EVENTIFY

DESARROLLO DEL PROYECTO

CONTENIDO

1. Introducción

A continuación, se detalla el proceso llevado para el desarrollo de la aplicación web llamada **Eventify**, la cual ofrece al usuario la posibilidad de conectarse con su cuenta personal a la conocida red social, Facebook, con el fin de mostrar de forma atractiva y accesible sobre un mapa los próximos eventos cercanos a él, con su respectiva información detallada.

Para lograr esto, se va a hacer uso de varias APIs para comunicarnos y recuperar información tanto de Google Maps como de Facebook, entre otros. Asimismo, dada la simpleza de la aplicación, aprovecharemos y usaremos la tecnología MEAN, cuyas siglas representan a:

- **MongoDB**, que actuará como base de datos con el fin de tener datos persistentes, a pesar de que en este caso no lo necesitemos.
- **ExpressJS**, un framework de Javascript que funcionará como nuestro backend.
- **Angular**, otro framework que actuará como frontend.
- Y, por último, **NodeJS**, el cual usaremos para hacer funcionar nuestro servidor.

En nuestro caso usaremos Angular CLI, ya que nos facilitará y acelerará mucho la creación del proyecto y sus componentes.

2. Prerrequisitos

Ahora que queremos empezar a desarrollar, lo primero que debemos asegurar es tener instalado en nuestro sistema tanto NPM (https://www.npmjs.com/get-npm) como Angular CLI. El primero nos permitirá instalar paquetes de librerías o servicios en nuestro ordenador o proyecto, como, por ejemplo, la instalación de angular-cli:

```bash
$ npm install -g angular-cli
```

3. Creando la aplicación

Una vez tengamos todo eso controlado, procederemos a crear la aplicación de angular con un simple comando (para más documentación sobre angular CLI tan solo habría que investigar su repositorio público: https://github.com/angular/angular-cli/wiki):

```bash
$ ng new mean-app
```

Este comando se encarga de crear los archivos base de Angular e instalar todas las dependencias vía npm. Una vez creada toda la aplicación, podremos abrir nuestro servidor para luego acceder escribiendo `http://localhost:4200` desde nuestro navegador.

```bash
$ ng serve
```

4. Añadiendo Express

Para poder continuar con nuestro proyecto, necesitaremos añadir Express a la aplicación para poder "apuntar" a la carpeta que creará Angular al compilar, y para poder manejar las rutas que tendrá Eventify.

```bash
$ npm install --save express body-parser
```

Instalamos tanto express como body-parser por si en el futuro quisiéramos parsear información JSON que nos llegara a través de peticiones POST.

Tras esto ya podremos crear nuestro archivo `server.js` que apuntará a otra carpeta que crearemos, también llamada "server". En el archivo tendremos el código dedicado al servidor, el cual se detalla en los comentarios de la imagen inferior:

```
// Nuestras dependencias
const express = require('express');
const path = require('path');
const http = require('http');
const bodyParser = require('body-parser');

// Las rutas API
const api = require('./server/routes/api');

// Creación de la aplicación EXPRESS
const app = express();

// Para parsear peticiones POST en el futuro
app.use(bodyParser.json());
app.use(bodyParser.urlencoded({ extended: false }));

// Apuntar el PATH a nuestra carpeta DIST
app.use(express.static(path.join(__dirname, 'dist')));

// Setteando la ruta de nuestra API
app.use('/api', api);

// En nuestra app, siempre devolver nuestro INDEX en cualquier ruta
app.get('*', (req, res) => {
  res.sendFile(path.join(__dirname, 'dist/index.html'));
});

/*** * * * * * * * * * ***/
/**
 * Asignamos el puerto para la app de EXPRESS
 */
const port = process.env.PORT || '4000';
app.set('port', port);

/**
 * Creando el Servidor HTTP...
 */
const server = http.createServer(app);

/**
 * Puerto abierto en el 4000
 */
server.listen(port, () => console.log(`API running on localhost:${port}`));
```

Como podemos observar, el código de arriba se encarga de configurar una aplicación de express con una ruta de nuestra propia API abierta, y que redirige siempre hacia nuestro archivo localizado en `dist/index.html`. Para Eventify, en nuestro caso, no necesitaremos modificar más el código del servidor. Antes de nada, debemos crear el archivo `api.js` dentro de las carpetas `server/routes/`.

