

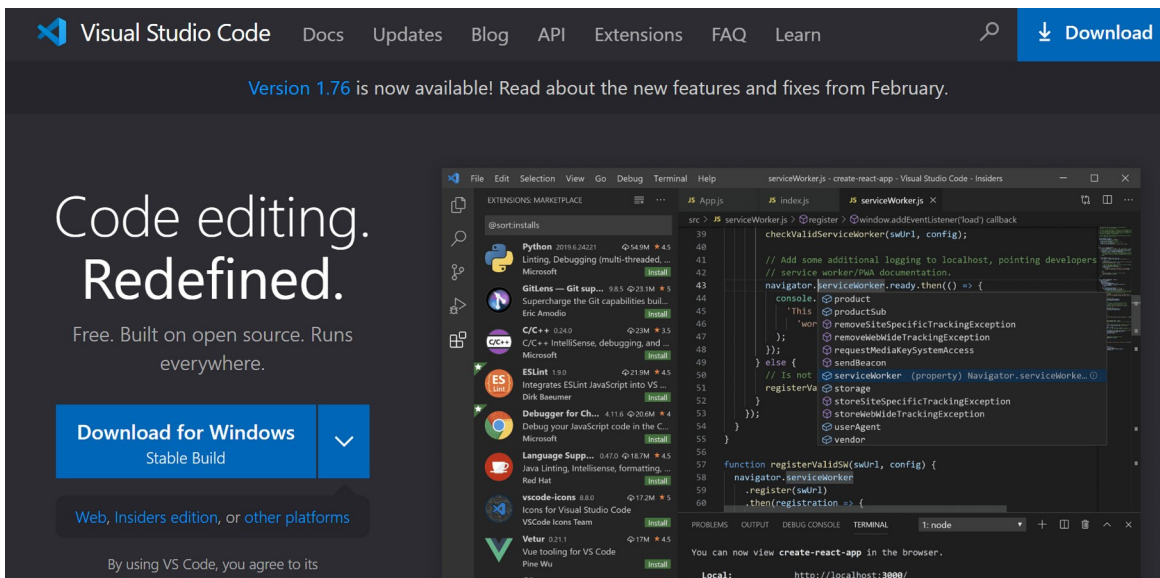
# Aula 1 – Introdução ao Front-End

## Objetivo da aula:

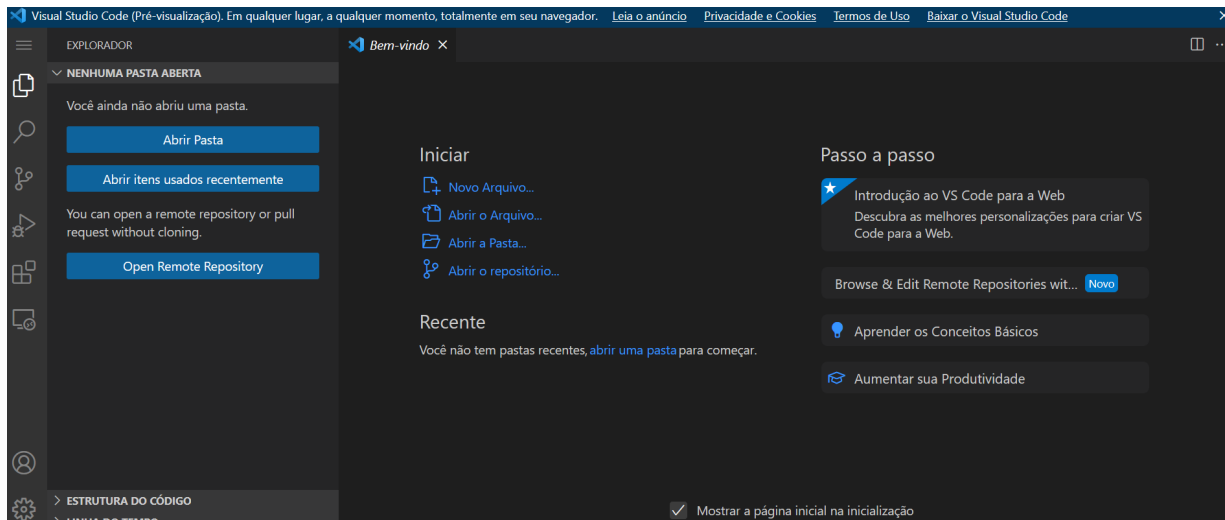
- Configurar IDE de desenvolvimento
- Instalando React e Node
- Rodando meu primeiro projeto React
- Entender a estruturação de arquivos e pastas React
- Trabalhando com Componentes e Propriedades
- Renderizar componentes e suas propriedades.
- Manipular Estados
- Atividade

## 1 – IDE de desenvolvimento

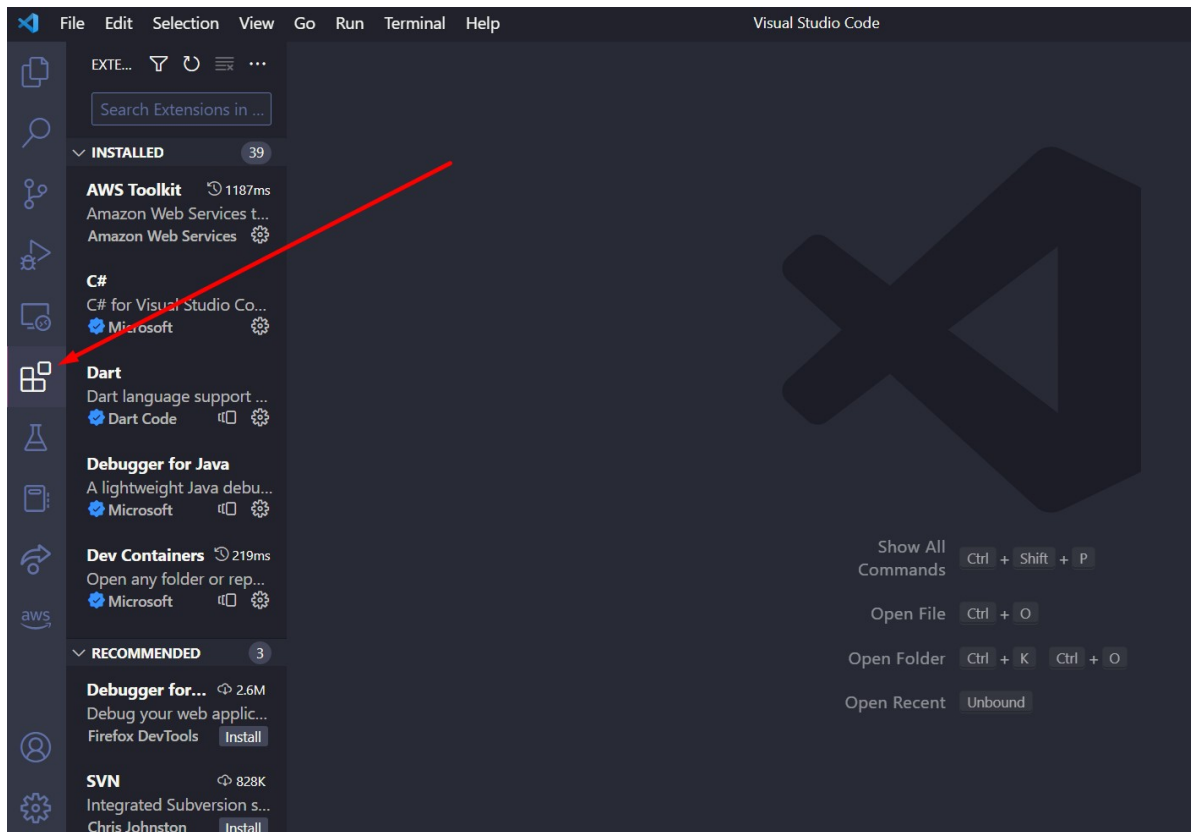
Visual Studio Code: <https://code.visualstudio.com>



VS Code Online: <https://vscode.dev>



2 – Quais extensões podem facilitar nossa vida?  
(recomendação)



- Dracula Official
- Color Highlight
- Material Icon Theme
- Live Server
- Prettier - Code formatter

