



React I

Bootcamp Desenvolvedor React

Raphael Gomide

2021

React I

Bootcamp Desenvolvedor

Raphael Gomide

© Copyright do Instituto de Gestão e Tecnologia da Informação.

Todos os direitos reservados.

Sumário

Capítulo 1. Introdução ao React	4
Considerações iniciais	4
React	4
Instalação e configuração	5
Características do React.....	6
Arquitetura do React	9
Capítulo 2. Function Components	10
Capítulo 3. React Hooks.....	11
O Hook useState.....	11
O hook useEffect	16
Capítulo 4. Conteúdo extra – Class Components.....	21
Capítulo 5. Glossário de termos importantes.....	23
Componente	23
JSX	23
Importação e exportação de arquivos.....	24
props.....	24
Closure	25
State	26
Hooks.....	26
Referências.....	27

Capítulo 1. Introdução ao React

Considerações iniciais

Prezado aluno, antes de abordarmos o assunto propriamente dito, peço para que utilize esta apostila como **referência** e não como o principal material do módulo.

Como o **Bootcamp** é bastante prático, o principal e mais importante conteúdo situa-se nas **videoaulas**. A orientação é que a apostila de módulos dos Bootcamps contenha **aproximadamente** 30 páginas.

Em resumo, o estudo somente da apostila não garante o aprendizado. Entretanto, o estudo e prática das videoaulas, mesmo que sem a apostila, pode sim garantir o aprendizado.

Outro detalhe importante são os exemplos desta apostila, que estão vinculados a projetos criados no **CodeSandBox** e não foram implementados nas videoaulas.

Portanto, pode ser considerado um bom material complementar à apostila e ao módulo como um todo.

React

O React foi criado por colaboradores do Facebook e se denomina uma biblioteca JavaScript para a construção de interfaces para o usuário. Foi inicialmente concebido para resolver um problema do Facebook de manter o estado das notificações que os usuários recebiam, que por muitas vezes não sincronizava corretamente.

A própria equipe de desenvolvimento do React o denomina como "*a JavaScript library for building user interfaces*".

- Site oficial: <https://reactjs.org/>.
- Repositório oficial no Github: <https://github.com/facebook/react>.

Instalação e configuração

O principal pré-requisito para a criação de apps com React é o Node.js, já que o React utiliza diversos de seus pacotes e necessita, por padrão, de diversas configurações para transpilação e empacotamento da aplicação. Assim, a configuração de um projeto React “do zero” não é nada trivial, em regra.

Para resolver esse problema, que poderia afastar entusiastas e principalmente novos desenvolvedores, foi criada uma ferramenta para simplificar o *scaffolding* de um novo projeto, denominada [create-react-app](#), também conhecida como CRA. Para garantir uma melhor compatibilidade entre os projetos do professor e dos alunos – evitando assim bugs desnecessários de incompatibilidade – será utilizado um projeto base, que contém o CRA + bibliotecas auxiliares. Mais detalhes sobre o projeto base podem ser vistos nas videoaulas.

De qualquer forma, para criar um projeto React com o create-react-app em sua última versão, basta executar o seguinte comando (considerando que o Node.js já está devidamente instalado e configurado):

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

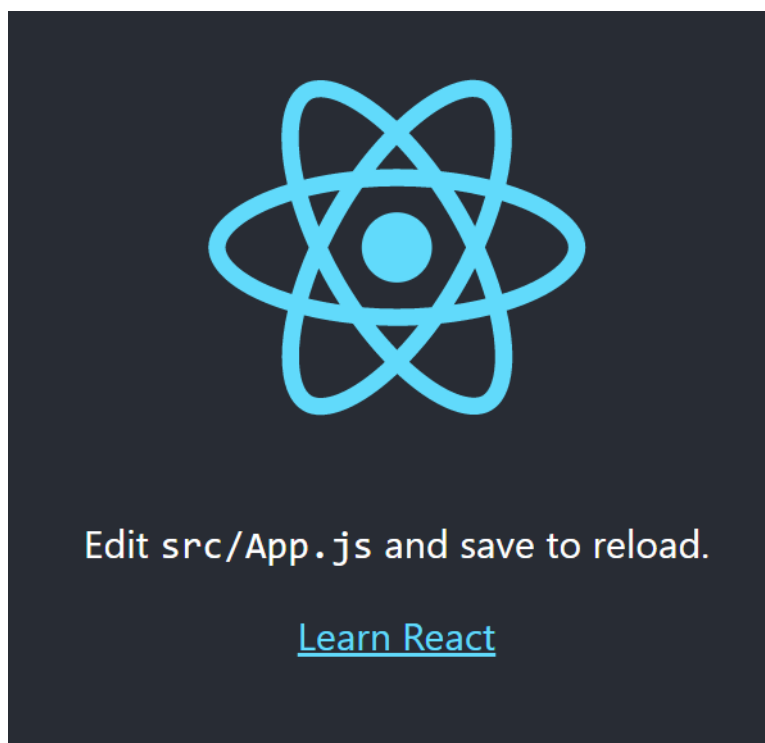
O comando acima faz o download do create-react-app, monta o *scaffolding* do projeto "my-app", faz o download de todas as dependências necessárias e, em seguida, descarta o pacote create-react-app (que de fato não é necessário para a continuidade do desenvolvimento em si, por isso a utilização do comando [npx](#)).

