

React Native

React Native



- Início em 27/06/2023;
- Término em 20/07/2023;
- Total de 8 sessões;
- Horário: das 9h às 12h;
- Dias da semana: 3as e 5as;
- Conteúdo do curso:

Será enviado para o e-mail dos participantes

- Avaliação - Atividades ao longo do curso;

Bibliografia



1. React Native for Mobile Development - Akshat Paul, Abhishek Nalwaya. Apress, 2019.
2. Fullstack React Native - Devin Abbot, Houssein Djirdeh, Anthony Accomazzo, and Sophia. FullStack, 2017.
3. Learning React Native - Bonnie Eisenman. O'Reilly, 2016.
4. React and React Native – Adam Boduch, Roy Derks. Packt, 2020.
5. www.tutorialspoint.com
6. <https://reactnative.dev/> Site oficial do React Native
7. <https://www.javatpoint.com/react-native-tutorial>
8. <https://docs.expo.dev/versions/latest/> Site oficial do Expo SDK

Foco do Curso



- Capacitação para o desenvolvimento de apps móveis Android e iOS;
- Apresentando:
 - Fundamentos do React Native.
 - Componentes principais:
 - Layouts
 - Navegação entre telas.
 - Entradas do usuário.
 - Imagens.
 - Botões.
 - APIs (Componentes utilizam as APIs)
 - Tratando *end-points*.

Obs.: Vocês precisam trazer o celular para as aulas!!!

React Native

Motivação



- O Facebook criou o React Native para construir seus aplicativos móveis;
- A motivação para fazer isso se originou do fato de o React para a *web* ter dado certo;
- O React já é uma ferramenta consagrada para o desenvolvimento de interface do usuário;
- Se precisamos de um aplicativo nativo, por que lutar contra isso?
- Basta adequar o React para funcionar com os elementos nativos do Sistema Operacional dos dispositivos móveis;

React Native

Visão Geral



- Um app Android é escrito em Kotlin(linguagem oficial) ou Java;
- Um app iOS é escrito Swift ou Objective-C;
- No React Native, escreve-se JavaScript usando componentes React;
- Em tempo de execução, o React Native cria as visualizações Android e iOS correspondentes para esses componentes;
- Os componentes React Native suportam as mesmas visualizações do Android e iOS;
- Os apps React Native parecem e funcionam como qualquer outro app Android ou iOS;

React Native

Visão Geral



- Aprender mais de uma linguagem de programação para criar um aplicativo móvel é custoso;
- A solução é utilizar uma plataforma React apropriada para o novo destino da renderização;
- O React Native faz chamadas assíncronas para o sistema operacional móvel subjacente, que chama as APIs do *widget* nativo;
- Há um mecanismo JavaScript e a API do React é basicamente a mesma do React para a *web*;
- A diferença está no alvo, em vez de um DOM, há chamadas de API assíncronas;

React Native

Visão Geral



- A mesma biblioteca React usada na *web* é usada pelo React Native e executada pela JavaScriptCore;
- As mensagens que são enviadas para APIs da plataforma nativa são assíncronas;
- O React Native vem com componentes implementados para plataformas móveis, em vez de componentes que são elementos HTML;
- O React Native utiliza o JSX que é uma extensão da linguagem JavaScript;
- Em vez de uma equipe de UIs para web, uma equipe iOS e outra Android, há somente uma equipe de UI React;

React Native

Visão Geral



- Assim como no React, componentes constituem a essência do React Native;
- Permitem dividir a Interface do Usuário em partes **independentes e reutilizáveis**;

React Native

Navegadores móveis



- Os navegadores móveis carecem de muitos recursos dos aplicativos móveis;
- Os navegadores não podem replicar os mesmos *widgets* nativos, da plataforma móvel, com elementos HTML;
- Os *widgets* nativos da plataforma são consistentes com o restante da plataforma;
- As interações do usuário em dispositivos móveis são diferentes das interações em um projeto *web*;
- Aplicações *web* assumem a presença de um mouse;

React Native

Navegadores móveis



- As coisas mudam quando o usuário usa os dedos para interagir com a tela;
- As plataformas móveis têm o que é chamado de sistema de gestos para lidar com isso;
- O React Native usa os componentes reais da plataforma móvel;

React Native

Não é uma solução genérica



- O React Native não é uma solução multiplataforma que permite escrever um único aplicativo que será executado nativamente em qualquer dispositivo;
- iOS e Android são diferentes em essência;
- As experiências do usuário são diferentes;
- Escrever um único aplicativo que seja executado em ambas as plataformas é equivocados;
- O objetivo é ter componentes React Native em todos os lugares, e não escrever uma vez e executar em qualquer lugar;

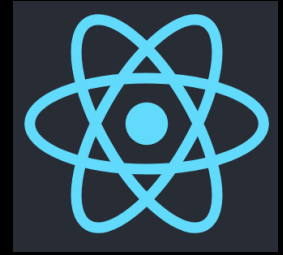
React Native

Não é uma solução genérica



- Em alguns casos, o aplicativo aproveita um *widget* específico do iOS ou um *widget* específico do Android;
- Isso fornece uma melhor experiência do usuário para a plataforma específica;

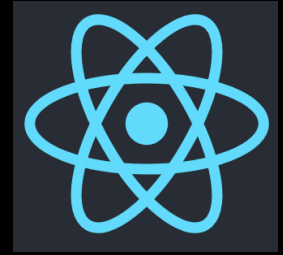
React Native Alguns Componentes



■ Alguns dos principais componentes do React Native:

REACT NATIVE UI COMPONENT	ANDROID VIEW	IOS VIEW	WEB ANALOG	DESCRIPTION
<code><View></code>	<code><ViewGroup></code>	<code><UIView></code>	A non-scrolling <code><div></code>	A container that supports layout with flexbox, style, some touch handling, and accessibility controls
<code><Text></code>	<code><TextView></code>	<code><UITextView></code>	<code><p></code>	Displays, styles, and nests strings of text and even handles touch events
<code><Image></code>	<code><ImageView></code>	<code><UIImageView></code>	<code></code>	Displays different types of images
<code><ScrollView></code>	<code><ScrollView></code>	<code><UIScrollView></code>	<code><div></code>	A generic scrolling container that can contain multiple components and views
<code><TextInput></code>	<code><EditText></code>	<code><UITextField></code>	<code><input type="text"></code>	Allows the user to enter text

React Native Alguns Componentes



■ Alguns dos principais componentes do React Native:

Basic Components

Most apps will end up using one of these basic components.

View

The most fundamental component for building a UI.

Text

A component for displaying text.

Image

A component for displaying images.

TextInput

A component for inputting text into the app via a keyboard.

ScrollView

Provides a scrolling container that can host multiple components and views.

StyleSheet

Provides an abstraction layer similar to CSS stylesheets.

React Native Alguns Componentes



■ Alguns dos principais componentes do React Native:

User Interface

These common user interface controls will render on any platform.

Button

A basic button component for handling touches that should render nicely on any platform.

Switch

Renders a boolean input.

List Views

Unlike the more generic `ScrollView`, the following list view components only render elements that are currently showing on the screen. This makes them a performant choice for displaying long lists of data.

FlatList

A component for rendering performant scrollable lists.

SectionList

Like `FlatList`, but for sectioned lists.

React Native Alguns Componentes



■ Alguns dos principais componentes do React Native:

Others

These components may be useful for certain applications. For an exhaustive list of components and APIs, check out the sidebar to the left (or menu above, if you are on a narrow screen).

ActivityIndicator

Displays a circular loading indicator.

Alert

Launches an alert dialog with the specified title and message.

Animated

A library for creating fluid, powerful animations that are easy to build and maintain.

Dimensions

Provides an interface for getting device dimensions.

KeyboardAvoidingView

Provides a view that moves out of the way of the virtual keyboard automatically.

Linking

Provides a general interface to interact with both incoming and outgoing app links.

Modal

Provides a simple way to present content above an enclosing view.

PixelRatio

Provides access to the device pixel density.

RefreshControl

This component is used inside a `ScrollView` to add pull to refresh functionality.

StatusBar

Component to control the app status bar.

