**React Query – Introduction**

So far, all our projects had only one page. But what if we want to have an app with **multiple pages in React**? We need to use **React Router**. React router is not part of official React library and there are other alternatives out there that you can use, but is definitely the most used option.

**Installing it**

To install it, just navigate to your project directory, open the terminapl and run npm install react-router-dom@6

**Setting up**

Ap far as the setup, we want to go to **App.js** and we want to import 3 things:

* **Browser Router**: which is going to connect to the actual browser.
* **Routes**: which is going to be a parent for all our routes
* **Route**: which we will use to setup a single page

In the example below, we have a simple setup for a home page and a “testing”. Note that the home page is always referenced with just a “/”. By changing the link in the browser, we can access both pages.

import { BrowserRouter, Routes, Route } from "react-router-dom";

function App() {

  return (

    <BrowserRouter>

      <Routes>

        <Route path="/" element={<div>home page</div>} />

        <Route

          path="about"

          element={

            <div>

              <h2>about</h2>

            </div>

          }

        />

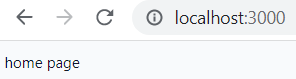
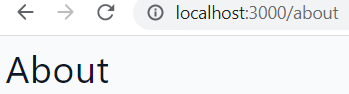
      </Routes>

    </BrowserRouter>

  );

}

export default App;

Obviously, we won’t be writing all our code in **App.js**, but instead importing components to it, which is much easier to manage.

import { BrowserRouter, Routes, Route } from "react-router-dom";

import Home from "./pages/Home";

import About from "./pages/About";

import Products from "./pages/Products";

function App() {

  return (

    <BrowserRouter>

      <Routes>

        <Route path="/" element={<Home />} />

        <Route path="about" element={<About />} />

        <Route path="products" element={<Products />} />

      </Routes>

    </BrowserRouter>

  );

}

export default App;

**Navigating around the project**

Unlike the traditional multi page application where we use the **href** (link element) in this case if we want to navigate around our project internally (grab the linked component from React) we us the Link component.

If we want to go somewhere externally (for example, if I want to go to Google.com), then yes, you still need to use the **href** link component like so.

To use the Link component, we just need to provide the **URL** we have defined for that page (excluding the home URL). This is done in the component where the link is. For example, we can set up a button in the **Home** component that points to the **About** page:

|  |  |
| --- | --- |
| import { Link } from "react-router-dom";  const Home = () => {    return (      <section className="section">        <h2>Home Page</h2>        <Link to="/about" className="btn">          About        </Link>      </section>    );};  export default Home |  |

**Error Page**

What if by mistake the user types some link that doesn’t exist? If we don’t do anything, it simply doesn’t display anything.

In order to handle this, we need to set up an **error** page. **The link must be in the end**, and with **\*** which basically **catches everything, if all the other pages before did not exist**.

We can then also add a **back home** button in our error page.

|  |
| --- |
| **App.jsx** |
| import { BrowserRouter, Routes, Route } from "react-router-dom";  import Home from "./pages/Home";  import About from "./pages/About";  import Products from "./pages/Products";  import Error from "./pages/Error";  function App() {    return (      <BrowserRouter>        <Routes>          <Route path="/" element={<Home />} />          <Route path="about" element={<About />} />          <Route path="products" element={<Products />} />          <Route path="\*" element={<Error />} />        </Routes>      </BrowserRouter>    );  }  export default App; |
| **Error.jsx** |
| import { Link } from "react-router-dom";  const Error = () => {    return (      <section className="section">        <h2>404</h2>        <p>page not found</p>        <Link to="/">Back Home</Link>      </section>    );  };  export default Error; |
|  |

**Shared Layouts – Navbar and Footers**

You might want to **add the same component to all the pages** (for example **navbar** and **footers**). One way to achieve this is by adding them to the **App.jsx** in the order you wish them to show up, outside the Routes component.

import { BrowserRouter, Routes, Route } from "react-router-dom";

import Home from "./pages/Home";

import About from "./pages/About";

import Products from "./pages/Products";

import Error from "./pages/Error";

function App() {

  return (

    <BrowserRouter>

      <nav>our navbar</nav>

      <Routes>

        <Route path="/" element={<Home />} />

        <Route path="about" element={<About />} />

        <Route path="products" element={<Products />} />

        <Route path="\*" element={<Error />} />

      </Routes>

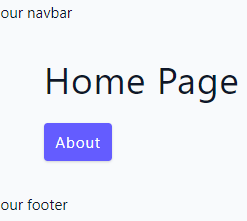
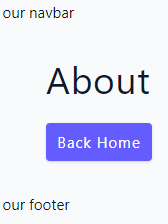
      <footer>our footer</footer>

