Professor Rodrigo Bossini
https://www.rodrigobossini.com.br

ReactJS
Ciclo de Vida de componentes

1 Introdução

Componentes React definidos por meio de classes possuem um **ciclo de vida** que é caracterizado pela seguinte coleção de métodos. O construtor também faz parte dele.

- O **construtor**
- O método **render**
- O método **componentDidMount**
- O método **componentDidUpdate**
- O método **componentWillUnmount**

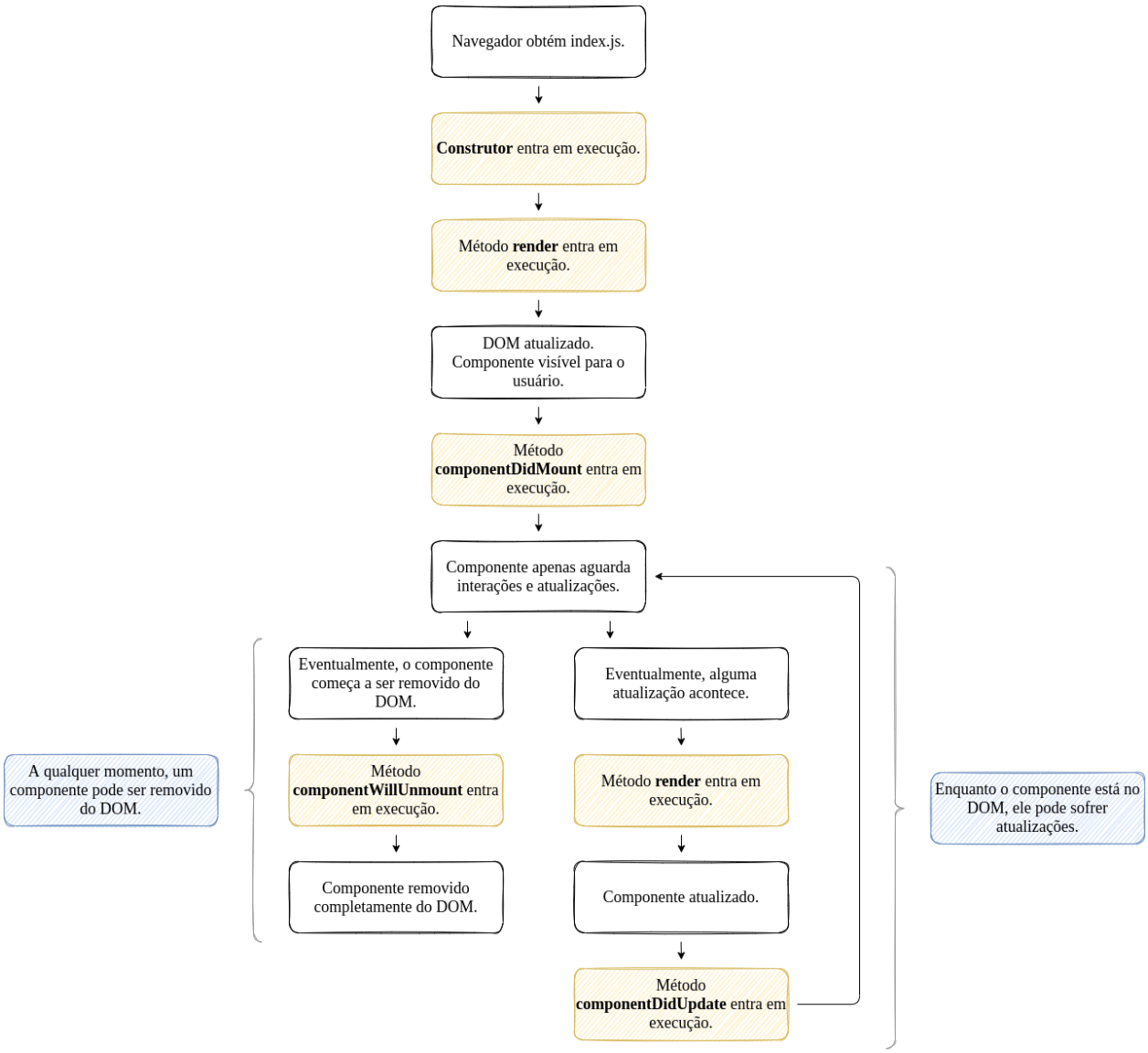
Nota. Há outros métodos que fazem parte do ciclo de vida de componentes React. Alguns são obsoletos e outros são raramente utilizados. Veja o Link 1.1.

Link 1.1
https://reactjs.org/docs/react-component.html

A Figura 1.1 ilustra o funcionamento do ciclo de vida de um componente React. É importante observar que **cada componente tem o seu próprio ciclo de vida**.

Produção: agosto de 2021

Figura 1.1



Neste material, utilizamos a aplicação desenvolvida anteriormente para estudar as principais características do ciclo de vida de componentes React.

2 Desenvolvimento

2.1 (Abrindo o projeto) Abra um terminal e utilize

cd diretorio-em-que-se-encontra-o-seu-projeto

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**.

2.2 (Testando os métodos do ciclo de vida) No arquivo **index.js**, inclua o conteúdo exibido no Bloco de Código 2.2.1. Implementamos os métodos do ciclo de vida do componente e exibimos uma mensagem de log, apenas para registrar a sua execução.

Bloco de Código 2.2.1

```
class App extends React.Component {
…
 constructor(props) {
       super(props)
       this.state = {
          latitude: null,
          longitude: null,
          estacao: null,
          data: null,
          icone: null,
          mensagemDeErro: null
       }
       console.log('construtor')

    }
…
    componentDidMount(){
       console.log('componentDidMount')
    }

    componentDidUpdate (){
       console.log('componentDidUpdate')
    }

    componentWillUnmount (){
       console.log('componentWillUnmount')
    }
    render() {
       console.log("render")
       return (
       ...
```

Abra o Chrome Dev Tools (CTRL + SHIFT + I) e clique no botão da aplicação algumas vezes, como mostra a Figura 2.2.1. O resultado esperado aparece na Figura 2.2.2.