Para a manutenção de pacotes, o React utiliza por padrão o [yarn](#), que é uma ferramenta que funciona como uma alternativa ao comando **npm**, que é nativo do Node.js. Para mais detalhes dos comandos do yarn acesse este [cheat sheet](#). Um outro detalhe importante é que será utilizada a versão 1.x do Yarn, já que a versão atual (2.x) foi totalmente remodelada e ainda não foi totalmente "abraçada" pela comunidade. Portanto, pode-se afirmar que a versão 1.x é mais compatível, pelo menos por enquanto. As videoaulas iniciais demonstram como instalar o Yarn corretamente.

O servidor de desenvolvimento do React é executado, por padrão, na **porta 3000**. Para executar o seu projeto, acesse a pasta raiz do mesmo e escreva o seguinte comando em seu terminal de comandos: **yarn start** (CRA padrão) **ou yarn dev** (projeto base das videoaulas).

Isso faz com que o servidor de desenvolvimento do React seja executado e o navegador padrão do Sistema Operacional seja inicializado em uma nova aba apontando para o endereço <http://localhost:3000>, onde há um projeto padrão do React inicializado.

Aplicação inicial do create-react-app

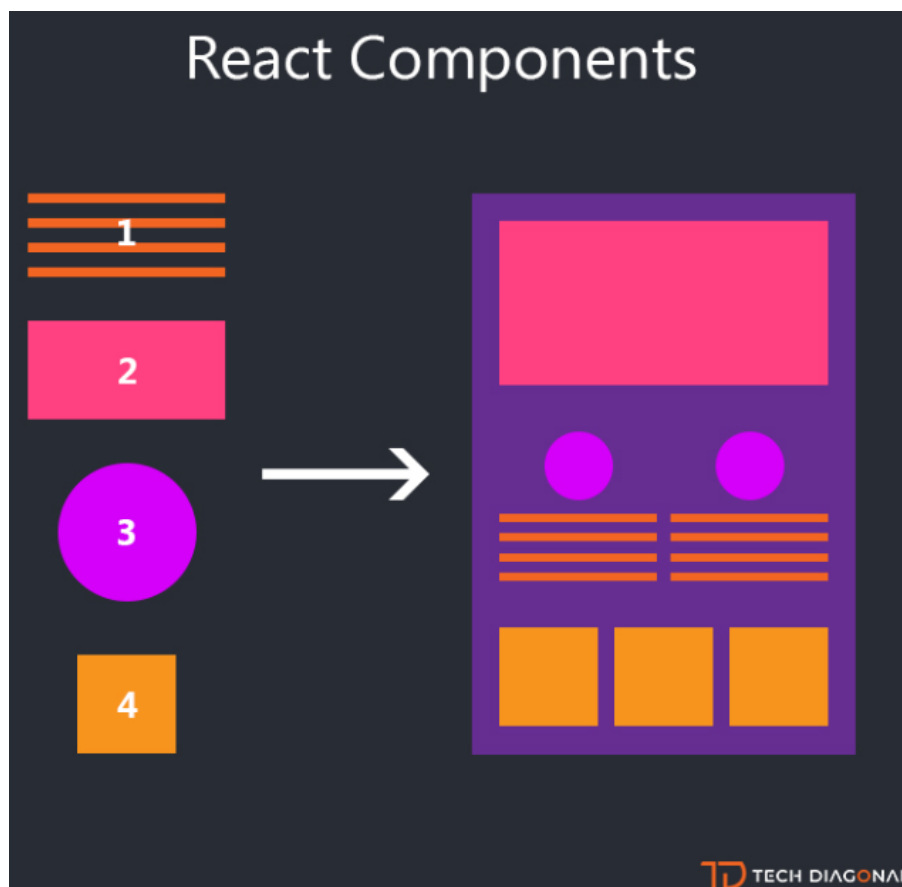


Características do React

O React se encontra atualmente na versão 17.x e foi totalmente reescrito internamente após a versão 15, sem afetar os projetos dos desenvolvedores.

Por ser baseado em componentes, o React permite muita reutilização de código. A figura abaixo ilustra este comportamento.

Reutilização de componentes no React.



Fonte: techdiagonal.com.

Perceba, na imagem acima, que os componentes **1** e **3** foram reutilizados duas vezes e que o componente **4** foi reutilizado três vezes.

A seguir são listadas algumas características importantes sobre o React.

