



Front End Developer

Técnicas de Programação Avançada para a Web

Sara Monteiro

sara.monteiro.prt@msft.cesae.pt

React: os estados

Como podemos verificar, o react não actualiza o nosso DOM directamente.

Isto deve-se ao facto do react usar um DOM Virtual criado para nos dar mais funcionalidades dentro da nossa framework. Desta forma, ele só actualiza o DOM do Browser quando detecta uma alteração ao estado de algo. Aí entrará a nossa funcionalidade *State*.

Gerir o estado

Para gerir os estados e informarmos o nosso Browser que algo mudou, usaremos a função do react useState.

```
import { useState } from 'react'
```

[documentação](#)

Gerir o estado

O `useState` é considerado um *Hook* do React. *Hooks* são todas as funções do React que começam por `use` e nos dão ferramentas para aceder aos estados e ciclo de vida da aplicação.

Existem duas regras para os *Hooks*:

- Só podem ser alterados dentro das funções de componentes.
- Devem ser chamados logo no início do componente.

```
function App() {  
  useState();  
  
  const user = {  
    name: '',  
  }  
}
```

[Documentação Hooks](#)

Gerir o estado

Passemos agora para a aplicação prática do uso dos Estados

O useState permite-nos armazenar num array dois valores através da forma: **const[state, setState] = useState(estadoInicial).**

No caso da nossa alteração de conteúdo, seria:

Valor Inicial
Função de
actualização do
estado

Valor no momento

```
const [selectedTopic, setSelectedTopic] = useState('Por favor  
carrega num botão');
```

Gerir o estado

```
function handleSelect(selectedBtn){  
  setSelectedTopic(selectedBtn);  
}
```

```
<TabButton onSelect={() => handleSelect('sql')}>Mat  
  TabButton<  
    <div>  
      {selectedTopic}  
    </div>  
  </menu>  
  
<Login onSelect={() => handleCreateUser('Sara')}>C
```

Gerir o estado

Demos então o primeiro passo importante para tornar as nossas aplicações dinâmicas.

Não só a abordagem é eficiente como nos permite tratar os estados dentro de cada componente, abrindo o caminho da interactividade!

Criar um componente



Exercício:

Tendo em conta o seguinte bloco de código faça com que, ao clicar, o valor seja actualizado para 75€

```
export default function Discount() {  
  return (  
    <div>  
      <p data-testid="price">$100</p>  
      <button>Apply Discount</button>  
    </div>  
  );  
}
```


Gerir o Estado

Já sabemos usar o estado para actualizar o DOM, iremos agora fazer um exemplo mais aprofundado e perto da realidade. No meu git irão encontrar o ficheiro coreConcepts para nos dar uma base de dados.

Preparemos a nossa tab de content para os dados:

```
<TabButton onSelect={() =>
  TabButton>
</menu>
<div id='tab-content'>
  <h3></h3>
  <p></p>
  <pre>
    <code></code>
  </pre>
</div>
```

Gerir o Estado

Em seguida actualizamos o código para receber os dados do nosso objecto:

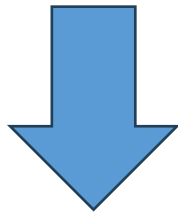
```
const [selectedTopic, setSelectedTopic] = useState('components');
```

```
<menu>
<TabButton onSelect={() => handleSelect('jsx')}>JSX</TabButton>
<TabButton onSelect={() => handleSelect('props')}>Props</TabButton>
<TabButton onSelect={() => handleSelect('state')}>State</TabButton>
</menu>
<div id='tab-content'>
  <h3>{EXAMPLES[selectedTopic].title}</h3>
  <p>{EXAMPLES[selectedTopic].description}</p>
</div>
```

Props e condicionais

Neste momento quando iniciamos o nosso browser ele carrega por defeito os componentes. E no caso de nós querermos mostrar conteúdo **apenas** quando o user seleciona algo?

```
const [selectedTopic, setSelectedTopic] = useState('components');
```



```
const [selectedTopic, setSelectedTopic] = useState();
```

Props e condicionais : ternários

Surge então o conceito de condicionais em React, que será usado neste exemplo para imprimir algo caso tenhamos um selectedTopic.

```
{!selectedTopic ?( <p>Pf selecciona um tópico:</p>) :  
( <div id='tab-content'>  
  <h3>{EXAMPLES[selectedTopic].title}</h3>  
  <p>{EXAMPLES[selectedTopic].description}</p>  
  <pre>  
    <code>{EXAMPLES[selectedTopic].code}</code>  
  </pre>  
</div>)}</pre>
```

[Ternário em JS](#)