Dado que nuestra aplicación no requiere de datos capturados desde el lado del servidor, en nuestro archivo `api.js` no habrá nada de interés, pero, con el fin de ejemplificar poniendo el caso de que diéramos uso a MongoDB, a continuación, se muestra cómo sería el código si nuestros eventos estuvieran almacenados en una base de datos o si quisiéramos cogerlos de una API externa desde el servidor:

```
const express = require('express');
const router = express.Router();

const axios = require('axios');
const API = 'https://jsonplaceholder.typicode.com';

//MONGODB
const mongojs = require('mongojs');

//mongodb://<dbuser>:<dbpassword>@ds157500.mlab.com:57500/eventify  //colecciones
const db = mongojs('mongodb://ennaxor:                lab.com:57500/eventify', ['events']);

/* GET API */
router.get('/', (req, res) => {
  res.send('api works');
});

/* EJEMPLO SACANDO DATOS DE UNA API COMO AXIOS */
router.get('/posts', (req, res) => {
  axios.get(`${API}/posts`)
    .then(posts => {
      res.status(200).json(posts.data);
    })
    .catch(error => {
      res.status(500).send(error)
    });
});

/* EJEMPLO SACANDO DATOS DE MONGOFB */
// Lista de eventos
router.get('/events', (req, res) => {
    db.events.find(function(err, docs){
        if(err){
          res.status(500).send(err);
        }
        res.status(200).json(docs);
    });

});
```

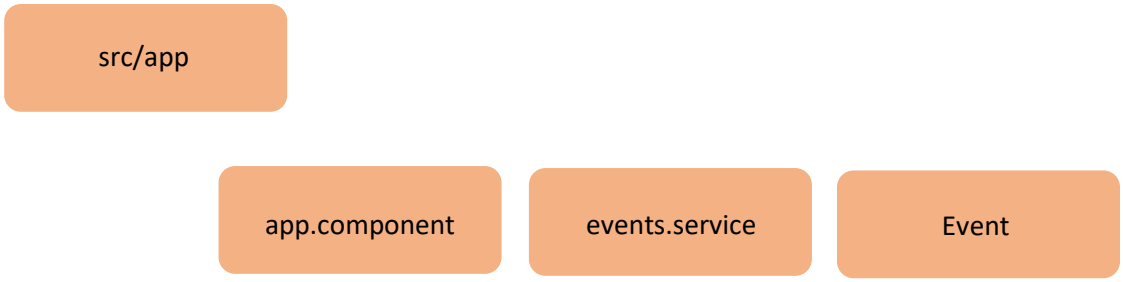
```
// Evento en concreto según una ID
router.get('/event/:id', (req, res) => {
    db.events.findOne({ id: mongojs.ObjectId(req.params.id)}, function(err, doc){
        if(err){
          res.status(500).send(err);
        }
        res.status(200).json(doc);
    });
});

module.exports = router;
```

Para seguir con el ejemplo, si ahora compilásemos nuestra aplicación con `ng build` y `ng server.js`, en nuestro navegador veríamos la colección de posts sacados de la API usada (para la ejemplificación, se ha usado el plugin de Chrome JSON Viewer).


```
[
  {
    "userId": 1,
    "id": 1,
    "title": "sunt aut facere repellat provident occaecati excepturi optio reprehenderit",
    "body": "quia et suscipit\nsuscipit recusandae consequuntur expedita et cum\nreprehenderit molestiae ut ut quas totam\nnostrum rerum est autem sunt rem eveniet architecto"
  },
  {
    "userId": 1,
    "id": 2,
    "title": "qui est esse",
    "body": "est rerum tempore vitae\nsequi sint nihil reprehenderit dolor beatae ea dolores neque\nfugiat blanditiis voluptate porro vel nihil molestiae ut reiciendis\nqui aperiam non debitis possimus qui neque nisi nulla"
  },
  {
    "userId": 1,
    "id": 3,
    "title": "ea molestias quasi exercitationem repellat qui ipsa sit aut",
    "body": "et iusto sed quo iure\nvoluptatem occaecati omnis eligendi aut ad\nvoluptatem doloribus vel accusantium quis pariatur\nmolestiae porro eius odio et labore et velit aut"
  },
```

