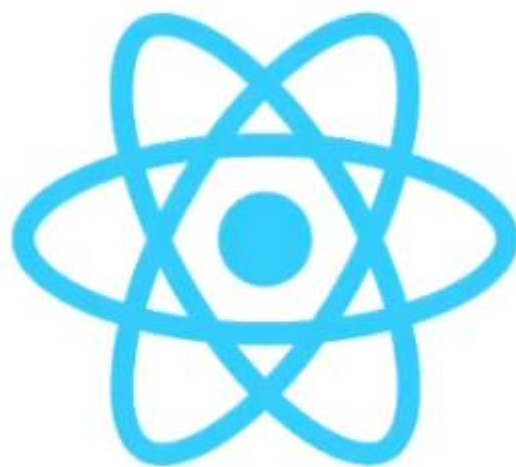


# INSTALANDO E CRIANDO UM PROJETO REACT-NATIVE.

Aula -1

-FERNANDO LUCAS (MASSAN)



**Fundamentos**  
**Desenvolvimento Mobile com React Native**

# O que é React Native?



- # Framework de criação de aplicações nativas mobile
- # Projeto Open Source mantido pelo Facebook, desde 2015
- # Multiplataforma
- # Podemos manipular cada plataforma de forma diferente

# Características React Native

# Fast Refresh que da feedback quase que instantâneo das alterações

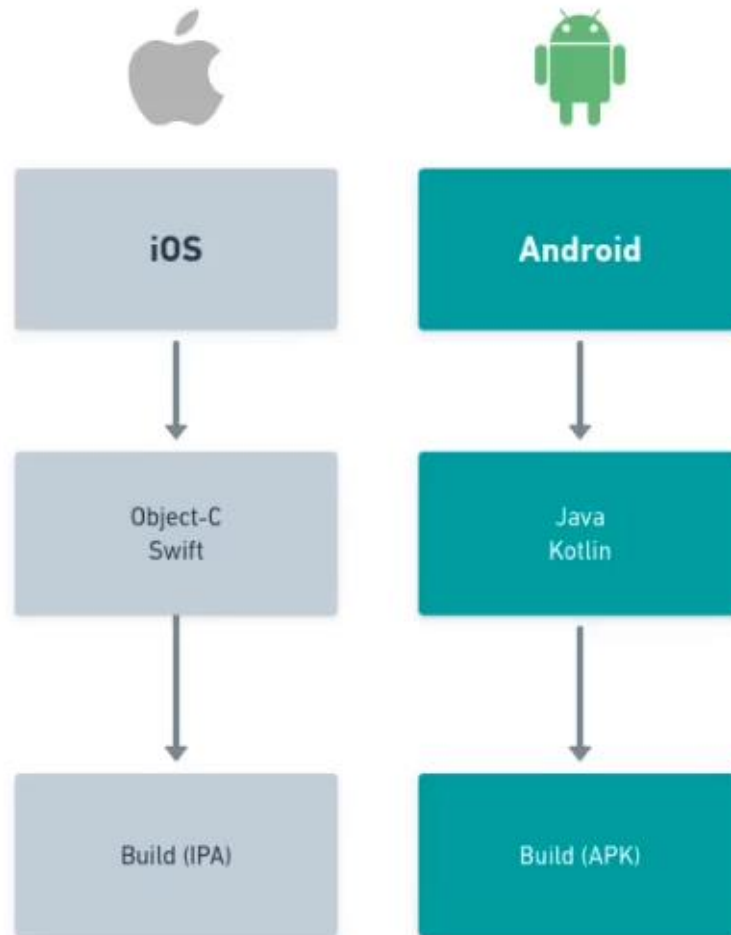
# Uma base de código para as plataformas iOS e Android

# Javascript, uma linguagem muito popular.

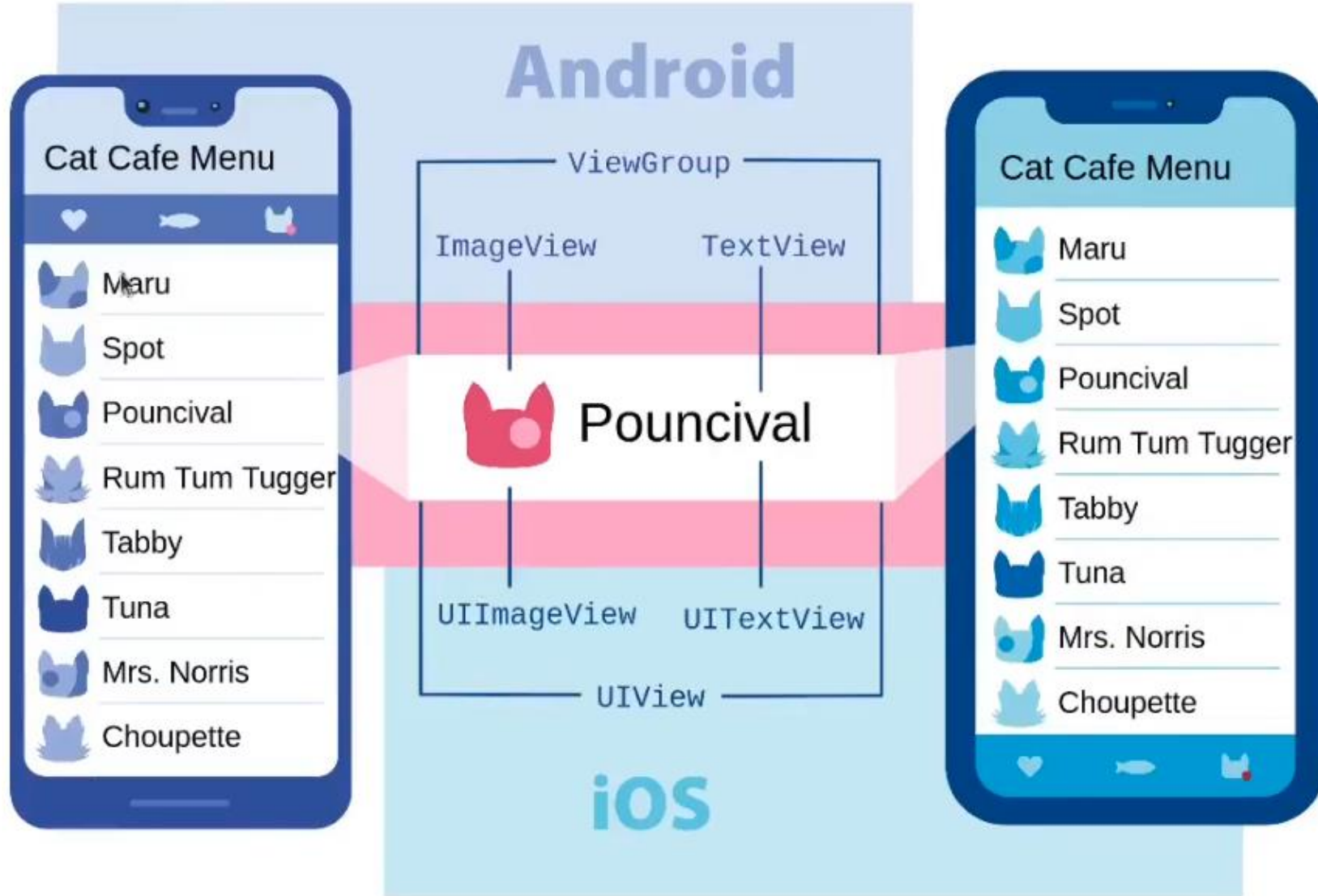
# Comunidade ativa com inúmeras bibliotecas e UI frameworks



