UNIVERSIDAD
REY JUAN CARLOS

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

Aplicación WEB para la creación de eventos deportivos en localización determinada.

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 2019, siendo calificada por el siguiente tribunal:

Presidente:

Secretario:

Vocal:

y habiendo obtenido la siguiente calificación:

Calificación:

Fuenlabrada, a de de 2019

Dedicado a

mi familia / mi abuelo / mi pareja

Agradecimientos

Después de casi un año de esfuerzo comienzo a escribir este apartado para finalizar mi trabajo de fin de grado. Pese a las dificultades por falta de tiempo, ya que me encontraba trabajando, considero que el resultado final ha sido satisfactorio. Me gustaría agradecer a todas aquellas personas que me han ayudado y apoyado durante el proceso.

En primer lugar me gustaría agradecer a mis padres y hermana, quiénes siempre han estado apoyándome y sufriendo tanto o más que yo.
También me gustaría agradecer a mi pareja, sin su motivación para que me pusiera las pilas y acabará con el trabajo puede que se hubiera alargado más de lo necesario. Ha sido una gran influencia para mí durante toda la carrera y sabe bien lo que se sufre puesto que estudió lo mismo y conoce lo duro que es.

En último lugar me gustaría agradecer a la persona que considero que fue y será allá donde esté mi mayor apoyo, mi abuelo. Gracias a él me decidí a estudiar esta carrera y no me arrepiento, pese al pésimo primer año que tuve siempre confió en mis capacidades y estuvo ahí para aconsejarme sabiamente. Lo que más lamento es que nunca me verá graduarme, por eso me gustaría dedicarle este proyecto a él.

¡Muchas gracias a todos!

Índice general

1. Resumen **1**

2. Introducción **2**

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

3. Objetivos y metodología **13**

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

4. Diseño e implementación **19**

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

4.4.5. Tiempo 33

4.4.6. Lista de amigos y añadir amigos 34

4.4.7. Perfil 35

4.4.8. Logout 36

5. Resultados **37**

5.1. Objetivos de la prueba 37

5.2. Desarrollo de la prueba 37

5.3. Resultado de la prueba 38

6. Conclusiones **40**

6.1. Consecución de objetivos 40

6.2. Aplicación de lo aprendido 41

6.3. Lecciones aprendidas 42

6.4. Trabajos futuros 43

Bibliografía **44**

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 intiutivo 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*[1] *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 ó 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*.

[1] `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 sintáxis XML.
Este lenguaje de marcado se encuentra regulado por el Consorcio **W3C**[2].

Algunas de las mejoras introducidas con el HTML5 son:

- Ampliación del elemento **input** permitiendo tipos de datos como range, email, url, search, entre otras.
- Introducción de elementos de audio y vídeo 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.

[2] `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*[3]. 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**[4], concretamente la v5, para la realización del *Front-End* y **ExpressJS**[5] y **NodeJS**[6] para el *Back-End*[7] .

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*[8] 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.

[3] https://es.wikipedia.org/wiki/Framework
[4] https://angular.io
[5] https://expressjs.com/es
[6] https://nodejs.org/es/
[7] https://es.wikipedia.org/wiki/Front-end_y_back-end
[8] https://bbvaopen4u.com/es/actualidad/api-rest-que-es-y-cuales-son-sus-ventajas-en-el

Existe una gran cantidad de información en internet para su aprendizaje y al igual que su antecesor, mantiene una gran rivalidad con **React**[9], 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.

[9] `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, construye 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

[10] 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

[11] 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 y crear nuevos a partir de ellos.

Un flujo o *stream* es una secuencia de eventos ordenados en el tiempo de los que podemos

[10] `https://developers.google.com/maps/documentation/geocoding/intro`
[11] `https://developers.google.com/maps/documentation/javascript/tutorial`

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**[12] es **Mongoose**[13].

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
- Boolean
- Number
- String
- Date
- Expresiones regulares
- Array
- Documento embebido

[12] `https://es.wikipedia.org/wiki/CRUD`
[13] `https://mongoosejs.com/docs/guide.html`

- ObjectId
- 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[14] (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 autoorganización por parte del equipo.
- Aceptar el cambio de requisitos durante su desarrollo.

Existen múltiples metodologías ágiles entre las que se pueden destacar **SCRUM**[15] y **Extreme Programming (XP)**.