5. Rutas y componentes en Angular

Nuestra aplicación va a ser singlepage, lo que quiere decir que no nos complicaremos creando múltiples rutas ya que sólo habrá una componente que modificaremos y daremos estilo. Además de eso, solamente trabajaremos con un tipo de objeto: los eventos, los cuales ni éstos ni los usuarios logueados almacenaremos en ninguna base de datos. Por este motivo, tan solo crearemos un servicio para manejar los eventos y una clase Evento con todos los atributos que queramos guardar, a parte de la componente de la propia aplicación. En resumidas cuentas, éstas son las partes que compondrán nuestro proyecto (excluyendo los .html y .css):



Ahora bien, si creyésemos oportuno crear otra componente dedicada a los eventos, podríamos hacerlo con angular-cli usando el siguiente comando (cambiando posts por events:

```
BASH

$ ng generate component posts
```

Este comando se encarga de crear un directorio nuevo llamado posts (o events) y dentro de éste sus correspondientes archivos .html y .css, aparte del requerido elemento `events.component.js`. Como ya hemos dicho antes, Angular cli nos acelera todo el proceso de creación de estos elementos, por lo que además de añadirse los archivos mencionados, también se encarga de generar la componente Events en el archivo `src/app/app.module.js` y la declara en el módulo `@NgModule` para poder trabajar con ella.

```
// RUTA
const ROUTES = [
  {
    path: '',
    redirectTo: 'events',
    pathMatch: 'full'
  },
  {
    path: 'events',
    component: EventsComponent
  }
];

@NgModule({
  declarations: [
    AppComponent,
    EventsComponent
  ],
```

En la imagen superior se puede apreciar esto que se ha dicho último. Además de ello, deberemos definir las rutas de nuestra aplicación. Esto se puede hacer de muchas maneras en Angular, pero aquí mostraremos una de las más sencillas y más aplicables para una aplicación web singlepage. En la declaración de la ruta lo que hacemos es decirle al "router" de express que siempre que en el navegador se visite la ruta `/` se redirija a ruta `/events` y para ello luego declaramos esa ruta y asignamos la componente que queremos que se ejecuta.

Aún nos faltaría crear el servicio para manejar nuestros servicios. Para ello volveremos a angular cli y a sus sencillos comandos: `ng generate service events,` lo cual creará el archivo .ts en nuestro directorio principal. Después de esto simplemente tendríamos que añadir el servicio como proveedor en el código de la última imagen mostrada.

Para acabar con esta primera parte creando las bases de nuestra aplicación, tendremos que asegurarnos de incluir `<base href="/">` en nuestro archivo `src/index.hrml` y añadir un `router-outlet` donde queramos que nuestra ruta se renderice, es decir, en `src/app/app.component.html`. A partir de ahora para ver los cambios en nuestro código en el navegador, tendremos que usar los comandos mencionados anteriormente. Para facilitar el trabajo lo que haremos será abrir nuestro archivo `package.json` y modificar la línea oportuna con el fin de sólo necesitar usar el comando de `npm run build` y abrir el navegador.

JSON

```
{
  "name": "mean-app",
  // meta data
  "scripts": {
    // Other scripts
    "build": "ng build && node server.js"
  },
  "private": true,
  "dependencies": {
    ...
  },
  "devDependencies": {
    ...
  }
```

