

# Programação Mobile

Professor: Elton Sarmanho<sup>1</sup>

E-mail: [eltonss@ufpa.br](mailto:eltonss@ufpa.br)



<sup>1</sup>Faculdade de Sistemas de Informação - UFPA/CUNTINS

10 de junho de 2025

# Roteiro

Introdução

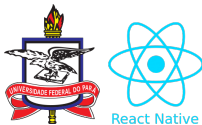
Principais Componentes React Native

Design

Usabilidade

Listas





# Programação Mobile

Professor: Elton Sarmanho<sup>1</sup>

E-mail: eltonss@ufpa.br



<sup>1</sup>Faculdade de Sistemas de Informação - UFPA/CUNTINS

10 de junho de 2025



# Conceitos Introdutórios



# Introdução

## ▶ Node.js:

- ▶ Ambiente de tempo de execução JavaScript de código aberto e multiplataforma.
- ▶ Permite executar JavaScript fora do navegador, principalmente para servidores e ferramentas de linha de comando.
- ▶ Essencial para o ecossistema de desenvolvimento web moderno.
- ▶ Site oficial: <https://nodejs.org/en>

## ▶ Expo:

- ▶ Framework e plataforma para construir aplicativos universais React Native (Android, iOS e Web).
- ▶ Simplifica o desenvolvimento, abstraindo a configuração nativa complexa.
- ▶ Permite testar rapidamente em dispositivos reais sem grandes configurações.



## Pré-requisitos

Para seguir este tutorial, você precisará de:

- ▶ Um sistema operacional **Linux**.
- ▶ Acesso a um **terminal** (linha de comando).
- ▶ Uma **conexão ativa com a internet**.
- ▶ **Opcional:** Um smartphone (Android ou iOS) para testar os aplicativos Expo.



# Instalação do Node.js: Método Alternativo (Manual)

Para o arquivo `node-v22.16.0-linux-x64.tar.xz` baixado

1. **Descompactar o arquivo:** (Navegue até o diretório onde o arquivo está)

```
tar -xf node-v22.16.0-linux-x64.tar.xz
```

2. **Mover para um diretório do sistema (ex: /usr/local):**

```
sudo mv node-v22.16.0-linux-x64 /usr/local/node
```

3. **Adicionar ao PATH:**

```
nano ~/.bashrc
```

//Adicione a seguinte linha no final do arquivo:

```
export PATH="/usr/local/node/bin:$PATH"
```

4. **Recarregar o shell:**

```
source ~/.bashrc
```



# Instalação do Node.js: Método Alternativo (Manual)

Para o arquivo `node-v22.16.0-linux-x64.tar.xz` baixado

1. **Descompactar o arquivo:** (Navegue até o diretório onde o arquivo está)
2. **Mover para um diretório do sistema (ex: `/usr/local`):**
3. **Adicionar ao PATH:**
4. **Recarregar o shell:**
5. **Verificar a instalação:**

```
node -v
```

```
npm -v
```





# Instalação do Node.js: Método Recomendado (NVM)

## Node Version Manager

- ▶ Recomendamos usar o **NVM (Node Version Manager)**.
- ▶ **Vantagens do NVM:**
  - ▶ Instalar e gerenciar múltiplas versões do Node.js.
  - ▶ Trocar facilmente entre as versões (útil para diferentes projetos).
  - ▶ Evita problemas de permissão com o 'sudo'.



# Instalação do NVM

Execute os comandos no seu terminal:

**1. Baixar e executar o script de instalação do NVM:**

- ▶ URL ⇒ `https://raw.githubusercontent.com/nvm-sh/nvm/master/install.sh`
- ▶ `curl -o- URL | bash`

**2. Configurar o seu shell (Bash/Zsh):** Após a instalação, feche e reabra o terminal, ou execute:

```
source ~/.bashrc  
# OU (se usar Zsh)  
source ~/.zshrc
```

**3. Verificar a instalação do NVM:**

```
nvm --version
```



# Instalação do Node.js com NVM

## 1. Instalar a versão LTS (Long Term Support) do Node.js:

```
nvm install --lts
```

- ▶ Você também pode instalar uma versão específica: `nvm install 20` ou `nvm install 22.16.0`

## 2. Definir a versão padrão (opcional, mas recomendado):

```
nvm alias default node # Usará a versão LTS ins
```

## 3. Verificar a instalação do Node.js e npm:

```
node -v  
npm -v
```

- ▶ Você deve ver as versões instaladas.



## Instalação do Expo: Expo CLI (Computador)

O Expo CLI é a ferramenta de linha de comando para criar e gerenciar seus projetos Expo.

### 1. Instalação:

```
sudo apt install npm
```

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

### 2. Verificar a instalação:

```
expo --version
```

### 3. Como usar o Expo depois de instalado:

```
expo init meu-primeiro-app-expo
```



## Instalação do Expo: Expo Go (Dispositivo Móvel)

O Expo Go é um aplicativo móvel para testar seus projetos Expo em tempo real.

