

## Trabajo final de grado

**Estudios:** Grado en Ingeniería Informática

**Título:** Aplicación móvil para una red social deportiva

**Documento:** Memoria trabajo final de grado

**Alumno:** Daniel Bermudez Rodriguez

**Tutor:** Gustavo Ariel Patow

**Departamento:** Informática, Matemática Aplicada y Estadística

**Área:** Lenguajes y Sistemas Informáticos

**Convocatoria (mes/año)** Septiembre/2018



---

## Contenido

---

1.	Introducción, motivaciones, propósito y objetivos del proyecto .....	6
1.1	Introducción.....	6
1.2	Motivaciones.....	7
1.3	Propósito.....	7
1.4	Objetivos del proyecto.....	7
1.5	Estructura del documento .....	8
2.	Estudio de viabilidad .....	10
2.1	Recursos técnicos.....	10
2.2	Recursos humanos.....	11
2.3	Coste económico.....	11
2.4	Conclusión viabilidad del proyecto .....	13
3.	Metodología.....	14
4.	Planificación .....	18
4.1	Planificación inicial.....	18
4.2	Planificación final .....	21
5.	Marco de trabajo y conceptos previos .....	23
5.1	Introducción sistemas operativos móviles.....	23
5.2	Ejemplos sistemas operativos móviles .....	24
5.3	Sistemas operativos móviles en España .....	27
5.4	Sistema operativo utilizado.....	27
6.	Requisitos del sistema.....	29
6.1	Requisitos funcionales .....	29
6.2	Requisitos no funcionales .....	32
7.	Estudio y decisiones .....	34
7.1	Aplicación servidor.....	34
7.1.1	Java Development Kit (JDK).....	34
7.1.2	Servidor aplicaciones Wildfly 9 .....	35
7.1.3	Java EE.....	35
7.1.4	JAX-RS.....	37
7.1.5	RESTEasy .....	39
7.1.6	Enterprise Java Beans .....	39
7.1.7	Java Persistence Api .....	40
7.1.8	Hibernate .....	41
7.1.9	MySQL .....	44

7.1.10	Eclipse .....	45
7.1.11	Postman .....	45
7.1.12	JSON .....	47
7.2	Aplicación cliente .....	48
7.2.1	Android.....	48
7.2.2	Android Studio .....	54
7.2.3	Retrofit + GSON.....	56
7.2.4	Glide .....	57
7.2.5	Firebase.....	58
7.2.6	Firebase Realtime Database.....	60
7.2.7	Firebase Cloud Messaging.....	62
7.3	Otros.....	63
7.3.1	GanttProject .....	64
7.3.2	Visual Paradigm .....	64
7.3.4	Microsoft Word .....	65
8.	Análisis y diseño del sistema.....	66
8.1	Análisis de los requisitos del sistema.....	66
8.1.1	Actores del sistema.....	66
8.1.2	Partes del sistema .....	66
8.1.3	Diagramas y fichas de casos de uso .....	68
8.1.3.1	Registro .....	68
8.1.3.2	Imagen de perfil .....	69
8.1.3.3	Iniciar sesión.....	70
8.1.3.4	Principal.....	71
8.1.3.5	Cerrar sesión .....	72
8.1.3.6	Crear evento deportivo .....	73
8.1.3.7	Buscar eventos deportivos.....	74
8.1.3.8	Ver mis eventos.....	76
8.1.3.9	Ver detalle evento deportivo .....	78
8.1.3.10	Modificar evento.....	81
8.1.3.11	Ver perfil usuario .....	82
8.1.3.12	Modificar perfil .....	83
8.1.3.13	Administrar notificaciones.....	84
8.2	Diseño del sistema .....	85
8.2.1	Modelo entidad-relación .....	85
8.2.2	Diagrama de clases modelo de datos .....	88

8.2.3	Diseño interfaz usuario .....	89
8.3	Diseño API REST .....	97
8.3.1	Recurso usuario.....	98
8.3.2	Recurso ubicación .....	104
8.3.3	Recurso país .....	105
8.3.4	Recurso provincia.....	106
8.3.5	Recurso municipio.....	106
8.3.6	Recurso deporte.....	107
8.3.7	Recurso evento .....	107
8.3.8	Recurso participante .....	113
8.3.9	Recurso notificación.....	115
8.3.10	Recurso imagen.....	116
9.	Implementación y pruebas .....	119
9.1	Aplicación servidor.....	119
9.1.1	Estructura del proyecto.....	119
9.1.3	Modelo de clases.....	140
9.1.4	Buscador eventos .....	141
9.1.5	Tratamiento imágenes .....	143
9.2	Aplicación Android.....	146
9.2.1	Estructura del proyecto .....	146
9.2.2	Implementación Retrofit .....	151
9.2.3	Listar eventos deportivos .....	154
9.2.4	Implementación del foro.....	159
9.2.5	Manejo notificaciones .....	161
9.2.6	Ubicación GPS.....	162
9.3	Pruebas .....	164
10.	Implantación y resultados.....	165
10.1	Resultados en función de los objetivos.....	165
10.2	Validez legal de la aplicación.....	195
11.	Conclusiones .....	196
12.	Trabajo futuro .....	198
13.	Bibliografía .....	199
14.	Anexos.....	202

## 1. Introducción, motivaciones, propósito y objetivos del proyecto

### 1.1 Introducción

Por salud, ocio o hábito ha crecido en España la población que realiza deporte. Según la publicación elaborada por la Subdirección General de Estadística y Estudios de la Secretaría General Técnica del Ministerio de Educación, Cultura y Deporte [1] en la encuesta de hábitos deportivos en España en 2015, los resultados indican que el 53,5% de la población de 15 años en adelante practicó deporte en el último año, lo que supone un incremento del 9.2% sobre la encuesta anterior de 2010. La mayor parte de ellos, el 86,3% con gran intensidad, al menos una vez a la semana.

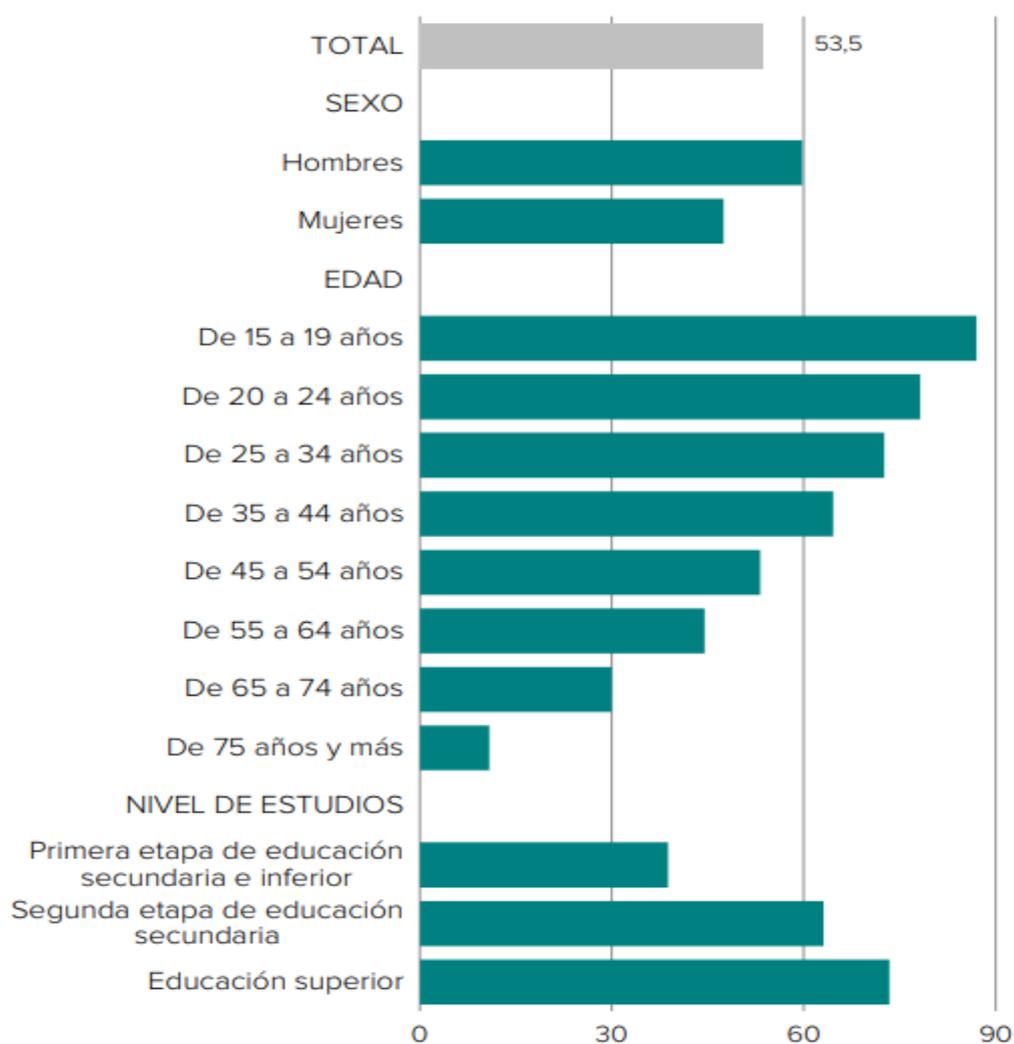


Figura 1.1: Personas que practicaron deporte en el último año según sexo, edad y nivel de estudios. 2015  
(En porcentaje de la población total investigada de cada colectivo)

Por otro lado, a día de hoy el teléfono móvil se ha convertido en una herramienta indispensable y de uso cotidiano en nuestras vidas. Como se ha mencionado anteriormente, muchas personas son aficionadas a la práctica de algún deporte, y actualmente existe un sinnúmero de aplicaciones móviles destinadas a la monitorización de la práctica deportiva. Sin embargo, en ocasiones la práctica de deporte o la motivación se encuentran limitadas no por motivos técnicos, sino debido a la ausencia de compañeros con los cuales realizar dicha actividad o compartir una afición.

En este proyecto, se pretende aprovechar el gran uso y potencial del teléfono móvil para desarrollar una aplicación de tipo red social deportiva, mediante la cual poder compartir y realizar las prácticas deportivas preferidas entre los usuarios.

## 1.2 Motivaciones

La idea de realizar este proyecto surgió principalmente por dos motivos.

En primer lugar, como persona que practica diferentes modalidades deportivas con frecuencia, en ocasiones, esta práctica se ha visto limitada o incapacitada por el simple hecho de no contar con las suficientes personas o por pereza de realizarla de forma individual. Actualmente, utilizo diversas aplicaciones móviles de monitorización del ejercicio, pero he echado en falta aplicaciones móviles con las cuales poder gestionar y planificar quedadas deportivas en grupo.

Por otro lado, a nivel académico realizar este proyecto me sirve como aprendizaje de como diseñar y desarrollar una aplicación móvil para Android. El hecho de desarrollar desde cero una aplicación de manera propia, engloba una gran cantidad de campos de conocimiento de la informática, bases de datos, ingeniería del software, comunicación entre cliente-servidor, etc.

## 1.3 Propósito

El principal propósito para el desarrollo de esta aplicación es la de proporcionar a los usuarios una aplicación móvil de tipo red social destinada exclusivamente a la práctica de deporte en grupo, mediante la gestión de eventos deportivos, adaptándose a las preferencias y disponibilidades horarias de los diferentes usuarios.

Básicamente, se pretende aprovechar el éxito de las redes sociales y los dispositivos móviles focalizando su uso en la práctica de disciplinas deportivas.

## 1.4 Objetivos del proyecto

El principal objetivo de este proyecto es el desarrollo de una aplicación móvil que correrá sobre un sistema operativo Android, y su correspondiente servidor de administración, que permita la creación y gestión de actividades deportivas, funcionando como una red social que permita encontrar personas con las cuales poder compartir la práctica de deporte. Mediante la aplicación, los usuarios registrados podrán crear sus propios eventos deportivos personalizados, definiendo la modalidad deportiva, adaptar el evento a su disponibilidad horaria y localización

entre otros aspectos. Por otro lado, al margen de gestionar eventos propios, se podrán buscar eventos en los cuales inscribirse. Se pretende diseñar una interface de usuario que sea cómoda, intuitiva y sencilla de utilizar por parte de los usuarios finales. Por lo tanto, partiendo completamente de cero, las actividades principales a realizar son:

- Análisis, diseño e implementación de una aplicación en Android, y de las diferentes librerías y framework a utilizar.
- Análisis, diseño e implementación de una aplicación servidor en Java EE, y de las diferentes librerías y framework a utilizar.
- Análisis, diseño e implementación de una base de datos para almacenar la información del sistema.
- Análisis, diseño e implementación de un protocolo de intercambio de datos entre las aplicaciones cliente y servidor mediante una API REST.
- Finalización y estructuración de la documentación del proyecto desarrollado.

