# TD 4 Angular

## Partie 1 :

### Changements avec AngularJS

- **Les contrôleurs** : L'architecture traditionnelle MVC est remplacée par une architecture réactive à base de composants web. L'architecture MVC est une architecture classique que l'on retrouve dans beaucoup de Framework (Symfony, Django, Spring) permettant de découper le Modèle, la Vue et le Contrôleur.
- **Les directives** : La définition existante de l'objet Directive est retirée, et remplacée par trois nouveaux types de directives à la place : les composants, les directives d'attributs et les directives structurelles.
- **Le $scope** : Les scopes et l'héritage de scope sont simplifiés et la nécessité d'injecter des $scopes est retirée.
- **Les modules** : Les modules AngularJS sont remplacés par les modules natifs d'ES6.
- **jQLite** : Cette version plus légère de jQuery était utilisée dans AngularJS. Elle est retirée dans Angular, principalement pour des raisons de performance.
- **Le two-way data-binding** : Pour les mêmes raisons de performances, cette fonctionnalité n'est pas disponible de base. Cependant, et il est toujours possible d'implémenter ce mécanisme avec Angular.

Angular est un Framework orienté composants. Un composant est l'assemblage d'un morceau de code HTML, et d'une classe JavaScript dédiée à une tâche particulière. Ces composants reposent sur le standard des Web Components.

Il y a quatre éléments à la base de toute application Angular :

- Le Module
- Le Composant. (ou *Component* en anglais)
- Le Template.
- Les métadonnées.

Un module est une brique de l'application. Chaque module est dédié à une fonctionnalité particulière

Les **composants** sont la base des applications Angular pour gérer les interactions avec l'utilisateur. On définit la logique d'application d'un composant, ce qu'il doit faire pour supporter la vue à l'intérieur d'une classe, et cette classe interagit avec la vue à travers un ensemble de propriétés et de méthodes.

Un ***template*** est une forme de HTML spéciale qui dit à Angular ce que doit afficher le composant.

Les **métadonnées** indiquent à Angular comment il doit traiter une classe.
Nous ajoutons l'annotation *@component* pour indiquer à Angular que la classe est un composant. Les **annotations** ont besoin de paramètres. Le décorateur *@Component* prend en paramètre un objet qui contient les informations dont Angular a besoin pour créer et lier le composant et son template.

- **Selector** : un sélecteur est un identifiant unique dans votre application, qui indique à Angular de créer et d'insérer une instance de ce composant à chaque fois qu'il trouve une balise *<mon-app></mon-app>* dans du code HTML d'un composant parent.
- **Template** : Il est également possible d'écrire le code du template dans un fichier à part. Dans ce cas, il faut remplacer *template* par *templateUrl*, et indiquer chemin relatif vers le fichier du template.

```
1   @Component({
2    selector: 'mon-app',
3    template: '<p>Ici le template de mon composant</p>'
4   })
5   export class AppComponent { ... }
```

Le transpileur est un outil qui permet de publier son code pour les navigateurs qui ne supportent pas encore l'ES6

TypeScript et angular :

- Créer un fichier Hello.ts

Typage Typescript :

```
// Syntaxe pour déclarer une variable typé en TypeScript
var variable: type;
```

```
// On déclare un nombre :
var lifePoint: number = 100;

// On déclare une chaîne de caractère :
var name: string = 'Green Lantern';

// On déclare une variable qui correspond à une classe de notre application !
var greenLantern: Hero = new Hero(lifePoint, name);

// On peut créer un autre héros de type Hero :
var superMan: Hero = new Hero(lifePoint, 'Superman');

// un tableau de héros, qui contient tous les héros de mon application !
var heros: Array<Hero> = [greenLantern, superMan];
```

Ajoutez :
lifePoint = ‘some-string’;

-Que se passe t-il ?
- Corrigez l’erreur.

Les Web Components sont composés de :

- Les éléments personnalisés (*Custom Elements*) ;
- Le DOM de l’ombre (*Shadow DOM*) ;
- Les templates HTML (*HTML Templates*) ;
- Les imports HTML (*HTML Imports*).

**Partie 2 : Exercice d’application :**

Nous utilisons stackblitz pour le développement. En vous basant sur la partie 1, étudier l’exemple HelloWorld : https://stackblitz.com/edit/angular-hello-world

**Partie 3 : Exercice**

Ouvrez
https://stackblitz.com/angular/ybmvyemqokb?file=src%2Fapp%2Fapp.component.ts

Ouvrez le fichier template product-list.component.html.

