



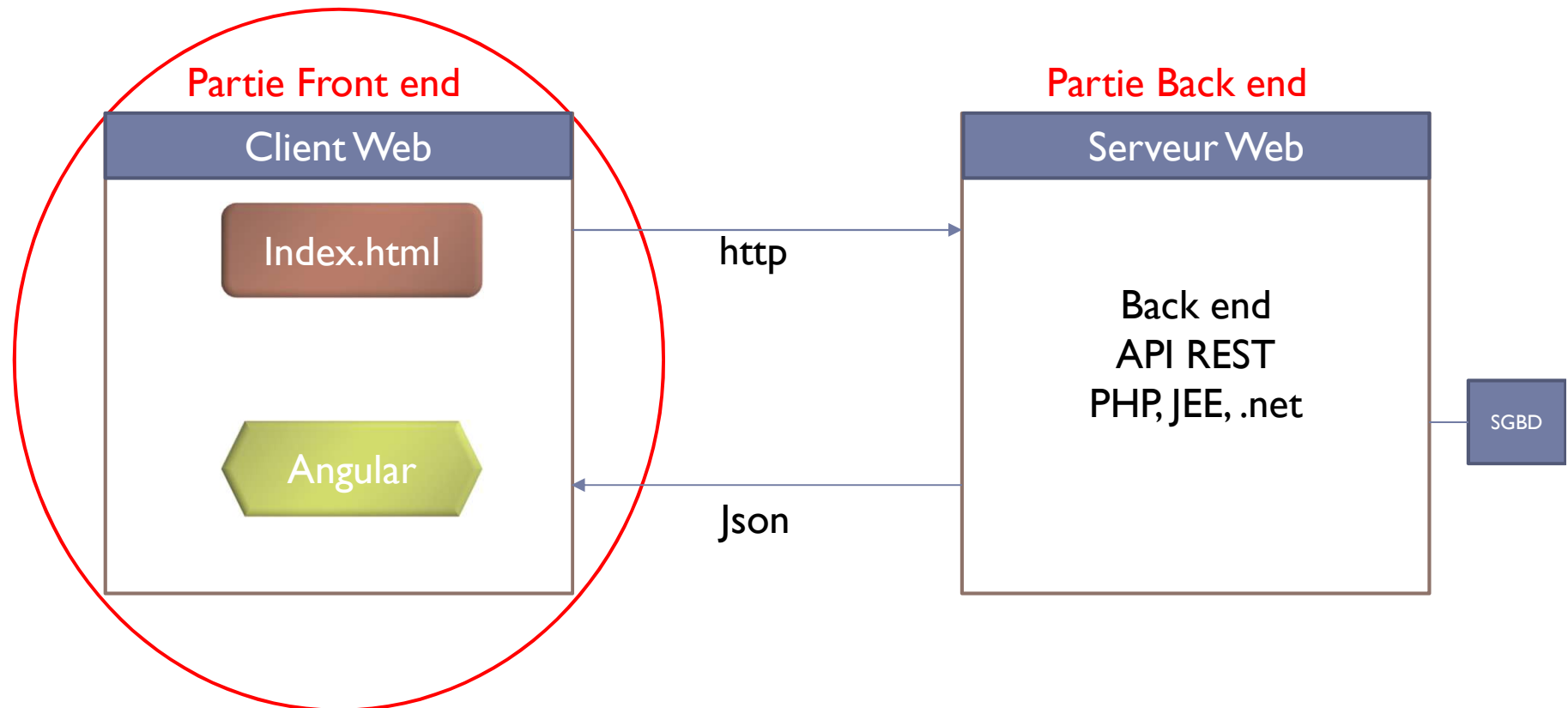
Développement Web

Partie Front-end : Le Framework Angular

Imene LAHYANI

2023/2024

Introduction



Angular

- ▶ Angular permet de créer la partie Front-end des applications Web de type **SPA (Single Page Application reactive)**
- ▶ Une SPA est une application qui contient une seule page HTML (index.html),
- ▶ Pour naviguer entre les différentes parties de l'application, Javascript est utilisé pour envoyer des **requetes http** vers le serveur pour **recuperer du contenu dynamique au format JSON** (generalement)
- ▶ Ce contenu JSON est affiché par la suite coté client au format HTML sur la meme page.

Différentes versions de Angular

▶ Angular 1 (Angular JS) (juin 2012)

Première version de Angular

Elle est basé sur une architecture MVC coté client. Les applications Angular 1 sont ecrites en **Javascript**.

▶ Angular 2 (Angular) (septembre 2016)

Est une réécriture de Angular 1 qui est plus performante, mieux structuré et represente le futur de Angular

Les applications de Angular 2 sont ecrites en **Typescript** qui est compilé et traduit en javascript avant d'être exécutés par les browsers Web,

Angular 2 est basé sur une programmation à base de composants Web (web components)

▶ Angular 4 (Mars 2017)

Est une simple mise à jour de Angular 2

Différentes versions de Angular

- ▶ **Angular 5 : Novembre 2017**
est une simple mise à jour de Angular 4
- ▶ **Angular 6 : Mai 2018**
est une simple mise à jour de Angular 5
- ▶ **Angular 7 : Septembre 2018**
Est une simple mise à jour de Angular 6
- ▶ **Angular 8**
- ▶ **Angular 9**
- ▶ **Angular 10**
- ▶ **Angular 11, 12, 13, 14, 15, 16,**
- ▶ **Angular 17 : 09 Novembre 2023**

Angular avec Typescript

- ▶ Pour développer une application Angular, il est recommandé d'utiliser **Typescript** qui sera compilé et traduit en **Javascript**.
- ▶ Typescript est un langage de script structuré et orienté objet qui permet de simplifier le développement d'applications Javascript



Démarrer avec Angular

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

The screenshot shows the Angular documentation website. The browser address bar displays `https://angular.io/docs`. The website has a blue header with the Angular logo, navigation links for 'DOCS', 'COMMUNITY', and 'BLOG', a search bar, and social media icons. A left sidebar contains a table of contents with links to 'Introduction', 'Getting started', 'Understanding Angular', 'Developer guides', 'Best practices', 'Angular tools', 'Tutorials', 'Updates and releases', 'Reference', 'Documentation contributors guide', and 'Docs Versions'. The main content area is titled 'Introduction to the Angular docs' and includes a paragraph about Angular as a framework, a paragraph about the docs' purpose, and two featured cards: 'Build your first Angular app' (Homes App Tutorial) and 'What is Angular' (Platform overview).

