Single page Application

**React Router**

# What is Routing?

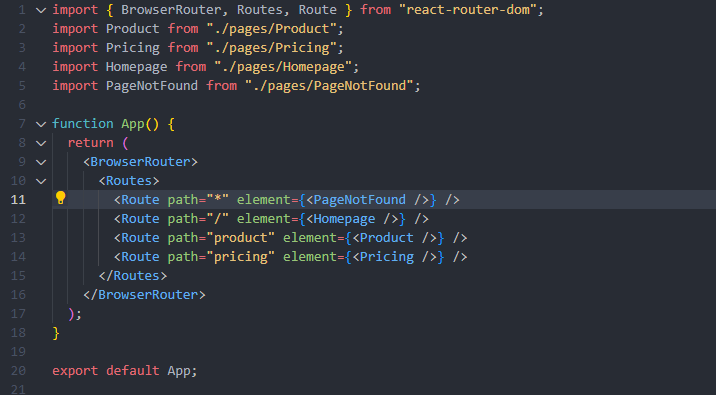
* We match different URLs to different Views (components)
  + A match is called **ROUTE**
* The user can navigate between different application screens, using the URL
* This works on the ‘**CLIENT-SIDE**’
* React Relies on 3rd party package
  + **React Router**
* Routing is **FUNDAMENTAL**

# What is SPA (single Page app)?

* Application that is **EXECUTED ENTIRELY on the CLIENT (browser)**
* SPA’s rely on **ROUTES**
* When the user Clicks on a **Router Link**
  + The **URL CHANGES**
  + The **DOM IS UPDATED**
  + **JS (react) is updating the DOM**
* The Page is **NEVER RELOADED**
* Feels like a NATIVE APP
* The SPA can also **COMMUNICATE with a SERVER through API**

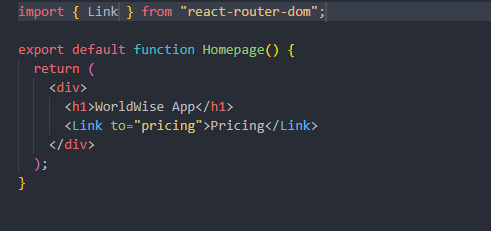
# React Router Library

* Install it in the project using
  + **Npm I react-router-dom**
  + **Npm I react-router-dom@6** – to install VERSION 6 of React Router
* We will import and define the **BrowserRouter**
* We import and define the **Routes**
* We import and define a **Route**
  + We need to define the **path = “desired path”**
    - Path = “product”
  + We need to define the element, the **React Component** for that route
    - Element={<Product/>}
* Now we can access the ‘/product’ page
* Use **path=”\*”**  for any other route that is not in the project

****

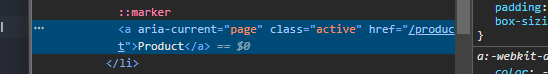
# Linking between Routes

* we are not using **<a>** elements since this will **reload the** whole page
* WE ARE USING the **LINK** element
  + <Link to=”pricing”> Pricing </Link>

****

* We have created a new component that will only handle the Navigation

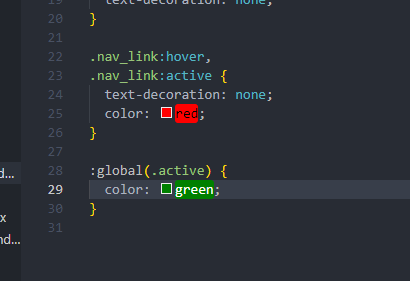
With **NavLink**, React Router gives us the **Active** class into the element so we can use it



# Ways to STYLE a REACT APP

1. Inline CSS
   1. Straight to the JSX element itself
      1. <div style={{width:’80’ display:’flex’}}></div>
   2. Scoped only to this element
2. External CSS
   1. className prop
   2. entire app
   3. A nightmare in large apps
   4. Never used in large apps
3. CSS Modules
   1. We write a CSS file for each component
   2. One external file per component
4. Utility-first CSS
   1. Tailwind
      1. Define utility classes
      2. And define whatever CSS you want without leaving the JSX
5. No CSS
   1. Use predefined UI libraries
      1. MUI
      2. Chakra UI
      3. Mantine

