Práctica Angular – Parte 7

Tic-Tac-Toe

Objetivos

- De manera tutorizada extender la anterior versión del juego de las tres en raya.
- Posteriormente, sin ayuda, el alumno deberá realizar una serie de actividades con tal de ampliar aún más la funcionalidad de la aplicación.

Introducción

Para poner en práctica los conceptos vistos en el tutorial anterior, vamos a ampliar la funcionalidad de la aplicación de las tres en raya:

- Implementaremos rutas, para lo cual:
  - Incluiremos la posibilidad de navegar (mediante pestañas):
    - Página de bienvenida, para iniciar una partida, para continuar con una partida previa
- Crearemos una conexión al backend mediante un servicio
  - Recuperaremos del servidor partidas anteriores para poder continuarlas
- Crearemos un formulario mínimo para que el jugador introduzca su nombre
  - El formulario usará la característica del doble *binding*
- Mejoras estéticas

Pasos a seguir

Organizaremos el trabajo de la siguiente manera:

1. Creación de rutas
   - Carga *fake* (hardcoded) de una partida previa
2. Conexión con el *backend* para obtener una partida previa real
3. Creación del formulario para el nombre del jugador
4. Mejora del aspecto de la aplicación mediante estilos

Paso 2. Creación de la conexión con el backend

Este paso se compone de las siguientes fases:

- Importar **HttpClientModule** en AppModule.
- Crear un servicio que llamaremos **MyHttpService** para encapsular los detalles de la conexión Http.
  - Implementaremos el método **get()**, pues solamente recuperaremos datos, no grabaremos.
