

# Componentes Básicos do React Native

## Projeto – App Health

# O que vamos aprender

## Componentes básicos:

- View
- Image
- Scrollview
- Text
- text Input
- Stylesheet

## Componentes de interface do usuário:

- Button
- Switch

## Documentação

# Objetivos

- O App tem como funcionalidade calcular o **IMC** de uma pessoa, tendo como entrada de dados a sua **altura** e o seu **peso**.
- Neste App será estudado os componentes **View**, **Text**, **Text Input** e **Button**.

**Obs:** Abrir a *documentação* do *React Native* para conhecer melhor cada componente.

# Acessar a Documentação 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.

**Obs:** Ao clicar sobre um componente ele mostra um exemplo de como usar.

# Mão no Código! – Construindo o App Health

1) Inicie o seu projeto executando os seguintes passos:

- No CMD altere a versão do NODE em uso.

```
C:\Users\marti>nvm use 16.20.0
```

```
C:\Users\marti>nvm list
```

- Criar e acessar a pasta de projetosApp;
- Criar o projeto.

```
C:\Users\marti>mkdir projetosAPP
```

```
C:\Users\marti>cd projetosApp
```

```
C:\Users\marti\projetosAPP>npx create-expo-app appHealth
```

2) Após criar o projeto execute os seguintes passos:

- Acessar a pasta do projeto;
- Abrir o projeto no Visual Studio Code;

```
C:\Users\marti\projetosAPP>cd appHealth
```

```
C:\Users\marti\projetosAPP\appHealth>code .
```

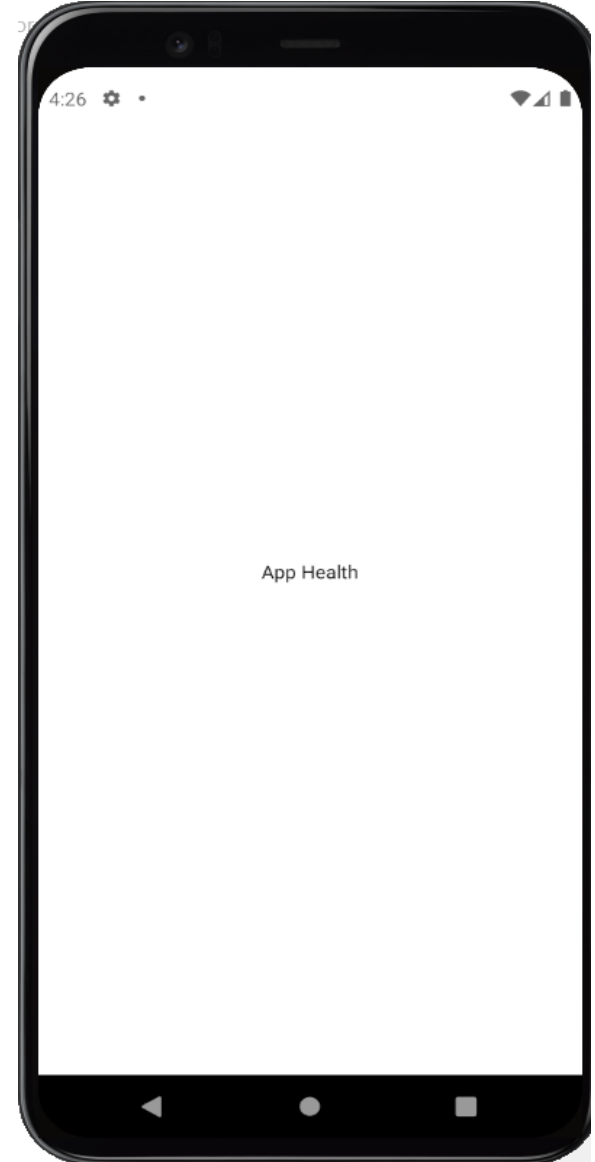
3) No Terminal do Visual Studio execute os seguintes passos: (**CTRL + SHIFT + `**) → Abrir o terminal

- Abrir o projeto em modo de produção pelo **EXPO** usando o aplicativo **Expo Go**

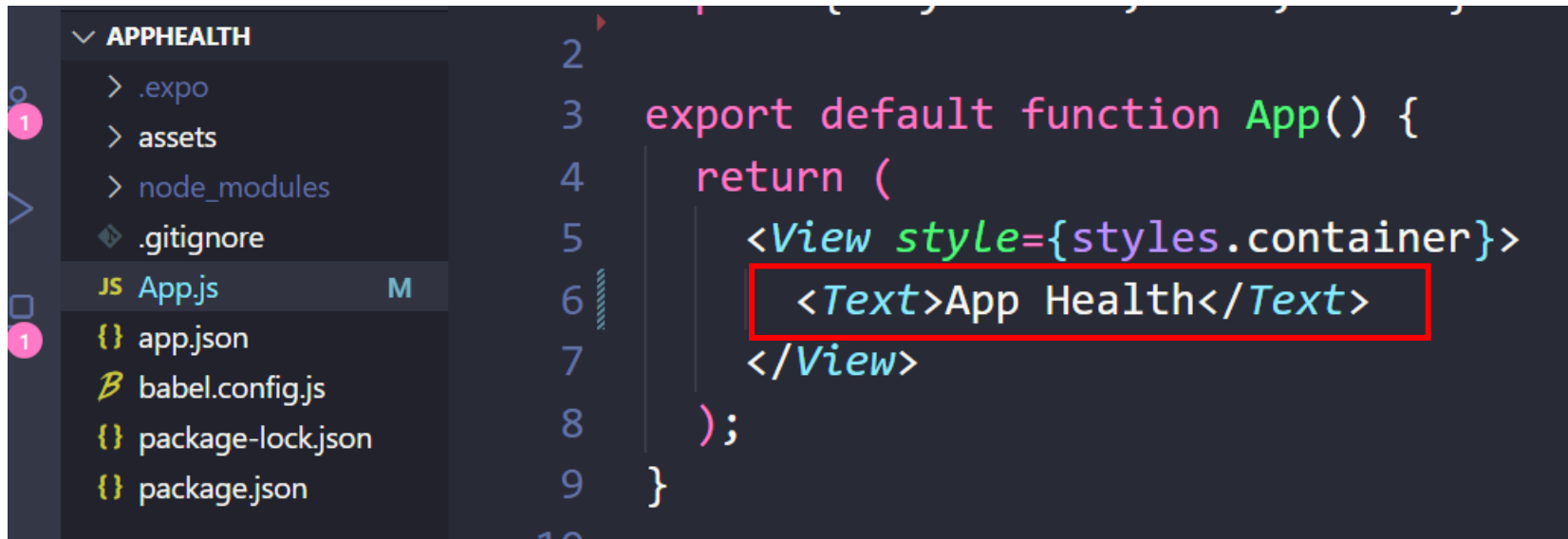
```
C:\Users\marti\projetosAPP\appHealth> npx expo start
```

