

ReactJS

Hooks

1 Introdução

Desde a sua versão 16.8, o React inclui um mecanismo denominado **Hooks**. A primeira frase que a documentação oficial cita a seu respeito é essa.

"Hooks are a new addition in React 16.8. They let you use state and other React features without writing a class."

Ou seja, é um mecanismo voltado a componentes funcionais - aqueles definidos por meio de funções - que viabiliza o uso de recursos outrora disponíveis apenas a componentes definidos por meio de classes.

Aliás, veja a sua documentação aqui.

<https://reactjs.org/docs/hooks-intro.html>

Vale destacar os seguintes pontos da documentação.

- **Completely opt-in.** You can try Hooks in a few components without rewriting any existing code. But you don't have to learn or use Hooks right now if you don't want to.
- **100% backwards-compatible.** Hooks don't contain any breaking changes.
- **There are no plans to remove classes from React.**
- **Classes confuse both people and machines.** In addition to making code reuse and code organization more difficult, we've found that classes can be a large barrier to learning React. You must understand how this works in JavaScript, which is very different from how it works in most languages.

Neste material, estudaremos sobre os principais Hooks do React. Veja a seguir alguns deles. Perceba que o nome deles costuma começar com a palavra **use**. Há razões para isso, claro. Veja uma explicação - da documentação oficial - para o Hook `useState`, por exemplo, que viabiliza o uso de estado em componentes funcionais. Há, outros detalhes sobre os quais estudaremos adiante.

"You might be wondering: why is `useState` not named `createState` instead? "Create" wouldn't be quite accurate because the state is only created the first time our component renders. During the next renders, `useState` gives us the current state. Otherwise, it wouldn't be "state" at all!"

Nome	Razão de ser/ O que faz	Exemplo
<code>useState</code>	Viabiliza o uso de estado por componentes funcionais. Devolve uma variável "de estado" e uma função capaz de alterar o seu valor.	<code>const [variavel, setVariavel] = useState("")</code>
<code>useEffect</code>	Permite causar "efeitos colaterais" - daí seu nome - após a renderização de um componente funcional. Ou seja, executar uma função depois de ele ter sido renderizado e montado na DOM.	<code>useEffect (() => console.log ("Renderizou"))</code>
<code>useContext</code>	Permite acessar o "contexto", um objeto que pode ser utilizado para compartilhar dados entre componentes.	Veremos no futuro próximo.
<code>useReducer</code>	É uma alternativa ao <code>useState</code> . Pode ser interessante utilizá-lo quando há um procedimento "complexo" para atualização do estado.	Veremos no futuro próximo.
<code>useRef</code>	É uma espécie de caixa capaz de armazenar um valor mutável na sua propriedade "current". Similar a uma variável de instância.	Em breve.

Há inúmeros outros Hooks. Também é possível criar nossos próprios Hooks, como veremos.

2 Desenvolvimento

As seções 2.1 e 2.2 mostram duas formas diferentes para a criação da aplicação. Use a primeira caso nunca tenha feito a instalação da PrimeReact e da PrimeFlex. São as bibliotecas de componentes e de utilitários que utilizaremos. Vale a pena fazer uma primeira vez para aprender. Caso já saiba sobre o assunto e já tenha feito anteriormente, use a segunda.

2.1 (Possibilidade 1: Novo projeto e componente principal. Instalação passo a passo da PrimeReact e da PrimeFlex) Crie um projeto com

```
npx create-react-app hooks-react
```

Use

```
cd hooks-react
```

para navegar até o diretório em que se encontra o seu projeto. Use

```
code .
```

para obter uma instância do VS Code vinculada a esse diretório. No VS Code, clique **Terminal >> New Terminal** para obter um novo terminal interno do VS Code, o que simplifica o trabalho. Neste terminal, digite

```
npm start
```

para colocar a aplicação em funcionamento. Uma janela do seu navegador padrão deve ser aberta fazendo uma requisição a **localhost:3000**.

Apague todos os arquivos existentes na pasta **src**. A seguir, crie uma pasta chamada **components**, subpasta de **src**. Definiremos todos os componentes dessa aplicação em arquivos dentro dela.

Na pasta **components**, crie um arquivo chamado **App.js**. Veja seu conteúdo inicial. Optamos por um componente funcional pois não precisaremos de estado inicialmente.

(Dependências) Utilizaremos componentes da biblioteca PrimeReact e os utilitários e grid system da PrimeFlex. As suas respectivas documentações podem ser visitadas a seguir.

PrimeReact

<https://www.primefaces.org/primereact/>

PrimeFlex

<https://www.primefaces.org/primeflex/>

Faça a instalação da PrimeReact com

```
npm install primereact
npm install primeicons
npm install react-transition-group
```

A PrimeFlex pode ser instalada com

```
npm install primeflex
```

A seguir, importe o CSS da PrimeReact, PrimeIcons, PrimeFlex e um dos temas descritos na documentação, na página **Get Started**. Isso pode ser feito no arquivo **index.js**. Veja.

```
import React from 'react'
import ReactDOM from 'react-dom'
import App from './components/App'
import 'primereact/resources/primereact.min.css'
import 'primeicons/primeicons.css'
import 'primeflex/primeflex.css'
```

```
import 'primereact/resources/themes/bootstrap4-light-purple/theme.css'  
ReactDOM.render(  
  <App />,  
  document.querySelector('#root')  
)
```

2.2 (Possibilidade 2: Obtendo um modelo inicial que já contém as dependências instaladas) Caso não tenha realizado os passos da seção 2.1, você pode obter uma cópia do projeto com as dependências configuradas. Há um modelo disponível aqui.

https://github.com/professorbossini/pessoal_react_modelo_primereact_primeflex

Para obter uma cópia para você, execute o seguinte comando usando um terminal vinculado ao diretório que representa o seu workspace (fora de qualquer outro projeto React!)

git clone

https://github.com/professorbossini/pessoal_react_modelo_primereact_primeflex
hooks-react

O projeto obtido conterá um arquivo chamado **package.json** que descreve as dependências. Para fazer o seu download, use o seguinte comando.

npm install

Neste momento já deve ser possível colocar o projeto em funcionamento com o seguinte comando.

npm start

Deve ser possível acessar a aplicação no endereço a seguir a partir de agora.

localhost:3000

A seguir, pode ser interessante alterar a URL associada ao remote **origin** utilizando uma URL de um repositório próprio seu. Assim você poderá fazer seu próprio controle de versão. Para tal, depois de criar um repositório no Github, use

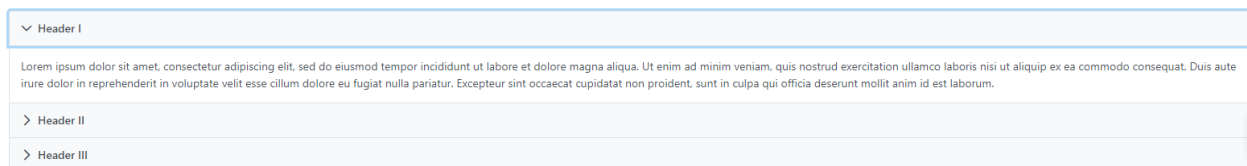
```
git remote set-url origin url-do-seu-repositorio-no-github
```

A partir de agora, as operações push envolvendo o remote origin serão direcionadas ao seu repositório remoto. Aliás, você já pode executar o seguinte comando para ter certeza de que tudo está configurado corretamente.

```
git push -u origin master
```

2.3 (A aplicação) A aplicação que desenvolveremos é composta por diversos componentes que definiremos. O objetivo é ilustrar o uso dos diferentes Hooks que o React disponibiliza.

2.4 (Primeiro componente: Um Accordion) O primeiro componente que desenvolveremos é um Accordion. Veja.



Esse é um Accordion próprio da PrimeReact. Veja mais alguns exemplos na sua documentação.

<https://primefaces.org/primereact/showcase/#/accordion>

Muito embora esse componente esteja pronto para uso, optaremos por implementar o nosso próprio. Essa implementação será um prato cheio para o estudo dos Hooks React. Assim, clique com o direito na pasta **components** e crie um arquivo chamado **Accordion.js**. Veja seu conteúdo inicial. Repare que ele é um componente funcional.

```
import React from 'react'

const Accordion = () => {
  return (
    <div>
      Accordion
    </div>
  )
}

export default Accordion
```

A seguir, passamos a utilizá-lo no componente **App**. Veja.

```
import React from 'react'
import Accordion from './Accordion'

const App = () => {
  return (
    <div>
      <Accordion />
    </div>
  )
}

export default App
```

2.5 (A lista de itens a serem exibidos) Caberá ao componente **App** armazenar a lista de itens a serem exibidos. Ele a entregará ao **Accordion** via props, como de costume. Veja.

```
import React from 'react'
import Accordion from './Accordion'

const itens = [
  {
    titulo: "Java",
    conteudo: "Linguagem compilada e interpretada."
  },
  {
    titulo: "Python",
    conteudo: "Linguagem interpretada e dinamicamente tipada."
  },
  {
    titulo: "Javascript",
    conteu-
do: "Interpretada. Executa do lado do cliente e do lado do servidor também."
  }
]

const App = () => {
  return (
    <div>
      <Accordion itens={itens} />
    </div>
  )
}

export default App
```

O Accordion recebe a lista via props e, por enquanto exibe o conteúdo de seus elementos de maneira simples. Veja.


```
import React from 'react'

const Accordion = ( { itens } ) => {
  return (
    <div>
      {
        itens.map((item, indice) => (
          <div key={indice}>
            <h4 >{item.titulo}</h4>
            <p>{item.conteudo}</p>
          </div>
        ))
      }
    </div>
  )
}

export default Accordion
```

A expectativa é que a aplicação esteja exibindo algo parecido com isso aqui.

Java

Linguagem compilada e interpretada.

Python

Linguagem interpretada e dinamicamente tipada.

Javascript

Interpretada. Executa do lado do cliente e do lado do servidor também.

Nota. Usar um índice como key não é recomendável. Veja o que a documentação fala sobre isso.

"We don't recommend using indexes for keys if the order of items may change. This can negatively impact performance and may cause issues with component state. "

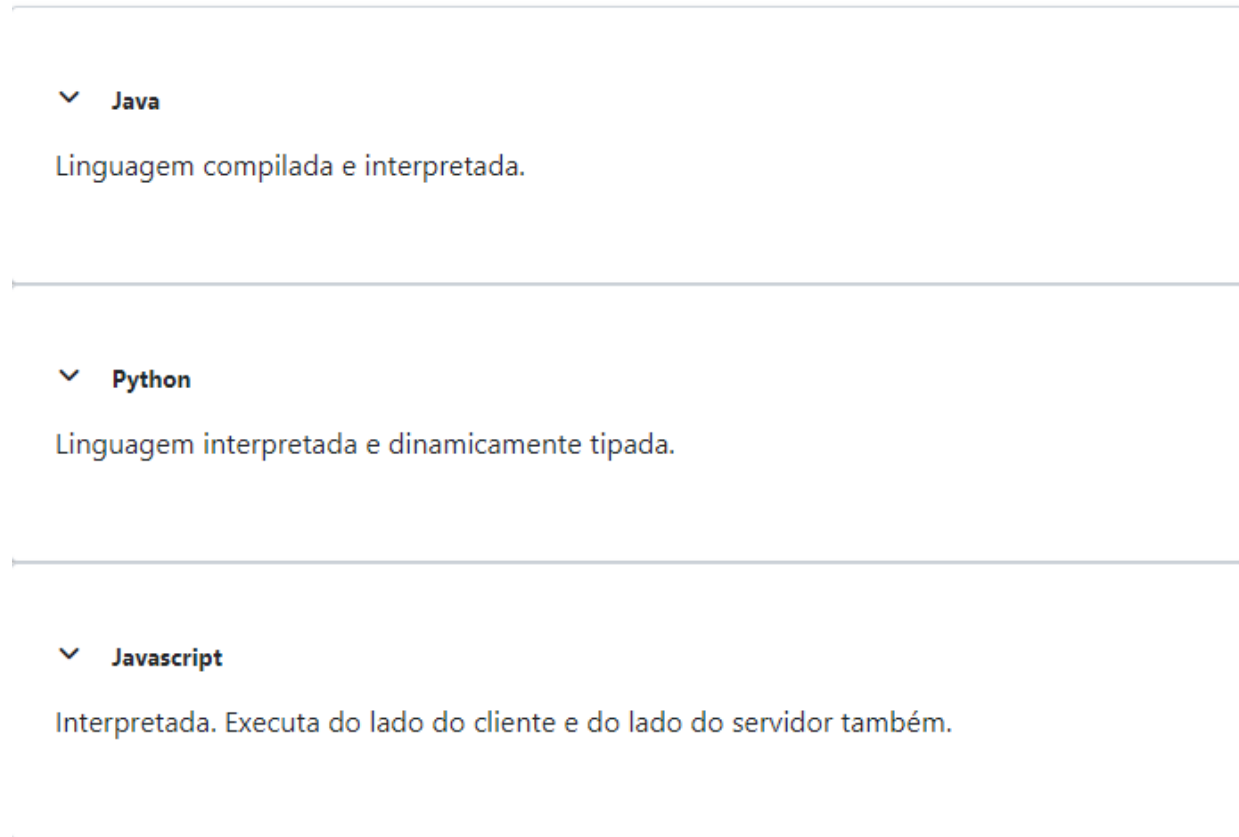
Veja mais aqui.

