REDUX

. ¿Qué es Redux?

No es más que un simple patrón para el manejo de la información. No es solo aplicable a angular, de hecho, es un patrón muy común utilizado en aplicaciones con REACT. Pero lo podemos utilizar en cualquier aplicación que utilice vainilla JavaScript plano o VueJS por ejemplo

. ¿Para qué sirve?

* Cuál es el estado de la aplicación
* Cómo se encuentra determinada variable
* Cómo o quién cambió alguna variable
* Cómo cambió la información y en qué momento

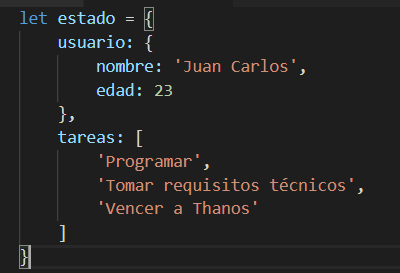
. ¿Cuándo no es conveniente usarlo?

En aplicaciones pequeñas

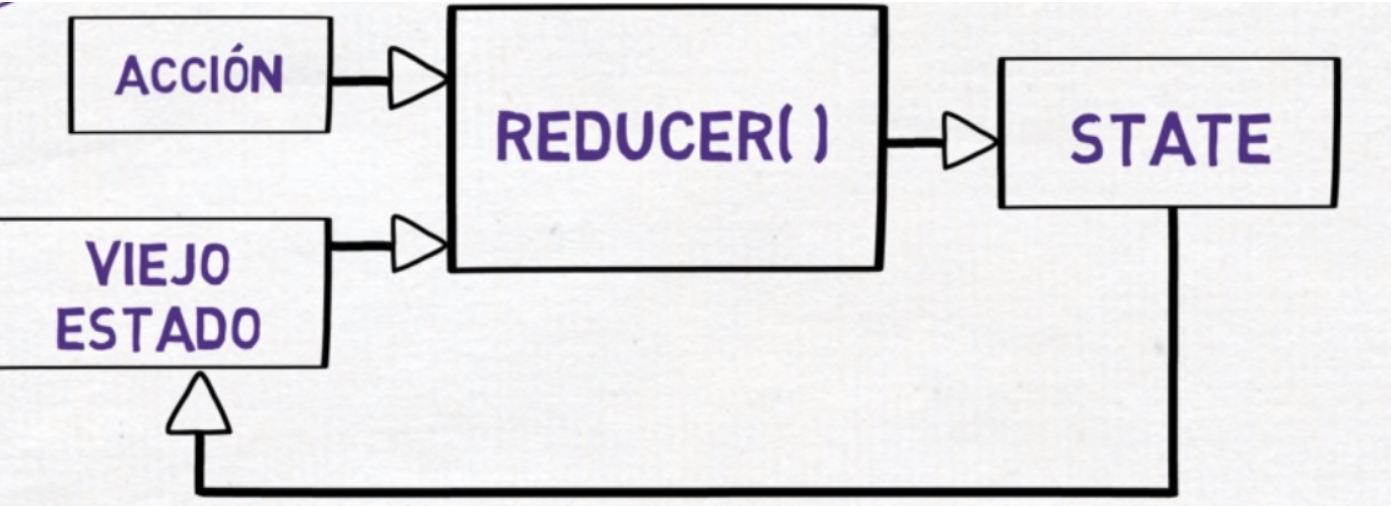
IDEAS PRINCIPALES

* Todos los datos de la aplicación, se encuentra en una estructura previamente DEFINIDA
* Toda la información se encontrará almacenada en único lugar llamado STORE
* El store JAMAS se modifica de forma directa
* Cualquier interacción de usuario y/o código, dispara acciones que describen qué sucedió
* El valor actual de la información de la aplicación se llama estado – STATE
* Un nuevo state es creado, en base a la combinación del viejo state y una acción que realiza una función llamada REDUCER

Ejemplo de estado – STATE:



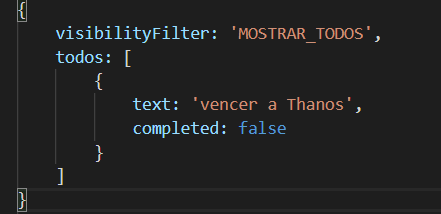
Ahora digamos que yo ya he vencido a thanos y quiero cambiar el estado, primero tendría que lanzar una **ACCIÓN** que sería lo que tendría la información que describe lo que voy a hacer. A continuación cojo la **ACCIÓN Y MI VIEJO ESTADO** y se lo tendría que pasar al **REDUCER()** que es solamente una función que recibe los argumentos, como si por ejemplo en vez de llamarse REDUCER() se llamase HOLAMUNDO(acción, viejo estado). Esta función produce un nuevo estado – state.