- Pressione a letra “**a**”

Para abrir o **Emulador do Android Studio**;

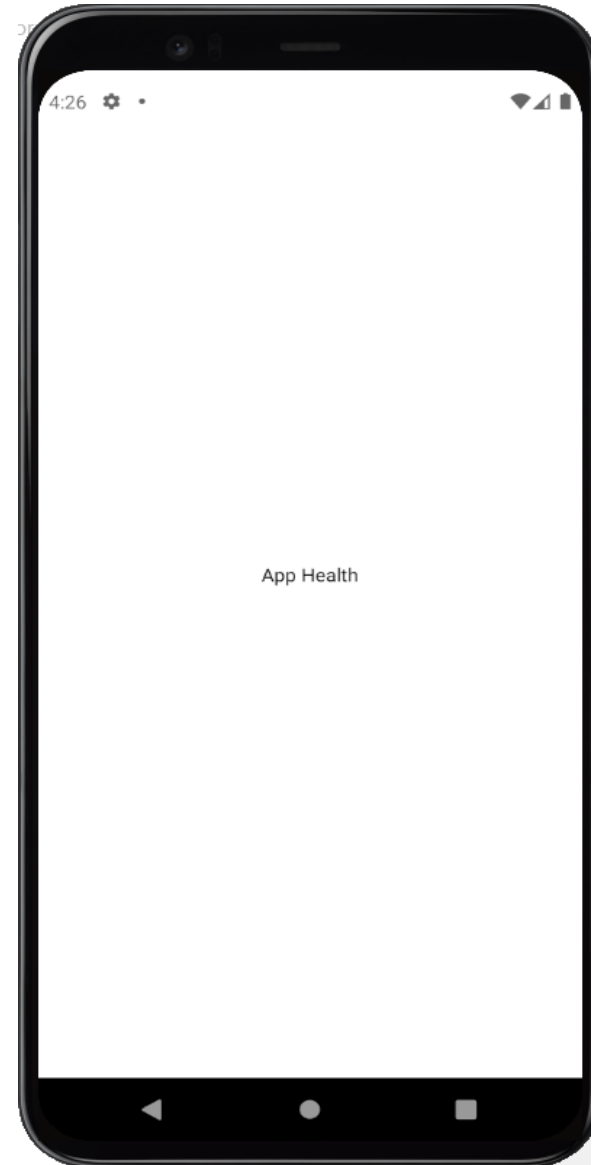


## 4) Vamos começar alterando o texto de renderização na tela.



```
1 2
3  export default function App() {
4    return (
5      <View style={styles.container}>
6        <Text>App Health</Text>
7      </View>
8    );
9  }
```

**Obs:** Tudo que colocamos no return da função App é renderizado na tela.





## 5) Criando uma **Estrutura de Pasta** para Organizar os Objetos.

1. Na pasta do projeto “appHealth”, criar a pasta **src**
2. Dentro do **src**, criar a pasta **components**
3. Dentro de **components** criar a pasta **Title**
4. Dentro de **Title** criar o arquivo **index.js**

**Obs:** Tudo que colocamos no return da função App é renderizado na tela.

## 6) No arquivo **index.js** faça:

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

export default function Title(){
  return(
    <View>
      <Text>
        App Health
      </Text>
    </View>
  );
}
```

**6.1** Realizando as importações das bibliotecas e componentes

**6.2** Criando a função de Título que será renderizada na tela do App.

**6.3** Agora vamos até o arquivo principal da nossa aplicação o **App.js** importar esse título.

7) No arquivo **App.js** vamos **remover** da View o **Text** do App e **importar** o arquivo **index.js** com o título:

```
import { StyleSheet, View } from 'react-native';
import Title from './src/components/Title';

