CURSO DE PROGRAMACIÓN JAVA FULLSTACK

Angular

4. Ejercicio

Arturo Bernal Mayordomo

Edición: Noviembre 2019

Semana 4

Curso de Angular

Ejercicio semana 4

Desarrollo de aplicaciones web con Angular
Cefire 2019
Autor: Arturo Bernal Mayordomo

Index

1. - Ejercicio

En este ejercicio, vamos a adaptar el proyecto para trabajar con servicios web y que los cambios sean permanentes. Para ello, vamos a adaptar tanto el servicio EventosService como los componentes afectados.

La URL base donde están alojados los servicios web es:

http://arturober.com/angular-semana4

A partir de ahora cuando hablemos de un servicio, por ejemplo '/events', la ruta completa será concatenando la URL base al servicio:

http://arturober.com/angular-semana4/events

Si quieres puedes crear un archivo de constantes en src/app (**app.constants.ts** por ejemplo) y guardar ahí la ruta:

```
export const SERVICES = 'http://arturober.com/angular-semana4';
```

No hace falta usar la función catchError para este ejercicio. Si se produce un error, trátalo directamente en la segunda función del método **subscribe**. Con mostrarlo por la consola de error → console.error(error), es suficiente.

Importante: debemos actualizar la interfaz IEvento para reflejar que los eventos ahora tendrán un campo id (clave primaria en la base de datos del servidor):

```
export interface IEvento {
   id?: number;
   title: string;
   image: string;
   date: string;
   description: string;
   price: number;
}
```

Las propiedades con ? detrás implican que son opcionales (se accede a ellas sin el interrogante). Ya que, por ejemplo, cuando creamos un objeto IEvento para añadir en el formulario, este aún no tendrá id asignada. Por eso nos interesa que al crear un objeto IEvento, TypeScript no nos obligue a establecer el campo id.

Las interfaces para las respuestas del servidor las puedes crear por ejemplo, dentro de un archivo llamado **responses.ts** en la carpeta **interfaces** (o si prefieres, organízalo de otra forma).

1.1 *Obtener eventos*

Los eventos se obtienen llamando al servicio **'/events'** por **GET**. Este servicio te devuelve un objeto JSON como el que se describe con esta interfaz:

```
export interface EventosResponse {
  ok: boolean;
  events?: IEvento[];
  error?: string;
}
```

Es decir, si el servidor obtiene los eventos correctamente devolverá la propiedad **ok** → **true** y la propiedad **events** (array con los eventos), mientras que si hay algún error devuelve **ok** → **false** y la propiedad **error** con el mensaje de error.

No te preocupes si se devuelven más campos de los que necesitas en los objetos que representan a un evento, ignóralos. Es posible que en el futuro sí que aprovechemos alguno de ellos.

El método **getEventos** del servicio, ya no devolverá directamente IEvento[], sino:

```
getEventos(): Observable<IEvent[]> {
    ...
}
```

En el método **ngOnInit** del componente **eventos-show**, suscríbete al servicio para obtener el array de eventos. Inicialízalo previamente como array vacío para que no de problemas la directiva *ngFor.

1.2 *Añadir un evento*

Añadimos un evento llamado al servicio **'/events/'** usando **POST**. Como datos enviamos el objeto IEvento con los datos del evento (el objeto que generamos con el formulario). Este servicio nos devolverá un JSON como el descrito en esta interfaz:

```
export interface EventoResponse {
  ok: boolean;
  event: IEvento;
  error?: string;
  errors?: string[];
}
```

Si el campo ok es true, nos devolverá el evento guardado en la base de datos (**event**) con la id asignada y la ruta de la imagen. Si es false, nos devolverá un array de string (**errors**) con los errores de validación. En el método **map**, si detectamos que ok es false, podemos lanzar el error con **throw respuesta.errors**, y si no, devolvemos **respuesta.event**. El método a crear en **EventosService** es:

```
addEvent(event: IEvento): Observable<IEvento> {
  ...
}
```

El método de añadir evento del componente **evento-add**, debe llamar a este servicio y suscribirse. Y cuando este nos devuelva el evento insertado, es cuando debemos usar el EventEmitter para informar a eventos-show que debe de añadirlo y reiniciar el objeto del formulario (**importante**: debemos emitir el evento que nos devuelve el servidor y no el del formulario, ya que entre otras cosas, tendrá un id).

1.3 *Borrar evento*

Para borrar un evento llamaremos al servicio **'/events/{idEvento}'** usando el método **DELETE** (ejemplo → /events/7). Este servicio sólo informa de si se ha borrado correctamente en la base de datos con una respuesta así:

```
export interface OkResponse {
  ok: boolean;
  error?: string;
}
```

Como se puede adivinar, el atributo error sólo existirá si ok es false. El método a crear en el servicio **EventosService** devolverá un booleano en el observable indicando que todo ha ido bien (true), si no, lanzará (throw) el error.

```
deleteEvento(idEvent: number): Observable<boolean> {
  ...
}
```

El método de borrar evento en **evento-item**, llamará a este servicio y se suscribirá, emitiendo el evento de borrado a **eventos-show** cuando el servicio le haya respondido correctamente.