



LI262

Applications web : Angular





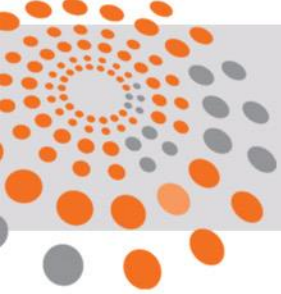
Introduction

Nicolas Lethuillier

Dev web @ Coaxys

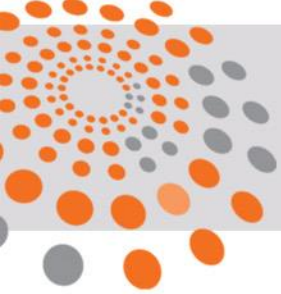
Expérience dans diverses technos web
Développement front-end





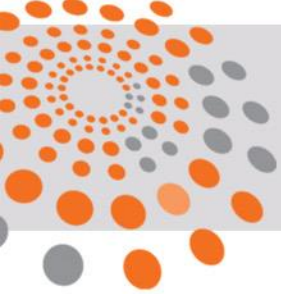
Organisation

- Durée : 5 jours
- Horaires : 9h - 17h
- Lieu : Dixisoft



Prérequis

- Avoir participé à un projet web
- Être autonome en HTML, CSS, Javascript (ES5)
- Être muni(e) d'un ordinateur avec MacOS, Windows ou Linux



Objectifs

- Comprendre ce qu'est une application web
- Être autonome sur **Angular** (version 8)
- Être autonome sur **Bootstrap**
- Être capable de prendre en main le développement front d'une application



Déroulement

- **Lundi** : présentation d'Angular, TP au fil de l'eau
- **Mardi** : Tour of Heroes (tutoriel Angular)
- **Mercredi** : Tour of Heroes (tutoriel Angular)
- **Jeudi** : présentation de Bootstrap, application à ToH
- **Vendredi** : fin du design et des fonctionnalités de ToH
- **Votre participation est importante !**
Cette formation doit être interactive, n'hésitez pas à poser des questions ou faire des commentaires

Au menu...



I.

Contexte

Les applications web, les frameworks, les outils



II.

Langages

HTML5, CSS3, ES6, TypeScript



III.

TypeScript

Intérêt, fonctionnement



IV.

Angular : initialisation

Présentation, nouveau projet, déploiement, modules



V.

Angular : composants

Premier composant, directives, binding, lifecycle hooks

Au menu...



VI. **Angular : routing**
Navigation dans une SPA



VII. **Tour of Heroes (1/2)**
Initialisation du projet



VIII. **Outils avancés**
Services, pipes, directives, EventEmitter



IX. **HTTP & RxJS**
Appels serveurs, Observables



X. **Tour of Heroes (2/2)**
Fin du tutoriel officiel



XI. **Dernier round**
Subject, RxJS avancé, AOT



CONTEXTE



- Au XX^{ème} siècle, on avait des sites web et des applications :



Sites web

Vitrines, sites simples...



Applications

*Logiciels installés destinés
à effectuer des actions*

- En 2019, on a aussi des applications web :



Applications web
Vitrines, sites simples...



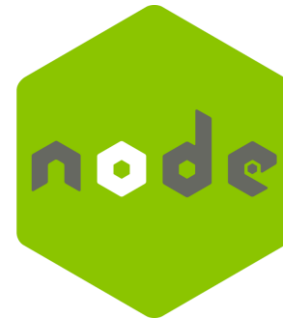
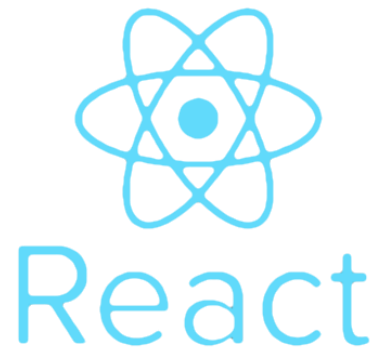
Applications
*« Clients lourds » pour exploiter
toute la puissance de la machine*

- Une **application web** est donc un site web qui ne se contente pas d'afficher de l'information, mais qui permet à l'utilisateur :
 - De consulter, modifier des informations qui lui sont propres
 - Effectuer des actions qui ont un impact sur sa navigation et/ou celle des autres



- Avec la mode du Cloud, de nombreux éditeurs font des versions « web » de leurs logiciels

Word Online, Photopea, etc.





- Angular permet de créer des applications « *Single Page* » (SPA)
- Il utilise le paradigme de la **programmation orientée composants**
- Il utilise les langages et extensions de fichiers bien connus du web (HTML, CSS, JavaScript)... en ajoutant quelques bonus










- Une application Angular est donc composée :
 - De fichiers `.html`
 - De fichiers `.css`
 - De fichiers `.js`
 - De fichiers `.ts`
 - De fichiers `.conf`, `.properties`, etc.
- Elle inclut également un `index.html` à la racine
- L'arborescence, elle, est (quasiment) libre !



- Une application Angular est une application Node.js
- La racine du projet possède toujours les éléments suivants :

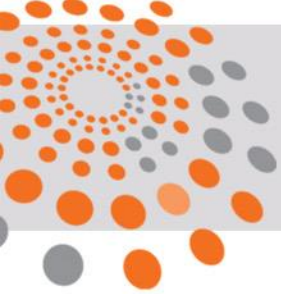
	<code>node_modules</code>	Code des dépendances (bibliothèques, etc.)
	<code>src</code>	Code de l'application
	<code>angular-cli.json</code>	Configuration d'angular-cli
	<code>package.json</code>	Liste des dépendances
	<code>tsconfig.json</code>	Configuration de TypeScript



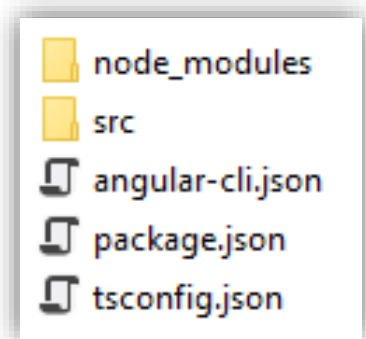


webpack

- Webpack est un outil couteau-suisse, indépendant d'Angular mais requis pour en faire
- Il gère le découpage en modules, le packaging, ...
- Bonne nouvelle : vous n'aurez pas (trop) à vous en soucier
C'est aussi une de ses qualités !



- Npm est le « gestionnaire de paquets officiel de Node.js » (*Wikipedia*)
- Il permet d'installer, désinstaller et administrer les dépendances de nos projets
- Celui-là, par contre, c'est un incontournable !



- Exemples :

`npm install`

*// Téléchargement et installation de toutes les dépendances du projet
// Cette commande lit le fichier « **package.json** », télécharge les paquets,
// et les installe dans le répertoire « **node_modules** »*

`npm update`

// Mise à jour des dépendances du projet



- Exemples :

```
npm install html2pdf
```

*// Installation de la bibliothèque **html2pdf** pour le projet actuel*

```
npm install -s ng2-social-share
```

*// Installation de **ng2-social-share** pour le projet actuel*

*// avec sauvegarde dans le « **package.json** » du projet*

```
npm install -g ng-youtube-embed
```

*// Installation globale de **ng-youtube-embed***

// Tous les projets en local peuvent désormais l'utiliser



- Angular-cli (*Command Line Interface*) permet d'administrer son projet Angular en ligne de commande.
- Exemples :

```
ng new my-project
```

```
// Création d'un nouveau projet Angular  
// avec les paramètres par défaut
```

```
ng serve
```

```
// Lancement de l'application « en live »
```



webpack



Angular CLI

- On va donc commencer par installer tout ça !



- Etape 1/3 : **Node.js / npm**

<https://nodejs.org/>

Node.js® is a JavaScript runtime built on Chrome's V8 JavaScript engine.

Download for Windows (x64)

10.16.3 LTS

Recommended For Most Users

12.9.0 Current

Latest Features

[Other Downloads](#) | [Changelog](#) | [API Docs](#)

[Other Downloads](#) | [Changelog](#) | [API Docs](#)

Or have a look at the [Long Term Support \(LTS\) schedule](#).

Sign up for [Node.js Everywhere](#), the official Node.js Monthly Newsletter.



- Etape 1/3 : **Node.js / npm**

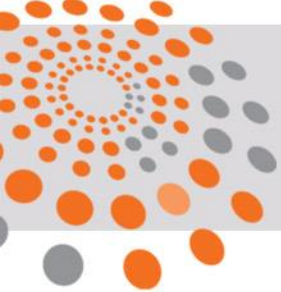
```
C:\> Invite de commandes

Microsoft Windows [version 10.0.18362.295]
(c) 2019 Microsoft Corporation. Tous droits réservés.

C:\Users\NicolasLETHUILLIER>node -v
v10.16.3

C:\Users\NicolasLETHUILLIER>npm -v
6.9.0

C:\Users\NicolasLETHUILLIER>
```

- Etape 2/3 : **angular-cli**

npm install -g @angular/cli

```
C:\Users\NicolasLETHUILLIER>npm install -g @angular/cli
C:\Users\NicolasLETHUILLIER\AppData\Roaming\npm\ng -> C:\Users\NicolasLETHUILLIER\AppData\Roaming\npm\node_modules\@angular\cli\bin\ng

> @angular/cli@8.2.2 postinstall C:\Users\NicolasLETHUILLIER\AppData\Roaming\npm\node_modules\@angular\cli
> node ./bin/postinstall/script.js

? Would you like to share anonymous usage data with the Angular Team at Google under
Google's Privacy Policy at https://policies.google.com/privacy? For more details and
how to change this setting, see http://angular.io/analytics. No
+ @angular/cli@8.2.2
added 240 packages from 185 contributors in 19.928s

C:\Users\NicolasLETHUILLIER>
```

```
C:\Users\NicolasLETHUILLIER>
```

- Etape 3/3 : **l'environnement !**

Installer l'IDE de votre choix

Mon conseil :