Figura 2.2.1

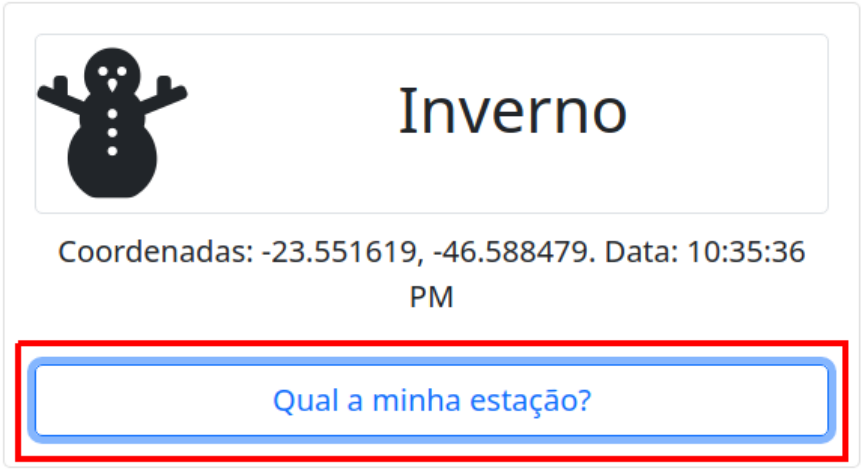
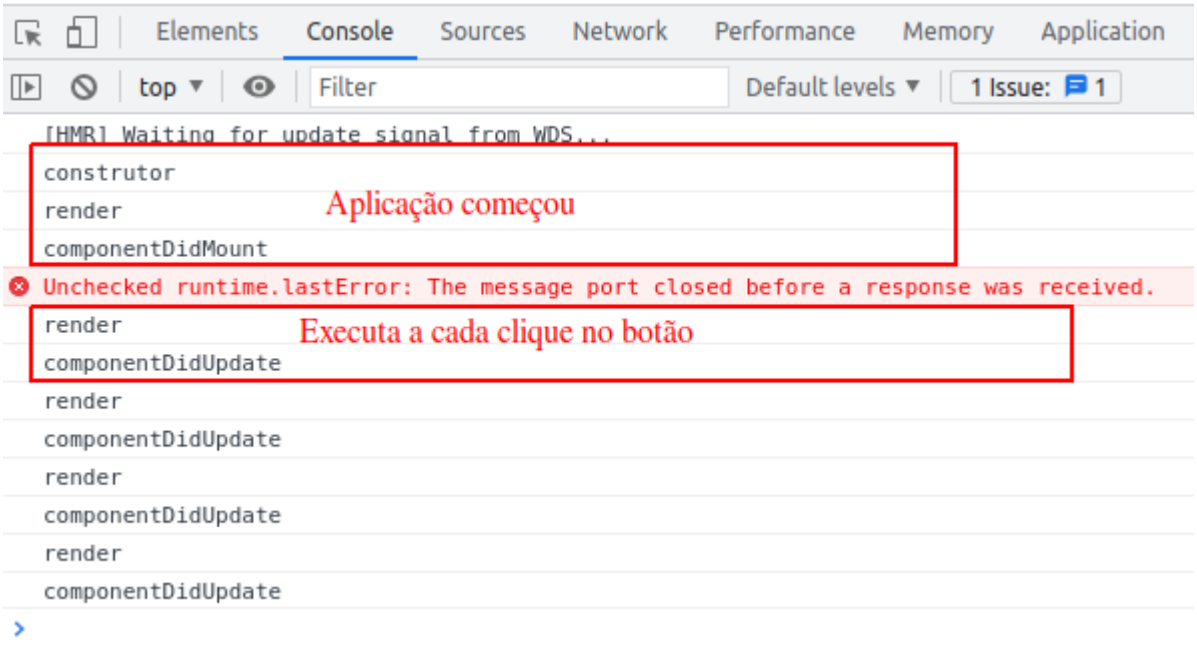


Figura 2.2.2



O método **componentWillUnmount** somente é colocado em execução quando um componente é removido do DOM. Apenas com o intuito de simular o seu funcionamento, adicione o botão destacado no Bloco de Còdigo 2.2.2. Utilizamos o método **unmountComponentAtNode de ReactDOM** para fazer a remoção do div cujo id é root.

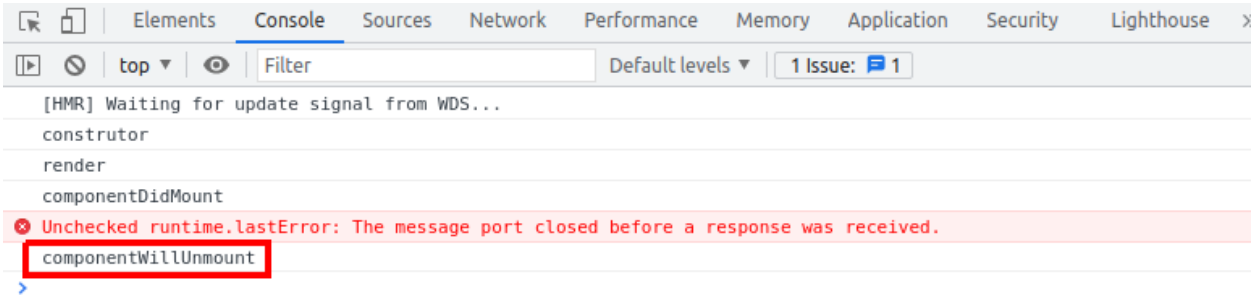
Bloco de Código 2.2.2

```
...
<button onClick={this.obterLocalizacao} className="btn btn-outline-primary w-100 mt-
2">Qual a minha estação?</button>

<button  className="btn btn-outline-danger w-100 mt-2"
            onClick={() =>
                ReactDOM.unmountComponentAtNode(document.querySelector('#root'))}>
              Unmount
 </button>
</div>
...
```

A Figura 2.2.3 mostra o resultado esperado no console.

Figura 2.2.3



Lembre-se que o componente principal da aplicação, representado por App, é filho da div cujo id é root. Quando ela é removida do DOM, todos os seus filhos também são. Isso quer dizer que, **após clicar no botão de teste, o componente App será removido e a tela deverá ficar em branco**. Depois do teste você pode remover o botão adicionado neste passo. Remova também os métodos do ciclo de vida e logs realizados pelo método render e pelo construtor. Eles poderão ser implementados no futuro, quando necessário.

A Tabela 2.2.1 mostra exemplos de aplicações para cada método, incluindo o construtor, do ciclo de vida de componentes React.

Tabela 2.2.1

Método / Construtor	Uso
Construtor	Executa uma única vez quando o componente é instanciado. Bom momento para fazer configurações iniciais que precisam ser feitas uma única vez.
render	Chamado logo depois de o componente ser instanciado e toda vez que ele tiver seu estado atualizado. Idealmente, o método render se encarrega apenas de produzir a expressão JSX, por questões de desempenho.
componentDidMount	Comumente utilizado para fazer a obtenção de dados iniciais, necessários para a exibição do componente, o que pode ser feito por meio de uma requisição HTTP, por exemplo.
componentDidUpdate	Obtenção de dados novos, possivelmente via novas requisições HTTP, em função de atualizações feitas no estado do componente.
componentWillUnmount	Hora certa para liberar recursos alocados. Um timer alocado pelo componente poderia ser desalocado aqui, por exemplo.

Nota. É recomendável reservar o construtor para atividades mais rápidas, como inicializar o estado do componente. O método componentDidMount, por outro lado, pode ser utilizado para atividades de inicialização um pouco mais demoradas, especialmente pelo fato de, no momento de sua execução, o componente já estar visível para o usuário. Em geral, é melhor mostrar um componente parcialmente visível para o usuário rapidamente do que demorar muito para mostrar um componente completo.

2.3 (Obtendo as informações do usuário com o método componentDidMount) Suponha que desejamos exibir a estação climática do usuário já no momento em que a aplicação entra em execução. Podemos fazê-lo, por exemplo, no método componentDidMount. Caso deseje, ele pode clicar no botão para atualizar as informações exibidas. Veja uma possível implementação do método componentDidMount no Bloco de Código 2.3.1.