<https://reactjs.org/docs/lists-and-keys.html>

2.6 (Itens do Accordion serão Cards) Vamos utilizar o componente Card da PrimeReact para representar os itens do Accordion. Veja a seguir. Repare como cada Card possui um título composto por um ícone e pelo título do item. Seu corpo exibe o conteúdo do item.

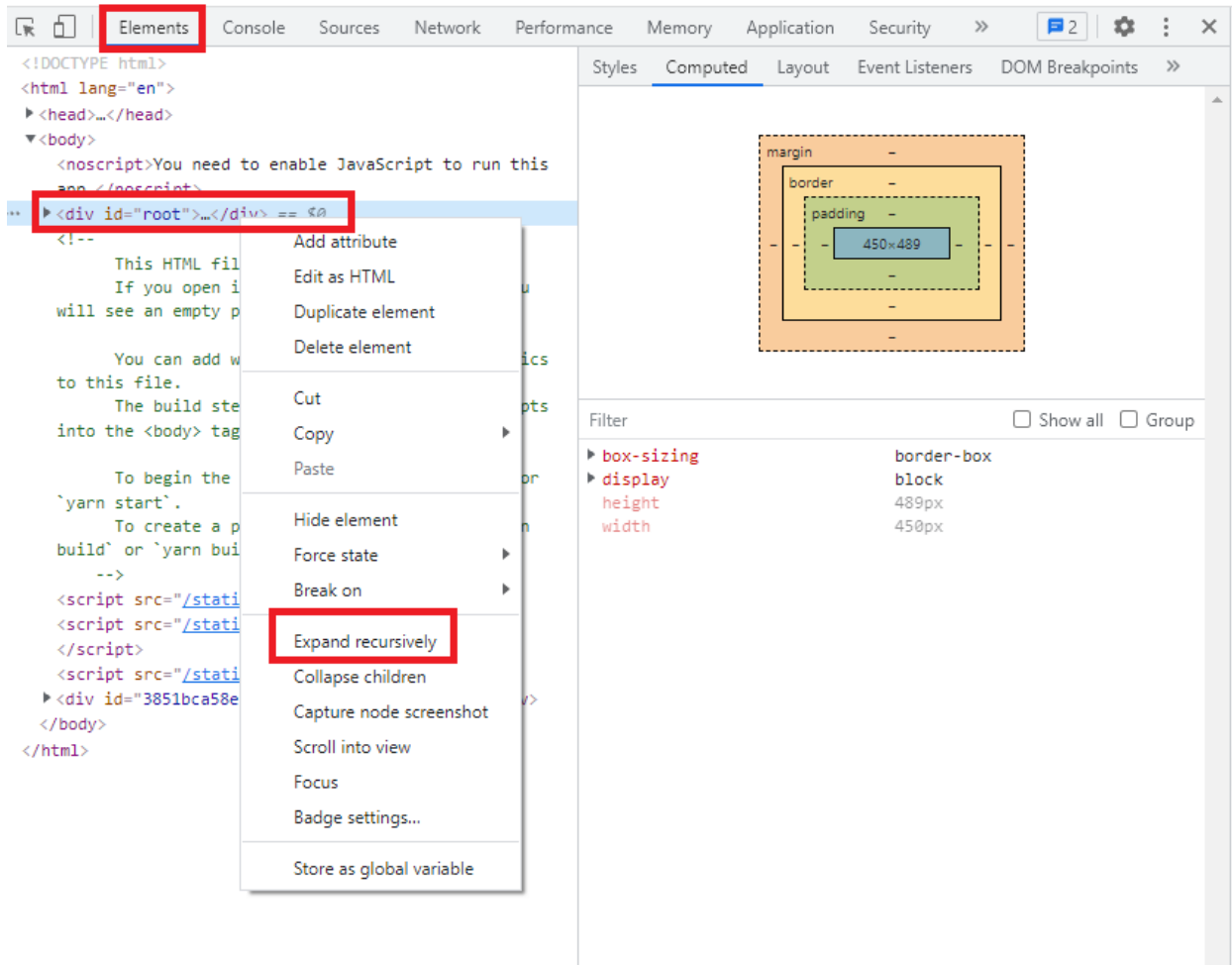
```
import React from 'react'
import { Card } from 'primereact/card'
const Accordion = ( { itens } ) => {
  const expressaoJSX = itens.map((item, indice) => {
    return (
      <Card key={indice} className="border-1 border-400">
        <div>
          <i className="pi pi-angle-down"></i>
          <h5 className="inline ml-3">{item.titulo}</h5>
        </div>
        <p>{item.conteudo}</p>
      </Card>
    )
  })
  return (
    <div>
      {
        expressaoJSX
      }
    </div>
  )
}
export default Accordion
```

Veja o resultado esperado agora.



2.7 (Investigando as medidas de espaçamento de um Card da PrimeReact)

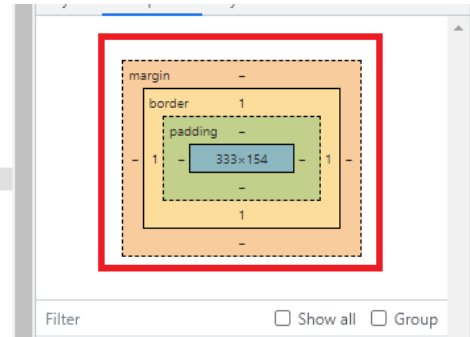
Repare que os cartões têm, por padrão, alguma medida que separa o conteúdo da borda. Pode ser algo como padding ou margin. Para descobrir de onde vêm esses valores, inspecionamos os elementos no Chrome Dev Tools, na aba elements. Para ver todos eles, clique com o direito na div cujo id é root e escolha **Expand Recursively**. Veja.



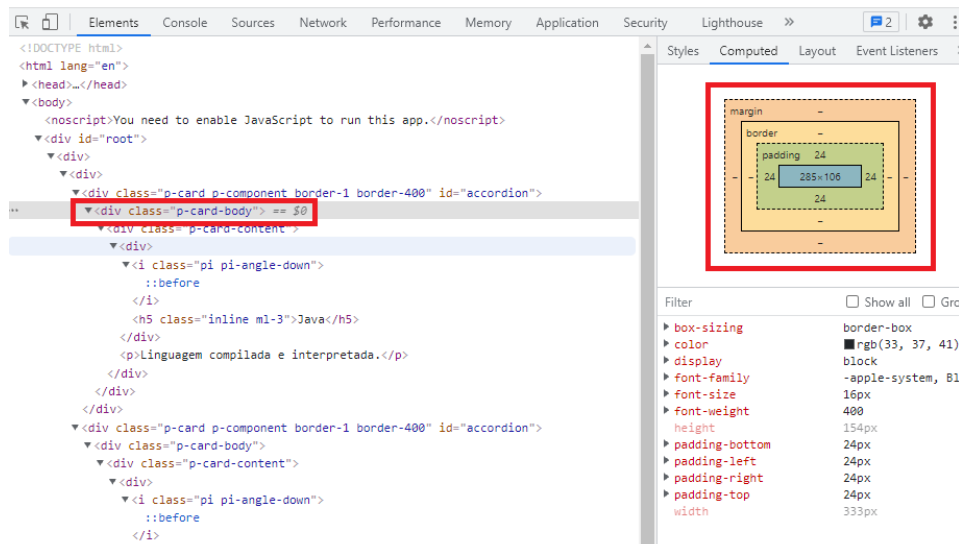
A seguir, inspecione a estrutura do primeiro cartão. Ele é uma simples div sob o efeito de algumas classes CSS. Uma delas se chama **p-card**. Ela poderia ser responsável por esse espaçamento. Talvez seja seu padding. Mas passe o mouse sobre ela e observe o quadro na aba **computed**, como a seguir.

```
<html lang="en">
<head>...</head>
<body>
  <noscript>You need to enable JavaScript to run this app.</noscript>
  <div id="root">
    <div>
      <div>
        <div class="p-card p-component border-1 border-400" id="accordion"> == $0
          <div class="p-card-body">
            <div class="p-card-content">
              <div>
                <i class="pi pi-angle-down">
                  ::before
                </i>
              </div>
            </div>
          </div>
        </div>
      </div>
    </div>
  </body>
</html>
```

passa o mouse aqui



Repare que os valores de margin e padding na caixa são iguais a um símbolo "-". Ou seja, esse componente não possui margin e nem padding. Ele tem uma borda de um pixel de espessura. Obviamente não é a borda que está causando esse espaçamento. Vasculhe um pouco mais e encontre uma outra div sob o efeito de uma classe CSS chamada **p-body**. Faça o mesmo procedimento passando o mouse sobre ela para obter o seguinte resultado.



Perceba que essa div possui padding de 24 pixels horizontal e verticalmente. Podemos, portanto, escrever um seletor CSS utilizando essa classe e fazer as alterações desejadas. Clique com o direito na pasta **components** e crie um arquivo chamado **Accordion.css**. Veja seu conteúdo. Utilizamos **important** indicando que todos os estilos previamente definidos devem ser sobrepostos.

```
.p-card-body{  
  padding: 8px !important  
}
```

Para que esse seletor tenha efeito, faça o seguinte import no arquivo **Accordion.js**.

```
import React from 'react'  
import Accordion from './Accordion'  
import './Accordion.css'  
const itens = [  
  ...
```

Observe o resultado agora. Compare com o anterior.

▼ **Java**

Linguagem compilada e interpretada.

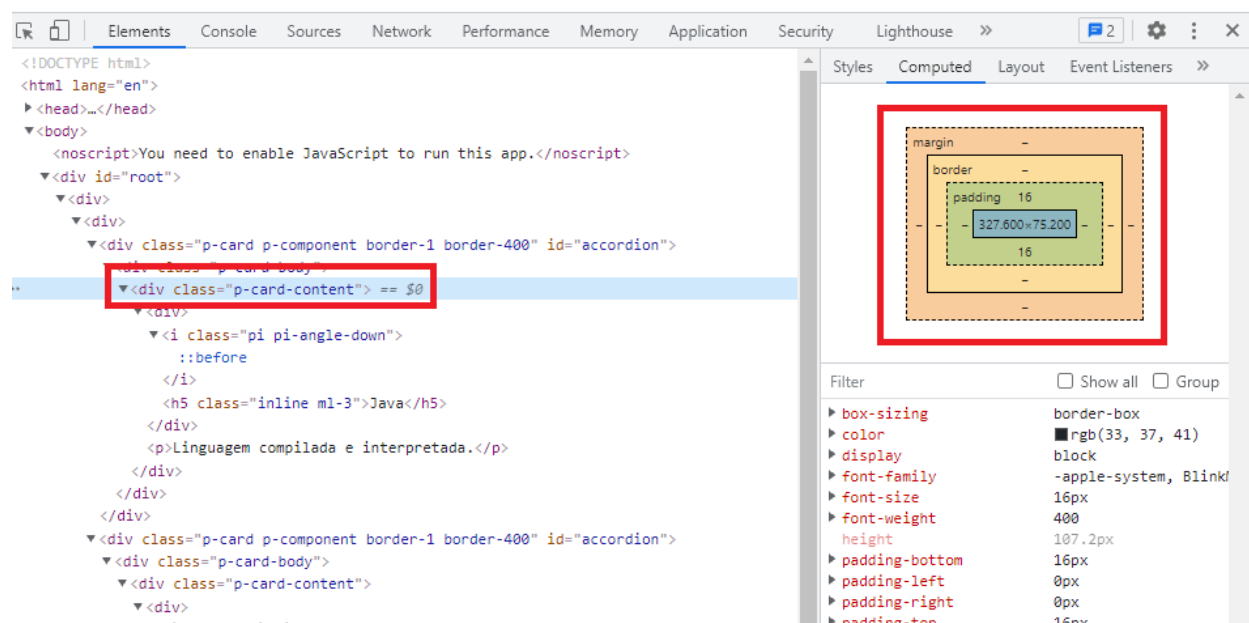
▼ **Python**

Linguagem interpretada e dinamicamente tipada.

▼ **Javascript**

Interpretada. Executa do lado do cliente e do lado do servidor também.

