

Programação para Dispositivos Móveis

Prof^a Letícia Pieper Jandt

80 Horas



Antes da navegação...

O que são hooks?

Hooks permitem que você use diferentes recursos do React a partir dos seus componentes. Você pode usar os Hooks integrados ou combiná-los para criar os seus próprios.

Hooks de estado

O estado permite que um componente “lembre” informações como entrada do usuário.

Por exemplo, um componente de formulário pode usar o estado para armazenar o valor de entrada, enquanto um componente de galeria de imagens pode usar o estado para armazenar o índice de imagem selecionado.

Para adicionar estado a um componente, use um destes Hooks:

`useState` - declara uma variável de estado que você pode atualizar diretamente.

EX: `const [index, setIndex] = useState(0);`

Hooks de contexto

O contexto permite que um componente receba informações de pais distantes sem passá-las como props.

Por exemplo, o componente de nível superior do seu aplicativo pode passar o tema de UI atual para todos os componentes abaixo, não importa quão profundos.

`useContext` - lê e subscreve um contexto.

```
EX: const theme = useContext(ThemeContext);
```

Hooks de referência

Refs permitem que um componente mantenha algumas informações que não são usadas para renderização, como um nó DOM ou um ID de tempo limite.

Diferentemente do estado, atualizar uma ref não renderiza novamente seu componente.

Refs são uma “saída de emergência” do paradigma React. Elas são úteis quando você precisa trabalhar com sistemas que não são React, como as APIs de navegador integradas.

Hooks de referência

`useRef` - declara uma ref. Você pode armazenar qualquer valor nela, mas na maioria das vezes ela é usada para armazenar um nó DOM.

```
const inputRef = useRef(null);
```

Hooks de efeito

Os efeitos permitem que um componente se conecte e sincronize com sistemas externos.

Isso inclui lidar com rede, DOM do navegador, animações, widgets escritos usando uma biblioteca de UI diferente e outros códigos não React.

`useEffect` - conecta um componente a um sistema externo.

Hooks de efeito

```
useEffect(() => {  
  const connection = createConnection(roomId);  
  connection.connect();  
  return () => connection.disconnect();  
}, [roomId]);
```

Efeitos são uma “saída de emergência” do paradigma React. Não use Efeitos para orquestrar o fluxo de dados do seu aplicativo. Se você não estiver interagindo com um sistema externo, talvez não precise de um Efeito.

1 npx create-expo-app aula02 --template

2 Template BLANK

3 abrir no visualcode (digite code . no cmd)

Navegação React-native

Em um navegador da web, você pode vincular a diferentes páginas usando uma `<a>`tag de âncora.

Quando o usuário clica em um link, a URL é empurrada para a pilha do histórico do navegador. Quando o usuário pressiona o botão Voltar, o navegador retira o item do topo da pilha do histórico, então a página ativa agora é a página visitada anteriormente.

Navegação React-native

O React Native não tem uma ideia interna de uma pilha de histórico global como um navegador da web -- é aqui que o React Navigation entra na história.

O stack navigator nativo do React Navigation fornece uma maneira para seu aplicativo fazer a transição entre telas e gerenciar o histórico de navegação.

Navegação React-native

Se seu aplicativo usa apenas um stack navigator, então ele é conceitualmente similar a como um navegador da web lida com o estado de navegação - seu aplicativo empurra e retira itens da pilha de navegação conforme os usuários interagem com ele, e isso faz com que o usuário veja telas diferentes.

React-navigation (<https://reactnavigation.org/docs/getting-started>)

createNativeStackNavigator

é uma função que retorna um objeto contendo 2 propriedades: Screen e Navigator.

Ambos são componentes React usados para configurar o navegador. O Navigator deve conter Screen elementos como seus filhos para definir a configuração para rotas.

React-navigation (<https://reactnavigation.org/docs/getting-started>)

NavigationContainer

é um componente que gerencia nossa árvore de navegação e contém o estado de navegação .

Este componente deve envolver toda a estrutura dos navegadores.
Normalmente, renderizaríamos este componente na raiz do nosso aplicativo, que geralmente é o componente exportado de App.js.