### 1. Baixar o Expo Go no seu smartphone:

- ▶ **Android:** Abra a **Google Play Store**, pesquise por "Expo Go" e instale.
- ▶ **iOS:** Abra a **App Store**, pesquise por "Expo Go" e instale.

### 2. Conectar seu dispositivo ao projeto:

- ▶ Com `npx expo start` rodando no seu computador:
- ▶ **Android:** Abra o aplicativo **Expo Go**, toque em "Scan QR Code" e escaneie o código QR do seu terminal.
- ▶ **iOS:** Use o aplicativo **Câmera** do iPhone para escanear o código QR. Um banner aparecerá para abrir no Expo Go.



## Verificação da Instalação do Expo

- ▶ Após escanear o QR Code, seu aplicativo deve carregar no Expo Go.
- ▶ Alterações no código do seu projeto no computador serão refletidas instantaneamente no seu dispositivo.

**Parabéns! Você tem o Node.js e o Expo configurados para começar a desenvolver seus aplicativos React Native.**



## Dicas Finais

- ▶ **Mantenha suas ferramentas atualizadas:**
  - ▶ `npm install -g npm@latest` (para atualizar o npm)
  - ▶ `nvm install --lts` (para atualizar Node.js com NVM)
- ▶ **Consulte a documentação oficial:**
  - ▶ Node.js: <https://nodejs.org/en/docs/>
  - ▶ Expo: <https://docs.expo.dev/>
- ▶ **Comunidade:** Participe de fóruns e grupos para tirar dúvidas.



# Principais Componentes





## O que vamos aprender

- ▶ View
- ▶ Image
- ▶ ScrollView
- ▶ Text
- ▶ TextInput
- ▶ StyleSheet
- ▶ Button
- ▶ Switch
- ▶ Documentação



## Como começar ?

- ▶ VS Code
- ▶ Gemini Code Assist
- ▶ Expo Go + Celular (Não vamos usar Emulador)



# Metodologia

- ▶ *learning by doing*
- ▶ Documentação:  
<https://reactnative.dev/docs/getting-started>



# Projeto 1

## Calculadora de IMC

Peso (kg):

Altura (m):

**Calcular IMC**

Seu IMC é:

**23,02**



## Componente View

- ▶ O componente mais fundamental para construir a interface do usuário.
- ▶ É um contêiner flexível que suporta layout com **flexbox**, estilo, e manuseio de toque.
- ▶ Equivale à `div` no HTML, mas com capacidades de estilo e layout otimizadas para mobile.
- ▶ **Documentation**
- ▶ Pode ser usado para criar layouts complexos, agrupando outros componentes.
- ▶ É a base para a maioria dos elementos visuais em uma aplicação React Native.

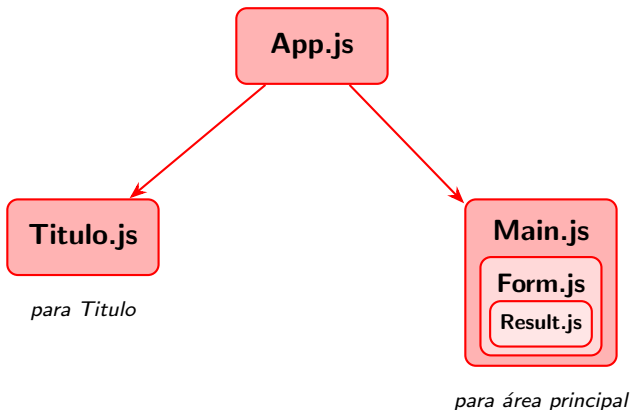


# Vamos Codar!

- ▶ Execute: **cd meu-primeiro-app → npm start**



# Hierarquia dos Componentes



## └ Principais Componentes React Native

- ▶ Título → Título do app
- ▶ Main → Estilizar área principalmente
- ▶ Form → Estrutura do Formulário
- ▶ Result → Visualizar área de Resultado





# Projeto 1!

## Calculadora de IMC

Peso (kg):

Altura (m):

**Calcular IMC**

Seu IMC é:

**23,02**



# Passagem de Parâmetros (Props) em React Native

Em React Native, os componentes são a base da UI. Para torná-los dinâmicos e reutilizáveis, usamos **props** (abreviação de "properties-properties").

## ▶ O que são Props?

- ▶ São argumentos que você passa para os componentes React.
- ▶ Permitem que você envie dados de um componente "pai" para um componente "filho".
- ▶ São **somente leitura** (read-only): um componente nunca deve modificar suas próprias props.

## ▶ Como usar Props?

- ▶ O componente "filho" recebe as props como um objeto no seu primeiro argumento.
- ▶ Você acessa os valores das props usando a sintaxe de ponto, por exemplo, `props.nomeDaPropriedade`.