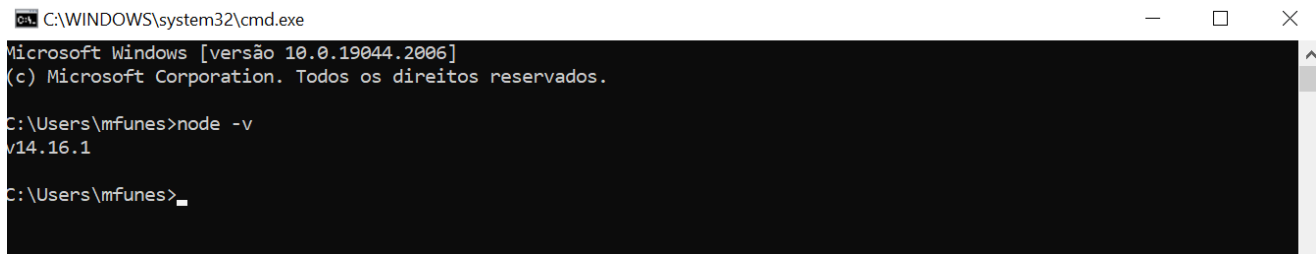
### 3 - Front-end

React: <https://pt-br.reactjs.org/>

Node: <https://nodejs.org/en/>

Abra o CMD e verifique a instalação do node

node -v



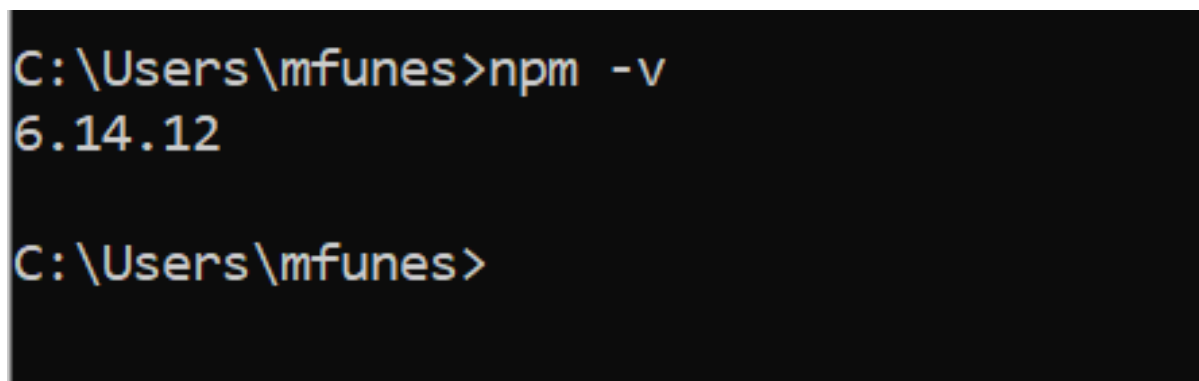
```
C:\WINDOWS\system32\cmd.exe
Microsoft Windows [versão 10.0.19044.2006]
(c) Microsoft Corporation. Todos os direitos reservados.

C:\Users\mfunes>node -v
v14.16.1

C:\Users\mfunes>
```

Gerenciador de pacotes

npm -v (o mais utilizado)



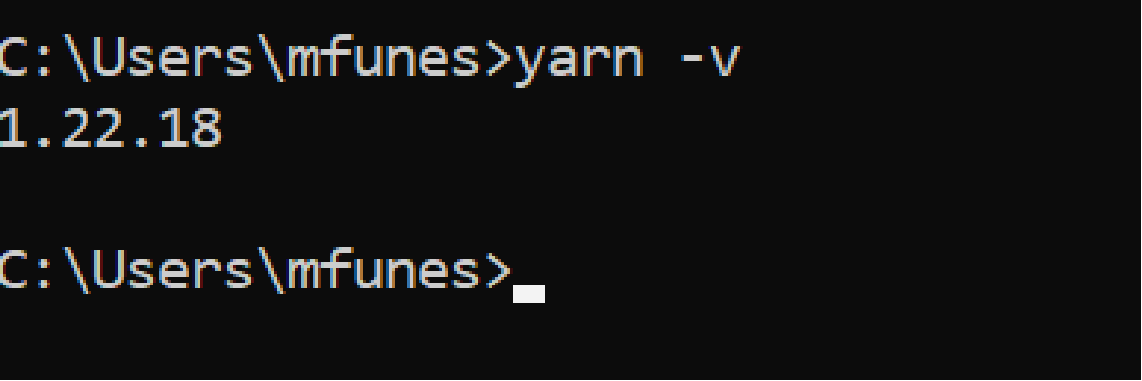
```
C:\Users\mfunes>npm -v
6.14.12

C:\Users\mfunes>
```

Outra opção de gerenciador de pacote é o YARN

<https://classic.yarnpkg.com/lang/en/docs/install/#windows-stable>

```
npm install --global yarn
```

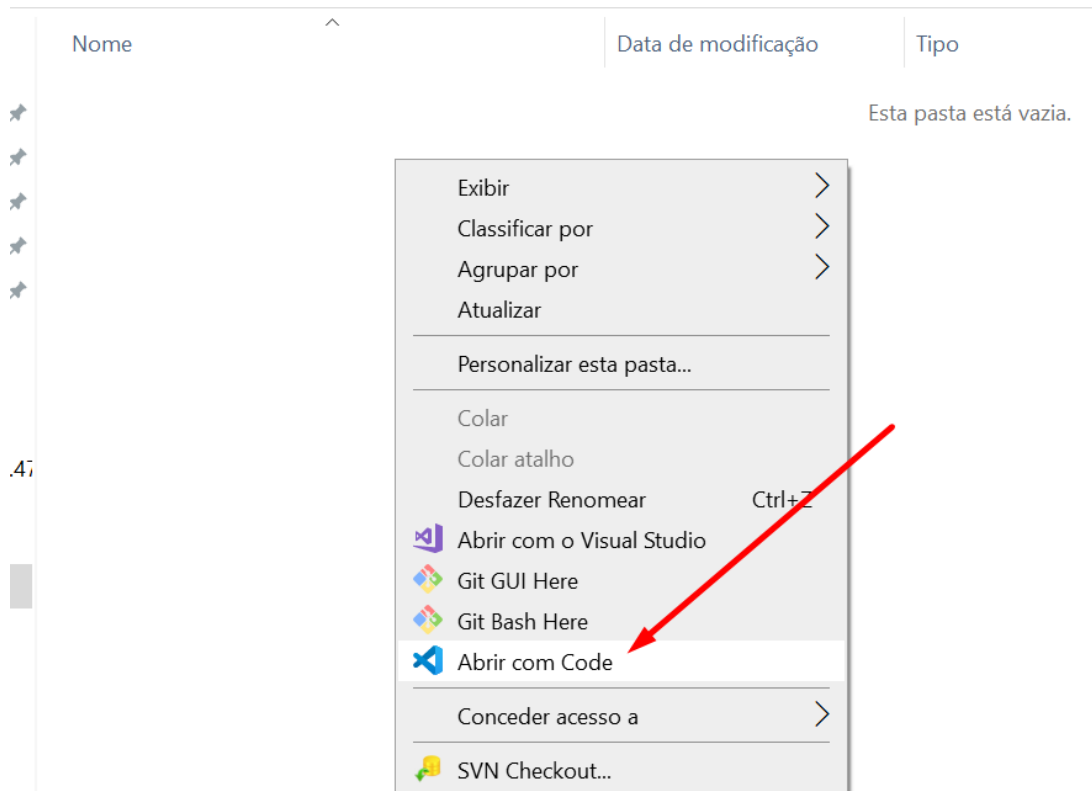
A screenshot of a Windows command prompt terminal. The first line shows the command 'yarn -v' being executed, and the second line shows the output '1.22.18'. The third line shows the prompt 'C:\Users\mfunes>' with a cursor.

```
C:\Users\mfunes>yarn -v  
1.22.18  
  
C:\Users\mfunes>_
```

