

# Getting Started

quarta-feira, 30 de agosto de 2023 21:28

## Why learn React?


Essa questão é facilmente respondida neste [link](#). Fora isto, o que pretendo fazer nesta série de arquivos é documentar uma série de informações que julgo importante para o estudo desta biblioteca do JavaScript. Dito isto, vamos aos detalhes.

## JavaScript e DOM

O tutorial nos mostra que, por ser uma linguagem imperativa, a atualização de documentos HTML por meio da manipulação da árvore DOM com JavaScript pode ser um pouco verbosa. E é por este motivo que recorremos ao react, pois nos permite utilizar uma programação **declarativa**. Em resumo, este tipo de "técnica" consiste em realizar uma série de pedidos à biblioteca de forma que esta decide a forma como vai atualizar o DOM para a exibição no navegador.

## Primeiros passos no desenvolvimento

Para este pequeno exemplo, criamos um documento HTML simples e já adicionei os scrips que carregam o React e o Babel, que compila o JSX em JavaScript.

```
<> index.html >  html
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <meta charset="UTF-8">
5      <meta name="viewport" content="width=device-width,
6          initial-scale=1.0">
7      <title>Getting started with react</title>
8  </head>
9  <body>
10     <div id="app"></div>
11
12     <script crossorigin src="https://unpkg.com/react@18/umd/
13         react.development.js"></script>
14     <script crossorigin src="https://unpkg.com/react-dom@18/
15         umd/react-dom.development.js"></script>
16
17     <!-- Adding babel for JSX conversion because React uses
18         JSX -->
19     <script src="https://unpkg.com/@babel/standalone/babel.
20         min.js"></script>
21     <script src="index.js" type="text/jsx"></script>
22 </body>
23 </html>
```

## E o que raios é JSX?

JSX segue o modelo de escrita do HTML, mas com algumas [mudanças](#). Trata-se de uma extensão de sintaxe, que traduz, assim como o HTML faz, a linguagem de marcação em elementos de interface gráfica.

Com o nosso arquivo .js, fazemos o básico: pegamos a <div> pelo id e inserimos nela um título <h1> usando, desta vez, o React.

```
<> index.html X JS index.js X

JS index.js > ...
1  const app = document.getElementById("app");
2  ReactDOM.render(<h1>Getting started with react!</h1>, app);
3
```

Um detalhe interessante é que, para funcionar corretamente, devemos informar que o tipo de escrita é do tipo JSX. Deste modo:

```
<script src="index.js" type="text/jsx"></script>
```

Assim, obtemos o resultado que queremos:

# Getting started with react!

## Conceitos básicos sobre desenvolvimento React

### Components

- Components: pequenas divisões de código da interface, são como blocos que constroem, aos poucos, toda a estrutura da UI;

Em React, componentes são funções, mas que escrevemos com letra inicial maiúscula (para não confundir com uma tag HTML normal) e usamos com colchetes angulares (<>) durante a passagem de parâmetro.

```
function Header() {
  return <h1>Getting started with Components in React!</h1>
}

ReactDOM.render(<Header />, app);
```

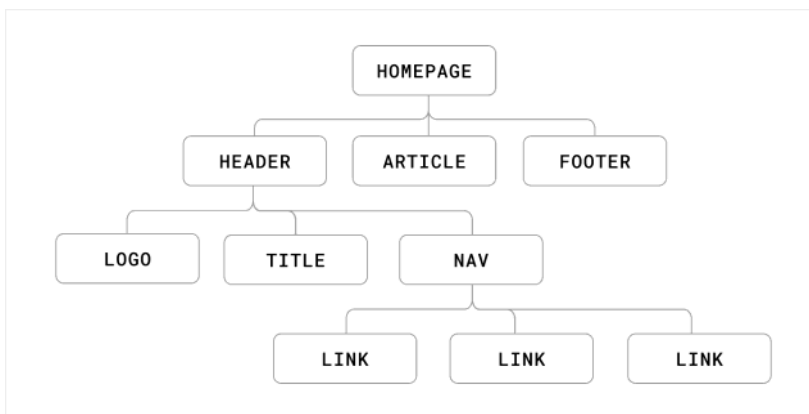
Podemos aninhar componentes do seguinte modo:

```
function Header() {
  return <h1>Getting started with Components in React!</h1>
}

function HomePage() {
  return (
    <div>
      <Header />
    </div>
  )
}

ReactDOM.render(<HomePage />, app);
```

Assim, podemos criar uma árvore que elucida um aninhamento mais complexo dos componentes que podem ser criados em uma página principal.