Bloco de Código 2.3.1

```
class App extends React.Component {

  ...
  componentDidMount(){
      this.obterLocalizacao()
  }
  ...
```

Ao atualizar a aplicação no navegador, o resultado será diferente dependendo das configurações de permissão do usuário:

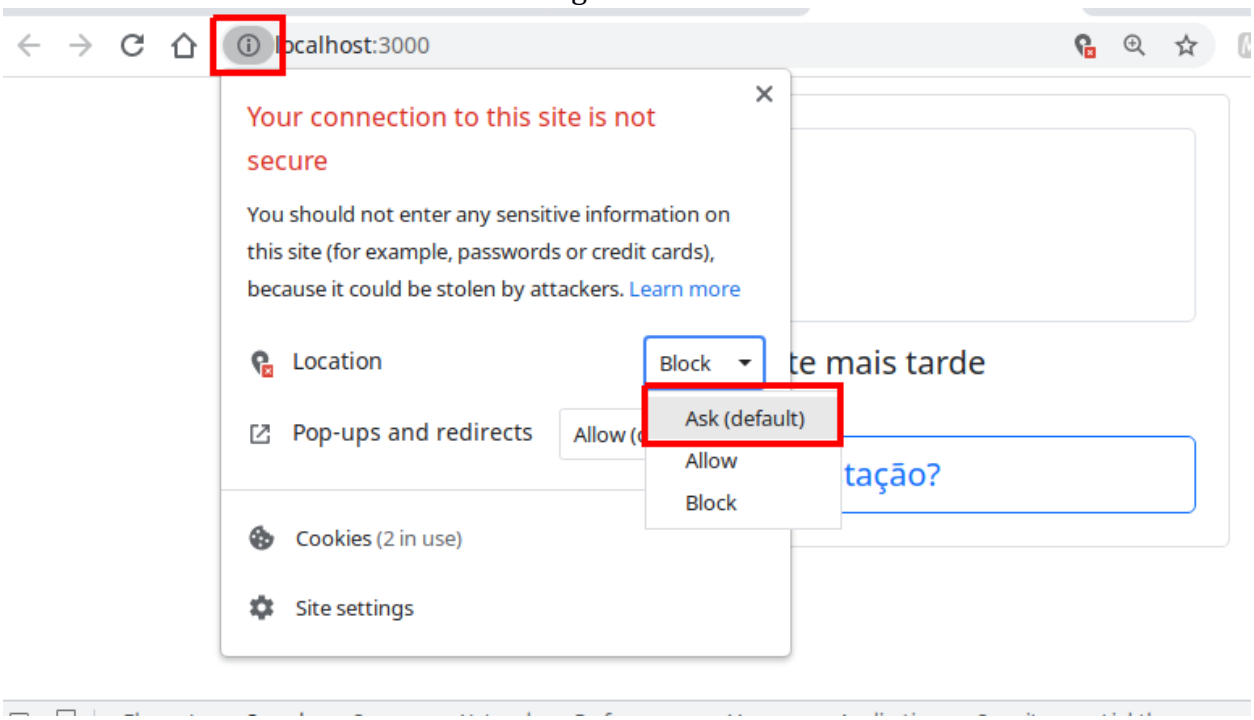
- Caso o navegador esteja configurado para permitir o acesso à localização, as informações serão exibidas imediatamente.

- Caso o navegador esteja configurado para bloquear o acesso à localização, a aplicação deverá exibir o texto que diz ao usuário que ele precisará tentar novamente mais tarde.

- Caso ainda não exista opção previamente selecionada, o navegador deverá perguntar ao usuário se ele permite o acesso às suas informações de localização assim que a aplicação for carregada.

Para fazer testes, você pode configurar o navegador para que ele pergunte ao usuário se ele deseja permitir ou bloquear o acesso às suas informações de localização, como mostra a Figura 2.3.1.

Figura 2.3.1



Atualize o navegador após fazer essa atualização.

2.4 (Inicialização do estado sem utilizar o construtor) É possível inicializar o estado da aplicação. Essa possibilidade pode ser de interesse por ser menos "verbosa". A inicialização de estado sem construtor é feita como mostra o Bloco de Código 2.4.1.

Bloco de Código 2.4.1

```
class App extends React.Component {

  //o construtor deixa de ser escrito explicitamente, comentamos a sua definição
  // constructor(props) {
  //     super(props)
  //     this.state = {
  //         latitude: null,
  //         longitude: null,
  //         estacao: null,
  //         data: null,
  //         icone: null,
  //         mensagemDeErro: null
  //     }
  //     console.log('construtor')

  // }
  //inicializando o estado sem usar o construtor.
  state = {
      latitude: null,
      longitude: null,
      estacao: null,
      data: null,
      icone: null,
      mensagemDeErro: null
  }
```

2.5 (Refatorando a aplicação: um componente para a exibição da estação climática. Passando estado via props) Nossa aplicação possui um único componente responsável por resolver todos os problemas, o que viola diversos princípios do desenvolvimento, como a **alta coesão**. Como sabemos, essa prática tende a dar origem a código de difícil manutenção, baixo nível de reusabilidade etc. Por essa razão, vamos criar um componente cuja finalidade será apenas exibir os dados referentes à estação climática e data, uma vez que eles estiverem disponíveis.

- Começamos criando um arquivo chamado **EstacaoClimatica.js** na pasta **src**. Para tal, clique com direito sobre ela e escolha **New File.**

- A seguir, faça a definição do componente como mostra o Bloco de Código2.5.1. Note que utilizamos a mesma definição existente no arquivo index.js, começando a partir da definição do cartão do Bootstrap. Assim, basta recortar esse trecho e colar no corpo do novo componente.

Bloco de Código 2.5.1

```
import React from 'react'
export class EstacaoClimatica extends React.Component {
   render ( ){
        return (
     <div className="card">
        {/* o corpo do cartão */}
        <div className="card-body">
           {/* centraliza verticalmente, margem abaixo */}
           <div className="d-flex align-items-center border rounded mb-2" style={{ height: '6rem' }}>
              {/* ícone obtido do estado do componente */}
              <i className={`fas fa-5x ${this.state.icone}`}></i>
              {/* largura 75%, margem no à esquerda (start), fs aumenta a fonte */}
              <p className=" w-75 ms-3 text-center fs-1">{this.state.estacao}</p>
           </div>
           <div>
              <p className="text-center">
                 {/* renderização condicional */}
                 {
                    this.state.latitude ?
                       `Coordenadas: ${this.state.latitude}, ${this.state.longitude}. Data: ${this.state.data}`
                       :
                       this.state.mensagemDeErro ?
                          `${this.state.mensagemDeErro}`
                          :
                          'Clique no botão para saber a sua estação climática'

                 }
              </p>
           </div>
           {/* botão azul (outline, 100% de largura e margem acima) */}
           <button onClick={this.obterLocalizacao} className="btn btn-outline-primary w-100 mt-
2">Qual a minha estação?</button>
        </div>
     </div>
   )

 }
}
```

- A seguir, ajustamos o arquivo **index.js** para que o componente principal passe a fazer uso do componente recém-criado. Veja o Bloco de Código 2.5.2.