Visual Studio Code

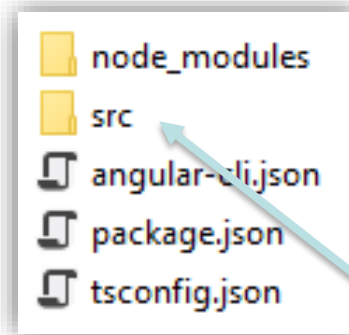
#Microsoft

#Libre

#bourré_d'extensions

#idéal_pour_le_web

- Vous avez désormais toutes les briques nécessaires pour générer, développer et packager un projet Angular
- Nous avons vu ce qu'il y a à la racine d'un projet :



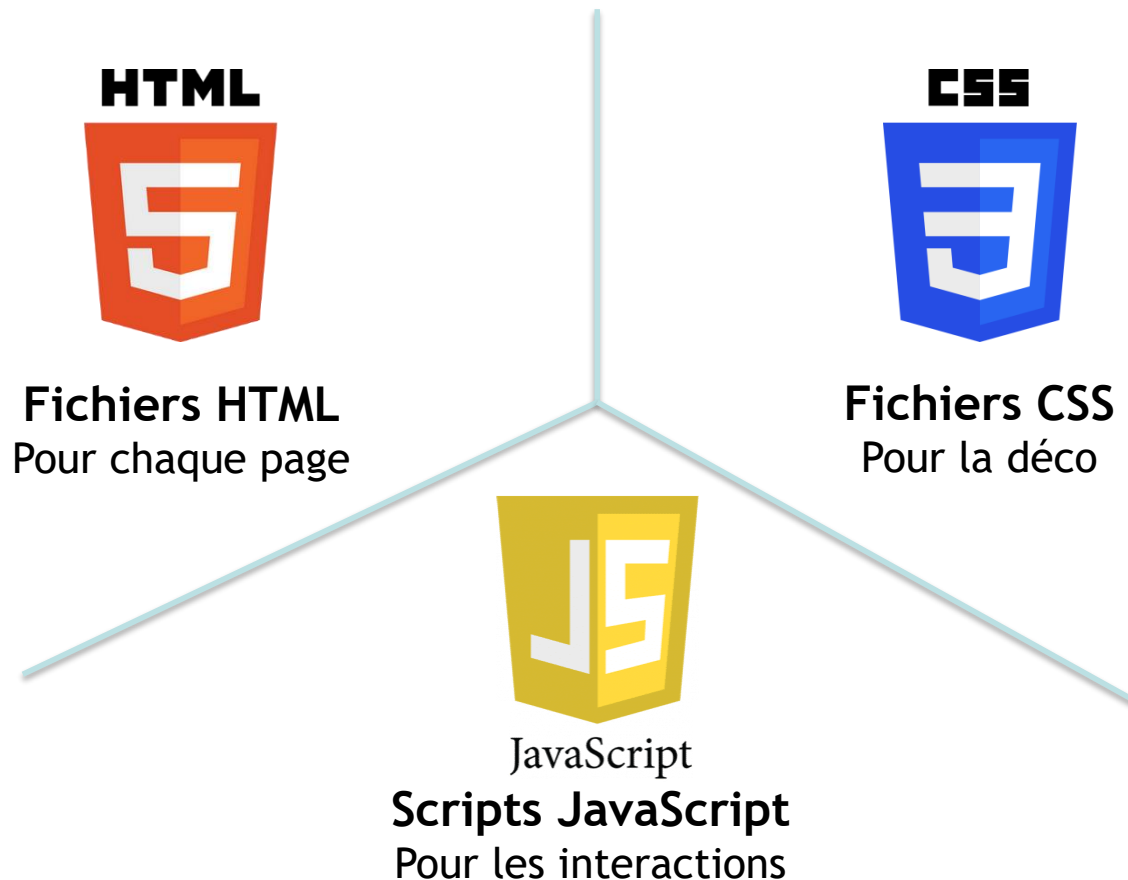
- Nous allons maintenant parler des sources, et en particulier des langages mis en jeu.
- Des questions ?



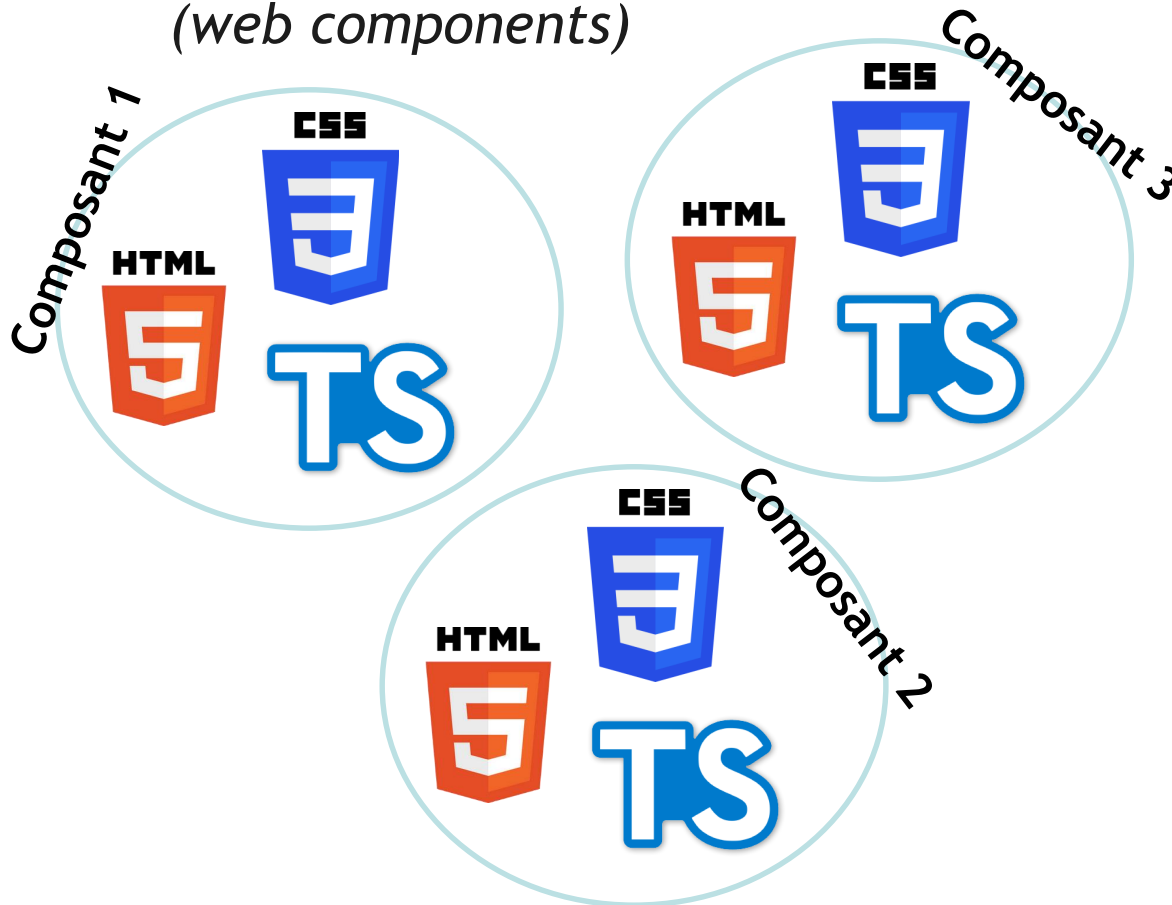
LANGAGES



- Un projet web est en général composé de :



- Un projet Angular est, lui, composé de composants web (*web components*)



TS

Fichiers TypeScript

HTML

index.html

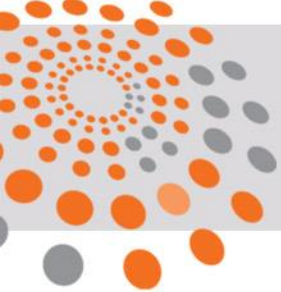


- La programmation **orientée composants** consiste à découper son application en une multitude de composants web indépendants
- Les composants sont des briques qui contiennent chacune :
 - Son code (fichier TypeScript)
 - Son template (fichier HTML)
 - Son style (fichier CSS)



- Exemple :

The screenshot displays the 'L'Œuvre Cinéphile' website. The header includes navigation links: 'ORCHESTRE À VENTS', 'ORCHESTRE À CORDES', 'CHAMBRE', 'DUO', and 'CHERCHER'. Social media icons for Twitter and Facebook are also present. The main banner features a collage of Marvel characters (Captain America, Iron Man, Doctor Strange) with the text 'Marvel, Captain America: First Avenger, Iron Man 3, Doctor Strange, The Avengers' and 'Marvel Suite'. Below the banner, the 'FICHE TECHNIQUE' section provides details: 'Composé par Michael Giacchino, Alan Silvestri, Brian Tyler', 'Orchestre à cordes', 'Créé le 09/07/2019', and tags 'Cinéma/TV', 'Marvel', and 'Medley'. A play button icon and a 4:37 duration are shown. The musical score section, titled 'MARVEL SUITE', lists the conductor 'Conducteur', the composers 'M. Giacchino, A. Silvestri, B. Tyler', and the arranger 'Arr. Nicolas Lethuillier'. The score is for Violon I, with a tempo of 164 and dynamics like *mp* and *mf*.



