Introduction

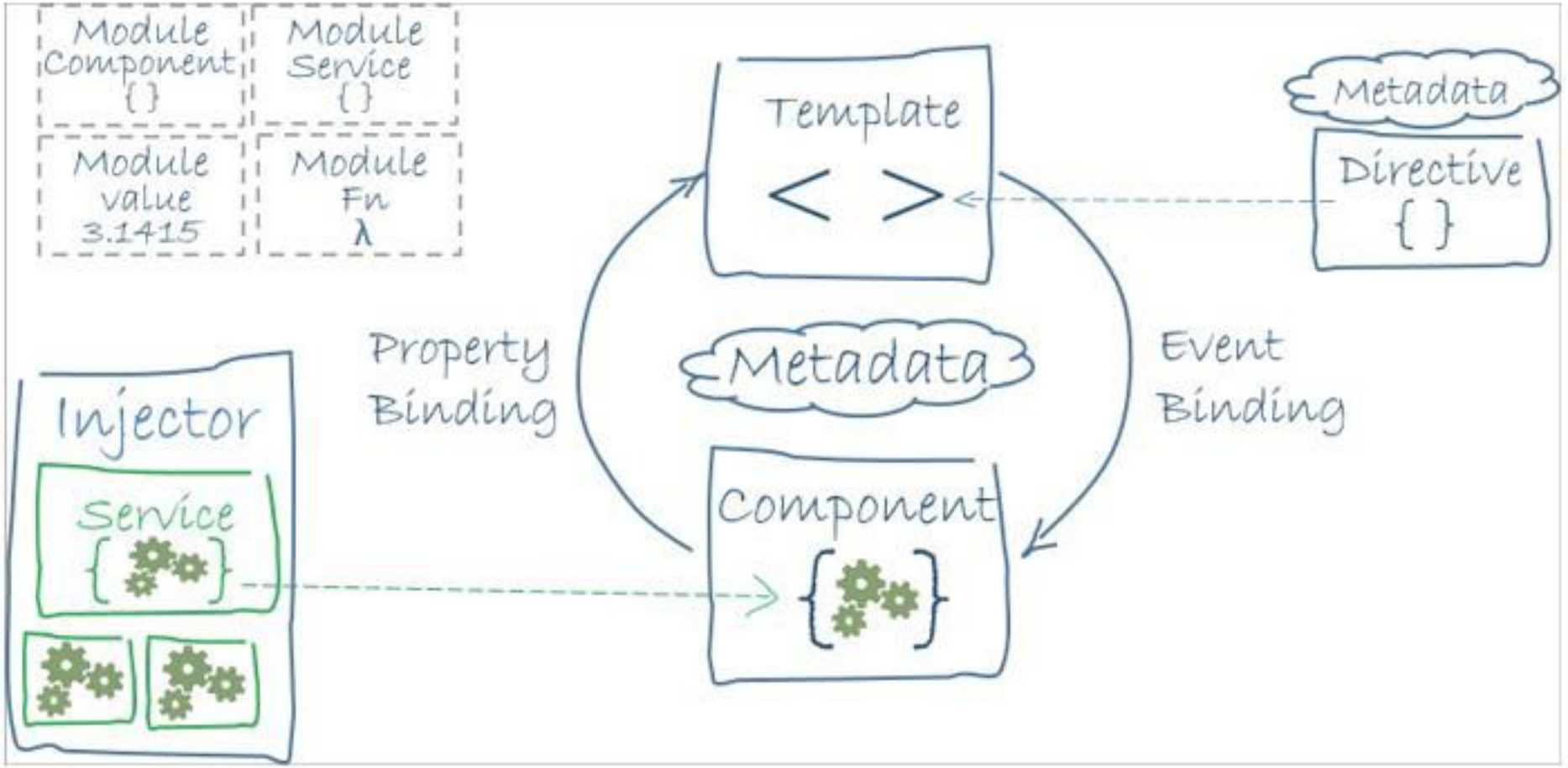
Introduction

- Framework **JavaScript** (ou **TypeScript**)
- Présenté par **Google** en 2009
- Permettant de créer des applications web et mobiles
  - **Front-End** (l'ensemble des éléments visibles et accessibles directement sur une application web ou une application web mobile)
  - **Single page Application** (Une SPA est une application qui contient une seule page HTML (index.html) récupérée du serveur).
- Respecte l'architecture MVVM (Model-View-ViewModel)
  - **Model** : représenté généralement par une classe référencée par la couche d'accès aux données (classe ou interface **TypeScript**).
  - **View:** contenant la disposition et l'apparence de ce qu'un utilisateur voit à l' écran, recevant l'interaction avec l'utilisateur : clic, saisie, survol...
  - **ViewModel:** remplaçant du contrôleur dans l'architecture **MVC**, connecté à la vue par le **data binding**, représenté dans **Angular** par un fichier *.**component.ts**.
- Angular utilise les langages suivants:
  - **HTML** pour les vues. **HTML** pour les vues.
  - **CSS** pour les styles.
  - **TypeScript** pour les scripts depuis la version 2.