Props e condicionais : o &&

Usando o && só irá imprimir o código caso exista selectedTopic.

```
{selectedTopic && (<div id='tab-content'>
  <h3>{EXAMPLES[selectedTopic].title}</h3>
  <p>{EXAMPLES[selectedTopic].description}</p>
  <pre>
    <code>{EXAMPLES[selectedTopic].code}</code>
  </pre>
</div>)}
You, 15 minutes ago • condicionais
```

Props e condicionais : variáveis

Por último, podemos armazenar o conteúdo dentro de uma variável.

```
let tabContent = <p>Pf selecciona um tópico:</p>
if(selectedTopic){
  tabContent = ( <div id='tab-content'>
    <h3>{EXAMPLES[selectedTopic].title}</h3>
    <p>{EXAMPLES[selectedTopic].description}</p>
    <pre>
    <code>{EXAMPLES[selectedTopic].code}</code>
    </pre>
    </div>);
}
```

```
TabButton>
</menu>
<div id='tab-content'>
  {tabContent}
</div>
```

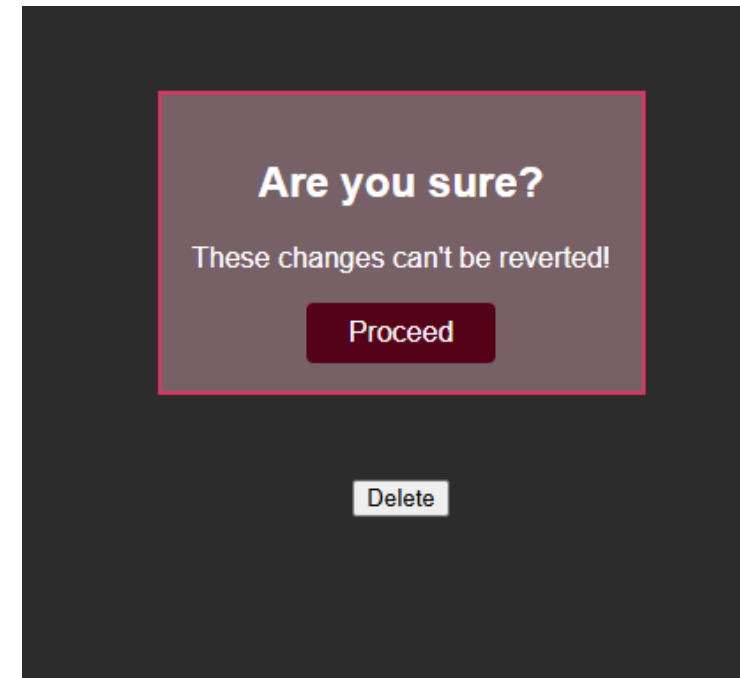
Condições



Exercício:

Usando o ficheiro delete.jsx, faça com que, ao clicar no delete, a div de alerta apareça.

Nota: A div de alerta deve iniciar escondida e, quando aparece e é clicada, desaparece.



Estilos Dinâmicos

Neste momento não há uma indicação visual de qual tab está clicada. Nas aplicações “reais” o estilo do botão fica activo quando o conteúdo correspondente está selecionado. Como proceder então?

Como já vimos, existe em react um atributo chamado `className` que nos permite ter classes dinâmicas.

Estilos Dinâmicos

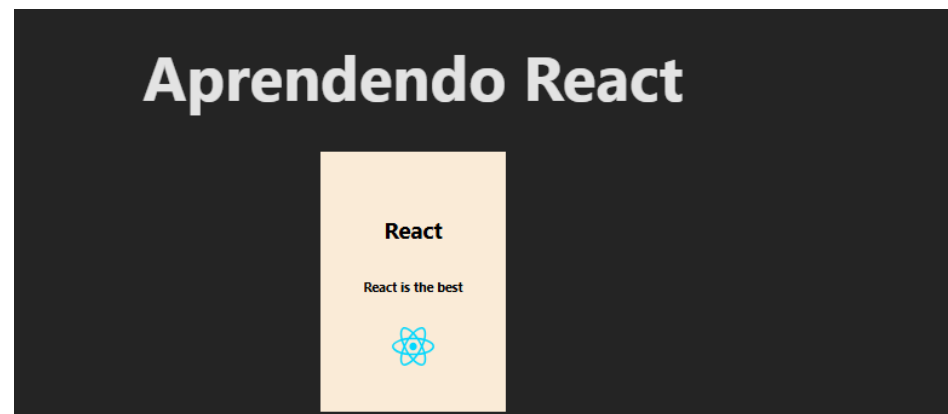
```
✓ export function TabButton({children, onSelect, isActive}){  
  return <button className={isActive? 'active': ''} onClick=  
    {onSelect}>{children}</button>  
}
```

```
<TabButton isActive={selectedTopic == 'jsx'} onSelect={() => handleSelect('jsx')}>JSX</TabButton>  
<TabButton isActive={selectedTopic == 'props'} onSelect={() => handleSelect('props')}>Props</TabButton>  
<TabButton isActive={selectedTopic == 'state'} onSelect={() => handleSelect('state')}>State</TabButton>  
</menu>
```

Listas dinâmicas

A última coisa a aprender em relação ao manuseamento de propriedades em react será iterar e mostrar listas de dados nos nossos componentes: arrays, etc.

