

React Native

Projeto de Interfaces de Dispositivos Móveis

Introdução
Aula 01

Introdução

- **O que é o React?**

- React ou React.js ou RieactJS é uma biblioteca Javascript para criar interfaces de usuário. Lançado em 2013 e mantido pelo Facebook.

- **O que é o React Native?**

- Em 2015, o Facebook anunciou o módulo React Native, que em conjunto com o React, possibilita o desenvolvimento de aplicativos para Android e iOS, utilizando os componentes de ambas as plataformas, sem recorrer ao HTML.

Plataformas de Desenvolvimento

- Existem duas principais formas de começar a desenvolver usando o React Native:
 - Expo CLI
 - React Native CLI

<https://facebook.github.io/react-native/docs/getting-started.html>

Expo CLI

- If you are coming from a web background, the easiest way to get started with React Native is with Expo tools because they allow you to start a project without installing and configuring Xcode or Android Studio.
- Expo CLI sets up a development environment on your local machine and you can be writing a React Native app within minutes.
- For instant development, you can use Snack to try React Native out directly in your web browser.

React Native CLI

- If you are familiar with native development, you will likely want to use React Native CLI. It requires Xcode or Android Studio to get started.
- If you already have one of these tools installed, you should be able to get up and running within a few minutes.
- If they are not installed, you should expect to spend about an hour installing and configuring them.

Qual iremos usar?

- Iremos usar o **Expo CLI**, pela facilidade de configuração. No entanto, sintam-se livres para escolher outra plataforma.
- Este site apresenta uma ótima análise entre as duas plataformas:
 - <https://levelup.gitconnected.com/expo-vs-react-native-cli-a-guide-to-bootstrapping-new-react-native-apps-6f0fcafee58f>

Instalação

- Considerando o UBUNTU
 - Instale a versão 10 ou superior do node.js:
 - `sudo apt-get install curl software-properties-common`
 - `curl -sL https://deb.nodesource.com/setup_12.x | sudo -E bash -`
 - `sudo apt-get install nodejs`
 - `node -v` (opcional, apenas para ver a versão)
 - <https://tecadmin.net/install-latest-nodejs-npm-on-ubuntu/>

Instalação

- Instale o Expo
 - **npm install -g expo-cli** (talvez precise do sudo)

Primeira aplicação

- Cria a pasta do projeto, usando o **expo** (terminal):
 - **expo init <nome do projeto>**
- Entre na pasta do projeto e instala as dependências:
 - **cd <nome do projeto>**
 - **npm install**

Primeira aplicação

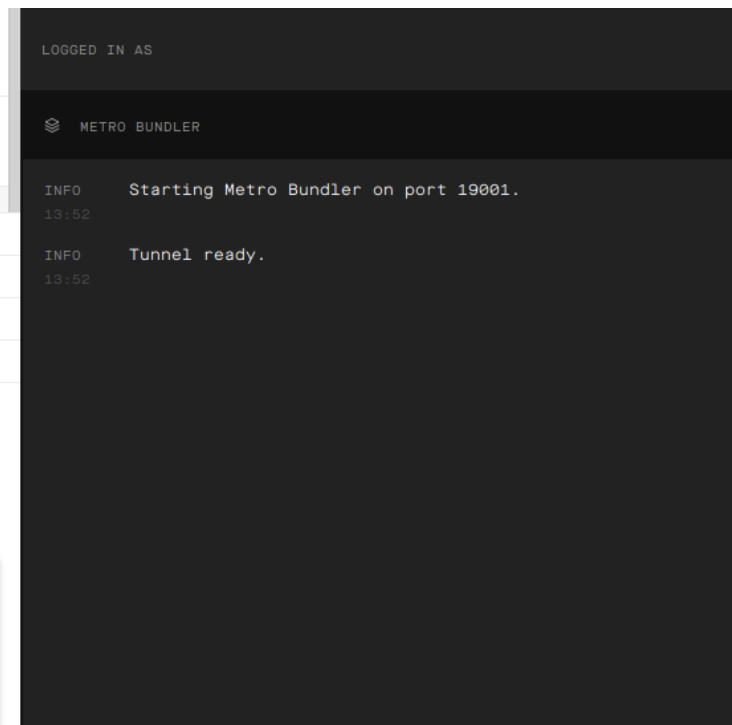
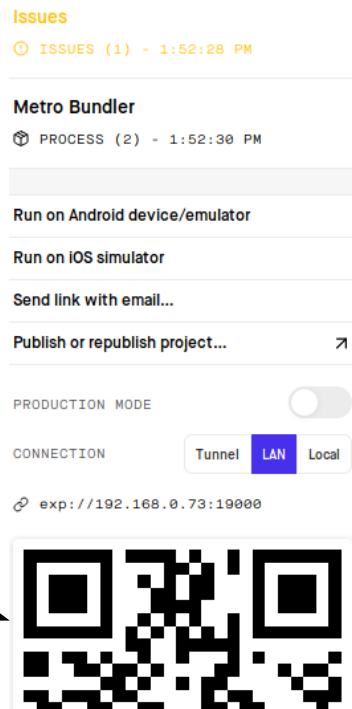
- Inicie o servidor da aplicação:
 - **npm start**
 - ou
 - **expo start**

