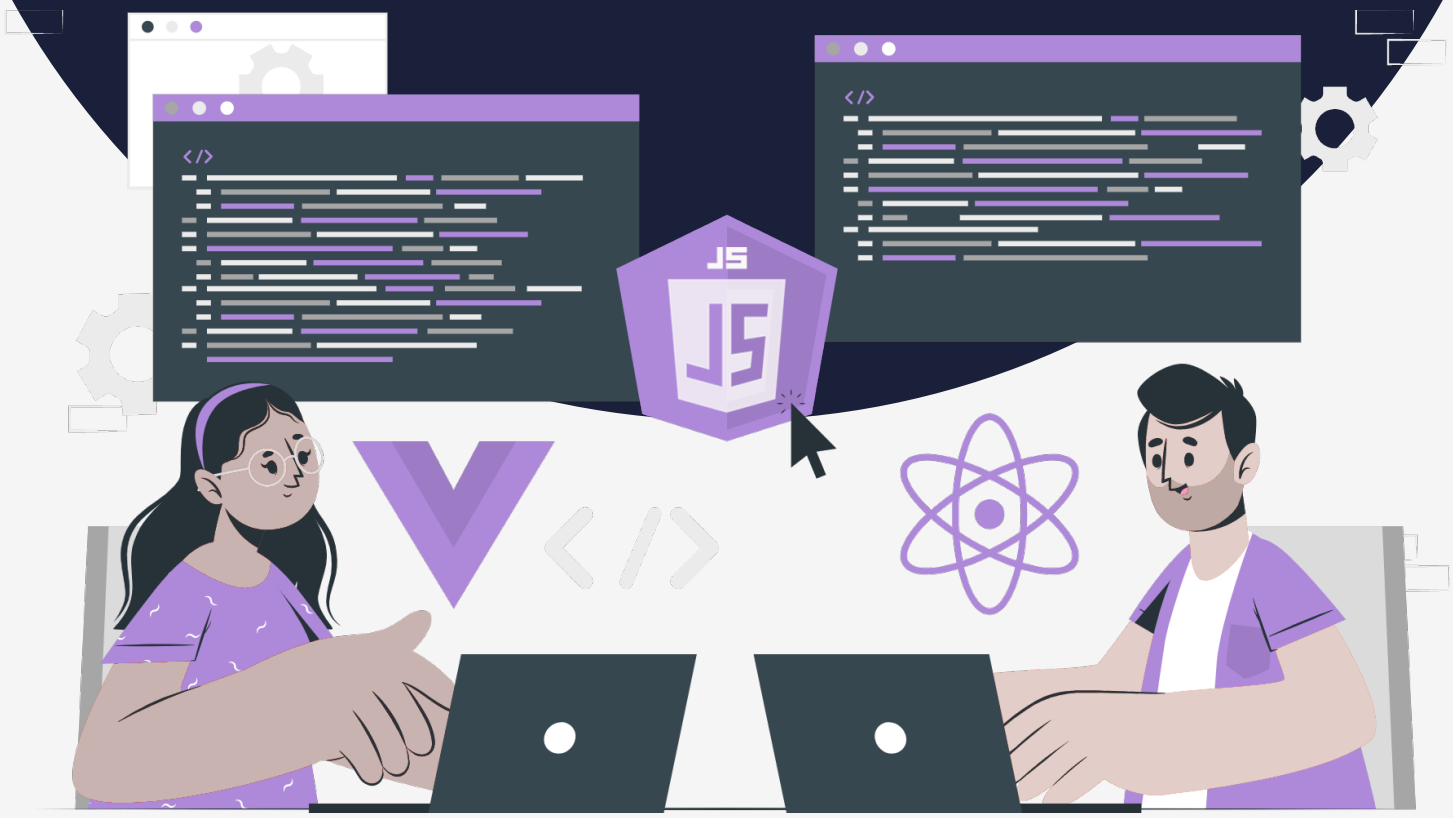
Lesson:

React Router DOM (V6)



PW SKILLS

Topics Covered:

1. What is Routing?
2. Configuring the Router.
3. Problems with <a> tag.
4. Nested routing.

What is Routing?
Routing in React is the process of determining which components to render based on the current URL or route. React Router is a popular library for routing in React, which provides the ability to map different routes to different components and handle navigation within a React application.