Iremos então criar um componente de cartões de componentes que se repetirá conforme o que temos no objecto CORE_CONCEPTS e colocá-lo a seguir ao Header, em substituição dos links que estão actualmente.



Listas dinâmicas

```
✓ import reactLogo from '../assets/react.svg'  
  import '../components/ComponentsCard.css'
```

```
✓ export default function ComponentsCard(props){  
  ✓ return(  
  ✓   <div className="c-card">  
    <h3>{props.title}</h3>  
    <h6>{props.description}</h6>  
    <img src={reactLogo} alt="" />  
  </div>  
  )  
}
```

```
✓ .c-card {  
  padding: 2em;  
  width: 20%;  
  background-color: antiquewhite;  
  color: black;  
  margin: 0px auto 0px auto;  
}
```

```
✓ .container {  
  display: flex;  
  flex-wrap: wrap;  
  gap: 16px;  
}
```

Listas dinâmicas

```
<Header/>  
<div className='container'>  
  {CORE_CONCEPTS.map((item) =>  
    <ComponentsCard  
      key = {item.title}  
      {...item}  
    />  
  )}  
</div>  
  
<CourseGoal {...courseInfo} />
```

Listas dinâmicas

Aprendendo React

Components

The core UI building block - compose the user interface by combining multiple components.



JSX

Return (potentially dynamic) HTML(ish) code to define the actual markup that will be rendered.



Props

Make components configurable (and therefore reusable) by passing input data to them.



State

React-managed data which, when changed, causes the component to re-render & the UI to update.



Lista



Exercício:

Defina um array de objectos de lista de compras com quantidade e item. Crie um componente que itere o array e replique o layout de cada item.

Lista de Compras

batatas - 5kg

pães - 2

chocolate - 1 embalagem

Rotas

Conforme a nossa aplicação vai crescendo, surge a necessidade de dividir as funcionalidades por várias páginas e não estar a apresentar o conteúdo todo na Homepage.

Surge então o conceito de Rotas, que é usado tanto nas aplicações *ClientSide* como nas *ServerSide* para carregar um url diferente da Homepage.

Rotas de páginas de Single Page Applications

É importante indicar que, apesar de irmos acrescentar páginas à nossa aplicação, a navegação será na mesma uma navegação de ClientSide e, logo, não irá haver novos carregamentos de dados.

Tecnicamente o nosso Website continuará a ser uma SPA com múltiplas páginas, mas com diferentes urls.

Para criar estas funcionalidades iremos instalar um pacote chamado React Router.

Routing: o que é?

- é o processo pelo qual uma aplicação web usa o URL atual do navegador para decidir que conteúdo mostrar ao utilizador. Por exemplo, um utilizador que acede à página `/wiki/Node.js` na Wikipedia espera ver algo diferente da página `/wiki/React_(JavaScript_library)`, mas o base é o mesmo layout.
- o **routing** possibilita uma experiência de utilizador mais rica e envolvente, organizando o conteúdo da aplicação

Routing: o que é?

<https://codecademy.com/articles?search=node>

Protocol

Domain

Path

Query

React Router

```
EndDeveloper\SJM_24\TAP\TAPTests\OwningReactTAP> npm install react-router-dom
```

```
added 3 packages, and audited 262 packages in 3s
```

```
100 packages are looking for funding  
  run `npm fund` for details
```

```
found 0 vulnerabilities
```

