Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

MyEvents – Tu gestor de eventos personal

Trabajo Fin de Grado

Grado en Ingeniería Informática

Autor: Manuel Puchades Bresó

Tutor: Pedro José Valderas Aranda

Curso: 2017-2018

Agradecimientos

A mis padres.

Sin vosotros nada en estos 4 años hubiera podido ser posible.

A Pedro, mi tutor.

Por aguantar mis constantes correos (sobre cualquier cosa) y mis múltiples dudas y cuestiones.

A Platzi.

Por ayudar a miles de personas a formarse un futuro en este mundo. Seguid haciendo lo que hacéis.

A Alejandro, a David B, a David M y a Pablo.

En mayor o menor medida, habéis sido parte de esto. Gracias por cruzaros en mi camino.

A todos en general.

Muchísimas gracias.

Resumen

Este trabajo está centrado en la creación de una aplicación web destinada a la gestión personalizada de eventos de todo tipo. La aplicación permite al usuario llevar un seguimiento de los eventos más populares afines a sus gustos, crear eventos llevados a cabo por él, ver los eventos creados por otros usuarios, etc. Este proyecto es un pequeño ejemplo de cómo construir una aplicación web fácilmente con la ayuda de Javascript, Node y MongoDB, así como con librerías como Bootstrap para facilitar todo el proceso.

Palabras clave: Eventos, gestión, seguimiento, MVC, MongoDB.

Resum

Este treball està centrat en la creació d'una aplicació web destinada a la gestió personalitzada d'esdeveniments de qualsevol tipus. L'aplicació permet a l'usuari portar un seguiment dels esdeveniments més populars afins als seus gustos, crear esdeveniments duts a terme per ell, vore els esdeveniments creats per altres usuaris, etc. Este projecte és un xicotet exemple de com construir una aplicación web fàcilment amb l'ajuda de *Javascript*, Node, i *MongoDB*, així com amb llibreries com *Bootstrap*, etc per a facilitar tot el procés.

Paraules clau: Esdeveniments, gestió, seguiment, MVC, MongoDB.

Abstract

This work is focused on the creation of a web application aimed at the personalized management of events of all kinds. The application allows the user to keep track of the most popular events related to their tastes, create events carried out by him, see events created by other users, etc. This project is a small example of how to build a web application easily with the help of Javascript, Node and MongoDB, as well as with libraries like Bootstrap to facilitate the whole process.

Keywords: Events, management, monitoring, MVC, MongoDB.

Tabla de contenidos

CAPÍTULO 1
Introducción

A continuación, se ofrece una breve introducción acerca del presente trabajo. Los aspectos que se mencionan aquí serán explicados con más detalle en los próximos capítulos.

1.1 Motivación

Nuestra sociedad se halla en un momento crucial en el que todo cuanto conoce está pasando en eso que llaman 'internet', en el que lo físico cada vez es menos físico y en el que todo va al ritmo que el desarrollo del software marca. Cualquier innovación de cualquiera de las más grandes empresas es llevada a cabo por algún tipo de software. Es por eso que es casi ineludible el aprender cómo desarrollar software.

Además de por dicha obligación casi imposible de evitar, la necesidad personal del autor de crear un proyecto desde cero se junta con la pasión del mismo por los eventos de cualquier indole. Así nace MyEvents.

1.2 Objetivos

El objetivo principal de este proyecto es la creación de una aplicación web (principalmente mediante el uso del lenguaje JavaScript) que permita al usuario gestionar sus eventos, listarlos, crear nuevos, suscribirse a otros, etc. Una gestion total.

Como objetivo adicional, se incluye la posibilidad de que la aplicación web sea visible en todo tipo de pantallas. Así, utilizaremos librerías *responsive* para

maquetar todo el front-end de nuestra aplicación (*Bootstrap*, *MaterializeCSS*, etc).

1.3 Estructura de la memoria

Esta memoria está dividida en múltiples capítulos, en los que se cubrirán distintos aspectos acerca del trabajo realizado, así como también se dará información referente a la base de los conceptos que se explican.

En primer lugar, se revisarán las aplicaciones o sistemas existentes similares a la aplicación desarrollada, viendo sus virtudes y defectos. A continuación, se realizará una breve presentación de las herramientas y tecnologías utilizadas. Después de esto se explicará la arquitectura del sistema, es decir: elementos que participan, cómo se comunican y que tipo de información se envían.

Después de esto, se pasará a explicar la metodología de desarrollo que se ha seguido dando ejemplos que justifiquen la utilizada. A continuación, se analizarán los requisitos de usuario, mediante la utilización de casos de uso, escenarios y diagramas de clases. Después, se dará una visión del diseño de la aplicación mediante los bocetos utilizados y los modelos de la BBDD existentes.

Después del diseño nos meteremos de lleno en el código de la aplicación mediante ejemplos del código de todas las partes del sistema (interfaces, conexión al servidor, servidor, etc). Junto a cada ejemplo del código se dará una breve explicación del funcionamiento, lugar del sistema al que pertenece, etc.

Para acabar se realizarán unas pequeñas conclusiones que incluirán un resumen del trabajo realizado y una opinión personal.

Como apéndice se realizará un manual de usuario de la aplicación.

CAPÍTULO 2

Estado del arte

En este apartado se va a revisar la principal aplicación o sistema existente similar a la aplicación desarrollada. Daremos una breve introducción sobre que es, daremos unos puntos a favor y en contra y acabremos el apartado dando una pequeña visión de las diferencias entre ambas aplicaciones.

2.1 Meetup

Figura 1.- Logo de Meetup. Fuente: [1]

Meetup [1] es una plataforma de redes sociales creada por Scott Heiferman, Matt Meeker y Peter Kamali en el año 2002. Esta plataforma permite a sus miembros reunirse en la vida real vía grupos unidos por un interés común (deporte, cultura, libros, tecnología, idiomas, etc..). La utilización del sitio es gratuita para los asistentes a las reuniones de los grupos, pero para los creadores de los grupos es de pago.

Es clara la relación entre *Meetup* y *MyEvents*. Su logo (Figura 1) puede asemejarse al de *MyEvents*. La base de *Meetup* fue la piedra angular sobre la que surgió la idea de *MyEvents*, apoyada en las ganas de mejorar un producto ya existente (en la medida de lo posible).

Algunos puntos a favor (que trataremos de imitar) y algunos puntos en contra (que trataremos de mejorar) de *Meetup* son los siguientes:

Puntos a favor

- Interfaz impecable.
- Gran variedad de funciones.
- Una infinidad de posibilidades de mejora

Puntos en contra

- Dificultad de uso en puntos concretos de la aplicación.
- Excesivos tiempos de carga en momentos puntuales.

En resumen, trataremos de imitar la interfaz gráfica de Meetup, así como su gran variedad de funciones. También, trataremos de mejorar la facilidad de uso y los tiempos de carga en toda la aplicación.

CAPÍTULO 3

Análisis de requisitos

El proceso de análisis de requisitos [2][3][4][5] es un proceso de descubrimiento, refinamiento y especificación. Tanto el desarrollador como el cliente (o el potencial futuro usuario) tienen un papel activo en esta etapa.

El cliente (o usuario) intenta replantear un sistema confuso, a nivel de descripción de datos, funciones y comportamiento, en detalles concretos. El desarrollador actúa como interrogador, como consultor, como persona que resuelve problemas (y como negociador).

En el caso de este proyecto, ambas figuras (cliente y desarrollador) se juntan en una misma persona (el desarrollador), facilitando el proceso.

El análisis de requisitos puede parecer una tarea relativamente sencilla en este caso, pero las apariencias engañan. Es una tarea de ingeniería del software que cubre el hueco entre la definición del software a nivel sistema y el diseño del software.

El análisis de requisitos del software se puede subdividir en cinco áreas de esfuerzo:

- Reconocimiento del problema
- Evaluación y síntesis
- Modelado
- Especificación
- Revisión

A continuación, pasaremos a describir los casos de uso y escenarios utilizados para la especificación de requisitos.

3.1 Introducción

En este apartado vamos a definir 3 casos de uso diferentes, cada uno con su correspondiente escenario. Detallaremos cada uno de ellos, incluyendo las personas involucradas y demás detalles relevantes.

3.2 Casos de uso + Escenarios

Como se ha dicho en la introducción, vamos a definir 3 casos de uso con sus correspondientes escenarios. El primero, el caso de uso que consiste en encontrar un evento. El segundo, el caso de uso que consiste en crear un evento en concreto. El tercero y último, el caso de uso que consiste en ver todos los eventos creados por un usuario en específico.

3.2.1 - Encontrar un evento

El primero trata el caso de uso en el que el usuario necesita encontrar un evento al que asistir este fin de semana, pues está solo en casa y es un gran amante de los eventos culturales.

Escenario

Carlos está solo en casa, sus padres se han ido de viaje y le han dejado a cargo de la casa. Enciende su ordenador y entra en la aplicación web de MyEvents. Anoche durmió mal, por lo que no se acuerda de iniciar sesión. Accede a la página principal y se dirige a la parte baja de la web. Allí, selecciona la categoría charlas (desde pequeño ha sido un gran aficionado a ellas). Está un rato navegando por las diferentes charlas que se imparten hasta que encuentra una que le llama la atención. Hace *click* en entrar y se abre la página donde le recuerda que debe iniciar sesión para ver los datos de la charla. Hace click en el link e inicia sesión. Vuelve a la página de la charla y ya puede ver todos los datos de la charla. Se los apunta en su teléfono móvil y se dispone a prepararse para salir a la charla que ha seleccionado.

3.2.2 - Crear un evento

El segundo trata el caso de uso en el que el usuario necesita crear un evento organizado por la escuela a la que asiste y de la que es parte de la delegación de alumnos.

Escenario

María forma parte de la delegación de alumnos de la escuela (la ETSINF, Escuela Técnica Superior de Ingeniería Informática) donde cursa la carrera desde hace 4 años. Su escuela va a organizar la jornada de historia de la informática, donde se realizarán actos durante todo el día para divulgar dicha historia. Como María es una de las encargadas de gestionar los eventos, le encargan crear el evento para que todos los alumnos lo puedan ver. Accede a la aplicación web de MyEvents e inicia sesión con el usuario de la escuela en la página. Cuando entra, elimina el evento que organizaron la semana pasada porque a Aarón se le olvidó hacerlo. Una vez eliminado, hace click en 'Crear evento' en la barra superior y rellena los datos y hace click en 'Guardar'. Una vez creado, utiliza su propio usuario para comprobar que aparece correctamente.

Para ello, inicia sesión con su usuario y se dirige a la parte inferior de la página y selecciona la categoría 'Eventos' y observa que aparece correctamente el evento. Aprovecha y haciendo 'click' en un enlace que permite ver los eventos creados por el usuario que ha creado el evento que se está viendo, entra en el perfil de la ETSINF y observa que se ha borrado correctamente el evento y ya no aparece.

3.2.3 - Ver los eventos de un usuario

El tercero trata el caso de uso en el que el usuario necesita saber los eventos creados por otro usuario, dado que asistió a uno de sus eventos y le gustó.

Escenario

Carlos es un amante de la lectura y aunque es un poco tímido, últimamente se ha animado a conocer gente nueva, y a través de MyEvents ha acudido a reuniones sobre amantes de la lectura en cafeterías de su ciudad. La última a la que asistió

organizada por un club de lectura de libros de misterio, en un café céntrico, fue muy divertida, por lo que decide buscar si tienen eventos nuevos a los que asistir.

Entra en la aplicación web e inicia sesión en su cuenta. Se dirige a los eventos más populares y observa que hay un evento organizado por el club de lectores mencionado, aunque como no es sobre libros de su temática favorita decide no ir. Aprovecha y hace click en el enlace para ir al perfil del club de lectura. Observa (con gran alegría) que han organizado otro para la semana que viene (en la que no tiene nada que hacer), por lo que se apunta los datos de la reunión y celebra el haberse dado cuenta de que tenían otra de su temática favorita.

CAPÍTULO 4

Metodología

Una metodología de desarrollo [6] es un marco de trabajo que es usado para estructurar, planear y controlar el proceso de desarrollo en sistemas de información.

A lo largo de toda la historia varios métodos han ido surgiendo diferenciándose entre ellos por su fortaleza y debilidad. El entorno de trabajo para desarrollo consiste en:

- Una filosofía de desarrollo con el enfoque del proceso de desarrollo.
- Herramientas, modelos y métodos para asistir el proceso de desarrollo.

Estos métodos son a menudo vinculados a algún tipo de organización, que además desarrolla, apoya el uso y promueve la metodología.

Como es de esperar, cada metodología de desarrollo tiene su propio enfoque. Cada metodología tiene sus ventajas y sus desventajas. Algunos de los más utilizados son los siguientes:

- Modelo en cascada (proceso secuencial)
- Prototipado (uso de prototipos sin necesidad de implementar todo)
- Incremental (desarrollo 'poco a poco')
- Espiral (desarrollo dividido en segmentos)

En el proyecto se ha utilizado el modelo incremental, por lo que pasaremos a detallar dicha metodología un poco más.

4.1 Modelo incremental

El modelo incremental (cuya estructura puede verse en la figura 2) combina elementos del modelo en cascada con la metodología interactiva de construcción de prototipos. Se basa en la filosofía de construir incrementando las funcionalidades del programa. Este modelo aplica secuencias lineales de forma escalonada mientras progresa el tiempo en el calendario. Cada secuencia lineal produce un incremento del desarrollo.

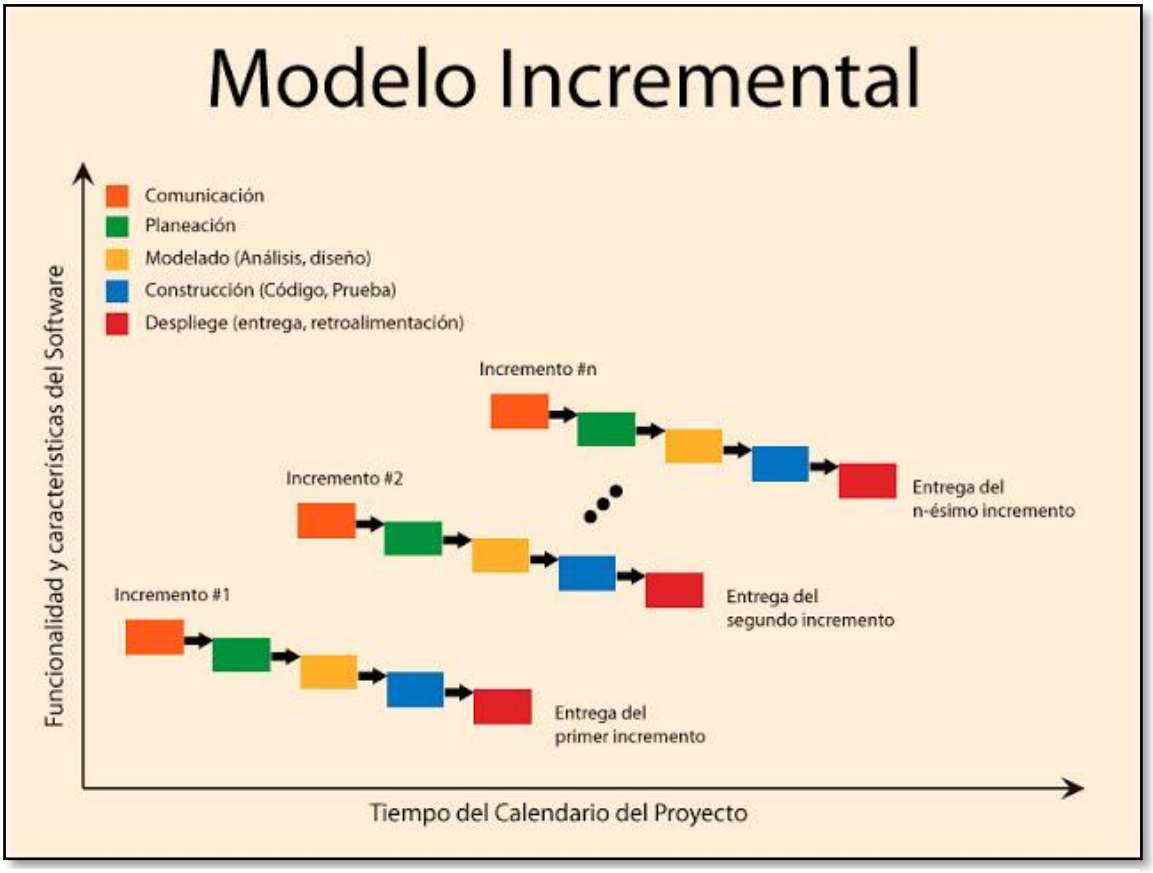


Figura 2.- Modelo incremental. Fuente: [7]

Cuando se utiliza un modelo incremental, el primer incremento es a menudo un producto esencial, sólo con requisitos básicos. Este modelo se centra en la 'entrega' de un producto operativo con cada incremento. Los primeros incrementos sin versiones incompletas del producto final, pero proporcionan al usuario la funcionalidad que precisa y también una plataforma para evaluar.

