

FIAP

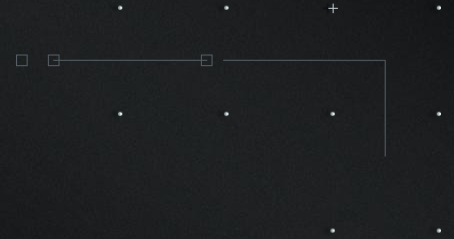
FIAP

SLIDER ▢■◀





React – Hooks - useContext



useContext

O `useContext` é um Hook do React que facilita a utilização de contextos. Contextos no React são uma forma de compartilhar dados entre componentes sem a necessidade de passar propriedades manualmente de pai para filho em cada nível da árvore de componentes.

É uma função nativa do React adicionada na versão 16.8.0, popularmente conhecida como Hook de contexto. É uma funcionalidade recente, logo códigos legados normalmente ainda não a utilizam.

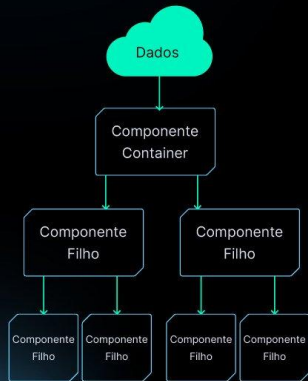
Essa função permite a utilização de contextos de forma mais simplificada e menos verbosa, porém por se tratar de um Hook, seu uso é exclusivo em componentes funcionais.

O “Contexto”, nada mais é do que um componente React que se preocupará única e exclusivamente com a parte lógica, sendo assim os estados, efeitos e até mesmo os “handlers” de eventos, podem ficar agrupados no componente de “Contexto”.

useContext

Transmissão de dados com Context API

Sem Context API



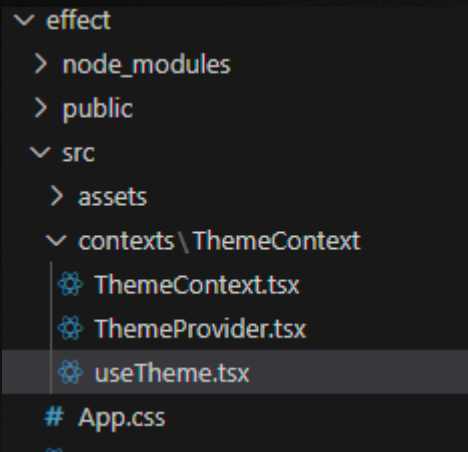
VS

Com Context API



useContext

1. Vamos criar uma pasta chamada de **contexts**, e dentro dela outra pasta chamada de **ThemeContext**
2. Dentro da pasta criada vamos criar 3 arquivos: **ThemeContext.tsx**, **useTheme.tsx** e **ThemeProvider.tsx**



useContext

3. Criando nosso contexto no arquivo ThemeContext.tsx

```
1 // Importa as bibliotecas React, createContext e useContext do pacote 'react'.
2 import { createContext } from "react"
3
4 // ThemeContextType define uma interface para o contexto de tema.
5 // Especificamente, o contexto tem uma propriedade theme (uma string representando o tema)
6 // e um método changeTheme (uma função que não aceita argumentos e não retorna nada).
7 export interface ThemeContextType {
8   theme: string
9   changeTheme: (theme: string) => void
10 } ✨
11
12 // Cria um contexto React chamado ThemeContext usando a função createContext. O contexto é tipado como ThemeContextType | undefined,
13 // indicando que seu valor pode ser do tipo ThemeContextType ou undefined. O valor inicial do contexto é undefined.
14 const ThemeContext = createContext<ThemeContextType | undefined>(undefined)
15
16 // Exporta o contexto ThemeContext.
17 export default ThemeContext
```