- Conclusion :
 - L'intérêt principal des composants est la **réutilisabilité**
On ne duplique plus le code !
 - Un autre intérêt : les **scopes CSS**
Chaque composant a son fichier CSS, qui n'impacte que lui
→ *On peut donc clarifier le code, avec des standards pour les classes, etc.*
sans craindre les dommages collatéraux
 - On peut créer des composants de toutes tailles :
 - Un lecteur PDF de 2000px de haut
avec un bouton « Enregistrer » et un bouton « Imprimer »
 - Une note de 1 à 5 étoiles
 - L'important est de rationnaliser



- Quelle que soit la technologie web employée, des connaissances en HTML sont primordiales
- **Attention** : créer des composants permet d'« étendre » le HTML en rajoutant des balises



- Exemple :

```
▶<div _ngcontent-ekv-c5 class="length">...</div>
▶<div _ngcontent-ekv-c5 class="band">...</div>
  <!--bindings={
    "ng-reflect-ng-if": "4"
  }-->
▶<star-mark _ngcontent-ekv-c5 _ngghost-ekv-c7 ng-reflect-mark="
  <div _ngcontent-ekv-c5 class="created">Crée le 20/08/2019</di
▶<div _ngcontent-ekv-c5 class="tagsbox">...</div>
</div>
▼<div _ngcontent-ekv-c5 class="fieldset listen">
  <div _ngcontent-ekv-c5 class="legend">Ecouter</div>
▶<audio-player _ngcontent-ekv-c5 _ngghost-ekv-c8 ng-reflect-aud
  </audio-player>
</div>
▶<div _ngcontent-ekv-c5 class="fieldset download">...</div>
```



- Il faut donc :
 - Faire attention au nommage (par défaut préfixé « app- »)
 - Bien connaître ses balises HTML
- Allez, un petit quiz ?



HTML5 ou composant ?


 fieldset


 viewer


 video


 datalist

 h7


 star

 legend


 bucket

 icon

 progress

 link

 article

 audio

 code

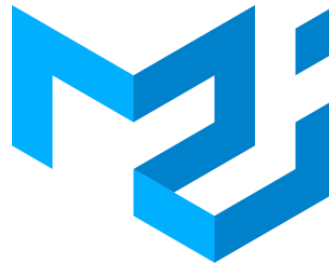
 pre



- Chaque composant a son propre fichier CSS
- On peut donc avoir deux éléments avec les mêmes classes mais des styles différents (scopes CSS)
- On peut également utiliser le fichier **style.css** à la racine du répertoire « src »
 - Celui-ci impacte l'ensemble du site (composants inclus)



- On peut utiliser n'importe quel framework CSS existant



- Angular n'apporte aucun style et/ou pratique quelconque



- Angular permet l'utilisation de **préprocesseurs CSS**



- Mais ce n'est pas l'objet de ce cours, nous en resterons à CSS 😊



- Angular est développé en **TypeScript**
- C'est donc le langage principalement utilisé dans les applications Angular
- TypeScript est une « *surcouche* » de **JavaScript** :
*il apporte des fonctionnalités supplémentaires
et surtout des contraintes de développement*
- Qualité +++



- Créé par Microsoft en 2012
par Anders Hejlsberg, l'inventeur du C#
- Supporte la spécification ECMAScript 6
- Utilise donc beaucoup, notamment, les fonctions fléchées :

```
myArray.forEach(function(item) {  
    // Do something with item  
});
```



```
myArray.forEach(item => {  
    // Do something with item  
});
```



- Le principal apport de **TypeScript**, comme son nom l'indique, est la **gestion de types** que n'a pas JavaScript
- On va donc manipuler des **types** et des **classes** comme on le fait en **Java** ou en **C#**
- Dès lors, toute variable pourra être typée
On s'appliquera donc à le faire systématiquement



- Au build, le code TypeScript est compilé en code JavaScript, pour se retrouver avec la triade du web habituelle :
 - *HTML*
 - *CSS*
 - *JavaScript*
- Google Chrome permet cependant de debugger le TypeScript (voir plus tard)



- Si vous avez **Visual Studio**, il est possible que **TypeScript** soit déjà installé.
- Sinon :

```
C:\Users\NicolasLETHUILLIER>tsc -v  
Version 3.5.3
```

`npm install -g typescript`

```
C:\Users\NicolasLETHUILLIER>npm install -g typescript  
C:\Users\NicolasLETHUILLIER\AppData\Roaming\npm\tsserver -> C:\Users\NicolasLETHUILLIER\AppData\Roaming\npm\typescript\bin\tsserver  
C:\Users\NicolasLETHUILLIER\AppData\Roaming\npm\tsc -> C:\Users\NicolasLETHUILLIER\AppData\Roaming\npm\typescript\bin\tsc  
+ typescript@3.5.3  
added 1 package from 1 contributor in 1.123s
```

9q96q J b9ck966 fLOW J coufLipnfoL Ju J'TJ32



TYPESCRIPT





- On l'a vu : **TypeScript** est compilé pour donner du **JavaScript**
- On peut donc utiliser, dans du code **TypeScript**, tout ce que l'on connaît en **JavaScript**
- Cependant, on veillera à respecter les bonnes pratiques et usages d'**ES6**



- Bons usages ES6 (1/3) :

Fonctions fléchées

```
myArray.forEach(function(item) {  
  // Do something with item  
});
```



```
myArray.forEach(item => {  
  // Do something with item  
});
```

```
sum = function(a, b) {  
  return a + b;  
}
```



```
sum = (a, b) => a + b;
```




- Bons usages ES6 (2/3) :

Egalité de types

```
"2" == 2
```

TRUE

Puisqu'on développe avec un langage fortement typé, on bannit ==

```
"2" === 2
```

FALSE

Je suis toujours censé savoir quel type je manipule

```
"2" != 2
```

FALSE

```
"2" !== 2
```

TRUE

Si je dois comparer une chaîne et un entier, j'utilise des fonctions comme `parseInt()`

- Bons usages ES6 (3/3) :

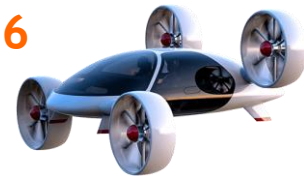
Déclaration de variables



ES5



ES6



```
var counter = 12;

if (counter > 10) {
  var message = "C'est long...";

  if (counter < 20) {
    message = "ça va quand même.";
  }
}

console.log(message);
```

```
const counter = 12;

let message;

if (counter > 10) {
  message = "C'est long...";

  if (counter < 20) {
    message = "ça va quand même.";
  }
}

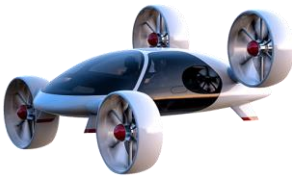
console.log(message);
```



- Bons usages ES6 (3/3) :

Déclaration de variables

ES6



```
const counter = 12;

let message;

if (counter > 10) {
  message = "C'est long...";
}

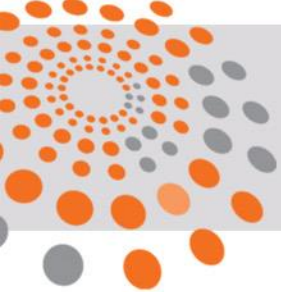
if (counter < 20) {
  message = "ça va quand même.";
}

console.log(message);
```

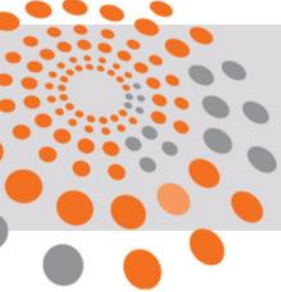
✗ **var** : à bannir
C'est le « fourre-tout » version Préhistoire

➡ **const** : pour les constantes
On ne peut pas modifier la valeur après l'initialisation

➡ **let** : pour les variables
On déclare une variable (avec éventuellement un type, voir ci-après)



- La caractéristique principale de TypeScript est d'ajouter à JavaScript des **types**
- Jusqu'alors, JavaScript avait quelques types basiques (entiers, chaînes, etc.) mais faisait preuve de beaucoup de souplesse dans les opérations quotidiennes ; quant aux classes...
- Avec **TypeScript**, la rigolade, c'est terminé



- TypeScript met à disposition quelques types basiques :
 - `number`
 - `string`
 - `boolean`
 - `void`
 - `undefined`
- Ces types existent en JavaScript...
mais on a rarement besoin de s'en préoccuper
- Les classes plus sophistiquées qui existent en JavaScript
sont accessibles : `Date`, `Array`, etc.



- En TypeScript, il n'est pas obligatoire de typer ses variables quand on les déclare
 - Mais ce serait bête de s'en priver, non ?
- Exemples :

```
let value; // Sans type, ça marche, mais on ne sait pas ce que c'est  
let value: number; // Voilà : value est un nombre (entier ou décimal)  
let value = 12; // Lorsqu'on initialise la variable dès la déclaration, le type est facultatif, car déduit
```

- On peut également typer « à la volée » :

```
let human: { name: string, gender: string };  
let human = { name: "Roberto", gender: "male" };
```

Ici, **human** est typé

Ici aussi, déduit des propriétés



- **Objectif** : typer toutes les variables, y compris dans les fonctions

```
function formatTime(hours: number, minutes: number): string {  
    return hours + ':' + minutes;  
}
```

```
function createHuman(name: string): { name: string, age: number } {  
    // Miracle of birth  
    return { name, age: 0 };  
}
```

ES6 : il n'est pas nécessaire d'écrire
name: name
si le nom est le même



- On peut, en TypeScript, créer des classes, de la même manière qu'en C# ou en Java
- Les classes peuvent avoir :
 - Des attributs
 - Un ou plusieurs constructeurs
 - Des méthodes
- Pour pouvoir y accéder en-dehors du fichier de définition, on la préfixe avec le mot-clé « **export** »

- Exemple :

```
export class Human {  
    private _name: string;  
    get name(): string {  
        return this._name;  
    }  
    set name(value: string) {  
        this._name = value;  
    }  
    constructor(birthName: string) {  
        this.name = birthName;  
    }  
    public eat(food: Course[]): void {  
        // eat  
    }  
    public pray(): void {  
        // pray  
    }  
    public love(): void {  
        // love  
    }  
}
```

Un attribut avec ses getter et setter

Un constructeur

Une méthode avec un argument (typé)

D'autres méthodes...