## Tradicional



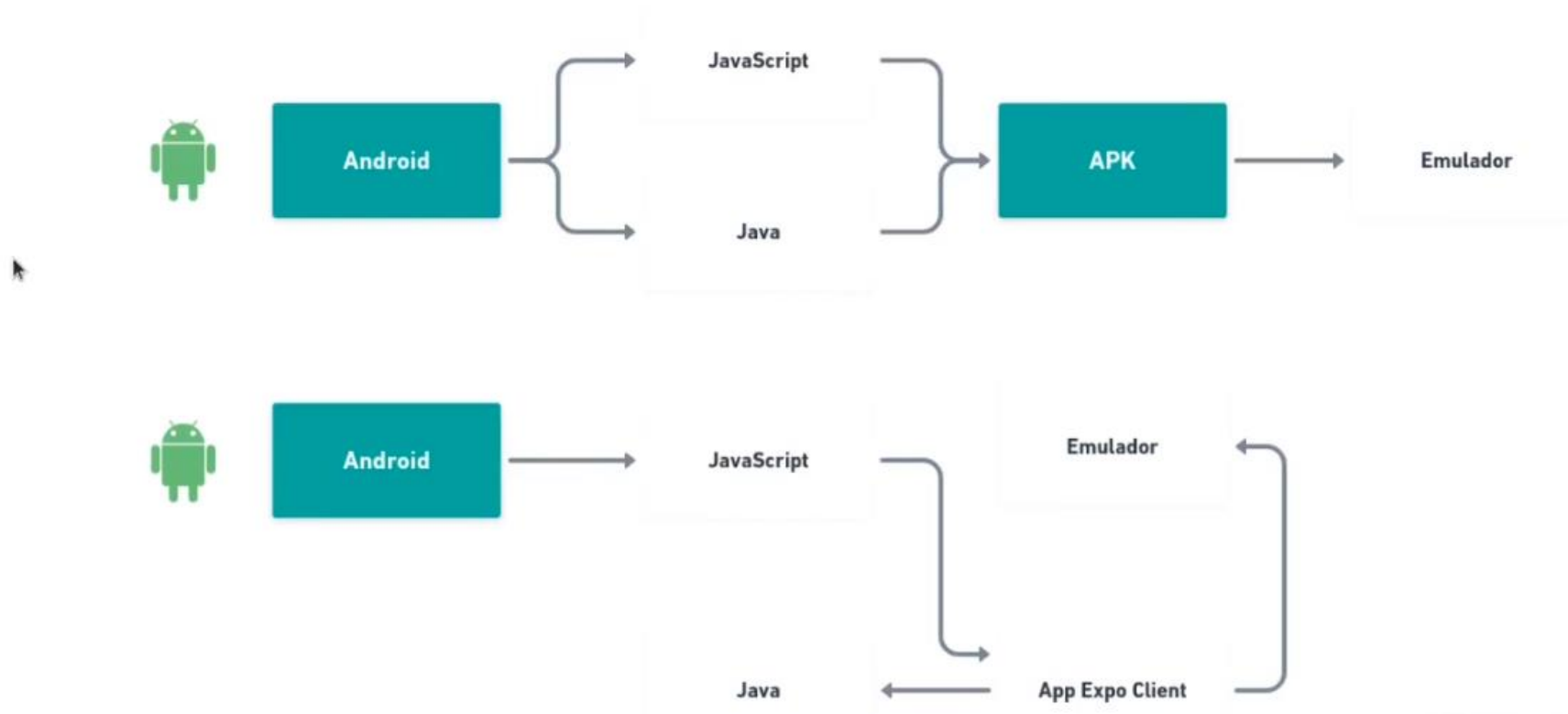




## O que é Expo?

# SDK com um conjunto de funcionalidades prontas para usar (câmera, vídeo, integrações)

# Em minutos você tem seu ambiente de desenvolvimento pronto sem precisar de um emulador;



# ABRA O CMD

Digite o comando

`0 node -v`

Esse comando vai verificar se existe o node instalado na sua máquina, se não tiver instale.

`0 npm install -g expo-cli`

Esse comando vai instalar o expo, uma espécie de framework para o react-native.

`0 expo init "nomedoapp" (npx create-expo-app nomedoseuapp)`

Esse comando irá baixar e criar sua primeira aplicação em react-native.



# SELECCIONE O SEGUNDO VALOR

```
C:\Users\Massan>expo init leanges
WARNING: The legacy expo-cli does not support Node +17. Migrate to the new local Expo CLI: https://blog.expo.dev/the-new-expo-cli-f4250d8e3421.
```

The global expo-cli package has been deprecated.

The new Expo CLI is now bundled in your project in the expo package.  
Learn more: <https://blog.expo.dev/the-new-expo-cli-f4250d8e3421>.

To use the local CLI instead (recommended in SDK 46 and higher), run:  
> npx expo <command>

Migrate to using:

```
> npx create-expo-app --template
```

```
? Choose a template: » - Use arrow-keys. Return to submit.
```

```
----- Managed workflow -----
```

```
> blank a minimal app as clean as an empty canvas
```

```
blank (TypeScript) same as blank but with TypeScript configuration
```

```
tabs (TypeScript) several example screens and tabs using react-navigation and TypeScript
```

```
----- Bare workflow -----
```

```
minimal bare and minimal, just the essentials to get you started
```

Prompt de Comando

```
C:\Users\Maçã-HOME>node -v
v18.12.1

C:\Users\Maçã-HOME>npx create-expo-app --template
✓ Choose a template: » Blank
✓ What is your app named? ... teste
✓ Downloaded and extracted project files.
> npm install
npm WARN deprecated @npmcli/move-file@1.1.2: This functionality has been moved to @npmcli/fs
npm WARN deprecated uuid@3.4.0: Please upgrade to version 7 or higher. Older versions may use Math.random() in certain circumstances, which is known to be problematic. See https://v8.dev/blog/math-random for details.
npm WARN deprecated uglify-es@3.3.9: support for ECMAScript is superseded by `uglify-js` as of v3.13.0

added 1221 packages, and audited 1222 packages in 54s

69 packages are looking for funding
  run `npm fund` for details

5 moderate severity vulnerabilities

To address all issues (including breaking changes), run:
  npm audit fix --force

Run `npm audit` for details.

🎉 Your project is ready!

To run your project, navigate to the directory and run one of the following npm commands.

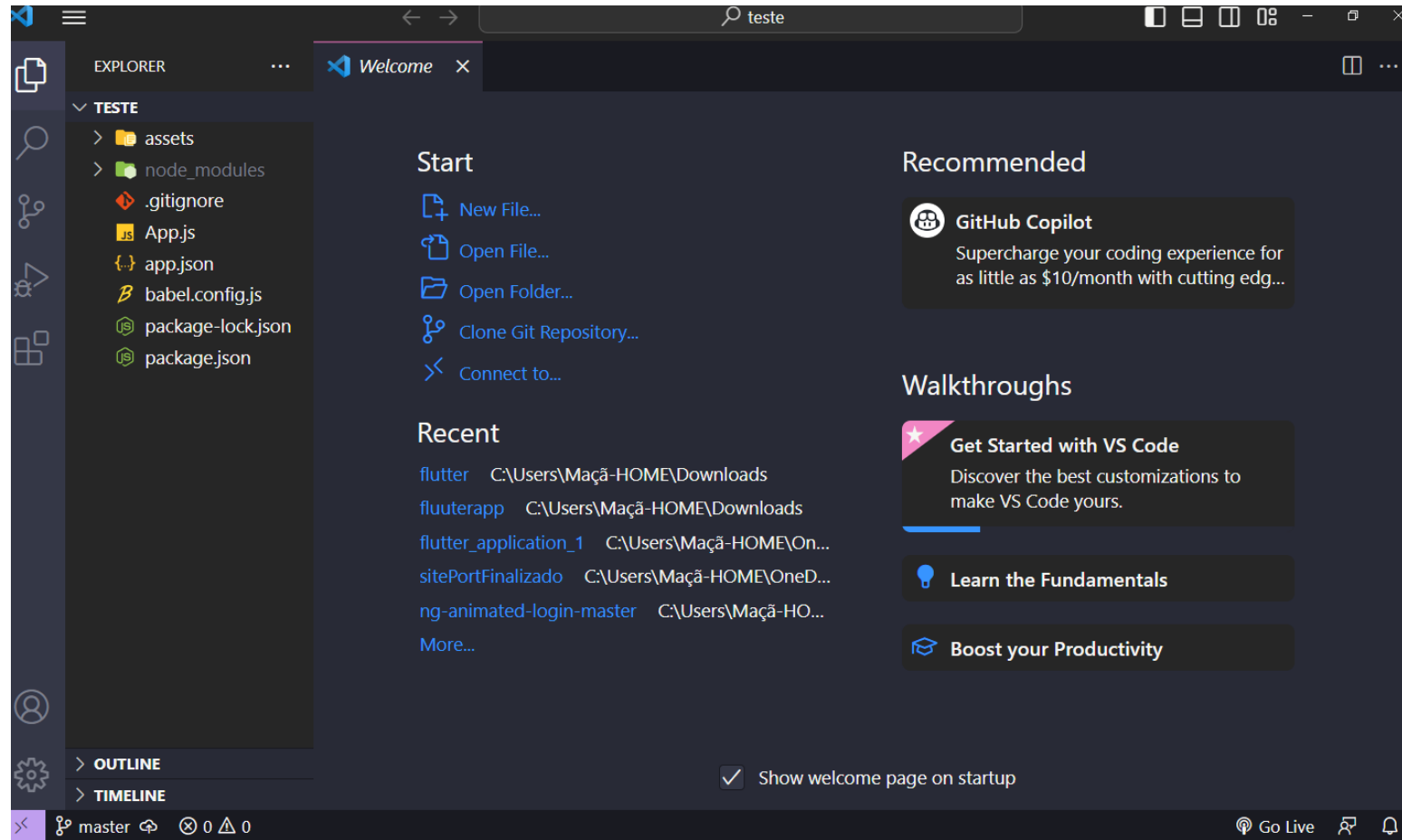
- cd teste
- npm run android
- npm run ios # you need to use macOS to build the iOS project - use the Expo app if you need to do iOS development without a Mac
- npm run web

A new version of `create-expo-app` is available
You can update by running: npm install -g create-expo-app

npm notice
npm notice New major version of npm available! 8.19.2 -> 10.0.0
npm notice Changelog: https://github.com/npm/cli/releases/tag/v10.0.0
npm notice Run npm install -g npm@10.0.0 to update!
npm notice

C:\Users\Maçã-HOME>
```

