Gerenciamento de Estados com Redux

UFAM

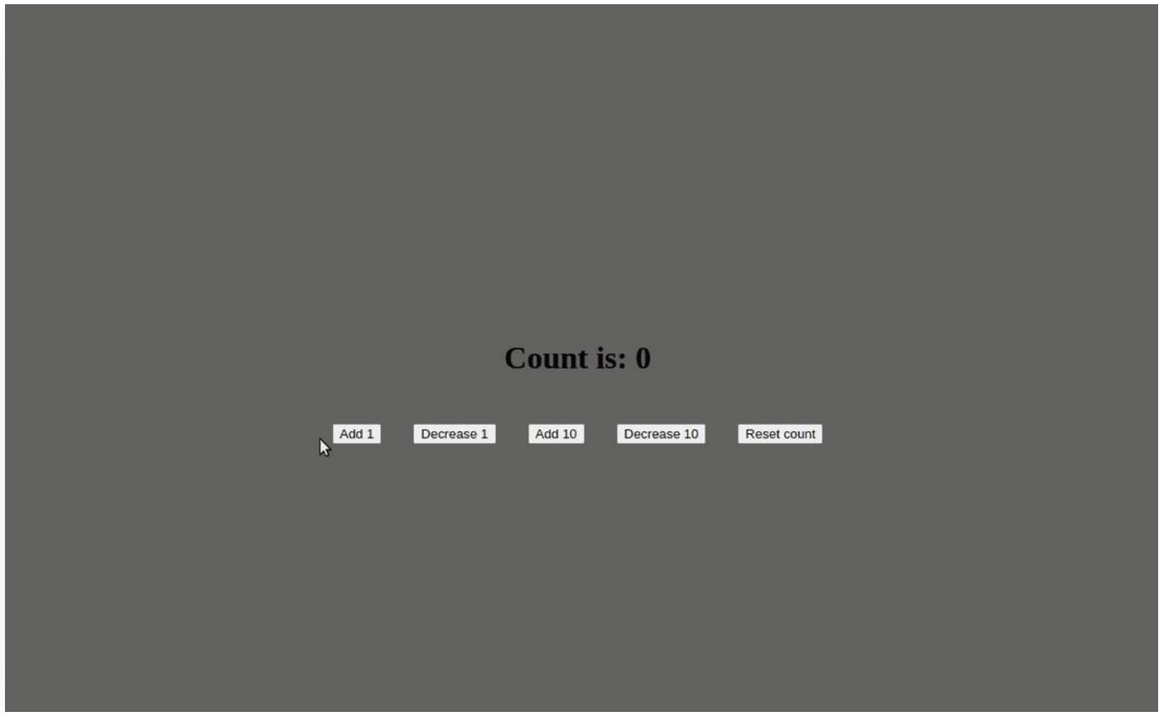
Capacitação em Desenvolvimento Web Full Stack

Douglas Silva de Melo

Gerenciamento de Estados

- O estado é um objeto que contém informações sobre um determinado componente.
- Para implementar o estado em nossos componentes, o React nos fornece um gancho chamado useState.

```jsx
// App.js
import { useState } from 'react'

function App() {

  const [count, setCount] = useState(0)

  return (
    <div className="App">
      <p>Count is: {count}</p>

      <div>
        <button onClick={() => setCount(count+1)}>Add 1</button>
        <button onClick={() => setCount(count-1)}>Decrease 1</button>

        <button onClick={() => setCount(count+10)}>Add 10</button>
        <button onClick={() => setCount(count-10)}>Decrease 10</button>

        <button onClick={() => setCount(0)}>Reset count</button>
      </div>
    </div>
  )
}

export default App
```



E quando temos um projeto maior?