# CSS Modules

* A library
* Comes out of the box with Vite and Create React App
* 1 External css file / component
  + NameOfTheComponent.module.css
* We cannot use html element only
  + Using straight html elements will SELECT ALL ELEMENTS FROM THE APP
* We have to use **CLASSES** 
  + **.navigation**
* Import it using
  + Import styles from ‘./filename.modules.css’
* Use is as a JS variable
  + <element className={styles.nav}/>
* From a CSS Module, we can set up a **GLOBAL**  class
  + :global(.test){
  + Css styles
  + }
  + 

# Nested Routes

* We need nested routes when we want a part of the UI to be controlled by a part of the URL
* Nested Routes 🡺 /app/cities || /app/countries || /app/form

## Declaring Nested Routes

* We are declaring Nested Routes inside our **ROUTE** element from the main component
* We use the <Route></Route> opened and closed element and we define child routes
  + A screen shot of a computer code

    Description automatically generated

## Outlet Element - <Outlet/>

* This is the element that we are using to display the NESTED ROUTE elements

A computer screen shot of a computer code

Description automatically generated

# Set Default Path or Default Nested path

* To set a default path, we use the **index** attribute instead of the **path att**
  + <Route index element={<List/>}/>
* We have to CREATE A DEFAULT
* A screen shot of a computer code

  Description automatically generated

# Storing state in the URL

* Easy and accessible to all components in the APP – **IT’s GLOBAL**
* A good way to ‘pass’ data from one page to another
* Makes it possible to bookmark / share the page with the exact UI state it had at the time

# Storing State in URL With React Router

* We use 2 types of variables
  + **Params**
  + **Query Strings**

# Steps to use PARAMS

1. Create a new Route
   1. 
2. Link to the Route
   1. Add the <Link> element to the item that we will click on like this
   2. We attach only the ID, without the /
   3. A screen shot of a computer code

      Description automatically generated
3. Read the state
   1. We use the useParams hook in the component where we want to use the data from URL
   2. A screen shot of a computer screen

      Description automatically generated

# Storing State in URL with Query string

* We need top add it to the path where the user will go to in the **Link**  element
* A screen shot of a computer code

  Description automatically generated

## useSearchParams()

* We use this Hook to read the URL query string from the URL
* Just like the useState, it returns an array of 2 values
  + The current location
  + A function to update the query string



* To get the query strings, we need to call the **get()** method on the 1st value
  + urlState.get(‘lat’)

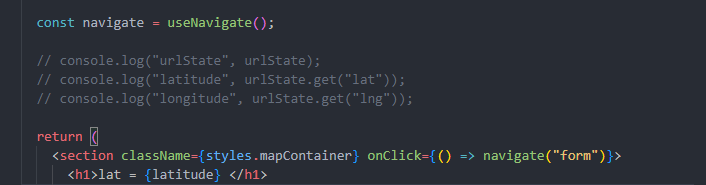
A screen shot of a computer code

Description automatically generated

# One of the biggest advantages of storing the state in URL is that you can share the URL and another person can see exactly what you are seeing, with the same STATE

# useNavigate() hook

* this is used to navigate programmatically, without the user clicking on something
* Is actually used usually after the user submits a form

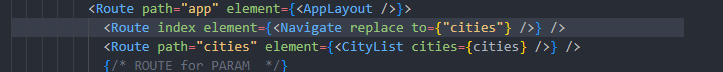
In our practice app, we are using it when a user Clicks on the Map, to navigate to the form component, but without the <Link> element

* Using navigate(-1) we move back 1 step (just like a back button)
* A screen shot of a computer program

  Description automatically generated
* We can also do navigate(-) or navigate (1)
  + Back or forward navigation

# <Navigate /> Component

* The declarative way to Navigate Programatically



* We can use <Navigate replace to={…}/> to Navigate Programatically to a specific path
  + Like we have used it, we are going straight to /app/cities