



Software Developer

Integração de Sistemas de Informação

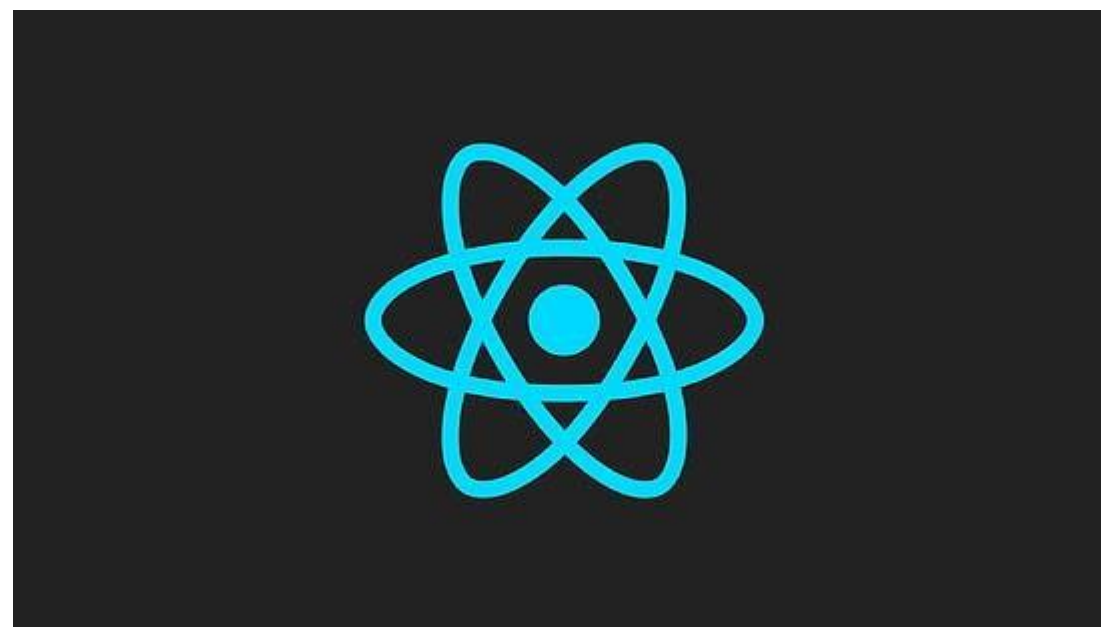
Sara Monteiro

sara.monteiro.prt@msft.cesae.pt

React: o que é e porquê usar?

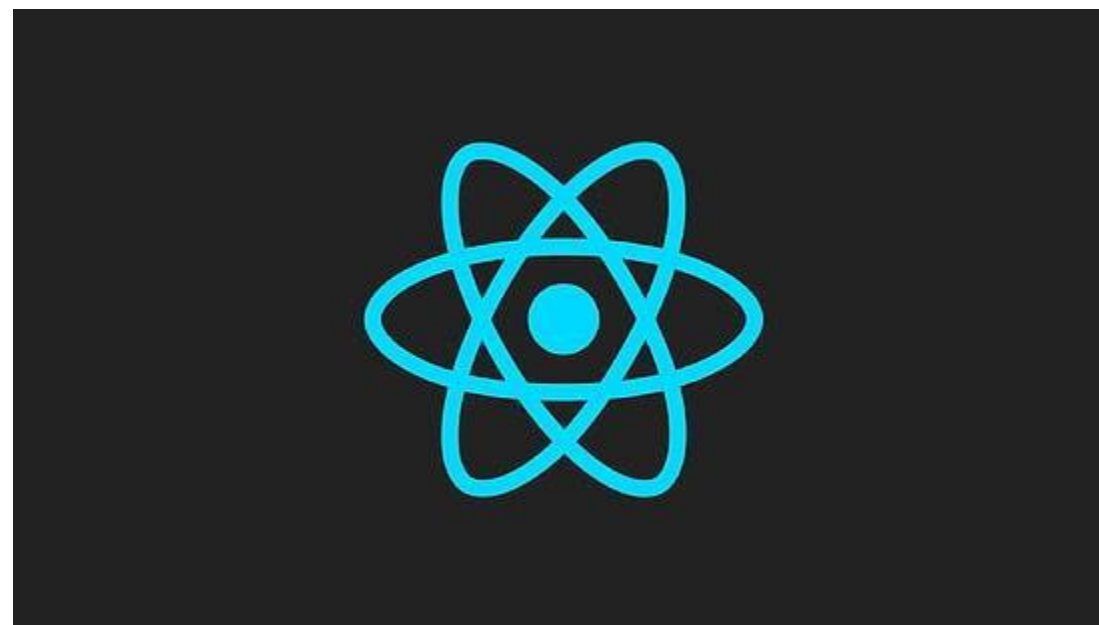
- Biblioteca JS para web que cria interfaces dinâmicas
- É a framework de FE com mais procura do mercado.
- Torna as transições de elementos numa pagina web mais suaves e rápidas, sem carregar a página constantemente. Ex: Netflix

<https://react.dev/>



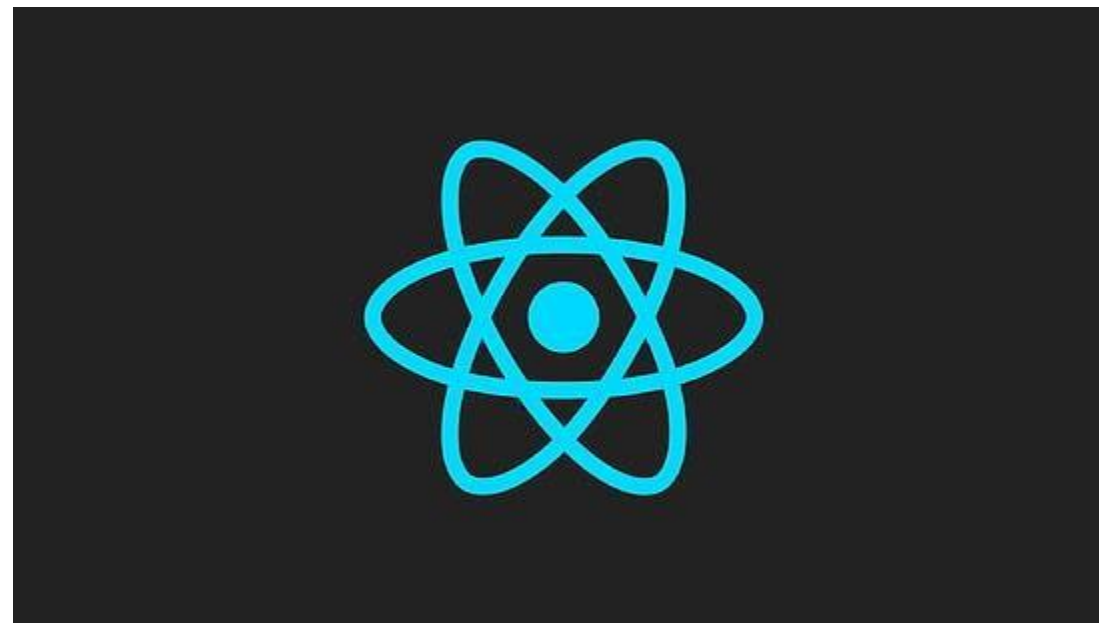
React: o que é e porquê usar?

- Assemelha-se a uma aplicação mobile e consegue facilmente ser adaptado para dispositivos móveis pelo react native
- O React é rápido e as suas aplicações conseguem processar grandes updates.



React: o que é e porquê usar?

- É modular e permite-nos fazer pequenos ficheiros de Código reutilizáveis, resolvendo os problemas de manutenção do JS, que é mais propenso a erros.
- É escalável e flexível: tanto funciona para projecto pequenos como para tratamento de muitos dados.



React: porquê usar?

React/JS:

- Com react escrevemos código declarativo: definimos os estados no UI e o react descobre as etapas para chegar lá.
- Em JS escrevemos código imperativo: definimos os passos, não o objectivo.