Embora tenhamos ajustado essa medida de padding, repare que há alguma definição de padding ou margin que separa o conteúdo da borda, especialmente na parte de cima e na parte de baixo. Inspeccionando um pouco mais, encontramos uma div sob o efeito de uma classe CSS chamada `.p-card-content`. Veja que ela é responsável pela medida que estávamos procurando.



Adicione este conteúdo ao arquivo **Accordion.css** e veja o resultado.

```
.p-card-body{  
  padding: 8px !important  
}  
  
.p-card-content {  
  padding-top: 4px !important;  
  padding-bottom: 4px !important;  
}
```

2.8 (id para o Card que define o Accordion e seu uso nos seletores CSS)

Mesmo que importássemos o arquivo `Accordion.css` direto no arquivo `Accordion.js`, tudo aquilo que ele define teria impacto em toda a aplicação. Assim, podemos definir um id para o cartão que estamos utilizando para garantir que essas regras se aplicam somente a seus descendentes.

No arquivo **Accordion.js**, fazemos o seguinte ajuste.

```
import React from 'react'  
import { Card } from 'primereact/card'  
import './Accordion.css'  
const Accordion = ( { itens } ) => {  
  
  const expressaoJSX = itens.map((item, indice) => {  
    return (  
      <Card id="accordion" key={indice} className="border-1 border-400">  
        <div>
```


O arquivo **Accordion.css** fica assim.

```
/* O espaço entre #accordion e .p-card-body é fundamental.  
Dessa forma, estamos selecionando todos os elementos  
que estejam sob o efeito da classe p-card-  
bo-  
dy e que sejam descendentes (filhos diretos ou indiretos) de um elemento que tenha  
a id igual a accordion.  
Se o espaço for removido, selecionaremos elementos que estejam sob o efeito da classe  
CSS p-card-boy e que tenham o id igual a accordion.  
*/  
#accordion .p-card-body{  
    padding: 8px !important;  
}  
  
#accordion .p-card-content {  
    padding-top: 4px !important;  
    padding-bottom: 4px !important;  
}
```

2.9 (Tratando o evento click no título de cada item) Desejamos contrair/expandir um item da lista conforme seu item seja clicado. Podemos associar uma função à propriedade **onClick** à div que define o título. O índice do mapa usando para construir os itens visuais nos permite diferenciar um item dos demais, sabendo, portanto, qual deles foi clicado. Veja. Depois desse ajuste, abra o console do seu navegador e clique em alguns itens. Lembre-se de clicar exatamente em seu título, na parte superior.

```
...  
const Accordion = ( { itens } ) => {  
  const itemClicado = (indice) => {  
    console.log(indice)  
  }  
  
  const expressaoJSX = itens.map((item, indice) => {  
    return (  
      <Card id="accordion" key={indice} className="border-1 border-400">  
        <div onClick={() => itemClicado(indice)}>  
          <i className="pi pi-angle-down"></i>  
        </div>  
      </Card>  
    )  
  })  
}
```

2.10 (Hook useState: armazenando o índice do componente clicado no estado do componente) O hook useState tem como finalidade permitir que componentes funcionais tenham estado. Ele tem as seguintes características:

- Deve ser chamado "dentro" da função que define um componente.
- Devolve uma variável e uma função capaz de alterar o seu valor.
- Recebe um valor opcional como parâmetro que será considerado o valor inicial da variável.
- Quando necessário, a variável deve ser atualizada utilizando a função devolvida pelo Hook.
- Uma vez que a variável tenha sido atualizada, um novo ciclo de renderização é executado.

(Desestruturação de arrays em Javascript) Antes de olhar para o uso do `useState`, vale à pena entender como funciona a operação de desestruturação de arrays do Javascript. É muito simples. Veja este exemplo. Você pode digitá-lo no próprio console do navegador.

```
const frutas = ['banana', 'maçã']  
//como acessar cada uma?  
console.log(frutas[0])  
console.log(frutas[1])  
//desestruturando, declaramos uma variável que irá referenciar cada objeto  
const [f1, f2] = frutas  
console.log(f1)  
console.log(f2)
```

A seguir, armazenamos o índice do item clicado em uma variável de estado e exibimos o seu valor para um primeiro teste.

```
import React, { useState } from 'react'
...
const Accordion = ( { itens } ) => {

  const [indiceAtivo, setIndiceAtivo] = useState (null)
  //OBS: Sem desestruturação ficaria parecido com isso:
  /*
  const estado = useState(null)
  const indiceAtivo = estado[0]
  const setIndiceAtivo = estado[1]

  */

  const itemClicado = (indice) => {
    setIndiceAtivo(indice)
  }

  const expressaoJSX = itens.map((item, indice) => {
    return (
      <Card id="accordion" key={indice} className="border-1 border-400">

        <div onClick={() => itemClicado(indice)}>
          <i className="pi pi-angle-down"></i>
          <h5 className="inline ml-3">{item.titulo}</h5>
        </div>
        <p>{item.conteudo}</p>
      </Card>
    )
  })
  return (
    <div>
      <p>{indiceAtivo}</p>
      {
        expressaoJSX
      }
    </div>
  )
}
```

Veja a equivalência entre operações envolvendo estado feitas por componentes funcionais e por aqueles definidos por meio de classes.

Operação	Componentes definidos por meio de classes	Componentes funcionais
Inicializar o estado	<code>state = { valor: 0 }</code>	<code>const [valor, setValor] = useState(0)</code>
Acessar uma variável de estado	<code>this.state.valor</code>	<code>valor</code>
Atualizar uma variável de estado	<code>this.setState({ valor: 1 })</code>	<code>setValor(1)</code>

Compare as mesmas operações quando o estado envolve mais de uma variável.

Operação	Componentes definidos por meio de classes	Componentes funcionais
Inicializar o estado	<code>state = { valor: 0, nome: "" }</code>	<code>const [valor, setValor] = useState(0) const [nome, setNome] = useState("")</code>
Acessar uma variável de estado	<code>this.state.valor this.state.nome</code>	<code>valor nome</code>
Atualizar uma variável de estado	<code>this.setState({ valor: 1, nome: 'abc' })</code>	<code>setValor(1) setNome('abc')</code>

2.11 (Expandindo e contraindo os itens do Accordion) A fim de contrair e expandir os itens do Accordion de acordo com os cliques do usuário, faremos o seguinte:

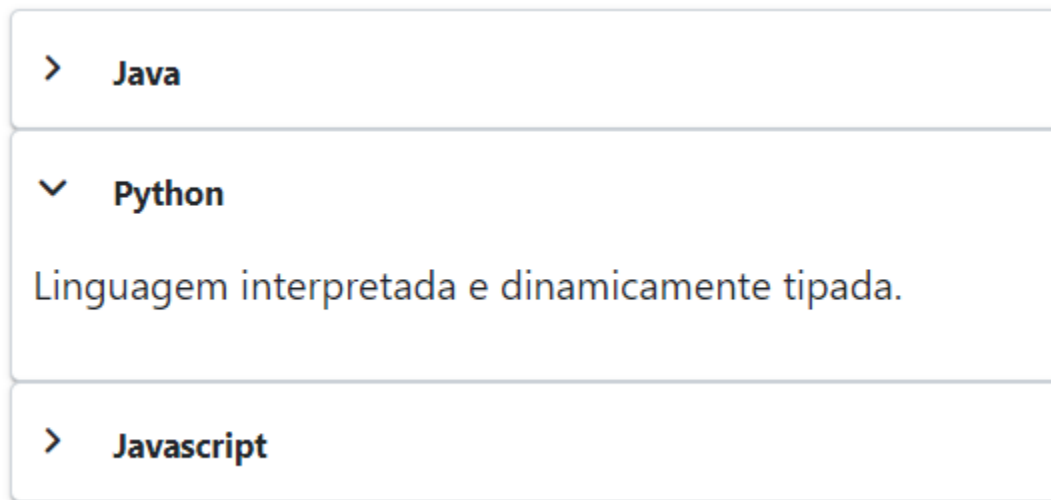
1. A cada renderização, comparar cada índice dos itens existentes no Accordion com o índice atual, armazenado no estado do componente.
2. O item do Accordion que tiver índice diferente do índice atual, terá a classe **hidden** aplicada a seu parágrafo que exibe seu conteúdo. Além disso, a classe que determina seu ícone será **pi-angle-right**.
3. O item do Accordion que tiver índice igual ao índice atual, não terá classe alguma aplicada a seu parágrafo que exibe seu conteúdo, o que quer dizer que ele será exibido normalmente. Além disso, a classe que determina seu ícone será **pi-angle-down**.

Veja.

```
import React, { useState } from 'react'
import { Card } from 'primereact/card'
import './Accordion.css'
const Accordion = ( { itens } ) => {
  const [indiceAtivo, setIndiceAtivo] = useState (null)
  //OBS: Sem desestruturação ficaria parecido com isso:
  /*
  const estado = useState(null)
  const indiceAtivo = estado[0]
  const setIndiceAtivo = estado[1]
  */
  const itemClicado = (indice) => {
    setIndiceAtivo(indice)
  }
  const expressaoJSX = itens.map((item, indice) => {
    const classExibirConteudo = indice === indiceAtivo ? '' : 'hidden'
    const classExibirIcone = indice === indiceAtivo ? 'pi-angle-down' : 'pi-angle-right'
    return (
      <Card id="accordion" key={indice} className="border-1 border-400">

        <div onClick={() => itemClicado(indice)}>
          <i className={`pi ${classExibirIcone}`}></i>
          <h5 className="inline ml-3">{item.titulo}</h5>
        </div>
        <p className={classExibirConteudo}>{item.conteudo}</p>
      </Card>
    )
  })
  return (
    <div>
      {
        expressaoJSX
      }
    </div>
  )
}
export default Accordion
```

Faça um novo teste, clicando nos títulos dos itens no Accordion. O resultado esperado é parecido com esse aqui.



Ao clicar em um item, ele expande e os demais contraem. Se for clicado de novo, tudo permanece como está.

2.12 (Segundo componente: Um componente de “busca”) O segundo componente que utilizaremos permitirá realizar “buscas”. Ele possui um campo em que um termo de busca é informado e depois exibe uma lista composta por itens que tenham relação com o termo de busca. A busca será realizada diretamente na base da **Wikipedia**. Ela pode ser acessada por meio do seguinte link.

en.wikipedia.org/w/api.php?action=query&list=search&format=json&srsearch=

Ao final, concatenamos o termo de busca. Tente acessar o seguinte link direto no seu navegador para fazer um teste. Neste exemplo, o termo de busca é “java”.

<https://en.wikipedia.org/w/api.php?action=query&list=search&format=json&srsearch=java>

O resultado esperado é parecido com esse aqui.

```
{
  "batchcomplete": "",
  "continue": {
    "sroffset": 10,
    "continue": "-||"
  },
  "query": {
    "searchinfo": {
      "totalhits": 34496,
      "suggestion": "jazz",
      "suggestionsnippet": "jazz"
    },
    "search": [
      {
        "ns": 0,
        "title": "Java",
        "pageid": 69336,
        "size": 58360,
        "wordcount": 6010,
        "snippet": "<span class=\"searchmatch\">Java</span> (Indonesian: Jawa, Indonesian pronunciation: [ˈdʒawa]; Javanese: ꦗꦮ; Sundanese: ᮊᮘ) is one of the Greater Sunda Islands in Indonesia. It is bordered",
        "timestamp": "2021-10-01T01:11:29Z"
      },
      {
        "ns": 0,
        "title": "Java (disambiguation)",
        "pageid": 15628,
        "size": 2692,
        "wordcount": 364,
        "snippet": "Oracle <span class=\"searchmatch\">Java</span> virtual machine, an abstract computing machine enabling a computer to run a <span class=\"searchmatch\">Java</span> program"
      }
    ]
  }
}
```

Nota. Para que o Chrome exiba o objeto JSON formatado assim, visite a Chrome Web Store e adicione a extensão JSON Viewer. Veja seu link aqui.