4.2 Aplicado en MyEvents

La metodología incremental [7][8] ha sido aplicada en el desarrollo de este proyecto siguiendo las bases de dicho marco de trabajo: Generar código operativo de forma rápida, probar y depurar, etc.

Es decir, cada funcionalidad de la aplicación ha sido desarrollada mediante un incremento de esta metodología. Esto puede versen la figura 3.

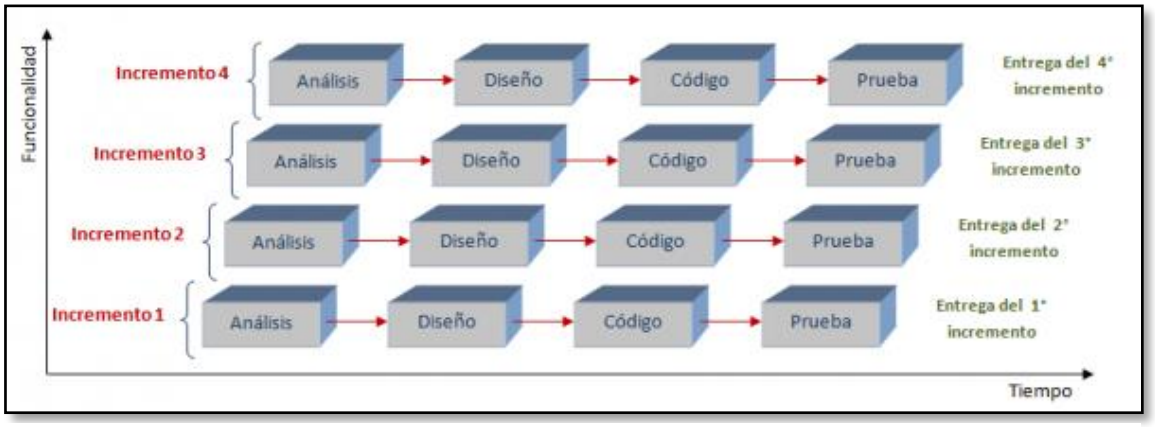


Figura 3.- Fases de los incrementos. Fuente: [8]

Como hemos dicho, cada incremento en el desarrollo de la aplicación ha sido interpretado como una nueva funcionalidad de esta. Y para desarrollar la funcionalidad n, antes (lógicamente) ha debido desarrollarse la funcionalidad n-1 (correspondiente al incremento n-1).

Primero se comienza con un análisis de lo necesario, lo que se quiere hacer. Después, se diseña lo analizado. Más tarde se codifica todo lo especificado en el análisis y en el diseño. Para finalizar, se prueba su funcionamiento junto a todos los demás incrementos.

Ahora, para justificar el uso de esta metodología se van a dar unas ventajas y unas desvantajas sobre haber utilizado esta y no otra.

4.3 Ventajas y desventajas

Entre las ventajas que pueden encontrarse en la utilización de esta metodología podemos encontrar las siguientes:

- Mediante esta metodología se genera código operativo de forma rápida y en etapas tempranas del ciclo de vida.
- Es una metodología más flexible, por lo que se reduce el coste en el cambio del alcance y requisitos.
- Es más fácil probar y depurar en una iteración más pequeña (incremento) que en una etapa más grande (en una metodología sin incrementos).
- Es más fácil gestionar riesgos (incremento a incremento).
- Cada iteración es un hito gestionado fácilmente.

Para el uso de esta metodología se requiere una experiencia para definir los incrementos y distribuir en ellos las tareas de forma proporcionada. Es por esto que algunos de los inconvenientes que aparecen en esta metodología son los siguientes:

- Cada fase de un incremento es rígida y no se superpone con las demás.
- Pueden surgir problemas referidos a la arquitectura del sistema porque no todos los requisitos se han reunido, ya que se supone que todos ellos se han definido al inicio.

Es por todo esto que la metodología de desarrollo utilizada ha sido la incremental.

CAPÍTULO 5

Contexto tecnológico

En este apartado se va a presentar el contexto tecnológico sobre el que se ha desarrollado la aplicación. Primero se van a presentar las herramientas (el editor de código y sus complementos) utilizadas. Después, se dará una amplia visión de las tecnologías utilizada.

5.1 ATOM - Editor de código

ATOM es un editor de código fuente de código abierto disponible en todos los sistemas operativos (macOS, Linux y Windows) con soporte para plug-ins escritos en Node.js y control de versiones Git integrado, desarrollado por GitHub.

La mayor parte de los paquetes tienen licencias de software libre y son desarrollados y mantenidos por la comunidad de usuarios, por lo que las posibilidades son ilimitadas.

ATOM está basado en Electron, un framework que permite crear aplicaciones de escritorio multiplataforma usando Chromium y Node.js. Está escrito en CoffeeScript (lenguaje que se compile a JavaScript) y Less (lenguaje que se compila a CSS).

Se ha elegido este y no cualquier otro editor de código por la facilidad de uso y la posibilidad de potenciarlo con la instalación de paquetes de código libre.

5.2 HTML

HTML (HyperText Markup Language) [9] hace referencia al lenguaje de marcado para la elaboración de páginas web. Es un estándar que sirve de referencia del software que conecta con la elaboración de páginas web en sus diferentes

versiones, define una estructura básica y un código para la definición de contenido de una página web, como texto, imágenes, vídeos, juegos, etc.

Se considera el lenguaje web más importante siendo su invención crucial en la aparición, el desarrollo y expansión de la World Wide Web (WWW). Es el estándar que se ha impuesto en la visualización de páginas web y es el que todos los navegadores actuales han adoptado.

Se ha utilizado (como bien dice su definición) para la definición de la estructura de la aplicación web en su totalidad.

5.3 JavaScript

JavaScript es un lenguaje de programación interpretado, dialecto del estándar ECMAScript. Es orientado a objetos, basado en prototipos, imperativo, débilmente tipado y dinámico.

Se utiliza principalmente en desarrollos en la parte del cliente (client-side) para regir las interacciones del usuario con la parte visual de la aplicación.

Tradicionalmente se venía utilizando en páginas web HTML para realizar operaciones y únicamente en el marco de la aplicación cliente, sin acceso a funciones del servidor. Actualmente es ampliamente utilizado también para enviar y recibir información del servidor junto con ayuda de tecnologías como AJAX. JavaScript se interpreta en el navegador al mismo tiempo que sus sentencias se van cargando junto al código HTML.

Se ha utilizado JavaScript puro para el manejo de las interacciones del usuario con la aplicación, para validar formularios, animar la aplicación, etc.

También se ha utilizado para otras funciones con los frameworks correspondientes.

5.4 Node.js

Node.js es un entorno JavaScript del lado del servidor, de código abierto, basado en eventos. Node ejecuta JavaScript utilizando el motor V8 de Google. Aprovechando este motor, Node proporciona un entorno de ejecución del lado del servidor que compila y ejecuta código JavaScript a velocidades increíbles.

Además de la alta velocidad de ejecución de JavaScript, la verdadera clave detrás de Node.js es el bucle de eventos. Para escalar grandes volúmenes de clientes, todas las operaciones intensivas de I/O en Node se llevan a cabo de forma asíncrona. Para evitar el enfoque tradicional para generar código asíncrono, Node mantiene un bucle de eventos que gestiona todas las operaciones asíncronas.

En MyEvents, Node.js es el encargado de mantener el servidor, recibiendo las peticiones desde el front-end de la aplicación, pidiendo los objetos a la BBDD y devolviendo la respuesta al front-end de nuevo.

5.5 Nodemon

Cuando desarrollamos código en el servidor con Node tenemos un problema cuando hacemos cambios, pues tenemos que parar la ejecución del script y volver a lanzarlo manualmente. Esto es claramente ineficiente.

Para solucionar este asunto existe Nodemon, cuya tarea es reiniciar el servidor automáticamente en cuestión de milisegundos.

Para instalar Nodemon:

```
npm install -g nodemon –save—dev
```

(-dev para que este paquete solo se instale en el entorno de desarrollo)

y para utilizarlo, en MyEvents hemos puesto (en el archivo package.json) en el apartado “start” (script que se ejecuta al lanzar el servidor) la orden:

```
nodemon index.js
```

lo que provoca que, al iniciar el servidor, se lance automáticamente este script y se mantenga a la espera de cualquier cambio en el código para reiniciar el servidor.

5.6 Express

Express.js, según sus creadores, es un *framework* de desarrollo de aplicaciones web minimalista y flexible para Node.js. Es robusto, rápido, flexible y muy simple. Entre otras características, ofrece Router de URL's (GET, POST, PUT, etc..) que permite recibir y mandar peticiones a la BBDD desde el front-end. También es un buen middleware via Connect (para conectarse a la BBDD existente).

Para instalarlo, simplemente:

```
npm install -g express --save
```

5.7 MongoDB

MongoDB es una base de datos (BBDD) orientada a documentos. Esto quiere decir que, en lugar de guardar los datos en registros, guarda los datos en documentos, y estos en colecciones. Estos documentos son almacenados en BSON (Binary JSON), que es una representación binaria de JSON.

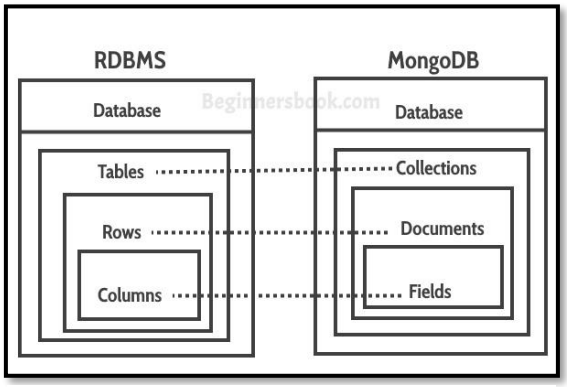


Figura 4.- Relación entre BBDD relaciones y MongoDB. Fuente: [4]

MongoDB es una BBDD de tipo no relacional y BBDD como MySQL son de tipo relacionales. Como puede verse en la figura 2, en MongoDB no existen las tablas, existen las colecciones. No existen las filas, existen los documentos. No existen las columnas, existen los campos de cada documento.

Una de las diferencias más importante con respecto a las bases de datos relacionales, es que no es necesario seguir un esquema. Los documentos de una misma colección pueden tener esquemas diferentes.

¿Cómo funciona MongoDB?

MongoDB [10][11][12] está escrito en C++, aunque las consultas se hacen pasando objetos en JSON como parámetro. Es algo lógico, dado que los propios documentos se almacenan en BSON. MongoDB viene de serie con una consola desde la que podemos ejecutar los distintos comandos. Esta consola está construida sobre JavaScript, por lo que las consultas se realizan utilizando este lenguaje.

En este caso he utilizado Robo3T, una interfaz gráfica que se conecta a la base de datos de MongoDB y te permite visualizar los datos, crearlos y editarlos de una forma muy cómoda.

Para facilitar el manejo de la BBDD MongoDB desde JavaScript he utilizado el framework *mongoose*, el más utilizado para este fin. Describiremos este framework en el siguiente punto.

Como extra, hay que apuntar que hay que lanzar el script demonio de MongoDB antes de operar con la base de datos. Para ello, hay que dirigirse al directorio /bin dentro de los archivos descargados de MongoDB (o incluir la ruta a esa carpeta en $PATH) y ejecutar el siguiente comando:

```
./mongod
```

5.8 Mongoose

Mongoose es una biblioteca de JavaScript que permite definir esquemas con datos fuertemente tipados. Una vez que se define un esquema, Mongoose permite

crear un modelo basado en un esquema específico. Un modelo se asigna a un documento MongoDB a través de la definición del esquema del modelo.

Una vez que haya definido los esquemas y modelos, Mongoose contiene muchas funciones diferentes que le permiten validar, guardar, eliminar y consultar sus datos utilizando las funciones comunes de MongoDB.

Para instalarlo, hay que dirigirse a la carpeta del proyecto de Node.js y ejecutar lo siguiente:

```
npm install mongoose --save
```

5.9 Body-parser

Body-parser es una librería para convertir las peticiones al backend a formato JSON. Es decir, los datos enviados por POST al servidor son formateados a JSON por parte del body-parser. Para instalar, hay que dirigirse a la carpeta del proyecto de Node.js y ejecutar lo siguiente:

```
npm install body-parser --save
```

5.10 Connect-multiparty

Connect-multiparty es una librería que permite subir archivos al backend de la aplicación (MongoDB) y trabajar con la variable files de Node.js. Para instalar, dirigirse a la carpeta del proyecto de Node.js y ejecutar lo siguiente:

```
npm install connect-multiparty --save
```

5.11 Bootstrap

Bootstrap es un framework web o conjunto de herramientas de código abierto para el diseño de sitios y aplicaciones web. Contiene plantillas de diseño con tipografía, formularios, botones, cuadros, menús de navegación y otros elementos de diseño basados en HTML y CSS, así como extensiones de JavaScript adicionales.

5.12 Angular

Angular es un *framework* para aplicaciones web desarrollado en *TypeScript*, mantenido por *Google*, que se utiliza para crear y mantener aplicaciones web de una sola página. Su objetivo es aumentar las aplicaciones basadas en navegador con capacidad de Modelo Vista Controlador (MVC).

La biblioteca lee el HTML que contiene atributos de las etiquetas personalizadas adicionales, entonces obedece a las directivas de los atributos personalizados, y une las piezas de entrada o salida de la página a un modelo representado por las variables estándar de *JavaScript*. Los valores de las variables de *JavaScript* se pueden configurar manualmente, o ser recuperados de JSON's estáticos o dinámicos.

CAPÍTULO 6

Arquitectura del sistema

En este apartado se va a dar una visión global de la arquitectura del sistema de la aplicación, comenzando por los elementos que participan en ella, siguiendo con las comunicaciones entre estos elementos y acabando con los formatos de la información que se envían.

6.1 Elementos que participan

En este subapartado vamos a hablar sobre los elementos que conforman la aplicación. Estos son (como pueden verse en la figura 3): una aplicación web cliente, un servidor web y una API Rest que se comunica con la base de datos (MongoDB). Primero hablaré sobre la aplicación web, su función y con los elementos que se conecta. Después, hablaré del servidor web que maneja la BBDD de la aplicación. Para finalizar, detallaré la API Rest diseñada y utilizada para dicho fin. Un esquema de todos los elementos es el siguiente:

Figura 5.- Elementos que conforman la aplicación. Fuente: [5]

6.1.1 Aplicación web (cliente)

La aplicación web es el front-end, la parte visible en el navegador, con la que interactuamos. Está construida con HTML, CSS, TypeScript, Angular y Bootstrap. HTML ha sido utilizado como he dicho anteriormente para definir la estructura de la página, que junto a Angular ha servido para ser la base de la parte visible. TypeScript, a su vez, ha sido el lenguaje utilizado para interactuar con el servidor web y este, a su vez, con la BBDD. CSS y Bootstrap, por último, han sido utilizados para maquetar todo más fácilmente. Bootstrap para poner la base y CSS para pequeños retoques.

