

ReactJS

Público alvo: Desenvolvedores que queiram aprender a utilizar a biblioteca ReactJS.

Pré-requisitos: HTML, CSS e JavaScript.

Índice

Copyright	4
Equipe	5
Histórico de edições	5
Capítulo 01 - O ReactJS	6
Arquitetura do ReactJS	7
ReactJS e React Native	7
ES6	8
Exercícios	9
Capítulo 02 - Configurando o ambiente	10
Navegadores	10
Node.js	11
Visual Studio Code	12
Instalando o React	13
Capítulo 03 - Criando primeira aplicação	14
Iniciando projeto	14
Estrutura de um projeto ReactJS	16
Executando projeto	16
Live Reload	18
Capítulo 04 - Render	20
Arquivo HTML	21
Arquivo JavaScript	22
Exibindo estruturas	23
Exercícios	24
Capítulo 05 - JSX	25
Expressões com JSX	26
Fechamento de tags obrigatório	26
Agrupando elementos	27
Capítulo 06 - Componentes	28
Criando primeiro componente	29
Unindo componentes	30
Capítulo 07 - Props	31
Exemplo prático	32
Construtor e super	33
Exercícios	35

Capítulo 08 - State	36
Exemplo prático	37
Props e State	39
Exercícios	40
Capítulo 09 - Eventos	41
Evento Click	42
Evento Change	43
Evento Submit	44
Exercícios	45
Capítulo 10 - Formulário	46
Exemplo prático	46
Exercícios	49
Capítulo 11 - CSS	50
CSS inline	50
CSS em JavaScript	51
Anexando folhas de estilo	52

Copyright

As informações contidas neste material se referem ao curso de Lógica de Programação e estão sujeitas às alterações sem comunicação prévia, não representando um compromisso por parte do autor em atualização automática de futuras versões.

A Apex não será responsável por quaisquer erros ou por danos acidentais ou consequenciais relacionados com o fornecimento, desempenho, ou uso desta apostila ou os exemplos contidos aqui.

Os exemplos de empresas, organizações, produtos, nomes de domínio, endereços de email, logotipos, pessoas, lugares e eventos aqui apresentados são fictícios. Nenhuma associação a empresas, organizações, produtos, nomes de domínio, endereços de email, logotipos, pessoas, lugares ou eventos reais é intencional ou deve ser inferida.

A reprodução, adaptação, ou tradução deste manual mesmo que parcial, para qualquer finalidade é proibida sem autorização prévia por escrito da Apex, exceto as permitidas sob as leis de direito autoral.

Equipe

Conteúdos	Diagramação	Revisão
Ralf S. de Lima	Fernanda Pereira	Fernanda Pereira

Histórico de edições

Edição	Idioma	Edição
1 ^a	Português	Julho de 2020

Capítulo 01 - O ReactJS

O ReactJS é uma biblioteca criada e mantida pelo Facebook para a criação de componentes front-end. O ReactJS têm uma comunidade muito ativa, pois a tecnologia é gratuita e os desenvolvedores podem auxiliar o Facebook com melhorias.

Podemos desenvolver utilizando as linguagens JavaScript e TypeScript, neste curso teremos como foco utilizar o JavaScript, pois facilitará o aprendizado, além de muitas empresas utilizarem a linguagem JavaScript pura para o desenvolvimento de interfaces componentizadas web.

Empresas de pequeno, médio e grande porte podem utilizar ReactJS para elaborar seu site ou sistema interno, abaixo veja algumas das empresas de renome que utilizam ReactJS:



Facebook



Netflix



Instagram



Airbnb



Walmart



Uber

Arquitetura do ReactJS

O ReactJS é baseado em uma arquitetura de componentes, onde cada parte do site é um pequeno componente, e esse componente pode se comunicar com outros componentes e até ser reutilizado em alguns momentos.

A ideia do ReactJS é criar uma estrutura pequena para cada parte do site, fornecendo um código limpo, bem estruturado, organizado e reutilizável.

Para quem já desenvolve em alguma linguagem, sendo para back-end ou front-end, geralmente tem que manipular arquivos com códigos muito extensos, já o ReactJS que trabalha com a arquitetura de componentes que se separam em pequenas classes, facilitando a compreensão da estrutura dos projetos.

ReactJS e React Native

O React possui duas versões, a web que é conhecida por ReactJS criada em 2013 e uma versão híbrida para desenvolvimento mobile que é a React Native criada em 2015. A grande vantagem de desenvolver utilizando a biblioteca React é que ambas são muito parecidas, sendo assim o desenvolvedor poderá atuar tanto na criação de aplicativos móveis quanto para sites.

Dispositivos híbridos desenvolvidos pelo React Native são muito estáveis, possuem fácil compreensão para desenvolvedores front-end, além de poder exportar a aplicação para os principais sistemas operacionais mobile, assim o desenvolvedor não precisa aprender várias linguagens, frameworks e utilizar IDE's ou sistemas operacionais diferentes.

Há uma gama interessante de complementos que podemos utilizar com o ReactJS e o React Native, facilitando o desenvolvimento dos sistemas. Realmente o React tem um mercado amplo, compreendendo os conceitos básicos podemos desenvolver muitas aplicações interessantes para o mercado.

ES6

O ReactJS tem suporte ao ES6, ECMAScript 6 ou ECMAScript 2015, que é um JavaScript com diversas funcionalidades extras que facilitam o desenvolvedor a criar funções e manipular dados, as principais características do ECMAScript são:

- Suporte a classes
- Arrow functions
- Variáveis e constantes (let, var e const)
- Filter
- Map

O ES6 foi criado em 2015 e é uma atualização do JavaScript convencional, esse JavaScript é o mesmo suportado pelos navegadores e não é uma versão exclusiva para o ReactJS. Podemos desenvolver sem auxílio de nenhuma framework ou biblioteca script em ES6.

Exercícios

- a) O que é o ReactJS?
- b) Qual a diferença entre ReactJS e React Native?
- c) Neste capítulo vimos que algumas empresas como Netflix, Airbnb, Uber, Instagram, Netflix e Walmart utilizam o React em seus projetos, pesquise por mais empresas que adotam o uso do React.
- d) Explique sobre a arquitetura do React.
- e) Em quais linguagens podemos desenvolver utilizando ReactJS?
- f) Pesquise sobre as diferenças entre um framework e uma biblioteca e informe o que é o React, um framework ou uma biblioteca?
- g) O que são dispositivos híbridos? Como podemos criar utilizando o React?

Capítulo 02 - Configurando o ambiente

Neste capítulo será abordado sobre nosso ambiente e desenvolvimento, instalando e configurando os softwares necessários.

A configuração do nosso ambiente pode seguir o mesmo padrão para os sistemas operacionais Windows, Linux e macOS.

Navegadores

Inicialmente vamos precisar ter um navegador, você pode optar por qualquer um de sua preferência, em nossos exemplos será utilizado o Google Chrome.