## Exemplo do seu código:

```
1 import React from 'react';
2 import { Text } from 'react-native';
3 export default function Result (props) {
4   return (
5     <Text>
6       Seu IMC : { props.valor }
7     </Text>
8   );
9 }
```

- ▶ Neste exemplo, o componente Result recebe um objeto props.
- ▶ Ele acessa a propriedade valor desse objeto para exibir o valor do IMC.
- ▶ Um componente pai passaria o valor assim: <Result valor={25.5} />



## Gerenciamento de Estado com useState

Enquanto **props** permitem a passagem de dados de um componente pai para um filho, o **estado** (state) permite que um componente gerencie e atualize seus próprios dados internos.

### ▶ O que é Estado?

- ▶ Dados que podem mudar ao longo do tempo dentro de um componente.
- ▶ Quando o estado de um componente muda, o React o renderiza novamente para refletir as alterações na UI.

### ▶ O Hook useState:

- ▶ É uma função do React que permite adicionar estado a componentes de função.
- ▶ Retorna um par de valores:
  - ▶ O valor atual do estado.
  - ▶ Uma função para atualizar esse valor.



### ► Sintaxe Básica:

```
1  import React, { useState } from 'react';  
2  function Contador() {  
3      const [count, setCount] = useState(0);  
4      return (  
5          <View>  
6              <Text>Contagem: {count}</Text>  
7              <Button title="Aumentar" onPress={() =>  
                  setCount(count + 1)} />  
8          </View>  
9      );  
10 }
```

### ► Pontos Chave:

- Importe `useState` de `'react'`.
- A função de atualização (`setCount` no exemplo) é assíncrona.
- A alteração do estado causa uma nova renderização do componente.



## Funções em JavaScript e React Native

Uma **função** é um bloco de código reutilizável que executa uma tarefa específica. Em JavaScript, elas são um conceito fundamental, e no React Native, são a base para a criação de componentes funcionais.

### ► Definição Básica:

- Um conjunto de instruções que pode ser chamado (executado) a qualquer momento.
- Pode receber **parâmetros** (entradas) e retornar um **valor** (saída).



### ► Declaração de Função (JavaScript):

```
1 function saudacao(nome) {  
2     return "Olá, " + nome + "!";  
3 }  
4 const mensagem = saudacao("Mundo"); // Chamada da função
```



## ▶ Funções como Componentes em React Native:

- ▶ No React Native moderno, a maioria dos componentes é criada como funções.
- ▶ Essas funções recebem props como argumento e retornam elementos JSX (a descrição da UI).
- ▶ São chamadas de **Componentes Funcionais**.

## ▶ Exemplo de Componente Funcional:

```
1  import React from 'react';
2  import { Text, View } from 'react-native';
3
4  function BoasVindas(props) {
5      <View>
6          <Text>Bem-vindo, {props.usuario}!</Text>
7      </View>
8  };
9  }
10 // Uso: <BoasVindas usuario="Ana" />
```





▶ **Arrow Functions (Funções de Seta):**

- ▶ Uma sintaxe mais concisa para escrever funções, muito comum em React.
- ▶ Exemplo: `const minhaFuncao = (param) => { /* código */ }`;



## Voltando ao Projeto 1

- ▶ Criar duas Funções
  1. *calcularImc*
  2. *validatorIMC*
    - ▶ Recomendação: Setar valores como *null*



## Propriedade onChangeText

A propriedade `onChangeText` é essencial para capturar a entrada de texto do usuário em componentes como `TextInput` no React Native.

▶ **Finalidade:**

- ▶ É uma função de callback que é invocada sempre que o texto em um `TextInput` muda.
- ▶ Permite que você reaja às alterações do usuário e atualize o estado do seu componente.



### ▶ Exemplo de Uso com TextInput e useState:

```
1 import React, { useState } from 'react';
2 import { TextInput, Text, View } from 'react-native';
3 function CampoDeTexto() {
4   const [textoDigitado, setTextoDigitado] = useState('');
5
6   <TextInput onChangeText={novoTexto =>
7     setTextoDigitado(novoTexto)}
8     value={textoDigitado}/>;
9 }
```

### ▶ Pontos Importantes:

- ▶ Combine onChangeText com a propriedade value para criar um **campo de entrada controlado**.
- ▶ O estado (textoDigitado) mantém o valor do TextInput, e onChangeText é responsável por atualizá-lo.



### ► Exemplo de Uso com TextInput e useState:

```
1 import React, { useState } from 'react';
2 import { TextInput, Text, View } from 'react-native';
3 function CampoDeTexto() {
4   const [textoDigitado, setTextoDigitado] = useState('');
5
6   <TextInput onChangeText={setTextoDigitado}
7             value={textoDigitado}/>;
8 }
```

### ► Pontos Importantes:

- Combine onChangeText com a propriedade value para criar um **campo de entrada controlado**.
- O estado (textoDigitado) mantém o valor do TextInput, e onChangeText é responsável por atualizá-lo.



