DÉVELOPPEMENT DES APPLICATIONS MOBILES HYBRIDES MULTIPLATEFORMES

CH 6 - Introduction au framework Angular

FORMATEUR : MAHAMANE SALISSOU YAHAYA : ysalissou@gmail.com

(TechnoLAB -ISTA)

4ième année

OCTOBRE 2020

- AngularJS est un **framework** JavaScript libre et open source développé par Google. Il permet la construction d'applications client reactives en HTML et en **TypeScript**. Angular est d'ailleurs lui-même écrit en TypeScript.
- Avant tout, Angular est framework front-end et généralement, on fait tourner la plupart des scripts dans le navigateur du visiteur.
- Angular est un framework orienté composant. Tu vas écrire de petits composants, et assemblés, ils vont constituer une application complète.

- Un composant est un groupe d'éléments HTML, dans un template, dédiés à une tâche particulière. Pour cela, tu auras probablement besoin d'un peu de logique métier derrière ce template, pour peupler les données, et réagir aux événements par exemple.
- Cette orientation composante est largement partagée par de nombreux frameworks front-end : **ReactJS**, **Vue JS**, …
- Les applications Angular peuvent être écrites en ES5, ES6, ou TypeScript.
- Et tu te demandes peut-être qu'est-ce que TypeScript, et ce qu'il apporte de plus.
- JavaScript est dynamiquement typé.
- JavaScript (JS) est une des implémentations d'une spécification standardisée, appelée ECMAScript.
- TypeScript est dynamiquement typé. Il ajoute une tonne de fonctionnalités à JavaScript, comme les classes, les constantes, ….
- Pour mieux comprendre le fonctionnement d'Angular, rien de mieux que développer un petit projet web basé sur celui-ci

NodeJS et (NPM ou YARN)

- Aujourd'hui, pratiquement tous les outils JavaScript moderne sont faits pour **Node.js** et **NPM**. Tu devras installer Node.js et NPM sur ton système.

- Comme la meilleure façon de le faire dépend de ton système d'exploitation, le mieux est d'aller voir le site officiel. Assure-toi d'avoir une version suffisamment récente de Node.js (en exécutant **node --version**).

Yarn VS Npm

- **Yarn** est plus rapide et a plus de fonctionnalités que **npm**. La vitesse est améliorée grâce à l'utilisation de flux d'installations parallèles, de la mise en cache hors ligne et de la mise en file d'attente des demandes. Il a même vu le jour parce que
  - Npm présentait des problèmes de cohérence lors de l'installation de dépendances sur des machines différentes.
  - Npm n'était pas assez rapide.

- Ainsi, plus un projet possède de dépendances, plus yarn se démarquera en termes de rapidité. Gardez en tête que Yarn n'est qu'une surcouche maligne, et que maligne c'est à la fois une forme d'intelligence et de fourberie.

- **Angular CLI**

  Est un outil en ligne de commande pour démarrer rapidement un projet, déjà configuré avec Webpack comme un outil de construction, des tests, du packaging, etc. CLI veut dire « Command Line Interface »

- **Webpack**

  Est un outil qui est aujourd'hui incontournable dès lors que l'on travaille sur des projets JavaScript complexes.

  Il va nous permettre de morceler notre code sous forme de modules qui seront ensuite fusionnés en un seul fichier par Webpack.

  Il dispose, en plus, d'un système de "loaders" qui vont permettre d'inclure de nouveaux types de fichiers ou d'appliquer des transformations spécifique (comme une transformation ES2015->ES5).

Rappel des procédures d'installation des outils

- **Installation NODE, (NPM OU YARN)**

  INSTALLATION :

  - ✓ Pour l'installation de Node JS et NPM, voir le site https://nodejs.org
  - ✓ Pour Yarn : https://yarnpkg.com/lang/en/docs/install

  VOIR LES VERSIONS :

  - ✓ **node –v**
  - ✓ **npm –v**
  - ✓ **yarn –v**

- Installation Angular CLI

  INSTALLATION :

  - ✓ **npm install -g @angular/cli**

  VERSION :

  - ✓ **ng –version**

  UPDATE

  - ✓ **npm uninstall -g angular-cli**
  - ✓ **npm cache clean or npm cache verify (if npm > 5)**
  - ✓ **npm install -g @angular/cli@latest**

Créer un nouveau projet

- **ng new nom_du_projet** # Cette commande va créer un nouveau dossier nom_du_projet **.**
- A la question Would you like to add Angular routing?, répondez y, puis choisissez le préprocesseur SCSS.
- Ça va mettre un peu de temps à se créer, mais pas de panique vous êtes sur la bonne voie ;-).
- Si vous rencontrez une exception à la création du projet (sous Windows notamment), n'hésitez pas à supprimer le cache npm et recréer votre projet :
  - ✓ **$ rm -rf nom_du_projet** # Suppresion du dossier, uniquement si ça bug.
  - ✓ **$ npm cache clean --force**
  - ✓ **$ ng new nom_du_projet**
- Une fois la création terminée, on va pouvoir lancer notre projet :
  - ✓ **$ cd nom_du_projet**
  - ✓ **$ ng serve –open**
- Puis recompilez vos packages npm :
  - ✓ **$ npm install**
- **Vous n'avez plus qu'à visualiser votre application depuis votre navigateur.**

