**Programme de Formation Angular 18**

**Module 1: Introduction à Angular**

* **Introduction à Angular**
  + Qu'est-ce qu'Angular?
  + Architecture d'Angular : Modules, Composants, et Services
  + Configuration de l'environnement de développement
  + Introduction à Angular CLI
* **Installation et Configuration**
  + Installation de Node.js et Angular CLI
  + Création d'une nouvelle application Angular
  + Structure des dossiers et fichiers dans une application Angular
  + Introduction au fichier angular.json

**Module 2: Les Bases d'Angular**

* **Composants Angular**
  + Création d'un composant
  + Structure d'un composant (template, style, et logique)
  + Communication entre les composants (Input, Output)
  + Cycle de vie des composants
* **Templates et Data Binding**
  + Interpolation et Property Binding
  + Event Binding
  + Two-Way Data Binding avec ngModel
  + Directives structurelles
  + Directives d'attributs ([ngStyle], [ngClass])
* **Services et Dependency Injection**
  + Introduction aux services
  + Création et utilisation d'un service
  + Injection de dépendances dans Angular
  + Le rôle des providers

**Module 3: Gestion des Formulaires**

* **Formulaires Template-Driven**
  + Création d'un formulaire
  + Validation des formulaires
  + Gestion des erreurs et affichage des messages d'erreur
  + Utilisation de ngForm, ngModel, et les directives associées
* **Formulaires Reactive**
  + Introduction aux formulaires réactifs
  + Création d'un formulaire réactif avec FormBuilder
  + Validation réactive et gestion des états des champs
  + FormGroup, FormControl, et FormArray

**Module 4: Gestion des Données et HTTP**

* **HTTP Client et API REST**
  + Introduction à HttpClientModule
  + Envoi de requêtes HTTP (GET, POST, PUT, DELETE)
  + Gestion des Observables avec RxJS
  + Traitement des erreurs HTTP
* **RxJS et Programmation Réactive**
  + Introduction à RxJS
  + Concepts de base : Observable, Observer, et Subscription
  + Opérateurs RxJS courants : map, filter, mergeMap, switchMap
  + Manipulation des flux de données asynchrones

**Module 5: Routage et Navigation**

* **Routage de base**
  + Configuration des routes dans Angular
  + Navigation entre les routes
  + Paramètres de route et query parameters
  + Protection des routes avec AuthGuard
* **Routage avancé**
  + Chargement paresseux (Lazy Loading) des modules
  + Préchargement des modules
  + Utilisation des routes enfants

**Module 6: Gestion de l'État avec NgRx**

* **Introduction à NgRx**
  + Concepts de base : Store, Actions, Reducers, Selectors
  + Configuration de NgRx dans un projet Angular
  + Création et gestion du Store
  + Utilisation d'Effects pour gérer les effets secondaires

**Module 7: Tests Unitaires et End-to-End**

* **Tests unitaires avec Jasmine et Karma**
  + Introduction à Jasmine et Karma
  + Écriture de tests pour les composants, services, et directives
  + Test des formulaires et des services HTTP
* **Tests End-to-End avec Protractor**
  + Introduction à Protractor
  + Écriture de tests E2E pour les composants et le routage
  + Configuration et exécution des tests

**Module 8: Optimisation et Déploiement**

* **Optimisation de la Performance**
  + Optimisation du chargement des modules
  + Compression et minimisation des bundles
  + Utilisation de Change Detection Strategy
* **Déploiement d'une Application Angular**
  + Build et préparation pour la production
  + Déploiement sur Firebase, Netlify, ou un serveur web
  + Configuration d'un serveur pour le déploiement (Nginx, Apache)

**Module 9: Projet Final**

* **Développement d'une application complète**
  + Définition des exigences du projet
  + Mise en œuvre des concepts appris
  + Présentation du projet et retour d'expérience

**Durée et Méthodologie**

* **Durée:** Le programme peut être étalé sur 8 à 10 semaines, en fonction du rythme d'apprentissage des participants.
* **Méthodologie:** Chaque module doit comprendre des sessions théoriques suivies de travaux pratiques. Les participants doivent être encouragés à poser des questions et à résoudre des exercices pour renforcer leur compréhension.

**Matériel Requis**

* Ordinateur avec un environnement de développement (VS Code, Node.js, Angular CLI)
* Accès à une API REST pour les exercices pratiques (ou la création d'une API fictive)
* Documentation officielle Angular et ressources en ligne pour approfondir les sujets

**Fin de programme**

1. **Introduction à Angular**
2. **Qu'est-ce qu'Angular?**

Angular est un **framework JavaScript open-source** développé par Google pour créer des applications web dynamiques à page unique (**Single Page Applications - SPA**). Angular utilise **TypeScript** comme langage principal, une version typée de JavaScript, ce qui permet d'améliorer la maintenabilité et la lisibilité du code.