## 1.5 Estructura del documento

La memoria se encuentra estructurada en los siguientes capítulos:

- **Capítulo 1. Introducción, motivaciones, propósito y objetivos del proyecto.** En este capítulo se explican las motivaciones, los propósitos y los objetivos del proyecto acompañados de una introducción. También se presenta un resumen de los diferentes capítulos de la memoria.
- **Capítulo 2. Estudio de viabilidad.** Se realiza un análisis y estudio de la viabilidad del proyecto. Analizando los recursos tecnológicos, recursos humanos y realizando un estudio del coste total de la realización del proyecto. Finalmente, se justifica la viabilidad del proyecto.
- **Capítulo 3. Metodología.** Se define y justifica la metodología utilizada para el desarrollo del proyecto.
- **Capítulo 4. Planificación.** Se define la planificación de trabajo, las tareas planificadas y el tiempo estimado en una planificación inicial. Se realiza una comparación y justificación de las diferencias entre la planificación inicial y la final.
- **Capítulo 5. Marco de trabajo y conceptos previos.** Se introducen los conceptos teóricos básicos necesarios para una mejor comprensión del resto de la memoria.
- **Capítulo 6. Requisitos del sistema.** Se describen los requisitos funcionales y no funcionales que tiene que cumplir el sistema para poder cumplir todos los objetivos desde el punto de vista de software y de hardware.

- **Capítulo 7. Estudios y decisiones.** Definición de todas las herramientas utilizadas durante el desarrollo del proyecto. Se categorizan las herramientas en 3 bloques. Las utilizadas en la aplicación servidor, aplicación cliente y las destinadas a redactado de la memoria.
- **Capítulo 8. Análisis y diseño del sistema.** En este capítulo se detalla el análisis y diseño realizado para satisfacer los objetivos en función de los requerimientos del proyecto. Se presenta el análisis con diagramas y fichas de caso de uso por cada funcionalidad del sistema. Por otro lado, en cuanto al diseño se presenta el modelo físico de datos, el diseño de la API REST y las maquetas de las diferentes pantallas disponibles en la aplicación.
- **Capítulo 9. Implementación y pruebas.** En este capítulo se define el proceso de construcción de la aplicación. Se define la estructura del proyecto de ambas aplicaciones junto a los algoritmos de las funcionalidades más relevantes del desarrollo de la aplicación. Finalmente, se define el proceso de pruebas realizado durante la implementación.
- **Capítulo 10. Implantación y resultados.** En este capítulo se muestran los resultados obtenidos a través de imágenes y ejemplos del uso de la aplicación. Por otro lado, se comenta la validez legal de la aplicación en función de la LOPD y la LLSICE.
- **Capítulo 11. Conclusiones.** En este capítulo se exponen las conclusiones en función del grado de asimilación de los requisitos y objetivos del proyecto.
- **Capítulo 12. Trabajo futuro.** Se detallan las posibles mejoras, ampliaciones y trabajos futuros que se pueden realizar.
- **Capítulo 13. Bibliografía.** Se muestra el listado de las referencias utilizadas para la realización del proyecto y la documentación.
- **Capítulo 14. Anexos.** Se muestra material de soporte utilizado para el desarrollo de la aplicación. En este caso, se muestran los scripts SQL utilizados para la inserción de los datos utilizados.

---

## 2. Estudio de viabilidad

---

En este capítulo se valorará la viabilidad del proyecto teniendo en cuenta diferentes aspectos como los recursos técnicos, los recursos humanos, los requisitos tecnológicos y el coste económico.

### 2.1 Recursos técnicos

Para el desarrollo del proyecto principalmente se han utilizado dos dispositivos:

- **Dispositivo móvil:** Su función principal es la de instalar y ejecutar la aplicación cliente desarrollada, comunicándose con el servidor a través de la API REST. En concreto, se ha utilizado el teléfono LG X Power que dispone de las siguientes características a destacar:
  - pantalla IPS de 5,3 pulgadas.
  - procesador MediaTek MT6735 con cuatro núcleos a 1.3 GHz.
  - 2 Gb de memoria RAM.
  - Sistema operativo Android 6.0.1 Marshmallow.
  
- **Ordenador portátil:** Su función principal es la de actuar como servidor, el cual aloja la aplicación servidor encargada de gestionar las peticiones REST recibidas en la API diseñada por parte de las aplicaciones clientes. Concretamente, se ha utilizado el modelo Acer Aspire v5-552G con las siguientes características:
  - Sistema operativo Windows 10 de 64 bits.
  - 8 Gb de memoria RAM.
  - Procesador AMD A10-5757M, 2500Mhz y 4 procesadores.

Para la instalación de la aplicación cliente en el dispositivo móvil se ha utilizado un cable USB, de esta forma, conectando el móvil al ordenador habilitando previamente la depuración USB nos permite instalar la aplicación Android en el móvil.

En cuanto a los requisitos tecnológicos utilizados serán definidos en el *capítulo 7*.

## 2.2 Recursos humanos

Durante el desarrollo de la aplicación, se han realizado diversas tareas que podríamos categorizar en 3 roles distintos:

- **Jefe de proyecto:** Su principal función es la de definir una metodología de trabajo a llevar a cabo durante el desarrollo de la aplicación. Se encargará de coordinar y dirigir el proyecto realizando el seguimiento de las distintas tareas a realizar. Es el responsable de que la implementación de los requerimientos definidos, ya sea en la etapa de desarrollo o redacción de la memoria, se realice correctamente.

En este proyecto el rol de jefe de proyecto principalmente ha sido asumido por el tutor.

- **Analista:** Es el encargado de definir y diseñar la aplicación a nivel técnico y especificar las distintas funcionalidades y como se tendrán que llevar a cabo. Responsable de analizar y escoger las herramientas más adecuadas para la elaboración del proyecto. Encargado de redactar la memoria del proyecto bajo la supervisión del jefe de proyecto.
- **Programador:** Se encarga de ejecutar el trabajo asignado por el analista utilizando las distintas herramientas asignadas.

Este proyecto ha sido realizado por la misma persona que ha tenido que asumir los roles de analista y programador, yo mismo.

## 2.3 Coste económico

Para definir el coste económico del proyecto se han tenido en cuenta los recursos técnicos y humanos mencionados en los apartados anteriores.

En cuanto a los recursos técnicos, no se ha realizado ninguna inversión ya que se disponía de ambos con anterioridad a la iniciación del proyecto y no ha hecho falta compra adicional de ningún hardware específico.

No obstante, para la contabilización del coste económico tendremos en cuenta la amortización del hardware. La fórmula utiliza es la siguiente:

$$\text{amortización} = \frac{\text{Coste Hardware}}{\text{Tiempo de amortización}} * \text{Tiempo del proyecto}$$

Considerando:

- **Tiempo de amortización:** 36 meses.
- **Tiempo del proyecto:** 7 meses.
- **Coste dispositivo móvil:** 170 €
- **Coste ordenador portátil:** 565 €

Obtenemos:

- **Coste amortización móvil:** 33 €
- **Coste amortización ordenador:** 110 €

Por lo tanto, consideraremos un coste total de 143 € en amortización de hardware.

En el caso de los recursos humanos, se ha realizado una clasificación de las tareas realizadas por los roles de programador y analista. Como se ha mencionado anteriormente, los dos roles en este proyecto han sido llevado a cabo por una misma persona, pero a la hora de realizar el coste económico se contabiliza el precio/hora en función del rol desempeñado en la tarea en cuestión.

Se ha clasificado el precio/hora de cada rol de la siguiente manera:

- **Analista:** 18 €/h
- **Programador:** 9€/h

Las diferentes tareas a contabilizar son:

- Formación en programación Android
- Estudio y diseño de la aplicación servidor
- Estudio de las herramientas a utilizar para la aplicación servidor
- Estudio y diseño de la aplicación Android
- Estudio de las herramientas a utilizar para la aplicación Android
- Implementación aplicación servidor
- Implementación aplicación Android
- Pruebas
- Realización documentación

En la siguiente tabla podremos visualizar el detalle del coste económico realizado para la elaboración del proyecto:

<i>Tareas</i>	<i>Rol</i>	<i>Horas</i>	<i>Precio/hora</i>	<i>Coste total</i>
Formación programación Android	Programador	30	9	270 €
Implementación aplicación servidor	Programador	100	9	900 €
Implementación aplicación Android	Programador	130	9	1.170 €
Pruebas	Programador	40	9	360 €
Estudio y diseño de la aplicación servidor	Analista	50	18	900 €
Estudio de las herramientas a utilizar para la aplicación servidor	Analista	40	18	720 €
Estudio y diseño de la aplicación Android	Analista	70	18	1.260 €
Estudio de las herramientas a utilizar para la aplicación Android	Analista	70	18	1.260 €
Documentación	Analista	90	18	1.620 €
Amortización Hardware				143 €
<b>Total</b>				<b>8.603 €</b>

## 2.4 Conclusión viabilidad del proyecto

Una vez realizado el análisis de la viabilidad del proyecto, se decide su realización por lo siguiente motivos:

- No supone ningún tipo de inversión hardware ya que se disponía previamente de todo el material necesario para su realización. Tan solo contamos los gastos en amortización del material.
- En cuanto al software y herramientas utilizadas no se ha requerido ningún gasto adicional en licencias de software.
- En referencia a los recursos humanos, el estudio se ha realizado en el caso hipotético de una contratación real, pero como se ha comentado anteriormente, en este proyecto los roles de programador y analista han sido llevado a cabo por la misma persona. Por lo tanto, el coste real ha sido nulo.

Por lo tanto, se concluye en que la realización del trabajo es viable.

---

### 3. Metodología

---

La metodología de desarrollo de software consiste en un marco de trabajo utilizado para estructurar, planificar y controlar el proceso de desarrollo. Antes de iniciar el desarrollo del proyecto, se ha realizado un estudio de las diferentes metodologías existentes. Finalmente, se ha decidido utilizar una metodología propuesta por el jefe de proyecto. Ajustándola a las características y necesidades del proyecto a realizar.

Los pasos de la metodología definida son los siguientes:

1. Escoger el trabajo que queremos realizar.
2. Decidir lenguajes de programación y herramientas a utilizar.
3. Estudio de los lenguajes de programación escogidos y del funcionamiento de las herramientas a utilizar.
4. División y estructuración del trabajo a realizar en diferentes partes. Realizando una división en partes, identificando previamente diferentes bloques funcionales, que permita su desarrollo de forma separada a las otras partes.
5. Desarrollar la primera parte correspondiente al orden diseñado en la estructuración realizada en el paso anterior.
6. Realizar pruebas para confirmar si la parte se ha desarrollado correctamente:
  - En caso de que la parte desarrollada no supere las comprobaciones realizadas. Se vuelve al paso 5 para realizar las modificaciones necesarias sobre la parte desarrollada o en anteriores partes si es conveniente.
  - En caso de superar con éxito las comprobaciones realizadas sobre la parte desarrollada tenemos dos opciones:
    - Si quedan partes por desarrollar volvemos al paso 5.
    - Si se han finalizado todas las partes se inicia el paso 7.
7. Unión de todas las partes desarrolladas y realizar las pruebas de comprobación de la unión.
  - En caso de que la unión de las partes no funcionase como se esperaba, se vuelve al paso 5 para realizar las modificaciones necesarias para solucionar los problemas derivados de la unión de las partes.

- Si la unión se realiza con éxito se inicia el paso 8.
8. Generar diferentes modelos de ejemplo a partir de la unión de las partes y comprobar su funcionamiento.
    - En caso de que las pruebas realizadas sobre el modelo generado, se vuelve al paso 5 para realizar las modificaciones necesarias.
    - Si los resultados son satisfactorios accedemos al paso 9.
  9. Realizar documentación del proyecto.

En la figura 3.1 se muestra el diagrama en función de la metodología descrita.