Introduction

Getting started >

Understanding Angular >

Developer guides >

Best practices >

Angular tools >

Tutorials >

Updates and releases >

Reference >

Documentation contributors guide >

Docs Versions ▾

Introduction to the Angular docs

Angular is an application-design framework and development platform for creating efficient and sophisticated single-page apps.

These Angular docs help you learn and use the Angular framework and development platform, from your first application to optimizing complex single-page applications for enterprises. Tutorials and guides include downloadable examples to help you start your projects.

Build your first Angular app

Work through a full tutorial to create your first application.

[Homes App Tutorial](#)

What is Angular

Get a high-level overview of the Angular platform.

[Platform overview](#)

Installation des outils nécessaires

- ▶ Pour faciliter le développement d'une application angular, les outils suivants doivent être installés:
 - **Node JS** : installe l'outil npm (Node package manager) qui permet de télécharger et installer des bibliothèques JavaScript
<https://nodejs.org/en/download/>
 - Installer Angular CLI (Command Line Interface) qui permet de générer , compiler, tester, deployer des projets angular
<http://cli.angular.io> en utilisant :

npm install -g @angular/cli

Création d'un nouveau projet Angular

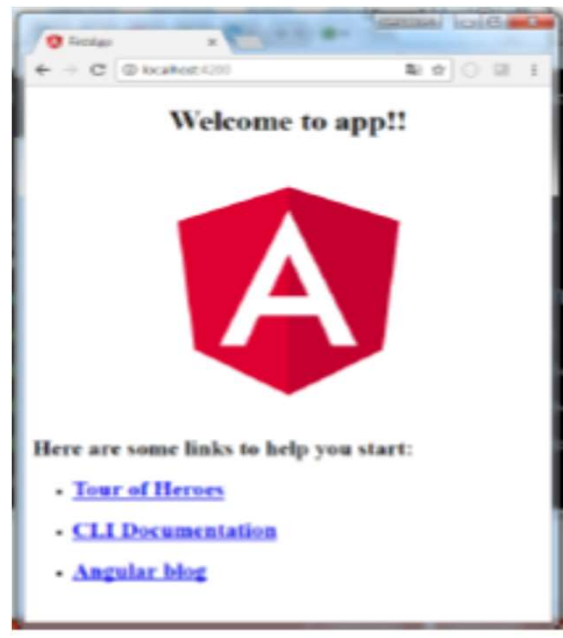
- ▶ Afin de générer la structure d'un projet Angular, on utilise Angular CLI via la commande **ng** suivi de l'option new et du nom du dossier
 - ▶ **ng new FirstApp**
 - ▶ **ng:** genere les differents fichiers requis par une application basique Angular et installe toutes les dependances requises par ce projet.

Exécuter un projet Angular

- ▶ Afin d'exécuter le projet Angular, on exécute la commande
- ▶ `ng serve`
- ▶ Cette commande :
 - ▶ compile le code source du projet pour transpiler le code TypeScript en Javascript
 - ▶ En meme temps , démarre un serveur web local basé sur node Js pour déployer l'application localement (port par défaut du serveur : 4200)

Tester un projet Angular

- ▶ Afin de tester un projet Angular, il suffit d'ouvrir votre navigateur et taper : <http://localhost:4200>

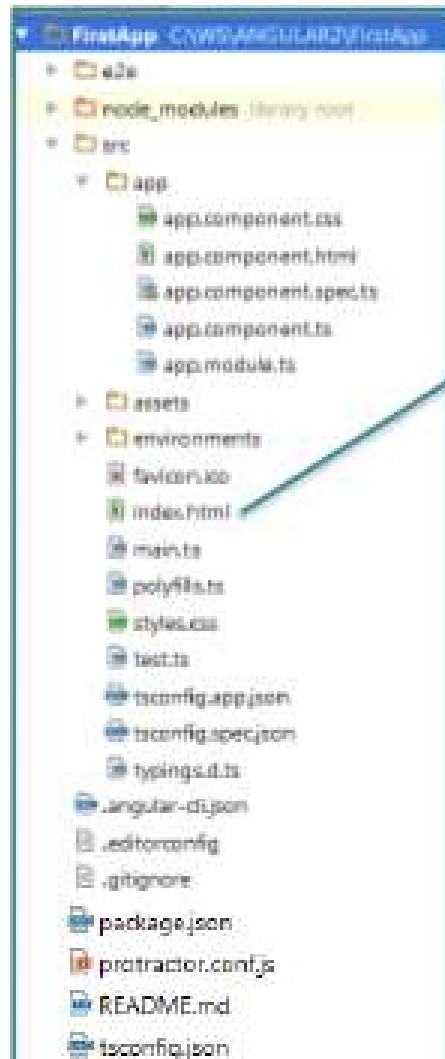


Editeur de projets

- ▶ Plusieurs editeurs professionnels peuvent etre utilisés pour editer le code:
 - ▶ Web Storm, PHP Storm,
 - ▶ Visual Studio Code
 - ▶ Eclipse avec plugin Angular
- ▶ D'autres editeurs classiques peuvent etre utilisés :
 - ▶ Atom
 - ▶ Sublime Text

