



Version d'Angular : 5.0.0.

Version du support : 0.5.

Première rédaction : 13 juin 2018.

Dernière mise à jour : 10 décembre 2018.

IDE utilisée : Visual Studio Code.

# Table des matières

1 - Angular.....	4
1.1 - Présentation d'Angular.....	5
1.1.1 - Historique.....	5
1.1.2 - Objectif de ce document.....	6
1.1.3 - Présentation d'Angular.....	6
1.1.4 - Installation à partir de Node.js.....	7
1.2 - Création et exécution du premier projet.....	8
1.2.1 - Création du projet.....	8
1.2.2 - Lancement de l'application.....	8
1.3 - Modification du premier projet.....	9
1.3.1 - Arborescence du projet. projet1/src/.....	9
1.3.1.1 - Le fichier index.html.....	10
1.3.1.2 - Le fichier main.ts.....	11
1.3.2 - Modification du titre.....	13
1.3.3 - Modification du CSS.....	14
1.4 - Détails.....	15
1.4.1 - Diagramme de classes de l'application.....	15
1.4.2 - app.module.ts.....	16
1.4.3 - app.component.ts.....	17
1.4.4 - app.component.html.....	18
1.4.5 - app.component.css.....	19
1.4.6 - Quelques remarques sur les templates.....	20
1.4.7 - Quelques remarques sur la valorisation des attributs.....	21
1.5 - Modèle de code.....	22
1.5.1 - Modèle de classe app.module.ts.....	22
1.5.2 - Modèle de classe app.component.ts.....	23
1.5.3 - Modèle de template.....	24
2 - Première application.....	25
2.1.1 - Rendu.....	26
2.1.2 - index.html.....	27
2.1.3 - app.module.ts.....	28
2.1.4 - app.component.ts.....	29
2.1.5 - app.component.html.....	30
2.1.6 - Exercice : remplir une liste à partir d'un tableau statique.....	31
3 - Interactions utilisateur.....	32
3.1 - Qu'est-ce que l'event binding ?.....	33
3.2 - S'abonner à un événement.....	34
3.2.1 - Principe.....	34
3.2.2 - Premier exemple : afficher un texte lors d'un clic.....	35
3.2.3 - Deuxième exemple : un incrément.....	38
3.3 - Récupérer une entrée utilisateur.....	40
3.3.1 - Comment manipuler l'objet \$event ?.....	40
3.3.2 - Récupérer une valeur saisie.....	41
3.3.3 - Exercice : additionner 2 valeurs saisies.....	43
3.3.4 - Exercice : récupérer la sélection dans une liste.....	44
3.3.5 - Utiliser un typage fort pour \$event.....	45
3.3.6 - Une alternative grâce à un template reference variable.....	46
3.3.7 - Utiliser un template reference variable avec un événement.....	47
3.3.8 - Filtrer les entrées utilisateur.....	48
4 - Les formulaires.....	50

4.1 - Présentation.....	51
4.2 - Formulaire basé sur un template.....	52
4.3 - Exercice : formulaire avec liste déroulante.....	55
5 - Services.....	56
5.1 - Présentation.....	57
5.1.1 - Notion d'injection de dépendances.....	57
5.1.2 - Notion de service.....	57
5.1.3 - Syntaxes.....	58
5.1.3.1 - Création du service.....	58
5.1.3.2 - Référencement du service dans le composant.....	58
5.2 - Exemple : number-service.....	59
5.3 - Exercice : calculs.....	61
6 - Requêtes HTTP.....	62
6.1 - Présentation.....	63
6.2 - 1e exemple : Requête GET, afficher une ville.....	65
6.3 - 2e exemple : Requête GET, afficher les villes.....	71
6.4 - 3e exemple : Requête GET paramétrée, afficher une ville.....	75
6.5 - 4e exemple : Requête POST, ajouter une ville.....	77
6.6 - Version pour des données MySQL.....	79
6.6.1 - Les scripts PHP.....	79
6.6.2 - Les modifications dans la classe de type Service.....	85
6.7 - Exercice : la même chose mais avec un « formulaire ».....	86
7 - Routes.....	87
7.1 - Préparation.....	88
7.2 - Exemple.....	91
7.3 - Exercice.....	92
8 - Déploiement.....	93
8.1 - Version simplifié.....	94
9 - Annexes.....	95
9.1 - Versions.....	96
9.2 - Démarche de base.....	97
9.3 - Scripts Projet2.....	99
9.4 - Projet TodoList.....	100
9.5 - Front-end, back-end, etc.....	102
9.6 - Index des tableaux.....	103
9.7 - Index des illustrations.....	104

# 1 - ANGULAR

Version 5 d'Angular

Version 0.1 du support de cours.

Les projets correspondant à ce support sont dans :

/pascal/\_\_\_supports/Angular/AngularV5

Lancement du serveur dans le dossier racine de l'application :

```
| cd /pascal/___supports/Angular/AngularV5/projetXXX
```

```
| $ ng serve -o
```

L'IDE utilisée : Visual Studio Code.

---

## 1.1 - PRÉSENTATION D'ANGULAR

<https://angular.io/>

### 1.1.1 - Historique

Concept / Date	Description
Internet	Internet est le réseau informatique mondial, c'est l'infrastructure globale, basée sur le protocole IP, et sur laquelle s'appuient de nombreux autres services. Dont le web.
Autres services du NET	Messagerie, FTP,
WEB	Le World Wide Web, c'est le système qui nous permet de naviguer de pages en pages en cliquant sur des liens grâce à un navigateur. Donc cela comprend au minimum un serveur Web (Apache par exemple), un langage hypertexte (HTML), un navigateur.
1991	Annonce publique de HTML. Communication d'informations. HTML a été inventé pour permettre d'écrire des documents hypertextuels liant les différentes ressources d'Internet avec des hyperliens.
1996	CSS (version initiale), pris en compte par les navigateurs en 2000 ! Mise en page des pages WEB.
1995	JavaScript (Standardisé en 1997). Interaction avec les pages HTML.
1998	XML (recommandation du W3C). SGML simplifié mais strict.
2002	AJAX (Asynchronous JavaScript and XML). Il permet des interactions entre l'utilisateur et des backends HTTP il est possible d'échanger des informations et de générer du contenu à partir de ces interactions.
2010	AngularJS : simplifier JavaScript.
'Septembre 2016	Angular (Angular 2.0.0)
'Mars 2017	Angular 4
'Novembre 2017	Angular 5
'Mai 2018	Angular 6

### 1.1.2 - Objectif de ce document

Angular pour le Web.

### 1.1.3 - Présentation d'Angular

What is Angular?

Angular is a platform that makes it easy to build applications with the web. Angular combines declarative templates, dependency injection, end to end tooling, and integrated best practices to solve development challenges. Angular empowers developers to build applications that live on the web, mobile, or the desktop

<https://angular.io/docs>

Angular est un framework JavaScript libre et open-source développé par Google, au même titre que MooTools, Prototype ou Dojo. Il a pour but de simplifier la syntaxe JavaScript, et de combler les faiblesses de JavaScript en lui ajoutant de nouvelles fonctionnalités et ainsi de faciliter la réalisation d'applications web monopage (SPA : Single Page Application).

Angular utilise la bibliothèque open source jQuery. Si jQuery n'est pas présent dans le chemin du script, Angular reprend sa propre implémentation de jQuery lite. Si jQuery est présent dans le chemin, Angular l'utilise pour manipuler le DOM2.

Angular est fondé sur la croyance que la **programmation déclarative** doit être utilisée pour construire les interfaces utilisateur et les composants logiciels de liaison, tandis que la **programmation impérative** excelle pour exprimer la logique métier. Le framework adapte et étend HTML pour servir du contenu dynamique de façon améliorée grâce à un **data-binding bidirectionnel** permettant la synchronisation automatique des modèles et des vues. En conséquence, Angular minore l'importance des manipulations DOM et améliore la testabilité du code.

L'objectif de conception du framework est de découpler les manipulations du DOM de la logique métier. Cela améliore la testabilité du code.

Il faut considérer que tester une application est aussi important que l'écriture de l'application elle-même. La difficulté de la phase de test est considérablement impactée par la façon dont le code est structuré.

Découpler les couches cliente et serveur (\*) d'une application permet au développement logiciel de progresser en parallèle ainsi que de favoriser la réutilisabilité de chacune des couches.

Angular suit le patron de conception logicielle Modèle-Vue-Contrôleur (MVC : Model-View-Controller) et encourage le couplage faible entre la présentation, les données, et les composants métiers. En utilisant l'injection de dépendances, Angular apporte aux applications web côté client les services traditionnellement apportés côté serveur, comme les contrôleurs de vues. En conséquence, une bonne partie du fardeau supporté par le back-end est supprimée, ce qui conduit à des applications web beaucoup plus légères.

(\*) couches front-end et back-end plutôt.

### **1.1.4 - Installation à partir de Node.js**

Il faut Node.js (version 8.x minimum) d'installer (serveur et gestionnaire de paquets [\*]).

Vérifiez avec :

```
| $ node -v
```

npm doit être de version 5.x minimum.

Vérifiez avec :

```
| $ npm -v
```

Installation d'Angular.

```
| $ npm install -g @angular/cli
```

[\*] <https://nodejs.org/en/download/>

node-v8.11.3-x64.msi pour Windows 64 bits.

---

## 1.2 - CRÉATION ET EXÉCUTION DU PREMIER PROJET

<https://angular.io/guide/quickstart>

### 1.2.1 - Création du projet

Emplacement du projet1 : /pascal/\_\_\_supports/Angular/AngularV5/projet1

Ouvrez une console (Touche Windows + R).