useContext

4. Criando nosso Hook no arquivo `useTheme.tsx`

```
1 import { useContext } from "react"
2 // Importa o contexto ThemeContext de '../contexts/ThemeContext'.
3 import ThemeContext from '../ThemeContext'
4
5 // Define uma função chamada useTheme que retorna o contexto ThemeContext.
6 const useTheme = () => {
7   // Utiliza o hook useContext para obter o valor atual do contexto ThemeContext.
8   // O hook useContext é usado em React para acessar o valor de um ThemeContext fornecido como argumento. Neste caso,
9   // ThemeContext é o contexto do tema.
10   const context = useContext(ThemeContext)
11
12   // Verifica se o valor obtido do contexto é false (ou seja, undefined, null, false, 0, etc.).
13   // Isso geralmente significa que o componente que está tentando usar o contexto não está envolvido em um ThemeProvider.
14   if (!context) {
15     // Caso o valor do contexto seja false, lança um erro indicando que a função useTheme
16     // deve ser usada dentro de um componente que envolve um ThemeProvider.
17     // Este é um mecanismo de segurança para garantir que o contexto esteja sendo utilizado corretamente
18     // e que o provedor esteja presente na hierarquia de componentes.
19     throw new Error("useTheme must be used within a ThemeProvider")
20   }
21   return context
22 }
23
24 export default useTheme
25
```

useContext

5. Criando nosso Provider no arquivo ThemeProvider.tsx

```
1  import { PropsWithChildren, useState } from "react"
2  // Importa o contexto ThemeContext de '../ThemeContext'.
3  import ThemeContext from "../ThemeContext"
4
5  // Define uma interface chamada ThemeProviderProps que estende PropsWithChildren para usar o children como propriedade.
6  export interface ThemeProviderProps extends PropsWithChildren { }
7
8  // Define um componente chamado ThemeProvider que recebe children como propriedade.
9  const ThemeProvider = ({ children }: ThemeProviderProps) => {
10   // Define um estado chamado theme com o valor inicial "red" e uma função chamada setTheme para atualizar o estado.
11   const [theme, setTheme] = useState("red")
12
13   // Define uma função chamada changeTheme que recebe um tema como argumento e atualiza o estado theme com o valor do argumento.
14   const changeTheme = (theme: string) => {
15     setTheme(theme)
16   }
17
18   // Retorna o componente ThemeContext.Provider com o valor do estado theme e a função changeTheme.
19   return (
20     <ThemeContext.Provider value={{ theme, changeTheme }}>
21       {children}
22     </ThemeContext.Provider>
23   )
24 }
25
26 export default ThemeProvider
27
```


useContext

Até agora nós criamos 3 arquivos diferentes, que vamos usar pra uma mesma finalidade: Controlar o tema de alguns components da nossa aplicação através do useContext.

ThemeContext

É responsável pela criação do contexto e pela definição da interface do TypeScript

useTheme

É o Hook que vamos usar nos components que quisermos alterar o nosso Contexto

ThemeProvider

Funciona como uma instancia do nosso contexto, podemos ter N providers de um mesmo contexto numa aplicação, cada um com um valor diferente. No nosso caso vamos usar apenas uma instancia englobando nossa aplicação

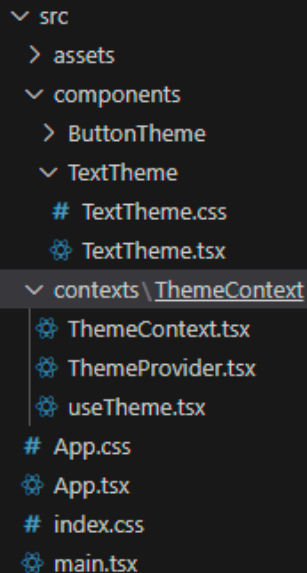
useContext

6. Do jeito que está, nosso código ainda não faz nada de especial. Vamos primeiro colocar o nosso provider no App.tsx. Podemos apagar todo o código que não vamos usar, deixar um texto simples por lá por enquanto

```
1  import './App.css'
2  import ThemeProvider from '../contexts/ThemeContext/ThemeProvider'
3
4  function App() {
5    return (
6      <ThemeProvider>
7        Hello World!
8      </ThemeProvider>
9    )
10 }
11
12 export default App
13
```

