Práctica Angular – Parte 2

Tic-Tac-Toe

Objetivos

Mediante Angular CLI crear la estructura:

- Crear el proyecto, cuyo nombre será “tictactoe”.
- Crear GameModule y dentro de este crear:
  - GameComponent
  - HeaderComponent
  - BoardComponent
  - SquareComponent

Proceso a seguir

1.1 Creación del proyecto

Comenzamos con la creación del proyecto. Desde un terminal y situados en nuestra carpetas de prácticas, ejecutamos el asistente de Angular CLI para crear un nuevo proyecto (sin routing y con CSS):

```
>ng new tictactoe
```

Una vez finalizado el asistente ya podemos abrir el proyecto con nuestro IDE preferido y comprobar que la aplicación se despliega correctamente en http://localhost:4200/

Comprobado el correcto despliegue de la aplicación, mejor si detenemos en estos momentos el *server*, puesto que seguidamente vamos a realizar muchos cambios que impedirán temporalmente el buen funcionamiento de la aplicación.

1.2 Creación de GameModule y de GameComponent

Para crear estos elementos, nos situamos dentro de la carpeta “tictactoe” y ejecutamos los comandos necesarios:

```
>cd tictactoe
>ng g m game
```

Con lo anterior ya hemos creado el elemento GameModule (el ficheros game.module.ts)

Ahora procedemos a crear el elemento GameComponent (dentro de GameModule):

```
>ng g c game/game
```

Lo anterior ha creado cuatro ficheros dentro de la carpeta src → app → game → game (notad que tenemos dos directorios “game” anidados, el primero pertenece al módulo, GameModule, y el segundo al componente, GameComponent):

- game.component.css
- game.component.html
- game.component.spec.ts
- game.component.ts

Y también ha actualizado GameModule para contemplar en este módulo el nuevo componente.

src → app → game → game.module.ts

```
import { NgModule } from '@angular/core';
import { CommonModule } from '@angular/common';
import { GameComponent } from './game/game.component';

@NgModule({
  declarations: [GameComponent],
  imports: [ CommonModule ]
})
export class GameModule { }
```

Ahora nos queda por hacer algunas cosas manualmente:

Primero: Importar GameModule en AppModule para que el módulo raíz tenga registrado este nuevo módulo.

src → app → app.module.ts

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

import { AppComponent } from './app.component';

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

Segundo: Exportar GameComponent en GameModule, de esta manera GameComponent estará disponible para AppComponent (Queremos una relación jerárquica entre ellos).

src → app → game → game.module.ts

```
import { NgModule } from '@angular/core';
import { CommonModule } from '@angular/common';
import { GameComponent } from './game/game.component';

@NgModule({
  declarations: [GameComponent],
  imports: [
    CommonModule
  ],
```

```
  exports: [ GameComponent ]
})
export class GameModule { }
```

Tercero: En la plantilla de AppComponent eliminaremos bastante del código HTML creado por el asistente de Angular CLI e incluiremos una instancia de GameComponent, es decir, el tag asociado a GameComponent. Con esto definimos la jerarquía padre-hijo entre AppComponent y GameComponent.

Por tanto, nos fijamos cuál es el tag de GameComponent:

src → app → game → game → game.component.ts

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

@Component({
  selector: 'app-game',
  templateUrl: './game.component.html',
  styleUrls: ['./game.component.css']
})
export class GameComponent implements OnInit {
  ...
}
```

Y nos dirigimos a la plantilla de AppComponent:

src → app → app.component.html

```
<!--The content below is only a placeholder and can be replaced.-->
<div style="text-align:center">
  <h1>
    Welcome to {{ title }}!
  </h1>
  <img width="300" alt="Angular Logo"
src="
C9zdmciIHZpZXdCb3g9IjAgMCAyNTAgMjUwIj4KICAgIDxwYXRoIGZpbGw9IiNERDAwMzEiIGQ9I
k0xMjUgMzBMMzEuOSA2My4ybDE0LjIgMTIzLjFMTI1IDIzMGw3OC45LTQzLjcgMTQuMi0xMjMuM
XoiIC8+CiAgICA8cGF0aCBmaWxsPSIjQzMwMDJGIiBkPSJNMTI1IDMwdjIyLjItLjFWFWMjMwbDc4L
jktNDMuNyAxNC4yLTEyMy4xTDEyNSAzMHoiIC8+CiAgICA8cGF0aCAgZmlsbD0iI0ZGRkZGRiIgZ
```