Exemplo

Criar um projecto em React

- Sandbox online: basta no browser colocar `react.new`
- Localmente:
 - ❖ Possibilidade de configurar com as nossas necessidades
 - ❖ Uso no Visual Studio Code ou noutro à nossa escolha

Node



- ambiente de execução de JavaScript V8 do Chrome.
- software de código aberto para construir aplicações com linguagem de servidor
- representado pelo seu gestor de pacotes npm

Criar um projecto em React usando o Vite

- Existem várias formas para começar um projecto React localmente, e todas necessitam a instalação do [node.js](https://nodejs.org/).
- Por questões de simplicidade e eficiência usaremos o [vite](https://vitejs.dev/).

```
PS C:\Users\Utilizador\OneDrive - CESAE\FrontEndDeveloper\SJM_24\TAP\TAPTests> npm create vite@latest ReactTAP
Need to install the following packages:
create-vite@5.5.2
Ok to proceed? (y) █
```


Criar um projecto em React usando o Vite

```
> npx
> create-vite ReactTAP

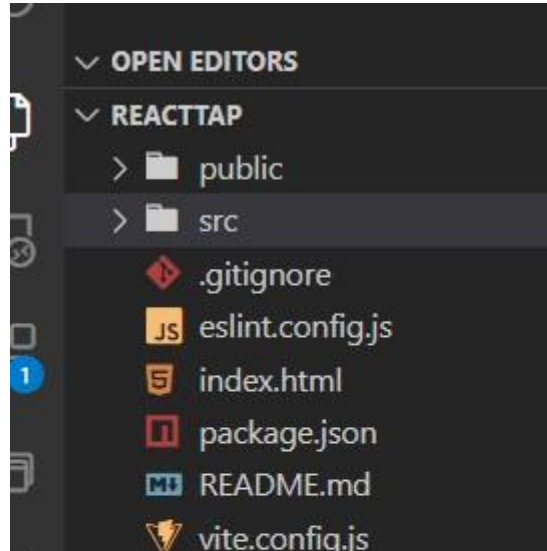
✓ Package name: ... reacttap
? Select a framework: » - Use arrow-keys. Return to submit.
  Vanilla
  Vue
> React
  Preact
  Lit
  Svelte
  Solid
  Qwik
  Others
```

```
✓ Package name: ... reacttap
✓ Select a framework: » React
? Select a variant: » - Use arrow-keys. Return to submit.
  TypeScript
  TypeScript + SWC
> JavaScript
  JavaScript + SWC
  Remix ↗
```

Criar um projecto em React usando o Vite

Após a instalação verificamos que temos um projecto que pode ser aberto com qualquer editor de código.

No terminal iremos agora correr npm install para termos todos os pacotes completos e o projecto está pronto a usar!



```
operable program or batch file.
PS C:\Users\Utilizador\OneDrive - CESAE\FrontEndDeveloper\SJM_24\TAP\TAPTests\ReactTAP> npm install

added 263 packages, and audited 264 packages in 22s

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

found 0 vulnerabilities
```

Criar um projecto em React usando o Vite

Se correremos no terminal `npm run dev` verificamos que nos é criado um servidor de desenvolvimento para o nosso projecto react. Ao clicar, temos acesso ao início da nossa aplicação!

Nota: o projecto só corre se o servidor estiver iniciado porque o código JSX precisa de ser compilado.

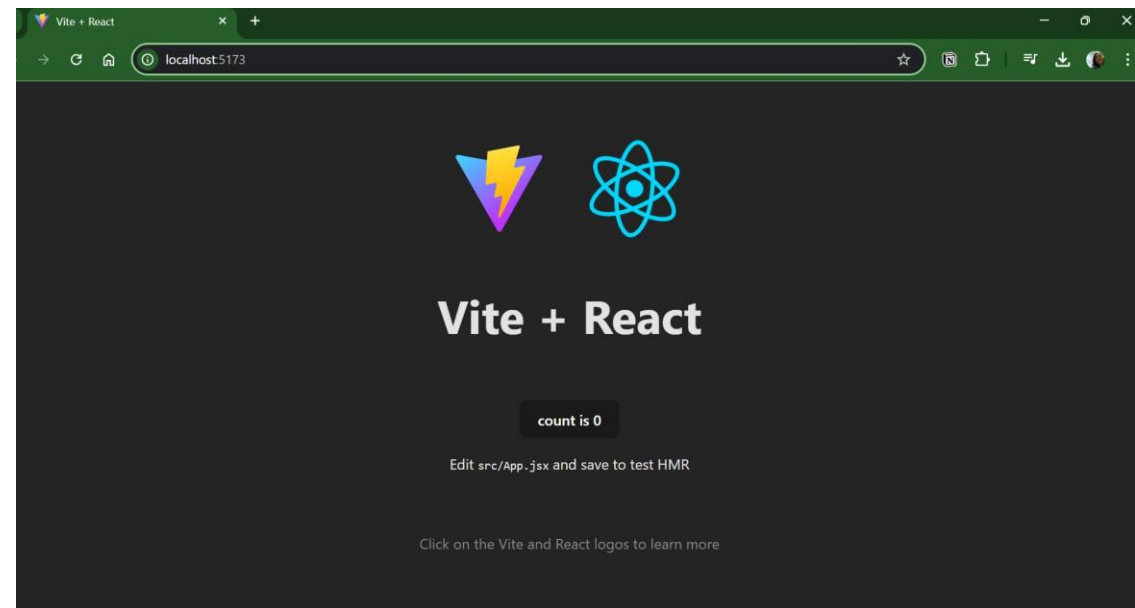
```
TERMINAL  DEBUG CONSOLE  OUTPUT  PORTS  DEVDB  PROBLEMS

found 0 vulnerabilities
PS C:\Users\Utilizador\OneDrive - CESAE\FrontEndDeveloper\SJM_24\TAP\TAPTests\ReactTAP> npm run dev

> reacttap@0.0.0 dev
> vite

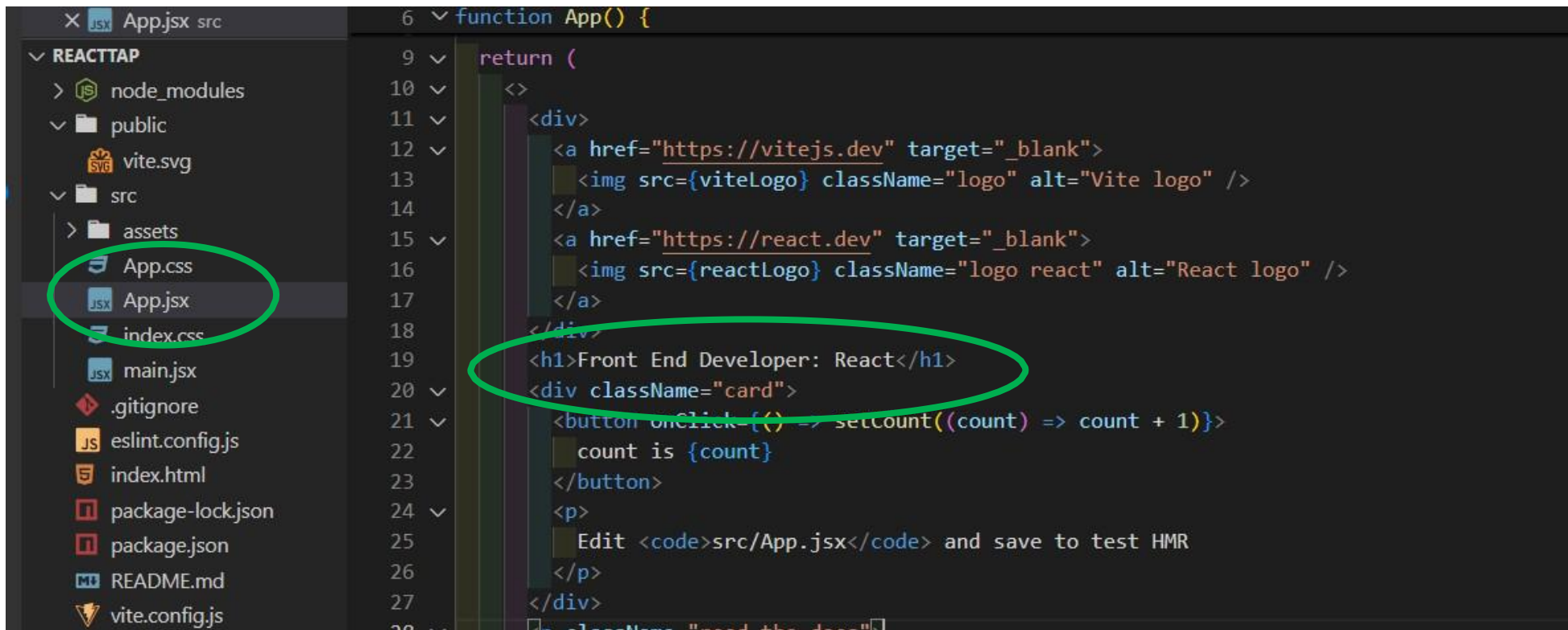
VITE v5.4.2 ready in 391 ms

→ Local:   http://localhost:5173/
→ Network: use --host to expose
→ press h + enter to show help
```



Customizar o nosso Projecto

Antes de entrar na nossa aprendizagem,
iremos personalizar e familiarizar-nos com o
projecto, editando o título inicial para Front
End Developer: React

