



por Iuri Silva



MÓDULO 6

React

Pegue o café e prepare-se para mais um módulo. Lembre-se de que, ao final de cada módulo, você pode avaliar seus conhecimentos por meio de um questionário no Notion.

✨ 10 TÓPICOS NESTE MÓDULO

Introdução

Essa sem dúvida é a tecnologia mais requisitada do mercado frontend. Quando queremos falar em flexibilidade no desenvolvimento com aplicações rápidas e de alta escalabilidade, o React vem logo em mente. O React irá te mostrar um novo mundo do JavaScript e o poder que ele tem nas criações de sites. Você está pronto para entrar para o mundo do React? Então vamos nessa!

O que é

React é uma biblioteca JavaScript para a criação de interfaces de usuário que utilizam o padrão SPA (Single Page Application).

"Está bem luri, mas o que é uma SPA?"

SPA é um padrão em que seu carregamento dos recursos (como CSS, JavaScript e HTML das páginas) ocorre apenas uma única vez: na primeira vez em que o usuário acessa a aplicação. Isso faz com que o carregamento das novas páginas seja bem mais rápido.

React foi criado pelo Meta (antiga empresa Facebook) em meados de 2011. Com o intuito de resolver um problema que os engenheiros do Facebook identificaram, que é a renderização da aplicação. Com React, na medida em que os dados mudam apenas os pedaços (que contém os dados) de uma determinada parte é renderizada. Diferente quando trabalhamos em uma aplicação feita com HTML, CSS e JavaScript onde a renderização é feita na página toda.

"E como isso é possível?"

Com o React, as interfaces do usuário são compostas por pedaços de código isolados, chamados de "componentes". Isso torna o código mais fácil de dar manutenção e reutiliza o mesmo código para outras funcionalidades.

Create React App

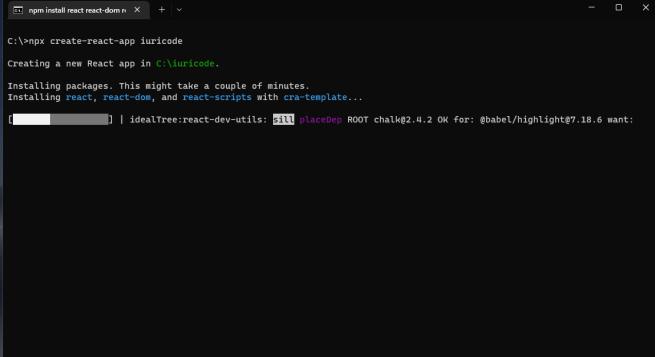
É importante ressaltar que você está explorando apenas uma amostra do material. Na versão completa, no módulo Bônus, aprofundamos a discussão sobre a substituição do CRA pelo Vite, fornecendo uma explicação detalhada sobre por que o Vite é melhor do que o CRA.

Create React App (CRA) é um conjunto de ferramentas e funcionalidades pré-configuradas que possibilitam o início fácil de um projeto React.

Além de configurar seu ambiente de desenvolvimento para utilizar as funcionalidades mais recentes do JavaScript, ele fornece uma experiência de desenvolvimento agradável, e otimiza a sua aplicação para produção. Será necessário ter o [Node](#) na sua máquina para criar um projeto em React.

Após instalar o Node já podemos criar um projeto, para isso damos o seguinte comando em nosso terminal:

```
npx create-react-app iuricode
```



```
npm install react react-dom
C:\>npx create-react-app iuricode
Creating a new React app in C:\iuricode.
Installing packages. This might take a couple of minutes.
Installing react, react-dom, and react-scripts with cra-template...
[██████████] | idealTree:react-dev-utils: sill placeDep ROOT chalk@2.4.2 OK for: @babel/highlight@7.18.6 want:
```

Explicando o comando:

- npx: npx é um executor responsável por executar as bibliotecas que podem ser baixadas do site npm.
- create-react-app: responsável por criar as funcionalidades.
- iuricode: nome do projeto.