useContext

7. Agora dentro da pasta components, vamos criar dois components. Primeiro o TextTheme. Vamos criar uma pasta e dentro da pasta um arquivo TextTheme.tsx e TextTheme.css. Depois vamos criar uma pasta ButtonTheme, e dentro um arquivo ButtonTheme.tsx e ButtonTheme.css. Deve ficar mais ou menos assim:



```

▼ src
  > assets
  ▼ components
    > ButtonTheme
    ▼ TextTheme
      # TextTheme.css
      ⚙ TextTheme.tsx
  ▼ contexts \ ThemeContext
    ⚙ ThemeContext.tsx
    ⚙ ThemeProvider.tsx
    ⚙ useTheme.tsx
  # App.css
  ⚙ App.tsx
  # index.css
  ⚙ main.tsx
```



The screenshot shows a file explorer with the following structure:

- src
 - assets
 - components
 - ButtonTheme
 - TextTheme
 - TextTheme.css
 - TextTheme.tsx
 - contexts \ ThemeContext
 - ThemeContext.tsx
 - ThemeProvider.tsx
 - useTheme.tsx
 - App.css
 - App.tsx
 - index.css
 - main.tsx

useContext

8. Vamos criar nosso componente de texto, que vai ser um paragrafo simples que recebe um texto e usa o nosso Hook useTheme para definir uma classe especifica. A esquerda o tsx e a direita o css

```
1 import { HTMLAttributes, PropsWithChildren } from "react"
2 import useTheme from "../../contexts/ThemeContext/useTheme"
3 import './TextTheme.css'
4
5 // Define uma interface chamada TextThemeProps que estende HTMLAttributes
6 // e PropsWithChildren para usar o children como propriedade.
7 interface TextThemeProps extends HTMLAttributes<HTMLParagraphElement>, PropsWithChildren { }
8
9 const TextTheme = ({ children, ...props }: TextThemeProps) => {
10   // Utiliza o hook useTheme para obter o valor do contexto ThemeContext.
11   const { theme } = useTheme()
12
13   // Retorna um elemento p com as propriedades e estilos definidos.
14   // O valor de theme é adicionado à lista de classes do elemento p.
15   // A classe de css vai ser trabalhada no arquivo TextTheme.css
16   return <p
17     {...props}
18     className={`
19       text-theme
20       ${theme}
21       ${props.className}
22     `}
23   >
24     {children}
25   </p>
26 }
27
28 export default TextTheme
29
```

```
1 .text-theme.red {
2   color:  #ff4d4d;
3 }
4
5 .text-theme.blue {
6   color:  #646cff;
7 }
```

useContext

9. Vamos fazer o mesmo com o nosso ButtonTheme. Mas temos uma diferença aqui: O Nosso botão vai alterar o nosso contexto, através do segundo parâmetro do nosso useTheme, que é uma função.

```
1 import useTheme from '../../contexts/ThemeContext/useTheme'
2 import './ButtonTheme.css'
3
4 const ButtonTheme = () => {
5   // Utiliza o hook useTheme para obter o valor do contexto e a função changeTheme
6   const { theme, changeTheme } = useTheme()
7
8   // Retorna um botão com a classe button-theme e o tema atual.
9   // Quando o botão é clicado, a função changeTheme é chamada com o tema oposto ao atual.
10  // Isso altera o tema do aplicativo de "red" para "blue" e vice-versa.
11  return (
12    <button
13      type='button'
14      className={`
15        button-theme
16        ${theme}
17      `}
18      onClick={() => changeTheme(theme === 'red' ? 'blue' : 'red')}
19    >
20      Change Theme
21    </button>
22  )
23 }
24
25 export default ButtonTheme
26
27
```

```
1 .button-theme {
2   padding: 0.5rem 1rem;
3   border-radius: 8px;
4   border: 1px solid transparent;
5   font-size: 1rem;
6   margin: 0;
7 }
8
9 .button-theme.red {
10   background-color: #ff4d4d;
11   color: #ffffff;
12 }
13
14 .button-theme.blue {
15   background-color: #646cff;
16   color: #ffffff;
17 }
```

