Mise en place de Redux

Le package « Redux Toolkit »

Mise en place de Redux

Objectif de Redux Toolkit

Pour réaliser la mise en place du store, nous allons utiliser « Redux Toolkit ».

Ce package est fourni par l'équipe de développeurs de Redux.
Ses principaux objectifs sont :

- Simplifier la configuration du store Redux.
- Générer des « Action Creators ».
- Faciliter l'écriture des « Reducers ».

Pour utiliser le Toolkit, il faut l'installer.

" npm install –save @reduxjs/toolkit "

Action Creators

Mise en place de Redux

Les Action Creators

Le Toolkit permet de générer les Action Creators à l'aide de « createAction ».

```
import { createAction } from '@reduxjs/toolkit';

// Action Creators
export const counterIncr = createAction('counter/increment');
export const counterReset = createAction('counter/reset');
```

L'avantage de l'utiliser la méthode du toolkit est :

- Création automatique d'une méthode qui génère un objet « Action »
  { type: « action_type » , payload: « action_payload » }
- Le type peut être obtenu via de la méthode « toString() » de l'action creator.

Les Action Creators

Il est possible de customiser l'objet généré par la méthode « createAction ».
Cela permet d'ajouter dans le Payload : une valeur par défaut, un identifiant généré...

Dans ce cas, il est nécessaire d'utiliser un callback pour définir la valeur du « Payload »

```
export const messageAdd = createAction('message/add', (content) => {
    return {
        payload: {
            id: nanoid(),
            content,
            createAt: new Date().toISOString()
        }
    };
});
```

Reducers

Mise en place de Redux

Les Reducers

L'implémentation classique d'un Reducer est un bloc "switch" avec différents "case" pour résoudre chaque action.
Cette approche fonctionne bien, mais elle peut également être source d'erreur.
Quelques exemples : oublier le cas par défaut, ne pas définir l'état initial, etc...

Le Toolkit permet de créer un Reducer à l'aide de la méthode « createReducer ».
Cette méthode utilise un builder qui possède les méthodes suivantes :

- builder.addCase(actionCreator, callback)
- builder.addMatcher(matcherPredicate, callback)
- builder.addDefaultCase(callback)

Les Reducers

La méthode « createReducer » attend comme paramètres :

- L'état initial du State
- Un callback qui définit le « Builder » avec les différents évent à traiter.

Les Action Creators créés par le Toolkit sont utilisables comme valeur d'évent.

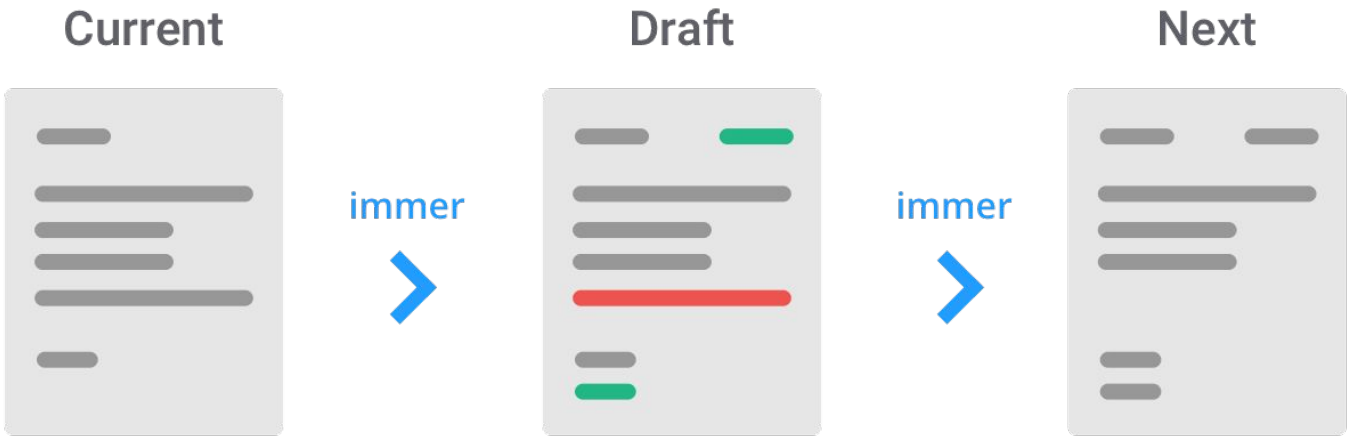
Il n'est pas obligatoire d'ajouter l'évent par défaut, s'il n'est pas customisé.

```
import { createReducer } from '@reduxjs/toolkit';
import { counterIncr, counterReset } from '../actions/counter-action';

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

// Reducer for "Counter"
const counterReducer = createReducer(initialState, (builder) => {
    builder
        .addCase(counterIncr, (state, action) => {
            return {
                ...state,
                count: state.count + action.payload
            };
        })
        .addCase(counterReset, (state) => {
            return {
                ...state,
                count: 0
            }
        })
});

export default counterReducer;
```

Les Reducers - La bibliothèque « Immer »

La méthode « createReducer » utilise « Immer » pour faciliter l'écriture du code.

