Tutoriel pour apprendre Angular 4 pas à pas

Par Lamine BENDIB

Date de publication : 28 octobre 2017

Que vous soyez nouveau sur AngularJS (v1.x) ou sur Angular (v2+), vous souhaitez apprendre à maîtriser rapidement les *components*, modules et les services d'Angular.

Ce tutoriel fournit les bases dont vous avez besoin pour commencer à créer une application Angular. Nous parlerons de ce que cette version apporte de plus par rapport aux précédentes, nous allons-nous focaliser sur le fonctionnement d'Angular 4 et principalement sur le composant de base qui est le « component ». Nous répondrons également aux questions clés comme qu'est-ce qu'un component ? Quand devrions-nous utiliser le *data binding* ? Pourquoi avons-nous besoin d'un service ? Et comment construire une application Angular ?

À la fin de ce tutoriel, vous aurez les bases nécessaires pour créer une application Angular.

Commentez

I - Pourquoi utiliser Angular ?

Il existe de nombreux Frameworks JavaScript (React, Vue, Ember…). Au-delà du choix personnel, voici ce qu'apporte Angular :

- Angular rend notre HTML plus expressif en y ajoutant des conditions, des boucles et des variables ;
- Grâce au *data binding* nous pouvons facilement afficher des champs à partir de nos models, vérifier les changements et mettre à jour la page en conséquence ;
- Angular prône la modularité ; nos applications sont sous forme de blocks réutilisables ;
- Angular intègre par défaut le support de communication avec les services back-end.

II - Angular changelog

Avec l'arrivée d'Angular 2 et 4, le Framework portera le nom d'*Angular*, *AngularJs* sera utilisé pour les versions 1.x. Google utilisera dorénavant la **SEMVER** (Semantic versioning) pour son Framework.

Angular n'a pas eu de version 3 tout simplement parce que le package @angular/router était en v3 alors que tous les autres étaient en v2 ; les développeurs ont donc décidé de passer directement en v4.

Toujours plus petit et plus rapide

Le but de la team d'Angular était de rendre les applications plus compactes et plus rapides en :

- réduisant la taille du code généré de 60 %;
- sortant les animations du package *@angular/core* ; plus besoin d'importer les animations si on ne les utilise pas.

Nouvelles fonctionnalités

- Amélioration du `*ngIf` :
  on peut désormais utiliser la syntaxe *else*, ce qui n'était pas possible avant :

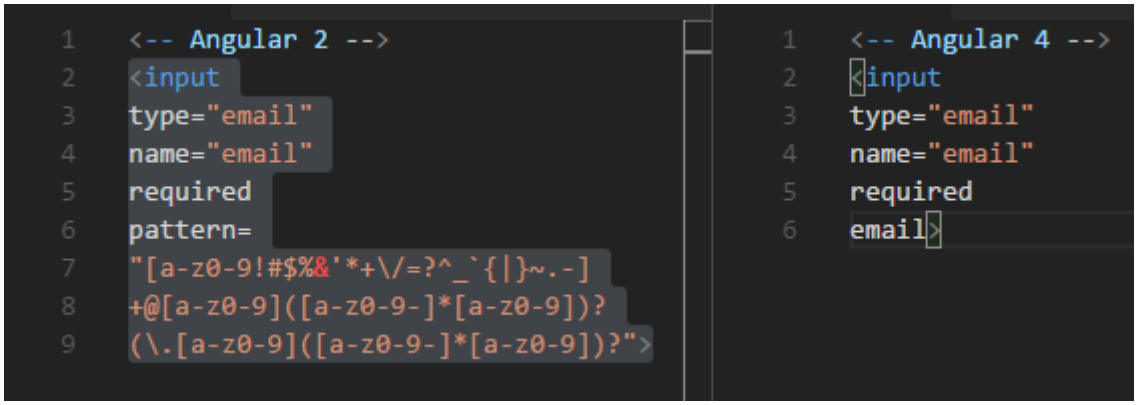
ngIf-2.html

```
<-- Angular 2 -->
<p *ngIf="condition">
    Condition is true
</p>
<p *ngIf="!condition">
    Else statement
</p>
```

ngIf-4.html

```
<-- Angular 4 -->
<p *ngIf="condition;else templateName">
    Condition is true
</p>
<ng-Template #templateName>
    <p>
        Else statement
    </p>
</ng-Template>
```

- Ajout de la validation des e-mails :
  la validation d'e-mail est maintenant gérée par Angular, plus besoin d'utiliser le pattern



```
<-- Angular 2 -->
<input
type="email"
name="email"
required
pattern=
"[a-z0-9!#$%&'*+\/=?^_`{|}~.-]
+@[a-z0-9]([a-z0-9-]*[a-z0-9])?
(\.[a-z0-9]([a-z0-9-]*[a-z0-9])?">
```

```
<-- Angular 4 -->
<input
type="email"
name="email"
required
email>
```

- Compatibilité TypeScript 2.1 et 2.2.

III - Les bases

Avant de commencer à développer, nous allons d'abord choisir le langage que l'on utilisera et l'éditeur qui supportera entièrement ce langage, puis nous parlerons des modules et de ce qu'ils signifient en Angular.

III-A - Choix du langage

Étant donné qu'Angular est un framework JavaScript, nous pouvons utiliser tous les langages JavaScript compilés existants, le plus utilisé étant **TypeScript** (Angular est d'ailleurs développé en TypeScript).
Voici quelques spécificités de TypeScript :

- open source ;
- sur ensemble de JavaScript ;
- transcompile le code en JavaScript ;
- typage puissant (toutes les classes, fonctions, variables peuvent avoir un type) ;
- approche objet basée sur les classes et non sur les prototypes.

Pour plus d'information sur TypeScript, suivez ce **lien** et si vous avez envie de faire joujou avec ce langage je vous invite au **playground**.

III-B - Choix de l'IDE

Énormément d'éditeurs supportent TypeScript, nativement ou grâce à un plugin. Personnellement j'utilise **« Visual Studio Code »** et pour faciliter le développement Angular, j'utilise en plus de cela une **extension Vs Code** faite par **John Papa**, un poids lourd du milieu.
Si vous souhaitez avoir d'autres outils pour développer, je vous conseille cet **article** qui en répertorie quelques-uns.

