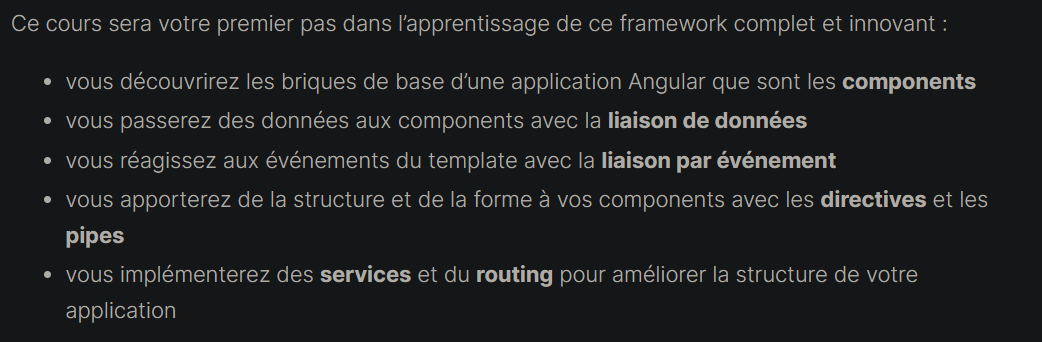
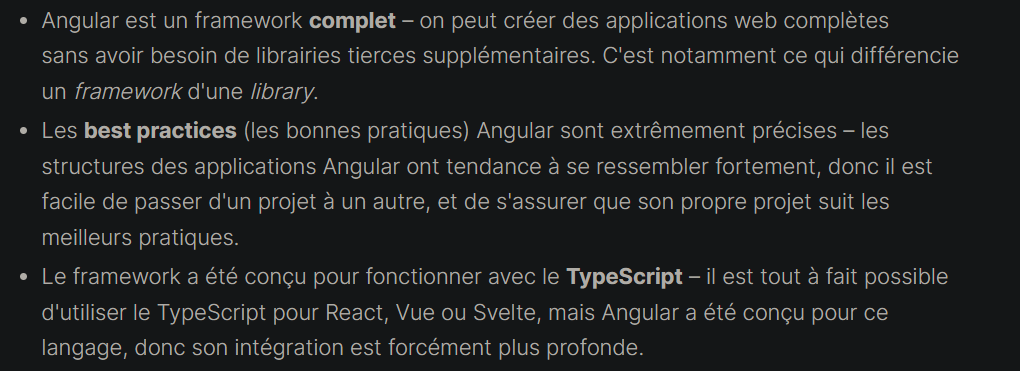
**Angular !**

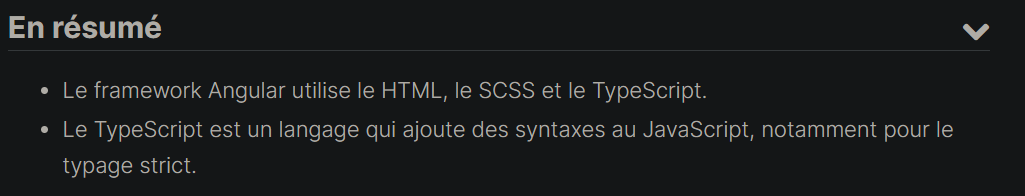


Documentation Typescirpt

<https://www.typescriptlang.org/>

Les avantages d’Angular !

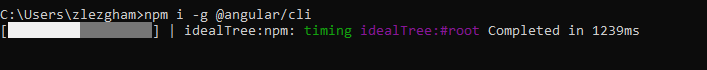




To develop using angular, you need :

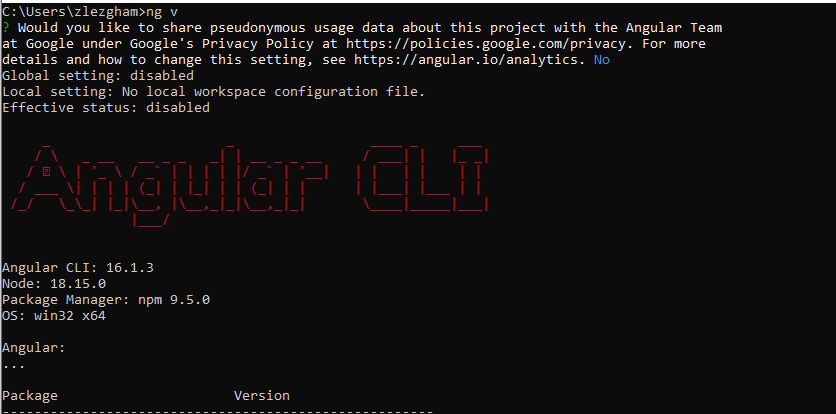
* Node
* Npm (node package manager)
* Cli : command line interface => allows us to create / manipulate / deploy our angular projects.

To install cli :

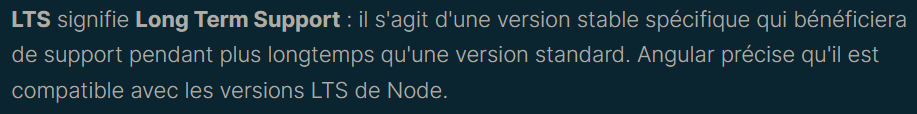


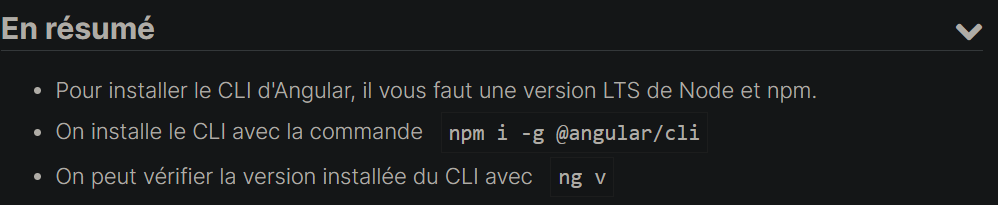
-g means that we install the package globaly !

Ng is a cli command !



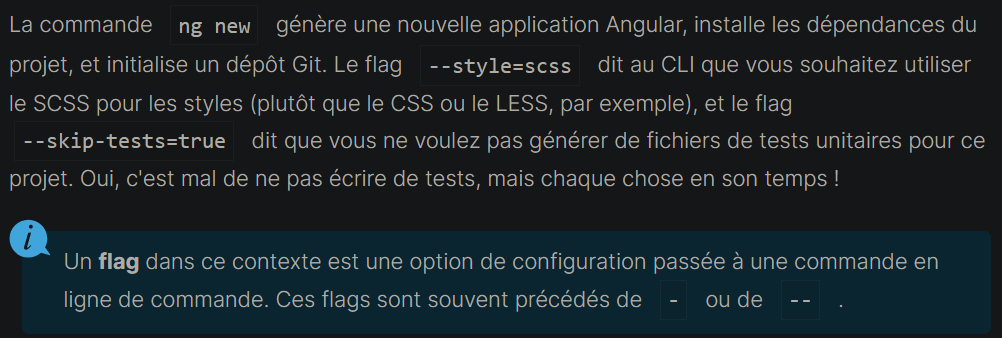
ng v to check the version

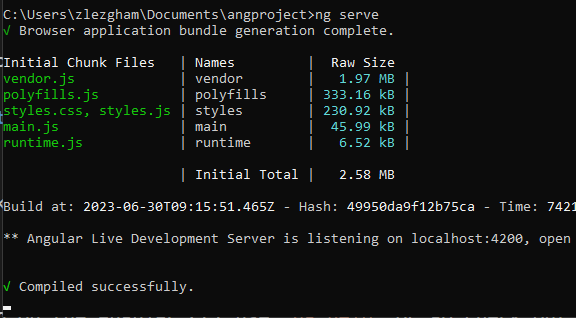




Create a new angular project !



