Tutoriel d'utilisation du framework Angular

Logan SAGET

30/01/2026

Table des matières

1 Initialisation d'un projet :

1.1 Commandes utiles à la création d'un projet (dans le dossier voulu) :

```
ng new [nomProjet]
// Ou encore
ng new [nomProjet]
     --style=scss  // Langage du style
     --skip-tests=true

cd nomProjet
ng serve // Lancement du serveur
```

2 Bases d'un projet Angular :

2.1 Création et utilisation d'un composant :

Génération automatique :

```
    ng generate component page-accueil
```

Cette commande génère 3 fichiers :

- fichier HTML
- fichier de style
- fichier de script

Dans le fichier script on peut retrouver la classe qui défini ce composant :

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

@Component({
    selector: 'app-test',
    imports: [],
    templateUrl: './test.html',
    styleUrls: ['./test.scss'],
})

export class Test {

}
```

On pourra bien sûr rajouter des lignes dans cette classe (comme des attributs par exemple).

```
export class Comp1{
  title!: string;
  description!: string;
  createdAt!: Date;
  snaps!: number;
}
```

Afin d'initialiser ces attributs (dès l'appel du composant), on implémente l'interface OnInit :

```
export class Comp1 implements OnInit{
  title!: string;
  description!: string;
  createdAt!: Date;
  snaps!: number;

  ngOnInit(): void {
    this.title = "premierComposant";
    this.description = "premierComposant";
    this.createdAt = new Date();
    this.snaps = 0;
  }
}
```

Une fois initialisée, on peut utiliser ces variables dans le fichier.html du composant :

```
<H2>{{title}}</H2>
<p>{{description}}</p>
```

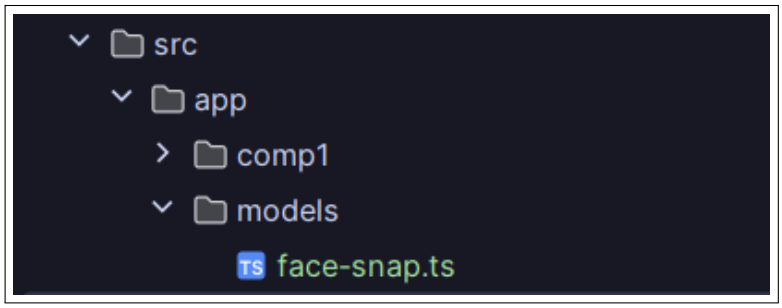
Ce qui donnera :

Pour ajouter une image, on la défini d'abbord comme nouvel attribut, puis on utilise une balise src entre "[]" :

```
<img [src]="url"> <!-- url etant un attribut defini dans
    componant.js -->
```

2.2 Personnalisation des composants :

Dans un premier lieu, on crée un package dans l'application dans lequel on va ajouter des classes :



Dans cette classe, on défini les attributs qui se trouvaient initialement dans le fichier ts du composant.

```
export class FaceSnap {
  constructor(public title: string,
              public description: string,
              public createdAt: Date,
              public snaps: number,
              public url: string) {}
      }
```

Dans le fichier "app.ts", nous allons maintenant déclarer des objets de type FaceSnap :

```
export class App implements OnInit {

  snap1!: FaceSnap;
  snap2!: FaceSnap;
  snap3!: FaceSnap;

  ngOnInit() {
    this.snap1 = new FaceSnap("premierComposant","
       premierComposant",new Date(),0,url);
    this.snap2 = new FaceSnap("2emeComposant","2
       emeComposant",new Date(),10,url);
    this.snap3 = new FaceSnap("3emeComposant","3
       emeComposant",new Date(),100,url);
  }
}
```

Une fois la classe crée, on peut retirer les attributs du composant et simplement crée un objet de type FaceSnap :

```
export class Comp1 implements OnInit{
  @Input() faceSnap!: FaceSnap;
}
```

Enfin, on peut appeler nos composants dans l'html de l'application en initialisant l'attribut crée :

```
<app-comp1 [faceSnap]="snap1"></app-comp1>

<app-comp1 [faceSnap]="snap2"></app-comp1>

<app-comp1 [faceSnap]="snap3"></app-comp1>
```

Avec cette manipulation, on peut personnaliser nos composants.

Il ne faut pas oublier dans le html du composant de remplacer les appels avec faceSnap.attribut :