Installer votre éditeur de projets

Structure du projet Angular



index.html

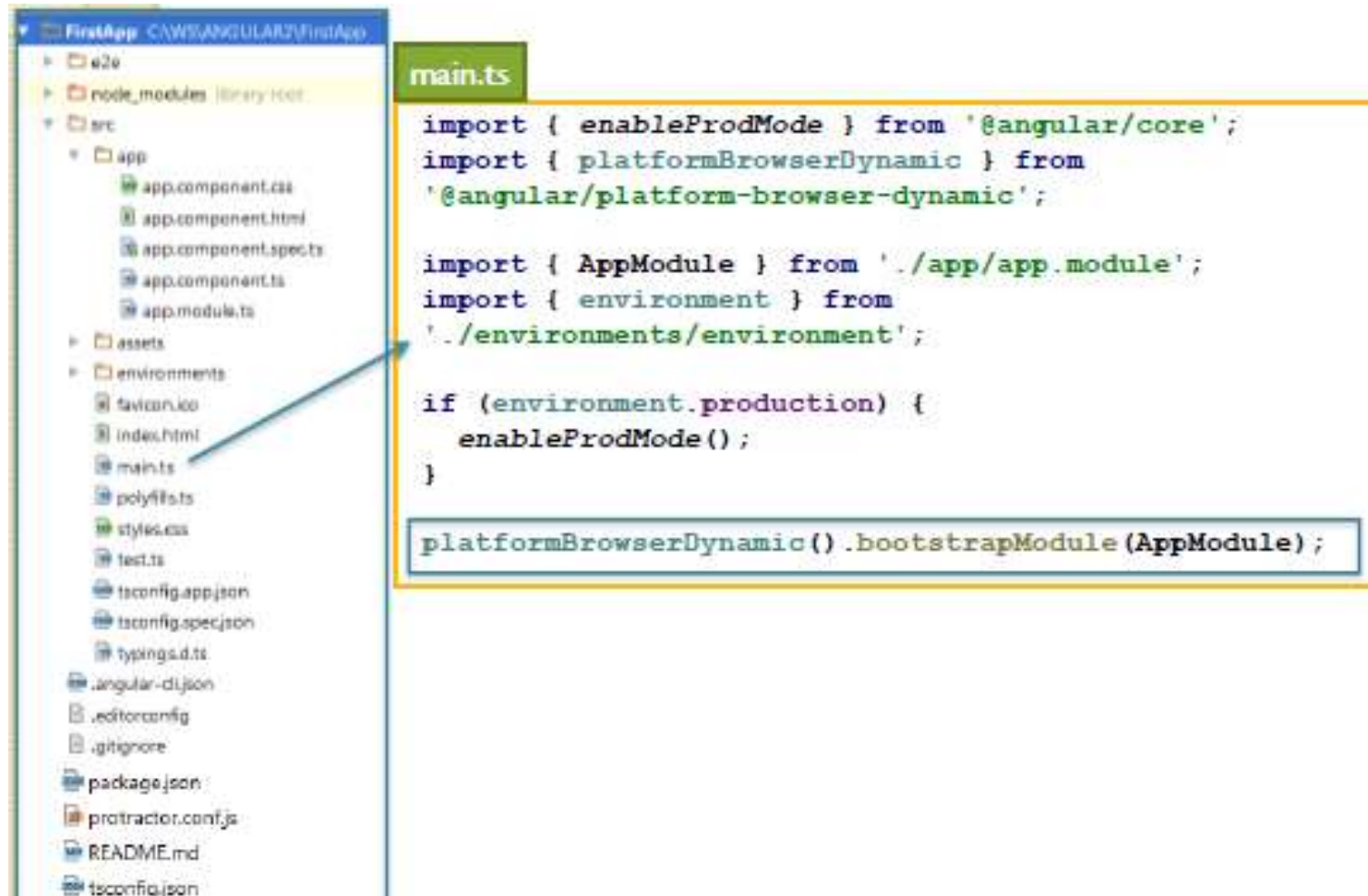
```
<!doctype html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <title>FirstApp</title>
  <base href="/">

  <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>
```

Structure du projet Angular



The image shows a screenshot of an Angular project structure on the left and the content of the `main.ts` file on the right. A blue arrow points from the `main.ts` file in the file explorer to the code editor.

File Explorer Structure:

- FirstApp C:\WS\ANGULAR\FirstApp
 - e2e
 - node_modules library root
 - src
 - app
 - app.component.css
 - app.component.html
 - app.component.spec.ts
 - app.component.ts
 - app.module.ts
 - assets
 - environments
 - favicon.ico
 - index.html
 - main.ts
 - polyfills.ts
 - styles.css
 - test.ts
 - tsconfig.app.json
 - tsconfig.spec.json
 - typings.d.ts
 - .angular-cli.json
 - .editorconfig
 - .gitignore
 - package.json
 - protractor.conf.js
 - README.md
 - tsconfig.json

main.ts Code:

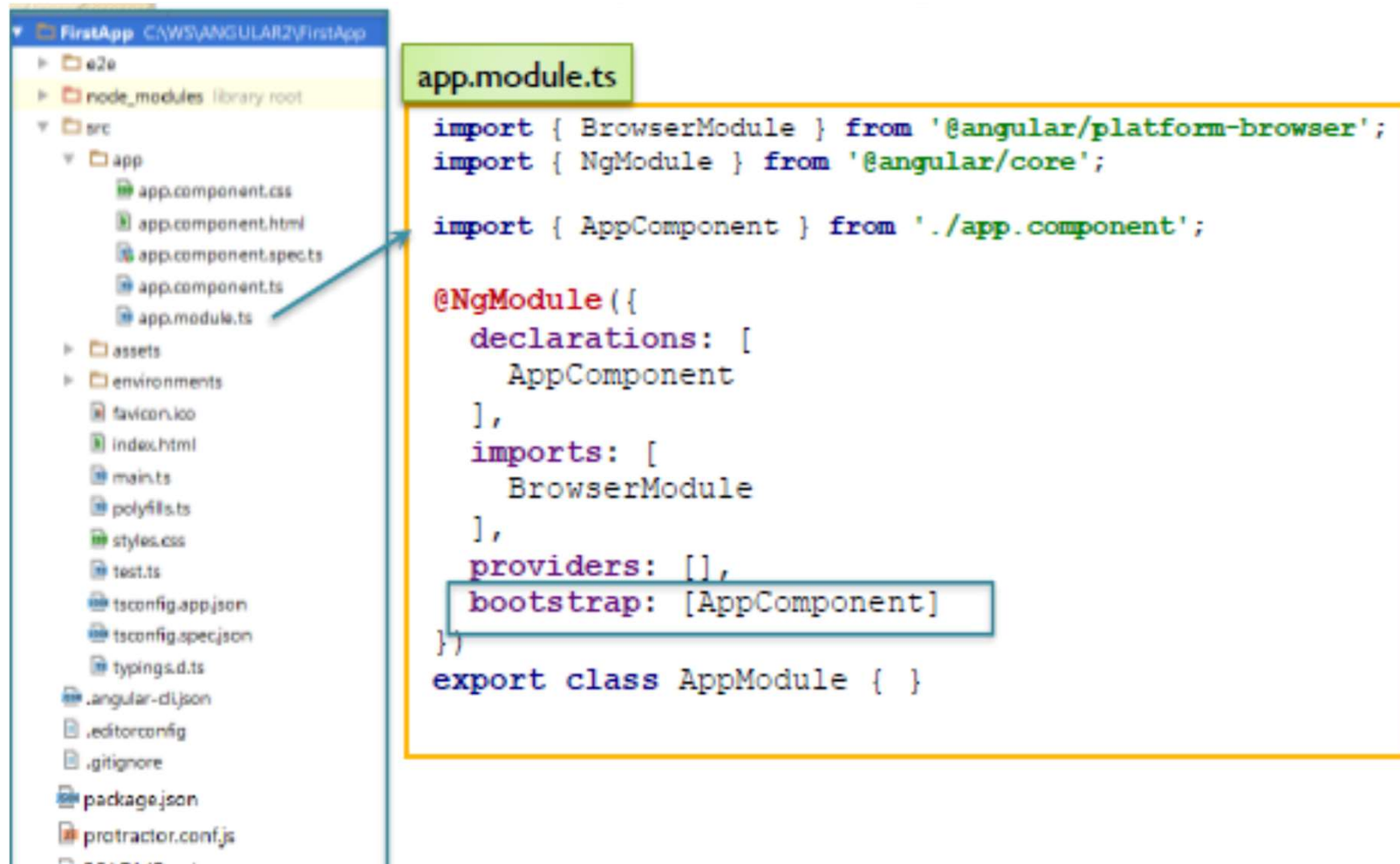
```
import { enableProdMode } from '@angular/core';
import { platformBrowserDynamic } from
  '@angular/platform-browser-dynamic';

import { AppModule } from './app/app.module';
import { environment } from
  './environments/environment';

if (environment.production) {
  enableProdMode();
}

platformBrowserDynamic().bootstrapModule(AppModule);
```

Structure du projet Angular



The image shows a file explorer on the left and a code editor on the right. The file explorer displays the project structure for 'FirstApp' at 'C:\WS\ANGULAR2\FirstApp'. The 'src' directory contains an 'app' subdirectory with files 'app.component.css', 'app.component.html', 'app.component.spec.ts', 'app.component.ts', and 'app.module.ts'. An arrow points from 'app.module.ts' in the file explorer to the code editor. The code editor shows the content of 'app.module.ts' with a green header label 'app.module.ts' and an orange border. The code imports 'BrowserModule' and 'NgModule' from '@angular/platform-browser' and '@angular/core', and 'AppComponent' from './app.component'. It then defines an '@NgModule' with 'declarations' for 'AppComponent', 'imports' for 'BrowserModule', 'providers' as an empty array, and 'bootstrap' as '[AppComponent]'. Finally, it exports the 'AppModule' class. A blue box highlights the 'bootstrap' property in the 'providers' array.

```
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 { }
```

Structure du projet Angular

The diagram illustrates the structure of an Angular project. On the left, a file explorer shows the project hierarchy. The main part of the diagram consists of three yellow boxes containing code snippets for `app.component.ts`, `app.component.html`, and `app.component.css`. Arrows point from the file explorer to these boxes. A small browser window on the right shows the rendered output: "Welcome to app!!".

File Explorer:

- FirstApp C:\WS\ANGULAR2\FirstApp
 - e2e
 - node_modules library root
 - src
 - app
 - app.component.css
 - app.component.html
 - app.component.spec.ts
 - app.component.ts
 - app.module.ts
 - assets
 - environments
 - favicon.ico
 - index.html
 - main.ts
 - polyfills.ts
 - styles.css
 - test.ts
 - tsconfig.app.json
 - tsconfig.spec.json
 - typings.d.ts
 - .angular-cli.json
 - .editorconfig
 - .gitignore
 - package.json
 - protractor.conf.js
 - README.md
 - tsconfig.json

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

Browser Preview:

Welcome to app!!