<https://chrome.google.com/webstore/detail/json-viewer/gbmdgpbipfallnflgajpaliibnhdgobh>

2.13 (Criação do componente de Busca) Começamos criando um componente chamado Busca. Clique com o direito na pasta **components** e crie um arquivo chamado **Busca.js**. Veja seu código inicial.

```
import React from 'react'

const Busca = () => {
  return (
    <div>

    </div>
  )
}

export default Busca
```

A seguir, o componente **App** passa a fazer uso do componente **Busca**. Para tal, deixaremos de utilizar o componente **Accordion**. Guardaremos em uma variável a expressão JSX do componente que desejamos usar. Veja.

```
import React from 'react'
import Accordion from './Accordion'
import Busca from './Busca'
import './Accordion.css'
const itens = [
  {
    titulo: "Java",
    conteudo: "Linguagem compilada e interpretada."
  },
  {
    titulo: "Python",
    conteudo: "Linguagem interpretada e dinamicamente tipada."
  },
  {
    titulo: "Javascript",
    conteu-
do: "Interpretada. Executa do lado do cliente e do lado do servidor também
."
  }
]
const App = () => {
  const expressaoJSX = <Busca />
  return (
    <div>
      {expressaoJSX}
    </div>
  )
}

export default App
```

O componente **Busca** utiliza um **InputText** da PrimeReact para permitir que o usuário digite seu termo de busca. Veja.

```
import React from 'react'
import { InputText } from 'primereact/inputtext'
const Busca = () => {
  return (
    <div>
      <span className="p-input-icon-left">
        <i className="pi pi-search"></i>
        <InputText />
      </span>
    </div>
  )
}

export default Busca
```

2.14 (Tratando o termo de busca com o hook useState) A seguir, utilizamos uma variável de estado para armazenar o que o usuário digita. Ela é atualizada a cada interação do usuário com o campo de texto. Veja.

```
import React, { useState } from 'react'
import { InputText } from 'primereact/inputtext'
const Busca = () => {
  const [termoDeBusca, setTermoDeBusca] = useState('')
  return (
    <div>
      <span className="p-input-icon-left">
        <i className="pi pi-search"></i>
        <InputText
          onChange={(e) => setTermoDeBusca(e.target.value)}
        />
      </span>
    </div>
  )
}

export default Busca
```

2.15 (Hook useEffect: Realizando as buscas) O hook useEffect permite que especifiquemos uma função a ser executada em três ocasiões diferentes.

I. Apenas quando um componente é renderizado pela primeira vez.

II. Quando um componente é renderizado pela primeira vez e em todas as outras vezes em que ele for renderizado de novo.

III. Quando o componente é renderizado pela primeira vez e todas as outras vezes em que ele for renderizado de novo, desde que algum valor de seu conhecimento tenha sido alterado desde a última renderização.

Perceba a semelhança do que podemos fazer com o hook useEffect com o que fazemos com os ciclos de vida de um componente React definido por meio de uma classe.

A seguir, veja exemplos das três formas.

Exemplo de uso	Quando executada	Significado de seus parâmetros
<code>useEffect(() => { console.log('oi') })</code>	Sempre que o componente renderiza, incluindo a primeira vez.	O primeiro parâmetro é simplesmente a função a ser executada.
<code>useEffect(() => { console.log('oi') }, [])</code>	Somente quando o componente renderiza pela primeira vez.	O primeiro parâmetro é a função a ser executada. O segundo parâmetro é uma lista de expressões com o seguinte significado: A função especificada somente será executada quando pelo menos um dos valores existentes na lista tiver sido alterado desde a última renderização. Como a lista é vazia, jamais haverá um item com essa característica e, assim, a função executa somente na primeira vez que o componente renderiza.
<code>useEffect(() => { console.log('oi') }, [termoDeBusca])</code>	Da primeira vez que o componente renderiza e em todas as outras vezes que ele for renderizado, desde que termoDeBusca tenha sofrido alguma alteração.	O primeiro parâmetro é a função a ser executada. O segundo parâmetro indica a lista de dependências para que a função seja executada.

A seguir, registramos três funções que devem causar "efeitos colaterais" ao componente depois de ele renderizar. Depois de salvar o arquivo, atualize o navegador e digite algumas letras no campo textual. Como o termoDeBusca é alterado sempre, duas funções sempre executarão. Por outro lado, uma delas executa somente uma vez.

```
import React, { useState, useEffect } from 'react'
import { InputText } from 'primereact/inputtext'
const Busca = () => {
  const [termoDeBusca, setTermoDeBusca] = useState('')
  useEffect (() => {
    console.log ("executando todas as vezes...")
  })
  useEffect(() => {
    console.log("executando somente uma vez")
  }, [])
  useEffect(() => {
    console.log("Executando somente quando o termo é alterado")
  }, [termoDeBusca])
  return (
    <div>
      <span className="p-input-icon-left">
        <i className="pi pi-search"></i>
        <InputText
          onChange={(e) => setTermoDeBusca(e.target.value)}
        />
      </span>
    </div>
  )
}

export default Busca
```

Faremos uma requisição a cada vez que o usuário alterar o termo de busca. Ele é, portanto, uma condição que deve fazer parte do segundo argumento de **useEffect**. As requisições serão feitas com o **axios**. Faça a sua instalação assim.

npm install axios

Ajuste o componente Busca assim.

```
import React, { useState, useEffect } from 'react'
import axios from 'axios'
import { InputText } from 'primereact/inputtext'
const Busca = () => {
  const [termoDeBusca, setTermoDeBusca] = useState('')
  // mantenha somente um useEffect
  useEffect(() => {
    // 
  }, [termoDeBusca])
  return (
    <div>
      <span className="p-input-icon-left">
        <i className="pi pi-search"></i>
        <InputText
          onChange={(e) => setTermoDeBusca(e.target.value)}
        />
      </span>
    </div>
  )
}

export default Busca
```

Lembre-se que as requisições feitas pela axios operam de maneira assíncrona. Quando uma requisição é feita, obtemos uma Promise que nos permite acessar o resultado esperado em algum momento no futuro, quando a computação associada a ela for concluída.

Caso tenhamos interesse em utilizar a construção `async/await`, podemos fazê-lo. Entretanto, a função que entregamos ao Hook `useEffect` não pode ser marcada com `async`. Tente fazer isso desta forma.

```
const Busca = () => {
  const [termoDeBusca, setTermoDeBusca] = useState('')
  // mantenha somente um useEffect
  // não pode
  useEffect(async () => {
    const res = await axios.get('https://reactjs.org/')
  }, [termoDeBusca])
  return (
    <div>
      <span className="p-input-icon-left">
        <i className="pi pi-search"></i>
        <InputText
          onChange={(e) => setTermoDeBusca(e.target.value)}
        />
      </span>
    </div>
  )
}
```

O console do navegador deve exibir uma mensagem de erro parecida com essa aqui.

```
src\components\Busca.js
Line 7:15: Effect callbacks are synchronous to prevent race conditions. Put the async
function inside:

useEffect(() => {
  async function fetchData() {
    // You can await here
    const response = await MyAPI.getData(someId);
    // ...
  }
  fetchData();
}, [someId]); // Or [] if effect doesn't need props or state
```

Ele diz que as funções que causam "efeitos colaterais", ou seja, aquelas que passamos como parâmetro para o hook `useEffect` operam de maneira síncrona por natureza, a fim de evitar condições de corrida. Observe, também, que a própria mensagem de erro dá uma sugestão de correção: definir uma nova função e utilizar a construção `async/await` envolvendo esta função. Veja.

```
const Busca = () => {
  const [termoDeBusca, setTermoDeBusca] = useState('')
  // mantenha somente um useEffect
  useEffect(() => {
    //definimos a função
    const aux = async () => {
      const res = await axios.get('https://reactjs.org/')
      console.log(res)
    }
    // e chamamos a seguir
    aux()
  }, [termoDeBusca])
  return (
    <div>
      <span className="p-input-icon-left">
        <i className="pi pi-search"></i>
        <InputText
          onChange={(e) => setTermoDeBusca(e.target.value)}
        />
      </span>
    </div>
  )
}
```

Se preferir manipular suas promises diretamente, sem a construção `async/await`, ou seja, utilizando as funções **then** e **catch**, isso também é possível. Veja.

```
useEffect(() => {  
    //definimos a função  
    const aux = () => {  
        axios.get('https://reactjs.org/')  
            .then((res) => console.log(res))  
    }  
    // e chamamos a seguir  
    aux()  
}, [termoDeBusca])
```

Opte pela que for mais conveniente para você. A seguir, faremos o tratamento de promises usando a construção `async/await`.

Veja mais uma vez o link que utilizaremos para realizar buscas junto à Wikipedia.

<https://en.wikipedia.org/w/api.php?action=query&list=search&format=json&srsearch=java>

Podemos dizer que ele tem

- uma "Base URL", que é <https://en.wikipedia.org/w/api.php>
- Os quatro parâmetros **action**, **list**, **format** e **srsearch** com os valores **query**, **search**, **json** e **java** associados, respectivamente.

Assim, a requisição GET a ser feita utilizando o **axios** é a seguinte. Repare que limpamos a função utilizada no hook `useEffect`, removendo os testes anteriores.

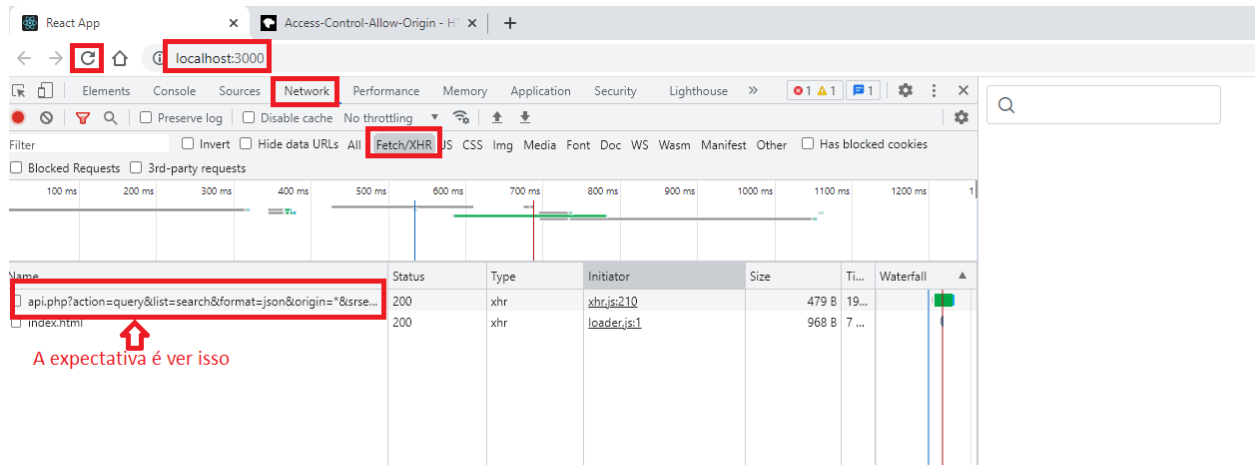
Nota: Adicionamos o parâmetro **origin** à requisição. Leia mais sobre ele aqui.

<https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Access-Control-Allow-Origin>

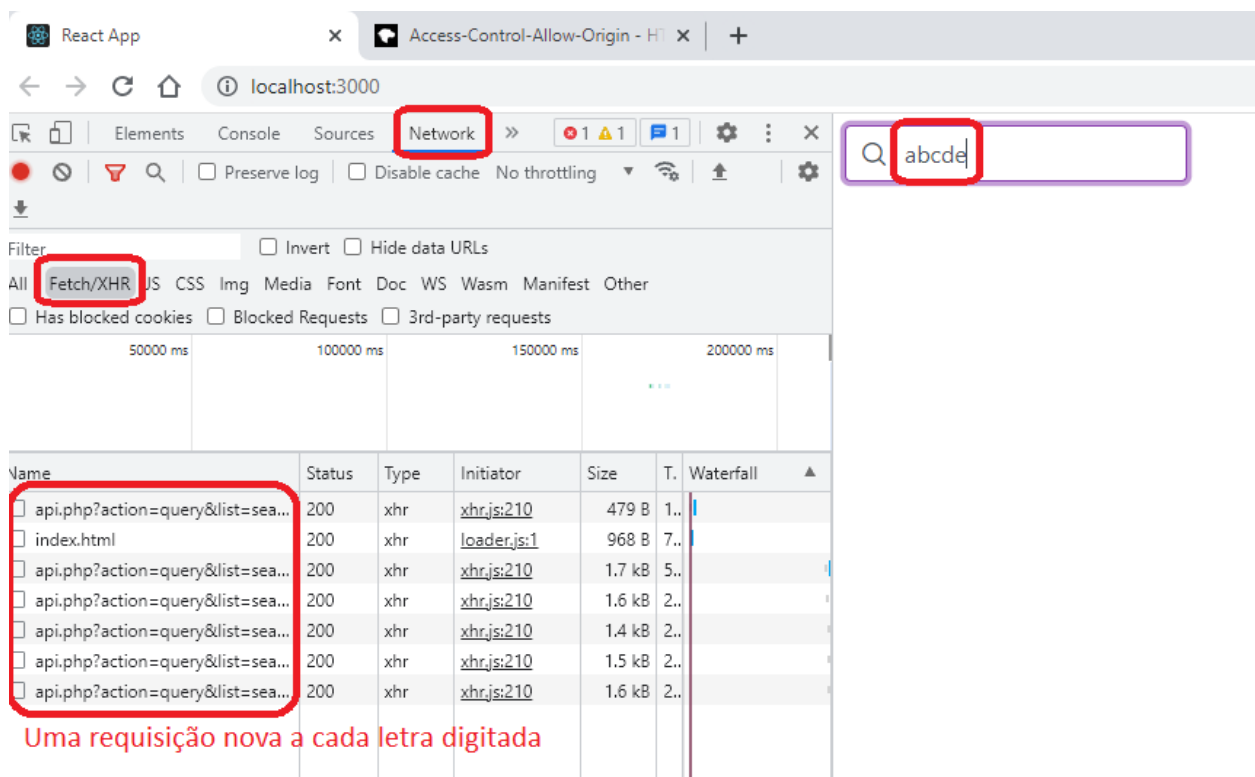
Veja esse trecho, em particular: *For requests without credentials, the literal value "*" can be specified as a wildcard; the value tells browsers to allow requesting code from any origin to access the resource. Attempting to use the wildcard with credentials results in an error.*

```
const Busca = () => {
  const [termoDeBusca, setTermoDeBusca] = useState('')
  useEffect(() => {
    //define a função
    const fazBusca = async () => {
      await axios.get(
        'https://en.wikipedia.org/w/api.php',{
          params: {
            action: 'query',
            list: 'search',
            format: 'json',
            // instruindo o navegador a permitir
            // conteúdo de qualquer origem
            origin: '*',
            srsearch: termoDeBusca
          }
        })
    }
    //chama a função
    fazBusca()
  }, [termoDeBusca])
  return (
    <div>
      <span className="p-input-icon-left">
        <i className="pi pi-search"></i>
        <InputText
          onChange={(e) => setTermoDeBusca(e.target.value)}
        />
      </span>
    </div>
  )
}
```

