**Qu'est-ce qu'Angular ?**

En bref, [Angular](https://angular.io/) est un **framework JavaScript** qui vous permet de développer des applications "efficaces et sophistiquées", comme dit la documentation. Il permet notamment de créer ce qu'on appelle des **Single Page Applications** (ou **SPA**) : des applications entières qui tournent dans une seule page HTML grâce au JavaScript

Installation d’Angular : il faut tout d’abord avoir NodeJS installé, et á partir de Angular CLI taper la commande suivante :

* npm i -g @angular/cli 🡪 i signifie installe, -g d’une manière globale sur la machine.
* ng v 🡪 indique la version d’Angular utilisé.

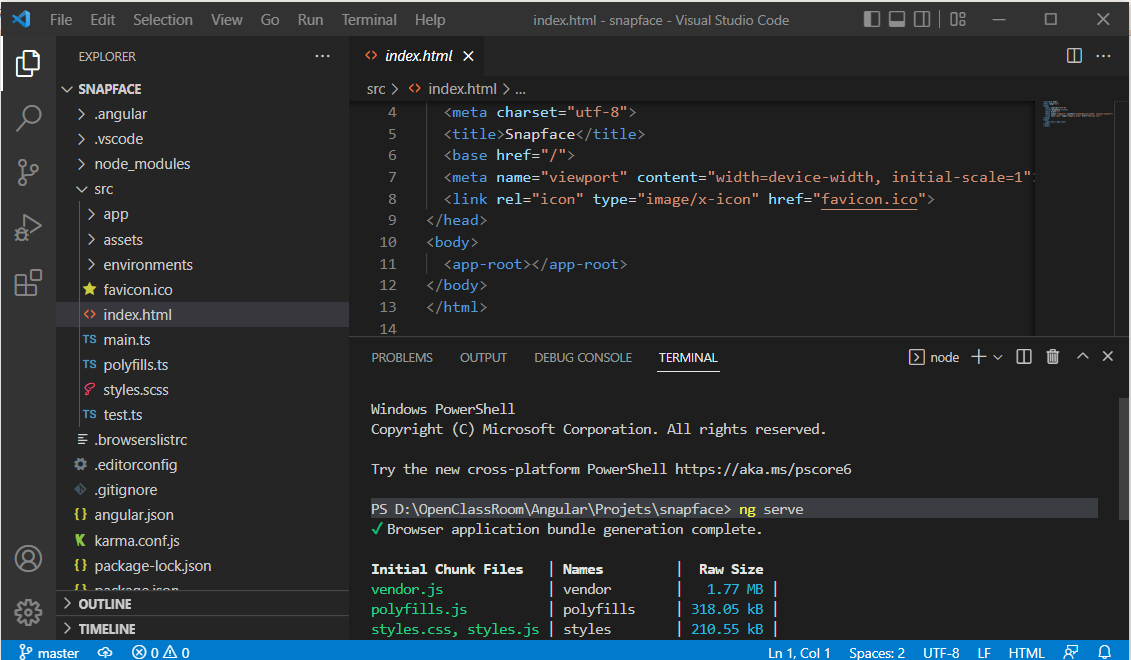
### Créez une nouvelle application

Ouvrir cmd et naviguer vers le dossier dans lequel voulez-vous créer votre 1er projet, et tapez la commande suivante :

* ng new snapface --style=scss --skip-tests=true 🡪 snapface le nom du projet, les SCSS sera le type de styles á appliquer, et les tests d’angular seront ignorés.
* S’il vous demande d’ajouter un module de routing, tapez n.

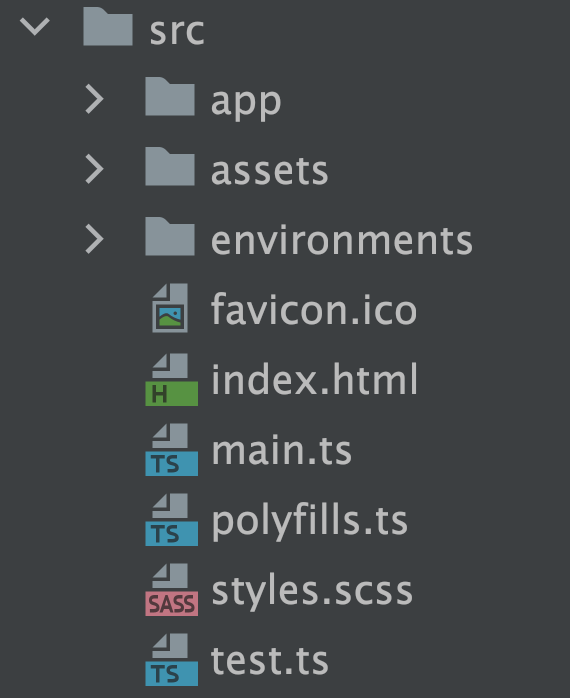
Si la commande ng new ne s’exécute pas, essayer de lancer cmd autant qu’administrateur et tapez la commande suivantes 🡪 Set-ExecutionPolicy RemoteSigned

Si tous se passe bien ouvrir le dossier d’application snapface avec votre IDE préféré (Visual Studio Code) et lancer le serveur Angular avec la commande ng serve.



L’application est lancée sur l’URI : <http://localhost:4200/>

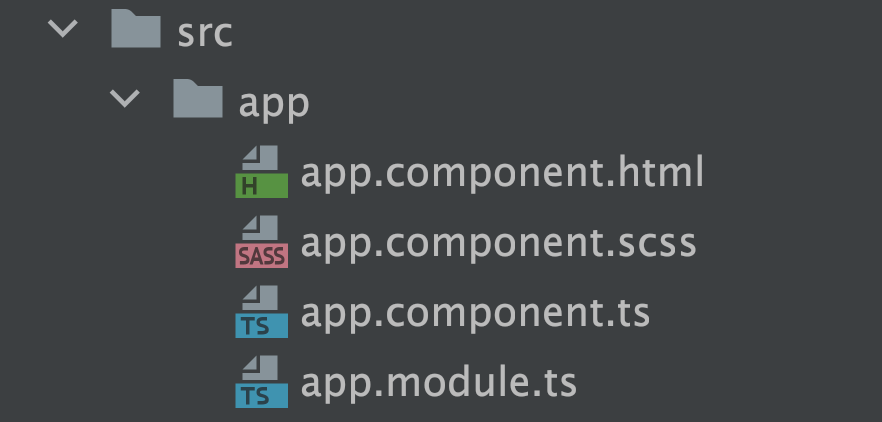
Une application Angular contient beaucoup de fichiers qui ne nous intéresseront pas. Vous allez majoritairement travailler dans le dossier src :



index.html  est le fichier principal de votre projet et si vous regardez dedans le  <body>  ne contient qu'une balise  <app-root>  vide. D'où vient tout ce que l'on voit dans le navigateur, dans ce cas ?

 La balise <app-root> correspond à la racine de votre application, ou plus précisément à l'App Component, le **component** racine de votre application. Angular sait qu'il doit remplacer cette balise par le fameux AppComponent, donc allons voir où il se trouve ?

Dans le sous-dossier app :

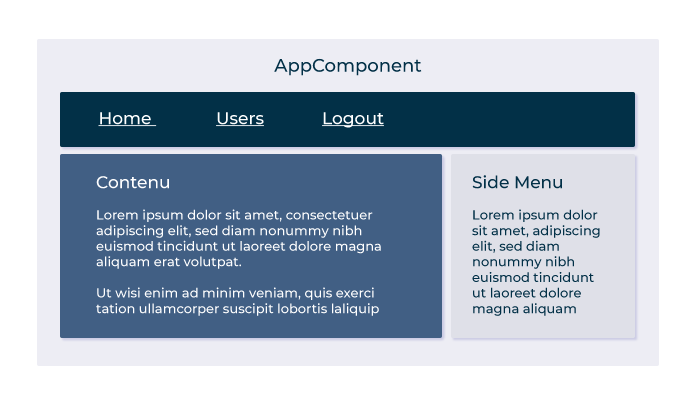


À part le fichier  app.module.ts  auquel on s'intéressera plus tard, vous avez trois fichiers correspondant à AppComponent : un fichier HTML, un fichier SCSS, et un fichier TS.

Si vous regardez dans le fichier  app.component.html  , vous y trouverez le contenu que vous voyez dans votre navigateur ! Mais comment Angular sait-il qu'il faut remplacer  <app-root>  par ce contenu ? Le secret se trouve dans  app.component.ts  :

### Qu'est-ce qu'un component ?

Les components sont les **composants de base** d'une application Angular. Une application Angular peut être vue comme une **arborescence** de components avec AppComponent comme component racine. On pourrait imaginer une application simple qui ressemblerait à ça :



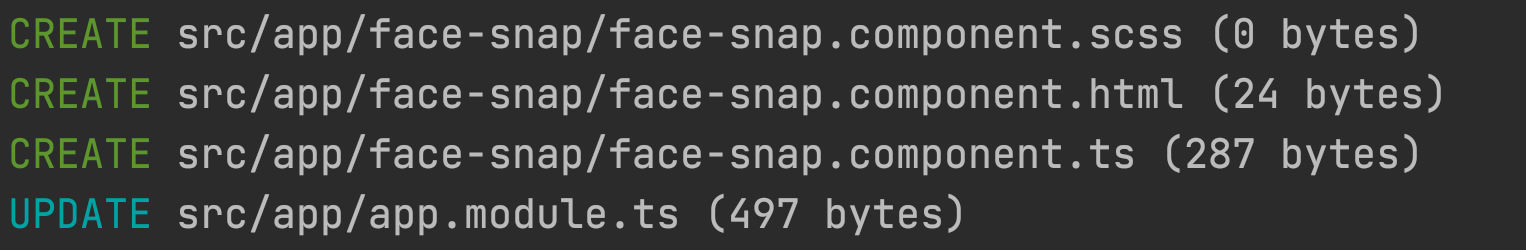
Ici, une structure possible serait d'avoir un component pour le menu en en-tête, un autre pour le contenu principal, et un dernier pour le menu latéral. On aurait donc HeaderMenuComponent, MainContentComponent et SideMenuComponent.