6.1.2 Servidor web

El servidor web es el encargado de recibir las peticiones desde el cliente, procesarlas y mandarlas a la BBDD. De forma análoga, es el encargado de recibir las respuestas de la BBDD, procesarlas y devolverlas al cliente. El servidor del proyecto está escrito y desarrollado con Node.js.

6.1.3 API Rest

Para manejar las conexiones a la base de datos he creado una API Rest con JavaScript para que el cliente (los diferentes componentes) puedan realizar las gestiones con los objetos de la BBDD.

6.1.4 Base de datos (MongoDB)

La base de datos utilizada es MongoDB por su flexibilidad y facilidad de uso. En mi caso he utilizado Robo3T (interfaz visual de gestión de la BBDD MongoDB) para facilitar (se puede ver la organización de los archivos en la figura 7) la gestión de los objetos existentes en la BBDD. Se pueden observar todos los documentos guardados (figura 6), editarlos, eliminarlos, etc.

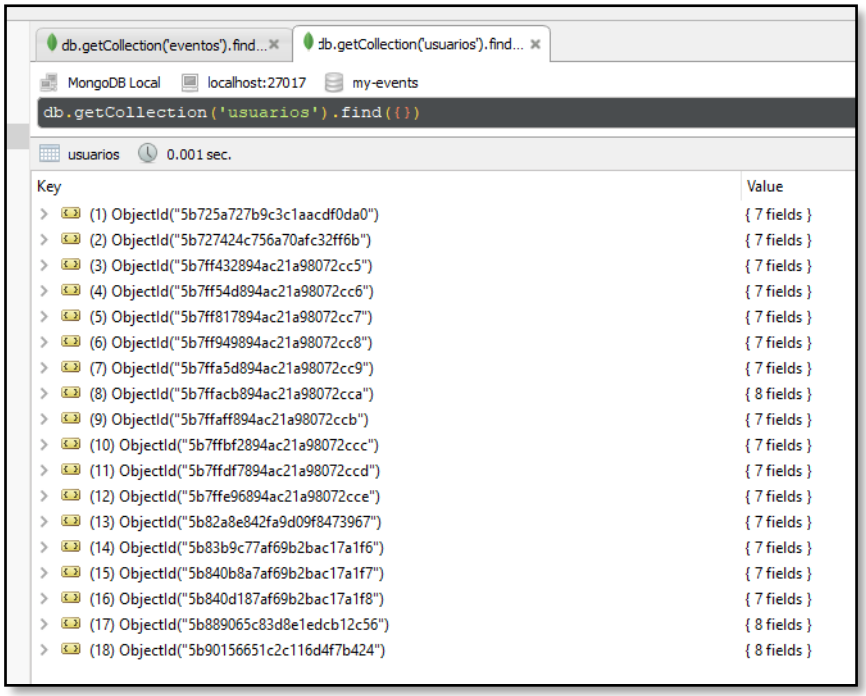


Figura 6.- Robo3T (Visor de documentos).

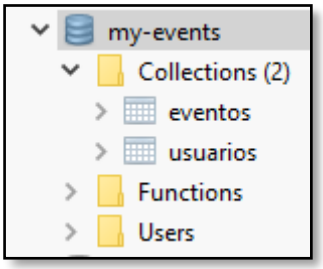


Figura 7.- Robo3t (Organización de Robo3T)

6.2 Comunicaciones entre elementos

Las comunicaciones que hay son varias y entre varios elementos. Hay comunicaciones del front-end al servidor, del servidor a la BBDD y viceversa.

6.2.1 Conexiones HTTP

La conexión HTTP sucede entre Angular y Node.js. Una vez recibida en el servidor por Node, esta petición se hará servir de Express (y de mongoose) para obtener lo deseado de la BBDD y volver a Angular con los datos. La ruta un poco más explicada:

- Angular → Node.js
    - En cualquiera de nuestros componentes (como hemos mostrado en el ejemplo de la API) crearemos una conexión HTTP mediante el servicio correspondiente al servidor.

- Node.js → Express
    - Express gestiona las rutas de los endpoints, por lo que una vez recibida la petición por parte del servidor, este se la pasa a Express para que la gestiona con sus rutas.

- Express → MongoDB
    - Express y Mongoose se conectan a la BBDD y reciben los datos solicitados. Una vez tienen los datos, se realiza el mismo camino pero a la inversa.

6.3 Formato de la información que se envía

Toda la información que se envía en las conexiones HTTP definidas anteriormente son objetos JSON. Por lo que la información que se envía siempre va a ser JSON. Hay objetos JSON de tipo Evento y objetos JSON de tipo Usuario. A continuación, se describe cada tipo de objeto JSON.

6.3.1 Descripción objetos JSON

Como he dicho en el apartado de la BBDD (MongoDB), he utilizado el programa Postman para observar los resultados de las peticiones a la API. Así, vamos a detallar los formatos de todos los objetos JSON que se envían y se reciben en las conexiones. Es decir:

Formato objeto JSON tipo Evento


```
"evento": {
    "_id": "5b8609058154c82698975e56",
    "titulo": "JS Conf 2019",
    "fecha": "2019-05-10T22:00:00.000Z",
    "lugar": "Museo Reina Sofia",
    "categoria": "Evento",
    "tematica": "Informática",
    "organizador": "JavaScript Valencia",
    "imagen": "aCBCNcQxqjE6mX82SDPCc6f6.png",
    "__v": 0,
    "descripcion": "Evento sobre JavaScript anual"
}
```


Figura 8.- Formato JSON Evento

El formato del objeto JSON tipo Evento puede verse en la figura 8. Contiene todos los atributos del objeto más el identificador de cada documento de la BBDD de MongoDB.

Formato objeto JSON tipo Usuario


```
"usuario": {
    "_id": "5b889065c83d8e1edcb12c56",
    "nickname": "josep.pedrerol",
    "password": "chiringuito",
    "nombre": "Josep",
    "apellidos": "Pedrerol",
    "descripcion": "Experto en comunicación y especialista en programas deportivos",
    "imagen": "B1Y6FXxuASvo4ICzcvbX0q-q.png",
    "__v": 0
}
```


Figura 9.- Formato JSON Usuario

El formato del objeto JSON puede observarse en la figura 9. Contiene todos los atributos del objeto más el identificador de cada documento de la BBDD de MongoDB.

CAPÍTULO 7

Diseño de la aplicación

En este apartado daremos un repaso sobre el diseño de la aplicación desarrollada. Primero, enseñaremos las interfaces de la aplicación. Una vez presentadas las interfaces, acabaremos describiendo los modelos de objetos que tienen la BBDD, explicando los atributos más importantes.

7.1 Diseño de interfaces

A continuación, ilustraremos las interfaces de todas las áreas de la aplicación junto a una pequeña explicación de cada una.

7.1.1 - Pantalla principal (sin autenticar)

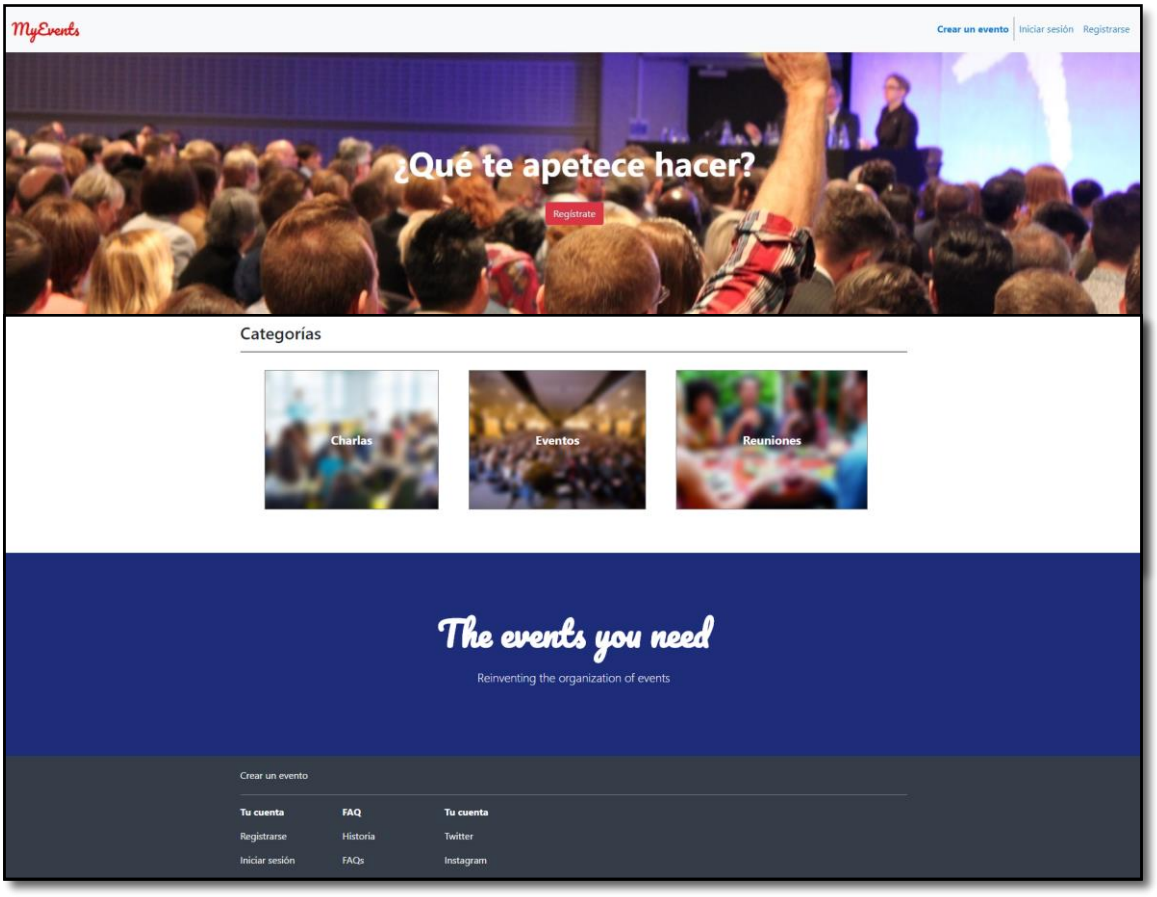


Figura 10.- Home de MyEvents

Como se puede observar en la figura 10, el home de la aplicación (no estando autenticados) goza de simpleza y una gran elegancia. Arriba del todo, en la barra de navegación, aparece el logo de MyEvents junto a (a la derecha del todo) las opciones de iniciar sesión y registrarse.

Justo bajo aparece el jumbotron que da la bienvenida a la página y te invita a registrarte. Nada más acabar el jumbotron aparecen los eventos más populares, ordenados en una organización de 3x2. Cada card de un evento da acceso a la pantalla de información de cada uno.

Al acabar la sección de eventos populares, aparece la sección de categorías, que te da acceso a los eventos de cada categoría en la que hagas click. Para acabar el home, el footer, presente en toda la página. Te da opciones generales.

7.1.2 - Pantalla principal (autenticados)

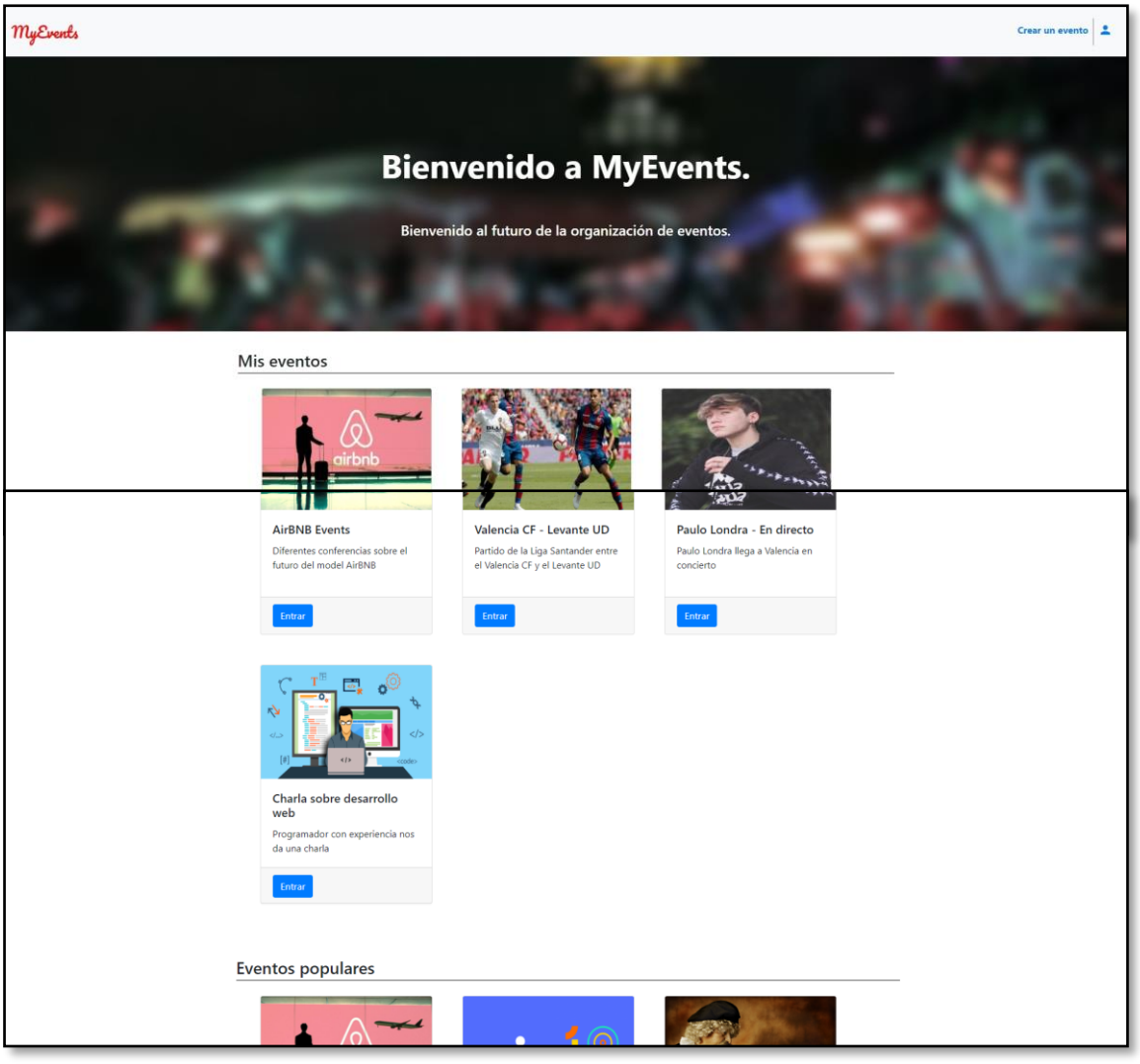


Figura 11.- Home MyEvents (autenticados)

Las diferencias entre el home autenticado (figura 11) y el no autenticados (figura 10) es que en este, se tiene acceso a crear evento directamente (ya que estamos logueados) mientras que en el otro nos redirige a la pantalla de login. También se tiene acceso (mediante el icono) al apartado de perfil.

También está el jumbotron (aunque con un diseño diferente) y los eventos creados por el usuario con el que hemos iniciado sesión. A partir de esta sección, los eventos populares y todo lo demás es lo mismo (eventos populares, categorías y footer).