    </BrowserRouter>

  );

}

export default App;

**Nested Routes**

**With the previous setup, the navbar and footer show up in all pages, which might not be what you want**. In order to setup a shared layout (without sharing in every page), we need the **routes to be nested**. So, instead of self-closing the Route component, we add all the other links/components inside it. In the example below, About, Products and Error links are nested inside the Home link:

import { BrowserRouter, Routes, Route } from "react-router-dom";

import Home from "./pages/Home";

import About from "./pages/About";

import Products from "./pages/Products";

import Error from "./pages/Error";

function App() {

  return (

    <BrowserRouter>

      <nav>our navbar</nav>

      <Routes>

        <Route path="Home" element={<Home />}>

          <Route path="about" element={<About />} />

          <Route path="products" element={<Products />} />

          <Route path="\*" element={<Error />} />

        </Route>

      </Routes>

      <footer>our footer</footer>

    </BrowserRouter>

  );

}

export default App;

Bare in mind that now the link to access the children pages is always referring to the parent page. For example, the link for the **Home** page in this case would be <http://localhost:3000/Home> and to the **About** page http://localhost:3000/Home/about. So, we would have to also change the link in the component

|  |  |
| --- | --- |
| **Before** | **After** |
| <Link to="/about" className="btn"> | <Link to="/Home/about" className="btn"> |

**Outlet Component**

We still have a problem with this nested structure setup. **We see the Home** **page regardless of the link provided.**

In order to fix this, we need to go to the parent (in our case **Home**) and import the Outlet component and display it. **Whatever we set up around this** Outlet **component is going to be the shared layout across the pages that are nested inside the parent**.

To correct way to do this is by creating the shared component (e.g. **Navbar**) in a different component and put all the links there. Then, import that component in the **Home** page or whichever the parent is.

Notice that the **Navbar** is on top, and the Outlet component is where you want the content of **About** or **Products** to be.

|  |
| --- |
| **Navbar.jsx** |
| import { Link } from "react-router-dom";  const Navbar = () => {    return (      <nav className="navbar">        <Link to="/">Home</Link>        <Link to="/about">About</Link>        <Link to="/products">Products</Link>      </nav>    );  };  export default Navbar; |
| **Home.jsx** |
| import { Link, Outlet } from "react-router-dom";  import Navbar from "../components/Navbar";  const Home = () => {    return (      <>        <Navbar />        <section className="section">          <Outlet />        </section>      </>    );  };  export default Home; |

**Index Pages**

The application still does not show content in the **Home** page. If you check the **Home.jsx** file, there is a **Navbar**, and then the **Outlet** **which is the content of the other nested pages**, but no content for **Home**.

To do this, we have to create a another component which we are going to call **SharedLayout** (name optional) and use it in the parent in **App.jsx**. Then, create an **index element** which is essentially the **Home** page (parent page).

Now, the parent component (**Home.jsx**) **will only have its own content**. The **sharing will be done in** **SharedLayout.jsx** component. So basically, what the **Outlet** component will display will depend on the link, or what page you want to display.

|  |
| --- |
| **App.jsx** |
| import { BrowserRouter, Routes, Route } from "react-router-dom";  import Home from "./pages/Home";  import About from "./pages/About";  import Products from "./pages/Products";  import Error from "./pages/Error";  import SharedLayout from "./pages/SharedLayout";  function App() {    return (      <BrowserRouter>        <Routes>          <Route path="/" element={<SharedLayout />}>            <Route index element={<Home />} />            <Route path="about" element={<About />} />            <Route path="products" element={<Products />} />            <Route path="\*" element={<Error />} />          </Route>        </Routes>      </BrowserRouter>    );  }  export default App; |
| **Home.jsx** |
| const Home = () => {    return (      <>        <section className="section">          <h2>Home Page</h2>        </section>      </>    );  };  export default Home; |
| **SharedLayout.jsx** |
| import { Link, Outlet } from "react-router-dom";  import Navbar from "../components/Navbar";  const Home = () => {    return (      <>        <Navbar />        <Outlet />      </>    );  };  export default Home; |
| **Navbar.jsx** |
| import { Link } from "react-router-dom";  const Navbar = () => {    return (      <nav className="navbar">        <Link to="/">Home</Link>        <Link to="/about">About</Link>        <Link to="/products">Products</Link>      </nav>    );  };  export default Navbar; |
|  |

**Active Links**

Let’s see how we can set up **Active Links**, which the **link that is active will have a specific style.** To do that, let’s create another file **StyleNavbar.jsx** (which is going to replace the **Navbar.jsx**)

**NavLink** component has the property isActive (which is provided by React Router) and then based on that value we can set up some styling. By using **NavLink** component instead of Link, a class isActive will be added to the active page component.