Architecture de Angular

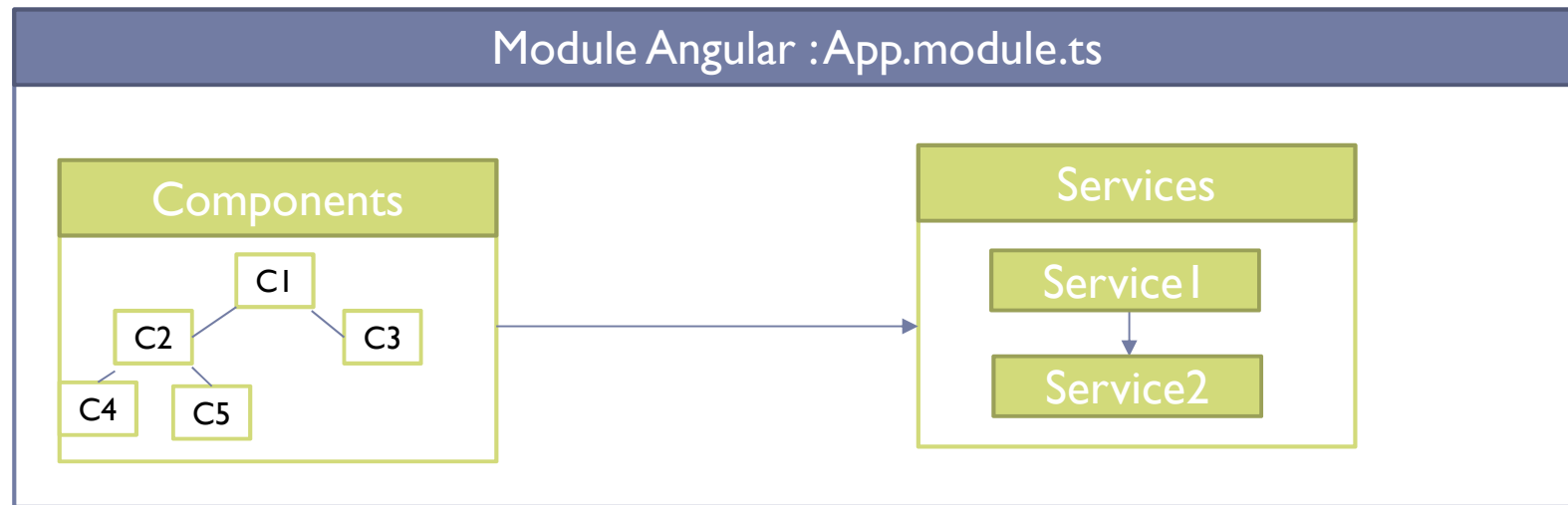
- ▶ Une application Angular se compose de :

- ▶ Un ou plusieurs modules dont un est principal

Chaque module peut inclure:

- Des composants Web: la partie visible de l'application IHM
 - Des services pour la logique applicative

Architecture de Angular



Modules

- ▶ Les applications Angular sont modulaires
- ▶ Angular possède son propre système de modularité appelés modules angulaires ou NgModules
- ▶ Chaque application possède au moins une classe de module racine appelé classiquement **AppModule**
- ▶ Un module angulaire est une classe avec un décorateur **@NgModule**
- ▶ Les décorateurs sont des classes qui modifient les classes Javascript