7.1.3 - Login

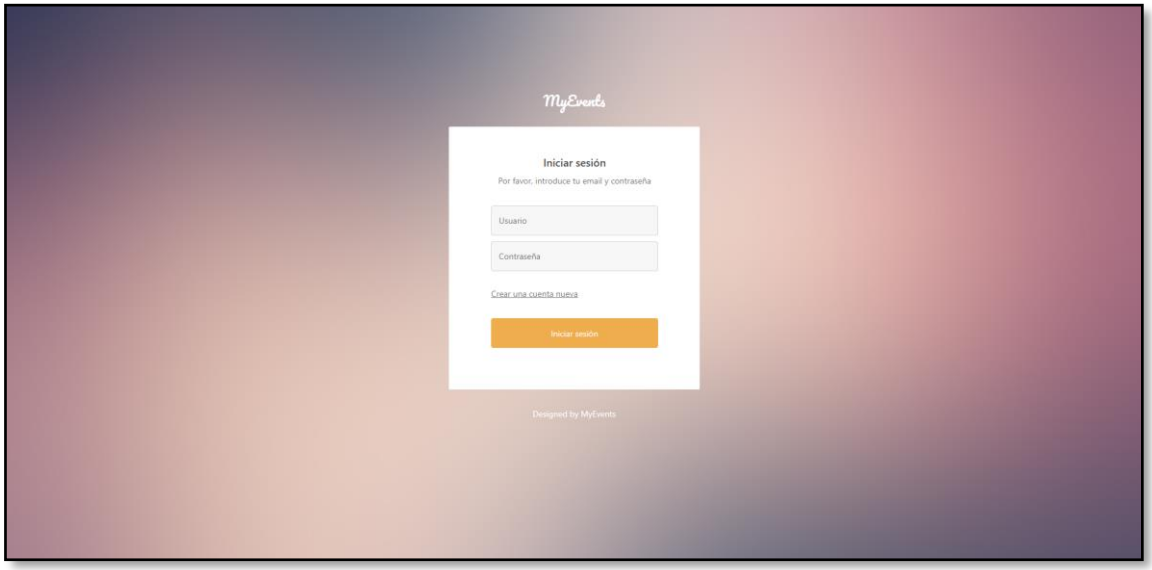


Figura 12.- Login de MyEvents

En la pantalla de login (figura 12) se puede rellenar el usuario y la contraseña (previamente creados). También existe un enlace para ir a la pantalla de registro para poder crear un nuevo usuario.

7.1.4 - Registro

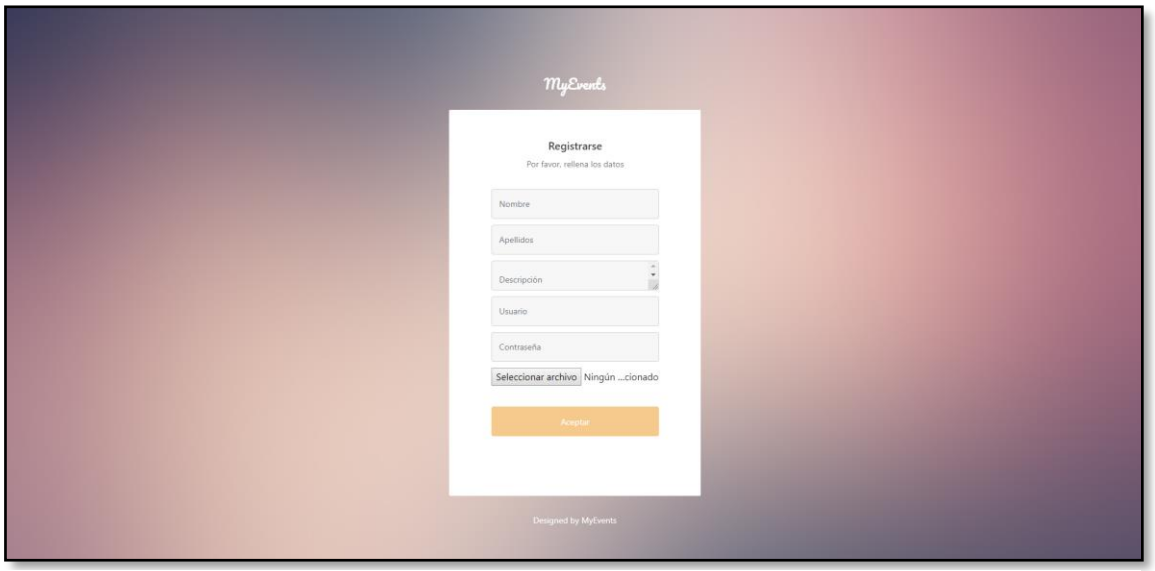
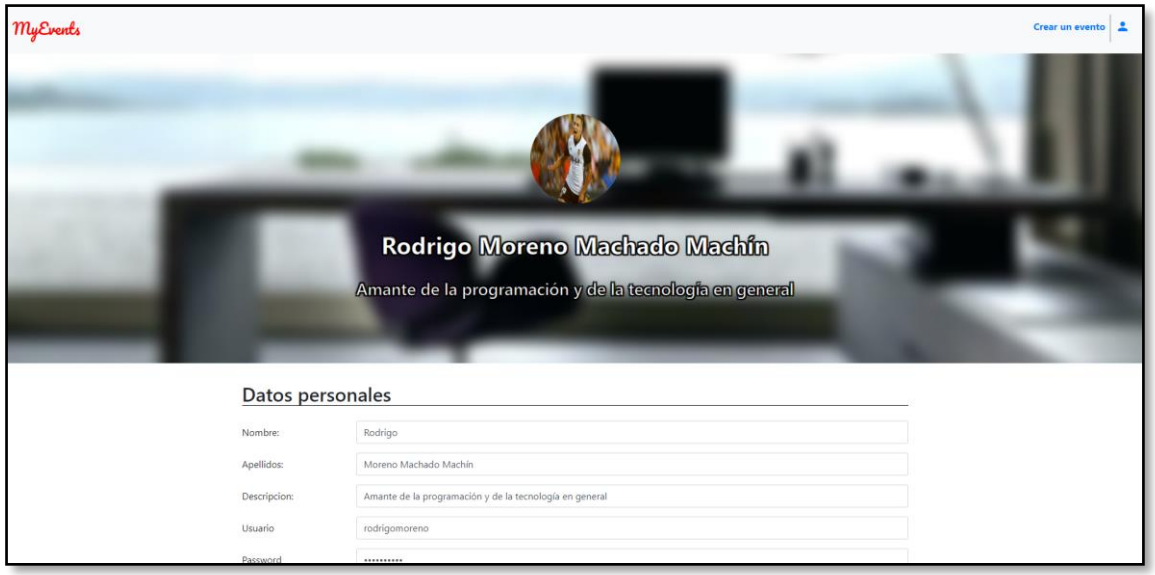


Figura 13.- Registro de MyEvents

En la pantalla de registro (figura 13) se pueden rellenar los campos de nombre, apellidos, descripción, usuario y contraseña. También está el campo para seleccionar la imagen de perfil que se desee y el botón para registrarse con los datos puestos.

7.1.5 - Perfil



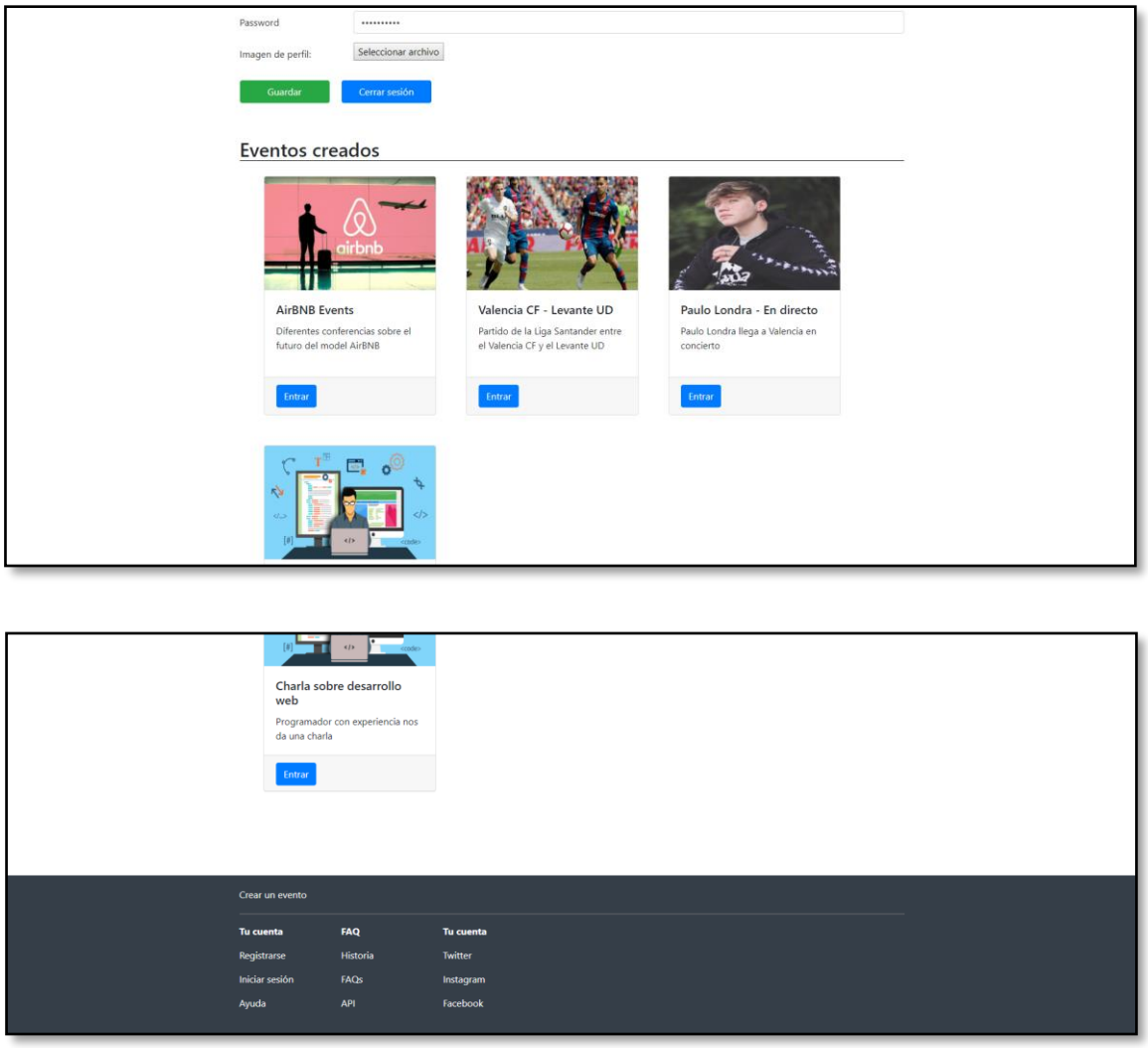


Figura 14.- Perfil en MyEvents

Como se puede observar en la figura 14, la pantalla de perfil del usuario contiene la barra de navegación ya presente en el home autenticado junto a un jumbotron que contiene la imagen de perfil del usuario, su nombre y su descripción.

Debajo de esto aparecen los datos del usuario en un formulario, junto a dos botones para que se puedan modificar (y guardar) los datos y cerrar sesión. Debajo de los datos aparece la sección de eventos creados por el usuario, para una fácil navegación a ellos.

Debajo de todo, el footer, presente en todas las páginas.

7.1.6 - Detalle del evento

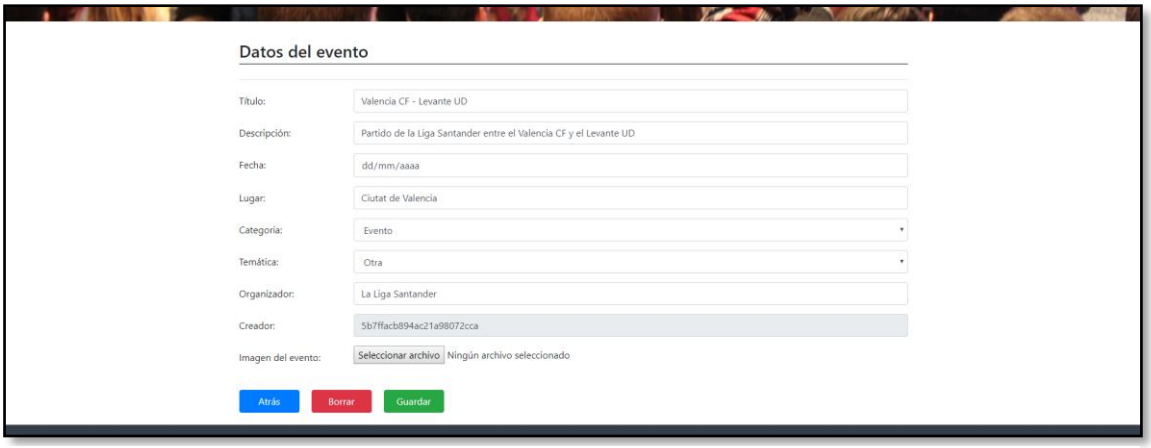


Figura 15.- Detalle del evento si somos creadores

En la pantalla a la que accedemos (figura 15) si hacemos clic en el botón de 'Entrar' de la tarjeta de cada evento, aparecen los datos en un formulario para poder actualizar los datos (o borrar el evento directamente). Si el evento al que accedemos no es creado por nosotros, aparecen los datos del evento y nada más (figura 16), sin posibilidad de editar o borrar (lógicamente).

Figura 16.- Detalles del evento si no somos creadores

7.1.7 - Crear un evento

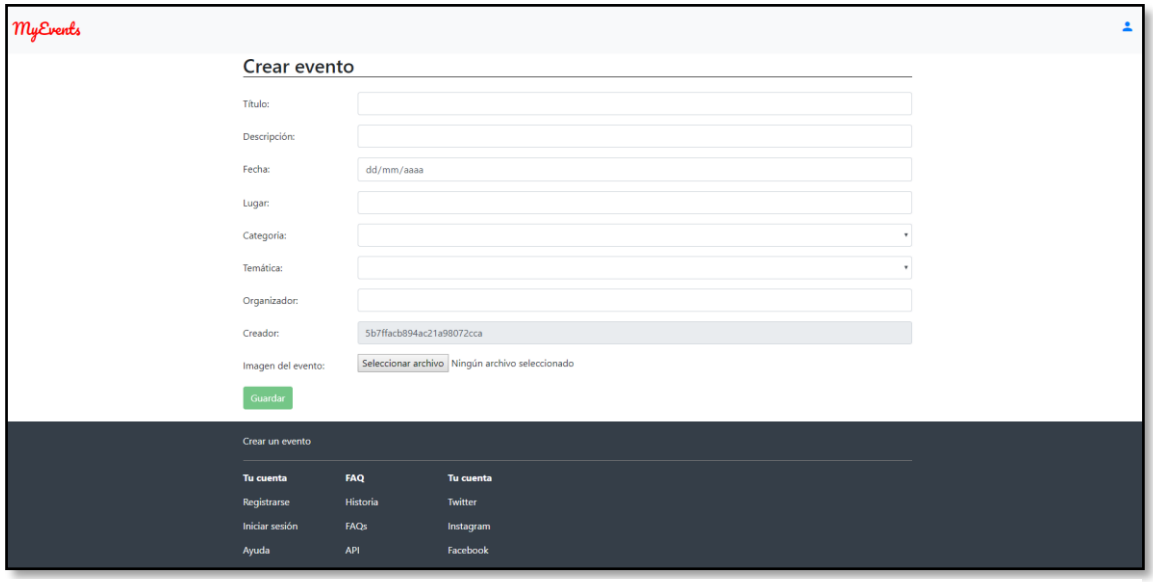


Figura 17.- Formulario para crear evento

En la pantalla de crear evento (figura 17) aparecen los datos a introducir para crear un evento, incluyendo el título, la descripción, la fecha, el lugar, un desplegable para elegir la categoría, otro para elegir la temática y el organizador. También existe un campo desabilitado que indica el id del usuario justo arriba del campo para seleccionar la imagen del evento. El botón de guardar finaliza esta sección. El footer, como siempre, marca el final de la página.

