

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 equivocado;
- O objetivo é ter componentes React Native em todos os lugares, e não escrever uma vez e executar em qualquer lugar;

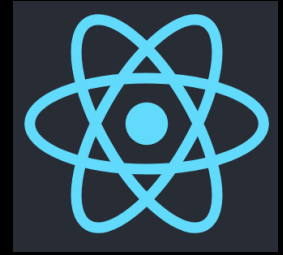
React Native

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;

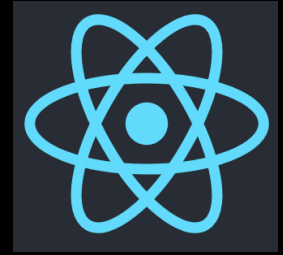
React Native Alguns Componentes



■ 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

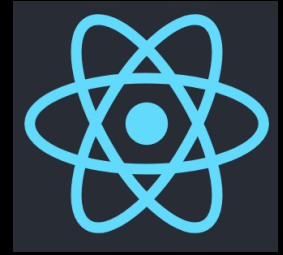
```
taxa = valor => {  
    fator = 3;  
    return fator * valor; // Obrigado a escrever o return quando na presença de {}  
}
```

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>;
```

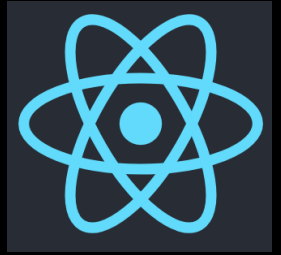
React Native

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

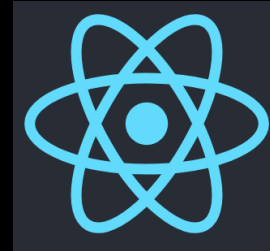


■ Classe:

```
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;
```

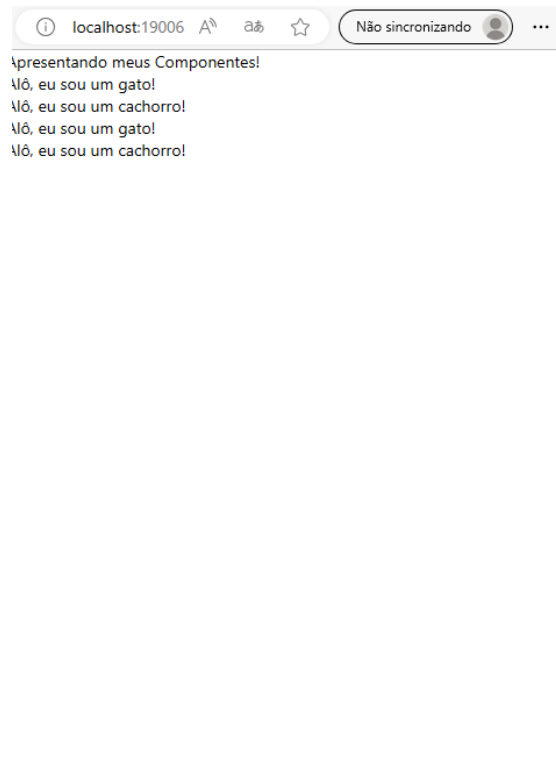
React Native component - revendo



```
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>
  );
};

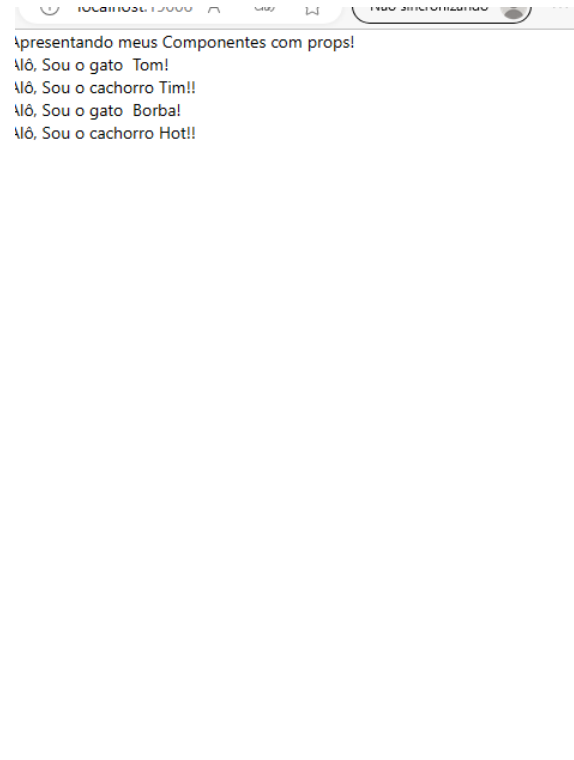
const Cachorro = (props: CachorroProps) => {
  return (
    <Text>Alô, Sou o cachorro {props.nome}</Text>
  );
};
```

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



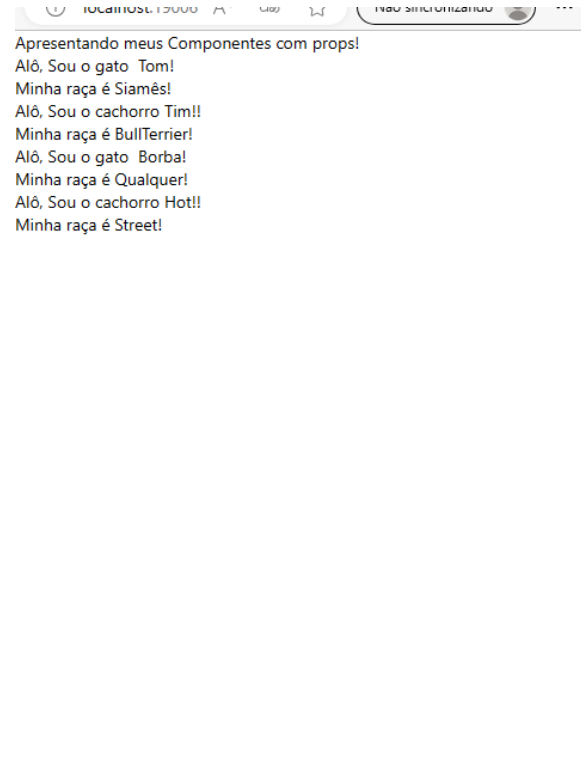
```
const Cachorro = (props:CachorroProps) => {  
  return (  
    <View>  
      <Text>Alô, Sou o cachorro {props.nome}!!</Text>  
      <Text>Minha raça é {props.raca}!</Text>  
    </View>  
  );  
};
```

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



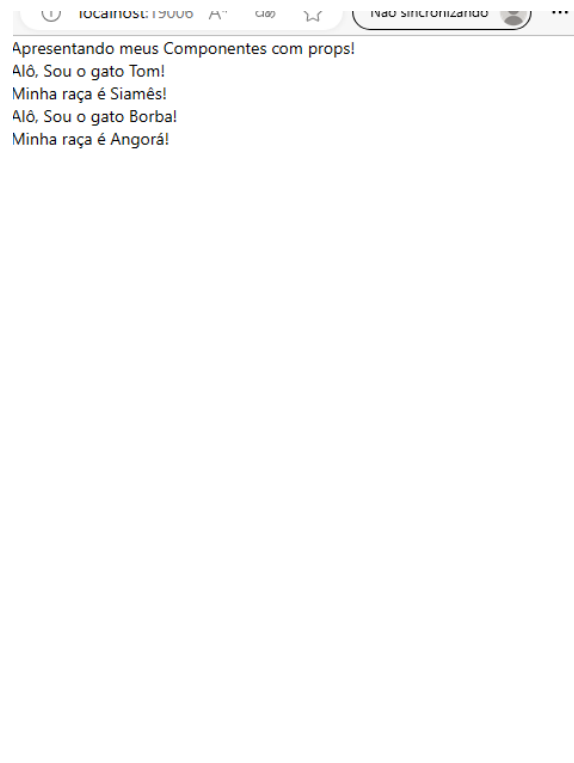
```
import React from 'react';
import {Text, View} from 'react-native';

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');

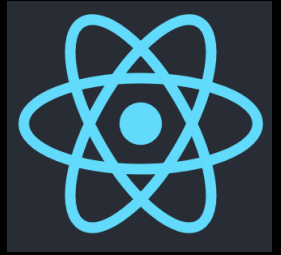
  return (
    <View>
      <Text>
        Eu sou o gato {nome}, e eu estou {taFaminto ? 'faminto' : 'satisfeito'}, minha raca é {raca} e
        minha cor é {cor}!
      </Text>
    </View>
  );
};
```

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



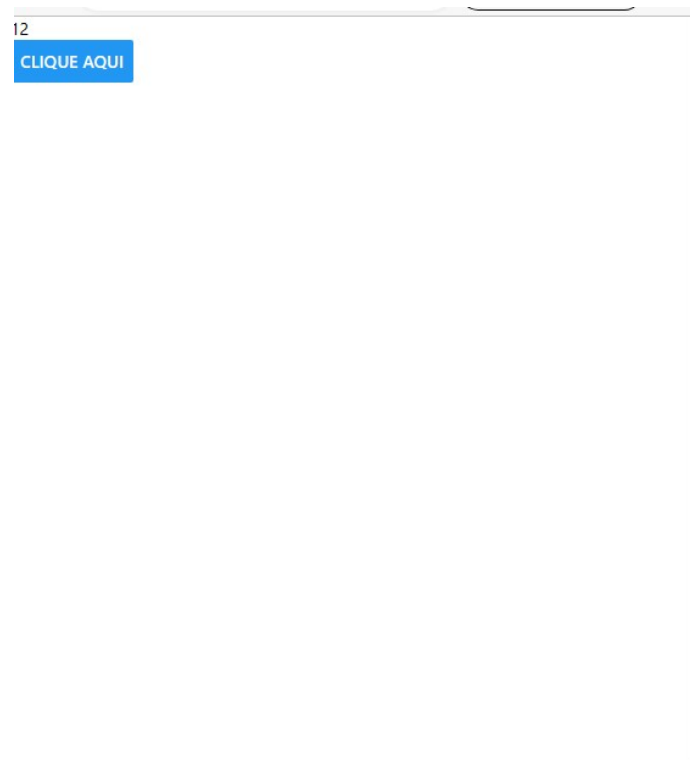
```
import React, { useState } from "react";
import { View, Text, Button } from "react-native";

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))`

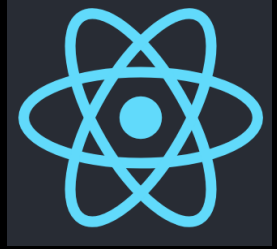
Executa a função em cada elemento do array.

`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 seleccionados 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]])`

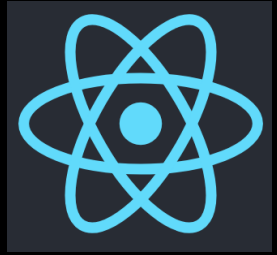
Adiciona elementos ao início do array.

`valueOf()`

Retorna o array em si.

Arrays

Alguns métodos - revendo



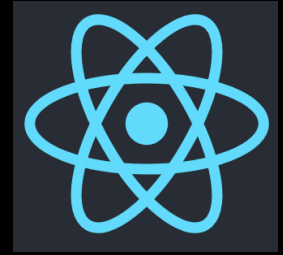
```
cores = ["Azul", "Verde", "Amarelo", "Vermelho", "Preto"];
carros = ["Astra", "Fiat 500", "Uno Way", "BMW", "Audi", "Fusca", "Citroen"];
notas = [5.5, 4.3, 8.5, 10.0, 9.1, 3.5, 6.1];

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

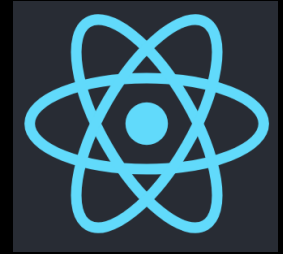


■ Criando o Primeiro Exemplo:














- 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

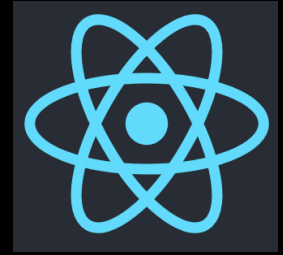
> Press j | open debugger
> Press r | reload app
> Press m | toggle menu

> 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

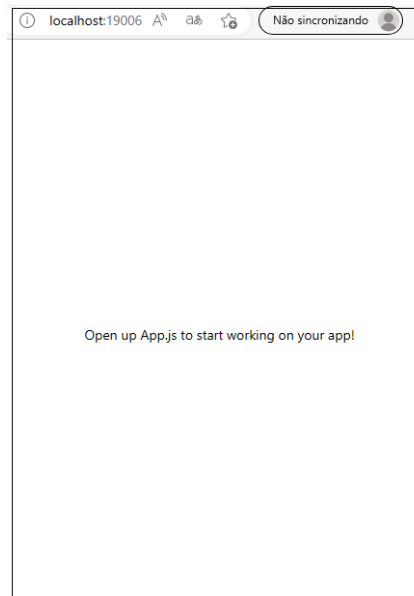
> Press a | open Android
> Press w | open web

> 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



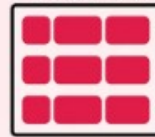
- Criado para para acomodar diferentes tamanhos de tela;
- Flexbox é o que nome diz: um modelo de caixa que é flexível;
- A caixa atua como um *container* que possui os elementos filhos dentro dessa caixa;
- Tanto o *container* quanto os elementos filho são flexíveis;
- Oferece três principais propriedades: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

React Native Layout - Flexbox



CSS Flexbox

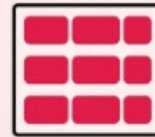
flex-direction



row



column

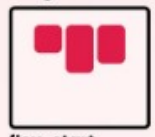


row-reverse

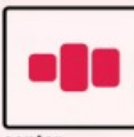


column-reverse

align-items



flex-start



center



flex-end



stretch

justify-content



flex-start



center



flex-end



space-between

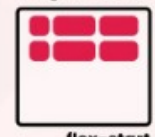


space-around

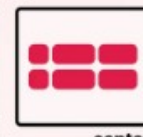


space-evenly

align-content



flex-start



center



flex-end



stretch



space-between



space-around

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

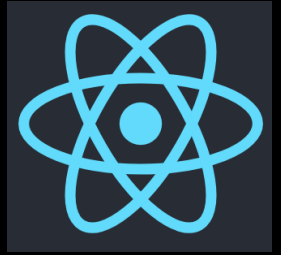


```
import React from "react";  
  
import { Text, View } from "react-native";  
  
import styles from "./styles";    // Aqui os estilos serão importados  
  
export default function App() {  
  return (  
    <View style={styles.container}>  
      <View style={styles.box}>  
        <Text style={styles.boxText}>Sou uma Caixa!!!</Text>  
      </View>  
    </View>  
  );  
}
```

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 }
```

```
  })
```

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

```
const styles = StyleSheet.create({
```

```
  container: {
```

```
    flex: 1,
```

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

```
      android: {
```

```
        backgroundColor: 'green',
```

```
      },
```

```
      ios: {
```

```
        backgroundColor: 'red',
```

```
      },
```

```
      default: {
```

```
        // other platforms, web for example
```

```
        backgroundColor: 'blue',
```

```
      },
```

```
    })),
```

```
  },
```

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"  
}  
}  
);
```

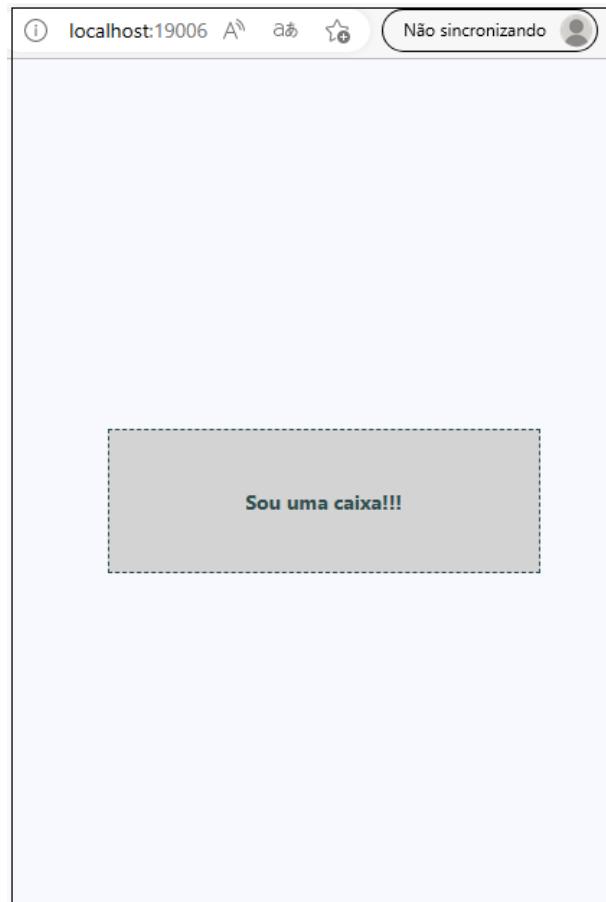
Obs.: Salvar o arquivo com o nome **styles**, do tipo js, no diretório raiz do aplicativo.

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";

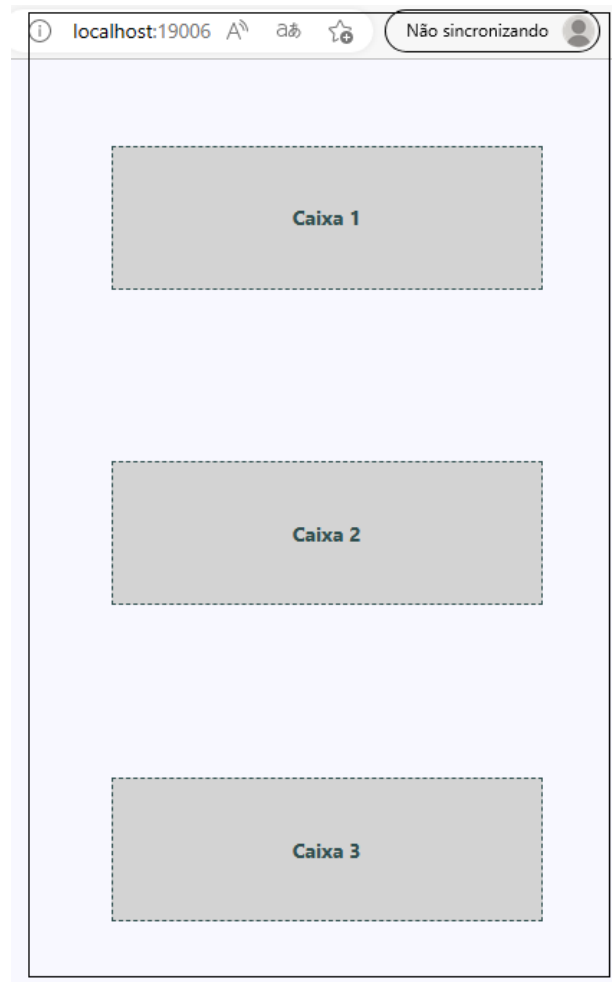
export default function App() {
  return (
    <View style={styles.container}>
      <View style={styles.box}>
        <Text style={styles.boxText}>#Caixa 1</Text>
      </View>
      <View style={styles.box}>
        <Text style={styles.boxText}>#Caixa 2</Text>
      </View>
      <View style={styles.box}>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

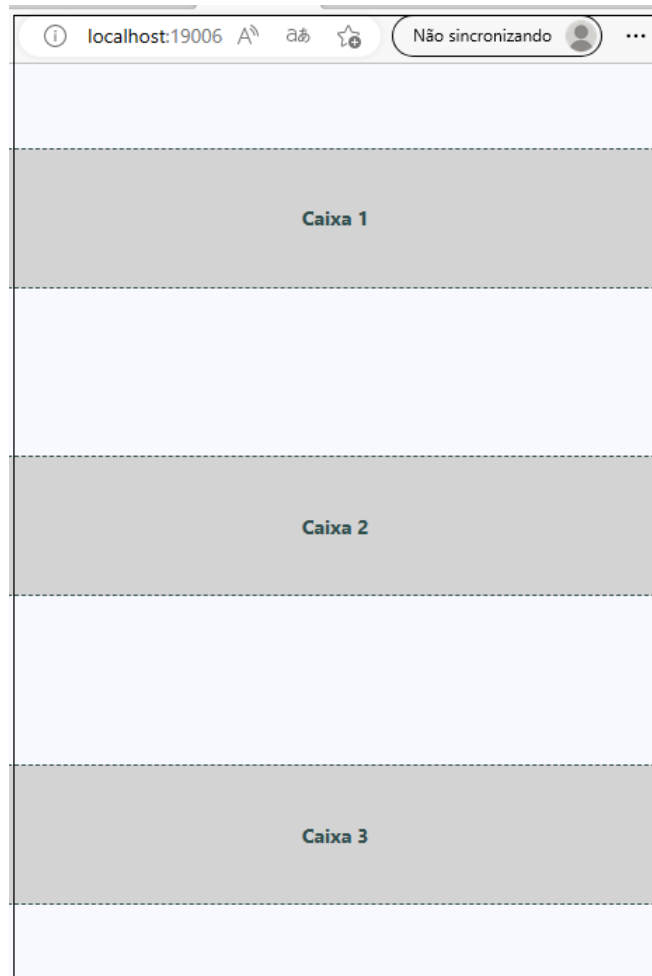


```
box: {  
  height: 100,  
  justifyContent: "center",  
  alignSelf: "stretch",  
  alignItems: "center",  
  backgroundColor: "lightgray",  
  borderWidth: 1,  
  borderStyle: "dashed",  
  borderColor: "darkslategray"  
},  
boxText: {  
  color: "darkslategray",  
  fontWeight: "bold"  
}  
});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

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

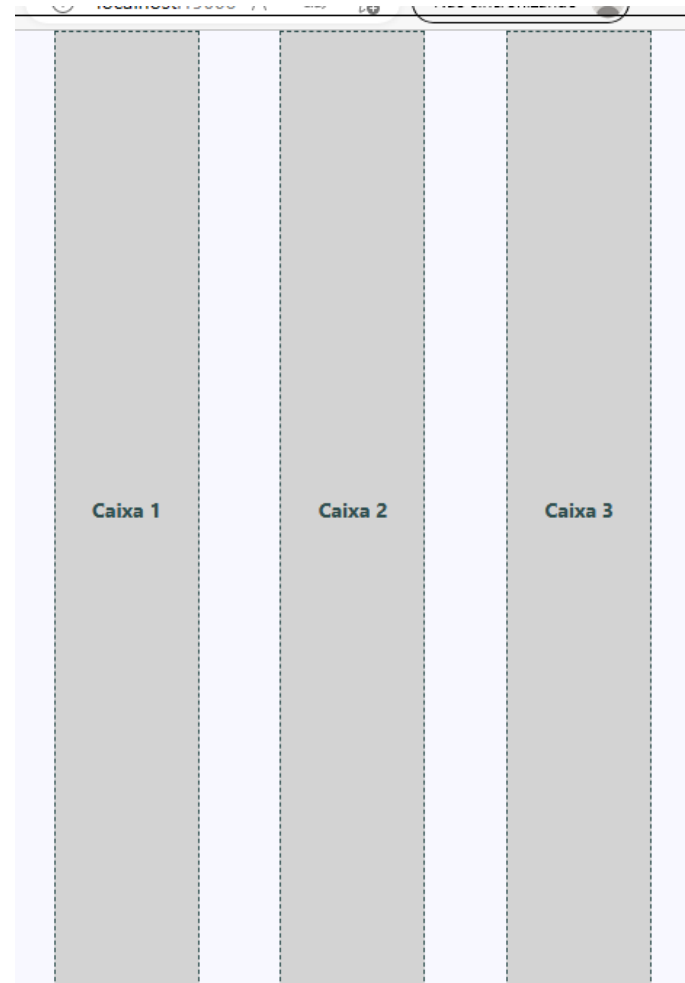


```
box: {  
  width: 100,  
  justifyContent: "center",  
  alignSelf: "stretch",  
  alignItems: "center",  
  backgroundColor: "lightgray",  
  borderWidth: 1,  
  borderStyle: "dashed",  
  borderColor: "darkslategray"  
},  
boxText: {  
  color: "darkslategray",  
  fontWeight: "bold"  
}  
});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

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!)

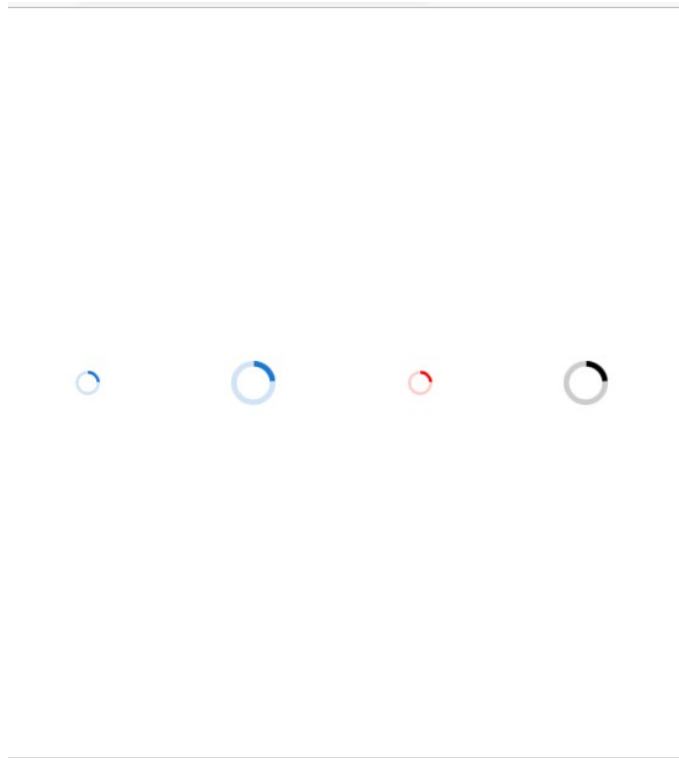
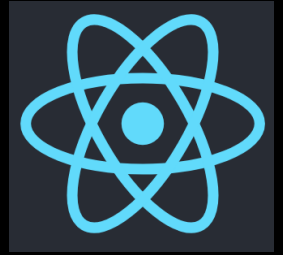
React Native Componente - **ActivityIndicator**



```
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):
 - `npx expo install react-native-webview@11.26.0`

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.
 - Até 25 minutos para concluir!