Tudo pronto para criarmos nossa primeira interface web com react

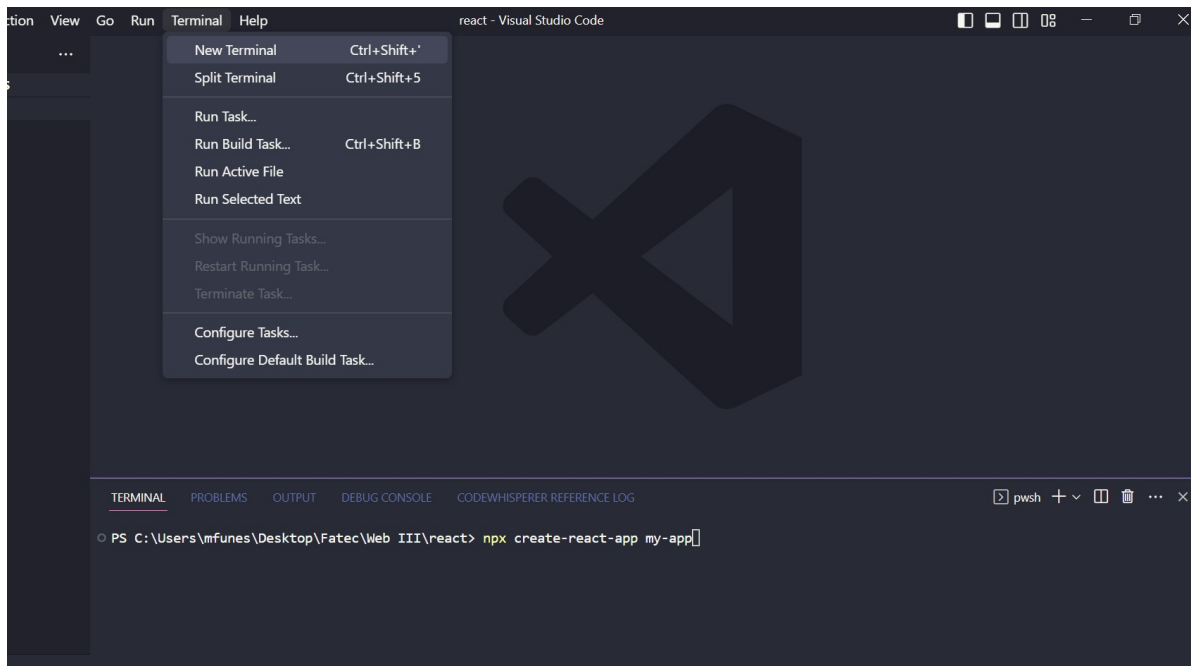
<https://pt-br.reactjs.org/docs/create-a-new-react-app.html>

1 – Crie uma pasta onde desejar, dentro da pasta crie utilize o botão direito do mouse para começar um novo projeto no VSCODE.



2 – Dentro do VSCode, abra o terminal e utilize o comando abaixo para criar seu primeiro projeto em REACT.

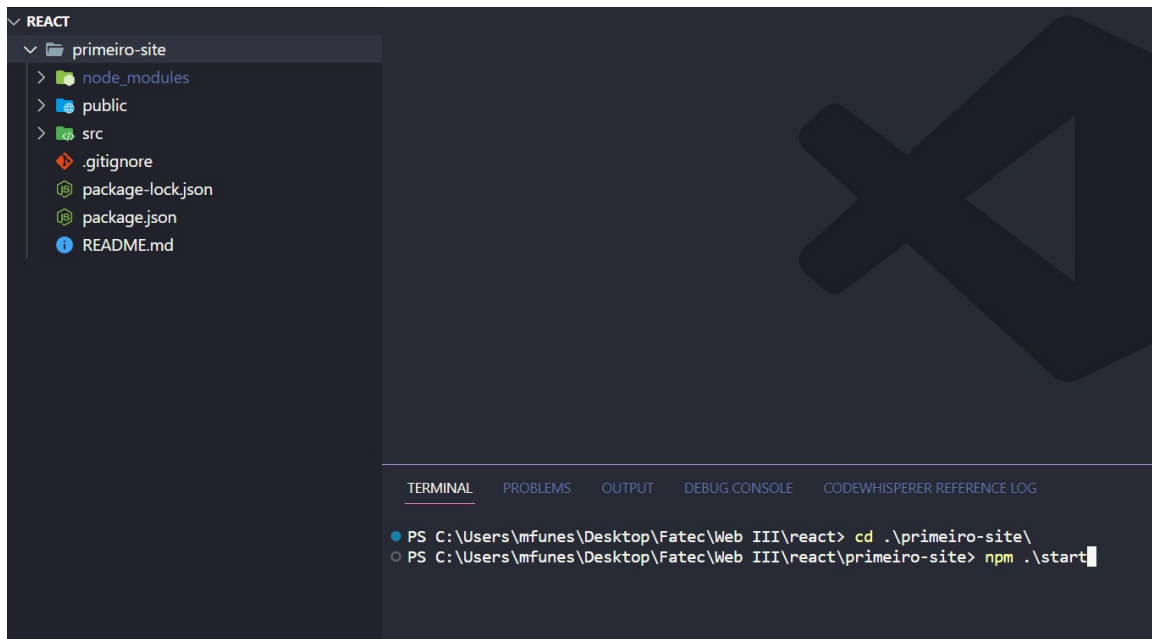
```
npx create-react-app primeiro-site
```



3 – Após o download de todos os componentes do React serem instalados podemos agora entrar na pasta onde ele foi instalado (primeiro-site) e “rodarmos” o projeto.

```
cd primeiro-site
```

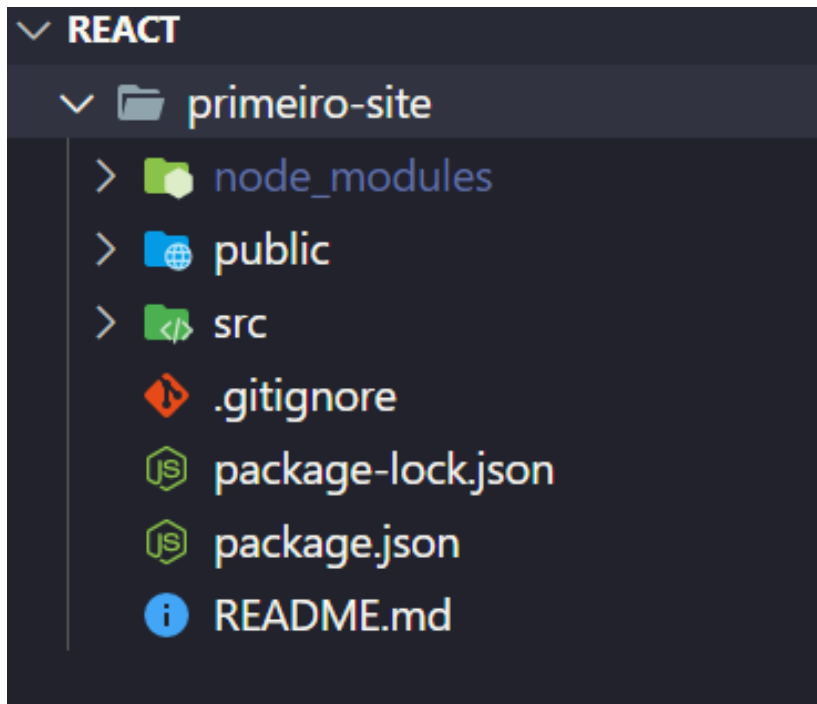
```
npm start
```



4 - Você deve ser capaz de ver o site executando no navegador:

```
You can now view primeiro-site in the browser.  
  
Local:      http://localhost:3000  
On Your Network:  http://172.24.160.1:3000  
  
Note that the development build is not optimized.  
To create a production build, use npm run build.  
  
webpack compiled successfully
```

