

React Native - Gerenciamento de estado

O que vamos ver hoje?

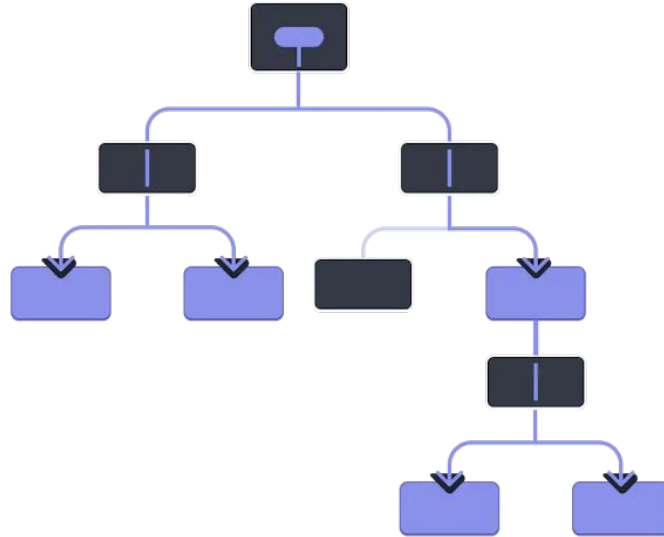
- Bibliotecas de gerenciamento de estado
- Context API

O que é gerenciamento de estado?

Gerenciamento de estado

- Normalmente, quando precisamos passar dados de uma tela para outra simplesmente passamos por props.
- Mas imagine um efeito cascata em que as informações que estão na tela pai precisam ir para um filho 3 ou 4 níveis abaixo. Isso é um baita problema!
 - Isso é chamado de *Prop drilling*

Definição - Prop Drilling



Gerenciamento de estado

- É com esse problema que surgiram as ferramentas para gerenciar o estado da aplicação
- As mais utilizadas são **Redux** e **Context API** (nativa do React/React Native)

Gerenciamento de estado - Redux

- <https://redux.js.org/>
- 4 conceitos principais:
 - **Actions** são funções que alteram algum estado.
 - **Reducers** são funções que recebem um estado e o atualizam com a devida Action.
 - **Store** é onde os estados ficam guardados.
 - **Dispatch** é quem presta atenção em eventos dentro da aplicação

Gerenciamento de estado - Context API

- <https://react.dev/learn/passing-data-deeply-with-context>
- <https://react.dev/reference/react/useContext>
- Nativa do ambiente React/React Native
- Mais simples de começar a usar e mais focada em passar propriedades para componentes sem se preocupar com os níveis de *nesting* (componentes dentro de outros componentes).

Gerenciamento de estado - Resumo

- Utilizamos o **Redux** nos casos em que:
 - Aplicações que precisam ter estados alterados com uma alta frequência;
 - Aplicações com muitos estados para serem controlados;
 - Lógica para a atualização de um estado é muito complexa.
- **Desvantagens do Redux**
 - Questões importantes sobre o Redux são:
 - Implementação requer vários passos;
 - Ferramenta muito complexa para projetos mais simples.

Gerenciamento de estado - Resumo

- Usamos o context em:
 - Aplicações que precisam acessar propriedades globalmente em outros componentes;
- **Desvantagens**
 - O Context API deixa de ser interessante quando:
Muitas propriedades/objetos precisam ser passadas para diversos componentes;
A aplicação tem várias regras de negócios que precisam alterar vários estados de propriedades/objetos.

Gerenciamento de estado - Resumo

- Qual utilizar? Depende!
 - Redux possui muito mais funções mas é complexa de utilizar
 - Context API evoluiu muito nas últimas versões e vem ganhando espaço e é muito mais simples de entender e utilizar
- Neste curso utilizaremos a **Context API**

Primeiros passos

Primeiros Passos

- Não é necessário fazer nenhuma instalação pois é uma ferramenta nativa do React
- Vamos iniciar um novo projeto **com *npx***
create-expo-app --template
- Abrir o projeto no VSCode
- Criar uma pasta src
- Criar uma pasta screens
- Criar duas telas: *Login.tsx* e *Home.tsx* exibindo apenas o nome da página