VER PÁGINA DA WEB.

INIBIR PÁGINA DA WEB.

React Native Componente - Modal



- Utilizado para apresentar um conteúdo em um plano acima de uma exibição prévia;
- Propriedades em:
 - <https://reactnative.dev/docs/modal> (vamos visitar!)

React Native Componente - Modal



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

const App = () => {
  const [modalVisible, setModalVisible] = useState(false);
  return (
    <View style={styles.centeredView}>
      <Modal
        animationType="fade"
        transparent={true}
        visible={modalVisible}
        onRequestClose={() => {
          Alert.alert('Modal foi fechado!');
          setModalVisible(!modalVisible);
        }}>
        <View style={styles.centeredView}>
          <View style={styles.modalView}>
            <Text style={styles.modalText}>React Native ON!</Text>
            <Button
              onPress={() => setModalVisible(!modalVisible)}
              title = "Esconder Modal"
              color = "red"
            />
          </View>
        </View>
      </Modal>
    </View>
  );
}
```


React Native Componente - Modal



```
<Button
  onPress={() => setModalVisible(!modalVisible)}
  title = "Mostrar Modal"
  color = "green"
/>
</View>
);
};
```

```
const styles = StyleSheet.create({
  centeredView: {
    flex: 1,
    justifyContent: 'center',
    alignItems: 'center',
    marginTop: 22,
  },
});
```

React Native Componente - Modal



```
modalView: {  
  margin: 20,  
  backgroundColor: 'white',  
  borderRadius: 20,  
  padding: 35,  
  alignItems: 'center',  
  shadowColor: '#000',  
  shadowOffset: {  
    width: 0,  
    height: 2,  
  },  
  shadowOpacity: 0.25,  
  shadowRadius: 4,  
  elevation: 5,  
},  
modalText: {  
  marginBottom: 15,  
  textAlign: 'center',  
},  
});  
  
export default App;
```

Criar e Executar o projeto ExemploModal, no celular (quando aparecer o modal, toque no Voltar de hardware).

O evento `onRequestClose` age quando o usuário toca no botão Voltar do hardware do Android ou no botão de menu na Apple TV.

React Native Componente - Modal



MOSTRAR MODAL

React Native ON!

ESCONDER MODAL

React Native Componente - Pressable



- É um componente que pode detectar vários estágios de interações de pressionamento em qualquer um de seus filhos definidos;
- Alguns eventos do Pressable:
 - `onPressIn` - chamado imediatamente quando um toque é acionado, antes de `onPressOut` e `onPress`.
 - `onPressOut` - chamado quando um toque é liberado.
 - `onLongPress` - chamado se o tempo após `onPressIn` durar mais de 500 milissegundos. Este tempo pode ser personalizado com `delayLongPress`.
 - `onPress` - chamado após o `onPressOut`.
 - `pressRetentionOffset` - distância adicional, fora da visualização, na qual um toque é considerado um pressionamento, antes que `onPressOut` seja acionado.

React Native Componente - Pressable



■ InteractionState:

- Reflete o estado atual da interação do usuário com a exibição.

■ Valores:

- focused: booleano
 - Se a exibição está com o foco, no momento.
- hovered: booleano
 - Se a exibição está sendo focalizada por um mouse.
- **pressed**: booleano
 - Se a exibição está sendo pressionada.

■ Propriedades em:

- <https://reactnative.dev/docs/pressable> (vamos visitar!)

React Native Componente - Pressable



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

const App = () => {
  const [timesPressed, setTimesPressed] = useState(0);

  let textLog = "";
  if (timesPressed > 1) {
    textLog = timesPressed + ' x Pressionado';
  } else if (timesPressed > 0) {
    textLog = 'Componente Pressable';
  }
  return (
    <View style={styles.container}>
      <Pressable
        onPress={() => {
          setTimesPressed(current => current + 1);
        }}
        style={({pressed}) => [
          {backgroundColor: pressed ? 'yellow' : 'pink'},
          styles.wrapperCustom,
        ]}
        onPressOut={() => Alert.alert('Soltou!')}>
        {{{pressed}}} => (
          <Text style={styles.text}>{pressed ? 'Pressionado!' : 'Pressione Me'}</Text>
        )
      </Pressable>
    </View>
  );
}
```

React Native Componente - Pressable



```
<View style={styles.logBox}>
  <Text>{textLog}</Text>
</View>
</View>
);
};

const styles = StyleSheet.create({
  container: {
    flex: 1,
    justifyContent: 'center',
    alignItems: 'center',
  },
  text: {
    fontSize: 16,
  },
  wrapperCustom: {
    borderRadius: 8,
    padding: 6,
    height: 50,
    width: '70%',
    justifyContent: 'center',
    alignItems: 'center',
    elevation: 5,
  },
});
```

React Native Componente - Pressable



```
logBox: {  
  padding: 20,  
  margin: 10,  
  borderWidth: StyleSheet.hairlineWidth,  
  borderColor: '#f0f0f0',  
  backgroundColor: '#f9f9f9',  
},  
});
```

```
export default App;
```

Criar e Executar o projeto ExemploPressable.

React Native Componente - Pressable



Pressione Me

11 x Pressionado

React Native Componente - Text



- Componente utilizado para exibição de textos;
- O texto suporta aninhamento, estilo e toque;
- Alguns eventos do Text:
 - `onPressIn` - chamado imediatamente quando um toque é acionado, antes de `onPressOut` e `onPress`.
 - `onPressOut` - chamado quando um toque é liberado.
 - `onResponderMove` – o usuário está movendo o dedo.
 - `onPress` – chamado após o `onPressOut`.
 - `selectable` – permitir ou não o copia e cola do texto.
- Propriedades em:
 - <https://reactnative.dev/docs/text> (vamos visitar!)

React Native Componente - Text



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

import TextExample from './Exemplo_Text.js'

const App = () => {

  return (

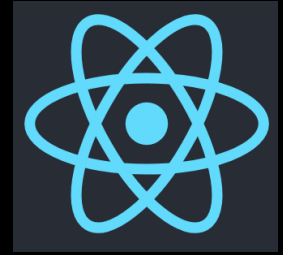
    <TextExample/>

  )

}

export default App
```

React Native Componente - Text



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

const apertou = ()=> {Alert.alert('Clicou no Texto')};
const TextExample = () => {
  return (
    <View style = {styles.container}>
      <Text style = {styles.text}>
        <Text style = {styles.capitalLetter}>
          exemplo de texts
          {\n}
        </Text>
        <Text onPress={() => apertou()}>
          orem ipsum dolor sit amet, sed do eiusmod.
          {\n}
        </Text>
        <Text>
          Ut enim ad <Text style = {styles.wordBold}>minim </Text> veniam,
          quis aliquip ex ea commodo consequat.
          {\n}
        </Text>
        <Text style = {styles.italicText}>
          Duis aute irure dolor in reprehenderit in voluptate velit esse cillum.
          {\n}
        </Text>
        <Text style = {styles.textShadow}>
          Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.
        </Text>
      </Text>
    </View>
  )
}
```

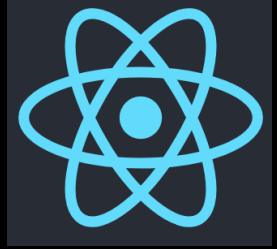
React Native Componente - Text



```
export default TextExample
const styles = StyleSheet.create ({
  container: {
    alignItems: 'center',
    marginTop: 100,
    padding: 20
  },
  text: {
    color: '#41cdf4',
  },
  capitalLetter: {
    color: 'red',
    fontSize: 20
  },
  wordBold: {
    fontWeight: 'bold',
    color: 'black'
  },
  italicText: {
    color: '#37859b',
    fontStyle: 'italic'
  },
  textShadow: {
    textShadowColor: 'red',
    textShadowOffset: { width: 2, height: 2 },
    textShadowRadius : 5
  }
})
```

Criar e Executar o projeto ExemploText.

React Native Composante - Text



exemple de texts

orem ipsum dolor sit amet, sed do eiusmod.

Ut enim ad **minim** veniam, quis aliquip ex ea commodo consequat.

Duis aute irure dolor in reprehenderit in voluptate velit esse cillum.

Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

React Native Componente - TextInput



- Componente utilizado para inserção de texto no aplicativo por meio de um teclado;
- Existem propriedades para vários recursos, como correção automática, capitalização automática, e diferentes tipos de teclado, como teclado numérico;
- Alguns eventos do TextInput:
 - `onPressIn` - chamado imediatamente quando um toque é acionado, antes de `onPressOut` e `onPress`.
 - `onPressOut` - chamado quando um toque é liberado.
 - `onResponderMove` - o usuário está movendo o dedo.
 - `onPress` - chamado após o `onPressOut`.
 - `autoCapitalize` - capitaliza automaticamente certos caracteres.
 - `autoComplete` - especifica dicas de preenchimento automático para o sistema.
 - `onChangeText` - função que é chamada quando o texto da entrada de texto é alterado. O texto alterado é passado como um único argumento de string para o manipulador de retorno de chamada.

Propriedades em:

React Native Componente - TextInput



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

const TextInputExample = () => {
  const [text,onChangeText] = React.useState("");
  const [number,onChangeNumber] = React.useState("");

  return (
    <View>
      <StatusBar hidden={false} />
      <Text style={styles.titulo}>
        Login:
      </Text>
      <TextInput
        style={styles.input}
        onChangeText={onChangeText}
        value={text}
        autoComplete='email'
        autoCapitalize='none'
      />
      <Text style={styles.titulo}>
        Senha:
      </Text>
```


React Native Componente - TextInput



```
<TextInput
  style={styles.input}
  onChangeText={onChangeNumber}
  value={number}
  placeholder=""
  keyboardType="numeric"
  secureTextEntry={true}
/>
</View>
);
};
```

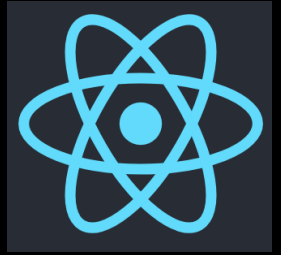
React Native Componente - TextInput



```
const styles = StyleSheet.create({
  input: {
    height: 40,
    margin: 12,
    borderWidth: 1,
    padding: 10,
  },
  titulo: {
    margin: 6,
    margin: 12,
    borderWidth: 1,
    padding: 10,
  },
  titulo: {
    margin: 6,
    padding: 5,
  },
});
export default TextInputExample;
```

Criar e Executar o projeto ExemploTextInput.

React Native Componente - TextInput



Login:

Senha:

React Native

Quinto Exemplo

TextInput + Button, com crítica



Login:

Senha:

SUBMETER

React Native Quinto Exemplo TextInput + Button, com crítica



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

```
const TextInputButtonExample = () => {
```

```
  const [text,onChangeText] = React.useState('');  
  const [number,onChangeNumber] = React.useState('');
```

```
  const apertou = () => {  
    if (!text.trim()) {  
      alert('Faltou Login');  
    }  
    if (!number.trim()) {  
      alert('Faltou Senha');  
    }  
    else{  
      alert('Partiu!!!');  
    }  
  }  
};
```

React Native Quinto Exemplo TextInput + Button, com crítica



```
return (  
  <View style={styles.container}>  
    <StatusBar hidden={false} />  
    <Text style={styles.titulo}>  
      Login:  
    </Text>  
    <TextInput  
      style={styles.input}  
      onChangeText={onChangeText}  
      value={text}  
      autoComplete = 'email'  
      autoCapitalize='none'  
    />  
    <Text style={styles.titulo}>  
      Senha:  
    </Text>  
    <TextInput  
      style={styles.input}  
      onChangeText={onChangeNumber}  
      value={number}  
      placeholder=""  
      keyboardType="numeric"  
      secureTextEntry={true}  
    />  
  </View>  
)
```

React Native Quinto Exemplo TextInput + Button, com crítica



```
<View style={{marginTop: 25}}>
  <Button
    onPress={apertou}
    title = "Submeter"
    color="#606070"
  />
</View>
</View>
);
};
const styles = StyleSheet.create({
  container: {
    flex: 1,
    padding: 35,
  },
  input: {
    width: '100%',
    height: 40,
    paddingHorizontal: 5,
    borderWidth: 0.5,
    marginTop: 15,
  }
});
```

React Native

Quinto Exemplo

TextInput + Button, com crítica



```
titulo: {  
  margin: 6,  
  padding: 5,  
  fontWeight: 'bold',  
},  
});  
export default TextInputButtonExample;
```

Criar e Executar o projeto QuintoExemplo

React Native

Exercício A4



- Alterar o Exercício A3 para que a URL seja fornecida através de um TextInput;
- Crítica básica na entrada(se preenchida ou não!);
- Tempo máximo para execução: 20 minutos!
- Instalar, no diretório do projeto(prompt do DOS);
 - `npx expo install react-native-webview@11.26.0`

URL:

VER PÁGINA DA WEB.

INIBIR PÁGINA DA WEB.

React Native Networking - Fetch/async/await



- Muitos aplicativos móveis precisam carregar recursos de uma URL remota;
- A API Fetch é a ferramenta padrão para executar operações de rede;
- Sintaxe:
 - `const nome_qualquer = await fetch(resource[,options]);`
 - `resource` – URL ou um objeto `Request`
 - `options` - opções de configuração como propriedades, método, cabeçalhos, corpo, credenciais e muito mais.
- O `fetch()` inicia uma requisição e retorna uma promessa;
- Quando a solicitação é concluída, a promessa é resolvida com um objeto `response`;
- Se a solicitação falhar, devido a problemas de rede, a promessa será rejeitada;
- A sintaxe `async/await` se encaixa bem com `fetch()` porque simplifica o trabalho com promessas;

React Native Networking - Fetch/async/await



■ Exemplo 01:

```
const fetchMoviesJSON = async () => {  
  try {  
    const response = await fetch('https://reactnative.dev/movies.json',);  
    const json = await response.json(); // A resposta é um objeto JSON  
    return json.movies;  
  } catch (error) {  
    console.error(error);  
  }  
};
```

```
fetchMoviesJSON().then(movies => { movies });
```

React Native Networking - Fetch/async/await



- O `fetch()` não lança um erro quando o servidor retorna um status HTTP sem sucesso, por exemplo erros de cliente (400–499) ou erros de servidor (500–599);
- O `fetch()` não lança um erro para um URL ausente(erro 404), e considera isso como uma solicitação HTTP concluída;
- O `fetch()` rejeita apenas se uma solicitação não pôde ser feita ou uma resposta que não pôde ser recuperada. Exemplo: problemas de rede, sem conexão com a Internet, *host* não encontrado, ou servidor não está respondendo;
- Para suprir esta necessidade existe a propriedade **response.ok** que permite distinguir os status de resposta HTTP boa da ruim;
- A propriedade é definida como **true** somente se a resposta tiver o status entre 200-299;

React Native Networking - Fetch/async/await



■ Exemplo 02:

```
const fetchMoviesJSON = async () => {  
  try {  
    const response = await fetch('https://reactnative.dev/movies.json');  
    const json = await response.json();  
  
    if (!response.ok) {  
      const message = 'Um erro: ' + {response.status};  
    }  
    else{  
      return json.movies;  
    }  
  } catch (error) {  
    console.error(error);  
  }  
};  
  
fetchMoviesJSON().then(movies => { movies });
```

React Native Networking - Fetch/async/await



■ Exemplo 03: Pegar dados complementares de um CEP.

CEP:

SUBMITER

Logradouro:

Bairro:

Localidade:

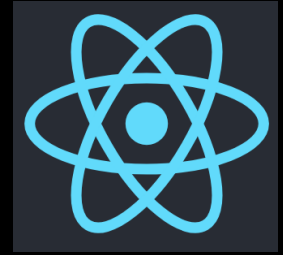
React Native Networking - Fetch/async/await



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

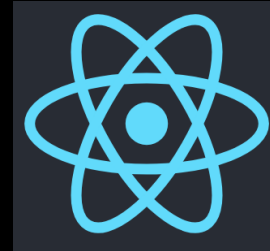
const ExemploPegaCEP = () => {
  const [cepnumber,onChangeNumber] = React.useState("");
  const [logradouro,onLogradouro] = React.useState("");
  const [bairro,onBairro] = React.useState("");
  const [localidade,onLocalidade] = React.useState("");
  const [encontrou,onEncontrou] = React.useState(true);
```

React Native Networking - Fetch/async/await



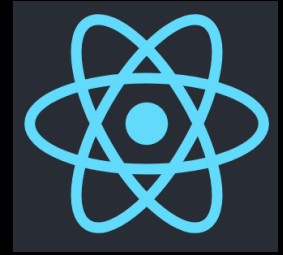
```
const PegaCEP = async () => {  
  try {  
    const response = await fetch('http://152.92.181.90:8080/CursoJavaScript/PegaCEPJson',{  
      method: 'POST',  
      headers: {  
        // Accept: 'application/json','Content-Type': 'application/json;charset=UTF-8',  
        'Content-type': 'application/x-www-form-urlencoded; charset=UTF-8',  
      },  
      body: JSON.stringify({cep:cepnumber})    // Tem que ser JSON !!!  
    });  
  
    if (!response.ok) {  
      alert('Algo não funcionou: ' + response.status);  
    }  
    else{  
      const json = await response.json();  
      onLogradouro(json.logradouro);  
      onBairro(json.bairro);  
      onLocalidade(json.localidade);  
      onEncontrou(json.encontrou);  
    }  
  } catch (error) {  
    console.error(error);  
  }  
};
```


React Native Networking - Fetch/async/await



```
const apertou = () => {  
  if (!cepnumber.trim()) {  
    alert('Tem que digitar!!!' );  
  }  
  else if (cepnumber.trim().length<8) {  
    alert('Tem que ter 8 dígitos!!!');  
  }  
  else{  
    PegaCEP();  
  }  
};
```

React Native Networking - Fetch/async/await



```
return (  
  <View style={styles.container}>  
    <StatusBar hidden={false} />  
    <Text style={styles.titulo}>  
      CEP:  
    </Text>  
    <TextInput  
      style={styles.input}  
      onChangeText={onChangeNumber}  
      value={cepnumber}  
      keyboardType="numeric"  
      autoCapitalize='none'  
    />  
    <View style={{marginTop: 25}}>  
      <Button  
        onPress={apertou}  
        title = "Submeter"  
        color="#606070"  
      />  
    </View>  
    <Text style={styles.titulo}>  
      Logradouro: {encontrou ? logradouro : '?'}  
    </Text>  
    <Text style={styles.titulo}>  
      Bairro: {encontrou ? bairro : '?'}  
    </Text>  
    <Text style={styles.titulo}>  
      Localidade: {encontrou ? localidade : '?'}  
    </Text>  
  </View>  
)
```

React Native Networking - Fetch/async/await

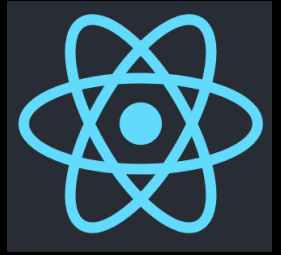


```
const styles = StyleSheet.create({
  container: {
    flex: 1,
    padding: 35,
  },
  input: {
    width: '100%',
    height: 40,
    paddingHorizontal: 5,
    borderWidth: 0.5,
    marginTop: 15,
  },
  titulo: {
    margin: 6,
    padding: 5,
    fontWeight: 'bold',
  },
});
```

```
export default ExemploPegaCEP;
```

Criar e Executar o projeto ExemploPegaCEP.

React Native Networking - Fetch/async/await



CEP:

22785275

SUBMITER

Logradouro: Estrada dos Bandeirantes

Bairro: Vargem Grande

Localidade: Rio de Janeiro

React Native API Alert



- Inicia uma caixa de diálogo de alerta com um título e uma mensagem;
- Opcionalmente, fornece uma lista de botões;
- Tocar em qualquer botão acionará o respectivo retorno de chamada onPress e descartará o alerta;
- Por padrão, o único botão será um botão 'OK';
- Esta API funciona tanto no Android quanto no iOS;
- O alerta, que permite ao usuário inserir algumas informações, está disponível apenas no iOS;
- No iOS, pode-se especificar qualquer número de botões;
- No Android, no máximo três botões podem ser especificados. O Android tem um conceito de botão neutro, negativo e positivo;
- Ao especificar um botão, será o 'positivo' (como 'OK'), dois botões significam 'negativo', 'positivo' (como 'Cancelar', 'OK'), três botões significam 'neutro', 'negativo', 'positivo' (como 'Mais tarde', 'Cancelar', 'OK');

React Native API Alert



- Os alertas no Android podem ser descartados tocando fora da caixa de alerta;
- Informações sobre como usar em:
 - <https://reactnative.dev/docs/alert> (vamos visitar!)

React Native API Alert – Primeiro Exemplo



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

const showAlert = () =>
  Alert.alert('Título do Alerta', 'Texto da Mensagem do Alerta', [
    {
      text: 'Cancelar',
      onPress: () => Alert.alert('Cancelar Pressionado'),
      style: 'cancel',
    },
  ],
  {
    cancelable: true,
    onDismiss: () =>
      Alert.alert(
        'Este alerta foi descartado tocando fora da caixa de diálogo de alerta.',
      ),
  },
  );
```

React Native API Alert – Primeiro Exemplo



```
const App = () => (  
  <View style={styles.container}>  
    <Button title="Mostrar Alerta" onPress={showAlert} />  
  </View>  
);  
  
const styles = StyleSheet.create({  
  container: {  
    flex: 1,  
    justifyContent: 'center',  
    alignItems: 'center',  
  },  
});  
  
export default App;
```

Criar e Executar o projeto ExemploAlert, no celular.

React Native API Alert – Segundo Exemplo



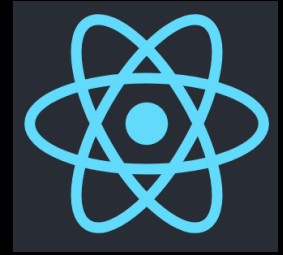
```
import React, { Component } from 'react';
import { Alert, Button, View, StyleSheet } from 'react-native';

export default class App extends Component { // Usou classe em vez de função

  openAlert={()=>{
    alert('Alerta com 1 botão');
  }}

  openTwoButtonAlert={()=>{
    Alert.alert('Alô você!', 'Dois botões', [
      {text: 'Sim', onPress: () => Alert.alert('Botão Sim Clicado')},
      {text: 'Não', onPress: () => Alert.alert('Botão Não Clicado'), style: 'cancel'},
    ],
    {
      cancelable: true
    }
  );
}}
```

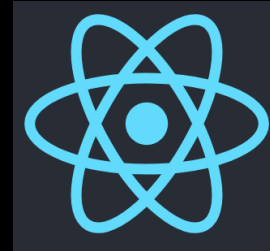
React Native API Alert – Segundo Exemplo



```
openThreeButtonAlert={()=>{
  Alert.alert(
    'Alô você!!', 'Três botões', [
      {text: 'Mais Tarde', onPress: () => Alert.alert('Botão Mais Tarde Clicado')},
      {text: 'Sim', onPress: () => Alert.alert('Botão Sim Clicado')},
      {text: 'Não', onPress: () => Alert.alert('Botão Não Clicado')},
    ],
    {
      cancelable: false
    }
  );
}

render() {
  return (
    <View style={styles.mainWrapper}>
      <Button title='1 Botão' onPress={this.openAlert}/>
      <Button title='2 Botões' onPress={this.openTwoButtonAlert}/>
      <Button title='3 Botões' onPress={this.openThreeButtonAlert}/>
    </View>
  );
}
```

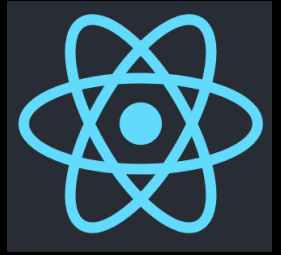
React Native API Alert – Segundo Exemplo



```
const styles = StyleSheet.create({  
  mainWrapper: {  
    flex: 1,  
    alignItems: 'center',  
    justifyContent: 'center',  
    flexDirection: 'row',  
  }  
});
```

Criar e Executar o projeto ExemploAlert02, no celular.

React Native API Alert – Segundo Exemplo



1 BOTÃO 2 BOTÕES 3 BOTÕES

React Native List Views



- O React Native possui os componentes FlatList e SectionList para apresentar uma listas de dados;
- O FlatList exibe uma lista de rolagem de dados estruturados de forma semelhante;
- O FlatList funciona bem para longas listas de dados, onde o número de itens pode mudar ao longo do tempo;
- Ao contrário do ScrollView mais genérico, o FlatList renderiza apenas os elementos que estão sendo exibidos na tela, e não todos os elementos de uma vez;
- O FlatList requer dois props: **data** e **renderItem**;
- **Data** é a fonte de informação para a lista e **renderItem** pega um item da fonte e retorna um componente formatado para renderizar;

React Native List Views



- O `SectionList` permite apresentar uma lista seccionada em sua interface do usuário;
- O componente `ListView` foi depreciado;

React Native Componente - FlatList



- O componente FlatList é usado para renderizar grandes quantidades de conteúdo rolável;
- O FlatList renderiza cada item de um array de dados de entrada, usando a propriedade renderItem;
- A propriedade renderItem pega um item do array de dados e mapeia-o para um elemento React;
- Cada item de dados deve ser um objeto com um **id único**, para que o React possa determinar quando os itens forem reorganizados;
- Propriedades e métodos em:
 - <https://reactnative.dev/docs/flatlist> (vamos visitar!)

React Native Componente - FlatList



■ Sintaxe:

```
<FlatList data={} renderItem={} [keyExtractor={item => item.id}] />
```

data: é ao array de itens que deseja-se passar para o FlatList.

renderItem: é uma função que pega cada objeto item da prop **data** e os renderiza no componente list.

keyExtractor: esta prop é usada para extrair a chave exclusiva de um determinado item. Caso o array contenha o campo **key** ou **id**, não precisa incluir esta prop. Por padrão, FlatList procurará **key** ou **id**.

React Native props - FlatList



ItemSeparatorComponent: Este componente é usado para renderizar um espaço entre cada item.

ListEmptyComponent: Este componente é exibido quando a lista está vazia.

ListFooterComponent: Este componente é exibido na parte inferior de todos os itens.

ListFooterComponentStyle: É usado para estilizar a visão interna do ListFooterComponent.

ListHeaderComponent: Este componente é exibido na parte superior de todos os itens.

ListHeaderComponentStyle: É usado para estilizar a exibição interna ListHeaderComponent.

columnWrapperStyle: Este é um estilo personalizado usado para linhas de vários itens.

extraData: Esta é a propriedade que instrui a lista a se renderizar novamente.

getItemLayout: Esta é uma otimização opcional que permite pular a medição de conteúdo dinâmico se você souber o tamanho dos itens.

horizontal: Se true, os itens serão renderizados horizontalmente em vez de verticalmente.

initialNumToRender: especifica o número de itens a serem renderizados no primeiro lote.

React Native Componente – FlatList props



initialScrollIndex: Se fornecido, começará a partir do item initialScrollIndex em vez do item superior.

inverted: Inverte a direção do scroll.

numColumns: Esta propriedade é usada para exibir várias colunas.

onEndReached: Este prop é chamado apenas uma vez quando a posição de rolagem está dentro do conteúdo renderizado. Em combinação com onEndReachedThreshold pode ser usado para obter uma paginação de listas.

onEndReachedThreshold: Este prop nos diz o quão perto estamos do fim.

onRefresh: Se este parâmetro for fornecido, um RefreshControl padrão será adicionado.

onViewableItemsChanged: Este prop é invocado quando a visibilidade de uma linha muda.

progressViewOffset: é definido quando o deslocamento de carregamento é necessário. Está disponível apenas para dispositivos Android.

progressViewOffset: é definido quando o deslocamento de carregamento é necessário. Está disponível apenas para dispositivos Android.

refreshing: enquanto espera por novos dados de uma atualização, defina isso como verdadeiro.

removeClippedSubviews: Isso pode melhorar o desempenho de rolagem ao rolar por listas longas. O valor padrão no Android é verdadeiro.

viewabilityConfigCallbackPairs: exibe uma lista de pares.

React Native Componente - FlatList



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

const DATA = [ {
  id: 'bd7acbea-c1b1-46c2-aed5-3ad53abb28ba',
  title: 'Primeiro Item',
},
{
  id: '3ac68afc-c605-48d3-a4f8-fbd91aa97f63',
  title: 'Segundo Item',
},
{
  id: '58694a0f-3da1-471f-bd96-145571e29d72',
  title: 'Terceiro Item',
},
];

type ItemProps = {title: string};
const Item = ({title}: ItemProps) => (
  <View style={styles.item}>
    <Text style={styles.title}>{title}</Text>
  </View>
);
```

React Native Componente - FlatList



```
const App = () => {  
  return (  
    <SafeAreaView style={styles.container}>  
      <FlatList  
        data={DATA}  
        renderItem={({item}) => <Item title={item.title} />} // Código da renderização aqui  
        keyExtractor={item => item.id} // Desnecessário  
      />  
    </SafeAreaView>  
  );  
};  
  
const styles = StyleSheet.create({  
  container: {  
    flex: 1,  
    marginTop: StatusBar.currentHeight || 0,  
  },  
});
```

React Native Componente - FlatList



```
item: {  
  backgroundColor: '#ffcd5d',  
  padding: 20,  
  marginVertical: 1,  
  marginHorizontal: 16,  
},  
title: {  
  fontSize: 32,  
},  
});
```

```
export default App;
```

Criar e Executar o Projeto ExemploFlatList01.

React Native Componente - FlatList



Primeiro Item

Segundo Item

Terceiro Item

React Native Componente – FlatList Outro exemplo



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

const Item = ({name}) => {
  return(
    <View style={styles.item}>
      <Text style={{color: 'black'}}>{name}</Text>
    </View>
  );
}

export default function App() {

  const countries = [
    {
      id: '1',
      name: 'Brasil',
    },
    {
      id: '2',
      name: 'Uruguai',
    },
    {
      id: '3',
      name: 'Argentina',
    },
    {
      id: '4',
      name: 'Paraguai',
    },
    {
      id: '5',
      name: 'Chile',
    },
  ]
}
```

React Native Componente – FlatList Outro exemplo



```
const renderItem = ({item})=>(
  <Item name={item.name}/>
);
return (
  <View style={styles.container}>
    <FlatList
      data={countries}
      renderItem={renderItem}    // Invoca uma função para renderizar
      keyExtractor={(item) => item.id}  // Desnecessário
    />
  </View>
);
}
```

```
const styles = StyleSheet.create({
  container: {
    flex: 1,
    marginTop:30,
    padding:2,
  },
});
```

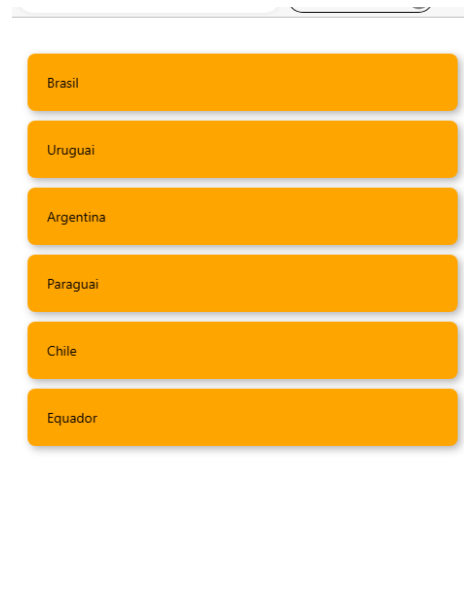

React Native Componente - FlatList Outro exemplo



```
item: {  
  backgroundColor: 'orange',  
  padding: 20,  
  marginVertical: 5,  
  marginHorizontal: 16,  
  borderRadius: 8,  
  shadowColor: '#000',  
  shadowOffset: { width: 3, height: 3 },  
  shadowOpacity: 0.3,  
  shadowRadius: 8,  
},  
});
```

Criar e Executar o Projeto ExemploFlatList02.

React Native Componente - FlatList



React Native Componente – FlatList onEndReached - onEndReachedThreshold

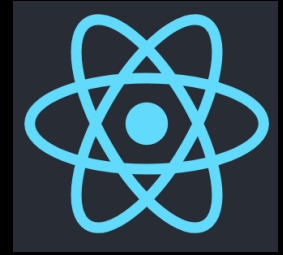


```
import React from 'react';  
import {SafeAreaView, Text, StyleSheet, View, FlatList} from 'react-native';
```

```
const DATA = [  
  { id: "1",  
    title: "Faraday",  
  },  
  { id: "2",  
    title: "Isaac Newton",  
  },  
  { id: "3",  
    title: "LaPlace",  
  },  
  { id: "4",  
    title: "Volta",  
  },  
  { id: "5",  
    title: "Marie Curie",  
  },  
  { id: "6",  
    title: "Edison",  
  },  
  { id: "7",  
    title: "Benjamin Franklin",  
  },  
  { id: "8",  
    title: "Einstein",  
  },  
  { id: "9",  
    title: "Lattes".  
}
```

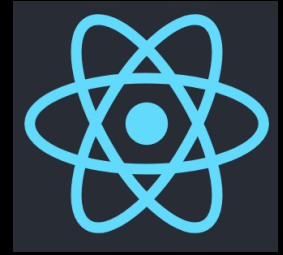
React Native Componente – FlatList

onEndReached - onEndReachedThreshold



```
{ id: "10",  
  title: "Da Vinci",  
},  
{ id: "11",  
  title: "Galileu",  
},  
{ id: "12",  
  title: "Copérnico",  
},  
{ id: "13",  
  title: "Tesla",  
},  
{ id: "14",  
  title: "Sabin",  
},  
{ id: "15",  
  title: "Darwin",  
},  
{ id: "16",  
  title: "Fourier",  
},  
{ id: "17",  
  title: "Karnaugh",  
},  
{ id: "18",  
  title: "Stephen Hawking",  
},  
{ id: "19",  
  title: "Arquimedes",  
},  
{ id: "20",  
  title: "Leonardo da Vinci",  
},
```

React Native Componente – FlatList onEndReached - onEndReachedThreshold



```
const App = () => {

  const ItemView = ({item}) => {
    return (
      <Text style={styles.itemStyle}
        onPress={() => alert(item.id)}      // Quando clicar no item, alerta!!!
      >
        {item.id}{ ' - '}{item.title.toUpperCase()}
      </Text>
    );
  };

  const chegouAoFim = () => {
    alert("Acabouuuu!");
  }

  return (
    <SafeAreaView style={{flex: 1}}>
      <View style={styles.container}>
        <FlatList
          data={DATA}
          keyExtractor={item => item.id}
          renderItem={ItemView}
          onEndReached={chegouAoFim}
          onEndReachedThreshold={0.1}
        />
      </View>
    </SafeAreaView>
  );
}
```

React Native Componente – FlatList onEndReached - onEndReachedThreshold



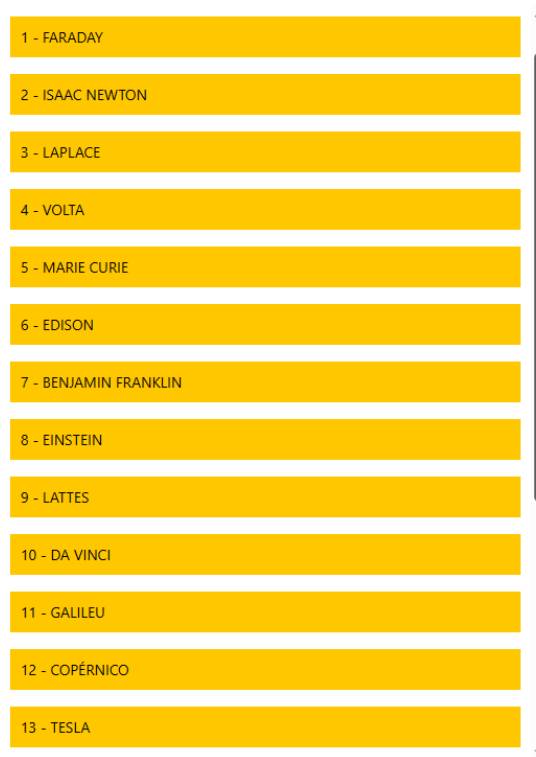
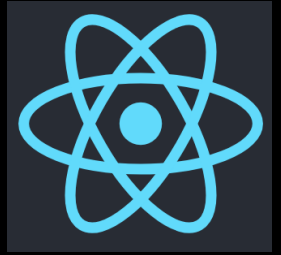
```
const styles = StyleSheet.create({
  container: {
    margin: 10,
    paddingTop: 40,
    backgroundColor: 'white',
  },
  itemStyle: {
    backgroundColor: '#ffc700',
    padding: 10,
    marginVertical: 8,
    color: 'black',
  },
});

export default App;
```

Criar e Executar o Projeto ExemploFlatList06.

React Native Componente – FlatList

onEndReached - onEndReachedThreshold



React Native

Exercício A5



- Alterar o Exemplo Anterior para que quando o usuário clicar em algum item, seja apresentado um modal com o id e o nome do cientista;
- Tempo máximo para execução: 15 minutos!

React Native Hook - useEffect



- O useEffect Hook pode ser visto como as funcionalidades `componentDidMount`, `componentDidUpdate` e `componentWillUnmount` combinadas;
- Por padrão, `useEffect` é executado após a primeira renderização e após cada atualização;
- Sintaxe:
 - `useEffect(<função>[, <dependência>])`
- Ao fornecer uma dependência, a função será executada a cada atualização. Exemplo:

```
const [marcaCarro, setMarcaCarro] = useState("");
```

```
useEffect(() => {  
  PegaCarros()  
}, [marcaCarro])
```

Como o `useState` é assíncrono, o `useEffect` garantirá que a função `PegaCarros` somente será invocada após a atualização do valor de `marcaCarro`.

React Native Componente – FlatList Consumindo end point



```
import React, {useEffect, useState} from 'react';
import {ActivityIndicator, FlatList, Text, View, Alert, StyleSheet} from 'react-native';

const App = () => {
  const [isLoading, setLoading] = useState(true);
  const [data, setData] = useState(null);
  const [title, setTitle] = useState("");
  const [rodape, setRodape] = useState("");

  const getMovies = async () => {
    try {
      const response = await fetch('https://reactnative.dev/movies.json'); // Não testou o response!!!
      const json = await response.json();
      setData(json.movies);
      setTitle(json.title);
      setRodape(json.description);
    } catch (error) {
      console.error(error);
    } finally {
      setLoading(false);
    }
  }
}
```

React Native Componente - FlatList Consumindo end point



```
const Titulo = () => {  
  return (  
    <View style={styles.headerFooterStyle}>  
      <Text style={styles.textStyle}>  
        {title}  
      </Text>  
    </View>  
  );  
};  
  
const Rodape = () => {  
  return (  
    <View style={styles.headerFooterStyle}>  
      <Text style={styles.textStyle}>  
        {rodape}  
      </Text>  
    </View>  
  );  
};  
  
useEffect(() => {  
  getMovies();  
}, []);
```

React Native Componente - FlatList Consumindo end point



```
return (  
  <View style={styles.container}>  
    {isLoading ? (  
      <ActivityIndicator />  
    ) : (  
      <FlatList  
        ListHeaderComponent={Titulo}  
        ListFooterComponent={Rodape}  
        data={data}  
        keyExtractor={({id}) => id}  
        renderItem={({item}) => (  
          <Text style={styles.itemStyle}>  
            {item.title}, {item.releaseYear}  
          </Text>  
        )}  
      />  
    )}  
  </View>  
);  
};
```

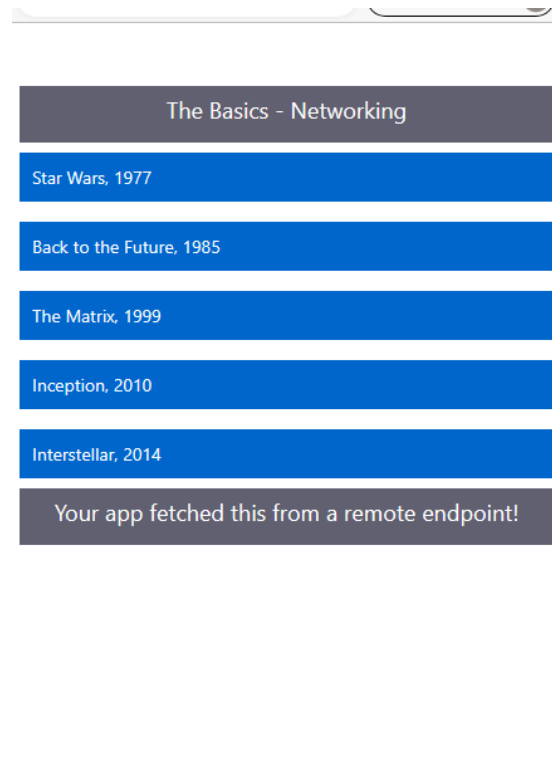
React Native Componente – FlatList Consumindo end point



```
const styles = StyleSheet.create({
  container: {
    flex:1,
    margin: 10,
    paddingTop: 40,
    backgroundColor: 'white',
  },
  itemStyle: {
    backgroundColor: '#0066CC',
    padding: 10,
    marginVertical: 8,
    color: 'white',
  },
  headerFooterStyle: {
    width: '100%',
    height: 45,
    backgroundColor: '#606070',
  },
  textStyle: {
    textAlign: 'center',
    color: 'fff',
    fontSize: 18,
    padding: 7,
  },
});

export default App;
```

React Native Componente – FlatList Consumindo end point



React Native

Exercício A6



- Crie e execute um projeto que quando o usuário clicar em algum dos botões abaixo, é apresentada uma relação de alguns carros da marca selecionada;
- url = "http://152.92.181.90:8080/CursoJavaScript/EnviarMenuJson";
- Método: POST;
- O *body* deve conter a informação: marca=marca selecionada;
- Até 25 minutos para concluir!

Fiat

GM

VW

Ford

React Native

Exercício A6



■ Retornará um objeto JSON, Carros, conforme o exemplo abaixo:

```
{"msg":"Operação realizada com sucesso!","Carros":  
[{"marca":"Ford","ano":2011,"cor":"Cinza","id":12,"modelo":"Ka"},  
{"marca":"Ford","ano":2021,"cor":"Preta","id":13,"modelo":"Taurus"},  
{"marca":"Ford","ano":2003,"cor":"Vermelha","id":14,"modelo":"EcoSport"}],"erro":false}
```


React Native Componente - Image



- O componente Image é usado para exibir diferentes tipos de imagens, incluindo imagens da web, imagens locais temporárias e imagens do disco local;
- Propriedades e métodos em:
 - <https://reactnative.dev/docs/image> (vamos visitar!)

React Native Componente - Image



```
import React from 'react';  
  
import {View, Image, StyleSheet} from 'react-native';  
  
const styles = StyleSheet.create({  
  container: {  
    paddingTop: 50,  
  },  
  tinyLogo: {  
    width: 50,  
    height: 50,  
  },  
  logo: {  
    width: 66,  
    height: 58,  
  },  
});
```

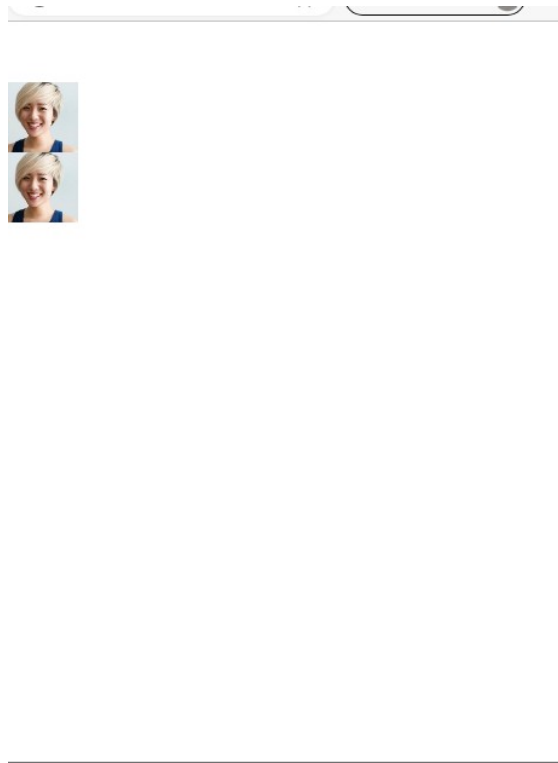
React Native Componente - Image



```
const DisplayAnImage = () => {  
  return (  
    <View style={styles.container}>  
      <Image  
        style={styles.logo}  
        source={{uri:"https://randomuser.me/api/portraits/women/44.jpg",  
          }}  
      />  
      <Image  
        style={styles.logo}  
        source={{uri:"https://randomuser.me/api/portraits/women/44.jpg",  
          }}  
      />  
    </View>  
  );  
};  
  
export default DisplayAnImage;
```

Criar e Executar o Projeto ExemploImage01.

React Native Componente - Image



React Native Componente - ImageBackground



- O componente ImageBackground tem as mesmas propriedades do componente <Image>, e é utilizado para apresentar uma imagem de fundo;
- Propriedades e métodos em:
 - <https://reactnative.dev/docs/imagebackground> (vamos visitar!)

React Native Componente - ImageBackground



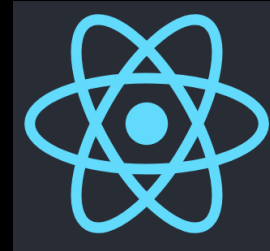
```
import React from 'react';
import {ImageBackground, StyleSheet, Text, View} from 'react-native';

const image = {uri: 'https://reactjs.org/logo-og.png'};

const App = () => (
  <View style={styles.container}>
    <ImageBackground source={image} resizeMode="cover" style={styles.image}>
      <Text style={styles.text}>Inside</Text>
    </ImageBackground>
  </View>
);

const styles = StyleSheet.create({
  container: {
    flex: 1,
  },
  image: {
    flex: 1,
    justifyContent: 'center',
  },
});
```

React Native Componente - ImageBackground

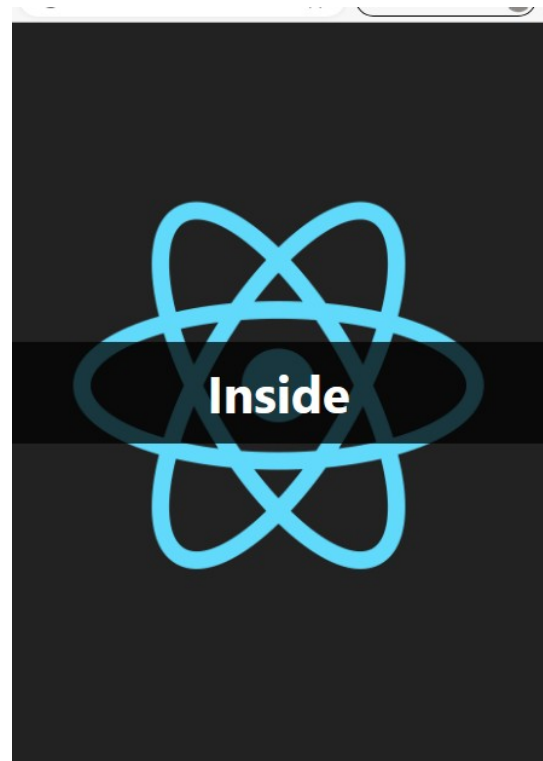
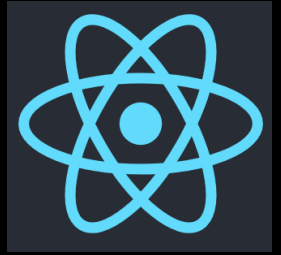


```
text: {  
  color: 'white',  
  fontSize: 42,  
  lineHeight: 84,  
  fontWeight: 'bold',  
  textAlign: 'center',  
  backgroundColor: '#000000c0',  
},  
});
```

```
export default App;
```

Criar e Executar o Projeto ExemploImageBackground01.

React Native Componente - ImageBackground



React Native Componente – FlatList Com Imagens



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

function Item({ item }) {
  return (
    <View style={styles.listItem}>
      <Image source={{uri:item.photo}} style={{width:60, height:60,borderRadius:30}} />
      <View style={{alignItems:"center",flex:1}}>
        <Text style={{fontWeight:"bold"}}>{item.name}</Text>
        <Text>{item.position}</Text>
      </View>
      <TouchableOpacity style={{height:50,width:50, justifyContent:"center",alignItems:"center"}}>
        <Text style={{color:"green"}} onPress={()=>ligar()}>Ligar</Text>
      </TouchableOpacity>
    </View>
  );
}

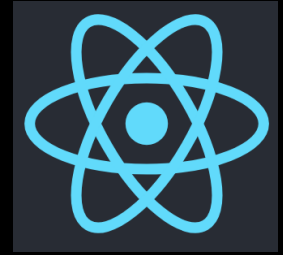
function ligar(){
  Alert.alert("Ligando....");
}
```

React Native Componente - FlatList Com Imagens



```
export default class App extends React.Component {  
  state = {  
    data:[  
      {  
        "name": "Miyah Myles",  
        "email": "miyah.myles@gmail.com",  
        "position": "Data Entry Clerk",  
        "photo": "https://images.unsplash.com/photo-1494790108377-be9c29b29330?ixlib=rb-  
0.3.5&q=80&fm=jpg&crop=entropy&cs=tinysrgb&w=200&fit=max&s=707b9c33066bf8808c934c8ab394dff6 "  
      },  
      {  
        "name": "June Cha",  
        "email": "june.cha@gmail.com",  
        "position": "Sales Manager",  
        "photo": "https://randomuser.me/api/portraits/women/44.jpg"  
      },  
      {  
        "name": "Iida Niskanen",  
        "email": "iida.niskanen@gmail.com",  
        "position": "Sales Manager",  
        "photo": "https://randomuser.me/api/portraits/women/68.jpg"  
      },  
    ]  
  }  
}
```

React Native Componente - FlatList Com Imagens



```
{
  "name": "Renee Sims",
  "email": "renee.sims@gmail.com",
  "position": "Medical Assistant",
  "photo": "https://randomuser.me/api/portraits/women/65.jpg"
},
{
  "name": "Jonathan Nu\u00f1ez",
  "email": "jonathan.nu\u00f1ez@gmail.com",
  "position": "Clerical",
  "photo": "https://randomuser.me/api/portraits/men/43.jpg"
},
{
  "name": "Sasha Ho",
  "email": "sasha.ho@gmail.com",
  "position": "Administrative Assistant",
  "photo": "https://images.pexels.com/photos/415829/pexels-photo-415829.jpeg?h=350&auto=compress&cs=tinysrgb"
},
{
  "name": "Abdullah Hadley",
  "email": "abdullah.hadley@gmail.com",
  "position": "Marketing",
  "photo": "https://images.unsplash.com/photo-1507003211169-0a1dd7228f2d?ixlib=rb-0.3.5&q=80&fm=jpg&crop=entropy&cs=tinysrgb&w=200&fit=max&s=a72ca28288878f8404a795f39642a46f"
},
}
```

React Native Componente - FlatList Com Imagens



```
{  
  "name": "Veeti Seppanen",  
  "email": "veeti.seppanen@gmail.com",  
  "position": "Product Designer",  
  "photo": "https://randomuser.me/api/portraits/men/97.jpg"  
},  
{  
  "name": "Bonnie Riley",  
  "email": "bonnie.riley@gmail.com",  
  "position": "Marketing",  
  "photo": "https://randomuser.me/api/portraits/women/26.jpg"  
}  
]  
}
```

React Native Componente - FlatList Com Imagens



```
render(){  
  return (  
    <View style={styles.container}>  
      <FlatList  
        style={{flex:1}}  
        data={this.state.data}  
        renderItem={({ item }) => <Item item={item}/>}  
        keyExtractor={item => item.email}  
      />  
    </View>  
  );  
}
```

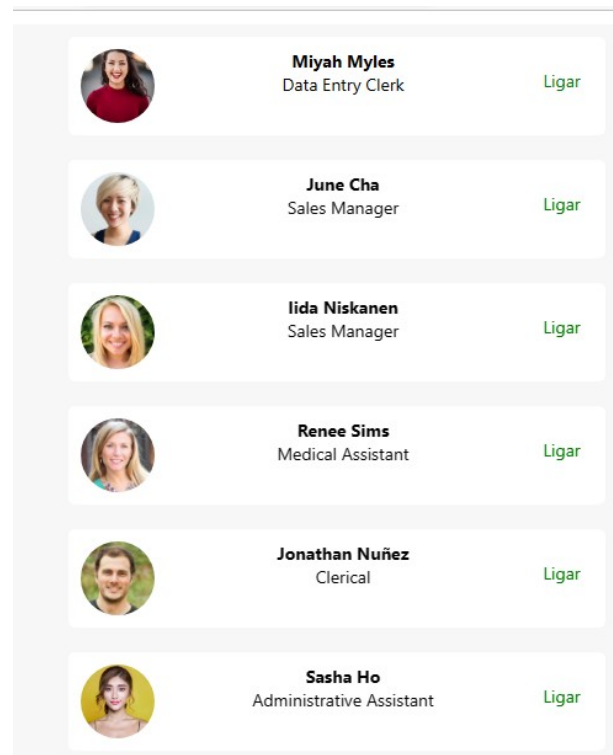
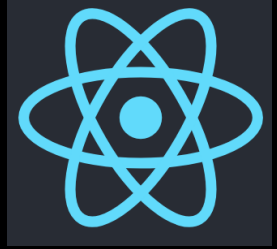
React Native Componente - FlatList Com Imagens



```
const styles = StyleSheet.create({
  container: {
    flex: 1,
    backgroundColor: '#F7F7F7',
    marginTop:60
  },
  listItem:{
    margin:10,
    padding:10,
    backgroundColor:"#FFF",
    width:"80%",
    flex:1,
    alignSelf:"center",
    flexDirection:"row",
    borderRadius:5
  }
});
```

Criar e Executar projeto ExemploFlatList05.