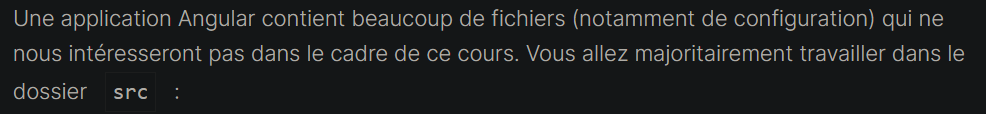


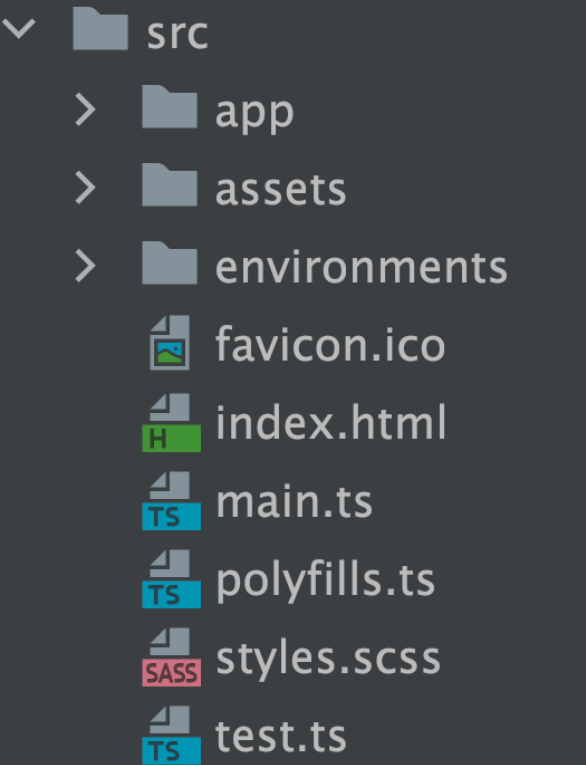


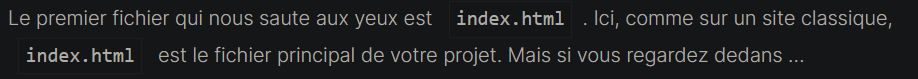
ng serve to start the dev server !

it starts on port 4200

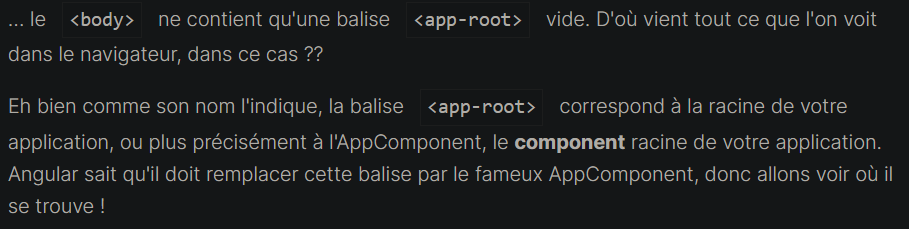




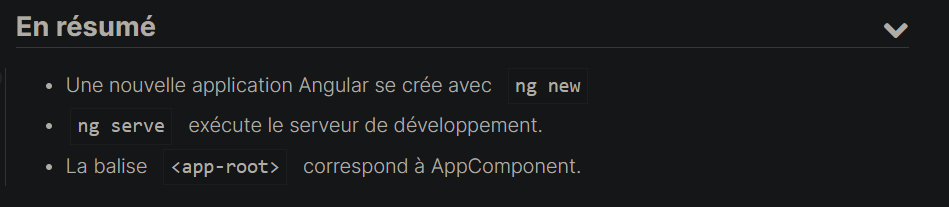












**Les components dans Angular !**

Le mot *component* en anglais signifie "composant", et à juste titre : les components sont les composants de base d'une application Angular. Une application Angular peut être vuecomme une arborescence de components avec AppComponent comme component racine.

En pratique, un component va associer un fichier HTML, un fichier SCSS et un fichier TypeScript : c'est le cas de l'AppComponent généré par le CLI, par exemple.

Pour créer un nouveau composant, on utilise cli :

Depuis une ligne de command, on exécute :

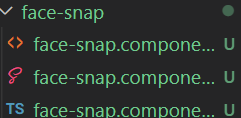


Angular traduit face-snap en FaceSnapComponent !

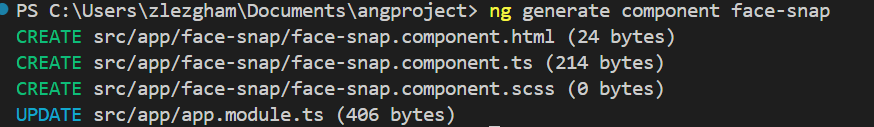
Comme pour app-root qui se traduit en AppComponent

-s => S .

Cela crée un dossier avec 3 fichiers, un html, un scss et un ts :



et ça modifie le app.module.ts aussi !



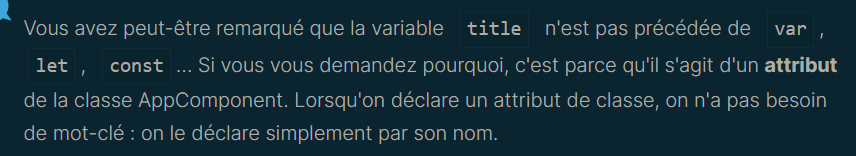


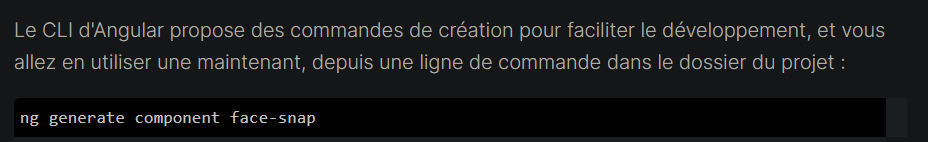
En ajoutant le faceSnapComponent dans les imports et dans les declarations !

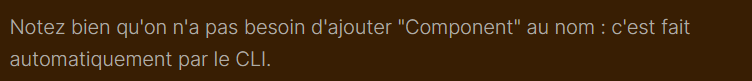
Un décorateur vient apporter des modifications à une classe comme @Component qui fait en sorte que le component en question soit utilisable dans notre application.



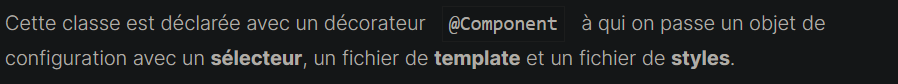
.html s’appelle le fichier template.



