

# FlatList, Array e Object

Ewerton J. Silva

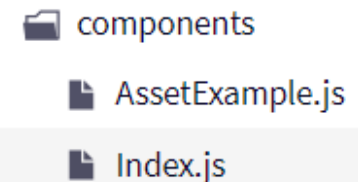
[ewerton.silva41@etec.sp.gov.br](mailto:ewerton.silva41@etec.sp.gov.br)

# 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}> Exemplo 8 </Text>
8     </View>
9   );
10 }
11
12
13
14 const styles = StyleSheet.create({
15   container: {
16     flex: 1,
17     justifyContent: 'center',
18     backgroundColor: '#E1F5FE',
19     padding: 8,
20   },
21   paragraph: {
22     margin: 6,
23     fontSize: 18,
24     fontWeight: 'bold',
25     textAlign: 'center',
26     color: '#03A9F4',
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.

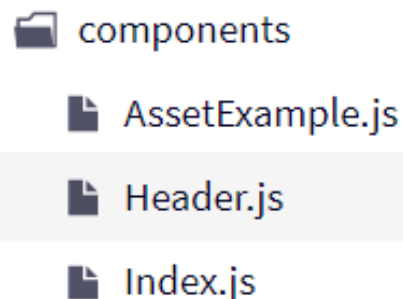
Este componente vai retornar uma View com um Text dentro.



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: '#03A9F4',
21      padding: 8,
22    },
23  });
```

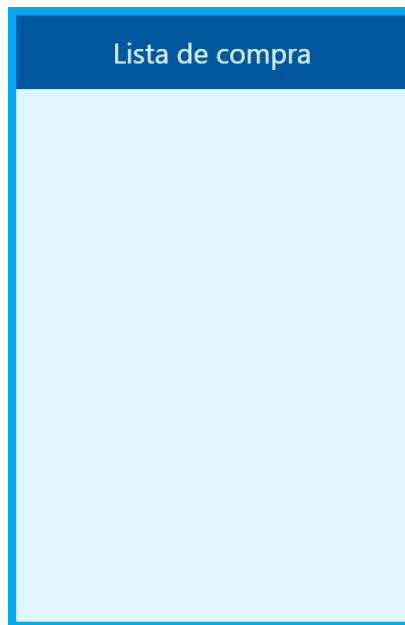
Adicione em “components” um novo componente com o nome “Header.js”, ele representa o cabeçalho de nosso exemplo, deixe seu código conforme o exemplo ao lado.



```
1  import React from 'react';
2  import { Text, View, StyleSheet, } from 'react-native';
3
4  export default function Header() {
5    return (
6      <View style={styles.header}>
7        <Text style={styles.text}> Lista de compra </Text>
8      </View>
9    );
10 }
11
12 const styles = StyleSheet.create({
13   header: {
14     height: 60,
15     padding: 15,
16     backgroundColor: '#01579B',
17   },
18   text: {
19     color: '#E1F5FE',
20     fontSize: 23,
21     textAlign: 'center',
22   },
23 });
```

Volta ao “Index.js”, faço o import do “Header” e insira-o dentro da “View” do componente principal.

Altere o código do “Index.js” para que fique como apresentado ao lado.



```
1  import React, { useState } from 'react';
2  import { View, StyleSheet } from 'react-native';
3
4  import Header from './Header';
5
6  export default function Index() {
7    return (
8      <View style={styles.container}>
9        <Header />
10     </View>
11   );
12 }
13
14
15 const styles = StyleSheet.create({
16   container: {
17     flex: 1,
18     backgroundColor: '#E1F5FE',
19   },
20 });
```

No “Header” podemos passar o valor que será apresentado no cabeçalho através de uma “prop”, assim, temos a possibilidade de chamar o cabeçalho com um título diferente em cada tela, aproveitando o mesmo componente diversas vezes!

```
1  import React, { useState } from 'react';
2  import { View, StyleSheet } from 'react-native';
3
4  import Header from './Header';
5
6  export default function Index() {
7    return (
8      <View style={styles.container}>
9        <Header title='Lista de compra' />
10      </View>
11    );
12  }
13
14
15  const styles = StyleSheet.create({
16    container: {
17      flex: 1,
18      backgroundColor: '#E1F5FE',
19    },
20  });
```



Volte ao “Header.js” e prepare-o para receber propriedades em sua chamada. Podemos fazer isso indicando no parâmetro de criação da função o comando “props”.

Em seguida acessamos os valores passados por meio das “props” com o comand:

“props.nome\_parâmetro\_definido\_chamada\_componente”



```
4  export default function Header(props) {  
5    return (  
6      <View style={styles.header}>  
7        <Text style={styles.text}> {props.title} </Text>  
8      </View>  
9    );  
10 }
```

Existem recursos em Java Script que permitem simplificar a estrutura do código, vamos utilizar um deles que é chamado de “destructuring”.

A sintaxe de atribuição via desestruturação (destructuring assignment) é uma expressão JavaScript que possibilita extrair dados de arrays ou objetos em variáveis distintas.

No parâmetro do “Header” insira entre “{ }” o nome da propriedade, assim podemos utiliza-la para exibir seu valor sem a necessidade do comando “props”