Primeira aplicação

- O Metro Bundler:

Você pode baixar sua aplicação diretamente no seu dispositivo, via Expo!

Baixe o aplicativo “Expo” na loja do seu smartphone e leia o QR code.



Primeira aplicação

- No terminal, pressione **w**, caso queira rodar no seu navegador.



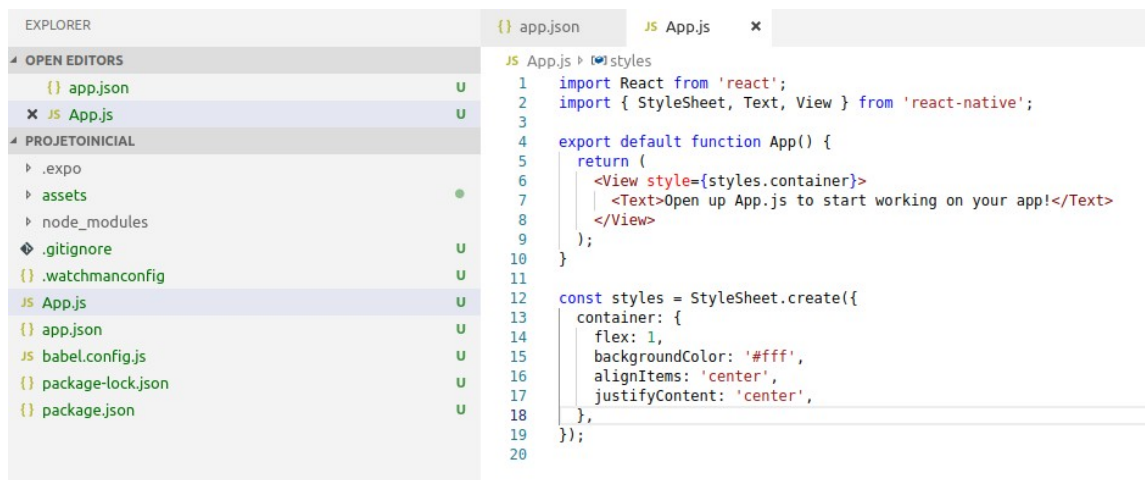
Primeira aplicação

- Abrindo no navegador (porta 19006):

Open up App.js to start working on your app!

Primeira aplicação

- Abra o projeto no **VSCode** (ou no que você preferir), e modifique a linha do **<Text>**.

