

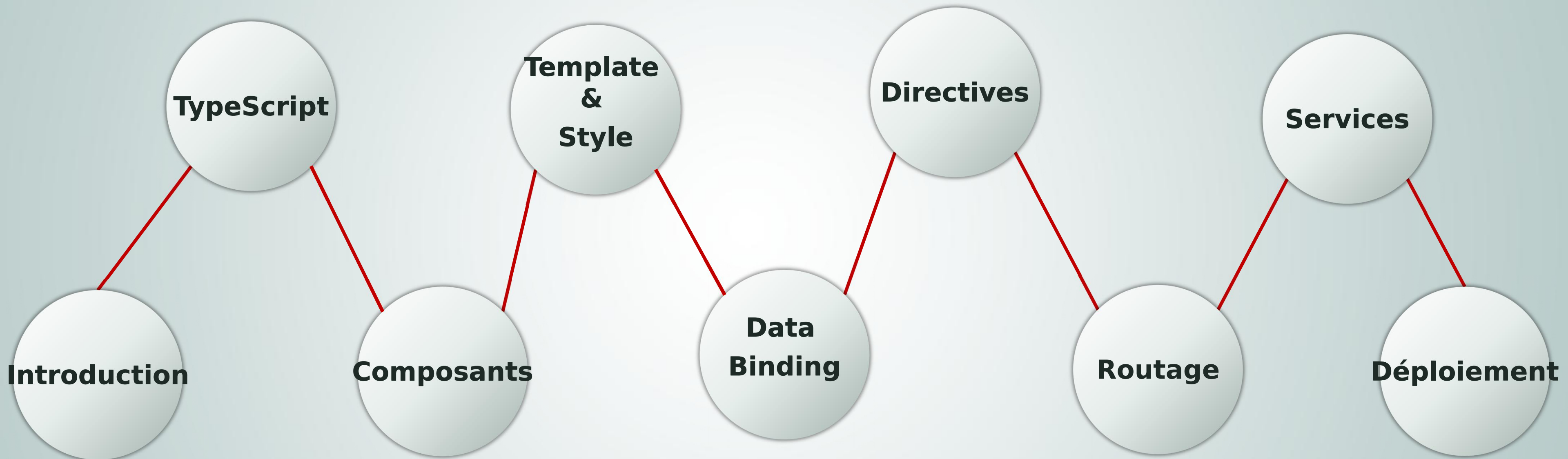


# Introduction à Angular

Safa SAOUDI

# Agenda

2



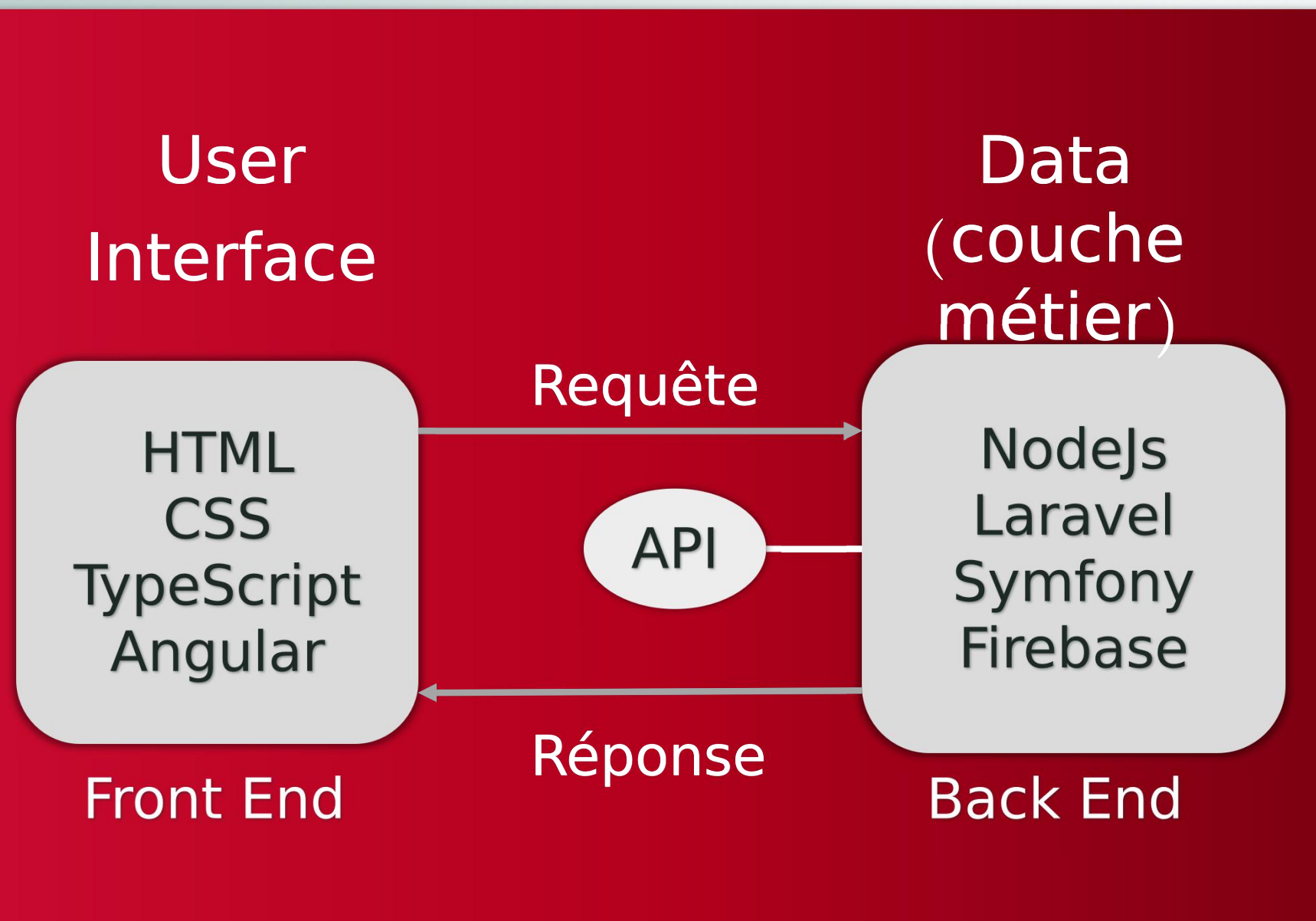
A dark gray rectangular bar is positioned on the left side of the slide, partially overlapping the text.

