



ReactJS

Prof. Joziel Júnior

Descrição

Biblioteca JavaScript para o desenvolvimento front-end de aplicações web e aplicativos para dispositivos móveis.

Propósito

Atualmente, com o constante crescimento das redes sociais, a biblioteca JavaScript se impõe como uma das principais bibliotecas para a criação de interface de usuário com grandes projetos, como, por exemplo, Facebook, Instagram, Netflix e WhatsApp, o que aponta uma grande demanda por profissionais nessa área de desenvolvimento.

Preparação

Para um melhor entendimento deste conteúdo, recomendamos ter conhecimento em JavaScript e HTML. O ambiente de desenvolvimento utilizado aqui é o Windows 10. Você deve instalar o Node.js e sempre utilizar a versão LTS, o gerenciador de pacotes NPM ou, de preferência, sendo os navegadores recomendados o Google Chrome e o Mozilla Firefox, com uma extensão React Developer Tools instalada, assim como o editor para os códigos-fonte Visual Studio Code (VSCode).

[Download](#) dos códigos dos exemplos desenvolvidos com o VSCode.

Objetivos

Módulo 1

Introdução e preparação do ambiente

Elaborar diferentes ambientes de desenvolvimento para o React.

Módulo 2

Conceitos básicos e sintaxe

Aplicar a sintaxe por meio da implementação de exemplos.

Módulo 3

Usando React com requisições HTTP/Ajax & React Hooks

Usar requisições HTTP/Ajax com o novo conceito de React Hooks.

Módulo 4

Usando rotas e Redux

Traçar rotas com React Router e o controle da camada de negócio com React Redux.

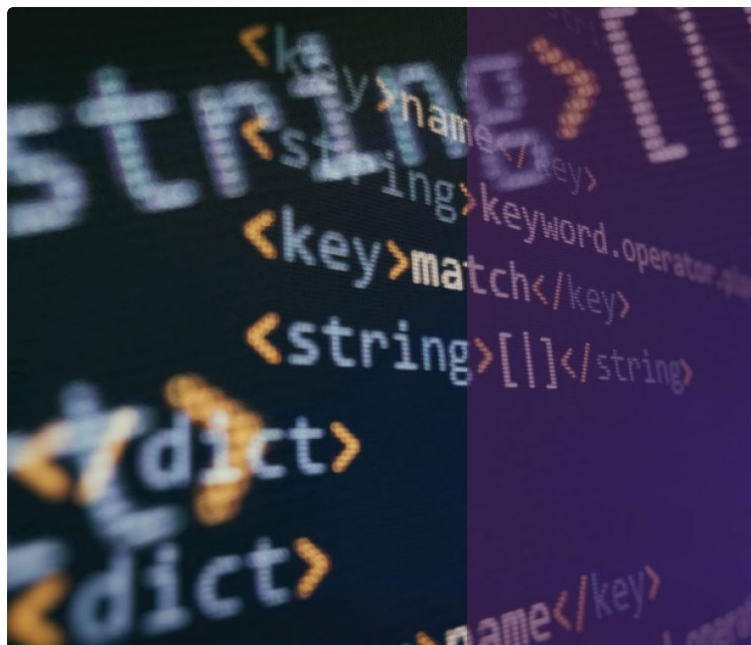


Introdução

Neste estudo, faremos uma breve introdução do React, desde os principais problemas que existiam nas aplicações web e mobile até o desafio que seus criadores tiveram ao idealizar essa biblioteca. Em seguida, abordaremos assuntos referentes ao chamado “ecossistema” para desenvolver aplicações em React, com uma pequena revisão necessária para um melhor entendimento dessa biblioteca. Para isso, explicaremos conceitos componentes, propriedades (props) e estado (state), preparando, por fim, nosso ambiente de desenvolvimento.

Também aprenderemos a realizar requisições HTTP em APIs web, consumindo dados de uma API de forma nativa mediante a utilização de Ajax. Os dados recebidos são tratados com Hooks, lidando com os possíveis efeitos colaterais que as requisições assíncronas podem trazer à aplicação.

Por fim, apresentaremos os conceitos de roteamento, que é a maneira utilizada pelo React tanto para manipular interfaces e componentes por meio de rotas quanto para gerenciar melhor a camada de negócio, de forma centralizada, com o React Redux.



1 - Introdução e preparação do ambiente

Ao final deste módulo, você será capaz de elaborar diferentes ambientes de desenvolvimento para o React.

Modelo baseado em componentes

Utilizando a própria documentação, o React é uma biblioteca JavaScript para criar interfaces de usuário. Desenvolvida pelo Facebook, ela possui código aberto e é mantida por uma imensa comunidade de desenvolvedores individuais, sendo utilizada por grandes empresas, como, por exemplo, Netflix, Instagram, Tesla, Walmart e Airbnb.

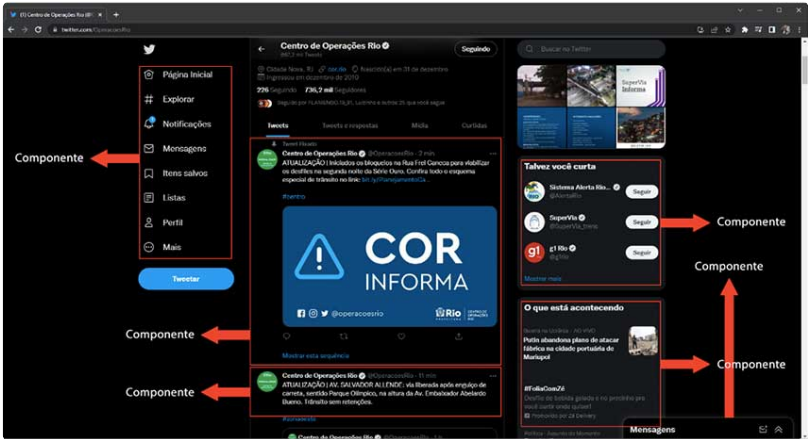
A manutenção de atualizações da biblioteca facilita a manipulação do front-end em páginas web e aplicativos móveis via React Native. O React trabalha em um modelo que podemos chamar de composicional, ou seja, o programador precisa enxergar seu ambiente de trabalho, o aplicativo, a tela ou a página segmentada em diferentes pedaços, que serão chamados de componentes.

Na imagem a seguir, observe o exemplo de um menu principal de um aplicativo ou rodapé, um cabeçalho e até uma área de propagandas e anúncios de uma página HTML.



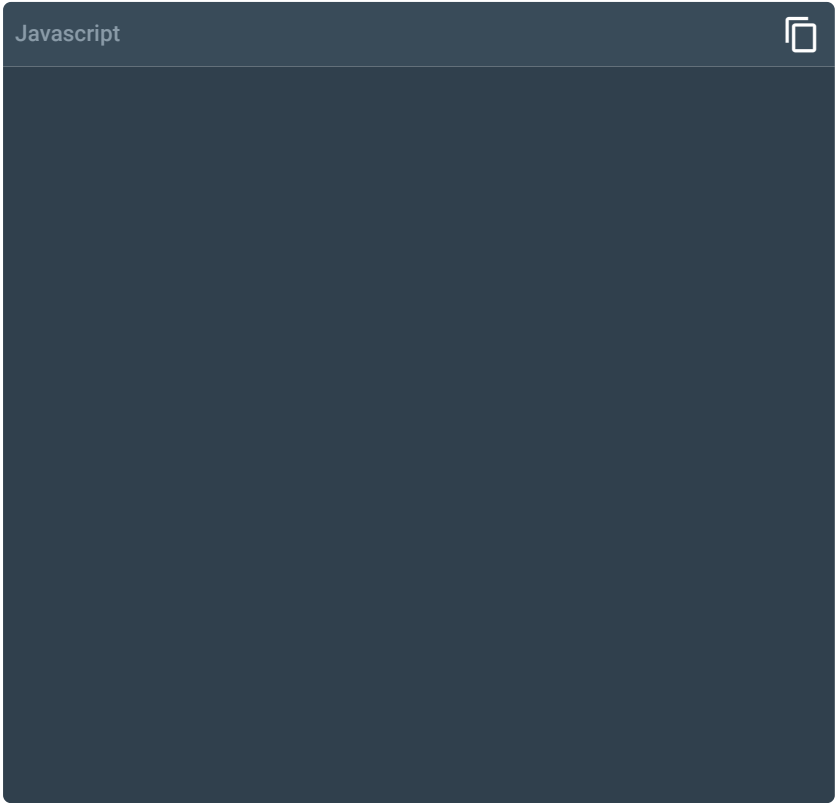
Exemplo de componentes em uma página ou aplicativo.

A seguir, observe o seguinte exemplo:



Componentes em uma página ou aplicativo (Twitter).

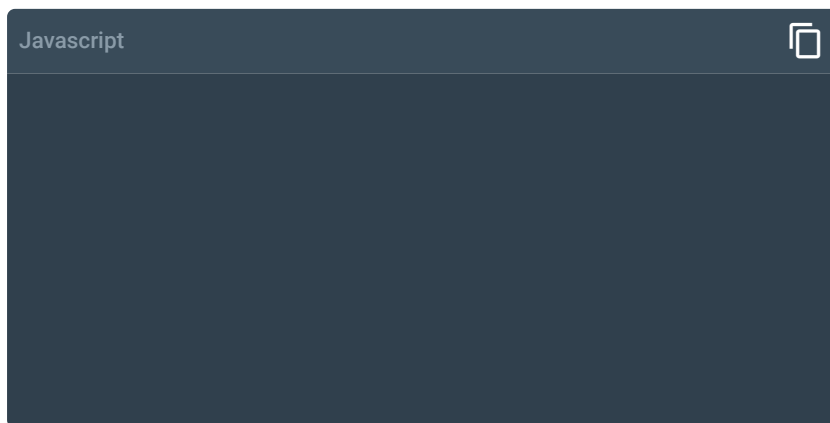
Uma forma simples de representar os componentes seria a utilização de funções em JavaScript. Veja, no exemplo a seguir, duas funções isoladas e uma terceira reutilizando as anteriores:



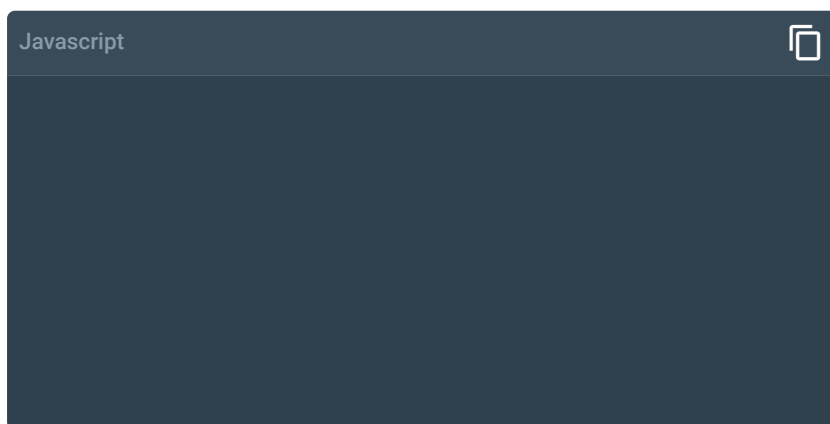
Natureza declarativa

Outra característica do React é sua natureza declarativa, a qual, diferentemente da forma imperativa, é conhecida como procedural. Na natureza procedural, o programador precisa criar toda a lógica para tratar os dados, como, por exemplo, percorrer uma lista ou vetores de dados e manipulá-los para a criação de uma nova lista.

Além disso, o React não é recomendado em aplicações que precisem de manutenção ou mudanças frequentes, tornando o código com alto grau de complexidade. Observe um código escrito em JavaScript na forma procedural:



Na programação declarativa, existe um alto nível de abstração. Um exemplo seria a linguagem SQL, na qual, em uma consulta a um banco de dados, o resultado esperado é declarado sem se preocupar com o resultado em si. Em JavaScript, diferentes funções facilitam esse processo, como, por exemplo, percorrer uma lista, tornando o código mais limpo, previsível e simples de depurar. A título de ilustração, segue o trecho do mesmo código escrito anteriormente no formato declarativo:



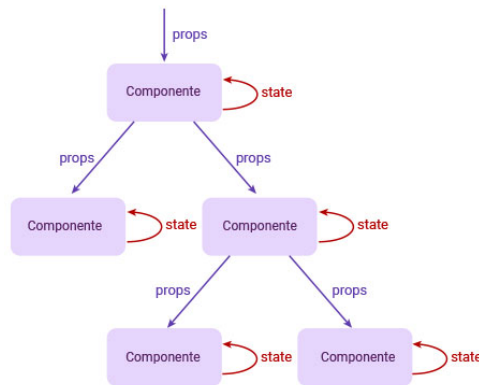
Vinculação unidirecional

O React segue uma forma de fluxo de dados unidirecional também conhecido como vinculação unidirecional. Em geral, esse conceito significa que os dados possuem uma e apenas uma maneira de serem passados para outra parte da sua aplicação.

Reflexão

Esse fluxo geralmente obedece à forma de um componente pai para o seu componente filho. Com isso, o filho consegue apenas ler dados do pai sem permitir uma alteração dessa informação no componente pai.

Os dados armazenados no componente pai são chamados de estado (states); o valor passado para o componente filho, de propriedades (props). Caso o filho precise atualizar alguma informação no pai, serão utilizados ações ou eventos, como, por exemplo, ao clicar em um botão.



Fluxo de dados unidirecional.

Vamos resumir agora o fluxo de dados unidirecional:

- Dados sempre de cima para baixo;
- Ações ou eventos no sentido de baixo para cima;
- As propriedades (props) são passadas apenas de pais para filhos e não podem ser alteradas;
- O estado (state) mantém o estado interno de cada componente e pode ser alterado pelos filhos.

JSX é obrigatório?

O que seria JSX? Uma nova linguagem? Basicamente, trata-se de uma extensão de sintaxe JavaScript cujo emprego não é obrigatório na codificação. No entanto, a maioria dos programadores acha prática sua utilização em trabalhos com uma interface dentro do código JavaScript, permitindo ao React mostrar mensagens mais úteis de aviso e erro.

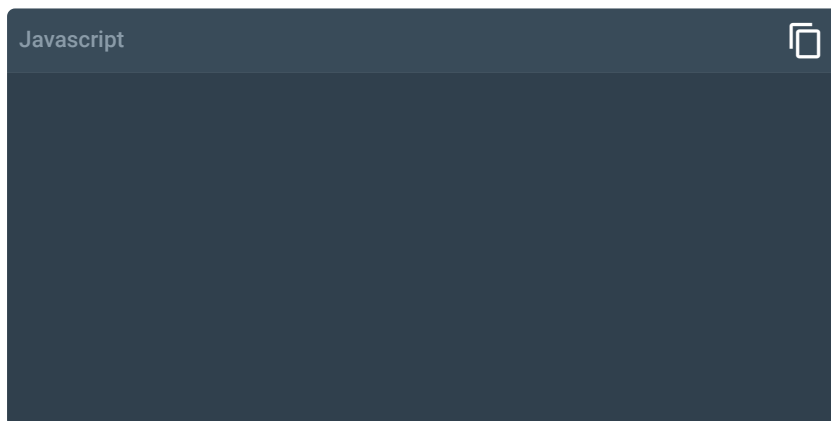
Um exemplo de como o JSX facilita nossa codificação pode ser visto no código adiante. Entretanto, observe primeiramente um código sem a utilização de JSX:

Javascript





Agora veja o mesmo código reescrito com JSX:



O JSX consegue produzir elementos do React, podendo causar uma certa estranheza, porém, sua utilização será demonstrada neste material em diversas oportunidades. Com isso, comprovaremos sua facilidade e o quanto ganhamos na manipulação das interfaces.

Comentário

Vimos até aqui as diferentes características do React. Mas o que o faz ser tão diferente de outras linguagens? Por que utilizá-lo?

Além de oferecer a facilidade de utilizar o JavaScript, atualmente muitos programadores estão migrando para o TypeScript, o que revela a flexibilidade dessa biblioteca. Outro ponto é o mercado de trabalho: em uma rápida pesquisa, podemos encontrar oportunidades incríveis com diferentes níveis de experiência, incluindo o trabalho remoto.

Somado à facilidade de escrita e compreensão (basicamente, HTML e JavaScript), assim como ao entendimento dos conceitos de componentes, propriedades e estados, o tempo para se dominar a biblioteca se torna bem curto. Por fim, graças ao Virtual [DOM](#), o React consegue realizar atualizações muito mais rápidas que a execução dessas mudanças no DOM real.

Essa otimização o torna muito mais eficiente, evitando problemas relacionados à complexidade dos processos pelos quais as páginas reais geralmente são atualizadas.

DOM

O Document Object Model (DOM) é uma interface multiplataforma e independente de linguagem, que trata um documento XML ou HTML como uma estrutura de árvore na qual cada nó é um objeto que representa uma parte do documento.

Ambiente para testes

CodePen

Para iniciarmos nossos estudos em React, vamos compartilhar duas ferramentas de edição on-line para testar nossos códigos:

- [CodePen](#).
- [CodeSandBox](#).

Ambas são poderosas ferramentas, possuindo suas peculiaridades. As duas requerem a criação de uma conta para o melhor aproveitamento, permitindo salvar o progresso de seus códigos. Por conta disso, falaremos sobre elas em detalhes nesta seção.

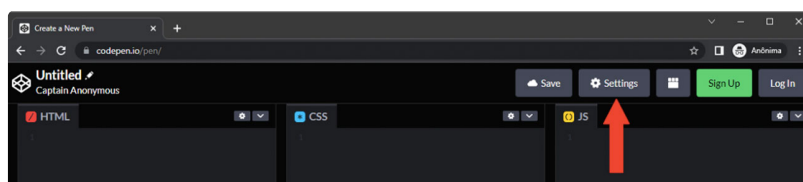
O CodePen apresenta uma interface mais simples e algumas limitações na licença gratuita disponível. Ele, por exemplo, não permite a criação de arquivos separados para importar e exportar nossos componentes. Além disso, algumas configurações são menos intuitivas na comparação com o CodeSandBox.

Segue um breve passo a passo de como deixar nosso ambiente pronto para pequenos códigos React:

Inicialmente, configuraremos nosso ambiente clicando em Start Coding:

Página inicial do CodePen.

Acesse o painel de configurações clicando em Settings:



Acessando painel de configuração.

Em seguida, na aba JS (1), selecione, em JavaScript Preprocessor, a opção Babel (2). Em seguida, em Add External Scripts/Pens, adicione os links externos respectivamente representados pelos números (3) e (4) e confirme clicando em Close (5), como a imagem apresenta:

Acessando painel de configuração.

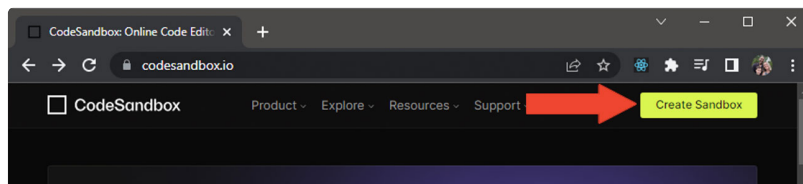
A partir de agora, temos um ambiente para testar nossos códigos em React.

CodeSandBox

Ferramenta muito parecida com a CodePen, a grande diferença do CodeSandBox é sua capacidade, mesmo com plano gratuito, de permitir ao usuário criar uma estrutura de pastas e arquivos e, com isso, manipular seus componentes. Além disso, ele já possui templates para React integrados, não sendo necessário adicionar scripts externos.

Observe um passo a passo de como deixar esse ambiente pronto para nosso conteúdo:

Inicialmente, clique em Create Sandbox:



Página inicial do CodeSandBox.

Em seguida, escolha o template React:

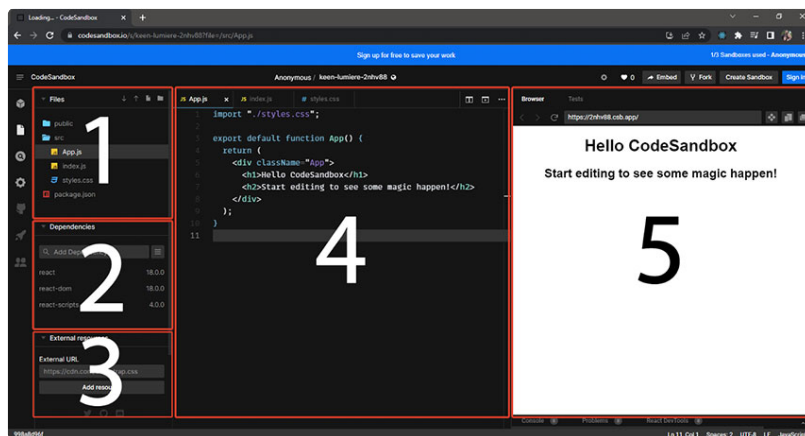
Escolhendo nosso ambiente desejado: React.

Nosso ambiente já foi criado. Diferentemente do CodePen, aqui temos:

- Um painel de estrutura de arquivos (1);
- Dependências utilizadas que podem ser alteradas de acordo com a necessidade de alguma versão específica (2);
- Links externos como um repositório e a utilização de Google Fonts (3);
- O painel de edição do arquivo selecionado (4);

- A área do resultado final, um navegador que exibirá nosso código compilado (5).

Verifique esta imagem para observar mais detalhes:



Ambiente CodeSandbox.

Sinta-se à vontade para escolher o melhor ambiente de desenvolvimento. Escrevemos e disponibilizamos a maior parte do nosso conteúdo no CodePen. Ao longo deste estudo, iniciaremos a configuração do Visual Studio Code com os códigos-fonte dos exemplos aqui publicados.

Ferramentas para desenvolvimento

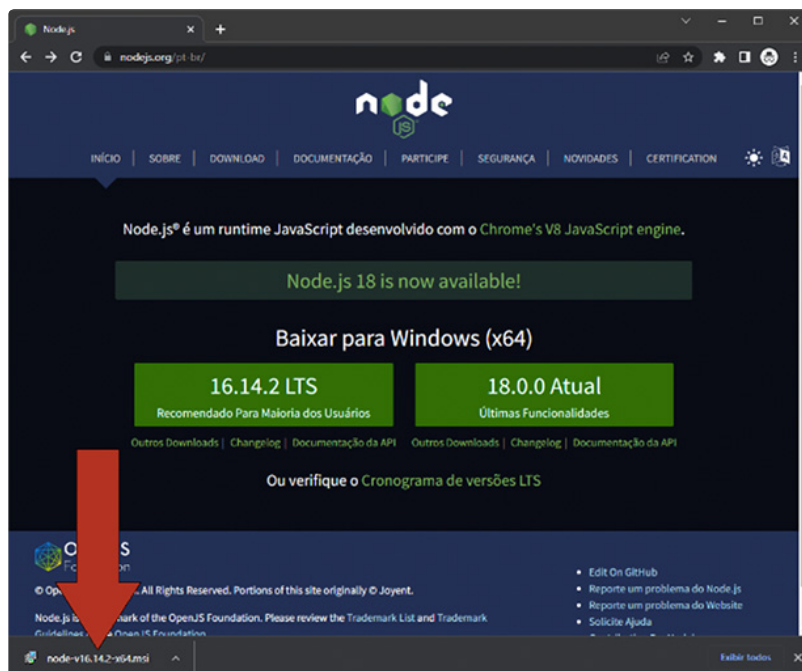
Node.js

Apresentaremos agora alguns softwares necessários para a compilação de nossos códigos React e algumas ferramentas que aceleram a codificação, a depuração e a análise das nossas aplicações. De início, saiba que a instalação do Node.js é obrigatória, mas a da extensão de navegador e do editor é opcional.

Seguindo a documentação na página oficial, o Node.js é um runtime JavaScript de código aberto e projetado para o desenvolvimento de aplicações escaláveis de rede. O intuito desta seção é explicar como se deve instalar a versão para o nosso ambiente de desenvolvimento.

Página oficial do Node.js para realizar o Download.

Atenção: lembre-se de sempre baixar a versão LTS; neste conteúdo, utilizaremos a versão 16.14.2 LTS.



Download concluído.

Vamos prosseguir agora com a instalação, atentando para as opções selecionadas nestas imagens:

Tela de instalação do Node.js.

Tela de instalação do Node.js. Escolha o local de instalação (em nosso caso, "c:\Program Files\nodejs\").

Tela de configuração do cliente.

Tela de instalação do Node.js. Confirme as instalações de ferramentas necessárias.

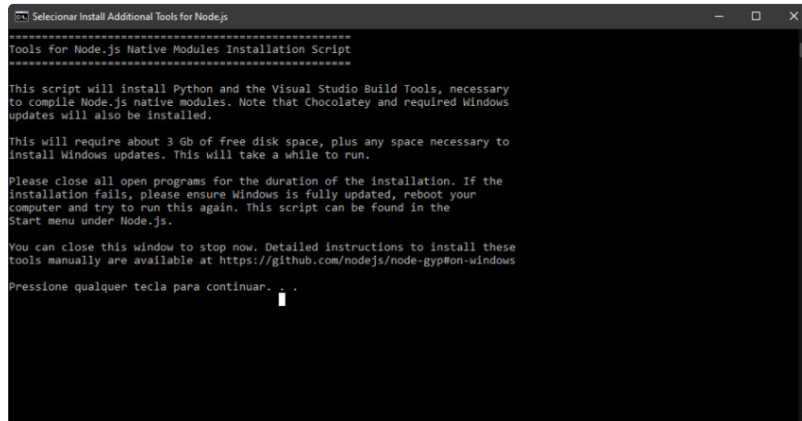
Tela de instalação do Node.js: instalação concluída.

Tela de instalação do Node.js. Confirme algumas ferramentas necessárias; para cada ferramenta, é preciso realizar uma confirmação

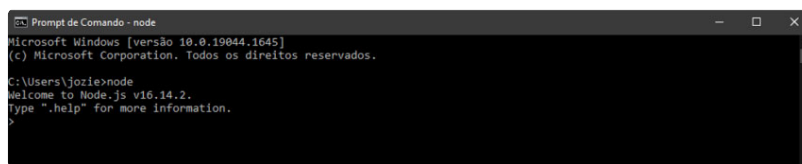
no prompt.

Na sequência, são geradas as telas de instalação dos pacotes necessários para o Node.js. Após a finalização, digite enter para concluir.

Para confirmar se a instalação foi instalada corretamente, digite “node” no prompt de comandos do Windows. A versão será exibida (v16.14.2, em nosso caso) deste modo:



Para continuar a configuração siga a sequência:

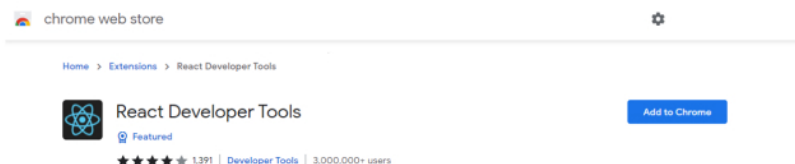


Tela de instalação do Node.js confirmando a instalação da versão 16.14.2.