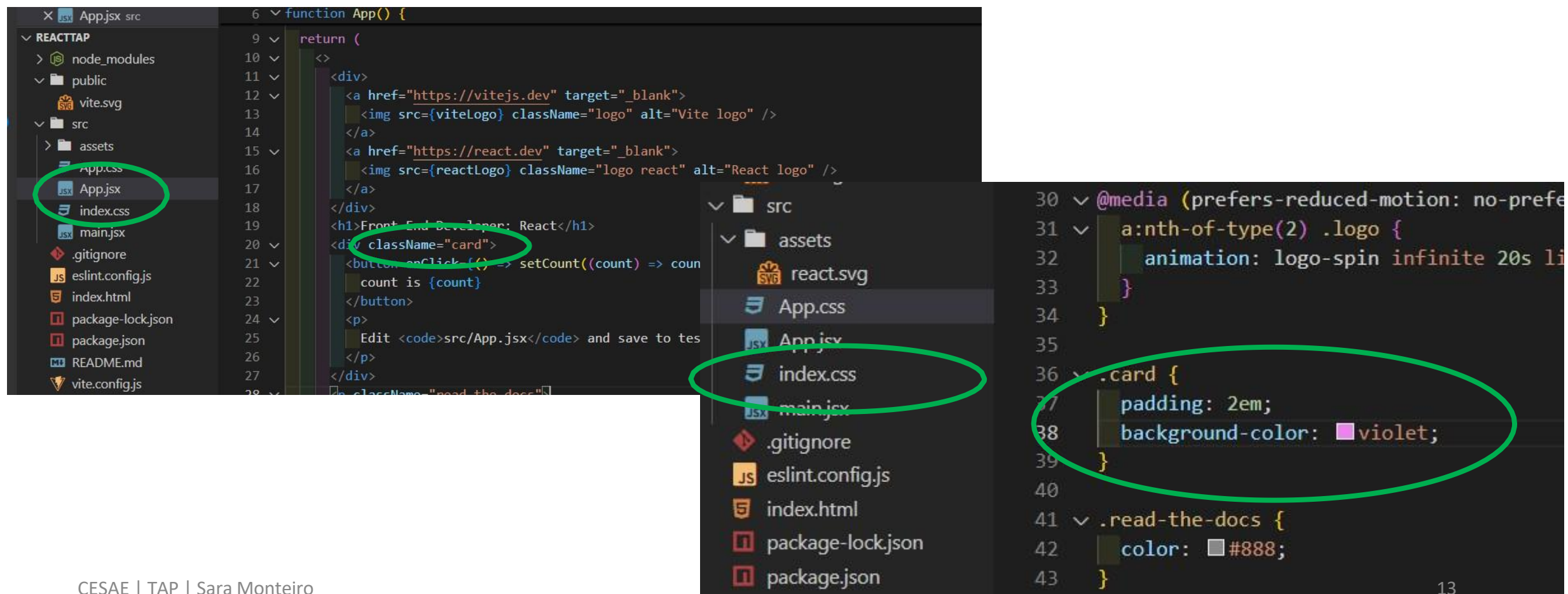


The screenshot shows a code editor with a dark theme. On the left, the file explorer displays the project structure. The 'src' folder is expanded, and 'App.jsx' is selected and circled in green. The main editor area shows the content of 'App.jsx'. The code defines a function 'App()' that returns a JSX element. The JSX element contains a div with two links, two images, and a h1 tag. The h1 tag, which contains the text 'Front End Developer: React', is circled in green. Below the h1 tag, there is a button that increments a count, and a paragraph of text.

```
6  function App() {
9    return (
10     <>
11     <div>
12       <a href="https://vitejs.dev" target="_blank">
13         <img src={viteLogo} className="logo" alt="Vite logo" />
14       </a>
15       <a href="https://react.dev" target="_blank">
16         <img src={reactLogo} className="logo react" alt="React logo" />
17       </a>
18     </div>
19     <h1>Front End Developer: React</h1>
20     <div className="card">
21       <button onClick={() => setcount((count) => count + 1)}>
22         count is {count}
23       </button>
24       <p>
25         Edit <code>src/App.jsx</code> and save to test HMR
26       </p>
27     </div>
28   )
29 }
```

Customizar o nosso Projecto

Em seguida mudaremos a cor de fundo do card para violeta, usando CSS.



```
function App() {  
  return (  
    <div>  
      <a href="https://vitejs.dev" target="_blank">  
        <img src={viteLogo} className="logo" alt="Vite logo" />  
      </a>  
      <a href="https://react.dev" target="_blank">  
        <img src={reactLogo} className="logo react" alt="React logo" />  
      </a>  
    </div>  
    <h1>Front End Developer: React</h1>  
    <div className="card">  
      <button onClick={() => setCount((count) => count + 1)}>  
        count is {count}  
      </button>  
      <p>  
        Edit <code>src/App.jsx</code> and save to test  
      </p>  
    </div>  
  )  
}
```

```
@media (prefers-reduced-motion: no-preference) {  
  a:nth-of-type(2) .logo {  
    animation: logo-spin infinite 20s linear;  
  }  
}  
  
.card {  
  padding: 2em;  
  background-color: violet;  
}  
  
.read-the-docs {  
  color: #888;  
}
```

Customizar o nosso Projecto



Front End Developer: React

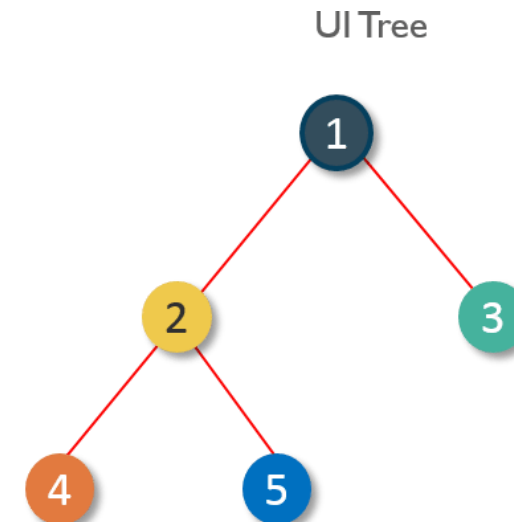
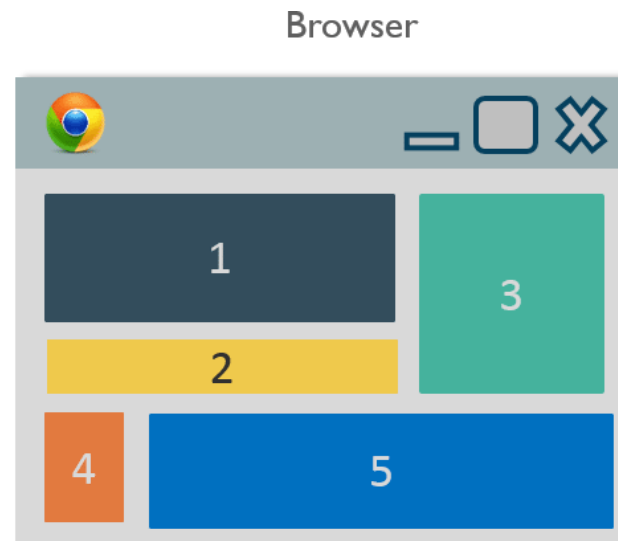
count is 3

Edit `src/App.jsx` and save to test HMR

Click on the Vite and React logos to learn more

React Essenciais: Componentes

- Blocos de Código reutilizáveis
- Permite ao programador pegar em interfaces complexas e parti-lo em pequenas partes que podem ser usadas várias vezes
- É também vantajoso porque centralizamos os componentes e caso tenhamos que fazer mudanças é apenas num sítio
- A estrutura fica separada: diferentes componentes gerem lógica diferente, simplificando o processo em apps complexas