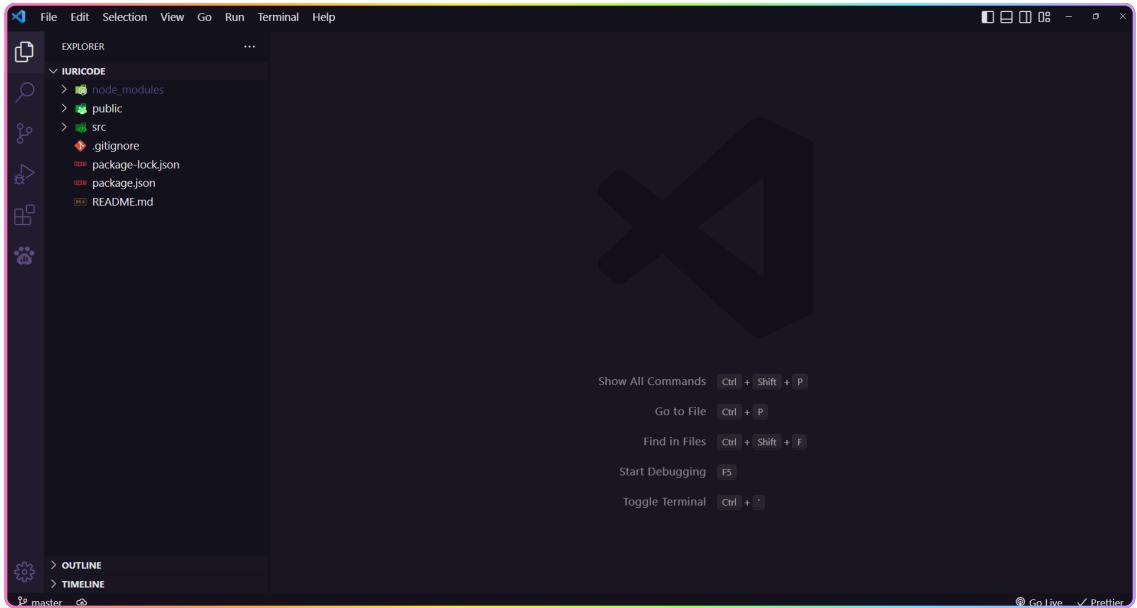
Após o npx instalar os pacotes, iremos dar o seguinte comando para entrar na aplicação criada.

```
cd iuricode
```

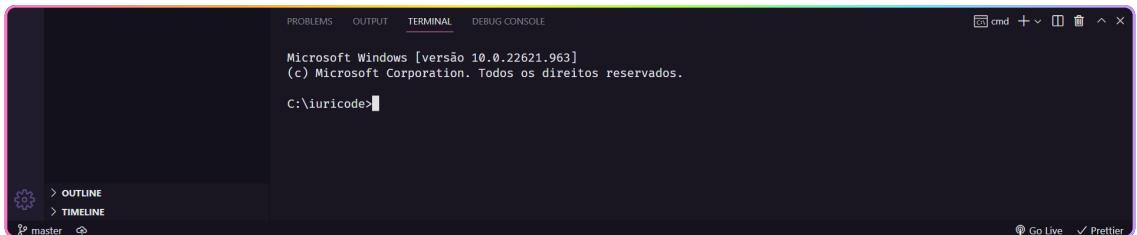
- cd: permite a mudança do diretório atual, nesse caso acessamos a pasta iuricode

Depois de estar no diretório do projeto criado podemos dar o seguinte comando para abrir o projeto dentro do nosso Visual Studio Code:

```
code .
```



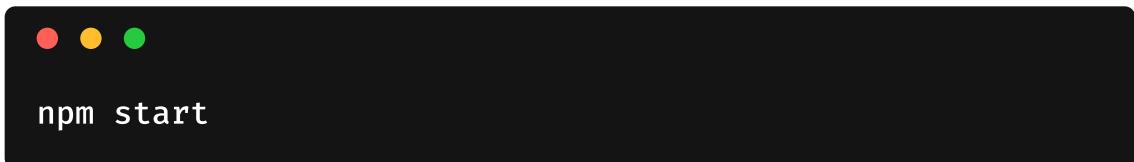
Após abrir o Visual Studio Code pressione as teclas Ctrl + ' (aspas simples), dessa forma irá abrir um terminal integrado no Visual Studio Code.