  - Dado que no tenemos un *backend* usaremos una utilidad *online* de terceros (más sobre esto después)
- Modificar GameComponent para recuperar la partida salvada.
- Supervisar el estado de la petición para controlar posibles errores

Importar HttpClientModule en AppModule

Debemos informar al módulo raíz de la aplicación que vamos a usar el servicio HTTP. Esto implica añadir HttpClientModule a la lista de módulos a importar:

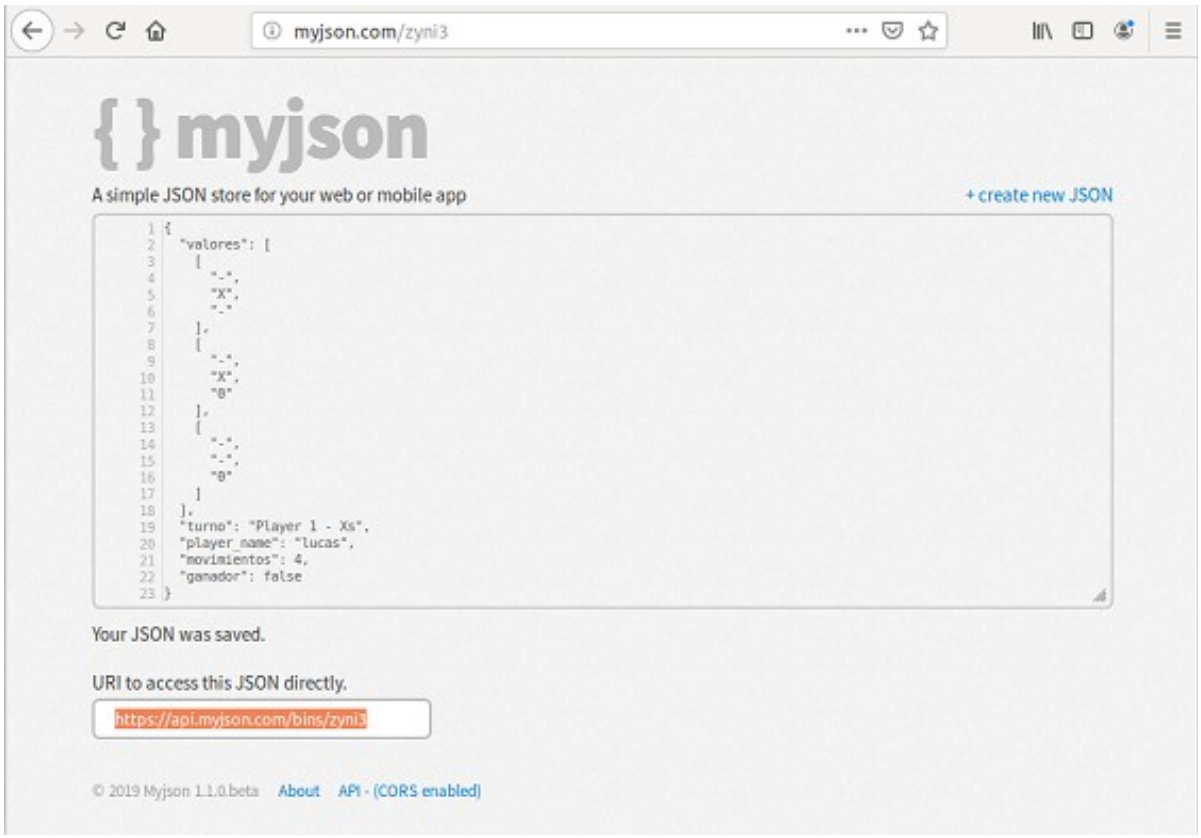
app.module.ts

```
import { BrowserModule } from '@angular/platform-browser';
...
import { HttpClientModule } from '@angular/common/http';

import { AppComponent } from './app.component';
import { IndexComponent } from './index/index.component';
import { GameComponent } from './game/game/game.component';
...

@NgModule({
  declarations: [
    AppComponent,
    IndexComponent
  ],
  imports: [
    BrowserModule,
    GameModule,
    RouterModule.forRoot(appRoutes),
    HttpClientModule
```

```
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

Creación del servicio

Dado que no tenemos un *backend* usaremos una utilidad *online* de terceros que permite guardar un JSON y luego poder recuperarlo cuantas veces necesitemos (muy útil para la fase de desarrollo de nuestras aplicaciones).



Para recuperar la partida guardada utilizaremos la URL https://api.myjson.com/bins/zyni3 aunque si lo preferís podéis crear vuestro JSON y obtener otra URL, es indiferente.

El JSON que recuperaremos es el siguiente:

```
{"valores":[["-","X","-"],["-","X","0"],["-","-","0"]],"turno":"Player  1  -
Xs","player_name":"lucas", "movimientos": 4, "ganador": false}
```

Ahora pasamos a generar el servicio cliente. Mediante Angular CLI creamos el servicio mediante el que encapsularemos la lógica HTTP. De esta forma sabemos que los componentes que requieran este servicio se mantendrán agnósticos de los pormenores del HTTP:

```
ng g s myhttp
```

Una vez que finalice el asistente procedemos a la implementación del servicio:

myhttp.service.ts

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

@Injectable({  providedIn: 'root'})
export class MyhttpService {
  readonly baseURL = "https://api.myjson.com/bins/zyni3";
  constructor(private httpClient: HttpClient) { }
  getSavedGame() {
    return this.httpClient.get(this.baseURL);
  }
}
```

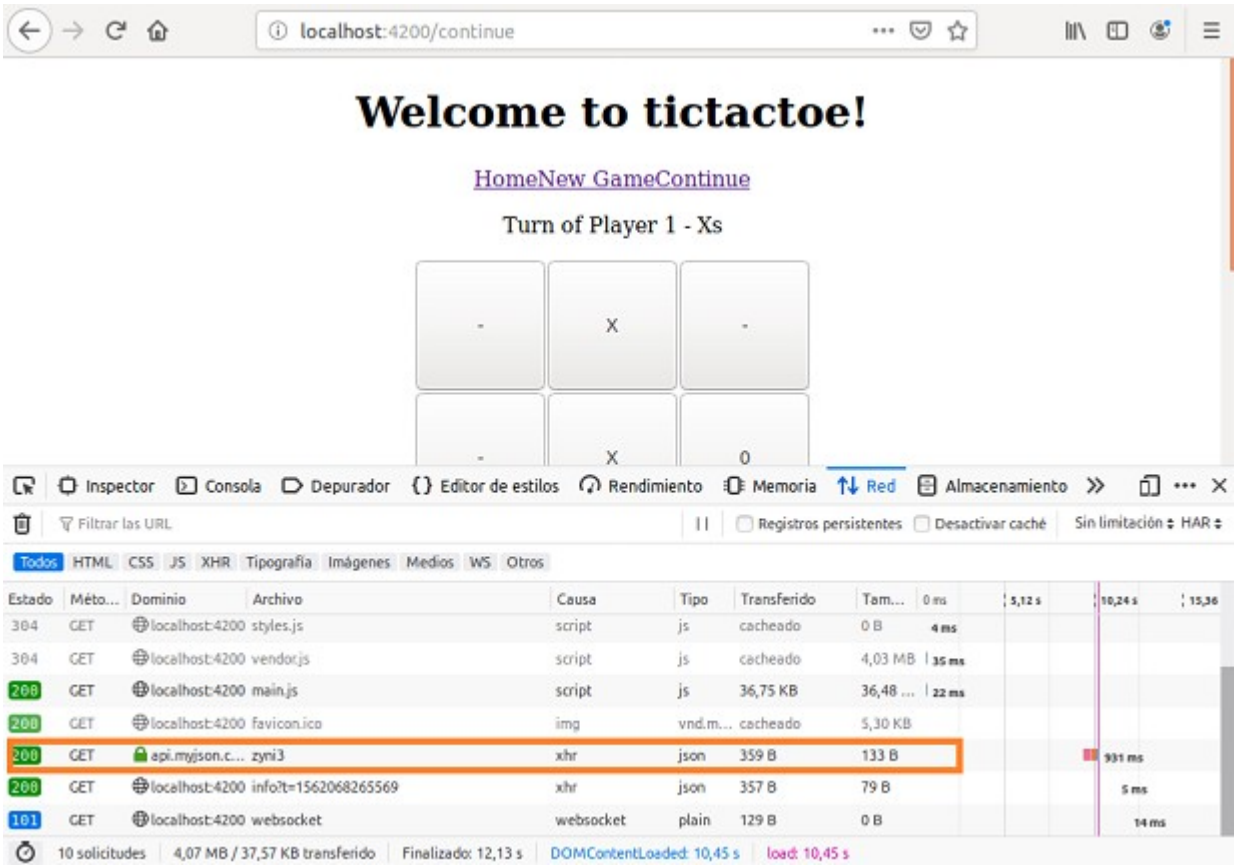
Modificar GameComponent para recuperar los datos del *backend*

Una de las cosas que debe hacer este componente es usar el servicio anterior para obtener la partida guardada previamente en el *backend*. Los datos recibidos siguen un formato JSON que se adapta a la interfaz **State**, por lo que se usan directamente para actualizar el estado de la aplicación.

```
import { Component, OnInit } from '@angular/core';
import {State, StateService} from '../state.service';
import {ActivatedRoute} from '@angular/router';
import { MyhttpService } from '../../myhttp.service';

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

