Utiliser les composants Angular

Framework front-end

Source : https://openclassrooms.com/fr/courses/7471261-debutez-avec-angular/7471268-tirez-un-maximum-de-ce-cours

Table des matières

Afficher ses composants c'est bien gentil mais si on n'affiche que des contenus dans les fichiers HTML on va vite avoir fait le tour du site.

Voyons alors comment passer des données directement depuis les composants.

Préparation

On va utiliser un truc qu'on connaît bien maintenant : on va passer des propriétés à la classe FaceSnap mais avant de faire ça, on va changer un comportement de TypeScript qui nous oblige à initialiser toutes les variables que l'on déclare. On va donc se rendre dans le fichier **tsconfig.json** et ajouter la propriété suivante dans le tableau **compilerOptions** :

```
    "strictPropertyInitialization": false,
```

Une fois ceci fait, on va ajouter des propriétés dans notre classe **face-snap.ts** et également les initialiser en important « OnInit », une fonction appelée par Angular à chaque initialisation de composant. Notre fichier devient donc ceci **:**

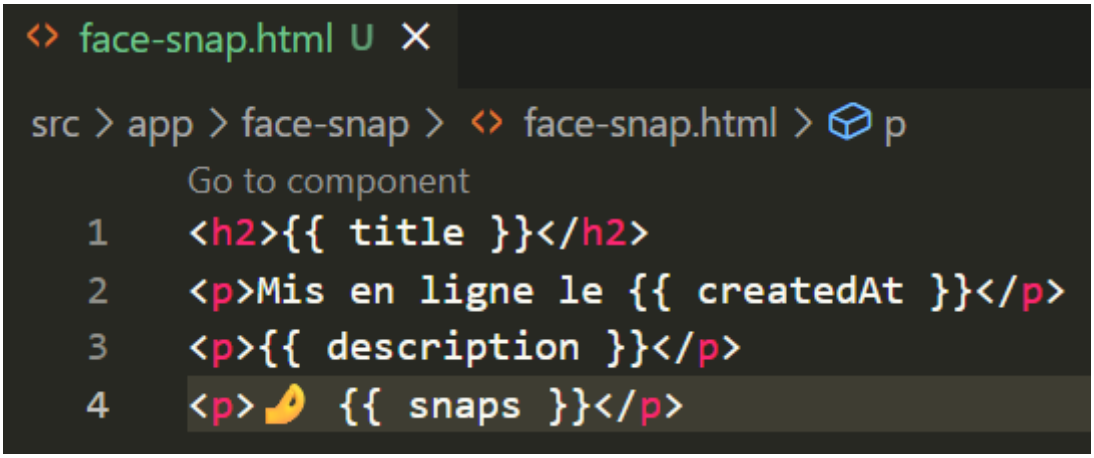
src > app > face-snap > TS face-snap.ts > FaceSnap

```ts
import { Component, OnInit } from '@angular/core';

@Component({
  selector: 'app-face-snap',
  imports: [],
  templateUrl: './face-snap.html',
  styleUrl: './face-snap.scss',
})
// Export de la classe et iplementation de l'interface OnInit
export class FaceSnap implements OnInit {
  title: string;
  description: string;
  createdAt: Date;
  snaps: number;

  // Cette méthode est appelée par défaut à chaque instance du composant
  // On lui dit donc ce qu'elle doit faire. En l'occurence ça ressemble pas mal à un constructeur :)
  ngOnInit() {
    this.title = 'Archibald';
    this.description = 'Mon meilleur ami depuis toujours !';
    this.createdAt = new Date();
    this.snaps = 5;
  }
}
```

String interpolation

La méthode la plus simple et de loin : Une fois les propriétés du composant déclarée, on peut y avoir accès avec une syntaxe très simple : les doubles accolades **{{ }}**

Voyons comment ça se passe dans notre composant. Modifions le fichier **face-snap.html** :



Et voilà. Simple. Basique.

Attribute binding

Ajoutons une propriété imageUrl dans notre classe face-snap, en tant que string, et initialisons-la avec cette valeur : `https://cdn.pixabay.com/photo/2015/05/31/16/03/teddy-bear-792273_1280.jpg`

On pourrait utiliser la **string interpolation** pour passer la propriété imageUrl dans l'attribut src de la balise <img>, comme ceci :

```
<img src="{{ imageUrl }}" alt="Mon image">
```

Mais il existe une autre façon de faire, l'**attribute binding**, où l'on va laisser Angular placer la valeur de la propriété en question dans l'attribut entre crochets, comme ceci :

```
<img [src]="imageUrl" alt="Mon image">
```

Ici, Angular va comprendre qu'il doit mettre la valeur de la propriété **imageUrl** dans l'attribut **src**, au lieu de fonctionner avec cette espèce de concaténation avec les doubles accolades.

Comment choisir ?

