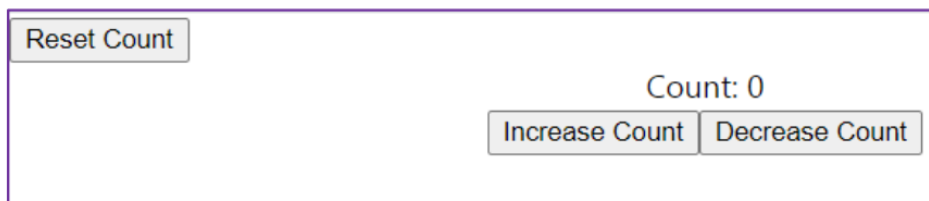
	PLB Consultant Formation React	
<b>Période :</b> Du 06 au 08 Juillet 2022		
<b>Formateur :</b> Mehdi M'tir		

## ReactJS

### Atelier : Intégration d'une SPA avec ReactJS et Redux

On se propose dans cette activité de créer avec React et Redux un simple compteur qu'on peut incrémenter, décrémenter ou le remettre à zéro.



#### 1. Création d'un nouveau projet react

- Créer un nouveau projet react intitulé my-react-redux-app via create-react-app :  
 > `npx create-react-app my-react-redux-app`
- Vérifier alors le lancement de l'application par défaut sur : `http://localhost:3000`

#### 2. Installation du redux et react-redux

- Les modules "redux" et "react-redux" sont deux modules npm qui s'installent au moyen de la commande suivante :  
 > `npm install redux react-redux --save`

#### 3. Création du Store

- Au niveau du fichier index.js, il faut d'abord créer le composant `<Provider>`, qui prend en **props** le store. Le rôle de ce composant est de distribuer le store à toute l'application.

```
import React from 'react';
import ReactDOM from 'react-dom';
import './index.css';
import App from './App';
import * as serviceWorker from './serviceWorker';

// Importation du module react-redux
import { Provider } from 'react-redux';
// Importation du fichier store
import store from './redux/store';
```

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

- Pour mieux organiser la structure de notre projet, créer un nouveau dossier nommé **redux** sous le répertoire **/src**.
- Pour créer le Store, créer un fichier javascript nommé **store.js** sous le répertoire **./src/redux**. (chemin : **./src/redux/store.js**)

```
import { createStore } from 'redux';
import rootReducer from './rootReducer';

//créer un store avec la fonction createStore
const store = createStore(rootReducer);
export default store;
```

On initialise le store en lui faisant passer le reducer.

- Créer un fichier **rootReducer.js** sous le même répertoire **./src/redux**

```
import { combineReducers } from 'redux';
import counterReducer from './Counter/counter.reducer';

//Combine les différents reducers en un seul reducer
const rootReducer = combineReducers({
  counter: counterReducer,
});
export default rootReducer;
```

La fonction `combineReducers()`, issue de la bibliothèque `redux`, comme son nom l'indique combine les différents reducers en un seul.

La fonction `counterReducer`, que nous n'avons pas encore créé, est un reducer de notre compteur.

#### 4. Configuration du reducer et des actions

- Au niveau du dossier `redux`, créer un nouveau dossier appelé `Counter` (chemin : **./src/redux/Counter**).

- Au niveau de ce dossier Counter, créer trois nouveaux fichiers :  
"counter.types.js", "counter.actions.js" et "counter.reducer.js"

- Dans le fichier `counter.types.js`, ajouter les deux types : `INCREMENT` et `DECREMENT` que nous aurons besoin respectivement pour incrémenter et décrémenter notre compteur.

```
//Constantes définissant les types d'actions
export const INCREMENT = 'INCREMENT';
export const DECREMENT = 'DECREMENT';
```

Nous devons exporter les deux constantes, afin qu'elles puissent être importées dans d'autres fichiers.

