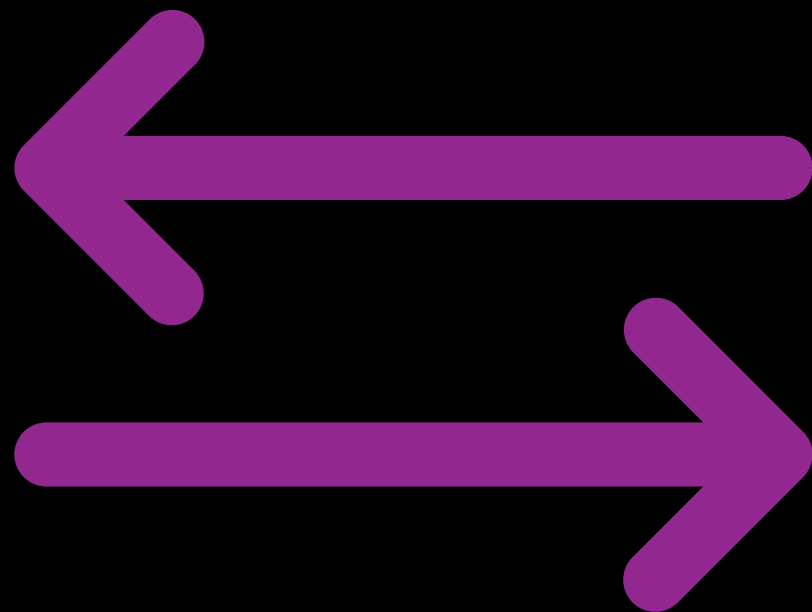




DEV EM
DOBRRO.



Migração para o
Redux Moderno

O que é Redux?



Situação atual
e métodos
que foram
depreciados

```
1 import { createStore, applyMiddleware, combineReducers, compose } from 'redux'
2 import thunk from 'redux-thunk'
3
4 import postsReducer from '../reducers/postsReducer'
5 import usersReducer from '../reducers/usersReducer'
6
7 const rootReducer = combineReducers({
8   posts: postsReducer,
9   users: usersReducer,
10 })
11
12 const store = createStore(rootReducer)
13
```

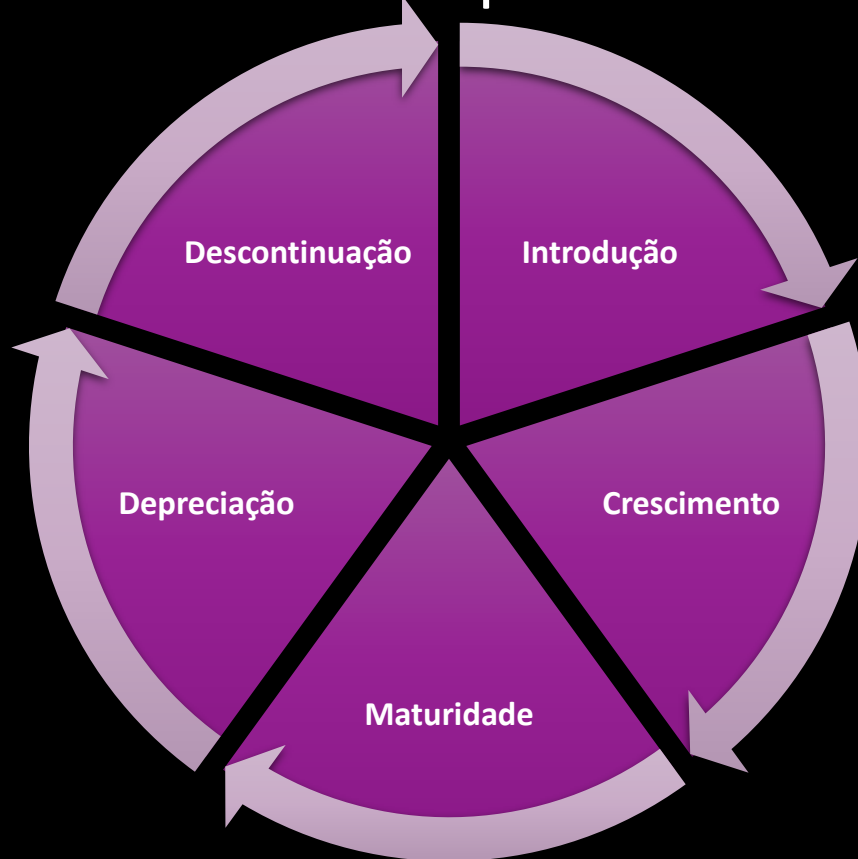
O que significa "depreciação"?