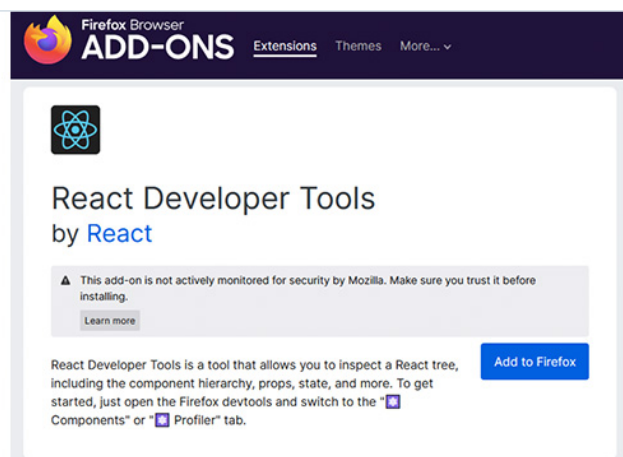
Extensão no navegador (opcional)

Algumas ferramentas podem ser de grande ajuda no desenvolvimento de nosso material: uma delas é a extensão React Devtools, disponível para Google Chrome e Mozilla Firefox. Não pretendemos detalhar o seu uso, mas, ainda assim, sugerimos a leitura de sua documentação para um melhor entendimento desse assunto.

O React Devtools permite ao programador inspecionar uma árvore de componentes React no navegador, verificando as propriedades (props) e o estado (state) de seus componentes. Para isso, instale a extensão no Google Chrome via Chrome Web Store (ou no Mozilla Firefox por meio do Firefox Browser Add-ons).

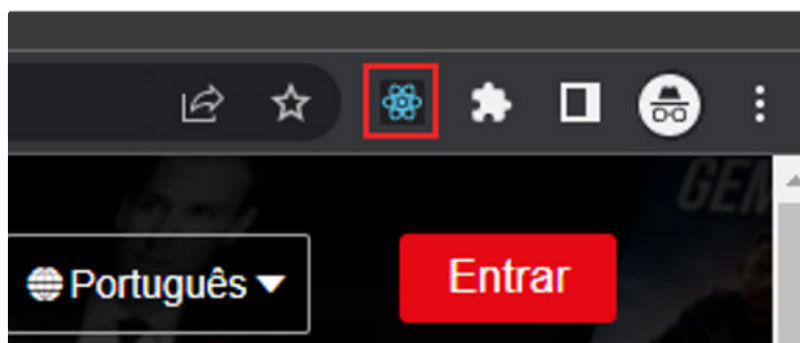


Instalando a extensão React Developer Tools no Google Chrome.



Instalando a extensão React Developer Tools no Mozilla Firefox.

Logo que a instalação for concluída, atualize a página. Caso o ícone fique azul, isso significa que se trata de uma página criada com React.



Extensão React Developer Tools instalada no Google Chrome.

Ao inspecionar a página do Netflix, verifica-se que duas novas abas foram adicionadas: Components e Profiler.



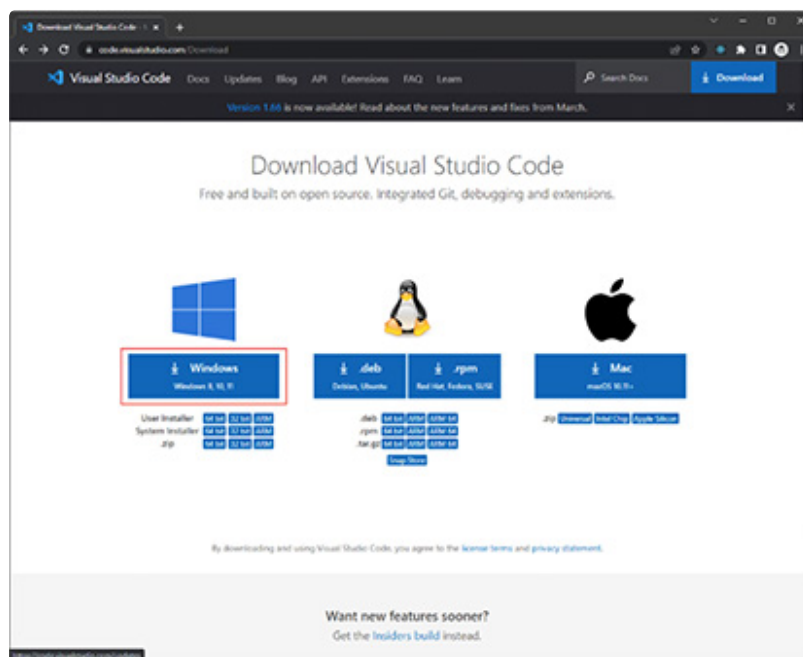
Novas abas ao inspecionar a página do Netflix no Google Chrome.

Editor Visual Studio Code

O Visual Studio Code é um editor gratuito de códigos desenvolvido pela Microsoft que possui versões para Windows, Linux e MacOS.

Atualmente, ele suporta quase todas as principais linguagens, como, por exemplo, HTML, C# e Rubi.

Vamos utilizá-lo neste material, mas fique à vontade para usar qualquer outro editor de sua preferência! **Faremos agora um breve passo a passo de sua instalação:**



Download na página oficial do VSCode.

Download concluído (versão Windows).

A seguir, observe o passo a passo de instalação do VSCode.

Tela do processo de instalação do VSCode.

Tela do processo de instalação do VSCode. Recomendamos marcar, pelo menos, a última opção.

Tela do processo de instalação do VSCode confirmando as opções.

Tela do processo de instalação do VSCode em instalação.

Tela do processo de instalação do VSCode com a instalação concluída.

Tela inicial do VSCode.

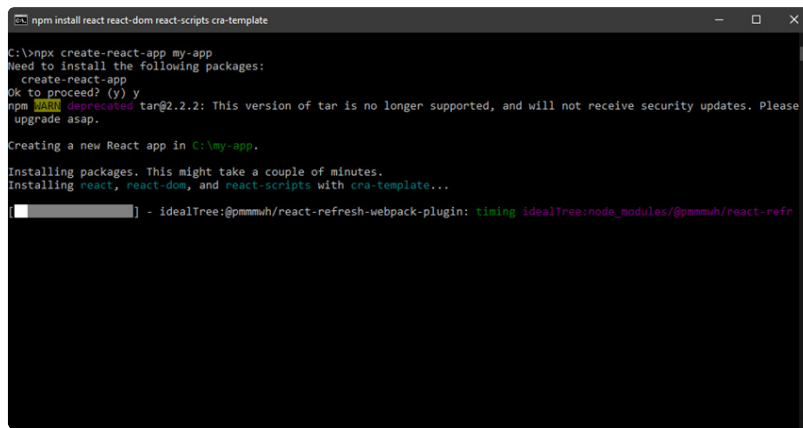
Primeiro App React

Create React App

Após a instalação do Node.js, em que foram instaladas todas as dependências necessárias para iniciar o desenvolvimento de nossos projetos, vamos criar nossa primeira tela com o React utilizando uma ferramenta que facilita a estrutura inicial do projeto, configurando nosso ambiente com funções mais recentes do JavaScript e otimizando nosso aplicativo para produção.

Criaremos uma pasta para organizar nossos aplicativos React e ao abrir nosso terminal, no modo administrador, executar o comando e confirmar todo o processo, conforme imagem a seguir:

```
npx create-react-app my-app
```

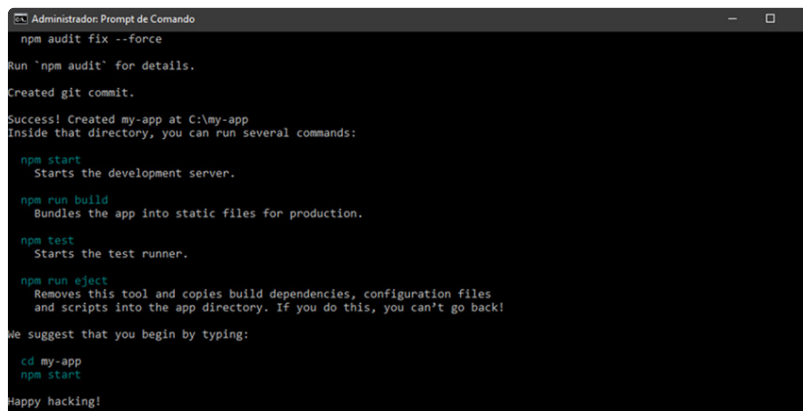
```
npm install react-dom react-scripts cra-template

C:\>npx create-react-app my-app
Need to install the following packages:
  create-react-app
ok to proceed? (y) y
npm WARN deprecated tar@2.2.2: This version of tar is no longer supported, and will not receive security updates. Please
upgrade asap.

Creating a new React app in C:\my-app.
Installing packages. This might take a couple of minutes.
Installing react, react-dom, and react-scripts with cra-template...
[Progress bar] - idealTree:@pmmmh/react-refresh-webpack-plugin: timing idealTree:node_modules/@pmmmh/react-refr
```

Instalando o pacote create-react-app e criando nosso primeiro projeto React.

A seguir, as instruções:



```
Administrador: Prompt de Comando

npm audit fix --force

Run 'npm audit' for details.

Created git commit.

Success! Created my-app at C:\my-app
Inside that directory, you can run several commands:

  npm start
    Starts the development server.

  npm run build
    Bundles the app into static files for production.

  npm test
    Starts the test runner.

  npm run eject
    Removes this tool and copies build dependencies, configuration files
    and scripts into the app directory. If you do this, you can't go back!

We suggest that you begin by typing:

  cd my-app
  npm start

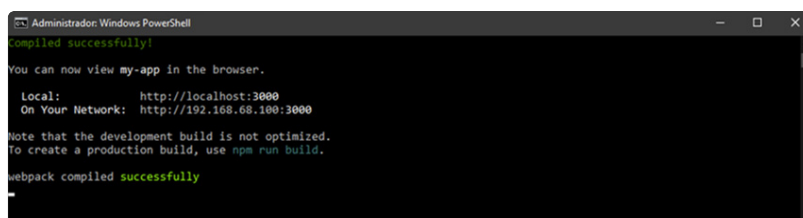
Happy hacking!
```

Após a instalação concluída, vamos seguir as instruções.

Para iniciar nosso servidor React, digite o seguinte comando: `npm start`

Informações sobre a tabela. Imagem: Executando `npm start`.

Para as configurações anteriores temos o seguinte resultado:



```
Administrador: Windows PowerShell

Compiled successfully!

You can now view my-app in the browser.

  Local:            http://localhost:3000
  On Your Network:  http://192.168.68.100:3000

Note that the development build is not optimized.
To create a production build, use npm run build.

webpack compiled successfully
```

Compilação concluída.

Na imagem a seguir temos a primeira execução de uma aplicação React.

Nossa primeira aplicação React em execução.

Se você conseguiu chegar até aqui sem erros, parabéns! Caso tenha tido algum problema, revise os itens anteriores e verifique se o processo de

instalação apresentou alguma falha, refazendo todo o procedimento de acordo com o que mencionamos.

Comentário

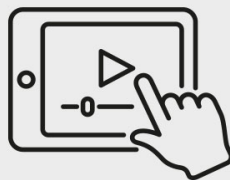
Mas não se preocupe: durante este estudo, vamos nos aprofundar em conceitos básicos que são importantes para o melhor entendimento dessa ferramenta.



Criação de um App React

Neste vídeo, apresentaremos os principais conceitos que envolvem a criação do App React.

Para assistir a um vídeo sobre o assunto, acesse a versão online deste conteúdo.



Falta pouco para atingir seus objetivos.

Vamos praticar alguns conceitos?

Questão 1

Biblioteca JavaScript usada para criação de interface de usuário, o React tem como uma de suas principais características sua natureza declarativa. Qual dos exemplos a seguir não pode ser considerado uma programação declarativa?

A

HTML

B

SQL

C

XML

D

PHP

E

XSLT

Parabéns! A alternativa D está correta.

A linguagem declarativa é baseada no aspecto lógico e funcional: ela descreve o que se faz, e não exatamente como suas instruções

funcionam. Já a linguagem procedural é focada na mudança de estado de variáveis, o que acontece em PHP.

Questão 2

O React utiliza um fluxo de dados unidirecional. Em relação às vantagens e ao fluxo unidirecional, qual das alternativas a seguir não está correta?

A

O fluxo de dados unidirecional torna a depuração muito mais fácil.

B

Ter o fluxo de dados em uma única direção torna o programa menos propenso a erros e dá ao desenvolvedor mais controle.

C

Quando o programador conhece a trajetória dos dados, de onde eles estão vindo e aonde estão indo, pode-se realizar uma simulação para encontrar os problemas com mais eficiência.

D

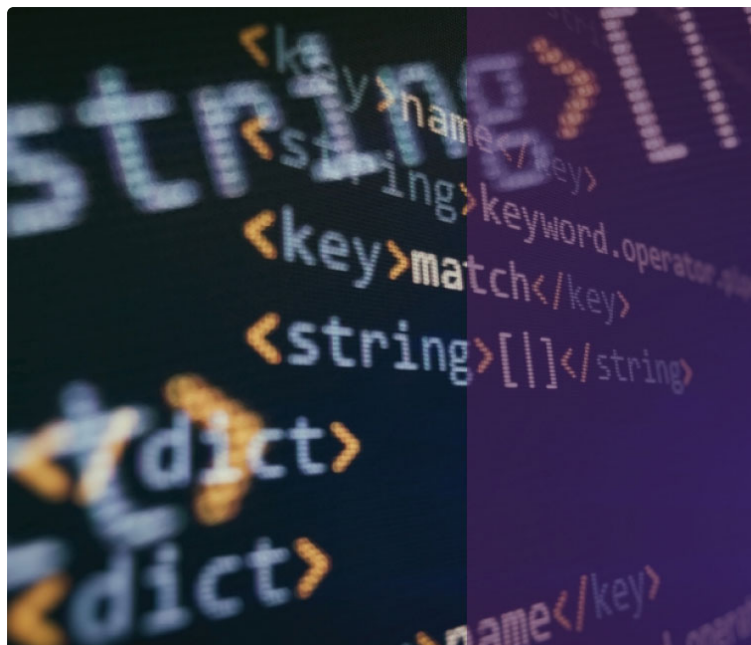
No React, cada estado é de propriedade de um componente.

E

As alterações feitas no estado do componente afetarão todos os componentes ligados a ele.

Parabéns! A alternativa E está correta.

Por padrão, as alterações nos estados afetam apenas os filhos daquele componente; portanto, os componentes hierarquicamente acima (pais) e lateralmente (irmãos) não serão afetados. Caso queira que esse efeito ocorra, é preciso efetuar a elevação do estado.



2 - Conceitos básicos e sintaxe

Ao final deste módulo, você será capaz de aplicar a sintaxe por meio da implementação de exemplos.

Conceitos básicos

Hello, world!

Neste estudo, abordaremos os principais conceitos básicos para se criar o famoso “Hello, world!”. Nossa proposta é:

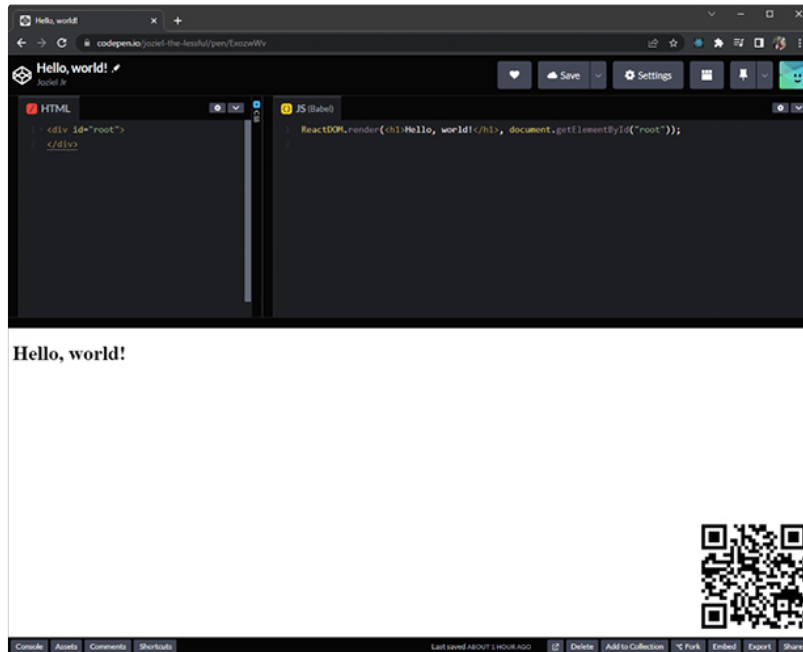
- Buscar a familiarização com o JSX;
- Ver o processo de renderização de elementos;
- Estudar os componentes, as propriedades, os estados e o ciclo de vida de um componente.

Também é importante aprender como utilizar binds e de que modo se consegue manipular eventos, listas, chaves e alguns exemplos de formulários.

Como tradição, todo programador sempre busca como criar o “Hello, world!” na linguagem nova que ele está estudando. Dizem que, se não o fizer, seu estudo ficará comprometido e ele não conseguirá aprender. Brincadeiras à parte, veja agora como é extremamente fácil gerar nossa primeira marcação HTML:

```
ReactDOM.render(<h1>Hello, world!</h1>,  
document.getElementById("root"));
```

Utilizando nosso ambiente de testes visto anteriormente, teste o código e veja o resultado. Acesse o código on-line no site <http://tiny.cc/aizquz> ou utilize o QR Code em seu celular (localizado no canto inferior direito) para acompanhar:



Hello, world!

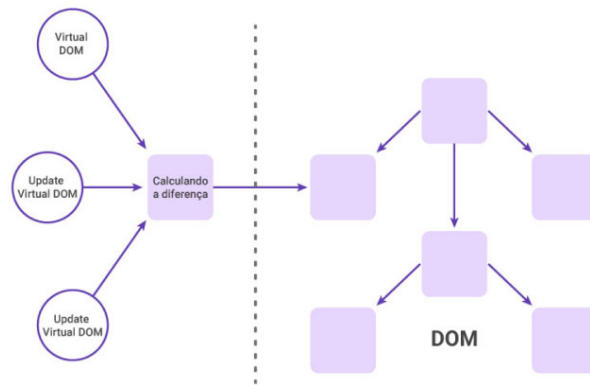
Temos aqui nosso primeiro método React, o `render()`, que possui a seguinte sintaxe:

```
ReactDOM.render(element, container[, callback]);
```

Sua função é renderizar um elemento do React no DOM, no contêiner passado por parâmetro, e retornar uma referência ao componente, podendo haver um retorno nulo caso o componente não possua estado. A atualização no DOM passa por um processo de comparação, refletindo o elemento do React mais recente.

Se a callback opcional for fornecida, ela será executada depois de o componente ter sido renderizado ou atualizado. Esse algoritmo de comparação é chamado de Diffing.

Trata-se da forma que os desenvolvedores do React encontraram para resolver um problema: manipular o DOM. Uma cópia do DOM é criada em memória (Virtual DOM) e mantida sempre atualizada com o DOM real, embora essa atualização passe por um processo denominado reconciliação.



Aplicando o processo de reconciliação.

Caso os novos dados sejam iguais aos anteriores, o Virtual DOM vai comparar as estruturas anteriores e confirmar que não houve alteração, evitando um novo pedido de renderização no DOM do navegador. Esse processo melhora a performance e o desempenho do navegador.

Sempre que houvesse uma atualização no DOM, ou seja, uma mudança na árvore, o React precisaria descobrir como atualizá-la de forma mais eficiente. As melhores soluções possuíam uma complexidade de ordem de $O(n^3)$, em que n é o número de elementos dessa árvore.

Exemplo

Uma árvore com 1.000 elementos exigiria uma ordem de um bilhão de comparações. A utilização do algoritmo Diffing consegue reduzir essa ordem para uma de ordem $O(n)$.

Para um estudo mais aprofundado desse algoritmo, sugerimos a leitura da documentação no site oficial da React.

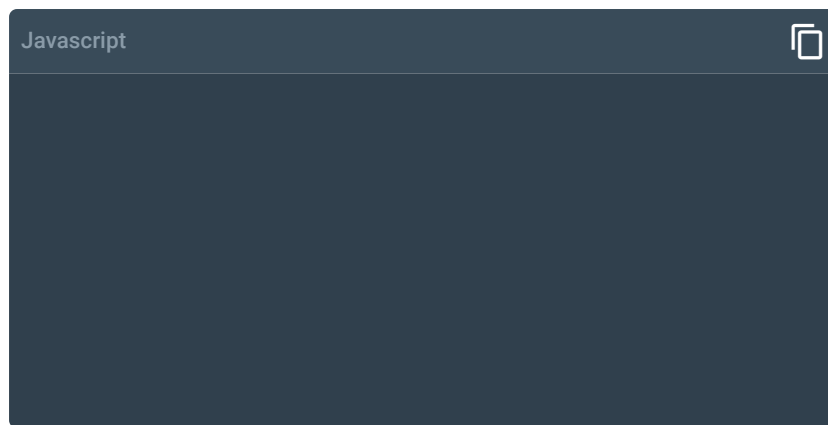
JSX

Vimos anteriormente uma prévia da utilização de JSX. Neste conteúdo, daremos detalhes sobre suas principais características, iniciando com a sintaxe de uma declaração de variável:

```
const element = <h1>Hello, world!</h1>;
```

Você pode estar se perguntando: essa sintaxe de tags seria HTML? Uma string? Trata-se, na verdade, de uma extensão de sintaxe JavaScript.

Vimos anteriormente como é possível tornar o código bem mais simples e fácil de entender. Este exemplo declara duas variáveis e usa o método `render()` novamente:



Veja agora o endereço <http://tiny.cc/njzquz> do código on-line e a imagem do resultado obtido:

Primeira imagem de variáveis em JSX.

Podemos usar qualquer expressão em JavaScript dentro das chaves em JSX. Se quiséssemos usar uma expressão matemática, como o valor de PI, por exemplo, junto a uma função que retorne um campo de outra variável, teríamos o resultado gerado na imagem a seguir e disponível no endereço <http://tiny.cc/1kzquz>.

```
HTML
<div id="root">
  </div>

JS
const user = {
  firstName: "João",
  lastName: "Roberto"
};

function formatName(user) {
  return user.firstName + " " + user.lastName;
}

const element = (
  <h1>
    Olá, {formatName(user)}, o valor de PI é {Math.PI}
  </h1>
);

ReactDOM.render(element, document.getElementById("root"));
```

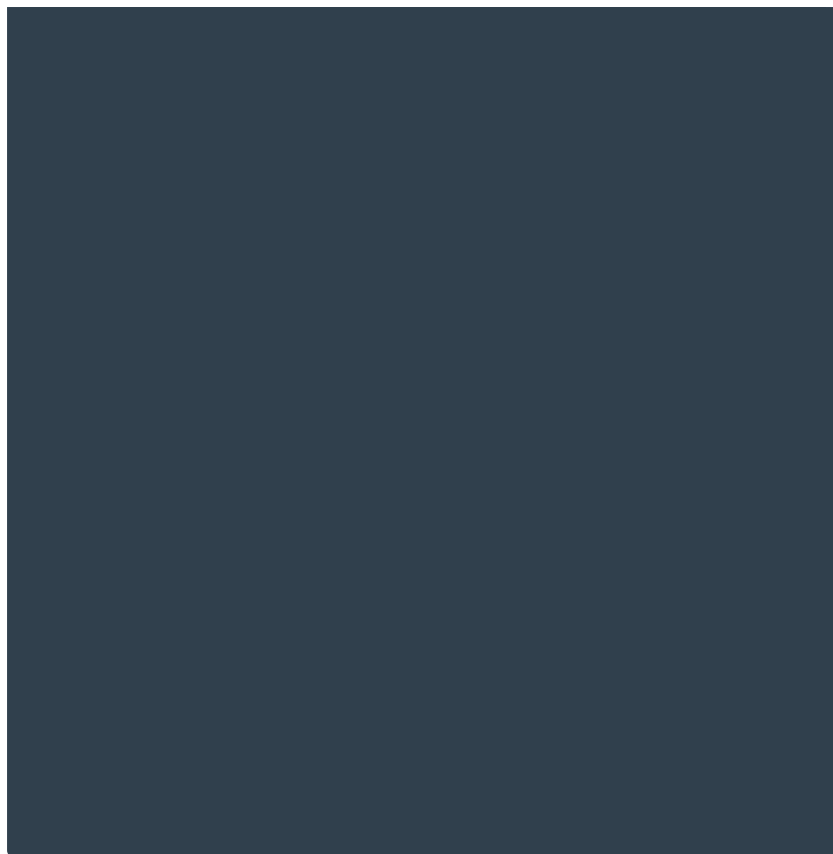
Olá, João Roberto, o valor de PI é 3.141592653589793



Segunda imagem de variáveis em JSX.

Também podemos usar JSX como expressões. Sempre depois de suas compilações, os valores ficarão disponíveis para serem utilizados em um laço condicional ou de repetição (<http://tiny.cc/hkzquz>):





Veja na imagem a seguir o resultado do código apresentado.

```
const user = {
  firstName: "João",
  lastName: "Roberto"
};

function formatName(user) {
  return user.firstName + " " + user.lastName;
}

function getGreeting(user) {
  if (user) {
    return (
      <h1>
        Olá, {formatName(user)}, o valor de PI é {Math.PI}
      </h1>
    );
  }
  return <h1>Olá, desconhecido, qual seu nome?</h1>;
}

ReactDOM.render(getGreeting(), document.getElementById("root"));
```

Olá, desconhecido, qual seu nome?

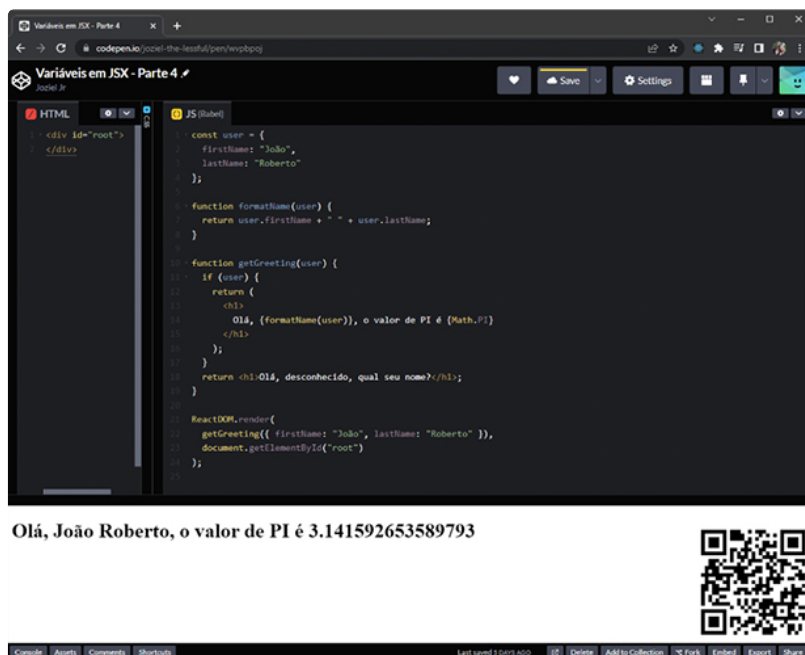
Terceira imagem de variáveis em JSX.

Como não passamos nenhum parâmetro na função `getGreeting()`, a própria função retornou com o elemento da linha 18. Mas como podemos fazer para entrar na linha 14, isto é, na condicional verdadeira que criamos?

É simples. basta passar um objeto do tipo usuário nessa função com esta sintaxe:

```
getGreeting({ firstName: "João", lastName: "Roberto" })
```

O resultado pode ser visto a seguir (<http://tiny.cc/qkzquz>):



Quarta imagem de variáveis em JSX.