We can do this is inline CSS, where is apply conditional rendering to each link:

import { NavLink } from "react-router-dom";

const Navbar = () => {

  return (

    <nav className="navbar">

      <NavLink

        to="/"

        style={({ isActive }) => {

          return { color: isActive ? "red" : "green" };

        }}

      >

        Home

      </NavLink>

      <NavLink

        to="/about"

        style={({ isActive }) => {

          return { color: isActive ? "red" : "green" };

        }}

      >

        About

      </NavLink>

      <NavLink

        to="/products"

        style={({ isActive }) => {

          return { color: isActive ? "red" : "green" };

        }}

      >

        Products

      </NavLink>

    </nav>

  );

};

export default Navbar;

or you can add a class conditionally instead of the style, which is the most common approach.

import { NavLink } from "react-router-dom";

const Navbar = () => {

  return (

    <nav className="navbar">

      <NavLink

        to="/"

        className={({ isActive }) => (isActive ? "link active" : "link")}

      >

        Home

      </NavLink>

      <NavLink

        to="/about"

        className={({ isActive }) => (isActive ? "link active" : "link")}

      >

        About

      </NavLink>

      <NavLink

        to="/products"

        className={({ isActive }) => (isActive ? "link active" : "link")}

      >

        Products

      </NavLink>

    </nav>

  );

};

export default Navbar;

Where the link class has a general style and active has the active link style (red)

.link {

  color: var(--grey-500);

}

.active {

  color: var(--primary-500);

}

**URL parameters**

Let’s see how we can read URL parameters and have some dynamic pages.

Imagine we have a page displaying a bunch of products, and once we click on the product, we display a separate page. Here’s the key: **you don’t want to set up a separate page for every product**. It makes more sense to **set up one page which will act as a placeholder, and then get the data we need to display**.

In order to set up parameters, we want to go with a colon **:** and then the parameter name (which can be anything), which in our case is productId. Note that you can use nested structure for **Products** and **SingleProduct** components, however we won’t use it in the example.

function App() {

  return (

    <BrowserRouter>

      <Routes>

        <Route path="/" element={<SharedLayout />}>

          <Route index element={<Home />} />

          <Route path="about" element={<About />} />

          <Route path="products" element={<Products />} />

          <Route path="products/:productId" element={<SingleProduct />}/>

          <Route path="\*" element={<Error />} />

        </Route>

      </Routes>

    </BrowserRouter>

  );

}

So now whatever we write after <http://localhost:3000/products/> is going to be our productId. You can see that by logging the useParams component in the SingleProduct component, after manually adding “*123*” to the link.

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

const SingleProduct = () => {

  console.log(useParams());

  return (

    <section className="section product">

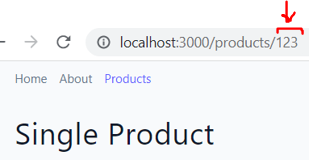
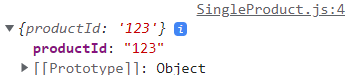
      <h2>single product</h2>

    </section>

  );

};

export default SingleProduct;

That means that in the product page, we will iterate over an array of projects and display them. Then each of these products are going to have a speficic ID as always. We are going to use that ID to dinamically set the Link:

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

import products from "../data";

const Products = () => {

  return (

    <section className="section">

      <h2>products</h2>

      <div className="products">

        {products.map((product) => {

          return (

            <article key={product.id}>

              <h5>{product.name}</h5>

              <Link to={`/products/${product.id}`}>more info</Link>

            </article>

          );

        })}

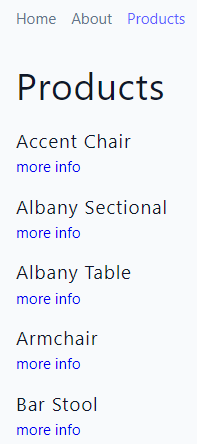
      </div>

    </section>

  );

};

export default Products;



Then in the SingleProduct component we use find() method to find the product in the array of products that matches that ID and fetch the data for that product.