Funções Arrow - revendo



- Facilidade incluída no ES6;
- Encurta a escrita das funções;
- Representada pelo operador `=>`;
- Exemplos:

```
alo = function() {  
    return "Alô Vocês!";  
}
```

```
alo = () => {  
    return "Alô Vocês!";  
}
```

```
alo = () => "Alô Vocês";
```

```
alo = (val) => "Alô " + val;
```

```
alo = val => "Alô " + val;
```

```
bem = nome => "Seja bem-vindo ${nome}!!!";
```

```
calc = (num1, num2) => num1 * num2;
```

Funções Arrow - revendo



■ Exemplos:

```
1) soma = function(x,y) {  
    return x+y;  
}
```

Fica → soma = (x, y) => x + y; // Desobrigado a escrever o return

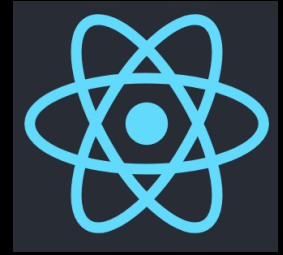
```
taxa = valor => {  
    fator = 3;  
    return fator * valor; // Obrigado a escrever o return quando na presença de {}  
}
```

Funções JSX - revendo



- JSX significa JavaScript XML;
- Deve ser transformado em JavaScript válido;
- Ganhou popularidade com o React, mas desde então também viu outras implementações;
- Permite escrever elementos HTML em JavaScript e colocá-los no DOM sem nenhum método createElement() e/ou appendChild();
- JSX converte *tags* HTML em elementos de react;

Funções JSX - revendo



■ Exemplos:

1) `const myElement = <h1>I Love JSX!</h1>;` `// HTML no JavaScript!!!`

```
const root = ReactDOM.createRoot(document.getElementById('root'));  
root.render(myElement);
```

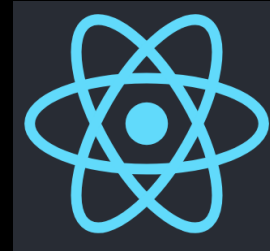
2) `const myElement = (

 Apples
 Bananas
 Cherries

)`;
`const root = ReactDOM.createRoot(document.getElementById('root'));`
`root.render(myElement);`

3) `const myElement = (
 <div>
 <h1>I am a Header.</h1>
 <h1>I am a Header too.</h1>
 </div>
)`;
`const root = ReactDOM.createRoot(document.getElementById('root'));`
`root.render(myElement);`

Funções JSX - revendo



- Expressões JSX são escritas dentro de chaves { };
- A expressão pode ser uma variável ou propriedade React ou qualquer outra expressão JavaScript válida;
- O JSX executará a expressão e retornará o resultado;
- Exemplo;

```
const myElement = <h1>React is {5 + 5} times better with JSX</h1>;
```
- O React/React Native suporta a instrução **if**, mas não dentro do JSX;
- Para poder usar instruções condicionais no JSX, deve-se colocar a instrução **if** fora do JSX ou usar uma expressão ternária;

Funções JSX - revendo



■ Exemplos com if:

1) const x = 5;

```
let text = "Goodbye";
```

```
if (x < 10) {  
  text = "Hello";
```

```
}
```

```
const myElement = <h1>{text}</h1>;
```

```
const root = ReactDOM.createRoot(document.getElementById('root'));  
root.render(myElement);
```

2) const x = 5;

```
const myElement = <h1>{(x) < 10 ? "Hello" : "Goodbye"}</h1>;
```

React Native

Componentes - função e classe



- Com o React/React Native, podem ser criados componentes usando classes ou funções;
- Originalmente, os componentes de classe eram os únicos componentes que podiam ter estado;
- Desde a introdução da API Hooks do React, podem ser adicionados estados aos componentes de função;
- Quando um projeto é criado, o **padrão é a função**;
- Em nosso treinamento, utilizaremos, em sua maioria, funções;

React Native Componentes - função e classe



■ Função:

```
import React from 'react';
import {Text, View} from 'react-native';

const App = () => {
  return (
    <View
      style={{
        flex: 1,
        justifyContent: 'center',
        alignItems: 'center',
      }}>
      <Text>Estou Aqui Para Vocês!</Text>
    </View>
  );
};

export default App;
```

React Native Componentes - função e classe



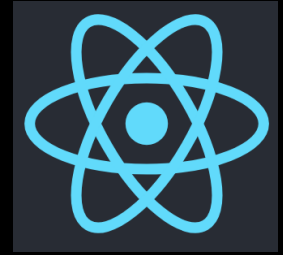
■ Classe:

```
import React, {Component} from 'react';
import {Text, View} from 'react-native';

class App extends Component {
  render() {
    return (
      <View
        style={{
          flex: 1,
          justifyContent: 'center',
          alignItems: 'center',
        }}>
        <Text>Estou Aqui Para Vocês!</Text>
      </View>
    );
  }
}

export default App;
```

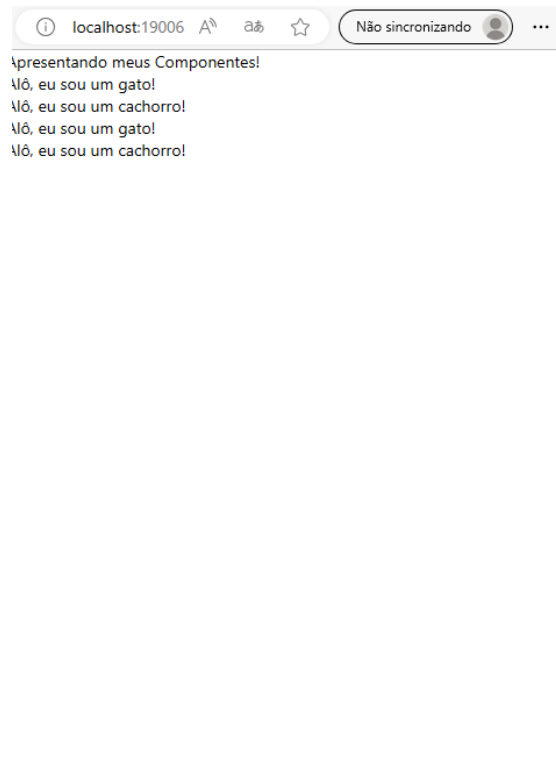
React Native component - revendo



```
import React from 'react';
import {Text} from 'react-native';

const Gato = () => {                                //Identifica o componente
  return <Text>Alô, eu sou um gato!</Text>;          // Renderiza o Text
};
const Cachorro = () => {                            //Identifica o componente
  return <Text>Alô, eu sou um cachorro!</Text>;      // Renderiza o Text
};
const MeuAnimais = () => {
  return (
    <View>
      <Text>Apresentando meus Componentes!</Text>
      <Gato/>                                         // Invocando os componentes
      <Cachorro/>
      <Gato/>
      <Cachorro/>
    </View>
  );
};
export default MeuAnimais;
```

React Native component - revendo



React Native props - revendo



- As propriedades dos componentes React Native são simplesmente escritas como props;
- No React Native, a maioria dos componentes pode ser personalizada, no momento de sua renderização, com diferentes parâmetros;
- Esses parâmetros são conhecidos como props. Eles são imutáveis, qual seja, não podem ser alterados;

React Native props - revendo



```
import React from 'react';
import {Text, View} from 'react-native';

type GatoProps = {
  nome: string;
};
type CachorroProps = {
  nome: string;
};

const Gato = (props: GatoProps) => {
  return (
    <Text>Alô, Sou o gato {props.nome}!</Text>
  );
};

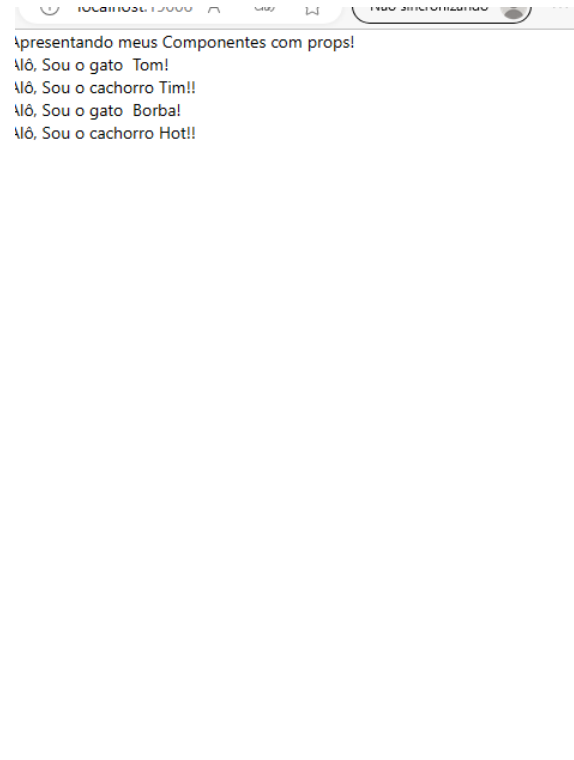
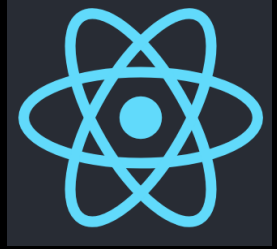
const Cachorro = (props: CachorroProps) => {
  return (
    <Text>Alô, Sou o cachorro {props.nome}!!</Text>
  );
};
```

React Native props - revendo



```
const MeuAnimais = () => {  
  return (  
    <View>  
      <Text>Apresentando meus Componentes com props!</Text>  
      <Gato nome='Tom'/>  
      <Cachorro nome='Tim'/>  
      <Gato nome='Borba'/>  
      <Cachorro nome='Hot'/>  
    </View>  
  );  
};  
export default MeuAnimais;
```