Em nosso terminal integrado iremos dar o seguinte comando para rodar nossa aplicação:



```
You can now view iuricode in the browser.  
Local: http://localhost:3000  
On Your Network: http://192.168.1.113:3000  
Note that the development build is not optimized.  
To create a production build, use npm run build.  
webpack compiled successfully
```



```
npm start
```

- npm start: é usado para executar a aplicação.

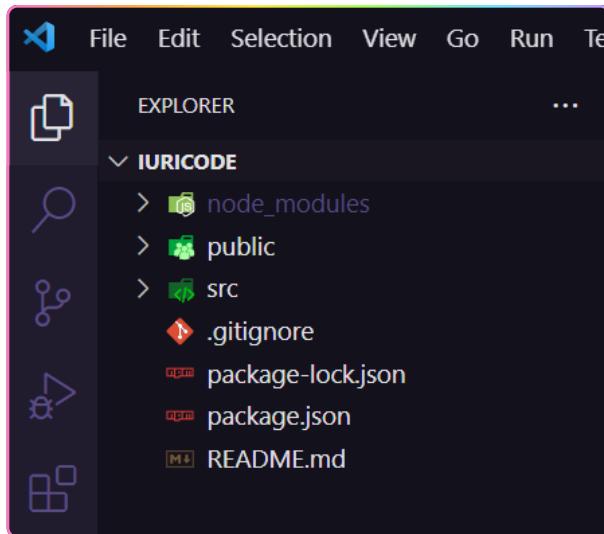
Após realizar o comando irá abrir (automaticamente) uma aplicação com o padrão do CRA na porta 3000 (três mil). Caso não seja iniciado automaticamente a aplicação no navegador você pode acessá-la em:

<http://localhost:3000/>

Uma coisa super importante é que às vezes será preciso parar o comando npm start e rodar novamente. Em muitos dos casos será necessário fazer essa tarefa quando mexemos nas configurações do projetos, mudamos os formatos dos arquivos, instalamos um novo pacote e entre outros casos.

Estrutura da pasta

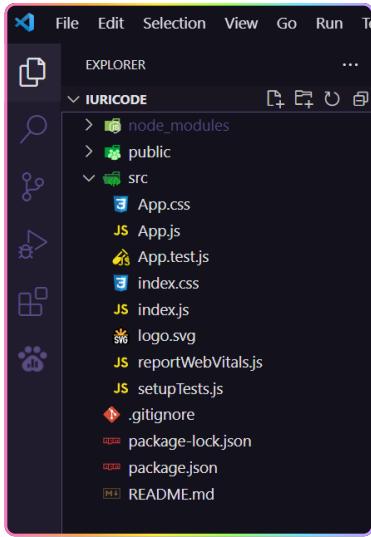
Está bem, mas o que exatamente o CRA criou para nós, luri? Como dito anteriormente, o CRA criou um conjunto de ferramentas e funcionalidades pré-configuradas para a gente iniciar nossa aplicação React. Na imagem a seguir iremos ver quais foram os conjuntos criados:



- **node_modules**: É o diretório criado pelo npm para rastrear cada pacote que você instala. Dependemos dele para que nossos pacotes sejam executados.
- **public**: que contém os recursos públicos da aplicação, como imagens. Além disso, temos um arquivo HTML que contém o id "root". É neste id que nossa aplicação React será renderizada e vai ser exibida.

- `src`: Contém os arquivos do componente React que é o responsável por renderizar a aplicação toda.
- `.gitignore`: Aqui ficaram os arquivos ignorados na hora de subir no Git.
- `package-lock.json`: Ele descreve as características das dependências usadas na aplicação, como versões, sub-pendências, links de verificação de integridade, dentre outras coisas.
- `package.json`: É um repositório central de configurações para ferramentas. Também é onde o npm armazena os nomes e versões de todos os pacotes instalados.
- `README.md`: É um arquivo que contém informações necessárias para entender o objetivo do projeto.

