Démarrer avec Angular

Framework front-end

Sources et contenu complémentaires : https://openclassrooms.com/fr/courses/7471261-debutez-avec-angular/7549261-decouvrez-le-framework-angular

Qu'est-ce qu'Angular ?

Angular est un framework JavaScript qui vous permet de développer des applications "efficaces et sophistiquées", comme dit la documentation. Il permet notamment de créer ce qu'on appelle des Single Page Applications (ou SPA) : des applications entières qui tournent dans une seule page HTML grâce au JavaScript, et surtout sans temps de chargements.

Le développement Angular passe par trois langages principaux :

- **HTML** pour structurer – toutes vos connaissances avec ce langage vous seront utiles, et Angular viendra vous ajouter quelques nouveautés
- **SCSS**, ou **SASS** pour les styles – le SCSS est une surcouche du CSS qui y apporte des fonctionnalités supplémentaires, mais qui permet également d'écrire du CSS pur si on le souhaite
- **TypeScript** pour tout ce qui est dynamique, comportement et données – un peu comme le JavaScript sur un site sans framework. Pas d'inquiétude si vous ne connaissez pas TypeScript : La syntaxe est quasiment identique à une exception près : Contrairement au JavaScript, TypeScript introduit le typage **strict** de nos variables. Contrairement à Javascript où l'on peut déclarer une variable sans la typer, et on verra bien ce qui atterrit dedans… En TypeScript, nous avons une sécurité supplémentaire et surtout **obligatoire**.

Comment ça marche ?

Le projet généré par les instructions dans la procédure d'installation est organisé en différents dossiers que nous n'aborderons pas pour l'instant, car ils contiennent pour certains des informations importantes par rapport à la **configuration**, les instructions de build, etc. Nous allons nous concentrer dans un premier temps sur le dossier **src/**.

Angular utilise une logique complètement différente de ce que nous avons vu pour l'instant, et pour s'en rendre compte, nous allons regarder le fichier src/index.html du projet que nous allons créer.

src > index.html > ...

```
1  <!doctype html>
2  <html lang="en">
3    <head>
4      <meta charset="utf-8">
5      <title>Mon Premier projet Angular</title>
6      <base href="/">
7      <meta name="viewport" content="width=device-width, initial-scale=1">
8      <link rel="icon" type="image/x-icon" href="favicon.ico">
9    </head>
10   <body>
11     <app-root></app-root>
12   </body>
13 </html>
14
```

Comme vous pouvez le voir il n'y a quasiment rien, simplement cette mystérieuse balise **<app-root></app-root>**. Deux problèmes se posent alors :

- On ne connaît pas cette balise, quel est son rôle ?
- Pourquoi n'y a-t-il rien dedans alors qu'il y a bien du contenu si on se rend sur http://localhost:4200/ ?

C'est précisément là que se joue toute la magie d'Angular, **il sait qu'il doit charger du contenu dans cette balise** grâce à une série d'instructions bien précises. Et ce contenu, on appelle ça un **composant (component en anglais).**

Allons voir comment ça se passe exactement.

Les composants

Dans le dossier **/src/app**, on retrouve plusieurs fichiers mais seuls trois vont nous intéresser pour l'instant :

- **app.scss**, le moins important des trois pour l'instant puisqu'il ne contient rien, cependant, retenez que c'est ici que l'on va écrire le code SCSS pour styliser notre page.
- **app.html**, qui contient… Tout le code HTML affiché en page d'accueil, y compris les SVG complets, une syntaxe bien particulière de boucle **@for**, et une balise <style> remplie de CSS. Comme précisé en bas du fichier, ce n'est qu'un **placeholder** et c'est voué à être remplacé.
- Et enfin, comme à chaque fois qu'on a besoin de faire parler des fichiers entre eux en informatique, on a besoin d'un chef d'orchestre : **app.ts**

src > app > TS app.ts > ...

```ts
import { Component, signal } from '@angular/core'; // Inclusions des classes
import { RouterOutlet } from '@angular/router'; //  Javascript nécessaires

// Déclaration du composant
@Component({
  selector: 'app-root', // Indication de la balise dans laquelle il doit être chargé
  imports: [RouterOutlet], // Important de la classe RouterOutlet
  templateUrl: './app.html', // Fichier HTML à charger pour ce composant
  styleUrl: './app.scss' // Fichier SCSS à charger pour ce composant
})
// Export de la classe avec des paramètres passés en tant que propriété de la classe
// (Ici, le titre qui va s'afficher sur la page)
export class App {
  protected readonly title = signal('snapface');
}

```

Afin de mieux comprendre comment fonctionne un composant, on va créer le nôtre.

Pour cela, on va couper le serveur local en faisant simplement CTRL + C là où on a fait le **ng serve**, puis on va exécuter la commande suivante : **ng generate component mon-composant**

Le CLI Angular va alors créer un dossier dans **/src/app** reprenant le nom de notre composant, et en créant les trois fichiers dont nous avons parlé plus haut.

On retrouve dans notre fichier **mon-composant.html** une simple balise <p> pour vérifier que le composant marche. On va alors tenter de l'afficher en faisant deux choses :

- Premièrement, dans notre composant **racine** (app-root ne s'appelle pas comme ça pour rien), nous allons lui dire d'importer notre nouveau composant. On va donc modifier notre fichier **app.ts** pour y inclure la syntaxe d'import

app.ts M

src > app > app.ts > ...

```ts
import { Component } from '@angular/core'; // Inclusion des classes Javascript nécéssaires
import { MonComposant } from "./mon-composant/mon-composant"; // Import du composant

@Component({
  selector: 'app-root', // Indication de la balise dans laquelle le composant va être chargé
  imports: [
    MonComposant // Déclaration du fait que ma classe App dépend de la classe MonComposant
],
  templateUrl: './app.html', // Fichier html à charger
  styleUrl: './app.scss' // Fichier SCSS à charger
})
// Export de la classe
export class App {
}
```

- Deuxièmement, on va prévoir un espace dans **app.html** pour accueillir notre nouveau composant. Pour cela, on va s'en référer au nom qui lui a été donné dans **/app/mon-composant.ts**, à savoir **app-mon-composant**. Voici donc le fichier **app.html** :

app.html M

src > app > app.html > app-mon-composant

Go to component

```html
<app-mon-composant></app-mon-composant>
```

Avant d'aller plus loin, résumons.

Angular est un **framework**, c'est-à-dire un système de fichiers normés, conçus pour fonctionner en harmonie et **structurer** notre développement.

Il fonctionne avec une arborescence de **composants**, qui sont en quelque sorte des **briques de code** que l'on va fabriquer avec plusieurs paramètres : leur contenu, leur style, leur destination, et leurs dépendances.

Angular permet de coordonner des fichiers de contenu, et de style dans une structure grâce à des comportements décrits en TypeScript.

Et voilà, Angular va donc dérouler le scénario suivant en lisant les fichiers TypeScript que nous avons écrits :

1. Il va charger le composant **racine** du projet.
2. Il va voir que ce composant importe des **dépendances**, à savoir d'autres composants comme « mon-composant ». Il va alors également charger ces composants.
3. Il va regarder quels sélecteurs sont déclarés dans chaque composant, et utiliser ce sélecteur pour trouver **où** charger le composant dans le composant **racine**.