Practico 3 @October 13, 2022

Ejercicio integrador con lo visto anteriormente y agregando Bootstrap

En este practivo veremos la creación de un componente, cómo agregarlo a nuestro módulo principal, trabajar con templates, data binding, interpolación y directivas.

1. Instalamos Node y npm

Para poder utilizar angular lo que debemos instalar es **Node.js** que es un entorno de ejecución para javascript construido con el motor de JavaScript V8 de Chrome y **NPM** o (*Node Package Manager*) es una **Command Line Utility** que nos permite interactuar, de una forma muy simple, con un repositorio enorme de proyectos *open-source*.

2. Configuramos el entorno de desarrollo

Instalamos Angular CLI `npm install -g @angular/cli`

3. Creamos nuestro proyecto

`ng new NombreDelProyecto` (si no se coloca un nombre se crea en el proyecto actual)

4. Ejecutamos nuestro proyecto

`ng serve --open` (--open habre una ventana en el navegador en `http://localhost:4200/`)

5. Instalamos Bootstrap

Instalamos la librería Bootstrap (nos da estilos y nos permite lograr diseños responsive de forma simple). Para ello, parados sobre nuestro proyecto usamos npm para descargarla (recordemos que npm es como Nuget pero para librerías o módulos de JavaScript):

```
npm install bootstrap@3 --save
```

El --save lo que hace es guardar la referencia a este módulo en el package.json

Vemos como se impacta el package.json

```
"dependencies": {
    "@angular/animations": "^6.1.0",
    "@angular/common": "^6.1.0",
    "@angular/compiler": "^6.1.0",
    "@angular/core": "^6.1.0",
    "@angular/forms": "^6.1.0",
    "@angular/http": "^6.1.0",
    "@angular/platform-browser": "^6.1.0",
    "@angular/platform-browser-dynamic": "^6.1.0",
    "@angular/router": "^6.1.0",
    "bootstrap": "^3.3.7", //aca aparecio boostrap
    "core-js": "^2.5.4",
    "rxjs": "~6.2.0",
    "zone.js": "~0.8.26"
  },
```

Debemos luego en el `angular.json` en projects/:NOMBRE DEL PROJECTO:/styles agregar lo siguiente, para poder usar boostrap en nuestra aplicacion.

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

6. Creamos nuestro componente

Para eso lanzaremos el siguiente commando `ng generate component HomeworksList` Una carpeta llamada homeworks-list, con 4 archivos:

- homeworks-list.component.spec.ts (Archivo con pruebas) (Se puede tanto eleminar como mover a la carpeta e2e)
- homeworks-list.component.ts (Archivo con la clase del componente y la metadata)

- homeworks-list.component.html (Archivo que contiene la vista)
- homeworks-list.component.css (Archivo que contiene el css del componente)

Este commando nos agrega el componente automaticamnte al `app.module.ts` dentro del array declarations

```
{
  declarations: [
    AppComponent,
    HomeworksListComponent,
    HomeworksFilterPipe,

  ],
....
```

7. Agregamos el html.

Agregamos en nuestro archivo de vista (`homeworks-list.component.html`) nuestro template basico, que contiene un panel con el titulo del programa y una tabla en la que ostraremos los datos de los homeworks:

```
<div class='panel panel-primary'>
    <div class='panel-heading'>
        Homeworks List
    </div>

    <div class='panel-body'>
        <!-- Aca filtramos las tareas  -->
        <!-- Selector de filtro:  -->
        <div class='row'>
            <div class='col-md-2'>Filter by:</div>
            <div class='col-md-4'>
                <input type='text' />
            </div>
        </div>
        <!-- Muestra filtro:  -->
        <div class='row'>
            <div class='col-md-6'>
                <h3>Filtered by: </h3>
            </div>
        </div>

        <!-- Mensaje de error -->
        <div class='has-error'> </div>

        <!--Tabla de tareas -->
        <div class='table-responsive'>
            <table class='table'>
```

```
                <!--Cabezal de la tabla -->
                <thead>
                    <tr>
                        <th>Id</th>
                        <th>Description</th>
                        <th>DueDate</th>
                        <th>Score</th>
                        <th>
                            <button class='btn btn-primary'>
                               Show Exercises
                            </button>
                        </th>
                    </tr>
                </thead>
                <!--Cuerpo de la tabla-->
                <tbody>
                    <!-- Aca va todo el contenido de la tabla  -->
                </tbody>
            </table>
        </div>
    </div>
</div>
```