### Créez un component

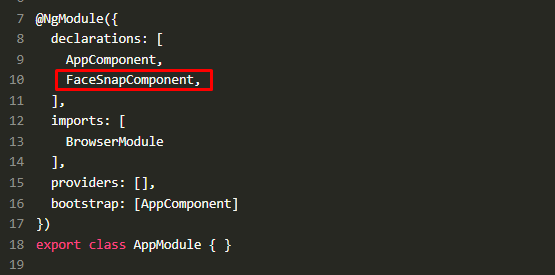
Pour crée un component tapez la commande suivante :

* ng generate component face-snap 🡪 face-snape est le nom du component crée.

Cette commande génère un dossier dans le src/app qui porte le nom du composant (ici c’est face-snap), et qui contient 3 fichiers :



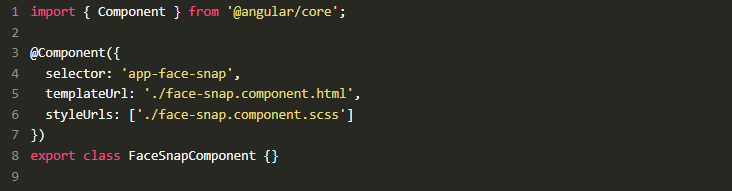
Sachez que pour utiliser un component dans une application Angular, il faut le **déclarer** dans un **module** – AppModule étant le module principal de l'application, Tous les components que vous créerez dans ce cours seront déclarés ici. Src/app/app.module.ts



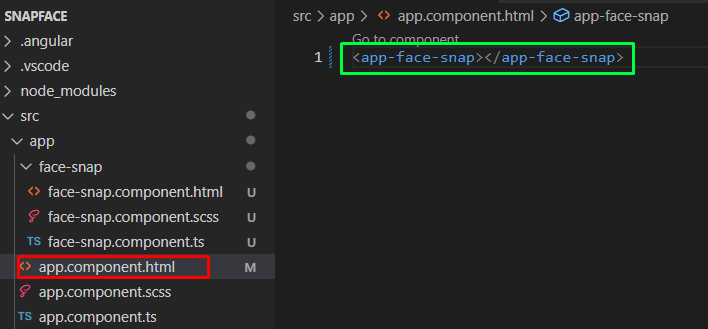
Fichier HTML du composant (Dans ce cas : face-snap.component.html) : c’est le fichier qui va contenir les balise html du composant.

Fichier SCSS du composant (Dans ce cas : face-snap.component.scss) : c’est le fichier qui va contenir le style css á appliquer sur le composant.

Fichier TypeSript du composant (Dans ce cas : face-snap.component.ts) : fichier TypeSript du composant, il contient une classe portant le nom du composant, et qui est déclarée avec un décorateur  @Component  à qui on passe un objet de configuration avec un **sélecteur**, un fichier de **template** et un fichier de **styles**.



Sélecteur (ici c’est app-face-snap) : sera utilisé comme balise html dans le app.component.html afin que le composant soit affiché dans l’application, car Comme vous le savez, votre application est une arborescence de components avec AppComponent comme racine :

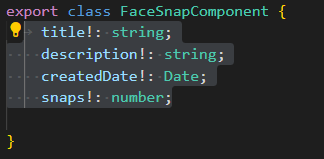


Si on répète le <app-face-snap></app-face-snap> plusieurs fois dans le app.component.html , alors il sera affiché plusieurs fois.

Afficher des données dynamiques dans un composant :

Alors, pour cela il faut déclarer les donnés dans le fichier TypeScript du composant, exactement dans la classe du composant.

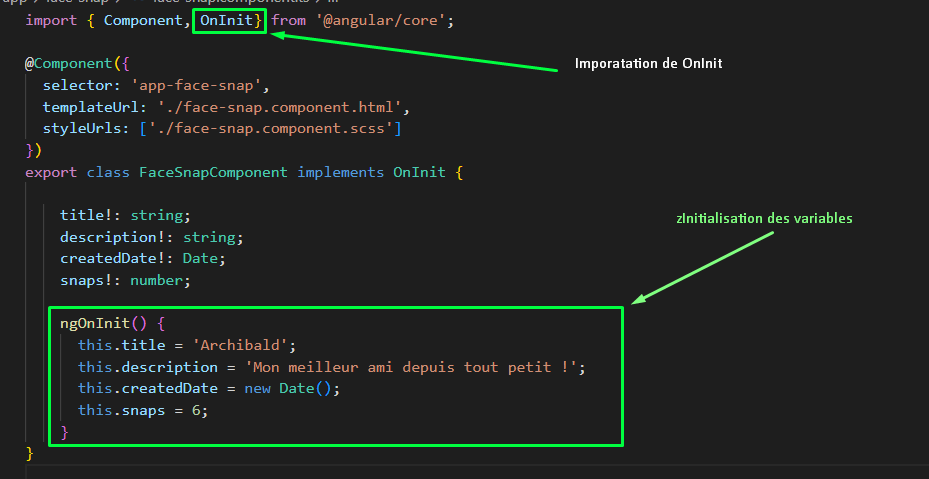
Alors pour ignorer l’initialisation des variables au moment d’initialisation, on ajoute le ! á la variable.



Dans JavaScript il y’a 3 type de variable primitive string, number, boolean.

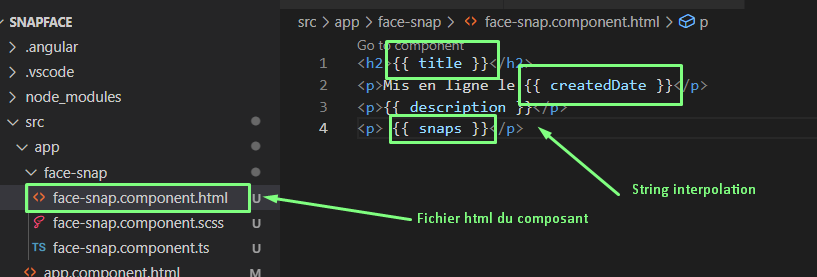
L’initialisation des variables se fait dans la méthode ngOnInit (), qui est une méthode appelé une seul fois par instance, au moment de sa création.

Pour pouvoir ajouter la méthode ngOnInit() il faut importer OnInit qui est une interface que la classe du composant doit implémenter.

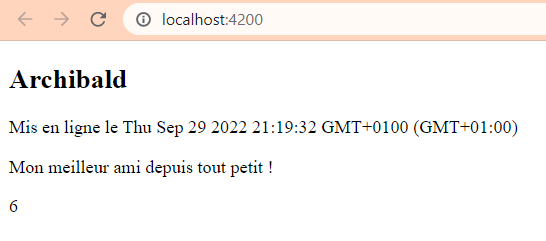


Affichez du contenu venant du TypeScript dans le composant avec la string interpolation.

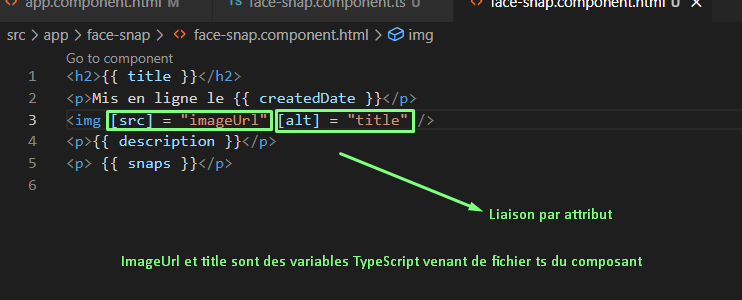
Dans le fichier html du composant vous pouvez afficher le propriétés (les données) du composants avec la string interpolation



Et comme <app-face-snape></app-face-snap> déjà dans le composant racine (app.component.html) le serveur va répondre avec le résultat suivant :



Afficher des données avec l’attribut binding : on l’utilise généralement pour affecter une valeur á un attribut d’un élément HTML.



Dans l’attribut binding , l’attribut est délimité par des crochets [], et la valeur c’est une variable TypeScript.

## Réagissez aux événements : Prenons le cas de l’évènement CLICK, On va ajouter un bouton qui rajoute le nombre de snaps, donc ce buton tout d’abord doit être relié á une méthode d’ajoute de snap qu’on doit implémenter dans le fichier TypeSript du composant (face-snap.component.ts) :



onAddSnap() sera la méthode a appelé lors de l’évènement CLICK.

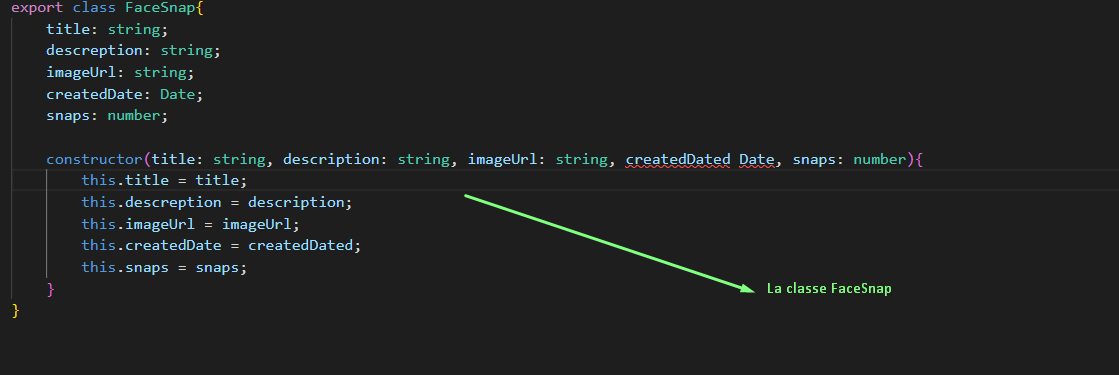


Dans l’élément buton l’évènement sera délimité par () et sa valeur sera la méthode onAddSnap() du fichier TypeSript du composant.