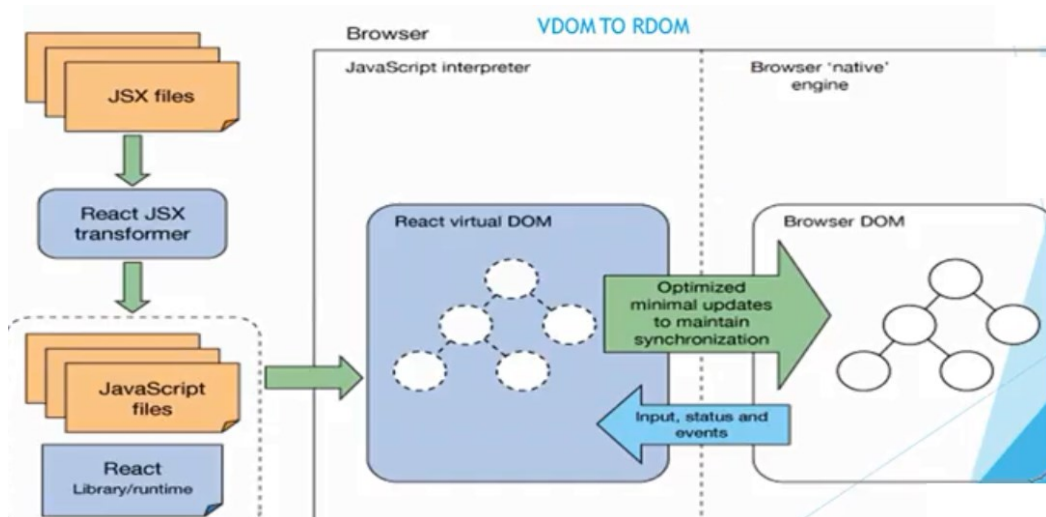
- **Componentizável:** o React também preza pela criação de componentes, assim como grande parte dos *frameworks* de JavaScript modernos. Componentes são blocos de código que, por serem altamente customizáveis, podem ser reutilizados em diversas partes de uma aplicação, onde cada instância do componente pode possuir o seu próprio estado.
- **Declarativo:** seguindo os princípios do desenvolvimento reativo, a criação de componentes é bastante declarativa (ao invés de imperativa), o que faz com que o React **reaja** a mudanças no estado da aplicação de forma **eficiente**.

- **Virtual DOM:** em aplicações web, a manipulação do DOM é de responsabilidade do Virtual DOM no React, que cria uma estrutura em memória do DOM e só efetua as atualizações realmente necessárias, o que garante melhor desempenho e, conseqüentemente, melhor experiência do usuário, já que o app tende a ser mais fluido e performático.
- **One-way data flow:** o React recomenda que concentremos o estado da aplicação no componente pai. Em regra, os filhos recebem dados do estado através de propriedades (**props**), que são, em regra, somente-leitura. Este processo faz com que a lógica de alteração do estado se concentre em somente um componente, o que leva a menos *bugs* na aplicação. Os componentes filhos podem alterar o estado do componente pai através de eventos que, uma vez disparados, invocam funcionalidades do componente pai, que então possuem permissão para alterar o estado da aplicação. Entretanto, nada impede que componentes filhos também possuam estado.
- **Learn Once, Write Anywhere:** o React dá suporte ao desenvolvimento em diversas outras plataformas, com destaque para o [React Native](#), que suporta por padrão *apps* nativos de dispositivos móveis com as plataformas Android e iOS. Entendendo bem o funcionamento do React para *web*, faz com que se aprenda a desenvolver para outras plataformas mais facilmente.
- **JavaScript moderno (ES6+):** com o React basta que o desenvolvedor aprenda JavaScript, os conceitos modernos do ES6+ e conheça a arquitetura do React. Isso permite mais flexibilidade e melhora a curva de aprendizado.
- **JSX (JavaScript XML):** o React suporta o **JSX** para facilitar a criação de componentes, tornando-os mais declarativos. A escrita com JSX torna o código de construção de componentes muito semelhante ao HTML. Entretanto, é necessária a utilização obrigatória do Babel para transpilar o código JSX, tornando a aplicação compatível aos navegadores. Isso é feito automaticamente pelo **create-react-app** e de forma transparente para o desenvolvedor.

Arquitetura do React

A figura abaixo ilustra a arquitetura de aplicações com React:

Arquitetura de um projeto React .



Fonte: biznomy.com.

Analisando a imagem da esquerda para a direita e de cima para baixo, é possível perceber o seguinte:

1. É feita uma transformação (transpilação) de componentes feitos em JSX para código JavaScript, que junto ao código do React propriamente dito são hospedados com a aplicação (em produção).
2. Durante a execução da aplicação, o React cria o VirtualDOM que monitora o DOM e só efetua a manipulação quando necessário e de forma eficiente.

Capítulo 2. Function Components

Atualmente, a equipe do React recomenda que os componentes sejam escritos como **Function Components**, ou seja, componentes baseados em **funções**. Existe também a abordagem de **Class Components**, que tende a ficar obsoleta e será vista em um outro capítulo.

Este tipo de componente é também considerado como “somente leitura”, pois os dados chegam através de propriedades (mais conhecidas como **props**), que são geralmente mantidas pelo componente pai, que pode então possuir **estado**. Assim, não há manipulação interna de estado em **Function Components** (a não ser que seja adotada a estratégia dos **React Hooks**, que será vista no próximo capítulo). As **props** são utilizadas também para a comunicação entre componentes não só com dados, mas também com funções.

Algumas características importantes dos **Function Components**:

- Podem possuir funções internas para abstrair melhor o código. Essas funções internas são mais conhecidas como **closures**, pois conseguem absorver o escopo externo do componente. Mais detalhes serão vistos nos tópicos sobre **React Hooks**;
- Manipulam *props*.
- Retornam apenas **um** elemento JSX.

Para entender melhor a estrutura de uma aplicação React com o create-react-app, consulte o material das videoaulas.

Exemplos de código de **function components** com explicações mais detalhadas serão vistos a partir do próximo capítulo.

Capítulo 3. React Hooks

A funcionalidade de **React Hooks** é aplicada aos **Function Components** e permite, por exemplo, utilização de **estado** e controle de **efeitos colaterais**.

“Por baixo dos panos”, hooks são simplesmente funções. Entretanto, essas funções (hooks) possuem funcionalidades importantes como a reatividade. Tudo isso é feito pelo próprio React e basta que o desenvolvedor entenda o *mindset* de funcionamento.