Modules

Src/app/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 { }
```

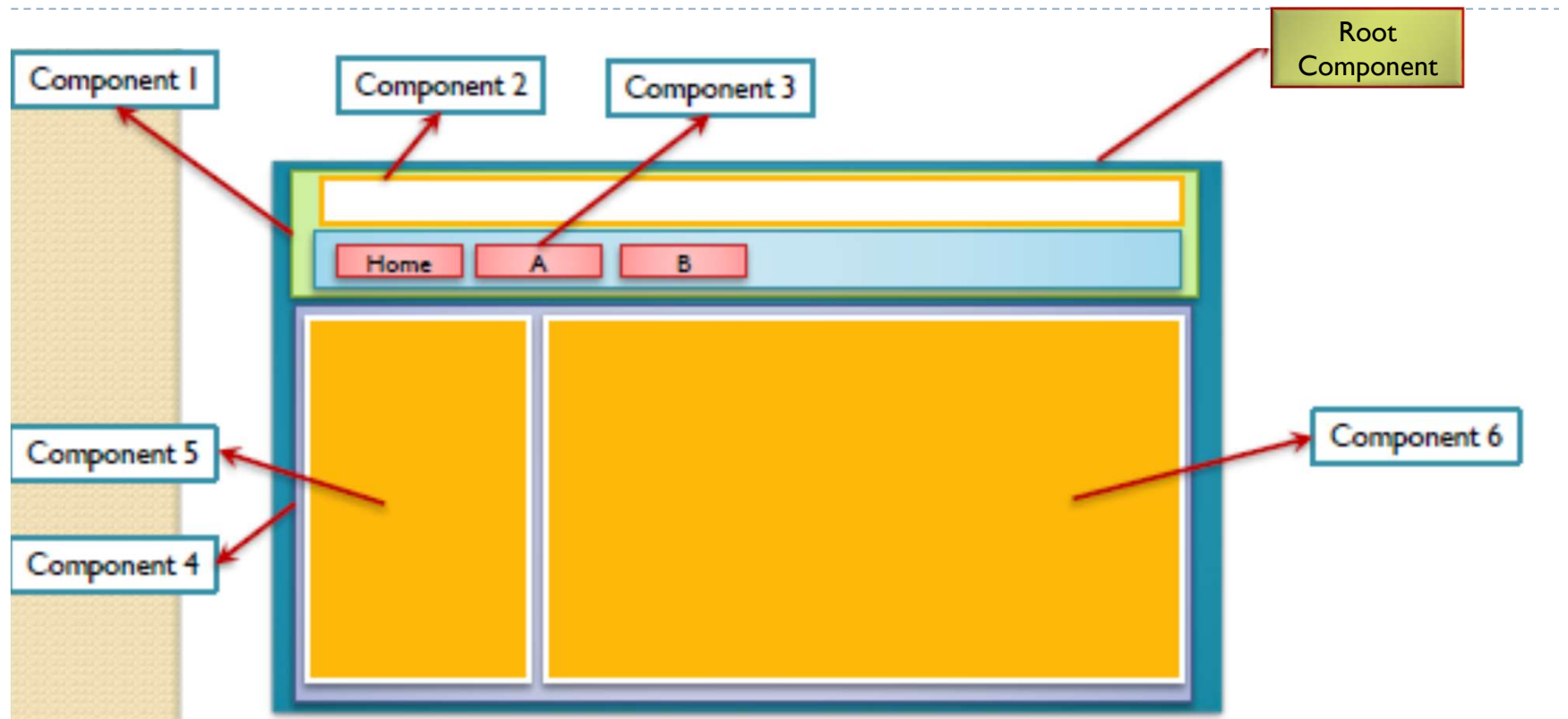
@NgModule

- ▶ @NgModule est un décorateur qui prend en parametre un objet Javascript dont les proprietes decrivent le module.
- ▶ Les propriétés les plus importantes sont :
 - ▶ **imports**: importer d'autres modules
 - ▶ **exportes**: importer des classes utilisable dans d'autres modules
 - ▶ **Declarations**: declarer les composants qui vont etre utilisés dans ce module
 - ▶ **Providers** : declarer les services
 - ▶ **Bootstrap** : declarer le composant racine du module
(seul le composant racine doit définir cette propriété)

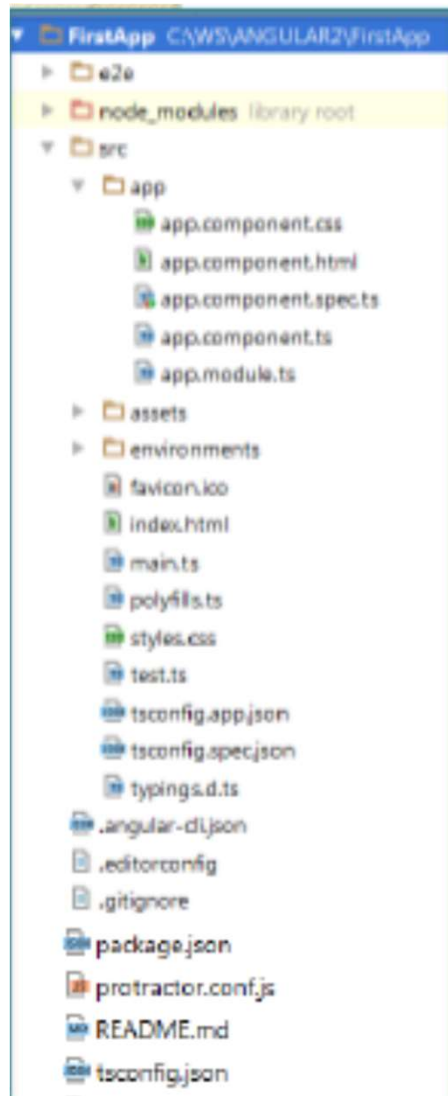
Les composants

- ▶ Les composants sont des elements importants dans Angular
- ▶ L'application est formé par un ensemble de composants
- ▶ Chaque composant peut imbriquer d'autres composants définissant ainsi une structure hierarchique
- ▶ Le composant racine s'appelle Root component

Les composants



Les composants



- Chaque composant se compose principalement des éléments suivants:
 - HTML Template : représentant sa vue
 - Une classe représentant sa logique métier
 - Une feuille de style CSS
- Les composants sont facile à mettre à jour et à échanger entre les différentes parties des applications.

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

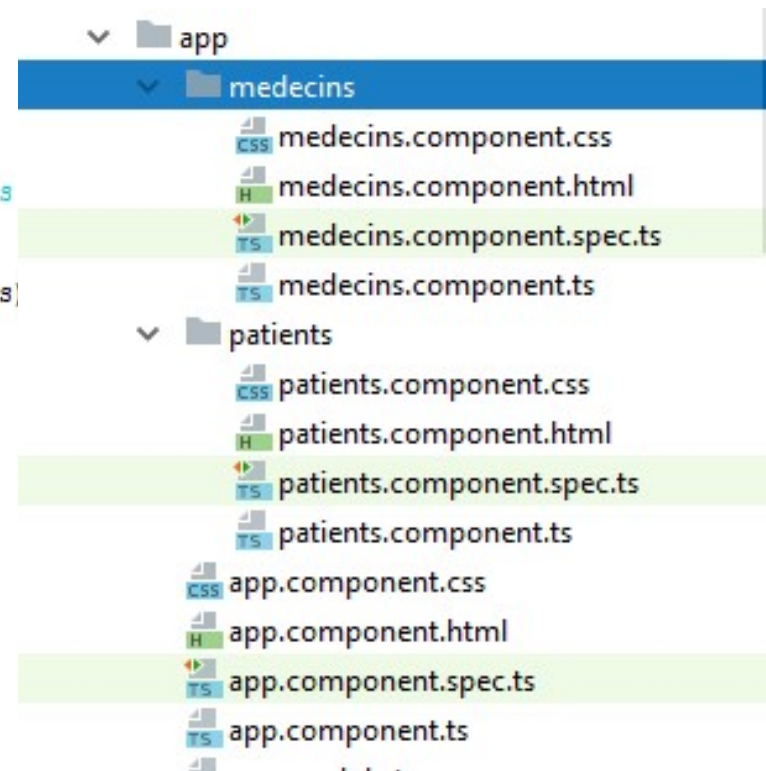
```
<div style="text-align:center">
  <h1>
    Welcome to {{{title}}}!!
  </h1>
</div>
```

app.component.css

Création de nouveaux composants

- ▶ Pour créer facilement des composants angular, on peut utiliser la commande **ng** comme suit :
- ▶ **ng generate component** nom_composant

```
C:\Angular3\FirstA>ng g c
? What name would you like to use for the component? patients
CREATE src/app/patients/patients.component.html (27 bytes)
CREATE src/app/patients/patients.component.spec.ts (642 bytes)
CREATE src/app/patients/patients.component.ts (277 bytes)
CREATE src/app/patients/patients.component.css (0 bytes)
UPDATE src/app/app.module.ts (495 bytes)
```



Création de nouveaux composants

medecins.component.ts

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

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

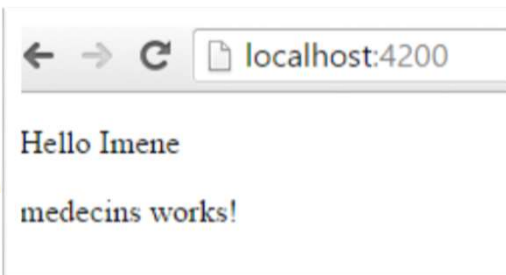
  constructor() { }

  ngOnInit() {
  }

}
```

medecins.component.html

```
<p>
  medecins works!
</p>
```



Déclaration d'un nouveau composant

- Pour utiliser un composant, ce dernier doit être déclaré dans le module principal `app.module.ts` (automatique avec ng g c)

App.module.ts

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

import { AppComponent } from '../app.component';
import { MedecinsComponent } from '../medecins/medecins.component';
import { PatientsComponent } from '../patients/patients.component';