- Desenvolvedores estão recomendando que você não use mais aquilo.
- Alternativas melhores.
- Algum momento no futuro, aquilo vai parar de funcionar .
- Não será mais suportado.



Ciclo de vida do software

- Tudo no mundo da tecnologia tem um ciclo de vida, inclusive os softwares. O ciclo normalmente se parece com isso:



A diferença entre
"depreciação" e
"descontinuação".

Depreciação: É como a montadora dizer:
*"Temos carros melhores agora, talvez você
queira considerar trocar o seu. Mas se você
realmente quiser, ainda pode usar o
modelo antigo, só que não é a melhor
ideia"*.

Descontinuação: É a montadora dizer:
*"Paramos totalmente de fabricar e dar
suporte a esse carro. Se você tiver algum
problema com ele, estamos oficialmente
fora. E, sinceramente, não recomendamos
que você use mais"*.

- No mundo do software, "*depreciação*" é um aviso de que algo está se tornando obsoleto e que há alternativas melhores. Enquanto "*descontinuação*" é quando esse algo é oficialmente retirado e não é mais suportado de forma alguma.

```
    from setTimeout
    afterSomeTime = (time) => new Promise(resolve => {
      setTimeout(() => {
        resolve(true);
      }, time);
    });
    const callAfterSomeTime = (callback, time) => afterSomeTime(time).then(callback);

    callAfterSomeTime(() => console.log('Hello after 1500ms'), 1500);

    const getData = async (url) => fetch(url);

    document
      .querySelector('#submit')
      .addEventListener('click', function() {
        const name = document.querySelector('#name').value;

        // send to backend
        const user = await fetch(`/users?name=${name}`);
        const posts = await fetch(`/posts?userId=${user.id}`);
        const comments = await fetch(`/comments?post=${posts[0].id}`);
        //display comments on DOM
      });
```

Ln 24, Col 1 Space

Por que atualizar nosso código?

Segurança

- Correções de vulnerabilidades.

Novos Recursos

- Funções e ferramentas atualizadas.

Desempenho

- Código mais rápido e eficiente.

Compatibilidade

- Funciona bem com novas tecnologias.

Manutenção

- Código mais limpo e fácil de trabalhar.

A importância de acompanhar as atualizações da biblioteca.



Estar Atualizado:

Usar as melhores ferramentas.



Comunidade:

Mais ajuda e discussões online.



Evitar Surpresas:

Menos mudanças chocantes.



Longevidade:

Facilita atualizações futuras.

Evolução do Redux: Transição para Padrões Modernos

Evolução Histórica

- 2015: Configuração manual da loja, redutores escritos à mão, connect do React-Redux
- Atual: configureStore do Redux Toolkit, createSlice, API de hooks do React-Redux

Desafios

- Muitos ainda trabalham com bases de código Redux antigas.
- Padrões modernos do Redux proporcionam compactação e manutenção mais fácil.

Benefícios da Migração

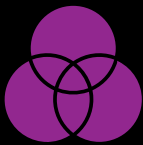
- Base de código otimizada e de fácil manutenção.
- Transição pode ser feita de forma incremental: padrões antigos e novos coexistem harmoniosamente.

O que é o Redux Toolkit (RTK)?

Abordagem oficialmente recomendada para escrever lógica Redux.