Bloco de Código 2.5.2

```
...
import { EstacaoClimatica } from './EstacaoClimatica'

class App extends React.Component {
…
render() {
        // console.log("render")
        return (
            // responsividade, margem acima
            <div className="container mt-2">
                {/* uma linha, conteúdo centralizado, display é flex */}
                <div className="row justify-content-center">
                {/* oito colunas das doze disponíveis serão usadas para telas médias em diante */}
                    <div className="col-md-8">
                    <EstacaoClimatica />
                    </div>
                </div>
            </div>
        )
    }
...
```

- Salve ambos arquivos e atualize a aplicação no navegador. Repare que ele mostra um erro, que deve ser parecido com aquele que a Figura 2.5.1 destaca.

Figura 2.5.1

TypeError: Cannot read property 'icone' of null

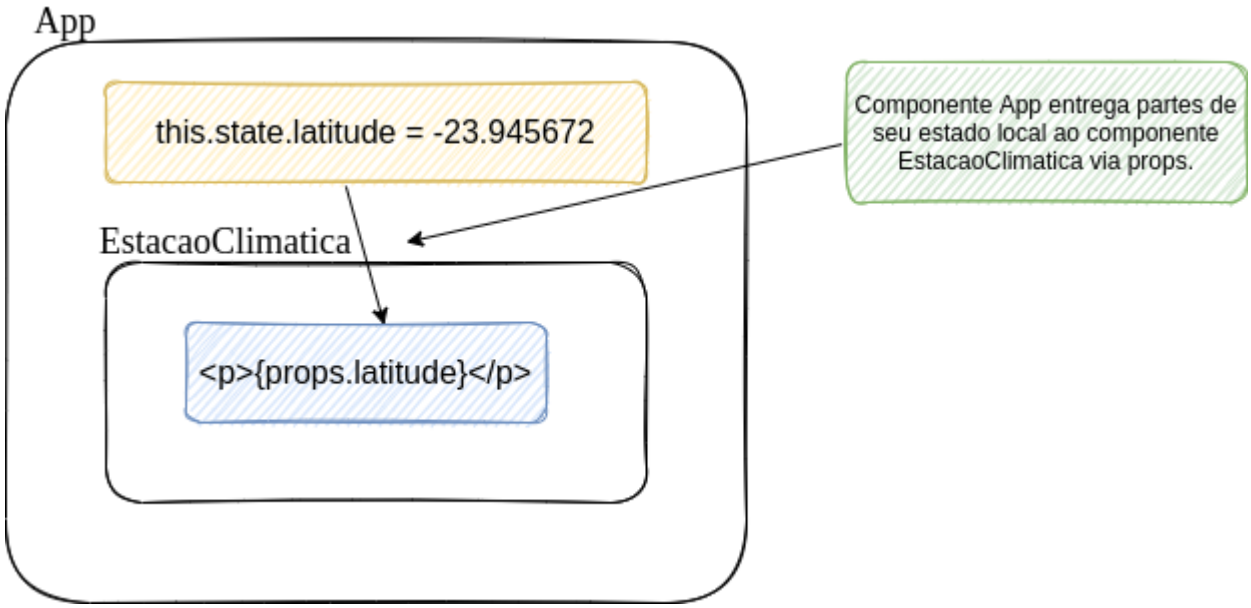
EstacaoClimatica.render
src/EstacaoClimatica.js:11

```
   8 | {/* centraliza verticalmente, margem abaixo */}
   9 | <div className="d-flex align-items-center border rounded mb-2" style={{ height: '6rem' }}>
  10 |     {/* ícone obtido do estado do componente */}
> 11 |     <i className={`fas fa-5x ${this.state.icone}`}></i>
  12 | ^   {/* largura 75%, margem no à esquerda (start), fs aumenta a fonte */}
  13 |     <p className=" w-75 ms-3 text-center fs-1">{this.state.estacao}</p>
  14 | </div>
```

O que acontece é que **o componente EstacaoClimatica está tentando acessar os dados de interesse utilizando a expressão this.state**. Ela se refere a seu estado em particular e não ao estado do componente principal. Além disso, ele ainda não foi definido. Aliás, cada componente tem seu próprio estado que, a princípio, não pode ser acessado pelos demais.

Como o componente principal é responsável por obter os dados de interesse e os armazenar em seu próprio estado, precisamos de algum mecanismo que permita que ele os entregue ao componente EstacaoClimatica uma vez que estiverem disponíveis. Para tal, **o componente App pode entregar partes de seu estado ao componente EstacaoClimatica utilizando o mecanismo props.** A Figura 2.5.2 ilustra a ideia.

Figura 2.5.2



Ajustamos, portanto, o componente EstacaoClimatica (arquivo **EstacaoClimatica.js**) para que ele acesse os dados de interesse por meio de seu objeto props. Em particular, repare que a função que entra em execução quando o botão é clicado permanece fazendo parte do componente App, já que ela altera seu estado. Assim, ela também deve ser passada via props. Veja o Bloco de Código 2.5.3.

Bloco de Código 2.5.3

```
export class EstacaoClimatica extends React.Component {
   render(){
      return (
         <div className="card">
            {/* o corpo do cartão */}
            <div className="card-body">
               {/* centraliza verticalmente, margem abaixo */}
               <div className="d-flex align-items-center border rounded mb-2"
                                 style={{ height: '6rem' }}>
                  {/* ícone obtido do estado do componente */}
                  <i className={`fas fa-5x ${this.props.icone}`}></i>
                  {/* largura 75%, margem no à esquerda (start), fs aumenta a fonte */}
                  <p className=" w-75 ms-3 text-center fs-1">{this.props.estacao}</p>
               </div>
               <div>
                  <p className="text-center">
                     {/* renderização condicional */}
                     {
                        this.props.latitude ?
                           `Coordenadas: ${this.props.latitude}, ${this.props.longitude}. Data:
${this.props.data}`
                           :
                           this.props.mensagemDeErro ?
                              `${this.props.mensagemDeErro}`
                              :
                              'Clique no botão para saber a sua estação climática'

                     }
                  </p>
               </div>
               {/* botão azul (outline, 100% de largura e margem acima) */}
               <button onClick={this.props.obterLocalizacao} className="btn btn-outline-primary w-100
mt-2">Qual a minha estação?</button>
            </div>
         </div>
      )
   }}
```