Atributos em JSX devem seguir algumas regras, como, por exemplo, ao especificar strings literais como atributos:

```
const element = <a href="https://www.ensineme.com.br/img/logo.png">
  link </a>;
```

Também é possível usar chaves, incorporando expressões JavaScript em um atributo: `const element = ;`

Dica

Não utilize aspas e chaves no mesmo atributo: use sempre aspas para valores em string e chaves para expressões.

Por padrão, o React DOM utiliza o camelCase como convenção ao dar nomes de propriedades em vez dos nomes de atributos do HTML; logo, usaremos `className` no lugar de `class` e `tabIndex` em vez de `tabindex`.

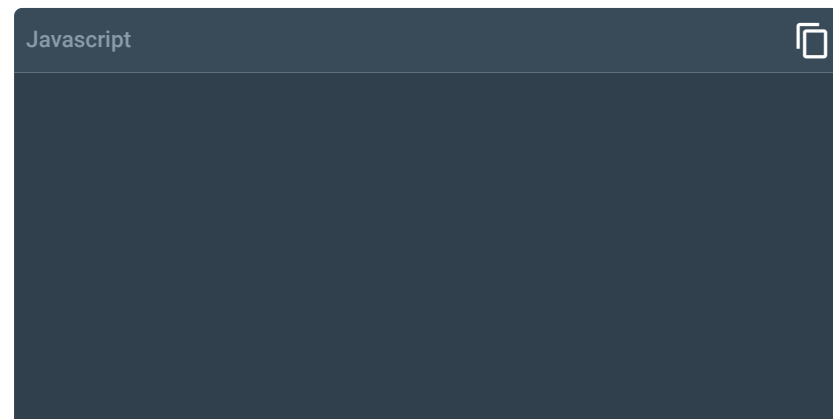
Se uma tag está vazia, podemos fechá-la imediatamente, o que é igualmente utilizado em XML:

```
const elemento = <img src={user.photoPerfilUrl} />;
```

A tag JSX pode conter elementos filhos:



A prevenção contra ataques de injeção (*injection attacks*) é muito comum em um aplicativo web. O JSX consegue prevenir ataques XSS (cross-site-scripting), pois o React DOM disponibiliza qualquer valor incorporado no JSX antes de renderizá-lo. Tudo é convertido para string antes de ser renderizado.



O JSX facilita (e muito) o aprendizado em React. O Babel (compilador do React) compila o JSX para chamadas `React.createElement()`, e essas chamadas realizam verificações que nos ajudam a criar um código sem bugs, criando objetos. Esses objetos são lidos pelo React e usados para construir o DOM, mantendo-o atualizado.

Veremos agora dois exemplos de códigos em que são gerados os mesmos objetos pelo Babel:

Exemplo

Observe o primeiro:

```
const element = <h1 className="greeting">Hello, world!</h1>;
```

Agora, veja o segundo:

```
const element = React.createElement("h1", { className: "greeting" },  
"Hello, world!");
```

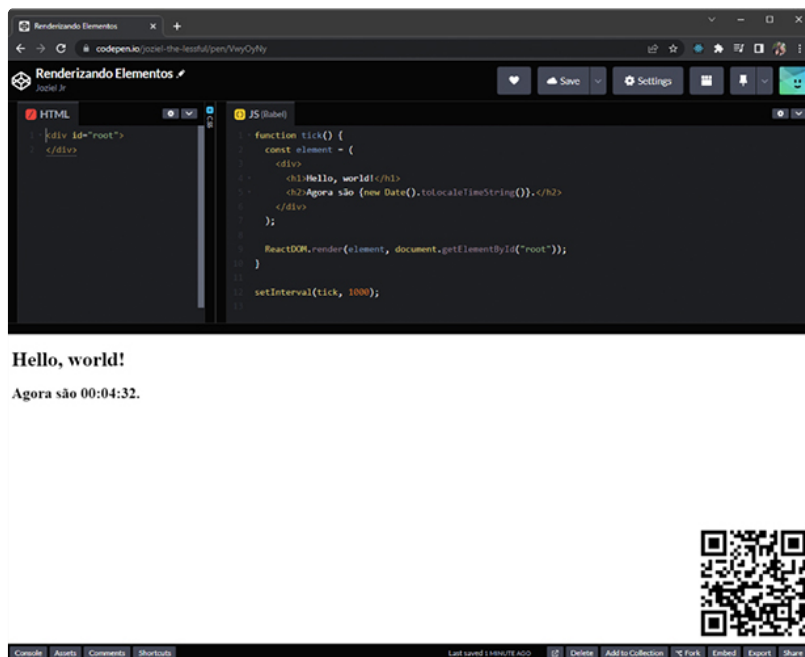
Ambos geram um mesmo objeto, o qual, de forma simplificada, apresenta-se da seguinte forma:



Renderizando elementos

Em nossos exemplos anteriores, a marcação `<div id="root">` sempre foi utilizada para renderizar nossos elementos. Com o que aprendemos até agora, já sabemos que os elementos React são imutáveis, ou seja, uma vez criados, não é possível alterar seus elementos filhos ou atributos. Além disso, só conseguiremos atualizar a interface se criarmos um novo elemento React e enviarmos por meio do `render()`.

O React atualiza no DOM apenas os dados novos. Veremos isso na prática com este código:



Exemplo de React DOM.

Inspeccionando a página, vemos que a única mudança ocorre na marcação `<h2>`, o que comprova o que estudamos anteriormente: o React DOM faz toda essa comparação.

Segue a imagem demonstrando tal mudança. Verifique também o endereço <http://tiny.cc/jmzquz> (CodePen) e inspecione o elemento citado:

Inspeccionando a página.

Estruturação de componentes

Componentes de função

Podemos considerar que o componente é o coração do React. Um componente constitui um bloco de código independente e reutilizável, permitindo dividir toda a interface do usuário em pedaços menores, os quais, por sua vez, aceitam entradas chamadas de props e retornam elementos React.

O React trabalha com **dois tipos de componentes**:



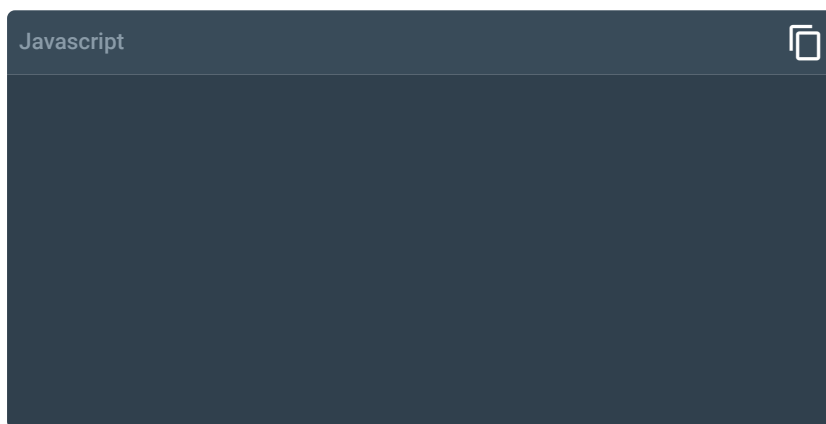
Componentes funcionais



Componentes de classe

O componente funcional basicamente é uma função JavaScript que retorna um elemento React responsável por descrever basicamente o que será apresentado na aplicação. Além de poder receber parâmetros, comumente chamados de props, esse componente, por padrão, não possui gerenciamento de estado do componente, necessitando dos Hooks para isso.

Observe o exemplo de código de um componente funcional que recebe props como parâmetro:



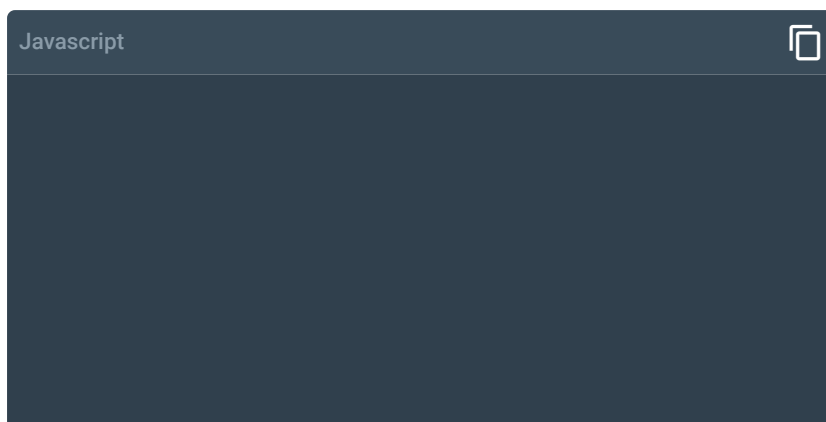
Componentes de classe

Componentes de classe são classes do ES6 (JavaScript) que também retornam um elemento do React. Eles:

- Gerenciam o próprio estado;
- Possuem grande nível de poder dentro da aplicação;
- Herdam os chamados métodos de ciclo de vida do React;

- Lidam com partes lógicas;
- Manipulam eventos por intermédio de métodos que podem ser chamados de qualquer lugar do componente ou em seus filhos.

Observe um exemplo de um componente do tipo classe que, ao final, gera o mesmo elemento React do exemplo anterior:



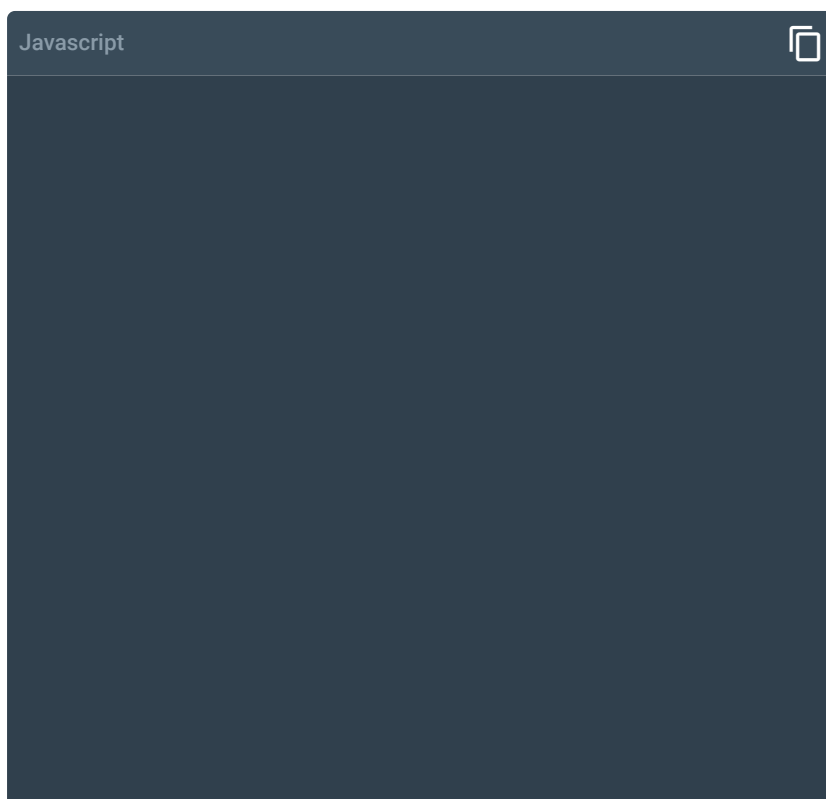
Processo de renderização de componentes

Só utilizamos até agora exemplos cujas funções retornam sempre um elemento React representando tags do DOM; porém, ao se definir uma classe, resta a dúvida: como é feita a passagem do parâmetro?

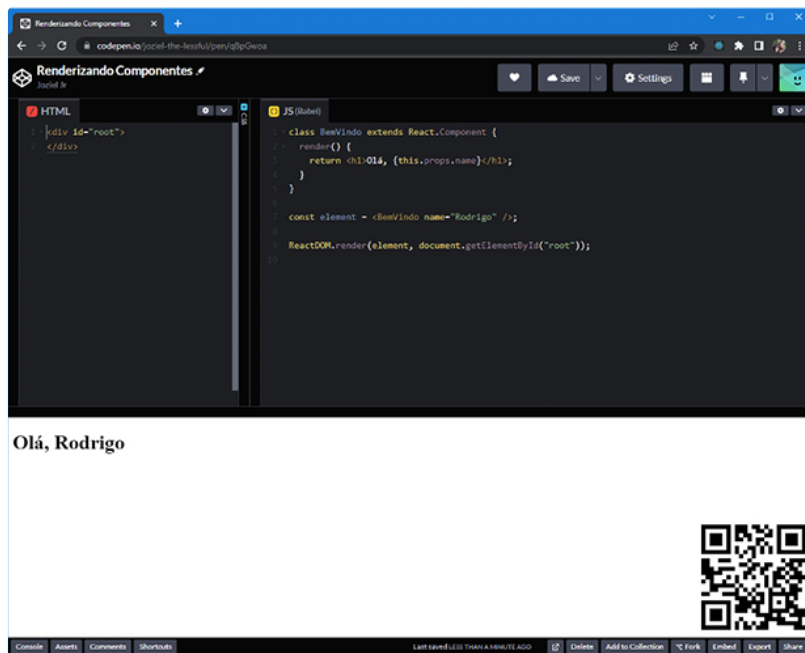
Vamos usar a classe criada anteriormente (BemVindo), definindo um valor para o name com a seguinte sintaxe:

```
const element = <BemVindo name="Rodrigo"/>;
```

Reescrevendo nosso código em <http://tiny.cc/g90ruz>, temos isto:



Para o React, a classe <BemVindo> com o atributo que foi passado (name="Rodrigo") é chamada de props. Para usar esse atributo, é preciso usar a palavra reservada "this". Veja o resultado do código:



Exemplo de atributo props.

Nossa classe do exemplo utilizou apenas um atributo (props.name). Se tivéssemos outros, teríamos esta sintaxe:

```
const element = <BemVindo name="Rodrigo" age="20" country="Brasil"/>;
```

O React passaria como props o seguinte valor: `{name:'Rodrigo', age:'20', country:'Brasil'}`

E, em nossa classe, acessaríamos conforme o campo necessário.

Atenção!

O React DOM utiliza o camelCase na convenção dos nomes, tratando componentes com nomes iniciados com letras minúsculas como uma tag do DOM e aqueles iniciados com maiúsculas como componentes. Desse modo, é necessário que ele esteja no escopo ao criarmos nosso componente do tipo classe.

Prefira sempre escolher componentes funcionais quando precisar exibir apenas alguma interface, já que sua construção será mais simples, principalmente se reescrevermos essa função com arrays functions. Os componentes de classe são mais recomendados em tarefas mais complexas que manipulam eventos, precisam lidar com alguma lógica e gerenciam o estado.

Manipulando componentes

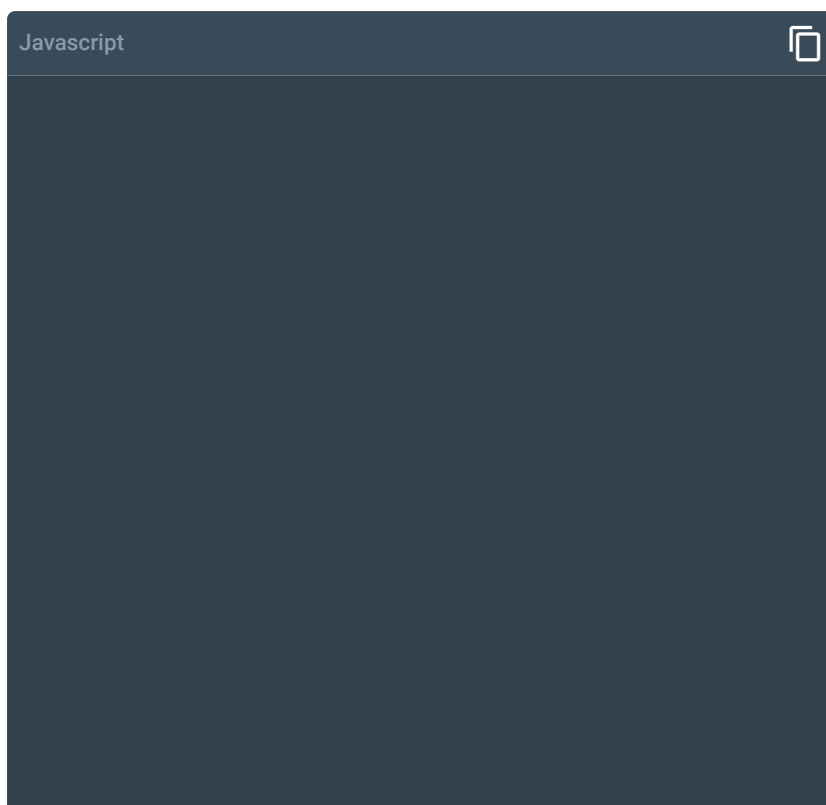
Já sabemos criar componentes, mas precisamos agora entender como manipulá-los. Configurando uma composição de componentes, como, por exemplo, uma página, teremos formulário, botões, barra de menu e outros elementos que serão componentes em nosso projeto.

No próximo exemplo, montaremos uma página simples com cabeçalho, coluna de navegação, coluna principal, topo e rodapé:

Estrutura de uma página.

Iniciemos pelo componente Header, que será um `<header>` com nome de classe `'cabecalho'`, contendo ainda um `<h1>` de nome de classe `'boasVindas'` com um texto "Bem-vindo" e uma props com um atributo `name`.

Esse componente também possui um `<h2>` de nome de classe `'titulo'` com texto "Manipulando Componentes", conforme aponta este código:



Em seguida, há outro componente chamado Navegacao. Trata-se de uma `<div>` de nome de classe `'navegacao'` e um texto "Barra de Navegação":



Continuando, vemos que o próximo componente, cujo nome é Principal, possui uma <div> com nome de classe 'principal' e uma "Coluna Principal" como texto incluso:

```
JavaScript
```

Finalizando os componentes, de forma isolada, existe o Rodape , um <footer> com classe nomeada de 'rodape' que contém um parágrafo <p> com texto "Rodapé":

```
JavaScript
```

Até aqui não houve nenhuma novidade em relação à criação desses componentes. Mas como fazer para chamar um componente dentro do outro? Observe esta sintaxe:

```
JavaScript
```

Com mais de um componente, é preciso encapsulá-los dentro de uma tag DOM (no exemplo, <div>) e obedecer à seguinte sintaxe:

```
JavaScript
```



Deve-se criar um componente Topo, conforme o diagrama inicial, que é a junção dos componentes Header e Navegacao. Lembre-se de que a Header possui uma mensagem de boas-vindas com o nome do visitante (um parâmetro, ou seja, uma props). Eis o código:

```
JavaScript
```

Já temos toda a estrutura escrita. Contudo, qual delas será utilizada para substituir o elemento “root” em nosso ambiente de testes?

Como solução, criaremos um componente reunindo todos os componentes criados (<http://tiny.cc/433ruz>):

```
JavaScript
```



Veja a seguir a saída esperada:



Exemplo de página.

Ao inspecionar o código compilado na imagem a seguir, observe que criamos uma tag <div> desnecessária:

Código compilado.

Não precisaríamos da <div> “app”. No entanto, como se consegue incluir o componente de um mesmo nome sem acrescentar um novo nó? Podemos utilizar Fragmentos, o que nos permite agrupar uma lista de componentes sem adicionar nós extras ao DOM. Veja sua sintaxe:

JavaScript

Ou, de uma forma mais simplificada:

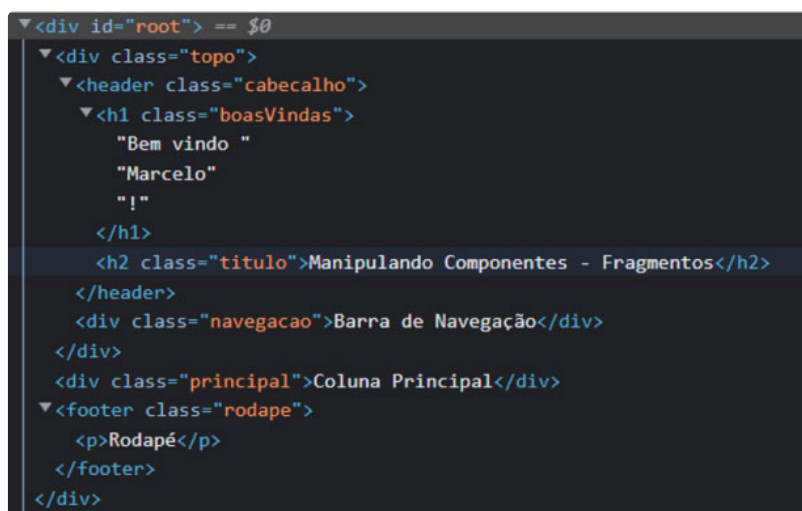
JavaScript

Verifique o resultado do nosso exemplo anterior com a utilização da forma simplificada de fragmentos (<http://tiny.cc/rw5ruz>):



Exemplo de página com fragmentos.

E inspecionando o código compilado a seguir, o que confirma a eliminação da <div>:



Código compilado.

Sempre é recomendado dividir componentes em unidades menores, simplificando a codificação e criando componentes reutilizáveis que podem ser utilizados em outros aplicativos ou projetos. No início, tal processo será demorado, mas ter esses componentes disponíveis tornará seus futuros trabalhos mais rápidos, limpos e prontos para sua produção.

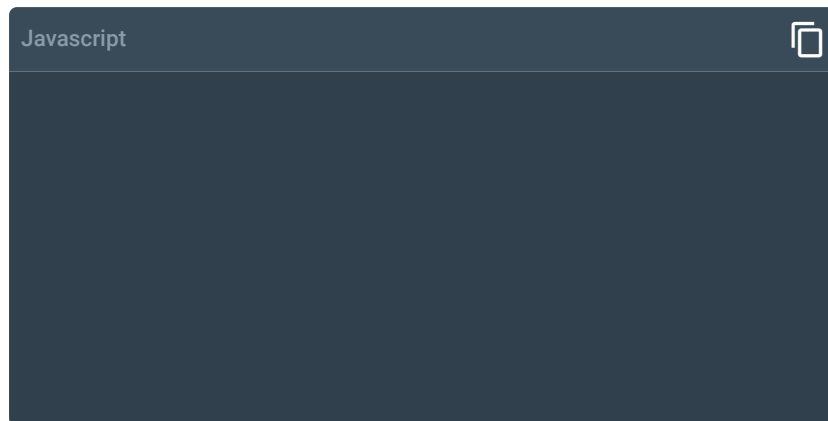
Nesse exemplo, podemos ver a utilização de props, passando o nome do usuário como atributo ou carregando esse valor na classe <Header> com o this.

Dica

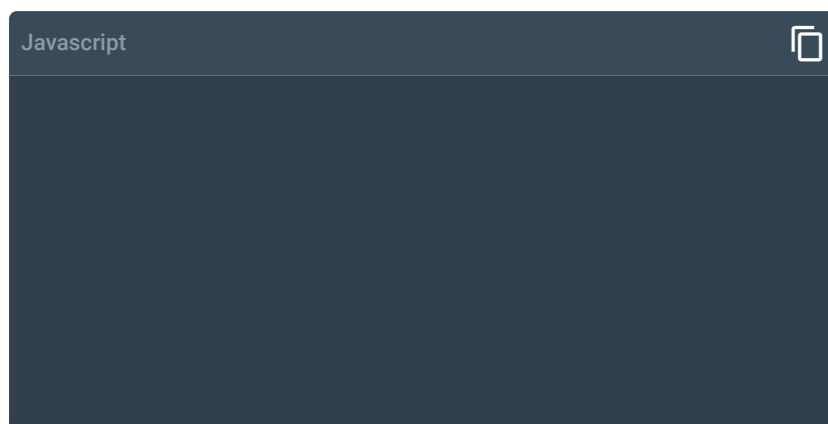
Ao empregarmos o this, usamos o valor da props renderizado naquele instante.

Outro importante conceito é que os componentes React agem sempre com funções puras em relação ao seus props. Função pura é aquela que

não altera suas entradas, retornando sempre o mesmo resultado para as mesmas entradas. Veja um exemplo de uma função pura:



De maneira oposta, uma função impura é aquela que altera a própria entrada na qual o valor pode variar. Observe este exemplo:



O problema dessa função está no valor da variável global `c`, que possui um estado compartilhado e está fora do escopo da função, não havendo previsão do seu resultado. O estado compartilhado acaba provocando uma dependência de tempo.

Entretanto, as interfaces são dinâmicas, mudando conforme o tempo e as ações do usuário. Mas como não ir contra essa regra de função pura? Veremos a resposta a seguir, estudaremos o conceito de estados (state) e o ciclo de vida dos componentes.

Estado e ciclo de vida dos componentes

States

Enquanto as props são parâmetros passados por funções e imutáveis, os states são declarados e manipulados dentro dos componentes,

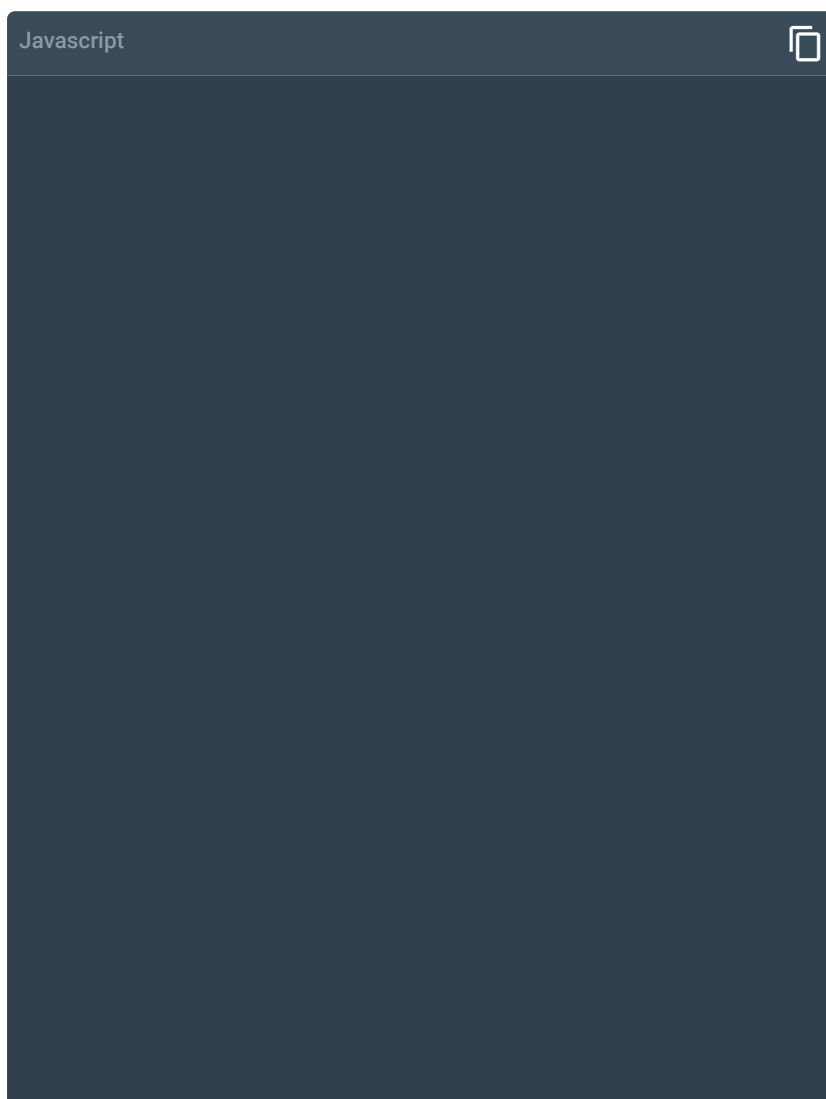
podendo ser editáveis. Para cada modificação no seu valor (`setState`), uma nova renderização é feita, tal qual ilustra a imagem a seguir.



