TRABAJO FIN DE GRADO

INGENIERÍA INFORMÁTICA

Plataforma social para actividades deportivas

Plataforma web y aplicación móvil para realizar actividades deportivas

Autor

Pablo Delgado Garcia

Director

José Manuel Benítez Sánchez

Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación

—

Granada, 07 de Julio de 2020

SPORTIFY

Plataforma social para actividades deportivas

Plataforma web y aplicación móvil para realizar actividades deportivas

Autor
Pablo Delgado García

Director
José Manuel Benítez Sánchez

Departamento de Ciencias de la Computación e Inteligencia Artificial

—

Granada, 07 de Julio de 2020

SPORTIFY: Plataforma social para actividades deportivas

Pablo Delgado García

Palabras clave: Deporte, App, Aplicación, Web, PHP, Social ,Laravel

Resumen

El objetivo de este proyecto es diseñar e implementar un software formado por dos grandes componentes: una aplicación móvil Android y una plataforma web, que compartan información a través de una base de datos común. Sus principales funcionalidades están centradas en la creación de eventos deportivos por los usuarios que permiten establecer relaciones fuera del sistema facilitando el encontrar gente con la que realizar actividades.

El proyecto surgió por la necesidad propia, y de conocidos cercanos, de buscar personas con las que realizar actividades deportivas. Numerosas veces me encontré en la situación tener ganas de, por ejemplo, jugar a fútbol, pero no encontrar suficientes conocidos con los que juntarme. En ocasiones el hecho de faltar 2 o 3 personas para completar equipos hacían imposibe jugar y finalmente desistíamos, pues nuestro circulo de conocidos es limitado.

El sistema trata principalmente de eliminar esa barrera delimitadora que es nuestor círculo social y extenderlo a todos los usuarios de la aplicación. Esto nos permite contar con nuestro conocidos, y además dar a conocer nuestra oferta de actividad a más gente.

Para el desarrollo será necesario un servidor que albergará principalmente una base de datos con la información del sistema y los diferentes archivos tales como imágenes o los diferentes archivos de la página web.

A lo largo de este documento se describirán aspectos como la metodología utilizada, el análisis de requisitos, el diseño, la implementación, las pruebas y diferentes aspectos relacionados con el desarrollo del proyecto.

SPORTIFY: Social network for sports activities

Pablo Delgado García

Keywords: Sport, App, Application, Web, PHP, Social, Laravel

Abstract

The objective of this project is to design and implement a software consisting of: an Android mobile application and a web platform, that share information through a common database. Its main features are focused on the creation of sporting events by users who establish relationships outside the system, making it easier to find people with whom they carry out activities.

The project arose from my need, and from close acquaintances, to find people with whom they carry out sports activities. Many times I found myself in the situation wanting, for example, to play soccer, but not finding acquaintances with whom to get together. Sometimes, the lack of 2 or 3 people to complete teams made it impossible to play and we finally gave up, because our circle of acquaintances is limited.

The system mainly tries to eliminate that delimiting barrier that is our social circle and extend it to all users of the application. This allows us to count on our acquaintances, and also make our offer of activity known to more people.

For the development, a server will be necessary that mainly houses a database with the system information and the different story files such as images or the different files on the website.

Throughout this document aspects such as the methodology used, the requirements analysis, the design, the implementation, the tests and different aspects related to the development of the project are described.

Yo, **Pablo Delgado García**, alumno de la titulación TITULACIÓN de la **Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación de la Universidad de Granada**, con DNI 74746981V, autorizo la ubicación de la siguiente copia de mi Trabajo Fin de Grado en la biblioteca del centro para que pueda ser consultada por las personas que lo deseen.

Fdo: Pablo Delgado García

Granada a 07 de Julio de 2020 .

D. **José Manuel Benítez Sánchez**, Profesor del Área del Departamento Ciencias de la Computación e Inteligencia Artificial de la Universidad de Granada.

Informa:

Que el presente trabajo, titulado ***Plataforma social paraactividades deportivas, Plataforma web y aplicación móvil para realizar actividades deportivas***, ha sido realizado bajo su supervisión por **Pablo Delgado García**, y autorizo la defensa de dicho trabajo ante el tribunal que corresponda.

Y para que conste, expiden y firman el presente informe en Granada a 07 de Julio de 2020 .

El director:

José Manuel Benítez Sánchez

Agradecimientos

Agradecer en primer lugar a mi familia. Tanto mis padres por ayudarme a poder realizar los estudios y su apoyo, al igual que el de mis tíos y tías con sus ánimos. Mi grupo de amigos desde el instituto, siempre dispuestos a ayudar, divertirse y desconectar. A mi pareja, que me ha acompañado durante todo el último curso, dandome apoyo y motivación día a día. Pero si hay gente sin la que me hubiera sido imposible terminar esta carrera sería el grupo "*qe"; un grupo de amigos desde primero de carrera hasta el final, al que se fue incluyendo gente a lo largo del grado. En especial a Cecilia y Carlos, pilares desde el principio haciendo que madrugar cada día no pareciese una obligación; pero sobretodo Nabil. Nabil no solo me ha ayudado infinidad de veces en términos educativos, ha pasado con creces el nivel de compañero a amigo con todas sus letras y en mayúsculas. Por último a Miguel, amigo desde los 0 años y compañero de piso desde mi primer año en Granada, sin él huiera sido todo mucho más difícil.
Muchas gracias a todos.

Índice general

Índice de figuras

Capítulo 1

Introducción

A continuación se desarrollará una introducción sobre diferentes aspectos del comienzo del proyecto, información a tener en cuenta desde el principio y que marca por tanto el devenir del desarrollo. Desarrollaremos temas como la motivación que lleva a elegir este proyecto, el obteivo del mismo; y la estructura que seguirá y compondrá este documento.

1.1. Motivación

La propuesta original surgió charlando con mis compañeros de piso. Tratábamos de encontrar gente con la que ir a jugar a fútbol durante el fin de semana, pero no resultaba fácil encontrar al menos 10 personas utilizando a nuestros conocidos; y no habían sido pocas las veces en las que se cancelaba el plan por no encontrar gente suficiente.

Con esta problemática surgió la idea de una web/aplicación similar a otras como *BlaBlaCar* o *Wallapop* que tratan de conectar a gente desconocida para suplir cierta necesidad. Fue entonces cuando tomé nota de la idea y comencé a detallar de forma más precisa algunos aspectos.

La idea es tener un sistema web para acceder principalmente desde un ordenador y una aplicación móvil para dispositivos Android con las principales funcionalidades de la web, además de ciertas herramientas útiles para el usuario como marcador o reloj.

Cuando comenté la idea tanto a mis compañeros de piso como a otros amigos todos coincidían en que sería realmente útil disponer de esta aplicación por la cantidad de veces que nos habíamos encontrado en esta situación. Además, me aportaron ideas que podrían formar del proyecto y algunas lo han hecho, como la existencai de un grupo de conversación en cada evento y la posibilidad de tener un marcado y reloj en la aplicación móvil (aunque

esto ya lo tenía en mente).

1.2. Objetivos

El objetivo principal de este proyecto es crear una plataforma online que facilite la conexión entre diferentes usuarios. Esta iniciativa busca ofrecer la posibilidad de crear eventos en los que estos usuarios se apuntan y participan. Poder acceder a foros para compartir experiencias, dudas y opiniones será otra de las facetas de este sistema.

Esto podrá potenciar el espíritu deportivo de aquellas personas que no quieran ejercitarse solas, o no tengan un grupo de amigos cercano con sus gustos y aficiones. Tener la posibilidad de encontrar gente si, por ejemplo, te faltan 3 personas para jugar a baloncesto, puede hacer que mucha gente no tenga que cancelar sus planes.

1.3. Estructura del documento

El documento se ha estructurado teniendo en cuenta principalmente la metodología software elegida.

Este capítulo 1, **Introducción**, pone en contexto del problema a resolver y la manera elegida para ello. A continuación se detallan los primeros avances relacionados con la elección del tutor y nuestra toma de contacto.

En el segundo capítulo, **Estudio previo**, tratamos de estudiar plataformas o sistemas similares para ver qué podemos mejorar de ellos o incorporar a nuestro sistema, y plasmaremos la funcionalidad del sistema.

El capítulo 3, **Metodología y planificación**, desarrolla la metodología software elegida para el proyecto, describo la planificación establecida y un presupuesto en función del coste material y personal.

Tras esto comenzamos con los pasos propios de la metodología seleccionada: metodología en cascada. Por tanto, procedemos a detallar la **Fase de análisis** en el capítulo4: estructuramos y describimos los requisitos del sistema, tanto funcionales como no funcionales, de información junto con diagramas de casos de uso. Continuamos con la **Fase de diseño** en el capítulo 5: proponemos una solución que nos garantice un buen funcionamiento del sistema y el cumplimiento total de los requisitos planteados haciendo uso diferentes estructuras como bocetos, diagramas de flujo y el modelado de la base de datos del sistema. El siguiente paso se detalla en el capítulo 6, **Fase de implementación**: describimos el proceso de codificación y desarrollo del producto como tal, acabando con un sistema funcional; aquí se incluye tanto el desarrollo de la página web como de la aplicación móvil. Tras esto

llega el momento de la **Fase de pruebas** en el caítulo 7: debemos verificar y validar que el sistema realiza las funciones deseadas y de forma correcta.

Para acabar, el caítulo 8, **Conclusiones**, trata de analizar el resultado del desarrollo del proyecto en relación a los objetivos planteados y los logrados. Terminamos el documento mostrando la información bibliográfica y de documentación.

Capítulo 2

Estudio previo

Al llegar el momento de proponer las Trabajos de Fin de Grado y la elección de tutor, conseguí ponerme en contacto con José Manuel Benítez Sánchez, quien desde el principio fue receptivo con el proyecto y me ayudó a definir determinados aspectos para poder aceptar el proyecto.

Tras la primera reunión con el tutor, me marcó las pautas para comenzar el proyecto. Estos primeros pasos consisten en hacer un breve estudio sobre plataformas similares actuales y mostrar la funcionalidad del sistema.

2.1. Estudio etnográfico

Antes de plantear y dejar cerrada la idea del proyecto es necesario hacer un estudio etnográfico que nos permita conocer sistemas similares ya existentes, tanto para incluir algunas de sus funcionalidades como para tratar de mejorarlas y obtener una ventaja sobre estos sistemas.

Me voy a centrar en dos aplicaciones: **Timpik**[11] y **Meetup**[12]. En primer lugar hay que tener en cuenta que ambas plataformas están desarrolladas por empresas y grupos, además de no disponer de un tiempo limitado, por tanto el resultado final que yo obtenga al acabar este proyecto no tendrá la totalidad de esas funcionalidades y seguramente no tenga un resultado tan profesional. Por tanto, de ambas he obtenido ideas para incluir en mi proyecto.

En ambos casos se dispone tanto de una aplicación móvil como de versión web, en las que podemos iniciar sesión y registarnos. Aquí es donde yo añado y diferencio un usuario individual de un usuario que representa a una entidad o empresa, lo que permite gestionar el alquiler tanto de pistas de deporte como de equipamiento para el mismo, una funcionalidad que aparece de forma similar en Timpik. Hay que destacar que Meetup no es solo para

eventos deportivos, lo cual le permite tener más usuarios pero en caso de que solo te interesen este tipo de eventos quizá sería mejor usar Sportify al estar centrada en este tipo.

Por otro lado, de Timpik obtuve la idea de, no solo tener los eventos para organizarlos, sino que estos mantuvieran los resultados y permitiesen verlos. Además, hasta donde he probado estas plataformas, no poseen herramientas que ayuden a la ejecución del propio evento como podrían ser el uso de marcador y cronómetro en la aplicación móvil, algo que puede ser realmente útil.

En resumen, siendo cierto que ya existen plataformas similares y en general más completas (ya sea por el tiempo disponible de desarrollo o por ser creadas por una empresa con muchos recursos y un equipo de desarrolladores), creo que la visión de Sportify y su sencillez juegan a su favor, pues entrar a una plataforma y tener muchas opciones está bien, pero hay usuarios a los que quizá les pueda 'abrumar' y prefieran algo más directo y posiblemente más intuitivo.

2.2. Funcionalidad

Desde un principio era evidente que sería fundamental tener claros los conceptos sobre la funcionalidad del sistema.

Tras comentar con el tutor la primera aproximación, me puntualizó diferentes aspectos que se incluyeron en la funcionalidad final.

En este apartado se describirán las funciones del sistema desde una visión más general, pues en el 4 punto de este docuemnto (fase de análisis) se describirá de forma precisa cada una de las funcionalidades.

El sistema mantendrá un sistema de usuarios diferenciando entre adminstradores, usuarios individuales y usuarios entidad. El administrador tiene la posibilidad de crear, eliminar y modificar a cualquier tipo de usuario; tienen control sobre elementos como los eventos y foros. El usuario individual puede darse de alta en el sistema y configurar su perfil. Su función principal es la de crear/unirse a eventos, junto a las de inscribirse en equipos, participar en foros y grupos, valorar a otros usuarios, ser amigo de otros usuarios y alquilar equipamiento y pistas. El usuario entidad es similar al individual pero representa a una empresa/organización que dispone de ciertas pistas/establecimientos, equimaiento para alquilar por parte de los usuarios...

Esta es la funcionalidad completa, que es de la que dispondrá el usuario a través del portal web; sin embargo, las funcionalidades de la aplicación móvil son reducidas, pensando en la simplicidad y uso concreto de ésta. En el móvil podremos inicar sesión, ver todos los eventos, consultar nuestros eventos,

apuntarnos/desapuntarnos de eventos, ver nuestros equipos, desapuntarnos de dichos equipos, acceder a los grupos de conversación de los eventos y tener a nuestra disposición un cronómetro y un marador que puedan ser útiles a la hora de desempeñar el evento.

Capítulo 3

Metodología y Planificación

3.1. Metodología de Software

Como se nos ha explicado por activa y por pasiva durante la carrera, y lo que hace que se nos considere *ingenieros* informáticos es el hecho de seguir ciertas directrices que conocemos como **metodologías de software**.

Por tanto, es evidente que para la consecución de un proyecto de este calibre haría que elegir uno de estos marcos de trabajo para estructurar, planificar y controlar el desarrollo del mismo. Durante la carrera hemos visto las diferentes metodologías de software que existen hasta el momento. En asignaturas obligatorias como Fundamentos de Ingeniería de Software u otras de la rama de Ingeniería de Software como Dirección y Gestión de Proyectos o Metodologías de Desarrollo Ágil nos han explicado las principales metodologías. Podemos diferenciar dos grandes grupos dentro de estas metodologías: tradicionales y ágiles.

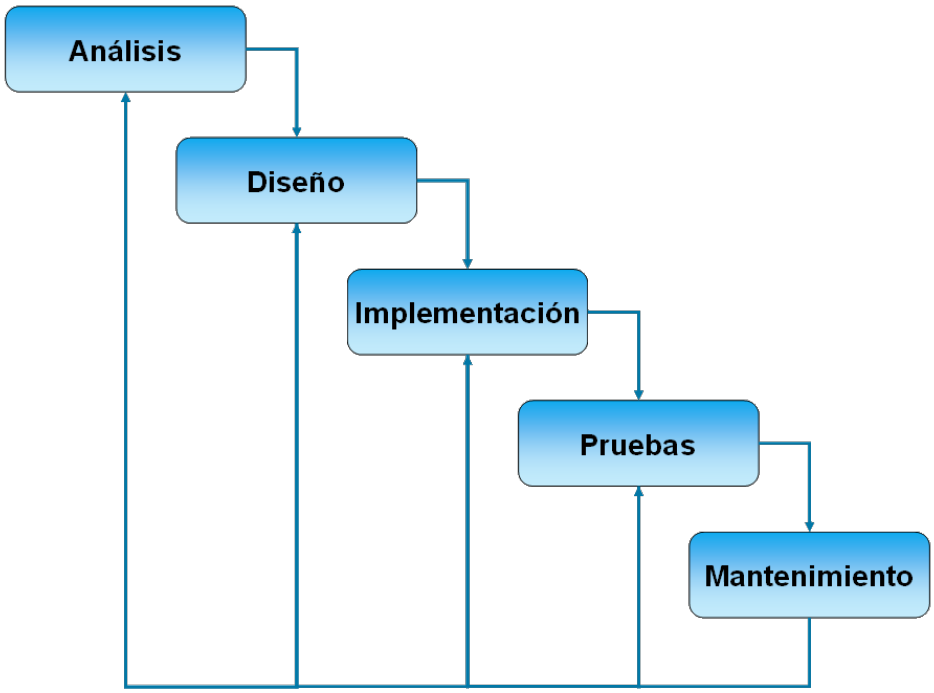
Tras evaluar diferentes opciones y estar familiarizado con estas, era el momento de tomar una decisión.

Desde un principio tuve claro que elegiría una metodología tradicional. Le principal razón para esto era elhecho de que son las que más hemos visto durante la carrera y se me hacía más afín. Además, estas metodologías aportan un gran seguimiento de cada fase, lo que me ayudaría a controlar mejor el desarrollo; es un proceso intuitivo en cuanto al orden en la división de las fases del proyecto; y al no tener un cliente como tal, su poca implicación no es algo que influya. También el hehco de que la planificación sea tan trascendente hace que me mentalice de tener que cumplir los plazos para no retrasar el resto del proyecto[1].

Ahora de entre las tradiconales debía elegir una en concreto. Me dacanté por el **modelo en cascada**. Fue el primer método utilizado par ael desarro-

llo de software, y carece de repetitividad como puede ser el modelo iterativo o en espiral. Esta compuesto por diferentes fases sucesivas, que no permiten el comienzo de una sin la finalización de la anterior.[2]

Figura 3.1: Diagrama modelo en cascada



De nuevo tuve en cuenta mi experiencia durante la carrera y en desarrollos propios; es una metodología con la que me siento cómodo. Debido a su estructura lógica, favorece que evitemos los errores conceptuales al dedicar bastante tiempo al principio a la organización y estudio del problema. Esto también hace que desde un principio podamos hacernos una idea de la duración del proyecto y su coste. Además, uno de sus problemas es la falta de capacidad de trabajo simultáneo entre varios equipos, pero al ser un proyecto individual esto no es problema. Tampoco nos influye uno de sus factores negativos, lo complejo de hacer entender al cliente las especificaciones. Por tanto, tenemos a nuestra disposición la mayoría de sus ventajas centradas en su gran nive de organización y planificación, y dejamos de lado algunos de sus inconvenientes relacionados con la aplicación de cambios (no todos).

3.2. Planificación

Tras estas primeras decisiones, y una vez elegida la metodología de desarrollo, era necesario llevar a cabo una planificación general del proyecto. Esta planificación no se centraría en tareas concretas, sino en los aspectos más generales y cuánto deben tomarme cada uno de ellos.

La planificación quedó dividida en 7 partes más una de "descanso"por época de exámenes. La primera fase ya la había realizado, pero la dejé señalada en el diagrama: elección de metodología, herramientas y descripción de la funcionalidad. Para ello me tomé dos semanas; dejando después tiempo de descanso pues lo estaba compaginando con 5 asignaturas que cursaba durante el primer cuatrimestre, lo que me impedía dedicarle todo el tiempo deseado. Tras esto llevé a cabo la planificación aquí descrita, consensuada con el tutor. Durante el mes de diciembre me centré en la obtención de los requisitos del sistema y el diseño de la base de datos, campos que procederé a detallar más adelante. El mes de enero estuvo ocupado por los examenes finales, cierto descanso y el comienzo de ls prácticas de empresa desed el 7 de enero (prácticas que se mantienen durante todo el desarrollo del proyecto). A partir de aquí comienza el desarrollo en sí: desde febrero hasta mediados de abril se desarrolla el sistema web junto a la base de datos; y el resto de abril y mayo dedicado a la implementación de la aplicación móvil en Android. Desde aquí hasta el final del proyecto se ultiman los detalles de la documentación necesaria y se terminan de realizar las pruebas correspondientes para tratar de detectar posibles errores y tener tiempo para solucionarlos.

La planificación se comenzó a cumplir como se indica en la Figura 3.2, excepto cierto retraso antes de comenzar el desarrollo. Tras la época de exámenes fue necesaria una reunión con el tutor para recibir su *feedback* de lo realizado hasta el momento, y llevé a cabo las modificaciones pertinentes y actualización de la documentación y memoria para llevarla al día y qu eno quedase como carga para las últimas semanas.

A esto se debe añadir una produccion más lenta de la esperada, ocasionada por la situación de confinamiento vivida durante el desarrollo, junto con el comienzo de una enfermedad que me fue diagnosticada durante esta época (colitis) que me dificultaba la concentración y me quitaba tiempo al tener que ir a las diferentes citas médicas y realización de pruebas (incluyendo dos colonoscopias). Estos retraso han mermado en cierto nivel el desarrollo tanto del proyecto como de la memoria, lo que ha limitado algunos aspectos que se preveían inicialmente para el proyecto pero que finalmente no han podido incluirse principalmente por falta de tiempo; pero que podrían, junto a otros, formar parte de unas posibles vías futuras de desarrollo en caso de continuar el proyecto más allá del Trabajo de Fin de Grado.

3.3. Presupuesto

Algo importante a la hora de desarrollar un proyecto de software es tener en cuenta si su coste es asumible, o saber si es rentable económicamente en caso de obtener beneficio del sistema. Esto implica analizar los recursos tanto humanos como tecnológicos, darles un valor y calcular un presupuesto, al menos aproximado, para conocer a qué nos enfrentemaos en términos económicos. Para ello, realizaré un presupuesto aproximado inical según lo que tengo previsto.

En el apartado humano solo me encuentro yo como diseñador, programador, analista,... por tanto debo estalecer un precio por hora de trabajo y conocer la cantidad horas trabajadas. Dentro de este total de horas podemos dividir a grandes rasgos entre horas de programación/desarrollo y horas de documentación. Como ya tenemos una planificación estrucutrada, aproximamos las horas diarias de trabajo y hacemos el cálculo.

Para calcular el coste de una hroa de trabajo utilizaré datos estadísticos. El sueldo medio de un ingeniero informático recién egresado ronda los 20000€/año; esto si establecemos como horas efectivas de trabajo unas 1800 horas/año (quitando festivos, fines de semana,...), nos da un precio de unos **11€/hora**.