[14] https://www.quora.com/What-is-Object-Document-Mapping
[15] https://es.wikipedia.org/wiki/Scrum_(desarrollo_de_software)

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. Para este proyecto sólo se ha utilizado **Github** como repositorio dónde almacenar el código y desarrollar distintas ramas de desarrollo que finalmente se ha unido en el producto final.

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 unirlo 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:

- 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, es decir, la mayor velocidad de carga posible y que el tiempo de espera por cada acción sea bajo.
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 la previa autenticación, 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 un partida en el menor tiempo posible.
2. Tras autenticarse, 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.
  Las 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 servicios y por último pruebas de funcionalidad.

  Los prototipos desarrollados y su relación con los requisitos de la aplicación son los siguientes:

  En primer prototipo sería el correspondiente a las páginas de Login y Registro, los cuales satisfacen al primero punto de los requisitos expuestos en el apartado anterior.

  El segundo de los prototipos era la página principal dónde se mostraban y creaban las partidas, satisfaciendo los requisitos, de la experiencia de usuario, 1, 2, 3, 5, 6, 7.

  Mediante el tercer prototipo (sección de consulta del tiempo) se satisface el punto número 3 de la experiencia de usuario.

  El cuarto prototipo es el correspondiente a la página de añadir y eliminar amigos, por el cual quedaría cubierto el punto número 9 de la sección de experiencia de usuario.

  El quinto y último prototipo es la página de perfil, quedando satisfecho todo el apartado de requisitos de perfil de usuario, expuesto anteriormente.

- **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.

El despliegue se ha realizado de forma local desde mi ordenador personal, mediante el comando `npm serve` se lanza el servidor que servirá la aplicación. Otro dispositivo conectado en la misma red deberá lanzar una petición contra la dirección IP 192.168.1.44 y el puerto 3000 (definido puerto por defecto en el servidor expressJS).

Previamente se habilitó la configuración del firewall del ordenador para aceptar peticiones externas en dicho puerto; en caso contrario hubiesen sido rechazadas.

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.
Las tecnologías utilizadas para el desarrollo de la parte *Front-End* son Angular, Angular Material y NodeJS; mientras que las utilizadas para el *Back-End* son ExpressJS, NodeJS y MongoDB.
La comunicación entre la parte del cliente y la del servidor se realizará mediante servicios *REST*. El servidor (ExpressJS) realizará la obtención y escritura de datos en la BBDD (MongoDB) mediante **Mongoose**.

- 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.