- Misturar Interface com a Lógica
- Dificuldades na Manutenção e no Debugging:
  - Problemas para rastrear e solucionar bugs relacionados ao estado.
  - Agrupar
- Problemas de Escalabilidade:
  - Dificuldades em manter e expandir a arquitetura de estado conforme a aplicação cresce.
- Complexidade no Compartilhamento de Estado:
  - Desafios ao compartilhar estado entre componentes distantes na árvore de componentes.
- Performance Subotimizada:
  - Renderizações desnecessárias e uso ineficiente de recursos, afetando a performance.
- Problemas de Reutilização de Lógica de Estado:
  - Dificuldades em reutilizar lógica de estado entre diferentes componentes.
- Dificuldades com Estados Globais ou Compartilhados:
  - Gerenciamento complicado de estados que precisam ser acessados por múltiplos componentes.
- Problemas de Refatoração:
  - Refatorar a aplicação se torna mais complexo sem um gerenciamento de estado consistente.
- Os **redutores** podem resolver este problema.

E quando temos um projeto maior?

- Muitos componentes compartilham o **mesmo estado** (Toda vez que muda o estado em um, tem que renderizar todos).
- Todos os componentes são renderizados novamente.

```jsx
// App.js
import { useReducer } from 'react'
import './App.scss'

function App() {

  function reducer(state, action) {
    switch (action.type) {
      case 'ADD': return { count: state.count + 1 }
      case 'SUB': return { count: state.count - 1 }
      case 'ADD10': return { count: state.count + 10 }
      case 'SUB10': return { count: state.count - 10 }
      case 'RESET': return { count: 0 }
      default: return state
    }
  }

  const [state, dispatch] = useReducer(reducer, { count: 0 })

  return (
    <div className="App">
      <p>Count is: {state.count}</p>

      <div>
        <button onClick={() => dispatch({type: 'ADD'})}>Add 1</button>

        <button onClick={() => dispatch({type: 'SUB'})}>Decrease 1</button>

        <button onClick={() => dispatch({type: 'ADD10'})}>Add 10</button>
        <button onClick={() => dispatch({type: 'SUB10'})}>Decrease 10</button>

        <button onClick={() => dispatch({type: 'RESET'})}>Reset count</button>
      </div>
    </div>
  )
}

export default App
```

Um redutor(*reducer*), que é a função que irá consolidar todas as mudanças de estado possíveis

Uma função de envio (*dispach*), que enviará as ações de modificação para o redutor.

Torna o gerenciamento de estado mais modular e previsível.

E o Redux?

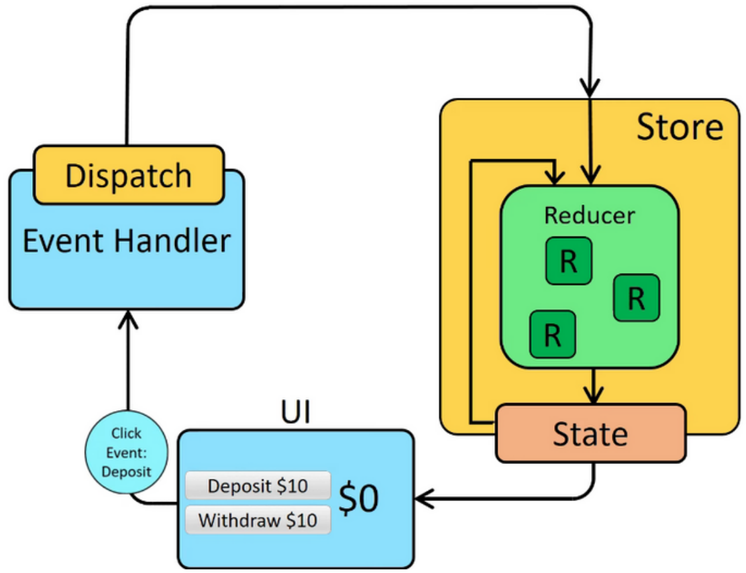
- Redux serve para gerenciar o estado de um aplicativo de forma centralizada, previsível e escalável.
- Ele oferece uma abordagem para armazenar e controlar o estado global do aplicativo.
- Uma biblioteca agnóstica, o que significa que pode ser implementada em qualquer aplicativo front-end, não apenas no React.

E o Redux?