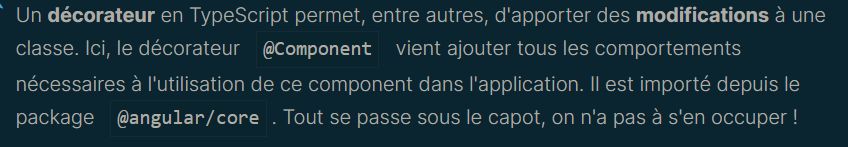


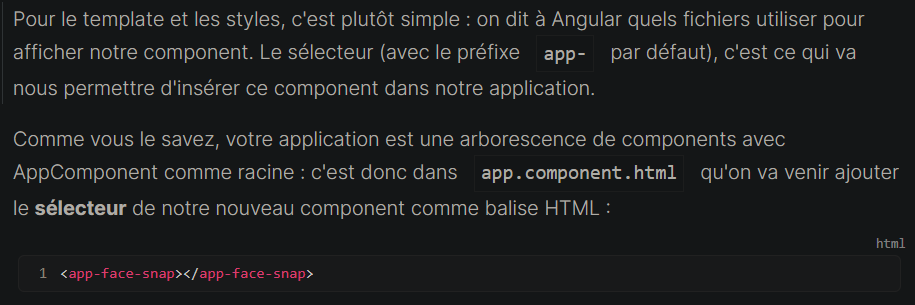


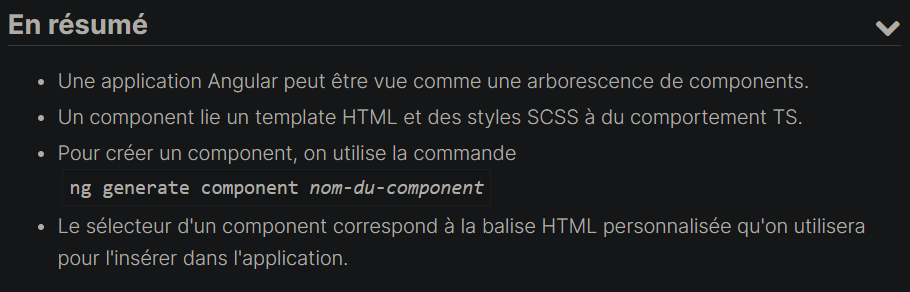




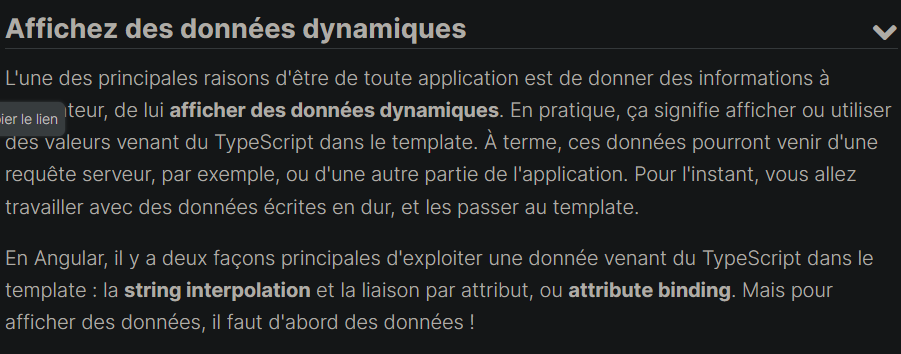


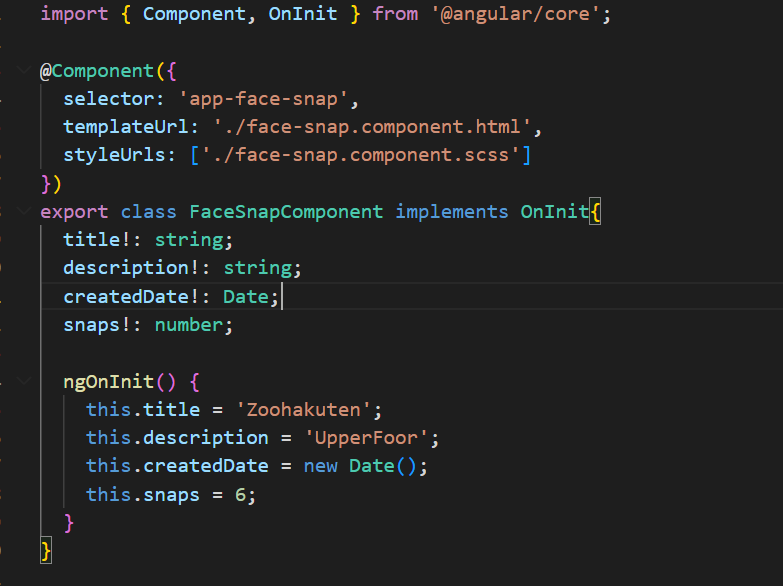






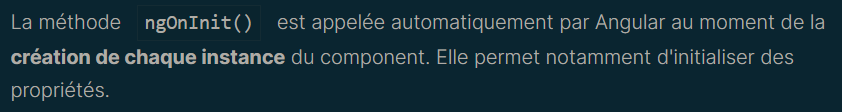
Data-Binding : on lie des valeurs de variables typescript à des éléments affichés aux utilisateurs dans notre template.

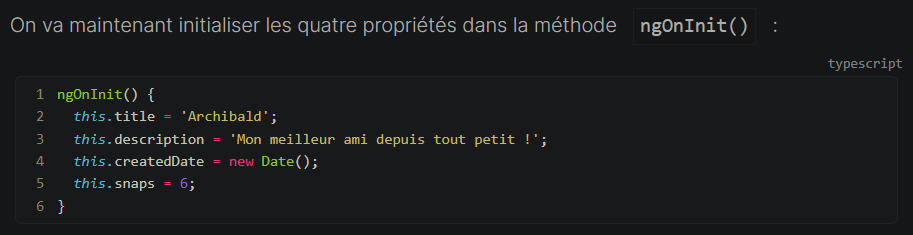


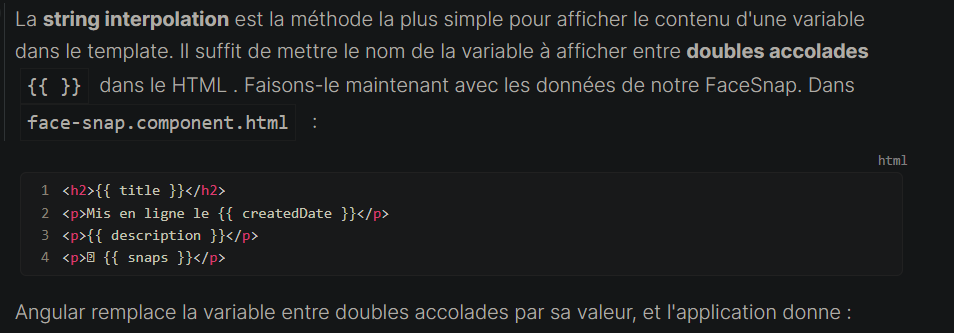


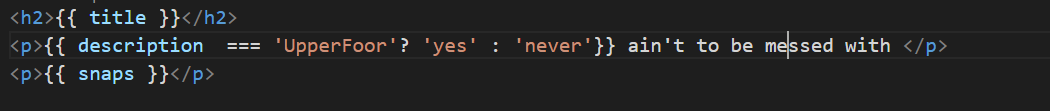






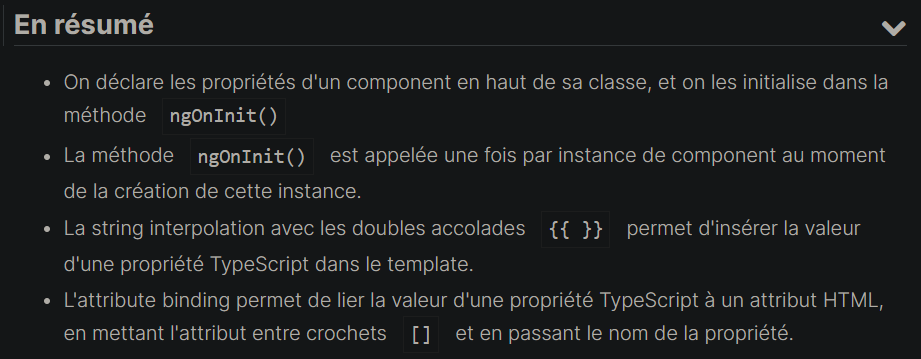




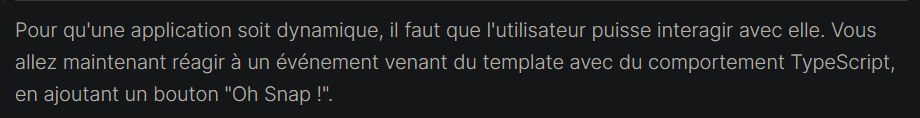


Attribut binding :



