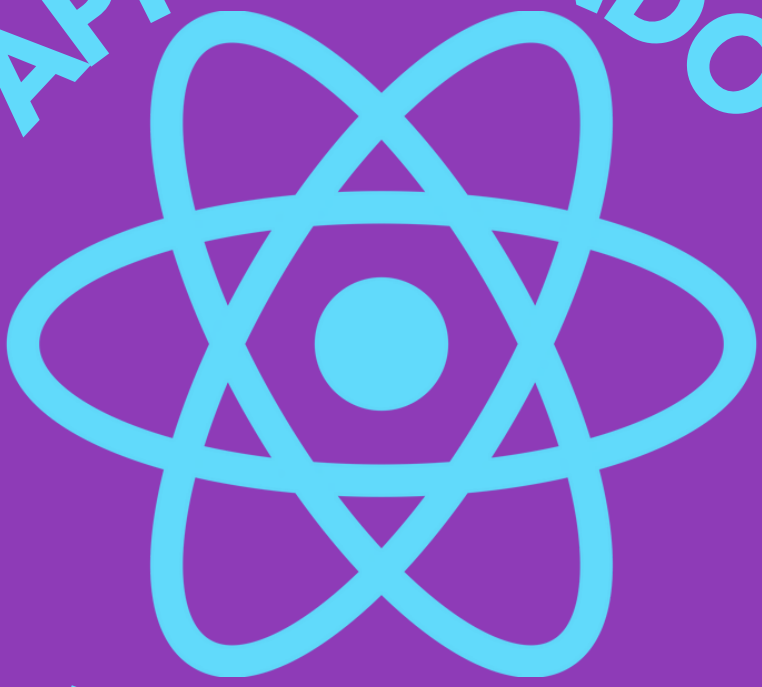
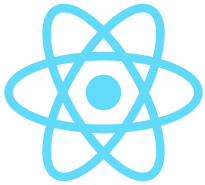
CODI academy

APRENDENDO DO ZERO



YURI MEDEIROS



PARTE 1: INTRODUÇÃO

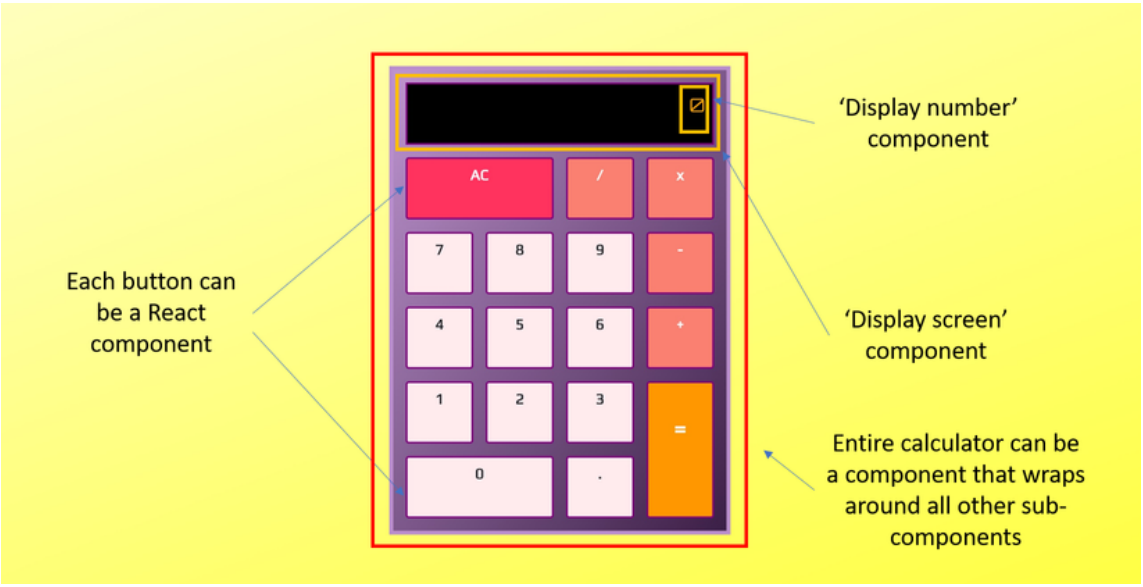
Para começar precisamos saber o que é o React. Para isso, nada melhor do que utilizar a definição de quem o criou, o próprio Facebook, que o define assim:

Uma biblioteca JavaScript declarativa, eficiente e flexível para criar interfaces visuais

Com isso, temos que:

1. *O React não é um framework, mas uma biblioteca (library).*
2. *O React serve para criar interfaces visuais (UI).*

Em poucas palavras, é uma biblioteca **JavaScript** para criação de interfaces para o usuário, desenvolvida e mantida pelo **Facebook**, sua primeira release saiu em 2013.
É uma lib open-source com mais de 1k de colaboradores ativos no GitHub.
Ele está presente no nosso dia-a-dia mais do que você imagina, em empresas grandes como Facebook, Instagram, AirBnB, NFL, Yahoo e muito mais. O mercado para essa biblioteca só cresce.
Sem sombras de dúvida uma das características que esta fazendo a comunidade adotar o **React** para o desenvolvimento de suas aplicações cada vez mais é a facilidade de aprendizagem.
A forma como seus códigos são declarativos, facilitando na visualização e a manutenção do código.



