REACT JS ROUTER

REACT Router

Routing is a process in which a user is directed to different pages based on their action or request. ReactJS Router is mainly used for developing Single Page Web Applications. React Router is used to define multiple routes in the application. When a user types a specific URL into the browser, and if this URL path matches any 'route' inside the router file, the user will be redirected to that particular route.

React Router plays an important role to display multiple views in a single page application. Without React Router, it is not possible to display multiple views in React applications. Most of the social media websites like Facebook, Instagram uses React Router for rendering multiple views.

To add React Router in the application, it has to be installed with the following command:

>npm install react-router-dom

REACT Router – Types

<BrowserRouter>: It is used for handling the dynamic URL.

<HashRouter>: It is used for handling the static request.

Let the BrowserRouter is used in the demo application.

REACT Router –Components

Components to be imported from the browser package is BrowserRouter, Routes, Route, Outlet, Link.

BrowserRouter: BrowserRouter is a router implementation that uses the HTML5 history API (pushstate, replacestate, and popstate events) to keep your UI in sync with the URL. It is the parent component used to store all other components.

Routes**:** This is a new component introduced in v6 and an upgrade of the component. The main advantages of Switch Over Routes is Instead of traversing in sequence, routes are selected based on the best match. An application can have multiple <Routes>.

Route: A route is a conditionally shown component that provides UI when its path matches the current URL. <Route>s can be nested. The first <Route> has a path of / and renders the Layout component. The nested <Route>s inherit and add to the parent route.

REACT Router –Components

Outlet: The <Outlet> renders the current route selected.

Links: The Links component creates links for different routes and implements navigation around the application. It works as an HTML anchor tag. <Link> is used to set the URL and keep track of browsing history.

REACT Routing – Application Creation

Step-1: Create an application with the command

>create-react-app routedemo

Step-2: Now, install the router package with the command

>npm install react-router-dom

Step-3: Restructure the project folder ‘routedemo’ by adding ‘pages’ folder under the ‘src’ folder.

Step-4: Create **Layout.js, Home.js, Login.js, Contact.js** and **NoPage.js** files in ‘**pages**’ folder.

REACT Routing – Application Creation

Layout.js

```
import { Outlet, Link } from "react-router-dom";
function Layout() {
  return (
    <>
      <nav>
        <ul> <li>
            <Link to="/">Home</Link> </li>
          <li>
            <Link to="/login">Login Page</Link> </li>
          <li>
            <Link to="/contact">Contact</Link> </li>
        </ul>   </nav>
      <Outlet />
    </>
  ) };
export default Layout;
```

Note: Here, Home.js is the default route and it doesn't have any path only '/'. Setting the path to * will act as a catch-all for any undefined URLs. This is great for a 404 error page.

REACT Routing – Application Creation

Home.js

```
function Home(){
  return <h1>Home</h1>;
};
export default Home;
```

Contact.js

```
function Contact() {
  return(
  <><h1>Contact Me</h1>
    VIT Vellore<br />
              Katpadi - Chennai Salai<br />
  </>
  )
};
export default Contact;
```

Login.js

```
function Login(){
  return(
    <><h3>Email Login</h3>
    <form>
      Email Id:<input type="email" /><br />
      Password: <input type="password" /><br />
      <input type="submit" value="LOGIN" />
    </form>
    </>
  )
}
export default Login
```

NoPage.js

```
function NoPage() {
  return <h1>404 Page Not Found</h1>;
};

export default NoPage;
```

REACT Routing – Application Creation

Step-5: Update the App.js file with the following modifications. Import all the above ‘js’ files from page folder, BrowserRouter, Routes and Route from ‘react-router-dom’.

```
import {BrowserRouter,Routes,Route} from "react-router-dom"
import Layout from "./pages/Layout";
import Home from "./pages/Home";
import Login from "./pages/Login";
import Contact from "./pages/Contact";
import NoPage from "./pages/NoPage";
```

