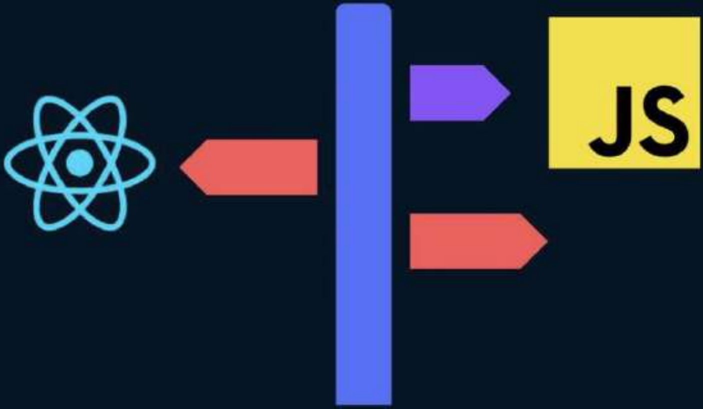
@pyplane_code

React Router Dom



Introduction

In this tutorial we will implement dynamic routing to our simple React js application using React Router Dom version 6

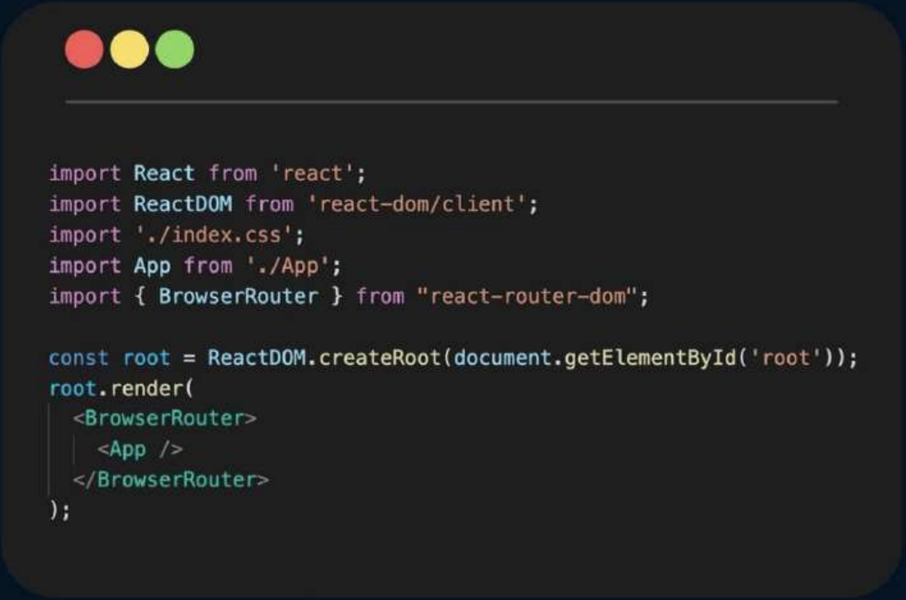
We will cover:

- basic routes
- accessing detail pages
- 2 types of nested routes
- dealing with authentication
- and more!

Let's get started:

- npx create-react-app myApp
- npm install react-router-dom@6

To get started open the index.js file and import Browser Router which will wrap our entire application providing the HTML5 history API (pushState, replaceState, popstate) for the synchronization of the UI and URL