III-C - Préparation de l'environnement

L'installation de notre environnement de développement Angular se fait en deux étapes.

III-C-1 - Installation de Node.js et npm (Node Package Manager)

npm est un gestionnaire de paquets de Node.js qui nous permet d'installer des bibliothèques, des paquets et des applications.
Nous l'utiliserons afin d'installer toutes les bibliothèques pour Angular ainsi que pour exécuter les scripts qui transcompilent notre code et lancent notre application.

III-C-2 - Mise en place de l'application

Pour cela il faut :

- créer le dossier de l'application ;
- ajouter le *package.json* ainsi que les fichiers de configuration ;
- installer les paquets ;
- créer le *root* de l'application (Angular Module) ;
- créer le *main.ts* qui permet de charger le module *root* ;
- créer la page web, généralement *index.html*.

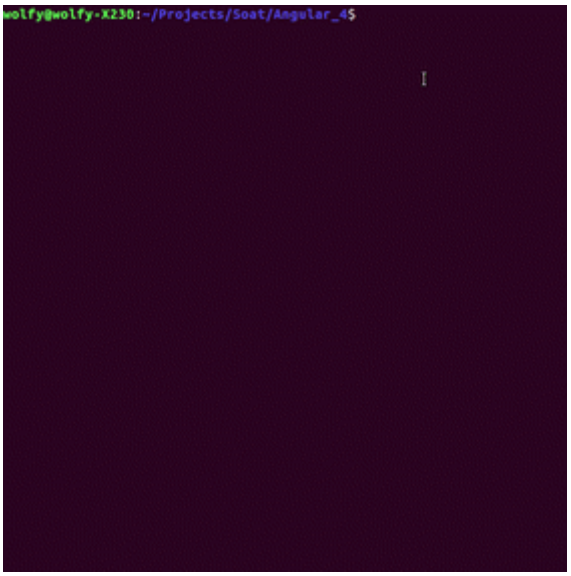
Il existe différentes façons de procéder pour réaliser ces étapes :

1. Manuellement, en se basant sur le **quickstart** fourni par Google ou directement en récupérant le résultat sur **Github** (ne vous fiez pas au titre, le projet a été mis à jour et tourne sous Angular 4. Pour vérifier cela, regardez le fichier *package.json* à la racine) ;
2. En utilisant **Angular Cli** : c'est la façon la plus simple et la plus rapide pour créer un squelette de projet.

Nous utiliserons Angular Cli pour la suite de ce tutoriel ; le **projet utilisé** se trouve sur Github.

III-D - Création et lancement du projet

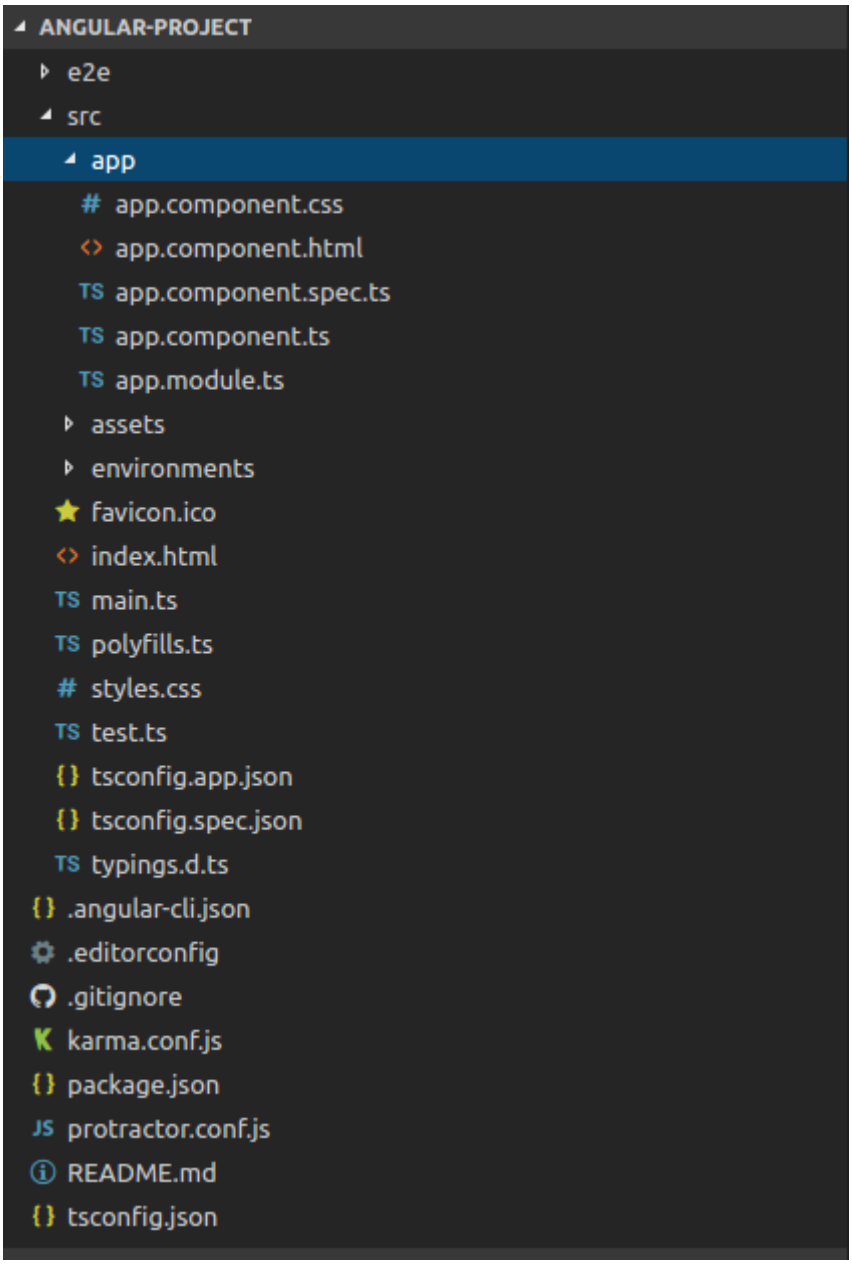
Nous devons tout d'abord installer Angular Cli avec la commande suivante `ng install -g @angular/cli`. Après cela nous pourrons créer notre premier projet avec la commande `ng new angular-project`.



