



Calculadora básica

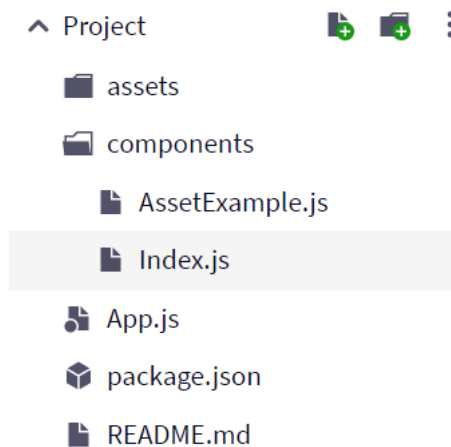
Ewerton J. Silva

ewerton.silva41@etec.sp.gov.br

Criando componente principal

```
1  import React, { useState } from 'react';
2  import { Text, View, StyleSheet, TextInput, TouchableOpacity } from 'react-native';
3
4  export default function Index() {
5    return (
6      <View style={styles.container}>
7        <Text style={styles.paragraph}>
8          Exemplo 5
9        </Text>
10     </View>
11   );
12 }
```

```
14  const styles = StyleSheet.create({
15    container: {
16      flex: 1,
17      justifyContent: 'center',
18      backgroundColor: '#FF80AB',
19      padding: 8,
20    },
21    paragraph: {
22      margin: 6,
23      fontSize: 18,
24      fontWeight: 'bold',
25      textAlign: 'center',
26      color: '#C51162',
27    },
28  });
```



Em um novo projeto dentro da pasta “components” crie um novo arquivo com o nome “Index.js” e insira o código apresentado ao lado.

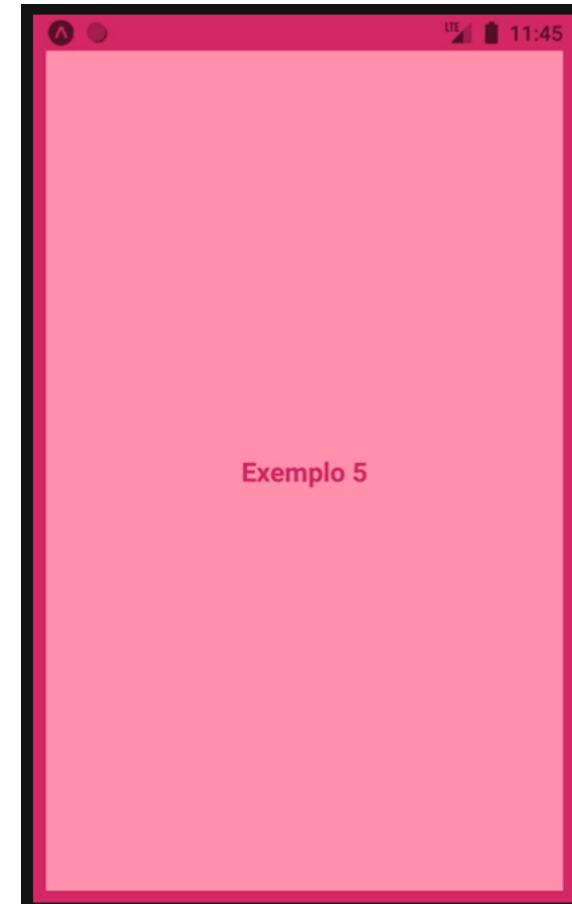
Este componente vai retornar uma View com um Text dentro.

App.js

Em “App.js”, vamos importar o componente “Index.js” e utilizá-lo dentro do “App.js”.