```
1  import React, { useState } from 'react';
2  import { Text, View, StyleSheet, } from 'react-native';
3
4  export default function Header({title}) {
5    return (
6      <View style={styles.header}>
7        <Text style={styles.text}> {title} </Text>
8      </View>
9    );
10 }
11
12 const styles = StyleSheet.create({
13   header: {
14     height: 60,
15     padding: 15,
16     backgroundColor: '#01579B',
17   },
18   text: {
19     color: '#E1F5FE',
20     fontSize: 23,
21     textAlign: 'center',
22   },
23 });
```



**Saiba mais em:**

[Entenda desestruturação no Javascript](#)  
[Atribuição via desestruturação](#)



Props também pode ter um valor padrão, este valor pode ser definido após a definição do código do componente funcional. Basta fazer uma referência a função onde a “prop” está definida como parâmetro, seguida do comando “defaultProps”, em seguida inicie um objeto contendo as propriedades e seus valores default.

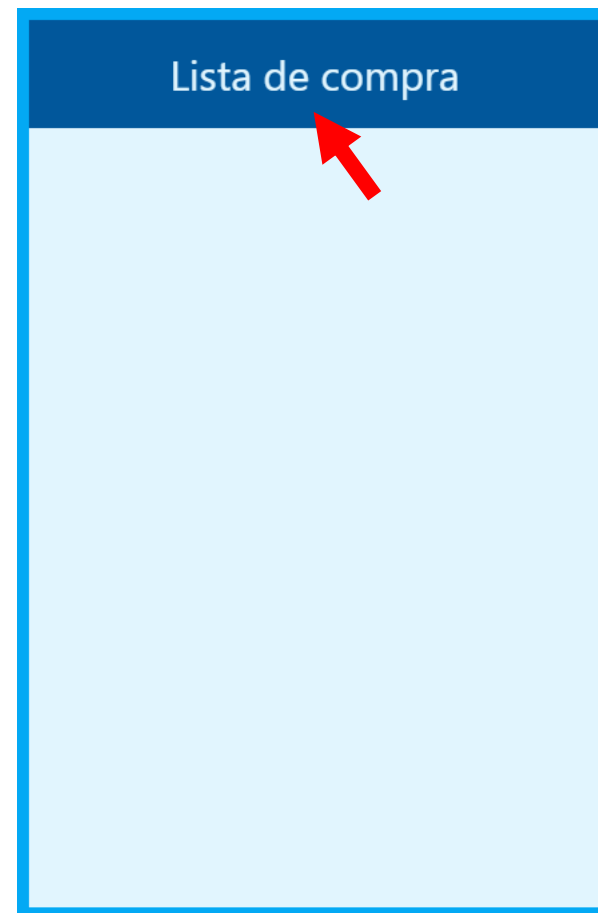
```
4   export default function Header({title}) {  
5     return (  
6       <View style={styles.header}>  
7         <Text style={styles.text}> {title} </Text>  
8       </View>  
9     );  
10  }  
11  
12  Header.defaultProps = {  
13    title: 'Lista de compra',  
14  }
```



Um valor “default” em uma propriedade será acionado, sempre que um componente for chamado sem que seja atribuído um valor para uma “prop” existente.

Assim quando voltamos ao “Index.js” e não passamos nenhuma propriedade na chamada do “Header”, o valor “default” será automaticamente apresentado.

```
6  export default function Index() {  
7    return (  
8      <View style={styles.container}>  
9        <Header />  
10     </View>  
11   );  
12 }  
13
```



Ainda no “Index.js” vamos importar duas bibliotecas que vão permitir gerar “ids” aleatórios e não repetidos em nosso aplicativo, seu uso será necessário, pois para trabalhar com “FlatList”, é necessário o uso de objetos que contenham campos específicos para armazenar um código de identificação.

Lembre-se que sempre que adicionamos uma nova biblioteca, esta deve ser referenciada, para isso adicionamos ela como uma “dependência” no arquivo “package.json”.

```
1 import React, { useState } from 'react';  
2 import { View, StyleSheet, FlatList, Text } from 'react-native';  
3 import 'react-native-get-random-values';
```

✖ components/Index.js (3:8) 'react-native-get-random-values' is not defined in dependencies. [Add dependency](#)

```
1 import React, { useState } from 'react';  
2 import { View, StyleSheet } from 'react-native';  
3 import { } from 'uuidv4';
```

✖ components/Index.js (3:18) 'uuidv4' is not defined in dependencies. [Add dependency](#)

```
1 import React, { useState } from 'react';  
2 import { View, StyleSheet } from 'react-native';  
3 import 'react-native-get-random-values';  
4 import { v4 as uuidv4 } from 'uuid';
```

[O que é UUID? Porque usá-lo?](#)

Dentro da função principal no “Index”, vamos inserir um “state”, nele serão armazenados os itens da lista de compras. Os itens adicionados irão ficar armazenados em um “array” de objetos, assim cada objeto será armazenado em uma posição do “array”.

Saiba mais sobre arrays em java script em:

[Array](#)

[JavaScript Arrays](#)

```
7   export default function Index() {  
8  
9     const [items, setItems] = useState([]);  
10  
11    return (  
12      <View style={styles.container}>  
13        <Header />  
14      </View>  
15    );  
16  };  
17 }
```



Um objeto é composto por chave e valor, em nosso exemplo temos o “id” que representa o código de identificação do item inserido e o “text” que vai representar o nome do item.


Saiba mais sobre objetos no java script em:

[O básico sobre objetos JavaScript](#)

[JavaScript - Criando Objetos](#)

[Trabalhando com objetos](#)

