Fiche récap : Maîtrisez les Observables pour dynamiser vos apps Angular

Développement

1. Créer un Observable et y souscrire avec subscribe() ou le pipe async
2. Transformer et filtrer les valeurs avec les opérateurs bas niveau
3. Éviter les fuites mémoire avec take() et takeUntilDestroyed()
4. Passer d’un Observable à un autre avec les opérateurs haut niveau

Code 💻

```typescript
// Créer un Observable avec interval()
interval$ = interval(1000);

// Souscrire à un Observable
this.interval$.subscribe();

// Chaîner des opérateurs avec pipe()
interval$ = interval(1000).pipe(
  filter(value => value % 2 === 0),
  map(value => `Valeur ${value}`),
  tap(text => console.log(text))
);

// Utiliser switchMap pour annuler l’opération précédente
autoComplete$ = input$.pipe(
  switchMap(query => fetchSuggestions(query))
);

```

Bonnes pratiques 👍

- ✅ Utiliser pipe() pour chaîner les opérateurs sur un Observable
- ✅ Afficher les émissions dans le DOM avec le pipe async
- ✅ Employer map() pour transformer les valeurs émises
- ✅ Filtrer les valeurs utiles avec l’opérateur filter()
- ✅ Réagir sans modifier les données avec l’opérateur tap()
- ✅ Limiter les émissions avec take() si leur nombre est connu
- ✅ Se désabonner automatiquement avec takeUntilDestroyed()
- ✅ Choisir l’opérateur haut niveau adapté à chaque contexte

Erreurs classiques 👎

- ❌ Oublier de se désabonner d’un Observable actif
- ❌ Utiliser subscribe() sans stratégie de désabonnement
- ❌ Afficher un Observable dans le DOM sans pipe async
- ❌ Appliquer les opérateurs dans un ordre incohérent
- ❌ Ignorer le typage explicite d’un Observable
- ❌ Souscrire plusieurs fois au même Observable inutilement
- ❌ Utiliser concatMap avec des flux très fréquents
- ❌ Laisser des Observables actifs après destruction du composant

Définitions 🔎

Observable
Objet RxJS émettant des valeurs typées dans le temps, pouvant se compléter ou émettre une erreur.

pipe()
Méthode permettant d’appliquer une suite d’opérateurs à un Observable.

subscribe()
Méthode qui permet de souscrire à un Observable.

take()
Opérateur qui limite un Observable à un nombre défini d’émissions avant de se compléter.

switchMap()
Opérateur haut niveau qui annule le flux en cours pour en lancer un nouveau dès qu'une nouvelle valeur est émise.