En général, on va utiliser l'**attribute binding** pour gérer les attributs de nos éléments HTML susceptibles de **changer** (des classes rajoutées au **hover** par exemple, ou la source d'une image…). En revanche pour ce qui est du **contenu** des balises HTML en question, on va utiliser la **string interpolation**.

Évènements

Et si on permettait à nos utilisateurs d'augmenter le nombre de snaps en cliquant sur un bouton ? Pour faire ça, on va rajouter une méthode dans la classe FaceSnap (**face-snap.ts**) :

```
onAddSnap() {
  this.snaps++;
}
```
Puis un bouton dans face-snap.html en utilisant une syntaxe très similaire à celle de l'**attribute binding** : l'**event binding**. Comme ceci :

```
<p>
    🤌 {{ snaps }}
    <button (click)="onAddSnap()">Oh Snap!</button>
</p>
```

Ici, puisqu'on est sur la liaison d'une **méthode** et non plus d'une **propriété**, on va utiliser des parenthèses autour de l'évènement **()** plutôt que des crochets autour de l'attribut **[]**

*C'est pareil de mettre **onclick** comme attribut HTML non ?*

Ça pourrait, mais non. La différence subtile est que si on utilise l'attribut **onclick**, alors Javascript va chercher une fonction Javascript déclarée au préalable. Or, **onAddSnap()** n'est pas une fonction déclarée au préalable mais une **méthode de la classe FaceSnap**. C'est donc à Angular qu'on va laisser le soin de lier le bouton à la méthode, tout en restant dans le **scope** de ce composant, en utilisant cette syntaxe.

Réutiliser un composant avec plusieurs données

L'intérêt de s'embêter à structurer un composant, c'est bien de le rendre réutilisable non ? Pour cela, nous allons emprunter un peu de logique à la POO, et nous allons garder notre composant FaceSnap, mais l'utiliser avec un **modèle** qui me permettra de créer plusieurs FaceSnap.

Nous allons donc créer un sous-dossier **models** dans le répertoire **src/app**, puis nous allons créer le fichier face-snap-model.ts

src > app > face-snap > models > TS face-snap-model.ts > FaceSnapModel

```ts
export class FaceSnapModel {
  title: string;
  description: string;
  createdDate: Date;
  snaps: number;
  imageUrl: string;
  hasSnapped: boolean;

  constructor(title: string, description: string, imageUrl: string, createdDate: Date, snaps: number) {
    this.title = title;
    this.description = description;
    this.imageUrl = imageUrl;
    this.createdDate = createdDate;
    this.snaps = snaps;
    this.hasSnapped = false;
  }
}
```

Voyez comme on a la même logique qu'en PHP : On nomme la classe, on déclare et on type ses propriétés, puis on a un **constructeur**.

Maintenant, il faut dire à notre composant **FaceSnap** d'accepter des objets de type **FaceSnapModel** en tant que propriété. Pour cela, nous allons **importer notre modèle**, puis la classe **Input** depuis angular/core, et déclarer notre propriété de la classe FaceSnap comme ceci :

```ts
@Input() faceSnap: FaceSnapModel;
```

Nous pouvons supprimer le reste dans le fichier face-snap.ts, y compris l'implémentation de OnInit.

Enfin, dans le composant **racine** (app.ts) nous allons importer le modèle, puis transvaser des données dans le modèle, comme ceci :

```ts
export class App {
  mySnap: FaceSnapModel;
  ngOnInit() {
    this.mySnap = new FaceSnapModel(
      'Archibald',
      'Mon meilleur ami depuis tout petit !',
      new Date(),
      0,
      'https://cdn.pixabay.com/photo/2015/05/31/16/03/teddy-bear-792273_1280.jpg',
    );
  }
}
```

Puis nous allons dire à notre template app.html d'utiliser de l'**attribute binding** pour que notre composant **FaceSnap** se voit passer la propriété faceSnap, qui prendra la valeur de **mySnap** déclaré dans **app.ts**

```
<app-face-snap [faceSnap]="mySnap"></app-face-snap>
```

La ligne :

```
@Input() faceSnap: FaceSnapModel;
```

Dans face-snap.ts nous a permis de faire passer la valeur de cette propriété par une autre class, la classe app.ts.

La propriété faceSnap de la classe face-snap.ts est donc devenue utilisable comme si elle était là depuis le début, sauf qu'elle a été transférée depuis le composant **racine**.

Grâce à ce fonctionnement, on peut maintenant envisager de charger plusieurs fois le même composant avec un snap différent à chaque fois, comme ceci :

```
<app-face-snap [faceSnap]="mySnap"></app-face-snap>
<app-face-snap [faceSnap]="mySnap2"></app-face-snap>
<app-face-snap [faceSnap]="mySnap3"></app-face-snap>
```

Le composant sera chargé de manière indépendante à chaque appel dans sa balise, mais à chaque fois avec un snap différent.