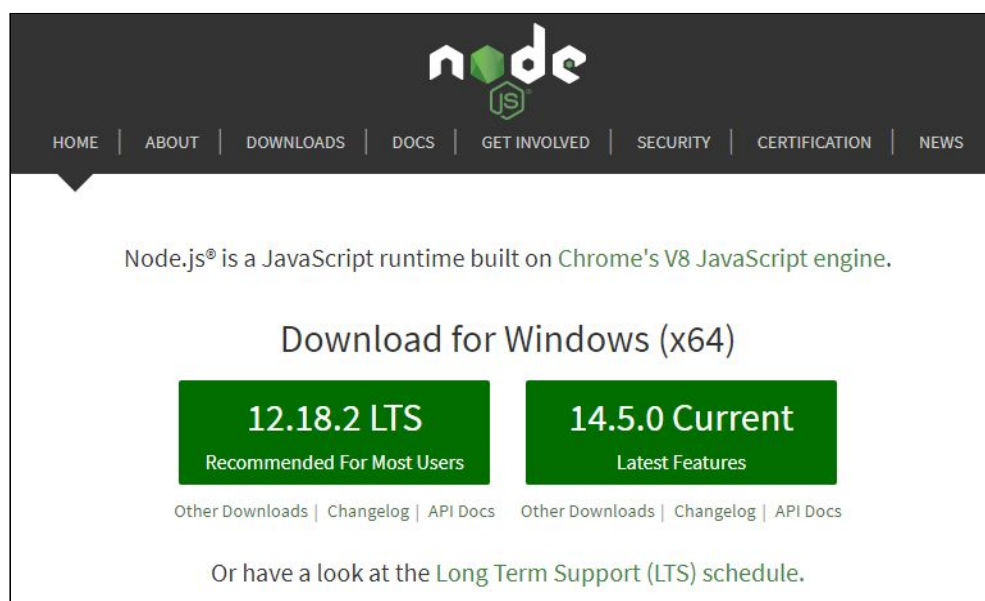


Node.js

O Node.js terá a finalidade de realizar algumas funcionalidades do nosso projeto em ReactJS como por exemplo gerenciar os pacotes complementares como arquivos JavaScript e CSS, além de conseguirmos realizar testes em tempo real e outras funcionalidades incríveis que veremos no decorrer do curso.

Vale lembrar que o Node.js é totalmente gratuito e disponível nos principais sistemas operacionais. Ele será útil se você deseja utilizar outras ferramentas como o Angular por exemplo.

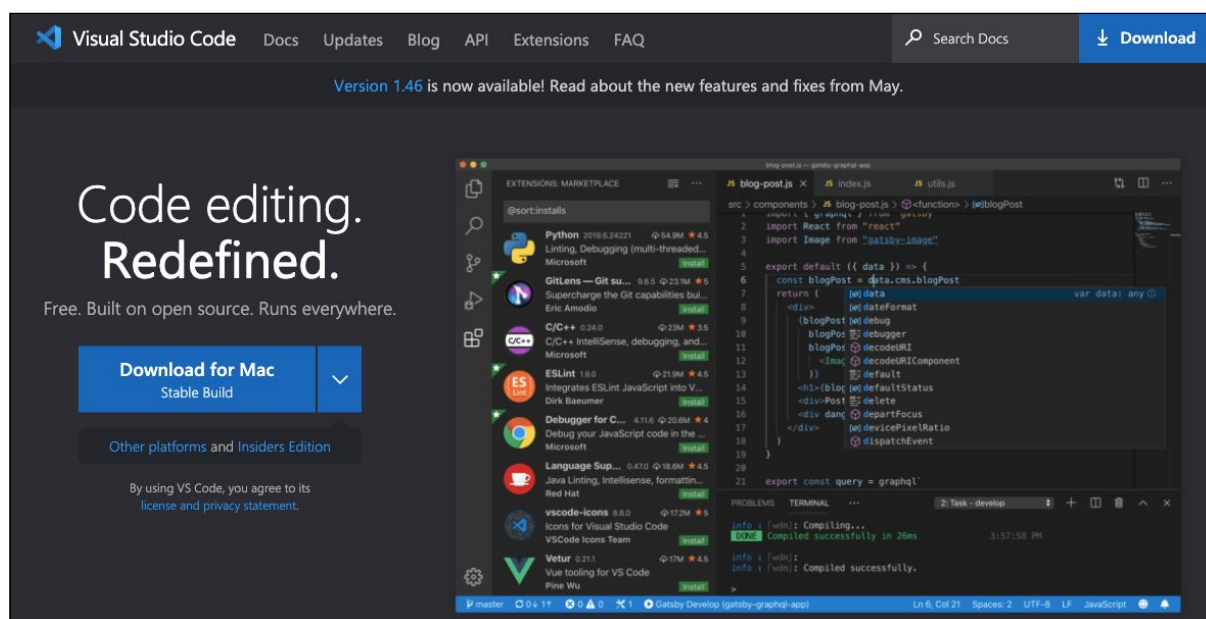
Quando estiver com o Node.js instalado você terá acesso ao NPM (Node Package Modules), com ele podemos gerenciar bibliotecas, frameworks e complementos de maneira bem simples.



Visual Studio Code

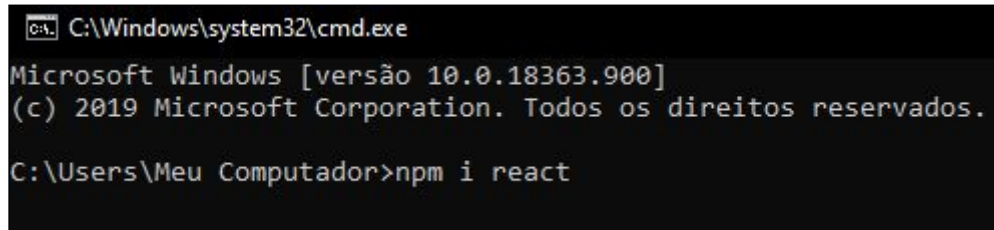
Para desenvolvermos nossos projetos básicos de lógica e algoritmos podemos adotar diversos editores. Tendo como base o mercado atual, será utilizado o Visual Studio Code da Microsoft, uma versão leve, gratuita e disponível nos principais sistemas operacionais.

Este editor possui uma série de complementos interessantes para desenvolvermos em várias linguagens como: Java, C#, PHP, Node.js, Python, JavaScript, TypeScript, entre outras.



Instalando o React

Assim que instalar o Node.js podemos baixar o ReactJS através do prompt de comando (Windows) ou terminal (MacOS e Linux), para isso vamos seguir o comando que está na imagem abaixo:



```
C:\Windows\system32\cmd.exe
Microsoft Windows [versão 10.0.18363.900]
(c) 2019 Microsoft Corporation. Todos os direitos reservados.
C:\Users\Meu Computador>npm i react
```

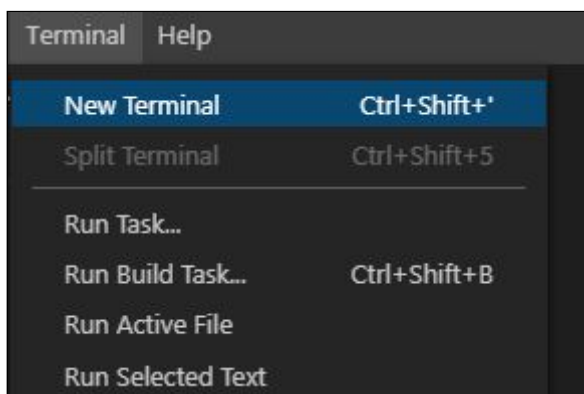
Tenha paciência, pois é um processo demorado. Depois de instalado teremos toda a estrutura necessária para desenvolvermos nossas aplicações com o ReactJS.

