



GRADO EN INGENIERÍA EN TECNOLOGÍAS DE LA  
TELECOMUNICACIÓN

Curso Académico 2018/2019

Trabajo Fin de Grado/Máster

APLICACIÓN WEB PARA LA CREACIÓN DE  
EVENTOS DEPORTIVOS EN LOCALIZACIÓN  
DETERMINADA

Autor : Jesús José Moreno Pinto

Tutor : Pedro de las Heras Quirós



# Trabajo Fin de Grado/Máster

Título del Trabajo con Letras Capitales para Sustantivos y Adjetivos

**Autor :** Jesús José Moreno Pinto

**Tutor :** Pedro de las Heras Quirós

La defensa del presente Proyecto Fin de Carrera se realizó el día            de  
de 20XX, siendo calificada por el siguiente tribunal:

**Presidente:**

**Secretario:**

**Vocal:**

y habiendo obtenido la siguiente calificación:

**Calificación:**

Fuenlabrada, a            de            de 20XX

*Dedicado a  
mi familia / mi abuelo / mi abuela*

# Agradecimientos

Aquí vienen los agradecimientos... Aunque está bien acordarse de la pareja, no hay que olvidarse de dar las gracias a tu madre, que aunque a veces no lo parezca disfrutará tanto de tus logros como tú... Además, la pareja quizás no sea para siempre, pero tu madre sí.

# Índice general

<b>1. Resumen</b>	<b>1</b>
<b>2. Introducción</b>	<b>2</b>
2.1. Desarrollo web . . . . .	2
2.2. Tecnologías web . . . . .	3
2.2.1. Lenguaje . . . . .	3
2.2.2. Framework . . . . .	5
2.2.3. Librería . . . . .	6
2.2.4. API . . . . .	7
2.2.5. Programación reactiva . . . . .	8
2.3. Base de datos . . . . .	9
2.3.1. MongoDB . . . . .	9
2.3.2. Mongoose . . . . .	10
2.4. Herramientas de trabajo en equipo . . . . .	10
2.4.1. Git y Github . . . . .	11
2.4.2. Jira . . . . .	11
2.4.3. Herramientas para la comunicación entre desarrolladores . . . . .	12
<b>3. Objetivos y metodología</b>	<b>13</b>
3.1. Motivación . . . . .	13
3.2. Objetivos . . . . .	14
3.3. Requisitos . . . . .	14
3.4. Metodología y plan de trabajo . . . . .	16
3.4.1. Metodología . . . . .	16

3.4.2. Plan de trabajo . . . . .	16
<b>4. Diseño e implementación</b>	<b>19</b>
4.1. Arquitectura utilizada . . . . .	19
4.2. Búsqueda de herramientas . . . . .	20
4.3. Entorno de desarrollo . . . . .	22
4.3.1. Creación de la aplicación . . . . .	22
4.4. Diseño de las páginas . . . . .	24
4.4.1. Diseño principal y Registro . . . . .	24
4.4.2. Registro de usuarios . . . . .	25
4.4.3. Login . . . . .	26
4.4.4. Home . . . . .	28
<b>5. Resultados</b>	<b>31</b>
<b>6. Conclusiones</b>	<b>33</b>
6.1. Consecución de objetivos . . . . .	33
6.2. Aplicación de lo aprendido . . . . .	34
6.3. Lecciones aprendidas . . . . .	35
6.4. Trabajos futuros . . . . .	36
<b>A. Manual de usuario</b>	<b>37</b>
<b>Bibliografía</b>	<b>39</b>

# Capítulo 1

## Resumen

Mediante este proyecto se ha implementado una aplicación web cuyo principal objetivo es la creación de puntos de encuentro con carácter deportivo, de manera que se pueda jugar un partido de tu deporte favorito en caso de no disponer de los conocidos necesarios para un día concreto. De esta manera se fomenta el establecimiento de nuevas amistades con el deporte como punto en común.

La realización de esta aplicación ha buscado que la creación y unión de los usuarios a la partida fuera lo más rápido e intuitivo posible. Para dicha aplicación se han utilizado las siguientes tecnologías de desarrollo como Angular5, Angular Material, Visual Studio Code, HTML5, CSS3, nodeJS, expressJS, javascript, typescript, MongoDB , Maps JavaScript API, Geocoding API.



# Capítulo 2

## Introducción

El desarrollo del Trabajo Fin de Grado se centra en el ámbito deportivo y la utilización de herramientas web para facilitar su realización. A continuación se expondrá una pequeña introducción de conceptos básicos sobre el desarrollo web y las tecnologías utilizadas para la realización de dicha aplicación.

### 2.1. Desarrollo web

La vida actual gira entorno a la evolución tecnológica en la que nos encontramos sumergidos, en un periodo muy corto de tiempo hemos sufrido una gran evolución.

Las tecnologías web se encuentran fuertemente arraigadas en nuestro día a día y con motivo de ello deben renovarse continuamente para facilitar el desarrollo de aplicaciones en el menor tiempo posible, facilitar su mantenimiento y mejorar su rendimiento.

Debemos retroceder a 1991, año en el que se publicó la primera página web por Tim Berners-Lee.

La web 1.0 se denomina así para definir todo lo creado anteriormente al fenómeno de la Web 2.0 y se trataban de páginas estáticas en HTML donde el papel del usuario era totalmente pasivo.

La web 2.0 era de carácter social, al contrario que su predecesor, se basaba en usuarios activos y se produjo un gran auge de los blogs, redes sociales, webs creadas por usuarios, etc.

Por último nos encontramos ante la Web 3.0 dada por el avance tecnológico hacia inteligencia artificial y de la web semántica, donde es primordial el diseño responsive para la adaptación a cualquier dispositivo.

En el diseño web debemos tener en cuenta las herramientas a utilizar, diseño y la metodología de desarrollo utilizada, con el continuo desarrollo de las tecnologías se ha reducido la dificultad de implementación de las dichas web.

## 2.2. Tecnologías web

Son todas aquellas herramientas utilizadas en la creación de sitios web o aplicaciones online, podemos diferenciar entre librería, framework, API y lenguaje.

### 2.2.1. Lenguaje

Según wikipedia: *“un lenguaje de programación<sup>1</sup> es un lenguaje formal que proporciona una serie de instrucciones que permiten a un programador escribir secuencias de órdenes y algoritmos a modo de controlar el comportamiento físico y lógico de una computadora con el objetivo de que produzca diversas clases de datos. A todo este conjunto de órdenes y datos escritos mediante un lenguaje de programación se le conoce como programa.”*

Para el caso del desarrollo web podemos diferenciar entre lenguajes para el Front-End y Back-End, en el lado del cliente debemos tener en cuenta que el navegador compila e interpreta sólo archivos HTML, CSS y como lenguaje de lógica Javascript.