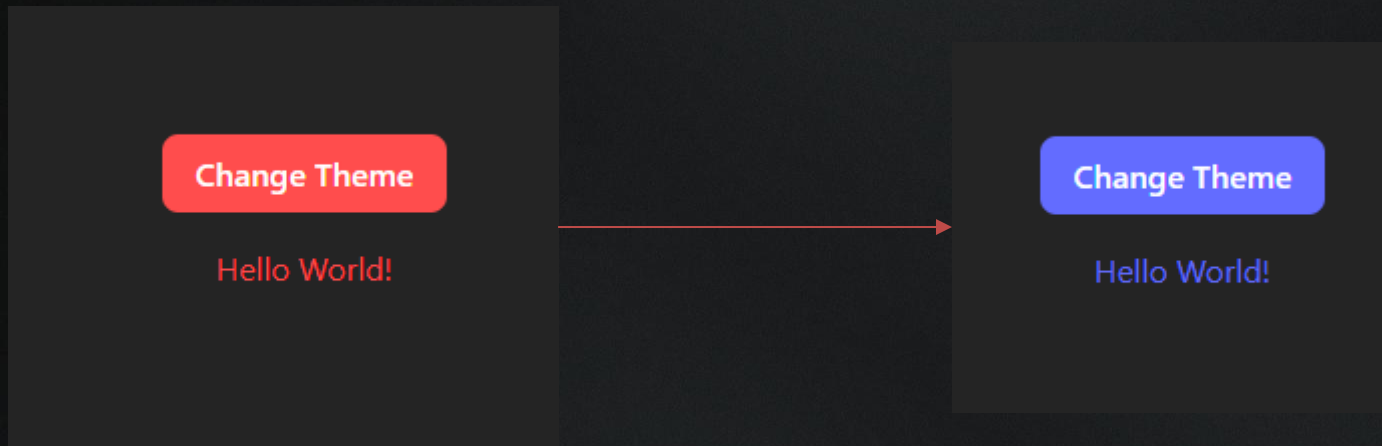

useContext

10. Agora vamos voltar ao App.tsx e adicionar os componentes dentro do Provider

```
1  import './App.css'
2  import ButtonTheme from './components/ButtonTheme/ButtonTheme'
3  import TextTheme from './components/TextTheme/TextTheme'
4  import ThemeProvider from './contexts/ThemeContext/ThemeProvider'
5
6  function App() {
7    return (
8      <ThemeProvider>
9        <ButtonTheme />
10       <TextTheme>
11         Hello World!
12       </TextTheme>
13     </ThemeProvider>
14   )
15 }
16
17 export default App
```

useContext

Ao abrir a aplicação tudo deve estar funcionando ao clicar no botão



useContext

Fluxograma de exemplo

https://miro.com/app/board/uXjVKzCy0fc=/?share_link_id=260324356017

Documentação

<https://react.dev/reference/react/useContext>

Outros Hooks

<https://react.dev/reference/react/hooks>

Exercício

Crie um novo projeto React.

- Implemente um contexto chamado ThemeContext que armazene informações sobre o tema da aplicação.
- O contexto deve conter um objeto com propriedades como backgroundColor e textColor.
- Crie um provedor de contexto chamado ThemeProvider que envolve o conteúdo da aplicação e fornece valores padrão para o tema.
- Implementação do Tema:
 - Escolha cores padrão para backgroundColor e textColor.
 - Utilize o ThemeProvider para fornecer essas informações aos componentes filhos.
- Crie um componente funcional chamado ThemedComponent que utiliza o hook useContext para acessar o contexto ThemeContext.
- Utilize as informações do contexto para estilizar o componente. Por exemplo, use a cor de fundo e a cor do texto fornecidas pelo contexto.
- Crie instâncias de ThemedComponent em diferentes partes da sua aplicação para observar como o tema é aplicado.

Dúvidas, críticas ou sugestões?

FIAP