Angular TP-03

Créez un component

Depuis le dossier principal de votre projet, tapez la commande suivante :
ng generate component students
Le CLI a créé un nouveau sous-dossier «students» et y a créé un fichier template, une feuille de styles et un fichier component.
Le CLI nous prévient également qu'il a mis à jour le fichier app.module.ts : ouvrez-le maintenant pour voir de quoi il s'agit :

```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
import { AppComponent } from './app.component';
import { StudentsComponent } from './students/students.component';

@NgModule({
  declarations: [
    AppComponent,
    StudentsComponent
  ],
```

Le CLI a ajouté students à l'array declarations de votre module et le statement import en haut du fichier.

Regardez maintenant le fichier students.component.ts :

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

Vous constaterez que le CLI a créé un sélecteur : app-students. Nous pouvons donc utiliser ce sélecteur dans notre code pour y insérer ce component.

Revenez dans app.component.html et modifiez-le comme suit :

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

Dans votre navigateur, vous verrez le même titre qu'avant et, sur une deuxième ligne, le texte "students works". Il s'agit du texte par défaut créé par le CLI que vous trouverez dans students.component.html :

String interpolation
https://angular.io/guide/template-syntax#interpolation-and-template-expressions

Ouvrez ensuite students.component.html, supprimez le contenu, et entrez le code ci-dessous :

```
<li class="list-group-item">
  <h4>Liste des étudiants</h4>
</li>
```

Ensuite, ouvrez app.component.html, et remplacez tout le contenu comme suit :

```
<div class="container">
  <div class="row">
    <div class="col-xs-12">
      <h2>Les étudiants</h2>
      <ul class="list-group">
        <app-students></app-students>
        <app-students></app-students>
        <app-students></app-students>
      </ul>
    </div>
  </div>
</div>
```

Les classes CSS utilisées ici sont des classes issues de Bootstrap.

Modifiez students.component.html ainsi :

```
<li class="list-group-item">
  <h4>Etudiant : {{ studentName }}</h4>
</li>
```

Ouvrez maintenant student.component.ts :

Ajoutez maintenant la ligne de code suivante en haut de la déclaration de class :

```
export class StudentsComponent implements OnInit {

  studentName: string = 'Henri';
```

La déclaration de type string n'est pas obligatoire, c’est la syntaxe TypeScript pour fixer le type d’une varaible.

Maintenant, ajoutez la variable studentStatus qui est initialisée avec « present » et qui doit s’afficher après le nom.

String interpolation avec une méthode

Dans le fichier StudentsComponent ajoutez une méthode dans le constructeur :

```
  getStatus() {
    return this.studentStatus;
  }
```

Dans le template rajouter l'interpolation de {{getStatus()}} après {{studentName}}.

Property binding

On va simuler les droits de l'application. Dans AppComponent ajoutez dans la class la variable isAuth initialisée à 'false'.

Puis dans le constructeur de la class ajouter une méthode pour simuler l'authentification au bout de 4 secondes.

```
setTimeout(
    () => {
        this.isAuth = true;
    }, 4000
);
```

Ajoutez un bouton au template app.component.html, en dessous de la liste des étudiants :

```
</ul>
<button class="btn btn-success" [disabled]="!isAuth">Tous présent</button>
```

La propriété disabled permet de désactiver le bouton. Afin de lier cette propriété au TypeScript, il faut le mettre entre crochets et l'associer à la variable.
Le point d'exclamation inverse la valeur de isAuth. Le bouton sera donc caché si isAuth est false.
Pour en voir l'effet, rechargez la page dans votre navigateur et observez comment le bouton s'active au bout de quatre secondes.

Event binding

Dans app.component.html, modifier le bouton comme suit :

```
    <button class="btn btn-success"
            [disabled]="!isAuth"
            (click)="allPresent()">Tous Présents</button>
```

Comme vous pouvez le constater, on utilise les parenthèses () pour créer une liaison à un événement.
Maintenant rajouter dans la class de app.component.ts la méthode suivante :

```
allPresent() {
    alert('Ils sont tous là !');
}
```

Dans le navigateur testez le bouton après identification.

Two-way binding
https://angular.io/api/forms/FormsModule#formsmodule

Pour pouvoir utiliser le two-way binding, il vous faut importer FormsModule depuis @angular/forms dans votre application. Ajoutez-le à l'array imports de votre AppModule et en haut du fichier avec le mot clé import :

```
import { FormsModule } from '@angular/forms';
```

```
@NgModule({
  imports: [
    BrowserModule,
    FormsModule
  ],
```

Le two-way binding emploie le mélange des syntaxes de property binding et d'event binding : des crochets et des parenthèses [()], on utilisera ici la directive NgModel.
https://angular.io/api/forms/NgModel#ngmodel
Ajoutez en dessous de <h4> un <input> dans votre template students.component.html :

```
<input type="text" class="form-control" [(ngModel)]="studentName">
```

Testez !