Deixe o código de “App.js” conforme o exemplo ao lado, aqui iremos retornar uma “View” com o componente criado anteriormente.

```
1  import * as React from 'react';
2  import { View, StyleSheet } from 'react-native';
3  import Constants from 'expo-constants';
4
5  import Index from './components/Index';
6
7  export default function App() {
8    return (
9      <View style={styles.container}>
10        <Index />
11      </View>
12    );
13  }
14
15  const styles = StyleSheet.create({
16    container: {
17      flex: 1,
18      justifyContent: 'center',
19      paddingTop: Constants.statusBarHeight,
20      backgroundColor: '#C51162',
21      padding: 8,
22    },
23  });
```



Volte ao “Index.js” e insira os componentes apontados ao lado.

Neste exemplo também estamos utilizando uma forma diferente para definição de estilização, onde passamos para a propriedade “style” um objeto de estilo e também parâmetros de estilo dentro de uma array. Com esta técnica é possível combinar diferentes objetos de estilo, ou passar parâmetros específicos de acordo com a necessidade de formatação.

```
<View style={styles.container}>
  <Text style={styles.paragraph}> Exemplo 5 </Text>

  <Text style={styles.txtSaida}> Calculadora básica </Text>

  <Text style={styles.textLabel}> 1º número </Text>
  <TextInput style={styles.txtEntrada} />

  <Text style={styles.txtSaida}> + </Text>

  <Text style={styles.textLabel}> 2º número </Text>
  <TextInput style={styles.txtEntrada} />

  <Text style={[styles.txtSaida, {margin: 0}]}> = </Text>

  <Text style={styles.textLabel}> Total </Text>
  <TextInput style={styles.txtEntrada} />

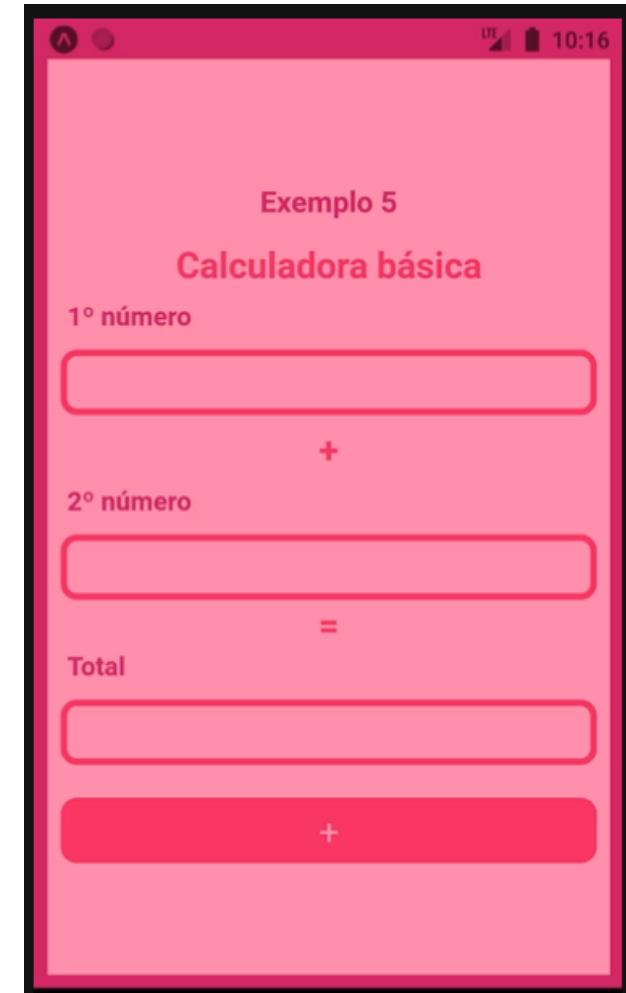
  <TouchableOpacity style={ styles.button}>
    <Text style={styles.textButton}> + </Text>
  </TouchableOpacity>
</View>
```



