NgRx
Ce qu'on a compris (ou pas)

Eric, Delmerie, Caroline & Sarah

SOMMAIRE

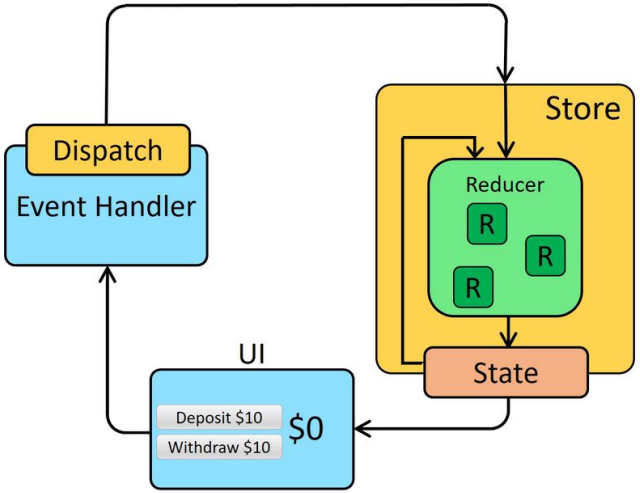
- Historique de NgRx
- Utilité de NgRx
- Programmation réactive
- Store/State
- Reducer
- Effect

Ng = Angular
Rx = RxJs

Historique

Ngrx est une librairie fournissant les éléments (actions, dispatcher, reducer ...etc.) pour une architecture "réactive" type state management, basée sur Redux.

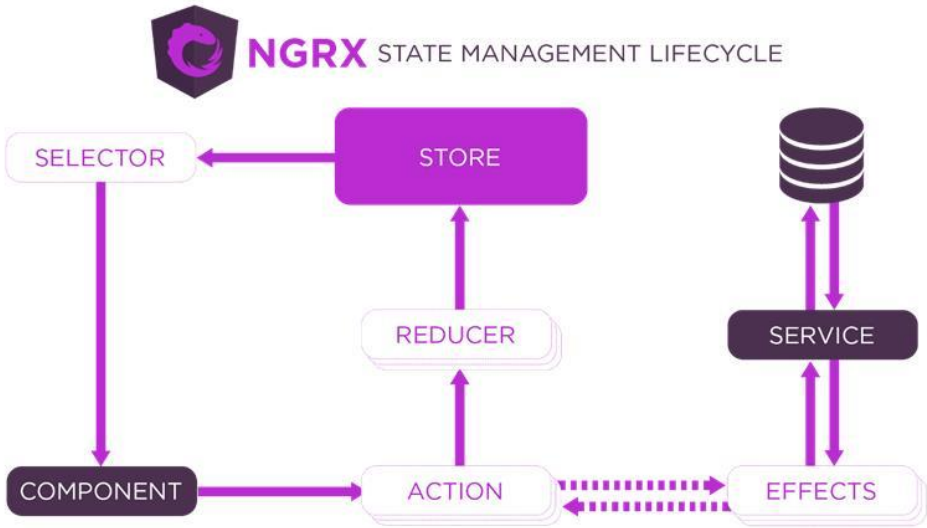
Elle fonctionne avec RxJS (Observable), implémentée avec Typescript donc adaptée pour Angular.



Architecture Redux

NgRx : quelle utilité ?

- Pour les grosses applications
- Fais gagner du temps au débuggage
  (quand on maîtrise son sujet)
- Chaque élément a un rôle bien défini
  (ouvert aux modifications futures)