6. Acceso del usuario

Ahora que tenemos nuestra aplicación funcionando con una base sólida, nos centraremos en implementar la primera funcionalidad de nuestro proyecto: el acceso a la aplicación mediante Facebook. Existen múltiples servicios y librerías que podríamos usar para conectar con la API de Facebook y acceder a la información de cada usuario logueado, por lo que, analizando todos estos recursos, en esta ocasión utilizaremos la librería `NGX-FACEBOOK` (https://github.com/zyra/ngx-facebook), cuya instalación está detallada en el propio repositorio. Una vez realizados todos los pasos de las instrucciones, tendremos que inicializar el servicio que nos aporta la librería en nuestra componente.

```
54
55      //llamada por primera vez antes del ngOnInit()
56      constructor(private fb: FacebookService,
57                  private http: Http,
58                  private router: Router,
59                  private eventsService: EventsService,
60                  private mapsAPILoader: MapsAPILoader,
61                  private mapApiWrapper: GoogleMapsAPIWrapper,
62                  private ngZone : NgZone) {
63
64          console.log('Initializing Facebook');
65
66          fb.init({
67              appId: '631545433723193',
68              version: 'v2.9'
69          });
```

Como estamos utilizando la API de Facebook, tendremos que hacernos una cuenta en la consola de desarrolladores de éste (https://developers.facebook.com) y crear nuestra aplicación. Una vez hecho esto, podremos pedir el uso de inicio de sesión con su sistema para que nos devuelvan las credenciales necesarias y esa ID que deberemos asignar en la inicialización del servicio anterior.

Con el servicio inicializado podremos hacer uso de las funciones que nos aporta, como el de loguearnos o recibir información sobre su perfil. Ya que nuestra aplicación requiere de permisos más privilegiados para capturar los eventos del perfil de ese usuario, tendremos que utilizar el método de acceso con más opciones.

```
199     /**
200     * LOGIN CON MÁS PERMISOS. Recibimos lista de amigos, de eventos...
201     */
202     loginWithOptions() {
203         const loginOptions: LoginOptions = {
204             enable_profile_selector: true,
205             return_scopes: true,
206             scope: 'public_profile, user_friends, email, pages_show_list, user_events'
207         };
208
209         this.fb.login(loginOptions)
210             .then((res: LoginResponse) => {
211                 console.log('Logged in', res);
212                 this.loggedIn = true;
213                 localStorage.setItem('auth_token', res.authResponse.accessToken);
214                 myGlobals.setUSER_TOKEN(res.authResponse.accessToken);
215                 this.getProfile();
216
217         })
218             .catch(this.handleError);
219
220     }
```

Mientras el usuario exista en Facebook, podremos acceder a su perfil y guardar en nuestra localStorage el token que nos devuelve la API al recibir la petición de acceso, para que, de esta forma, podamos controlar qué enseñar y qué no enseñar según esté logueado o desconectado. Los datos que nos interesan del usuario para Eventify son solamente los eventos a los que está ligado, y su nombre, para así ilustrarlo en la interfaz y llegar al usuario más fácilmente, lo cual se verá más adelante cuando demos estilo a nuestra aplicación.

7. Recibiendo los eventos

El siguiente paso será acceder a los eventos del usuario (independientemente de su asistencia) con el TOKEN que hemos recuperado de la API de Facebook. Ahora bien, lo normal sería pensar que la información de los eventos la podríamos coger de la misma API (llamada Graph API, https://developers.facebook.com/docs/graph-api), pero realizar esta petición ya no es posible desde 2014 ya que Facebook complicó sobremanera la búsqueda de eventos por ubicación, lo cual es lo que justo necesitamos para Eventify.

Por este motivo, realizando una búsqueda exhaustiva de otros servicios y APIs que nos proporcionen estos datos, resalta el servicio creado por un usuario llamado Tobilg cuyo repositorio está público en Github (https://github.com/tobilg/facebook-events-by-location) y que nos reporta justamente lo que queremos para nuestro proyecto. Aun así, pese a tener a nuestra disposición este servicio, usaremos otro basado en él (https://github.com/pabloppp/Facebook-Events); desarrollado por Pablo Pernías que simplifica la librería para poder usarla en el lado del cliente en lugar de en el del servidor.

Una vez añadida la dependencia a nuestro proyecto, en nuestro caso simplemente descargada y añadida a nuestra carpeta de `assets`, por fin abriremos nuestro servicio `events.service` e importaremos el de los eventos para poder empezar a gestionar la información que recibamos de los eventos.

