

# Programmation réactive

**RxJS dans angular,  
firebase et nodeJS**

# **Florent CLAPIÉ**

**Co-fondateur de  
PolyLearn / Switchy.io**

**ancien eistien**



# Reactive Extensions - ReactiveX

An API for asynchronous programming  
with observable streams



# Angular

Framework front se basant sur RxJS

TypeScript (Javascript typé)

=> Single Page Application



# Firebase

BaaS : Base de données temps réel dans le cloud.



# Planning des cours

## 3 jours

- **Jour 1** : Programmation réactive + angular
- **Jour 2** : Intégration de firebase et de RxJS
- **Jour 3** : Projet à rendre pour notation ! 😊

# Déroulement

- 9h-12h30
- 
- 13h30-17h00

# Des questions

# Les outils

- Node Package Manager npm
- Visual Studio code
- angular-cli

# Angular

**et pourquoi pas ?  
VueJS / React / Angular !**

# Angular : GET Started

angular-cli : CLI for Angular applications

```
npm install -g @angular/cli
```

```
ng new yolo
```

```
cd yolo
```

```
ng serve
```

# Notre premier composant

ng g component feature/new-cmp

# Component (src/app/app.component.ts)

```
import { Component } from '@angular/core';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  title = 'Tour of Heroes';
}
```

## Component (src/app/app.component.html)

```
<app-root></app-root>
```

```
<h1>{{title}}</h1>
```

## Ce qu'il faut connaître (part1)

- @if (a > b) / @else
- @for (item of items; track item.id)

Les directives :

- (click)="yolo()"
- input [(ngModel)]="hero.name"

Les pipes :

- {{title | uppercase}}

## Ce qu'il faut connaître (part2)

Component interactions :

- Input()
- Output()

Les services (DI) :

- On verra ça en même temps que firebase

## Ce qu'il faut connaître (part3)

Fonctionnement :

- Dirty Checking (immutable / mutable)
- Two Way Data Binding

# ES6 syntax

```
function() {}
```

```
() => {}
```

```
let x = ['rr']
```

```
let y = ['1', ...x]
```

```
let x = {'0': true}
```

```
let y = {'1': true, ...x}
```

# Playground

## Éditeur en ligne

Stackblitz : Visual Studio Code dans le cloud



Super COOL

# Petit exercice

## Le YOLO challenge

Afficher un nombre de YOLO défini par l'utilisateur

2 composants à faire à minima :

- 1 composant pour entrer le nombre de YOLO
- 1 composant pour display les YOLO

Durée : 1 heure

# Les avantages de Typescript

- Variables typées (interface)
  - AOT (optimisation du code à la compilation)
- => Plus besoin de réfléchir lorsque l'on code !
-  // ou presque

# Interface

```
interface CandyBags {  
    numberOfCandy: number;  
    brand: string;  
    opened: boolean;  
    jeNeSersÀRien?: any;  
}
```

# Pour aller plus loin dans Angular

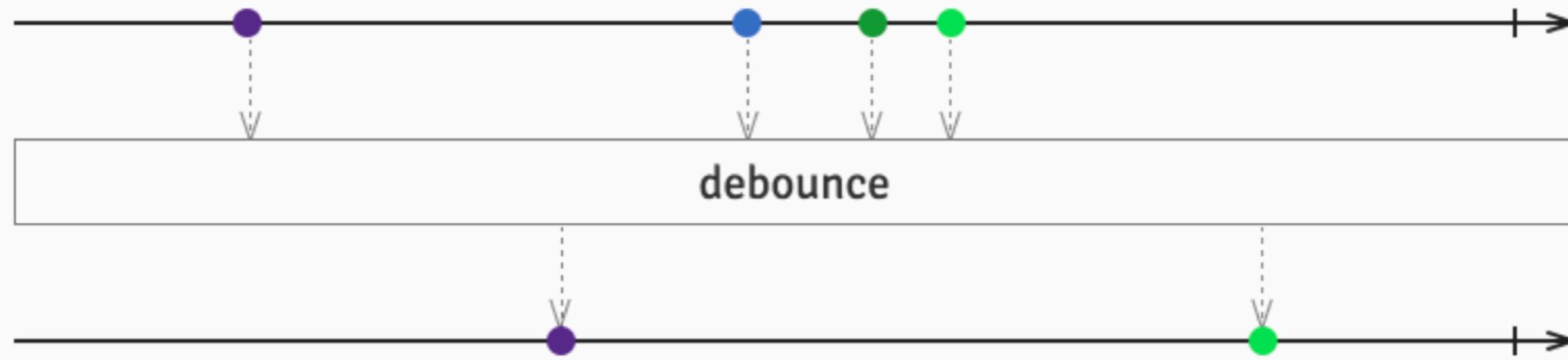
- Smart Components VS Dumb Components
- Lazy Loading
- Architecture REDUX
- Change Detection Strategy: OnPush
- Server Side Rendering (SSR)