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

import products from "../data";

const SingleProduct = () => {

  const { productId } = useParams();

  const product = products.find((product) => product.id === productId);

  const { image, name } = product;

  return (

    <section className="section product">

      <img src={image} alt={name} />

      <h5>{name}</h5>

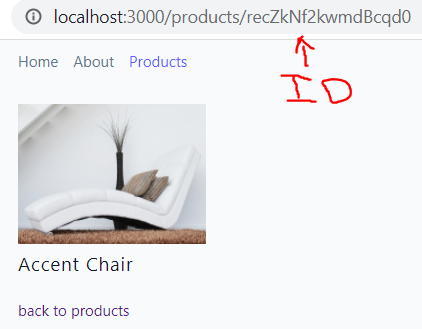
      <Link to="/products">back to products</Link>

    </section>

  );

};

export default SingleProduct;



**useNavigate()**

Another way to navigate to another page is programatically (for example, by submiting a form) is by using **useNavigate()**.

So imagine we have a **login** page, and a dashboard page that displays “Hello, <person’s name>” once the user logs in.

|  |  |
| --- | --- |
|  |  |

First thing to do is add the **login** and **dashboard** pages to the **App.jsx**.

We will also define a user state value in the **App.jsx** file and pass it to the other files (normally you would do this in global context).

The **Login** component would have to indicate where to navigate once the handleSubmit function runs, which is the /dashboard in this case.

import { useState } from "react";

import { useNavigate } from "react-router-dom";

const Login = ({ setUser }) => {

  const [name, setName] = useState("");

  const [email, setEmail] = useState("");

  const navigate = useNavigate();

  const handleSubmit = async (e) => {

    e.preventDefault();

    if (!name || !email) return;

    setUser({ name: name, email: email });

    navigate("/dashboard");

  };

  return (

    <section className='section'>

      <form className='form' onSubmit={handleSubmit}>

        <h5>login</h5>

        <div className='form-row'>

          <label htmlFor='name' className='form-label'>

            name

          </label>

          <input

            type='text'

            className='form-input'

            id='name'

            value={name}

            onChange={(e) => setName(e.target.value)}

          />

        </div>

        <div className='form-row'>

          <label htmlFor='email' className='form-label'>

            email

          </label>

          <input

            type='email'

            className='form-input'

            id='email'

            value={email}

            onChange={(e) => setEmail(e.target.value)}

          />

        </div>

        <button type='submit' className='btn btn-block'>

          login

        </button>

      </form>

    </section>

  );

};

export default Login;

Then the Dashboard would simply grab the state value and display it. Note that we use conditional chain because in the beginning of the program name is **undefined**.

const Dashboard = ({ user }) => {

  return (

    <section className="section">

      <h4>Hello, {user?.name}</h4>

    </section>

  );

};

export default Dashboard;

**Protected Route**

To restrict a certain page, wrap that page in the <ProtectedRoute></ProtectedRoute> component (which is a newly created component).

So for example, let’s say we want to only dispay the Dashboard page if the user exists. We wrap the dashboard component as shown below.

import { useState } from "react";

import { BrowserRouter, Routes, Route } from "react-router-dom";

import Home from "./pages/Home";

import About from "./pages/About";

import Products from "./pages/Products";

import Error from "./pages/Error";

import SharedLayout from "./pages/SharedLayout";

import SingleProduct from "./pages/SingleProduct";

import Dashboard from "./pages/Dashboard";

import Login from "./pages/Login";

import ProtectedRoute from "./pages/ProtectedRoute";

function App() {

  const [user, setUser] = useState(null);

  return (

    <BrowserRouter>

      <Routes>

        <Route path="/" element={<SharedLayout />}>

          <Route index element={<Home />} />

          <Route path="about" element={<About />} />

          <Route path="products" element={<Products />} />

          <Route path="products/:productId" element={<SingleProduct />} />

          <Route path="login" element={<Login setUser={setUser} />} />

          <Route

            path="dashboard"

            element={

              <ProtectedRoute user={user}>

                <Dashboard user={user} />

              </ProtectedRoute>

            }

          />

          <Route path="\*" element={<Error />} />

        </Route>

      </Routes>

    </BrowserRouter>

  );

}

export default App;

Then in the ProtectedRoute component, we grab the children (which is whatever page we want to protect) and set a condition: if there is no user, navigate to **Home** page, else return children (the children in this case).

import { Navigate } from "react-router-dom";

const ProtectedRoute = ({ children, user }) => {

  if (!user) {

    return <Navigate to="/" />;

  }

  return children;

};

export default ProtectedRoute;