React Native props - revendo



React Native props - revendo



```
import React from 'react';
import {Text, View} from 'react-native';

type GatoProps = {
  nome: string;
  raca: string;
};

type CachorroProps = {
  nome: string;
  raca: string;
};

const Gato = (props: GatoProps) => {
  return (
    <View>
      <Text>Alô, Sou o gato {props.nome}</Text>
      <Text>Minha raça é {props.raca}</Text>
    </View>
  );
};
```

React Native props - revendo



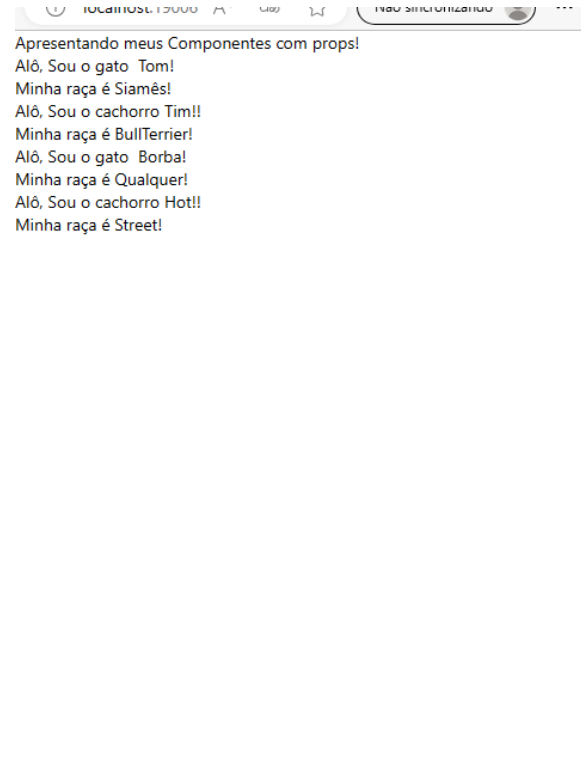
```
const Cachorro = (props:CachorroProps) => {  
  return (  
    <View>  
      <Text>Alô, Sou o cachorro {props.nome}!!</Text>  
      <Text>Minha raça é {props.raca}!</Text>  
    </View>  
  );  
};
```

React Native props - revendo



```
const MeuAnimais = () => {  
  return (  
    <View>  
      <Text>Apresentando meus Componentes com props!</Text>  
      <Gato nome='Tom' raca='Siamês'/>  
      <Cachorro nome='Tim' raca='BullTerrier'/>  
      <Gato nome='Borba' raca='Qualquer'/>  
      <Cachorro nome='Hot' raca='Street'/>  
    </View>  
  );  
};  
export default MeuAnimais;
```

React Native props - revendo



React Native props - revendo



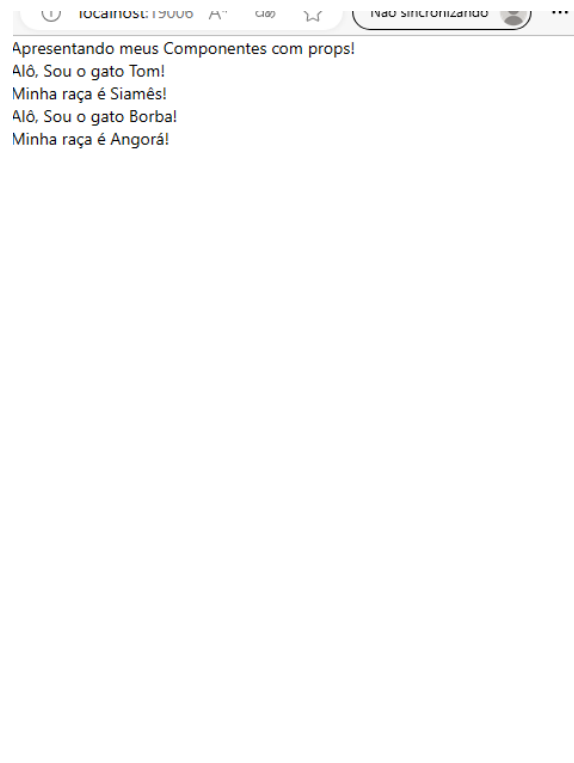
```
import React from 'react';
import {Text, View} from 'react-native';

const Gato = ({nome,raca}) => {
  return (
    <View>
      <Text>Alô, Sou o gato {nome}</Text>
      <Text>Minha raça é {raca}</Text>
    </View>
  );
};

const MeuAnimais = () => {
  return (
    <View>
      <Text>Apresentando meus Componentes com props!</Text>
      <Gato nome='Tom' raca={'Siamês'}/>
      <Gato nome='Borba' raca={'Angorá'}/>
    </View>
  );
};

export default MeuAnimais;
```

React Native props - revendo



React Native state - revendo



- *State* armazena os estados dos dados de um componente;
- Útil para lidar com dados que mudam com o tempo ou que originam da interação do usuário;
- O *state* é a memória dos componentes;
- O Hook facilitou o uso do *state* permitindo seu uso em uma função;
- O **useState** é um *hook* utilizado para manipular os estados do componente;
- O **useState** Hook permite rastrear o estado de um componente de função;

React Native state - revendo



- O **useState** aceita um estado inicial e retorna dois valores, a saber:
 - O estado atual.
 - Uma função que atualiza o estado.
- Exemplos:
 - `const [cor, setCor] = useState("");`
 - `const [cor, setCor] = useState("Azul");`
 - `const [numero, setNumero] = useState(0);`

React Native state - revendo



```
import React, {useState} from 'react';
import {Button, Text, View} from 'react-native';

const Gato = ({nome}) => {
  const [taFaminto, setTaFaminto] = useState(true);

  return (
    <View>
      <Text>
        Eu sou o gato {nome}, e eu estou {taFaminto ? 'faminto' : 'satisfeito'}!
      </Text>
      <Button
        onPress={() => {setTaFaminto(false);}}
        disabled={!taFaminto}
        title={taFaminto ? 'Quero leite, por favor!' : 'Obrigado!'}
      />
    </View>
  );
};
```

React Native state - revendo



```
const MeuAnimais = () => {  
  return (  
    <View>  
      <Text>Apresentando meus Componentes com props e state!</Text>  
      <Gato nome='Tom'/>  
    </View>  
  );  
};  
  
export default MeuAnimais;
```

React Native state - revendo



Apresentando meus Componentes com props e state!
Eu sou o gato Tom, e eu estou faminto!

QUERO LEITE, POR FAVOR!

Apresentando meus Componentes com props e state!
Eu sou o gato Tom, e eu estou satisfeito!

OBRIGADO!

React Native state - revendo



```
import React, {useState} from 'react';
import {Button, Text, View} from 'react-native';

const Gato = ({nome}) => {
  const [taFaminto, setTaFaminto] = useState(true);
  const [raca, setRaca] = useState('Angorá');
  const [cor, setCor] = useState('Amarelo');

  return (
    <View>
      <Text>
        Eu sou o gato {nome}, e eu estou {taFaminto ? 'faminto' : 'satisfeito'}, minha raca é {raca} e
        minha cor é {cor}!
      </Text>
    </View>
  );
};
```

React Native state - revendo



```
const MeuAnimais = () => {  
  return (  
    <View>  
      <Text>Apresentando meus Componentes com props e state!</Text>  
      <Gato nome='Tom'/>  
    </View>  
  );  
};  
export default MeuAnimais;
```

React Native state - revendo



Apresentando meus Componentes com props e state!
Eu sou o gato Tom, e eu estou faminto, minha raca é Angorá e minha cor é Amarelo!

React Native state - revendo



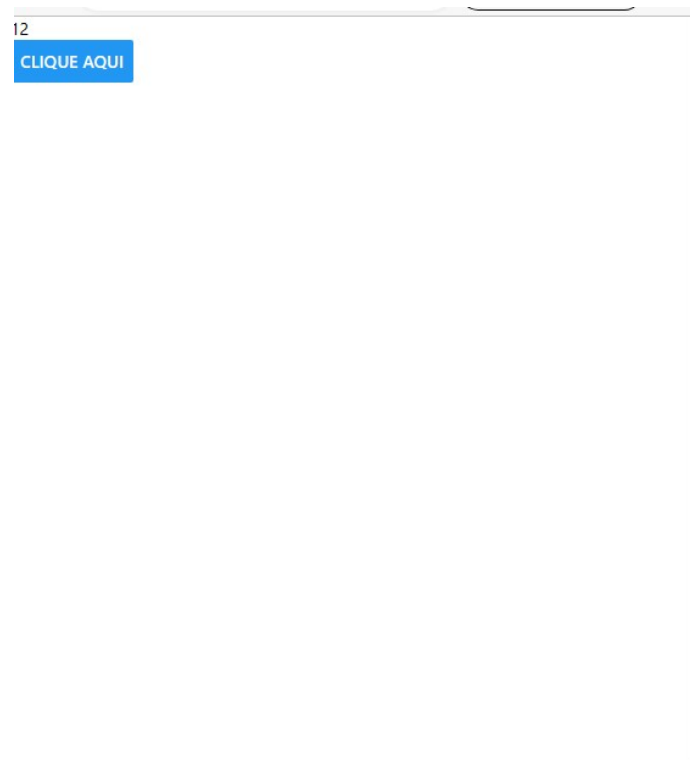
```
import React, { useState } from "react";
import { View, Text, Button } from "react-native";

const Contador = () => {
  const [contador, setContador] = useState(10);
  return (
    <View>
      <Text>{contador}</Text>
      <Button title="clique aqui" onPress={() => setContador(contador + 1)} />
    </View>
  );
};

const App = () => {
  return (
    <View>
      <Contador />
    </View>
  );
};
export default App;
```

Obs.: O estado é alterado a cada clicada no botão.