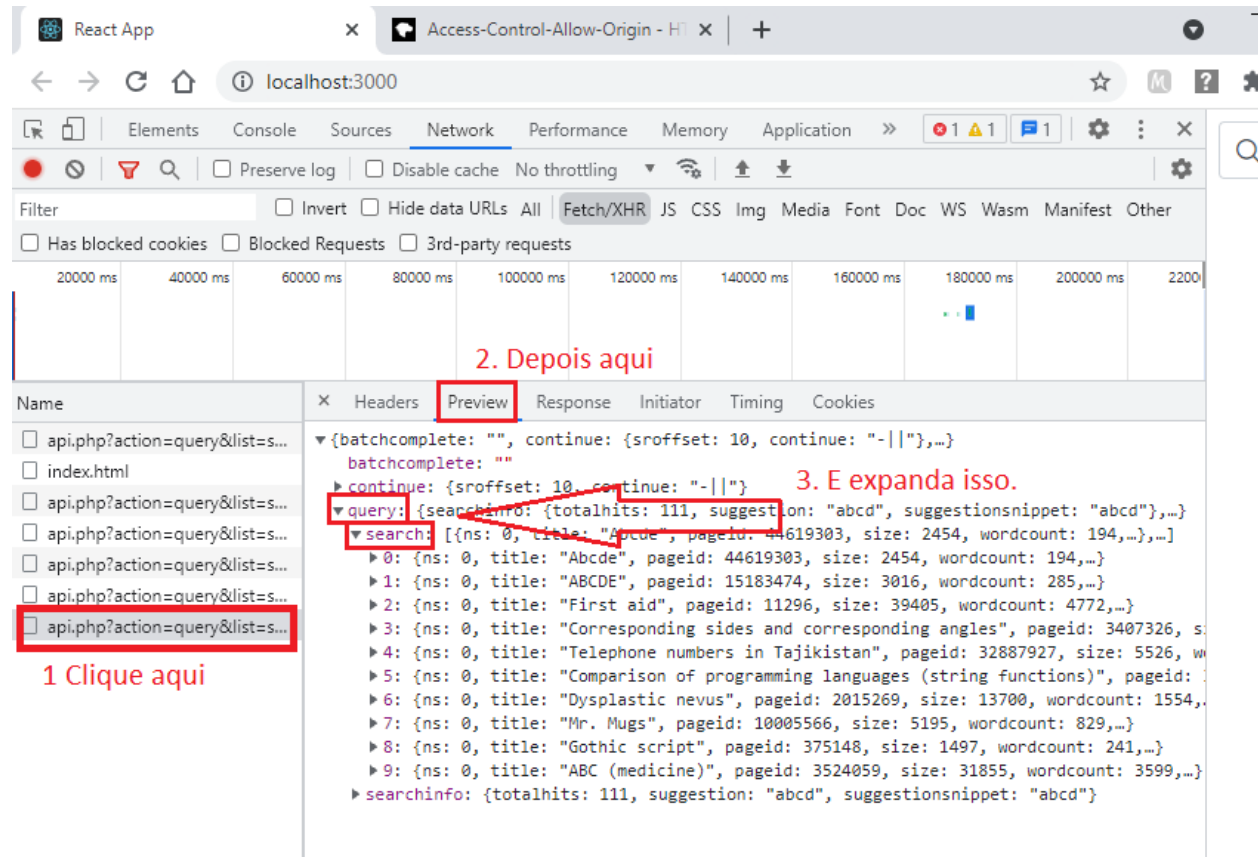
A aplicação não mostrará nada na tela por enquanto. Entretanto, é possível visualizar a requisição realizada no Chrome Dev Tools. Vá até a aba **Network** e clique **Fetch/XHR**, como a seguir. Clique atualizar no navegador para ver o resultado. Certifique-se de estar usando uma aba em que a aplicação está aberta.



Faça um novo teste. Digite algo no campo de busca e repare que uma nova requisição é realizada a cada alteração no campo. Veja.



Para conhecer a estrutura do resultado devolvido pela Wikipedia, Clique em uma das requisições realizadas e depois clique em **Preview**, como a seguir. Expanda **query** e depois **search**. Veja que search está associada a um array. Cada elemento no array é um resultado.



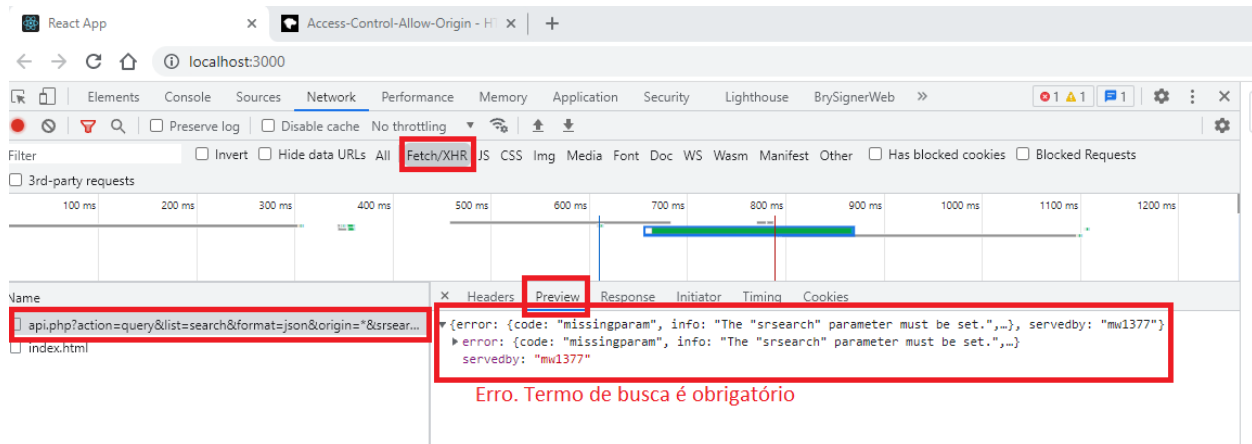
2.16 (Exibindo os resultados da busca) Uma vez que tenhamos o resultado de uma requisição, desejamos exibi-lo em uma lista. Para isso, vamos declarar uma nova variável que fará parte do estado do componente. Ela fará referência à lista de resultados e inicialmente será vazia. Veja.

```
const Busca = () => {  
  const [termoDeBusca, setTermoDeBusca] = useState('')  
  const [resultados, setResultados] = useState([])  
  useEffect(() => {  
    ...  
  })  
}
```

Quando a axios nos entrega o resultado, o corpo da resposta fica associado a uma propriedade chamada **data**. Isso é próprio da axios. Vamos aplicar o operador de desestruturação do Javascript para pegar somente esta parte que nos é de interesse. Uma vez obtida, ela é armazenada na lista que faz parte do estado. Veja.

```
const Busca = () => {
  const [termoDeBusca, setTermoDeBusca] = useState('')
  const [resultados, setResultados] = useState([])
  useEffect(() => {
    //define a função
    const fazBusca = async () => {
      const { data } = await axios.get(
        'https://en.wikipedia.org/w/api.php',{
          params: {
            action: 'query',
            list: 'search',
            format: 'json',
            // instruindo o navegador a permitir
            // conteúdo de qualquer origem
            origin: '*',
            srsearch: termoDeBusca
          }
        }
      )
      setResultados(data)
    }
    //chama a função
    fazBusca()
  }, [termoDeBusca])
  return (
```

Acesse novamente a aplicação e clique atualizar no seu navegador. Clique na requisição disponível, clique Preview e veja o resultado.



Podemos tratar este problema de duas formas.

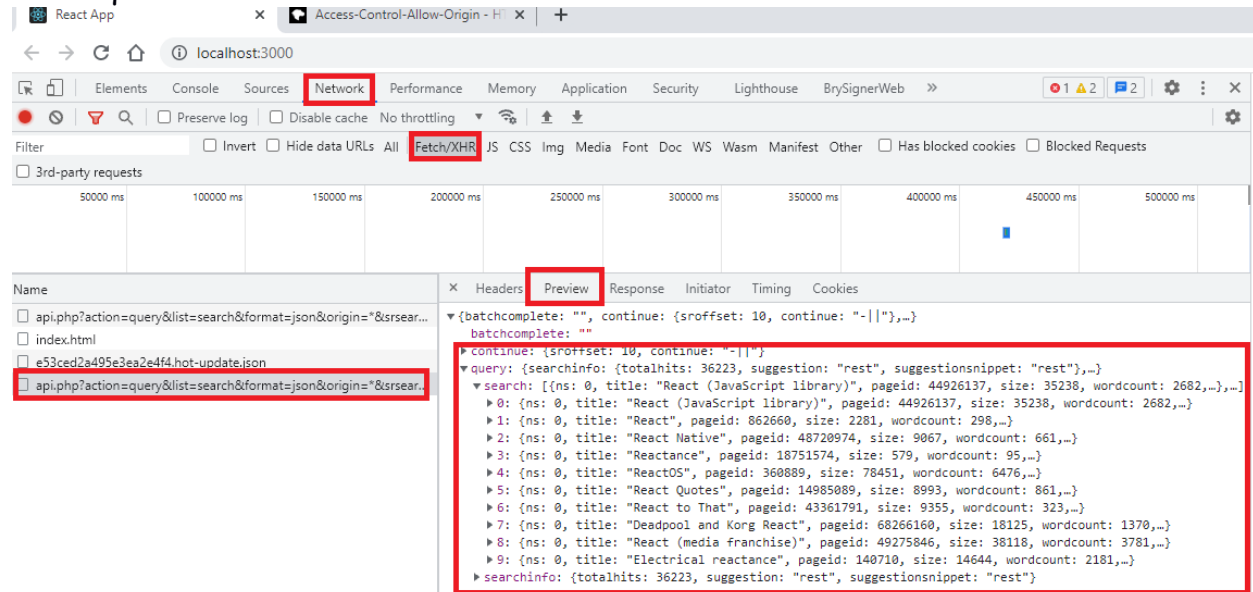
I. Configuramos um termo de busca padrão, utilizando automaticamente assim que o componente renderiza pela primeira vez.

II. Evitamos que a busca seja realizada caso o termo seja igual à cadeia vazia.

A primeira possibilidade fica assim. Digamos que o termo padrão é "React".

```
const Busca = () => {  
  const [termoDeBusca, setTermoDeBusca] = useState('React')  
  const [resultados, setResultados] = useState([])  
  ...  
}
```

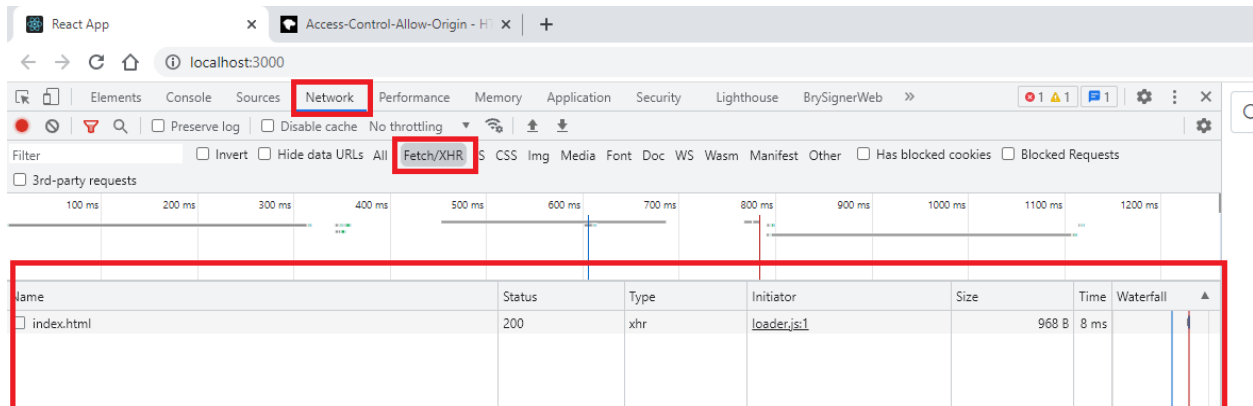

Atualize novamente a página do navegador. O resultado esperado se parece com esse aqui.



