
# Introdução React Native

Ewerton J. Silva

ewerton.silva41@etec.sp.gov.br

# Desenvolvimento mobile

A algum tempo atrás o desenvolvimento mobile era realizado utilizando “Objective-C” para IOS, hoje em dia é mais utilizado o “Swift”, para Android tínhamos antes o Java e hoje em dia o Kotlin.

Para IOS tínhamos arquivos com a extensão .ipa e o java gera arquivo .apk no Android, este arquivos podem ser instalados em nosso dispositivo para execução das aplicações desenvolvidas por essas linguagens.

Assim tínhamos que programar um aplicativo para cada plataforma.

# React Native

A partir de 2015 o react native começou a ser amplamente utilizado, ele permite que criemos uma aplicação react que gera arquivos .ipa e .apk.

As aplicações react native são nativas, pois ele implemente dentro da nossa aplicação o java script core, assim o sistema operacional mobile conseguem interpretar o código java script para construção e manipulação da interface do aplicativo.

Assim esse código não é convertido em código nativo como acontece com o xamarin em C#, que precisa de fazer a conversão do código gerado para o android ou o ios.

Com isso temos alta performasse no uso dos aplicativos, pois o componentes que inserimos no react native são nativos em cada uma das plataformas para o qual está sendo utilizado.

# Criação do projeto

Para o aprendizado dos conceitos de react native não é necessário configurar e instalar todo o ambiente no computador, no site https://expo.io/ é possível codificar e testar sem a necessidade de nenhuma instalação.

Com as ferramentas, serviços e React da Expo, você pode criar, implantar e iterar rapidamente em aplicativos nativos do Android, iOS e da web a partir da mesma base de código JavaScript.

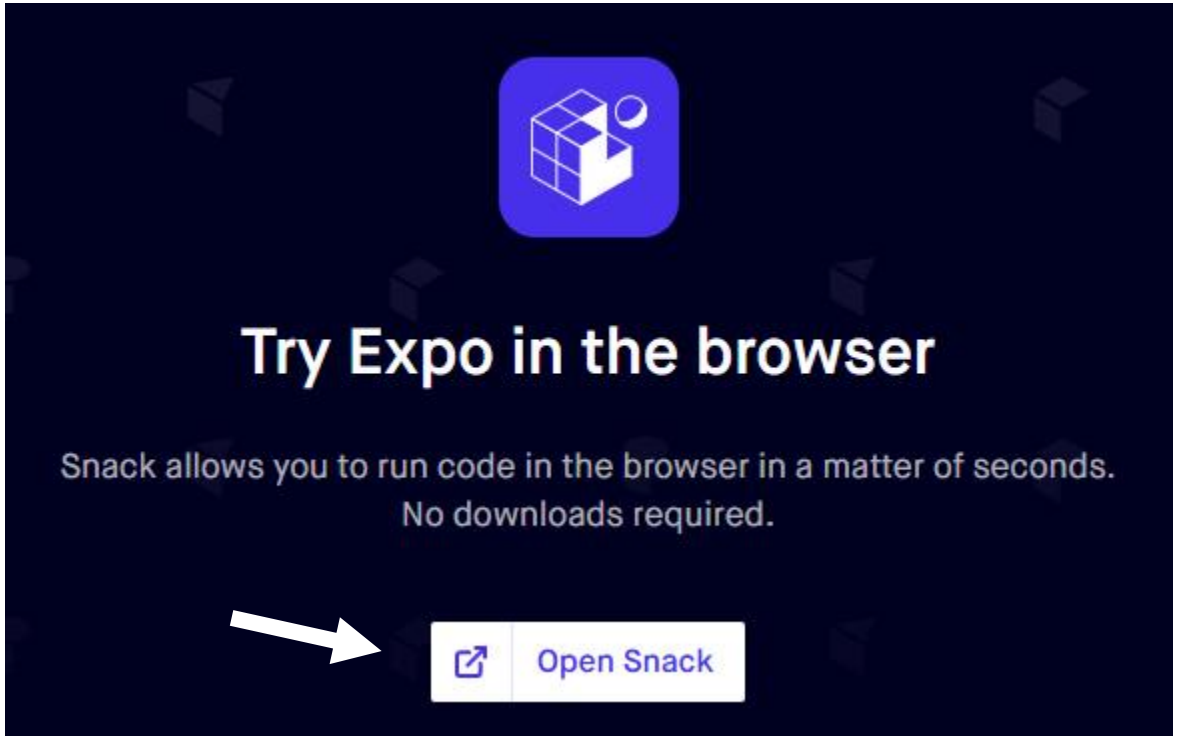
Com acesso a recursos do dispositivo, como câmera, localização, notificações, sensores, controle de toque e muito mais, tudo com APIs universais.

O serviço de criação fornece binários prontos para a loja de aplicativos e gerencia certificados, sem a necessidade de tocar no Xcode ou no Android Studio.

As atualizações sem fio permitem que você atualize seu aplicativo a qualquer momento, sem os problemas e os atrasos de envio à loja.

