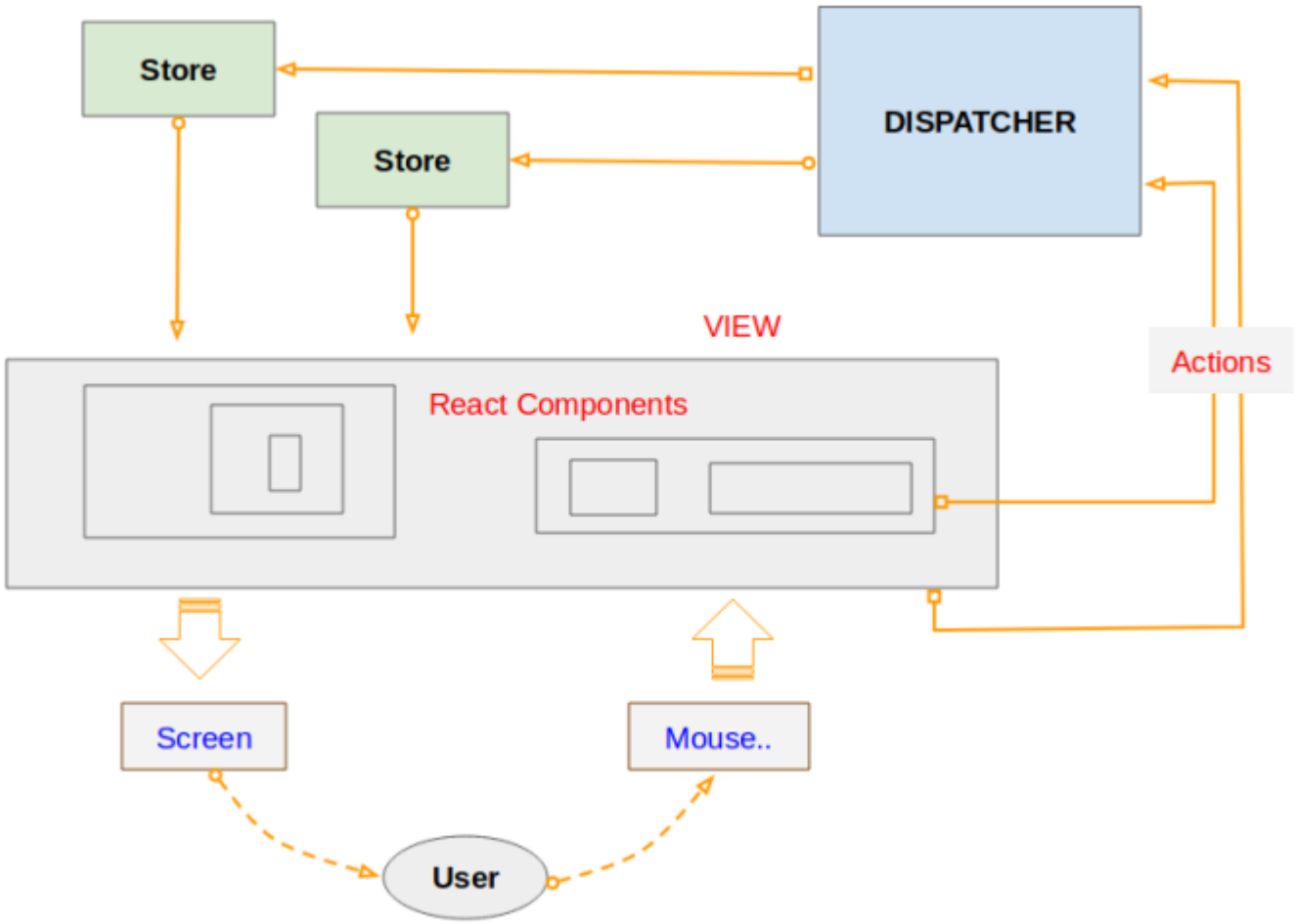
Introduction :

- ReactJS était seulement une bibliothèque pour créer des Component et rendre ces Component sur l'interface.
- ReactJS n'avait pas la capacité de gérer l'état des applications. Après, Facebook a introduit une bibliothèque Javascript nommé **Flux** pour gérer l'état des applications et elle est une bibliothèque créée pour soutenir le React.
- Après sa naissance, le **Redux** a provoqué un vif écho et a immédiatement attiré l'attention de la communauté React. Actuellement, le Redux et le Flux existent en parallèle, mais le Redux est populaire et plus largement utilisé.