# AGORA ABRA O PROJETO CRIADO NO VSCODE.

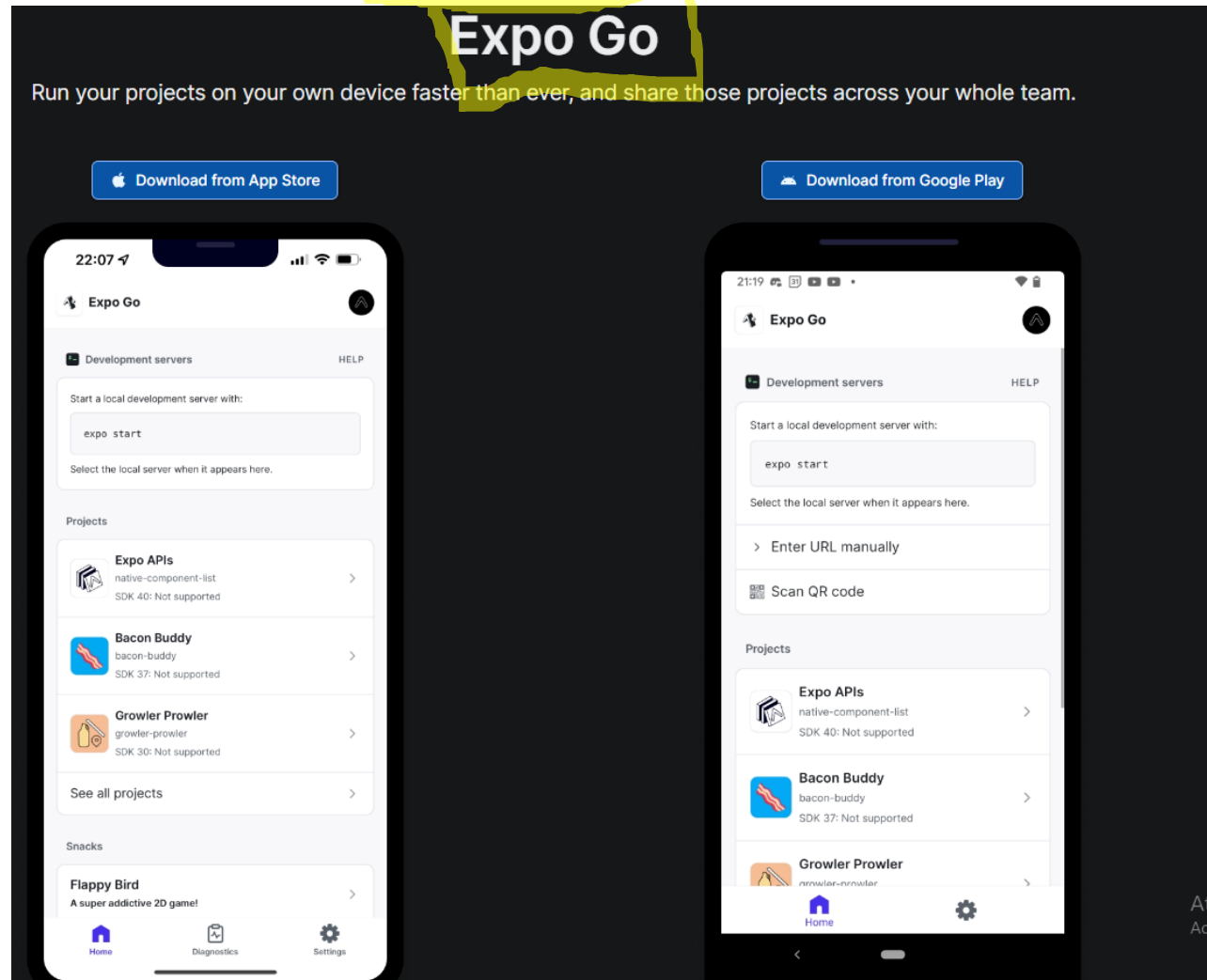


# EMULADOR

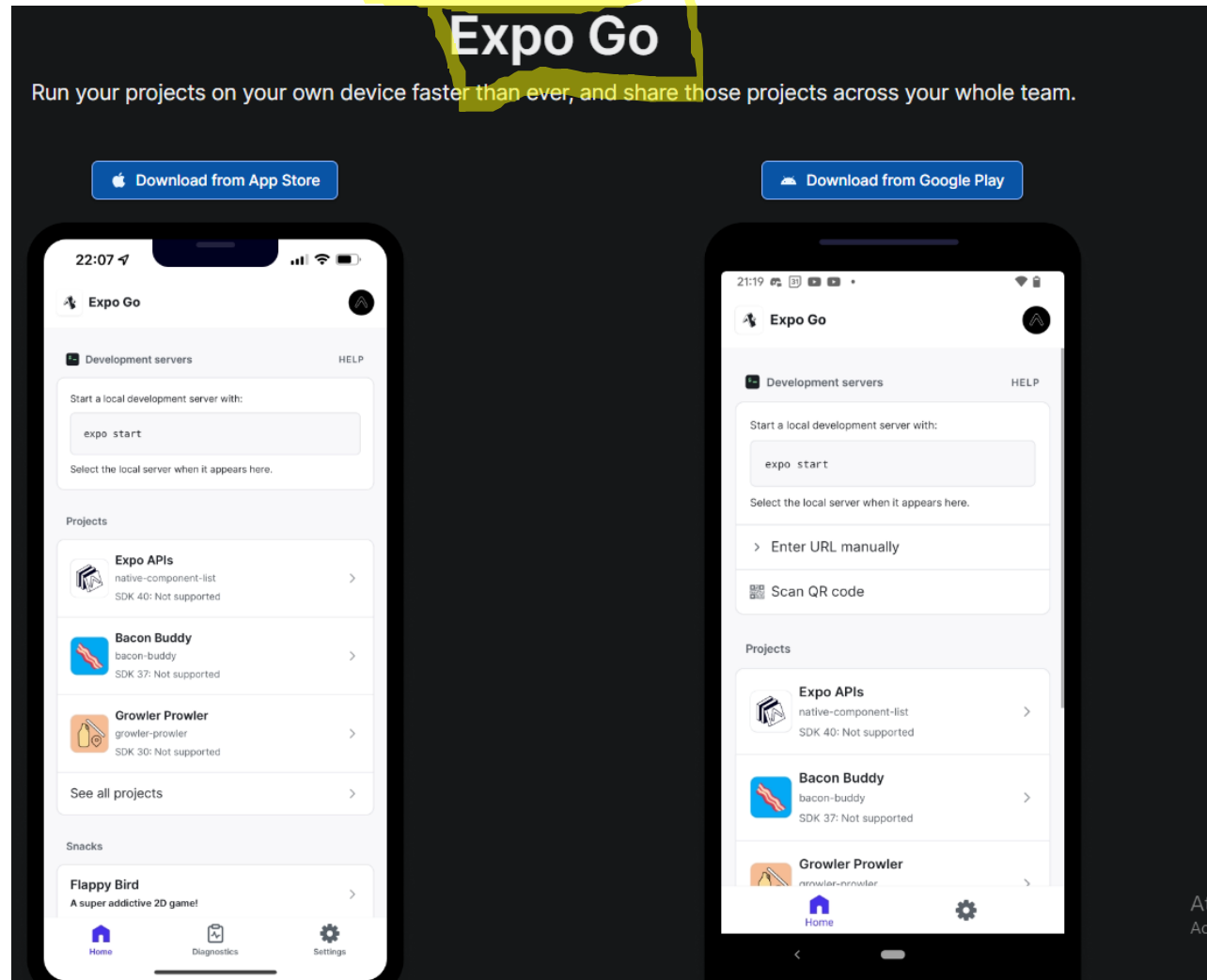
Nós vamos utilizar nosso próprio celular para verificar as alterações que vamos fazer no projeto em tempo real.

- 1 - Abra o terminal do vs code no cmd
- 2 - execute o comando - `npx expo start`
- 3 - scaneie o qrcode.

AGORA ENTRE NA SOTRE DO SEU CELULAR E BAIXE O APLICATIVO.  
EXPO.



AGORA ENTRE NA SOTRE DO SEU CELULAR E BAIXE O APLICATIVO.  
EXPO.

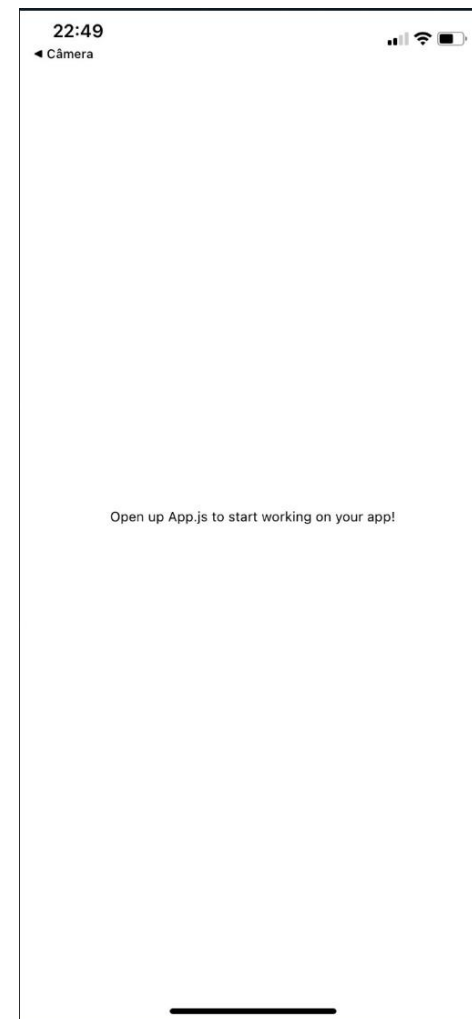




# VEJA A MENSAGEM NO SEU CELULAR

Open up app.js start on your app.

