A large, abstract graphic on the left side of the slide consists of several thick, light blue curved lines forming a complex, organic shape against a dark gray background.

2022

Residência de Software

React JS

Como dar instruções?

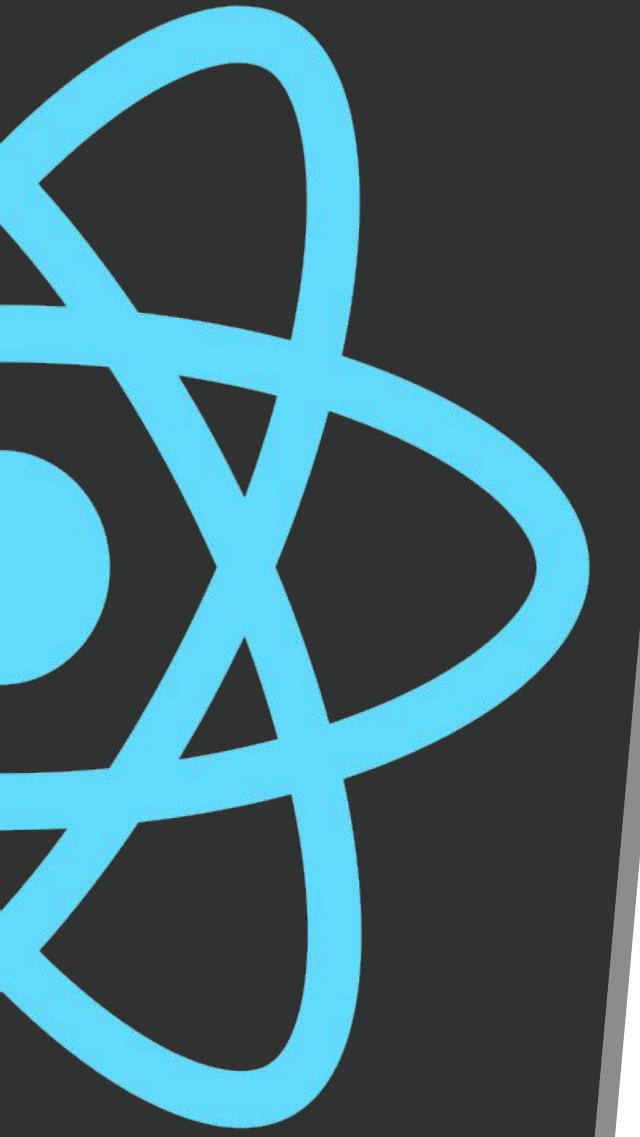


O que vamos aprender?

- Recapitação Javascript - ES6
- O que é react
- Criação de projetos com React
- Componentização
- Criação de interfaces dinâmicas
- Renderização condicional
- Manipulação de estados
- Consumo de APIs - AXIOS
- Funções Assíncronas
- Styled Components
- Gerenciadores de Pacotes

Avaliações

- Avaliação conceito (10 pontos)
 - Formulário de Auto Avaliação
- Avaliação teórica (30 pontos)
 - Formulário de múltipla escolha
- Projeto final em grupo (60 pontos)
 - Desenvolvimento de uma aplicação frontend com consumo de API

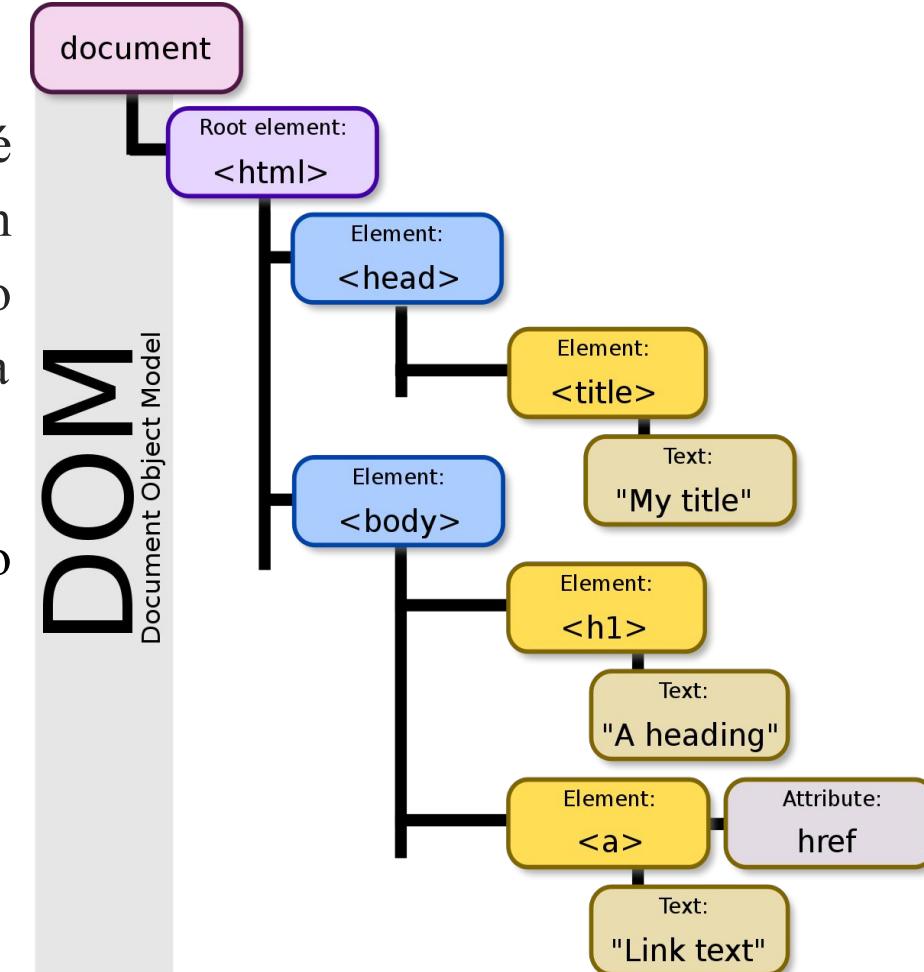


Recapitulação

❖ O que é DOM?

Quando uma página da web é carregada, o navegador cria um modelo de objeto de documento (do inglês Document Object Model) da página.

O modelo HTML DOM é construído como uma árvore de objetos:

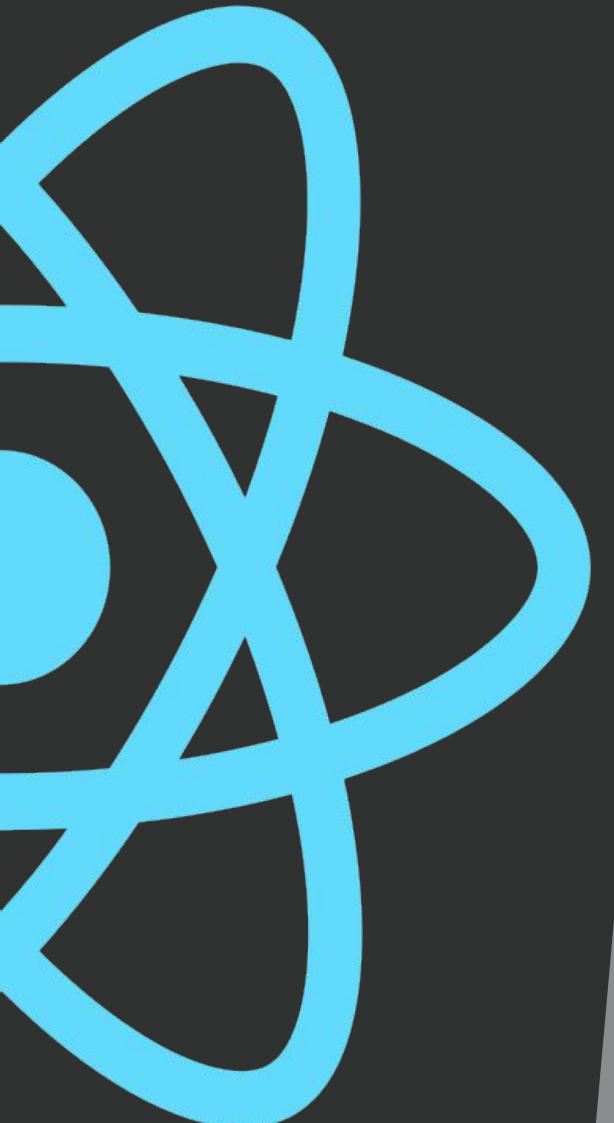


Com o modelo de objeto, o JavaScript obtém todo o poder de que precisa para criar HTML dinâmico, sendo capaz de:

- Alterar todos os elementos HTML na página
- Alterar todos os atributos HTML na página
- Alterar todos os estilos CSS na página
- Remover elementos e atributos HTML existentes
- Adicionar novos elementos e atributos HTML
- Reagir a todos os eventos HTML existentes na página
- Criar novos eventos HTML na página

- Qualquer coisa criada pelo navegador Web no modelo da página Web poderá ser acessado através do objeto Javascript document;
- Usa-se o DOM principalmente para atualizar uma página Web
- Pode-se mover itens dentro de uma página ou criar efeitos CSS sem precisar recarregar a página.

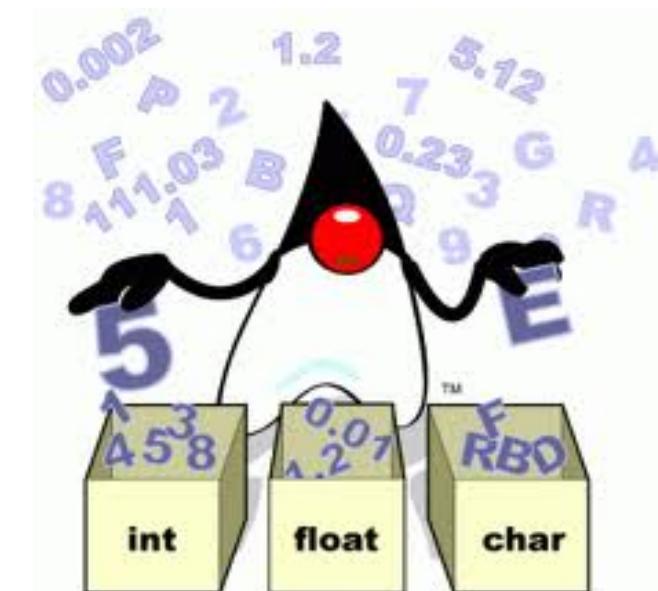
Método	Descrição
<code>document.getElementById(id)</code>	Encontra um elemento por id de elemento
<code>element.setAttribute(attribute, value)</code>	Altera o valor do atributo de um elemento HTML
<code>document.createElement(element)</code>	Cria um elemento HTML
<code>document.getElementById(id).onclick = function(){code}</code>	Adicionando código de manipulador de eventos a um evento onclick



Tipos de Dados

Vamos conhecer alguns tipos de dados existentes em JavaScript

- ❖ String
- ❖ Number
- ❖ Boolean
- ❖ Undefined e Null
- ❖ Object
- ❖ Array



String é uma cadeia de caracteres. Podem ser representados através das aspas simples (''), aspas duplas ("") ou template strings (` `).

As Template Strings permitem que trabalhemos com strings multi-linhas e utilizemos expressões de linguagem utilizando a formatação \${}.

```
1
2 var nome = "Pedrinho"
3 var frase = `Acorda ${nome}, que hoje tem campeonato`
4 //Acorda Pedrinho, que hoje tem campeonato
5
```

NUMBER

São tipos de dados os quais conseguimos manipular de forma numérica.

```
23 // Int (Inteiro)  
3.14 // Float (Real)  
NaN // Not a Number  
Infinity // Infinito
```

True ou False. Muito utilizados em funções condicionais, entre outros.



Undefined e null são dois tipos de caracteres muito confundidos no JavaScript, porém, entender a diferença entre eles é crucial durante o desenvolvimento das nossas aplicações.