Dentro da pasta `src` temos alguns arquivos que não iremos utilizar na aplicação, mas antes deletá-los irei explicar sobre os mesmos pois é sempre bom saber para que servem.



- App.js: Esse arquivo contém todo o conteúdo que aparece em tela quando damos o comando npm start.
- index.js: Esse arquivo é feito a conexão do código React com o arquivo index.html que contém o id "root".
- App.test.js: É um arquivo de teste, ele é executado quando você roda o comando npm test.
- App.css & index.css: São arquivos de estilos CSS.
- logo.svg: Esse arquivo SVG é a logo do React.

- `reportWebVitals.js`: Esse arquivo permite medir o desempenho e a capacidade de resposta da sua aplicação.
- `setupTests.js`: Tudo o que ele possui são alguns métodos `expect` personalizados.

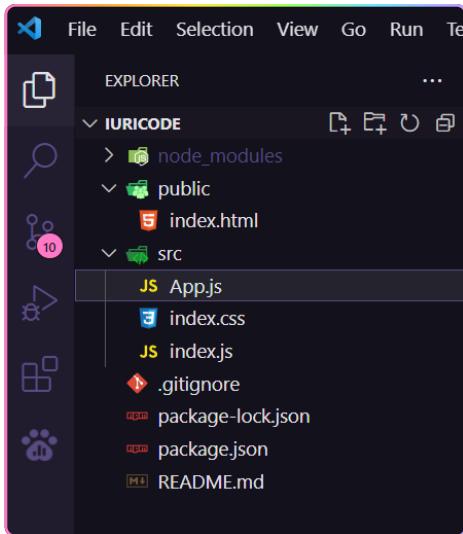
Geralmente deletamos os seguintes arquivos dentro da pasta `src`:

- `App.css`
- `logo.svg`
- `reportWebVitals.js`
- `setupTests.js`
- `App.test.js`

Além desses arquivos que não serão utilizado por nós, iremos limpar também os arquivos dentro da pasta `public` e as importações das imagens no `index.html`:

- `favicon.ico`
- `logo192.png`
- `logo512.png`
- `manifest.json`
- `robots.txt`

O resultado final da nossa aplicação será essa:



JSX

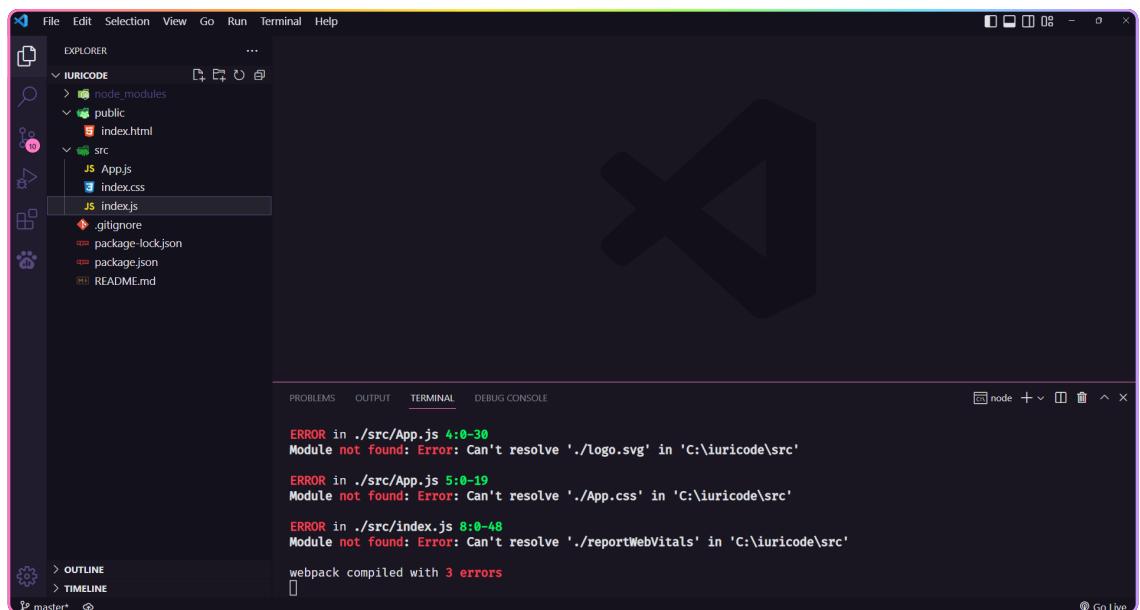
No topo do arquivo index.js, é importando o React, ReactDOM, index.css, App e serviceWorker. Ao importar o React, você está, na verdade, extraíndo código para converter o JSX em JavaScript. JSX são os elementos similares ao HTML.