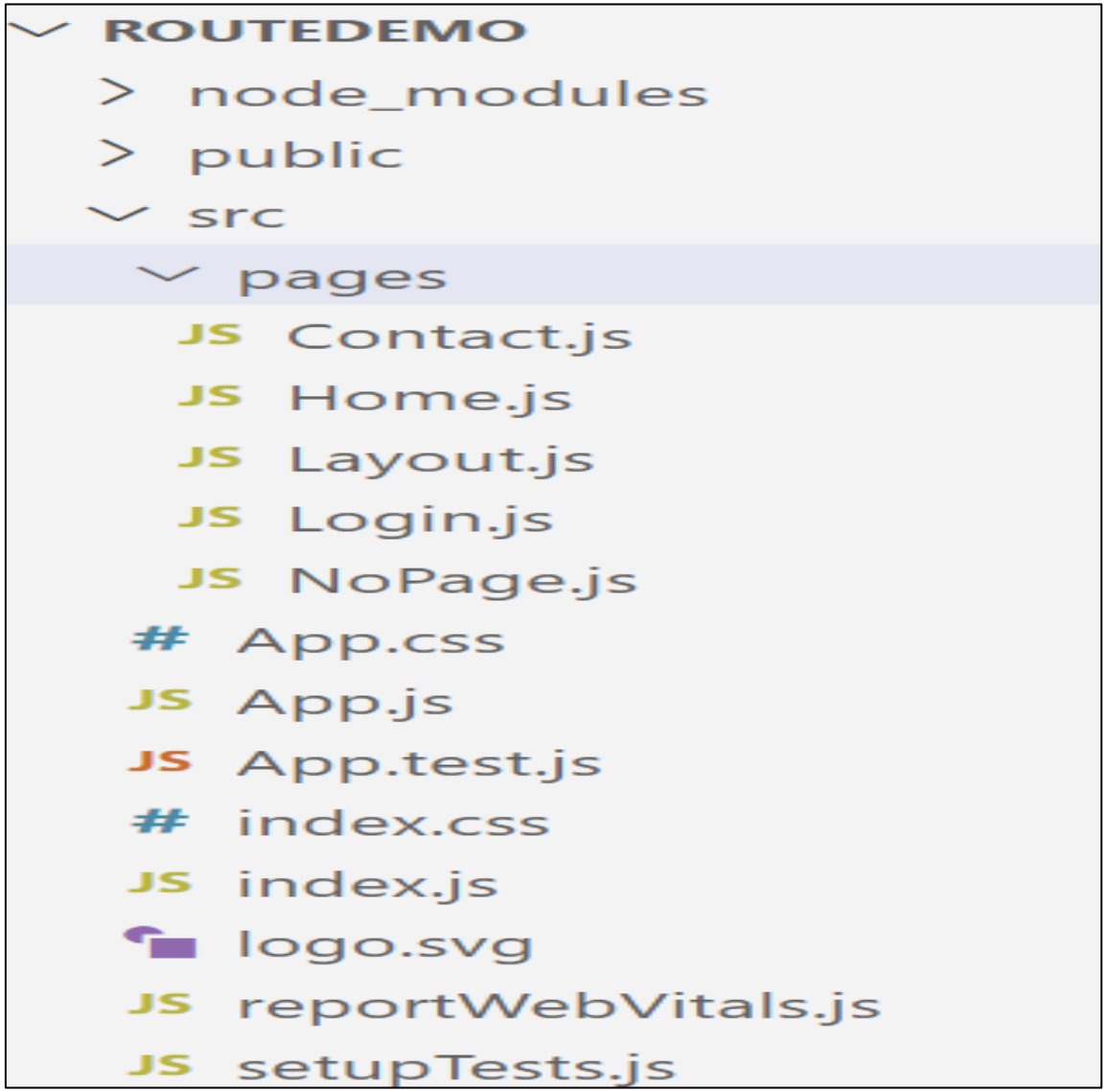
REACT Routing – Application Creation

Step-6: Update the default App() of App.js with the following code.

