Fms Academy 2022
Formation Java Spring Angular
M09.04 : Angular/NgRx

mohamed.el-babili@fms-ea.com

Sommaire

Introduction

- Pré requis
- Programmation Synchrone ou asynchrone
- RxJs
- NgRx
- Install Node & Angular

Projet Angular

- Bdd avec Json Server
- Page d'accueil
- Modèle
- Service
- Ajout d'un composant
- Gestion du composant
- Ajout d'un composant de navigation
- Gestion du composant de navigation
- Synthèse et problématique
- Service centralisé de gestion des évènements

Exploitation de NgRx

- Install/dépendances
- Actions
- State
- Reducer
- Effects
- Component
- Synthèse
- Test Appli + Redux Devtools
- Ajout fonctionnalités
- Selectors
- Entity

Conclusion

- Bilan
- Ressources

Pré requis

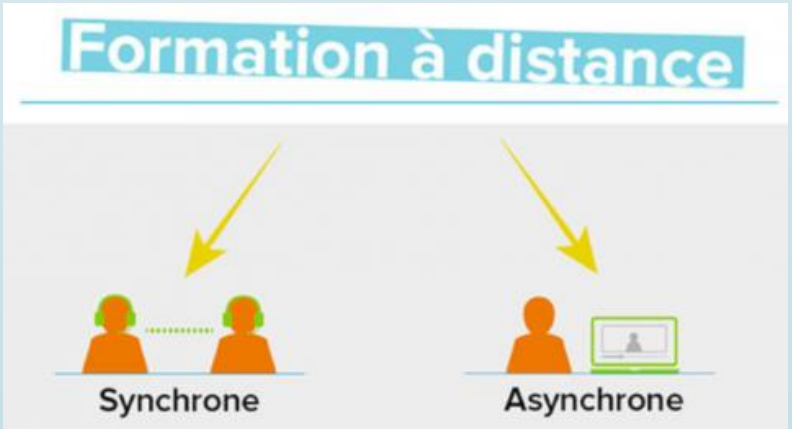
Liste des pré requis pour suivre ce cours dans les meilleures conditions :

- Typescript(...arrow function, subscribe) + POO (class, interface, généricité…) + Algorithmique + Uml (diagramme de séquence)
- Web : Html, Css, Dom, Event, Bootstrap, Node/Npm
- Angular : module, composant, service, routage, com, programmation reactive, Injection des dépendances, Décorateurs…
- Design pattern en général + Programmation synchrone/asynchrone + Protocole http (get/post) + Base de donnée/Json/Api…

En résumé, l'idéal reste d'avoir suivi les modules précédents !

Programmation
Synchrone ou asynchrone

Savez vous pourquoi Teams Freeze ?

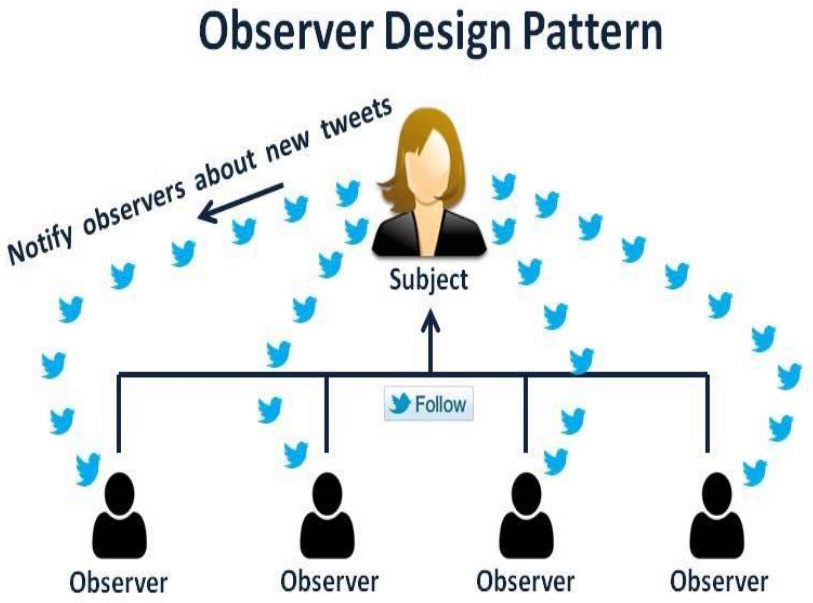


RxJs

- *Reactive extensions for Javascript* est une librairie permettant de travailler sur des **flux de données asynchrones** représentés à l'aide de **séquence d'observables**.
- Il faut être à l'aise avec les **Arrow functions** (fonctions anonymes ou expressions lambda) pour aborder sereinement la notion **d'Observables !**
- En effet, on parle ici de **programmation réactive** qui se base sur le **Design pattern Observer** avec l'idée que des **observables** (subject) vont émettre des évènements asynchrones, interceptés par les **observateurs** (observers)

Voir dossier labo/manip opérateurs…

```
const data$ = new Observable(observer => { // on définit ici un observable (qui peut être observé)
  observer.next('first call' + counter++);   //voici l'appel qui sera sollicitée par les observateurs ayant souscrit
  setTimeout(() => {
    observer.next('second call' + counter++); //2ème appel 5 secondes plutard ...
  }, 5000);
});

let counter = 0;
const observer1 = data$.subscribe({   //l'observateur1 souscrit à l'observable et gère 2 cas :
  next: value => console.log('1-' + value),  //chaque fois qu'une valeur est émise par l'observable, il l'affiche
  error: err => console.log('1-' + err),     //affichage en cas d'erreur
});

//et puis un autre observateur sur le même Observable
data$.subscribe((value) => {
  console.info('2--' + value);
});

//création d'un observable à partir d'un tableau de string
const dataTable$ = from(['monday','tuesday','wednesday', 'thursday' , 'friday' , 'saturday' , 'sunday']);
dataTable$.subscribe(val => console.log(val));  //souscription à celui-ci

//voilà donc un tuyau qui permet d'appliquer plusieurs traitement via les opérateurs tap, map, ...
//chaque opérateur ou fonction renvoi un observable qui sera traité par l'opérateur suivant
data$.pipe(
  tap(item => console.log('3---' +item)),     //1ère affichage + renvoi de la source
  map(dat => dat + ' end')                    //map : renvoi un observable avec chaque donnée transformée
).subscribe(                                  //souscription à celui-ci
  dis => console.log('hi ' + dis)
);

//of crée un Observable sur une liste de chiffres qui passent par un tuyau qui filtre(pair) puis modifie les datas avec map
//avant de renvoyer un observable auquel il faut souscrire si on veut avoir un affichage
const ob$: Observable<number> = of(1, 2, 3, 4, 5, 6, 7, 8, 9, 10).pipe(filter( v => v % 2 === 0), map( v => v * 10));
ob$.subscribe(
    value => console.log(value)
);
```

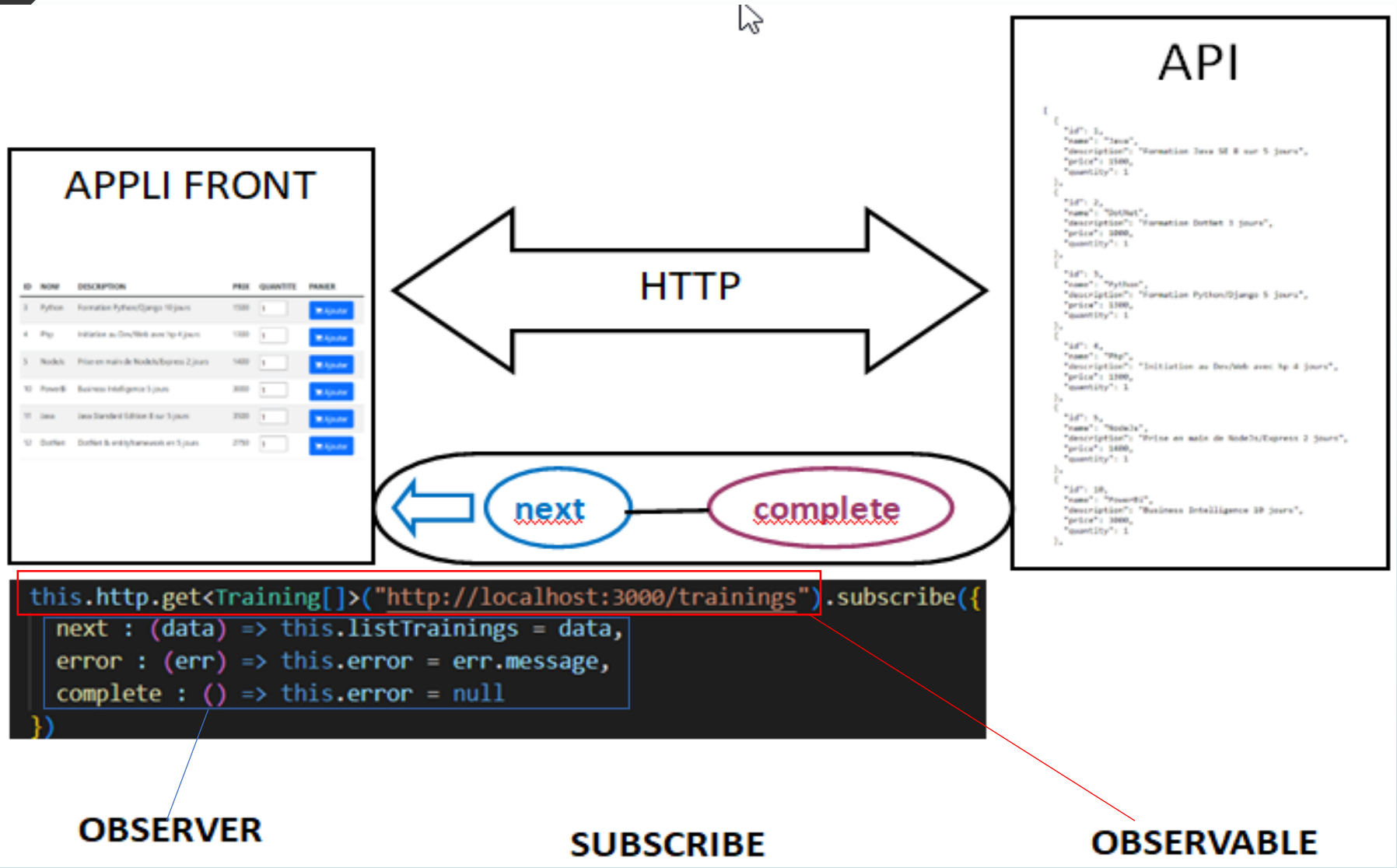


https://github.com/elbabili/fms-ngrx-aircrafts

Pattern Observer

Pourquoi les patterns ?

