

React Native



INICIE SUA JORNADA NO MUNDO MOBILE



O1

O que preciso para começar?



Requisitos mínimos para iniciar

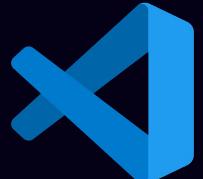
Antes de começar a navegar pelo vasto mundo do React Native, é necessário ter alguns conhecimentos de JavaScript, que estarei listando agora.



- Conceitos avançados de JavaScript:
- Closures
- Callbacks e Promises
- Arrow functions
- Destructuring
- Async/Await
- Let e Const
- Template literals
- Arrow functions
- Classes
- Módulos (import/ export)

Configurando o ambiente

Geralmente, usamos o VSCode como ambiente para o desenvolvimento de aplicativos por muitos motivos, como plugins que agilizam a criação do código ou a questão do uso de snippets.



Já para configurar o ambiente usaremos o Expo, pois ele facilita as instalações além de já possuir diversas bibliotecas prontas para uso.

Porém para começarmos a instalação do expo é necessário ter o Nodejs instalado, já que precisaremos dele para usar o npm e instalar o Expo.



Instalando Expo

Para instalar o Expo basta abrir o (cmd) e executar o seguinte código e então você poderá usar o Expo sempre que precisar.

```
npm install -g expo-cli
```

Criando seu primeiro projeto mobile

Após a instalação do Expo podemos agora criar um novo projeto, fazer isso é bem simples, basta apenas executar o seguinte código no (cmd) e estando na pasta que deseja que o projeto fique:

```
npx create-expo-app nome --template
```

Então no (cmd) será necessário escolher como será gerado o seu projeto, onde as opções são :

- Default
- Blank
- Blank (TypeScript)
- Navigation(TypeScript)
- Blank (Bare)

Como está iniciando use o blank, que é um projeto vazio sem nenhuma configuração, caso já saiba TypeScript pode usar o blank (TypeScript), pois a opção Navigation já virá com o expo router.



02

Entendendo a estrutura básica



Primeiro contato

Em seu projeto vai terá um arquivo chamado App.js ou App.tsx onde terá um código semelhante a esse.

```
1 ~ import { View, Text } from 'react-native'  
2   import React from 'react'  
3  
4 ~ export default function App() {  
5   ~ return (  
6     ~ <View>  
7       ~ <Text>App</Text>  
8     ~ </View>  
9   )  
10 }
```

Vamos começar explicando que a maioria dos projetos atualmente é construída usando funções em vez do conceito de classes, que era mais comum anteriormente.

Como o projeto funciona?

Ao usar o conceito de funções, retornamos a parte visual do aplicativo, que o React Native transforma na interface do usuário.

Então, é importante compreender que tudo o que está dentro dos parênteses após o “return” representa a aparência do seu projeto.

Antes desse “return”, podemos incluir funções e variáveis que serão usadas durante o desenvolvimento do projeto.



Entendo a parte visual do projeto

Em React Native, usamos peças para criar telas de aplicativo. Essas peças são como as coisas que você vê em páginas da web, mas são feitas especialmente para telefones e tablets.

Cada peça tem um trabalho diferente. Por exemplo, o Button (botão) faz algo quando você toca nele, como enviar uma mensagem, enquanto a View (visualização) é como uma caixa que pode conter outras coisas, como texto ou imagens.

Com essas peças, podemos fazer telas legais para nossos aplicativos. À medida que você aprende mais sobre React Native, descobre mais peças e como usá-las para fazer aplicativos bonitos e úteis.



No celular ao lado está sendo usado o Button e um View, é algo bem simples de se ver e imagino que visualmente consiga entender melhor.

O View é uma caixa que você define o tamanho e orientação de tudo que está presente dentro dessa caixa, mas que para definir é preciso usar estilização que veremos mais tarde.

O Button presente dentro da caixa está estilizado com o nome “press me” e mudado a cor para rosa, pois ele é inicialmente azul, além de estar orientado ao centro.