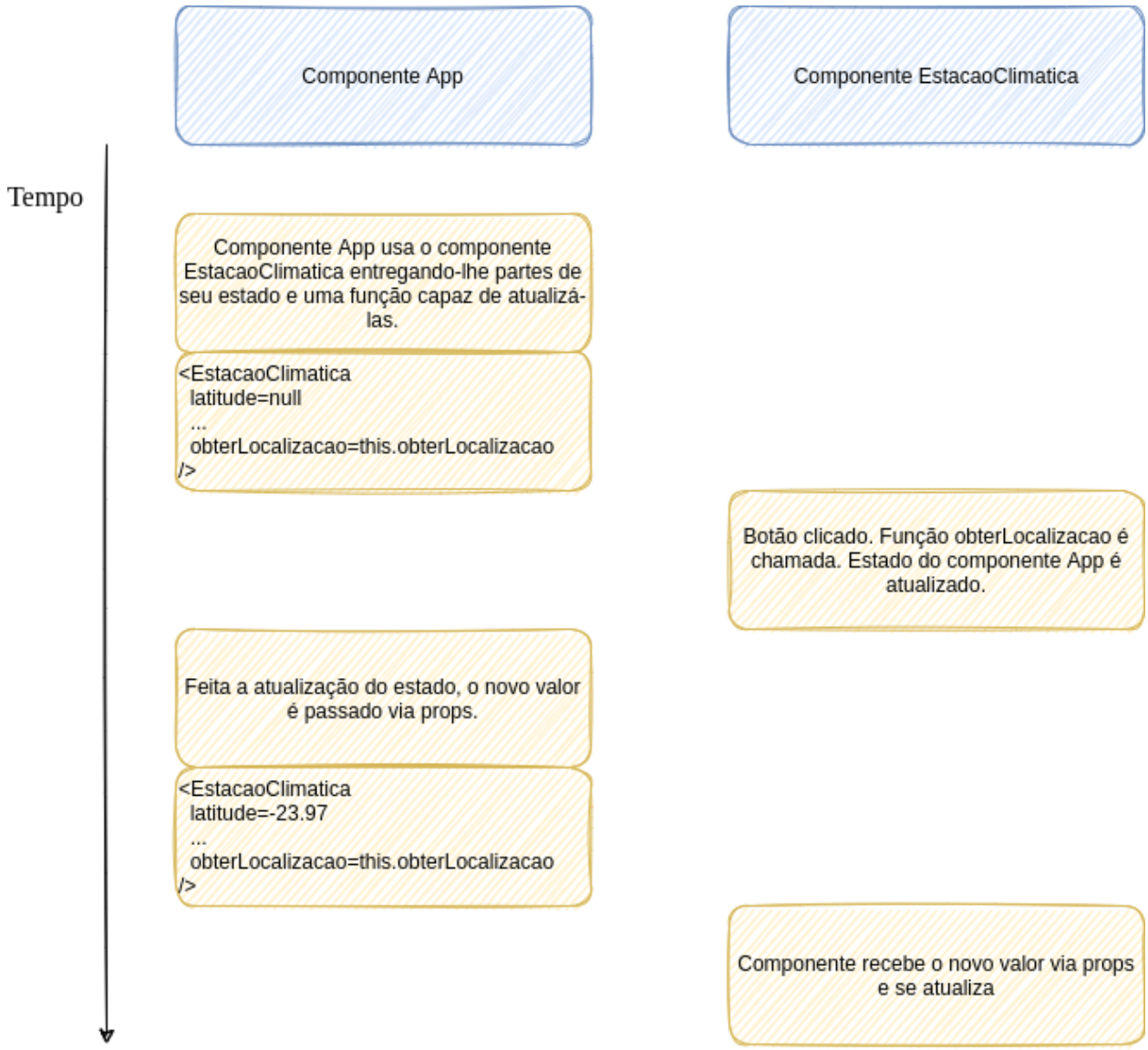
O Bloco de Código 2.5.4 mostra a forma como o componente App entrega partes do seu estado ao componente EstacaoClimatica via props, incluindo a função de obtenção de localização (que, a rigor, não faz parte de seu estado).

Bloco de Código 2.5.4

```
…
class App extends React.Component {
…
render() {
        // console.log("render")
        return (
            // responsividade, margem acima
            <div className="container mt-2">
                {/* uma linha, conteúdo centralizado, display é flex */}
                <div className="row justify-content-center">
                {/* oito colunas das doze disponíveis serão usadas para telas médias em diante */}
                    <div className="col-md-8">
                    <EstacaoClimatica
                        icone={this.state.icone}
                        estacao={this.state.estacao}
                        latitude={this.state.latitude}
                        longitude={this.state.longitude}
                        data={this.state.data}
                        mensagemDeErro={this.state.mensagemDeErro}
                        obterLocalizacao={this.obterLocalizacao}
                    />
                    </div>
                </div>
            </div>
        )
    }
…
```

Repare que o componente EstacaoClimatica recebe partes do estado do componente App e as atualiza utilizando uma função que também pertence ao componente App. Assim, o componente EstacaoClimatica somente pode alterar o estado do componente App indiretamente, utilizando uma função que ele mesmo define e decide passar via props. Feita a atualização do estado do componente App, o novo valor é entregue ao componente EstacaoClimatica via props, que se atualiza. Veja a Figura 2.5.3.

Figura 2.5.3


Componente App
Componente EstacaoClimatica
Tempo
Componente App usa o componente EstacaoClimatica entregando-lhe partes de seu estado e uma função capaz de atualizá-las.
<EstacaoClimatica
latitude=null
...
obterLocalizacao=this.obterLocalizacao
/>
Botão clicado. Função obterLocalizacao é chamada. Estado do componente App é atualizado.
Feita a atualização do estado, o novo valor é passado via props.
<EstacaoClimatica
latitude=-23.97
...
obterLocalizacao=this.obterLocalizacao
/>
Componente recebe o novo valor via props e se atualiza

2.7 (Um exemplo de aplicação para o método componentWillUnmount) Suponha que desejamos que o horário, uma vez exibido, seja incrementado a cada segundo. Utilizando Javascript "puro", podemos conseguir esse efeito usando a função **setInterval**. Lembre-se que ela recebe uma função e um valor numérico - em milissegundos - como parâmetro. A sua missão é agendar a execução da função, o que deve acontecer a cada intervalo de tempo especificado. Quando a repetição não for mais de interesse, devemos chamar a função **clearInterval**, a fim de evitar que a execução agendada prossiga indefinidamente. Isso será feito no método **componentWillUnmount**.

- Declaramos uma variável que referenciará o timer criado. É por meio dela que poderemos cancelar a sua execução, quando necessário. Veja o Bloco de Código 2.7.1.

Bloco de Código 2.7.1

```
...
export class EstacaoClimatica extends React.Component {
...
    timer = null
...
```

O componente EstacaoClimatica será o responsável por atualizar a data a cada segundo. Por isso, ela passa a ser definida em seu estado, como mostra o Bloco de Código 2.7.2.

Bloco de Código 2.7.2

```
export class EstacaoClimatica extends React.Component {

    state = {
        data: null
    }
...
```

- Definimos os métodos do ciclo de vida do componente EstacaoClimatica e registrando a sua execução no console. Além disso, o timer é colocado em execução assim que o componente é renderizado e cancelado quando ele estiver prestes a ser removido. Veja o Bloco de Código 2.7.3.

Bloco de Código 2.7.3

```
export class EstacaoClimatica extends React.Component {

    timer = null

    componentDidMount(){
        console.log("componentDidMount")
        this.timer = setInterval (() => {
            this.setState({data: new Date().toLocaleTimeString()})
        }, 1000)
    }

    componentWillUnmount() {
        console.log('componentWillUnmount')
        clearInterval(this.timer)
    }

    componentDidUpdate(){
        console.log ('componentDidUpdate')
    }
    render(){
        console.log ('render')
...
```

- O componente deixa de utilizar a data recebida via props e passa a utilizar aquela definida em seu estado. Veja o Bloco de Código 2.7.4. Além disso, ele deixa de verificar se uma mensagem de erro deve ser exibida, pois ele somente será renderizado caso o usuário tenha dado permissão de acesso à localização.