```
const [items, setItems] = useState([  
  {id: uuid(), text: 'Leite'},  
  {id: uuid(), text: 'Ovos'},  
  {id: uuid(), text: 'Farinha'},  
  {id: uuid(), text: 'Baunilha'},  
  {id: uuid(), text: 'Fermento'},  
]);
```



Agora que definimos o “state items” em uma array de objetos, já temos os recursos necessários para utilização de uma “FlatList”, mas ainda falta importar alguns componentes para uso.

Importe os componentes apontados abaixo e adicione a “FlatList” na função default.



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

```
7  export default function Index() {
8
9      const [items, setItems] = useState([
10         {id: uuid(), text: 'Leite'},
11         {id: uuid(), text: 'Ovos'},
12         {id: uuid(), text: 'Farinha'},
13         {id: uuid(), text: 'Baunilha'},
14         {id: uuid(), text: 'Fermento'},
15     ]);
16
17     return (
18         <View style={styles.container}>
19             <Header />
20             <FlatList /> ←
21         </View>
22     );
23 }
24
```

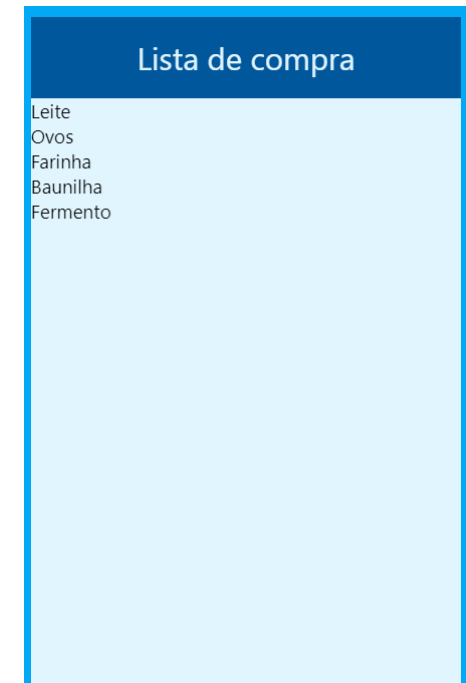
Vamos passar duas propriedades na declaração da “FlatList”:

**data:** que vai receber o array de objetos que criamos como um state.

**renderItem:** é a propriedade responsável por apresentar os valores dentro da FlatList. Neste exemplo utilizamos uma arrow function que tem como parâmetro passado o valor definido na propriedade “data” (que irá percorrer cada registro do array), como retorno da função será apresentado um “Text” com a chave de nome “text” apresentada.

Assim temos na saída os produtos da lista de compra.

```
<View style={styles.container}>
  <Header />
  <FlatList
    data={items}
    renderItem={ ({item}) => <Text>{item.text}</Text> }
  />
</View>
```



# FlatList

O “FlatList” foi utilizado, pois ele tem melhor desempenho em relação ao “ScrollView”, nele é necessário passar pelo menos dois parâmetro, que são “data” com a função de apontar um array de objetos e “renderItem” que permite configurar como cada item será apresentado dentro do “FlatList”.

**Saiba mais sobre FlatList e ScrollView nos links abaixo:**

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

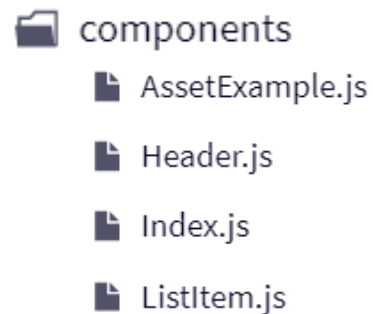
<https://reactnative.dev/docs/flatlist>

<https://dev.to/alexandrefreire/react-native-list-views-introducao-e-exemplo-14d7>

<https://oieduardorabelo.medium.com/react-native-criando-grids-com-flatlist-b4eb64e7dcd5>



Crie um novo componente com o nome de “ListItem.js” e deixe-o conforme o código abaixo:



```
1  import React from 'react';
2  import { Text, View, StyleSheet, TouchableOpacity } from 'react-native';
3
4  export default function ListItem({item}) {
5    return (
6      <TouchableOpacity style={styles.listItem}>
7        <View style={styles.listItemView}>
8          <Text style={styles.listItemText}>{item.text}</Text>
9        </View>
10     </TouchableOpacity>
11   );
12 }
13
14 const styles = StyleSheet.create({
15   |
16 });
```


Neste componente temos a propriedade “item” onde utilizamos o conceito de “destructuring”. O componente vai apresentar dentro de um “TouchableOpacity” uma “View” com um “Text”, dessa forma cada item da lista estará dentro de um container que vai agir como um botão.