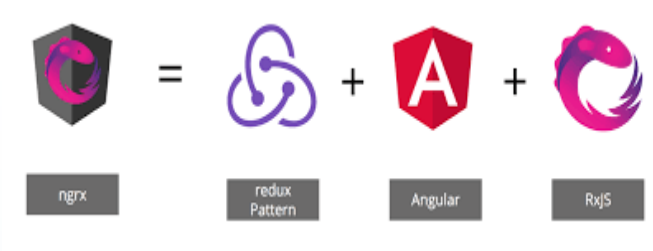
- Des « recettes d'expert » ayant fait leurs preuves
- Un vocabulaire commun pour les architectes logiciels
- Incontournable dans le monde de la P.O.O



L'Observateur souscrit à l'observable, donc chaque fois qu'une donnée arrive, il est notifié et peut agir, on lui donne la main.

Renvoi un observable, on dit aussi qu'il est le Sujet ou Subject qui notifie l'observateur ici

NgRx



- Il s'agit d'une implémentation de Redux (React) pour Angular, elle exploite la programmation reactive avec RxJS.
- Particulièrement intéressant pour les grandes applications, il permet de **centraliser l'état de l'application** dans un objet unique contenant des données partagées par l'ensemble des composants.
- En effet, la complexité des applications web usant du modèle mvc notamment a montré ses limites. NgRx est donc un choix technique très intéressant de lors qu'on a bien assimilé son modèle fruit de l'expérience des développeurs web aguerris.

Install NodeJs & Angular

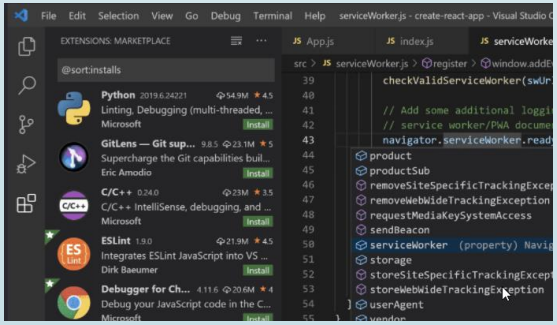
- Install Node :
  - Éviter d'installer les outils complémentaires
  - Pour vérifier que c'est fait
  - Les chemins d'exécution sont ajoutés pendant l'install aussi vous n'avez pas à le faire, si ce n'est pas le cas, il faut vérifier le path.

```
C:\Users\El-BabiliM>node --version
v16.13.1

C:\Users\El-BabiliM>npm --version
8.1.2
```

- Install Angular :
  - npm install -g @angular/cli

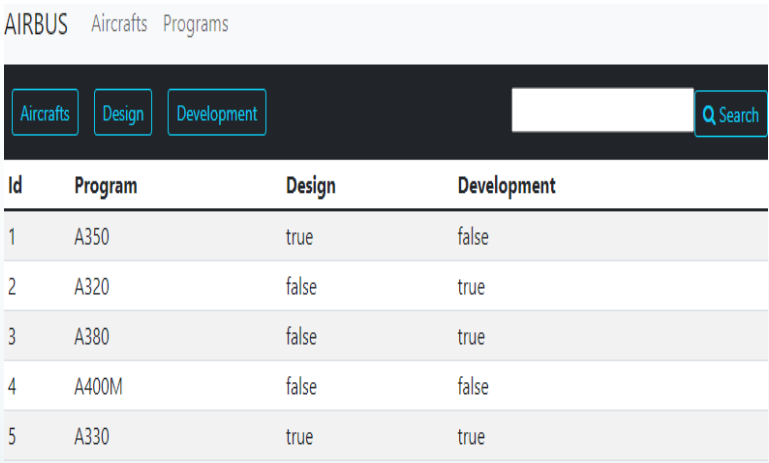
```
Angular CLI: 13.0.4
Node: 16.13.1
Package Manager: npm 8.2.0
OS: win32 x64

Angular: 13.0.3
... animations, common, compiler, compiler-cli, core, forms
... platform-browser, platform-browser-dynamic, router

Package                                  Version
------------------------------------------------------------
@angular-devkit/architect                0.1300.4
@angular-devkit/build-angular            13.0.4
@angular-devkit/core                     13.0.4
@angular-devkit/schematics               13.0.4
@angular/cli                             13.0.4
@schematics/angular                      13.0.4
rxjs                                     7.4.0
typescript                               4.4.4
```

- Choisir un Ide : VSC



1/ projet Angular

Soit une application de gestion du cycle de vie d'un avion



- 1.1 Création d'un projet et ouverture dans un IDE
  - ng new airbus-app-ngrx
- 1.2 Ajout des dépendances front et back
  - front : npm install bootstrap
  - Back : npm install json-server concurrently
- 1.3 Modif fichier "Angular.json"

```
"styles": [
  "src/styles.css",
  "node_modules/bootstrap/dist/css/bootstrap.min.css"
],
```

- 1.4 Modif fichier "Package.json"
  - `"start": "concurrently \"ng serve\"  \"json-server --watch db.json\"",`

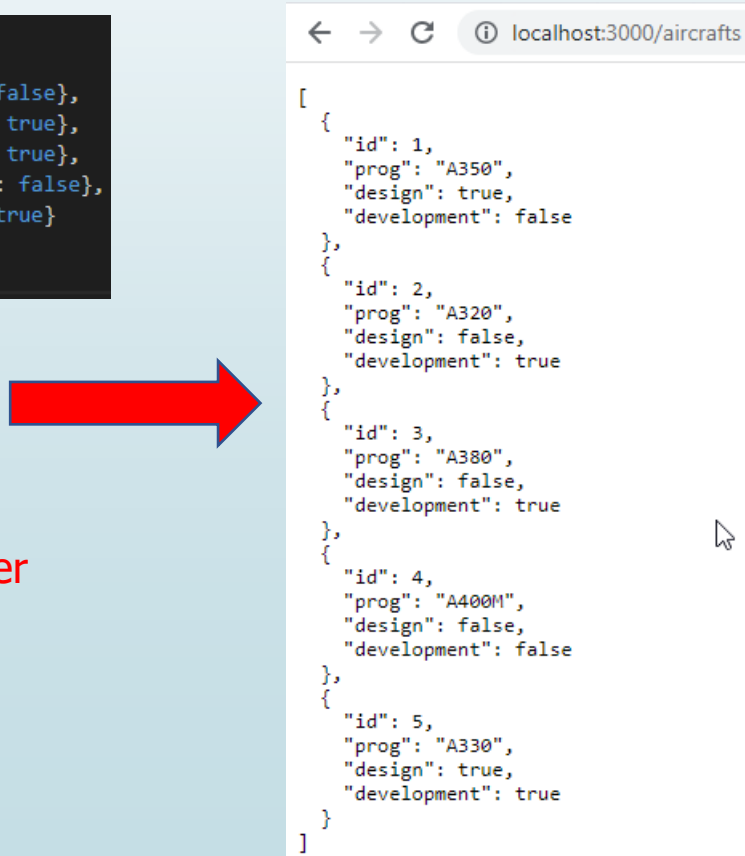
2/ Ajout d'une base de donnée

- Comment ne pas rester bloqué lorsqu'on développe une application front qu'on souhaite tester alors qu'elle nécessite des accès vers le back qui n'est toujours pas opérationnel ? Réponse : **Json Server** !

- 2.1 Ajout d'une base de donnée "db.json" à la racine du projet

```
{
  "aircrafts": [
    {"id": 1,"prog": "A350","design": true,"development": false},
    {"id": 2,"prog": "A320","design": false,"development": true},
    {"id": 3,"prog": "A380","design": false,"development": true},
    {"id": 4,"prog": "A400M","design": false,"development": false},
    {"id": 5,"prog": "A330","design": true,"development": true}
  ]
}
```

- 2.2 Tester en lançant l'appli : npm start

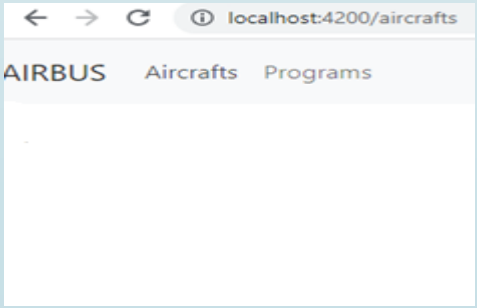


Une et seule fois que ça marche, on souhaite afficher
Cette liste dans notre application !