# Introduction

Architecture Angular

# Architecture d'une App Web

4

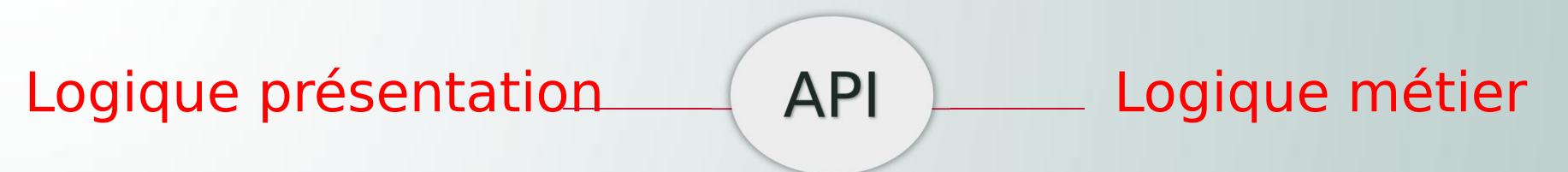


## API

Application Programming Interface

Joue le rôle d'un déclencheur de traitement coté serveur (BackEnd) suite à demande ou Requête (FrontEnd)

En Angular : les demandes == Services [ **≠ Service Web** ]



Développeur FrontEnd + Développeur BackEnd

=

Développeur FullStack

# Les Versions Angular

5

AngularJS

V1: 2009

V 1.6.7 Nov 2017

V 2.0 : Sept 2016

Angular  
2

Angular  
4

V 4.0 : Dec 2017

V 4.3 : Mars 2017

V 5.0 : Nov 2017

Angular  
5

Angular  
6

Avril 2018

# Pourquoi Angular

6

## SPA

Angular est un libre framework JavaScript, développé par Google et utilisé pour créer des applications Web basées sur une seule page (Single Page Application)

## Plateforme

Angular utilise des fonctionnalités de plate-forme Web modernes pour offrir de nouvelles expériences, ayant une installation à haute performance avec zéro-étape. Il permet de créer des applications mobiles natives avec Ionic Framework, NativeScript et React Native ainsi que des applications installées sur votre ordinateur sur Mac, Windows et Linux.

## Performance

Angular transforme les modèles en code optimisé, servant une vue de votre application sur node.js, .NET, PHP, et d'autres serveurs en manipulant que HTML et CSS.

Les applications angulaires se chargent rapidement avec le nouveau routeur de composants



# Préparation de l'environnement

7

## NodeJs

Framework JavaScript coté serveur:

[www.NodeJs.org](http://www.NodeJs.org)

npm install -g npm@latest

## Angular CLI

Command Line Interface: npm install -g

@angular/cli

(npm : Node Package Manager)

## TypeScript

Language d'Angular : npm install -g typescript

## Visual Code Studio

A télécharger depuis:

<https://code.visualstudio.com/>

## Git

A télécharger depuis: <https://git-scm.com/downloads>

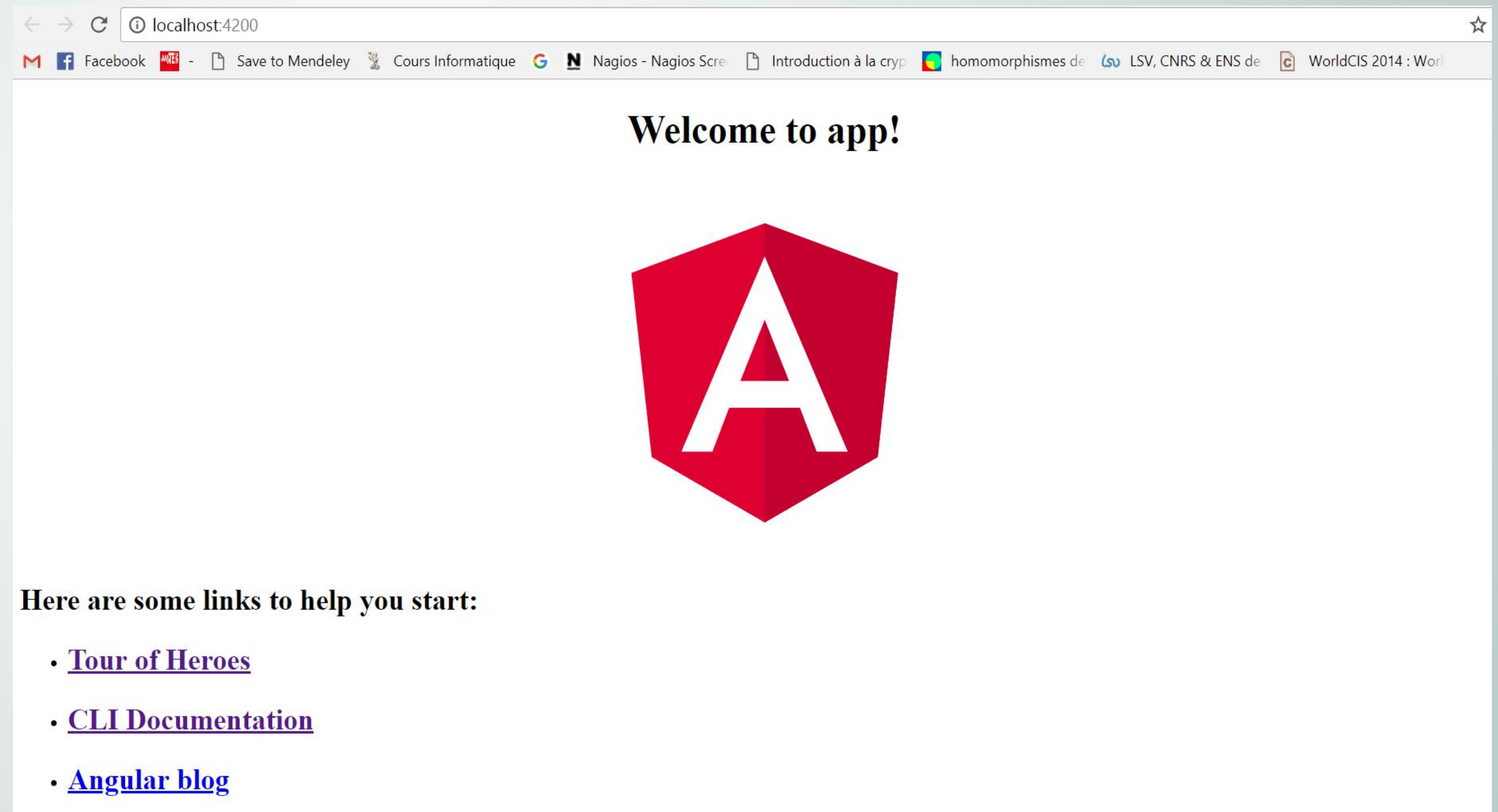
# Lancement première App Angular

8

Créer Votre première application en deux lignes

`ng new nomProjet`

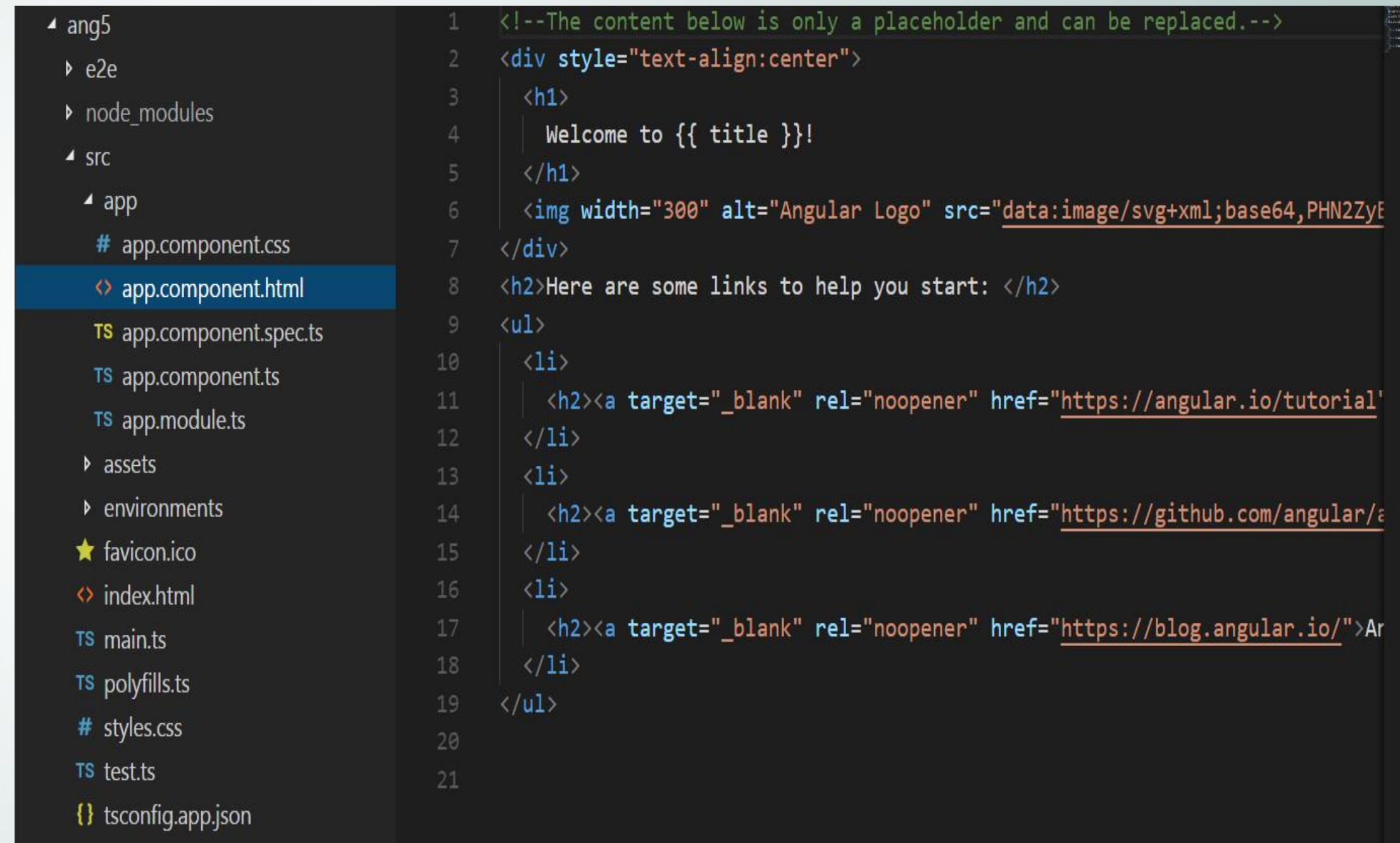
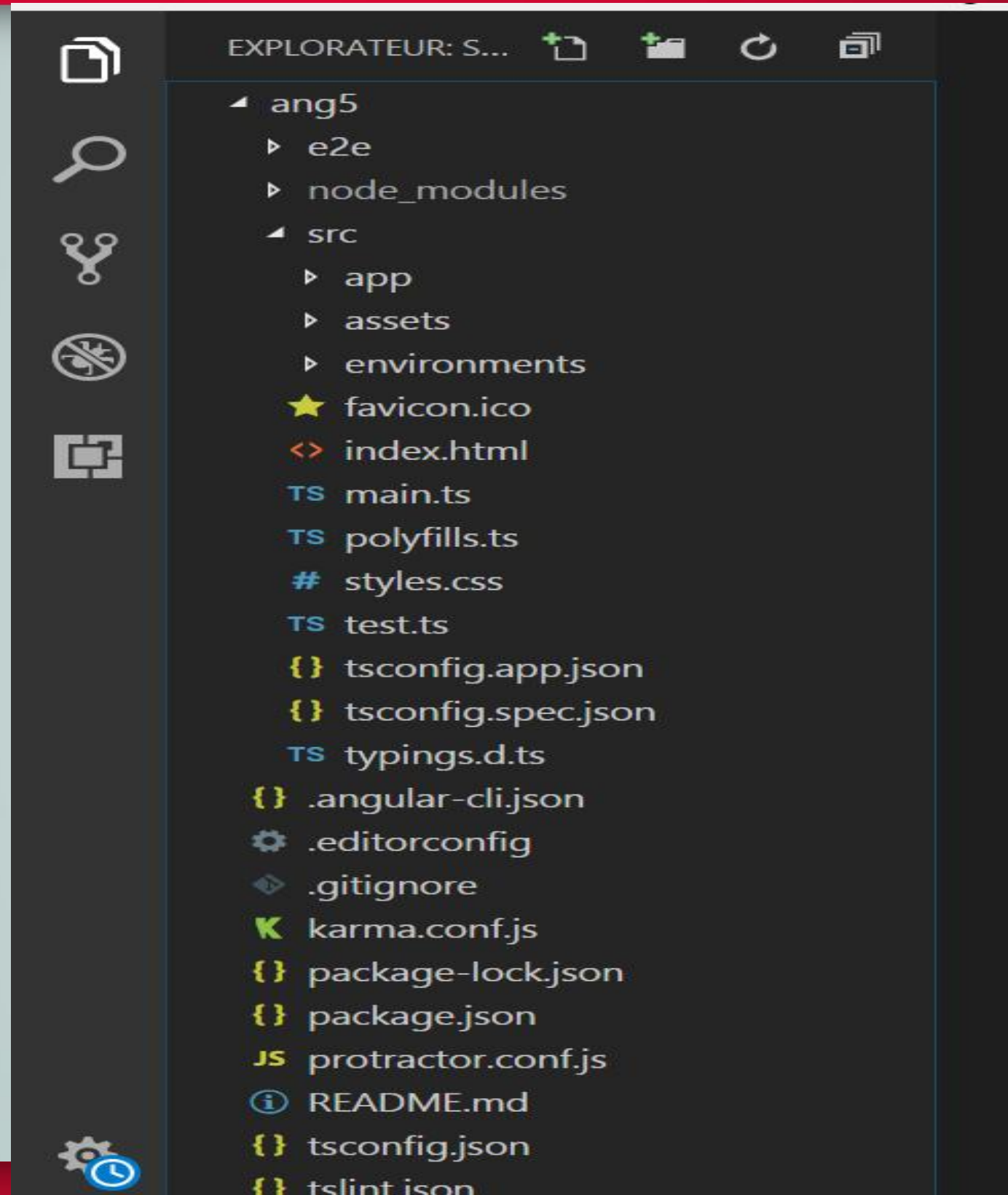
`ng serve --open`