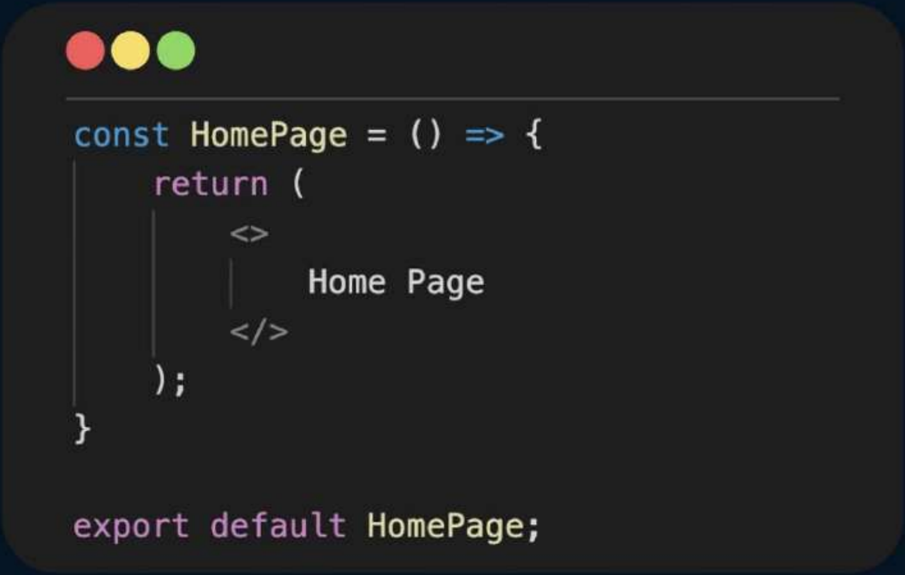
```
import React from 'react';
import ReactDOM from 'react-dom/client';
import './index.css';
import App from './App';
import { BrowserRouter } from "react-router-dom";

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(
  <BrowserRouter>
    <App />
  </BrowserRouter>
);
```

Next create a pages folder in the src directory.

Inside create pages:

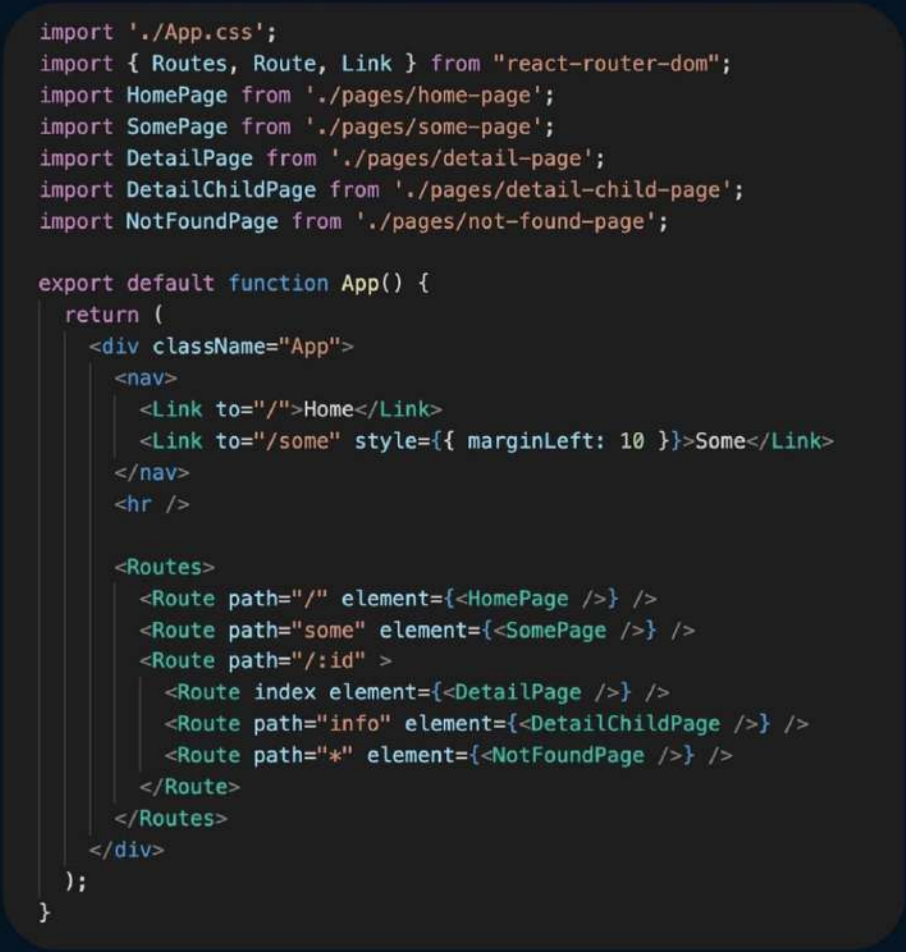
- home-page.js
- some-page.js
- detail-page.js
- child-detail-page.js
- not-found-page.js