Primeiros Passos

```
1 import React from "react";
2 import { StyleSheet, Text, View } from "react-native";
3
4 const Login = () => {
5   return (
6     <View style={styles.container}>
7       <Text>Login</Text>
8     </View>
9   );
10 };
11
12 const styles = StyleSheet.create({
13   container: {
14     flex: 1,
15     alignItems: "center",
16     justifyContent: "center",
17   },
18 });
19
20 export default Login;
```

```
1 import React from "react";
2 import { StyleSheet, Text, View } from "react-native";
3
4 const Home = () => {
5   return (
6     <View style={styles.container}>
7       <Text>Home</Text>
8     </View>
9   );
10 };
11
12 const styles = StyleSheet.create({
13   container: {
14     flex: 1,
15     alignItems: "center",
16     justifyContent: "center",
17   },
18 });
19
20 export default Home;
```

Primeiros Passos


- Instale o navigation e o stack navigation
 - `npm install @react-navigation/native`
 - `npx expo install react-native-screens react-native-safe-area-context`
 - `npm install @react-navigation/native-stack`
- Em *src* crie uma pasta *routes* e dentro dela o arquivo *app.routes.tsx*
- Vamos criar a configuração de telas e a tipagem neste arquivo

Primeiros Passos

```
1  import {
2    createNativeStackNavigator,
3    NativeStackNavigationProp,
4  } from "@react-navigation/native-stack";
5  import Login from "../screens/Login";
6  import Home from "../screens/Home";
7
8  type StackRoutesParams = {
9    login: undefined;
10   home: undefined;
11 };
12
13 export type StackNavigatorRoutesProps =
14   NativeStackNavigationProp<StackRoutesParams>;
15
16 const Stack = createNativeStackNavigator<StackRoutesParams>();
17
18 const StackRoutes = () => {
19   return (
20     <Stack.Navigator screenOptions={{ headerShown: false }}>
21       <Stack.Screen name="login" component={Login} />
22       <Stack.Screen name="home" component={Home} />
23     </Stack.Navigator>
24   );
25 };
26
27 export default StackRoutes;
```


Primeiros Passos


- Na pasta *routes* crie um arquivo *index.tsx*. Nele vamos configurar o navigator



```
1 import { NavigationContainer } from "@react-navigation/native";
2 import React from "react";
3 import StackRoutes from "../app.routes";
4
5 export default function Routes() {
6   return (
7     <NavigationContainer>
8       <StackRoutes />
9     </NavigationContainer>
10  );
11 }
```

Primeiros Passos

- No arquivo *App.tsx* vamos inserir nosso arquivo de rotas



```
1  import "react-native-gesture-handler";
2  import Routes from "../src/routes";
3
4  export default function App() {
5    return <Routes />;
6  }
```

Primeiros Passos

- Nas telas *Login* e *Home* vamos adicionar um botão e fazer as configurações do navigation
- Desta vez vamos fazer uso do hook ***useNavigation()***

Primeiros Passos



```
1 import { useNavigation } from "@react-navigation/native";
2 import React from "react";
3 import { Button, StyleSheet, Text, View } from "react-native";
4 import { StackNavigatorRoutesProps } from "../routes/app.routes";
5
6 const Login = () => {
7   const navigation = useNavigation<StackNavigatorRoutesProps>();
8
9   return (
10     <View style={styles.container}>
11       <Text>Login</Text>
12       <Button
13         title="Ir para tela Home"
14         onPress={() => navigation.navigate("home")}
15       />
16     </View>
17   );
18 };
19
20 const styles = StyleSheet.create({
21   container: {
22     flex: 1,
23     alignItems: "center",
24     justifyContent: "center",
25   },
26 });
27
28 export default Login;
```



```
1 import { useNavigation } from "@react-navigation/native";
2 import React from "react";
3 import { Button, StyleSheet, Text, View } from "react-native";
4
5 const Home = () => {
6   const navigation = useNavigation();
7   return (
8     <View style={styles.container}>
9       <Text>Home</Text>
10       <Button
11         title="Voltar para tela de login"
12         onPress={() => navigation.goBack()}
13       />
14     </View>
15   );
16 };
17
18 const styles = StyleSheet.create({
19   container: {
20     flex: 1,
21     alignItems: "center",
22     justifyContent: "center",
23   },
24 });
25
26 export default Home;
27
```

Context API

Configurando Context API

- Chegou a hora de criar nosso contexto
- Na pasta *src* crie uma pasta *contexts*
- Dentro dela crie um arquivo *UserContext.tsx*
- Nesse arquivo vamos configurar nosso contexto com dados do usuário