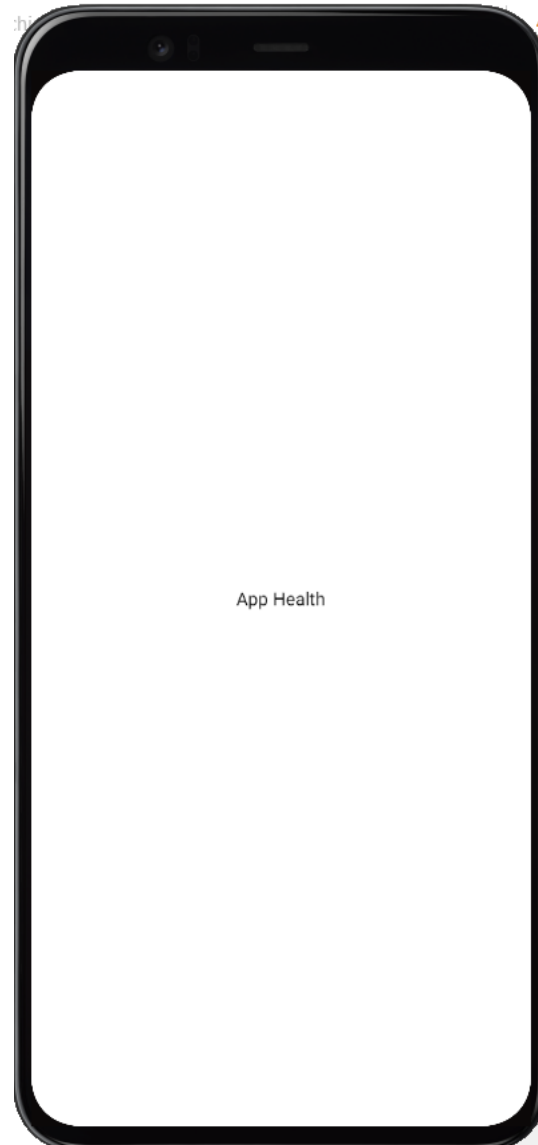
export default function App() {
  return (
    <View style={styles.container}>
      <Title/>
    </View>
  );
}
```

**7.2** Importando o arquivo **index.js** que está dentro da pasta Title. Como o arquivo se chama **index**, não precisa colocar o nome dele.

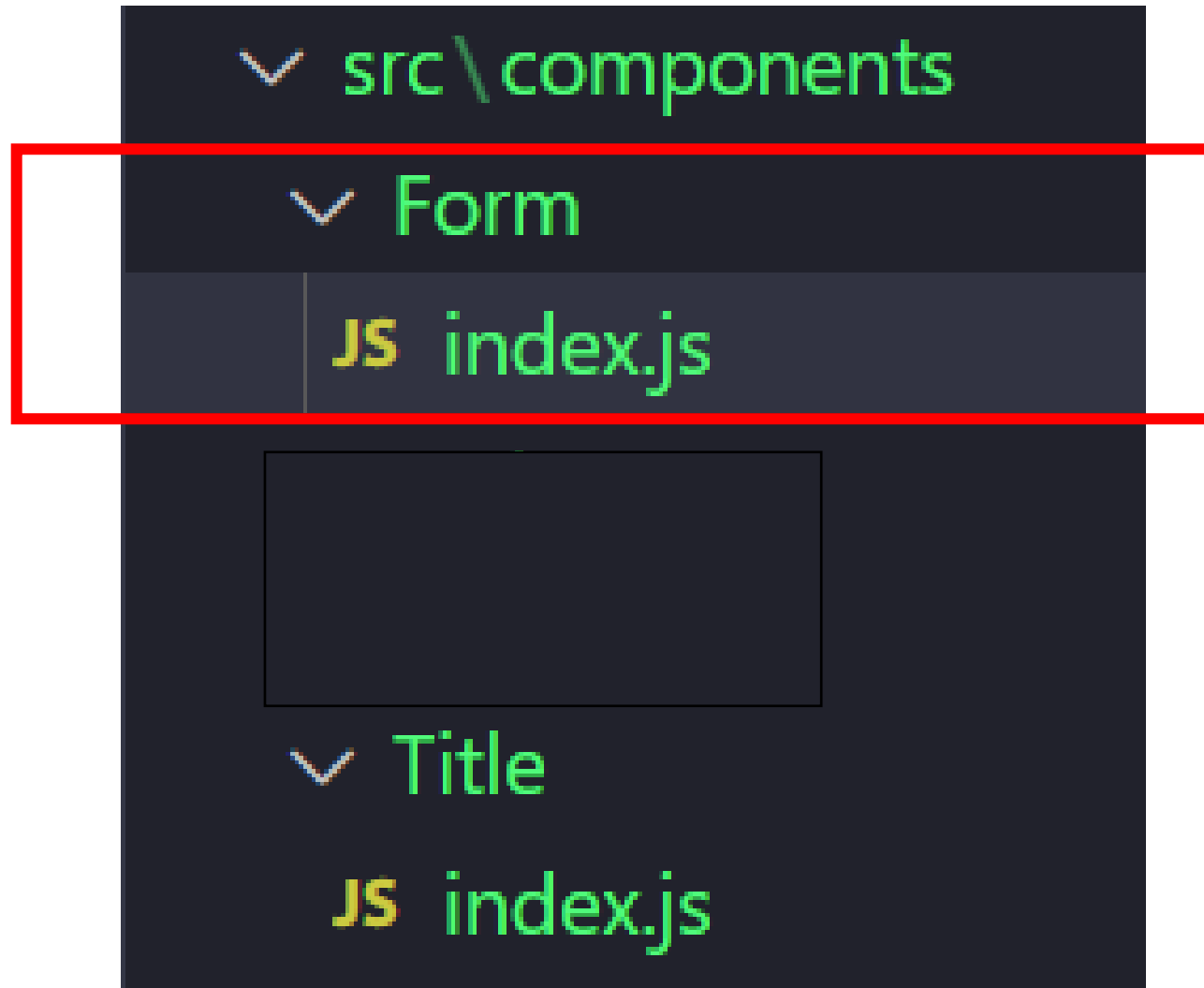
**7.1** Chamando a função **Title**. Tudo que tiver dentro da função será renderizado na tela.

**7.3** Após salvar o título deve aparecer na tela. Podemos fazer uma alteração no título do arquivo **index.js** para ver se será atualizado na tela.

## 8) Tela do Aplicativo até o momento:



13) Agora vamos criar o componente **Form**, e o arquivo **index.js**



14) Agora vamos criar o **código** do formulário no **index.js** (**copiar o código da Title, alterar o código e criar o formulário**)

Vamos utilizar os componentes **Text**, **TextInput** e alguns **parâmetros** desses componentes.

src\components

Form

JS index.js

Title

JS index.js

.gitignore

JS App.js

app.json

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

export default function Form(){
  return(
    <View>
      <View>

        <Text>Altura</Text>
        <TextInput
          placeholder="Ex: 1.75"
          keyboardType="numeric"
        />

        <Text>Peso</Text>
        <TextInput
          placeholder="Ex: 70.750"
          keyboardType="numeric"
        />
      </View>
    </View>
  );
}
```

Parâmetro que informa qual o tipo de teclado o React Native deve abrir quando o usuário clicar no campo .

15) Agora vamos **importar** o **Form** dentro do arquivo principal da aplicação **App.js** e chamar a função **<Form/>**