Voilà, vous avez un projet Angular 4. Pour le lancer, rien de plus simple : il suffit de faire `cd angular-project`, `npm install` ensuite `ng serve -o`.

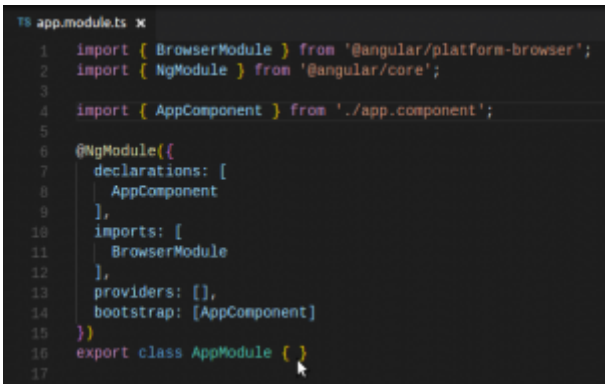
IV - Anatomie d'un projet Angular

Voici à quoi ressemble le squelette d'un projet Angular :



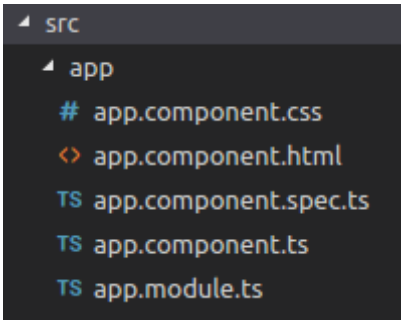
Le code du projet vit dans src/app. C'est ici que nous allons créer des composants, services et modèles.

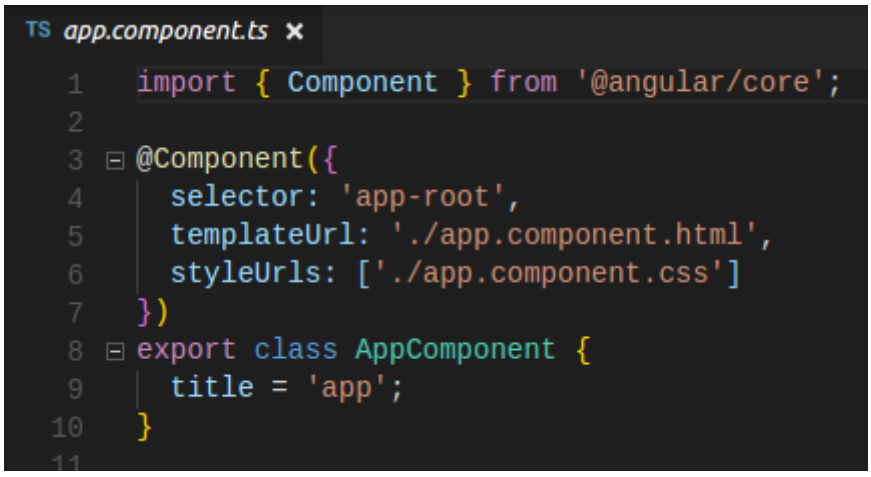
App.module est le module racine (*root module*) qui permet l'organisation et la compilation du reste de l'application. Nous y déclarerons les *components* que nous allons créer pour qu'ils puissent être utilisés dans toute l'application. Nous verrons plus tard comment créer nos propres modules.



Les propriétés à l'intérieur du décorateur `@ngModule` seront mises à jour, à mesure que l'application grossit. Par exemple lors de la création d'un composant, celui-ci devra être inclus dans le tableau des imports.
Pour plus d'informations, vous trouverez tout ce qu'il vous faut dans la doc d'Angular : **Bootstrapping**, **Architecture**, **NgModule**.

AppComponent (app.component.ts, app.component.html, app.component.css) est le composant racine (*root component*). Il s'agit de la vue principale de l'application ; elle comprendra des sous-composants qui se focaliseront sur des tâches spécifiques, car nous ne pouvons pas y mettre toutes les fonctionnalités de l'application.

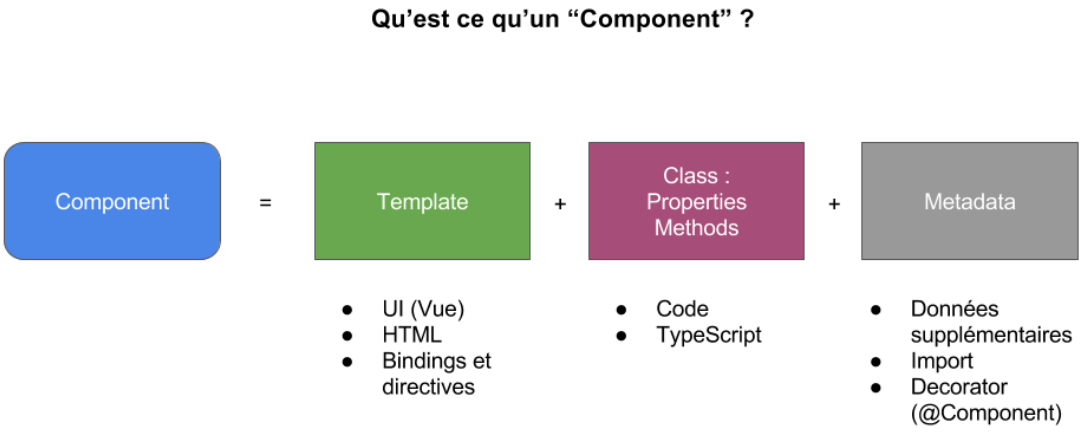




Mais c'est quoi au juste un « component » ?

V - Introduction au component

V-A - Qu'est-ce qu'un component ?