Configurando Context API



```
1 import { createContext, ReactNode, useState } from "react";
2
3 export type UserContextDataProps = {
4   user: string;
5   signIn: (name: string) => void;
6 };
7
8 type UserContextProviderProps = {
9   children: ReactNode;
10 };
11
12 export const UserContext = createContext<UserContextDataProps>(
13   {} as UserContextDataProps
14 );
15
16 export function UserContextProvider({ children }: UserContextProviderProps) {
17   const [user, setUser] = useState<string>("");
18
19   function signIn(name: string) {
20     setUser(name);
21   }
22
23   return (
24     <UserContext.Provider value={{ user, signIn }}>
25       {children}
26     </UserContext.Provider>
27   );
28 }
```

Configurando Context API

- Na tela *Login* vamos criar um input para receber o nome do usuário e usar o contexto

```
1 import { useNavigation } from "@react-navigation/native";
2 import React, { useContext, useState } from "react";
3 import { Button, StyleSheet, Text, TextInput, View } from "react-native";
4 import { StackNavigatorRoutesProps } from "../routes/app.routes";
5 import { UserContext } from "../contexts/UserContext";
6
7 const Login = () => {
8   const navigation = useNavigation<StackNavigatorRoutesProps>();
9   const { user, signIn } = useContext(UserContext);
10
11   return (
12     <View style={styles.container}>
13       <Text>Login</Text>
14       <TextInput
15         style={styles.input}
16         placeholder="Digite seu nome"
17         onChangeText={signIn}
18         value={user}
19       />
20       <Button
21         title="Ir para tela Home"
22         onPress={() => navigation.navigate("home")}
23       />
24     </View>
25   );
26 };
27
28 const styles = StyleSheet.create({
29   container: {
30     flex: 1,
31     alignItems: "center",
32     justifyContent: "center",
33   },
34   input: {
35     height: 40,
36     margin: 12,
37     borderWidth: 1,
38     padding: 10,
39   },
40 });
41
42 export default Login;
43
```


Configurando Context API

- Na tela *Home* vamos recuperar o valor do contexto e exibir em tela

```
1 import { useNavigation } from "@react-navigation/native";
2 import React, { useContext } from "react";
3 import { Button, StyleSheet, Text, View } from "react-native";
4 import { UserContext } from "../contexts/UserContext";
5
6 const Home = () => {
7   const navigation = useNavigation();
8   const { user } = useContext(UserContext);
9   return (
10     <View style={styles.container}>
11       <Text>{user}</Text>
12       <Button
13         title="Voltar para tela de login"
14         onPress={() => navigation.goBack()}
15       />
16     </View>
17   );
18 };
19
20 const styles = StyleSheet.create({
21   container: {
22     flex: 1,
23     alignItems: "center",
24     justifyContent: "center",
25   },
26 });
27
28 export default Home;
29
```

Configurando Context API

- Agora precisamos envolver nossa aplicação com o *provider* do contexto
- No arquivo *App.tsx* faça a importação do provider e envolva o arquivo de rotas

```
1 import "react-native-gesture-handler";
2 import Routes from "./src/routes";
3 import { UserContextProvider } from "./src/screens/contexts/UserContext";
4
5 export default function App() {
6   return (
7     <UserContextProvider>
8       <Routes />
9     </UserContextProvider>
10  );
11 }
```

Resultado

Resultado

- Rode o projeto com ***npx expo*** e abra o emulador

Resultado

The image displays two side-by-side mobile app screens, each with a light gray background and a dark gray border. The left screen shows the 'Login' screen with a text input field containing 'Rafael' and a blue button labeled 'IR PARA TELA HOME'. The right screen shows the result of the login attempt, with the name 'Rafael' displayed above a blue button labeled 'VOLTAR PARA TELA DE LOGIN'.