React Native Componente – FlatList Com Imagens



React Native Componente - RefreshControl



- Componente usado com o ScrollView ou o FlatList para atualizar uma lista ao deslizar a tela para baixo;
- O deslizar para baixo aciona um evento onRefresh;
- Passos para atualizar uma lista ao deslizar para baixo com o RefreshControl;
 - Criar uma lista na tela principal do app, usando o componente FlatList, por exemplo.
 - Criar uma função específica para fazer a requisição dos dados da sua lista.
 - Incluir o componente RefreshControl na lista alvo.
 - Definir as props **refreshing(booleano)**, codificação obrigatória, e **onRefresh(callback)** do componente RefreshControl.
- Propriedades e métodos em:
 - <https://reactnative.dev/docs/refreshcontrol>

React Native Componente - RefreshControl



```
import React, {useState, useEffect} from 'react';
import {SafeAreaView, Text, StyleSheet, View, FlatList, RefreshControl, StatusBar} from 'react-native';

const App = () => {
  const [refreshing, setRefreshing] = useState(false);
  const [dataSource, setDataSource] = useState([]);

  useEffect(() => {
    getData();
  }, []);

  const getData = () => {
    fetch('https://jsonplaceholder.typicode.com/todos')
      .then((response) => response.json())
      .then((responseJson) => {
        setDataSource(responseJson);
        setRefreshing(false);
      })
      .catch((error) => {
        console.error(error);
      });
  };
};
```

// Não utilizou o async / fetch / wait

React Native Componente - RefreshControl



```
const ItemView = ({item}) => {  
  return (  
    <Text  
      style={styles.itemStyle}  
      onPress={() => getItem(item)}>  
        {item.id} { ' - ' } {item.title.toUpperCase()}  
      </Text>  
    );  
  };  
};  
  
const getItem = (item) => {  
  alert('Id : ' + item.id + '\n\nTarefa : ' + item.title + '\n\nCompletada: ' + item.completed);  
};  
  
const onRefresh = () => {  
  setRefreshing(true);  
  setDataSource([]);  
  getData();  
};
```

// Só para exemplificar uma chamada por item selecionado

React Native Componente - RefreshControl



```
return (  
  <SafeAreaView style={{flex: 1}}>  
    <View style={styles.container}>  
      <StatusBar hidden={false} />  
      <FlatList  
        data={dataSource}  
        keyExtractor={item => item.id}  
        renderItem={ItemView}  
        refreshControl={  
          <RefreshControl  
            refreshing={refreshing}  
            onRefresh={onRefresh}  
          />  
        }  
      />  
    </View>  
  </SafeAreaView>  
);  
};
```

React Native Componente - RefreshControl

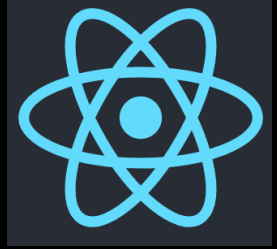


```
const styles = StyleSheet.create({
  container: {
    paddingTop: 40,
    backgroundColor: 'white',
  },
  itemStyle: {
    backgroundColor: '#0066CC',
    padding: 10,
    marginVertical: 8,
    marginHorizontal: 10,
    color: 'white',
  },
});
```

```
export default App;
```

Executar projeto ExemploRefreshControl01.

React Native Componente - RefreshControl



React Native Componente – RefreshControl Outro Exemplo



```
import React, {useState, useEffect} from 'react';
import {SafeAreaView, StyleSheet, ActivityIndicator, FlatList, Text, View, RefreshControl, StatusBar} from 'react-native';

const App = () => {
  const [refreshing, setRefreshing] = useState(true);
  const [dataSource, setDataSource] = useState([]);

  useEffect(() => {
    getData();
  }, []);

  const getData = () => {
    fetch('https://jsonplaceholder.typicode.com/posts')
      .then((response) => response.json())
      .then((responseJson) => {
        setRefreshing(false);
        setDataSource(responseJson);
      })
      .catch((error) => {
        console.error(error);
      });
  };
};
```

React Native Componente – RefreshControl Outro Exemplo



```
const ItemView = ({item}) => {  
  return (  
    <Text  
      style={styles.itemStyle}  
      onPress={() => getItem(item)}>  
        {item.title}  
      </Text>  
    );  
  };  
};  
  
const ItemSeparatorView = () => {  
  return (  
    <View  
      style={{  
        height: 0.5,  
        width: '100%',  
        backgroundColor: '#C8C8C8'}}  
    />  
  );  
};  
  
const getItem = (item) => {  
  alert('Id : ' + item.id + ' Body : ' + item.body);  
};
```

React Native Componente – RefreshControl Outro Exemplo



```
const onRefresh = () => {
  setDataSource([]);
  setRefreshing(true);
  getData();
};

return (
  <SafeAreaView style={{flex: 1}}>
    <View style={styles.container}>
      <StatusBar hidden={false} />
      {refreshing ? <ActivityIndicator /> : null}
      <FlatList
        data={dataSource}
        ItemSeparatorComponent={ItemSeparatorView}
        enableEmptySections={true}
        renderItem={ItemView}
        refreshControl={
          <RefreshControl
            refreshing={refreshing}
            onRefresh={onRefresh}
          />
        }
      />
    </View>
  </SafeAreaView>
```


React Native Componente – RefreshControl Outro Exemplo

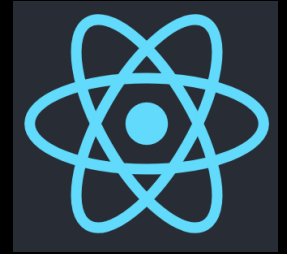


```
const styles = StyleSheet.create({
  container: {
    justifyContent: 'center',
    flex: 1,
    marginTop: 10,
  },
  itemStyle: {
    fontSize: 20,
    padding: 10,
  },
});
```

```
export default App;
```

Criar e Executar projeto ExemploRefreshControl02.

React Native Componente – RefreshControl Outro Exemplo



sunt aut facere repellat provident occaecati excepturi optio reprehenderit

qui est esse

ea molestias quasi exercitationem repellat qui ipsa sit aut

eum et est occaecati

nesciunt quas odio

dolorem eum magni eos aperiam quia

magnam facilis autem

dolorem dolore est ipsam

nesciunt iure omnis dolorem tempora et accusantium

optio molestias id quia eum

et ea vero quia laudantium autem

in quibusdam tempore odit est dolorem

React Native Componente - ScrollView



- O componente ScrollView é um container rolável genérico, que rola vários componentes filhos, e exibições, em seu interior;
- No ScrollView, os componentes podem rolar na direção vertical e horizontal;
- Por padrão, o container ScrollView rola seus componentes e exibições na vertical;
- Para rolar seus componentes na horizontal, utiliza-se o props horizontal;
- Propriedades e métodos em:
 - <https://reactnative.dev/docs/scrollview> (vamos visitar!)

React Native Componente - ScrollView



```
import React from 'react';
import { StyleSheet, Text, SafeAreaView, ScrollView, StatusBar, } from 'react-native';
const App = () => {
  return (
    <SafeAreaView style={styles.container}>
      <ScrollView style={styles.scrollView}>
        <Text style={styles.text}>
          Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do
          eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad
          minim veniam, quis nostrud exercitation ullamco laboris nisi ut
          aliquip ex ea commodo consequat. Duis aute irure dolor in
          reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla
          pariatur. Excepteur sint occaecat cupidatat non proident, sunt in
          culpa qui officia deserunt mollit anim id est laborum.
        </Text>
      </ScrollView>
    </SafeAreaView>
  );
};
```

React Native Componente - ScrollView

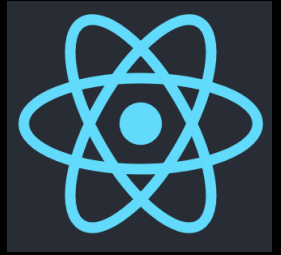


```
const styles = StyleSheet.create({
  container: {
    flex: 1,
    paddingTop: StatusBar.currentHeight,
  },
  scrollView: {
    backgroundColor: 'pink',
    marginHorizontal: 20,
  },
  text: {
    fontSize: 42,
  },
});

export default App;
```

Executar Projeto ExemploScrollView01

React Native Componente - ScrollView



Lorem ipsum dolor
sit amet, consectetur
adipiscing elit, sed
do eiusmod tempor
incididunt ut labore
et dolore magna
aliqua. Ut enim ad
minim veniam, quis
nostrud exercitation
ullamco laboris nisi
ut aliquip ex ea

React Native Componente - `SectionList`



- O `SectionList` é um componente para exibição de uma lista de dados logicamente seccionada;
- Propriedades e métodos em:
 - <https://reactnative.dev/docs/sectionlist> (vamos visitar!)

React Native Componente - SectionList



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

const DATA = [
  {
    title: 'Pratos Principais',
    data: ['Pizza', 'Burger', 'Risotto'],
  },
  {
    title: 'Acompanhamentos',
    data: ['Batata Frita', 'Anéis de Cebola', 'Salada Mista'],
  },
  {
    title: 'Drinks',
    data: ['Água', 'Refrigerante', 'Cerveja'],
  },
  {
    title: 'Sobremesa',
    data: ['Sorvete', 'Pudim'],
  },
];
```


React Native Componente - SectionList



```
const App = () => (  
  <SafeAreaView style={styles.container}>  
    <SectionList  
      sections={DATA}  
      keyExtractor={(item, index) => item + index}  
      renderItem={({item}) => (  
        <View style={styles.item}>  
          <Text style={styles.title}>{item}</Text>  
        </View>  
      )}  
      renderSectionHeader={({section: {title}}) => (  
        <Text style={styles.header}>{title}</Text>  
      )}  
    />  
  </SafeAreaView>  
);
```

React Native Componente - SectionList

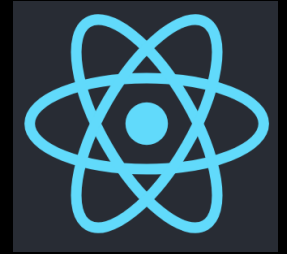


```
const styles = StyleSheet.create({
  container: {
    flex: 1,
    paddingTop: StatusBar.currentHeight,
    marginHorizontal: 16,
  },
  item: {
    backgroundColor: '#f9c2ff',
    padding: 20,
    marginVertical: 8,
  },
  header: {
    fontSize: 32,
    backgroundColor: '#fff',
  },
  title: {
    fontSize: 24,
  },
});

export default App;
```

Executar Projeto ExemploSectionList01

React Native Componente - SectionList



Pratos Principais

Pizza

Burger

Risotto

Acompanhamentos

Batata Frita

Anéis de Cebola

Salada Mista

Drinks

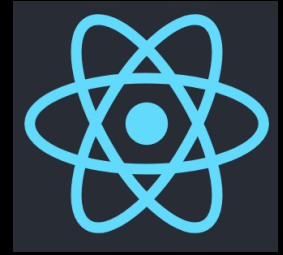
Água

React Native Componente - StatusBar



- O StatusBar é um componente para controlar a barra de status do app;
- A barra de status é a área, geralmente na parte superior da tela, que exibe a hora atual, Wi-Fi e informações da rede móvel, nível da bateria e/ou outros ícones de status;
- Propriedades e métodos em:
 - <https://reactnative.dev/docs/statusbar> (vamos visitar!)

React Native Componente - StatusBar



```
import React, {useState} from 'react';
import {Button, Platform, SafeAreaView, StatusBar, StyleSheet, Text, View,} from 'react-native';
import type {StatusBarStyle} from 'react-native';

const Separator = () => <View style={styles.separator} />;

const STYLES = ['default', 'dark-content', 'light-content'];
const TRANSITIONS = ['fade', 'slide', 'none'];

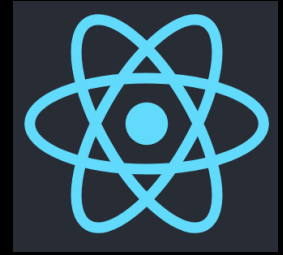
const App = () => {
  const [hidden, setHidden] = useState(false);
  const [statusBarStyle, setStatusBarStyle] = useState(STYLES[0]);
  const [statusBarTransition, setStatusBarTransition] = useState(TRANSITIONS[0]);

  const changeStatusBarVisibility = () => setHidden(!hidden);

  const changeStatusBarStyle = () => {
    const styleId = STYLES.indexOf(statusBarStyle) + 1;
    if (styleId === STYLES.length) {
      setStatusBarStyle(STYLES[0]);
    } else {
      setStatusBarStyle(STYLES[styleId]);
    }
  };

  const changeStatusBarTransition = () => {
    const transition = TRANSITIONS.indexOf(statusBarTransition) + 1;
    if (transition === TRANSITIONS.length) {
      setStatusBarTransition(TRANSITIONS[0]);
    } else {
      setStatusBarTransition(TRANSITIONS[transition]);
    }
  };
};
```

React Native Componente - StatusBar



```
return (  
  <SafeAreaView style={styles.container}>  
    <StatusBar  
      animated={true}  
      backgroundColor="#61dafb"  
      barStyle={statusBarStyle}  
      showHideTransition={statusBarTransition}  
      hidden={hidden}  
    />  
    <Text style={styles.textStyle}>  
      StatusBar Visibilidade: {'\n'} {hidden ? 'Oculto' : 'Visível'}  
    </Text>  
    <Text style={styles.textStyle}>  
      StatusBar Style: {'\n'} {statusBarStyle}  
    </Text>  
    {Platform.OS === 'ios' ? (  
      <Text style={styles.textStyle}>  
        StatusBar Transition: {'\n'} {statusBarTransition}  
      </Text>  
    ) : null}  
  <View style={styles.buttonsContainer}>  
    <Button title="Alternar StatusBar" onPress={changeStatusBarVisibility} />  
    <Separator />  
    <Button title="Trocar Estilo da StatusBar" onPress={changeStatusBarStyle} />  
    {Platform.OS === 'ios' ? (  
      <Button  
        title="Alterar Transição da Barra de Status"  
        onPress={changeStatusBarTransition}  
      />  
    ) : null}  
  </View>  
</SafeAreaView>  
};
```

React Native Componente - StatusBar

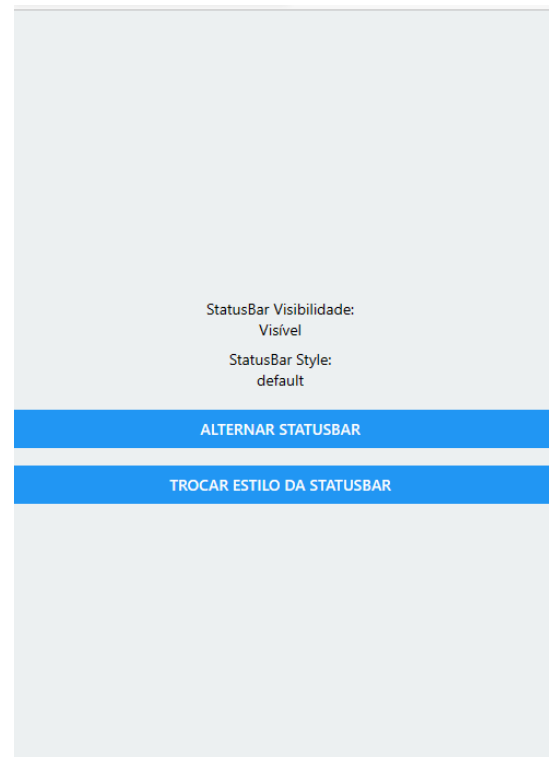
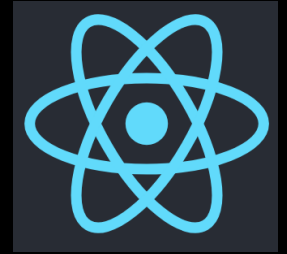


```
const styles = StyleSheet.create({
  container: {
    flex: 1,
    justifyContent: 'center',
    backgroundColor: '#ECF0F1',
  },
  buttonsContainer: {
    padding: 10,
  },
  textStyle: {
    textAlign: 'center',
    marginBottom: 8,
  },
});
```

```
export default App;
```

Criar e Executar o Projeto ExemploStatusBar01, no celular!!!

React Native Componente - StatusBar



React Native Componente - Switch



- O Switch é um componente de controle *booleano* que define seu valor como verdadeiro ou falso;
- Possui um método de retorno, `onValueChange`, que atualiza sua propriedade de valor;
- Propriedades e métodos em:
 - <https://reactnative.dev/docs/switch> (vamos visitar!)

React Native Componente - Switch



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

const App = () => {
  const [isEnabled, setIsEnabled] = useState(false);

  const toggleSwitch = () => setIsEnabled(previousState => !previousState);

  return (
    <View style={styles.container}>
      <Text style={styles.textStyle}>Switch Exemplo</Text>
      <Text style={styles.textStyle}>{isEnabled ? 'on' : 'off'}</Text>
      <Switch
        trackColor={{false: '#767577', true: '#81b0ff'}}
        thumbColor={isEnabled ? '#00ff00' : '#ff0000'}
        ios_backgroundColor="#3e3e3e"
        onValueChange={toggleSwitch}
        value={isEnabled}
      />
    </View>
  );
};
```

React Native Componente - Switch

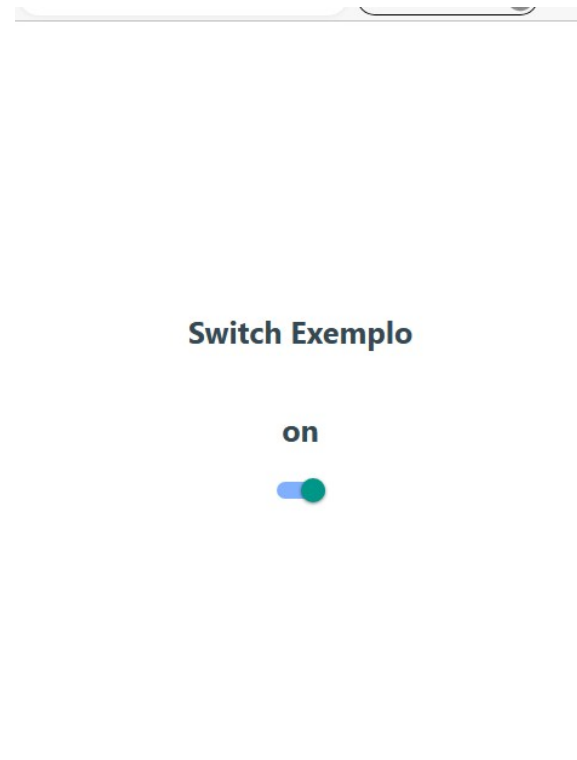
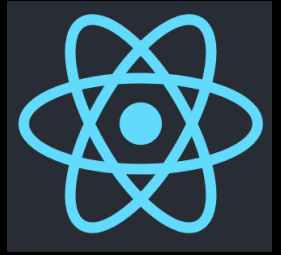


```
const styles = StyleSheet.create({
  container: {
    flex: 1,
    alignItems: 'center',
    justifyContent: 'center',
  },
  textStyle: {
    margin: 24,
    fontSize: 25,
    fontWeight: 'bold',
    textAlign: 'center',
    color: '#344953'
  },
});
```

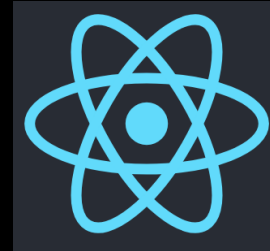
```
export default App;
```

Executar Projeto ExemploSwitch01.

React Native Componente - Switch

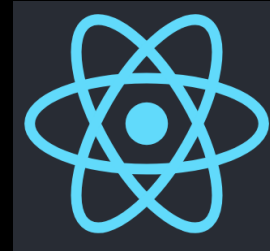


React Native Componente - TouchableHighlight



- O TouchableHighlight é um componente *wrapper* que faz com que visualizações respondam aos toques com características próprias;
- Ao pressionar para baixo, a visualização aumenta;
- TouchableHighlight deve ter um filho (não zero ou mais de um). Se desejar ter vários componentes filhos, envolva-os em uma exibição;
- Propriedades e métodos em:
 - <https://reactnative.dev/docs/touchablehighlight> (vamos visitar!)

React Native Componente - TouchableHighlight



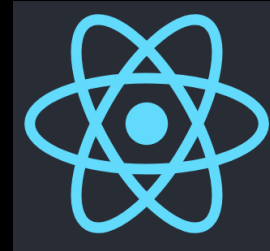
```
import React, {useState} from 'react';
import {StyleSheet, Text, TouchableHighlight, View} from 'react-native';

const TouchableHighlightExample = () => {
  const [count, setCount] = useState(0);

  const onPress = () => setCount(count + 1);

  return (
    <View style={styles.container}>
      <TouchableHighlight onPress={onPress}>
        <View style={styles.button}>
          <Text>Toque Aqui!!!</Text>
        </View>
      </TouchableHighlight>
      <View style={styles.countContainer}>
        <Text style={styles.countText}>{count || null}</Text>
      </View>
    </View>
  );
};
```

React Native Componente - TouchableHighlight

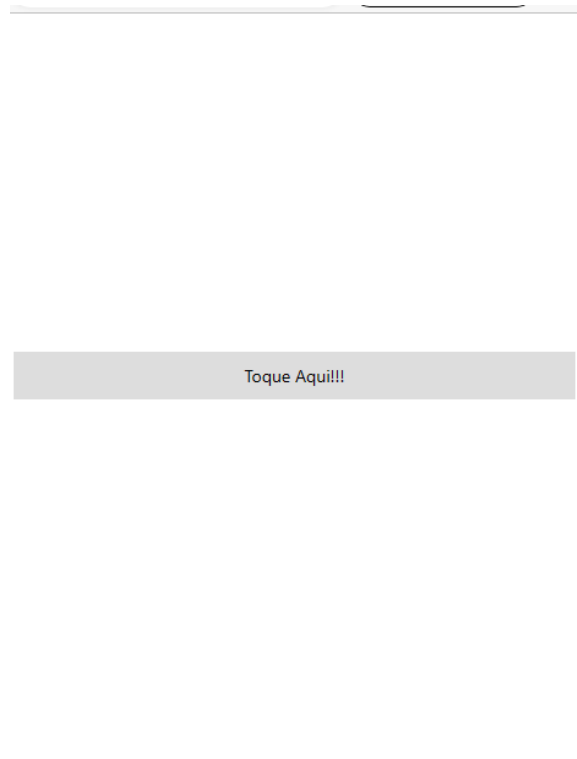
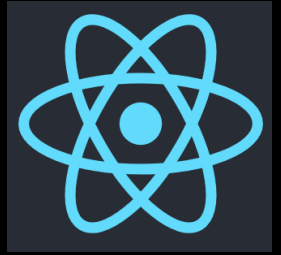


```
const styles = StyleSheet.create({
  container: {
    flex: 1,
    justifyContent: 'center',
    paddingHorizontal: 10,
  },
  button: {
    alignItems: 'center',
    backgroundColor: '#dddddd',
    padding: 10,
  },
  countContainer: {
    alignItems: 'center',
    padding: 10,
  },
  countText: {
    color: '#FF00FF',
  },
});
```

```
export default TouchableHighlightExample;
```

Executar Projeto ExemploTouchableHighlight01.