React Native state



React Native state - revendo



```
const TesteNetInfo = () => {  
  const netInfo = useNetInfo();  
  const [autorizado, setAutorizado] = useState(false);  
  
  const requestCameraPermission = async () => {  
    try {  
      const granted = await PermissionsAndroid.request(PermissionsAndroid.PERMISSIONS.ACCESS_FINE_LOCATION, {  
        title: 'Permissão para dados da Rede',  
        message: 'Acesso aos dados da Rede',  
        buttonNegative: 'Cancelar',  
        buttonPositive: 'OK',  
      },  
    );  
    if (granted === PermissionsAndroid.RESULTS.GRANTED) {  
      setAutorizado(true);  
    } else {  
      setAutorizado(false);  
    }  
    } catch (err) {  
      setAutorizado(false);  
    }  
  };  
};
```

Obs.: Como está contida na função maior TesteNetInfo, a função requestCameraPermission pode usar **setAutorizado**.

React Native state - revendo



```
1) const [estado, setEstado] = useState({nome:'Maria', idade:30});
    const atualizaNome = () => {
      setEstado({ ...estado, nome: 'Joana' });
    };
    const atualizaIdade = () => {
      setEstado({ ...estado, idade:estado.idade + 1 });
    };
```

2) Arrays

```
const [array, setArray] = useState([1, 2, 3, 4, 5]);
const addItem = () => {
  setArray([...array,6]);
};
```

```
3) const [mapRegion, setMapRegion] = useState({
  latitude: -22.999,
  longitude: -43.4191,
});
const atualizaLocalizacao = () => {
  setMapRegion({ ...mapRegion, latitude: -22.999});
};
```

Arrays

Alguns métodos - revendo



`concat(array1[,...,arrayn])`

Concatena arrays e retorna um array com os arrays concatenados.

`every(function())`

Retorna true se todos os elementos atendem aos critérios estabelecidos pela função invocada.

`entries()`

Retorna um Array Iterator com um par chave/valor.

`fill(dado[,inicio[,fim]])`

Preenche o array com o dado. Os dados existentes são sobrepostos.

`filter(function)`

Cria um novo array selecionando apenas os elementos que satisfazem uma condição especificada.

`find(function)`

Retorna o primeiro elemento do array que atendeu ao critério da função invocada.

`findIndex(function)`

Retorna o índice do primeiro elemento do array que atendeu ao critério da função.

`forEach(function(parametros))`

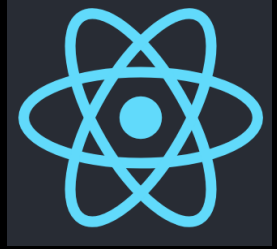
Executa a função em cada elemento do array.

`includes(dado[,indice])`

Retorna true se o array contém o dado específico, a partir de determinado índice.

Arrays

Alguns métodos - revendo



`indexOf(dado[,posicao])`

Retorna a posição do dado.

`Array.isArray(objeto)`

Retorna true se o objeto é um array.

`join(separador)`

Retorna o array como uma string.

`lastIndexOf(dado[,posicao])`

Retorna a última posição do dado pesquisado.

`length`

Retorna o número de elementos no array..

`map(function)`

Transforma cada elemento de um array e cria um novo array com os valores transformados.

`pop()`

Remove o último elemento do array.

`push(dado1[,dado2...[,dadon]])`

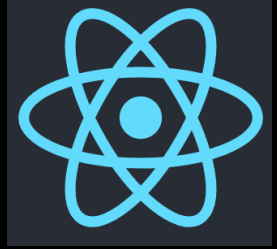
Adiciona elementos ao fim do array.

`reverse()`

Inverte a ordem dos elementos do array.

Arrays

Alguns métodos - revendo



`shift()`

Remove o primeiro elemento do array.

`slice(inicio,fim)`

Retorna os elementos selecionados em um novo array. Suporta índices negativos, neste caso a operação é feita de trás para frente.

`sort()`

Classifica os elementos do array em ordem ascendente.

`splice(posicao,quantos,dado1[..dadoN])`

Adiciona ou remove elementos do array.

`toString()`

Retorna uma string com os valores separados por vírgula.

`unshift(dado1[,dado2.....[,dadon]])`

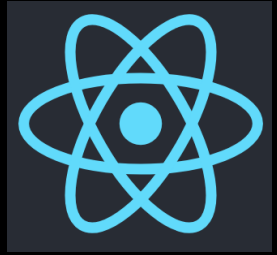
Adiciona elementos ao início do array.

`valueOf()`

Retorna o array em si.

Arrays

Alguns métodos - revendo



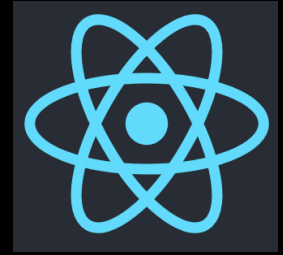
```
cores = ["Azul", "Verde", "Amarelo", "Vermelho", "Preto"];
carros = ["Astra", "Fiat 500", "Uno Way", "BMW", "Audi", "Fusca", "Citroen"];
notas = [5.5, 4.3, 8.5, 10.0, 9.1, 3.5, 6.1];

document.write("<h4> cores array -> " + cores + "<br></h4>");
document.write("<h4> carros array -> " + carros + "<br></h4>");
document.write("<h4> notas array -> " + notas + "<br></h4>");
document.write("<h4> concat cores + carros -> " + cores.concat(carros) + "<br></h4>");
document.write("<h4> join cores -> " + cores.join(' ') + "<br></h4>");
document.write("<h4> every, todas as notas>=7 -> " + notas.every(checaNota) + "<br></h4>");
document.write("<h4> filter, array com notas>=7 -> " + notas.filter(checaNota) + "<br></h4>");
document.write("<h4> find, primeiro elemento com nota>=7 -> " + notas.find(checaNota) + "<br></h4>");
document.write("<h4> includes, verifica se tem 10 -> " + notas.includes(10) + "<br></h4>");
document.write("<h4> indexOf, qual o indice do 10 -> " + notas.indexOf(10) + "<br></h4>");
document.write("<h4> map, array com notas/2 -> " + notas.map(divideNota) + "<br></h4>");
document.write("<h4> reverse, inverte cores -> " + cores.reverse() + "<br></h4>");
document.write("<h4> slice, parte carros -> " + carros.slice(1,4) + "<br></h4>");
document.write("<h4> sort, ordena carros -> " + carros.sort() + "<br></h4>");
maisCarros = carros;
maisCarros.splice(3,0,"LandRover","Ferrari");
document.write("<h4> splice, adiciona carros -> " + maisCarros + "<br></h4>");
maisCarros.splice(3,2);
document.write("<h4> splice, remove carros -> " + maisCarros + "<br></h4>");
maisCarros.splice(3,2,"LandRover","Ferrari");
document.write("<h4> splice, adiciona e remove carros -> " + maisCarros + "<br></h4>");

function checaNota(nota){
  return nota>=7.0;
}
function divideNota(nota){
  return nota/2;
}
```

Arrays

Alguns métodos - revendo



Utilizando Arrays – Métodos

cores array -> Azul,Verde,Amarelo,Vermelho,Preto

carros array -> Astra,Fiat 500,Uno Way,BMW,Audi,Fusca,Citroen

notas array -> 5.5,4.3,8.5,10,9.1,3.5,6.1

concat cores + carros -> Azul,Verde,Amarelo,Vermelho,Preto,Astra,Fiat 500,Uno Way,BMW,Audi,Fusca,Citroen

join cores -> Azul Verde Amarelo Vermelho Preto

every, todas as notas>=7 -> false

filter, array com notas>=7 -> 8.5,10,9.1

find, primeiro elemento com nota>=7 -> 8.5

includes, verifica se tem 10 -> true

indexOf, qual o indice do 10 -> 3

map, array com notas/2 -> 2.75,2.15,4.25,5,4.55,1.75,3.05

reverse, inverte cores -> Preto,Vermelho,Amarelo,Verde,Azul

slice, parte carros -> Fiat 500,Uno Way,BMW

sort, ordena carros -> Astra,Audi,BMW,Citroen,Fiat 500,Fusca,Uno Way

splice, adiciona carros -> Astra,Audi,BMW,LandRover,Ferrari,Citroen,Fiat 500,Fusca,Uno Way

splice, remove carros -> Astra,Audi,BMW,Citroen,Fiat 500,Fusca,Uno Way

splice, adiciona e remove carros -> Astra,Audi,BMW,LandRover,Ferrari,Fusca,Uno Way