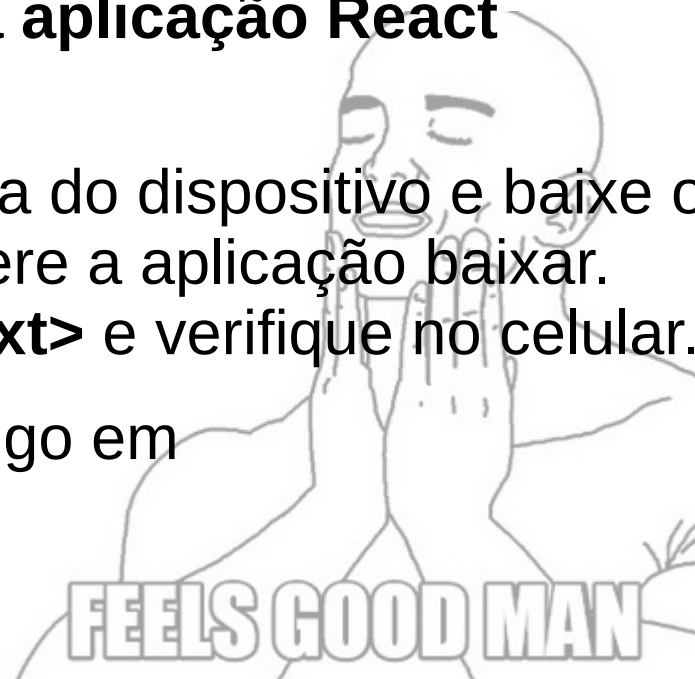


The screenshot shows the VS Code interface. On the left, the Explorer sidebar displays the project structure: 'app.json', 'App.js', 'assets', 'node_modules', '.gitignore', '.watchmanconfig', 'App.js', 'app.json', 'babel.config.js', 'package-lock.json', and 'package.json'. The 'App.js' file is selected. The main editor area shows the content of 'App.js', which includes imports for React, StyleSheet, Text, and View from 'react-native'. The App function returns a View containing a Text element. The Text element's content is 'Open up App.js to start working on your app!'. The styles object is defined with a container style that has a flex of 1, a background color of '#fff', and text alignment and justification set to 'center'.

```
1 import React from 'react';
2 import { StyleSheet, Text, View } from 'react-native';
3
4 export default function App() {
5   return (
6     <View style={styles.container}>
7       <Text>Open up App.js to start working on your app!</Text>
8     </View>
9   );
10 }
11
12 const styles = StyleSheet.create({
13   container: {
14     flex: 1,
15     backgroundColor: '#fff',
16     alignItems: 'center',
17     justifyContent: 'center',
18   },
19 });
20
```

Parabéns!

- **Você conseguiu executar sua primeira aplicação React Native!**
- Tente executar no seu celular. Vá na loja do dispositivo e baixe o aplicativo Expo. Leia o QR Code e espere a aplicação baixar. Modifique novamente o texto entre **<Text>** e verifique no celular.
- Você também pode executar o seu código em <https://snack.expo.io/>



Analizando...

```
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>  
    </View>  
  );  
}
```

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

Código baseado no ES6 (ES2015)
<https://babeljs.io/docs/en/learn/>

Código JSX: uma sintaxe que une
Javascript e XML.

<Text> é um componente pré-programado
que permite mostrar texto.
<View> é como a <div> ou .

O básico

- React Native usa componentes nativos ao contrário de componentes web para construir aplicações.
- Para entender o básico de React Native, inicialmente devemos entender o conceito de **JSX, Componentes, state e props**.

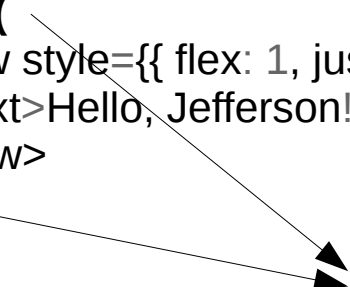
Hello World

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

Uso de componentes que requerem um render o qual deve retornar JSX.



```
export default class HelloWorldApp extends Component {  
  render() {  
    return (  
      <View style={{ flex: 1, justifyContent: "center", alignItems: "center" }}>  
        <Text>Hello, Jefferson!</Text>  
      </View>  
    );  
  }  
}
```



Não esqueça o "(" e o ")"

ES2015 (ES06)

- É um conjunto de melhoramentos sobre o Javascript. O React Native já vem com o suporte a essa versão do Javascript.
- Import, from, class e extends são exemplos de características do ES06.

