Criar um Hook Customizado de Local Storage

O objetivo é criar um hook, useLocalStorage, que encapsule o acesso ao localStorage. Isso permitirá armazenar e recuperar dados de forma reutilizável em diferentes componentes.

Passos:

1. O hook deve salvar e recuperar valores no localStorage.
2. Ele deve aceitar uma chave e um valor inicial.
3. O hook retorna o valor atual armazenado e uma função para atualizar o valor.

Exemplo de Implementação:

```
import { useState, useEffect } from 'react';

function useLocalStorage(key, initialValue) {
  const [storedValue, setStoredValue] = useState(() => {
    try {
      const item = window.localStorage.getItem(key);
      return item ? JSON.parse(item) : initialValue;
    } catch (error) {
      console.error(error);
      return initialValue;
    }
  });

  const setValue = (value) => {
    try {
      setStoredValue(value);
      window.localStorage.setItem(key, JSON.stringify(value));
    } catch (error) {
      console.error(error);
    }
  };

  return [storedValue, setValue];
}

export default useLocalStorage;
```

Como Usar:

```
import useLocalStorage from './useLocalStorage';

function App() {
  const [name, setName] = useLocalStorage('name', 'Visitante');

  return (
    <div>
      <h1>Bem-vindo, {name}!</h1>
      <input
        type="text"
        value={name}
        onChange={(e) => setName(e.target.value)}
      />
    </div>
  );
}
```

Atividade 2

Compartilhar Tema com Context API

Neste exercício, você criará um contexto para gerenciar o tema da aplicação e aplicar classes CSS com base no tema selecionado.

Passos:

1. Criar um ThemeContext que fornece o tema atual e uma função para alternar entre light e dark.
2. Usar um ThemeProvider para envolver os componentes que precisam do tema.
3. Alterar as classes CSS ou estilos com base no tema.

Exemplo de Implementação:

```
import React, { createContext, useState, useContext } from 'react';

const ThemeContext = createContext();

export const ThemeProvider = ({ children }) => {
  const [theme, setTheme] = useState('light');

  const toggleTheme = () => {
    setTheme((prevTheme) => (prevTheme === 'light' ? 'dark' : 'light'));
  };

  return (
    <ThemeContext.Provider value={{ theme, toggleTheme }}>
      {children}
    </ThemeContext.Provider>
  );
};

export const useTheme = () => useContext(ThemeContext);
```

Como Usar:

1. **No componente principal, envolver a aplicação com `ThemeProvider`.**

```
import React from 'react';
import { ThemeProvider } from './ThemeProvider';
import AppContent from './AppContent';

function App() {
  return (
    <ThemeProvider>
      <AppContent />
    </ThemeProvider>
  );
}

export default App;
```

Aplicar o tema no componente `AppContent`.

```jsx
import React from 'react';
import { useTheme } from './ThemeProvider';

function AppContent() {
  const { theme, toggleTheme } = useTheme();

  return (
    <div className={theme === 'light' ? 'light-theme' : 'dark-theme'}>
      <h1>Olá, mundo!</h1>
      <button onClick={toggleTheme}>Alternar Tema</button>
    </div>
  );
}

export default AppContent;
```

Definir as classes CSS para `light-theme` e `dark-theme` no seu arquivo de estilos:

```css
.light-theme {
  background-color: white;
  color: black;
}

.dark-theme {
  background-color: black;
  color: white;
}
```