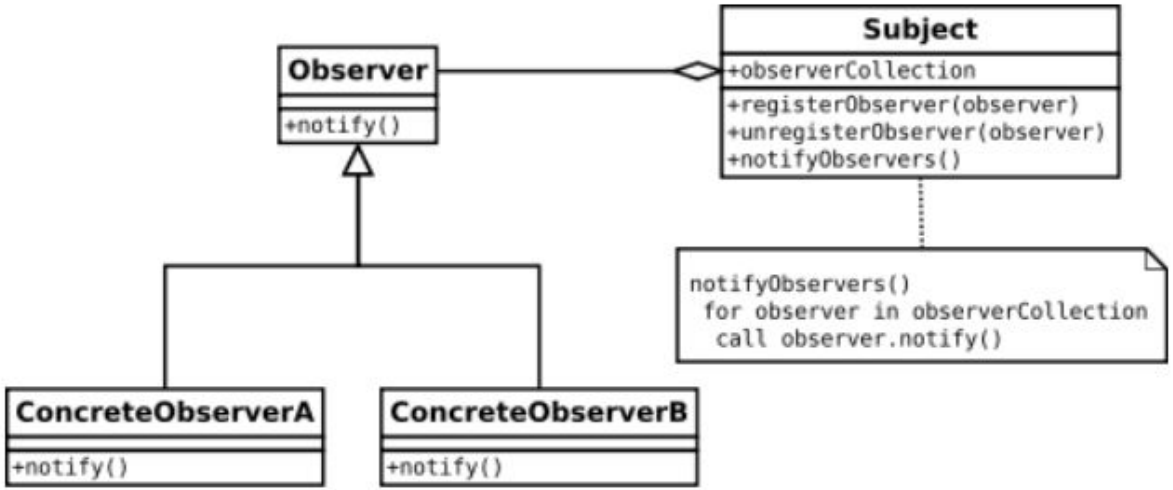
La programmation réactive

- RxJS : bibliothèque de programmation réactive utilisant des *Observables* pour faciliter la composition de code asynchrone

- RxJS permet de gérer les séquences d'événements.

- Un événement se produit par exemple lorsque l'utilisateur ou le navigateur modifie d’une quelconque façon une page.

Les observables



Les observables

→ next() est appelé à l'arrivée d'un élément

→ error() : lorsqu'une erreur survient

→ complete() est appelé en fin de séquence

```
1  import { Observable } from 'rxjs';
2
3  const data$ = new Observable(observer => {
4
5      observer.next(1);
6      observer.next(2);
7      observer.next(3);
8      observer.complete();
9
10 });
11
12 data$.subscribe({
13     next: value => console.log(value),
14     error: err => console.error(err),
15     complete: () => console.log('DONE!')
16 });
```