**Models Package :**

Le package models sous Angular est créer dans le src/app/models.

Donc au lieu de mettre beaucoup d’attributs dans le fichier ts du composant, c’est mieux de les regroupées dans une seul classe qu’on va finalement agréger dans le composant ts.



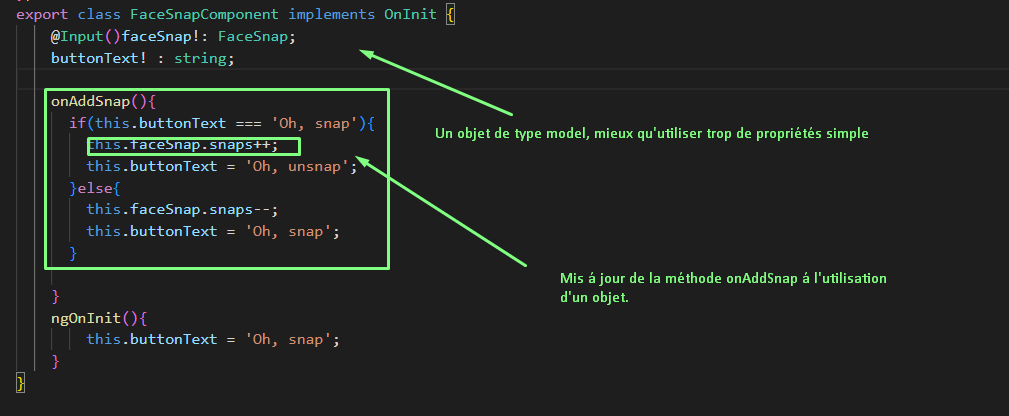
La classe FaceSnap c’ est notre 1ère model ,Donc elle est crée dans le dossier src/app/models.



**Ajoutez des propriétés personnalisées :**

Angular vous permet d'ajouter des **propriétés personnalisées** à vos components de manière à ce que vous puissiez les injecter **depuis le component parent**. Ce que ça vous permettrait de faire avec  FaceSnapComponent, ce serait de faire en sorte que  AppComponent  centralise les données de plusieurs FaceSnaps, génère une instance du component pour chaque FaceSnap, et injecte ce FaceSnap à cette instance.

Alors pour qu'une propriété puisse être injectée **depuis l'extérieur** d'un component, il faut lui ajouter le décorateur @Input(). Créez maintenant une propriété  faceSnap  de type  FaceSnap  (votre nouveau type !) et mettez-lui ce décorateur :

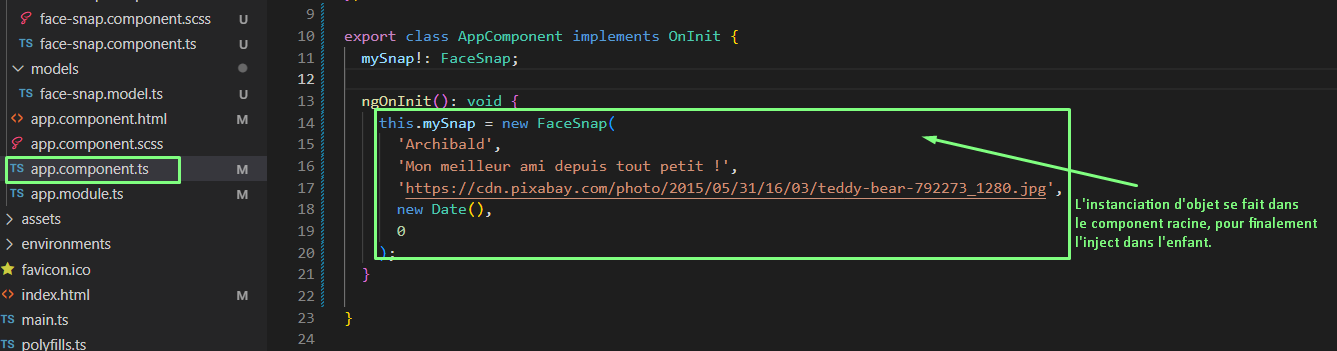


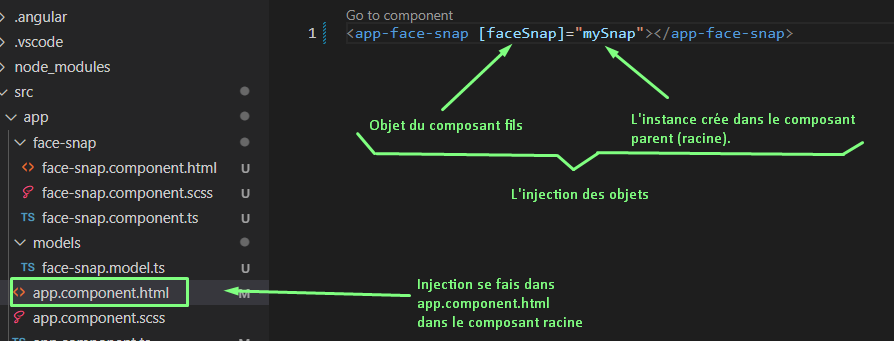
Le décorateur @Input() permet au composant de dire á l’exterieur que j’attend l’initialisation de cette variable par quelqu’un.

Et aussi une mis á jour du face-snap.component.html :



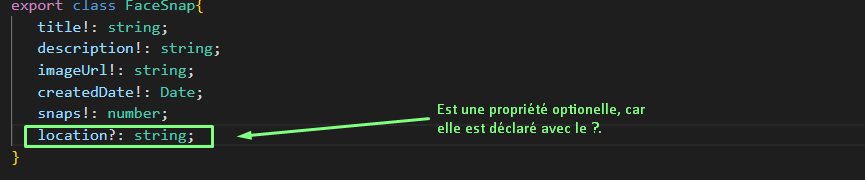
Donc l’instanciation de l’objet faceSnap sera faite dans le appComponent exactement dans app.component.ts et l’injection se fait dans le fichier app.component.html.



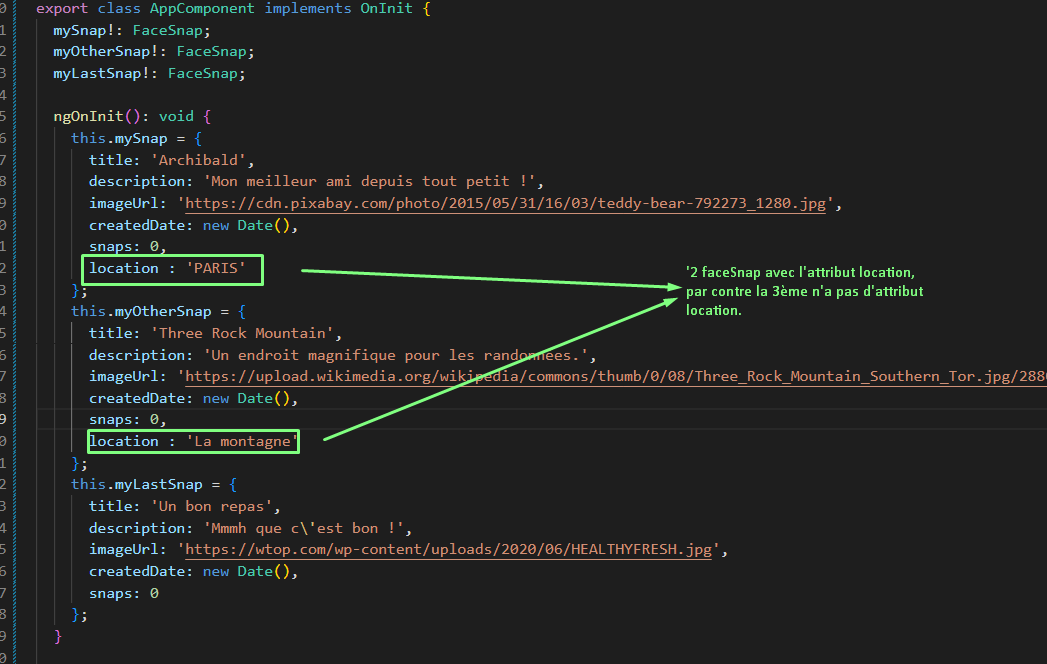


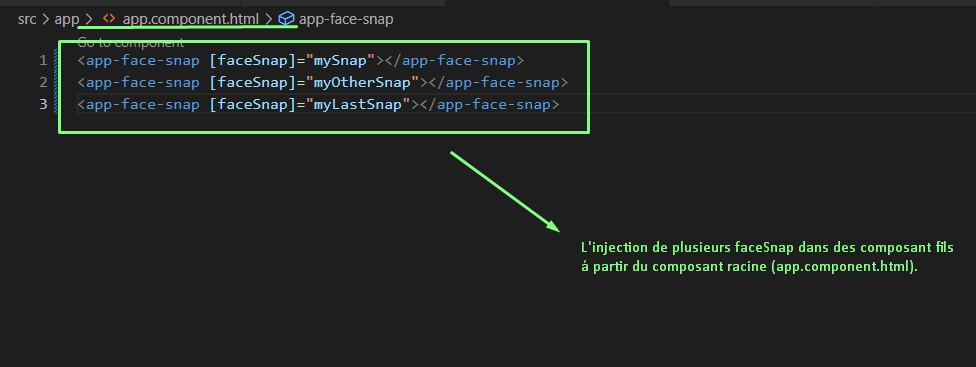
**Directive**est une classe qui vient **ajouter du comportement** à l'élément sur lequel elle est posée. L'astérisque au début du nom  \*ngIf  nous montre qu'il s'agit d'une directive **structurelle**, qui viendra donc toucher à la structure du document.