Un « component » Angular contient :

- un **template** contenant l'interface utilisateur en HTML. Nous utiliserons le *databinding* afin de rendre la vue dynamique ;
- une **Class** contenant le code associé à la vue, des propriétés et méthodes logiques qui seront utilisées dans la vue ;
- des **metadata** nous permettant de définir la classe comme étant un composant Angular (*component*).

Voici à quoi ressemble un composant Angular :

```
1. import { Component } from '@angular/core';
2.
3. @Component({
4.         selector: 'soat-products',
5.         templateUrl: './product.component.html'
6. })
7.
8. export class ProductComponent {
9.         pageTitle: string = "Products Page"
10.
11.         someFunction(){
12.                 //Do your stuff here
13.         }
14.
15. }
```

Nous allons décortiquer ce composant.

V-B - Création de la classe

```
1. /*export: to use the class elsewhere
2. class: keyword to create a class
3. ProductComponent: class name*/
```

```
4. export class ProductComponent {
5.     //PropertyName: Data type = "Default value"
6.         pageTitle: string = "Products Page"
7. }
```

On crée une classe en utilisant le mot-clé **class**. Par convention, le nom de la classe porte le nom de la fonctionnalité, suivi de « Component ».
Le mot-clé **export** signifie que la classe peut être utilisée ailleurs.
Dans cet exemple, nous avons défini une propriété de type *string* que nous pouvons utiliser dans la vue.

V-C - Définition des metadatas avec le décorateur

Une classe devient un composant Angular lorsque nous lui ajoutons le *metadata* adéquat, ce metadata étant défini grâce à la fonction « Component ». Avec TypeScript, on utilise ce que l'on appelle un *Décorateur* (Decorator). Un décorateur est une fonction qui ajoute des metadatas dans la classe, elle est toujours préfixée d'un **@**.

```
1. //the component decorator
2. @Component({
3.         selector: 'soat-products', //Directive name used in the view
4.         templateUrl: './product.component.html' //the view of this component
5. })
```

- **selector** : définit le nom de la directive du composant, qui sera utilisé dans la vue.
- **templateUrl** : définit le chemin relatif vers la vue gérée par ce composant.

V-D - Les imports

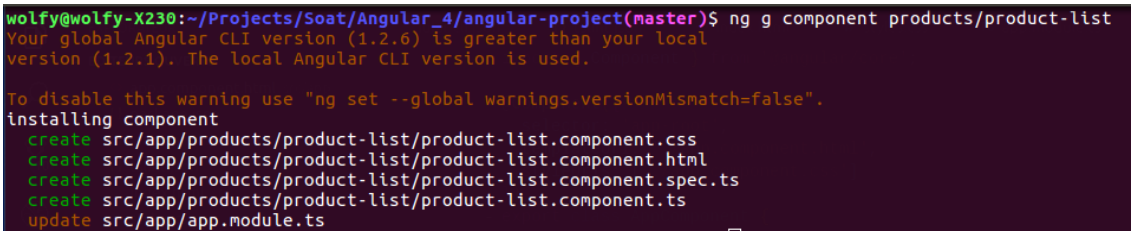
Avant d'utiliser une classe externe, il faut l'importer.

```
//import keyword {The member we need} from 'the module containing the member'
import { Component } from '@angular/core';
```

V-E - Mise en pratique

Nous allons maintenant créer notre premier composant Angular. Pour cela nous allons utiliser Angular Cli, mettez-vous à la racine de votre projet et tapez :

```
ng g component products/product-list
```

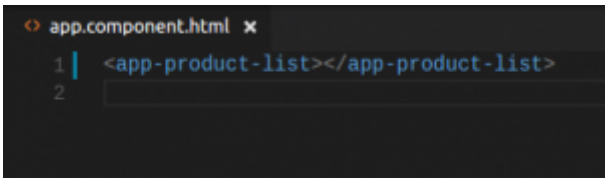


Cette commande nous permet de créer tous les fichiers liés à notre *component*, tout en mettant à jour le fichier *app.module.ts*, et ce, en déclarant le nouveau *component* créé :

```
1. import { ProductListComponent } from './products/product-list/product-list.component';
2.
3. @NgModule({
4.     declarations: [
5.         AppComponent,
6.         ProductListComponent //component declaration
7.     ],
```

```
8.      imports: [
9.          BrowserModule, FormsModule
10.      ],
11.      providers: [],
12.      bootstrap: [AppComponent]
13. })
```

Voilà, nous avons notre premier *component*. Pour l'utiliser, il suffit d'ajouter la balise dans la vue de notre composant racine *app.component.html* en utilisant le ***selector***



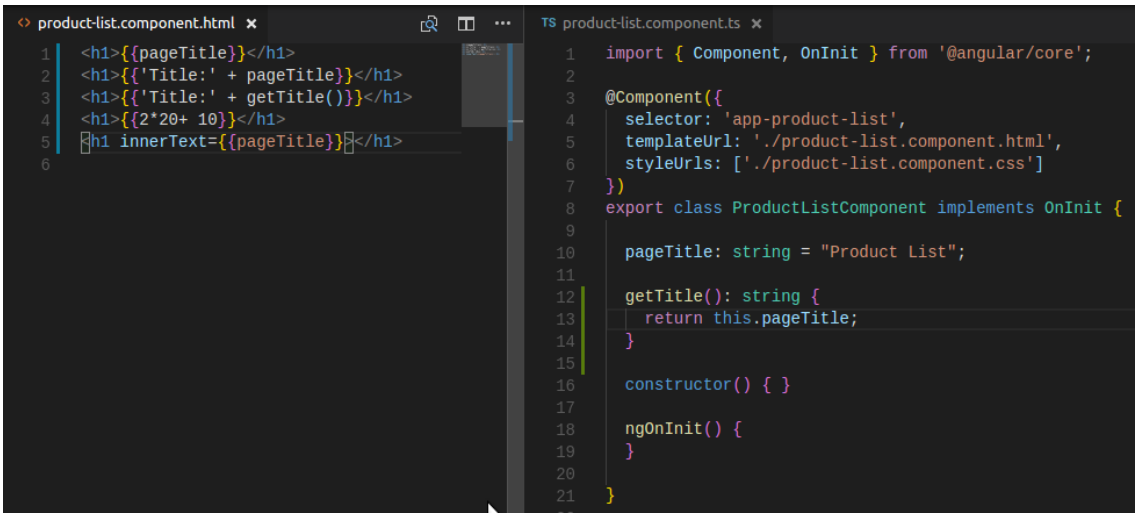
En lançant notre serveur (`ng serve -o`), nous affichons le contenu de *product-list.component.*
Pour le moment nous affichons uniquement du texte en dur, le but étant de pouvoir envoyer des données depuis le composant vers la vue. Ceci est faisable grâce au ***data binding***.