Como bien se explica en el repositorio, inicializamos el servicio de eventos de Facebook para que, a posteriori, podamos acceder al método de `setToken(tkn)` y pasarle en el debido momento el token de nuestro usuario logueado.

Antes de poder hacer eso, tendremos que completar nuestro archivo `Event.ts`. Crearemos un objeto de tipo interfaz, ya que todos los eventos que recibamos serán iguales y querremos sacar los mismos datos para almacenarlos en un array de este tipo de objeto.

De todos los datos que recibimos de la API, los que nos interesan son los siguientes:

```
export interface Event {
    name: string;
    description: string;
    start_time: string;
    end_time: string;
    attending: number;
    picture: string;
    latitude: number;
    longitude: number;
    place: string;
}
```

Por lo que, cuando vayamos a llamar al método en cuestión, el resultado que nos devuelva lo almacenaremos en una variable de tipo `Event[] = []`:

```
searchEvents(lat, lng){
    var eventOptions = {
        lat: lat,
        lon: lng,
    //  since: 1500595200,
    //  until: 1503187200,
        filter: true
    }
    this.fbEvent.setToken(this.mytkn)
        .then(() => {
          return this.fbEvent.getEvents(eventOptions);
        })
        .then(evs => {
          for(let i of evs){
                //console.log(evs);
                this.events.push({
                    name: i.name,
                    description: i.description,
                    start_time: i.start_time,
                    end_time: i.end_time,
                    attending: i.attending_count,
                    picture: i.cover.source,
                    latitude: i.venue.location.latitude,
                    longitude: i.venue.location.longitude,
                    place: i.venue.location.city
                });
          }
          localStorage.setItem("storageEvents", JSON.stringify(this.events));

        })
        .catch(err => {
          console.log(err);
        })
}
```

Este array de eventos, para que no se elimine al actualizar la página, tendremos que añadirlo también a un elemento en el localStorage, tal y como se hace al final del código de la imagen anterior.

La pregunta es, ¿cuándo llamamos a este método declarado `searchEvents(lat, lng)`? Notaremos que para poder hacer uso de la API además del token de usuario de Facebook, también necesitaremos sus coordenadas, es decir, su ubicación.

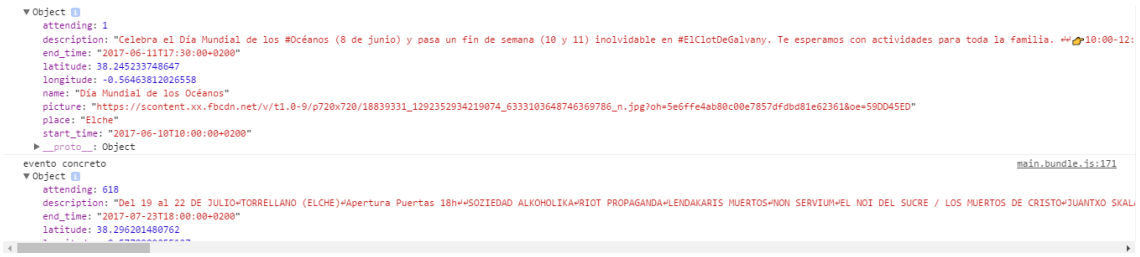
Para conocer las coordenadas de un usuario, podemos usar la ya incluida librería de Google Maps que nos facilita la geolocalización. Sin tener que importar nada, en el `ngOnInit()` de nuestra componente haremos lo siguiente:

```
if(navigator.geolocation){
    navigator.geolocation.getCurrentPosition((position) => {
        Promise.all([
            this.myLat = position.coords.latitude,
            this.myLong = position.coords.longitude,
            this.lat = this.myLat,
            this.lng = this.myLong,
            this.getLocationName(this.lat, this.lng),
            //guardar coordenadas para html
            this.myLat = this.myLat.toString(),
            this.myLong = this.myLong.toString(),
            this.myCoords = this.myLat.substring(0,7) + ", " + this.myLong.substring(0,7)
          ]).then(() => this.eventsService.searchEvents(this.myLat, this.myLong));

    });
    Promise.all([
        this.subscription = this.eventsService.getAllEvents()
            .subscribe(ev => {
                this.events  = ev;
            })
    ]).then(() => this.drawEvents());
```