Lista dos componentes básicos

View

Similar a uma div em HTML, é usada para agrupar e estilizar outros componentes.

```
<View>
  /* Outros componentes aqui */
</View>
```

Text

Usado para exibir texto na tela.

```
<Text>Texto aqui</Text>
```

Image

Usado para exibir imagens na tela.

```
<Image source={require('./caminho/para/imagem.png')} />
```

TextInput

Permite que os usuários insiram texto.

```
<TextInput placeholder="Digite aqui" />
```



Lista dos componentes básicos

Button

Cria um botão que os usuários podem pressionar.

```
<Button title="Clique aqui" onPress={() => {}}>
```

TouchableOpacity

Semelhante a um botão, mas oferece feedback visual ao ser pressionado.

```
<TouchableOpacity onPress={() => {}}>
  {/* Conteúdo do botão aqui */}
</TouchableOpacity>
```

ScrollView

Permite a rolagem de conteúdo na tela.

```
<ScrollView>
  {/* Conteúdo rolagem aqui */}
</ScrollView>
```



Lista dos componentes básicos

FlatList

Usada para renderizar uma lista de itens de forma eficiente.

```
<FlatList  
    data={dados}  
    renderItem={({item}) => <Text>{item.texto}</Text>}  
/>
```

Esses são os componentes básicos que mais serão utilizados por você no início de sua jornada.

O título de cada componente é clicável e te levará a página da documentação do componente que tiver clicado caso tenha dúvidas sobre os componentes e as funções que o mesmo possui.

Com o conhecimento fornecido até agora você é capaz de criar telas, porém telas feias, sem nenhum estilo, então no próximo capítulo iremos executar o projeto e aprender a estilização dos componentes.



03

Estilizando seu código



Iniciando seu projeto

Agora vamos executar o projeto e você poderá ver seu primeiro aplicativo, e também só é necessário executar o seguinte código:

```
npm start
```

Depois de terminar de executar o código, você pode acessar o aplicativo pelo aplicativo da expo “Expo Go”, pela web ou por um emulador de celular, as instruções se encontram no próprio (cmd).

Para usar o Expo Go, é necessário criar uma conta e fazer login com essa conta no terminal (cmd), para que quando você execute o projeto, ele apareça automaticamente no seu aplicativo Expo Go.

Para começar o processo de login pelo (cmd), basta utilizar o seguinte comando e preencher os dados que forem solicitados:

```
npx expo login -h
```

Entretanto você pode também conectar no seu Expo Go através do QR Code que aparece no (cmd).

O Expo Go é indicado para pessoas que não possuem um computador com bom desempenho, pois assim ainda conseguem emular em um celular, já que emular web trás algumas restrições visuais.



Estilizando os componentes

A estilização de componentes é o que da forma ao componente, cada componente tem suas limitações de estilização, o que funciona e o que não funciona nele, para saber o que funciona basta acessar a documentação.

Estilos com o StyleSheet

O StyleSheet é a maneira mais comum de definir estilos em um arquivo separado (sendo importado para a tela) ou na mesma função. Isso ajuda na organização e reutilização dos estilos.

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

const App = () => {
  return (
    <View style={styles.container}>
      <Text style={styles.text}>Hello, world!</Text>
    </View>
  );
};

const styles = StyleSheet.create({
  container: {
    flex: 1,
    justifyContent: 'center',
    alignItems: 'center',
  },
  text: {
    fontSize: 20,
    color: 'blue',
  },
});

export default App;
```



Estilos em Linha (Inline Styles)

Você aplica os estilos diretamente no componente usando a propriedade `style`. Essa abordagem é simples e útil para estilos rápidos e específicos.

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

const App = () => {
  return (
    <View style={{ flex: 1, justifyContent: 'center', alignItems: 'center' }}>
      <Text style={{ fontSize: 20, color: 'blue' }}>Hello, world!</Text>
    </View>
  );
};

export default App;
```