Existen lenguajes como Typescript, Coffesscript.. que deben ser compilados a Javascript para su uso por parte del navegador.

#### Typescript

Se trata de un lenguaje de programación de alto nivel orientado a objetos, como ya se ha comentado anteriormente se “transpila” en Javascript nativo, es un superset.

---

<sup>1</sup>[https://es.wikipedia.org/wiki/Lenguaje\\_de\\_programacion](https://es.wikipedia.org/wiki/Lenguaje_de_programacion)

Al contrario que Javascript, Typescript está realmente orientado a objetos, posee herramientas como la herencia o sobrecarga (resulta similar a Java). En el caso de esta aplicación su uso ha sido necesario puesto que Angular está hecho en Typescript.

## Javascript

Como hemos comentado anteriormente es el lenguaje que interpretará el navegador, entrará en juego cada vez que la página tenga que hacer algo más que representar información estática. Es un lenguaje de programación orientado a objetos de tipado muy débil y dinámico.

El ámbito de utilización del lenguaje no se reduce sólo al lado del cliente, también es utilizado en frameworks de la parte del servidor basados en NodeJS.

## HTML5

Es la quinta revisión del lenguaje principal de la World Wide Web, el HTML. Podemos diferenciar entre 2 variantes de sintaxis para HTML, una básica que se sirve de HTML conocida como **HTML5** y otra variante conocida como **XHTML5** que se sirve como sintaxis XML. Este lenguaje de marcado se encuentra regulado por el Consorcio **W3C**<sup>2</sup>.

Algunas de las mejoras introducidas con el HTML5 son:

- Ampliación del elemento **input** permitiendo tipos de datos como range, email, url, search...
- Introducción de elementos de audio y video para incrustar contenido multimedia.
- Aparece el elemento **canvas** que permite la generación de gráficos y ‘dibujar’ lo que se quiera dentro de él. Un ejemplo actual de su uso sería Google Maps.

Estas serían algunas de las mejoras introducidas por las cuales se reduce el uso de plugins por parte del desarrollador.

---

<sup>2</sup>[https://es.wikipedia.org/wiki/World\\_Wide\\_Web\\_Consortium](https://es.wikipedia.org/wiki/World_Wide_Web_Consortium)

### CSS3

el lenguaje encargado de darle el estilo al documento HTML o XML es el CSS. CSS3 es la tercera versión y algunas de las nuevas funcionalidades introducidas serían las esquinas redondeadas de los elementos, gradientes, transiciones o animaciones y nuevos diseños como multi-columnas, cajas flexibles o *grids* (diseño de cuadrícula).

#### 2.2.2. Framework

Por definición un **framework**<sup>3</sup>. es un conjunto estandarizado de conceptos, prácticas y criterios para enfocar un tipo de problemática particular que sirve como referencia, para enfrentar y resolver nuevos problemas de índole similar.

Hoy en día existen multitud de frameworks para el desarrollo de aplicaciones web y es imprescindible conocer el entorno en el que se ejecuta cada uno de ellos, podemos diferenciar entre los orientados para cliente (Front-End), orientado a servidor (Back-End) o a ambos (Full-Stack). Actualmente se usan una gran cantidad de frameworks, para el desarrollo de esta aplicación se han utilizado: **Angular**<sup>4</sup>, concretamente la v5, para la realización del Front-End y **ExpressJS**<sup>5</sup> y **NodeJS**<sup>6</sup> para el Back-End<sup>7</sup>.

#### Angular

Se trata de un framework Front-End desarrollado en typescript, el objetivo es que el peso de la lógica y renderizado sea tarea del navegador y así liberar al servidor de trabajo. Mediante peticiones REST obtendremos la información necesaria para el correcto funcionamiento de nuestra aplicación.

Fue creado por Google con la intención de mejorar su framework anterior (**AngularJS**), pese a ser la evolución de éste son incompatibles.

---

<sup>3</sup><https://es.wikipedia.org/wiki/Framework>

<sup>4</sup><https://angular.io>

<sup>5</sup><https://expressjs.com/es>

<sup>6</sup><https://nodejs.org/es/>

<sup>7</sup>[https://es.wikipedia.org/wiki/Front-end\\_y\\_back-end](https://es.wikipedia.org/wiki/Front-end_y_back-end)

Existe una gran cantidad de información en internet para su aprendizaje y al igual que su antecesor, mantiene una gran rivalidad con **React**<sup>8</sup>, que fue creado por Facebook.

El lenguaje utilizado para el desarrollo de una aplicación Angular es typescript,

## **NodeJS**

Se trata de un entorno en tiempo de ejecución multiplataforma, de código abierto, para la capa del servidor (pero no limitándose a ello) basado en el lenguaje de programación ECMAScript, asíncrono, con I/O de datos en una arquitectura orientada a eventos y basado en el motor V8 de Google.

## **ExpressJS**

Se trata de un framework Back-End de aplicaciones web Node.js mínima y flexible que proporciona un conjunto sólido de características para las aplicaciones web y móviles. Dispone de una robusta API que permite a los usuarios configurar las rutas para la interacción entre el Front-End y la base de datos.

### **2.2.3. Librería**

Una librería es un archivo o conjunto de archivos , en un lenguaje dado, utilizado para facilitar la programación del desarrollador.

En el caso de esta aplicación podríamos destacar librerías utilizadas durante el desarrollo como:

## **Rxjs**

Es una librería básica en la programación reactiva en la que se trabaja con flujos de datos de forma asíncrona, en el caso de Angular se trabaja con los `.subscribe()` y operadores que actúan sobre la respuesta, una vez es obtenida.

Ofrecen una solución óptima para tareas asíncronas complejas.

---

<sup>8</sup><https://reactjs.org/>

### **Eslint**

Una librería de gran ayuda que analiza el código desarrollado y encuentra problemas como podrían ser líneas de código que nunca se ejecutarán y variables que nunca se han utilizado entre otras muchas. Se utiliza para mantener una serie de reglas en la codificación que facilitarán su desarrollo y mantenimiento, sobre todo en el caso de que se desarrolle en equipo ya que se evita que los hábitos de cada uno dificulten el seguimiento del código (se mantiene cierta estructura común para todos).

### **Webpack**

Se trata en un empaquetador de módulos para modernas aplicaciones Javascript. En el momento que webpack procesa tu aplicación, contruye un grafo de dependencias el cual mapea cada módulo que necesita la aplicación y genera 1 o más paquetes.