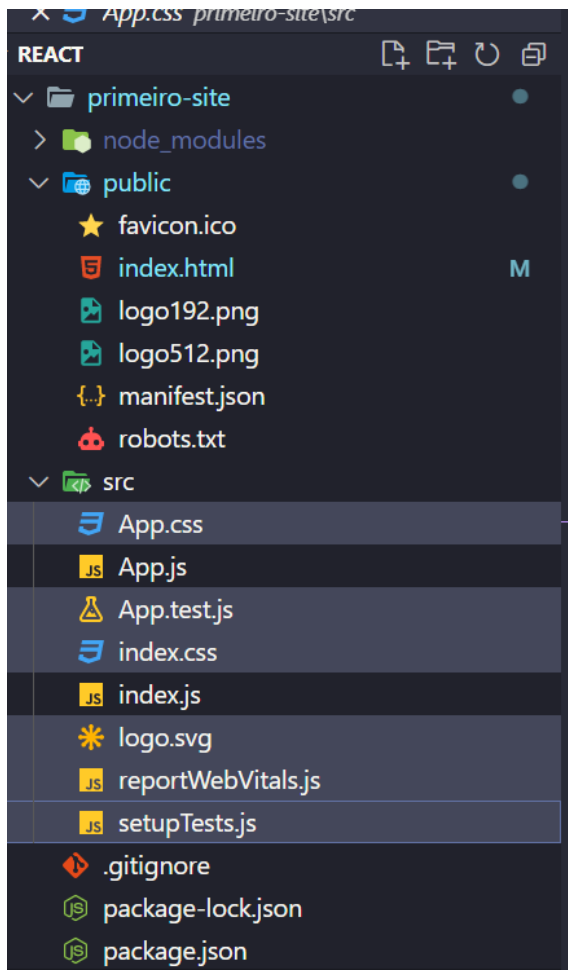
5 – Entendendo a estrutura de um projeto em REACT



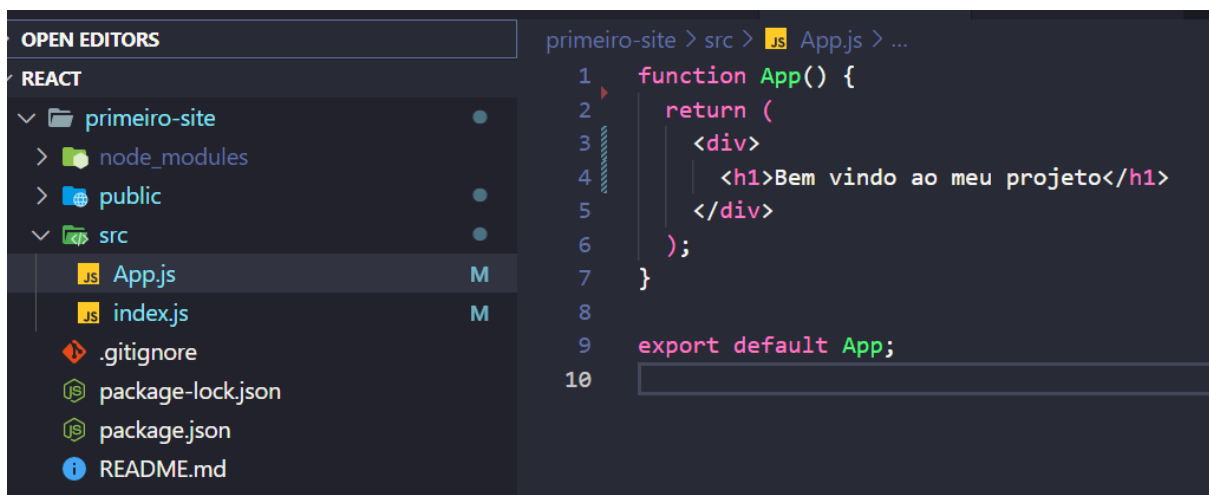
- Package.json: informações principais do projeto (não editamos esse arquivos, ele é atualizado automaticamente)
- Pasta public: pasta que contem os arquivos que serão de produção, ou seja, serão de acesso público.
- Pasta SRC (SOURCE): pasta principal que contém toda a estrutura da solução front-end.

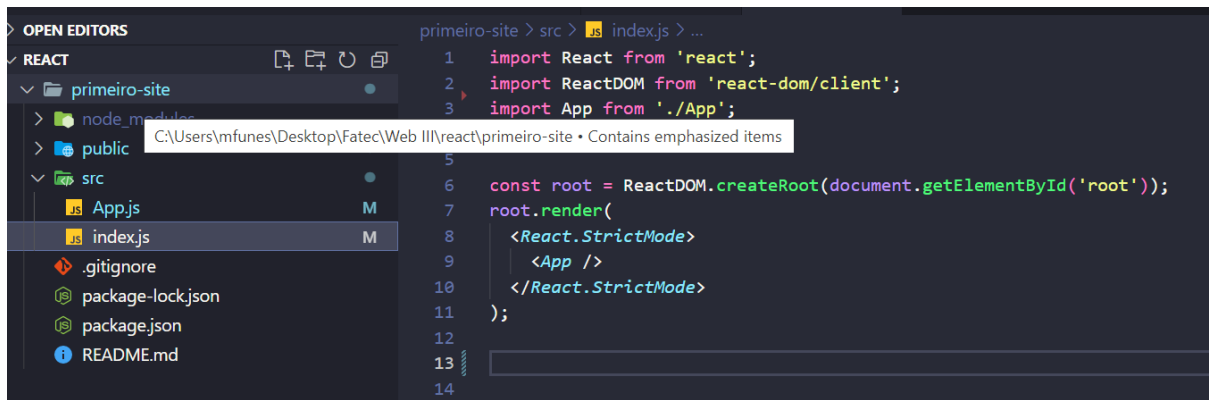
6 – Para deixar o projeto mais limpo, podemos deletar os arquivos abaixo:





7 - Começando a customização da pagina Home. Vamos limpar alguns códigos em App.js e index.js





The screenshot shows a code editor with a file explorer on the left and a code editor on the right. The file explorer shows a project named 'primeiro-site' with files like 'App.js', 'index.js', '.gitignore', 'package-lock.json', 'package.json', and 'README.md'. The code editor shows the following code in 'index.js':

```
1 import React from 'react';
2 import ReactDOM from 'react-dom/client';
3 import App from './App';
4
5
6 const root = ReactDOM.createRoot(document.getElementById('root'));
7 root.render(
8   <React.StrictMode>
9     <App />
10  </React.StrictMode>
11 );
12
13
14
```

## 8 – Entendendo a chamada de componentes dentro do React.

O que é um componente?



### Declarativo

React faz com que a criação de UIs interativas seja uma tarefa fácil. Crie views simples para cada estado na sua aplicação, e o React irá atualizar e renderizar de forma eficiente apenas os componentes necessários na medida em que os dados mudam.

Views declarativas fazem com que seu código seja mais previsível e simples de depurar.

### Baseado em componentes

Crie componentes encapsulados que gerenciam seu próprio estado e então, combine-os para criar UIs complexas.

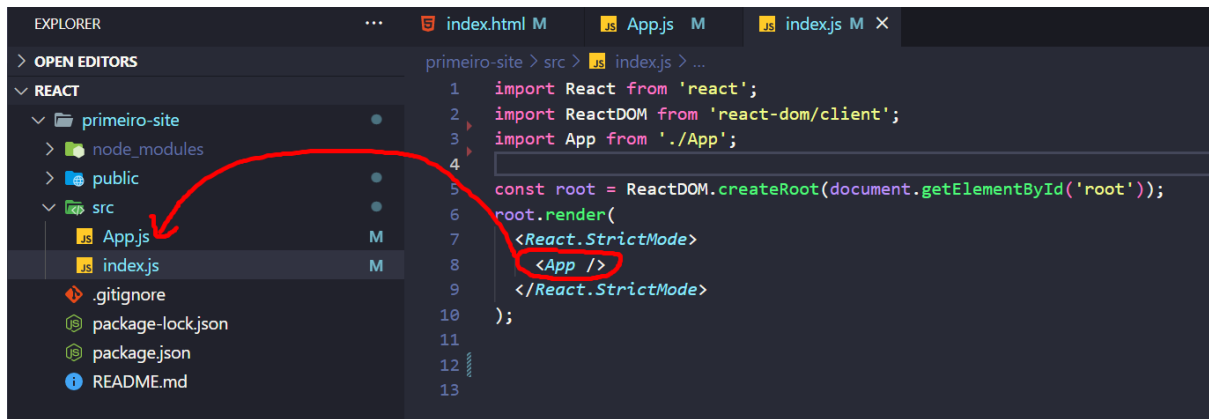
Como a lógica do componente é escrita em JavaScript e não em templates, você pode facilmente passar diversos tipos de dados ao longo da sua aplicação e ainda manter o estado fora do DOM.