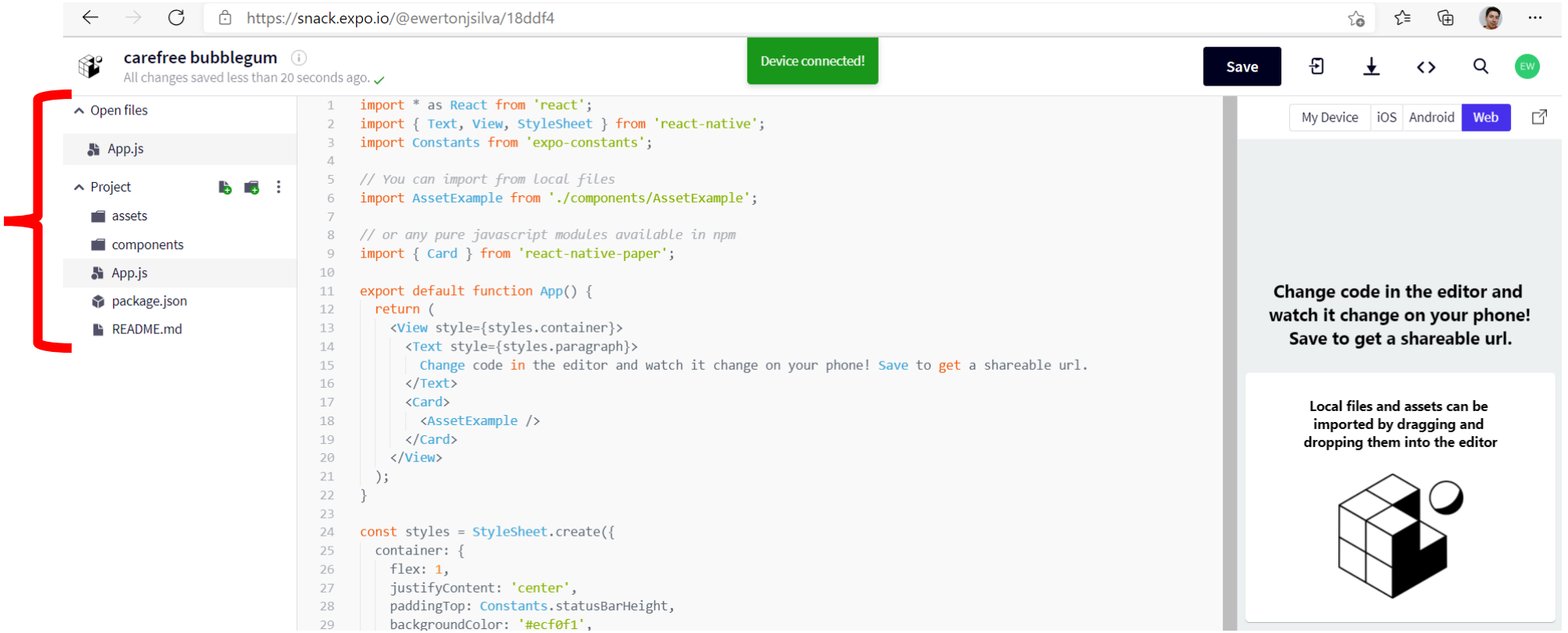
# Acesso ao Expo via navegador

Acesse o site e clique em “Open Snack” para iniciar o editor de códigos:



# Visão geral do Expo via navegador

Uma nova janela será iniciada, onde temos no lado esquerdo os arquivos de um projeto inicial padrão.

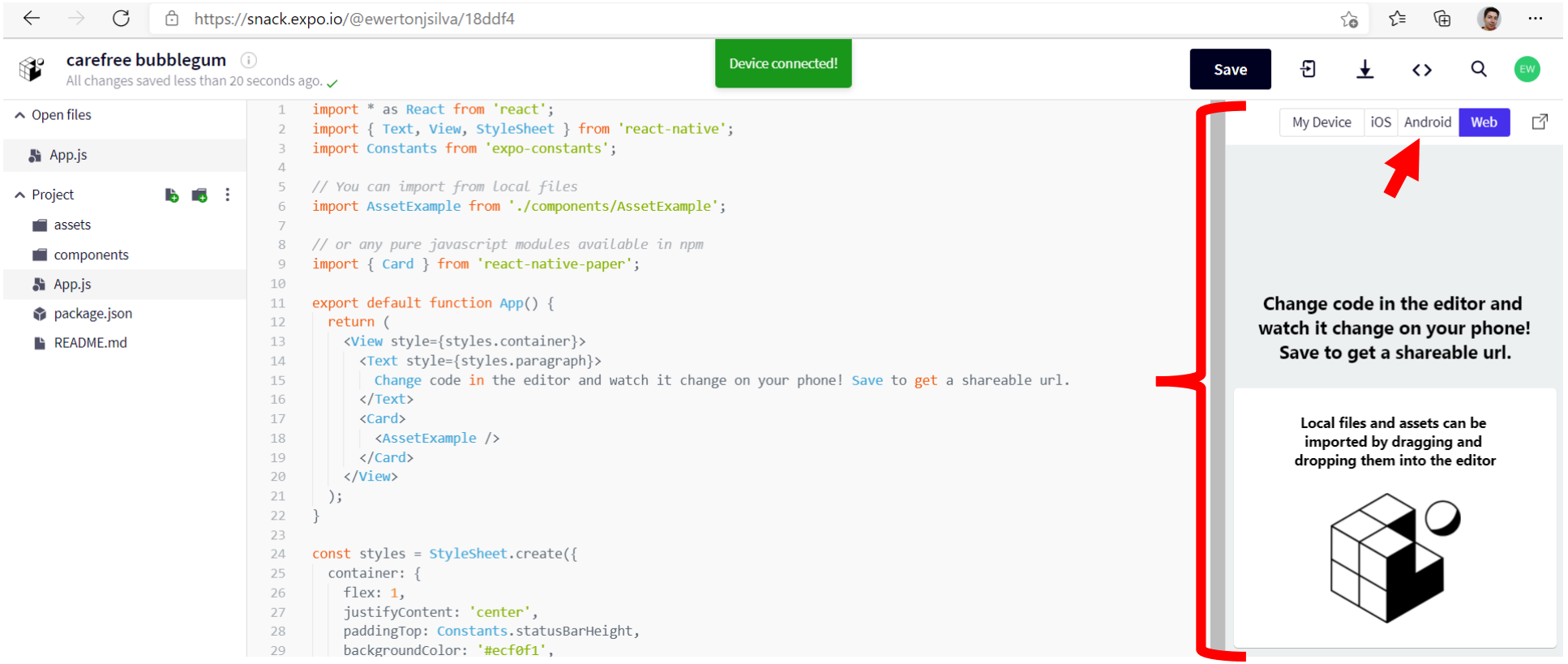


# Visão geral do Expo via navegador

No meio a seção de código.

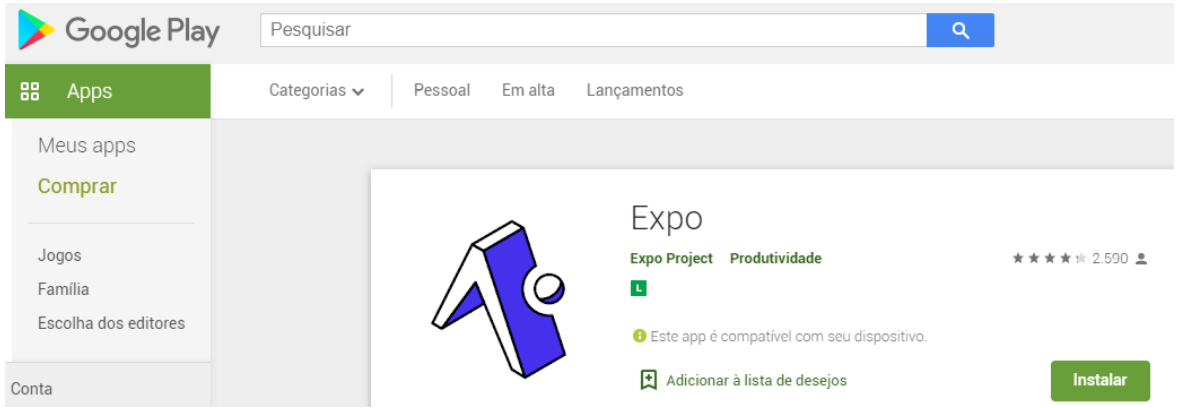
# Visão geral do Expo via navegador

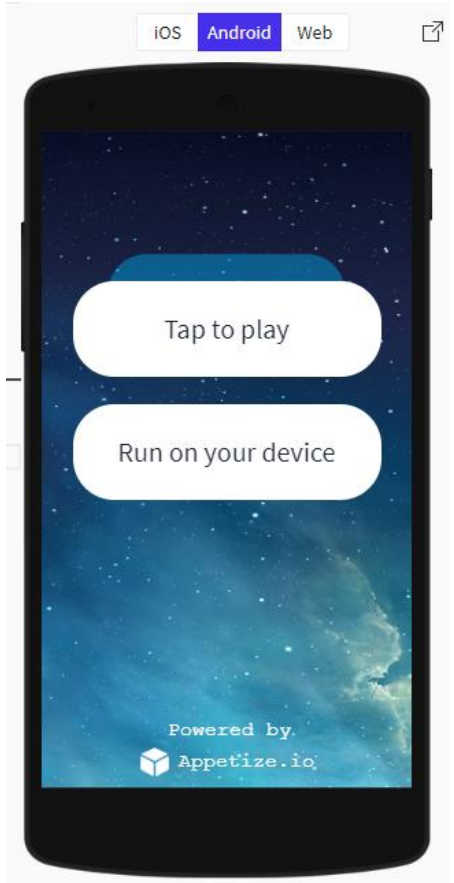
No canto direito a visão de execução. Onde é possível alternar para Android que é o foco da nossa disciplina.