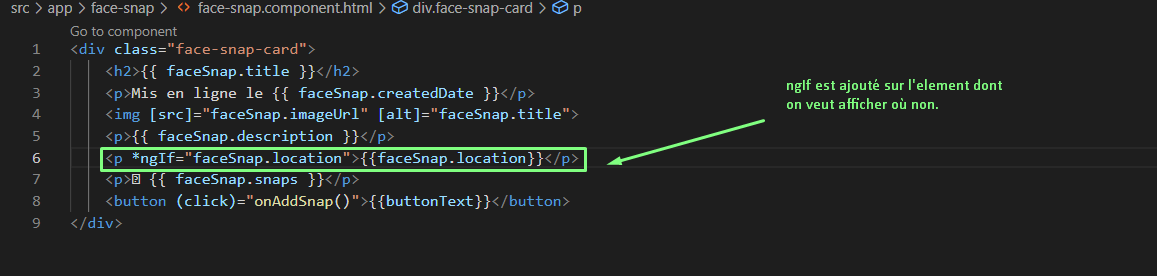
Alors pour bien comprendre \*ngIf nous avons besoins d’un propriété optionnelle dans la classe FaceSnap :



Et dans le appComponent ts on va instancier plusieurs faceSnap (mySnap, myOtherSnap, et myOtherSnap) dont chacune possède un attribut location.



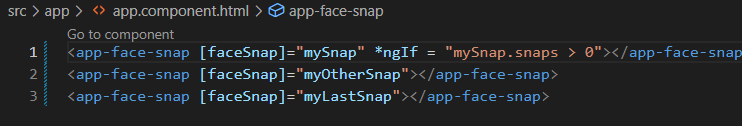


Affichage de l’attribut optionnelle avec \*ngIf : 

Donc avec le code ci dessus on affiche la propriété location que pour les composants qui la contient, ngIf prend comme valeur une condition, dans notre cas, faceSnap.snaps est la conditions, et á son absence la condition est fausse par conséqent l’element <p> sera pas ajoutér au DOM, ducoup sera pas afficher.

Undefined = 0 = false = chaine vide, en JavaScript.

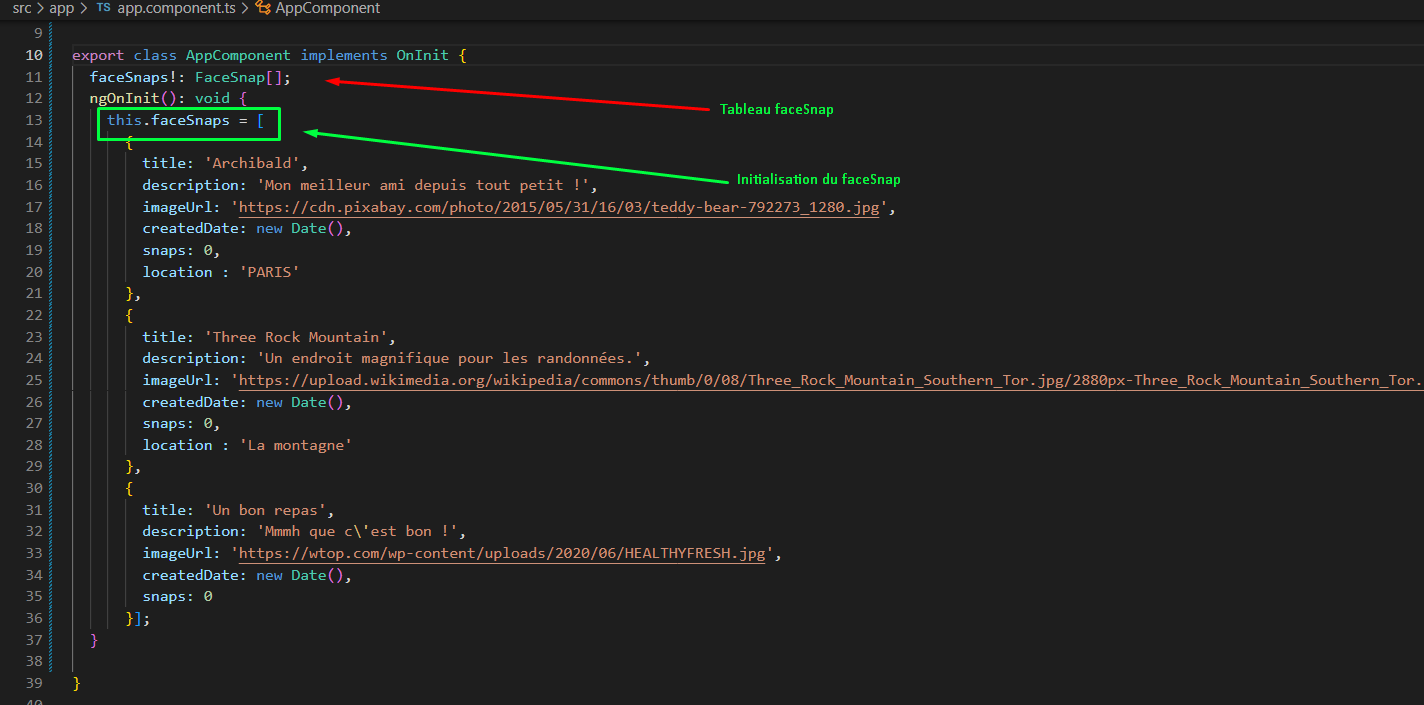
On peut utilisé \*ngIf directment sur les composant, par exemple afficher que les facesnaps qui ont le nombre de snaps supérieur á 0 ;

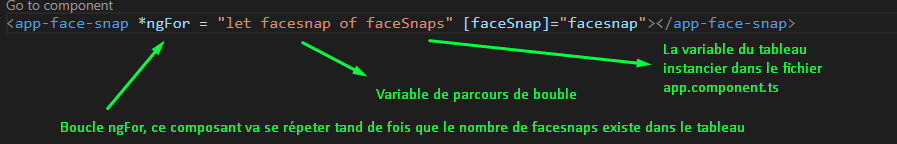


Ici le 1èr facesnap sera pas afficher car son attribut snaps égal á 0.

**Afficher un tableau de facesnaps avec \*ngFor**

Alors dans app.component.ts au lieu de créer plusieurs variables, c’est mieux á la place de créer un tableau et de l’afficher ensuite avec \*ngFor.





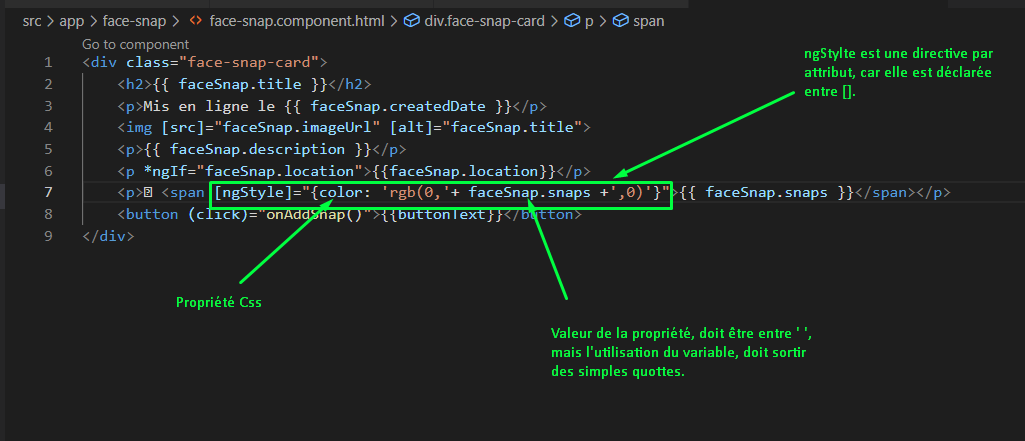
L’affichage est devenu maintenant plus simple par l’utilisation de \*ngFor, donc á chaque fois en prends la variable de parcours ici c’est facesnap, et on l’inject dans le composant, ce qui nous donnes plusieur composant dynamique.

Les styles en Angular :

Angular limite les styles d'un component au component même : les enfants d'un component n'héritent pas des styles de leur parent. Vous allez donc avoir des styles et des classes qui sont encapsulés. La seule exception est le fichier de styles principal styles.scss qui sert justement à déclarer des styles et des classes globaux.

La directive [**ngStyle]** prendra comme argument un objet où :

* Les **clés**sont les styles CSS à modifier ;
* Les **valeurs**sont les valeurs que doivent prendre ces styles.



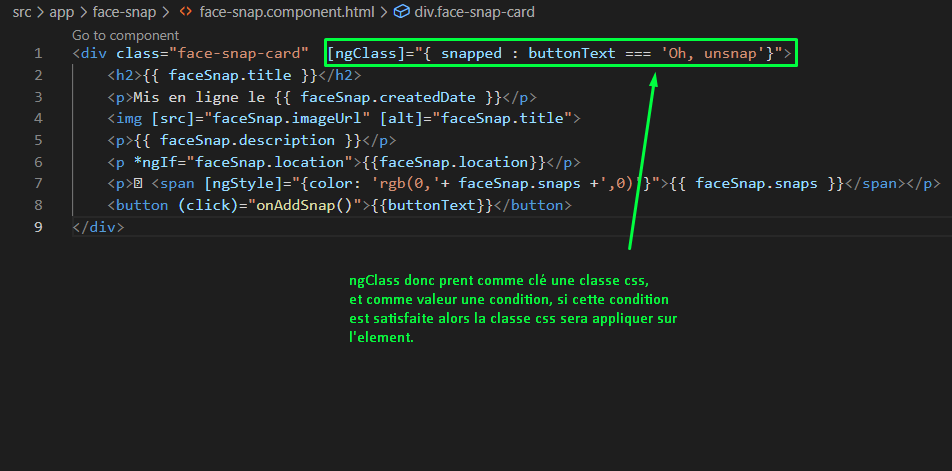
Dans Example ci-dessus le nombre de snaps sera écrit en vert en fonction du nombre des snaps.