Introduction-Historique

- **Angular 1 (Agular JS) :**
  - Première version de Angular qui est la plus populaire.
  - Elle est basée sur une architecture MVC coté client. Les applications Angular 1 sont écrite en Java Script.
- **Angular 2 (Angular) :**
  - Est une réécriture de Angular 1 qui est plus performante, mieux structurée et représente le futur de Angular.
  - Les applications de Angular2 sont écrites en Type Script qui est compilé et traduit en Java Script avant d'être exécuté par les Browsers Web.
  - Angular 2 est basée sur une programmation basée sur les Composants Web (Web Component)
- **Angular 4, 5, 6,7,8,… :** sont de simples mises à jour de Angular 2 avec des améliorations au niveau performances.

Architecture de base d'une application Angular



Installation

- Pour installer **Angular**, il faut télécharger et installer **NodeJS** (Dernière version stable LTS)
- **Pour installer Angular, exécuter la commande**

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

- **Pour installer une version bien précise (par exemple 8)**

```
npm install -g @angular/cli@8
```

- **Pour vérifier la version d'Angular installée, exécuter la commande**

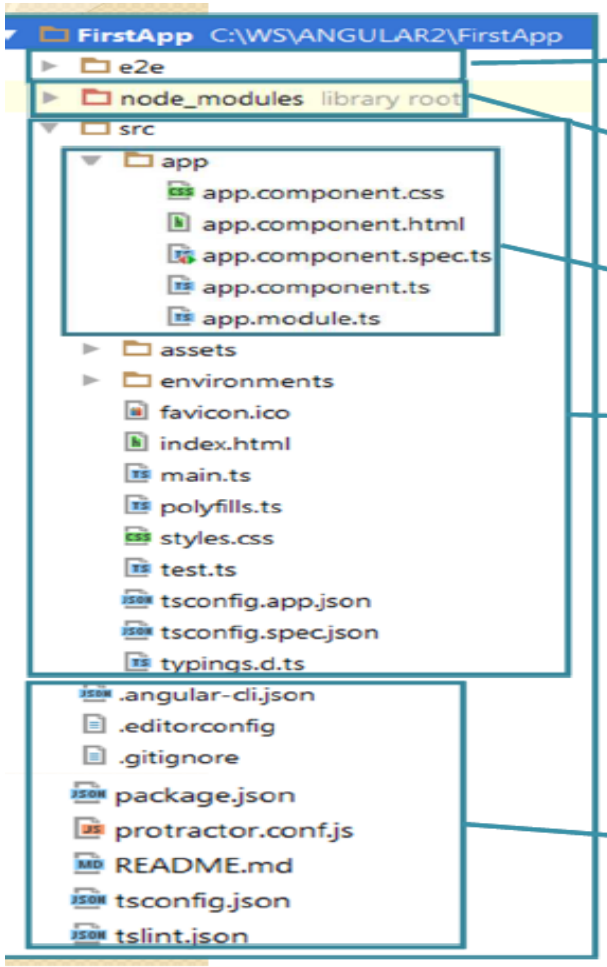
```
ng version
```

- **npm** (**n**ode **p**ackage **m**anager) : le gestionnaire de paquets par défaut pour une application **JavaScript**.
- **angular-cli** (**c**ommand **l**ine **i**nterface) : outil proposé par **Google** pour faciliter la création et la construction d'une application **Angular** en exécutant directement des commandes.

Commandes CLI à utiliser fréquemment

Commandes	Besoins
ng version	afficher la version de angular Cli
ng new firstApp	Créer un projet angular
ng serve (ou npm start)	Exécuter un projet
ng serve -o	lancer le projet et ouvrir une nouvelle fenêtre dans le navigateur
ng g component componentName (ng g c componentName), *(ng g c ComponentName - -skip-tests=true)*	Créer un composant
ng g service serviceName	Créer un service
ng g class className	Créer une classe
ng g i interfaceName	Créer une interface

Structure du Projet Angular



Dossier des scripts des tests unitaires

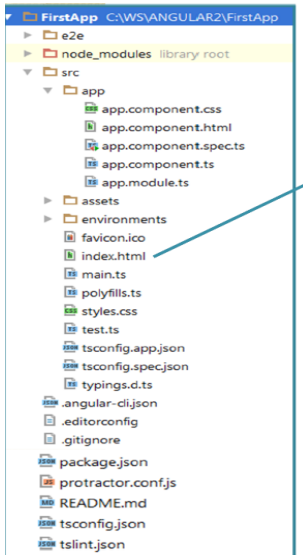
Dépendances externes du projets (Librairies Java Script et CSS)