Se formos parar pra pensar bem, a principal função do React é organizar o conteúdo que será mostrado ao usuário.

PARTE 2: JSX

É uma extensão do JavaScript muito semelhante a uma string ou um pedaço de HTML.
O React não obriga o uso de JSX, porém ele nos auxilia muito com o modo visual, o que nos permite trabalhar com o código dentro do JavaScript.
Sem sombra de dúvidas é necessário utilizar o JSX durante o desenvolvimento com React, ele nos permite trabalhar com if, for, atribuir trechos de código a variável, passar por funções e muito mais.
Pegamos alguns exemplos do site do React de código com JSX e sem JSX pra você entender melhor os benefícios que eles nos traz.

Código sem JSX:

```
class Hello extends React.Component {
 render() {
 return React.createElement('div', null, `Hello ${this.props.toWhat}`);
  }
}

ReactDOM.render(
  React.createElement(Hello, { toWhat: 'World' }, null),
  document.getElementById('root')
)
```

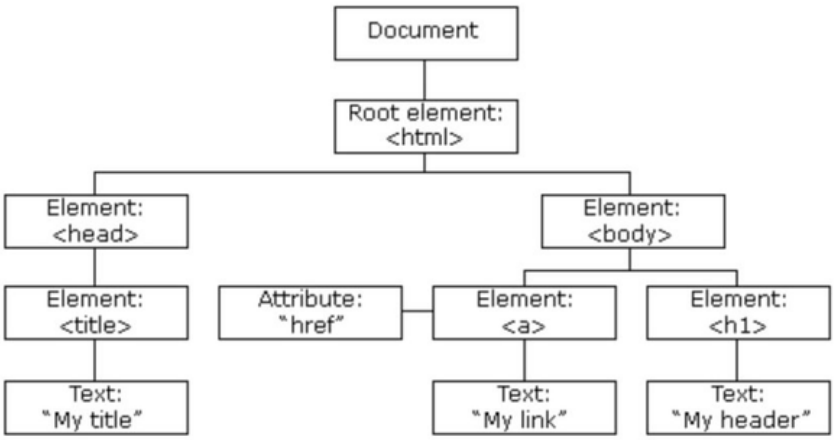
Código com JSX:

```
class Hello extends React.Component {
 render() {
 return <div>Hello{this.props.toWhat}</div>;
  }
}

ReactDOM.render(
  <Hello toWhat="World"/>,
  document.getElementById('root')
)
```

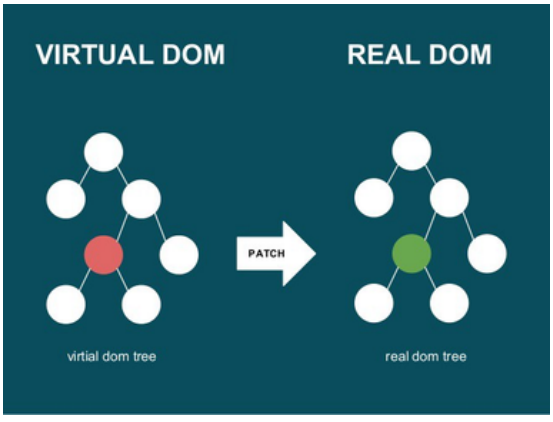
Explicando de forma rápida, o JSX é uma extensão de sintaxe do JavaScript (daí o nome, JavaScript Syntax eXtension) que nos permite escrever HTML dentro do JavaScript.
Nos primórdios, o JSX era um dos grandes motivos das críticas e piadas em relação ao React. Em teoria estávamos voltamos no tempo ao misturar HTML com JavaScript.

PARTE 3: DOM & DOM VIRTUAL

Antes de explicarmos o que é o famoso DOM Virtual do React, precisamos entender primeiro o que é o DOM.
O Modelo de Documento por Objetos (do inglês Document Object Model), como o próprio nome já diz, é uma representação do documento.
Assim que a página é carregada, o navegador cria o HTML DOM.
O documento é construído na estrutura de uma árvore, e seus objetos são os nós presentes nessa estrutura.