Angular se distingue par plusieurs fonctionnalités clés :

* **Data Binding bidirectionnel** : Les modifications apportées aux données dans le modèle se répercutent automatiquement dans l'interface utilisateur et vice-versa.
* **Injection de dépendances** : Un mécanisme qui permet de créer et de gérer les services dans une application de manière modulaire.
* **Modularité** : Angular segmente les fonctionnalités dans des modules, composants, services, etc., facilitant ainsi l'organisation et la réutilisation du code.

### 2. ****Architecture d'Angular : Modules, Composants, et Services****

Angular suit une architecture basée sur les composants et les modules pour organiser l'application.

* **Modules** : Ce sont des conteneurs logiques qui regroupent des composants, des services et d'autres ressources. Le module principal est AppModule dans lequel tous les composants sont déclarés. D'autres modules peuvent être créés pour diviser l'application en sous-parties logiques (par exemple, CustomerModule, AdminModule).
* **Composants** : Ce sont des blocs de construction réutilisables de l'interface utilisateur. Chaque composant possède une vue (HTML), une logique (TypeScript), et un style (CSS). Le composant racine est AppComponent, à partir duquel tous les autres composants dérivent.
* **Services** : Les services fournissent des fonctionnalités partagées entre les différents composants, comme l'accès aux données ou la gestion de la logique métier. Ils sont généralement injectés dans les composants à l'aide de l'injection de dépendances.

### 4. ****Configuration de l'environnement de développement****

Pour commencer avec Angular, il est nécessaire de configurer l'environnement de développement.

#### Outils principaux :

* **Node.js et npm** : Angular dépend de Node.js pour exécuter certains outils en ligne de commande. npm (Node Package Manager) permet d'installer les bibliothèques nécessaires, y compris Angular lui-même. [lien node js](https://nodejs.org/en)
* **Angular CLI (Command Line Interface)** : Il s'agit d'un outil qui permet de créer, générer et déployer des applications Angular avec des commandes simples.

# Installation global d'Angular CLI sur l’ordi : npm install -g @angular/cli

Il est aussi possible d’installer Angular dans un dossier spécifique dans votre ordinateur sans qu’il ne soit installé globalement

* 1. Créer un dossier
  2. Ouvrez le avec VS code
  3. Initialiser un projet node avec la commande : npm init –y
  4. Installer CLI dans le ce dossier : npm install @angular/cli@18 --save-dev
  5. Verifier la version : npx ng version
  6. Créer un nouveau projet dans ce dossier : npx ng new nom-du-nouveau-projet
  7. Deplacez vous dans le projet et verifier la version : npx ng version

**Editeur de code** : Utilisez un éditeur moderne tel que **Visual Studio Code**, qui prend en charge TypeScript, et offre de nombreuses extensions pour améliorer la productivité.

### 5. ****Introduction à Angular CLI****

L'**Angular CLI** (Command Line Interface) est un outil puissant qui facilite la gestion des projets Angular. Avec Angular CLI, vous pouvez générer des composants, des services, des modules et bien d'autres entités sans avoir à écrire la structure de base manuellement.

Voici quelques-unes des commandes de base :

**Créer un nouveau projet** : **ng new mon-projet**

**Servir l'application** (lancer un serveur de développement local) : ng serve

**Générer un composant** : ng generate component mon-composant

**Générer un service** : ng generate service mon-service

1. **Structure des dossiers et fichiers dans une application Angular**

Après avoir généré un projet avec Angular CLI, vous trouverez une structure de fichiers bien définie :

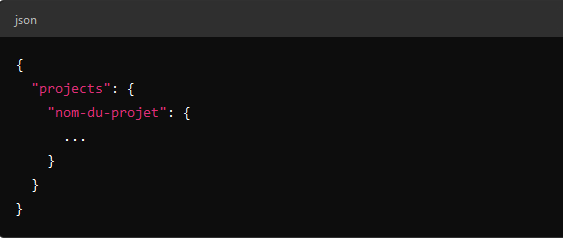
* **src/** : C'est le dossier principal où se trouve le code source de l'application. Il contient :
  + **app/** : Le cœur de l'application avec le composant racine (AppComponent).
  + **assets/** : Utilisé pour stocker les fichiers statiques comme les images, les fichiers CSS globaux, etc.
  + **environments/** : Contient des fichiers de configuration d'environnement (ex. environment.ts pour le développement et environment.prod.ts pour la production).
* **app/** : Ce répertoire contient les modules et composants Angular :
  + **app.component.ts** : Le fichier TypeScript qui définit la logique du composant racine.
  + **app.component.html** : Le fichier de template HTML associé au composant.
  + **app.component.css** : Le fichier de style pour ce composant spécifique.
* **node\_modules/** : Contient toutes les dépendances installées via npm.
* **angular.json** : Fichier de configuration principal d'Angular (voir la section suivante).
* **package.json** : Décrit les dépendances du projet, les scripts npm, et la version du projet.
* **tsconfig.json** : Contient la configuration TypeScript.