JSX

- JSX é uma sintaxe que une Javascript com XML. JSX permite que você escreva em linguagem de marcação **dentro** do código da sua aplicação.
- Parece com HTML, mas em vez de usar `<div>` ou `<spam>`, nós usamos componentes React. Neste caso, `<Text>`, para mostrar texto e `<View>`, que funciona como uma `<div>` ou `<spam>`.

Exercício

- Use o `const styles = StyleSheet.create...` dentro do exemplo do HelloWorld. Assim, evitamos o uso de estilos dentro do View: `style={{ flex: 1, justifyContent: "center", alignItems: "center" }}`.

Props


- A maioria dos componentes podem ser customizados quando são criados, com diferentes parâmetros. Estes parâmetros de criação são conhecidos como **props**.
- Por exemplo, um componente básico do React Native é o **Image**. Nele, você pode usar o prop **source** para controlar qual imagem será mostrada.

Props (exemplo)

```
import React, { Component } from 'react';
import { AppRegistry, Image } from 'react-native';

export default class Bananas extends Component {
  render() {
    let pic = {
      uri: 'https://upload.wikimedia.org/wikipedia/commons/d/de/Bananavarieties.jpg'
    };
    return (
      <Image source={pic} style={{width: 193, height: 110}}/>
    );
  }
}
AppRegistry.registerComponent('ProjetoInicial', () => Bananas);
```

PROPS



The diagram consists of two arrows. The first arrow originates from the `source={pic}` prop in the `<Image>` tag and points towards the word **PROPS**. The second arrow originates from the `style={{width: 193, height: 110}}` prop in the same tag and also points towards the word **PROPS**.

Props

- Note que o lado direito de um props começa com “{” e fecha “}” (source={pic}).
- Isso permite o uso de código Javascript **dentro** do JSX (no caso, a variável **pic**)

Reusando componentes

- Os seus componentes também podem personalizar os props. Isto tornar possível que um único componente possa ser usado em diversos locais da aplicação.
- Vamos criar um componente **Heroi** com a props **nome**.
- Depois, vamos reusar o componente Heroi no componente principal **Vingadores**.

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

```
class Heroi extends Component {
  render() {
    return (
      <View style={{alignItems: 'center'}}>
        <Text>Olá {this.props.nome}</Text>
      </View>
    );
  }
}
```

Nosso props, chamado **nome**.

```
export default class Vingadores extends Component {
  render() {
    return (
      <View style={{alignItems: 'center', top: 50}}>
        <Heroi nome='Hulk' />
        <Heroi nome='Capitão América' />
        <Heroi nome='Homen-Aranha' />
      </View>
    );
  }
}
```

Uso do componente Heroi, passando valores para o props **nome**.

```
// skip this line if using Create React Native App
AppRegistry.registerComponent('ProjetoInicial', () => Vingadores );
```

States (estados)

- Existem dois tipos de dados que controlam um componente: **props** e **state**. **props** são inicializados pelo componente-pai e permanecem com o mesmo valor durante toda a vida do componente. Para dados que vão mudar com o tempo, você deve usar o **state**.
- Geralmente, você deve inicializar o **state** no construtor, e então chamar o método **setState** quando você quer mudá-lo.

```

class Blink extends Component {
  constructor(props) {
    super(props);
    this.state = { mostra: true };

    setInterval(
      () => {
        this.setState( estadoAnterior => {
          return { mostra: !estadoAnterior.mostra };
        });
      },
      1000);
  }
  render() {
    if (!this.state.mostra) {
      return null;
    }

    return (
      <Text>{this.props.texto}</Text>
    );
  }
}