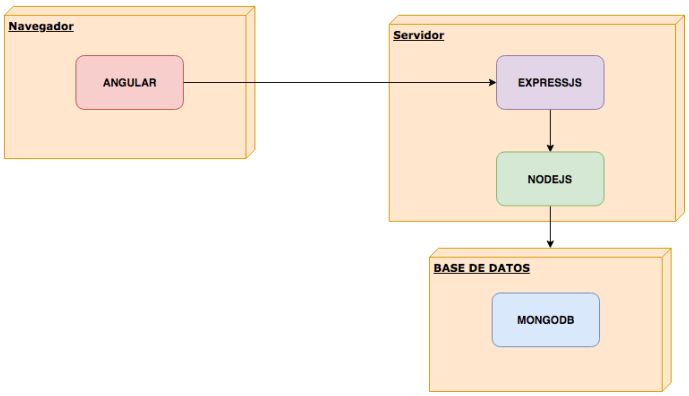

Figura 4.1: Arquitectura MEAN

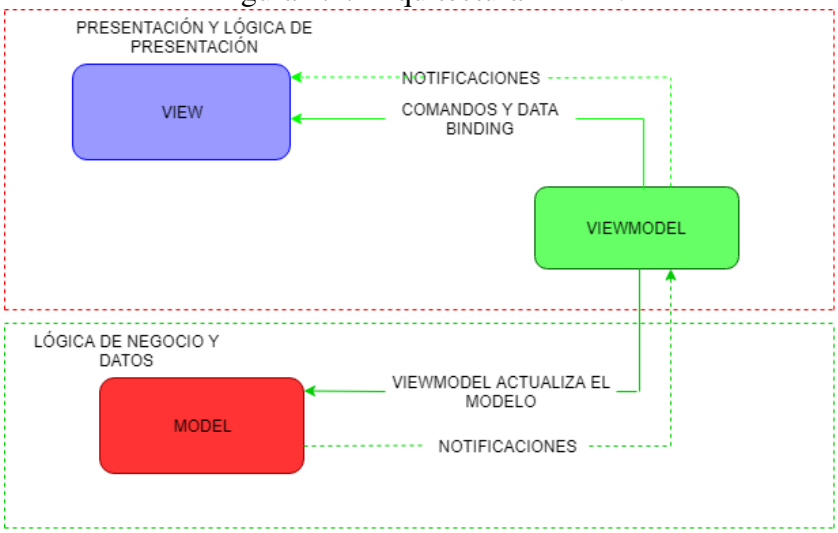

Figura 4.2: MVVM

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**[1].

Mapa

Existen APIs de mapas de código abierto como **LeafletJS**[2] , para nuestro caso he optado por la de **Google**[3]. 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**[4], 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.

---

[1] `https://developers.google.com/maps/documentation/geolocation/intro?hl=es-419`

[2] `https://leafletjs.com/`

[3] `https://developers.google.com/maps/documentation/javascript/tutorial?hl=es`

[4] `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[5]) 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-End*.
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:

[5] `https://palantir.github.io/tslint/`

- Nombre de tu proyecto.
- 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-End*, a continuación vamos con el *Back-End*:

Mediante el comando `npm init` y tras completar una serie de preguntas para inicializar el package.json con ciertos parámetros (nombre proyecto, autor, etc), 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 autenticarse 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-End* 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 autenticado 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-End* 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 desbloquerá 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 autenticado 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** (figuras 4.5 4.6) 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-End* 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* (ver figura 4.7).

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 más 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* (ver figura 4.8).

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 gameClicked, 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. En el caso de ser *host* le damos la opción al usuario de eliminar dicha partida de la colección, eliminándolo tanto en el documento *game* de la colección *games* como en el array del campo *games* en los documento *user* correspondientes.

   En el caso de ser el *host* también se habilitará un buscador con el que se podrán añadir gente de tu lista de amigos, al introducir 3 caracteres sobre el buscador se lanzará un servicio mediante el cuál obtendrá los amigos que coincidan parcialmente con lo introducido. Una vez obtenida la respuesta del servicio desplegará la opciones posibles en el *input* y solo pinchando sobre una se rellenará el input con el valor y habilitará el botón **Añadir**. Al pulsar sobre añadir el componente *friends-searcher* emite un evento que captura el componente home con los valores obtenidos e imprimirá un *popUp* con el mensaje de que ese amigo ya está añadido si ya existía en la información de la partida (objeto gameClicked), si no existe lo añadirá a dicho objeto y lo imprimirá en la pantalla.

   Por último al ser *host* también existirá la opción de eliminar jugadores pulsando sobre un botón al lado de cada nombre de los jugadores. Al pulsarlo se enviarán el **id** de la partida y el del jugador a eliminar, se eliminará del documento *game* obtenido mediante el **id** y

se eliminará el **id** de dicha partida en el documento del jugador eliminado.

Para el caso de no ser *host*, existirá un botón mediante el cual añadirte a la partida, aparecerá en el caso de que el filtro 'Resto' sea el pulsado y que el número de jugadores no haya llegado a su tope (ver figura 4.9).