Bloco de Código 2.7.4

```
...
<div>
   <p className="text-center">
      {/* renderização condicional */}
      {
        this.props.latitude ?
        `Coordenadas: ${this.props.latitude}, ${this.props.longitude}. Data: ${this.state.data}`
        :
        'Clique no botão para saber a sua estação climática'

      }
   </p>
</div>
...
```

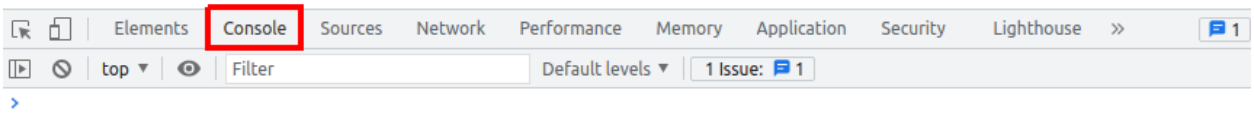
- O componente **App**, por sua vez, deixa de passar data e mensagem de erro ao componente EstacaoClimatica. Ele decide o que exibir em função da variável mensagemDeErro, que faz parte de seu estado. O componente EstacaoClimatica somente será colocado na árvore caso não exista uma mensagem de erro armazenada. Caso contrário, um componente textual simples será exibido. Veja o Bloco de Código 2.7.5.

Bloco de Código 2.7.5

```
class App extends React.Component {
…
render() {
      // console.log("render")
      return (
         // responsividade, margem acima
         <div className="container mt-2">
            {/* uma linha, conteúdo centralizado, display é flex */}
            <div className="row justify-content-center">
            {/* oito colunas das doze disponíveis serão usadas para telas médias em diante */}
               <div className="col-md-8">
                  {
                     this.state.mensagemDeErro ?
                        <p className="border rounded p-2 fs-1 text-center">
                           É preciso dar permissão para acesso à localização.
                           Atualize a página e tente de novo, ajustando a configuração
                           no seu  navegador.
                        </p>
                     :
                     <EstacaoClimatica
                           icone={this.state.icone}
                           estacao={this.state.estacao}
                           latitude={this.state.latitude}
                           longitude={this.state.longitude}
                           obterLocalizacao={this.obterLocalizacao}
                     />
                  }
               </div>
            </div>
         </div>
      )
   }
```

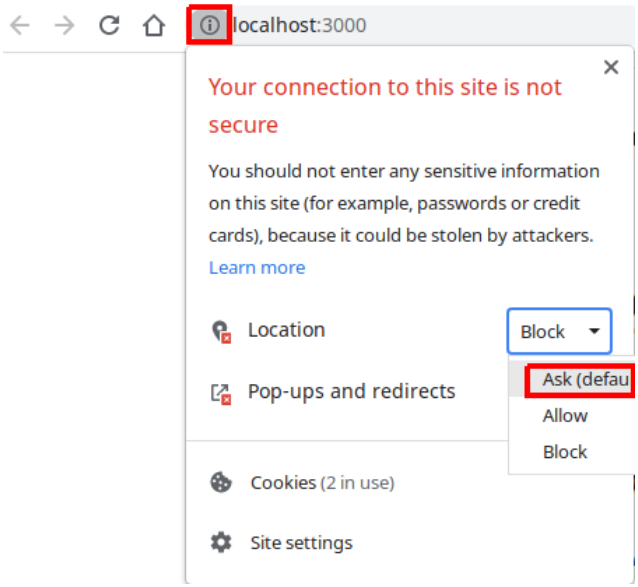
Para testar, certifique-se de que o console do Chrome Dev Tools está aberto, como na Figura 2.7.1.

Figura 2.7.1



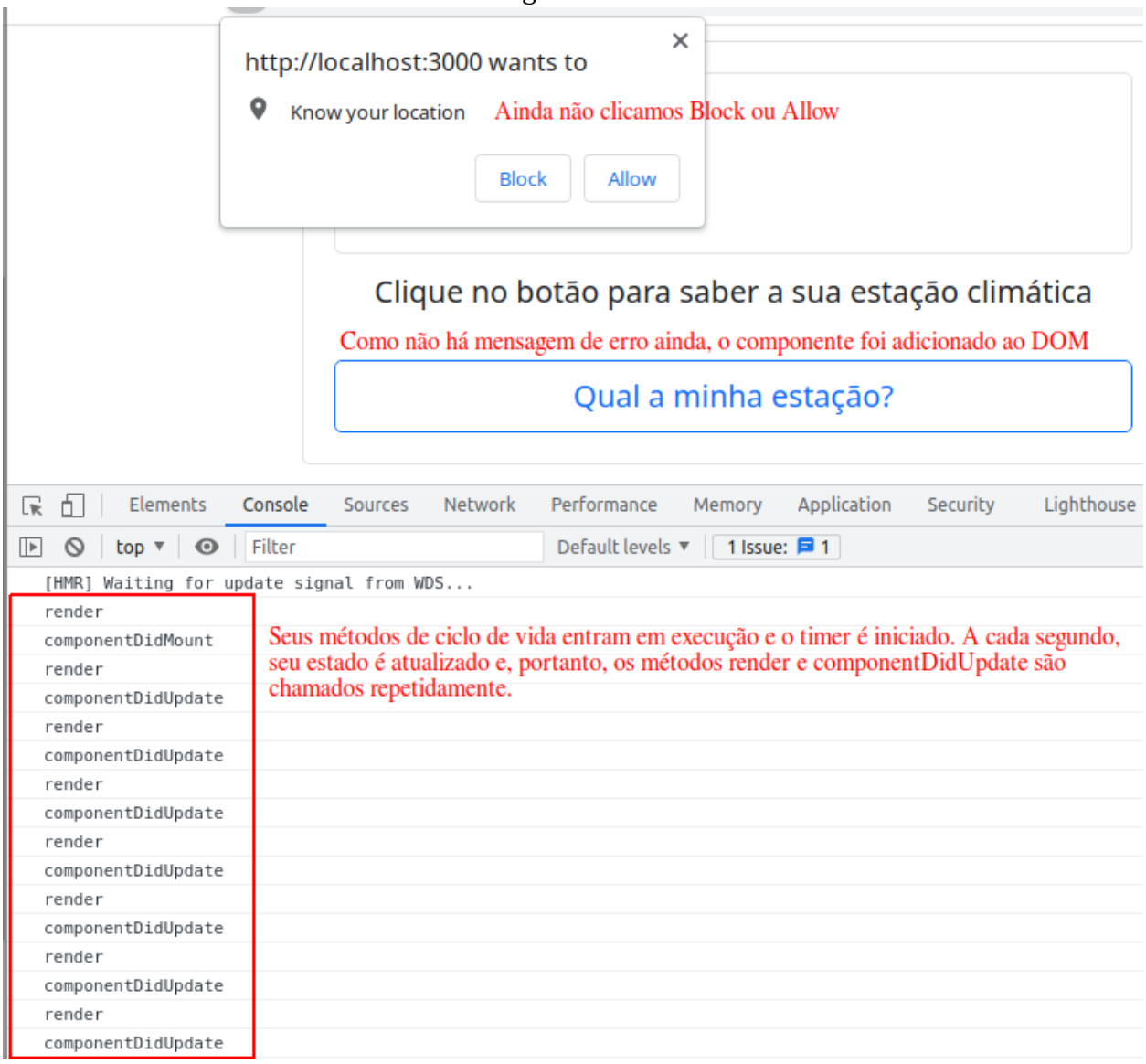
A seguir, instrua o navegador a pedir a permissão do usuário, como na Figura 2.7.2.