VI - Data binding

Angular a défini quatre techniques de *Data binding* pour synchroniser le *template* et le *component*. Il est ainsi possible de propager une donnée du *component* vers le DOM et du DOM vers le *component*. Ces formes de *data binding* sont décrites ci-après.

VI-A - L'interpolation {{…}}

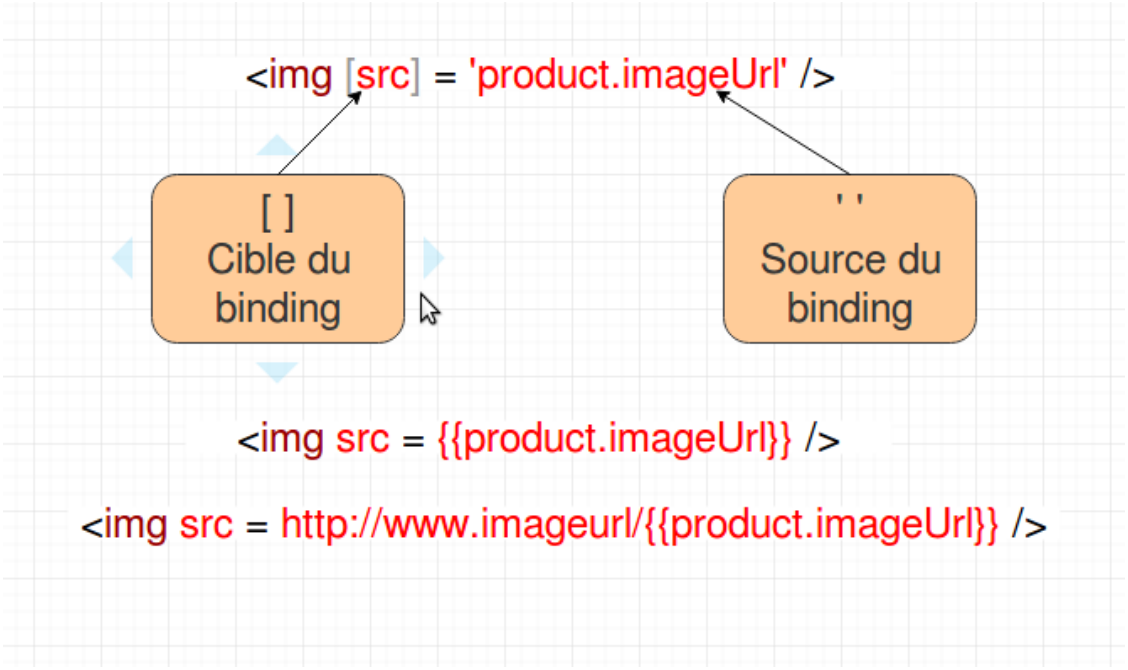
L'interpolation est un binding **unidirectionnel** qui permet de modifier le DOM à partir du modèle si un changement est intervenu sur une valeur de ce dernier. Elle est réalisée grâce à la double accolade {{ }} comme le montre l'image suivante :



L'interpolation peut être faite entre deux balises HTML ou au niveau de l'attribut, elle permet aussi de faire des calculs arithmétiques et de la concaténation.

VI-B - Le property binding […]

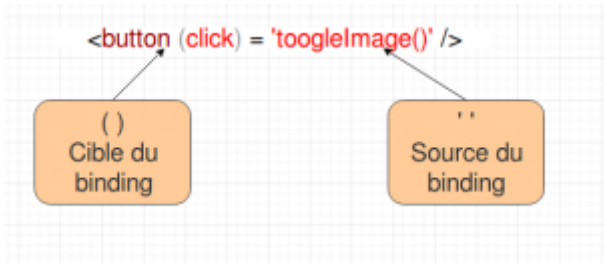
Permet de valoriser une propriété d'un composant ou d'une directive à partir du modèle si un changement est intervenu sur une valeur de ce dernier.



Pour effectuer un binding au niveau de l'attribut d'une balise HTML, il suffit de le mettre entre deux crochets [ ]. Quant à la source du binding, elle doit être entre deux quotes ' '.
Nous pouvons également utiliser l'interpolation, mais par convention mieux vaut utiliser le binding d'attributs, sauf dans le cas où la valeur source ne représente qu'une partie de la valeur de l'attribut, comme dans le dernier exemple.

VI-C - L'event binding (…)

Permet d'exécuter une fonction portée par un *component* suite à un événement émis par un élément du DOM.



C'est le même principe que pour le binding d'attributs sauf qu'ici au lieu d'utiliser des crochets [ ] sur un attribut, nous utiliserons des parenthèses ( ) autour d'un événement et nous faisons appel à une fonction dans notre classe.

VI-D - Le two-way data binding [(…)]

C'est une combinaison du property binding et de l'event binding sous une unique annotation. Dans ce cas-là, le component se charge d'impacter le DOM en cas de changement du modèle et le DOM avertit le *component* d'un changement via l'émission d'un événement.
La syntaxe utilisée dans Angular pour déclarer un tel d*ata binding* est la suivante : [( )]. Pour vous en souvenir, sachez que cette annotation se nomme *« banana in the box »*.