### ****Introduction au fichier angular.json****

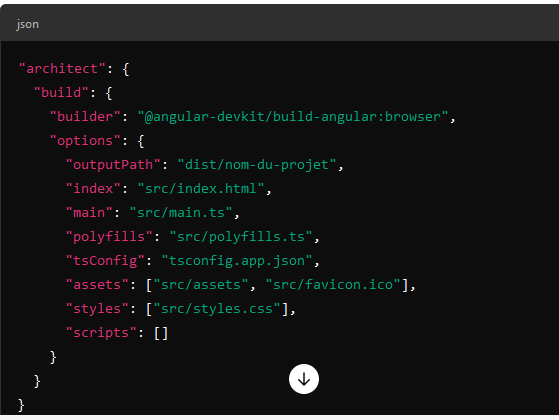
Le fichier **angular.json** est un fichier crucial dans un projet Angular. Il définit la configuration du projet et des différents environnements. Il permet de personnaliser le comportement d'Angular CLI.

Voici quelques sections clés du fichier **angular.json** :

* **Projects** : Liste les applications et bibliothèques Angular incluses dans le projet. Chaque application a sa propre configuration.

****

* **Architect** : Définit les configurations pour construire, tester, et lancer le projet. Par exemple, pour la construction de l'application :

****

* **Serve** : Contient la configuration utilisée pour exécuter ng serve (pour lancer l'application en mode développement).
* **Build Options** : Définit les fichiers à inclure lors de la compilation, tels que les fichiers principaux TypeScript (main.ts), le fichier HTML (index.html), les styles, et les polyfills.

Le fichier **angular.json** est donc essentiel pour contrôler le comportement de l'application Angular lors de son développement et de son déploiement. Vous pouvez modifier les configurations pour répondre aux besoins spécifiques de votre projet, comme l'ajout de chemins de styles globaux ou de scripts externes.

**Les Bases d'Angular**

#### ****Composants Angular****

Les composants sont les blocs de construction fondamentaux d'une application Angular. Chaque composant est une unité encapsulée de l'interface utilisateur, qui combine la logique, le template HTML, et les styles.

##### **1. Création d'un composant**

Pour créer un composant, vous pouvez utiliser Angular CLI. Supposons que nous voulons créer un composant appelé user-profile.

* Commande pour créer un composant :

**ng generate component user-profile** ou en version courte :

**ng g c user-profile**

Cette commande génère un dossier user-profile avec les fichiers suivants :

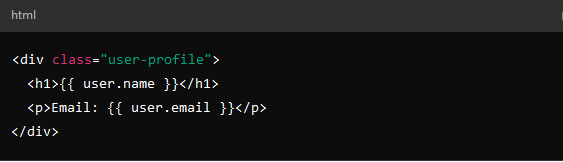
* user-profile.component.ts : Fichier TypeScript contenant la logique du composant.
* user-profile.component.html : Fichier HTML pour le template du composant.
* user-profile.component.css : Fichier CSS pour les styles du composant.
* user-profile.component.spec.ts : Fichier pour les tests unitaires du composant.

##### **2. Structure d'un composant (template, style, et logique)**

Chaque composant Angular se compose de trois parties principales : le template, les styles, et la logique.

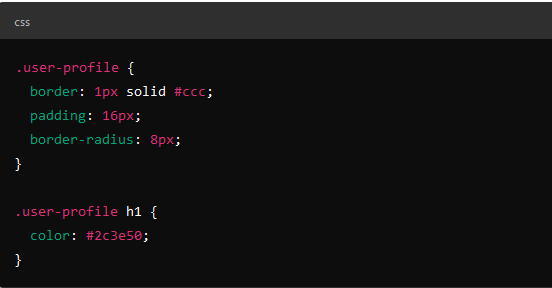
**Template (HTML) :** Définit la structure HTML du composant.

**Exemple - user-profile.component.html** :



**Styles (CSS) :** Contient les styles spécifiques au composant.

**Exemple - user-profile.component.css** :

****

* **Logique (TypeScript) :** Gère la logique du composant, les propriétés, et les méthodes.

**Exemple - user-profile.component.ts** :

****

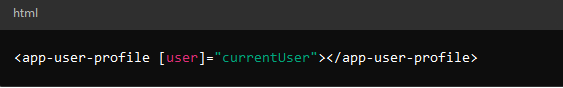
##### **3. Communication entre les composants (Input, Output)**

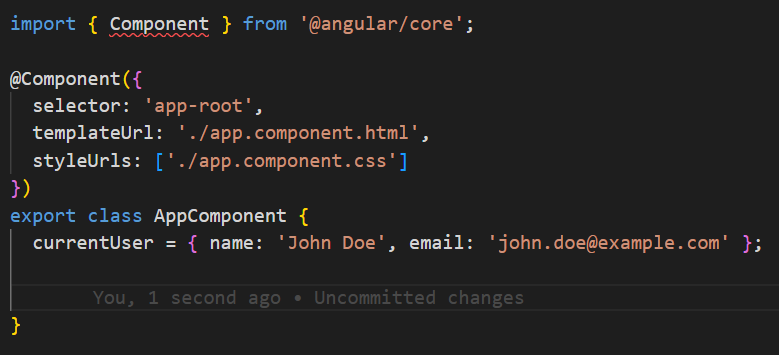
La communication entre les composants peut se faire de différentes manières, principalement à l'aide des décorateurs @Input et @Output.