ARCHITECTURE FLUX



Lorsqu'un utilisateur exécute une action sur l'interface (Par exemple clique sur un bouton). Un objet Action sera créé, ceci est un objet normal, il contient des données telles que : Type d'action, où se déroule, quand, location (correspondant à l'événement de bouton), remplacez state ,..

View : une composition hiérarchique des React Component.

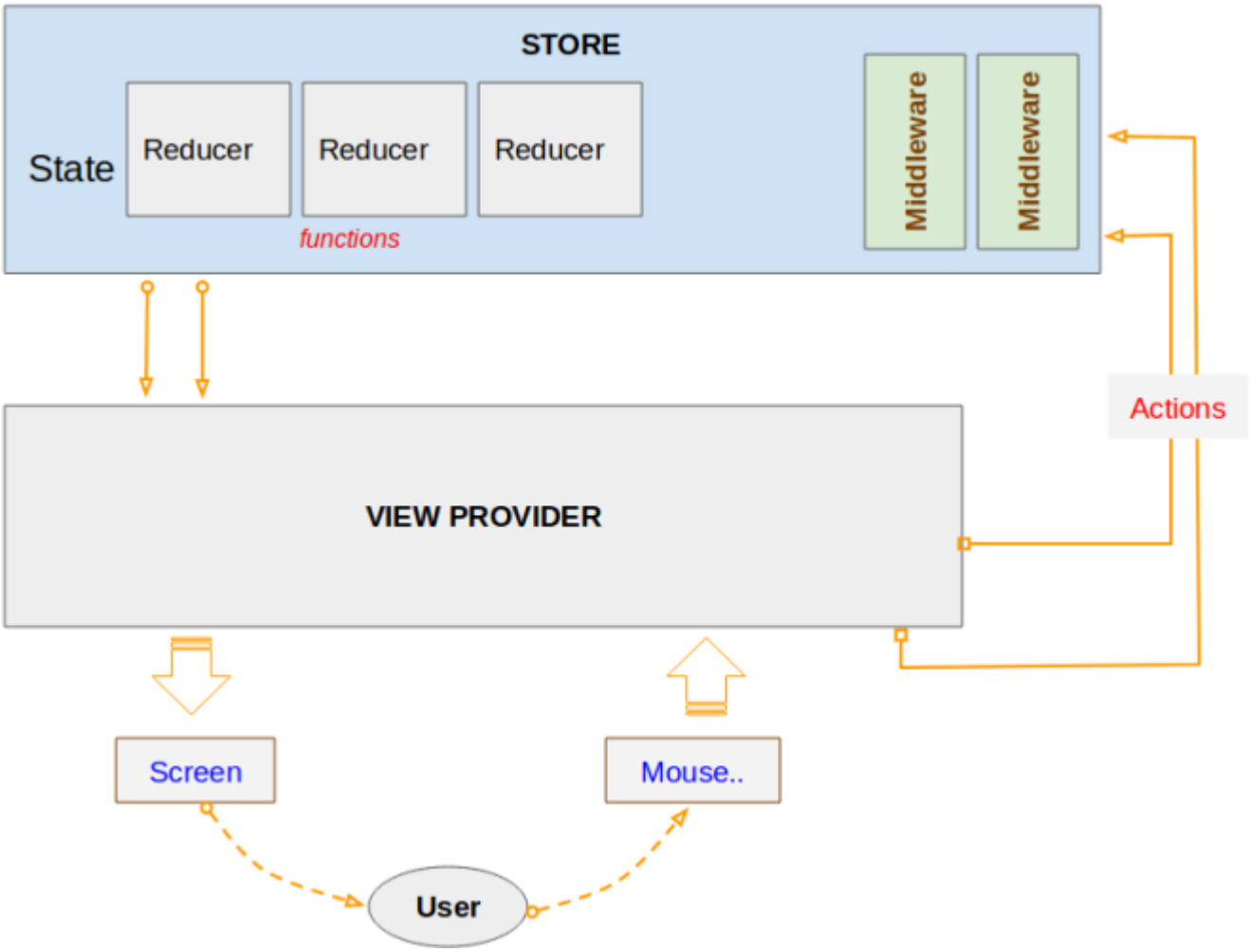
Action : un objet pur créé pour stocker les informations relatives à un événement de l'utilisateur (Clique sur l'interface ,..), il comprend les informations telles que : type de l'action, temps et location, ses coordonnées et quel state qu'il vise à changer.