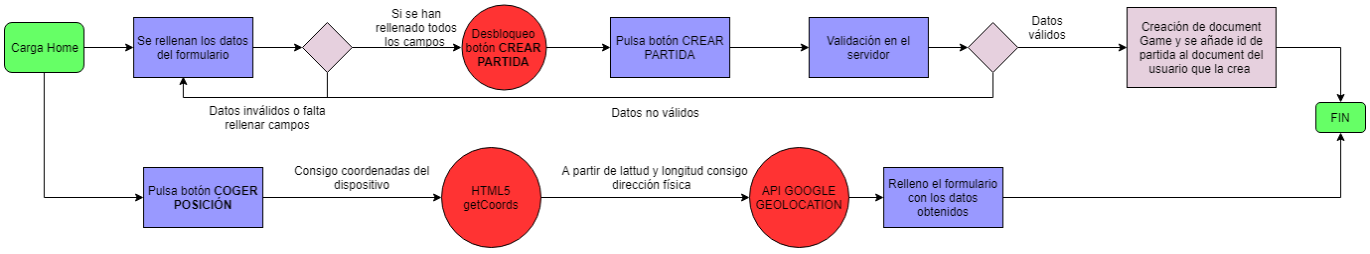

Figura 4.7: Flujo creación de partida

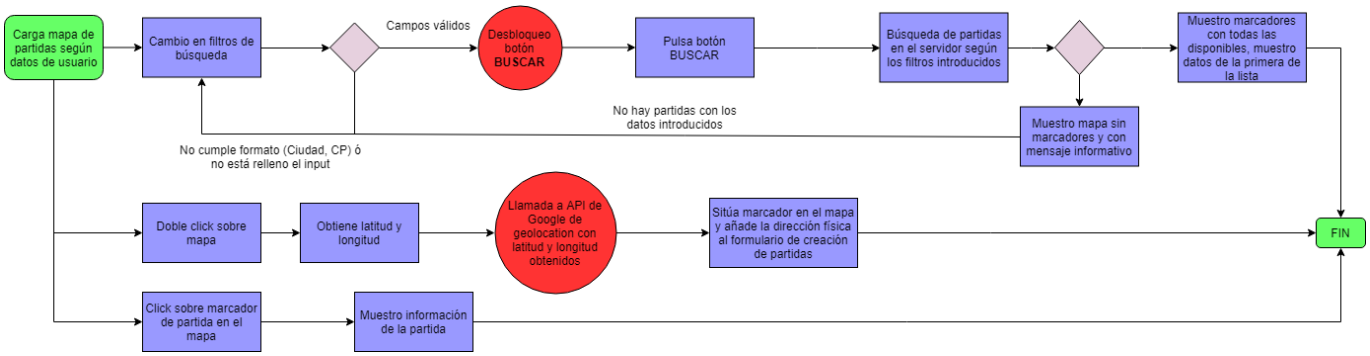

Figura 4.8: Flujo acciones sobre mapa

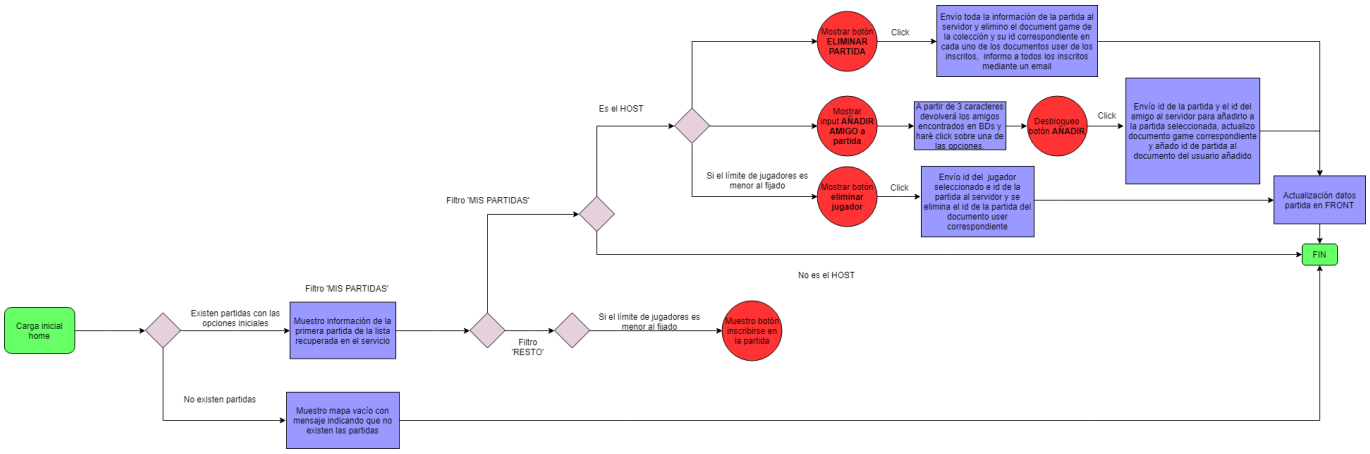

Figura 4.9: Flujo información de partida

4.4.5. Tiempo

En esta pestaña definiremos el tiempo atmosférico de los próximos 5 días, en primer lugar definimos el estado *weather* en el módulo *routing* y la carga del WetaherComponent.