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

Hello Imene

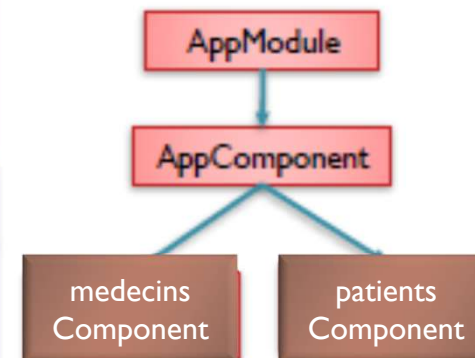
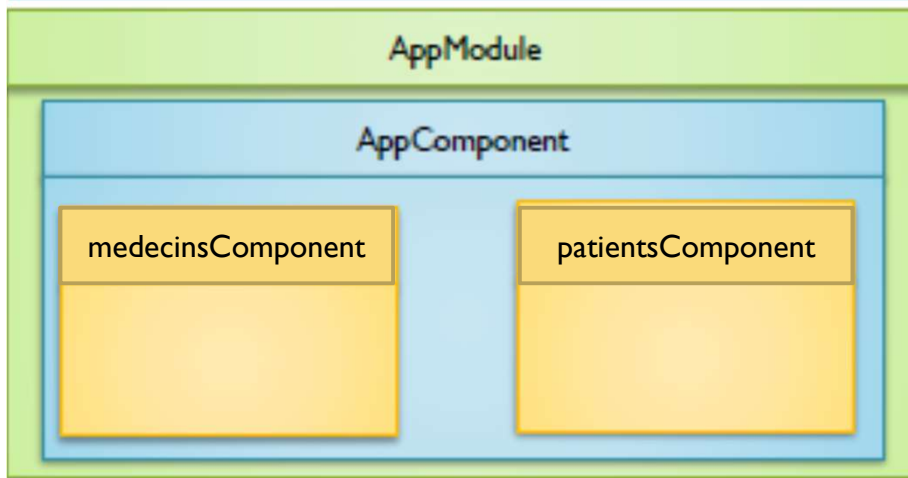
medecins works!

patients works!

Utilisation d'un nouveau composant

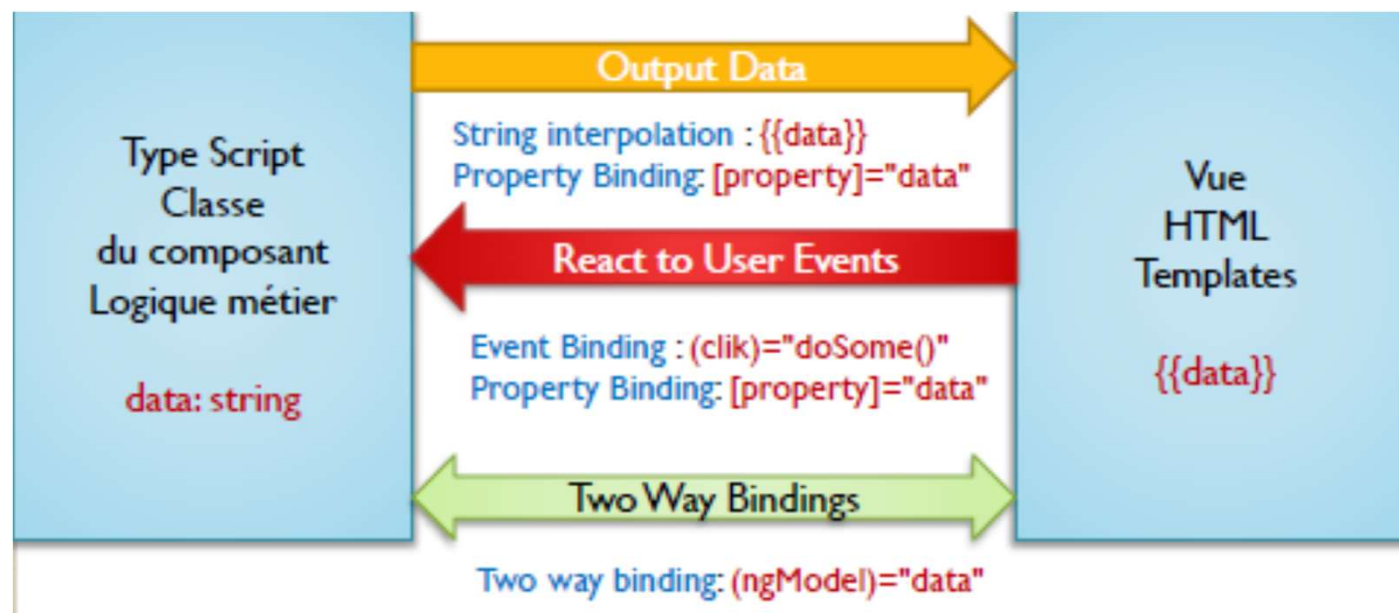
- Un composant peut être inséré dans n'importe quelle partie HTML de l'application en utilisant son selector associé

```
<p> Hello Imene </p>  
<div><app-medecins> </app-medecins></div>  
<div><app-patients></app-patients></div>
```



Data Binding

- ▶ Pour insérer dynamiquement les données de l'application dans les vues du composant, Angular définit des techniques pour assurer la liaison des données.
- ▶ Data Binding = communication