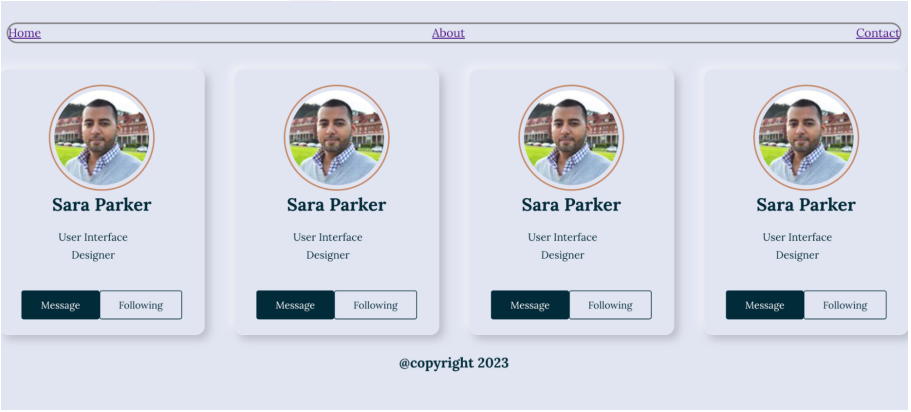
With React Router, you can specify the different routes in your application and associate them with specific components. When the user navigates to a different URL or route within the application, the corresponding component is displayed on the page.

To install this package

```
npm i react-router-dom
```

React Router DOM is an npm package that enables you to implement dynamic routing in a web app. It allows you to display pages and allow users to navigate them.

Let's take an example and create a page as like below



It has Home, About and Contact tabs. If we Click on any particular tab, then it redirects to that particular page. So We first have to create a routing configuration.

Configuring the Router

So first We have to import createBrowserRouter from 'react-router-dom'
createBrowserRouter is the recommended router for all React Router web projects. Generally It is a function that will help us create routing.This alone won't work, We need to provide this appRouter to our app. For that there is a component RouterProvider which is coming from `react-router-dom` and render this RouterProvider in your index.js.

To create the routing , we do:

```javascript
import {createBrowserRouter, RouterProvider } from
'react-router-dom'


const appRouter = createBrowserRouter([{}])


const root =
ReactDOM.createRoot(document.getElementById('root'));
```

```
 root.render(
   <React.StrictMode>
   <RouterProvider router={appRouter} />
   </React.StrictMode>
 );
```

createBrowserRouter allows us to define the routes as an array of objects in which we can specify a path, the element to be rendered when the path is browsed.

```javascript
import React from 'react';
import ReactDOM from 'react-dom/client';
import './index.css';
import App from './App';
import About from './About';
import Contact from "./Contact"
import { createBrowserRouter, RouterProvider } from
'react-router-dom';


const appRouter = createBrowserRouter([
  {
    path:"/",
    element:<App/>,
   },
  {
    path:"/about",
    element:<About/>,
  },
  {
    path:"/contact",
    element:<Contact/>,
  },
])
```

```
const root =
ReactDOM.createRoot(document.getElementById('root'));
root.render(
  <React.StrictMode>
  <RouterProvider router={appRouter} />
  </React.StrictMode>
);
```

If we go to **localhost:3000/about** , It shows us the about page .

Let's create a navbar so that we can visit our other pages also by using it.

JavaScript

```
import React from 'react'
import './App.css;

const Header = () =>{
  return (

    <>
   <div className="nav-items">
        <ul>
          <a href="/">
            <li>Home</li>
          </a>
          <a href="/about">
            <li>About</li>
          </a>
          <a href="/contact">
            <li>Contact</li>
          </a>


         </ul>
       </div>
     </>
   )
 }

 export default Header;
```

If we click on that About or contact , it is redirected to that respective page

Problems with <a> tag:
It will reload the entire page when it is clicked, It disrupts user experience and can result in a slower page load time. This causes problems for single page applications(SPAs).