```
1  import React from 'react';
2  import { Text, View, StyleSheet, TouchableOpacity } from 'react-native';
3
4  export default function ListItem({item}) {
5    return (
6      <TouchableOpacity style={styles.listItem}>
7        <View style={styles.listItemView}>
8          <Text style={styles.listItemText}>{item.text}</Text>
9        </View>
10     </TouchableOpacity>
11   );
12 }
13
14 const styles = StyleSheet.create({
15   |
16 });
```

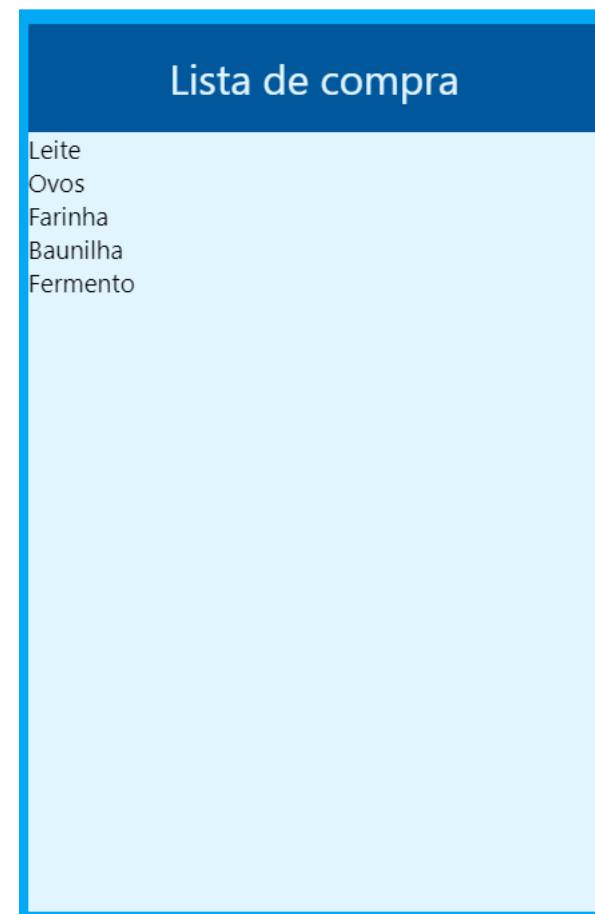


É possível passar um componente na propriedade “renderItem”, faremos isto no “Index.sj”. Faça a importação deste componente, e declare-o na propriedade “renderItem”.

```
5 import Header from './Header';  
6 import ListItem from './ListItem';
```

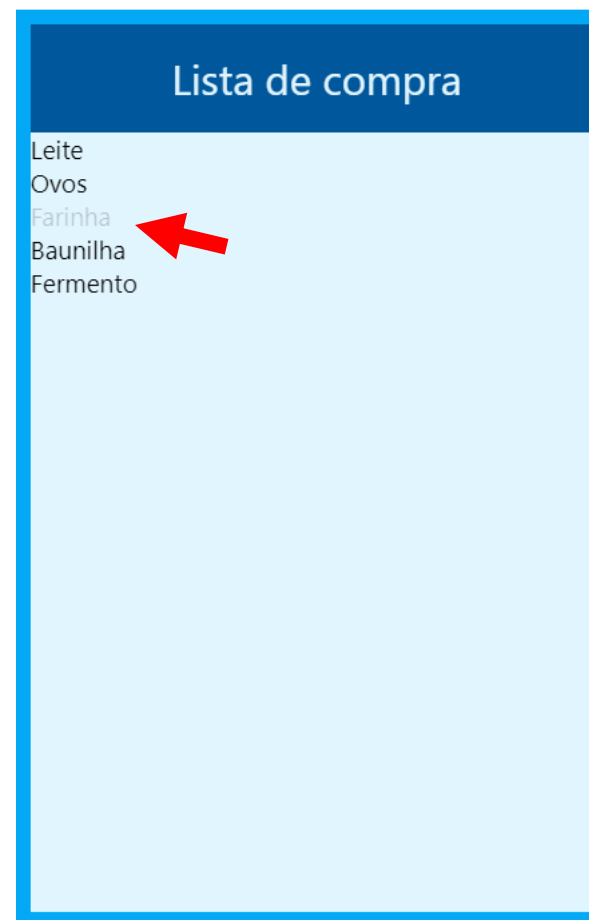


```
<FlatList  
  data={items}  
  renderItem={ ({item}) => <ListItem item={item} /> }  
/>
```



Ao testar, será possível clicar nos itens da lista, pois eles são renderizados dentro de um `TouchableOpacity`, isto nos permite saber qual item foi tocado.

```
<FlatList  
  data={items}  
  renderItem={ ({item}) => <ListItem item={item} /> }  
/>
```



Após a definição do componente ListIte.js, configure o “StyleSheet” conforme o código apresentado abaixo:

```
16  const styles = StyleSheet.create({
17    listItem: {
18      padding: 15,
19      backgroundColor: '#B3E5FC',
20      borderBottomWidth: 1,
21      borderColor: '#81D4FA',
22    },
23    listItemView: {
24      flexDirection: 'row',
25      justifyContent: 'space-between',
26      alignItems: 'center',
27    },
28    listItemText: {
29      fontSize: 18,
30      color: '#01579B',
31    },
32  });
```

Lista de compra
Leite
Ovos
Farinha
Baunilha
Fermento

Para representar o ícone de exclusão de um item da lista, será necessário importar uma biblioteca que permita trabalhar com ícones.

Para utilizá-la basta fazer uma referência e definir as propriedades apontadas no exemplo abaixo:

```
1 import React from 'react';
2 import { Text, View, StyleSheet, TouchableOpacity } from 'react-native';
3 import { } from '@expo/vector-icons';
```

✓ No errors, 1 warning

PROBLEMS LOGS

✗ [components/ListItem.js \(3:18\)](#) '@expo/vector-icons' is not defined in dependencies.

[Add dependency](#)