Capítulo 03 - Criando primeira aplicação

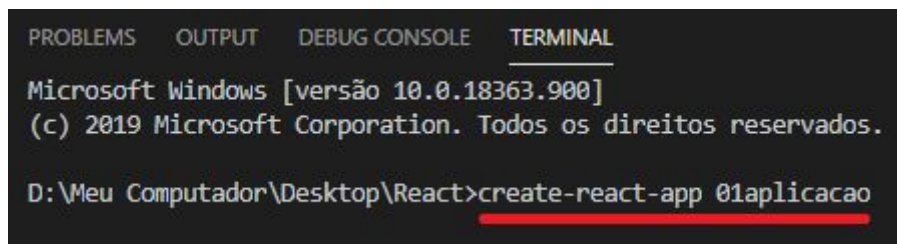
Nessa etapa vamos criar nossa primeira aplicação em ReactJS, sendo assim crie uma pasta para salvar nossos exemplos, em seguida abra o Visual Studio Code e selecione a pasta criada.

Iniciando projeto

Com o Visual Studio Code aberto, selecione a opção **New Terminal**:



Quando selecionar a opção **New Terminal**, abrirá uma opção na parte inferior do Visual Studio Code para podermos trabalhar com comandos do Node.js, digite o comando para criar uma nova aplicação React: **npx create-react-app 01aplicacao** e pressione a tecla Enter, agora só esperar alguns minutos.



Ao término do processo de criação do projeto, teremos as seguintes mensagens no terminal:

```
Success! Created 01aplicacao at D:\Meu Computador\Desktop\React\01aplicacao
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.

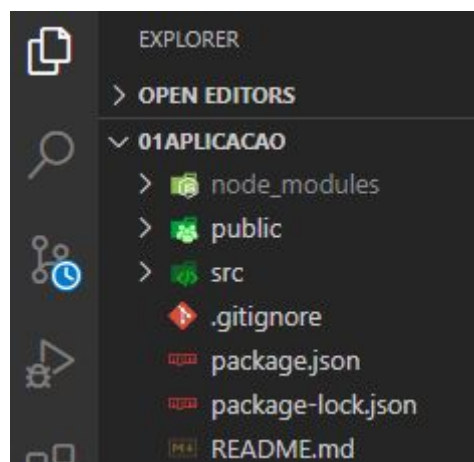
npm run eject
  Removes this tool and copies build dependencies, configuration files
  and scripts into the app directory. If you do this, you can't go back!

We suggest that you begin by typing:

cd 01aplicacao
npm start

Happy hacking!
```

Finalizando o processo de criação do projeto, selecione o diretório **01aplicacao** através do comando: **File -> Open Folder**, assim poderemos trabalhar com os arquivos criados e também executar o projeto. Assim que selecionar o projeto, verifique na lateral esquerda do Visual Studio Code a estrutura de arquivos:



Estrutura de um projeto ReactJS

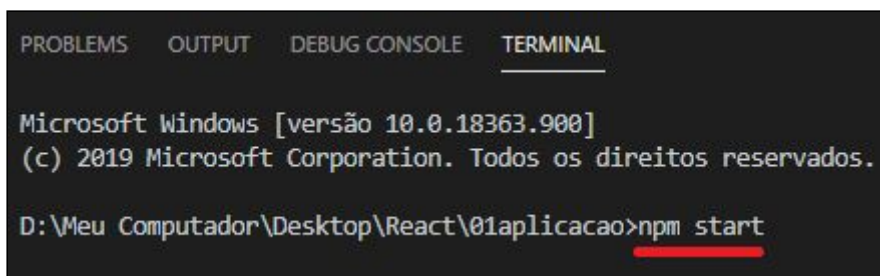
A estrutura de arquivos dispostas em um novo projeto ReactJS são divididas da seguinte maneira:

- **node_modules:** Local onde serão baixados os arquivos via Node.
- **public:** Arquivos públicos como imagens.
- **src:** Diretório onde está o projeto (HTML, CSS e JS).
- **package-json:** Informações do projeto, versão do Node.js, React, entre outros dados para manter o projeto padronizado para todos os envolvidos.

Há outros arquivos como o **.gitignore** que trabalha para o versionamento de código para quem usa Github ou outras plataformas do gênero, além de um arquivo **package-lock.json**, com informações extras sobre as dependências do projeto.

Executando projeto

Para executar o projeto, precisamos estar com o terminal do Visual Studio Code aberto e digitar o comando **npm start**, esse processo pode levar alguns minutos:



```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL

Microsoft Windows [versão 10.0.18363.900]
(c) 2019 Microsoft Corporation. Todos os direitos reservados.

D:\Meu Computador\Desktop\React\01aplicacao>npm start
```


Assim que finalizar o processo, o navegador será executado no endereço: localhost:3000, que é o endereço padrão das aplicações ReactJS, note também que teremos uma página de boas vindas:



Conseguindo chegar até aqui, parabéns! Nosso projeto ReactJS está 100% em funcionamento, agora podemos aprender a trabalhar com suas funcionalidades básicas e deixá-lo(a) pronto para o mercado front-end.

Live Reload

Ainda mexendo no nosso projeto **01aplicacao**, vamos compreender um dos motivos pelo qual os desenvolvedores gostam de utilizar o ReactJS. Há uma função chamada live reload, essa função permite que a aplicação seja atualizada no navegador de maneira automática sempre que o desenvolvedor salvar o projeto.

Para exemplificar vamos alterar um arquivo e testar o **live reload**, procure o arquivo **App.js** e altere o texto entre as tags `<p></p>`:

```
5  function App() {
6      return (
7          <div className="App">
8              <header className="App-header">
9                  <img src={logo} className="App-logo" alt="logo" />
10                 <p>Aprendendo ReactJS na Apex!</p>
11                 <a
12                     className="App-link"
13                     href="https://reactjs.org"
14                     target="_blank"
15                     rel="noopener noreferrer"
16                 >
17                     Learn React
18                 </a>
19             </header>
20         </div>
21     );
22 }
```

Alteração realizada, salve o projeto e abra o navegador, certifique-se que esteja com o endereço localhost:3000, note que o resultado será exibido sem precisar se preocupar em atualizar o projeto no navegador:

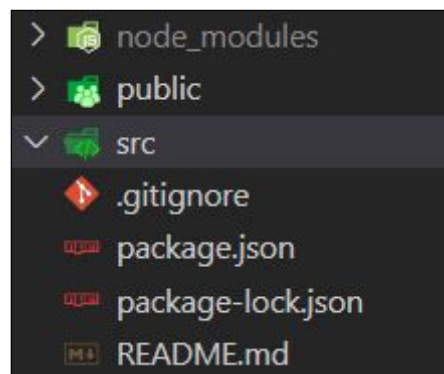


Nos próximos capítulos iremos criar nossas próprias páginas, porém você pode ver como é fácil iniciar seus projetos em ReactJS.