Dispatcher : Le seul point de l'application à recevoir des objets Action à gérer.

Store : Store écoute des Action, gère des données et l'état de l'application. Les Store basent sur les objets action pour répondre au **user interface** correspondant.

ARCHITECTURE REDUX

- Redux apprend l'architecture de Flux mais il omit la complexité inutile.
- Redux n'a pas de concept **dispatcher**.
- Redux n'a que **store** au lieu de plusieurs **store** comme le Flux.
- Les objets Action sera reçus et gérés directement par **store**.



View provider : représente un View Framework pour inscrire avec **store**. Dans lequel, View Framework peut être React ou Angular ...

Action : un objet pur créé pour stocker les informations relatives à l'événement d'un utilisateur (cliquez sur l'interface, ...). Il inclut les informations telles que : le type d'action, l'heure de l'événement, l'emplacement de l'événement, ses coordonnées et quel state qu'il vise à modifier.

Store : Gère l'état de l'application et la fonction dispatch(action).

Middleware : (Logiciel intermédiaire) fournit un moyen d'interagir avec les objets Action envoyés à **store** avant leur envoi à **reducer**. Avec Middleware, on peut effectuer des tâches telles que la rédaction de journaux, la génération d'erreurs, la création de requêtes asynchrones (asynchronous requests) ou la distribution (dispatch) de nouvelles actions, …

reducer : (Modificateur) Une fonction pure pour renvoyer un nouvel état à partir de l'état initial.

Remarque :

Reducer ne modifie pas l'état de l'application. Au lieu de cela, il créera une copie de l'état d'origine et le modifiera pour obtenir un nouvel état.

PRATIQUE

1*/ Installation et importation

> npm install redux react-redux

```
// src/index.js
import React from 'react';
import {createStore} from 'redux';
import {Provider, connect} from 'react-redux';
```

2*/ Création d'un reducer

Un reducer accepte 2 paramètres : état et action. L'état peut être n'importe quoi (entier, chaine, objet ,..) L'action est un objet avec une propriété type sous forme d'une chaine.

Exemple : countReducer dans src/index.js

```
// reducer
const countReducer = function (state=0,action){
    switch(action.type){
        case "INCREMENT":
            return state+1
        case "DECREMENT":
            return state-1
        default :
            return state
    }
}
```

3*/ Création d'un magasin Redux (store)

```
// store
let store = createStore(countReducer);
```

4*/ Ajout des boutons au composant

```
// composant
const Component = ({count, handleIncrementClick, handleDecrementClick}) => (
    <div>
        <h1>Coucou React Redux {count}</h1>
        <button onClick={handleDecrementClick}>décrémenter</button>
        <button onClick={handleIncrementClick}>incrémenter</button>
    </div>
);
```

5*/ Le container :

- Mappe (relie) : Sélection et conversion de l'état de redux store
- Mappage de la répartition du magasin sur les fonctions de rappel

```
// container
const mapStateToProps = state => {
    return{
        count : state
    }
}
const mapDispatchToProps = dispatch => {
    return {
        handleIncrementClick: () => dispatch({ type : 'INCREMENT'}),
        handleDecrementClick: () => dispatch({ type : 'DECREMENT'})
    }
}
const Container = connect(mapStateToProps,mapDispatchToProps)(Component);
```

6*/ Le composant principal :

import {Provider} : connexion de Redux à react à l'aide de la bibliothèque react-redux

```
// App
const App = () => (
    <Provider store={store}>
        <Container />
    </Provider>
);
```

Le <Provider/> fournira l'ensemble de l'arborescence de composants avec l'arborescence d'état globale.

7*/ Rendu de la page

```
ReactDOM.render(
    <App />,
    document.getElementById('app')
);
```

APPLICATION : Refactoriser le code

Déplacer le code similaire dans des fichiers séparés pour garder le projet bien rangé et maintenu : le composant Container, le composant Presentational, l'initialisation du magasin et le réducteur. La structure finale du répertoire devrait ressembler à ceci :

```
src
├── configure-store.js
├── counter
│   ├── component.js
│   ├── container.js
│   └── reducer.js
└── index.js
```