### **Angular Material**

Esta librería está centrada en el diseño de componentes de Angular creada por Google, en ella podrás encontrar una gran cantidad de elementos para facilitar la creación y diseño de componentes y aplicaciones.

Además se tratan de componentes que ya son responsive, esto quiere decir que se adaptarán a distintos terminales (móviles, tablets, portátiles y ordenadores de sobremesa).

## **2.2.4. API**

La palabra API viene dada por **Application Programming Interface** (Interfaz de Programación de Aplicaciones), este concepto hace referencia a todos aquellos procesos, funciones y métodos que da una biblioteca de programación como capa de abstracción para que lo utilice otro programa.

Gracias a las APIs podemos recurrir a la funcionalidad que dé una en concreto y así ahorrar el trabajo de desarrollarla de cero. en el caso de esta aplicación se han utilizado 2 APIs propiedad de Google:

### Geocoding API

<sup>9</sup> Es una API de Google capaz de convertir las direcciones en coordenadas geográficas, en el caso de caso de esta aplicación se ha utilizado esta funcionalidad y la inversa **Reverse geocoding**, que consiste en la obtención de direcciones a través de las coordenadas..

### Maps Javascript API

<sup>10</sup> Se trata de una API de Google que permite la creación de mapas personalizados con tu propio contenido y mostrarlos en páginas web. Para el caso de uso, se han utilizado marcadores de posición personalizados (pelotas de deporte) y mediante el evento de doble click sobre el mapa se han obtenido las coordenadas de la posición del mapa para su posterior almacenamiento.

Permite capturar multitud de eventos para realizar el tratamiento de ellos en función de la necesidad de la aplicación.

Para el uso de las APIs de Google es necesario la creación de una key, en cada petición http deberá incluirse la key ya que sino no se tendrá acceso a la funcionalidad de las APIs utilizadas. Hace poco entró en vigor una nueva norma por la cual deberá dejarse registrado un método de pago, en caso de exceder el número de peticiones máximas por mes en cada API se procederá al cobro automático por cada uso fuera del servicio mínimo.

### 2.2.5. Programación reactiva

Anteriormente se ha hecho uso de un término muy importante en la programación web de hoy en día, se trata de la **reactividad**.

La programación reactiva es la programación con flujos de datos asíncronos, el típico evento de click en una página web es un flujo de datos que puedes observar y por el cual desencadenar una serie de eventos sobre el contenido de la página. Se pueden realizar multitud de acciones sobre los datos cómo filtrarlos, combinarlos con otros, crear nuevos a partir de ellos...

---

<sup>9</sup><https://developers.google.com/maps/documentation/geocoding/intro>

<sup>10</sup><https://developers.google.com/maps/documentation/javascript/tutorial>

Un flujo o stream es una secuencia de eventos ordenados en el tiempo de los que podemos diferenciar 3 tipos:

- Valor.
- Error.
- Señal de completado.

Estos eventos emitidos de forma asíncrona pueden ser capturados mediante la definición de funciones que serán ejecutadas cuando se emita un valor, funciones de error cuando se emita el error por un fallo y otra función cuando el 'completado' es emitido. La forma de enterarse de estas emisiones es mediante el concepto de subscripción, donde el 'observador' sería la función y el 'observado' sería el stream.

## 2.3. Base de datos

La base de datos utilizada por la aplicación es MongoDB y la librería mediante la cual se generan los documentos y realizan las operaciones **CRUD**<sup>11</sup> es **Mongoose**<sup>12</sup>.

### 2.3.1. MongoDB

MongoDB es una base de datos NoSQL orientada a documentos de código abierto y gratuita; al contrario que las bases de datos relacionales, donde los datos se almacenan en filas de una tabla, estos datos quedan almacenados en documentos BSON.

Los documentos no tienen una esquema fijo y su estructura básica está basada en 2 partes: clave, valor. Los tipos de datos que soporta son:

- |           |                         |
|-----------|-------------------------|
| ■ Null    | ■ String                |
| ■ Boolean | ■ Date                  |
| ■ Number  | ■ Expresiones regulares |

---

<sup>11</sup><https://es.wikipedia.org/wiki/CRUD>

<sup>12</sup><https://mongoosejs.com/docs/guide.html>



- Array
- ObjectId
- Documento embebido
- Código javascript

Cada documento creado dispone de un `_id` único que por defecto es de tipo `ObjectId`, en caso de no crearlo el desarrollador se crea uno por defecto. Otro concepto importante es el de **colección**, es un conjunto de documentos que pueden tener esquemas distintos.

### 2.3.2. Mongoose

Mongoose es un Object Document Mapper<sup>13</sup> (ODM), mediante el cual permite definir objetos con un esquema fuertemente tipado que es asignado a un documento MongoDB.

Dispone de múltiples herramientas mediante las cuales puedes definir funciones de validación de datos personalizadas, indicar que un campo es requerido ó convertir strings a minúsculas o mayúsculas entre otras muchas.

## 2.4. Herramientas de trabajo en equipo

En el desarrollo de páginas y aplicaciones web es fundamental una buena organización, aquí entran en juego las metodologías ágiles y herramientas para el trabajo en equipo. En busca de una reducción en el tiempo de desarrollo con el mejor resultado posible aparecieron las metodologías ágiles cuyas bases son:

- La mayor prioridad es la satisfacción del cliente.
- Entrega periódica de software.
- Transmisión de la información preferiblemente cara a cara.
- Capacidad de autorganización por parte del equipo.
- Aceptar el cambio de requisitos durante su desarrollo.

---

<sup>13</sup><https://www.quora.com/What-is-Object-Document-Mapping>

Existen múltiples metodologías ágiles entre las que se pueden destacar **SCRUM**<sup>14</sup> y **Extreme Programming (XP)**.

La metodología ágil consiste en una serie de buenas prácticas para maximizar el rendimiento del equipo de desarrollo y como ayuda para ello se utilizan herramientas como **Github**, **Jira**, **Slack** y **Skype** entre otras.

### 2.4.1. Git y Github

Git es un software de control de versiones pensado para lograr la mayor eficiencia y confiabilidad del mantenimiento de versiones de aplicaciones. Crea una copia del proyecto en desarrollo en un repositorio y mediante commits se almacenan diversas versiones del mismo que se pueden recuperar posteriormente. Permite la creación de ramas (branches) de versiones que en el momento deseado pueden unirse al proyecto principal (master).