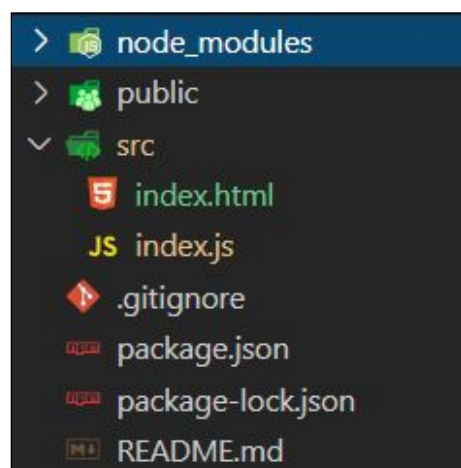
Capítulo 04 - Render

O comando `render` no React, tem como funcionalidade retornar uma estrutura, podendo ser desde um botão até uma página completa com formulários, tabelas, imagens, textos, vídeos e demais elementos.

Será criado um novo projeto em ReactJS, vamos seguir exatamente os mesmos passos do capítulo anterior, fique à vontade para escolher o nome deste novo projeto. Assim que criado, vamos selecionar o projeto e remover todos os arquivos dentro da pasta **src**:



Vamos criar dois arquivos no diretório **src**, um na extensão HTML e outro JavaScript:



Arquivo HTML

O arquivo HTML será responsável por exibir o render gerado no arquivo JavaScript. Teremos uma estrutura base e uma div com um identificador, no seu arquivo HTML monte a seguinte estrutura:

```
index.html X
src > index.html > ...
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <meta charset="UTF-8">
5      <meta name="viewport" content="width=device-width, initial-scale=1.0">
6      <title>ReactJS na Apex</title>
7  </head>
8  <body>
9
10     <div id="root"></div>
11
12 </body>
13 </html>
```

Na linha 10 foi criada uma div com o identificador **root**, você pode estar criando outros nomes, porém **root** é algo muito usual nos principais tutoriais sobre a biblioteca.

Arquivo JavaScript

O arquivo JavaScript será responsável por enviar uma estrutura que será exibida para o usuário, além de definirmos as ações que deverão ser executadas através de determinados eventos que podem ser lançados.

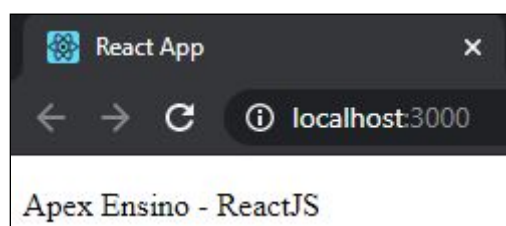
Veja abaixo um exemplo simples de como visualizar uma mensagem:

```
JS index.js ×  
src > JS index.js  
1 import React from 'react';  
2 import ReactDOM from 'react-dom';  
3  
4 ReactDOM.render(<p>Apex Ensino - ReactJS</p>, document.getElementById('root'));
```

Sempre que formos trabalhar com funcionalidades do ReactJS precisamos importar uma biblioteca, sendo assim a primeira linha criamos um objeto chamado **React** que importar da biblioteca **react**.

Para manipularmos elementos de uma página HTML precisamos habilitar o uso do DOM (Document Object Model), sendo assim na segunda linha é criado o objeto **ReactDOM**, que faz referência a biblioteca **react-dom**, será através dele que poderemos manipular o id **root** criado na página HTML.

Para finalizar temos o comando **render**, que pertence ao **ReactDOM**, esse comando fará com que algum componente no JavaScript possa ser enviado para a página HTML. No caso desse exemplo queremos exibir uma frase que está entre as tags `<p></p>`, e depois da vírgula referenciamos o elemento com o identificador **root** que está na página HTML, vamos ver o resultado:



Exibindo estruturas

Podemos exibir estruturas mais complexas como tabelas, formulários, listas ou uma estrutura envolvendo várias tags HTML simultaneamente.

Geralmente essas estruturas ficam salvas em classes ou constantes, nessa etapa vamos exibir uma listagem de cursos, veja o código abaixo para compreender a estrutura:

```
1 // Importações
2 import React from 'react';
3 import ReactDOM from 'react-dom';
4
5 // Criando uma constante com uma estrutura HTML
6 const lista = (
7   <ul>
8     <li>HTML</li>
9     <li>CSS</li>
10    <li>JavaScript</li>
11    <li>TypeScript</li>
12    <li>Node.js</li>
13    <li>ReactJS</li>
14    <li>Angular</li>
15  </ul>
16 );
17
18 // Exibindo constante no elemento com id root
19 ReactDOM.render(lista, document.getElementById('root'));
```

Na linha 06 é criada uma constante, entre os parênteses adicionamos uma listagem contendo sete cursos, e na linha 19 chamamos a constante para que o elemento seja exibido dentro da div com id **root** no arquivo HTML.

Exercícios

Utilizando como base a estrutura criada neste capítulo, crie uma constante para cada questão e exiba individualmente:

- a) Criar um formulário contendo os campos nome, senha e um botão de envio.
- b) Exiba uma tabela com as colunas: produto, marca e valor. Adicione cinco produtos.
- c) Exiba um vídeo e uma descrição.
- d) Crie uma estrutura contendo uma div para agrupar um título (h1) e dois parágrafos (p).
- e) Crie um calendário utilizando uma tabela. Dependendo do mês escolhido, caso haja algum feriado crie um link para uma página explicando sobre o feriado.

Capítulo 05 - JSX

O uso do JSX é algo muito comum quando utilizamos o ReactJS, pois sua estrutura é muito mais enxuta que do JavaScript convencional, mesmo assim o desenvolvedor poderá optar em utilizar o JavaScript puro se quiser.

JSX ou JavaScript XML é um padrão adotado pelo ReactJS que vai deixar seu código muito mais fácil de ser compreendido.

Para compreender melhor o uso do JSX veja a imagem abaixo:

```
1 // Importações
2 import React from 'react';
3 import ReactDOM from 'react-dom';
4
5 // Criando uma constante com uma estrutura HTML
6 const mensagem = <h1>Utilizando JSX :)</h1>;
7
8 // Exibindo mensagem no elemento com id root
9 ReactDOM.render(mensagem, document.getElementById('root'));
```

Agora vamos fazer o mesmo exemplo, mas sem o uso do JSX:

```
1 // Importações
2 import React from 'react';
3 import ReactDOM from 'react-dom';
4
5 // Criando uma constante com uma estrutura HTML
6 const mensagem = React.createElement('h1', {}, 'Não utilizando JSX :(');
7
8 // Exibindo mensagem no elemento com id root
9 ReactDOM.render(mensagem, document.getElementById('root'));
```

Note que precisamos utilizar o comando **createElement**, esse comando existe no JavaScript puro, e quando não utilizamos o JSX precisamos referenciá-lo em nosso projeto.