# Design da Aplicação



# Projeto 1!

## Calculadora de IMC

Peso (kg):

Ex: 70.5

Altura (m):

Ex: 1.75

Calcular IMC

Seu IMC é:

**23,02**



## Estilização com StyleSheet

No React Native, a estilização é feita usando JavaScript, e a API StyleSheet é a forma recomendada para criar e gerenciar estilos.

- ▶ **O que é StyleSheet?**

- ▶ É uma abstração que otimiza a criação de estilos no React Native.
- ▶ Permite definir estilos de forma semelhante ao CSS, mas em JavaScript.

- ▶ **Método StyleSheet.create():**

- ▶ É o método principal para criar um objeto de estilo.
- ▶ Recebe um objeto JavaScript onde cada chave é um nome de estilo e o valor é outro objeto com as propriedades de estilo.
- ▶ Exemplo: `StyleSheet.create({ container: { flex: 1, ... } })`





## ► Exemplo de Uso no App.js:

```
1  import { StyleSheet, View } from 'react-native';
2  export default function App() {
3    return (
4      <View style={styles.container}>
5        /* Conteúdo do seu app */
6      </View>
7    );
8  }
9  const styles = StyleSheet.create({
10    container: {
11      flex: 1, // Ocupa todo o espaço disponível
12      backgroundColor: '#e0e0e0', // Cor de fundo suave
13      paddingTop: 50, // Espaçamento superior
14      paddingHorizontal: 20, // Espaçamento lateral
15      alignItems: 'center', // Centraliza Horizontal
16    },});
```



► **Benefícios:**

- **Performance:** Estilos são processados uma única vez e referenciados por ID.
- **Organização:** Mantém os estilos encapsulados e legíveis.
- **Reutilização:** Estilos podem ser facilmente reutilizados em diferentes componentes.



# Projeto 1

Calculadora de IMC

Peso (Kg)

Ex: 70.5

Altura (m)

Ex: 1.75

**CALCULAR IMC**

Preencha os campos corretamente



# Projeto 1

- ▶ Vamos aplicar Estilo no Componente **Form.js**



# Projeto 1

## Calculadora de IMC

Peso (kg):

Ex: 70.5

Altura (m):

Ex: 1.75

Calcular IMC

Preencha os campos corretamente



## Perguntas?

# Obrigado!

Dúvidas? Entre em contato ou consulte a documentação oficial.



# Usabilidade da Aplicação



- ▶ Mensagem de Erro
- ▶ Vibration
  - ▶ Docs
- ▶ Share
  - ▶ Docs





# API Vibration

A API `Vibration` no React Native permite adicionar feedback tátil (vibração) ao seu aplicativo, melhorando a experiência do usuário em resposta a ações ou eventos.

## ▶ O que é?

- ▶ Uma API nativa do React Native que permite controlar o vibrador do dispositivo.
- ▶ Proporciona um feedback físico para o usuário, ideal para confirmações, alertas ou interações.

## ▶ Como usar?

- ▶ Importe `Vibration` de `'react-native'`.
- ▶ Utilize o método `Vibration.vibrate()`.
- ▶ Para parar a vibração, utilize `Vibration.cancel()`.



► **Permissões (Android):**

- No Android, é necessário adicionar a permissão VIBRATE no arquivo `AndroidManifest.xml`:

```
1 <uses-permission android:name="android.permission.  
    VIBRATE"/>
```



# API Share

A API Share no React Native permite que seu aplicativo interaja com o sistema de compartilhamento nativo do dispositivo, possibilitando que os usuários compartilhem conteúdo com outros aplicativos.

## ▶ O que é?

- ▶ Uma API que fornece acesso à caixa de diálogo de compartilhamento nativa do sistema operacional (Android e iOS).
- ▶ Permite compartilhar texto, URLs e, em alguns casos, outros tipos de dados com aplicativos instalados no dispositivo (e-mail, redes sociais, mensageiros, etc.).

## ▶ Como usar?

- ▶ Importe Share de `'react-native'`.
- ▶ Utilize o método assíncrono `Share.share()`.