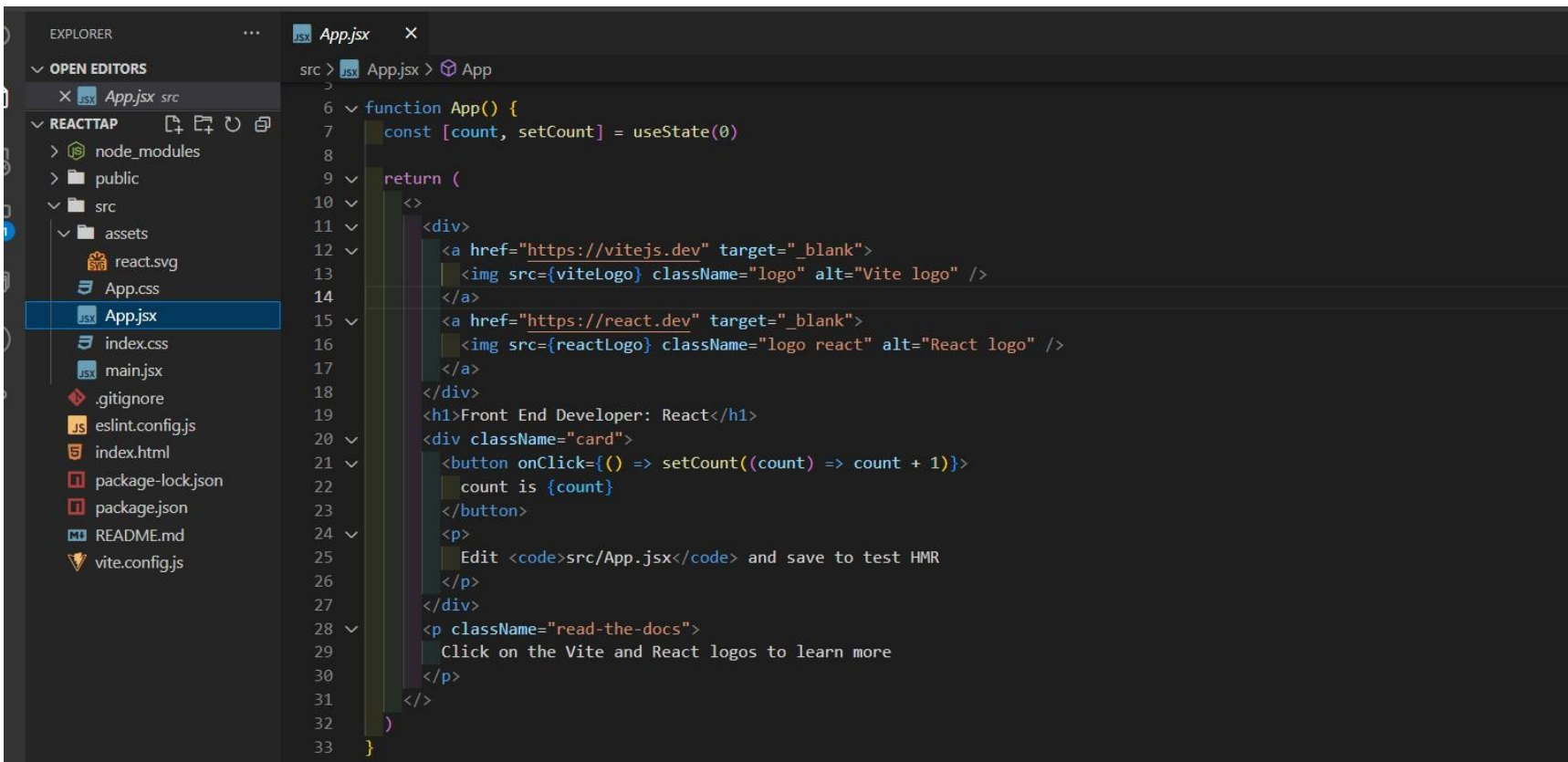
React Essenciais:

Ficheiros .jsx

- Se analisarmos o index.html do nosso projecto, iremos verificar que o script nos indica um ficheiro de js atípico, com a extensão .jsx.
- JSX significa JavaScript Syntax eXtension e é usado para descrever e criar elementos HTML em JS de uma forma declarativa.
- O código JSX não é lido por browser e precisa de ser compilado em JS pelo react.

```
index.html X
index.html > html > body
1  <!doctype html>
2  <html lang="en">
3  <head>
4    <meta charset="UTF-8" />
5    <link rel="icon" type="image/svg+xml" href="/vite.svg" />
6    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
7    <title>Vite + React</title>
8  </head>
9  <body>
10    <div id="root"></div>
11    <script type="module" src="/src/main.jsx"></script>
12  </body>
13 </html>
14
```


React Essenciais: Ficheiros .jsx



```
6 function App() {
7   const [count, setCount] = useState(0)
8
9   return (
10     <>
11       <div>
12         <a href="https://vitejs.dev" target="_blank">
13           <img src={viteLogo} className="logo" alt="Vite logo" />
14         </a>
15         <a href="https://react.dev" target="_blank">
16           <img src={reactLogo} className="logo react" alt="React logo" />
17         </a>
18       </div>
19       <h1>Front End Developer: React</h1>
20       <div className="card">
21         <button onClick={() => setCount((count) => count + 1)}>
22           count is {count}
23         </button>
24         <p>
25           Edit <code>src/App.jsx</code> and save to test HMR
26         </p>
27       </div>
28       <p className="read-the-docs">
29         Click on the Vite and React logos to learn more
30       </p>
31     </>
32   )
33 }
```

Estes ficheiros são então os potenciadores para a criação dos nossos componentes.

Para isso, devem obedecer a duas regras:

1. Nome da função deve começar por uma letra maiúscula
2. A função deve retornar um valor renderizável.

React Essenciais: Cria um componente

Iremos criar o nosso primeiro componente, um header para nossa aplicação! No ficheiro App.jsx criamos então a função, colocamos o HTML que pretendemos e retornamos o mesmo renderizado.

Depois podemos usar o nosso componente chamando-o pelo seu nome.

```
function Header(){  
  return(  
    <header>  
    <h4>Aprendendo React!</h4>  
  </header>  
  );  
}  
  
function App() {  
  const [count, setCount] = useState(0)  
  
  return (  
    <>  
      <div>  
        <Header/>  
        <a href="https://vitejs.dev" target="_blank">  
          <img src={viteLogo} className="logo" alt="Vite logo" />  
        </a>  
      </div>  
    </>  
  )  
}
```

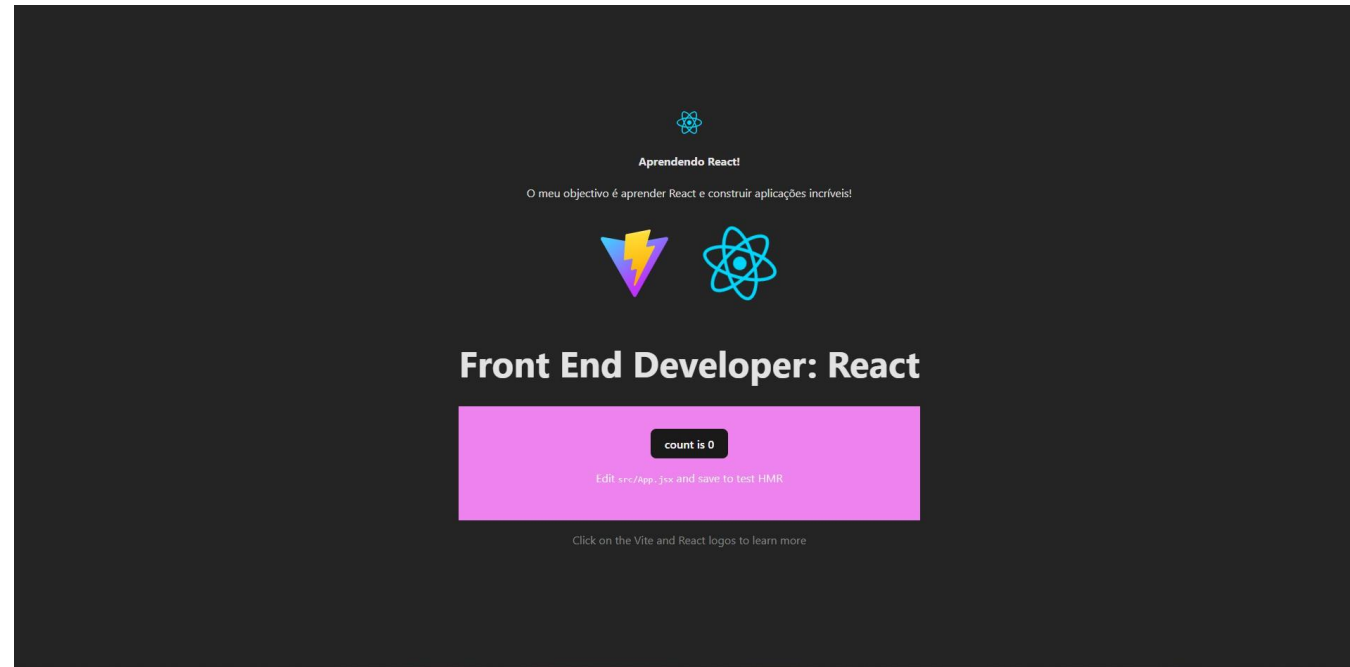
Criar um componente



Exercício:

Crie um componente chamado MainGoal que imprima um paragrafo a dizer “o meu objectivo é aprender React e construir aplicações incríveis!”.