Existem diversos *hooks* que podem ser utilizados. Inclusive, o desenvolvedor pode criar os seus próprios *hooks*. Entretanto, para fins de simplicidade e para não fugir muito do escopo do módulo, serão apresentados somente os dois principais Hooks, que são **useState** e **useEffect**.

O Hook useState

O hook **useState** visa prover a manipulação de **estado** em uma aplicação. O termo **estado** pode ser definido como "**dados** que se **alteram** com o **tempo**, geralmente através da interação do usuário com **botões**, **inputs** etc."

A declaração de `useState` é geralmente feita com **array destructuring**, que é uma funcionalidade do ES6+. A sintaxe padrão de uma declaração de `useState` é a seguinte:

```
const [variable, setVariable] = useState(0);
```

Assim, o desenvolvedor define alguma variável que vai ser monitorada pelo React (ex: `name`), e escreve a linha acima – `const [name, setName] = useState('');`

O hook **useState** retorna por padrão um **array com dois elementos**. O primeiro elemento é o **valor inicial**, que é definido na **declaração** de **useState**. O segundo elemento é uma **função atualizadora**. Para simplificar a escrita, é muito comum a utilização de **array destructuring**, conforme o código acima. Além disso,

useState pode ser inicializado com um valor padrão, como nos exemplos acima (0 e '').

A seguir, serão vistos detalhes de implementação de uma aplicação React com **useState** através do estudo [deste app](#), que está disponível na plataforma CodeSandBox.

```
JS App.js x
1 import React, { useState } from "react";
2 import "./styles.css";
```

- A linha 1 mostra a **importação** de **React**, que é o módulo **principal (default)** da biblioteca **"react"** e **useState**, que é um dos módulos **opcionais** de **"react"**.
- A linha 2 mostra a importação do arquivo styles.css. Pela descrição da linha, pode-se inferir que o arquivo styles.css encontra-se na mesma pasta do arquivo App.js (que está sendo estudado).

```
4 const getTotalVowelsFrom = (text) => {
5   if (text.trim() === "") {
6     return 0;
7   }
8
9   return text
10    .toLowerCase()
11    .split("")
12    .filter(
13      (char) =>
14        char === "a" ||
15        char === "e" ||
16        char === "i" ||
17        char === "o" ||
18        char === "u"
19    ).length;
20 };
```

- As linhas 4 a 20 mostram uma implementação de uma função JavaScript.

- Perceba que não há nenhum vínculo com o React, ou seja, esta função poderia estar em um arquivo externo também e ser utilizada via **import**.
- Esta função implementa o cálculo da quantidade de vogais a partir de determinado texto.

```

22  const getTotalConsonantsFrom = (text) => {
23    if (text.trim() === "") {
24      return 0;
25    }
26
27    const vowels = getTotalVowelsFrom(text);
28
29    return (
30      text
31        .toLowerCase()
32        .split("")
33        .filter((char) => char !== " ").length - vowels
34    );
35  };

```

- As linhas 22 a 35 mostram também uma implementação de uma função JavaScript.
- Perceba que não há nenhum vínculo com o React, ou seja, esta função poderia estar em um arquivo externo também e ser utilizada via **import**.
- Perceba também que esta função reaproveita a implementação de **getTotalVowelsFrom**.
- Esta função implementa o cálculo da quantidade de consoantes a partir de determinado texto.

```

37  export default function App() {
38    const [name, setName] = useState("");
39
40    const handleNameChange = (event) => {
41      const newName = event.target.value;
42      setName(newName);
43    };
44
45    const vowels = getTotalVowelsFrom(name);
46    const consonants = getTotalConsonantsFrom(name);
47

```

- As linhas 37 a 46 mostram um trecho da implementação do **componente App**.
- Perceba que **App** é, no fim das contas, uma **função**. Um detalhe importante é ter começado com **letra maiúscula** (**A**pp), que é uma **convenção** e **recomendação** dos padrões do **React**.
- Na linha 38 há um exemplo de implementação com **useState**.
- Uma possível interpretação desta linha seria: “**React**, crie uma **variável** “**name**” e uma **função modificadora** “**setName**”, inicializando a **variável** com o valor “” (string vazia)”.
- Nas linhas 40 a 43 há uma implementação da função **handleNameChange** que, a partir do evento, extrai o valor digitado pelo usuário e invoca a função **setName**.
- Quando a função **setName** é executada, o **React** faz uma nova renderização do componente aplicando, claro, as técnicas de renderização performática (**Virtual DOM** e **Conciliation**). Este é o comportamento padrão.
- A função mostrada nas linhas 40 a 43 acima **não** é imediatamente **executada**. Essas linhas demonstram somente a **declaração** da função. A execução será mostrada em outra imagem.
- As linhas 45 e 46 demonstram o cálculo de vogais e consoantes a partir da variável de estado **name**. O cálculo é armazenado em nas variáveis **vowels** e **consonants**. Isso garante que essas variáveis estarão sempre atualizadas antes do componente ser renderizado.
- A renderização do componente será mostrada na próxima imagem.
- Perceba também que esta função reaproveita a implementação de **getTotalVowelsFrom**.
- Esta função implementa o cálculo da quantidade de consoantes a partir de determinado texto.

```

48     return (
49         <div>
50             <div>
51                 <h1>Exemplo com useState - v1.0.1</h1>
52             </div>
53             <p>
54                 <input
55                     placeholder="Digite o seu nome"
56                     type="text"
57                     value={name}
58                     onChange={handleNameChange}
59                 />
60             </p>
61             <p>
62                 O seu nome é {name} e possui {name.length}
63                 caracteres, sendo {vowels} vogais e {consonants} consoantes.
64             </p>
65         </div>
66     );
67 }
68