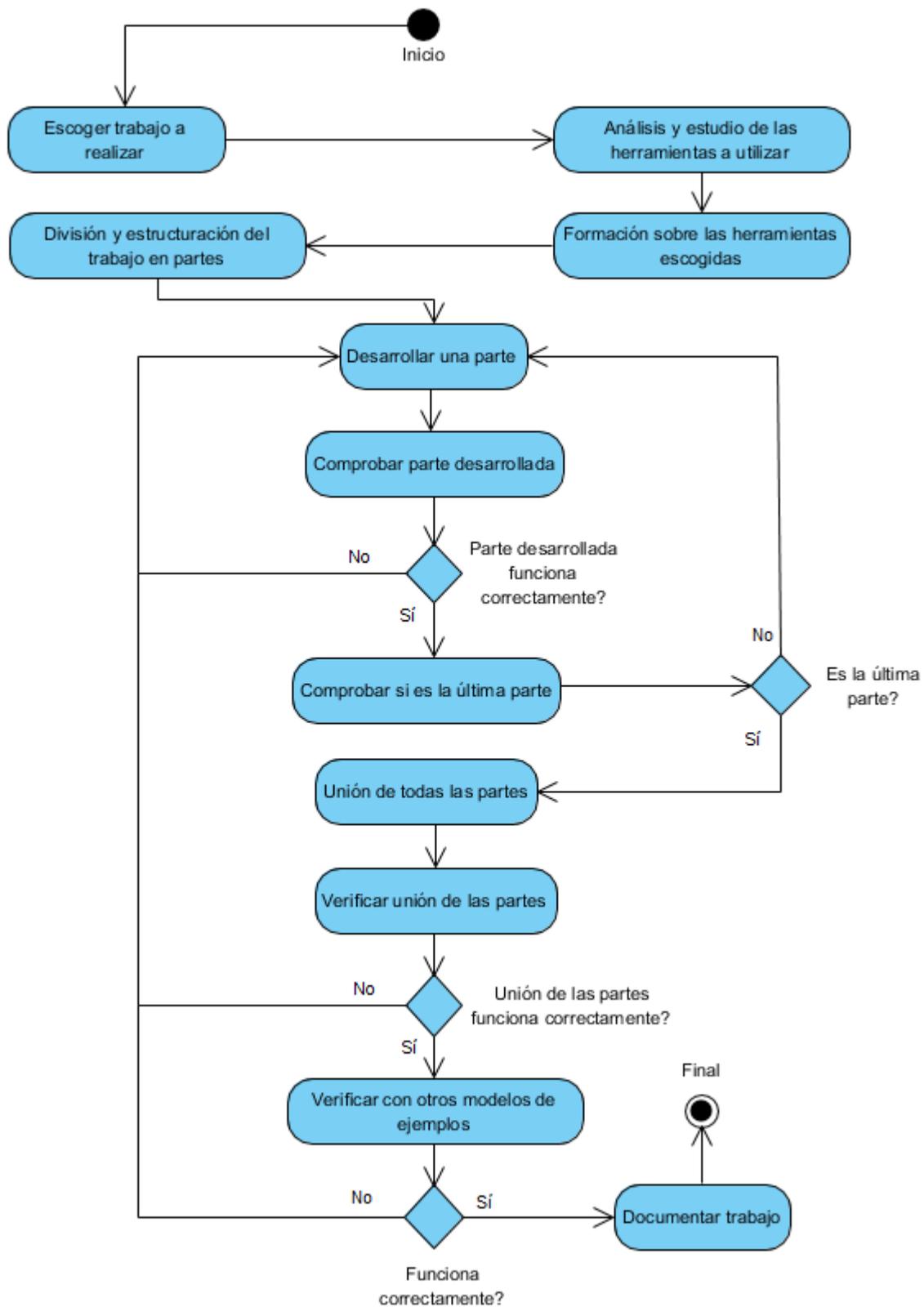


Figura 3.1: Diagrama metodología utilizada

En el momento en que se decidió y definió la idea de proyecto a realizar, esta fue expuesta al jefe de proyecto y posteriormente aprobada. Basándonos en las características del proyecto, se decidió utilizar la metodología expuesta anteriormente, ya que se identificaron diversos módulos funcionales que podían ser desarrollados de forma individual para su posterior unión. Facilitando de esta manera el cumplimiento de los objetivos ya que la división del desarrollo en módulos facilita la organización y comprensión desde el punto de vista humano. Utilizando esta metodología nos resulta más fácil detectar errores, corregirlos y tener una visión de conjunto de la estructura y funcionamiento de los módulos. Aproximadamente cada 3 semanas se han ido realizando reuniones con el jefe de proyecto.

---

## 4. Planificación

---

### 4.1 Planificación inicial

La planificación de este proyecto ha estado influenciada por la metodología de trabajo seleccionada. La primera reunión con el jefe de proyecto tuvo lugar la primera semana de febrero de 2018, en el cual se decidió de forma definitiva el desarrollo de la aplicación móvil basada en una red social deportiva.

Como que nunca se había desarrollado previamente una aplicación móvil de Android, una primera parte de la planificación del trabajo se destinó al estudio y familiarización de las diferentes estructuras y componentes básicos utilizados en Android. Por otro lado, se estudió el ciclo de vida de una aplicación de este tipo y el diseño de interfaces de usuario.

Una vez confirmada la idea de proyecto, realizada la formación inicial en Android y decidida la metodología de trabajo, se diseña la planificación de trabajo siguiente:

- 1 Formación programación aplicaciones móvil en Android.
- 2 Análisis y estudio de las herramientas a utilizar para el desarrollo del proyecto.
- 3 Identificar, dividir y estructurar los diferentes módulos funcionales de la aplicación.
- 4 Instalación y configuración entorno de trabajo.
- 5 Diseño e implementación del modelo de datos a utilizar.
- 6 Diseño de la API REST.
- 7 Implementación de los módulos funcionales (Implementación API REST en la aplicación servidor, diseño e implementación interface de usuario y programar funcionalidad en la aplicación cliente y realizar pruebas de funcionamiento):
  - 7.1 Iniciar sesión
  - 7.2 Cerrar sesión
  - 7.3 Registrar usuario
  - 7.4 Imagen perfil usuario
  - 7.5 Modificar datos usuario
  - 7.6 Obtener información usuario
  - 7.7 Obtener ubicación GPS usuario
  - 7.8 Crear evento deportivo
  - 7.9 Imagen evento deportivo
  - 7.10 Modificar evento deportivo
  - 7.11 Buscador de eventos deportivos
  - 7.12 Obtener información evento
  - 7.13 Obtener eventos creados por un usuario
  - 7.14 Obtener eventos apuntados de un usuario
  - 7.15 Crear foro en un evento deportivo
  - 7.16 Cancelar evento deportivo

- 7.17 Registrar usuario en un evento deportivo
- 7.18 Dar de baja un usuario de un evento deportivo
- 7.19 Gestionar cola de participantes en un evento completo
- 7.20 Sistema de notificaciones push

- 8 Pruebas generales del proyecto
- 9 Documentar el proyecto durante el desarrollo.

En un principio, se distribuyó la planificación de trabajo a ser realizada en 5 meses, desde febrero de 2018 a junio de 2018 tal y como se puede observar en la figura 4.1.

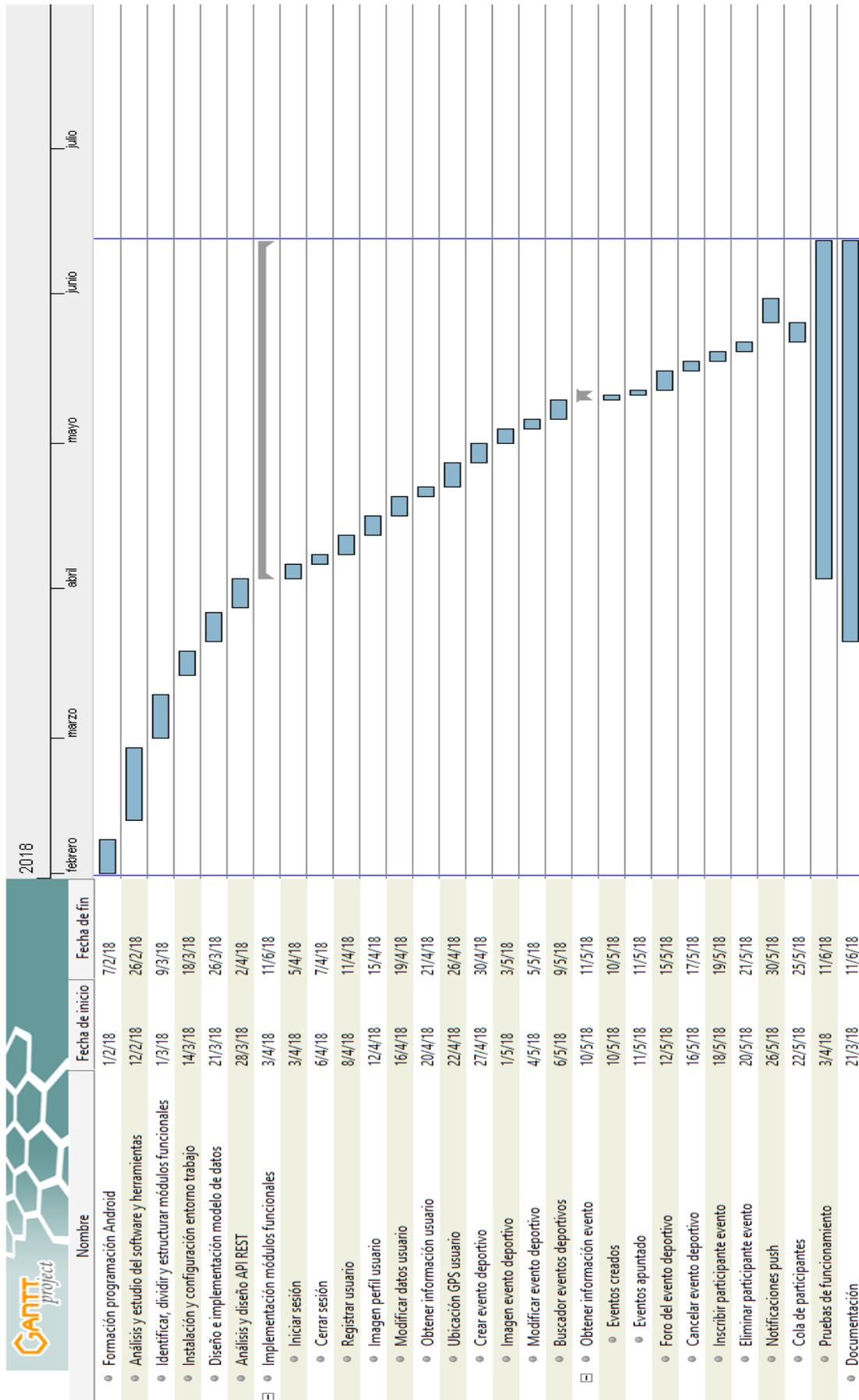


Figura 4.1: Diagrama de Gantt de la planificación inicial

## 4.2 Planificación final

Pese a que la planificación inicial diseñada, normalmente durante el desarrollo de un proyecto de software esta sufre variaciones, tareas que se alargan o acortan su duración, se añaden o suprimen tareas e incluso el período de entrega puede retrasarse. Como se ha visto en el apartado anterior, se planificó desarrollar el proyecto en 5 meses para de esta forma poder entregar y exponer el proyecto en la convocatoria de junio. Por motivos laborales, en el mes de mayo se decidió modificar la planificación inicial atrasando el periodo de entrega al mes de septiembre. De esta forma, se disponía de un total de 7 meses para la realización del proyecto.

La principal razón, como he nombrado anteriormente, se debe a motivos laborales que impedían dedicar el volumen de horas diarias estimadas en la planificación inicial.

Esto sumado a los típicos contratiempos producidos en el desarrollo de una aplicación y la aparición de diversas sub-tareas que en un principio se englobaron como una sola tarea, propició la decisión de atrasar la entrega y poder realizar el proyecto sin prisas, dedicando mayor atención al desarrollo y testeo de los módulos funcionales.

Se consideró también el realizar un sobreesfuerzo para poder terminar el proyecto como se había indicado en la planificación inicial, pero esto podría afectar a la calidad del desarrollo del proyecto por lo cual se acabó descartando.

En la figura 4.2 se muestra el diagrama de Gantt de la planificación final.