Para implementar a segunda possibilidade, mantemos o valor inicial da variável **termoDeBusca** igual à cadeia vazia e adicionamos uma estrutura condicional à função que faz a busca. Veja.

```
const Busca = () => {
  const [termoDeBusca, setTermoDeBusca] = useState('')
  const [resultados, setResultados] = useState([])
  useEffect(() => {
    //define a função
    const fazBusca = async () => {
      const { data } = await axios.get(
        'https://en.wikipedia.org/w/api.php',{
          params: {
            action: 'query',
            list: 'search',
            format: 'json',
            // instruindo o navegador a permitir
            // conteúdo de qualquer origem
            origin: '*',
            srsearch: termoDeBusca
          }
        })
      setResultados(data)
    }
    // só chama a função se existir termo de busca
    if (termoDeBusca){
      //chama a função
      fazBusca()
    }
  }, [termoDeBusca])
  return (
    ...
```

Atualize a página novamente. O resultado esperado agora se parece com esse aqui.



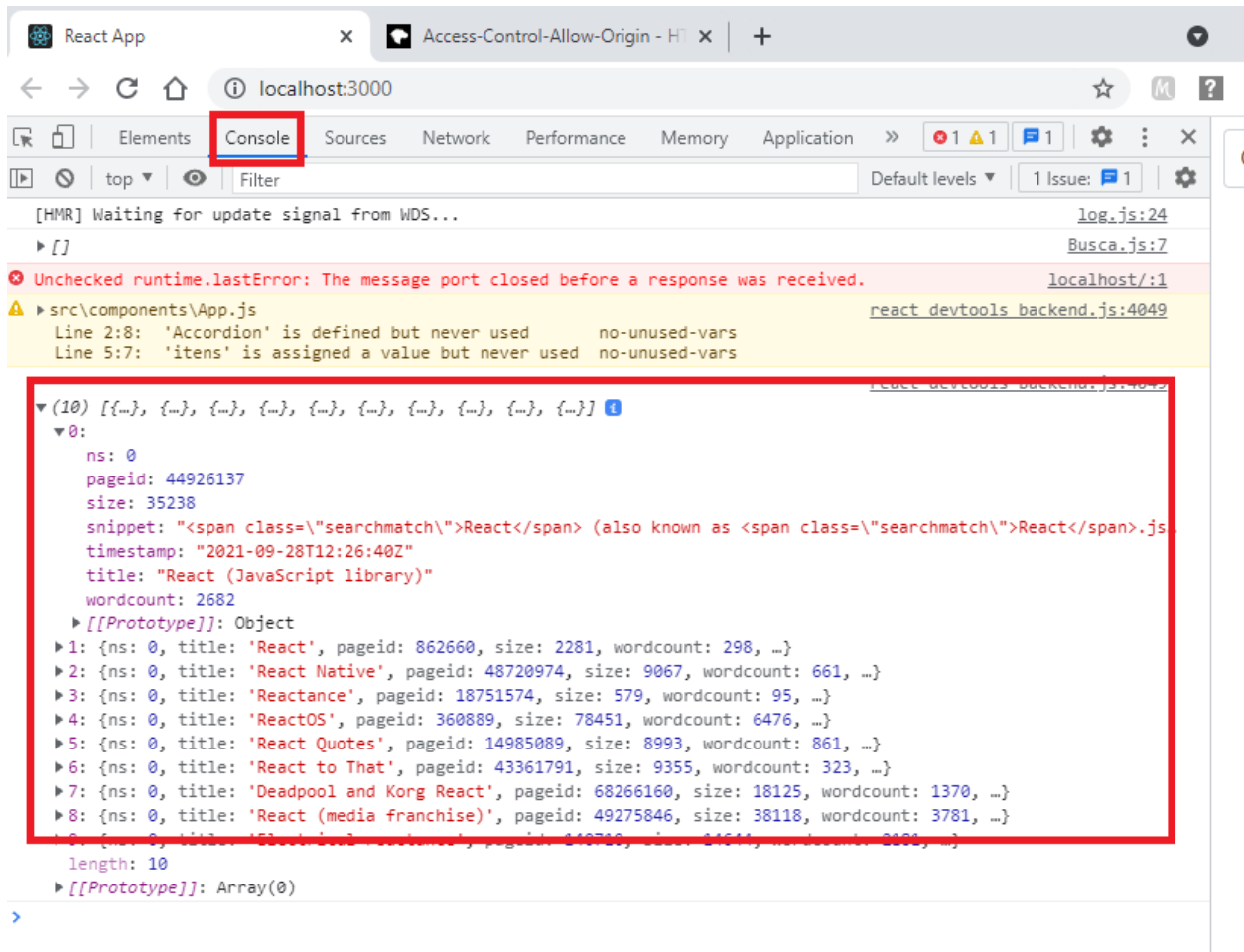
Neste material, **manteremos a primeira alternativa**, com o termo **React** configurado como padrão. Ajuste novamente o valor padrão da variável **termoDeBusca** e remova a estrutura condicional. Veja.

```
const Busca = () => {
  const [termoDeBusca, setTermoDeBusca] = useState('React')
  const [resultados, setResultados] = useState([])
  useEffect(() => {
    //define a função
    const fazBusca = async () => {
      const { data } = await axios.get(
        'https://en.wikipedia.org/w/api.php',{
          params: {
            action: 'query',
            list: 'search',
            format: 'json',
            // instruindo o navegador a permitir
            // conteúdo de qualquer origem
            origin: '*',
            srsearch: termoDeBusca
          }
        }
      )
      setResultados(data)
    }
    // execução incondicional
    fazBusca()
  }, [termoDeBusca])
  return (
    ...
```

O objeto que armazenamos na variável de estado **resultados** é o corpo inteiro da resposta. Estamos interessados somente no objeto associado à chave **search**. Como vimos, ela é filha de **query** que, por sua vez, está na raiz do corpo da resposta. Façamos, portanto, o seguinte ajuste.

```
const Busca = () => {
  const [termoDeBusca, setTermoDeBusca] = useState('React')
  const [resultados, setResultados] = useState([])
  console.log(resultados)
  useEffect(() => {
    //define a função
    const fazBusca = async () => {
      const { data } = await axios.get(
        'https://en.wikipedia.org/w/api.php',{
          params: {
            action: 'query',
            list: 'search',
            format: 'json',
            // instruindo o navegador a permitir
            // conteúdo de qualquer origem
            origin: '*',
            srsearch: termoDeBusca
          }
        }
      )
      setResultados(data.query.search)
    }
    // execução incondicional
    fazBusca()
  }, [termoDeBusca])
  return (
```

Atualize novamente a página e, agora, visite o console do navegador. Expanda alguns itens do resultado obtido. Você deve visualizar algo assim.



```
const Busca = () => {  
  ...  
  return (  
    <div>  
      <span className="p-input-icon-left">  
        <i className="pi pi-search"></i>  
        <InputText  
          onChange={(e) => setTermoDeBusca(e.target.value)}  
        />  
      </span>  
      {  
        resultados.map((resultado) => (  
          <div  
            key={resultado.pageid}  
            // margem e borda  
            className="my-2 border border-1 border-400">  
              <div  
                // borda, padding e ajuste textual  
                className="border-bottom border border-1  
                  border-400 p-2 text-center font-bold">  
                  {resultado.title}  
                </div>  
                { /* padding */ }  
                <div className="p-2">  
                  {resultado.snippet}  
                </div>  
              </div>  
            </div>  
          </div>  
        ))  
      }  
    </div>  
  )  
}
```

Após atualizar a página, o resultado deve ser algo assim.

<input type="text"/>
React (JavaScript library)
<code>React</code> (also known as <code>React.js</code> or <code>ReactJS</code>) is a free and open-source front-end JavaScript library for building user interfaces or UI components. It is maintained
React Native
<code>React Native</code> is an open-source UI software framework created by Facebook, Inc. It is used to develop applications for Android, Android TV, iOS, macOS
React
<code>REACT</code> or <code>React</code> may refer to: <code>REACT</code> (telescope), a telescope at Fenton Hill Observatory, New Mexico, US <code>React</code> (JavaScript library) , a JavaScript library
Reactance
<code>Reactance</code> may refer to: Electrical <code>reactance</code> , the opposition to a change in voltage due to capacitance (capacitive <code>reactance</code>) or in current due to inductance
ReactOS
<code>ReactOS</code> is a free and open-source operating system for amd64/i686 personal computers intended to be binary-compatible with computer programs and device
Deadpool and Korg React
Deadpool and Korg <code>React</code> is a 2021 American superhero promotional short film featuring the Marvel Comics characters Deadpool and Korg. The film was directed

Digite também algum termo de busca e veja o resultado. Digitando devagar, perceba que uma nova busca é realizada a cada alteração no campo.

2.17 (HTML que faz parte dos resultados: Ataque XSS (Cross Site Scripting)?)

Observe o texto de cada resultado. Cada um deles mostra um trecho de HTML que, certamente, não desejamos exibir em nossa página. Poderíamos, por exemplo, fazer uma espécie de localizar/substituir para remover esses trechos. Esta solução seria um tanto trabalhosa e não muito elegante. Há também alguns pacotes que fazem esse trabalho de maneira eficiente. Um exemplo é o pacote **striptags**. Ele pode ser instalado assim.

npm install striptags

Feita a sua instalação, ele pode ser usado da seguinte forma.

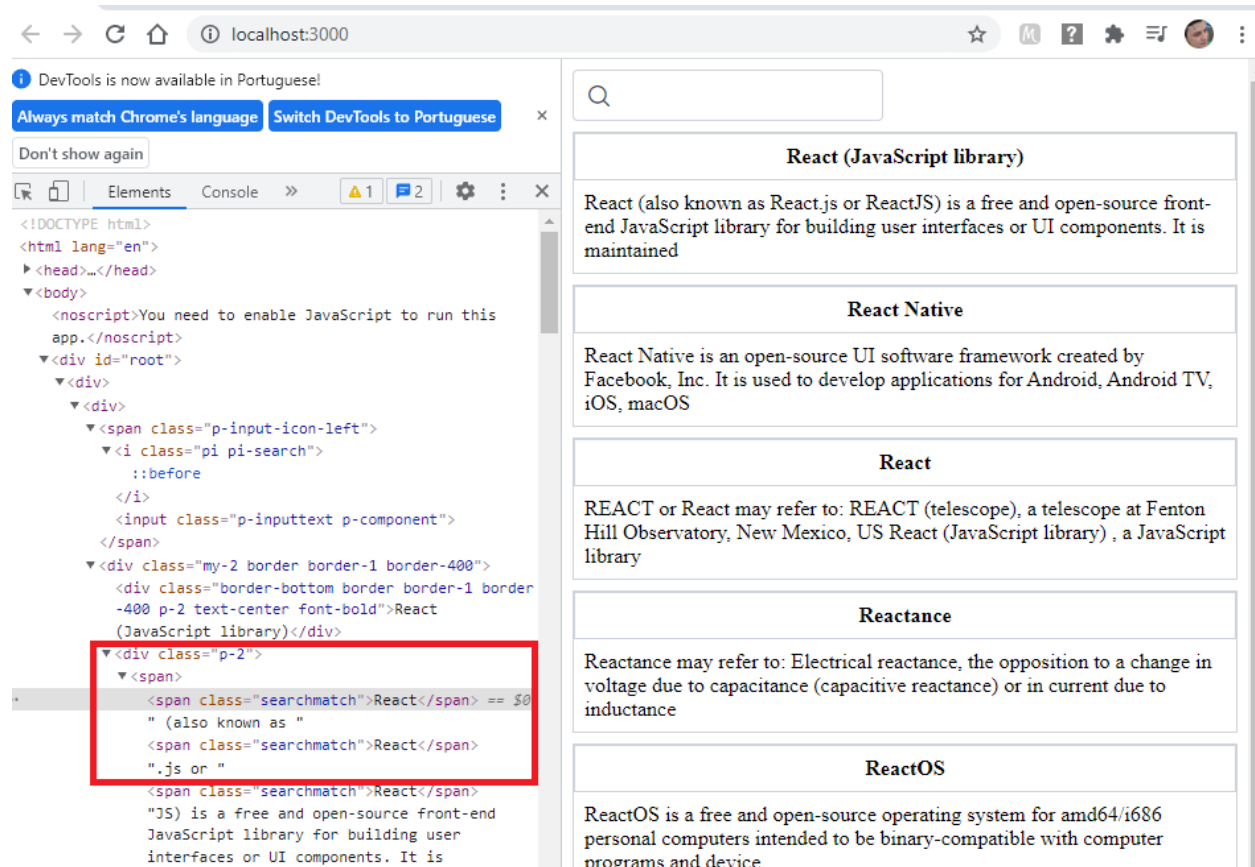
```
...
import striptags from 'striptags'
const Busca = () => {
  ...
  return (
    <div>
      <span className="p-input-icon-left">
        <i className="pi pi-search"></i>
        <InputText
          onChange={(e) => setTermoDeBusca(e.target.value)}
        />
      </span>
      {
        resultados.map((resultado) => (
          <div
            key={resultado.pageid}
            // margem e borda
            className="my-2 border border-1 border-400">
              <div
                // borda, padding e ajuste textual
                className="border-bottom border border-1 border-
400 p-2 text-center font-bold">
                {resultado.title}
              </div>
              {/* padding */}
              <div className="p-2">
                {striptags(resultado.snippet)}
              </div>
            </div>
          </div>
        ))
      }
    </div>
  )
}
```

Atualize a página e veja o resultado.