En la carga inicial del componente se lanzará un servicio mediante el cual obtendremos la ciudad con la que se registró el usuario y CP. A continuación mediante otro servicio obtendremos un JSON con todas las ciudades disponibles para consultar, este JSON solo se solicitará en la primera llamada y quedará almacenado para búsquedas posteriores durante la sesión.

Si la ciudad se encuentra entre las disponibles se realizará la llamada a la API de Openweathermap, en dicha llamada hay que incluir una *key* para poder obtener la información además de otros parámetros como son:

- ID de la ciudad obtenida del JSON.
- Key de la API.
- Idioma.
- Unidades métricas.

Al tener la posibilidad de pago y optar por la versión gratuita los datos obtenidos serán de los próximos 6 días y de cada 3 horas. En nuestro caso parsearemos la información de 9:00 a 21:00. Hay que tener en cuenta que toma la hora actual como referencia en la petición y si son las 11:00 cogerá como primera hora de referencia las 12:00 (Empieza a contar desde las 00:00).

De toda la información disponible nos quedaremos con la hora, temperatura media e **id** de la imagen del tiempo con la que formaremos una URL para mostrar un icono propio de la API. Todos estos datos se asignan a un objeto que se renderizará en el HTML mediante el *data-binding*.

Por último añadiremos un buscador para buscar otras ciudades, el proceso será el mismo que antes cambiando la ciudad del usuario por defecto del usuario por la que se desea buscar ahora. En caso de no existir la ciudad dentro de las disponibles se informará mediante una imagen con mensaje.

4.4.6. Lista de amigos y añadir amigos

Aquí distinguiremos 3 componentes:

1. **Friends:**

   El correspondiente a la página principal, contendrá a los otros 2 componentes que son tablas para mostrar la lista de amigos añadidos y la otra con todos los disponibles, en primer lugar se declara dicho estado en el módulo de *routing*. Este componente se encargará de comunicar los otros 2 componentes, cuando se elimine un amigo de la lista se informará al otro mediante un evento y se añadirá a la lista de disponibles. Cuando se agregue uno se informará a la tabla de amigos para que lo muestre en los disponibles.

2. **All-users-table:**

   Haremos uso de Angular Material y de su componente genérico de tablas, dispondrá de paginado, ordenación en función del deporte y número de elementos a mostrar variable (5 y 10). En la carga inicial de la página obtendremos todos los usuarios disponibles, enviaremos en el servicio los siguientes datos:

   - Nombre del usuario.
   - Ordenación: ascendente o descendente.
   - Número de elementos.
   - Página.
   - Filtro: por defecto será *null*, se modifica en caso de introducir un nombre específico o parcial para la búsqueda.

   En la primera carga devolverá los 5 primeros usuarios ordenados alfabéticamente por deporte. Si modificas la ordenación lanzará el servicio automáticamente, al igual que si modificas el número de elementos o cargas la siguiente página.
   Existe la posibilidad de buscar un nombre de usuario particular, cada vez que se introduzca un carácter en el *input* sobre la tabla se lanzará el servicio con los parámetros fijados. En el caso de no encontrar resultados mostrará un mensaje bajo la tabla.

Al pulsar sobre un botón en en cada fila de la tabla, añadirá el usuario correspondiente y enviará un evento al componente padre (Friends) para que se lo comunique al componente de la tabla que muestra la lista de amigos añadidos, que lanzará el servicio para obtener la lista y actualizará la tabla.

3. **Friends-table:**
   Para esta tabla el servicio inicial obtendrá la lista de amigos del usuario enviando los mismos parámetros para la paginación, etc.
   En este caso dispondrá de un botón eliminar, cuando se elimine el usuario seleccionado correctamente se realizará la misma comunicación entre componentes que cuando se agregaba, sin embargo en este caso será la tabla donde se muestran todos los usuarios disponibles la que lanzará el servicio para obtener la lista actualizada.

4.4.7. Perfil

Debemos disponer de una página donde editar los datos guardados cuando nos registramos, al igual que con el resto lo primero será registrar la página en el *routing*.

El componente mostrará una tabla con los datos básicos del usuario:

- Nombre de usuario.
- Correo.
- Contraseña.
- Deporte.
- Ciudad.
- CP.

Cada campo se podrá modificar, una vez enviados los datos al servidor se realizarán las validaciones necesarias para cada campo como que el nombre o correo sean únicos.

