Formation React + Redux + Env

# Action asynchrone

Nous allons maintenant créer une action asynchrone. C’est un tout petit peu différent des actions communes. Récupérons une liste de film sur une API extérieure par exemple.

Pour ce faire, créez un fichier « index.js » dans le dossier « api » et insérez-y le code suivant :

import fetch from 'isomorphic-fetch';

export const getMovies = () => fetch('https://facebook.github.io/react-native/movies.json')

.then(

response => response.json(),

error => console.log('An error occured : ', error),

);

Ici, nous importons « isomorphic-fetch », le package nous permettant d’utiliser l’API fetch pour faire des appels API.

Nous avons ensuite une méthode, « getMovies », retournant la liste des films au format json, ou loggant un message en console en cas d’erreur.

Installons donc le package isomorphic-fetch et profitons-en pour installer « redux-thunk », un package permettant les appels ajax asynchrones :

npm install --save isomorphic-fetch redux-thunk

Nous avons besoin d’une actionsTypes et d’un ActionCreator afin de faire cet appel dans les règles de l’art.

L’actionsTypes :

export const GET\_MOVIES = "GET\_MOVIES";

L’action va être séparée en deux. Premièrement, l’actionCreator, puis l’appel à l’API ainsi que le dispatch de l’actionCreator :

export const getMoviesSuccess = movies => ({

type: types.GET\_MOVIES,

movies,

});

export const getMovies = () => dispatch =>

api.getMovies()

.then(json => dispatch(getMoviesSuccess(json)),

);

N’oubliez pas d’importer votre api :

import \* as api from '../Api';

La bonne pratique voudrait que l’ont créé un fichier « moviesAction.js » pour cette action, mais dans un souci de flemme, je mets tout dans « helloAction.js ».

Nous avons donc « getMoviesSuccess » qui est notre actionCreator, et « getMovies » notre méthode qui va appeler l’API, récupérer les données et seulement ensuite dispatcher notre objet json.

Cette action, il faut la gérer, et c’est le reducer qui s’en occupe. Nous créons donc un fichier « MoviesReducer.js » et y créons un reducer :

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

export default function (state = {}, action) {

switch (action.type) {

case types.GET\_MOVIES:

return action.movies;

default:

return state;

}

}

(Attention, nous initialisons par défaut state en objet cette fois, car l’API que nous appelons nous renvoi un objet !)

Nous ajoutons ce reducer dans notre fichier « index.js » du dossier « reducer » avec la clef « movies » :

const allReducers = combineReducers({

hello: HelloReducer,

movies: MoviesReducer,

});

Enfin, et c’est le plus important ici, nous allons ajouter un middleware nommé « redux-thunk » nous permettant d’effectuer des appels ajax asynchrones. Grâce à ça, le retour des appels aux APIs externes seront attendu avant que le code suivant s’exécute. Rendez-vous dans le store et importez-y « redux-thunk » :

import thunk from 'redux-thunk';

Il faut aussi importer la méthode « applyMiddleware » du package « redux » :

import { createStore, applyMiddleware } from 'redux';

Et utilisez ces éléments comme suit :

const store = createStore(

allReducers,

applyMiddleware(thunk),

);

On appelle la méthode « applyMiddleware() » à l’intérieur de « createStore() ». Elle prend en paramètre un ou plusieurs middlewares, selon votre bon vouloir.

Il ne reste plus qu’à appeler notre action dans notre component. Pour cela, rien ne change par rapport aux actions synchrones : nous ajoutons l’action au component via le container et nous l’appelons comme une fonction dans le component. Nous n’oublierons pas les PropTypes ainsi que l’affichage.

Voici le container :

import { connect } from 'react-redux';

import { bindActionCreators } from 'redux';

import { sayHello, getMovies } from '../Actions/HelloActions';

import Hello from '../Components/Hello';

const mapStateToProps = state => ({

hello: state.hello,

movies: state.movies,

});

const matchDispatchToProps = dispatch => bindActionCreators({ sayHello, getMovies }, dispatch);

export default connect(mapStateToProps, matchDispatchToProps)(Hello);

Constatez l’ajout de « getMovies » dans les imports et dans la méthode binsActionCreators ainsi que la prop « movies » dans « mapStateToProps ».

Pour le component, on va ajouter une méthode « loadMovies() » :

loadMovies() {

this.props.getMovies();

}

Elle se contentera d’appeler notre action « getMovies ». Elle sera déclenchée par un bouton. Comme j’ai la flemme de faire l’affichage, je vais simplement afficher le résultat de notre requête en console. Voici donc notre méthode « render() » :

render() {

console.log(this.props.movies);

return (

<div>

<h1>{this.props.hello}</h1>

<input type="text" onChange={this.onChange} value={this.state.value} />

<br /><br />

<button type="button" onClick={this.loadMovies}>Load movies</button>

</div>

);

}

Et enfin, le plus important, les PropTypes :

Hello.propTypes = {

sayHello: PropTypes.func.isRequired,

getMovies: PropTypes.func.isRequired,

hello: PropTypes.string.isRequired,

movies: PropTypes.shape({

title: PropTypes.string,

description: PropTypes.string,

movies: PropTypes.arrayOf(

PropTypes.objectOf(

PropTypes.string,

),

),

}).isRequired,

};

Hello.defaultProps = {

movies: {},

};

Remarquez l’ajout de « getMovies » ainsi que « movies ». Ce dernier est décrit assez précisément afin de pouvoir mieux comprendre les messages d’erreurs encore une fois. En effet, il faut jouer avec les « isRequired » et les « defaultProps » (nécessaires lorsque « isRequired » n’est pas définie) afin de bien détailler l’objet que vous devez récupérer. Ça vous permettra de savoir si oui ou non vous connaissez bien ce que l’API doit vous renvoyer. Aussi, ça aide avec l’auto complétion du nom des props lorsque vous faites un affichage.

Notez que l’utilisation des objets n’est en rien différente d’un autre langage : vous taperez donc « movies.title » pour afficher le titre par exemple.

Notez également que je n’ai pas décrit en détail ce que me fournit l’API. J’ai simplement dit qu’il s’agissait d’un objet de string.

Le cycle de notre action asynchrone est donc maintenant terminé ! Si vous exécutez votre application, vous pourrez constater dans la console de développement (F12) que le clic sur le bouton vous permet d’obtenir les informations concernant la requête à l’API :



Aussi, vous pouvez voir que notre application nous retourne un objet vide « {} » lors du chargement du component. Il s’agit là de la valeur par défaut que nous avons donné dans notre reducer. Puisque nous n’exécutons pas l’action au chargement du component, la prop « movies » prend la valeur par défaut attribuée dans le reducer. S’il n’y avait pas ça, alors vous auriez eu une erreur dans votre console.

Ceci marque la fin de ce chapitre. Voilà.