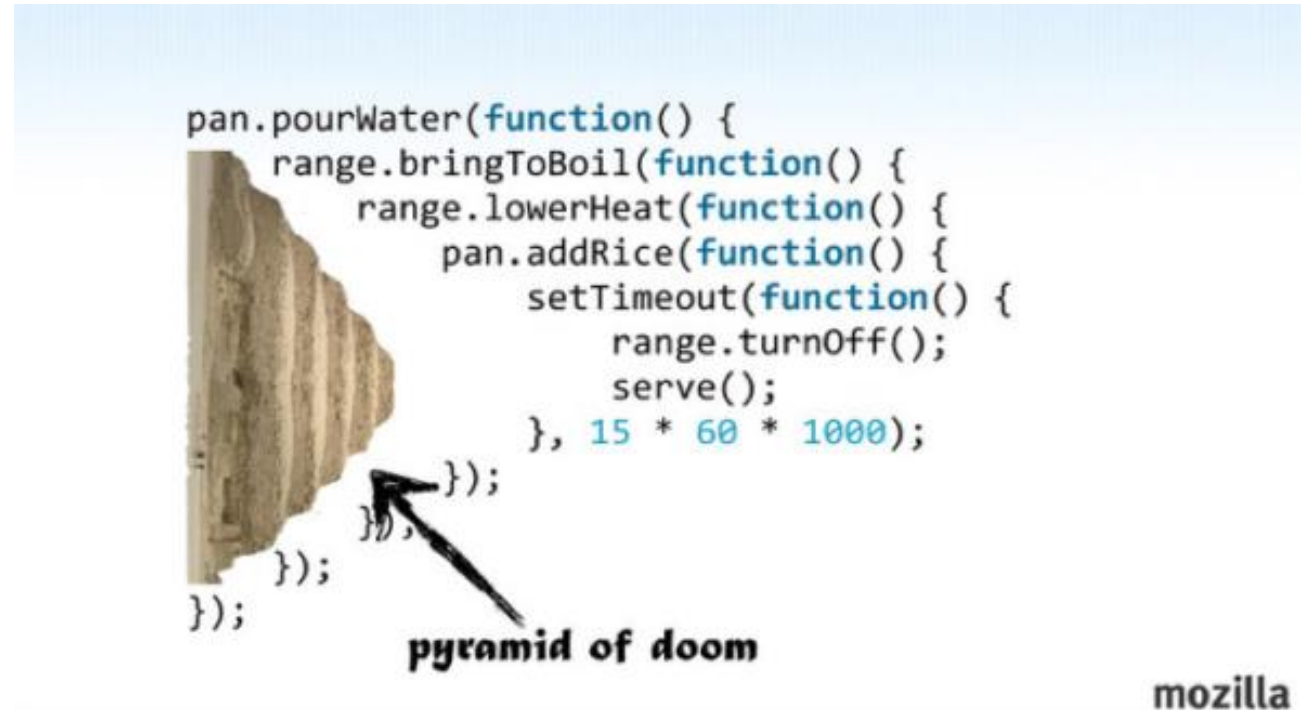




Promesses & Observables

Animé par Mazen Gharbi

Promesses



Promesses

- ▷ Les "promises" font désormais parti des fonctionnalités ES6.
- ▷ Malheureusement les "promises" ES6 n'implémentent pas la méthode ``finally``... *Mais maintenant si !*
- ▷ Les "promises" ne sont pas « lazy » ;
- ▷ Les "promises" ne sont pas annulables.