React Native

Ambiente de desenvolvimento



- Instalação do ambiente para o desenvolvimento:
 - Instalação do NodeJS e npm(Node Package Manager):
 - Fazer download em nodejs.org e instalar.
 - Instalação global do React Native(prompt do DOS):
 - `npm install -g create-react-native-app`
 - Instalação global React Native CLI(prompt do DOS):
 - `npm install -g react-native-cli`
 - Após as instalações, emitir os comandos(prompt do DOS):
 - `node --version` e `npm --version`
 - Atualizar o React Native:
 - `npm install -g react-native-git-upgrade`

React Native

Ambiente de desenvolvimento



- Instalação do ambiente para o desenvolvimento:
 - Instalação global do Expo CLI(prompt do DOS):
 - `npm install -g expo-cli`
 - Instalação do Expo Go no aparelho móvel:
 - Baixar do Google Play Store ou Apple Store.

React Native

Primeiro Exemplo - Criando

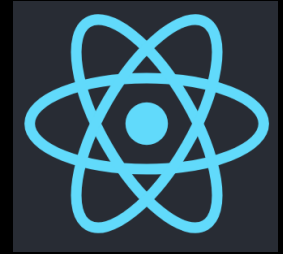


■ Criando o Primeiro Exemplo:














- No diretório raiz, emitir o comando(prompt do DOS):
 - **expo init PrimeiroExemplo**OU
- No diretório raiz, emitir o comando(prompt do DOS):
 - **create-react-native-app PrimeiroExemplo**
 - Utilizar o *default*

React Native

Primeiro Exemplo - Diretório

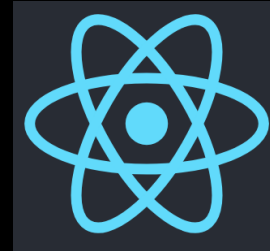


■ Conteúdo do diretório do Primeiro Exemplo:

 .expo	18/05/2023 17:51	Pasta de arquivos	
 android	18/05/2023 17:48	Pasta de arquivos	
 ios	18/05/2023 17:48	Pasta de arquivos	
 node_modules	18/05/2023 17:49	Pasta de arquivos	
 .gitattributes	26/10/1985 05:15	Documento de Te...	1 KB
 .gitignore	26/10/1985 05:15	Documento de Te...	1 KB
 App	26/10/1985 05:15	Arquivo JavaScript	1 KB
 app	18/05/2023 17:48	Arquivo Fonte JSON	1 KB
 babel.config	26/10/1985 05:15	Arquivo JavaScript	1 KB
 index	26/10/1985 05:15	Arquivo JavaScript	1 KB
 metro.config	26/10/1985 05:15	Arquivo JavaScript	1 KB
 package	18/05/2023 17:48	Arquivo Fonte JSON	1 KB
 package-lock	18/05/2023 17:49	Arquivo Fonte JSON	514 KB

React Native

Primeiro Exemplo - Diretório



■ Conteúdo do diretório do Primeiro Projeto:

- **node_modules** contém todos os pacotes de terceiros em nosso aplicativo. Quaisquer novas dependências e dependências de desenvolvimento vão aqui.
- **babel** é um *transpiler* que compila JavaScript experimental mais recente em versões mais antigas para que fique compatível com diferentes plataformas.
- **App.js** é onde reside o código de nosso aplicativo.
- **app.json** é um arquivo de configuração que nos permite adicionar informações sobre nosso aplicativo Expo.
- **package.json** é onde fornecemos informações do aplicativo para nosso gerenciador de pacotes bem como especificar todas as dependências do nosso projeto.
- **android** é a pasta onde ficam as informações necessárias a um app Android.
- **ios** é a pasta onde ficam as informações necessárias a um app Android.

React Native

Primeiro Exemplo - Iniciar



Iniciar o Primeiro Exemplo:

- Sob o diretório PrimeiroExemplo, emitir o comando(prompt do DOS):
 - **npx expo start** OU
 - **npm start**
- NPM é um gerenciador de pacotes usado para instalar, excluir e atualizar pacotes Javascript;
- NPX é um executor de pacotes, e é usado para executar pacotes Javascript diretamente, sem instalá-los;


React Native

Primeiro Exemplo - Iniciar



- Pressionar **w** para ver a execução no navegador web. Irá reclamar pedindo para instalar pacotes de navegação web. Siga as instruções!

```
Logs for your project will appear below. Press Ctrl+C to exit.
Starting Webpack on port 19006 in development mode.



> Metro waiting on exp://192.168.1.65:19000
> Scan the QR code above with Expo Go (Android) or the Camera app (iOS)

> Web is waiting on http://localhost:19006

> Press a | open Android
> Press w | open web

> Press j | open debugger
> Press r | reload app
> Press m | toggle menu

> Press ? | show all commands

> Open in the web browser...
```

React Native

Primeiro Exemplo - Iniciando



- Abrir o Expo Go no dispositivo móvel, selecionar QR Code e apontar para o QR Code da tela do prompt.

```
Logs for your project will appear below. Press Ctrl+C to exit.
Starting Webpack on port 19006 in development mode.



> Metro waiting on exp://192.168.1.65:19000
> Scan the QR code above with Expo Go (Android) or the Camera app (iOS)
> Web is waiting on http://localhost:19006

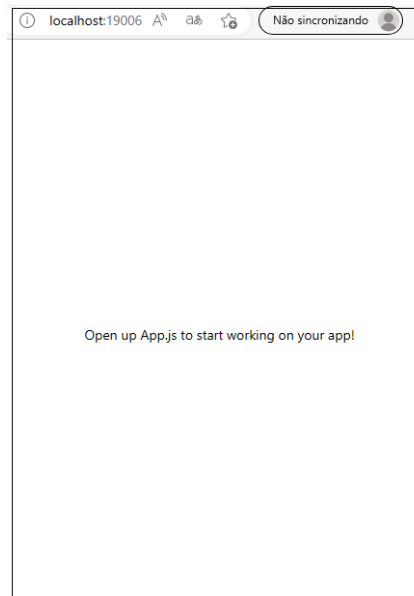
> Press a | open Android
> Press w | open web

> Press j | open debugger
> Press r | reload app
> Press m | toggle menu

> Press ? | show all commands
> Open in the web browser...
```

React Native

Primeiro Exemplo - Tela web



React Native

Primeiro Exemplo - Código Original



```
import { StatusBar } from 'expo-status-bar';
import React from 'react';
import { StyleSheet, Text, View } from 'react-native';

export default function App() {
  return (
    <View style={styles.container}>
      <Text>Open up App.js to start working on your app!</Text>
      <StatusBar style="auto" />
    </View>
  );
}
const styles = StyleSheet.create({
  container: {
    flex: 1,
    backgroundColor: '#fff',
    alignItems: 'center',
    justifyContent: 'center',
  },
});
```

Obs.: Editar o arquivo App.js.

React Native

Primeiro Exemplo – Mudando



```
import { StatusBar } from 'expo-status-bar';
import React from 'react';
import { StyleSheet, Text, View } from 'react-native';

export default function App() {
  return (
    <View style={styles.container}>
      <Text style={styles.text}>Open up App.js to start working on your app!</Text>
      <StatusBar style="auto" />
    </View>
  );
}

const styles = StyleSheet.create({
  container: {
    flex: 1,
    backgroundColor: 'fff',
    alignItems: 'center',
    justifyContent: 'center',
  },
  text: {
    fontWeight: "bold",
    color: '#2196f3',
    fontSize: 18
  }
});
```

Obs.: Ao salvar a alteração, será visualizada automaticamente.

React Native

Exercício A1



- Crie e execute um projeto tendo como código o Gato Faminto.