Expressões com JSX

As expressões com JSX têm como finalidade exibir informações que podem estar contidas em variáveis, constantes, funções com retornos e até mesmo cálculos. Vamos exemplificar realizando a soma de dois valores:

```
1 | // Importações
2 | import React from 'react';
3 | import ReactDOM from 'react-dom';
4 |
5 | // Criando uma constante com uma estrutura HTML
6 | const expressoes = <h1>{5 + 5}</h1>;
7 |
8 | // Exibindo mensagem no elemento com id root
9 | ReactDOM.render(expressoes, document.getElementById('root'));
```

Na linha 06 adicionamos as chaves {}, toda a expressão estará entre essas chaves. Na imagem acima é realizada a soma de 5+5, o navegador não irá retornar essa estrutura, mas sim o resultado.

Fechamento de tags obrigatório

No HTML possuímos tags que agrupam dados como: h1, p, section, figure, porém há tags que não possuem um fechamento como: img, input e br, no caso do React todas as tags HTML precisam ter um fechamento, veja por exemplo a estrutura de um input:

```
5 | // Criando uma constante com uma estrutura HTML
6 | const estrutura = <input type="text" />;
```

Agrupando elementos

Muitas vezes precisamos trabalhar com um conjunto de tags, como por exemplo dois parágrafos, uma imagem e um título e assim por diante, neste caso para que funcione é sempre necessário agrupar elementos utilizando tags como `div` e `forms`, veja a imagem abaixo para ver um exemplo:

```
1 | // Importações
2 | import React from 'react';
3 | import ReactDOM from 'react-dom';
4 |
5 | // Criando uma constante com uma estrutura HTML
6 | const expressoes = (
7 |   <div>
8 |     <h1>ReactJS</h1>
9 |     <p>Aprendendo na Apex Ensino!</p>
10 |   </div>
11 | );
12 |
13 | // Exibindo mensagem no elemento com id root
14 | ReactDOM.render(expressoes, document.getElementById('root'));
```

Na linha 06 abrimos um parênteses, em seguida adicionamos uma `div` para englobar uma tag de título e outra de parágrafo. Se não houver essa `div` para agrupar, haverá um erro de compilação:

```
Failed to compile

./src/index.js
Line 8:9:  Parsing error: Adjacent JSX elements must be wrapped in an enclosing tag.

6 | const expressoes = (
7 |   <h1>ReactJS</h1>
> 8 |   <p>Aprendendo na Apex Ensino!</p>
  |   ^
9 | );
```

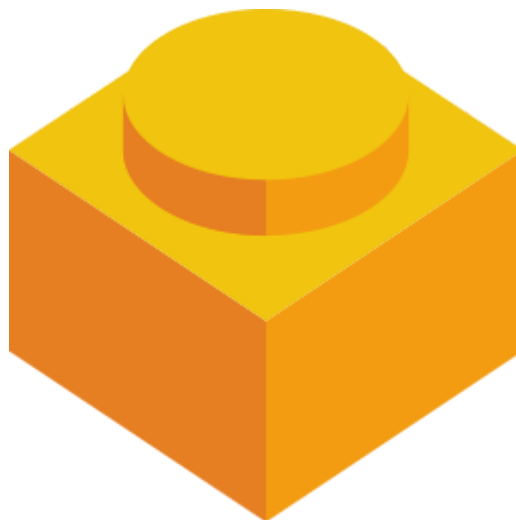
Capítulo 06 - Componentes

No ReactJS utilizamos a arquitetura de componentes, a ideia de utilizar essa arquitetura é criar funções que retornam componentes HTML.

Cada componente pode ser uma parte de uma página como um formulário, tabela, menu, rodapé, cabeçalho...

Para criar um componente utilizamos a palavra reservada **class**, importante ressaltar que não apenas a estrutura HTML é elaborada em um componente, mas também as funcionalidades através de eventos.

Se você já viu aquele brinquedo que podemos encaixar peças, podendo criar diversos tipos de objetos como carros, prédios, robôs, entre outras estruturas, podemos utilizar essa referência para compreendermos o uso de componentes no React, sendo assim um componente complementa outro.



Criando primeiro componente

Vamos criar um componente chamado Aluno, neste componente iremos exibir apenas uma frase de boas vindas com o nome do aluno:

```
1 // Importações
2 import React from 'react';
3 import ReactDOM from 'react-dom';
4
5 // Criando um componente chamado Pessoa
6 class Pessoa extends React.Component{
7     render(){
8         return <h1>Olá, meu nome é Ralf</h1>
9     }
10 }
11
12 // Exibindo mensagem no elemento com id root
13 ReactDOM.render(<Pessoa />, document.getElementById('root'));
```

Na imagem anterior você pôde ver a estrutura de um componente, a linha 06 onde temos a palavra reservada **class** realiza uma herança utilizando outra palavra reservada que é a **extends**, que chama o objeto **React** que possui uma funcionalidade chamada **Component**.

Na linha 07 é executado o render, que faz com que toda a estrutura que esteja dentro do render seja exibida, porém para que funcione precisamos retornar, sendo assim utilizamos a palavra reservada **return**.

Para finalizar, na linha 13 chamamos o componente **<Pessoa />**, assim toda a estrutura que está depois do return irá aparecer entre as tags **<div></div>** do arquivo HTML.

Unindo componentes

Vamos supor que teremos dois componentes e necessitamos que o componente **Aluno** chame o componente **Curso**, como podemos fazer isso?

Observe a imagem abaixo e veja como é simples executar essa função:

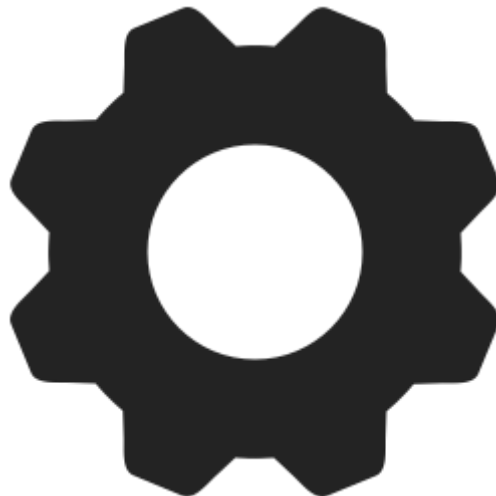
```
1 // Importações
2 import React from 'react';
3 import ReactDOM from 'react-dom';
4
5 // Criando um componente de Curso
6 class Curso extends React.Component{
7   render(){
8     return <p>O curso que faço na Apex é Front-End</p>;
9   }
10 }
11
12 // Criando um componente chamado Pessoa
13 class Pessoa extends React.Component{
14   render(){
15     return(
16       <div>
17         <h1>Olá, meu nome é Ralf</h1>
18         <Curso />
19       </div>
20     );
21   }
22 }
23
24 // Exibindo mensagem no elemento com id root
25 ReactDOM.render(<Pessoa />, document.getElementById('root'));
```

Na linha 06 é criado o componente **Curso**, esse componente retorna apenas um parágrafo. Para chamarmos esse componente, na linha 18 é feita a referência, note que estamos dentro do componente **Pessoa**, podemos chamar quantos componentes quisermos, lembrando que para mais de dois elementos é necessário estarem agrupados.

Capítulo 07 - Props

O termo props remete a propriedades que um componente pode ter, no HTML temos diversos atributos como id, class, name, value, placeholder, entre outras propriedades. O que muda do HTML para o ReactJS é que o nome das props são dadas pelo desenvolvedor, sendo assim não há palavras específicas para trabalhar com props.