Tendremos que usar las promesas de Angular con el fin de que no se ejecute la función de búsqueda de eventos hasta que no sepamos y guardemos las coordenadas del usuario. Como podemos observar, las volveremos a usar más adelante, para coger los eventos del servicio y tenerlos en la App.Component, ya que es ésta la que se encarga de dibujar los eventos y su información sobre un mapa. Cabe mencionar que también se usarán `Observables` para que no se adelanten funciones antes de recibir los datos necesarios.

8. Dibujando los eventos

Llegados a este punto, a través del token del usuario logueado y de sus coordenadas captadas por la geolocalización hemos conseguido recibir la información de los eventos cercanos al usuario a un radio de unos 25 Km aproximadamente. Lo único que nos faltaría para acabar esta simple aplicación sería dibujar sobre un mapa de Google Maps en marcadores los eventos recibidos.



Para ello, usaremos la API de Google Maps, y la librería llamada `angular2-google-maps` (https://angular-maps.com/), que facilita la conexión entre angular y esta API, como bien indica su nombre. Tras incluirla en nuestro proyecto como se indica en los pasos a seguir, tendremos que volver a hacernos con una clave de desarrollador, pero esta vez en la consola de desarrollador de Google.

API	Solicitudes	Errores	Proporción de errores	Latencia, mediana	Latencia (98%)	
Google Maps Directions API	–	–	–	–	–	Inhabilitar
Google Maps Distance Matrix API	–	–	–	–	–	Inhabilitar
Google Maps Elevation API	–	–	–	–	–	Inhabilitar
Google Maps Geocoding API	–	–	–	–	–	Inhabilitar
Google Maps JavaScript API	–	–	–	–	–	Inhabilitar
Google Places API Web Service Private API	–	–	–	–	–	Inhabilitar

Tendremos que habilitar todas las APIs que queramos para hacer funcionar nuestra aplicación, y en nuestro `@NgModule` asignar la clave que nos han devuelto en la consola:

```
43      RouterModule.forRoot(ROUTES), //añade la ruta a la app
44      FacebookModule.forRoot(),
45      AgmCoreModule.forRoot({
46        apiKey: 'AIzaSyC-maeGYU-LxKvFNdp-APTGVQrNMsErmCA'
47      })
48    ],
```

Por fin empezaremos a crear el código de la interfaz, ya que para dibujar el mapa de Google Maps usaremos las siguientes etiquetas en nuestro HTML:

```
65                    <sebm-google-map
66                          [latitude]="lat"
67                          [longitude]="lng"
68                          [zoom]="zoom"
69                          [disableDefaultUI]="false"
70                          [zoomControl]="false"
71                          (mapClick)="mapClicked($event)"
72                          [usePanning]="true"
73                          *ngIf="isLoggedIn()">
74
75                          <sebm-google-map-marker
76                              *ngFor="let m of markers; let i = index"
77                              (markerClick)="clickedMarker(m.label, i)"
78                              [latitude]="m.lat"
79                              [longitude]="m.lng"
80                              [markerDraggable]="m.draggable"
81                              (dragEnd)="markerDragEnd(m, $event)">
82
```

Aun así, todavía nos quedaría transformar esos eventos que ahora tenemos como objetos de JSON en marcadores visibles y atractivos para el usuario. Por ello, crearemos otra interfaz, pero esta vez en la propia componente con el propósito de ejemplificarlo de otra manera. Esta interfaz, llamada `marker`, constará de las mismas variables que el objeto de tipo evento.

```
306   interface marker {
307       lat: number;
308       lng: number;
309       latlng: string;
310       label?: string;
311       description: string;
312       start_time: string;
313       attending: number;
314       picture: string;
315       draggable: boolean;
316       place: string;
317       end_time: string;
318   }
319
```