Déplacez-vous dans le dossier /pascal/\_\_\_supports/Angular/AngularV5/

```
| $ cd /pascal/___supports/Angular/AngularV5/
```

Tapez à la console la commande suivante :

```
| $ ng new projet1
```

Le dossier projet1 a été créé ainsi que tous les sous-dossiers et donc la première application Angular.

### 1.2.2 - Lancement de l'application

Allez dans le dossier du projet

```
| $ cd projet1
```

Lancez l'application ie le serveur

```
| $ ng serve -o
```

Note : -o = --open

L'application est disponible et visible dans un navigateur à localhost sur le port 4200

L'option -o fait que dès que vous modifierez du code source la page web sera mise à jour.

<http://localhost:4200/>



---

## 1.3 - MODIFICATION DU PREMIER PROJET

**IDE utilisée** : Visual Studio Code.

**Avec Visual Studio Code ouvrez le dossier /src/app/ du projet.**

### ***1.3.1 - Arborescence du projet. projet1/src/***

Contenu du dossier /projet1/src/

Ce dossier contient 2 fichiers importants : index.html et main.ts. C'est de là que ça démarre. Le script app.module.ts – situé dans le dossier app/ - est ensuite exécuté. Le composant référencé dans l'attribut bootstrap est exécuté – par défaut c'est la classe AppComponent – dans le fichier app.component.ts - qui a comme template le fichier app.component.html.

### 1.3.1.1 - Le fichier index.html

Le point d'entrée de l'application (après main.ts).

```
<!doctype html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <title>Projet1</title>
  <base href="/">

  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="icon" type="image/x-icon" href="favicon.ico">
</head>

<body>
  <app-root></app-root>
</body>
</html>
```

Il ne contient que cela.

Les éléments vont être ajoutés dynamiquement dans cet élément.

### 1.3.1.2 - Le fichier main.ts

Ce fichier contient les directives de l'application : mode production ou pas, la gestion des erreurs, le module principal.

```
import { enableProdMode } from '@angular/core';
import { platformBrowserDynamic } from '@angular/platform-browser-dynamic';
import { AppModule } from './app/app.module';
import { environment } from './environments/environment';

if (environment.production) {
  enableProdMode();
}
platformBrowserDynamic().bootstrapModule(AppModule)
  .catch(err => console.log(err));
```

L'instruction « import » permet d'importer des modules Node.js dans l'application. Certains modules sont ceux d'Angular : @angular/core, @angular/platform-browser-dynamic. D'autres sont propres à notre application : import { AppModule } from './app/app.module'.

#### Syntaxe de l'import :

```
import { NomDuModule } from './chemin/module';
```

Nom du module : nom du module tel qu'il est exporté.  
Chemin : le chemin des modules.

L'environnement par défaut est à production : false.

Dans C:\pascal\\_\_supports\Angular\AngularV5\projet1\src\environments\environment.ts

```
export const environment = {
  production: false
};
```

Contenu du dossier projet1/src/app/

Fichier	Description
app.component.css	Le CSS du template
app.component.html	Le template HTML : une div, un H2, et une ul,li
app.component.spec.ts	Du TypeScript : des fonctions anonymes fléchées et des appels asynchrones
app.component.ts	Du TypeScript : la logique applicative
app.module.ts	Du TypeScript : app.module est le module racine (root module) qui permet l'organisation et la compilation du reste de l'application. Nous y déclarerons les composants que nous allons créer pour qu'ils puissent être utilisés dans toute l'application.

### 1.3.2 - Modification du titre

Déplacez-vous dans le dossier /projet1/src/app.

Ouvrez le fichier `app.component.ts` avec votre éditeur préféré (**Visual Studio Code** ou WebStorm ou NotePad++ ou ...).

Modifiez le titre.

```
/*
app.component.ts
*/
import { Component } from '@angular/core';

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

Testez !!!

<http://localhost:4200/>

Le script définit un composant Angular (Angular Component). L'instruction « export » - c'est du TypeScript, crée le composant.

Le décorateur @Component – nécessitant d'import Component – précise qu'il s'agit d'un composant Angular.

L'attribut « selector » est l'« élément » HTML qui sera impacté par Angular.

L'attribut « templateUrl » définit la page HTML impactée (plutôt une portion de code HTML).

L'attribut « styleUrls » définit le fichier CSS utilisé pour styler la page.

L'instruction « export » permet de préciser que le composant devient lui-même un module Node.js.

Les 2 principales notions d'Angular :

Le composant : un composant Angular est une classe TypeScript qui contient des propriétés (et éventuellement des méthodes).

Le template : un template est du code HTML qui contient des « controls » HTML (des éléments HTML).

cf <https://v5.angular.io/guide/displaying-data>

You can display data by binding controls in an HTML template to properties of an Angular component.

### **1.3.3 - Modification du CSS**

Ouvrez le fichier suivant : `app.component.css`

Il est vide.

Ajoutez ce code :

```
h1 {  
  color: #369;  
  font-family: Arial, Helvetica, sans-serif;  
  font-size: 250%;  
}
```

Testez !!!

---

## **1.4 - DÉTAILS**

### **1.4.1 - *Diagramme de classes de l'application***

### 1.4.2 - *app.module.ts*

Le fichier `app.module.ts` est le module racine (root module) qui permet l'organisation et la compilation du reste de l'application. Nous y déclarerons les « composants » (composants) que nous allons créer pour qu'ils puissent être utilisés dans toute l'application.

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

import { AppComponent } from './app.component';

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



### 1.4.3 - *app.component.ts*

Ce script déclare un composant (avec l'annotation @Component).

Le « selector », la balise à rechercher, est <app-root>.

Le code HTML est dans un fichier externe nommé app.component.html situé dans le même dossier. Le code HTML est un template ie un modèle de code de la vue.

Idem pour le code CSS.

```
// app.component.ts
import { Component } from '@angular/core';

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

#### 1.4.4 - app.component.html

```
<!--The content below is only a placeholder and can be replaced.-->
<div style="text-align:center">
  <h1>
    Welcome to {{ title }}!
  </h1>
  
</div>
<h2>Here are some links to help you start: </h2>
<ul>
  <li>
    <h2><a target="_blank" rel="noopener"
href="https://angular.io/tutorial">Tour of Heroes</a></h2>
    </li>
  <li>
    <h2><a target="_blank" rel="noopener"
href="https://github.com/angular/angular-cli/wiki">CLI Documentation</a></
h2>
    </li>
  <li>
    <h2><a target="_blank" rel="noopener"
href="https://blog.angular.io/">Angular blog</a></h2>
    </li>
</ul>
```

### **1.4.5 - *app.component.css***

Le code CSS du template HTML.

Du CSS classique !

```
| h1 {  
  color: #369;  
  font-family: Arial, Helvetica, sans-serif;  
  font-size: 100%;  
| }
```

### 1.4.6 - Quelques remarques sur les templates

#### **Bof ! Bof ! Bof !**

Template inline or template file? (cf <https://v5.angular.io/guide/displaying-data>).

Dans les codes qui précèdent les templates sont des portions de code HTML externes – situés dans des fichiers .html - aux composants TypeScript codés dans des scripts .ts.

Il est possible de « valoriser » les templates dans le code ts directement. L'attribut utilisé ne sera plus « templateUrl » mais « template ». Le template – du code html – sera interpolé entre des ` (back quotes inversées. La « variable » à valoriser – par interpolation - est toujours entre des accolades doubles.

Le code précédent dans app.component.ts (template file) :

```
@Component ({  
  selector: 'app-root',  
  templateUrl: './app.component.html',  
  styleUrls: ['./app.component.css']  
})
```

Le même code dans un template inline :

```
@Component ({  
  selector: 'app-root',  
  template: `Welcome to {{ title }}!`,  
  styleUrls: ['./app.component.css']  
})
```

### 1.4.7 - Quelques remarques sur la valorisation des attributs

Constructor or variable initialization? (cf <https://v5.angular.io/guide/displaying-data>).

Dans les codes qui précèdent les variables-attributs sont initialisées directement.  
Il est possible de passer par un « constructor » pour les initialiser.

Initialisation d'attribut directement :

```
export class AppComponent {  
  title = 'Mon titre';  
}
```

Initialisation d'attribut via un constructeur :

```
export class AppComponent {  
  title: string;  
  
  constructor() {  
    this.title = 'Le titre de la page !';  
  }  
}
```

---

## 1.5 - MODÈLE DE CODE

### 1.5.1 - Modèle de classe *app.module.ts*

La classe *app.module.ts* – avec le décorateur `@NgModule` - est la classe **parent** de l'application. Ce fichier *app.module.ts* est le module racine (root module) qui permet l'organisation et la compilation du reste de l'application. Nous y déclarerons les « composants » (composants) que nous allons créer pour qu'ils puissent être utilisés dans toute l'application.

```
// app.module.ts
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { FormsModule } from '@angular/forms';
import { CommonModule } from '@angular/common';
import { RouterModule, Routes } from '@angular/router';
import { HttpClientModule } from '@angular/http';

import { MyComponent } from './app.component';
import { Todo } from './todo';

@NgModule({
  declarations: [
    MyComponent,
    Todo
  ],
  imports: [
    BrowserModule,
    FormsModule,
    HttpClientModule,
    CommonModule
  ],
  exports: [RouterModule],
  providers: [],
  bootstrap: [MyComponent]
})

export class AppModule { }
```

#### Notes :

L'import du noyau d'Angular.

Et d'autres import NG.

Et 2 imports perso.

Le décorateur – l'annotation - `@NgModule` pour la gestion des modules.

L'attribut « declaration » pour les modules perso,

l'attribut « import » pour les modules Node.js,

l'attribut « export » pour les routes,

l'attribut « providers » pour les services,

l'attribut « bootstrap » pour la classe de démarrage.