3/ Page d'accueil

- Ajout d'une page template de navigation avec menu déroulant dans la partie html de votre composant principal « app.component.html »

```
<nav class="navbar navbar-expand-sm bg-light navbar-light">

  <a class="navbar-brand" href="#">AIRBUS</a>

  <!-- Links -->
  <ul class="navbar-nav">
    <li class="nav-item">
      <a class="nav-link" routerLink="/aircrafts">Aircrafts</a>
    </li>
    <li class="nav-item">
      <a class="nav-link" href="#">Programs</a>
    </li>
  </ul>
</nav>

<router-outlet> </router-outlet>
```



4/ Modèle

- Ajout d'un modèle pour représenter nos avions ici et les manipuler dans notre application

src > app > model > TS aircraft.model.ts > Aircraft

```
1
2  export interface Aircraft {
3    msn:string;
4    prog:string;
5    design:boolean;
6    developpment:boolean;
7  }
```

5/ Service

Ajout d'un service permettant de communiquer avec le back end

- 5.1 ajouter HttpClientModule dans app.module.ts
- 5.2 ajouter un service : ng g s services/aircraft
- 5.3 injecter la dépendance dans le constructeur

```
src > app > TS app.module.ts > AppModule
1  import { HttpClientModule } from '@angular/common/http';
2  import { NgModule } from '@angular/core';
3  import { BrowserModule } from '@angular/platform-browser';
4
5  import { AppRoutingModule } from './app-routing.module';
6  import { AppComponent } from './app.component';
7
8  @NgModule({
9    declarations: [
10     AppComponent
11   ],
12   imports: [
13     BrowserModule,
14     AppRoutingModule,
15     HttpClientModule
16   ],
17   providers: [],
18   bootstrap: [AppComponent]
19 })
20 export class AppModule { }
21
```

```
import { HttpClient } from '@angular/common/http';
import { Injectable } from '@angular/core';
import { Observable } from 'rxjs';
import { environment } from 'src/environments/environment';
import { Aircraft } from '../model/aircraft.model';

@Injectable({providedIn: 'root'})  //Service + accessible dans toute l'appli

export class AircraftService {

  constructor(private http:HttpClient) { }

  //liste de tous les avions en base => une fois sur 2 on souhaite provoquer une erreur
  public getAircrafts():Observable<Aircraft[]> {
    let host = Math.random() > 0.5 ? environment.host : environment.unreachableHost;
    return this.http.get<Aircraft[]>(host+"/aircrafts");
  }

  //liste des avions en phase design
  public getDesignedAircrafts():Observable<Aircraft[]>{
    return this.http.get<Aircraft[]>(environment.host+"/aircrafts?design=true");
  }

  //liste des avions en phase de développement
  public getDeveloppementAircrafts():Observable<Aircraft[]>{
    return this.http.get<Aircraft[]>(environment.host+"/aircrafts?developpment=true");
  }

  //renvoi un avion à partir de l'id
  public getAircraftByMsn(id:number) : Observable<Aircraft> {
    return this.http.get<Aircraft>(environment.host + "/aircrafts/" + id)
```

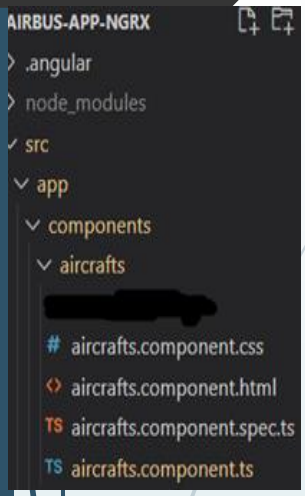
Renvoi un observable

- 5.4 Ajouter les méthodes pour exploiter la bdd

- 5.5 Ajouter les constantes dans le fichier « environnement.ts »

```
export const environment = {
  production: false,
  host : "http://localhost:3000",
  unreachableHost : "http://localhost:3005"
};
```

6/ Ajout d'un composant d'affichage



6.1 ajouter un composant d'affichages des avions dans un répertoire dédié

- ng g c components/aircrafts

6.2 L'enregistrer dans le système de routage afin qu'il puisse être sollicité

app > TS app-routing.module.ts > ...

```ts
import { NgModule } from '@angular/core';
import { RouterModule, Routes } from '@angular/router';
import { AircraftsComponent } from './components/aircrafts/aircrafts.component';

const routes: Routes = [
  {
    path : "aircrafts",
    component : AircraftsComponent
  }
];
```

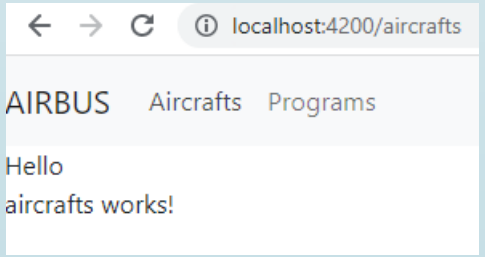
- 6.3 Ajouter un lien dans le composant principal

```html
<!-- Links -->
<ul class="navbar-nav">
  <li class="nav-item">
    <a class="nav-link" routerLink="/aircrafts">Aircrafts</a>
  </li>
  <li class="nav-item">
    <a class="nav-link" href="#">Programs</a>
  </li>
</ul>
```

- 6.4 Indiquer ou le composant aircrafts doit apparaître dans la page html

```html
    <li class="nav-item">
      <a class="nav-link" href="#">Programs</a>
    </li>
  </ul>
</nav>

<router-outlet> Hello </router-outlet>
```

Résultat final



7/ Gestion du composant : **option 1**

- 7.1 Gestion de la partie visuelle de notre composant aircrafts

```
<div>
    <ul class="navbar navbar-nav">
        <li>
            <button class="btn btn-outline-info m-2" (click)="getAllAircrafts()">Aircrafts</button>
            <button class="btn btn-outline-info m-2" (click)="getDesignedAircrafts()">Design</button>
            <button class="btn btn-outline-info m-2" (click)="getDevelopementAircrafts()">Development</button>
        </li>
    </ul>
    <ng-container *ngIf="error">
        <div class="p-2 text-danger"> {{error}}</div>
    </ng-container>
    <ng-container *ngIf="aircrafts">
            <table class="table table-striped">
                <tr>
                    <th>Id</th> <th>Program</th> <th>Design</th> <th>Development</th>
                </tr>
                <tr *ngFor="let aircraft of aircrafts">
                    <td>{{aircraft.id}}</td>
                    <td>{{aircraft.prog}}</td>
                    <td>{{aircraft.design}}</td>
                    <td>{{aircraft.development}}</td>
                </tr>
            </table>
    </ng-container>
</div>
```

Aircrafts | Design | Development

Id	Program	Design	Development
1	A350	true	false
2	A320	false	true
3	A380	false	true
4	A400M	false	false
5	A330	true	true

- 7.2 Gestion de la partie logique

```
export class AircraftsComponent implements OnInit {
  aircrafts : Aircraft[] | null = null; //soit un tableau d'avions soit null d'ou l'affectation
  error = null;

  constructor(private aircraftService:AircraftService) { }

  ngOnInit(): void {
  }

  getAllAircrafts() {
    //Option 1 : nous observons ici ce qui se passe lorsqu'on déclenche l'évènement : récupérer la liste d'avions en base
    this.aircraftService.getAircrafts().subscribe({
      next : (data) => this.aircrafts = data,
      error : (err) => this.error = err.message,
      complete : () => this.error = null
    })
    /*this.aircraftService.getAircrafts().subscribe(  //ecriture dépréciée
      data => { this.aircrafts = data
    }, err => {
      console.log(err)
    })*/
  }
```

7/ Gestion du composant : **option 2**

- 7.3 Gestion de la partie logique

```
export class AircraftsComponent implements OnInit {
  //aircrafts:Aircraft[] | null = null;  //option 1 : soit un tableau d'avions, soit null d'ou l'affectation
  aircrafts$:Observable<Aircraft[]> | null = null; //option 2 : aircrafts est de type observable contenant des avions
  //le cigle $ est une convention d'écriture pour indiquer qu'il s'agit d'un observable
  error = null;
  constructor(private aircraftService:AircraftService) { }

  ngOnInit(): void {
  }
  getAllAircrafts() {
    //Option 2 : la methode du service renvoi un Observable
    this.aircrafts$ = this.aircraftService.getAircrafts();  //delors il faut bien faire un subscribe puisqu'il n'est plus sollicité ici
    //en effet, l'appel sera fait côté html en précisant (pipe)"| async" toujours pour agir lorsque des données arrivent
```

- 7.4 Gestion de la partie html

Souscription à un Observable directement dans la vue

```
<table class="table">
    <tr>
        <th>Id</th> <th>Programme</th> <th>Etude</th> <th>Conception</th>
    </tr>
    <tr *ngFor="let aircraft of aircrafts$ | async">
        <td>{{aircraft.id}}</td>
        <td>{{aircraft.prog}}</td>
        <td>{{aircraft.design}}</td>
        <td>{{aircraft.development}}</td>
    </tr>
</table>
```

Le souci ici c'est qu'on a aucun moyen de récupérer les erreurs !

Tous les Avions | Avions en étude | Avions en construction

Id	Programme	Etude	Conception
1	A350	true	false
2	A320	false	true
3	A380	false	true
4	A400M	false	false
5	A330	true	true

7/ Gestion du composant : **option 3**

- 7.5 On souhaite gérer les erreurs et afficher des messages sur l'état du chargement

```
aircrafts$:Observable<AppDataState<Aircraft[]>> | null = null;
//option 3 : aircrafts est de type observable de structure de donnée AppDataState constituée de 3 éléments facultatifs
//le type générique ici sera dans notre cas une liste d'avions
//cette étape est indispensable afin de permettre à pipe de renvoyer le même type de donnée pour les 3 cas d'utilisations s,m et c
readonly dataStateEnum = DataStateEnum ;

constructor(private aircraftService:AircraftService) { }

ngOnInit(): void {
}
getAllAircrafts() {
  //Option 3 : méthode Pipe avec un ensemble d'opérateur + gestion des états du chargement des données
  //du coup, on peut appliquer un ensemble d'opérateur
  this.aircrafts$ = this.aircraftService.getAircrafts().pipe(
    map(data => ({dataState : DataStateEnum.LOADED, data : data})),
    //opérateur map reçoit une liste d'avions et retourne une fonction avec pour paramètre un objet contenant cette liste
    //il renvoi aussi une variable state qui précise l'état du chargement ici en cours
    startWith({dataState : DataStateEnum.LOADING}),   //dès que pipe est appelé, le premier état est spécifié ici
    catchError(err => of({dataState : DataStateEnum.ERROR, errorMessage : err.message}))
    //là aussi on retourne une fonction qui renvoie un Observable ici grâce à la méthode of
  );
```

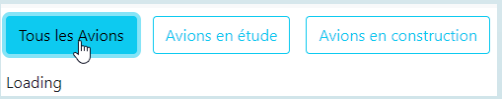
Pipe permet ici d'appliquer une succession de filtre à notre observable, chaque filtre renvoyant un observable

app > state > TS aircraft.state.ts > DataStateEnum

```
export enum DataStateEnum {
    LOADING,
    LOADED,
    ERROR
}

export interface AppDataState<T> {
    dataState? : DataStateEnum,
    data? : T,
    errorMessage?:string
}
```

```
<ng-container *ngIf="aircrafts$ | async as result " [ngSwitch]="result.dataState">

    <ng-container *ngSwitchCase="dataStateEnum.LOADING">
        <div class="p-2"> Loading</div>
    </ng-container>

    <ng-container *ngSwitchCase="dataStateEnum.LOADED">
        <div class="p-2"> Loaded</div>
    </ng-container>

    <ng-container *ngSwitchCase="dataStateEnum.ERROR">
        <div class="p-2 text-danger"> {{result.errorMessage}}</div>
    </ng-container>

    <table class="table">
        <tr>
            <th>Id</th> <th>Programme</th> <th>Etude</th> <th>Conception</th>
        </tr>
        <tr *ngFor="let aircraft of result.data">
            <td>{{aircraft.id}}</td>
            <td>{{aircraft.prog}}</td>
```