```

- As linhas 48 a 68 mostram, finalmente, a **renderização** do componente **App**.
- Para a renderização, é utilizada a sintaxe do JSX.
- Perceba que somente **um elemento é renderizado** (<div>). Esta div possui filhos (<div> + <h1>, <p> + <input> e um outro <p>) mas, no fim das contas, é considerado como **somente um elemento**. Esta é uma importante regra de renderização de componentes React.
- Nas linhas 54 a 59 há a declaração de um <input>, que pode ser utilizado para interação do usuário (digitação).
- Este <input> é **controlado pelo React**. É possível inferir isso devido ao fato das props **value** e **onChange** estarem “apontando” para **name** e **handleNameChange**, respectivamente.
- Perceba que, na linha 58, é feita a **referência** ao handleNameChange e não a execução da função (() => handleNameChange()). Esta função será executada somente **quando** o usuário digitar algo. A digitação automaticamente invoca o **evento onChange**.

- Com isso, temos um "ciclo fechado". O componente App renderiza inicialmente com o input vazio. **Quando** o usuário digita, **handleNameChange** é invocada, que faz a mudança de **estado** em **name** através de **setName**, que **invoca** uma nova renderização, que reflete os dados na tela. E assim sucessivamente.
- Nas linhas 61 a 64 há a renderização de um parágrafo que reflete os dados sempre atualizados de **name**, **vowels** e **consonants**. Perceba que utilizamos chaves (**{ }**) no **JSX** sempre que necessitarmos de uma **expressão JavaScript**.

O hook `useEffect`

O hook **`useEffect`** é bastante poderoso e pode ser utilizado para a sincronização de “efeitos colaterais” (*side effects*).

Segue alguns exemplos de efeitos colaterais que podem ocorrer em aplicações React:

- Manipulação de dados de Back End.
- Manipulação manual do DOM, como por exemplo a modificação de **`document.title`**.
- Inclusão/eliminação manual de Event Listeners.
- Inclusão/eliminação de intervals.
- Alteração de determinada variável de estado após a modificação de uma outra variável de estado.

O Hook **`useEffect`** tem um modelo mental onde a ideia principal é **sincronizar** o DOM conforme os valores de **`props`** e **`state`**. [Este link](#) possui um guia completo sobre o `useEffect`.

O hook **useEffect** permite utilizar um parâmetro extra, conhecido como array de dependências (**dependency array** ou, simplesmente, **deps**):

- Quando não há o parâmetro, **useEffect** é invocado **após qualquer atualização**. Esta é uma forma de utilização **desaconselhada**, pois pode provocar muitas renderizações desnecessárias dos componentes.
- Quando o parâmetro é [] (array vazio), **useEffect** é invocado apenas uma vez (após a primeira renderização do componente). Esta abordagem é mais comum para buscar dados do Back End e “alimentar” a página, por exemplo.
- Quando o parâmetro está preenchido com [state1, state2, prop1, prop2 etc], **useEffect** é invocado após a atualização de estado de **qualquer uma** dessas variáveis. É a utilização mais comum.
- Quando há retorno na função – **useEffect** utiliza esta função de retorno para eliminar recursos. Esta função é conhecida como *cleanup function*. É menos comum, mas é muito importante para lidar com eventos do DOM, eventos do navegador e intervals, por exemplo.

A seguir, serão vistos detalhes de implementação de uma aplicação React com **useEffect** através do estudo [deste app](#), que está disponível na plataforma CodeSandBox.

```
JS App.js  x  # styles.css
1  import React, { useEffect, useState } from "react";
2  import "./styles.css";
```

```

JS App.js # styles.css x
1  body {
2    font-family: sans-serif;
3  }
4
5  #keyboardNavigation {
6    position: relative;
7  }
8
9  #spanText {
10   position: absolute;
11   top: 0px;
12   left: 20px;
13 }

```

- As linhas 1 e 2 de App.js fazem as importações de dependências do React e do arquivo styles.css.
- As linhas 5 a 13 de styles.css definem uma estilização CSS no padrão relativo/absoluto para ser possível movimentar elementos – que é o propósito deste app. Mais informações sobre este padrão de posicionamento de elementos podem ser vistas [aqui](#).
- As próximas imagens referem-se somente ao arquivo App.js

```

4  export default function App() {
5    const [position, setPosition] = useState("0");

```

- As linhas 4 e 5 de App.js mostram a declaração do componente e a inicialização do estado, com a variável **position**.
- Esta variável será utilizada para definir o posicionamento de um parágrafo na renderização, que será visto a seguir.

```

7    useEffect(() => {
8      const handleKeyUp = (event) => {
9        const key = event.key;
10
11        if (key === "ArrowLeft") {
12          setPosition((+position - 10).toString());
13        }
14
15        if (key === "ArrowRight") {
16          setPosition((+position + 10).toString());
17        }
18      };
19
20      window.addEventListener("keyup", handleKeyUp);
21
22      return () => {
23        window.removeEventListener("keyup", handleKeyUp);
24      };
25    }, [position]);