# Visualização do aplicativo

Ao clicar na opção de teste no Android a tela ao lado é apresentada e aqui temos a opção de iniciar o emulador ou testar a aplicação em um dispositivo com android (Run on your device). Para testar a aplicação em seu dispositivo Android é necessário que o aparelho tenha o aplicativo expo instalado e esteja conectado a internet.

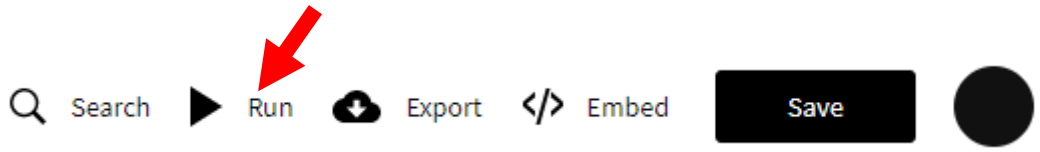


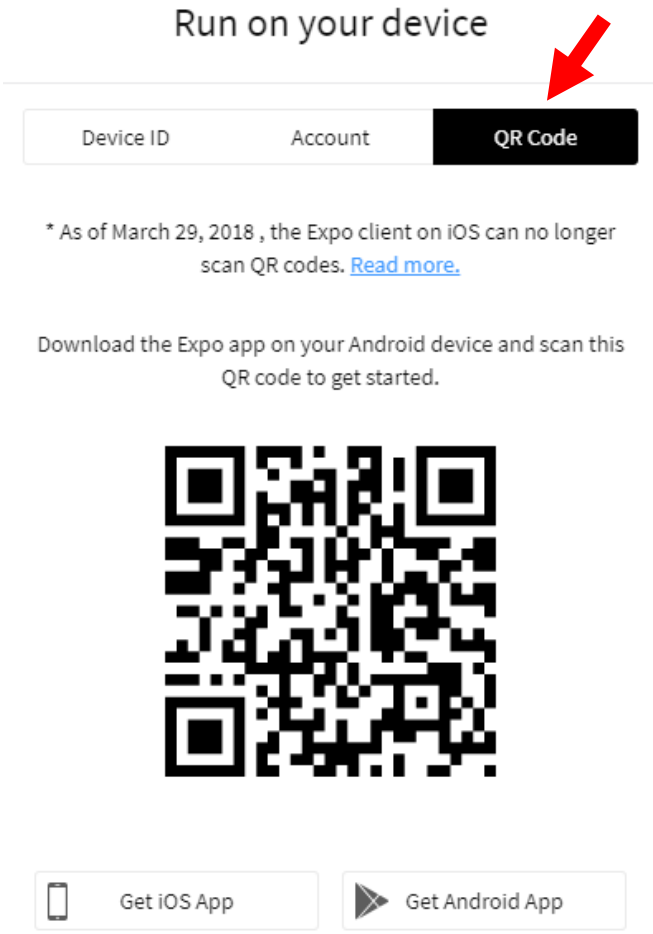



# Visualização do aplicativo

Clicando em Run on your device a janela abaixo será apresentada com um QRCode que deve ser lido com o aplicativo Expo instalado no Smartphone.

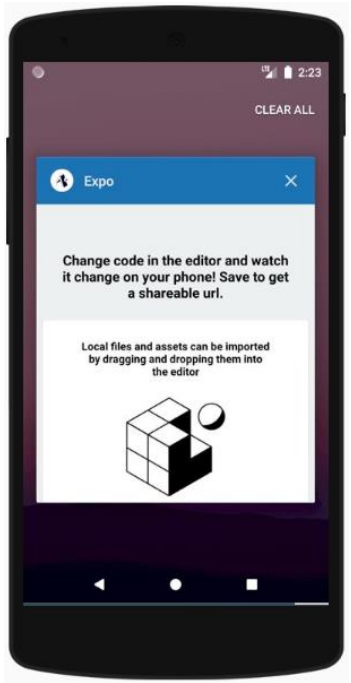
Esta opção também pode ser acessada clicando no botão “Run” no menu apresentado do lado direito da janela do navegador.

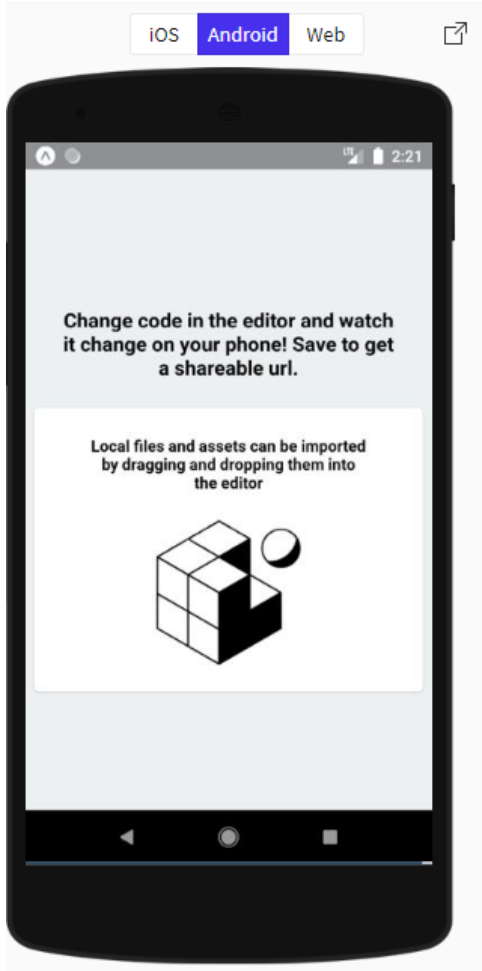


# Visualização do aplicativo

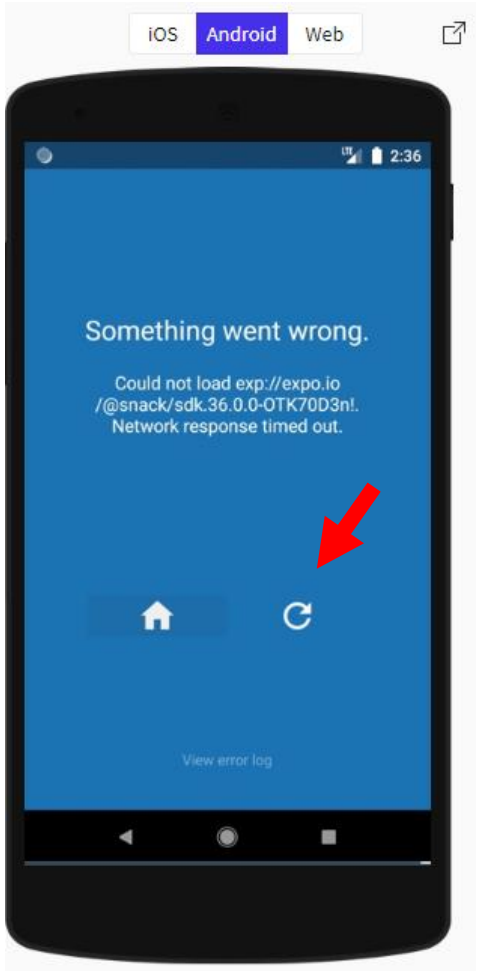
Ao alternar para o Android e clicar na opção “Tap to play” temos a visão do nosso código sendo executado em um emulador virtual de android.

