Front End III

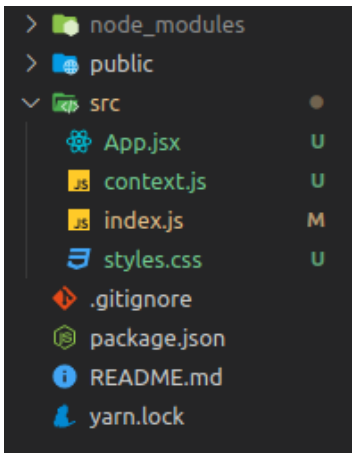
Exemplo de um Theme Context

Vamos realizar o passo a passo para criação de um dark mode e light mode, que são usados pela maioria dos website atualmente.

Passo a passo

Como base, vamos realizar um **create-react-app** para inicializar um projeto pronto para ser usado.

Em seguida, vamos limpar os arquivos, de modo que tenhamos apenas o seguinte:



O nosso App.jsx deveria ser tão simples quanto isso. Lembre-se de limpar dentro de index e app o que não for necessário.

```
import React from 'react'
import "./styles.css"
const App = () => {
 return (
```

```
    <div>
      CONTEXT TUTORIAL
    </div>
  )
}
export default App
```

Vamos criar um arquivo onde criaremos o nosso theme context com o light e dark mode. O arquivo terá o nome de **‘context.js’**.

Ele conta com o objeto com as nossas cores e o createContext para utilizar o nosso projeto.

Com o createContext, podemos fazer uso do componente Provider e o Hook do useContext, que nos permitirá consumir os dados que sejam passados via provider.

```
import React, {createContext} from 'react';
export const themes = {
 light: {
   font: 'black',
   background: 'white'
 },
 dark: {
   font: 'white',
   background: 'black'
 }
};
const ThemeContext =.createContext(themes.light);
export default ThemeContext;
```

➔ Exportamos um objeto com os nossos themes

➔ Exportamos o nosso context com o valor inicial ‘light’

Configurando o nosso ThemeContext

Vamos voltar para a nossa app.jsx. A primeira parte são os imports:

```
import React, { useState } from 'react'
import "./styles.css"
import ThemeContext, { themes } from './context'
```

Trazemos o componente **ThemeContext**, que vem com Key para preencher, por exemplo, o Provider que iremos usar.

Em seguida, dentro do nosso component App, vamos adicionar um state, deixando o theme.light como defaultVale, e criaremos uma função que seja responsável pela troca do theme.

No return, vamos usar o nosso componente **themeContext.Provider**, e damos a ele os valores theme (font e background) e a função para trocar a cor.

O nosso código terá a seguinte aparência:

```
const App = () => {
 const [theme, setTheme] = useState(themes.light);
 const handleChangeTheme = () => {
   if (theme === themes.dark) setTheme(themes.light)
   if (theme === themes.light) setTheme(themes.dark)
 }
 return (
   <ThemeContext.Provider value={{ theme, handleChangeTheme }}>
     <div>
       <h1>CONTEXT TUTORIAL</h1>
```

```
      </div>
    </ThemeContext.Provider>
  )
}
export default App
```

Context API + useState

Neste caso, ao invés de armazenar um dado estático (como apenas um título) no nosso Context, iremos nos apoiar na API de estado para poder implementar um dado dinâmico com até uma função setadora que permita atualizá-lo.

```
//...
const [theme, setTheme] = useState(themes.light);
const handleChangeTheme = () => {
    //...
}
return (
  <ThemeContext.Provider value={{ theme, handleChangeTheme }}>
    //...
  </ThemeContext.Provider>
)}
```

Melhoras na Performance

Sendo que o React renderiza todo o componente novamente toda vez que houver uma mudança de estado, há um risco de ter baixas de otimização por conta dessa implementação, de modo que uma boa prática poderia ser o seu uso juntamente com o hook **useMemo** para memorizar o valor do estado e da sua função setadora.

```
 //...
 const [theme, setTheme] = useState(themes.light);
 const handleChangeTheme = () => {
     //...
 }
 const providerValue =  useMemo(()=>({theme,
handleChangeTheme}),[theme,handleChangeTheme])

 return (
   <ThemeContext.Provider value={providerValue}>
     //...
   </ThemeContext.Provider>
```

Adicionando novos componentes

Vamos adicionar três componentes para dar mais conteúdo para o nosso projeto. Vamos criar uma pasta chamada components, onde criaremos arquivos chamados **Navbar.jsx**, **Body.jsx** e **Layout.jsx**.

O nosso import em App deveria ter a seguinte aparência:

```
 return (
   <ThemeContext.Provider value={providerValue}>
     <Layout>
       <Navbar />
       <Body />
     </Layout>
   </ThemeContext.Provider>
 )
}
export default App
```

Vamos ver como ficariam os códigos nos nossos arquivos

Navbar.jsx

```
import React from 'react'
import "../styles.css"

const Navbar = () => {
   return (
       <div className="navbar">
           <p>Inicio</p>
           <button>THEMED BUTTON</button>
       </div>
   )
}

export default Navbar
```

Body.jsx

```
import React from 'react'

const Body = () => {
   return (
       <div>
            <h1>CONTEXT TUTORIAL</h1>
           <p>Lorem ipsum dolor sit amet consectetur adipisicing elit.
Ratione, quis assumenda culpa distinctio necessitatibus provident
```

```
accusamus quas suscipit atque et officia modi veritatis. Adipisci nulla
temporibus eum. Beatae, quis esse!</p>
        </div>
    )
}

export default Body
```

Layout.jsx

```
import React, {useContext} from 'react'
import ThemeContext from '../context'

const Layout = ({children}) => {
    const {theme} = useContext(ThemeContext);

    return (
        <div style={{background: theme.background, color:theme.font}}>
            {children}
        </div>
    )
}

export default Layout
```

O layout é uma simples div que envolve o nosso projeto onde utilizaremos o nosso context.

Trabalhando no Button

Só resta dar o evento para o botão da nossa navbar, de modo que faça a mudança:

```
import React, {useContext} from 'react'
import "../styles.css"
import ThemeContext from '../context'
const Navbar = () => {
   const {theme, handleChangeTheme} = useContext(ThemeContext)
   return (
       <div className="navbar">
           <p>Inicio</p>
           <button
           onClick={handleChangeTheme}
           style={{background: theme.background, color:theme.font}}>
               Change Theme
               </button>
       </div>
   )}
export default Navbar
```

Final feliz :)

E com isto, já finalizamos! Temos um Theme context para utilizar em toda a nossa app sem a necessidade de passar via props.