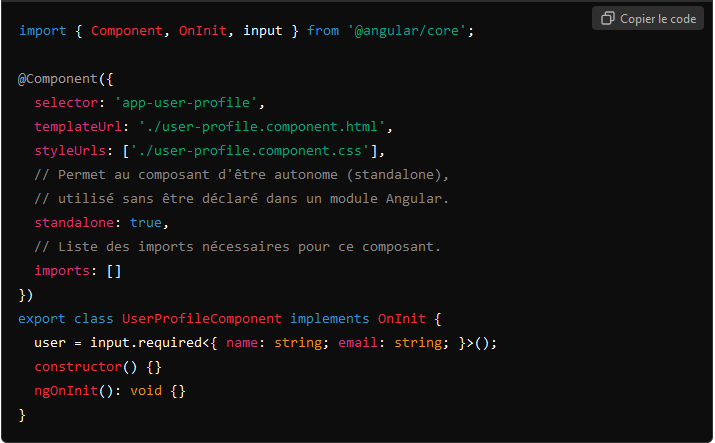
* **@Input** : Permet à un composant parent de passer des données à un composant enfant.

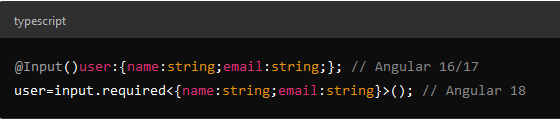
**C’est dans le composant enfant qu’on doit mettre le décorateur @Input**

**Exemple - Composant parent : app.component.html** :

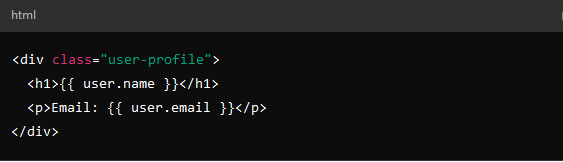
****

**app.component.ts** **Exemple - Composant enfant : user-profile.component.ts** :



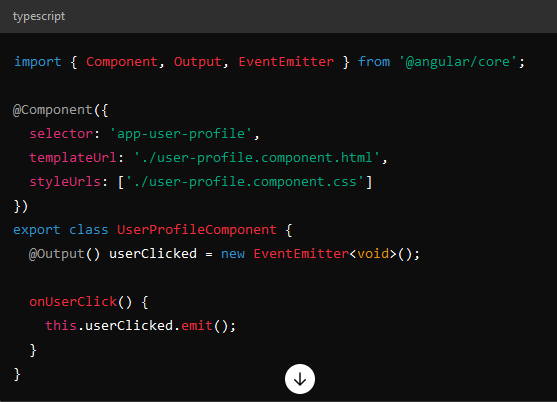


Le HTML du composant enfant :



**@Output** : Permet à un composant enfant d'émettre des événements vers son composant parent, toujours le decorateur doit etre dans l’enfant

**Exemple - Composant enfant : user-profile.component.ts** :

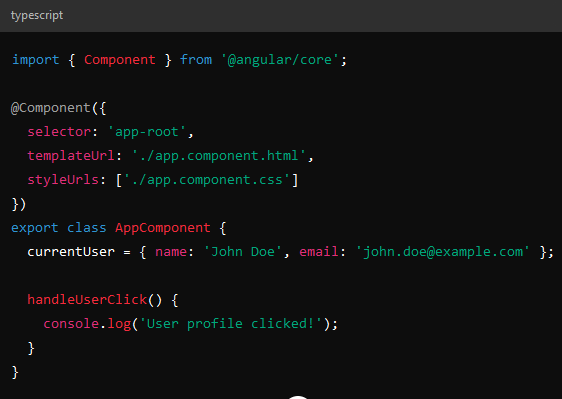


**Pour Angular18**

**Exemple - Composant parent : app.component.html** :



**Exemple - Composant parent : app.component.ts** :

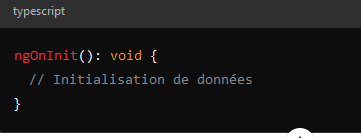


##### **4. Cycle de vie des composants**

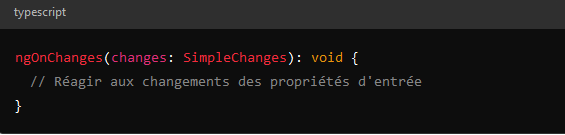
Les composants Angular passent par plusieurs phases au cours de leur cycle de vie. Angular fournit plusieurs hooks de cycle de vie qui permettent d'intervenir à différentes étapes du cycle de vie du composant.