Step 1

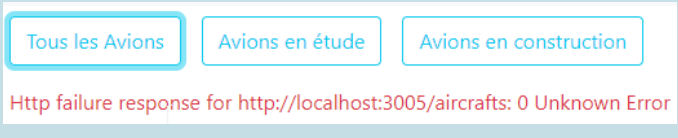


Step 2

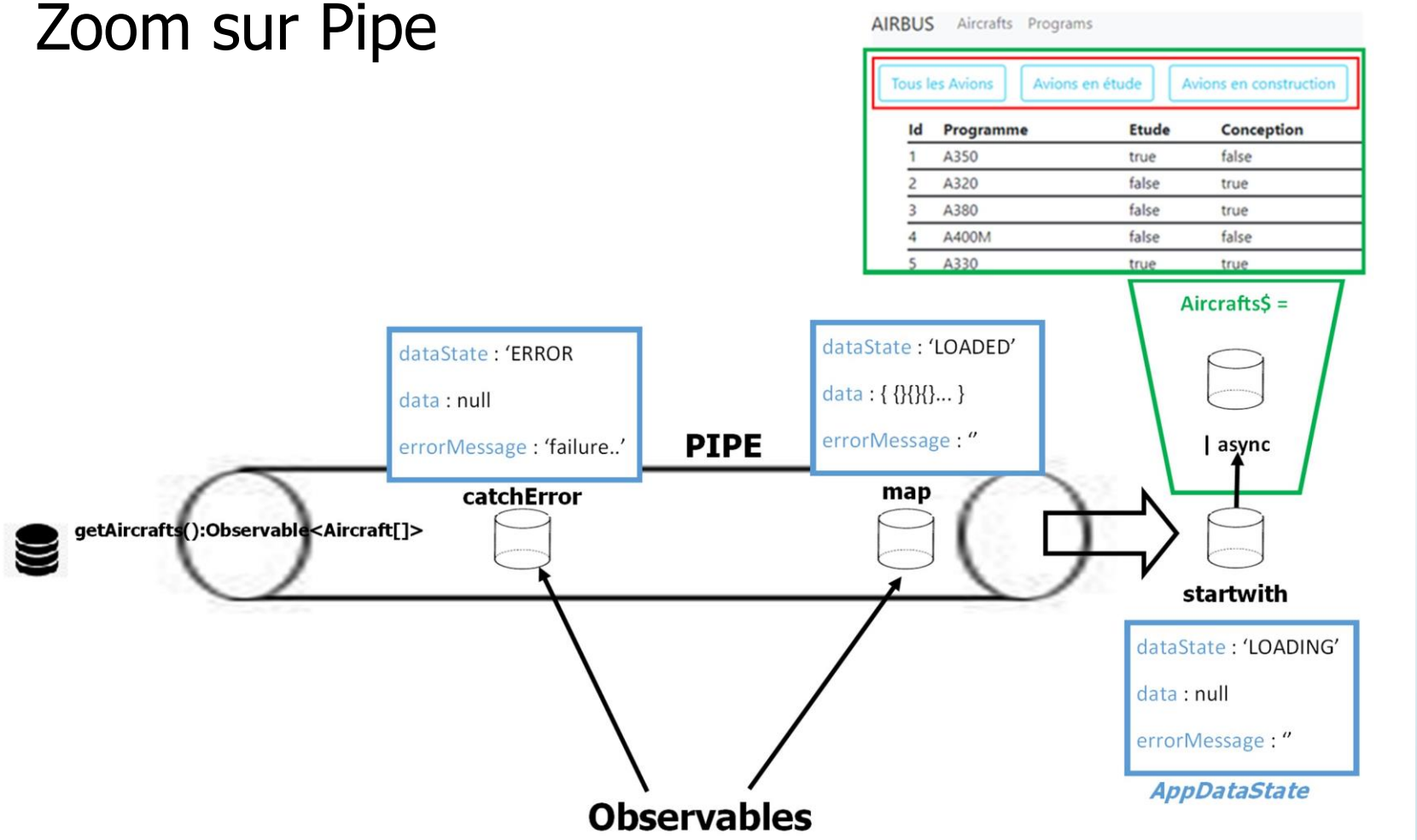
Loaded

Id	Programme	Etude	Conception
1	A350	true	false
2	A320	false	true
3	A380	false	true
4	A400M	false	false
5	A330	true	true

Error



Zoom sur Pipe



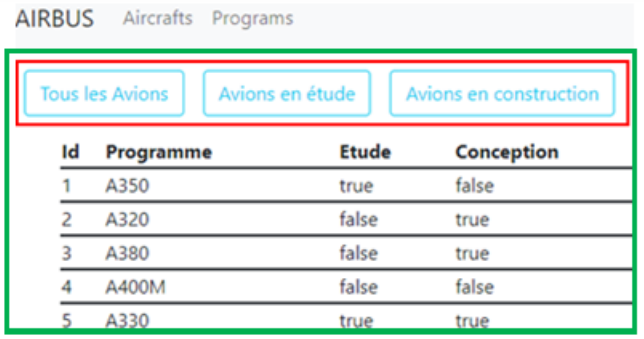
8/ Ajout d'un composant de navigation

L'idée est de découper/décomposer l'application :

- 8.1 Dans le composant d'affichage, ajouter un composant de navigation : **ng g c components\aircrafts\aircrafts-navbar**
- 8.2 dans le composant de navigation, (après les avoir retirer du composant aircrafts) ajouter les boutons de navigation dans la partie vue(html) et les méthodes correspondantes dans la partie logique(ts) du composant aircrafts-navbar

  Puis injecter

- 8.3 Sauf qu'il faut dorénavant inter agir avec les actions qui viennent du composant enfant aircrafts-navbar afin de solliciter les méthodes du composant parent aircrafts




```
app > components > aircrafts > aircrafts-navbar > aircrafts-navbar.component.html > ...
<nav class = "navbar navbar-expand-sm">
    <ul class="nav navbar-nav">
        <li>
            <button class="btn btn-outline-info m-2" (click)="getAllAircrafts()">Tous les Avions</button>
            <button class="btn btn-outline-info m-2" (click)="getDesignAircrafts()">Avions en étude</button>
            <button class="btn btn-outline-info m-2" (click)="getDeveloppementAircrafts()">Avions en construction</button>
        </li>
    </ul>
</nav>
```

```
app > components > aircrafts > aircrafts.component.html > app-aircrafts-navbar
<app-aircrafts-navbar></app-aircrafts-navbar>

<div class="container" *ngIf="aircrafts$ | async as result " [ngSwitch]="result.dataState">
```

Step 1

```
export class AircraftsNavbarComponent implements OnInit {
  @Output() eventEmitter : EventEmitter<any> = new EventEmitter();
  //Il s'agit de provoquer un évènement de sortie sur notre composant

  constructor() { }

  ngOnInit(): void {
  }

  //lorsque l'utilisateur clic sur un bouton l'action correspondante est émise
  getAllAircrafts() {
    this.eventEmitter.emit("ALL_AIRCRAFTS");
  }
```

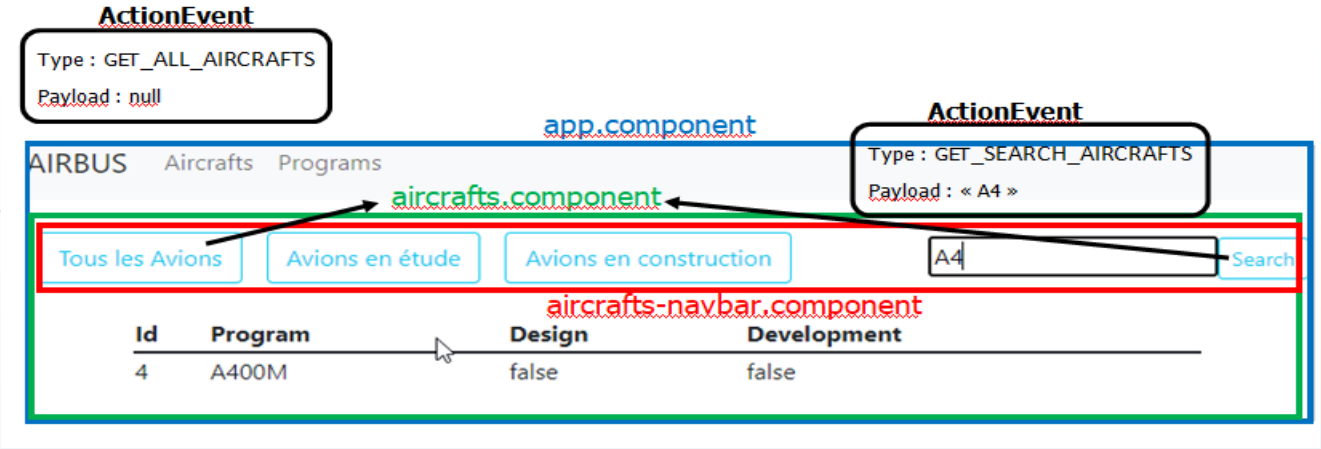
Step 2 Le composant parent écoute ici les évènements de son enfant

```
app > components > aircrafts > aircrafts.component.html > app-aircrafts-navbar
<app-aircrafts-navbar (eventEmitter)="onActionEvent($event)"></app-aircrafts-navbar>
```

Step 3

```
//En résumé, le composant parent écoute les évènements de l'enfant
//et lorsqu'il se produit qqchose la méthode ci dessous est appelé
onActionEvent($event : any){
  if($event == "ALL_AIRCRAFTS") this.getAllAircrafts();
}
```