Propriétés personnalisées https://angular.io/api/core/Input#input

Il faut utiliser le décorateur @Input() en remplaçant la déclaration de la variable studentName :
rajoutez en haut de studentsComponent l'import de Input :

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

Puis dans la class, modifiez studentName comme suit :

```
  @Input() studentName: string;
```

On peut fixer la variable depuis la balise <app-student> dans app.component.html

```
<app-students studentName="Henri"></app-students>
```

ajouter 2 noms d'étudiants au 2 autres balises app-students.

Testez !

Plutôt que de déclarer les noms dans le html, utilisez le Property Binding en déclarant les 3 variables dans la class app.component.ts.

studentOne, studentTwo …

Il faut lier le contenu des variables au template.
Utilisez les crochets [] pour lier le contenu de ces variables.

```
<app-students [studentsName]="studentsOne"></app-students>
```

Répétez l'opération pour les autres étudiants puis testez !

Maintenant, ajoutez une Input pour studentStatus, mais au lieu de le lier à une variable de app.component. Déclarez la valeur dans le html.

- Déclarez l'input,

- liez les variables dans le html en leurs donnant la valeur ‘present’ ou ‘absent’ pour les 3 étudiants.

Pour passer un string directement il faut employer les crochets pour le property binding et il faut le mettre entre apostrophes.

[maVar]="’monString’";

Testez !

Les directives structurelles

*ngIf : https://angular.io/api/common/NgIf#ngif

Dans le studentsComponent ajouter en-dessous de <li class="list-group-item"> :

```
<div style="width:20px;height:20px;background-color:red;"
        *ngIf="studentStatus === 'absent'"></div>
```

Testez !

*ngFor : https://angular.io/api/common/NgForOf#ngforof

Au lieu de déclarer les étudiants séparément, déclarez un tableau d’objets dans la class AppComponent :

```
  students = [
    {
      name: 'Henri',
      status: 'present'
    },
    {
      name: 'Louis',
      status: 'absent'
    },
    {
      name: 'Philippe',
      status: 'present'
    }
  ];
```

Supprimez les anciennes déclarations (studentOne...)

Puis utilisez la directive ngFor dans le Html. Remplacez les 3 lignes par :

```
<app-students *ngFor="let student of students"
                [studentName]="student.name"
                [studentStatus]="student.status"></app-students>
```

Testez !

Les directives par attribut

NgStyle : https://angular.io/api/common/NgStyle#ngstyle

On va mettre en évidence les étudiants en fonction de leur status.

Dans le Html de StudentsComponent remplacer la ligne <h4> par :
<h4 [ngStyle]="{color: getColor()}">étudiant : {{ studentName }} -- Statut : {{ getStatus() }}</h4>

Puis ajouter la fonction dans la class de StudentsComponent
```
getColor() {
    if(this.studentStatus === 'present') {
      return 'green';
    } else if(this.studentStatus === 'absent') {
      return 'red';
    }
}
```

Testez !

NgClass : https://angular.io/api/common/NgClass#ngclass

Dans StudentsComponent, on va utiliser des Class Bootstrap à la balise <li> en fonction du statut de l'étudiant :
```
<li [ngClass]="{'list-group-item': true,
                'list-group-item-success': studentStatus === 'present',
                'list-group-item-danger': studentStatus === 'absent'}">
```

Testez !

Les Pipes https://angular.io/api/core/Pipe

Commencez par créer un objet Date dans la class AppComponent :
lastUpdate = new Date();

Puis ajouter une balise paragraphe dans le html sous le titre <h2<
Ajouter le texte ‘Mise à jour le :’ et lié la variable.

Modifiez l’affichage de l’objet avec le pipe | date.
Mis à jour : {{ lastUpdate | date }}

Testez plusieurs formats : https://angular.io/api/common/DatePipe
date: 'short'
date: 'yMMMMEEEEd'

Il y a beaucoup de possibilités de formatage de DatePipe : vous trouverez plus d'informations dans la documentation d'Angular.

Utilisez une chaîne de Pipes

Par exemple : | date: 'yMMMMEEEEd' | uppercase

Async

Le pipe async permet de gérer des données asynchrones que l'application doit récupérer sur un serveur.
Pour simuler ce délai modifier lastUpdate :

```
lastUpdate = new Promise((resolve, reject) => {
    const date = new Date();
    setTimeout(
      () => {
        resolve(date);
      }, 3000
    );
  });
```

Puis ajouter dans la chaine de Pipe le Pipe async en 1er.

À ce stade, vous savez :

- comment créer les composantes d'une application Angular : les components. Vous savez leur passer des données, réagir aux événements qu'ils déclenchent et même faire les deux en même temps !
- utiliser des directives pour structurer votre application et en modifier le contenu de manière dynamique ;
- profiter des pipes pour modifier l'affichage des données sans en changer la nature.