- Undefined: Valor indefinido - Algo que não existe
- Null: Valor nulo - Objecto que não tem nada dentro dele

OBJECT

Um objeto é composto de Propriedades/Atributos e Funcionalidades/Métodos. É necessária uma atenção especial neste tipo de dado pois ele estará presente frequentemente em nosso cotidiano.

```
const pessoa ={
    nome: "Matheus",
    idade: 23,
    endereco:{
        cidade: "Rio de Janeiro",
        estado: "Rio de Janeiro"
    },
    gritar: function(){
        console.log("AAAAAAAAAAAAAAA");
    }
}
```

Acessando Propriedades/Atributos e Funcionalidades/Métodos.

```
const pessoa ={
    nome: "Matheus",
    idade: 23,
    endereco: {
        cidade: "Rio de Janeiro",
        estado: "Rio de Janeiro"
    },
    gritar: function(){
        console.log("AAAAAAAAAAAAAAAAAAAAAA");
    }
}

console.log(pessoa.nome);
console.log(pessoa.endereco.cidade);
console.gritar();
```

ARRAY

É uma lista, ou seja um agrupamento de dados. Podem receber qualquer tipo de dados dentro de si;

```
const bandasEmos = ["Fresno", "NXZero", "My Chemical Romance"]
const meuArray = [32, 441, "Matheus", true, {nome: "Thiago", idade: 24}, 12.45]
```

Em casos como esse, onde não há um outra parte do código onde queremos referenciar uma função e ela será apenas referenciada ao invés de chamar a função, podemos usar o conceito de função anônima, já criando a função no lugar onde antes apenas indicamos seu nome. Por exemplo :

```
var inputTamanho = document.getElementById("inputTamanho")
var outputTamanho = document.getElementById("outputTamanho")

inputTamanho.oninput = function(){
    outputTamanho.value = inputTamanho.value;
}
```

Manipulando Strings

Uma variável que armazena um string faz muito mais que isso! Ela permite, por exemplo, consultar o seu tamanho e realizar transformações em seu valor. Por exemplo :

```
var palavraTosca = 'Catapora'  
palavraTosca.length; //Tamanho da String  
palavraTosca.replace("pora", "pimba");
```

Manipulando Strings

Assim como em Java, podemos converter uma String para inteiro ou ponto flutuante usando o método parseInt e parseFloat :

```
var textoInteiro = "10"
var inteiro = parseInt(textoInteiro, 10)
var textoFloat = "53.94"
var float = parseFloat(textoFloat)
```

Números, assim como strings, também são imutáveis. O exemplo abaixo altera o número de casas decimais com a função `toFixed`. Esta função retorna uma string, mas, para ela funcionar corretamente, seu retorno precisa ser capturado:

```
var milNumber = 1000
var milString = milNumber.toFixed(2) //Recebe o retorno da função
console.log(milString)// Imprime a string "1000.00"
```

Arrays em Javascript

A utilização de arrays em javascript não é muito diferente do que foi visto em Portugol ou Java. Os pontos interessantes são que, por javascript não ser tipado, podemos armazenar valores de tipos diferentes em vetores no Javascript.

```
var variosTipos = ["Musica", 45, [2,4]]  
console.log(variosTipos[0]);
```

Arrays em Javascript

Para adicionar elementos ao vetor, podemos utilizar a função push que adiciona um elemento na última posição do array ou adicionar direto em um índice selecionado :

```
var cursos = ["Agronomia", "Tecnologia da Informação"]
cursos.push("Direito");
cursos[9] = "Medicina";
console.log(cursos);
```

Arrays em Javascript são
semelhantes às ArrayLists
em Java



Manipulando Arrays

- `.length()` => Permite ver o comprimento do array
- `.join(",")` => Junta todos os elementos do array em uma string
- `.pop()` => Remove o último elemento do array
- `.shift()` => Remove o primeiro elemento do array
- `.push()` => Adiciona um elemento ao final do array
- `.indexOf()` => Localiza um elemento no array
- `.find()` => Localiza um elemento no array

Concatenando Arrays

```
var umArray = [1,2,3]
var outroArray = [6,7,8]
var principalArrayConectado = [...umArray, 4,5, ...outroArray, 9,10,11]
console.log(principalArrayConectado);
```

ES6

Laços de Repetição e Condicionais

Todas as estruturas de laços de repetição e condicionais que vimos em disciplinas passadas funcionam em javascript. Apenas alguns exemplos de sintaxe:

```
✓ while(condutor <= 10){  
    //código a ser repetido  
}
```

```
if(condicao){  
    //código a ser executado  
}
```

```
for(*variável de controle*; /*condição*/; /*pós execução*/){  
    //código a ser repetido  
}
```

Em JavaScript, podemos criar um timer para executar um trecho de código após um certo tempo, ou ainda executar algo de tempos em tempos.

A função setTimeout permite que agendemos alguma função para execução no futuro e recebe o nome da função a ser executada e o número de milissegundos a esperar:

```
function executar(){
    console.log("Rodou...");
}

setTimeout(executar, 1000)
```

Se for um código recorrente, podemos usar o setInterval que recebe os mesmos argumentos mas executa a função indefinidamente de tempos em tempos:

```
function executando(){
    console.log("Executou...");
}

setInterval(executando, 1000)
```

É uma função útil para, por exemplo, implementar um banner rotativo, apresentado no exercício à seguir.

- Desestruturação consiste em você extrair do objeto apenas as propriedades que precisa. Muito útil para objetos que possuem muitas propriedades.

```
const{nome, idade, endereco, gritar}= pessoa
const{cidade} = endereco

console.log(nome);
console.log(idade);
console.log(cidade);
gritar();
```

ES6

Desestruturação no JavaScript

Funciona também com arrays, porém como o array utiliza [] a desestruturação fica dessa maneira:

```
var letras = ["A", "B", "C", "D"]
const [primeiro, segundo] = letras
```

```
var letras = ["A", "B", "C", "D"]
const [primeiro,,terceiro] = letras
```

ES6

Spread & Rest Operator...

Ambos são bem parecidos e podem gerar muita confusão porém, existem suas diferenças. Spread tem a ideia de “espalhar” os dados, enquanto o Rest tem a ideia de “pegar o resto dos dados”.

Este operador é muito útil para manipulação de Arrays, Objetos e também na implementação de funções mais dinâmicas.



...Spread

```
var umArray = [1,2,3]
var outroArray = [6,7,8]
var principalArrayConectado = [...umArray, 4,5, ...outroArray, 9,10,11]
console.log(principalArrayConectado);
```

ES6

...Rest

```
var nomes = ["Claudio", "Isabella", "Matheus", "Daniele", "Thiago"]
const [primeiro, ...resto] = nomes

console.log(primeiro);
console.log(resto);
```

ES6

Aplicação em objetos ...Spread

```
const pessoa ={
    nome: "Matheus",
    idade: 23,
    endereco:{
        cidade: "Rio de Janeiro",
        estado: "Rio de Janeiro"
    },
    gritar: function(){
        console.log("AAAAAAAAAAAAAAAAAAAAA");
    }
}

const pessoa2 = {...pessoa, idade: 20}
console.log(pessoa2);
```

ES6

Aplicação em objetos ...Rest

```
const pessoa ={
    nome: "Matheus",
    idade: 23,
    endereco:{
        cidade: "Rio de Janeiro",
        estado: "Rio de Janeiro"
    },
    gritar: function(){
        console.log("AAAAAAAAAAAAAAA");
    }
}

const {nome, ...resto} = pessoa
console.log(nome);
console.log(resto);
```

ES6

Aplicação em funções ...Spread

```
function somarNumeros(n1,n2){  
    console.log(n1+n2);  
}  
  
var numeros = [2, 3]  
somarNumeros(...numeros)
```

ES6

Aplicação em funções ...Rest

```
function nomesComRest(...nomes){  
    console.log(nomes);  
}  
  
nomesComRest("Carlin", "Jão", "JuJu", "Caio")  
nomesComRest("Carlin", "Caio")
```

ES6

Arrow Functions =>

Uma expressão arrow function possui uma sintaxe mais curta quando comparada a uma expressão de função.

```
ola = _ => 'Olá'
```

ES6

Arrow Functions =>

```
let ola = function(){  
    return 'Olá'  
}
```

```
ola = _ => 'Olá'
```

ES6

Método Map

O método `map()` invoca a função callback passada por argumento para cada elemento do Array e devolve um novo Array como resultado.

```
const registros = [
  {name:'REACT', aplicacao: 'Front-end'},
  {name:'.NET', aplicacao: 'Back-end'},
  {name:'CSS', aplicacao: 'Front-end'},
  {name:'SQL', aplicacao: 'Banco de dados'},
  {name:'REACT-NATIVE', aplicacao: 'Front-end'},
  {name:'JAVA', aplicacao: 'Back-end'},
  {name:'JAVASCRIPT', aplicacao: 'Front-end'},
  {name:'QUARKUS', aplicao: 'Back-end'}
]

const novoArray = registro.map((res, index) =>{
  return `Nome: ${res.nome} | Aplicação: ${res.aplicacao} | Posição: ${index}`;
})

console.log(novoArray);
```

ES6

Método Filter

- O método `filter()` cria um novo array com todos os elementos que passaram no teste implementado pela função fornecida.

```
const tecnologias = ['REACT', '.NET', 'CSS', 'SQL', 'JAVA', 'JAVASCRIPT', 'REACT-NATIVE']
const tecnologiasFiltradas = tecnologias.filter((tec)=>{
    return tec.toLocaleLowerCase().includes('j')
})

console.log(tecnologiasFiltradas);
```

```
const numeros = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
const numerosFiltrados = numeros.filter(num => num > 5)

console.log(novoArray);
```

ES6

Método Reduce

- O método `reduce()` executa uma função reducer (fornecida por você) para cada elemento do array, resultando num único valor de retorno.

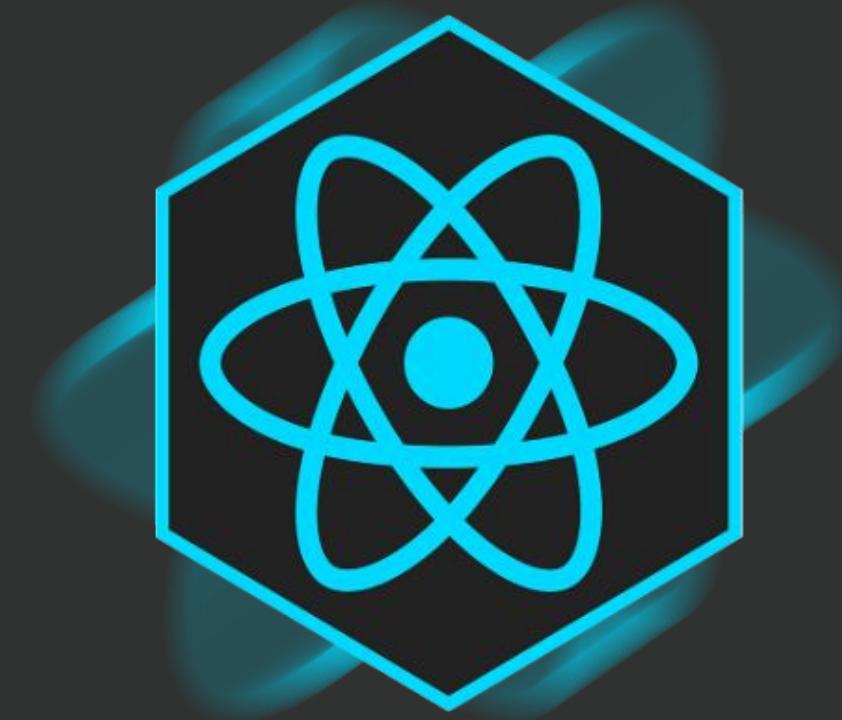
```
const numeros = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
const result = numeros.reduce((acumulador, valorCorrente) => acumulador + valorCorrente)

console.log(result);
```

ES6

05

Apresentação



REACT JS

História

- ❖ Biblioteca JavaScript (React.js ou ReactJS) criada em 2013.
- ❖ Foco em criar interfaces de usuário (frontend) em páginas web.
- ❖ É mantido pelo Facebook, Instagram, outras empresas e uma comunidade de desenvolvedores individuais.
- ❖ Em 2015, o Facebook anunciou o módulo de React Native, que em conjunto com o React, possibilita o desenvolvimento de aplicativos para Android e IOS
- ❖ Utiliza componentes de interface de usuário nativos de ambas plataformas, sem precisar recorrer ao HTML.
- ❖ Em uma pesquisa de 2018 sobre hábitos de desenvolvedores do site Stack Overflow, o React foi a terceira biblioteca ou framework mais citado pelos usuários e desenvolvedores profissionais, ficando atrás somente do Node.js e Angular, respectivamente.

O que é React?

- De acordo com a própria documentação, o React nada mais é do que uma biblioteca JavaScript para criar interfaces de usuário.
- Quando falamos em React, devemos sempre trazer em mente dois conceitos fundamentais da linguagem. Componentização e Manipulação de estados (states)
- Com React é possível criar componentes encapsulados que gerenciam seu próprio estado. Como a lógica do componente é escrita em JavaScript e não em templates, podemos facilmente passar diversos tipos de dados a longo da nossa aplicação e ainda manter o estado fora do DOM.

Por que o React?