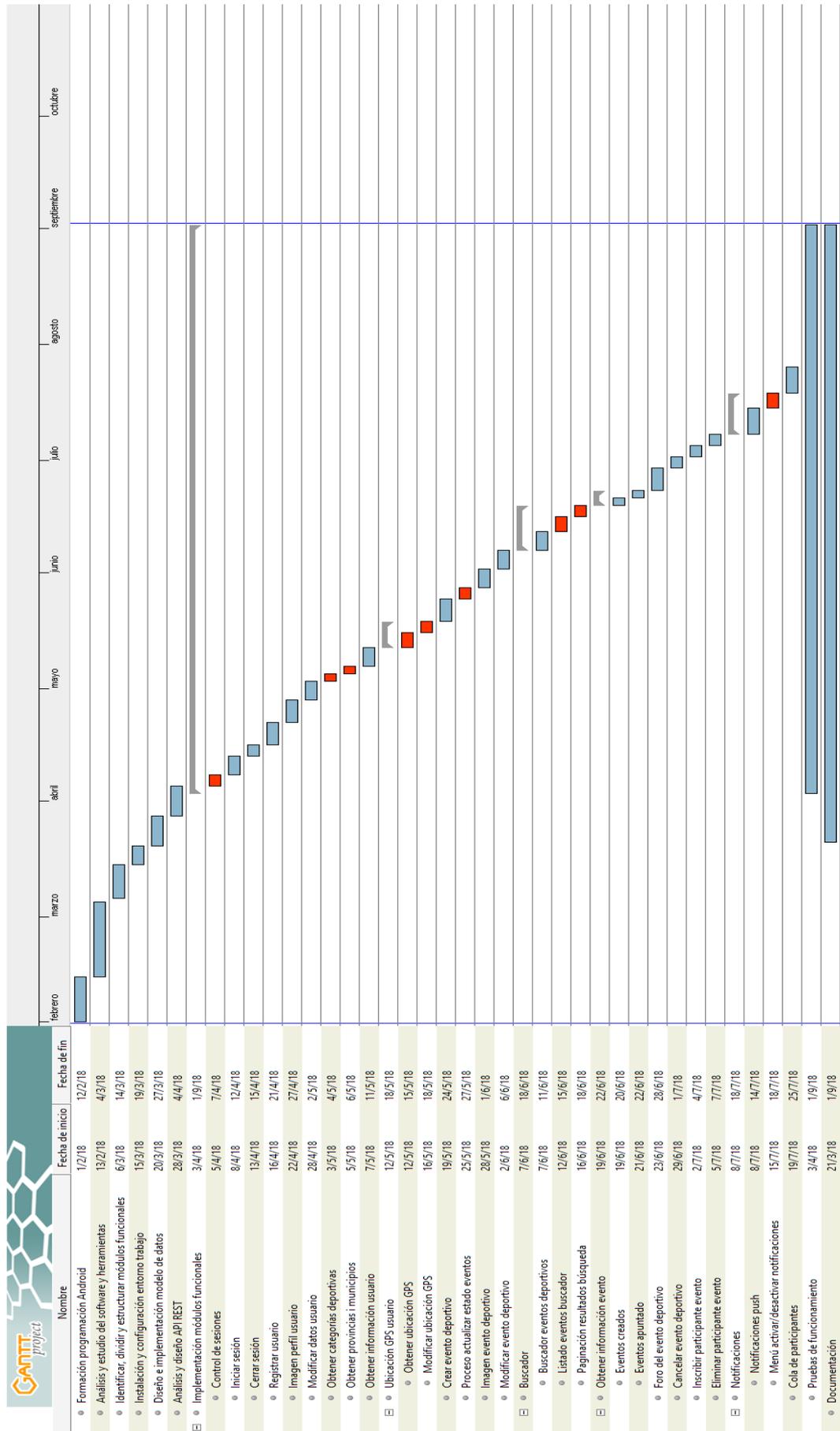


Figura 4.2: Diagrama de Gantt de la planificación final

---

## 5. Marco de trabajo y conceptos previos

---

En este apartado se explicará el trabajo realizado durante la etapa de análisis y estudio del proyecto y se introducirán una serie de conceptos previos para un mayor entendimiento del desarrollo del proyecto.

Una vez se decidió que se realizaría una aplicación para dispositivos móviles, antes de entrar en el diseño interno del desarrollo, había que escoger sobre qué plataforma se implementaría el proyecto y saber cómo funciona internamente.

### 5.1 Introducción sistemas operativos móviles

Un sistema operativo móvil consiste en un conjunto de programas de bajo nivel que permite la abstracción de las peculiaridades del hardware específico del teléfono móvil y provee servicios a las aplicaciones móviles, que se ejecutan sobre él. Se encuentran orientados hacia la conectividad inalámbrica.

La mayoría de los sistemas operativos utilizados en la actualidad se estructuran en capas [2]:

- **Kernel:** El núcleo proporciona el acceso a los distintos elementos del hardware del dispositivo móvil. Ofrece distintos servicios a las capas superiores como los controladores para el hardware, la gestión de procesos, el sistema de archivos y el acceso y gestión de la memoria.
- **Middleware:** También conocido como lógica de información entre aplicaciones, es el software que proporciona un enlace entre aplicaciones de software independientes. Hace posible la propia existencia de aplicaciones para móviles. Es totalmente transparente al usuario y ofrece servicios como el motor de mensajería y comunicaciones, gestión del dispositivo, seguridad, códecs multimedia...
- **Entorno de ejecución de aplicaciones:** Consiste en un gestor de aplicaciones y un conjunto de interfaces programables (APIs) abiertas para facilitar la creación de software a los desarrolladores.
- **Interfaz de usuario:** Facilitan la interacción con el usuario y el diseño de la presentación visual de la aplicación. Incluye los servicios de componentes gráficos y el del marco de interacción.

La figura 5.1.1 muestra una visión simplificada de la pila de software definida anteriormente, que conforman el marco de trabajo para los desarrolladores de aplicaciones para dispositivos móviles. Sobre estas capas, descansa y se ejecuta cualquier aplicación de un dispositivo móvil.

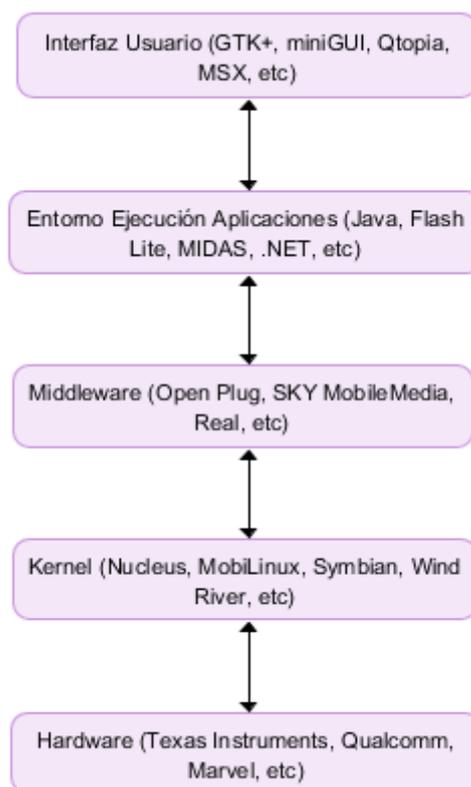


Figura 5.1.1: Estructura en capas de un sistema operativo móvil

## 5.2 Ejemplos sistemas operativos móviles

A continuación, se introducen algunos sistemas operativos móviles analizados.

### Windows 10 Mobile



Figura 5.2.1: Logo Windows 10 Mobile

Windows 10 Mobile (anteriormente llamado Windows Phone) [3], se trata de un sistema operativo propiedad de Microsoft, diseñado para teléfonos inteligentes y tabletas. Es de código cerrado y propietario y utiliza como núcleo Windows NT.

En febrero de 2010 se dio a conocer Windows Phone que integra servicios de Microsoft como OneDrive, juegos Xbox Live y Bing. Pero también se integra con otros servicios que no son de su propiedad, como Facebook y cuentas de Google.

A principios de 2015, Microsoft anunció que la marca Windows Phone sería reemplazada por Windows 10 Mobile con el objetivo de lograr una mayor integración y unificación con su homólogo para PCs Windows 10. Está diseñado para ser similar a las versiones de escritorio de Windows estéticamente.

Se encuentra programado en .NET, C#, VB.NET, Silverlight, native C/C++, WinRTP (XMLA) y DirectX.

## Android



Figura 5.2.2: Logo sistema operativo Android

El sistema operativo Android es el más utilizado en el mercado de dispositivos móviles, está basado en Linux, diseñado originalmente para cámaras fotográficas profesionales, luego fue vendido a Google y modificado para ser utilizado en dispositivos móviles.

Las aplicaciones se desarrollan habitualmente en el lenguaje Java con Android Software Development Kit (Android SDK), pero están disponibles otras herramientas de desarrollo. El desarrollo de aplicaciones para Android no requiere aprender lenguajes complejos de programación. Todo lo que se necesita es un conocimiento aceptable de Java y estar en posesión del kit de desarrollo de software o SDK, provisto por Google, el cual se puede descargar gratuitamente.

Google Play es la tienda y plataforma en línea de software desarrollado por Google para dispositivos Android. Google retribuye aproximadamente el 70% del precio de las aplicaciones. El desarrollo en Android actualmente requiere usar el IDE Android Studio, basado en IntelliJ, ya que Google ha dejado de dar soporte al IDE Eclipse.



Figura 5.2.3: Logo sistema operativo iOS

iOS es un sistema operativo móvil de la multinacional Apple Inc. Originalmente desarrollado para el iPhone, posteriormente se ha utilizado en dispositivos como el iPod Touch y el iPad. No permite la instalación de iOS en hardware de terceros.

Actualmente, es el segundo sistema operativo móvil más utilizado del mundo.

En el año 2008 fue liberado el Kit de desarrollo de software SDK, permitiendo así a los desarrolladores hacer aplicaciones para iPhone y iPod Touch, así como probarlas en el “iPhone simulator”.

El lenguaje principal para iPhone OS, al igual que en Mac OS, es Objective-C/Swift.

Los desarrolladores pueden poner un precio por encima del mínimo (\$0.99 dólares) a sus aplicaciones para distribuir las en la App Store, de donde recibirán el 70% del dinero que produzca la aplicación. Como alternativa, se puede optar por ofrecer las aplicaciones de forma gratuita y no realizar ningún pago por la distribución más allá de la cuota de socio.

## 5.3 Sistemas operativos móviles en España

En primer lugar, como la aplicación desarrollada está destinada para uso exclusivo en el territorio español, se realizó un estudio de los diferentes sistemas operativos móviles utilizados en España en los últimos años. Ver figuras 5.3.1 y 5.3.2. [4]

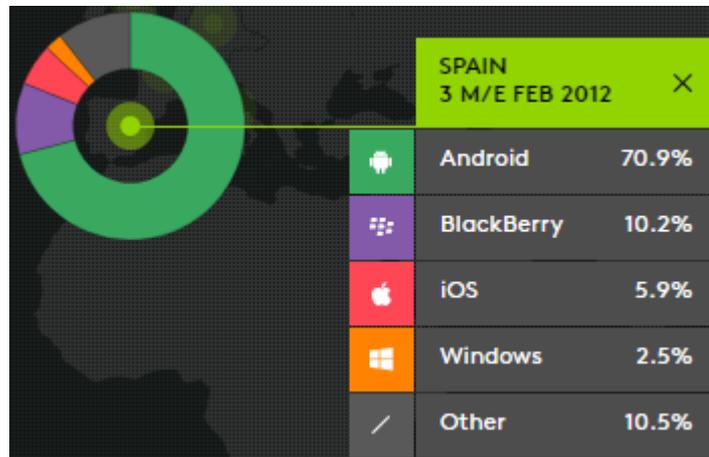


Figura 5.3.1: Cuota de mercado sistemas operativos móviles en España. (febrero, 2012)

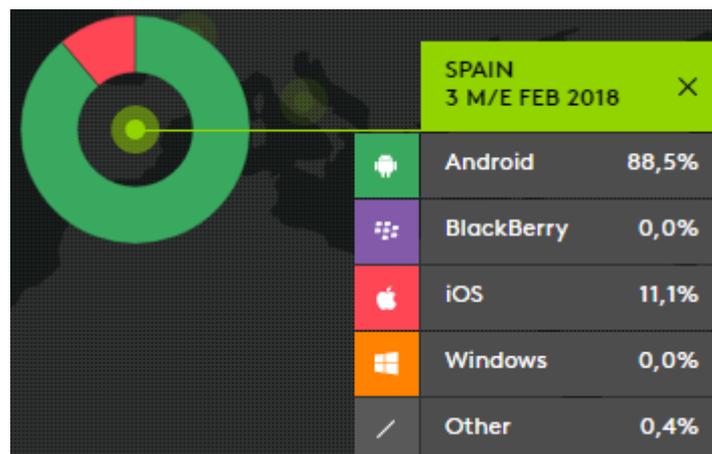


Figura 5.3.2: Cuota de mercado sistemas operativos móviles en España. (febrero, 2018)

## 5.4 Sistema operativo utilizado

Cabe destacar que desde un primer momento e incluso antes de realizar el análisis de los diferentes sistemas operativos, la principal elección para desarrollar el proyecto siempre fue Android. De todas formas, una vez realizado el análisis justificamos su elección.

