REACT: Hooks Avançados (useReducer)

DEVinHouse

Parcerias para desenvolver a sua carreira

SENAI

<LAB365>

AGENDA

- Hooks Avançados: **userReducer**

useReducer

O hook `useReducer` é uma **alternativa ao useState**.

Ele é preferível:

- quando se tem uma lógica complexa, que envolve múltiplos subvalores
  - ou quando o próximo estado depende do estado anterior

useReducer

Assim como no context, é necessário criar um novo arquivo,
que conterá toda a lógica necessária.

No exemplo, criamos uma pasta reducers e dentro dela o arquivo usuarioReducer.js

useReducer

O **resultado final** está na imagem ao lado, e nos próximos slides vamos entender como construir cada parte.

```
import { useReducer } from "react";

const initialState = {
  nomesAdicionais: [
    {
      id: Math.random(),
      nome: "Silas",
    },
  ],
  valorAdicional: 5,
};

const reducer = (state, action) => {
  const { nomesAdicionais, valorAdicional } = state;
  const { type, payload } = action;

  switch (type) {
    case "ADICIONA_PESSOA":
      const { nome, id } = payload;
      return {
        ...state,
        nomesAdicionais: [
          ...nomesAdicionais,
          {
            id: id,
            nome: nome,
          },
        ],
        valorAdicional: valorAdicional + 5,
      };

    default:
      return state;
  }
};

export const useUsuarioReducer = () => {
  return useReducer(reducer, initialState);
};
```

useReducer

O primeiro passo é criar um **estado inicial**, que pode ou não inicializar com valor

```
// é o estado inicial
const initialState = {
  nomesAdicionais:[
    {
      id: 1,
      nome: "Cleiton",
    }
  ],
  valorAdicional: 5,
}
```

useReducer

Em seguida criamos a função reducer, que **recebe** um estado e uma ação, e **retorna** um novo estado dependendo da *ação escolhida*.

Por exemplo, queremos adicionar ou remover uma pessoa da lista.

```
const reducer = (state, action) => {
  const { nomesAdicionais, valorAdicional } = state;
  const { type, payload } = action;

  switch (type) {

    case "ADICIONA_PESSOA":
      const { nome, id } = payload;
      return {
        ...state,
        nomesAdicionais: [
          ...nomesAdicionais,
          {
            id: id,
            nome: nome,
          },
        ],
        valorAdicional: valorAdicional + 5,
      };

    default:
      return state;
  }
};
```

useReducer

Por fim, importamos useReducer do React
e construímos um hook personalizado

```
import { useReducer } from 'react';

const initialState = {···
}

const reducer = (state, action) => {···
}

export const useUsuarioReducer = () => {
  return useReducer(reducer, initialState)
}
```

useReducer

Utilizaremos o reducer no arquivo em que faríamos o useState.

No exemplo, dentro do componente **Compartilhe.js**

- Sem useReducer

```
export const Compartilhe = () => {
  const [pessoaAdicional, setPessoaAdicional] = useState("");

  // ===== sem useReducer
  const [nomesAdicionais, setNomesAdicionais] = useState([
    {
      id: 1,
      nome: "Cleiton",
    },
  ]);
  const [valorAdicional, setValorAdicional] = useState(5);

  const adicionarPessoa = () => {
    setNomesAdicionais([
      ...nomesAdicionais,
      {
        id: Math.random(),
        nome: pessoaAdicional,
      },
    ]);
    setValorAdicional(valorAdicional + 5);

    // limpar o input
    setPessoaAdicional("");
  };

  const removerPessoa = (id) => {
    setNomesAdicionais(
      nomesAdicionais.filter((pessoaAdicional) => pessoaAdicional.id !== id)
    );
    setValorAdicional(valorAdicional - 5);
  };
```

- Com useReducer

```
export const Compartilhe = () => {
  const [pessoaAdicional, setPessoaAdicional] = useState("");

  const [ state, dispatch] = useUsuarioReducer();
  const { nomesAdicionais, valorAdicional } = state;

  const adicionarPessoa = () => {
    dispatch({
      type: 'ADICIONA_PESSOA',
      payload: {
        nome: pessoaAdicional,
        id: Math.random()
      },
    });
    setPessoaAdicional("");
  }

  const removerPessoa = (id) => {
    dispatch({
      type: 'REMOVE_PESSOA',
      payload: {
        idPessoa: id
      },
    });
  }
```

useReducer

Vamos entender cada parte

- estado inicial:

```
// // ===== sem useReducer
const [nomesAdicionais, setNomesAdicionais] = useState([
  {
    id: 1,
    nome: "Cleiton",
  }
]);
const [valorAdicional, setValorAdicional] = useState(5)
```

```
// // ===== com useReducer
const [ state, dispatch] = useUsuarioReducer();
const { nomesAdicionais, valorAdicional } = state;
```

useReducer

- adicionar pessoa

```
// // sem useReducer
const adicionarPessoa = () => {
  setNomesAdicionais([
    ...nomesAdicionais,
    {
      id: nomesAdicionais.length + 1,
      nome: pessoaAdicional,
    },
  ]);
  setValorAdicional(valorAdicional + 5);

  // limpar o input
  setPessoaAdicional("");
}
```

```
// // // com useReducer
const adicionarPessoa = () => {
  dispatch({
    type: 'ADICIONA_PESSOA',
    payload: {
      nome: pessoaAdicional,
      id: Math.random()
    },
  });
  setPessoaAdicional("");
}
```

useReducer

- remover pessoa

```
// // sem useReducer
const removerPessoa = (id) => {
  setNomesAdicionais(nomesAdicionais.filter((adicional) => adicional.id !== id));
  setValorAdicional(valorAdicional - 5);
}
```

```
// com useReducer
const removerPessoa = (id) => {
  dispatch({
    type: 'REMOVE_PESSOA',
    payload: id,
  });
}
```

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

EXERCÍCIO 08 - Criando o reducer

Iremos implementar a lógica do nosso projeto. Para isso, utilizaremos **reducer** para distribuir os dados e as funções entre os componentes de maneira centralizada.

- Na pasta `src`, adicione uma pasta `reducers` e, dentro dela, um arquivo `app-reducer.js`
- Crie um **estado inicial**, que pode estar vazio ou ser um objeto que contém como um dos parâmetros, por exemplo, uma lista contendo um tip (isso facilitará visualizar a renderização do componente).
- Crie uma função **reducer**, que recebe o estado atual e uma ação, e devolve um novo estado para cada tipo de ação. O reducer conterá toda a lógica da aplicação.
- Crie uma função **useAppReducer** que retornará o **useReducer** (importado do React) com os parâmetros reducer e estado inicial.

Crie os seguintes tipos de ação:

- '**CREATE_TIP**': traga a lógica implementada dentro da Sidebar com a função **handleCreateTip**
- '**FILTER_TIPS**': traga a lógica criada na semana 6 e adapte-a ao reducer (dica: utilize setFilter para setar o valor do estado do filtro)

Retorne a **SideBar** para substituir o conteúdo da função handleCreateTip.
Faça as alterações necessárias também no componente **Filter**

MATERIAL COMPLEMENTAR

- API de Referência dos Hooks – 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>