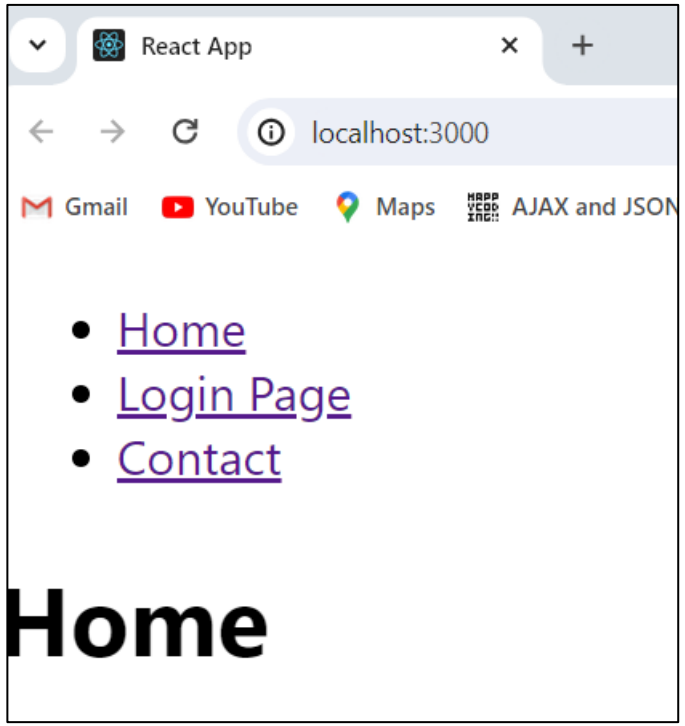
```
function App() {  return (    <div >
  <BrowserRouter>
    <Routes>
      <Route path="/" element={<Layout />}>
      <Route index element={<Home />} />
      <Route path="login" element={<Login />} />
      <Route path="contact" element={<Contact />} />
      <Route path="*" element={<NoPage />} />
      </Route>
  </Routes>
 </BrowserRouter>
</div>  );}
```

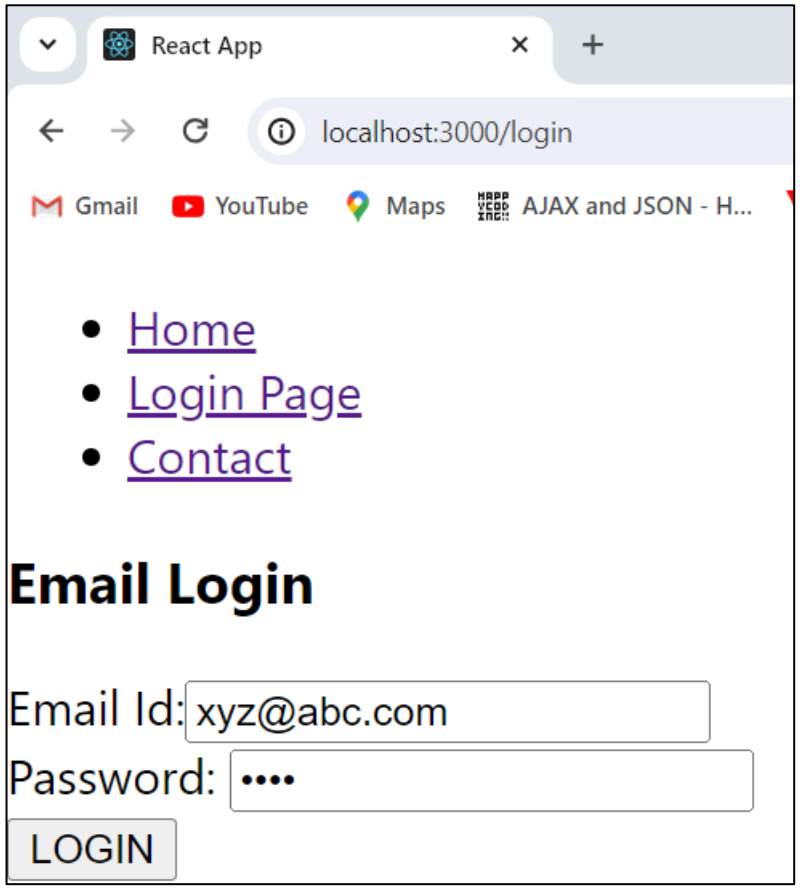
Step-7: Save all the files and run the project by providing the command ‘npm start’ from root folder