Structure et architecture d'un projet Angular

OCTOBRE 2020

Structure d'un projet

À l'intérieur d'un projet angular on trouve un certain nombre de dossiers et de fichiers :

- **e2e** : ce dossier stocke des scripts pour effectuer des tests unitaires, un ensemble d'énoncés et d'instructions qui permettent de vérifier que son code fonctionne selon un certain cahier des charges.
- **node_modules** : c'est dans ce dossier que sont installés tous les plugins Node installés via npm.
- **src** : c'est dans ce dossier que sont stockés les fichiers sources, le code quoi. C'est dans ce dossier que l'on passera 99% du temps.
- **package.json** : fichier de configuration pour Node
- **tsconfig.json** : fichier de configuration pour le compilateur de TypeScript (tsc).
- **tslint.json** : tslint est utilitaire qui permet de vérifier les fichiers TypeScript (bug, import non utilisé,...)
- **angular.json** : Angular CLI a lui-même son fichier de configuration si tu veux changer quelques-unes des options par défaut.
- **karma.conf.js** : est utilisé pour configurer les tests (on en reparlera dans un chapitre consacré aux tests).
- **src/main.ts** : Angular CLI génère par défaut un fichier séparé contenant cette logique de démarrage
- **src/index.html** : La CLI a créé un fichier pour nous, qui est la seule page de notre application (d'où le nom SPA, single page application).

Architecture Angular

Le bloc de base d'une application Angular est le module **NgModules** qui sert de contexte de compilation et d'exécution à un autre élément nommé **Composant**.

Un composant peut être vu comme la combinaison :

- D'une **Vue** : du contenu HTML
- D'un **Modèle** de données : les informations qui vont être affichées dans le contenu HTML
- D'un **Contrôleur**, qui va se charger de la logique derrière l'affichage des données dans la vue.

Un composant peut être constitué d'autres composants. Par exemple :

Tweet [Composant Root]
Entête (Titre, logo,...
Un contenu principal [Composant Content]
Tweets [Composant liste de Tweets]
Un tweet [Composant Tweet] est constitué de contenu

- ce contenu peut être soit une image [Composant image], soit du texte [Composant texte]
- ce contenu est aussi fait de commentaires [Composant Commentaire]

Architecture Angular

L'intérêt d'une architecture en composants est que si jamais on souhaite étendre une fonctionnalité particulière, plutôt que de la redéfinir, on va créer un composant qui pourra être appelé partout (afficher des tweets en page d'accueil, sur son profil, dans les résultats de recherche,...).

Le composant principal d'Angular est définit à l'intérieur du fichier **src/app/app.component.ts.**

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

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.scss']
})
export class AppComponent {
  title = 'trombischool-web';
}
```

On retrouve à côté de ce fichier d'autres qui forment avec lui le MVC du projet.

app.component.css
app.component.html
app.component.spec.ts
app.component.ts
app.module.ts

Architecture Angular

Une application a toujours au moins un module racine qui permet le lancement du projet (à l'exemple d'un fichier index.html en racine d'un site web).
C'est ce module qui va amorcer le composant Root (**AppComponent**).
Par convention, celui-ci s'appelle **AppModule** et est défini dans le fichier **src/app/app.module.ts**.

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

import { AppRoutingModule } from './app-routing.module';
import { AppComponent } from './app.component';

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

S'il fallait faire un comparatif avec un véhicule, les composants seraient des éléments comme le pare-brise, les rétroviseurs, les roues, ...tandis que le module Root serait le moteur, sans lequel le véhicule, même le plus beau au monde (avec les plus beaux composants), ne pourrait démarrer.

Templates Angular

OCTOBRE 2020

*ngFor

Permet de boucler sur les éléments d'un tableau à l'intérieur d'un template html.

Supposons que l'on ait définit la liste des mois de l'année dans une liste :

```
let months_of_year = ['Janvier', 'Fevrier', 'Mars', 'Avril', 'Mai', 'Juin', 'Juillet',
                      'Aout', 'Septembre', 'Octobre', 'Novembre', 'Décembre']
```

l'affichage de tous les éléments se fait simplement de la manière suivante :

```
<div *ngFor="let month of months_of_year">
   Mois de l'année : {{month}}
</div>
```

*ngIf

Comme vous pouvez le deviner, ***ngIf est le "if...else"** adapté aux templates.

```
<span *ngIf="isConnected">Je suis connecté.</span>
```

Pipe

Comme dans la plupart des moteurs de templates, Angular permet l'utilisation de pipes qui permettent de modifier une variable ou un contenu avant qu'il soit affiché.
Le framework propose un certain nombre de pipes prêts à l'emploi, comme titlecase, currency,...Mais il est tout à fait possible de créer son propre pipe.