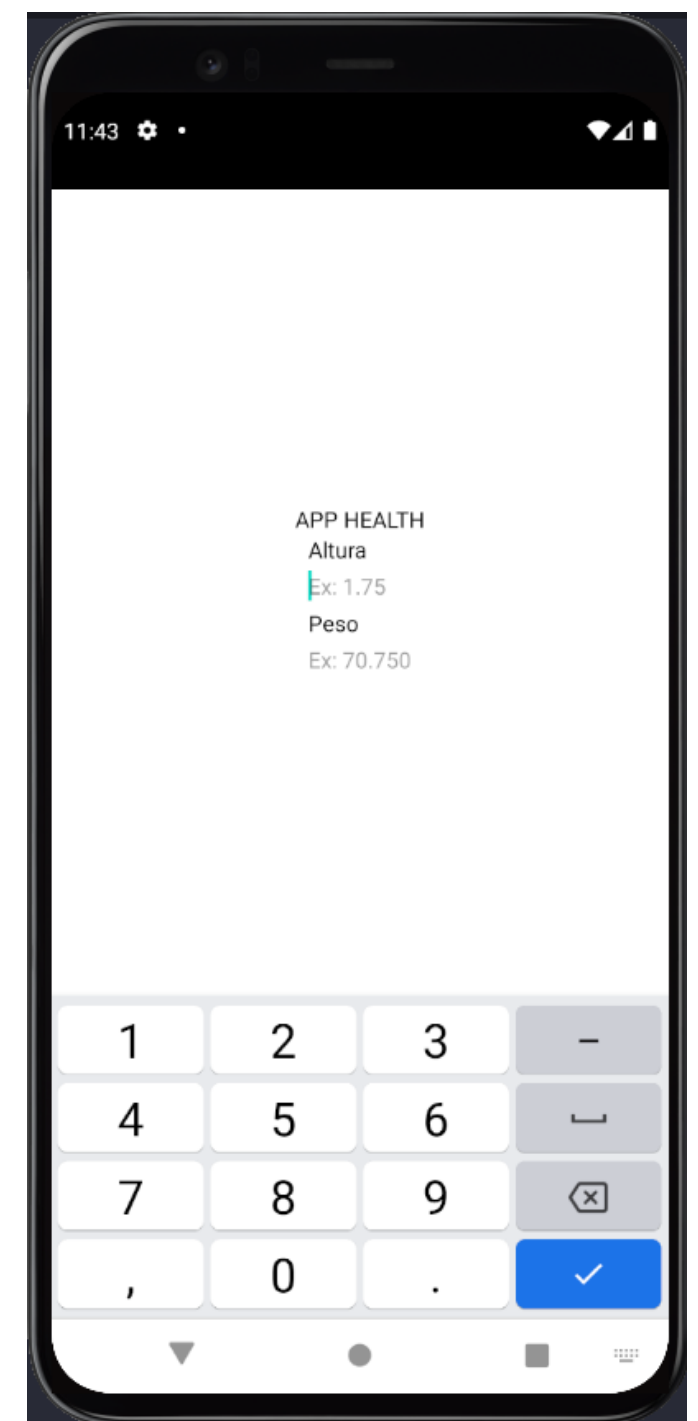
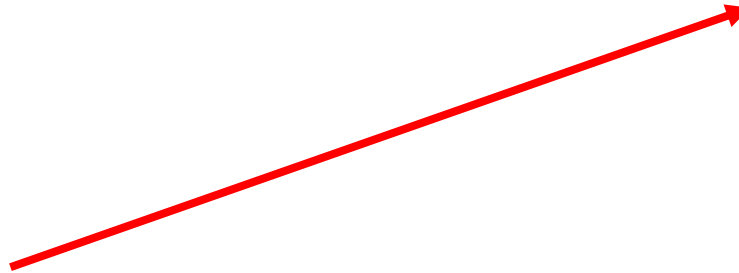
```
import React from 'react';
import { StyleSheet, Text, View } from 'react-native';
import Title from './src/components/Title';
import Form from './src/components/Form';
```

```
export default function App() {
  return (
    <View style={styles.container}>
      <Title/>
      <Form/>
    </View>
  );
}
```



17) Agora vamos **executar** o **App** para verificar se está **apresentando** os **componentes** na tela e abrindo o **teclado numérico** quando clicar no campo **Input**.

O **App** funcionou corretamente como esperado.

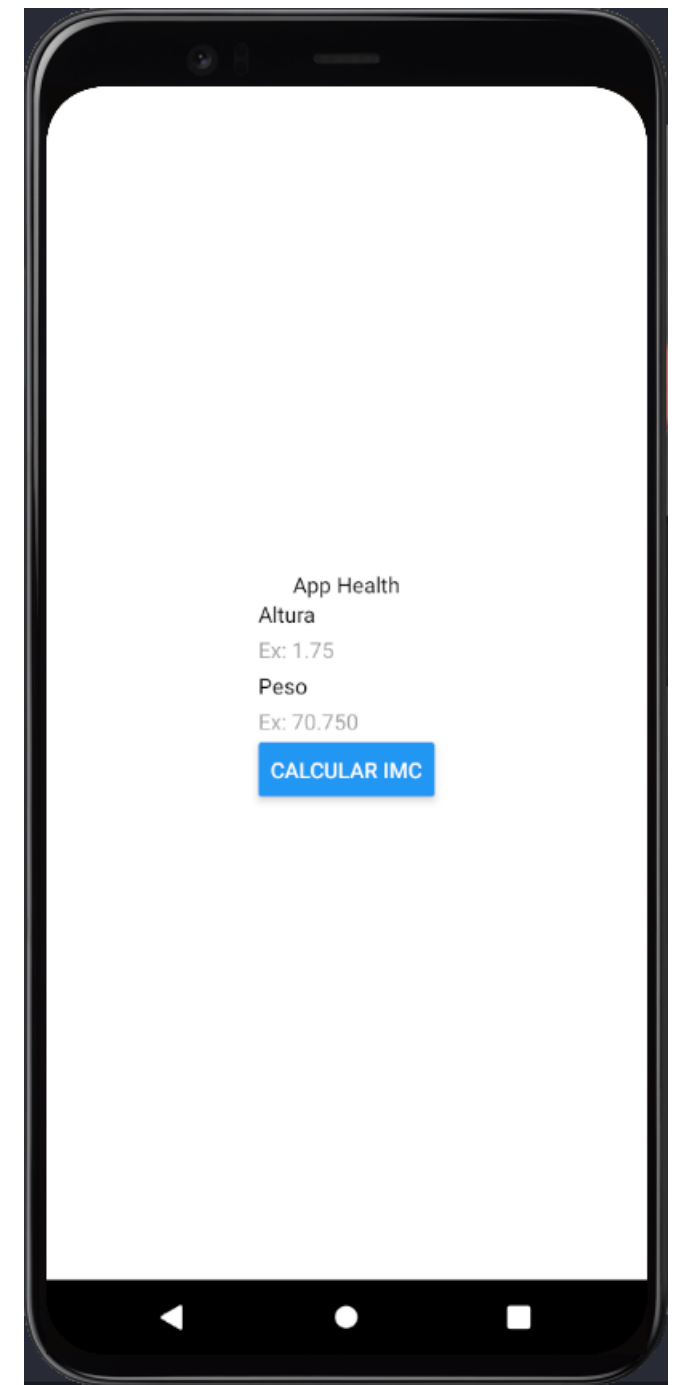


18) Agora vamos no **index.js** do **Form** implementar o componente **Button**. Logo abaixo do Peso e **importar o componente Button**.