"Mas o que é esse JSX, Iuri?"

Essa é uma extensão de sintaxe especial e válida para o React, seu nome vem do JavaScript XML. Normalmente, mantemos o código em HTML, CSS e JavaScript em arquivos separados. Porém no React, isso funciona de um modo um pouco diferente.

Nos projetos em React, não criamos arquivos em HTML separados, pois o JSX nos permite escrever uma combinação de HTML e JavaScript em um mesmo arquivo. Você pode, no entanto, separar seu CSS em outro arquivo.

Antes de trabalhar com o JSX, iremos tirar os erros de importações que estão aparecendo em seu projeto. Seu console deve estar com os seguintes alertas de erros:



The screenshot shows the Visual Studio Code interface. The Explorer sidebar on the left displays a project structure named 'IURICODE' containing 'node_modules', 'public' (with 'index.html'), 'src' (containing 'App.js', 'index.css', and 'index.js'), and files like '.gitignore', 'package-lock.json', 'package.json', and 'README.md'. The terminal at the bottom shows the following error messages from a webpack build:

```
ERROR in ./src/App.js 4:10-30
Module not found: Error: Can't resolve './logo.svg' in 'C:\iuricode\src'

ERROR in ./src/App.js 5:0-19
Module not found: Error: Can't resolve './App.css' in 'C:\iuricode\src'

ERROR in ./src/index.js 8:0-48
Module not found: Error: Can't resolve './reportWebVitals' in 'C:\iuricode\src'

webpack compiled with 3 errors
```

Isso porque excluímos alguns arquivos porém ainda estamos chamando eles nos arquivos existentes.

- No arquivo index.css você pode excluir todo o código de dentro dele.
- No arquivo index.js iremos tirar a importação do reportWebVitals e o reportWebVitals() em nosso código.
- No arquivo App.js deixe como está, iremos trabalhar nele agora.

Como dito anteriormente, o JSX possui uma sintaxe muito semelhante ao HTML. O código abaixo (no arquivo App.js) demonstra claramente esta característica. Apesar de muito parecido, o código a seguir não é HTML e sim um trecho de código JSX.

```
● ● ●  
1 function App() {  
2   return <h1>Sou um título</h1>;  
3 }  
4  
5 export default App;
```

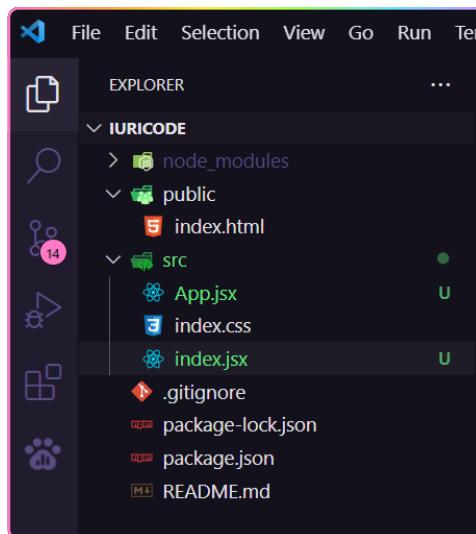
Deixe seu código no arquivo App.js dessa forma. Exclua as importações e os conteúdos de dentro do return.