La directive [ngClass] : permet d’appliquer une classe css sur un element html.

[ngClass] prend un objet en argument, comme  [ngStyle], mais dont le fonctionnement est différent :

* Les **clés** sont les **noms de classe CSS** à appliquer ;
* Les **valeurs** sont les **conditions** qui doivent être remplies pour que les classes correspondantes s'appliquent.

En gros :<div [ngClass]="{ 'class-name': condition }"></div>



maintenant nous avons ajoutés la fonctionnalité où la nouvelle classe  snapped  s'applique uniquement si l'utilisateur a cliqué sur le bouton "Oh Snap!".

Les **pipes**: Un pipe est appliqué dans le HTML et va **formater** la valeur qu'on lui passe selon le pipe utilisé **sans toucher à la donnée sous-jacente**.

Il existe trois pipes fournis par Angular pour modifier la casse :

* **LowerCasePipe** : on affiche le texte en **minuscules**
* **UpperCasePipe** : on affiche le texte en **majuscules**
* **TitleCasePipe** : on affiche le texte avec une majuscule au début de chaque mot, avec le reste du mot en minuscule.

On applique un pipe à une chaîne de caractères affichée avec la string interpolation. On rajoute le caractère pipe  |  puis le nom du pipe. Par exemple, si vous voulez afficher le titre des FaceSnap en majuscules, vous appliquez UpperCasePipe :

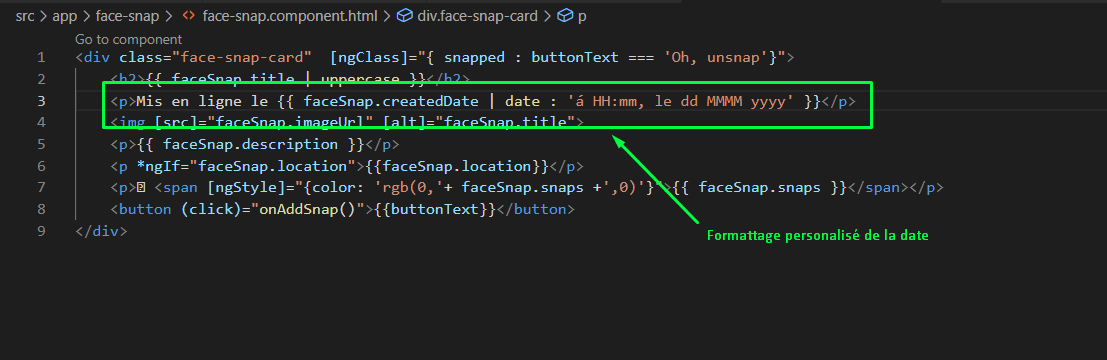
<h2>{{ faceSnap.title | uppercase }}</h2>

De même pour LowerCasePipe et TitleCasePipe :



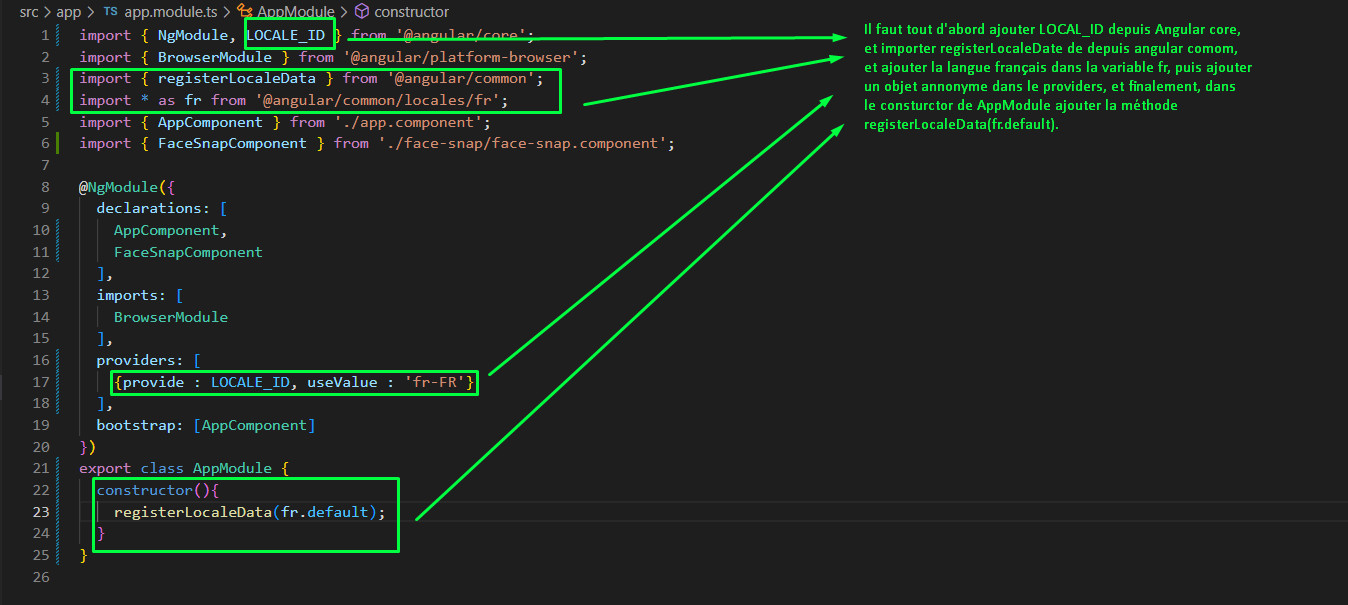
Formattez les dates :

* DatePipe permet de formater les dates, et sans configuration fournit un formatage par défaut.
* DatePipe fournit des configurations prédéfinies avec des noms comme  short,  longDate  ou  mediumTime
* DatePipe permet également de personnaliser totalement le format d'affichage des dates avec des chaînes de caractères qui encodent le format souhaité, par exemple  'à HH:mm, le d MMMM yyyy'



Changement de la langue par défaut de Angular :

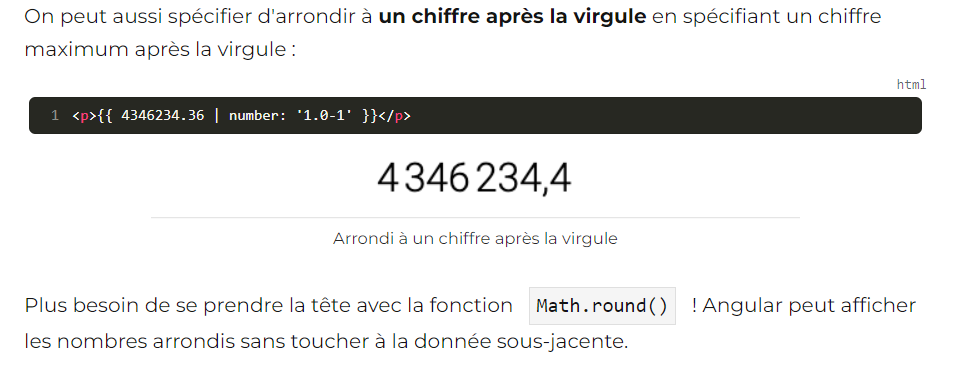
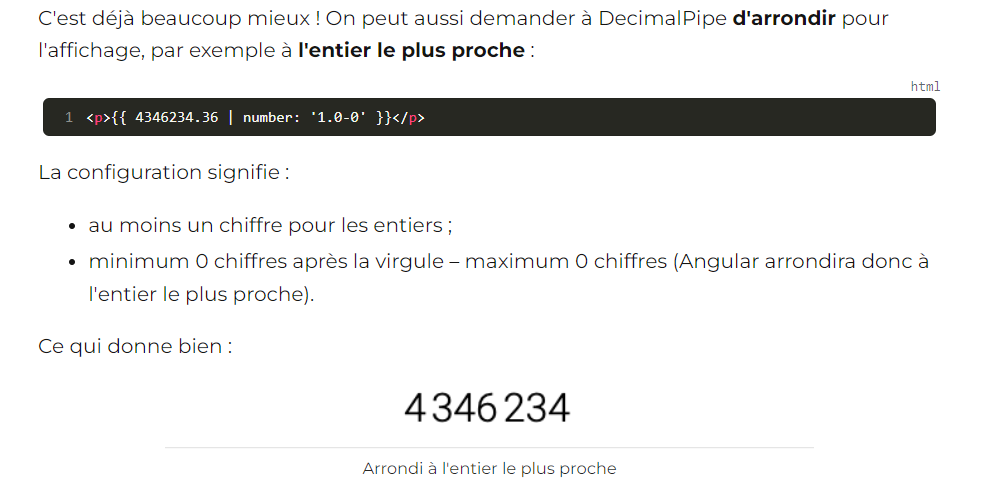
La langue par défaut d’Angular est l’anglais, ce qui fait que la date aussi sera afficher en anglais dans l’application, alors pour la changer il faut modifier le fichier app.module.ts :