Há também uma outra possibilidade: instruir o React a interpretar o código HTML como tal. Ou seja, ao invés de exibir o HTML como simples texto, fazer com que cada tag seja interpretada, dando origem a um elemento na DOM. Essa é uma opção bastante arriscada e requer a plena certeza de que a fonte dos dados é segura e que a resposta não será interceptada antes de chegar até nosso app. Isso acontece, pois, código Javascript malicioso poderia fazer parte do resultado e a nossa aplicação estaria instruindo o navegador do usuário a executá-lo. Essa **injeção** de código, quando feita de forma maliciosa, é conhecida como **ataque XSS**. O tema é tão delicado que o mecanismo que o React disponibiliza para que possamos utilizar este recurso é um tanto específico. Veja como fica.

```
const Busca = () => {  
  ...  
  return (  
    <div>  
      <span className="p-input-icon-left">  
        <i className="pi pi-search"></i>  
        <InputText  
          onChange={(e) => setTermoDeBusca(e.target.value)}  
        />  
      </span>  
      {  
        resultados.map((resultado) => (  
          <div  
            key={resultado.pageid}  
            className="my-2 border border-1 border-400">  
              <div  
                className="border-bottom border border-1 border-400 p-2  
                  text-center font-bold">  
                  {resultado.title}  
                </div>  
                <div className="p-2">  
                  <span dangerouslySetInnerHTML={{__html: resultado.snippet}}>  
                  </span>  
                </div>  
              </div>  
            </div>  
          ))  
        }  
      </div>  
    )  
  )  
}
```

No Chrome Dev Tools, abra a aba **Elements** e veja o resultado. Como a seguir, cada elemento **span** existente no resultado que nos foi devolvido pela Wikipedia realmente deu origem a um elemento na árvore.








Vamos manter esta solução, já que a aplicação que estamos desenvolvendo não envolve muitos detalhes que poderiam nos colocar em risco, como contas, senhas etc.

2.18 (Visitando a página de cada artigo) Vamos adicionar um botão que permite visitarmos a página de cada artigo encontrado. Utilizaremos um Button da própria PrimeReact. Utilizamos o método **open** do objeto **window** para abrir uma nova aba. Veja.

```
const Busca = () => {  
  ...  
  return (  
    ...  
    resultados.map((resultado) => (  
      <div  
        key={resultado.pageid}  
        className="my-2 border border-1 border-400">  
        <div  
          className="border-bottom border border-1 border-  
400 p-2 text-center font-bold">  
          {resultado.title}  
          <span>  
            <Button  
              icon="pi pi-send"  
              className=" ml-2 p-button-rounded  
                p-button-secondary"  
              onClick= (() => window.open(  
                `https://en.wikipedia.org?curid=${resultado.pageid}` ) )  
            />  
          </span>  
        </div>  
        <div className="p-2">  
          <span dangerouslySetInnerHTML={{__html: resultado.sni  
ppet}}></span>  
        </div>  
      </div>  
    ))  
  )  
}
```

Depois de atualizar a página, o resultado deve ser parecido com esse.

<input type="text" value="Q"/>
React (JavaScript library) 
React (also known as React.js or ReactJS) is a free and open-source front-end JavaScript library for building user interfaces or UI components. It is maintained
React Native 
React Native is an open-source UI software framework created by Facebook, Inc. It is used to develop applications for Android, Android TV, iOS, macOS
React 
REACT or React may refer to: REACT (telescope), a telescope at Fenton Hill Observatory, New Mexico, US React (JavaScript library) , a JavaScript library
Reactance 
Reactance may refer to: Electrical reactance, the opposition to a change in voltage due to capacitance (capacitive reactance) or in current due to inductance
ReactOS 
ReactOS is a free and open-source operating system for amd64/i686 personal computers intended to be binary-compatible with computer programs and device

Clique em alguns botões e verifique se uma nova aba é aberta e se a página referente àquele artigo é exibida.

2.19 (Erro ao apagar todo o termo de busca: evitamos que a busca seja realizada) Caso digite um termo de busca e depois apague o conteúdo por completo, você perceberá que a aplicação produz um erro. Isso acontece pois estamos realizando uma busca sem um termo de busca. O resultado não contém, portanto, as chaves query e search que esperamos e o erro acontece aí. Podemos resolver esse problema simplesmente evitando que a busca seja realizada neste caso. Veja.

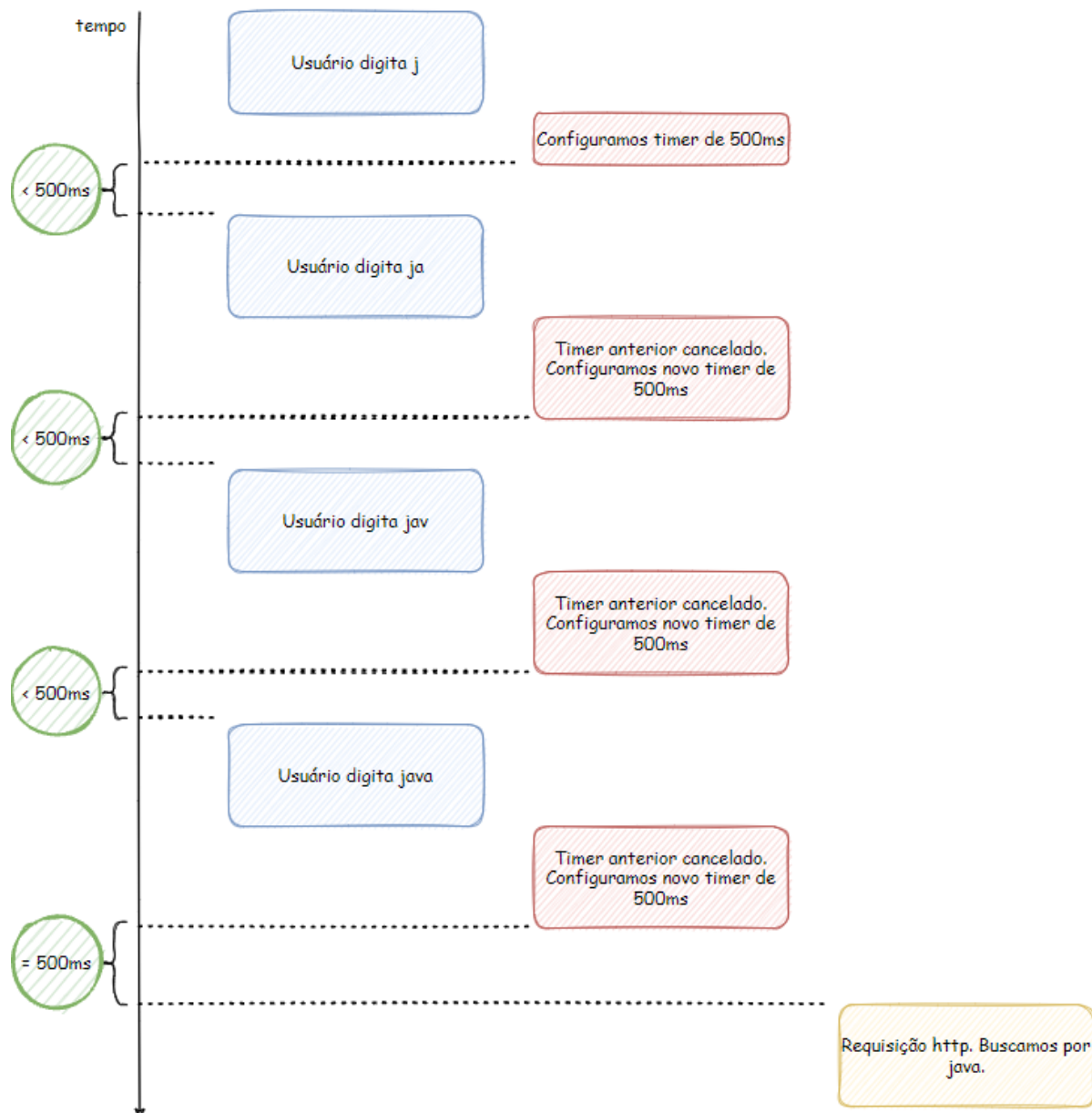
```
const Busca = () => {
  const [termoDeBusca, setTermoDeBusca] = useState('React')
  const [resultados, setResultados] = useState([])
  useEffect(() => {
    //define a função
    const fazBusca = async () => {
      const { data } = await axios.get(
        'https://en.wikipedia.org/w/api.php',{
          params: {
            action: 'query',
            list: 'search',
            format: 'json',
            // instruindo o navegador a permitir
            // conteúdo de qualquer origem
            origin: '*',
            srsearch: termoDeBusca
          }
        }
      )
      setResultados(data.query.search)
    }
    //execução condicional de novo
    if (termoDeBusca)
      fazBusca()
  }, [termoDeBusca])

  return (
```

2.20 (Ineficiência: Uma requisição por tecla digitada) Conforme o usuário digita, nosso aplicativo faz uma requisição a cada nova alteração no campo de busca, ou seja, a cada letra digitada pelo usuário. Mais ou menos como ilustrado a seguir.



Claramente, nossa aplicação é muito ineficiente. Se quisermos mantê-la fazendo as buscas conforme o usuário digita sem que ele tenha que clicar em um botão para isso, precisamos de uma estratégia diferente. Podemos, por exemplo, a cada vez que ele fizer uma alteração no campo de busca, configurar um timer com duração algo como um segundo. Claro, cabe a você decidir esse valor conforme testa a sua aplicação. Se o tempo do timer passar sem que o usuário tenha digitado algo novo, fazemos a requisição. Caso contrário, assim que o usuário digitar de novo, reiniciamos o timer. Veja.



Começamos configurando um timer a cada interação do usuário. Para isso, vamos usar a função **setTimeout**. Se necessário, dê uma olhada na sua documentação aqui.

<https://developer.mozilla.org/en-US/docs/Web/API/setTimeout>

Veja como a utilizamos.

```
const Busca = () => {  
  ...  
  useEffect(() => {  
    //define a função  
    const fazBusca = async () => {  
      const { data } = await axios.get(  
        'https://en.wikipedia.org/w/api.php',{  
          params: {  
            action: 'query',  
            list: 'search',  
            format: 'json',  
            // instruindo o navegador a permitir  
            // conteúdo de qualquer origem  
            origin: '*',  
            srsearch: termoDeBusca  
          }  
        }  
      )  
      setResultados(data.query.search)  
    }  
    setTimeout(() => {  
      //execução condicional de novo  
      if (termoDeBusca)  
        fazBusca()  
    }, 1000)  
  }, [termoDeBusca])  
}
```

Resta, é claro, cancelar cada timer conforme descrito. Para tal, podemos usar a função **clearTimeout**. Veja a sua documentação.

<https://developer.mozilla.org/en-US/docs/Web/API/clearTimeout>

Observe que, quando criamos um timeout, o navegador nos entrega um número. Ele é o identificador deste timeout. Seu identificador pode ser utilizado para cancelá-lo no futuro, por exemplo. E a função `clearTimeout` espera receber justamente um id de timeout que deve ser cancelado. Perfeito. Assim, vamos armazenar o id do timeout criado em uma constante. Veja.