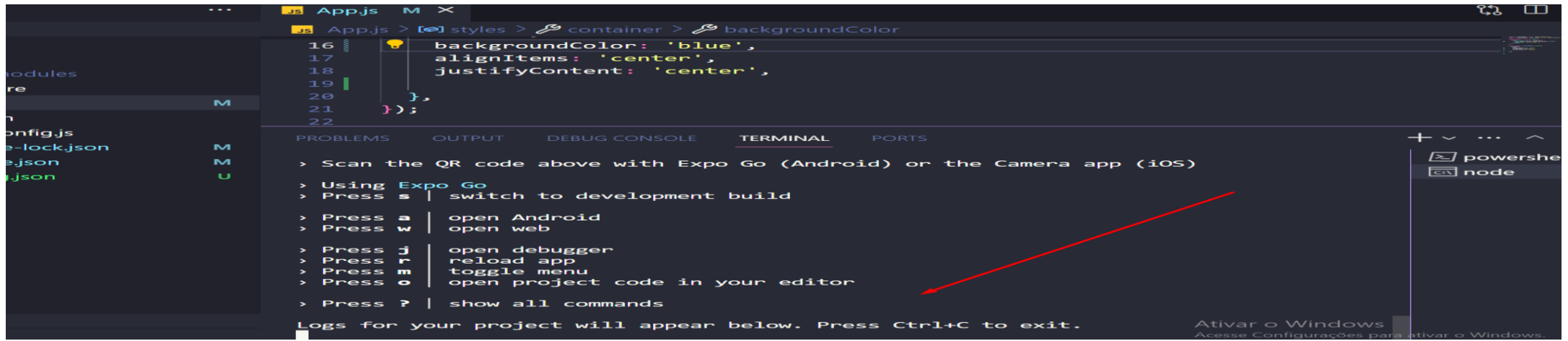
Vamos altera-la utilizando o fast refresh que é  
uma propriedade que atualiza nosso app  
instantaneamente.



# CASO VOCÊ NÃO CONSIGA CONECTAR O CELULAR

Caso você não consiga conectar o celular, você também pode verificar as alterações direto no navegador do computador.

Primeiro aperta no terminal do vscode as teclas CTRL+C para parar o servidor.



The screenshot shows the VS Code interface. The editor displays a file named `App.js` with the following code:

```
16 backgroundColor: 'blue',  
17 alignItems: 'center',  
18 justifyContent: 'center',  
19 },  
20 },  
21 },  
22 };
```

The terminal at the bottom shows the following commands and their outputs:

```
> Scan the QR code above with Expo Go (Android) or the Camera app (iOS)  
> Using Expo Go  
> Press s | switch to development build  
> Press a | open Android  
> Press w | open web  
  
> Press j | open debugger  
> Press r | reload app  
> Press m | toggle menu  
> Press o | open project code in your editor  
  
> Press ? | show all commands  
Logs for your project will appear below. Press Ctrl+C to exit.
```

A red arrow points to the terminal output. The right sidebar shows the file explorer with files like `App.js`, `AppLock.js`, `AppLock.json`, and `AppLock.json`.

# CASO VOCÊ NÃO CONSIGA CONECTAR O CELULAR

Com o servidor parado agora você pode escrever comando novamente.

Vamos instalar a dependência necessária para executar o arquivo no navegador.

Execute o seguinte comando.

```
npx expo install react-native-web@~0.19.6 react-dom@18.2.0
```

Agora espera a instalação acontecer.

```
> Installing 2 SDK 49.0.0 compatible native modules using npm
> npm install
up to date, audited 1237 packages in 3s
69 packages are looking for funding
  run `npm fund` for details
5 moderate severity vulnerabilities
To address all issues (including breaking changes), run:
  npm audit fix --force
Run `npm audit` for details.
E:\it2cev\3 - React-Native-Expo\P-OL>
```

Ativar o Windows  
Acesse Configurações pa

# CASO VOCÊ NÃO CONSIGA CONECTAR O CELULAR

Execute também o seguinte comando.

```
npx expo install @expo/webpack-config@~19.0.0
```

Agora espera a instalação acontecer.

```
> Installing 2 SDK 49.0.0 compatible native modules using npm
> npm install
up to date, audited 1237 packages in 3s
69 packages are looking for funding
  run `npm fund` for details
5 moderate severity vulnerabilities
To address all issues (including breaking changes), run:
  npm audit fix --force
Run `npm audit` for details.
E:\it2cev\3 - React-Native-Expo\P-OL>
```

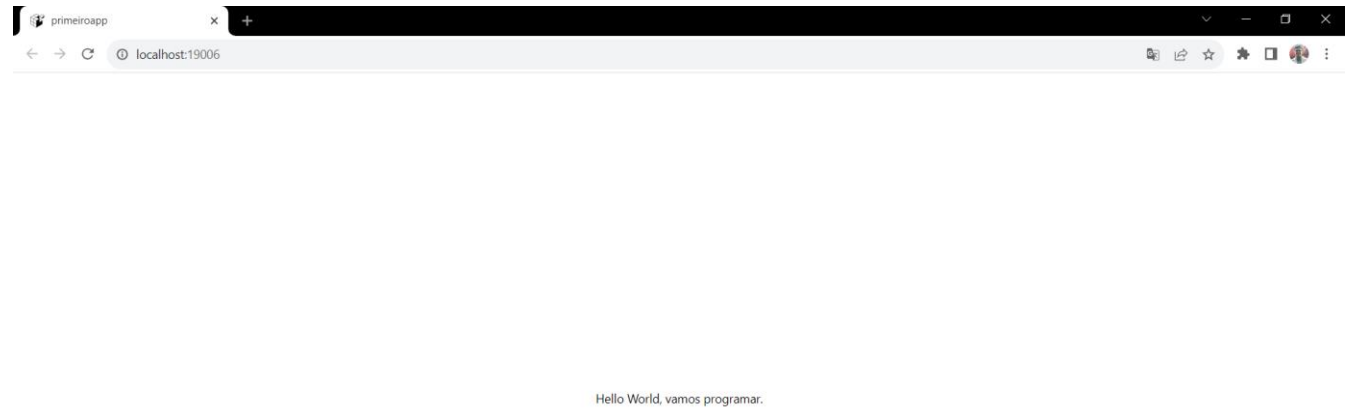
Ativar o Windows  
Acesse Configurações pa

# CASO VOCÊ NÃO CONSIGA CONECTAR O CELULAR

Depois disso execute o servidor novamente.

`Npx expo start`

Agora vai aparecer o QR code novamente, só que dessa vez você vai apertar a tecla `W` no terminal onde você digitou os comandos e assim ele vai abrir no seu navegador com seu aplicativo.



```

App.tsx gameplay
App.tsx > App
1 import { StatusBar } from 'expo-status-bar';
2 import React from 'react';
3 import { StyleSheet, Text, View } from 'react-native';
4
5 export default function App() {
6   return (
7     <View style={styles.container}>
8       <Text>Hello</Text>
9       <StatusBar style="dark" />
10    </View>
11  );
12 }
13
14 const styles = StyleSheet.create({
15   container: {
16     flex: 1,
17     backgroundColor: 'white',
18     alignItems: 'center',
19     justifyContent: 'center',
20   },
21 });
22

```

## VEJA A MENSAGEM NO SEU CELULAR

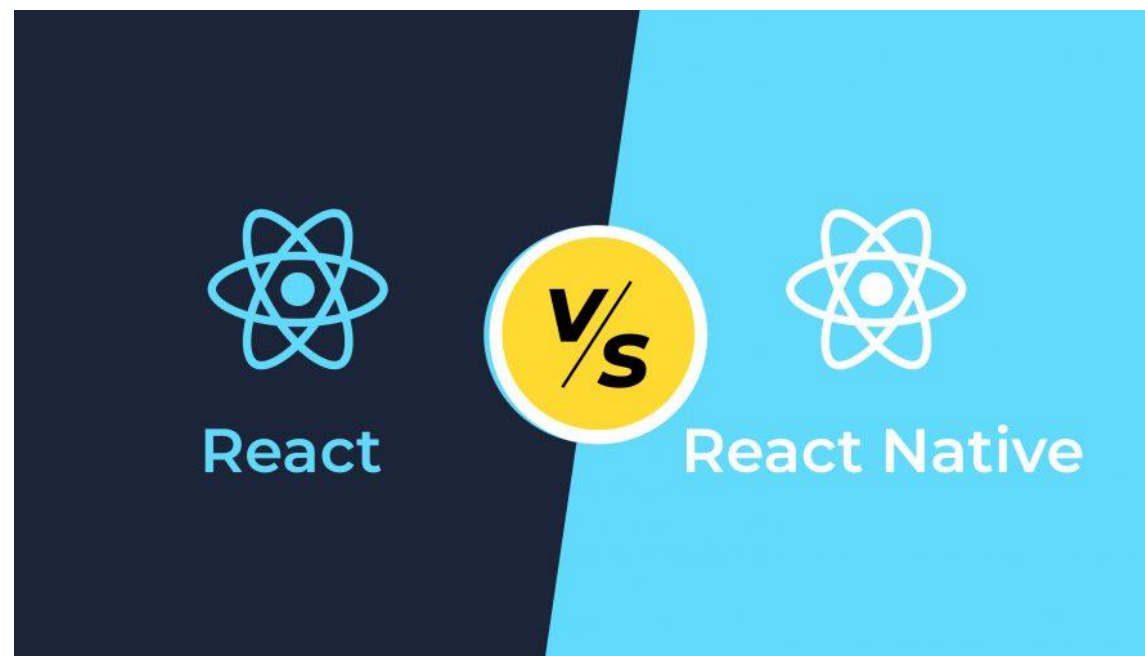
No arquivo app.tsx que é o arquivo que controla nosso aplicativo, vamos procurar o componente `<Text> </Text>` que renderiza nosso texto na tela, depois vamos alterá-lo e pressionar CTRL+S, para salvar nossas alterações.