Para melhor o processo de desenvolvimento podemos mudar (ou criar) os arquivos com o formato .js para .jsx

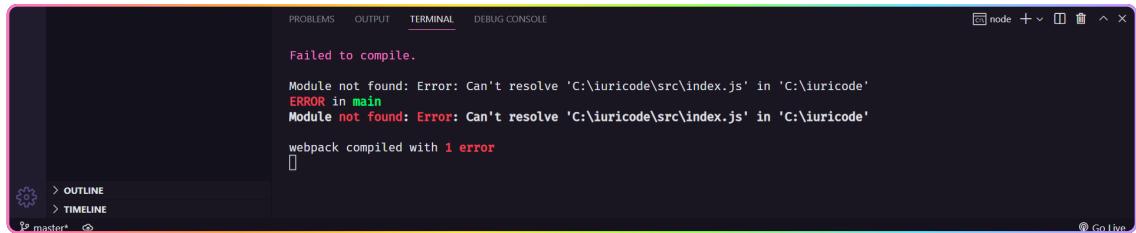
Alguns editores de código, como Visual Studio Code, podem vir mais preparados.

Por exemplo: se você utilizar .jsx, pode ser que tenha vantagens no autocomplete das tags e também no highlight do código.

Dessa forma sempre é bom mudar de .js para .jsx.



Após mudar o formato dos arquivos de .js para .jsx teremos o seguinte erro:



The screenshot shows the VS Code interface with the terminal tab selected. The output in the terminal is as follows:

```
PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

Failed to compile.

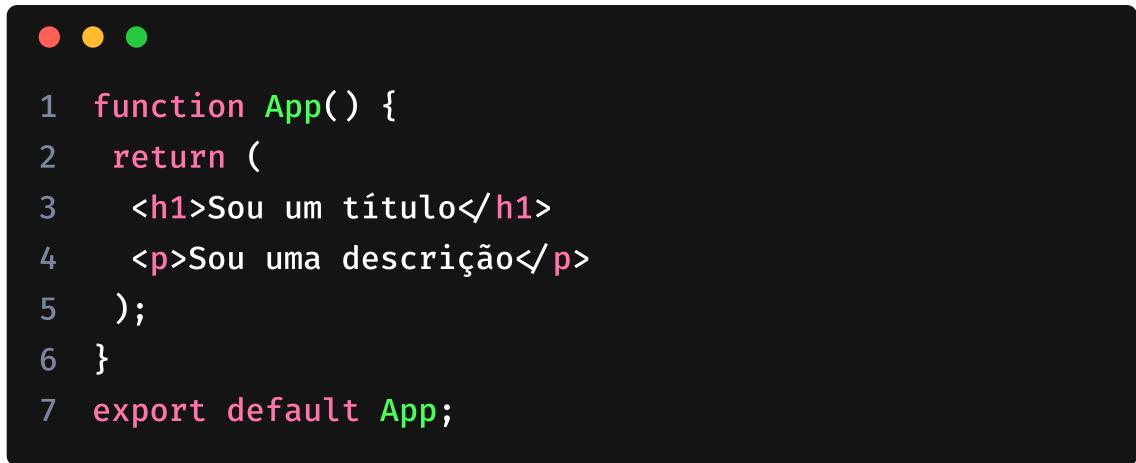
Module not found: Error: Can't resolve 'C:\iuricode\src\index.js' in 'C:\iuricode'
ERROR in main
Module not found: Error: Can't resolve 'C:\iuricode\src\index.js' in 'C:\iuricode'

webpack compiled with 1 error
```

The terminal also shows a small icon of a document with a lightning bolt.

Isso ocorreu por causa da mudança do formato, para resolver isso bastante parar o comando npm start e rodar novamente.

Uma coisa muito importante sobre o JSX é que sempre que utilizamos mais de uma tag HTML no retorno da função, precisamos englobadas em uma tag pai. No exemplo abaixo é uma demonstração incorreta.



```
1 function App() {
2   return (
3     <h1>Sou um título</h1>
4     <p>Sou uma descrição</p>
5   );
6 }
7 export default App;
```

Agora iremos corrigir a função englobando a tag H1 e P dentro de uma Div.

```
● ● ●  
1 function App() {  
2   return (  
3     <div>  
4       <h1>Sou um título</h1>  
5       <p>Sou uma descrição</p>  
6     </div>  
7   );  
8 }  
9 export default App;
```

Porém nem sempre queremos que o retorno tenha uma tag pai englobando tudo. Para isso temos o Fragment do React, que nada menos é uma tag vazia/sem significado.

```
● ● ●  
1 ...  
2 <>  
3   <h1>Sou um título</h1>  
4   <p>Sou uma descrição</p>  
5 </>  
6 ...
```

Você pode fazer dessa forma também (observe que importamos o React):

```
1 import React from "react";
2
3 function App() {
4   return (
5     <React.Fragment>
6       <h1>Sou um título</h1>
7       <p>Sou uma descrição</p>
8     </React.Fragment>
9   );
10 }
11
12 export default App;
```

Observação: para utilizar o Fragment é preciso da importação do React.

