Principe Redux

- Redux gère l'état général de l'application.
- Redux ne possède qu'un seul store général qui gère les états.
- Le changement d'un état est déclenché exclusivement à l'émission d'une action c'est à dire d'un événement.
- Le state dans Redux est **immutable** il ne peut pas être modifié, le fonction reducer() retourne toujours un nouveau state.
-

Exemple :

On doit émettre un simple objet qui est une action et qui doit obligatoirement comporter une propriété type qui définit l'action.
on peut ensuite lui mettre les propriété que l'on veut, sauf des fonctions.

```
/*

{
    type: "ADD_ARTICLE",
    payload: article
}

{
    type: "EDIT_ARTICLE",
    payload: article
}

{
    type: "REMOVE_ARTICLE",
    payload: article
}

*/
```

- Les changements d'état sont fait par des **pure functions** autrement dit seul un **reducer** pourras modifier l'état de notre application.
- Un reducer prend en paramètre >(le state courant, une action) et renvois un nouveau state.
- Une "pure fonction" c'est une fonction donné qui pour une entrée donné retournera toujour la même sortie, et qui n'aura pas d'effet tiere(side effects) du style appel d'un web service.

exemple de pure function :

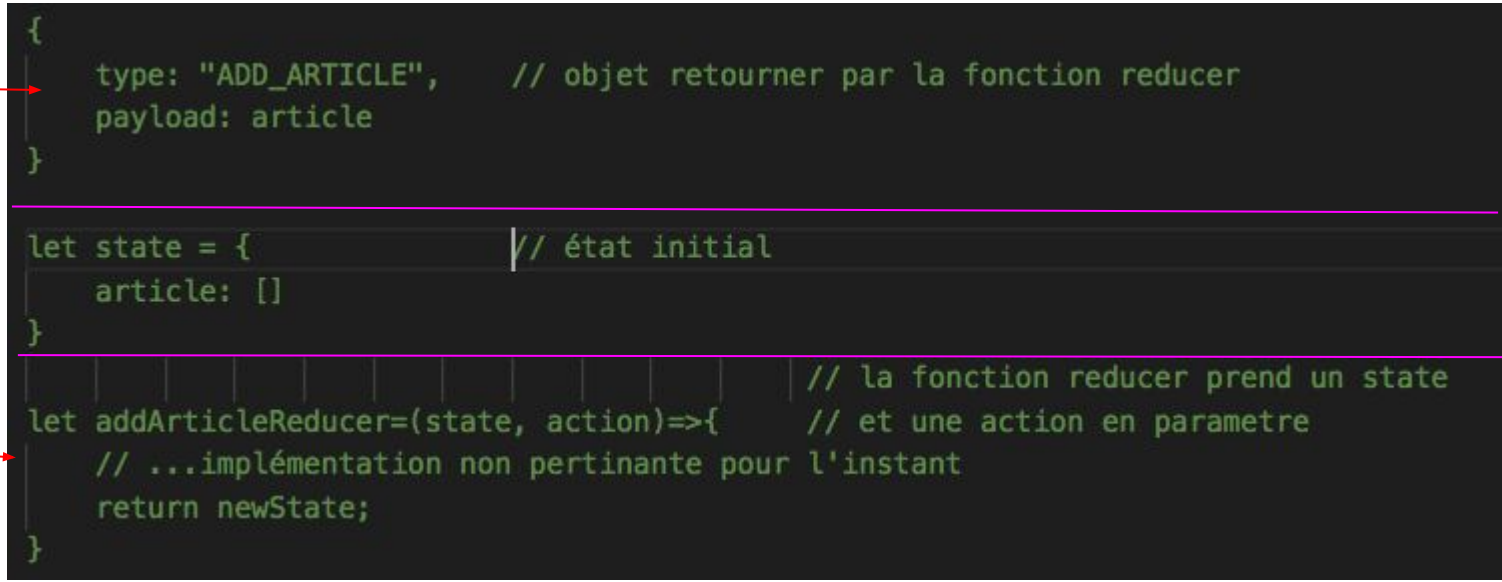
```
// pure function
add = (a, b)=>{     // si on met add(1,2) la fonction nous retournera toujours 3.
    return a+b;
}

// impur function
random = (a) =>{    // si on met random(1) la fonction nous retournera des nombres différents.
    retunr math.random() * a;
}
```

Exemple concret

Objet retourner par la fonction reducer.

L'état courant de notre composant.

Exemple de fonction reducer.



```
{
    type: "ADD_ARTICLE",    // objet retourner par la fonction reducer
    payload: article
}

let state = {               // état initial
    article: []
}

                                        // la fonction reducer prend un state
let addArticleReducer=(state, action)=>{    // et une action en parametre
    // ...implémentation non pertinante pour l'instant
    return newState;
}
```

Principe de Single Responsibility

- Avec Redux le store ne comporte aucune logique.
- Le store a pour mission unique de comporter des effets successif.
- La logique est de la responsabilité des reducers().

- On a un reducer qui gère le state de chaque properties dans un state.

exemple :

```
let state: {
    articles:[],            // un reducer pour la properties article
    outlet: [],             // un reducer pour la properties article
    lastArticleUpdate: ''   // un reducer pour la properties article

}
```

Note importante : Quand un reducer est dispatcher tous les reducer() entre en action.
Ce qui veut dire que l'on va utiliser un switch, et que tous les reducers non concerné qui rentre quand même en action quand un des reducers du store est dispatché retournerons un state inchangé.

Il retournerons par le états inchangé par défaut.