A la hora de desarrollar en equipo esto es algo fundamental pues los miembros del equipo podrían realizar desarrollos en paralelo para posteriormente unirlos en el proyecto principal.

Todo esto se realizaría mediante Github, es una plataforma de desarrollo colaborativo de software donde se almacenan proyectos públicos o privados (mediante pago) que utiliza el sistema de control de versiones Git.

### 2.4.2. Jira

Dentro de las fases durante el desarrollo del proyecto, la primera de todas sería el análisis y establecimiento de requisitos.

Jira es una plataforma para la gestión y administración de proyectos que permite planear y realizar un seguimiento del proyecto, es ideal para equipos de trabajo que implementen estrategias de desarrollo Agile como SCRUM.

Dentro de Jira existe un parámetro llamado **Incidencia-Issue** que puede representar tanto una tarea, un bug, un ticket de soporte, un feedback... que dispone de 3 atributos:

---

<sup>14</sup>[https://es.wikipedia.org/wiki/Scrum\\_\(desarrollo\\_de\\_software\)](https://es.wikipedia.org/wiki/Scrum_(desarrollo_de_software))

- Prioridad: indica la importancia.
- Estado: 'donde' está la incidencia respecto al flujo de trabajo.
- Resolución: se cambia el estado respecto al tipo de resolución aplicada.

### **2.4.3. Herramientas para la comunicación entre desarrolladores**

Debido a la posibilidad de la división espacial de los miembros del equipo resultan primordiales las herramientas para el trabajo a distancia con capacidad para compartir archivos.

#### **Slack**

Se trata de una herramienta de colaboración dedicada a la comunicación de grupos de trabajo que permite el intercambio de ficheros e información. El trabajo tiene lugar en canales, se pueden crear canales a partir de equipos o proyectos y dispone de chat a través del cual se compartirán los archivos.

Permite la conexión con aplicaciones externas que están siendo utilizadas durante el desarrollo como Github, quedando cada avance reflejado y notificado a todos los miembros del equipo.

# Capítulo 3

## Objetivos y metodología

En este capítulo se expone el ‘problema’ que ha motivado la realización de este proyecto, los objetivos y requisitos marcados previamente a su realización y los que se han cumplido tras la finalización de este proyecto.

### 3.1. Motivación

Hoy en día existen aplicaciones de diverso carácter cuyo objetivo es relacionar a gente con un tema común entre ellos. Puedes encontrar aplicaciones para conocer una posible pareja como sería Tinder, otras como Twitch que es una plataforma orientada a los jugadores de videojuegos; todas con un objetivo común por parte de los usuarios.

En mi caso la idea de este proyecto surgió debido a que para quedar con los amigos para jugar al baloncesto hacemos uso de un grupo de Whatsapp, esto limita bastante ya que la cantidad de gente con la que puedas realizar la actividad deportiva se limita a conocidos y podrían ser pocos y que no todos puedan el mismo día.

De esta necesidad surge la motivación de crear una aplicación web con la que aumentar la cantidad de jugadores posibles e incluso fomentar el conocer gente nueva con la que entablar una amistad, en este caso con tu deporte favorito como punto en común.

## 3.2. Objetivos

Analizando los motivos previamente expuestos, extraemos una serie de objetivos que se deben cumplir en la implementación del proyecto. Serían los siguientes:

1. La aplicación debe ser atractiva a la vista del usuario e intuitiva.
2. La aplicación debe estar optimizada.
3. La creación de una partida debe ser rápida y sencilla.

## 3.3. Requisitos

A partir de la motivación debemos plantearnos cuáles son las necesidades básicas del usuario cuando use la aplicación. Dichas necesidades son los requisitos que deberá cumplir la aplicación y que se tendrán en cuenta durante las fases de diseño y posteriormente durante el desarrollo. Hay que tener en cuenta que durante el propio desarrollo de la aplicación pueden surgir nuevos requisitos con motivo de mejorar lo previamente establecido.

Estos serían los requisitos que la aplicación debe cumplir:

### ■ Registro de usuario

1. Debe existir una interfaz para la creación de un usuario, para el posterior uso de la aplicación.
2. Todo el contenido de la aplicación será accesible tras el previo logeado, el contenido no será mostrado sin haberse registrado previamente.
3. El formulario de registro consistirá en un nombre de usuario, contraseña, correo, deporte favorito y datos de la localización del usuario.
4. El acceso será mediante usuario y contraseña, el correo será utilizado para notificaciones.

### ■ Experiencia de usuario

1. El usuario debe ser capaz de crear una partida en el menor tiempo posible.

2. Tras logearse, el usuario debe ver una lista de partidas adaptadas a sus datos (localización y deporte favorito).
3. El usuario debe poder filtrar por deporte y localización, además de distinguir de partidas en las que está inscrito y las que aún no.
4. Ante la posibilidad de que se juegue al aire libre, debemos dotar al usuario de la posibilidad de consultar las condiciones atmosféricas durante la semana.
5. Cada usuario debe poder crear una partida rellenando un formulario.
6. El host o creador de la partida será el único con capacidad para eliminar la partida.
7. El host de la partida podrá añadir a amigos agregados previamente.
8. Los usuarios inscritos en una partida deben recibir notificaciones de los que se inscriban posteriormente.
9. Posibilidad de agregar y eliminar usuarios de una lista de amigos.

#### ■ Servicios

1. Existirán notificaciones de inscripción de usuarios en partidas en las que uno esté previamente inscrito.
2. Notificaciones de eliminación de partida para los jugadores inscritos.
3. Existirá una interfaz a través de la cual visualizar las condiciones atmosféricas, dichos datos serán consultados mediante una API y podrán consultarse por localización específica mediante un buscador.
4. Mediante la elección de deporte favorito durante el registro, se dotará al usuario de un spinner personalizado que se mostrará durante la carga de datos de la aplicación.

#### ■ Contenido principal

1. Un usuario podrá consultar todas las partidas disponibles en una localización mediante un mapa donde se situarán todas las disponibles filtradas por deporte y las partidas en las que ya esté inscrito.
2. Mediante el doble click sobre el mapa se rellenará el campo de localización en el formulario. Si el usuario desea coger su posición actual, bastará con pinchar sobre el botón **‘Coger posición’** que se encontrará en el formulario.

3. Las partidas tendrán un límite de inscripción de 2 jugadores como mínimo y 30 como máximo.
4. La partida podrá ser eliminada únicamente por el creador.
5. Al hacer click sobre un icono de partida sobre el mapa, se mostrará la información de la partida: localización y jugadores inscritos.