Single Page Application(SPAs):

- React apps are SPAs.
- Having SPAs should not reload the entire page. It should not make network calls when we are changing pages.
- Loads a single HTML page and dynamically updates the page in response to user interaction without reloading the entire page.
- This approach allows for faster navigation and a more seamless user experience.

There are two types of Routing.
1.Client-side routing.
2.Server-side routing

1.Server-side routing:
- All our pages come from the server side.
- It makes a network call, gets the HTML, CSS, JS and loads the entire page.

2.Client-side routing:
- It dynamically updates the content of SPAs in response to changes in url.
- It doesn't do full page reloads.

So to solve this issue, react-router-dom gives us a `**Link**`. Let's remove the <a> tag and use the <Link> in the same example.

```javascript
import React from "react";
import { Link } from "react-router-dom";
import "./App.css";
const Header = () => {
  return (
    <>
      <div className='nav-items'>
      <ul>
       <Link to="/">
        <li>Home</li>
        </Link>
        <Link to="/about">
        <li>About</li>
        </Link>
        <Link to="/contact">
        <li>Contact</li>
        </Link>
      </ul>
    </div>
 </>
  );
};

export default Header;
```

Note:If we inspect that page in the element it shows <a> tag instead of <Link> tag. react router dom converts the Link tag to <a> tag because browsers only understand <a> tag .

Nested Routing:

Nested routing means routes inside another route.for example,If we click on the About, It redirects to the About page but in this page no Header and Footer is there. So to solve this problem or we can say to keep Header and Footer stick on to every page , we have to change to routing config i.e to make the About page children of <App/> .

Nesting routes with <Outlet/>:

An <Outlet> should be used in parent route elements to render their child route elements. This allows nested UI to show up when child routes are rendered. If the parent route matched exactly, it will render a child index route or nothing if there is no index route.

index.js

```javascript
import React from 'react';
import ReactDOM from 'react-dom/client';
import './index.css';
import App from './App';
import About from './About';
import Contact from "./Contact"
import { createBrowserRouter, RouterProvider } from
'react-router-dom';
import Body from './Body';

  const appRouter = createBrowserRouter([
  {
```

```
    path:"/",
    element:<App/>,
      children:[
       {
         path:"/",
         element:<Body/>,
       },
       {
        path:"/about",
        element:<About/>,
      },
      {
        path:"/contact",
        element:<Contact/>,
      }
      ]
  },
])

const root =
ReactDOM.createRoot(document.getElementById('root'));
root.render(
  <React.StrictMode>
  <RouterProvider router={appRouter} />
  </React.StrictMode>
);
```

App.js

```
JavaScript
import Footer from "./Footer";
import Header from "./Header";
import { Outlet } from "react-router-dom";

function App() {
  return (
    <div>
     <Header />

      {/*  This element will render either <About/> when the
url is "/about",<Contact/> when the url is "/contact" or null
if it is "/" */}

     <Outlet />

     <Footer />
    </div>
  );
}

export default App;
```

The **<Outlet>** element is used as a placeholder. In this case an <Outlet> enables the App component to render its child routes. Thus the<Outlet> element will render either a <About> or <Contact> element depending on the current location.

How "&&" works:

```javascript
function MyComponent({ condition }) {
  return (
    <div>
      <h1>Title</h1>
      {condition && <ConditionalComponent />}
    </div>
  );
}
```

- if condition is a truthy value, **<ConditionalComponent /> is rendered.**
- if condition is a falsy value, **<ConditionalComponent /> is not rendered.**

if the first operand **(condition)** is falsy, the AND operator (&&) stops and it does not evaluate the second operand **(<ConditionalComponent/>)**.

```javascript
// This will display an alert box.
true && alert('PW SKILL');
// This won't do anything.
false && alert('Nothing');
```

Why to avoid "&&"

In the above example, if the condition is true or false, We get what we would expect - **<ConditionalComponent />** is or is not rendered respectively.However, if condition doesn't evaluate to a boolean, it may cause trouble.

- if condition is 0, 0 is displayed in the UI.
- if condition is undefined, you'll get an error: "Uncaught Error: Error(...): Nothing was returned from render. This usually means a return statement is missing. Or, to render nothing, return null."

Note: To prevent avoidable UI bugs, we can use the javascript ternary operator for conditional rendering of React components instead of the logical AND operator.