# Architecture Projet Angular

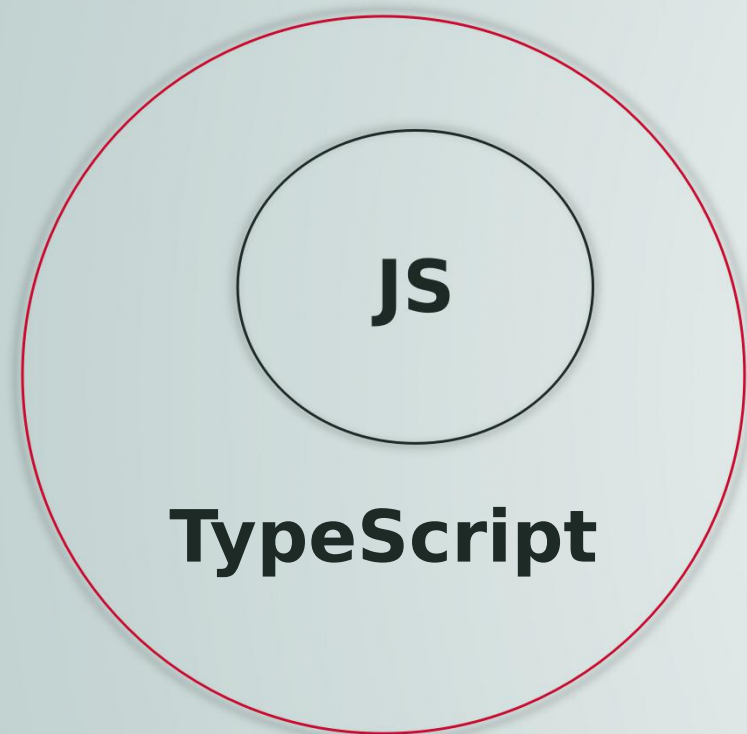
10





TypeScript

# Pourquoi le TypeScript



## Avantages

Transcompilé: en JS pour qu'il soit supporté par les navigateurs

Transtypage : qui permet de s'assurer qu'une variable ou une valeur passée vers ou retournée par une fonction soit du type prévu

POO: TypeScript est modulaire, classes et interfaces pour des applications plus robustes

Riche en fonctionnalités: fonctions lambda

Débogage: il permet de détecter les erreurs syntaxiques

## Compilation

Créer un dossier TypeScript et un fichier que vous nommez main.ts

```
mkdir TypeScript
```

```
code TypeScript/main.ts
```

```
cd Typescript
```

```
tsc main.ts
```

```
ls
```

```
node main.js /ou main.ts
```

## Déclaration des Variables

**var** : déclaration d'une façon globale

**let** : déclaration locale ( si vous voulez protéger vos variables, veuillez utiliser **let** )



## Déclaration des Types

1ère méthode :

La declaration est detectable via l'affectation

```
let age=10;
```

```
age="bonjour";
```

erreur!!

Si le type est any : elle prend n'importe quelle valeur

```
let age;
```

```
age=10;
```

```
age="Bonjour";
```

(consummation de mémoire)

2ème méthode:

```
let age: number;
```

## Les types le base

*number*

*string*

*boolean*

*any*

*array* : let a: number[]; / let a: any[] = [1,'b', true];

*enum* color {red = 1, green = 2, blue = 3};

```
let backgroundColor = color.red;
```

(Have a look on main.js )

Pour plus:

<http://www.typescriptlang.org/docs/handbook/basic-types.html>

## Types Assertion

Sont équivalents aux **cast** dans les autres languages : c'est une façon de dire au compilateur

“Crois-moi, je sais ce que je suis entrain de faire ”

```
let something = " Salut typescript ";
```

```
let x = something.charAt(3);
```

```
let y : number = ( <string!  
something).length ;
```

```
let y : number = (something as  
<string! ).length ;
```

## Arrow function ( Lambda )

Les fonctions anonymes

Exple:

```
let someFunction = (message: string) => {  
    return message;
```

```
}
```

```
let affiche = () => console.log('Salut tout le  
monde ');
```



# Les bases du TypeScript

15

## Interfaces

```
interface Contact {  
    nom :string  
    prenom: string  
    numero: number  
    photo: string  
    email: string  
}  
let createContact = (contact: Contact) => { }
```

```
createContact({  
    nom: 'ben Foulen',  
    prenom: 'Foulen',  
    numero : 123456,  
    photo: 'url',  
    email: 'hhj@gjhcom'  
});
```

## Classes

Regroupement des méthodes et des attributs dans le même endroit => C'est le concept d'Orienté Objet

```
Class Contact {  
    nom :string;  
    prenom: string;  
    afficheContact(){  
        console.log( " nom: " + this.nom, "  
    prenom : " + this.prenom);  
    }  
}
```

## Instanciación d'Objet

```
let contact = new contact ;  
contact.prenom= "Foulen";  
contact.nom= "BenFoulen";  
contact.afficheContact();
```

