**Etape 0 :** installer nvm (node version manager) <https://github.com/coreybutler/nvm-windows/releases>

**Etape 1 :** installer node.js (environnement d’exécution js) <https://nodejs.org/fr>

 permet d’instaler node.js et npm



**Etape 2 :** vérifier l’installation de npm (permets d’installer des packages js)



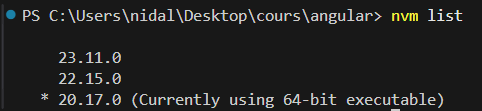
**Exécuter un fichier js dans la console :**

1. créer un fichier hello.js et écrire dedans 

2. Exécuter 

**Etape 3 :** installer angular

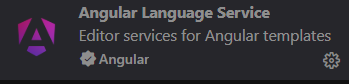
 ne pas utiliser la dernière version de node, mais la 20 plutôt

Permet de vérifier la version utilisé par angular

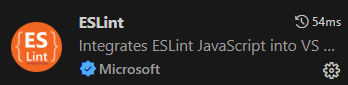
 Installer angular

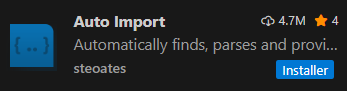
 permet de vérifier la version d’angular

**Extension VS code à installer :**









**Créer un nouveau projet angular**

 Créé un nouveau projet nommé « playing-cards » dans le répertoire courant du terminal

Voici les réponses que l’on donne à la suite de la création d’un nouveau projet

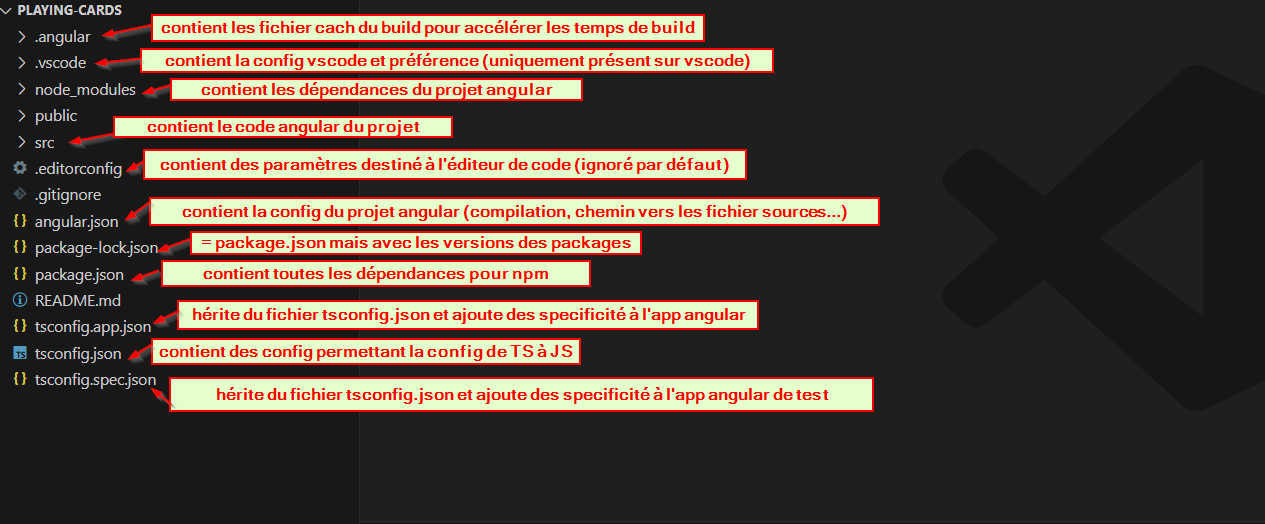


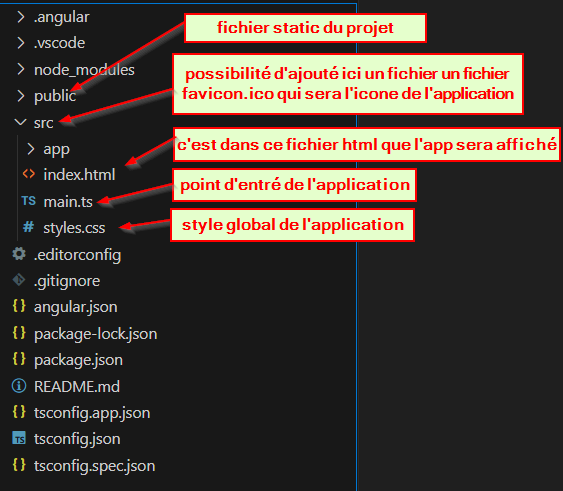
**Lancer le serveur du projet**

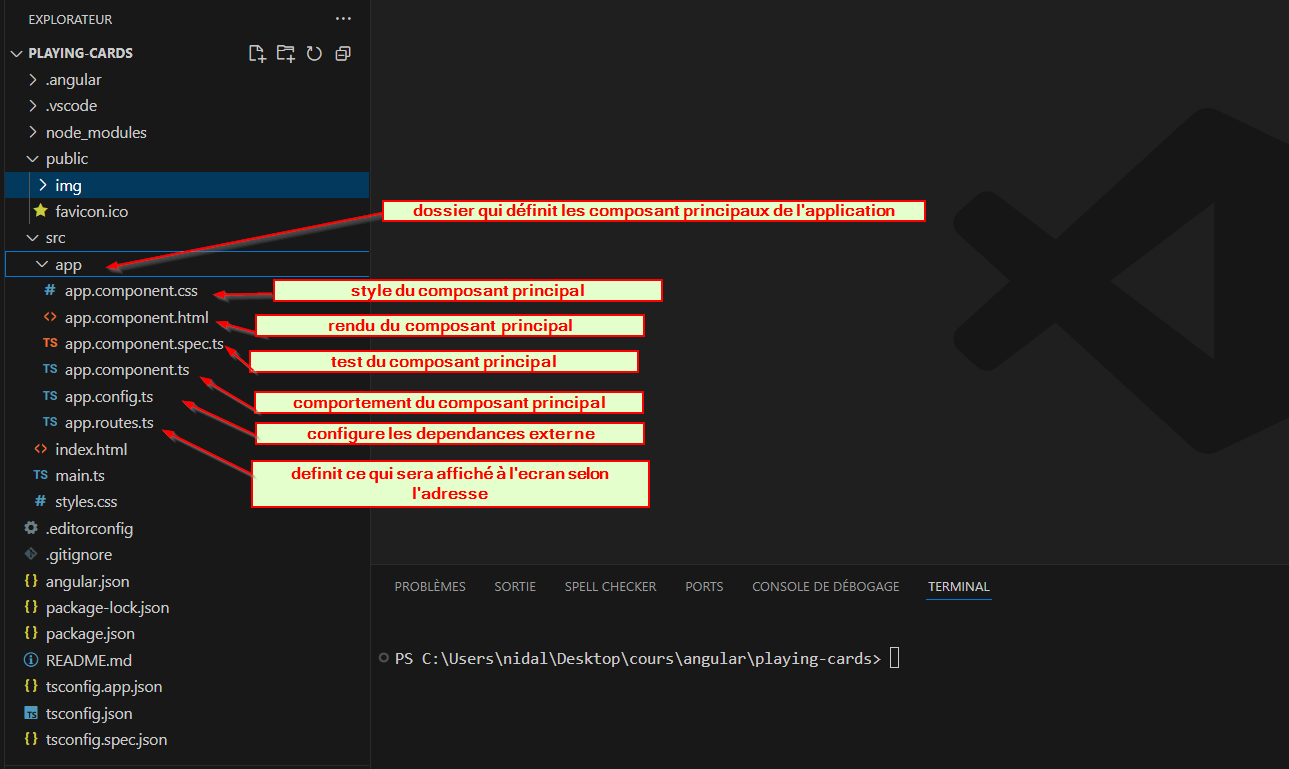
Ouvrir le dossier du projet dans un IDE puis