Envolve o pacote core do redux com métodos API e dependências essenciais para construir um app Redux.

Benefícios do RTK



Incorpora as melhores práticas sugeridas.



Simplifica tarefas do Redux.



Previne erros comuns.



Facilita a escrita de aplicações Redux.

Utilidades do RTK

Simplificação: Configuração da store, criação de reducers e atualização imutável.

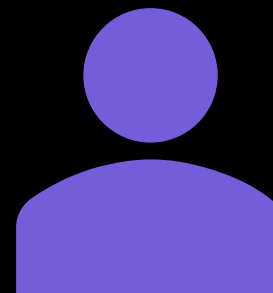
Criação Eficiente: Permite montar "slices" de estado rapidamente.

Otimização: Reduz complexidade de reducers, actions e middlewares.

Para quem é o RTK?



Novos usuários do Redux
configurando seu primeiro projeto.



Usuários experientes buscando
simplificar aplicações existentes.

Métodos e abordagens depreciados

```
const rootReducer = combineReducers({  
  cartProducts: cartReducer  
})  
const store = createStore(rootReducer)
```

- combineReducers

Métodos e abordagens depreciados

```
import { createStore, applyMiddleware, combineReducers, compose } from 'redux'
import thunk from 'redux-thunk'

import postsReducer from '../reducers/postsReducer'
import usersReducer from '../reducers/usersReducer'

const rootReducer = combineReducers({
  posts: postsReducer,
  users: usersReducer
})

const middlewareEnhancer = applyMiddleware(thunk)

const composeWithDevTools =
  window.__REDUX_DEVTOOLS_EXTENSION_COMPOSE__ || compose

const composedEnhancers = composeWithDevTools(middlewareEnhancer)

const store = createStore(rootReducer, composedEnhancers)
```

- applyMiddleware

Todas essas etapas
podem ser
substituídas por
uma única
chamada para o
Redux Toolkit

- Combinação de Redutores: `combineReducers` integrado para unir `postsReducer` e `usersReducer`.
- Criação da Store: `createStore` automático para estabelecer a store do Redux com o redutor raiz.
- Middleware Thunk: Adicionado automaticamente junto com `applyMiddleware`.
- Checagem de Erros: Middlewares incluídos para prevenir mutações acidentais do estado.
- DevTools: Conexão facilitada com Redux DevTools para debugging.

Passos para a migração



```
1 # NPM
2 npm install @reduxjs/toolkit
3
4 # Yarn
5 yarn add @reduxjs/toolkit
```

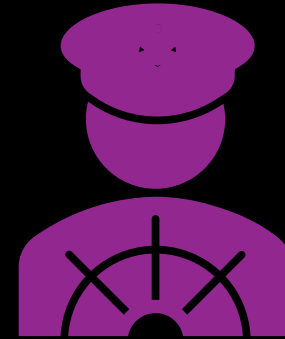


```
1 import { configureStore } from '@reduxjs/toolkit'
2
3 import postsReducer from '../reducers/postsReducer'
4 import usersReducer from '../reducers/usersReducer'
5
6 // Automatically adds the thunk middleware and the Redux DevTools extension
7 const store = configureStore({
8   // Automatically calls `combineReducers`
9   reducer: {
10     posts: postsReducer,
11     users: usersReducer
12   }
13 })
```