- O conjunto de ferramentas Redux é muito semelhante ao que acabamos de ver com useReducer, mas com mais algumas coisas. Existem três blocos de construção principais no Redux:
  - **Uma loja** - um objeto que contém os dados de estado do aplicativo
  - **Um redutor** - uma função que retorna alguns dados de estado, acionados por um tipo de ação
  - **Uma ação** - um objeto que informa ao redutor como alterar o estado. Ele deve conter uma propriedade de tipo.

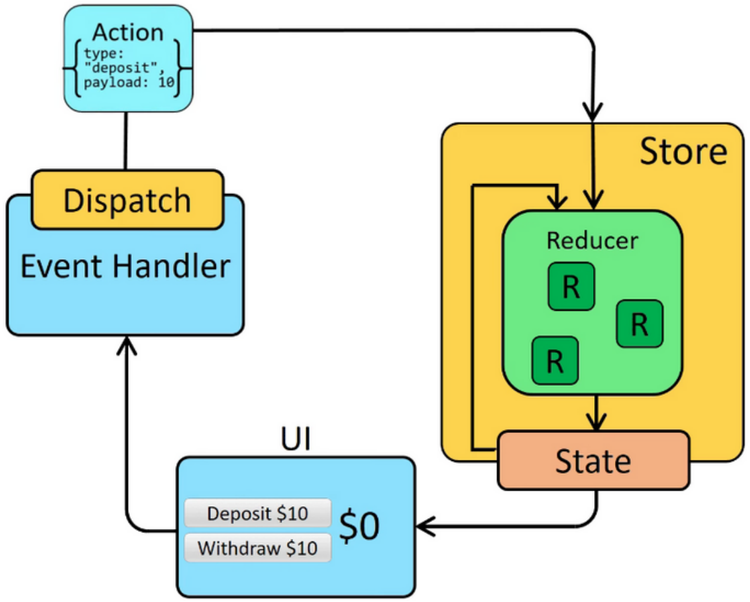
E o Redux?

- O conjunto de ferramentas Redux é muito semelhante ao que acabamos de ver com useReducer, mas com mais algumas coisas. Existem três blocos de construção principais no Redux:
  - **Uma loja** - um objeto que contém os dados de estado do aplicativo
  - **Um redutor** - uma função que retorna alguns dados de estado, acionados por um tipo de ação
  - **Uma ação** - um objeto que informa ao redutor como alterar o estado. Ele deve conter uma propriedade de tipo.



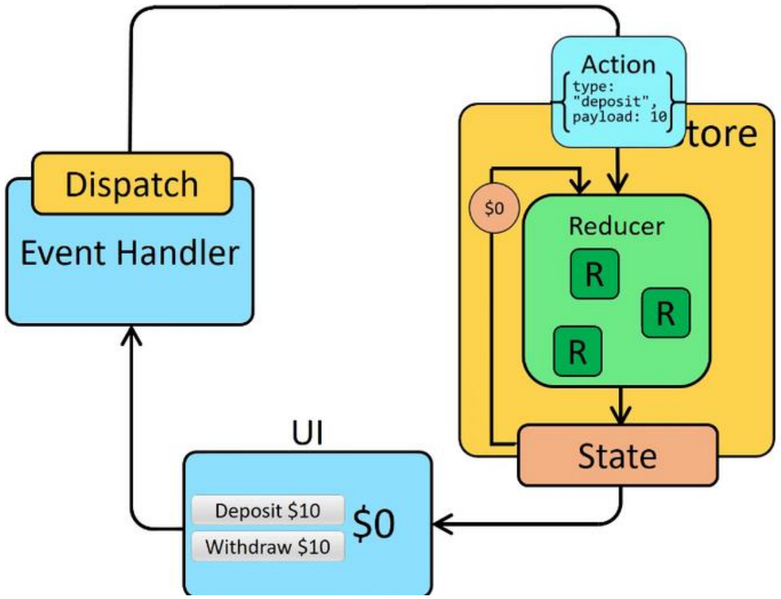
E o Redux?