**Description des fichiers du projet**

****

****

****

**Comment accéder au fichier static du projet stocké dans le dossier public**

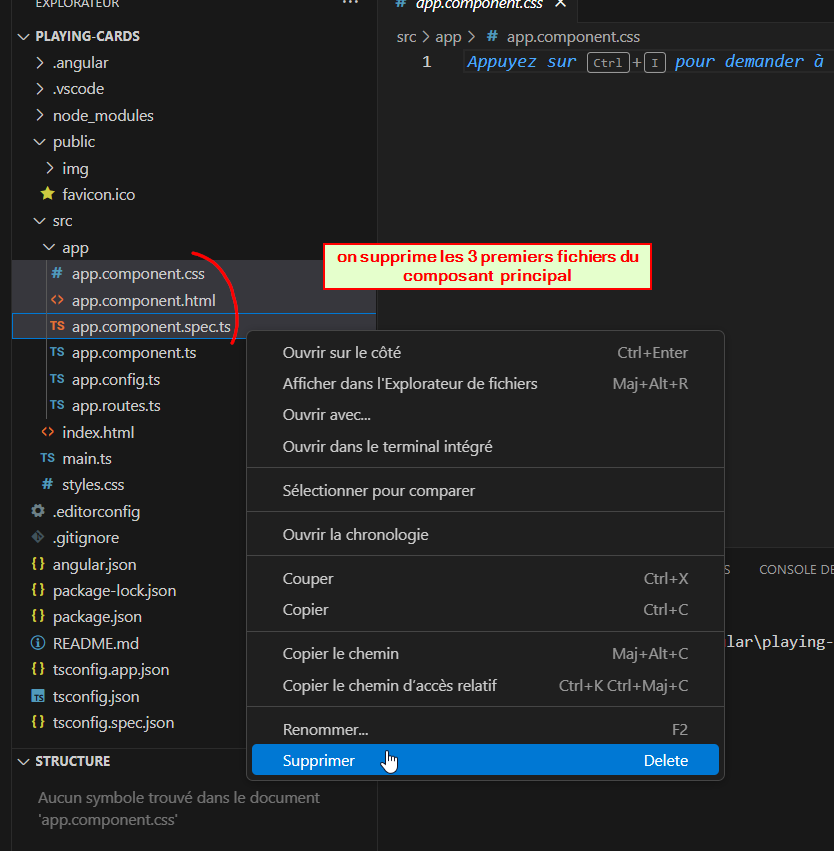
****

**Un composant :**

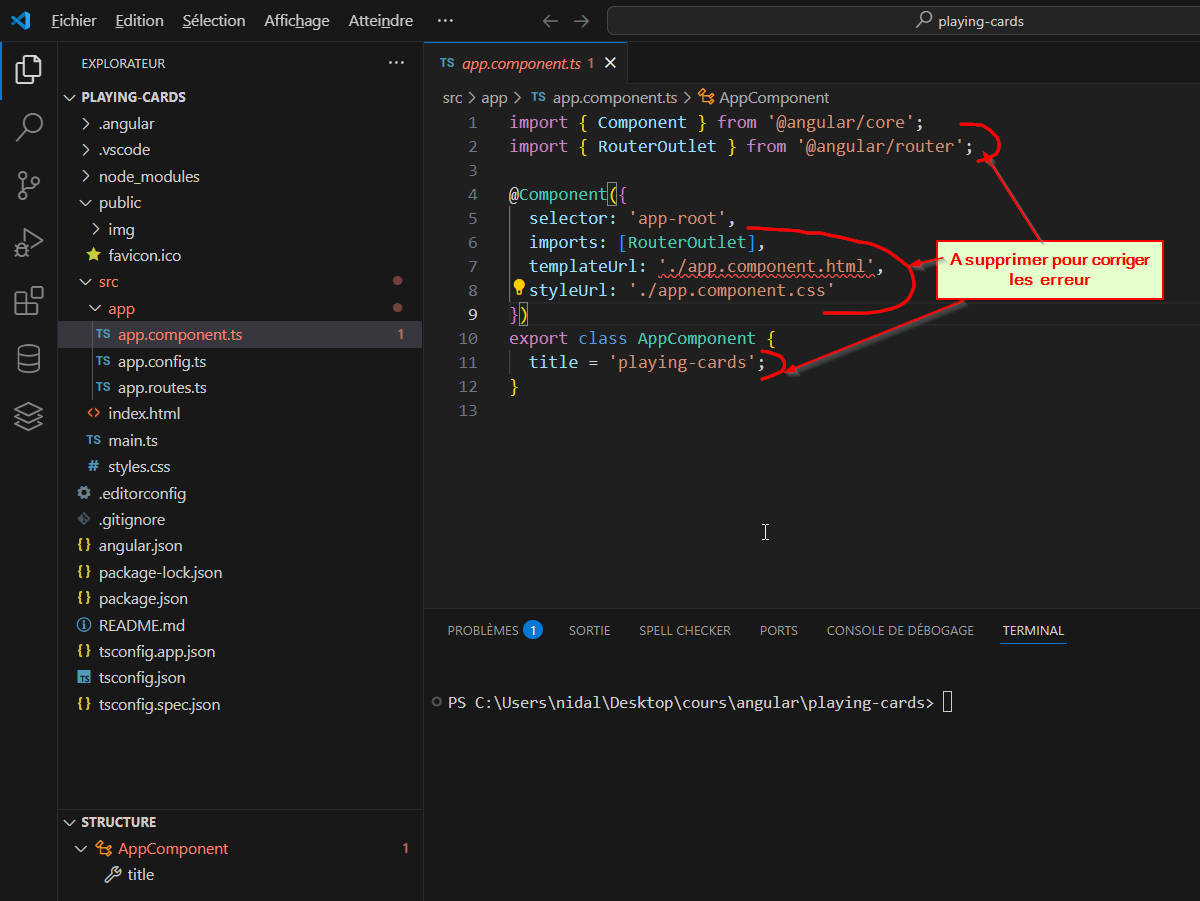
C’est une brique de notre application qui s’occuper d’un aspect bien spécifique de notre projet et contient:

* Le style (.css)
* Le rendu (.html)
* Le comportement (component.ts)
* Les test (spec.ts)

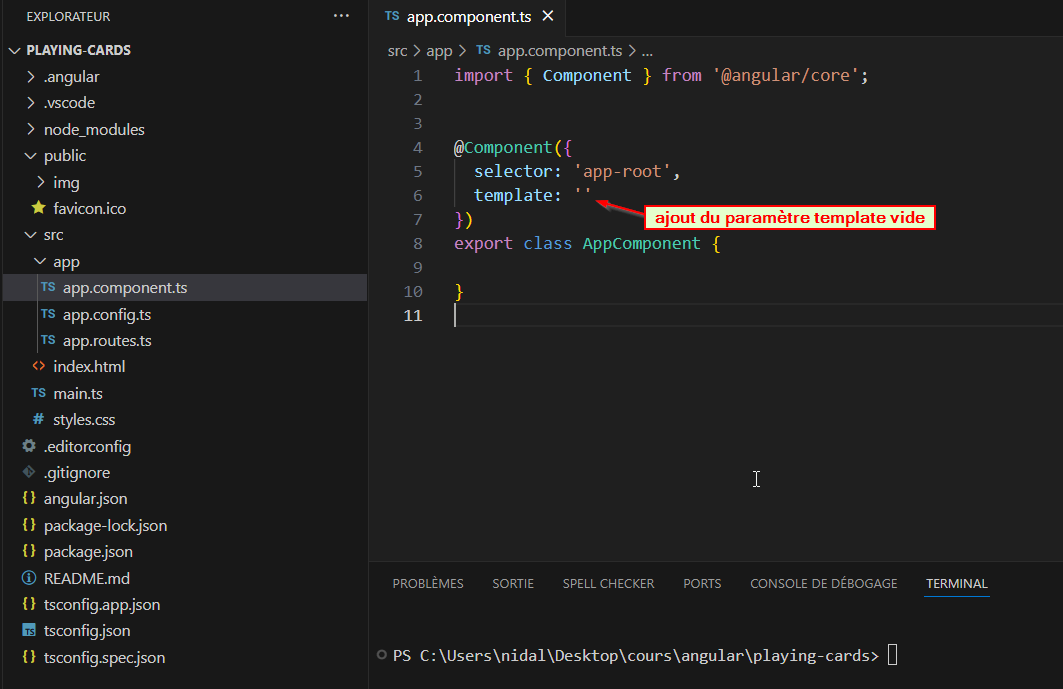
**On supprime les 3 premiers fichier du composant principal**

****

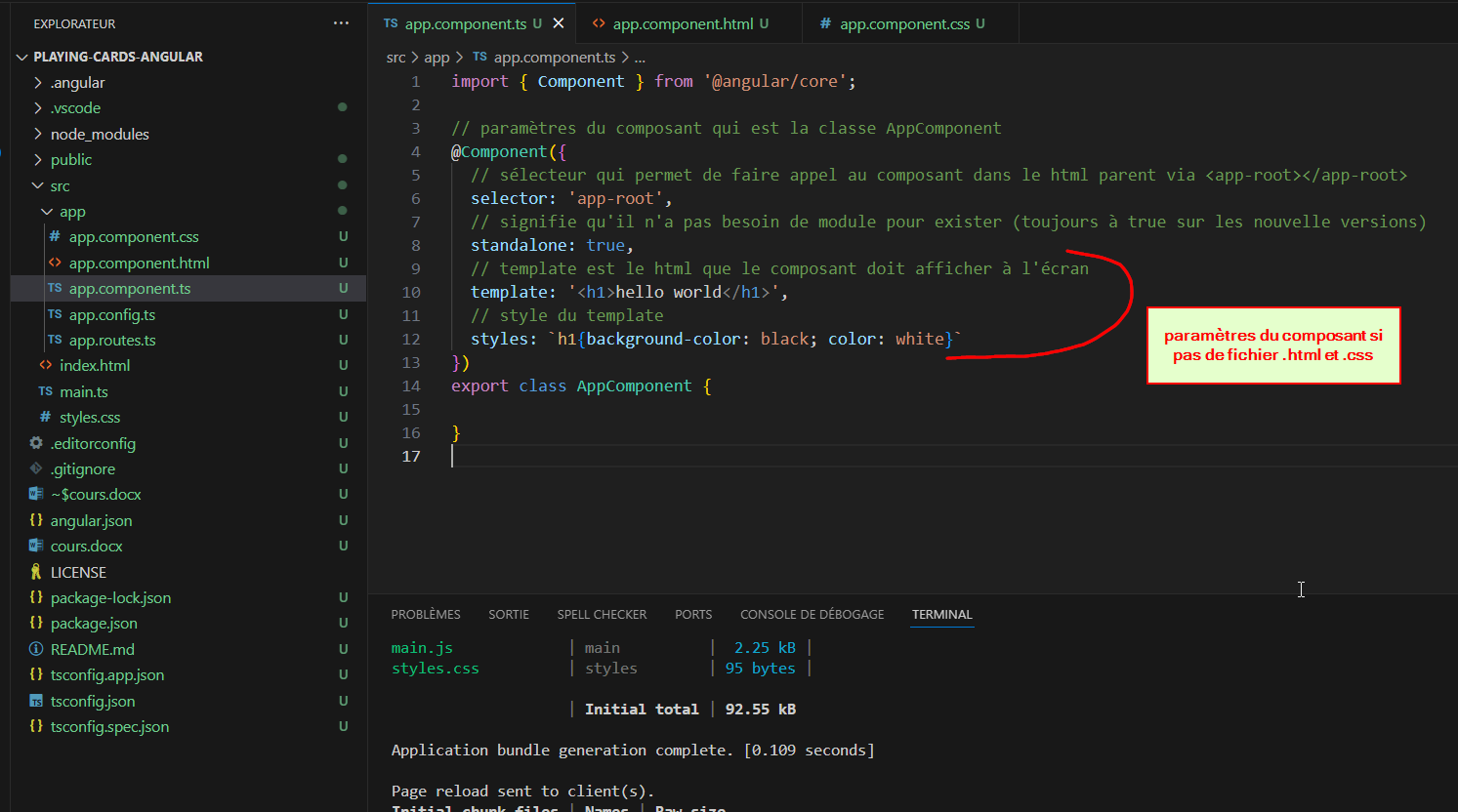
**Après suppression des 3 fichiers du composant principal, se rendre dans app.component.ts et supprimer ceci pour corriger les erreurs**