## Encapsulation

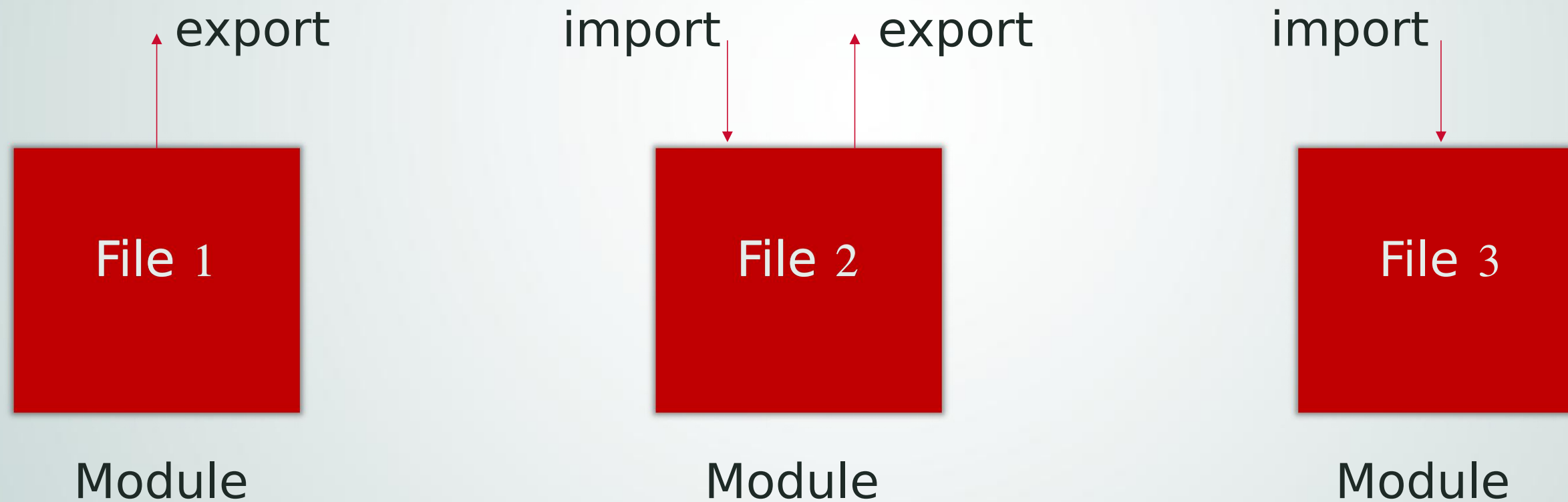
```
private nom: string;
```

Ou dans le constructeur directement

```
constructor (private _nom ? :string, private  
_prenom ? :string)
```

## Les Modules

Il s'agit d'importer un module ou une classe pour quelle soit utilisée dans un autre fichier

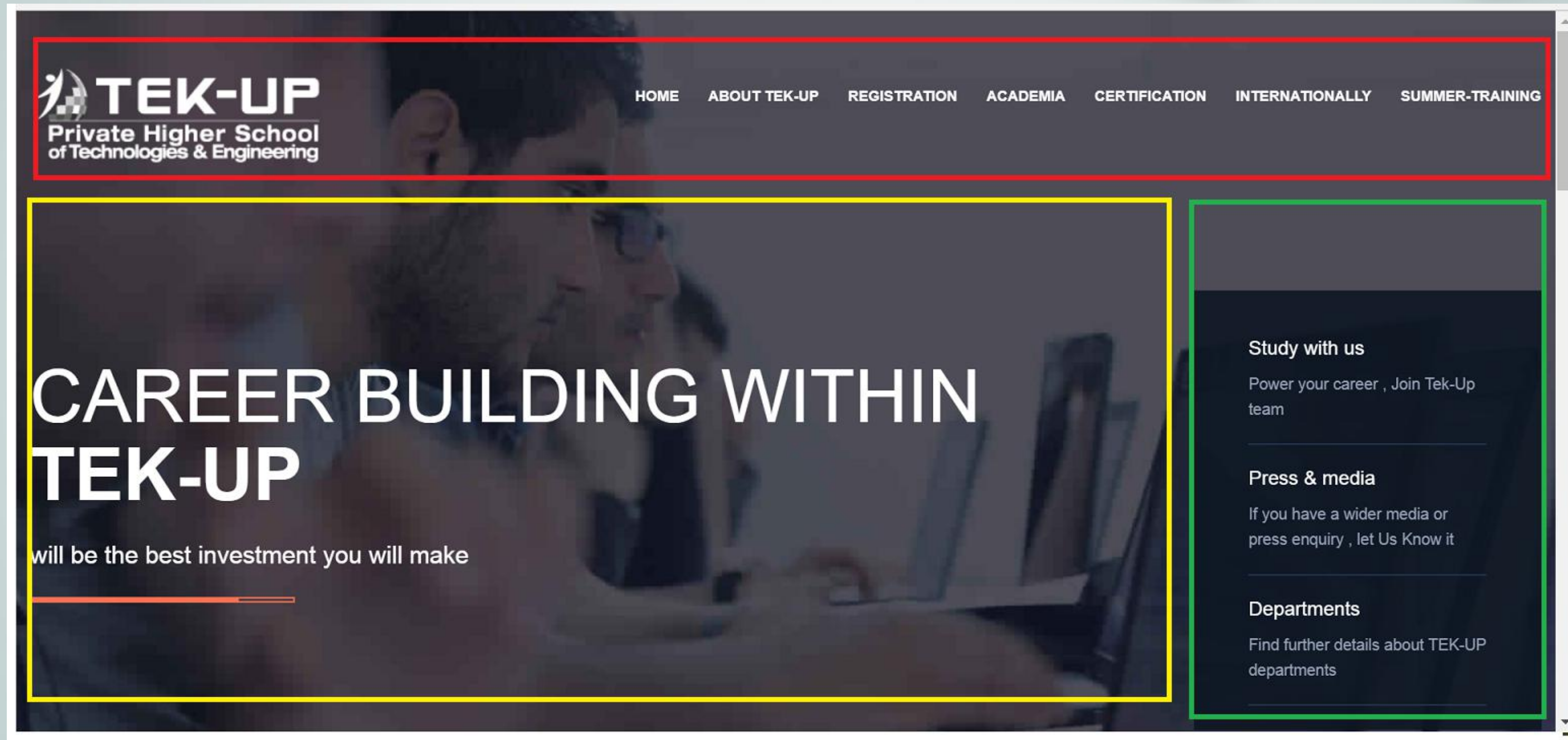




# Les Composants

# Components

19





## AppComponent

[Home](#)

[Users](#)

[Logout](#)

### Contenu

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur.

Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Duis aute irure dolor in

### Side menu

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo

**Un composant** est une **classe** qui permet de gérer une **vue**. Il se charge uniquement de cette vue là. Autrement, un composant est un fragment HTML géré par une classe JS.

Une application Angular est un arbre de composants dont la racine est l'application lancée par le navigateur au lancement. (AppComponent)  
Tous les autres composants seront emboîtés ou 'nested' dans celui-ci.