* **ngOnInit()** : Appelé après l'initialisation des propriétés du composant.

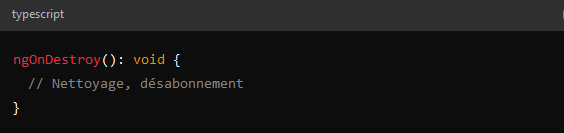
**Exemple** :



**ngOnChanges()** : Appelé lorsqu'une ou plusieurs propriétés d'entrée (@Input) du composant changent.

**ngOnDestroy()** : Appelé juste avant que le composant ne soit détruit. Utilisé pour libérer les ressources ou annuler les abonnements.

**Exemple** :

**ngAfterViewInit()** : Appelé après que la vue du composant a été initialisée. Utile pour des manipulations de DOM ou des intégrations avec des bibliothèques tierces.

**Exemple** :



* **Templates et Data Binding**

#### ****Interpolation****

**Objectif :** Afficher une variable dans le template.

1. **Création du composant simple :**

Création du composant simple :



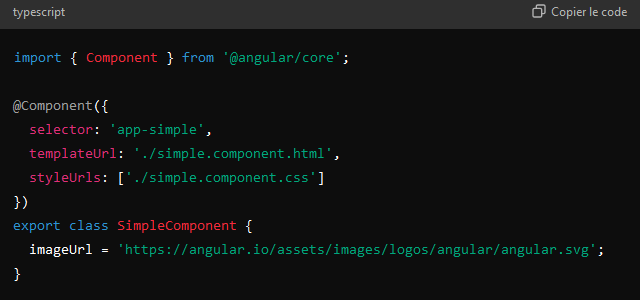
* simple.component.html :

Cela affichera "Hello, Angular!" dans un élément <h1>.

#### ****Property Binding****

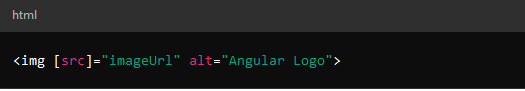
**Objectif :** Lier une propriété d'un élément HTML à une variable du composant.

1. **Code du composant :**
   * **simple.component.ts :**



Logo Angular : <https://angular.io/assets/images/logos/angular/angular.svg>

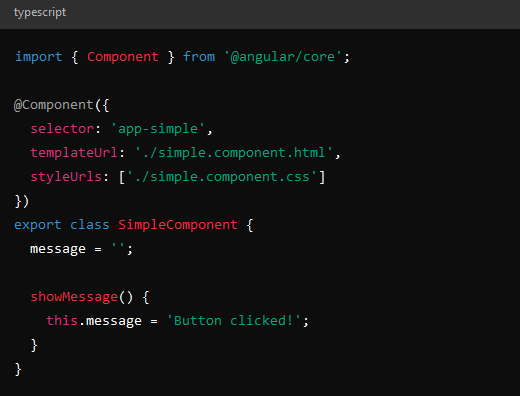
simple.component.html :

Cela affichera une image dont la source est définie par la variable imageUrl.

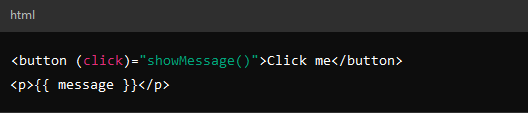
### 2. ****Event Binding****

**Objectif :** Réagir à un événement utilisateur en appelant une méthode du composant.

1. **Code du composant :**
   * **simple.component.ts :**



simple.component.html :

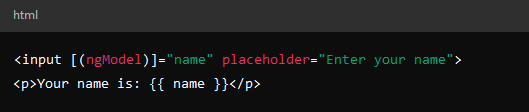
Cela affichera un bouton, et lorsque vous cliquez dessus, le message "Button clicked!" sera affiché sous le bouton.

### 3. ****Two-Way Data Binding avec**** ngModel

**Objectif :** Lier une entrée utilisateur à une variable du composant et refléter les modifications.

1. **Code du composant :**
   * **simple.component.ts :**

simple.component.html :



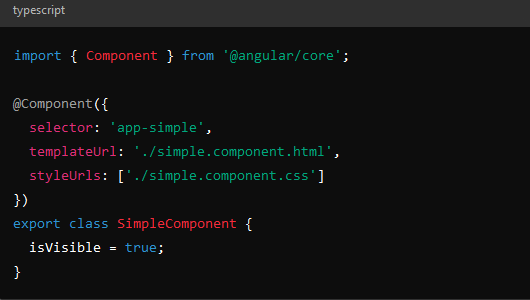
Cela affichera un champ de saisie. Le texte entré sera affiché sous le champ en temps réel.