Figura 2.7.2



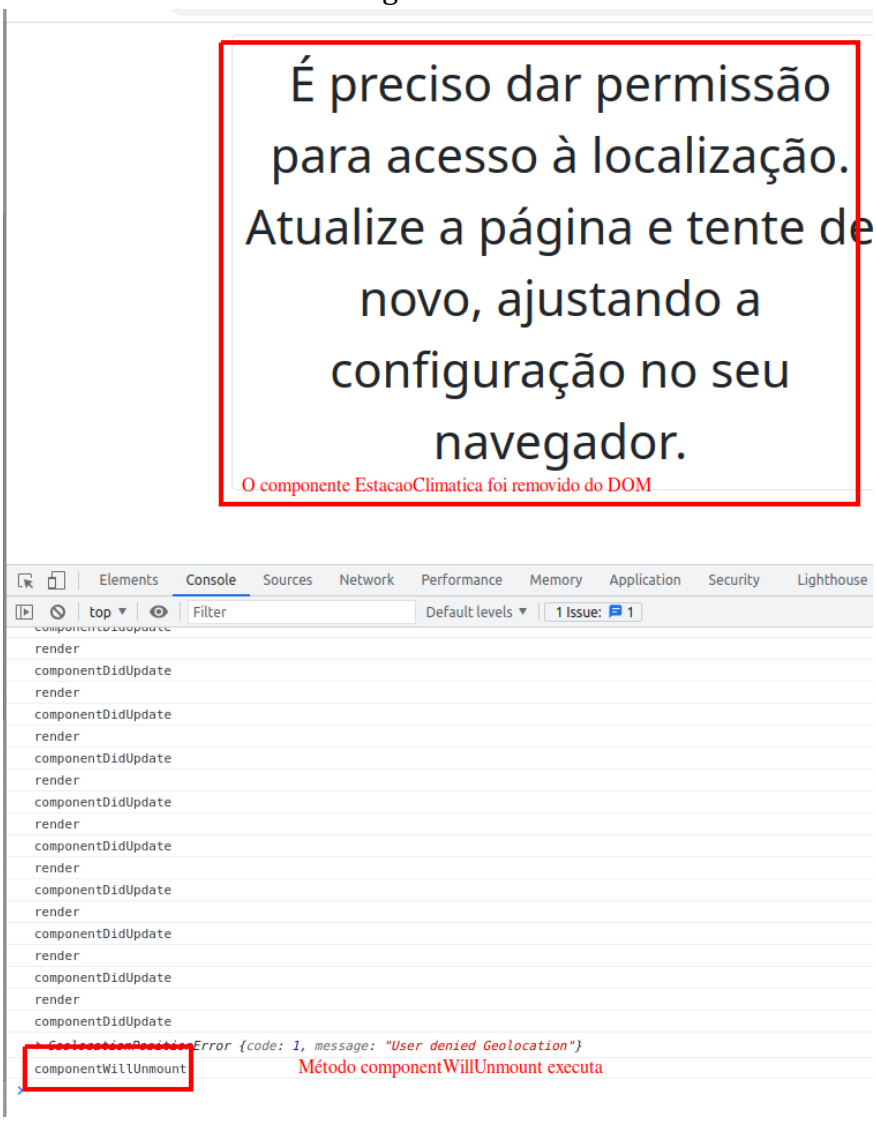
Atualize a página e observe o resultado no console, sem ainda clicar Allow ou Block. Como o usuário ainda não decidiu sobre a permissão, ainda não há mensagem de erro e, portanto, o componente EstacaoClimatico é adicionado ao DOM, o que faz com que seus métodos de ciclo de vida entrem em execução e o timer já comece a executar. Veja a Figura 2.7.3.

Figura 2.7.3


http://localhost:3000 wants to
Know your location
Ainda não clicamos Block ou Allow
Block
Allow
Clique no botão para saber a sua estação climática
Como não há mensagem de erro ainda, o componente foi adicionado ao DOM
Qual a minha estação?
Elements
Console
Sources
Network
Performance
Memory
Application
Security
Lighthouse
top
Filter
Default levels
1 Issue: 1
[HMR] Waiting for update signal from WDS...
render
componentDidMount
render
componentDidUpdate
render
componentDidUpdate
render
componentDidUpdate
render
componentDidUpdate
render
componentDidUpdate
render
componentDidUpdate
render
componentDidUpdate
Seus métodos de ciclo de vida entram em execução e o timer é iniciado. A cada segundo, seu estado é atualizado e, portanto, os métodos render e componentDidUpdate são chamados repetidamente.

Agora, clique no botão **Block**. Quando fizer isso, uma mensagem de erro será configurada no estado do componente principal. Com isso, ele será atualizado, o que inclui seus filhos. O componente EstacaoClimatica deixa de ser renderizado e, portanto, seu método **componentWillUnmount** entra em execução. É a chance que o React nos dá para liberar recursos alocados. Neste caso, desejamos interromper a execução do timer previamente configurado. Veja o resultado esperado na Figura 2.7.4.

Figura 2.7.4



Professor Rodrigo Bossini
https://www.rodrigobossini.com.br

Produção: agosto de 2021

2.8 (Exibindo um spinner) Quando a aplicação inicia, antes de o usuário decidir se deseja liberar ou bloquear o acesso à sua localização, pode ser de interesse exibir um componente que indica que o aplicativo está aguardando esta decisão. Poderíamos utilizar, por exemplo, um **Spinner** do Bootstrap. Veja a sua documentação no Link 2.8.1.

Link. 2.8.1
https://getbootstrap.com/docs/5.0/components/spinners/

Para definir o novo componente, clique com o direito na pasta **src** e crie um arquivo chamado **Loading.js**. A sua definição é um tanto simples. Veja o Bloco de Código 2.8.1.

Bloco de Código 2.8.1

```
import React, { Component } from 'react'

export default class Loading extends Component {
   render() {
      return (
         // centralizando nos dois eixos, borda e padding
         <div className="d-flex justify-content-center align-items-center  border rounded p-3">
            {/* text-whatever troca a cor */}
            <div className="spinner-border text-primary" style={{width: '3rem', height:
'3rem'}}>
               {/* deve ser utilizado somente por dispositivos de acessibilidade, como leitores de
tela  */}
               <span className="visually-hidden">Carregando...</span>
            </div>

         </div>
      )
   }
}
```

Para utilizar o novo componente, devemos ajustar o arquivo **index.js**. As verificações a serem realizadas são resumidas na Tabela 2.8.1.

Tabela 2.8.1

Verificação	Significado	Exibir
Ainda não há latitude e nem mensagem de erro	Usuário ainda não bloqueou e nem permitiu o acesso à sua localização	Componente Loading
Já existe mensagem de erro	Usuário bloqueou o acesso à sua localização	Texto com mensagem de erro
Já existe latitude	Usuário permitiu o acesso à sua localização	Componente EstacaoClimatica

O Bloco de Código 2.8.2 mostra o arquivo **index.js** atualizado.

