Single Page Routing

Up to this point, we have only used a single page to house all of our components. There is absolutely nothing wrong with that if it fits your needs. This design pattern can be problematic however if you have multiple different components that really aren't grouped together. Take a standard social media site for example. Do you want to have the login, signup, post, friends, etc components to ALL be on the screen at the same time? Of course not, that would be a very odd experience. One approach to fix that would be to use a series of `if() else` statements to conditionally render the page based off some criteria. That however is needlessly complex and can also cause problems. Let's assume you have the following page and want the following routes (assume that all of them are pre-pended with the base URL, something like example.com):

1. `/` - The main page about the application etc
2. `/login` - The login component
3. `/signup` - The signup component
4. `/settings` - The user settings once they're logged in.
5. `/user/id` - In this case the id is a user id and could be any value.
6. Etc.

With the above examples, you can see how that would have a TON of logic involved for if else statements. Interestingly enough, that is actually what routing does, just in a more concise way. The beautiful part about SPAs is that you can have these different routes without having the need for all of that extra logic. As mentioned before, one problem with React is that it is a library which means there is no built in functionality to do this. There are however plenty of extra packages that handle this exact functionality. The main package to use is `react-router-dom`.

React Router Dom

React router dom has become the de facto routing library in React. It is (for most uses) very straight forward and easy to use. Let's say you have the above application and want those routes. But let's also suppose that you want a `Header` component AND a `Footer` component to be part of every single component. Basically you want:

- MENU (Header Component)
- BODY (Component changes depending on the route)
- FOOTER (Footer component)

There are two approaches to handle this. The naive approach would be to simply add the `<Header />` and `<Footer />` to EVERY component and then handle the routes to conditionally render the components. That isn't needed however. We'll get into how that would work in a second, but let's take a look at the router itself.

Simple Routes

Declaring Routes

Technically there are multiple other approaches but the main one is outlined below. Because everything in React is expected to be some sort of component, you utilize nested components for routes in much the same way you use them elsewhere in your application. The most straight forward approach simply uses the `Routes`, `BrowserRouter`, and `Route` components that are part of `react-router-dom`. Think of this like a giant switch case statement:

```
// BrowserRouter component is aliased to make life easier
import { BrowserRouter as Router, Routes, Route } from "react-router-dom";

// Other imports / functionality
```

```
<Router>
    <div className="content">
      <Header /> Allows Header to appear on all pages
        <main>
            'Routes' looks through all of its child 'Route'
            elements and renders the first one whose path
            matches the current URL. Use a 'Routes' any time
            you have multiple routes, but you want only one
            of them to render at a time
            <Routes>
              The 'Routes' will render the first route that matches the URL.
              So, if we try to navigate to '/login' or '/signup', it will always
just render '/' instead.
              Using the 'exact' parameter means it will only return '/' if the
path is an exact match
              <Route exact path="/" element={<h1>Some Title</h1>} />
              <Route path="/login" element={<Login prop1={value}/>} />
              <Route path="/signup" element={<Signup prop1={value}/>} />
            </Routes>
        </main>
    </div>
    <Footer /> Allows Footer to appear on all pages
</Router>
```

It is important to note that you have to have what you want to render inside of the `element` attribute. There are several more attributes which you can take a look at in the documentation.

Adding Links

You have two options on how to use links. One option is to render something that the user can click on. For that you can use a `Link` (or `NavLink`):

- Option 1: Rendered Links a user can click

```
// The below could be used in ANY component you need routing in
import { Link } from "react-router-dom";

//... other code
```

```
// In the JSX
<ul>
  <li>
    <Link to="/">Home</Link>
  </li>
  <li>
    <Link to="/login">Login</Link>
  </li>
  <li>
    <Link to="/signup">Signup</Link>
  </li>
</ul>;
```

- Option 2: As an event handler in the component logic. This can be used to redirect someone beind the scenes without their direct interaction. What are some possible reasons for that?

```
// The below can be in any component you need links to. This one works well in
some cases but not in others. It depends on what you want for styling or logic
import { useNavigate } from "react-router-dom";
//...

// in component logic
const navigate = useNavigate();

// In JSX
<button onClick={() => navigate("/login")}>
  Login
</button>;
```

Variable Routes (Route Parameters)

As we mentioned above, sometimes you might need some sort of parameter in your route. While something like `/login` is perfect as you can render the exact same component ANY time someone goes to the login route, what about `/user/SOME_USER_ID`? That user id might change and while you might want to always render the user component, the data displayed will need to change whether it is `/user/2` or `/user/3`. That is where route parameters come into play. Luckily they are SUPER easy to use:

```
// DECLARING THE ROUTE TO USE A PARAMETER
// In JSX
<Route path="/user/:id" element={<User />} />;

// In the above, the `:id` could technically be anything but if you want it to be
a parameter, it has to start with a colon

// ACCESSING THE ROUTE PARAMETER IN THE COMPONENT
// Inside the user component, if you're at /user/2 or /user/3 you need to grab
that id parameter. That is easily done
```

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

export default function User() {
  // This uses object destructuring to pull out the id param.
  // The id variable could now be used wherever needed
  let { id } = useParams();

  // ... JSX
}
```

Everything Together

Now that you can use Links and routes together, you can mix and match them in any different components you might need to. One thing that is important to note though is think about what you might want on ALL of your pages or things that are consistent across multiple components. This might impact how you design your application as a whole.

Here's an example application with everything together. First, set up your different files that contain the different components that will be rendered depending on the current URL:

```
// Home.js

const Home = () => {
  return <h1>Home</h1>;
};

export default Home;

// About.js

const About = () => {
  return <h1>About</h1>;
};

export default About;

// Login.js

const Login = () => {
  return <h1>Login</h1>;
};

export default Login;
```

Then, set up your routes using Router and Routes

```
// App.js
import React from 'react';
import { BrowserRouter as Router, Routes, Route, Link } from 'react-router-dom';
import Home from './Home';
import About from './About';
import Login from './Login';

const App = () => {
  return (
    <Router>
      <div>
        <nav>
          <ul>
            <li>
              <Link to="/">Home</Link>
            </li>
            <li>
              <Link to="/about">About</Link>
            </li>
            <li>
              <Link to="/login">Login</Link>
            </li>
          </ul>
        </nav>

        <Routes>
          <Route path="/" exact element={<Home />} />
          <Route path="/about" element={<About />} />
          <Route path="/login" element={<Login />} />
        </Routes>
      </div>
    </Router>
  );
};

export default App;
```

Extras

There are a lot more options inside of React Router Dom. Let's take a look at the docs real quickly to see more information:

- [Nested Routing]:

```
// somewhere up the tree
<Routes>
  <Route path="/users/*" element={<Users />} />
</Routes>;

// and now deeper in the tree
function Users() {
  return (
    <div>
      <h1>Users</h1>
      <Routes>
        <Route path="account" element={<Account />} />
      </Routes>
    </div>
  );
}
```

Use this if you want a route like `/user` which renders a sub component and then has sub-routes like `/user/settings` and `/user/friends` etc.