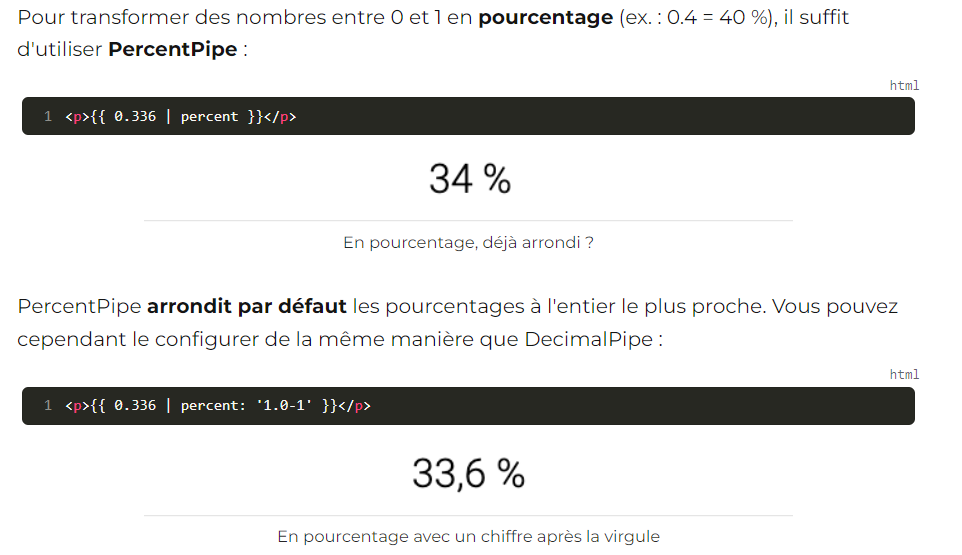


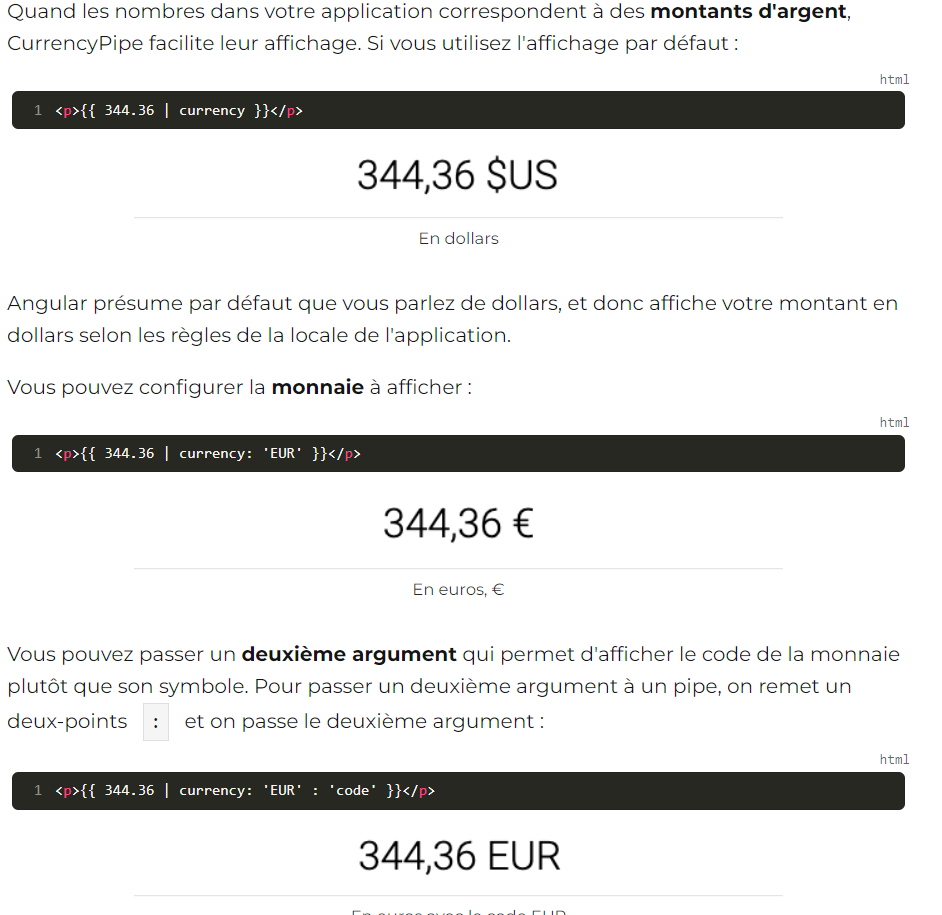
Formattez les chiffres :

Les pipes qui concernent le formatage des **nombres**, Il y en a trois :

* **DecimalPipe** – facilite l'affichage de nombres avec des **chiffres après la virgule** (qui met une virgule plutôt qu'un point, par exemple).
* **PercentPipe** – formate les chiffres en **pourcentage**.
* **CurrencyPipe** – permet d'afficher des nombres sous forme de **monnaie** très facilement.







**Partager vos données avec les Services :**

Dans une application complète Angular, vous aurez des dizaines, voire des centaines de components différents. Ces components auront très souvent besoin d'accéder aux **mêmes données**. Il y aura des **interactions** courantes avec ces données, comme l'écriture, la modification et la suppression. L'application peut également avoir besoin d'interagir avec un **serveur** concernant ces données, pour la lecture et l'enregistrement.

Pour mieux **organiser** votre code, pour **éviter de répéter** des blocs et pour toujours avoir des données **à jour** partout dans l'application, ce serait intéressant de pouvoir regrouper les données et leurs méthodes ensemble, de les **centraliser**. En Angular, cette centralisation se fait dans des **services**.

Juste avant de vous lancer dans la création de votre premier service, il faudra restructurer légèrement votre application.

Pour l'instant, AppComponent s'occupe de générer les FaceSnapComponents. Mais ce n'est pas un bon choix, surtout si vous voulez créer une structure plus complexe par la suite. Vous allez créer un nouveau component, **FaceSnapListComponent**, qui sera le parent de tous vos FaceSnapComponents.

Générez le nouveau component avec la commande :

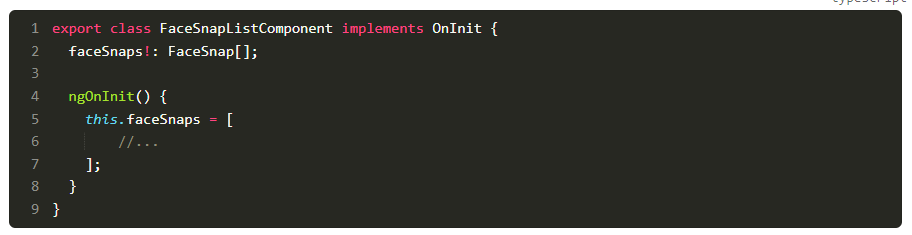
ng generate component face-snap-list

Il existe une version raccourcie pour cette commande : vous pouvez taper simplement :

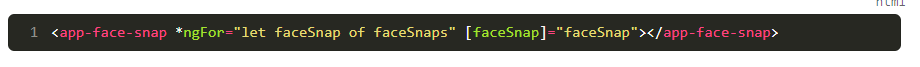
ng g c face-snap-list

Ce nouveau component sera le parent des FaceSnapComponent : il a donc besoin du tableau qui se trouve actuellement dans AppComponent. Il faut aussi lui transplanter le template d'AppComponent.

face-snap-list.component.ts  doit contenir :



Et  face-snap-list.component.html  :



La dernière étape de cette refactorisation est d'ajouter FaceSnapListComponent au template de AppComponent :



Visuellement, votre application n'a pas changé, mais sa structure a été améliorée :

La logique qui gère la liste des FaceSnaps se trouve maintenant dans un component prévu pour ;et AppComponent ne contient que ce qu'il faut pour afficher les éléments principaux de l'application.

Dans une application Angular, AppComponent ne sert généralement qu'aux **éléments structuraux principaux**. Tout ce qui est plus "spécialisé" va dans un component à part.

Ajoutez tout de suite un nouvel élément principal, HeaderComponent :

ng g c header

Pour afficher le header au-dessus de la liste de FaceSnap, il suffit de l'ajouter dans le template de AppComponent :



Je vous propose un header très simple pour l'instant :

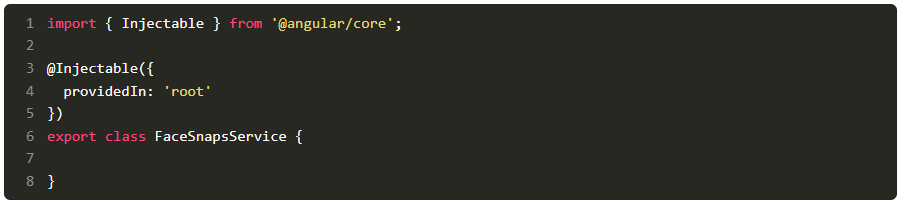
Son template :



**L’ajoute d’un module service**:

Vous allez maintenant créer un service qui va centraliser toutes les interactions avec les FaceSnaps, afin que tous les éventuels components de votre application dépendent des mêmes données.

Dans le dossier  app, créez un dossier  services  et dedans un fichier  face-snaps.service.ts.