9/ Gestion du composant de navigation



- On souhaite gérer plusieurs cas d'utilisation dans notre barre de navigation et passer des arguments aux évènements comme dans le cas d'une recherche par mots clés, il faut :
  - 9.1 définir une interface représentant nos évènements :

```
//nous définissons ici un objet évènement caractérisé par le type d'évt et son contenu indéfini
export interface ActionEvent {
    type : AircraftsActionsTypes,
    payload : any
}
```

```
export enum AircraftsActionsTypes {
    GET_ALL_AIRCRAFTS = "[Aircrafts] Get All Aircrafts",
    GET_DESIGNED_AIRCRAFTS = "[Aircrafts] Get Designed Aircrafts",
    GET_DEVELOPMENT_AIRCRAFTS = "[Aircrafts] Get Developed Aircrafts",
    GET_SEARCH_AIRCRAFTS = "[Aircrafts] Get Search Aircrafts"
}
```

  - 9.2 dans notre composant **aircrafts-navbar** on peut émettre dorénavant plusieurs évènements

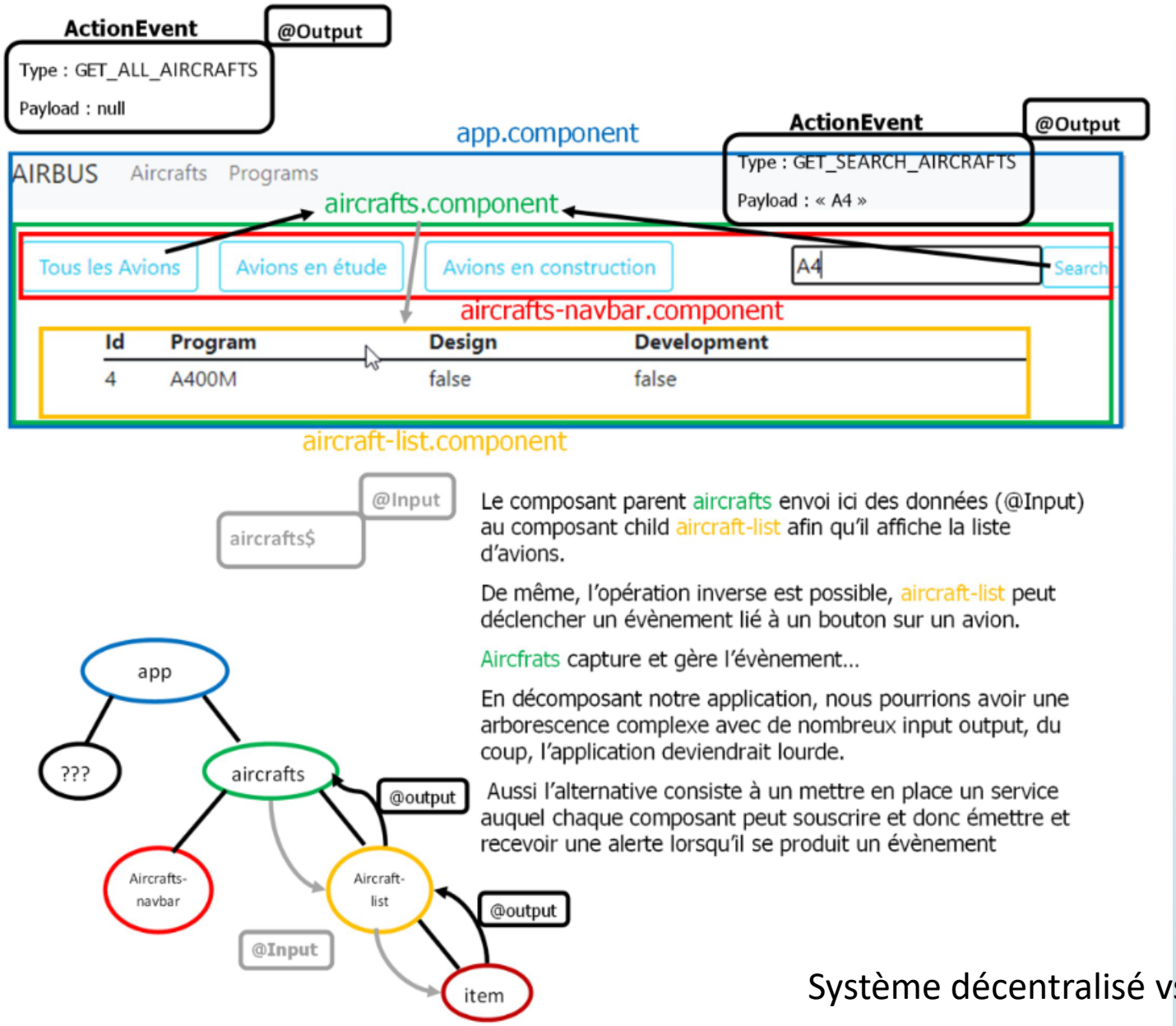
```
getAllAircrafts() {
  this.eventEmitter.emit({type : AircraftsActionsTypes.GET_ALL_AIRCRAFTS , payload : null});
  //dorénavant on émet notre objet évt ici avec un payload null puisqu'il n'y a pas d'arguments...
}

onSearch(value : any) {
  this.eventEmitter.emit({type : AircraftsActionsTypes.GET_SEARCH_AIRCRAFTS , payload : value});
  //émission de l'évènement avec le mot clé de recherche dans le payload
}
```

  - 9.3 gestion dans aircrafts

```
onActionEvent($actionEvent : ActionEvent){
  switch($actionEvent.type){  //qq soit l'évt, on le gère ici
      case AircraftsActionsTypes.GET_ALL_AIRCRAFTS :
      this.getAllAircrafts();
      break;

      case AircraftsActionsTypes.GET_SEARCH_AIRCRAFTS :
      this.search($actionEvent.payload);
      break;
  }
}
```

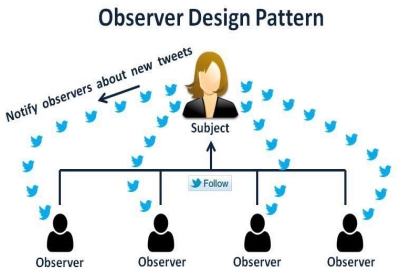
NB : procédure d'ajout d'un formulaire
1 : réaliser le formulaire ici dans la barre de navigation
2 : ajouter FormsModule dans app.module
3 : ajouter la méthode côté typescript
4 : ajouter la méthode correspondante dans le service
5 : gestion de l'évènement + test

10/ Synthèse & problématique



Système décentralisé vs centralisé

11/ Service centralisé de gestion des évènements



- 11.1 Ajouter un service dédié

app > state > TS event.service.ts > EventService

```
import { Injectable } from "@angular/core";
import { Subject } from "rxjs";
import { ActionEvent } from "./aircraft.state";

@Injectable({providedIn:"root"})    //spécifier que c'est un service accessible dans root (pas necessaire de le déclar
export class EventService {
    //Objet permettant la communication entre les composants
    eventSubject : Subject<ActionEvent> = new Subject<ActionEvent>();
    //type d'évènements publiés par notre objet
    eventSubjectObservable = this.eventSubject.asObservable();
    //Observable qui permet à tous les composants qui le souhaitent de recevoir les actions des autres

    publishEvent(event : ActionEvent){  //méthode de publication d'un évènement ou message
        this.eventSubject.next(event);  //tous les composants qui ont soucrit à notre service, recevront l'évt publié
    }                                   //ils devront faire auparavant un subscribe à eventSubjectObservable
}
```

Le sujet : La personne qui a un compte twitter (**Observable**)

- 11.2 Souscription d'un composant au service [**SUBSCRIBE**] + traitement ou pas dans le switch de la méthode appelée

```
constructor(private aircraftService:AircraftService, private eventService:EventService) { }

ngOnInit(): void {  //au boot du composant, il souscrit ici au service et dès qu'il reçoit un évènement de type ActionEvent
  this.eventService.eventSubjectObservable.subscribe((actionEvent : ActionEvent) => {
    this.onActionEvent(actionEvent);    //dès qu'une action arrive, le correspond peut réagir ou pas
  })
}
```

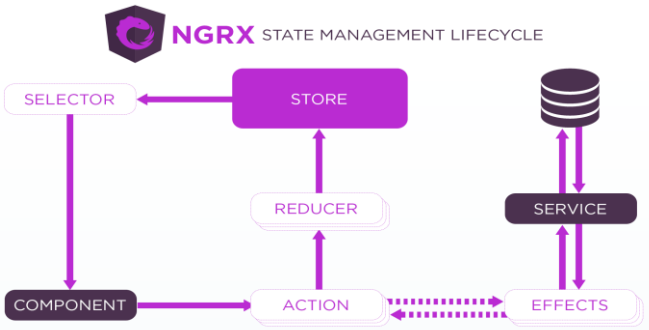
Un **observer** (fan) souscrit à suivre la personne, il reçoit les tweets et peut réagir

- 11.3 Publication d'un évènement [**PUBLISH**]

```
constructor(private eventService:EventService) { }
ngOnInit(): void {  }

//lorsque l'utilisateur clic sur un bouton l'action correspondante est PUBLIEE !
getAllAircrafts() {
  this.eventService.publishEvent({type : AircraftsActionsTypes.GET_ALL_AIRCRAFTS , payload : null});
  //this.eventEmitter.emit({type : AircraftsActionsTypes.GET_ALL_AIRCRAFTS , payload : null});
  //dorénavant on émet notre objet évt ici avec un payload null puisqu'il n'y a pas d'arguments...
}
```

Un fan publie un tweet, tous les observers le reçoivent

Exploitation de NgRx



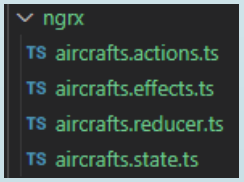
Comme dans les versions précédentes, nous souhaitons afficher la liste de tous les avions ainsi qu'un certain nombre de fonctionnalités, cette fois nous allons exploiter NgRx !