Adicione os estilos apresentados abaixo para que nosso exemplo fique como apresentado na imagem.

```
46 },
47 txtSaida: {
48   margin: 6,
49   fontSize: 22,
50   fontWeight: 'bold',
51   textAlign: 'center',
52   color: '#E91E63',
53 },
54 txtEntrada: {
55   borderWidth: 4,
56   textAlign: 'center',
57   fontSize: 22,
58   borderColor: '#E91E63',
59   height: 40,
60   color: '#E53935',
61   borderRadius: 10,
62   marginTop: 10,
63 },
64 button: {
65   backgroundColor: '#E91E63',
66   height: 40,
67   justifyContent: 'center',
68   borderRadius: 10,
69   marginTop: 20,
70 },
```

```
70 },
71 textButton: {
72   fontSize: 22,
73   color: '#FF80AB',
74   textAlign: 'center',
75 },
76 textLabel: {
77   fontSize: 16,
78   fontWeight: 'bold',
79   color: '#C51162',
80 },
81 });
```



A ideia de funcionamento do programa é a de receber dois número, somá-los e apresentar o resultado. Para isso iremos definir 3 states, um para cada número inserido e outro para o total.

```
4   export default function App() {  
5  
6     const [n1, setN1] = useState(0);  
7     const [n2, setN2] = useState(0);  
8     const [total, setTotal] = useState(0);  
9  
10    return (  

```

Com os states definidos o próximo passo é garantir a atualização de seus valores.

Nos “TextInput” de entrada dos dois valores que serão somados, insira o evento que vai promover a atualização dos valores e também passe para a propriedade “value” o valor do “state” correspondente.

Já no “TextInput” que vai apresentar o resultado, iremos passar o valor do “state” e também definiremos que a propriedade “editable” vai receber o valor “false”, caso contrário este objeto poderia ter seu valor alterado manualmente.

```
<Text style={styles.textLabel}> 1º número </Text>
<TextInput
  style={styles.txtEntrada}
  onChangeText={ (entrada) => setN1(entrada) }
  value={n1}
/>
```

```
<Text style={styles.textLabel}> 2º número </Text>
<TextInput
  style={styles.txtEntrada}
  onChangeText={ (entrada) => setN2(entrada) }
  value={n2}
/>
```


```
<Text style={styles.textLabel}> Total </Text>
<TextInput
  style={styles.txtEntrada}
  editable={false}
  value={total}
/>
```

Vamos criar uma função que vai realizar a soma dos valores inseridos nos dois primeiros “TextInput”, insira esta função entre os “states” e o “return” da função “default” do componente.

```
4   export default function Index() {  
5  
6     const [n1, setN1] = useState(0);  
7     const [n2, setN2] = useState(0);  
8     const [total, setTotal] = useState(0);  
9  
10    {  
11      function Soma() {  
12        setTotal(n1 + n2);  
13      }  
14    }  
15    return (  
16
```


A função “Soma” será chamada pelo evento “onPress” do “TouchableOpacity” que representa o botão “+”.

```
<TouchableOpacity style={ styles.button} onPress={() => Soma()}>  
  <Text style={styles.textButton}> + </Text>  
</TouchableOpacity>
```



Teste o programa e verifique que a soma não foi realizada e sim a ação de concatenar os valores inseridos no primeiro local com os valores do segundo local de entrada de dados, o que ocorreu aqui foi a concatenação de valores.

