TRYBE
Modulo II – Front-end

Bloco 18 – React: Context API (e Hooks)

Definição
Modalidades avançadas do React.
Context API é uma alternativa ao Redux proposta pelo React desde a versão 16.3.0.

1) Context API do React

Propósito
Permite compartilhar estado entre vários componentes em uma árvore de componentes.

Context, Provider e Consumer

Problema: prop-drilling.

Solucionar com context API:

1. createContext
const MyContext = React.createContext();

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

const MyContext = createContext(defaultValue);
```

Retorna um objeto que possui dois componentes como propriedades: Provider e Consumer.

2. MyProvider
Componente class com *<MyContext.Provider>* no rendimento, passando *value* possivelmente contendo estados, functions para atualiza-los.

Passando *{this.props.children}* no rendimento.

```
<MyContext.Provider value={/* algum valor compartilhado */}>
  <MyComponent>
    <MyOtherComponent>
      ...
    </MyOtherComponent>
  <MyComponent>
</MyContext.Provider>
```

O <MyProvider> embala todo o rendimento do App principal, para disponibilizar todos estados em todo lugar dela.

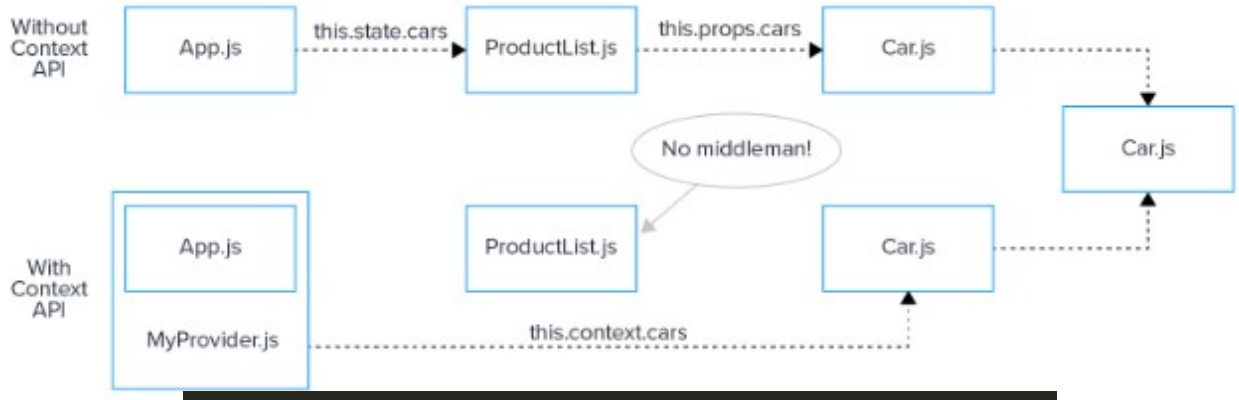
3. MyConsumer
Componente class com <MyContext.Consumer> /> .
Sempre tem como child uma function *{ (context) => () }*; e usa os dados passados. Function arrow ou com binding.

Dentro, usar *<React.Fragment>* para embalar todo dentro de um elemento como fazia com div no React puro.

Default value: se o provider não fornecer um value, por padrão será o passado como parametro do createContext.

```
function MyOtherComponent() {
  return (
    <MyContext.Consumer value={123}>
      {(value) => (
        <Something />
      )}
    </MyContext.Consumer>
  )
}
```

Fluxo dos dados com Context API:




```jsx
import React, { Component } from 'react';

const FamilyContext = React.createContext();

class GreatGrandfather extends Component {
  constructor(props) {
    super(props);
    this.state = {
      inheritance: 1000000,
    }
    this.spendInheritance = this.spendInheritance.bind(this);
  }

  spendInheritance() {
    this.setState((prevState) => ({ inheritance: prevState.inheritance - 1000 }));
  }

  render() {
    const contextValue = {
      inheritance: this.state.inheritance,
      spendInheritance: this.spendInheritance
    };

    return (
      <FamilyContext.Provider value={contextValue}>
        <Grandmother />
      </FamilyContext.Provider>
    );
  }
}
function Grandmother(props) {
  return <Father />;
}

function Father(props) {
  return <Daughter />;
}

function Daughter() {
  return (
    <FamilyContext.Consumer>
      {({ inheritance, spendInheritance }) => (
        <div>
          <span>
            `Tenho uma herança de R$ ${inheritance} que recebi do meu bisavô`
          </span>
          <button type="button" onClick={spendInheritance}/>
        </div>
      )}
    </FamilyContext.Consumer>
  );
}
```

Contexto em componentes de classe

Em componentes feitos com classe e não function, mais uma propriedade é disponível.