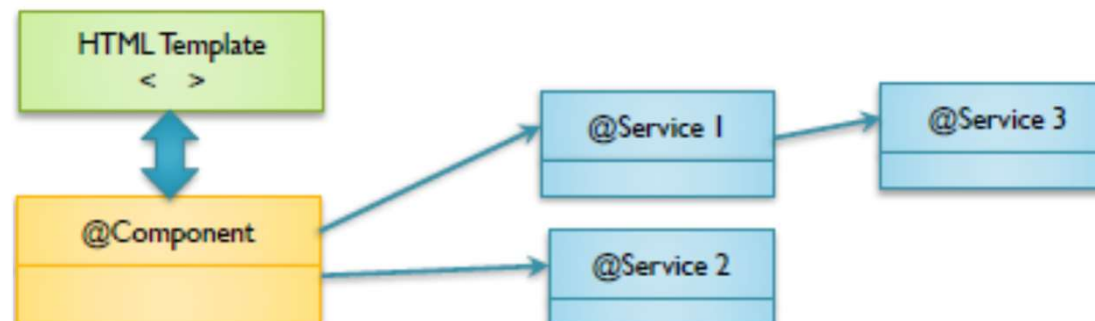


Les services

- ▶ Un **service** est une catégorie large qui englobe toute valeur, fonction ou fonctionnalité dont votre application a besoin
- ▶ Généralement, les composants se limitent à l'affichage et à la gestion des événements utilisateur dans la vue du composant.
- ▶ L'exécution des traitements en local ou en back end sont attribués aux **services**.
- ▶ Quant un événement survient dans la vue, **le composant fait appel à des fonctions dans les services pour effectuer des traitements et fournir des résultats**

Les services

- ▶ Ce sont les services qui interagissent avec la partie back end de l'application en envoyant des requetes http
- ▶ Ce sont les composants qui consomment les services , toutefois un service peut consommer d'autres services
- ▶ L'utilisation d'un service se fait via le principe de l'injection des dépendances



Les services

- ▶ Pour créer un service, il faut :
 - ▶ - créer un medecinService.ts qui contient une classe définissant le service;
 - ▶ Si le service va être utilisé dans d'autres composants, utiliser `@injectable`

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

@Injectable()
export class MedecinService
{
}
```
 - ▶ Pour utiliser le service dans le composant, il faut utiliser le principe d'injection des dépendances,
 - ▶ **Comment utiliser un service ? Comment injecter un service ?**

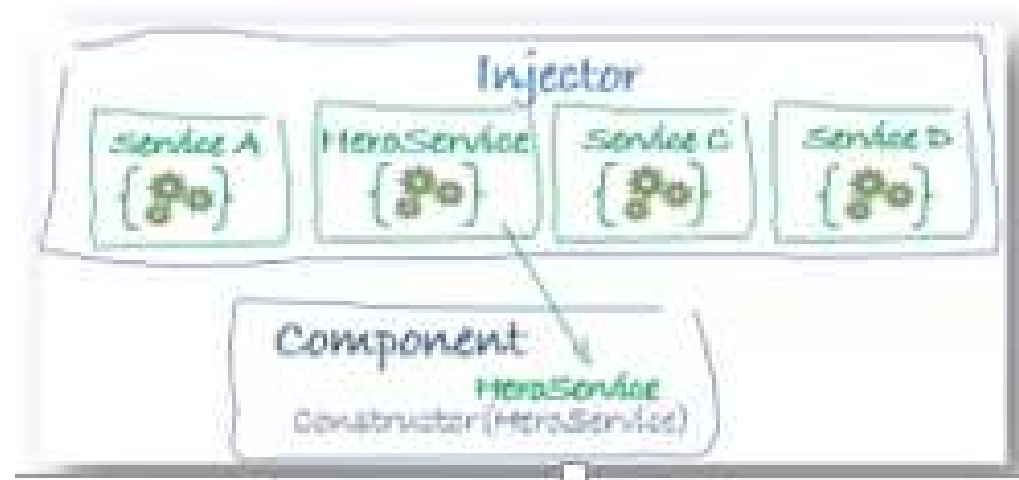
Injection des dépendances

- ▶ Lorsque Angular crée un composant, il demande d'abord à un injecteur les services requis
- ▶ Un injecteur maintient un conteneur d'instances de service qu'il a créé précédemment,
- ▶ Si le service est appelé pour la première fois et que le conteneur ne contient pas cette instance , il fait une, et l'ajoute avant de renvoyer le service à Angular,

Lorsque tous les services demandés ont été résolus et retournés, Angular peut appeler le constructeur du composant avec ces services comme arguments,

C'est ce qu'on appelle **l'injection des dépendances**

Injection des dépendances



Communication avec la partie Back end :

HTTP Client

- ▶ La plupart des applications frontales doivent communiquer avec un serveur via le protocole HTTP, pour télécharger des données et accéder à d'autres services back-end.
- ▶ Angular fournit une **API HTTP client** pour les applications Angular : la classe de service `HttpClient` dans `angular/common/http`.
- ▶ HTTP Client offre les méthodes suivantes :GET , PUT, POST, et DELETE,

Les Threads Observable

- ▶ Le modèle d'observateur est un **modèle de conception logicielle** dans lequel un objet, appelé **sujet**, maintient une liste de ses dépendants, appelés **observateurs**, et les **informe automatiquement des changements d'état**.

Les Threads Observable

- ▶ La responsabilité principale des Observables se trouve généralement **dans un service**.
- ▶ **Les composants vont ensuite s'y abonner pour recevoir les données.**

- ▶ **1/Création d'un Observable**

```
const monObservable = new Observable(...
```

- ▶ **2/Observer : Qui écoute l'Observable ?**

- ▶ L'observer est la personne (ou la fonction) qui écoute l'Observable.

```
const observable = new Observable(observer => { observer.next(5)});
```

- ▶ **3/S'abonner à un Observable**

```
observable.subscribe(value => { console.log(value)});
```

Les Threads Observable

- ▶ Les observables ne sont pas exécutés tant qu'un consommateur ne s'est pas abonné
- ▶ Le **subscribe()** initie le comportement de l'observable qui peut s'exécuter **de manière synchrone ou asynchrone** et peut produire une, plusieurs ou aucune valeur au fil du temps.

Les Threads Promise

- ▶ Les promesses s'exécutent immédiatement lorsqu'elles sont créées.
- ▶ Une promise est toujours **asynchrone** et peut produire au plus une valeur
- ▶ **// syntaxe**
- ▶ `let promise = new Promise(resolve => {resolve(123)});`
- ▶ `promise.then(value => {console.log(value); });`