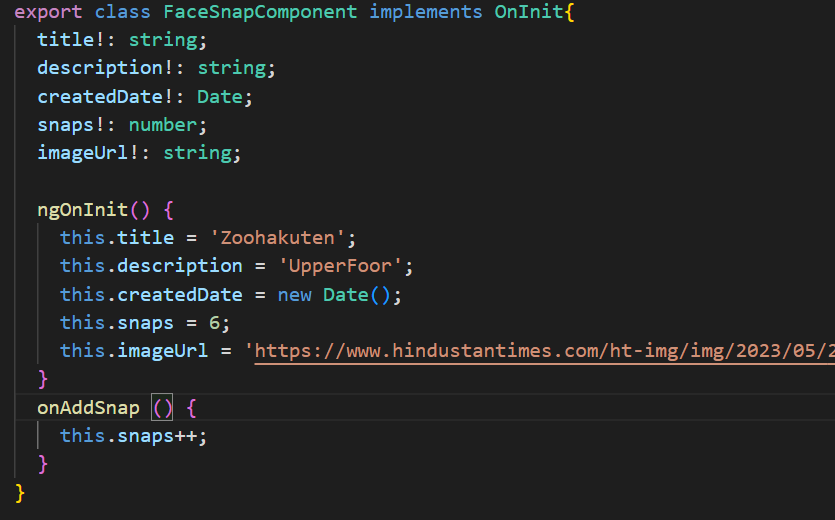


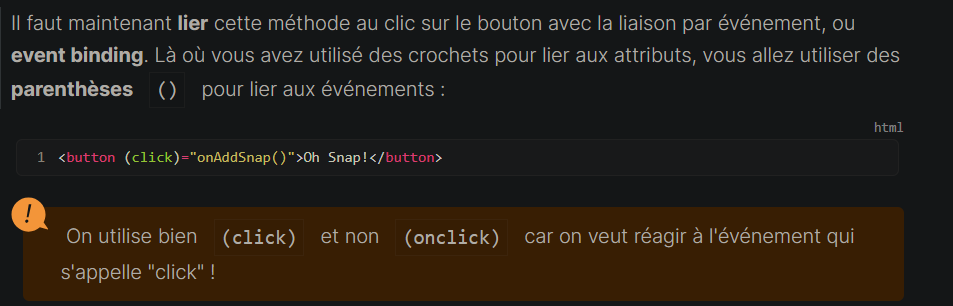
EVENT BINDING :



Régir à un événement du DOM

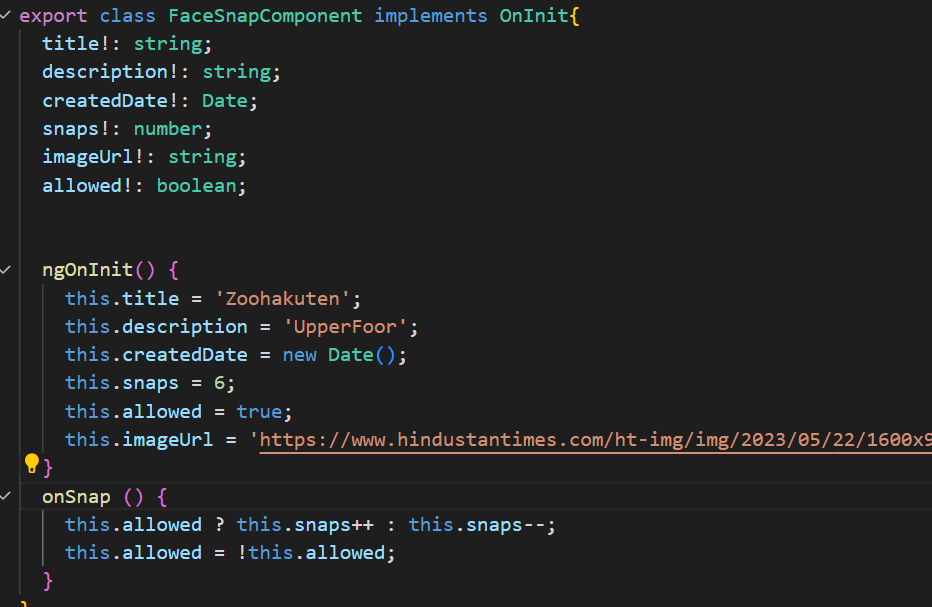


On met des parenthèses sur l’événement auquel on se lie, et on passe la méthode qui sera appela pour chaque clique ! 

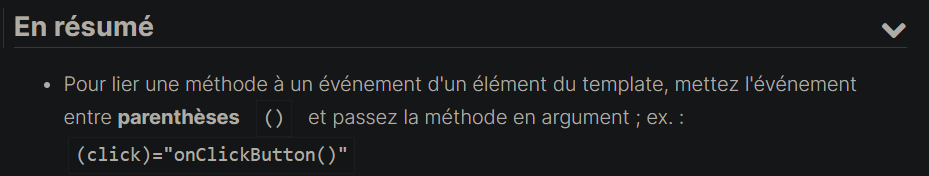


Les différentes instances des components sont totalement indépendantes.

Autoriser un seul snap, sinon décrémenter :

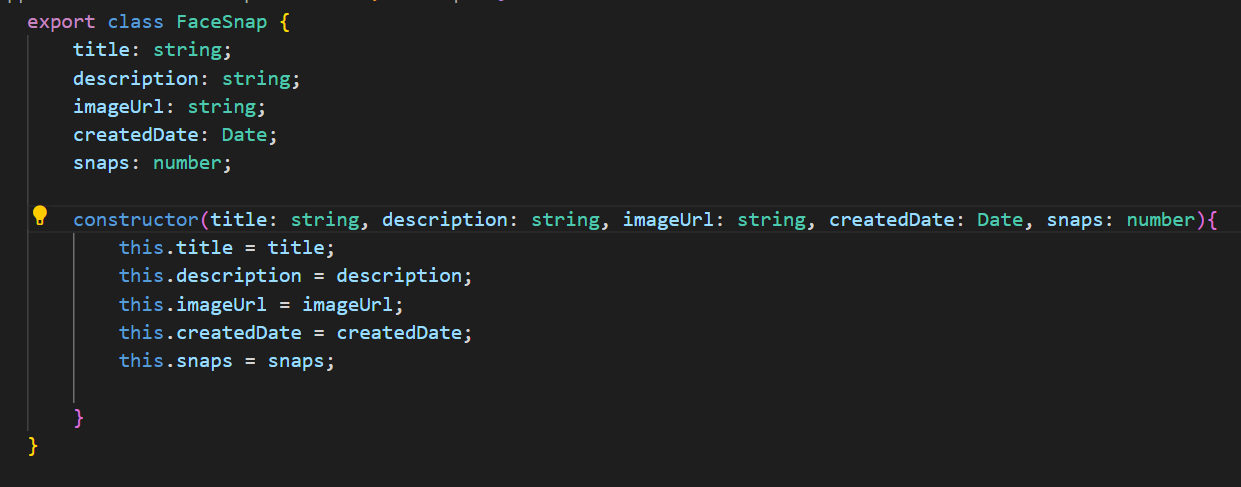






Instead of hard coding our component fields, we declare a class !