Le code de *app.module.ts* est une classe qui lorsqu'elle est exportée devient un module Node.js.

### 1.5.2 - Modèle de classe *app.component.ts*

Une classe de ce type – avec le décorateur @Component – est une classe type de composant Angular, par exemple d'une liste ou d'une case à cocher.

```
// app.component.ts
import { Component } from '@angular/core';

@Component({
  selector: 'app-root',
  templateUrl: 'xxx.html',
  styleUrls: ['./xxx.css']
})

export class NomDeClasse {
  // attributs
  variable: type = 0;

  // méthodes
  methode() {
    this.variable++;
  }
}
```

#### Notes :

L'import du noyau d'Angular.

Le décorateur – l'annotation - @Component.

L'attribut « selector », un élément html (une balise !) dans la page index.html,

l'attribut « templateUrl » pour la vue correspondante au composant, un fragment de code HTML,

l'attribut « styleUrls » pour la référence à une feuille de style (xxx.css).

La classe.

Ses attributs.

Ses méthodes.

### 1.5.3 - Modèle de template

Une <div> HTML avec un formulaire.

{{ message }} : une variable du template qui « recevra » une valeur d'un attribut du composant.

```
<!-- form0.html -->
<div>
  <h3>Form0</h3>
  <!-- <form> -->
  <form>
    <p>
      ID :
      <input name="id" />
      Prénom :
      <input name="firstName" />
    </p>
    <p>
      <button>Valider</button>
      <label id="lblMessage"> {{ message }} </label>
    </p>
  </form>
</div>
```



## **2 - PREMIÈRE APPLICATION**

### 2.1.1 - Rendu

Before app-root

Welcome to Accueil!



Superman ▼ Valider

**Here are some links to help you start:**

- [Tutoriel Angular](#)
- [WIKI](#)
- [Angular blog](#)

After app-root

### 2.1.2 - index.html

Le point d'entrée de l'application. Du classique !

L'élément `<app-root>` sera référencé dans le fichier `app.component.ts` via l'attribut « selector ».

```
<!doctype html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <title>Projet1</title>
  <base href="/">

  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="icon" type="image/x-icon" href="favicon.ico">
</head>

<body>
  <p>Before app-root</p>
  <app-root></app-root>
  <p>After app-root</p>
</body>
</html>
```

### 2.1.3 - *app.module.ts*

**Fonctionnalité de ce script** : ce fichier `app.module.ts` est le module racine (root module) qui permet l'organisation et la compilation du reste de l'application. Nous y déclarerons les « composants » (composants) que nous allons créer pour qu'ils puissent être utilisés dans toute l'application.

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

import { AppComponent } from './app.component';

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

### 2.1.4 - app.component.ts

Le modèle ...

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

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})

export class AppComponent {
  title = 'Accueil';

  image2 = '../assets/2.jpg';

  image3 = '../assets/3.jpg';

  image4 = '../assets/4.jpg';

  itemsList = [
    {valeur:'Superman'},
    {valeur:'Superwoman'},
    {valeur:'Batman'},
    {valeur:'Batgirl'},
    {valeur:'Robin'},
    {valeur:'Robine'},
    {valeur:'Flash'}
  ];

  urls = [
    "https://angular.io/tutorial",
    "https://en.wikipedia.org/wiki/Angular_(application_platform)"
  ];
}
```

### 2.1.5 - app.component.html

La vue ...

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

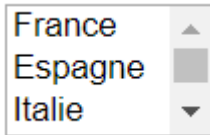
<p>
<img src='../assets/0.jpg' width="100" />
<img src='http://localhost:4200/assets/1.jpg' width="100" />
<img [src]='image2' width="100" />
<img src={{image3}} width="100" />
<img src={{image4}} width="100" />
</p>

<form>
  <select name="liste">
    <option *ngFor="let item of itemsList">
      {{item.valeur}}
    </option>
  </select>
  <input type="submit" />
</form>

<h2>Here are some links to help you start: </h2>
<ul>
  <li>
    <a target="_blank" rel="noopener" href="{{urls[0]}}">Tutoriel
    Angular</a>
  </li>
  <li>
    <a target="_blank" rel="noopener" [href]='urls[1] '>WIKI</a>
  </li>
  <li>
    <a target="_blank" rel="noopener"
    href="https://blog.angular.io/">Angular blog</a>
  </li>
</ul>
```

### 2.1.6 - Exercice : remplir une liste à partir d'un tableau statique

Trop facile !!!



Rajoutez un tableau de strings dans le composant, une liste de pays.

Rajoutez une liste dans le template html.

Affichez les items du tableau de strings dans la liste.

## **3 - INTERACTIONS UTILISATEUR**



---

### 3.1 - QU'EST-CE QUE L'EVENT BINDING ?

**C:\pascal\\_\_supports\Angular\AngularV5\projet2**

Lorsque l'utilisateur interagit avec élément HTML (clic sur un lien, un bouton, un item de liste ou saisit un texte) l'application peut réagir à ces événements.

L'event binding (liaison d'événement) permet à une application Angular d'exécuter du code lorsqu'un événement est déclenché.

La vue va notifier le composant d'une interaction.

Pour détecter les interactions de l'utilisateur le code doit être à l'écoute de certains événements.

C'est donc via le concept d'event binding que l'application s'abonne aux différentes interactions.

---

## 3.2 - S'ABONNER À UN ÉVÉNEMENT

### 3.2.1 - Principe

La syntaxe d'un event binding est la suivante : il faut décorer de parenthèses le nom de l'événement DOM à écouter et fournir la méthode à exécuter sur cet événement.

```
<button (click)="méthode()">Cliquez-ici</button>
```

Ici, l'élément HTML sait que lorsque l'événement (click) est déclenché, il faut exécuter le code suivant : méthode(). Cette méthode est déclarée dans le composant correspondant.

### 3.2.2 - Premier exemple : afficher un texte lors d'un clic

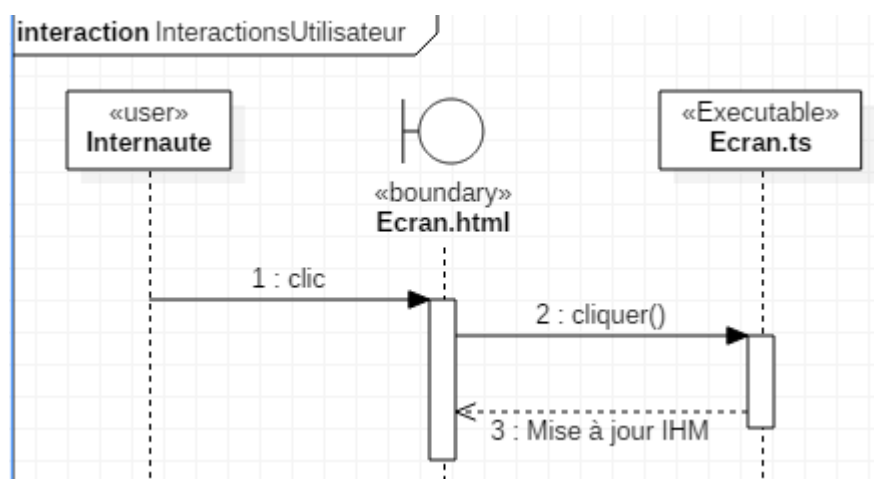
Afficher un texte lors du clic sur un bouton.

Avant	Après
	

#### Démarche :

Création d'un composant - une classe TypeScript - : Cliquer.ts,  
Création d'un template : cliquer.html,  
On garde le même sélecteur dans le fichier index.html,  
Référencement dans le fichier app.module.ts.

#### Diagramme de séquence :



## Le composant :

```
// Cliquer.ts
import { Component } from '@angular/core';

@Component({
  // Le sélecteur dans index.html
  selector: 'app-root',
  // L'élément HTML dans le template HTML (cliquer.html)
  templateUrl: 'cliquer.html'
})

export class Cliquer {
  // L'attribut de la classe qui correspond à la variable {{ message }}
  dans le template
  message:string = "";

  // La méthode qui sera sollicitée sur le clic du bouton
  cliquer() {
    this.message = "clic réalisé !";
  }
}
```

## Le template html :

```
<!-- cliquer.html -->
<p>
  <button (click)="cliquer()">Cliquer</button> {{message}}
</p>
```

### Le composant « application » :

```
// app.module.ts
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';

import { Cliquer } from './Cliquer';

@NgModule({
  declarations: [
    Cliquer
  ],
  imports: [
    BrowserModule
  ],
  providers: [],
  bootstrap: [Cliquer]
})

export class AppModule { }
```

### 3.2.3 - Deuxième exemple : un incrément

Autre exemple avec un composant qui, dans son template, affiche un compteur et possède un bouton qui appelle la méthode `incrementTotal()`.