- Exemple :

```
public eat(food: Course[]): void {  
    // eat  
}
```

Je ne peux pas appeler
`eat()`

→ Too few parameters

Je ne peux pas appeler
`eat([...], true)`

→ Too many parameters

Mais je souhaite que food soit facultatif
Je lui donne donc une valeur par défaut :

```
public eat(food: Course[] = []): void {  
    // eat  
}
```

Ou je précise simplement
que l'argument est facultatif

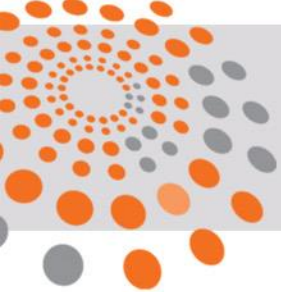
```
public eat(food?: Course[]): void {  
    // food peut ne pas être renseigné  
    // et sera undefined le cas échéant  
}
```

- Exemple :

```
function buildName(firstName: string, lastName = "Smith") {  
    return firstName + " " + lastName;  
}  
  
let result1 = buildName("Bob"); // works correctly now, returns "Bob Smith"  
let result2 = buildName("Bob", undefined); // still works, also returns "Bob Smith"  
let result3 = buildName("Bob", "Adams", "Sr."); // error, too many parameters  
let result4 = buildName("Bob", "Adams"); // ah, just right
```

```
function buildName(firstName = "Will", lastName: string) {  
    return firstName + " " + lastName;  
}  
  
let result1 = buildName("Bob"); // error, too few parameters  
let result2 = buildName("Bob", "Adams", "Sr."); // error, too many parameters  
let result3 = buildName("Bob", "Adams"); // okay and returns "Bob Adams"  
let result4 = buildName(undefined, "Adams"); // okay and returns "Will Adams"
```

Source : <http://typescriptlang.org/docs/handbook/functions.html>



- Les arguments « indénombrables »

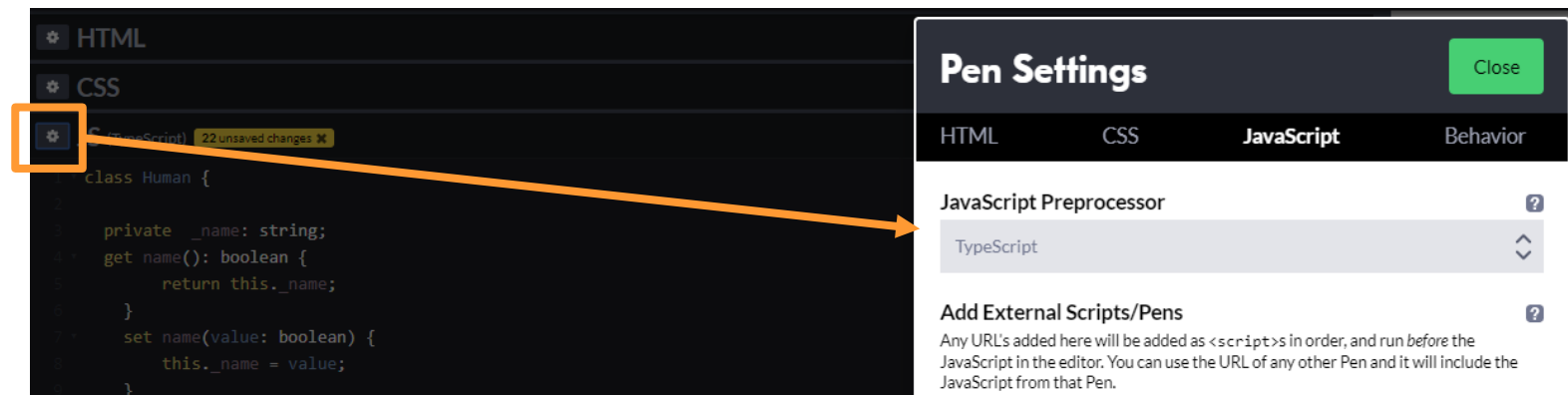
```
function buildName(firstName: string, ...restOfName: string[]) {  
    return firstName + " " + restOfName.join(" ");  
}  
  
// employeeName will be "Joseph Samuel Lucas MacKinzie"  
let employeeName = buildName("Joseph", "Samuel", "Lucas", "MacKinzie");
```

Source : <http://typescriptlang.org/docs/handbook/functions.html>

Avec « ... », on obtient un tableau
qui contient l'ensemble des paramètres passés

- Premier exercice : nous allons créer une classe **Human** avec différents attributs et méthodes
- Pour cet exercice, nous utiliserons **CodePen** car il offre un environnement clé en main pour exécuter du TypeScript

<https://codepen.io/>





- Nous n'aurons pas besoin des volets HTML et CSS
- Dans le volet *JS (TypeScript)*, créer une classe **Human** avec :
 - Un attribut « name » privé avec un getter et un setter
 - Un constructeur qui prend une chaîne en paramètre et qui initialise le nom de cet humain
- Après le code de la classe, vous pouvez écrire du code pour tester votre classe

C'est à vous

```
"use strict";

class Human {

    private _name: string;
    get name(): string {
        return this._name;
    }
    set name(value: string) {
        this._name = value;
    }

    constructor(birthName: string) {
        this.name = birthName;
    }
}

const human = new Human("Homer");
document.write(human.name);
```



- Ajouter à **Human** un attribut privé **children** de type « tableau de **Human** », sans getter ni setter
- Créer la méthode publique **addChildren** qui prend en paramètre un nombre indéterminé de chaînes de caractères
 - Cette méthode créera, pour chaque nom passé, un **Human** et l'ajoutera à la liste des enfants
- Créer la méthode publique **getChildren** qui retourne une copie de la liste des enfants

C'est à vous


```
"use strict";

class Human {

    private _name: string;
    get name(): string {
        return this._name;
    }
    set name(value: string) {
        this._name = value;
    }

    private _children: Human[] = [];

    constructor(birthName: string) {
        this.name = birthName;
    }

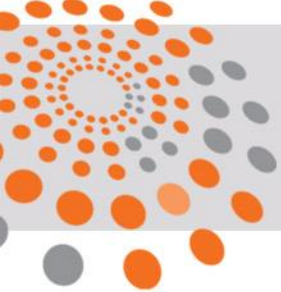
    public addChildren(...names: string[]) {
        names.forEach(name => this._children.push(new Human(name)));
    }

    public getChildren(): Human[] {
        return [...this._children];
    }
}
```



- Ajouter à **Human** une méthode publique **getChildrenHtml** qui retourne une chaîne de caractères contenant une liste HTML avec l'ensemble des noms des enfants
- Tester avec un code hors classe qui :
 - Crée un **Human** nommé *Homer*
 - Lui ajoute les enfants *Bart*, *Lisa* et *Maggie*
 - Affiche le nom d'*Homer*, puis la liste de ses enfants

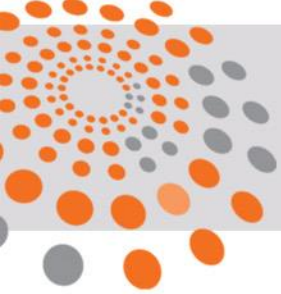
C'est à vous



```
public getChildrenHtml(): string {  
    return '<ul>' +  
        this.getChildren()  
            .map(child => '<li>' + child.name + '</li>')  
            .join('') +  
        '</ul>';  
}  
  
}  
  
const human = new Human("Homer");  
human.addChildren("Bart", "Lisa", "Maggie");  
document.write(human.name + human.getChildrenHtml());
```

Homer

- Bart
- Lisa
- Maggie

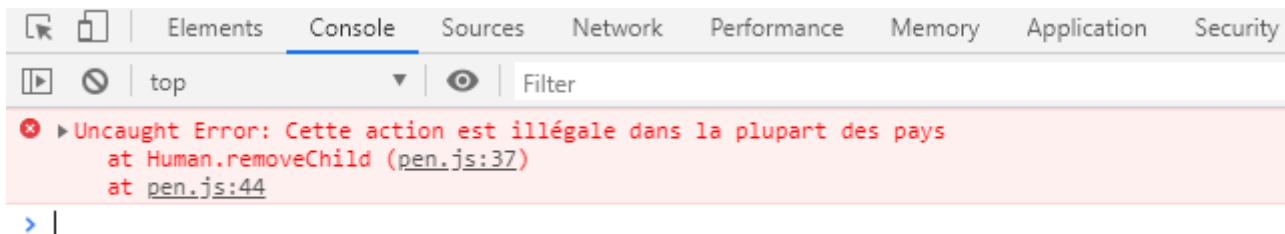


- Enfin, ajouter à Human une méthode publique **removeChildren**, qui prend un **Human** en paramètre, et qui lève une exception avec le message suivant :
« *Cette action est illégale dans la plupart des pays* »
- Tester en appelant la méthode

C'est à vous

```
public removeChild(child: Human) {  
    throw new Error('Cette action est illégale dans la plupart des pays');  
}  
  
}  
  
const human = new Human("Homer");  
human.addChildren("Bart", "Lisa", "Maggie");  
document.write(human.name + human.getChildrenHtml());  
human.removeChild(human.getChildren()[0]);
```

F12





- On souhaite que l'erreur apparaisse à l'écran, et dénonce Homer
- Modifier **removeChild** et complétez son appel pour afficher, à l'écran, le message suivant :
« Homer a tenté d'assassiner Bart. Le commissaire Wiggum est sur le coup. »

C'est à vous

```
public removeChild(child: Human) {  
    throw new Error(this.name + ' a tenté d\'assassiner ' + child.name + '. Le commissaire Wiggum est sur le coup.');
```



```
}  
  
const human = new Human("Homer");  
human.addChildren("Bart", "Lisa", "Maggie");  
document.write(human.name + human.getChildrenHtml());  
try {  
    human.removeChild(human.getChildren()[0]);  
} catch (e) {  
    document.write(e.message);  
}
```

Homer

- Bart
- Lisa
- Maggie

Homer a tenté d'assassiner Bart. Le commissaire Wiggum est sur le coup.