```
export class GameComponent implements OnInit {
  private _stateService: StateService;
       constructor(route:   ActivatedRoute,   stateService:   StateService,
myhttpService: MyhttpService) {
    this._stateService = stateService;
    if (route.snapshot.data.continue) {
      // Partida recupearada del backend
      myhttpService.getSavedGame().subscribe( (state: State) => {
        stateService.state = state;
      });
    } else {
      stateService.reset();
    }
  }
  ...
}
```

Una vez realizado lo anterior, podremos comprobar que al seleccionar “Continue” la partida guardada se obtiene del servidor:

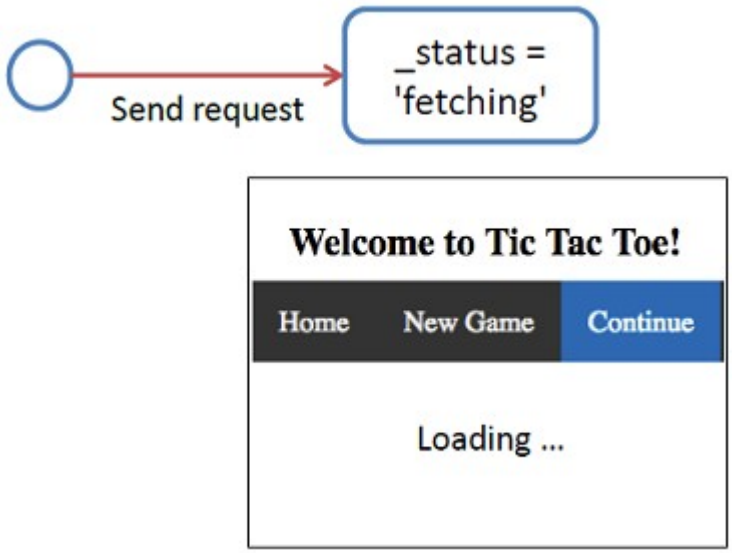


Supervisar el estado de la petición para controlar errores

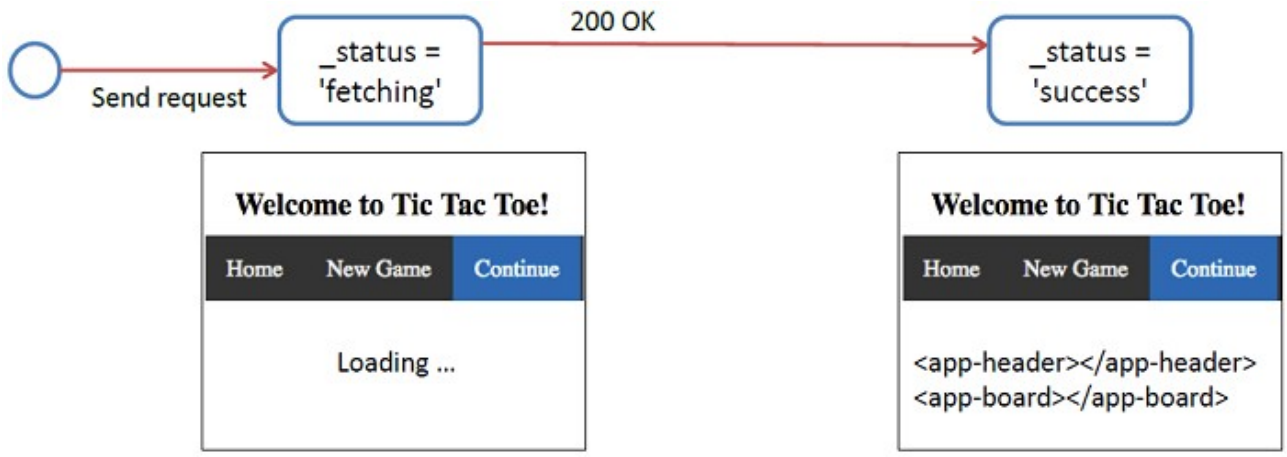
Por el momento tenemos la funcionalidad básica que nos habíamos propuesto. No obstante, faltaría controlar los posibles errores que se puedan producir al intentar comunicarse con el *backend* para recuperar los datos.

Una manera de controlar problemas es incluir en GameComponent una variable para almacenar el estado de la petición. Podemos llamar '_status' a esta variable.

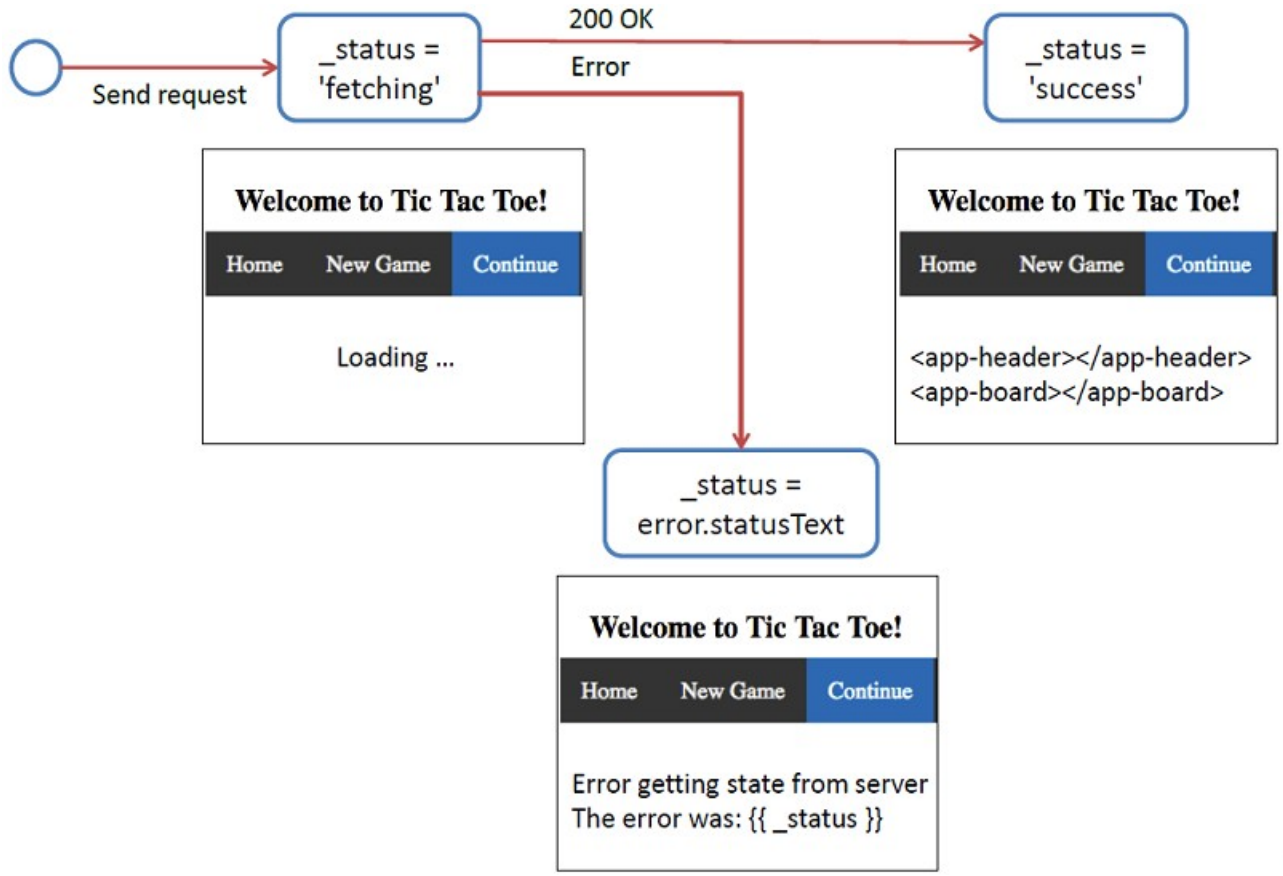
Si la petición está en curso le asignaremos el valor 'fetching' y aprovecharemos para mostrarle al usuario el mensaje 'Loading...'



Si el servidor responde con un código 200 es que todo ha ido bien, por lo que le asignaremos a '_status' el valor 'success' y mostraremos el tablero con la partida procedente del servidor:



En cambio, si el servidor responde con un error le asignaremos a '_status' el valor devuelto por el servidor y tal mensaje será el que le mostraremos al usuario:



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 {
  private _stateService: StateService;
  private _status = 'fetching';
      constructor(route:   ActivatedRoute,   stateService:   StateService,
myhttpService: MyhttpService) {
    this._stateService = stateService;
    if (route.snapshot.data.continue) {
      // Partida recupearada del backend
```

```
      myhttpService.getSavedGame().subscribe( (state: State) => {
        stateService.state = state;
        this._status = 'success';
      }, error => {
            this._status = error.statusText;
      });
    } else {
      stateService.reset();
      this._status = 'success';
    }
  }
 ...
}
```

Finalmente, hemos de modificar la plantilla de GameComponent para controlar qué se debe mostrar en función del valor de la variable '_status':

```
<div *ngIf="_status == 'fetching'">Loading...</div>
<div *ngIf="_status == 'success'">
  <app-header></app-header>
  <app-board></app-board>
  <app-footer></app-footer>
  <button (click)="handleResetButton()">Reset</button>
</div>
<div *ngIf="_status != 'success' && _status != 'fetching'">
  There is an error in the backend communication. Error: {{ _status }}
</div>
```

Ahora podemos comprobar qué todo funciona como se espera.

Para simular un fallo en la comunicación con el *backend*, podemos modificar la URL para apuntar a una dirección errónea, por ejemplo añadiendo un 0 al final de la URL. Obtendremos algo similar lo siguiente:

Conclusión

Hemos implementado un servicio para poder recuperar del *backend* una partida previamente existente. En la siguiente parte de la práctica implementaremos un formulario para que el usuario introduzca su nombre y haremos mejoras estéticas en la aplicación.