****

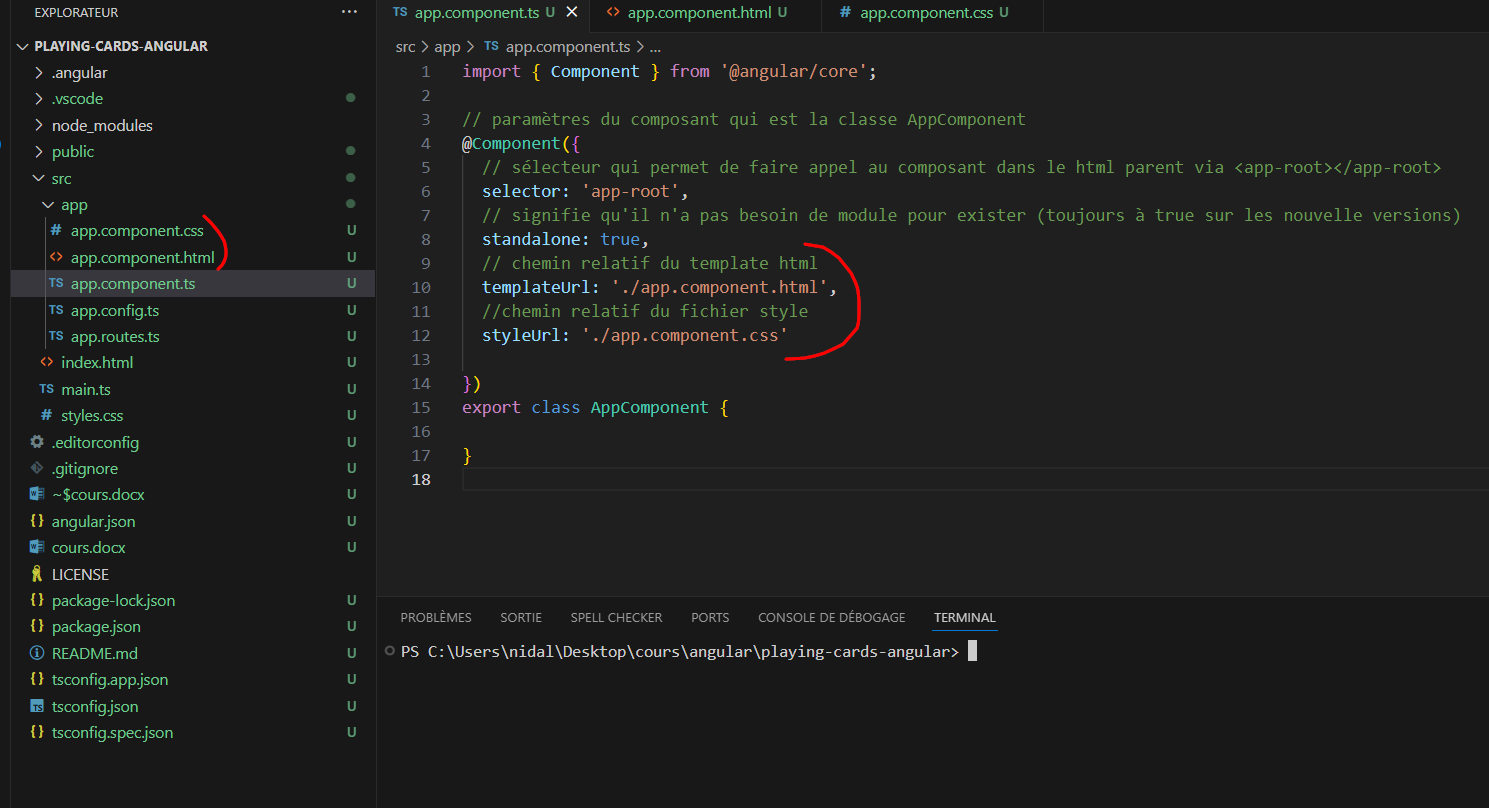
**Puis ajout du paramètre template vide au @Component**

****

**Voici les paramètres du fichier app.component.ts si pas de fichier .html et .css**

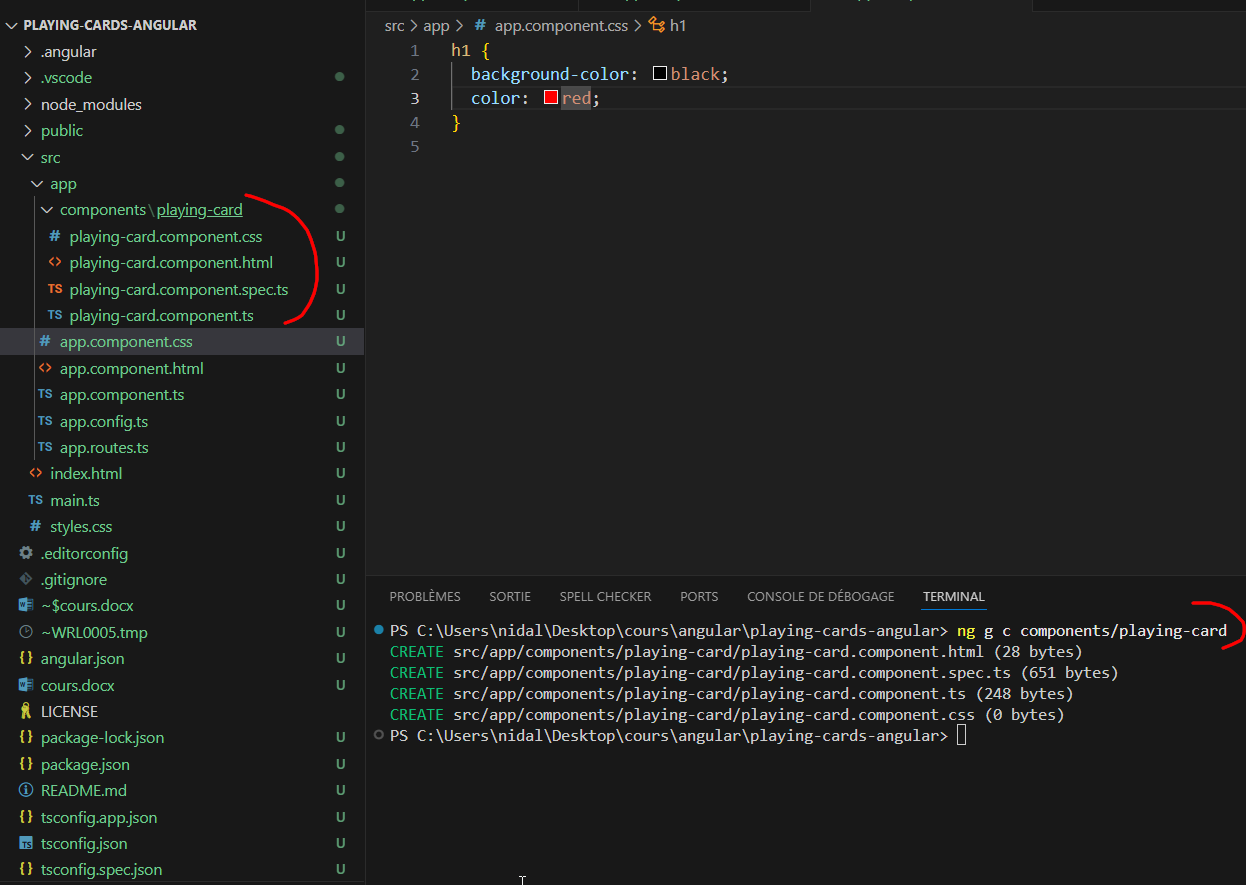
****

**Si on ajoute dans le dossier app/ les fichier app.component.css et app.component.html voici les paramètres à utiliser**

****

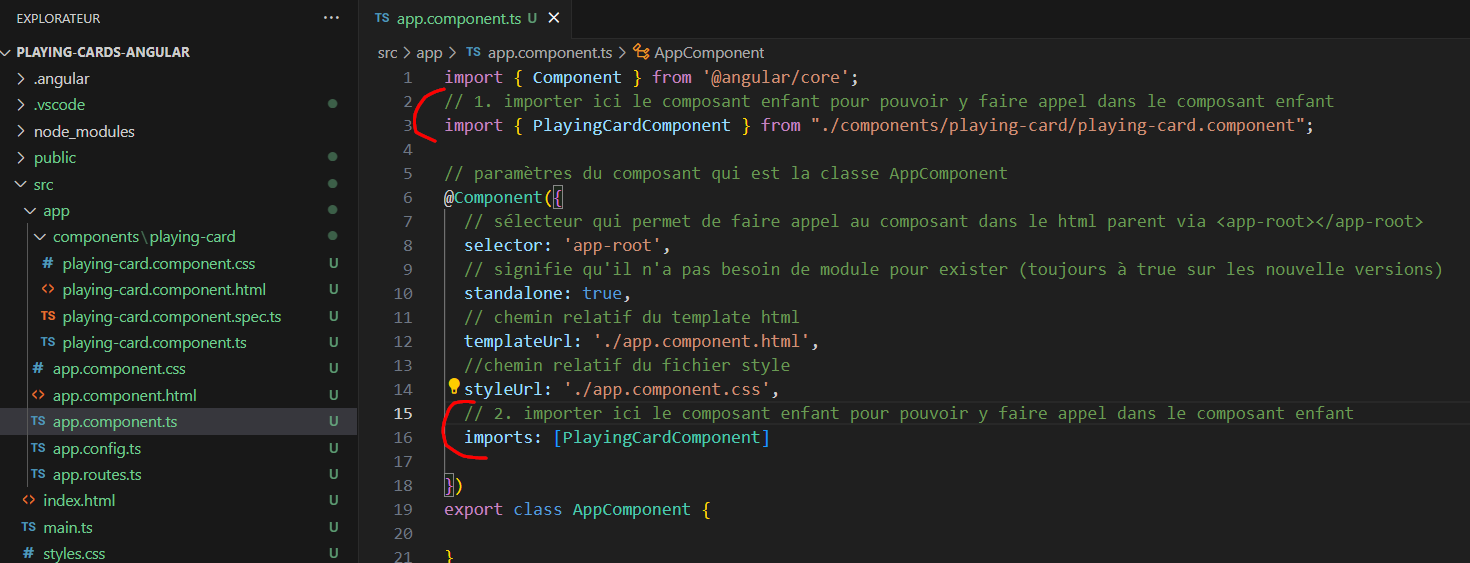
**Créer un nouveau composant**

**** Commande raccourci permettant de créer un nouveau composant nommé « playing-card » se situant dans le dossier components