routedemo – project folder structure

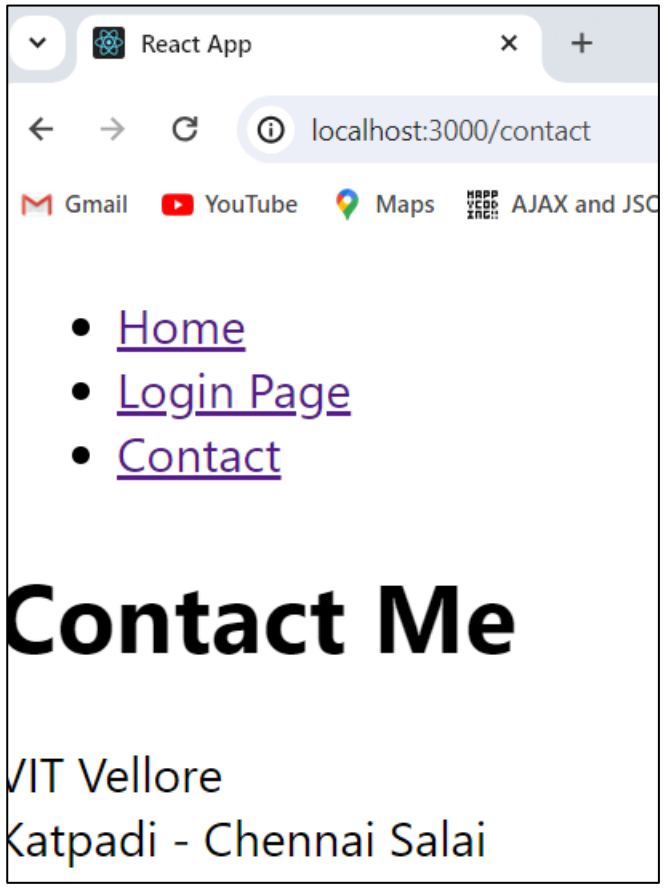


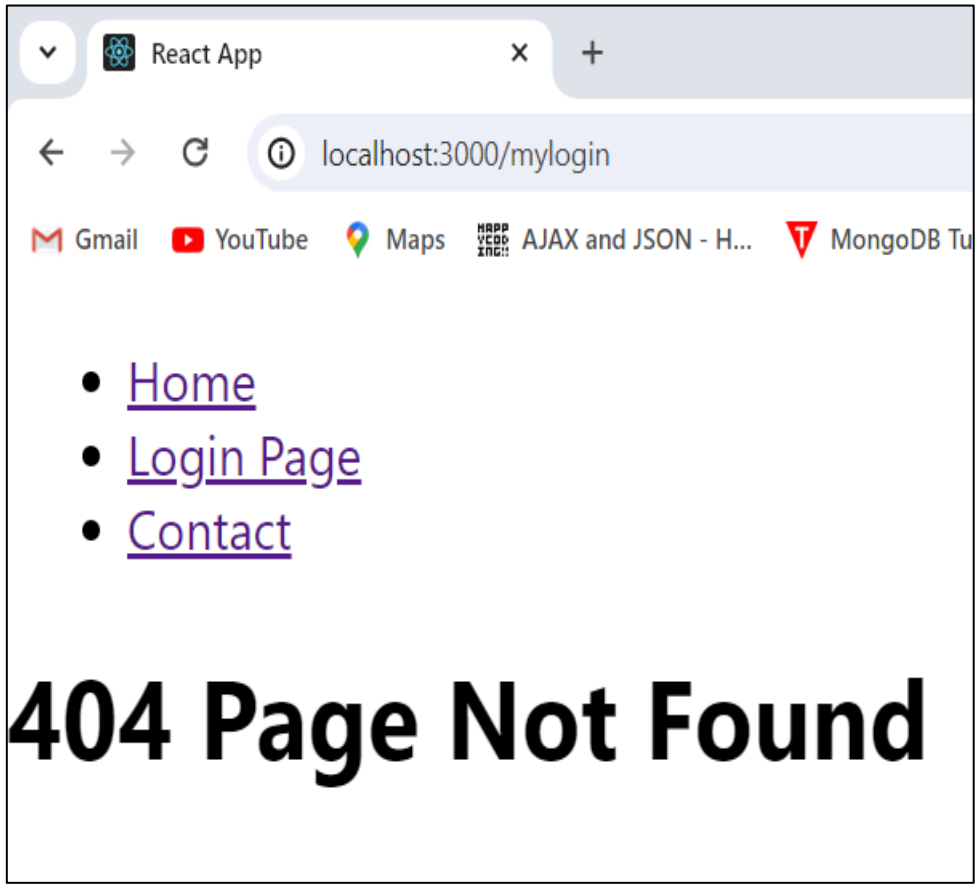
REACT Routing – Application Creation





REACT Routing – Application Creation





Nested Routes

- In between a Route component you can embed another Route: <Route>{/* Children routes go here*/}</Route>
- <Outlet /> behaves a bit like props.children in standard React. <Outlet /> is the placeholder location for where the nested children routes will be rendered.

REACT – Route Parameters

Passing parameters to routes in a React router enables components to access the parameters in a route's path. These parameters can be anything from a user object to a simple string or number. It supports the idea of creating reusable and interactive components.

Parameters, often referred to as ***params***, are dynamic parts of the URL that can change and are set to a specific value when a particular route is matched.