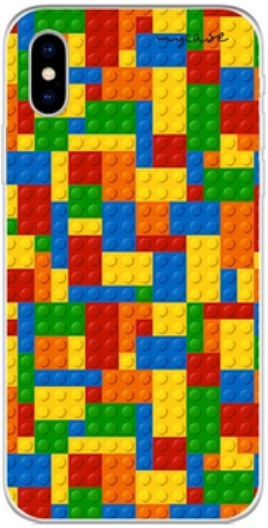


Quando um elemento precisa ser atualizado, você tem a opção de percorrer toda árvore ou procurar por um nó específico.
O problema é que o DOM não foi criado pra isso e ficar utilizando a API do DOM para modificar dados direto, em documentos grandes, nos causaria um problema, custaria caro.
O DOM Virtual é um conceito que já existia, porém depois do React ficou bem mais conhecido.
Ele atua como uma cópia do DOM, que pode ser atualizada com frequência sem nos acarretar problemas, e sem utilizar a API do DOM original.
Depois que todas as alterações – que devem ser persistidas no DOM – são finalizadas, ela será feita no DOM original da melhor maneira possível, nos evitando custos desnecessário, de maneira otimizada.
Esse é um dos principais motivos para os benchmarks de performance excelentes que o React tem.



PARTE 4: COMPONENTES

Um componente no React tem como finalidade separar as preocupações, acoplando cada vez mais as funcionalidades, e retornar um elemento React.
É necessário entender o que é e como funciona um componente para compreender melhor o funcionamento do React e aproveitar de todos seus benefícios.
Imagine que o Smartphone seja a aplicação, e cada peça Lego seja um componente.



Como foi dito anteriormente, o React tem como função principal auxiliar na criação da UI. Só deixei um detalhe importante de fora: o React usa componentes, e apenas componentes, para que que seja possível aumentar o máximo do reuso na sua aplicação.
Para exemplificar, vamos ver um pouco de código para entender como é um componente React:

```
import React from 'react';

export default function Button() {
  return (
    <button
      onClick={() => alert('Clicked')}
    >Click me</button>
  );
}
```

A função acima (sim, componentes React podem ser simplesmente funções em JavaScript) gera um botão que mostra um alert para o usuário ao ser clicado.
Se precisarmos de outro botão como esse, basta reaproveitá-lo sem a necessidade de adicionar mais código na aplicação. Lembrando que mais código é sinônimo de mais trabalho, mais testes unitários e mais potenciais bugs inseridos.
E se a necessidade for um botão parecido com esse, basta editarmos passando alguns parâmetros (vamos falar mais sobre eles na próxima seção) e transformá-lo no que precisamos.

Se houver algum bug no comportamento, ou mesmo no layout do botão, sabemos exatamente onde devemos mexer no código para fazer o conserto rápido.

Os componentes irão representar a parte da interface do usuário. Existem dois tipos de componentes, são eles: Stateless e Stateful.

Stateless

São componentes que não tem estados e podem receber propriedades.
As propriedades podem ser passadas durante sua declaração, pelo componente pai ou através do uso da arquitetura flux. Ele também pode passar propriedades para seus componentes filhos.

Stateful

É bem semelhante ao Stateless. Sua principal diferença é que ele possui um objeto de estado, apenas o componente onde o estado está presente que tem acesso e controle em cima dele.

Em resumo: o Stateless possui apenas um objeto de propriedades enquanto o Stateful possui um objeto de propriedades e um de estado.

Deixo aqui uma simples tabela pra você entender melhor o conceito de propriedades e estado de um componente e entender melhor quando usar cada um.

PROPS	STATE
Imutável	Mutável
Uma melhor performance	Escopo local
Pode ser passado para componentes filhos	Pertence apenas ao componente
	Mudanças de estados podem ser assíncronas
	Podem ser passado como propriedades para os componentes filhos.

PARTE 5: PROPS

Em todos os tipos de paradigmas no desenvolvimento de software, passar parâmetros é extremamente comum. Com os componentes do React isso não poderia ser diferente. A diferença é que no React, usamos os props (abreviação para properties).
A ideia é simples.