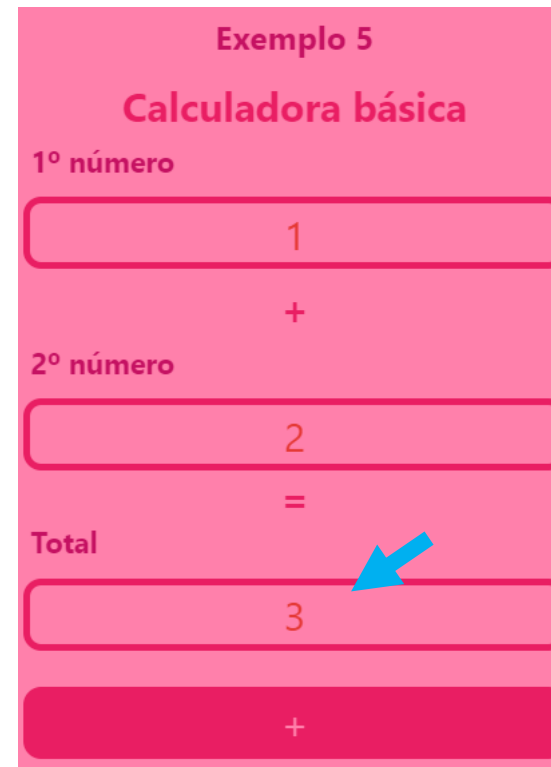
The screenshot shows a mobile application titled "Exemplo 5" and "Calculadora básica". It features two input fields for numbers. The first field, labeled "1º número", contains the digit "1". The second field, labeled "2º número", contains the digit "2". Between these fields is a plus sign "+" and an equals sign "=" indicating an addition operation. Below the second field, a label "Total" is positioned above a third input field that displays "12". A blue arrow points to the "12" in this field, highlighting the result of concatenating the two inputs. At the bottom of the screen is a large red button with a white plus sign "+". The top status bar of the phone shows the time as 10:52.

Sempre teste o resultado de seu aplicativo em todas as plataformas de destino

A solução para nosso problema seria realizar a conversão dos valores de “String” valores de entrada para “Int” valor numérico.

```
function Soma() {  
    setTotal(parseInt(n1) + parseInt(n2));  
}
```

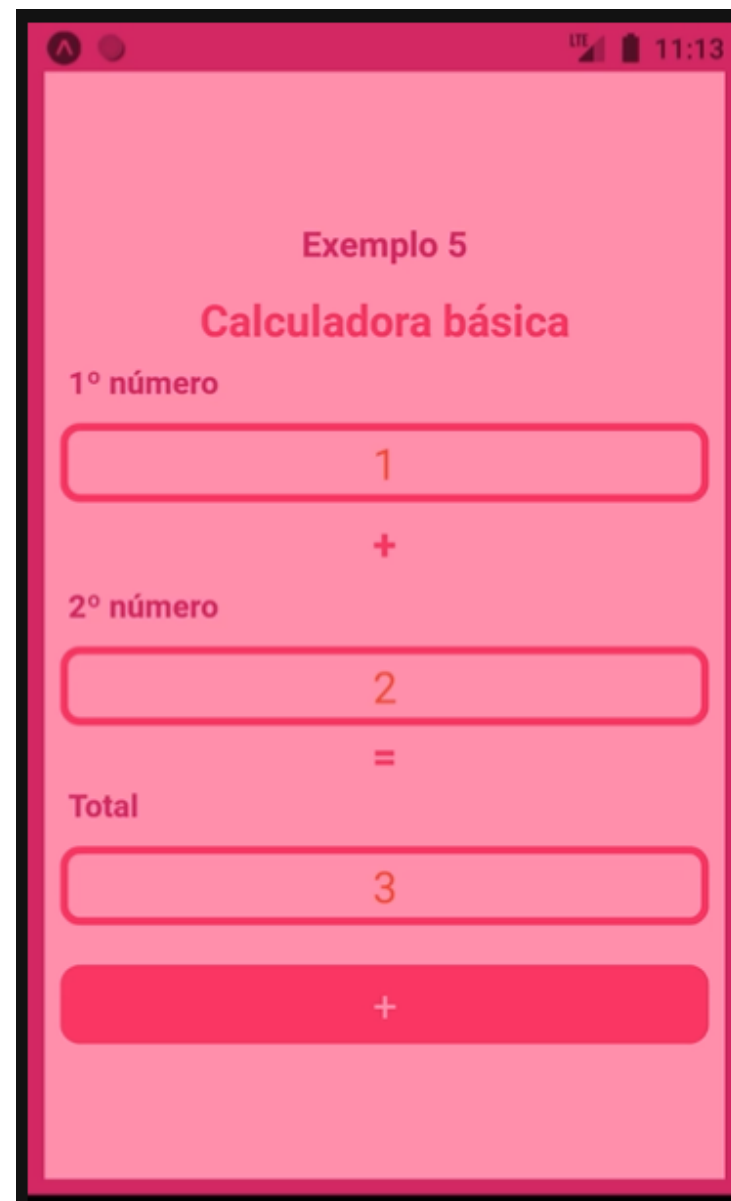
Entretanto a solução apresentada acima só funciona na versão WEB.



