Cours Angular : Observables et Signals

1. Introduction

Angular repose fortement sur la **programmation réactive** pour gérer les données asynchrones, les événements et l'état de l'application.

Deux concepts clés coexistent aujourd'hui : - **Observables (RxJS)** : base historique d'Angular - **Signals** : nouveau mécanisme réactif introduit pour simplifier la gestion d'état

Objectifs : - Comprendre les Observables - Savoir utiliser RxJS dans Angular - Comprendre les Signals - Comparer Observables et Signals - Savoir quand utiliser l'un ou l'autre

---

2. Les Observables (RxJS)

2.1 Qu'est-ce qu'un Observable ?

Un **Observable** représente un flux de données pouvant émettre : - plusieurs valeurs - de manière asynchrone - dans le temps

Exemples : - Appels HTTP - Événements utilisateur - WebSockets - Timers

---

2.2 Création d'un Observable

```
import { Observable } from 'rxjs';

const observable$ = new Observable(observer => {
  observer.next(1);
  observer.next(2);
  observer.complete();
});
```

2.3 Souscription (`subscribe`)

```
observable$.subscribe({
  next: value => console.log(value),
  error: err => console.error(err),
  complete: () => console.log('Terminé')
});
```

⚠️Sans `subscribe`, l'Observable **ne s'exécute pas**.

---

3. Observables dans Angular

3.1 Appels HTTP

```
import { HttpClient } from '@angular/common/http';

constructor(private http: HttpClient) {}

getUsers() {
  return this.http.get<User[]>('/api/users');
}
```

3.2 Utilisation dans un composant

```
users$ = this.userService.getUsers();
```

```
<ul>
  <li *ngFor="let user of users$ | async">
    {{ user.name }}
  </li>
</ul>
```

Le **pipe** `async` gère automatiquement la souscription et la désinscription.

---

4. Opérateurs RxJS essentiels

4.1 `map`

```
this.http.get<User[]>('/api/users')
  .pipe(map(users => users.length));
```

4.2 `filter`

```
.pipe(filter(user => user.active));
```

4.3 `tap`

```
.pipe(tap(data => console.log(data)));
```

4.4 `switchMap`

```
this.route.params.pipe(
  switchMap(params => this.http.get(`/api/user/${params['id']}`))
);
```

---

5. Gestion de la désinscription

5.1 Problème des fuites mémoire

```
subscription!: Subscription;

ngOnInit() {
  this.subscription = observable$.subscribe();
}

ngOnDestroy() {
  this.subscription.unsubscribe();
}
```

5.2 Bonne pratique : `async` pipe

♎Toujours privilégier `async` pipe dans le template.

---

6. Introduction aux Signals

6.1 Qu'est-ce qu'un Signal ?

Un **Signal** est une valeur réactive : - simple - synchrone - automatiquement suivie par Angular

♎Alternative plus simple aux Observables pour la gestion d'état.

6.2 Création d'un Signal

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

count = signal(0);
```

Lecture :

```
this.count();
```

Modification :

```
this.count.set(1);
this.count.update(v => v + 1);
```

---

7. Utilisation des Signals dans le template

```
<p>Compteur : {{ count() }}</p>
<button (click)="count.update(v => v + 1)">+1</button>
```

---

8. `computed` et `effect`

8.1 `computed`

```
doubleCount = computed(() => this.count() * 2);
```

8.2 `effect`

```
effect(() => {
  console.log('Valeur changée :', this.count());
});
```

`effect` réagit automatiquement aux changements.

---

9. Interaction Observables ↔ Signals

9.1 Observable vers Signal

```
import { toSignal } from '@angular/core/rxjs-interop';

users = toSignal(this.userService.getUsers(), { initialValue: [] });
```

9.2 Signal vers Observable

```
import { toObservable } from '@angular/core/rxjs-interop';

users$ = toObservable(this.users);
```

---

10. Cas pratique comparatif

Avec Observable

```
users$ = this.http.get<User[]>('/api/users');
```

Avec Signal

```
users = signal<User[]>([]);

loadUsers() {
  this.http.get<User[]>('/api/users').subscribe(data =>
this.users.set(data));
}
```

---

11. Observables vs Signals

Observables	Signals
Asynchrones	Synchrones
Puissants (RxJS)	Simples
Flux multiples	État local
Plus complexes	Très lisibles

---

12. Bonnes pratiques

- Utiliser **Observables** pour : HTTP, events, streams
- Utiliser **Signals** pour : état local, UI state
- Ne pas remplacer RxJS partout
- Combiner intelligemment les deux

---

13. Résumé

- RxJS est essentiel dans Angular
- Les Observables gèrent l'asynchrone
- Les Signals simplifient la gestion d'état
- Angular moderne encourage leur complémentarité

♎ Prochaine étape recommandée : **State management, NgRx vs Signals, architecture moderne Angular**