O componente seguinte, mostra para o usuário um Hello World:

```
import React from 'react';

export default function HelloWorld() {
  return (
    <h1>Hello World</h1>
  );
}
```

Imaginando que precisamos mudar a mensagem 'World' para alguma outra que for enviada dinamicamente, podemos reescrever esse componente usando as props dessa forma:

```
import React from 'react';

export default function HelloWithProps(props) {
  return (
    <h1>Hello {props.name}</h1>
  );
}
```

E com isso feito, podemos chamar esse componente dentro de outros assim:

```
import React from 'react';

import HelloWithProps from './hello-with-props';

export default function App() {
  return (
    <div>
      <HelloWithProps name="World" />
      <HelloWithProps name="Brazil" />
    </div>
  );
}
```

Além disso, podemos fazer uma validação das props que são passadas no componente para evitar bugs desnecessários e facilitar no desenvolvimento da aplicação usando os PropTypes:

```
import React from 'react';
import PropTypes from 'prop-types';

export default function HelloWithProps(props) {
  return (
    <h1>Hello {props.name}</h1>
  );
}

HelloWithProps.propTypes = {
   name: PropTypes.string.isRequired,
};
```

Agora, informamos explicitamente ao React para apenas aceitar a prop quando ela for uma string. Se qualquer outra coisa for passada para esse componente, a aplicação irá falhar e receberemos uma mensagem de erro nos avisando o porquê.

Aula 7: Hooks (Ganchos)

Resumidamente, Hooks é uma nova proposta que irá nos permitir utilizar estado, ciclo de vida, entre outras funcionalidades, sem a necessidade de escrevermos componentes com classe. A proposta já foi aceita e está disponível na versão 16.8 do React.

Os Hooks resolvem uma grande variedade de problemas encontrados durante o desenvolvimento de componentes. Por exemplo:

- Reutilização de lógica de estado entre components;
- Wrapper Hell (HOC, Render props—React DevTools);
- Componentes complexos e difíceis de se compreender;
- Componentes contendo grandes responsabilidades;
- Confusão ao utilizar classes (this, classes).

Imagine que você tenha um componente sem estado e ciclo de vida e, a partir deste momento, por alguma necessidade, precise desses recursos em seu componente?

```
import React from 'react'

interface Props {
  counter: number
}

function Counter({ counter }: Props) {
  return <p>{counter}</p>
}

export default Counter
```

No componente acima, apenas mostramos a propriedade counter recebida via props. Agora vamos necessitar de um estado local que será responsável por manter o valor do count. Surgindo essa necessidade, também aparece a obrigatoriedade de reescrevermos com classe.

Componente Counter com Classe:

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

interface Props {
  initialCount: number
}

interface State {
  count: number
}
```

```
class CounterClass extends Component<Props, State> {
  constructor(props: Props) {
    super(props)
    const { initialCount } = props
    this.state = {
      count: initialCount
    }
    this.increment = this.increment.bind(this)

  increment() {
    this.setState({ count: this.state.count + 1 })
  }

  render() {
    const { count } = this.state
    return (
      <button onClick={this.increment}>{count}</button>
    )
  }
} export default CounterClass
```

Trocando Class por Hooks

Agora temos a mesma lógica do componente anterior com Hooks. É notável a redução de escrita e a facilidade para compreender melhor como o componente irá se comportar.

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

interface Props {
  initialCount: number
}

function CounterHooks ({ initialCount }: Props) {
  const [count, setCount] = useState(initialCount)
  const increment = () => setCount(count + 1)
  return <button onClick={increment}>{count}</button>
}

export default CounterHooks
```

PARTE 7: create-react-app

Nos primeiros anos do React, uma das principais críticas era de que iniciar uma aplicação era muito mais complicado do que deveria ser, o que acabava excluindo diversos iniciantes. Antes de sequer começar a primeira linha de código em React, os desenvolvedores precisavam aprender no mínimo: Webpack e ES2015.
Hoje isso não é mais um problema, a equipe do Facebook vem trabalhando diariamente no projeto create-react-app.
Agora, com apenas 4 simples comandos, sem ter que criar e editar nenhuma configuração, temos um arcabouço completo para começar a desenvolver aplicações em React:

```
npm install -g create-react-app

create-react-app my-app
cd my-app/
npm start
```

PARTE 8: INSTALANDO

React é uma biblioteca JavaScript para construir interfaces de usuário. O React foi criado pelo Facebook e é mantido por uma comunidade de desenvolvedores e empresas individuais.

Por que usar o React?