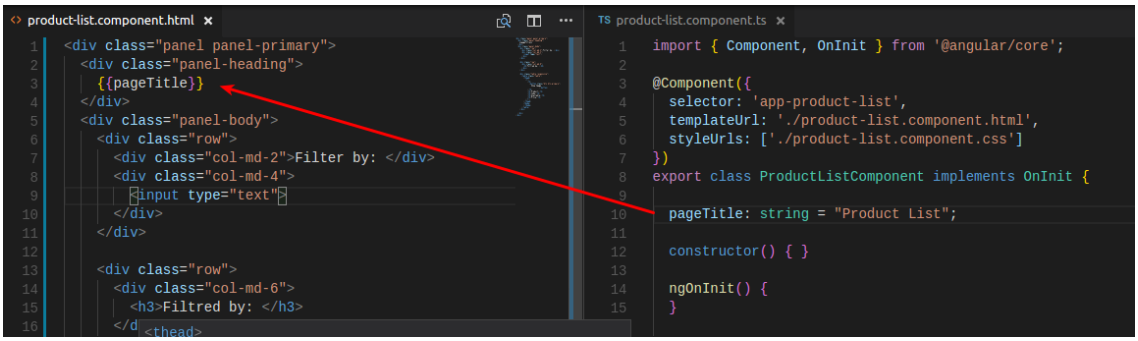
Pour plus de détails, vous trouverez une fois de plus tout ce qu'il vous faut dans la doc d'Angular sur le **Data-Binding**.

VI-E - Mise en pratique

Nous allons maintenant utiliser ce que nous avons appris pour afficher une liste de produits dans un tableau. Pour cela nous allons ouvrir product-list.component.html et créer notre template comme ceci :

```
<div class="panel panel-primary">
    <div class="panel-heading">
        Product List page
    </div>
    <div class="panel-body">
        <div class="row">
            <div class="col-md-2">Filter by: </div>
            <div class="col-md-4">

            </div>
        </div>

        <div class="row">
            <div class="col-md-6">
                <h3>Filtred by: </h3>
            </div>
        </div>

        <div class="table-responsive">
            <table class="table">
                <thead>
                    <tr>
                        <th>
                            <button class="btn btn-primary">
                                Show Image
                                                    </button>
                        </th>
                        <th>Product</th>
                        <th>Code</th>
                        <th>Available</th>
                        <th>Price</th>
                    </tr>
                </thead>
                <tbody>
                </tbody>
            </table>
        </div>

    </div>
</div>
```

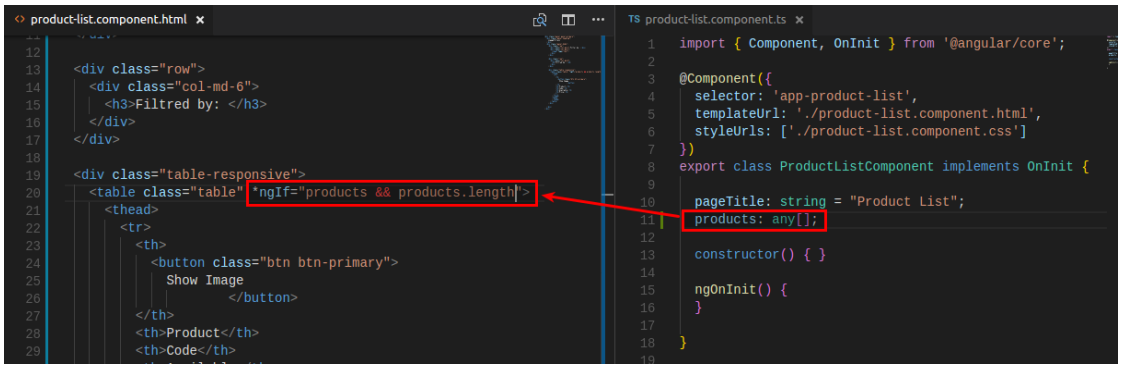
Nous allons maintenant utiliser l'interpolation pour binder le titre de la page comme ceci :



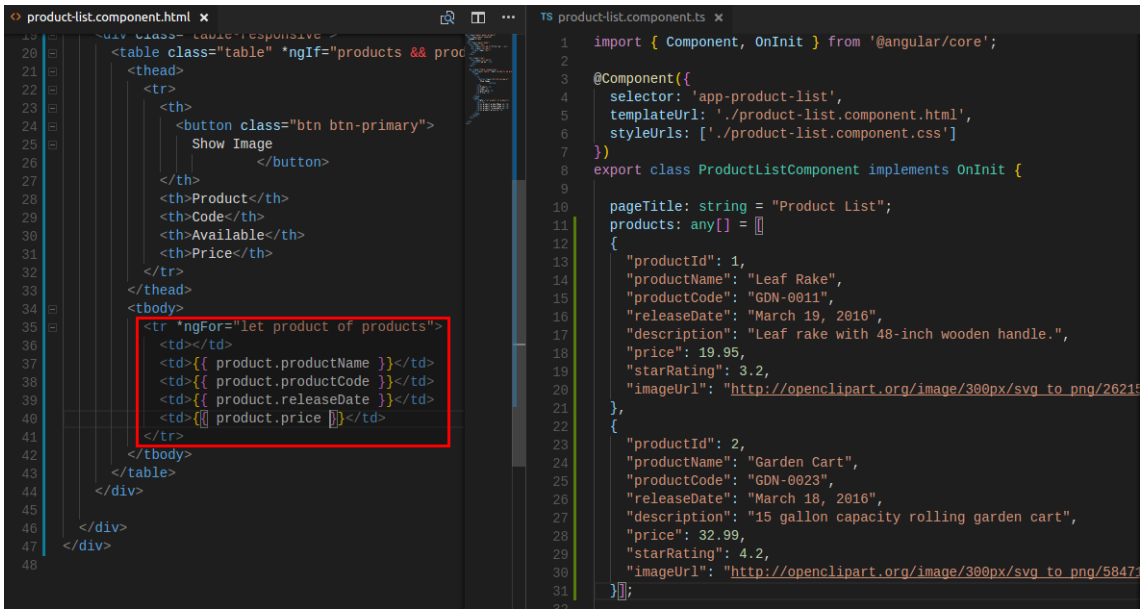
C'est bien beau d'avoir un tableau, mais un tableau avec des valeurs c'est mieux. Pour cela nous aurons besoin d'utiliser les **directives structurelles** *ngIf et *ngFor (non, les astérisques ne sont pas là par erreur).