### Aprenda uma vez, use em qualquer lugar

Não fazemos suposições sobre as outras tecnologias da sua stack, assim você pode desenvolver novos recursos com React sem reescrever o código existente.

O React também pode ser renderizado no servidor, usando Node, e ser usado para criar aplicações mobile, através do React Native.

Quando algum cliente acessa nossa página, o React irá chamar o `index.js` que irá renderizar nosso **componente** dentro do front-end (`App.js`)



Imagine que você precisa manter sempre o mesmo header de uma página mesmo se existir mudança de páginas pelo usuário. Podemos criar um componente para definir como ser o comportamento do header. Na imagem acima podemos entender que cada quadradro vermelho pode ser um componente diferente com modo de interagir diferentes.

9 - Crie um novo componente chamado Exemplo.js dentro da pasta src como abaixo indicado

The screenshot shows the VS Code interface. On the left, the Explorer sidebar displays the project structure: 'primeiro-site' (containing 'node\_modules', 'public', and 'src'), and 'src' (containing 'App.js', 'Exemplo.js', and 'index.js'). The 'Exemplo.js' file is selected. The main editor shows the code for 'Exemplo.js':

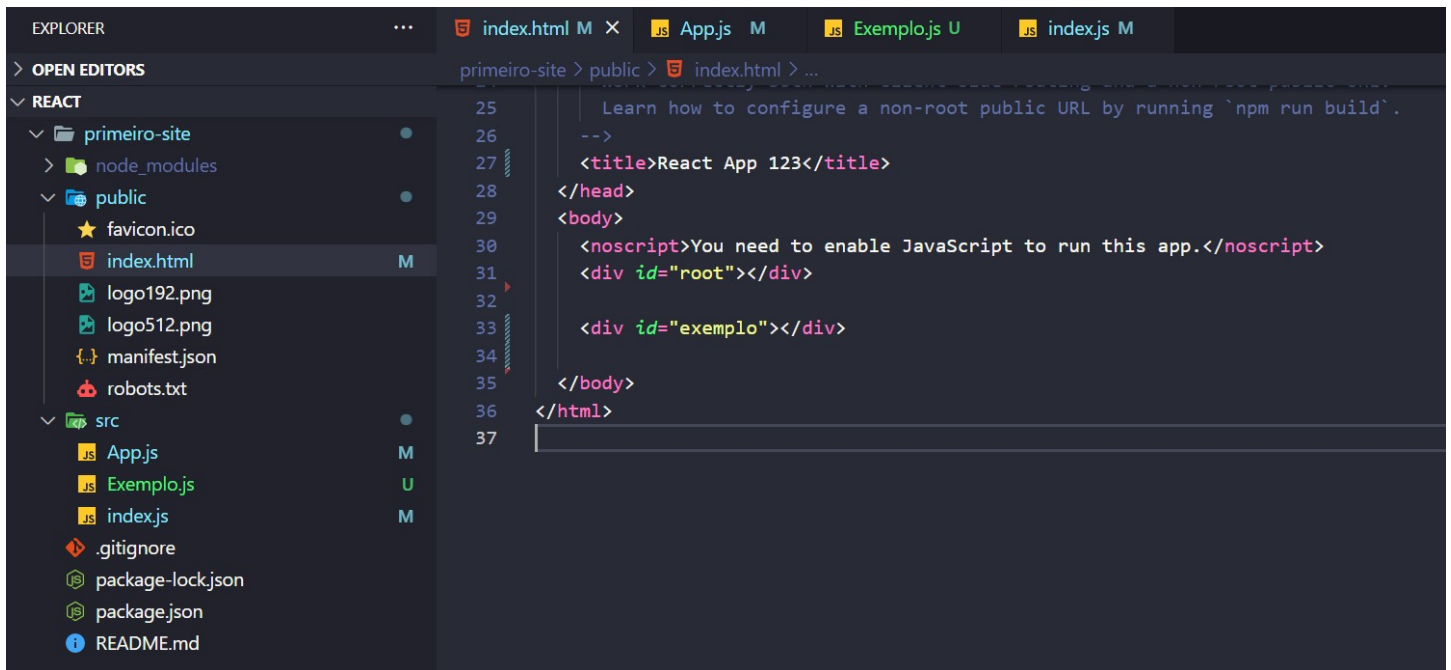
```
1 function exemplo() {  
2   return (  
3     <div>  
4       <h1>Será que funciona mesmo?</h1>  
5     </div>  
6   );  
7 }  
8  
9 export default exemplo;  
10
```

## 10 – Renderize o novo componente Exemplo.js no index.js

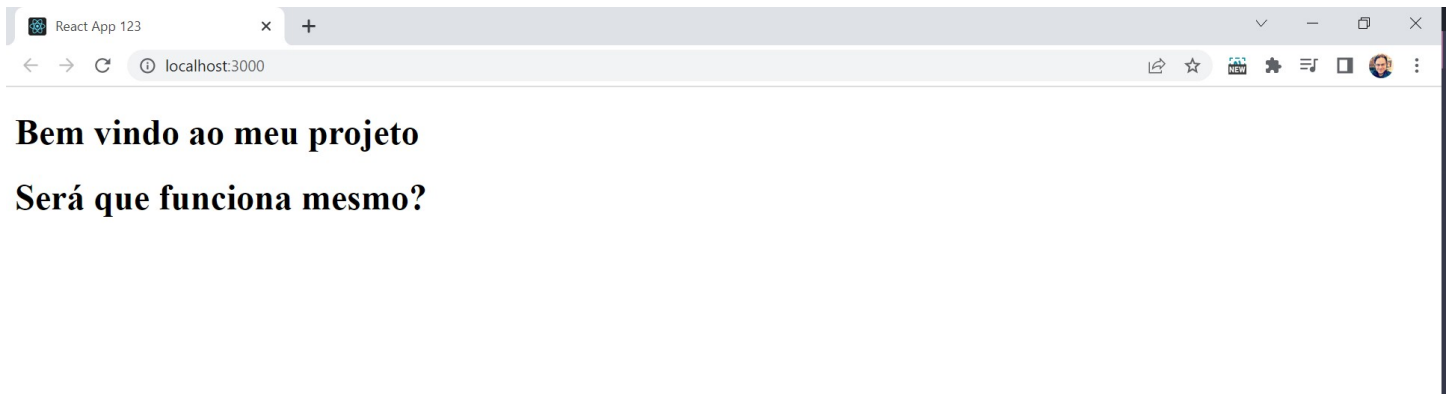
The screenshot shows the VS Code interface with the 'index.js' file selected in the Explorer sidebar. The main editor displays the code for 'index.js':

```
1 import React from 'react';  
2 import ReactDOM from 'react-dom/client';  
3 import App from './App';  
4 import Exemplo from './Exemplo'  
5  
6 const root = ReactDOM.createRoot(document.getElementById('root'));  
7 root.render(  
8   <React.StrictMode>  
9     <App />  
10   </React.StrictMode>  
11 );  
12  
13 const exemplo = ReactDOM.createRoot(document.getElementById('exemplo'));  
14 exemplo.render(  
15   <React.StrictMode>  
16     <Exemplo />  
17   </React.StrictMode>  
18 );  
19  
20  
21
```

