I-Introduction

Aperçu et histoire du framework:

Angular est un framework

- Il est géré par Google.
- Il utilise le TypeScript — *ce langage permet un développement beaucoup plus stable, rapide et facile.*

En 2010, la première version d'AngularJS est lancée.
Elle permet de créer plus facilement des Single Page Applications, des applications web qui imitent les applications natives : pas de rafraîchissement du navigateur, temps de chargement réduits, une UI beaucoup moins "internet" etc.

Cette version permet déjà de faire énormément de choses, mais souffre d'une syntaxe plutôt complexe ainsi que des limitations du JavaScript.

Google a donc choisi de complètement réécrire le framework pour sa version 2.
Cette version 2 évolue environ tous les 6 mois (ex: ajouts de nouvelles fonctionnalités)
Dans ce cours nous parlerons de la version Angular 5

Quelle est la différence entre une librairie et un framework ?

Définition d'une librairie

Une librairie est juste une collection de définitions de classe.
En l'utilisant on effectue une réutilisation de code, c'est-à-dire qu'on récupère le code qui a déjà été écrit par d'autres développeurs.
Les classes et les méthodes définissent normalement des opérations spécifiques dans une zone spécifique à un domaine. Par exemple, certaines librairies de mathématiques permet au développeur d'appeler une fonction sans refaire la mise en oeuvre du fonctionnement d'un algorithme.

Définition d'un framework

Dans le framework, tout le flux de contrôle est déjà là, et il y a un tas de choses prédéfinis que vous devrez remplir avec votre code.
Un framework est normalement plus complexe. Il définit un squelette où l'application définit ses propres fonctionnalités pour remplir le squelette. De cette façon, votre code sera appelé par le framework si cela convient.
L'avantage est que les développeurs n'ont pas à s'inquiéter de savoir si une conception est bonne ou non, mais simplement à implémenter des fonctions spécifiques à un domaine.

En résumé:

Librairie vous fournit un ensemble de fonctions/modules/API que vous pouvez utiliser pour résoudre un problème donné, mais cela ne change pas votre code au niveau structurel ou architectural.

Tandis que, les frameworks vous donnent également un ensemble de fonctions/modules/API mais cela change votre code au niveau structurel ou architectural.

Librairie – vous l'appelez, Framework – elle vous appelle.

INSTALLATION

Installez les outils

Vous devez installer les outils suivants si vous ne les avez pas déjà sur votre machine :

NODE.JS
> Téléchargez et installez la dernière version LTS de Node.js ici : https://nodejs.org/en/download/

NPM
> NPM est un package manager qui permet l'installation d'énormément d'outils et de libraries dont vous aurez besoin pour tout type de développement.
Pour l'installer, ouvrez une ligne de commande et tapez la commande suivante :

npm install -g npm@latest

ANGULAR/CLI
> Vous allez maintenant installer le CLI d'Angular de manière globale sur votre machine avec la commande suivante (avec sudo si besoin) :

npm install -g @angular/cli

A RETENIR:

Voici 3 commandes de terminal à connaître afin de circuler dans ses fichiers depuis la console:

ls : Visualiser les fichiers dans le dossier où vous vous trouvez
cd nom_du_dossier : Entrer dans un dossier
cd .. : Remonter d'un dossier

Créer un premier projet

Pour créer un nouveau projet Angular, naviguez vers le dossier souhaité depuis une ligne de commande et saisissez la commande suivante :
ng new mon-projet-angular

Ensuite, naviguez dans le dossier du projet et lancez le serveur de développement :
cd mon-projet-angular

Avant de plonger dans les différents dossiers, vous allez exécuter une commande pour installer Bootstrap dans votre projet. Depuis le dossier mon-projet-angular, avec une ligne de commande, tapez :
npm install bootstrap

Cette commande téléchargera Bootstrap et l'intégrera dans le package.json du projet.
Il vous reste une dernière étape pour l'intégrer à votre projet.