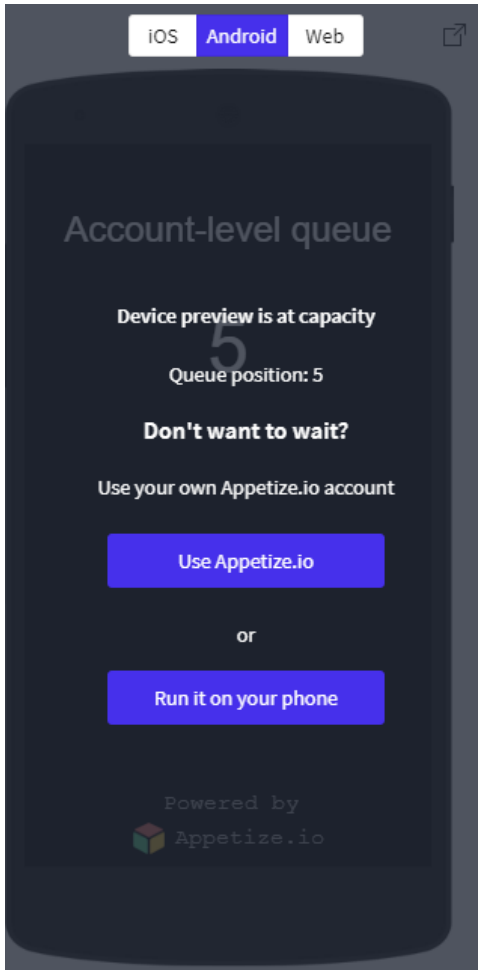



# Visualização do aplicativo

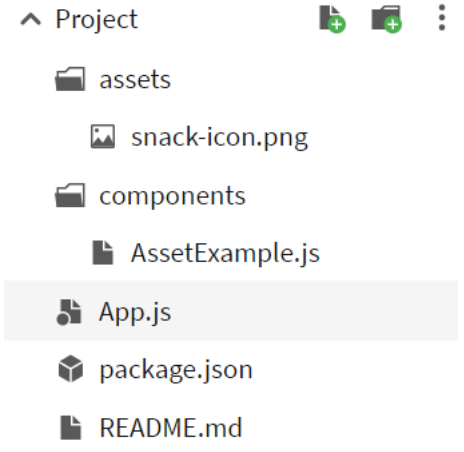
Em alguns momentos o emulador pode passar por instabilidades ou mesmo deixar você em uma fila de espera para execução de testes. Caso isso aconteça aguarde ou execute pela opção do seu Smartphone ou atualize a tela caso tenha problemas.

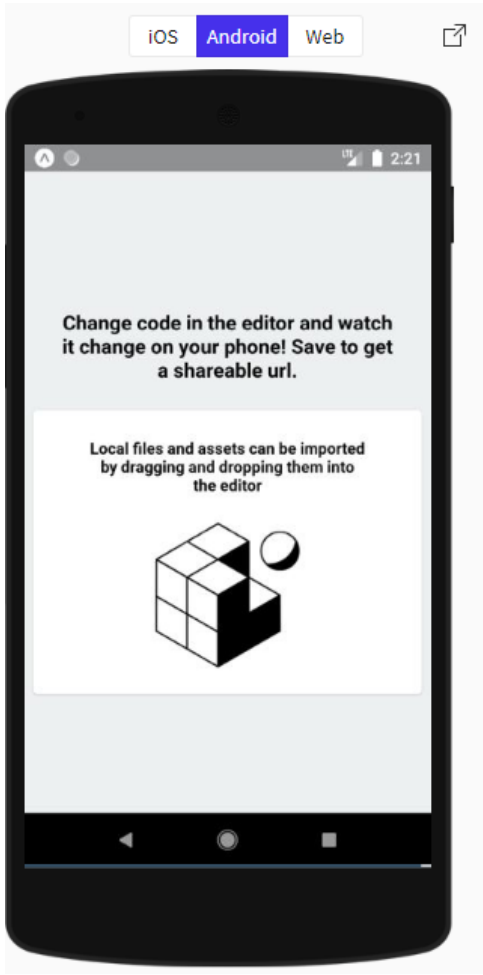


# Projeto padrão

No projeto inicial do expo temos a pasta “assets” onde fica a imagem exibida, a pasta “components“ onde está a programação do componente que apresenta o texto, seguido da imagem. Em seguida temos o arquivo “App.js” que é responsável pela apresentação do conteúdo visualizado no dispositivo. Por fim o arquivo “package.json” que serve para armazenar as dependências utilizadas no projeto e “README.md” que é um arquivo de ajuda.



# AssetExample.js

É o componente que retorna uma View com um texto e uma imagem, além disso, tem estilos JSX configurados que são aplicados nos objetos deste componente.

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

export default function AssetExample() {
  return (
    <View style={styles.container}>
      <Text style={styles.paragraph}>
        Local files and assets can be imported by dragging and dropping them into the editor
      </Text>
      <Image style={styles.logo} source={require('../assets/snack-icon.png')} />
    </View>
  );
}
```