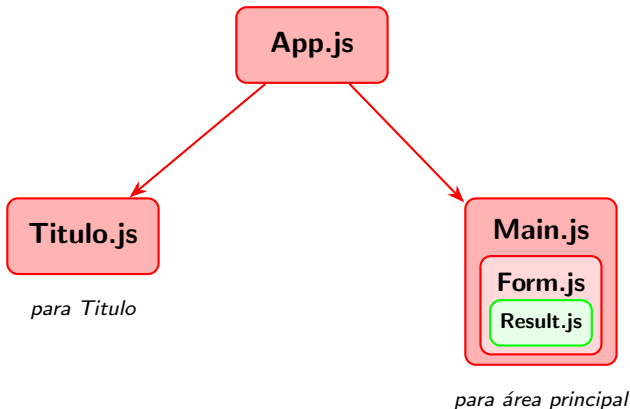


# API Share

- ▶ **Método Principal:** `Share.share(content, options?)`:
  - ▶ `content` (objeto): O conteúdo a ser compartilhado.
    - ▶ `message` (string): Mensagem de texto para compartilhar.
    - ▶ `url` (string): URL para compartilhar (Android e iOS).
    - ▶ `title` (string): Título para a caixa de diálogo de compartilhamento (iOS).
  - ▶ `options` (objeto, opcional): Opções adicionais para o compartilhamento.
    - ▶ `dialogTitle` (string): Título da caixa de diálogo no Android.
    - ▶ `tintColor` (string): Cor do texto do botão no iOS.
- ▶ **Considerações:**
  - ▶ A disponibilidade de opções de compartilhamento depende dos aplicativos instalados no dispositivo do usuário.
  - ▶ É uma funcionalidade nativa do sistema operacional, o que garante uma experiência consistente para o usuário.



## Hierarquia dos Componentes



- ▶ No decorrer do desenvolvimento você deve testar em ambas as plataformas



# Listas No React Native



# Conteúdo

- ▶ Condicionais
- ▶ Lista e Scroll





# Condicionais em React Native (Renderização Condicional)

Em React Native, a **renderização condicional** é a capacidade de exibir diferentes elementos ou componentes da UI com base em uma condição. Isso permite criar interfaces dinâmicas que se adaptam aos dados ou ao estado do aplicativo.

▶ **Conceito:**

- ▶ Controlar o que é renderizado na tela com base em expressões lógicas (verdadeiro/falso).

▶ **Métodos Comuns:**

1. **Operador Ternário** (`condicao ? expressao1 : expressao2`):

- ▶ Sintaxe concisa para escolher entre duas opções.
- ▶ Ideal para renderização "se/senão".

```
1 <View>
2   {isLoading ? <ActivityIndicator /> : <Text>
3     Dados Carregados</Text>}
4 </View>
```



# Condicionais em React Native (Renderização Condicional)

## ► Métodos Comuns:

### 1. Operador Lógico && (condicao && expressao):

- Renderiza um componente **apenas se** a condição for verdadeira.
- Se a condição for falsa, nada é renderizado.
- Útil para renderizar algo ou nada.

```
1 <View>  
2   {showWarning && <Text>Cuidado!</Text>}  
3 </View>
```



# Condicionais em React Native (Renderização Condicional)

## ► Métodos Comuns:

### 1. Instruções if/else (fora do JSX):

- Você pode usar if/else dentro da função do componente para retornar diferentes blocos de JSX.
- Mais verboso, mas útil para lógicas complexas.

```
1 function MensagemBoasVindas(props) {  
2   if (props.isLoggedIn) {  
3     return <Text>Bem-vindo de volta!</Text>;  
4   }  
5   return <Text>Por favor, faça login.</Text>;  
6 }
```

## ► Importância:

- Permite criar interfaces de usuário flexíveis e responsivas.
- A UI se adapta dinamicamente às interações do usuário e ao estado da aplicação.



## Listas

```

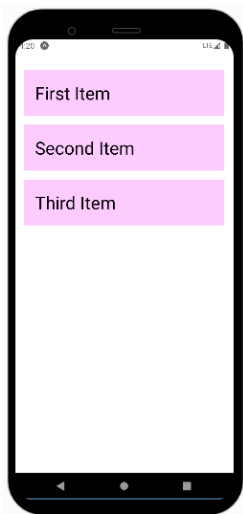
Windsurf: Refactor | Explain | Generate JSDoc | x
export default function Result(props) {
  //Peguei direto da documentação do react native
  Windsurf: Refactor | Explain | x
  const onShare = async () => {
    const result = await Share.share({
      message:
        'Meu imc é: ' + props.valor.toFixed(2),
    });
  });

  return (
    <View>
      <View>
        <Text>
          {props.msg} {props.valor ? props.valor.toFixed(2) : ''}
        </Text>
      </View>
      {props.valor !== null && props.valor > 0 ?
        <View style={styles.shareBox}>
          <TouchableOpacity onPress={onShare}>
            <Text>Compartilhar</Text>
          </TouchableOpacity>
        </View>:null
      }
    </View>
  );
}

```



## Componente FlatList: Introdução



## Componente FlatList: Introdução

O componente FlatList é a maneira mais eficiente de renderizar listas longas ou que mudam dinamicamente no React Native. Ele é otimizado para performance.