- Simples
- Fácil de aprender
- Abordagem nativa
- Fácil testabilidade
- Componentização
- Comunidade Ativa
- Facilidade e diversidade de instalações de pacotes
- Criação de SPAs (Single Page Applications)

Algumas aplicações que utilizam React:

- ✓ Facebook
- ✓ Uber
- ✓ Instagram
- ✓ WhatsApp
- ✓ Khan Academy
- ✓ Airbnb
- ✓ Dropbox
- ✓ Flipboard
- ✓ Netflix
- ✓ Paypal

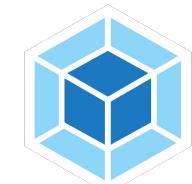
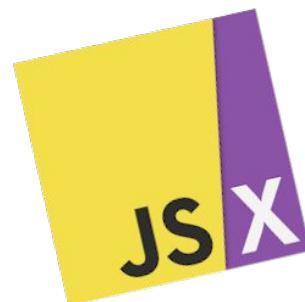
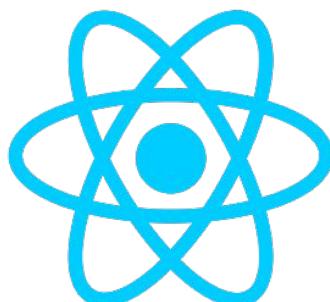
Como o React funciona?

SPA (Single Page Application)

- Melhor experiência do usuário
- Performance
- Responsabilidade do client-side
- Facilidade de manutenção

Como o React funciona?

- JSX - Linguagem permite escrever o HTML dentro do Javascript.
- Babel - converte o código para uma versão da linguagem que o processador entenda. No React, converte o JSX em JavaScript através da transpilação.
- Webpack:
 - Utiliza todo o código gerado pelo babel e cria bundle(um pacote) com um único código para o browser. Para cada tipo de arquivo, o código é convertido de uma maneira diferente.
 - Loaders ensinam a fazer as importações de arquivos de diferentes fontes (CSS, imagens, etc...)
 - Live Reload do Webpack Server



webpack

Gerenciador de pacotes

- npm: Gerenciador de pacotes padrão do Node.js
- yarn: Gerenciador de pacotes criado pelo Facebook
- Repositório onde ficam armazenados os pacotes
- Cliente que permite o envio / download de código do repositório.



yarn



Como o React funciona?

Instalação do yarn (OPCIONAL)

- Método 1: Através do instalador contido neste [link](#).
- Método 2: Através do Chocolatey. Para isso, primeiro é necessário instalar o Chocolatey na sua máquina. Você pode fazer a instalação do Chocolatey através das instruções contidas neste [link](#). Depois, basta executar o comando.

`choco install yarn`



Estrutura Inicial do React

Com o auxílio do npx, package runner do npm, iremos executar um comando que nos propiciará a configuração inicial de uma aplicação react. No terminal, vamos executar o comando **npx create-react-app nome-do-app**.

Instalação dos pacotes e configurações iniciais de um projeto react

```
C:\Users\mathe\OneDrive\Documentos\Aulas SerraTec_2022\Matheus_Material\Exemplo-Slide>npx create-react-app estrutura-inicial
Need to install the following packages:
  create-react-app
Ok to proceed? (y) y
npm WARN deprecated tar@2.2.2: This version of tar is no longer supported, and will not receive security updates. Please upgrade asap.
Creating a new React app in C:\Users\mathe\OneDrive\Documentos\Aulas SerraTec_2022\Matheus_Material\Exemplo-Slide\estrutura-inicial.
Installing packages. This might take a couple of minutes.
Installing react, react-dom, and react-scripts with cra-template...

added 1380 packages in 2m
190 packages are looking for funding
  run 'npm fund' for details
Initialized a git repository.

Installing template dependencies using npm...
npm WARN deprecated source-map-resolve@0.6.0: See https://github.com/lydell/source-map-resolve#deprecated
added 39 packages in 8s

190 packages are looking for funding
  run 'npm fund' for details
Removing template package using npm...

removed 1 package, and audited 1419 packages in 7s
190 packages are looking for funding
  run 'npm fund' for details
6 high severity vulnerabilities

To address all issues (including breaking changes), run:
  npm audit fix --force
Run 'npm audit' for details.

Created git commit.

Success! Created estrutura-inicial at C:\Users\mathe\OneDrive\Documentos\Aulas SerraTec_2022\Matheus_Material\Exemplo-Slide\estrutura-inicial
Inside that directory, you can run several commands:

  npm start
    Starts the development server.

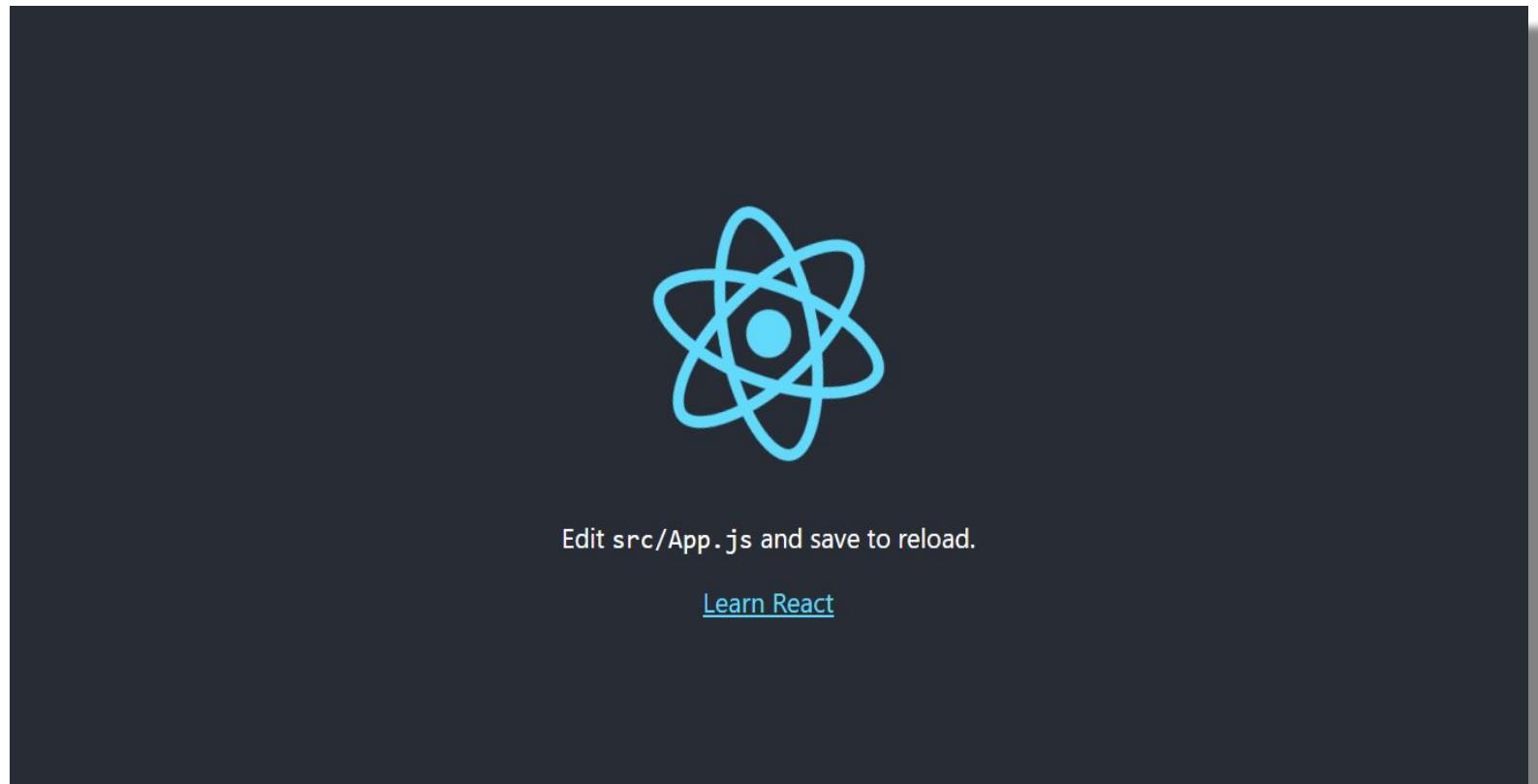
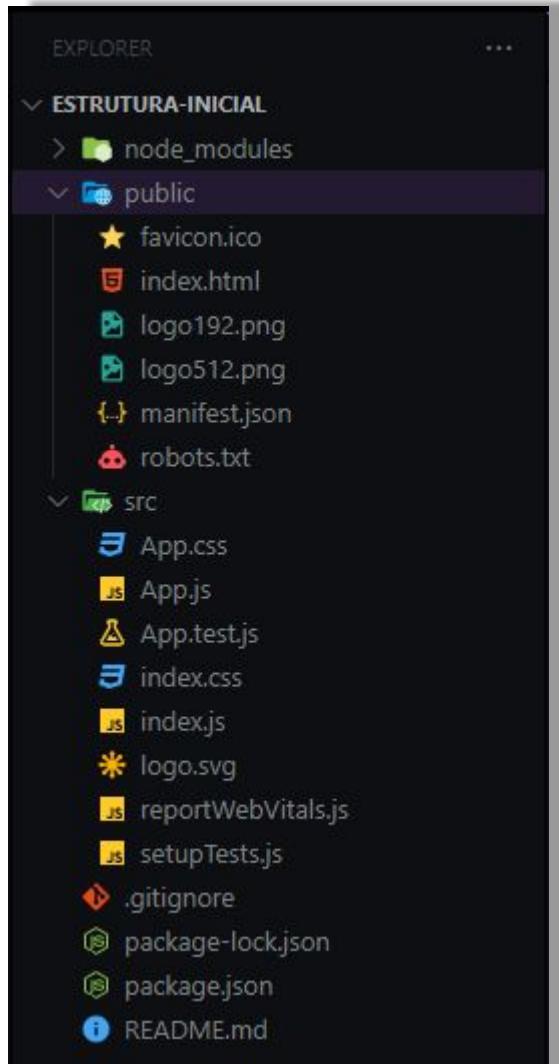
  npm run build
    Bundles the app into static files for production.

  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 estrutura-inicial
```

Estrutura Inicial do React



Estrutura Inicial do React

Após a criação do projeto react e de conhecermos sua estrutura inicial e como ele trabalha podemos startar nosso projeto e ver o resultado gerado no navegador deste projeto inicial.

Para isso utilizamos o comando `npm start` ou `yarn start` no cmd dentro do diretório do projeto.

```
C:\Users\mathe\OneDrive\Documentos\Aulas SerraTec_2022\Matheus_Material\Exemplo-Slide\estrutura-inicial>npm start
> estrutura-inicial@0.1.0 start
> react-scripts start

(node:10196) [DEP_WEBPACK_DEV_SERVER_ON_AFTER_SETUP_MIDDLEWARE] DeprecationWarning: 'onAfterSetupMiddleware' option is deprecated. Please use the 'setupMiddlewares' option.
(Use `node --trace-deprecation ...` to show where the warning was created)
(node:10196) [DEP_WEBPACK_DEV_SERVER_ON_BEFORE_SETUP_MIDDLEWARE] DeprecationWarning: 'onBeforeSetupMiddleware' option is deprecated. Please use the 'setupMiddlewares' option.
Starting the development server...
Compiled successfully!

You can now view estrutura-inicial in the browser.

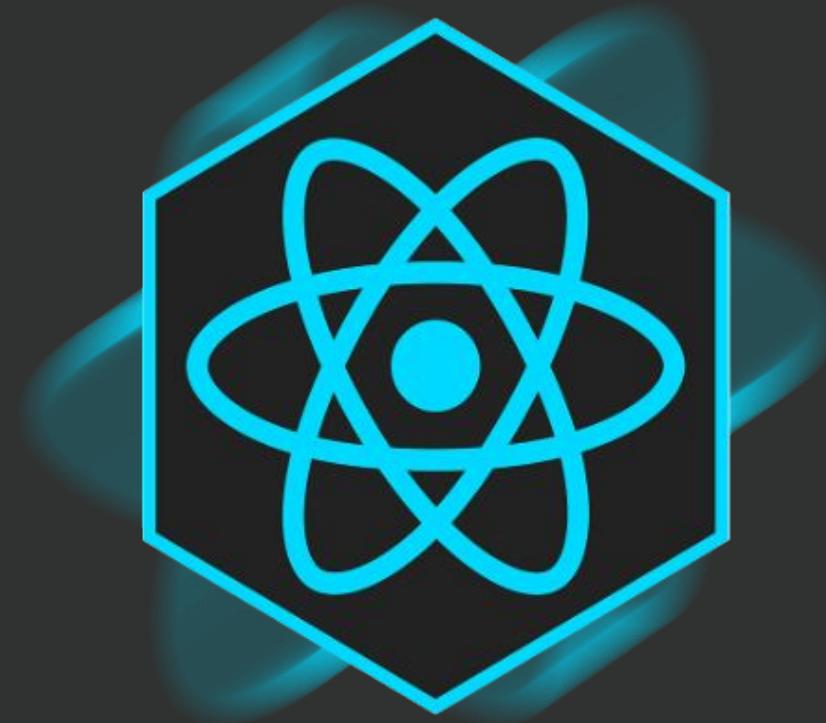
  http://localhost:3000

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

webpack compiled successfully
```