- O conjunto de ferramentas Redux é muito semelhante ao que acabamos de ver com useReducer, mas com mais algumas coisas. Existem três blocos de construção principais no Redux:
  - **Uma loja** - um objeto que contém os dados de estado do aplicativo
  - **Um redutor** - uma função que retorna alguns dados de estado, acionados por um tipo de ação
  - **Uma ação** - um objeto que informa ao redutor como alterar o estado. Ele deve conter uma propriedade de tipo.



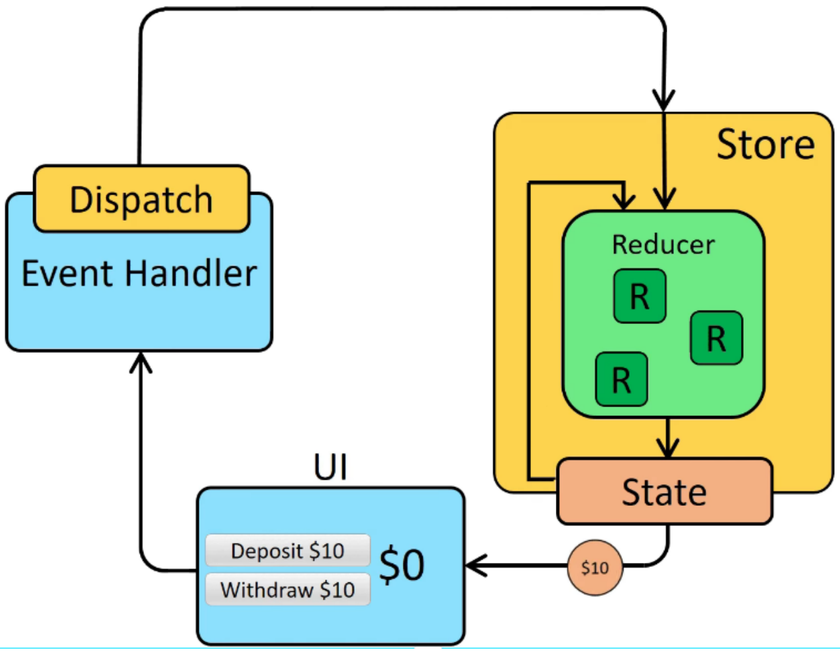
E o Redux?

- O conjunto de ferramentas Redux é muito semelhante ao que acabamos de ver com useReducer, mas com mais algumas coisas. Existem três blocos de construção principais no Redux:
  - **Uma loja** - um objeto que contém os dados de estado do aplicativo
  - **Um redutor** - uma função que retorna alguns dados de estado, acionados por um tipo de ação
  - **Uma ação** - um objeto que informa ao redutor como alterar o estado. Ele deve conter uma propriedade de tipo.