# Programmation réactive

## Qu'est ce que c'est ?

The Observer pattern done right

ReactiveX is a combination of the best ideas from  
the **Observer** pattern, the **Iterator** pattern, and **functional programming**



# Stream != Observable

- Streams are pull-based, Observables are push-based. This may sound too abstract, but it has significant consequences that are very concrete.
- Stream can only be used once, Observable can be subscribed to many times

# À quoi ça sert ?

Observable :

- permet de passer des messages entre publishers et subscribers
- pratique pour la programmation asynchrone
- et capable de handle plusieurs valeurs au cours du temps

Observables are declarative—that is, you define a function for publishing values, but it is not executed until a consumer subscribes to it. The subscribed consumer then receives notifications until the function completes, or until they unsubscribe.

-- Doc angular

# Utilisation 1

```
x: string;
subscription: Subscription;
public getObservable(): Observable<string> {
  return of('Yolo')
}

ngOnInit() {
  this.subscription = this.getObservable().subscribe(str => {
    this.x = str;
  });
}
ngOnDestroy() {
  this.subscription.unsubscribe()
}
```

⚠ : il faut penser à unsubscribe dans le  
ngOnDestroy

# Utilisation 2

```
x: Observable<string>;  
  
public getObservable(): Observable<string> {  
    return of('Yolo', 'Yolo1', 'Yolo2')  
}  
  
ngOnInit() {  
    this.x = this.getObservable();  
}
```

Dans le .html

```
{{x | async}}
```

# Les opérateurs

AREA	OPERATORS
Creation	<code>from</code> , <code>fromPromise</code> , <code>fromEvent</code> , <code>of</code>
Combination	<code>combineLatest</code> , <code>concat</code> , <code>merge</code> , <code>startWith</code> , <code>withLatestFrom</code> , <code>zip</code>
Filtering	<code>debounceTime</code> , <code>distinctUntilChanged</code> , <code>filter</code> , <code>take</code> , <code>takeUntil</code>
Transformation	<code>bufferTime</code> , <code>concatMap</code> , <code>map</code> , <code>mergeMap</code> , <code>scan</code> , <code>switchMap</code>
Utility	<code>tap</code>
Multicasting	<code>share</code>

# Ressource intéressante

<http://rxmarbles.com/>

# Naming conventions

Le \$ signifie qu'il s'agit d'un observable

**numberOfStudents\$: Observable<number>;**

# Angular's HttpClient returns observables from HTTP method calls

- Observables do not mutate the server response (as can occur through chained `.then()` calls on promises). Instead, you can use a series of operators to transform values as needed.
- HTTP requests are cancellable through the `unsubscribe()` method.
- Requests can be configured to get progress

# TP Star wars (The Star Wars API)

1. Afficher la liste des planètes (<https://swapi.co/>)
2. Filtrer par terrain (utilisation d'un stream)

Indice : Utilisation de **Subject**

# Recherche par terrain

```
Utilisation de BehaviorSubject();  
let subject = new behaviorSubject();  
subject.onNext('grass')
```

# Limiter le nombre de recherche

**Utilisation de l'opérateur debounce**

**On affiche les résultats (utilisation de CombineLatest)**

```
combineLatest(  
    planets$, search$, (planets, search) => {  
        return planets.filter(  
            planet => planet.terrains.contains(search)  
        )  
    }  
)
```

# Gestion de la pagination

Utilisation de l'opérateur expand

Article de blog sur l'utilisation de expand <https://blog.angularindepth.com/rxjs-understandingexpand-a5f8b41a3602>

# Firebase

# Les services en Angular

```
import { Injectable } from '@angular/core';
@Injectable()
export class FirebaseService {
  constructor() { }
}

src/app/app.module.ts (providers)
providers: [
  FirebaseService,
],
---  
*** Injectons le service
```

```
constructor(
  private firebaseService: FirebaseService
) {}
```

```
---  
*** Mise en place de angularfire
https://github.com/angular/angularfire2  
---  
*** TP : la bibliothèque angevine
Objectif : pouvoir récupérer les livres de Firestore.  
Nouvelles approches à utiliser : les Firestore !  
1. créer une collection de livres dans firestore  
2. créer une collection de catégories dans firestore  
3. faire la page books/:id et afficher le livre + la catégorie  
---  
*** Réalise du code  
API Rick et Morty : https://rickandmortyapi.com/  
Le challenge est d'afficher tous les personnages avec les fonctionnalités de filtre et de search  
((online))((category).png)  
---  
*** Outils de dev  
MongoDB or Firebase  
---  
*** Qui me pousse à ?  
*** Pouvez nous améliorer ?  
---  
*** Deep learning with Angular 🔥
https://medium.com/angular-in-depth/create-your-own-image-classifier-with-angular-and-tensorflow-je-50fb239424  
---  
*** Podcasts API  
---  
*** Utilisation du plateau
```