#### Démarche :

Création d'un composant - une classe TypeScript - : `Increment.ts`,

Création du template : `increment.html`,

Création d'un sélecteur dans le fichier `index.html`,

Référencement dans le fichier `app.module.ts`.

```
// Increment.ts
import { Component } from '@angular/core';

@Component({
  // Le sélecteur dans index.html
  selector: 'increment',
  // L'élément HTML dans le composant HTML (increment.html)
  templateUrl: 'increment.html'
})

export class Increment {
  total:number = 0;

  // La méthode qui sera sollicitée sur le clic du bouton
  incrementTotal() {
    this.total++;
  }
}
```

#### L'élément html :

```
<!-- increment.html -->
<p>
  <button (click)="incrementTotal()">Incrémenter</button> Total :
  {{total}}
</p>
```

Dans le fichier `index.html`, n'importe où dans le `<body>` :

```
<increment></increment>
```

Dans le fichier app.module.ts :

```
// app.module.ts
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';

import { Increment } from './increment';

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

---

## 3.3 - RÉCUPÉRER UNE ENTRÉE UTILISATEUR

<https://angular.io/guide/user-input>

### 3.3.1 - Comment manipuler l'objet \$event ?

Les exemples précédents n'utilisent que la méthode click d'un bouton ; cependant il existe, bien entendu, plusieurs types d'événements et notamment ceux renvoyant une entrée utilisateur. L'exemple le plus concret est une touche du clavier sur lequel l'utilisateur va taper.

Un objet est disponible dans la variable `$event` fournie par Angular. C'est cet objet qui contient la donnée que nous voulons récupérer. Une première solution est de passer cet objet `$event` dans une méthode et d'aller chercher l'entrée utilisateur à l'intérieur.

Dans l'exemple qui suit, nous allons concaténer les entrées une par une, en les séparant d'une virgule.

```
// SaisieClavier.ts
import { Component } from '@angular/core';

@Component({
  selector: 'app-root',
  templateUrl: 'saisieClavier.html'
})

export class SaisieClavier {
  saisies = '';

  onFrappe(e: any) {
    this.saisies += e.target.value + ',';
  }
}
```

```
<!-- saisieClavier.html -->
<input (keyup)="onFrappe($event)">
<p>{{saisies}}</p>
```

Le composant possède une méthode perso `onFrappe()` qui prend un paramètre, c'est l'événement du navigateur, dont il est possible de récupérer le `target.value`, correspondant à la valeur de l'input à chaque déclenchement d'événement. Cette méthode `onFrappe()` est alors appelée sur l'événement `keyup` de l'input présent dans le template du composant.

La structure et la forme de l'objet `$event` est définie par l'élément qui lève l'événement. Ici, l'événement `keyup` vient du DOM, donc `$event` doit être un modèle d'événement du DOM. En ce sens, `$event.target` fournit un `HTMLInputElement`, qui contient une propriété `value` qui contient les entrées utilisateur.

C'est la méthode `onKeyUp` qui extrait la valeur de l'objet `$event` et qui va manipuler la donnée.

En tapant `bonjour` au clavier, nous aurons l'affichage suivant : `b,bo,bon,bonj,bonjo,bonjou,bonjour`



### 3.3.2 - Récupérer une valeur saisie

L'exemple précédent sollicitait 2 éléments HTML, ici ce sont 3 éléments HTML qui sont impactés. L'exemple qui suit est encore différent puisqu'il s'agit de récupérer à partir d'un événement sur un élément la valeur d'un autre élément et d'affecter un autre élément.

x :

Récupérer la saisie

333

#### Démarche :

Identifier l'input avec #id,  
passer l'id (sans le #) comme argument de l'appel de la fonction ou id.value.

Si vous passez id vous récupérez un objet de type any et vous affectez à this.res l'argument.value.  
Si vous passez id.value vous récupérez un objet de type number et vous affectez à this.res l'argument.

### Le template : recupSaisie.html

```
<!-- recupSaisie.html -->
<div>
  <p>
    <label> x : </label>
    <input #x value="3" />
  </p>
  <p>
    <button (click)="recupSaisie(x.value)">Récupérer la
saisie</button>
  </p>
  <p>
    <label>{{res}}</label>
  </p>
</div>
```

### Le composant : RecupSaisie.ts

```
// RecupSaisie.ts
import { Component } from '@angular/core';

@Component({
  selector: 'app-root',
  templateUrl: 'recupSaisie.html',
  providers: []
})

export class RecupSaisie {
  res: number;

  recupSaisie(x: number) {
    this.res = x;
  }
}
```

### app.module.ts

Modifiez :

- ✓ les imports,
- ✓ les déclarations,
- ✓ bootstrap.

### 3.3.3 - Exercice : additionner 2 valeurs saisies


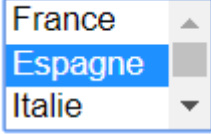
Rendu :

x :  y :

8

### 3.3.4 - Exercice : récupérer la sélection dans une liste

Afficher une liste remplie avec un tableau « statique ».  
Lors du clic sur un item, l'item est affiché en dessous.

Avant	Après
 <p>Ici sera affichée la sélection</p>	 <p>Espagne</p>

### 3.3.5 - Utiliser un typage fort pour \$event

Dans l'exemple précédent, la signature de la méthode onKeyUp est la suivante :

```
| onKeyDown(event:any) {
```

Le paramètre event n'est pas typé. Cependant, comme TypeScript nous offre le typage, il est préférable d'utiliser le type adéquat. Il est donc possible de réécrire la méthode onKeyUp de la sorte :

```
| onKeyUp(event: KeyboardEvent) {  
|     this.keys += (<HTMLInputElement>event.target).value + ',';  
| }
```

### 3.3.6 - Une alternative grâce à un template reference variable

Alternativement, Angular propose une fonctionnalité, ou plutôt une syntaxe, appelée template reference variable (variable de référence au template). Ces variables donnent accès directement à un élément du DOM. Pour déclarer une variable de référence au template, il suffit d'ajouter un identifiant/un nom précédé par un dièse.

Les template reference variable fonctionnent autant avec les composants qu'avec les éléments natifs du DOM comme les div, input, span, etc.

En restant sur l'exemple courant, nous définissons une référence à notre champ d'input avec l'identifiant myInput.

```
<input #myInput (keyup)="onKeyUp($event)">
```

Une syntaxe canonique existe aussi, pour ceux qui n'aiment pas utiliser le caractère #.

```
<input ref-myInput (keyup)="onKeyUp($event)">
```

L'identifiant myInput est devenu une référence vers l'élément <input> lui-même, ce qui signifie qu'il est maintenant possible d'accéder aux propriétés de l'élément <input>, dont la valeur courante, accessible via la propriété value, et cela partout dans le template.

Il est alors possible d'utiliser l'interpolation pour afficher la valeur de myInput dans une balise <p>.

```
@Component({
  selector: 'app-input-keystroke',
  template: `
    <input #myInput (keyup)="0">
    <p>{{myInput.value}}</p>`
})
```

Ici, le template se gère tout seul. Il n'y a plus de lien entre le template et le composant, en somme, aucun binding n'est fait entre les deux.

Il est important de comprendre chaque partie de ce code, et surtout ce bout de code qui peut sembler intrigant à premier abord.

```
(keyup)="0"
```

Sans cette partie, le code correspondant ne fonctionnerait pas. En effet, Angular met à jour le binding uniquement si quelque chose est fait en réponse aux événements asynchrones tels que les entrées clavier.

Plus précisément, sans opérations asynchrones, Angular ne sait pas qu'il faut exécuter une détection de changement et ne peut pas mettre à jour la vue.

C'est pourquoi un binding est fait au niveau de l'événement keyup vers un code qui au final ne fait rien. Le binding est fait sur le nombre 0, car c'est la déclaration la plus courte possible. Cela peut sembler surnois, mais reste nécessaire pour enclencher les mécanismes d'Angular et faire tourner son application à moindre coût en termes d'exécution.

### 3.3.7 - Utiliser un template reference variable avec un événement

Après avoir compris comment fonctionne les template reference variable et sachant utiliser les bindings d'événement, il est intéressant de voir comment les deux fonctionnalités peuvent se combiner.

Le principe est simple, il faut dans un premier temps une méthode qui prend en paramètre une chaîne de caractères, puis utiliser cette méthode en lui donnant la valeur de l'input par le biais d'une template reference variable.

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

@Component({
  selector: 'app-input-keystroke',
  template: `
    <input #myInput (keyup)="onKeyUp(myInput.value)">
    <p>{{keys}}</p>`
})

export class InputKeystrokeComponent{

  keys = "";

  onKeyUp(keyToAdd: string) {
    this.keys += keyToAdd + ',';
  }
}
```

La méthode onKeyUp reçoit donc directement la valeur du champ input. Elle n'a plus besoin d'aller chercher l'information en utilisant event.target.value.

Dans le template, il faut donc envoyer la valeur dans la méthode onKeyUp, et cela se fait grâce à myInput.value, myInput étant le template reference variable de l'input.

Avec cette approche, nous avons un code plus clair et le composant accède uniquement aux données dont il a besoin de la vue, rien de plus.

### 3.3.8 - Filtrer les entrées utilisateur

Dans un cas réel, comme un formulaire, il n'est pas forcément nécessaire de récupérer toutes les entrées unitairement. En général, on souhaite récupérer la valeur de notre champ lorsque l'utilisateur appuie sur la touche [Entrée].

Lorsqu'un binding est fait sur l'événement (keyup), la méthode associée est levée à chaque entrée du clavier. Une première solution est alors de filtrer les clés en examinant chaque `$event.keyCode`, et en ne mettant à jour notre valeur uniquement si la clé correspond à Enter.

```
@Component({
  selector: 'app-root',
  template: '<input (keyup)="onKeyUp($event)" />'
})
export class AppComponent {
  data: string = '';
  onKeyUp(event: KeyboardEvent) {
    if (event.keyCode === 13) { // event.key ==
"Enter" fonctionne aussi
      // utiliser this.data
    }
    else {
      //
      this.data += event.key;
    }
  }
}
```

Le `keyCode` correspondant à 13 est la touche [Entrée]. Ainsi, à chaque fois que le `keyCode` est différent de 13, il suffit de concaténer la propriété `key` dans une chaîne de caractères `data`. Lorsque le `keyCode` est égal à 13, il suffit d'utiliser cette propriété `data`.