- On peut également écrire les chaînes de caractères avec des « backticks » (ou « backquote ») : `
- Cela rend parfois le code plus propre/plus clair
- L'important est de respecter les conventions établies sur votre projet

```
public removeChild(child: Human) {  
    throw new Error(`${this.name} a tenté d'assassiner ${child.name}. Le commissaire Wiggum est sur le coup.`);  
}
```


TypeScript

Corrigé

```
"use strict";

class Human {

    private _name: string;
    get name(): string {
        return this._name;
    }
    set name(value: string) {
        this._name = value;
    }

    private _children: Human[] = [];

    constructor(birthName: string) {
        this.name = birthName;
    }

    public addChildren(...names: string[]) {
        names.forEach(name => this._children.push(new Human(name)));
    }

    public getChildren(): Human[] {
        return [...this._children];
    }

    public getChildrenHtml(): string {
        return '<ul>' +
            this.getChildren()
                .map(child => '<li>' + child.name + '</li>')
                .join('') +
            '</ul>';
    }

    public removeChild(child: Human) {
        throw new Error(`${this.name} a tenté d'assassiner ${child.name}. Le commissaire Wiggum est`);
    }

}

const human = new Human("Homer");
human.addChildren("Bart", "Lisa", "Maggie");
document.write(human.name + human.getChildrenHtml());
try {
    human.removeChild(human.getChildren()[0]);
} catch (e) {
    document.write(e.message);
}
```

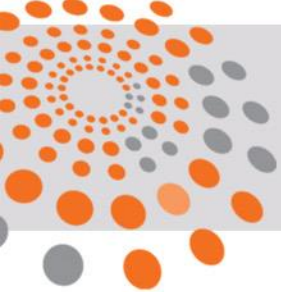


- Vous êtes désormais (quasiment) experts en TypeScript
- Il existe d'autres choses plus complexes, comme les **interfaces**, ou les **énumérations**... mais rien de dépayçant
- Nous pouvons maintenant nous lancer dans **Angular** !



ANGULAR : INITIALISATION





- **Angular** est un framework JavaScript créé par Google
- **AngularJS** a été créé en 2009
- En 2016, Google publie la v2 d'AngularJS, si différente qu'elle est baptisée sobrement « **Angular** »



- **Angular** est donc un framework jeune, qui évolue beaucoup



Mai 2016



Mars 2017



Novembre 2017



Mai 2018



Octobre 2018

Angular 8.0



Mai 2019

<https://update.angular.io/>



- Première étape : créer un nouveau projet
- Pour ce faire, nous utiliserons angular-cli installé précédemment
- Angular-cli met à disposition différentes commandes, avec plein de paramètres bien documentés

<https://angular.io/cli/new>

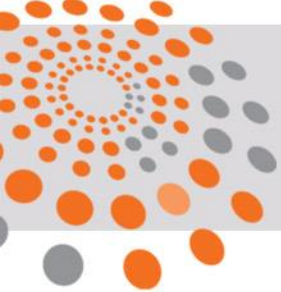


<https://angular.io/cli/new>

- `minimal`, `skipTests` : pour les projets perso
« *Tester, c'est douter* »
- `routing` : pour de la navigation (Attention : `false` par défaut)
- `style` : pour l'extension des fichiers style (css, sass, scss...)



- Angular-cli permet également de compiler (build) un projet Angular
- On utilise alors la commande `ng build` avec différentes options
 - `baseHref` : pour déployer sur un sous-dossier
 - `configuration` : pour indiquer l'environnement cible

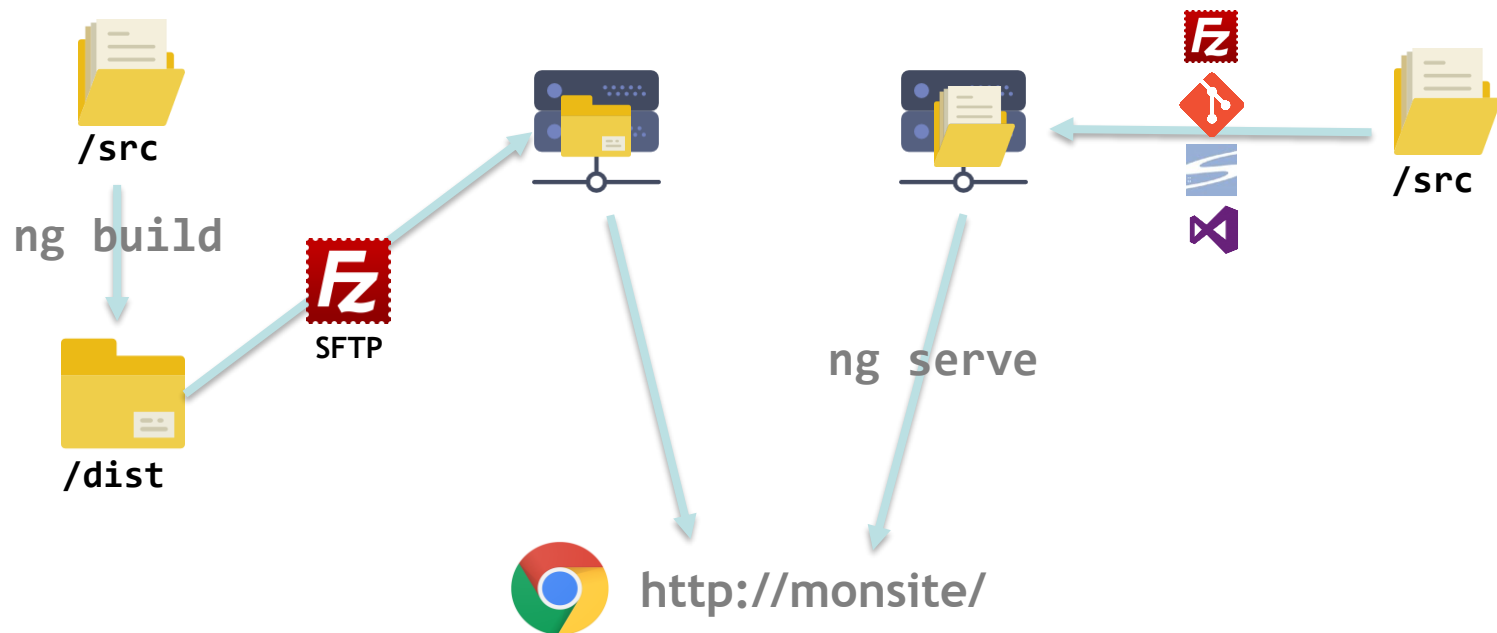


- Les sources ainsi compilées (en fichiers HTML, CSS, JS) se retrouvent dans un répertoire **dist** à la racine du projet
- Une fois le projet compilé, il suffit de transférer le contenu de ce répertoire **dist** sur un serveur web (Apache, tomcat, wamp...)
- De cette façon, il n'y a aucun prérequis à l'exécution d'une application Angular sur le serveur
 - *Puisque ce n'est que du HTML, CSS, JS*



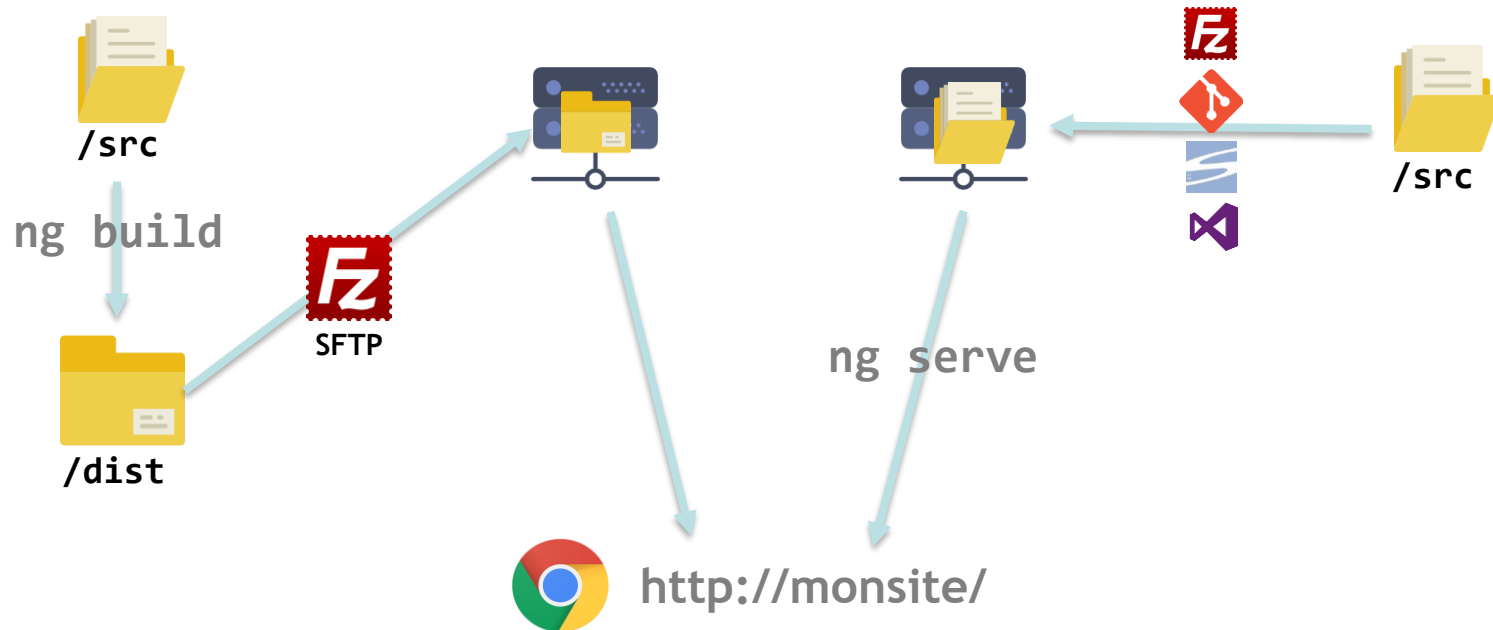
- Mais pour le développement, builder puis uploader serait un peu contraignant...
- Angular-cli intervient encore, avec la commande `ng serve`
- `ng serve` compile l'application (sans passer par le `dist`) puis l'exécute sur un serveur qui lui est propre (par défaut sur <http://localhost:4200>)
 - `baseHref`, `configuration` : voir déploiement
 - `liveReload` : recharge la page à chaque modification de fichier
 - `open` : ouvre le navigateur par défaut à l'adresse de l'application
 - `host`, `port` : pour configurer l'adresse de l'application