React Native Componente - TouchableHighlight



React Native Componente - TouchableOpacity



- O TouchableOpacity é um componente *wrapper* que faz com que as visualizações respondam aos toques com características próprias;
- Ao pressionar para baixo, a opacidade da visualização agrupada diminui, escurecendo-a;
- A opacidade é controlada agrupando os filhos em um Animated.View, que é adicionado à hierarquia de exibição;
- Propriedades e métodos em:
 - <https://reactnative.dev/docs/touchableopacity>

React Native Componente - TouchableOpacity



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

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

const App = () => {
  const [count, setCount] = useState(0);
  const onPress = () => setCount(prevCount => prevCount + 1);

  return (
    <View style={styles.container}>
      <View style={styles.countContainer}>
        <Text>Conta Toques: {count}</Text>
      </View>
      <TouchableOpacity style={styles.button} onPress={onPress}>
        <Text>Toque Aqui!!!</Text>
      </TouchableOpacity>
    </View>
  );
};
```

React Native Componente - TouchableOpacity



```
const styles = StyleSheet.create({
  container: {
    flex: 1,
    justifyContent: 'center',
    paddingHorizontal: 10,
  },
  button: {
    alignItems: 'center',
    backgroundColor: '#DDDDDD',
    padding: 10,
  },
  countContainer: {
    alignItems: 'center',
    padding: 10,
  },
});
```

```
export default App;
```

Executar Projeto ExemploTouchableopacity01.

React Native Componente - TouchableOpacity

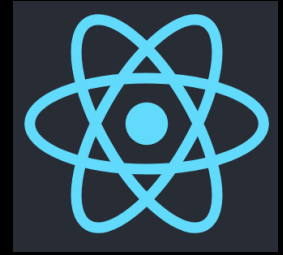


React Native Componente - TouchableWithoutFeedback



- O TouchableWithoutFeedback é um componente usado quando o usuário deseja manipular a funcionalidade de toque, mas não deseja exibir nenhum *feedback* visual quando toca;
- Propriedades e métodos em:
 - <https://reactnative.dev/docs/touchablewithoutfeedback> (vamos visitar!)

React Native Componente - TouchableWithoutFeedback



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

const TouchableWithoutFeedbackExample = () => {
  const [count, setCount] = useState(0);

  const onPress = () => {
    setCount(count + 1);
  };

  return (
    <View style={styles.container}>
      <View style={styles.countContainer}>
        <Text style={styles.countText}>ContaToques: {count}</Text>
      </View>
      <TouchableWithoutFeedback onPress={onPress}>
        <View style={styles.button}>
          <Text>Toque Aqui!!!</Text>
        </View>
      </TouchableWithoutFeedback>
    </View>
  );
};
```

React Native Componente - TouchableWithoutFeedback



```
const styles = StyleSheet.create({
  container: {
    flex: 1,
    justifyContent: 'center',
    paddingHorizontal: 10,
  },
  button: {
    alignItems: 'center',
    backgroundColor: '#DDDDDD',
    padding: 10,
  },
  countContainer: {
    alignItems: 'center',
    padding: 10,
  },
  countText: {
    color: '#FF00FF',
  },
});
export default TouchableWithoutFeedbackExample;
```

Executar Projeto ExemploTouchableWithoutFeedback01.

React Native Componente - TouchableWithoutFeedback



React Native Componente Exemplo Touchable com todos



```
import { StatusBar } from 'expo-status-bar';
import React from 'react';
import { Alert, Platform, StyleSheet, Text, TouchableHighlight, TouchableOpacity, TouchableWithoutFeedback, View } from
  'react-native';

const _onPressButton = () =>{
  Alert.alert('Botão foi tocado!!!!') ;
}
const _onLongPressButton = () =>{
  Alert.alert('Botão tocado por longo tempo!!!') ;
}
export default function App() {
  return (
    <View style={styles.container}>
      <TouchableHighlight onPress={_onPressButton} underlayColor="white">
        <View style={styles.button}>
          <Text style={styles.buttonText}>TouchableHighlight</Text>
        </View>
      </TouchableHighlight>
      <TouchableOpacity onPress={_onPressButton}>
        <View style={styles.button}>
          <Text style={styles.buttonText}>TouchableOpacity</Text>
        </View>
      </TouchableOpacity>
    </View>
  );
}
```

React Native Componente Exemplo Touchable com todos



```
<TouchableWithoutFeedback
  onPress={_onPressButton}
>
  <View style={styles.button}>
    <Text style={styles.buttonText}>TouchableWithoutFeedback</Text>
  </View>
</TouchableWithoutFeedback>
<TouchableHighlight onPress={_onPressButton} onLongPress={_onLongPressButton} underlayColor="white">
  <View style={styles.button}>
    <Text style={styles.buttonText}>Touchable with Long Press</Text>
  </View>
</TouchableHighlight>
</View>
);
}
```

React Native Componente Exemplo Touchable com todos

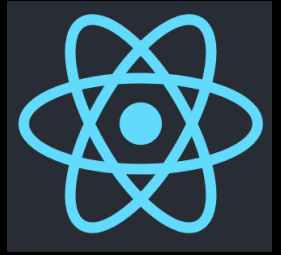


```
const styles = StyleSheet.create({
  container: {
    paddingTop: 60,
    alignItems: 'center'
  },
  button: {
    marginBottom: 30,
    width: 260,
    alignItems: 'center',
    backgroundColor: '#5ead97'
  },
  buttonText: {
    padding: 20,
    color: 'white',
    fontSize: 16
  }
});
```

Executar Projeto ExemploTouchableAll, no celular!!!.

React Native Componente

Exemplo Touchable com todos



TouchableHighlight

TouchableOpacity

TouchableWithoutFeedback

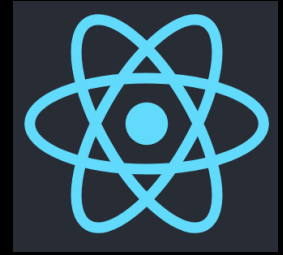
Touchable with Long Press

React Native Componente - KeyboardAvoidingView



- O KeyboardAvoidingView é um componente que ajustará automaticamente sua altura, posição ou preenchimento inferior, com base na altura do teclado, para permanecer visível enquanto o teclado virtual é exibido;
- Propriedades e métodos em:
<https://reactnative.dev/docs/keyboardavoidingview> (vamos visitar!)

React Native Componente - KeyboardAvoidingView



```
import React from 'react';
import {View,KeyboardAvoidingView,TextInput,StyleSheet,Text,Platform,TouchableWithoutFeedback,Button,Keyboard} from
  'react-native';

const KeyboardAvoidingComponent = () => {
  return (
    <KeyboardAvoidingView
      behavior={Platform.OS === 'ios' ? 'padding' : 'height'}
      style={styles.container}>
      <TouchableWithoutFeedback onPress={Keyboard.dismiss}>
        <View style={styles.inner}>
          <Text style={styles.header}>Cabeçalho</Text>
          <TextInput placeholder="Usuário" style={styles.textInput} />
          <View style={styles.btnContainer}>
            <Button title="Enviar" onPress={() => null} />
          </View>
        </View>
      </TouchableWithoutFeedback>
    </KeyboardAvoidingView>
  );
};
```

React Native Componente - KeyboardAvoidingView



```
const styles = StyleSheet.create({
  container: {
    flex: 1,
  },
  inner: {
    padding: 24,
    flex: 1,
    justifyContent: 'space-around',
  },
  header: {
    fontSize: 36,
    marginBottom: 48,
  },
  textInput: {
    height: 40,
    borderColor: '#000000',
    borderBottomWidth: 1,
    marginBottom: 36,
  },
});
```

React Native Componente - KeyboardAvoidingView



```
btnContainer: {  
  backgroundColor: 'white',  
  marginTop: 12,  
},  
});
```

```
export default KeyboardAvoidingComponent;
```

Criar e Executar Projeto ExemploKeyboardAvoidingView, no celular.

React Native Componente - KeyboardAvoidingView



Cabeçalho

Usuário

ENVIAR

React Native Navigation



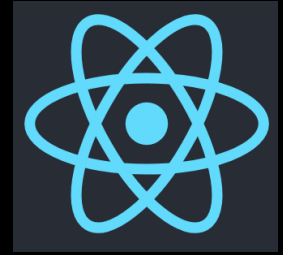
- Aplicativos móveis raramente são compostos de uma única tela;
- O gerenciamento da apresentação e da transição entre várias telas geralmente é feito por um navegador;
- Existem dois tipos de navegação incorporados em apps móveis: navegação de pilha e navegação com guias;
- O React Navigation atende ao dois tipos, tanto no Android quanto no iOS;
- Em um navegador da web, pode-se criar links para páginas diferentes usando uma marca de âncora (`<a>`);
- Quando o usuário clica em um link, o URL é enviado para a pilha do histórico do navegador;
- Quando o usuário pressiona o botão Voltar, o navegador exibe o item do topo da pilha do histórico, de modo que a página ativa agora é a página visitada anteriormente;
- O React Native não tem uma pilha de histórico global como um navegador da web - é aqui que o React Navigation entra na história;

React Native Navigation



- O navegador de pilha nativo do React Navigation fornece uma maneira do aplicativo fazer a transição entre as telas e gerenciar o histórico de navegação;
- Se o aplicativo usar apenas o navegador de pilha, ele é conceitualmente semelhante a como um navegador da web lida com o estado de uma navegação;
- O aplicativo envia e exibe itens da pilha de navegação a medida que os usuários interagem com ele;
- Uma diferença fundamental entre como isso funciona em um navegador da web e no React Navigation é que o navegador de pilha nativo do React Navigation fornece os gestos e animações que você esperaria no Android e no iOS;
- Guia inicial de uso em:
<https://reactnavigation.org/docs/getting-started> (vamos visitar!)

React Native Navigation - Instalação



■ Instalação dos pacotes com todos os comandos emitidos no diretório do projeto alvo:

- No prompt do DOS:
 - `npm install @react-navigation/native`
- No prompt do DOS, pacote para resultado na web(opção w):
 - `npx expo install react-native-web@~0.18.10 react-dom@18.2.0 @expo/webpack-config@^18.0.1`
- No prompt do DOS, dependências Expo:
 - `npx expo install react-native-screens react-native-safe-area-context`
- No prompt do DOS, mais dependências:
 - `npm install react-native-screens react-native-safe-area-context`
- No prompt do DOS, **somente para máquinas MAC**:
 - `npx pod-install ios`
- No prompt do DOS, biblioteca native-stack:
 - `npm install @react-navigation/native-stack`
- No prompt do DOS, biblioteca complementar:
 - `npx expo install expo-splash-screen@~0.16.2 react@18.0.0 react-native@0.69.9`

React Native Componente NavigationContainer



- Todo o aplicativo fica agrupado no NavigationContainer;

```
import * as React from 'react';
import {NavigationContainer} from '@react-navigation/native';

export default function App() {
  return (
    <NavigationContainer>
      {/* Código do app navegável aqui */}
    </NavigationContainer>
  );
}
```

- O uso de maiúsculas e minúsculas no nome da rota não importa. Preferimos capitalizar nossos nomes de rota;

React Native Navigation – Código Básico



```
import * as React from 'react';
import { View, Text } from 'react-native';
import { NavigationContainer } from '@react-navigation/native';
import { createNativeStackNavigator } from '@react-navigation/native-stack';

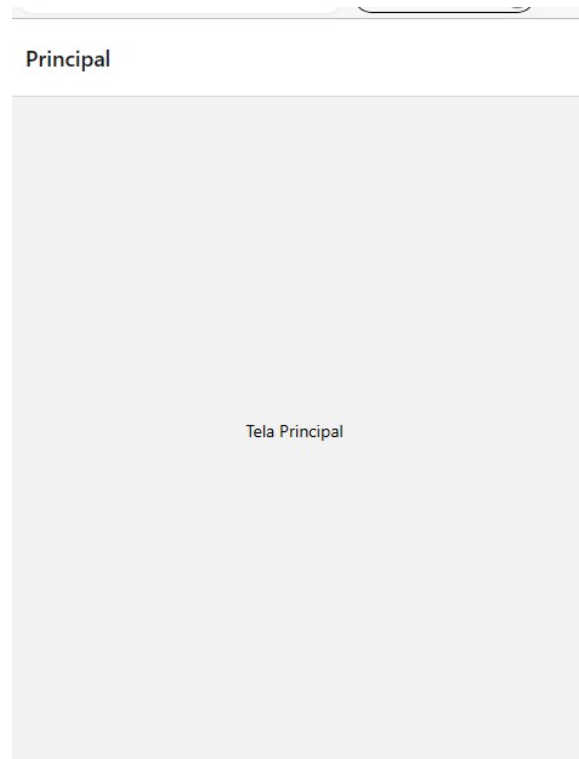
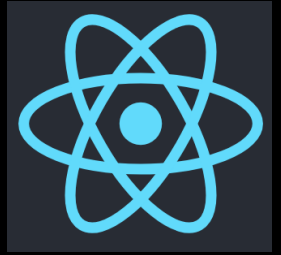
function HomeScreen() {
  return (
    <View style={{ flex: 1, alignItems: 'center', justifyContent: 'center' }}>
      <Text>Tela Principal</Text>
    </View>
  );
}

const Stack = createNativeStackNavigator();

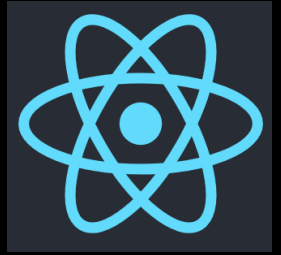
function App() {
  return (
    <NavigationContainer>
      <Stack.Navigator>
        <Stack.Screen name="Principal" component={HomeScreen} />
      </Stack.Navigator>
    </NavigationContainer>
  );
}

export default App;
```

React Native Navigation – Código Básico



React Native Navigation – Movendo entre Telas



- Adicionando mais uma rota à pilha, a rota Detalhes;
- Uma rota pode ser especificada usando o componente Screen;
- O componente Screen aceita uma prop name que corresponde ao nome da rota que usaremos para navegar, e uma prop component que corresponde ao componente que ele irá renderizar;

React Native Navigation – Movendo entre Telas e passando parâmetros

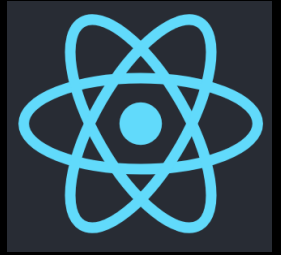


```
import * as React from 'react';
import { Button, View, Text, StyleSheet } from 'react-native';
import { NavigationContainer } from '@react-navigation/native';
import { createNativeStackNavigator } from '@react-navigation/native-stack';

const Separator = () => <View style={styles.separator} />;

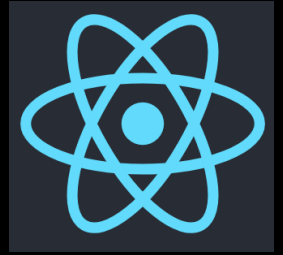
function HomeScreen({ navigation }) {
  return (
    <View style={{ flex: 1, alignItems: 'center', justifyContent: 'center' }}>
      <Text>Tela Principal</Text>
      <Button
        title="Para Detalhes"
        onPress={() => {navigation.navigate('Detalhes',{itemId:86,otherParam: 'Qualquer Coisa',});}}
      />
    </View>
  );
}
```

React Native Navigation – Movendo entre Telas e passando parâmetros



```
function DetailsScreen({route, navigation}) {  
  
  const { itemId, otherParam } = route.params;  
  return (  
    <View style={{ flex: 1, alignItems: 'center', justifyContent: 'center' }}>  
      <Text>Detalhes da Tela</Text>  
      <Text>itemId:{JSON.stringify(itemId)}</Text>  
      <Text>Outro Parâmetro:{JSON.stringify(otherParam)}</Text>  
      <Separator />  
      <Button  
        title="Para Detalhes Novamente"  
        onPress={() => navigation.push('Detalhes', {itemId: Math.floor(Math.random() * 100),})}  
      />  
      <Separator />  
      <Button title="Vai Para Principal" onPress={() => navigation.navigate('Principal')} />  
      <Separator />  
      <Button title="Voltar" onPress={() => navigation.goBack()} />  
    </View>  
  );  
}
```

React Native Navigation – Movendo entre Telas e passando parâmetros



```
const Stack = createNativeStackNavigator();

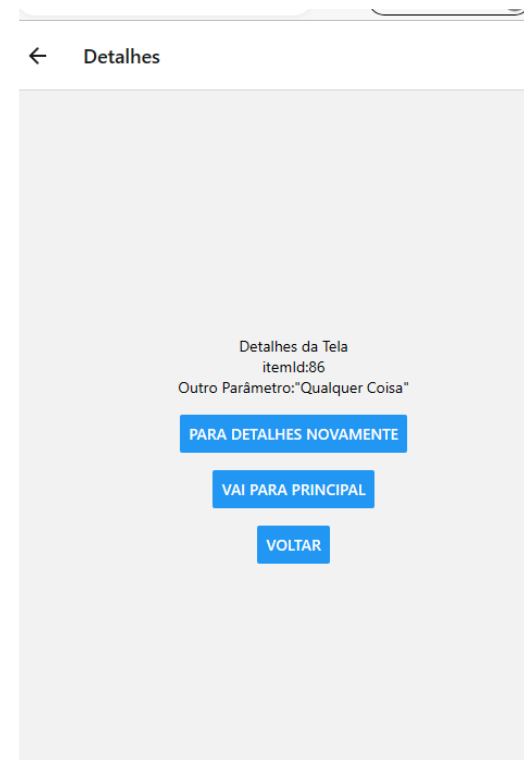
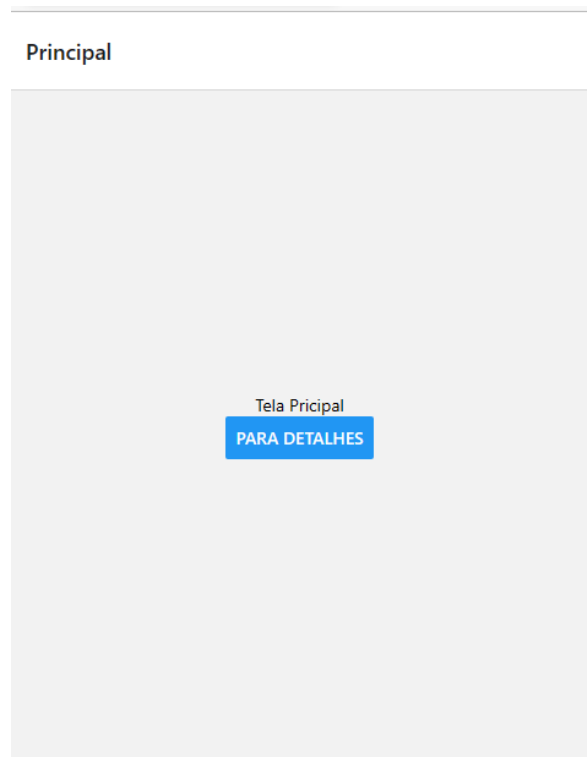
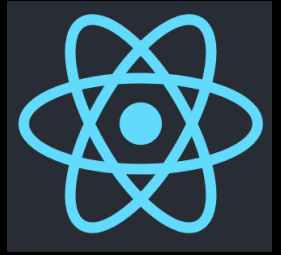
function App() {
  return (
    <NavigationContainer>
      <Stack.Navigator>
        <Stack.Screen name="Principal" component={HomeScreen} />
        <Stack.Screen name="Detalhes" component={DetailsScreen} />
      </Stack.Navigator>
    </NavigationContainer>
  );
}

const styles = StyleSheet.create({
  separator: {
    marginVertical: 8,
  },
});

export default App;
```

Executar o projeto ExemploNavigation01.

React Native Navigation – Movendo entre Telas e passando parâmetros



React Native Navigation

Parâmetros iniciais e atualização



```
import * as React from 'react';
import { Text, View, Button } from 'react-native';
import { NavigationContainer } from '@react-navigation/native';
import { createNativeStackNavigator } from '@react-navigation/native-stack';

function HomeScreen({ route, navigation }) {
  const { itemId } = route.params;
  return (
    <View style={{ flex: 1, alignItems: 'center', justifyContent: 'center' }}>
      <Text>Home Screen</Text>
      <Text>itemId: {JSON.stringify(itemId)}</Text>
      <Button
        title="Atualiza Parâmetro"
        onPress={() => navigation.setParams({itemId: Math.floor(Math.random() * 100),})}
      />
    </View>
  );
}
```

React Native Navigation

Parâmetros iniciais e atualização



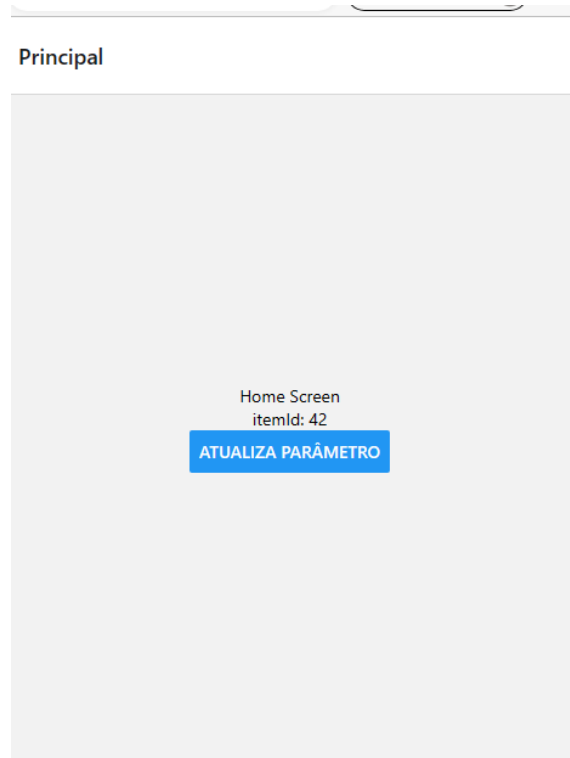
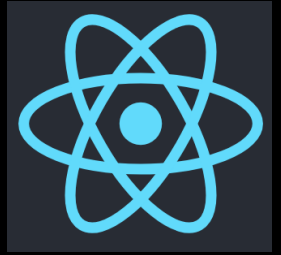
```
const Stack = createNativeStackNavigator();

export default function App() {
  return (
    <NavigationContainer>
      <Stack.Navigator>
        <Stack.Screen
          name="Principal"
          component={HomeScreen}
          initialParams={{ itemId: 42 }}
        />
      </Stack.Navigator>
    </NavigationContainer>
  );
}
```

Executar o projeto ExemploNavigation02.

React Native Navigation

Passando parâmetros entre Telas



React Native Navigation

Passando parâmetros para tela anterior



- Os parâmetros não são úteis apenas para passar alguns dados para uma nova tela;
- Também podem ser úteis para passar dados para uma tela anterior;

React Native Navigation

Passando parâmetros para tela anterior



```
import * as React from 'react';
import { Text, TextInput, View, Button } from 'react-native';
import { NavigationContainer } from '@react-navigation/native';
import { createNativeStackNavigator } from '@react-navigation/native-stack';

function HomeScreen({ navigation, route }) {

  return (
    <View style={{ flex: 1, alignItems: 'center', justifyContent: 'center' }}>
      <Button
        title="Cria a postagem"
        onPress={() => navigation.navigate('CriaPost')}
      />
      <Text style={{ margin: 10 }}>Postado: {route.params?.post} </Text>
    </View>
  );
}
```

React Native Navigation

Passando parâmetros para tela anterior



```
function CreatePostScreen({ navigation, route }) {  
  const [postText, setPostText] = React.useState("");  
  return (  
    <>  
      <TextInput  
        multiline  
        placeholder="Escreva aqui o que pensas agora."  
        style={{ height: 200, padding: 10, backgroundColor: 'white' }}  
        value={postText}  
        onChangeText={setPostText}  
      />  
      <Button  
        title="OK"  
        onPress={() => {  
          navigation.navigate({ name: 'Principal', params: { post: postText}, merge: true,});  
        }}  
      />  
    </>  
  );  
}
```

const Stack = createNativeStackNavigator();

React Native Navigation

Passando parâmetros para tela anterior

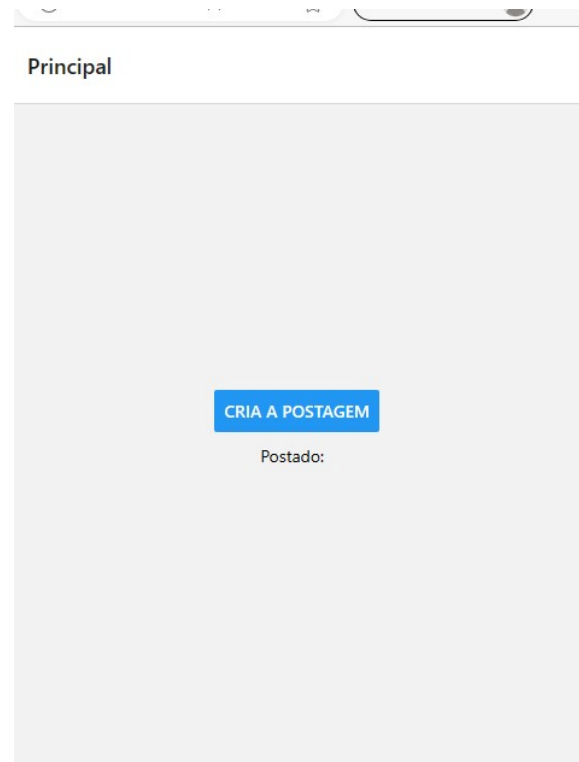


```
export default function App() {  
  
  return (  
  
    <NavigationContainer>  
  
      <Stack.Navigator mode="modal">  
  
        <Stack.Screen name="Principal" component={HomeScreen} />  
  
        <Stack.Screen name="CriaPost" component={CreatePostScreen} />  
  
      </Stack.Navigator>  
  
    </NavigationContainer>  
  
  );  
}
```

Executar o projeto ExemploNavigation03.