### 4. ****Directives structurelles (****@If****,**** @For,......****)****

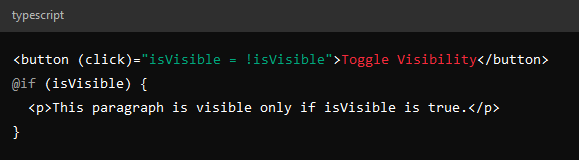
#### @If

**Objectif :** Afficher un élément en fonction d'une condition.

1. **Code du composant :**
   * **simple.component.ts :**

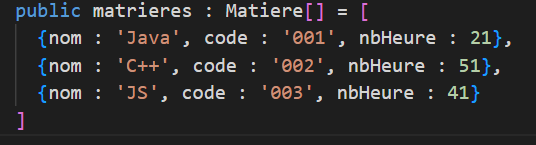


simple.component.html :

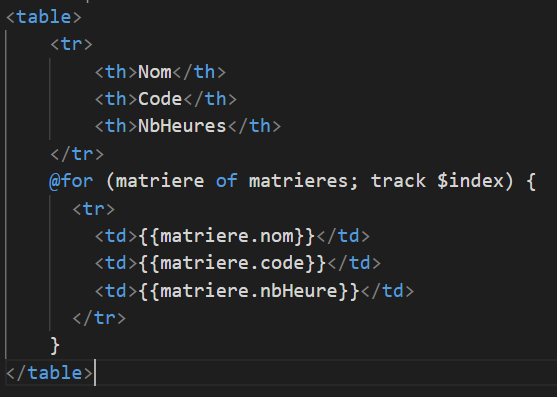


* @for

**simple.component.ts :**



**simple.component.ts :**

****

Le mot **track** est un identifiant unique pour chaque éléments de la boucle, angular vas se servir de la valeur pour identifier les éléments de façon unique, ici **index** est l’index de chaque élément dans le tableau, on pouvait mettre ‘matiere.code’ si ‘code’ est identifiant unique.

### 5. ****Directives d’attributs (****[ngStyle], [ngClass])

### Les directives d'attributs ngStyle et ngClass sont des outils puissants en Angular pour manipuler dynamiquement les styles CSS d'un élément HTML en fonction des données de votre application. Cela permet de créer des interfaces utilisateur plus interactives et réactives.

### ngStyle : Il faudra impoter ngStyle d’abord dans le ts

### 

**Dans le html :**

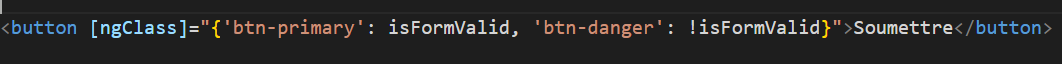


En supposant que ‘isImportant’ est bien déclaré dans le ts comme boolean

* ngClass : Il faudra impoter ngClass d’abord dans le ts



Dans le Html



Il faudra aussi que ces classes existent, soit vous les créer dans le fichier css du composant ou utiliser les classes de votre framWork Css installer dans le projet Angular.

* **Services et Dependency Injection**

### Introduction aux Services

En Angular, les services sont des classes qui encapsulent des fonctionnalités réutilisables, comme l'accès à des données, les appels d'API, ou la gestion d'événements. Ils permettent de séparer les préoccupations et de rendre votre code plus modulaire et testable.

### Création et Utilisation d'un Service

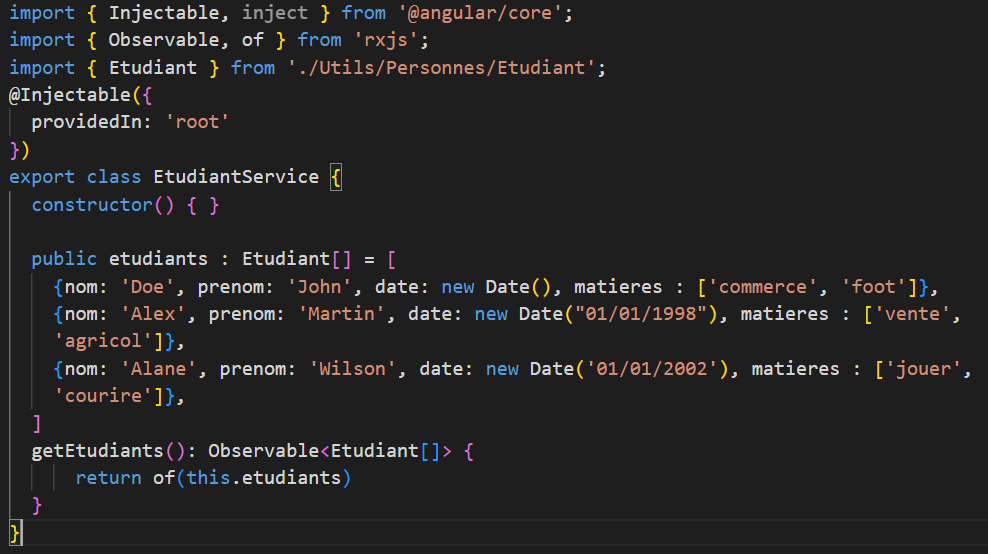
**1. Création du Service**

Créons un service nommé EtudiantService avec un composant EtudiantComponent qui simule la récupération d'informations sur des étudiants à partir d'une API :