Instalando:

No CMD:

```
npm install @react-navigation/native
```

```
@react-navigation/native-stack
```

```
npm install react-native-screens
```

```
react-native-safe-area-context
```

package.json forma 1

alterar o script do start

```
"start": "expo start --tunnel"
```

depois:

```
npm install @expo/ngrok
```


package.json (forma 2)

rodar:

```
npx expo install react-native-web react-dom  
@expo/metro-runtime
```

depois

```
npx expo start
```

depois de carregar, aperta w

JS App.js > [🔍] default

```
1  import { NavigationContainer } from '@react-navigation/native';
2  import { createNativeStackNavigator } from '@react-navigation/native-stack';
3  import * as React from 'react';
4  import { Text, View } from 'react-native';
5
6  function HomeScreen() {
7    return (
8      <View style={{ flex: 1, alignItems: 'center', justifyContent: 'center' }}>
9        <Text>Home Screen</Text>
10      </View>
11    );
12  }
13
14  const Stack = createNativeStackNavigator();
15
```

app.js

```
15
16 ✓ function App() {
17   ✓ return (
18     ✓ <NavigationContainer>
19     ✓   <Stack.Navigator>
20     ✓     <Stack.Screen name="Home" component={HomeScreen} />
21     ✓   </Stack.Navigator>
22     </NavigationContainer>
23   );
24 }
25
26 | export default App;
```

CONTINUAÇÃO - app.js

VAMOS TESTAR

npm start

vamos ver uma tela com uma barra de navegação vazia e uma área de conteúdo cinza contendo seu `HomeScreen` componente.

Os estilos que você vê para a barra de navegação e a área de conteúdo são a configuração padrão para um `stack navigator`.

Toda a configuração de rota é especificada como props para nosso navegador. Não passamos nenhum props para nosso navegador, então ele usa apenas a configuração padrão.

Vamos adicionar uma segunda tela ao nosso navegador de pilha nativo e configurar a HomeScreen para renderizar primeiro:

```
6 function HomeScreen() {
7   return (
8     <View style={{ flex: 1, alignItems: 'center', justifyContent: 'center' }}>
9       <Text>Home Screen</Text>
10    </View>
11  );
12 }
13
14 function DetailsScreen() {
15   return (
16     <View style={{ flex: 1, alignItems: 'center', justifyContent: 'center' }}>
17       <Text>Details Screen</Text>
18    </View>
19  );
20 }
```

alterar o App.js


```
25 function App() {  
26   return (  
27     <NavigationContainer>  
28       <Stack.Navigator initialRouteName="Home">  
29         <Stack.Screen name="Home" component={HomeScreen} />  
30         <Stack.Screen name="Details" component={DetailsScreen} />  
31       </Stack.Navigator>  
32     </NavigationContainer>  
33   );  
34 }
```

alterar o App.js

Agora nossa stack tem duas rotas , uma Home e uma Details.

Uma rota pode ser especificada usando o Screen componente. O Screen componente aceita um propriedade name que corresponde ao nome da rota que usamos para navegar, e uma componente que corresponde ao componente que ele vai renderizar.

a rota do home corresponde ao `HomeScreen()`, e a rota `Details` corresponde ao `DetailsScreen()`.

A rota inicial para a stack é a Home.

Vamos trocar a rota inicial para `Details` e dar um refresh para ver o resultado.

Navegando entre telas

Se fosse um navegador da web, poderíamos escrever algo assim:

```
<a href="details.html">Detalhar</a>
```

No native em vez de usar âncoras, usaremos o *navigation* que é passado para nossos componentes de tela.

Navegando entre telas

Se fosse um navegador da web, poderíamos escrever algo assim:

```
<a href="details.html">Detalhar</a>
```

No native em vez de usar âncoras, usaremos o *navigation* que é passado para nossos componentes de tela.

Alterar App.js - HomeScreen

```
4 import { Button, Text, View } from 'react-native';
5
6 function HomeScreen({ navigation }) {
7   return (
8     <View style={{ flex: 1, alignItems: 'center', justifyContent: 'center' }}>
9       <Text>Home Screen</Text>
10      <Button
11        title="Ir para detalhes"
12        onPress={() => navigation.navigate('Details')}
13      />
14    </View>
15  );
16 }
17
```

ADICIONE BUTTON AO IMPORT!