Deixe a função “Soma” conforme o exemplo abaixo, onde definimos uma constante para armazenar o valor da soma já convertido em formato numérico, dessa forma temos a variável conta com seu valor sendo do tipo numérico, na sequência passamos esse valor para o “state” total e como ele é de saída de dados, converteremos este para “String” novamente.

```
function Soma() {  
  const conta = parseInt(n1) + parseInt(n2);  
  setTotal(conta.toString());  
}
```

Saiba mais sobre conversão de String em:

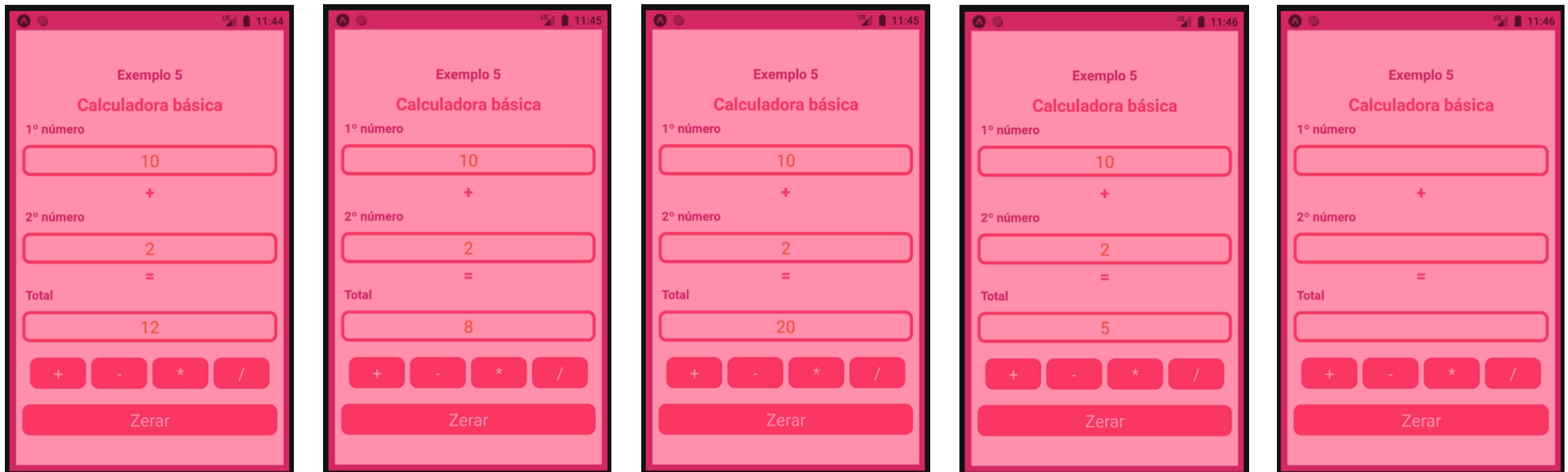


Sugestão de leitura

JavaScript: Convertendo String para número
if...else

Atividade

Crie botões para as quatro operações (+, -, *, /) e um para limpar os dados apresentados em tela. (Obs: leia o texto [JavaScript: Convertendo String para número](#) antes de iniciar a atividade!)



Referências

- <https://www.alura.com.br/artigos/convertendo-string-para-numero-em-javascript>
- https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Reference/Global_Objects/parseInt
- https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Reference/Global_Objects