Coste documentación

*Días documentación * Horas documentación/día=Horas documentación*
68 días * 2 horas/día = **136 horas**
*Horas documentación * Precio por hora = Coste documentación*
136 horas * 11€/hora = **1.496 €**

Coste programación

*Días programación * Horas programación/día=Horas programación*
121 días * 3 horas/día = **363 horas**
*Horas programación * Precio por hora = Coste programación*
363 horas * 11€/hora = **3.993 €**

Por tanto, el coste de personal total del proyecto es de **5.489 €**. Y este será el grueso del coste del proyecto. En cuanto al coste de equipo lo fundamental es un ordenador. Haciendo búsquedas podemos encontrar un ordenador con las características necesarias tales como 8 GB de RAM, un procesador Intel Core i5-8400, 100TB de HDD y 128GB de SSD por unos 550€(fuente PcComponentes[13]). Para calcular el valor de su uso debemos tener en cuenta su valor inicial (550€), su valor residual (valor que tendrá

cuando acabe su vida útil, suponemos 50€) y su tiempo de vida útil (de media, un ordenador dura 5 años). Con esto calculamos la depreciación anual del equipo:

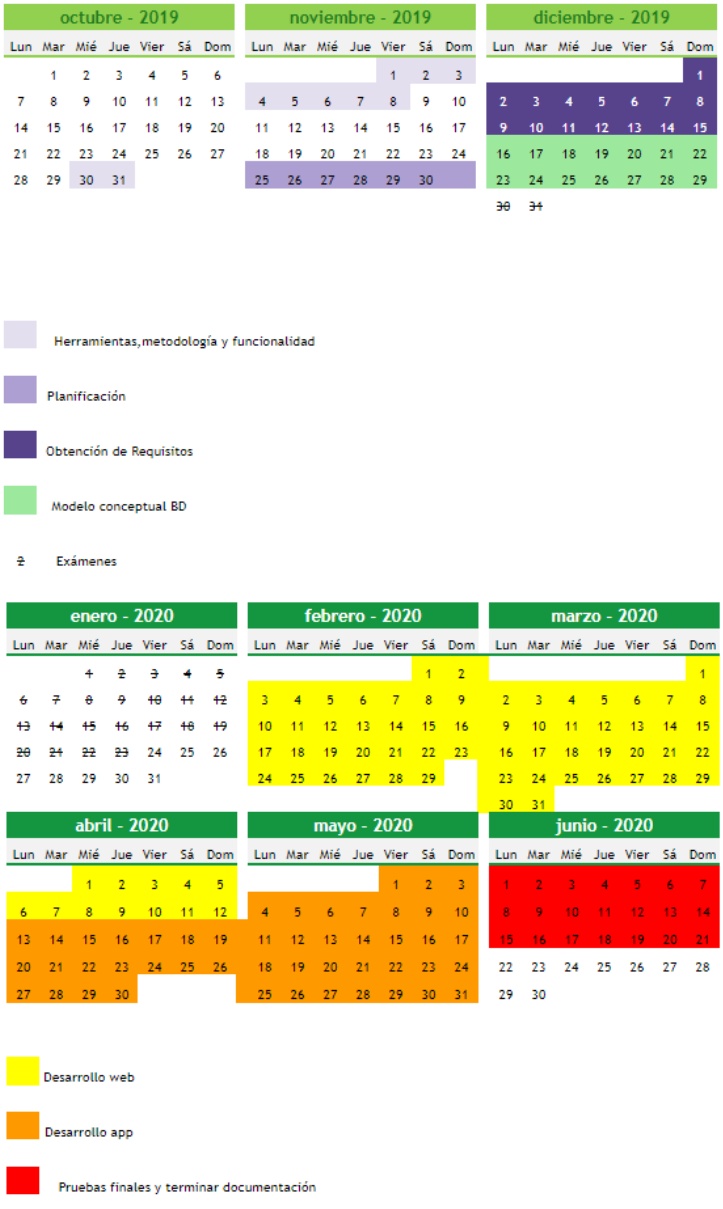
(Valor inicial- Valor residual)/ Tiempo vida útil = Depreciación anual
(550€- 50€)/5 años = **100 €/año**

Si aproximamos el uso del equipo a un año, obtenemos que el gasto en el ordenador equivale a 100€. por último debemos añadir le coste del *hosting*/servidor donde tendremos alojada nuestra web y nuestra base datos. Para ello podemos usar el servicio de Sered que nos prove de un servidor con SSD con unas grandes características por unos 75€/año, incluyendo un dominio. Teniendo esto en cuenta el coste de equipo sería de 100€más 75€anuales de mantenimiento.

Por tanto, el presupuesto final del proyecto es de **5.589 €** más **75€** cada año.

Cabe destacar que para el proyecto real yo ya dispongo de un ordendor (con mejores especificaciones que las indicadas en el apartado anterior) y que utilizo mi equipo para trabajar en local y usando los servicos de contenedores Docker que me aportaba la UGR, por lo que a efectos prácticos los gastos en equipo y mantenimiento son nulos.

Figura 3.2: Calendario de la planificación inicial del proyecto



Capítulo 4

Fase de Análisis

En esta sección, y primera fase relacionada con el proyecto como tal, nos centramos en analizar las necesidades del sistema para los potenciales usuarios. Es decir, aquí vamos a **especificar los requisitos** tanto funcionales como no funcionales y de información que consideremos necesarios para la correcta implementación del sistema.

4.1. Requisitos funcionales

RF 1. Gestión Usuario individual

- RF 1.1 Registro usuario individual: El usuario individual se da de alta en el sistema.
- RF 1.2 Dar de baja usuario individual: El usuario individual se da de baja en el sistema.
- RF 1.3 Modificar datos usuario individual: EL usuario individual modificará sus datos de perfil.
- RF 1.4 Visualizar lista usuarios individuales: Al administrador se le permite visualizar la lista de usuarios individuales.
- RF 1.5 Visualizar datos usuario individual: Al administrador se le permite visualizar los datos correspondientes a un usuario individual concreto.

RF 2. Gestión Usuario entidad

- RF 2.1 Registro usuario entidad: El usuario entidad se da de alta en el sistema.
- RF 2.2 Dar de baja usuario entidad: El usuario entidad se da de baja en el sistema.

RF 2.3 Modificar datos usuario entidad: El usuario entidad modificará sus datos de perfil.

RF 2.4 Visualizar lista usuarios entidad: Al administrador se le permite visualizar la lista de usuarios entidad.

RF 2.5 Visualizar datos usuario entidad: Al administrador se le permite visualizar los datos correspondientes a un usuario entidad concreto.

RF 3. Gestión administradores

RF 3.1 Dar de alta administrador: Solo un administrador puede dar de alta a otro administrador. Desde el principio habrá dado de alta un número de administradores.

RF 3.2 Visualizar lista administradores: Al administrador se le permite visualizar la lista de administradores.

RF 3.3 Dar de baja: Un administrador puede dar de baja a otro (tiene que haber al menos un administrador en todo momento).

RF 4. Gestión eventos

RF 4.1 Dar de alta evento: Ambos tipos de usuarios pueden dar de alta eventos.

RF 4.2 Eliminar evento: Un administrador y el usuario que haya creado dicho evento podrán eliminar el evento.

RF 4.3 Editar evento: Un administrador y el usuario que haya creado dicho evento podrán modificar el evento.

RF 5. Gestión Competición

RF 5.1 Creación competición: Ambos tipos de usuarios podrán crear un evento de tipo liga o torneo.

RF 5.2 Eliminar competición: un administrador y el creador de la competición podrán eliminarlo.

RF 5.3 Visualización competición: Todo usuario inscrito en la competición podrá ver la liga o el torneo.

RF 6. Unirse a un evento: Un usuario individual podrá unirse a un evento ya creado.

RF 7. Unirse a un torneo: Un usuario individual podrá unirse a una competición ya creada. Podrá unirse como usuario o como equipo junto a otro/s usuario/s.

RF 8. Gestión grupo de conversación

RF 8.1 Creación grupo de conversación: Al crearse un evento se establece un grupo de conversación entre los integrantes de dicho evento.

RF 8.2 Eliminación grupo de conversación: Al terminar un evento se destruye el grupo de conversación.

RF 8.3 Envío de mensajes: Los integrantes del grupo podrán enviar mensajes al grupo de conversación y leer los de los demás.

RF 9. Listado de eventos: Los 3 tipos de usuarios podrán listar los eventos disponibles.

RF 10. Definir interés usuario: Un usuario individual podrá seleccionar las temáticas que más le interesan.

RF 11. Listado de mis eventos: Los 2 tipos de usuarios podrán listar los eventos de los que forman parte.

RF 12. Filtrado eventos: Los 3 tipos de usuarios podrán buscar los eventos por nombre.

RF 13. Localizar evento: El usuario podrá abrir la dirección del evento en Google Maps.

RF 14. Funcionalidades extra aplicación móvil: La aplicación móvil dispone de herramientas útiles para el desarrollo del evento como un cronómetro y un marcador.

RF 15. Gestión valoraciones:

RF 15.1 Usuario valora usuario: Un usuario podrá valorar a otro con el que haya compartido evento después de que este finalice con valores entre 1 y 10.

RF 15.2 Ver valoracoin: En los litados de los eventos se podrá ver la valoracion de cada usuario.

RF 16. Solicitud alquiler de pista: Cada evento dispondrá de una sección para elegir el usuario entidad donde quieran reservar la pista (en caso de ser necesario) de entre las entidades que ofrecen pistas de ese tipo de actividad.

RF 17. Pistas alquilables: Cada usuario entidad dispondrá de un tablón donde mostrará las pistas de las que dispone divididas por horas para mostrar su disponibilidad.

RF 18. Equipamiento alquilable: Cada usuario entidad registrará el equipamiento que tiene disponible para alquilar.

RF 19. Tablón de anuncios: A los usuarios le aparecerán en su página principal los eventos relacionados con sus temáticas en las que se interesa. A los usuarios no identificados se le mostrarán todos los eventos.

RF 20. Foros temáticos: Cada temática dispondrá de un foro. Los usuarios registrados podrán acceder al foro para ver otros mensajes y comentar.

RF 21. Alquiler pista: El usuario podrá seleccionar una hora de la pista que se encuentre disponible.

RF 22. Calendario: Cada usuario dispondrá de un calendario en el que aparecerán sus eventos.

RF 23. Log-in: Con esta función cualquier tipo de usuario inicia sesión en la plataforma.

RF 24. Solicitud alquiler equipamiento: Cada evento dispondrá de una sección para alquilar equipamiento, si el usuario entidad seleccionado tiene esa opción.

RF 25. Gestión equipos:

- RF 25.1 Creación equipo: Un usuario individual podrá crear un equipo.
- RF 25.2 Eliminación equipo: El creador y un administrador del equipo podrán eliminarlo.
- RF 25.3 Invitación equipo: EL creador podrá invitar a otro usuario a formar parte.
- RF 25.4 Recepción invitación a equipo: Un usuario que haya recibido una invitación a un equipo podrá aceptarla o rechazarla.
- RF 25.5 Salir de equipo: Un integrante podrá dejar el equipo.

RF 26. Gestión amigos:

- RF 26.1 Invitar amigo: Podrás mandar solicitud de amistad a otro usuario.
- RF 26.2 Aceptar/rechazar solicitud: Podrás aceptar/rechazar una solicitud de amistad.
- RF 26.3 Búsqueda usuarios: Podrás buscar usuarios por nombre de usuario para enviar solicitud de amistad.
- RF 26.4 Listado amigos: Podrás ver un listado de los usuarios que tienes como amigos.

RF 27. Gestión enfrentamientos:

RF 27.1 Crear enfrentamiento: El creador de un evento podrá crear enfrentamientos entre los participantes.

RF 27.2 Modificar enfrentamiento: EL creador de un evento podrá modificar un enfrentamiento (añadir resultado).

RF 27.3 Listado enfrentamientos: Podrás ver un listado de los enfrentamientos de un evento.

4.2. Requisitos no funcionales

RNF 1. Tipos de filtros: Se debe poder filtrar por diferentes deportes (Futbol 11, Futbol 7, Futbol Sala, Baloncesto, Volleyball, Tenis individual, Tenis por parejas, Padel individual, Padel por parejas, Running, Golf, Montañismo, Balonmano, Ciclismo, Natación, Otro deporte).

RNF 2. Tipos de ordenación: Se debe poder ordenar el listado de eventos por los diferentes atributos (Ordenar por fecha- más reciente primero/más lejano primero; Ordenar por valoración del creador- Mejor calificación primero)

RNF 3. Tipos de eventos: Los eventos pueden ser de un solo día o estar repartidos en varios (competiciones).

RNF 4. Dos usuarios no pueden tener el mismo nombre de usuario.

RNF 5. La clave de los usuarios se guardará en la base de datos tras pasar por un cifrado.

4.3. Requisitos de información

RI 1. Datos del usuario individual:

a) Nombre de usuario
b) Contraseña
c) Nombre
d) Apellidos
e) Descripción
f) Fotografía de perfil
g) Teléfono
h) Correo electrónico
i) Fecha de nacimiento

RI 2. Datos del usuario entidad:

a) Nombre de usuario
b) Contraseña
c) Nombre de entidad
d) Pistas disponibles
e) Descripción
f) Fotografía de perfil
g) Teléfono
h) Correo electrónico
i) Dirección
j) Materiales/equipo disponibles

RI 3. Datos del administrador:

a) Nombre de usuario
b) Contraseña
c) Teléfono
d) Correo electrónico

RI 4. Datos de evento-competición:

a) Identificador
b) Título
c) Fecha inicio
d) Fecha final
e) Descripción
f) Deporte
g) Número mínimo de participantes
h) Número máximo de participantes
i) Ubicacion
j) Estado
k) Número actual de participantes
l) Grupo de conversación
m) Usuario fundador
n) Tipo Eento
ñ) Tipo Participantes

RI 5. Datos de grupo de conversación:

a) Identificador

RI 6. Información mensaje de grupo de conversación:

a) Identificador
b) Identificador grupo de mensaje
c) Usuario emisor
d) Contenido
e) Fecha-hora

RI 7. Información mensaje foro:

a) Identificador
b) Identificador foro
c) Usuario emisor
d) Contenido
e) Fecha-hora

RI 8. Información equipo:

a) Identificador
b) Nombre
c) Logotipo
d) Descripción
e) Creador
f) Componentes

RI 9. Datos de foro temático:

a) Identificador de foro
b) Temática de foro

RI 10. Datos de pista:

a) Identificador de pista
b) Temática de pista
c) Ubicacion
d) Entidad

RI 11. Datos de equipamiento:

a) Identificador de equipamiento
b) Temática de equipamiento

c) Tipo

d) Nombre

e) Entidad

RI 12. Datos de enfrentamiento:

a) Identificador de primer participante

b) Identificador de segundo participante

c) Identificador de evento

d) Fecha

e) Puntos participante 1

f) Puntos participante 2

Cabe destacar que habrá enfrentamientos diferenciados entre equipos o usuarios individuales.

4.4. Diagramas de Casos de Uso

Un caso de uso no es más que la descripcion de una actividad o acción dentro del sistema. Los diagramas de casos de uso tratan de reflejar gráficamente estas acciones. Estos diagrmas tienen diferentes componentes: actores (rol del usuario en el sistema), casos de uso (operación específica dentro del sistema) y relaciones (que interacciones y conexiones tienen los actores entre ellosy con los casos de uso)[4].

Aquí se muestran los diagramas de casos de uso de las principales actividades que tienen lugar en el sistema:

Figura 4.1: Diagrama Casos de Uso de Gestión de Amigos

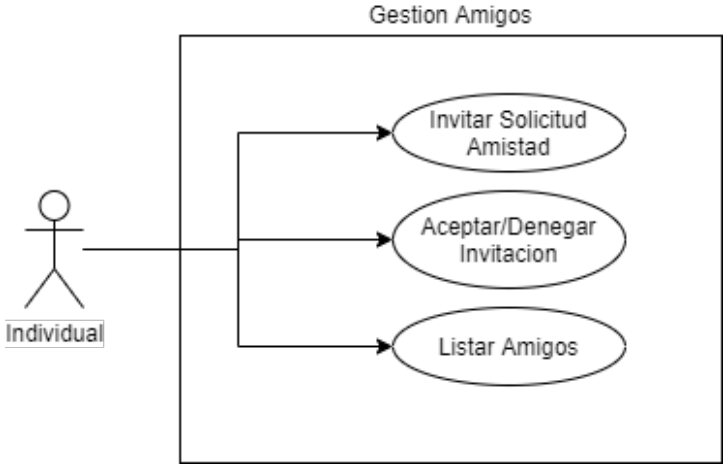


Figura 4.2: Diagrama Casos de Uso de Gestión de Calendarios

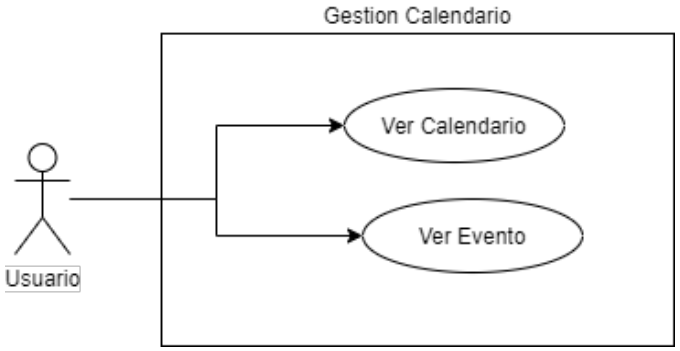


Figura 4.3: Diagrama Casos de Uso de Gestión de Equipos

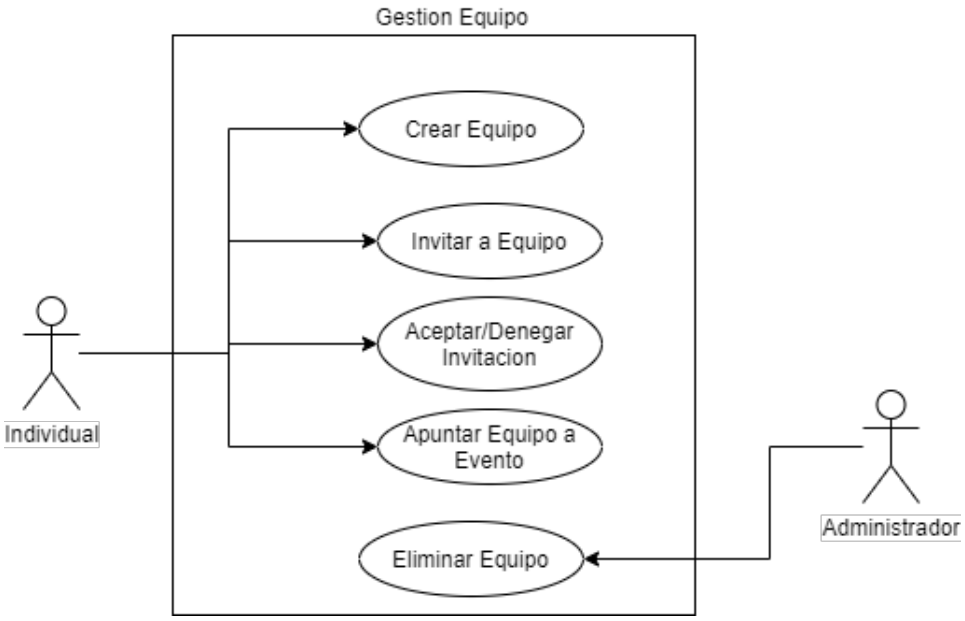


Figura 4.4: Diagrama Casos de Uso de Gestión de Eventos

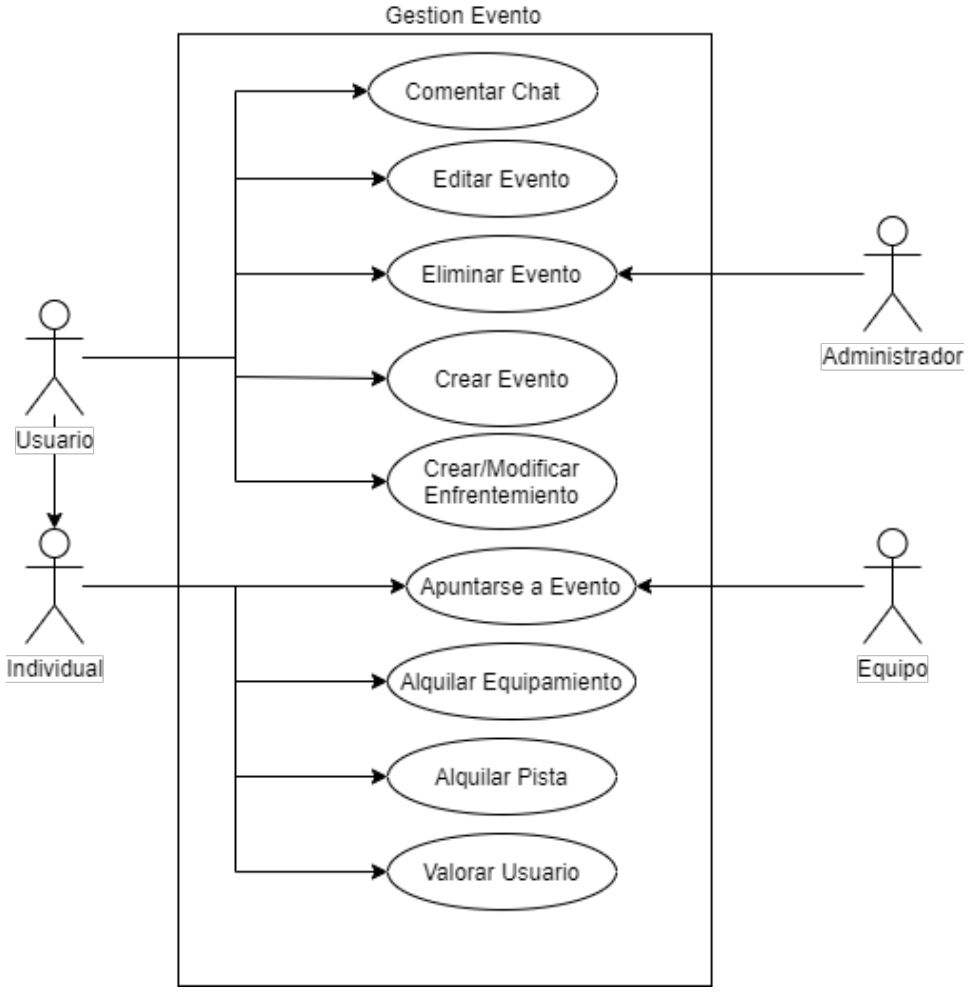


Figura 4.5: Diagrama Casos de Uso de Gestión de Foros

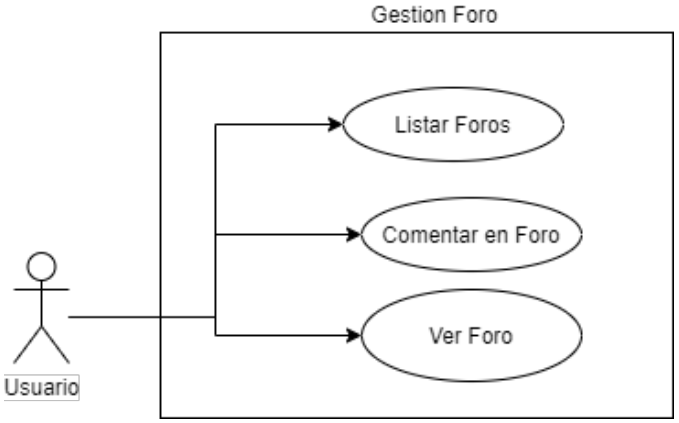


Figura 4.6: Diagrama Casos de Uso de Gestión de Pistas y Equipamientos

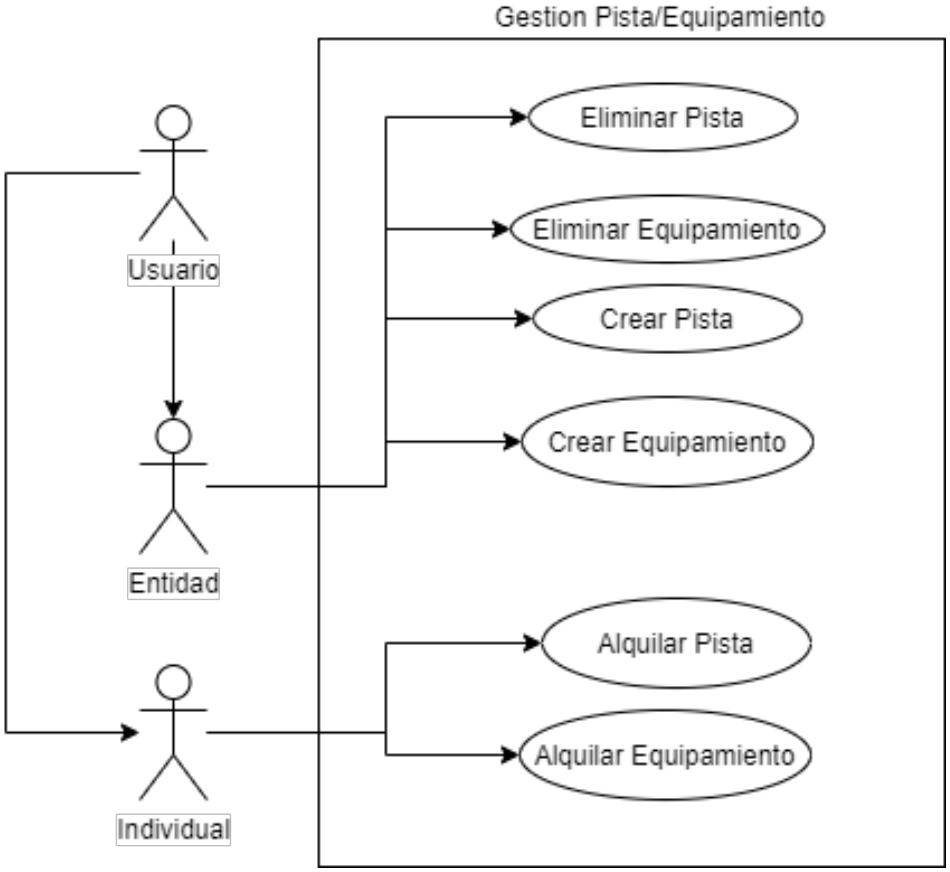


Figura 4.7: Diagrama Casos de Uso de Gestión de Tematicas

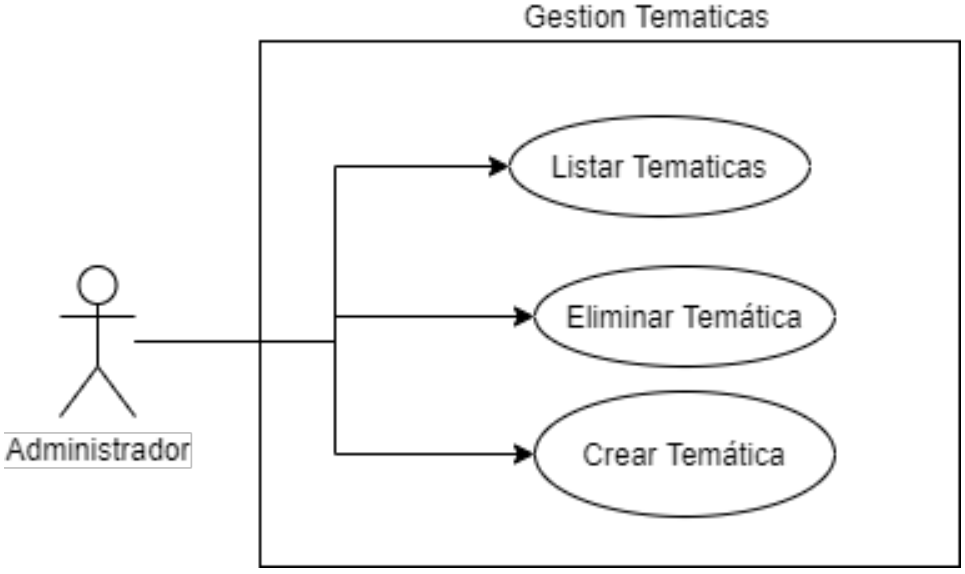
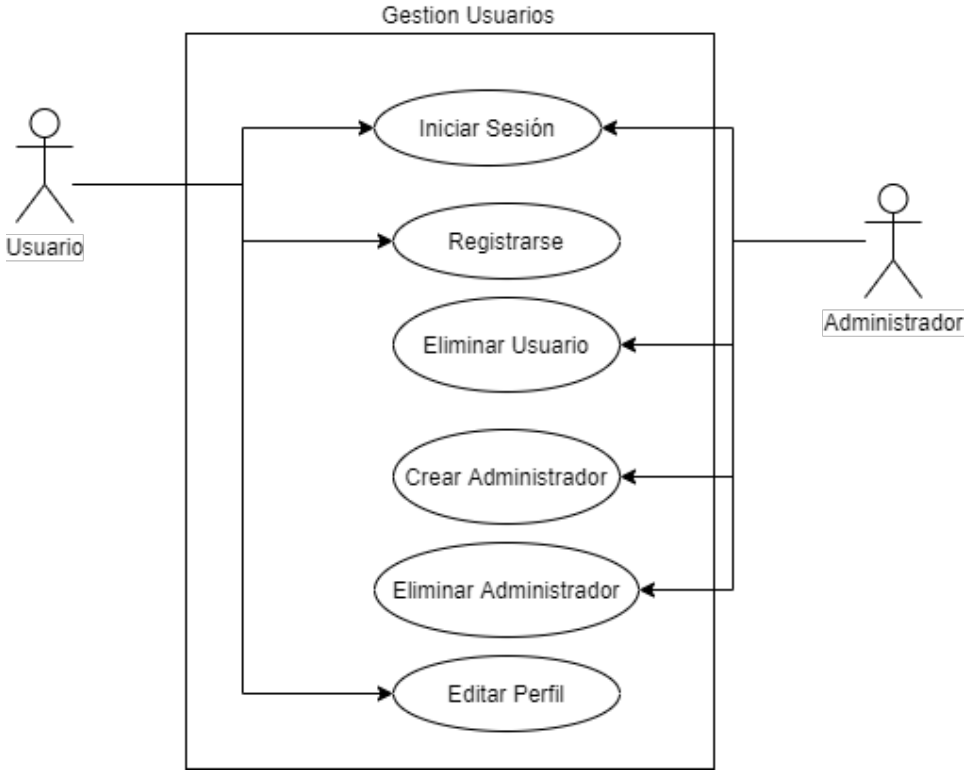


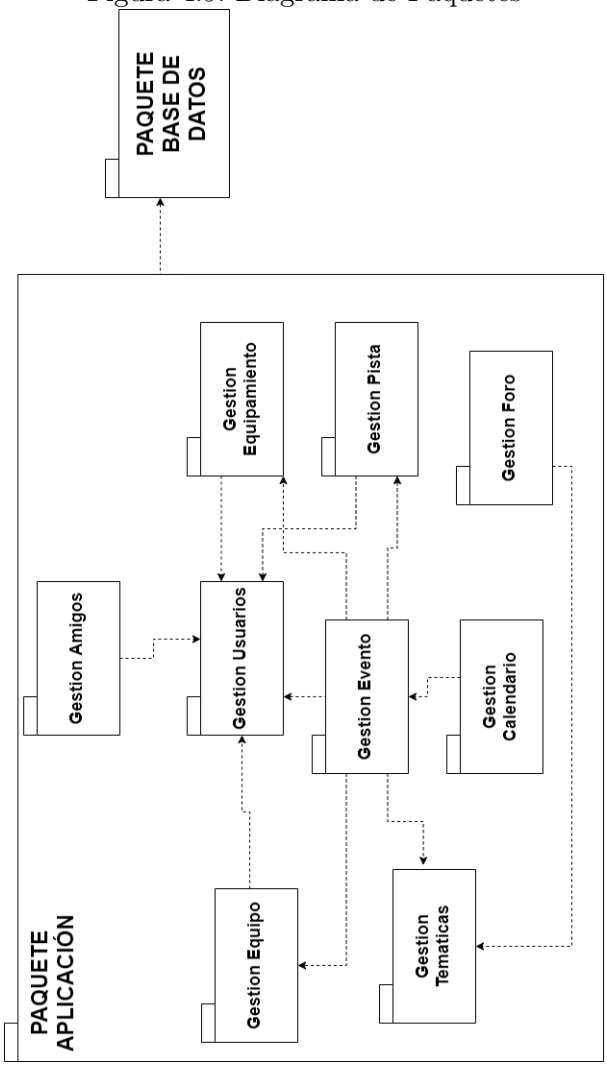
Figura 4.8: Diagrama Casos de Uso de Gestión de Usuarios



4.5. Diagrama de Paquetes

Este tipo de diagramas muestan las dependencias entre los paquetes que conforman el sistema diseñado. En este caso se muestran las dependencias generales entre los diferentes paquetes que hemos obtenido al realizar los diagramas de casos de uso, lo que nos permite entender que estos paquetes aún siendo compactos, necesitan de ciertas características o elementos de algunos otros paquetes. Y por supuesto todos estos paquetes, englobados en uno genérico que he llamado 'Paquete Aplicación' necesitan a la base de datos para obtener los datos del sistema. Aquí podemos ver dicho diagrama:

Figura 4.9: Diagrama de Paquetes



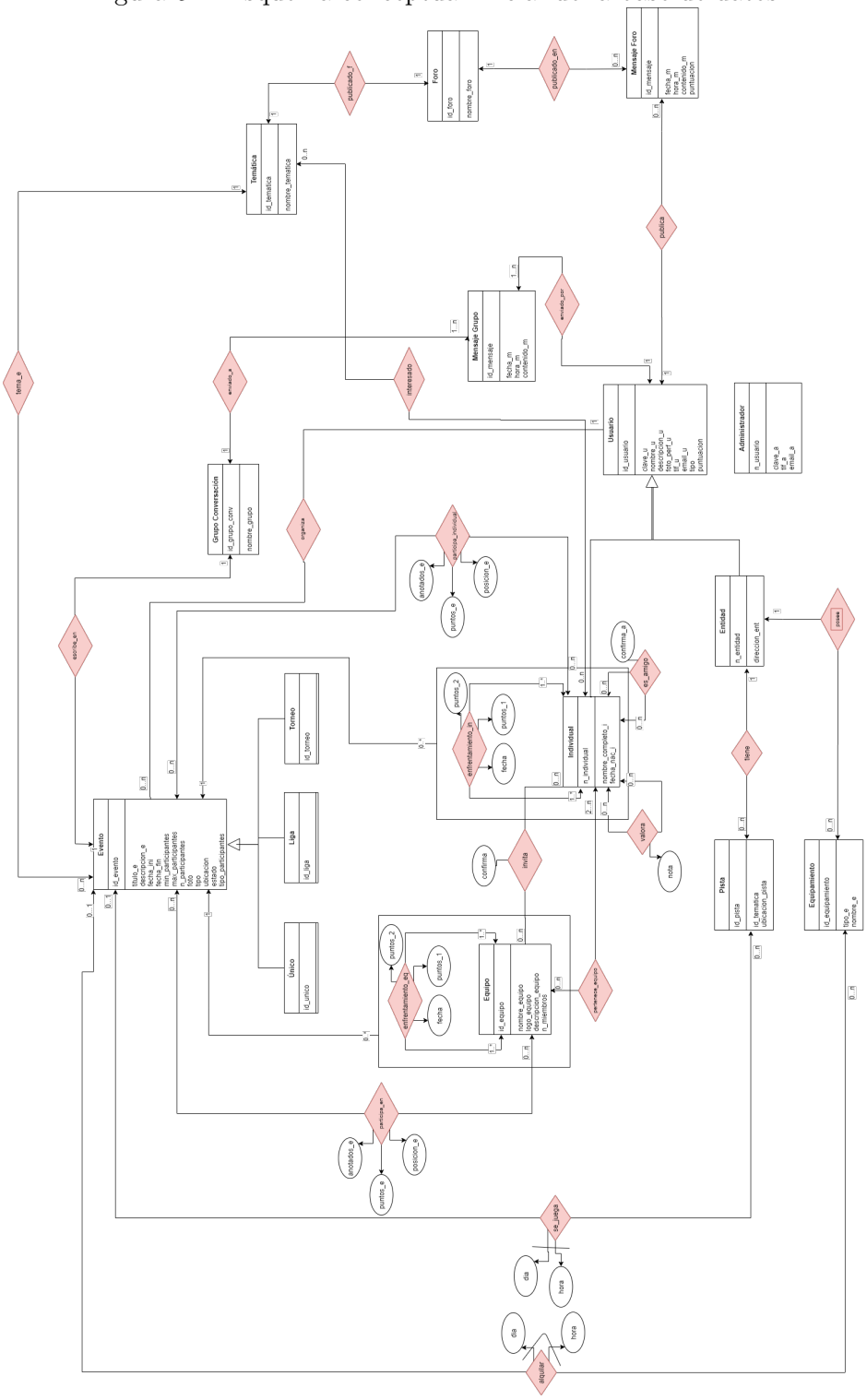
Capítulo 5

Fase de Diseño

5.1. Modelo Base de Datos

Una vez analizados los requisitos del sistema, y teniendo claros sobretodo los requisitos de información, se procede al diseño de la base de datos del sistema. Debido a lo estudiado durante el grado y mi experiencia, así como las características del proyecto, decidí utilizar una base de datos relacional utilizando SQL. Esta se compone de tablas (que las veremos a continuación), y relaciones entre ellas, que facilitan la obtención de información al hacer consultas y utilizar la información necesaria para cada momento de la utilización del sistema. Este tipo de bases de datos nos permiten garantizar la no duplicidad de registros en una misma tabla, así como una integridad referencial. Lo primero es llevar a cabo el esquema conceptual. Este esquema nos mostrará cada una de las tablas, con sus diferentes columnas cada una y las relaciones entre estas tablas. La figura 5.1 muestra el esquema inicial de la base de datos.

Figura 5.1: Esquema conceptual inicial de la base de datos



Lo primero es crear cada una de las entidades principales, como el usuario, evento,... Y luego pasar a relacionarlas entre ellas. De estas relaciones obtendremos nuevas tablas.

Una vez completado el diseño, lo siguiente es proceder al conocido como paso a tablas. Se trata de, a partir de este esquema, obtener las tablas correspondientes a las entidades principales y las relaciones entre ellas. Estas son las tablas que obtenemos en primera instancia.

Entidades:

1. Evento(id_evento,titulo_e,descripcion_e,fecha_ini_fecha_fin,min_participantes, max_participantes,n_participantes,foto,tipo,ubicacion,estado)
2. Unico(id_unico)
3. Liga(id_liga)
4. Torneo(id_torneo)
5. Grupo_Conversacion(id_grupo_conv,nombre_grupo)
6. Equipo(id_equipo,nombre_equipo,logo_equipo,descripcion_equipo,n_miembros)
7. Administrador(n_usuario,clave_a,tlf_a,email_a)
8. Usuario(id_usuario,clave_u,nombre_u,descripcion_u,foto_perf_u,tlf_u,email_u, tipo,puntuacion)
9. Individual(id_individual,nombre_completo_i,fecha_nac_i)
10. Entidad(id_entidad,direccion_ent)
11. Pista(id_pista,deporte_pista,ubicacion_pista)
12. Equipamiento(id_equipamiento,tipo_e,nombre_e)
13. Mensaje_Grupo(id_mensaje,fecha_m,hora_m,contenido_m)
14. Tematica(id_tematica,nombre_tematica)
15. Foro(id_foro,nombre_foro)
16. Mensaje_Foro(id_mensaje,fecha_m,hora_m,contenido_m)

Relaciones:

17. alquilar(id_equipamiento,dia,hora,id_evento)
18. se_juega(id_pista,dia,hora,id_evento)
19. tiene(id_pista,id_entidad)

20. posee(id_equipamiento,id_entidad)
21. pertenece_equipo(id_equipo,id_individual)
22. invita(id_equipo,id_individual,confirma)
23. es_amigo(id_individual1,id_individual2)
24. valora(id_individual1,id_individual2,nota)
25. participa_en(id_equipo,id_evento,anotados_e,puntos_e,posicion_e)
26. participa_individual(id_individual,id_evento,anotados_e,puntos_e,posicion_e)
27. organiza(id_evento,id_usuario)
28. escribe_en(id_grupo_conv,id_evento)
29. tema_e(id_evento,id_temática)
30. enviado_a(id_mensaje,id_grupo_conv)
31. enviado_por(id_mensaje,id_usuario)
32. publica(id_mensaje,id_usuario)
33. publicado_en(id_mensaje,id_foro)
34. publicado_f(id_foro,id_tematica)
35. enfrentamiento_in(participante_1,participante_2,id_evento,fecha,puntos_1,puntos_2)
    (*)los participantes son ids de individuales
36. enfrentamiento_eq(participante_1,participante_2,id_evento,fecha,puntos_1,puntos_2)
    (*)los participantes son ids de equipos

Una vez tenemos las tablas, para reducir el número y facilitar la interacción entre la smismas procemos a la fusión de tablas, de esta manera, eliminamos algunas tablas añadiendo nuevos campos a otras. Aquí el resultado:

Fusión

1. 1-28.Evento(id_evento,titulo_e,descripcion_e,fecha_ini,fecha_fin,min_participantes, max_participantes,n_participantes,foto,tipo,ubicacion,estado,id_grupo_conv)
2. (1-28)-29.Evento(id_evento,titulo_e,descripcion_e,fecha_ini,fecha_fin,min_participantes, max_participantes,n_participantes,foto,tipo,ubicacion,estado, id_grupo_conv,id_tematica)
3. 12-17.Equipamiento(id_equipamiento,dia,hora,tipo_e,nombre_e,id_evento)

4. 11-18.Pista(id_pista,dia,hora,id_evento,deporte_pista,ubicacion_pista)S
5. ((1-28)-29)-27.Evento(id_evento,titulo_e,descripcion_e,fecha_ini,fecha_fin,min_participantes, max_participantes,n_participantes,foto,tipo,ubicacion,estado,id_grupo_conv,id_tematica, id_usuario_creador)
6. 13-30.Mensaje_Grupo(id_mensaje,fecha_m,hora_m,contenido_m,id_grupo_conv)
7. (11-18)-19.Pista(id_pista,dia,hora,id_evento,deporte_pista,ubicacion_pista,id_entidad)
8. (12-17)-20.Equipamiento(id_equipamiento,dia,hora,tipo_e,nombre_e,id_evento,id_entidad)
9. 16-31.Mensaje_Foro(id_mensaje,fecha_m,hora_m,contenido_m,id_usuario)
10. (13-30)-31.Mensaje_Grupo(id_mensaje,fecha_m,hora_m,contenido_m,id_grupo_conv,id_usuario)
11. (16-31)-33.Mensaje_Foro(id_mensaje,fecha_m,hora_m,contenido_m,id_usuario,id_foro)
12. 15-34.Foro(id_foro,nombre_foro,id_tematica)

Antes de dar por terminada esta base de datos, repensé la forma de almacenar la ubicación de los eventos. Me informé sobre la API de Google Maps para integrarla en la web y la aplicacióon, por lo que en última instancia eliminé el campo 'ubicación' y lo cambié por dos campos: 'lat' y 'lng', representando la latitud y longitud de la ubicación para representarlo en un mapa.

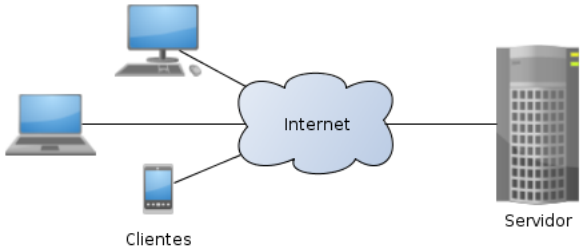
Tras el paso a tablas, lo siguiente es la implementación de la base datos, algo realmente sencillo utilizando la plataforma *phpMyAdmin*.

5.2. Diseño de Arquitectura

En este apartado detallaremos los diferentes elementos que forman parte del proyecto y como se relacionan entre ellos, de manera que quede clara la arquitectura del sistema.

A grandes rasgos, se trata de una típica arquitectura cliente-servidor [3] en la que interactuan dos partes: un solicitante de recursos y un proveedor.

Figura 5.2: Esquema Arquitectura cliente-servidor



En nuestro caso, el conjunto de clientes se representa por los navegadores web que consultan y solicitan la información y archivos de la página web, y por los dispositivos Android que realicen las peticiones a través de la aplicación móvil. Mientras, el servidor posee todos los archivos de la página web (todos los recursos necesarios para responder a las solitudes) y además aloja la base de datos de la que se obtiene la información. Pasamos a analizar cada uno de estos grupos con sus correspondientes módulos, incluyendo la aplicación Android.

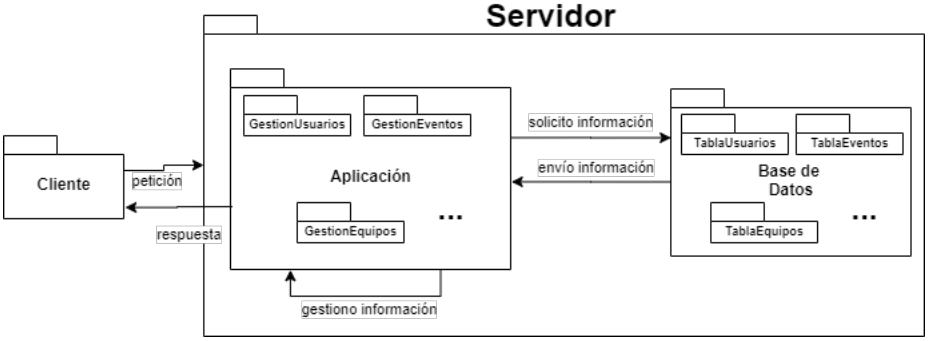
5.2.1. Módulos en servidor

El servidor se compone de dos grandes elementos: la aplicación y la base de datos. La aplicación es el software desarrollado utilizando Laravel, que es el que contiene todos los archivos relacionados con la web como son los archivos HTML o CSS para mostrar la web o los archivos PHP que permiten hacer su contenido dinámico. Por otr lado tenemos la base de datos. Estando sobre el sistema de gestión de base de datos MySQL, contiene todos los datos del sistema como usuarios, eventos, etc. junto con sus relaciones.

La relación entre estos dos módulos es lo que dará la respuesta correcta al cliente que realice determinadas solicitudes a este servidor, siendo esto totalmente opaco para dicho cliente. La aplicación solicita a la base de datos los datos necesarios según la solicitud del cliente, los gestiona de la forma

correspondiente y responde al cliente de forma satisfactoria. Es decir, es el paquete de la aplicación el que le da la funcionalidad al sistema.