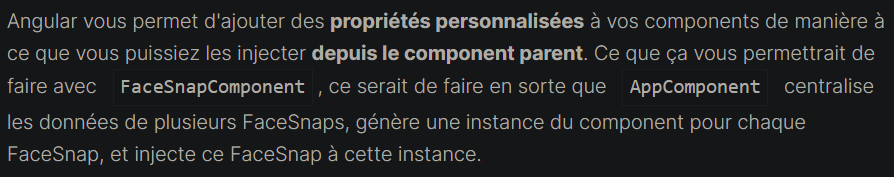
We either do this ! takes a lot of time and flemme

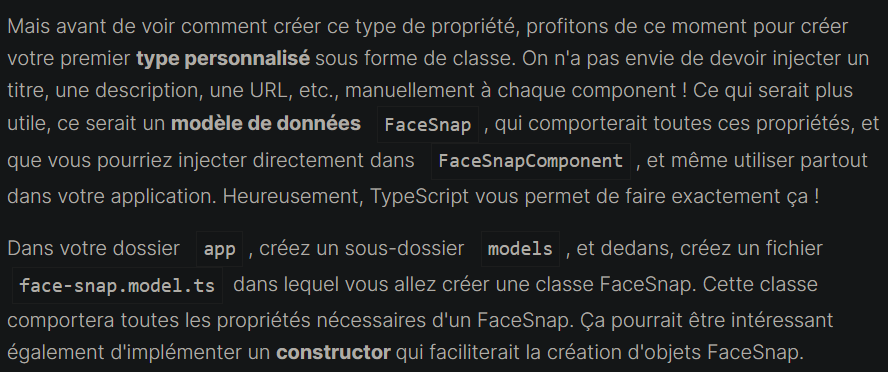


Or !



Le fait d’écrire public dit à typescript que cette classe aura ses propriétés et qu’on peut les passer comme argument à notre constructeur.



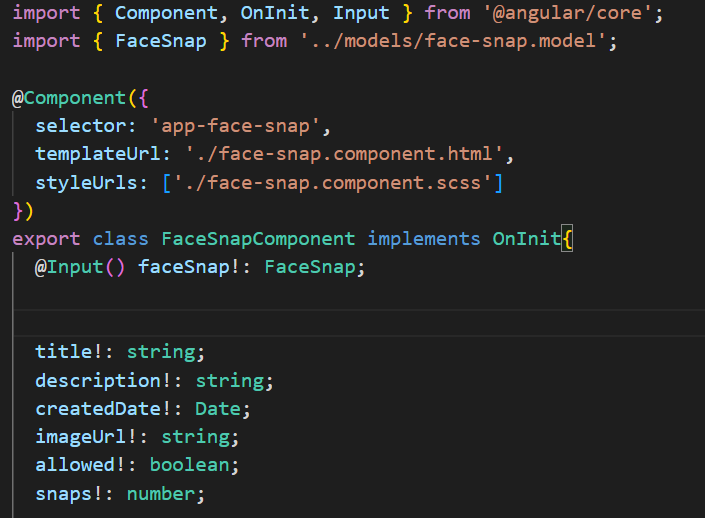


Now we can inject our special type from parent component to child component !

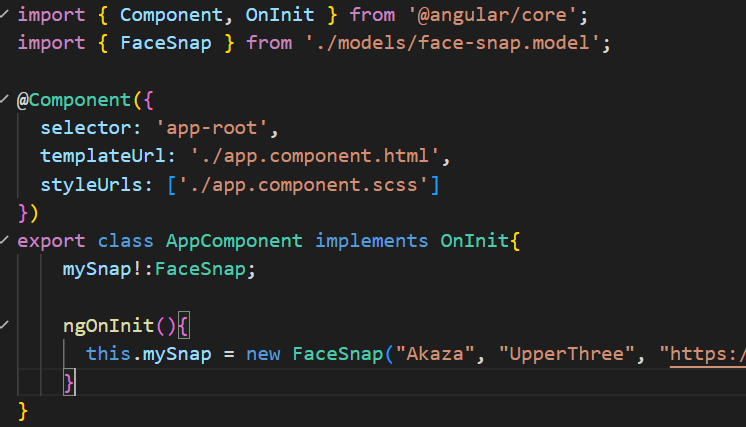
To do so, on utilise un décorateur :

@Input !

Dans le child component :

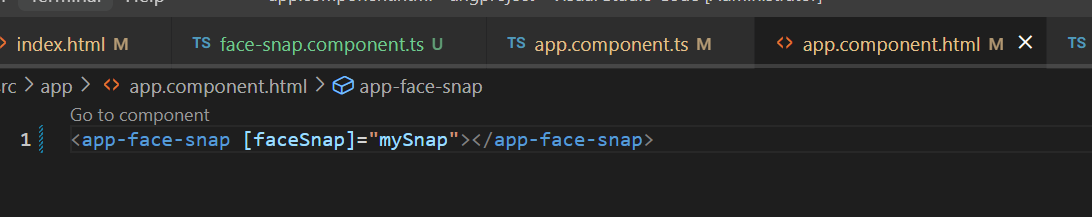


Once we created the module FaceSnap, in appcomponent.ts



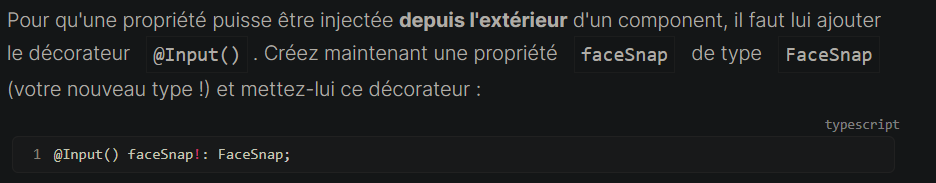
We declare an attribute mySnap of type FaceSnap !

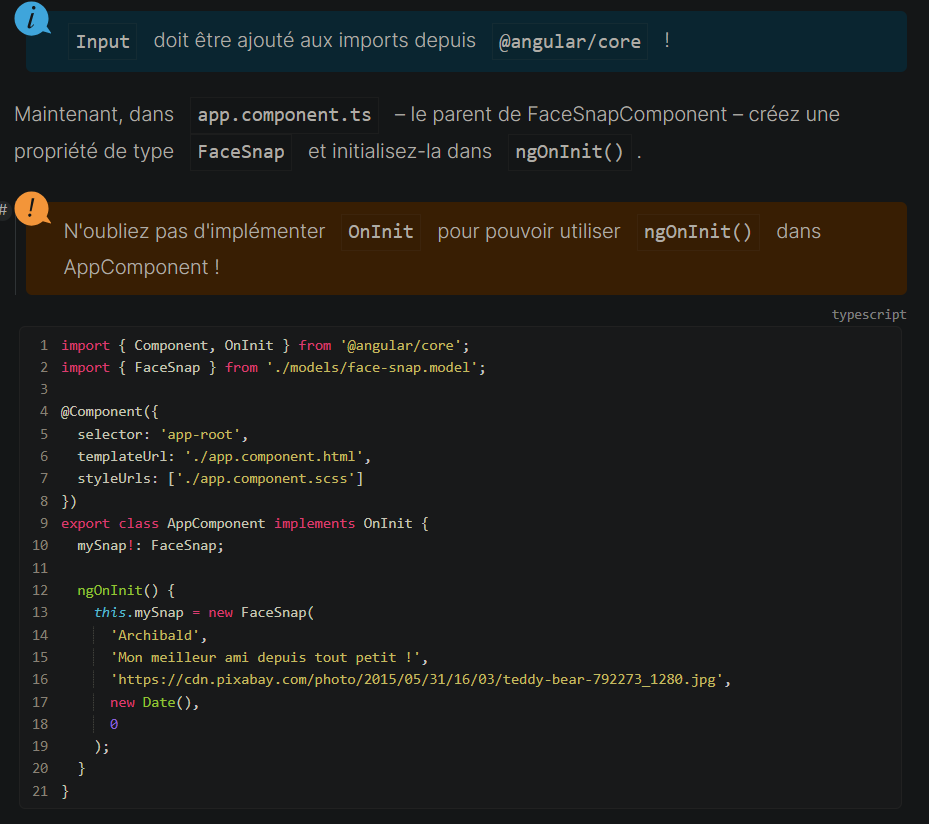
In appcomponent.html we use attribute binding !