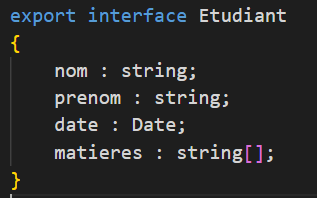
La commande : ‘ng g c Etudiant’ génère un composant EtudiantComponent

La commande : ‘ng g s Etudiant’ génère un service EtudiantService

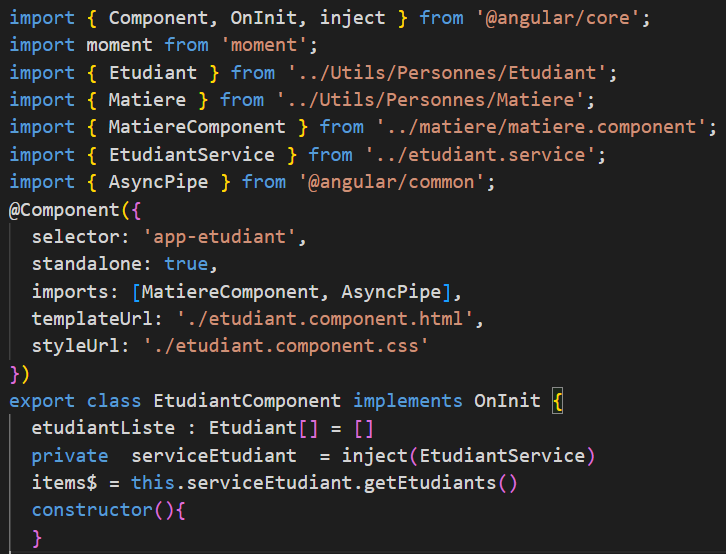
Dans le etudiant.service.ts :



Avec un fichier d’interface etudiant :



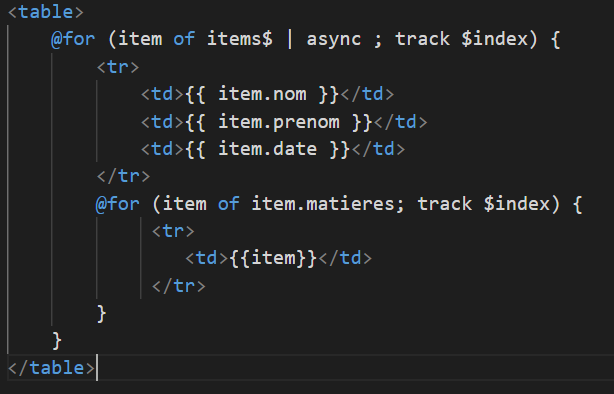
Dans le composant etudiant :



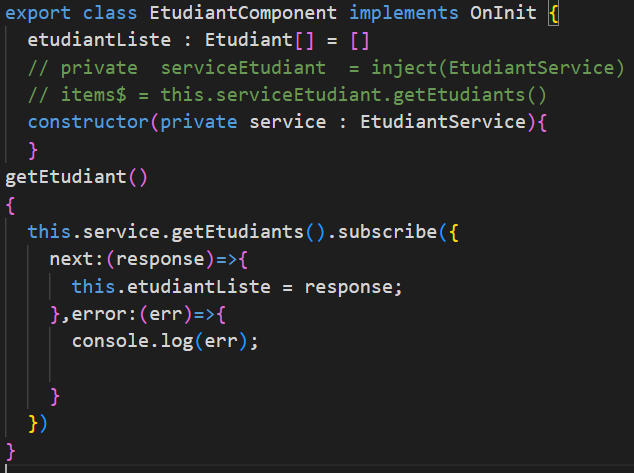
**items$:** C'est une propriété du composant, souvent de type Observable. Le $ à la fin est une convention courante pour indiquer qu'il s'agit d'un Observable. Un Observable est un objet qui émet des valeurs au fil du temps, ce qui est idéal pour gérer les données asynchrones comme les requêtes HTTP.