NativeWind

NativeWind é uma biblioteca que permite usar classes utilitárias do Tailwind CSS em aplicações React Native, que porém é preciso algumas configurações para ser usado.

```
import { styled } from 'nativewind';

const StyledView = styled(View);
const StyledText = styled(Text);

const App = () => {
  return (
    <StyledView className="flex-1 justify-center items-center bg-gray-100">
      <StyledText className="text-2xl text-blue-500">
        Hello, world!
      </StyledText>
    </StyledView>
  );
};
```



04

React Hooks



useState

O useState é parecido com a declaração de variáveis, porém ele funciona diferente, pois quando alteramos o useState ele é dinamicamente alterado, mudando assim a interface, e caso tenha curiosidade de testar saberá que variáveis não podem fazer isso.

O método de declaração de um estado é mostrado no código abaixo:

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

const ExemplouseState = () => {
  const [count, setCount] = useState(0);

  return (
    <View>
      <Text>Contagem: {count}</Text>
      <Button title="Incrementar" onPress={() => setCount(count + 1)} />
    </View>
  );
};

export default ExemplouseState;
```

O exemplo usa o hook useState para criar um contador simples em um aplicativo React Native. A variável de estado "count" é inicializada com o valor 0 e é atualizada sempre que o botão é pressionado. Ao pressionar o botão, a função "setCount" é chamada para incrementar o valor de "count", causando a atualização do componente e a exibição da contagem atualizada na tela.



useEffect

O useEffect é executado quando queremos que funções aconteçam antes da renderização da interface, porém essa função pode ser chamada mais vezes dependendo daquelas chaves lá embaixo [], caso esteja vazio só rodará uma vez, se tiver uma variável ele vai executar sempre que a variável sofrer alteração.

```
import React, { useEffect, useState } from 'react';
import { Text, View, Button } from 'react-native';

const ExemploUseEffect = () => {
  const [mounted, setMounted] = useState(false);

  useEffect(() => {
    console.log('O componente foi montado');
    setMounted(true);
    return () => console.log('O componente foi desmontado');
  }, []);

  return (
    <View>
      <Text>{mounted ? 'Componente montado' : 'Aguardando montagem'}</Text>
    </View>
  );
};

export default ExemploUseEffect;
```

Este exemplo usa o hook useEffect em um componente funcional em React Native. Quando o componente é montado, "O componente foi montado" é exibido no console. O useEffect é configurado com uma dependência vazia para ser executado uma vez após a montagem. Além disso, mostra como limpar o efeito com a mensagem "O componente foi desmontado" na desmontagem.



useContext

O useContext é utilizado para acessar e utilizar o contexto em componentes funcionais em React Native. Ele permite o acesso a valores fornecidos por um provedor de contexto acima na hierarquia de componentes, facilitando a comunicação entre componentes e o compartilhamento de dados e funcionalidades em toda a aplicação.

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

const ThemeContext = React.createContext('light');

const ExemploUseContext = () => {
  const theme = useContext(ThemeContext);

  return (
    <View>
      <Text>Tema atual: {theme}</Text>
    </View>
  );
};

export default ExemploUseContext;
```

Neste exemplo, o hook useContext é usado para acessar o contexto em um componente funcional em React Native. Um contexto de tema é criado com createContext e fornecido em um nível superior na árvore de componentes. Dentro do componente ExemploUseContext, o hook useContext é usado para acessar o valor do tema fornecido pelo contexto e exibi-lo na tela.



useRef

O useRef é usado para criar uma referência a um elemento do DOM ou a um valor mutável dentro de um componente funcional em React Native. Ele é útil para armazenar referências a elementos, como inputs, ou para manter valores que não precisam estar no estado do componente, mas precisam persistir entre renderizações.