Uma característica muito importante do props é que apenas é possível ler, não podemos realizar alterações em uma props. Futuramente você irá aprender como alterar dados com o comando **state**.



Exemplo prático

A ideia do props é enviar informações complementares para um componente, sempre que um componente chamar outro componente, podemos passar esses valores através de alguma propriedade.

Vamos supor um formulário para atualizar os dados de um produto, internamente temos um formulário com os campos fixos como por exemplo nome do produto e valor, para preencher os dados podemos passar essas informações através de props. Para compreender essa explicação, observe a imagem abaixo:

```
1  // Importações
2  import React from 'react';
3  import ReactDOM from 'react-dom';
4
5  // Criando um componente de Formulário
6  class Formulario extends React.Component{
7      render(){
8          return(
9              <form>
10                 <input type="text" value={this.props.produto} />
11                 <input type="text" value={this.props.valor} />
12             </form>
13          );
14      }
15  }
16
17  // Exibindo mensagem no elemento com id root
18  ReactDOM.render(
19      <Formulario produto="Computador" valor="2000" />,
20      document.getElementById('root')
21  );
```

Na linha 18 quando vamos informar o componente que será exibido em nossa página HTML, informamos as propriedades **produto** e **valor**, já nas linhas 10 e 11 utilizamos o comando: **{this.props.produto}** e **{this.props.valor}** para termos acesso às propriedades enviadas.

Construtor e super

Para quem já desenvolveu utilizando linguagens como Java, PHP, C# e TypeScript por exemplo, já utilizou o construtor, agora se você não teve a experiência de utilizar essa funcionalidade pode ficar tranquilo(a) que vamos abordar uma breve explicação.

O construtor é um método executado automaticamente, quando temos um construtor em nossa aplicação ele será a primeira ação a ser realizada, depois ele executa as demais funcionalidades do nosso componente.

Além do construtor temos uma outra palavra reservada que é a **super**, ela executa o construtor de outra classe, geralmente chamamos essa classe de classe pai, pois é uma classe importante que deriva funcionalidades a classes filhas.

Quando criamos um componente nós realizamos uma herança, note que utilizamos o comando **extends React.Component** para criar cada novo componente, neste caso o **React.Component** é a classe pai, sendo assim precisamos executar as funcionalidades dela primeiro e depois as demais de nosso componente, e como faremos isso? Bem podemos utilizar a palavra reservada **super**, que chama automaticamente o construtor da classe pai para o nosso componente.

Agora você pode estar se perguntando o motivo de fazermos o uso de construtores e da palavra **super**, a ideia é termos a garantia de que tenhamos acesso a propriedades dos nossos componentes antes de realizarmos o render.

Utilizando o construtor e o **super** garante que primeiro teremos os valores das propriedades que são passadas via props e depois podemos pegar esses valores e realizarmos ações específicas, como por exemplo enviar para um **state**.

Por falar em **state**, veremos o que ele faz no próximo capítulo, mas temos que saber que o **state** está diretamente ligado com o construtor e a palavra reservada **super**, a imagem a seguir mostra uma estrutura básica utilizando construtor e **super**:

```
1 // Importações
2 import React from 'react';
3 import ReactDOM from 'react-dom';
4
5 // Criando um componente de Formulário
6 class Formacao extends React.Component{
7
8     // Construtor
9     constructor(props){
10         super(props);
11     }
12
13     // Render
14     render(){
15         return <h1>O construtor será útil com state :)</h1>;
16     }
17 }
18
19 // Exibindo mensagem no elemento com id root
20 ReactDOM.render(<Formacao />, document.getElementById('root'));
```

Veja que o construtor que está na linha 09 fica antes da função **render()**, tudo que estiver entre as chaves do construtor será executado antes de qualquer coisa, garantindo alguma informação ou execução de funcionalidade.

Exercícios

Com base no capítulo 07 - props, renda as questões abaixo:

- a) Responda com suas próprias palavras: O que são props?
- b) Qual a finalidade de utilizarmos props?
- c) Como podemos alterar os dados de um props?
- d) Implemente um exemplo de formulário, onde as propriedades: nome e idade sejam enviadas e exibidas em uma tag de título (h1).
- e) Qual a finalidade do construtor?



Capítulo 08 - State

O uso de **states** no ReactJS é algo indispensável, **state** ou estados são informações que podem sofrer alterações a qualquer momento. Um state será um objeto que pode sofrer alterações a qualquer momento, sendo por algum evento de teclado ou mouse.

O **state** também complementa o uso do **props**, lembre-se que o **props** são propriedades que podem ser passadas para um componente, porém essas propriedades apenas podem ser exibidas e não alteradas, então como podemos fazer para manipular essa informação? Utilizando o state para armazenar o valor de props e exibir em elementos de formulário por exemplo.

Utilizamos os **states** no construtor do componente, assim se formos manipular essas informações via HTML, garantimos um local para poder armazenar as alterações realizadas.

Veja a imagem abaixo como criar um **state** em nosso componente:

```
// Construtor
constructor(props){
  super(props);
  this.state = {nomeDoState : "Algum valor..."}
}
```

Exemplo prático

Para compreender a utilização do state vamos criar um exemplo bem simples utilizando um campo de texto e exibindo em uma tag de título:

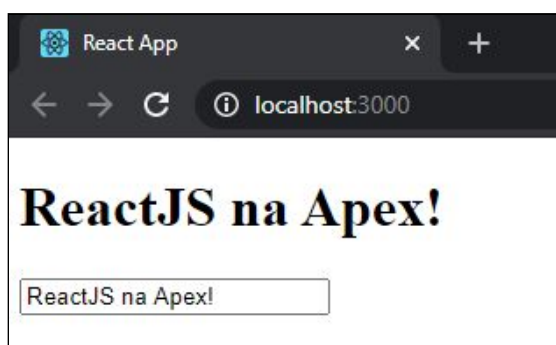
```
1 // Importações
2 import React from 'react';
3 import ReactDOM from 'react-dom';
4
5 // Criando um componente
6 class Pagina extends React.Component{
7
8     // Construtor
9     constructor(props){
10         super(props);
11         this.state = {valor : ""}
12     }
13
14     // Executa a ação ao digitar
15     acao = (elemento) => {
16         this.setState({valor : elemento.target.value})
17     }
18
19     // Render
20     render(){
21         return (
22             <div>
23                 <h1>{this.state.valor}</h1>
24                 <input type="text" onChange={this.acao} />
25             </div>
26         );
27     }
28 }
29
30 // Exibindo componente no elemento com id root
31 ReactDOM.render(<Pagina />, document.getElementById('root'));
```

Na imagem anterior podemos notar uma estrutura mais complexa, mas vamos por partes. Iniciamos pela linha 24, há um campo de texto com um evento **onChange**, ele será executado sempre que houver alguma alteração neste campo, note que para chamar uma função utilizamos o comando **this**, que irá referenciar alguma variável ou método do componente, seguido do nome da função.

Na linha 11 veja que temos a criação de um **state** chamado valor que inicialmente não possui nenhuma informação. O **state** quando criado precisa iniciar com algum valor.