La modificación de contraseña cargará un componente con 3 campos a rellenar, la contraseña actual, contraseña nueva y confirmación de la contraseña. Será el componente *new-password* el encargado de validar que la nueva y la confirmación coincidan y que se han rellenado los 3 campos antes de desbloquear el botón para su validación en el servidor.

Al pulsar sobre el botón **Confirmar** se le enviará la información al servidor. En el caso de que al tratar de modificar algún dato el servidor fallará o no pasará la validación se informará mediante un *popUp*. Si se modifica correctamente se informará mediante un mensaje sobre la tabla, con un botón para quitar el mensaje.

4.4.8. Logout

En el *header* se declararon los botones de navegación que cargaban cada estado y su componente correspondiente. En el caso del *logout* lo único que hará será llamar al servicio *authenticationService* y la operación *logout*, eliminará datos propios de la clase usados para la petición en el resto de servicios que son:

- Nombre de usuario.
- Token de sesión.
- ID del usuario.

También se eliminará del **Local Storage** el objeto *currentUser* y se redirigirá al *login*, completando el cierre de sesión.

Capítulo 5

Resultados

Como última fase del proyecto se debe realizar un prueba funcional de la aplicación desarrollada, cuyo objetivo es comprobar la experiencia de los usuarios y verificar que los requisitos definidos inicialmente han sido satisfechos y depurar posibles fallos de la aplicación.

5.1. Objetivos de la prueba

El objetivo principal de la prueba es obtener resultados sobre la funcionalidad de la aplicación mediante el uso por parte de los usuarios. El producto final debe cumplir los requisitos marcados inicialmente entre los que destaca que sea los más intuitivo y rápido posible la creación de partidas.

5.2. Desarrollo de la prueba

Para el desarrollo de la prueba era necesario disponer de un dispositivo, ya fuera ordenador de mesa, portátil o móvil. Como servidor he utilizado mi ordenador personal.

Una vez lanzada la aplicación, se comienza con la prueba. La prueba está compuesta de los siguientes pasos:

1. Registro de perfil.

2. Login.

3. Añadir/eliminar amigo.

4. Crear un partida.

5. Eliminar una partida creada.

6. Añadirse a una partida.

7. Consultar el tiempo.

8. Modificar datos del perfil.

Tras la realización de estos pasos se ha rellenado un encuesta de valoración de usabilidad, se puntúa de 0 a 10 ciertos parámetros de calidad.

5.3. Resultado de la prueba

Tras la prueba se han establecido los puntos más importantes a valorar acerca de la aplicación: usabilidad, utilidad, diseño e intuitividad.

Durante la prueba no se dieron indicaciones de cómo se podía obtener la ubicación para la partida, pese a ello descubrieron rápidamente que mediante el doble *click* sobre el mapa podía obtenerse automáticamente la dirección física.

En la gráfica 5.1 se pueden observar las puntuaciones de cada usuario, pese a no tener datos suficientes se puede observar que en general ha obtenido una gran valoración. Destaca la usabilidad (fluidez de la app) y la utilidad.

