Práctica Angular – Parte 3

Tic-Tac-Toe

Objetivos

Implementar el estado de la aplicación:

- Crear e implementar el servicio StateService dentro de GameModule
- Proporcionar el servicio a GameModule
- Inyectar StateService en HeaderComponent, BoardComponent y SquareComponent
- Modificar SquareComponent para evitar clics cuando no se deba

Proceso a seguir

1.1 Implementación del servicio

Esto lo haremos mediante Angular CLI. Queremos que el servicio se llame StateService y que resida dentro del módulo GameModule, por lo que hacemos lo siguiente:

```
>ng g s game/state
```

Lo anterior nos ha generado dos ficheros dentro del módulo game:

- state.service.spec.ts
- state.service.ts

No obstante, notad que el servicio no se encuentra dentro de ninguna carpeta específica, como sí sucede con los componentes.

A continuación vamos a implementar el servicio.

Lo primero a tener en cuenta es que necesitaremos una interfaz que denominaremos “State”, que definirá el turno del jugador y la situación en tiempo real del tablero:

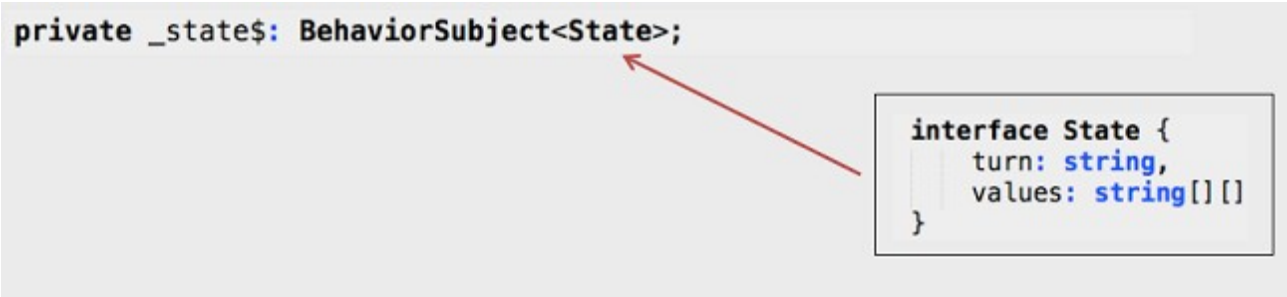
src → app → game → state.service.ts

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

export interface State {
  turno: string,
  valores: string[][]
}

@Injectable({
  providedIn: 'root'
})
export class StateService {
  constructor() { }
}
```

Lo siguiente que necesitamos es una propiedad, que llamaremos "_state$", cuyo tipo será BehaviorSubject<State>. Se trata de un componente de Angular que implementa el patrón Observer, en este caso sobre la interfaz "State". Gracias a esta propiedad, los componentes que lo necesiten podrán suscribirse al estado de la aplicación para recibir notificaciones de cambios o incluso para poder modificar el estado:




```
import { BehaviorSubject } from 'rxjs';
...
export class StateService {
  private _state$: BehaviorSubject<State>;
  constructor() { }
}
```

Ahora vamos a definir el constructor del servicio, en el que inicializamos la propiedad "estado" con los valores por defecto:

```
...
export class StateService {
  private _state$: BehaviorSubject<State>;
  constructor() {
    this._state$ = new BehaviorSubject({
        turno: 'PLAYERX',
         valores:[
            ['-','-','-'],
            ['-','-','-'],
            ['-','-','-']
         ]
    });
  }
}
```

A continuación, definiremos otros métodos para el servicio:

Este primer método permitirá a los componentes suscribirse a los cambios en el estado:

```
get state$(): BehaviorSubject<State> {
    return this._state$;
}
```

Estos otros dos métodos nos permitirán leer y modificar el estado:

```
get state(): State {
    return this._state$.getValue();
}

set state(state: State) {
    this._state$.next(state);
}
```

Por otro lado, tendremos un método para actualizar actualizar una casilla concreta del tablero:

```
updateValue(row, col) {
    if(this.state.valores[row][col] === '-') {
        let newValue = this.state.turno === 'PLAYERX' ? 'X':'0';
```

```
        let newTurn = this.state.turno === 'PLAYERX' ? 'PLAYER0':'PLAYERX';
        this.state.valores[row][col] = newValue;
        this.state.turno = newTurn;
    }
}
```

Y finalmente, tendremos un método para resetear el juego:

```
reset() {
    this.state = {
        turno: 'PLAYERX',
         valores:[
            ['-','-','-'],
            ['-','-','-'],
            ['-','-','-']
        ]
    };
}
```

En este momento ya tenemos lista la implementación de este servicio.

1.2 Proporcionar el servicio a GameModule

Para que los componentes de GameModule puedan tener acceso al servicio anterior, necesitamos informar de la existencia del servicio al módulo al que corresponden tales componentes.

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 './header/header.component';
import { BoardComponent } from './board/board.component';
import { SquareComponent } from './square/square.component';
import { StateService } from './state.service';

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