Windows Mobile ha sido descartado de forma inmediata, ya que su cuota de mercado en España ha sido reducida de un 2.5% a un 0%, tal y como se puede observar en los gráficos del apartado anterior. Tendencia que se está siguiendo mundialmente, no solo en España, por el poderoso posicionamiento en el mercado de Android e iOS.

Por lo tanto, no merecía la pena desarrollar el proyecto en una plataforma la cual está prácticamente en desuso, ya que hay que tener en cuenta que las aplicaciones van dirigidas a

los usuarios, y cuanto más usuarios utilicen la aplicación más renombre puede tener y, al fin y al cabo, más éxito se puede obtener. Por estos motivos, muchos desarrolladores deciden no utilizar aplicaciones para este software.

Una vez descartado Windows 10 Mobile el análisis se centró en las dos grandes plataformas, en cuanto a cuota de mercado, Android e iOS.

Analizando la cuota de mercado, vemos como en España en febrero de 2018, el porcentaje de uso de dispositivos con Android es del 88,5% frente al 11,1% de dispositivos con iOS. Diferencia significativa que nos hace decantar por Android.

Por otro lado, iOS no permite la instalación en hardware de terceros con lo cual se limita única y exclusivamente a sus dispositivos mientras que Android abarca un sector más grande en cuanto a variedad de fabricantes de dispositivos móviles.

En cuanto al desarrollo del proyecto, utilizar Android no nos supone ningún gasto adicional ya que con el dispositivo móvil del que se dispone, corre sobre el sistema operativo Android.

En referencia al lenguaje de programación utilizado en los sistemas operativos, Android utiliza Java, un lenguaje de programación que ya conocía y había utilizado con anterioridad, tan solo había que adaptarse y aprender a utilizar las herramientas que nos ofrece el SDK. Java es un lenguaje de programación muy conocido mundialmente por lo que la comunidad Java es muy grande y es fácil el encontrar información requerida durante el periodo de desarrollo.

En cambio, si se hubiese utilizado iOS como sistema operativo para llevar a cabo el proyecto, hubiese requerido del aprendizaje de un nuevo lenguaje Objective C, que, a nivel personal y académico, hubiese aumentado los conocimientos adquiridos, pero a nivel de realización del proyecto hubiese podido atrasar y complicar su desarrollo.

En conclusión, la aplicación móvil para una red social deportiva se realiza destinada para dispositivos móviles con el sistema operativo Android.

---

## 6. Requisitos del sistema

---

A continuación, se describen los requisitos que debe cumplir la aplicación, tanto funcionales como no funcionales, de tal forma, que se permitan conocer todos los requisitos necesarios para que el proyecto cumpla con los objetivos, tanto desde el punto de vista de software como de hardware.

### 6.1 Requisitos funcionales

Los requisitos funcionales explican que tiene que hacer la aplicación, es decir, todas las funcionalidades disponibles sin especificar como se desarrollarán.

En cuanto a los usuarios que utilicen la aplicación los requisitos funcionales son los siguientes:

- Un usuario no registrado, podrá registrarse en la aplicación indicando los siguientes datos:
  - Nombre
  - Apellidos
  - Nombre de usuario (Nick)
  - Dirección de correo electrónico
  - Contraseña
  - Provincia y Municipio de residencia
  - Categorías deportivas favoritas
  - Foto de perfil
- Un usuario registrado podrá iniciar sesión en la aplicación.
- Cerrar la sesión actual de un usuario logeado en el sistema.
- Modificar la información de perfil:
  - Nombre
  - Apellidos
  - Nombre de usuario (Nick)
  - Provincia y Municipio de residencia
  - Categorías deportivas favoritas
  - Foto de perfil
- Acceder a los eventos del usuario:

- Eventos creados
  - Eventos registrados, es decir, en los que participa
  - Eventos en cola, es decir, los eventos en que el usuario no está registrado, pero se encuentra en la lista de espera ya que el evento está completo en cuanto al número de participantes.
- Acceder a la información del perfil del usuario que ha iniciado sesión o cualquier otro usuario registrado en la aplicación.
- Crear eventos deportivos personalizados indicando:
    - Título evento
    - Descripción evento
    - Día y hora de celebración del evento
    - Duración estimada
    - Número de participantes
    - Privacidad del foro
    - Ubicación del evento seleccionando provincia y municipio o indicando en un mapa la ubicación exacta.
    - Categoría deportiva del evento
    - Imagen del evento
- Modificar datos eventos deportivos creados.
- Suspender un evento deportivo creado.
- Eliminar usuarios registrados o en cola de un evento deportivo creado.
- Acceder al detalle de los eventos deportivos creados o de otros usuarios, visualizando:
    - Información del evento
    - Lista de participantes registrados
    - Lista de participantes en cola
    - Foro del evento
    - Ubicación en mapa del evento
- Participar en el foro de los eventos deportivos.
- Darse de alta en eventos deportivos ajenos.
- Darse de baja en eventos deportivos ajenos en el que el usuario está dado de alta.
- Buscador de eventos deportivos, en función de:
    - Título del evento
    - Fecha del evento

- Proximidad a una ubicación seleccionada en el mapa o mediante un municipio en concreto.
- Categorías deportivas seleccionadas.
- Activar o desactivar notificaciones:
  - Un nuevo usuario se registra en un evento creado
  - Un usuario se da de baja en un evento creado
  - Administrador elimina al usuario de un evento en el que se estaba registrado
  - Administrador modifica datos evento en el que se está registrado
  - Administrador cancela evento en el que se está registrado

En cuanto al funcionamiento interno de la aplicación, podemos definir los siguientes requisitos funcionales:

- Listar eventos deportivos a un usuario que haya iniciado sesión en función de su municipio de residencia y sus categorías deportivas favoritas seleccionadas.
- Registrar ubicación GPS del dispositivo móvil que ha iniciado sesión, si se ha permitido previamente al acceso a la ubicación.
- Manejo automático de la paginación del listado de eventos deportivos resultante de una búsqueda de forma automática al realizar “scroll”.
- Proceso automático de actualización del estado de los eventos en función de la fecha actual y su fecha de celebración (abierto, completo, suspendido o finalizado).
- Gestionar el envío de notificaciones electrónicas.

## 6.2 Requisitos no funcionales

Los requisitos no funcionales del sistema son aquellos que describen cualidades o criterios que sirven para juzgar el funcionamiento general del sistema, en lugar de las funcionalidades específicas.

Estos requisitos pueden ser características de seguridad, compatibilidad, usabilidad, escalabilidad, recursos necesarios para el buen funcionamiento, etc.

Para controlar y mantener el acceso de los usuarios a la aplicación, hay que guardar datos de carácter personal de los diferentes usuarios registrados. Por lo tanto, algunos requisitos no funcionales para mantener la seguridad del sistema son:

- El servidor debe asegurar el acceso a la aplicación únicamente a los usuarios dados de alta en el sistema y garantizar el acceso a su perfil.
- El servidor debe asegurar la persistencia y lógica de los datos.
- Se conserva cifrada la contraseña de los usuarios en base de datos.
- Evitar que se puedan obtener datos privados de otros usuarios.

En cuanto al rendimiento de la aplicación, el intercambio de información entre la aplicación y el servidor a través de la API REST debe tener unos tiempos de respuesta rápidos.

Para poder aprovechar el potencial de la aplicación, es aconsejable que el dispositivo móvil sobre el que corre la aplicación disponga de un receptor GPS y este sea activado y se permita su uso en la aplicación. De esta forma, se pueden localizar eventos deportivos en función de la ubicación actual del dispositivo. El dispositivo debe tener conexión a internet para poder interactuar con el servidor.

En referencia a la usabilidad, la interface de usuario debe de ser sencilla y fácil de utilizar por parte de los usuarios. Debe permitir de forma intuitiva la navegación por las diferentes pantallas de la aplicación.

En términos de escalabilidad, el servidor debe desarrollar una API completa y fácil de utilizar. La cual permita la gestión de todos los datos de los usuarios y sus eventos deportivos creados. Independiente de la plataforma utilizada.

En el apartado 2.7 se mencionaron los recursos técnicos con los cuales se ha desarrollado y testeado la aplicación mostrando un rendimiento aceptable. En cuanto a los requerimientos mínimos para poder ejecutar la aplicación en dispositivo móvil:

- Se requiere un dispositivo móvil con sistema operativo Android, versión mínima aceptada 5.0 LOLLIPOP (Nivel de API 21). Por lo tanto, cualquier dispositivo móvil que soporte como mínimo la versión mencionada podrá ejecutar la aplicación desarrollada.

- No se declara una versión máxima de API en la cual la aplicación está diseñada a ejecutarse. A partir de la versión 5.0 de Android en adelante es compatible para ejecutar la aplicación.

Según un estudio realizado por Google en Julio de 2018 [5] estiman el porcentaje de dispositivos que utilizan diferentes versiones de Android.

Versión	Nombre sistema	API	Distribución
2.3.3 – 2.3.7	Gingerbread	10	0.2%
4.0.3 – 4.0.4	Ice Cream Sandwich	15	0.3%
4.1.x	Jelly Bean	16	1.2%
4.2.x		17	1.9%
4.3		18	0.5%
4.4	KitKat	19	9.1%
5.0	Lollipop	21	4.2%
5.1		22	16.2%
6.0	Marshmallow	23	23.5%
7.0	Nougat	24	21.2%
7.1		25	9.6%
8.0	Oreo	26	10.1%
8.1		27	2.0%

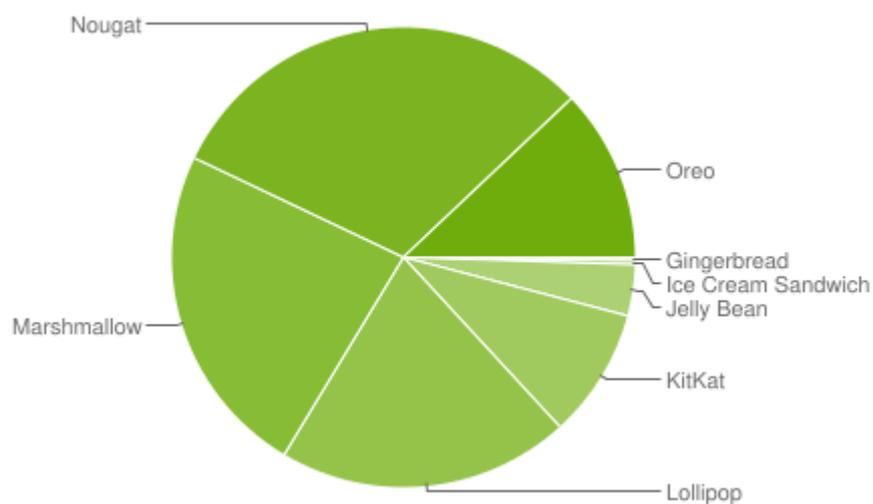


Figura 6.2.1: Cantidad relativa de dispositivos que usan una versión determinada de la plataforma Android

Por lo tanto, según los datos recopilados por Google actualmente la aplicación podría ejecutarse en un 86.8% de los dispositivos móviles que utilizan Android como sistema operativo.

---

## 7. Estudio y decisiones

---

En este apartado se definen los diferentes programas y librerías utilizados durante todo el proyecto. Se tienen en cuenta tanto los programas y librerías utilizadas en el servidor, en la aplicación móvil y los destinados a la realización de la documentación. Se justifica el motivo de la elección de dichas herramientas.

