Formation React + Redux + Env

# Mise en place du cycle Redux

Redux est composé en différentes parties, chacune ayant son rôle distinct. Le tout fonctionne ensemble pour faire circuler l’information, de la saisie utilisateur jusqu’au Store, un simple objet au format JSON.

Commençons tout de suite par définir notre arborescence. Les différentes parties de Redux sont :

* Actions
* Store
* Components & Containers
* Reducers

Une Action est déclenchée par un utilisateur. Celle-ci est envoyée au Reducer qui va traiter l’action et envoyer le résultat au Store. Ce store est distribué aux Component/Containers Afin d’afficher les données à l’écran via le point d’entrée (c.f. step-1).

Une interrogation importante subsiste néanmoins :

**Qu’est-ce qu’un component ?**

Il s’agit simplement d’un élément ou un groupe d’élément afficher à l’écran. Par exemple un component « social » contenant les différents boutons de réseaux sociaux (Twitter, Facebook, etc.). L’application React comprend la plupart du temps un component principal qui remplacera notre fameux « Hello World » et c’est dans ce component principal que les autres components seront utilisés.

Concernant les autres parties de Redux, elles seront décrites plus tard.

Maintenant, créons les dossiers correspondants dans « dev/js/ », là où se trouve déjà notre point d’entrée « index.jsx » (Créez bien un dossier « Components » et un autre « Containers », les deux fonctionnent ensembles mais doivent être distinct pour plus de lisibilité).

En matière d’organisation, nous allons créer un dossier « AppWrapper » et un dossier « Api ». « AppWrapper » va contenir notre component principal à afficher à l’écran et « Api » contiendra nos **appels asynchrones vers une api extérieure**.

Bien, maintenant, codons !

### Component React

Nous allons créer notre premier component et l’afficher à l’écran. Pour ce faire, créez un dossier « Hello » dans le dossier « Components » et placez-y un fichier nommé « index.jsx » avec le contenu suivant :

import React from 'react';

const Hello = () => <h1>Hello World From Component !</h1>;

export default Hello;

Il s’agit d’une toute simple fonction écrire en ES2015 et retournant un virtual DOM, ce fameux DOM créé par Facebook pour React. Ce virtual DOM ressemble énormément à du HTML alors vous ne serez pas perdu. Nous exportons en suite notre composant afin de l’inclure… dans le point d’entrée !

Voici donc notre point d’entrée modifié :

import React from 'react';

import ReactDOM from 'react-dom';

import Hello from './Components/Hello';

ReactDOM.render(

<Hello />,

document.querySelector('#root'),

);

Notez l’import du component « Hello » et son utilisation en tant que balise, comme en HTML encore une fois !

Si vous regardez bien, je n’importe pas le fichier index.jsx directement, mais seulement le dossier « Hello ». WebPack s’occupe tout seul de m’importer mon fichier « index.jsx ». Si le nom du fichier est différent, alors il faudra importer le fichier. Ici aussi, l’extension ne sera pas obligatoire car résolue automatiquement.

« Comment cela ce fesse ? » me direz-vous ? J’ai modifié très rapidement notre fichier de configuration « webpack.config.js » comme suit :

output: {

// ...

},

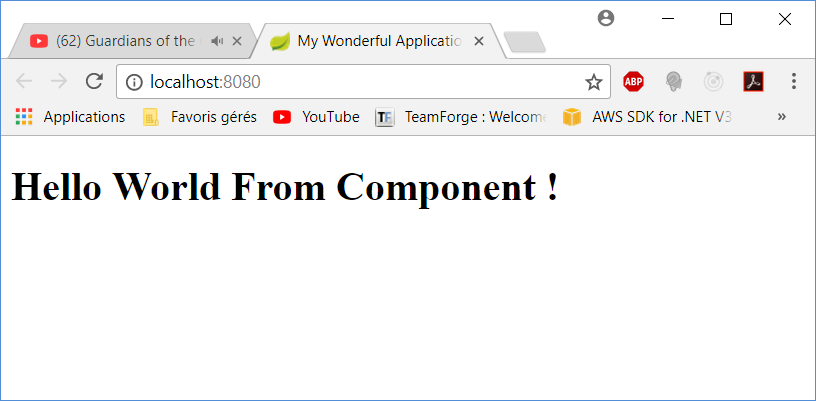
resolve: {

extensions: ['\*', '.js', '.jsx'],

},

« resolve » permet à Webpack de résoudre certaines extensions tout seul.

Lancez votre application et vous devriez constater l’affichage suivant :



Nous avons donc correctement implémenté notre premier component React ! Plutôt simple n’est-ce pas ? Mais insuffisant évidemment… passons à Redux. \*Tin tin tiiiiiiiinnnn !! \*

### Création des fichiers

Nous allons créer notre cycle Redux.

#### Actions

Puisque nous avons déjà notre arborescence, allez dans le dossier « Action » et créez-y un fichier nommé « ActionsTypes.js » comme suit :

export const SAY\_HELLO = "SAY\_HELLO";

Les ActionsTypes, ou types d’actions, ce sont seulement les noms qui vous permettrons d’identifier vos actions dans votre cycle Redux. Créons une action pour mieux comprendre.

Toujours dans le même dossier « Actions », créez un fichier « HelloActions.js » avec le code suivant :

import \* as types from './ActionsTypes';

const sayHello = hello => ({

type: types.SAY\_HELLO,

hello,

});

export default sayHello;

Si vous avez bien regardé précédemment, nous avons exporté la constante de notre fichier ActionsTypes, nous pouvons donc toutes les récupérer ici grâce à l’écriture « import \* as » suivi d’un nom de variable de votre choix. Nous utilisons ensuite cette variable pour renseigner notre action.

Une action, c’est simplement un objet qui sera envoyé au reducer et qui contient au **minimum une clef** nommée conventionnellement « type ».

Notre action « sayHello » contient une autre clef « hello », qui est une variable. Il s’agit d’une syntaxe ES2015. La clef sera créée automatiquement, avec pour valeur le contenu de la variable. Ici, il s’agit du paramètre passé à notre action.

#### Reducers

Allez maintenant dans le dossier « Reducers » et créez-y un fichier « HelloReducer.js » comme suit :