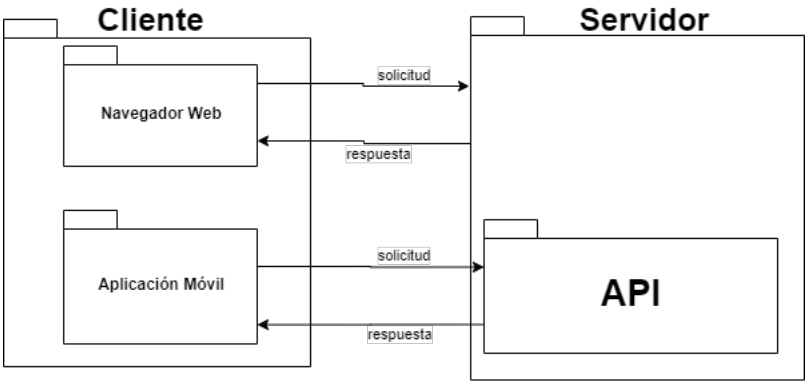
Figura 5.3: Modulo Servidor



5.2.2. Módulos en cliente

El caso del cliente es más simple, pues para solicitudes en formato de página web simplemente necesitamos un navegador. Este realiza las peticiones necesarias al servidor (GET, POST,...) y se encarga de recibir las respuestas de éste y mostrarlas en la pantalla. Sin embargo, desde la aplicación móvil utilizams una API, que actúa de forma similar a las consultas desde el navegador pero devolviendo únicamente estructuras de datos, no páginas visibles (en el siguiente punto se tratará con más detalle). Su estructura se puede representar de la forma mostrada en la Figura 5.4.

Figura 5.4: Modulo Cliente

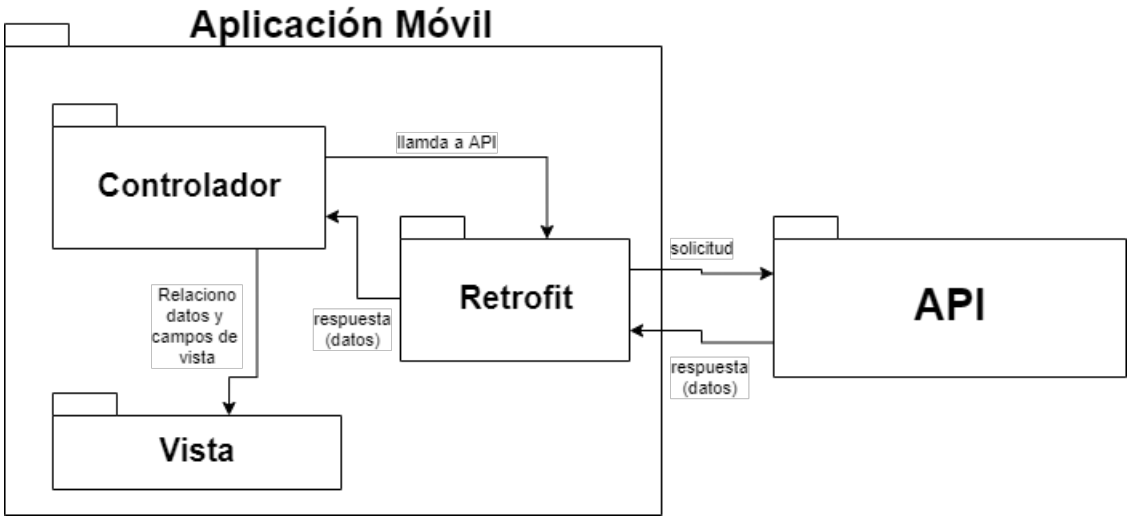


5.2.3. Arquitectura Apliación Android

Como ya he mencionado, las solicitudes por parte de la aplicación móvil al servidor se diferencian respecto a las de los navegadores en el contenido de sus respuestas. Mientras al navegador se le envía una página web, la aplicación recibe estrucutras de datos (concretamente datos de tipo JSON). Esto lo consigue utilizando la API que vamos a generar en el proyecto que utiliza Laravel, el cual se encuentra en el servidor. En este sentido, al igual que las consultas web, dentro del servidor se solicitan datos a la base de datos y se trabaja con ellos dentro. La diferencia fundamental es que una vez tenemos estos datos, quien se encarga de mostrarlos y decidir qué hacer con cada uno de ellos es la propia aplicación en su código, siendo parte fundamental en el cliente.

Esta información se consigue utilizando la librería *Retrofit* para haer solicitudes a la API y recibir la información correspondiente (esto se vé más en detalle en el capítulo de la implementación de la aplicación). El esquema de la arquitectura de la aplicación móvil, en paralelismo con el modelo cliente-servidor, se muestra en la Figura 5.5.

Figura 5.5: Arquitectura Aplicación Móvil



5.3. Diagramas de Flujo

El diagrama de flujo representa la sucesión de páginas/lugares por las que pasa el usuario según la decisión que toma el usuario; se trata de una

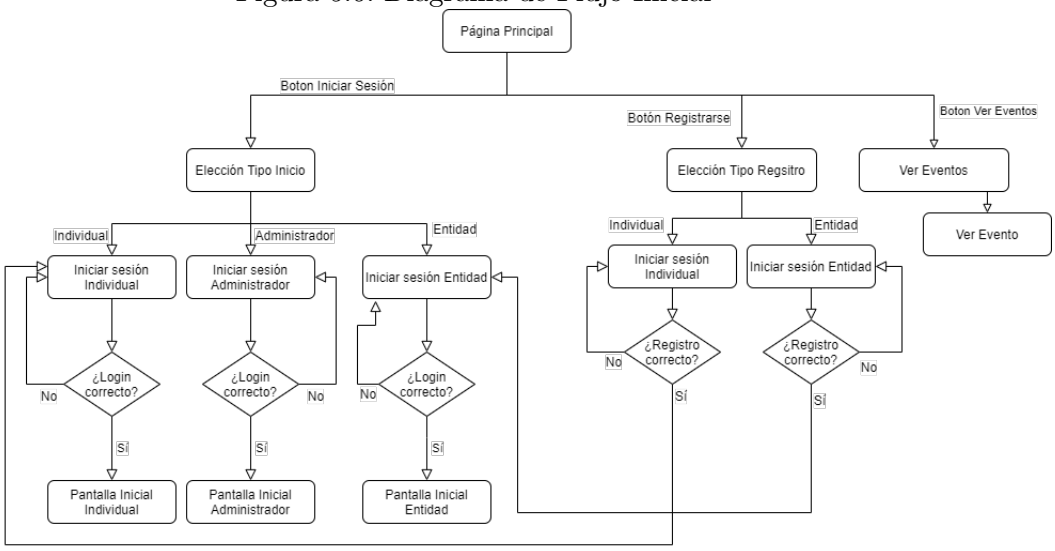
representación gráfica del proceso de utilización del sistema.

Concretamente he diseñado un diagrama de flujo para los usuarios sin registrar, y uno para cada tipo de usuario (entidad, individual y administrador). Cabe mencionar que los usuarios que ya hayan iniciado sesión tienen a su disposición un menú lateral desplegable que permite acceder directamente a las páginas principales. Comenzamos con los flujos de la web y continuaremos con el flujo en la aplicación móvil.

5.3.1. Diagramas de flujo web

Flujo de usuario no registrado:

Figura 5.6: Diagrama de Flujo Inicial



La figuroa 5.3 refleja el diagrama de flujo del usuario individual en la web, mientras que la 5.4 refleja la misma información para en usuario entidad, y la 5.5 para los administradores:

Figura 5.7: Diagrama de Flujo Usuario Individual

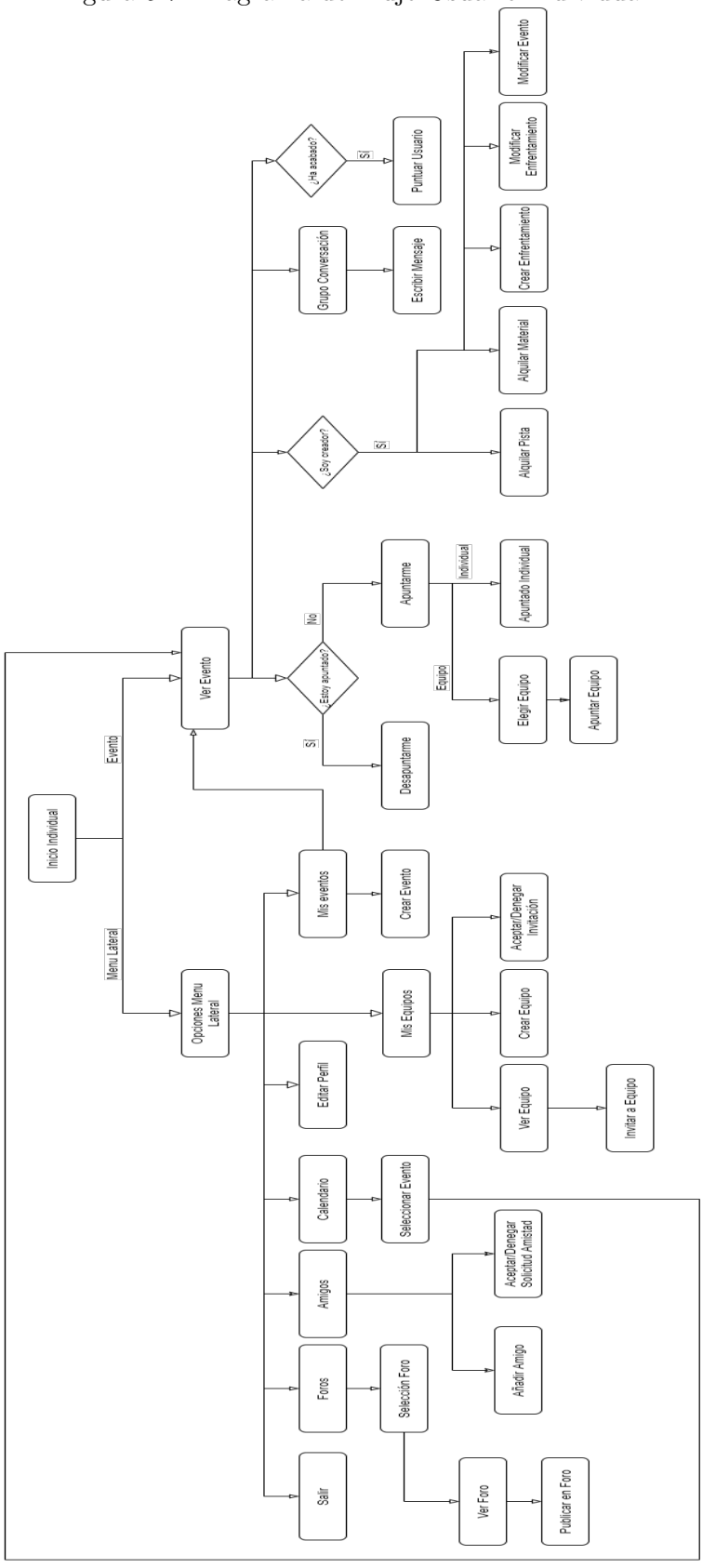


Figura 5.8: Diagrama de Flujo Usuario Entidad

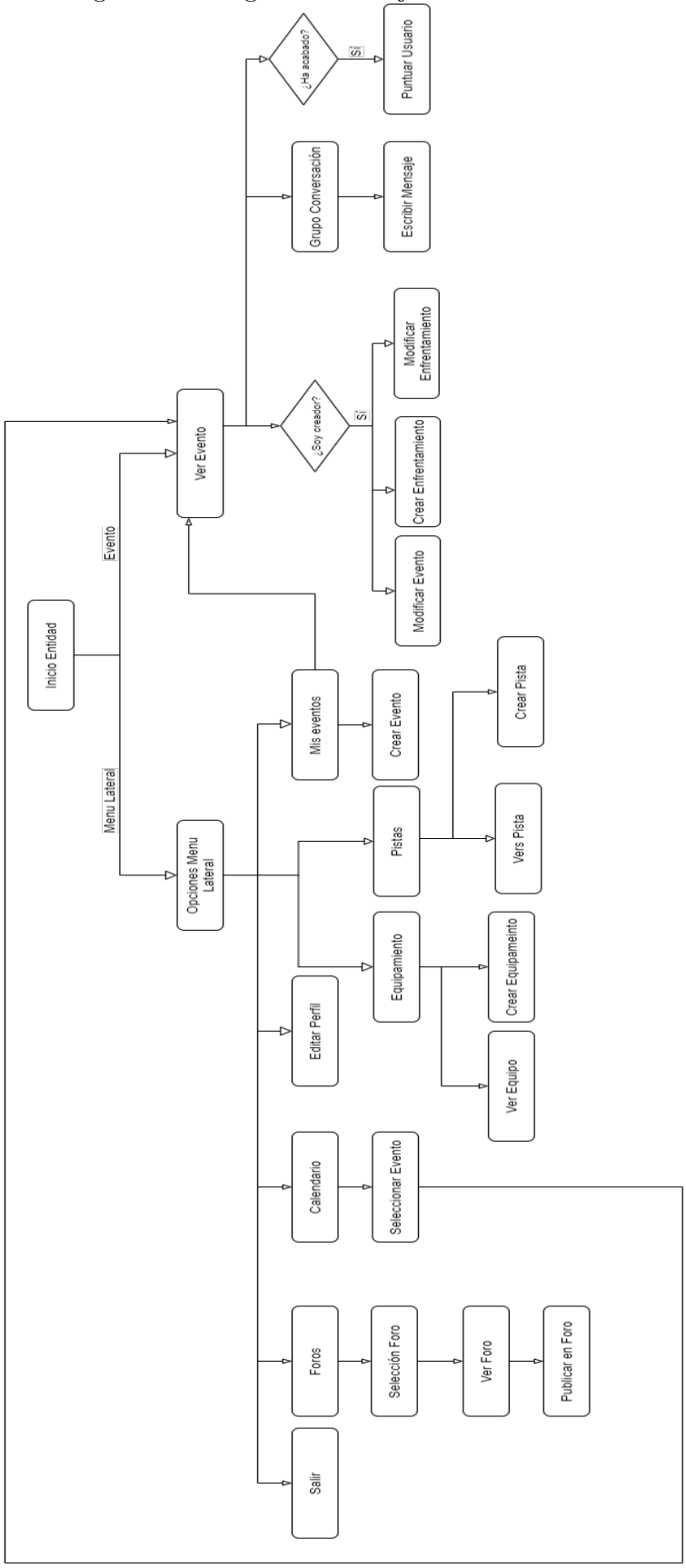
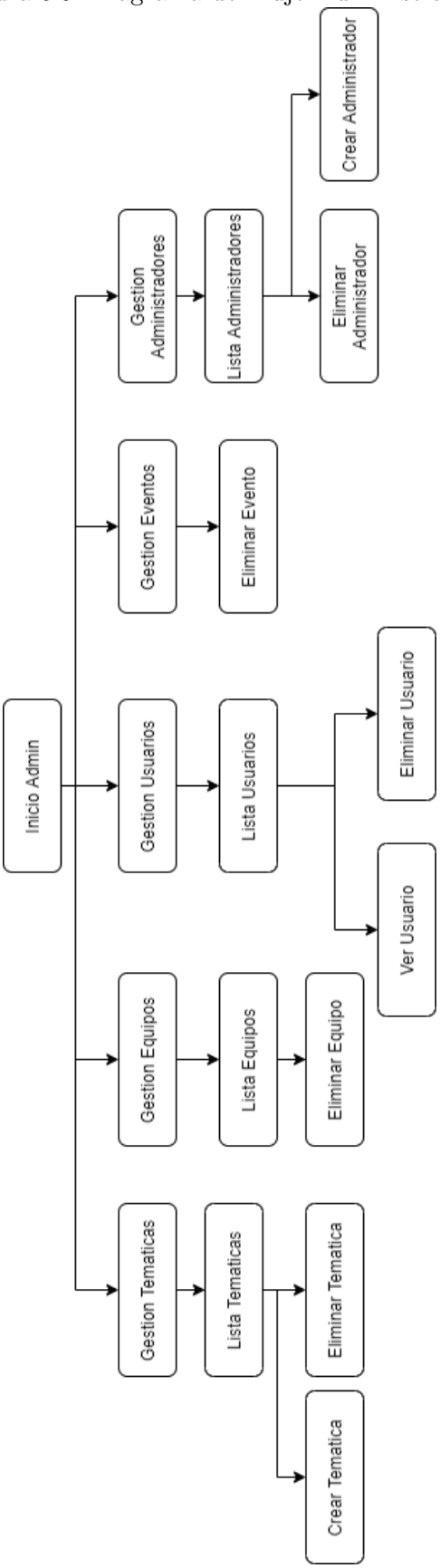


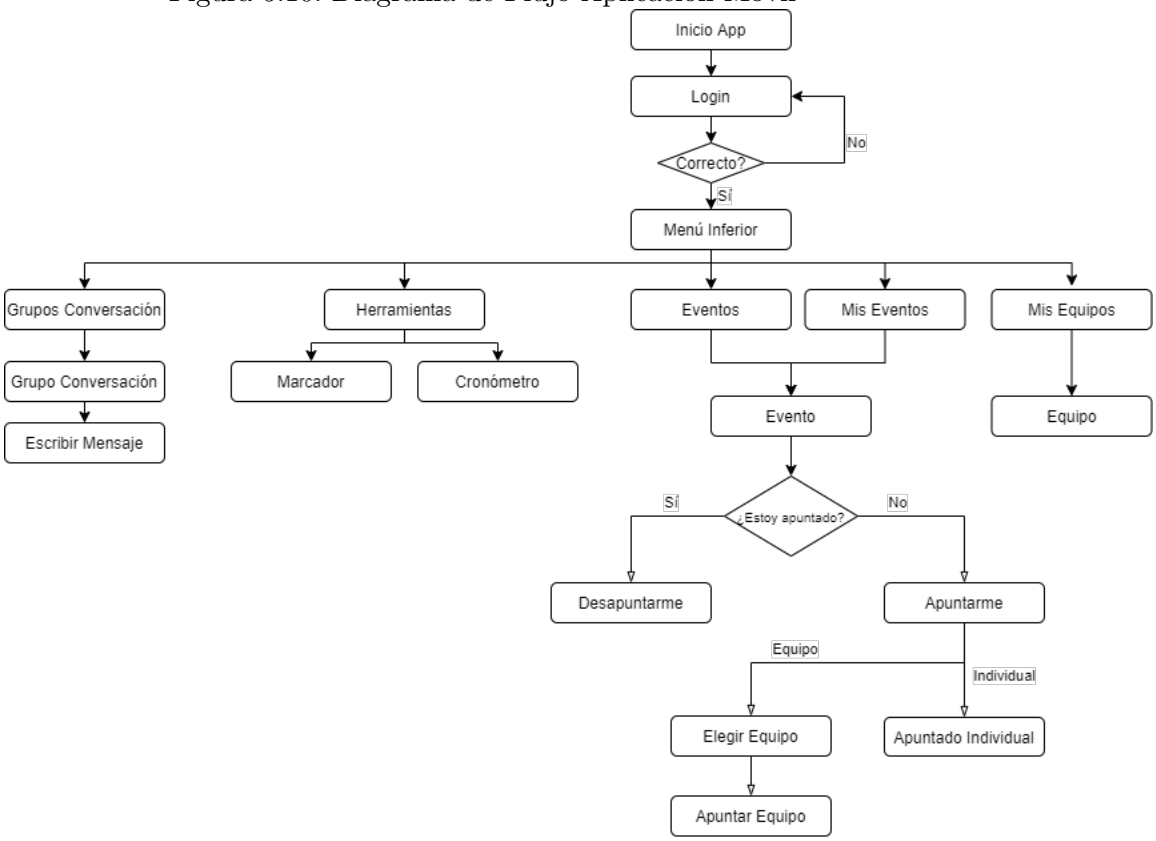
Figura 5.9: Diagrama de Flujo Administrador



5.3.2. Diagrama de flujo app

La aplicación al ser sólo utilizada por el usuario individual únicamente tenemos un diagrama de flujo para representar su funcionamiento.

Figura 5.10: Diagrama de Flujo Aplicación Móvil



5.4. Bocetos

En esta sección se incluyen imagenes de los principales bocetos. Estos diseños sirven como referencia aproximada a la representación gráfica del sistema, para utilizarlos como guía durante el desarrollo real. Más que centrarse en detalles, se basa en la estructura de la página y establecer dónde encontraremos los diferentes elementos. Esta sección se divide en los bocetos realizados para la página web y los bocetos utilizados para la aplicación móvil.