7.2 Modelo de la BD que usa el servidor

La BBDD (MongoDB) almacena documentos de dos tipos (evento y usuario), cada una con sus campos correspondientes. A continuación, detallaremos los modelos de cada tipo de documento que almacena la BBDD.

7.2.1 - Tipo evento

evento_model.js

```
'use strict'

var mongoose = require('mongoose');
var Schema = mongoose.Schema;

var EventoSchema = Schema({
  titulo: String,
  descripcion: String,
  fecha: Date,
  lugar: String,
  categoria: String,
  tematica: String,
  organizador: String,
  imagen: String,
  idCreador: String
});

module.exports = mongoose.model('Evento', EventoSchema);

```

Figura 18.- Modelo Evento en BBDD

El modelo de cada evento que posee la BBDD es el que se observa en la figura 18. Guarda el título del evento, la descripción de este, la fecha, el lugar, la categoría, la temática, el organizador, la imagen y el id del creador. Todo esto se basa en los Schemas de mongoose. Luego se exporta y se importa en todos los archivos necesarios.

7.2.2 - Tipo usuario

usuario_model.js

```
'use strict'

var mongoose = require('mongoose');
var Schema = mongoose.Schema;

var UsuarioSchema = Schema({
  nickname: {type: String, required: true, unique: true},
  password: {type: String, required: true},
  nombre: {type: String},
  apellidos: {type: String},
  descripcion: {type: String},
  imagen: {type: String}
});

module.exports = mongoose.model('Usuario', UsuarioSchema);

```

Figura 19.- Modelo Usuario en BBDD

El modelo de cada usuario que posee la BBDD es el que se observa en la figura 19. Guarda el nick del usuario (que debe ser obligatorio y único), la contraseña (que debe ser obligatorio), el nombre, los apellidos, la descripción y la imagen. Todo esto se basa en los Schemas de mongoose. Luego se exporta y se importa en todos los archivos necesarios.

CAPÍTULO 8

Implementación

En este apartado explicaremos, en primer lugar, la amplia estructura de archivos utilizada para desarrollar el proyecto, con imágenes explicativas. Una vez explicada la estructura, para finalizar el apartado, daremos una pincelada sobre varios ejemplos de código de todas las partes del proyecto.

8.1 Estructura del proyecto

Primero, daremos un vistazo a la estructura de archivos de la parte visual de la aplicación cliente (front-end), explicando el contenido de cada carpeta. Una vez hayamos acabado con el front-end, pasaremos a dar un vistazo a todas las carpetas que forman el servidor web (el back-end), desde los controllers hasta los modelos. Para continuar, daremos unas pinceladas sobre la API Rest que consistirán en una explicación sobre como funciona nuestra API. Finalizaremos con una pequeña explicación sobre como funciona nuestro proyecto y como hemos trabajado con MongoDB.

8.1.1 Front-end de la aplicación web (interfaces)

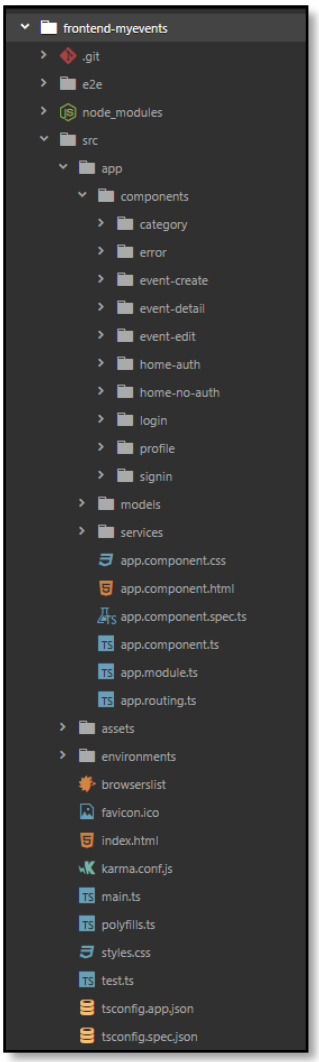


Figura 20.- Estructura archivos front-end

Como se puede observar en la figura 20, la parte del front-end y del back-end están totalmente separadas en archivos y carpetas diferentes para una mayor y más fácil organización. Así, podemos observar los apartados del código de la aplicación web:

1.- **Carpeta de cada componente (app/components)**: Todos los componentes que representan los apartados de nuestra web están separados por carpetas (un ejemplo de una carpeta puede verse en la figura 21). Cada carpeta de cada componente tiene los siguientes ficheros:

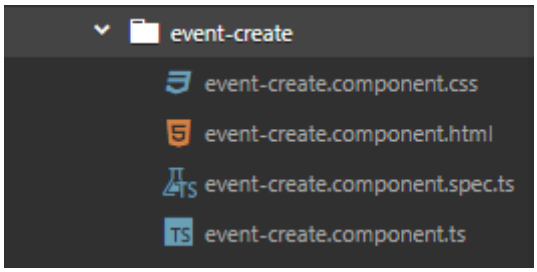


Figura 21.- Carpeta del componente crear-evento

- El archivo .css para darle estilos específicos a ese componente.
- El archivo .html para definir la estructura de ese componente.
- El archivo .spec.ts para definir tests unitarios para los archivos .ts.
- El archivo .ts para definir las interacciones, etc del componente.

2.- **Carpeta de los modelos (app/models)**: Cada 'modelo' es una definición de los tipos de objetos que puede devolver la BBDD. En nuestro caso, la aplicación solo define dos modelos (y la BBDD solo maneja dos tipos de objetos), usuario y evento. Es decir:

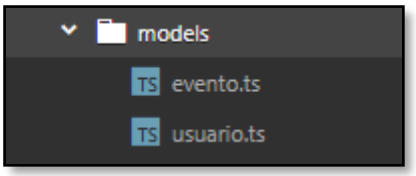


Figura 22.- Carpeta modelos front-end

Cada archivo (se pueden ver en la figura 22) contiene la definición de los atributos de cada tipo de objeto que serán devueltos por el servidor provenientes de la BBDD.

3.- **Carpeta de los servicios (app/services)**: En la carpeta de servicios (figura 23) se encuentran los archivos .ts que manejan los servicios de los que se pueden hacer servir los componentes para conectar con el servidor y a su vez con la BBDD y así, manejar los datos. En nuestro caso, tenemos 6 servicios (1 por cada tipo de objeto (2), 1 por cada servicio de subir imágenes de cada tipo de objeto

(2), 1 para manejar el login de los usuarios (1) y 1 último especial que se detallará al final de este apartado.). Es decir:

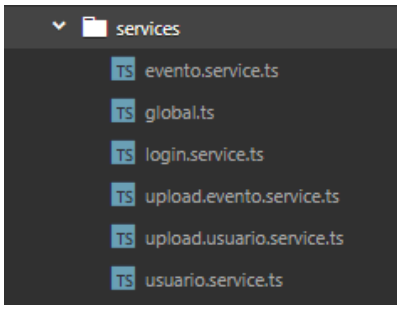


Figura 23.- Carpeta de servicios

Como podemos comprobar tenemos varios servicios:

- **Evento.service**: Es el servicio que maneja todas las operaciones que tengan que ver con objetos de tipo Evento: Obtener un evento, obtener todos los eventos, actualizar un evento, borrar un evento, etc.

- **Evento.upload.service**: Es el servicio encargado de asignar imágenes a atributos de objetos de tipo Evento (atributo imagen de un objeto de tipo Evento, por ejemplo).

- **Usuario.service**: Es el servicio que maneja todas las operaciones que tengan que ver con objetos de tipo Usuario: Crear un usuario, modificar un usuario, etc.

- **Usuario.upload.service**: Es el servicio encargado de asignar imágenes a atributos de objetos de tipo Usuario (atributo foto de perfil, por ejemplo).

- **Login.service**: Es el servicio encargado de manejar el login de los usuarios.

- **Global**: Este "servicio" es especial, pues no es un servicio en si. Es el archivo encargado de mantener las diferentes variabes que definen los endpoints para los APIs de los diferentes tipos de objetos.

```
export var GlobalEventos = {
  urlEventos: 'http://localhost:3500/api-eventos/'
};

export var GlobalUsuarios = {
  urlUsuarios: 'http://localhost:3500/api-usuarios/'
};
```

Figura 24.- Endpoints Global.ts

El servidor está escuchando en la ruta http://localhost:3500 y hoy 2 endpoints diferentes (figura 24). Así, solo tendremos que importar una de las dos variables (dependiendo del endpoint con el que queramos trabajar) o las dos. Una vez importadas, tendremos que asignar variables a el valor correspondiente a la variable importada. Por ejemplo:

```
this.url = GlobalEventos.urlEventos;
```

Así, podremos conectarnos a la BBDD (al endpoint que maneja los eventos) con la variable url.

4.- **Resto de archivos de la carpeta app (app/..)**: Entre estos archivos (que se pueden ver en la figura 25) se encuentran los correspondientes a la definición del principal componente de angular (app.component) junto al archivo que maneja los módulos. También se incluye el archivo que guarda las rutas de los diferentes componentes de la aplicación.

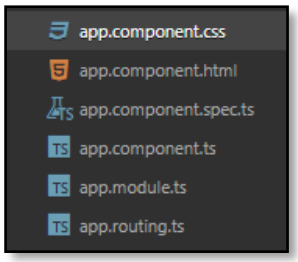


Figura 25.- Resto de archivos del front-end

5.- **Carpetas de assets y de environments**: En estas carpetas (figura 26) se encuentran, correspondientemente:

- **Assets/img**: Todas las imágenes que hay de serie en la aplicación, aquellas que no manejan los usuarios (banners, jumbotron, etc).

- **Environments**: Carpeta donde se manejan los entornos (en nuestro caso, se mantiene el entorno de producción).

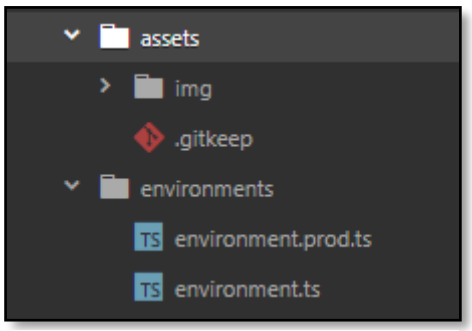


Figura 26.- Carpetas assets y environments

6.- **Resto de archivos de la carpeta src y de fuera de ella**: El resto de los archivos (figura 27) son archivos de configuración de la aplicación y de angular.

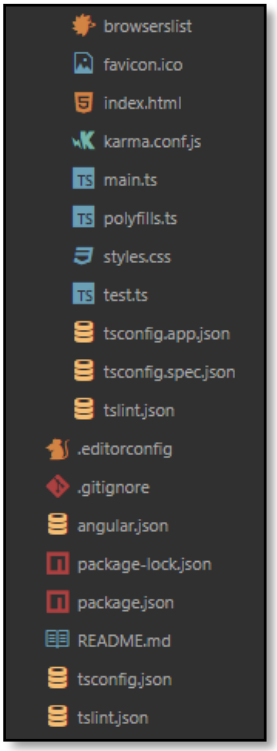


Figura 27.- Resto de archivos back-end

8.1.2 Back-end (servidor web)

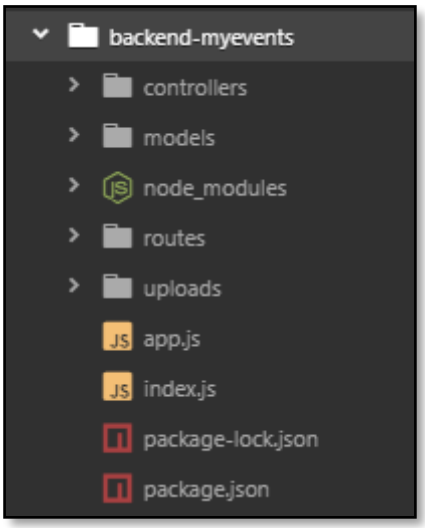


Figura 28.- Carpetas back-end

Como se ha visto en el front-end (aplicación web), en el back-end (servidor web) también está todo organizado en carpetas (que se pueden ver en la figura 28).

1.- **Carpeta controllers**: En esta carpeta (figura 29) tenemos los dos archivos que manejan los controladores de cada tipo de objeto de la BBDD (usuario y evento). Estos controladores se encargan de realizar las peticiones a la BBDD y devolver el resultado deseado. Así se organiza la carpeta:

Figura 29.- Carpeta controllers

2.- **Carpeta models**: En la carpeta models (figura 30) se encuentran los modelos de cada tipo de objeto de la BBDD (evento y usuario) que se corresponden con los modelos que hay en el front-end de la aplicación web. Es decir:

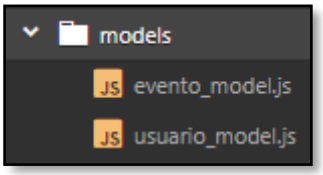


Figura 30.- Carpeta models

3.- **Carpeta routes**: En la carpeta routes (figura 31) se encuentran los dos archivos correspondientes a las rutas de cada endpoint encargadas de manejar las distintas conexiones HTTP (GET, POST, PUT, DELETE) conectándolas con el método correspondiente del controller (BBDD). Es decir:

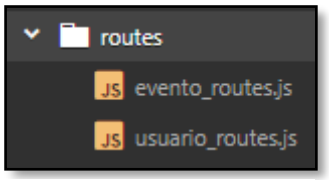


Figura 31.- Carpeta routes

4.- **Carpeta uploads (y resto de archivos)**; En la carpeta uploads (figura 32) se suben las imágenes correspondientes a cada tipo de objeto de la BBDD, diferenciados en dos carpetas (evento y usuario). Es decir:

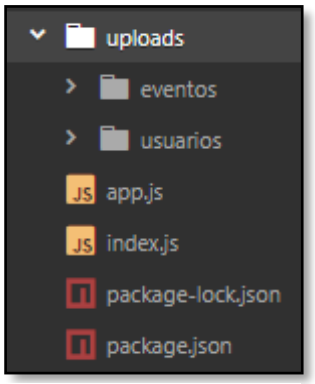


Figura 32.- Carpeta uploads y resto de archivos

El resto de los archivos son archivos de configuración del servidor.

8.1.3 API Rest

Como se ha explicado en el apartado 6.1.3, hemos creado una API para gestionar las interacciones con la BBDD. Más específicamente, hay una api 'dividida en dos' como se ha explicado (api-usuarios y api-eventos, aunque básicamente hacen las mismas operaciones, una especie de CRUD..).

Para mostrar un ejemplo de cómo crear una funcionalidad de la API Rest se va a ejemplificar (por ejemplo) la creación de la función de obtener todos los eventos. Es decir, paso por paso:

1.- En primer lugar hay que crear el método en el controlador correspondiente (como es obtener todos los eventos, deberemos crearlo en el archivo evento_controller.js).