- ▶ **Por que usar FlatList?**

- ▶ **Performance Superior:** Renderiza apenas os itens visíveis na tela, economizando memória e processamento.
- ▶ **Rolagem Suave:** Lida com listas grandes sem travamentos, mesmo em dispositivos mais antigos.
- ▶ **Funcionalidades Integradas:** Suporta rolagem infinita (lazy loading), pull-to-refresh, cabeçalhos/rodapés, e mais.



## Componente FlatList: Propriedades Essenciais

Para utilizar o FlatList, algumas propriedades são fundamentais para definir como seus dados serão exibidos.

- ▶ **data (array):**
  - ▶ A fonte de dados para a lista.
  - ▶ Deve ser um array de objetos.
- ▶ **renderItem (função):**
  - ▶ Uma função que recebe um objeto { item, index, separators } e retorna um componente React para renderizar cada item.
- ▶ **keyExtractor (função):**
  - ▶ Uma função que recebe um item e seu índice, e retorna uma string única para cada item.
  - ▶ Essencial para a performance do React, pois ajuda a identificar quais itens foram alterados, adicionados ou removidos, otimizando as atualizações.



Veja um exemplo prático de como o FlatList é implementado em um componente React Native.

```
1 import { FlatList, Text, View, StyleSheet } from 'react-  
  native';  
2 const DATA = [  
3   { id: '1', title: 'Item Um' },  
4   { id: '2', title: 'Item Dois' }  
5   // ... muitos mais itens  
6 ];  
7  
8 function MinhaLista() {  
9   return (  
10    <FlatList  
11      data={DATA}  
12      renderItem={({ item }) => <Text>{item.title}</Text>  
13      keyExtractor={(item, index) => index.toString()  
14    } />  
15  );
```





# Câmera No React Native



# Conteúdo

- ▶ API da câmera
- ▶ permissão e propriedades disponíveis



## API Camera: Introdução

A API Camera no React Native permite que seu aplicativo acesse e utilize a câmera do dispositivo para tirar fotos e gravar vídeos.

- ▶ **O que é?**

- ▶ Um módulo que oferece uma interface para interagir com a câmera nativa do dispositivo.

- ▶ **Bibliotecas Comuns:**

- ▶ expo-camera: Para projetos Expo. Mais fácil de configurar e usar.
- ▶ react-native-camera: Para projetos React Native CLI. Oferece mais controle e personalização.

- ▶ **Funcionalidades Principais:**

- ▶ Pré-visualização da câmera.
- ▶ Captura de fotos.
- ▶ Gravação de vídeos.
- ▶ Controle de flash, zoom, foco e balanço de branco.
- ▶ Troca entre câmeras frontal e traseira.



# API Camera: Introdução

## ▶ Documentação

▶ *`npx expo install expo-camera`*



## API Camera: Permissões

Para que seu aplicativo possa acessar a câmera e o microfone (para vídeo), é necessário solicitar permissões ao usuário.

- ▶ **Por que Permissões?**

- ▶ Por questões de segurança e privacidade, os sistemas operacionais (Android e iOS) exigem que os aplicativos solicitem explicitamente o acesso a recursos sensíveis como a câmera e o microfone.

- ▶ **Como Solicitar (Exemplo com Expo):**

- ▶ Utilize o módulo expo-camera e a API Permissions do Expo.
- ▶ É uma boa prática verificar e solicitar permissões no início do uso da câmera.



## API Camera: Importações Essenciais

Para utilizar a API da câmera, é fundamental importar os componentes e hooks corretos da biblioteca expo-camera e do React Native.

### ▶ **Componentes e Hooks da Câmera:**

- ▶ **CameraView:** O componente principal que exibe a pré-visualização da câmera em tempo real.
- ▶ **CameraType:** Uma enumeração que define os tipos de câmera disponíveis (geralmente 'front' ou 'back').
- ▶ **useCameraPermissions:** Um hook personalizado do Expo para gerenciar o status das permissões da câmera de forma reativa.

### ▶ **Importações Comuns do React Native:**

- ▶ **useState:** Hook para gerenciar o estado dentro de componentes funcionais.
- ▶ **Button:** Componente para criar botões interativos.
- ▶ **StyleSheet:** API para criar e gerenciar estilos de forma otimizada.



```
1 import { CameraView, CameraType, useCameraPermissions }  
    from 'expo-camera';  
2 import { useState } from 'react';  
3 import { Button, StyleSheet, Text, TouchableOpacity, View  
    } from 'react-native';
```



## API Camera: Gerenciamento de Estado e Permissões

O código da câmera depende do gerenciamento de estado para controlar a câmera ativa e o status das permissões.

### ▶ Estado da Câmera Ativa (facing):

- ▶ `const [facing, setFacing] = useState('back');`
- ▶ A variável `facing` armazena a câmera atualmente selecionada ('back' para traseira, 'front' para frontal).
- ▶ `setFacing` é a função usada para alternar entre as câmeras.