```jsx
const styles = StyleSheet.create({
  container: {
    alignItems: 'center',
    justifyContent: 'center',
    padding: 24,
  },
  paragraph: {
    margin: 24,
    marginTop: 0,
    fontSize: 14,
    fontWeight: 'bold',
    textAlign: 'center',
  },
  logo: {
    height: 128,
    width: 128,
  }
});
```

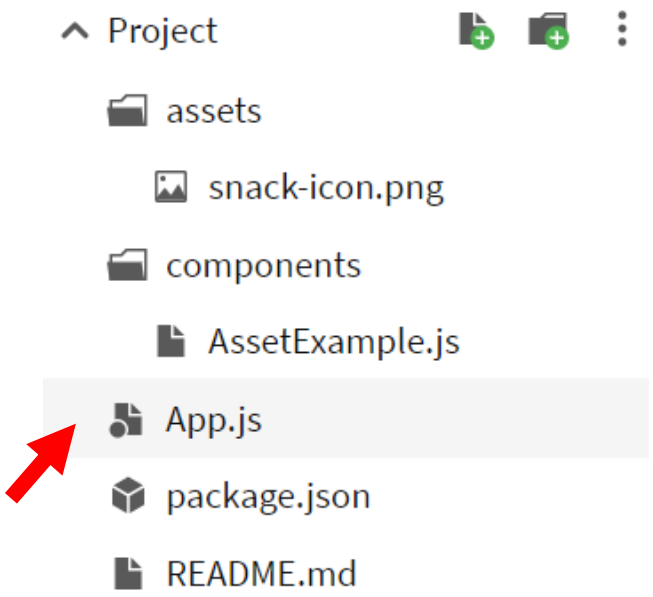
# App.js

É o componente principal, a tela inicial do aplicativo, onde temos uma View apresentando um Text com uma frase e um card com a renderização do componente AssetExample.js, além das configurações de estilo deste componente.

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

// You can import from local files
import AssetExample from './components/AssetExample';

// or any pure javascript modules available in npm
import { Card } from 'react-native-paper';

export default function App() {
  return (
    <View style={styles.container}>
      <Text style={styles.paragraph}>
        Change code in the editor and watch it change on your phone! Save to get a shareable url.
      </Text>
      <Card>
        <AssetExample />
      </Card>
    </View>
  );
}

const styles = StyleSheet.create({
  container: {
    flex: 1,
    justifyContent: 'center',
    paddingTop: Constants.statusBarHeight,
    backgroundColor: '#ecf0f1',
    padding: 8,
  },
  paragraph: {
    margin: 24,
    fontSize: 18,
    fontWeight: 'bold',
    textAlign: 'center',
  },
});
```

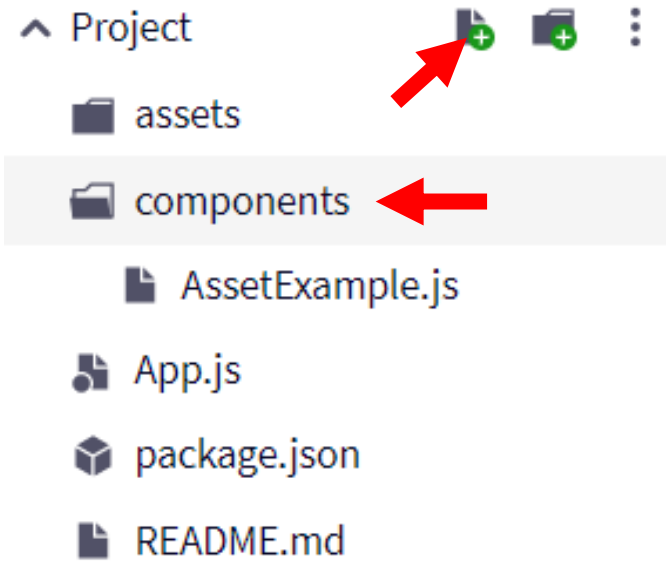
# Projeto no Expo

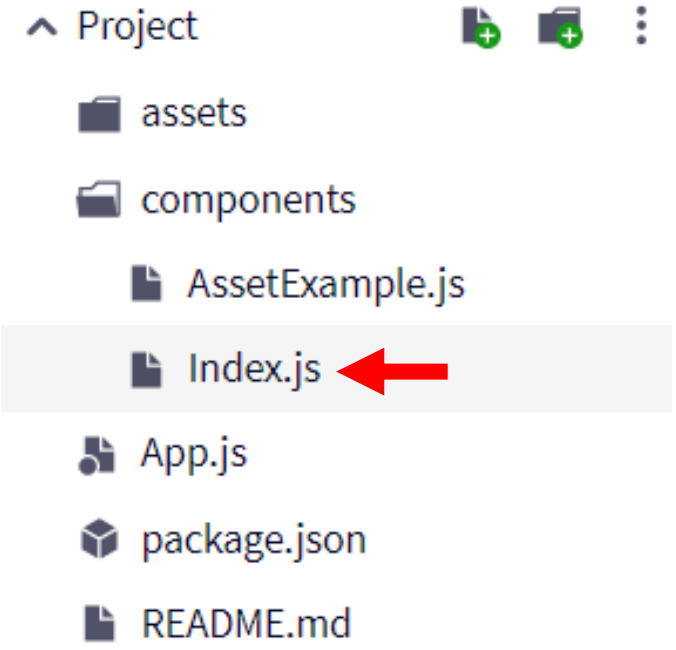
O arquivo App.js não deve ser movido para nenhuma pasta, pois é o principal arquivo do projeto, ele que é carregado quando executamos o aplicativo.



# Criando nosso componente

Selecione a pasta “components” e depois clique em “new file” para criar um arquivo com o nome “Index.js”



# Estrutura de um componente

Um componente básico pode até quatro seções de código definidas, sendo a primeira referente ao componentes que serão importados para ela, depois o código que representa a programação do componente, a definição de qual a classe padrão que será exportada e o local onde inserimos o código de estilização.

```
1   // importar componentes
2
3   // declaração de funções do componente
4
5   // apontar a função default
6
7   // jsx (estilização)
```

# IMPORT

Inicialmente importamos o componente React que faz a referencia a biblioteca de código que utilizaremos, além dos objetos “View” e “Text” do react-native, estes serão utilizados para apresentar o conteúdo do nosso componente.

```
1   // importar componentes
2   import React from 'react';
3   import { View, Text } from 'react-native';
```

Documentação sobre Elementos do React em: https://react-native-elements.github.io/react-native-elements/docs/text.html

# Definindo função

O componente é declarado como uma função, onde definimos seu nome, entre parênteses são definidos os parâmetros, caso não exista deixamos vazio.

```
1   // importar componentes
2   import React from 'react';
3   import { View, Text } from 'react-native';
4
5   // declaração de funções do componente
6   function Index() {
7
8   }
```

# Retorno da função

Toda função deve retornar um elemento que é chamado através do código “return”.

```
5   // declaração de funções do componente
6   function Index() {
7     return (
8       
9     );
10  }
```

# Retorno de função

O comando “return” irá retornar uma “View” que representa um container, este que terá nosso texto a ser exibido utilizando “Text”

```
5   // declaração de funções do componente
6   function Index() {
7     return (
8       <View>
9         <Text>
10          Introdução React Native
11        </Text>
12      </View>
13    );
14  }
```

# Definindo classe padrão

Para finalizar devemos definir qual será a função será executada por padrão quando nosso componente for chamado.

```
5   // declaração de funções do componente
6   function Index() {
7     return (
8       <View>
9         <Text>
10          Introdução React Native
11        </Text>
12      </View>
13    );
14  }
15
16  // apontar a função default
17  export default Index;
```

# Importando componente criado

Agora volte ao componente App.js.