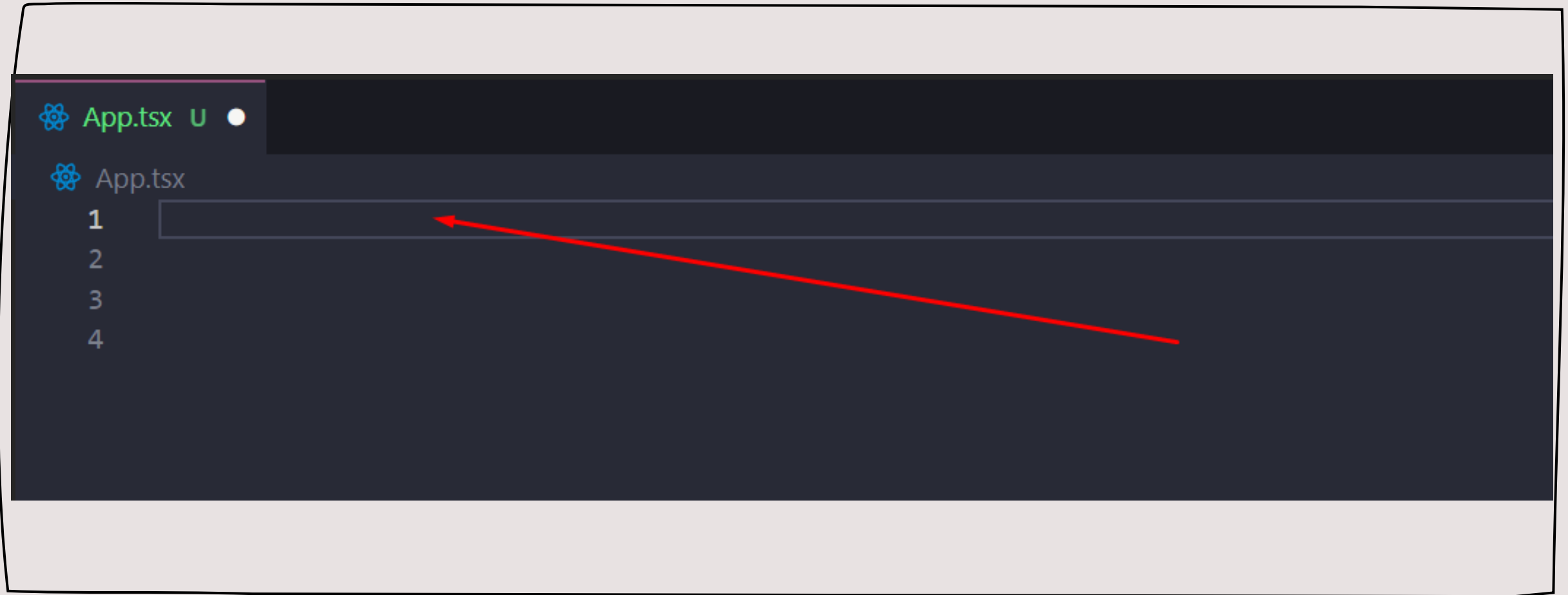
(Agora verifique a tela do seu celular. 😊)



# USANDO E UTILIZANDO A BIBLIOTECA DO REACT

Estamos utilizando o React-native para criar nosso app, essa framework consegue utilizar bibliotecas de outra linguagem que é o React.





TA AGORA VAMOS CRIAR NOSSO ARQUIVO DO 0

1 - Apague todo arquivo app.tsx

# AGORA VAMOS SEGUIR.

Vamos utilizar o comando

Ele vai nos permitir utilizar a biblioteca do React no nosso projeto.

```
import React from 'react';
```

Toda interface que for adicionada no nosso app tem que retornar algum componente.

```
export default function App() {  
  return(  
  );  
}
```

note que a função que renderiza (faz aparecer o componente na nossa tela ) está literalmente pedindo um componente para ser dada como correta.

# ADICIONANDO O COMPONENTE

Primeiro vamos trazer do React nossos componentes.

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

Agora que já importamos os componentes vamos poder utilizá-los dentro da nossa função.

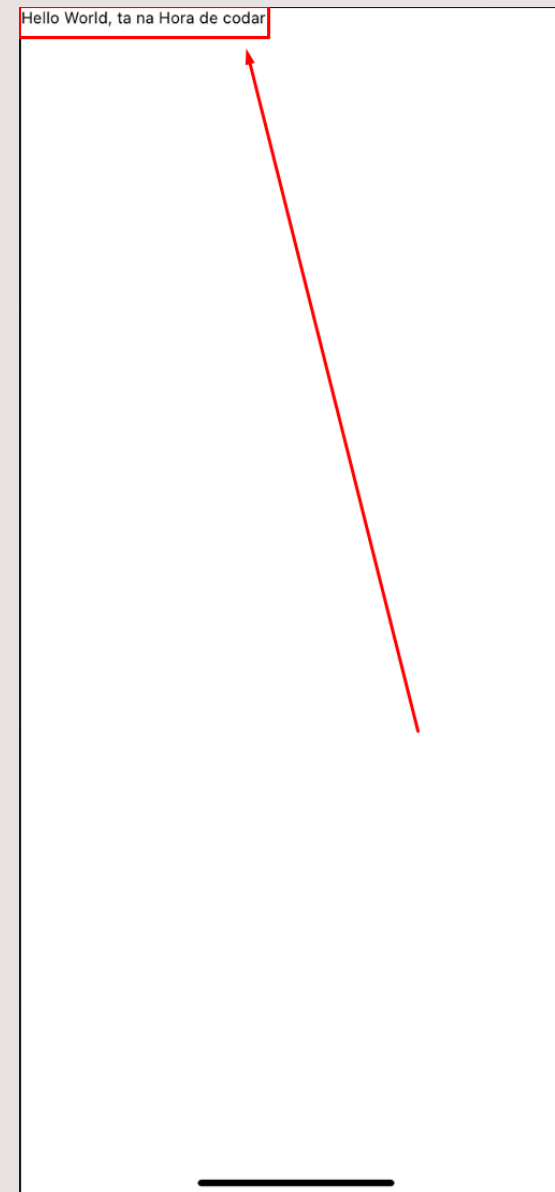
```
export default function App() {  
  return(  
    <View>  
      <Text></Text>  
    </View>  
  )  
}
```

Agora complete com a seguinte frase "Hello World, ta na Hora de codar".

Onde você acha que a frase deve ficar?

# CADÊ MEU TEXTO ?



Bom kkkkkk você deve ter percebido que o seu texto está no canto superior do seu celular.  
(Em alguns modelos quase não será visível)



# AGORA VAMOS ORGANIZAR AS COISAS.

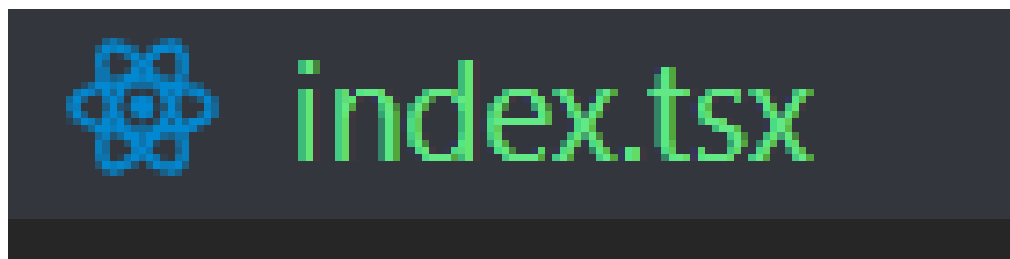
Primeiro vamos criar algumas pastas para separar os arquivos principais da instalação com os arquivos que vamos adicionar.

- 1- Criar um pasta "src"
- 2- Dentro dela vamos criar uma pasta "screens"
- 3- Agora dentro da pasta "screens" vamos criar dois arquivos.

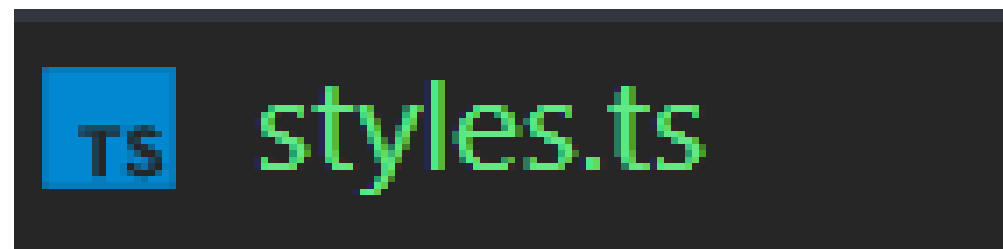
 `index.tsx` `styles.ts`



ESSES DOIS ARQUIVOS VÃO FUNCIONAR DA SEGUINTE FORMA.



Esse será utilizado para guardar os componentes da tela atual que o app estiver.



Esse será utilizado para guarda os estilos da tela atual da nosso aplicação.  
(Lembrando muito o css).

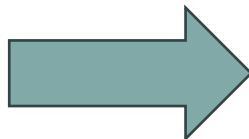
# PRONTO- AGORA VAMOS IMPORTAR AS FUNÇÕES

Vamos copiar todos os códigos iniciais que estavam no arquivo `app.tsx` e vamos colar eles dentro do arquivo `index.tsx`.