Modificação do estado de um componente.

Como ainda não estudamos Hooks (assunto que será abordado no próximo módulo), empregaremos agora um componente do tipo classe para exemplificar o uso de estados em React. Criaremos uma aplicação simples com um único botão incrementando uma variável desse componente.

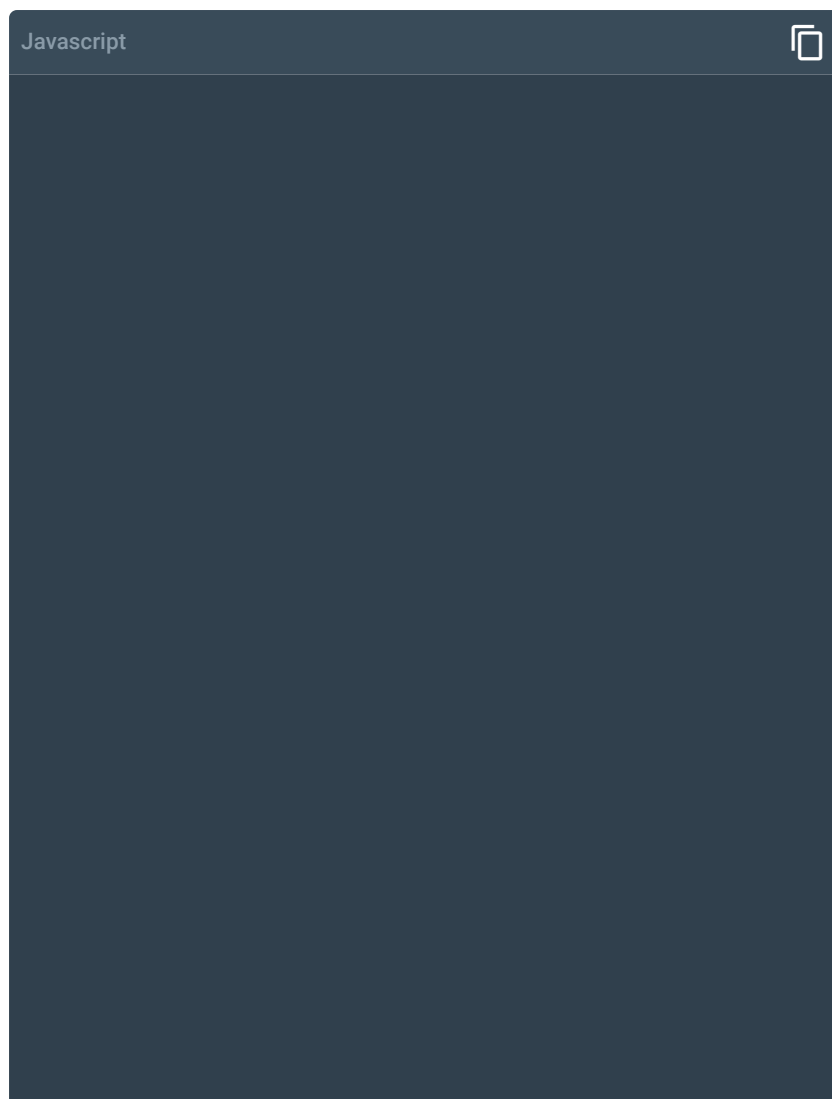
É importante observar a chamada do construtor da classe. Observe o código:



Sugerimos verificar isso on-line no endereço <http://tiny.cc/d27ruz> e acompanhar o resultado, que pode ser visto na imagem a seguir:

Uso de estados em React.

Sempre que precisarmos utilizar valores que serão modificados na renderização, essas variáveis deverão ser declaradas como atributos no estado da classe. Vejamos outro exemplo:



Observe a adição de um método `mudarUsuario`, que tem como objetivo alterar o estado por meio das propriedades passadas. Propositalmente, escolhemos não alterar o sobrenome e a nacionalidade dentro do método, demonstrando que o React faz uma mesclagem e renderizando apenas os valores que foram atualizados (nome e idade).

Outro método adicionado foi o incremento da idade em uma unidade. Veja o resultado (<http://tiny.cc/w57ruz>):

Renderização realizada pelo React.

É preciso seguir algumas recomendações ao manipular estados. Sempre que precisássemos atualizar um estado, intuitivamente usaríamos o seguinte:

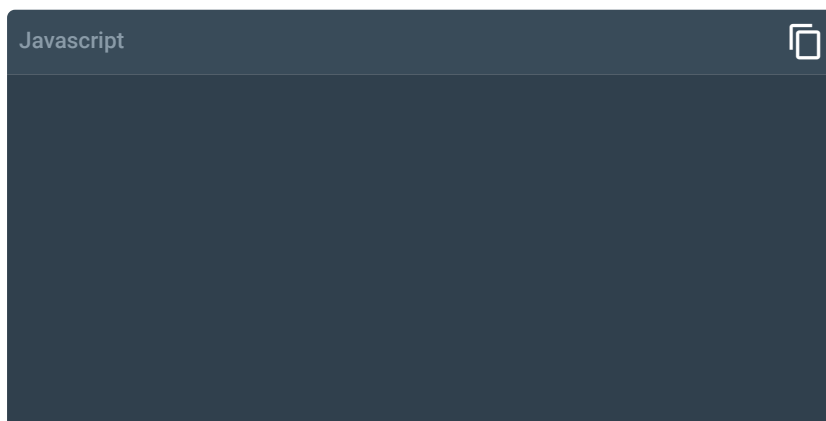
```
this.state.numero = this.state.numero + 1;
```

Entretanto, isso não renderizará novamente nosso componente: o único local em que podemos atribuir `this.state` é no construtor. Para atualizarmos seu valor, utilizamos sempre o `setState`.

Outro problema está na sincronização dos estados; afinal, o React pode agrupar várias chamadas `setStates` em uma única atualização na aplicação. Eis um exemplo de código que pode acabar retornando um valor diferente devido à utilização de `states` e `props`:

```
this.setState({ numero: this.state.numero + this.props.outroValor });
```

Uma forma de corrigir seria reescrever o nosso parâmetro como uma função que, conforme demonstra código adiante, terá como primeiro parâmetro o estado anterior e como segundo as propriedades:



Outra opção é fazer isso com uma notação com `arrows functions`:



Acompanhe os links <http://tiny.cc/k67ruz> e <http://tiny.cc/r67ruz> para verificar que tais recomendações foram seguidas e que os exemplos anteriores foram reescritos da forma correta.

Ciclo de vida dos componentes

Nosso estudo sobre o ciclo de vida dos componentes pretende descrever seus métodos e comportamentos, além de apontar quais são os mais utilizados. Em qualquer linguagem de programação, ciclo de vida significa o tempo em que uma aplicação, um objeto ou parte de um código está em execução.

Esse processo sempre começa por uma inicialização, podendo sofrer alguma alteração e terminando quando ela está “morta”. Os componentes seguem o mesmo conceito: iniciados quando o componente é renderizado, eles serão atualizados sempre que houver a necessidade e finalizados ao serem removidos da tela.

Diagrama do ciclo de vida de um componente React.

Fases do ciclo de vida

O ciclo de vida é dividido em três fases: montagem, atualização e desmontagem. Cada uma contém métodos que auxiliam na codificação da nossa aplicação, sendo executados no momento já definido e apropriado para aquele componente.

Descreveremos a seguir essas três fases, apontando seus métodos e, em seguida, detalhando cada um deles.

Montagem

Na primeira fase, configuramos os estados iniciais (states) e as propriedades (props) do componente, preparando-o para sua renderização, isto é, para que ele seja escrito efetivamente no DOM do navegador e o que será feito após sua primeira renderização.

Observe agora seus métodos e, principalmente, a ordem na qual eles são chamados quando instanciamos um componente:

- `constructor();`
- `static getDerivedStateFromProps();`
- `render();`
- `componentDidMount().`

Atualização

Esta fase se inicia quando o componente React já está no navegador e aumenta conforme ele recebe atualizações. Essas atualizações podem ocorrer por meio de novas props ou modificação do seu estado. Observe seus métodos e a ordem em que são chamados:

- `static getDerivedStateFromProps();`
- `shouldComponentUpdate();`
- `getSnapshotBeforeUpdate();`
- `render();`

- `componentDidUpdate()`.

Desmontagem

Nesta fase, o componente não é mais necessário, sendo removido do DOM. Ela possui apenas um único método: `componentWillUnmount()`.

Métodos constructor (props)

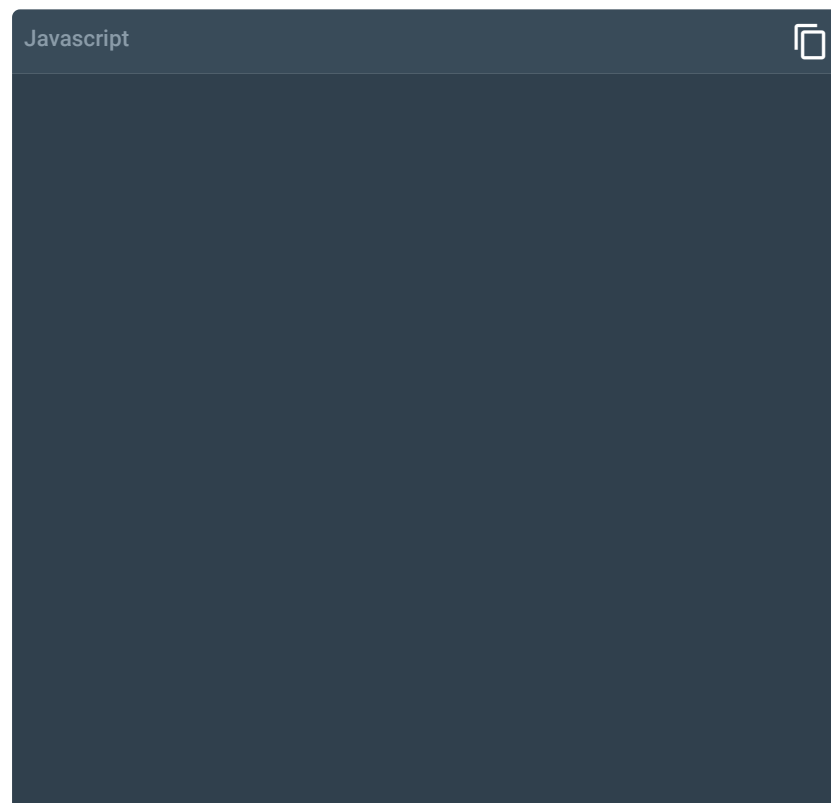
Trata-se de um dos principais métodos do ciclo de vida do React.

Embora sua implementação não seja obrigatória, ele é chamado antes de o componente ser montado, sendo utilizado normalmente para a atribuição inicial de nossos estados iniciais (`this.state`).

Caso seu componente seja uma subclasse de `component`, deve-se chamar sempre `super(props)` no início do código, permitindo, com isso, a utilização de (`this.props`) e evitando erros ou bugs na execução.

Normalmente, ele também é utilizado com binds de outros métodos.

Observe um trecho de código do método construtor:



`static getDerivedStateFromProps (props, state)`

Método raramente utilizado, ele é aplicado quando o state possui uma dependência temporal de props.

Exemplo

Um componente gráfico que compare seus filhos anteriores e os subsequentes para decidir quais devem ser animados. Mesmo assim, uma solução melhor seria utilizar outro método mais simples: o componente `DidUpdate`.

O `static getDerivedStateFromProps` sempre é executado imediatamente antes de o método `render`, ambos ocorrendo na montagem inicial e sempre que haja alguma atualização. Ele admite dois parâmetros de entrada (`props` e `state`) preenchidos automaticamente pelo React e retorna um objeto para atualizar o `state` ou um `null`, o que indica que não houve mudança.

`render()`

Trata-se do único método obrigatório em um componente do tipo classe. Quando é chamado, ele examina `this.props` e `this.state`, retornando um elemento do tipo React a ser carregado pelo componente.

Sempre que um componente for carregado para ser renderizado e a qualquer momento em que houver uma atualização, esse método será chamado. Na documentação, recomenda-se que a função `render()` deva ser pura, não modificando o `state`, o que fará com que os componentes sejam mais fáceis de se utilizar.

Caso precise interagir com o navegador, utilize o método `componentDidMount()` ou outros métodos do ciclo de vida. Se o método `shouldComponentUpdate()` retornar falso, o método `render()` não será chamado.

`componentDidMount()`

Método chamado imediatamente após um componente ser montado, ou seja, inserido na árvore DOM, sendo chamado o método `render()` e ocorrendo a renderização de fato do componente. É um local ideal para scripts que podem dificultar a renderização da página ou aplicação, pois, com a tela já renderizada, não haverá bloqueio e travamentos nesse processamento.

Utilize-o caso precise carregar dados de origem de alguma API externa. Não se esqueça de liberar esse recurso em `componentWillUnmount()`.

É possível chamar o `setState()` diretamente nesse método. Entretanto, embora o estado intermediário do componente não seja visto pelo usuário, isso adicionará uma renderização extra e tudo ocorrerá antes de o navegador atualizar a tela, o que pode gerar uma queda de desempenho, pois o `render()` foi chamado duas vezes ao final do processo.

shouldComponentUpdate (nextProps, nextState)

Método que verifica se os valores atualizados do componente precisam ser novamente renderizados, ajudando na performance em alguns casos de constantes atualizações. Por padrão, retorna um valor verdadeiro; se, ao contrário, ele retornar um falso, os métodos `render()` e `componentDidUpdate()` não serão executados.

getSnapshotBeforeUpdate (prevProps, prevState)

Executado antes de a árvore DOM ser renderizada, ele é utilizado para armazenar valores anteriores do estado do componente após uma atualização do DOM. Qualquer valor retornado por ele será usado como parâmetro no método `componentDidUpdate()`.

componentDidUpdate (prevProps, prevState, snapshot)

Executado sempre que um estado do componente é atualizado logo após a chamada do `componentDidMount()`. É um local considerado ideal para realizar as requisições de rede enquanto as props atuais e as anteriores são comparadas.

Quando você precisar atualizar o estado de um componente (`setState`), lembre-se sempre de encapsular essa parte do código com algum bloco condicional, evitando, assim, um loop infinito e gerando renderização extra sem necessidade, o que afeta a performance do componente.

O último parâmetro vem do retorno do método `getSnapshotBeforeUpdate()`, que só será utilizado ao implementarmos em nosso componente; caso contrário, ele não é definido. Dificilmente, porém, implementaríamos esse método em nosso componente. Se o método `shouldComponentUpdate()` retornar falso, o `componentDidUpdate()` não será executado.

componentWillUnmount()

Executado antes de o componente ser destruído, ele é chamado na fase de desmontagem do componente. Devem ser codificados nesse método:

- Toda a limpeza relacionada a temporizadores;
- Alguma dependência de assinatura gerada no `componentDidMount()`;
- Logout;
- Tokens de autenticação que precisam ser removidos;

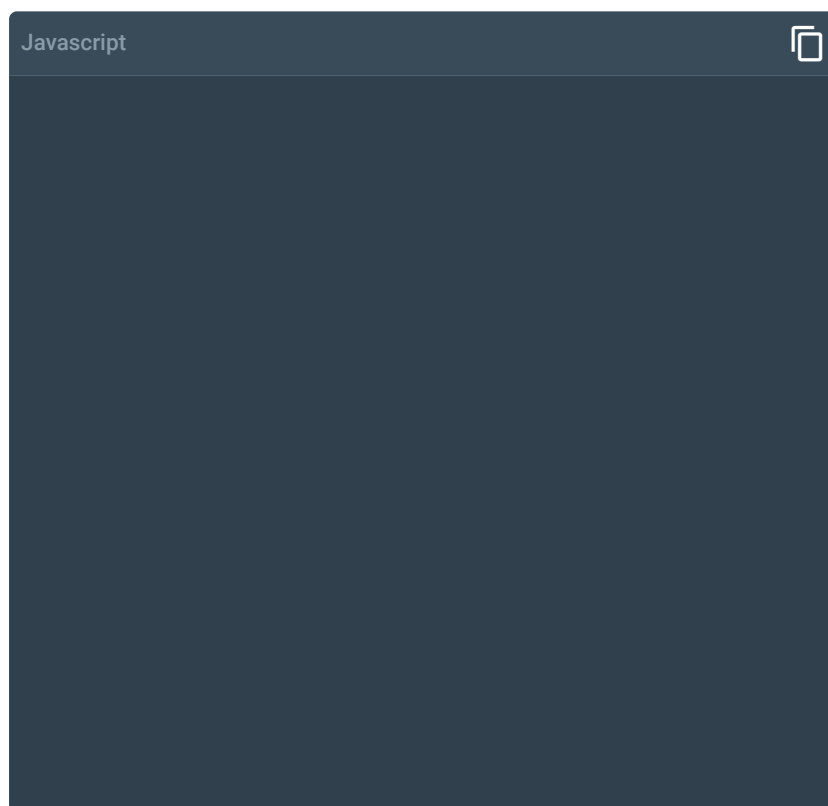
- Cancelamento de requisições de rede.

Bind e eventos

Bind no React

Bind é uma ligação que permite alterar o comportamento do **this** de determinado elemento. Em um evento, o this se refere, na maioria dos casos, ao componente que invocou esse evento.

Vejamos o exemplo de sua utilização em JavaScript:

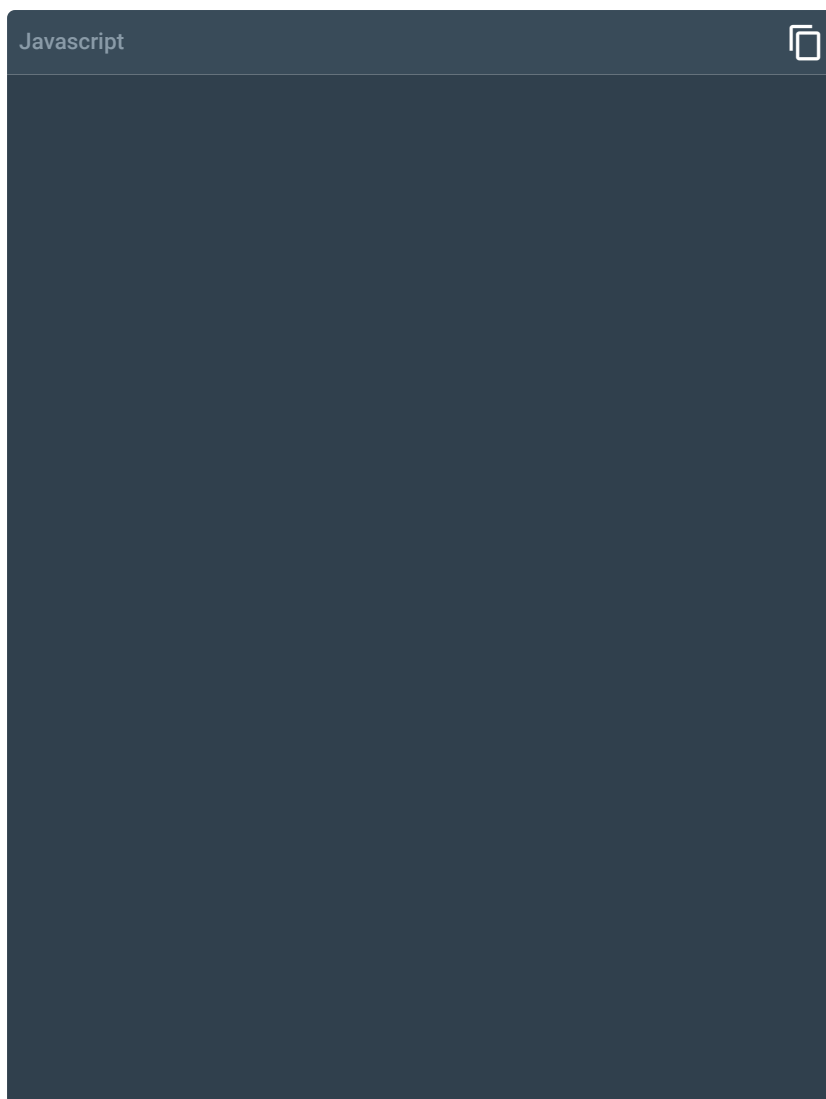


Nesse exemplo (<http://tiny.cc/55mruz>), conseguimos criar uma função `exibirContexto` que basicamente exibe o `this` no console (linha 2). Em seguida, criamos um objeto `Usuário` com os atributos `nome`, `idade` e `nacionalidade`.

Note que, se tentarmos acessar os atributos dentro da função `exibirContexto()`, haverá um erro, pois eles não existem no contexto da função. No entanto, utilizando o `bind` conforme a linha 12 da imagem adiante, conseguimos acessar os atributos de usuário. Não se esqueça de utilizar o `this`.

Exemplo de bind.

Em React, é possível usar o `bind` para substituir alguma arrow function. Reutilizaremos agora parte de um código visto anteriormente, o que demonstra como sua utilização facilita o entendimento na codificação.



Dentro do construtor, criamos uma função `bindMudarUsuario` (linha 8), fazendo o `bind` na função `mudarUsuario` (11) e passando o `this` como parâmetro. Já na linha 23, passamos simplesmente como referência o `this.bindMudarUsusario` no evento `onClick`, o que facilita o uso das funções, conforme aparece em <http://tiny.cc/p4mrz>.

Manipulando eventos

Vamos estudar agora como o React manipula eventos. Para isso, veremos desde como é feita a passagem de uma função para o evento até de que modo é repassado um state.

Basicamente, um evento é o resultado de uma ação. Quando o usuário clica com o botão esquerdo do mouse ou pressiona alguma tecla, a aplicação recebe um evento, um `onClick` ou um `keyEvent`, por exemplo.

Comparemos o mesmo evento, mostrando como ele é escrito em HTML e como o seria em React:

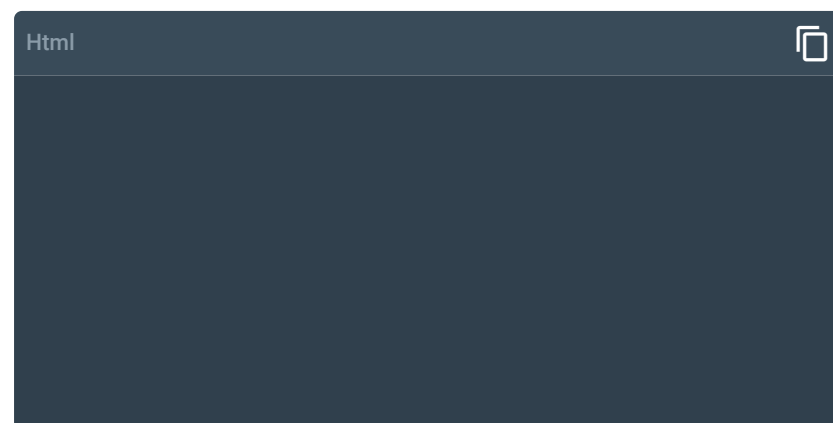
```
<button onclick="enviarForm()">Enviar formulário</button>
```

Quando nosso botão (Enviar formulário) for pressionado pelo usuário, o evento onclick será chamado e a função enviarForm(), executada. Nesse exemplo HTML, contudo, há um problema: a manipulação de diferentes eventos.

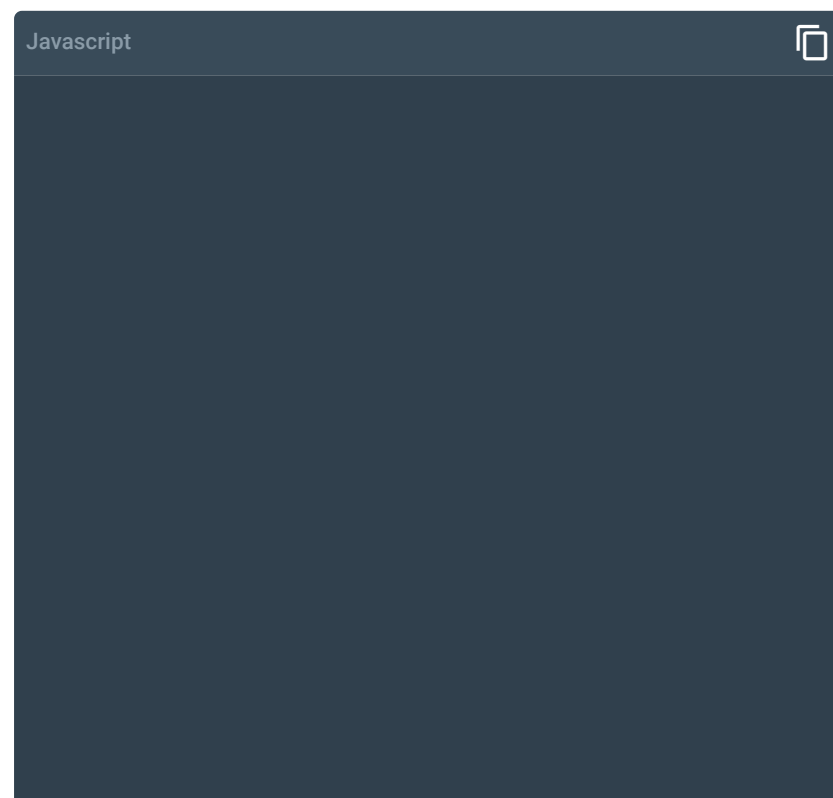
Os eventos em React são nomeados usando camelCase. É possível passar uma função no lugar de uma string.

```
<button onClick={enviarForm}>Enviar formulário</button>
```

Não estamos chamando o método enviarForm(), e sim uma referência {enviarForm}. Outra diferença: não podemos impedir o comportamento padrão do React retornado um booleano falso. Em vez disso, devemos chamar o preventDefault explicitamente. Por exemplo, em um HTML, caso precisássemos evitar que uma nova página fosse aberta, escreveríamos assim:

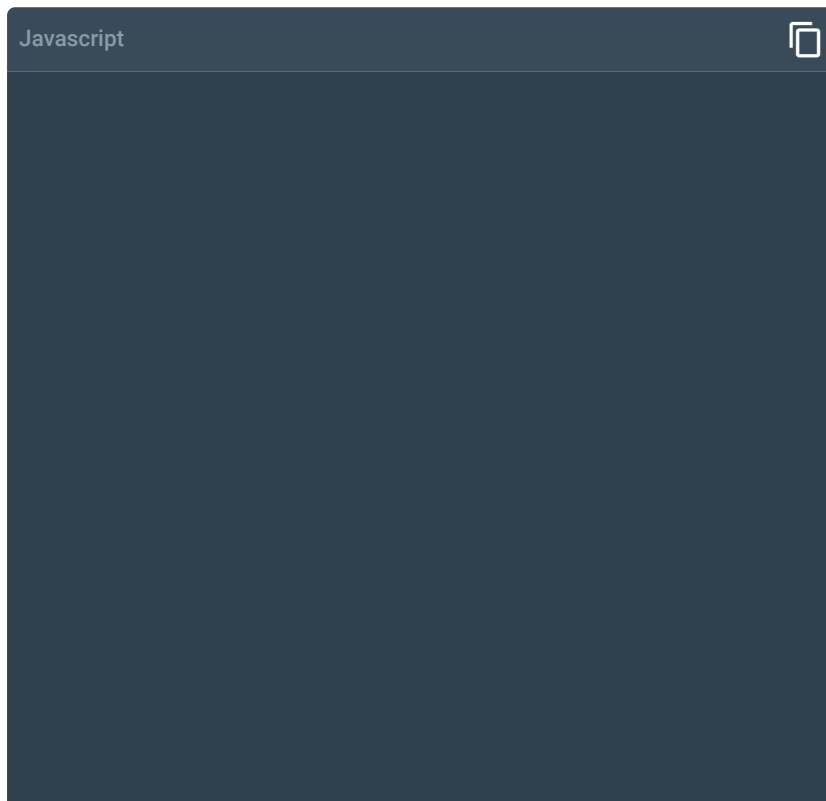


Já em React, isso seria reescrito da seguinte forma:



Note que não utilizamos os parênteses ao chamar a função em `onClick`. Caso os colocássemos, assim que a página fosse exibida, a função seria executada sem clicar no link.