Angular, quant à lui, nous permet directement de filtrer les clés pour nous. Une syntaxe particulière est à disposition pour les événements relatifs au clavier. Il va alors être possible d'être à l'écoute d'événements préfiltrés, prenons l'exemple de `keyup.enter`.

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

```
@Component({
  selector: 'app-input-filter',
  template: `
    <input #myInput (keyup.enter)="updateValuesFromEnter(myInput.value)">
    <p>{{value}}</p>
  `
})
export class InputFilterComponent implements OnInit {

  value = '';
  constructor() { }

  ngOnInit() {
  }
}
```



```

updateValuesFromEnter(value:string)
{
  this.value = value + ' from Enter';
}
}

```

Toutefois, si l'utilisateur clique ailleurs sur la page et perd le focus sur le champ, la valeur ne sera pas mise à jour, car nous écoutons uniquement l'événement keyup qui est filtré sur la touche [Entrée].

Rajoutons alors un event binding (blur), qui est l'événement déclenché lors de la perte de focus, ce qui transforme notre composant pour donner le code suivant.

```

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

@Component({
  selector: 'app-input-filter',
  template: `
    <input #myInput (keyup.enter)="updateValuesFromEnter(myInput.value)"
      (blur)="updateValuesFromBlur(myInput.value)">
    <p>{{value}}</p>
  `
})
export class InputFilterComponent implements OnInit {

  value = "";
  constructor() { }

  ngOnInit() {
  }

  updateValuesFromEnter(value:string)
  {
    this.value = value + " from Enter";
  }

  updateValuesFromBlur(value:string)
  {
    this.value = value + " from Blur";
  }
}

```

Le composant possède alors deux méthodes : updateValuesFromEnter et updateValuesFromBlur. Chaque méthode est appelée lorsque, respectivement, keyup.enter et blur sont appelés.

Bien entendu, il n'est pas obligatoire de faire cette séparation entre l'événement blur et l'événement keyup.enter.

```

<input #myInput (keyup.enter)="updateValues(myInput.value)"
  (blur)="updateValues(myInput.value)"/>

```

Il est parfaitement possible d'avoir une même méthode que l'on va utiliser pour les deux événements. Il suffit d'avoir une méthode updateValues qui prend en paramètre la valeur de l'input, keyup.enter et blur appellent donc la même méthode.

## 4 - LES FORMULAIRES

---

## 4.1 - PRÉSENTATION

C:\pascal\\_\_supports\Angular\AngularV5\projetForm

### Form1

ID :  Prénom :

Message ici !

### Form1

ID :  Prénom :

Submit, 1:Annabelle

---

## 4.2 - FORMULAIRE BASÉ SUR UN TEMPLATE

```
// Personne.ts
export class Personne {
  constructor(public id: number, public firstName: string, public name:
string, public email?: string) { }
}
```

```
// form1.ts

import { Component } from '@angular/core';
import { Personne } from './Personne';

@Component({
  selector: 'app-root',
  templateUrl: './form1.html'
})

export class Form1 {

  message: string = "Message ici !";

  personne = new Personne(1, 'Annabelle', '', '');

  submitted = false;

  /* logForm(contenuForm) {
    this.message = contenuForm.firstName;
    console.log(contenuForm.id);
    console.log(contenuForm.firstName);
    //this.message = "Submit from ...";
  } */

  onSubmit() {
    this.submitted = true;
    this.message = "Submit, " + this.personne.id + ":" +
this.personne.firstName;
  }

  // TODO: Remove this when we're done
  get diagnostic() { return JSON.stringify(this.personne); }
}
```

```
<!-- form1.html -->
<div>
  <h3>Form1</h3>
  <!-- Sur l'événement Submit ... -->
  <form (ngSubmit)="onSubmit()">
    <p>
      ID :
      <input type="text" name="id" [(ngModel)]="personne.id"
size="5"/>
      Prénom :
      <input type="text" name="firstName"
[(ngModel)]="personne.firstName" value="Pascal" />
    </p>
    <p>
      <button>Valider</button>
    </p>
    <p>
      <label id="lblMessage"> {{ message }} </label>
    </p>
  </form>
</div>
```

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

import { PersonneForm } from './personne-form.component';

@NgModule({
  imports: [
    BrowserModule,
    FormsModule
  ],
  declarations: [
    PersonneForm
  ],
  providers: [

  ],
  bootstrap: [ PersonneForm ]
})
export class AppModule { }
```

---

### 4.3 - EXERCICE : FORMULAIRE AVEC LISTE DÉROULANTE

## 5 - SERVICES



---

## 5.1 - PRÉSENTATION

C:\pascal\\_\_\_supports\Angular\AngularV5\projetServices

### 5.1.1 - Notion d'injection de dépendances

### 5.1.2 - Notion de service

L'objectif d'un service est de contenir toute la logique fonctionnelle et/ou technique de l'application. Au contraire des composants qui ne doivent contenir que la logique propre à la manipulation de leurs données.

Le code du service est stocké dans un fichier .ts.  
C'est une classe qui importe Injectable.  
Cette classe est « décorée » par l'annotation @Injectable.  
Les méthodes sont codés dans la classe.

Le service est importé dans un composant.  
Le service est référencé dans un composant dans l'attribut « providers ».  
Le service est instancié dans le constructeur de la classe « component ».  
Le service est « consommé » dans le constructeur ou dans des méthodes de la classe du « component ».

### 5.1.3 - Syntaxes

#### 5.1.3.1 - Création du service

```
// nom-service.ts
import { Injectable } from '@angular/core';

@Injectable()
export class ClasseService {
  getXXX(): type {
    // Code ...
    return XXX;
  }
}
```

#### 5.1.3.2 - Référencement du service dans le composant

```
// XXXComponent.ts
import { ClasseService } from './nom-service';
import { Component } from '@angular/core';

@Component({
  selector: 'selecteur',
  templateUrl: 'xxx.html',
  providers: [ClasseService]
})

export class XXXComponent {
  attribut: type;

  constructor(private service : ClasseService){
    this.attribut = this.service.getXXX();
    console.log(this.attribut);
  }
}
```

---

## 5.2 - EXEMPLE : NUMBER-SERVICE

Affiche :

Numbers !!!

1  
3

```
// number-service.ts
import { Injectable } from '@angular/core';
/*
L'objectif d'un service est de contenir toute la logique fonctionnelle et/
ou technique de l'application.
Au contraire des composants qui ne doivent contenir que la logique propre
à la manipulation de leurs données.
*/
@Injectable()
export class NumberService {
  getOne(): number {
    return 1;
  }
  getThree(): number {
    return 3;
  }
}
```

```
// NumberComponent.ts
import { NumberService } from './number-service';
import { Component } from '@angular/core';

@Component({
  selector: 'numbers',
  templateUrl: 'numbers.html',
  providers: [NumberService]
})

export class NumberComponent {
  one: number;
  three: number;

  constructor(private numberService : NumberService){

    this.one = this.numberService.getOne();
    console.log(this.one);

    this.three = this.numberService.getThree();
    console.log(this.three);

  }
}
```

```
<!-- numbers.html -->
<div>
<label>{{one}}</label>
<br>
<label>{{three}}</label>
</div>
```

et dans app.module.ts

```
import { NumberComponent } from './NumberComponent';
```

et aussi dans les déclarations de @NgModule

```
@NgModule({
  declarations: [
    NumberComponent
  ]
})
```

et encore dans l'attribut bootstrap

```
bootstrap: [NumberComponent]
```

et dans index.html

```
<numbers></numbers>
```

---

## 5.3 - EXERCICE : CALCULS

### 1e étape !

**Rendu, dans des labels d'une page HTML avec des valeurs statiques :**

Calculs !!! (avec 3 et 5)

Addition : 8

Soustraction : -2

Multiplication : 15

Division : 0.6

### Énoncé :

Une classe calcul-service.ts qui opère les 4 opérations arithmétiques de base.

### 2e étape, réutilisation du service avec des valeurs saisies !

## Calculs avec service de calcul

x :  y :

Addition

Soustraction

Multiplication

Division

8

## 6 - REQUÊTES HTTP

---

## 6.1 - PRÉSENTATION

<https://angular.io/guide/http>

**C:\pascal\\_\_\_supports\Angular\AngularV5\projetHTTP**

Before you can use the HttpClient, you need to import the Angular HttpClientModule. Most apps do so in the root AppModule.

```
// app.module.ts
import { HttpClientModule } from '@angular/common/http';
```

```
@NgModule({
  imports: [
    BrowserModule,
    // import HttpClientModule after BrowserModule.
    HttpClientModule,
  ],
```

Having imported HttpClientModule into the AppModule, you can inject the HttpClient into an application class as shown in the following XXXService example.

```
// XXXService.ts
import { Injectable } from '@angular/core';
import { HttpClient } from '@angular/common/http';

@Injectable()
export class XXXService {
  constructor(private http: HttpClient) { }
}
```

The XXXService fetches JSON file with a get() method on HttpClient. An other method of class XXXService.

```
getXXX() {
  return this.http.get('url');
}
```

A component, injects the XXXService and calls the getXXX service method.

