

**Formateur**: Mr Delangle Maximilien

**Date :** 20/08/2019

# L’objectif

Ce cours est structuré pour des néophytes. Il fournira une initiation à Angular

# Qu’est-ce que Angular?

Angular est un framework fondé sur la technologie javascript, il possède également un langage appelé le TypeScript qui à était développé par Microsoft .

Angular est un framework “Front” dynamique et moderne il permet de communiquer avec différentes technologies “back”, que ce soit nodejs avec Express par exemple ou encore php avec Symfony etc ...

# Installation d’angular

Angular dispose de sa propre interface de commande ou CLI pour Command Line Interface en anglais, il va nous permettre d'exécuter des commandes depuis une console (Shell ) pour la création et structuration de votre code angular, mais aussi pour la production de votre app.

Pour avoir accès au CLI nous devrons installer au préalable nodeJs sur votre machine pour avoir accès à npm ( Node Package Manager ).

Pour celà il vous faudra le télécharger ici <https://nodejs.org/en/download/>

Puis ouvrir un Shell et avec npm installez la dernière mise à jour de ce dernier avec cette ligne de commande .



Le -g est un flag pour préciser que l’installation sera globale, c’est à dire installer sur votre machine entièrement.

Grâce à npm qui va nous permettre comme ci-dessus d’installer différente dépendance nous allons installer en global le CLI d’angular de cette manière.



Puis maintenant à l’aide des commandes cd, ls et mkdir

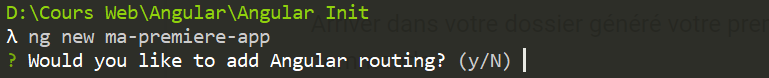
Créer un dossier de cours dans lequel vous générez votre application.

Arriver dans votre dossier générer votre première app Angular avec cette ligne de commande.

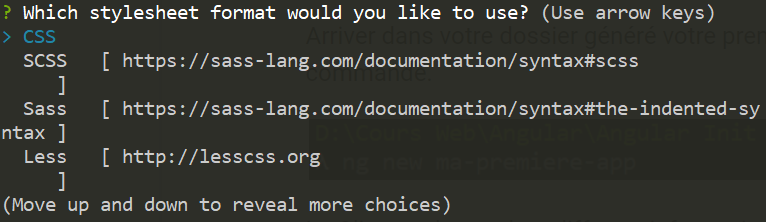


Le Cli vous demandera different information la première et non des moindres il vous demandera si vous souhaitez installer un module de routing.

Je vous conseille de l’installer en faisant y :



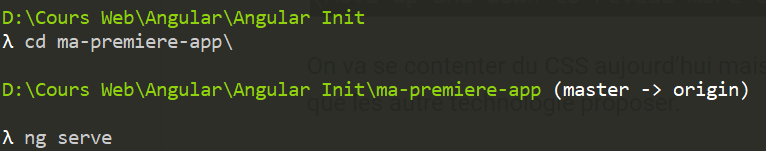
Puis il vous demandera avec quel langage vous souhaitez développer votre application vous avez le choix entre ses 4 technologies.



On va se contenter du CSS aujourd’hui mais le SCSS marche également très bien ainsi que les autre technologie proposées.

Puis il vous génère une sacré liste de fichier dès le début .

Je rentre dans le dossier en ligne de commande puis je lance le serveur local de cette manière .



En exécutant cette dernière commande on peut remarquer qu’il nous compile tous les fichiers et qu’ils sont visiblent sur le port 4200 et bien allons voir !

Pour celà il nous suffit de lancer <http://localhost:4200/> .

Et voilà votre première application est lancée .

# Structure d’Angular

A l’aide d’une editeur de texte un peu élaboré ( VSCode / Atom ) ou de votre IDE ( phpstorm / webstorm ect … ) lancez votre dossier récemment créé.

Dans un premier temps on peut voir 3 dossiers principaux le dossier e2e qui contient different information de configuration d’angular.

le dossier node\_modules qui contient toute les librairies nécessaire au fonctionnement de l’application, c’est ici qu’une librairie sera installé lorsqu’on fera un npm install .

Puis le dossier Src qui contient tout les script de notre application ce sera notre principal dossier de pratique.

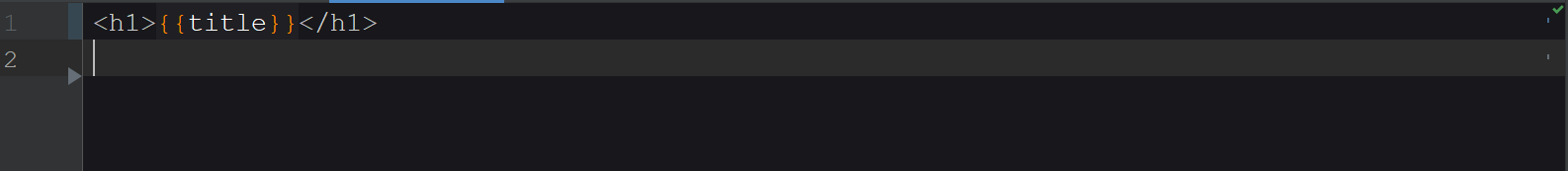
Dedans nous retrouverons 3 autre dossiers assez explicite app, asset et environnement.

Mais aussi d’autre fichiers à la racine, un fichier html et d’autre de différent langage ( sccs css ts ect … )

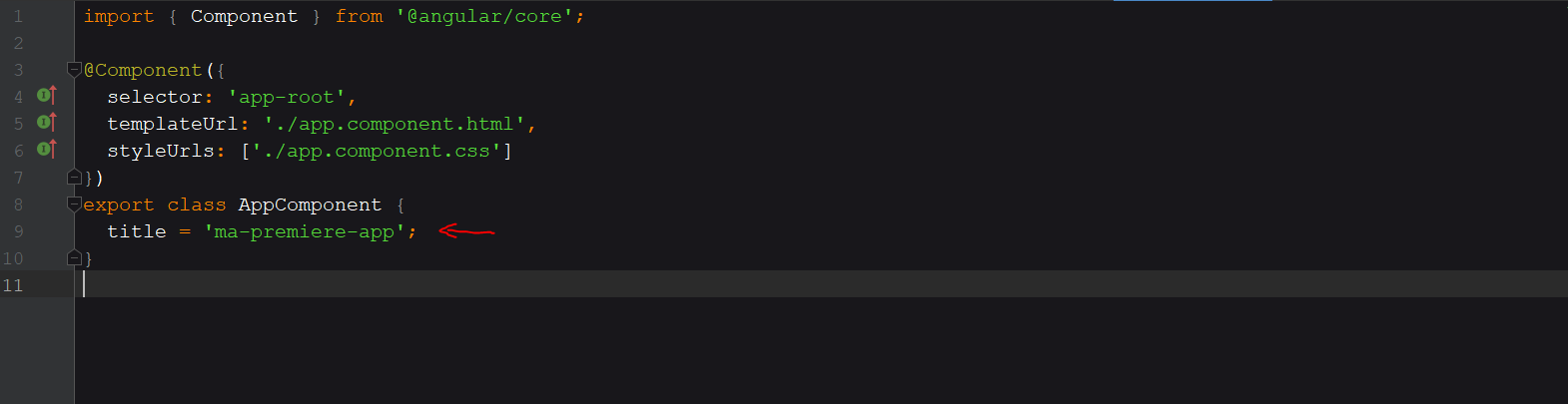
Dans le dossier app on pourra apercevoir différent fichier app.component.html , .ts et .css.

Pour bien comprendre le fonctionnement d’angular nous supprimerons le fichier app.component.spec.ts et on supprimera tout le code du fichier app.component.html

On laissera juste une balise h1 structuré de cette manière



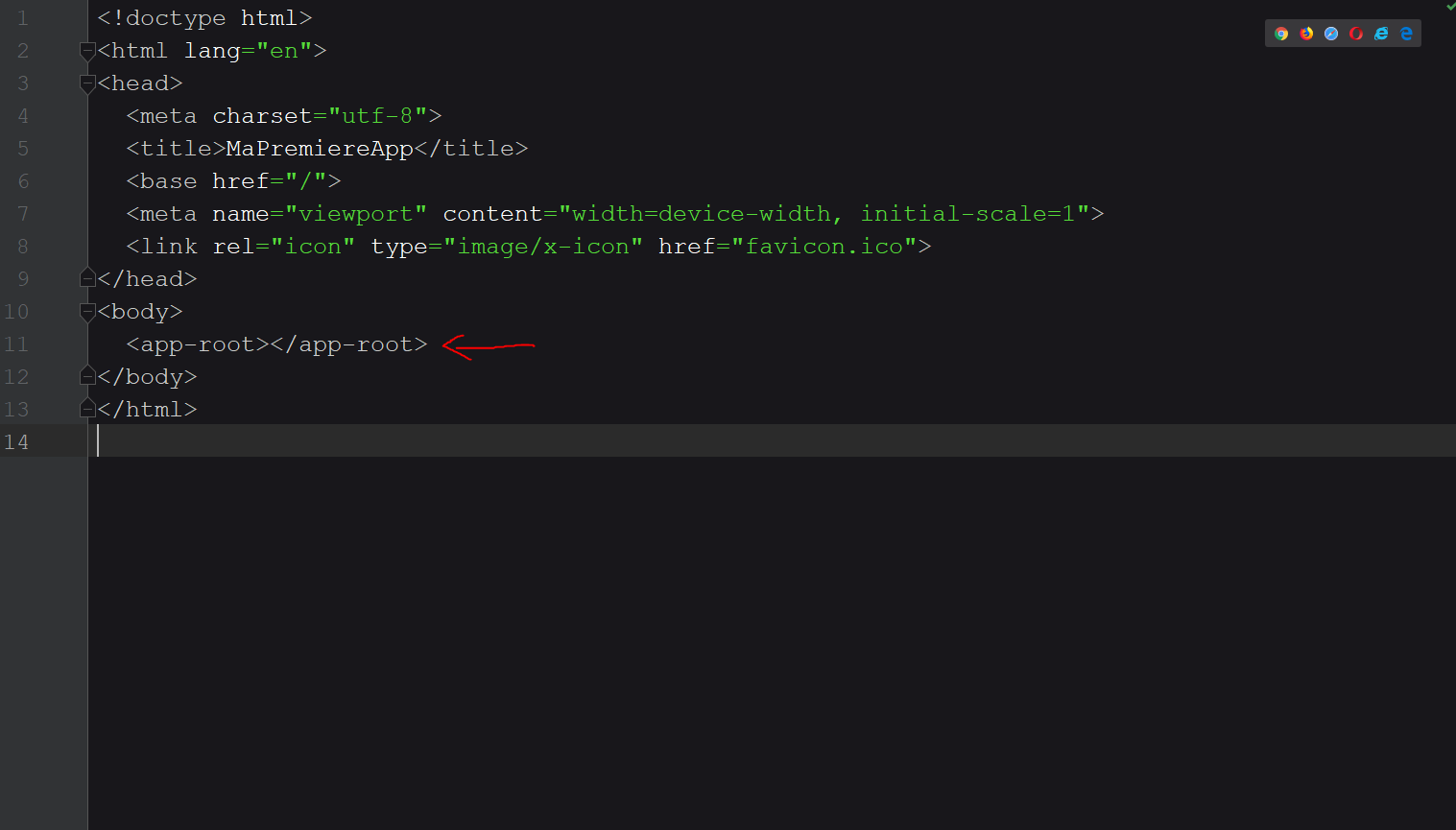
On a une interpolation grâce au langage mustache le “{{}}” qui récupère la valeur de la clef title qui est déclarer dans l’app.component.ts.