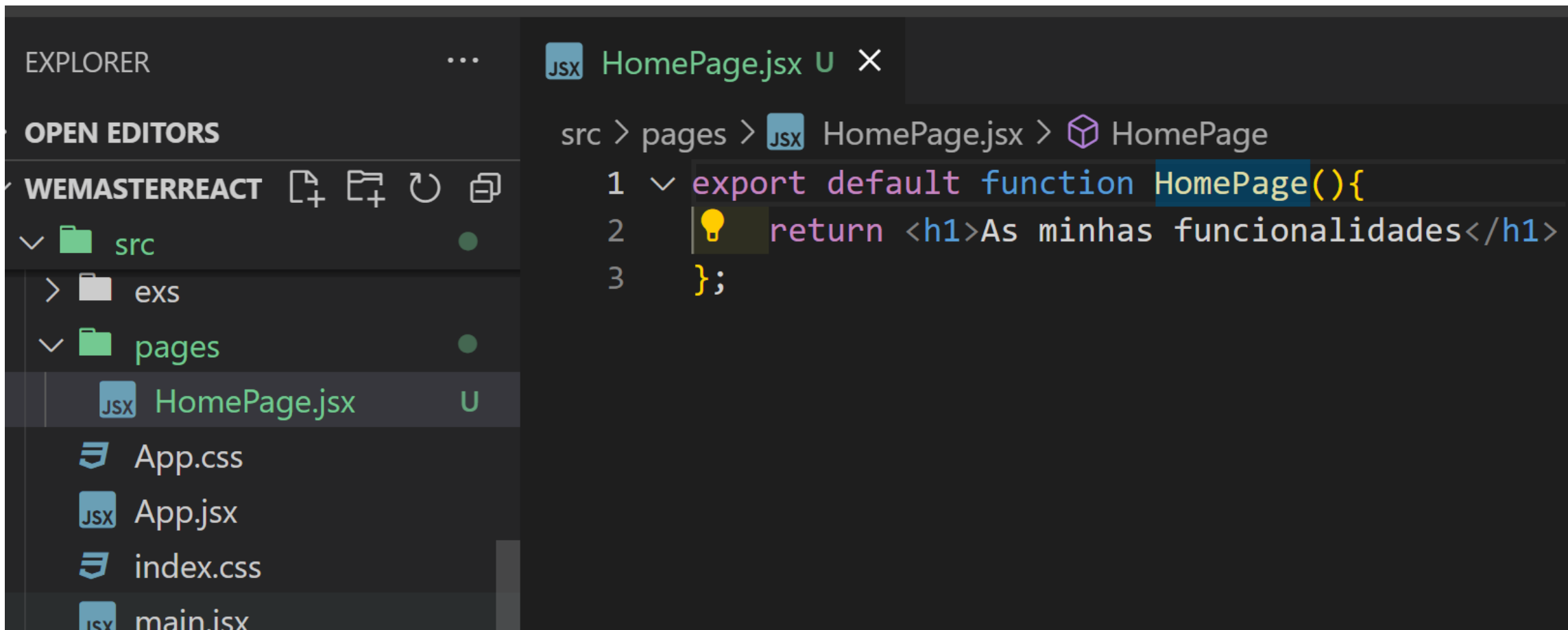
```
PS C:\Users\Utilizador\OneDrive - CESAE\FrontEndDeveloper\SJM_24\TAP\TAPTests\OwningReactTAP>
```

ReactRouter

React Router: definir Rotas

1. Definir as rotas que queremos e que página irá ser carregados em cada uma. Dentro da página iremos carregar componentes.
2. Activar a rota seleccionada e carregar as definições para a mesma
3. Configurar a navegação entre as várias rotas

React Router



The screenshot shows the Visual Studio Code interface. On the left, the Explorer sidebar shows the project structure: a folder named 'WEMASTERREACT' containing a 'src' folder, which in turn contains 'exs' and 'pages' folders. The 'pages' folder is expanded, showing 'HomePage.jsx' as the selected file. The Open Editors sidebar on the right shows the same file. The main editor area displays the code for 'HomePage.jsx' with the following content:

```
src > pages > JSX HomePage.jsx > 🏠 HomePage
1  export default function HomePage(){
2    return <h1>As minhas funcionalidades</h1>
3  };
```

React Router

```
import HomePage from './pages/HomePage'
import IndexShopping from './pages/IndexShopping'

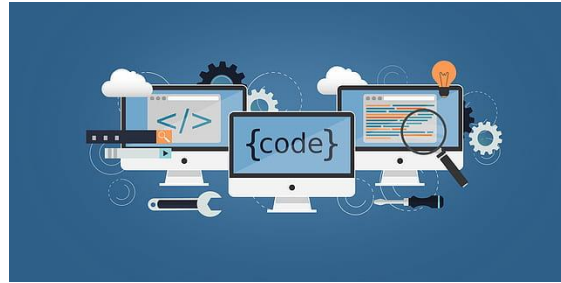
const router = createBrowserRouter([
  {path: '/', element: <HomePage/>},
  {path: '/shopping-list', element: <IndexShopping/>}
]);

function App() {
  return <RouterProvider router= {router}/>
```

React Router

As minhas funcionalidades

Condições



Exercício:

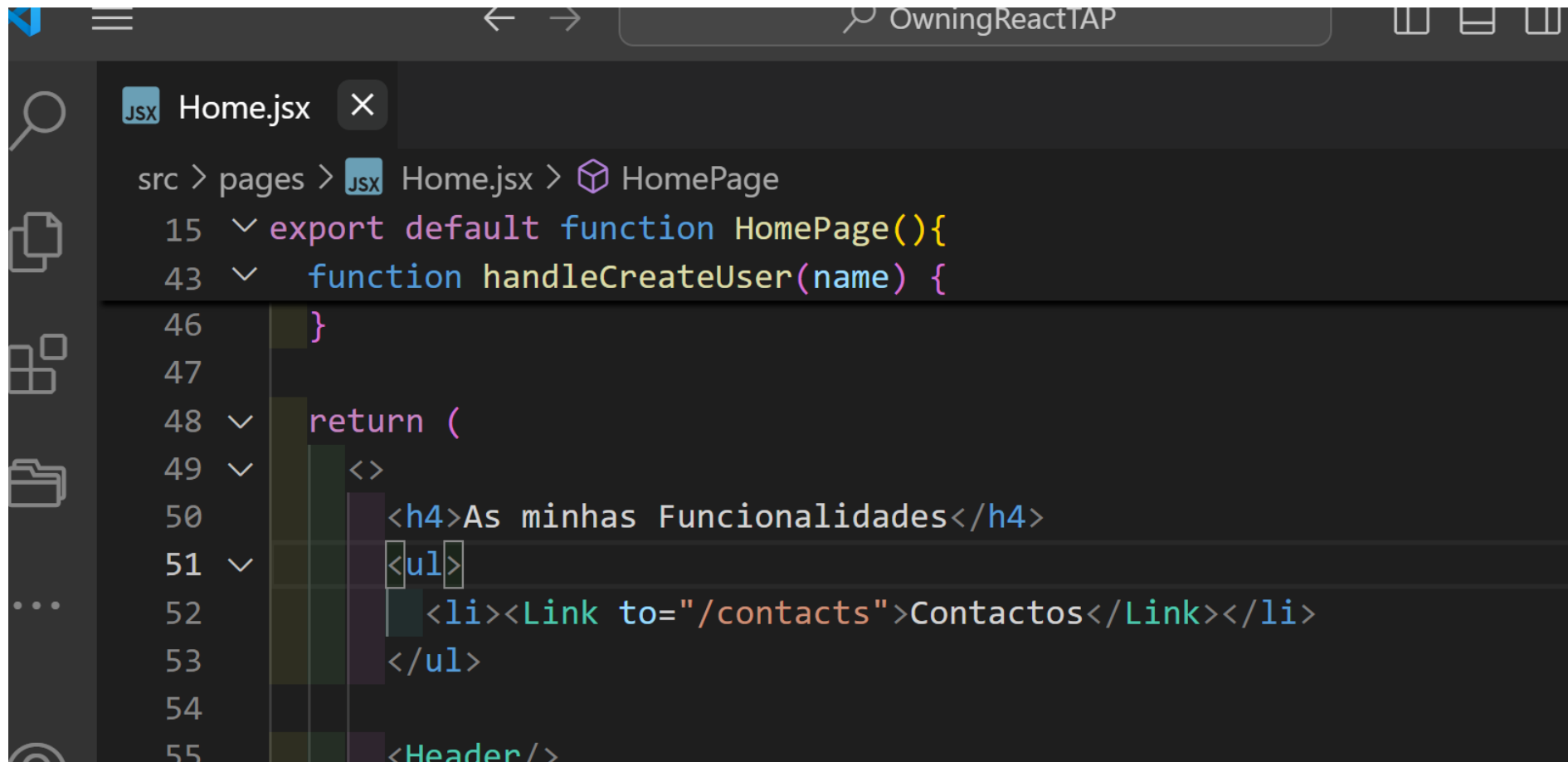
Usando a lógica das rotas, crie uma rota de Contactos que retorne uma página com os seus dados carregados através de um objecto: Nome, profissão e email.

React Router: sintaxe alternativa

```
✓ const routeDefinitions = createRoutesFromElements(  
  ✓ <Route>  
    <Route path= '/' element= {<HomePage/>}/>  
    <Route path= '/contacts' element= {<Contact/>}/>  
  </Route>  
);  
  
const router = createBrowserRouter(routeDefinitions)  
✓ function App() {  
  return <RouterProvider router={router}/>;  
}  
  
export default App
```

React Router: navegação entre páginas

Nota:
A navegação com
<a><a/> também
funciona, mas as
transições entre páginas
carregam do servidor e
não ficam tão fluídas.