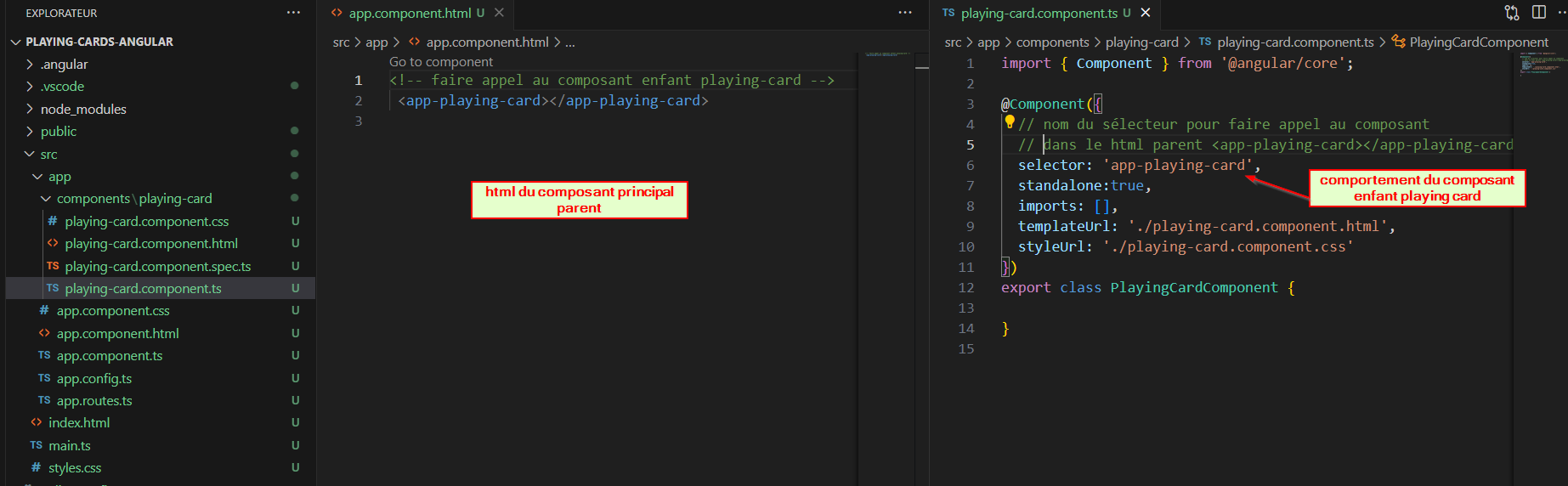
****

**Faire appel à au composant enfant playing-card dans le composant principal parent**

(Se fait automatiquement) mais d’abord importer le composant enfant dans le composant parent principal app.component.ts

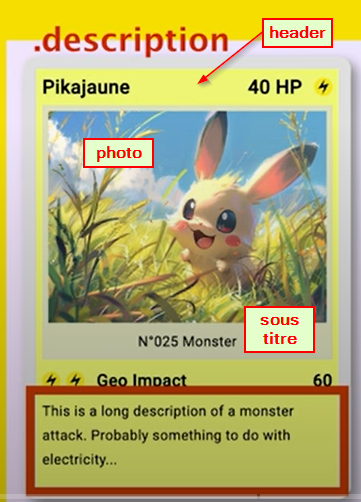
****

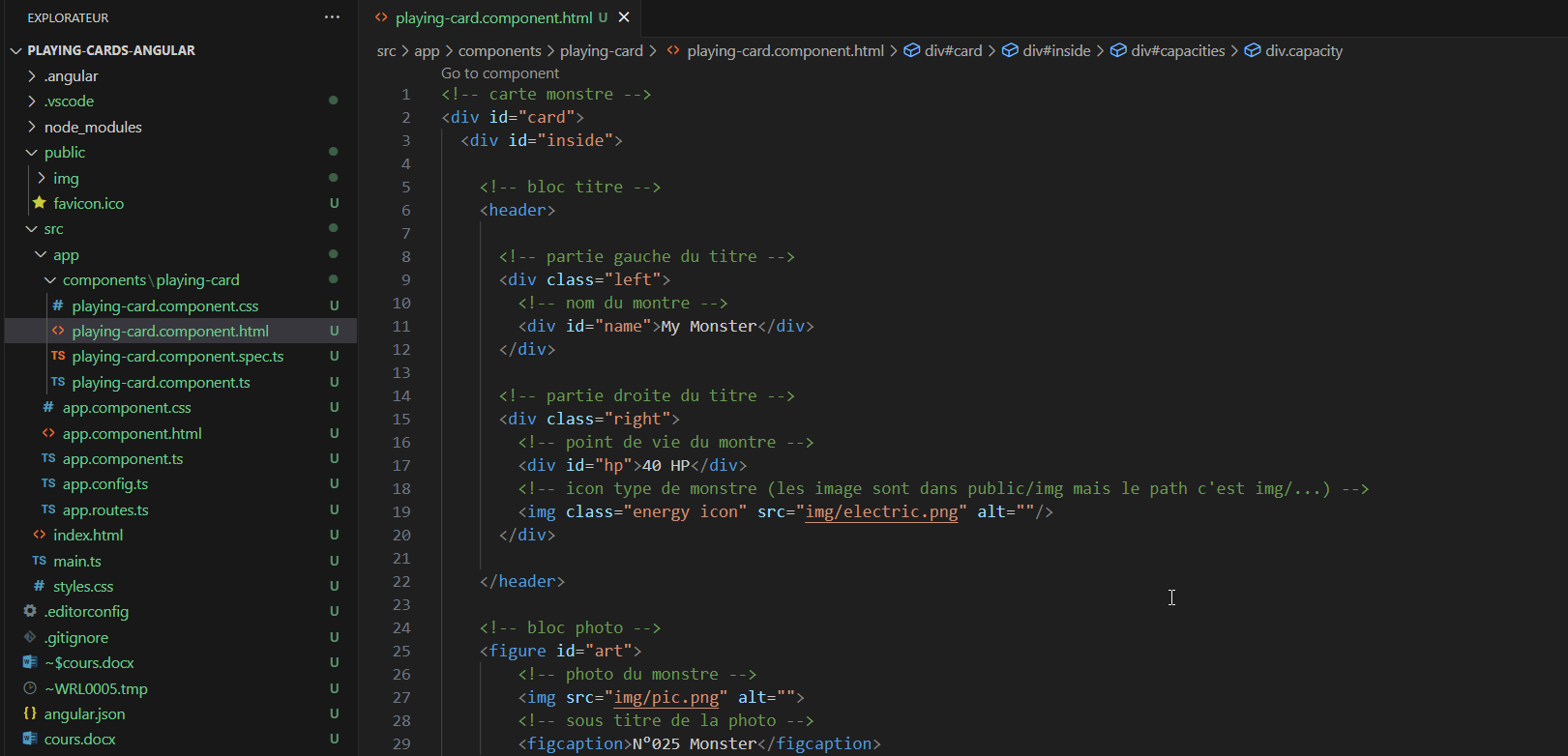
**Puis utiliser la baliser html du sélecteur du composant enfant**

****

**Modifier playing-card.component.html pour y insérer une card**

Tout d’abord créer la partie html : ici nous avons mis en place un template de carte pokemon

****

****

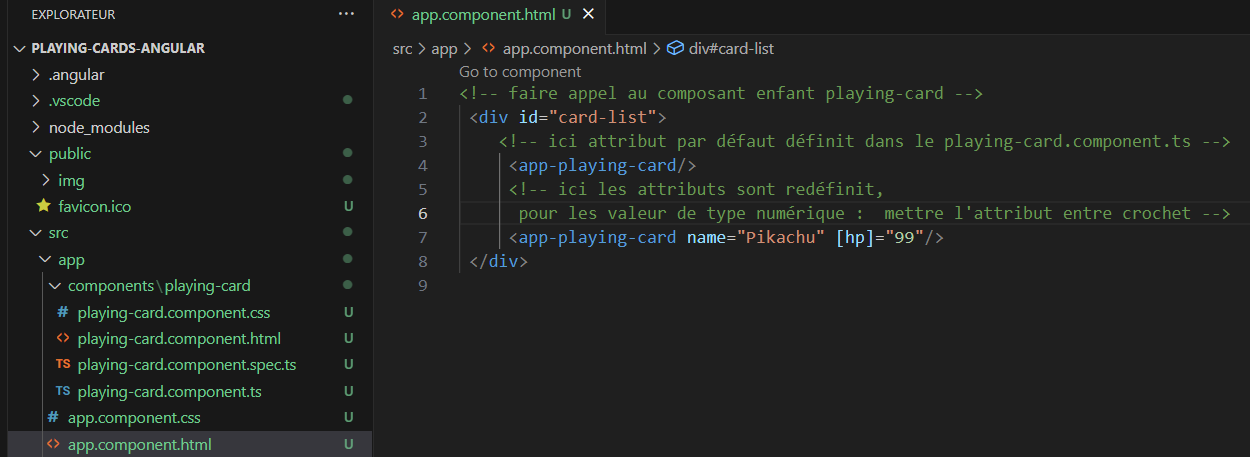
**Puis construire le style de la card html dans playing-card.component.css**

**Puis nous avons la possibilité de modifer le fichier style.css qui a étendu global (font-family, background de la page)**

**On initialiser les attributs d’un monstre dans la classe PlayingCardComponent du module playing-card.component.ts**

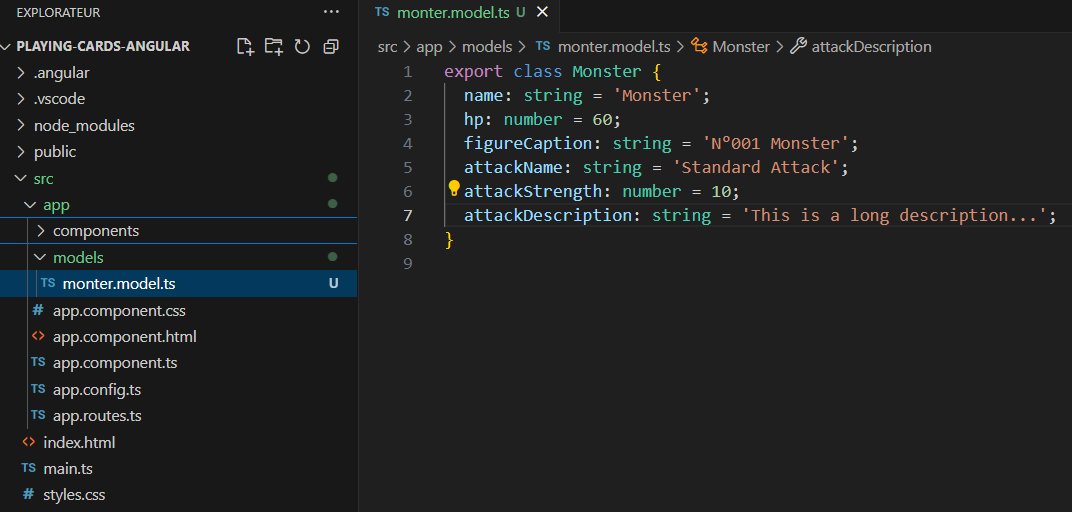
****

**Pour utiliser les propriétés qu’il y a dans le playing-card.component.ts on fait appel aux attribut de la classe PlayingCardComponent entre accolade dans le html du composant principal parent tout en ayant la possibilité de les redéfinir avec les Input() html**