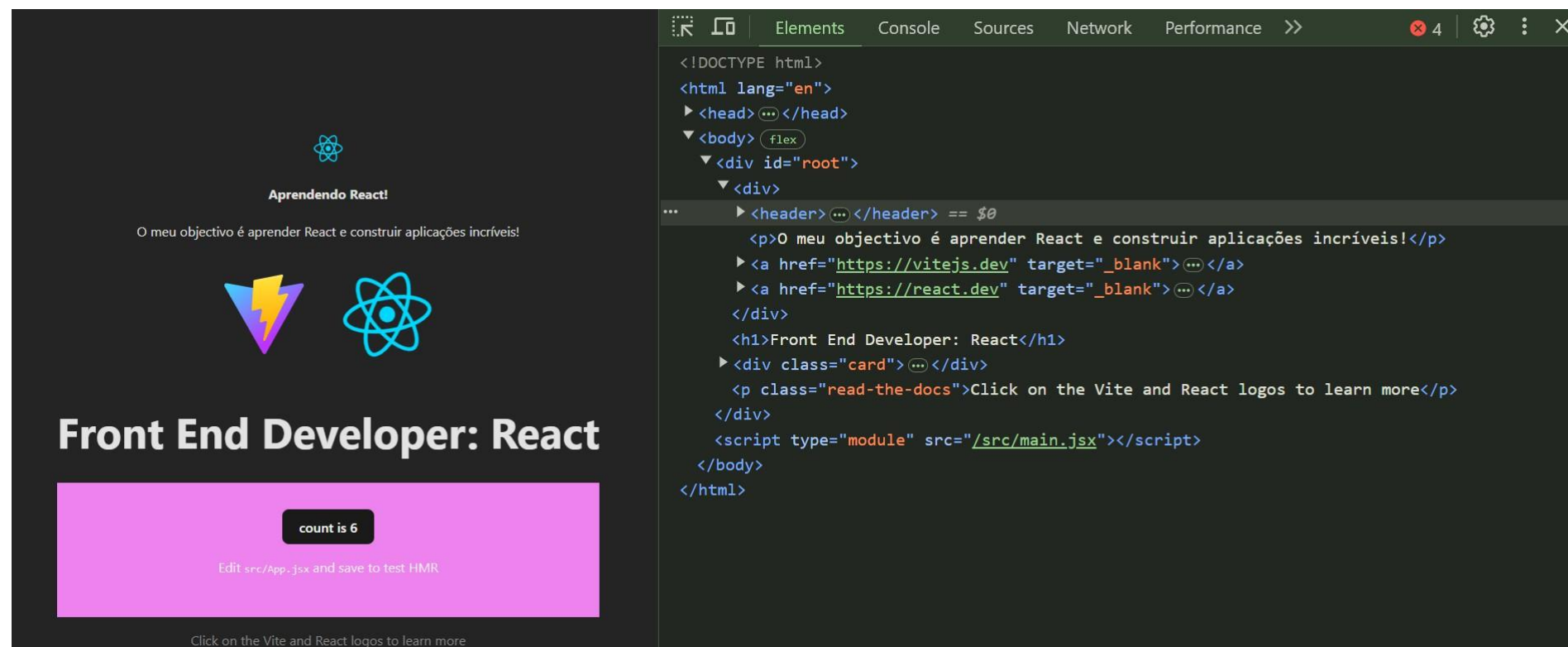
Deverá criar o componente de raíz e usá-lo dentro do código JSX da APP.



React Essenciais:

Analizando a construção do html

Se inspecionarmos a nossa página principal, verificamos que a nossa aplicação foi transformada numa página normal de html. Como é que o react fez isso?



The image shows a web browser window displaying a React application. The application has a dark theme and contains the following text and elements:

- A React logo at the top.
- The text "Aprendendo React!".
- A paragraph: "O meu objectivo é aprender React e construir aplicações incríveis!".
- Two logos: a Vite logo (a yellow lightning bolt) and a React logo.
- The heading "Front End Developer: React".
- A pink rectangular area containing a button that says "count is 6".
- Below the pink area, the text "Edit src/App.jsx and save to test HMR".
- At the bottom, a link: "Click on the Vite and React logos to learn more".

On the right side of the image, the browser's developer tools are open, showing the "Elements" tab. The rendered HTML structure is as follows:

```
<!DOCTYPE html>
<html lang="en">
  <head> ... </head>
  <body> flex
    <div id="root">
      <div>
        <header> ... </header> == $0
        <p>O meu objectivo é aprender React e construir aplicações incríveis!</p>
        <a href="https://vitejs.dev" target="_blank"> ... </a>
        <a href="https://react.dev" target="_blank"> ... </a>
      </div>
      <h1>Front End Developer: React</h1>
      <div class="card"> ... </div>
      <p class="read-the-docs">Click on the Vite and React logos to learn more</p>
    </div>
    <script type="module" src="/src/main.jsx"></script>
  </body>
</html>
```

React Essenciais:

Analizando a construção do html

```
index.html X
index.html > html > body
1 <!doctype html>
2 <html lang="en">
3 <head>
4   <meta charset="UTF-8" />
5   <link rel="icon" type="image/svg+xml" href="/vite.svg" />
6   <meta name="viewport" content="width=device-width, initial-scale=1.0" />
7   <title>Vite + React</title>
8 </head>
9 <body>
10   <div id="root"></div>
11   <script type="module" src="/src/main.jsx"></script>
12 </body>
13 </html>
14
```

```
> JSX main.jsx
1 import { StrictMode } from 'react'
2 import { createRoot } from 'react-dom/client'
3 import App from './App.jsx'
4 import './index.css'
5
6 createRoot(document.getElementById('root')).render(
7   <StrictMode>
8     <App />
9   </StrictMode>,
10 )
1
```

React Essenciais: conteúdo dinâmico

Por vezes iremos precisar de colocar conteúdo que seja dinâmico, ou seja, dependente de lógica de código ou do backend. Ex: username no login

Como criamos então os nossos componentes para aceitarem essa lógica? Usando {oNossoCodigo} nos nossos componentes.

```
App.jsx  X
rc > JSX App.jsx > [?] moduleDescriptions
2  import reactLogo from './assets/react.svg'
3  import viteLogo from '/vite.svg'
4  import './App.css'
5
6  ⚡
7  const moduleDescriptions = ['Aprendendo React', 'Aprendendo JS', 'Aprendendo SQL'];
8
9  function genRandomInt(max) {
10    return Math.floor(Math.random() * (max + 1));
11  }
12  const content = moduleDescriptions[genRandomInt(2)];
13
14  function Header(){
15    return(
16      <header>
17      <h4>{content}</h4>
18    </header>
19  );
20  }
```

Podemos directamente colocar código, variáveis, etc.

React Essenciais: conteúdo dinâmico

A estrutura de lógica pode também ser usada para carregar imagens, etc..

Verificamos por exemplo que o reactLogo já está importado e podemos substituir no header o caminho pela referência.

```
✓ import { useState } from 'react'
import reactLogo from './assets/react.svg'
import viteLogo from '/vite.svg'
import './App.css'
```

```
✓ function Header(){
✓   return(
✓     <header>
      <h4>{content}</h4>
    </header>
  );
}
```

```
✓ function Header(){
✓   return(
✓     <header><img src={reactLogo} alt=""/>
      <h4>{content}</h4>
    </header>
  );
}
```