React Native

Exercício A2



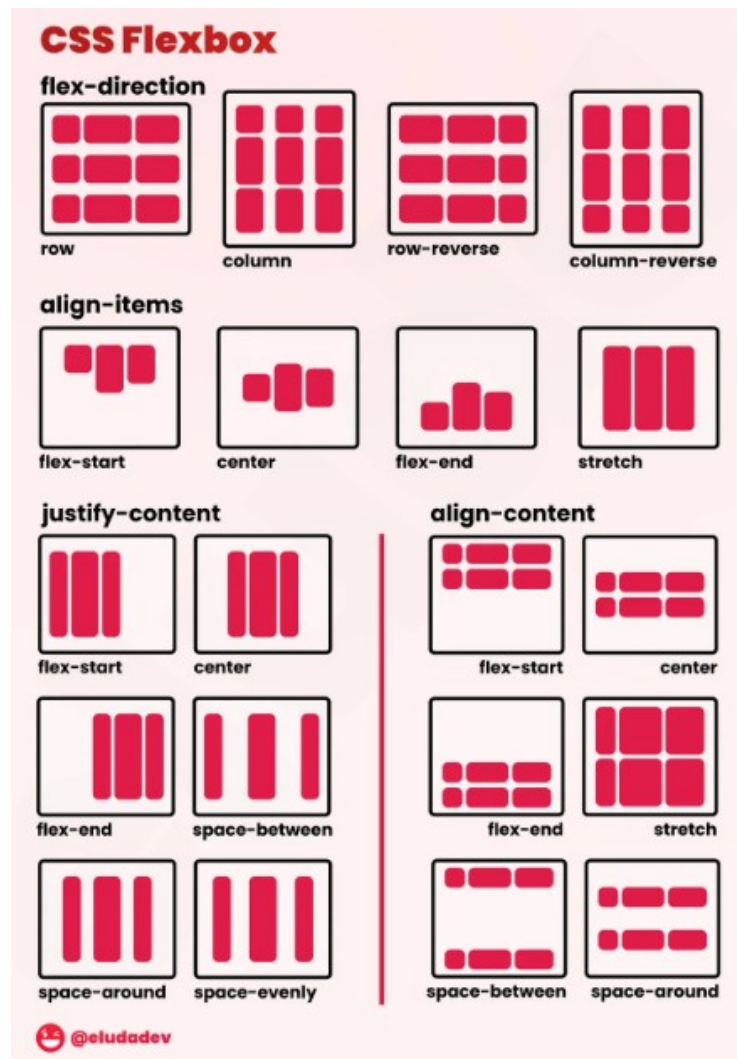
- Crie e execute um projeto tendo como código o Contador.

React Native Layout - Flexbox



- Criado para para acomodar diferentes tamanhos de tela;
- Flexbox é o que nome diz: um modelo de caixa que é flexível;
- A caixa atua como um *container* que possui os elementos filhos dentro dessa caixa;
- Tanto o *container* quanto os elementos filho são flexíveis;
- Oferece três principais propriedades: `flexDirection`, `justifyContent` e `alignItems`;
- Detalhes de uso em:
 - <https://reactnative.dev/docs/flexbox> (vamos visitar!)

React Native Layout - Flexbox



React Native Layout - Flexbox - FlexDirection



- Controla a direção na qual os filhos de um nó são dispostos;
- Também chamado de eixo principal;
- O eixo transversal é o eixo perpendicular ao eixo principal;
- Propriedades:
 - column (valor padrão) Alinha os filhos de cima para baixo. Se o empacotamento estiver ativado, a próxima linha começará à direita do primeiro item na parte superior do container.
 - row Alinha os filhos da esquerda para a direita. Se o agrupamento estiver ativado, a próxima linha começará no primeiro item à esquerda do container.
 - column-reverse Alinha os filhos de baixo para cima. Se o agrupamento estiver ativado, a próxima linha começará à direita do primeiro item na parte inferior do container.
 - row-reverse Alinha os filhos da direita para a esquerda. Se o agrupamento estiver ativado, a próxima linha começará no primeiro item à direita do container.

React Native Layout - Flexbox - JustifyContent



- Descreve como alinhar filhos dentro do eixo principal do *container*;
- Propriedades:
 - flex-start(valor padrão) Alinha os filhos de um container ao início do eixo principal do *container*.
 - flex-end Alinha os filhos de um *container* ao final do eixo principal do *container*.
 - center Alinha os filhos de um *container* no centro do eixo principal do *container*.
 - space-between Espaço, uniformemente, os filhos no eixo principal do *container*, distribuindo o espaço restante entre os filhos.
 - space-around Distribui, uniformemente, os filhos no eixo principal do *container*, distribuindo o espaço restante ao redor dos filhos.
 - space-evenly Distribui filhos, uniformemente, dentro do *container* de alinhamento ao longo do eixo principal.

React Native Layout - Flexbox - AlignItems



- Descreve como alinhar filhos ao longo do eixo transversal do *container*;
- É muito semelhante a `justifyContent`, mas em vez de ser aplicado ao eixo principal, `alignItems` se aplica ao eixo cruzado;
- Propriedades:
 - `stretch` (valor padrão) Alonga os filhos de um container para corresponder à altura do eixo transversal do container.
 - `flex-start` Alinha os filhos de um container ao início do eixo cruzado do container.
 - `flex-end` Alinha os filhos de um container ao final do eixo transversal do container.
 - `center` Alinha os filhos de um container no centro do eixo transversal do container.
 - `baseline` Alinha os filhos de um container ao longo de uma linha de base comum. Filhos individuais podem ser definidos como a linha de base de referência para seus pais.

React Native

Segundo Exemplo

Layout – Style – importando



```
import React from "react";
import { Text, View } from "react-native";
import styles from "./styles";    // Aqui os estilos serão importados

export default function App() {
  return (
    <View style={styles.container}>
      <View style={styles.box}>
        <Text style={styles.boxText}>Sou uma Caixa!!!</Text>
      </View>
    </View>
  );
}
```

Códigos do livro React and React Native – Adam Boduch, Roy Derks

React Native Segundo Exemplo Layout – Style – importando



```
import {Platform, StyleSheet, StatusBar} from "react-native";
```

```
export default StyleSheet.create({  
  container: {  
    flex: 1,  
    flexDirection: "column",  
    alignItems: "center",  
    justifyContent: "space-around",  
    backgroundColor: "ghostwhite",  
    ...Platform.select({  
      ios: { paddingTop: 20 },  
      android: { paddingTop: StatusBar.currentHeight }  
    })  
  },  
},
```

React Native

Segundo Exemplo

Layout – Style – importando



```
box: {  
  width: 300,  
  height: 100,  
  justifyContent: "center",  
  alignItems: "center",  
  backgroundColor: "lightgray",  
  borderWidth: 1,  
  borderStyle: "dashed",  
  borderColor: "darkslategray"  
},  
boxText: {  
  color: "darkslategray",  
  fontWeight: "bold"  
}  
}  
);
```

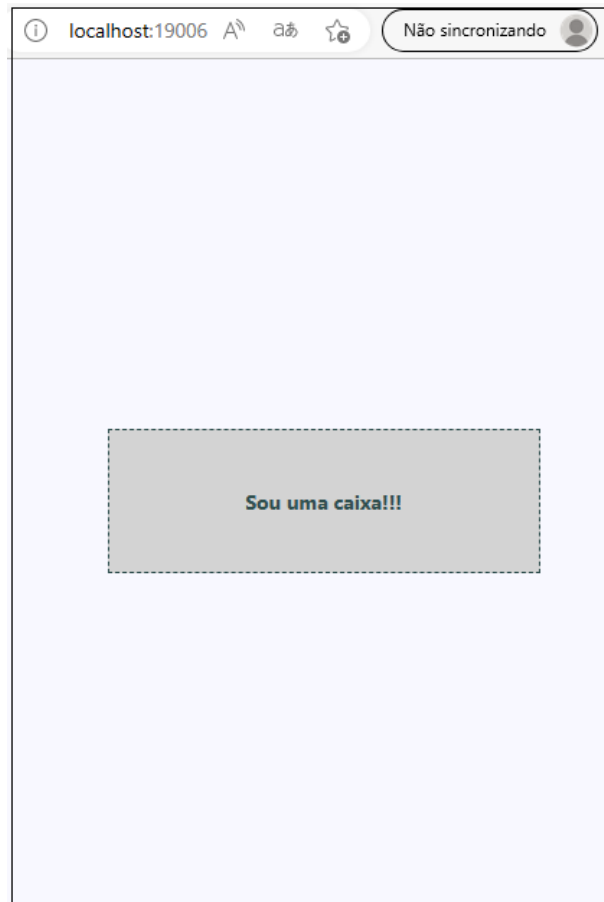
Obs.: Salvar o arquivo com o nome **styles**, do tipo js, no diretório raiz do aplicativo.

Criar e Executar o projeto SegundoExemplo.