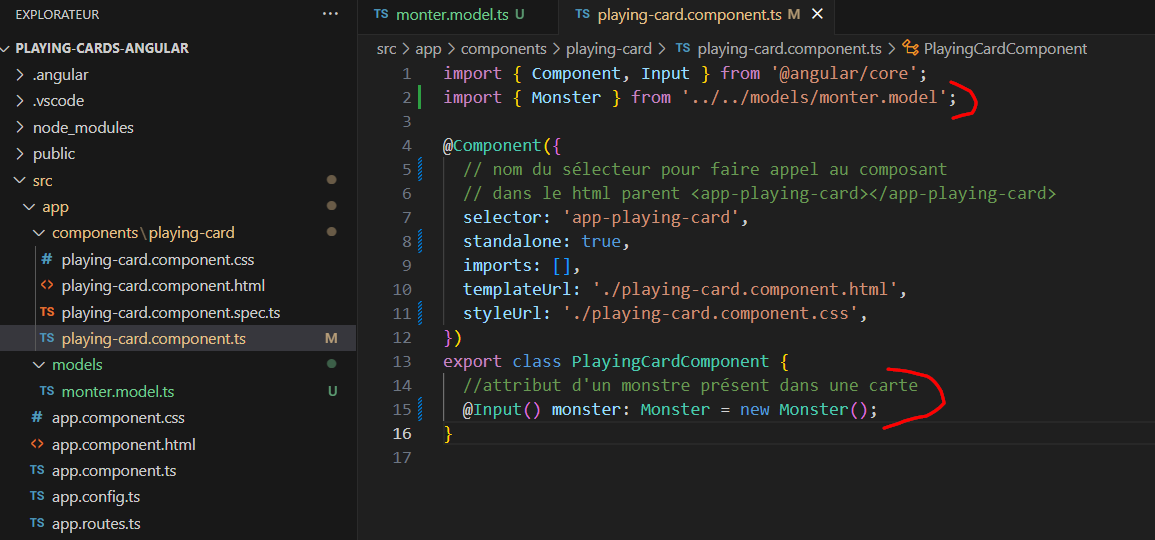
****

**Pour créer un nouvel model Monster**

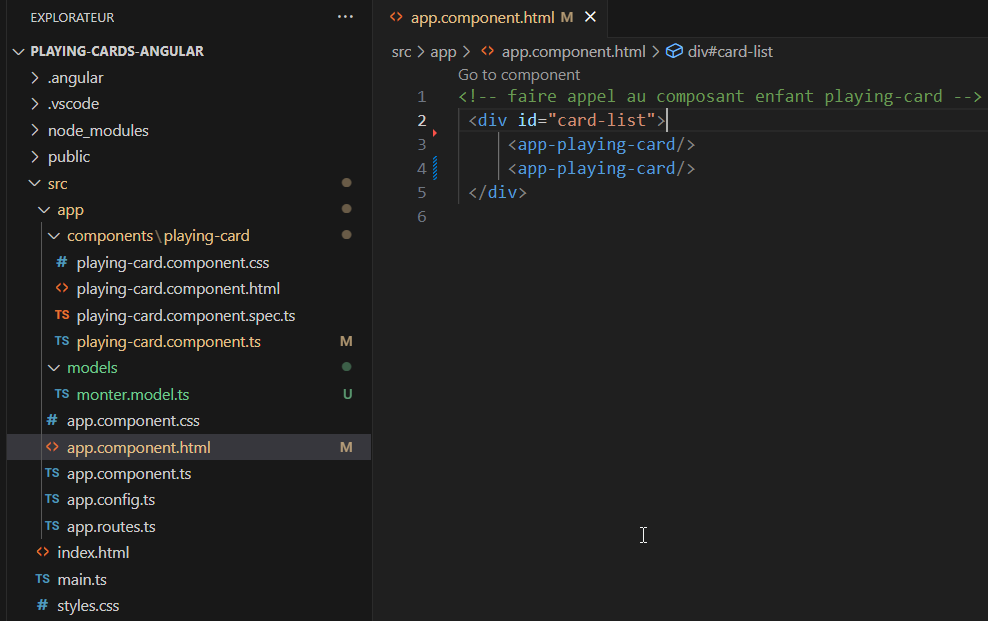
**Créer un nouveau dossier nommé « models » dans lequel on va créer un module monster.model.ts**

****

**Puis dans playing\_card.component.ts on instancie un Monster au lieu d’afficher des attributs qui ne sont pas en rapport avec la class PlayingCardComponent**

****

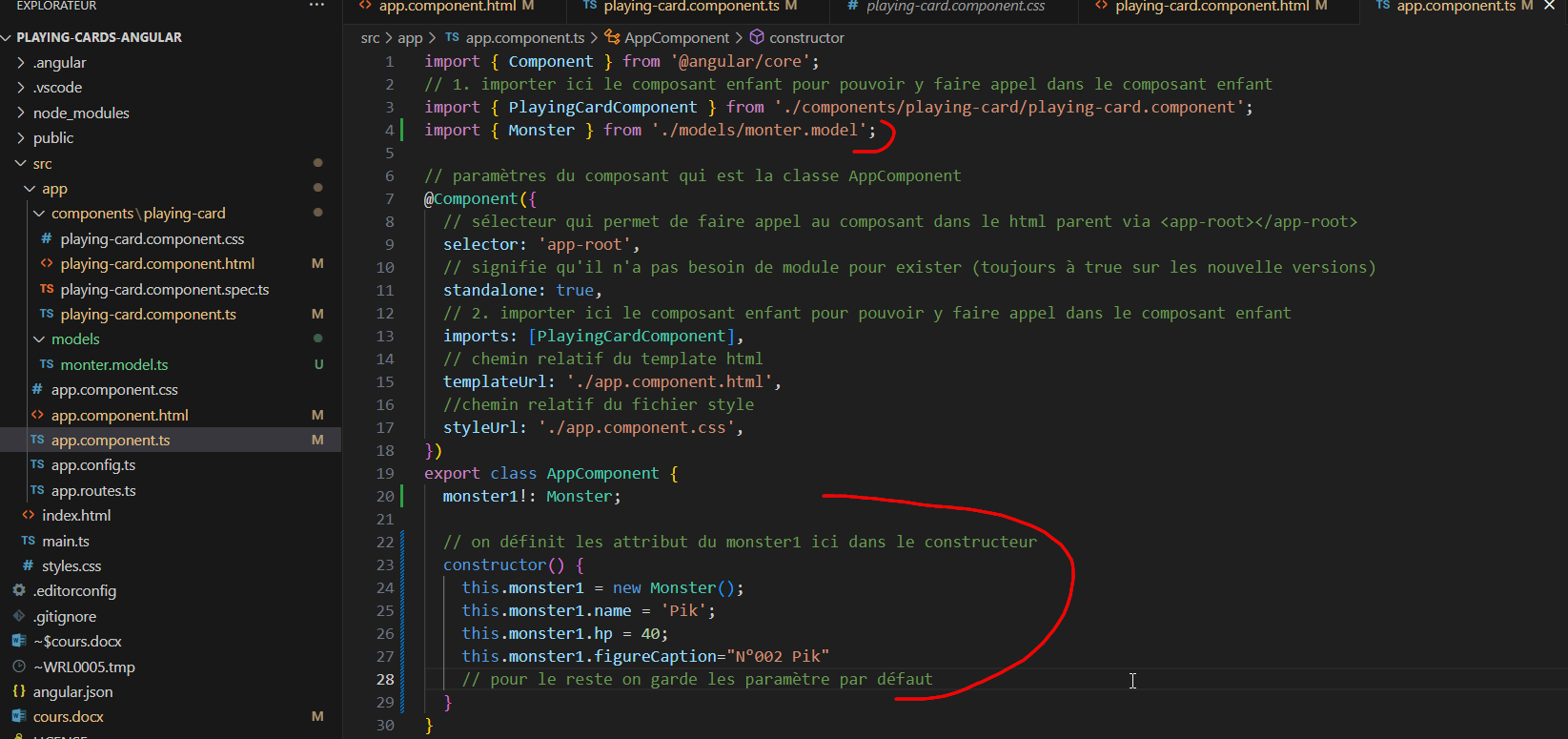
**Il est nécessaire de supprimer les input html dans le app.component.hmtl car les attribut ne sont plus disponible**

****

**Puis enfin supprimer les attributs de playing-card.component.html qui ne sont plus disponible**

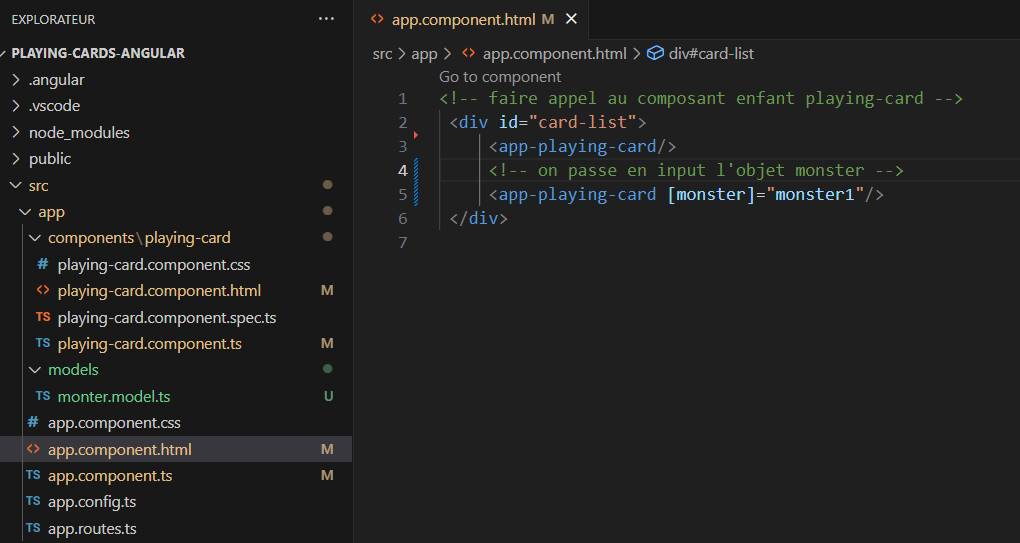
****

**Dans app.component.ts on va initialiser et définir un nouveau monster que l’on pourra passer en paramètre du app.component.html**

****

Le « ! » signifie que l’indique à ts que l’on va initialiser monster1 plus tard dans le code : dans le constructor