React Native Navigation

Passando parâmetros para tela anterior



React Native Navigation

Configurando a barra de cabeçalho



```
import * as React from 'react';
import { View, Text } from 'react-native';
import { NavigationContainer } from '@react-navigation/native';
import { createNativeStackNavigator } from '@react-navigation/native-stack';
```

```
const Stack = createNativeStackNavigator();
```

```
function HomeScreen() {
  return (
    <View style={{ flex: 1, alignItems: 'center', justifyContent: 'center' }}>
      <Text>Tela Principal</Text>
    </View>
  );
}
```

React Native Navigation

Configurando a barra de cabeçalho

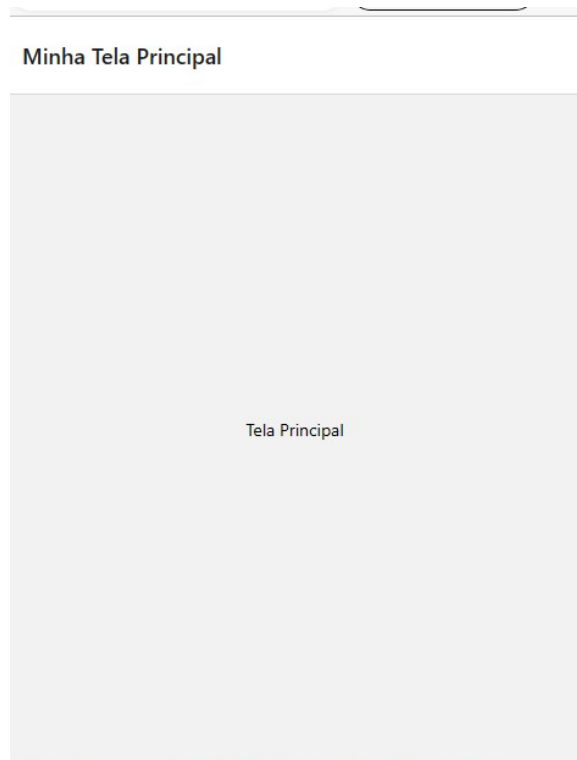
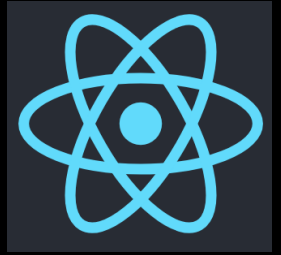


```
function App() {  
  return (  
    <NavigationContainer>  
      <Stack.Navigator>  
        <Stack.Screen  
          name="Home"  
          component={HomeScreen}  
          options={{ title: 'Minha Tela Principal' }}  
        />  
      </Stack.Navigator>  
    </NavigationContainer>  
  );  
}  
  
export default App;
```

Executar o projeto ExemploNavigation04.

React Native Navigation

Configurando a barra de cabeçalho



React Native Navigation

Configurando a barra de cabeçalho



```
import * as React from 'react';
import { View, Text, Button } from 'react-native';
import { NavigationContainer } from '@react-navigation/native';
import { createNativeStackNavigator } from '@react-navigation/native-stack';

function HomeScreen({ navigation }) {
  return (
    <View style={{ flex: 1, alignItems: 'center', justifyContent: 'center' }}>
      <Text>Tela Principal</Text>
      <Button title="Vai Para o Perfil" onPress={() => navigation.navigate('Profile', { name: 'Cabeçalho Customizado' })} />
    </View>
  );
}

function ProfileScreen({ navigation }) {
  return (
    <View style={{ flex: 1, alignItems: 'center', justifyContent: 'center' }}>
      <Text>Tela Perfil</Text>
    </View>
  );
}

const Stack = createNativeStackNavigator();
```


React Native Navigation

Configurando a barra de cabeçalho



```
function App() {  
  return (  
    <NavigationContainer>  
      <Stack.Navigator>  
        <Stack.Screen  
          name="Home"  
          component={HomeScreen}  
          options={{  
            title: 'Minha Tela Principal',  
            headerStyle: {  
              backgroundColor: '#f4511e',  
            },  
            headerTintColor: '#fff',  
            headerTitleStyle: {  
              fontWeight: 'bold',  
            },  
          }}  
        />  
      </Stack.Navigator>  
    </NavigationContainer>  
  );  
}
```

React Native Navigation

Configurando a barra de cabeçalho



```
<Stack.Screen
  name="Profile"
  component={ProfileScreen}
  options={({ route }) => ({ title: route.params.name, headerStyle: { backgroundColor: '#00ffff', } })}
/>
</Stack.Navigator>
</NavigationContainer>

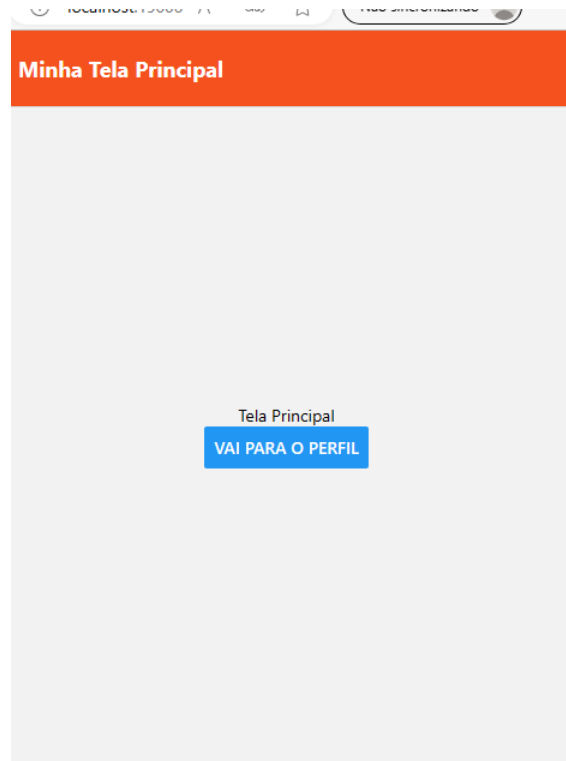
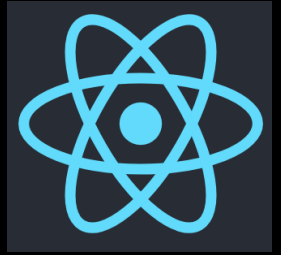
);
}

export default App;
```

Executar o projeto ExemploNavigation05.

React Native Navigation

Configurando a barra de cabeçalho



React Native Tab Navigation



- Possivelmente, o estilo mais comum de navegação em apps móveis é a navegação baseada em guias;
- Podem ser guias na parte inferior da tela ou na parte superior, abaixo do cabeçalho, ou mesmo no lugar do cabeçalho;
- Usado para alternar entre diferentes telas de rota;
- Instalar o pacote, no diretório do projeto:
 - No prompt do DOS:
 - `npm install @react-navigation/bottom-tabs`

React Native Tab Navigation



```
import * as React from 'react';
import { Text, View } from 'react-native';
import { NavigationContainer } from '@react-navigation/native';
import { createBottomTabNavigator } from '@react-navigation/bottom-tabs';
```

```
function HomeScreen() {
  return (
    <View style={{ flex: 1, justifyContent: 'center', alignItems: 'center' }}>
      <Text style={{fontWeight: 'bold',   fontSize: 40,}}>Principal!</Text>
    </View>
  );
}
```

```
function SettingsScreen() {
  return (
    <View style={{ flex: 1, justifyContent: 'center', alignItems: 'center' }}>
      <Text style={{fontWeight: 'bold',   fontSize: 40,}}>Configurações!</Text>
    </View>
  );
}
```

```
const Tab = createBottomTabNavigator();
```

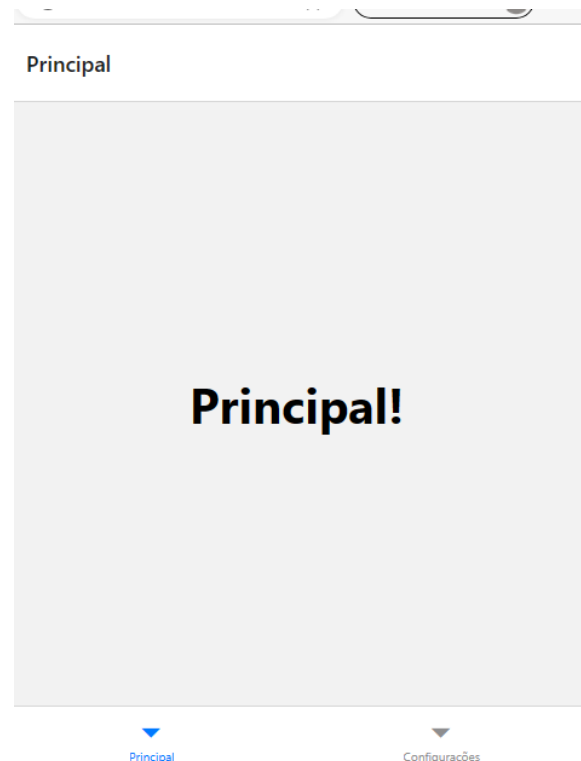
React Native Tab Navigation



```
function MyTabs() {  
  return (  
    <Tab.Navigator>  
      <Tab.Screen name="Principal" component={HomeScreen} />  
      <Tab.Screen name="Configurações" component={SettingsScreen} />  
    </Tab.Navigator>  
  );  
}  
  
export default function App() {  
  return (  
    <NavigationContainer>  
      <MyTabs />  
    </NavigationContainer>  
  );  
}
```

Executar o projeto ExemploNavigationTab01.

React Native Tab Navigation



React Native Tab Navigation - Customizando



```
import * as React from 'react';
import {Text, View } from 'react-native';
import {Icons } from '@expo/vector-icons';
import {createBottomTabNavigator} from '@react-navigation/bottom-tabs';
import { NavigationContainer } from '@react-navigation/native';
```

```
function HomeScreen() {
  return (
    <View style={{ flex: 1, justifyContent: 'center', alignItems: 'center' }}>
      <Text style={{fontWeight: 'bold',fontSize: 40,}}>Principal!</Text>
    </View>
  );
}
```

```
function SettingsScreen() {
  return (
    <View style={{ flex: 1, justifyContent: 'center', alignItems: 'center' }}>
      <Text style={{fontWeight: 'bold',fontSize: 40,}}>Configurações!</Text>
    </View>
  );
}
```

```
const Tab = createBottomTabNavigator();
```

Obs.: Ícones em <https://icon-sets.iconify.design/ion/>

React Native Tab Navigation - Customizando



```
export default function App() {  
  return (  
    <NavigationContainer>  
      <Tab.Navigator  
        screenOptions={({ route }) => ({  
          tabBarIcon: ({ focused, color, size }) => {  
            let iconName;  
            if (route.name === 'Principal') {  
              iconName = focused ? 'home': 'home-outline';  
            } else if (route.name === 'Configurações') {  
              iconName = focused ? 'ios-list' : 'ios-list';  
            }  
            return <Ionicons name={iconName} size={size} color={color} />;  
          },  
          tabBarActiveTintColor: 'tomato',  
          tabBarInactiveTintColor: 'gray',  
        })}  
      <  
        <Tab.Screen name="Principal" component={HomeScreen} />  
        <Tab.Screen name="Configurações" component={SettingsScreen} />  
      </Tab.Navigator>  
    </NavigationContainer>  
  );  
}
```

Executar o projeto ExemploNavigationTab02.

React Native Tab Navigation - Customizando



React Native Stack e Tab Navigation



```
import * as React from 'react';
import { Button, Text, View } from 'react-native';
import { NavigationContainer } from '@react-navigation/native';
import { Ionicons } from '@expo/vector-icons';
import { createBottomTabNavigator } from '@react-navigation/bottom-tabs';

function HomeScreen({ navigation }) {
  return (
    <View style={{ flex: 1, justifyContent: 'center', alignItems: 'center' }}>
      <Text style={{ fontWeight: 'bold', fontSize: 40,}}>Principal!</Text>
      <Button
        title="Vai Para Configurações" onPress={() => navigation.navigate('Configurações')}
      />
    </View>
  );
}
```

Obs.: Ícones em <https://icon-sets.iconify.design/ion/>

React Native Stack e Tab Navigation



```
function SettingsScreen({ navigation }) {  
  return (  
    <View style={{ flex: 1, justifyContent: 'center', alignItems: 'center' }}>  
      <Text style={{ fontWeight: 'bold', fontSize: 40, }}>Configurações!</Text>  
      <Button title="Vai Para Principal" onPress={() => navigation.navigate('Principal')} />  
    </View>  
  );  
}
```

```
const Tab = createBottomTabNavigator();
```

```
export default function App() {  
  return (  
    <NavigationContainer>  
      <Tab.Navigator  
        screenOptions={({ route }) => ({  
          tabBarIcon: ({ focused, color, size }) => {  
            let iconName;  
            if (route.name === 'Principal') {  
              iconName = focused ? 'home': 'home-outline';  
            } else if (route.name === 'Configurações') {  
              iconName = focused ? 'ios-list' : 'ios-list';  
            }  
          }  
        })
```

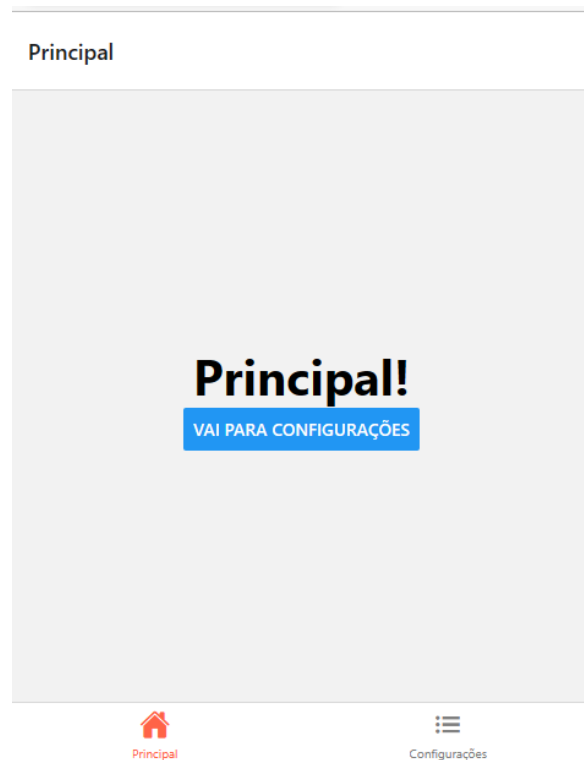
React Native Stack e Tab Navigation



```
return <Ionicons name={iconName} size={size} color={color} />;
    },
    tabBarActiveTintColor: 'tomato',
    tabBarInactiveTintColor: 'gray',
  }}}>
  <Tab.Screen name="Principal" component={HomeScreen} />
  <Tab.Screen name="Configurações" component={SettingsScreen} />
</Tab.Navigator>
</NavigationContainer>
);
}
```

Executar o projeto ExemploNavigationStackTab01.

React Native Stack e Tab Navigation



React Native API BackHandler



- A API do BackHandler detecta pressionamentos do **botão de hardware** para a navegação de retorno;
- Permite o registro *listeners* de eventos para a ação de retorno do sistema;
- **Válido apenas para Android;**
- As assinaturas de eventos são chamadas na ordem inversa (ou seja, a última assinatura registrada é chamada primeiro);
- Se uma assinatura retornar *true*, as assinaturas registradas anteriormente não serão chamadas;
- Métodos e tipos em:
<https://reactnative.dev/docs/backhandler#backhandler-hook>

React Native API BackHandler



- O `BackHandler.addListener` cria um *listener* de evento e retorna um objeto `NativeEventSubscription` que deve ser limpo usando o método `NativeEventSubscription.remove`;
- Métodos e tipos em:
<https://reactnative.dev/docs/backhandler#backhandler-hook> (vamos visitar!)

React Native API BackHandler



```
import React, { useEffect } from "react";
import { Text, View, BackHandler, Alert, StatusBar, StyleSheet } from "react-native";

const App = () => {

  useEffect(() => {

    const sair = () => {
      Alert.alert("Epa!", "Quer ir embora mesmo?", [
        {
          text: "NÃO",
          onPress: () => null,
          style: "cancel"
        },
        { text: "SIM", onPress: () => BackHandler.exitApp() }
      ]);
      return true;
    };

    const backHandler = BackHandler.addEventListener("hardwareBackPress", sair);

    return () => backHandler.remove();
  }, []);

  return (
    <View style={{marginVertical: 80 }} >
      <StatusBar hidden={false} />
      <Text style={styles.title}>Exemplo BackHandler - Toque no Voltar </Text>
    </View>
  );
};
```

React Native API BackHandler



```
const styles = StyleSheet.create({
  title: {
    textAlign: 'center',
    marginVertical: 8,
    fontSize: 18,
  },
});

export default App;
```

Criar e Executar o projeto ExemploBackHandler01.

React Native API Vibration



- Permite realizar a vibração do aparelho móvel;
- No Android, a duração da vibração é padronizada para 400 milissegundos, e uma duração de vibração arbitrária pode ser especificada passando um número como valor para o argumento padrão;
- No iOS, a duração da vibração é fixada em aproximadamente 400 milissegundos;
- O método `vibrate()` pode receber um argumento padrão com uma matriz de números que representam o tempo em milissegundos;
- Pode-se definir `repeat` como `true` para percorrer o padrão de vibração em um loop até que `cancel()` seja chamado;
- No Android, os índices ímpares da matriz padrão representam a duração da vibração, enquanto os pares representam o tempo de separação;
- No iOS, os números na matriz do padrão representam o tempo de separação, pois a duração da vibração é fixa;
- Informações sobre como usar em:
 - <https://reactnative.dev/docs/vibration>

React Native API Vibration



```
import React from 'react';
import { Button, Platform, Text, Vibration, View, SafeAreaView, StyleSheet, } from 'react-native';

const Separator = () => {
  return <View style={Platform.OS === 'android' ? styles.separator : null} />;
};

const App = () => {
  const ONE_SECOND_IN_MS = 1000;

  const PATTERN = [
    1 * ONE_SECOND_IN_MS,
    2 * ONE_SECOND_IN_MS,
    3 * ONE_SECOND_IN_MS,
  ];

  const PATTERN_DESC =
    Platform.OS === 'android' ? 'espera 1s, vibrate 2s, espera 3s' : 'espera 1s, vibrate, espera 2s, vibrate, espera 3s';

  return (
```

React Native API Vibration



```
<SafeAreaView style={styles.container}>
  <Text style={[styles.header, styles.paragraph]}>Vibration API</Text>
  <View>
    <Button title="Vibra Uma Vez" onPress={() => Vibration.vibrate()} />
  </View>
  <Separator />
  <View>
    <Button
      title="Vibra por 10 segundos"
      onPress={() => Vibration.vibrate(10 * ONE_SECOND_IN_MS)}
    />
  </View>
  <Separator />
  <Text style={styles.paragraph}>Padrão: {PATTERN_DESC}</Text>
  <Button
    title="Vibra pelo Padrão"
    onPress={() => Vibration.vibrate(PATTERN)}
  />
```

React Native API Vibration



```
<Separator />

  <Button

    title="Vibra pelo Padrão até Cancelar"

    onPress={() => Vibration.vibrate(PATTERN, true)}

  />

<Separator />

<Button

  title="Pára Vibração Padrão"

  onPress={() => Vibration.cancel()}

  color="#FF0000"

/>

</SafeAreaView>

);

};
```

React Native API Vibration



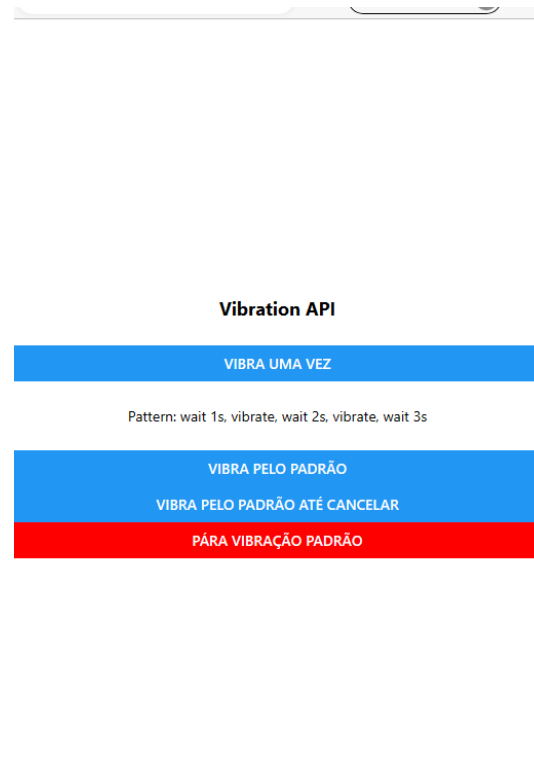
```
const styles = StyleSheet.create({
  container: {
    flex: 1,
    justifyContent: 'center',
    paddingTop: 44,
    padding: 8,
  },
  header: {
    fontSize: 18,
    fontWeight: 'bold',
    textAlign: 'center',
  },
  paragraph: {
    margin: 24,
    textAlign: 'center',
  },
  separator: {
    marginVertical: 8,
    borderBottomColor: '#737373',
    borderBottomWidth: StyleSheet.hairlineWidth,
  },
});
```

```
export default App;
```

React Native API Vibration



Executar projeto ExemploVibration no celular.



React Native Expo CheckBox



- O pacote fornece uma implementação de CheckBox que pode ser usada diretamente no projeto;
- Um CheckBox é um botão que existe em um dos dois estados — está marcado ou não;
- Isso o torna um candidato perfeito para o Hook `useState()`;
- Informações sobre como usar em:
 - <https://docs.expo.dev/ui-programming/implementing-a-checkbox/>

React Native

Expo CheckBox – Primeiro Exemplo



```
import Checkbox from 'expo-checkbox';
import React, { useState } from 'react';
import { StyleSheet, Text, View } from 'react-native';

export default function App() {
  const [isChecked, setChecked] = useState(false);

  return (
    <View style={styles.container}>
      <View style={styles.section}>
        <Checkbox style={styles.checkbox} value={isChecked} onChange={setChecked} color={isChecked ? '#ff0000' : undefined}/>
        <Text style={styles.paragraph}>Normal checkbox</Text>
      </View>
      <View style={styles.section}>
        <Checkbox
          style={styles.checkbox}
          value={isChecked}
          onChange={setChecked}
          color={isChecked ? '#4630EB' : undefined}
        />
      </View>
    </View>
  );
}
```

React Native Expo CheckBox – Primeiro Exemplo



```
    />  
    <Text style={styles.paragraph}>Custom colored checkbox</Text>  
  </View>  
  <View style={styles.section}>  
    <Checkbox style={styles.checkbox} disabled value={isChecked} onChange={setChecked} />  
    <Text style={styles.paragraph}>Disabled checkbox</Text>  
  </View>  
</View>  
);  
}
```

React Native Expo CheckBox – Primeiro Exemplo



```
const styles = StyleSheet.create({
  container: {
    flex: 1,
    marginHorizontal: 16,
    marginVertical: 32,
  },
  section: {
    flexDirection: 'row',
    alignItems: 'center',
  },
  paragraph: {
    fontSize: 15,
  },
  checkbox: {
    margin: 8,
  },
});
```

Criar e Executar o projeto ExemploCheckBox01 no celular.

React Native

Expo CheckBox – Segundo Exemplo



```
import React, { useState } from "react";
import { View, StyleSheet, Text, Button, Platform , Alert} from "react-native";
import CheckBox from "expo-checkbox";

export default function App() {
  const [agree, setAgree] = useState(false);

  return (
    <View style={styles.container}>
      <View style={styles.wrapper}>
        <CheckBox
          value={agree}
          onChange={() => setAgree(!agree)}
          color={agree ? "#4630EB" : undefined}
        />
        <Text style={styles.text}>
          Li e concordo com os termos e condições.
        </Text>
      </View>
      <Button
        title="Assinar"
        disabled={!agree}
        onPress={() => Alert.alert('Partiu!!!')}
      />
    </View>
  );
}
```

React Native

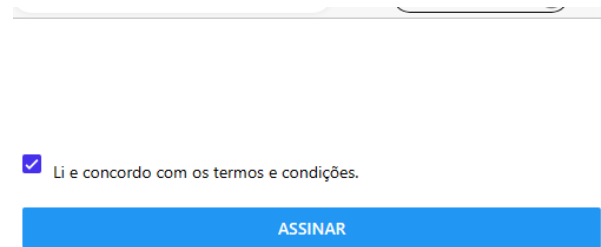
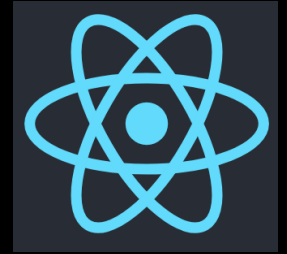
Expo CheckBox – Segundo Exemplo



```
const styles = StyleSheet.create({
  container: {
    width: "100%",
    padding: 16,
    paddingTop: 100,
  },
  wrapper: {
    display: "flex",
    flexDirection: "row",
    alignContent: "center",
    paddingVertical: 15,
  },
  text: {
    lineHeight: 30,
    marginLeft: 10,
  },
});
```

Criar e Executar o projeto ExemploCheckBox02 no celular.

React Native Expo CheckBox – Segundo Exemplo

A mockup of a mobile app interface. It features a white background with a thin gray header bar at the top. Below the header, there is a blue checkbox that is checked, followed by the text "Li e concordo com os termos e condições." in a small, gray font. At the bottom, there is a solid blue rectangular button with the word "ASSINAR" in white, uppercase letters.