# Architecture Composant

21

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

- **@Component** décorateur qui permet d'ajouter un comportement à notre classe et de spécifier que c'est un Composant Angular.
- **selector** permet de spécifier le tag (nom de la balise) associé à ce composant.
- **templateUrl**: spécifie l'url du template associé au composant
- **styleUrls**: tableau des feuilles de styles associé à ce composant
- **Export** de la classe afin de pouvoir l'utiliser

# Créer un composant

22

**ng generate component**  
nomComponent

Créant deux composants

Home et About

(une autre façon réduite de création :

ng g c Home

ng g c About

**NB:** chaque component sera déclaré dans le fichier app.module.ts

```
@NgModule({  
  declarations: [  
    AppComponent,  
    HomeComponent,  
    AboutComponent  
  ],  
})
```

# Imbriquer des composants

23

```
<ul>
  <li><a routerLink="">Home</a></li>
  <li><a routerLink="about">About</a></li>
</ul>
```

```
<app-home></app-home>
```

**Composant  
père**

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

  constructor() { }

  ngOnInit() {
  }

}
```

**Composant  
fils**

# Insertion code externe

24

Afin d'envoyer du code dans un composant nous utilisons la balise `<ng-content></ng-content>`.

Cette balise doit être insérée dans le composant qui veut donner la possibilité à son composant parent d'ajouter du contenu externe dans son template.

```
<ul>
  <li><a routerLink="">Home</a></li>
  <li><a routerLink="about">About</a></li>
</ul>

<app-home> Ecrire qlq chose en externe </app-home>
```

```
1  <p>
2    home works!
3  </p>
4  <ng-content></ng-content>
5
```





# Template et Style

styleUrl  
templateUrl



A dark gray rectangular bar is positioned on the left side of the slide, partially overlapping the text.

# Modules

# Structure d'un Module

28

**Le Module** en Angular fait référence à un endroit où vous pouvez regrouper les composants, les directives et les services liés à l'application.

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

import {FormsModule} from '@angular/forms';
import { AppComponent } from './app.component';
import { AboutComponent } from './about/about.component';
import { HomeComponent } from './home/home.component';
```

**le NgModule doit être importé du angular core**

**Ainsi que tout Module qu'on a besoin dans l'application**

# Structure d'un Module

29

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

## Déclaration

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

## Importer

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

## Providers

Cela va contenir tous les services créés

## Bootstrap

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



# Data Binding

# Data Binding

31

Data binding (liaison de données) c'est la communication entre business logic (typescript) et les views (HTML).

**TypeScript**

**(Business  
Logic)**

**Interpolation**

**Property Binding**

**Event Binding**

**Two way Data Binding**

**Template**

**(HTML)**

# Interpolation

32

## Syntax

```
export class <class_name>
{
    variableName = 'any string';
}
```

## Example

```
export class AppComponent
{
    title = 'This is my demo app';
}
```

**Html**

```
<p> {{variableName}} </p>
```

**AppComponent.html**

```
<p> {{title}} </p>
```



# Property Binding

33

```
<form>
  <input type="text" class="txt" name="item" placeholder="Task todo...">
  <input type="submit" class="btn" name="item" [value]= "bnTxt" >
</form>
</div>
```

```
export class HomeComponent implements OnInit {
  cpt :number = 2;
  bnTxt :string = "Add new Item";
  task :string = "Tutorial web";
  constructor() { }
```

# Event Binding

34

```
<form>
  <input type="text" class="txt" name="item" placeholder="Task todo...">

  <input type="submit" class="btn" name="item" [value]= "bnTxt" (click)="ajouItem()" >
</form>
```

```
ngOnInit() {
}

ajouItem() {
  //Un traitement à faire
}
}
```

# Two-way Binding

35

**Two-way Binding = property Binding + event**

**Binding  
Syntaxe:**

**[(ng-Model)]= proprety**

Afin de pouvoir utiliser le two-way binding, il vous faut importer le module **FormsModule** du fichier **forms**

```
import { FormsModule } from '@angular/forms';
import { AppComponent } from './app.component';
import { HomeComponent } from './home/home.component';
import { AboutComponent } from './about/about.component';
```

```
imports: [
  BrowserModule,
  FormsModule
],
```



# Les Directives

Les **directives** sont **des instructions** intégrées dans le DOM que vous utiliserez presque systématiquement quand vous créerez des applications Angular. Elles permettent **d'attacher** un comportement aux éléments du DOM, décorées avec l'annotation **@Directive**.

Il existe deux types principaux de directive :

- **les directives structurelles**
- **les directives par attribut**



Ce sont des directives qui, comme leur nom l'indique, modifient la structure du document ( DOM )

Elles sont généralement précédées par le préfix **\*** .  
Les directives les plus connues sont :

**\*ngIf**

**\*ngFor**

**[ngSwitch]**

## ngFor

Son but est de répéter un modèle HTML donné pour chaque valeur d'un tableau, en lui passant chaque fois la valeur du tableau en tant que contexte pour l'interpolation ou la liaison de données.

Syntaxe : **\*ngFor='let <value> of <collection>'.**

```
<p class="life-container" *ngFor="let task of tasks">
  {{task}}
</p>
```

## ngIf

La directive `ngIf` est utilisée lorsque vous souhaitez afficher ou supprimer un élément en fonction d'une condition. Si la condition est fausse, l'élément auquel la directive est attachée sera supprimé du DOM.

```
<div class="col" *ngIf="cpt !=0">  
  <p class="life-container" *ngFor="let task of tasks">  
    {{task}}  
  </p>  
</div>
```

En Inspectant l'élément: on vérifie la structure du DOM

# Directives Structurelles

41

## ngIf else (Ang 4 /5 )

```
<div *ngIf = 'condition ;else  
other_content' |  
    content here ...  
</div>  
<ng-template #other_content |  
    other content here..  
</ng-template>
```

```
<div class="col" *ngIf="cpt !=0; else alternative2">  
    <p class="life-container" *ngFor="let task of tasks">  
        {{task}}  
    </p>  
</div>  
<ng-template #alternative2>  
    <p class="life-container" >  
        Aucun Task ajouté  
    </p>  
</ng-template>
```

## ngSwitch / ngSwitchCase

Cette directive nous permet de rendre différents éléments en fonction d'une condition donnée, en effet la directive NgSwitch est un ensemble de directives fonctionnant en conjonction.

```
<div class="col" *ngIf="cpt !=0; else alternative2">
  <ul *ngFor="let task of tasks"
    [ngSwitch]="task.charAt(0)">

    <li *ngSwitchCase="'A'"> {{ task }} </li>
    <li *ngSwitchCase="'E'"> {{ task }} </li>
    <li *ngSwitchCase="'I'"> {{task }} </li>

  </ul>
</div>
```



À la différence des directives structurelles, les directives d'attribut modifient le comportement d'un objet déjà existant.