### 7.1 Aplicación servidor

Descripción de las herramientas utilizadas para el desarrollo de la aplicación servidor como el IDE utilizado, el lenguaje de programación y framework utilizados durante el desarrollo. Cabe destacar que la aplicación servidor se trata de una aplicación desarrollada en la plataforma Java Enterprise Edition, conocido informalmente como Java empresarial, con la finalidad de desarrollar una aplicación basada en una arquitectura multicapa bajo un servidor de aplicaciones Wildfly 9.

#### 7.1.1 Java Development Kit (JDK)



Figura 7.1.1.1: Logo Java Development Kit

Para el desarrollo de la aplicación servidor, es necesaria la instalación del software que provee las herramientas de desarrollo para la implementación de programas en lenguaje Java. JDK incluye Java Runtime Environment, el compilador Java y las API de Java. El compilador *javac* se encarga de convertir el código fuente (.java) en bytecode (.class), para posteriormente ser interpretado y ejecutado por la máquina virtual de Java. Se trata de un componente imprescindible para el desarrollo de la aplicación servidor.

## 7.1.2 Servidor aplicaciones Wildfly 9



Figura 7.1.2.1: Logo de la marca Wildfly

Como servidor de aplicaciones se utiliza el servidor Wildfly 9 (antiguo JBoss) [6] ya que se trata de un servidor Java EE 7 de perfil completo y de código abierto implementado en Java. En concreto, se ha utilizado la versión 9.0.2 Final, la cual nos proporciona un conjunto completo de las APIs de Java EE 7. Puede ser utilizado en cualquier sistema operativo para el cual esté disponible la máquina virtual de Java. Se trata de un proyecto de software libre sujeto a los requisitos de la GNU Lesser General Public License (LGPL), versión 2.1. Existen otros servidores de aplicaciones Java EE certificados como Oracle WebLogic o GlassFish.

Uno de los principales motivos de utilizar este servidor de aplicaciones es que al instalarlo y configurarlo nos proporciona una serie de herramientas, que conforman proyectos de código abierto más representativos en el mundo de Java, como Hibernate y RESTEasy. Otros de los motivos de utilizar Wildfly se debe a que ya se había utilizado anteriormente en una asignatura de la universidad, por lo tanto, ya se conocían sus beneficios y su funcionamiento, aspectos que facilitaban el desarrollo del proyecto.

## 7.1.3 Java EE

Como se ha mencionado anteriormente para el desarrollo de la aplicación servidor se ha utilizado la plataforma de programación Java Enterprise Edition. [7]

La cual nos permite desarrollar y ejecutar software en el lenguaje de programación Java. Nos permite utilizar arquitecturas multicapa apoyadas en componentes de software modulares ejecutándose bajo un servidor de aplicaciones, en nuestro caso el ya mencionado Wildfly 9.

Se trata de un conjunto de especificaciones estandarizadas a disposición de las aplicaciones empresariales que son implementadas por los servidores de aplicaciones.

El modelo clásico de capas en la arquitectura de las aplicaciones Java EE se divide en las siguientes:

- **Cliente:** Encargada de presentar la información al usuario final. Podría ser un navegador, dispositivo móvil, otra aplicación, etc.
- **Capa Web:** Intermediario entre el cliente y la capa de negocio. Obtiene las peticiones en un formato en concreto, en nuestro caso JSON, y responden al cliente usando tal formato.
- **Capa de negocio:** Contiene la lógica de negocio de la aplicación, encargado de proporcionar y persistir los datos de la capa cliente en función de las peticiones recibidas en la capa Web.
- **Sistemas de información:** Base de datos en la cual se persisten los datos manejados en la aplicación. En el caso del proyecto se ha utilizado una base de datos relacional MySQL.

En concreto, para el proyecto se ha utilizado la siguiente arquitectura en capas:

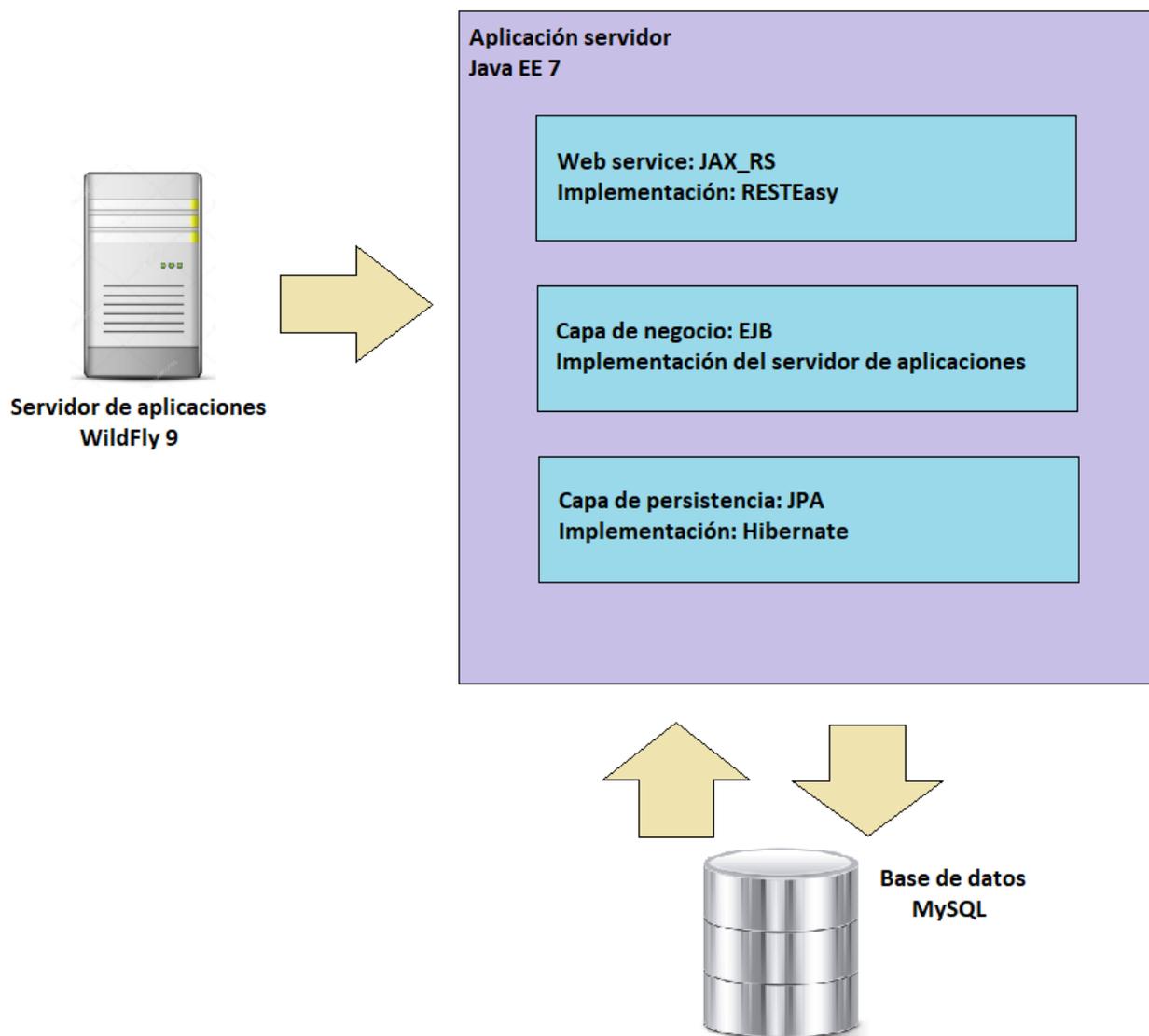


Figura 7.1.3.1: Arquitectura en capas aplicación servidor Java EE 7

Uno de los principales motivos de escoger Java EE es la estructuración que nos aporta su especificación, permitiendo la división en diferentes capas que se centren en un área específica, permitiendo abordar los problemas de una forma más concreta a nivel de cada capa. Además, cada API/especificación está preparada para funcionar en compañía de las demás de forma nativa y, por tanto, en su conjunto forman una solución estructurada para desarrollar una aplicación end-to-end (de principio a fin).

En los siguientes apartados se explican las diferentes capas y sus implementaciones.

## 7.1.4 JAX-RS

JAX-RS: Java API para RESTful Web Services se trata de una API de Java que proporciona funcionalidades para la creación de servicios web de acuerdo con el estilo arquitectónico REST.

Java EE 7 con JAX-RS 2.0 aporta varias características que simplifican el desarrollo de servicios RESTful.

En concreto, JAX-RS es una especificación con las siguientes características:

- Colección de anotaciones para declarar clases de recursos y los tipos de datos que admiten.
- Un conjunto de interfaces que permite a los desarrolladores de aplicaciones obtener acceso al contexto en tiempo de ejecución.
- Una infraestructura extensible para integrar los manejadores de contenido personalizados.

Algunas de las diferentes notificaciones utilizadas son:

- **@ApplicationPath:** Esta anotación identifica la ruta de la aplicación que sirve como el URI base de todos los recursos. Se utiliza en la clase de configuración, es decir, la subclase de `javax.ws.rs.core.Application`.

```
@ApplicationPath("/rest")
public class MyApplication extends Application {
}
```

- **@Path:** Identifica la ruta URI a la que responderá una clase de recurso o un método de clase. Hay que crear una clase anotada con `@Path` y definir la ruta relativa al URI base.

```
@Path("/usuario")
@RequestScoped
public class UsuarioRESTService extends GenericRESTService {
    . . .
}
```

```
@Path("login")
public Response autenticacion ( . . . ) {
    . . .
}
```

- **@GET, @PUT, @POST, @DELETE:** Especifican el tipo de petición HTTP que espera el recurso.

```
@POST
@Path("login")
public Response autenticacion ( . . . ) {
    . . .
}
```

- **@Consumes, @Produces:** Especifican el tipo de datos aceptado para la petición y respuesta.

```
@POST
@Path("login")
@Consumes(MediaType.APPLICATION_JSON)
@Produces(MediaType.APPLICATION_JSON)
public Response autenticacion ( . . . ) {
    . . .
}
```

- **@PathParam:** Enlaza un parámetro a un segmento de ruta.
- **@Context:** Devuelve todo el contexto del objeto HttpServletRequest. Utilizado mayoritariamente para el control de sesiones.

```
@Path("logout/{id}")
@DELETE
@Produces(MediaType.APPLICATION_JSON)
public Response logout(@Context HttpServletRequest req, @PathParam("id") Long
idUserario) {
    . . .
}
```

El servidor de aplicaciones Wildfly 9 utiliza el framework RESTEasy para dar soporte a JAX-RS.

## 7.1.5 RESTEasy

RESTEasy [8] se trata de la implementación proporcionada por el servidor Wildfly de la especificación JAX-RS de java EE.

Proporciona varios framework para soportar la construcción de aplicaciones y servicios web REST. Se trata de un proyecto de código abierto respaldado por JBoss.

Mediante RESTEasy los servicios web se tratan como POJO (Plain Old Java Object) a los cuales se les añade funcionalidades mediante anotaciones.

Este framework no es de uso exclusivo en el servidor Wildfly, sino que es portable para Tomcat y otros muchos servidores de aplicaciones.

Por lo tanto, para realizar el servicio web (web services) de la aplicación se utilizaba el estándar REST (Representational State Transfer) arquitectura mediante el uso del protocolo HTTP, proporciona una API mediante sus métodos (GET, POST, PUT, DELETE, etc.) permite realizar operaciones entre la aplicación servidor que ofrece el servicio y la aplicación cliente.

Para el intercambio de datos se ha utilizado el formato JSON.

## 7.1.6 Enterprise Java Beans

Enterprise JavaBeans [9] se trata de una de las interfaces de programación de aplicaciones (API) que forma parte del estándar de construcción de aplicaciones empresariales Java EE.

El objetivo de los EJB es proporcionar a los programadores un modelo que les permita abstraerse de los problemas generales en cuanto al desarrollo de una aplicación de tipo empresarial (conurrencia, transacciones, seguridad, etc.), para de esta forma priorizar el desarrollo en la lógica de negocio y dejar el resto de responsabilidades al contenedor de aplicaciones donde se ejecutará la aplicación.

Este modelo está basado en diferentes componentes que aportan flexibilidad y reutilización.

En este caso, el servidor de aplicaciones escogido utiliza la versión 3.2 de EJB.

Los componentes *Sessions Beans* (Beans de sesión), son los componentes que contienen la lógica de negocio que solicitan los clientes de la aplicación servidor.