In React Router, parameters can be used to capture values from the URL, which can then be used to further personalize or specify what to render in a particular view.

By passing parameters, we're making routes dynamic, allowing us to reuse the same component for different data based on the parameter value.

REACT – Route Parameters

Every route in the React Router is associated with a path. This path is a string or a regular expression, which React Router matches with the current URL. If the current URL matches the path of a route, the associated component or render prop function of that route gets rendered. The 'Route path' is where we can specify parameters that we want to pass to our components.

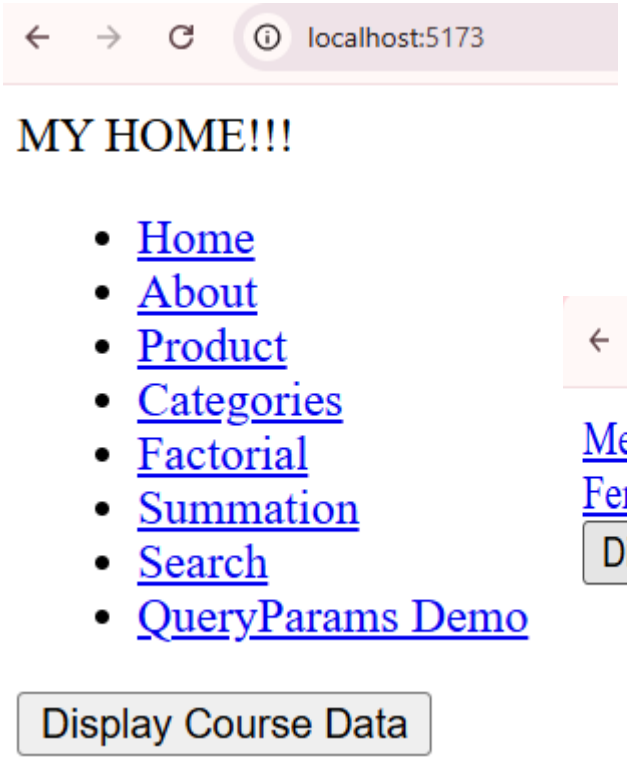
When defined in a route, parameters allow React Router to understand which part of the URL should be captured as a dynamic argument. These parameters can then be accessed in the routed components using the props passed by React Router. This way, parameters help in creating dynamic and personalized routes in a React application.