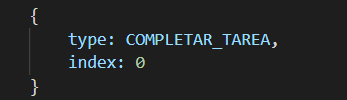
La ventaja de utilizar REDUX es que, si la aplicación crece mucho y perdemos el “control” de esta misma, este patrón es muy útil para resolver este problema, frente al simple servicio – service -> componente – component de angular

LAS 4 ESTRUCTURAS IMPORTANTES DEL PATRÓN REDUX (ACTION – REDUCER – STATE- STORE)

* **ACTION:** Es la única fuente de información que se envía por interacciones de usuario o programa. Por lo general se busca que las acciones sean lo más simples posibles. Tiene únicamente 2 propiedades:
  + **Type -**> Que nos describe que es lo que quiere hacer o la acción que va a realizar
  + **Payload -**> Es opcional. No siempre nosotros vamos a tener que enviar información para realizar alguna acción
* **REDUCER:** Recibe siempre 2 argumentos y siempre debe de retonar un state. Los dos argumentos que recibe el reducer son:
  + **oldState -**> Es el estado actual de la aplicación



* + **action ->** Es un objeto plano que indica que hay hacer



* **STATE:** Tiene ciertas reglas que debemos tomar en cuenta, como, por ejemplo:
  + - **El state es de sólo lectura**
    - **Nunca se cambiará el state de forma directa**
    - **Hay funciones que están prohibidas de JavaScript (Push, o la manipulación directa del objeto oldState) en difinitva cualquiera cosa que modifique ARRAYS O OBJETOS de JS, ya que en JS los objetos se pasan por referencia siempre y queremos siempre que nuestro state sea inmutable, y por lo tanto generar nuevos states**
* **STORE:** Es un objeto que tiene las siguientes responsabilidades:
  + - **Permite la lectura del state vía:** getState**()**
    - **Permite crear un nuevo estado utilizando:** dispath**(ACTION)**
    - **Permite notificar los cambios vía subscribe -> (cómo ya estamos acostumbrados con el observable de angular y rxjs)**

INFORMACIÓN REFERENCIAL:

<https://redux.js.org/introduction/getting-started>

La comunidad de redux en español es grande y también tienen traducida la página de manera bastante buena en https://es.redux.js.org

**PROYECTO EXPLICATIVO CON NODE.JS TANTO PARA EL USO EN JS COMO EL USO EN TYPESCRIPT**

[**https://github.com/jcguerrero21/REDUX-EXPLICACION-VAINILLAJS**](https://github.com/jcguerrero21/REDUX-EXPLICACION-VAINILLAJS)

PATRÓN REDUX EN ANGULAR

Supongamos que nosotros vamos a hacer una aplicación que tenga un componente padre, que a su vez se comunique con un componente hijo y este a su vez se comunique con un componente nieto y todos ellos compartan un variable que cambie en todos los lugares simultáneamente cuando nosotros realizamos una acción.



Tendríamos que hacerlo utilizando Inputs, Outputs y sus EventEmitter o bien creando un servicio que lo hiciese. Sin embargo, si utilizásemos el patrón **REDUX** el cuento cambiaría.

Código fuente ejemplo aplicación Angular(SIN-REDUX): <https://github.com/jcguerrero21/REDUX-BASICO-ANGULAR/releases/tag/SIN-REDUX>

Podemos buscar toda la información acerca de REDUX en angular en su librería de GitHub o en su página web

<https://github.com/ngrx/platform>

<https://ngrx.io/>

Explicación utilizando REDUX:

<https://github.com/jcguerrero21/REDUX-BASICO-ANGULAR/releases/tag/CON-REDUX>