Componentes

No exemplo anterior foi mostrado a utilização do JSX, o exemplo nada mais é do que um componente no React.

No exemplo anterior foi mostrado a utilização do JSX, o exemplo nada mais é do que um componente no React.

Os componentes permitem você dividir a UI em partes independentes, reutilizáveis, ou seja, trata cada parte da aplicação como um bloco isolado, livre de outras dependências externas. Componentes são como as funções JavaScript. Eles aceitam entradas e retornam elementos React que descrevem o que deve aparecer na tela.

"E como isso é importante?"

Vou criar um simples exemplo. Imagine que temos um card que representa a apresentação de uma notícia. Sua estrutura poderá se parecida como essa:

```
● ● ●  
1 <article>  
2     
3   <div>  
4     <h2>Título de um post</h2>  
5     <p>Sou apenas uma pequena descrição</p>  
6   </div>  
7   <a href="www.iuricode.com/efront">Acessar post</a>  
8 </article>
```

Até aqui está tudo bem. Mas imagine que queremos 6 (seis) cards iguais a esse. Normalmente com HTML iremos duplicar esse código em 6 (seis) vezes, correto?

Agora imagine que por algum motivo seja preciso adicionar uma tag span para sinalizar a data da postagem. Teríamos que mudar em todos! Pior ainda, imagine que existe este card em várias páginas em nossa aplicação. Concordo que iria demorar muito para dar manutenção neste código, por causa de uma simples alteração?

E como o React resolve isso? Com os componentes.

Para criar um componente é bem simples, basta você criar um novo arquivo, exemplo, Card.jsx e criar a estrutura de um componente.

```
● ● ●  
1 function Card() {  
2   return (  
3     Aqui vai o código do exemplo anterior  
4   );  
5 }  
6  
7 export default Card;
```

Após criar o arquivo basta chamar/importar em nosso arquivo App.js

```
1 import Card from './Card';
2
3 function App() {
4   return (
5     <div>
6       <Card />
7       <Card />
8       <Card />
9     </div>
10 );
11 }
12 export default App;
```

O que vale ser comentado aqui é que um componente sempre irá começar com a letra maiúscula, exemplo, Card, Footer, Menu... caso a primeira letra seja minúscula o React irá interpretar como se fosse uma tag HTML. Outra coisa importante é o './Card' é o caminho onde se encontra o arquivo. Nesse caso o arquivo se encontra na mesma raiz do arquivo App.js.

Você pode importar dizendo o formato do arquivo como: './Card.tsx'.

Lembra da pasta public? No exemplo do Card tivemos o seguinte trecho de código:



```
1 
```

Bem, tudo que tiver dentro da pasta public pode ser acessado apenas com o nome e formato do arquivo, diferente do HTML tradicional que teria que escrever o caminho das pasta até acessar o arquivo desejado.

Mas você deve estar se perguntando “Ok Iuri, mas se caso eu queira que cada componente do arquivo Card tenha um título, descrição e/ou imagem diferente das outras?”

A resposta para isso é props!

Fim da amostra 😢

Fico feliz que tenha interesse em comprar o eFront.

Esta é apenas uma pequena amostra (não abordamos todos os tópicos do módulo de React) de como funciona a didática do eFront.

Esta amostra não possui informações como: sumário, introdução, acesso aos projetos, acesso a comunidade no Discord, questionários, leituras complementares, e minha biblioteca pessoal no Notion com +500 links para iniciar bem no frontend.

A versão completa conta com
400 páginas e 16 módulos

[COMPRAR AGORA](#)