```
import { FontAwesome } from '@expo/vector-icons';
```

```
<View style={styles.listView}>
  <Text style={styles.listItemText}>{item.text}</Text>
  <FontAwesome name='remove' size={20} color='firebrick' />
</View>
```



Lista de compra	
Leite	✗
Ovos	✗
Farinha	✗
Baunilha	✗
Fermento	✗

# Apagando um item da lista

Adicione a função apontada ao lado após a declaração do state.

Nesta função passamos como parâmetro o “id” do item que será excluído. Já no comando de alterar o valor do state iremos passar como parâmetro seu conteúdo e como instrução da função o retorno de todos os itens que não tenham o “id” passado via parâmetro.

Dentro do comando “filter” executamos uma função que vai percorrer item por item e só retorna os que tem o “id” diferente.

Saiba mais sobre o uso do “filter” nos links abaixo:


- <https://wtricks.com.br/como-usar-filter-em-javascript/>
- <https://blog.betrybe.com/javascript/javascript-filter/>

```
8  export default function Index() {
9
10     const [items, setItems] = useState([
11         {id: uuid(), text: 'Leite'},
12         {id: uuid(), text: 'Ovos'},
13         {id: uuid(), text: 'Farinha'},
14         {id: uuid(), text: 'Baunilha'},
15         {id: uuid(), text: 'Fermento'},
16     ]);
17
18     function deleteItem (id) {
19         setItems( prevItems => {
20             return prevItems.filter(item => item.id !== id);
21         });
22     }
23
24     return (
```

# Passando função como parâmetro

Agora no “renderItem” do “FlatList” iremos passar outra propriedade, desta vez o valor passado será a função “deleteItem” através da “prop” com o mesmo nome.

```
<FlatList  
  data={items}  
  renderItem={ ({item}) => <ListItem item={item} deleteItem={deleteItem} /> }  
/>
```





Volte ao arquivo “ListItem” para acrescentarmos a função de apagar itens como uma prop. No evento “onPress” inserimos a execução da função passando o “id” do item selecionado.

Após fazer as alterações teste o programa.

```
5 export default function ListItem({item, deleteItem}) {  
6   return (  
7     <TouchableOpacity style={styles.listItem}>  
8       <View style={styles.listItemView}>  
9         <Text style={styles.listItemText}>{item.text}</Text>  
10        <FontAwesome  
11          name='remove'  
12          size={20}  
13          color='firebrick'  
14          onPress={ () => deleteItem(item.id) }  
15        />  
16      </View>  
17    </TouchableOpacity>  
18  );  
19 }
```

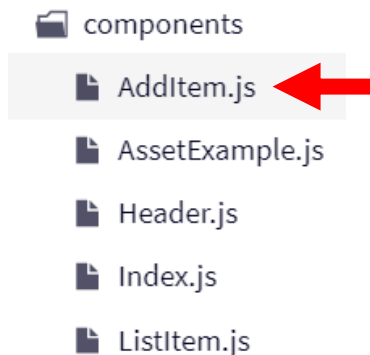


Lista de compra	
Leite	✖
Ovos	✖
Farinha	✖
Baunilha	✖
Fermento	✖

Lista de compra	
Leite	✖
Ovos	✖
Farinha	✖
Fermento	✖

Agora vamos criar um novo componente com a função de adicionar itens a lista.

Insira o código ao lado para definir o componente.



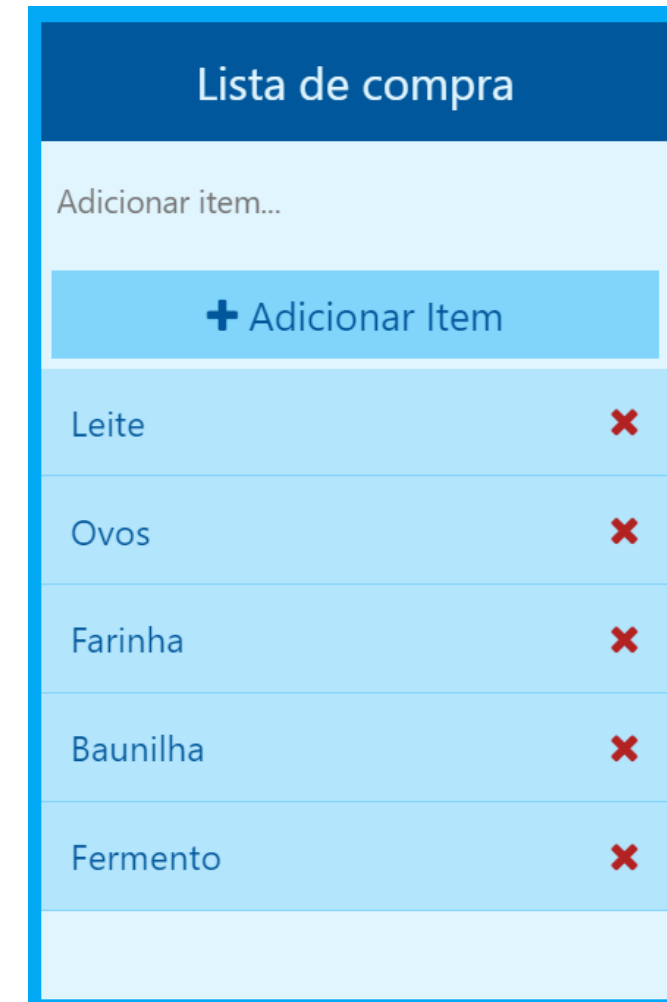
```
1  import React, { useState } from 'react';
2  import { Text, View, StyleSheet, TextInput, TouchableOpacity } from 'react-native';
3  import { FontAwesome } from '@expo/vector-icons';
4
5  export default function AddItem({title}) {
6    return (
7      <View>
8        <TextInput placeholder='Adicionar item...' style={styles.input} />
9        <TouchableOpacity style={styles.button}>
10          <Text style={styles.buttonText}>
11            <FontAwesome name='plus' size={20} /> Adicionar Item
12          </Text>
13        </TouchableOpacity>
14      </View>
15    );
16  }
```