```
D0iTTEyNSA1Mi4xTDY2LjggMTgyLjZoMjEuN2wxMS43LTI5LjJoNDkuNGwxMS43IDI5LjJIMTgzT
DEyNSA1Mi4xem0xNyA4My4zaC0zNGwxNy00MC45IDE3IDQwLjl6IiAvPgogIDwvc3ZnPg==">
</div>
<h2>Here are some links to help you start: </h2>
<ul>
  <li>
    <h2><a target="_blank" rel="noopener"
href="https://angular.io/tutorial">Tour of Heroes</a></h2>
  </li>
  <li>
    <h2><a target="_blank" rel="noopener" href="https://angular.io/cli">CLI
Documentation</a></h2>
  </li>
  <li>
    <h2><a target="_blank" rel="noopener"
href="https://blog.angular.io/">Angular blog</a></h2>
  </li>
</ul>
```

Ahora eliminamos el contenido de la plantilla desde el <h2> hasta el final y antes de cerrar el añadimos el tag asociado a GameComponent:

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

Ponemos en marchar el *server* y una vez esté listo deberíamos ver lo siguiente en el navegador:

Welcome to tictactoe!

game works!

1.3 Creación de HeaderComponent y de BoardComponent

Ahora procedemos a crear el elemento HeaderComponent (dentro de GameModule):

```
>ng g c game/header
```

Lo anterior ha creado cuatro ficheros dentro de la carpeta src → app → game → game:

- header.component.css
- header.component.html
- header.component.spec.ts
- header.component.ts

Y también ha actualizado GameModule para contemplar en este módulo el nuevo componente.

src → app → game → game.module.ts

```
import { NgModule } from '@angular/core';
import { CommonModule } from '@angular/common';
import { GameComponent } from './game/game.component';
import { HeaderComponent } from './game/header.component';

@NgModule({
  declarations: [GameComponent, HeaderComponent],
  imports: [
    CommonModule
  ]
})
export class GameModule { }
```

Seguidamente, pasamos a crear el elemento BoardComponent (dentro de GameModule):

```
>ng g c game/board
```

Lo anterior ha creado cuatro ficheros dentro de la carpeta src → app → game → game:

- board.component.css
- board.component.html
- board.component.spec.ts
- board.component.ts

Y también ha actualizado GameModule para contemplar en este módulo el nuevo componente.

src → app → game → game.module.ts

```
import { NgModule } from '@angular/core';
import { CommonModule } from '@angular/common';
import { GameComponent } from './game/game.component';
import { HeaderComponent } from './game/header.component';
import { BoardComponent } from './game/board.component';

@NgModule({
  declarations: [GameComponent, HeaderComponent, BoardComponent],
  imports: [
    CommonModule
  ]
})
export class GameModule { }
```

Ahora, incluiremos una instancia de HeaderComponent y otra de BoardComponent en la plantilla de GameComponent, creando así una jerarquía padre hijo entre ellos. Borramos el HTML original de este fichero y escribimos lo siguiente:

src → app → game → game → game.component.html

```
<app-header></app-header>
<app-board></app-board>
```

Por el momento, haremos que HeaderComponent muestre un texto fijo. No haremos por ahora con BoardComponent:

src → app → game → header → header.component.html

```
<p>
  Turn of Player 1 - Xs
</p>
```

Ponemos en marchar el *server* y una vez esté listo deberíamos ver lo siguiente en el navegador:

Welcome to tictactoe!

Turn of Player 1 - Xs

board works!

1.4 Creación de SquareComponent

Ahora procedemos a crear el elemento SquareComponent (dentro de GameModule), el cual, recordemos, representa una casilla del tablero y ha de mostrar el valor por defecto "-", además de detectar pulsaciones del usuario:

```
>ng g c game/square
```

Lo anterior ha creado cuatro ficheros dentro de la carpeta src → app → game → game:

- square.component.css
- square.component.html
- square.component.spec.ts
- square.component.ts

Y también ha actualizado GameModule para contemplar en este módulo el nuevo componente.

src → app → game → game.module.ts

```
import { NgModule } from '@angular/core';
import { CommonModule } from '@angular/common';
import { GameComponent } from './game/game.component';
import { HeaderComponent } from './game/header.component';
import { BoardComponent } from './game/board.component';
import { SquareComponent } from './game/square.component';

@NgModule({
       declarations:    [GameComponent,    HeaderComponent,    BoardComponent,
SquareComponent],
  imports: [
```

```
    CommonModule
  ]
})
export class GameModule { }
```

Ahora vamos a la plantilla de SquareComponent para especificar la lógica de presentación. Reemplazamos su contenido por el siguiente:

src → app → game → square → square.component.html

```
<button (click)="handleSquareClick()"> - </button>
```

Con lo anterior estamos indicando que se pinte un botón y que cuando se presione se invoque el método handleSquareClick del fichero square.component.ts (que aún no existe).

A continuación, nos dirigimos al fichero de clase de SquareComponent para proporcionar una implementación, dummy por el momento, para el método handleSquareClick:

src → app → game → square → square.component.ts