```

- As linhas 7 e 25 de App.js mostram a declaração do `useEffect`.
- Na linha 25 informamos que este efeito será executado sempre que houver alguma mudança em **position**. Em outras palavras, podemos dizer que a função de **useEffect sincroniza os dados** conforme o valor de **position**.
- A implementação deste `useEffect` possui uma função **handleKeyUp** para lidar com a digitação por parte do usuário. Esta função filtra somente as teclas “seta esquerda” (**ArrowLeft**) e “seta direita” (**ArrowRight**), ignorando todas as outras teclas.
- Nas linhas 20 a 24, temos a atribuição (linha 20) e a remoção (linhas 22 a 24) do evento para monitorar o evento **keyup**.
- Perceba que a **remoção** do evento é implementada como um retorno (**return**) de uma **arrow function** e só é de fato executada quando o componente é **removido** do **DOM**.
- Para mais exemplos com **useEffect**, verifique as videoaulas.

```
27     return (  
28       <div>  
29         <div className="App">  
30           <h1>Exemplos com useEffect</h1>  
31         </div>  
32  
33         <div id="keyboardNavigation">  
34           <span id="spanText" style={{ left: position + "px" }}>  
35             Clique uma vez aqui e navegue com as setas esquerda e direita do  
36             teclado.  
37           </span>  
38         </div>  
39       </div>  
40     );  
41   }  
42 }
```

- Nas linhas 27 a 42, temos a renderização do componente, utilizando as definições de styles.css e o estado com **position**.
- Para mais exemplos com **useEffect**, verifique as videoaulas.

Capítulo 4. Conteúdo extra – Class Components

Observação importante: as Class Components ainda são suportadas pela versão 17.x do React, mas tendem a se tornar obsoletas. O React atualmente dá preferência às **function components** com **Hooks**, que foram abordadas nos capítulos anteriores. Portanto, as **Class Components** não serão abordadas nas videoaulas. De qualquer forma, deixarei este material aqui como referência.

A abordagem de **Class Components** foi a primeira adotada pelo React e continua funcionando atualmente.

Com a criação dos **React Hooks** em 2018, a utilização de **Class Components** passou a ser considerada **verbosa e pouco declarativa** e, por isso, tende a ser descartada com o tempo, mesmo que a própria equipe do React tenha afirmado que vai manter o suporte a ambas as abordagens, pelo menos a curto prazo.

Class Components baseiam-se em **classes** do JavaScript e herdam obrigatoriamente do objeto **Component** do React, possuindo, essencialmente:

- **Construtor de classe:** aqui é geralmente definido o estado do componente com **this.state**, que pode ou não ser baseado em **props** (mais detalhes sobre **props** ainda serão vistos na apostila). A invocação de **super()**, que invoca instruções importantes de Component, é **obrigatória**.
- **Método render():** principal método de um Class Component, que realiza a renderização do componente em tela. Normalmente utiliza-se muito JSX, Object Destructuring e Array.map para se interpolar os dados.
- **Lifecycle methods:** são métodos opcionais de Class Components que capturam determinado evento no ciclo de vida do componente. Para mais detalhes sobre os lifecycle methods, acesse [este link](#). Os principais lifecycle methods são:
 - **componentDidMount:** ocorre **após** o componente ser renderizado pela primeira vez, ou seja, acontece somente uma vez durante o ciclo de vida

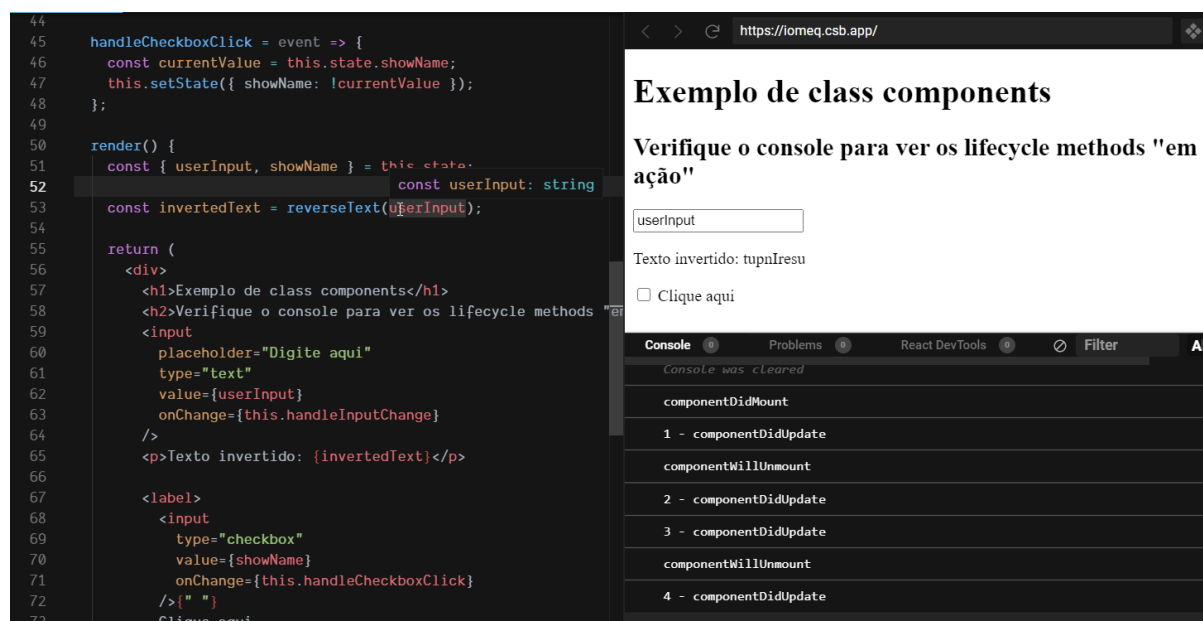
do componente. É um bom local para requisições HTTP únicas, por exemplo.