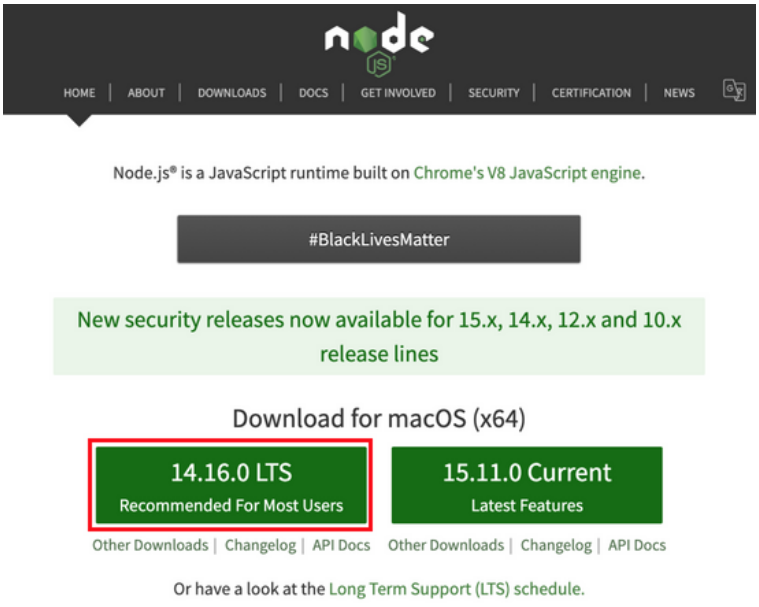
React é uma biblioteca JavaScript de código aberto criada pelo Facebook, não um framework (como o Angular). A principal diferença é que o React tem um foco principal, que é construir e renderizar interfaces de usuário (UI), e trabalha com um grande ecossistema de outras bibliotecas para fornecer roteamento, gerenciamento de estado e interação com servidores, serviços e APIs. Com esse foco, a criação de UI gira em torno da criação de visualizações declarativas e reutilizáveis simples chamadas Componentes . Componentes são pequenos pedaços encapsulados de código JavaScript que podem gerenciar seu próprio estado e visualizar a lógica. Eles podem ser facilmente compostos juntos para criar interfaces de usuário complexas (pense em blocos de construção). React irá atualizar e renderizar com eficiência apenas os componentes que precisam ser atualizados quando os dados do aplicativo são alterados . Isso torna o React simples, flexível e rápido.

Para executar React, nosso ambiente será composto pelas seguintes ferramentas de desenvolvedor:

- Node.js
- Node Package Manager - NPM
- Modern Web Browser (Google Chrome)
- React Developer Tools
- Code Editor (Visual Studio Code)
- Create React App

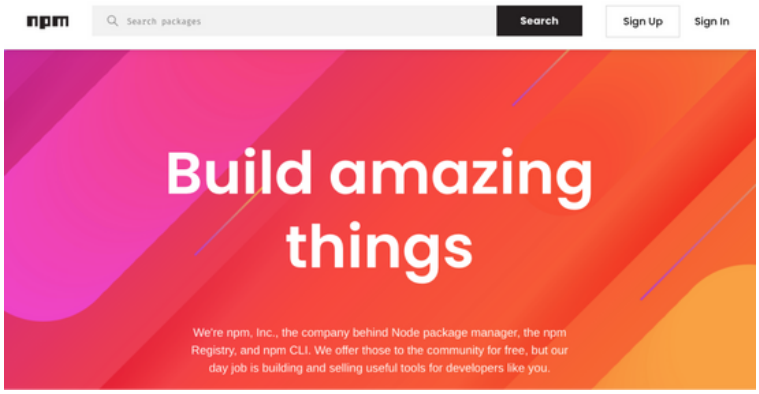
Node.js

Node é o tempo de execução que permite que o JavaScript seja executado em um servidor. Ele também permite ferramentas JavaScript modernas em seu computador e acesso a uma enorme biblioteca de código JavaScript conhecido como pacotes. Este será nosso principal uso de node e será discutido na próxima seção. Se você não tiver uma versão atual do Node instalada, visite o **site do Node** (https://nodejs.org/en/) para **baixar** e **instalar** a versão **LTS** mais recente para seu sistema operacional.



Gerenciador de pacotes de nós – NPM

Ao instalar o Node, como na etapa anterior, o Node Package Manager (NPM) e sua interface de linha de comando (CLI) são instalados ao mesmo tempo. O NPM é o gerenciador de pacotes do Node e hospeda milhares de pacotes de código gratuitos. Vamos orientá-lo no uso da CLI do NPM quando precisarmos instalar o código necessário durante o curso. Você pode visitar e pesquisar o site do NPM em https://www.npmjs.com/.