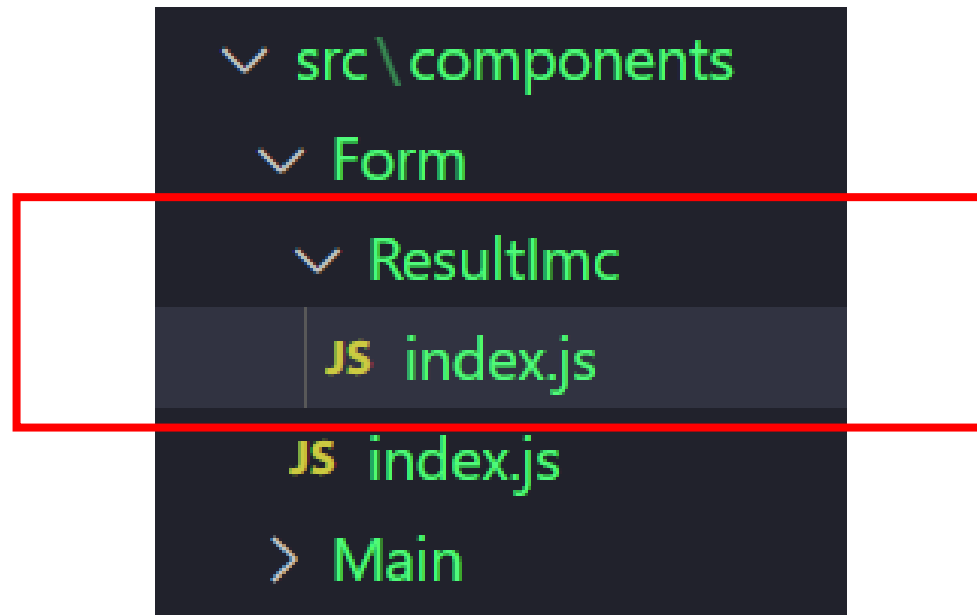
```
import { View, Text, TextInput, Button } from 'react-native';
```

```
<Text>Peso</Text>
  <TextInput
    placeholder="Ex: 70.750"
    keyboardType="numeric"
  />
  <Button
    title="CALCULAR IMC"
  />
</View>
```

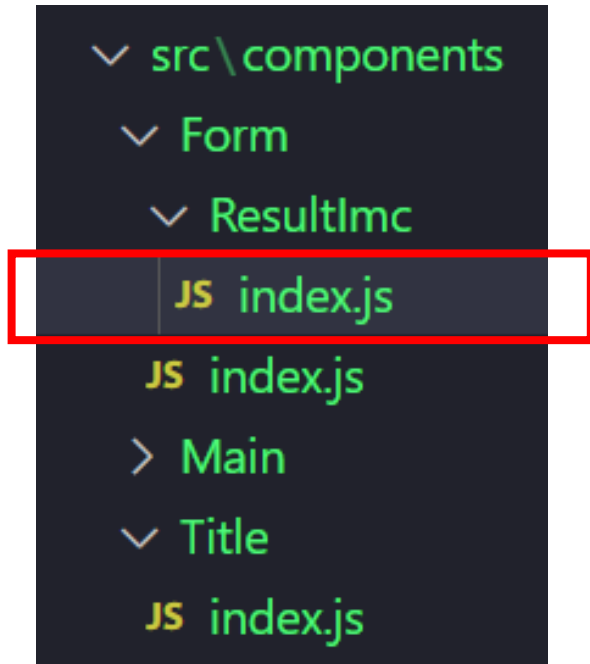
Visualizando o  
botão na tela do  
**App**.



19) Vamos criar dentro de **Form** um componente chamado **ResultImc**, para mostrar o resultado do calculo do **IMC**



20) No arquivo **index.js** do componente **ResultImc**, vamos implementar o código. (copiar o código do title e fazer as alterações)



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

export default function ResultImc(props){
  return(
    <View>
      <Text>{props.messageResultImc}</Text>
      <Text>{props.resultImc}</Text>
    </View>
  );
}
```

Função recebendo uma **props**.

Text renderizando a **props**. Para imprimir a **mensagem**.

Text recebendo a **props**. Para Imprimir o **resultado**.

As **Props** em **React Native**, são as **propriedades** que podemos passar de um componente para outro podendo ser utilizada internamente, seja para exibir ou aplicar alguma lógica própria do componente

Quando o **React** vê um elemento representando um **componente** definido pelo usuário, ele passa atributos **JSX** e componentes filhos para esse componente como um único objeto. Nós chamamos esse objeto de **"props"**.

21) Agora na **View** principal no arquivo **index.js** do componente **Form**, vamos chamar o componente **ResultImc**.

```
src\components
  Form
    ResultImc
      JS index.js
    JS index.js
  Main
  Title
    JS index.js
```

```
<Button
  title="Calcular IMC"
/>
</View>
<ResultImc messageResultImc={messageImc} ResultImc={imc}/>
</View>
);
}
```

Nessa **linha de código** estamos chamando a função **ResultImc** onde ela vai passar duas **props**. (**messageResultImc** e **resultImc**). E os valores **mensagem** e **imc** serão criados na **lógica** do calculo do IMC.

22) Agora vamos ao **index.js** do componente **Form**, implementar a nossa lógica.

1º Vamos importar o **State** para o nosso formulário para **gerenciar** de forma **dinâmica** os valores dos nossos componentes, como **TextInput**, **Button** e **ResultImc**.

E importar o **index.js** do componente **ResultImc**.

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

22) - 2º Agora vamos criar as **constantes** para cada **State** dos nossos componentes que vamos **gerenciar** os seus valores.