## 11 – Mostre onde o React deve colocar a renderização feita no index.js dentro de index.html por meio do id=exemplo.



## 12 – Esse deve ser o resultado



## 13 – Reutilizando componentes.

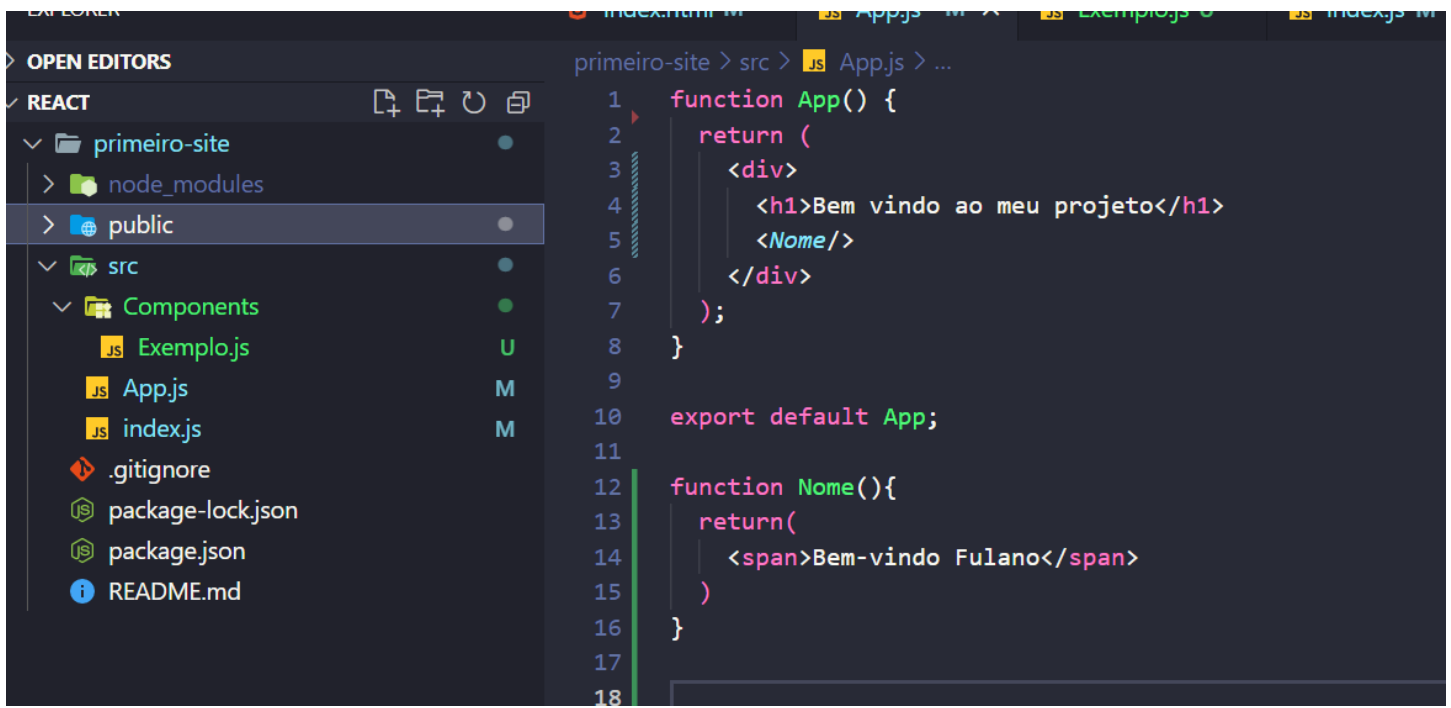
Criei um novo componente chamado Nome dentro do App.js e reutilize o componente dentro do componente App já existente.

The screenshot shows the VS Code interface with the Explorer sidebar on the left and the Editor on the right. The Explorer sidebar shows the project structure for 'primeiro-site', including 'node\_modules', 'public', and 'src'. The 'src' folder is expanded, showing 'App.js', 'Exemplo.js', and 'index.js'. The Editor shows the code for 'App.js' with the following content:

```
1 function App() {
2   return (
3     <div>
4       <h1>Bem vindo ao meu projeto</h1>
5       <Nome/>
6     </div>
7   );
8 }
9
10 export default App;
11
12 function Nome(){
13   return(
14     <span>Bem-vindo Fulano</span>
15   )
16 }
17
18
```

14 – Perceba que não seria uma boa prática deixar todos os nossos componentes dentro do App.js pois no futuro ficaria difícil achar e organizar. Portanto, vamos criar uma pasta chamada Components para que todos os componentes possam ficar organizados.



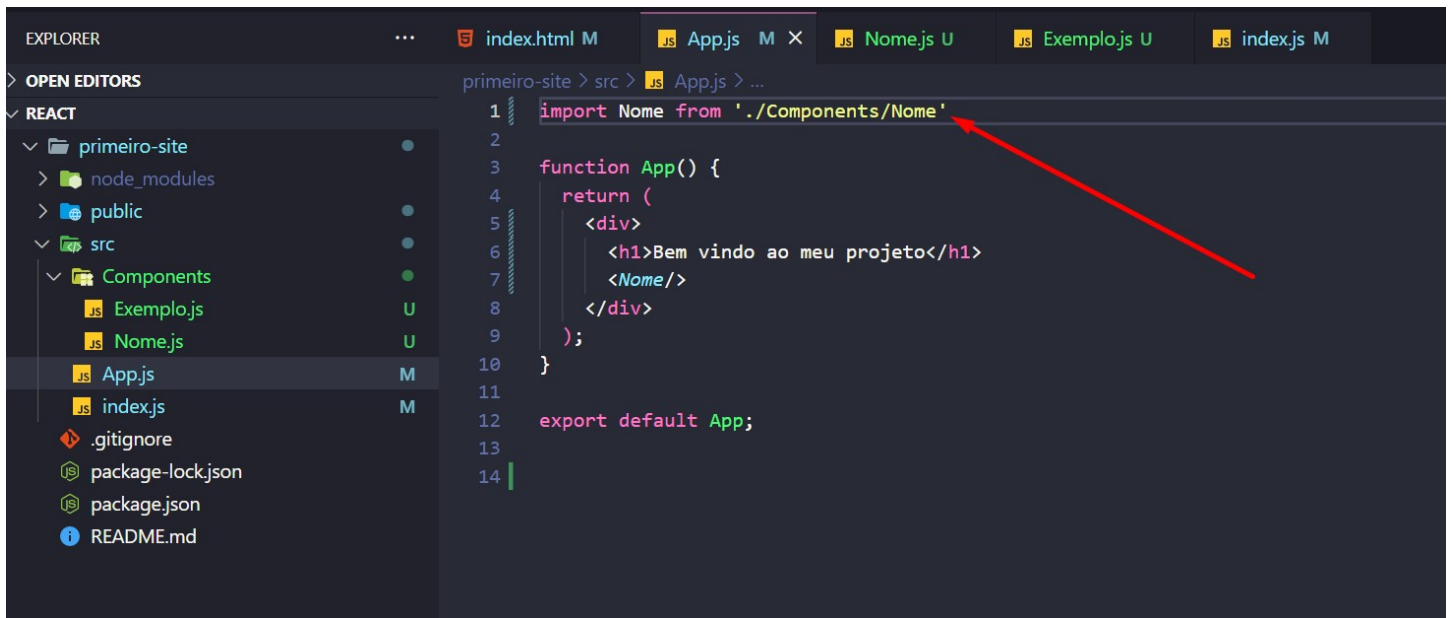


The screenshot shows a code editor with a sidebar on the left displaying the project structure. The sidebar is titled 'OPEN EDITORS' and shows a tree view of the project. The tree view is expanded to show the 'src' directory, which contains a 'Components' folder. Inside 'Components', there are three files: 'Exemplo.js', 'App.js', and 'index.js'. The 'App.js' file is selected. The main editor area shows the code for 'App.js'. The code defines a function 'App()' that returns a JSX element. The JSX element consists of a 'div' containing an 'h1' with the text 'Bem vindo ao meu projeto' and a 'Nome' component. The 'Nome' component is imported from 'Exemplo.js'. The code also defines a function 'Nome()' that returns a JSX element consisting of a 'span' with the text 'Bem-vindo Fulano'.

```
1 function App() {
2   return (
3     <div>
4       <h1>Bem vindo ao meu projeto</h1>
5       <Nome/>
6     </div>
7   );
8 }
9
10 export default App;
11
12 function Nome(){
13   return(
14     <span>Bem-vindo Fulano</span>
15   )
16 }
17
18
```

15 – Crie um novo arquivo dentro Components chamado nome.js e mude nosso componente de App.js para o arquivo criado. Não se esqueça de atualizar as importações.