8. El código del componente

Particularmente utilizaremos la propiedad `templateUrl` en nuestro componente para decirle cual es el html que debe levatnar, para esto modificamos el archivo `homeworks-list.component.ts` y le agregamos el siguiente código:

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

@Component({
  selector: 'app-homeworks-list',
  templateUrl: './homeworks-list.component.html',
  styleUrls: ['./homeworks-list.component.css']
})
export class HomeworksListComponent implements OnInit {
    pageTitle:string = "Homeworks List"

    constructor() { }

    ngOnInit() {
    }
}
```

9. Agregamos el componente nuevo a través de su selector.

Lo que haremos aquí es usar el selector `app-homeworks-list` en el root component, es decir el `app.component.ts` para asi luego poder usar el html de homeworks list `homeworks-list.component.html` en el `app.component.html`

`app.component.ts` quedaría asi:

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

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
    title:string = 'Homeworks Angular';
    name:string = "Santiago Mnedez";
    email : string = "santi17mendez@hotmail.com";
    address = {
        street: "la dirección del profe",
        city: "Montevideo",
        number: "1234"
    }
}
```

Y nuestro `app.component.html`:

```
<app-homeworks-list></app-homeworks-list>
```

Sin embargo, con esto no basta, ya que para que un componente pueda usar a otro componente (a través de su selector), estos deben pertenecer al mismo módulo, o el módulo del componente que importa debe importar al mdulo del otro componente.

En consecuencia, vamos a `app.module.ts`, y aseguraros que se encuentre el import:

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

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

¿Como hace el componente para saber a dónde buscar el component? Cómo ya dijimos, ahora lo encuentra porque pertenecen al mismo modulo. El módulo que sea dueño de este component es examinado para encontrar todas las directivas que pertenecen al mismo.

6. Usando Data Binding para mostrar datos dinmicos

Hacer cambio en el `homeworks-list.component.html` y poner:

```
<div class='panel-heading'>
    {{pageTitle}}
</div>
```

Veamos que pasa.

7. Utilizando *ngIf para elegir hacer algo o no

En el template, cambiamos `<table clas="table">` por lo siguiente:

```
<table class='table' *ngIf='homeworks && homeworks.length'>
```

Esto todava no va a tener resultado hasta que en el paso siguiente agreguemos la property 'homeworks'.

8. Utilizando *ngFor para iterar sobre elementos de forma dinamica

Ahora crearemos la clase Exercise y Homeworks (en una carpeta llamada models), y agregaremos la propiedad 'homeworks' a nuestro componente.

Clase Exercise en la carpeta models:

```
export class Exercise {
    id: string;
    problem: string;
    score: number;

    constructor(id:string = "", problem:string = "", score:number = 0) {
        this.id = id;
        this.problem = problem;
        this.score = score;
    }
}
```

Clase Homework en la carpeta models:

```
import { Exercise } from './Exercise';

export class Homework {
    id: string;
    description: string;
    dueDate: Date;
    score: number;
    exercises: Array<Exercise>;

    constructor(id:string, description:string, score:number, dueDate:Date, exercises:
 Array<Exercise>){
        this.id = id;
        this.description = description;
        this.score = score;
        this.dueDate = dueDate;
        this.exercises = exercises;
    }
}
```

Nuestro componente:

```
import { Component, OnInit } from '@angular/core';
import { Homework } from '../models/Homework';
import { Exercise } from '../models/Exercise';

@Component({
  selector: 'app-homeworks-list',
  templateUrl: './homeworks-list.component.html',
  styleUrls: ['./homeworks-list.component.css']
})
export class HomeworksListComponent implements OnInit {
    pageTitle:string = "Homeworks List";
    homeworks:Array<Homework> = [
        new Homework("1", "Una tarea", 0, new Date(), [new Exercise("1", "Un Problem
a", 0)]),
        new Homework("2", "Otra tarea", 0, new Date(), [])
    ];

    constructor() { }

    ngOnInit() {
    }
}
```

