

Redux / Redux Toolkit

...

C'est quoi Redux

- Redux est une librairie JavaScript qui permet de gérer les états.
- Ce n'est pas seulement une librairie React, il y a une version pour différents frameworks : Vue.js, Angular, React.



la librairie résout quel problème ?

Quand une application prend de l'ampleur, la gestion des états et la communication entre les différents composants qui vont utiliser cet état peut devenir complexe.

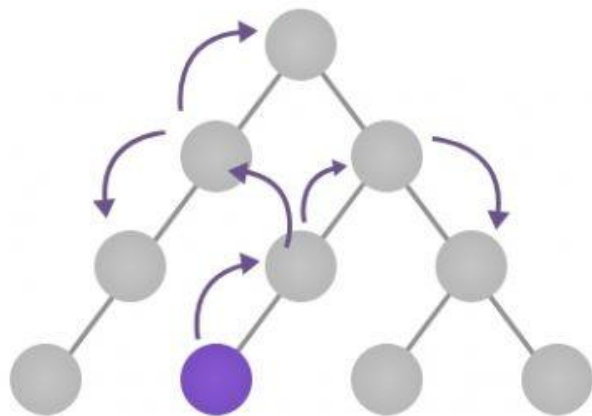
Stateflow initial de React

- il est multidirectionnel
- pour pouvoir partager, un état dans plusieurs composants, il faut parfois faire remonter l'état pour pouvoir le faire passer en props à travers beaucoup de composant pour enfin pouvoir l'utiliser

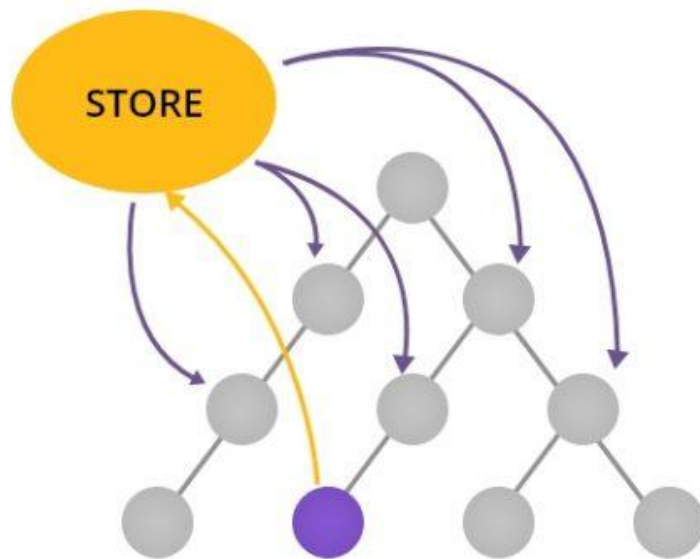
Stateflow de Redux


- il est unidirectionnel
- on stocke les états en dehors du flow de l'application, dans un store
- la logique derrière la gestion d'état est plus simple, plus intuitive, ce qui permet un débogage plus simple

Without Redux



With Redux



 Component initiating change

Installation

```
# NPM
npm install @reduxjs/toolkit

# Yarn
yarn add @reduxjs/toolkit
```

```
# If you use npm:
npm install react-redux

# Or if you use Yarn:
yarn add react-redux
```

si vous voulez vous prendre la tête

```
# NPM
npm install redux

# Yarn
yarn add redux
```

Attention ! ne pas installer la version toolkit et la version core sur le même projet

Comment mettre le store en place

app/store.js

```
import { configureStore } from '@reduxjs/toolkit'

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

index.js

```
import React from 'react'
import ReactDOM from 'react-dom'
import './index.css'
import App from './App'
import { store } from './app/store'
import { Provider } from 'react-redux'