React Native API Dimensions



- É atualizada conforme as dimensões da tela são atualizadas;
- Embora as dimensões estejam disponíveis imediatamente, elas podem mudar (por exemplo, devido à rotação do dispositivo, dispositivos dobráveis etc.);
- Possui um `addEventListener` para acompanhar as alterações no tamanho da tela;
- Métodos e tipos em:

<https://reactnative.dev/docs/dimensions>

React Native API Dimensions



```
import React, {useState, useEffect} from 'react';
import { SafeAreaView, StyleSheet, Text, View, Dimensions} from 'react-native';
```

```
const App = () => {
  const [height, setHeight] = useState("");
  const [width, setWidth] = useState("");
  useEffect(() => {
    setHeight(Dimensions.get('window').height);
    setWidth(Dimensions.get('window').width);
  }, []);
  return (
    <SafeAreaView style={{flex: 1}}>
      <View style={styles.container}>
        <Text style={styles.header}>
          React Native Dimensions{'\n'}
          Para obter a Altura e a Largura com React Native
        </Text>
        <Text style={styles.textStyle}>
          Altura: {height}
        </Text>
        <Text style={styles.textStyle}>
          Largura: {width}
        </Text>
      </View>
    </SafeAreaView>
  );
};
```

React Native API Dimensions



```
const styles = StyleSheet.create({
  container: {
    flex: 1,
    backgroundColor: 'white',
    justifyContent: 'center',
    padding: 10,
  },
  header: {
    fontSize: 22,
    fontWeight: '600',
    textAlign: 'center',
    paddingVertical: 20,
  },
  textStyle: {
    textAlign: 'center',
    fontSize: 18,
  },
});

export default App;
```

Executar o projeto ExemploDimension01.

React Native API Dimensions



React Native Dimensions
Para obter a Altura e a Largura com React
Native

Altura: 724
Largura: 479

React Native API PixelRatio



- PixelRatio fornece acesso à densidade de *pixels* e à escala da fonte do dispositivo;
- Deve-se obter uma imagem de resolução mais alta se estiver em um dispositivo de alta densidade de *pixels*;
- Uma boa regra é multiplicar o tamanho da imagem exibida pela proporção de *pixels*;
- Métodos e tipos em:

<https://reactnative.dev/docs/pixelratio>

React Native API PixelRatio



```
import React from 'react';

import { Image, PixelRatio, ScrollView, StyleSheet, Text, View, StatusBar } from 'react-native';

const size = 50;

const cat = {
  uri: 'https://reactnative.dev/docs/assets/p_cat1.png',
  width: size,
  height: size,
};
```

React Native API PixelRatio



```
const App = () => (  
  <ScrollView style={styles.scrollContainer}>  
    <View style={styles.container}>  
      <StatusBar hidden={false} />  
      <Text>Current Pixel Ratio is:</Text>  
      <Text style={styles.value}>{PixelRatio.get()}</Text>  
    </View>  
    <View style={styles.container}>  
      <Text>Current Font Scale is:</Text>  
      <Text style={styles.value}>{PixelRatio.getFontScale()}</Text>  
    </View>  
    <View style={styles.container}>  
      <Text>On this device images with a layout width of</Text>  
      <Text style={styles.value}>{size} px</Text>  
      <Image source={cat} />  
    </View>  
    <View style={styles.container}>  
      <Text>require images with a pixel width of</Text>  
      <Text style={styles.value}>  
        {PixelRatio.getPixelSizeForLayoutSize(size)} px  
      </Text>  
      <Image  
        source={cat}  
        style={{  
          width: PixelRatio.getPixelSizeForLayoutSize(size),  
          height: PixelRatio.getPixelSizeForLayoutSize(size),  
        }}  
      />  
    </View>  
  </ScrollView>  
>);
```

React Native API PixelRatio

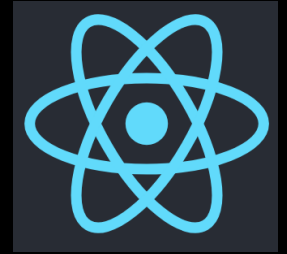


```
const styles = StyleSheet.create({
  scrollContainer: {
    flex: 1,
  },
  container: {
    justifyContent: 'center',
    alignItems: 'center',
  },
  value: {
    fontSize: 24,
    marginBottom: 12,
    marginTop: 4,
  },
});
```

```
export default App;
```

Executar o projeto ExemploPixelRatio01.

React Native API PixelRatio



Current Pixel Ratio is:

1

Current Font Scale is:

1

On this device images with a layout width of

50 px



require images with a pixel width of

50 px



React Native API Animated



- A biblioteca Animated foi projetada para tornar as animações fluidas, poderosas e fáceis de construir e manter;
- A função de animação vai determinar qual é o tipo de movimentação que veremos acontecer na tela, sendo que a API fornece 3 opções:
 - `Animated.timing()`: realiza a movimentação em um determinado intervalo de tempo.
 - `Animated.spring()`: cria um movimento oscilatório inspirado em uma mola.
 - `Animated.decay()`: o movimento começa com uma certa velocidade que diminui até parar.

React Native API Animated



- Para criar uma animação são necessárias, basicamente, 3 etapas:
 - Instanciar o valor da animação;
 - Configurar a função escolhida;
 - Atrelar a animação a um elemento e mudar suas propriedades com base no valor da animação.
- Métodos e tipos em:
<https://reactnative.dev/docs/animated>

React Native API Animated Primeiro Exemplo - Timing



```
import React, { useState, useEffect } from 'react';
import { Animated, View, Button, StatusBar, StyleSheet } from 'react-native';

export default function App() {

  const [redSquareAnim] = useState(new Animated.Value(0))

  const onPressTiming = () => {
    Animated.timing(redSquareAnim, {toValue: 200, duration: 1000}).start()
  }

  return (
    <View style={styles.container}>
      <StatusBar hidden={false} />
      <Button
        onPress={() => onPressTiming()}
        title = "Clique-me. Timing"
        color = "blue"
      />
    </View>
  )
}
```

React Native API Animated Primeiro Exemplo - Timing



```
<Animated.View style={{
  backgroundColor: 'red',
  height: 50,
  width: 50,
  marginBottom: 20,
  useNativeDriver: true,
  transform: [{translateY: redSquareAnim}]
}}/>
</View>
)
}
const styles = StyleSheet.create({
  container: {
    flex:1,
    alignItems: 'center',
  },
});
```

Criar e Executar o projeto ExemploAnimated01.

React Native API Animated - Spring



```
import React, { useState, useEffect } from 'react';
import { Animated, View, Button, StatusBar, StyleSheet } from 'react-native';

export default function App() {

  const [redSquareAnim] = useState(new Animated.Value(0))

  const onPressSpring = () => {
    Animated.spring(redSquareAnim, {toValue: 200, speed: 10, bounciness: 20}).start()
  }

  return (
    <View style={styles.container}>
      <StatusBar hidden={false} />
      <Button
        onPress={() => onPressSpring()}
        title = "Clique-me. Spring"
        color = "blue"
      />
    </View>
  )
}
```

React Native API Animated - Spring



```
<Animated.View style={{  
  backgroundColor: 'red',  
  height: 50,  
  width: 50,  
  marginBottom: 20,  
  transform: [{translateY: redSquareAnim}]  
}}/>  
  </View>  
)  
}  
const styles = StyleSheet.create({  
  container: {  
    flex: 1,  
    alignItems: 'center',  
  },  
});
```

Criar e Executar o projeto ExemploAnimated02.

React Native API Animated - Decay



```
import React, { useState, useEffect } from 'react';
import { Animated, View, Button, StatusBar, StyleSheet } from 'react-native';

export default function App() {

  const [redSquareAnim] = useState(new Animated.Value(0))

  const onPressDecay = () => {
    Animated.decay(redSquareAnim, {velocity: 0.41, deceleration: 0.998}).start()
  }

  return (
    <View style={styles.container}>
      <StatusBar hidden={false} />
      <Button
        onPress={() => onPressDecay()}
        title = "Clique-me. Decay"
        color = "blue"
      />
    </View>
  )
}
```

React Native API Animated - Decay



```
<Animated.View style={{  
  backgroundColor: 'red',  
  height: 50,  
  width: 50,  
  marginBottom: 20,  
  transform: [{translateY: redSquareAnim}]  
}}/>  
  </View>  
)  
}  
const styles = StyleSheet.create({  
  container: {  
    flex: 1,  
    alignItems: 'center',  
  },  
});
```

Executar o projeto ExemploAnimated03.

React Native API Animated – Tudo Junto



```
import React, { useState, useEffect } from 'react';
import { Animated, View, Button, StatusBar, StyleSheet } from 'react-native';

export default function App() {

  const [redSquareAnim] = useState(new Animated.Value(0))
  const [blueSquareAnim] = useState(new Animated.Value(0))
  const [blackSquareAnim] = useState(new Animated.Value(0))

  const onPressDecay = () => {
    Animated.decay(blueSquareAnim, { velocity: 0.41, deceleration: 0.999, useNativeDriver: true }).start()
  }
  const onPressSpring = () => {
    Animated.spring(blackSquareAnim, { toValue: 400, speed: 10, bounciness: 20, useNativeDriver: true }).start()
  }
  const onPressTiming = () => {
    Animated.timing(redSquareAnim, { toValue: 400, duration: 1000, useNativeDriver: true }).start()
  }
}
```

React Native API Animated - Tudo Junto



```
return (  
  
  <View style={styles.container}>  
    <StatusBar hidden={false} />  
    <View style={{flexDirection: 'row'}}>  
      <Button  
        onPress={() => onPressTiming()}  
        title = "Timing"  
        color = "red"  
      />  
      <Button  
        onPress={() => onPressDecay()}  
        title = "Decay"  
        color = "blue"  
      />  
      <Button  
        onPress={() => onPressSpring()}  
        title = "Spring"  
        color = "black"  
      />  
    </View>  
  </View>
```

React Native API Animated – Tudo Junto



```
<View style={{flexDirection: 'row',padding: 20,}}>
  <Animated.View style={{
    backgroundColor: 'red',
    height: 50,
    width: 50,
    marginBottom: 20,
    transform: [{translateY: redSquareAnim}]
  }}/>
  <Animated.View style={{
    backgroundColor: 'blue',
    height: 50,
    width: 50,
    marginBottom: 20,
    transform: [{translateY: blueSquareAnim}]
  }}/>
  <Animated.View style={{
    backgroundColor: 'black',
    height: 50,
    width: 50,
    marginBottom: 20,
    transform: [{translateY: blackSquareAnim}]
  }}/>
</View>
</View>
```

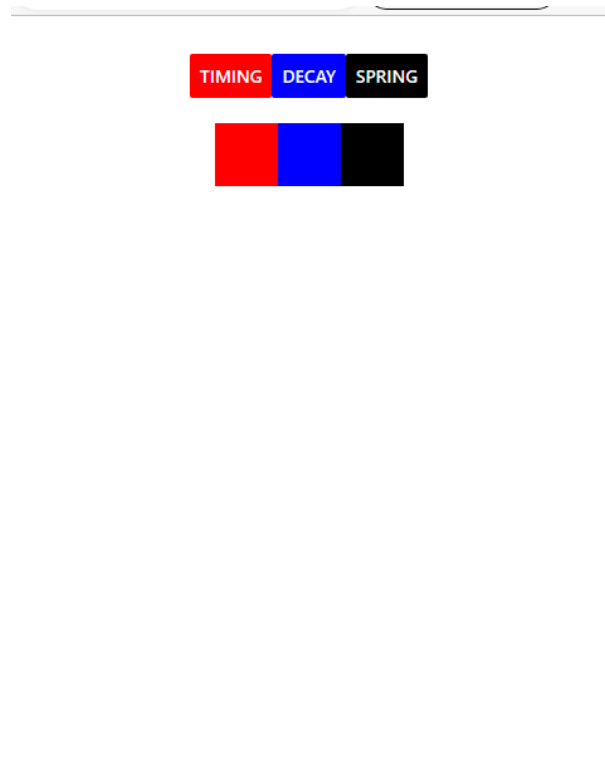
React Native API Animated – Tudo Junto



```
const styles = StyleSheet.create({  
  container: {  
    flex:1,  
    alignItems: 'center',  
    padding: 10,  
    margin: 20,  
  },  
});
```

Criar e Executar o projeto ExemploAnimated04.

React Native API Animated – Tudo Junto



React Native API NetInfo



- Permite obter informações sobre a rede informando:
 - Tipo de conexão;
 - Qualidade da conexão;
- Pacote desenvolvido por terceiros;
- Para instalar o pacote, sob diretório do projeto emitir(prompt do DOS):
 - `npm install --save @react-native-community/netinfo`
- Informações sobre como usar em:
 - <https://github.com/react-native-netinfo/react-native-netinfo/blob/master/README.md>

React Native API NetInfo Android permissões



READ_CALENDAR: 'android.permission.READ_CALENDAR'
WRITE_CALENDAR: 'android.permission.WRITE_CALENDAR'
CAMERA: 'android.permission.CAMERA'
READ_CONTACTS: 'android.permission.READ_CONTACTS'
WRITE_CONTACTS: 'android.permission.WRITE_CONTACTS'
GET_ACCOUNTS: 'android.permission.GET_ACCOUNTS'
ACCESS_FINE_LOCATION: 'android.permission.ACCESS_FINE_LOCATION'
ACCESS_COARSE_LOCATION: 'android.permission.ACCESS_COARSE_LOCATION'
ACCESS_BACKGROUND_LOCATION: 'android.permission.ACCESS_BACKGROUND_LOCATION'
RECORD_AUDIO: 'android.permission.RECORD_AUDIO'
READ_PHONE_STATE: 'android.permission.READ_PHONE_STATE'
CALL_PHONE: 'android.permission.CALL_PHONE'
READ_CALL_LOG: 'android.permission.READ_CALL_LOG'
WRITE_CALL_LOG: 'android.permission.WRITE_CALL_LOG'

React Native API NetInfo Android permissões



ADD_VOICEMAIL: 'com.android.voicemail.permission.ADD_VOICEMAIL'

USE_SIP: 'android.permission.USE_SIP'

PROCESS_OUTGOING_CALLS: 'android.permission.PROCESS_OUTGOING_CALLS'

BODY_SENSORS: 'android.permission.BODY_SENSORS'

SEND_SMS: 'android.permission.SEND_SMS'

RECEIVE_SMS: 'android.permission.RECEIVE_SMS'

READ_SMS: 'android.permission.READ_SMS'

RECEIVE_WAP_PUSH: 'android.permission.RECEIVE_WAP_PUSH'

RECEIVE_MMS: 'android.permission.RECEIVE_MMS'

READ_EXTERNAL_STORAGE: 'android.permission.READ_EXTERNAL_STORAGE'

WRITE_EXTERNAL_STORAGE: 'android.permission.WRITE_EXTERNAL_STORAGE'

BLUETOOTH_CONNECT: 'android.permission.BLUETOOTH_CONNECT'

BLUETOOTH_SCAN: 'android.permission.BLUETOOTH_SCAN'

BLUETOOTH_ADVERTISE: 'android.permission.BLUETOOTH_ADVERTISE'

ACCESS_MEDIA_LOCATION: 'android.permission.ACCESS_MEDIA_LOCATION'

React Native API NetInfo Android permissões



ACCEPT_HANOVER: 'android.permission.ACCEPT_HANOVER'

ACTIVITY_RECOGNITION: 'android.permission.ACTIVITY_RECOGNITION'

ANSWER_PHONE_CALLS: 'android.permission.ANSWER_PHONE_CALLS'

READ_PHONE_NUMBERS: 'android.permission.READ_PHONE_NUMBERS'

UWB_RANGING: 'android.permission.UWB_RANGING'

BODY_SENSORS_BACKGROUND: 'android.permission.BODY_SENSORS_BACKGROUND'

READ_MEDIA_IMAGES: 'android.permission.READ_MEDIA_IMAGES'

READ_MEDIA_VIDEO: 'android.permission.READ_MEDIA_VIDEO'

READ_MEDIA_AUDIO: 'android.permission.READ_MEDIA_AUDIO'

POST_NOTIFICATIONS: 'android.permission.POST_NOTIFICATIONS'

NEARBY_WIFI_DEVICES: 'android.permission.NEARBY_WIFI_DEVICES'

READ_VOICEMAIL: 'com.android.voicemail.permission.READ_VOICEMAIL',

WRITE_VOICEMAIL: 'com.android.voicemail.permission.WRITE_VOICEMAIL',

React Native API NetInfo AndroidManifest.xml



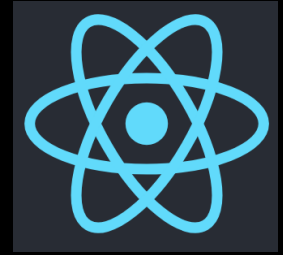
```
<uses-permission android:name="android.permission.INTERNET"/>
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"/>
<uses-permission android:name="android.permission.ACCESS_WIFI_STATE"/>
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/>

<uses-permission android:name="android.permission.SYSTEM_ALERT_WINDOW"/>
<uses-permission android:name="android.permission.VIBRATE"/>

<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE"/>
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
```

Obs.: Caminho para o AndroidManifest.xml: **DiretórioProjeto** → **android** → **app** → **src** → **main**
Arquivo usado no próximo exemplo.

React Native Exemplo Real AndroidManifest.xml



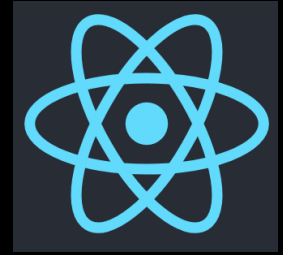
```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.principal.beezoo"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk
        android:minSdkVersion="16"
        android:targetSdkVersion="19" />

    <uses-permission android:name="android.permission.WAKE_LOCK" ></uses-permission>
    <uses-permission android:name="android.permission.RECEIVE_BOOT_COMPLETED" ></uses-permission>
    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" ></uses-permission>
    <uses-permission android:name="android.permission.RECORD_AUDIO" ></uses-permission>

    <application
        android:allowBackup="true"
        android:icon="@drawable/ic_adb_black_48dp"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
        <activity
            android:name=".MenuPrincipalActivity"
            android:label="@string/app_name"
            android:screenOrientation="portrait" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <activity
            android:name=".CriarDataBaseActivity"
            android:label="@string/title_activity_criar_data_base" >
        </activity>
```

React Native API NetInfo - Exemplo



```
import {useNetInfo} from "@react-native-community/netinfo";
import {View, Text, Button, StatusBar,PermissionsAndroid, StyleSheet, Alert} from "react-native";
import React, {useState, useEffect} from 'react';

const TesteNetInfo = () => {
  const netInfo = useNetInfo();
  const [autorizado,setAutorizado] = useState(false);

  const requestNetPermission = 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); }
  };
};
```

React Native API NetInfo - Exemplo



```
const MinhaRede = () => {  
  return (  
    <View style={styles.container}>  
      <Text>Tipo: {netInfo.type}</Text>  
      <Text>{netInfo.isConnected ? "Conectado" : "Não"}</Text>  
      <Text>{netInfo.isWifiEnabled ? "WiFi-ON" : "WiFi-OFF"}</Text>  
      <Text>{netInfo.isInternetReachable ? "Internet_ON" : "Internet_OFF"}</Text>  
      <Text>{(netInfo.isWifiEnabled) ? (<Text>Força do Sinal: {netInfo.details.strength}</Text>) : "" }</Text>  
      <Text>{(netInfo.isWifiEnabled) ? (<Text>ssid: {netInfo.details.ssid}</Text>) : "" }</Text>  
      <Text>{(netInfo.isWifiEnabled) ? (<Text>ipAddress: {netInfo.details.ipAddress}</Text>) : "" }</Text>  
      <Text>{(netInfo.isWifiEnabled) ? (<Text>Link Speed: {netInfo.details.linkSpeed}</Text>) : "" }</Text>  
      <Text>{(netInfo.isWifiEnabled) ? (<Text>Operadora: {netInfo.details.carrier}</Text>) : "" }</Text>  
    </View>  
  );  
};
```

React Native API NetInfo - Exemplo



```
return (  
  <View style={styles.container}>  
    <StatusBar hidden={false} />  
    <Button title="Pedir Permissão" onPress={requestNetPermission} />  
    {autorizado ? (<MinhaRede/>) : (<Text></Text>)}  
  </View>  
);  
};  
const styles = StyleSheet.create({  
  container: {  
    flex:1,  
    alignItems: 'center',  
  },  
});  
  
export default TesteNetInfo;
```

Obs.: **Para obter o ssid**: no iOS, definir a opção de configuração shouldFetchWiFiSSID; no Android, definir a permissão ACCESS_FINE_LOCATION no AndroidManifest.xml.

Executar projeto ExemploNetInfo no celular.

React Native Construindo um APK



- O APK (Android Application Pack) representa o pacote da aplicação a ser instalado em um aparelho Android;
- O IPA (iOS App Store Package) é o pacote similar para o iOS;
- Passos a seguir:

- Abrir uma conta no Expo:
 - <https://expo.dev/signup>
- Emitir, no prompt do DOS, o comando global:
 - `npm install -g eas-cli`
- No arquivo app.json, adicionar conjunto de configurações abaixo, antes de executar o comando `build`:

```
{
  "expo": {
    "name": "Your App Name",
    "icon": "./path/to/your/app-icon.png",
    "version": "1.0.0",
    "slug": "your-app-slug",
    "ios": {
      "bundleIdentifier": "com.yourcompany.yourappname",
      "buildNumber": "1.0.0" },
    "android": {
      "package": "com.yourcompany.yourappname",
      "versionCode": 1
    }
  }
}
```

React Native Construindo um APK



■ Passos a seguir:

- Emitir, no prompt do DOS, sob o diretório do projeto alvo:
 - **eas build -p android**
- Após a execução deste comando será gerado um arquivo . AAB, novo arquivo de distribuição.
- Para gerar um .apk, modifique o arquivo gerado eas.json, adicionando as propriedades:

```
{
  "build": {
    "preview": {
      "android": {
        "buildType": "apk"
      }
    },
    "preview2": {
      "android": {
        "gradleCommand": ":app:assembleRelease"
      }
    },
    "preview3": {
      "developmentClient": true
    },
    "production": {}
  }
}
```

eas = Expo Application Services

React Native Construindo um APK



■ Passos a seguir(continuação):

- Emitir, no prompt do DOS, sob o diretório do projeto alvo:
 - `eas build -p android --profile preview`
- Após o fim desta operação o arquivo apk será gerado no site do Expo com a identificação `Android Play Store build`
- Fazer o download do apk gerado para algum desktop.
- Após o download, fazer o upload para o Google Drive:
- No celular Android, entrar no Google Drive e baixar o aplicativo.
- Como é um apk ,não baixado do Play Store Google, habilitar, no celular, a permissão para utilizar apks de terceiros.
- Link como referência para geração do apk:
 - <https://docs.expo.dev/build-reference/apk/>
- Os comandos de geração levam um expressivo tempo para executarem no plano gratuito do Expo(é o meu!).
- O plano gratuito inclui 30 compilações (até 15 iOS) por mês em uma fila de baixa prioridade;
- <https://expo.dev/pricing> (para conhecer os valores cobrados)

React Native

Exercício A7



- Crie e execute um projeto utilizando a facilidade de navegação por guias;
- Este projeto unirá o exemplo 'Consulta CEP' com o Exercício A6;
- Na tela principal haverá as guias:
 - Principal, CEP, Carros.
- Ao clicar em CEP, desvia e executa a tela de CEPs;
- Ao clicar em Carros, desvia e executa a tela de carros;
- Na tela principal, coloquem alguma imagem em *background*;

React Native API Expo AsyncStorage



- AsyncStorage é um sistema de armazenamento não criptografado, assíncrono, **persistente** e de par key-value que **é global** para o aplicativo.
- Pacote desenvolvido por terceiros;
- Onde os dados são armazenados:
 - Android - SQLite
 - iOS - valores pequenos (não excedendo 1.024 caracteres) são serializados e armazenados em um arquivo manifest.json comum, enquanto valores maiores são armazenados em arquivos dedicados individuais (denominados como chave de hash MD5)
 - macOS - Igual ao iOS
 - Web - window.localStorage
 - Windows - SQLite
- Para instalar o pacote, sob diretório do projeto emitir(prompt do DOS):
 - **npm install @react-native-async-storage/async-storage**
- Informações sobre como usar em:
 - <https://react-native-async-storage.github.io/async-storage/docs/usage>

React Native API Expo AsyncStorage



- Somente pode armazenar dados do tipo *string*;
- Para armazenar os dados de um objeto, precisa serializá-los primeiro;
- Para dados que podem ser serializados para JSON, usar `JSON.stringify()` ao salvar os dados e `JSON.parse()` para carregá-los.
- Exemplo com valor em string:

```
const storeData = async (value) => {  
  try {  
    await AsyncStorage.setItem('my-key', value);  
  } catch (e) {  
    // saving error  
  }  
};  
  
const getData = async () => {  
  try {  
    const value = await AsyncStorage.getItem('my-key');  
    if (value !== null) {  
      // value previously stored  
    }  
  } catch (e) {  
    // error reading value  
  }  
};
```

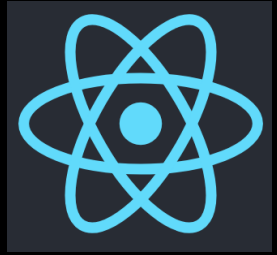
React Native API Expo AsyncStorage



■ Exemplo com objeto:

```
const storeData = async (value) => {  
  try {  
    const jsonValue = JSON.stringify(value);  
    await AsyncStorage.setItem('my-key', jsonValue);  
  } catch (e) {  
    // saving error  
  }  
};  
  
const getData = async () => {  
  try {  
    const jsonValue = await AsyncStorage.getItem('my-key');  
    return jsonValue != null ? JSON.parse(jsonValue) : null;  
  } catch (e) {  
    // error reading value  
  }  
};
```

React Native API Expo AsyncStorage



```
import React from 'react';
import {View,Text, StyleSheet, TextInput, StatusBar, Button} from 'react-native';
import AsyncStorage from '@react-native-async-storage/async-storage';

const TextInputExample = () => {
  const [token,onChangeToken] = React.useState("");

  const Separator = () => <View style={styles.separator} />;

  const apertouGravar = async () => {
    try {
      await AsyncStorage.setItem('token', token);
    } catch (e) {
      console.log('Deu ruim!!!');
    }
  };

  const apertouLer = async () => {
    try {
      const token = await AsyncStorage.getItem('token');
      if (token !== null) {
        alert(token);
      }
    } catch (e) {
      console.log('Deu ruim!!!');
    }
  };
};
```

React Native API Expo AsyncStorage



```
return (  
  <View>  
    <StatusBar hidden={false} />  
    <Text style={styles.titulo}>  
      Texto para Armazenar:  
    </Text>  
    <TextInput  
      style={styles.input}  
      onChangeText={onChangeToken}  
      value={token}  
    />  
    <Separator/>  
    <Button  
      onPress={() => apertouGravar()}  
      title = "Clique-me para Gravar."  
      color = "red"  
    />  
    <Separator/>  
    <Button  
      onPress={() => apertouLer()}  
      title = "Clique-me para Ler"  
      color = "green"  
    />  
  </View>  
)  
};
```

React Native API Expo AsyncStorage



```
const styles = StyleSheet.create({
  input: {
    height: 40,
    margin: 12,
    borderWidth: 1,
    padding: 10,
  },
  titulo: {
    margin: 6,
    padding: 5,
    fontWeight: 'bold',
  },
  separator: {
    marginVertical: 8,
    //borderBottomColor: '#737373',
    //borderBottomWidth: StyleSheet.hairlineWidth,
  },
});

export default TextInputExample;
```

Executar o projeto ExemploAsyncStorage01(no celular!).