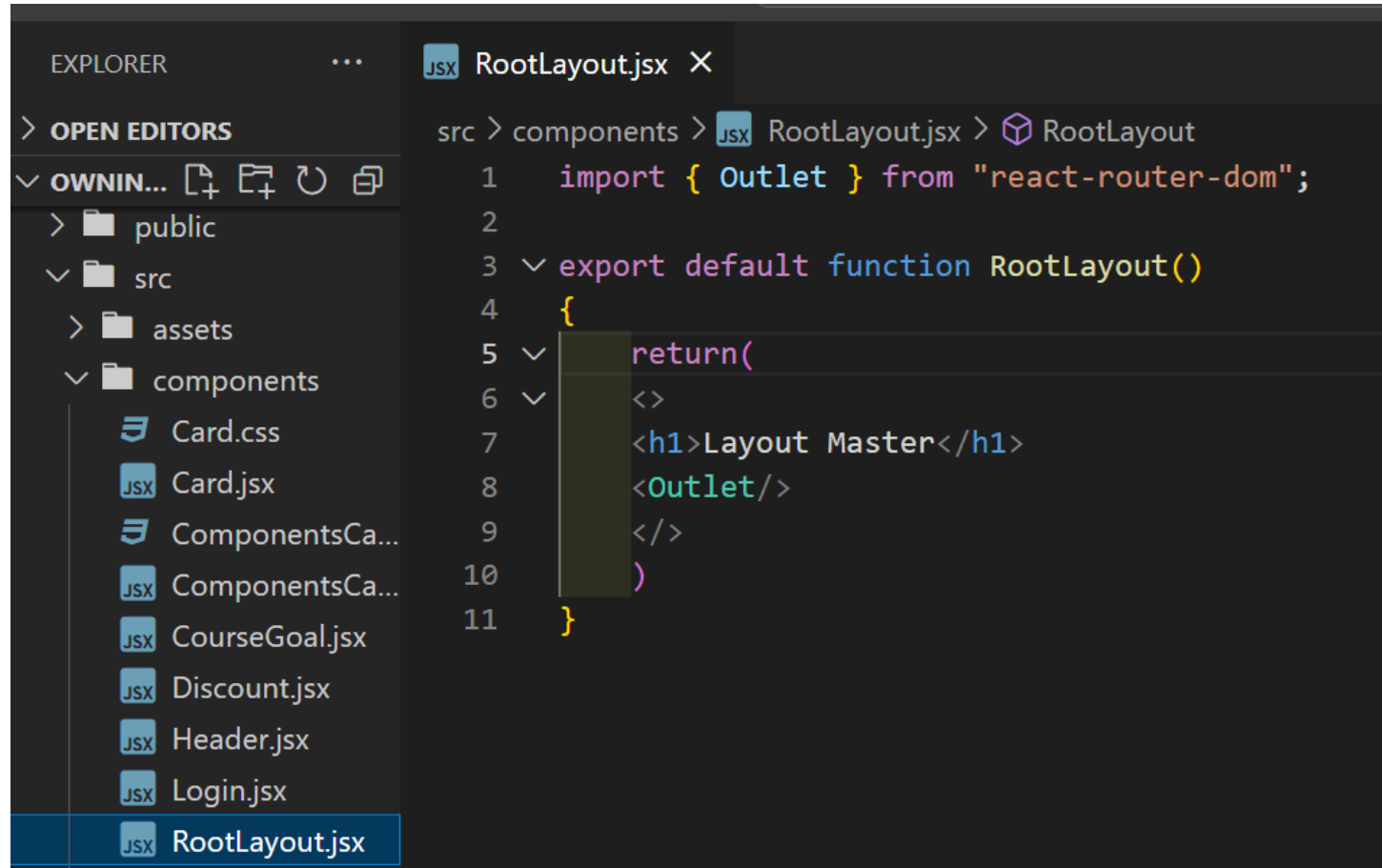
```
src > pages > Home.jsx > HomePage
15  export default function HomePage(){
43  function handleCreateUser(name) {
46  }
47
48  return (
49  <>
50    <h4>As minhas Funcionalidades</h4>
51    <ul>
52      <li><Link to="/contacts">Contactos</Link></li>
53    </ul>
54
55    <Header/>
```

React Router: Layouts e Nested Routes

E se quisermos criar um layout comum a todas as rotas? Por exemplo um footer e uma navbar?

Criaremos componentes que irão ser renderizados em todas as rotas. Estes componentes são vistos como um layout master que abriga as rotas que definirmos.