Para visualizar o nosso componente, será necessário importa-lo, para referencia-lo basta inserir a linha apontada abaixo, onde indicamos o nome da classe que foi definida como principal e o caminho onde o arquivo se encontra, nesta seção não é necessário inserir o “.js” da extensão do arquivo, pois nosso framework se encarrega disso.

```
1   import * as React from 'react';
2   import { Text, View, StyleSheet } from 'react-native';
3   import Constants from 'expo-constants';
4
5   // You can import from local files
6   import AssetExample from './components/AssetExample';
7
8   // or any pure javascript modules available in npm
9   import { Card } from 'react-native-paper';

11  import Index from './components/Index';
```

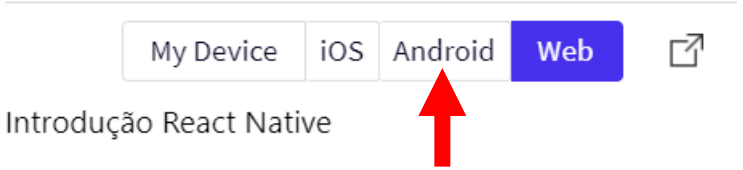
# App.js

Deixe a definição da função principal do “App.js” chamando apenas o “Index” que importamos como apontado ao lado.

```
1   import * as React from 'react';
2   import { Text, View, StyleSheet } from 'react-native';
3   import Constants from 'expo-constants';
4
5   // You can import from local files
6   import AssetExample from './components/AssetExample';
7
8   // or any pure javascript modules available in npm
9   import { Card } from 'react-native-paper';
10
11  import Index from './components/Index';
12
13  export default function App() {
14    return (
15      <Index />
16    );
17  }
```

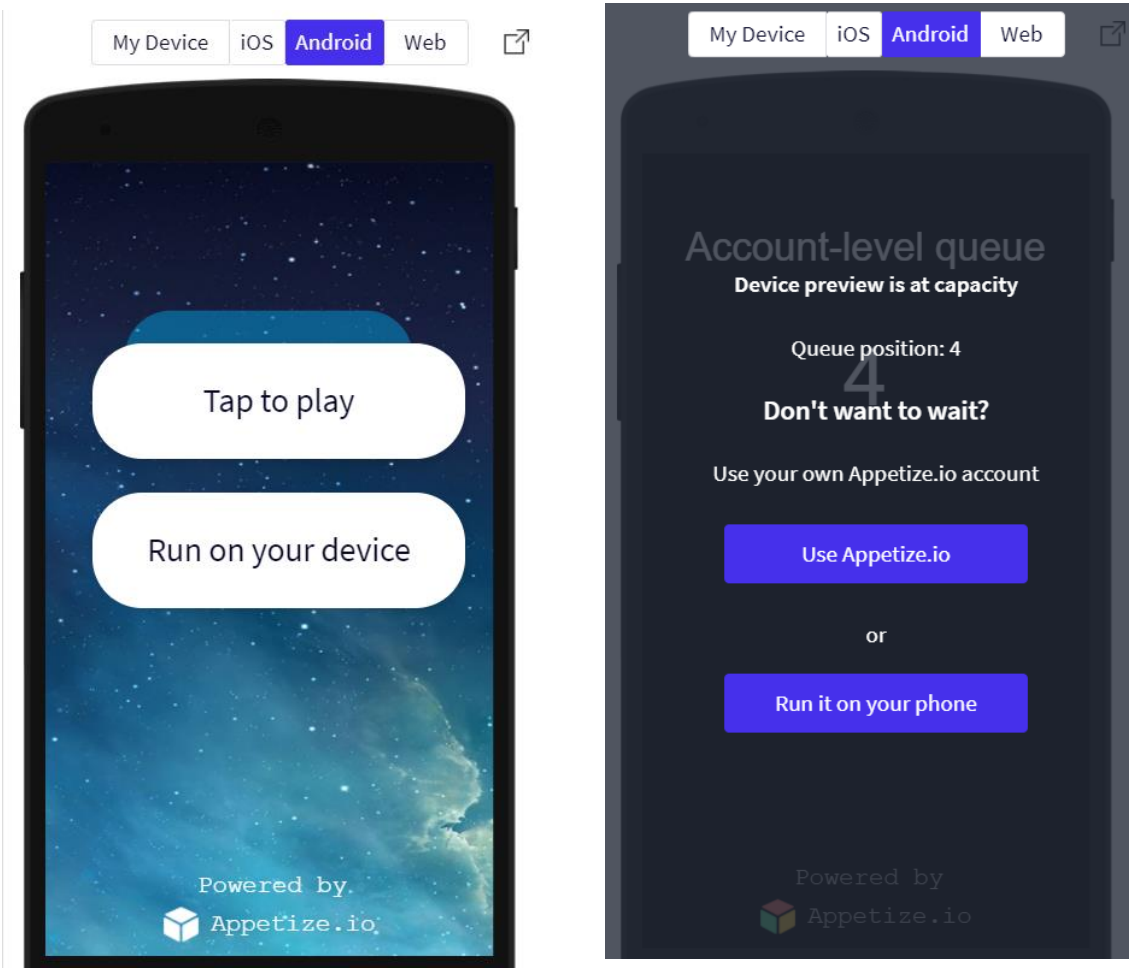
# Visualizando o resultado no Andoid

Inicialmente a aplicação é apresentada no emulador “Web” para alternar para o Android será necessário clicar em “Android”



# Visualizando o resultado no Andoid

Em seguida clique em “Tap to play” e aguarde sua vez para visualizar o aplicativo criado.



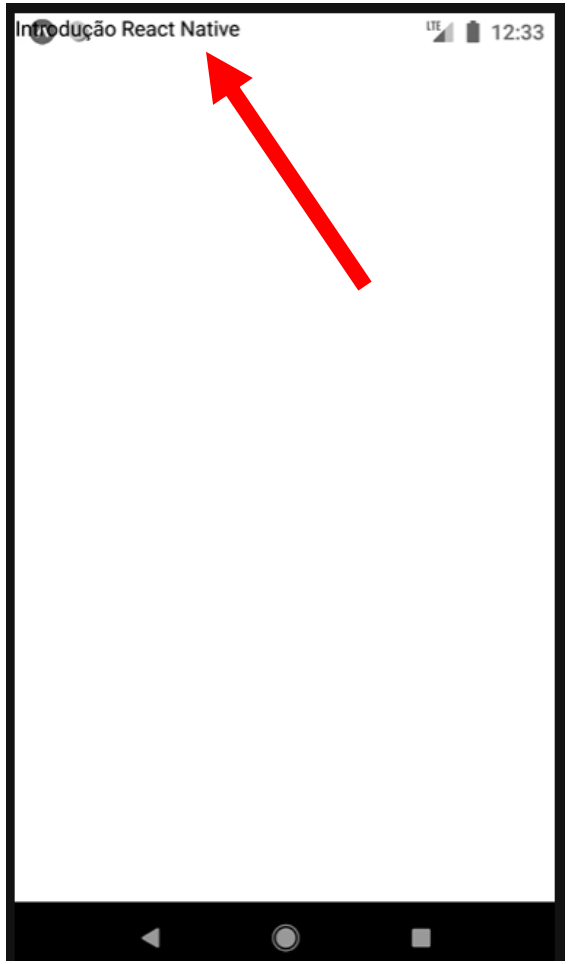
# Chamando componente para exibição

Visualizando nosso aplicativo, veremos que o texto inserido ocupa a barra de status do Android.

```
13   export default function App() {
14     return (
15       <Index />
16     );
17   }
```

# JSX

Para corrigir a sobreposição devemos estilizar o componente como quando utilizamos CSS em uma página WEB, porém em React Native JSX o é a linguagem utilizada para estilizar, esta é bem semelhante ao CSS.



# JSX

Ainda no componente App.js observe que deixamos no import alguns componentes como a View e o Constants, iremos utiliza-los para corrigir a visualização!

