Bloque 5

Programación con JavaScript

Ejercicio bloque 5

Programación con JavaScript
Cefire 2017/2018
Autor: Arturo Bernal Mayordomo

Índice

Ejercicio bloque 5....3
Cambio en los servicios....3
Interfaces y clases....3
Añadir evento....4
Mostrar eventos....4
Otras cosas a tener en cuenta....5
Optativo (Google Maps)....6
Localizar un evento al añadir (Autocomplete)....6

Ejercicio bloque 5

El ejercicio del bloque 5 será una adaptación del ejercicio anterior a TypeScript, añadiendo funcionalidad de geolocalización.

Dependencias NPM necesarias:

- Cambiaremos las dependencias de babel (ES2015+) por las de TypeScript además de las de Google Maps para la parte opcional → **google-maps-promise**, **typescript** y **ts-loader**.

- Archivos de definición de tipos para Google Maps y Handlebars (para quienes realicen la parte optativa) → **@types/googlemaps**, **@types/handlebars**.

En la configuración del compilador de TypeScript (tsconfig.json) debes establecer como objetivo la versión ES2015 (**"target": "es2015"**) o no detectará el tipo ni la clase Promise. Una solución alternativa sería instalar es6-promise y @types/es6-promise, que emulan las promesas en ES5 e importar dicha clase en el código: import {Promise} from 'es6-promise';

Cambio en los servicios

En lugar de apuntar a http://arturober.com/exercise3, los servicios ahora apuntarán a http://arturober.com/bloque5, ya que los eventos ahora tendrán información extra como se explicará a continuación (por lo demás funciona de forma idéntica).

Interfaces y clases

Los eventos ahora tendrán información extra de geolocalización (latitud, longitud y distancia). Os proporciono las interfaces IEvent con las propiedades de un evento y también IResponse que representa el objeto de respuesta del servidor.

```
export interface IEvent {
    id?: number, // Opcional, hasta que no se inserta, no existe
    name: string,
    date: string, // Si se prefiere, se puede usar Date en lugar de string
    description: string,
    image: string,
    price: number,
    lat: number,
    lng: number,
    distance?: number // Opcional, distancia del evento a tu posición
}

export interface IResponse {  // El servidor siempre responde con ok y opcionalmente otra propiedad
    ok: boolean,
    error?: string,
    event?: IEvent,
    events?: IEvent[],
}
```

Las clases deberán indicar en cada método el tipo de parámetro que reciben y lo que devuelven. En caso de devolver una promesa, se tiene que indicar qué tipo de dato retornará la promesa. Ejemplo: Promise<EventItem[]> (Array de objetos EventItem).

```
export class Http {
   // Devuelve Promise<IResponse> con el objeto JSON que devuelve el servidor
   static ajax(method: string, url: string, data: any = null): Promise<IResponse> {}
}

export class EventItem implements IEvent {

   // Propiedades heredadas de IEvent

   constructor(eventJSON: IEvent) {...}

   static getEvents(): Promise<EventItem[]> {...}

   static getEvent(id: number): Promise<EventItem> {...}

   post(): Promise<boolean> {...}

   delete(): Promise<boolean> {...}

   toHTML(): HTMLDivElement {...}
}
```

Añadir evento

Al añadir un evento, se debe geolocalizar al usuario (navigator.geolocation) y asignar latitud y longitud a las propiedades **lat** y **lng** respectivamente en el objeto EventItem que vayamos a enviar. Si algo falla, se pueden poner valores por defecto.

Mostrar eventos

Para mostrar eventos, debemos geolocalizarnos primero (si algo falla se puede establecer una latitud y longitud por defecto), y llamar al servicio que nos devuelve los eventos pasando la geolocalización (y así que nos calcule la distancia el servidor) de esta manera (el primer parámetro es la latitud y el segundo la longitud):

http://arturober.com/bloque5/events/38.34526/-0.2345234

Los eventos ahora incluirán un dato extra (distance) que nos indicará la distancia en km a nuestra posición actual. Muéstralo en la card de la siguiente forma:

```
….
            <small class="text-muted">
                dd/mm/yyyy
                 <span class="float-right">Price €</span>
                 <span class="float-right mr-3">Distance Km</span>
            </small>
….
```

Otras cosas a tener en cuenta

Todas las funciones y métodos auxiliares deben tener los parámetros y el tipo de dato que devuelven (si devuelve alguno) tipados. Además, se deben tipar las variables cuya asignación no deje claro el tipo de datos (o se declaren sin asignar datos).

```
}
    let myEvents: any[]
let myEvents = [];
```

```
}
    let myEvents: EventItem[]
let myEvents: EventItem[] = [];
```

Optativo (Google Maps)

Esta parte es opcional y dirigida a quien disponga de más tiempo para hacerla y le interese prácticar un poco con la API de Google Maps.

Se recomienda crear una clase auxiliar (en otro archivo) con al menos estos métodos:

```
export class GMaps {
  constructor(coords: {latitude: number, longitude: number}, divMap: HTMLDivElement) {}

  getMap(): Promise<google.maps.Map> {} // Crea el mapa y lo devuelve (el objeto)

  // Crea un marker y lo devuelve
  createMarker(lat: number, lng: number, color: string): google.maps.Marker { }

  // Crea el elemento Google Places Autocomplete y lo devuelve (El evento 'change' asígnalo fuera)
  getAutocomplete(input: HTMLInputElement): google.maps.places.Autocomplete {}
}
```

Localizar un evento al añadir (Autocomplete)

En el formulario de añadir un evento, añade un input extra y un mapa. El input actuará de Autocomplete (librería Places de Google), y cuando se seleccione un lugar, esa será la latitud y longitud que se envíen con el evento en lugar de nuestra posición actual (Hay un ejemplo de mapas con Autocomplete subido).