

React Hooks e Context API

Nesse documento vamos desmistificar esses dois conceitos dentro das aplicações front-end e mobile.

React Hooks


 Exemplo Real

Context API

 Exemplo Real

React Hooks

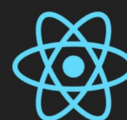
Os **Hooks** são funções que nos permitem manipular os recursos de **estado** e **ciclo de vida** do React a partir de **Componentes Funcionais**.

 **Hooks** são uma nova adição ao React **16.8**. Eles permitem que você use alguns recursos do React sem escrever uma classe (Stateful Component), aliás, eles **não funcionam** dentro de classes.

Hooks de forma resumida - React

Hooks são uma nova adição no React 16.8. Eles permitem que você use o state e outros recursos do React sem

 <https://pt-br.reactjs.org/docs/hooks-overview.html>



Até então, a forma mais tradicional de compartilharmos funcionamento entre componentes era pelos patterns já conhecidos, os HOC's (**Higher-order components**) e as **Render Props**.

Componentes de Ordem Superior - React

Um componente de ordem superior (HOC, do inglês Higher-Order Component) é uma técnica avançada do

 <https://pt-br.reactjs.org/docs/higher-order-compone...>



Render Props - React

O termo "render prop" se refere a uma técnica de compartilhar código entre componentes React passando

 <https://pt-br.reactjs.org/docs/render-props.html>



O grande problema desses padrões é que você precisa modificar boa parte do código do componente para que o mesmo se adapte ao funcionamento compartilhado, aumentando sua **verbosidade** e perdendo boa parte do **isolamento de responsabilidade**, ou seja, você acaba perdendo qual parte do código faz o que.

Além disso, os dois patterns utilizados anteriormente acabavam criando muitos níveis na renderização final do **React** tornando impossível uma visualização da árvore de componentes (DOM) para identificar qual era cada componente.

Existem outras motivações por trás dos **Hooks** mas a maioria gira na complexidade e verbosidade desnecessária em torno dos patterns já existentes principalmente quando os componentes ficam maiores.

Introdução aos Hooks - React

Hooks são uma nova adição ao React 16.8. Eles permitem que você use o state e outros recursos do React sem

 <https://pt-br.reactjs.org/docs/hooks-intro.html#moti...>



Além dos **Hooks** padrão que são disponibilizados pelo **React**, podemos criar nossos próprios **Hooks** de acordo com a necessidade da aplicação, para isso basta extrair a lógica de um componente em funções reutilizáveis.

Criando seus próprios Hooks - React

Hooks são uma nova adição ao React 16.8. Eles permitem que você use o state e outros recursos do React sem

 <https://pt-br.reactjs.org/docs/hooks-custom.html>



Exemplo Real

Imagine uma aplicação que terá que carregar uma lista de registros cadastrados em uma API, por exemplo, uma lista de imóveis, e essa funcionalidade terá o mesmo funcionamento tanto no **ReactJS** quanto no **React Native**.

Em um cenário real, esses dados devem ser carregados apenas quando a tela de listagem for carregada para que a aplicação tenha mais performance, e nesse caso é que vamos utilizar os Hooks **useEffect** para disparar uma função que fará a requisição à API quando a página for carregada e o **useState** para armazenar os dados que vão ser retornados da API, aproveitando assim o ciclo de vida do componente.

Segue um exemplo de Componente que faz exatamente o que foi descrito acima:

```
import React, { useEffect, useState } from 'react'; import api from
'../../services/api'; export default function PropertyList() { const
[properties, setProperties] = useState([]); useEffect(() => { async
function loadProperties() { const { data } = await
api.get('properties'); setProperties(data); } loadProperties(); },
[]); return ( <ul> { !properties.length ? <li>Nenhum imóvel
encontrado.</li> : properties.map(property => ( <li
key={property.id}> <h1>{property.title}</h1>
<p>{property.description}</p> </li> ) </ul> ); }
```

Context API

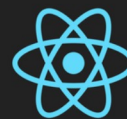
A **Context API** é um conceito que deixou de ser experimental com a chegada da versão **16.3** do React, com ela podemos trabalhar com dados que serão reutilizados em diversos componentes na aplicação, ou seja, temos dados acessíveis a nível global.

Context - React

Contexto (context) disponibiliza uma forma de passar dados entre a árvore de componentes sem precisar passar



<https://pt-br.reactjs.org/docs/context.html>



Até então, qualquer projeto iniciado que fizesse uso do React e precisasse de um gerenciador de estados, automaticamente era pensado no **Redux** ou **MobX**, mas com a **Context API** temos uma nova alternativa, ela permite que seja montado um gerenciador de estado apenas com as funções que são exportadas do próprio React.



Mas reforçando que a **Context API** pode ser utilizada em conjunto com o **Redux** ou **MobX**, o mais importante é analisar as necessidades da aplicação previamente.

Alguns exemplos de funcionalidades que podemos usar a **Context API**:

- Dados de autenticação de um usuário;
- Carrinho de compras;
- Timezone;
- i18n (Internacionalização ou multi-idíomas);
- Configuração de tema.



Exemplo Real

Imagine uma aplicação que terá a funcionalidade de autenticação compartilhada entre o **ReactJS** e o **React Native**, o usuário em ambas aplicações pode se autenticar utilizando email e senha.

Em um cenário real, os dados retornados da autenticação podem ser usados em várias partes da aplicação, e nesse caso vamos utilizar a **Context API** para criar um contexto onde todas as funções de autenticação serão inseridas e os dados serão disponibilizados sem a complexidade de configurar nada externo.

Segue um exemplo de um contexto de Autenticação com as funções de Login (`signIn`) e Logout (`signOut`), e os dados do usuário (`user`) e uma variável para indicar se o usuário está logado (`signed`):

```
import React, { createContext, useState } from 'react'; import * as
auth from '../services/auth'; interface User { name: string; email:
string; } interface AuthContextData { signed: boolean; user: User |
null; signIn(): Promise<void>; signOut(): void; } const AuthContext
= createContext<AuthContextData>({} as AuthContextData); export
const AuthProvider: React.FC = ({ children }) => { const [user,
setUser] = useState<User | null>(null); async function signIn() {
const response = await auth.signIn(); setUser(response.user); }
function signOut() { setUser(null); } return ( <AuthContext.Provider
value={{ signed: !!user, user, signIn, signOut }}> {children}
</AuthContext.Provider> ); };
```

