REACT: Hooks Avançados (useContext)

DEVinHouse

Parcerias para desenvolver a sua carreira

SENAI

<LAB365>

AGENDA

- Revisão de **useState**
- Hooks Avançados: **useContext**

useState

Revisaremos a utilização do hook useState por meio de um exercício

EXERCÍCIO 06 - Manipulação de formulário

Dentro do **Sidebar**:

- Crie um **estado** que recebe o valor de tip e possa alterá-lo (#exemplo: tip e setTip)
- - O **valor de tip** deve ser um **objeto** cujas propriedades correspondam aos valores de cada input (`titulo`, `descricao`, `categoria`, `linguagem`, `video`)
- Tranforme cada **input em controlado**, atrelando as propriedades de tip
- Crie uma função `**handleCreateTip**` atrelada ao botão que envia os dados do formulário. A princípio, faça o `console.log` dos dados para testar se a função está recebendo os valores de tip corretamente. Então insira alguns dados de exemplo e submeta o formulário para verificar se um objeto com os valores dos campos está sendo impresso corretamente no console.

INTERVALO DE AULA

DEV!

Finalizamos o nosso primeiro período de hoje. Que tal descansar um pouco?!

Nos vemos em 20 minutos.

Início: 20:20
Retorno: 20:40

useContext

Como sabemos, os componentes funcionais nos permitem usar Hooks.

Para facilitar a utilização de contextos foi criado o hook **useContex**

useContext

As recomendações são **as mesmas** do CONTEXT API:

- O uso de estados globais deve ser moderado, pois dificulta a reutilização de componentes
- Sempre que possível, quebre em pequenos contextos, pois quando um valor do contexto é atualizado todos os componentes ligados a ele serão renderizados novamente
- Indicado em casos como:
  - tema
  - idioma
  - se o usuário está autenticado

useContext

Também funciona da mesma maneira que no CONTEXT API
a **criação** do contexto e do provider.

E não esquecer de “abraçar” todos os descentes que consumirão os dados do Provider

```jsx
import { createContext, useState } from "react";

const UsuarioContext = createContext();

const UsuarioProvider = ({ children }) => {
  const [tipoDePlano, setTipoDePlano] = useState("free");

  return (
    <UsuarioContext.Provider
      value={{
        nome: "Rosana Rezende",
        tipoDePlano,
        alteraTipoDePlano: (tipoDePlano) => {
          setTipoDePlano(tipoDePlano);
        },
      }}
    >
      {children}
    </UsuarioContext.Provider>
  );
};
```

```jsx
import { UsuarioProvider } from './context/usuarioContext';

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(
    <UsuarioProvider>
      <App />
    </UsuarioProvider>
);
```

useContext

O que muda é que será necessário importar o hook useContext do React
e retornar uma **função** (custom hook)
que permitirá o uso do contexto em cada componente.

```
import { createContext, useContext, useState } from "react";

const UsuarioContext = createContext();

// Irá abraçar os componentes que irão compartilhar os dados
const UsuarioProvider = ({ children }) => {···
};

// Permite usar os dados dentro de cada componente
const useUsuario = () => {
  return useContext(UsuarioContext);
};

export { UsuarioProvider, useUsuario };
```

useContext

A utilização dos dados do contexto em cada componente fica muito mais fácil.

Basta **importar a função** criada, e utilizar os dados.

```
import { useUsuario } from "./context/usuarioContext";

export default function App() {
  const { tipoDePlano } = useUsuario();

  return (
    <div className="App">
      <Cabecalho />
      {tipoDePlano === "free" ? <Gratuito /> : <Pago />}
    </div>
  );
}
```

EXERCÍCIO 07 - Criando o contexto

Vamos adicionar contexto ao projeto para **manipular o tema da aplicação**
(alternar entre lightMode e darkMode).

- Na pasta `**src**`, adicione uma pasta `**contexts**` e, dentro dela, um arquivo `app-context.js`
- Crie um o contexto **AppContext** (lembre de utilizar o hook createContext)
- Em seguida crie o **AppProvider**, que vai agrupar toda a lógica de manipulação do tema.
  Ele deve retornar o provider do contexto criado. Lembre-se de receber `children` como prop desse componente e, em seguida, renderizar o `children` dentro do provider. Dentro do Provider crie um estado que altera o tema da aplicação e envie as propriedades como `value`.
- Ainda dentro do arquivo app-context.js, crie um custom hook chamado **useAppContext**, que deve retornar o contexto da aplicação.
- Lembre-se de exportar o **useAppContext** e o **AppContext** do arquivo *app-context.jsx*

Conectando o contexto na aplicação:

- No arquivo App.js, adicione o componente **AppProvider** como container do componente Home.
- No componente **Home**, chame o hook `**useAppContext()**`, e use seu retorno para alterar o className da estilização que deseja alterar.
- Lembre-se de criar um botão/ícone que ao ser clicado irá alterar o retorno do tema escolhido.

MATERIAL COMPLEMENTAR

- API de Referência dos Hooks – React
- Context – React

AVALIAÇÃO DOCENTE

O que você está achando das minhas aulas neste conteúdo?

Clique aqui ou escaneie o QRCode ao lado para avaliar minha aula.

Sinta-se à vontade para fornecer uma avaliação sempre que achar necessário.

DEVinHouse

Parcerias para desenvolver a sua carreira

OBRIGADO!

SENAI

<LAB365>