Les observables

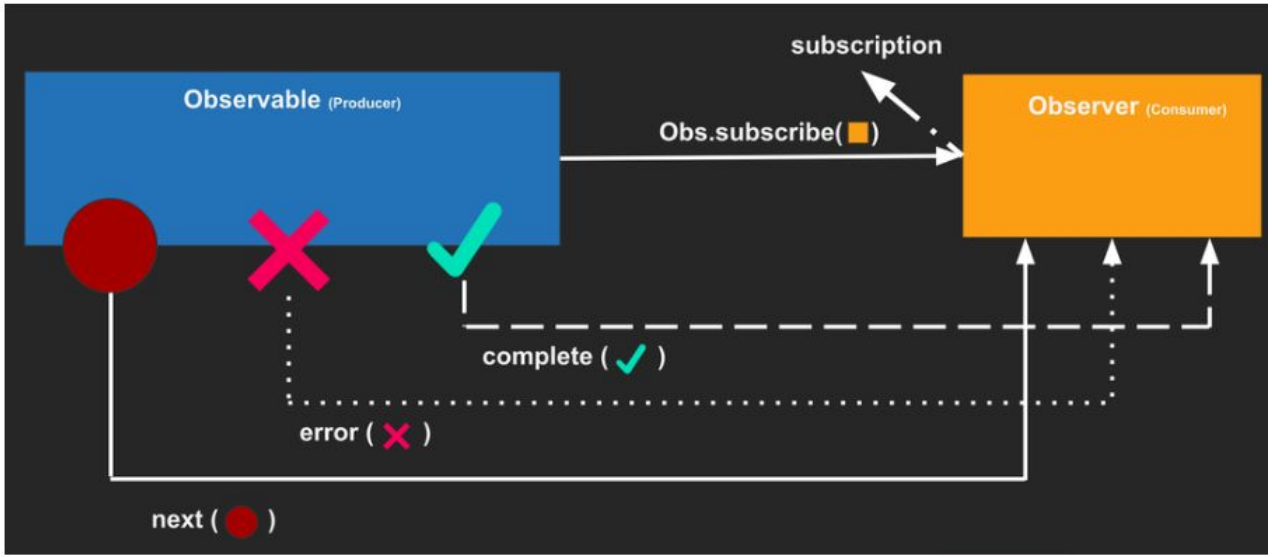


Schéma de souscription à un observable avec la méthode `subscribe`

Les observables : les opérateurs

⟶ Pipe : pour chaîner les opérateurs

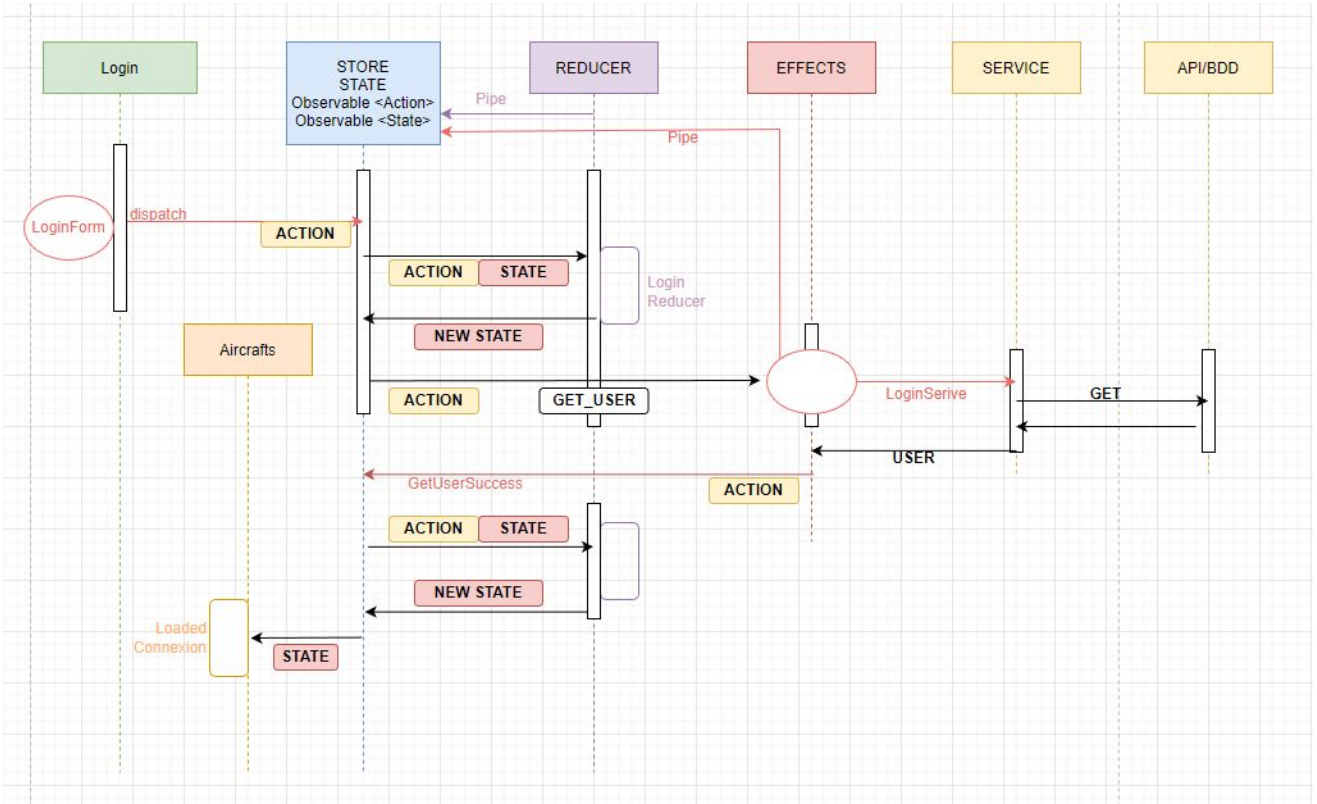
⟶ Tap :

```
monObservable.pipe(tap(item => console.log(item)).subscribe();
```