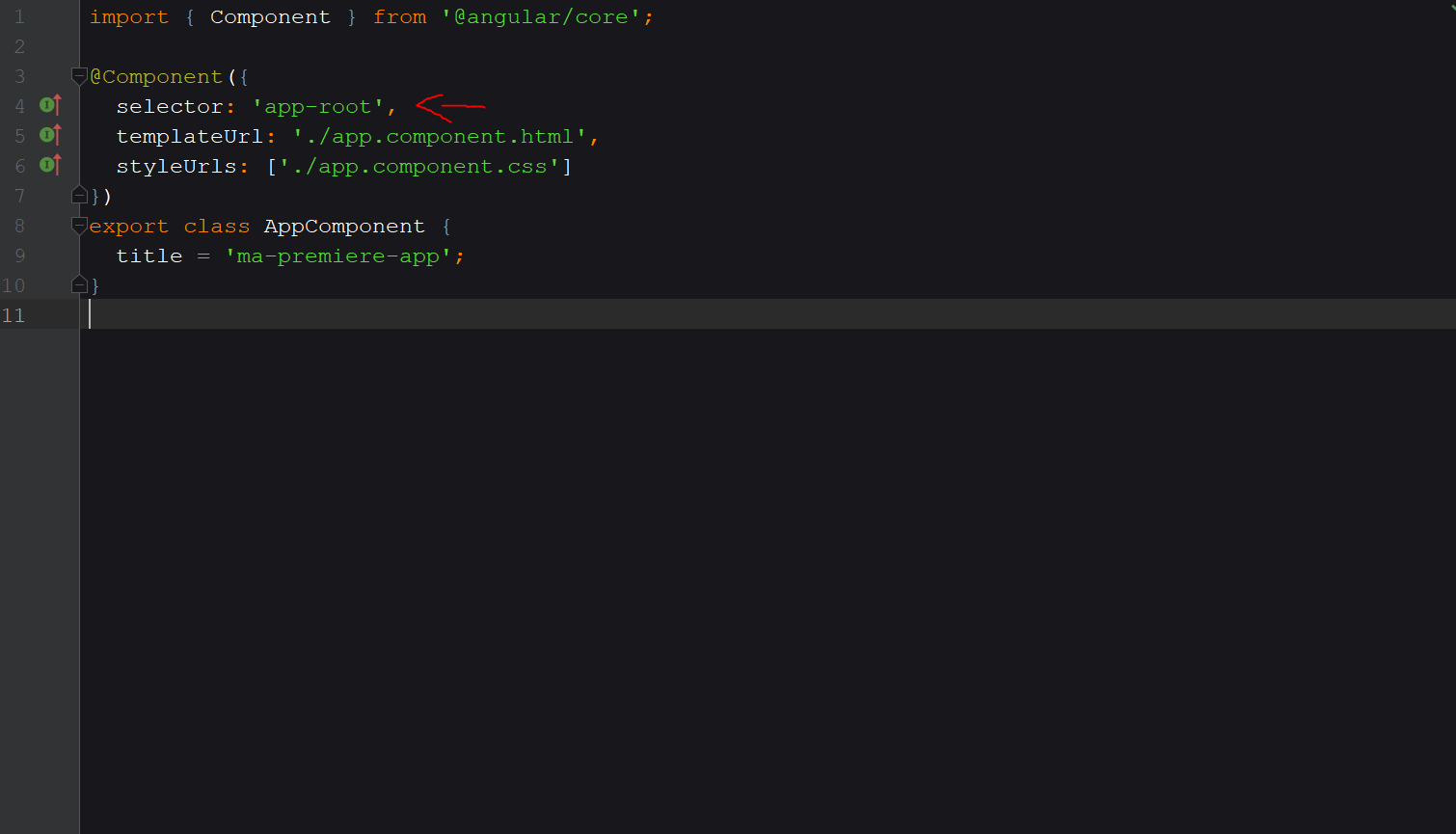
Et instinctivement le fichier app.component.css servira à la styliser le composant !

C’est très bien tout ça mais des composants font bien quelque chose, qu’est-ce qu’est un composant ?

Et bah Angular structure les choses tel qu’un composant et un morceau qui structure une page web, ainsi chaque composant est appelé en fonction de la structure dans l’index .html.

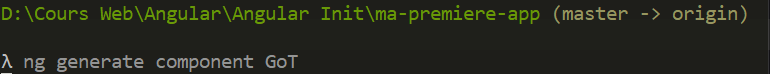


ici la balise app-root définit le composant app.composant , le nom de la balise est définie dans le fichier typescript du composant.



On peut donc passer différent informations et paramètres à chaque composant via leurs fichiers typescript.

Et si on génère un deuxième composant, oui j’ai bien dit généré car le CLI d’angular ne nous sert pas qu’a créé une structure vierge, mais également a créé des composant grâce à cette ligne de commande .



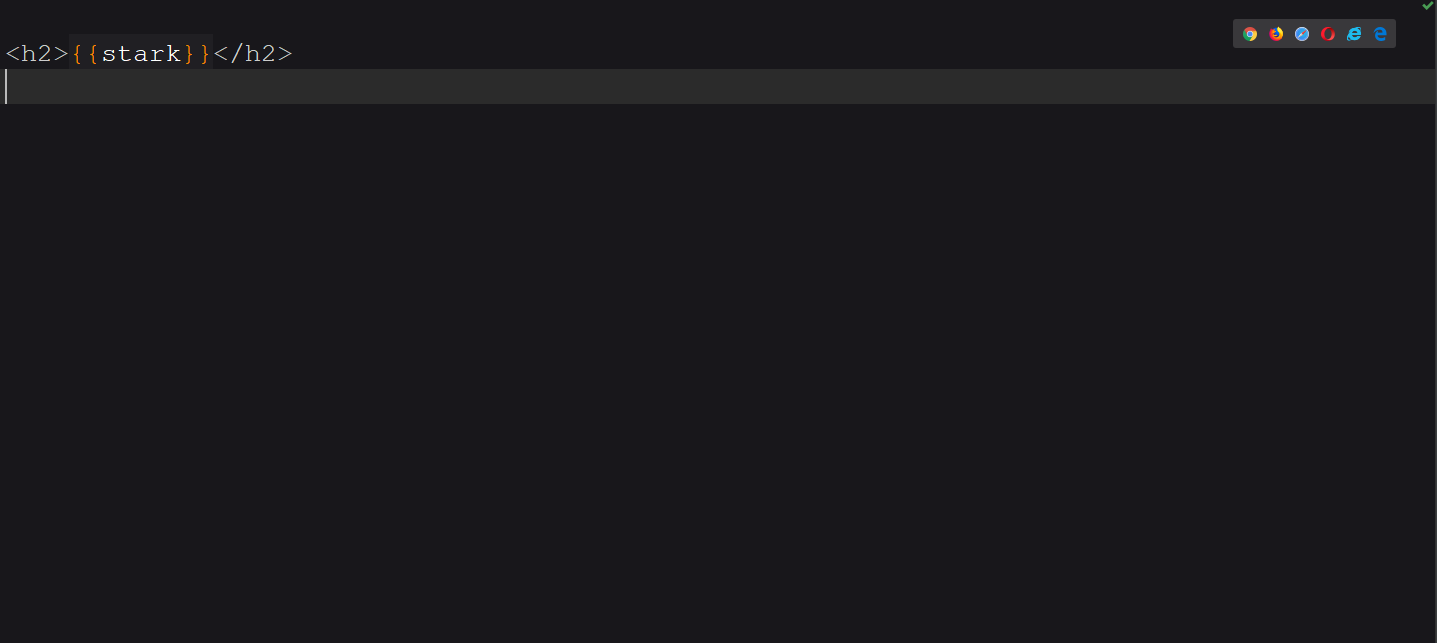
Je dis à cette ligne de commande de me créer un composant que je vais appellé GoT en référence à Games of Thrones bien sûr.

On vois que dans notre dossier app le CLI nous a généré un dossier go-t avec dedans une structure de composant classique.

Nous allons créer une variable stark qui aura pour variable Sanssa.

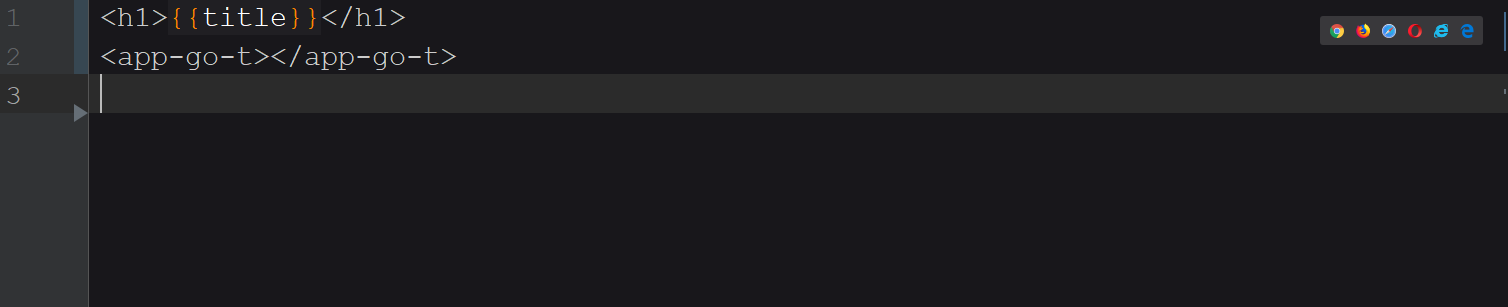


Puis nous allons simplement dans un h2 inséré notre variable via le fichier go-t.html



Puis pourquoi n'essayons nous pas d'insérer notre composant directement dans le composant de app-component ?