```

```

export default class BlinkApp extends
Component {
  render() {
    return (
      <View>
        <Blink texto='1' />
        <Blink texto='2' />
        <Blink texto='3' />
        <Blink texto='4' />
      </View>
    );
  }
}

// skip this line if using Create React Native App
AppRegistry.registerComponent('ProjetoInicial', ()
=> BlinkApp);

```

Analizando

- O construtor de Blink inicializa o **props** da superclasse **Component**.
- Depois, ele inicializa a variável, do tipo **objeto, state**. Essa variável tem apenas uma propriedade (**mostra**), inicializada com o valor **true**.
- Por último, o construtor chama o método **setInterval**, da superclass **Component**.

Analizando

- O método **setInterval** recebe dois parâmetros:
 - Uma função que será executado em um determinado intervalo;
 - Um valor inteiro que representa o intervalo, em milisegundos, o qual irá executar a função do primeiro parâmetro.

Analizando

```
...
setInterval(
  () =>{
    this.setState( estadoAnterior =>{
      return {mostra:!estadoAnterior.mostra}
    });
  },1000);
...
```

- A função recebida pelo `setInterval` (dentro do retângulo) é no formato `() => { ... }`.
- Dentro dessa função, no lugar das `...`, é chamado o método **setState** o qual também recebe como parâmetro uma função (fundo cinza).
- A função interior ao **setState** tem o formato **estadoAnterior=>{ ... }**.
- **estadoAnterior** é o parâmetro de entrada (poderia ser qualquer nome). Ele representa o **último** estado da variável **state**, inicializada no construtor.
- A lógica da função **estadoAnterior=>{ ... }**, é modificar o valor da propriedade **mostra**, negando (!) seu último valor.

Analizando

- Ainda no component **Blink**, temos a implementação do método **render**. A lógica dele é:
 - se a propriedade `mostras` é falsa, retorno **null**, ou seja, não mostre nada na tela.
 - Caso contrário, retorne um JSX com a props **texto**, dentro de um `<Text>`.

Analizando

- A classe, ou componente principal é o BlinkApp. Nele, apenas rescrevemos o método **render**. A lógica dele é simples:
 - apenas retorna um JSX, o qual é uma `<View>` onde dentro da mesma eu chamo o componente Blink, passando como parâmetro o valor do props texto de Blink.

Estilos

- Estilo em React Native são escritos usando Javascript.
- Todos os componentes do core aceitam a props chamada **style**. O style aceita código semelhante ao CSS, no entanto nomes são escritos usando camel-case: backgroundColor no lugar de background-color.
- Você também pode definir os estilos como objetos Javascript. Vamos ao exemplo:

Estilos

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

```
const styles = StyleSheet.create({  
  bigBlue: {  
    color: 'blue',  
    fontWeight: 'bold',  
    fontSize: 30,  
  },  
  red: {  
    color: 'red',  
  },  
});
```

Criação de dois estilos diferentes em uma variável separada.

```
export default class LotsOfStyles extends Component {  
  render() {  
    return (  
      <View>  
        <Text style={styles.red}>just red</Text>  
        <Text style={styles.bigBlue}>just bigBlue</Text>  
        <Text style={[styles.bigBlue, styles.red]}>bigBlue, then red</Text>  
        <Text style={[styles.red, styles.bigBlue]}>red, then bigBlue</Text>  
      </View>  
    );  
  }  
}
```

No caso de arrays, o último estilo de precedência.

```
AppRegistry.registerComponent('AwesomeProject', () => LotsOfStyles);
```

Tamanho (Altura e Largura)

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

export default class FixedDimensionsBasics extends Component {
  render() {
    return (
      <View>
        <View style={{width: 50, height: 50, backgroundColor: 'powderblue'}} />
        <View style={{width: 100, height: 100, backgroundColor: 'skyblue'}} />
        <View style={{width: 150, height: 150, backgroundColor: 'steelblue'}} />
      </View>
    );
  }
}