#### ■ Perfil

1. El usuario podrá editar su información de perfil: nombre de usuario, correo, deporte favorito, ciudad y código postal.
2. El usuario podrá añadir y eliminar de la lista de amigos.

### 3.4. Metodología y plan de trabajo

#### 3.4.1. Metodología

Durante el desarrollo de este TFG se ha seguido una metodología de realimentación basada en SCRUM. Cada parte de la aplicación final se ha obtenido a partir de prototipos previos hasta obtener la funcionalidad óptima y la mayor sencillez posible durante el uso del conjunto de la aplicación.

Diferenciamos entre distintas etapas: obtención de requisitos, aprendizaje de las tecnologías, desarrollo, fase de pruebas y evaluación del resultado final.

#### 3.4.2. Plan de trabajo

En el plan de trabajo podemos diferenciar las siguientes fases:

- **Fase 1 - Establecimiento de requisitos y búsqueda y aprendizaje de las herramientas:** en esta fase inicial se han extraído los requisitos iniciales de la aplicación, así como las herramientas necesarias para su desarrollo y su aprendizaje.
- **Fase 2 - Desarrollo de prototipos:** es la fase más extensa en cuanto a tiempo, durante el desarrollo de cada parte de la aplicación se ha desarrollado un prototipo como base para su posterior mejora.

La tareas realizadas durante esta fase se agrupan en: establecimiento de requisitos, diseño de base de datos y estructura de los documentos, diseño de interfaces de usuario y servicios requeridos, desarrollo de las interfaces y y servicios y por último pruebas de funcionalidad.

- **Fase 3 - Despliegue y pruebas:** en esta fase final se ha procedido al despliegue de la aplicación y pruebas por parte de usuarios reales a toda la funcionalidad de la aplicación para su posterior mejora ,en caso de ser necesaria.





# Capítulo 4

## Diseño e implementación

Durante este capítulo se va a describir en profundidad los procesos de diseño e implementación del proyecto. En primer lugar se explica la arquitectura utilizada, las herramientas utilizadas durante el desarrollo y por último su despliegue.

### 4.1. Arquitectura utilizada

El modelo de arquitectura elegido ha sido la denominada **MEAN Stack** (figura 4.1), cuyo acrónimo hace referencia a las arquitecturas desarrolladas con MongoDB, ExpressJS, Angular y NodeJS.

- MongoDB: base de datos NoSQL donde se almacenará la información.
- NodeJS: Permite ejecutar javascript en el lado del servidor.
- ExpressJS: Framework Javascript del lado del servidor.
- Angular: Framework Javascript del lado del cliente.

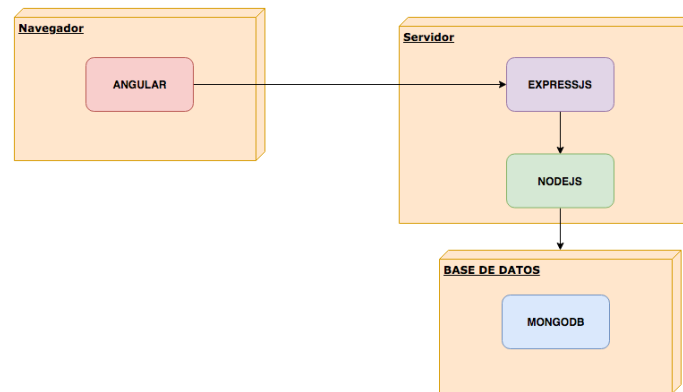


Figura 4.1: Arquitectura MEAN

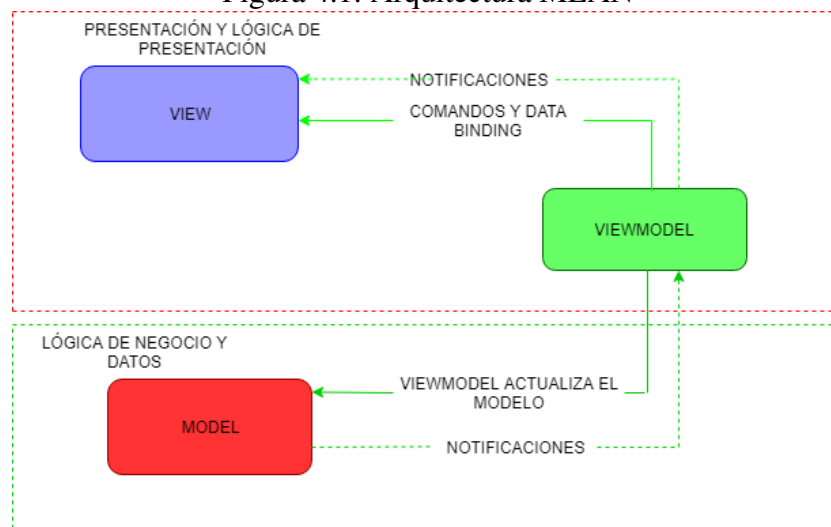


Figura 4.2: MVVM

Estas 4 tecnologías nos permitirán crear aplicaciones escalables y en tiempo real. Dicho modelo de arquitectura se correspondería con el patrón **MVVM** (figura 4.2), en el cual cada vista dispone de su propia lógica y existe un sistema de data-binding entre las plantillas.

## 4.2. Búsqueda de herramientas

A partir de los requisitos establecidos previamente, necesitamos encontrar herramientas con las que construir una aplicación con sistema de geolocalización para el usuario y un mapa donde situar las partidas, consulta del tiempo atmosférico y con una interfaz de usuario (IU) agradable a la vista.

### ***Sistema de geolocalización***

Para la geolocalización existen numerosas APIs como la propia de HTML5 o la de Google. Para esta aplicación se han utilizado ambas, en el caso de querer geolocalizar la posición actual del usuario se utiliza la API de HTML5 a la que previamente hay que dar permisos en el navegador y en el caso de introducir una dirección de forma manual en el formulario de la creación de partida o pulsando 2 veces sobre el mapa se usa la de **Google**<sup>1</sup>.

### ***Mapa***

Existen APIs de mapas de código abierto como **LeafletJS**<sup>2</sup>, para nuestro caso he optado por la de **Google**<sup>3</sup>. En la funcionalidad de esta aplicación no aportará gran diferencia en su uso, sin embargo la opción del Street View nos permitirá ver previamente a la quedada deportiva donde se sitúa la pista, muy útil en caso de no conocer el sitio.

### ***Consulta tiempo atmosférico***

Al igual que para la geolocalización existen diversas APIs, tanto de código abierto como de pago, que se pueden utilizar.