- Pour ce faire, il faut installer les dépendances :
  - npm install @ngrx/store
  - npm install @ngrx/effects
  - npm install @ngrx/store-devtools

- Ajouter les 3 modules dans app.module

```
StoreModule.forRoot({}),                    //spécifier le reducer
EffectsModule.forRoot([]),                  //spécifier les effects
StoreDevtoolsModule.instrument()  //en l'activant ici, à chaque action de NgRx dans l'appli
                                  //le plugin redux (chrome) permet l'analyse du state durant le dev
```

- Ajouter un dossier ngrx et nos fichiers :
  - State, Actions, Effects, Reducer



1/ Les Actions ou évènements

Ce sont tous les évènements déclenchés par l'utilisateur tel que le clic sur un bouton par exemple. Elles sont caractérisés par leur **type**. Les actions provoquées par l'User sont interceptées par le STORE puis relayées vers le REDUCER puis le(s) EFFECTS; ce(s) dernier(s) peut lui même lancer une action, en réponse à un accès en bdd, avec un type donné, par ex « GET_ALL_AIRCRAFTS_SUCCESS », delors l'action contient des données dans son **payload** !

ACTION

- **TYPE** : nature de l'action
- **PAYLOAD** : data s'il y en a

```
import { Action } from "@ngrx/store";
import { Aircraft } from "../model/aircraft.model";

export enum AircraftsActionsTypes{
    //Action : Get all aircrafts
    //s'agissant de l'action consistant à afficher tous les avions, nous avons 3 états possible
    GET_ALL_AIRCRAFTS = "[Aircrafts] Get All Aircrafts",                //demande tous les avions
    GET_ALL_AIRCRAFTS_SUCCESS = "[Aircrafts] Get All Aircrafts Success",//demande ok
    GET_ALL_AIRCRAFTS_ERROR = "[Aircrafts] Get All Aircrafts Error",    //demande nok
}

//Get all aircrafts
export class GetAllAircraftsAction implements Action{
    type: AircraftsActionsTypes = AircraftsActionsTypes.GET_ALL_AIRCRAFTS;
    constructor(public payload:any) {
    }
}
export class GetAllAircraftsActionSuccess implements Action{
    type: AircraftsActionsTypes = AircraftsActionsTypes.GET_ALL_AIRCRAFTS_SUCCESS;
    constructor(public payload:Aircraft[]) {
    }
}
export class GetAllAircraftsActionError implements Action{
    type: AircraftsActionsTypes = AircraftsActionsTypes.GET_ALL_AIRCRAFTS_ERROR;
    constructor(public payload:string) {   //message d'erreur
    }
}

export type AircraftsActions = GetAllAircraftsAction | GetAllAircraftsActionSuccess | GetAllAircraftsActionError;
```

2/ State

NgRx est un pattern de gestion d'état (**State**) qu'il stocke dans un **Store** et repose sur 3 principes :

- *Single source of truth* : Le State est stocké dans un Store unique.
- *State readonly* : Immutabilité du State. Une fois créé, la seule façon de le modifier est de le cloner.
- *Changes are made with pure functions* : Permet d'éviter les effets de bords (modif du state impossible)

app > ngrx > TS aircrafts.state.ts > initState

```ts
import { Aircraft } from "../model/aircraft.model";

export enum AircraftsStateEnum{  // les différents états du state
    LOADING = "Loading",     //chargement en cours
    LOADED = "Loaded",       //chargé
    ERROR = "Error",         //erreur
    INITIAL = "Initial"      //état initial
}
export interface AircraftsState {    //structure de notre STATE
    aircrafts : Aircraft[],           //liste d'avions qui s'affichent
    errorMessage:string,             //un message d'erreur
    dataState : AircraftsStateEnum   //état du state
}
//il est nécessaire de définir l'état initial du state avec des valeurs par défaut
export const initState : AircraftsState = {
    aircrafts : [],
    errorMessage:"",
    dataState : AircraftsStateEnum.INITIAL
}
```

3/ Reducer

Le Reducer est une fonction pure, elle se comporte toujours de la même manière, elle reçoit dans notre cas en paramètre le state et l'action. **Il a vocation à changer l'état(state) de l'application en fonction des actions.** Au boot de l'appli, le store fait appel au reducer en lui transmettant le state, celui-ci n'étant pas encore crée, « initState » sera utilisé avec les paramètres par défaut. Le reducer va ensuite réaliser des tests pour répondre aux actions émises via un switch comme ci-dessous, il renvoi au store un clone du state avec les changements associés.

En bref, le reducer reçoit le state actuel + action dispatchée dans le store et retourne le new state

```
export function AircraftsReducer(state : AircraftsState = initState, action:Action) {
    switch(action.type){    //pour chaque action, on retourne un clone du state (immutable)
        case AircraftsActionsTypes.GET_ALL_AIRCRAFTS:
            console.log("loading...");
            return {...state, dataState:AircraftsStateEnum.LOADING }   //renvoi clone du state + le nouveau dataState

        case AircraftsActionsTypes.GET_ALL_AIRCRAFTS_SUCCESS:
        // Action a été reçu par l'effect qui a fait une demande en base, reçoit les datas et génère l'action pour indiquer que tout est ok
            return {...state, dataState:AircraftsStateEnum.LOADED, aircrafts:(<AircraftsActions> action).payload}
            // renvoi clone + nouveau dataState + liste des avions en base contenu dans le payload

        case AircraftsActionsTypes.GET_ALL_AIRCRAFTS_ERROR :
            return {...state, dataState:AircraftsStateEnum.ERROR, errorMessage:(<AircraftsActions> action).payload}

        default :
            return {...state}
    }
}   //en bref : le reducer reçoit state actuel + action dispatchée dans le store et retourne le new state
```

4/ Effects

L'effect est sollicité uniquement pour une liste d'actions données. Si le type d'action reçue est gérée, il pourra effectuer des accès en base via une api par ex et renverra le résultat sous forme d'une nouvelle action avec des données ou des erreurs. Ici encore, la programmation réactive à l'aide d'RxJs permet de ne pas bloquer l'application, les datas arrivent de manière asynchrone.

```
@Injectable ()  //décorateur spéficie qu'il s'agit d'un service
export class AircraftsEffects {
    constructor(private aircraftService: AircraftService, private effectActions : Actions){
    }

    getAllAircraftsEffect:Observable<Action> = createEffect(     //nous souhaitons créer un effect ici sous condition, donc on écoute les actions
        () => this.effectActions.pipe(                           //dès qu'une action arrive, on vérifie son type
                ofType(AircraftsActionsTypes.GET_ALL_AIRCRAFTS),  //si l'action est de type GET_ALL_AIRCRAFTS alors étape suivante : mergeMap
                mergeMap((action) => {
                    return this.aircraftService.getAircrafts().pipe(  //attente réception des données en base (liste des avions)
                        map((aircrafts) => new GetAllAircraftsActionSuccess(aircrafts)), //si reçu, alors on retourne un Observable<Action>
                                                                                         //dont le payload est la liste des avions
                                                                                        //l'action une fois émise va être traité par le Reducer
                                                                                       //case AircraftsActionsTypes.GET_ALL_AIRCRAFTS_SUCCESS:
                        catchError((err)=>of(new GetAllAircraftsActionError(err.message)))     //s'il y a erreur, génération d'une autre action
                    )
                })
        )
    );
}
```

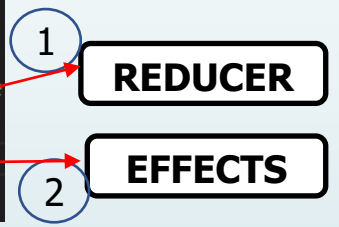
5/ Component

- 1/ Spécifier d'abord dans le module Reducer et Effects

```
StoreModule.forRoot({airbusState : AircraftsReducer}),    //spécifier le reducer
EffectsModule.forRoot([AircraftsEffects]),                //spécifier les effects
```

- 2/ L'utilisateur clic sur un bouton qui affiche la liste des avions

```
export class AircraftsNavbarComponent implements OnInit {
  constructor(private store:Store<any>) { //injection du store en spécifiant ou pas le type du state
  }
  ngOnInit(): void {
  }
  getAllAircrafts(){
    //User a cliqué sur le bouton afficher tous les avions aussi il faut dispatcher l'action à l'aide du store
    this.store.dispatch(new GetAllAircraftsAction({}));
    //Le reducer et l'effect ont reçu la notification du Store et ils ont pris le relais
  }
```



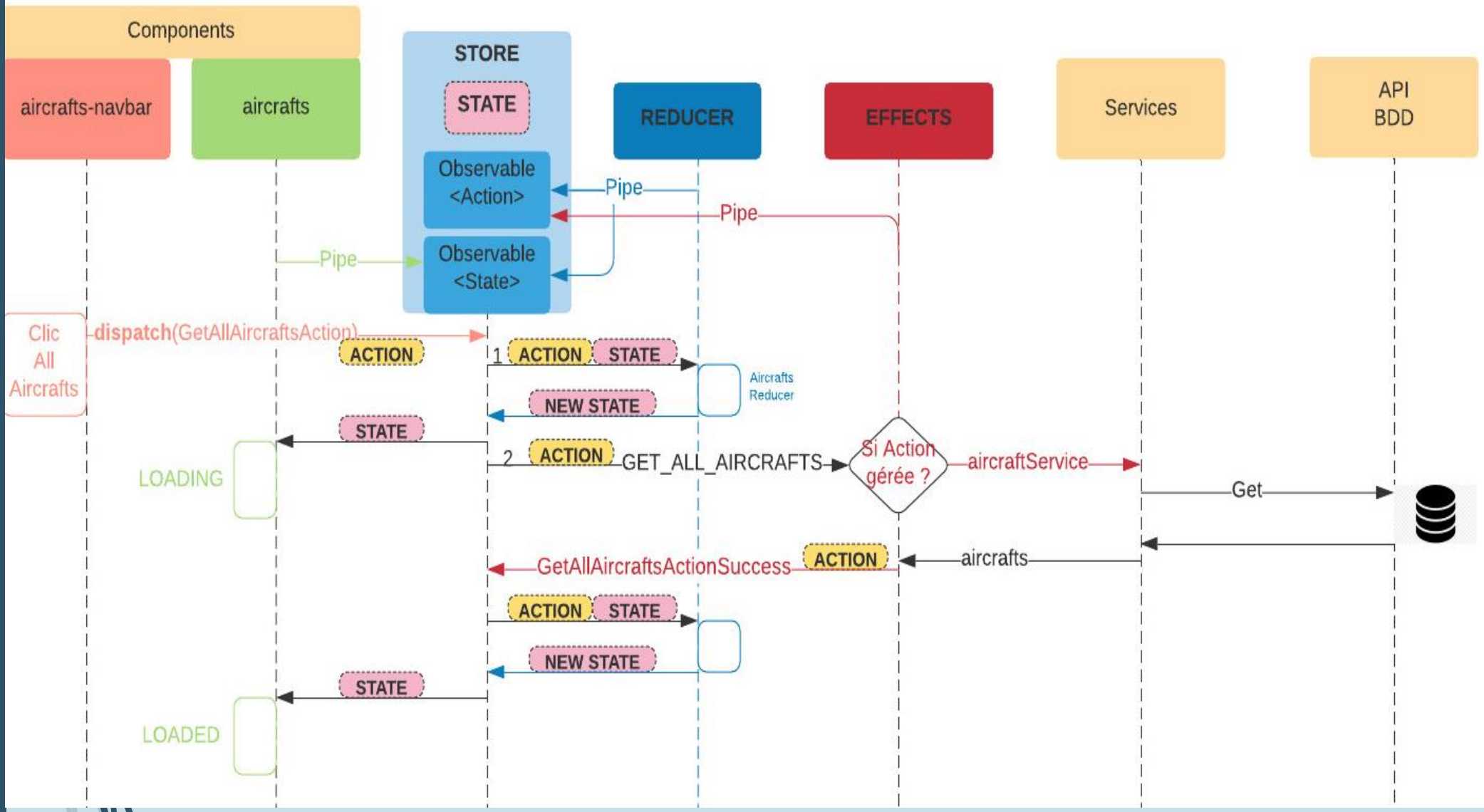
- 3/ Gestion des données dans le composant dédié

```
export class AircraftsComponent implements OnInit {
  aircraftsState$:Observable<AircraftsState> | null = null;

  readonly aircraftsStateEnum = AircraftsStateEnum;

  constructor(private store:Store<any>) {
  }

  ngOnInit(): void {  //avant tout, notre composant doit faire un pipe vers le store
    this.aircraftsState$ = this.store.pipe(//on écoute ce qui se passe dans le store, dès qu'on reçoit les données, on peut faire un map
          //dit autrement : nous recevons le state dès qu'il change afin de permettre l'affichage adéquat de ses données
          map((state)=> state.airbusState)
    );
  }
```