06

HOOKS



REACT JS

Componentes

O que está acontecendo

Futebol - Ontem
Brasil goleia Coreia do Sul por 5 a 1 em preparação para a Copa do Mundo



Assuntos do Momento: [Coreia](#), [Neymar](#)

#TheBoys 
Nem todo Super é herói.
 Promovido por Prime Video Brasil

Assunto do Momento em Rio de Janeiro
Bianca
17,6 mil Tweets

Assunto do Momento em Rio de Janeiro
Méier
2.442 Tweets

Guerra na Ucrânia - AO VIVO
Zelensky diz que Rússia controla 20% do território ucraniano; guerra no país completa 100 dias



[Mostrar mais](#)

Top 10 em filmes no Brasil hoje



Filmes para a família toda



Séries de humor besteirol





CANAIS SEGUIDOS

- YoDa V Rising
- Gaules Tropico 6
- tauseshi League of Legends
- Vellus Eventos especiais
- ZEmerson Teamfight Tactics
- TippyTippz Legends of Runet...
- Damianizando Diablo Immortal
- TNTsportBr 10 Novas Vídeos

Mostrar mais

CANAIS RECOMENDADOS

- Smuy Legends of Runete...
- sacy VALORANT
- courtesy League of Legends
- Kenzy League of Legends
- BleidoMaul Bleido Immortal
- Laposa Diablo Immortal

Mostrar mais

AO VIVO

Canais ao vivo que achamos que vai gostar

- AO VIVO 52 mil espectadores
- AO VIVO 614 espectadores
- AO VIVO 3,7 mil espectadores
- AO VIVO 104 espectadores

Mostrar mais

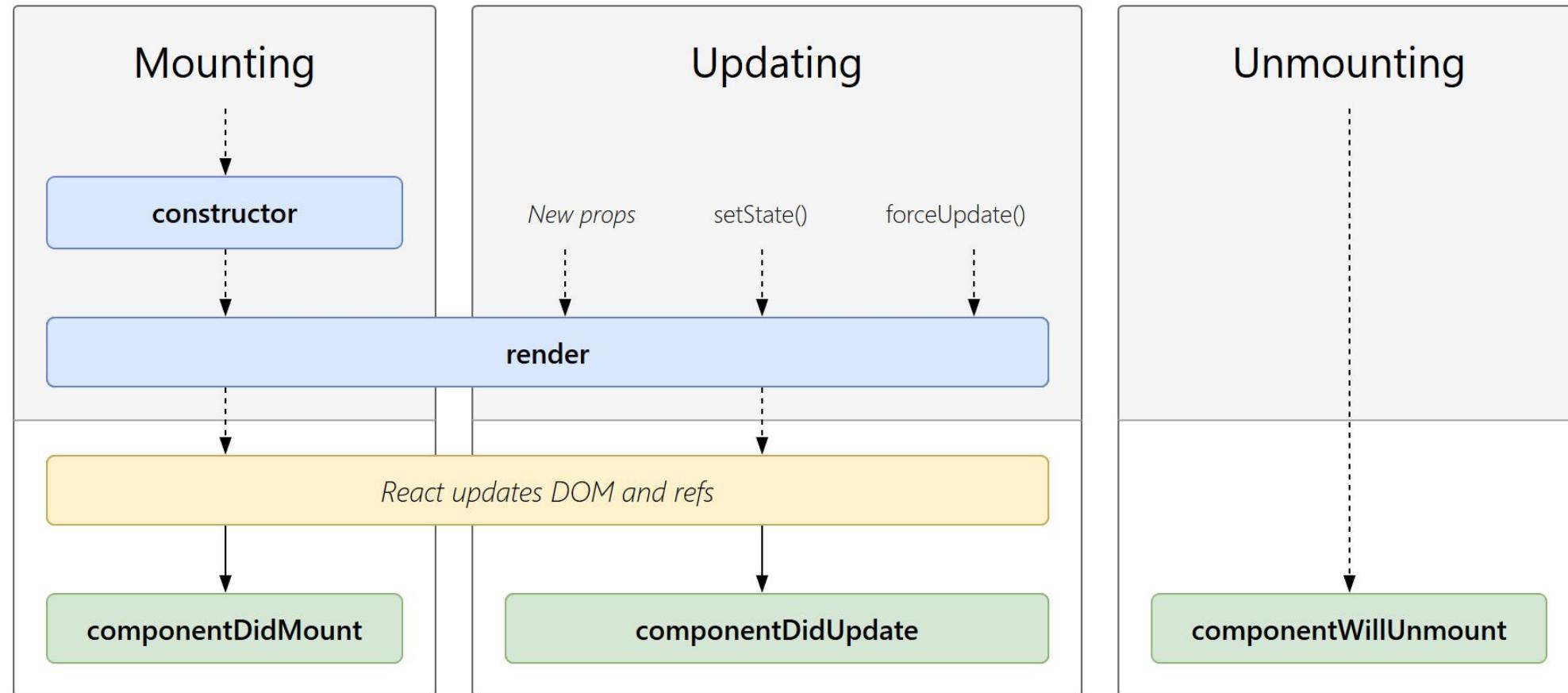
Categorias que achamos que vai gostar



Métodos de Ciclo de Vida (Lifecycle Methods)

- Cada componente do React tem um ciclo de vida que você pode monitorar e manipular durante suas três fases principais.
- As três fases são: Montagem, Atualização e Desmontagem.
- Montagem: inserção de elementos no DOM.
- Atualização: Como o nome já diz, acontece quando um componente é atualizado (mudança no state ou props do componente).
- Desmontagem: remoção de elementos no DOM

Métodos de Ciclo de Vida (Lifecycle Methods)



HOOKS

- Os componentes de função, no React, se parecem com isto:

```
const App = () =>{  
  return (  
    <h1> Hello World</h1>  
  );  
}
```

ou isto:

```
function App() {  
  return (  
    <h1> Hello World</h1>  
  );  
}
```

- Antigamente era conhecido estes exemplos como “componentes sem estado”. Porém agora foi introduzindo a habilidade de utilizar o state do React neles, portanto preferimos o nome “componentes de função”.
- Hooks não funcionam dentro de classes. Mas você pode usá-los em vez de escrever classes.

O que é um Hook?

- Um Hook é uma função especial que te permite utilizar recursos do React. Por exemplo, useState é um Hook que te permite adicionar o state do React a um componente de função. Vamos aprender outros Hooks mais tarde.

Quando eu deveria usar um Hook?

- Se você escreve um componente de função e percebe que precisa adicionar algum state para ele, anteriormente você tinha que convertê-lo para uma classe. Agora você pode usar um Hook dentro de um componente de função existente. Vamos fazer isso agora mesmo!

Usando o State do Hook

Aplicação utilizando useState

```
import React, {useState} from "react";

export const App = () =>{

  const [nome, setNome] = useState( )

  const handleClick = () =>{
    if(nome === 'Fulano'){
      setNome('')
    }else{
      setNome('Funalo')
    }
  }

  return(
    <>
      <div> Hello {nome}</div>
      <button onClick={handleClick}>Clique aqui</button>
    </>
  )
}
```

O que passamos para o useState como argumento?

- O único argumento para o Hook useState() é o state inicial. Diferente de classes, o state não tem que ser um objeto.
- Podemos manter um número ou uma string se for tudo que precisamos. No nosso exemplo, apenas queremos um número para quantas vezes o usuário clicou, então passamos 0 como state inicial para nossa variável. (Se quiséssemos guardar dois valores diferentes no state, chamaríamos useState() duas vezes.)

O que o useState faz?

- Ele declara um variável state. Nossa variável é chamada de count mas poderíamos chamar de qualquer coisa, como banana. Esta é uma maneira de “preservar” alguns valores entre as chamadas de funções — useState é uma nova maneira de usar as mesmas capacidades que o this.state tem em uma classe. Normalmente, variáveis “desaparecem” quando a função sai mas variáveis de state são preservadas pelo React.

MODO ANTIGO

```
class Example extends React.Component {  
  constructor(props) {  
    super(props);  
    this.state = {  
      count: 0  
    };  
  }  
}
```

MODO NOVO

```
import React, { useState } from 'react';  
  
function Example() {  
  // Declarar uma nova variável de state, na qual chamaremos de "count"  
  const [count, setCount] = useState(0);  
}
```

Usando Effect Hook

Aplicação utilizando useEffect

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

function Exemplo() {
  const [count, setCount] = useState(0);

  // Similar ao componentDidMount e componentDidUpdate:
  useEffect(() => {
    // Atualiza o título do documento usando a API do browser
    document.title = `Você clicou ${count} vezes`;
  });

  return (
    <div>
      <p>Você clicou {count} vezes</p>
      <button onClick={() => setCount(count + 1)}>
        Clique aqui
      </button>
    </div>
  );
}
```

O que o useEffect faz?

- Usando esse Hook, você diz ao React que o componente precisa fazer algo apenas depois da renderização. O React irá se lembrar da função que você passou (nos referiremos a ele como nosso “efeito”), e chamá-la depois que realizar as atualizações do DOM. Nesse efeito, mudamos o título do documento, mas podemos também realizar busca de dados ou chamar alguma API imperativa.

```
useEffect(() => {
  document.title = `Você clicou ${count} vezes`;
});
```

Por que useEffect é chamado dentro de um componente?

- Colocando useEffect dentro do componente nos permite acessar o state count (ou qualquer outra prop) direto do efeito. Nós não precisamos de uma API especial para lê-los — já está no escopo da função. Hooks adotam as closures do JavaScript e evitam APIs específicas do React onde o JavaScript já provê uma solução.

Comunicação direta

- Comunicação direta se refere quando temos a necessidade de passar informação do componente pai para o componente filho.
- Passamos essas informações através das **props**

```
import React from 'react';
import {ComponenteFilho} from './ComponenteFilho'

export const ComponentePai = () =>{

  return (
    <>
      <ComponenteFilho
        nome = "Matheus"
        idade = {23}
        nerd = {true}
      />
    </>
  );
}
```

```
import React from "react";

export function ComponenteFilho (props){

  return(
    <>
      <div>
        <span>{props.nome}</span>
      </div>
      <div>
        <span>{props.idade}</span>
      </div>
      <span>{props.nerd ? "Verdadeiro" : "Falso"}</span>
    </>
  )
}
```

Comunicação indireta

- A comunicação indireta ocorre quando temos a necessidade de passarmos informações do componente filho para o componente pai.
- Aplicação no componente Pai.

```
import React, { useState } from 'react';
import { IndiretoFilho } from './IndiretoFilho';

export const IndiretoPai = () =>{

  const [pai, setPai] = useState(
    {
      nome:'?',
      idade:0,
      nerd:false
    }
  )

  function informacaoDoPai (nome,idade,nerd){
    setPai({nome:nome, idade:idade, nerd:nerd})
  }

  return (
    <>
      <div>
        <span>{pai.nome}</span>
      </div>
      <div>
        <span>{pai.idade}</span>
      </div>
      <div>
        <span>{pai.nerd ? "Verdairo":"Falso"}</span>
      </div>
      <IndiretoFilho funcaoInformacoes={informacaoDoPai}/>
    </>
  );
}
```

- Aplicação no componente Filho.