```
//  
showData() {  
  this.XXXService.getXXX()  
    .subscribe((data: Classe) => this.object = {  
      attr1: data['attr1'],  
      attr2: data['attr2']  
    });  
}
```



---

## 6.2 - 1<sup>E</sup> EXEMPLE : REQUÊTE GET, AFFICHER UNE VILLE

Les données sont stockées dans un fichier nommé « ville.json ».  
Ce fichier est traité par un script PHP nommé « GetVille.php ».  
Ces fichiers sont stockés dans l'arborescence d'un site web dans un dossier nommé « AngularRessources/json/ ».

### Afficher une ville stockée dans un fichier JSON

Afficher la ville

Code postal : 75000

Nom : Paris

#### ville.json

```
{
  "cp": 75000,
  "nom": "Paris"
}
```

#### GetVille.php

```
<?php
/*
 * GetVille.php
 */

header("Access-Control-Allow-Methods: POST, GET, OPTIONS");
header("Access-Control-Allow-Headers: *");
header("Access-Control-Allow-Origin: *");

// Récupération du contenu du fichier sous forme de flux de caractères
$content = file_get_contents("ville.json");

// Affichage du contenu du fichier
echo $content;
?>
```

**Commentaire** : il est essentiel d'autoriser le cross-domain.

Premier test :

<http://localhost:80/AngularRessources/json/GetJSonVille.php>

Le diagramme de classes allégé de cette application.

**Description :** l'Interface ville.ts.

```
| // ville.ts  
| export interface Ville {  
|     cp: string;  
|     nom: string;  
| }
```

ou une classe :

```
| // ville.ts  
| export class Ville {  
|     constructor(public cp: string, public nom: string) { }  
| }
```

**Description** : tout est ici ; dans la requête HTTP et dans la méthode getVille() qui renvoie la RESPONSE.

```
// ville.service.ts
import { Injectable } from '@angular/core';
import { HttpClient } from '@angular/common/http';

@Injectable()
export class VilleService {
  constructor(private http: HttpClient) { }

  url = 'http://localhost:80/AngularRessources/json/GetVille.php';

  // Le dossier assets est dans le dossier /ProjetHTTP/src/
  //url = 'http://localhost:4200/assets/ville.json';

  getVille() {
    return this.http.get(this.url);
  }
}
```

**Description** : l'objet RESPONSE est traité ici avec la méthode subscribe() ; Les données reçues (data) sont affectées à un objet « ville ».

```
// ville.component.ts
import { Component } from '@angular/core';
import { VilleService } from '../ville.service';
import { Ville } from '../ville';

@Component({
  selector: 'app-root',
  templateUrl: '../ville.component.html',
  providers: [VilleService],
  styles: ['.error {color: red;}']
})

export class VilleComponent {
  error: any;
  headers: string[];
  ville: Ville;

  constructor(private villeService: VilleService) { }

  afficherVille() {
    this.villeService.getVille()
      .subscribe(data => this.ville = {
        cp: data['cp'],
        nom: data['nom']
      });
  }
}
```

**Description** : lors du clic sur le bouton on sollicite la méthode afficherVille() du composant ville.component.ts.

```
<!-- ville.component.html -->
<h3>Afficher une ville stockée dans un fichier JSON</h3>
<div>
  <button (click)="afficherVille()">Afficher la ville</button>

  <span *ngIf="ville">
    <p>Code postal : {{ville.cp}}</p>
    <p>Nom : {{ville.nom}}</p>
  </span>
</div>
```

**Description** : le module de l'application.

```
// app.module.ts
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { HttpClientModule } from '@angular/common/http';

import { AppComponent } from './app.component';
import { VilleComponent } from './ville/ville.component';

@NgModule({
  imports: [
    BrowserModule,
    // import HttpClientModule after BrowserModule.
    HttpClientModule
  ],
  declarations: [
    AppComponent,
    VilleComponent
  ],
  bootstrap: [ VilleComponent ]
})

export class AppModule {}
```

---

## 6.3 - 2E EXEMPLE : REQUÊTE GET, AFFICHER LES VILLES

Les données sont stockées dans un fichier nommé « villes.json ».  
Ce fichier est traité par un script PHP nommé « GetVilles.php ».  
Ces fichiers sont stockés dans l'arborescence d'un site web dans un dossier nommé « AngularRessources/json/ ».

### villes.json

```
[{"cp": "75000", "nom": "Paris"}, {"cp": "69000", "nom": "Lyon"}, {"cp": "13000", "nom": "Marseille"}, {"cp": "44000", "nom": "Nantes"}]
```

### GetVilles.php

```
<?php
/*
 * GetVilles.php
 */
header("Access-Control-Allow-Methods: POST, GET, OPTIONS");
header("Access-Control-Allow-Headers: *");
header("Access-Control-Allow-Origin: *");

// Récupération du contenu du fichier sous forme de flux de caractères
$contents = file_get_contents("villes.json");

// Affichage du contenu du fichier
echo $contents;
?>
```

Premier test :

<http://localhost:80/AngularRessources/json/GetJSonVilles.php>

## La vue : villeForm.component.html.

### Commentaires :

les inputs sont identifiés avec #nomID.

Sur le clic une ou 2 méthodes sont liées.

ngIf : si l'attribut « villes » du composant est présent alors on remplit la liste.

ngFor : boucle de parcours de l'attribut « villes » (un tableau).

```
<!-- villeForm.component.html -->
<h3>CRUD villes via un Formulaire. Stockage {{titre}}</h3>
<div>
  Code postal :
  <input type="text" #cp size="5" />
  Nom de la ville :
  <input type="text" #nom size="50" />
  <p>
    <button (click)="clear()">Effacer</button>
    <button (click)="clear(); afficherVilles()">Afficher les
villes</button>
    <button (click)="clear();
afficherVillesParam(cp.value)">Afficher une ville param</button>
    <button (click)="clear(); ajouterVille(cp.value,
nom.value)">Ajouter une ville</button>
    <button (click)="clear(); supprimerVille(cp.value)">Supprimer
une ville</button>
    <button (click)="clear(); modifierVille(cp.value,
nom.value)">Modifier une ville</button>
  </p>
</div>

<span *ngIf="villeParam">
  <p>Code postal : {{villeParam.cp}}</p>
  <p>Nom : {{villeParam.nom}}</p>
</span>

<div *ngIf="villes">
  <p>Les villes</p>
  <p *ngFor="let v of villes">
    {{v.cp}} : {{v.nom}}
  </p>
</div>

<!-- <span *ngIf="error">
  <p>Code erreur : {{error.error}}</p>
  <p>Erreur : </p>
</span> -->

<p>{{message}}</p>
```



## Le composant : villeForm.component.ts.

### Commentaires du code :

Importation du service et de la classe Ville.  
Le service comme « providers ».

Déclaration et instanciation d'un objet « Ville » initialisé à vide.  
Déclaration d'un tableau de « Ville ».

Le constructeur paramétré.  
La méthode clear() qui du fait du « binding » nettoie l'écran et la méthode afficherVille() qui sollicite la méthode du service et exploite le résultat.

### Code :

```
// villeForm.component.ts
import { Component } from '@angular/core';
import { VilleService } from '../ville.service';
import { Ville } from '../ville';

@Component({
  selector: 'app-root',
  templateUrl: './villeForm.component.html',
  providers: [VilleService],
  styles: ['.error {color: red;}']
})

export class VilleFormComponent {
  ville = new Ville('', '');
  villeParam: Ville; // Pour plus tard
  villes: Ville[];
  message: string;
  titre: string = "JSON";
  error: any; // Pour plus tard

  constructor(private villeService: VilleService) { }

  clear() {
    this.ville.cp = undefined;
    this.ville.nom = undefined;
    this.villeParam = undefined;
    this.villes = undefined;
    this.error = undefined;
    this.message = "";
  }

  afficherVilles() {
    // Requête AJAX
    let req = this.villeService.getVilles();
    req.subscribe(data => {
      this.villes = data as Ville[];
    });
  }
}
```

## Le service : villes.service.ts.

### Commentaires du code :

Ajout, par rapport au code de la section précédente, de la méthode getVilles() et du type de source – pour infos - .

```
// ville.service.ts
import { Injectable } from '@angular/core';
import { HttpClient } from '@angular/common/http';

@Injectable()
export class VilleService {
  constructor(private http: HttpClient) { }

  // Les infos sont dans un fichier nommé "villes.json"
  type = 'json';

  // Les infos sont dans une table nommée "villes" de la BD "cours"
  //type = 'mysql';

  url = 'http://localhost:80/AngularRessources/json/GetVille.php';

  getVille() {
    return this.http.get(this.url);
  }

  getVilles() {
    let url = 'http://localhost:80/AngularRessources/' + this.type +
'/GetVilles.php';
    return this.http.get(url);
  }
}
```

---

## 6.4 - 3E EXEMPLE : REQUÊTE GET PARAMÉTRÉE, AFFICHER UNE VILLE

Les données sont stockées dans un fichier nommé « villes.json ».

Ce fichier est traité par un script PHP nommé « GetVillesParam.php ».

Ces fichiers sont stockés dans l'arborescence d'un site web dans un dossier nommé « AngularRessources/json/ ».

```
<?php

/*
 * GetVillesParam.php
 * http://localhost/AngularRessources/json/GetVillesParam.php?cp=75000
 */

header("Access-Control-Allow-Methods: POST, GET, OPTIONS");
header("Access-Control-Allow-Headers: *");
header("Access-Control-Allow-Origin: *");

// Récupération du contenu du fichier sous forme de flux de caractères
$content = file_get_contents("villes.json");

// json_decode(chaine, tableau_associatif = true)
// chaine: It is encoded string which must be UTF-8 encoded data
$objets = json_decode($content, true);

// Récupération du paramètre
$cp = filter_input(INPUT_GET, "cp");
if ($cp != null) {
    /*
     * Traitement : Boucle sur le tableau d'objets
     */
    $message = "";
    $objet = array();
    $trouve = false;
    foreach ($objets as $indice => $objet) {
        if ($objet["cp"] == $cp) {
            $objet["cp"] = $objet["cp"];
            $objet["nom"] = $objet["nom"];
            $trouve = true;
        }
    }
    if (!$trouve) {
        $objet[$cp] = "Introuvable";
    }
} else {
    $objet["Erreur"] = "Input manquant";
}
// « Affichage » du contenu de l'objet
echo json_encode($objet);
?>
```

<http://localhost/AngularRessources/json/GetVillesParam.php?cp=75000>

### Le service (ajout de la méthode getVilleParam()).

URL avec un paramètre de type GET.