```
18  const styles = StyleSheet.create({
19    input: {
20      height: 60,
21      padding: 8,
22      fontSize: 16,
23    },
24    button: {
25      backgroundColor: '#81D4FA',
26      padding: 9,
27      margin: 5,
28    },
29    buttonText: {
30      color: '#01579B',
31      fontSize: 20,
32      textAlign: 'center',
33    },
34  });
```

Volte ao “index.js”, faça o import do “AddItem” e adicione-o após o “Header”, o aplicativo deve ficar como na imagem abaixo.

```
6  import Header from './Header';
7  import ListItem from './ListItem';
8  import AddItem from './AddItem';
```

```
26  return (
27    <View style={styles.container}>
28      <Header />
29      <AddItem />
30      <FlatList
```



Volte ao componente “AddItem”, pois agora vamos definir um “state” com a função de armazenar o valor inserido, este, por sua vez representa o valor que será adicionado na lista.

```
5  export default function AddItem({title}) {  
6  
7    const [text, setText] = useState('');   
8  
9    return (  
10     <View>  
11       <TextInput  
12         placeholder='Adicionar item...'  
13         style={styles.input}  
14         onChangeText={ (textValue) => setText(textValue) }   
15       />  
16     <TouchableOpacity style={styles.button}>
```

No “Index.js” adicione uma função com o objetivo de adicionar itens a lista. Esta função tem como parâmetro a ser passado o “item” a ser adicionado, nossa lista será alterado com o comando “setItems” assim como alteramos para excluir, iremos executar uma arrow function que passa como parâmetro o valor atual do state “items” e vai retornar um array...

```
20     function deleteItem (id) {
21         setItems( prevItems => {
22             return prevItems.filter(item => item.id !== id);
23         });
24     }
25
26     function addItem (item) {
27         setItems(prevItems => {
28             return [...prevItems, item];
29         });
30     }
31
32     return (
```

... O retorno deste array será um item com “id” e “texto” passados como um array com apenas um objeto (que representa o item a ser adicionado), junto com o parâmetro da função que trás os itens já existentes na lista.

Para fazer esta junção de dois arrays em um, utilizamos um operador do java script chamado “spread”.


```
26  function addItem (item) {  
27      setItems(prevItems => {  
28          return [{id: uuid(), text: item}, ...prevItems];  
29      });  
30  }
```



**Saiba mais sobre o operador Spread em:**


[https://www.luiztools.com.br/post/4-segredos-do-operador-spread-em-javascript/?gclid=Cj0KCQjwyN-DBhCDARIsAFOELTnLxSfkqMVcNOS-1qotaXAoDy6rmYQsVE1vmgFIVL0VWnRWQb6eF5EaAhEGEALw\\_wcB](https://www.luiztools.com.br/post/4-segredos-do-operador-spread-em-javascript/?gclid=Cj0KCQjwyN-DBhCDARIsAFOELTnLxSfkqMVcNOS-1qotaXAoDy6rmYQsVE1vmgFIVL0VWnRWQb6eF5EaAhEGEALw_wcB)

Em java script é comum a utilização de recursos que deixam o código mais enxuto, podemos por exemplo, mudar o nome do parâmetro da função pra “text” e passar o valor para a chave “text” do objeto com o mesmo nome do parâmetro...



```
function addItem (text) {  
  setItems(prevItems => {  
    return [{id: uuid(), text: text}, ...prevItems];  
  });  
}
```

Como temos dois valores com o mesmo nome, o java script permite que passemos apenas um nome, assim ele já entende que queremos passar um valor para uma chave de um objeto.



```
function addItem (text) {  
  setItems(prevItems => {  
    return [{id: uuid_v4(), text}, ...prevItems];  
  });  
}
```

Com a função criada, devemos passa-la na chamada do componente “AddItem” como um parâmetro.