Mediante mongoose (utilizando su API) utilizaremos el modelo del objeto correspondiente (en esto caso, Evento) junto a la función que deseamos realizar (en este caso es la función Evento.find({})). Observamos que si no le pasamos nada entre los corchetes (parámetros) nos va a devolver todos los campos de todos los documentos. Una vez encontrados los eventos queremos que nos los ordene por la fecha, por lo que añadimos .sort('-fecha').exec();. Lo que hace esto es ordenador todos los Eventos una vez obtenidos de la BBDD y ejecutar su ordenación. Después, gestionamos los códigos de satisfacción o de error y nos quedaría el método que aparece en la figura 33.

```
getEventos: function(req, res){

  Evento.find({}).sort('-fecha').exec((err, eventos) => {

    if (err) return res.status(500).send({message: 'Error al devolver los datos.'});

    if (!eventos) return res.status(404).send({message: 'No existen eventos que mostrar.'});

    return res.status(200).send({eventos});

  });

},
```

Figura 33.- Método getEventos de la API

Una vez tengamos el método creado en el controlador, tenemos que crear su ruta correspondiente (mediante GET en este caso) en el archivo evento_routes.js (figura 34).

```
router.get('/eventos', EventoController.getEventos);
```

Figura 34.- Ruta para método getEventos

Como se puede observar, utilizamos el método router de Express para gestionar las rutas del servidor para las conexiones HTTP. Así, una vez creada la ruta en el archivo de rutas, deberemos cargar el archivo de rutas en el archivo app.js (base del servidor). Esto se puede ver en la figura 35.

```
//Cargar archivos de rutas
var evento_routes = require('./routes/evento_routes');
```

Figura 35.- Cargar rutas en app.js

Una vez cargado el archivo de rutas, deberemos definir el endpoint de los eventos para la API (también en el archivo app.js). Dicha definición se puede ver en la figura 36.

```
//Definir rutas
app.use('/api-eventos', evento_routes);
```

Figura 36.- Definición endpoint eventos en app.js

Ahora, cada vez que queramos utilizar una de las rutas creadas, deberemos hacer una petición a la url /api-eventos/ruta-de-la-peticion.

Una vez cargas y definidas las rutas, deberemos crearnos un servicio para cada tipo de datos en la BBDD (evento.service.ts en este caso) y desde ahí, crear un método que haga una petición HTTP a cualquier de las rutas. El método del servicio correspondiente a getEventos se puede ver en la figura 37.

```
getEventos(): Observable<any>{
  let headers = new HttpHeaders().set('Content-Type', 'application/json');
  return this._http.get(this.url+'eventos', {headers: headers});
}
```

Figura 37.- Método getEventos en el servicio para los eventos

Una vez definido el método en el servicio, deberemos consumirlo desde el archivo .ts de cualquiera de nuestros componentes (puede verse un ejemplo de esto en la figura 38). En nuestro caso, lo necesitaremos en nuestro componente home. Así:

```
getEventos(){
  this._eventoService.getEventos().subscribe(
    response => {
```

Figura 38.- Método getEventos en el componente

Y una vez recibamos los datos en la respuesta, podemos manejarlos a nuestro antojo (siempre respetando los atributos definidos en el modelo). Así se creó una funcionalidad de la API Rest de MyEvents.

8.1.4 Base de datos (MongoDB)

Para comprobar el funcionamiento de la API junto a la BBDD (MongoDB) sin necesidad de implementar la funcionalidad en el front-end de la aplicación, se ha utilizado el programa Postman, que permite consumir APIs y recoger los datos en la misma interfaz. Su pantalla principal puede verse en la figura 39.

Figura 39.- Pantalla principal de Postman

8.2 Ejemplos de código

En este último apartado comenzaremos comentando el código del front-end (interfaces), después el código de la conexión al servidor (services) y acabaremos con el código del servidor web en si mismo y de la BBDD.

8.2.1 Front-end de la aplicación web (interfaces)

Daremos algunos ejemplos de código (no todos porque sería una extensión demasiado grande) de las interfaces de los componentes. Primero, enseñaremos la forma en la que se cargan los eventos creados por el usuario en el home autenticados.

Home

```
<!-- Mis eventos + Eventos populares + Categorias -->
<div class="container">
  <h3 class="titulito">Mis eventos</h3>

  <ul>
    <div class="row">
      <li *ngFor="let evento of eventosCreador" class="evento">
        <div class="col-sm-4">
          <div class="card" style="width: 18rem;">
            <img class="card-img-top" src="{{url+'get-image/'+evento.imagen}}" *ngIf="evento.imagen">
            <div class="card-body">
              <h5 class="card-title">{{evento.titulo}}</h5>
              <p class="card-text">{{evento.descripcion}}</p>
            </div>
            <div class="card-footer">
              <small class="text-muted">
                <a [routerLink]="['/event-detail', evento._id]" class="btn btn-primary largo">Entrar</a>
              </small>
            </div>
          </div>
        </div>
      </li>
    </div>
  </ul>
```

Figura 40.- Pequeño trozo del componente home

Como puede verse en la figura 40, mediante Bootstrarp (y su componente de cartas) se maqueta la disposición de las cartas en el home. Después, con la directiva *ngFor de Angular, recorremos el array de Eventos obtenido en el archivo .ts del componente. También utilizamos las directivas [routerLink] que hace el trabajo de redirigirnos a otro componente especificado.

```
ngOnInit(){
  console.log("Estamos en el home autenticados.");
  this.sesionUsuario = JSON.parse(localStorage.getItem('usuario'));
  console.log(this.sesionUsuario);
  this._route.params.subscribe(params => {
    let id = params.id;

    this.getEventosCreador(id);
  });

  this.getEventos();
}

getEventosCreador(id){
  this._eventoService.getEventosCreador(this.sesionUsuario._id).subscribe(
    response => {
      this.eventosCreador = response.eventos;
    },
    error => {
      console.log(<any>error);
    }
  );
}
```

Figura 41.- Archivo .ts del componente home

En el método ngOnInit() (método que se ejecuta al iniciarse cada componente obligatoriamente y que podemos ver en la figura 41) recogemos el id del usuario de los parámetros de la ruta y lo utilizamos para llamar al método de recoger los eventos del creador especificado con el id recogido.

El método de obtener los eventos de un creador especificado (por su id) utiliza el servicio del evento para llamar al mismo evento y luego suscribirse al resultado y asignar la respuesta recibida al array de eventos (que se recorrerá en el array del *ngFor).

Login

```
<form id="Login">

  <div class="form-group">
    <input type="text" [(ngModel)]="usuario.nickname" name="nickname" class="form-control" placeholder="Usuario">
  </div>

  <div class="form-group">
    <input type="password" [(ngModel)]="usuario.password" name="password" class="form-control" placeholder="Contraseña" >
  </div>

  <div class="forgot registry">
    <a [routerLink]="['/signin']">Crear una cuenta nueva</a>
  </div>

  <button type="submit" class="btn btn-primary" (click)="validateLogin();">Iniciar sesión</button>

</form>
```

Figura 42.- Componente login

Para el componente de login (figura 42) utilizamos la directiva [(ngModel)] para asignar los datos introducidos en cada campo con el objeto de tipo Usuario que se utilizará para validar el inicio de sesión al hacer click (llamando al método validateLogin() que puede verse en la figura 43) en el botón de iniciar sesión.

```
validateLogin(){
  if(this.usuario.nickname && this.usuario.password){
    this._loginService.validateLogin(this.usuario).subscribe(
      result => {
        console.log('Result: ', result);
        if(result['status'] === 'success'){
          let usua = result['data'][0];
          console.log(usua);
          localStorage.clear();
          localStorage.setItem('usuario', JSON.stringify(usua));
          this._router.navigate(['/auth']);
        }else{
          alert('Wrong nickname and password')
        }
      },
      error => {
        console.log('Error: ', error)
      });
  }else{
    alert("Introduce nick y usuario");
  }
}
```

Una vez llamado al método, este utiliza el servicio de login para validar el usuario, y en función de si el resultado es satisfactorio o no, tomar un comportamiento u otro.

Perfil

```
40  <!-- Datos personales si usuario que accede es el mismo (editables) -->
41  <div class="container" id="container" *ngIf="sesionUsuario._id == usuario._id">
42
43    <h2 style="border-bottom: 1px solid black; margin-bottom: 25px;">Datos personales</h2>
44
45    <form #editUserForm="ngForm" (ngSubmit)="onSubmit(editUserForm)">
46
47        <!-- Nombre -->
48        <div class="form-group row margensito">
49          <label for="inputNombre" class="col-sm-2 col-form-label">Nombre:</label>
50          <div class="col-sm-10">
51            <input type="text" class="form-control" name="nombre" #nombre="ngModel" [(ngModel)]="usuario.nombre" required>
52          </div>
53        </div>
```

Figura 43.- Componente perfil

```
136  <!-- Datos personales si usuario que accede no es el mismo (no editables) -->
137  <div class="container" id="container" *ngIf="sesionUsuario._id != usuario._id">
138
139    <h2 style="border-bottom: 1px solid black; margin-bottom: 25px;">Datos personales</h2>
140
141    <form>
142
143      <!-- Nombre -->
144      <div class="form-group row">
145        <label for="staticNombre" class="col-sm-2 col-form-label">Nombre:</label>
146        <div class="col-sm-10">
147          <input type="text" readonly class="form-control-plaintext" id="staticNombre" value="{{usuario.nombre}}">
148        </div>
149      </div>
```

Figura 44.- Archivo .ts del componente perfil

Dependiendo de si el usuario que accede al perfil de un usuario es el del mismo usuario o no, los campos serán editables o no (como puede verse en las figuras 43 y 44).

8.2.2 Conexión al servidor

A continuación, se darán unos ejemplos del código de la parte de la conexión al servidor (services). Se detallarán los servicios para los objetos de tipo Evento, los de tipo Usuario y finalmente el servicio para subir imágenes.

Evento.service + Usuario.service

```typescript
import { Injectable } from '@angular/core';
import { HttpClient, HttpHeaders } from '@angular/common/http';
import { Observable } from 'rxjs';
import { Evento } from '../models/evento';
import { GlobalEventos } from './global';

@Injectable()
export class EventoService{
  public url: string;

  constructor(private _http: HttpClient){
    this.url = GlobalEventos.urlEventos;
  }

  guardarEvento(evento: Evento): Observable<any>{
    let params = JSON.stringify(evento);
    let headers = new HttpHeaders().set('Content-Type', 'application/json');
    return this._http.post(this.url+'guardar-evento', params, {headers: headers});
  }

  getEventos(): Observable<any>{
    let headers = new HttpHeaders().set('Content-Type', 'application/json');
    return this._http.get(this.url+'eventos', {headers: headers});
  }

  getEventosCreador(idCread):Observable<any>{
    let headers = new HttpHeaders().set('Content-Type', 'application/json');
    return this._http.get(this.url+'get-eventos-creador/'+idCread, {headers: headers});
  }
```

Figura 45.- Archivo evento.service

Como se puede observar en la figura 45 correspondiente a los servicios de los eventos, al principio se importan todo lo necesario. Una vez importado se utiliza Injectable() para poder definir un inyectable que pueda ser utilizado por los componentes para conectarse con el servidor. Luego, unos cuantos métodos de las funciones disponibles para interactuar con objetos de tipo Evento. Los de tipo Usuario (figura 46) siguen el mismo patrón.

```typescript
import { Injectable } from '@angular/core';
import { HttpClient, HttpHeaders } from '@angular/common/http';
import { Observable } from 'rxjs';
import { Usuario } from '../models/usuario';
import { GlobalUsuarios } from './global';

@Injectable()
export class UsuarioService{
  public url: string;

  constructor(private _http: HttpClient){
    this.url = GlobalUsuarios.urlUsuarios;
  }

  guardarUsuario(usuario: Usuario): Observable<any>{
    let params = JSON.stringify(usuario);
    let headers = new HttpHeaders().set('Content-Type', 'application/json');
    return this._http.post(this.url+'guardar-usuario', params, {headers: headers});
  }

  getUsuarios(): Observable<any>{
    let headers = new HttpHeaders().set('Content-Type', 'application/json');
    return this._http.get(this.url+'usuarios', {headers: headers});
  }

  getUsuario(id): Observable<any>{
    let headers = new HttpHeaders().set('Content-Type', 'application/json');
    return this._http.get(this.url+'usuario/'+id, {headers: headers});
  }
```

Figura 46.- Archivo usuario.service

Upload.service

```typescript
makeFileRequest(url: string, params: Array<string>, files: Array<File>, name: string){
  return new Promise(function(resolve, reject){
    var formData:any = new FormData();
    var xhr = new XMLHttpRequest();

    for(var i = 0; i<files.length; i++){
      formData.append(name, files[i], files[i].name);
    }

    xhr.onreadystatechange = function(){
      if(xhr.readyState == 4){
        if(xhr.status == 200){
          resolve(JSON.parse(xhr.response));
        }else{
          reject(xhr.response);
        }
      }
    }

    xhr.open('POST', url, true);
    xhr.send(formData);

  });
}
```

Figura 47.- Archivo upload.service

8.2.3 Servidor

En este apartado daremos algunas pinceladas sobre el código del servidor. Primero, daremos algún ejemplo de los controllers y luego daremos algún pequeño ejemplo de las rutas y de la configuración propia del servidor.

Evento controller

```
86  getEventosByTipo: function(req, res){
87    var categoriaEvento = req.params.categoriaEvento;
88
89    if(categoriaEvento == null) return res.status(404).send({message: 'El evento no es correcto.'});
90
91    Evento.find({categoria: categoriaEvento}, function(err, eventosCategoria){
92      if(err){
93        console.log("Algo va mal");
94      } else {
95        return res.status(200).send({eventosCategoria});
96      }
97    });
98  },
```

Figura 48.- Método getEventosByTipo del controller de los eventos

El ejemplo utilizado en la figura 48 es el método del controller de los eventos utilizado para obtener los eventos de x categoría (utilizado a la hora de recoger los eventos de tipo Charla, Evento y Reunión).

Usuario controller

```
39  valUser: function(req, res){
40    Usuario.find({
41      nickname: req.body.nickname, password: req.body.password
42    }, function(err, usuario){
43      if (err) res.status(500).send({message: 'Error al validar el usuario'});
44      if(usuario.length === 1){
45        return res.status(200).send({
46          status: 'success',
47          data: usuario
48        });
49      } else {
50        return res.status(200).send({
51          status: 'failed',
52          message: 'Login fallado'
53        });
54      }
55
56    });
57  },
```

Figura 49.- Método para validar a los usuarios del controller de los usuarios

El ejemplo utilizado en la figura 49 es el método del controller de los usuarios utilizado para validar a los usuarios. Dependiendo del resultado, se devuelve un estado (success) u otro (failed) al front-end.

Evento routes

```
var express = require('express');
var EventoController = require('../controllers/evento_controller');
var router = express.Router();
var multipart = require('connect-multiparty');
var multipartMiddleware = multipart({uploadDir: './uploads/eventos'});

//Rutas para GET
router.get('/home', EventoController.home);
router.get('/evento/:id?', EventoController.getEvento);
router.get('/eventos', EventoController.getEventos);
router.get('/get-image/:imagen', EventoController.getImageFile);
router.get('/get-eventos-creador/:id', EventoController.getEventosCreador);
router.get('/get-eventos-categoria/:categoriaEvento', EventoController.getEventosByTipo);

//Rutas para POST
router.post('/test', EventoController.test);
router.post('/guardar-evento/', EventoController.guardarEvento);
router.post('/subir-imagen/:id', multipartMiddleware, EventoController.uploadImagen);

//Rutas para PUT
router.put('/actualizar-evento/:id', EventoController.updateEvento);

//Rutas para DELETE
router.delete('/borrar-evento/:id', EventoController.deleteEvento);

//Exportación
module.exports = router;
```

Figura 50.- Archivo de configuración de rutas para los eventos

En las rutas en el servidor para las peticiones de la api de los eventos (figura 50) tenemos las típicas operaciones CRUD y alguna específica para alguna funcionalidad específica.