React Router: Layouts e Nested Routes



```
EXPLORER
> OPEN EDITORS
OWNIN...
> public
src
  assets
  components
    Card.css
    Card.jsx
    ComponentsCa...
    ComponentsCa...
    CourseGoal.jsx
    Discount.jsx
    Header.jsx
    Login.jsx
    RootLayout.jsx

src > components > RootLayout.jsx > RootLayout
1  import { Outlet } from "react-router-dom";
2
3  export default function RootLayout()
4  {
5    return(
6      <>
7        <h1>Layout Master</h1>
8        <Outlet/>
9      </>
10    )
11  }
```

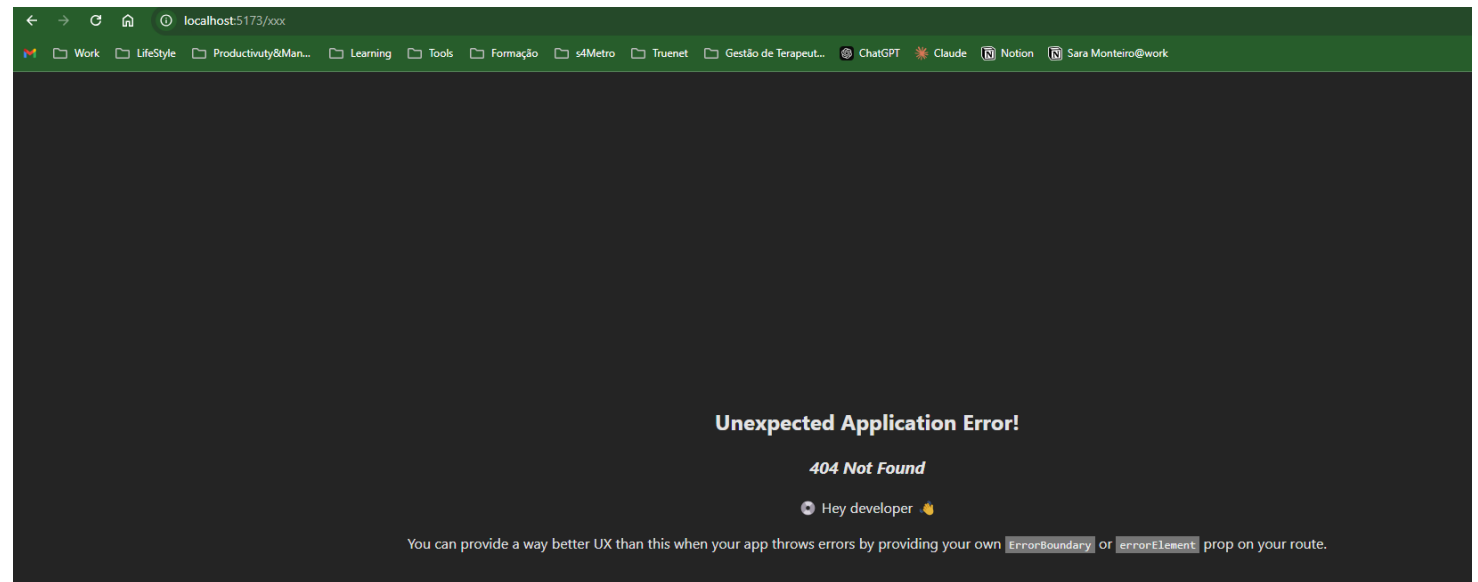
React Router: Layouts e Nested Routes

```
const router = createBrowserRouter([
  { path: '/',
    element: <RootLayout/>,
    children: [
      {path: '/',element: <HomePage/> },
      {path: '/contacts',element: <Contact/> }
    ],
  },
]);
```