Ahora sí, en nuestro método `drawEvents()` podremos añadir a un nuevo array de marcadores, uno a uno, los eventos que hayamos recibido de nuestro servicio, teniendo en cuenta que podrían haber duplicados (eventos colocados en las mismas coordenadas), por lo que tendríamos que hacer un rápido apaño.

```
139             for(var i of this.events){
140                 formattedDateS = this.convertDate(i.start_time);
141                 formattedDateE = this.convertDate(i.end_time);
142
143                 //valor de lat + long para comprobar duplicados
144                 latlngAux = ""+i.latitude+i.longitude;
145
146                 if(this.markers.length > 0){
147                     for(var j of this.markers){
148                         if(j.latlng == latlngAux){
149                             latAdded = 0.002;
150                             lngAdded = 0.002;
151                         }
152                         else{
153                             latAdded = 0;
154                             lngAdded = 0;
155                         }
156                     }
157                 }
158
159                 this.markers.push({
160                     lat: i.latitude+latAdded,
161                     lng: i.longitude+lngAdded,
162                     latlng: latlngAux,
163                     label: i.name,
164                     draggable: false,
165                     description: i.description,
166                     start_time: formattedDateS,
167                     end_time: formattedDateE,
168                     attending: i.attending,
169                     picture: i.picture,
170                     place: i.place
171                 });
172
173             }
```

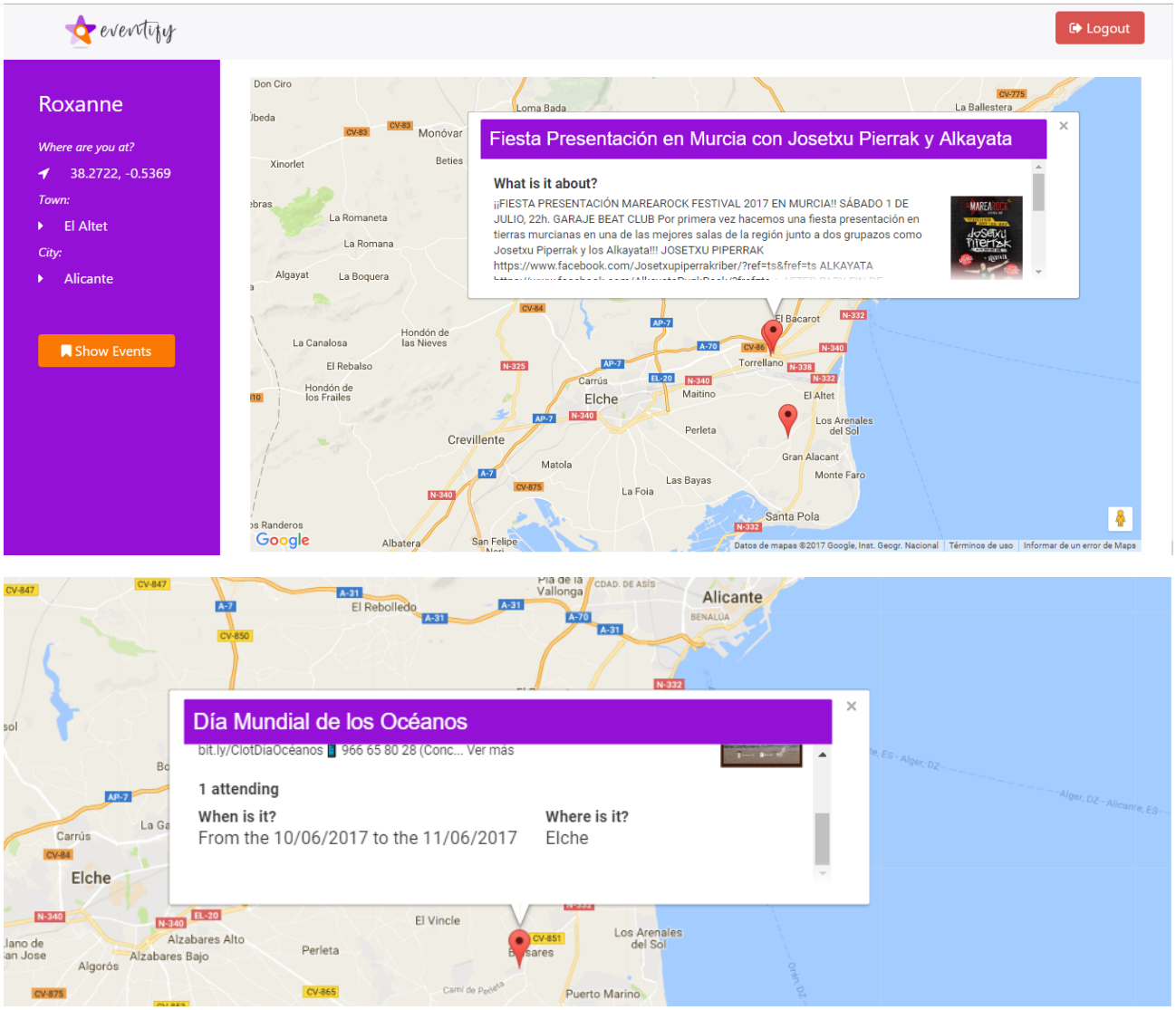
En nuestro HTML dibujaremos nuestros marcadores, colocando a su vez toda la información relevante que consideremos enseñar al usuario.

```
<sebm-google-map-info-window>
  <div id="iw-container">
      <div class="iw-title">{{m.label}}</div>
      <div class="iw-content">
          <div class="iw-subTitle">What is it about?</div>
          <img src="{{m.picture}}" height="115" width="83">
          <p>{{m.description}}</p>
          <div class="iw-subTitle">{{m.attending}} <b>attending</b></div>
          <div class="when">
            <div class="iw-subTitle ">When is it?</div>
            <p class="s-time">From the {{m.start_time}} to the {{m.end_time}}</p>
          </div>
          <div class="where">
            <div class="iw-subTitle ">Where is it?</div>
            <p class="s-time">{{m.place}}</p>
          </div>

      </div>
```