Les directives les plus connues sont :

- **ngModel**
- **ngStyle**
- **ngClass**

## ngStyle

Cette directive permet d'appliquer des styles à un objet du DOM de manière dynamique.

```
<br><span [ngStyle]="{color: getColor()}">
  {{task}}
</span>
```

```
getColor() {
  if(this.task.charAt(0) == 'A' ) {
    return 'green';
  } else if(this.task.charAt(0) == 'B' ) {
    return 'red';
  }
}
```




## ngClass

Cette directive permet d'appliquer des styles à un objet du DOM de manière dynamique.

```
<div [ngClass]="['colorer', 'arrierplan'] "  
  class="encadrer"> {{task}}  
</div>
```

Home.component.c

ss

```
.encadrer{ border: inset 3px  black; }  
.colorer{ color:  blueviolet; }  
.arrierplan{background-color:  salmon; }
```



# Les Pipes

## Pipe

Les pipes prennent des données en input, les transforment, et puis affichent les données modifiées dans le DOM.

Il y a des pipes fournis avec Angular, et vous pouvez également créer vos propres pipes si vous en avez besoin.

## Angular pipes

- **Lowercasepipe**
- **Uppercasepipe**
- **Datepipe**
- **Currencypipe**
- **Jsonpipe**
- **Percentpipe**
- **Decimalpipe**
- **Slicepipe**



A dark gray rectangular bar is positioned on the left side of the image, partially overlapping the text.

Routage

- Tout système de routage permet d'associer une route à un traitement
- Angular est SPA. Pourquoi parle-on de route ??
  - Séparer différentes fonctionnalités du système
  - Maintenir l'état de l'application
  - Ajouter des règles de protection

## Routage Angular

Il s'agit des instructions d'affichage à suivre pour chaque URL, c'est-à-dire quel(s) component(s) il faut afficher à quel(s) endroit(s) pour un URL donné.

Puisque le routing d'une application est fondamentale pour son fonctionnement, on déclare les routes dans

**app.module.ts**

Etapes:

1- Créer un fichier app-routing.module.ts : Importer le service de routing d'angula

**Import { RouterModule, Routes } from '@angular/router';**

( **RouterModule** va permettre de configurer les routes dans votre projet )

( **Routes** va permettre de créer les routes )

2- Ajouter les modules au fichier app.module.ts

3- Ajouter la balise de routage à votre component

# app-routing.module

51

```
1  import { NgModule } from '@angular/core';
2  import { Routes, RouterModule } from '@angular/router';
3  import { HomeComponent } from '../home/home.component';
4  import { AboutComponent } from '../about/about.component';
5
6  const routes: Routes = [
7    {
8      path: '',
9      component: HomeComponent
10   },
11   {
12     path: 'about',
13     component: AboutComponent
14   }
15 ];
16
17 @NgModule({
18   imports: [RouterModule.forRoot(routes)],
19   exports: [RouterModule]
20 })
21 export class AppRoutingModule { }
```



Component réalisant le routage:  
(app-component dans notre exemple)

```
<ul>
  <li><a routerLink="">Home</a></li>
  <li><a routerLink="about">About</a> </li>
</ul>
<!-- <app-home></app-home> -->
<router-outlet></router-outlet>
```



Component  
réalisant une  
navigation (about-  
component dans  
notre exemple)

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

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

export class AboutComponent implements OnInit {
  goals : any;
  private router: Router;

  sendMeHome() {
    this.router.navigate(['']); // path : '' of home component
  }
}
```

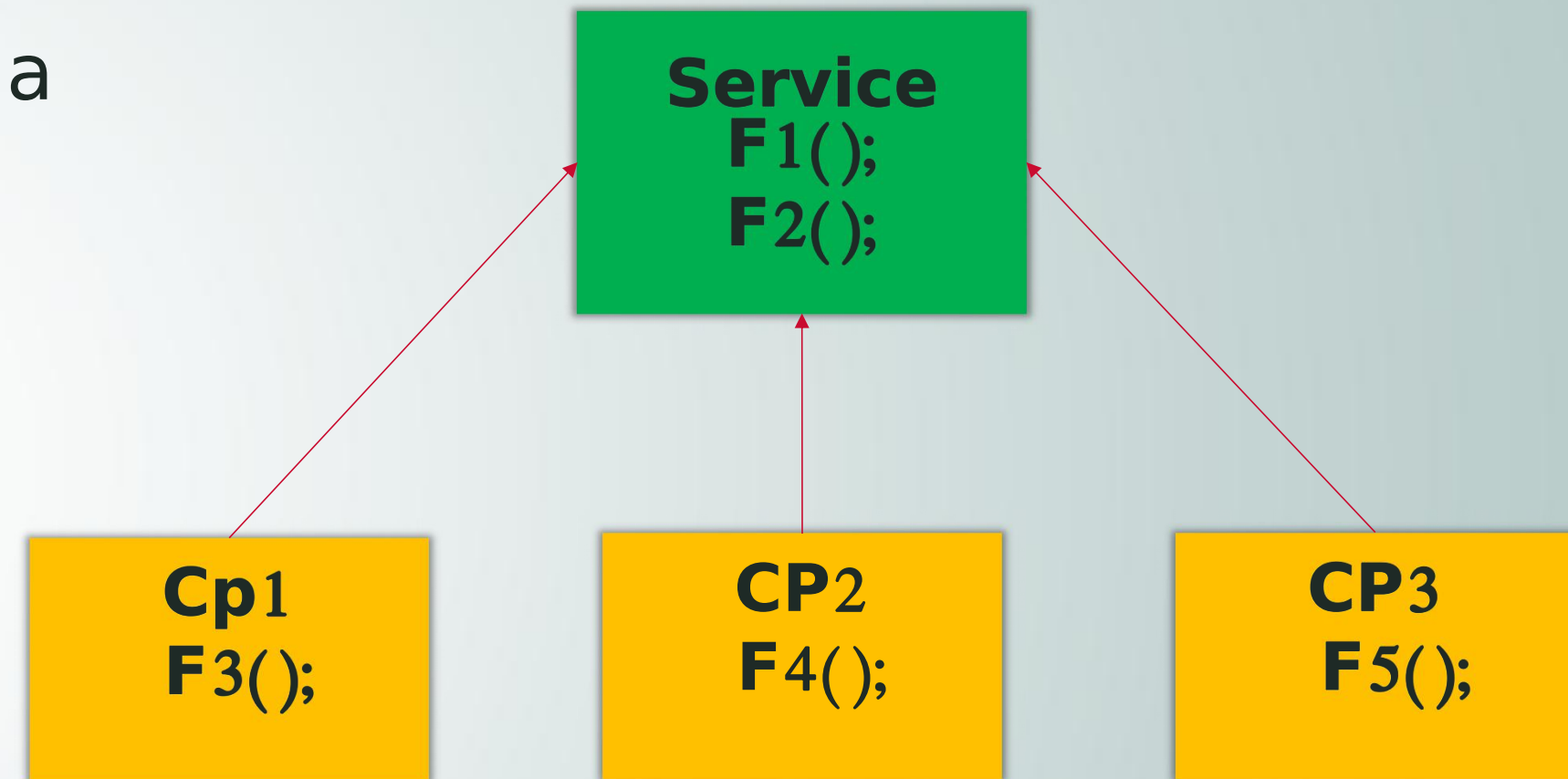


Services

- **Un service** est une **classe** qui permet d'exécuter un traitement
- Un service est un **médiateur** entre la vue et la logique
- Un service est associé à un **composant** en utilisant **l'injection de dépendance**

## Un service peut donc :

- Interagir avec les données (fournit, supprime et modifie)
- Assurer une interaction entre classes et composants
- Faire tout traitement métier (calcul, tri, extraction ...)



# Créer un Service

56

A partir du CLI:

| **ng generate service nomDuService**

Ou bien (forme réduite)

| **ng g s nomDuService**