REACT – Route Parameters

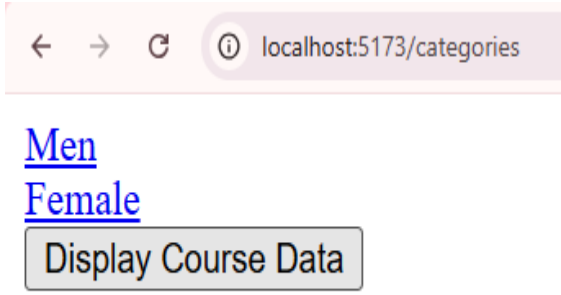
- Used to pass data that is specific to a particular route
- Defined in the route path using a colon (:)
- Accessed using the useParams hook in React Router
- Syntax: /path/:parameter

REACT – NESTED ROUTES & ROUTE PARAMETERS

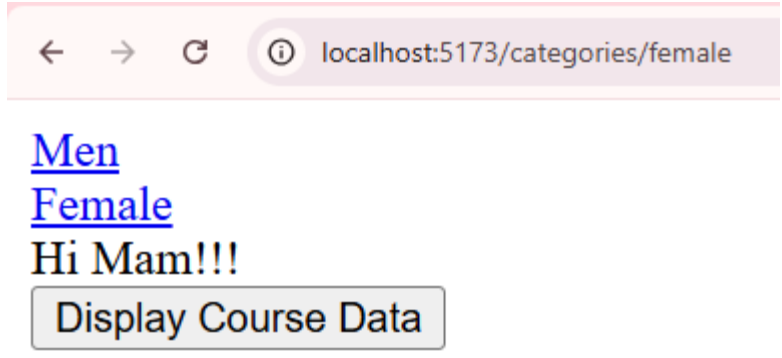
```
function App() {
 return (
  <>
   <div>
    <BrowserRouter>
    <Routes>
     <Route path="/" element = {<Home/>}/>
     <Route path="about" element={<About />} />
     <Route path="product" element={<Product />} />
     <Route path="*" element={<NoPage />} />
     <Route path="categories" element={<Categories/>}>
                <Route path="male" element={<Male/>}/>
                  <Route path="female" element={<Female/>}/>
              </Route>
     <Route path="/fact/:id" element={<Fact />} />
     <Route path="/firstvar/:vara/secondvar/:varb" element={<Summation />} />
     <Route path="search" element={<Search />} />
     <Route path="queryparamdemo" element={<DemoQueryParam />} />
    </Routes>
    </BrowserRouter>
   </div>
```