5.4.1. Bocetos web

Figura 5.11: Boceto Columna Lateral Entidad

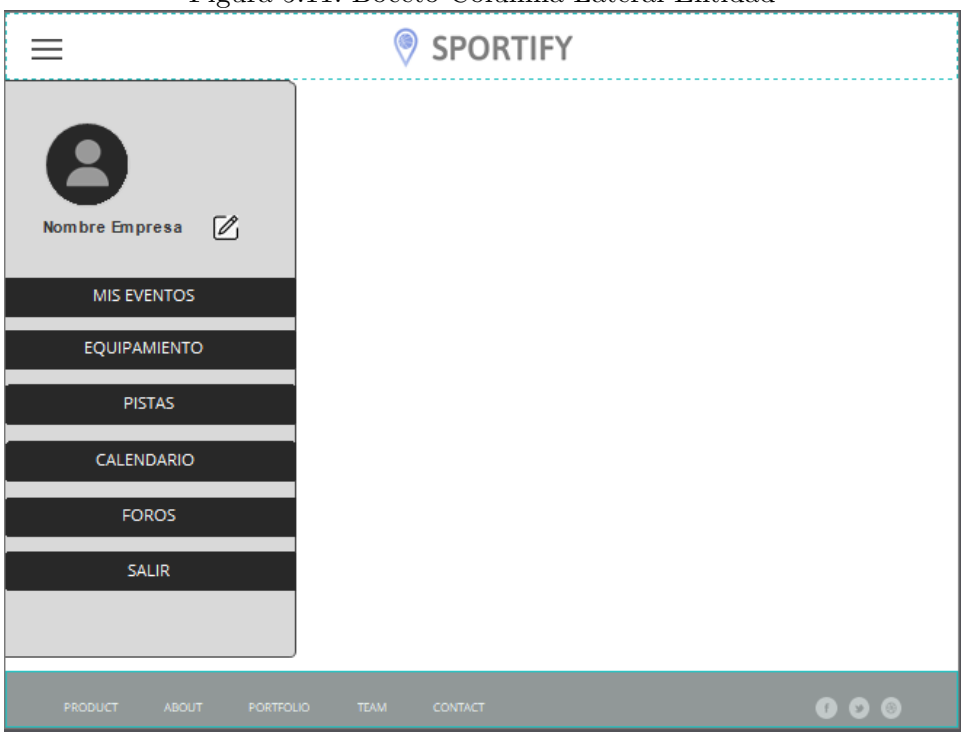


Figura 5.12: Boceto Columna Lateral Individual

Figura 5.13: Boceto Calendario

Figura 5.14: Boceto Horarios de Pistas

Figura 5.15: Boceto Perfil Entidad

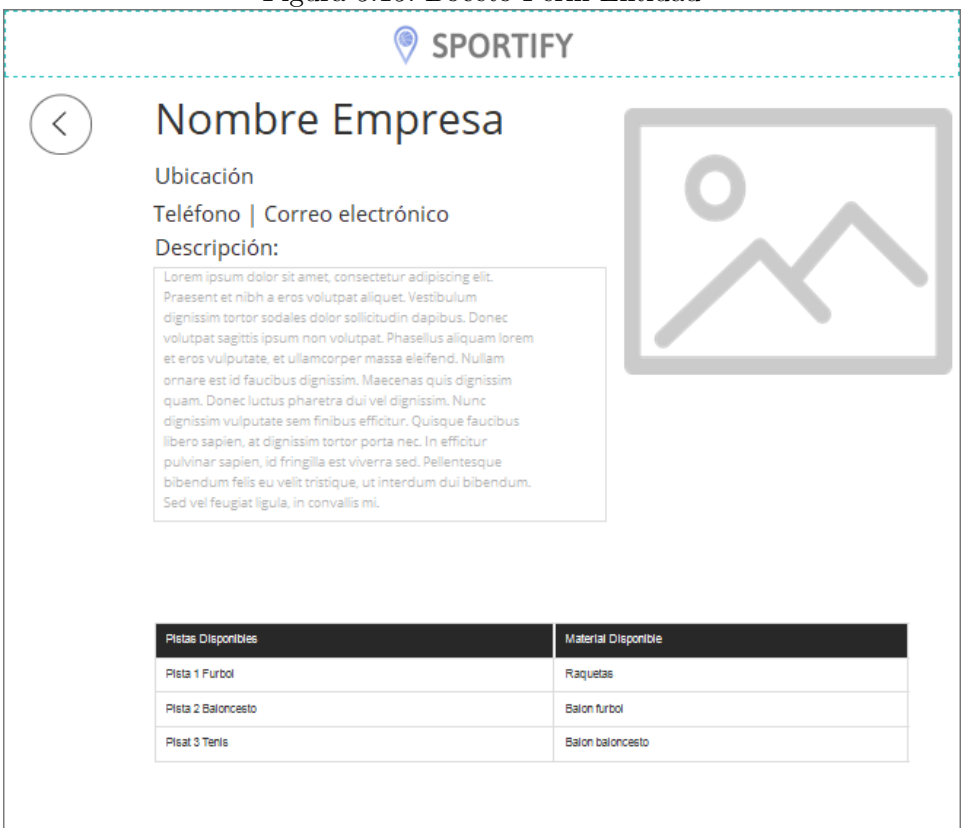


Figura 5.16: Boceto Información Evento

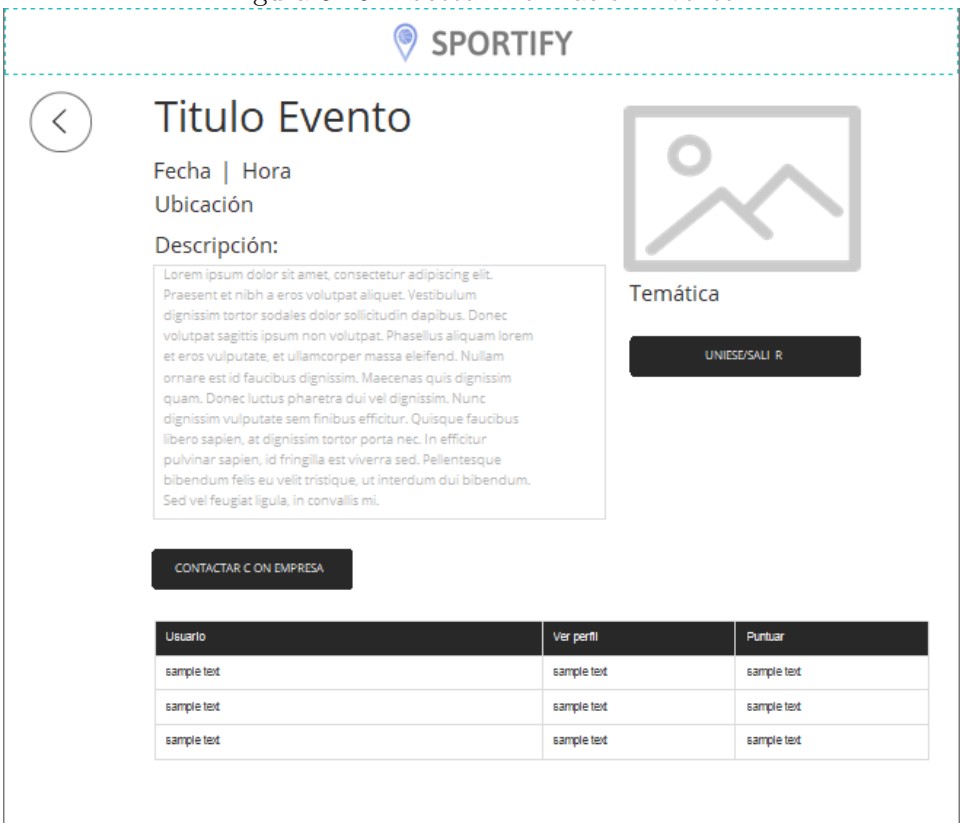


Figura 5.17: Boceto Pagina de Inicio

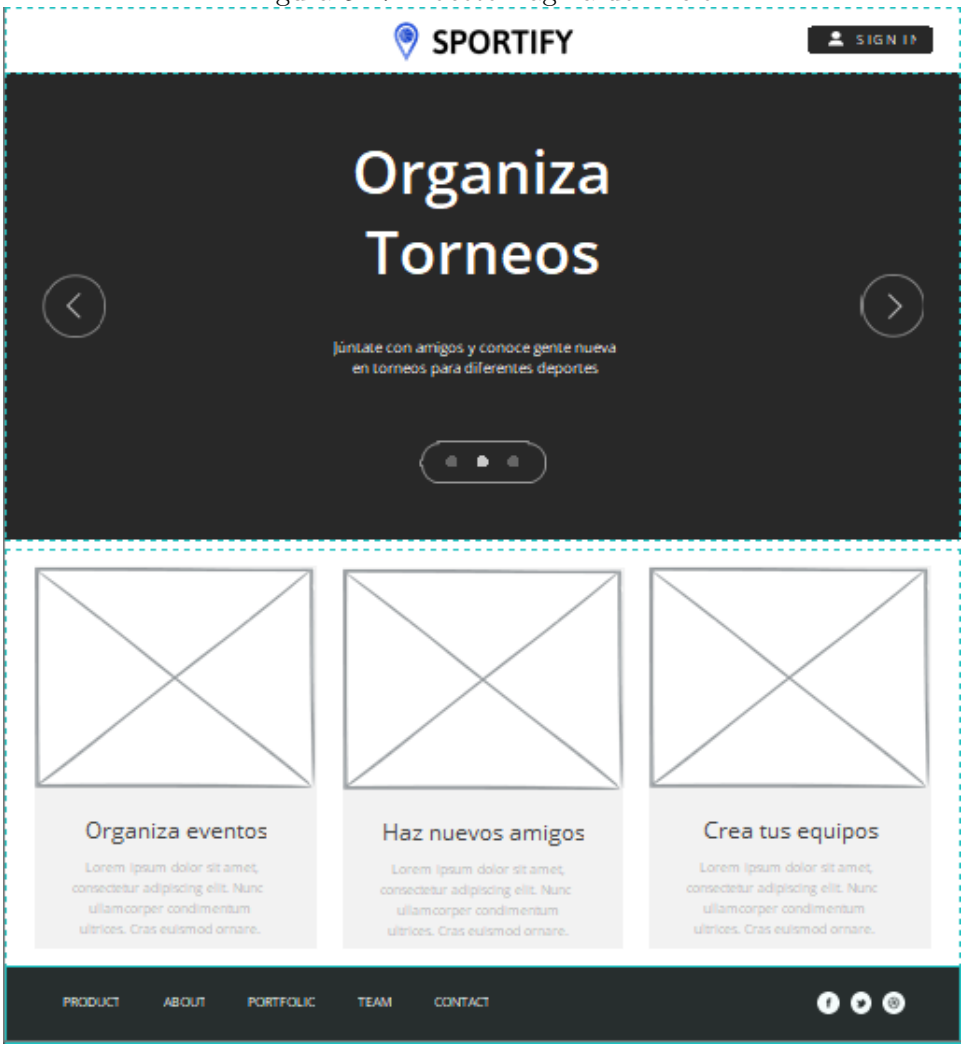


Figura 5.18: Boceto Pagina Principal Usuario

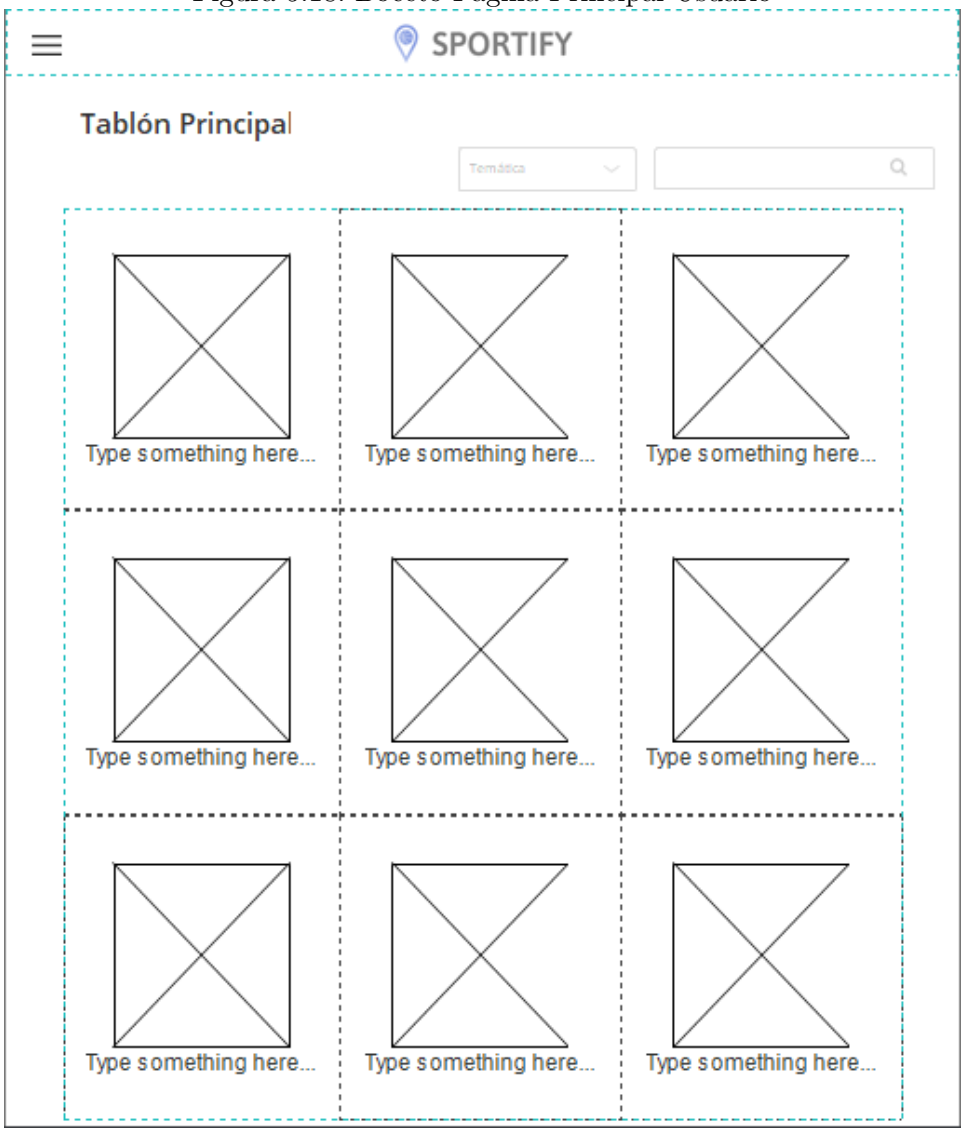


Figura 5.19: Boceto Listado de Empresas para alquilar Equipamiento

5.4.2. Bocetos App

Figura 5.20: Boceto Inicio de Sesión

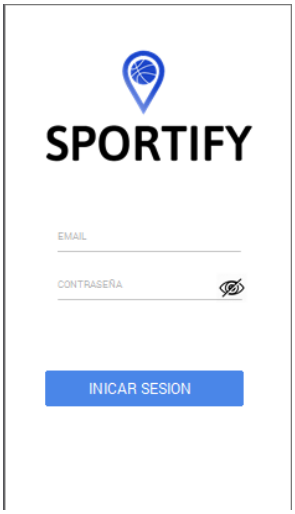


Figura 5.21: Boceto Herramientas

Figura 5.22: Boceto Listado de Eventos

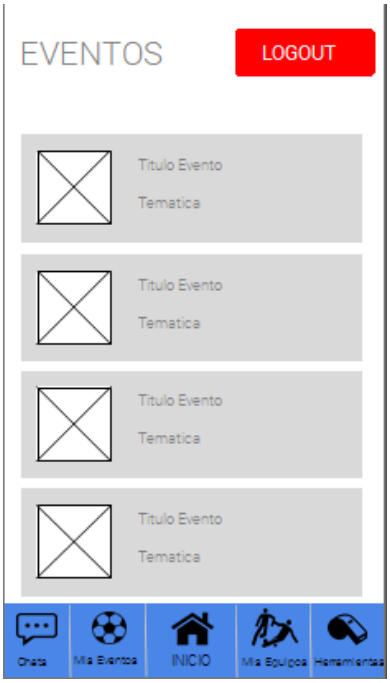


Figura 5.23: Boceto Listado de Mis Eventos

Figura 5.24: Boceto Evento Concreto

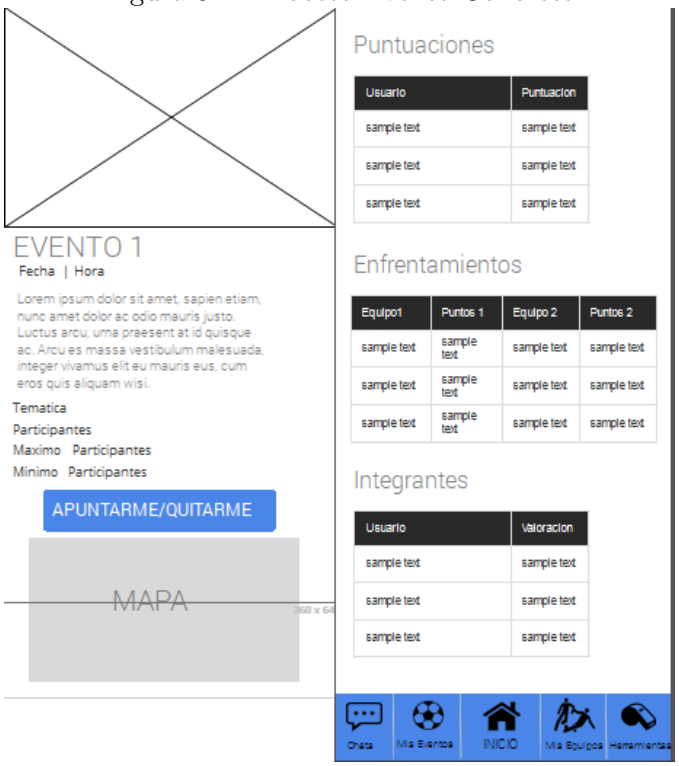


Figura 5.25: Boceto Listado de mis Grupos de Conversación

Figura 5.26: Boceto Grupo de Conversacion concreto

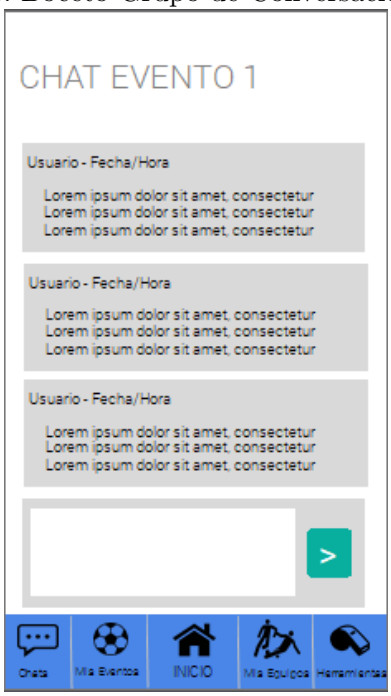


Figura 5.27: Boceto Listado de Mis Equipos

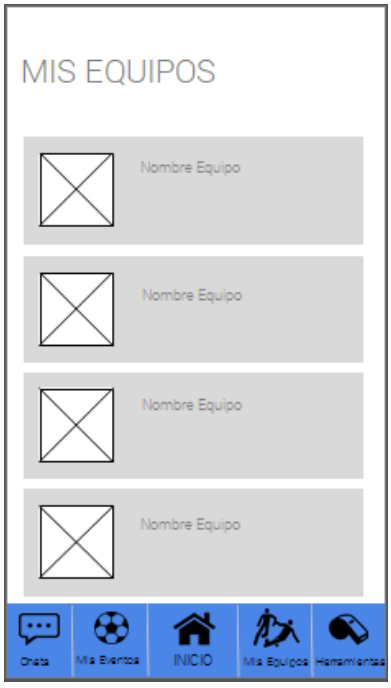
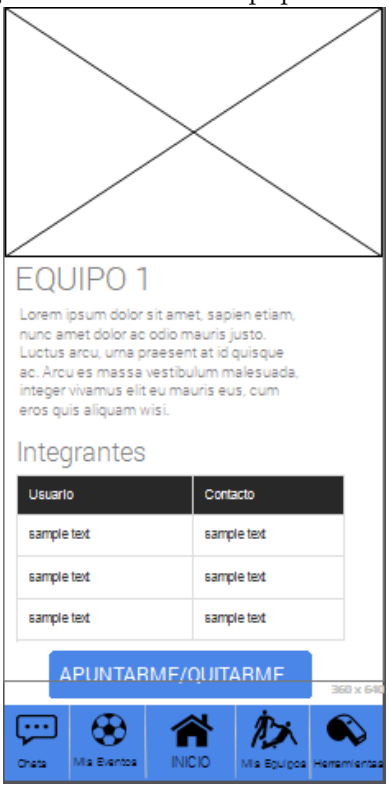


Figura 5.28: Boceto Equipo concreto



Capítulo 6

Fase de Implementación

En esta sección de la documentación me centraré en describir el desarrollo estricto de la codificación y programación del sistema. Habrá dos grandes distinciones en este apartado: uno dedicado al desarrollo de la página web y otro al de la aplicación móvil.

Para desarrollar ambos productos he utilizado un framework. Un framework trata de ofrecer una funcionalidad definida y auto-contenida; está construido utilizando patrones de diseño, y destaca por su alta cohesión y su bajo acoplamiento[5]. En concreto, he utilizado al framework **Laravel**. En el siguiente punto especificaré por qué y para qué.

Cabe destacar el uso de la biblioteca **Bootstrap** para facilitar el diseño y organización de la web, ya que su sistema de grids, entre otras herramientas, agilizan la producción y permiten usar ciertos estilos; además de ser de gran utilidad al tratar de diseñar un sitio adaptativo a diferentes tamaños según la pantalla (*responsive*).

6.1. Herramientas

Las principales herramientas utilizadas a lo largo del desarrollo son las siguientes:

- **Visual Studio Code** para la programación en HTML, CSS, PHP,... Para el desarrollo utilizaré el framework Laravel, en el cual me centraré en el apartado de implementación.
- **Android Studio** para el desarrollo de la aplicación Android.
- **Draw.io** es la web que utilizaré para el diseño de esquemas/diagramas como puede ser el modelo conceptual de la base de datos.

- **phpMyAdmin** para trabajar sobre la base de datos del sistema.
- **Google Drive** para ubicar diferentes archivos de texto, imagenes, diagramas, etc y tenerlos disponibles en cualquier dispositivo.
- **Overleaf** para llevar a cabo la docmentación utilizando LaTeX, permitiéndome también acceder desde diferentes dispositivos.
- **XAMPP** para desarrollar y probar el sistema de forma local.
- **Postman** para probar las consultas a la API.
- **Justinmind** para el desarrollo de los bocetos.

6.2. Framework Laravel

Laravel es un conocido framework de código abierto para desarrollar aplicaciones y webs basado en PHP, cuya idea central es la de desarrollar un código de PHP limpio y estructurado. Permite utilizar una sintaxis elegante y expresiva para crear código fácilmente y con muchas funcionalidades.

Una de las principales razones por las que elegí este framework es que posee un sistema de ruteo (organización de las rutas del sistema) que facilita la organización del proyecto, además de poder ser utilizado como una API, pudiendo acceder fácilmente desde la aplicación móvil utilizando el formato JSON.

Laravel se organiza siguiendo la estructura Modelo Vista Controlador (MVC). Se trata de una estructura software que separa la aplicación en tres partes: los datos, la interfaz y la lógica de control.

Utiliza una herramienta llamada Eloquent ORM, lo que permite acceder e interactuar con la base de datos. Esto formaría la parte conocida como "Modelo", pues creamos unas clases que extienden de "Model"para representar las entidades del sistema.

En cuanto al apartado de la "Vista", utiliza el motor de plantillas Blade, que nos permite utilizar una sintaxis que inlcuye funcionalidad PHP en código HTML de forma mucho más simple, permitiendo por ejemplo parametrizar diferentes elementos.

Por último, la lógica se incluye en unos controladores que nos permiten realizar las operaciones necesarias para, por ejemplo, hacer consultas a la base de datos y mostrar la plantilla adecuada [6].

Debido a todas estas funcionalidades y facilidades a la hora de programar, además de la organización y estructura que permite, fue el framework que elegí. Hay que añadir que es uno de los frameworks más utilizado en la

actualidad en el mundo del diseño web, lo cual me ayudo a descubrirlo, a poder encontrar soluciones a problemas en la web debido a su gran comunidad y me ayudó a confiar en sus posibilidades.

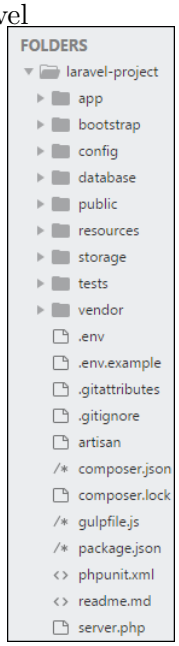
6.3. Programación web

Lo primero que hice fue instalar XAMPP para probar tanto la web como la base de datos de forma local, ubicando la carpeta del proyecto de lavarel en la carpeta de XAMPP. Para la creación de este proyecto se comienza con un comando en la consola de la forma

composer
create-project laravel/laravel sportify
(es necesario tener
instalado tanto PHP como Composer)

Este comando nos genera el sistema de carpetas y archivos que organiza y prepara nuestro proyecto para comenzar a programar que podemos ver en la figura 6.1.

Figura 6.1: Estructura Proyecto Laravel



Ahora pasamos a configurar nuestro proyecto modficando el archivo *.env*, done establecemos detalles como el nombre de la aplicación y, lo más importante, establecemos la conexión con la base de datos. Pero para esto, lógicamente, primero debemos implementar la base datos que ya tenemos diseñada.

En mi caso utilizo la herramienta **phpMyAdmin** que prove de una interfaz gráfica cómoda e intuitiva en la gestión de bases de datos con MySQL. Procedo a contruir la base de datos tal y como está diseñada, teniendo en cuenta las relaciones de dependencia entre tablas, obligatoriedades y demás. Una vez está correctamente implementada, creao un usuario para acceder a esta base de datos desde la aplicación (esto se configura también en el archivo *.env*).