Cette bibliothèque permet de réaliser des modifications sur des données immutable, à l'aide d'une écriture « mutable » sur une copie des données.



Les Reducers - Utilisation de Immer

Grâce à l'intégration de Immer, il est donc possible de modifier les données à l'aide d'opérations simples.

- Pour les valeurs :
  - Affectation d'une nouvelle valeur
  - Incrémentation et décrémentation

- Pour les tableaux (Array) :
  - Utilisation de l'opérateur d'accès
  - Ajout d'éléments (push, unshift)
  - Retirer des éléments (pop, shift)

```
import { createReducer } from '@reduxjs/toolkit';
import { counterIncr, counterReset } from '../actions/counter-action';

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

// Reducer for "Counter" with Immer
const counterReducer = createReducer(initialState, (builder) => {
    builder
        .addCase(counterIncr, (state, action) => {
            state.count += action.payload
        })
        .addCase(counterReset, (state) => {
            state.count = 0
        })
});

export default counterReducer;
```

⚠️ Quand l'écriture « Mutable » est utilisée, il ne faut pas renvoyer de state !

La configuration du store

Mise en place de Redux

Configuration du store

Il est recommandé de générer l'instance du store avec la méthode « configureStore », celle-ci permet de réaliser la configuration du store Redux simplement.

Cette méthode a comme paramètre un objet « options » avec les valeurs suivants :

- Reducer : L'ensemble des différents reducers de l'application.
- Middleware : Les fonctionnalités à ajouter au « Dispatcher » du store.
- DevTools : Un booléen pour activer les outils pour développeur.
- PreloadedState : Une ancienne instance du state *(Restauration de données)*.
- Enhancers : Les fonctionnalités à ajouter au store Redux.

Exemple de la configuration d'un store

Dans cette exemple, la store est initialisé avec les options suivants :

- Reducer :
  Un objet qui combine les différents reducers avec un alias.
- DevTools:
  Désactivé en build de production.

Le store Redux fonctionnera donc avec un unique reducer, qui sera le résultat de la combinaison des reducers aliasés.

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

// Import des différents Reducers
import counterReducer from './reducers/counter-reducer';
import messageReducer from './reducers/message-reducer';

// Création du store avec la méthode "configureStore"
const store = configureStore({
    // Fusion des différents Reducers
    reducer: {
        counter: counterReducer,
        message: messageReducer
    },

    // Activation des outils de dev
    devTools: process.env.NODE_ENV !== 'production'
});

// Export du store créé
export default store;
```

Configuration des Middlewares dans un store

L'option « middleware » prend comme argument un tableau de Middleware à ajouter.

Lorsque cette option n'est pas présente, le « configureStore » va automatiquement appeler la méthode « getDefaultMiddleware » pour ajouter les middlewares suivants :

- **Immutability Check** [Development]
  Génère une erreur lorsqu'une mutation des données est réalisée dans le store.
- **Serializability Check** [Development]
  Génère une erreur quand le store contient des données non sérialisables.
- **Redux Thunk** [Development & Production]
  Permet d'ajouter des effets de bord *(Celui-ci sera abordé en détail plus tard)*

Configuration des Middlewares dans un store

Il est conseillé de conserver les middlewares par défaut lors de la configuration.

Pour cela, il faut utiliser un callback qui permettra de combiner le(s) middleware(s) à ajouter avec ceux définis par défaut.

```
import { configureStore } from '@reduxjs/toolkit';
import reduxLogger from 'redux-logger';

// Création du store avec la méthode "configureStore"
const store = configureStore({
    // Les options "reducers" et "devTools" sont masquées pour la lisibilité

    // Ajout des middlewares
    middleware: (getDefaultMiddleware) => getDefaultMiddleware().concat(reduxLogger)
});

// Export du store créé
export default store;
```