```
...
export class SquareComponent implements OnInit {

  constructor() { }

  ngOnInit() {
  }

  handleSquareClick() {
    console.log("Square click");
  }

}
```

Otra cosa que tenemos que hacer en este fichero es definir propiedades que nos permitan identificar una celda en particular. Una celda es la intersección de una fila y una columna, por tanto, vamos a definir dos propiedades: *row* y *col*:

```
...
export class SquareComponent implements OnInit {
  row: number;
  col: number;
...
```

Además, hemos de tener en cuenta algo muy importante: es el componente BoardComponent quien en el momento de pintar cada una de las nueve casillas se encargará de pasar el índice de fila y de columna correspondiente, por lo cual las dos propiedades anteriores son de entrada, esto es, deben estar precedidas por el decorador @Input():

```
...
import {Component, Input, OnInit} from '@angular/core';
...
export class SquareComponent implements OnInit {
  @Input() row: number;
  @Input() col: number;
...
```

Como sabemos, necesitamos mostrar nueve casillas en el tablero. Esto lo haremos en la plantilla de BoardComponent, mostrando el carácter “-”, “X” o “0” que le toque a cada casilla, para lo cual se necesitará tener acceso al componente que sostiene el estado de la aplicación, es decir, el servicio StateService, el cual aún no existe. Por tanto, temporalmente, vamos a crear una implementación dummy en el fichero de clase de BoardComponent (más adelante crearemos el servicio real):

src → app → game → board → board.component.ts

```
...
export class BoardComponent implements OnInit {

  private valores: string[][] = [
    ['-','-','-'],
    ['-','-','-'],
    ['-','-','-']
  ];
...
}
```

Ahora en la plantilla de BoardComponent, borramos el contenido existente y escribimos el siguiente, basado en la directiva estructural *ngFor para pintar el tablero:

src → app → game → board → board.component.html

```
<div *ngFor="let fila of valores; index as i">
  <app-square *ngFor="let columna of fila; index as j"
      [row]="i" [col]="j">
  </app-square>
</div>
```

Notad que con la sintaxis de los corchetes el componente padre (BoardComponent) pasa los valores de i (fila actual) y j (columna actual) a las propiedades de entrada (row, col) del componente hijo (SquareComponent).

Levantemos el servidor y comprobemos que se muestra lo siguiente en el navegador:

Si abrimos la consola del navegador, veremos que al pulsar sobre alguna casilla se escribe un mensaje. Esto indica que se está llamando correctamente a la función handleSquareClick de SquareComponent:

Vamos a mejorar la implementación del método handleSquareClick para que muestre correctamente la fila y columna pulsada:

src → app → game → square → square.component.ts

```
...
handleSquareClick() {
    console.log("Square click", this.row, this.col);
}
```

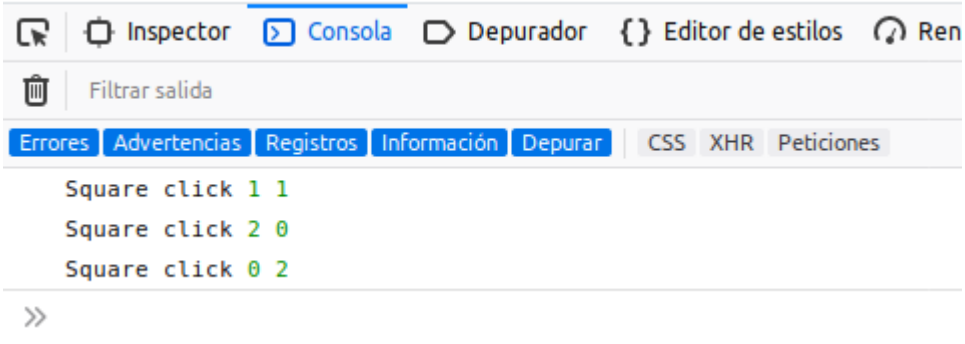
Por otro lado, vamos a mejorar el aspecto del tablero creando algunos estilos CSS para SquareComponent:

src → app → game → square → square.component.css

```
button { height: 100px; width: 100px; }
```

Comprobemos que se muestra lo siguiente en el navegador:

Si abrimos la consola del navegador, veremos que al pulsar sobre alguna casilla se escribe un mensaje, pero ahora aparecen correctamente las coordenadas de la celda pulsada:



Conclusión

En esta parte de la práctica hemos creado el proyecto, los módulos y componentes necesarios. Hemos implementando los componentes, aunque en algunos casos con código dummy. También hemos creado código en algunas plantillas y hemos establecido las relaciones jerárquicas entre componentes.

En la siguiente parte de la práctica implementaremos el componente que mantendrá el estado de la aplicación, StateService, con lo que eliminaremos el código dummy y tendremos realizada prácticamente toda la funcionalidad necesaria.