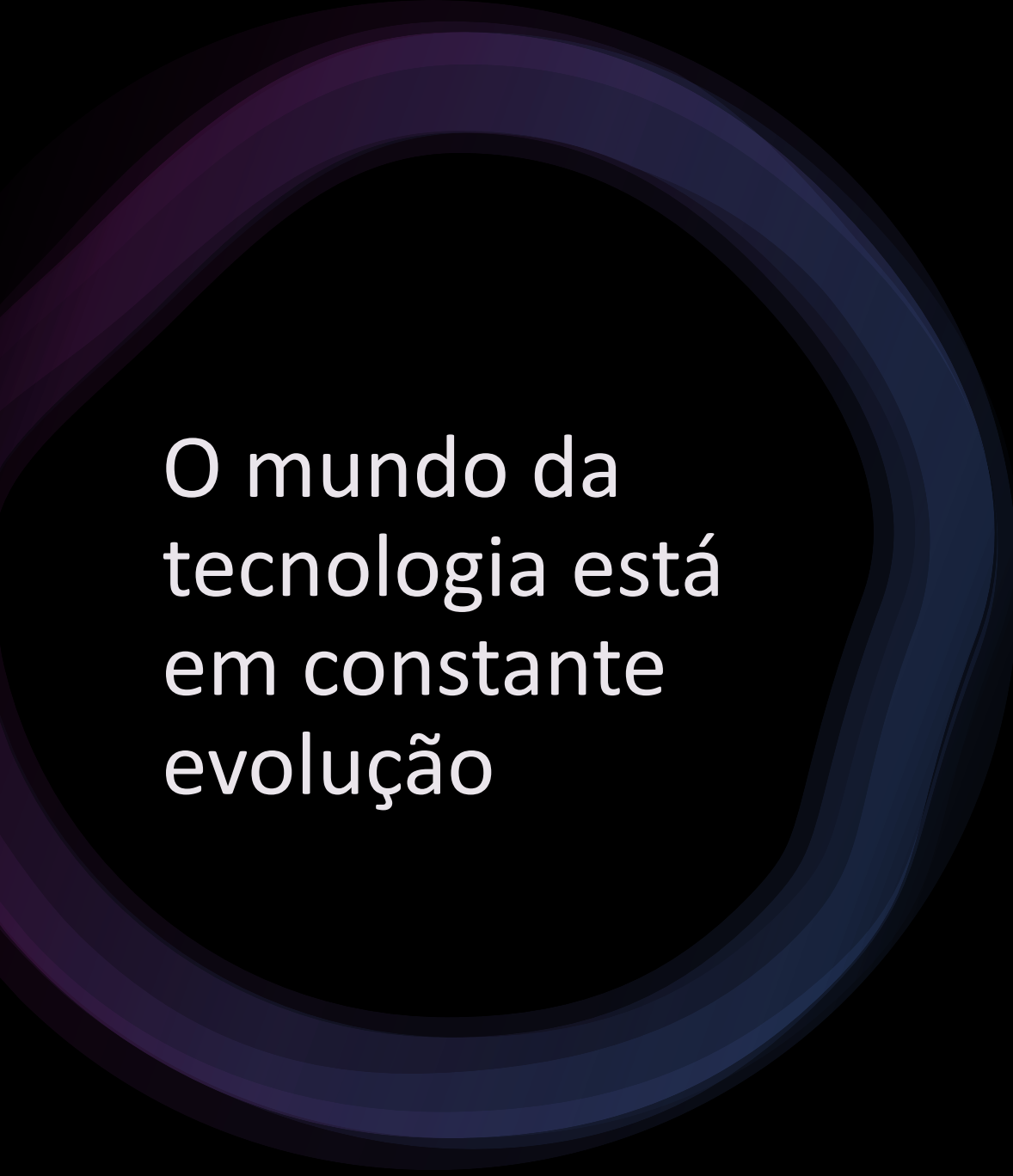
Mudanças & Benefícios da Migração:



Simplificação e Eficiência: Com a migração, conseguimos simplificar a configuração da store, otimizar a lógica dos reducers e tornar a atualização imutável mais direta.



Adaptabilidade: Usando recursos como "slices", a gestão do estado torna-se mais modular e ágil, facilitando a manutenção e expansão do código.



O mundo da tecnologia está em constante evolução

- As mudanças que vimos são apenas um exemplo do progresso contínuo em nossa área. Encorajo todos vocês a manterem a curiosidade acesa, estarem sempre atualizados e conectados às novidades da comunidade. Afinal, nossa aprendizagem e crescimento nunca param!



Vamos pro código.