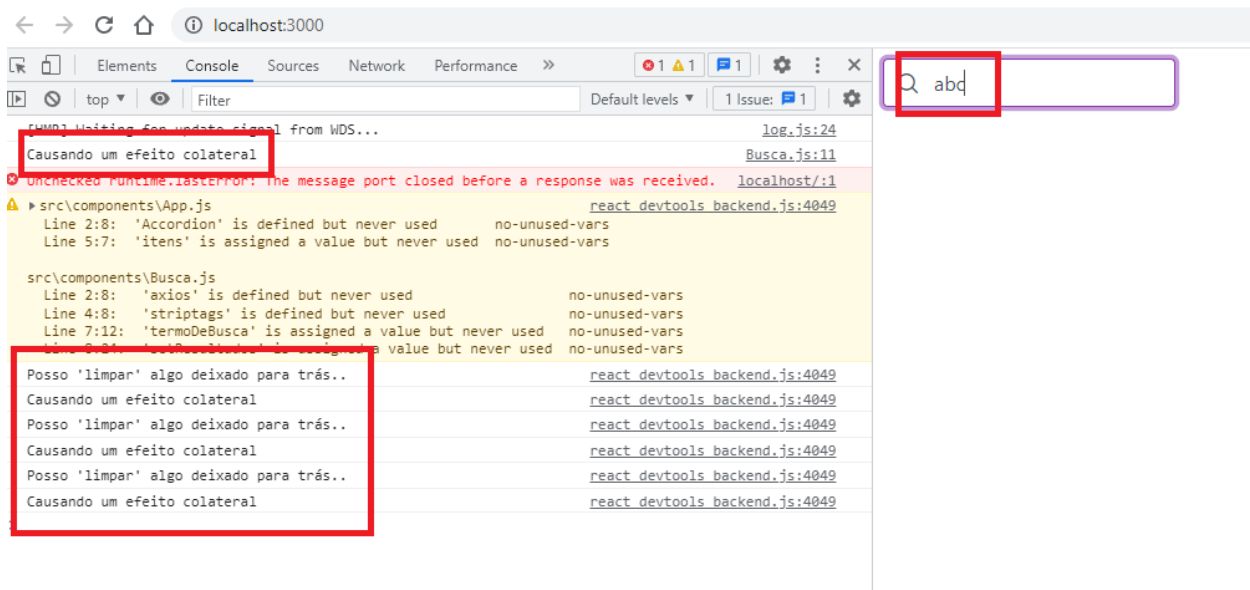
```
const Busca = () => {  
  ...  
  useEffect(() => {  
    //define a função  
    const fazBusca = async () => {  
      const { data } = await axios.get(  
        'https://en.wikipedia.org/w/api.php',{  
          params: {  
            action: 'query',  
            list: 'search',  
            format: 'json',  
            // instruindo o navegador a permitir  
            // conteúdo de qualquer origem  
            origin: '*',  
            srsearch: termoDeBusca  
          }  
        }  
      )  
      setResultados(data.query.search)  
    }  
    const timeoutID = setTimeout(() => {  
      //execução condicional de novo  
      if (termoDeBusca)  
        fazBusca()  
    }, 1000)  
  }, [termoDeBusca])  
}
```

Para chamar a função **`clearTimeout`** no momento certo, vamos fazer uso de mais um mecanismo provido pelo hook **`useEffect`**.

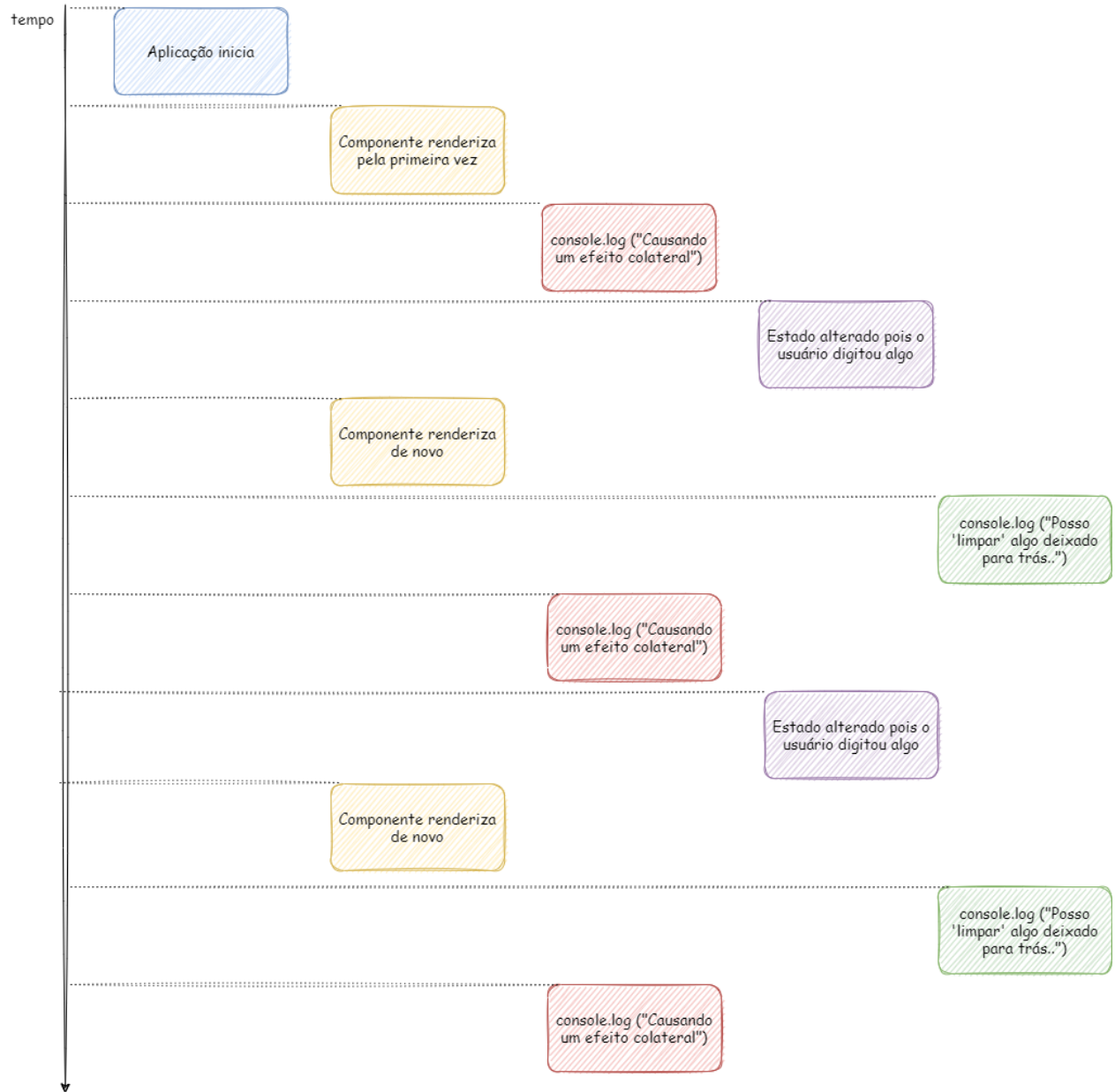
O hook `useEffect` pode devolver uma função. Ela é executada sempre antes de uma nova renderização. Para entender melhor o seu funcionamento, façamos o seguinte teste. Comente, momentaneamente, a chamada a `useEffect` que temos na aplicação. A seguir, chame `useEffect` da seguinte forma.

```
const Busca = () => {  
  // novo useEffect  
  useEffect(() => {  
    console.log ("Causando um efeito colateral")  
    return () => {  
      console.log ("Posso 'limpar' algo deixado para trás...")  
    }  
  })  
  //comente só por um momento  
  // useEffect(() => {  
  //   //define a função  
  //   const fazBusca = async () => {  
  //     const { data } = await axios.get(  
  //       'https://en.wikipedia.org/w/api.php',{  
  //         params: {  
  //           action: 'query',  
  //           list: 'search',  
  //           format: 'json',  
  //           // instruindo o navegador a permitir  
  //           // conteúdo de qualquer origem  
  //           origin: '*',  
  //           srsearch: termoDeBusca  
  //         }  
  //       }  
  //     )  
  //     setResultados(data.query.search)  
  //   }  
  //   const timeoutID = setTimeout(() => {  
  //     //execução condicional de novo  
  //     if (termoDeBusca)  
  //       fazBusca()  
  //   }, 1000)  
  // }, [termoDeBusca])  
}
```

Agora, execute a aplicação e atualize a página no navegador. Faça com que o componente seja renderizado mais algumas vezes, digitando algo no campo de busca. Veja o resultado no console do navegador. Ele deve ser parecido com esse aqui.



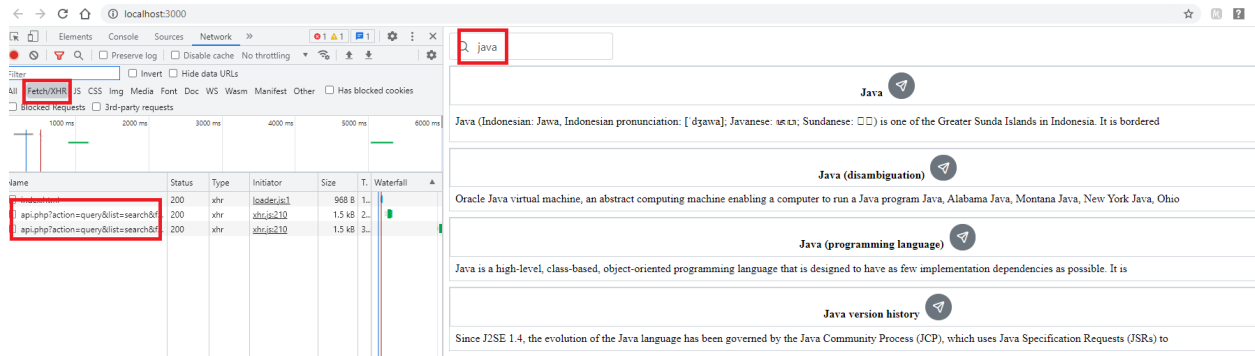
Veja o diagrama a seguir. Ele ilustra bem a ordem de execução das funções.



A chamada à função `clearTimeout` fica, portanto, da seguinte forma. Aliás, apague o `useEffect` que utilizamos para este último teste e “descomente” aquele que tínhamos previamente. Veja.

```
const Busca = () => {  
  ...  
  useEffect(() => {  
    //define a função  
    const fazBusca = async () => {  
      const { data } = await axios.get(  
        'https://en.wikipedia.org/w/api.php',{  
          params: {  
            action: 'query',  
            list: 'search',  
            format: 'json',  
            // instruindo o navegador a permitir  
            // conteúdo de qualquer origem  
            origin: '*',  
            srsearch: termoDeBusca  
          }  
        }  
      )  
      setResultados(data.query.search)  
    }  
    const timeoutID = setTimeout(() => {  
      //execução condicional de novo  
      if (termoDeBusca)  
        fazBusca()  
    }, 1000)  
    return () => {  
      clearTimeout(timeoutID)  
    }  
  }, [termoDeBusca])  
}
```

Faça um novo teste, digitando “java” no campo de busca. Se necessário, aumente o tempo do timeout para ver o resultado. A expectativa é que uma requisição somente aconteça quando, entre uma tecla digitada e outra, exista pelo menos a quantidade de tempo especificada no timeout. O exemplo a seguir mostra duas requisições. A primeira foi feita para o termo inicial. A segunda, somente depois de a palavra ter sido digitada completamente.



Repare que a primeira vez que a aplicação executa, há uma demora para que o resultado das buscas envolvendo o termo padrão seja exibido. Isso acontece pois, de fato, a busca só acontece depois do timeout. Podemos ajustar isso da seguinte forma.

I. Verificamos se há um termo de busca e se ainda não há resultados. Esta condição indica que podemos fazer uma busca pela primeira vez.

II. Se a condição anterior for falsa, então registramos o timeout e devolvemos a função de limpeza.

Veja.

```
const Busca = () => {  
  ...  
  useEffect(() => {  
    //define a função  
    const fazBusca = async () => {  
      const { data } = await axios.get(  
        'https://en.wikipedia.org/w/api.php',{  
          params: {  
            action: 'query',  
            list: 'search',  
            format: 'json',  
            // instruindo o navegador a permitir  
            // conteúdo de qualquer origem  
            origin: '*',  
            srsearch: termoDeBusca  
          }  
        }  
      )  
      setResultados(data.query.search)  
    }  
  
    if (termoDeBusca && !resultados.length){  
      fazBusca()  
    }  
    else{  
      const timeoutID = setTimeout(() => {  
        //execução condicional de novo  
        if (termoDeBusca)  
          fazBusca()  
      }, 1000)  
      return () => {  
        clearTimeout(timeoutID)  
      }  
    }  
  }, [termoDeBusca])  
}
```

Referências

React - A JavaScript library for building user interfaces. 2021. Disponível em <<https://reactjs.org/>>. Acesso em agosto de 2021.