Um estilo pode ser aplicado diretamente na chamada de um componente. Neste exemplo utilizamos uma View como um container que vai abrigar o componente Index.

**Obs:** Funções só retornam um componente!

```
1  import * as React from 'react';
2  import { [illegible], View, StyleSheet } from 'react-native';
3  [illegible] Constants from 'expo-constants';
4
5  // fazer o import permite acessar outros componentes
6  import Index from './src/components/Index';
7
8  export default function App() {
9    return (
10     <View style={{flex: 1, paddingTop: Constants.statusBarHeight}}>
11       <Index />
12     </View>
13   );
14 }
```

# JSX

Já temos um objeto estilizado e pronto para ser utilizado, ele chama-se “container” e foi criado para estilizar um “View” para que a mesma não sobreposse a barra de status.

Para corrigir nosso problema basta deixar o “Index” dentro de uma “View” estilizada, conforme apontado na imagem ao lado.

```
13  export default function App() {
14    return (
15      <View style={styles.container}>
16        <Index />
17      </View>
18    );
19  }
20
21  const styles = StyleSheet.create({
```

# JSX

No “App.js” só utilizamos um objeto de estilo que é o chamado “container”, podemos apagar o outro, deixando como no exemplo a direita.

```
21  const styles = StyleSheet.create({
22    container: {
23      flex: 1,
24      justifyContent: 'center',
25      paddingTop: Constants.statusBarHeight,
26      backgroundColor: '#ecf0f1',
27      padding: 8,
28    },
29    paragraph: {
30      margin: 24,
31      fontSize: 18,
32      fontWeight: 'bold',
33      textAlign: 'center',
34    },
35  });
```

```
13  export default function App() {
14    return (
15      <View style={styles.container}>
16        <Index />
17      </View>
18    );
19  }
20
21  const styles = StyleSheet.create({
22    container: {
23      flex: 1,
24      // justifyContent: 'center',
25      paddingTop: Constants.statusBarHeight,
26      backgroundColor: '#ecf0f1',
27      padding: 8,
28    },
29  });
```

# Simplificando App.js

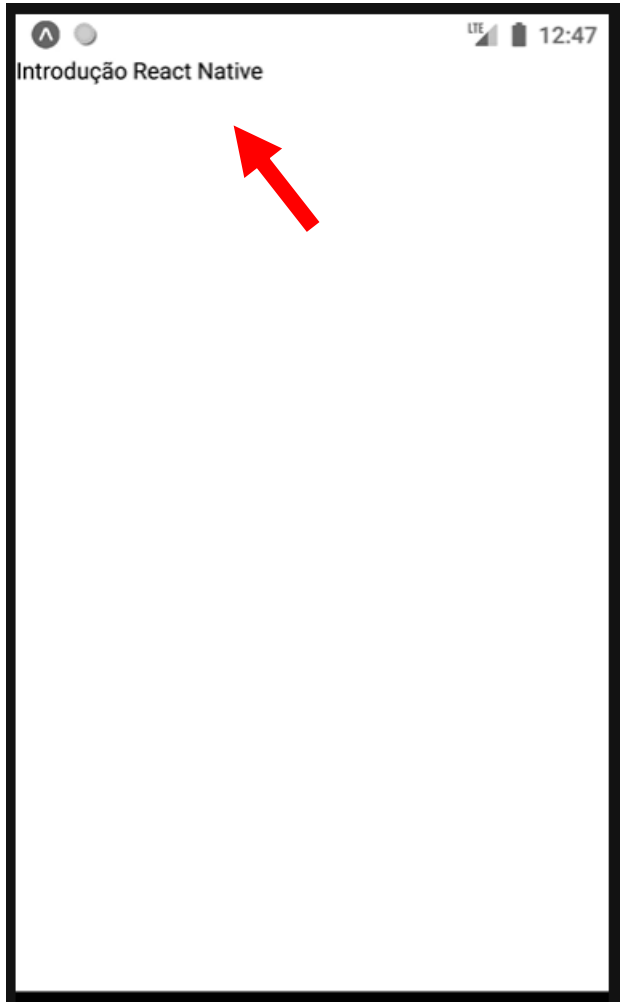
Também não precisamos de algumas coisas que foram importadas, apague aquilo que não faz parte do exemplo conforme apontado abaixo, para que o App.js fique com a o código da imagem a direita.

```
1   import * as React from 'react';
2   import { Text, View, StyleSheet } from 'react-native';
3   import Constants from 'expo-constants';
4
5   // You can import from local files
6   import AssetExample from './components/AssetExample';
7
8   // or any pure javascript modules available in npm
9   import { Card } from 'react-native-paper';
10
11  import Index from './components/Index';
12
13  export default function App() {
```

```
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: '#ecf0f1',
21      padding: 8,
22    },
23  });
```

Comentar esta linha

# O programa deve ficar como na imagem abaixo:



# React Native e o Flexbox

O React Native utiliza o Flexbox para organização visual dos componentes, o que simplifica, a construção de layouts. Basicamente, o parâmetro flex: 1 significa que o container ocupa 100% de altura e largura na tela.

O parâmetro flexDirection: column significa que os elementos seguirão o fluxo baseado em colunas, que é de cima para baixo. A outra opção seria flexDirection: row, onde os elementos são ordenados da esquerda para a direita. Por default, O fluxo padrão é o de colunas.

# JSX

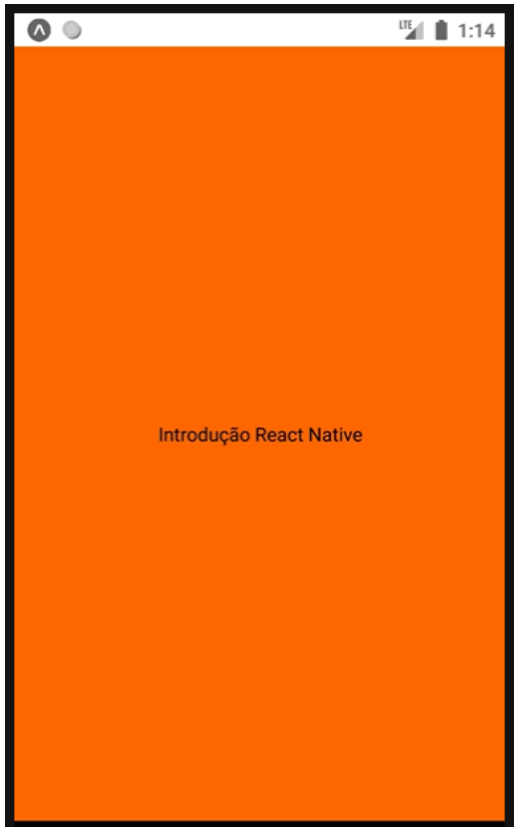
Volte ao arquivo “Index.js” e adicione a propriedade de estilização na View. Começaremos aplicando uma cor, pois assim teremos uma noção do espaço que nosso componente ocupa dentro de App.js.

```
5   // declaração de funções do componente
6   function Index() {
7     return (
8       <View style={{backgroundColor: '#f60'}}>
9         <Text>
10          Introdução React Native
11        </Text>
12      </View>
13    );
14  }
```