```
return (  
  <View style={styles.container}>  
    <Header />  
    <AddItem addItem={addItem} />  
    <FlatList  
      data={items}  
      renderItem={({item}) => <ListItem item={item} deleteItem={deleteItem} /> }  
    />  
  </View>  
)
```

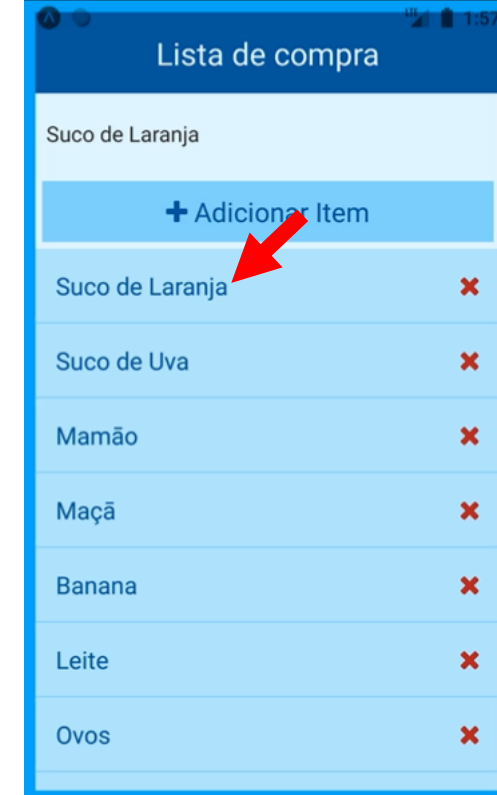
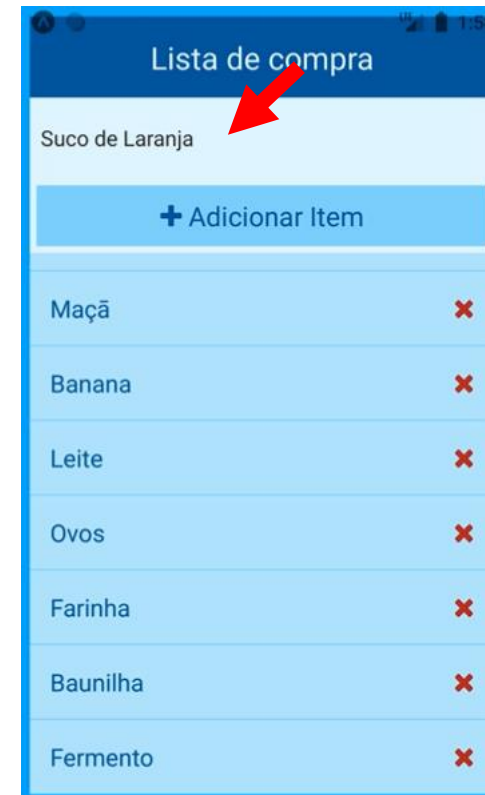




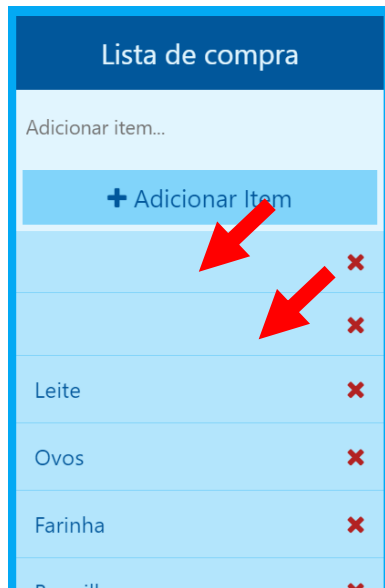
Volte ao componente “AddItem” para definir a “prop” “addItem” como parâmetro na chamada deste componente e adicione o evento “onPress” ao “TouchableOpacity” que representa o botão para adicionar item. Teste o programa.



```
5 export default function AddItem({title, addItem}) {
6
7   const [text, setText] = useState('');
8
9   return (
10     <View>
11       <TextInput
12         placeholder='Adicionar item...'
13         style={styles.input}
14         onChangeText={ (textValue) => setText(textValue) }
15       />
16       <TouchableOpacity style={styles.button} onPress={ () => addItem(text)}>
17         <Text style={styles.buttonText}>
18           <FontAwesome name='plus' size={20} /> Adicionar Item
19         </Text>
20       </TouchableOpacity>
21     </View>
22   );
23 }
```



Nosso aplicativo ainda tem um problema, pois permite adicionar itens em branco a lista, para resolver este problema deixe a função “AddItem” como apresentado no exemplo, neste caso será necessário importar a biblioteca “Alert” no “Index.js”. Teste no emulador!



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

```
26 function addItem (text) {  
27   if (text === '') {  
28     Alert.alert('Erro', 'O valor do item não pode ser vazio', [{text: "OK"}] );  
29   } else {  
30     setItems(prevItems => {  
31       return [{id: uuid(), text}, ...prevItems];  
32     });  
33   }  
34 }
```

**Mais sobre a biblioteca Alert em:**  
<https://reactnative.dev/docs/alert>



Ao verificar os Logs do aplicativo, observa-se que existe um comando “depreciado”, em linguagens modernas que recebem mudanças constantes é comum acontecer isto. Um comando “deprecated” está desatualizado e com o tempo não ira mais funcionar, é muito importante ficar atento a este tipo de aviso.

Procure na internet como resolver o problema para evitar que seu aplicativo apresente erros.

The image shows a sequence of steps to resolve a deprecation warning. At the top, a Chrome console message states: "Chrome: uuidv4() is deprecated. Use v4() from the uuid module instead." A red arrow points to this message. Below, a Google search for "uuidv4() is deprecated" is shown, with a red arrow pointing to the search bar. The search results show a Stack Overflow link titled "React: Deep requiring is deprecated as of uuid, Please ...". A red arrow points to the Stack Overflow logo. The Stack Overflow page shows the question details, including the date "17 de mar. de 2020" and the error message. The "2 Answers" section is visible, with the top answer having 12 votes. A red circle highlights the correct code snippet in the answer: 

```
import { v4 as uuid_v4 } from "uuid";  
uuid_v4()
```


A sugestão encontrada foi a de modificar a forma que a biblioteca é importada, dessa forma será necessário atualizar as dependências do aplicativo, ao fazer isso iremos gerar alguns erros...