## Props

- Props: seguindo um comportamento similar ao de argumentos em uma função, são propriedades que podem ser passadas para componentes em react.

Dito isto, fica fácil de imaginar que se eu precisar personalizar alguma informação que vai aparecer em um componente, eu precisaria recorrer aos props para alcançar meu objetivo. Principalmente se eu não sei previamente qual informação vai ser gerada no layout.

Dentro do componente *HomePage*, eu posso passar uma propriedade *title* dentro do meu componente *Header*. A tag filha *Header*, deste modo, pode aceitar props como parâmetro, que é nada mais do que um objeto com a propriedade *title*. Assim, posso passar este objeto de uma maneira desestruturada, evidenciando suas propriedades, de modo que posso utilizá-las no código do meu componente.

No entanto, não basta utilizar o parâmetro normalmente dentro de uma tag, devo usar as chaves ({}), para passar essas variáveis.

```

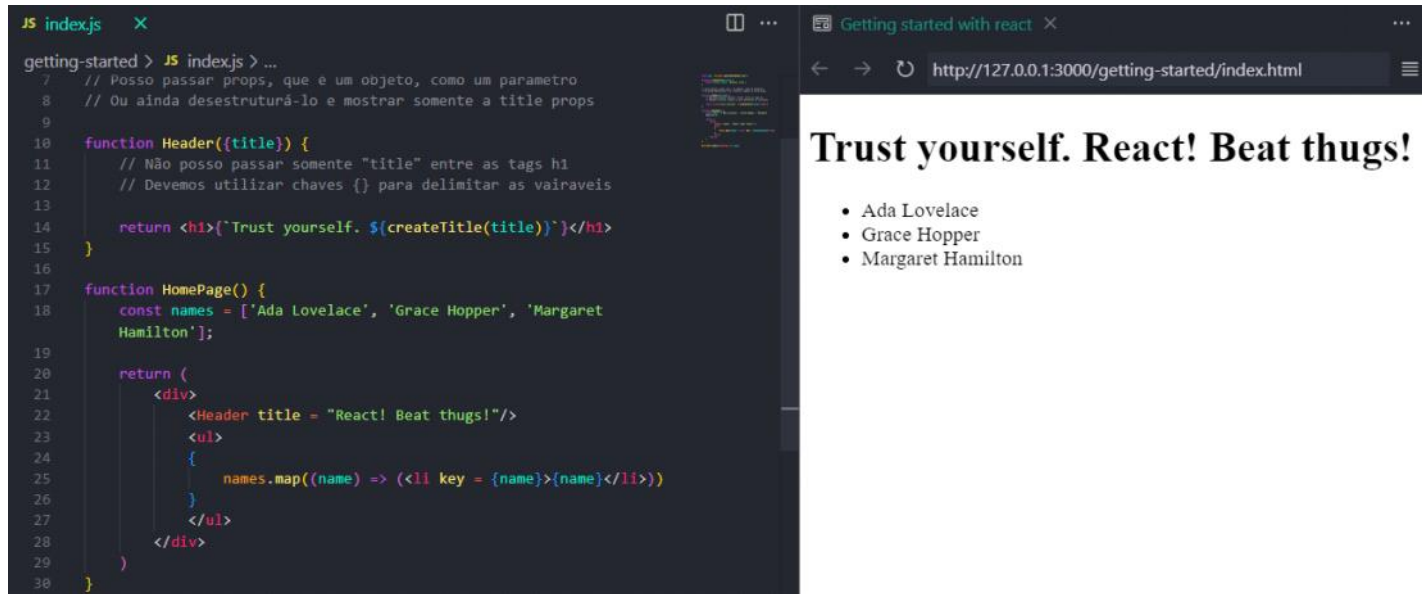
1  const app = document.getElementById("app");
2
3  // Posso passar props, que é um objeto, como um parametro
4  // Ou ainda desestruturá-lo e mostrar somente a title props
5
6  function Header({title}) {
7    // Não posso passar somente "title" entre as tags h1
8    // Devemos utilizar chaves {} para delimitar as variáveis
9
10   return <h1>{title}</h1>
11 }
12
13 function HomePage() {
14   return (
15     <div>
16       <Header title = "React! Beat thugs!" />
17     </div>
18   )
19 }
20
21 ReactDOM.render(<HomePage />, app);
  
```

React! Beat thugs!