```
<H2>{{faceSnap.title}}</H2>
<p>{{faceSnap.description}}</p>
<img [src]="faceSnap.url">
```

3 Gestion des événements :

La gestion des événements est beaucou plus simple grâce à angular. Il suffit de créer dans le TypeScript du composant des actions. Par exemple, on veut pouvoir liker un post et retirer le like :

```
export class Comp1 implements OnInit{
  @Input() faceSnap!: FaceSnap;

  snapButton!: string;
  snapOrNot!: boolean;

  ngOnInit(): void {
    this.snapOrNot = false;
    this.snapButton = "snaps"
  }

  onAddSnap() :void{
    if(this.snapOrNot === false){
      this.faceSnap.addSnap();
      this.snapOrNot = true;
      this.snapButton = "UnSnap";
    }else{
      this.faceSnap.delSnap();
      this.snapOrNot = false;
      this.snapButton = "snaps";
    }
  }
}
```

Enfin, on lie cet évènement à un bouton, ici, on parle de l'évènement "click", qu'on va donc écrire entre parenthèse dans la balise du bouton pour le définir :

```
<p>
  <button (click)="onAddSnap()">{{snapButton}}</button>
  {{faceSnap.snaps}}
</p>
```

4 Mise en place d'un service :

4.1 Création du service :

La mise en place d'un service est utile afin de regrouper les méthodes qui seront utiles à plusieurs composants. On les utilise aussi afin de faire appel à des API.

On définit un service de la manière suivante :

```
    @Injectable({
  providedIn: 'root'
})
export class FaceSnapsService {
  private faceSnaps: FaceSnap[] = [
    new FaceSnap("premierComposant", "premierComposant",
       new Date(), 0)
    new FaceSnap("2emeComposant", "2emeComposant", new
       Date(), 10)
    new FaceSnap("3emeComposant", "3emeComposant", new
       Date(), 100)
  ];
}
```

4.2 Utilisation d'un service :

En utilisant la propriété injectable, on peut l'utiliser dans les différentes classes en l'injectant dans les constructeurs :

```

  Export class SingleFaceSnap{
   constructor(private snapsService: FaceSnapsService)
  }
```

Dans notre exemple, notre service possède des méthodes afin de trouver via un id un snap :

```
    @Injectable({
  providedIn: 'root'
})
export class FaceSnapsService {
  private faceSnaps: FaceSnap[] = [
    new FaceSnap("premierComposant", "premierComposant",
       new Date(), 0)
    new FaceSnap("2emeComposant", "2emeComposant", new
       Date(), 10)
    new FaceSnap("3emeComposant", "3emeComposant", new
       Date(), 100)
  ];

    getSnapFaces(): FaceSnap[] {
      return [...this.faceSnaps] // Pour retourner un
         tableau independant meme si il a les memes
         references
  }

  getSnapById(id: string): FaceSnap {
    const trouverSnap = this.faceSnaps.find(FaceSnap =>
       FaceSnap.id === id );
    if (!trouverSnap) {
      throw new Error("No snap found with ID " + id);
    }else{
      return trouverSnap;
    }
  }

  snapFaceById(faceSnapId: string, snapType: SnapType) {
      const trouverSnap = this.getSnapById(faceSnapId);
      trouverSnap.snap(snapType);
    }
}
```

Afin d'utiliser la méthode getSnapById, on va simplement injecté le service dans la classe ou l'on veut l'utiliser, puis l'appeler :

```

  export class SingleFaceSnap implements OnInit{

  constructor(private snapsService: FaceSnapsService) {
  }

  faceSnap!: FaceSnap;

  ngOnInit(): void {
    this.faceSnap = this.snapsService.getSnapById(
       faceSnapId);
```

```
  }

  }
```

5 Navigation et routes :

5.1 Définition des routes :

Dans le fichier routes de l'application, on retrouve un tableau de Routes. On va y insérer toutes les routes de notre application :