Comme ceci



On laisse angular recompiler et ?

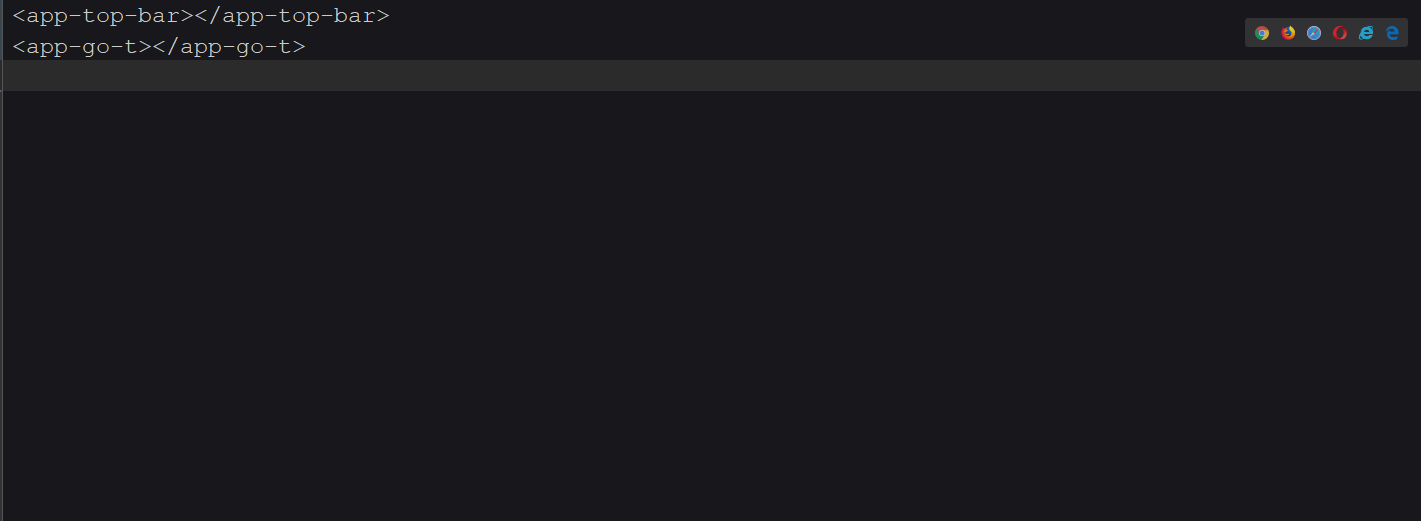
Ca marche ! On peut donc insérer des composants dans d’autre composant ça commence à devenir intéressant ! .

# Composant topBar

Attaquons nous à la création d’un composant qui restera que graphique pour le moment.

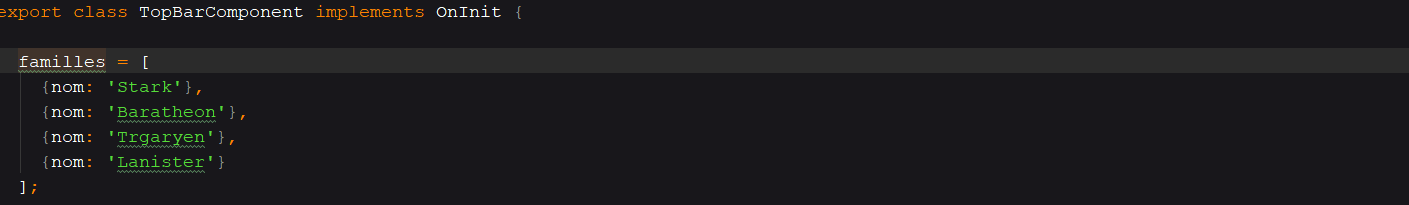
On génère un composant qu’on appellera topBar, puis insérons le dans notre app-component.

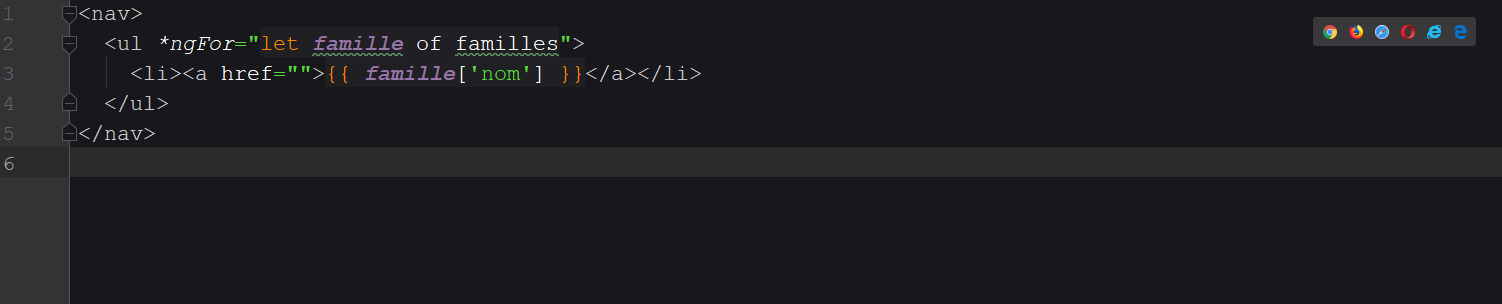
Ce qui nous donnera



Je crée une topBar avec un peu de style, dans ma topBar j’aurais 4 liens qui seront pour les 4 grandes familles de GoT. Stark, Baratheon, Targaryen, Lanister.

Les données de ses liens viendront d’une liste depuis notre topBar.component.ts .



Puis dans mon html je les affiches 

On peut voir que j’utilise une syntaxe assez particulière, un attribut “\*ngFor” qui n'est rien d’autre qu’un attribut qui permet de dire au moteur de rendu, attention j’ai un tableau et il faut que tu foreach dessus, puis pour chaque donnée de familles tu la stock dans une variable famille.

Ce qui nous permet comme cela d’afficher chaque éléments d’un tableau de JSON.

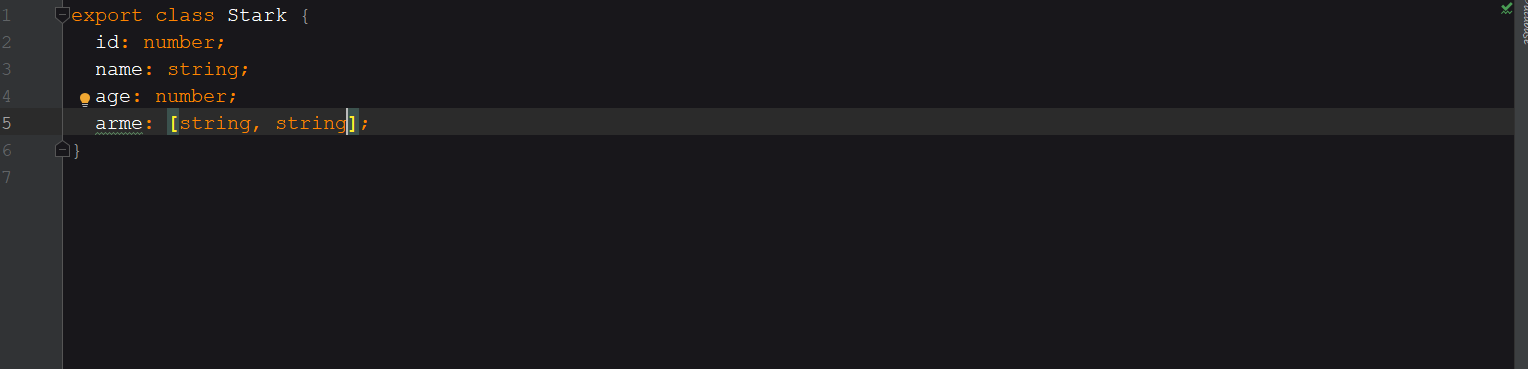
D’ailleur qui dit JSON dit Objet ( JavaScript Object Notation )

Voyons un peu les objets.

# Classe Angular

Qui dit passez des paramètres depuis un fichier ts, dit classe pour structurer nos objets !

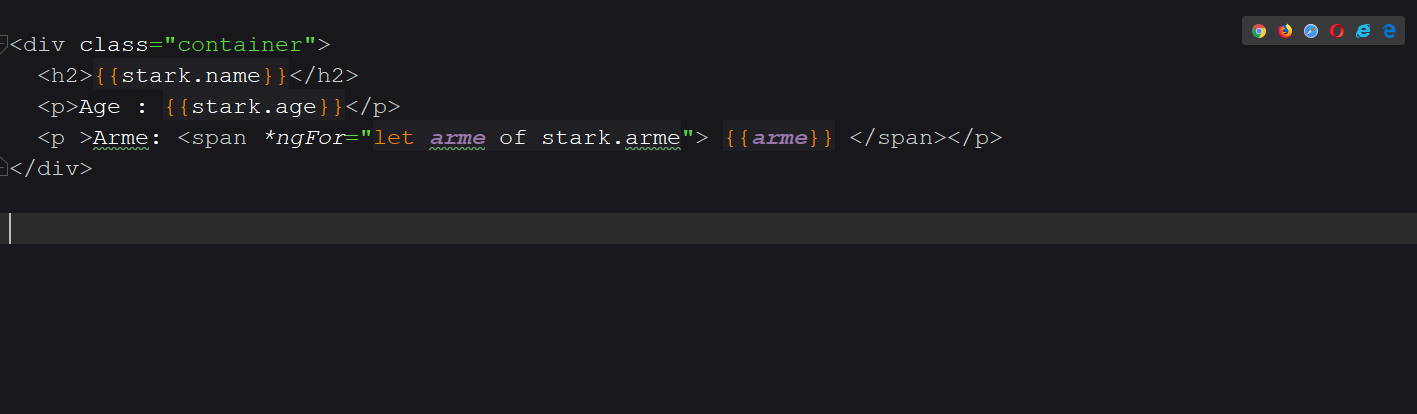
Je crée donc ma première class angular, pour cela il suffit de créer un fichier dans notre dossier app qu’on appellera simplement stark.ts , puis dans ce fichier j’exporte une class de cette manière

Puis dans notre go-t.component.ts, j’instancie un objet stark de cette manière.