- **componentDidUpdate**: ocorre **após** toda e qualquer renderização subsequente do componente, ou seja, pode ocorrer diversas vezes durante o ciclo de vida do componente. É um bom local para a implementação de **efeitos colaterais** (manipulação manual do DOM, cálculos com base no objeto state, utilização de biblioteca de terceiros etc.).
- **componentWillUnmount**: ocorre **antes** do componente ser removido do DOM. Ocorre somente uma vez durante o ciclo de vida do componente. É o local adequado para remover eventListeners, utilizar clearInterval etc.

É muito importante assistir às videoaulas para praticar a criação e manutenção de class componentes, que é feita nos vídeos através de diversos exemplos práticos.

O seguinte [link](#) apresenta mais um exemplo prático de utilização de **class components** com **lifecycle methods**.

Projeto com class components e lifecycle methods.



The screenshot displays a web application running in a browser at <https://iomeq.csb.app/>. The application title is "Exemplo de class components". It features a text input field with the placeholder "Digite aqui", a checkbox labeled "Clique aqui", and a paragraph showing the inverted text of the input. The browser's console window is open, showing a list of lifecycle methods: componentDidMount, 1 - componentDidUpdate, componentWillUnmount, 2 - componentDidUpdate, 3 - componentDidUpdate, componentWillUnmount, and 4 - componentDidUpdate. The source code in the background shows the implementation of these methods and the UI components.

Fonte: codesandbox.io.

Capítulo 5. Glossário de termos importantes

Componente

Um componente React pode ser considerada uma **função** que encapsula determinado comportamento (estado + funções).

Em regra, realiza processamento e retorna dados renderizados (HTML + CSS). Se bem escritos, os componentes podem ser muito bem reaproveitados, promovendo o reuso e tornando o código mais consistente.

Uma aplicação React é geralmente composta por diversos componentes. Os componentes não precisam necessariamente possuir estado, já que os dados podem ser modificados através de **props** que, em algum momento, são o estado de algum outro componente de nível superior.

JSX

JSX (JavaScript XML) é uma linguagem de marcação muito semelhante ao HTML, porém em JavaScript.

JSX não é interpretado nativamente pelos navegadores e é muito utilizada pelo **React** para a escrita da renderização de **componentes**.

As principais diferenças entre o HTML tradicional e o JSX são:

- As classes CSS são escritas como **className** ao invés de **class**.
- O atributo **for** de `<label>` é escrito como **htmlFor**.
- As tags que não possuem conteúdo devem ser obrigatoriamente fechadas com `</>`, como por exemplo `<input type="text" />`
- É possível inserir expressões JavaScript através de chaves `{ }`.

- Quando a expressão JavaScript é um objeto, as chaves acabam sendo duplas. O par exterior refere-se à sintaxe do JSX e o inferior à delimitação do objeto, que é realmente feita com as chaves. Exemplo: `<p style={{ fontSize: '2rem' }}>Teste</p>`

Importação e exportação de arquivos

O React utiliza muito do JavaScript moderno (ES6+). Para a utilização de arquivos externos são utilizadas as palavras-chave **import** e **export**.

Em regra, utilizamos **import** para incluir funções, dados, componentes presentes em outros arquivos do projeto e/ou bibliotecas.

Em regra, utilizamos **export** para que o arquivo em questão possa ser utilizado por outros componentes no projeto.

props

As **props** são as propriedades de componentes. Devem ser consideradas como dados “somente leitura”, ou seja, não é recomendável alterar seus valores.