Ouvrez le fichier angular.json du dossier source de votre projet.
Dans "architect/build/options/styles", modifiez l'array styles comme suit :

```
"styles": [
   "node_modules/bootstrap/dist/css/bootstrap.css",
   "src/styles.css"
]
```

Maintenant vous pouvez lancer le serveur de développement local:
ng serve ou ng serve --open

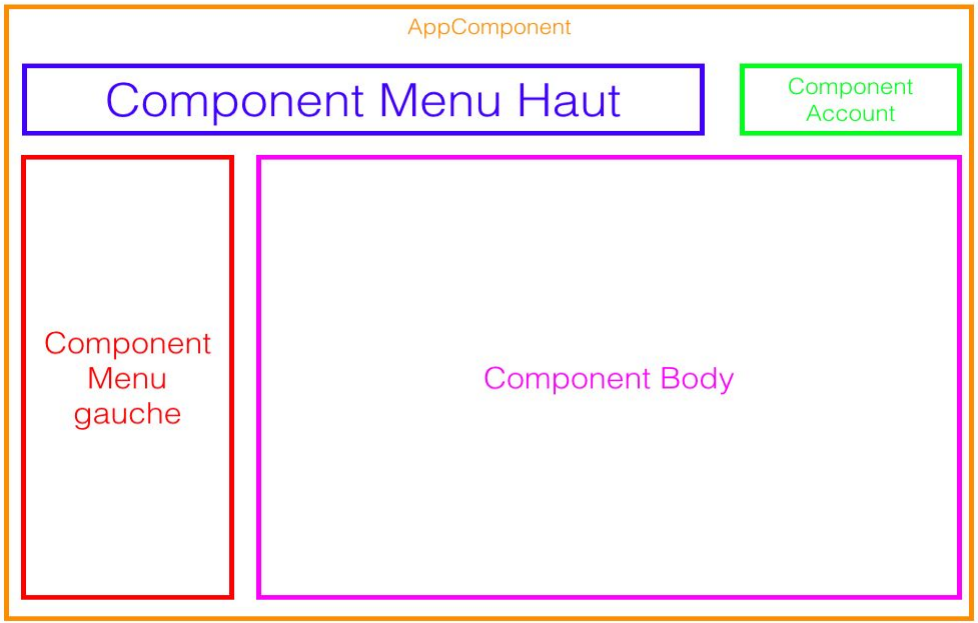
Si tout se passe bien, vous verrez les informations du serveur qui se lance à l'adresse localhost:4200 et votre navigateur se lancera automatiquement en ouvrant une page "Welcome to app!!" et le logo Angular.

Félicitations, votre environnement de développement est prêt !

Structure du code

I - Les composants

Les components sont les composantes de base d'une application Angular : une application est une arborescence de plusieurs components.
Exemple : **`AppComponent`** est notre composant principal : tous les autres composants de notre application seront emboîtés ou "nested" dans celui-ci.



II - Les dossiers et fichiers générés d'Angular

Le dossier `e2e` est généré pour les tests end-to-end

Le dossier `node modules` contient toutes les dépendances pour votre application : les fichiers source Angular et TypeScript, par exemple

Le dossier `src` (celui qui nous intéresse) contient tous les fichiers sources pour votre application.

https://www.youtube.com/watch?v=z_dqh_OEkjA&list=PLuWyq_EO5_ALVh9pDGwt7sq6NeXqyaaKv&index=5

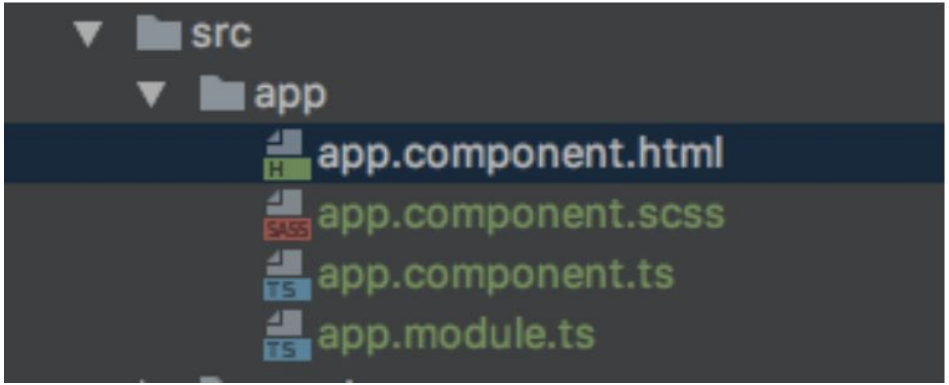
index.html

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

On constate ici qu'il n'y a pas tout le contenu que nous voyons dans le navigateur
Il y a seulement une balise vide `<app-root>`
Il s'agit d'une balise Angular. Afin de voir ce qu'elle contient, ouvrez le dossier `app` :

Le dossier app



Ce dossier contient le module principal de l'application et les trois fichiers du composant principal `AppComponent` :

- son template en HTML
- sa feuille de styles en SCSS
- son fichier TypeScript, qui contiendra sa logique.

app.component.html : correspond au code HTML que vous voyez dans votre navigateur.