9. Dando estilo

Hemos implementado todas las funcionalidades previstas para esta simple aplicación web, así que lo único que nos faltaría por hacer es hacerla usable y accesible creando una buena hoja de estilos que se adapte a lo que el usuario querría ver al utilizar Eventify.

Una vez hecha, el resultado final de nuestro producto estaría acabado y listo para lanzar al mercado.



10. Mejoras

La aplicación desarrollada carece de muchas posibles funcionalidades que harían de la experiencia de usuario una infinitamente mejor. A continuación, se listan algunas ideas y posibilidades que podrían hacer de este proyecto básico de ejemplo, un posible producto de mercado que se podría lanzar en el futuro:

- Diseño responsive para multiplataforma
- Forma de buscar por distancia o cambiar el radio de búsqueda
- Botones de asistencia en los propios marcadores y feedback al propio Facebook
- Catálogo de eventos con el fin de ordenarlos y buscar por categoría
- Lista de amigos y muestra de su asistencia a los eventos
- Botones para compartir la información en otras RRSS
- Crear un evento desde la propia aplicación y feedback a Facebook

11. Bibliografía

A la hora de implementar esta aplicación se han usado las siguientes fuentes:

- Google Maps API: https://angular-maps.com/docs/getting-started.html
- Google Maps API repo: https://github.com/SebastianM/angular-google-maps
- Facebook Events: https://github.com/tobilg/facebook-events-by-location
- Facebook Events API: https://github.com/pabloppp/Facebook-Events
- Facebook Login repo: https://github.com/zyra/ngx-facebook
- Facebook Login: https://zyra.github.io/ngx-facebook-example/
- Graph API: https://developers.facebook.com/docs/graph-api
- Blogs:https://www.reddit.com/r/Angular2/comments/4etjfi/integration_of_facebook_login_with_angular_2/
- Angular CLI doc: https://cli.angular.io/
- Angular CLI doc: https://scotch.io/tutorials/use-the-angular-cli-for-faster-angular-2-projects
- Bootstrap 4 doc: https://v4-alpha.getbootstrap.com/
- FontAwesome doc: http://fontawesome.io/get-started/
- Ejemplo de MongoDB: https://mlab.com/
- Google Console: https://console.developers.google.com/
- Facebook Console: https://developers.facebook.com/