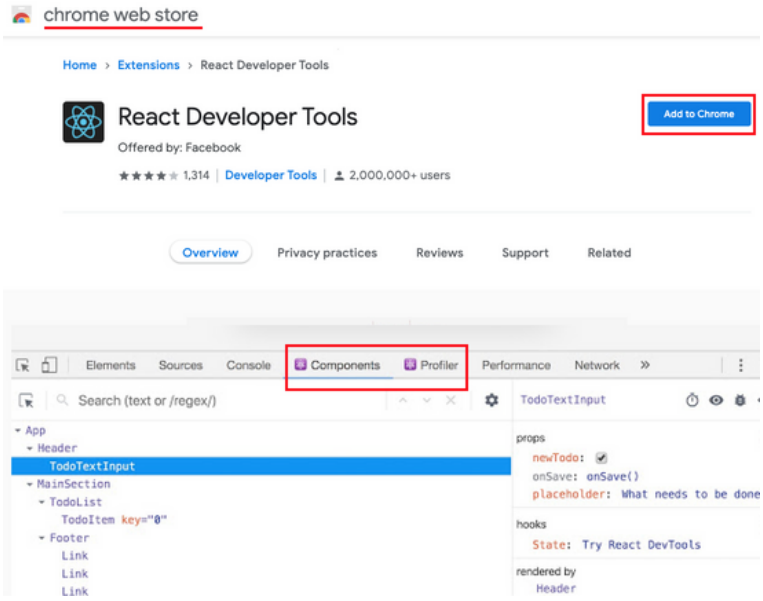


Navegador da Web moderno (Google Chrome)

Para o React Development, você pode usar qualquer navegador da Web moderno. Neste curso, estamos usando o Google Chrome com React Developer Tools que instalaremos em seguida. Se você não tiver o Chrome instalado, pode optar por pesquisar o Google Chrome e selecionar Download.

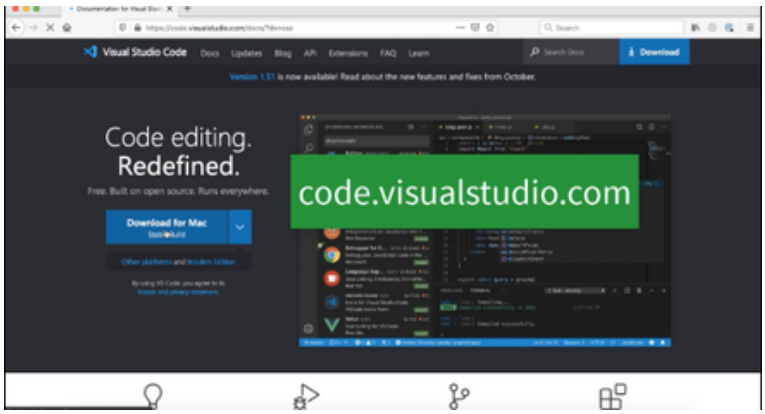
Ferramentas do desenvolvedor do React

Com o Chrome instalado, procure por Chrome Extension – React Developer Tools do Facebook, siga o link de pesquisa para a Chrome Web Store e Add To Chrome . Esta extensão permite que você inspecione os componentes do React no Chrome Dev Tools no navegador. Quando você tem um aplicativo React rodando com o Chrome Dev Tools aberto, você notará duas novas guias: Components e Profiler . Veremos isso com mais detalhes durante o curso.



Editor de código (Código do Visual Studio)

Quando se trata de editores de código, existem muitas boas opções para JavaScript. Neste curso, usaremos o Visual Studio Code (VS Code) que é gratuito, amplamente utilizado e disponível para todos os sistemas operacionais. Para instalar o VS Code, navegue até o site do VS Code (https://code.visualstudio.com/) e siga as instruções de instalação para seu sistema operacional.



Criar aplicativo React

Com a configuração do seu editor, escolha um local para armazenar seus arquivos de código. Se ainda não tiver um local, você pode simplesmente criar uma pasta em seus Documentos ou em sua área de trabalho para começar. No Mac, abra um terminal ou equivalente. No Windows, abra o Prompt de Comando do Windows, Bash, WSL, PowerShell ou equivalente.

Com um Explorador de Arquivos, abra o terminal na pasta do projeto escolhido. Como alternativa, com uma janela de terminal aberta, altere o diretório para a pasta do seu projeto. Neste exemplo de código da lição, criamos uma pasta na área de trabalho chamada React. Em seguida, mude para esse diretório do terminal para executar o seguinte comando Create React App do NPMpara configurar nosso projeto react-hello inicial .

```
npm init react-app react-hello
```

Se você estiver no Windows 10 e o comando react-app acima apresentar um erro , execute-o antes de tentar novamente:

```
npm install -g create-react-app
```