Para evitar esse comportamento, usemos o `preventDefault`, cancelando o comportamento padrão do objeto. Também precisamos realizar uma ligação com o `bind`:



O parâmetro `"e"` utilizado em `handleClick()` é um evento sintético. Isso significa que se trata de um evento específico passado como instância e acumulado, ou seja, o evento `"e"` será reutilizado algumas vezes e suas propriedades serão anuladas após o callback do evento ser acionado e finalizado.

Outra forma seria por meio de arrows functions que já incorporam o `bind` em sua implementação, não havendo a necessidade da utilização de eventos sintéticos. Observe novamente como podemos reescrever o código com arrows functions:

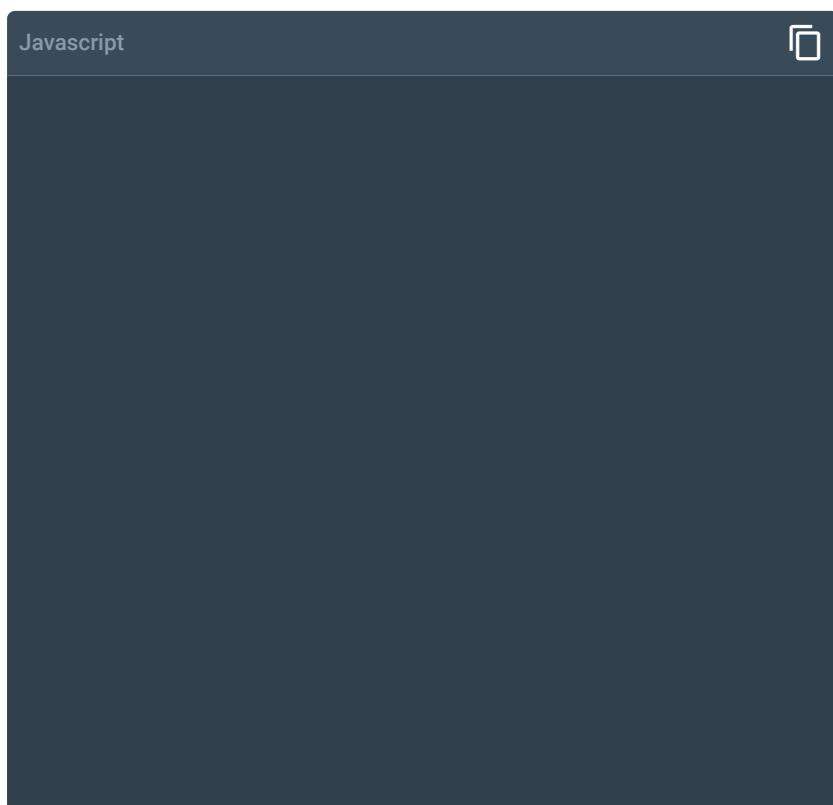




O React define esses eventos de acordo com as especificações do W3C, garantindo uma compatibilidade entre navegadores. Para mais detalhes sobre eventos sintéticos, sugerimos a leitura de sua documentação em <https://pt-br.reactjs.org/docs/events.html>.

Geralmente, não é necessário utilizar o `addEventListener` para adicionar ouvintes a um elemento no DOM depois que ele é criado. É possível somente definir um ouvinte quando o elemento é inicialmente renderizado.

Caso precisássemos passar algum parâmetro no evento, bastaria definir dentro da nossa função as entradas necessárias e, por meio de arrows functions, repassá-las:

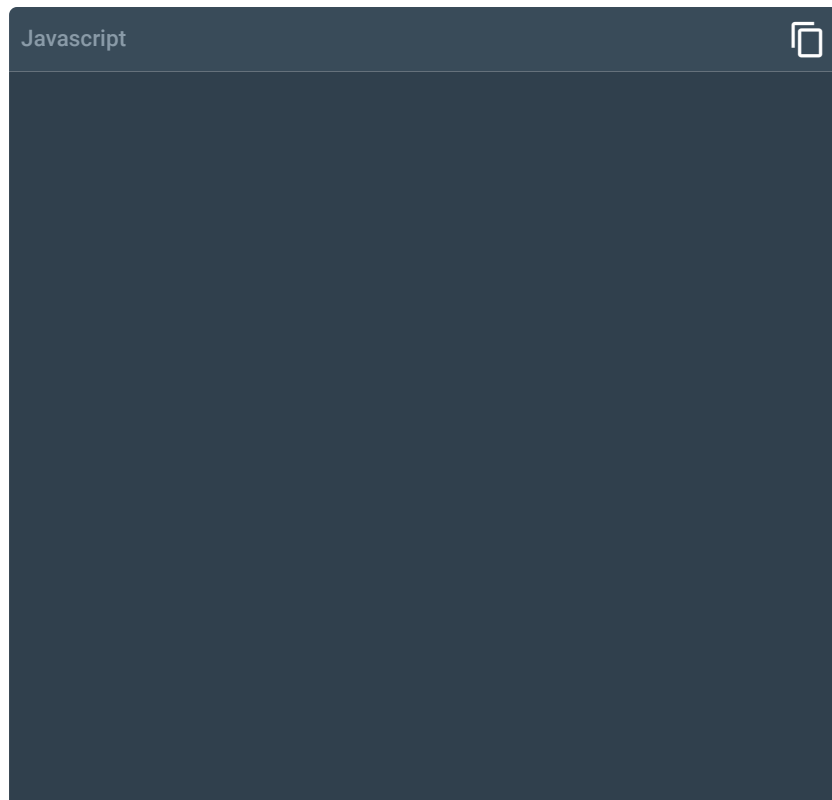


Elementos visuais

Listas e chaves

Em React, manipular listas se torna uma tarefa bem simples: de forma idêntica à do JavaScript, pode-se usar as principais funções para auxiliar na criação de novas listas. Com a utilização do JSX, criar elementos visuais se torna rápido, mas sempre existirá a necessidade de criar uma chave (key) em cada elemento, identificando de forma única e evitando uma perda de performance na renderização desse elemento.

Neste estudo, veremos alguns exemplos e práticas comuns na criação de listas e suas sintaxes. No primeiro exemplo (<http://tiny.cc/w3mrurz>), com o auxílio da função map, geraremos uma lista de elementos populados por um array de carros:



Veja a seguir a saída esperada:

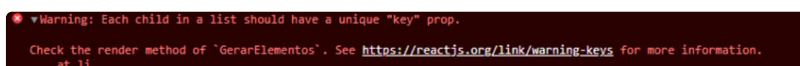
Exemplo de array.

Outra forma de gerar esses elementos seria, por meio de props, passar um array com o conteúdo desejado. Também renomearemos as variáveis, tornando a função mais genérica para qualquer lista de objetos passados via props.



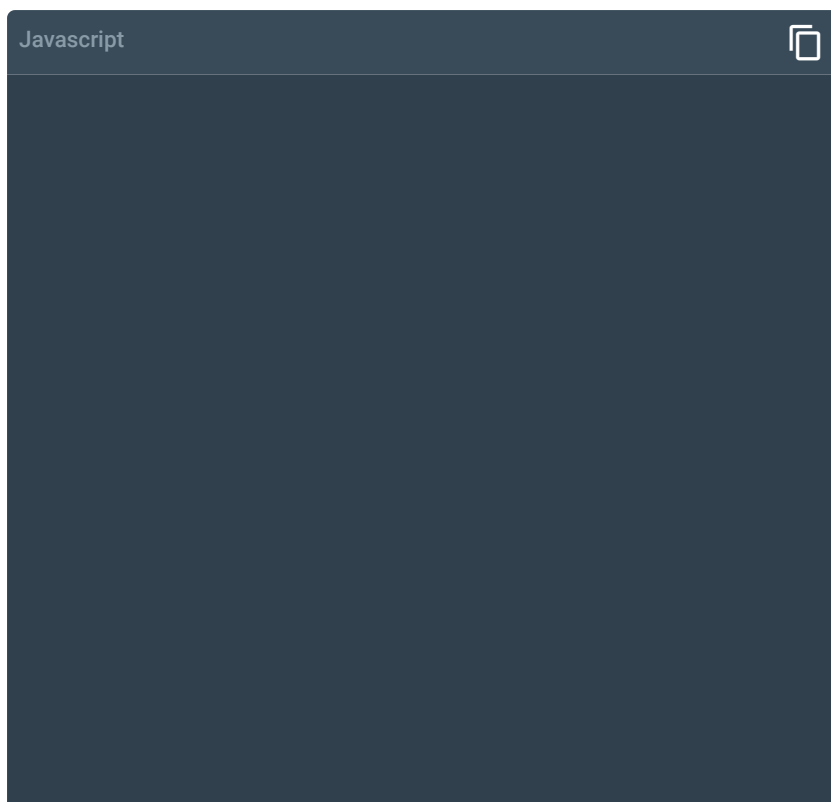


Veja que o resultado é o mesmo, porém, uma mensagem de aviso aparece no console do navegador:



Mensagem no console.

Desse modo, precisamos ter uma chave para cada elemento gerado. Nesse exemplo, podemos modificar nossa lista, atribuindo como chave o próprio valor do elemento. Caso não o faça, o React atribuirá os índices como chave. O código pode ser verificado neste endereço: <http://tiny.cc/94mruz>.



A chave é de extrema importância para o React, auxiliando na fácil identificação de qual elemento foi alterado, adicionado ou removido. Ela sempre deve ser única.

Na maioria das vezes, utilizamos um ID do objeto. Caso não tenhamos um ID estável para isso, podemos usar o índice gerado na listagem. Mas

use-o como último recurso.

```
const listaElementos = elementos.map((e, index) => <li key={index}>{e}</li>);
```

É importante destacar que as chaves são únicas apenas em componentes irmãos, não precisando sê-lo de forma global, podendo ser as mesmas na criação de dois arrays diferentes. Caso precise utilizar o mesmo valor de chave, defina-a explicitamente como uma props com nome diferente.

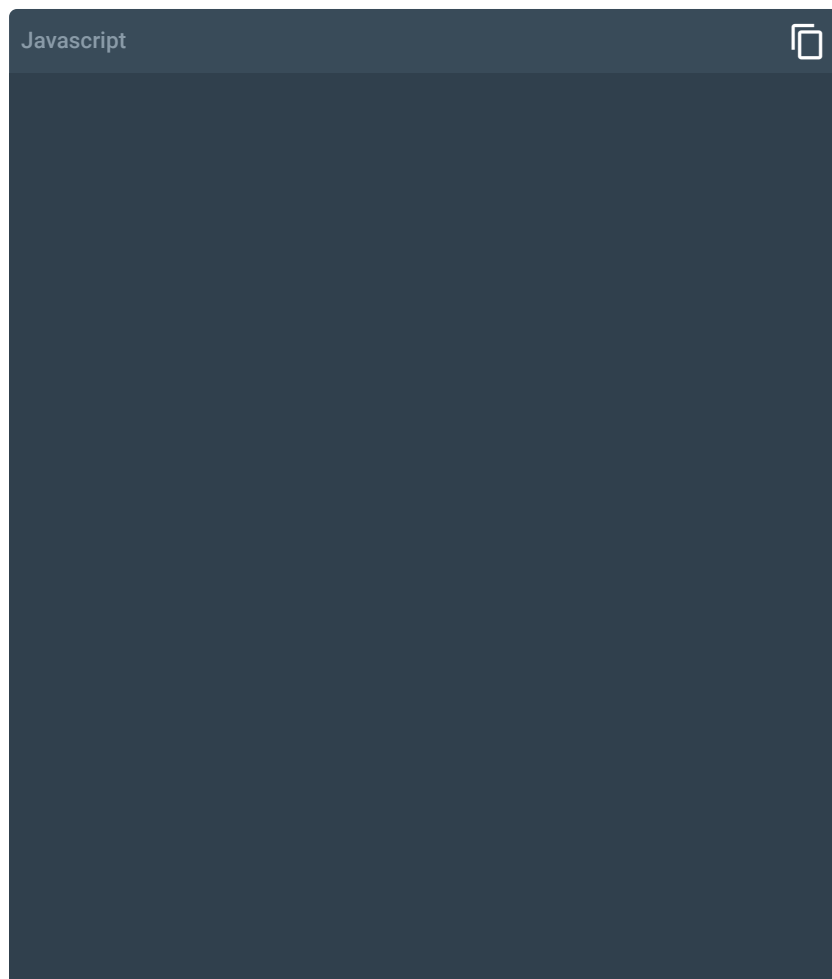
Formulários

Veremos agora como o React trata de elementos nos formulários, aproveitando os estados dos componentes e facilitando a manipulação dos dados nas funções que precisamos compartilhar entre eles.

Iniciando com um formulário simples, vamos:

- Demonstrar as diferenças na utilização de algumas tags usualmente empregadas em páginas HTML;
- Utilizar alguns eventos;
- Ver alguns comportamentos (<http://tiny.cc/t6mrurz>).

Observe o exemplo a seguir:





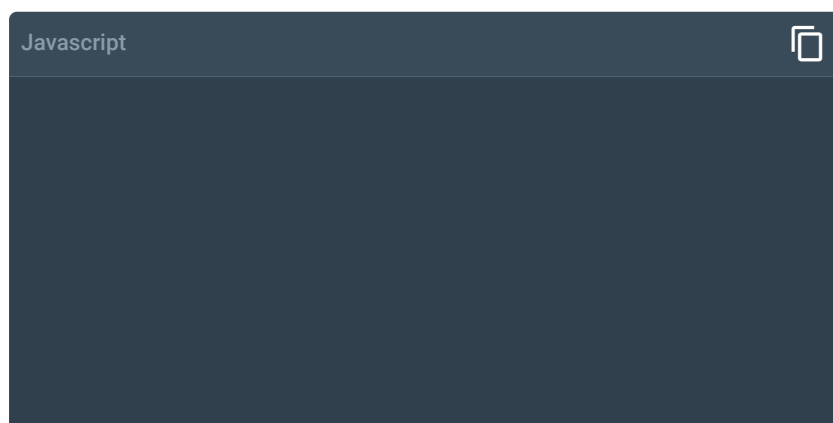
Veja a seguir a saída esperada:

Formulário - Parte 1



Exemplo de formulário.

Nesse exemplo, repare que as duas funções que retornam os valores digitados de nome e idade são idênticas, mudando apenas o campo. Com isso, podemos utilizar uma prática comum: nomear os elementos e, dentro do nosso estado, utilizá-los.



Todo elemento possui um nome que podemos definir ao criarmos. No estado, podemos utilizá-lo por meio de colchetes, tornando nosso código bem mais simples (<http://tiny.cc/87mruz>).

Em outro exemplo, utilizaremos dois elementos com comportamentos um pouco diferentes daqueles utilizados em HTML: TextArea e Select.



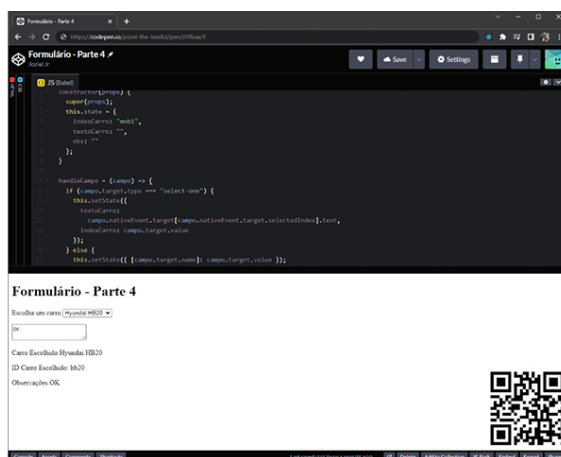


O texto no elemento `<textarea>` é definido por seus filhos. No React, utilizaremos o atributo `value`. Já no `<select>`, geralmente é preciso definir o elemento selecionado por padrão. No React, esse valor é definido no elemento raiz (`select`), e novamente no atributo `value`.

Este exemplo que emprega os dois elementos (<http://tiny.cc/29mrucz>):

Exemplo de formulário.

Note que há um problema no carro escolhido: o texto que selecionamos para exibir é a própria chave. Uma solução encontrada foi modificar nossa função `handleCampo`, que vai verificar qual é o tipo do alvo (`target`) usado. Observando a árvore do elemento, podemos retornar o valor de que precisamos e salvá-lo no estado como `textoCarro`. Vejamos o resultado em <http://tiny.cc/o9mrucz>:



Exemplo de formulário.

Em todos esses exemplos, usamos o conceito de componentes controlados no qual o valor é controlado pelo React. No entanto, acaba sendo cansativo tratar de todos os eventos e inputs.

Saiba mais

Existem soluções completas que facilitam o uso de forms. Sugerimos a leitura da documentação no site oficial para mais detalhes: <https://pt-br.reactjs.org/docs/forms.html#alternatives-to-controlled-components>.



Criação de um formulário com React

Neste vídeo, apresentaremos os principais conceitos sobre a criação de um formulário com React.

Para assistir a um vídeo sobre o assunto, acesse a versão online deste conteúdo.



Falta pouco para atingir seus objetivos.

Vamos praticar alguns conceitos?

Questão 1

Considerando o React, avalie as afirmativas a seguir:

- I. Um elemento descreve o que será renderizado em tela. Um componente é uma função ou uma classe, contendo opcionalmente parâmetros que retornam um elemento.
- II. Algumas das vantagens de se utilizar o React são o aumento de performance da aplicação devido ao uso de Virtual DOM e a capacidade de renderização tanto no lado cliente quanto no do servidor.
- III. States e props são objetos JavaScript que armazenam informações que influenciam na exibição de um componente. Ambos são definidos pelos próprios componentes, podendo ser repassados como parâmetros de uma função.

Aponte a alternativa correta.

- A** I, II e III.
- B** II e III.
- C** III.
- D** I e III.
- E** I e II.

Parabéns! A alternativa E está correta.

Apenas os states são definidos pelo próprio componente como se fossem variáveis locais de uma função. Já as props são definidas fora do escopo do componente, sendo repassadas como parâmetros de uma função.

Questão 2

Sobre o ciclo de vida, marque a alternativa correta.

A

O `getSnapshotBeforeUpdate` é executado logo antes de a renderização ser confirmada no DOM. Qualquer valor retornado será encaminhado para o constructor.

B

O `componentWillUnmount` será usado para executar quaisquer solicitações de rede de saída e para criar ouvintes de eventos associados ao componente.

C

O `render` não é considerado uma função do ciclo de vida, e seu uso não é obrigatório.

D

O `componentDidMount` é executado após a primeira renderização e no local em que todas as atualizações de estado devem ocorrer.

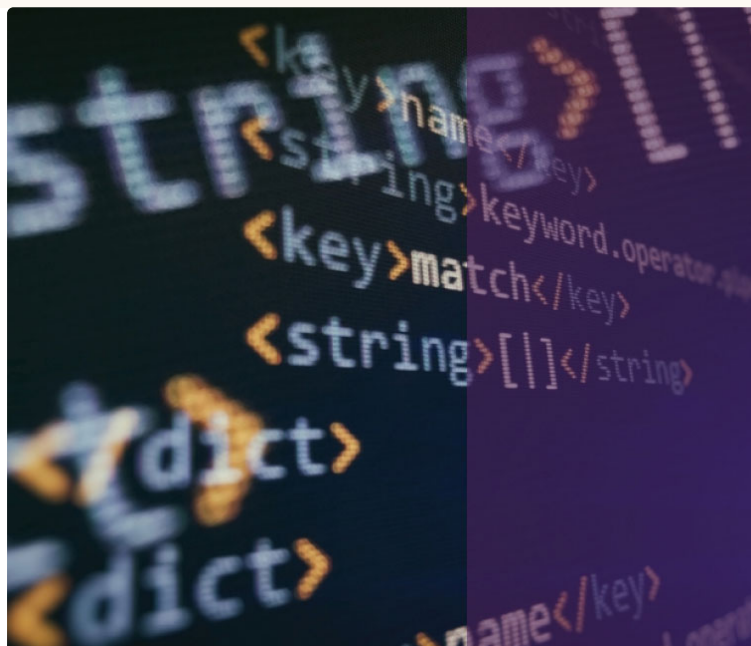
E

Para atualizar um estado, utilizamos o método `setState` dentro do método constructor.

Parabéns! A alternativa A está correta.

O `getSnapshotBeforeUpdate` realmente é executado logo antes de a renderização ser confirmada no DOM, mas seu valor retornado é encaminhado para o `componentDidUpdate`, sendo útil para capturar informações do DOM, por exemplo. O `ComponentWillUnmount` é usado para cancelar (em vez de criar ou executar solicitações) e destruir ouvintes de eventos associados ao componente. O `render` é considerado uma função do ciclo de vida, sendo, entre eles, o único obrigatório. O único local onde não devemos executar uma atualização de estados é dentro do constructor: apenas a atribuição

inicial do nosso estado precisa ser executada no método constructor.



3 - Usando React com requisições HTTP/Ajax & React Hooks

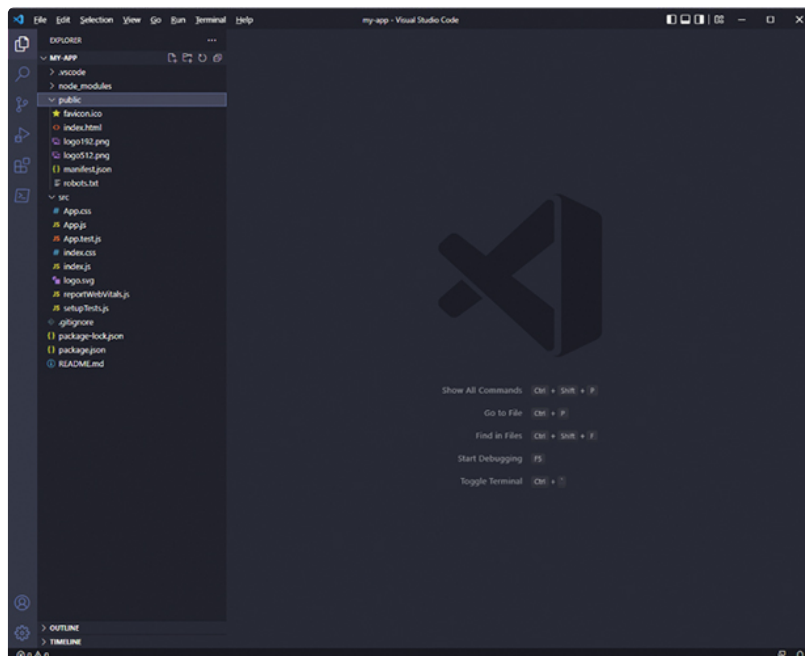
Ao final deste módulo, você será capaz de usar requisições HTTP/Ajax com o novo conceito de React Hooks.

Ambiente Visual Studio Code

Preparando ambiente

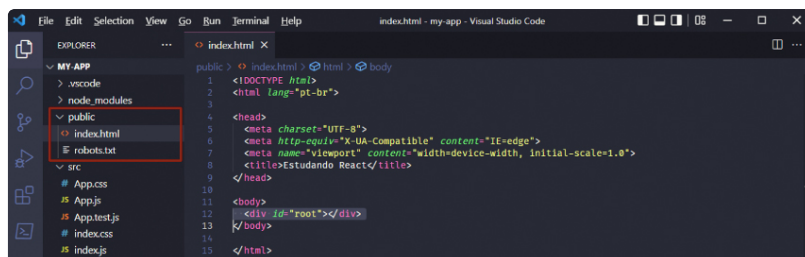
A partir deste módulo, utilizaremos o nosso editor Visual Studio Code. Para isso, vamos criar um projeto, organizar as pastas e importar e exportar componentes, bem como utilizar uma API externa, com suas requisições utilizando Hooks.

Usaremos o ambiente criado anteriormente, modificando alguns arquivos e preparando-o para podermos iniciar o nosso estudo com requisições. Vamos escolher no Visual Studio Code e abrir a pasta do projeto criado (my-app). Veja a estrutura dos arquivos:



Ambiente Visual Studio Code.

O script criou uma pasta `node_modules` na qual estão todos os módulos necessários para a utilização do React. Na pasta `public`, em que está a nossa página inicial (`index.html`) que contém a `<div id="root">` sempre utilizada em nossos exemplos, podemos remover todo seu conteúdo, deixando apenas uma estrutura básica para exibir nosso componente e apagando alguns arquivos que não serão usados.



Pasta public.

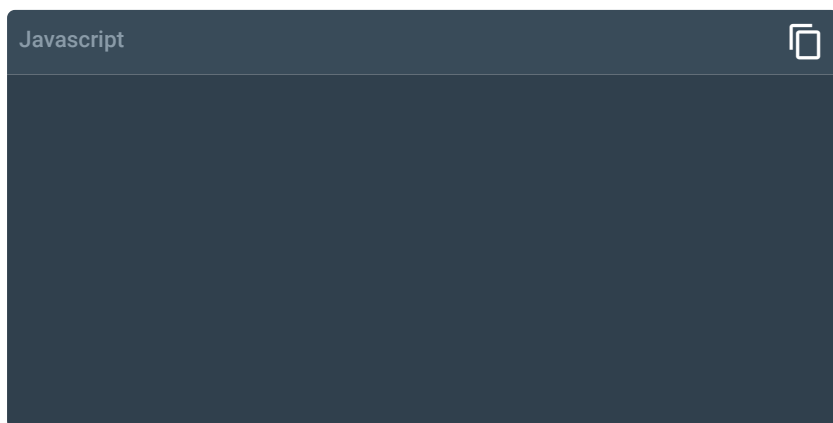
Dentro da pasta `src` fica nosso componente, que está produzindo toda a dinâmica da logo a girar do React. O arquivo que faz esse efeito é o `index.js`, que carrega um único componente, chamado de `App`, cuja lógica se encontra inteiramente em outro arquivo: `App.js`.

Aqui já começamos a ver uma diferença em relação aos exemplos anteriores, pois sempre utilizávamos neles um arquivo único. Abrindo inicialmente o `App.js`, veja o código a seguir:





Os dois primeiros imports foram utilizados para inserir tanto a logo do React que gira na página inicial (linha 8) quanto um arquivo CSS. Na última linha, definiremos qual função terá acesso por outro componente, sem nos esquecermos do parâmetro default. Também removeremos os arquivos que não vamos utilizar. Nosso componente App ficará assim:



No último arquivo a ser modificado, o index.js, identificamos uma composição de componentes – nesse caso, apenas o App. Verifica-se novamente a presença de arquivos de estilização (App.css) e de alguns para testes e reports.