Each page will look in the same way except the detail page



```
const HomePage = () => {
    return (
        <>
            Home Page
        </>
    );
}

export default HomePage;
```

For this simplified example we will place our navigation and routes in the same App.js file

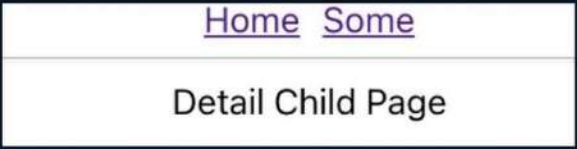


```
import './App.css';
import { Routes, Route, Link } from "react-router-dom";
import HomePage from './pages/home-page';
import SomePage from './pages/some-page';
import DetailPage from './pages/detail-page';
import DetailChildPage from './pages/detail-child-page';
import NotFoundPage from './pages/not-found-page';

export default function App() {
  return (
    <div className="App">
      <nav>
        <Link to="/">Home</Link>
        <Link to="/some" style={{ marginLeft: 10 }}>Some</Link>
      </nav>
      <hr />

      <Routes>
        <Route path="/" element={<HomePage />} />
        <Route path="some" element={<SomePage />} />
        <Route path="/:id" >
          <Route index element={<DetailPage />} />
          <Route path="info" element={<DetailChildPage />} />
          <Route path="*" element={<NotFoundPage />} />
        </Route>
      </Routes>
    </div>
  );
}
```

If we click on the the "some" link we will be taken to the some page. To display the detail page we need to go to for example localhost:3000/1/. Let's check if the id is numeric and if it's not let's return to the home page

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

const DetailPage = () => {
    const navigate = useNavigate()
    const params = useParams()

    const { id } = params
    const isInteger = /^-?[0-9]+$/.test(id + '');

    useEffect(() => {
        if (!isInteger) {
            navigate("/")
        }
        // eslint-disable-next-line
    }, [])

    return (
        <>
            Detail Page id: {id}
        </>
    );
}

export default DetailPage;
```

Now to see the info page as a child of the detail page we need to go to localhost:3000/1/info and we will see:

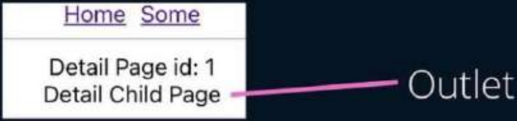

Home Some

Detail Child Page

However if we would like to nest the info (child page) into the detail page we would need to use the Outlet component. Before we do that let's modify our Routes in the App.js

```
<Routes>
  <Route path="/" element={<HomePage />} />
  <Route path="some" element={<SomePage />} />
  <Route path="/:id" element={<DetailPage />} >
    <Route path="info" element={<DetailChildPage />} />
    <Route path="*" element={<NotFoundPage />} />
  </Route>
</Routes>
```

Now let's add the Outlet component to our detail page (parent component). This component represents where the child component will be rendered when we access the cihld url



Home Some

Detail Page id: 1
Detail Child Page

```
import { useEffect } from "react";
import { Outlet, useNavigate, useParams } from "react-router-dom";

const DetailPage = () => {
    const navigate = useNavigate()
    const params = useParams()

    const { id } = params
    const isInteger = /^-?[0-9]+$/.test(id + '');

    useEffect(() => {
        if (!isInteger) {
            navigate("/")
        }
        // eslint-disable-next-line
    }, [])

    return (
        <div>
            Detail Page id: {id}
            <br />
            <Outlet />      ——— Outlet
        </div>
    );
}

export default DetailPage;
```

The final slide is about authentication. Let's assume we are using the Context API and let's create ther RequireAuth Component

```
const RequireAuth = ({ children, redirectTo }) => {
  const authCtx = useContext(AuthContext)
  return authCtx.isLoggedin ? children : <Navigate to={redirectTo} />;
}
```

Now we can use it in our Routes like this:

```
<Route
      path="/"
      element={
        <RequireAuth redirectTo="/login">
          <HomePage />
        </RequireAuth>
      }
    />
```

If we are not logged in we will be taken to the login page. In other case we will see the home page