```
export const routes: Routes = [
  {path: 'facesnaps', component: FaceSnapList}, // route qui va afficher le contenu de FaceSnapList
  {path: '', component: HubPage}, // route vide, donc le hub
];
```

Ensuite, dans notre app.ts, on va pouvoir importer "RouterOutlet". Grâce à ceci, il sera possible dans le HTML de définir, au lieu d'un composant précis, l'objet qui se trouve à l'emplacement de la route dans l'url :

```
<app-header/>
<router-outlet/> // Le composant trouve a cette route
```

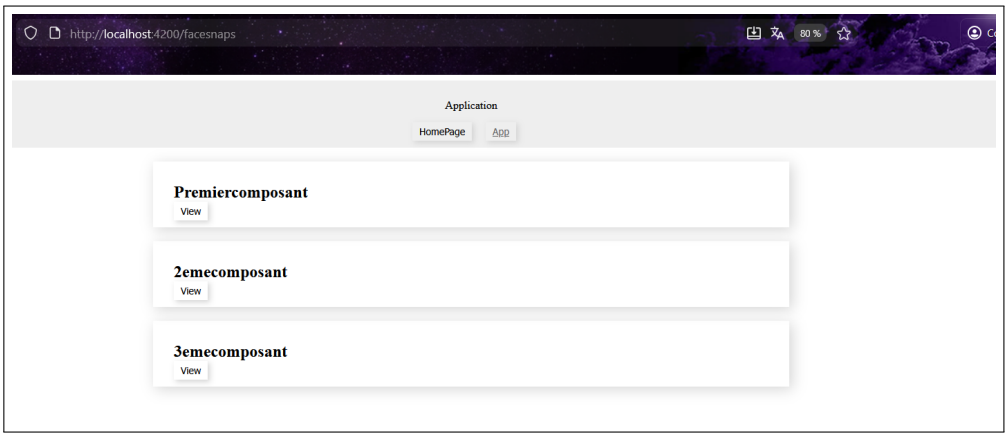
5.2 Naviger entre les composants :

Pour naviguer, il suffit d'utiliser routerLink (qu'on oublie pas d'importer dans le .ts de la où on l'utilise). Par exemple, avec un composant header :

```
<header>
  <p>Application</p>
  <nav>
    <button routerLink="" routerLinkActive="active" [routerLinkActiveOptions]="{exact: true}">HomePage</button>
    <button routerLink="facesnaps" routerLinkActive="active">App</button>
  </nav>
</header>
```

Ici, routerLinkActive sert à donné la classe active à l'élément dont la route est celle selectionnée.

Maintenant, on peut naviguer dans la pages sans problème entre le hub et l'app :



5.3 Accéder à un élément en particulier :

Il est possible dans les routes de définir une variable qui changera :

```
1
2 export const routes: Routes = [
3   // La route avec l'id du snap afficher
4   {path: 'facesnaps/:id', component: SingleFaceSnap},
5   {path: 'facesnaps', component: FaceSnapList},
6   {path: '', component: HubPage},
7 ];
```

Grâce aux méthodes pour récupérer un snap via un id, on peut dans une classe qui reprends nos snap (copié collé de Comp1) faire :

```
1
2 constructor(private snapsService: FaceSnapsService,
3   private route: ActivatedRoute) {
4   }
5
6   faceSnap!: FaceSnap;
7
8   ngOnInit(): void {
9     const faceSnapId = this.route.snapshot.params['id'];
        // Recup le face snap qui correspond a l'id dans
        l'url
10    this.faceSnap = this.snapsService.getSnapById(
        faceSnapId); // On va recup le faceSnap qui
        correspond a cette id pour l'afficher
11  }
```

Il ne manque plus qu'un moyen de mettre l'id du face snap dans l'url, j'utilise un bouton lié par un événement à cette méthode dans Comp1 :

```
1
2  constructor(private router: Router){}
3
4    @Input() faceSnap!: FaceSnap; // Accepte une valeur qui
         vient du parent
5
6    protected onview() {
7     this.router.navigateByUrl('facesnaps/'+this.faceSnap.
         id); // Methode pour inclure l'id choisi dans l'
         url
8    }
```

On peut maintenant choisir quel composant regarder en cliquant sur le bouton view :