- Pour un déploiement externe (serveur de dev, qualif, prod...), on a deux options :



Angular : initialisation

Déploiement



Simple, ne nécessite aucune installation sur le serveur



Requiert des manipulations manuelles



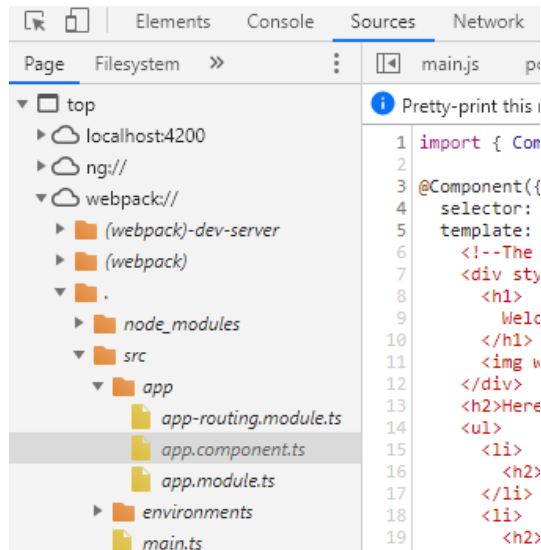
Automatisable, meilleure gestion de versions, intégration continue



Nécessite d'installer Node.js, angular-cli sur le serveur (voire Git, TFS, etc.)

The diagram illustrates the deployment workflow:

- /src** (Source Code) is processed by **ng build** to create **/dist** (Distribution).
- /dist** is transferred via **SFTP** to a **Server**.
- The **Server** runs **ng serve** to serve the application.
- The **Server** is connected to a **Reverse Proxy** (represented by a red 'Fz' icon and a blue 'X' icon).
- The **Reverse Proxy** serves the application from **/src**.
- The final output is accessible via the URL **http://monsite/** (indicated by a Chrome icon).



87



Angular : initialisation

Créons un premier projet !

- Ouvrir une invite de commandes et se placer dans le répertoire où vous souhaitez créer votre premier projet

```
C:\>
λ cd C:\Users\NicolasLETHUILLIER\Documents\Formations\Web+angular
C:\Users\NicolasLETHUILLIER\Documents\Formations\Web+angular>
λ ng new test --commit=false --minimal=true --routing=true --skipGit=true --skipTests=true --style=css|
```

- Le projet se crée et npm télécharge toutes les dépendances (répertoire `node_modules` à la racine)



Angular : initialisation

Créons un premier projet !

- Le projet est généré avec une page test
- On peut donc dès maintenant le lancer, et voir ce qui se passe

```
C:\Users\NicolasLETHUILLIER\Documents\Formations\Web+angular
λ cd test

C:\Users\NicolasLETHUILLIER\Documents\Formations\Web+angular\test
λ ng serve
10% building 3/3 modules 0 active i [wds]: Project is running at http://localhost:4200/webpack-dev-server/
i [wds]: webpack output is served from /
i [wds]: 404s will fallback to //index.html

chunk {main} main.js, main.js.map (main) 10.6 kB [initial] [rendered]
chunk {polyfills} polyfills.js, polyfills.js.map (polyfills) 251 kB [initial] [rendered]
chunk {runtime} runtime.js, runtime.js.map (runtime) 6.09 kB [entry] [rendered]
chunk {styles} styles.js, styles.js.map (styles) 16.3 kB [initial] [rendered]
chunk {vendor} vendor.js, vendor.js.map (vendor) 4.1 MB [initial] [rendered]
Date: 2019-08-22T12:38:23.410Z - hash: ac5f2940/e1541c97125 - time: 9809ms
** Angular Live Development Server is listening on localhost:4200, open your browser on http://localhost:4200/ **
i [wds]: Compiled successfully.
```

```
i [mqw]: c0mbij7e9 zncce22t0j7λ
** Vu8nj9u g1a6 DeleJobwneuf 2eL6eU i2 jiz7eUj7u8 ou j0c9jmozf:4500' obeu λonu p1om2eU ou mffb:\\j0c9jmozf:4500\
D9f6: 50Jd-08-25J15:28:52:4J05 - H92μ: 9C2f5d4016424Jc01453 - j7w6: 0800w2
cμnuk (A8uqou) A8uqou:J2' A8uqou:J2:w00 (A8uqou) 4:J w0 [tutit9:] [A8uq0e0q]
```

Angular : initialisation

Créons un premier projet !

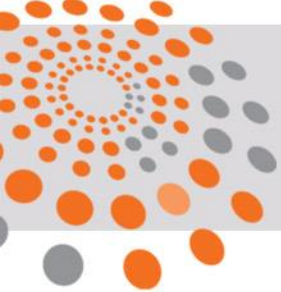
Welcome to test!



Here are some links to help you start:

- [Tour of Heroes](#)
- [CLI Documentation](#)
- [Angular blog](#)





Angular : initialisation

Créons un premier projet !

- Ouvrir la racine du projet avec votre IDE préféré

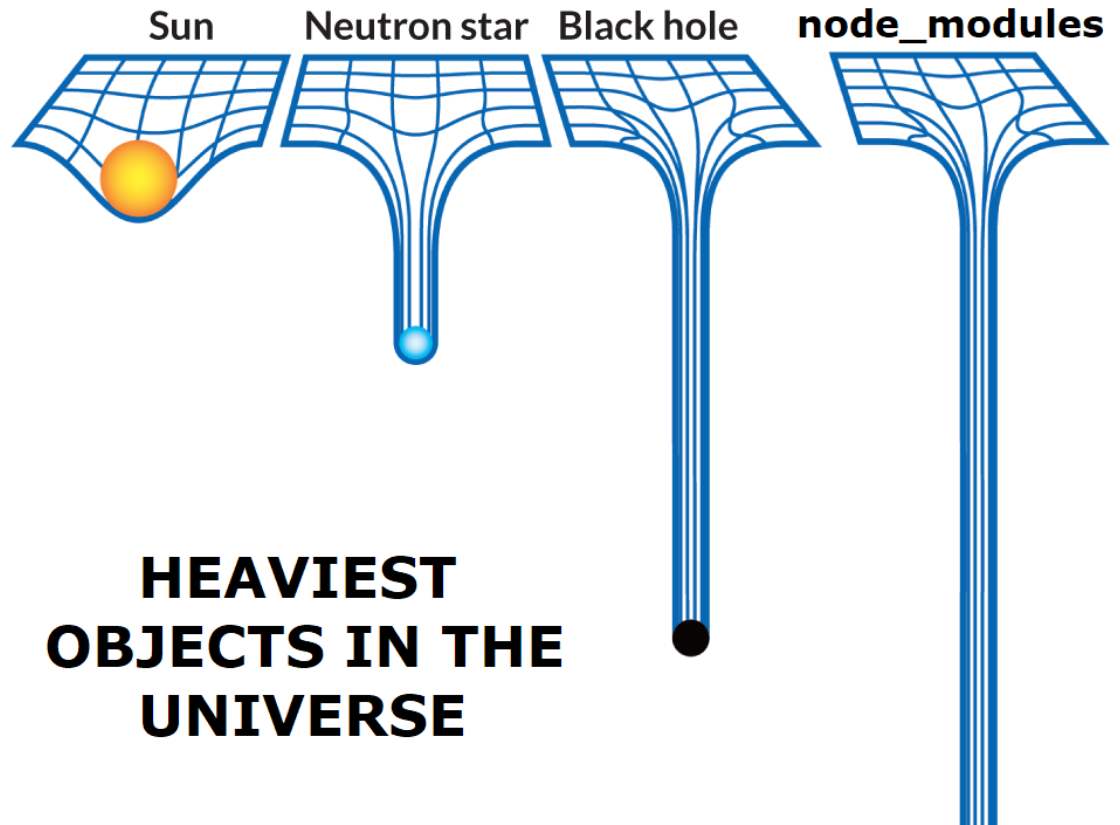
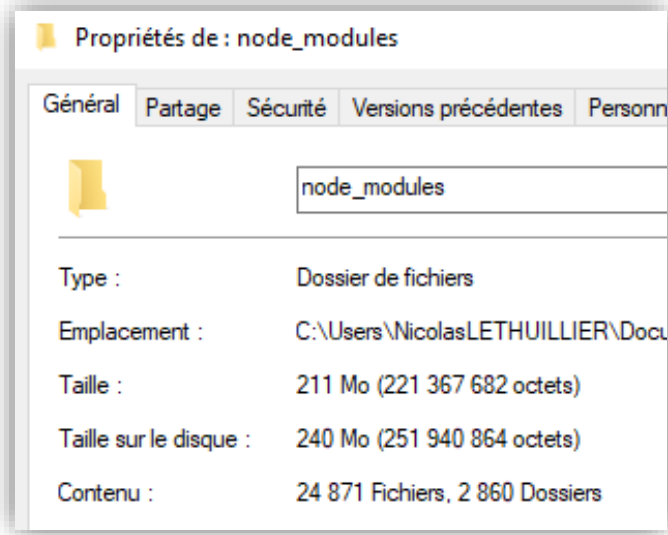
Rappel : Visual Studio Code ♥

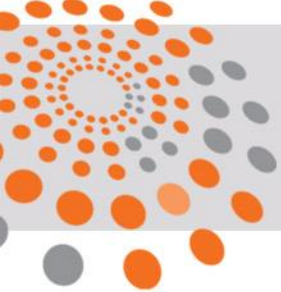
- Observons les fichiers à la racine...
 - **angular.json**
 - Configuration du projet, de build, des assets, etc.
 - **package.json**
 - Les commandes « raccourcies » npm
 - L'ensemble des dépendances de l'application

Si on exécute un `npm install -s myLibrary`, elle viendra s'ajouter ici
 - **node_modules**

Angular : initialisation

Créons un premier projet !