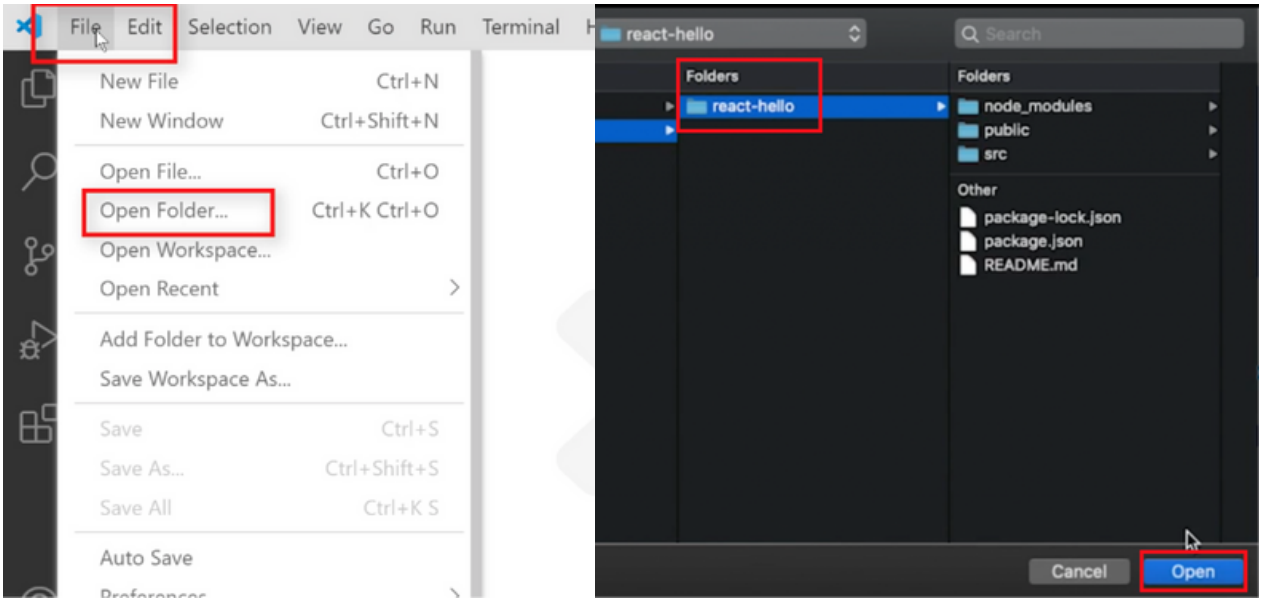
Na próxima lição, começaremos a codificar nosso aplicativo.

PARTE 9: HELLO WORLD

Na última lição, configuramos nosso ambiente de desenvolvimento e usamos Create-React-App para iniciar nosso aplicativo React hello world. Nesta lição, começamos a trabalhar em alguns fundamentos da codificação React.

Abrir projeto no editor

Dentro do VS Code , escolha no menu Arquivo | Abra a Pasta . Navegue até o local da pasta react-hello que criamos na última lição e abra o projeto.



O único arquivo src que usaremos nesta lição é o **index.js** para que os outros arquivos possam ser excluídos ou ignorados. Em seguida, abra o arquivo index.js e exclua todo o seu conteúdo.

Vamos criar um Componente React simples. A primeira etapa é sempre importar a própria biblioteca React (instalada na pasta node_modules do NPM quando criamos o projeto na última etapa). Além disso, no arquivo index.js principal , importamos o renderizador DOM do React chamado ReactDOM . É importante usar as letras maiúsculas e minúsculas exatas importando React, ReactDOM e as strings que identificam a fonte do código. Quando você vê um nome entre aspas ('react') sem informações de caminho relativo, você sabe que a fonte do código é de um pacote NPM instalado .

```
import React from 'react';
import ReactDOM from 'react-dom';
```

A próxima etapa é criar uma função JavaScript que retorne um elemento HTML . Por si só, a sintaxe semelhante a HTML retornada da função não é JavaScript (JS) válido, mas é compilada para JS válido.

```
function HelloWorld() {
  return <div>Hello World!</div>}
```

O passo final é dizer ao React para renderizar nosso Component simples para a página da web Document Object Model (DOM). Isso é feito usando a segunda parte do código React que importamos na parte superior do arquivo, ReactDOM , que recebe dois argumentos: o Componente a ser renderizado e o local no DOM onde queremos renderizá-lo. Fornecemos o Component como uma tag de elemento HTML e fornecemos a localização usando um método de API da Web , como getElementById('root') ou querySelector('#root') .