## 16 – Entendendo propriedades em React (props)

Crie um propriedade chamada aluno dentro de App.js, depois faça com que nosso componente Nome.js passe a reconhecer aluno como uma propriedade válida por meio de props:

```
primeiro-site > src > App.js > App
1 import Nome from './Components/Nome'
2
3 function App() {
4   return (
5     <div>
6       <h1>Bem vindo ao meu projeto</h1>
7       <Nome aluno="Márcio" />
8     </div>
9   );
10 }
11
12 export default App;
13
14
```

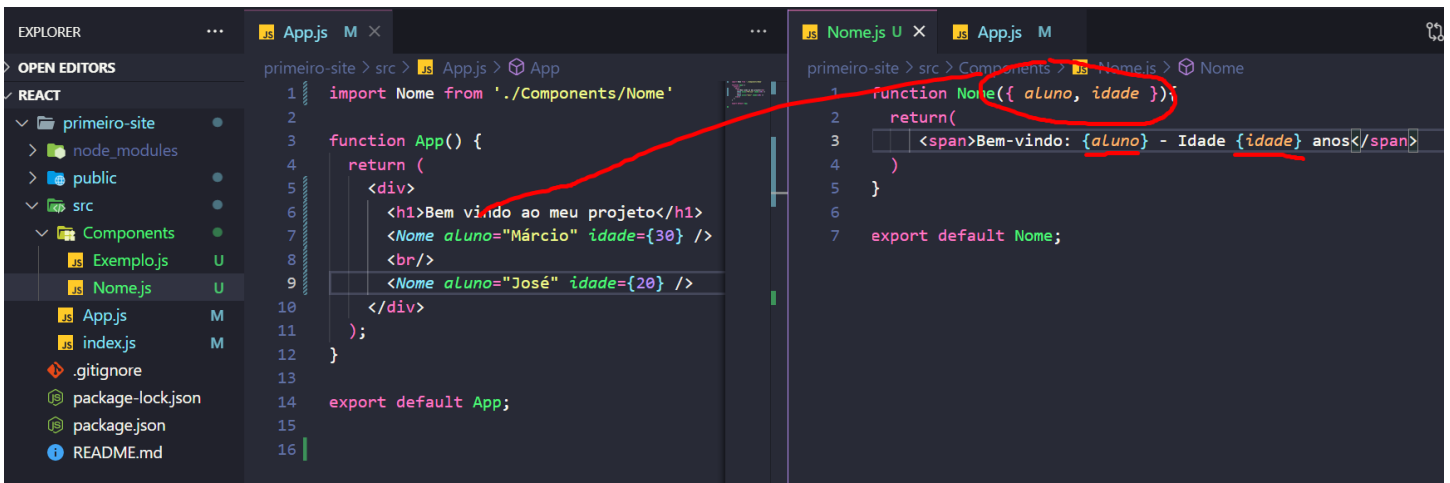
```
primeiro-site > src > Components > Nome.js > Nome
1 function Nome(props){
2   return(
3     <span>Bem-vindo: {props.aluno}</span>
4   )
5 }
6
7 export default Nome;
```

17 – Outra forma de trabalhar está em desestruturar as propriedades.

```
primeiro-site > src > App.js > App
1 import Nome from './Components/Nome'
2
3 function App() {
4   return (
5     <div>
6       <h1>Bem vindo ao meu projeto</h1>
7       <Nome aluno="Márcio" />
8     </div>
9   );
10 }
11
12 export default App;
13
14
```

```
primeiro-site > src > Components > Nome.js > default
1 function Nome({ aluno }){
2   return(
3     <span>Bem-vindo: {aluno}</span>
4   )
5 }
6
7 export default Nome;
```

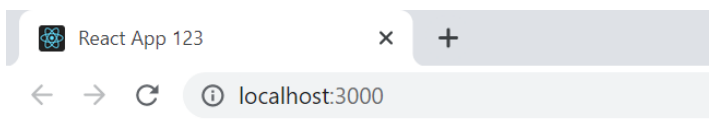
18 – Vamos deixar esse componente mais dinâmico passando novas propriedade desestruturadas. Além do nome passe também a idade de dois alunos.



## 19 – Trabalhando com Estados (useStates)

As propriedades que fizemos são estatáticas onde nos passamos o valor fixo. Mas imagine ter que atualizar a página (F5) toda vez que o usuário quiser atualizar os dados na tela. Para evitar isso trabalhamos manipulando os estados.

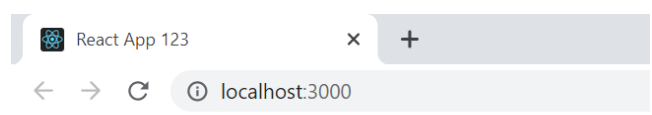
Faça um botão que ao ser clicado mude o estado do nome presente na propriedade Aluno:



**Minha página Web!**

**Olá Aluno**

Mudar nome



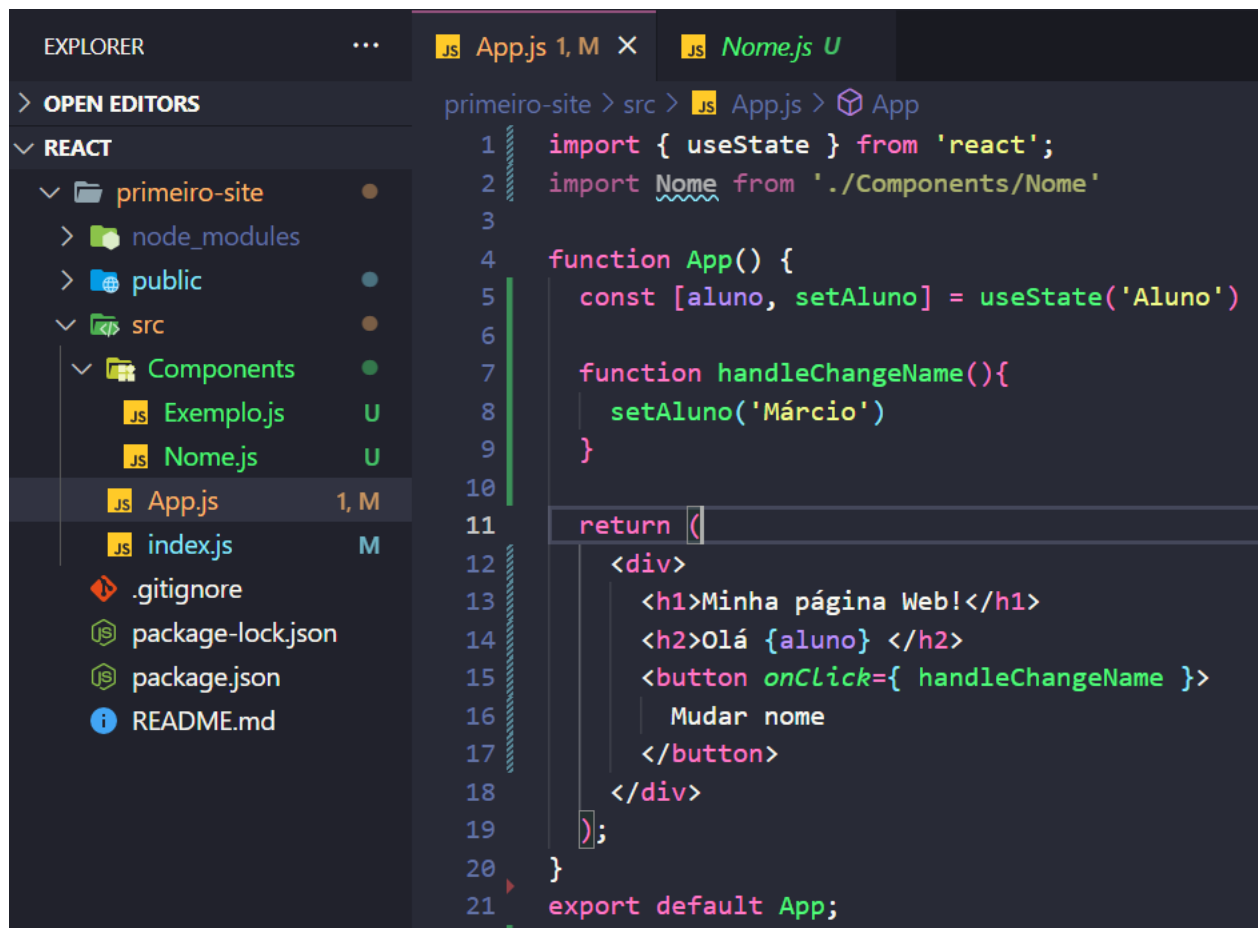
**Minha página Web!**

**Olá Márcio**

Mudar nome

Antes de ver a resposta abaixo, tente fazer sozinho(a)!