```
<> app.component.html TS app.module.ts ●
1  import { BrowserModule } from '@angular/platform-browser';
2  import { NgModule } from '@angular/core';
3
4  | import { DataService } from '../data.service';
5
```

```
],
providers: [DataService],
```



# Injection de dépendance

57

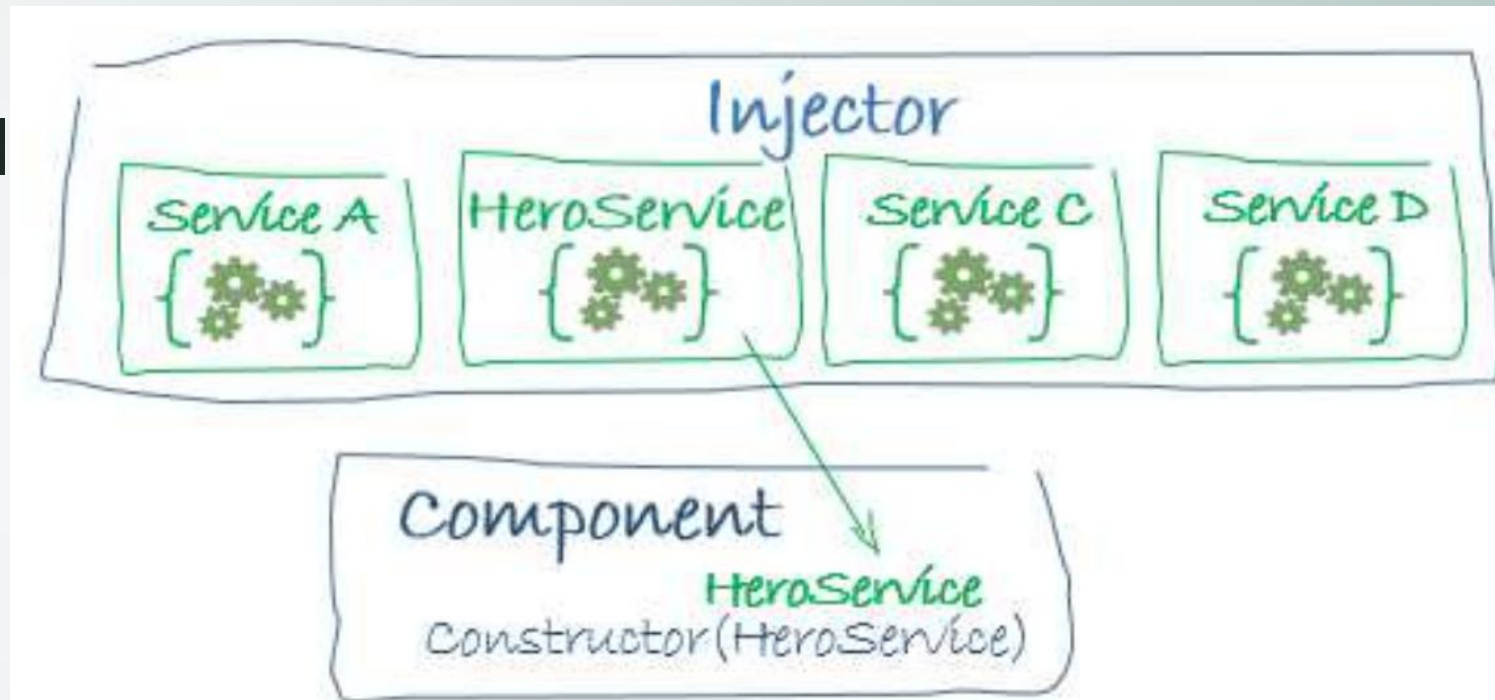
Pour être utilisé dans l'application, un service doit être **injecté**.

**Où ?**

Au niveau du constructeur du composant souhaitant l

**Comment ?**

@Injectable + import via **l'Injector**



```
import { Injectable } from '@angular/core';
import { BehaviorSubject } from 'rxjs/BehaviorSubject';
@Injectable()

export class DataService {
```



Le niveau choisi pour l'injection est très important. 3 niveaux possibles pour cette injection :

- 1- dans **AppModule** : ainsi, la même instance du service sera utilisée par tous les composants de l'application et par les autres services
- 2- dans **AppComponent** : comme ci-dessus, tous les composants auront accès à la même instance du service
- 3- dans un autre service: Le service à injecter doit être visible pour le service cible et le service cible doit obligatoirement avoir la décoration @Injectable.



# HTTP Service

- Le module permettant la consommation d'API externe s'appelle le module HTTP.

Un Http Service nous aidera à aller chercher des données externes, poster, etc. Nous devons importer le module **httpClient** pour utiliser le service http.

- Afin d'utiliser le module HTTP, il faut l'importer de **@angular/common/http**

```
import { HttpClientModule } from  
'@angular/common/http';
```

- Afin d'utiliser le module HTTP, il faut l'injecter dans le composant ou le service dans lequel vous voulez l'utiliser.

```
constructor (private http:HttpClient)
```



Déploiement

## Build

- 1- Ng build
- 2- Ng build --prod
- 3- Ng build --prod --base-href='myURL'

## Github Deploying

- 1- npm i -g angular-cli-ghpages
  - 2- ng build --prod --base-href="<https://YOURUSERNAME.github.io/REPO-NAME/>"
  - 3- git add .  
git commit -m "first commit"  
git remote add origin  
git@github.com:yourinfo/yourgit.git  
git push -u origin master
- (NB que vous devez configurer votre git au préalable  
git config --global user.name "your git username"  
git config --global user.email "email@example.com")