

Angular 18 & 19

Communauté de devs

Entrer →

Pourquoi cette présentation

Voyons ensemble les dernières nouveautés, quelques exemples pertinents. Et les bonnes pratiques associées.

- **Nouveautés** -> Présentation générale des nouveautés Angular.
- **Signal vs zone.js** -> L'intérêt de se passer de zone.js.
- **Signal vs Observable et complémentarité** -> Angular réactif.
- **Migration** -> Aide à la montée de version.
- **Architecture** -> Un exemple d'architecture employé en mission.

Contrôle de flux (@if, @for, @switch)

@IF

```
1 @if (user) {  
2   <div>  
3     Bonjour {{ user.name }} !  
4   </div>  
5 } @else {  
6   <p>Utilisateur inconnu</p>  
7 }
```

@FOR

```
1 <ul>  
2   @for (item of items; track item) {  
3     <li>{{ item }}</li>  
4   }  
5 </ul>
```

@SWITCH

```
1 @switch (status) {  
2   @case ('loading') {  
3     <p>Chargement...</p>  
4   }  
5   @case ('success') {  
6     <p>Succès </p>  
7   }  
8   @default {  
9     <p>Erreur </p>  
10 }  
11 }
```



Composant standalone et lazy-loadé

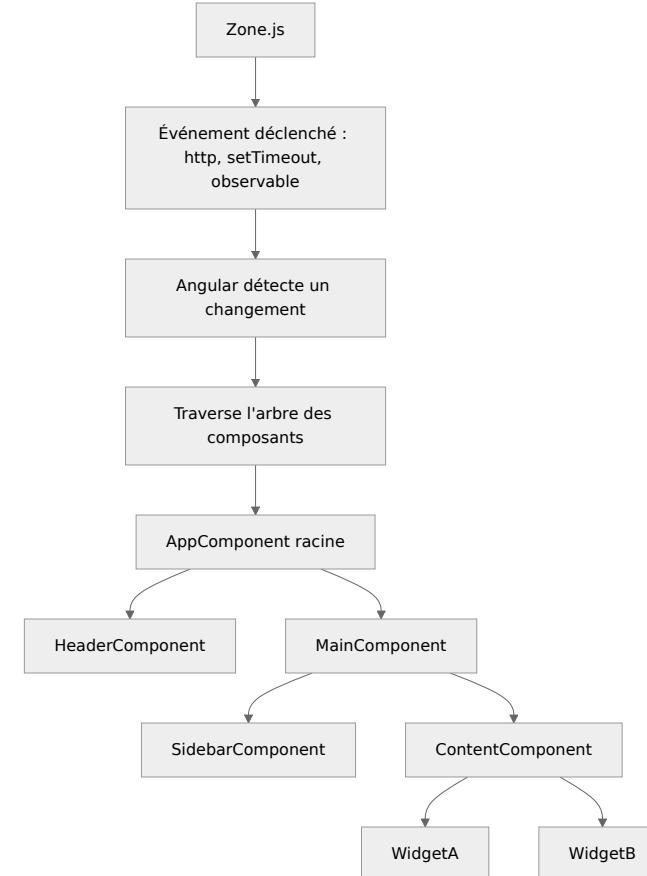
```
@Component({
  standalone: true,
  selector: "app-user-card",
  imports: [CommonModule],
  templateUrl: "./user-card.component.html",
})
export class UserCardComponent {
  user$ = this.userService.getUser();
}
```

```
@defer (when product$ | async) {
<product-view [product]="product" />
} @placeholder {
<div class="loader">Chargement du produit...</div>
} @loading {
<app-loading-spinner />
} @error {
<p class="text-red-500">Erreur de chargement ☹</p>
}
```

Zone.js pour une application réactive

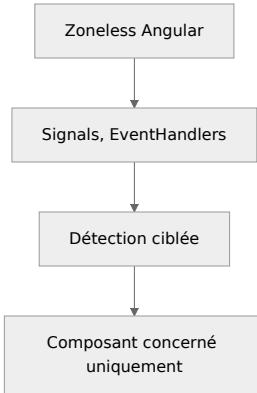
- Librairie JavaScript qui **patch** les API async
(setTimeout, Promises, XHR...)
- Angular (depuis V2) s'appuie dessus pour **savoir automatiquement quand lancer la détection des changements**

Trop magique et trop gourmand



Zoneless pour une meilleure application réactive

- Dépendance externe progressivement retiré d'Angular.
- Angular au travers des signals, sait exactement quel composant mettre à jour.



```
TypeScript
providers: [
  provideZoneChangeDetection('noop') //in main.ts
]

changeDetection: ChangeDetectionStrategy.OnPush

const count: Signal<number> = signal(0);
const double = computed(() => count() * 2);

effect(() => {
  console.log("Double:", double());
```

```
Template html
<div>
  <p>{{count()}}</p>
  <p>{{double()}}</p>
</div>
```

Signal props binding

- Intègre les capacités des signals (computed, effect, etc)

Input()

```
1 protected name = input.required<string>();
```

```
1 <mon-composant-enfant [name]='Grégory'></mon-composant-enfant>
```

Output()

```
1 protected setName = output<string>();  
2  
3 this.setName.emit('Grérory')
```

```
1 <mon-composant-enfant (setName)='setName($event)'></mon-composant-enfant>
```

Signal vs Observable

- Pas de librairie externe
- Aucun risque de fuite de mémoire (unsubscribe automatique)

```
<div>
  <p>{{name$ | async}}</p>
</div>
```

```
<div>
  <p>{{name()}}</p>
</div>
```

```
this.name$.subscribe(data => console.log(data))
```

```
console.log(this.name())
```