```
import React from "react";

export function IndiretoFilho(props){
    return(
        <>
            <h3>Filho</h3>
            <button onClick={() => props.funcaoInformacoes('Carlos',72, true)}>Enviar informações</button>
        </>
    )
}
```

Comunicação controlado

- Um componente controlado se dá quando um state está atrelado ao input, fazendo com que o resultado apresentado em tela seja reflexo da evolução de seu estado React. Fazendo o estado React ser a “única fonte da verdade”

```
import React, { useState } from "react";

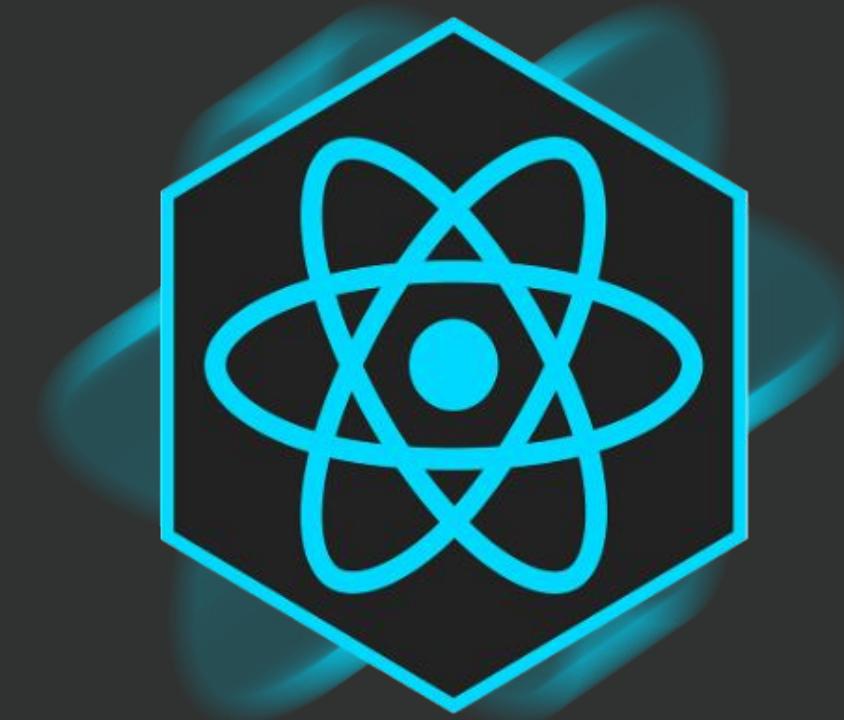
export function Input(){
    const[nome,setName] = useState("Inicial")

    function handleChange(e){
        setName(e.target.value)
    }

    return(
        <>
            <label>Nome: </label>
            <input type="text" value={nome} onChange={(e)=> handleChange(e)}/>
        </>
    )
}
```

07

NAVEGAÇÃO



REACT JS

Proposta da aula

- Construção aplicativos web com react
 - Renderização condicional
 - React Router
- Estilizando Componentes
 - Estilização Inline
 - Styled Components

Renderização Condicional

- Pode-se criar componentes distintos
- Renderizar apenas alguns dos elementos
- Tudo isso conforme o estado da sua aplicação
- Funciona da mesma forma que condições em Javascript
- Usando operadores como if para atualizar a UI desejada

Considere estes dois componentes:

```
function SaudacaoUsuario(){
    return (<h1> Bem-vindo de volta!</h1>)
}
```

```
function SaudacaoVisitante(){
    return (<h1> Por favor, registre-se.</h1>)
}
```

Aplicando condição ternária

```
import React, { useState } from "react";

export function RenderizacaoCondisional () {
    const [isLogado, setIsLogado] =useState(false)

    function SaudacaoUsuario(){
        return (<h1> Bem-vindo de volta!</h1>)
    }

    function SaudacaoVisitante(){
        return (<h1> Por favo, registre-se.</h1>)
    }

    function logar(){
        setIsLogado(true)
    }

    return(
        <>
        {isLogado ? <SaudacaoUsuario/> : <SaudacaoVisitante/>}
        <button onClick={() => logar()}>{isLogado ? 'Logoff' : 'Login'}</button>
        </>
    )
}
```

Aplicação com função

```
import React, { useState } from "react";

✓ export function RenderizacaoCondisional () {
    const [isLogado, setIsLogado] =useState(false)

    ✓ function SaudacaoUsuario(){
        return (<h1> Bem-vindo de volta!</h1>)
    }

    ✓ function SaudacaoVisitante(){
        return (<h1> Por favo, registre-se.</h1>)
    }

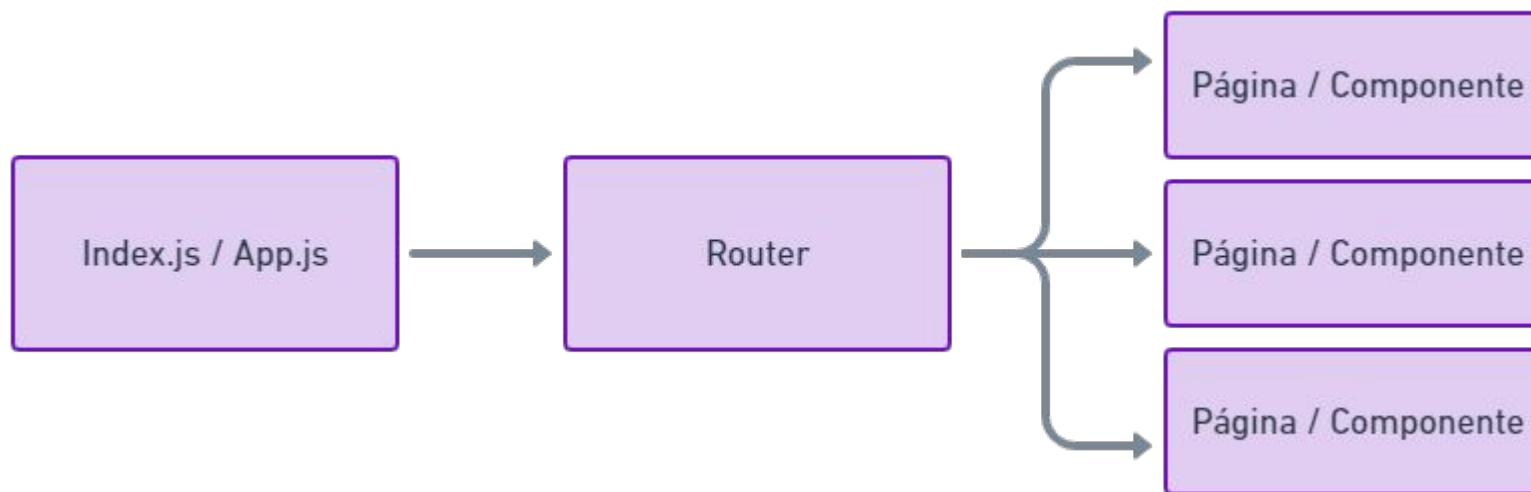
    ✓ function logar(){
        setIsLogado(true)
    }

    ✓ function validarLoginLogoff(props){
        ✓ if(props){
            return <SaudacaoUsuario/>
        }
        return <SaudacaoVisitante/>
    }

    return(
        <>
        {validarLoginLogoff(isLogado)}
        <button onClick={() => logar()}>{isLogado ? 'Logoff' : 'Login'}</button>
        </>
    )
}
```

React Router

O React Router permite criarmos rotas para navegação na aplicação permitindo carregar uma Página / Componente específico.



Como funciona?

Definimos as rotas baseada nas URLs da aplicação.

No navegador a aplicação React utiliza uma rota padrão <http://localhost:3000> para acessar a página principal.

Ao criar uma nova página / componente podemos definir que a rota para a exibição deste componente será por exemplo: <http://localhost:3000/sobre>.

Baseado na rota da URL o React router irá renderizar o componente respectivo a rota estabelecida.

Podemos configurar as Rotas em qualquer lugar da nossa aplicação, porém é aconselhável a configuração nos arquivos Index.js, App.js ou em um componente específico para controlar as Rotas.

Para configurar as rotas da aplicação utilizaremos um pacote específico para esse tipo de implementação que é o React-Router-Dom.

Comando de instalação:

```
npm install react-router-dom@6
```

Documentação: <https://reactrouter.com/docs/en/v6/getting-started/overview>

React Router

Agora que já temos o pacote instalado vamos configurar as nossas rotas.

Primeira etapa é importar o pacote BrowserRouter.

Depois podemos englobar todas as páginas que terão rotas estabelecidas.

funcionou?

```
import React from "react";
import {
  BrowserRouter,
} from "react-router-dom";

import {Home} from '../pages/Home'
import {QuemSomos} from '../pages/QuemSomos'

export function App(){
  return(
    <BrowserRouter>
      <Home/>
      <QuemSomos/>
    </BrowserRouter>
  )
}
```

React Router

Precisamos importar mais dois pacotes do react-router-dom [Routes](#) e [Route](#).

[Routes](#), receberá vários componentes [Route](#) e dado o caminho que for passado na URL um deles será renderizado.

```
import React from "react";
import {
  BrowserRouter,
  Routes,
  Route
} from "react-router-dom";

import {Home} from '../pages/Home'
import {QuemSomos} from '../pages/QuemSomos'

export function App(){
  return(
    <BrowserRouter>
      <Routes>
        <Route path="/" element={<Home />}/>
        <Route path="/quemsomos" element={<QuemSomos />}/>
      </Routes>
    </BrowserRouter>
  )
}
```

- O path="/" combinado com o atributo exact garantem que, se a rota tiver somente "/" ele vai ser renderizado.
- Isso evita conflitos com as outras rotas que possuam o valor passado no path.
- Podemos ainda passar uma rota default para páginas não encontradas após a última rota.
- Uma rota que representa a página 404 do nosso sistema.
 - <Route path='*' component={ComponenteDePagina404} />
- Agora é possível acessar: <http://localhost:3000/QuemSomos> e visualizar nosso componente.

Exemplo de aplicação página 404 Not Found

```
import React from "react";

import {
  BrowserRouter,
  Routes,
  Route
} from "react-router-dom";

import {Home} from '../pages/Home'
import {QuemSomos} from '../pages/QuemSomos'
import {NotFound} from '../pages/PaginaNotFound'

export function App(){
  return(
    <BrowserRouter>
      <Routes>
        <Route path="/" element={<Home />}/>
        <Route path="/quemsomos" element={<QuemSomos />}/>
        <Route path="*" element={<NotFound />}/>
      </Routes>
    </BrowserRouter>
  )
}
```

- Podemos navegar entre páginas sem precisar dar refresh
- Para isso utilizaremos outro componente do React Router Dom, [Link](#)
 - import { Link } from 'react-router-dom'
- E usar dentro do JSX o componente dessa forma:
 - <Link to="/quemsomos">Ir para a página Quem Somos</Link>

```
import React from "react";
import { Link } from "react-router-dom";

export const Home = () =>{

    return(
        <>
            <h1>HOME</h1>
            <Link to="/quemsomos">Ir para pagina quem somos</Link>
        </>
    )
}
```

React Router

- Outra forma de navegar é utilizar um **Hook** do react router dom chamado `useNavigate`

```
import React from "react";
import { useParams, useNavigate } from "react-router-dom";

export const QuemSomos = () =>{

  let { nome } = useParams();
  let navigate = useNavigate();

  function handleClick(){
    navigate("/");
  }

  return(
    <>
      <h3>Usuário logado: {nome}</h3>
      <h1>Quem Somos?</h1>
      <h2> Aqui você irá descobrir quem somos e o que fazemos</h2>
      <button onClick={handleClick}>Voltar para home</button>
    </>
  )
}
```

React Router

- Podemos passar parâmetros nas rotas enviando informações de uma página para outra.
- Precisamos informar na rota que será passado um parâmetro.

```
import React from "react";

import {
  BrowserRouter,
  Routes,
  Route
} from "react-router-dom";

import {Home} from '../Pages/Home/index'
import {QuemSomos} from '../Pages/QuemSomos/index'
import {NotFound} from '../Pages/PaginaNotFound/index'

export function Root (){
  return(
    <BrowserRouter>
      <Routes path>
        <Route path="/" element={<Home />}/>
        <Route path="/quemsomos/:nome" element={<QuemSomos />} />
        <Route path="*" element={<NotFound />} />
      </Routes>
    </BrowserRouter>
  )
}
```

React Router

- Na página Home baseado em nosso exemplo informamos o parâmetro a ser passado para a página Quem somos.

```
import React, { useState } from "react";
import { Link } from "react-router-dom";

export const Home = () =>{

  const [usuario] = useState({
    nome:"Matheus",
    idade: 23
  })

  return(
    <>
      <h1>HOME</h1>
      <Link to={`/quemsomos/${usuario.nome}`}>Ir para pagina quem somos</Link>
    </>
  )
}
```

- Na página Quem Somos precisamos recuperar o parâmetro que está vindo na URL para exibirmos em tela ou manipular o valor.
- Utilizamos um **Hook** que importamos do react-router-dom o **useParams**
- Conseguimos obter a variável passada através da desestruturação.