Ahora que tenemos la base de datos implementada y concectada con nuestro proyecto de Laravel es hora de codificar la web como tal. Para ello, lo primero es definir en el archivo *web.php* la ruta hacia la página principal de la siguiente forma:

```
Route::get('/',function(){
    return view('welcome');
})
```

Lo que hace este código es crear una solicitud de tipo GET al tener la ruta '/' que nos carga la vista llamada 'welcome' (esto accede al archivo 'welcome.blade.php', pues cada vista debe ser de formato *nombre_archivo.blade.php*). Estas vistas se encuentran dentro de la estructura del proyecto en la ruta /app/resources/views. El hecho de unas le motor de plantillas Blade nos permite crear una "plantilla"para todas las páginas del sitio para tener una estructura homogénea, y simplemente añadir contenido a ciertas secciones, para esto, dentro de esta carpeta *views* creo una nueva llamada *layout*, donde ubico el archivo que servirá de base para todo el proyecto, denominado *app.blade.php*. Este archivo contiene lo siguiente:

```
<!doctype html>
<html>
    <head>
        <title>Sportify @yield('title')</title>

        <meta name="viewport" content="width=device-width,
            initial-scale=1, shrink-to-fit=no" />
        <meta name="description" content="P gina inicial TFG
            Pablo Delgado" />
        <meta name="author" content="Pablo Delgado" />
        <meta name="csrf-token" content="{{ csrf_token() }}">


        <!-- Aqu  se encuentran los enlaces a los archivos
            CSS necesarios -->


        @yield('css')









    </head>
    <body>

        <!-- NavBar -->


        @include('inc.navbar')
        <div class="overlay"></div>
        <div class="main-container">


          @yield('content')


          <!-- Footer -->
          <div class="page-footer font-small ">

```

```
38                        <!-- C digo del footer de la web -->
39
40
41                    </div>
42
43
44
45
46                </div>
47
48                <!-- Aqu  se encuentran los scripts comunes -->
49
50                @yield('script')
51
52
53
54            </body>
55        </html>
```

He usado comentarios para indicar donde se encuentran diferentes secciones para mostrar más claramente la estructura del archivo qu esu contenido total. Aquí podemos ver diferetnes aspectos de las funcionalidades de Laravel:

- @include('archivo'): Esto implica que el archivo llamado *archivo.blade.php* se inserta en este ubicación, lo que nos facilita la reutilización de código. En caso por ejemplo de que el archivo se encuentre en una subcarpet, como es el caso mostrado con el archivo *navbar* que se encuentra en la carpeta *inc*, nos referimos a él como *inc.navbar*. Este caso concreto nos permite incluir la barra superior (navbar) en todas las páginas que "hereden"de este archivo.
- @yield('campo'): Esto nos permite incluir contenido en esta parte para las páginas que heredan/extienden esta plantilla. Esto obviamente ayuda a la lipmiezadl código.
- @extends('archivo'): Colocando esto en un archivo *blade.php* indicamos que estamos heredando ese archivo, lo que nos permite modificar las partes donde aparezca un @yield() para incluir el contenido que corresponda.
- @section('campo'): Entre un @section('campo') y un @endsection inlcuimos el contenido que queremos que se muestre en la plantilla extendida en el lugar del @yield('campo') correspondiente.

Con estas directivas creamos conexiones entre archivos, heredando de unos a otros o incluyendo archivos completos dentro de otros. Estas posibildades permiten evitar una gran repetición de código (por ejemplo, el caso visto de la barra superior o el menu lateral desplegable que tiene cada usuario) y en cuanto a uniformidad de estilo y composición de varias vistas.

El archivo *welcome.blade.php* contiene la página principal y extiende de esta plantilla que hemos visto. Esta página nos muestra información sobre las posibilidades que ofrece el proyecto, con la intención de atraer usuarios. Desde aquí generamos enlaces a las páginas para iniciar sesión o registrarse (según el tipo de usuario con el que queramos iniciar sesión o registrarnos o haremos en una página u otra). Además podemos acceder al listado de eventos que hay sin estar registrado y pudiendo acceder a la información de cada uno; para ello creo dos nuevas rutas a las que les pasamos información obtenida de la base de datos.

```
1         Route::get('/eventos-info', function () {
2             $eventos = DB::table('evento')->get();
3
4             return view('eventos',[
5                 'eventos' => $eventos
6             ]);
7         });
8
9         Route::get('/eventos-info/{id}', function ($id) {
10            $evento = DB::table('evento')->where('id_evento',$id)->
                  first();
11            $tematica = DB::table('tematica')->where(['id_tematica' =>
                   $evento->id_tematica])->first();
12
13            return view('evento_concreto',[
14                'evento' => $evento,
15                'tematica' => $tematica->nombre_tematica
16            ]);
17        });
```

Le primera nos muestra todos los eventos obteniendolos de la tabla *evento* de la base de datos, y las manda a la vista *eventos.blade.php* con el nombre *eventos*. Para acceder a este dato, y todo aquel que mandemos de esta misma forma a una vista, debemos incluir el nombre de la variable precedido por un '$' y encontrarse entre dos llaves de la siguiente manera: {{ $variable }}. En este caso estamos mandando un objeto que en Laravel se conoce como *Collection* que es similar a un array o vector, compuesto en este acos por varios eventos. Para acceder a cada uno de estos en la vista (por ejemplo para mostrarlo) debemos hacer un bucle, y aquí tmabién nos ayuda Laravel: en lugar de tener que insertar PHP directamente, utilizamos la directiva *@foreach($eventos as $evento)...@endforeach*. De esta forma, entre el principio y el fin de esta directiva tratamos cada evento de fomra individual accediendo a {{ $evento }}, incluso podemos acceder a las diferentes variables de cada evento; por ejemplo si el evento tiene un titulo, podemos mostrarlo usando {{ $evento->titulo }}, y al estar dentro del bucle mostrariamos todos los titulos de los eventos que hayamos pasado.

La segunda, sin embargo, según el identificador de evento que le pasemos a la ruta (esto depende el botón que hayamos pulsado en la página anterior) obtiene toda la información de ese evento concreto, el nombre de la temática

a la que pertenece y la envía a otra vista, donde podremos ver la información concreta de ese evento.

El siguiente paso es permitir los registros e inicios de sesión. Para esto, utilizamos una funcionalidad que nos provee Laravel, denominada *Auth*. Para ello ejecutamos el siguiente comando:

```
1        php artisan make:auth
```

Esto nos genera un conjunto de archivos y carpetas que nos permiten incios de sesión, registros y recuperación de contraseñas. Estos archivos asumen que pretendemos iniciar sesión con la información de los usuarios en una tabla llamada üsers", por lo que debemos modificar esto para inicar sesión según nuestros 3 tipos de usuarios. Para ellos debemos crear un Modelo para cada uno de estos usuarios, pero este modelo debe extender a la clase *Authenticable*, donde indicamos los campos que lo componen, su clave primaria y un campo conocido como *guard*, que es lo que permite a Laravel mantener el inicio de sesión e identificar correctametne al usuario. Aquí muestro el modelo del usuario individual:

```
1  <?php
2
3  namespace App;
4
5  use Illuminate\Notifications\Notifiable;
6  use Illuminate\Foundation\Auth\User as Authenticatable;
7
8  class Individual extends Authenticatable
9  {
10     use Notifiable;
11
12     protected $guard = 'individual';
13     protected $primaryKey = 'id_individual';
14
15     /**
16      * The attributes that are mass assignable.
17      *
18      * @var array
19      */
20     protected $fillable = [
21         'id_individual','nombre_completo_i','fecha_nac_i', 'email', '
               password','foto'
22     ];
23
24     /**
25      * The attributes that should be hidden for arrays.
26      *
27      * @var array
28      */
29     protected $hidden = [
30         'password', 'remember_token',
31     ];
32
33 }
```

Para completar esta configuración debemos modificar el archivo *app/config/auth.php* añadiendo nuevos *guards, providers y passwords* para permitir el correcto inicio de sesión de cada tipo de usuario. El comando mencionado crea también vistas con variables para el inicio de sesión o registro, pero ahora hay que aportar el valor correcto a esas variables según el tipo de usuario. Para ello, debemos crear **Controllers**. Los contrllers incluyen la lógica, y se encuentran en *app/Http/Controllers*. Debemos crear aquí una carpeta para cada tipo de usuario y así tener correctamente organizados los diferentes controllers. A continuación, debemos incluir en cada una de esas carpetas una denominada *auth*, donde encontraremos los controladores de login, registro y recperación de contraseñas. Estos controladores se encargan de llevar a cabo estas funciones y dar valor a esas variables que mencioné antes para que tengan sentido y funcionen las vistas. Por ejemplo, el archivo de login del usuario individual incluye estas funciones (entre otras) para mostrar el formulario de inicio de sesión y para iniciar sesión en si.

```
1  public function showLoginForm()
2     {
3         return view('auth.login',[
4             'title' => 'Login Individual',
5             'loginRoute' => 'individual.login',
6             'forgotPasswordRoute' => 'individual.password.request',
7         ]);
8     }
9
10    /**
11     * Login the individual.
12     *
13     * @param \Illuminate\Http\Request $request
14     * @return \Illuminate\Http\RedirectResponse
15     */
16    public function login(Request $request)
17    {
18        $this->validator($request);
19
20        if(Auth::guard('individual')->attempt($request->only('email','
              password'),$request->filled('remember'))){
21            //Authentication passed...
22
23            return redirect()
24                ->intended(route('individual.home'))
25                ->with('status','You are Logged in as Individual!');
26        }
27
28        //Authentication failed...
29        return $this->loginFailed();
30    }
```

Como vemos, en este caso, al hacer el login correctamente (con email y contraseña) nos redirige a una ruta llamada *individual.home*; esta ruta es la siguiente en el archivo de rutas:

```
1  Route::prefix('/individual')->name('individual.')->namespace('
```

```
    Individual')->group(function(){
2
3       Route::get('/dashboard','HomeController@index')->name('home')->
            middleware('auth:individual');
4
5       ...
6
7   });
```

La línea con *prefix* indica que dentro de esta fución se incluyen todas las rutas del usuario individual. Esto sirve para ahorrar repetición de código y no tener que escribir en cada ruta *individual/...*, pues con estar dentro de esta función ya se aplica a todas. Aquí podemos ver que dentro hay una ruta de tipo GET. Esa ruta se descompone de la siguiente forma: es de tipo GET, nos lleva a la url *dashboard* (realmente es *individual/dashboard*), llama a la función *index* dentro del controlador *HomeController* (este controlador se encuentra obviamente en la carpeta de individual dentro de los Controllers), tinee de nombre *home* para poder acceder usando la función *route()* en lugar de tener que poner la url completa a la hora de solicitarla; y finalmente un *middleware* que evita que podemas acceder a esa url sin haber iniciado sesión como usuario individual. Y esta es la estrucura de las rutas: indicar si es GET o POST, una url, una referencia a una función de algún controller, un nombre (opcional) y un middleware para solo poder acceder habiendo iniciado sesión.

El siguiente paso era ya comenzar con las funcionalidades propias del sistema. Decidí comenzar el desarrollo para el usuario individual, luego para la entidad y finalmente para el administrador.

Había que comenzar con la pantalla tras el inicio de sesión. Esta pantalla nos mostraría unas "tarjetasçon eventos y enalces a ellos, puediendo filtrar para que nos mostrase todos los eventos que hay o solo aquellos con una tematica que interese al usuario. Esta funcionalidad la he programdao usando JQuery, de forma que según el valor de un elemento *select* HTML se muestran unos u otros eventos. Ya hemos visto la ruta para mostrar esta página, aquí como ejemplo de cómo se obtiene y manda la información desde controladores se muestra la función *index* de *HomeController* del usuairo individual:

```
1 public function index(){
2         //Obtengo los eventos de las tematicas que interesqan al
              individual, para mostrarlos en su pagina principal
3
4         $tematicas = DB::table('interesado')->select('id_tematica')->
              where(['id_individual' => Auth::guard('individual')->user
              ()->id_individual])->get();
5
6         $tem=[];
7         foreach($tematicas as $tm){
8             array_push($tem,$tm->id_tematica);
```

```
9           }
10
11          $eventos = DB::table('evento')->whereIn('id_tematica', $tem)->
                get();
12          $todos = DB::table('evento')->get();
13
14          $user = Usuario::find(Auth::guard('individual')->user()->
                id_individual);
15
16
17
18          return view('individual.home',[
19              'nombre' => $user->nombre_u,
20              'cabecera_sidebar' => 'Menu Usuario',
21              'edit_profile' => 'individual.profile',
22              'imagen' => Auth::guard('individual')->user()->foto,
23              'home' => 'individual.home',
24              'logoutRoute' => 'individual.logout',
25              'eventos_interesado' => $eventos,
26              'eventos_todos' => $todos
27          ]);
28      }
29
30  });
```

A continuación desarrollo el menú lateral, que igual que en el menú superior, tiene diferentes condicionales en la vista (*@if(condicion)...@else ... @endif*) para mostrar unos textos, botones o enlaces según el tipo de usuario que se encuentra visitando la web. Luego incluimos este menú desplegable lateral con la directiva *@include('inc.sidebar')*. Todos los menus laterales incluyen una cabecera con información del usuario y un botón para que el propio usuario edite su perfil. Siguiendo el boceto del menú lateral del usuario individual, incluyo los 6 botones necesarios: Mis eventos, Mis Equipos, Calendario, Amigos, Foros y Salir. Cada uno de estos tiene un enlace a la ruta correspondiente según lo que se desee mostrar, pasando a las vistas tanto objetos *collection* como ya hemos visto, como valor a variables para variar ciertos enlaces, lo que permite utilizar una misma vista para todos los usuarios sin necesidad de hacer una individual para cada uno; por ejemplo, si hay un botón que lleva a otra página o hacer cierta acción pero esto depende del tipo de usuario, ponemos la ruta/url de ese botón como una variable *$ruta* y desde la función que corresponda en cada usuario se le da un valor. Aquí un ejemplo:

```
1  ...
2  <div class="col-md-6">
3      <h4 class="mt-5"> A adir comentario </h4>
4          <form method = "post" action = "{{ route($rutaComentar, ['id'
               => $foro->id_foro]) }}" >
5              @csrf
6              <div class = "form-group" >
7                  <textarea id='cuerpo_comentario' class = "form-control
                       " name = "cuerpo_comentario" > </textarea>
8              </div>
```

```
            <div class = "form-group" >
                <input type = "submit" class = "btn btn-success" value
                    = "Comentar" />
            </div>
        </form>
</div>
...

//A continuacion contenido de una funcion de un controlador


...

return view('foro',[
            'nombre' => $user->nombre_u,
            'cabecera_sidebar' => 'Menu Usuario',
            'edit_profile' => 'individual.profile',
            'imagen' => Auth::guard('individual')->user()->foto,
            'home' => 'individual.home',
            'logoutRoute' => 'individual.logout',
            'foro' => $foro,
            'mensajes' => $mensajes,
            'rutaForos' => 'individual.foros',
            'rutaComentar' => 'individual.accion_comentar'
        ]);
}
...
```

En la parte superior vemos, por un lado como la acción de un formulario en la vista está definida por la variable *$rutaComentar*, por otro lado está la parte de la función del controlador donde mandamos información a la vista y vemos como le asigna el valor 'individual.accion_comentar' a la variable 'rutaComentar'.

Después de haber desarrollado la funcionalidad de editar el perfil (esto lo hice simultáneo para los 3 tipos de usuario), me centré en las funcinalidades de los Eventos, pieza clave del proyecto. Lo primero fue ubicar un botón para acceder a un formulario en la pantalla de 'Mis Eventos'(muestro los eventos a los que estoy apuntado) donde rellenar los datos del evento. Este formulario no tiene ningúna complicación, excepto por la representación de la ubicación. Como ya he mencionado en otro punto, adapté la representación de la ubicación (primero era una cadena de textoy ahora dos números reales) para utilizar la API de Google Maps. Para esto debí crear una cuenta en GooleCloud, algo sencillo, lo que provee de una *KEY* para utilizar los mapas de Google en mi sitio. Incluyendo un script de Google con nuestr *KEY* podemos incluir el mapa. Ahora debo crear una función en JavaScript para mostrar el mapa y guardar en unas variables ocultas las coordenadas que más tarde almacenaré en la base de datos:

```
function initMap() {
    var myLatlng = {lat: 37.1881714, lng: -3.6066699};

    var map = new google.maps.Map(
```

```
5           document.getElementById('map'), {zoom: 10, center: myLatlng});
6
7       var vMarker = new google.maps.Marker({
8           position: new google.maps.LatLng(myLatlng),
9           draggable: true
10      });
11
12      google.maps.event.addListener(map, 'click', function(evt) {
13          $("#lat").val(evt.latLng.lat().toFixed(6));
14          $("#lng").val(evt.latLng.lng().toFixed(6));
15
16          map.panTo(evt.latLng);
17          vMarker.setPosition(evt.latLng);
18        });
19
20
21              google.maps.event.addListener(vMarker, 'dragend', function
                    (evt) {
22                  $("#lat").val(evt.latLng.lat().toFixed(6));
23                  $("#lng").val(evt.latLng.lng().toFixed(6));
24
25                  map.panTo(evt.latLng);
26              });
27
28          // Centrar mapa en las coordenadas de la marca
29          map.setCenter(vMarker.position);
30
31          // A adir marca al mapa
32          vMarker.setMap(map);
33  }
```

Simplemente colocamos un marcador (típica marca roja para señalar una ubicación) y peritimos que se arrastre o simplemente haciendo clic, se varie el valor de estas variables ocultas que tenemos en el formulario. El botón del formulario realiza una solicitud POST mediante una ruta a un nuevo controlador llamado *EventsController* (cada usuario tiene su propio controlador para los eventos). Aquí se encarga de subir y guardar la foto elegida para el evento, crear un nuevo evento con los datos del formulario, crear su correspondiente grupo de conversación/chat y apunto al creador, pues al crear el evento, automáticamente se apunta como un participante más.

Una vez podemos crear eventos, debemos poder ver su información. Creamos una ruta de tipo GET que permite recibir un identificador de evento, para así cargar su información en una nueva vista. Esta incluye diferentes tablas (para ello he usado el plugin Datatables de Bootstrap, pues provee de funcionalidades como búsqueda por texto y ordenar por cada una de sus columnas, además de poder mostrar de forma paginada eliginedo la cantidad de filas por página) con los participantes, clasificación actual y enfrentamientos. Mediante condicionales en la vista, usando variables obtenidas en el controlador correspondiente, diferneciamos si el usuario puede apuntarse o no, bien porque ya haya empezado el evento, ya esté apuntado (podría desapuntarse) o está lleno. Es importante saber si el evento es para equipos

o individuos; si es para equipos, al pulsar en apuntarnos se nos muestra una lista con nuestros equipos y elegimos con cual nos queremos apuntar, mientras que si es individual simplemente se apunta el usuario; aunque en ambos casos lógicamente además de añadir las entradas correspondientes a las tablas pertinentes, debe editarse el evento para incrementar el número de participantes (del mismo modo ue se debe reducir al desapuntarse). Para mostrar la lista de equipos utilizamos *pop-ups* de tipo modal utilizando clases de Bootstrap, JavaScript y la directiva *@include()* para incluir el archivo con el elemento emergente. Si participas en un evento puedes acceder a su chat, y comunicarte con el resto de participantes. Además, en caso de ser el creador del evento puedes editar ciertos valores del evento. Como creador del evento tmabié puedes realizar más acciones: alquilar pista y/o equipamiento para el evento (accedemos a una vista con la información de los horarios libres de la pista/equipamiento y se alquila), crear enfrentamientos entre participantes y modificar estos enfrentamientos para incluir el resultado. Si el evento se trata de na liga, el hecho de que un resultado sea favorable a un particpante le añadirá un punto en caso de victoria y 1 en caso de empate; si se trata de un torneo, se pedirá que se indique a qué fase pasa (cuartos de final, semifinal...); en caso de ser un evento normal único sipmlemente se añade la información del resultado. Además, volvemos autilizar la API de Google Maps para mostrar la ubicación del evento en un mapa.

La siguiente rama a desarrollar será el calendario. Para esto, me he ayudado del calendario en JQuery denominado *FullCalendar*. Este plugin nos permite mostrar un calendario con multitud de opciones. En este caso nos encargamos de mostrar los eventos en los que participa el usuario que tiene sesión iniciada (de forma individual o con alguno de sus equipos) pudiendo pasar entre meses y, al clicar en un evento (de diferentes colores), ver la información de dicho evento. Para poder usar este calendario debemos incluir un script externo, y además, en este caso concreto para mostrar lo que yo necesito, crearlo con esta configuración:

```
1  <script>
2      $(document).ready(function() {
3          // page is now ready, initialize the calendar...
4          $('#calendar').fullCalendar({
5
6              monthNames: ['Enero','Febrero','Marzo','Abril','Mayo','
                   Junio','Julio','Agosto','Septiembre','Octubre','
                   Noviembre','Diciembre'],
7              monthNamesShort: ['Ene','Feb','Mar','Abr','May','Jun','Jul
                   ','Ago','Sep','Oct','Nov','Dic'],
8              dayNames: ['Domingo','Lunes','Martes','Mi rcoles','Jueves
                   ','Viernes','S bado'],
9              dayNamesShort: ['Dom','Lun','Mar','Mi  ','Jue','Vie','S  b
                   '],
10             firstDay: 1,
11             buttonText: {
```

```
12                  today: 'Hoy'
13              },
14              header: {
15                  left: 'prev,next ',
16                  center: 'title',
17                  right: 'today'
18              },
19              events : [
20                  @foreach($eventos as $evento)
21                  {
22                      title : '{{ $evento->titulo_e }}',
23                      start : '{{ $evento->fecha_ini }}',
24                      end : '{{ $evento->fecha_fin }}T23:59:59',
25                      allDay: false,
26                      color : '#'+(Math.random()*0xFFFFFF<<0).toString
                            (16),
27                      url : '{{ url('individual/evento/'.$evento->
                            id_evento) }}'
28                  },
29                  @endforeach
30              ]
31          })
32      });
33  </script>
```

Este script muestra el calendario donde se encuentra el elemento con *id=calendar*. Aquí surgió un problema, pues en principio no debería configurar el idioma de forma manual, pero esta última versión de Laravel no era compatible con la configuración de idioma del plugin FullCalendar, por lo que tengo que hacerlo manualmente. Además, configuro la cabecera y muestro los eventos que he calculado anteriormente en el controlador utilizando colores aleatorios en los eventos.

Pasamos a la gestión de los amigos. Esta es una sección muy simple que en una versión futura y con más tiempo de desarrollo podría perfeccionarse y aumentar su funcionalidad. De momento, se pueden enviar solicitudes de amistad mediante un botón muestra un desplegable con los usuarios individuales del sistema (dispone de un buscador por nombre programado utilizando JavaScript), aceptar o denegar esas invitaciones (si tenemos solicitudes pendientes nos aparece un boton resaltado en la página de Amigos) y ver una lista de nuestros amigos. Esta es seguramente la sección funcional con mayor margen de mejora.

Pasamos a la administración de los equipos. Al igual que en el caso de los eventos, disponemos de una página principal con "tarjetas"de equipos (pudiendo acceder a cada uno de ellos) y un botón para crear un equipo (formulario básico son su información, además de aountar al creador al equipo). También incluye un botón donde podremos ver si tenemos invitaciones, pudiendo aceptarlas o denegarlas. Dentro de la información de cada equipo, obviamente se muestran su nombre, descripción e imagen, además de una tabla con los integrantes, un boton para salir del equipo y otro para invitar a nuevos usuarios (también con un buscador por nombre de usuario).