Signal ET Observable

Observable

- Flux de données asynchrones : appels HTTP, WebSocket, timers, etc.
- Opérations complexes sur les flux : transformation, filtrage, combinaison de plusieurs flux.
- `toObservable()`

Signal

- Réactivité locale et simple : gestion d'état dans un composant ou service.
- Optimisation des performances : Angular peut mieux détecter les changements.
- `toSignal()`

```
const users$ = this.searchControl.valueChanges.pipe(  
  debounceTime(300),  
  distinctUntilChanged(),  
  filter(query => query.length > 2),  
  switchMap(query =>  
    this.userService.searchUsers(query).pipe(  
      catchError(() => of([])) // En cas d'erreur, on retourne une liste vide  
    )  
  )  
);
```

Rappel sur les pipes RxJS

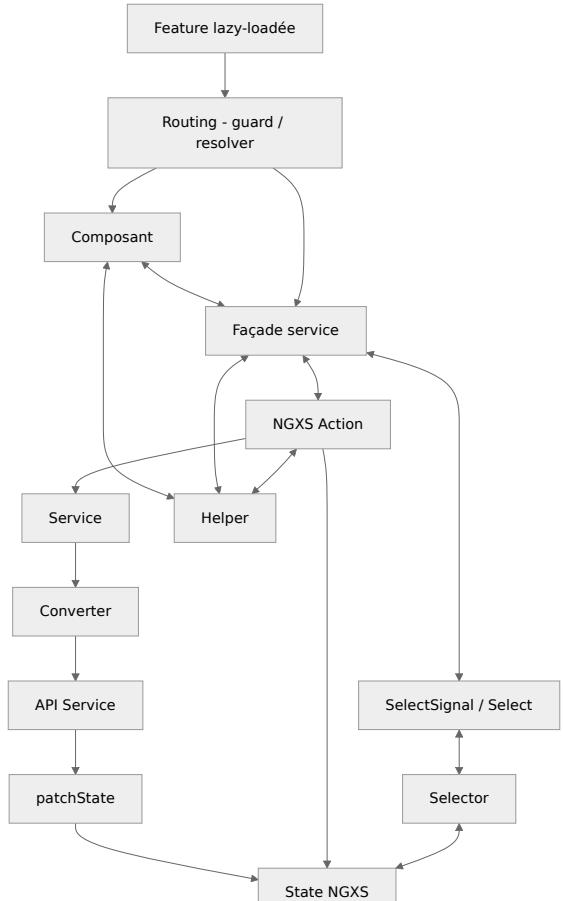
Les pipes permettent de chaîner des opérateurs pour transformer ou manipuler des observables :

- map(fn) : transforme les valeurs.
- switchMap(fn) : annule l'observable précédent et en crée un nouveau.
- mergeMap(fn) : fusionne plusieurs observables.
- concatMap(fn) : exécute les observables les uns après les autres.
- filter(fn) : filtre les valeurs.
- take(n) : prend les n premières valeurs.
- debounceTime(ms) : attend un délai avant d'émettre.
- distinctUntilChanged() : ignore les doublons consécutifs.
- catchError(fn) : intercepte les erreurs.

Migrations

- migration vers standalone => `ng generate @angular/core:standalone`
- Adoption du nouveau contrôle de flux (@if, @for, @switch) => `ng generate @angular/core:control-flow`
- Migration vers l'API inject() pour l'injection de dépendances => `ng generate @angular/core:inject`
- Migration des `@Input()` vers les signaux (`input()`) => `ng generate @angular/core:signal-input-migration`
- Chargement paresseux des routes avec des composants autonomes => `ng generate @angular/core:route-lazy-loading`
- Migration vers le nouveau système de build application => `ng update --migrate-only explicit-standalone-flag`
=> <https://angular.dev/reference/migrations>
=> <https://angular.dev/update-guide>

Exemple d'architecture



```
< individual-retirement
  > components
  > constants
  > converters
  > features
    > arbitration
      > components
      > constants
      > converters
      > forms
      > guards
      > helpers
      > mocks
      > models
      > pages
      > resolvers
    > services
      > mock
        TS arbitration-api.service.spec.ts
        TS arbitration-api.service.ts
        TS arbitration.service.spec.ts
        TS arbitration.service.ts
    > store
      > mock
        TS arbitration-facade.service.ts
        TS arbitration.action.ts
        TS arbitration.navigate.actions.ts
        TS arbitration.navigate.state.spec.ts
        TS arbitration.navigate.state.ts
        TS arbitration.selectors.spec.ts
        TS arbitration.selectors.ts
        TS arbitration.state.spec.ts
        TS arbitration.state.ts
      > validators
        TS arbitration-routing.module.ts
        TS arbitration.module.ts
```

■ helper

Index

1. [Angular 18 & 19](#)
2. [Pourquoi cette présentation](#)
3. [Contrôle de flux \(@if, @for, @switch\)](#)
4. [Composant standalone et lazy-loadé](#)
5. [Zone.js pour une application réactive](#)
6. [Zoneless pour une meilleure application réactive](#)
7. [Signal props binding](#)
8. [Signal vs Observable](#)
9. [Signal ET Observable](#)
10. [Rappel sur les pipes RxJS](#)
11. [Migrations](#)
12. [Exemple d'architecture](#)
13. [Index](#)

Merci de votre attention