Afficher la liste des produits :

```
<h2>Produits</h2>

<div *ngFor="let product of products">
```

Pour afficher les noms des produits, utilisez la syntaxe {{ }}

- Ajoutez un lien avec l'attribut title

```
<a [title]="product.name + ' details'">
    {{ product.name }}
  </a>
```

- Ajoutez les descriptions de produits. Sur l'élément <p>, utilisez une directive pour qu'Angular ne crée l'élément que si le produit actuel a une description.

```
 <p *ngIf="product.description">
  Description: {{ product.description }}
 </p>
```

- Ajoutez un bouton pour que les utilisateurs puissent partager un produit avec des amis. Liez l'événement 'click' du bouton à la méthode share() (dans product-list.component.ts).

```
<button (click)="share()">
  Partager
 </button>
```

L'application dispose désormais d'une liste de produits et d'une fonction de partage.

**Composants :**

*Les composants* définissent des parties dans l'interface utilisateur et permettent de réutiliser des ensembles de fonctionnalités de l'interface utilisateur.

Un composant se compose de

- Une classe de composants : qui gère les données et les fonctionnalités.
- Template HTML qui détermine l'interface utilisateur.
- Styles des composants qui définissent l'apparence.

Une application angular comprend une arborescence de composants, dans laquelle chaque composant a un objectif et un comportement spécifique.

- app-root : Il s'agit du premier composant à charger et du parent de tous les autres composants.
- app-top-bar est le nom du magasin et le bouton panier
- app-product-list est la liste de produits

**Input**

Les produits sont importés de la liste des produits définis dans produits.ts

**Question** : créer une nouvelle fonction d'alerte qui prend un produit en entrée.

- Faites un clic droit sur le dossier app et utilisez le Angular Generator pour générer un nouveau composant nommé product-alerts.
- Le générateur crée des fichiers de démarrage pour les trois parties du composant :
    - product-alerts.component.ts
    - product-alerts.component.html
    - product-alerts.component.css
- Ouvrez product-alerts.component.ts.
- Le décorateur. Cela indique que la classe suivante est un composant. Il fournit des métadonnées sur le composant, y compris son sélecteur, ses modèles et ses styles.@Component()
- La définition de composant exporte également la classe ProductAlertsComponent, qui gère les fonctionnalités du composant.

- Recevoir un produit en entrée :
- Importer Inputdepuis @angular/core

```
- import { Component, OnInit } from '@angular/core';
- import { Input } from '@angular/core';
-
- @Component({
-   selector: 'app-product-alerts',
-   templateUrl: './product-alerts.component.html',
-   styleUrls: ['./product-alerts.component.css']
- })
- export class ProductAlertsComponent implements OnInit {
-
-   constructor() { }
-
-   ngOnInit() {
-   }
-
- }
-
```

In the ProductAlertsComponent class definition, define a property named product with an @Input() decorator. The @Input() decorator indicates that the property value passes in from the component's parent, the product list component.

```
export class ProductAlertsComponent implements OnInit {
  @Input() product;
  constructor() { }
```

Modifiez : product-alerts.component.html

```
<p *ngIf="product.price > 700">
  <button>M'alerter</button>
</p>
```

- Passez le produit actuel en entrée au composant grâce au binding

```
<button (click)="share()">
   Partager
 </button>

 <app-product-alerts
 [product]="product">
</app-product-alerts>
```

**Output**

Pour que le bouton "M'alerter" fonctionne, vous devez configurer deux choses:

- le composant d'alerte produit pour émettre un événement lorsque l'utilisateur clique sur "M'alerterr"
- le composant de liste de produits pour agir sur cet événement

1. Ouvrez product-alerts.component.ts.
2. Importer Output et EventEmitter

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

Dans la classe component, définir une propriété notify avec un décorateur @Output() decorator et une instance EventEmitter().
Ceci permet le component product alert d'emettre un évènement lorsque la valeur de la propriété notify change.

```
export class ProductAlertsComponent implements OnInit {
  @Input() product;
  @Output() notify = new EventEmitter();
  constructor() { }
```

Dans la template product alert, product-alerts.component.html,
Mettre à jour le boutton M'alerter avec un évènement "binding" pour appeler la méthode notify.emit().

```
<p *ngIf="product.price > 700">
```

```
 <button (click)="notify.emit()">M'alerter</button>
</p>
```

Définissez le comportement qui doit se produire lorsque l'utilisateur clique sur le bouton.

C’est le composant de liste de produits, et non le composant d'alertes de produit, qui agit lorsque l'enfant déclenche l'événement. Dans product-list.component.ts, définissez une méthode onNotify() similaire à la méthode share():

```
share() {
  window.alert('Le produit a été partagé');
}

onNotify() {
  window.alert('Vous serez alertés');
}
```

Enfin, mettez à jour le composant de liste de produits pour recevoir la sortie du composant d'alertes de produit.
Dans product-list.component.html, liez le app-product-alertscomposant (qui affiche le bouton "M'avertir") à la méthode onNotify() du composant de liste de produits.

```
share() {
  window.alert('Le produit a été partagé');
}

onNotify() {
  window.alert('Vous serez alertés');
}
```