```
const MyContext = createContext();

class MyComponent extends React.Component {
  componentDidMount() {
    const value = this.context;
    // ...
  }

  render() {
    const value = this.context;
    // ...
  }
}

MyComponent.contextType = MyContext;
```

contextType
Para acessar *valor* do contexto apartir de *this.context* sem passar por um consumer, em todas etapas do ciclo de vida.
Aceita *apenas um contexto*, precisando de mais, vai ter que construir estrutura de consumer.

Para abilitar ele, no final do arquivo:
MyComponent.contextType = myContext;

Recapitulando o dia:

```
## createContext

- Cria um contexto, para compartilhamento de
informações para uma árvore de componentes.;

- Provider
  Distribui as informações para os arquivos filhos;

- Consumer
  Usado sempre que um componente for consumir um
  valor passado pelo provider;
  Pode ser qualquer tipo de valor;

- contextType
  Fazer com que o componente acesse os valores
  através do this.context;
  Só pode ser utilizado se houver apenas um contexto;
```

<u>Dicas diversas</u>

Ao refatorar de Redux para Context:

- Como fazer requisição de API fora thunk: ver code aula ao vivo 18.1. Resumindo: criando functions para require, success e failure e manipulando estados;

- Sem esquecer de tirar connect do componente para evitar criar conflitos de props;

- No final, tirar toda estrutura e inclusive dependências do Redux.

---

2) React Hooks - useState e useContext

Referência: link.
Propósito: fazer um *componente funcional* simples porém ainda utilizar estados, contextos etc.

Porque React Hooks?

Mais leve e limpo, menos linhas.

Processo de refatorar componente classe para funcional

1. Modificar estrutura básica do componente: linha de class para function, tirar o render para deixar apenas return

2. Gerenciar estados sem constructor via *useState*

- import React, { useState } from "react";
- const[myStateName, myFunctionName] = useState(defaultValueOfState);
- tirar os this.state para deixar apenas nome dos setters dentro do useState
- substituir os setState por chamada do useState com novo parâmetro.

```
const [count, setCount] = useState(0);
```

```
<button onClick={() => setCount(count + 1)}>+</button>
```

useState retorna um array, com 1eiro elemento estado e 2ndo a function para alterar o estado.

3. Gerenciar ciclo de vida via *useEffect*
(mais detalhes debaixo)

4. Ter acesso ao context para centralizar valores simplesmente com useContext
Vantagem: usar facilmente o valor em outros arquivos sem montar todo Consumer.

```
<ThemeContext.Consumer>
  { theme => (
    // Aqui temos o tema
  ) }
</ThemeContext.Consumer>
```

→ equivalente a →

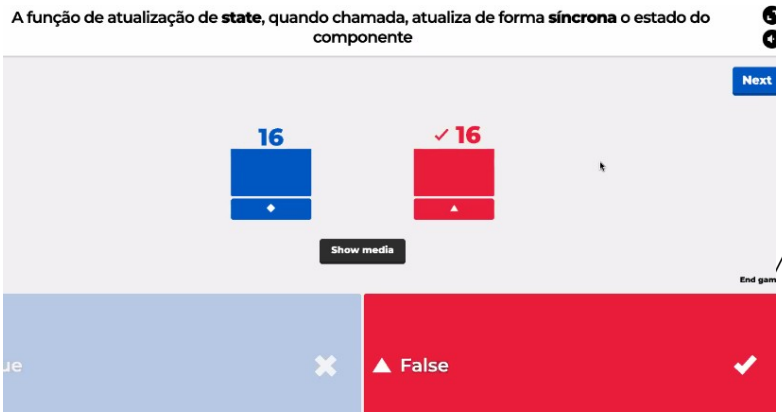
```
const theme = useContext(ThemeContext)
```

Possibilidade: usar hooks como store

Tutorial completo aqui.

Dicas diversas

Kahoot:



// que nem setState

// aprenderemos em que caso usar um ou outro

---

3) useEffect e Hooks customizados

useEffect

Quando disparado, **executa a função** que recebe como **primeiro parâmetro**.

Importar
import React, { useState, useEffect } from “react”;

3 ciclos de vida
useEffect(() => {}, []); → **componentDidMount.**
Se receber um array vazio [], executa a função apenas quando o componente for montado, se o array tiver elementos, executa cada vez que um desses elementos for atualizado.

useEffect(() => {}); → **componentDidUpdate.**
Na ausência de segundo parâmetro, executa a função sempre que o componente é renderizado.

*useEffect(() => { **return () => {}** });* → **componentWillUnmount.**
Retornando uma function, ela é executada quando o componente é desmontado e também antes da próxima renderização.