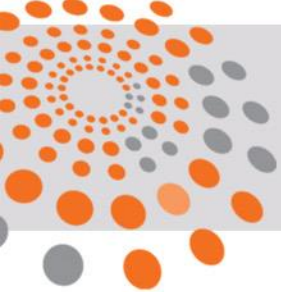




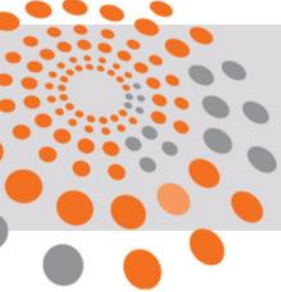
« *Il ne faut pas confondre Module et Module* »

Source : <http://www.learn-angular.fr/les-modules-angular/>

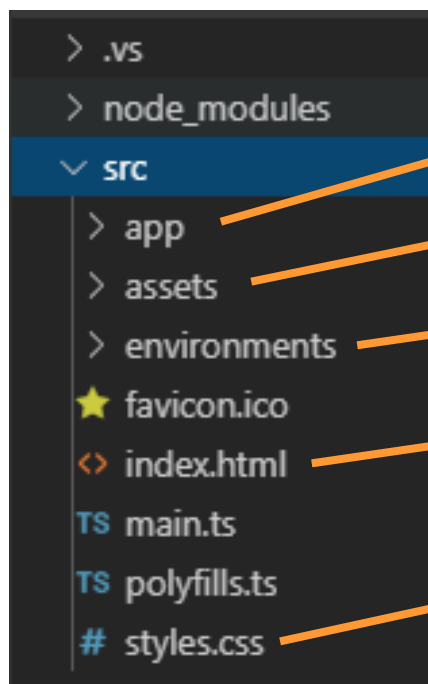
- Les modules au sens ES6
 - Ce qui est exporté et importé d'un élément à l'autre (mots-clés export, import)
- Les modules au sens Angular
 - Des classes « hub » qui regroupent l'ensemble des déclarations, imports de bibliothèques externes, etc. et indiquent le composant « de départ » de l'application



- Les modules au sens ES6
 - On retrouve notamment :
 - Les composants
 - Les services *à venir*
 - Les pipes *à venir*
 - Les directives *à venir*
- Les modules au sens Angular
 - On a en général **deux** modules
 - `AppModule` dans `app.module.ts`
 - `AppRoutingModule` dans `app-routing.module.ts`
 - Il arrive qu'on en crée d'autres pour découper l'application
 - Ex. : `admin.module.ts`, `backoffice.module.ts`



- Ouvrir le répertoire **src** du projet



Le code que vous écrivez

Les assets (images, fonts, etc.)

La configuration des environnements

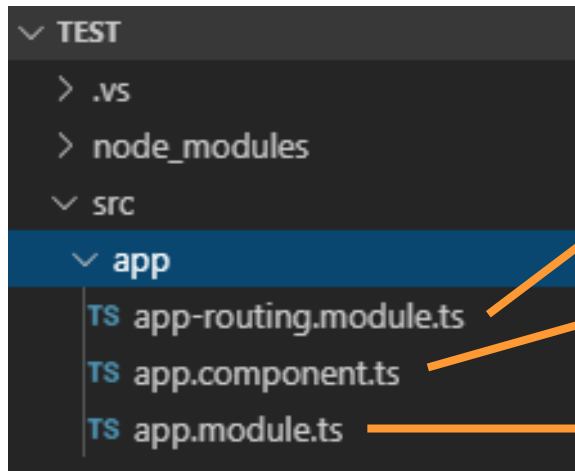
La page de base (que nous verrons plus tard)

La feuille de style qui s'appliquera
à tous les composants et toutes les pages

L'arborescence est libre

*Avec les fichiers de configuration,
on peut changer cette organisation*

- Ouvrir le répertoire **app**



Les routes de l'application

Le premier composant généré

Le fameux **AppModule**

Dans une application Angular, les conventions veulent que chaque module porte son nom en extension :

```
*.component.ts / *.component.html / *.component.css  
  *.module.ts  
  *.service.ts  
  *.pipe.ts
```

- Ouvrir le fichier `app.module.ts`

Les composants,
directives,
pipes
les « vues », le front

Les modules à importer
internes (routing) ou externes

Les services
mis à disposition des composants

Le composant de départ
la première page chargée

```
TS app.module.ts •
src > app > TS app.module.ts > ...
1  import { BrowserModule } from '@angular/platform-browser';
2  import { NgModule } from '@angular/core';
3
4  import { AppRoutingModule } from './app-routing.module';
5  import { AppComponent } from './app.component';
6
7  @NgModule({
8    declarations: [
9      AppComponent
10   ],
11   imports: [
12     BrowserModule,
13     AppRoutingModule
14   ],
15   providers: [],
16   bootstrap: [AppComponent]
17 })
18 export class AppModule { }
19
20
```



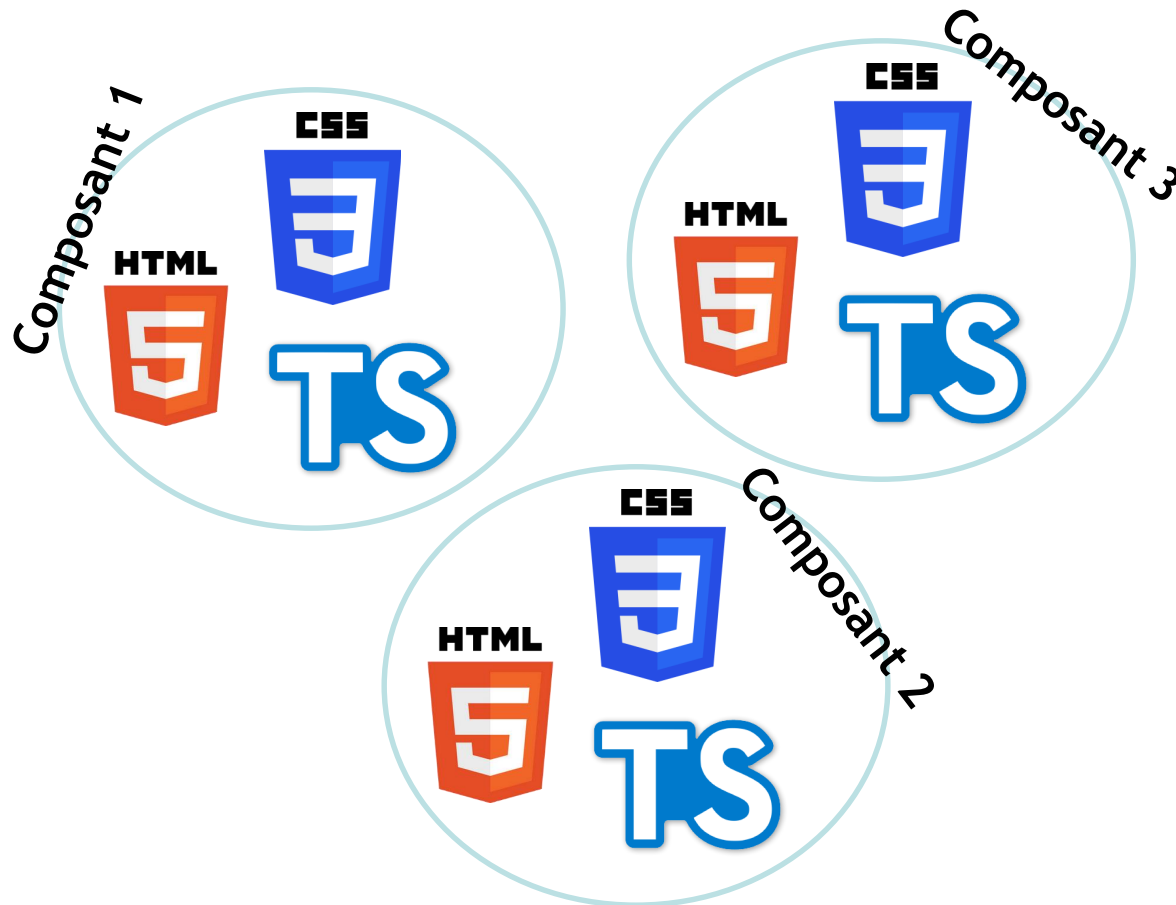
- On ne regarde pas le fichier de routing pour l'instant car il est vide, ce ne serait pas très parlant
- Maintenant qu'on a vu toute la mécanique d'Angular, on va s'intéresser aux **composants**, pour pouvoir créer des pages, des blocs, etc.
- Des questions ?



ANGULAR : COMPOSANTS



- Pour rappel, une application web, c'est :

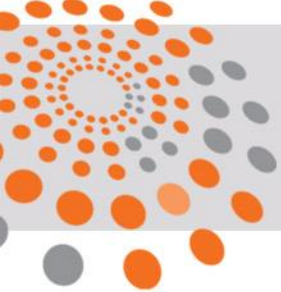


TS

Fichiers TypeScript



index.html



Angular : composants

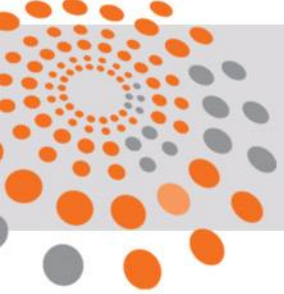
Racine de l'application

- Ouvrir le fichier `index.html` à la racine du projet

```
<> index.html x
src > <> index.html > ...
 1  <!doctype html>
 2  <html lang="en">
 3  <head>
 4    <meta charset="utf-8">
 5    <title>Test</title>
 6    <base href="/">
 7
 8    <meta name="viewport" content="width=device-width, initial-scale=1">
 9    <link rel="icon" type="image/x-icon" href="favicon.ico">
10  </head>
11  <body>
12    <app-root></app-root>
13  </body>
14  </html>
15
```

Le sous-dossier éventuel
d'exécution de l'app

La racine de l'app
→ là où le composant
« *bootstrap* » va venir
s'insérer au démarrage