**Dans app.component.html on passe en paramètre monster1**

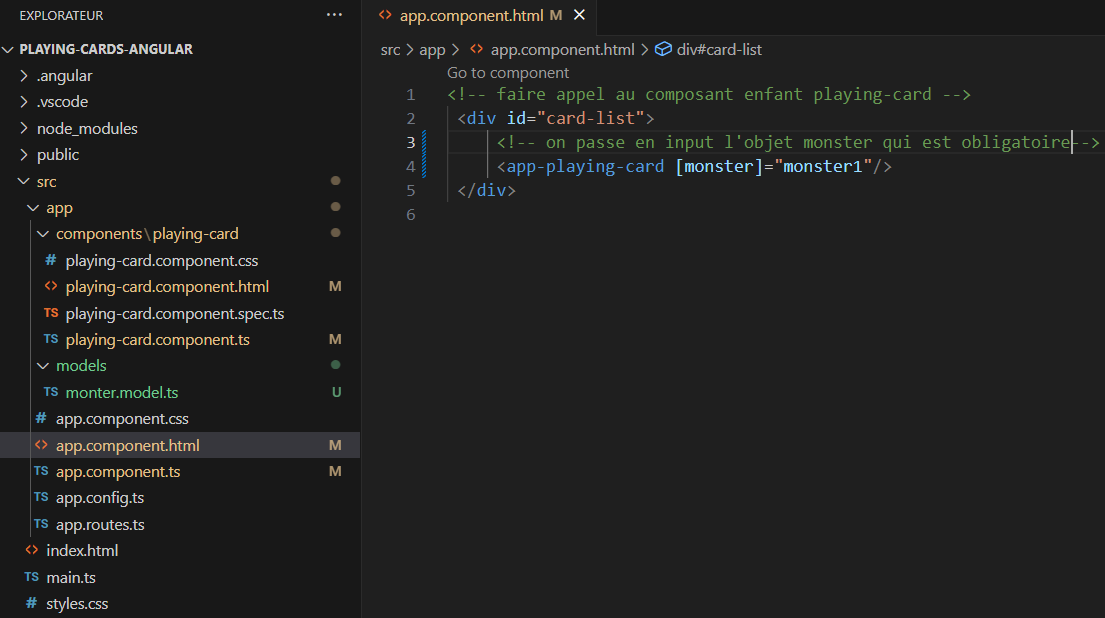
****

**On a aussi la possibilité de définir que l’argument « monster » est obligatoire dans le html parent app.component.html**

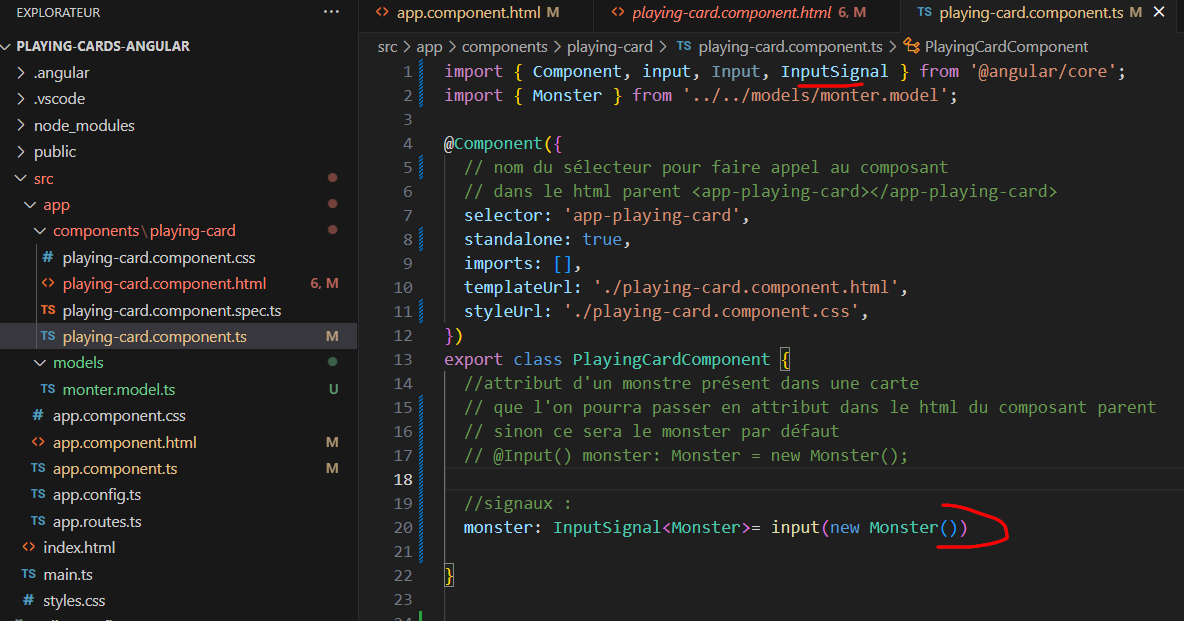
* **Se rendre dans playing-card.component.ts et ajouter le paramètre « required :true » dans le Input()**

****

**On supprime dons les balise html dans app.component.html qui n’ont pas de paramètre monster définit car cela provoque maintenant une erreur**

****

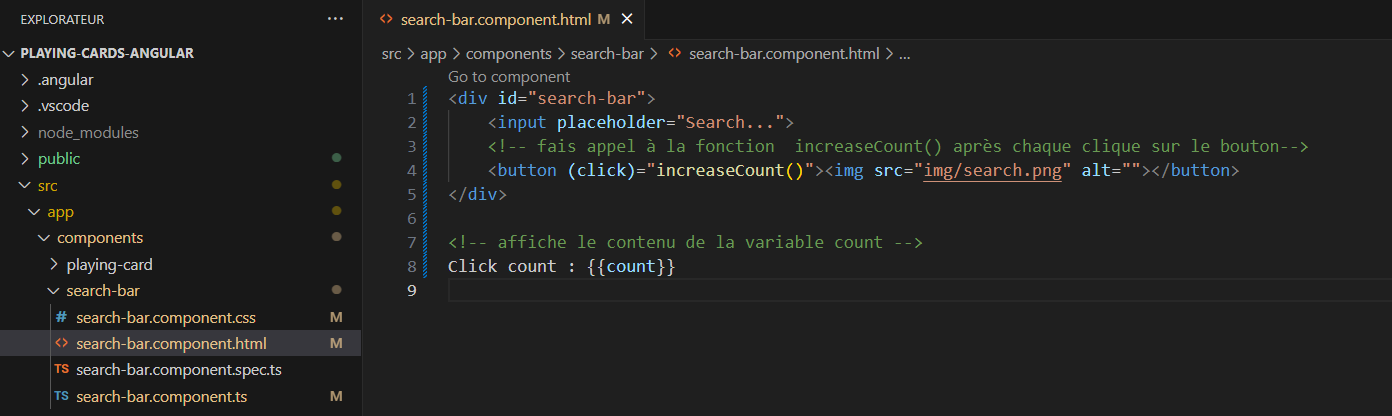
**Les signaux à la place de Input() plus récent et plus performant**

****

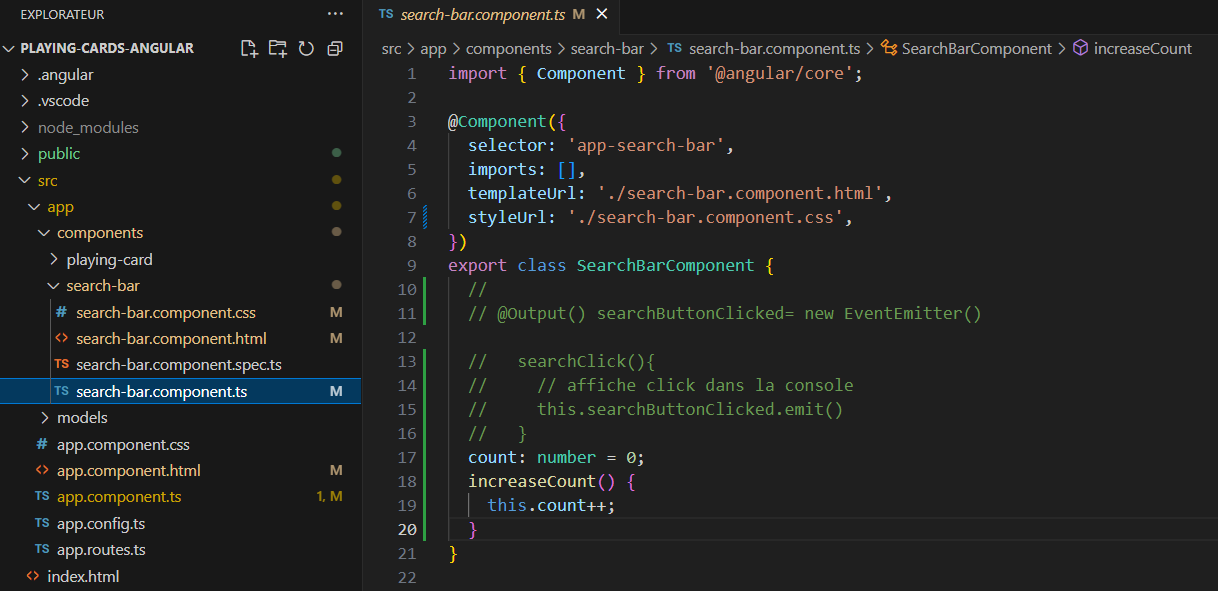
**Si on a un signal dans le component, dans le html du component courant, il faut ajouter des parenthèses après le nom de l’objet**