### ▶ Estado das Permissões (permission):

- ▶ `const [permission, requestPermission] = useCameraPermissions();`
- ▶ `permission` é um objeto que contém informações sobre o status da permissão (ex: `permission.granted`).
- ▶ `requestPermission` é uma função para solicitar a permissão da câmera ao usuário.





## API Camera: Gerenciamento de Estado e Permissões

O código da câmera depende do gerenciamento de estado para controlar a câmera ativa e o status das permissões.

### ▶ **Lógica de Verificação de Permissão:**

- ▶ O aplicativo verifica se as permissões estão sendo carregadas (!permission).
- ▶ Se as permissões não foram concedidas (!permission.granted), uma interface é exibida solicitando ao usuário que conceda o acesso à câmera.



```
1  const [facing, setFacing] = useState('back');
2  const [permission, requestPermission] =
    useCameraPermissions();
3
4  if (!permission) {
5    // Camera permissions are still loading.
6    return <View />;
7  }
8  if (!permission.granted) {
9    // Camera permissions are not granted yet.
10   return (
11     <View style={styles.container}>
12       <Text style={styles.message}>We need your
          permission to show the camera</Text>
13       <Button onPress={requestPermission} title="grant
          permission" />
14     </View> ); }
```



## API Camera: Funcionalidades de Controle

O controle da câmera, como alternar entre as câmeras frontal e traseira, é implementado através de funções e interações com o usuário.

- ▶ **Função toggleCameraFacing:**

- ▶ `function toggleCameraFacing() { ... }`
- ▶ Esta função é responsável por mudar o valor da variável de estado facing.
- ▶ Ela utiliza uma função de callback para `setFacing`, garantindo que a atualização do estado seja baseada no valor atual do facing.

- ▶ **Lógica de Alternância:**

- ▶ `setFacing(current => (current === 'back' ? 'front' : 'back'));`



## Exemplo de Função:

```
1  function toggleCameraFacing() {  
2      setFacing(current => (current === 'back' ? 'front' : '  
        back')));  
3  }
```



## API Camera: Estrutura de Renderização

A interface do usuário para a câmera é construída usando o componente `CameraView` e elementos interativos.

- ▶ **Componente `CameraView`:**

- ▶ `<CameraView style={styles.camera} facing={facing}>`
- ▶ Este é o componente que renderiza a pré-visualização da câmera.
- ▶ A propriedade `facing` é vinculada ao estado `facing`, controlando qual câmera é exibida.

- ▶ **Controle de Câmera (Flip Camera):**

- ▶ Um `TouchableOpacity` é aninhado dentro da `CameraView`.
- ▶ Ao ser pressionado (`onPress`), ele chama a função `toggleCameraFacing`, permitindo ao usuário alternar entre as câmeras frontal e traseira.



## Exemplo de Renderização:

```
1  return (  
2    <View style={styles.container}>  
3      <CameraView style={styles.camera} facing={facing}>  
4        <View style={styles.buttonContainer}>  
5          <TouchableOpacity style={styles.button} onPress={  
            toggleCameraFacing>  
6            <Text style={styles.text}>Flip Camera</Text>  
7          </TouchableOpacity>  
8        </View>  
9      </CameraView>  
10    </View>  
11  );  
12 }
```



## Dever de Casa

- ▶ Capturar/Salvar Foto
- ▶ Capturar/Compartilhar Foto



# Navegação No React Native





# Navegação em React Native: Introdução

A navegação é um aspecto crucial de qualquer aplicativo mobile, permitindo que os usuários se movam entre diferentes telas e seções da aplicação.

## ▶ O que é Navegação?

- ▶ O processo de gerenciar e transitar entre as diferentes telas (ou "rotas") de um aplicativo.
- ▶ Essencial para criar uma UX fluida e intuitiva.

## ▶ Por que é Importante?

- ▶ Permite organizar o conteúdo em seções lógicas.
- ▶ Ajuda o usuário a entender onde ele está no aplicativo e como pode se mover.

## ▶ Desafios em React Native:

- ▶ React Native não possui uma solução de navegação embutida e universalmente aceita para todos os casos de uso.
- ▶ É necessário utilizar bibliotecas de terceiros para implementar a navegação.



## ► Instalação Básica (Expo):

```
1 npm install @react-navigation/native
2 npm install @react-navigation/elements
3 npx expo install react-native-screens react-native-
  safe-area-context
4 npm install @react-navigation/native-stack
5 npm install @react-navigation/drawer
6 npm install @react-navigation/bottom-tabs
7 npx expo install react-native-gesture-handler react-
  native-reanimated
```

- Inclui dependências essenciais para otimização de tela e manipulação de área segura.



# Navegação em React Native: React Navigation

**React Navigation** é a solução de navegação mais popular e robusta para aplicativos React Native.

- ▶ **O que é?**

- ▶ Uma biblioteca de navegação extensível e totalmente personalizável.