Promesses

Codons ça ensemble

app.component.ts

```
let promise = new Promise((resolve, reject) => {
  setTimeout(() => {
    resolve('Résultat positif');
  }, 1000);
});

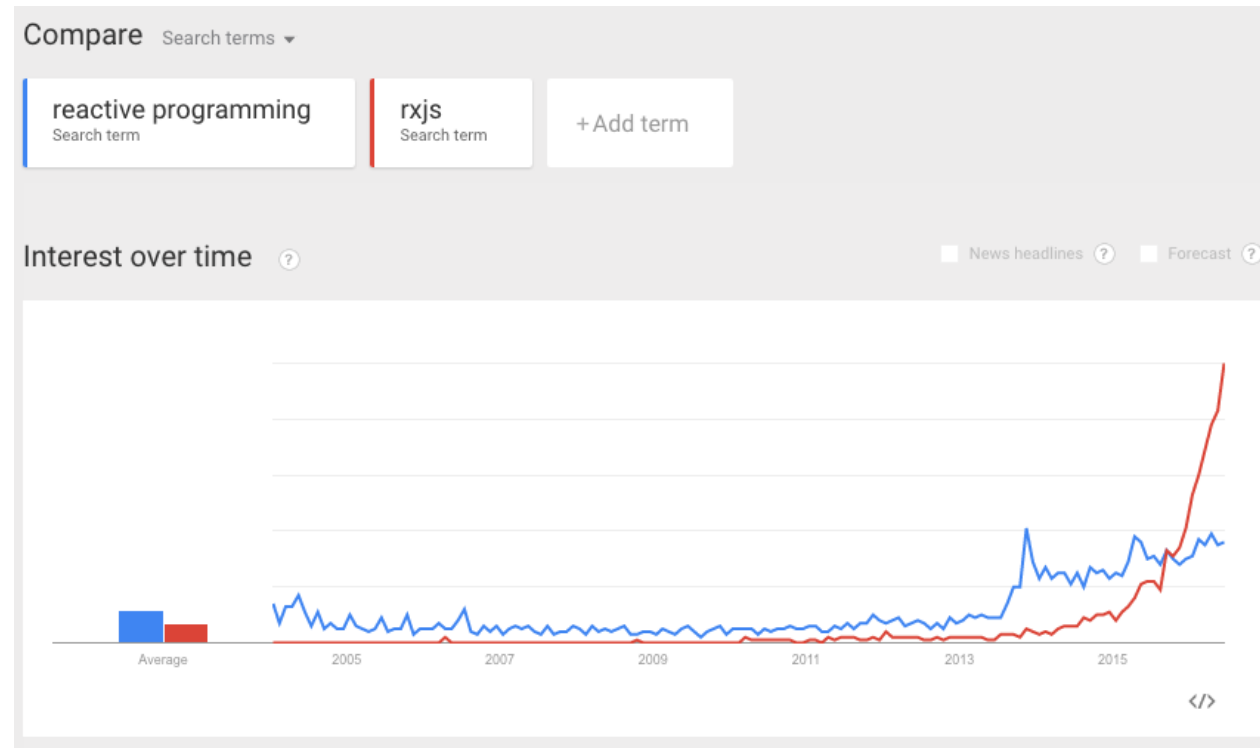
promise
  .then((res) => {
    console.log(res);

    return new Promise((resolve, reject) => resolve('Encore OK!'));
  })
  .then((res) => {
    console.log(res);

    return 'Primitive';
  })
  .then((res) => {
    console.log(res);
    throw new Error('On rentre dans le catch !');
  })
  .catch((err) => {
    console.error(err);
  });
```

Observables

- ▷ L'utilisation des "observables" est l'une des nouveautés d'Angular ;
- ▷ Les "observables" répondent au paradigme "Reactive Programming".



Observables

- ▷ Les "observables" créent un canal de communication qui permet de transmettre plusieurs valeurs au fil du temps.
- ▷ Les "observables" sont massivement utilisés dans Angular et dans la plupart des frameworks modernes qu'ils soient "front-end" ou "back-end".
- ▷ Il est prévu que les "observables" soient intégrés dans ES7.

Création d'observable

<https://stackblitz.com/edit/angular-observables-create>

Contrairement aux promesses, il est possible d'annuler un Observable de l'extérieur :

<https://stackblitz.com/edit/angular-observables-create-cancel>

Création d'un observable

```
this._data = new Observable<string>(observer => {  
  let valueIndex = 0;  
  let valueInterval;  
  let completeTimeout;
```

Représente l'observateur

```
valueInterval = setInterval(() => {
```

```
  observer.next(`message number ${valueIndex}`); // Envoie un message à l'observateur  
  valueIndex++;
```

```
  if (this.throwError) {  
    observer.error(new Error('error')); // Envoie une erreur  
  }
```

```
}, 1000);
```

```
completeTimeout = setTimeout(() => {  
  observer.complete(); // Ferme la communication  
}, 4000);
```

```
/* TEAR DOWN logic. */
```

```
return () => {  
  clearTimeout(completeTimeout);  
  clearInterval(valueInterval);  
}
```

```
});
```

La fonction « Tear down logic » est appelé au moment où l'observateur reçoit l'instruction « complete » ou de désabonne via « unsubscribe ». Cette fonction vide la mémoire manuellement et permettra d'éviter les fuites mémoire

Création d'un observateur

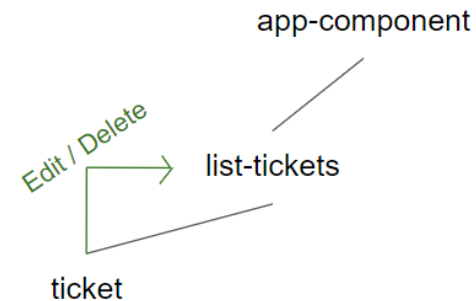
```
public subscribe() {  
    this.finished = false;  
    this.success = false;  
    this.error = null;  
    this.valueList = [];  
  
    this._data  
        .finally(() => this.finished = true)  
        .retry(3)  
        .subscribe(  
            value => this.valueList.push(value), // .next  
            error => this.error = error, // .error  
            () => this.success = true // .complete  
        );  
}
```

app.component.ts

Les observables dans Angular

Les « Outputs » de nos composants

`EventEmitter` hérite d'`Observable`.