Bloco de Código 2.8.2

```
import Loading from './Loading'
…
 render() {
        // console.log("render")
        return (
            // responsividade, margem acima
            <div className="container mt-2">
                {/* uma linha, conteúdo centralizado, display é flex */}
                <div className="row justify-content-center">
                {/* oito colunas das doze disponíveis serão usadas para telas médias em diante */}
                    <div className="col-md-8">
                        {
                          (!this.state.latitude && !this.state.mensagemDeErro) ?
                          <Loading />
                          :
                          this.state.mensagemDeErro ?
                             <p className="border rounded p-2 fs-1 text-center">
                                É preciso dar permissão para acesso à localização.
                                Atualize a página e tente de novo, ajustando a configuração
                                 no seu navegador.
                             </p>
                          :
                          <EstacaoClimatica
                                icone={this.state.icone}
                                estacao={this.state.estacao}
                                latitude={this.state.latitude}
                                longitude={this.state.longitude}
                                obterLocalizacao={this.obterLocalizacao}
                          />

                        }
                    </div>
                </div>
            </div>
        )
    }
```

2.9 (Valores padrão para props) Suponha que desejamos exibir um texto abaixo do componente Loading, explicando ao usuário que ele precisa responder ao pedido de acesso à sua localização. Poderíamos fazê-lo como mostra o Bloco de Código 2.9.1.

Bloco de Código 2.9.1

```
export default class Loading extends Component {
   render() {
      return (
         // centralizando nos dois eixos, borda e padding
         <div className="d-flex flex-column justify-content-center align-items-center  border
rounded p-3">
            {/* text-whatever troca a cor */}
            <div className="spinner-border text-primary" style={{width: '3rem', height:
'3rem'}}>
               {/* deve ser utilizado somente por dispositivos de acessibilidade, como leitores de
tela  */}
               <span className="visually-hidden">Carregando...</span>
            </div>


         <p className="text-primary">Por favor, responda à solicitação de localização</p>
         </div>
      )
   }
}
```

Entretanto, o componente Loading pode ser utilizado em outros contextos, que não necessariamente envolvem a localização do usuário. Talvez queiramos deixar em aberto a mensagem a ser exibida, permitindo que ela seja escolhida no contexto de uso do componente. Podemos fazê-lo facilmente via props. O Bloco de Código 2.9.2 ilustra o ajuste a ser feito no componente **Loading**.

Bloco de Código 2.9.2

```
export default class Loading extends Component {

   render() {
      return (
         // centralizando nos dois eixos, borda e padding
         <div className="d-flex flex-column justify-content-center align-items-center  border
rounded p-3">
            {/* text-whatever troca a cor */}
            <div className="spinner-border text-primary" style={{width: '3rem', height:
'3rem'}}>
               {/* deve ser utilizado somente por dispositivos de acessibilidade, como leitores de
tela  */}
               <span className="visually-hidden">Carregando...</span>
            </div>


          <p className="text-primary">{this.props.mensagem}</p>
         </div>
      )
   }
}
```

Cabe ao componente principal, definido no arquivo **App.js**, decidir qual texto será exibido. Veja o Bloco de Código 2.9.3.

Bloco de Código 2.9.3

```
...
<div className="col-md-8">
                    {
                      (!this.state.latitude && !this.state.mensagemDeErro)?
                      <Loading mensagem="Por favor, responda à solicitação de localização"/>
                      :
                      this.state.mensagemDeErro ?
                        <p className="border rounded p-2 fs-1 text-center">
                          É preciso dar permissão para acesso à localização. Atualize a página e
tente de novo, ajustando a configuração no seu navegador.
                        </p>
                      :
                      <EstacaoClimatica
                            icone={this.state.icone}
                            estacao={this.state.estacao}
                            latitude={this.state.latitude}
                            longitude={this.state.longitude}
                            obterLocalizacao={this.obterLocalizacao}
                      />

                    }
                </div>
...
```

Suponha que desejamos tornar o componente Loading um pouco mais flexível, no seguinte sentido. Ele já permite que seja especificada a mensagem a ser utilizada. Entretanto, pode ser de interesse que ele tenha uma mensagem mais genérica a ser exibida por padrão, talvez um simples “Carregando”. Poderíamos fazê-lo como destaca o Bloco de Código 2.9.4.

Bloco de Código 2.9.4

```
export default class Loading extends Component {

   render() {
      return (
         // centralizando nos dois eixos, borda e padding
         <div className="d-flex flex-column justify-content-center align-items-center  border
rounded p-3">
            {/* text-whatever troca a cor */}
            <div className="spinner-border text-primary" style={{width: '3rem', height:
'3rem'}}>
               {/* deve ser utilizado somente por dispositivos de acessibilidade, como leitores de
tela  */}
               <span className="visually-hidden">Carregando...</span>
            </div>


          <p className="text-primary">{this.props.mensagem || 'Carregando'}</p>
         </div>
      )
   }
}
```

O React possui, no entanto, um mecanismo próprio para isso. Seu uso torna o código mais limpo e elegante. Componentes React possuem uma propriedade chamada **defaultProps**. Basta associarmos a ela um objeto JSON que possui todos os props de interesse associados a seus valores padrão. Uma vez feita a sua definição, o teste explícito feito anteriormente já não é necessário. Veja o Bloco de Código 2.9.5.

Bloco de Código 2.9.5

```
import React, { Component } from 'react'

export default class Loading extends Component {

    render() {
        return (
            // centralizando nos dois eixos, borda e padding
            <div className="d-flex flex-column justify-content-center align-items-center  border
rounded p-3">
                {/* text-whatever troca a cor */}
                <div className="spinner-border text-primary" style={{width: '3rem', height:
'3rem'}}>
                    {/* deve ser utilizado somente por dispositivos de acessibilidade, como leitores de
tela  */}
                    <span className="visually-hidden">Carregando...</span>
                </div>


             <p className="text-primary">{this.props.mensagem}</p>
            </div>
        )
    }
}
//fora da classe que define o coponente
Loading.defaultProps = {
    mensagem: "Carregando"
}
```

No componente **App**, faça testes passando e deixando de passar o props mensagem, como destacam os blocos de código 2.9.6 e 2.9.7.

Bloco de Código 2.9.6

```
<div className="col-md-8">
    {
      (!this.state.latitude && !this.state.mensagemDeErro)?
          <Loading/>
      :
      this.state.mensagemDeErro ?
          <p className="border rounded p-2 fs-1 text-center">
            É preciso dar permissão para acesso à localização. Atualize a página e tente de
            novo, ajustando a configuração no seu navegador.
          </p>
      :
            <EstacaoClimatica
                icone={this.state.icone}
                estacao={this.state.estacao}
                latitude={this.state.latitude}
                longitude={this.state.longitude}
                obterLocalizacao={this.obterLocalizacao}
            />

      }
</div>
```

Bloco de Código 2.9.7

```
<div className="col-md-8">
    {
      (!this.state.latitude && !this.state.mensagemDeErro)?
          <Loading mensagem="Por favor, responda à solicitação de localização"/>
      :
      this.state.mensagemDeErro ?
          <p className="border rounded p-2 fs-1 text-center">
            É preciso dar permissão para acesso à localização. Atualize a página e tente de
            novo, ajustando a configuração no seu navegador.
          </p>
      :
            <EstacaoClimatica
                icone={this.state.icone}
                estacao={this.state.estacao}
                latitude={this.state.latitude}
                longitude={this.state.longitude}
                obterLocalizacao={this.obterLocalizacao}
            />

      }
</div>
```

Referências

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