Angular : composants

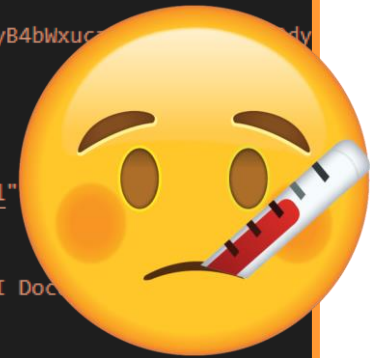

Premier composant

- C'est `AppComponent`, qui était renseigné en tant que `bootstrap` dans le `AppModule`
- C'est donc ce composant qui est inséré dans le `app-root`
- Ouvrir `app.component.ts`

Angular : composants

Premier composant

```
src > app > TS app.component.ts > ...  
1 import { Component } from '@angular/core';  
2  
3 @Component({  
4   selector: 'app-root',  
5   template: `  
6     <!--The content below is only a placeholder and can be replaced.-->  
7     <div style="text-align:center">  
8       <h1>  
9         Welcome to {{title}}!  
10      </h1>  
11      Tutorial</a>  
17      </li>  
18      <li>  
19        <a target="_blank" rel="noopener" href="https://angular.io/cli">CLI Doc</a>  
20      </li>  
21      <li>  
22        <a target="_blank" rel="noopener" href="https://blog.angular.io/">Angular blog</a></li>  
23    </ul>  
24    <router-outlet></router-outlet>  
25  `,  
26   styles: []  
27 })  
28  
29 export class AppComponent {  
30   title = 'test';  
31 }  
32
```





- Commençons par faire un peu de ménage, voulez-vous ?
- Dans le répertoire `app`, créer les fichiers :
 - `app.component.html`
 - `app.component.css`
- Copier le HTML de `app.component.ts` (ce qui est entre les quotes) dans `app.component.html`
- Remplacer « `template:` » par « `templateUrl:` » et indiquez le chemin du fichier
- Remplacer « `styles: []` » par « `styleUrls:` » en indiquant le chemin de la feuille de style (dans un tableau !)



Angular : composants

Premier composant

- Voilà à quoi ressemble, classiquement, un composant :

```
<> index.html TS app.component.ts ●
src > app > TS app.component.ts > ...
1  import { Component } from '@angular/core';
2
3  @Component({
4    selector: 'app-root',
5    templateUrl: './app.component.html',
6    styleUrls: ['./app.component.css']
7  })
8  export class AppComponent {
9    title = 'test';
10
11    // Constructors, methods, etc.
12  }
13
```



Angular : composants

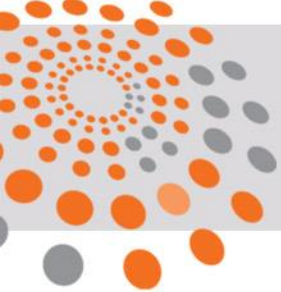
Premier composant

- Observons le fichier HTML du composant...

```
<> index.html    TS app.component.ts    <> app.component.html
src > app > <> app.component.html > ...
1  <!--The content below is only a placeholder and can be replaced.-->
2  <div style="text-align:center">
3      <h1>
4          Welcome to {{title}}!
5      </h1>
6      Tour of Heroes</a></h2>
12     </li>
13     <li>
14         <h2><a target="_blank" rel="noopener" href="https://angular.io/cli">CLI Documentation</a></h2>
15     </li>
16     <li>
17         <h2><a target="_blank" rel="noopener" href="https://blog.angular.io/">Angular blog</a></h2>
18     </li>
19 </ul>
20 <router-outlet></router-outlet>
21
```

Appel à un attribut de la classe TS

Entrée du routing



Angular : composants

Binding (1/2) : les bases

- {{ ... }} permet d'utiliser un attribut (public) de la classe TS

```
1 <!--The content below is only a placeholder and can be replaced.-->
2 <div style="text-align:center">
3   <h1>
4     Welcome to {{title}}!
5   </h1>
6   
7 </div>
```

```
✓ @Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
✓ export class AppComponent {
  title = 'test';

  // Constructors, methods, etc.
}
```

Dans les accolades, on peut même écrire du JavaScript



```
<div style="text-align:center">
  <h1>
    Welcome to {{title.toUpperCase()}}!
  </h1>
  
3      <h1>
4      | Welcome to {{getTitleFromDatabase()}}!
5      </h1>
6      
  <h1>
    Welcome to {{title}}!
  </h1>
  <img width="{{customWidth}}" alt="Angular Logo" sr
</div>
<h2>Here are some links to help you start: </h2>
<ul>
```

- Qu'on peut écrire également :

```
<!--The content below is only a placeholder and can be
<div style="text-align:center">
  <h1>
    Welcome to {{title}}!
  </h1>
  <img [width]="customWidth" alt="Angular Logo" src=
</div>
<h2>Here are some links to help you start: </h2>
<ul>
```

attr='{{ ... }}'

et

[attr]='...'

sont strictement équivalents

- Nous parlerons plus tard des **directives**, mais trois d'entre elles nous intéressent déjà :
 - *ngIf
 - *ngFor
 - *ngSwitch
- Ces termes se placent dans les balises HTML pour ajouter de la mécanique à l'affichage

```
<h1 *ngIf="title">
  Welcome to {{title}}!
</h1>
```

```
<ul>
  <li *ngFor="let link of links">{{link.url}}</li>
</ul>
```

Attribut de la classe

Angular : composants

ngIf, ngFor, ngSwitch

```
<h1 *ngIf="title">
  Welcome to {{title}}!
</h1>
```

```
<ul>
  <li *ngFor="let link of links">{{link.url}}</li>
</ul>
```

- Attention : un ***ngIf** et un ***ngFor** ne peuvent pas se trouver sur la même balise
- Pour pallier cette lacune, il existe la balise **ng-container**

```
<ng-container *ngIf="canDisplayTiles()">
  <div *ngFor="let tile of tiles" [width]="tile.width" [height]="tile.height">{{tile.name}}</div>
</ng-container>
```

- **ng-container** n'apparaît pas dans le HTML final et n'a aucune incidence sur le DOM



- Il est temps de créer notre premier composant !
- Pour ce faire, deux méthodes :
 - A la main, en créant les fichiers, en les remplissant *from scratch* et en mettant à jour le **AppModule**
Permet de bien comprendre la mécanique, mais un peu fastidieux
 - Avec angular-cli : **ng generate component 'monComposant'**
- angular-cli crée automatiquement le composant dans **/app** suivi de l'arborescence éventuellement précisée, avec les fichiers TS, HTML et CSS qui vont bien puis ajoute le composant à l'**AppModule**



- Utilisons donc la méthode la plus simple et créons un composant **custom-button**

ng generate component custom-button

- Et allons voir un peu à quoi ça ressemble...

custom-button.component.ts

```
import { Component, OnInit } from '@angular/core';

@Component({
  selector: 'app-custom-button',
  templateUrl: './custom-button.component.html',
  styleUrls: ['./custom-button.component.css']
})
export class CustomButtonComponent implements OnInit {

  constructor() { }

  ngOnInit() {
  }

}
```

`<p>custom-button works!</p>`

custom-button.component.html

```
import { AppComponent } from './app.component';
import { CustomButtonComponent } from './custom-button.component';

@NgModule({
  declarations: [
    AppComponent,
    CustomButtonComponent
  ],
  imports: [
```

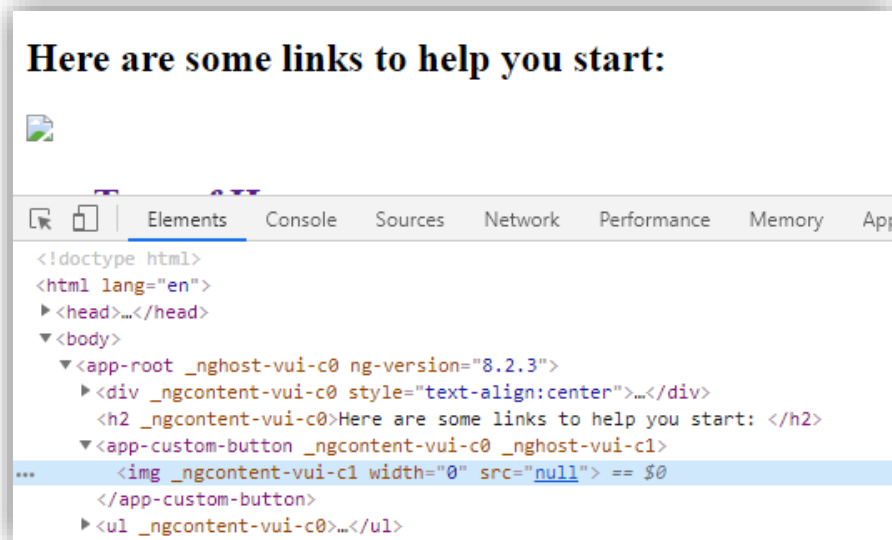
app-module.ts



- On peut dès à présent utiliser notre nouveau composant dans un autre composant, en utilisant le selector défini dans le décorateur (ici « `app-custom-button` »)
- Ouvrir `app.component.html`
- Ajouter quelque part :
`<app-custom-button></app-custom-button>`
- Tester



- Modifier le composant **custom-button** pour lui ajouter :
 - Un attribut « **width** » public de type nombre
 - Un attribut « **source** » public de type chaîne de caractères
- Modifier le template de **custom-button** pour y mettre une image dont la largeur sera **width** et la source **source**
- Tester





- Initialiser `width` et `source` avec les valeurs de votre choix
- Constater le changement
- Corsons un peu la chose :
 - Créer un attribut public `links` qui sera une liste d'objets avec pour propriétés « `title` » et « `target` »
 - Créer une méthode publique `generateLinks` qui remplit cette liste avec des liens quelconques
 - Dans le template (HTML), ajouter une liste qui ne s'affiche que lorsque `links` existe et possède des éléments (`*ngIf`), et qui contient autant d'items avec des liens qu'il y a d'éléments dans `links` (`*ngFor`)
 - Ajouter à l'image `(click)='generateLinks()'`
 - Tester !