```
import React from "react";
import { useParams, useNavigate } from "react-router-dom";

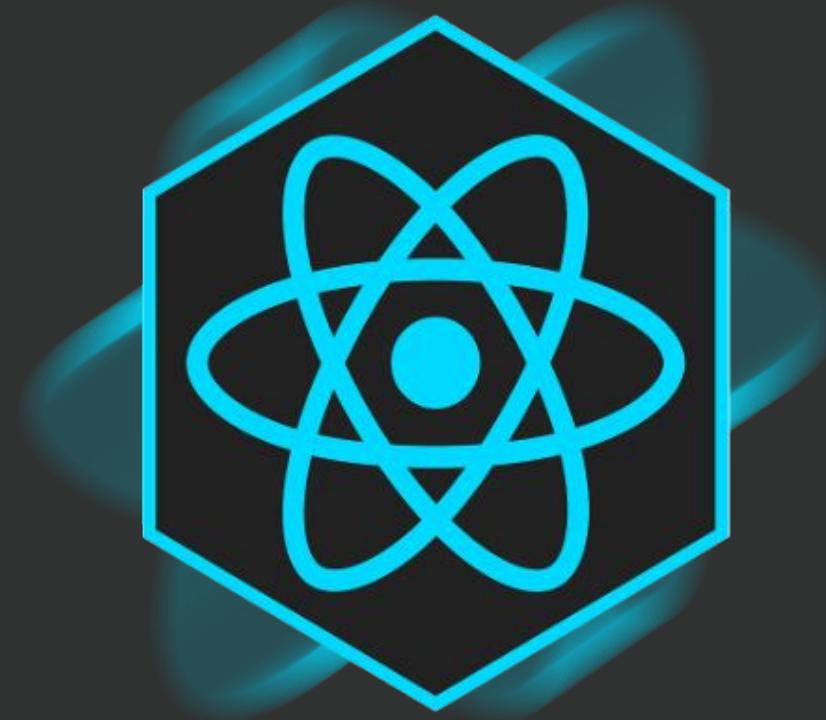
export const QuemSomos = () =>{

  let { nome } = useParams();
  let navigate = useNavigate();

  function handleClick(){
    navigate("/");
  }

  return(
    <>
      <h3>Usuário logado: {nome}</h3>
      <h1>Quem Somos?</h1>
      <h2> Aqui você irá descobrir quem somos e o que fazemos</h2>
      <button onClick={handleClick}>Voltar para home</button>
    </>
  )
}
```

Estilização



REACT JS

Estilização Inline

- A propriedade style é usada para adicionar estilos em tempo de renderização dinamicamente.

```
<h3 style={{background: '#C793E1', color:'#F9EFFF', textAlign: 'center'}}>Usuário logado: {nome}</h3>
<h1>Quem Somos?</h1>
```

```
const styleInline ={
  background: '#C793E1',
  color:'#F9EFFF',
  textAlign: 'center'
}

return(
  <>
    <h3 style={styleInline}>Usuário logado: {nome}</h3>
    <h1>Quem Somos?</h1>
    <h2> Aqui você irá descobrir quem somos e o que fazemos</h2>
    <button onClick={handleClick}>Voltar para home</button>
  </>
)
```

Estilização CSS Externo

- Uma outra possibilidade é referenciar uma folha de estilo externa CSS.
- Estamos trabalhando com JSX, estamos desenvolvendo tudo no mundo Javascript e por esse motivo ao declarar uma classe em algum elemento existe uma pequena diferença na nomenclatura. `class` é uma palavra reservada do Javascript então quando quisermos referenciar uma classe de estilos utilizamos a nomenclatura `className`.

```
import "./style.css";

function QuemSomos() {
  const { nome } = useParams();

  return (
    <>
      <h1 className="usuario-logado">Usuário logado: {nome}</h1>
      <h1>Quem Somos</h1>
    </>
  );
}
```

Styled Components

- Precisaremos instalar outro pacote do React
 - npm install styled-components --save

```
import styled from 'styled-components'

export const Button = styled.button`
  cursor: pointer;
  background: transparent;
  font-size: 1rem;
  color: #C793E1;
  border: 2px solid #C793E1;
  border-radius: 0.4rem;
  margin: 0 1rem;
  padding: 0.25rem 1rem;
  transition: 0.5s all ease-out;
  &:hover{
    background-color: #C793E1;
    color: #F9EFFF;
  }
`
```

Styled Components

- Podemos criar componentes de estilo substituindo os elementos HTML por componentes que já carregam o próprio estilo.
- Com styled-components é possível receber **props** de estilo e alterar algum estilo de acordo com as **props**.

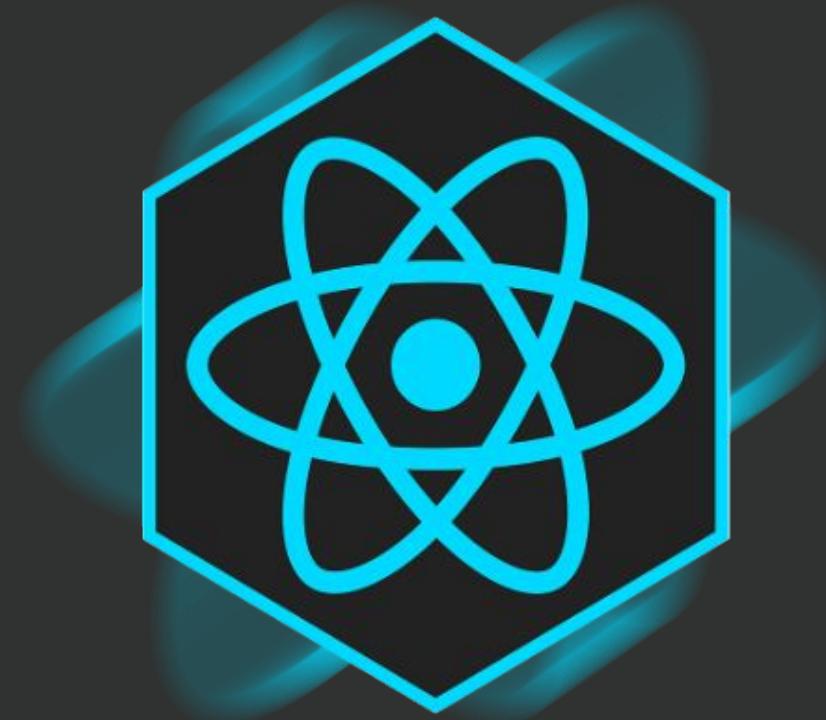
```
<Link to={`/quemsomos/${usuario.nome}`}>
  <Button cor='red'>Ir para pagina quem somos</Button>
</Link>
```

```
import styled from 'styled-components'

export const Button = styled.button`
  cursor: pointer;
  background: transparent;
  font-size: 1rem;
  color: ${props => props.cor};
  border: 2px solid #C793E1;
  border-radius: 0.4rem;
  margin: 0 1rem;
  padding: 0.25rem 1rem;
  transition: 0.5s all ease-out;
  &:hover{
    background-color: #C793E1;
    color: #F9EFFF;
  }
`
```

09

AXIOS



REACT JS

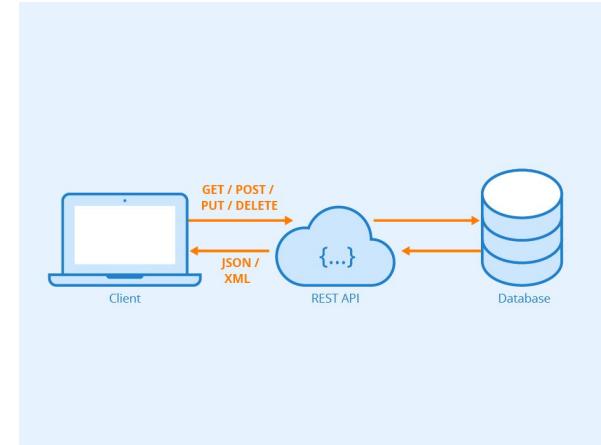
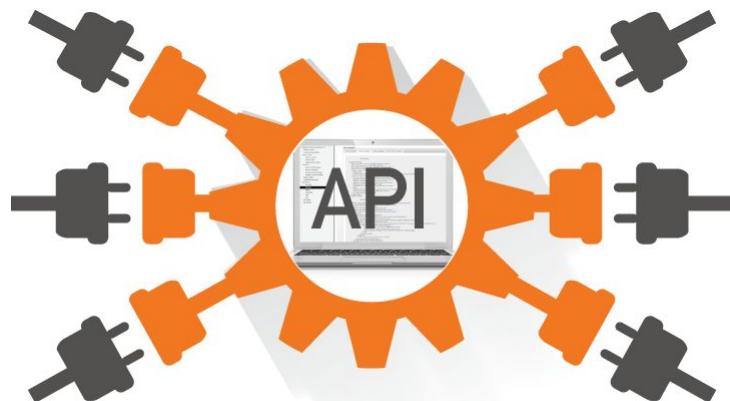
Proposta da aula

- Trabalhando com APIs (APIs Calls)
 - Relembrando o conceito de APIs Calls
- Funções assíncronas
 - Async Await e Then
- Axios
 - Pacote de instalação
 - Configuração
 - Consumo de APIs com axios



Relembrando o conceito de API Calls

É possível fazer requisições por meio de API. Isso significa que você pode enviar ou receber informações para/de outro aplicativo, e os dados ou atributos capturados podem ser usados para exibir conteúdos, integrar aplicações ou sincronizar dados entre bancos de dados para enriquecer informações.



- Funções assíncronas: São funções que não acontecem ao mesmo passo do tempo de execução. Quando uma função assíncrona é chamada, ela retorna uma **Promise**, ou seja, uma promessa de retorno;
- Se eu declarar uma variável que receberá uma resposta de uma função assíncrona e logo em seguida tentar acessá-la, ela poderá não conter este valor neste momento, o que pode trazer um comportamento indesejável a nossa aplicação.

- Para que aguardemos a resposta de uma função assíncrona, podemos utilizar métodos `Async Await` ou `Then()`.
- Desta forma, poderemos trabalhar com a segurança de que já recebemos o valor da nossa Promise

```
const getData = async () => {
  const response = await axios.get("https://api.github.com/users/Abjoia")
  setUsuario(response.data)
}
```

```
const getData = async () => {
  axios
    .get("https://api.github.com/users/Abjoia")
    .then((response) => setUsuario(response.data))
}
```

- Com o Axios instalado, podemos setar as configurações iniciais para utilizá-lo em nossa aplicação. Para isso, dentro de src, vamos criar uma pasta chamada Services e um arquivo chamado api.js dentro dela.
- Dentro deste arquivo, vamos importar o axios e configurar nossa baseURL.

```
import axios from "axios";

export const api = axios.create({
  baseURL: 'https://api.github.com/
})
```

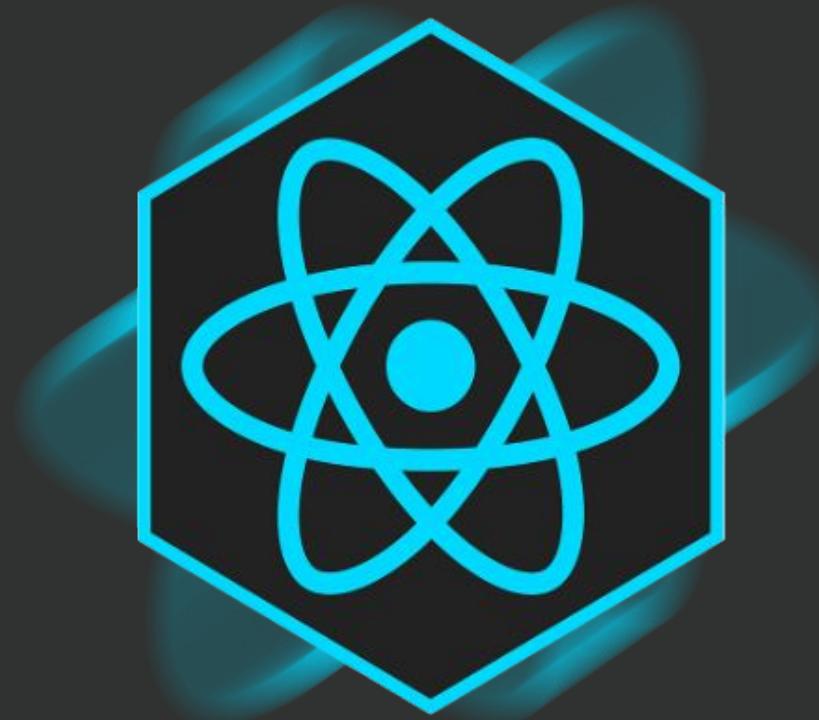
- Agora basta importarmos o arquivo api.js em qualquer lugar da nossa aplicação onde queremos fazer um requisição à API declarada na baseURL.
- Exemplo:

```
import { api } from "./Services/api";

export const App = () =>{

    const [usuario, setUsuario] = useState({login:"", name:""});
    useEffect(()=>{
        const user = 'Facebook'
        const getData = async () => {
            const response = await api.get(`users/${user}`);
            setUsuario(response.data)
        }
        getData()
    }, [])
}
```

Context API



REACT JS

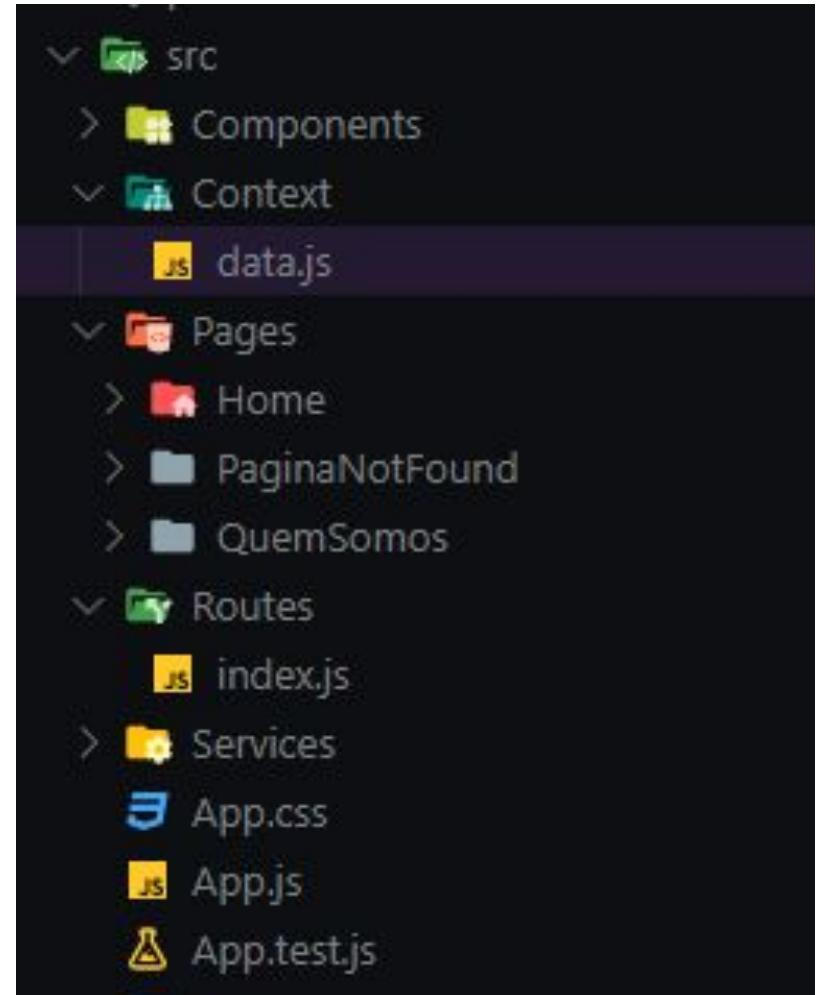
- Em uma aplicação típica do React, os dados são passados de cima para baixo (de pai para filho) via **props**, mas esse uso pode ser complicado para certos tipos de props (como preferências locais ou tema de UI), que são utilizadas por muitos componentes dentro da aplicação.

Contexto (context) fornece a forma de compartilhar dados como esses, entre todos componentes da mesma árvore de componentes, sem precisar passar explicitamente **props** entre cada nível.

- Contexto (context) é indicado para compartilhar dados que podem ser considerados “globais” para a árvore de componentes do React. Usuário autenticado ou o idioma preferido, são alguns casos comuns.
- Usando contexto, nós podemos evitar passar **props** através de elementos intermediários.

Então como criamos um context?

- Devemos criar um diretório dentro de src chamado context para manter a organização e dentro dela criamos o arquivo de contexto.
- Como será um arquivo de dados não teremos JSX, por esse motivo criamos com a extensão js



- Dentro de data.js definimos o estado do contexto que serão os dados expostos a nível global na aplicação
- Iniciamos o contexto através do `React.createContext()`

```
import React from "react";

export const estadoContexto ={
    nome:"Matheus"
}

export const DataContext = React.createContext(null)
```

- Agora precisamos englobar toda aplicação para que os dados sejam acessíveis pelos componentes.
- Os componentes mais alto nível que possuímos são App.js e index.js podemos aplicar em qualquer um deles.
- O DataContext contém uma propriedade Provider que precisamos utilizar passando o estadoContexto em value

```
import React from "react";
import { Root } from "./Routes";

import { DataContext, estadoContexto } from "./Context/data";

export const App = () =>{

  return(
    <DataContext.Provider value={estadoContexto}>
      <Root />
    </DataContext.Provider>
  )
}
```

- Agora basta importar o contexto no componente para acessar os dados globais.

```
import React, { useContext } from "react";
import { Link } from "react-router-dom";
import { DataContext } from "../../Context/data";
import { Button } from "../QuemSomos/style";

export const Home = () =>{

    const {nome} = useContext(DataContext)

    return(
        <>
            <h1>HOME</h1>
            <Link to={`/quemsomos/${nome}`}>
                <Button cor='red'>Ir para pagina quem somos</Button>
            </Link>
        </>
    )
}
```

CONTEXT API

- Podemos deixar a implementação do context mais interessante e adequada combinando com um componente. aumentando o poder da context API
- Com essa refatoração o contexto vai ficar assim
=>

```
import React, { useState } from "react";

export const estadoContexto ={
    nome:"Matheus"
}

export const DataContext = React.createContext(null)

export function Contexto(props){
    const[ nome, setNome] = useState(estadoContexto)

    function handleSetNome(e){
        setNome(e.target.value)
    }

    return(
        <DataContext.Provider value={{nome, handleSetNome}}>
            {props.children}
        </DataContext.Provider>
    )
}
```

CONTEXT API

- No App.js importamos o componente ao invés de importar o DataContext e o estadoContext
- O contexto engloba toda aplicação todos elementos filho de contexto são enviados por `props` automaticamente, que podem ser recuperados com `props.children` no componente contexto

```
import React from "react";
import { Root } from "./Routes";

import {Contexto} from "./Context/data";

export const App = () =>{

  return(
    <Contexto>
      <Root />
    </Contexto>
  )
}
```

CONTEXT API

- No componente contexto retornamos `DataContext.Provider` englobando todos os filhos de context, logo englobando toda a aplicação.
- Essa implementação permite uma organização maior no código e facilita na criação de estados.

```
export function Contexto(props){  
  const[ nome, setNome ] = useState(estadоГontexto)  
  
  function handleSetNome(e){  
    setNome(e.target.value)  
  }  
  
  return(  
    <DataContext.Provider value={{nome, handleSetNome}}>  
      {props.children}  
    </DataContext.Provider>  
  )  
}
```

CONTEXT API

- Os componentes continuam usando os dados do contexto da mesma forma

```
import React, { useContext } from "react";
import { Link } from "react-router-dom";
import { DataContext } from "../../Context/data";
import { Button } from "../QuemSomos/style";

export const Home = () =>{

  const {nome, handleSetName} = useContext(DataContext)

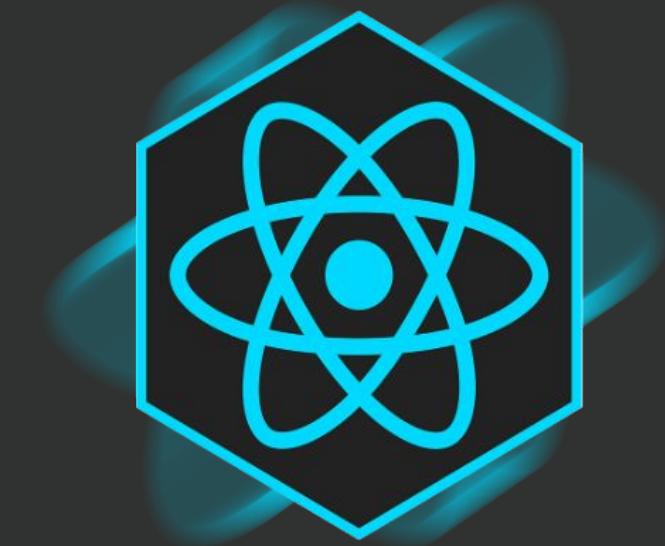
  return(
    <>
      <h1>HOME</h1>
      Nome:
      <input type="text" placeholder="Digite seu nome" value={nome || ''} onChange={(e) => handleSetName(e)} />
      <Link to={`/quemsomos/${nome}`}>
        <Button cor='red'>Ir para pagina quem somos</Button>
      </Link>
    </>
  )
}
```

```
let { nome } = useContext(DataContext);
let navigate = useNavigate();