On fera bien attention d’importer notre classe au début de notre fichier et bien sur à bien respecter le typage déclarer dans notre classe lors de notre instance.

Puis il nous suffira de binder les données dans le front comme jusqu'à maintenant.

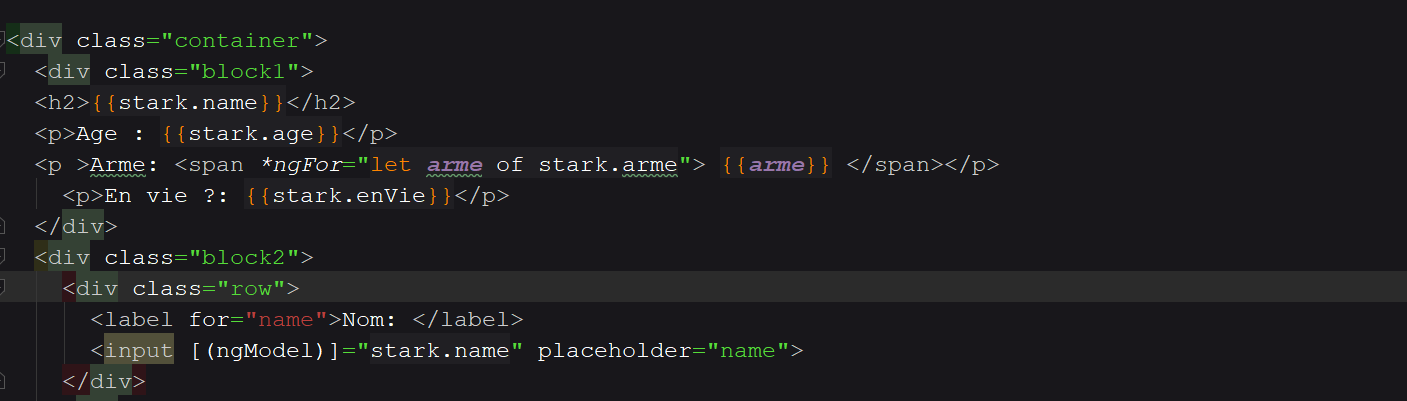


Si maintenant on veut modifier nos valeurs mais depuis le front, angular nous permet cela ! Grâce à un nouvel attribut qu’on appellera ngModel il nous permet donc de faire ceci.

Pour ça rajouter une propriété boolean en vie a notre classe.

Puis dans notre component.ts, rajouter la valeur de notre classe à notre objet.

Enfin dans l’HTML utiliser le ngmodel de cette façon:



Le NgModel fera le lien directement avec la valeur et permettra de changer celle ci et dans le même temps de l’afficher, alors en réalité ce n’est pas instantané mais ceci se déroule en milliseconde, ce qui donne un effet instantané.

Attention , pour les armes on partira du principe que nous ne voulons pas modifier la valeur. En effet pour tout ce qui est tableau de valeur c’est une autre histoire que nous verrons plus tard.

Retournons voir notre app.module.ts,

On peut remarquer qu’il a quelque peu changer.

En effet lorsque vous créez un nouveau composant il ajoute dans les déclarations votre composant, tout comme dans les imports vous pouvez apercevoir le FormsModule qui est importer depuis la librairies angular/forms c’est précisément celle ci qui nous permet d'interagir avec le ngModel.

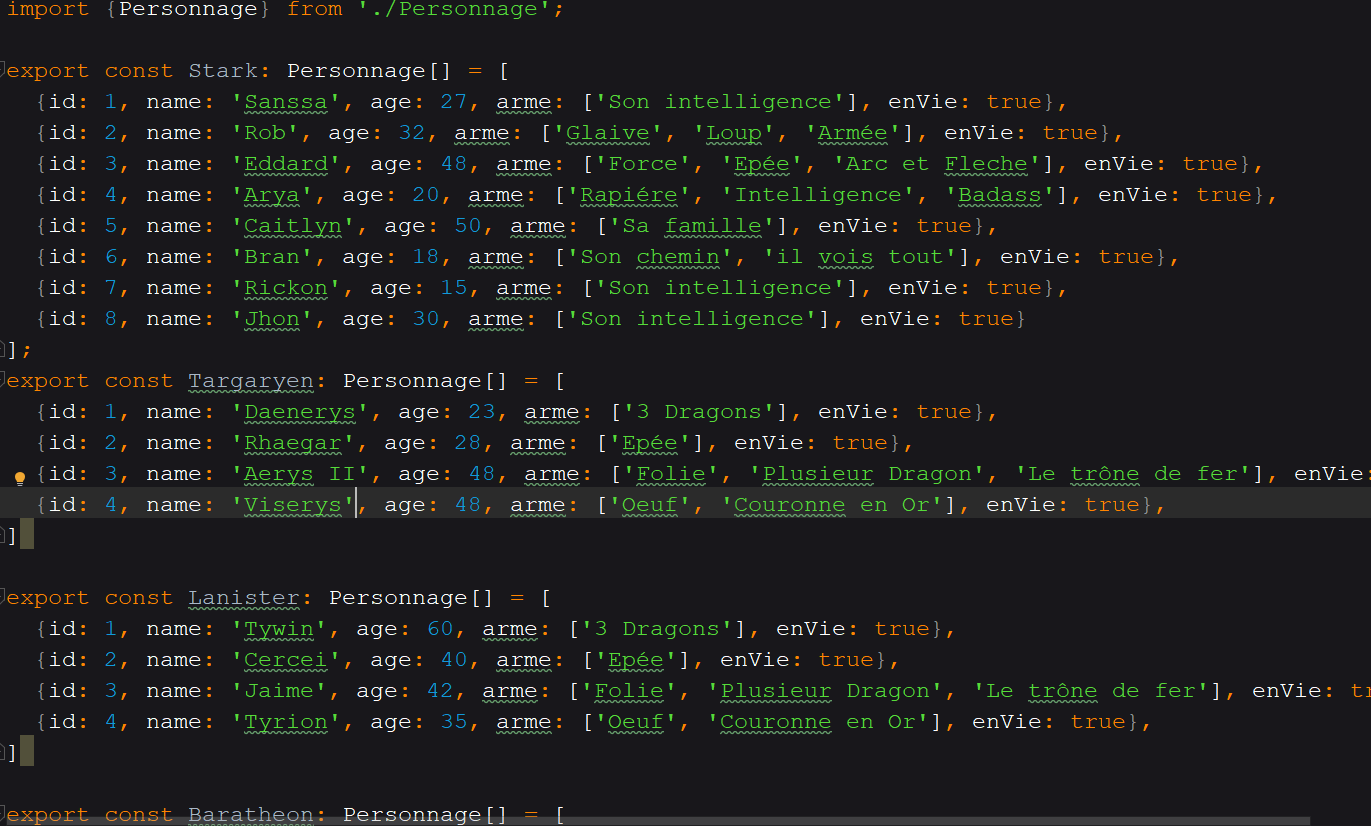
# Simulation de données.

Maintenant imaginons que nous récupérons un jeux de données depuis un back-end.

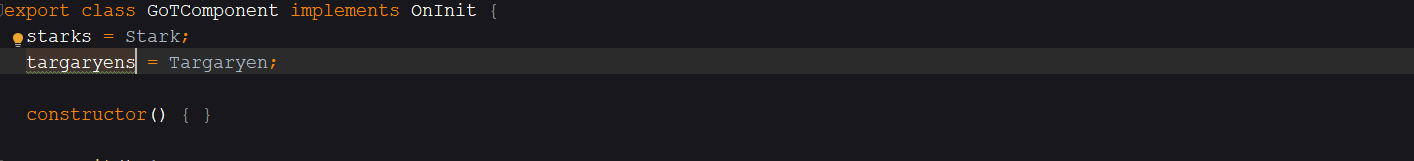
Pour ce faire nous allons créer ce jeux de données à la main , à la racine de votre dossier app créé un fichier data.ts .

Modifier votre class et appelons la personnage

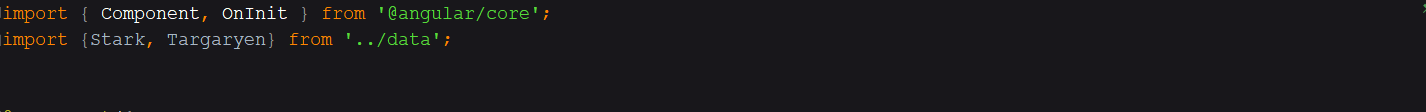
Dans data.ts importer votre classe Personnage, puis crée des constantes contenant un tableau d’objet personnage.