Para finalizar precisamos criar um método para ser executado quando houver um evento **onChange**, na linha 15 há uma função que por parâmetro passa um elemento, esse elemento é a nossa caixa de texto, pois podemos utilizar esse evento para outros elementos, tornando a reutilização de uma função possível. Para realizarmos uma alteração no **state** utilizamos a função **setState**, que atribui um novo dado ao **state**.

Agora quando digitarmos conseguiremos exibir essa informação na nossa tag de título `<h1></h1>`, veja abaixo um exemplo:



Props e State

Muitas vezes algum componente é enviado com alguma propriedade e precisamos manipular em um campo de texto, caixa de combinação ou qualquer elemento com possibilidade de alterar a informação. Sendo assim podemos obter essa informação por **props** e adicionar o valor em um **state**, veja o exemplo abaixo:

```
5 // Criando um componente
6 class Pagina extends React.Component{
7
8     // Construtor
9     constructor(props){
10         super(props);
11         this.state = {valor : this.props.mensagem}
12     }
13
14     // Executa a ação ao digitar
15     acao = (elemento) => {
16         this.setState({valor : elemento.target.value})
17     }
18
19     // Render
20     render(){
21         return (
22             <div>
23                 <input type="text" onChange={this.acao} value={this.state.valor} />
24             </div>
25         );
26     }
27 }
28
29 // Exibindo component no elemento com id root
30 ReactDOM.render(
31     <Pagina mensagem="Aprendendo ReactJS" />,
32     document.getElementById('root')
33 );
```

Na linha 11 criamos um **props** chamado valor e chamamos o **props** através do comando **this.props.mensagem** para obter a mensagem que está sendo passada na linha 32.

Para finalizar na linha 23 adicionamos um **value** para poder exibir dentro da caixa de texto o valor do **state** e mudar dependendo do que for digitado.

Exercícios

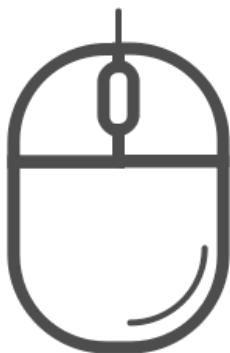
Com base neste capítulo, desenvolva as seguintes questões:

- a) O que é um state?
- b) Como criar um state?
- c) Qual a diferença entre um props e um state?
- d) Crie uma aplicação, onde o usuário digita um texto, em seguida retorne o texto e a quantidade de caracteres informados.
- e) Implemente um campo de texto, quando informada uma palavra, exiba a quantidade de vogais e consoantes.

Capítulo 09 - Eventos

No JavaScript a utilização de eventos é praticamente obrigatória, temos diversos como por exemplo `onClick`, `onChange`, `onDbClick`, `onKeyUp`, `onSubmit`, entre outros. O ReactJS segue o mesmo padrão, podemos reutilizar praticamente todos os eventos do JavaScript em suas aplicações utilizando essa biblioteca.

Neste capítulo iremos abordar os principais eventos, porém você pode acessar o site oficial do ReactJS (<https://reactjs.org/docs/events.html>) e averiguar todos os eventos possíveis que podem ser utilizados.



Evento Click

O evento de **onClick** pode ser utilizado em qualquer elemento HTML, a usabilidade é a mesma de um JavaScript convencional, sendo assim esse evento pode ser utilizado em imagens, textos, elementos de formulário, tabelas, entre outros componentes da linguagem HTML.

Na imagem a seguir exemplificamos o uso do evento de clique através de um botão, onde será exibida uma mensagem:

```
1 // Importações
2 import React from 'react';
3 import ReactDOM from 'react-dom';
4
5 // Criando um componente
6 class Pagina extends React.Component{
7
8     // Executa a ação ao clicar
9     acao = () => {
10         alert("Evento de clique")
11     }
12
13     // Render
14     render(){
15         return <button onClick={this.acao}>Clique aqui</button>
16     }
17 }
18
19 // Exibindo componente no elemento com id root
20 ReactDOM.render(<Pagina />, document.getElementById('root'));
```

Evento Change

O evento **onChange** é responsável por executar uma ação quando realizamos alguma alteração no elemento HTML. Vamos exemplificar o seu uso criando uma caixa de combinação contendo duas cidades, ao selecionar uma cidade, será exibido um texto condizente com a cidade:

```
5 // Criando um componente
6 class Pagina extends React.Component{
7
8     // Executa a ação ao selecionar uma cidade
9     acao = (elemento) => {
10
11         switch(elemento.target.value){
12             case "Blumenau":
13                 alert("Cidade onde é realizada a Oktoberfest")
14                 break
15
16             case "Brusque":
17                 alert("Cidade onde é realizada a Fenarreco")
18                 break
19
20             default:
21                 alert("Favor selecionar uma cidade")
22         }
23     }
24 }
25
26 // Render
27 render(){
28     return (
29         <select onChange={this.acao}>
30             <option>Escolha uma cidade</option>
31             <option>Blumenau</option>
32             <option>Brusque</option>
33         </select>
34     );
35 }
36 }
```

Evento Submit

O evento **onSubmit** é utilizado quando temos um elemento de submit em nosso formulário, veja abaixo como implementar esse evento em seus projetos ReactJS:

```
1  // Importações
2  import React from 'react';
3  import ReactDOM from 'react-dom';
4
5  // Criando um componente
6  class Pagina extends React.Component{
7
8      // Executa a ação ao enviar o formulário
9      acao = (evento) => {
10
11          // Não realiza a atualização na página
12          evento.preventDefault();
13
14          // Mensagem
15          alert("Enviando formulário");
16
17      }
18
19      // Render
20      render(){
21          return (
22              <form onSubmit={this.acao}>
23                  <input type="submit" />
24              </form>
25          );
26      }
27  }
28
29  // Exibindo componente no elemento com id root
30  ReactDOM.render(<Pagina />, document.getElementById('root'));
```

Exercícios

Com base nos conceitos abordados de ReactJS até este capítulo, elabore as questões abaixo:

- a) Peça uma idade e retorne se é maior de idade ou menor de idade.
- b) O usuário irá selecionar um valor, em seguida retorne o sucessor e o antecessor.
- c) Peça uma palavra, exiba a quantidade de caracteres e a quantidade de vogais e consoantes.
- d) Peça uma palavra e verifique se essa palavra é um palíndromo.
- e) Crie um componente chamado aluno com as propriedades: nome, nota1 e nota2. Exiba essas informações em um formulário.

Capítulo 10 - Formulário

Trabalhar com formulários é uma prática muito comum, porém como manipular isso através do ReactJS? Neste capítulo faremos um pequeno exemplo prático, mas antes disso você precisa ter o conhecimento em: props, state e eventos, caso tenha alguma dúvida favor rever os capítulos condizente com cada assunto e depois continuar essa etapa.

Exemplo prático

Para iniciarmos nosso exemplo com formulários, crie a seguinte estrutura:

```
1 // Importações
2 import React from 'react';
3 import ReactDOM from 'react-dom';
4
5 // Criando um componente
6 class Formulario extends React.Component{
7
8     // Construtor
9     constructor(props){
10         super(props);
11
12         this.state = {
13             produto:'',
14             valor:null
15         }
16     }
17 }
```

Inicialmente importamos as dependências necessárias para o funcionamento do ReactJS, criamos nossa classe e implementamos um construtor com props e state.