# ALTERAÇÕES NECESSÁRIAS

```
export default function App() {  
  return(  
    <View>  
      <Text>Hello World, ta na Hora de codar</Text>  
    </View>  
  )  
}
```



```
export function SingnIn() {  
  return(  
    <View>  
      <Text>Hello World, ta na Hora de codar</Text>  
    </View>  
  );  
}
```


# VOLTAMOS AGORA PARA DENTRO DO NOSSO ARQUIVO

 App.tsx

Vamos agora chamar o arquivo  index.tsx ou seja a função que acabamos de copiar pra lá.

```
import { SingnIn } from '../src/screens/singin';
```

Dessa forma iremos acessar a função que criamos dentro do arquivo  index.tsx

Agora ainda dentro do arquivo  App.tsx vamos alterar nossa função principal para renderizar a função que está sendo importada pelo comando acima.

```
export default function App() {  
  return(  
    <SingnIn />  
  );  
}
```

# AGORA VAMOS ESTILIZAR NOSSO ARQUIVO.

Para isso vamos acessar o arquivo `styles.ts`

1- Vamos importar a biblioteca StyleSheet do react-native.

```
import { StyleSheet } from "react-native";
```

2- Agora vamos criar o objeto que vai guardar nossos estilos.

```
export const styles = StyleSheet.create({  
  container:{  
    flex:1,  
    backgroundColor:'red'  
  }  
});
```

# PEQUENA EXPLICAÇÃO

O que estamos fazendo é criando um arquivo de estilo e importando ele para dentro da tela da nossa aplicação.

Você já fez essa lógica quando estudamos HTML e CSS no semestre passado.

Você criou um arquivo chamou styles.css e depois referenciou ele dentro do index HTML.



# DENTRO DO ARQUIVO index.tsx

1 - Primeiro vamos testar os "stylos" sem referenciar o arquivo.

```
<View style={{ flex: 1, backgroundColor: 'red', alignItems: 'center', justifyContent: 'center' }}>
```

2 - Dessa forma vamos perceber que o código ficou enorme e bagunçado, vamos passar o objeto.

```
import { styles } from './styles'
```

```
<View style={styles.container}>
```

3 - pequenos ajustes.

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

# AGORA VAMOS FALAR SOBRE ESTADO

No React o estado pode ser utilizado para armazenar valores! E nossa interface pode reagir a essa mudança.

No nosso arquivo  `index.tsx` `SignIn` `U` vamos importar outro componente do React `"TextInput"`.

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

Agora dentro da nossa `<view>` vamos passar nosso novo componente `<TextInput/>..`

```
  <TextInput />  
</View>
```

# AGORA VAMOS CRIAR UMA ESTILIZAÇÃO PARA NOSSO NOVO COMPONENTE.

Acesse o arquivo `styles.ts` e nele vamos editar os styles da nossa caixa de texto.

```
export const styles = StyleSheet.create({
  container: {
    flex: 1,
    justifyContent: 'center',
    alignItems: 'center'
  },
  input: {
    height: 50,
    width: 200,
    borderBottomWidth: 2
  }
});
```

# REFERENCIADO OS NOVOS STYLOS

Agora voltamos até o arquivo `index.tsx` e vamos chamar os estilos da nossa nova caixa de texto igual fizemos com os estilo da nossa `<view>`.

```
<TextInput style={styles.input} />
```

Agora temos uma caixa de texto, um componente presente em diversas aplicações do nosso dia a dia.

# AGORA VAMOS TRABALHAR A ALTERAÇÃO DOS ESTADOS.

## Analogia

E o que consiste essas alterações de estado ?

Vamos pensar da seguinte forma, vamos imaginar que temos um programa que controla a lâmpada de uma sala, essa lâmpada possui dois estados Ligada e desligada.

estado1: ligada = Sala iluminada

estado2: desligada = Sala escura

Quando alteramos o estado da nossa lâmpada, a nossa sala reagi e muda a sua forma.

# AGORA VAMOS TRABALHAR A ALTERAÇÃO DOS ESTADOS.

Traduzindo...

Com alterações de estado o nosso aplicativo consegue reagir em tempo real a interação do usuário com a plataforma.

Exemplo:


Quando o usuário tenta criar uma senha e ela possui especificações no nível de segurança como:

Letras maiúsculas., minúsculas, 12 caracteres mínimos e etc.

A medida que o usuário digita, a plataforma já informa pra ele se essa senha será aceita.

# A ALTERAÇÃO DOS ESTADOS.

Agora no nosso arquivo `index.tsx` vamos criar um objeto que vai ler o que o usuário digitar e mostrar em tela uma alteração de estado de acordo com o que foi digitado.

```
export function SignIn () {  
  return(  
    <View style={styles.container}>  
      <Text>  
        a vida é como voce encara as situações  
      </Text>  
      <TextInput style={styles.input}/>  
  
      <Text>  
        Você digitou:   
      </Text>  
    </View>  
  )  
}
```

Vamos criar um componente `<Text>` pra identificar onde irá aparecer o conteúdo que será alterado.

# A ALTERAÇÃO DOS ESTADOS.

Agora vamos Importar componente que irá nos ajudar a verificar essas alterações.

```
import React, { useState } from 'react';
```

Agora dentro da na função singin() vamos criar um objeto para ser utilizado como controlador das alterações desses estados utilizando o componente que acabamos de importar o `useState`

```
export function SignIn () {  
  ⚡ const [text, setText] = useState("");  
  
  return(  
    <View style= {styles.container}>  
      <Text>  
        a vida é como voce encara as situações  
      </Text>  
      <TextInput style={styles.input}/>  
  
      <Text>  
        Você digitou:  
      </Text>  
    </View>  
  )  
}
```

`const` = Declaração do objeto.


`text` = Nome do estado.

`setText` = Nome da função que atualiza o estado.



# A ALTERAÇÃO DOS ESTADOS.

```
return(  
  <View style= {styles.container}>  
    <Text>  
      a vida é como voce encara as situações  
    </Text>  
    <TextInput style={styles.input}/>  
  
    <Text>  
      Você digitou: {text}  
    </Text>  
  </View>
```




Agora chamamos nosso objeto dentro do nosso novo <text/>.

E no nosso estado alteramos o seu valor inicial para.

```
const [text, setText] = useState ("Você não digitou nada ainda");
```

# A ALTERAÇÃO DOS ESTADOS.

```
return(  
  <View style= {styles.container}>  
    <Text>  
      a vida é como voce encara as situações  
    </Text>  
    <TextInput style={styles.input}/>  
  
    <Text>  
      Você digitou: {text}  
    </Text>  
  </View>
```



Agora chamamos nosso objeto dentro do nosso novo <text/>.

E no nosso estado alteramos o seu valor inicial para.

```
const [text, setText] = useState ("Você não digitou nada ainda");
```

# A ALTERAÇÃO DOS ESTADOS.

Agora na nossa caixa de texto `<textInput/>` Vamos utilizar a função `onChangeText` ela vai verificar quando houver alguma alteração na caixa de texto.

Você também pode usar ela junto com o `console.log()` para verificar as alterações sem tempo real no console.

```
onChangeText={console.log}
```

Dentro do código da caixa de texto ela deve ficar assim.

```
<TextInput style={styles.input}  
  onChangeText={setText}  
/>
```

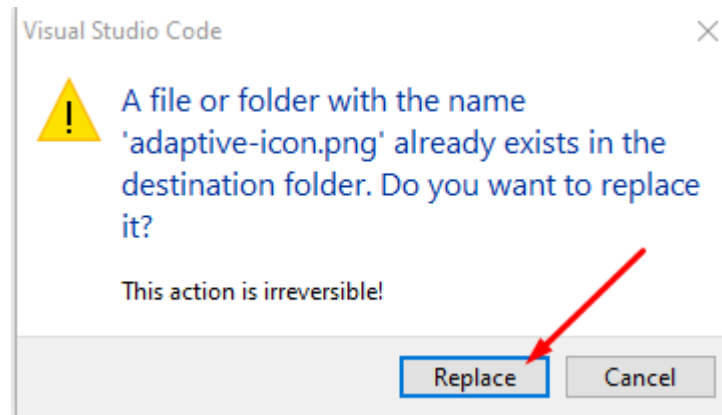
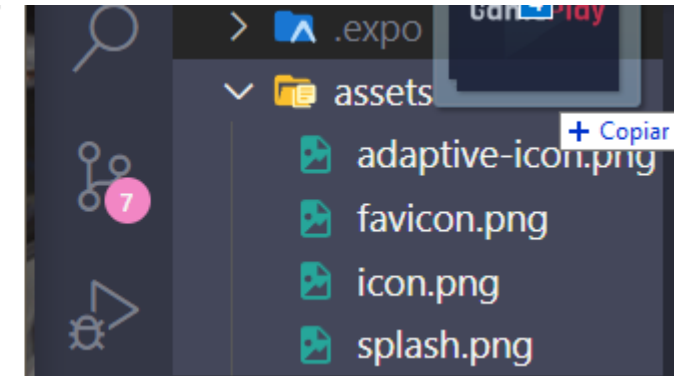
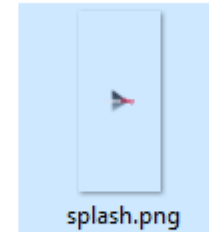
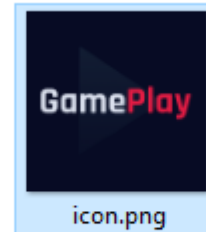
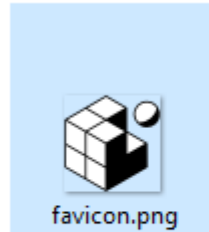
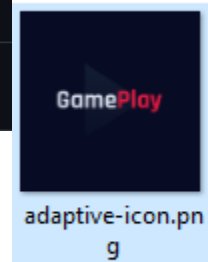
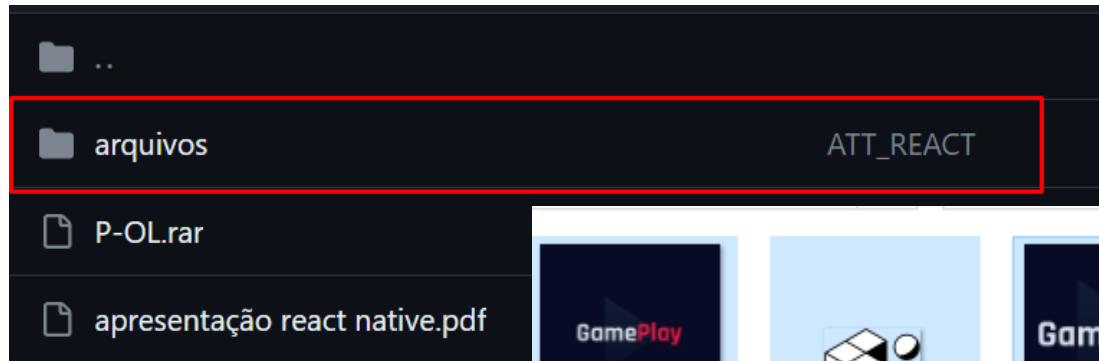
# SPLASH SCREEN