Y en el template cambiamos el `<tbody>` por lo siguiente:

```
<tbody>
  <tr *ngFor='let aHomework of homeworks'>
      <td>{{aHomework.id}}</td>
```

```
      <td>{{aHomework.description }}</td>
      <td>{{aHomework.dueDate}}</td>
      <td>{{aHomework.score}}</td>
      <td>
          <div>
              <table>
                  <thead>
                      <tr>
                          <th>Problem</th>
                          <th>Score</th>
                      </tr>
                  </thead>
                  <tbody>
                      <tr *ngFor='let aExercise of aHomework.exercises'>
                          <td>{{aExercise.problem}}</td>
                          <td>{{aExercise.score}}</td>
                      </tr>
                  </tbody>
              </table>
          </div>
      </td>
  </tr>
</tbody>
```

9. Agregando Two-Way Binding:

En nuestro HomeworksListComponent, agregamos la property listFilter:

```
text: string='';
```

En el template asociado, reemplazamos los dos primeros divs de class "row" que aparecen:

```
<div class='row'>
    <div class='col-md-2'>Text:</div>
    <div class='col-md-4'>
        <input type='text' [(ngModel)]='text' />
    </div>
</div>
<div class='row' *ngIf='text'>
    <div class='col-md-6'>
        <h3>Text: {{text}} </h3>
    </div>
</div>
```

Vemos que no nos anda.

Para ello vamos al `app.module.ts` y agregamos el import a FormsModule:

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

import { AppComponent } from './app.component';
import { HomeworksListComponent } from './homeworks-list/homeworks-list.component';

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

10. Usando Pipes en Angular

Para ello, simplemente cambiamos:

```
<td>{{aHomework.description  | uppercase}}</td>
... // o
<td>{{aHomework.description  | lowercase}}</td>
```

11. Agregando Event Binding para los Exercises

Lo que haremos ahora es la lógica del mostrado de imagenes con Event Binding, para ello: En `homeworks-list.component.ts`, agregamos la siguiente property a la clase:

```
showExercises:boolean = false;
```

A su vez agregamos la siguiente función:

```
toggleExercises(): void {
    this.showExercises = !this.showExercises;
}
```

Quedando:

```
import { Component, OnInit } from '@angular/core';
import { Homework } from '../models/Homework';
import { Exercise } from '../models/Exercise';

@Component({
  selector: 'app-homeworks-list',
  templateUrl: './homeworks-list.component.html',
  styleUrls: ['./homeworks-list.component.css']
})
export class HomeworksListComponent implements OnInit {
    pageTitle:string = "Homeworks List";
    listFilter:string = "";
    showExercises:boolean = false;
    homeworks:Array<Homework> = [
        new Homework("1", "Una tarea", 0, new Date(), [new Exercise("1", "Un Problem
a", 0)]),
        new Homework("2", "Otra tarea", 0, new Date(), [])
    ];

    constructor() { }

    ngOnInit() {
    }

    toggleExercises(): void {
        this.showExercises = !this.showExercises;
    }

}
```

Y en el template hacemos estos dos cambios:

1. En cada click al botón que tenemos en el header de la tabla, llamamos a la función `toggleExercises()`:

```
<button (click)='toggleExercises()'class='btn btn-primary'>
    {{showExercises ? 'Hide' : 'Show'}} Exercises
</button>
```

1. En el mostrado de los Exercises, agregamos la condición de que solo se muestre si la property lo indica.

```
<div *ngIf='showExercises'>
    <table>
        <thead>
            <tr>
                <th>Problem</th>
```

```
                <th>Score</th>
            </tr>
        </thead>
        <tbody>
            <tr *ngFor='let aExercise of aHomework.exercises'>
                <td>{{aExercise.problem}}</td>
                <td>{{aExercise.score}}</td>
            </tr>
        </tbody>
    </table>
</div>
```