****

**Ecouter un évènement click dans le html sur un élément puis faire appel à une fonction (ici increaseCount() )**

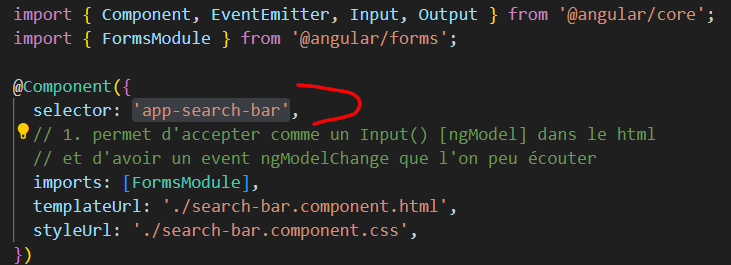
****

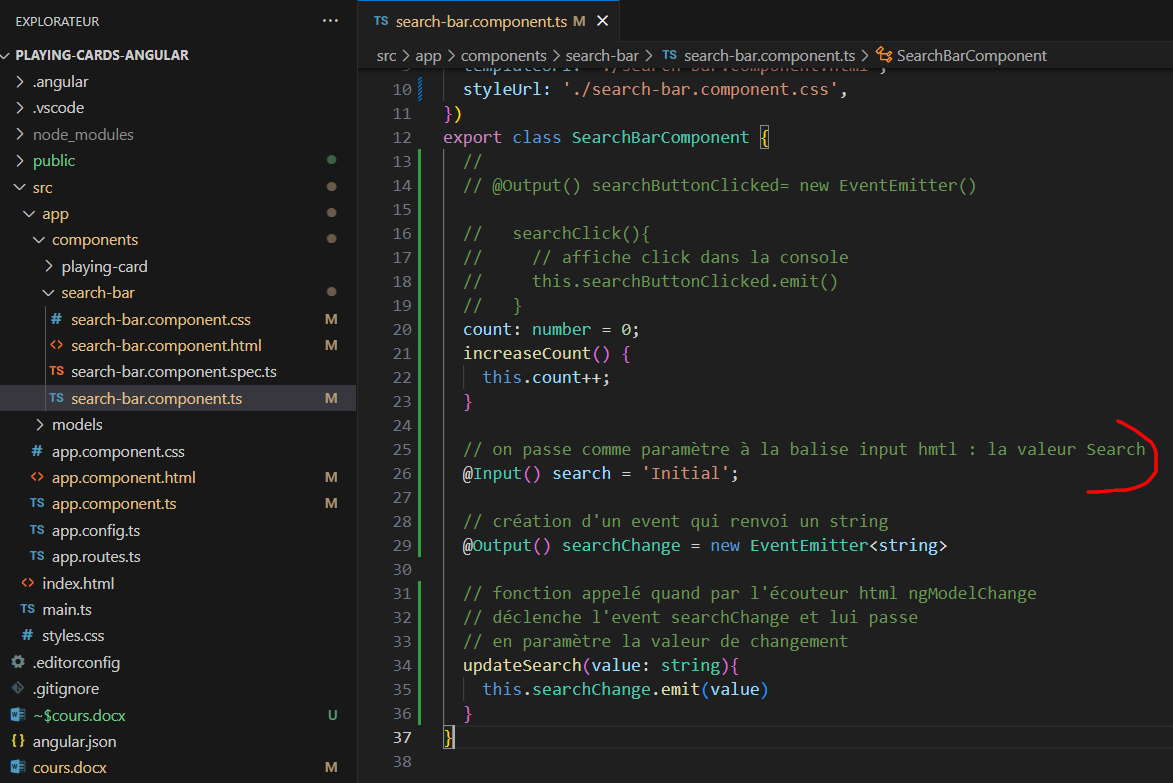
**Créer la fonction dans search-bar.component.ts**

****

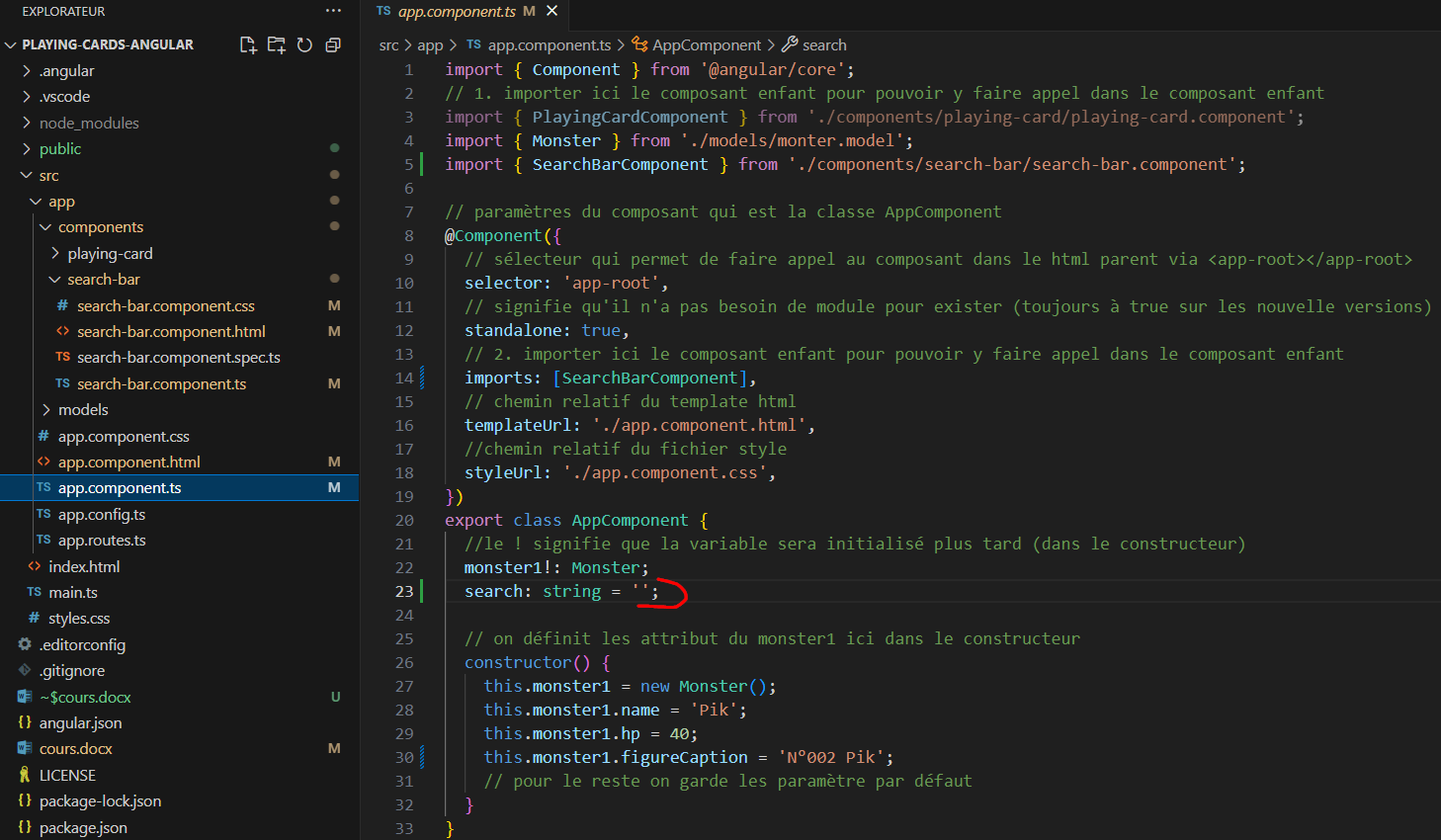
**Input() : créer une variable Input() dans un composant permet de que dans le html du composant parent on puisse lui passer un argument**

1. on crée la variable Input() dans le playing-card.component.ts (le sélecteur du composant se nomme 'app-search-bar')

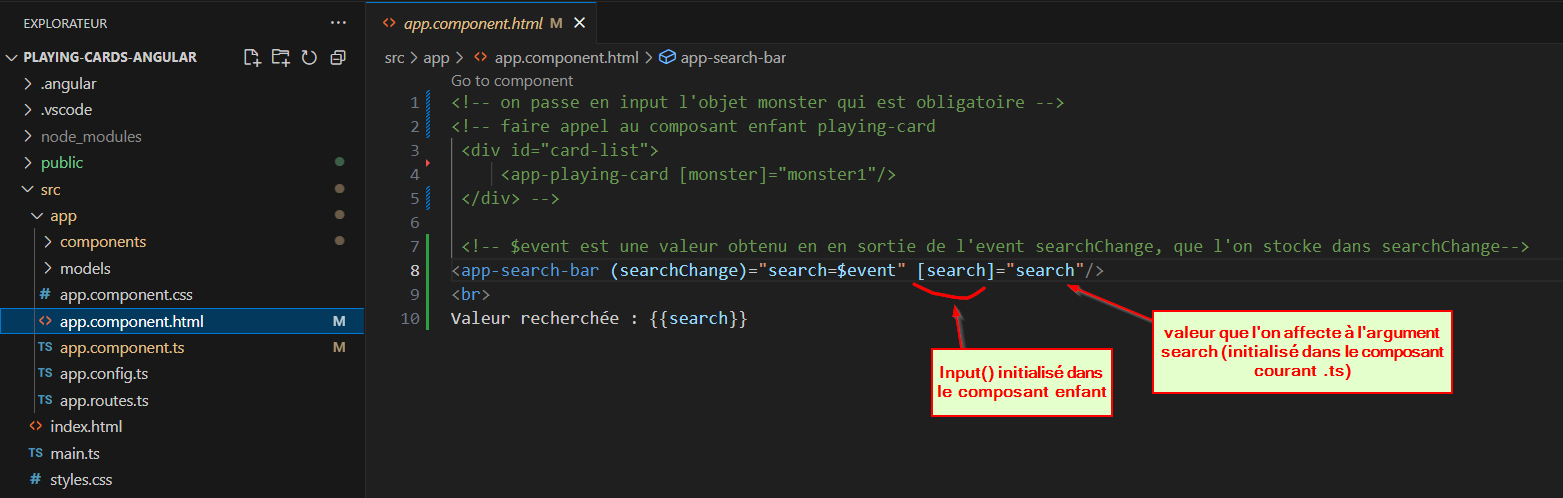




2. On crée une variable dans le composant parent principal app.component.ts



3. dans le html du composant parent principal, on fait appel au composant enfant en lui passant une variable à l’argument Intput() search, le nom de l’Input doit être entre crochets, et la variable qu’on passe en argument doit être entre guillemets

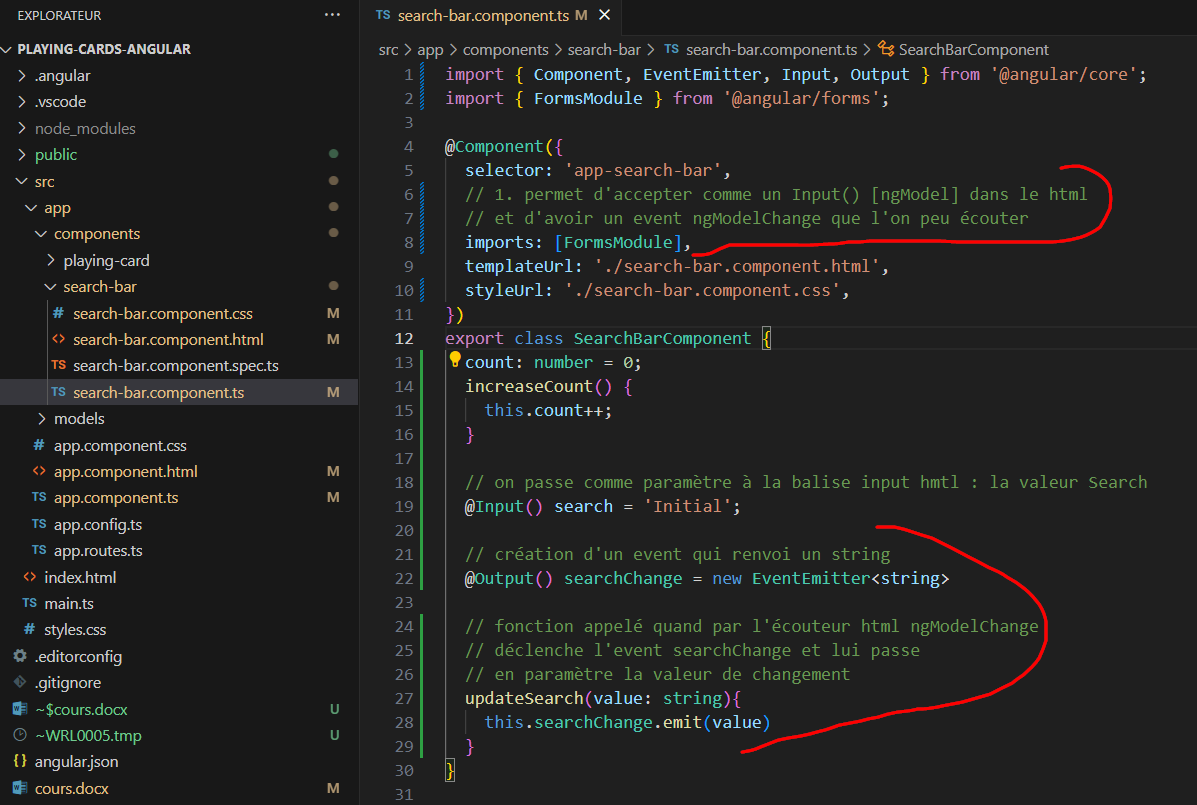


On note que l’Input() sert à avoir la possibilité que le composant enfant créé puisse accepté un argument en paramètre