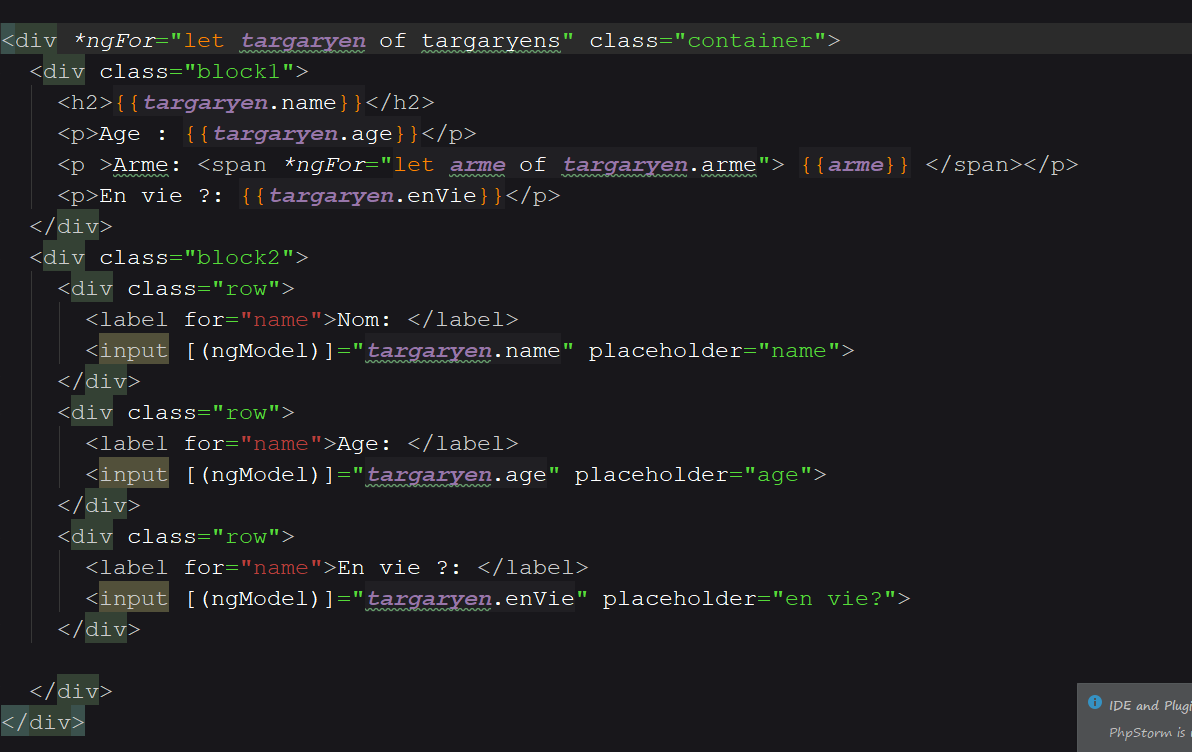


Ensuite dans notre fichier typescript il nous suffira par la suite de déclarer une variable contenant notre constante



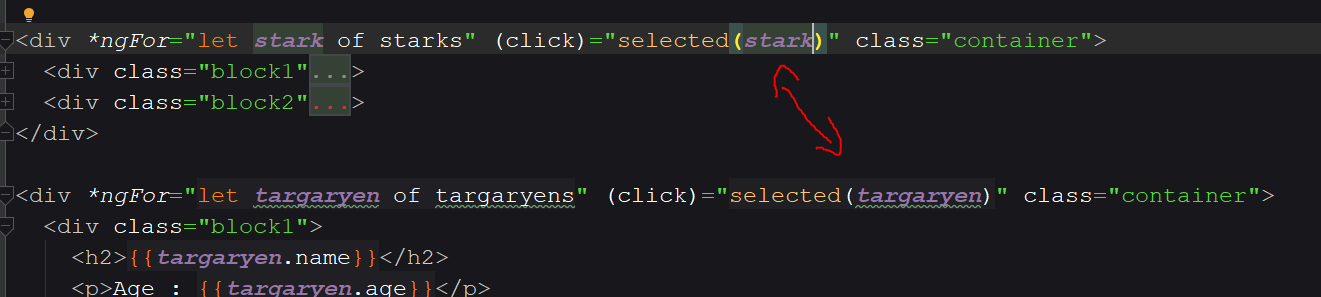
Sans oublier d’importer les bonnes librairies





# Déclencher des événements

Pour déclencher des événements sur angular il nous faudra utiliser un type d’événements entre parenthèse par exemple (click) et assigner celle ci a une fonction déclaré dans le typescript. Par exemple nous pourrions avoir.



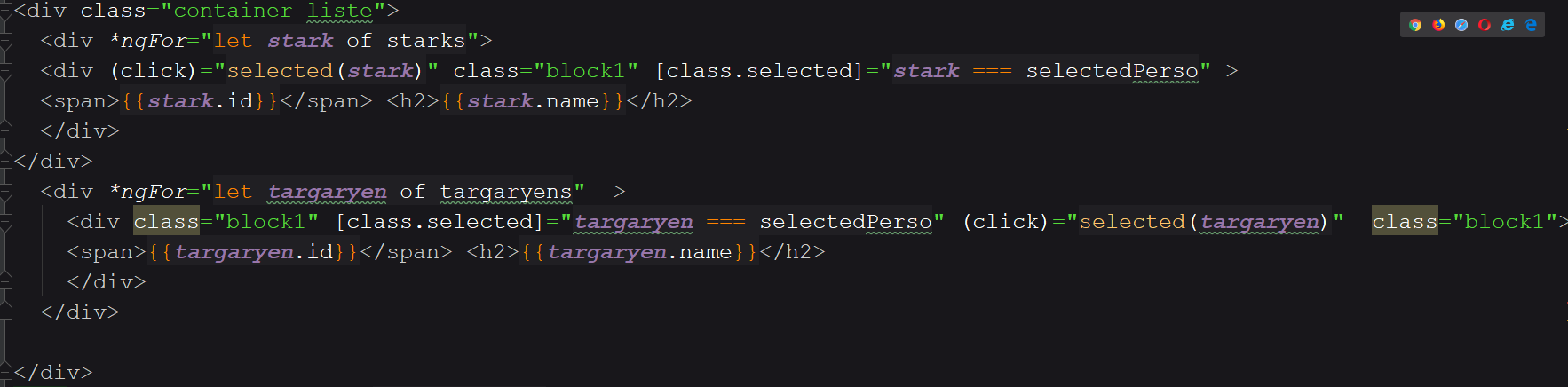
Qui déclencheront la fonction selected en lui passant un objet Personnage en paramètre.

Puis on codera notre fonction selected comme ceci



Pour renvoyer un objet de type Personnage en fonction de la valeur reçu tout ceci stocker dans une variable “selectedPerso” qui nous permettra d’afficher un formulaire pour modifier la valeur dynamiquement.

Puis dans notre fichier html nous modifions la structure comme ceci .



Et le css pour arriver a un rendu comme celui ci.



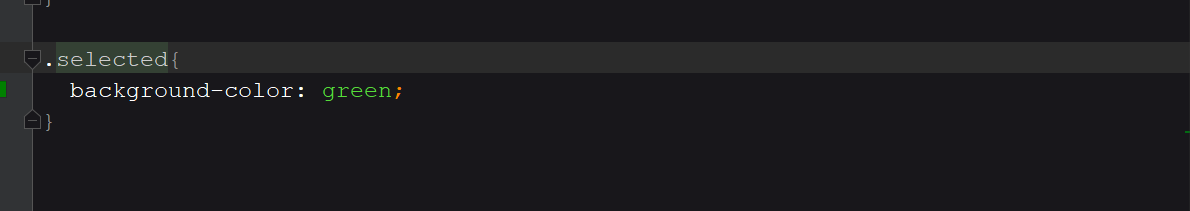
Dans l’html nous pourrons apercevoir un attribut [classe.selected]=”stark === selectedPerso”.

Ceci nous permet de dire que quand la classe de l’élément est le même que la classe de l’élément sélectionné, Angular ajoute la classe CSS sélectionnée. Lorsque les deux classes sont différents, Angular supprime la classe.

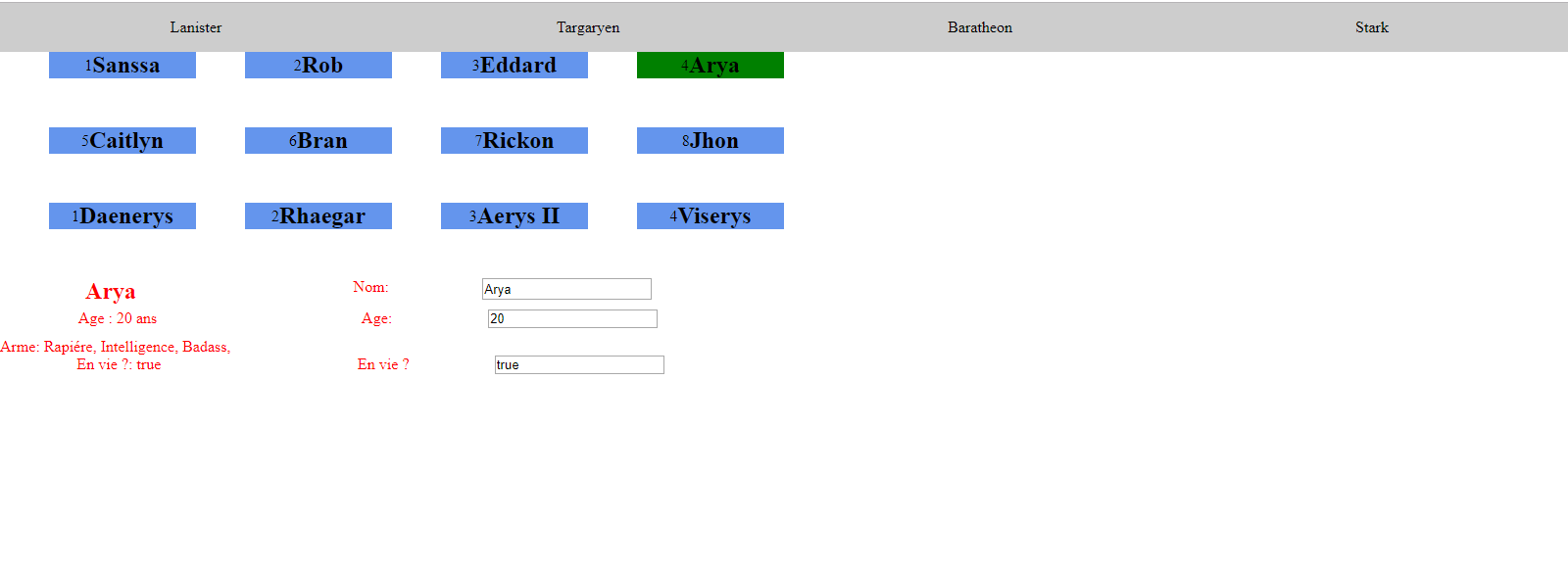
Puis nous allons afficher le formulaire si l’élément est cliqué nous aurons un objet selectedPerso donc si il existe un objet selectedPerso on affichera le formulaire grâce au ngIf de cette façon on aura



On peut s’amuser en changeant la couleur de l’élément sélectionné en effet en cliquant sur un élément on ajoute une classe et cette classe peut être designer du coup on aurait dans le css quelque chose comme ceci.



Ce qui nous donnera un rendu comme celà



# Consommer une api.

La principale utilité d’angular est la consommation de web service / api REST.

Pour cela on aura besoin de créer des services et interfaces .