- *ngIf : permet de supprimer ou de recréer l'élément courant d'après l'expression passée en paramètre. Si l'expression assignée au ngIf est évaluée à *false*, alors l'élément sera supprimé du DOM, puis sera recréé si l'expression est évaluée à *true*.
- *ngFor : permet d'itérer sur un tableau et de créer un template pour chaque élément.

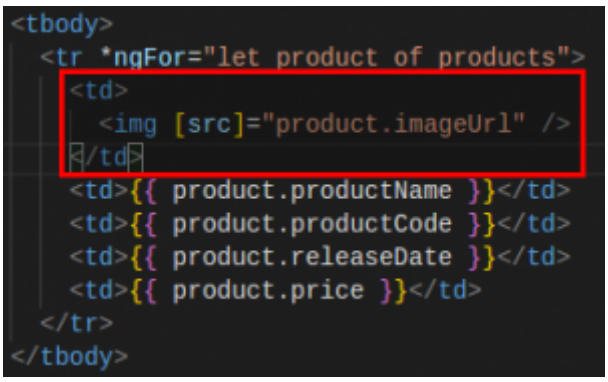
Nous ne souhaitons pas afficher de tableau tant que la liste ne contient aucun élément ; pour cela nous utiliserons le *ngIf comme suit :



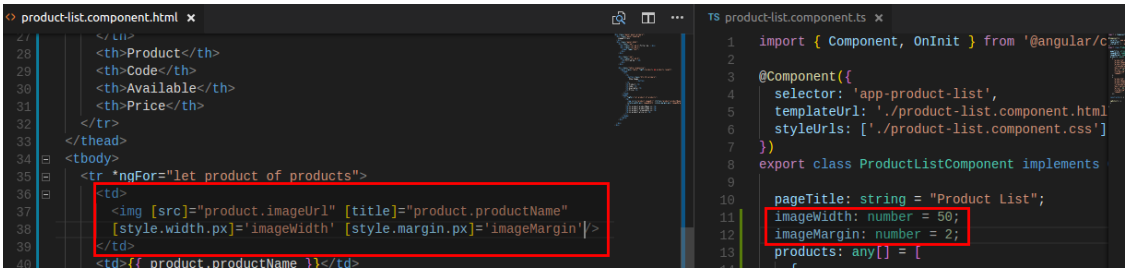
Maintenant nous allons peupler notre liste de produits et en afficher les valeurs avec *ngFor comme suit (vous trouverez les valeurs dans src/api/products/products.json) :



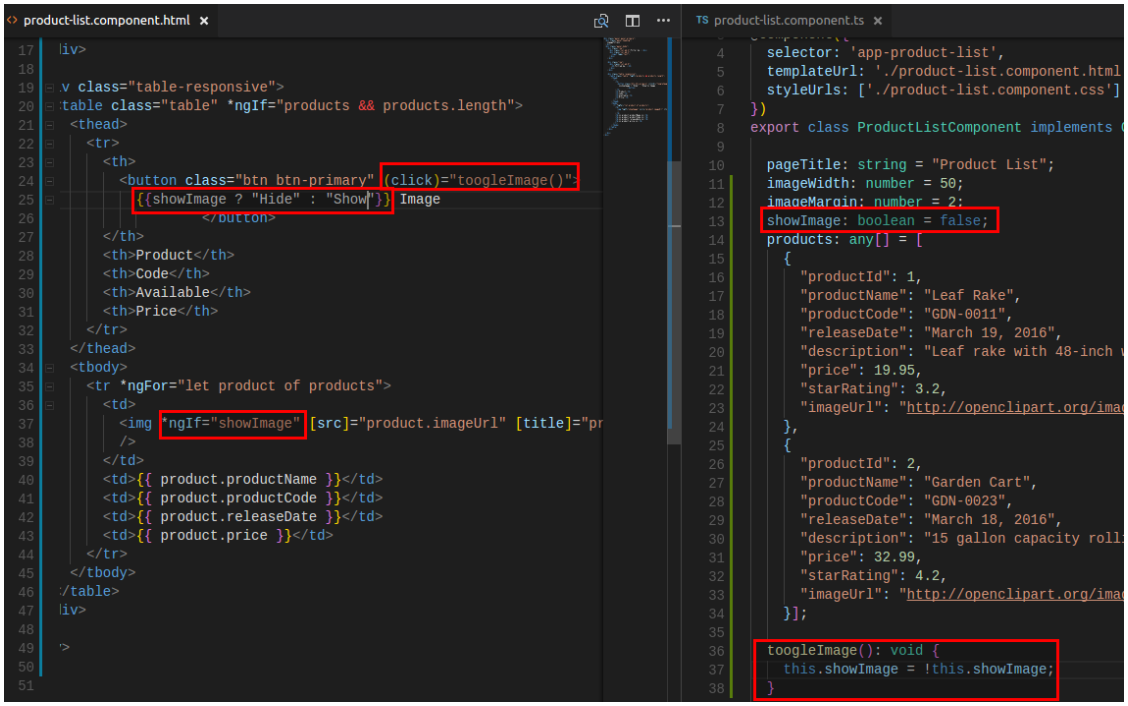
Nous utiliserons le *property binding* pour afficher les images des produits comme ceci :



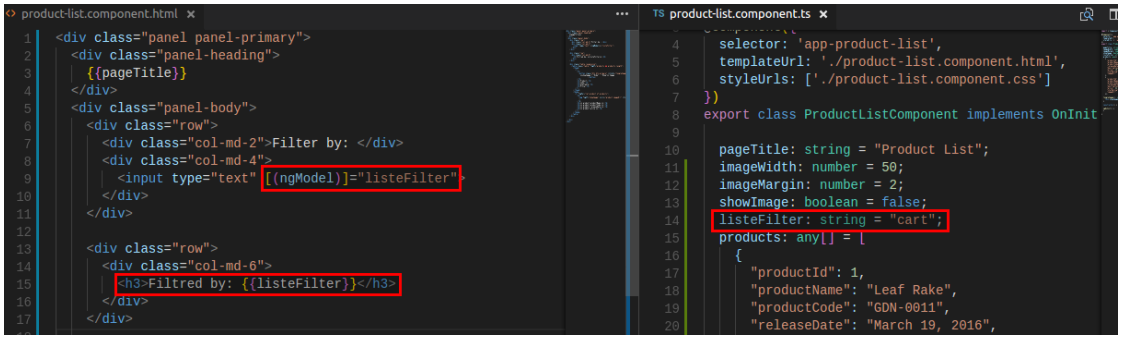
En revanche nos images sont trop grandes… Nous allons donc créer deux variables dans notre classe pour gérer la taille et le margin :