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

const ExemploUseRef = () => {
  const inputRef = useRef(null);

  const focusInput = () => {
    inputRef.current.focus();
  };

  return (
    <View>
      <TextInput ref={inputRef} />
      <Button title="Focus Input" onPress={focusInput} />
    </View>
  );
};

export default ExemploUseRef;
```

O exemplo usa useRef para criar uma referência a um elemento do DOM em um componente funcional em React Native. A referência é para um TextInput e é atribuída à propriedade "ref". Ao pressionar um botão, a função "focusInput" é chamada para acessar a referência ao TextInput e dar foco ao input. Isso demonstra o uso de useRef para interagir com elementos do DOM em componentes funcionais.



05

Navegação



React Navigation

O React Navigation é uma biblioteca react para navegar entre as telas, elas possuem várias formas conhecidas, Stack, Tab e Drawer, que serão explicadas as frente. Apesar dessa biblioteca funcionar o expo tem uma biblioteca que já vem no próprio expo porém precisa ser configurado o Expo Router.

Os métodos de navegação tem sempre uma mesma forma de navegar que é através do nome da tela, porém antes é sempre necessário cadastrar a tela no seu arquivo de rotas.

No link do título tem as instalação para usar a biblioteca.

```
// screens/HomeScreen.js
import React from 'react';
import { View, Text, Button } from 'react-native';

const HomeScreen = ({ navigation }) => {
  return (
    <View style={{ flex: 1, justifyContent: 'center', alignItems: 'center' }}>
      <Text>Home Screen</Text>
      <Button
        title="Go to Details"
        onPress={() => navigation.navigate('Details')}
      />
    </View>
  );
};

export default HomeScreen;
```

Este código é um exemplo navegação entre telas, onde na tela temos o botão e quando clicando ele navegará para a tela que possui o nome passado, que nesse caso foi Details, que você pode ver em aspas simples, para navega entre telas com essa biblioteca sempre será usado o “navigation.navigate(nome)”.



Stack

O método de navegação Stack funciona exatamente o que significa pilha, pois ele vai empilhando as telas uma atrás da outras e quando você coloca para voltar ele desempilha as telas voltando uma por uma.

Essa seria a forma de você inicializar o arquivo para cadastrar telas, usando a tag Stack.Screen você pode cadastrar o componente referente a cada tela e os nomes das telas, além de poder fazer configurações adicionais a tela.

```
import { createStackNavigator } from '@react-navigation/stack';

const Stack = createStackNavigator();