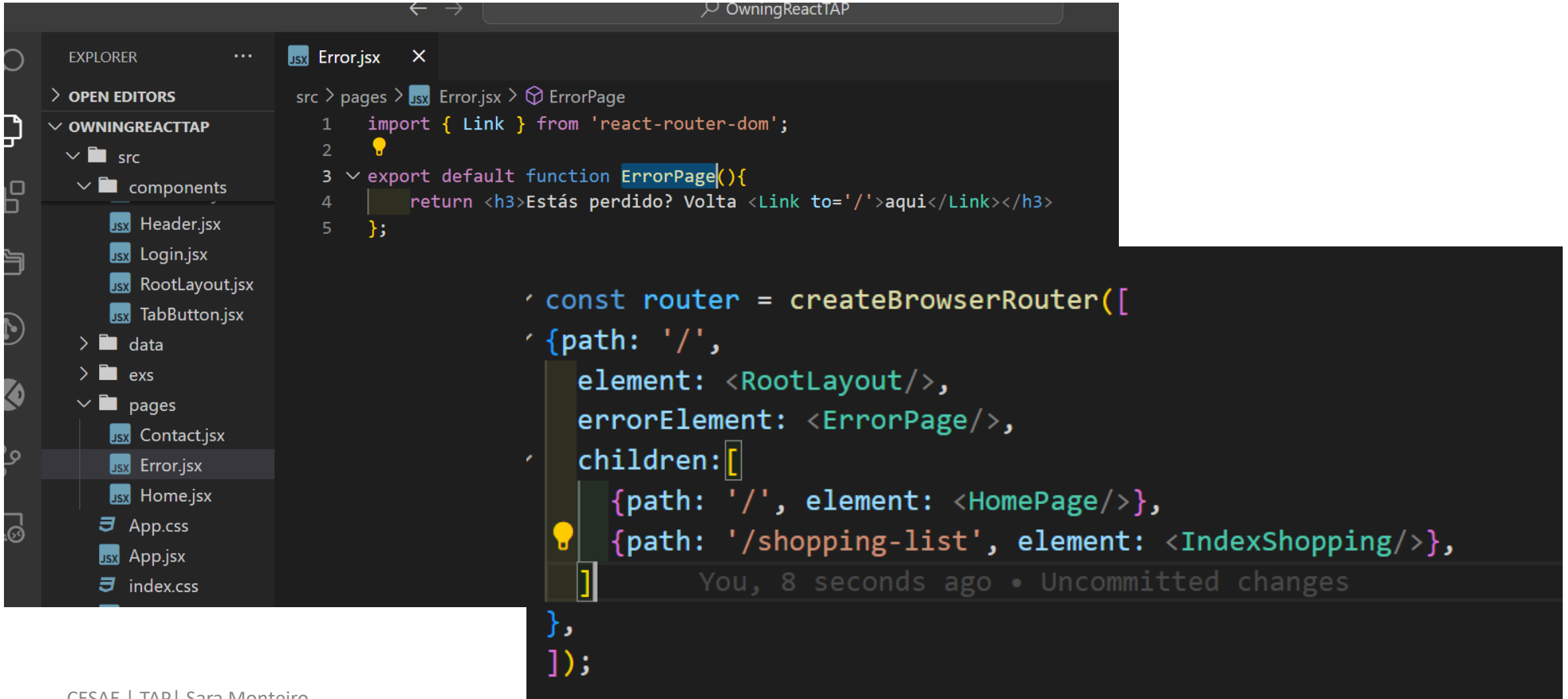
React Router: rota de erro

É bastante usual que, por erro, o utilizador coloque um url que não se encontra registado nas nossas rotas. O que acontece neste caso é aparecer uma página de erro.

É boa prática definirmos nós uma página que oriente o utilizador para voltar à HomePage, por exemplo.



React Router: rota de erro



The screenshot shows a code editor with the following structure:

- EXPLORER**
 - OPEN EDITORS
 - OWNINGREACTTAP
 - src
 - components
 - Header.jsx
 - Login.jsx
 - RootLayout.jsx
 - TabButton.jsx
 - data
 - exs
 - pages
 - Contact.jsx
 - Error.jsx (selected)
 - Home.jsx
 - App.css
 - App.jsx
 - index.css

src > pages > Error.jsx

```
1 import { Link } from 'react-router-dom';
2
3 export default function ErrorPage(){
4   return <h3>Estás perdido? Volta <Link to='/'>aqui</Link></h3>
5 };

const router = createBrowserRouter([
  {path: '/',
    element: <RootLayout/>,
    errorElement: <ErrorPage/>,
    children:[
      {path: '/', element: <HomePage/>},
      {path: '/shopping-list', element: <IndexShopping/>},
    ]
  },
]);
```

You, 8 seconds ago • Uncommitted changes

React Router: rotas com parâmetros

- No React as **rotas com parâmetros** permitem criar **URLs dinâmicas**, o que facilita a navegação entre diferentes páginas sem precisar definir cada rota manualmente.
- Usam-se para exibir detalhes de um item, perfis de utilizador, filtros de categoria e paginação dinâmica.

React Router: rotas com parâmetros

```
1  
2  ✓ const router = createBrowserRouter([  
3    {path: '/', element: <HomePage/>},  
4    { path: '/user/:name', element: <User/> },  
5  ])  
6
```

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

```
export default function HomePage(){  
  let { name } = useParams(); //  
  return <h1>O meu user é o: {name}</h1>  
}
```

Funcionalidade 1



Realize e entregue a tarefa proposta.

Recursos

- <https://developer.mozilla.org/en-US/docs/Web/JavaScript>
- <https://react.dev/>