ReactDOM.render(
  <Provider store={store}>
    <App />
  </Provider>,
  document.getElementById('root')
)
```

Ce que comporte le store

On peut voir que pour configurer le store on va faire un objet avec un reducer

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

C'est dans ce reducer que l'on va appeler et déterminer les états que l'on va utiliser dans toute l'application.

Maintenant, il faut créer les états que l'on va appeler dans ce store.

Configuration d'un état

features/counter/counterSlice.js

```
import { createSlice } from '@reduxjs/toolkit'

const initialState = {
  value: 0,
}

export const counterSlice = createSlice({
  name: 'counter',
  initialState,
  reducers: {
    increment: (state) => {
      // Redux Toolkit allows us to write "mutating" logic in reducers. It
      // doesn't actually mutate the state because it uses the Immer library,
      // which detects changes to a "draft state" and produces a brand new
      // immutable state based off those changes
      state.value += 1
    },
    decrement: (state) => {
      state.value -= 1
    },
    incrementByAmount: (state, action) => {
      state.value += action.payload
    },
  },
})

// Action creators are generated for each case reducer function
export const { increment, decrement, incrementByAmount } = counterSlice.actions

export default counterSlice.reducer
```


Décortiquons ce fichier

CreateSlice

Pour créer / initialiser un état, on va faire appel à la fonction `createSlice` que l'on importe depuis `redux/toolkit` et qui va prendre un objet en argument.

→ l'état se retrouve dans une fonction que l'on nomme comme bon nous semble avec `Slice` au bout.

```
import { createSlice } from '@reduxjs/toolkit'

const initialState = {
  value: 0,
}

export const counterSlice = createSlice({
```

À l'intérieur de la fonction createSlice

On remarque 3 éléments:

- 1) name
- 2) initialState
- 3) reducers

→ pas besoin de passer trop de temps sur le nom,
il s'agit tout simplement du nom de l'état

→ l'initialState est la valeur de base, la première valeur de cet état. On va faire de cet initialState un objet avec une valeur "value", pour pouvoir aller la récupérer plus tard.

```
export const counterSlice = createSlice({  
  name: 'counter',  
  initialState,  
  reducers: {
```

```
    const initialState = {  
      value: 0,  
    }
```

Les reducers

```
reducers: {  
  increment: (state) => {  
    // Redux Toolkit allows us to write "mutate  
    // doesn't actually mutate the state because  
    // which detects changes to a "draft state"  
    // immutable state based off those changes  
    state.value += 1  
  },  
  decrement: (state) => {  
    state.value -= 1  
  },  
  incrementByAmount: (state, action) => {  
    state.value += action.payload  
  },  
}
```

C'est quoi un reducer

→ Une fonction qui prend en entrée l'état actuel d'une app et une action

STATE

- l'état actuel du store

ACTION

- renvoie un nouvel état.

PAYLOAD

- utilisé pour mettre à jour l'état du store en fonction des informations contenues dedans

Reducers

Dans cet exemple, on remarque “reducers”
contient plusieurs fonction:

- increment
- decrement
- incrementByAmount

```
reducers: {  
  increment: (state) => {  
    // Redux Toolkit allows us to write "mutate" state  
    // doesn't actually mutate the state because it uses  
    // which detects changes to a "draft state" based on  
    // immutable state based off those changes  
    state.value += 1  
  },  
  decrement: (state) => {  
    state.value -= 1  
  },  
  incrementByAmount: (state, action) => {  
    state.value += action.payload  
  },  
}
```

Reducers

→ on remarque que ces deux reducers prennent comme argument l'état : state

→ Lorsque l'on va les utiliser, on va aller chercher la valeur de l'état l'augmenter ou la diminuer de 1

```
const initialState = {  
  value: 0,  
}
```

```
increment: (state) => {  
  // Redux Toolkit allows us  
  // doesn't actually mutate  
  // which detects changes  
  // immutable state based  
  state.value += 1  
},  
decrement: (state) => {  
  state.value -= 1  
},
```

Reducers

→ Pour le dernier reducer, on voit qu'il prend deux arguments, un état et une action : state , action

→ Lorsque l'on va l'utiliser, on va aller chercher la valeur de l'état pour lui additionner une valeur qui sera égal au payload de l'action

On va voir plus tard, en pratique, ce que représente "action.payload" , c'est tout à fait normal que ce soit flou pour le moment