- Ajouter ensuite dans le fichier ***counter.actions.js***, les deux actions : ***increaseCounter*** et ***decreaseCounter***.

```
//Import des constantes d'actions
import { INCREMENT, DECREMENT } from './counter.types';
// Créateurs d'actions
export const increaseCounter = () => {
  return {
    type: INCREMENT,
  };
};
export const decreaseCounter = () => {
  return {
    type: DECREMENT,
  };
};
```

- Enfin, ajouter dans le fichier ***counter.reducer.js*** le reducer de notre compteur.

```
//Import des constantes d'actions
import { INCREMENT, DECREMENT } from './counter.types';

const INITIAL_STATE = {
  count: 0,
};
const reducer = (state = INITIAL_STATE, action) => {
  switch (action.type) {
    case INCREMENT:
      return {
        ...state, count: state.count + 1,
      };

    case DECREMENT:
      return {
        ...state, count: state.count - 1,
      };
    default: return state;
  }
};
export default reducer;
```

Nous avons déclaré une constante appelée `INITIAL_STATE` pour définir l'état par défaut de notre reducer. La fonction du reducer prend deux arguments : `state` et

action. Le state prend une valeur initiale de `INITIAL_STATE`, et l'action reçoit toutes les données transmises par notre créateur d'actions à partir du fichier ***counter.actions.js***.

Cette fonction est exportée (export default) de façon à être utilisée lors de l'instruction ***createStore(rootReducer)*** dans le fichier ***store.js***.

À l'intérieur de la fonction de reducer, nous avons utilisé l'instruction switch pour évaluer le type d'action (`INCREMENT` ou `DECREMENT`) et de retourner le state mis à jour. Nous avons utilisé l'opérateur (...) pour créer un nouveau state avec celui existant en faisant partie. (NB : nous ne modifions jamais l'état d'origine).

## 5. Création des boutons d'incrémentation et décrémentation avec JSX

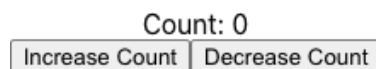
- Ouvrir le fichier App.js (chemin: ./src/App.js) et remplacer le code existant par celui-ci :

```
import React from 'react';
import './App.css';

function App() {
  return (
    <div className='App'>
      <div>Count: 0</div>
      <button>Increase Count</button>
      <button>Decrease Count</button>
    </div>
  );
}

export default App;
```

Dans votre navigateur, vérifier que votre compteur est à zéro et que les deux boutons : "Increase Count" et "Decrease Count" sont ajoutés.



## 6. Connexion du state et les actions de redux à un composant

- La dernière étape consiste à utiliser le state et les actions de redux au niveau de notre composant. Pour ce faire, modifier le code source du fichier ***App.js*** par celui-ci :

```

import React from "react"
import "../App.css"
import { connect } from "react-redux"

import {
  increaseCounter,
  decreaseCounter,
} from "../redux/Counter/counter.actions"

function App(props) {
  return (
    <div className="App">
      <div>Count: {props.count}</div>
      <button onClick={() => props.increaseCounter()}>
        Increase Count
      </button>
      <button onClick={() => props.decreaseCounter()}>
        Decrease Count
      </button>
    </div>
  )
}

const mapStateToProps = state => {
  return {
    count: state.counter.count,
  }
}

const mapDispatchToProps = dispatch => {
  return {
    increaseCounter: () => dispatch(increaseCounter()),
    decreaseCounter: () => dispatch(decreaseCounter()),
  }
}

export default connect(mapStateToProps, mapDispatchToProps)(App)

```

La méthode ***connect()*** permet de connecter les composants souhaitant récupérer des données du store ou souhaitant y dispatcher des actions pour altérer ces données.

Cette méthode prend deux fonctions comme arguments : ***mapStateToProps*** et ***mapDispatchToProps***.

Il s'agit de deux fonctions qui vont respectivement parcourir le state de l'application et les actions à dispatcher, et qui vont mettre à jour le store puis les passer aux props de notre composant React.

Finalement, vérifier que votre application est entièrement fonctionnelle.

## **7. Remise à zéro du compteur**

En adoptant la même démarche et en créant cette fois-ci un nouveau composant (`components/reset.js`), faire le nécessaire afin d'ajouter un troisième bouton Reset à l'application permettant de mettre à zéro le compteur.