// skip this line if using Create React Native App
AppRegistry.registerComponent('AwesomeProject', () => FixedDimensionsBasics);
```

Tamanho dinâmico

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

export default class FlexDimensionsBasics extends Component {
  render() {
    return (
      // Try removing the `flex: 1` on the parent View.
      // The parent will not have dimensions, so the children can't expand.
      // What if you add `height: 300` instead of `flex: 1`?
      <View style={{flex: 1}}>
        <View style={{flex: 1, backgroundColor: 'powderblue'}} />
        <View style={{flex: 2, backgroundColor: 'skyblue'}} />
        <View style={{flex: 3, backgroundColor: 'steelblue'}} />
      </View>
    );
  }
}
```

// skip this line if using Create React Native App

```
AppRegistry.registerComponent('AwesomeProject', () => FlexDimensionsBasics);
```

O valor do **flex** preenche a tela de acordo com os componentes irmãos.

Quanto maior o **flex**, maior a parcela de preenchimento.

Flexdirection

- row (default), column, row-reverse ou column-reverse

```
...  
<View style={{flex: 1, flexDirection: 'row'}}>  
  <View style={{width: 50, height: 50, backgroundColor: 'powderblue'}} />  
  <View style={{width: 50, height: 50, backgroundColor: 'skyblue'}} />  
  <View style={{width: 50, height: 50, backgroundColor: 'steelblue'}} />  
</View>  
...
```

Justifycontent

- **flex-start (default), flex-end, center, space-between, space-around.**

```
...  
<View style={{  
  flex: 1,  
  flexDirection: 'column',  
  justifyContent: 'space-between',  
}}>  
  <View style={{width: 50, height: 50, backgroundColor: 'powderblue'}} />  
  <View style={{width: 50, height: 50, backgroundColor: 'skyblue'}} />  
  <View style={{width: 50, height: 50, backgroundColor: 'steelblue'}} />  
</View>  
...
```

Align items

- **stretch (default), flex-start, flex-end, center, baseline**

```
<View style={{  
  flex: 1,  
  flexDirection: 'column',  
  justifyContent: 'center',  
  alignItems: 'center',  
}}>  
  <View style={{width: 50, height: 50, backgroundColor: 'powderblue'}} />  
  <View style={{width: 50, height: 50, backgroundColor: 'skyblue'}} />  
  <View style={{width: 50, height: 50, backgroundColor: 'steelblue'}} />  
</View>
```


Align Self

- Mesmos valores do align items

```
<View style={{  
  flex: 1,  
  flexDirection: 'column',  
  justifyContent: 'center',  
  alignItems: 'center',  
}}>  
  <View style={{width: 50, height: 50, backgroundColor: 'powderblue', alignSelf: 'flex-start'}} />  
  <View style={{width: 50, height: 50, backgroundColor: 'skyblue'}} />  
  <View style={{width: 50, height: 50, backgroundColor: 'steelblue'}} />  
</View>
```

Flex Wrap

```
<View style={{
  flex: 1,
  flexDirection: 'column',
  flexWrap: 'wrap'
}}>
  <View style={{width: 50, height: 50, backgroundColor: 'powderblue'}} />
  <View style={{width: 50, height: 50, backgroundColor: 'skyblue'}} />
  <View style={{width: 50, height: 50, backgroundColor: 'steelblue'}} />
  <View style={{width: 50, height: 50, backgroundColor: 'powderblue'}} />
  <View style={{width: 50, height: 50, backgroundColor: 'skyblue'}} />
  //REPEAT...
</View>
```

Absolute

```
<View style={{  
  flex: 1  
}}>  
  <View style={{position: 'absolute', top: 40, left: 40, width: 50, height: 50, backgroundColor: 'powderblue'}} />  
  <View style={{position: 'absolute', top: 50, left: 50, width: 50, height: 50, backgroundColor: 'skyblue'}} />  
  <View style={{position: 'absolute', top: 60, left: 60, width: 50, height: 50, backgroundColor: 'steelblue'}} />  
</View>
```

Yoga playground

- <https://yogalayout.com/playground>