- ▶ **Vantagens:**

- ▶ **Performance:** Utiliza componentes nativos para transições e animações suaves.
- ▶ **Flexibilidade:** Suporta vários tipos de navegadores (Stack, Tab, Drawer, etc.).
- ▶ **Comunidade Ativa:** Grande suporte da comunidade e documentação abrangente.
- ▶ **Integração com Expo:** Funciona perfeitamente com projetos Expo.



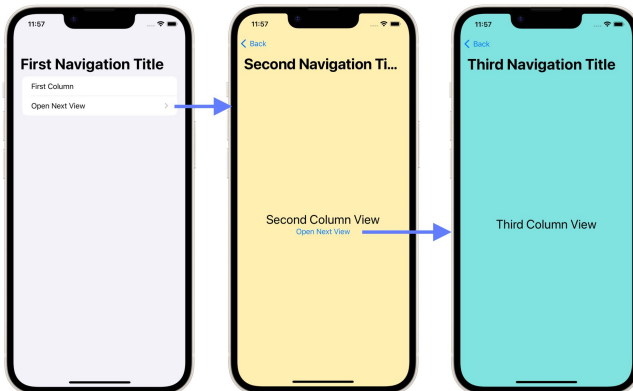
## Navegação em React Native: Tipos de Navegadores

React Navigation oferece diferentes tipos de navegadores para atender a diversas necessidades de UI e fluxo de usuário.

- ▶ **Stack Navigator** (`@react-navigation/stack`):

- ▶ Cria uma pilha de telas, onde cada nova tela é colocada no topo da pilha.
- ▶ Ideal para fluxos lineares, como formulários de várias etapas ou detalhes de itens.
- ▶ Permite navegar para frente e para trás.





## Navegação em React Native: Tipos de Navegadores

React Navigation oferece diferentes tipos de navegadores para atender a diversas necessidades de UI e fluxo de usuário.

- ▶ **Tab Navigator** (`@react-navigation/bottom-tabs` ou `@react-navigation/material-top-tabs`):
  - ▶ Exibe abas (tabs) na parte inferior ou superior da tela.
  - ▶ Ótimo para navegação de nível superior, permitindo que o usuário alterne rapidamente entre seções principais do aplicativo.





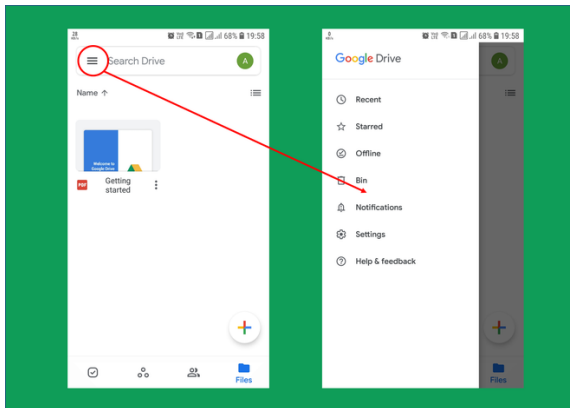
## Navegação em React Native: Tipos de Navegadores

React Navigation oferece diferentes tipos de navegadores para atender a diversas necessidades de UI e fluxo de usuário.




- ▶ **Drawer Navigator** (`@react-navigation/drawer`):
  - ▶ Um menu "gaveta" que desliza da lateral da tela.
  - ▶ Comum para navegação em aplicativos com muitas seções ou configurações.










## Referências I

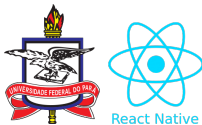
-  Chornobay, B. (2023). *React Native in Action* (2<sup>a</sup> ed.). Shelter Island, NY: Manning Publications.
-  Gandarillas, J. (2023). *Practical React Native: Build cross-platform mobile and web apps with Expo, Hooks, and TypeScript* (2<sup>a</sup> ed.). Birmingham, UK: Packt Publishing.
-  Seemann, R. (2024). *Mastering React Native: Building cross-platform mobile applications with the latest Expo, React Native, and design patterns* (4<sup>a</sup> ed.). Birmingham, UK: Packt Publishing.



## Referências II

-  O'Reilly Media. (2024). *Learning React Native: Build Native Mobile Apps with JavaScript*. (Verifique por edições mais recentes, o livro é atualizado frequentemente).
-  Facebook (Meta Platforms, Inc.). (s.d.). *React Native Documentation*. Recuperado de <https://reactnative.dev/docs>
-  Expo. (s.d.). *Expo Documentation*. Recuperado de <https://docs.expo.dev/>





# Programação Mobile

Professor: Elton Sarmanho<sup>1</sup>

E-mail: [eltonss@ufpa.br](mailto:eltonss@ufpa.br)



<sup>1</sup>Faculdade de Sistemas de Informação - UFPA/CUNTINS

10 de junho de 2025