Na linha 4, podemos ver como é feita a importação de um componente seguindo a sintaxe:

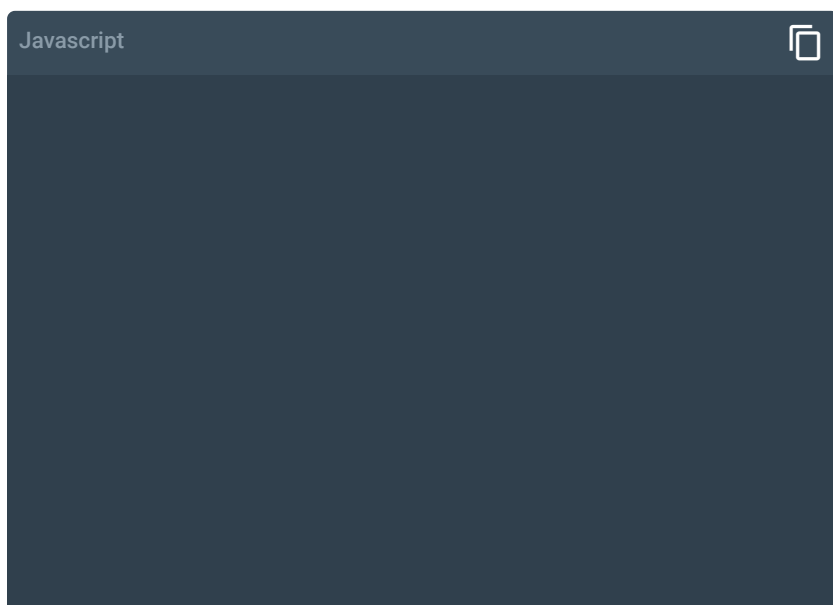
```
import NomeComponente from 'Nome_Arquivo_Componente';
```

Na linha 7, está salvo o nó da árvore DOM a ser renderizada em nosso componente. Já na 8 ocorre a renderização. O modo restrito definido pelo React (linha 9) serve para ativar verificações e os avisos adicionais para os componentes que serão renderizados dentro dele. Tais verificações são executadas apenas no modo de desenvolvimento.

Removendo os arquivos e os códigos que não vamos utilizar, nossa estrutura e nosso código final ficam da seguinte forma:

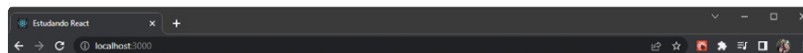
Estrutura do APP.

Observe o código a seguir:





E eis o resultado da página final:



Olá

Uma sugestão para praticar e revisar nossos exemplos: reescreva para o Visual Studio separando as pastas por seção.

Hooks

Apresentação

Antes de ver como são feitas as requisições HTTP em React, precisamos estudar uma nova função responsável por oferecer uma dinâmica diferente, facilitando bastante a manipulação de estados em nossos componentes sem modificar suas hierarquias. Isso fez com que o uso de componentes de classe no dia a dia se tornasse desnecessário.

O [Hooks](#) permite a quebra do nosso componente em funções menores, mantendo seus relacionamentos sem a necessidade de utilizar os métodos de ciclo de vida. Neste material, falaremos sobre alguns dos principais Hooks utilizados, o useState e o useEffect.

Hooks

Tooltip - Hooks

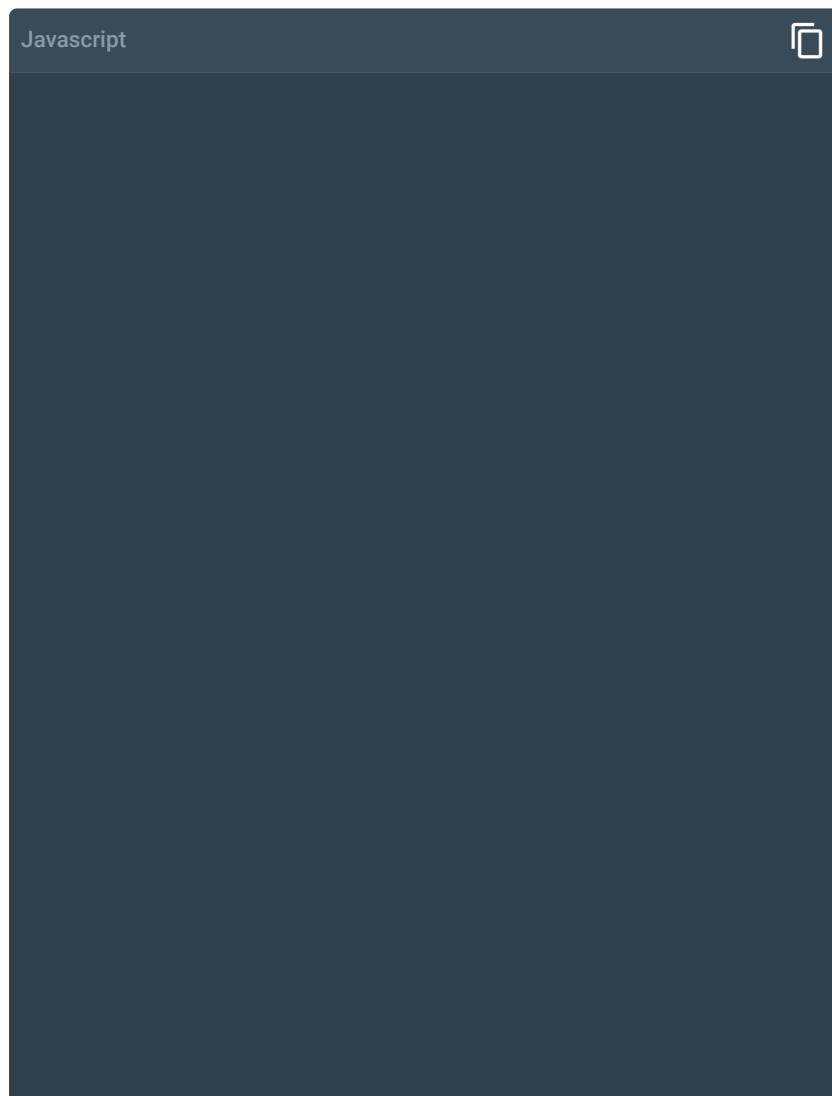
Ganchos

Função useState

Quando utilizamos uma classe, definimos no construtor o estado com this.state com alguns atributos. Para atualizá-los, usamos dentro da classe o this.setState, passando os valores dos atributos que gostaríamos de modificar. O React renderiza nosso componente apenas nos pontos em que eles foram atualizados.

O `useState` é uma função que retorna um par de valores. O primeiro é o valor atual. Já o segundo se trata de uma função que, idêntica ao `setState`, é utilizada para atualizar, só que sem se mesclar com o antigo valor.

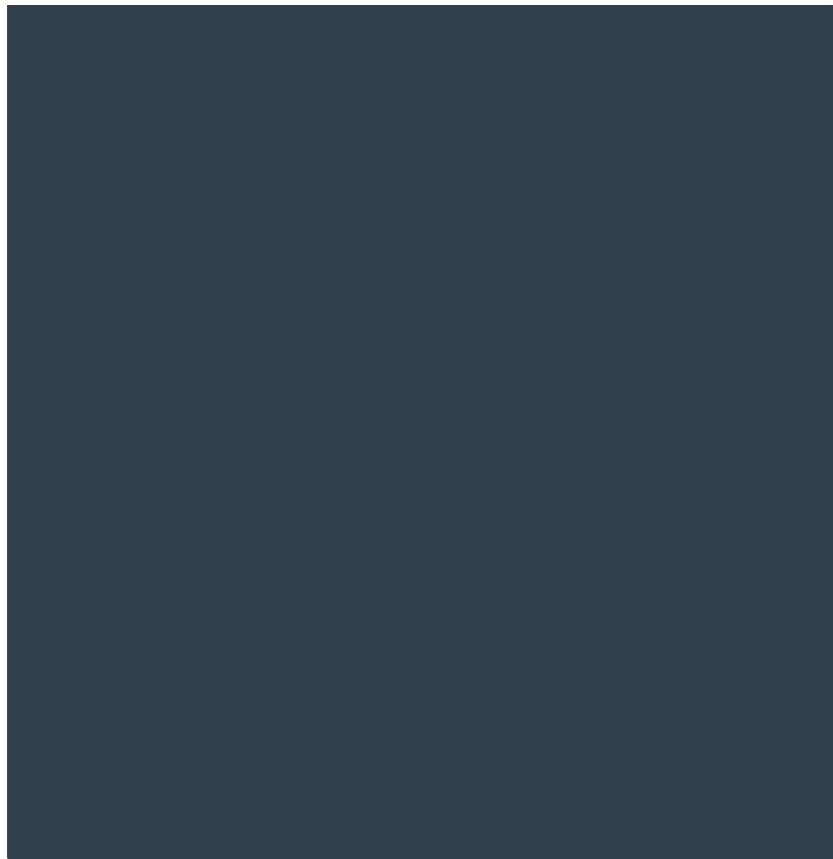
Além de sua sintaxe, observe como o conceito de desestruturação de arrays facilita a atribuição dessas variáveis:



Por convenção, utilizamos sempre um nome para a variável e a função para modificá-la com `set` na frente. É possível usar variáveis ou um array como parâmetro no `useState`.

Para utilizarmos as variáveis, repetiremos (como fizemos em todos os nossos exemplos) as tags `jsx` sem a necessidade de utilizar o `this`. Não se esqueça de importar a função logo no início do código. Vamos demonstrar mais um exemplo de `useState`, incrementando uma variável ao pressionar um botão:





Demonstramos aqui as três formas de utilizar o `setValor`. Use a notação com a qual esteja mais familiarizado, como função ou arrow function.

Saiba mais

Para saber mais detalhes sobre o `useState`, leia a documentação oficial em <https://pt-br.reactjs.org/docs/hooks-state.html>.

Função `useEffect`

Outra função muito utilizada serve, segundo a documentação oficial, para executar efeitos colaterais em nossos componentes funcionais. Isso pode parecer confuso, mas já usamos tal conceito nos componentes de classe. Combinados, os métodos `componentDidMount`, `componentDidUpdate` e `componentWillUnmount` definem o `useEffect`.

Componentes do tipo classe não possuem um método no qual seja preciso executar determinado código a cada vez que haja alguma atualização no DOM. Essa atualização sempre é chamada depois de toda renderização.

Uma grande vantagem da utilização do `useEffect` é que, diferentemente dos métodos que cuidam das atualizações dos componentes de classe, esses efeitos não bloqueiam o navegador ao efetivarem a atualização da tela, pois a maior parte deles não precisa ocorrer de forma síncrona.

No uso do `useEffect`, vimos que são gerados efeitos colaterais: alguns deles não precisam de limpeza; outros, sim. Os efeitos sem limpeza não retornam nada na função. Já aqueles que precisam dela retornam uma função que será executada no momento que aquele componente for desmontado. Veja sua sintaxe:

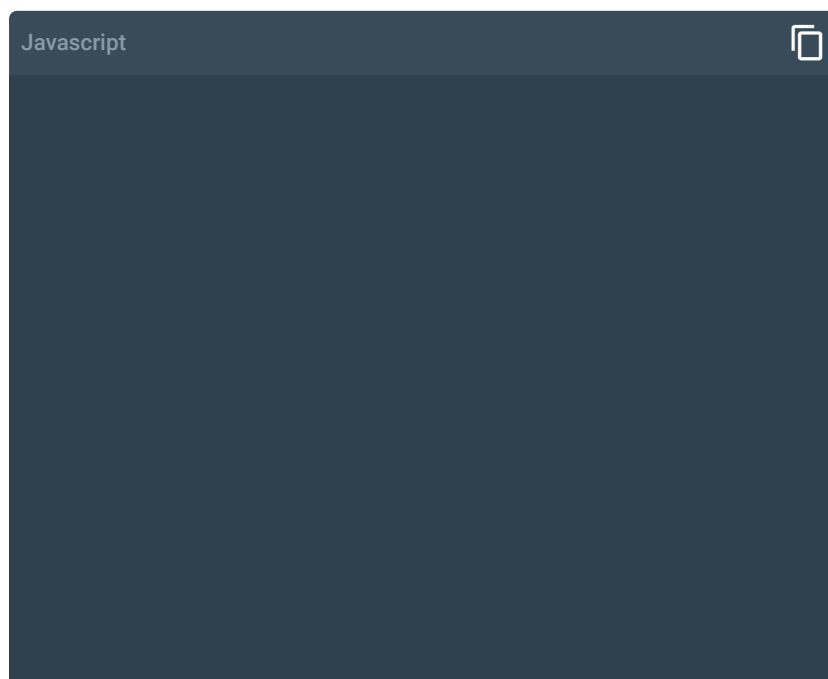


A função que for passada como primeiro parâmetro será executada quando ocorrer alguma atualização em sua dependência.

Exemplo

Se, na função passada, tivermos uma variável contador, quando essa variável for modificada, nosso efeito colateral será executado. Caso seja definido um retorno nesse efeito colateral, ela será chamada quando essa variável de dependência for destruída.

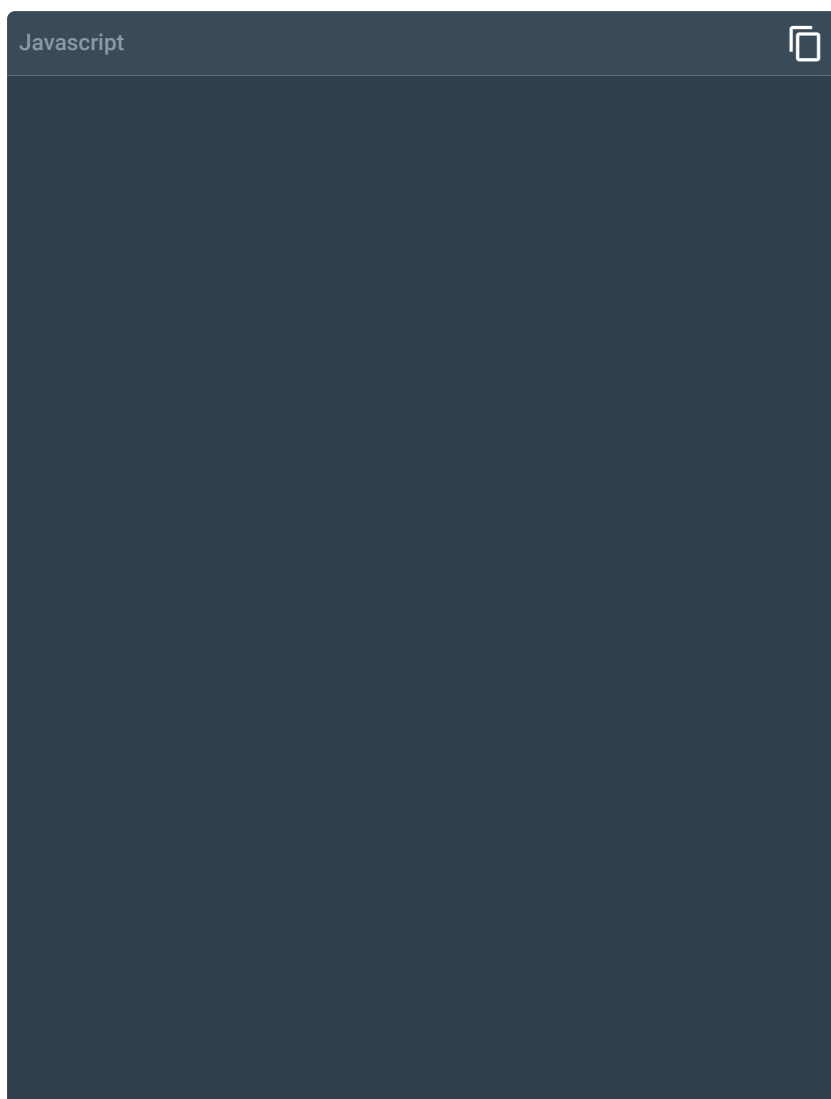
O segundo parâmetro é um vetor dessas dependências. Quando ele está em branco, o React entende que todas as variáveis utilizadas na função estão sendo monitoradas. Caso se queira excluir alguma, aí sim será preciso explicitar cada uma delas nesse vetor. Segue um exemplo do uso dessa função:





Ao se clicar no botão somar, percebe-se uma demora na atualização do título da página. Isso ocorre pelo fato de o React esperar a renderização de toda a tela finalizar para assim executar a função definida dentro do `useEffect`, conforme apresentado na imagem a seguir:

Em `App.js`, modificamos o código para que, após cinco segundos, seja possível remover todos os componentes da tela e ver o retorno da nossa função ser executada:



Se incluirmos o segundo parâmetro da função com um vetor vazio, o nosso título não será atualizado ao clicar em somar:



Veja a seguir a saída esperada:



Ao longo deste estudo, empregaremos o `useEffect` ao estudarmos as requisições. Por elas serem assíncronas, será necessário atualizar nossa aplicação quando recebermos esses resultados das requisições.

Acessando uma API

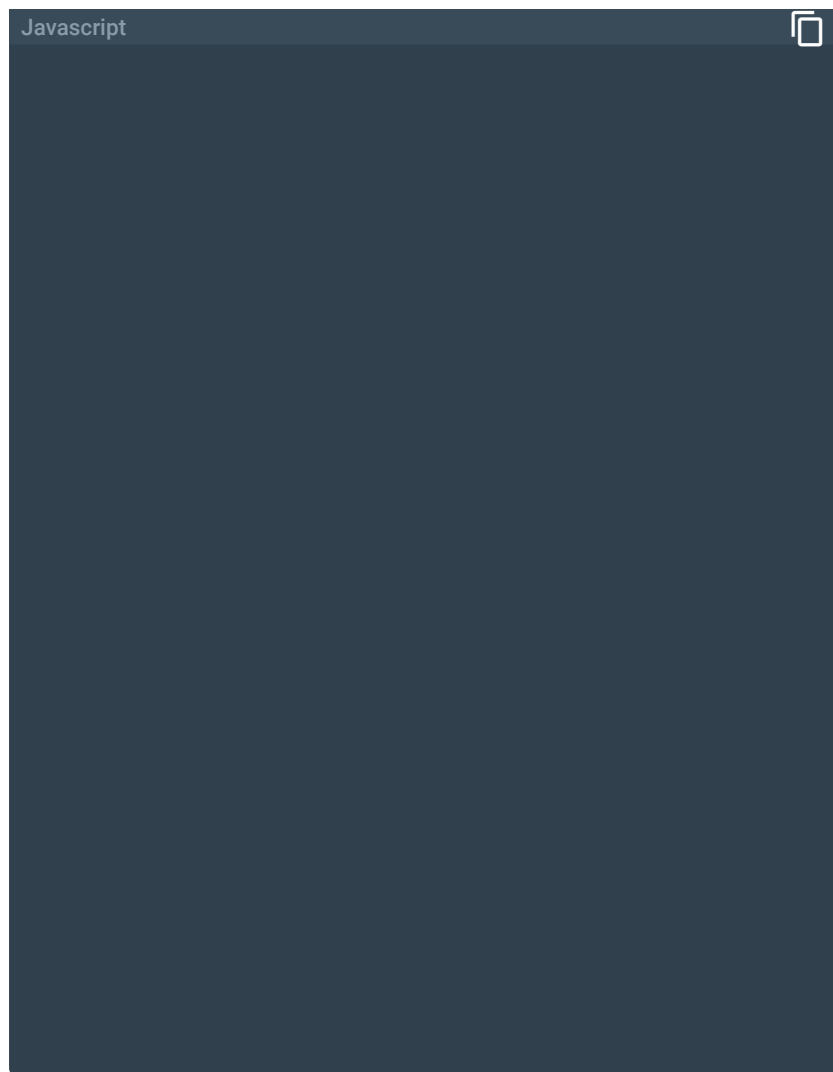
Biblioteca Fetch API

Vamos falar aqui como se deve consumir uma API externa, fazendo as requisições, além de apontar a melhor forma de salvar esses dados. O React permite utilizar diversas bibliotecas para isso. As bibliotecas mais populares são:

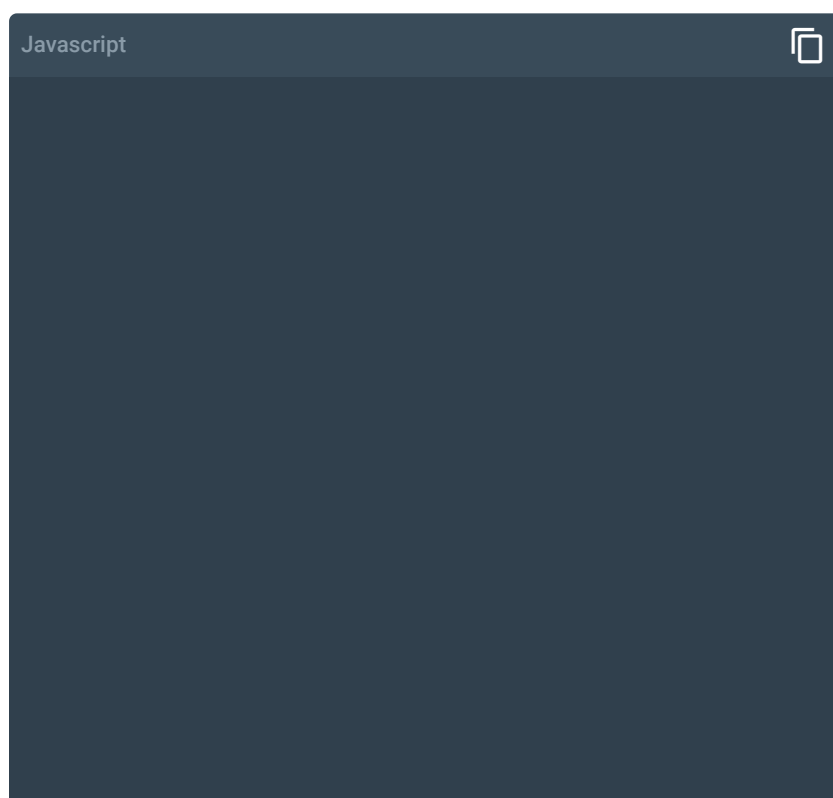
- Axios.
- jQuery Ajax.
- A nativa Fetch.

Como as requisições de API externa são assíncronas, uma boa prática é tratar dessa função com o `useEffect`. Inicialmente, utilizaremos uma API gratuita (<https://jsonplaceholder.typicode.com>) que gera dados no formato JSON e conta com diferentes tamanhos e formatos de dados.

Em nosso exemplo, usaremos os dados de 10 usuários (<https://jsonplaceholder.typicode.com/users>). Segue parte do JSON:

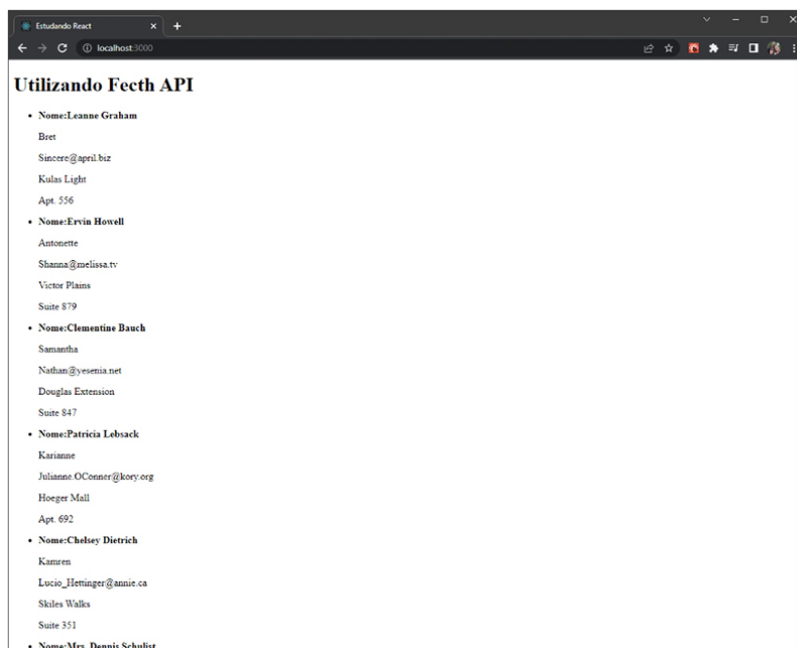


Para fins práticos, usaremos apenas uma variável para salvar os dados retornados. Com o auxílio de map, percorreremos a lista, associando a chave da tag para cada ID e preenchendo com os dados que escolhemos exibir.