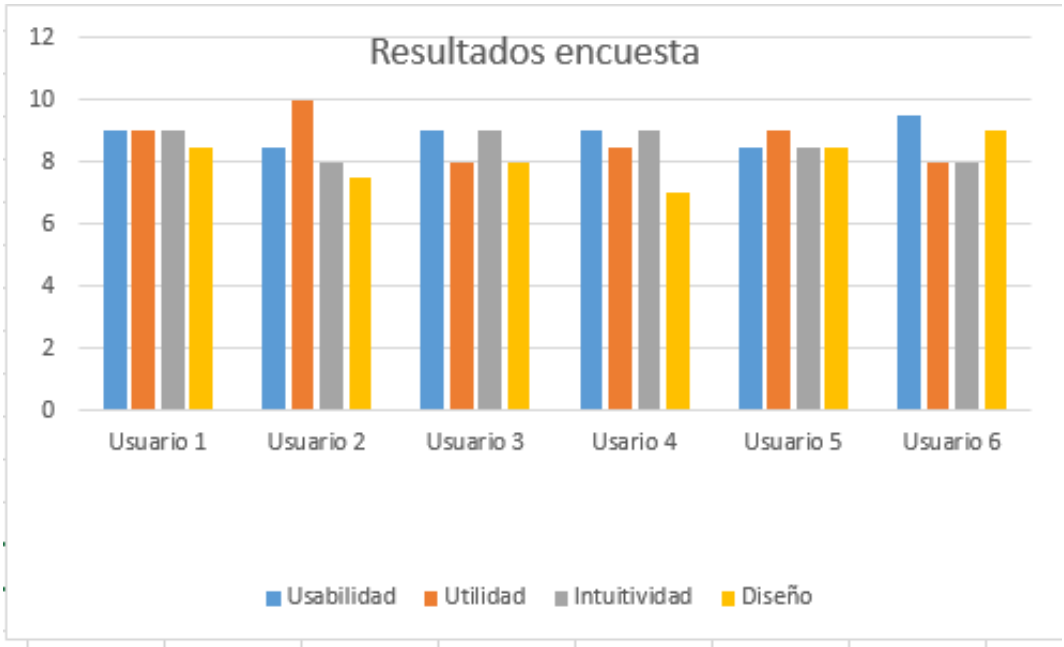

Figura 5.1: Encuesta

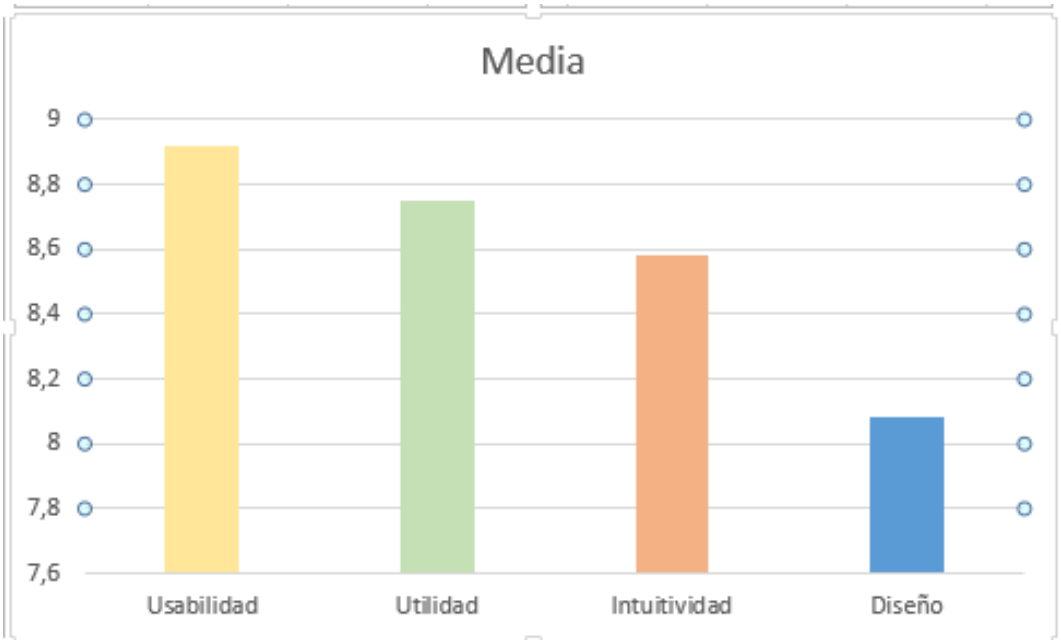

Figura 5.2: Nota media

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.
- Ingenería de Sistemas de la Información.

La primera asignatura (SAT), se centraba en el aprendizaje de la parte *Back-End* cuya práctica final consistió en la realización de una práctica basada en el *framework DJANGO*[1] y la base de datos **SQLite**.
A nivel de diseño *Front-End* 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-End*, entre las tecnologías impartidas estaban **jquery, HTML5 y CSS3**. Además de la utilización de una librería de diseño Web como es *Bootstrap*[2].

[1] `https://www.djangoproject.com/start/`
[2] `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.

Bibliografía

[1] Web oficial para angular material: https://v5.material.angular.io/

[2] Web oficial para angular5: https://v5.angular.io/docs

[3] Web oficial para expressjs: https://expressjs.com/es/

[4] Web oficial para mongodb: https://www.mongodb.com/es

[5] Web oficial para mongoose: https://mongoosejs.com/

[6] Web oficial para nodejs: https://nodejs.org/es/

[7] Web oficial para npm: https://www.npmjs.com/

[8] Web oficial para rxjs: https://rxjs-dev.firebaseapp.com/