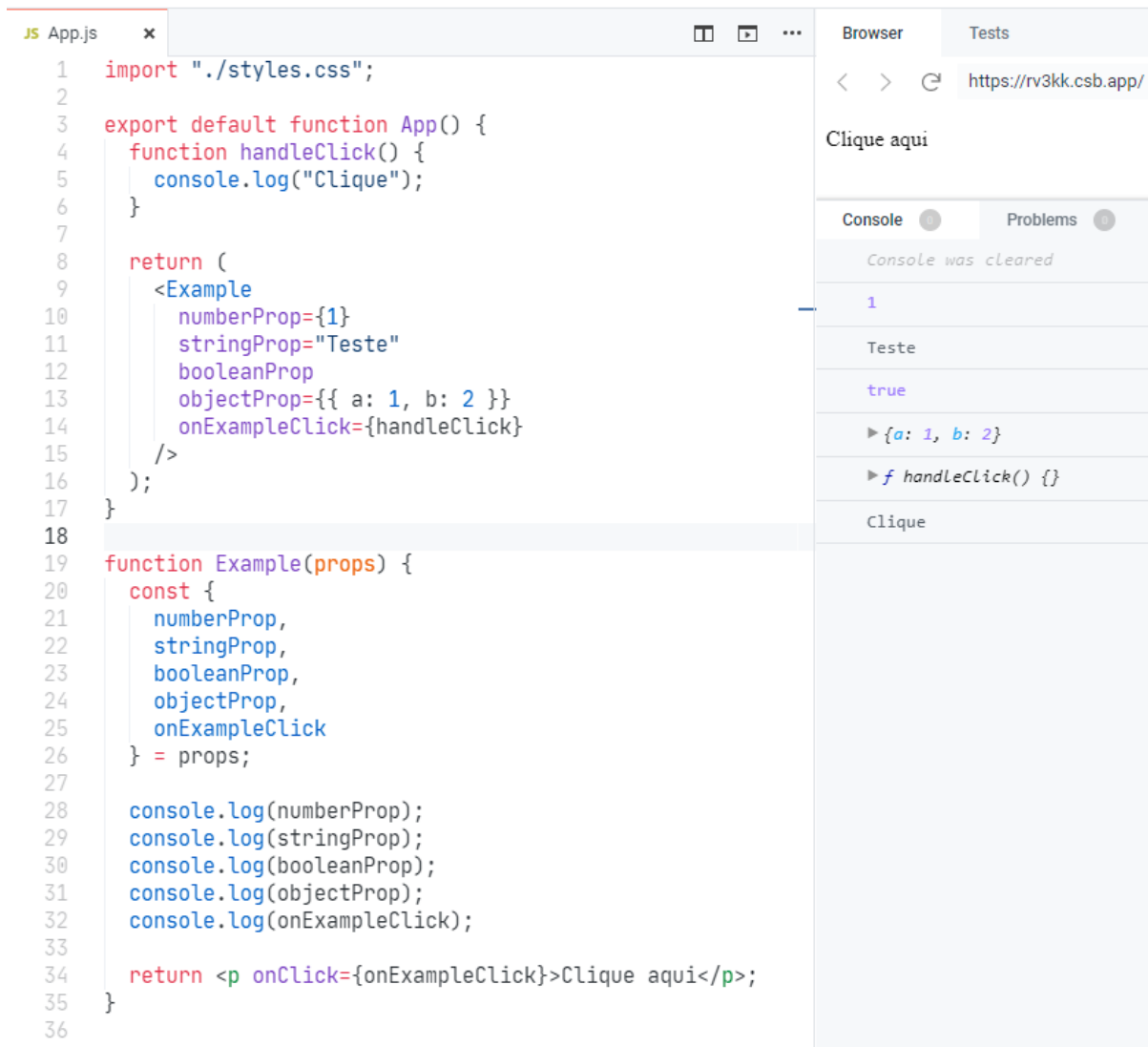
Uma **prop** geralmente refere-se a um **state** em um componente de nível superior. Uma boa analogia são os atributos de tags HTML.

Existe uma **prop** especial – **children**, que representa todos os elementos definidos como conteúdo de componentes.

As **props** podem conter dados de todos os tipos e outros componentes. A principal função das **props** é a **comunicação** entre **componentes**.

[Aqui](#) há um projeto com diversos exemplos de utilização de props.

Exemplos com utilização de props.



```

1  import "./styles.css";
2
3  export default function App() {
4    function handleClick() {
5      console.log("Clique");
6    }
7
8    return (
9      <Example
10        numberProp={1}
11        stringProp="Teste"
12        booleanProp
13        objectProp={{ a: 1, b: 2 }}
14        onExampleClick={handleClick}
15      />
16    );
17  }
18
19  function Example(props) {
20    const {
21      numberProp,
22      stringProp,
23      booleanProp,
24      objectProp,
25      onExampleClick
26    } = props;
27
28    console.log(numberProp);
29    console.log(stringProp);
30    console.log(booleanProp);
31    console.log(objectProp);
32    console.log(onExampleClick);
33
34    return <p onClick={onExampleClick}>Clique aqui</p>;
35  }
36

```

Browser: https://rv3kk.csb.app/

Clique aqui

Console: Console was cleared

1

Teste

true

{a: 1, b: 2}

f handleClick() {}

Clique

Closure

Em regra, uma **closure** é uma **função** implementada dentro do **escopo** de **outra função**. A vantagem desta implementação é que a **closure** tem acesso aos dados do escopo externo.

Closures são muito utilizadas pelo React na criação de componentes, através da criação de funções para lidar com ocorrência de eventos (cliques, digitação etc.).

State

State representa o **estado** de componentes. Uma boa definição para o termo **estado** é: **dado** que pode ser **modificado** com o **tempo**.

Atualmente, a principal e mais simples ferramenta para a manipulação de **estado** em aplicações **React** é a utilização do hook **useState**.

Alguns bons exemplos de **estado** em componentes: dados de formulários, dados vindos do Back End etc.

Hooks

Um **hook** é uma estrutura do React que pode ser vinculada (hooked) aos componentes. Internamente, os **hooks** são implementadas como **funções** e atuam diretamente na **reatividade** do React, ou seja, a manipulação de dados de hooks acarretam na renderização de componentes. É possível criar nossos próprios hooks.

Uma convenção importante na criação e utilização de hooks é que devem começar com o prefixo “**use**”. Os principais do ecossistema do React são: **useState** e **useEffect**.

Referências

FACEBOOK OPEN SOURCE. *Getting Started*. Disponível em
<<https://reactjs.org/docs/getting-started.html>>. Acesso em: 28 mai. 2021.

MDN – MOZILLA DEVELOPER NETWORK WEB DOCS. *Home*. Disponível em:
<<https://developer.mozilla.org/en-US/>>. Acesso em: 28 mai. 2021.