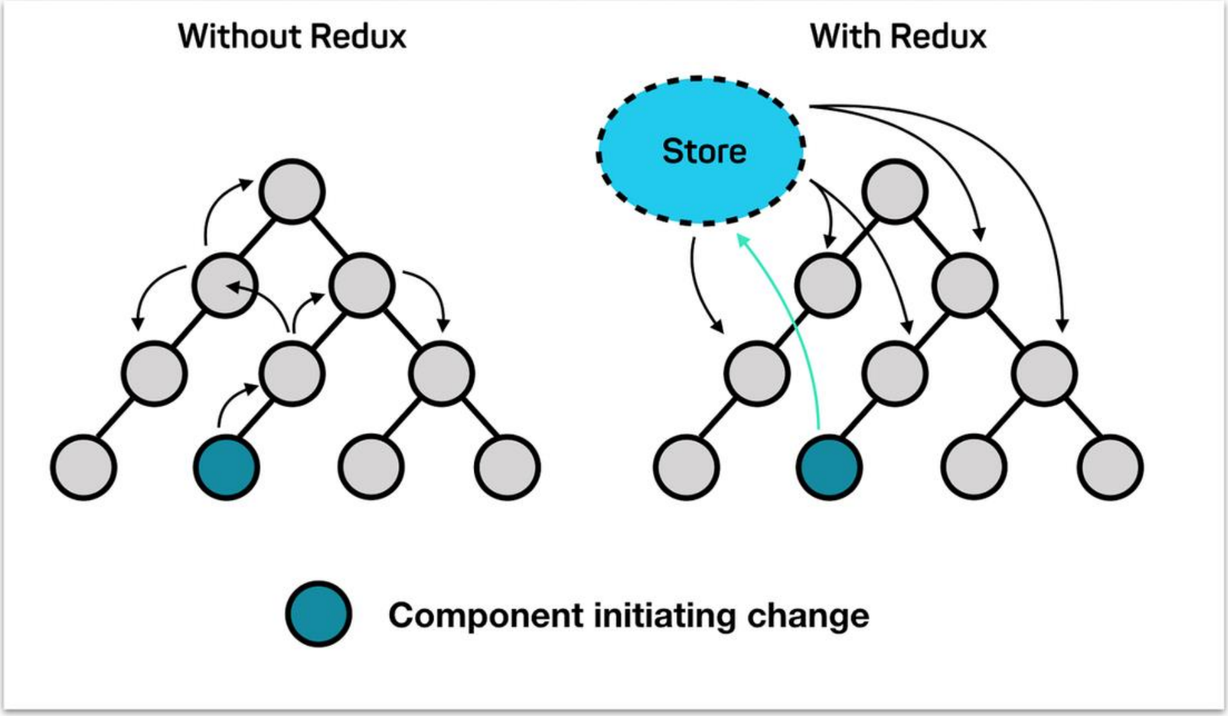


E o Redux?

- O conjunto de ferramentas Redux é muito semelhante ao que acabamos de ver com useReducer, mas com mais algumas coisas. Existem três blocos de construção principais no Redux:
  - **Uma loja** - um objeto que contém os dados de estado do aplicativo
  - **Um redutor** - uma função que retorna alguns dados de estado, acionados por um tipo de ação
  - **Uma ação** - um objeto que informa ao redutor como alterar o estado. Ele deve conter uma propriedade de tipo.



Qual é a diferença entre useState e Redux?



Vamos implementar o Redux

Criando o Counter Slice

src/redux/slices/count.slice.ts

```ts
import { createSlice } from "@reduxjs/toolkit";

export const countSlice = createSlice({
  name: "countSlice",
  initialState: {
    value: 0,
  },

  reducers: {
    increment(state) {
      state.value += 1;
    },
  },
});

export const { increment } = countSlice.actions;
export default countSlice.reducer;
```

Criar do Zero

```
npm create vite@latest
```

```
√ Project name: ... projeto-teste
√ Select a framework: » React
√ Select a variant: » TypeScript
```

```
cd projeto-teste
npm install
npm install react-redux @reduxjs/toolkit
npm run dev
```

Criando o Store

src/redux/store.ts

```ts
import { configureStore } from "@reduxjs/toolkit";
import countReducer from "./slices/count.slice";

export const store = configureStore({
  reducer: {
    count: countReducer,
  },
});

export type RootState = ReturnType<typeof store.getState>;
export type AppDispatch = typeof store.dispatch;
```

Importando o Store

index.tsx

```tsx
import React from "react";
import ReactDOM from "react-dom/client";
import "./index.css";
import App from "./App";
import reportWebVitals from "./reportWebVitals";
import { Provider } from "react-redux";
import { store } from "./redux/store";

const root = ReactDOM.createRoot(
  document.getElementById("root") as HTMLElement
);
root.render(
  <React.StrictMode>
    <Provider store={store}>
      <App />
    </Provider>
  </React.StrictMode>
);

reportWebVitals();
```

Utilizando o Redux

App.tsx

```tsx
import "./App.css";
import { useDispatch, useSelector } from "react-redux";
import { increment } from "./redux/slices/count.slice";
import { RootState } from "./redux/store";

function App() {
  const dispatch = useDispatch();
  const count = useSelector((state: RootState) => state.count);

  return (
    <div className="App">
      <p>{count.value}</p>
      <button onClick={() => dispatch(increment())}> +1 </button>
    </div>
  );
}

export default App;
```