

React Native - Hooks, Fontes e Async Storage

O que vamos ver hoje?

- useState
- useEffect
- Async Storage

Hooks

Hooks

- Permitem que componentes de função tenham acesso ao estado e outros recursos do React Native.
- Os hooks mais utilizados são o `useState` e o `useEffect`.
- O hook `useCallback` deve ser usado com cuidado.

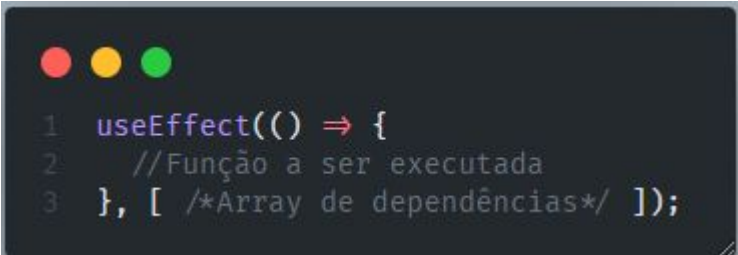
useState

- <https://react.dev/reference/react/useState>
- useState aceita um estado inicial e retorna dois valores:
 - O estado atual
 - Uma função que atualiza o estado
- O estado geralmente se refere aos dados ou propriedades da aplicação que precisam ser rastreados

```
const [text, setText] = useState('');
```

useEffect

- <https://react.dev/reference/react/useEffect>
- Permite que você execute efeitos colaterais no seu código.
- Em sua declaração recebe dois parâmetros:
 - Função a ser executada.
 - Array de dependências (pode ser vazio).



```
1  useEffect(() => {  
2    //Função a ser executada  
3  }, [ /*Array de dependências*/ ]);
```

useEffect

- Caso o array esteja vazio, o useEffect irá disparar somente no carregamento da página.

useCallback

- <https://react.dev/reference/react/useCallback>
- Evita que um componente seja renderizado novamente, a menos que suas props tenham sido alteradas.

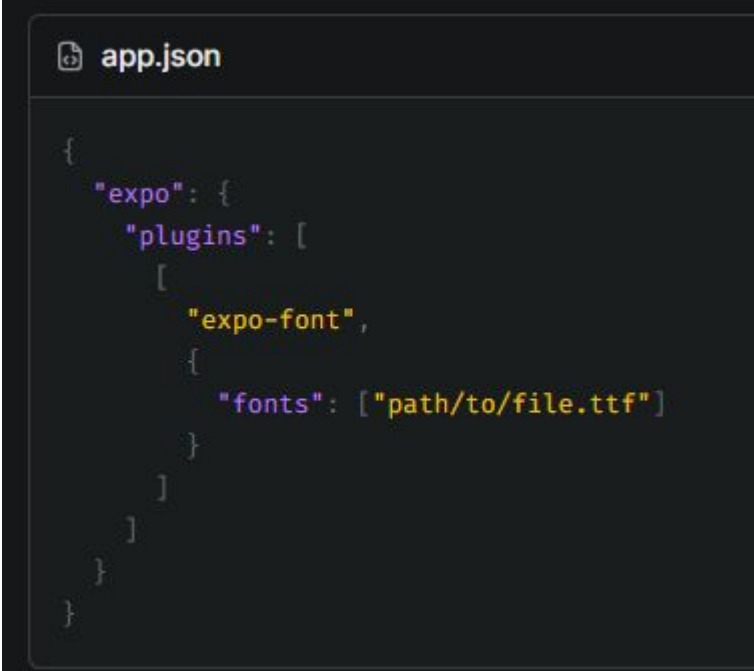
```
import { useCallback } from 'react';

export default function ProductPage({ productId, referrer, theme }) {
  const handleSubmit = useCallback((orderDetails) => {
    post('/product/' + productId + '/buy', {
      referrer,
      orderDetails,
    });
  }, [productId, referrer]);
```


Fontes

Fontes

- <https://docs.expo.dev/versions/latest/sdk/font/#installation>
- `npx expo install expo-font`



```
app.json
{
  "expo": {
    "plugins": [
      "expo-font",
      {
        "fonts": ["path/to/file.ttf"]
      }
    ]
  }
}
```

Fontes

```
import { useCallback } from 'react';
import { Text, View, StyleSheet } from 'react-native';
import { useFonts } from 'expo-font';
import * as SplashScreen from 'expo-splash-screen';

SplashScreen.preventAutoHideAsync();

export default function App() {
  const [fontsLoaded, fontError] = useFonts({
    'Inter-Black': require('./assets/fonts/Inter-Black.otf'),
  });

  const onLayoutRootView = useCallback(async () => {
    if (fontsLoaded || fontError) {
      await SplashScreen.hideAsync();
    }
  }, [fontsLoaded, fontError]);

  if (!fontsLoaded && !fontError) {
    return null;
  }

  return (
    <View style={styles.container} onLayout={onLayoutRootView}>
      <Text style={{ fontFamily: 'Inter-Black', fontSize: 30 }}>Inter Black</Text>
      <Text style={{ fontSize: 30 }}>Platform Default</Text>
    </View>
  );
}
```

Google Fonts

- <https://docs.expo.dev/develop/user-interface/fonts/#use-a-google-font>
- <https://fonts.google.com/>
- `npx expo install expo-font @expo-google-fonts/inter`

Google Fonts

```
import React from 'react';
import { View, Text, StyleSheet } from 'react-native';
import { useFonts, Inter_900Black } from '@expo-google-fonts/inter';

export default function App() {
  let [fontsLoaded, fontError] = useFonts({
    Inter_900Black,
  });

  if (!fontsLoaded || !fontError) {
    return null;
  }

  return (
    <View style={styles.container}>
      <Text style={{ fontFamily: 'Inter_900Black', fontSize: 40 }}>Inter Black</Text>
    </View>
  );
}
```

Async Storage

Async Storage

- Uma API de armazenamento de chave-valor assíncrona, não criptografada e persistente.
- <https://react-native-async-storage.github.io/async-storage/docs/install/>
- `npx expo install @react-native-async-storage/async-storage`

Async Storage - uso

Importing

```
import AsyncStorage from '@react-native-async-storage/async-storage';
```


Async Storage - uso

Storing string value

```
const storeData = async (value) => {  
  try {  
    await AsyncStorage.setItem('my-key', value);  
  } catch (e) {  
    // saving error  
  }  
};
```



Storing object value

```
const storeData = async (value) => {  
  try {  
    const jsonValue = JSON.stringify(value);  
    await AsyncStorage.setItem('my-key', jsonValue);  
  } catch (e) {  
    // saving error  
  }  
};
```



Async Storage - uso

Reading string value

```
const getData = async () => {  
  try {  
    const value = await AsyncStorage.getItem('my-key');  
    if (value !== null) {  
      // value previously stored  
    }  
  } catch (e) {  
    // error reading value  
  }  
};
```



Reading object value

```
const getData = async () => {  
  try {  
    const jsonValue = await AsyncStorage.getItem('my-key');  
    return jsonValue !== null ? JSON.parse(jsonValue) : null;  
  } catch (e) {  
    // error reading value  
  }  
};
```

Dúvidas?



Tarefa

- Crie uma tela de login
 - Inputs de usuário e de senha
 - Botão de confirmação
 - Guarde os valores do usuário e da senha usando o `useState`
 - Crie um `useEffect` sem dependência e exiba um alerta na tela dizendo que o usuário não está logado
 - No `onPress` do botão dispare um alerta mostrando os dados que o usuário digitou e salve os dados em `async Storage`