Pour visualiser une api il vous faudra télécharger postman au besoin un petit logiciel sympa qui permet de faire différentes requêtes sur une api et avoir une visualisation du rendu .

Pour l’exemple on va consommer une api avec des informations de match sportif situé a cette adresse <https://bridge.buddyweb.fr/api/tutorials/matches>

On aura une base de donnée comprenant 3 match de foot, avec pour id 1 , 2 et 3

Chaque objet aura des informations concernant les équipes , le score, logo et même une vidéo.

On va créer une nouvelle app angular avec le CLI pour commencer

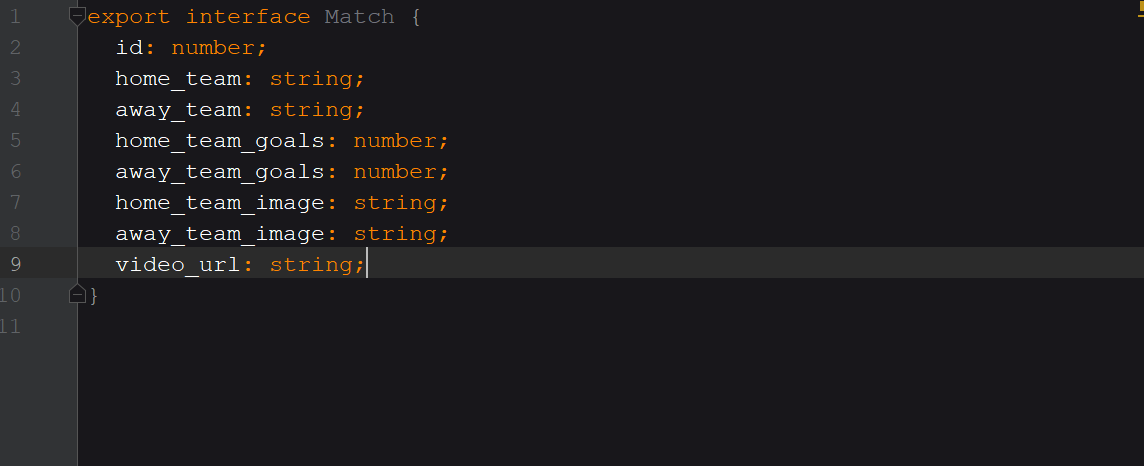
On fait un coup de ng new match-app, cd match-app et ng serve

On vide le fichier html de notre app-component

Dans notre dossier src/app on créé un nouveau dossier qu’on appellera interface.

On va avoir un fichier match.ts qui sera un peu comme la documentation d’une entité.

On récupère donc toute les propriétés d’un match et on les identifient ce qui nous donnera



Alors ce n’est pas du code en trop que d’avoir une interface, car ça permet une meilleur compréhension dans un premier temps, puis dans un deuxième temps cela permet à votre IDE d’identifier les source et donc si vous appelez une propriété qui n’est pas présente, il vous soulignera en rouge en vous stipulant qu’une erreur est présente .

On va créer ensuite notre service à l’aide du CLI on fera donc un ng g s service/match

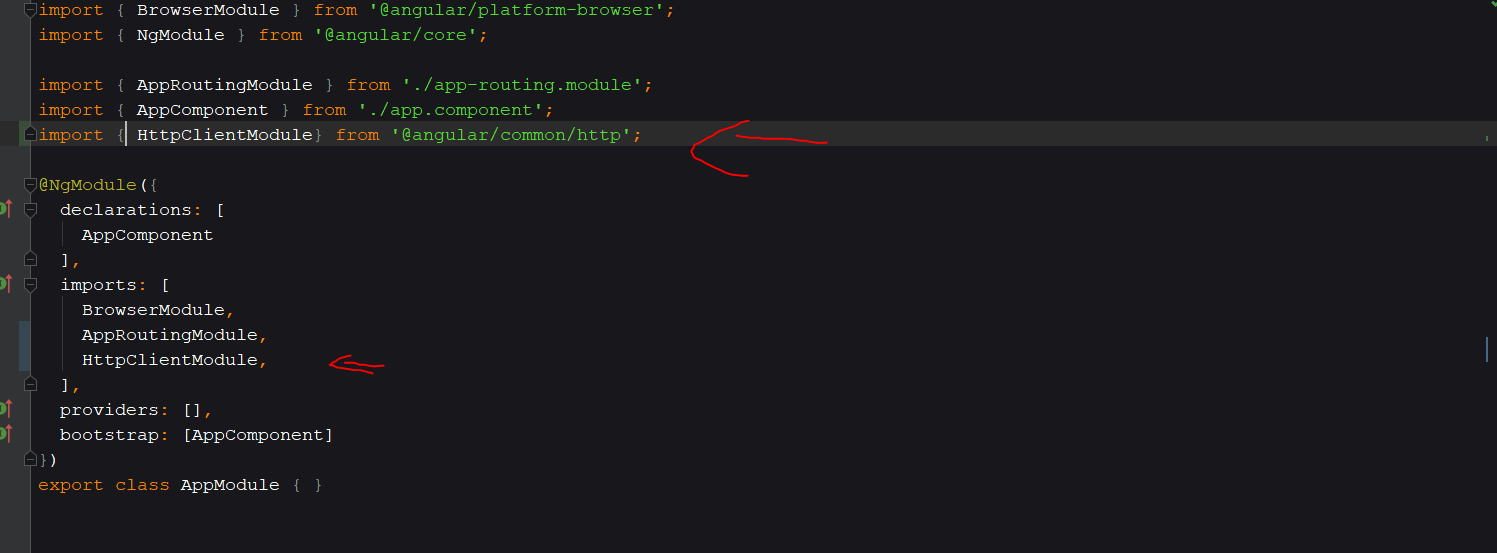
Il va nous créer un dossier services dans notre dossier app avec dedans un fichier service et un service .spec les .spec en générale servant au test ici on en servira pas .

Alors qu’est ce qu’un service, c’est un ensemble de méthode, un couteau suisse si on peut dire qui sera partager par divers composant au travers de votre application voir même par d’autre services.

Ce qui nous permet d’écrire le code qu’une fois et d'éviter la répétition on parle de DRY ( Don’t Repeat Yourself ).

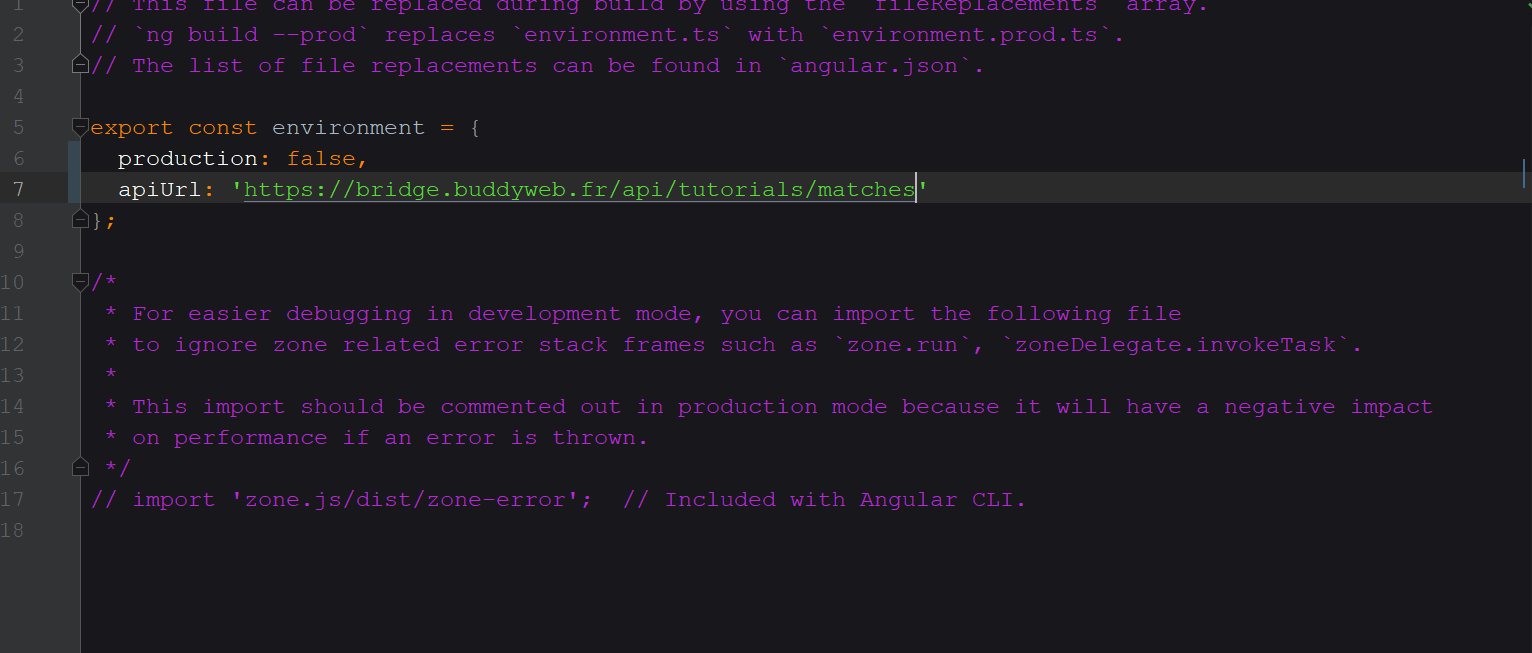
Dans match service on a besoin du module HttpClient qu’on va importer directement depuis l’app.module.ts

Pour cela il nous suffit de déclarer la librairie avec un important et de l’ajouter au tableau des imports comme ceci.



Puis une fois cela fait nous allons déclarer dans notre environnement une variable comprenant l’adresse url de l’api a ping.

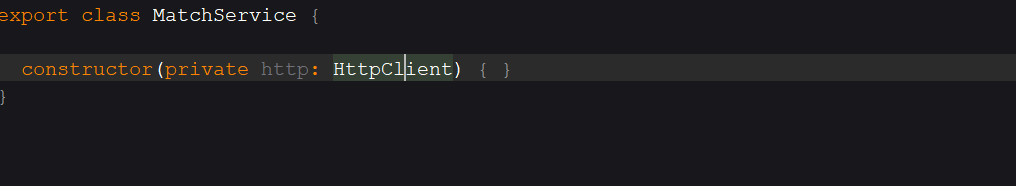
Je vais dans le dossier environnement, puis dans le fichier environement.ts je déclare ma variable de cette manière-ci.