Eis o resultado:

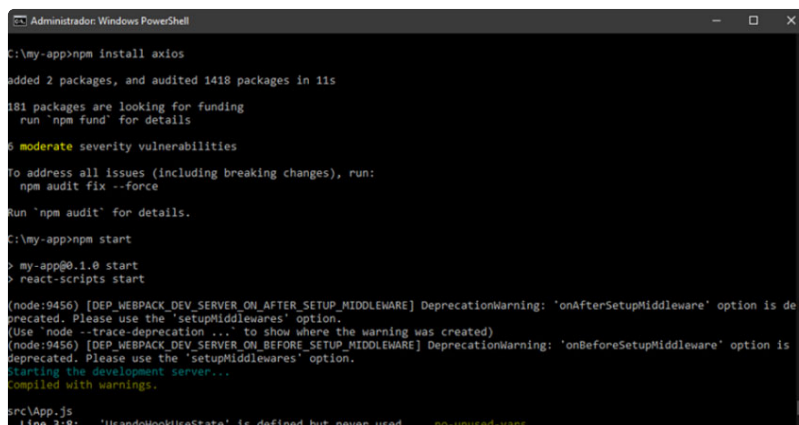


Biblioteca Axios

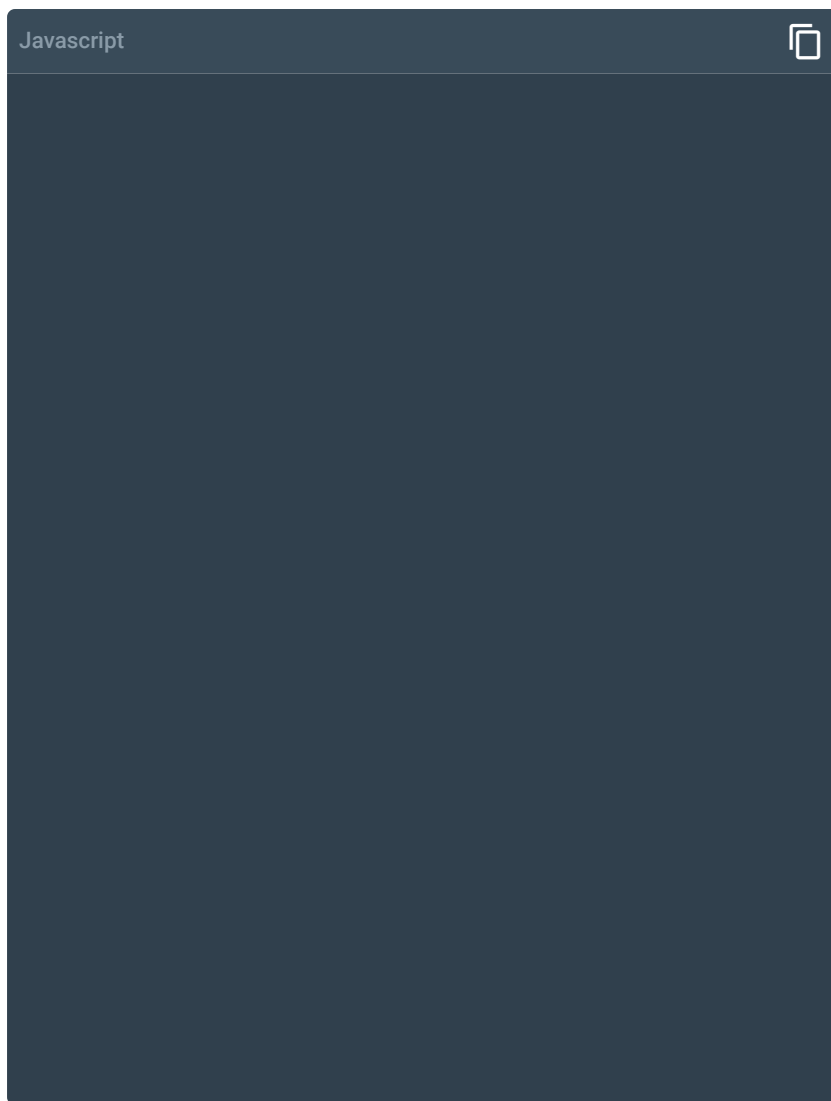
Primeiramente, é preciso instalar o módulo necessário para a utilização do Axios. Como vimos anteriormente, temos de abrir nosso terminal e, dentro da pasta de nossa aplicação, executar (como determina a documentação dele em <https://github.com/axios/axios>) este comando: `npm install axios` Observe a imagem a seguir:

`npm install axios`

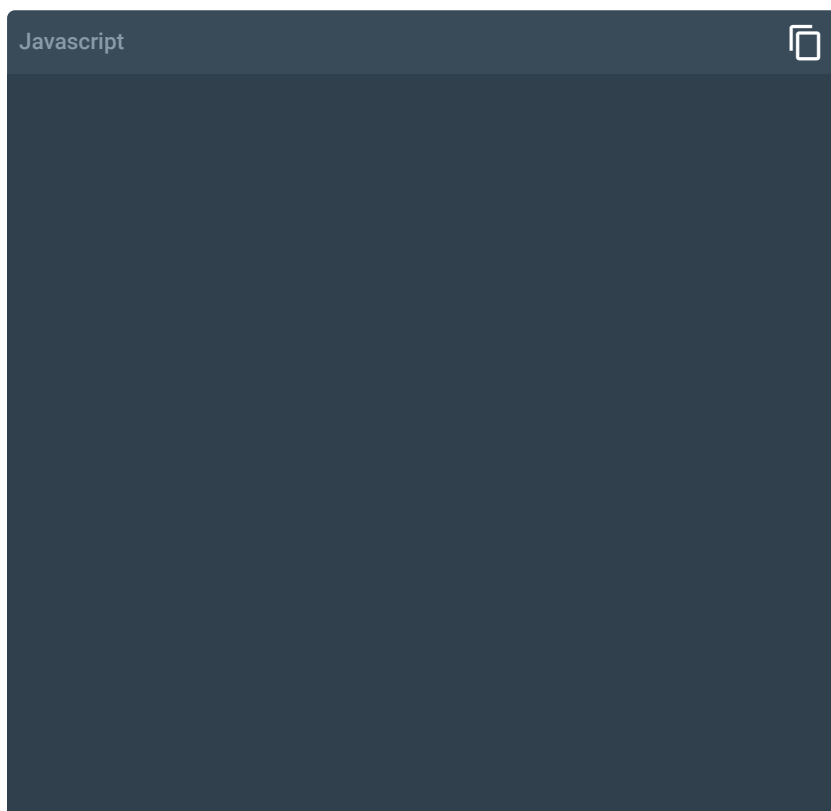
Observe a imagem a seguir:



Utilizaremos outro exemplo da nossa fonte de dados, só que, dessa vez, com 100 postagens (<https://jsonplaceholder.typicode.com/posts>):

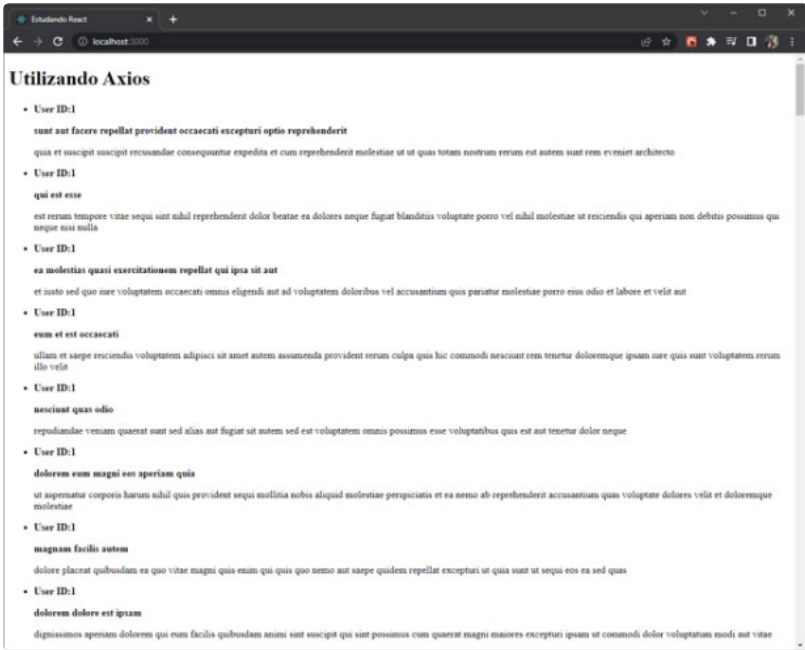


Na estrutura, existem diferentes postagens para o mesmo usuário. Veja o nosso código para utilizar essa API e visualizar o resultado de forma correta:





Eis o resultado:



Biblioteca jQuery

Novamente, é preciso instalar o módulo necessário para sua utilização. Isso será feito de forma idêntica, abrindo o terminal e, dentro da pasta de nossa aplicação, executando (conforme a documentação do jQuery em <https://jquery.com/download>) o seguinte comando: `npm install jquery`

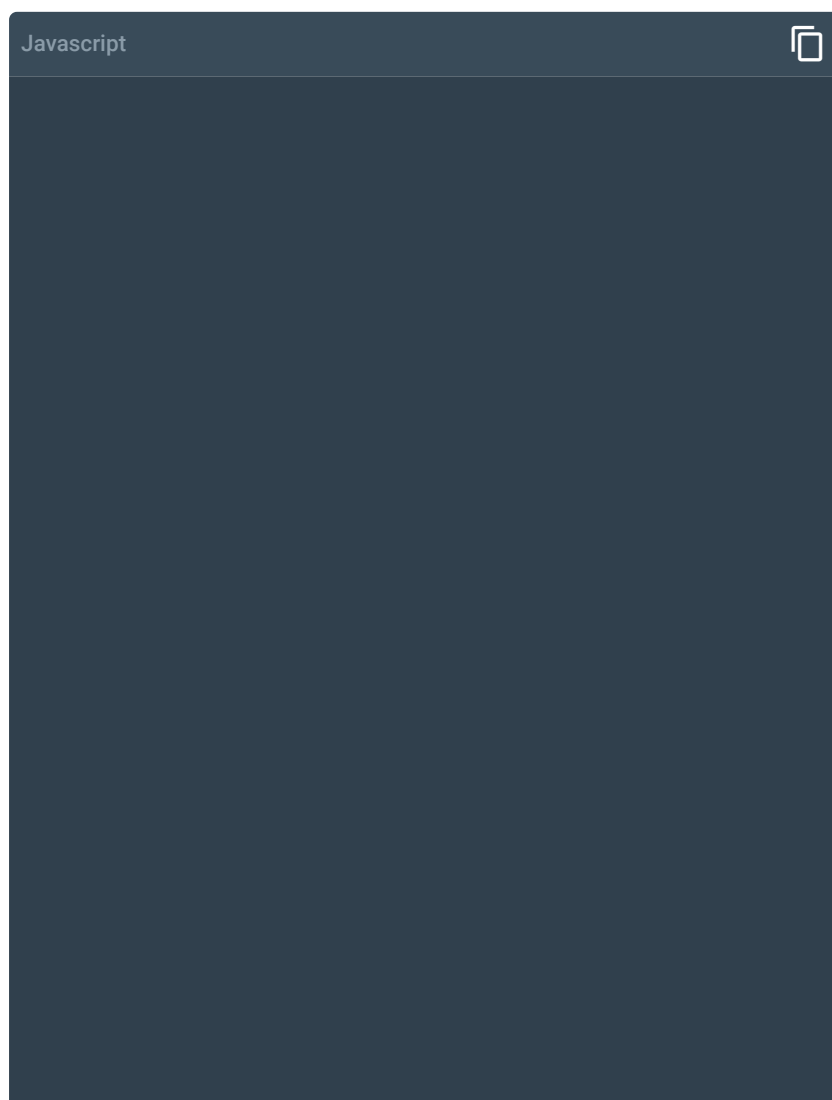
Observe a imagem a seguir:

Agora vejamos um exemplo com 500 comentários (<https://jsonplaceholder.typicode.com/comments>):





Sem grandes mudanças, nosso código ficou:



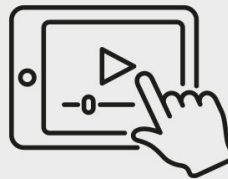
É importante observar que nenhum controle de dados foi utilizado. Caso o resultado retorne nulo, será preciso tratar de cada caso, ou seja, cada biblioteca trata de uma forma o retorno desses valores.



Acessando uma API com React

Neste vídeo, apresentaremos os principais conceitos sobre o acesso a uma API com React.

Para assistir a um vídeo sobre o assunto, acesse a versão online deste conteúdo.



Falta pouco para atingir seus objetivos.

Vamos praticar alguns conceitos?

Questão 1

O Hooks surgiu nas versões mais recentes do React 16.8, permitindo a utilização dos estados e de outros recursos sem a necessidade de criar classes. Qual das alternativas a seguir não está correta em relação à utilização de Hooks em nossas aplicações?

A

Hooks são funções que lhe permitem “ligar-se” aos recursos de state e ciclo de vida do React a partir de componentes funcionais.

B

O useState é chamado de dentro de um componente de classe para adicionar states locais a ele.

C

O useState possui mais de um argumento, que é basicamente o estado inicial que queremos mais tarde atualizar com setState.

D

O useEffect não permite executar efeitos colaterais em componentes funcionais.

E

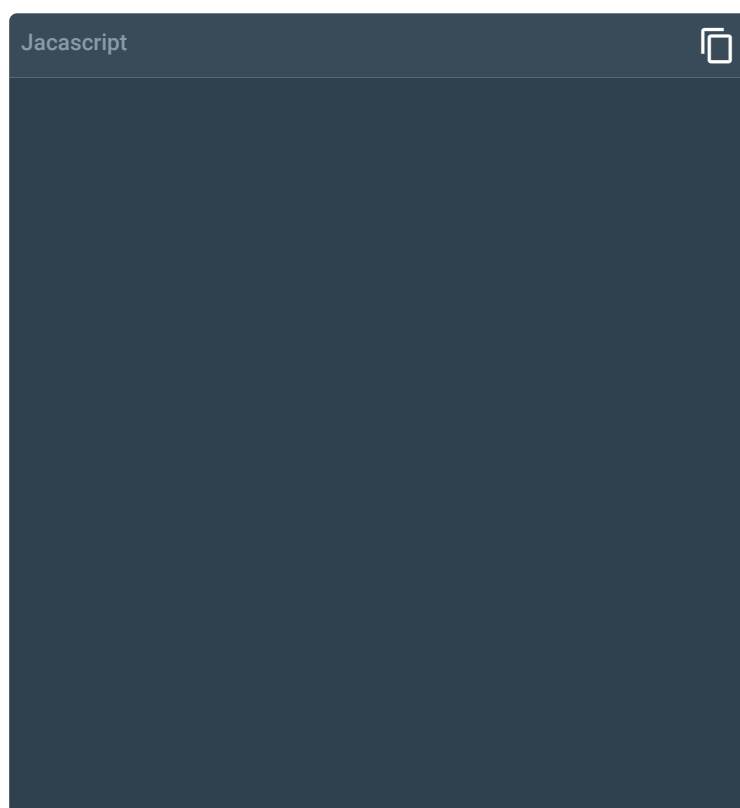
Uma das desvantagens de utilizar useEffect é que o navegador não entra em um estado de bloqueio ao atualizar a tela. Isso ocorre devido aos efeitos colaterais agendados pelo useEffect.

Parabéns! A alternativa A está correta.

Hooks são funções que permitem o uso de binds com recursos de state e ciclo de vida a partir de componentes funcionais.

Questão 2

Uma forma de usar requisições assíncronas em React é utilizar o Hook `useEffect`, auxiliando na resposta dessas requisições e executando alguma rotina desejada. O código abaixo não gera erro de compilação, porém, algo acontece: logo em seguida, a aplicação para de funcionar. Qual seria o motivo para isso?

**A**

A URL está incorreta, pois toda requisição externa exige parâmetros.

B

Não podemos iniciar nosso estado inicial com um vetor vazio.

Ao não utilizarmos o segundo parâmetro na função `useEffect`, estamos avisando ao React que todos os

C

estados dentro da função criada no primeiro parâmetro serão monitorados.

D

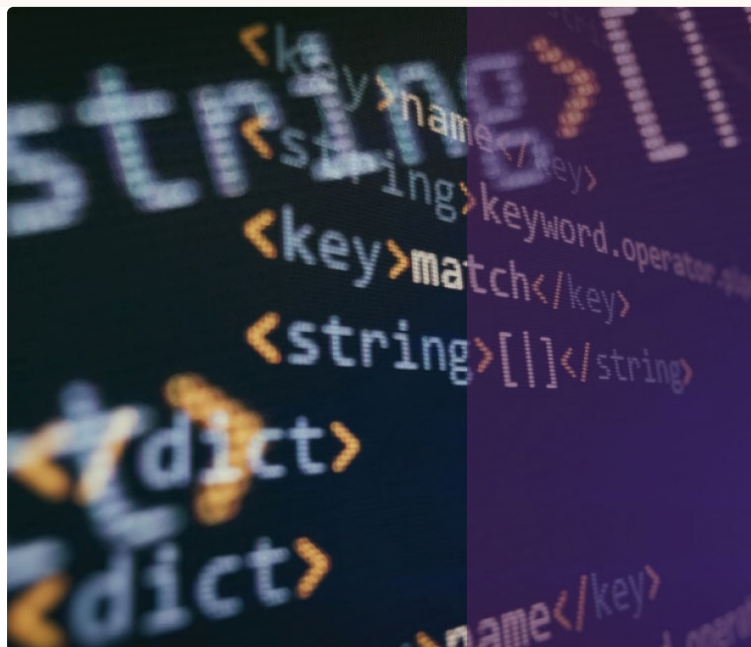
A ausência de chaves ao se utilizar a tag provoca o erro em nossa compilação.

E

Precisamos utilizar o this, o qual, por sua vez, usa o método map().

Parabéns! A alternativa C está correta.

Ao não utilizar o segundo parâmetro, o React considerou que qualquer mudança no estado atual seria um efeito colateral (dados). Isso gera um loop: a função é novamente executada, já que os dados foram alterados. Após algumas chamadas, o servidor do GitHub identifica um potencial risco de ataque HTTP Flood, bloqueando nosso IP e tornando os dados indisponíveis. Com isso, a nossa aplicação não funciona corretamente.



4 - Usando rotas e Redux

Ao final deste módulo, você será capaz de traçar rotas com React Router e o controle da camada de negócio com React Redux.

Biblioteca React Route

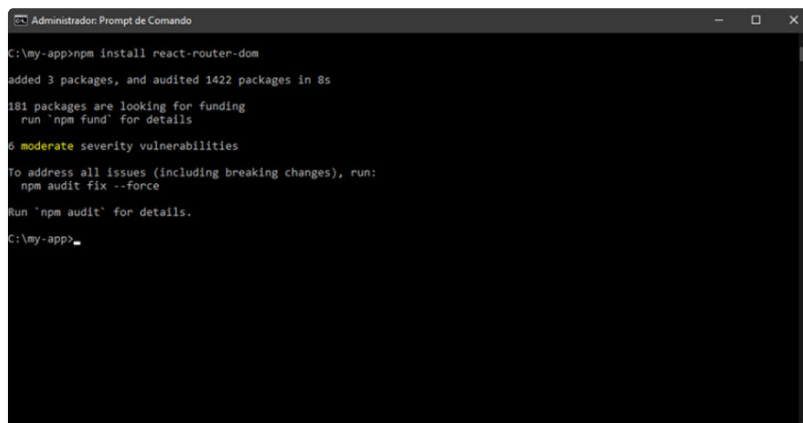
Instalação

Para finalizar este estudo, usaremos duas extensões muito importantes no React: React Router e React Redux. O Router é uma poderosa biblioteca de roteamento que ajuda adicionar novas telas e fluxos para nossa aplicação de uma forma rápida, mantendo a URL sincronizada com estado atual da nossa aplicação. Já o Redux surgiu devido à dificuldade que as aplicações possuíam ao compartilharem informações persistentes entre componentes.

Como qualquer outra biblioteca, precisamos instalar os módulos necessários para sua utilização. Seguindo a documentação oficial (<https://reactrouter.com/docs/en/v6/getting-started/installation>), digitaremos este comando em nosso terminal:

```
npm install react-router-dom
```

Observe a imagem a seguir:

A screenshot of a Windows Command Prompt window titled "Administrador: Prompt de Comando". The terminal shows the command `C:\my-app>npm install react-router-dom` being executed. The output indicates that 3 packages were added and 1422 packages were audited in 8 seconds. It also shows a message about 181 packages looking for funding and a warning about moderate severity vulnerabilities. The prompt returns to `C:\my-app>` after the installation is complete.

```
Administrador: Prompt de Comando
C:\my-app>npm install react-router-dom
added 3 packages, and audited 1422 packages in 8s
181 packages are looking for funding
  run `npm fund` for details
0 moderate severity vulnerabilities
To address all issues (including breaking changes), run:
  npm audit fix --force
Run `npm audit` for details.
C:\my-app>
```

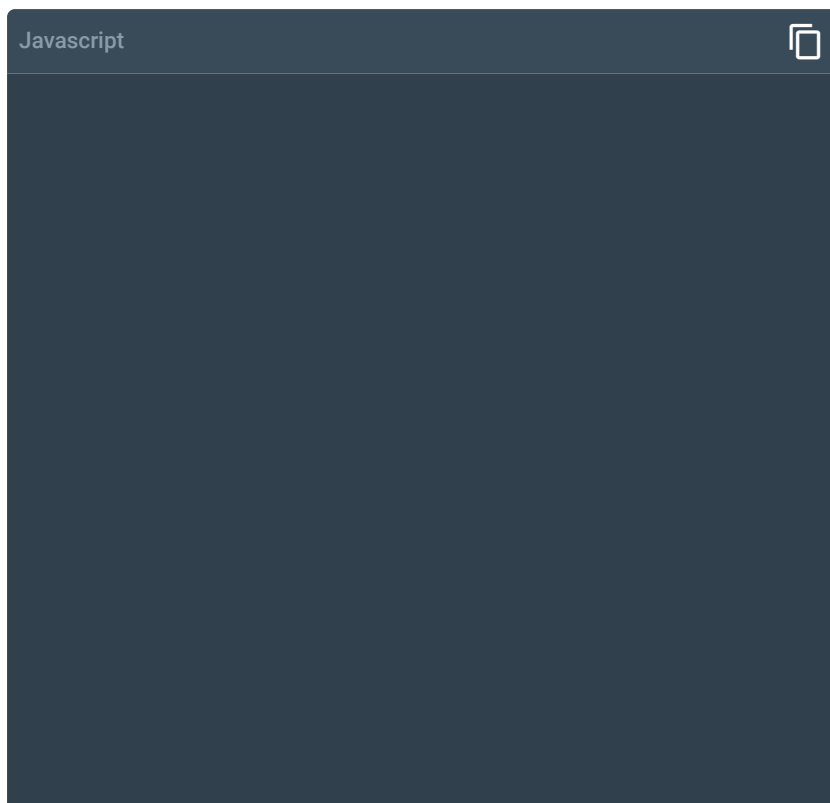
Após sua instalação, a biblioteca disponibiliza quatro elementos novos para se trabalhar:

- `BrowserRouter`;
- `Routes`;
- `Route`;
- `Link`.

Descreveremos a seguir cada um deles, abordando ainda suas sintaxes e funcionalidades.

BrowserRouter

O primeiro elemento possui uma sintaxe bem simples. Basicamente, ele encapsula nossa aplicação na origem de tudo, ou seja, index.js:



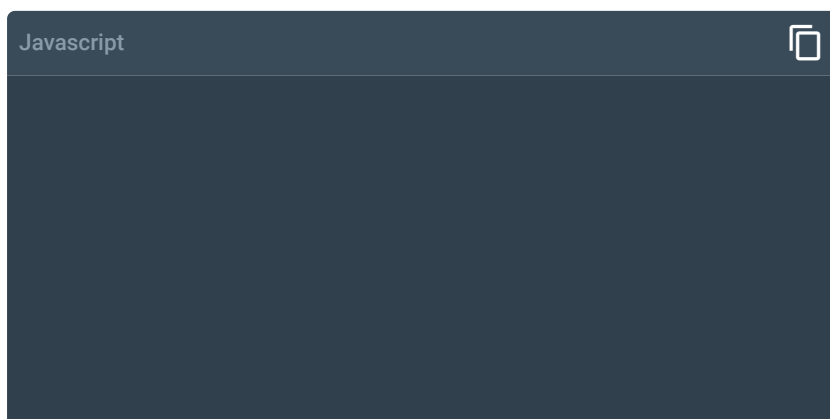
Nesse momento, a nossa aplicação não sofreu nenhuma mudança visual, só que agora podemos trabalhar com as rotas. Todas as rotas deverão ser declaradas dentro desse escopo; caso contrário, elas não serão reconhecidas pela aplicação.

Atenção!

Não deve haver outro BrowserRouter no projeto.

Routes

Ele é usado para encapsular todas as rotas. Sempre que o declararmos, a rota será buscada; assim que ela for localizada, a busca se verá interrompida. Sua sintaxe é simples, encapsulando as rotas em si:



Route

O mais importante elemento entre todos que estamos estudando, o Route possui a responsabilidade de renderizar a interface, seja ela uma página ou um componente. É possível declarar diferentes rotas contendo o respectivo caminho, o componente e, caso necessário, até o props. Sua sintaxe é a seguinte:

```
<Route path="sobre" element={<About />} />
```

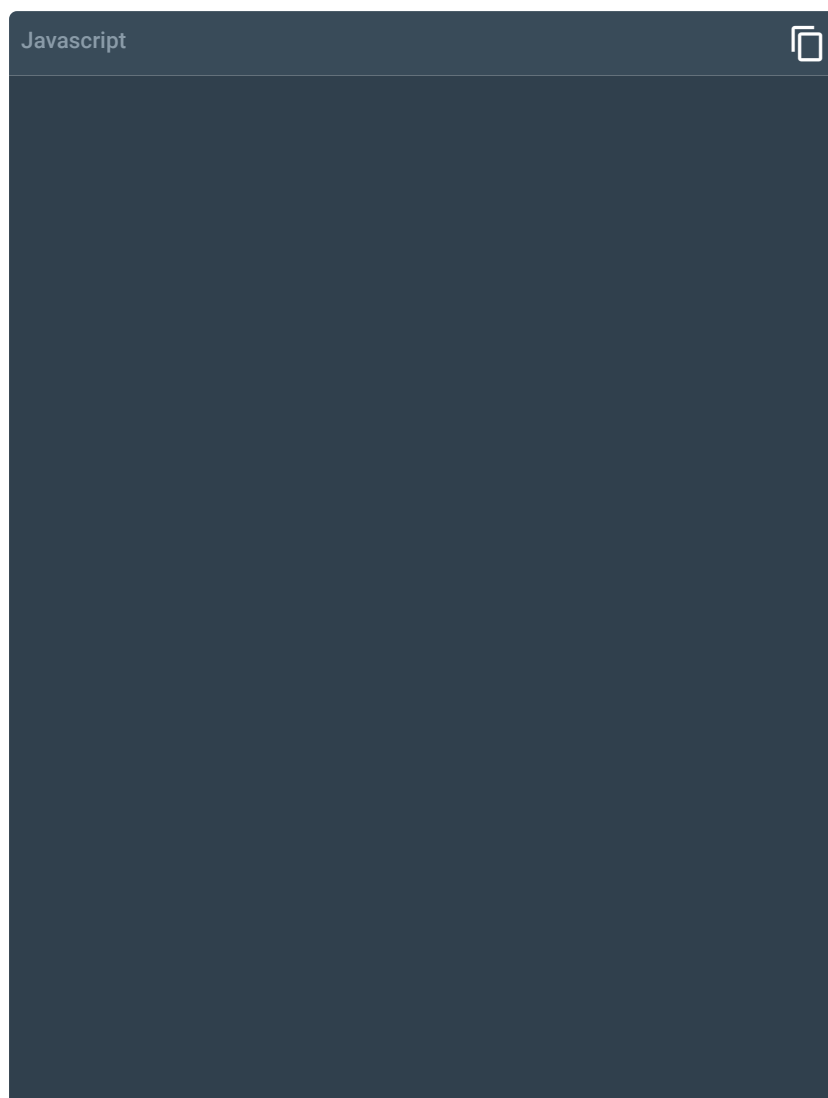
Link

Permite que haja a navegação em si. Eis a sua sintaxe:

```
<Link to="/about"><Sobre>/Link</Link>
```

Resultado Final

Modificando nosso App.js para criar nossas rotas, veja agora o resultado final com todos os exemplos estudados até este momento:



Observe que configuramos uma rota para Home, sendo o elemento e o caminho vazios (linhas 18 e 29), necessários apenas para eliminar as mensagens de aviso no console do navegador. Caso você não tenha

reparado, saiba que o histórico funciona automaticamente, permitindo o retorno à página anterior.

Biblioteca React Redux

Instalação

A Biblioteca React Redux é uma biblioteca oficial de vinculação da Redux UI para o React, embora ela também possa ser utilizada por diferentes frameworks, como Angular, Vue e Ember, por exemplo, o que mostra sua independência.

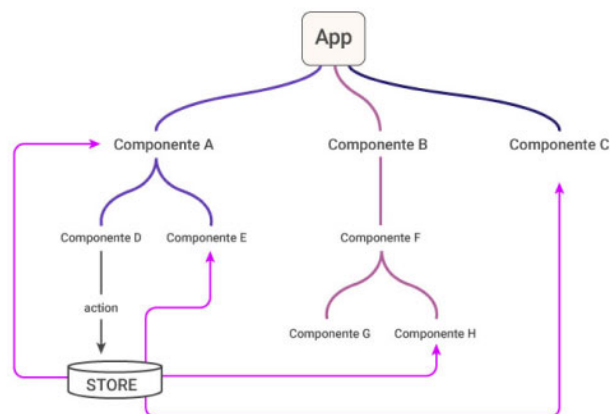
No React, por padrão, qualquer atualização em algum componente executará uma nova renderização de todos os componentes dentro daquele nó da árvore DOM, o que é custoso. Além disso, se os dados de determinado componente filho não forem alterados, a nova renderização fará um esforço desnecessário, pois o resultado da tela será o mesmo.

O React Redux implementa diversas otimizações, pulando renderizações desnecessárias e aumentando a performance da nossa aplicação. Um problema que os componentes enfrentavam se dava na passagem de props entre os componentes.

Veja a imagem a seguir e observe o percurso que um valor de props do componente D precisa fazer ao chegar ao seu destino, o componente H.

Passagem de props entre componentes.