function handleClick(){
  navigate("/");
}

return(
  <>
    <h3 style={styleInline}>Usuário logado: {nome}</h3>
    <h1>Quem Somos?</h1>
    <h2> Aqui você irá descobrir quem somos e o que fazemos</h2>
    <Button cor='#C793E1' onClick={handleClick}>Voltar para home</Button>
    {/* {getData()} */}
  </>
)
```

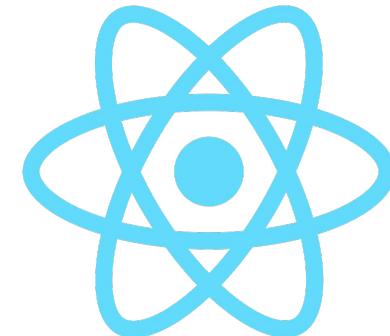
Deploy



REACT JS

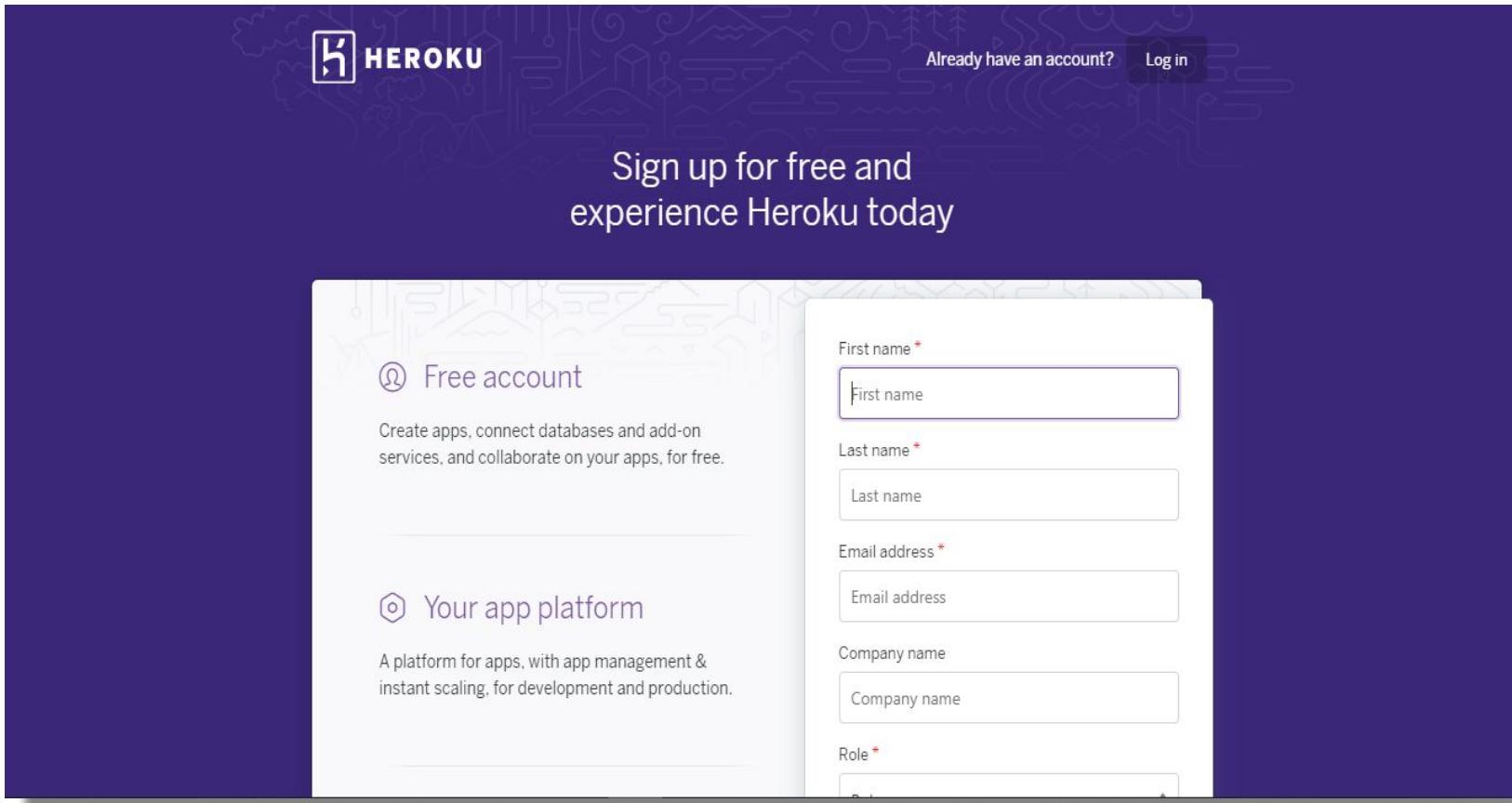
O que será necessário?

- Git
- Uma conta no Heroku
- CLI do Heroku



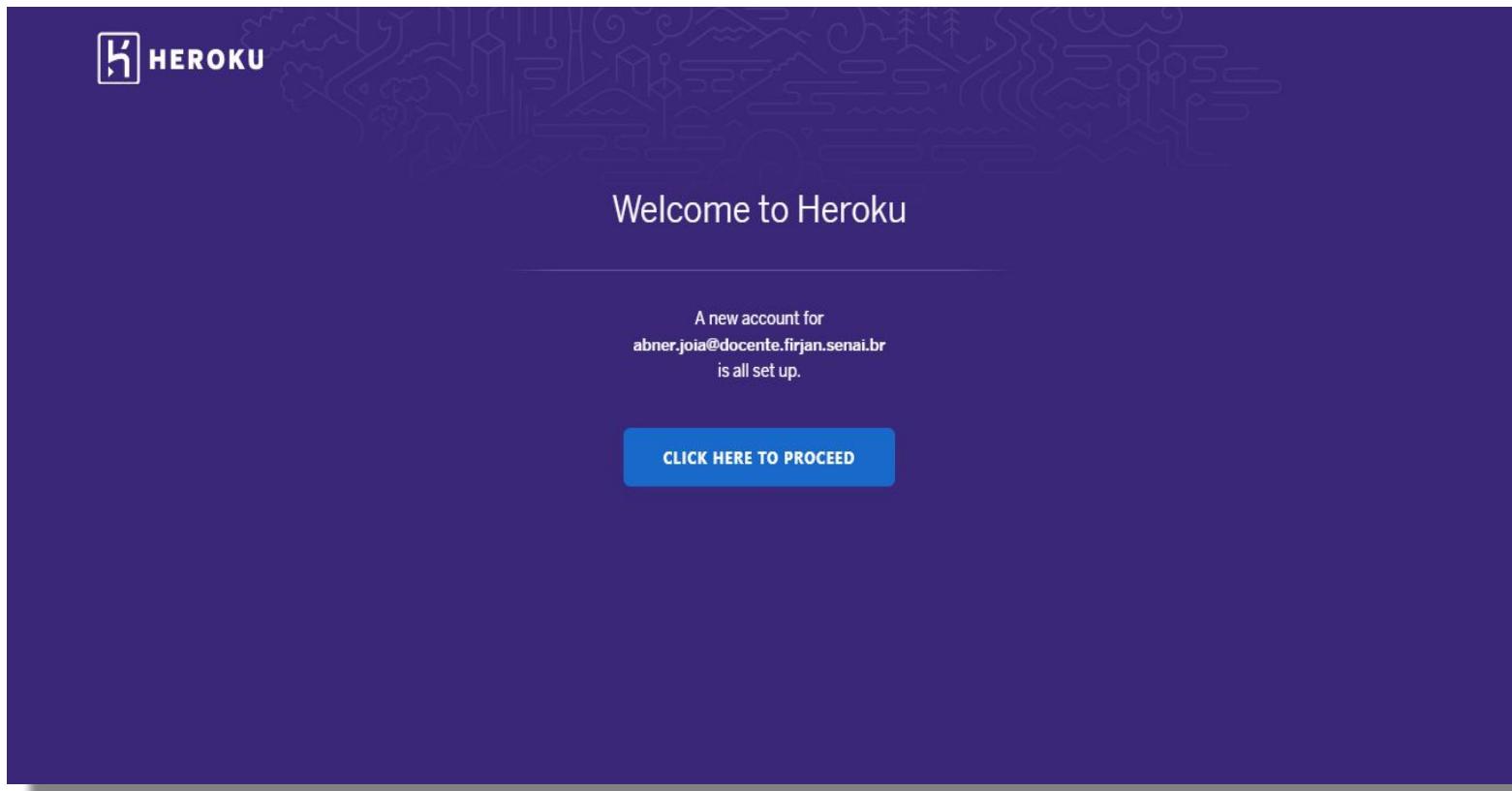
DEPLOY

- Criar conta no heroku <https://signup.heroku.com/>



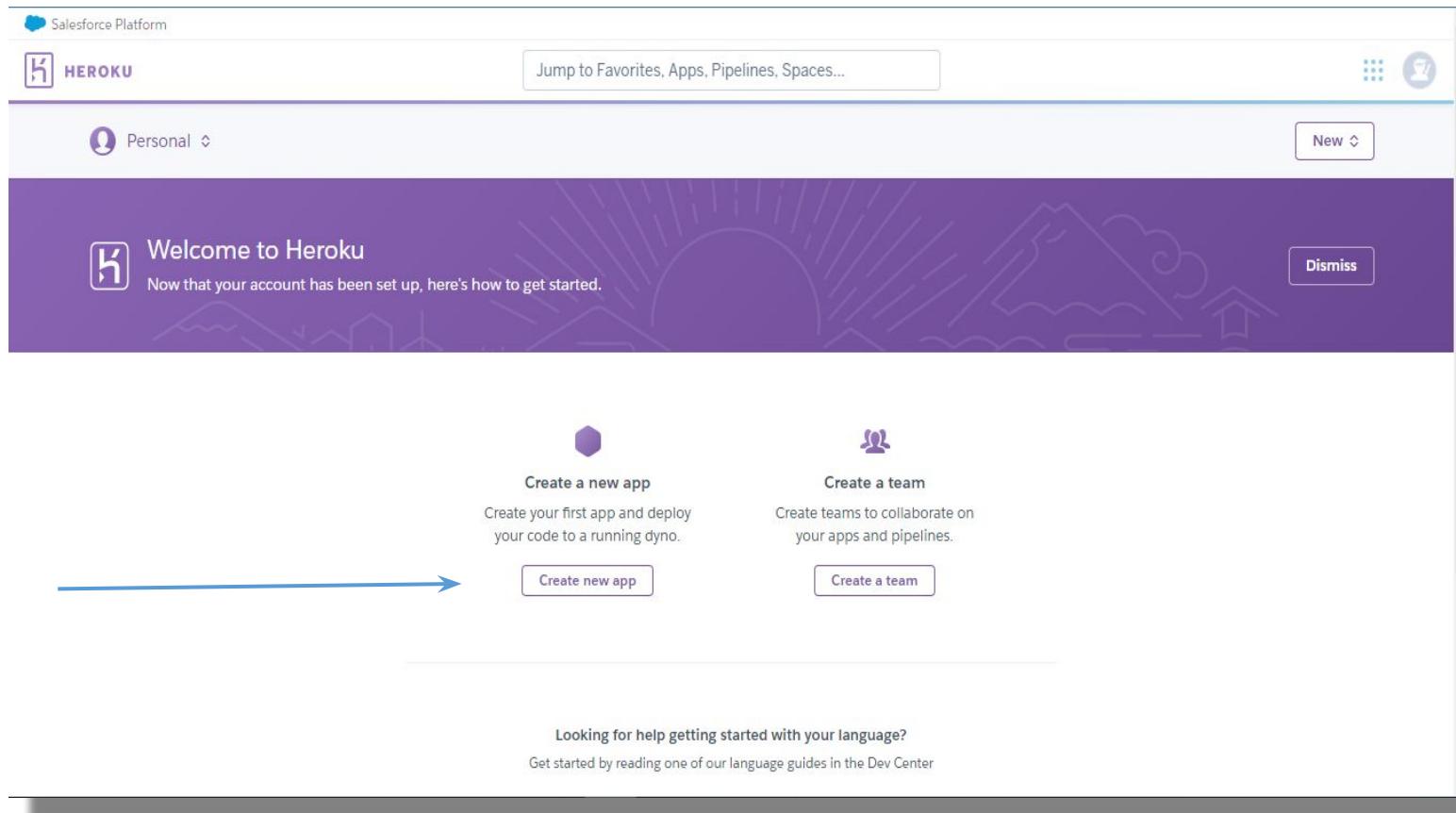
DEPLOY

- Depois de confirmar a conta pelo email e criar a senha você já terá uma conta ativada no heroku



DEPLOY

- Acessando sua conta vá até dashboard



DEPLOY

- Create new app

App name

meuprimeirositeshospedado



meuprimeirositeshospedado is available

Choose a region

United States

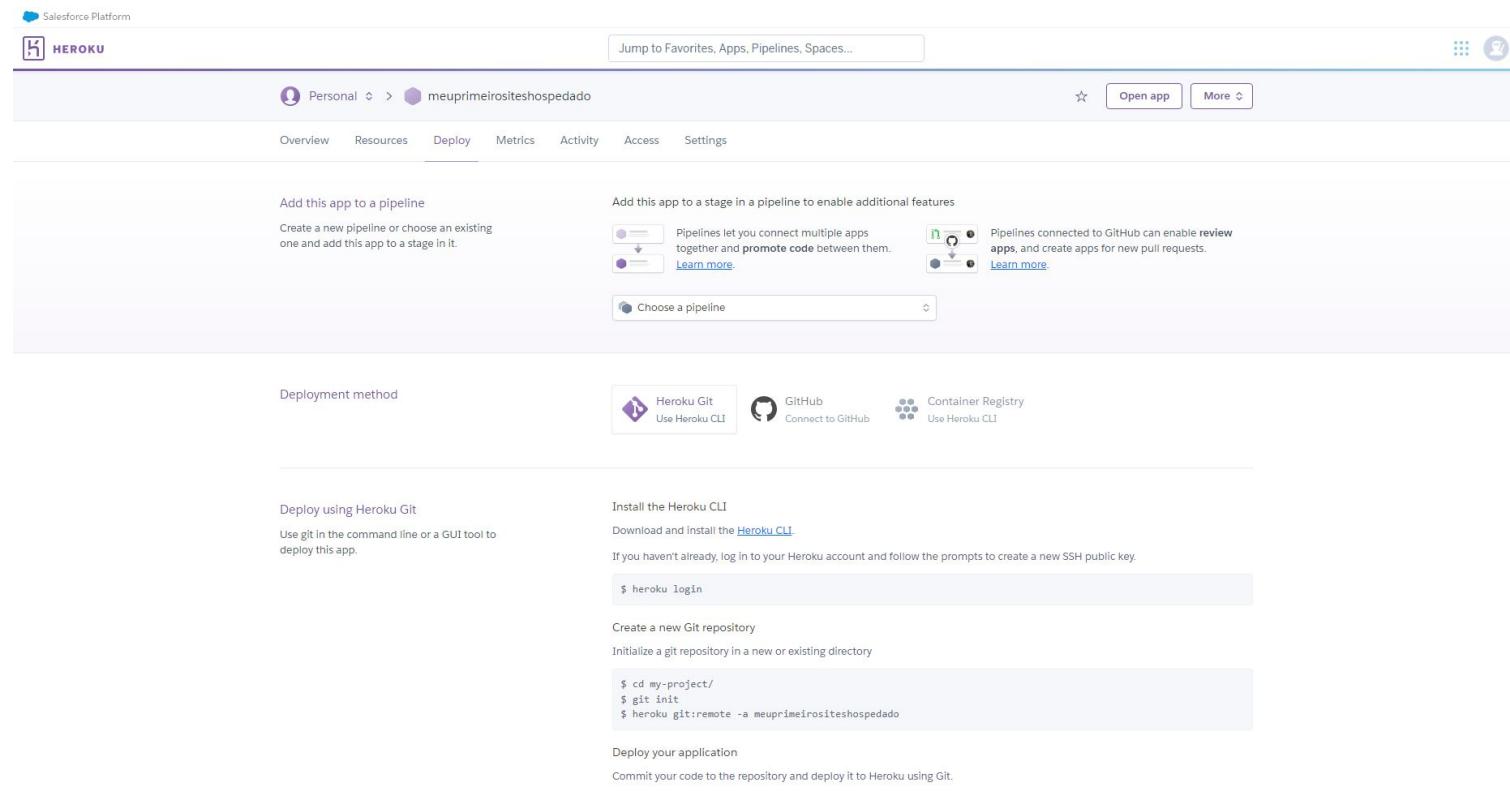


Add to pipeline...

Create app

DEPLOY

- Esta tela apresenta todas as informações de sua aplicação no Heroku
- Em Deploy temos alguns métodos e o passo a passo para realizar o Deploy da nossa aplicação.



DEPLOY

- O método que vamos utilizar é o Heroku Git
- Precisamos baixar Heroku CLI

Deploy using Heroku Git

Use git in the command line or a GUI tool to deploy this app.

Install the Heroku CLI

Download and install the [Heroku CLI](#).

If you haven't already, log in to your Heroku account and follow the prompts to create a new SSH public key.

```
$ heroku login
```

DEPLOY

- Depois da instalação do Heroku CLI seguimos com os próximos passos
- Todos os comandos sendo utilizados no GitBash

Create a new Git repository

Initialize a git repository in a new or existing directory

```
$ cd my-project/  
$ git init  
$ heroku git:remote -a meuprimeirositeshospedado
```

Deploy your application

Commit your code to the repository and deploy it to Heroku using Git.

```
$ git add .  
$ git commit -am "make it better"  
$ git push heroku master
```



You can now change your main deploy branch from "master" to "main" for both manual and automatic deploys, please follow the instructions [here](#).

Existing Git repository

For existing repositories, simply add the `heroku` remote

```
$ heroku git:remote -a meuprimeirositeshospedado
```