Tras las solicitudes, el cliente obtiene una vista del bean de sesión y no el objeto real. De esta forma, se permite al contenedor realizar ciertas operaciones sobre el bean de sesión real de forma transparente para el cliente como gestionar el ciclo de vida del bean o paralelamente solicitar instancias a otros contenedores.

Existen tres tipos de Beans de sesión según la anotación utilizada en su declaración:

- **@Stateless:** Utilizado para definir Beans de sesión sin estado, es decir no requieren mantener un estado entre diferentes peticiones. De esta forma se debe asumir que distintas solicitudes al contenedor de un mismo bean de sesión sin estado pueden devolver vistas a objetos diferentes. Por lo tanto, el bean de sesión sin estado puede ser compartido entre varios clientes.

El hecho de no mantener ningún estado hace que sean muy eficientes a nivel de uso de memoria y recursos en el servidor.

```

@Stateless
@LocalBean
public class UsuarioService {
    . . .
}

```

Con la anotación **@LocalBean** indicamos que el servicio actúa como una interface a ser utilizada desde un cliente local sin necesidad de definir una interface en concreto.

- **@Stateful:** Utilizado para crear Beans de sesión con estado, es decir, se mantiene el estado entre diferentes peticiones realizadas por un mismo cliente. El bean es creado justo antes de la primera invocación de un cliente, y se destruye a indicación del mismo cliente o por cierre de sesión. Son menos eficientes a nivel de uso de memoria y recursos en el servidor que los Beans de sesión sin estado.
- **@Singleton:** Se indica que se trata de un componente que puede ser compartido por muchos clientes instanciado y creado una única vez. A nivel de memoria y recursos de servidor son los más eficientes lo que su uso se limita a resolver ciertos problemas muy específicos.

La utilización de Beans promueve el concepto de inyección de dependencias mediante la utilización de la anotación **@EJB** que nos permite crear una dependencia de un bean definido, obteniendo una referencia del EJB indicado.

```

RequestScoped
public class UsuarioRESTService extends GenericRESTService {
    @EJB
    UsuarioService usuarioService;
    . . .
}

```

## 7.1.7 Java Persistence Api

Se trata de la especificación de EJB 3.x en la cual se definen los principios básicos de gestión de la capa de persistencia en el mundo de Java EE.

El principal objetivo del diseño de JPA es aprovechar las ventajas de la programación orientada a objetos para interactuar con una base de datos utilizando el patrón de mapeo objeto-relacional.

De esta manera, podemos establecer una correlación entre una base de datos relacional y un sistema orientado a objetos. Esta correlación es llamada ORM (Object Relational Mapping), mediante la utilización de anotaciones.

Una de las ventajas de JPA es que existen varias implementaciones (frameworks) de la especificación:

- Hibernate
- EclipseLink

- ObjectDb
- OpenJPA, etc.

Esto nos aporta flexibilidad a la hora de abordar que framework de persistencia escoger para nuestra aplicación.

La implementación utilizada por el servidor de aplicaciones Wildfly es Hibernate.

### 7.1.8 Hibernate

Hibernate [10] representa la implementación de la especificación definida en JPA.

Por lo tanto, su finalidad es construir una capa de persistencia apoyándonos en las definiciones y reglas de dicha especificación.

No por esto Hibernate se limita a la implementación de JPA, sino que añade diversas funcionalidades adicionales. Por ejemplo, Hibernate contiene la capacidad de trabajar con bases de datos NoSQL, algo que JPA no cubre.

Por lo tanto, Hibernate se trata de un framework ORM de mapeo objeto-relacional para la plataforma Java (disponible también para .Net mediante NHibernate) que se encarga del mapeo de entidades y atributos entre una base de datos relacional y el modelo de objetos de una aplicación mediante anotaciones en los Beans de las entidades que permiten establecer las relaciones o en versiones más antiguas mediante la configuración de ficheros XML.

Mediante Hibernate se permite la manipulación de datos en la base de datos operando sobre objetos aprovechando las características de la programación orientada a objetos. Nos ofrece un lenguaje de consulta HQL (Hibernate query language) con el cual realizar las consultas programáticamente.



Figura 7.1.8.1: Arquitectura Hibernate

Algunas de las anotaciones utilizadas por Hibernate son:

- **@Entity:** Se aplica a una clase de Java para indicar que se trata de una entidad a persistir.

```
@Entity
public class Usuario implements Serializable {
    . . .
}
```

- **@Table (name= “”):** Se utiliza para indicar el nombre de la tabla en base de datos de la clase a persistir. En caso de no indicar el nombre se utiliza el nombre de la clase.
- **@Id:** Se aplica a un atributo de Java de la clase a persistir para indicar que es la clave primaria.

```
@Id
@GeneratedValue(strategy = GenerationType.IDENTITY)
protected Long id;
```

- **@GeneratedValue:** Indica la estrategia de generación de la clave primaria. Por ejemplo, por auto-incremento.
- **@Column (name = ""):** Se aplica a un atributo de Java para indicar el nombre de la columna en base de datos u otras características como la longitud, si puede adoptar valores nulos, etc.

```
@Column(length = 100, nullable = false)
private String titulo;
```

- **@OneToMany:** indica una relación unidireccional de muchos a uno.

```
@Entity
public class Evento implements Serializable {
    . . .
    @JsonIgnore
    @ManyToOne(fetch = FetchType.EAGER)
    private Deporte deporte;
    . . .
}
```

```
@Entity
public class Deporte implements Serializable {
    . . .
    @OneToMany(cascade = CascadeType.ALL, mappedBy="deporte", fetch=FetchType.LAZY)
    @JsonIgnore
    private List<Evento> eventosCreados;
    . . .
}
```

- **@ManyToMany:** indica una relación de muchos a muchos entre dos entidades (n: m).

```
@Entity
public class Usuario implements Serializable {
    . . .
    @ManyToMany(cascade = CascadeType.ALL, fetch = FetchType.LAZY)
    @JsonIgnore
    private List<Deporte> deportesFavoritos;
    . . .
}
```

```
@Entity
public class Deporte implements Serializable {
    . . .
    @ManyToMany(cascade=CascadeType.ALL,mappedBy="deportesFavoritos",fetch=FetchType.
LAZY)
    @JsonIgnore
    private List<Usuario> usuarios;
    . . .
}
```

- **@OneToOne:** indica una relación de uno a uno entre dos entidades (1: 1).

```
@JsonIgnore
@OneToOne(fetch = FetchType.EAGER)
private Ubicacion ubicacionGPS;
```

A parte de que Hibernate es el framework que nos proporciona el servidor de aplicaciones escogido, este es probablemente el ORM más utilizado y más influencia tuvo en la elaboración de la especificación Enterprise Java Beans 3. Esto significa que se trata de una herramienta bastante probada en entornos de producción y se encuentra bastante madura, con lo cual se dispone en la red una gran cantidad de información para consultar en momentos de dudas o problemas durante el desarrollo.

### 7.1.9 MySQL

Para almacenar los datos de la aplicación se ha utilizado MySQL un sistema de gestión de bases de datos relacional programado en C y C++.

Conjuntamente se ha utilizado MySQL Workbench, que se trata de una herramienta que nos proporciona una interface gráfica para realizar la administración de la base de datos almacenada en el sistema gestor de base de datos MySQL.

Algunas de las funcionalidades utilizadas que nos ofrece MySQL Workbench son el editor de SQL, modelado de los datos, administración de la base de datos, etc.

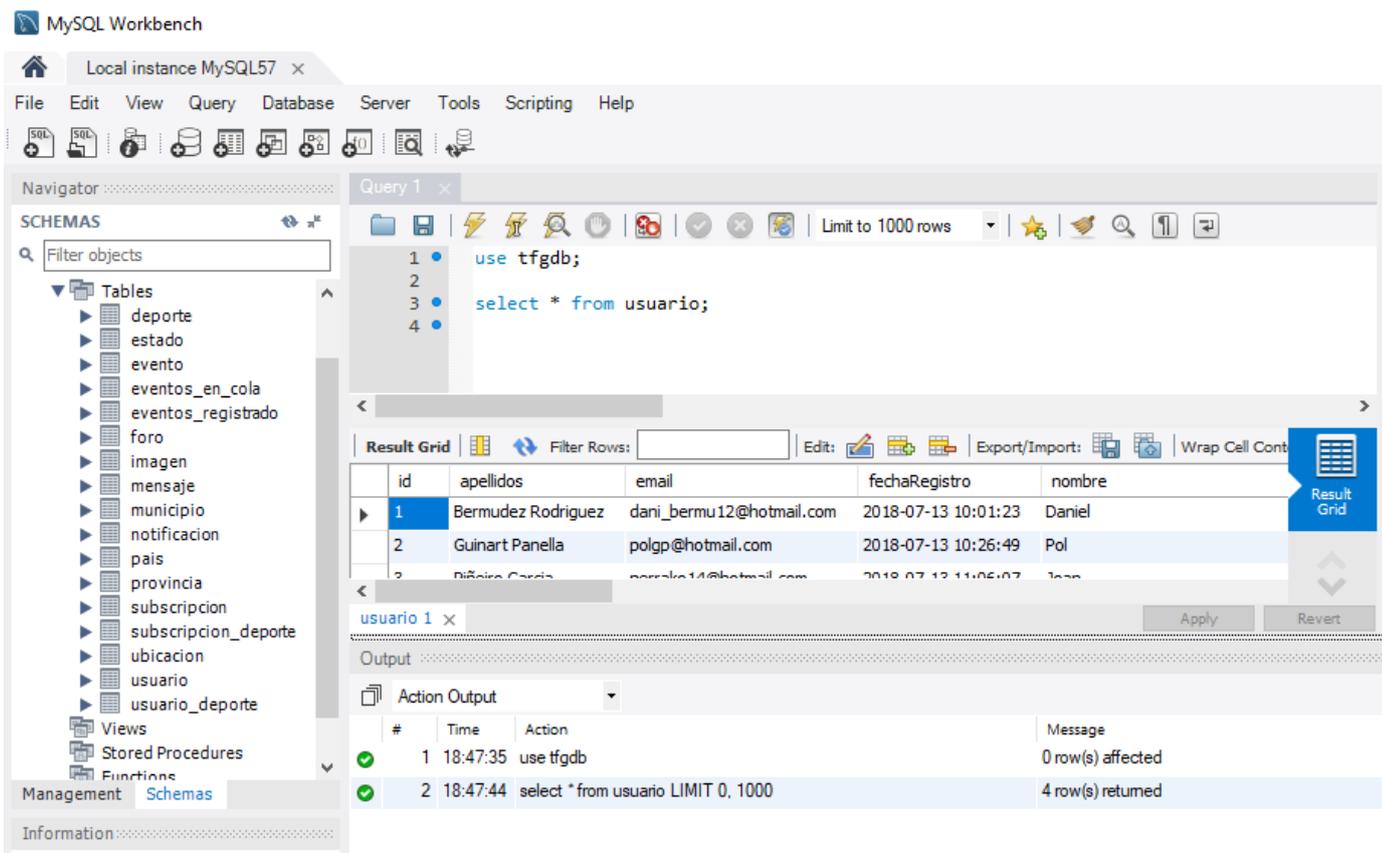


Figura 7.1.9.1: Interface gráfica Workbench MySQL

## 7.1.10 Eclipse

Eclipse se trata del entorno de programación (IDE) utilizado para el desarrollo de la aplicación servidor Java EE. En concreto la versión Oxygen 4.7. [11]

Se trata de una plataforma de software compuesto por un conjunto de herramientas de programación de código abierto multiplataforma.

Eclipse a día de hoy es considerado uno de los mejores IDEs para desarrollo Java, todo y que no es el único que lenguaje de programación que soporta.

Uno de los principales motivos para utilizar Eclipse es su versión *Eclipse IDE for Java EE developers* ya que dispone de una serie de plugins destinadas a desarrollar proyectos en esta plataforma. Otra razón de su utilización es que se trata de un IDE que no nos supone ningún tipo de coste económico.

## 7.1.11 Postman

Postman [12] se trata de una herramienta que nos permite el envío de peticiones HTTP REST sin necesidad de tener un cliente desarrollado, actuando como tal.

Podemos realizar dichas peticiones (GET, POST, DELETE, UPDATE...) a una dirección en concreto.