Criar um componente

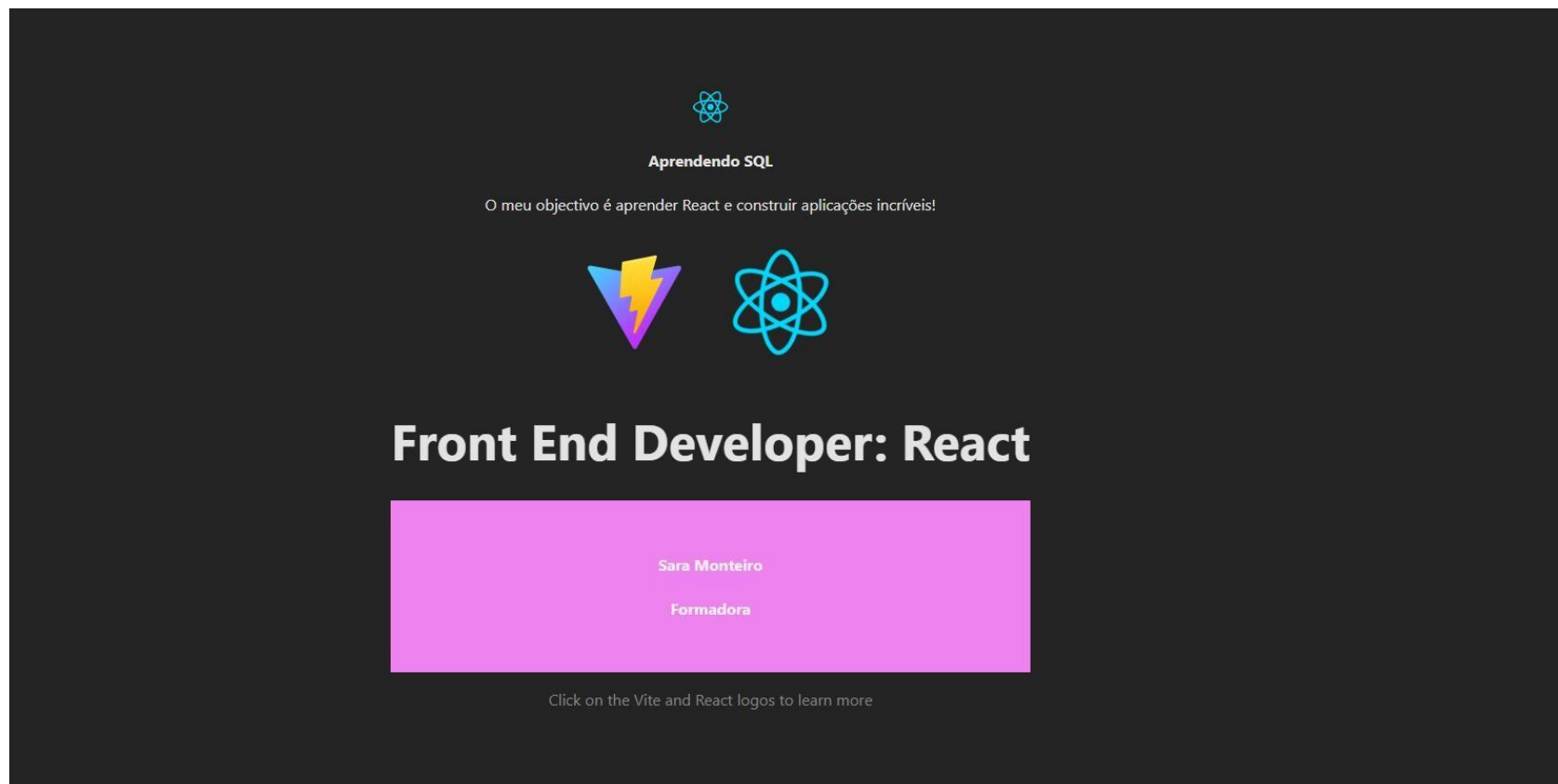


Centro para o Desenvolvimento
de Competências Digitais

No ficheiro que temos vindo a usar, o `app.jsx`, crie um componente `Card` e seus dados:

```
const userData =  
{ firstName: 'Sara',  
  lastName: 'Monteiro',  
  title: 'Formadora',};
```

Actualize o código de modo a que o card apareça com os dados do objecto.



React Essenciais: Props

Uma das vantagens de usar componentes, como falado, é que eles podem ser reutilizados com diferentes dados. Surge então o conceito de Props.

Para isso, iremos transformar o nosso card em componente para depois gerarmos vários cards com diferente conteúdo.

```
✓ function Card(){  
✓   return(  
✓     <div className="card">  
        <h4>{userData.firstName} {userData.lastName}</h4>  
        <h4>{userData.title}</h4>  
      </div>  
    );  
  }
```

```
return (  
  <div>  
    <div>  
      <Header/>  
      <MainGoal/>  
      <a href="https://vitejs.dev" target="_blank">  
        <img src={viteLogo} className="logo" alt="Vite logo" />  
      </a>  
      <a href="https://react.dev" target="_blank">  
        <img src={reactLogo} className="logo react" alt="React logo" />  
      </a>  
    </div>  
    <h1>Front End Developer: React</h1>  
    <Card/>  
    <p className="read-the-docs">  
      Click on the Vite and React logos to learn more  
    </p>  
  </div>  
)
```

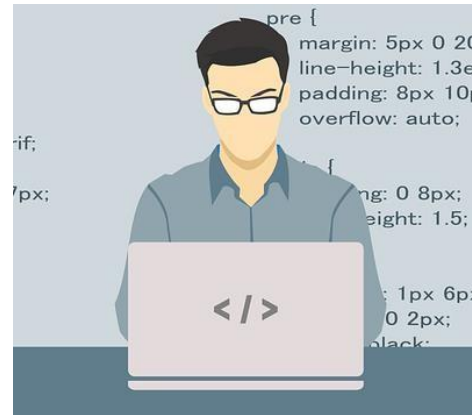
React Essenciais: Props

Usando os props, podemos simplesmente recriar o componente o número de vezes que quisermos, indicando os dados aquando chamamos o componente.

```
function Card(props){  
  return(  
    <div className="card">  
      <h6>{props.firstName}</h6>  
      <h6>{props.title}</h6>  
    </div>  
  )  
}
```

```
<h1>Front End Developer: React</h1>  
<Card  
  firstName="Sara"  
  lastName={userData.lastName}  
  title="Contabilista"  
>  
<Card  
  firstName="António"  
  title="Gestor"  
>  
<p className="read-the-docs">
```

Usar os props



Centro para o Desenvolvimento
de Competências Digitais

Crie um objecto com vários objectivos.

Pegando nesses objectivos, reutilize o componente mainGoal para replicar o componente com dados diferentes.



Aprendendo React

Aprender React e construir aplicações incríveis!

Fazer interfaces user friendly!

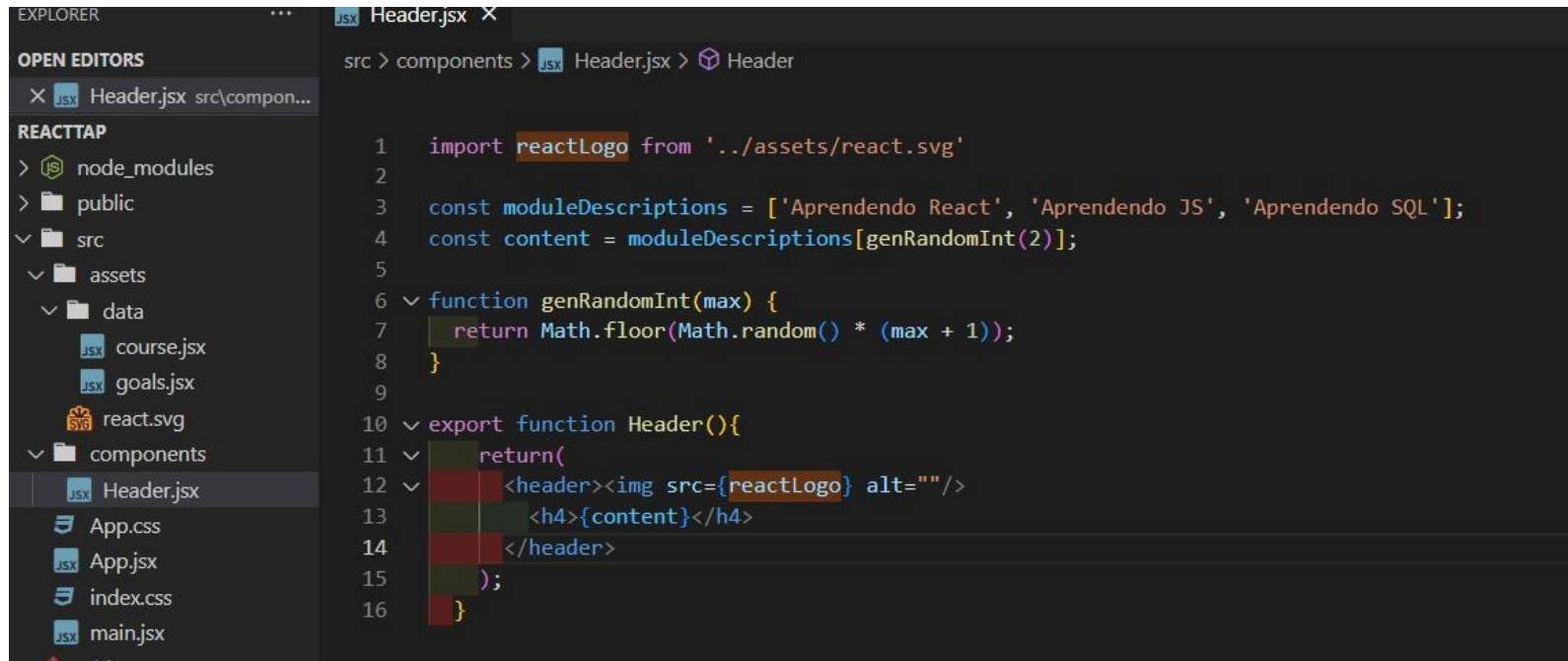
ter o meu código otimizado!

React Essenciais:

Reorganizar código

É boa prática em programação ter o conteúdo organizado e perceptível para quem pegue no nosso projecto.

Iremos, então, ter uma pasta de componentes com os componentes separados.