Por último (obviando el botón de salir/cerrar sesión), gestionamos los foros. La página principal de foros muestra un listado de los foros existentes, habiendo uno para cada una de las temáticas dadas de alta en el sistema.Al hacer click sobre uno de los foros entremos en él. Aquí enprimer lugar se nos muestra un listado de los mensajes que hay (obviamente obtenidos en el controlador oportuno con las consultas necesarias) y se nos permite añadir un comentario a dicho foro con un formulario. De cada mensaje vemos la foto y nombre del usuario que lo ha publicado, el contenido del mensaje y la hora y fecha del mismo.

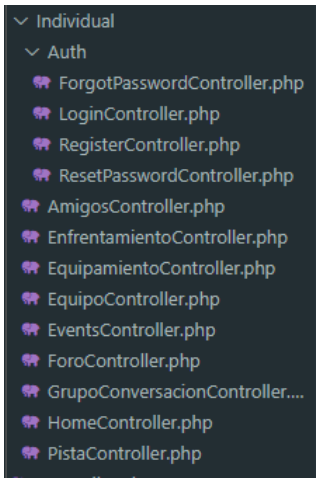


Figura 6.2: Controladores del Usario Individual

Para elementos como los mensajes en foros/grupos de conversación, el equipamiento y pistas para alquilar y las temáticas creé un modelo para facilitar el acceso a la base datos conforme fui aprendiendo mejor Laravel y fui acostumbrandome a sus ventajas no tan visibles a primera vista pero muy útiles. La imagen 6.2 muestra la lista de Controllers usados para el usuario individual.

Hasta aquí la funcionalidad del usuario individual en la web, ahora pasamos al usuario entidad/empresa.

La funcinalidad de este usuario es exactamente igual que la del usuario individual en lo relacionado con los foros, con "Mis Eventos"(en este caso son solo aquellos queha creado dicha entidad) y el calendario; por lo que la estructura y conjunto de controladores es muy similar. En su página principal se nos muestran directamente todos los eventos existentes, pues no tenemos temáticas preferidas.

En cuanto la gestión de los eventos, las entidades no pueden apuntarse a eventos, solo crearlos, por lo que el botón de apuntarse/desapuntarse no se muestra en la información del evento (esto se consigue con los condicionales que nos aporta Laravel sin necesidad de crear otra vista diferente). Por tanto, tampoco podemos alquilar equipamiento ni pistas, pero sí podemos crear y modificar enfrentamientos, al igual que participar en el grupo de conversación, si somos los creadores de dicho evento. Obviamente si somos los creadores también podemos editarlos.

Tenemos la sección de 'Equipamiento'. Aquí se nos muestra todo el equipamiento que tenemos como entidad, y se nos permite crear nuevo equipamiento. Esta creación tiene una peculiaridad: a la hora de registrarlo, además de introducir sus datos principales como nombre o temática, elegimos fecha de inicio, fecha de fin, hora de inicio y hora de fin. ¿Para qué? Para gestionar las reservas. Estos datos nos permitirán crear ."equipamientos"(entradas en la tabla de los equipamientos) para cada hora entre los días y la shoras indicadas. De esta forma, cuando el usuario individual quiere alquilar un equipamiento, se le muestra una tabla con todas las horas de los días en las que está disponible. Es decir, si seleccionamos que se cree entre el día 2 y 4 de junio entre las 5 y las 7, se nos generaría una posibilida de reserva/equipamiento el dia 2 a las 5, el día 2 a las 6, el día 2 a las 7, el día 3 a las 5... y así sucesivamente.

La gestión de las pistas es exactamente igual: una página con la lista de pistas y la posibilidad de crearlas con rangos horarios y de días.

La visión de una pista o equipamiento concretos nos muestra su nombre, temática y creador; además de una gran tabla con los días y las horas disponibles. En caso de ser un usuario individual además aparecería una coluna con un botón para reservar (se asignaría al evento desde el que se hubiese accedido a esta pantalla).

En cuanto a la entidad no hay mucho más que contar, pues como ya he mencionado es muy similar al usuario individual, así que pasamos al administrador.

El administrador es el único usuario que no puede registrarse él mismo. Solo otro administrador puede crearlo, para mantener ese tipo de usuario

limitado.

En su página principal tenemos enlaces a las funciones principales: dar de alta a un administador, gestionar los usuarios, gestionar los adminitradores y gestionar los eventos.Además disponemos también de un menú lateral desplegable con enlaces a la gestón de equipos y temáticas.

La gestión de usuarios, eventos y equipos es prácticametne idéntica: una tabla con la información de cada elemento y un botón para eliminar cada uno (aunque en el caso de los usuarios incluyen un botón para visitar el perfil de dicho usuario). Por otro lado, la gestión de temáticas y administradores es también muy similar pero incluyen la opción de crear nuevos; es decir, podemos crear nuevos administradores y nuevas temáticas. Cabe recalcar que la creación de una nueva temática implica la creación de un nuevo foro.

Como podemos ver, la versión del administrador está mas centrada en gestión, tratandose básicamente de un panel de control con los elementos más significativos; por lo que está enfocada al acceso mediante un ordenador a través de un navegador web. Por esto, su cantidad de controladores es más baja, pues su funcionalidad es muy concreta y más claramente diferenciada en secciones.

6.4. Programación app

6.4.1. Preparación Laravel

Antes de comenzar directamente con la programación estrictamente para la aplicación Android, debemos conocer qué necesitamos incluir en los archivos de nuestro proyecto web de Laravel para utilizarlo como una **API** (*Application Programming Interface*) [7] y acceder a la información necesaria sin tener que acceder a la base de datos directamente desde la aplicación, faacilitando en gran medida este paso de información y manteniendo el patrón MVC (Modelo-Vista-Controlador).

Al igual que para definir las rutas de la página web utilizabamos el archivo *web.php*, para definir las rutas necesarias para la API utilizamos *api.php*. Este archivo mantiene la misa estructura que le anterior, expresando las rutas de la misma forma: Route::(get/post)('url','Controlador@funcion'). Por tanto, dentro de los controladores (carpeta Controllers) he añadido una nueva carpeta llamada 'Api', para mantener aquí todos los controladores necesarios para la aplicación. Cabe destacar que la aplicación móvil está diseñada para que la utilicen únicamente los usuarios de tipo individual, pues son quienes más probablemente necesiten acceder a ciertas informaciónes desde su dispositivo móvil y a la funciones extras de la app; a parte de que obviamente la web puede verse en cualquier dispositivo móvil. Esto es

importante porque a la hora de programar las funciones necesarias en los controladores de la API damos por hecho que el tipo de usuario que accede es de tipo individual.

Además de las rutas, es necesario mantener la sesión del usuario para asegurar que los usuarios que accedan han iniciado sesión, como hacíamos en la web gracias al paquete Auth de Laravel. En este caso utilizamos uno llamado *Passport*. También hace falta definir un *guard* para este inicio de sesión, al igual que para la web, llamado 'api' y especificando que usaremos "Password"para identificar al usuario. Esto debemos hacerlo en el archivo *auth.php*.

```
'guards' => [
    'web' => [
        'driver' => 'session',
        'provider' => 'users',
    ],

    'api' => [
        'driver' => 'passport',
        'provider' => 'individuals',
        'hash' => false,
    ],

    'individual' => [
        'driver' => 'session',
        'provider' => 'individuals',
    ],

    'admin' => [
        'driver' => 'session',
        'provider' => 'admins',
    ],

    'entidad' => [
        'driver' => 'session',
        'provider' => 'entidads',
    ]
],
```

Figura 6.3: Guard para API

Lo siguiente que debemos hacer es modificar el archivo *AuthServiceProvider.php* para configurar aspectos de las rutas de Passport, además de establcer los tiempos de validez (hasta cuando se mantienen) de los tokens que permiten a los usuarios acceder a la aplicación.

Al instalar el paquete Passport, se nos modifica la base de datos, añadiendo algunas tablas que son las que se encargan de mantener los tokens que permitan acceder a los usuarios. Estas tablas se alteran al llevar acabo acciones como el inicio de sesión (login) o cerrado de sesión (logout), añadiendo nuevas entradas o modificándolas para que cierto usuario individual posea un token que le permita consumir el resto de funciones de la API para las que lo necesite. Esto además nos permite controlar quienes utilizan la app móvil consultando la base de datos.

Ahora que ya sabemos donde definir los controladores y, por tanto, sus funciones, el proceso es muy similar a las funciones diseñadas para la web; con la salvedad de que, al devolver información, en lugar de enviar una vista con ciertas variables (ahora carecería de sentido, pues la información hay que gestionarla en la aplicación móvil) se devuelve un objeto de tipo JSON [8]. Por ejemplo, para devolver el listado de todos los eventos, tendríamos la siguiente función en el *Api/EventsController.php*:

```
1 public function showEvents(){
2     $eventos = DB::table('evento')->get();
3
4     return response()->json(['data' => $eventos],200,[],
          JSON_NUMERIC_CHECK);
5 }
```

Aquí obtenemos los eventos de igual forma que en las cosultas que

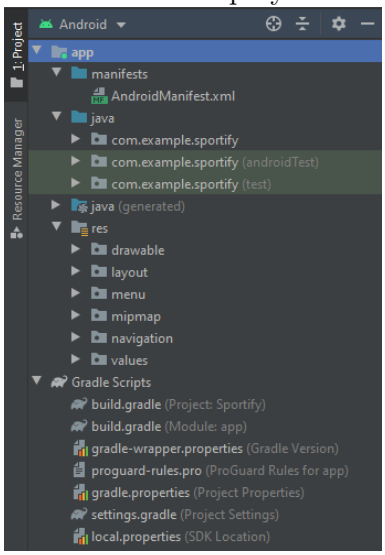
hacíamos para la página web (accediendo a la tabla correspondiente de la base de datos), pero devolvemos un JSON con un campo 'data' que contiene los eventos.

6.4.2. Programación Android Studio

Aquí es donde comienza el desarrollo de la aplicación móvil como tal. Dada la gran facilidad para programar y probar en dispositivos Android, el desarrollo lo llevaré a cabo utilizando la herramienta Android Studio, una de las más conocidas y utilizadas en el mundo de la programación de dispositivos Android. Además, el haber trabajado anteriormente con este software y conocerlo a cierto nivel me hace estar familiarizado con el entorno y entender ya los principios del mismo. Cabe destacar que entre lenguajes para programar en Android, como Kotlin, yo m decanto por Java para la 'lógica' y XML para las interfaces y aspectos gráficos.

El primer paso es obviamente descargar Android Studio [9]. El proceso de instalación es realmente sencillo, similar al de cualquier otro programa de, en mi caso, Windows 10. Una vez instalado el programa, comenzamos creando el proyecto; en nuestro caso se llamará *Sportify*. A la hora de crearlo debemos elegir el tipo de *Activity* (una actividad proporciona la ventana en la que la app dibuja su IU) [10]; para tener más libertad y hacer el desarrollo desde cero no utilizaré plantillas para inicio de sesión o similares pues suelen introducir elementos que no utilizaremos y pueden complicar el desarrollo, por lo que escogeremos una actividad en blanco. En la siguiente imagen podemos ver la estructura de árbol que nos genera la creación del proyecto.

Figura 6.4: Estructura proyecto Android



Los archivos dentro de *Gradle Scripts* nos permiten configurar en el proyecto elementos como dependecias con ciertos plugins o paquetes que necesitemos para el desarrollo. El archivo *AndroidManifest.xml* incluye también aspectos de configuración, como la declaración de las actividades, establecer cuál será la inicial o permitir la conexión a internet por parte de la app. Ahora pasamos a los contenidos más concretos del proyecto: dentro de la primera carpeta denominada *java/com.exampl. sportify* encontramos las diferentes clases de Java que aportan la lógica (por ejemplo, cada actividad cuenta con su propia clase de Java);dentro de la carpeta *res* encontramos los 'recursos gráficos' como imágenes (en *drawable*), las interfaces de las actividades (en *layout*), los iconos (en *mipmap*) o algunas constantes como colores o cadenas de texto en *values*. Como podemos ver, esta estructura ayuda a mantener el patrón MVC en el que nos estamos basando en todo el proyecto, pues por un lado mantenemos la lógica en archivos Java totalmente separados de las interfaces visuales XML, y separados ambos de los datos que obtenemos en algunas clases Java a través de la conexión con la API de Laravel que ya tenemos preparada.

Ya podemos comenzar con el desarrollo de la aplicación en si. Para ello, esa primera actividad/pantalla que tenemos al crear el proyecto debemos modificarla como conideremos. En este caso, será una pantalla de presentación con el logo y nombre de Sportify, junto a un botón que nos redirija hacia una nueva actividad para iniciar sesión. Para la gestión de estas acciones (pulsación de botones) y la referencia a diferentes variables de las vistas haré uso de la librería **Butterknife**. Esta librería nos facilita el acceso a los campos de la pantalla reduciendo considerablemente la cantidad de código necesario para ello. Para utilizarla, simplemente debemos incluir la siguiente dependencia en el archivo *build.gradle*:

implementation 'com.jakewharton:butterknife:10.2.1'
annotationProcessor 'com.jakewharton:butterknife-compiler:10.2.1'

Ahora nos centramos en como funcionan las clases de cada actividad. Cada una de estas clases tiene un métoido *onCreate()* que se ejecuta nada más comenzar dicha actividad, que, entre otras cosas, relaciona duicha actividad con su vista/layout correspondiente. En este método, para utilizar *Butterknife* escribimos 'ButterKnife.bind(this);'. A partir de aquí, para hacer referencia a, por ejemplo, un campo de texto de la vista simplemente usamos '@BindView(R.id.user) EditText username;' y ya tendríamos una variable llamada 'username' referenciando al campo con identificador 'user' del layout correspondiente (sin esta librería, el acceso a cada campo se consigue de forma más tediosa y repetitiva); y para gestionar las acciones tras, por ejemplo, pulsar un botón (como el que nos lleva a la actividad de login) se hace de la siguiente forma:

```
@OnClick(R.id.go_login)
void go_login() {
    Intent intent = new Intent(MainActivity.this, LoginActivity.class)
        ;

    startActivity(intent);
    finish();

 }
```

Al hacer click en el botón con un identificador igual a 'go_login' se ejecuta la función 'go_login'. Como se ve, esta función comienza una nueva actividad (función *startActivity()*), y para indicar desde que actividad hacia que actividad deseamos ir creamos un objeto tipo *Intent*, en el que indicamos en que actividad estamos (MainActivity.this) y que actividad queremos ejecutar (LoginActivity, que debemos de haberla creado ya, aunque a continuación explicaremos su codificación). Por úlimo, utilizamos *finish()* para que no se nos quede abierta la actividad de la que nos vamos y así ahorrar recursos.

El siguiente paso es proceder a crear la actividad de inicio se sesión a la pasamos tras pulsar este botón. Simplemente en el arbol de la estructura de nuestro proyecto, haciendo click derecho, indicamos que queremos crear una nueva actividad (una actividad en blanco), a la que llamaremos 'LoginActivity'. Esto nos genera su clase Java y su vista XML correspondientes, además de añadirse al archivo *AndroidManifest.xml*.

Ahora procedemos a diseñar esta vista. Primero ubicamos un título que nos indica que estamos en la pantalla de inicio de sesión. Tras esto, incluimos los campos de entrada del email y la contraseña. Es importante poner a cada uno de estos elementos una 'relación de posición', ya sea relacionándolos entre ellos o con los bordes de la pantalla, para ubicarlos correctamente y se adapten a los diferentes tamaños de los dispositivos. Estos campos de entrada van a ser especiales pues utilizaremos *Material Design* (debemos incluirlo como dependencia), lo que nos permite mejorar estos campos estéticamente, pues en el campo de 'Email' por ejemplo aparecerá escrito un texto indicando que ahí debe ubicarse el email, pero al clicar sobre este para escribir, este texto se desplaza justo arriba del campo y se hace más pequeño, permitiéndonos escribir. Para incluir este tipo de campo se hace la siguiente forma en el archivo XML:

```
<com.google.android.material.textfield.TextInputLayout
    android:id="@+id/til_email"
    android:hint="Email"
    android:layout_width="match_parent"
    android:layout_height="wrap_content">

    <com.google.android.material.textfield.TextInputEditText
        android:layout_width="match_parent"
```

```
9           android:inputType="textEmailAddress"
10          android:drawableStart="@drawable/email_icon"
11          android:drawableLeft="@drawable/email_icon"
12          android:drawablePadding="10dp"
13          android:imeOptions="actionNext"
14          android:layout_height="wrap_content" />
15
16  </com.google.android.material.textfield.TextInputLayout>
```

Como vemos, el elemento *com.google.android.material.textfield.TextInputLayout* engloba todo, y es donde aparece el texto informativo, mientras que *com.google.android.material.t* hace referencia al propio campo a rellenar. Como extra, en el campo de la contraseña se incluye un botón para ocultar o dejar ver el contenido de la misma (que sea legible o solo aparezcan circulos negros).

Preparación conexión API

Ahora llega la parte más compleja: extraer información a través de la API de Laravel conectando con el servidor. Para llevar esto a cabo es necesario crear ciertas clases y una interfaz que usaremos para solicitar las funciones. Para esto es clave la utilización de la librería **Retrofit**. Retrofit nos permite acceder a nuestra API como una interfaz de Java, facilitando en gran medida las consultas y conexiones. Para utilizar Retrofit simplemente necesitamos incluir las dependecias necesarias en el archivo *build.gradle*.

Dentro de *app/java/com.example.sportify*, que es donde tenemos las clases de Java necesitamos crear dos carpetas: *entities* (entidades) y *network* (red). Además crearemos las clases *TokenManager.java* y *Utils.java*, que a continuación explicaré su funionamiento.

- TokenManager.java: Aquí gestionamos lo relacionado con el token del usuario, que le da la capacidad de ser aceptado para sus consultas a la API; lo hacemos utilizando el patrón *Singleton*. Aquí creamos, accedemos y eliminamos el token.
- Utils.java: Aquí nos encargamos de obtener los posibles errores en las consultas.
- entities: Esta carpeta incluye clases Java que nos permiten obtener la información.
    - AccessToken.java: Nos permite obtener la información del token que obtenemos en formato JSON.
    - ApiError.java: Nos permite mostrar ciertos errores en los datos obtenidos.
    - Clases para los objetos según lo que queramos obtener: una clase para los Eventos, una clase para lso Equipos,...

- Clases para listas de objetos, por ejemplo si hacemos una consulta de eventos, creamos una clase que sea ListaEventos, lo que nos facilita luego el acceso a la misma.

- network: Aquí configuramos los aspectos necesarios para las consultas a la API y la configuración de Retrofit:
  - ApiService.java: Se trata de una interfaz que ofrece unas funciones que enlazan con la API.
  - RetrofitBuilder.java: Esta clase prepara Retrofit y sus funciones para acceder a la URL correcta con los campos necesarios (por ejemplo el token).

El archivo *ApiService.java* es tan sencillo como util para consultar la API. SU estrcutrua es la siguiente:

```
1     @POST("login")
2     @FormUrlEncoded
3     Call<AccessToken> login(@Field("email") String username, @Field("
          password") String password);
4
5
6     @POST("refresh")
7     @FormUrlEncoded
8     Call<AccessToken> refresh(@Field("refresh_token") String
          refreshToken);
9
10    @GET("eventos")
11    Call<ListaEventos> eventos();
```

En primer lugar se utiliza '@POST' o '@GET' según el tipo de consulta junto con la URL/ruta (su URL después de *.../public/api/*). En caso de que pasemos información a la consulta (por ejemplo el login) se le añaden los campos que le vamos a pasar la llamada (objeto **Call**). El caso de 'eventos' nos devuelve una lista de eventos (el JSON que devolvemos en Laravel). Pues en este archivo vamos definiendo las diferentes funciones que nos permitirán hacer estas consultas desde las actividades correspondientes.

Utilización de API

Ahora en las actividades debemos utilizar estas clases que hemos creado para acceder a la información que necesite cada actividad. La primera a gestionar es la consulta para iniciar sesión. En cada actividad que necesite conectar con la API necesitamos tener las siguientes variables:

- ApiService: Necesitamos una variable de ApiService para las consultas que necesiten un token qu eidentifique al usuario y otra para aquellas que no lo necesiten.

- TokenManager: Una instancai de TokenManager nos permite conocer si se ha iniciado la seisón correctamente y se permite acceder a las funciones que lo requieran.
- AwesomeValidator: Una instancia de esta clase (se incluye como dependencia del proyecto) nos permite validar ciertos aspectos de los campos que se quieran introducir (como el email o contraseña a la ora de inicar sesión).
- Call: Necesitaremos una instancia de Class¡...¿para el tipo de objeto que vayamos a recbir y queramos mostrar.

Todas estas variables se inicializan en la función *onCreate()* de la actividad. En el caso del login, gestionamos el click del botón de incio de sesión con ButterKnife y en la función correspondiente llamamos a la interfaz para hacer la consulta de la siguiente manera:

```
1  void login() {
2
3          String email = tilEmail.getEditText().getText().toString();
4          String password = tilPassword.getEditText().getText().toString
               ();
5
6          tilEmail.setError(null);
7          tilPassword.setError(null);
8
9          validator.clear();
10
11         if (validator.validate()) {
12             call = service.login(email, password);
13             call.enqueue(new Callback<AccessToken>() {
14                 @Override
15                 public void onResponse(Call<AccessToken> call,
                       Response<AccessToken> response) {
16
17                     Log.w(TAG, "onResponse: " + response);
18
19                     if (response.isSuccessful()) {
20                         tokenManager.saveToken(response.body());
21                         startActivity(new Intent(LoginActivity.this,
                               HomeActivity.class));
22                         finish();
23                     } else {
24                         if (response.code() == 422) {
25                             handleErrors(response.errorBody());
26                         }
27                         if (response.code() == 401) {
28                             ApiError apiError = Utils.converErrors(
                                   response.errorBody());
29                             Toast.makeText(LoginActivity.this,
                                   apiError.getMessage(), Toast.
                                   LENGTH_LONG).show();
30                         }
31                     }
32
33                 }
```

```
34
35                  @Override
36                  public void onFailure(Call<AccessToken> call,
                        Throwable t) {
37                      Log.w(TAG, "onFailure: " + t.getMessage());
38                  }
39              });
40
41          }
42
43      }
```

Esta función en primer lugar obtiene los valores que hemos introducido en los campos de 'email' y 'password' en la vista (accedemos a ellos también gracias a que estamos usando ButterKnife). A continuación hacemos la llamada a *service.login(emailo,password)*. 'service' es la referencia a ApiService, y llamamos a su función 'login' pasando los parámetros correspondientes. Tras esto gestionamos su respuesta (es realmente importante destacar qe gracias a Retrofit estas consultas se hacen en diferentes hilos/hebras, de forma paralela, lo que facilita la ejecución de la aplicación sin paralizarla para estas consultas): en este caso su respuesta es el token en caso de ser un login correcto (en este caso se guarda el token para el usuario y pasmaos a la actividad principal del usuario); si el login no es correcto se mostrarán los errores pertinentes (los *Toast* muestra un 'pop up' con el texto que indiquemos). Es muy importante definir un destructor en la actividad par aque, cuando acabe, se destruya el objeto Call y no quede esa 'llamada' abierta permanentemente.