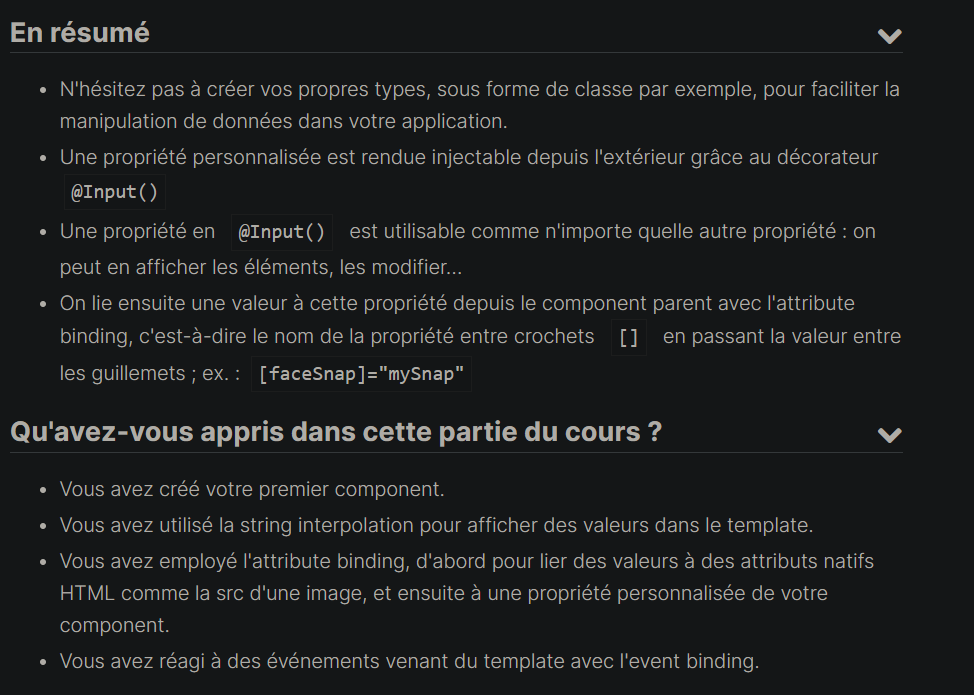
Since we declared input in my face-snap file, so facesnap knows that it will recieve an input of type FaceSnap that we name faceSnap !

And since we declared the attributein appcomponent we pass it as an argument.

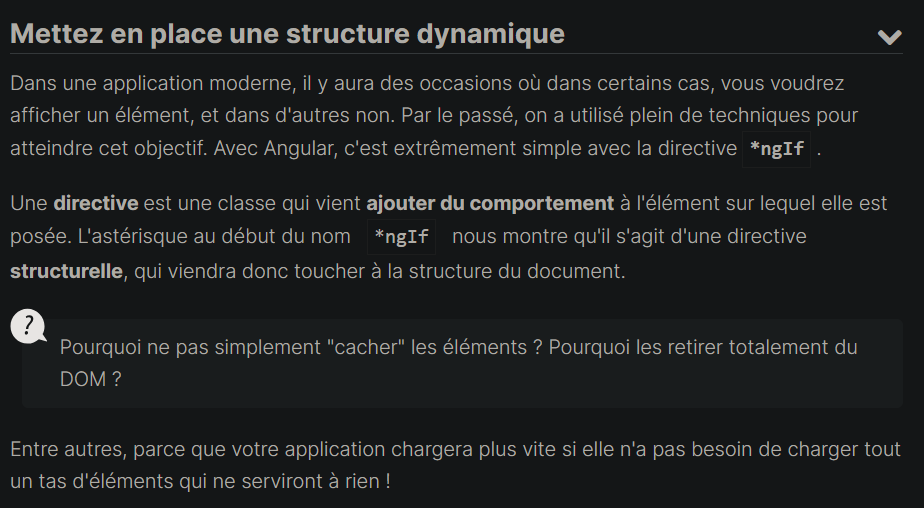


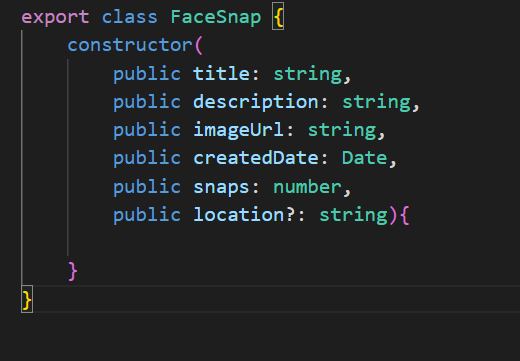






**Directive angulaire !**





Location ? : le ? permet d’indiquer que cet attribut est optionnel, il peut exister comme il ne le peut pas !



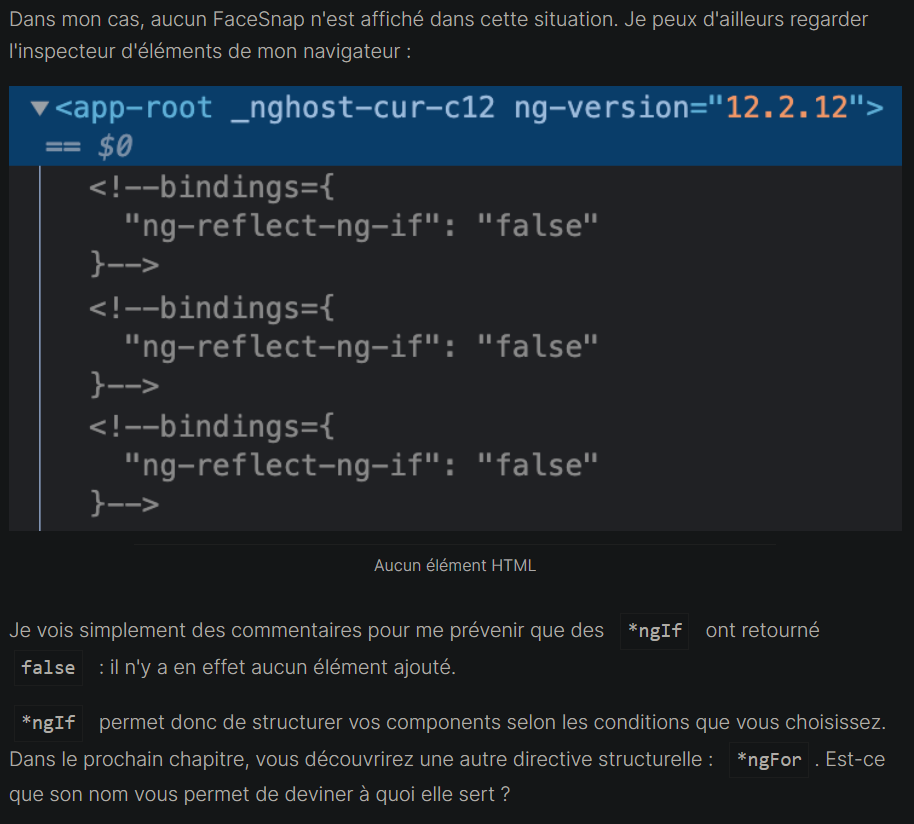


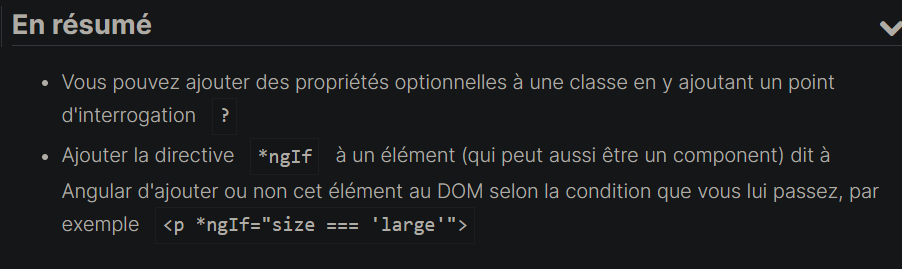


    <p \*ngIf="localisation === 'paris'">{{localisation}}</p>

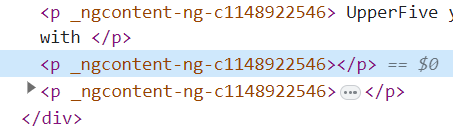
Ngif verifie si la condition est vérifié, si c’est le cas il affiche l’élement sinon, il n’affiche rien !