Login

IR PARA TELA HOME

Rafael

VOLTAR PARA TELA DE LOGIN

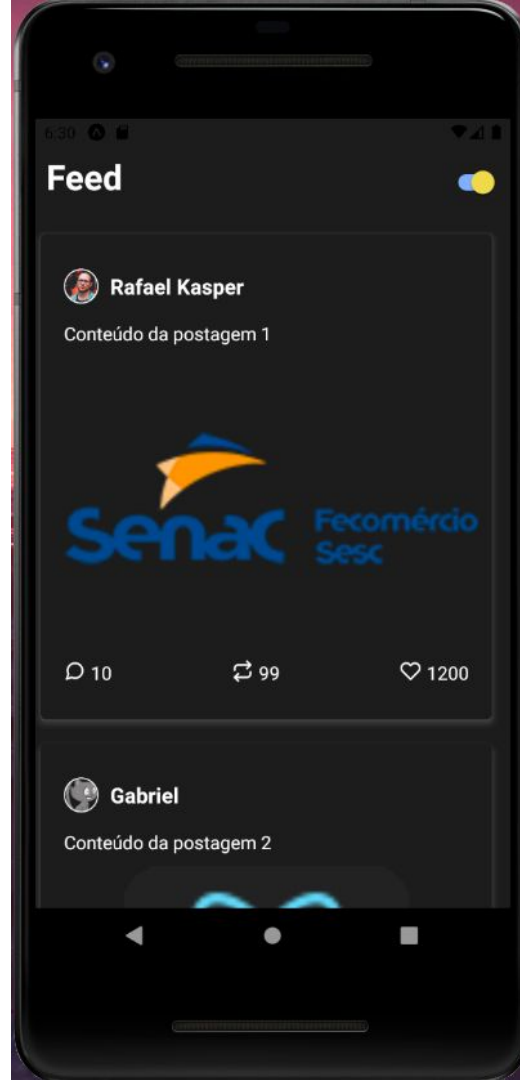
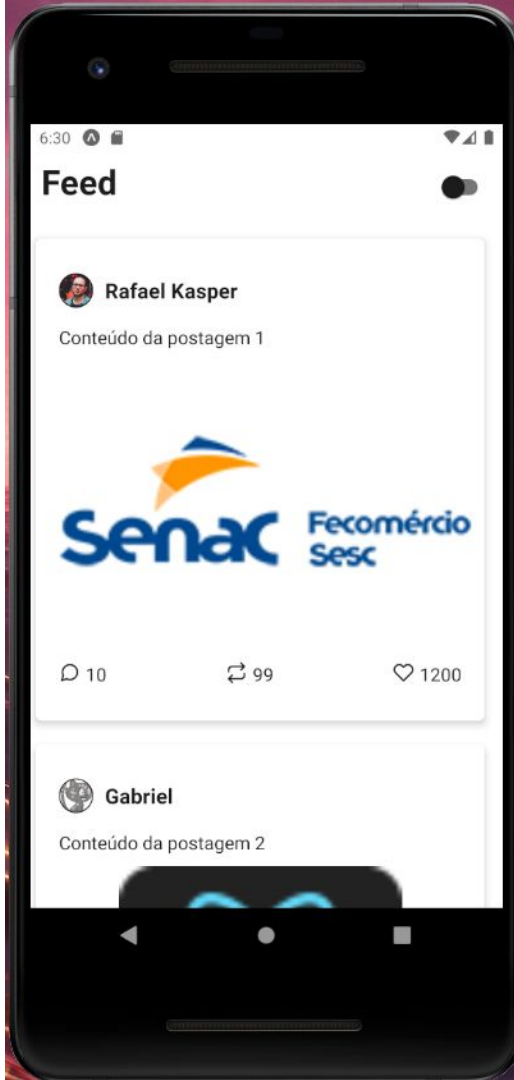
Dúvidas?



Tarefa

- Usando Context API implemente uma lógica de dark/light theme
- Para agilizar, pode aproveitar um projeto criado anteriormente!

Tarefa



Tarefa

▼ contexts

ThemeContext.tsx

```
1 import { createContext, ReactNode, useState } from "react";
2
3 export type ThemeContextDataProps = {
4   theme: string;
5   setTheme: (theme: string) => void;
6 };
7
8 type ThemeContextProviderProps = {
9   children: ReactNode;
10 };
11
12 export const ThemeContext = createContext<ThemeContextDataProps>({
13   } as ThemeContextDataProps
14 );
15
16 export function ThemeContextProvider({ children }: ThemeContextProviderProps) {
17   const [theme, setTheme] = useState<string>("dark");
18
19   return (
20     <ThemeContext.Provider value={{ theme, setTheme }}>
21       {children}
22     </ThemeContext.Provider>
23   );
24 }
```

```
import { ThemeContextProvider } from "../src/contexts/ThemeContext";
import React from "react";
import Home from "../src/screens/Home";

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

```
const { theme } = useContext(ThemeContext);
```

```
color: theme === "light" ? "#1C1C1C" : "#fff",
```

```
const colorScheme = useColorScheme();
```

```
useEffect(() => {
  if (colorScheme === "dark") {
    setIsEnabled(true);
    setTheme("dark");
  } else {
    setIsEnabled(false);
    setTheme("light");
  }
}, []);
```