Para falar de componentes vamos fazer a tela de Sigin. . (próximo slide)

- Entre no github e baixe a pasta assets. . (próximo slide)
- Substitua todos os itens da pasta assets da sua aplicação. . (próximo slide)
- Depois entre na pasta SRC nos arquivos que você baixou e copie tudo que estiver na pasta assets, para pasta SRC assets da sua aplicação. (próximo slide)
- Se você Reiniciar o emulador conseguirá verificar as alterações nas artes.



# SPLASH SCREEN

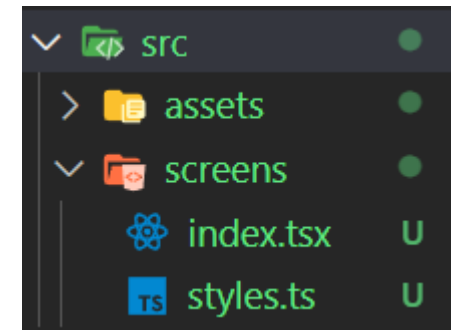
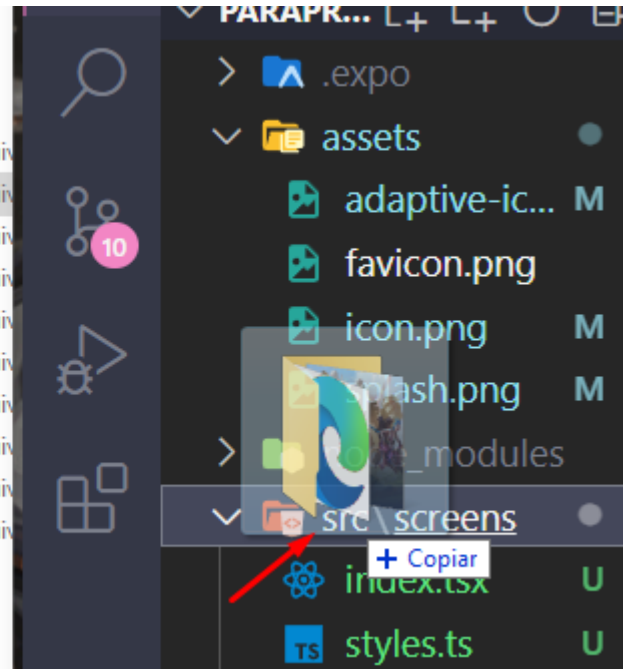


# SPLASH SCREEN

arquivos > src

Pesquisar em src

Nome	Data de modificação	Tipo
@types	21/09/2023 08:48	Pasta de arquivos
assets	21/09/2023 08:48	Pasta de arquivos
components	21/09/2023 08:48	Pasta de arquivos
configs	21/09/2023 08:48	Pasta de arquivos
global	21/09/2023 08:48	Pasta de arquivos
hooks	21/09/2023 08:48	Pasta de arquivos
routes	21/09/2023 08:48	Pasta de arquivos
screens	21/09/2023 08:48	Pasta de arquivos
services	21/09/2023 08:48	Pasta de arquivos
utils	21/09/2023 08:48	Pasta de arquivos



# SPLASH SCREEN

Agora o fundo da ENTRADA da aplicação estará branco precisamos mudar essa configuração.

acesse o arquivo `{..} app.json gameplay` e procure a linha `"backgroundColor": "#0E1647"`

essa linha vai controlar a cor de fundo padrão da nossa aplicação.

Posteriormente você pode escolher qualquer cor, mas dessa vez vamos escolher a cor

`"backgroundColor": "#0D133E"`

para combinar com os assets da nossa aplicação.

# COMPONENTES

Primeiro vamos limpar nossa função `Function Signin()`; e depois vamos importar nos componentes que vamos utilizar.

Deixei nossa function assim:

```
return(  
  <View style={styles.container}>  
    <Image source={} />  
  </View>  
);
```

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

Agora vamos utilizar o componente de imagem.

<IMAGE/>



# COMPONENTES

Agora precisamos importar a imagem vamos utilizar que está na pasta assets.

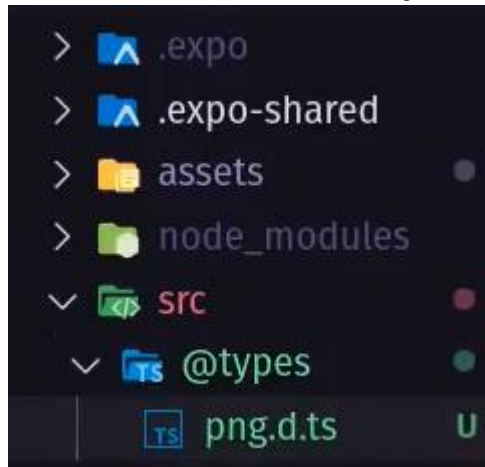
```
import IllustrationImg from '../assets/illustration.png';
```

O Nosso aplicativo não consegue definir a tipagem de um arquivo PNG, então vamos resolver isso agora.

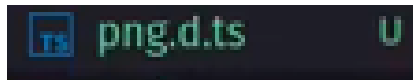
Tipagem dinâmica é uma característica de determinadas linguagens de programação, que não exigem declarações de tipos de dados, pois são capazes de escolher que tipo utilizar dinamicamente para cada variável, podendo alterá-lo durante a compilação ou a execução do programa.

# COMPONENTES

Vamos criar os seguintes arquivos e pastas.



Dentro do arquivo



Vamos definir o tipo de arquivo .Png. .



# COMPONENTES

Agora vamos linkar nossa imagem dentro da function();

```
return(  
  <View style={styles.container}>  
    <Image source={IllustrationImg} />  
  </View>  
>;  
)
```

Assim já podemos verificar ela no nosso app.



# COMPONENTES

Bom agora vamos continuar criando a nossa tela inicial de Sigin();

Atualmente estamos assim:




Queremos ficar assim:



# IMPORTANTE

Caso a imagem não apareça devemos criar logo seu arquivo styles.

 styles.ts U

```
image:{  
  width:'100%',  
  height:360  
}
```

E pra imagem não ficar achatada vamos criar a propriedade.

```
<Image source={IllustrationImg} style={styles.image} resizeMode='stretch' />
```

# COMPONENTES

Vamos adicionar alguns componentes e declarar alguns styles para eles, posteriormente criamos esses estilos .

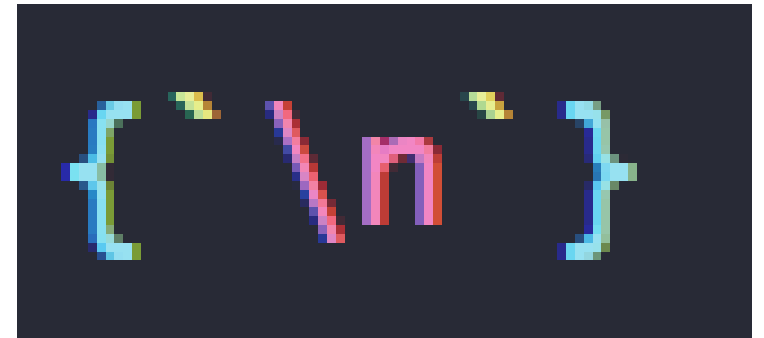
```
<View style = {styles.container}>

  <Image source={illustrationImg} style= {styles.image}/>

  <View style={styles.content}>
    <Text style = {styles.title}>
      Organize {`\n`}
      Suas Jogatinas {`\n`}
      facilmente
    </Text>

    <Text>
      Crie grupo para jogar seus games {`\n`}
      favoritos com seus amigos
    </Text>
  </View>
```

A tag abaixo serve para quebrar linha no texto.



# COMPONENTES

Vamos adicionar alguns componentes e declarar alguns styles para eles, posteriormente criamos esses estilos .