Afin de mettre en pratique l'*event binding*, nous allons créer une méthode qui affiche ou cache les images des produits :



Au tour du *two-way data binding*, nous allons filtrer la liste en tapant un texte dans l'input tout en l'affichant en bas :



Cela ne fonctionne pas ? C'est normal, car il nous faut importer le module contenant le ngModel dans notre app.module.ts :

```ts
import { FormsModule } from '@angular/forms';
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';

import { AppComponent } from './app.component';
import { ProductListComponent } from './products/product-list/product-list.component';

@NgModule({
  declarations: [
    AppComponent,
    ProductListComponent
  ],
  imports: [
    BrowserModule, FormsModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

Pour le moment, on ne filtre pas notre liste, nous verrons cela dans un deuxième temps. Nous allons d'abord parler des **Interfaces**.
Nous avons déclaré nos *products* comme étant de type *any*. Nous pouvons créer notre propre interface qui contiendra les différentes propriétés et méthodes :

```ts
import { IProduct } from '../product';
import { Component, OnInit } from '@angular/core';

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

  pageTitle: string = "Product List";
  imageWidth: number = 50;
  imageMargin: number = 2;
  showImage: boolean = false;
  listeFilter: string = "cart";
  products: IProduct[] = [
    {
      "productId": 1,
      "productName": "Leaf Rake",
      "productCode": "GDN-0011",
      "releaseDate": "March 19, 2016",
      "description": "Leaf rake with 48-inch wooden handle.
      "price": 19.95,
      "starRating": 3.2,
      "imageUrl": "http://openclipart.org/image/300px/svg_t
    },
```

```ts
export interface IProduct {
    productId: number;
    productName: string;
    productCode: string;
    releaseDate: string;
    description: string;
    price: number;
    starRating: number;
    imageUrl: string;
}
```

Rien de plus simple, il suffit d'utiliser le mot-clé **interface** précédé d'un **export** et de définir les propriétés de notre objet ainsi que leur type.

Maintenant nous pouvons filtrer notre liste, nous allons tout d'abord créer un nouveau tableau de type IProduct que l'on nommera filtredProducts, qui recevra les éléments filtrés et un getter/setter pour la variable listeFilter précédemment créée, contenant la chaîne de caractères sur laquelle nous souhaitons filtrer notre tableau :

```
filteredProducts: IProduct[];
private _listeFilter: string;

get listeFilter(): string {
  return this._listeFilter;
}
set listeFilter(value: string) {
  this._listeFilter = value;
  this.filteredProducts = this.listeFilter ? this.performFilter(this.listeFilter) : this.products;
}
```

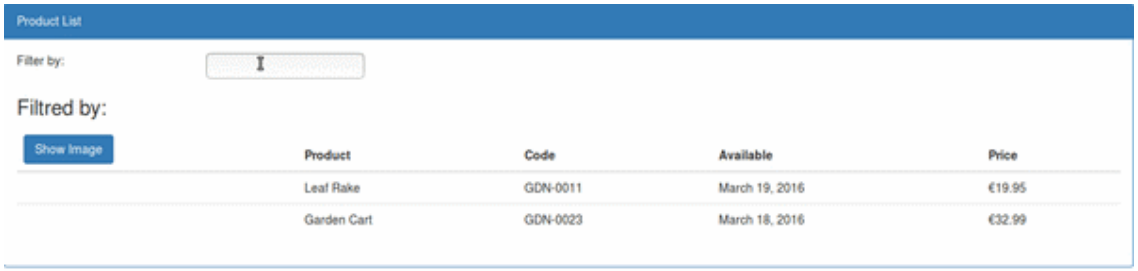
Le code est simple : si la valeur de listeFilter est valorisée, on effectue le filtre grâce à la méthode performFilter() sinon on renvoie la liste initiale. Voici notre méthode de filtre :

```
performFilter(filterBy: string): IProduct[] {
  filterBy = filterBy.toLocaleLowerCase();
  return this.products.filter((product: IProduct) =>
    product.productName.toLocaleLowerCase().indexOf(filterBy) !== -1);
}
```

Nous utilisons la méthode filter afin de créer un nouveau tableau contenant les valeurs qui passeront le test défini dans la fonction.
Maintenant il nous faut définir la valeur par défaut de notre variable filteredProducts. Nous ferons cela dans notre constructeur qui est initialisé lors de la première initialisation du *component* :

```
constructor() {
  this.filteredProducts = this.products;
}
```

Tout ce qui nous reste à faire est de modifier notre template afin de boucler sur filtredProducts au lieu de products et tadaa !



VII - Conclusion

Dans cette première partie, nous avons vu comment créer un projet Angular 4 from scratch en utilisant Angular CLI. Nous nous sommes familiarisés avec les bases d'une application Angular (structure du projet, module, component, template…). Nous avons également appris comment communiquer entre le DOM et notre *component* en utilisant les quatre techniques de *data binding* pour avoir un début de projet fonctionnel. Lors de la seconde partie, nous pourrons nous focaliser sur :

- la création de *Services* ;
- la création de *Pipes* ;
- l'utilisation de l'*HttpClient* pour les appels d'API ;
- l'utilisation des *routers* afin de naviguer dans notre application.

VIII - Remerciements

Nous remercions Soat qui nous autorise à publier ce tutoriel.

Nous tenons également à remercier **Winjerome** pour la mise au gabarit et **Claude Leloup** pour la correction orthographique.