Vamos adicionar um component externo?

vamos criar a pasta src

dentro dela pasta: PageCep

vamos salvar o arquivo:

Index.js dentro dela.

Vamos instalar o axios

no cmd:

```
npm install axios
```

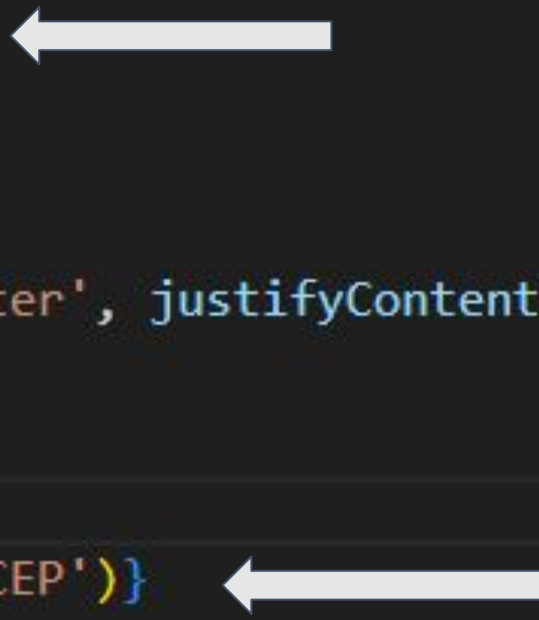
depois de novo

```
npx expo start
```

```
press w
```

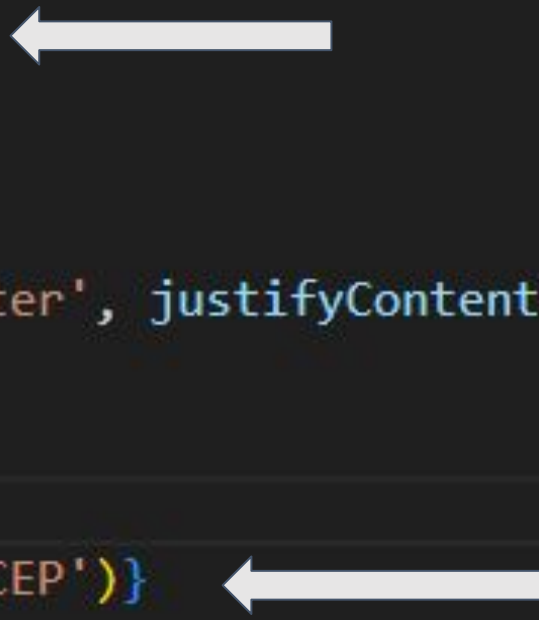
em app.js

```
5  import PageCep from "../src/PageCep/Index";
6
7  function HomeScreen({ navigation }) {
8    return (
9      <View style={{ flex: 1, alignItems: 'center', justifyContent: 'center' }}>
10        <Text>Home Screen</Text>
11        <Button
12          title="Ir para CEP"
13          onPress={() => navigation.navigate('CEP')}
14        />
15      </View>
16    );
17  }
```



em app.js

```
5  import PageCep from "../src/PageCep/Index";
6
7  function HomeScreen({ navigation }) {
8    return (
9      <View style={{ flex: 1, alignItems: 'center', justifyContent: 'center' }}>
10        <Text>Home Screen</Text>
11        <Button
12          title="Ir para CEP"
13          onPress={() => navigation.navigate('CEP')}
14        />
15      </View>
16    );
17  }
```



em app.js

```
function App() {  
  return (  
    <NavigationContainer>  
      <Stack.Navigator initialRouteName="Home">  
        <Stack.Screen name="Home" component={HomeScreen} />  
        <Stack.Screen name="Details" component={DetailsScreen} />  
        <Stack.Screen name="CEP" component={PageCep} /> ←  
      </Stack.Navigator>  
    </NavigationContainer>  
  );  
}
```

vejam como ficou no app.

exercícios

Crie uma nova tela, extilize as telas... e faça a navegação entre elas.