6/ Affichage dans le composant aircrafts

```
<app-aircrafts-navbar></app-aircrafts-navbar>
<ng-container *ngIf="aircraftsState$ | async as state" [ngSwitch]="state.dataState">
        <ng-container *ngSwitchCase="aircraftsStateEnum.INITIAL">
            <div class="p-2"> Initial State</div>
        </ng-container>
        <ng-container *ngSwitchCase="aircraftsStateEnum.LOADING">
            <div class="p-2"> Loading</div>
        </ng-container>
        <ng-container *ngSwitchCase="aircraftsStateEnum.ERROR">
            <div class="p-2 text-danger"> {{state.errorMessage}}</div>
        </ng-container>

        <ng-container *ngSwitchCase="aircraftsStateEnum.LOADED">
            <table class="table">
                <tr>
                    <th>Id</th> <th>Program</th> <th>Design</th> <th>Development</th>
                </tr>
                <tr *ngFor="let aircraft of state.aircrafts">
                    <td>{{aircraft.id}}</td>
                    <td>{{aircraft.prog}}</td>
                    <td>{{aircraft.design}}</td>
                    <td>{{aircraft.development}}</td>
                </tr>
        </table>
        </ng-container>
</ng-container>
```

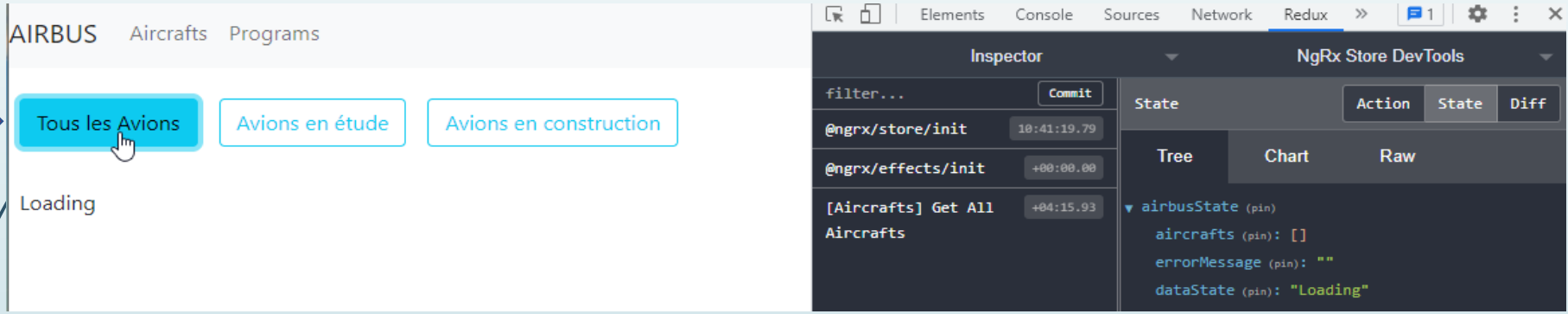
7/ Synthèse



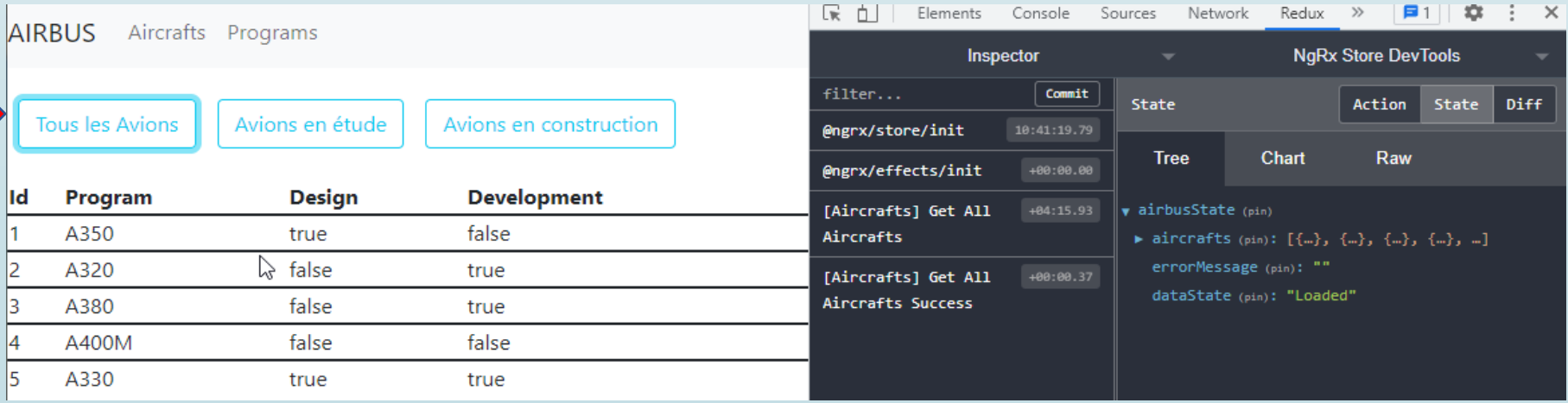
8/ Test de l'appli + Redux DevTools

Start

Loading



Loaded



9/ Rajout d'une fonctionnalité – step1

Rajouter une nouvelle fonctionnalités revient d'abord à rajouter des **actions** ou évènements à la suite des autres actions

```
//Action : Get Designed aircrafts
GET_DESIGNED_AIRCRAFTS = "[Aircrafts] Get Designed Aircrafts",
GET_DESIGNED_AIRCRAFTS_SUCCESS = "[Aircrafts] Get Designed Aircrafts Success",
GET_DESIGNED_AIRCRAFTS_ERROR = "[Aircrafts] Get Designed Aircrafts Error",
```

```
//Get Designed aircrafts
export class GetDesignedAircraftsAction implements Action {
    type: AircraftsActionsTypes = AircraftsActionsTypes.GET_DESIGNED_AIRCRAFTS;
    constructor(public payload:any) {
    }
}
export cla  enum AircraftsActionsTypes  nSuccess implements Action {
    type: AircraftsActionsTypes = AircraftsActionsTypes.GET_DESIGNED_AIRCRAFTS_SUCCESS;
    constructor(public payload:Aircraft[]) {
    }
}
export class GetDesignedAircraftsActionError implements Action {
    type: AircraftsActionsTypes = AircraftsActionsTypes.GET_DESIGNED_AIRCRAFTS_ERROR;
    constructor(public payload:string) {   //message d'erreur
    }
}

export type AircraftsActions = GetAllAircraftsAction | GetAllAircraftsActionSuccess | GetAllAircraftsActionError |
GetDesignedAircraftsAction | GetDesignedAircraftsActionSuccess | GetDesignedAircraftsActionError;
```

9/ Rajout d'une fonctionnalité – step2

Faire de même pour le **Reducer**, ajouter à la suite dans le switch

```
// Get Designed Aircrafts
case AircraftsActionsTypes.GET_DESIGNED_AIRCRAFTS :
    return {...state, dataState:AircraftsStateEnum.LOADING }
case AircraftsActionsTypes.GET_DESIGNED_AIRCRAFTS_SUCCESS :
    return {...state, dataState:AircraftsStateEnum.LOADED, aircrafts:(<AircraftsActions> action).payload}
case AircraftsActionsTypes.GET_DESIGNED_AIRCRAFTS_ERROR :
    return {...state, dataState:AircraftsStateEnum.ERROR, errorMessage:(<AircraftsActions> action).payload}
```

Pour l'**Effect**, il faut ajouter une nouvelle méthode toujours à la suite

```
getDesignedAircraftsEffect:Observable<Action> = createEffect (
    () => this.effectActions.pipe(
        ofType(AircraftsActionsTypes.GET_DESIGNED_AIRCRAFTS),
        mergeMap((action) => {
            return this.aircraftService.getDesignedAircrafts().pipe(
                map((aircrafts) => new GetDesignedAircraftsActionSuccess(aircrafts)),
                catchError((err) => of(new GetDesignedAircraftsActionError(err.message)))
            )
        })
    )
);
```

9/ Rajout d'une fonctionnalité – step3

Ajouter la méthode dans la partie html

```
<button class="btn btn-outline-info m-2 btn-sm" (click)="getAllAircrafts()">Aircrafts</button>
<button class="btn btn-outline-info m-2 btn-sm" (click)="getDesignedAircrafts()">Design</button>
```