On est prêt à coder nos méthodes ! Retournons dans notre service.

Première chose à faire c’est d’injecter notre httpClient dans notre service.

Pour cela dans le constructor on initialise notre variable.

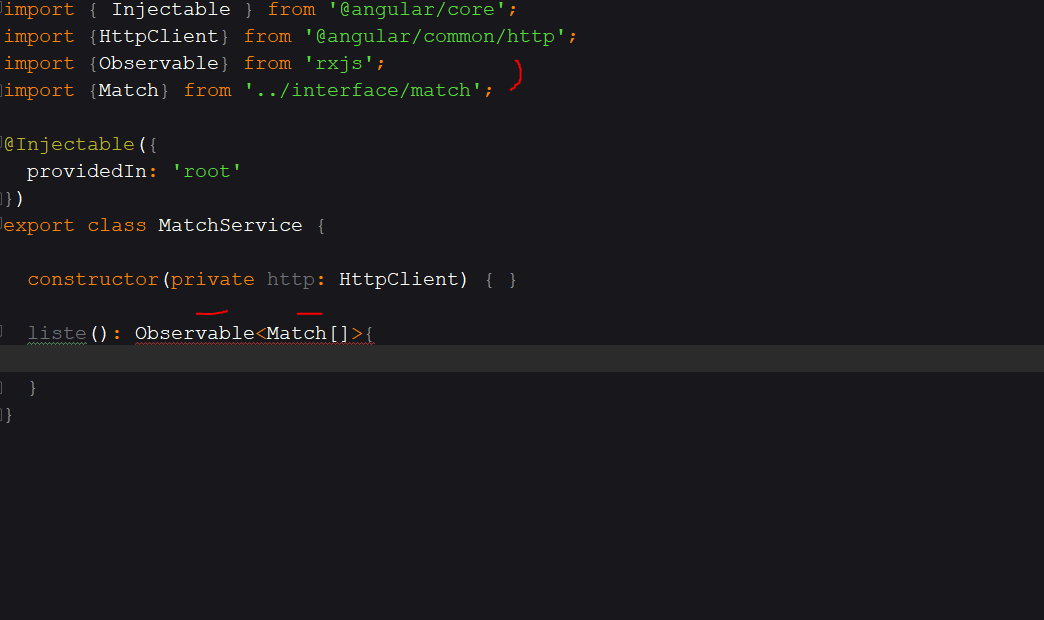


Puis à la suite on va pouvoir coder notre méthode list puis aprés show.

On va commencer par la fonction list qui va nous renvoyer un observable.

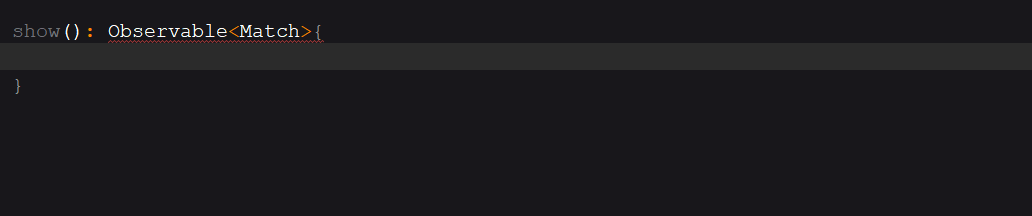
Un observable est une sorte de promise, pour cela on doit importer rxjs si notre IDE ne nous le fait pas automatiquement.

Et on lui dit que notre observable retourne un Array de match

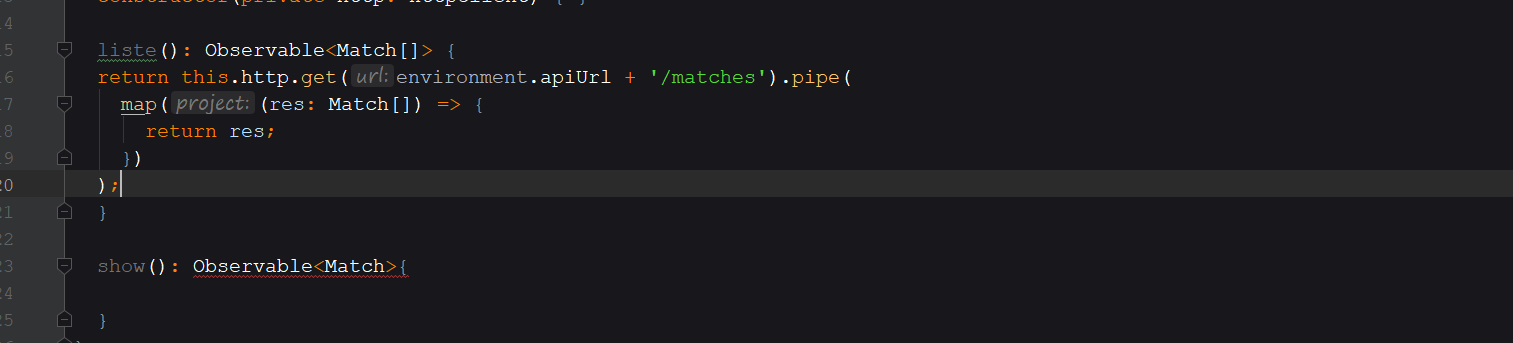


On pensera bien également à importer notre interface

Pour la fonction show c’est exactement la même chose a l'exception prés qu’on ne renvoie pas un Tableau mais juste un objet



Pour ce qui est de compléter la méthode list on fera simplement

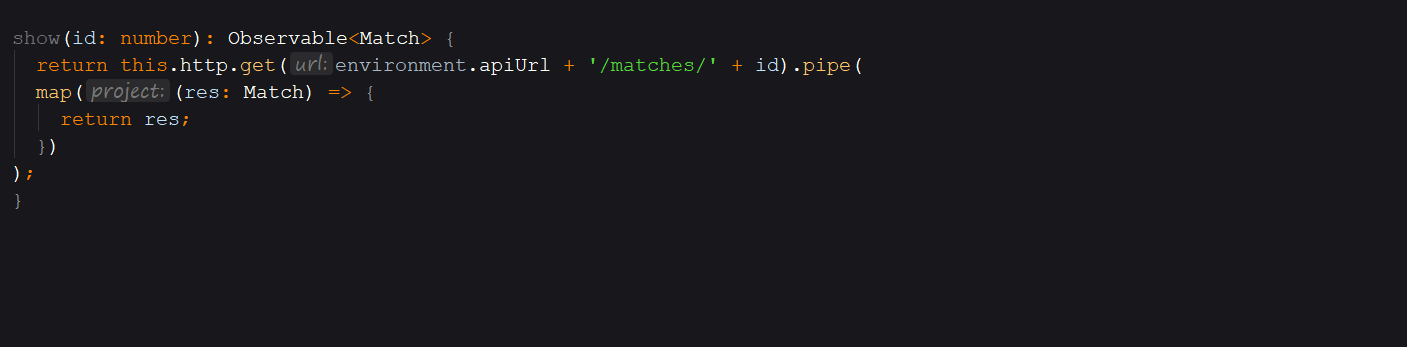


Décortiquons un peu cela, on return une méthode get de notre librairie http dans laquelle on passe notre url ( variable d'environnement) concaténé avec matches pour récupérer notre liste. La syntaxe devient un peu particulière on utilisera le .pipe c’est bizarre mais on s’y fait dans laquelle on appellera l'opérateur map rxjs

( Pour info un opérateur est un :Un opérateur permet de définir un Observable à partir d'un autre en y **appliquant quelques transformations**.)

qui nous retournera une réponse de type Match[].

Évidemment pour la fonction show c’est presque pareil a 2 différences près cette fois ci, elle prendra un paramètre et ne renverra pas un Match[] mais simplement un Match[].

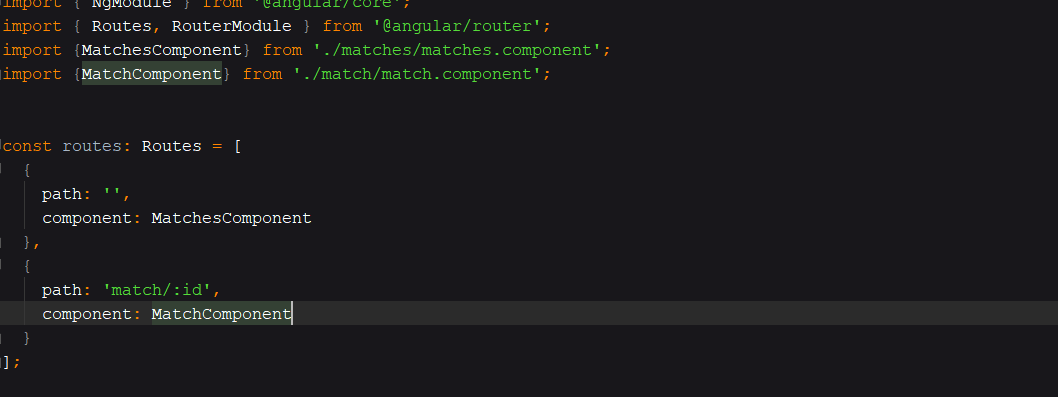


Maintenant direction app-routing.module.ts pour ajouter nos premières routes.

Pour cela nous avons une constante route[] prête à recevoir des informations sous forme de json.

Notre première route sera de déclarer à la racine notre composant Matches,

Puis notre deuxièmes routes sera le composant Match qui lui se trouvera à match/:id pour passer notre identifiant en paramètre de notre méthodes.