On a la possibilité d’utiliser les signaux à la place qui Input(), qui sont plus récent

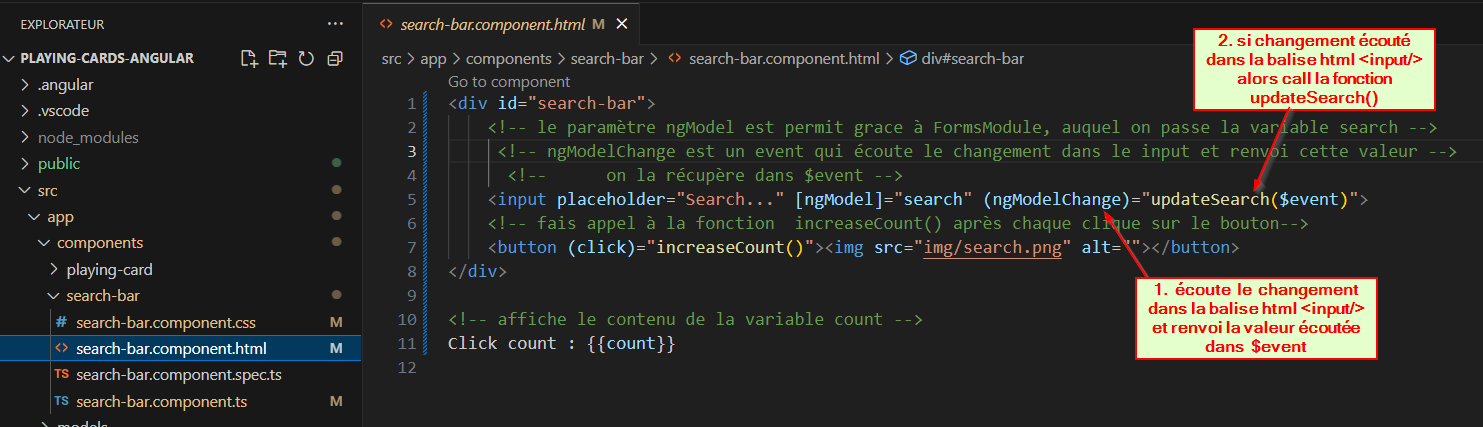
**Output() : créer un évènement de type Output() dans le composant enfant, sert au composant enfant à renvoyer une valeur qui sera disponible dans le html du composant parent si dans le composant parent html on écoute l’événement Output()**

1. Dans le composant enfant .ts on import FormModule, qui nous à la fois d’avoir dans le composant enfant un Input() nommé [ngModel] mais d’avoir l’event ngModelChange dans le composant enfant qui permet de rester à l’écoute d’un changement dans une balise html <input/> et de renvoyer cette valeur

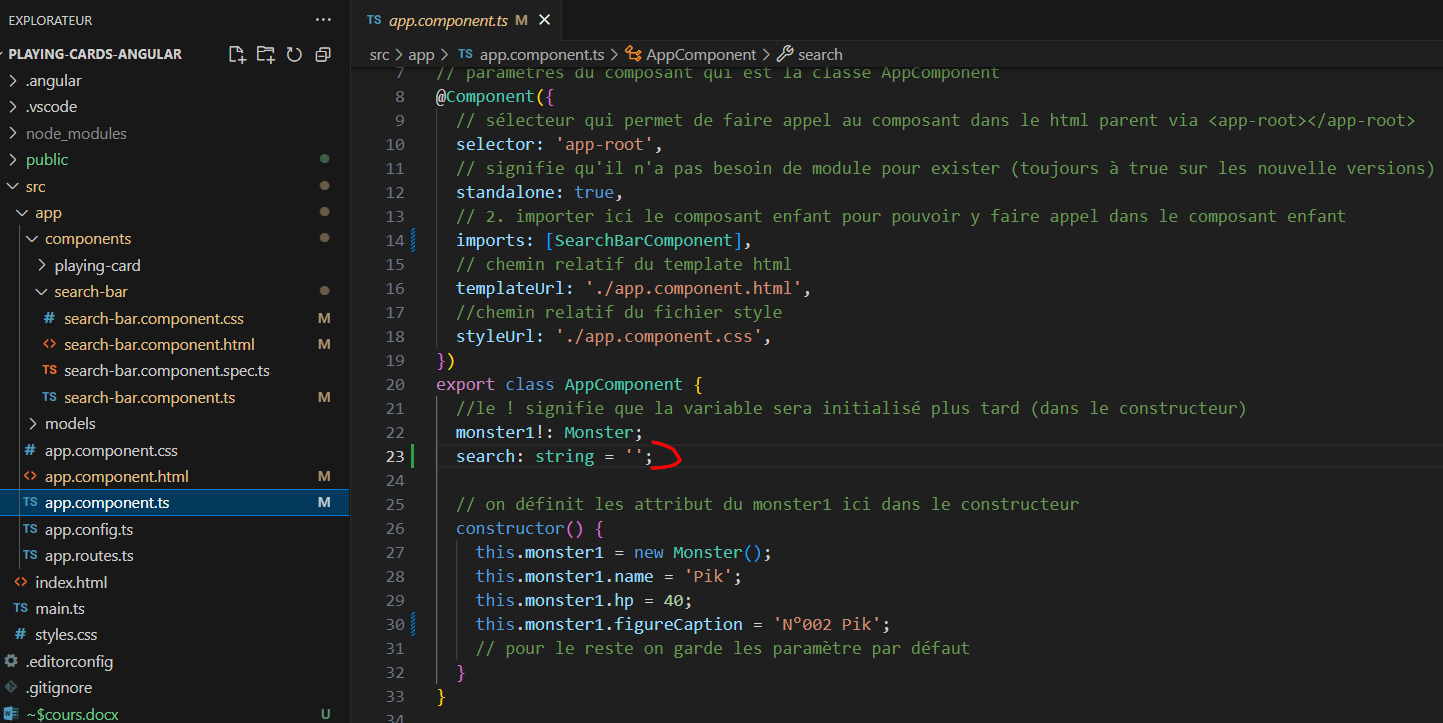


2. En seconde partie on crée l’évènement de type Output() ici nommé searchChange qui se chargera si déclenché, de renvoyer une valeur qu’on lui aura passé en paramètre (on pourra écouter cette évènement dans le composant parent html et ainsi récupérer la valeur renvoyé par cette event)

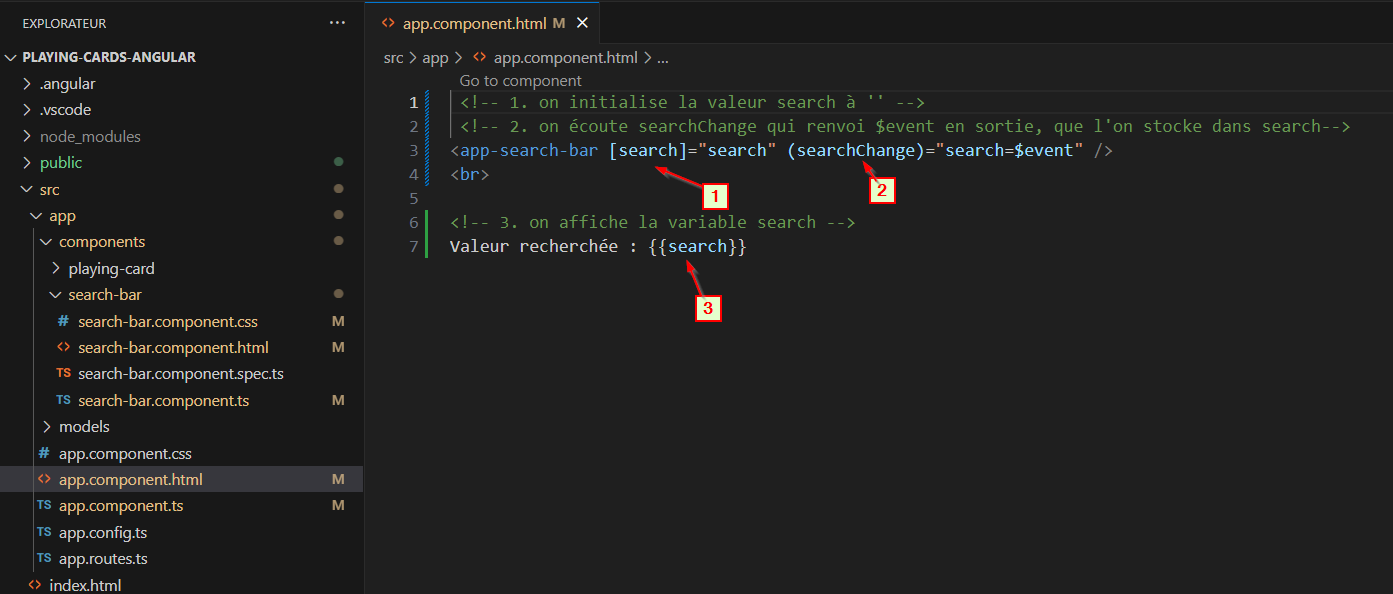
Dans le html du composant enfant, nous écoutant l’évent ngModelChange qui se charge d’écouter le changement mais aussi de renvoyer la valeur d’une balise html <input/>, et c’est ici que nous faisons appel à la fonction updateSearch() qui déclenche l’event searchChange et lui passe en paramètre la valeur value obtenue à partir ngModelChange



3. dans le composant parent app.component.ts on initialise une variable qui servira à stocker l’Output() de l’event searchChange



4. dans app.component.ts on fait appel au composant enfant app-search-bar, dans lequel on écoute l’event searchChange et on la stock dans la variable search, puis on l’affiche



Si le début du nom d’un événement Output() est similaire à une variable Input() possibilité de simplifier



En