⟶ Map : pour transformer ce qu'il y a dans le pipe

```
monOservable.pipe(map(item => item + 1)).subscribe(item =>
console.info(item));
```

Hiérarchie de la libraire

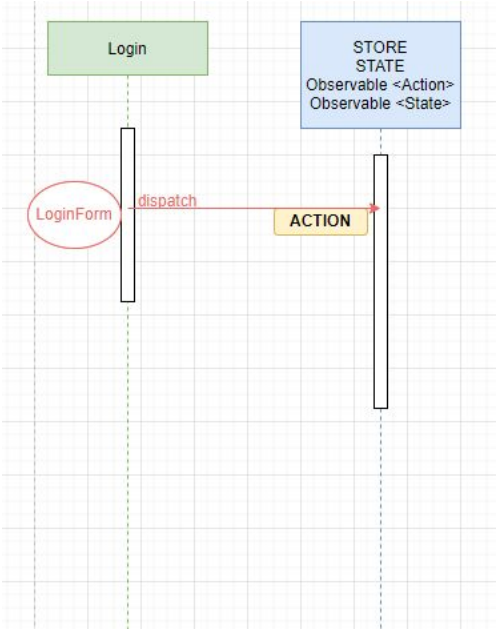


Store

Il fait le lien entre tous les acteurs de l'application.

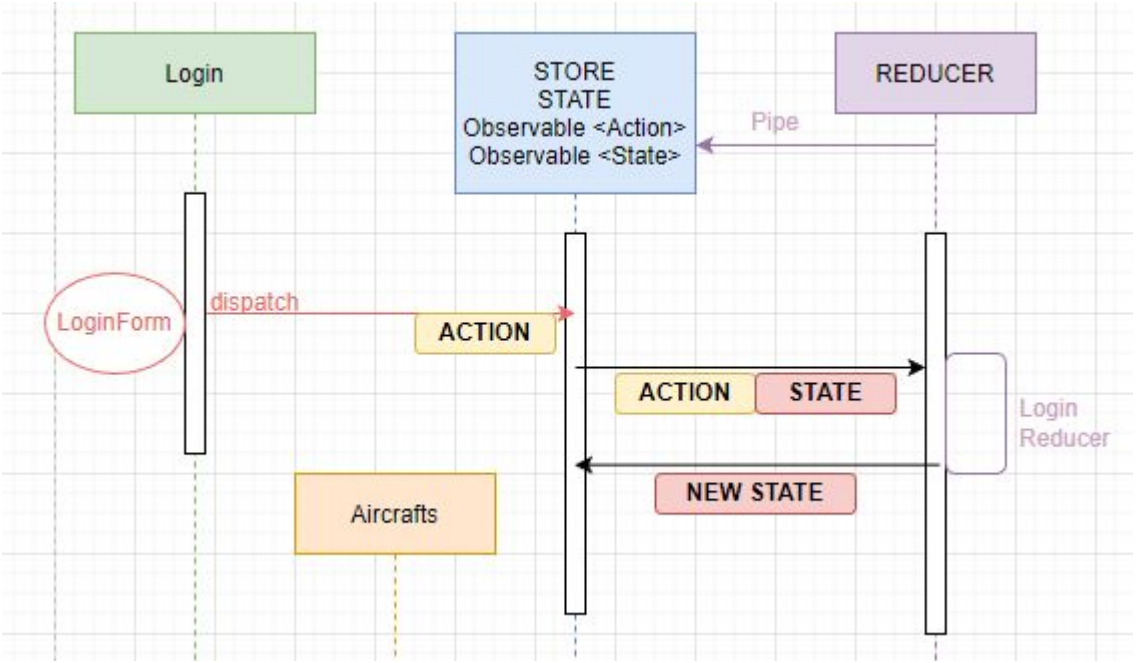
Il contient le **State** qui constitue l'état global de l'application à un instant T, il conserve les données que le **Reducer** lui renvoie.