```
useEffect(() => {
  document.title = `Você clicou ${count} vezes.`
}, [count])
```

Primeiro parâmetro: function sobre o que executar.
Segundo parâmetro: fala em quais situações esse effect deve executar.

Uso
Muito usado em casos de event listeners.

Costumizar hooks

Para quando criou um useEffect que quer **replicar** em outro componente.

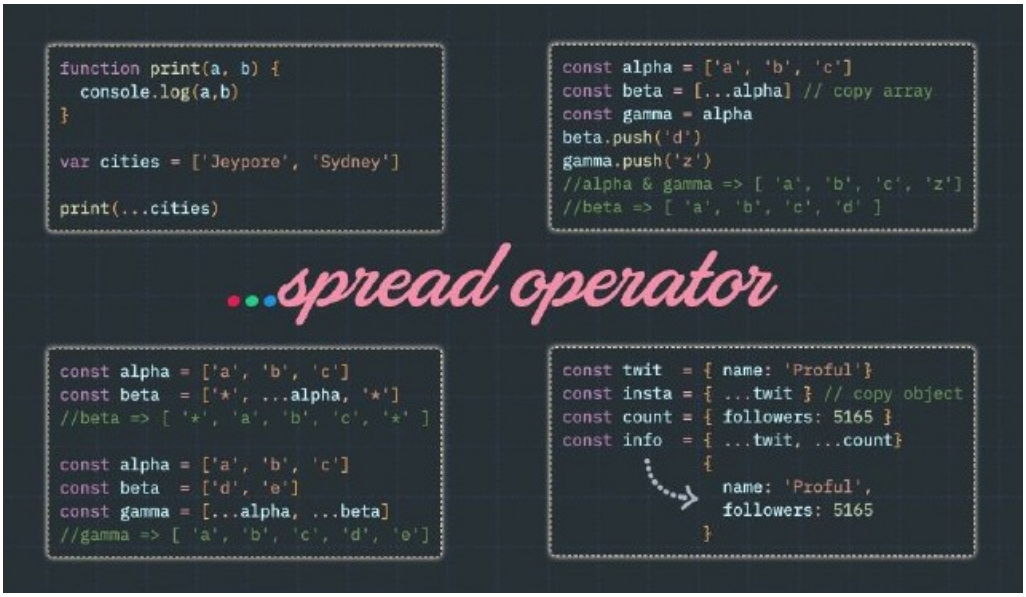
Procurar hooks preexistentes
Primeiro verificar que o hook que quer criar não existe já. Exemplos: 1 – 2 – 3 .

Organizar
Criar pasta hooks, arquivo myHookName.js;
Convenção de nome: inicia com “use”.

O que é hook caseiro
Function auxiliar para exportar e usar onde precisar.
Function que inicia com estado (que vai ser do componente que vai usar esse hook caseiro), aplica para ela um efeito dentro do ciclo de vida pertinente, retorna o estado.

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

const useContextMenuState = defaultValue => {
    const [isContextMenuOpen, setIsContextMenuOpen] = useState(defaultValue);

    useEffect(() => {
        document.oncontextmenu = () => setIsContextMenuOpen(true);
        window.onclick = () => setIsContextMenuOpen(false);

        return () => {
            document.oncontextmenu = () => {};
            window.onclick = () => {};
        };
    }, []);

    return isContextMenuOpen;
};

export default useContextMenuState;
```

Projeto final Star Wars context & hooks, dicas diversas:



Desafio de bug hunters

npm run test-coverage para saber sua cobertura de testes.

Bloco 19 – Projeto App de Receitas

//