# JSX

Adicione ao estilo apontado abaixo uma propriedade por vez, e verifique o que é alterada a cada inserção.

```
5   // declaração de funções do componente
6   function Index() {
7     return (
8       <View style={{backgroundColor: '#f60', flex: 1, alignItems: 'center', justifyContent: 'center'}}>
9         <Text>
10          Introdução React Native
11        </Text>
12      </View>
13    );
14  }
```



# JSX

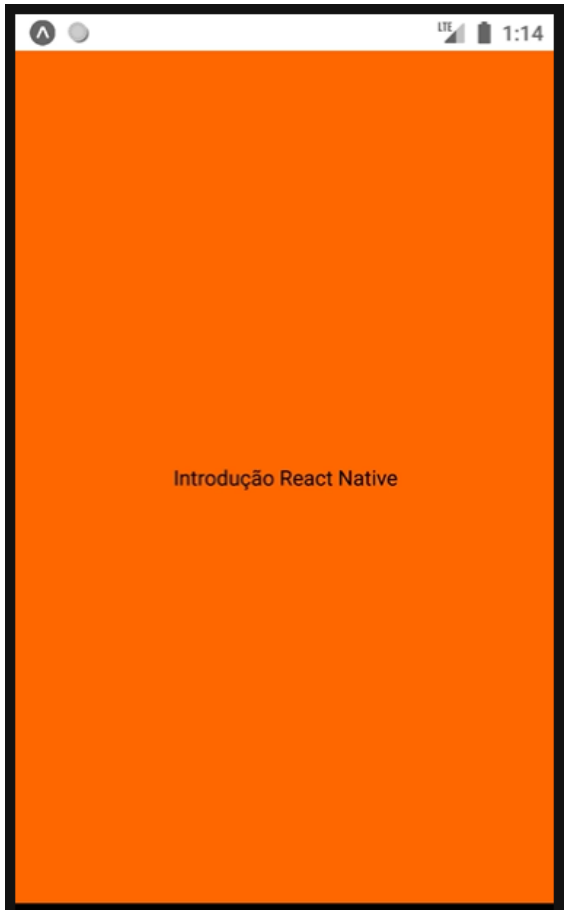
Agora iremos conhecer algumas propriedades de posicionamento:

backgroundColor: define a cor de fundo.

flex: define o quanto da tela a View irá ocupar.

alignItems: define o alinhamento horizontal.

justifyContent: define o alinhamento vertical.




Veja neste site algumas variações de cores que podem ser aplicadas, estas, podem referenciadas em hexadecimal ou string como por exemplo ‘green’:
http://www.erikasarti.com/html/tabela-cores-seguras-web-safe/

# Objeto para estilos

Outra maneira de definir estilos, é criando um objeto que contenha as configurações que desejamos aplicar. O componente que permite armazenar objetos de estilos é o StyleSheet, ele é um dos componentes da biblioteca “react_native”, importe ele no componente Index.js conforme apontado na imagem abaixo.

# Objeto para estilos

Agora vá até a última linha de código e insira as linhas apontadas abaixo, onde iremos instanciar um objeto do tipo StyleSheet em uma constante que será utilizada para referenciar um array de objetos que contém a estilização.

```
16   // apontar a função default
17   export default Index;
18
19   // jsx (estilização)
20   const styles = StyleSheet.create({
21
22   });
```

# Objeto para estilos

Nosso primeiro objeto de estilização vai se chamar “container” e nele iremos definir as mesmas propriedade que inserimos na propriedade “styles” da View anteriormente. Inicialmente o nome do objeto vai ficar sublinhado, indicando que não existem referências a ele no componente. Observe também que existe uma virgula após a ultima propriedade e outra após a definição do objeto, isso ajudar a não esquecermos antes de inserir uma nova propriedade ou objeto.

```
6   function Index() {
7     return (
8       <View style={}>
9         <Text>
10          Introdução React Native
11        </Text>
12      </View>
13    );
14  }
15
16  // apontar a função default
17  export default Index;
18
19  // jsx (estilização)
20  const styles = StyleSheet.create({
21    container: {
22      backgroundColor: '#f60',
23      flex: 1,
24      alignItems: 'center',
25      justifyContent: 'center'
26    },
27  });
```

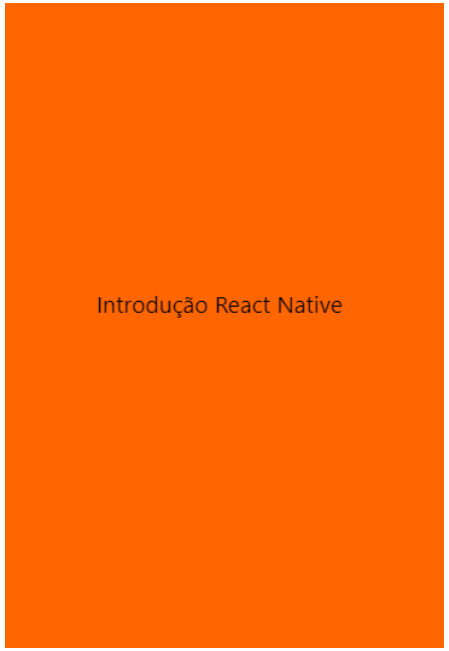
# Objeto para estilos

Voltando a View, altere o código como apontado ao lado.

Utilizando StyleSheet, podemos separar a estilização da estrutura do componente, deixando assim o código mais legível. Para aplicar o estilo criado basta declarar o nome da constantes definida no StyleSheet seguido do nome do objeto de estilização.

O resultado da aplicação vai continuar o mesmo!

```
8   <View style={styles.container}>
9     <Text>
10      Introdução React Native
11    </Text>
12  </View>
```



# Estilizando o texto

Agora iremos criar outro estilo para aplicar no texto em seguida declare o objeto de estilização no Text e crie um objeto chamado “texto”.

```
5   // declaração de funções do componente
6   function Index() {
7     return (
8       <View style={styles.container}>
9         <Text style={styles.texto}>
10          Introdução React Native
11        </Text>
12      </View>
13    );
14  }
```

```
16  // apontar a função default
17  export default Index;
18
19  // jsx (estilização)
20  const styles = StyleSheet.create({
21    container: {
22      backgroundColor: '#f60',
23      flex: 1,
24      alignItems: 'center',
25      justifyContent: 'center'
26    },
27    texto: {
28      color: '#fff',
29      fontSize: 30,
30    },
31  });
```

# Pratique

- Adicione mais componentes do tipo Text e aplique diferentes estilos em cada um deles. Veja no link a seguir os diferente estilos que podem ser aplicados em um Text:

Text - Expo Documentation

- Veja também alguns estilos que podem ser aplicados em Views

View Style Props - Expo Documentation

- Inicie outro projeto e tente trabalhar com mais de uma View, combinada com alguns Texts.

# Referências


- https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Guide/Fun%C3%A7%C3%B5es
- https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Reference/Operators/function
- https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Reference/Functions/Arrow_functions

Documentação React

Getting Started – React (reactjs.org)

Documentação dos componentes:

- API Reference - Expo Documentation