function MyStack() {
  return (
    <Stack.Navigator>
      <Stack.Screen name="Home" component={Home} />
      <Stack.Screen name="Notifications" component={Notifications} />
      <Stack.Screen name="Profile" component={Profile} />
      <Stack.Screen name="Settings" component={Settings} />
    </Stack.Navigator>
  );
}


```

Quando acabamos a base desse arquivo, importamos ele diretamente no arquivo App onde colocamos ele para que a aplicação tenha acesso a todas as telas.

Esse método possui várias funcionalidades e tanto os métodos quanto a forma de instalação estão no link no título.



Tabs

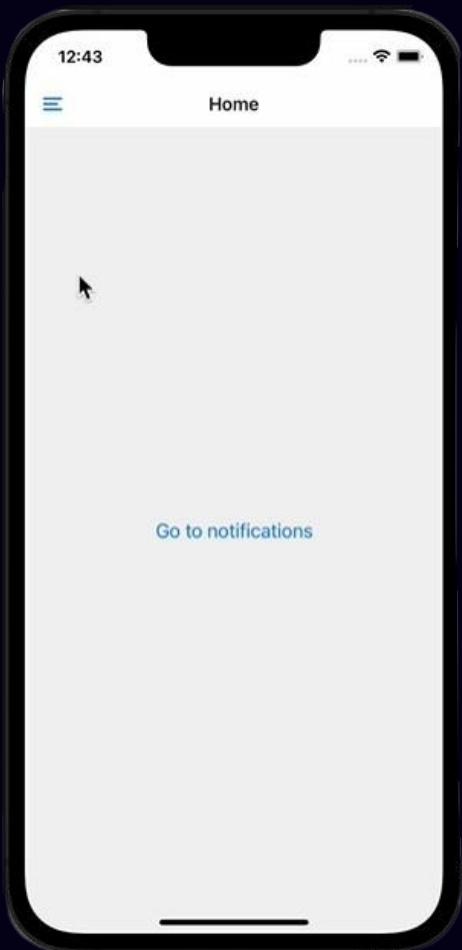
Como é possível ver na imagem ao lado a diferença para a navegação Stack é que a navegação Tab tem essa barra embaixo com n possíveis telas para se navegar e talvez um ícone.

Como o Stack ele tem as mesma funcionalidades porém algumas partes de sua instalação é diferente, então é só clicar no título para verificar a instalação.

O arquivo de rotas também tem diferenças por conta do import ser diferente por ser outro tipo de navegação.



Drawer



Como é possível ver na imagem ao lado a diferença para a navegação Stack é que a navegação Drawer tem esse ícone na esquerda onde abre uma aba para clicar em uma das n telas que podem ser cadastradas lá.

Como o Stack ele tem as mesma funcionalidades porém algumas partes de sua instalação é diferente, então é só clicar no título para verificar a instalação.

O arquivo de rotas também tem diferenças por conta do import ser diferente por ser outro tipo de navegação.



Expo router

O arquivo de rotas se torna muito mais simples usando esse método de navegação.

Todas as informações descritas pelo próprio Expo dessa forma de navegação:

- **Nativo** : construído com base em nosso poderoso conjunto React Navigation, a navegação do Expo Router é verdadeiramente nativa e otimizada para plataforma por padrão.
- **Compartilhável** : todas as telas do seu aplicativo podem ser automaticamente vinculadas diretamente. Tornando qualquer rota em seu aplicativo compartilhável com links.
- **Primeiro off-line** : os aplicativos são armazenados em cache e executados primeiro off-line, com atualizações automáticas quando você publica uma nova versão. Lida com todos os URLs nativos recebidos sem uma conexão de rede ou servidor.
- **Otimizado** : as rotas são otimizadas automaticamente com avaliação lenta na produção e agrupamento adiado no desenvolvimento.
- **Iteração** : atualização rápida universal no Android, iOS e web, junto com memorização de artefatos no bundler para manter você avançando rapidamente em escala.
- **Universal** : Android, iOS e web compartilham uma estrutura de navegação unificada, com a capacidade de acessar APIs específicas da plataforma no nível da rota.
- **Detectável** : Expo Router permite renderização estática em tempo de construção na web e link universal para nativo. Isso significa que o conteúdo do seu aplicativo pode ser indexado por mecanismos de pesquisa.



06

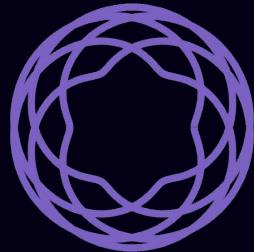
Links



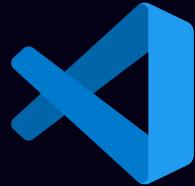
Links úteis



[Expo](#)



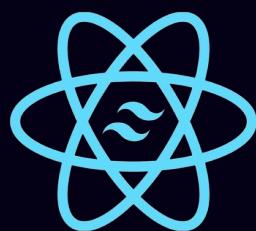
[React Navigation](#)



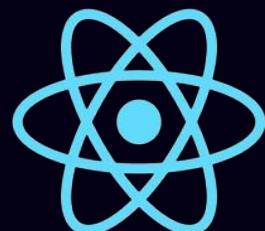
[VSCode](#)



[Nodejs](#)



[NativeWind](#)



[React Native](#)

A todos que leram até o final, agradeço. Deixei muitas coisas sem explicar, mas espero que busquem como se tornar um programador mobile de verdade, pois este e-book é só o primeiro passo.