The screenshot shows the Visual Studio Code interface. On the left, the Explorer sidebar displays the project structure: `node_modules`, `public`, `src` (containing `assets` and `components`), and `App.css`, `App.jsx`, `index.css`, `main.jsx`. The `components` folder is expanded, showing `Header.jsx`. The main editor displays the code for `Header.jsx`, which includes an import for `reactLogo`, an array of module descriptions, a random selection function, and the `Header` component function that renders a header with the logo and a random description.

```
1 import reactLogo from '../assets/react.svg'
2
3 const moduleDescriptions = ['Aprendendo React', 'Aprendendo JS', 'Aprendendo SQL'];
4 const content = moduleDescriptions[genRandomInt(2)];
5
6 function genRandomInt(max) {
7   return Math.floor(Math.random() * (max + 1));
8 }
9
10 export function Header(){
11   return(
12     <header><img src={reactLogo} alt=""/>
13     <h4>{content}</h4>
14   </header>
15 );
16 }
```

React Essenciais:

Reorganizar código

Para que o código consiga ser usado em outros ficheiros teremos que o exportar e depois importar

```
5  
6  
7 }  
8  
9 export default Header  
10  
11  
12 import ReactLogo from './assets/react.svg'  
13 import viteLogo from './vite.svg'  
14 import './App.css'  
15 import Header from './components/Header'
```

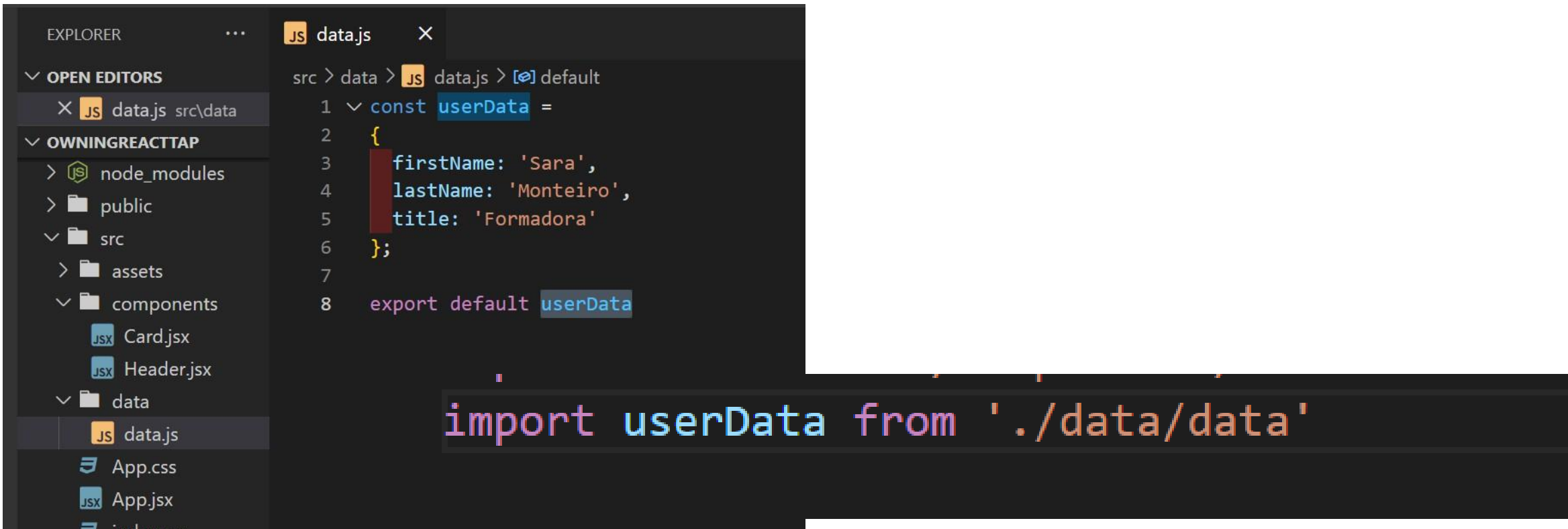
Reorganizar o Código



Usando a lógica de componentes, separe todos os componentes usados no App.jsx em ficheiros reutilizáveis.

Reorganizar Dados

Da mesma forma que organizámos os componentes, é também boa prática juntar os dados em pastas específicas. P/ex os objectos.



```
EXPLORER  ...  Js data.js  X

  OPEN EDITORS
  X Js data.js src\data

  OWNINGREACTTAP
  > node_modules
  > public
  > src
  > assets
  > components
  > Card.jsx
  > Header.jsx
  > data
  Js data.js
  App.css
  App.jsx
  index.css

src > data > Js data.js > [default]
1  const userData =
2    {
3      firstName: 'Sara',
4      lastName: 'Monteiro',
5      title: 'Formadora'
6    };
7
8  export default userData

import userData from './data/data'
```

Sintaxe Alternativa para Props 1



Centro para o Desenvolvimento
de Competências Digitais

```
<Card  
  firstName = 'Raquel'  
  title ='contabilista'  
>
```

```
<Card  
  firstName = {userData.firstName}  
  title ={userData.title}  
>
```

```
<Card  
  firstName = 'Raquel'  
  title ='contabilista'  
>
```

```
<Card {...userData}  
>
```


Sintaxe Alternativa para Props 2

```
✓ function Card({firstName, title}){  
✓   return(  
✓     <div className="card">  
      <h6>{firstName}</h6>  
      <h6>{title}</h6>  
    </div>  
  )  
}  
  
export default Card
```

Sintaxe Alternativa para Props 3

```
/ ** @jsx disable React/prop-types */  
function Card({firstName, title = 'programador'}){  
  return(  
    <div className="card">  
      <h6>{firstName}</h6>  
      <h6>{title}</h6>  
    </div>  
  )  
}  
  
export default Card
```

Reutilizar Componentes



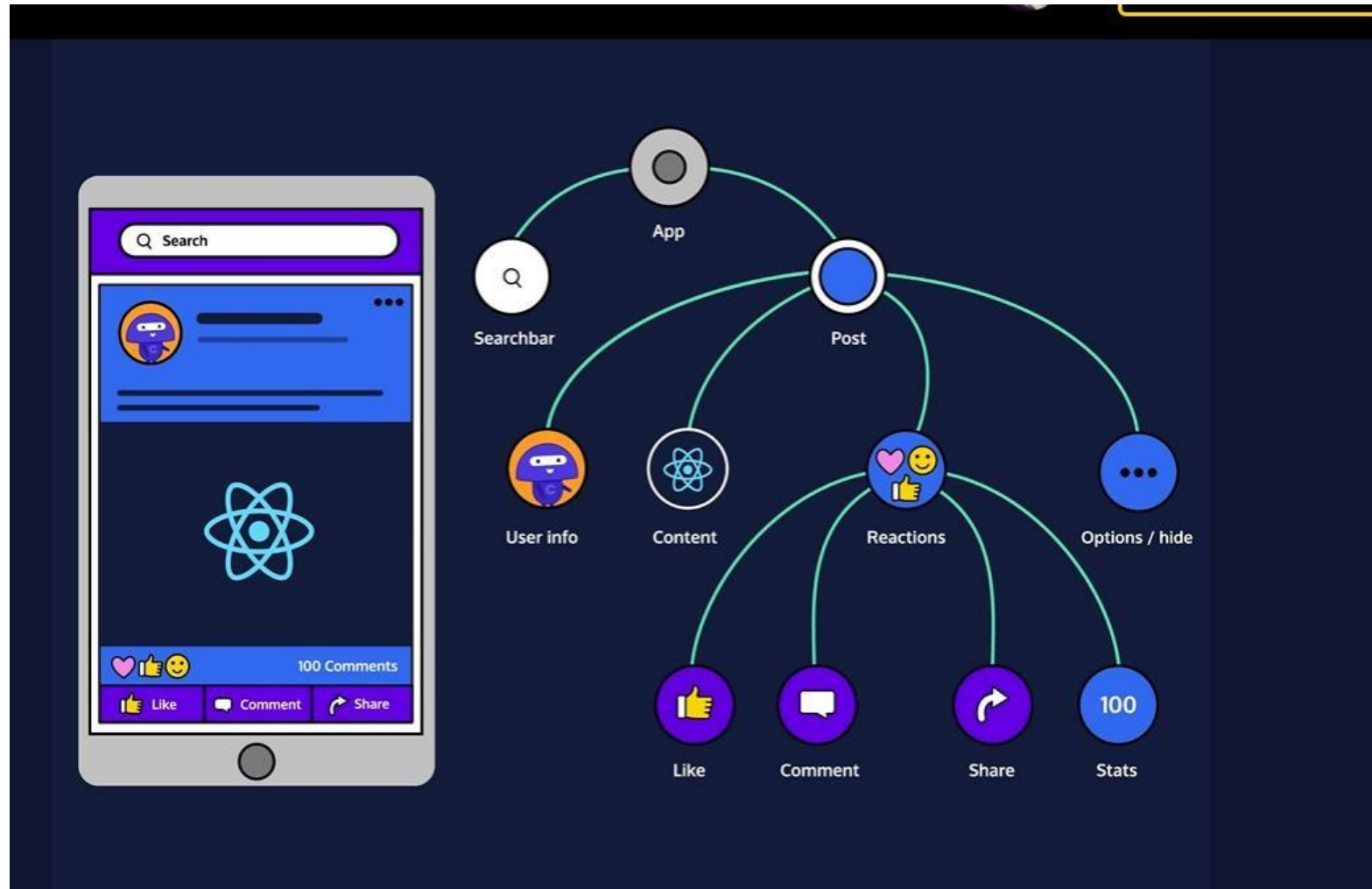
Tendo em conta a seguinte função, torne-a reutilizável (chamar na função APP() através do uso de um componente)

```
export function CourseGoal() { return (<h2>TITLE</h2>    <p>DESCRIPTION</p>    );}
```