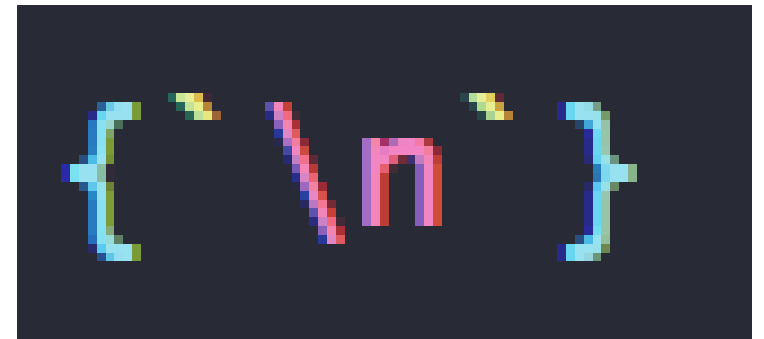
```
<View style = {styles.container}>

  <Image source={illustrationImg} style= {styles.image}/>

  <View style={styles.content}>
    <Text style = {styles.title}>
      Organize {`\n`}
      Suas Jogatinas {`\n`}
      facilmente
    </Text>

    <Text style = {styles.subtitle}>
      Crie grupo para jogar seus games {`\n`}
      favoritos com seus amigos
    </Text>
  </View>
```

A tag abaixo serve para quebrar linha no texto.

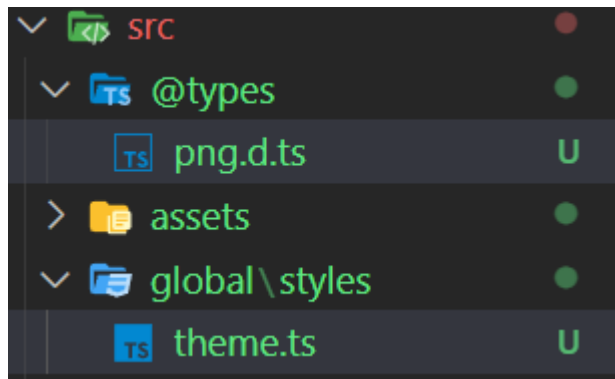


```
{`\n`}
```

# STYLOS

Vamos criar uma pasta para guardar nossas configurações globais.

Ela vai funcionar como uma espécie de arquivo que guarda algo que é universal no nosso aplicativo, exemplo: Cores de fundo, Cores de texto, Transições Etc.



Dentro da pasta Src crie uma pasta global, depois dentro da pasta global crie a pasta 'styles' e dentro da mesma crie o arquivo "theme.ts".



# STYLOS

Dentro do arquivo theme, vamos criar um objeto para exportar nossas cores.

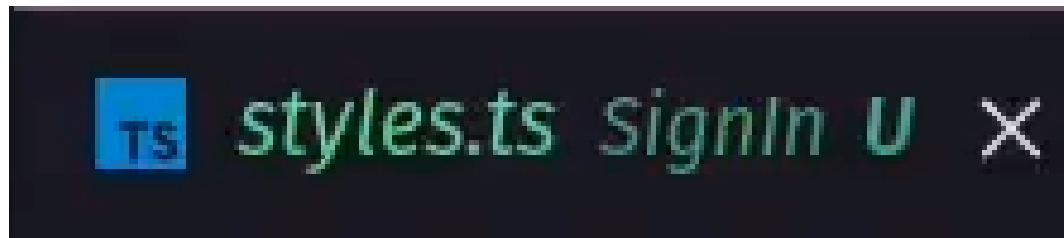
 theme.ts 

```
export const theme = {  
  colors: {  
    backgorung: '#0D133D',  
    headin: '#DDE3F0',  
  }  
}
```

# COMPONENTES

Você deve ter percebido que ficamos com vários erros no nosso código, isso é por conta que declaramos um `style` sem criar o mesmo ainda.

Então agora abra o arquivo `styles.ts` para começarmos a criar o estilo.



# STYLOS DO COMPONENT

No arquivo `styles.ts` vamos primeiro importar o nosso `theme.ts`.

```
import { theme } from '../global/styles/theme';
```

Agora vamos adicionar a nossa cor padrão do aplicativo que troucemos do arquivo `theme.ts`

```
export const styles = StyleSheet.create({
  container:{
    flex: 1,
    backgroundColor:'white',
    alignItems:'center',
    justifyContent: 'center',
    backgroundColor: theme.colors.backgorung
```

# COMPONENT

Agora que estamos criando a primeira tela vamos apagar o `<TextInput>` no arquivo

`styles.ts` U X

```
input: {  
  height: 50,  
  width: 200,  
  borderBottomWidth: 2  
}
```

Vamos definir agora os styles da imagem.

```
image: {  
  width: '100%',  
  height: 360  
}
```

TS styles.ts U X

Agora vamos continuar construindo os estilos da aplicação.

```
content:{  
  marginTop: -40  
},
```

```
title:{  
  color:theme.colors.headin,  
  fontSize:40,  
  marginBottom:16  
},
```

```
subtitle:{  
  color:theme.colors.headin,  
  fontSize:15,  
  textAlign:'center',  
  marginBottom:64  
}
```



Agora vamos trabalhar a statusbar que está fora do tema da nossa aplicação.



```
import {View,Text,TextInput,Image,StatusBar } from 'react-native'
```

Primeiro vamos apagar a linha

```
const [text, setText] = useState ("Você não digitou nada ainda");
```

Pois não vamos utilizar ela agora.

 index.tsx U X

```
<View style = {styles.container}>
```

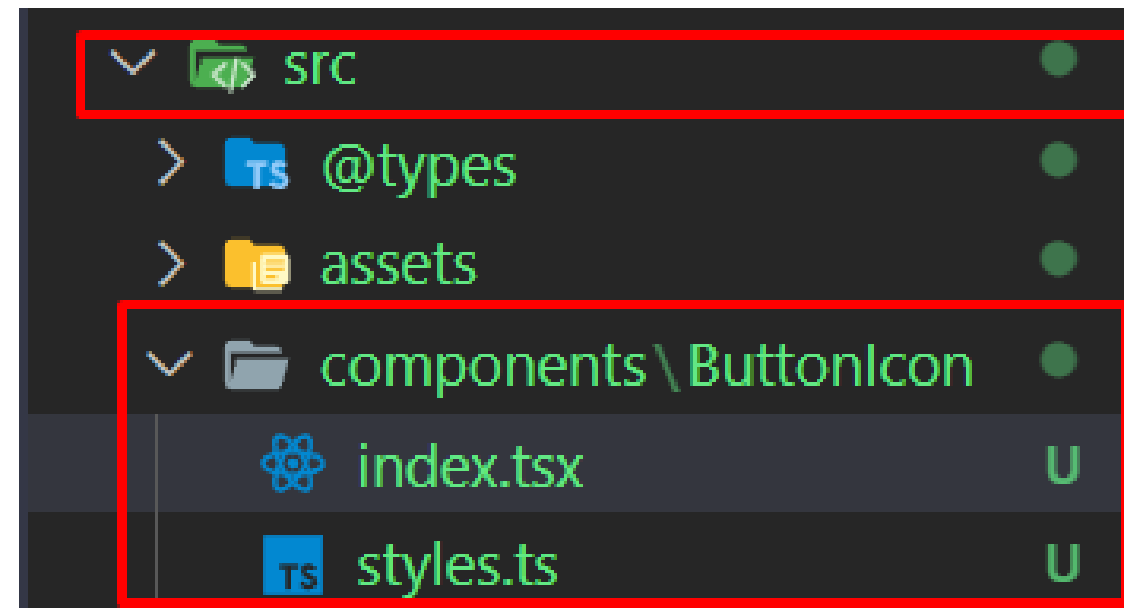
```
  <StatusBar barStyle='light-content' backgroundColor={"transparent"} translucent />
```

```
  <Image source={illustrationImg} style= {styles.image} resizeMode= 'stretch' />
```

CONFIGURAÇÃO CONCLUÍDA! PLIM PLIM !

# AGORA VAMOS CRIAR NOSSO BOTÃO DE LOGUIN COM DISC.

Dentro da pasta SRC crie as seguintes pastas e os seguintes arquivos.






# BOTÃO DE LOGUIN COM DISCORD.

Bom esses dois arquivos que criamos nós já sabemos como eles funcionam.

Vamos codar agora com mais rapidez e explicar apenas conteúdos novos.

 styles.ts U

```
import { StyleSheet } from "react-native";  
  
export const styles = StyleSheet.create ({});
```

# BOTÃO DE LOGUIN COM DISCORD.

Agora no arquivo index do nosso button.

index.tsx U X

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

import DiscordIMG from '../../assets/discord.png'
import {styles} from '../styles'

export function ButtonIcon () {
  return (
    <TouchableOpacity>
      <View>
        <Image />
      </View>
      <Text>
        Entrar com discord
      </Text>
    </TouchableOpacity>
  )
}
```

# BOTÃO DE LOGUIN COM DISCORD.

Agora vamos criar os estilos necessários

index.tsx U ×

```
export function ButtonIcon (){
  return(
    <TouchableOpacity style={styles.container}>
      <View style={styles.iconWrapper}>
        <Image source={DiscordIMG} style={styles.icon} />
      </View>
      <Text style={styles.title}>
        Entrar com discord
      </Text>
    </TouchableOpacity>
  )
}
```

## BOTÃO DE LOGUIN COM DISCORD.

Agora nós vamos até o arquivo que o `index.tsx` que está sendo renderizando pelo arquivo `app.tsx` e importaremos o componente `Buton` que acabamos de criar.

Lembre-se que nós criamos dois `index.tsx` tenha cautela para não confundir.

## BOTÃO DE LOGUIN COM DISCORD.

Agora nós vamos até o arquivo que o `index.tsx` que está sendo renderizando pelo arquivo `app.tsx` e importaremos o componente `Buton` que acabamos de criar.

Lembre-se que nós criamos dois `index.tsx` tenha cautela para não confundir.