L’observable n’est toujours pas appeler d’abord tanque **items$ n’es pas utiliser dans le HTML**

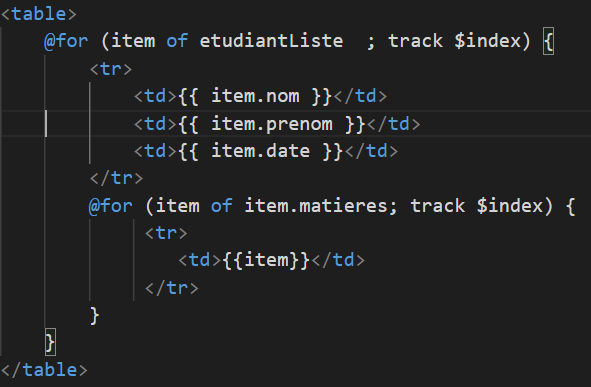
**Exemple :**



Autre façon de faire aussi:



On remplace le items$ par ‘etudiantListe’ dans le HTML en enlevant le async, ça donne le meme resultat



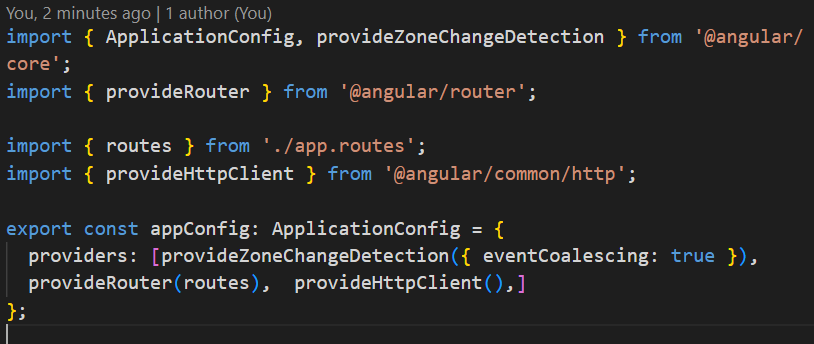
Le fait de mettre la déclaration du service dans le constructeur fait que l’instance est automatiquement crée.

La méthode **subscribe()** est utilisée dans Angular (via RxJS) pour **consommer un Observable**. Lorsque vous effectuez une requête HTTP (ou toute autre opération asynchrone), elle retourne généralement un **Observable**, et vous utilisez subscribe() pour réagir aux événements émis par cet Observable, comme la réception de données ou une erreur.

subscribe() prend un objet comme paramètre avec plusieurs options, ici :

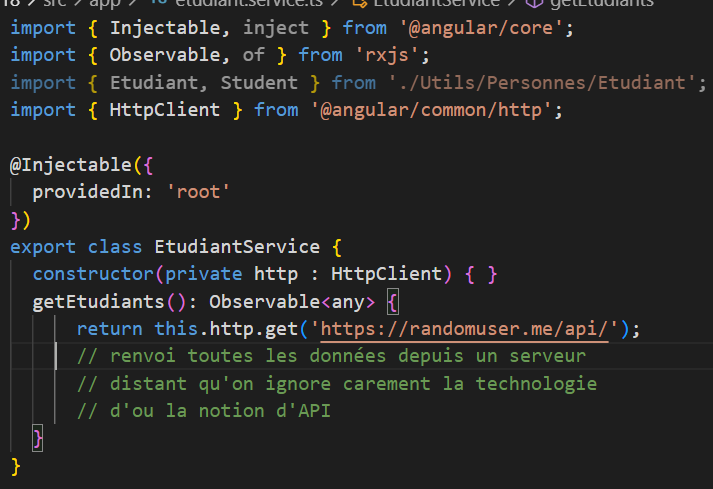
* **next** : Ce que l'Observable doit faire lorsqu'il reçoit une valeur (par exemple, la réponse de l'API).
* **error** : Ce que l'Observable doit faire lorsqu'une erreur survient (par exemple, si l'appel HTTP échoue).

Et il ne faut pas oublier de configurer le app.config en ajoutant la methode ‘**provideHttpClient**()’ :

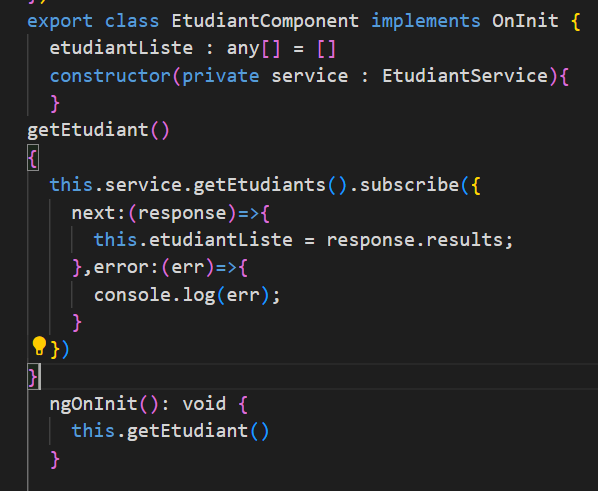


L’utilisation des verbes http exemple de get

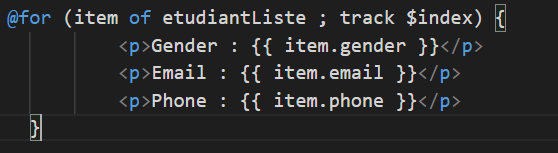
**Le service :**



**Le composant :**



**Le HTML** :



https://jsonplaceholder.typicode.com/users