Usuario_routes

```
var express = require('express');
var UsuarioController = require('../controllers/usuario_controller');

var router = express.Router();
var multipart = require('connect-multiparty');
var multipartMiddleware = multipart({uploadDir: './uploads/usuarios'});

//Rutas para GET
router.get('/home', UsuarioController.home);
router.get('/usuario/:id?', UsuarioController.getUsuario);
router.get('/usuarios', UsuarioController.getUsuarios);
router.get('/get-image/:imagen', UsuarioController.getImageFile);

//Rutas para POST
router.post('/loguear', UsuarioController.valUser);
router.post('/guardar-usuario', UsuarioController.guardarUsuario);
router.post('/subir-imagen/:id', multipartMiddleware, UsuarioController.uploadImagen);

//Rutas para PUT
router.put('/actualizar-usuario/:id', UsuarioController.updateUsuario);

//Rutas para DELETE
router.delete('/borrar-usuario/:id', UsuarioController.deleteUsuario);

//Exportación
module.exports = router;
```

Figura 51.- Archivo de configuración de rutas para los usuarios

Igual que en las rutas para el api de los eventos, en las rutas para el api de los usuarios están las mismas operaciones típicas CRUD y alguna específica también (validar).

Index.js (configuración servidor)

```
'use strict'

var mongoose = require('mongoose');
var app = require('./app');
var port = 3500;

mongoose.Promise = global.Promise;
mongoose.connect('mongodb://localhost:27017/my-events')
        .then(()=>{
          console.log("Conexión a MongoDB establecida satisfactoriamente.");

          //Creación del servidor
          app.listen(port, () => {
            console.log("Servidor corriendo en localhost:" + port);
          });

        })
        .catch(err => console.log("Error en la conexión a MongoDB: " + err));

```

Figura 52.- Archivo index..js

En este ejemplo (figura 52) se muestra el código que permite crear y lanzar el servidor escuchando en el puerto especificado.

App.js (configuración servidor 2)

```
'use strict'

var express = require('express');
var bodyParser = require('body-parser');

var app = express();

//Cargar archivos de rutas
var evento_routes = require('./routes/evento_routes');
var usuario_routes = require('./routes/usuario_routes');

//Middlewares -> Capa que se ejecuta antes de la acción de un controlador
app.use(bodyParser.urlencoded({extended:false}));
app.use(bodyParser.json());

//CORS
app.use((req, res, next) => {
    res.header('Access-Control-Allow-Origin', '*');
    res.header('Access-Control-Allow-Headers', 'Authorization, X-API-KEY, Origin, X-Requested-With, Content-Type, Accept, Access-Contr
    res.header('Access-Control-Allow-Methods', 'GET, POST, OPTIONS, PUT, DELETE');
    res.header('Allow', 'GET, POST, OPTIONS, PUT, DELETE');
    next();
});

//Definir rutas
app.use('/api-eventos', evento_routes);
app.use('/api-usuarios', usuario_routes);

//Exportación
module.exports = app;

```

Figura 53.- Archivo app.js

En el ejemplo de la figura 53 se muestra el código que configura el CORS (Cross-origin Resourse Sharing) y los endpoints de las dos api's.

8.2.4 Base de datos (BBDD)

El código que se relaciona con la BBDD es el correspondiente a los modelos del servidor y a los controllers. Ambos ya han sido vistos en las secciones 7.1.3 (controllers) y 6.2.1/6.2.2 (modelos).

CAPÍTULO 9

Conclusiones

En este apartado se van a dar unas pequeñas conclusiones sobre el trabajo realizado para acabar con una opinión personal del autor.

A lo largo de este trabajo se ha visto el tamaño que puede tomar una aplicación 'simple': cientos de archivos, carpetas, líneas de código, etc, es algo increíble que no eres consciente hasta que no te pones manos a la obra.

En un inicio la aplicación no iba a tener un diseño visual tan 'bonito', si no que iba a ser mucho más rudimentaria. En cambio, a medida que se iban estudiando alternativas y surgió Bootstrap, todo cambió. Funcionaba a las mil maravillas y se compenetraba a la perfección con Angular.

Aunque a veces Typescript me ha dado algún que otro problema por su fuerte tipado, ha valido la pena pues la potencia que tiene este 'lenguaje' es asombroso.

Por lo anteriormente expuesto podemos decir que la aplicación realizada ha conseguido cumplir con todos los requisitos expuestos a lo largo de esta memoria, y podría ser utilizada por multitud de personas con total éxito.

9.1 Mejoras

A continuación, vamos a dar algunas posibles mejoras que realizar a la aplicación de cara al futuro:

- Implementar la posibilidad de suscribirse a los eventos.
- Mapa interactivo en los datos del evento para ver el lugar exacto.
- Hacer la aplicación web responsive para cualquier tamaño de pantalla.
- Posibilidad de filtrar eventos por nombre, fecha, etc.

9.2 Opinión personal

Nunca en mi vida había experimentado tanta presión como en los últimos dos meses. Si ya de por si el trabajo tiene muchísima, cabe añadirle la casi nula experiencia a la hora de desarrollar proyectos full-stack (misma persona desarrollando front y back-end), debiendo de formarme para tal fin.

Claro está que ha sido una de las mejores experiencias de mi vida. Me ha ayudado a mejorar tanto a nivel organizativo a la hora de realizar el trabajo como a nivel emocional, pues he tenido que gestionar muchísimas emociones de presión, tristeza al ver que no funcionaba, etc.

Es por esto y por todo lo ocurrido en estos 4 años de carrera que no me arrepiento de absolutamente nada de lo que he hecho, siendo algo que considero indispensable que sea vivido por cualquier persona. Una experiencia que te cambia la vida para siempre.

Bibliografía

Materiales consultados

1. *https://es.wikipedia.org/wiki/Meetup*
2. *Bruegge, Bernd y Dutoit, Allen. Bruegge, 2002. Ingeniería de Software Orientado a Objetos.*
3. *Software Engineering Standards Committee of the IEEE. Computer Society. IEEE Recommended Practice for Software Requirements*
4. *Pressman, Roger. McGraw-Hill, 2002. Ingeniería del Software. Un enfoque práctico. 5ta. Edición.*
5. *Sommerville, Ian. Prentice-Hall, 2002. Ingeniería de Software. 6ta. Edición.*
6. *https://es.wikipedia.org/wiki/Metodolog%C3%ADa_de_desarrollo_de_software*
7. *http://isw-udistrital.blogspot.com/2012/09/ingenieria-de-software-i.html*
8. *http://marich.blogspot.es/1459223366/modelo-incremental/*
9. *https://es.wikipedia.org/wiki/HTML*
10. *https://www.genbeta.com/desarrollo/una-introduccion-a-mongodb*
11. *https://www.genbeta.com/desarrollo/mongodb-que-es-como-funciona-y-cuando-podemos-usarlo-o-no*
12. *https://es.wikipedia.org/wiki/MongoDB*

APÉNDICE

Manual de usuario de la aplicación

En este manual de usuario de la aplicación vamos a guiar paso a paso al usuario para la realización de cualquiera de las funcionalidades que posee la aplicación.

Login

1) Acceder a la página principal.
2) Hacer click en 'Iniciar sesión' en la barra de navegación arriba a la derecha.

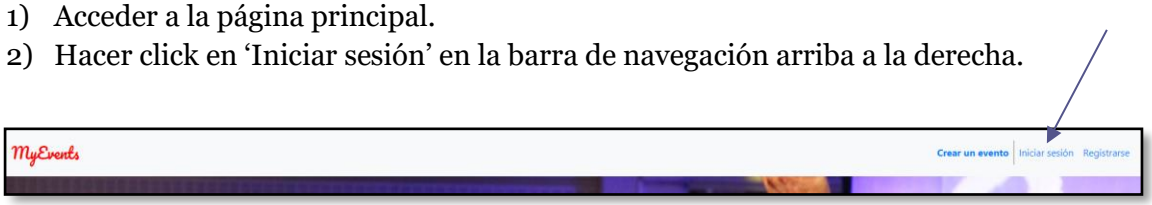


Figura 54.- Barra de navegación sin autenticar

3) Rellenar el formulario (figura 55).

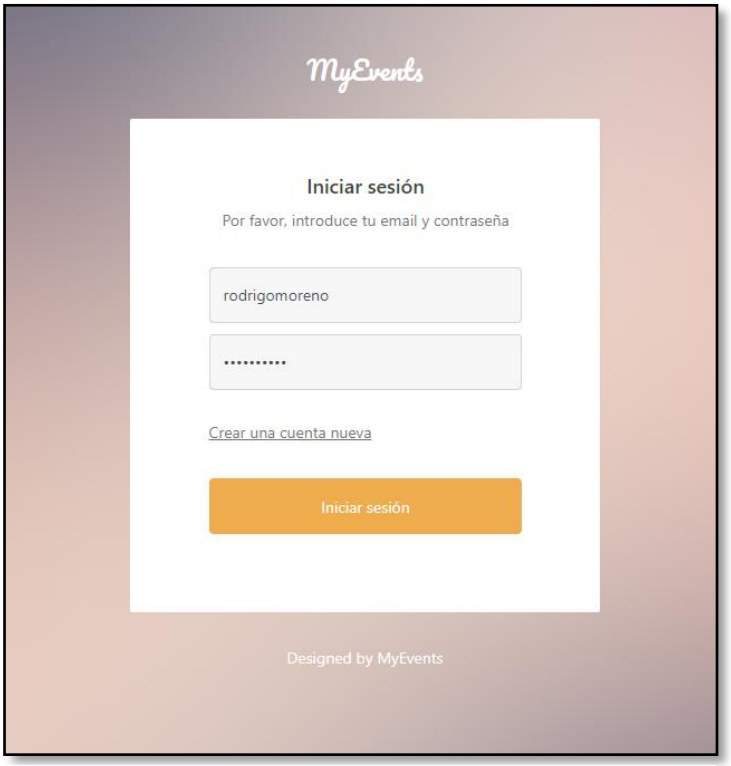


Figura 55.- Pantalla de login

4) Hacer click en ‘Iniciar sesión’.

Registrarse

1) Acceder a la pantalla de registro bien desde el home o bien desde la pantalla de login (figuras 56 y 57).

Figura 56.- Barra de navegación sin autenticar

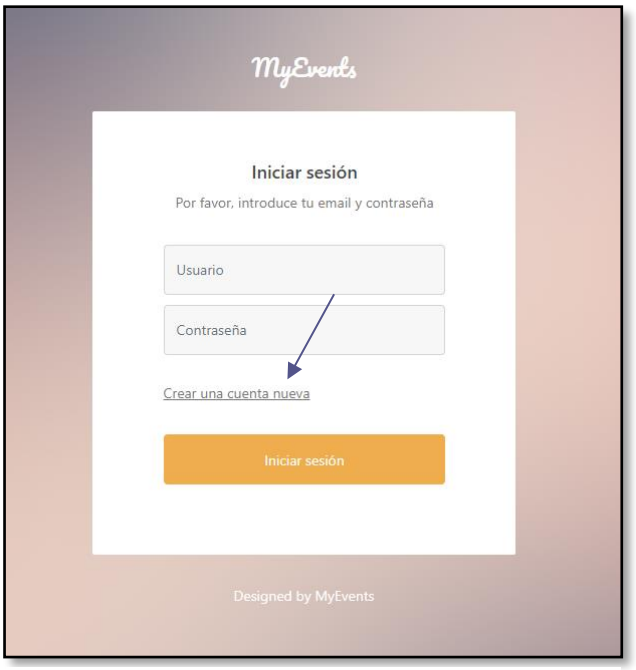


Figura 57.- Pantalla de login

2) Rellenar el formulario (figura 58).

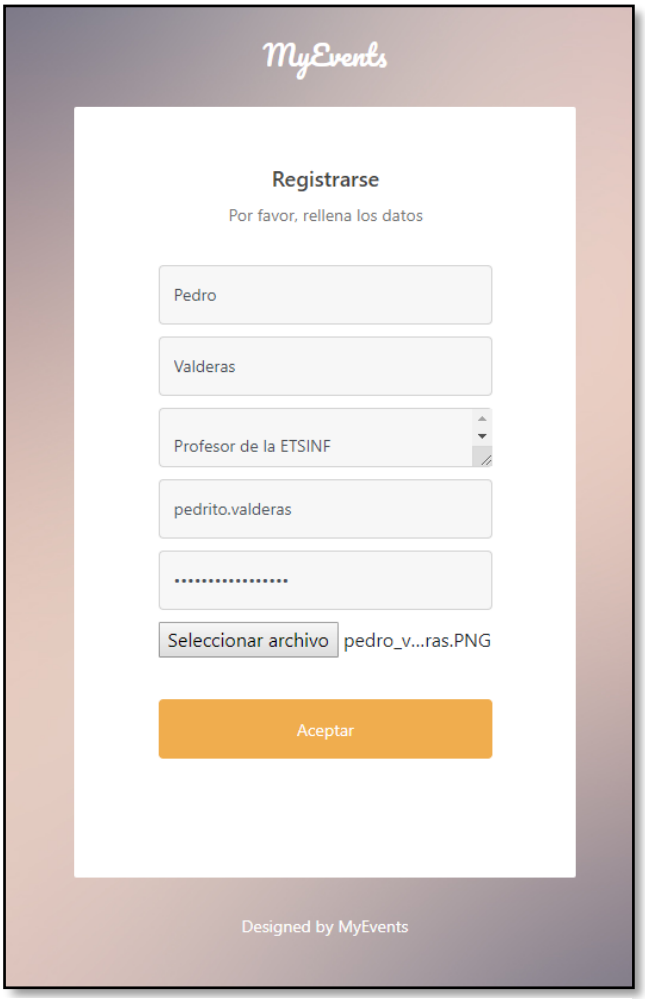


Figura 58.- Formulario de registro

3) Hacer click en 'Aceptar'.

Crear un evento

1) Loguearse.
2) Hacer click en 'Crear un evento' en la barra de navegación (figura 59).

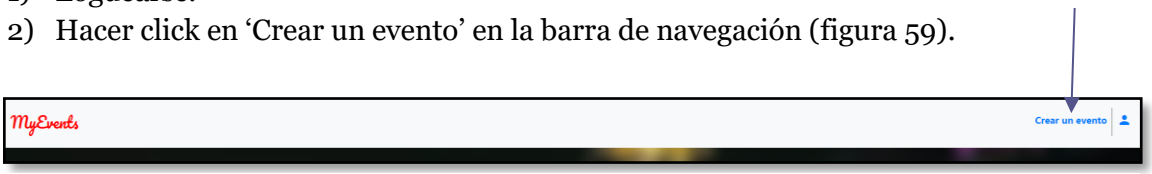


Figura 59.- Barra de navegación con autenticación

3) Rellenar el formulario (figura 60).