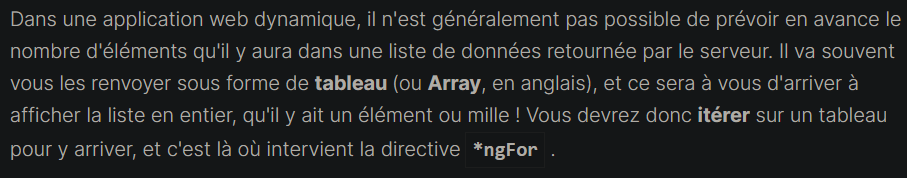




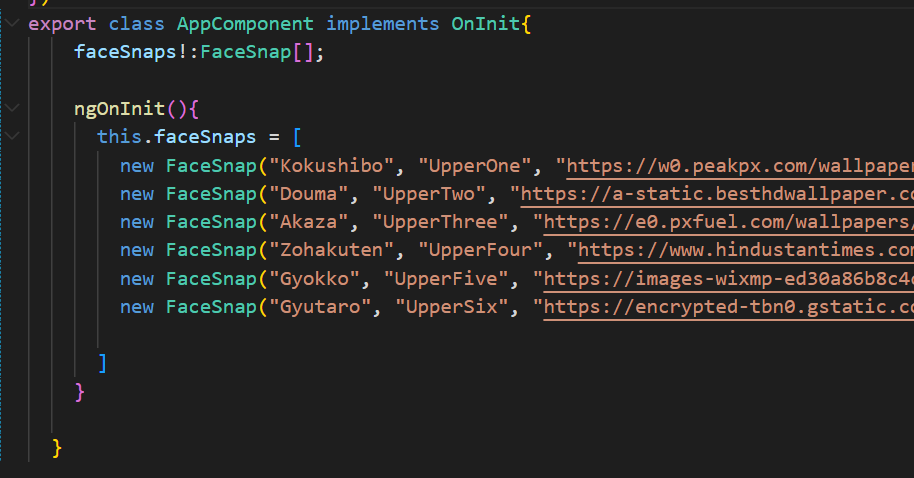
If we use ternaire we will have an empty balise !



\*ngFor :

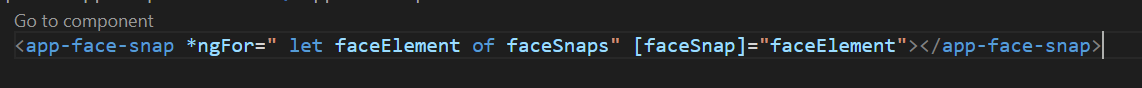


We change the code so that we have a loop !

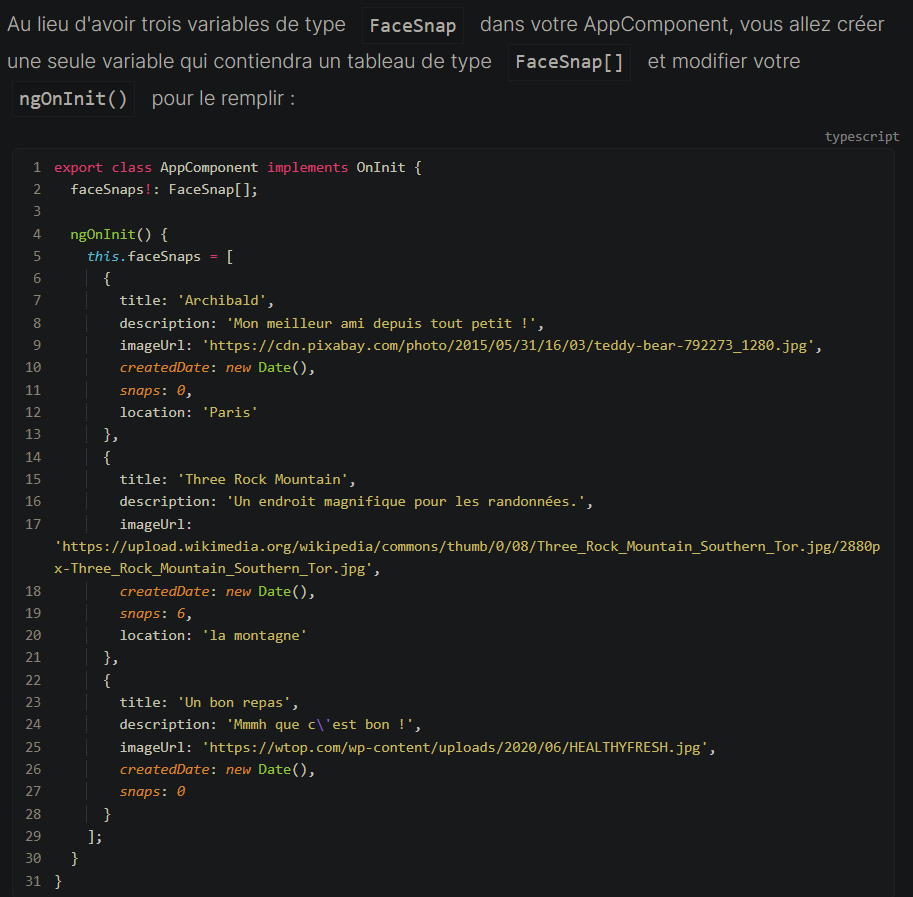


My appcomponent now has a table attribute that contains all my facesnap !

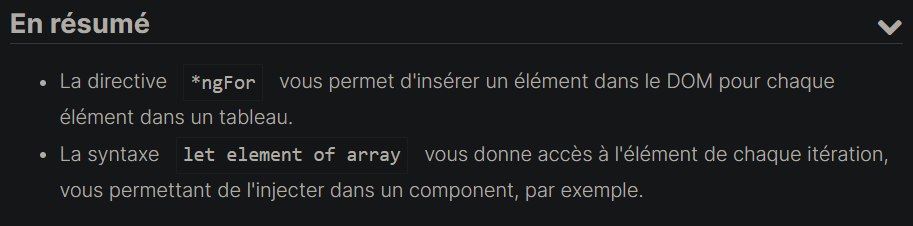
And instead of invoking app-face-snap for every facesnap :



