State Management

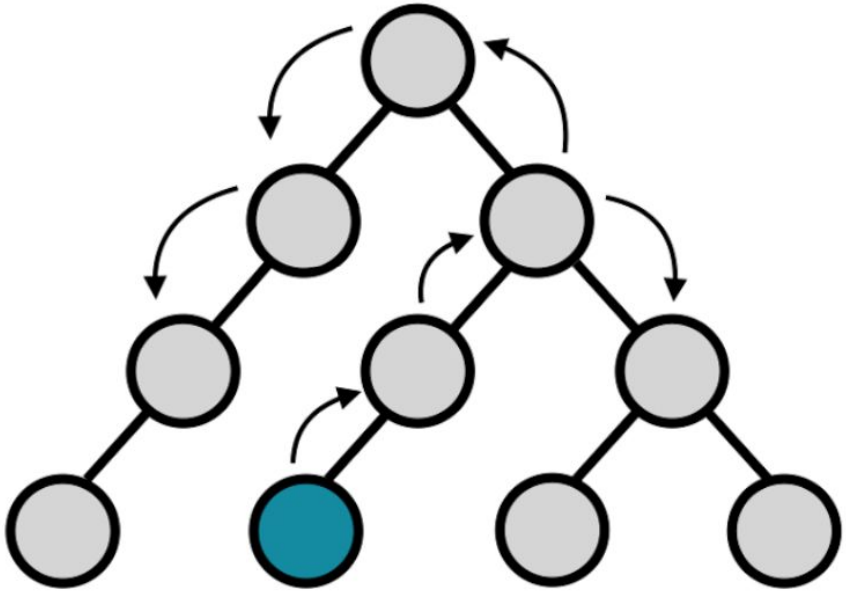
Introduction

State Management

Pourquoi utiliser un « state management »

Manipuler les données dans différents composants React peut s'avérer compliqué, car il faut faire transiter les données au sein de l'arborescence.

Pour éviter cela, il est conseillé de mettre en place un mécanisme de « state management ».



State management

Un système de « state management » a pour objectif de simplifier le stockage de données au sein d'une application.

En Javascript, il existe plusieurs solutions :

- **Data Store** : Flux (Non Conseillé), Redux, MobX, Zustand.
- **Observer pattern** : RxJS
- **État partagé** : Recoil, Jotai.

Dans le cadre de ce cours, nous allons utiliser la librairie « Redux ».

La librairie Redux

State Management

Un store, ça sert à quoi ?

Redux est une librairie JavaScript qui met en place le mécanisme de « Store ».

Le principe de fonctionnement d'un Store est :

- Création d'un conteneur centralisé pour les données.
- Interaction avec les données uniquement à travers de méthodes.
- Notification des changements des données via des événements.

Redux, ça fonctionne comment ?

Le fonctionnement du Store Redux se base sur l'utilisation de deux éléments :

- **Les Actions**
  Ce sont des objets Javascript qui représentent les actions à réaliser.
  Celle-ci contiennent :
  - Le type d'action.
  - Des données *(si cela est nécessaire)*.

- **Les Reducers**
  Ce sont des fonctions qui se déclencheront lorsque qu'une action est reçue.
  Leur objectif est de mettre à jour les données du Store.

Procédure pour créer un Store de Redux

Pour utiliser le store Redux, il faut installer le package « redux » .

“ npm install –save redux ”

Une fois installé, il faut :

➔ Définir les différents Actions.

➔ Créer les Reducer pour résoudre les actions.

➔ Configurer l’instance du store Redux.

Exemple de mise en place d'un store Redux

Pour cet exemple de création d'un store, nous allons mettre en place un compteur.

Les actions que le store devra être capable de résoudre sont les suivantes :

- « cpt/increment »
  Permet d'augmenter la valeur du compteur de X.
- « cpt/reset »
  Permet de définir la valeur du compteur à 0.

Exemple de mise en place d'un store Redux

```
// Configuration et création du store
// **********************************
import { legacy_createStore as createStore } from 'redux';

// Etat initial du state
const initialState = {
    count: 0
};

// Création du Reducer
const reducer = (state = initialState, action) => {
    switch (action.type) {
        case "cpt/increment":
            return {
                count: state.count + action.payload
            };
        case "cpt/reset":
            return {
                count: 0
            };
        default:
            return state;
    }
};

// Création du Store
const store = createStore(reducer);
```

```
// Utilisation du store
// ********************

// Affichage du contenu
const state1 = store.getState();
console.log('State 1', state1);

// Abonnement à la réception d'action
const unsubscribe = store.subscribe(() => {
    console.log('Action traité');
});

// Envoi d'une action dans le store
store.dispatch({ type: 'cpt/increment', payload: 1 });

// Affichage du contenu
const state2 = store.getState();
console.log('State 2', state2);

// Envoi d'action dans le store
store.dispatch({ type: 'cpt/exemple', payload: 'Hello' });
store.dispatch({ type: 'cpt/increment', payload: 4 });

// Affichage du contenu
const state3 = store.getState();
console.log('State 3', state3);

// Désabonnement
unsubscribe();
```

```
State 1 { count: 0 }
Action traité
State 2 { count: 1 }
Action traité
Action traité
State 3 { count: 5 }
```