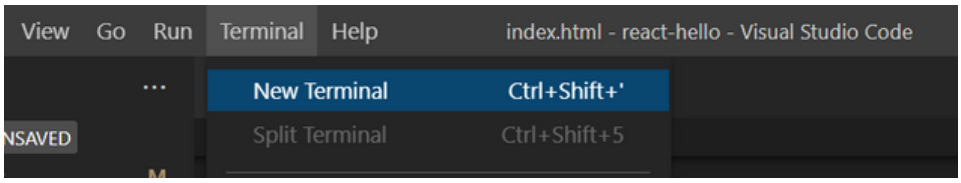
```
ReactDOM.render(
  <HelloWorld/>,
  document.getElementById('root')
);
```

O root no código acima é um div com um id de root no arquivo index.html na pasta pública do código configurado na lição anterior pelo comando Create-React-App que rodamos no terminal. O corpo do arquivo index.html contém o seguinte:

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

Executar projeto do terminal

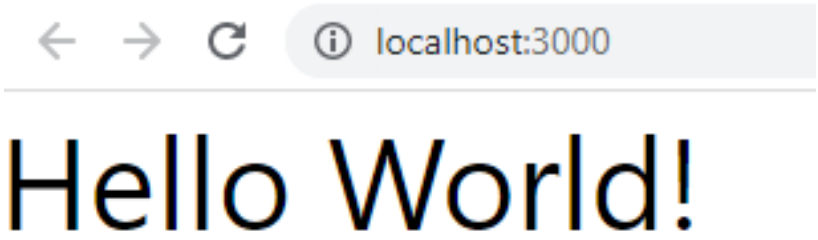
Você pode usar o terminal do seu computador ou o Terminal Integrado do VSCode escolhendo no menu Terminal | Novo Terminal.



No terminal de sua escolha, abra a pasta react-hello, inicie o aplicativo com o seguinte comando:

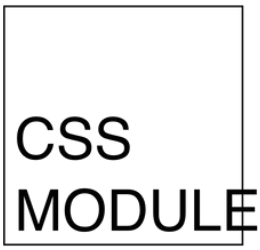
```
PS C:\Users\info\react-hello> npm start
```

Este comando inicia nosso aplicativo React e o exibe no navegador (localhost:3000 por padrão).



PARTE 10: CSS IN JS

Todas as boas práticas ao se escrever CSS apontavam para um mesmo caminho por anos: escrever CSS o mais desacoplado possível do JavaScript e do HTML, usando nomes bem descritivos para as classes.
O problema dessa abordagem é que ela não resolve o maior defeito do CSS: tudo o que você cria é global. Cada vez que você cria uma nova classe CSS ela é, por definição, global. Isso gera diversos problemas na manutenção, principalmente em aplicações maiores.
Com o grande sucesso da componentização no React, a forma de se escrever CSS vem mudando drasticamente.
A primeira revolução veio com o CSS Modules.



A ideia é simples: escrever o mesmo CSS de sempre, só com uma diferença: esse CSS só vai valer para aquele componente. Ou seja, será um módulo (daí o nome, CSS Modules).

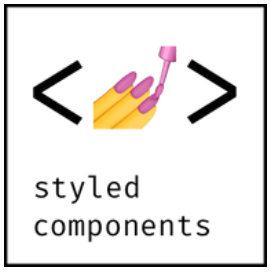
Um exemplo simples de uso do CSS Modules:

```
import React from 'react';

import styles from './styles.css';

const Main = () => (
  <button
    onClick={() => alert('Clicked')}
    className={styles.button}
  >Click me</button>
);

export default Main;
```

E o arquivo de estilos que só valerá para o componente acima:

```
.button {
  background-color: blue;
  border: none;
  color: white;
  padding: 5px 10px;
}
```

A segunda revolução veio com o styled-components.



Nesse, o próprio componente já teria seus estilos escritos em JavaScript, sem que nenhum arquivo CSS seja sequer criado.
Uma das vantagens do styled-components, é que o code-splitting fica muito mais fácil de ser feito, e assim como o CSS Modules, não existe CSS global. A desvantagem é que você não escreve o CSS padrão que muitos já estão acostumados, o que pode não ser muito intuitivo.

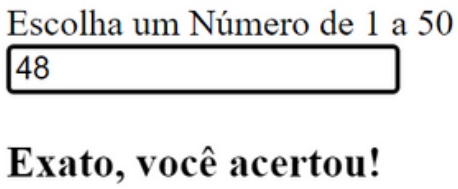
Um exemplo simples:

```
const Button = styled.a`
  display: inline-block;
  border-radius: 3px;
  padding: 0.5rem 0;
  margin: 0.5rem 1rem;
  width: 11rem;
  background: transparent;
  color: white;
  border: 2px solid white;
  ${props => props.primary && css`
    background: white;
      color: palevioletred;
   `}
`
```

PRÁTICAS

A seguir, vamos colocar os conhecimentos em prática realizando duas práticas completas,

A primeira é um minigame onde o usuário precisar chutar números até acertou qual número foi sorteador pelo sistema.



E a segunda é um Quiz, onde o usuário terá diversar alternativas para escolher e acertar.