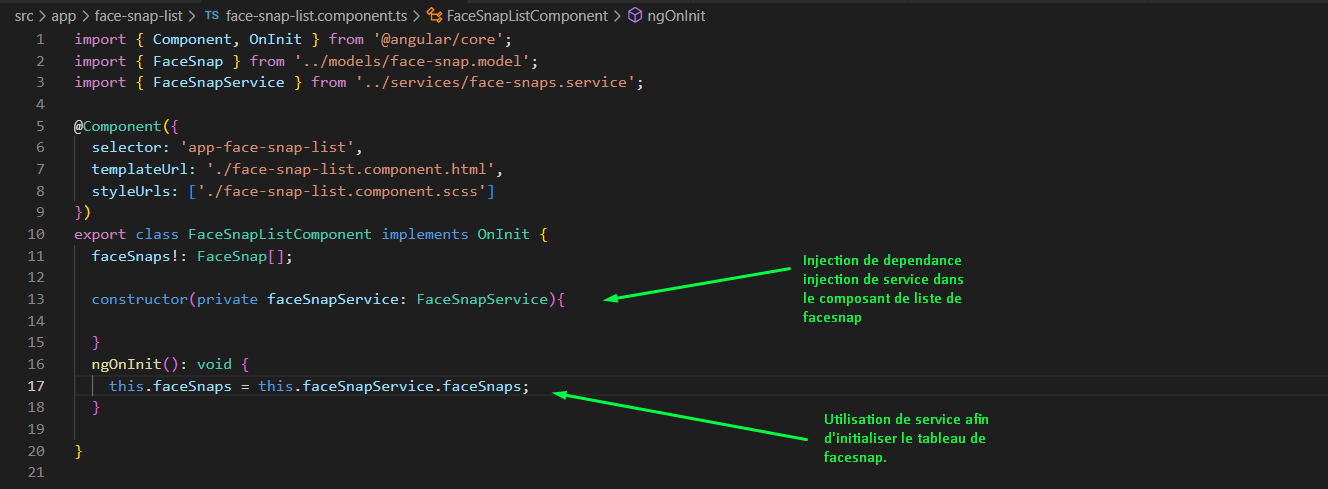
L'objet de configuration qui spécifie providedIn: 'root' dit à Angular d'enregistrer ce service à la racine de l'application. Ce sera très souvent le cas pour vos services, car ça permet de s'assurer de n'avoir qu'une seule instance du service, partagée par tous les partis intéressés.

Le premier élément que vous allez déporter dans votre service est le tableau des FaceSnaps. Il est important de savoir qu'un service n'a **pas de méthode  ngOnInit()**, car les services ne sont pas instanciés de la même manière que les components. Il faudra donc déclarer et initialiser le tableau dans la même expression :



Injecter votre service :

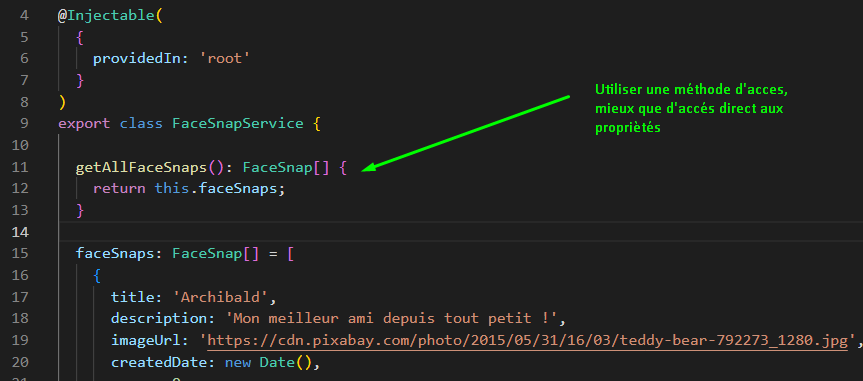
Pour pouvoir utiliser un service dans un component, il faut utiliser le système d'**injection de dépendances** que vous fournit Angular. C'est très simple : vous passez un argument du type du service au constructor du component, et Angular vous mettra à disposition la bonne instance du service. Concrètement pour votre application, dans FaceSnapListComponent :



Pour rappel, ajouter un **modificateur d'accès** comme  public  ou  private  à un argument du  constructor  crée une propriété avec ce nom-là dans la classe. Vous aurez donc accès au service via la propriété  faceSnapsService.

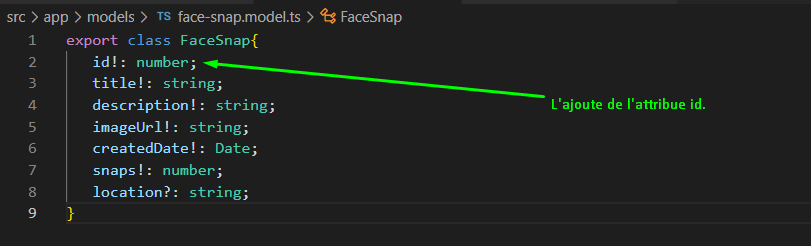
Centraliser les interaction :

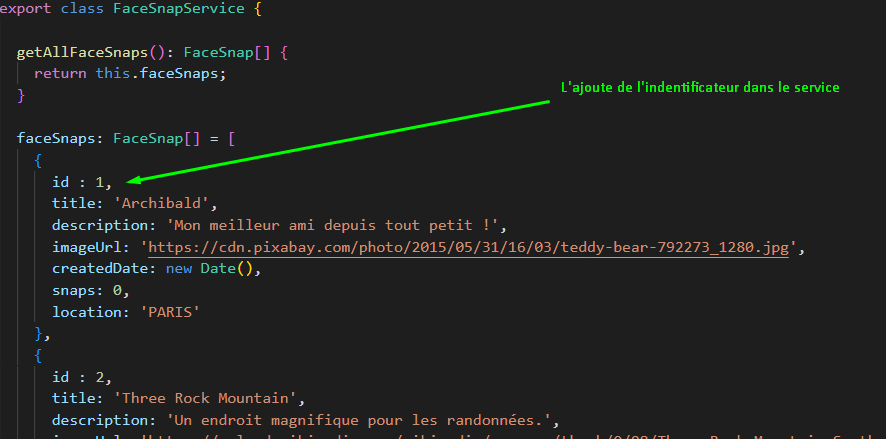
La première méthode que vous allez créer est  getAllFaceSnaps(). Cette méthode retournera, comme son nom l'indique, tous les FaceSnaps contenus dans le service, car il peut que certains faceSnape vient de d’autre source de données.



Mettre la fonctionnalité d’ajoute de snap dans le sevice.

Afin d’identifier le faceSnap qu’on veut augmenter son nombre de snaps, il faut ajouter un identificateur á la classe FaceSnap du model.

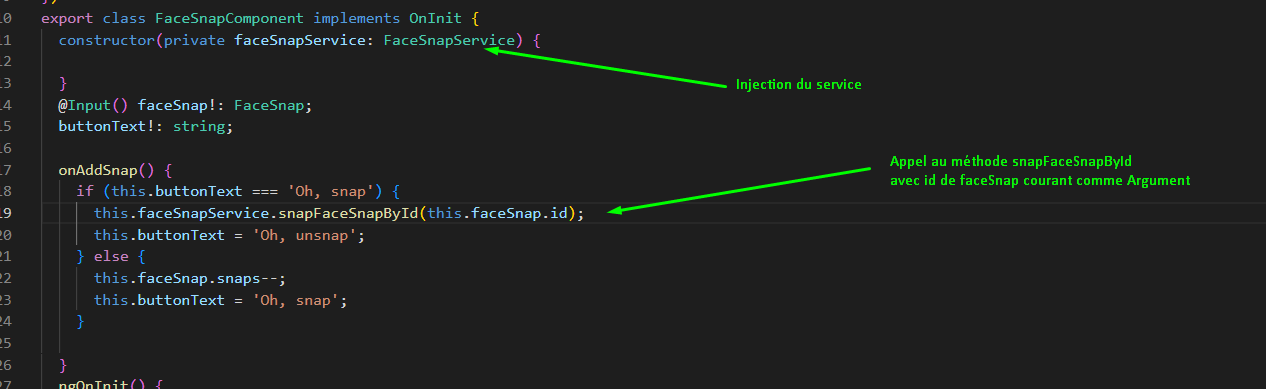




Maintenant on dispose d’identificateur pour un faceSnap, par conséquent on peut implémenter la méthode snapFaceSnapById dans la classe service :

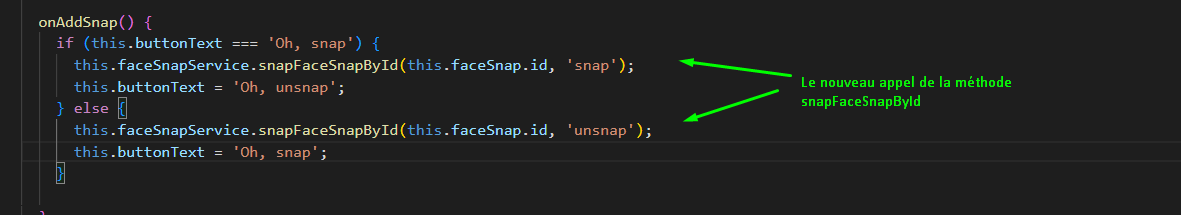


Le snap se fait á l’intérieur du composant faceSnapComponent, donc dedans on doit injecter le service afin de profiter de la fonctionnalité snapFaceSnapById :



Mettre á jour de la fonctionnalité snapFaceSnapById pour qu’il diminue le nombre de snaps.





**Passez en SPA avec le routing :**

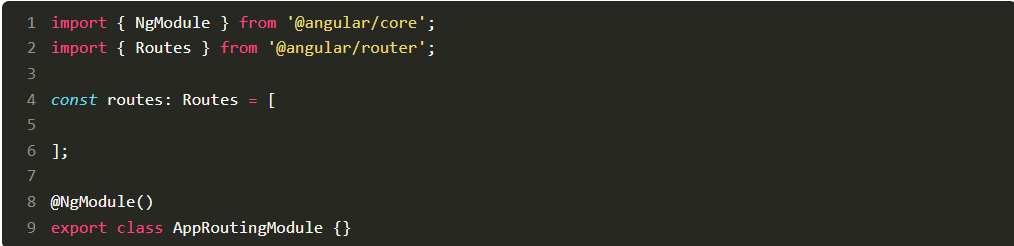
Les **Single Page Applications** (ou **SPA**) sont de plus en plus courantes dans le monde du développement web moderne. Elles permettent une **performance** sans égale en enlevant le besoin d'une application de demander, recevoir, puis afficher une nouvelle page HTML à chaque changement d'URL. Angular vous permet avec le **routing** de créer ce genre d'application – à chaque **URL** correspondra un **component**, et Angular remplacera le component actif sans émettre de requête au serveur.