O Redux trouxe uma forma de centralizar essas informações, ficando responsável também por repassá-las para o componente que precisar delas. Observe as imagens a seguir:



Organização das informações.

Breve resumo do fluxo dos dados em Redux.

Explicaremos agora alguns conceitos para a utilização do Redux de forma simples. Com um formulário, vamos simular o login de um usuário e transitar com esse usuário em outro componente.

Antes disso, porém, precisamos instalar algumas bibliotecas que permitem a utilização de certos métodos que auxiliam e agilizam nossa codificação. O intuito desta seção, afinal, é aprender a utilizar o Redux com a React.

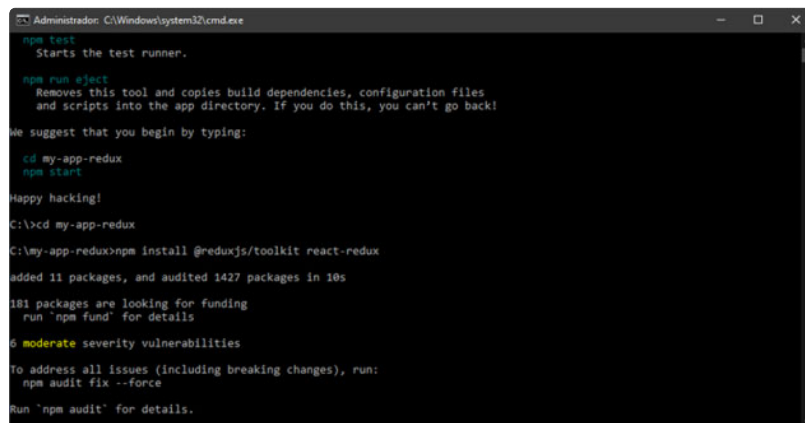
Saiba mais

Para um estudo mais detalhado sobre o Redux, leia sua documentação oficial: <https://react-redux.js.org/>.

Executaremos alguns comandos para a correta instalação das bibliotecas:

```
npm install react-redux
```

```
npm install @reduxjs/toolkit react-redux
```



```
Administrator: C:\Windows\system32\cmd.exe
npm test
  Starts the test runner.

npm run eject
  Removes this tool and copies build dependencies, configuration files
  and scripts into the app directory. If you do this, you can't go back!

We suggest that you begin by typing:

  cd my-app-redux
  npm start

Happy hacking!
C:\>cd my-app-redux
C:\my-app-redux>npm install @reduxjs/toolkit react-redux
added 11 packages, and audited 1427 packages in 10s
181 packages are looking for funding
  run `npm fund` for details
6 moderate severity vulnerabilities
To address all issues (including breaking changes), run:
  npm audit fix --force
Run `npm audit` for details.
```

Finalizando a instalação e iniciando nosso servidor, verifica-se que estão disponibilizados os métodos que ajudam na manipulação dos estados. Com o auxílio do fluxo de dados do Redux, iniciaremos agora nossa implementação, explicando cada bloco e trecho do código.

Store

Objeto onde salvaremos todos os estados que desejarmos compartilhar de forma global em nossa aplicação. Sempre que qualquer componente precisar ler um estado, basta consultar tal objeto e, assim, resgatar o conteúdo.

Para alterar o estado, porém, precisamos utilizar outro conceito: as actions.

Actions

São ações enviadas do nosso componente: elas acionam os reducers e informam qual função terá de ser executada e qual valor que ela usará como parâmetro.

Resumindo

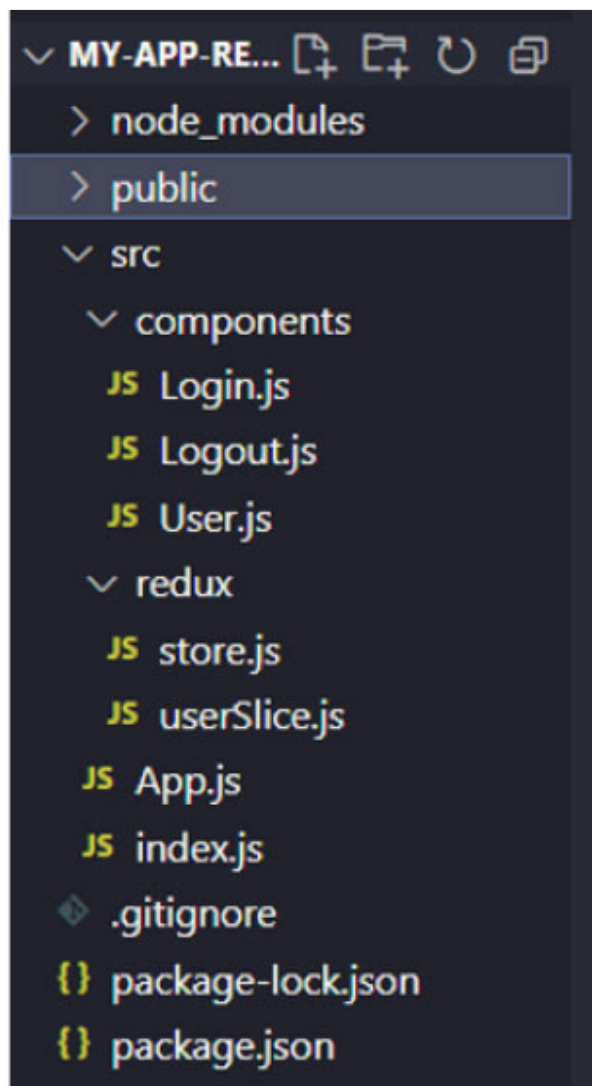
É como se um comando fosse enviado da seguinte forma: "O componente G quer alterar o estado B usando a função F".

Reducers

Depois que os dados são tratados, a atualização do estado é realizada e todos os outros componentes que precisam usar esse novo estado são avisados.

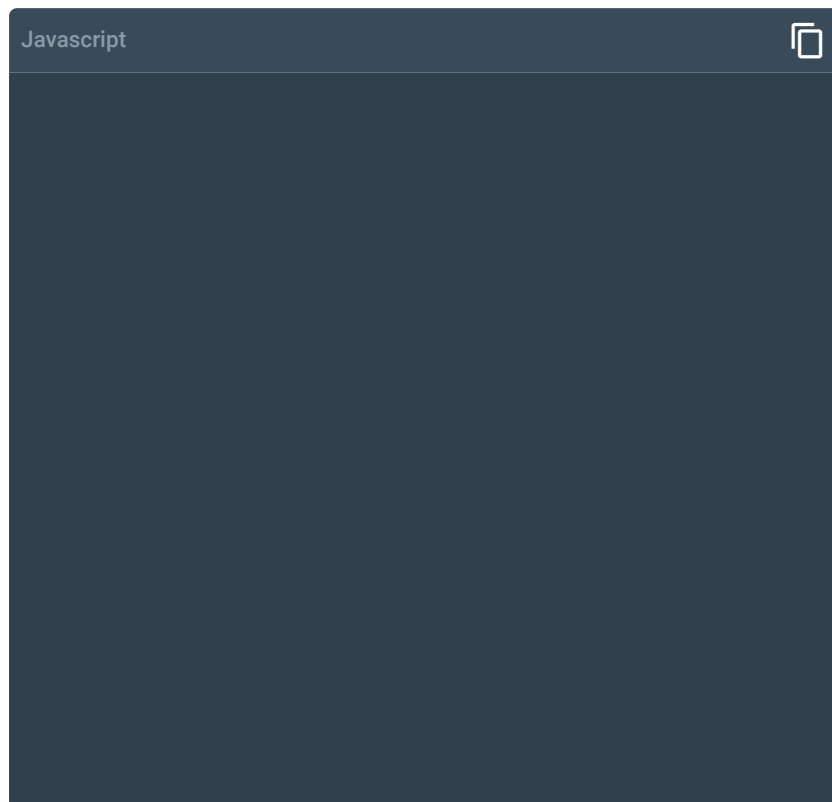
Slicer

A toolkit instalada nos trouxe um método que facilita a manipulação da lógica por trás de reducers e actions, juntando-os em um único arquivo. Veja como ficou a estrutura de pastas do nosso projeto:

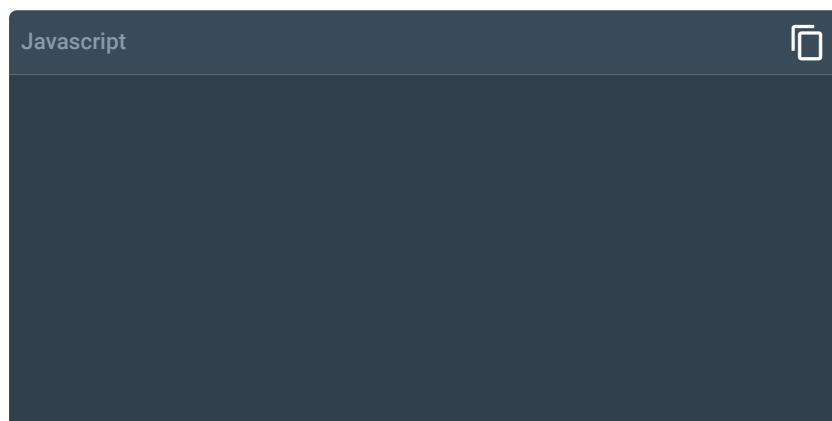


Estrutura do projeto.

Componentes dentro da pasta components são os arquivos que cuidam de actions, reducers e store dentro da pasta redux e, na raiz do projeto, aqueles que iniciam a nossa aplicação. Uma primeira modificação é encapsular com uma tag Provider o nó raiz (nesse caso, App):



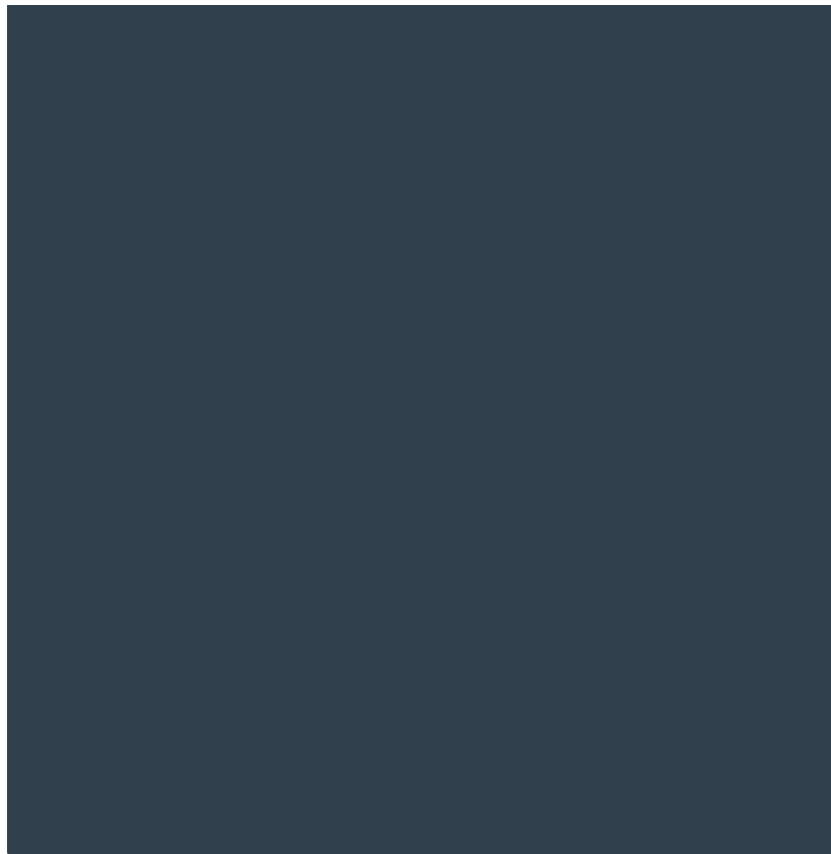
É necessário passar como parâmetro nossa store, que será definida da seguinte forma:



O método configureStore, que veio da biblioteca, recebe um parâmetro do tipo reducer. No nosso exemplo, ele é nomeado como user.

Definiremos agora nosso slicer com as lógicas de que precisamos para manipular o usuário com reducers e actions:





O slice é criado ao chamar o método `createSlice`, que recebe como primeiro parâmetro um nome. O segundo parâmetro será nosso estado inicial; o terceiro, nossos reducers.

Definimos dois reducers (`changeUser`), em que o primeiro parâmetro é o estado atual e o segundo, uma action que desestruturamos, obtendo apenas o payload da função. Seu retorno sempre deve conter todos os estados anteriores (`...state`) e os novos estados.

No logout, usamos apenas o estado atual sem a necessidade de definir uma action de retorno. Seu retorno contém todos os estados anteriores (`...state`) e os novos estados. Ao final do código, com o auxílio do slice, podemos exportar nossas actions (linha 19) e os reducers (linha 23).

Componentes Login e Logout

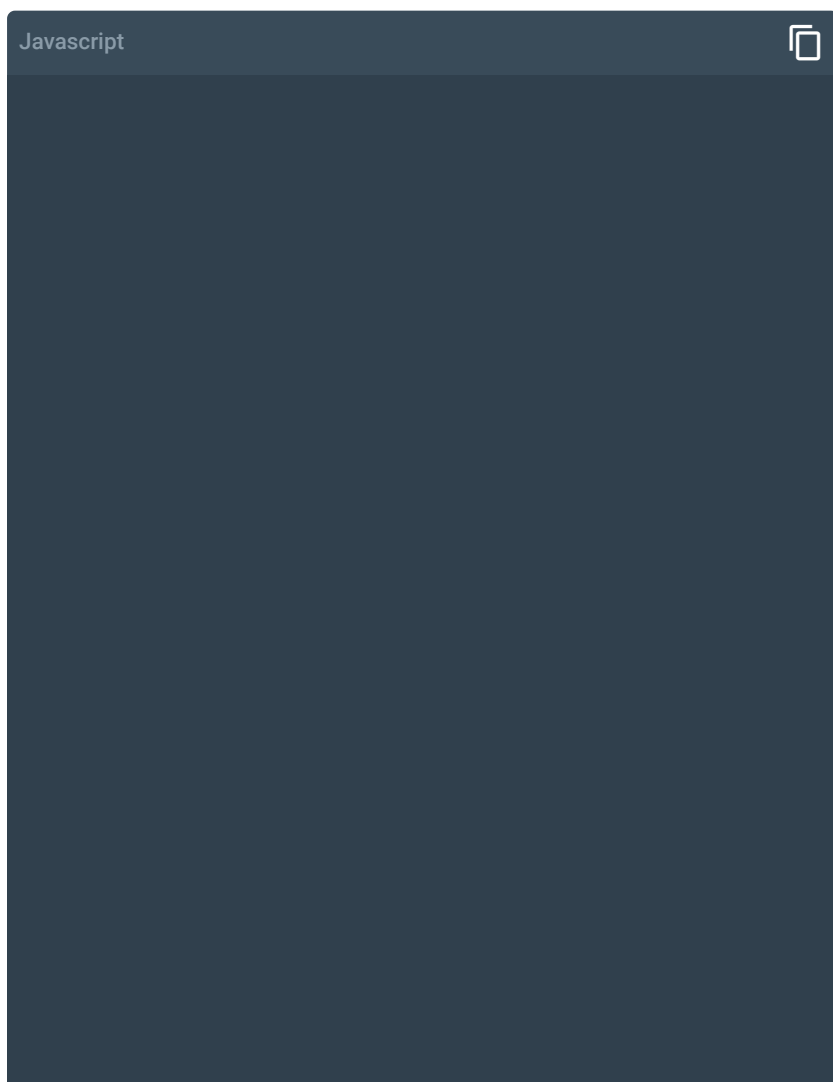
O próximo código a ser verificado é o nosso componente Login, que é responsável por mudar o usuário ao clicar no botão Login da nossa aplicação.



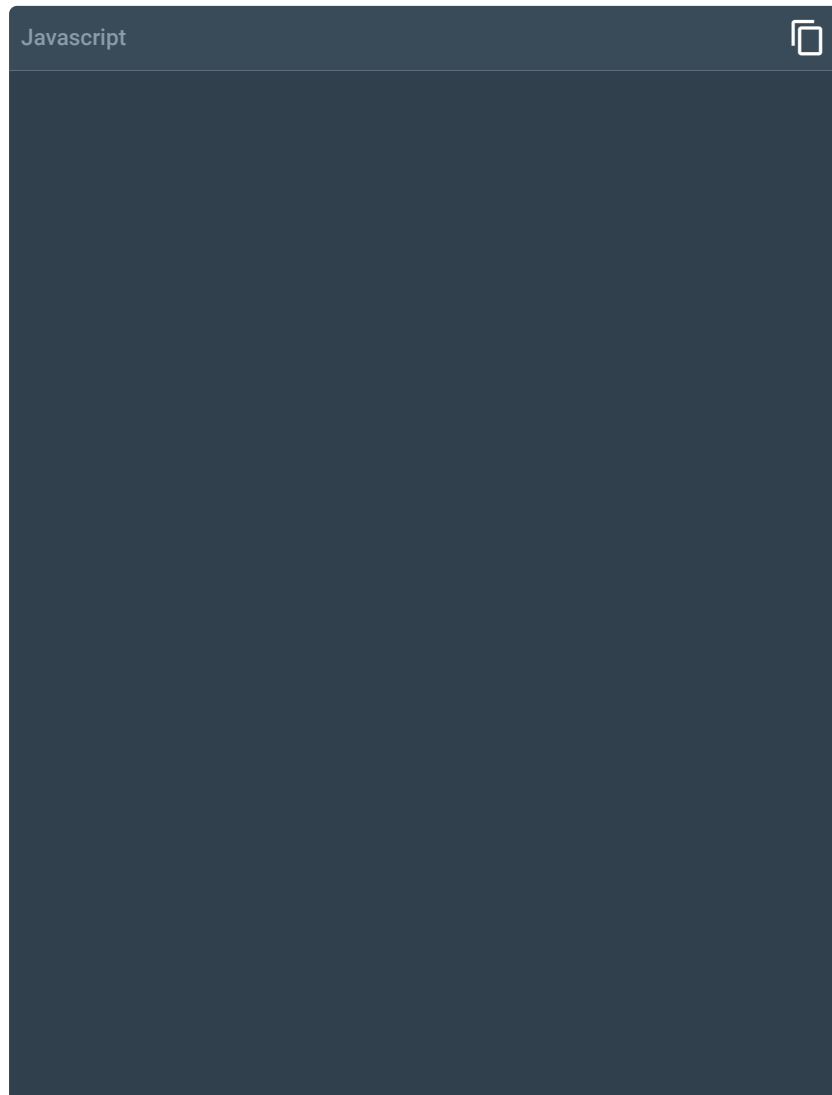


O `userDispatch` (linha 7) é um Hook do React Redux usado para a execução de nossas actions. Sua sintaxe é simples, recebendo como parâmetro uma action – no nosso código, o `changeUser(name)`.

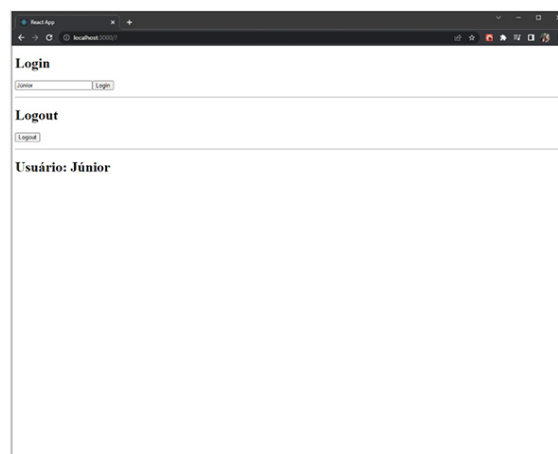
O Logout seria bem semelhante ao Login, utilizando a mesma função `userDispatch`. Observe nosso código:



Para finalizar, veja o nosso componente User. Utilizando outro Hook do React Redux, o useSelector, que recebe uma função que retém todo o estado da aplicação, novamente o desestruturamos para pegar apenas o nome do usuário.

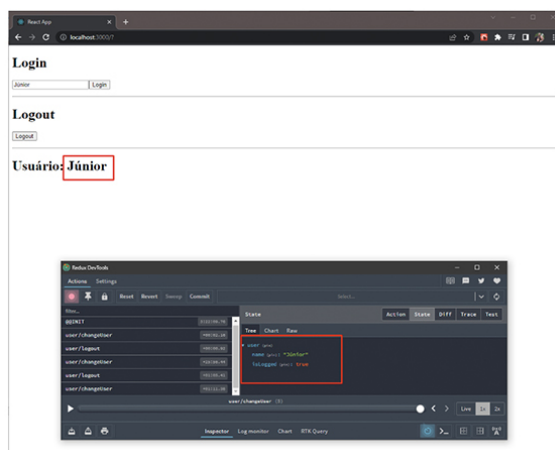


Eis o resultado da aplicação:

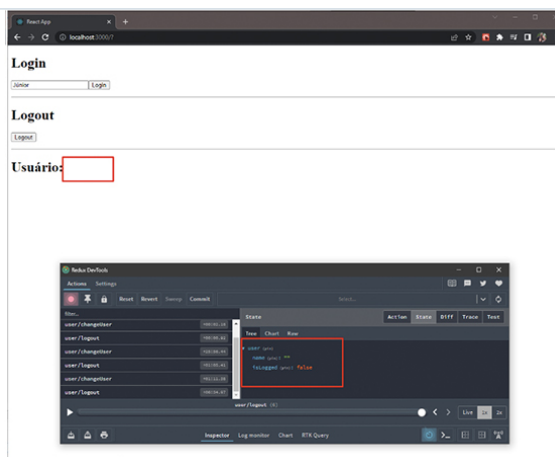


Tela de login.

Com ajuda de ferramentas externas (Redux DevTools), podemos depurar os estados da nossa aplicação no navegador, o que comprova o funcionamento do Redux.



Usuário logado.



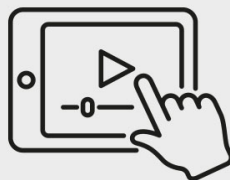
Usuário não logado.



Criação de telas de login e logout com React

Neste vídeo, apresentaremos como criar formulários de login e logout utilizando o React.

Para assistir a um vídeo sobre o assunto, acesse a versão online deste conteúdo.



Falta pouco para atingir seus objetivos.

Vamos praticar alguns conceitos?

Questão 1

O React Router é uma biblioteca que permite navegar entre outras interfaces em nossa aplicação. Neste material, falamos exclusivamente da versão mais recente (V6), modificando algumas sintaxes com novos recursos. Em relação aos principais

componentes dessa versão nova do Router, assinale a alternativa incorreta.

A

O Route é o componente menos importante, podendo ter apenas uma única rota contendo o caminho e o componente, sem permitir a passagem de parâmetros.

B

O link é o elemento que permite ao usuário navegar para outra página.

C

O BrowserRouter permite encapsular nossa aplicação. É necessário realizar isso na primeira chamada na raiz de toda a nossa árvore de componentes.

D

O switch foi substituído pelo Routes com algumas peculiaridades, como, por exemplo, removendo o componente Redirect dentro dele.

E

Existe uma pilha de histórico, o qual, por padrão, já é definido, não sendo necessária uma codificação extra para isso.

Parabéns! A alternativa A está correta.

O Route é o componente mais importante, pois é responsável pela renderização da aplicação. Ele permite a passagem de props e aceita diferentes rotas – e não apenas uma.

Questão 2

Biblioteca utilizada para auxiliar o gerenciamento de estado, o Redux centraliza as informações, sendo responsável por repassá-las para o componente que precisa utilizar essas informações. Assinale a opção correta a respeito do Redux.

A

Biblioteca JavaScript de código aberto exclusiva para o React.

B

O Redux gera cópias de objetos, permitindo distribuir as informações de seus estados. Com isso, ele permite aos componentes acessar essas informações por meio das próprias declarações.

C

O Redux é nativo do React, não precisando de uma instalação extra.

D

Actions são basicamente as instruções enviadas aos reducers, informando qual função terá de ser executada e que valor essa função utilizará como parâmetro.

E

Reducers recebem os dados não tratados e precisam que as actions sinalizem os componentes da atualização dos estados.

Parabéns! A alternativa D está correta.

Apesar de ser a biblioteca oficial do Redux UI para o React, a forma com que ela foi desenvolvida nos permite utilizá-la em qualquer framework, inclusive em seus “concorrentes” diretos, como Angular e Vue, por exemplo. Todos os estados são centralizados no objeto denominado Store, não havendo a necessidade de gerar cópias. O Redux requer a instalação de pelo menos uma biblioteca, a React Redux. Outra que pode auxiliar nesse caso é a Redux Toolkit: com vários métodos já definidos, ela ajuda na criação de nossos componentes. Os reducers, além de receberem os dados já tratados, são responsáveis por avisar todos os componentes que precisarem atualizar seus estados.

Considerações finais

Pontuamos neste conteúdo que o React nos permite sanar alguns problemas que existiam nas aplicações web e mobile, bem como os desafios que seus criadores tiveram ao idealizar essa biblioteca. Falamos sobre o “ecossistema” usado para desenvolver aplicações em React, incluindo os conceitos de componentes, propriedades (props) e estado (state), para a preparação do ambiente de desenvolvimento.

Em seguida, aprendemos a realizar requisições HTTP em APIs web, consumindo dados de uma API de forma nativa e utilizando o Ajax. Demonstramos ainda como é possível tratar dos dados recebidos com Hooks, lidando com os possíveis efeitos colaterais que as requisições assíncronas possam trazer para a aplicação.

Por fim, descrevemos os conceitos relativos a roteamento, maneira que o React utiliza para a manipulação entre interfaces e componentes por meio de rotas. Para o gerenciamento da camada de negócio da melhor maneira, também apresentamos o React Redux.

Para encerrar, ouça sobre os principais conceitos relacionados com a criação de um App React.

Para ouvir o *áudio*, acesse a versão online deste conteúdo.



Explore +

Complemente seus estudos sobre o ReactJS acessando os seguintes sites:

GITHUB. **facebook /react**. A declarative, efficient, and flexible JavaScript library for building user interfaces. Consultado na internet em: 23 jun. 2022.

MICROSOFT. **Tutorial**: React no Windows para iniciantes. Publicado em: 15 abr. 2022.

REACT REDUX. **React Redux**: official react bindings for Redux. Consultado na internet em: 23 jun. 2022.

REACT ROUTER. **Documentation**. React Router. Consultado na internet em: 23 jun. 2022.

Em relação ao Hook, recomendamos a leitura desta documentação para entender melhor sobre:

a) A motivação

REACT. **Usando effect Hook (Hook de efeito)** - documentação. Consultado na internet em: 23 jun. 2022.

b) Algumas regras para utilização de Hooks

REACT. **Regras dos Hooks** - documentação. Consultado na internet em: 23 jun. 2022.

Referências

MAJ, W. W. **Mostrar ciclos de vida menos comuns**. Projects Wojtekamj. Consultado na internet em: 23 jun. 2022.

REACT. **React** - uma biblioteca JavaScript para criar interfaces. Consultado na internet em: 23 jun. 2022.

SILVA, M. S. **React** - aprenda praticando: desenvolva aplicações web reais com uso de biblioteca React e de seus módulos auxiliares. 1. ed. Rio de Janeiro: Novatec, 2021.

STOYAN, S. **Primeiros passos com React**: construindo aplicações web. 1. ed. São Paulo: Novatec, 2019.



Material para download

Clique no botão abaixo para fazer o download do conteúdo completo em formato PDF.



Download material

O que você achou do conteúdo?



Relatar problema