React Native

Segundo Exemplo

Layout – Style – importando



React Native

Segundo Exemplo - Mudando Layout - Style – importando



```
import React from "react";
import { Text, View } from "react-native";
import styles from "./styles";

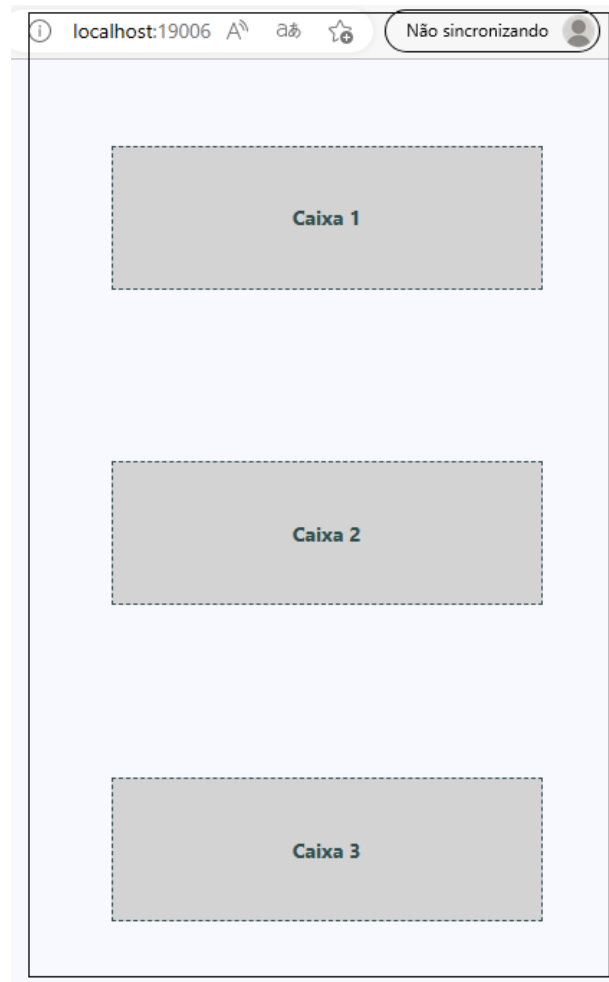
export default function App() {
  return (
    <View style={styles.container}>
      <View style={styles.box}>
        <Text style={styles.boxText}>#Caixa 1</Text>
      </View>
      <View style={styles.box}>
        <Text style={styles.boxText}>#Caixa 2</Text>
      </View>
      <View style={styles.box}>
        <Text style={styles.boxText}>#Caixa 3</Text>
      </View>
    </View>
  );
}
```

Relembrando: Em termos de tags HTML, a tag `<View>` é similar a tag `<div>`, enquanto a tag `<Text>` é similar tag `<p>`.

Obs.: Coloquem o celular nas posições horizontal e vertical.

React Native

Segundo Exemplo - Mudando Layout – Style – importando



React Native

Segundo Exemplo - Mudando Layout - Style – importando



```
import { Platform, StyleSheet, StatusBar } from "react-native";
```

```
export default StyleSheet.create({
```

```
  container: {
```

```
    flex: 1,
```

```
    flexDirection: "column",
```

```
    backgroundColor: "ghostwhite",
```

```
    alignItems: "center",
```

```
    justifyContent: "space-around",
```

```
    ...Platform.select({
```

```
      ios: { paddingTop: 20 },
```

```
      android: { paddingTop: StatusBar.currentHeight } })),
```


React Native

Segundo Exemplo - Mudando Layout – Style – importando

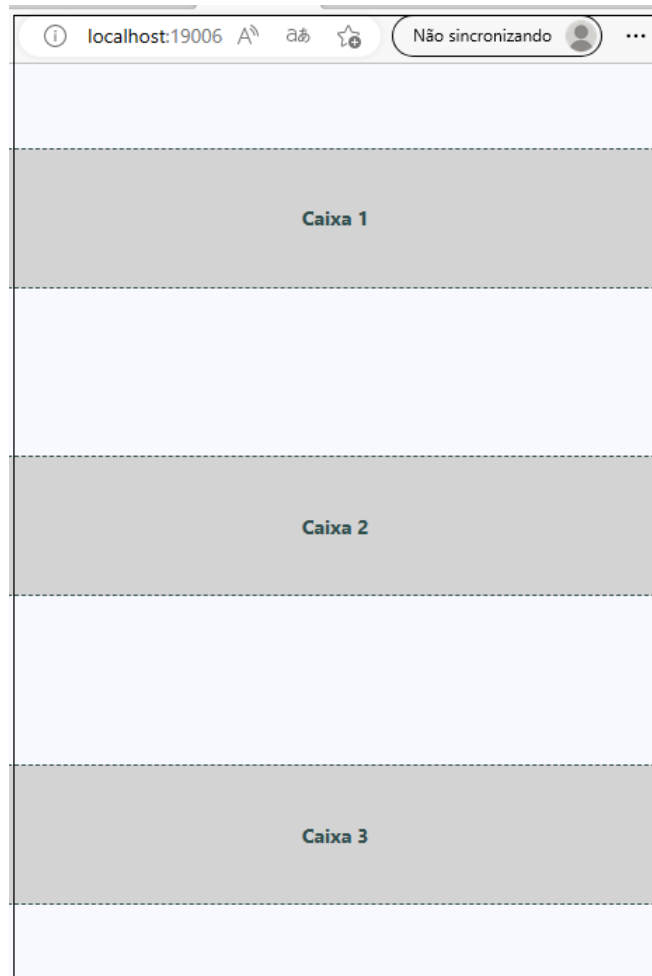


```
box: {  
  height: 100,  
  justifyContent: "center",  
  alignSelf: "stretch",  
  alignItems: "center",  
  backgroundColor: "lightgray",  
  borderWidth: 1,  
  borderStyle: "dashed",  
  borderColor: "darkslategray"  
},  
boxText: {  
  color: "darkslategray",  
  fontWeight: "bold"  
}  
});
```

Obs.: retirou-se, do box, a width.

React Native

Segundo Exemplo - Mudando Layout – Style – importando



React Native

Segundo Exemplo - Mudando Layout - Style – importando



```
import { Platform, StyleSheet, StatusBar } from "react-native";

export default StyleSheet.create({
  container: {
    flex: 1,
    flexDirection: "row",
    backgroundColor: "ghostwhite",
    alignItems: "center",
    justifyContent: "space-around",
    ...Platform.select({
      ios: { paddingTop: 20 },
      android: { paddingTop: StatusBar.currentHeight }
    })
  },
});
```

React Native

Segundo Exemplo - Mudando Layout - Style – importando

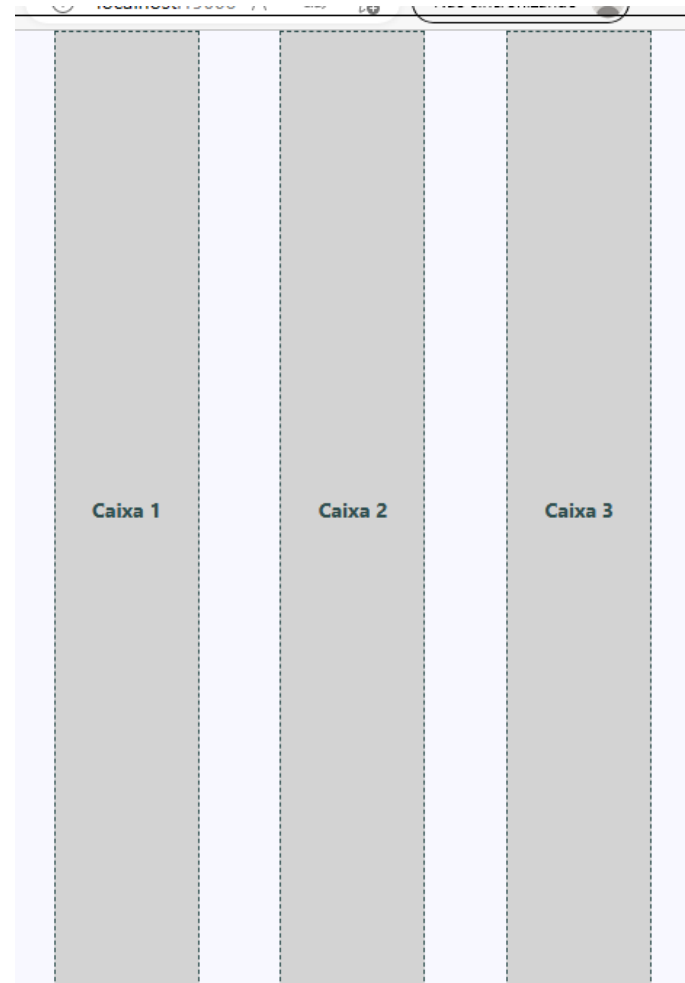


```
box: {  
  width: 100,  
  justifyContent: "center",  
  alignSelf: "stretch",  
  alignItems: "center",  
  backgroundColor: "lightgray",  
  borderWidth: 1,  
  borderStyle: "dashed",  
  borderColor: "darkslategray"  
},  
boxText: {  
  color: "darkslategray",  
  fontWeight: "bold"  
}  
});
```

Obs.: retirou-se, do box, a height.