Ce code est donc injecter dans la balise <app-root>

Pour réussir cette exploit, Angular utilise le fichier app.component.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.scss']
7  })
8  export class AppComponent {
9    title = 'app';
10 }
```

@Component est une fonction de décorateur qui spécifie les métadonnées angular du composant.

Ici, à l'intérieur du décorateur **`@Component()`** , vous trouvez un objet qui contient les éléments suivants :

- **`selector`** : il s'agit du nom qu'on utilisera comme balise HTML pour afficher ce component, comme vous l'avez vu avec `<app-root>` . Ce nom doit être unique et ne doit pas être un nom réservé HTML de type `<div>` , `<body>` etc. On utilisera donc très souvent un préfixe comme `app` , par exemple ;
- **`templateUrl`** : le chemin vers le code HTML à injecter ;
- **`styleUrls`** : un array contenant un ou plusieurs chemins vers les feuilles de styles qui concernent ce component ;

Créer un component

On utilisera la commande suivante dans notre terminal:

```
ng generate component mon-premier-component
```

Si tout se passe bien, vous devriez avoir ce résultat

```
CREATE src/app/mon-premier-component/mon-premier-component.component.css (0 bytes)
CREATE src/app/mon-premier-component/mon-premier-component.component.html (36 bytes)
CREATE src/app/mon-premier-component/mon-premier-component.component.spec.ts (721 bytes)
CREATE src/app/mon-premier-component/mon-premier-component.component.ts (327 bytes)
UPDATE src/app/app.module.ts (631 bytes)
```

Remarque: Cette commande nous a crée plusieurs fichiers. Parmis eux on constate la présence du fichier spec : il s'agit d'un fichier de test que vous pouvez supprimer, car vous ne vous en servirez pas dans le cadre de ce cours.

src > app > heroes > TS heroes.component.ts > ...

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

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

  constructor() { }

  ngOnInit(): void {
  }

}
```

Le fonction `ngOnInit()` est un crochet de cycle de vie.

`ngOnInit()` est appelé peu de temps après la création d'un composant. C'est un bon endroit pour mettre la logique d'initialisation.

On utilise `export` pour exporter la classe de composant pour pouvoir l'importer avec `import` ailleurs ... comme dans `AppModule`.

En créant notre component, Le fichier `app.module.ts` a été mis à jour.

`MonPremierComponent` a été ajouté à l'array `declarations` de votre module. Il a également ajouté le statement import en haut du fichier. Ce sont des étapes **nécessaires** pour que vous puissiez utiliser votre component au sein de votre application Angular.

app.module.ts

src > app > app.module.ts > AppModule

```
1  import { BrowserModule } from '@angular/platform-browser';
2  import { NgModule } from '@angular/core';
3
4  import { AppRoutingModule } from './app-routing.module';
5  import { AppComponent } from './app.component';
6  import { MonPremierComponentComponent } from './mon-premier-component/mon-premier-component.component';
7
8  @NgModule({
9    declarations: [
10     AppComponent,
11     MonPremierComponentComponent
12   ],
13   imports: [
14     BrowserModule,
15     AppRoutingModule
16   ],
17   providers: [],
18   bootstrap: [AppComponent]
19 })
20 export class AppModule { }
21
```

A présent, regardez dans le fichier `mon-premier-component.component.ts`

Un sélecteur : `app-mon-premier-component` a été créé. Nous pouvons donc utiliser ce sélecteur dans notre code pour y insérer ce component.

Revenez dans `app.component.html` et modifiez-le en ajoutant la balise

<app-mon-premier-component></app-mon-premier-component>

Dans votre navigateur, vous verrez apparaître le nouveau contenu

Structure du projet

AppComponent
=> regroupe les fichiers:
- app.component.html
- app.component.css
- app.component.ts)

MonPremierComponent
=> regroupe les fichiers:
- mon-premier-component.component.html
- mon-premier-component..component.css
- mon-premier-component..component.ts

Il est à l'intérieur du composant AppComponent