Para esta aplicación se ha utilizado la API de **Openweathermap**<sup>4</sup>, pese a ser de código abierto cierta funcionalidad está restringida a pago. En nuestro caso mostraremos la información del tiempo de los próximos 5 días de la ciudad con la que se registró el usuario y mediante un buscador podrá solicitar la información de la que quiera.

---

<sup>1</sup><https://developers.google.com/maps/documentation/geolocation/intro?hl=es-419>

<sup>2</sup><https://leafletjs.com/>

<sup>3</sup><https://developers.google.com/maps/documentation/javascript/tutorial?hl=es>

<sup>4</sup><https://openweathermap.org/>

## IU

Como uno de los requisitos fundamentales se ha fijado que la aplicación debe ser atractiva al usuario y eficiente en cuanto a que el tiempo de creación de las partidas debe ser el mínimo y lo más intuitivo posible. Para la maquetación se ha utilizado la tecnología **Flexbox** y para ciertos componentes se han usado los de angular material.

Gracias a Flexbox la aplicación se adaptará al dispositivo en el que esté siendo utilizada, readaptando las plantillas en función del tamaño disponible.

### 4.3. Entorno de desarrollo

Tras buscar la herramientas necesarias para el desarrollo empezamos a ‘montar’ el entorno de la aplicación, utilizaremos el IDE **Visual Studio Code**. Nos dotará de herramientas para el desarrollo como una consola de comandos, pluggins que facilitarán el desarrollo del typescript (como es TSLint<sup>5</sup>) e integración de sistemas de control de versiones como es Git.

#### 4.3.1. Creación de la aplicación

En primer lugar deberemos tener instalado NodeJS, el cual será el entorno de ejecución de nuestra aplicación, además de MongoDB.

Una vez hayamos instalado NodeJS deberemos instalar el intérprete de comandos de Angular (CLI), lo que nos facilitará en gran medida la creación de componentes para la parte de FRONT.

Mediante el comando: *ng new xxx* creamos toda la estructura del proyecto angular, dispondremos de comandos como: *ng generate service name*, *ng generate class name*, *ng generate directive name...* para la creación de componentes, servicios, clases y todo aquello que necesitemos.

En todo proyecto NodeJS disponemos de un archivo llamado package.json en el cual se definen propiedades del proyecto como son:

- Nombre de tu proyecto.

---

<sup>5</sup><https://palantir.github.io/tslint/>

- Versión.
- Dependencias
- Repositorio
- Autores
- Licencia

entre otros muchos. Dentro de la propiedad de dependencias incluiremos todas aquellas necesarias para nuestra aplicación que no haya metido por defecto el Angular CLI.

Para el uso de los mapas y angular material incluiremos la dependencia mediante los siguientes comandos:

1. `npm install - -save @agm/core`
2. `npm install - -save @angular/material @angular/cdk`
3. `npm install - -save @angular/animations`

Mediante el flag `- -save` dejaremos guardada esa dependencia en el `package.json`, de modo que ejecutando sobre la carpeta raíz del proyecto `npm install` se intalarán todas las dependencias para la ejecución del proyecto Angular. Todas las dependencias quedan en una carpeta llamada `node_modules`.

Con esto hemos acabado de iniciar la parte FRONT, a continuación vamos con el BACK:

Mediante el comando `npm init` y tras completar una serie de preguntas para inicializar el `package.json` con ciertos parámetros (nombre proyecto, autor...), incluiremos las siguientes dependencias:

1. **express**: dependencia necesaria para la creación de un servidor web `expressJS`.
2. **body-parser**: analiza el body de la petición y lo deja accesible a través del objeto `req.body`.

3. **path**: módulo para construir el path deseado y apuntar a los archivos generados por angular en la carpeta dist tras compilarlo.
4. **express-jwt**: middleware encargado de validar JsonWebTokens enviados en las peticiones.
5. **http**: módulo para crear el servidor y recibir las peticiones.
6. **jsonwebtoken**: módulo encargado de la creación del JsonWebTokens tras logearse con éxito.
7. **mongoose**: librería para interactuar con MongoDB.
8. **nodemailer**: módulo que permite enviar correos electrónicos, necesario para las notificaciones.

Con la instalación de todos estos módulos ya tendríamos listo el BACK con todos los elementos necesarios para recibir las peticiones y actuar en función de cada una.

## 4.4. Diseño de las páginas

En esta sección se realizará el diseño de las partes que compondrán nuestra aplicación, distinguiremos entre la página y los componentes que componen cada una de ellas. Como requisito para poder navegar entre las distintas páginas estableceremos un token, que será generado durante el login y sin el cual serás redirigido a la página de login.

### 4.4.1. Diseño principal y Registro

Nuestra aplicación dispondrá de un header mediante el cuál se navegará por las distintas secciones de la aplicación, dependiendo de la dirección cargada en el navegador se cargará una página y sus correspondientes componentes.

Por defecto la página cargada tras el login será el *HOME*, en caso de no haberse logeado previamente la aplicación le redirigirá al login.

Esto lo hará con la ayuda del módulo *express-jwt*, mediante el cual el servidor comprobará en cada petición si existe dicho token, si es válido y si no está fuera de fecha (dispone de una

duración de validez de 1 hora).

### **Header**

Como se definió anteriormente en los requisitos la página debe ser responsive, por lo que se adaptará a cualquier tamaño de pantalla.

Dispondrá de 4 botones y cada uno de ellos cargará los componentes correspondientes a su página, existirá un quinto botón situado a la derecha del todo del header encargado de cerrar la sesión.

Al reducirse el tamaño de la pantalla se distribuirán todos los botones de forma que entren todos en el header, aumentando su height si es necesario.

#### **4.4.2. Registro de usuarios**

Para el registro de los usuarios definiremos en el archivo app-routing.module el path *registry* y definiremos la carga del componente correspondiente a ese estado. (figura 4.3)

```
const routes: Routes = [
  { path: '', redirectTo: '/login', pathMatch: 'full' },
  { path: 'registry', component: RegistryComponent },
  { path: 'login', component: LoginComponent },
  { path: 'home', component: HomeComponent, canActivate: [AuthGuard] },
  { path: 'weather', component: WeatherComponent, canActivate: [AuthGuard] },
  { path: 'friends', component: FriendListComponent, canActivate: [AuthGuard] },
  { path: 'profile', component: ProfileComponent, canActivate: [AuthGuard] },
  { path: '**', component: NotFoundComponent }
];

@NgModule({
  imports: [ RouterModule.forRoot(routes) ],
  exports: [ RouterModule ]
})
export class AppRoutingModule {}
```

Figura 4.3: Estado registro

El componente estará compuesto por un formulario en el que se rellenarán los siguientes campos:

- **Nombre de usuario.**
- **Email:** para el envío de notificaciones.
- **Contraseña y confirmación.**



- **Deporte favorito:** en función del elegido tendrá un spinner diferente y cargará por defecto partidas cercanas de ese deporte.
- **Ciudad:** para la carga de partidas e información del tiempo.
- **Código postal:** para la carga de partidas e información del tiempo.