MQTT

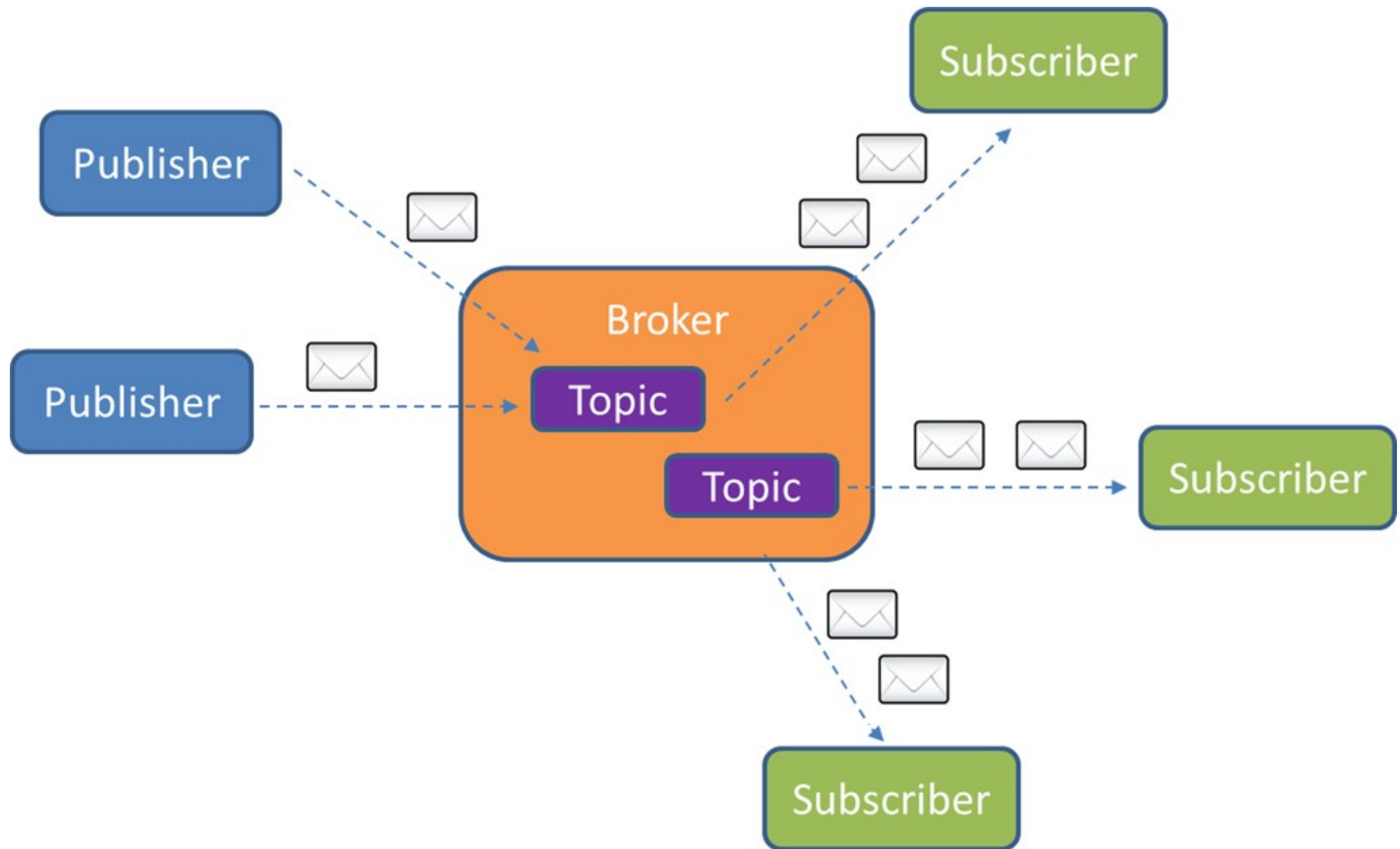


■ Message Queue Telemetry Transport(MQTT);

- Protocolo de transporte de mensagens no modelo publicador-subscritor
- Ideal para a comunicação em IoT
- Baseado em TCP/IP
- Leve, aberto e de fácil implementação
- Criado em 1999 por Andy Stanford-Clark (IBM) and Arlen Nipper (Arcom, agora Cirrus Link)
- Qualidade no serviço da entrega do dado

MQTT

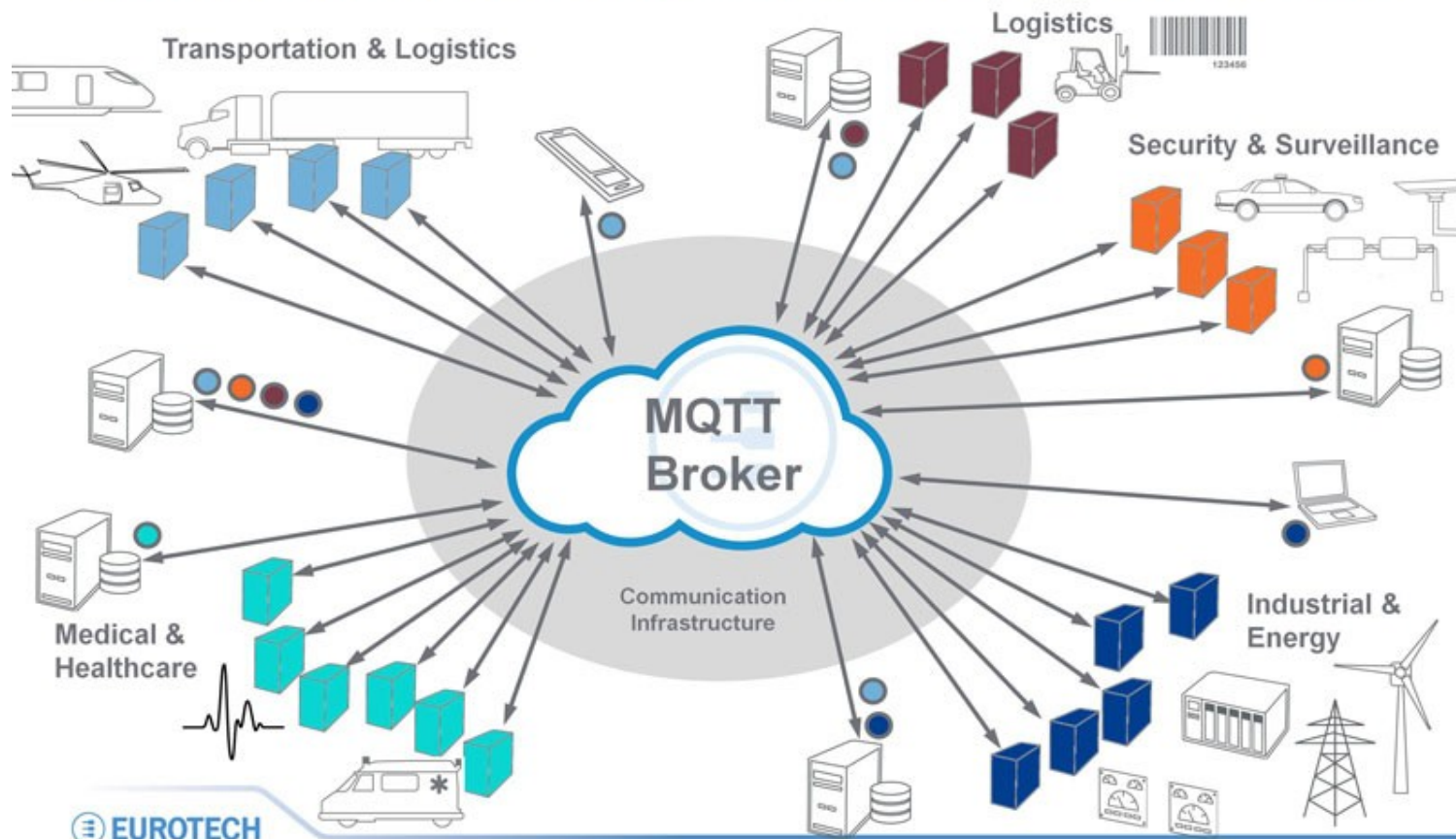
Message Queue Telemetry Transport



MQTT



The Internet of Things Decoupling Producers & Consumers of M2M Device Data



MQTT - *Brokers*



■ Alguns *Brokers* disponíveis:

- HiveMq
- CloudMQTT
- **Mosquitto**
- EMQ
- iot.eclipse.org
- Amazon 's Aws IoT
- Adafruit IO

MQTT - Mosquitto



- Disponível para *download* em mosquitto.org;
- Mantido pelo grupo do eclipse;
- Suporte às linguagens:
 - C
 - C++
 - C# e .Net
 - Java/Android
 - Phyton
 - JavaScript
 - Go

MQTT - Mosquitto



■ Algumas das funcionalidades:

- Autenticação por usuário/senha
- Especificação da qualidade da transmissão(QoS):
 - 0 – manda, no máximo, uma vez;
 - Não há garantia de entrega. O destinatário não confirma o recebimento da mensagem e a mensagem não é armazenada e retransmitida pelo remetente. **O nível 0 de QoS costuma ser chamado de "dispare e esqueça".**

MQTT - Mosquitto



■ Especificação da qualidade da transmissão(QoS):

1 – manda, pelo menos uma vez, com a garantia de entrega;

- Garante que uma mensagem seja entregue pelo menos uma vez ao receptor. O remetente armazena a mensagem até receber um pacote PUBACK do destinatário que confirma o recebimento da mensagem. É possível que uma mensagem seja enviada ou entregue várias vezes.
- O remetente usa o identificador de pacote em cada pacote para corresponder o pacote PUBLISH ao pacote PUBACK correspondente. Se o remetente não receber um pacote PUBACK em um período de tempo razoável, o remetente reenvia o pacote PUBLISH. Quando um receptor recebe uma mensagem com QoS 1, ele pode processá-la imediatamente.
- Se o cliente de publicação enviar a mensagem novamente, ele definirá um sinalizador duplicado (DUP). No QoS 1, este sinalizador DUP é usado apenas para fins internos e não é processado pelo broker ou cliente. O destinatário da mensagem envia um PUBACK, independentemente do sinalizador DUP

MQTT - Mosquitto



■ Especificação da qualidade da transmissão(QoS):

2 – manda, somente uma vez, com a garantia de entrega;

- QoS 2 é o mais alto nível de serviço em MQTT. **Este nível garante que cada mensagem seja recebida apenas uma vez pelos destinatários pretendidos. É mais seguro e mais lento.** A garantia é fornecida por pelo menos dois fluxos de solicitação/resposta (um handshake de quatro partes) entre o remetente e o destinatário. O remetente e o destinatário usam o identificador de pacote da mensagem PUBLISH original para coordenar a entrega da mensagem.
- Quando um receptor recebe um pacote de um remetente, ele processa a mensagem de publicação de acordo e responde ao remetente com um pacote PUBREC que reconhece o pacote PUBLISH. Se o remetente não receber um pacote PUBREC do destinatário, ele enviará o pacote PUBLISH novamente com um sinalizador duplicado (DUP) até receber uma confirmação.
- Depois que o remetente recebe um pacote PUBREC do destinatário, o remetente pode descartar com segurança o pacote PUBLISH inicial. O remetente armazena o pacote PUBREC do destinatário e responde com um pacote PUBREL.
- Depois que o receptor recebe o pacote PUBREL, ele pode descartar todos os estados armazenados e responder com um pacote PUBCOMP (o mesmo ocorre quando o remetente recebe o PUBCOMP). Até que o destinatário conclua o processamento e envie o pacote PUBCOMP de volta ao remetente, o destinatário armazena uma referência ao identificador de pacote do pacote PUBLISH original. Esta etapa é importante para evitar o processamento da mensagem uma segunda vez. Após o remetente receber o pacote PUBCOMP, o identificador de pacote da mensagem publicada fica disponível para reutilização.

MQTT - Mosquitto



■ QoS - Considerações:

- O cliente que publica a mensagem para o *broker* define o nível de QoS da mensagem ao enviar a mensagem para o *broker*;
- O *broker* transmite esta mensagem aos clientes assinantes usando o nível de QoS que cada cliente assinante define durante o processo de assinatura;
- Se o cliente assinante definir uma QoS inferior à do cliente publicador, o **broker** transmitirá a mensagem com qualidade de serviço inferior;
- O QoS dá ao cliente o poder de escolher um nível de serviço que corresponda à confiabilidade da rede e à lógica do aplicativo;
- Como o MQTT gerencia a retransmissão de mensagens e garante a entrega (mesmo quando o transporte subjacente não é confiável), a QoS facilita muito a comunicação em redes não confiáveis.

MQTT - Mosquitto



■ QoS – Qual escolher:

Use QoS 0 quando:

Você tem uma conexão completamente ou quase estável entre o remetente e o destinatário. Você não se importa se algumas mensagens forem perdidas ocasionalmente. A perda de algumas mensagens pode ser aceitável se os dados não forem tão importantes ou quando os dados forem enviados em intervalos curtos. Você não precisa de enfileiramento de mensagens. As mensagens são enfileiradas apenas para clientes desconectados se tiverem QoS 1 ou 2 e uma sessão persistente.

Use QoS 1 quando:

Você precisa obter todas as mensagens e seu caso de uso pode lidar com duplicatas. O nível 1 de QoS é o nível de serviço usado com mais frequência porque garante que a mensagem chegue pelo menos uma vez, mas permite várias entregas. Obviamente, seu aplicativo deve tolerar duplicatas e ser capaz de processá-las adequadamente.

Use QoS 2 quando:

É fundamental para seu aplicativo receber todas as mensagens exatamente uma vez. Geralmente, esse é o caso se uma entrega duplicada puder prejudicar os usuários do aplicativo ou os clientes assinantes. Esteja ciente da sobrecarga e de que a interação QoS 2 leva mais tempo para ser concluída.

MQTT - Mosquitto



■ Algumas das funcionalidades:

- Guarda da última mensagem recebida(Retain);
 - Até que a próxima mensagem seja publicada, o cliente assinante está totalmente no escuro sobre o status atual do tópico. Essa situação é onde as mensagens retidas entram em ação.
- Mensagem específica para indisponibilidade(willMessage);
 - O *broker* informa que o publicador está fora do ar.
- Arquivo com opções de configuração;
- Suporte à criptografia usando as opções baseadas em certificados SSL / TLS;
- Porta padrão 1883;
- Tamanho padrão das mensagens → 128 bytes
- *Keep alive* padrão → 15 segundos(para alterar, editar PubSubClient.h)

MQTT

Códigos de Retorno



`int state ()`

Returns the current state of the client. If a connection attempt fails, this can be used to get more information about the failure.

All of the values have corresponding constants defined in `PubSubClient.h`.

Returns

- `-4 : MQTT_CONNECTION_TIMEOUT` - the server didn't respond within the keepalive time
- `-3 : MQTT_CONNECTION_LOST` - the network connection was broken
- `-2 : MQTT_CONNECT_FAILED` - the network connection failed
- `-1 : MQTT_DISCONNECTED` - the client is disconnected cleanly
- `0 : MQTT_CONNECTED` - the client is connected
- `1 : MQTT_CONNECT_BAD_PROTOCOL` - the server doesn't support the requested version of MQTT
- `2 : MQTT_CONNECT_BAD_CLIENT_ID` - the server rejected the client identifier
- `3 : MQTT_CONNECT_UNAVAILABLE` - the server was unable to accept the connection
- `4 : MQTT_CONNECT_BAD_CREDENTIALS` - the username/password were rejected
- `5 : MQTT_CONNECT_UNAUTHORIZED` - the client was not authorized to connect

React Native API MQTT



- Permite utilizar o serviço MQTT;
- Pacote desenvolvido por terceiros;
- Para instalar o pacote, sob diretório do projeto emitir(prompt do DOS):
 - `npm install @react-native-async-storage/async-storage @rneui/base @rneui/themed`
 - `npm install react_native_mqtt`
- Informações sobre como usar em:
 - <https://www.emqx.com/en/blog/how-to-use-mqtt-in-react-native>

React Native API MQTT



■ Instruções para uso:

- Criar uma instância do cliente e definir o *broker*:

```
init({  
  size: 10000,  
  storageBackend: AsyncStorage,  
  defaultExpires: 1000 * 3600 * 24,  
  enableCache: true,  
  sync : {}  
});
```

```
const options = {  
  host: 'broker.emqx.io',  
  port: 8083,  
  path: '/testTopic',  
  id: 'id_' + parseInt(Math.random()*100000)  
};
```

```
client = new Paho.MQTT.Client(options.host, options.port, options.path);
```

React Native API MQTT



■ Instruções para uso:

– Conectar-se ao broker:

```
connect = () => {  
  this.setState(  
    { status: 'isFetching' },  
    () => {  
      client.connect({  
        onSuccess: this.onConnect,  
        useSSL: false,  
        timeout: 3,  
        onFailure: this.onFailure  
      });  
    }  
  );  
}
```

React Native API MQTT



■ Instruções para uso:

– Publicar em um tópico:

```
sendMessage = () =>{  
  var message = new Paho.MQTT.Message(options.id + ':' + this.state.message);  
  message.destinationName = this.state.subscribedTopic;  
  client.send(message);  
}
```


React Native API MQTT



■ Instruções para uso:

– Assinar um tópico:

```
subscribeTopic = () => {  
  this.setState(  
    { subscribedTopic: this.state.topic },  
    () => {  
      client.subscribe(this.state.subscribedTopic, { qos: 0 });  
    }  
  );  
}
```

React Native API MQTT



■ Instruções para uso:

– Cancelar uma assinatura:

```
unSubscribeTopic = () => {  
  client.unsubscribe(this.state.subscribedTopic);  
  this.setState({ subscribedTopic: " " });  
}
```

React Native API MQTT



```
import React, { Component } from 'react';
import { View, Text, StyleSheet, FlatList, } from 'react-native';
import { Input, Button } from '@rneui/base';
import AsyncStorage from '@react-native-async-storage/async-storage';
import init from 'react_native_mqtt';
```

```
init({
  size: 10000,
  storageBackend: AsyncStorage,
  defaultExpires: 1000 * 3600 * 24,
  enableCache: true,
  sync : {}
});
const options = {
  host: 'broker.emqx.io',
  port: 8083,
  path: '/testTopic',
  id: 'id_' + parseInt(Math.random()*100000)
};
```

```
client = new Paho.MQTT.Client(options.host, options.port, options.path);
```

React Native API MQTT

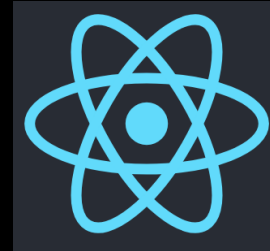


```
class App extends Component {
  constructor(props){
    super(props)
    this.state={
      topic: 'testTopic',
      subscribedTopic: '',
      message: '',
      messageList: [],
      status: ''
    };
    client.onConnectionLost = this.onConnectionLost;
    client.onMessageArrived = this.onMessageArrived;
  }

  onConnect = () => {
    console.log('onConnect');
    this.setState({ status: 'connected' });
  }

  onFailure = (err) => {
    console.log('Connect failed!');
    console.log(err);
    this.setState({ status: 'failed' });
  }
}
```

React Native API MQTT



```
connect = () => {
  this.setState(
    { status: 'isFetching' },
    () => {
      client.connect({
        onSuccess: this.onConnect,
        useSSL: false,
        timeout: 3,
        onFailure: this.onFailure
      });
    }
  );
}

onConnectionLost=(responseObject)=>{
  if (responseObject.errorCode !== 0) {
    console.log('onConnectionLost:' + responseObject.errorMessage);
  }
}

onMessageArrived = (message )=> {
  console.log('onMessageArrived:' + message.payloadString);
  newmessageList = this.state.messageList;
  newmessageList.unshift(message.payloadString);
  this.setState({ messageList: newmessageList });
}
```

React Native API MQTT



```
onChangeTopic = (text) => {  
  this.setState({ topic: text });  
}
```

```
subscribeTopic = () => {  
  this.setState(  
    { subscribedTopic: this.state.topic },  
    () => {  
      client.subscribe(this.state.subscribedTopic, { qos: 0 });  
    }  
  );  
}
```

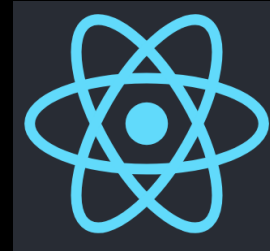
```
unsubscribeTopic = () => {  
  client.unsubscribe(this.state.subscribedTopic);  
  this.setState({ subscribedTopic: " " });  
}  
onChangeMessage = (text) => {  
  this.setState({ message: text });  
}
```

React Native API MQTT



```
sendMessage = () =>{
  var message = new Paho.MQTT.Message(options.id + ':' + this.state.message);
  message.destinationName = this.state.subscribedTopic;
  client.send(message);
}
renderRow = ({ item, index }) => {
  idMessage = item.split(':');
  console.log('>>>ITEM', item);
  return(
    <View
      style={
        styles.componentMessage,
        idMessage[0] == options.id ? styles.myMessageComponent: (idMessage.length == 1 ? styles.introMessage : styles.messageComponent),
      ]
    >
    <Text style={idMessage.length == 1 ? styles.textIntro : styles.textMessage}>
      {item}
    </Text>
    </View>
  )
}
```

React Native API MQTT



```
_keyExtractor = (item, index) => item + index;  
render() {  
  const { status, messageList } = this.state;  
  return (  
    <View style={styles.container}>  
      <Text  
        style={{  
          marginBottom: 50,  
          textAlign: 'center',  
          color: this.state.status === 'connected' ? 'green' : 'black'  
        }} >  
        ClientID: {options.id}  
      </Text>  
      {  
        this.state.status === 'connected' ?  
        <View>  
          <Button  
            type='solid'  
            title='DISCONNECT'  
            onPress={() => {  
              client.disconnect();  
              this.setState({ status: '', subscribedTopic: '' });  
            }}  
            buttonStyle={{ marginBottom: 50, backgroundColor: '#397af8' }}  
            icon={{ name: 'lan-disconnect', type: 'material-community', color: 'white' }}  
          />  
        
```


React Native API MQTT



```
<View style={{ marginBottom: 30, alignItems: 'center' }}>
  <Input
    label='TOPIC'
    placeholder=''
    value={this.state.topic}
    onChangeText={this.onChangeTopic}
    disabled={this.state.subscribedTopic}
  />
  {
    this.state.subscribedTopic ?
      <Button
        type='solid'
        title='UNSUBSCRIBE'
        onPress={this.unSubscribeTopic}
        buttonStyle={{ backgroundColor: '#397af8' }}
        icon={{ name: 'link-variant-off', type: 'material-community', color: 'white' }}
      />
      <Button
        type='solid'
        title='SUBSCRIBE'
        onPress={this.subscribeTopic}
        buttonStyle={{ backgroundColor: '#397af8' }}
        icon={{ name: 'link-variant', type: 'material-community', color: 'white' }}
        disabled={!this.state.topic || this.state.topic.match(/ /) ? true : false}
      />
    }
  </View>
```

Código completo no arquivo .rar enviado (projeto ExemploMQTT01, vamos executar de forma combinada).

React Native API Expo Location



- Biblioteca que fornece acesso à leitura de informações de geolocalização, sondagem da localização atual e assinatura de eventos de atualização de localização do dispositivo;
- Informações sobre como usar em:
 - <https://docs.expo.dev/versions/latest/sdk/location/>

React Native API Expo Location



```
import React, { useState, useEffect } from 'react';
import { Text, View, StyleSheet } from 'react-native';
import * as Location from 'expo-location';

export default function App() {
  const [location, setLocation] = useState(null);
  const [errorMsg, setErrorMsg] = useState(null);

  useEffect(() => {
    (async () => {
      let { status } = await Location.requestForegroundPermissionsAsync();
      if (status !== 'granted') {
        setErrorMsg('Permission to access location was denied');
        return;
      }

      let location = await Location.getCurrentPositionAsync({});
      setLocation(location);
    })();
  }, []);
```

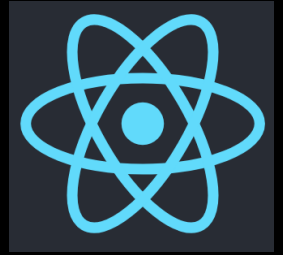
React Native API Expo Location



```
let text = 'Waiting..';
if (errorMsg) {
  text = errorMsg;
} else if (location) {
  text = JSON.stringify(location);
}
return (
  <View style={styles.container}>
    <Text style={styles.paragraph}>{text}</Text>
  </View>
);
}
const styles = StyleSheet.create({
  container: {
    flex: 1,
    alignItems: 'center',
    justifyContent: 'center',
    padding: 20,
  },
  paragraph: {
    fontSize: 18,
    textAlign: 'center',
  },
});
```

Executar o projeto ExemploLocation01.

React Native



FIM