We call it once and iterate over all my elements

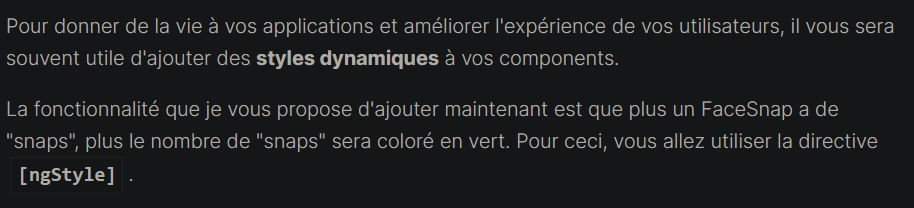


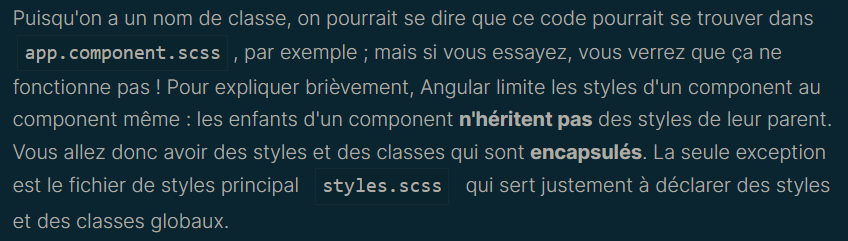




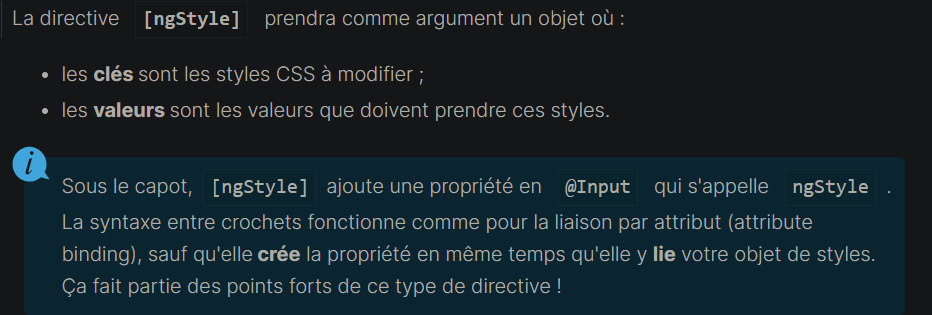
Ajout du style dynamique !

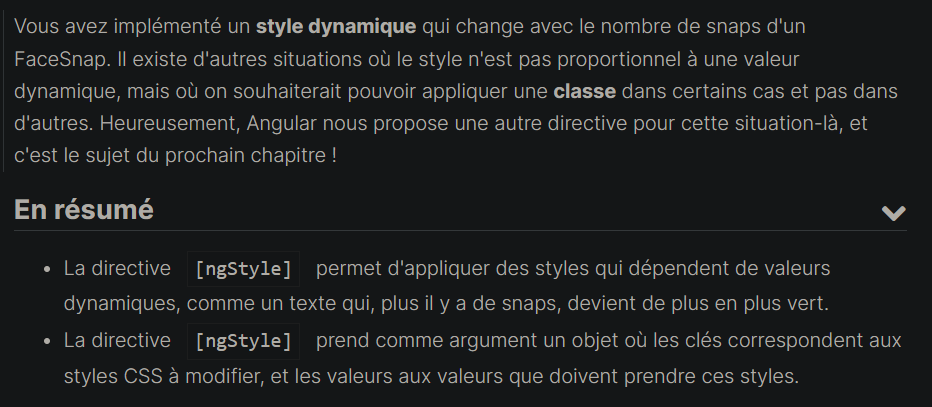
[ngStyle]



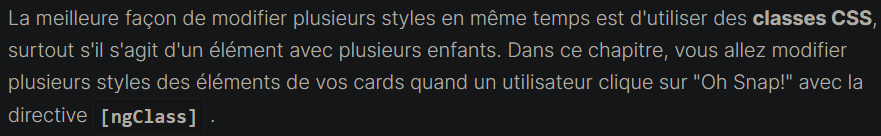


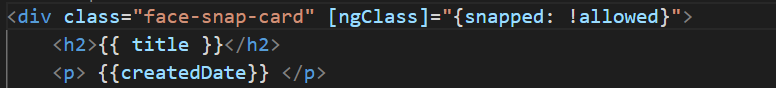






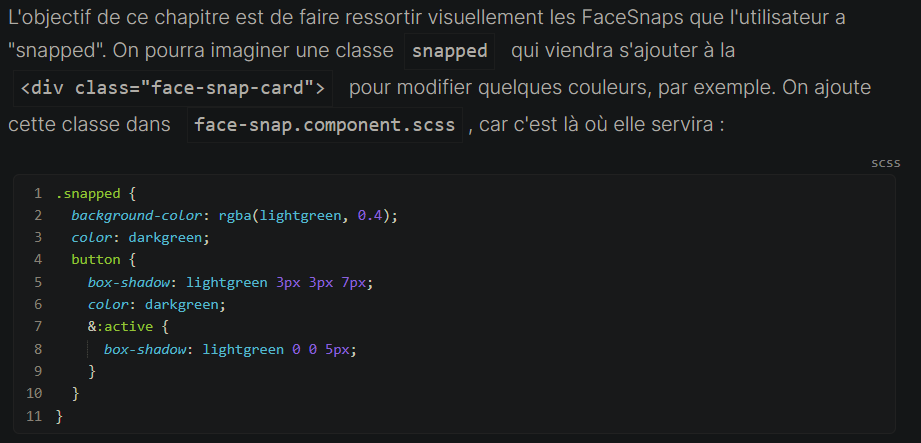
La directive [ngClass]

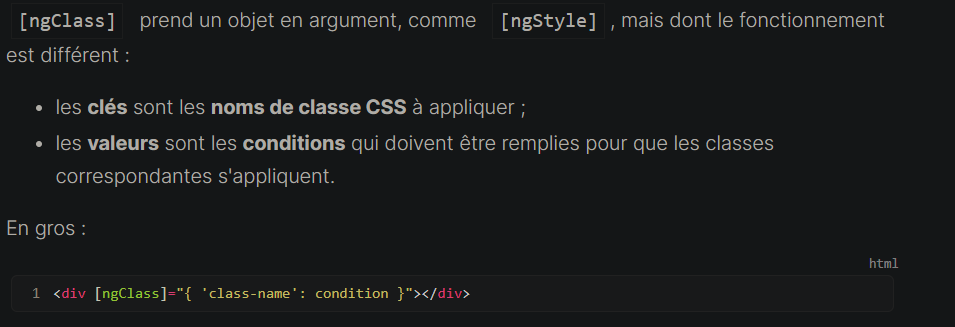


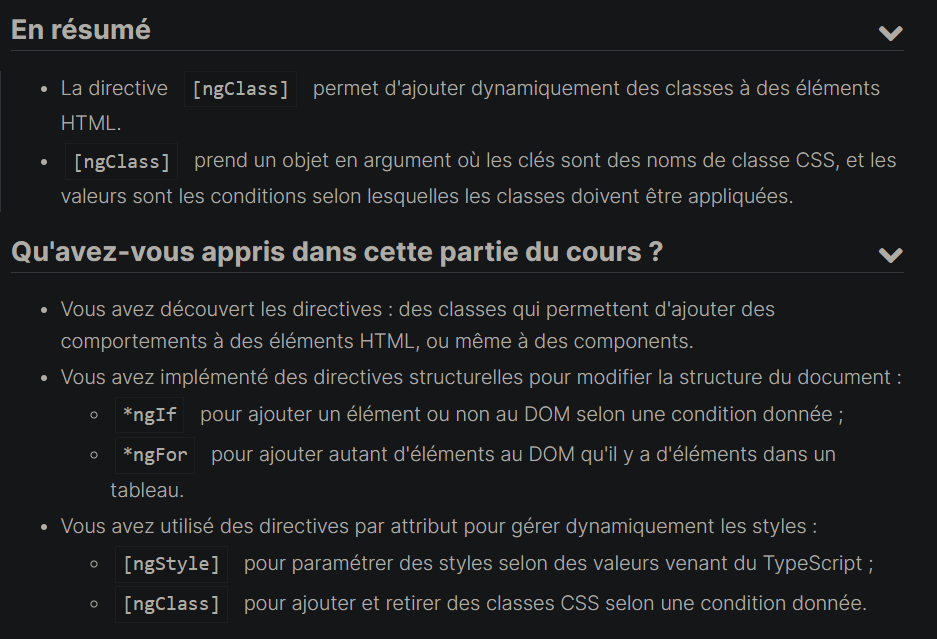


If allowed if false we apply the class snapped, if true we apply face-snap-card









Directive :

