Redux

1. Contenedor predecible del estado de las App JS

2. ReactJS (V) + Redux (MC) = MVC (Modelo - Vista – controlador)

3. Redux lleva el estado de las App y este es almacenado en un único store

4. Para cambiar el estado hay que emitir una acción

5. Una acción está compuesta por una descripción de la tarea (string constante) más los datos

6. Las acciones transforman el árbol de estado a través de los "Reducers" (son funciones puras)

7. El estado no se modifica directamente, se hace a través de Reducers mediante acciones

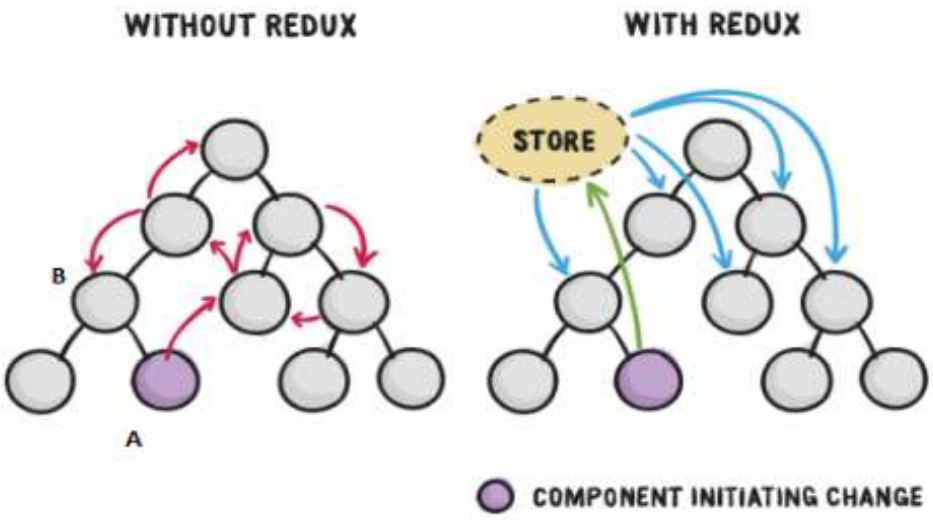
8. Cuando la App crece, no se agregan más stores, sino se agregan más Reducers (árbol)

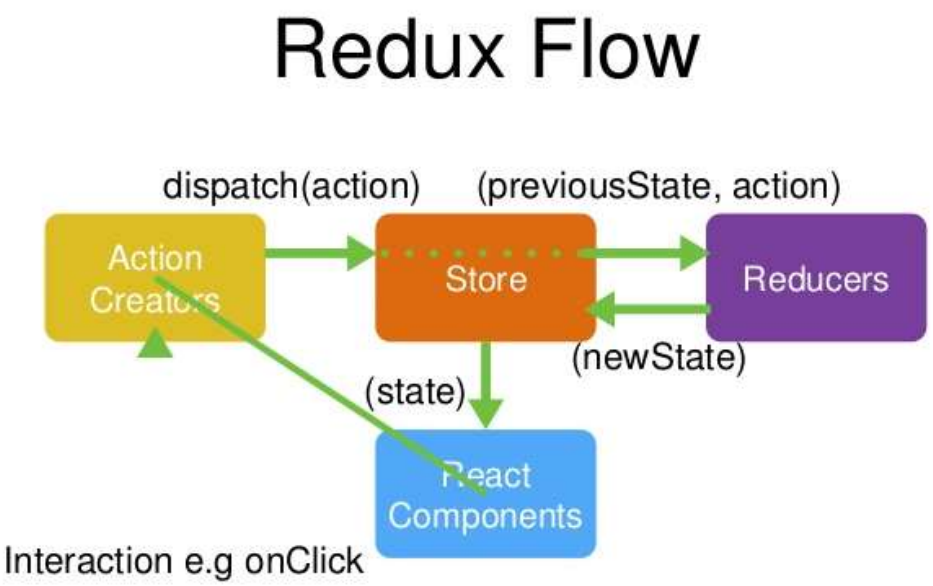
Principios Redux

Estado
1. Único store
2. Información fácilmente serializable (servidor y su cliente)

Estado es de sólo lectura
1. La acción es la que modifica el estado
2. Las acciones pueden ser almacenadas

Funciones Puras
1. Reducers Puros (no modifican los parámetros de entrada)
2. Los reducers tienen como entrada el estado anterior de la App y la acción a realizar, la salida será el estado nuevo.





Acciones

1. Son bloques de información que envían datos de la App al store mediante el método dispatch

2. La acción es un objeto plano de JS

```
{
	type: 'SUMAR',
	operando1: 1,
	operando1: 3
}
```

Creadores de Acciones

1. son funciones que crean acciones

Modo a)

```
function addTodoWithDispatch(numero1, numero2) {
      const accion = {
            type: 'SUMAR',
            operando1 : numero1,
            operando2 : numero2
      }

      dispatch(accion);
}
```

Modo b)

```
function addTodo(numero1, numero2) {
      return {
            type: 'SUMAR',
            operando1 : numero1,
            operando2 : numero2
      }
}

addTodo = (numero1, numero2) => ({
            type: 'SUMAR',
            operando1 : numero1,
            operando2 : numero2
})

dispatch(addTodo(1, 2));
```

2. Dispatch disponible a) Desde store.dispatch
                        b) connect()

Reducers

1. Son funciones puras que toman el estado anterior y la acción a realizar y como resultado produce un nuevo estado.

2. Los reducers
   a) Nunca modifican los argumentos
   b) Nunca generan efectos secundarios (llamadas a APIs)
   c) Nunca deben llamar a una función no pura

3. Sólo hacen cálculos simples (nada extraño)

4. Ej.

```
functionReducerAddTodo(state={}, action) {
      switch(action) {
            case 'SUMAR':
                  return Object.assign({}, state, { ... })
                  ...

            defaults:
                  return state;
      }
}
```

5. En un reducer no se modifica el state
6. Se hace una copia del estado y según la acción se actúa sobre los datos
7. Se retorna el estado modificado

Store

Component => acciones => Dispatch => Reducers => Estado (store)

.getState() => permite leer el store, permite acceder a sus objetos

.dispatch(acción) => Se utiliza para actualizarlo

.suscribe() => Se pueden escuchar los cambios producidos sobre el store (Listeners)