# O que é Redux Toolkit? (RTK)

**Redux Toolkit (RTK)** é a forma **oficial e recomendada** de usar Redux hoje.

Ele resolve o maior problema do Redux “clássico”:
➡ muita verbosidade
➡ muita repetição de código
➡ configuração confusa

RTK deixa tudo mais fácil, rápido e moderno.

## ✔ Vantagens do Redux Toolkit

### 1️⃣ Menos código

Com RTK, você cria **reducers + actions automaticamente** usando `createSlice()`.

### 2️⃣ Já vem configurado com:

- Redux DevTools
- Thunk (para requisições assíncronas)
- Middlewares úteis

### 3️⃣ Imutabilidade automática

Você pode escrever código "mutável", mas ele gera alterações imutáveis por baixo dos panos usando *Immer*.

## Exemplo usando Redux Toolkit

### 👉 Criando um Slice

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

const contadorSlice = createSlice({
  name: "contador",
  initialState: { valor: 0 },
  reducers: {
    incrementar: (state) => { state.valor += 1 },
    decrementar: (state) => { state.valor -= 1 },
```

```
    adicionar: (state, action) => { state.valor += action.payload },
  },
});

export const { incrementar, decrementar, adicionar } =
contadorSlice.actions;
export default contadorSlice.reducer;
```

### 👉 Criando a Store

```
import { configureStore } from "@reduxjs/toolkit";
import contadorReducer from "./contadorSlice";

export const store = configureStore({
  reducer: {
    contador: contadorReducer,
  },
});
```

### 👉 Usando no React

```
import { useSelector, useDispatch } from "react-redux";
import { incrementar } from "./contadorSlice";

function Contador() {
  const valor = useSelector((state) => state.contador.valor);
  const dispatch = useDispatch();

  return (
    <>
      <p>{valor}</p>
      <button onClick={() => dispatch(incrementar())}>+</button>
    </>
  );
}
```

## Qual usar: Redux “puro” ou Redux Toolkit?

| Redux Tradicional | Redux Toolkit |
| --- | --- |

| | |
|---|---|
| Muito código | Pouco código |
| Verboso | Simples |
| Difícil para iniciantes | Recomendado oficialmente |
| Mais configuração manual | Já vem com tudo pronto |

**Conclusão:**
👉 *Se você for começar um projeto hoje, use **Redux Toolkit**.*
Redux puro só é usado em projetos antigos.

## Quando usar Redux / Redux Toolkit?

Use quando:

- Muitos componentes precisam do mesmo estado
- Há requisições assíncronas complexas
- Estados precisam ser previsíveis e auditáveis
- Aplicação grande (equipe grande)

Não use quando:

- A aplicação é pequena
- O React Context resolve o problema
- Você não precisa de logs, middlewares, etc.

# Estrutura do projeto (Redux Toolkit + TS)

```
src/
 ├─ store/
 │   ├─ contadorSlice.ts
 │   └─ store.ts
 ├─ components/
 │   └─ Contador.tsx
 ├─ App.tsx
 └─ main.tsx (ou index.tsx)
```

Pronto: só **2 arquivos de Redux**.

# 1 Criando o Slice (Reducer + Actions automatizadas)

Arquivo: `src/store/contadorSlice.ts`

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

export interface ContadorState {
  valor: number;
}

const initialState: ContadorState = {
  valor: 0,
};

const contadorSlice = createSlice({
  name: "contador",
  initialState,
  reducers: {
    incrementar: (state) => {
      state.valor++;  // Imutabilidade automática
    },
    decrementar: (state) => {
      state.valor--;
    },
    adicionar: (state, action: PayloadAction<number>) => {
      state.valor += action.payload;
    },
  },
});

// Exporta actions e reducer
export const { incrementar, decrementar, adicionar } =
contadorSlice.actions;

export default contadorSlice.reducer;
```

### ✔ Por que isso é melhor?

- `createSlice()` gera as **actions** de forma automática.
- A mutação (state.valor++) é **segura** → o RTK converte para estado imutável.
- `PayloadAction<number>` tipa a action corretamente.

# 2 Criando a Store

Arquivo: `src/store/store.ts`

```
import { configureStore } from "@reduxjs/toolkit";
import contadorReducer from "./contadorSlice";

export const store = configureStore({
  reducer: {
    contador: contadorReducer,
  },
});

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

### ✔ Por que configurar assim?

- `configureStore` já inclui:
    - Redux DevTools
    - Thunk
    - Middlewares necessários
- Não precisa de configurações extras.

# 3 Provider no React

Arquivo: `src/main.tsx`

```
import React from "react";
```

```
import ReactDOM from "react-dom/client";
import App from "./App";
import { Provider } from "react-redux";
import { store } from "./store/store";

ReactDOM.createRoot(document.getElementById("root")!).render(
  <Provider store={store}>
    <App />
  </Provider>
);
```

## 4 Componente usando Redux Toolkit + TypeScript

Arquivo: `src/components/Contador.tsx`

```
import { useDispatch, useSelector } from "react-redux";
import { RootState, AppDispatch } from "../store/store";
import { incrementar, decrementar, adicionar } from
"../store/contadorSlice";

function Contador() {
  // Lê valor do estado global
  const valor = useSelector((state: RootState) =>
state.contador.valor);

  // Dispatch tipado
  const dispatch = useDispatch<AppDispatch>();

  return (
    <div>
      <h2>Valor atual: {valor}</h2>

      <button onClick={() => dispatch(incrementar())}>+</button>
      <button onClick={() => dispatch(decrementar())}>-</button>

      <button onClick={() => dispatch(adicionar(5))}>
        Adicionar 5
      </button>
    </div>
  );
```

```
}

export default Contador;
```

### ✔ O que melhorou?

- Código MUITO menor
- A tipagem flui automaticamente
- Reducers mais claros
- Sem switch-case
- Sem action types manuais
- Dispatch totalmente tipado

# 5 App.tsx

```
import Contador from "./components/Contador";

function App() {
  return (
    <div style={{ padding: "20px" }}>
      <h1>Redux Toolkit + TypeScript</h1>
      <Contador />
    </div>
  );
}

export default App;
```

# Comparação entre Redux Clássico e Redux Toolkit

| Item | Redux Clássico | Redux Toolkit |
|---|---|---|
| Criar actions | Manual | Automático |
| Criar reducer | switch-case gigante | createSlice |
| Imutabilidade | Spread manual | Automático (Immer) |

| | | |
|---|---|---|
| Configuração da store | Verbosa | Simples |
| Código total | Grande | Pequeno |
| Produtividade | Baixa | Alta |