Intégration de Redux dans un projet

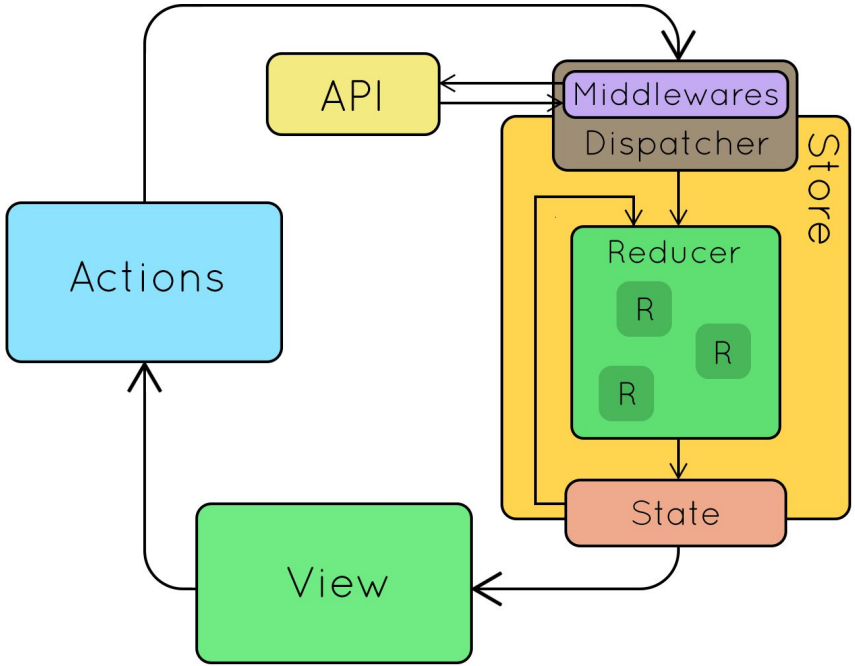
State Management

Présentation du pattern d'utilisation de Redux

Pour mettre en place le store Redux dans un projet, il est conseillé d'utiliser le pattern « one-way data flow ».

Ce pattern permet de simplifier l'utilisation de Redux :

- La vue consomme les données du store.
- Les interactions sont transmises au dispatcher via des actions dédiées.
- Les middlewares permettent d'ajouter des comportements avant les reducers.

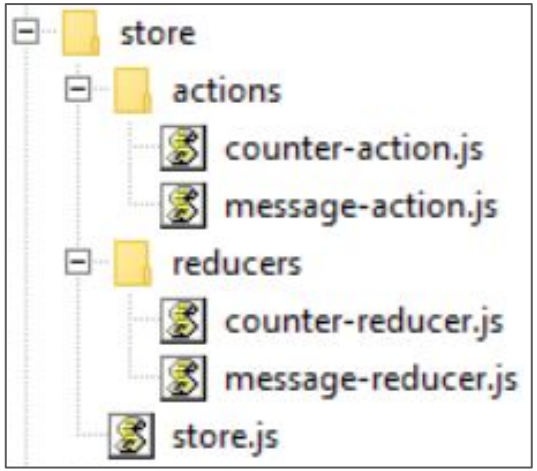


Intégration du pattern de Redux dans un projet

Pour ajouter le store dans nos projets avec ce pattern d'utilisation, il faudra mettre en place les éléments suivants :

- Des « Action Creators ».
- Des « Reducers ».
- La configuration du store Redux.

Structure de fichier :



Remarque : La structure utilisée dans ce support est « Par types d'éléments ».

Le fonctionnement des « Action Creators »

Les « Action Creators » sont des méthodes permettant de générer les objets « Action » de Redux.

L'objet généré par un « Action Creator » sera composé de :

- Un type (Obligatoire)
  L'identifiant de l'action
- Un Payload
  Les données nécessaire à l'action

```
// Action Types
export const COUNTER_INCR = 'counter/increment';
export const COUNTER_RESET = 'counter/reset';

// Action Creators
export const counterIncr = (step) => {
    return {
        type: COUNTER_INCR,
        payload: step
    };
};

export const counterReset = () => {
    return {
        type: COUNTER_RESET
    };
};
```

Le fonctionnement des « Reducers »

Un « Reducer » est une méthode qui a pour but de modifier le « State ».

Cette méthode doit posséder les deux paramètres suivants :

- State → La valeur du state actuel
- Action → L'action à réaliser

Cette méthode doit toujours renvoyer une nouvelle valeur de state.

```
import { COUNTER_INCR, COUNTER_RESET } from '../actions/counter-action';

// Initial state for "Counter"
const initialState = {
    count: 0
};

// Reducer for "Counter"
const counterReducer = (state = initialState, action) => {
    switch (action.type) {
        case COUNTER_INCR:
            return {
                ...state,
                count: state.count + action.payload
            }
        case COUNTER_RESET:
            return {
                ...state,
                count: 0
            }
        default:
            return state;
    }
};

export default counterReducer;
```

Best Practice

State Management

Bonne pratique pour « Redux »

Voici une brève liste de quelques bonnes pratiques importantes avec Redux :

- Avoir un seul store pour l'application.
- Ne jamais faire muter le State.
- Ne pas avoir de donnée non-sérialisable dans le State ou dans les actions.
- Les Reducers ne doivent pas avoir de « Side Effects »
  ➜ Pas de requête Ajax, pas de génération aléatoire, ...
- Ecrire le nom des types d'action sous la forme « domain/eventName »

Liste complete des bonnes pratiques disponible sur https://redux.js.org/style-guide/