

TrackBy

# TrackBy

- Tous les éléments à l'intérieur de la balise portant la directive `*ngFor` sont dupliqués pour chaque membre de la collection
- Lorsque un membre de la collection change, Angular détruit tous les éléments correspondants, et recrée des nouveaux éléments avec le nouveau membre
- Le fait de détruire et de recréer des éléments du DOM est une opération coûteuse, qui peut ralentir les performances de l'applciation si elle est faite trop souvent
- Pour savoir si un membre de la collection par défaut, Angular compare les références des objets

- On peut définir une autre fonction pour comparer les objets

```
<div *ngFor="let item of collection; trackBy: trackByFunction">  
  ...
```

```
...  
trackByFunction(index: number, item: any): any {  
  return item  
}  
...
```

- index est la position de l'objet dans la liste, item est une référence à l'objet
- Le retour de la fonction trackBy est utilisé pour déterminer si l'objet a changé
- Il est très important que la fonction trackBy renvoie un résultat unique pour chaque élément de la collection

View queries

# @ViewChild

- Retour sur les variables de template :

```
<audio #audio src="../assets/sample.mp3" ></audio>  
<button (click)="audio.play()">Play</button>  
<button (click)="audio.pause()">Pause</button>
```

- Dans certains cas, on veut pouvoir manipuler les éléments du template dans le composant
- Pour cela, Angular met à disposition les décorateurs d'attributs @ViewChild et @ViewChildren

```
@ViewChild('audio')  
audioRef: ElementRef<HTMLAudioElement>
```

- ViewChild fait une requête sur le template, et assigne le premier résultat correspondant à l'attribut du template

- Il est possible de requêter :
- Un élément du dom avec une variable de template associée

```
@ViewChild('audio')  
audioRef: ElementRef<HTMLAudioElement>
```

- Un composant ou une directive


```
@ViewChild(ChildComponent)  
componentRef: ChildComponent
```

```
@ViewChild(HighlightDirective)  
directiveRef: HighlightDirective
```

- Un provider dans les composants ou directives fils

```
@ViewChild(CounterService)  
childServiceRef: CounterService
```

# Quelques précisions sur ViewChild :

- ViewChild renvoie uniquement le premier résultat correspondant à la requête
- Sur le type du retour :
  -  Comme souvent avec les décorateurs, pas de type-checking
  - Pour un composant ou une directive, le type de retour est la classe elle-même
  - Pour un élément HTML, le type de retour est l'interface correspondant wrappée dans un ElementRef<T>
  - Si la requête n'a pas de résultat, le retour est undefined
- La requête n'est pas récursive : elle est limitée au template de composant
- Si au cours de la vie de votre composant le résultat de la requête change, la valeur de l'attribut change également

# @ViewChildren

- Le fonctionnement de @ViewChildren est très similaire a @ViewChild, sauf qu'il renvoie tous les résultats de la requête dans une QueryList
- Il est possible de fournir plusieurs sélecteurs, séparés par une virgule
- Comme pour ViewChild, le contenu de la QueryList change avec le composant
- Il est possible de récupérer un observable sur une QueryList qui notifie les subscribers des changements




# read

- La propriété read permet de changer le type de retour de la requête,
- Exemple pour avoir l'élément du DOM correspondant à un composant fils, plutôt que le composant lui-même

```
@ViewChild(ChildComponent, {read: ElementRef})  
childRef!: ElementRef<HTMLElement>
```

- Ou une directive sur un composant en particulier

```
@ViewChild(ChildComponent, {read: HighlightDirective})  
childDirectiveRef?: HighlightDirective
```

-  Manipuler directement le DOM peut rendre votre application vulnérable

# Exercice

- Coder un composant qui permet de choisir une piste audio parmi une liste
- Lorsque l'on change de piste, la lecture actuelle est remise à zero