Logique applicative de votre projet : Les composants, Les services ..
C'est dans ce dossier app ou vous allez passer votre temps de dev

Code source relatif à votre projet

Fichier de configuration du projet

Structure du Projet Angular



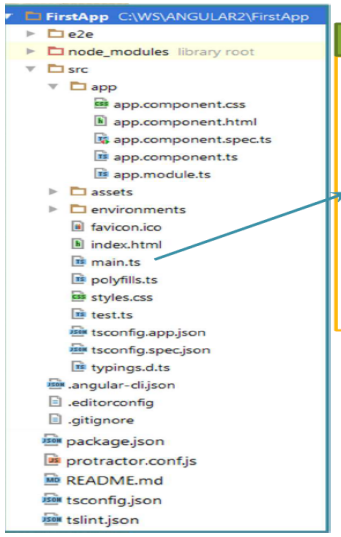
index.html

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



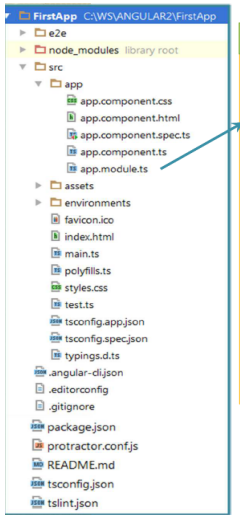
main.ts

```
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);
```



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 { }
```

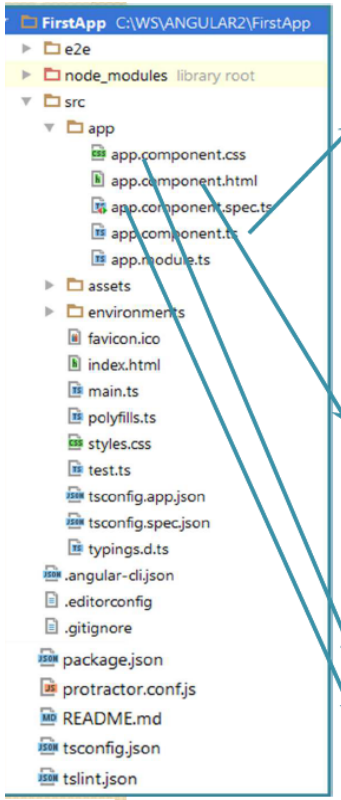
Declarations : C'est le tableau de composants. Si un nouveau composant est créé, il sera importé en premier et la référence sera incluse dans les déclarations

Imports : C'est un tableau de modules requis pour être utilisé dans l'application

Providers: Cela va contenir tous les services crées

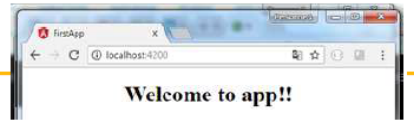
Bootstrap : Cela inclut le composant principal de l'application pour démarrer l'exécution

Structure du Projet Angular



app.component.ts

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



app.component.html

```
<div style="text-align:center">
  <h1>
    Welcome to {{title}}!!
  </h1>
</div>
```

app.component.css

Les fichiers spec sont des tests unitaires pour vos fichiers source. La convention pour les applications Angular2 est d'avoir un fichier .spec.ts pour chaque fichier .ts. Ils sont exécutés à l'aide du framework de test javascript Jasmine via le programme de tâches Karma lorsque vous utilisez la commande '**ng test**'.

Chaque composant se compose principalement des éléments suivants :

- HTML Template : représentant sa vue
- Une classe représentant sa logique métier (.ts)
- Une feuille de style CSS
- Un fichier spec sont des tests

Un composant peut être inséré dans n'importe quelle partie HTML de l'application en utilisant son sélecteur associé.

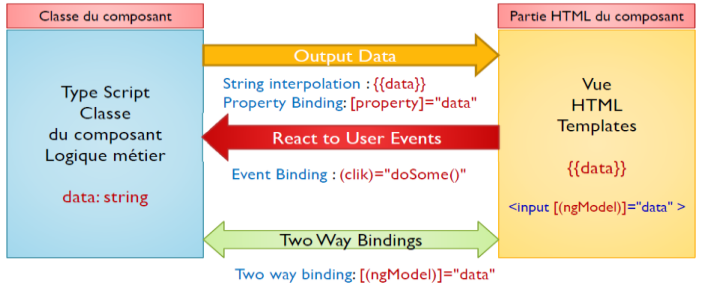
Un composant est une classe qui possède **le décorateur *@Component***. Ce décorateur possède les propriétés suivantes :

- **selector** : permet de spécifier le tag (nom de la balise) associé à ce composant.
- **templateUrl** : permet d'associer un fichier externe HTML contenant la structure de la vue du composant
- **styleUrls** : spécifier les feuilles de styles CSS associées à ce composant

DataBinding

- Pour insérer dynamiquement des données de l'applications dans les vues des composants, Angular définit des techniques pour assurer la liaison des données.

- **Data Binding = Communication**



String interpolation:est une technique de One Way Binding, elle utilise l'expression {{ }} pour afficher les données du composant dans la vue.
Exemple:

```
export class AppComponent {
  title = 'TP 1 Angular';

}
```

app.component.ts

```
<h2>{{title}}</h2>
```

app.component.html

Property Binding : est une autre technique One Way Binding. Elle permet de lier une propriété de la vue avec une propriété définie dans le composant.

Exemple:

```
urlImg="./assets/images/pc portable.jfif"
```

```
<img [src]=urlImg>
```

Event Binding : dans Angular , event binding est utilisé pour gérer les événements déclenchés comme le clic de bouton, le déplacement de la souris, etc. Lorsque l'événement se produit, il appelle la méthode spécifiée dans le composant.

Exemple:

```
afficher()
{
  console.log("hello");
}
```

```
<button (click)="afficher()">Afficher</button>
```

Two-way binding : nous avons vu que dans la le One Way Binding, tout changement dans la vue n'était pas reflété dans le composant. Pour résoudre ce problème, Angular Two Way Binding.

```
texte:string="hello";
```

```
<input type="text" [(ngModel)]=texte> {{texte}}
```

<u>Remarque :</u> il faut ajouter « ***FormsModule***» dans le tableau ***imports*** du fichier ***app.module.ts***

Les directives

Le DOM (**Document Object Model)** est une **interface de programmation** qui est une représentation du HTML d'une page web et qui permet d'accéder aux éléments de cette page web et de les modifier avec le langage JavaScript. Il faut voir le DOM comme un **arbre** où chaque élément peut avoir zéro ou plusieurs enfants, qui peuvent avoir eux-mêmes zéro ou plusieurs enfants, etc…

- Les directives sont des instructions intégrées dans le DOM.
- Quand Angular lit le template et rencontre une directive qu'il reconnait, il suit les instructions correspondantes.
- **La directive *ngFor** : elle est très utile dans le cas d'un tableau et qu'on a besoin de répéter un traitement donné.

Exemple :

```
<ul>
    <li *ngFor="let p of produits">{{ p.nom }}</li>
</ul>
```

- La directive *ngIf : est utilisée lorsque vous souhaitez afficher ou supprimer un élément en fonction d'une condition. Prend un booléen en paramètre. Tout comme nous sommes habitués à d'autres langages de programmation, la directive angular *ngIf* nous permet également de déclarer un bloc *else*. Ce bloc est affiché si l'instruction définie dans le bloc principal put être fausse.

Exemple :

```
<div *ngIf="x%2 != 0 ;else sinon "> impair</div>
   …
   …
<ng-template #sinon>pair</ng-template>
```

Les directives

- **La directive [ngStyle] :** Cette directive permet de modifier le style d'un élément HTML. Elle s'utilise conjointement avec le property binding pour récupérer des valeurs définies dans la classe.

Exemple:
```
<h4 [ngStyle]="{color: getColor()}">{{produits[0].nom}}</h4>
```

La fonction getColor() retourne la couleur du titre h4. Cette méthode est développée dans le fichier .ts.

- **La directive [ngClass]:**permet d'attribuer de nouvelles classes d'un élément **HTML**. S'utilise conjointement avec le **property binding** pour récupérer des valeurs définies dans la classe ou dans la feuille de style.

Exemple:

On définit deux classes rouge et bleu dans app.component.css

```
.rouge {
    color: red;
       }
 .bleu {
   color: blue;
   }
```

Pour associer la classe rouge à la balise <p>

```
<p [ngClass]="{'rouge': true}">
{{ nom }}
</p>
```

On peut aussi appeler une méthode dans la directive ngClass

Les pipes

- Les Pipes sont des filtres utilisables directement depuis la vue afin de transformer les valeurs lors du "binding".
- Il existe plusieurs pipes natives dans Angular comme par exemple ***DecimalPipe, UpperCasePipe, CurrencyPipe***, etc.
- Exemple: `{{nom|uppercase}},{{prix|currency:'TND':'symbol':'2.2-2'}}`
- Vous trouverez dans ce lien les pipes natives d'Angular : https://angular.io/api?type=pipe