Action = Exprime des événements uniques qui se produisent dans l'application.



Reducer

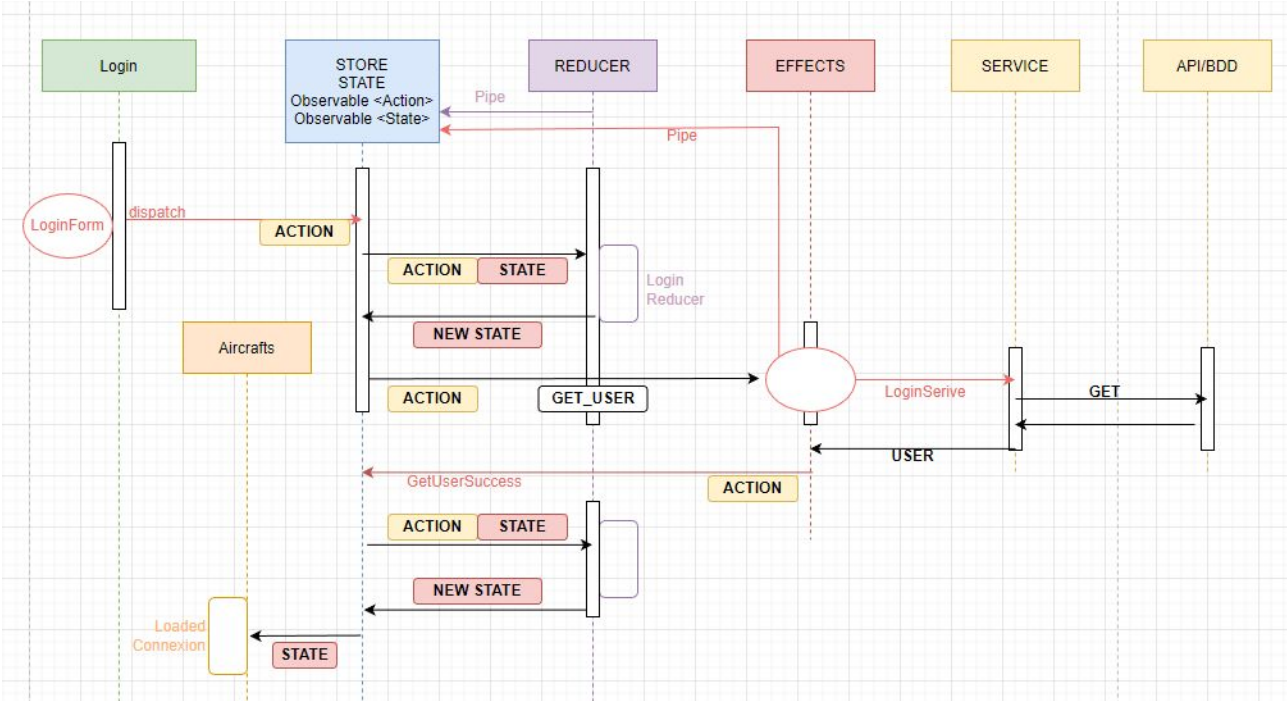
Gère l'état du **State** et lui renvoie les données par rapport à l'**Action** demandée.



Effect

Écoute le changement de **State** et fait l'action. Il renvoie les bonnes infos au **State**, qui reprend ensuite son chemin vers le **Reducer**.

Selector = Il est facultatif. Permet de faire des requêtes et de renvoyer des données à l'**Effect**.



Sources

- Explication Redux : https://dev.to/codebucks/what-is-redux-simply-explained-2ch7
- https://guide-angular.wishtack.io/angular/observables/creation-dun-observable
- https://www.learn-angular.fr/comprendre-rxjs/
- https://blog.ippon.fr/2021/10/25/introduction-a-rxjs/