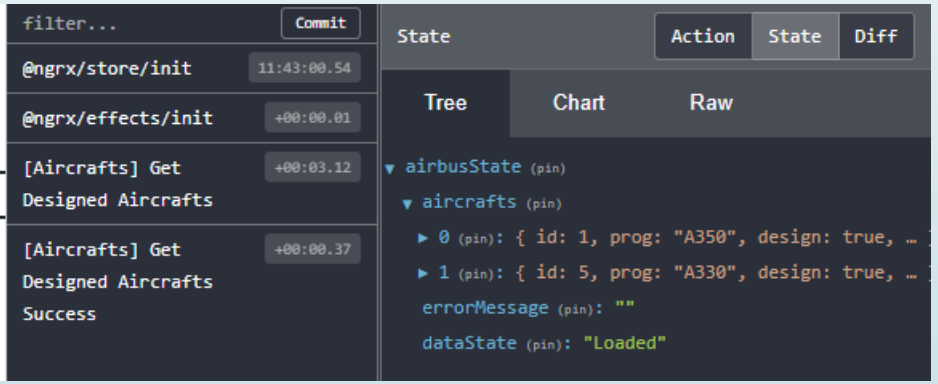
Dispatcher dans le store une nouvelle action

```
getAllAircrafts(){
  //User a cliqué sur le bouton afficher tous les avions aussi il faut dispatcher l'action à l'aide du store
  this.store.dispatch(new GetAllAircraftsAction({}));
  //Le reducer et l'effect ont reçu la notification du Store et ils ont pris le relais
}

getDesignedAircrafts(){
  this.store.dispatch(new GetDesignedAircraftsAction({}));
}
```

Résultat en cliquant sur cette nouvelle action

Tous les Avions | Avions en étude | Avions en construction

Id	Program	Design	Development
1	A350	true	false
5	A330	true	true



Idem pour le 3ème bouton

Tous les Avions | Avions en étude | Avions en construction

Id	Program	Design	Development
2	A320	false	true
3	A380	false	true
5	A330	true	true

10/ Selectors

Ce sont des **fonctions pures utilisées pour obtenir une partie du state**. Les sélecteurs fournissent de nombreuses fonctionnalités sur une sélection donnée. A titre d'exemple, **grâce à la mémorisation, le store garde le résultat d'un appel avec des arguments donnés afin d'éviter de solliciter notre fonction pour obtenir le même affichage par ex, ce qui rend l'appli plus performante**. En bref, les composants peuvent observer une partie du state à l'aide des selectors comme ici pour remonter une alerte, il faut créer le sélecteur puis l'ajouter à un composant.

Step 1

src > app > ngrx > TS aircrafts.selectors.ts > ...

```ts
import { createSelector, createFeatureSelector } from "@ngrx/store";
import { AircraftsState } from "./aircrafts.state";

//nous souhaitons renvoyer le nombre d'avions à la fois en phase d'étude et de construction pour alerter par ex
export const selectCountAlertAircrafts = createSelector(
  createFeatureSelector('airbusState'),  //nous spécifions ici le state sur lequel va agir notre selector
  (state: AircraftsState) => {    //nous récupérons notre state pour extraire des données, ici la liste d'avions
    let total: number = 0;
    state.aircrafts.forEach(a => {
      if (a.development == true && a.design == true) total++;
    })
    return total;
  }
);
```

Step 2

```ts
export class AircraftsComponent implements OnInit {
  aircraftsState$:Observable<AircraftsState> | null = null;
  (property) AircraftsComponent.countAlertAircfrats$: Observable<number> | undefined
  countAlertAircfrats$ : Observable<number> | undefined;

  constructor(private store:Store<any>) {
    this.countAlertAircfrats$ = store.select(selectCountAlertAircrafts);
  }
```

```html
<ng-container *ngIf="countAlertAircfrats$ | async as countAircfrats">
    <h6 class="text-danger"> total aircrafts under design & dev : {{countAircfrats}} </h6>
</ng-container>
```

Résultats :

AIRBUS Aircrafts Programs

Aircrafts | Design | Development | Search

Id	Program	Design	Development
1	A350	true	false
2	A320	false	true
3	A380	true	true
4	A400M	false	false
5	A330	true	true

total aircrafts under design & dev : 2

Un autre selector ici renvoi Le tableau correspondant Dans un autre composant

AIRBUS Aircrafts Alert

Id	Program	Design	Development
3	A380	true	true
5	A330	true	true

11/ Entity

Imaginons maintenant qu'on souhaite permettre l'achat de pièces tout au long du cycle de vie d'un avion aussi appelé gestion de la configuration d'un avion. Nous avons besoin d'une structure de donnée ordonnée. Nous pourrions utiliser un Tableau ou Map, sauf que chacun présente des avantages et des inconvénients.
Le tableau gère l'insertion des éléments de manière ordonnée certes mais pose pb s'agissant des doublons par ex. De même, une Map évite les doublons car souvent on utilise la clé pour l'id d'un élément mais l'insertion dans celle-ci n'est pas ordonné en fonction de l'id.
Voilà pourquoi, nous utilisons ici les 2 concepts combinés :

- `ids: [],`
- `entities: {}`

Step 1 **npm install @ngrx/entity --save**

Step 2 Notre state hérite dorénavant de EntityState

```
export interface AircraftsState extends EntityState<Operation> {    //structure de notre STATE
    aircrafts : Aircraft[],          //liste d'avions qui s'affichent
    errorMessage:string,             //un message d'erreur
    dataState : AircraftsStateEnum,   //état du state
}
//il est nécessaire de définir l'état initial du state avec des valeurs par défaut
export const initState : AircraftsState = {
    aircrafts: [],
    errorMessage: "",
    dataState: AircraftsStateEnum.INITIAL,
    ids: [],
    entities: {}
}
```

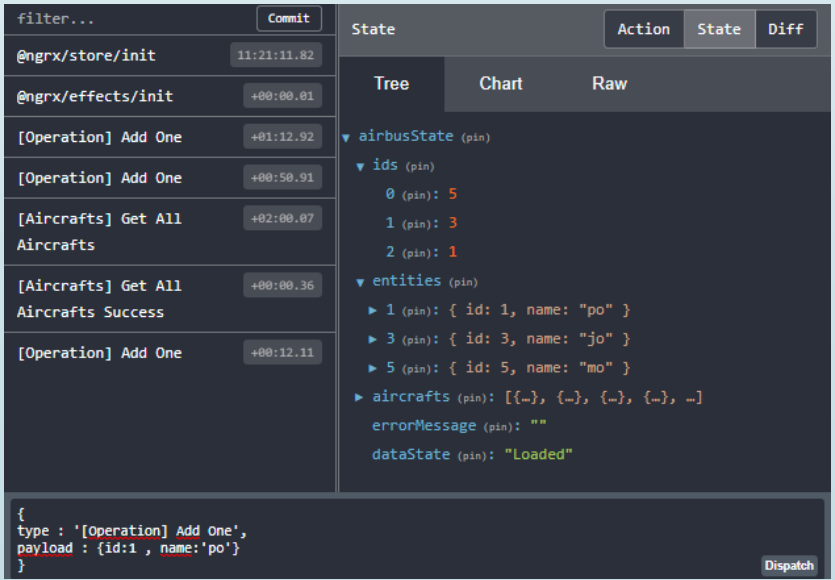
Step 3 Mise en place d'un adaptateur

```
export const adapter: EntityAdapter<Operation> = createEntityAdapter<Operation>({
    //on a besoin d'un adapter afin de manipuler nos entités avec un certain nombre de méthodes
    //sortComparer: sortByPriority  ->  il est possible d'ajouter des méthodes ici par ex pour trier
    //sinon le trie se fait par défaut sur l'id
});

export const initialState: AircraftsState = adapter.getInitialState({
    aircrafts: [],
    errorMessage: "",
    dataState: AircraftsStateEnum.INITIAL,
    ids: [],
    entities: {}
});

export function AircraftsReducer(state : AircraftsState = initialState, action:Action) : AircraftsState {
    switch(action.type){    //pour chaque action, on retourne un clone du state (immutable)
```

```
// Gestion Opérations
case OperationActionsTypes.ADD_OPERATION :
    return adapter.addOne((<AircraftsActions> action).payload, state);
case OperationActionsTypes.REMOVE_OPERATION :
    return adapter.removeOne((<AircraftsActions> action).payload, state);
```

Opération d'ajout ou de retrait d'une pièce dans notre exemple

Tests



12/ Bilan NgRx

. La prise en main est laborieuse et demande de réunir de nombreuses compétences d'une part, et de se familiariser avec les concepts clés d'autre part, mais une fois atteinte : la mécanique devient simple et répétitive.

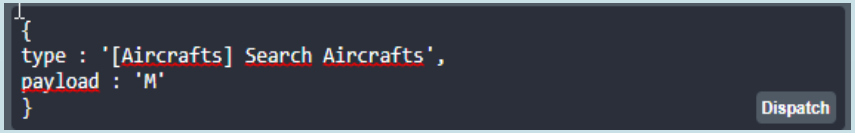
. Du coup, c'est facile de repérer les éléments clés dans une appli et d'ajouter de nouvelles fonctionnalités.

. Facile à tester car les fonctions pures ne sont pas imprévisibles, nous pouvons changer les paramètres d'entrées puis vérifier les sorties.

. L'état de l'appli ne peut être modifié par les composants, il faut passer par le store qui fait le relai avec le reducer, cette centralisation facilite grandement le débogage. De plus, on sait d'où peut venir un problème grâce à l'historisation, accessible via DevTools qui permet de suivre le state et les actions voir de revenir en arrière ou de dispatcher nous même une action.

. Typescript étant typé, NgRx en profite pour verrouiller les actions et lever rapidement les erreurs en phase de dev.

. **En bref : prise en main complexe mais bon compromis in fine**



Ressources

- https://angular.io/
- https://ngrx.io/
- https://rxjs.dev/
- https://blog.angular-university.io/
- https://guide-angular.wishtack.io/
- https://youtu.be/cWydxeZ64ho
- https://github.com/elbabili/fms-ngrx-aircrafts