```
export default function Form(){  
  const [altura, setAltura]= useState(null)  
  const [peso, setPeso]= useState(null)  
  const [messageImc, setMessageImc]= useState("Preencha o peso e a altura")  
  const [imc, setImc]= useState(null)  
  const [textButton, setTextButton]= useState("CALCULAR")  
  return(  

```

**Pronto.** Os nossos **estados** estão prontos para **gerenciar** os componentes.

22) - 3º Agora vamos **criar** uma outra **função** para pegar o **Peso** e **Altura** e calcular o **IMC**. Vamos criar a **function** logo abaixo dos estados.

```
export default function Form(){  
  
  const [altura, setAltura]= useState(null)  
  const [peso, setPeso]= useState(null)  
  const [messageImc, setMessageImc]= useState("Preencha o peso e a altura")  
  const [imc, setImc]= useState(null)  
  const [textButton, setTextButton]= useState("Calcular")
```

```
function imcCalculator(){  
  return setImc((peso/(altura*altura)).toFixed(2))  
}
```



22) - 4º Agora vamos **setar** os estados após serem calculados. **Por exemplo:** Se os **campos de altura e peso** estiverem com **valor**, retorna o peso e altura, **senão** temos que **avisar** o usuário que é necessário **preencher** para depois calcular.

Vamos **criar** mais uma **função** para **validar o IMC** (verificar se os campos de altura e peso não estão vazios), logo abaixo da função **calcular imc**

```
function validarImc(){
    if (peso != null && altura != null){
        imcCalculator()
        setAltura(null)
        setPeso(null)
        setMessageImc("Seu imc é igual:")
        setTextButton("Calcular novamente")
        return
    }
    setImc(null)
    setTextButton("Calcular")
    setMessageImc("Preencha o peso e a altura")
}
```

A função, começa verificando se o **peso** e a **altura** **não** estão vazios.

Caso seja **verdadeiro**, chamar a função para **calcular o IMC**, setar os campos **Altura** e **Peso** como **nulo** caso queira calcular novamente, **mudar** o **estado** da **mensagem** mostrando o **IMC do usuário** e mudar também o **estado do botão** para **calcular novamente**.

Caso seja **falso**, setar como **nulo** o IMC, voltar estado do botão para **“Calcular”** e voltar o estado da mensagem para **“Preencher o peso e altura”**.

23) Agora fazemos tudo isso funcionar. Vamos ao **TextInput** da **altura** e do **peso**, dentro do **Form**. Informar que queremos **setar** esses **estados**. Alterando os valores dos campos.

```
<Text>Altura</Text>
<TextInput
  onChangeText={setAltura}
  value={altura}
  placeholder="Ex: 1.75"
  keyboardType="numeric"
/>

<Text>Peso</Text>
<TextInput
  onChangeText={setPeso}
  value={peso}
  placeholder="Ex: 70.750"
  keyboardType="numeric"
/>
```

**onChangeText**, vamos utilizar para **alterar** o **estado** do campo **altura** e **peso** assim que digitarmos .

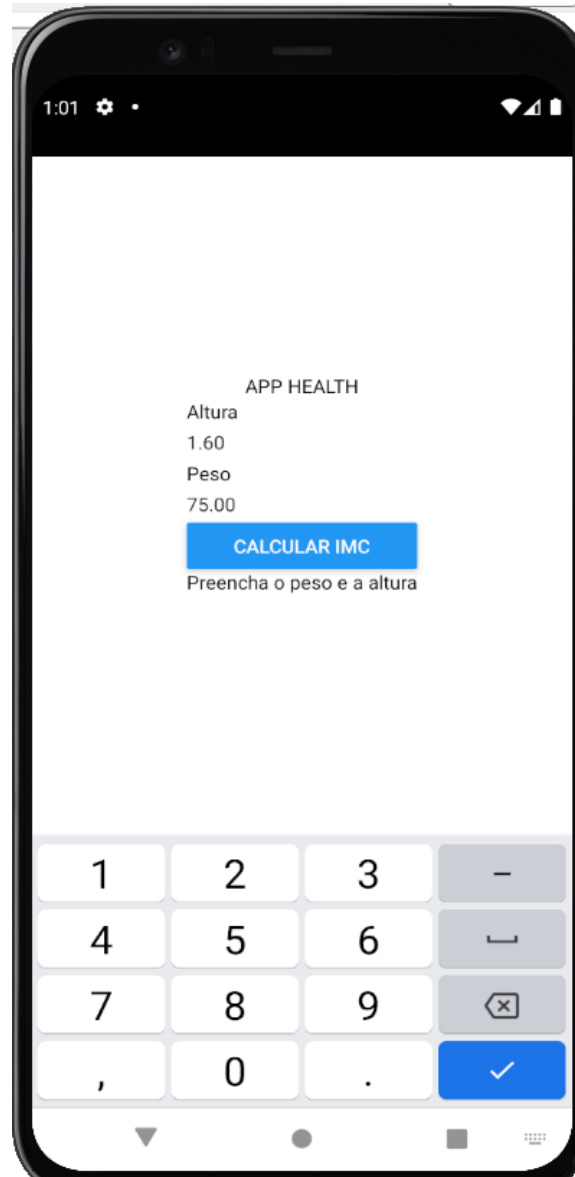
24) Agora para concluir e testa a lógica do nosso aplicativo, vamos até o **botão** no evento de **pressionar o botão**, chamar a **função** para **validar o calculo do IMC** assim que o **botão** for pressionado.