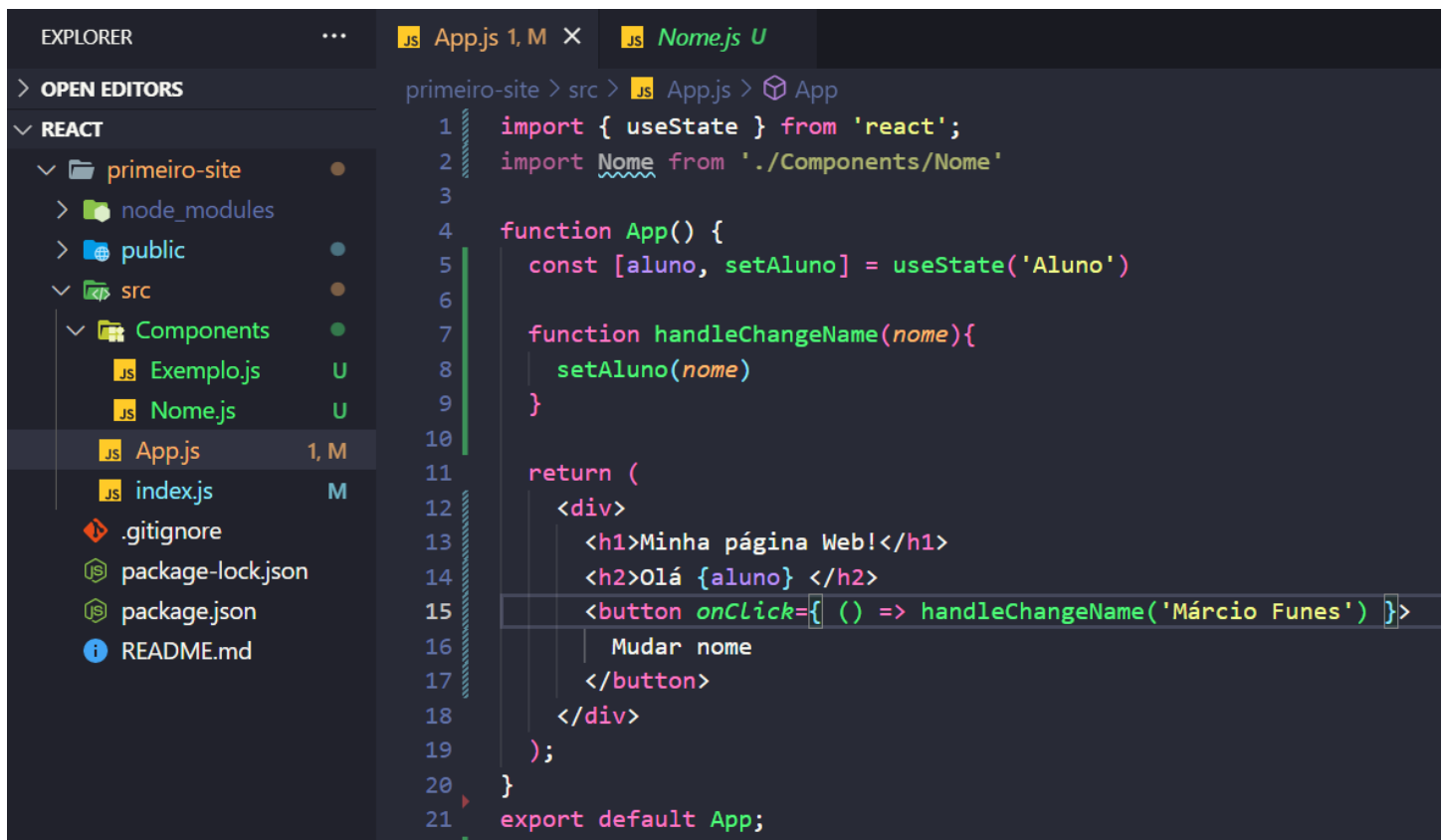


The image shows a VS Code editor interface. On the left, the Explorer sidebar displays the project structure under the name 'primeiro-site'. The structure includes a 'src' directory containing 'Components' (with 'Exemplo.js' and 'Nome.js') and 'App.js' (1, M). Other files in the root include 'index.js' (M), '.gitignore', 'package-lock.json', 'package.json', and 'README.md'. The main editor area shows the 'App.js' file with the following code:

```
1 import { useState } from 'react';
2 import Nome from './Components/Nome'
3
4 function App() {
5   const [aluno, setAluno] = useState('Aluno')
6
7   function handleChangeName(){
8     setAluno('Márcio')
9   }
10
11   return (
12     <div>
13       <h1>Minha página Web!</h1>
14       <h2>Olá {aluno}</h2>
15       <button onClick={ handleChangeName }>
16         Mudar nome
17       </button>
18     </div>
19   );
20 }
21 export default App;
```

20 – Agora vamos fazer com que o valor seja passado como um parâmetro dentro da função handleChangeName.

20 – Agora vamos fazer com que o valor seja passado como um parâmetro dentro da função handleChangeName.



The screenshot shows a VS Code editor with two tabs: 'App.js 1, M' and 'Nome.js U'. The Explorer sidebar on the left shows the project structure for 'primeiro-site', including 'src' and 'Components' folders. The main editor displays the code for 'App.js'.

```
1 import { useState } from 'react';
2 import Nome from './Components/Nome'
3
4 function App() {
5   const [aluno, setAluno] = useState('Aluno')
6
7   function handleChangeName(nome){
8     setAluno(nome)
9   }
10
11   return (
12     <div>
13       <h1>Minha página Web!</h1>
14       <h2>Olá {aluno} </h2>
15       <button onClick={() => handleChangeName('Márcio Funes')} >
16         Mudar nome
17       </button>
18     </div>
19   );
20 }
21 export default App;
```

Bora testar os conhecimentos?

Agora trabalhar com formulários em React para praticar: **Estados, Componentes e Propriedades**.

1 – Crie um novo componente na pasta Componentes chamado Cadastro.js.

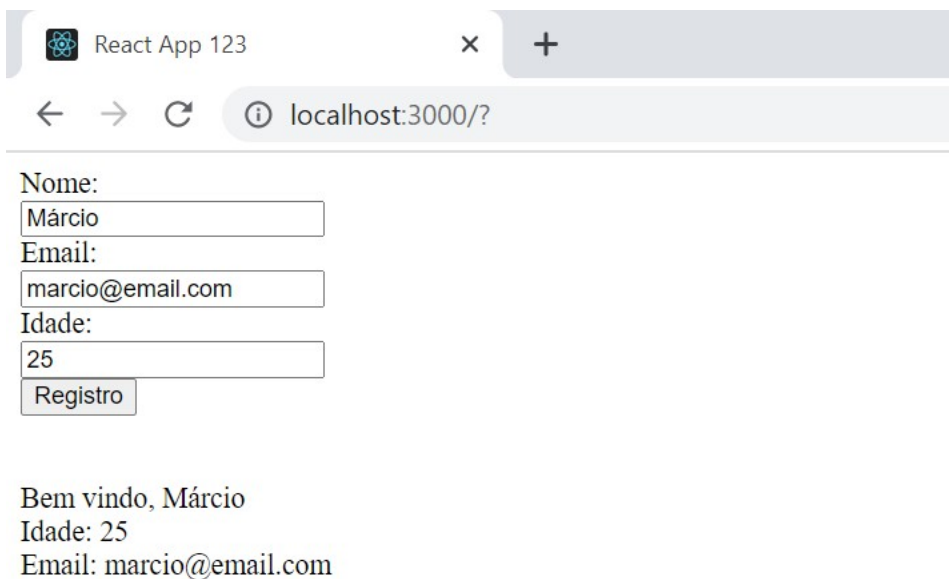
2 – Nesse componente crie um formulário que deverá solicitar ao usuário seu nome, e-mail e idade.

3 - Após clicar em registrar as mesmas informações deverão aparecer na tela.

4 - Deverá existir uma função chamada handleRegistro que irá gerenciar o estado das propriedades desse componente criado.

5 – No arquivo index.js crie uma nova renderização para o Componente Cadastro

6 – No arquivo index.html chame o cadastro por meio de div



React App 123

localhost:3000/?

Nome:

Email:

Idade:

Bem vindo, Márcio  
Idade: 25  
Email: marcio@email.com