TITLE: React

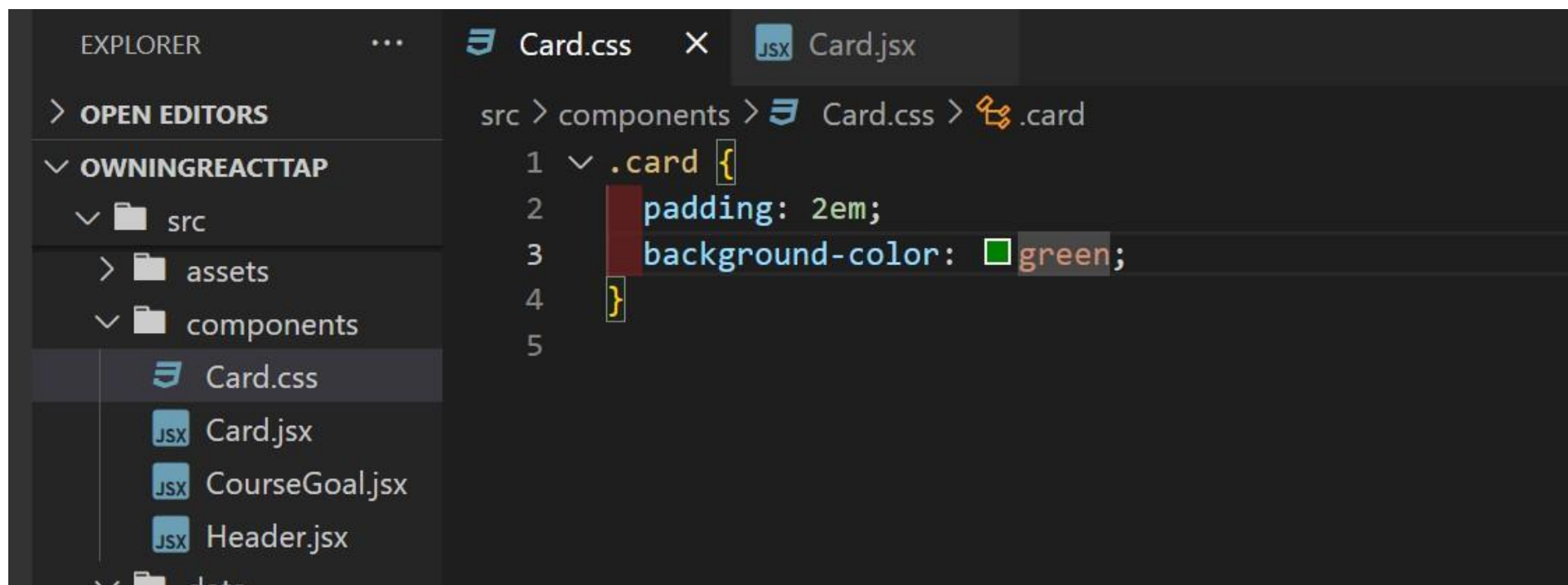
DESCRIPTION: aprender aprofundadamente o mundo do FE

Reutilizar Componentes



Organizar Código: boas práticas

- Em react, é boa prática darmos os nomes aos ficheiros de css conforme o componente que estamos a usar. Desta forma os estilos ficam junto com a estrutura do nosso HTML.



Composição dos Componentes

Existem situações em que podemos querer aprofundar a customização, como por exemplo em caso de botões, ao colocar o texto do que faz o botão.

Mas se fizermos por exemplo isto ele ignora o texto do botão. Como resolver?

```
export function TabButton(){  
  return <button></button>  
}  
  
<TabButton>Componentes</TabButton>  
</>
```

Composição dos Componentes

Iremos então aplicar os nossos props, o chamando *Component Composition*, para podermos usar uma tag react com um *children*.

Nota: Esta prop é a única que tem que ser chamada pelo nome fixo de *children*.

```
export function TabButton({children}){  
  return <button>{children}</button>  
}
```

```
<TabButton>Componentes</TabButton>  
/>
```


Eventos em React

Em *Vanilla JS* iríamos usar um selector para colocar o botão “à escuta”. Em React a programação é declarativa e não imperativa. Voltaremos então a declarar a função no botão.

Nota: o `onClick` pode tb ser usado em outros elementos que não botões.

```
4  export function TabButton({ children }) {  
5    return <button on<u>click</u>={children}></button>  
6  }  
7  }  
8  }  
9  }  
10 }
```

onAbort? (property) React.DOMAttributes<HTMLButtonElement>
onAbortCapture?
onAnimationEnd?
onAnimationEndCapture?
onAnimationIteration?
onAnimationIterationCapture?
onAnimationStart?

Eventos em React

```
2
3  ✓ export function TabButton({children}){
4  ✓    function handleClick(){
5    console.log('Hello World!')
6    }
7
8    return <button onClick={handleClick}>
      {children}</button>
9  }
```

Tornar as páginas dinâmicas

Tendo em conta que já aprendemos o básico dos componentes, iremos agora tornar as nossas páginas dinâmicas. Podemos, por exemplo, apresentar um texto diferente numa mesma div a cada vez que clicamos num botão.

```
<h3>Eventos Dinâmicos</h3>
<menu>
  <TabButton>Matéria JS</TabButton>
  <TabButton>Matéria React</TabButton>
  <TabButton>Matéria SQL</TabButton>
  <div>
    Conteúdo Dinâmico
  </div>
</menu>
```

Tornar as páginas dinâmicas

No componente:

```
export function TabButton({children, onSelect}){  
  return <button onClick={onSelect}>{children}</button>  
}
```

Onde o iremos usar
pt1:

```
import { TabButton } from './components/TabButton'  
  
function App() {  
  function handleSelect(selectedBtn){  
    console.log(selectedBtn)  
  }  
  
  return (  
    <div>  
      <TabButton onSelect={handleSelect}>1</TabButton>  
      <TabButton onSelect={handleSelect}>2</TabButton>  
    </div>  
  )  
}
```

Tornar as páginas dinâmicas

Onde o iremos usar pt2:

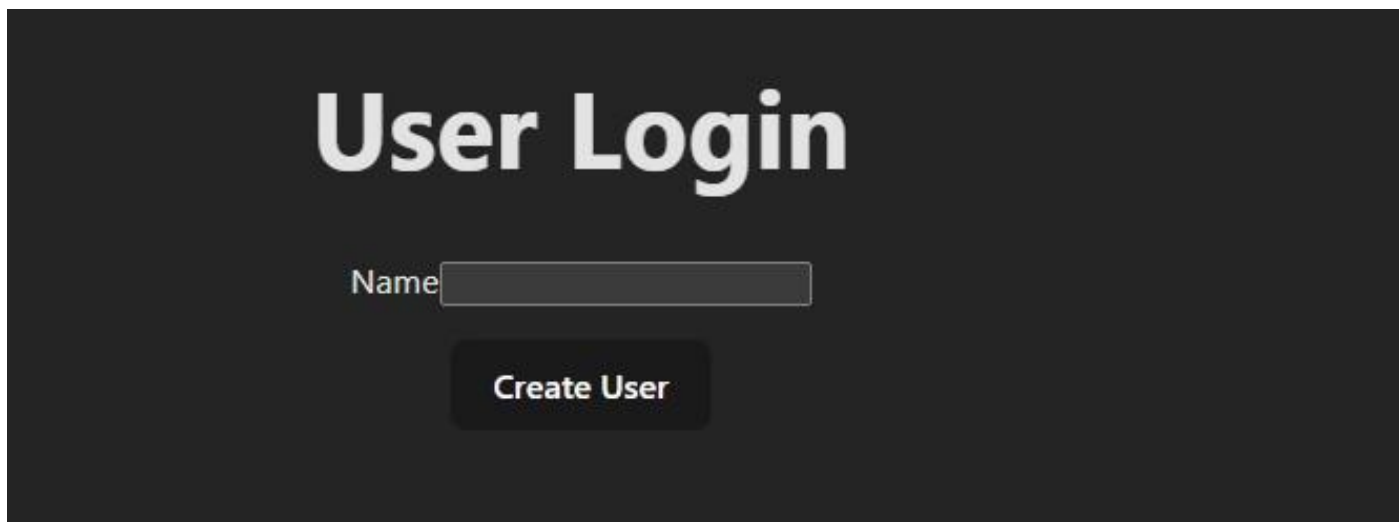
```
<h3>Eventos Dinâmicos</h3>
<menu>
  <TabButton onSelect={() => handleSelect('js')}>Matéria JS</
  TabButton>
  <TabButton onSelect={() => handleSelect('react')}>Matéria
  React</TabButton>
  <TabButton onSelect={() => handleSelect('sql')}>Matéria SQL</
  TabButton>
  <div>
    Conteúdo Dinâmico
  </div>
</menu>
```

Páginas Dinâmicas



Tendo em conta a seguinte função, torne-a reutilizável, enviando como parâmetro o nome do user:

Nota: o input é apenas para foco visual, é enviar o nome como parametro e fazer `console.log(user)` ao clicar em Create User. Ficheiro `states.jsx` no git.

A dark-themed user login form. At the top, the text 'User Login' is displayed in a large, bold, white sans-serif font. Below this, the word 'Name' is followed by a light gray rectangular input field. Underneath the input field is a dark gray rectangular button with the text 'Create User' in white.

Recursos

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