```
<Button  
    onPress={() => validarImc()}  
    title="Calcular IMC"  
/>  
</View>
```

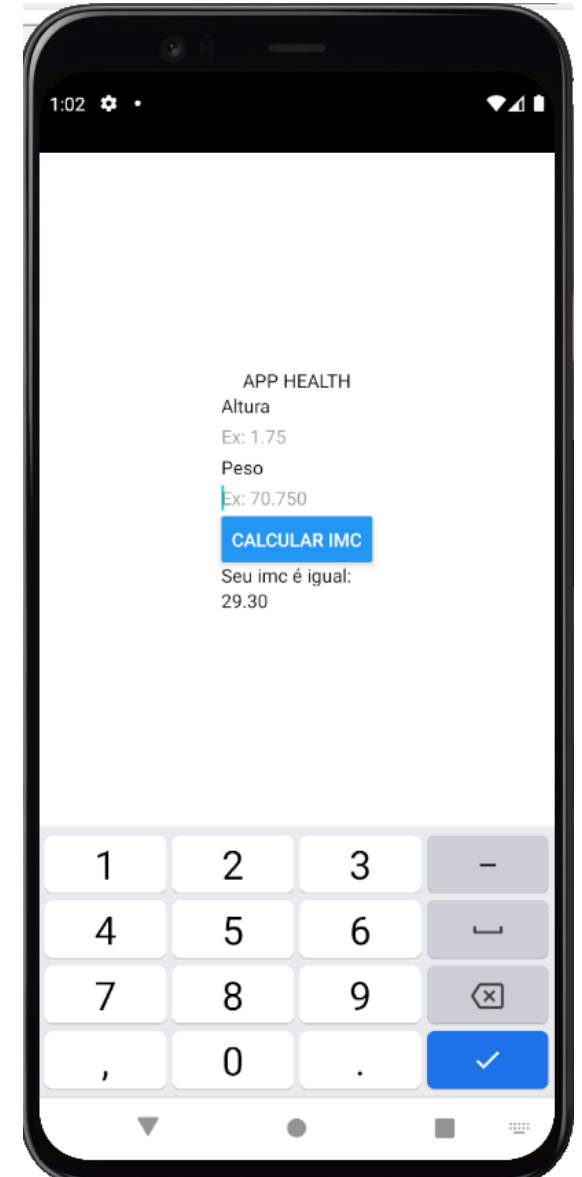
**onPress**, vai chamar de validação do imc quando o botão for pressionado.

25) Agora já podemos testar a lógica do nosso aplicativo e acompanhar o resultado.

Preenchendo  
os dados

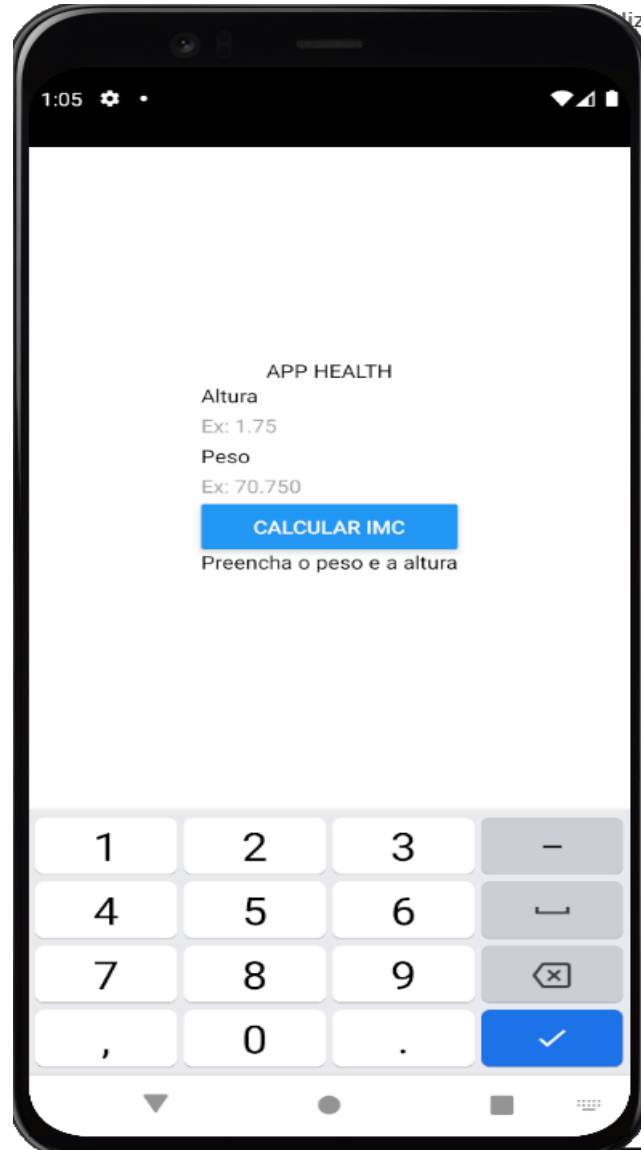


Resultado

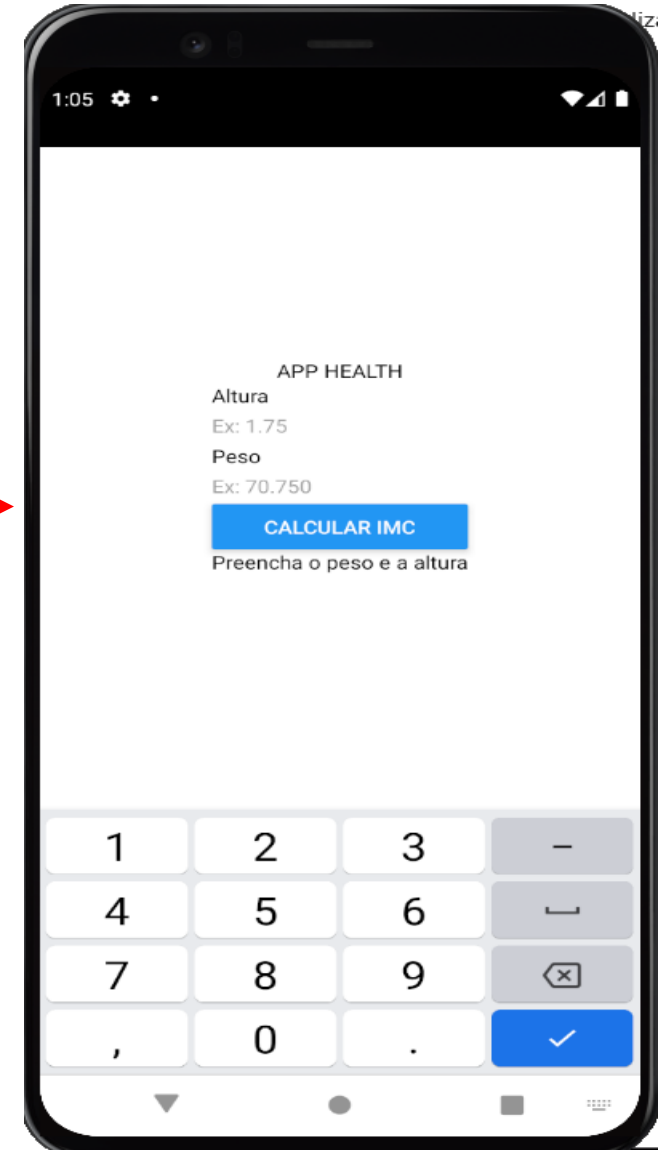


## 26) Fazendo o teste se tentarmos calcular o IMC com os campos vazios.

**Campos vazios**



**Resultado**



27) Agora precisamos tornar **dinâmico** o texto do nosso **botão**. Alterando entre **CALCULAR** e **CALCULAR NOVAMENTE**.

No **Form**, no componente **Button** vamos alterar o texto **estático** do **Title** pelo **estado** do Button.