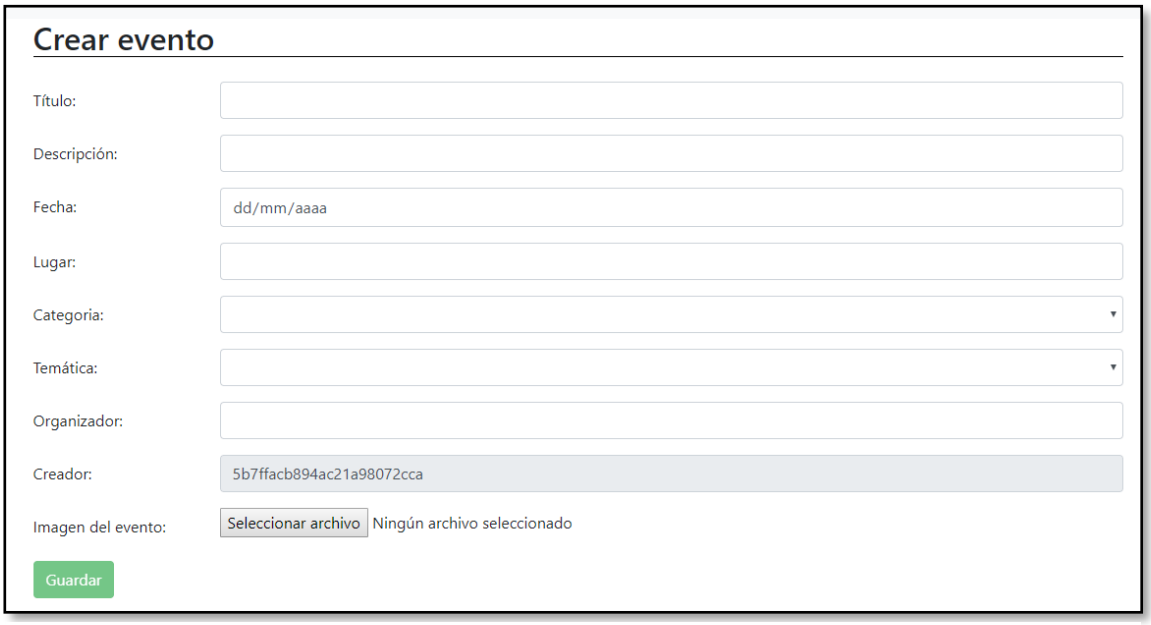


Figura 60.- Formulario crear un evento

4) Hacer click en 'Guardar'.

Editar los datos de un evento

1) Loguearse
2) Hacer click en el botón de 'Entrar' (figura 61) del evento al que queramos modificar sus datos para entrar a la pantalla de sus datos

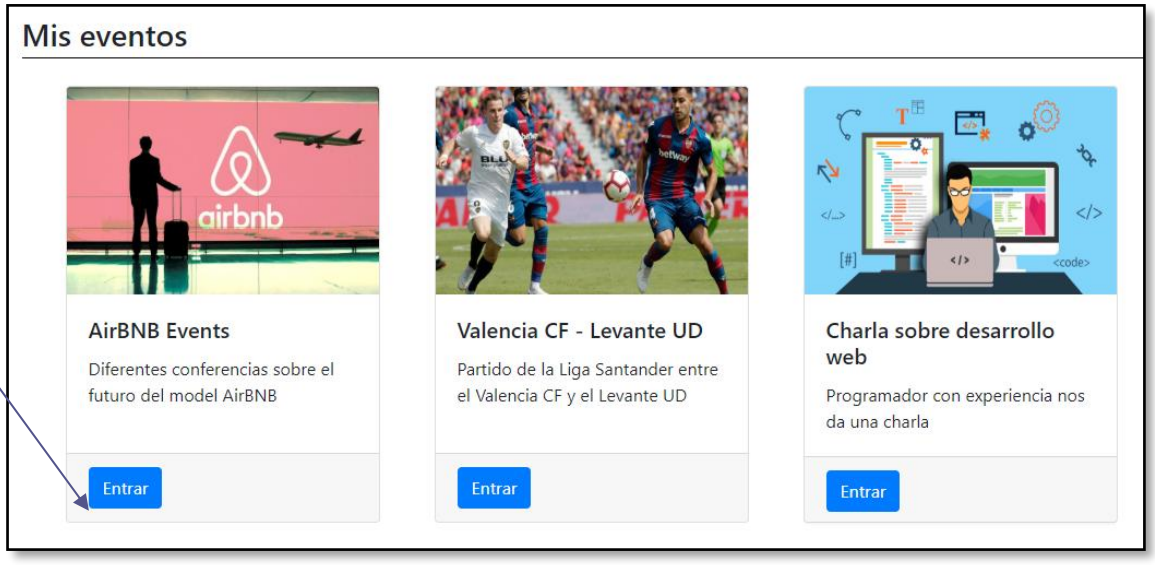


Figura 61.- Sección mis eventos con evento a modificar

3) Modificar el campo que queramos modificar y hacer click en 'Guardar' (figura 62).

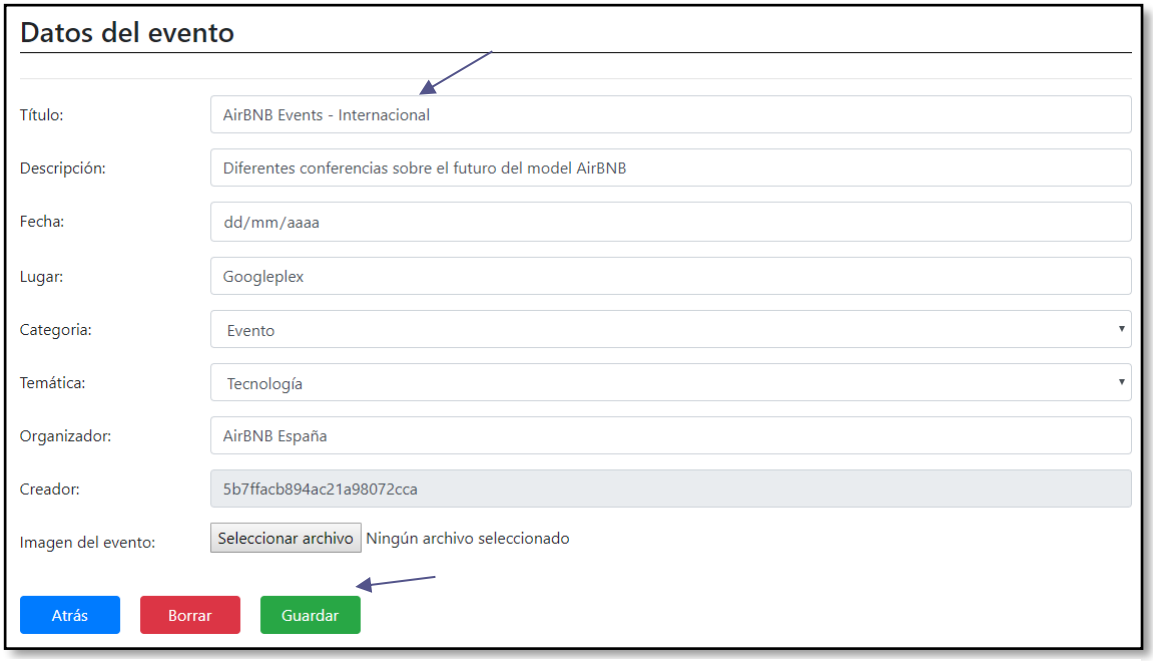


Figura 62.- Pantalla modificar datos de evento

4) Comprobar que los datos se han actualizado correctamente (figura 63).

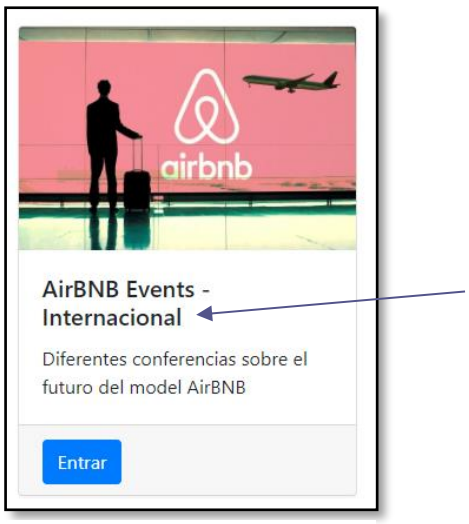


Figura 63.- Tarjeta del evento modificado

Borrar un evento

1) Loguearse
2) Hacer click en el botón 'Entrar' de la carta correspondiente al evento que queramos borrar (figura 64).

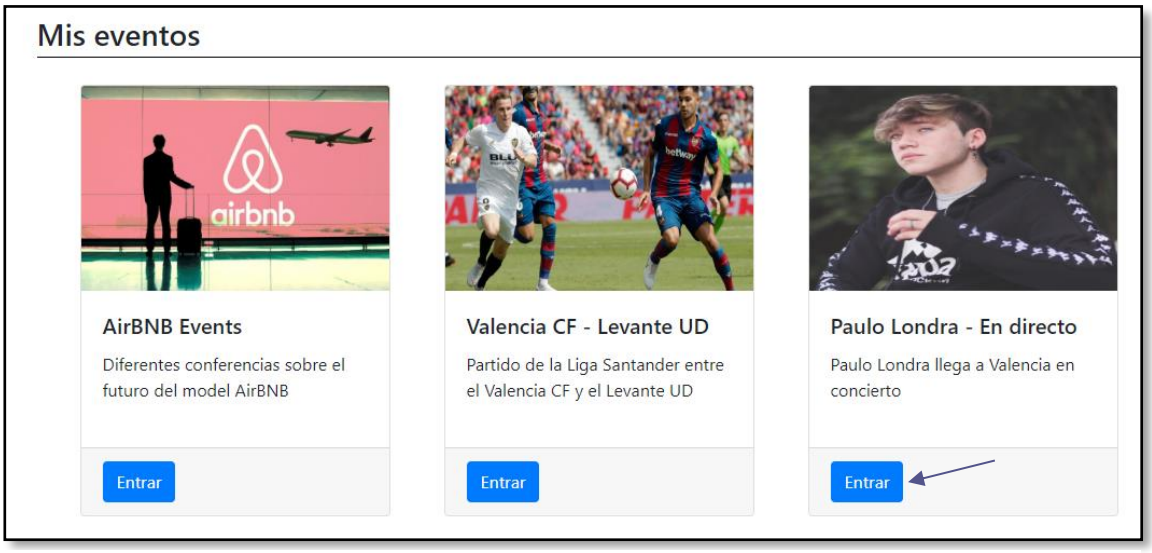


Figura 64.- Sección mis eventos con evento a borrar

3) Hacer click en el botón 'Borrar' (figura 65)

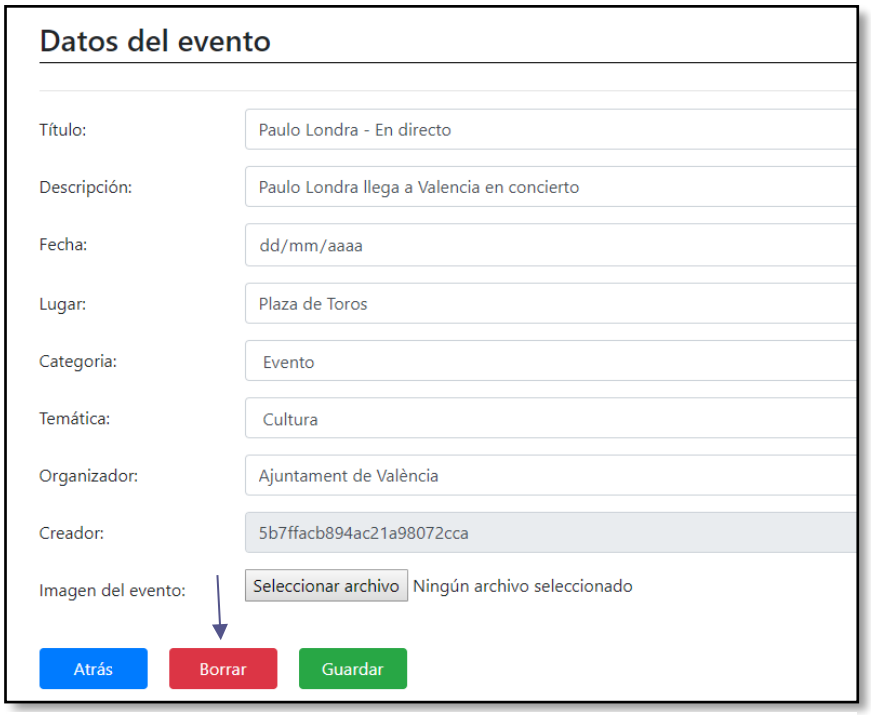


Figura 65.- Sección edición datos del evento con botón de borrar

4) Comprobar que el evento se ha borrado (figura 66).

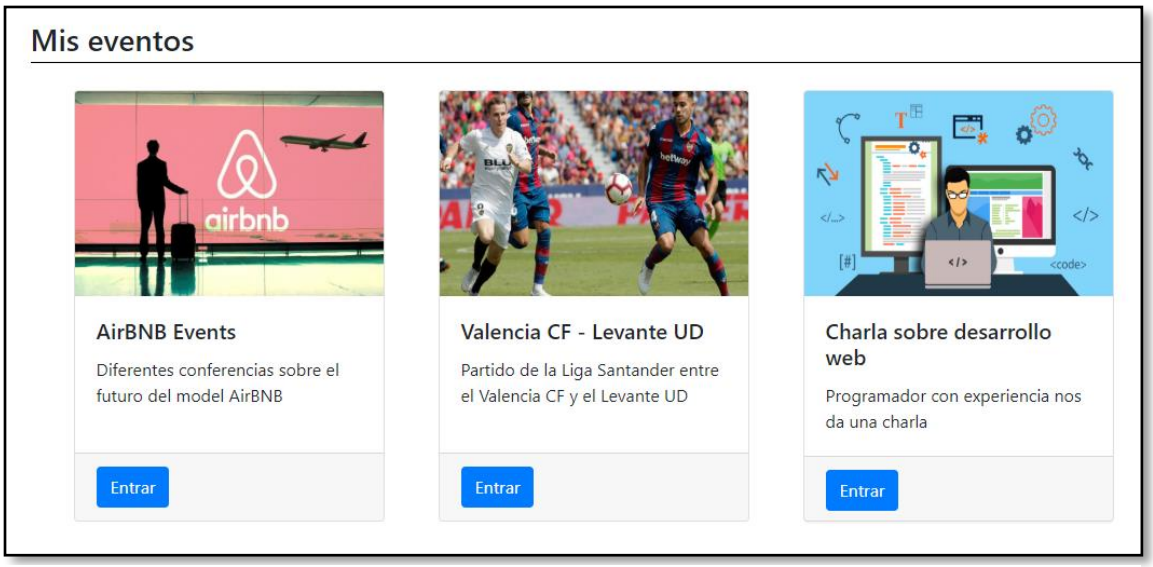


Figura 66.- Sección mis eventos con evento borrado

Filtrar eventos por categorías

1) Loguearse (opcional).
2) Hacer click en la imagen correspondiente al tipo de evento que nos interesa en la parte inferior del home (figura 67).

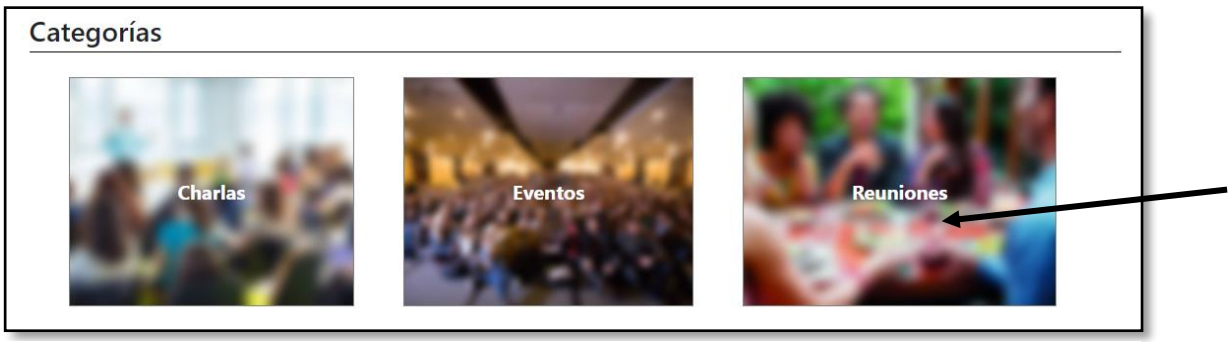


Figura 67.- Sección categorías

3) Comprobar que estamos en la categoría seleccionada (figura 68).

Figura 68.- Sección eventos eventos tipo reunión