CLICKING ON CATEGORIES



CLICKING ON FEMALE



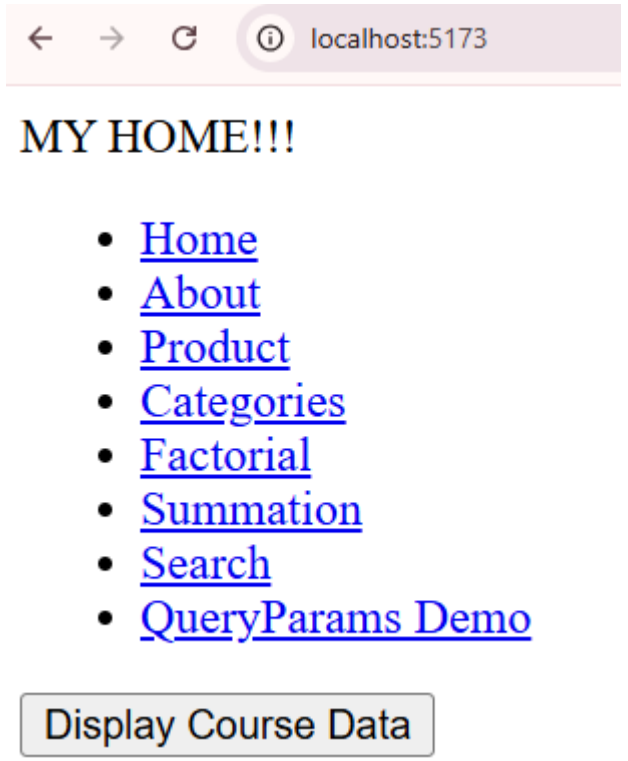
Route Params Retreival

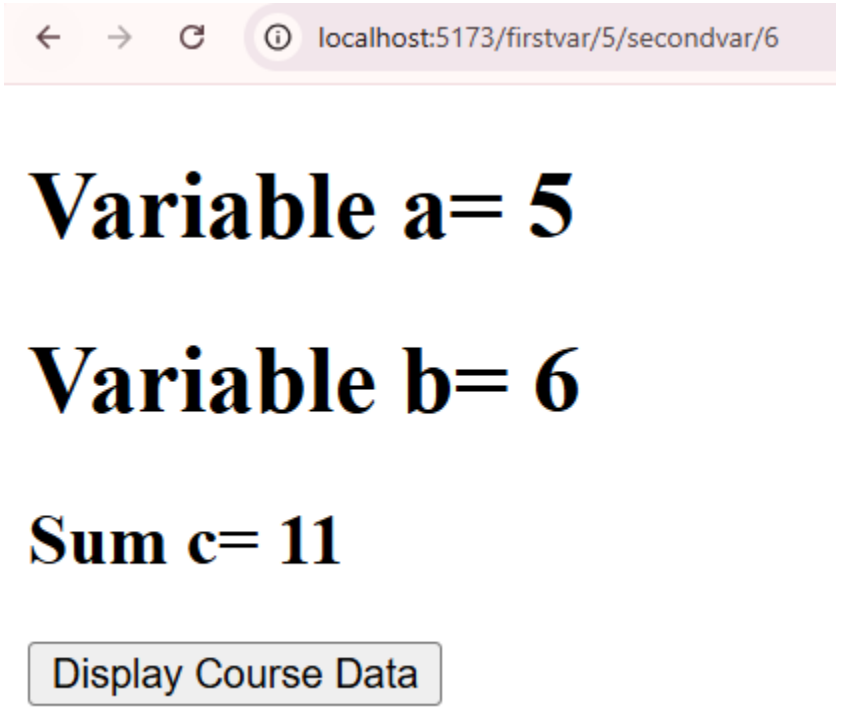
```
import { useParams } from "react-router-dom";

export default function Summation () {
  const { vara, varb } = useParams();
  var varc= Number(vara)+ Number(varb);

  return (
    <div>
      <h1>Variable a= {vara}</h1>
      <h1>Variable b= {varb}</h1>
      <h2>Sum c= {varc}</h2>
    </div>
  );
};
```

CLICKING ON SUMMATION LINK





REACT – Query Parameters

Query parameters are elements appended to a URL to convey information in an HTTP request. It is added to the URL after the route path, separated by a question mark (?)

Typically used in GET requests, they follow the "?" in a URL and are separated by "&" if multiple.

Commonly in key-value pairs like "?name=John&age=25," they support multiple values, empty flags, URL encoding, arrays, nested structures, pagination controls.

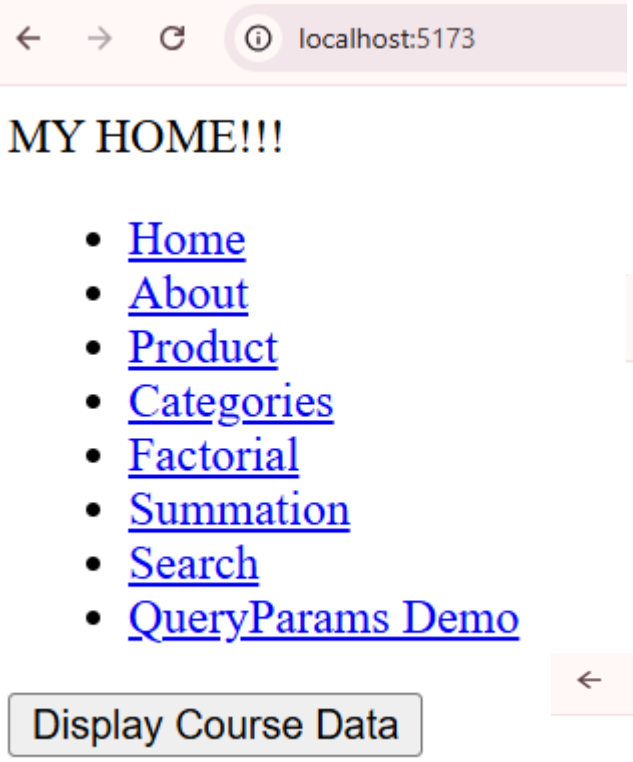
For instance, if you've ever seen a URL like **https://example.com/search?q=react+router**, the ***q=react+router*** part is a query string. The ***q*** is the key, and ***react+router*** is the value.

