from "react-router-dom";
  import Home from "./pages/Home";
  import About from "./pages/About";
  import Items from "./pages/Items";

  function App() {
    const routes = createRoutesFromElements(
      <>
        <Route path="/" element={<Home />} />
        <Route path="/about" element={<About />} />
        <Route path="/items" element={<Items />} />
      </>
    );
    const router = createBrowserRouter(routes);

    return (
      <>
        <RouterProvider router={router} />
      </>
    );
  }

  export default App;
```

In this example, the **createRoutesFromElements** function is used to create an array of route objects from JSX elements and avoid manually creating an array of route objects. The **createBrowserRouter** function is then used to create a custom **<BrowserRouter>** router with the array of route objects. Finally, the **RouterProvider** component is used to wrap the app and provide access to the custom router.

<Link> Component

The <Link> component is a core component of React Router that is used to navigate between different routes in your application. It is a declarative way to create hyperlinks that allows you to handle navigation without reloading the entire page. The to prop specifies the target route. When a user clicks on a <Link> component, React Router will automatically update the URL and render the appropriate component for the target route.

For Example,

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

function Home() {
  return (
    <>
      <main>
        <h1>Home Page</h1>
        <Link to="/about">About</Link>&nbsp;&nbsp;
        <Link to="/items">Items</Link>
      </main>
    </>
  );
}

export default Home;
```

The Link component can be used to create the back link, and the to prop is set to "/" to specify the target route. When the link is clicked, the user will be taken back to the home page.

```
import { Link } from "react-router-dom";
function About() {
  return (
    <>
      <main>
        <h1>About Page</h1>
        <Link to="/">back</Link>
      </main>
    </>
  );
}

export default About;
```

Nested Routes

Nested routes in React Router allow you to define routes that are nested within another route. This can be useful for building more complex UIs, such as layouts with multiple sections that have their own routes.

For Example,

```
import {
    createBrowserRouter,
    createRoutesFromElements,
    Route,
    RouterProvider,
  } from "react-router-dom";
```

```
import Home from "./pages/Home";
import About from "./pages/About";
import Items from "./pages/Items";
import Navbar from "./components/Navbar";

function App() {

  const router = createBrowserRouter([
    {
      path: "/",
      element: <Navbar />,
      children: [
        { index: true, element: <Home /> },
        { path: "/about", element: <About /> },
        { path: "/items", element: <Items /> },
      ],
    },
  ]);
  return (
    <>
      <RouterProvider router={router} />
    </>
  );
}

export default App;
```

Parent route is defined with the path of / and an element of Navbar. The children property is an array of child routes, which includes an index route, a route with the path of /about, and a route with the path of /items.

```
import { Link, Outlet } from "react-router-dom";

function Navbar() {
  return (
    <>
      <div className="nav">
        <Link to="/">
          <h4>HOME</h4>
        </Link>
        <Link to="/about">
          <h4>ABOUT</h4>
        </Link>

        <Link to="/items">
          <h4>ITEMS</h4>
        </Link>
      </div>
      <Outlet />
    </>
  );
}

export default Navbar;
```

The Navbar component renders links to the child routes using the Link component. The Outlet component renders the appropriate child component based on the current nested route. The Outlet component renders the appropriate child component based on the current nested route. For example, if the current URL is /about, the Outlet component will render the About component.

<NavLink> Component

The <NavLink> component is similar to the <Link> component in that it creates a hyperlink to a specified location. However, it adds the ability to add styling and active classes to the link when it is active. This can be useful for highlighting the current page or route in a navigation menu.

For Example,

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

function Navbar() {
  return (
    <>
      <div className="nav">
        <NavLink
          style={({ isActive }) => (isActive ? { color: "blue" } : undefined)}
          to="/"
        >
          <h4>HOME</h4>
        </NavLink>

        <NavLink
          style={({ isActive }) => (isActive ? { color: "blue" } : undefined)}
          to="/about"
        >
          <h4>ABOUT</h4>
        </NavLink>

        <NavLink
          style={({ isActive }) => (isActive ? { color: "blue" } : undefined)}
          to="/items"
        >
          <h4>ITEMS</h4>
        </NavLink>
      </div>
      <Outlet />
    </>
  );
}

export default Navbar;
```

In this example, the exact attribute is used with the to prop of the first NavLink. This ensures that the link is only active when the exact URL path matches the value of to.

Relative vs Absolute Paths

In web development, a path is a URL endpoint that specifies the location of a specific resource or content on a web server. A base URL is the root URL that serves as the starting point for all the other URLs in a website. In React Router, paths are used to define routes that map to specific components in your application. These routes can be either relative or absolute.

A relative path is a path that is relative to the current location. For example, if the current location is "/users", and you want to link to the "profile" page, the relative path would be "profile". The resulting link would be "/users/profile".

An absolute path is a path that starts with a forward slash ("/") and is relative to the root of the website. For example, if the current location is "/users", and you want to link to the "home" page, the absolute path would be "/home". The resulting link would be "/home".

In general, it is recommended to use relative paths whenever possible, as they are more flexible and can be used in different contexts. Absolute paths are useful when you need to link to a specific page that is not in the current directory or when you want to link to a page in a different part of the website.

Dynamic Routes

Dynamic routes are routes that contain parameters, which can be used to dynamically generate content based on user input or other data. In React Router, dynamic routes are defined using a colon (:) followed by the name of the parameter. For example, a dynamic route for a user profile page might look like /users/:id, where :id is the parameter that will be replaced with the actual user ID.

To access the parameter in your component, you can use the useParams hook provided by React Router. For example:

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

function UserProfile() {
  const { id } = useParams();
  return <div>User profile for user {id}</div>;
}
```

Summarizing it

Let’s summarize what we have learned in this Lecture:

- Learned about Routing Mechanism in React.

- Learned about React Router.
- Learned about types of React Router.
- Learned about createBrowserRouter.
- Learned about Nested Routes.
- Learned about <Link> and <NavLink> Components.
- Learned about Relative, Absolute and Dynamic Routes.

Some References:

- React Router Documentation: link
- Client Side Routing: link