Ajouter un module de routing :

D'abord, vous allez ajouter un module de routing à votre application. Dans le dossier  app, créez un fichier  app-routing.module.ts. Commencez par y déclarer une classe avec le décorateur  @NgModule()  :



Au-dessus de la déclaration de classe, vous allez initialiser une constante appelée  routes, de type  Routes  (importé depuis  @angular/router) et qui contiendra un tableau :



Ce tableau va lier les **routes** de votre application (les différentes URL) aux **components** correspondants. Créez dès maintenant une première route  facesnaps  qui affichera le component FaceSnapListComponent :



Pour enregistrer ces routes dans votre application, il faut les passer au routeur en passant un objet de configuration au décorateur  @NgModule()  de AppRoutingModule :



Ici, vous dites à Angular que les routes de ce fichier seront les routes à la **racine** de votre application, et vous réexportez le routeur configuré. Pour l'utiliser, il faut ajouter votre module de routing aux **imports** de AppModule, le module principal de votre application :



Il ne reste plus qu'à dire à Angular **où** il faut afficher le component dicté par la route. Dans le template d'AppComponent, remplacez la balise  <app-face-snap-list>  par une balise  <router-outlet>  :



Comme ça, ce sera le **routeur** qui décidera quel component doit être affiché, et **AppComponent** qui décidera à quel niveau ce component sera ajouté. Dans ce cas, on a décidé que le AppHeader restera présent peu importe la route active, ce qui est plutôt réaliste : on pourrait même envisager d'y placer un menu de navigation pour l'application.

Dès maintenant vous allez à <http://localhost:4200/facesnaps>, vous retrouverez l'affichage précédent.

Le but maintenant sera de créer une landing page pour la route "vide", qui permette d'accéder à la route  facesnaps.

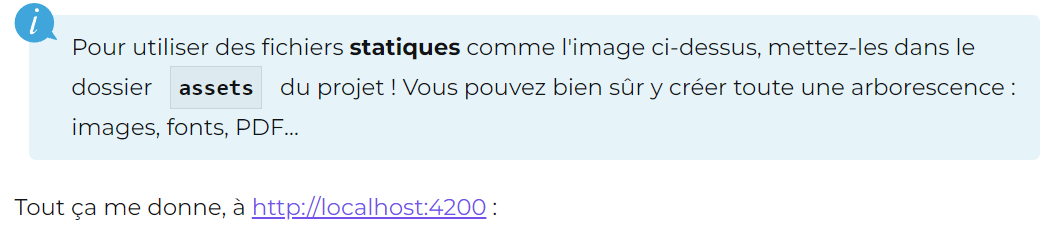
Pour créer LandingPageComponent, vous pouvez utiliser le CLI :



Pour l'enregistrer comme étant le component à afficher pour la route vide, il faut ajouter une route à votre tableau  routes  dans AppRoutingModule :







Nous avons maintenant deux routes fonctionnelles.

**Passez d’une route á l’autre :**

Il reste à faire de permettre à vos utilisateurs de **changer de route**!

### Créez des liens avec  routerLink

Le routeur d'Angular vous met à disposition une **directive** pour créer ultra facilement des liens vers des routes de vos applications, et cette directive s'appelle  routerLink, Elle est très facile à utiliser : on lui passe simplement la **chaîne de caractères** qui correspond à la **route** vers laquelle on veut créer le lien. Sur votre landing page :



Dans ce cas, on n'a même pas besoin de crochets ! Les crochets sont nécessaires à partir du moment où vous passez autre chose qu'une chaîne de caractères.

Maintenant, quand vous cliquez sur le lien de votre landing page, vous êtes redirigé **immédiatement** vers FaceSnapListComponent, sans chargement depuis le serveur.

Allons un peu plus loin encore ! Dans HeaderComponent, ajoutez un **menu de navigation**:



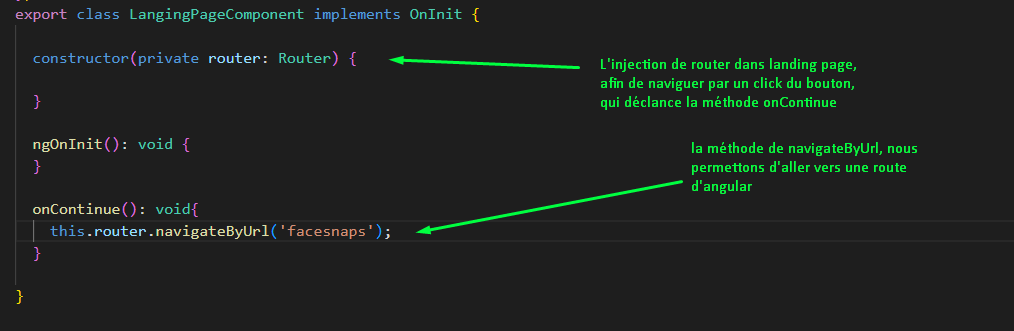
### Naviguez avec le Router

Il existe une autre approche pour changer de route dans une application Angular. Vous pouvez injecter le **Router** dans vos components et changer de route **programmatiquement**.

Les prochaines étapes sont :

* Transformer le lien sur la landing page en bouton ;
* Lier une méthode avec un nom approprié à ce bouton ;
* BONUS : injecter le Router dans LandingPageComponent – il s'importe depuis  @angular/router  et s'injecte comme un service.

On peut commencer par la **méthode** qui sera liée au bouton. J'ai choisi de l'appeler  onContinue(), donc dans landing-page.component.ts :



Et dans landing-page.component.html on transforme le lien en un bouton et on lance l’ecouter sur l’évenment click :

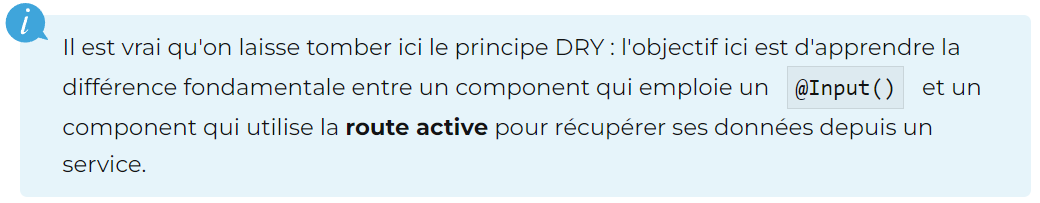


Comme ça nous avons deux méthode de navigation de route d’angular.

## Activez les routes avec ActivatedRoute

Route dynamique : c'est de créer une route avec un **paramètre**  id  pour visualiser un seul FaceSnap. Le component que vous allez associer à cette route pourra **récupérer** cet  id  afin de demander à FaceSnapsService de lui envoyer le FaceSnap correspondant.

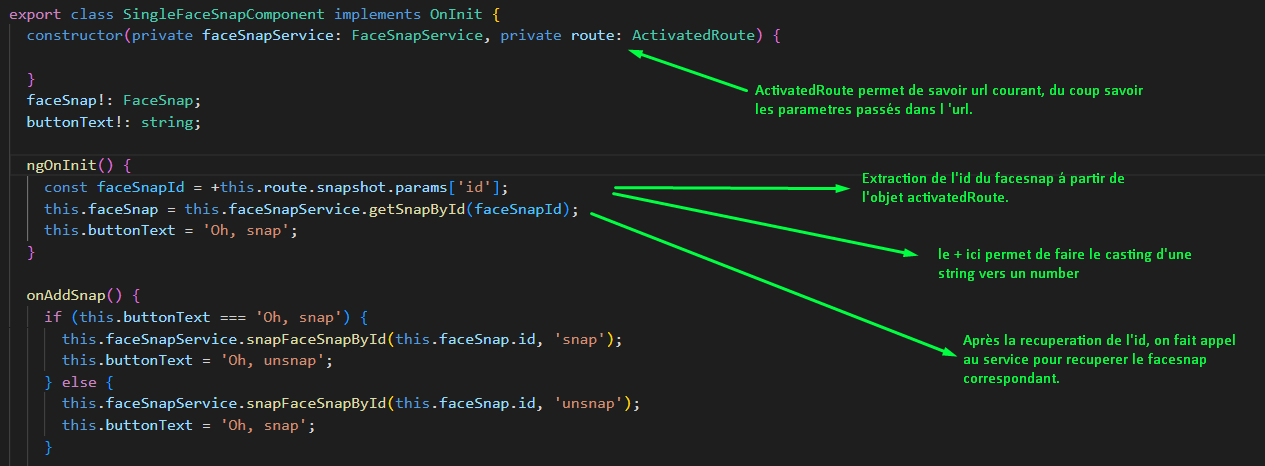
Générez un **nouveau component** appelé SingleFaceSnapComponent, et transposez-y tout depuis FaceSnapComponent : template, TS et styles.



Bien sûr ! On n'a aucun moyen de passer un FaceSnap au  @Input()  de ce component, car c'est le Router qui l'insère !

Heureusement, nous avons une solution !

Afin de récupérer les informations de la route activée, vous allez injecter **ActivatedRoute**dans SingleFaceSnapComponent, sans oublier d'enlever le décorateur  @Input()  de la propriété  faceSnap  :



Il nous reste qu’á d’ajouter une route dynamique vers le SingleFaceSnapComponent.