El formulario dispondrá de verificaciones mientras se rellena cada campo, una vez rellenados todos y solo cuando todos cumplan los requisitos se desbloqueará el botón aceptar para enviar los datos al BACK y realizar validaciones posteriores como que el usuario no exista ya en la aplicación.

En caso de que no exista el usuario se creará un documento User (figura 4.4) y se almacenará en la colección Users.

```
module.exports = mongoose.model('User', new Schema({
  userName: String,
  password: String,
  email: String,
  sport: String,
  city: String,
  postCode: String,
  registryDate: Date,
  admin: Boolean,
  friends: [String],
  games: [
    {
      _id: mongoose.Schema.Types.ObjectId,
      sport: String,
      postCode: String
    }
  ]
}));
```

Figura 4.4: Documento user

#### 4.4.3. Login

Al igual que en el caso del registro se definirá su estado y componente a cargar correspondiente, dispondrá de un formulario con 2 campos a rellenar que son nombre de usuario y contraseña.

Una vez rellenados se desbloqueará el botón **Siguiente** el cuál enviará los datos al servidor para su validación. En caso de éxito se enviará un token en la respuesta del servidor y se almacenará en el Local Storage, una vez logeado se redirigirá al estado home.

### Token

Para navegar por la aplicación hemos establecido un token que se crea durante el login, dicho token se almacena en el Local Storage. En el módulo encargado del routing se añade una propiedad en el objeto declarado para la carga del componente con su estado correspondiente que se encargará de comprobar si existe dicho token.

Para que disponga de esta funcionalidad se implementa la clase **CanActivate** y como comprobación se tratará de obtener el token declarado como *currentUser*, sólo permitirá la carga de los componentes si la comprobación devuelve true. En caso contrario no lo cargará y redirigirá a la página de login.

```
@Injectable()
export class AuthGuard implements CanActivate {

    constructor(private router: Router) { }

    canActivate() {
        if (localStorage.getItem('currentUser')) {
            // logged in so return true
            return true;
        }

        // not logged in so redirect to login page
        this.router.navigate(['/login']);
        return false;
    }
}
```

Figura 4.5: Implementación CanActivate

```
const routes: Routes = [
  { path: '', redirectTo: '/login', pathMatch: 'full' },
  { path: 'registry', component: RegistryComponent },
  { path: 'login', component: LoginComponent },
  { path: 'home', component: HomeComponent, canActivate: [AuthGuard] },
  { path: 'weather', component: WeatherComponent, canActivate: [AuthGuard] },
  { path: 'friends', component: FriendListComponent, canActivate: [AuthGuard] },
  { path: 'profile', component: ProfileComponent, canActivate: [AuthGuard] },
  { path: '**', component: NotFoundComponent }
];

@NgModule({
  imports: [ RouterModule.forRoot(routes) ],
  exports: [ RouterModule ]
})
export class AppRoutingModule { }
```

Figura 4.6: Declaración en el routing

La comprobación se realizará para todos los estados, exceptuando Login y Registro.

#### 4.4.4. Home

Esta será la página con la funcionalidad principal de la aplicación, lo primero de todo será declarar el estado en el archivo `app-routing.module.ts`.

Como elementos principales tendrá:

##### 1. Formulario de creación de partida:

El componente padre será `home.component`, dentro del cual insertaremos como directiva otro llamado `create-game`. Este componente será el formulario encargado de la creación de la partida, dispone de campos a rellenar que son *Nombre de la partida*, *Deporte*, *Nº límite de jugadores*, *Fecha y hora* y *Dirección*.

Cada campo del formulario tiene una propiedad `required` que una vez rellenados, y si la propiedad `error` (validaciones específicas) de cada campo es `null`, cambia el estado de la propiedad `valid` del objeto `Form` a `true`, con lo que se desbloqueará el botón crear que enviará los datos al BACK para su validación. Toda la funcionalidad del formulario será gracias al módulo `ReactiveForms`, el cual dota al tag *form* de herramientas para validación de datos y manejo de eventos.

Si no existe ya el nombre de la partida en la Base de Datos, se creará el documento `Game` y se almacenará en la colección `Games`, además se actualizará el documento correspondiente al usuario que la creó insertando el `id` (correspondiente al de la colección `Game`) de la partida en el campo `games`, además del código postal y el deporte.

Por último se le añade un botón de **Coger posición** con el que en lugar de introducir la dirección de forma manual, el evento `click` lanzará la petición para obtener la geolocalización mediante la funcionalidad de `HTML5`.

Una vez obtenidas las coordenadas con `navigator.geolocation.getCurrentPosition()`, utilizaremos la latitud y longitud obtenidas para llamar a la API de Google de Geolocation y

obtener los datos asociados como son calle, número, ciudad y código postal. Para el uso de API necesitaremos una key generada desde la página de Google para desarrolladores. En cada llamada a la API debemos incluir dicha key para tener acceso al servicio. Una vez obtenidos los datos solicitados, rellenaremos los campos del formulario para la creación de la partida gracias al data-binding.

## 2. Mapa:

Mediante el módulo *AgmCoreModule* haremos uso de la directiva *agm-map* en el html de la página home. De entre todas las propiedades de entrada nosotros haremos uso de la latitud, longitud y zoom para situar la región del mapa a mostrar, también haremos uso del evento soportado como es el doble click para fijar un marcador en el mapa (por defecto el doble click era para hacer zoom). Le pasaremos la información obtenida del mapa con la acción del doble click (latitud, longitud) a la API de Google para obtener la dirección física de ese punto y añadirla al formulario de creación.

Además de la directiva *agm-map*, utilizaremos *agm-marker* para situar los puntos de las partidas. Los parámetros de entrada serán latitud, longitud, icono y título; el único evento que se controlará es el click sobre el icono que se utilizará para mostrar la información de la partida.

Todos los parámetros de entrada se obtienen mediante un servicio y se asignan a variables de la clase en el *home.component.ts*, en el *home.component.html* se pasan dichas variables a los parámetros de entrada de la directiva.