```
import { v4 as uuidv4 } from 'uuid';
```




✖ components/Index.js (4:31) 'uuid' is not defined in dependencies. [Add dependency](#)

```
12  const [items, setItems] = useState([
13    {id: uuid(), text: 'Leite'},
14    {id: uuid(), text: 'Ovos'},
15    {id: uuid(), text: 'Farinha'},
16    {id: uuid(), text: 'Baunilha'},
17    {id: uuid(), text: 'Fermento'},
18  ]);
```



```
26  function addItem (text) {
27    if (text === '') {
28      Alert.alert('Erro', 'O valor do item não pode ser vazio', [{text: "OK"}] );
29    } else {
30      setItems(prevItems => {
31        return [{id: uuid(), text}, ...prevItems];
32      });
33    }
34  }
```



... E o programa deixa de funcionar, porém basta atualizar os locais apontados anteriormente para a nova versão do “uuid”

PROBLEMS LOGS

Chrome: Error: "uuid is not defined" in ReferenceError: uuid is not defined << at Index (components/Index.js.js:13:10 << << at Vo ([snack internals] ...

Chrome: Error: "uuid is not defined" in ReferenceError: uuid is not defined << at Index (components/Index.js.js:14:10 << << at Vo ([snack internals] ...

Chrome: Error: "uuid is not defined" in ReferenceError: uuid is not defined << at Index (components/Index.js.js:13:10 << << at Vo ([snack internals] ...

```
const [items, setItems] = useState([
  {id: uuidv4(), text: 'Leite'},
  {id: uuidv4(), text: 'Ovos'},
  {id: uuidv4(), text: 'Farinha'},
  {id: uuidv4(), text: 'Baunilha'},
  {id: uuidv4(), text: 'Fermento'},
]);
```

```
return [{id: uuidv4(), text}, ...prevItems];
```

**Did you know:** You can turn off automatic updates under Devices in the footer?

#### uuid is not defined

ReferenceError: uuid is not defined  
at Index  
(components/Index.js.js:13:10

at Vo ([snack internals]  
at ms ([snack internals]  
at dl ([snack internals]  
at sl ([snack internals]  
at Zs ([snack internals]  
at [snack internals]  
at t.unstable\_runWithPriority  
([snack internals]  
at qa ([snack internals]  
at Ja (https://snack-web-  
player.s3.us-west-  
1.amazonaws.com/v2/40/static/js/2.  
ca3bea86.chunk.js:2:4196363)

Para finalizar iremos ver um conceito muito importante ao programar em dispositivos móveis, que é o da identificação da plataforma. Em nosso caso será útil, pois a mensagem de alerta quando tentamos inserir um valor vazio na lista, só funciona quando estamos utilizando a versão Android ou IOS, na Web não temos a mensagem exibida.

Faça o import da biblioteca “Platform” e ainda no “index.js” antes do “return” da função default, insira o comando apontado! Assim ao executarmos o programa um log será gerado apresenta em que plataforma estamos!

```
2  import { View, StyleSheet, FlatList, Text, Alert, Platform } from 'react-native';
```

```
36  console.log(Platform.OS);  
37  
38  return (
```

PROBLEMS

LOGS

Chrome: web

PROBLEMS

LOGS

Android SDK built for x86: android

Agora que vimos como funciona a identificação de plataforma, altere a função “addItem” onde iremos personalizar a apresentação da mensagem de alerta conforme a plataforma.

```
26  function addItem (text) {
27    if (text === '') {
28      if (Platform.OS === 'web') {
29        alert('O valor do item não pode ser vazio');
30      } else {
31        Alert.alert('Erro', 'O valor do item não pode ser vazio', [{text: "OK"}] );
32      }
33    } else {
34      setItems(prevItems => {
35        return [{id: uuid_v4(), text}, ...prevItems];
36      });
37    }
38  }
```

**Saiba mais sobre a biblioteca Platform em:**

<https://docs.expo.io/versions/v37.0.0/react-native/platform-specific-code/>

O comando utilizado para identificar a plataforma pode ser simplificado utilizando o conceito de operador ternário.

**Saiba mais sobre este tipo de operador em:**

[https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Reference/Operators/Conditional\\_Operator](https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Reference/Operators/Conditional_Operator)

```
26     function addItem (text) {  
27         if (text === '') {  
28             Platform.OS === 'web'  
29             ? alert('O valor do item não pode ser vazio')  
30             : Alert.alert('Erro', 'O valor do item não pode ser vazio', [{text: "OK"}] );  
31         } else {  
32             setItems(prevItems => {  
33                 return [{id: uuid_v4(), text}, ...prevItems];  
34             });  
35         }  
36     }
```



# Referências

- [https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Reference/Functions/Arrow functions](https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Reference/Functions/Arrow_functions)
- <https://developerplus.com.br/como-detectar-a-plataforma-com-react-native/#:~:text=Al%C3%A9m%20de%20detectar%20a%20plataforma,em%20que%20voc%C3%AA%20est%C3%A1%20executando>
- <https://www.youtube.com/watch?v=Hf4MJH0jDb4>
- <https://www.igortuag.com/entenda-desestrutur%C3%A7%C3%A3o-no-javascript-destructuring-assignment/>
- [https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Reference/Operators/Destructuring assignment](https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Reference/Operators/Destructuring_assignment)
- <https://medium.com/trainingcenter/o-que-%C3%A9-uuid-porque-us%C3%A1-lo-ad7a66644a2b>