Vamos continuar nosso exemplo adicionando o seguinte código abaixo do construtor:

```
18 // Evento de teclado
19 eventoTeclado = (e) => {
20   let nome = e.target.name;
21   let valor = e.target.value;
22
23   this.setState({[nome]: valor})
24 }
25
26 // Executa a ação ao enviar o formulário
27 enviar = (evento) => {
28
29   // Não realiza a atualização na página
30   evento.preventDefault();
31
32   // Mensagem
33   alert("Produto: "+this.state.produto+"\nValor: "+this.state.valor);
34
35 }
```

Serão dois métodos, o **eventoTeclado** é um método que será disparado sempre que alterarmos alguma informação em um campo de texto, neste caso essa ação ocorrerá em dois campos, sendo eles o nome do produto e o valor.

Também há um método para efetuar o envio de dados, onde exibimos uma mensagem concatenando o nome do produto e o valor em um **alert**.

Nosso próximo passo é focar no **render**, assim podemos exibir nosso formulário com dois campos de texto e um botão.

```
37 // Render
38 render(){
39   return (
40     <form onSubmit={this.enviar}>
41       <input type="text" name="produto" placeholder="Produto" onChange={this.eventoTeclado} />
42       <input type="text" name="valor" placeholder="Valor" onChange={this.eventoTeclado} />
43       <input type="submit" />
44     </form>
45   );
46 }
47 }
```

Para finalizarmos este exemplo basta realizarmos a exibição em nosso arquivo HTML, para isso veja a imagem abaixo:

```
49 // Exibindo componente no elemento com id root
50 ReactDOM.render(<Formulario />, document.getElementById('root'));
```

Neste pequeno exemplo temos um formulário com dois campos, quando digitamos algo, os valores irão para seus respectivos states, e quando clicado no botão de envio, uma mensagem é exibida dependendo do valor contido nos states, assim podemos ter uma ideia melhor da utilização do ReactJS.

Você pode implementar validações como por exemplo: campos vazios, quantidade mínima de caracteres, máscaras, entre outras funcionalidades.

Exercícios

Com base nos assuntos vistos até este capítulo, elabore as questões propostas utilizando: eventos, props, state e componentizando as estruturas.

- a) Criar um sistema que realize a soma de dois valores. Valide se caso algum campo esteja vazio.
- b) Serão informadas três notas. Retorne a média e a situação do aluno, sendo que média 7 ou mais é aprovado e abaixo é reprovado.
- c) Informe dois valores, se forem iguais exiba a soma, se forem diferentes realize a multiplicação.
- d) O usuário informa um número, em seguida exiba a tabuada daquele número. Para essa questão você pode utilizar algum laço de repetição como: while, do/while ou for.
- e) Serão pedidos três números, em seguida retorne qual deles é o menor.

Capítulo 11 - CSS

Trabalhar com o front-end demanda também o uso do CSS, assim conseguimos deixar as páginas web mais bonitas, podendo manipular características como: fontes, bordas, sombras, margens, animações, entre outras opções.

CSS inline

A utilização de CSS inline ou em linha é muito simples, a ideia é estilizar um elemento na mesma linha onde é criado, veja abaixo um exemplo:

```
1 // Importações
2 import React from 'react';
3 import ReactDOM from 'react-dom';
4
5 // Criando um componente
6 class Pagina extends React.Component{
7
8     // Render
9     render(){
10         return <p style={{color: "green"}}>Utilizando CSS inline</p>
11     }
12 }
13
14 // Exibindo componente no elemento com id root
15 ReactDOM.render(<Pagina />, document.getElementById('root'));
```

CSS em JavaScript

Para trabalhar com CSS via JavaScript, podemos criar uma constante e criar seu estilo como se fosse um objeto:

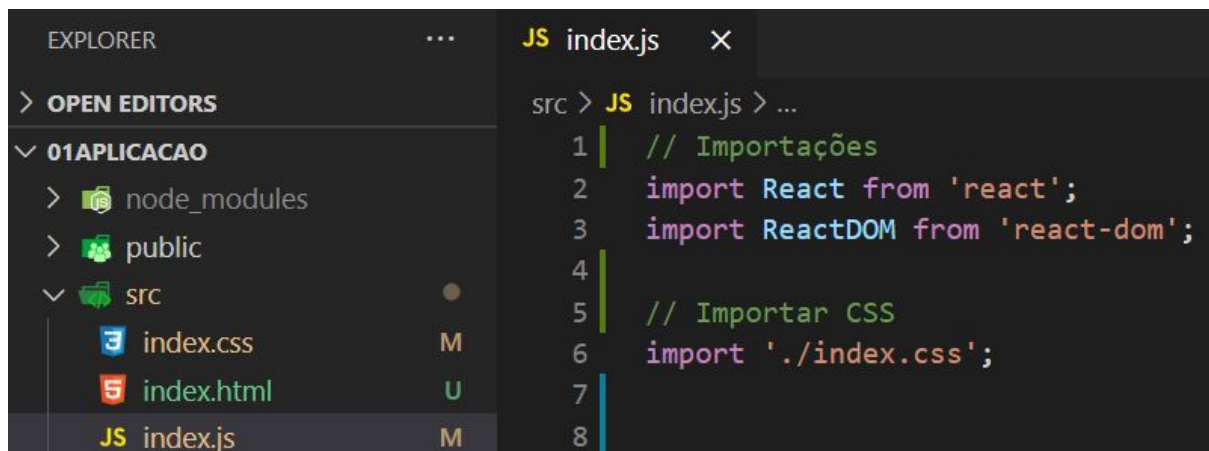
```
1 // Importações
2 import React from 'react';
3 import ReactDOM from 'react-dom';
4
5 // Criando um componente
6 class Pagina extends React.Component{
7
8     // Render
9     render(){
10
11         const meuEstilo = {
12             color: "white",
13             backgroundColor: "green",
14             padding: "5px"
15         }
16
17         return <p style={meuEstilo}>Utilizando CSS inline</p>
18     }
19 }
20
21 // Exibindo componente no elemento com id root
22 ReactDOM.render(<Pagina />, document.getElementById('root'));
```

Utilizando essa maneira podemos reutilizar essas características criadas em outros objetos, tornando nosso código reutilizável.

Anexando folhas de estilo

Em sites e sistemas maiores, o uso de arquivos CSS externos são muito utilizados, pois podemos utilizar em outros arquivos, além de não ter o CSS junto com nosso arquivo JavaScript que contém toda a estrutura do ReactJS.

A imagem abaixo exemplifica como podemos chamar um arquivo CSS e utilizar em nossos elementos:



```
EXPLORER
...
JS index.js X

src > JS index.js > ...
1 // Importações
2 import React from 'react';
3 import ReactDOM from 'react-dom';
4
5 // Importar CSS
6 import './index.css';
7
8
```

Note que na linha 06 realizamos uma importação, agora você pode estilizar tags HTML diretamente ou criar ids e classes.