Se deben definir una serie de filtros de búsqueda para las partidas:

- Input para la ciudad.
- Radio button para seleccionar una de las 4 opciones de deportes disponible.
- Radio button con las opciones de partidas: 'Mis partidas' y 'Otras'.

Nada mas cargar la página home, se obtendrán a través de un servicio los datos del usuario que se pasarán a otro servicio y así cargar de forma predefinida las partidas propias del jugador de su deporte favorito en su ciudad. En caso de no estar inscrito en ninguna se mostrará un mensaje debajo del mapa de que no hay partidas. Al rellenar los filtros y

pulsar un botón **Buscar** se lanzará el mismo servicio de búsqueda inicial pero con los nuevos parámetros.

Una vez obtenidos los datos se le asignarán a los objetos que utiliza el html en el mapa a través del data-binding.

### 3. Información de la partida sobre la que se ha hecho click:

Toda la información de una partida clickeada la guardamos en un objeto llamado game-Clicked, por defecto al obtener la información de todas nos quedaremos con la primera para mostrar su información. Al pulsar sobre el marcador de una partida, obtenemos el campo title y a partir de él obtenemos el objeto del array de las partidas y mostramos su información.

En el documento games hemos guardado un campo llamado **host** donde almacenamos el nombre del creador.

# Capítulo 5

## Resultados

En este capítulo se incluyen los resultados de tu trabajo fin de grado.

Si es una herramienta de análisis lo que has realizado, aquí puedes poner ejemplos de haberla utilizado para que se vea su utilidad.



# Capítulo 6

## Conclusiones

Anteriormente se han expuesto temas como la descripción del problema y motivación para el desarrollo del proyecto y el desarrollo de la solución y análisis del comportamiento en un entorno real. En este capítulo final se recogen las conclusiones del trabajo realizado, así como posibles mejoras y trabajos futuros.

### 6.1. Consecución de objetivos

Con la prueba en entorno real de la aplicación y el análisis de los resultados obtenidos se concluye que las necesidades básicas que motivaron el desarrollo del proyecto han sido satisfechas.

Mediante esta herramienta se ha alcanzado el objetivo principal que era facilitar el desarrollo de tu actividad deportiva favorita (dentro de las 4 opciones disponibles), en la cual era necesaria la presencia de más personas.

Dicho objetivo se ha cumplido mediante algo tan simple como la publicación de un punto de encuentro con una fecha determinada y dando la posibilidad de que se una tanta gente como quiera, en principio existe un límite de personas establecido por el creador de la partida aunque siempre existirá la posibilidad de ir aún no aparecer como uno de los posibles asistentes.

Se ha tratado de que la presentación de datos de la aplicación y navegación fueran lo más intuitivos y agradable a la vista posibles así como que llevará el menos tiempo posible la crea-



ción de partidas.

Más allá de los objetivos cumplidos por dicha aplicación, este proyecto ha cumplido otros como la utilización de tecnologías actuales con las que nunca había trabajado y el enfrentarme a las diversas partes que compone el desarrollo de un proyecto desde cero.

Se puede concluir que los objetivos marcados han sido cumplidos con éxito.

## 6.2. Aplicación de lo aprendido

Durante el grado he cursado una gran cantidad de asignaturas de programación, cada cuál me ha aportado habilidades muy útiles para el desarrollo de este proyecto.

Entre todas ellas puedo destacar 3 como las mas importantes ya que eran específicas del desarrollo web.

- Servicios y Aplicaciones Telemáticas.
- Desarrollo de Aplicaciones Telemáticas.
- Ingeniería de Sistemas de la Información.

La primera asignatura (SAT), se centraba en el aprendizaje de la parte **BACK** cuya práctica final consistió en la realización de una práctica basada en el framework *DJANGO*<sup>1</sup> y la base de datos **SQLite**.

A nivel de diseño FRONT fue muy básico ya que el objetivo fue centrarnos en el tratamiento de datos, fue una introducción al html y css.

La segunda de ellas (DAT) se centró en la parte FRONT, entre las tecnologías impartidas estaban **jquery**, **HTML5** y **CSS3**. Además de la utilización de una librería de diseño web como es *Bootstrap*<sup>2</sup>.

---

<sup>1</sup><https://www.djangoproject.com/start/>

<sup>2</sup><https://getbootstrap.com/docs/4.3/getting-started/introduction/>

La última de ellas (ISI) fue la más completa de ellas, se pusieron en práctica todos los conocimientos adquiridos mediante la realización de un proyecto final el cual se trataba de una plataforma web de minijuegos. Se dividió el proyecto en 3 partes:

1. Plataforma de acceso.
2. Lógica del minijuego.
3. Interfaz de usuario del juego (IU).

Durante la asignatura se estudiaron diversas tecnologías como MeteorJS, NodeJS, MongoDB y javascript como lenguaje. En mi caso tuve que realizar la IU mediante Canvas, por lo que sobre todo desarrolle en javascript.

El conjunto de todas las asignaturas y tecnologías empleadas me han servido como base sólida para el aprendizaje de otras más actuales con las que he realizado esta práctica final. Sin el aprendizaje de HTML, CSS, javascript y jquery la realización de una aplicación en Angular habría sido prácticamente imposible.

El haber aprendido java durante la carrera también ha facilitado el aprendizaje de typescript ya que tienen una estructura similar en la declaración de clases.

## 6.3. Lecciones aprendidas

Con el desarrollo del proyecto he podido desarrollar capacidades hasta entonces explotadas en menor medida, tanto en mi época como estudiante así como durante mi experiencia como profesional en el sector de las aplicaciones web.

En resumen podría enumerar la siguientes:

- Establecimiento de requisitos del proyecto.
- Elección de tecnologías a utilizar en función de los objetivos.
- Capacidad de autonomía para el aprendizaje de nuevas herramientas de desarrollo.

## 6.4. Trabajos futuros

A pesar de haber cumplido los objetivos básicos de lo que se pretendía con la mayor sencillez posible, podrían añadirse mejoras dotando de mayor robustez a la aplicación como:

1. Petición previa al agregar amigos.
2. Inclusión de más deportes.
3. Notificación en la aplicación en lugar de email.
4. Recuperación de contraseña.

Otro tipo de mejoras posibles serían las relacionadas con la ampliación de contenido y usabilidad de la aplicación:

1. Creación de ligas.
2. Publicación de estadísticas de los jugadores.

## **Apéndice A**

### **Manual de usuario**



# Bibliografía

- [1] E. Bonabeau, M. Dorigo, and G. Theraulaz. *Swarm Intelligence: From Natural to Artificial Systems*. Oxford University Press, Inc., 1999.