- ▷ Les « controls » de formulaires ;
- ▷ Le module http / httpClient

Les opérateurs RxJS

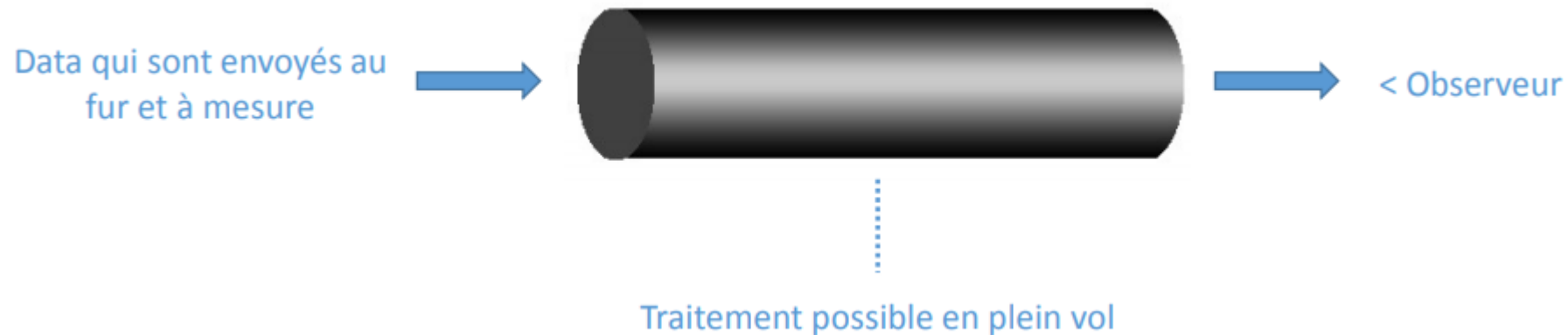
- ▷ Les "observables" disposent de nombreux opérateurs (une centaine) pour transformer le flux de données

<https://github.com/Reactive-Extensions/RxJS/blob/master/doc/gettingstarted/categories.md>

- ▷ Imaginons les Observable comme un tube où un flux de données circule.

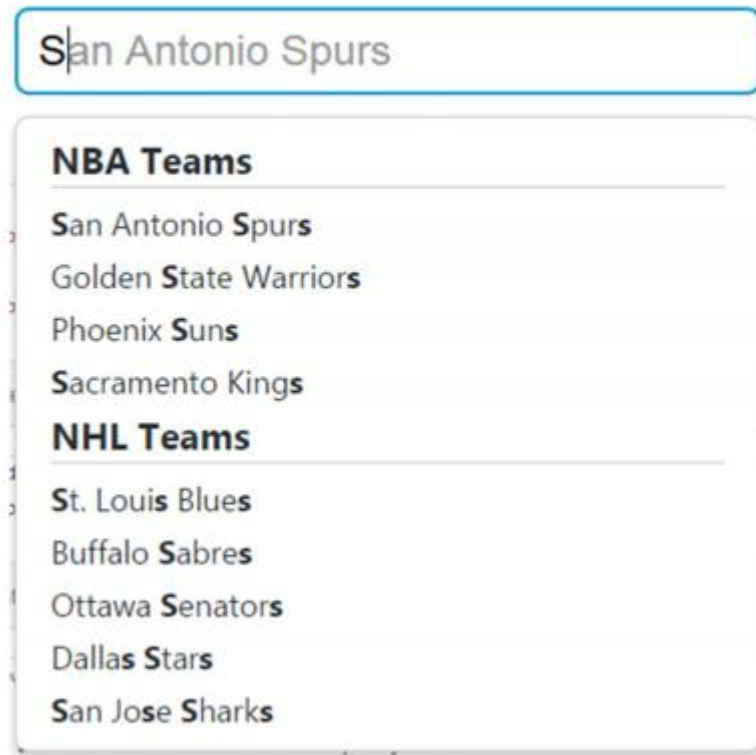
Les opérateurs RxJS

- Programmatiquement, ce flux Observer/Observable fonctionne comme un tableau



Les opérateurs RxJS

Multiple Datasets



San Antonio Spurs

NBA Teams

- San Antonio Spurs
- Golden State Warriors
- Phoenix Suns
- Sacramento Kings

NHL Teams

- St. Louis Blues
- Buffalo Sabres
- Ottawa Senators
- Dallas Stars
- San Jose Sharks

1. Ne pas faire une recherche bêtement à chaque fois que l'utilisateur appuie sur une touche
2. Ne pas requêter le serveur si la requête a déjà été faite précédemment
3. Ne pas lancer plusieurs requêtes serveur en simultané

Les opérateurs RxJS

```
85 let term = new FormControl();  
86  
87 term.valueChanges  
88   .filter(query => query.length >= 3)  
89   .debounceTime(400)  
90   .distinctUntilChanged()  
91   .switchMap(value => autoCompleteService.search(value).catch(error => Observable.of([])))  
92   .subscribe(results => theResults = results);
```

C'est tout.

Cold et Hot Observables



C'est parti

<https://stackblitz.com/edit/angular-observables-create-cancel-hot>

Hot Observable

```
this._hotData = this._data.publish(); // La conversion en HOT Observable !
```



```
this.hotDataSubscription = this._hotData.connect(); // On lance l'observable
```



```
// Enfin, on crée un observateur
```

```
this.subscription = this._hotData.subscribe(value => this.valueList.push(value));
```

Questions