```
getVilleParam(asCp) {  
    let url = 'http://localhost:80/AngularRessources/' + this.type +  
    '/GetVillesParam.php?cp=' + asCp;  
    return this.http.get(url);  
}
```

### Le component (ajout de la méthode afficherVillesParam()).

```
afficherVillesParam() {  
    this.villeService.getVilleParam('75011')  
        .subscribe(data => {  
            this.villeParam = data as Ville;  
        });  
}
```

---

## 6.5 - 4E EXEMPLE : REQUÊTE POST, AJOUTER UNE VILLE

Les données sont stockées dans un fichier nommé « villes.json ».  
Ce fichier est traité par un script PHP nommé « InsertVille.php ».  
Ces fichiers sont stockés dans l'arborescence d'un site web dans un dossier nommé « AngularRessources/json/ ».

```
<?php

/*
 * InsertVille.php
 */

header("Access-Control-Allow-Methods: POST, GET, OPTIONS");
header("Access-Control-Allow-Headers: *");
header("Access-Control-Allow-Origin: *");

// Le fichier source et destination
$nomFichier = "villes.json";
$tMessage = array();

/**
 * INSERT
 */
$cp = filter_input(INPUT_POST, "cp");
$nom = filter_input(INPUT_POST, "nom");

if ($cp != null && $nom != null) {

    // Récupération du contenu du fichier sous forme de flux de caractères
    $contenuFichier = file_get_contents($nomFichier);

    // json_decode(chaine, tableau_associatif = true)
    // chaine: It is encoded string which must be UTF-8 encoded data
    $tObjets = json_decode($contenuFichier, true);

    // Nouvelle ville : un tableau associatif
    $t = array();
    $t["cp"] = $cp;
    $t["nom"] = $nom;

    /*
     * Traitement
     * Ajout de la nouvelle ville dans le tableau des existants
     */
    $tObjets[] = $t;

    // Le tableau -> chaine JSON
    $chaineJSON = json_encode($tObjets);

    // Ecriture dans le fichier
    file_put_contents($nomFichier, $chaineJSON);
}
```

```

        $tMessage["message"] = "Nouvelle ville ajoutée dans le fichier JSON
via PHP";
    } else {
        $tMessage["message"] = "Message serveur : toutes les saisies sont
obligatoires !";
    }

    echo json_encode($tMessage);
?>

```

### Dans ville.service.ts

```

insertVille(asCp, asNom) {
    /* const headers = new HttpHeaders();
    headers.set('Content-Type', 'application/x-www-form-urlencoded');
    */
    let url = "http://localhost:80/AngularRessources/" + this.type +
"/InsertVille.php";
    let body = new HttpParams();
    body = body.set('cp', asCp);
    body = body.set('nom', asNom);
    return this.http.post(
        url,
        body);
}

```

### Dans ville.component.ts

```

ajouterVille() {
    this.villeService.insertVille('75000','Paris')
        .subscribe(data => {
            console.log(data);
            this.message = data["message"];
        })
    ;
    this.message = "ajouterVille : fin";
}

```

---

## 6.6 - VERSION POUR DES DONNÉES MySQL

### 6.6.1 - Les scripts PHP

```
; bd.ini
[section_connexion]
protocole=mysql
serveur=127.0.0.1
port=3306
bd=cours
ut=root
mdp=
```

```
<?php

/**
 * Connexion.php : une bibliothèque
 *
 * seConnecter() : PDO (à partir d'un fichier ini)
 * seDeconnecter() : void
 */
class Connexion {

    /**
     *
     * @param type $psCheminParametresConnexion
     * @return null
     */
    public function seConnecter($psCheminParametresConnexion) {

        $tProprietes = parse_ini_file($psCheminParametresConnexion);

        $lsProtocole = $tProprietes["protocole"];
        $lsServeur = $tProprietes["serveur"];
        $lsPort = $tProprietes["port"];
        $lsUT = $tProprietes["ut"];
        $lsMDP = $tProprietes["mdp"];
        $lsBD = $tProprietes["bd"];

        /*
         * Connexion
         */
        $pdo = null;
        try {
            $pdo = new
PDO("$lsProtocole:host=$lsServeur;port=$lsPort;dbname=$lsBD;", $lsUT,
$lsMDP);

            $pdo->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
            $pdo->setAttribute(PDO::ATTR_AUTOCOMMIT, FALSE);
            $pdo->exec("SET NAMES 'UTF8'");
        } catch (PDOException $ex) {
```

```

        }
        return $pdo;
    }

    /**
     *
     * @param PDO $pcnx
     */
    public function seDeconnecter(PDO &$pcnx) {
        $pcnx = null;
    }
}

?>

```

```

<?php

/**
 * Description of Transaxion
 *
 * @author Pascal
 */
class Transaxion {
    /**
     * Transaxion.php : une bibliotheque
     * initialiser()
     * valider()
     * annuler()
     */

    /**
     *
     * @param PDO $pcnx
     */
    public function initialiser(PDO &$pcnx) {
        $lbOK = true;
        try {
            $pcnx->beginTransaction();
        } catch (PDOException $ex) {
            $lbOK = false;
        }
        return $lbOK;
    }

    /**
     *
     * @param PDO $pcnx
     */
    public function valider(PDO &$pcnx) {
        $lbOK = true;
        try {
            $pcnx->commit();
        } catch (PDOException $ex) {
            $lbOK = false;
        }
    }
}

```



```
        return $lbOK;
    }

    /**
     *
     * @param PDO $pcnx
     */
    public function annuler(PDO &$pcnx) {
        $lbOK = true;
        try {
            $pcnx->rollback();
        } catch (PDOException $ex) {
            $lbOK = false;
        }
        return $lbOK;
    }
}
```

```
<?php

/*
 * GetVilles.php
 */

header("Access-Control-Allow-Methods: POST, GET, OPTIONS");
header("Access-Control-Allow-Headers: *");
header("Access-Control-Allow-Origin: *");

require_once 'Connexion.php';
require_once 'Ville.php';
require_once 'VilleDAO.php';

$cnx = new Connexion();
$pdo = $cnx->seConnecter("bd.ini");

$dao = new VilleDAO($pdo);
$tRecords = $dao->selectAll();
$t = array();
foreach ($tRecords as $objet) {
    $tObjet = array();
    $tObjet["cp"] = $objet->getCp();
    $tObjet["nom"] = $objet->getNomVille();
    $t[] = $tObjet;
}

$content = json_encode($t);

// Affichage du contenu du fichier
echo $content;
?>
```

```

<?php

/*
 * GetVillesParam.php
 * http://localhost/AngularRessources/mysql/GetVillesParam.php?cp=75000
 */

header("Access-Control-Allow-Methods: POST, GET, OPTIONS");
header("Access-Control-Allow-Headers: *");
header("Access-Control-Allow-Origin: *");

$cp = filter_input(INPUT_GET, "cp");

if ($cp != null) {

    require_once 'Connexion.php';
    require_once 'Ville.php';
    require_once 'VilleDAO.php';

    $cnx = new Connexion();
    $pdo = $cnx->seConnecter("bd.ini");

    $dao = new VilleDAO($pdo);
    $tRecords = $dao->selectOne($cp);
    if ($tRecords->getCp() != 0) {
        $tObjet = array();
        $tObjet["cp"] = $tRecords->getCp();
        $tObjet["nom"] = $tRecords->getNomVille();
    } else {
        $tObjet = array();
        $tObjet["cp"] = "Introuvable";
        $tObjet["nom"] = "";
    }
} else {
    $tObjet = array();
    $tObjet["cp"] = "Message serveur : toutes les saisies sont
obligatoires !";
    $tObjet["nom"] = "";
}

echo json_encode($tObjet);
?>

```

```

<?php

/*
 * InsertVille.php
 */

header("Access-Control-Allow-Methods: POST, GET, OPTIONS");
header("Access-Control-Allow-Headers: *");
header("Access-Control-Allow-Origin: *");

$message = array();

/**
 * INSERT
 */
$cp = filter_input(INPUT_POST, "cp");
$nom = filter_input(INPUT_POST, "nom");

if ($cp != null && $nom != null) {

    require_once 'Connexion.php';
    require_once 'Transaction.php';
    require_once 'Ville.php';
    require_once 'VilleDAO.php';

    $cnx = new Connexion();
    $tx = new Transaction();

    $pdo = $cnx->seConnecter("bd.ini");

    $ville = new Ville($cp, $nom, "033");
    $dao = new VilleDAO($pdo);

    /**
     * INSERT
     */
    $tx->initialiser($pdo);
    $affected = $dao->insert($ville);
    if ($affected == 1) {
        $lboK = $tx->valider($pdo);
        $message["message"] = "Nouvelle ville ajoutée dans la BD via
PHP";
    } else {
        $lboK = $tx->annuler($pdo);
        $message["message"] = "Probleme d ajout d une ville dans la BD
via PHP";
    }

} else {
    $message["message"] = "Message serveur : toutes les saisies sont
obligatoires !";
}

echo json_encode($message);
?>

```

### 6.6.2 - Les modifications dans la classe de type Service

Juste le chemin vers les scripts situés dans le dossier MySQL.