```
<Button
```

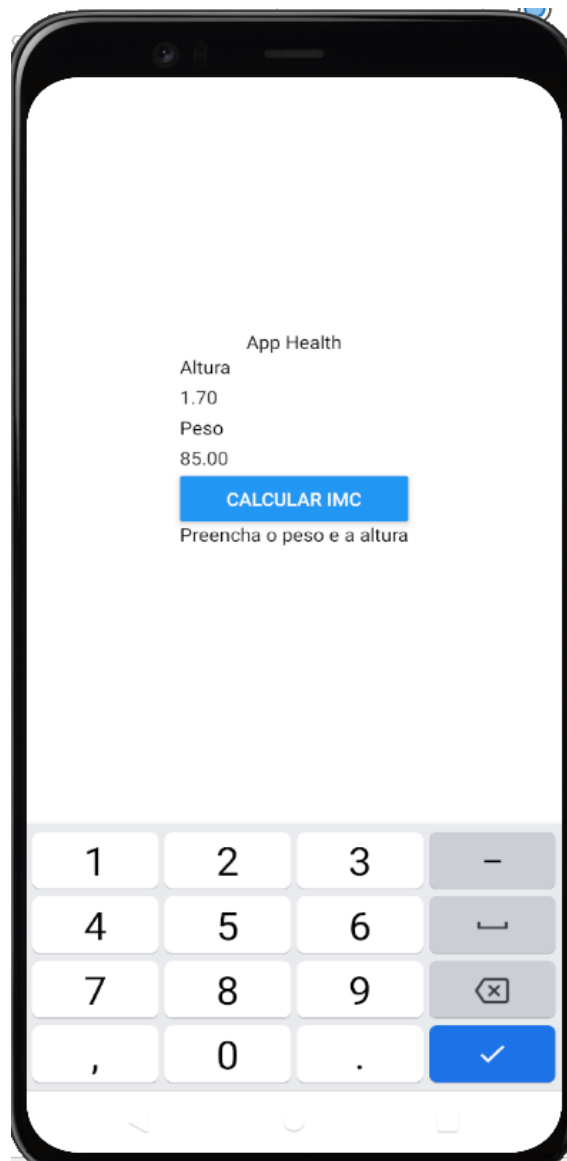
```
  onPress={() => validarImc() }
```

```
  title={textButton}
```

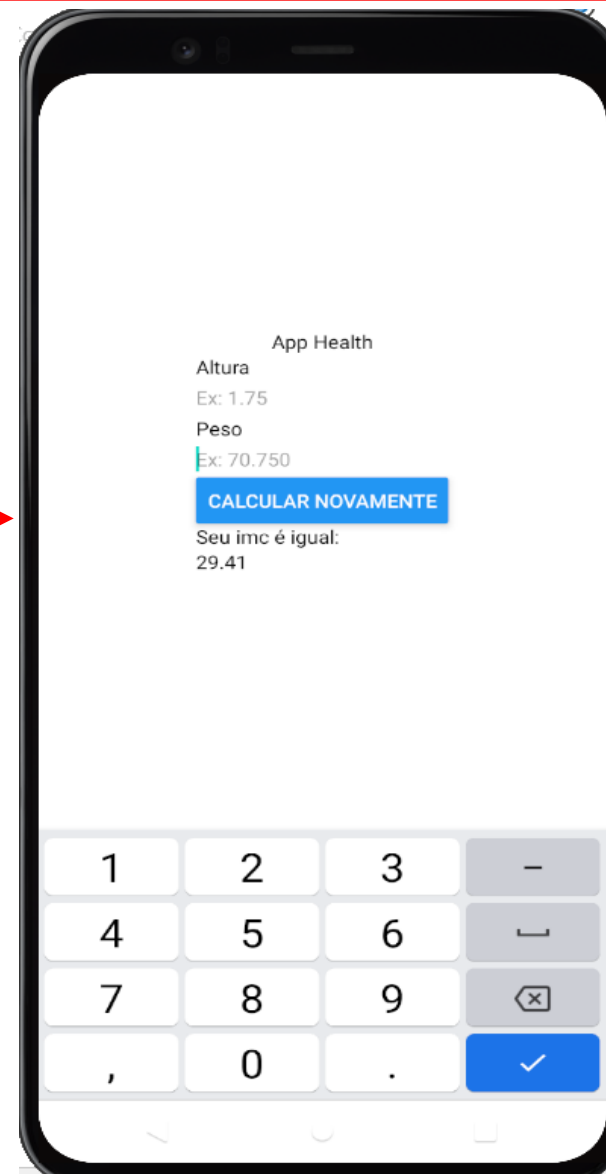
```
/>
```

### 30) Testando o estado dinâmico do texto do botão.

Texto do  
botão  
**antes**  
de calcular



Texto do  
botão  
**após**  
calcular



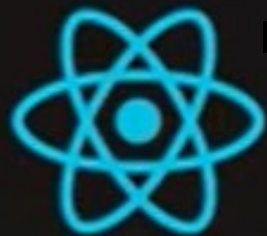
# CONCLUSÃO – PARTE 1

Chegamos ao final da primeira parte do nosso aplicativo, onde trabalhamos com os principais componentes do React Native (**View, Text, Title, InputText e Button**), States e Function.

Onde implementamos a **lógica** da nossa aplicação para **calcular o IMC de uma pessoa**, usando como entrada de dados o seu **peso** e a sua **altura**.

A próxima parte da nossa **aplicação** é trabalhar a **formatação** do nosso **aplicativo**. Vamos entender como trabalhar com estilos melhorando a experiência do usuário.





# React Native

**O que vamos ver na próxima aula:**

- Aprofundamento no **StyleSheet** com React Native
- Criar o **estilo do nosso app** pensando em uma **boa experiência do usuário**