Si el login es correcto pasamos a la actividad 'HomeActivity', así que debemos crearla. En este caso, al crear la actividad sí que usaremos una platilla de Android Studio, aquella que nos aporta una barra de navegación inferior, para los accesos rápidos/principales, los cuales se muestran en los bocetos.Esta actividad hace uso de elementos conocidos como *Fragment*, que se pueden enteder como 'diferentes pantallas' para una misma actividad, y esta es la forma en que se implementa esta barra inferior. Una vez creada podemos modificar la barra y añadir los 'fragments' que queramos. El fragment principal nos mostrará un boton para cerrar sesión y una lista de los eventos. Para ello, cada fragment posee una función *onCreateView()* equivalente al *onCreate()* de las actividades; y en este método configuramos aspectos como el token y el ApiService para poder hacer la consulta de la lista de eventos. Para pdoer mostrar los eventos en la lista de forma que veamos, por ejemplo, el nombre, la temática y la foto necesitamos crear un 'adaptador'. Este adaptador es una clase qu enos permite darle el aspecto que queramos a cada elemento de esa lista a la hora de mostrarla (estas listas se conocen como *ListView*). En ese adaptador diseñamos cómo se muestra cada evento y le decioms cómo debe asignar los valores. El caso concreto de la imagen es especial: uso la librería **Picasso** (incluimos la dependecia

pertinente). Esta librería nos permite obtener fácilmente (en una linea) una imagen desde una URL y asignaral a un campo *ImageView* del elemento de la lista. Además, las obtiene de forma paralela a la ejecución de la aplicación, lo que no detiene o ralentiza su ejecución.

Ahora necesitamos gestionar la visulaización de cada evento. Para ello debemos utilizar un 'listener' (está 'escuchando' esperando a que tenga lugar cierto suceso y ejecutar el código crrespondiente) en la lista que, al hacer click sobre uno de sus elementos, nos devuleve su índice en la lista; con esto podemos obtener el objeto completo y a continuación pasarlo a una nueva actividad que nos muestre esa información ('InfoEventoActivity.java' con su respectivo archivo XML). Para poder pasar el objeto 'Evento' de una actividad a otra, la clase 'Evento' de debe implementar *Serializable*.

Ahora esta actividad tiene la información del evento, pero falta la información referente a los enfrentamientos, integrantes, puntuaciones, si ha terminado el evento o si ya estás apuntado. Esto lo resolvemos con una nueva consulta mediante retrofit, devolviendo con JSON un objeto con toda esta infromación (esto requiere crear una clase Java para los participantes, otra para los enfrentamientos,... y de esta forma tratamos estos datos de forma ordenada, clara y legible). Hacemo esta llamada nada más crear la actividad, y nos encargamos de crear las tablas que hemos mencionado (*TableLayout*) y asignarles los valores y el estilo que queremos. Hacemos lo mismo con todos los datos del evento que queremos mostrar, incluyendo el mapa (utilizando la misma *KEY* que usamos para la web). El uiso del mapa tenía complejidad pues, aunque incluir un mapa no es complejo (incluyendo ciertas dependencias), como toda la información no entra en la pantalla directamente, necesitamos permitir *scroll* y esto a priori impide interactuar con el mapa. Por tanto, debo colocar una imagen invisible justo sobre el mapa y, con el código que mnuestro a continuación impedimos que se puede hacer *scroll* justo en la zona del mapa y así permitir al usuario moverse y hacer zoom por el mapa.

```
1  //transparente referencia a la imagen invisible y
2  //id_scroll al campo que permite el scroll
3  transparente.setOnTouchListener(new View.OnTouchListener() {
4      @Override
5      public boolean onTouch(View view, MotionEvent motionEvent) {
6
7          int action = motionEvent.getAction();
8          switch (action) {
9              case MotionEvent.ACTION_DOWN:
10                 // Disallow ScrollView to intercept touch events.
11                 id_scroll.requestDisallowInterceptTouchEvent(true);
12                 // Disable touch on transparent view
13                 return false;
14
15             case MotionEvent.ACTION_UP:
16                 // Allow ScrollView to intercept touch events.
17                 id_scroll.requestDisallowInterceptTouchEvent(false);
```

```
18                  return true;
19
20              case MotionEvent.ACTION_MOVE:
21                  id_scroll.requestDisallowInterceptTouchEvent(true);
22                  return false;
23
24              default:
25                  return true;
26          }
27      }
28  });
```

Así conseguimos ver la información del evento en concreto. Ahora, de la misma forma que hicimos en la web, según si estamos apuntados al evento o este ha acabado y de más opciones, gestionamos el botón de apuntarse/desapuntarse. En caso de estar apuntados se nos permite desapuntarnos (botón en rojo), si no podemos apuntarnos saldrá en gris (con el texto que explica por qué no podemos) y si nos podemos apuntar saldrá en azul. Si nos podemos apuntar y es un evento para usuarios indivuales al pulsarlo se hace una petición POST a la API y se nos apunta; mientras que si es un evento para equipos, se nos abre un diálogo con nuestros equipos (los hemos obtenido en la consulta anterior) y al elegir uno, nos pauntamos con dicho equipo.

Esta misma forma de listar y ver los eventos se utiliza para los eventos propios del usuario ('Mis Eventos') y de forma semejante para los equipos, donde podremos salir de un equipo al entrar en la actividad donde podemos ver su información.

En la versión móvil había planteado que, en lugar de acceder a los grupos de conversación de cada evento desde el propio evento, sería más útil que estos grupos tuviesen una pestaña propia para agilizar el acceso a estos. Por tanto, debemos creare un nuevo fragment para listar los grupos de conversación y permitir el acceso a estos de la misma forma que con los eventos o equipos. La pantalla de cada grupo mostrará la lista de mensajes hasta el momento (usuario, hora y contenido), un campo para escribir un nuevo mensaje y un botón para enviar el mismo. Esta función se llevará a cabo de forma similar que el rsto pero con los parámetros que le correspondan: llamando a una función del *ApiService* que invoque la ruta correspondiente de la API de Laravel.

Por último, nos dedicamos al desarrollo de las herramientas auxiliares de la aplicación: el reloj y el marcador. Para esto, creamos un nuevo fragment al que accedemos desde el menú infefrior en el que incluimos los campos que necesitamos: el reloj con sus botones de 'empezar','parar' y 'reiniciar', un campo en el que introducimos un número indicando cuantos minutos queremos que pasen para que suene (similar a una alarma), dos campos con los puntos del marcador y un par de botones de '+' y '-' en cada número

de marcador para aumentar y decrementar sus valores, además de un botón para poner ambos campos a 0.

Para el reloj haremos uso de unas variables en la actividad que mantendrán los minutos, segundos y milisegundos. Estos valores comenzarán a cambiar al pulsar el botón 'START' del reloj haciendo uso de un objeto tipo *Handler* qu enos permite crear una nueva hebra paralela que vaya alterando estos valores según pasa el tiempo, gracias a la utilidad *SystemClock*. De esta manera gestionamos también la pausa y reinicio del reloj, alterando los valores de estas variables que indican el tiempo y, según estas, modificar el texto que veremos en pantalla que nos muestra el tiempo transcurrido. Aquí también comprobamos si hemos introducido algún valor para establecer una alarma; si llegamos a los minutos indicados (haciendo los calculos correspondientes, pues SystemClock trabaja en milisegundos), el reloj continuará pero comenzará a reproducirse un sonido hasta que detengamos el tiempo. Tras esto podremos reiniciar de nuevo el reloj.

En cuanto al marcador, su funcinalidad es sencilla: cuando detectamos una pulsación en determinado botón aumentamos o decrementamos el valor de su correspondiente marcador (sin poder alcanzar valores negativos). Para finalizar configuramos el botón que establece ambos marcadores a 0. Para gestionar estos marcadores debemos obtener la cadena de texto del *TextViev* correspondiente, transformarla a entero, sumar (o restar o poner 0) y modificar dicho *TextView* (formateando de manera que ambos tengan siempre 2 dígitos).

6.5. Métricas

Una vez terminado el desarrollo voy a mostrar dos datos interesantes que reflejan el tamaño, y en parte la complejidad, del proyecto desarrollado. Estso datos son el tiempo invertido y las líneas de código del total del proyecto.

En cuanto al número de horas, es el mencionado en el apartado del presupuesto real. Cabe destacar que esta cifra es aproximada teniendo en cuenta los días trabajados y la cantidad de horas de media invertidas en el proyecto (teniendo en cuenta la simultaniedad del desarrollo delproyecto con la participación en la asignatura 'Prácticas de Empresa'). Como ya se mencionó en el capítulo correspondiente, las aproximadas son **700 horas** de implementación (esto incluye la codificación, solución de errores y la búsqueda de soluciones para dicho errores).

En términos de líneas de código, utilizaré dos plugins para calulcarlo: *VS Code Counter* para Visual Studio Code (donde he programado la web con Laravel) y *Statistic* para Android Studio. Aquí sí que podemos ser más

precisos que con las horas pues son datos que se obtienen del total de archivos de ambos proyectos. Tras utilizar estos plugins, obtenemos que en el desarrollo de la web (incluye la API, relación con la base de datos,...) tenemos un total de **47.024 líneas** repartidas en 214 archivos (de las cuales, aproximadamente 33.000 son generadas de forma automática por el framework Laravel); mientras que el desarrollo de la aplicación móvil cuenta con un total de **5.231 líneas** repartidas en un total de 94 archivos. Esto son un total de **52.255 líneas** de código para desarrollar la totalidad del proyecto.

Con estos datos puedo decir orgulloso que se trata mi, hasta día de hoy, mayor proyecto en solitario; aquel en el que he puesto más esfuerzo, tiempo y ganas de aprender para demostrar parte de lo que soy capaz de hacer tras estos 4 años de estudios.

Capítulo 7

Fase de Pruebas

Este apartado viene a mostrar la forma de verificar el correcto funcionamiento del sistema desarrollado. Estas pruebas se han hechho tanto durante el desarrollo como tras finalizarlo copmletamente.

Estas pruebas las he ido realizando tras terminar cada funcionalidad para asegurarme de su correcto funcionamiento antes de avanzar hacia la siguiente. Esto me permite seguir desarrollando sin temor a que algo ya desarrollado no esté completo. En el caso de la pruebas de la web, estas eran sencillas; por ejemplo, tras desarrollar la posibilidad de crear un evento (o considerar que estaría correcta) entraba en la web (estas pruebas las hacía en un comienzo de forma local) y accedía a dicha página y lo probaba. Si algo no ssalía como debería lo descubría utilizando el inspector de elementos del navegador y la función *dd()* de PHP que me permite visuaizar el contenido de ciertas variables. Si todo parece funcionar correctamente el último paso es visitar el contenido de la base de datos usando phpMyAdmin y cerciorarme de que así es. Estas pruebas, aun siendo 'sencillas', contienen la dificultad de probar todos (o la gran parte) de los errores posibles con gran variedad de valores y opciones; es decir, sin ser un método sofisticado, te permite conocer posibles incongruencias. Para esto, en ocasiones me puse en la piel de un 'usuario tipo' que pudiese cometer determinados errores o que lo hiciese correctamente; pero además permití a amigos/familiares utilizarlo para para obtener cierto *feedback* y entender mejor como se relacionaría un usuario ajeno con el sistema.

Las pruebas para la aplicación móvil son similares. Sin embargo, en este caso sí utilicé un software: *Postman*. Esto me resulta útil a la hora de conocer cómo son las respuestas de la API de mi apliación antes de comporbarlo en la aplicación y confirmar que, cuando estas se reciban en la aplicación, lo hacen con los formatos y estructuras correctos. Este software me permite realizar peticiones tanto GET como POST con los valores que considere oportunos, lo que hace que pueda coprobar los diferentes valores de los JSON que se

devuelven de la API para asegurarme de tratarlos correctamente. Tras esto, y haber desarrollado las estrucutras necesarias de la aplicación, paso a probar la aplicación en el emulador de dispositivo Android que posee Android Studio (me permite probar la aplicación en diferentes dispositivos con diferentes tamaños sin necesidad de disponer de dispositivos reales). A la hora de ejecutarla, utilizo la consola para imprimir determinados valores y confirmar el correcto funcionamiento o, en su defecto, conocer fácilmente la razón de el error, ya sea por errores de conexión, de la información obtenida con Retrofit o de errores meramente lógicos o propios de lso elementos Android. Tras estas pruebas, por si acaso, también ejecuto la aplicación en un dispositivo real.

Las pruebas, por tanto, se basan tanto en ver el resultado de cierta función o consulta en el navegador web o la aplicación móvil, como en ir depurando los valores de determinadas variables para conocer si el proceso, hasta ese momento, se comporta como tenemos pensado.

Si queremos comprobar que podemos crear eventos desde la página web, lo que tenemos que hacer es dividir esta tarea en algunas más pequeñas e ir asegurandose de que estas se van ejecutando correctamente. Primero debemos cerciorarnos de que el botón indicado nos dirige a la pantalla de creación, luego que realmente los campos a introducir se corresponden con los que necesitamos, que los valores que nos llegan tienen sentido en su contexto, que se ha creado el evento de forma correcta y la respuesta para que el usuario lo sepa. Con esta misma filosofía nos enfrentamos a las demás funciones del sistema:

- A la hora de mostrar un evento debemos asegurarnos que sus datos son correctos (en cuanto al tipo de datos, por ejemplo) y que se muestran visiblemente correctos (no se superponen elementos en pantalla dificultando la visión); además de comprobar que elementos que dependen de otros factores se ejecutan correctamente, como los mapas de la API de Google. Es importante cerciorarse de que, según el tipo de usuario, e incluso su papel en determinado evento (si es o no creador), deben determinar las posibilidades de acción de forma correcta.

- Para la creación, visualización y alquiler de pistas y equipamiento debemos comporbar que al crearlos se dan de alta las instancias que realmente deseamos según las horas y días indicados; además de probar a alquilar una pista/equipamiento por parte de un usuario.

- En cuanto a los equipos, comprobamos en primer lugar el correcto funcionamiento de su creación y listado. Después pasamos a comprobar que dentro del equipo se listan correctamente sus integrantes, además de permitir invitar y comprobar que al invitado le llega dicha solicitud y puede aceptarla/rechazarla.

- Tanto para los grupos de conversación como para los foros hay que cerciorarse de que al enviar un mensaje se crea de forma correcta (con el emisor, grupo/foro, fecha,.. correspondientes) y que al acceder a éstos podemos verlos ordenados correctamente (con su contenido, imagen de emisor,...).

- Para el calendario hay que comprobar que se permite pasar de mes a mes, recorrer el año,... Es fundamental comprobar que los eventos se muestran correctamente (en su fecha) y que al clicar en uno se nos envía a la página del mismo.

- Tmabién hay que asegurarse de que el lista y solicitudes de amistad se gestionan y generan de forma correcta, además de la modificación del perfil del usuario.

- En cuanto al administrador, debemos asegurarnos de que todas las funciones de creación, borrado y modificación de elementos (equipos, eventos, usuarios, otros administradores,...) se desarrollan de la manera esperada haciendo diferentes comprobaciones.

- Por último, en relación con la aplicación móvil concretametne, debemos comprobar que tanto el reloj como el marcador permiten su correcto funcionamiento. Es decir, que si indico un determinado numoro de minutos sonará la alarma, que puedo comenzar, pausar y reiniciar el reloj, que los botones de aumento y decremento de marcadores funciona (sin permitir valores negativos), y que podemos también resetear correctamente dichos marcadores.

Una vez hechas las comprobaciones 'por función', comienzo a repasar el sistema por completo para tratar de encontrar posibles inconsistencias en el sistema por interferencias y dependencias entre las funciones y proceder a solucionarlas; es decir, repetir las mismas pruebas pero una vez tenemmos el sistema completo. En este caso esas dependencias no han influido de forma negativa y no se han necesitado correcciones de este tipo.

El último paso es ubicar el proyecto web en un servidor y ejecutarlo ahí. En este caso, para no tener que comprar un hosting ni dominio, utilizo un contenedor que me aportó mi tutor pra desplegar el proyecto, transfiriendolo utilizando el software *Filezilla*. Este traslado se llevó a cabo sin mayor inconveniente y con ciertas modificaciones en la configuración de Laravel; además de crear la base de datos pertinente con MySQL.

Capítulo 8

Conclusiones

La gran y principal conclusión a la que llegamos tras el desarrollo del proyecto es que se han cumplido los requisitos marcados desde el principio. Es decir, todas las funcionalidades y características del sistema ideado desde noviembre se han logrado llevar a cabo durante estos meses de trabajo.

Todas las tareas relacionadas con la gestión de eventos se recreado tal y como se idearon. Los usuarios correspondientes pueden crear y modificar eventos, apuntarse a ellos (si condiciones como la fecha o el máximo de participantes lo permite), participar en sus grupos de conversación,... Y también su relación con otros elemtos como los enfrentamientos, quienes marcan por ejemplo la situación del torneo o liga correspondiente.

Los usuarios pueden definir sus gustos temáticos en su perfil, lo que le permite filtrar eventos por esta característica. Aquí entra el único detalle no logrado respecto a lo planificado: por falta de tiempo (darle prioridad a funciones más determinantes y comprobar que estas son correctas), no se han incluido filtros por temática ni ordenación por diferentes características (como la fecha) en los listados de eventos.

La gestión de los foros también se ha llevado a cabo como estaba planteado, con un foro temática en los que solo los usuarios registrados puedan participar. Del mismo modo que el calendario individual de cada usuario muestra todos sus eventos (individuales y de sus equipos) permitiendole acceder desde ahí a la información de cada evento.

Respecto a la aplicación móvil se han cumplido igualmente las metas establecidas. El usuario individual tiene la capacidad de acceder a los diferentes evento , incluyendo los suyos propios. Puede desapuntarse de aquellos en los que ya participa y apuntarse en aquellos que se le permita del mismo modo que en la web. De forma similar cumplimos los objetivos marcados para la gestión de equipos desde la app, que consite en poder listar los equipos a los que pertence, verlos individualmente y, eventualmente, salir de ellos.

En cuanto a los grupos de conversación, como se había ideado, se accede de una forma más directa, sin tener que entrar al evento. Se permite, al igual que en la web, ver los mensajes de dicho grupo y hacer aportaciones. Por último, en lo referente a las herramientas propias de la aplicación móvil como son el reloj (con alarma) y los marcadores, permiten desarrollar su función a la perfección, con los diferentes botones y valores posibles.

Y en lo referente a temas no funcionales, cabe destacar cierto desajuste en cuanto a las horas utilizadas tanto a nivel documental como de programación, algo que incrementaría el coste del producto en comparación con el presupuesto previsto (esto se detalla en el Apéndice A-Coste Real).

Por tanto, tras repasar las condiciones iniciales y haber terminado el desarrollo, podemos concluir que el proyecto ha sido un éxito. Además, es bastante interesante la idea de tener en cuenta ciertos detalles no incluidos a priori en el proyecto que podrían dar lugar a una continuidad y extensión del desarrollo más allá del Proyecto de Fin de Grado, de forma que la plataforma creciese y fuese aún más completa.

8.1. Observaciones personales

Este proyecto es algo de lo que puedo sentirme orgulloso como programador, como estudiante, como futuro ingeniero informático y, sobretodo, como persona. He sido capaz de plantearme ciertos desafíos y cumplir con estos mediante mi esfuerzo y trabajo algo complejo en ciertos momentos. No podemos olvidar que durante este tiempo hemos estado tres meses confinados en casa por la Covid-19. Esto, sin analizarlo, podría parecer incluso positivo, pues al estar más tiempo en casa sin distracciones puedes aprovechar más el tiempo en programar; pero eso es solo ver la punta del iceberg. El estar en casa constantemente sin poder salir ni despejarte afecta a las personas, por suerte o por desgacia no somos máquinas por mucho que convivamos con ellas. Este desgaste psicológico, añadido a la situación familiar de cada uno, puede afectar considerablemente al estudiante. En mi caso, sí aproveché para tratar de trabajar más duro, pero había días en los que era realmente complicado. A esto, debo añadir una situación especialmente particular por la que he pasado al mismo tiempo que la cuarentena. Tras casi dos meses de visitar médicos, y realizarme dos colonoscopias se me diagnóstico colitis ulcerosa. Sin ser una enfermedad grave en mi caso, la incertidumbre durante meses de qué le ocurre a mi cuerpo y la preparación y desarrollo de las colonoscopias no me permitían centarme todo lo que quería en el desarrollo del Trabajo de Fin de Grado, ya sea por estrés, dolores de cabeza o necesaidad de acudir al baño numerosas veces. Pero es esto también lo que hace que le de aun más valor a este proyecto: frente a estas dificultades, y tomando conocimietos en su mayoría de forma autodidacta, he completado el sistema planteado en un principio.

Apéndice A

Coste real

Debido a ciertos desajustes temporales que se explican a continuación, el coste del proyecto se encarece teniendo en cuenta las horas de trabajo realmente utilizadas, pues éstas superan a las inicialmente planificadas.

El coste por hora que utilizaré será el mismo, y el coste de equipo y mantenimiento se conserva con respecto al presupuesto inicial, por lo que solo cambian el número de horas empleadas.

El tiempo real invertido ha sido mayor que le previsto tanto en documentación como en programación. En cuanto a la documentación, al tener que ir revisando el tutor y adaptarme a sus correciones, estas horas han aumentado desde las 136 horas planificadas a unas 180, lo que supone un gasto de 1.980 €; mientras que las horas de programación sí que se han incrementado considerablemente. Esto se debe a la cantidad de conocimientos nuevos que he tenido que ir adquiriendo y afianzando, y los diferentes errores que van surgiendo en el desarrollo e implican una gran cantidad de horas de búsquedas y pruebas. Aproximadamente han sido unas 150 horas extra, lo que serían un total de 513 horas de programación que equivalen a **5.643 €**.

Si juntamos los diferentes valores, obtenemos **7.623 €** de gasto personal y de equipo más los **75 €** anuales del servidor. La diferencia entre el gasto planificado y el real es de 2.034 €, un incremento de aproximadamente el 36 % del gasto. Esto se puede achacar en gran parte al tiempo 'perdido' en informarme sobre las nuevas tecnologías a utilizar y resolver los problemas que estas me han provocado, pues realmente a la hora de la implemenación la mayoría de herrramientas o tecnologías utilizadas no las he aprendido durante el desarrollo del grado, aunque éste sí que me ha facilitado la tarea de aprender algunas de estas, al 'abrirme la mente' y prepararme para adaptarme y aprender con cierta facilidad. Además del tiempo invertido en la corrección de la memoria y demás documentación de acuerdo a las puntualizaciones del tutor.

Bibliografía

[1] Figueroa, R. G., Solís, C. J., & Cabrera, A. A. (2008). Metodologías tradicionales vs. metodologías ágiles. Universidad Técnica Particular de Loja, Escuela de Ciencias de la Computación. Visitado en 2020.

[2] RyteWiki. Modelo en Cascada , visitado en 2020.

[3] Cliente-servidor - Wikipedia. Arquitectura Cliente-servidor , visitado en 2020.

[4] Universidad Nacional Abierta y a Distancia (UNAD). Diagramas de Casos de Uso , visitado en 2020.

[5] Framework-Wikipedia. Framework , visitado en 2020.

[6] Laravel-Wikipedia. Laravel , visitado en 2020.

[7] ¿Qué es una API?-RedHat. API , visitado en 2020.

[8] JSON-JSON. JSON , visitado en 2020.

[9] Android Studio. Android Studio , visitado en 2020.

[10] Activity. Android Studio , visitado en 2020.

[11] Timpik. Timpik , visitado en 2020.

[12] Meetup. Meetup , visitado en 2020.

[13] Presupuesto Odenador. PcComponentes, visitado en 2020.