Arquitetura Redux

João Ferreira, Rafaela Cavalcanti, Robério Pereira, Mickeliny Sena

Como surgiu?

A arquitetura Redux surgiu em 2015, criada por Dan Abramov e Andrew Clark para gerenciar o estado de aplicações JavaScript, especialmente com React.

Qual seu propósito?

O propósito principal do Redux é gerenciar o estado de forma clara e previsível, especialmente em aplicações complexas.

- Previsibilidade do estado
- Imutabilidade
- Desacopla a lógica de estado da interface
- Fluxo de dados unidirecional
- Facilita a escrita de testes

Problemas que o Redux resolve:

Gerenciamento centralizado do estado global

Em aplicações React, o gerenciamento de estado em componentes pode se tornar confuso quando o estado precisa ser compartilhado entre componentes que estão distantes na árvore de componentes.

Unidirecionalidade do fluxo de dados

Em arquiteturas de gerenciamento de estado bidirecionais, é fácil perder o controle de como e quando o estado está sendo modificado.

Histórico e Time Travel Debugging

Em sistemas complexos, pode ser difícil saber exatamente como o estado chegou ao seu valor atual, dificultando a depuração.

Testabilidade

Em aplicações com estados complexos e interdependências entre componentes, o teste pode se tornar complicado.

Sincronização de Estado em Ambientes Complexos

Em aplicações que envolvem vários ambientes, manter o estado da aplicação sincronizado em todas essas fontes de dados pode ser desafiador.

Previsibilidade e Rastreabilidade:

Com Redux, as mudanças de estado são previsíveis e rastreáveis, pois seguem um padrão unidirecional de fluxo de dados.

Facilita Testes:

Como as mudanças de estado são feitas através de actions puras, é mais fácil testar as ações (actions) e os reducers isoladamente.

Gerenciamento de Estado Complexo:

Redux é particularmente útil em aplicações com estados complexos e muitos componentes interativos, onde o levantamento do estado para os componentes pais pode se tornar complicado sem uma gestão adequada.

Problemas existentes:

Complexidade inicial e sobrecarga de código

A configuração inicial de Redux pode ser pesada, especialmente para novos desenvolvedores.

Desempenho em grandes aplicações

Em aplicações com muitos componentes e atualizações frequentes de estado, o Redux pode causar problemas de desempenho se não for otimizado adequadamente.

Escalabilidade e manutenção

À medida que a aplicação cresce, o gerenciamento do store e a organização de reducers pode se tornar desafiador.

Boilerplate

Redux gera muito código "boilerplate" (ou repetitivo). Embora o Redux Toolkit tenha ajudado a simplificar e reduzir o número de arquivos e linhas de código necessários

Overengineering para aplicações simples

Muitas vezes, Redux é usado em aplicações pequenas onde o gerenciamento centralizado de estado não é realmente necessário.

Funcionamento da arquitetura

O Redux segue um fluxo unidirecional de dados, o que significa que o ciclo de atualização de estado segue uma única direção, tornando mais fácil rastrear como os dados fluem dentro da aplicação.

Store

O estado global da aplicação é armazenado em um único store.

Actions

As ações são objetos simples que descrevem o que deve mudar no estado da aplicação.

Dispatch

Quando um evento ocorre na interface, o componente dispara uma action para o store.

Reducer

O reducer é uma função que recebe o estado atual e a action disparada, e calcula o novo estado da aplicação com base na ação.

State Update

O store atualiza o estado com base no que o reducer retornou.

Subscription

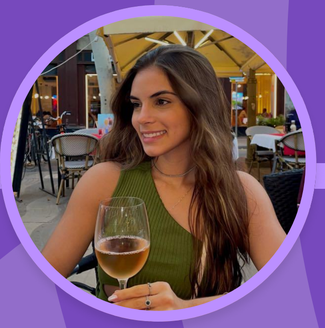
Os componentes da aplicação que precisam de partes do estado são atualizados automaticamente sempre que o estado global é alterado.

Fluxo Completo

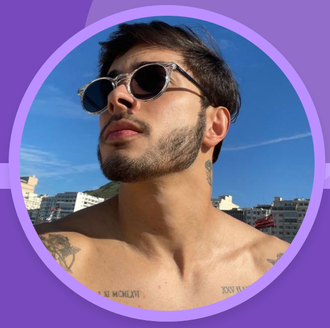
Componente dispara uma ação
↪ Store recebe a ação via dispatch
↪ Reducer processa a ação e retorna um novo estado
↪ Store atualiza o estado
Componentes que dependem do estado são atualizados.

Esse fluxo é repetido sempre que uma mudança de estado é necessária.

Equipe:



Rafaela Cavalcanti



Robério Albuquerque

Mickeliny Sena

João Victor Ferreira