```
<div>
   <h2>{{ 'charles edou nze' | titlecase }}</h2>
</div>
```

titlecase permet de mettre en capitale les premières lettres de chaque mot. Ce qui donnera le résultat suivant :

```
Charles Edou Nze
```

Plus d'informations sur les Pipes Angular : https://angular.io/guide/pipes

Binding

OCTOBRE 2020

Template Interpolation

Comme de nombreux langages de "templating", Angular utilise la syntaxe "double curly braces" pour l'interpolation:

src/app.component.ts

```ts
@Component({
    selector: 'wt-app',
    templateUrl: './app.component.html'
})
export class AppComponent {
    bookName = 'eXtreme Programming Explained';
}
```

src/app.component.html

```html
<div>
    <span>{{ bookName }}</span>
</div>
```

La syntaxe d'interpolation permet d'accéder directement aux propriétés du composant associé (un peu comme si toutes les expressions étaient préfixées par un this.).

Property binding

L'interpolation ne suffira pas pour tout contrôler (images, styles, etc...).

Mieux que le contrôle des attributs, le "property binding" nous permet de contrôler n'importe quelle propriété d'un élément du DOM en s'inspirant de la syntaxe native (Vanilla JS) : button.disabled = false ou encore button['disabled'] = false (pour désactiver un bouton par exemple).

Ce qui donne la syntaxe suivante :

src/app.component.html

```html
<img [alt]="bookName" [src]="bookPictureUrl">


<button type="button" [disabled]="!isAvailable">BUY</button>
```

Les données proviennent encore du code TypeScript du composant.

src/app.component.ts

```typescript
...

export class AppComponent {

    bookName = 'eXtreme Programming Explained';
    bookPictureUrl = 'https://robohash.org/xp?set=set4';
    isAvailable = false;

}
```

Attribute, class, and style bindings

Class Binding

```
<div [class.wt-important]="isImportant">Important Stuff</div>
```

https://angular.io/guide/template-syntax#class-binding

Style Binding

```
<button [style.background-color]="isValid ? null : 'red'">Save</button>
```

```
<button [style.font-size.em]="isImportant ? 2 : 1" >Big</button>
```

https://angular.io/guide/template-syntax#style-binding

Event binding

Grâce à l'interpolation et le "property binding", nous contrôlons le contenu affiché mais cela manque d'interactions avec l'utilisateur.

Il nous faut ajouter des "listeners" d'évènements déclenchés sur les éléments du DOM afin de modifier l'état de notre application et laisser Angular mettre à jour la "view".

La syntaxe Angular ci-dessous est équivalente au code natif
button.addEventListener('click', () => this.buy())

src/app.component.html

```
<button
        type="button"
        (click)="buy()">BUY</button>
```

src/app.component.ts

```
...
export class AppComponent {

    buy() {
        ...
    }

}
```

Two-way binding

Le "**two-way**" Data Binding : C'est une combinaison du Property Binding et du Event Binding sous une unique annotation.

Dans ce cas là, le component se charge d'impacter le DOM en cas de changement du modèle ET le DOM avertit le Component d'un changement via l'émission d'un évènement.

Le mécanisme se rapproche du fameux ngModel, mais avec des algorithmes de Data Binding différents.

Voici la syntaxe utilisée dans Angular pour déclarer un tel Data Binding :

```
<ma-taille [(taille)] ></ma-taille>
```

Il faut se souvenir que les parenthèses sont entre les crochets. Pour se souvenir de cela, cette notation se nomme "banana in a box"… ce que nous venons de voir aurait pu également s'écrire :

```
<ma-taille
  [taille]="maTaille"
  (tailleChange)="maTaille=$event"></ma-taille>
```

Template variables

Les variables de modèle vous aident à utiliser les données d'une partie du template dans une autre partie du template.

Avec les variables de template, vous pouvez effectuer des tâches telles que répondre aux entrées de l'utilisateur ou affiner les formulaires de votre application.

Dans ce template, vous utilisez le symbole de hachage, **#**, pour déclarer une variable de template.

La variable de template suivante, **#phone**, déclare une variable de téléphone sur un élément **<input>.**

Vous pouvez faire référence à une variable de template n'importe où dans le template du composant. Ici, un <button> plus bas dans le template fait référence à la variable phone

src/app/app.component.html

```
<input #phone placeholder="phone number" />

<!-- lots of other elements -->

<!-- phone refers to the input element; pass its `value` to an event handler -->
<button (click)="callPhone(phone.value)">Call</button>
```

@Input() and @Output()

Comme dans la plupart des moteurs de templates, Angular permet l'utilisation de pipes qui permettent de modifier

SVG as templates

Comme dans la plupart des moteurs de templates, Angular permet l'utilisation de pipes qui permettent de modifier