Ou melhor: usamos as chaves para inserir JavaScript dentro de JSX. Ou seja, posso escrever **props.title**, se tivesse passado props como parâmetro. Ou ainda uma string formatada, como em `{`Cool ${title}`}`.

```
function Header({title}) {  
  // Não posso passar somente "title" entre as tags h1  
  // Devemos utilizar chaves {} para delimitar as variáveis  
  
  return <h1>{`Trust yourself. ${title}`}</h1>  
}
```

Posso iterar listas usando map em um constructor.



Usamos, neste caso, as chaves para sair do JSX por diversas vezes (eu sei que são só duas). Fora isto, o react iria reclamar se não utilizássemos algum parâmetro key, pois é preciso identificar corretamente os elementos para saber o que atualizar no DOM. Um ID serviria, por exemplo.

## State

- State: podemos entender seu conceito como uma informação que muda com o tempo, geralmente pela ação do usuário.

Vamos a um exemplo: vamos adicionar um botão na nossa HomePage component e adicionar um `onClick = {}` para que ele realize uma função quando a gente, obviamente, clicar nele.

```
function HomePage() {
  const names = ['Ada Lovelace', 'Grace Hopper',
    'Margaret Hamilton'];

  return (
    <div>
      <Header title = "React! Beat thugs!" />
      <p>Nomes insanos: </p>
      <ul>
        {
          names.map((name) => (<li key = {name}>
            {name}</li>))
        }
      </ul>
      <button onClick = {}>Like</button>
      <Paragraph />
    </div>
  )
}
```

Para lidar com os eventos que se sucederão após o clique, podemos criar uma função dentro da própria HomePage, chamada de handleClick().

```
function HomePage() {
  function handleClick() {
    console.log("Increment like count.");
  }

  const names = ['Ada Lovelace', 'Grace Hopper',
    'Margaret Hamilton'];

  return (
    <div>
      <Header title = "React! Beat thugs!" />
      <p>Nomes insanos: </p>
      <ul>
        {
          names.map((name) => (<li key = {name}>
            {name}</li>))
        }
      </ul>
      <button onClick = {handleClick()}>Like</button>
      <Paragraph />
    </div>
  )
}
```

Ao passar a função como argumento para o evento onClick, eu imprimo no console a mensagem "Increment like count" toda vez que o botão like for acionado.

Assim, state pode ser usado para incrementar um valor toda vez que o botão for acionado. Usamos um react hook (gancho react?) para lidar com este state, chamado de React.useState(). Por si só, ele retorna um array. Como JavaScript é uma linguagem insana, podemos - pasmem - desestruturar arrays e acessar seus valores diretamente. Vamos ver como isso fica em código:

```
const [] = React.useState();
```

O primeiro valor deste array é chamado - pasmem de novo - de *value*. Podemos nomeá-lo como bem entendermos, mas é recomendado por um nome coerente com a nossa aplicação para ele. Além disto, podemos adicionar um segundo valor que vai marcar a atualização do nosso valor inicial. **Trata-se de uma função** e é comum usar a palavra set para os nomes dela.

```
const [likes, setLikes] = React.useState();
```

Deste modo, já podemos informar qual valor o nosso state inicial (likes) terá: zero.

```
const [likes, setLikes] = React.useState(0);
```

Poderemos checar a quantidade de likes dentro do nosso componente, inserindo o nosso state com as chaves que vimos anteriormente:

```
<button onClick = {handleClick}>Like ({likes})</button>
```

Por fim, podemos chamar nossa "função de atualização", setLikes para fazer o seu papel e atualizar likes da forma como queremos.

```
function handleClick() {  
  setLikes(likes + 1);  
}
```

(likes++ não funcionou quando testei)

**Note:** Unlike props which are passed to components as the first function parameter, the state is initiated and stored within a component. You can pass the state information to children components as props, but the logic for updating the state should be kept within the component where state was initially created.