React Native

Segundo Exemplo - Mudando Layout – Style – importando



React Native Terceiro Exemplo Layout - Grid



```
import React from "react";
import {View, StatusBar } from "react-native";
import styles from "./styles";
import Box from "./Box";

const boxes = new Array(12).fill(null).map((v, i) => i + 1);

export default function App() {
  return (
    <View style={styles.container}>
      <StatusBar hidden={false} />
      {boxes.map(i => (<Box key={i}>Caixa {i}</Box>))}
    </View>
  );
}
```

React Native Terceiro Exemplo Layout - Grid



```
import React from "react";
import PropTypes from "prop-types";
import { View, Text } from "react-native";
import styles from "./styles";

export default function Box({children}) {
  return (
    <View style={styles.box}>
      <Text style={styles.boxText}>{children}</Text>
    </View>
  );
}

Box.propTypes = {
  children: PropTypes.node.isRequired
};
```

React Native Terceiro Exemplo Layout - Grid



```
import { Platform, StyleSheet, StatusBar } from 'react-native';
```

```
export default StyleSheet.create({  
  container: {  
    flex: 1,  
    flexDirection: 'row',  
    flexWrap: 'wrap',  
    backgroundColor: 'ghostwhite',  
    alignItems: 'center',  
    ...Platform.select({  
      ios: { paddingTop: 20 },  
      android: { paddingTop: StatusBar.currentHeight }  
    })  
  },  
});
```


React Native Terceiro Exemplo Layout - Grid



```
box: {  
  height: 110,  
  width: 110,  
  justifyContent: 'center',  
  alignItems: 'center',  
  backgroundColor: 'lightgray',  
  borderWidth: 1,  
  borderStyle: 'dashed',  
  borderColor: 'darkslategray',  
  margin: 22  
},  
boxText: {  
  color: 'darkslategray',  
  fontWeight: 'bold'  
}});
```

React Native Terceiro Exemplo Layout - Grid



React Native Componente - **ActivityIndicator**



- Apresenta um indicador circular animado de carga;
- *Color, animating* e *size* são algumas das propriedades;
- Propriedades em:
 - <https://reactnative.dev/docs/activityindicator> (vamos visitar!)

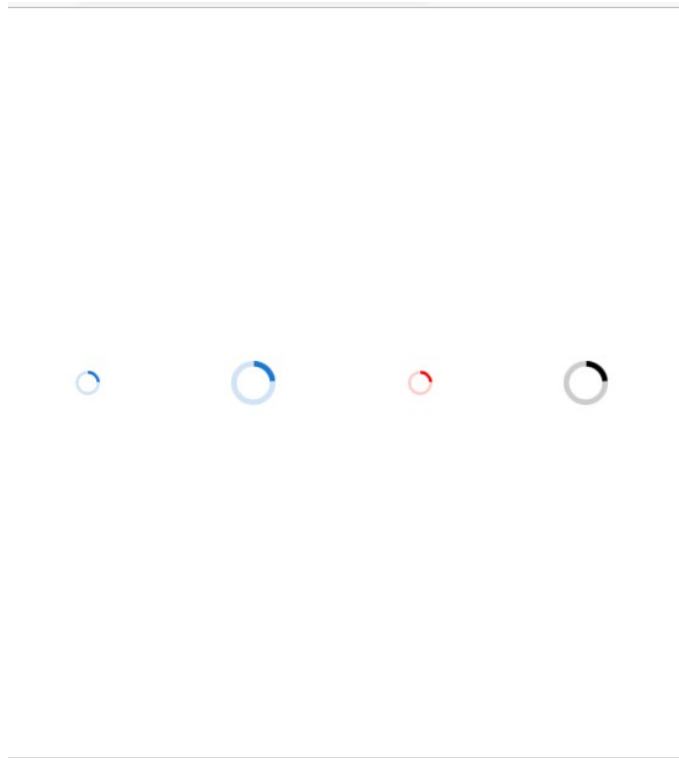
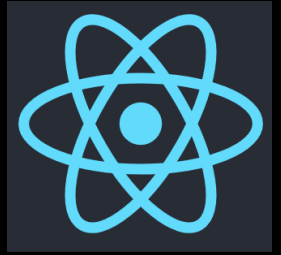
React Native Componente - **ActivityIndicator**



```
import React from 'react';
import {ActivityIndicator, StyleSheet, View} from 'react-native';
const App = () => (
  <View style={[styles.container, styles.horizontal]}>
    <ActivityIndicator />
    <ActivityIndicator size="large" />
    <ActivityIndicator size="small" color="#ff0000" />
    <ActivityIndicator size="large" color="#000000" />
  </View>
);
const styles = StyleSheet.create({
  container: {
    flex: 1,
    justifyContent: 'center', },
  horizontal: {
    flexDirection: 'row',
    justifyContent: 'space-around',
    padding: 10, },
});
export default App;
```

Criar e Executar o projeto ExemploActivityIndicator.

React Native Componente - `ActivityIndicator`



React Native Componente - Button



- Componente para implementar um botão básico;
- Oferece suporte para um nível mínimo de personalização;
- Propriedades *title* e evento *onPress()* são obrigatórias;
- Propriedades em:
 - <https://reactnative.dev/docs/button> (vamos visitar!)

React Native Componente - Button



```
import React, { Component } from 'react'
import { SafeAreaView, View, Text, Button, Alert, StyleSheet, StatusBar } from 'react-native'

const Separator = () => <View style={styles.separator} />;
const apertou = () => {Alert.alert('Clicou no Vermelho')};

const App = () => {
  return (
    <View style={styles.container}>
      <StatusBar hidden={false} />    // Depois de executar o exemplo, mudem para true
      <Text style={styles.title}>
        Exemplo com buttons!!!
      </Text>
      <Separator />
      <Button
        onPress={() => apertou()}    // Aqui, o Alert é por chamada de uma função
        title = "Botão Vermelho! Clique-me."
        color = "red"
      />
      <Separator />
      <Button
        onPress={() => Alert.alert('Clicou no Verde')} // Aqui, o Alert é chamado diretamente
        title = "Botão Verde! Clique-me."
        color = "green"
      />
    </View>
  );
};
```

React Native Componente - Button



```
const styles = StyleSheet.create({
  container: {
    flex: 1,
    marginHorizontal: 50,
    marginVertical: 40,
  },
  title: {
    textAlign: 'center',
    marginVertical: 8,
    fontSize: 20,
  },
  separator: {
    marginVertical: 8,
    //borderBottomColor: '#737373',
    //borderBottomWidth: StyleSheet.hairlineWidth,
  },
});
export default App;
```

Criar e Executar o projeto ExemploButton, no celular!

React Native Componente - Button



Exemplo com buttons!!!

BOTÃO VERMELHO! CLIQUE-ME.

BOTÃO VERDE! CLIQUE-ME.

React Native Componente - View



- Quando necessário agrupar elementos em um *container*, View pode ser o elemento recipiente;
- Quando desejar aninhar mais elementos dentro do elemento pai, ambos, pai e filho, podem ser uma View;
- Podem haver quantos filhos quiser;
- Quando quiser estilizar diferentes elementos, pode colocá-los dentro da View, pois ele suporta propriedade de estilo, flexbox etc.;
- View também suporta eventos de toque, que podem ser úteis para diferentes propósitos;
- Propriedades(são inúmeras!) em:
 - <https://reactnative.dev/docs/view> (vamos visitar!)

React Native Componente - View



```
import React from 'react';
import { StyleSheet, Text, View } from 'react-native';

export default function App() {
  return (
    <View
      style={{
        flexDirection: 'row',
        height: 250,
        padding: 10,
      }}>
      <View style={{backgroundColor: 'blue', flex: 0.3}} />
      <View style={{backgroundColor: 'red', flex: 0.4}} />
        <Text style={{fontSize:20,}}>Olha eu aí!</Text>
    </View>
  );
}
```

Criar e Executar projeto ExemploView no celular(no navegador o resultado é parcial).

React Native Componente - WebView



- É usado quando deseja-se renderizar uma página da *web* em um aplicativo móvel;
- Propriedades em:
 - <https://reactnative.dev/docs/0.61/webview> (vamos visitar!)
- Para instalar o pacote, no diretório do projeto(prompt do DOS):
 - `npm install --save react-native-webview`

React Native Componente - WebView



■ Código Exemplo:

```
import React, {Component} from 'react'
import {WebView} from 'react-native-webview'

const App = () => {
  return (
    <WebView
      source={{ uri: 'https://github.com/facebook/react-native',}}
      style={{marginTop: 30}}
      //onLoadEnd = {() => Alert.alert('Carregado!!!')} // Depois de executar o exemplo, teste estas instruções
      //onNavigationStateChange = {() => Alert.alert('Mudou algo!!!')}
    />
  )
};
export default App;
```

Criar e Executar o projeto ExemploWebView no celular, não roda sob o navegador.

React Native

Exercício A3



- Crie e execute um projeto com as seguintes funcionalidades:
 - Contém 2 botões, a saber:
 - Um, ao clicar, apresentará uma *webview*.
 - Outro, ao clicar, inibirá a apresentação.
 - A URL, por enquanto, é fixa e de livre escolha.