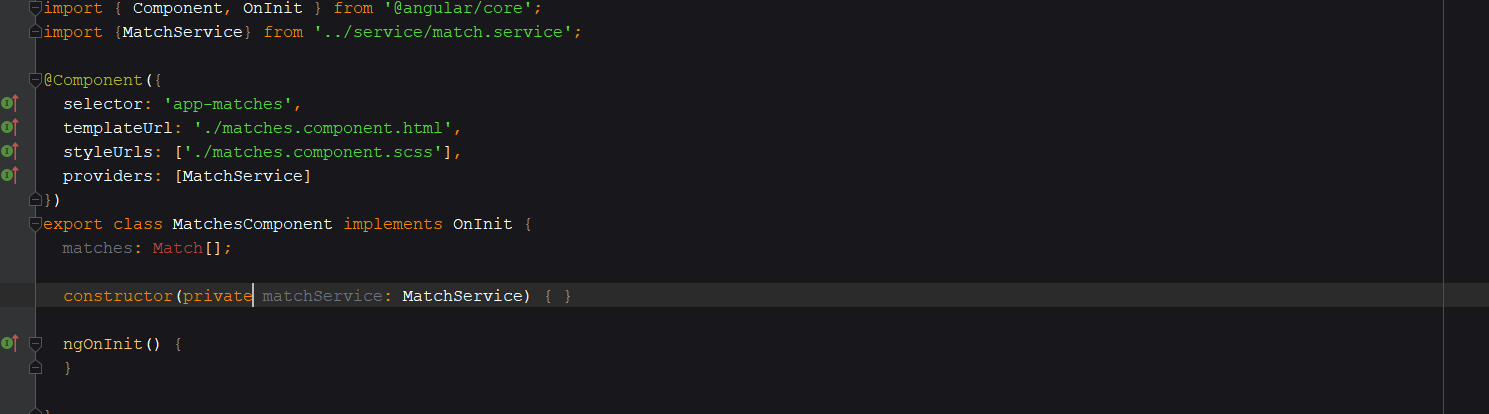


On vérifie bien que nos libraires sont correctement importer.

c’est ce fichier qui appellera les bon composants en fonction de l’url appelée.

Pour vérifier si tout fonctionne il nous suffit juste d'insérer <router-outlet></router-outlet> dans notre app.component.html.

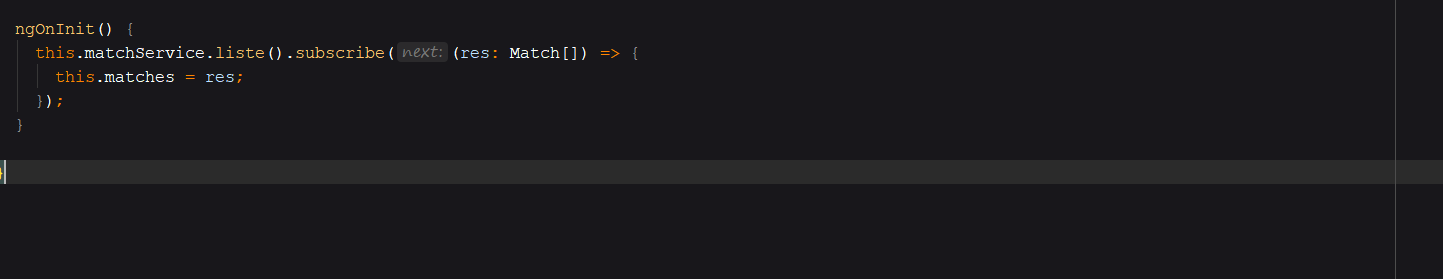
Maintenant dans notre Matches Components, on va injecter notre service et utiliser la méthode liste de cette maniére-ci.



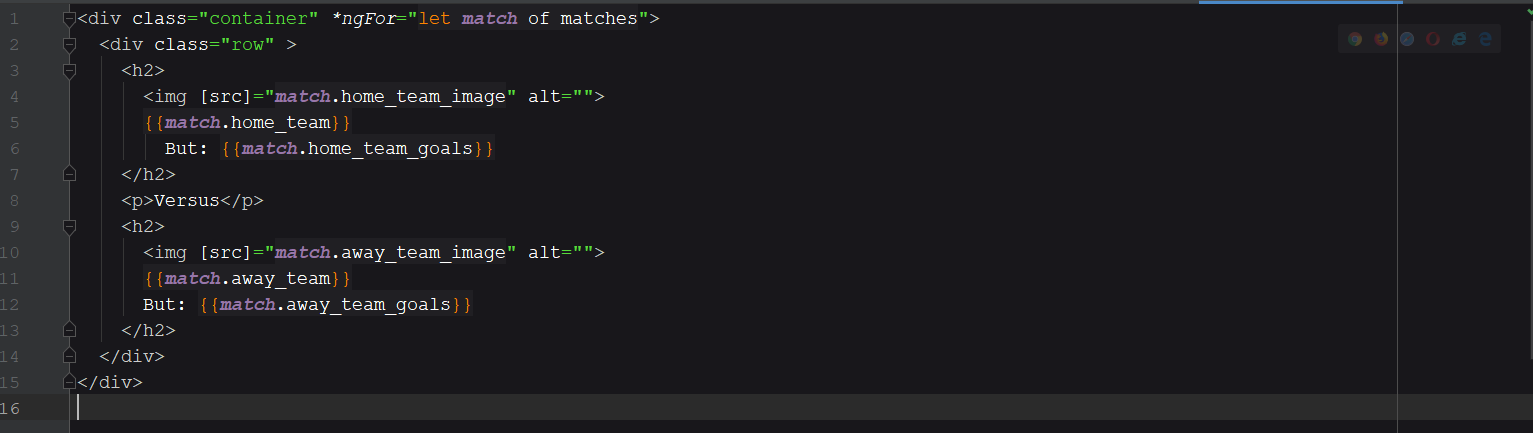
Le service est importer, stocker dans une variable.

Et un tableau est initialisé pour recevoir nos informations, attention de bien importer les librairies .

ngOnInit comme son nom le signifie est une fonction qui exécute des chose au chargement du composant, c’est ici qu’on utilise notre méthode directement et qu’on stockera le résultat dans notre variable.



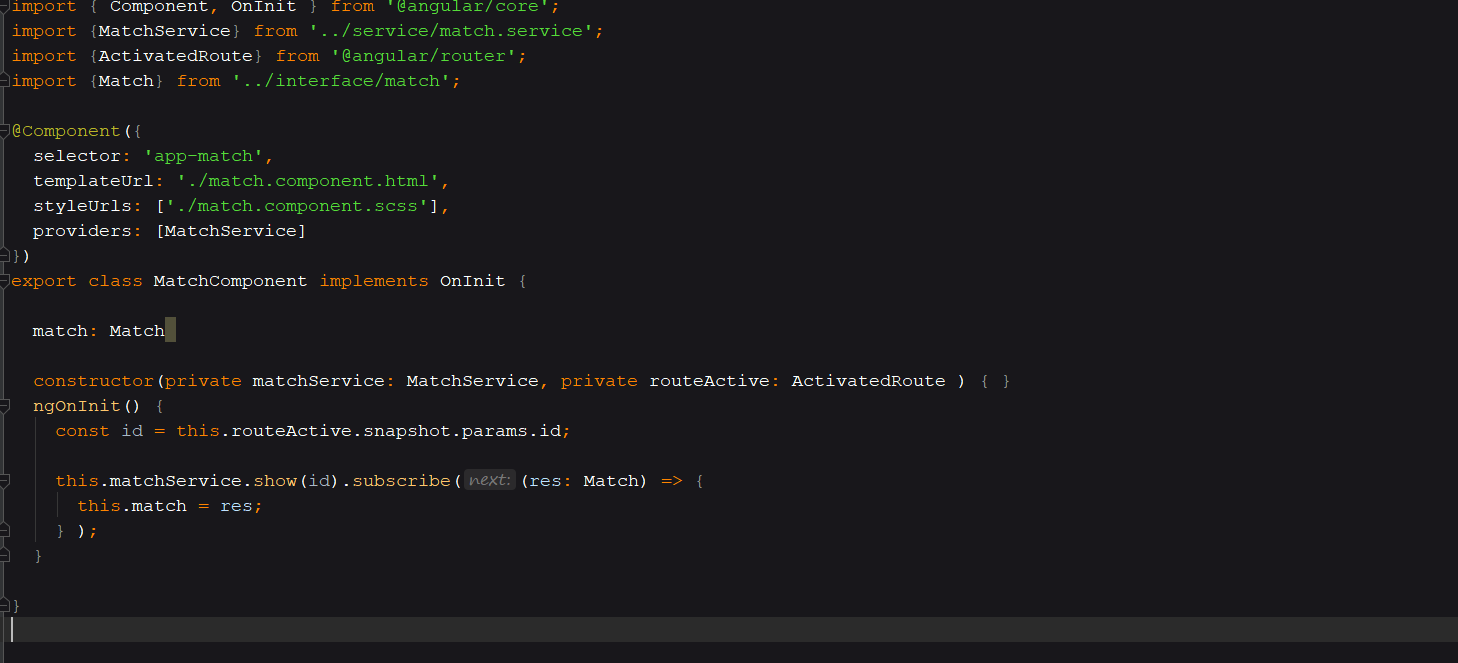
Puis dans notre vue il nous suffira de boucler comme on a pu voir jusqu'à maintenant .



On mettra l’attribut src directement entre [] ce qui nous permettra de faire appelle directement à notre variable.

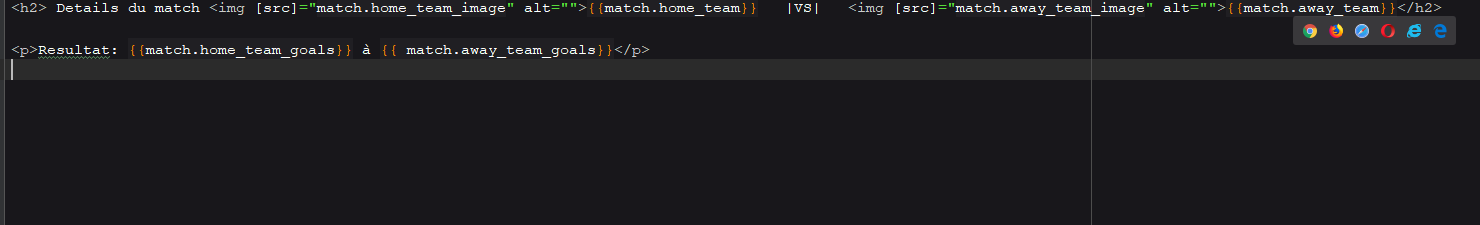
Maintenant on fait le même système dans le match.component.ts.

On injecte notre service de la même manière que précédemment.



La différence ici c’est que nous avons besoin de l’id passé en paramètre pour récupérer celui ci il nous faudra utiliser la librairie ActivatedRoute pour récupérer le params dans le snapshot.

Puis dans l’html structuré de façon à avoir un rendu viable:



Et voilà .