```
  imports: [
    CommonModule
  ],
  providers: [ StateService ],
  exports: [ GameComponent ]
})
export class GameModule { }
```

1.3 Inyectar el servicio en los componentes requeridos

Comenzamos por sustituir la implementación dummy de BoardComponent por una implementación adecuada que utilice el servicio creado anteriormente. Recordad que esta clase tenía “hardcoded” una matriz con los valores del tablero, en cambio ahora esta matriz le vendrá proporcionada por el servicio:

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

```
import { Component, OnInit } from '@angular/core';
import { StateService } from '../state.service';

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

  private valores: string[][];

  constructor(stateService: StateService) {
    this.valores = stateService.state.valores;
  }

  ngOnInit() {
  }
}
```

Ahora vamos a modificar SquareComponent. Necesitamos definir una propiedad para tener acceso al servicio que maneja el estado de la aplicación y así poder modificar el turno y el tablero cuando sea necesario:

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

```
import {Component, Input, OnInit} from '@angular/core';
import { StateService } from '../state.service';

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

  @Input() row: number;
  @Input() col: number;

  private _stateService: StateService;

  constructor(stateService: StateService) {
    this._stateService = stateService;
  }

  ngOnInit() {
  }

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

}
```

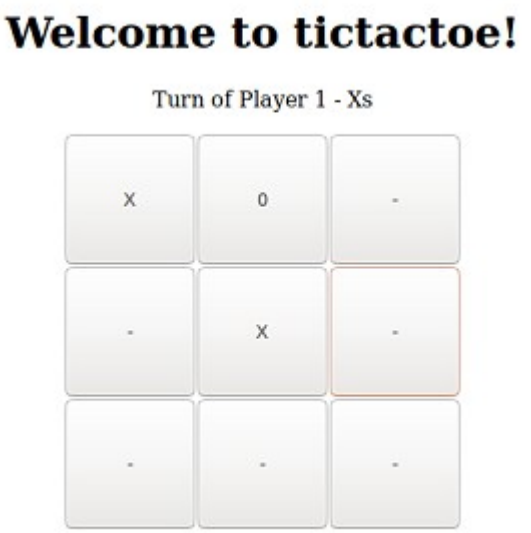
A continuación, hemos de modificar la plantilla de SquareComponent para poder mostrar el valor real que le corresponda a cada celda del tablero. Usaremos interpolación para realizar el binding y así la plantilla quede suscrita a los cambios en el estado del observable, esto es, de nuestro servicio:

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

```
<button (click)="handleSquareClick()"> {{ (_stateService.state$ | async).valores[row][col] }} </button>
```

Notad que con el pipe async siempre obtendremos el último valor del estado de la aplicación (el valor más reciente).

Ahora vamos a comprobar que la aplicación se despliega correctamente en http://localhost:4200/



Al pulsar cada celda se debe ir mostrando alternativamente el símbolo “X” y “0”.

A continuación vamos a modificar HeaderComponent para mostrar el turno correctamente. Para ello hemos de definir una propiedad que tenga acceso al servicio StateService:

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

```
import { Component, OnInit } from '@angular/core';
import { StateService } from '../state.service';

@Component({
  selector: 'app-header',
  templateUrl: './header.component.html',
  styleUrls: ['./header.component.css']
})
```

```
export class HeaderComponent implements OnInit {

  private _stateService: StateService;

  constructor(stateService: StateService) {
    this._stateService = stateService;
  }

  ngOnInit() {
  }

}
```

Ahora hemos de modificar la plantilla de HeaderComponent:

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

```
<p>
  Turn of {{ (_stateService.state$ | async).turno }}
</p>
```

Realizado el cambio anterior, verificamos en el navegador que efectivamente el mensaje del turno va conmutando en cada jugada.

1.4 Evitar pulsaciones indebidas en el tablero

Nos queda evitar que un usuario pueda pulsar sobre una casilla que ya tenga una ficha depositada previamente. Esto lo controlaremos en la plantilla de SquareComponent mediante CSS. Lo que queremos hacer es cambiar el aspecto del cursor del ratón para indicarle al usuario que no puede volver a pulsar sobre esa celda:

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

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

.clickable {
```

```
  cursor: pointer;
}

.not-clickable {
  cursor: not-allowed;
}
```

Ahora volvemos al fichero HTML de SquareComponent y modificamos el código como sigue:

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

```
<button
[class.clickable]="_stateService.state.valores[row][col] === '-'"
[class.not-clickable]="_stateService.state.valores[row][col] !== '-'"
(click)="handleSquareClick()">
{{ (_stateService.state$ | async).valores[row][col] }}
</button>
```

Realizado el cambio anterior, verificamos en el navegador que efectivamente el icono del cursor del mouse muestra una mano al posicionarnos sobre una celda libre, mientras que si la celda está ocupada se muestra el icono de prohibición.

Conclusión

Hasta aquí la práctica tutorizada del juego de las tres en raya. En la siguiente parte de la práctica se piden tres ejercicios para que la aplicación sea jugable.