REACT – Passing Query Parameters

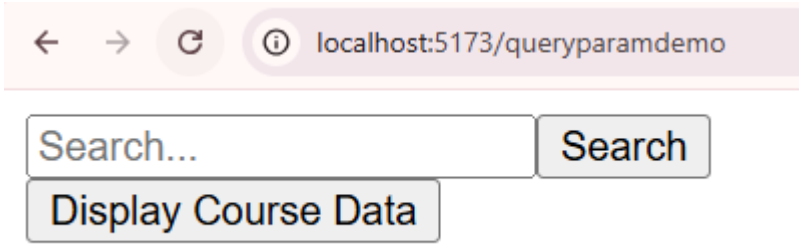
Used to pass data that is applicable to multiple routes or components

React Router 6 comes with a new hook called **useSearchParams** that allows you to access and manipulate the query parameters in the URL.

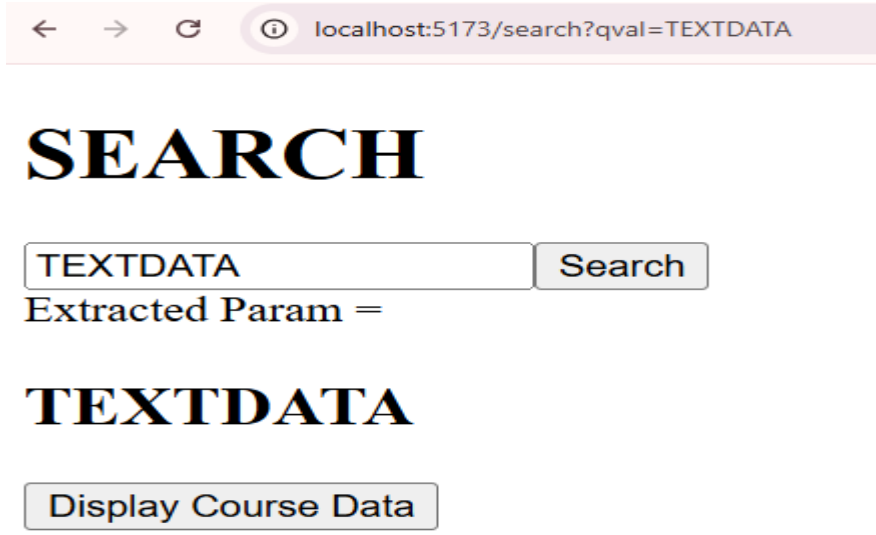
The following examples demonstrate how to pass and retrieve query parameters in a React component using React Router.



CLICKING ON QUERY PARAMS DEMO. TYPE IN THE INPUT FIELD AND CLICK SEARCH. NOTE THE URL



NOTE THE URL



REACT – Retrieve Query Parameters

The **useSearchParams** hook returns an array with two elements: the current search parameters and a function to update the search parameters.

To access the value of a specific query parameter, you can use the get method of the URLSearchParams object

```
const [searchParams, setSearchParams]=useSearchParams();
const query = searchParams.get("qval");
```

To update the value of a query parameter, you can call the set method of the URLSearchParams object and pass in the name and value of the parameter:

You can also update multiple query parameters at once by passing in an object:

```
setSearchParams({qval:term,tval:23});
```