```
incrementByAmount: (state, action) => {  
  state.value += action.payload  
},
```


la finalisation du Slice

On termine en exportant chaque reducer et en exportant la fonction qui contient notre état et nos reducers

→ Il ne reste plus qu'à implémenter cet état dans notre store pour pouvoir l'utiliser dans nos composants

features/counter/counterSlice.js

```
import { createSlice } from '@reduxjs/toolkit'

const initialState = {
  value: 0,
}

export const counterSlice = createSlice({
  name: 'counter',
  initialState,
  reducers: {
    increment: (state) => {
      // Redux Toolkit allows us to write "mutating" logic in reducers. It
      // doesn't actually mutate the state because it uses the Immer library,
      // which detects changes to a "draft state" and produces a brand new
      // immutable state based off those changes
      state.value += 1
    },
    decrement: (state) => {
      state.value -= 1
    },
    incrementByAmount: (state, action) => {
      state.value += action.payload
    },
  },
})

// Action creators are generated for each case reducer function
export const { increment, decrement, incrementByAmount } = counterSlice.actions

export default counterSlice.reducer
```

La mise en place de notre état dans le store

On va importer le fichier que l'on a créé pour faire notre état et on va tout simplement l'ajouter dans les reducers du store.

app/store.js

```
import { configureStore } from '@reduxjs/toolkit'
import counterReducer from '../features/counter/counterSlice'

export const store = configureStore({
  reducer: {
    counter: counterReducer,
  },
})
```

→ c'est avec le nom que l'on donne ici que l'on va pouvoir appeler notre état dans le futur (ici, counter)

Comment utiliser et modifier les états

Aller voir sur le repo github

<https://github.com/VincentDevi/redux-counter>

Comment aller chercher et modifier l'état

on va utiliser deux hooks qui sont compris dans react-redux

→ useSelector

on va utiliser ce hook, pour aller chercher la valeur d'un état.

→ useDispatch

on va utiliser ce hook, pour modifier la valeur d'un état

useSelector

On passe une fonction à notre useSelector qui va aller sélectionner la partie du store que l'on souhaite accéder.

```
import { useSelector } from "react-redux"; 5.6k (gzipped: 2.1k)

export const Result = () =>{
  const result = useSelector((state) => state.counter.value)
  return (
    <p>{result}</p>
  );
}
```

→ ici on va aller chercher le “counter” et prendre la valeur de cet état.

(Rappel : quand on a définit notre état le fichier Slice, on a définit la valeur initiale comme : {value: 0})

useDispatch

On passe un des réducteurs
qui a été défini dans le
l'état (dans le Slice),
→ ce qui va appeler cette
fonction et procéder aux
changements d'états définis.

```
import { useDispatch } from "react-redux"; 5.5k (gzipped: 2k)
import { increment } from "../../features/counterSlice";
💡
export const ButtonAddOne = () =>{
  const dispatch = useDispatch();
  return (
    <button onClick={()=>dispatch(increment())}>
      add one
    </button>
  );
}
```

useDispatch

Ici on peut voir un exemple où l'on définit sur le moment la valeur à ajouter, cette valeur va se retrouver dans le payload.

```
import { useRef } from "react"; 4.1k (gzipped: 1.8k)
import { useDispatch } from "react-redux"; 5.5k (gzipped: 2k)
import { incrementByAmount } from "../../features/counterSlice";

export const IncrementByAmount = () =>{
  const inputRef = useRef(0);
  const dispatch = useDispatch();
  return (
    <>
      <input type="number" ref={inputRef} />
      <button onClick={()=>dispatch((incrementByAmount(Number(inputRef.current.value))))}>
        increment by {inputRef.current.value}
      </button>
    </>
  );
}
```