```
// ville.service.ts
import { Injectable } from '@angular/core';
import { HttpClient, HttpHeaders, HttpParams } from
 '@angular/common/http';

@Injectable()
export class VilleService {
  constructor(private http: HttpClient) { }

  // Les infos sont dans un fichier nommé "villes.json"
  //type = 'json';

  // Les infos sont dans une table nommée "villes" de la BD "cours"
  type = 'mysql';

  getVilleParam(asCp) {
    console.log("getVilleParam dans VilleService");
    let url = 'http://localhost:80/AngularRessources/' + this.type +
    '/GetVillesParam.php?cp=' + asCp;
    return this.http.get(url);
  }

  getVilles() {
    let url = 'http://localhost:80/AngularRessources/' + this.type +
    '/GetVilles.php';
    return this.http.get(url);
  }

  insertVille(asCp, asNom) {
    let url = "http://localhost:80/AngularRessources/" + this.type +
    "/InsertVille.php";
    let body = new HttpParams();
    body = body.set('cp', asCp);
    body = body.set('nom', asNom);
    return this.http.post(
      url,
      body);
  }
}
```

---

## 6.7 - EXERCICE : LA MÊME CHOSE MAIS AVEC UN « FORMULAIRE »

+ supprimer une ville et modifier une ville.

## 7 - ROUTES

---

## 7.1 - PRÉPARATION

**C:\pascal\\_\_\_supports\Angular\AngularV5\projetRoutes**

<https://angular.io/tutorial/toh-pt5>

Nouveau projet :

```
| $ ng new projetRoutes
```

Le code de app.module.ts

```
| // app.module.ts
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';

import { AppComponent } from './app.component';

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



Dans le dossier du projet puis /src/app/

```
| ng generate module app-routing --flat --module=app
```

Affiche :

```
C:\pascal\__supports\Angular\AngularV5\projet2\src\app>ng generate module app-routing --flat --
module=app
CREATE src/app/app-routing.module.spec.ts (308 bytes)
CREATE src/app/app-routing.module.ts (194 bytes)
UPDATE src/app/app.module.ts (1823 bytes)
```

Cela crée les fichiers app-routing.module.spec.ts et app-routing.module.ts et modifie le fichier app.module.ts

```
// app-routing.module.spec.ts
import { AppRoutingModuleModule } from './app-routing.module';

describe('AppRoutingModule', () => {
  let appRoutingModule: AppRoutingModuleModule;

  beforeEach(() => {
    appRoutingModule = new AppRoutingModuleModule();
  });

  it('should create an instance', () => {
    expect(appRoutingModule).toBeTruthy();
  });
});
```

```
// app-routing.module.ts
import { NgModule } from '@angular/core';
import { CommonModule } from '@angular/common';

@NgModule({
  imports: [
    CommonModule
  ],
  declarations: []
})
export class AppRoutingModuleModule { }
```

## Les nouveautés !!!

```
// app.module.ts
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
import { RouterModule } from '@angular/router';

import { ROUTES } from './app-routing.module';

import { Accueil } from './accueil';
import { Authentification } from './authentification';
import { Inscription } from './inscription';

@NgModule({
  declarations: [
    Accueil,
    Authentification,
    Inscription
  ],
  imports: [
    BrowserModule,
    RouterModule.forRoot(ROUTES)
  ],
  providers: [],
  bootstrap: [Accueil]
})
export class AppModule { }
```

---

## 7.2 - EXEMPLE

```
// app-routing.module.ts
import { Routes } from '@angular/router';

import { Accueil } from './accueil';
import { Authentification } from './authentification';
import { Inscription } from './inscription';

export const ROUTES: Routes = [
  { path: '', redirectTo: '', pathMatch: 'full' },
  { path: 'accueil', component: Accueil },
  { path: 'authentification', component: Authentification },
  { path: 'inscription', component: Inscription }
];
```

---

## 7.3 - EXERCICE

## 8 - DÉPLOIEMENT

---

## 8.1 - VERSION SIMPLIFIÉ

<https://angular.io/guide/deployment>

```
ng build --base-href= /dossierDeBase/
```

Note : / représente la racine du site WEB, htdocs par exemple.

La production est générée dans le dossier /dist du projet.

```
| ng build --base-href=/projetHTTP/
```

Ensuite vous copiez le contenu du dossier /dist sur un serveur Apache ... par exemple dans  
C:\xampp\htdocs\

<http://localhost/projetHTTP/index.html>

## 9 - ANNEXES

---

## 9.1 - VERSIONS

```
| $ node -v  
8.11.1
```

```
| $ npm - v  
5.6.0
```

```
| $ ng - v  
Angular CLI: 6.0.8  
Node: 8.11.1  
OS: win32 x64  
Angular:  
...
```

Package	Version
@angular-devkit/architect	0.6.8
@angular-devkit/core	0.6.8
@angular-devkit/schematics	0.6.8
@schematics/angular	0.6.8
@schematics/update	0.6.8
rxjs	6.2.0
typescript	2.7.2



---

## 9.2 - DÉMARCHE DE BASE

- ✓ Création d'un projet
- ✓ Compilation du projet
- ✓ Ouverture du projet avec Visual Studio Code
- ✓ Création du fichier .css (la mise en forme de la vue)
- ✓ Création du fichier .html (la vue)
- ✓ Création d'un composant (« le contrôleur »)
- ✓ Création du selector (dans le fichier index.html qui est dans le dossier /src du projet)
- ✓ Référencement(s) dans le fichier app.module.ts

### Création d'un projet

```
| $ ng new diapo
```

### Compilation du projet nouvellement créé (allez dans le dossier du projet)

```
| $ ng serve -o
```

### Ouverture du projet avec Visual Studio Code

En se situant dans le dossier /projet/src/app.

### Création d'un fichier css

No comment !

### Création d'une « vue » HTML

Par exemple une div avec une <img>.

**Création d'un composant (component) ;** c'est une classe TypeScript donc ClasseComponent.ts.  
Le « bon » squelette est le suivant :

```
// NomDeLaClasse.ts
import { Component } from '@angular/core';

@Component({
  // Sélecteur présent dans un fichier .html autre que le template
  selector: 'nom-selector',
```

```

        templateUrl: 'chemin/fichier.html',
        styleUrls: ['chemin/fichier.css']
    })

    export class NomDeLaClasse {
    }

```

## Création d'un « selector »

Une balise perso qui va « recevoir » la vue par exemple <app-racine> dans le fichier index.html qui est dans le dossier /src de l'application.

## Le fichier app.module.ts

```

// app.module.ts
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { HttpClientModule } from '@angular/common/http';
import { FormsModule } from '@angular/forms';

import { AppComponent } from './app.component';

@NgModule({
  declarations: [
    AppComponent
  ],
  imports: [
    BrowserModule,
    // import HttpClientModule after BrowserModule.
    HttpClientModule,
    FormsModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }

```

---

## 9.3 - SCRIPTS PROJET2

```
// app.module.ts

import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
import { FormsModule } from '@angular/forms';
import { HttpClientModule } from '@angular/http';
/*
*/
import { MyComponent } from './app.component';
import { MyItem } from './item.component';
import { Todo } from './todo';

//import { NumberService } from './number-service';
import { NumberComponent } from './number.component';

/*
Le module HttpClientModule enregistre une dépendance sur la classe http.
Une fois ce module importé, il est possible d'injecter cette classe dans
tous les éléments ayant besoin d'effectuer des requêtes HTTP.
*/

@NgModule({
  declarations: [
    MyComponent,
    MyItem, // Déclaration du ItemComponent
    Todo,
    NumberComponent
  ],
  imports: [
    BrowserModule,
    FormsModule,
    HttpClientModule
  ],
  providers: [],
  bootstrap: [MyComponent, MyItem, Todo, NumberComponent]
})
export class AppModule { }
```

---

## 9.4 - PROJET TODOList

### Objectif

Un écran pour l'inscription.

Un écran pour l'authentification.

Un écran pour visualiser la TodoList.

Un écran pour mettre à jour la TodoList (Ajout, suppression, modification).

### Création du projet

```
| $ cd /pascal/___supports/Angular/AngularV5/
```

```
| $ ng new todolist
```

### Les écrans

[Authentification](#) [Inscription](#) [Todo List](#)

### Authentification

User :

Password :

Message ici !

[Authentification](#) [Inscription](#) [Todo List](#)

## Inscription

User :

Password :

Email :

Téléphone :

Message ici !

---

## 9.5 - FRONT-END, BACK-END, ETC

Terme	Description
Font-end	Partie de l'application qui gère l'interface utilisateur (principalement du code HTML, CSS, JavaScript, AJAX dans le cadre d'une application web).
Back-end	Partie de l'application qui gère les données des serveurs (principalement du PHP, du SQL, ...). La communication avec la partie Front est écrite en AJAX coté client et correspond à des services REST côté serveur.
Front-Office	Application interne (Intranet pour la gestion du contenu par exemple).
Back-Office	Application externe (Site interne ou application mobile pour le client final par exemple).

---

## 9.6 - INDEX DES TABLEAUX

### Index des tables

Titre.....	1
Historique.....	5
ContenuApp.....	13
Clic.....	36
ExerciceRecupererSelectionDansListe.....	45
Terminologie.....	91

## **Index des illustrations**

Logo.....	1
1er écran.....	9
Arborescence 1er projet.....	10
DCL Projet Base.....	16
2e écran.....	27
Exercice liste.....	32
Bouton Cliquer avant.....	36
Bouton Cliquer après.....	36
DSEQ Clic.....	36
Récupérer valeur saisie.....	42
Exercice Addition.....	44
Exercice Liste avant.....	45
Exercice Liste après.....	45
Formulaire 1.....	52
Formulaire 2.....	52
Exercice Formulaire.....	56
Exercice Formulaire Calculs.....	62
Afficher JSON Villes.....	66
DCL HTTP 1.....	67
CRUD Villes JSON.....	71