Principalmente, se ha utilizado para testear la API REST diseñada en el servidor sin necesidad de tener la aplicación cliente desarrollada. De esta forma podemos probar todas las funcionalidades de la API sin necesidad de escribir ni una sola línea de código.

Postman es gratuito, todo y que existe una versión Premium que añade ciertas funcionalidades. En este caso, la versión gratuita disponía de todas las funcionalidades necesarias requeridas durante el desarrollo del proyecto.

Se ha escogido su utilización ya que esta herramienta ya había sido utilizada con anterioridad en otros proyectos y ya se conocían sus funcionalidades. Además, dispone de una interface gráfica muy agradable y sencilla de utilizar la cual nos permite categorizar todas las peticiones REST realizadas al servidor desarrollado.

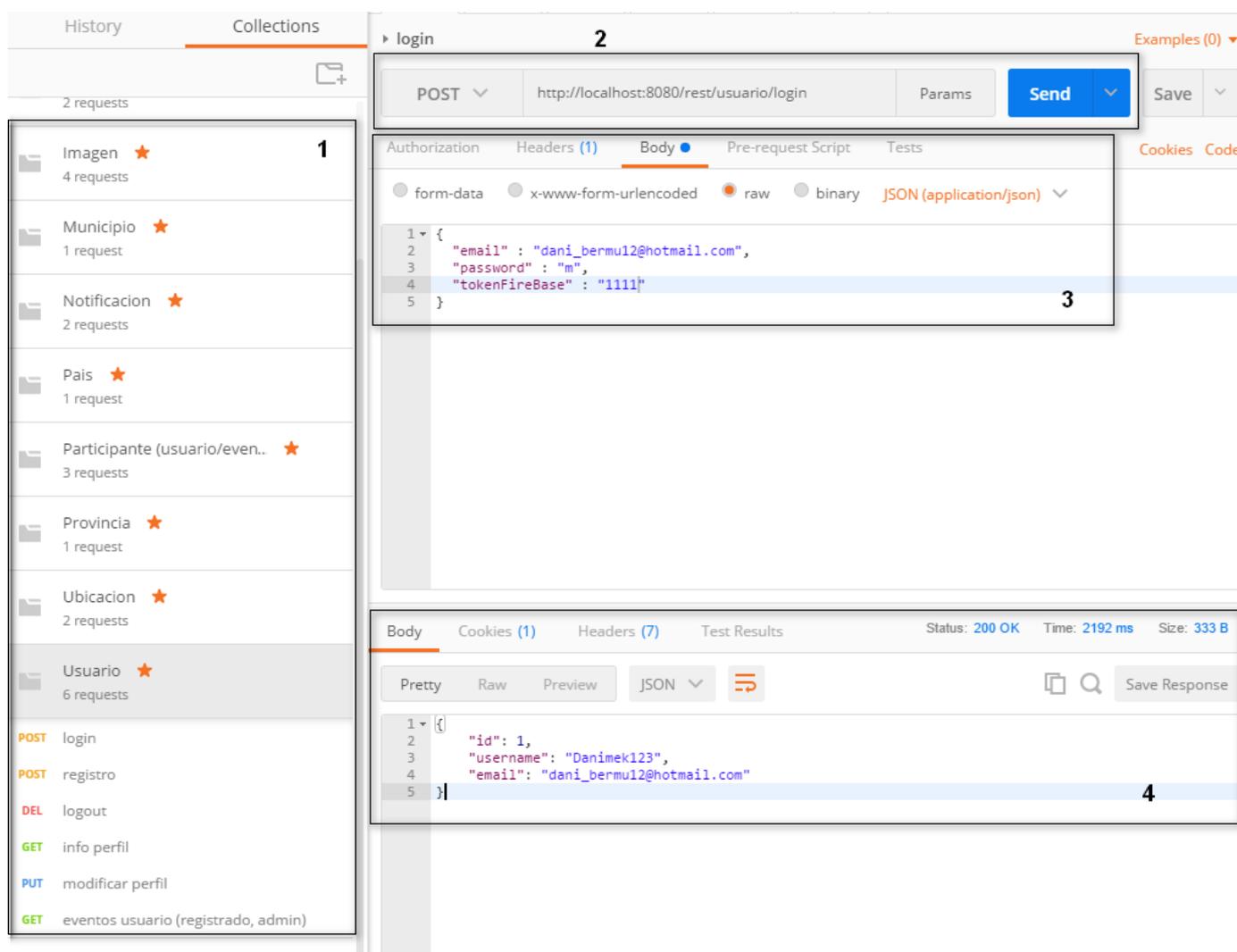


Figura 7.1.11.1: Interface gráfica Postman

Tal y como se observa en la Figura 7.1.11.1 se ha dividido la interface gráfica en diferentes secciones.

En la sección 1 se mantienen las diferentes colecciones de peticiones REST realizadas contra el servidor durante la etapa de desarrollo y testeo de la API REST diseñada.

En la sección 2 se introduce la URL destino a la cual se realiza la petición y el tipo de petición a realizar.

En la tercera sección se introduce y configura el contenido de la petición a realizar, en este caso se indican los datos a la hora de realizar la petición en formato JSON (JavaScript Object Notation).

Finalmente, es la sección 4 podemos visualizar la respuesta que nos da el servidor a la petición realizada y configurada en las dos anteriores secciones.

## 7.1.12 JSON

Aproximadamente a principio de los años 90 surgió el problema de que las máquinas pudieran entenderse entre sí, ya que pueden utilizar distintos sistemas operativos y sus programas estar desarrollados en diferentes lenguajes de programación.

Una de las soluciones fue la creación del estándar XML, y más tarde de JSON, los cuales son formatos de datos utilizados para intercambiar información entre diferentes máquinas independientemente de los aspectos comentados anteriormente.

JSON [13] es considerado un formato más legible que XML e igual de eficaz durante el proceso de comunicación entre máquinas.

JSON está formado por una colección de pares (nombre - valor) o por una lista ordenada de valores.

XML es un lenguaje de marcado orientado a documentos, en cambio, JSON es un metalenguaje orientado a datos.

Este formato es el que suele utilizarse en servicios web REST como el desarrollado en la aplicación por este motivo se decidió a su utilización.

```
{
  "email" : "test@test.com",
  "username" : "userTest",
  "password" : "m",
  "tokenFireBase" : "1111",
  "nombre" : "Daniel",
  "apellidos" : "Bermudez Rodriguez",
  "municipio" : 1,
  "deportesFavoritos" :
  [
    1 , 4 , 7
  ]
}
```

## 7.2 Aplicación cliente

En este apartado se mencionan y definen las diferentes herramientas utilizadas durante el desarrollo de la aplicación cliente.

Como se ha mencionado anteriormente, se ha desarrollado una aplicación para dispositivos móviles en Android utilizando el IDE Android Studio apoyándonos en diferentes librerías y frameworks.

### 7.2.1 Android

En el *capítulo 5*, se realiza una definición del sistema operativo Android y se justifica su elección. En este apartado, se mencionarán y definirán algunos conceptos básicos a tener en cuenta y que se harán referencia de ahora en adelante en la memoria.

#### Activity

Las actividades (activites) [14] son un componente de la aplicación que alberga una interface gráfica (pantalla) mediante la cual los usuarios interactúan para realizar cualquier tipo de acciones.

Una aplicación móvil Android normalmente consiste en múltiples actividades vinculadas entre sí. Generalmente, se define una actividad principal la cual se lanza en el momento en que se inicia la aplicación.

Una actividad, mediante la interacción de los usuarios en su interface gráfica, puede iniciar otras actividades. Cada vez que se inicia una nueva actividad, la anterior se detiene y el sistema la conserva en la pila de actividades. Esta pila actúa con la metodología LIFO (último en entrar primero en salir), por lo tanto, si el usuario termina de interactuar con la actividad actual y decide volver a la anterior (botón Atrás), la actividad se quita de la pila y se reanuda la ejecución de la actividad anterior.

Todo el proceso desde que se muestra la pantalla de una actividad hasta que se destruye es conocido como el ciclo de vida de una actividad.

Para crear una actividad se debe implementar el método *onCreate()* en una subclase de Activity. En este método se deben inicializar los componentes fundamentales de la actividad como por ejemplo mediante el método *setContentView()* definir la interfaz de usuario de la actividad.

#### Fragments [15]

Como las actividades son componentes a los cuales se les asocia una interface gráfica. A diferencia de las actividades, proporcionan una reutilización de componentes ya que se puede usar un mismo fragmento en diferentes actividades. Siempre van ligados a una actividad ya que representan un comportamiento o una parte de la interfaz de usuario en una actividad. En una misma actividad se permite combinar múltiples fragmentos para crear una interfaz multipanel.

El ciclo de vida de un fragmento se ve directamente afectado por el ciclo de vida de la actividad anfitriona.

Para crear un fragmento se debe implementar el método `onCreate()` en una subclase de `Fragment`. En este método se deben inicializar los componentes fundamentales del fragmento como por ejemplo mediante el método `onCreateView()` definir la interfaz de usuario de la actividad.

## Intents

Un `Intent` [16] es un objeto que representa la intención de hacer algo en la aplicación que se usa para solicitar una acción de otro componente de la aplicación.

Mediante la utilización de `Intents` se facilita la comunicación entre componentes. Existen 3 casos en los cuales usar este objeto:

- **Para lanzar una nueva actividad:** Para pasar de una actividad a otra, mediante el método `startActivity()` si le pasamos un objeto `Intent` podemos describir la nueva actividad a iniciar y los datos necesarios para ello. Ver figura 7.2.1.1.  
Además, podemos recibir en una actividad el resultado obtenido al finalizar otra actividad. Mediante el método `startActivityForResult()` se recibe el resultado como objeto `Intent`.
- **Para iniciar un servicio:** Un servicio es un componente que se ejecuta en segundo plano y no posee una interfaz gráfica. Por ejemplo, Android dispone de un servicio para ejecutar la cámara móvil del dispositivo.  
Pasando un `Intent` al método `startService()` se describe el servicio a iniciarse y los datos necesarios para su ejecución.
- **Para entregar un mensaje:** Un mensaje se trata de un aviso que las aplicaciones pueden recibir. Con los métodos `sendBroadcast()`, `sendOrderedBroadcast()` o `sendStickyBroadcast()` podemos utilizar un `Intent` definiendo el mensaje a entregar.

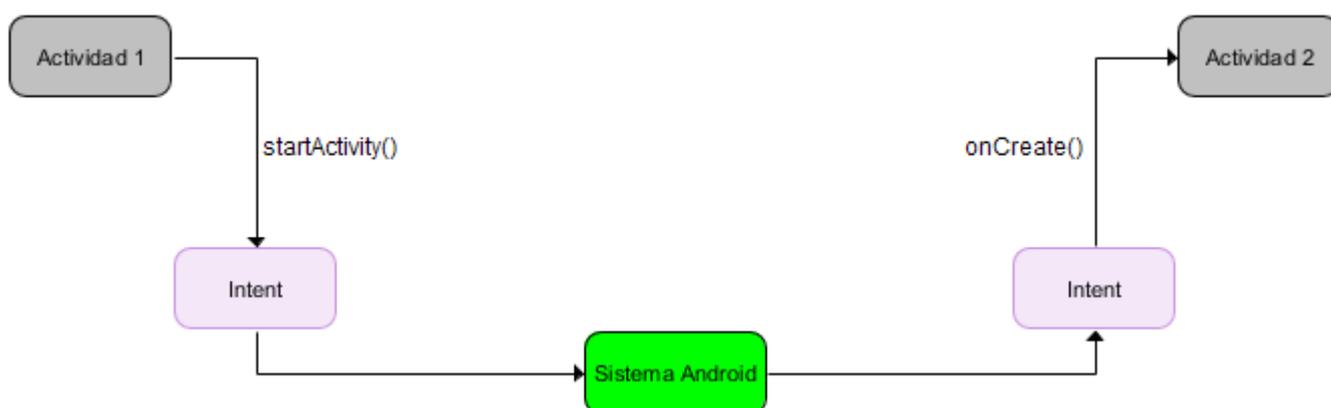


Figura 7.2.1.1: Descripción funcionamiento `Intents` para iniciar una nueva actividad. En primer lugar, la `Actividad 1` crea un `Intent` y lo entrega al método `startActivity()`. A continuación, el sistema `Android` busca en todas las aplicaciones un filtro de `Intents` que coincida con la `Intent` que ha recibido. Finalmente, cuando encuentra una coincidencia el sistema inicia la actividad coincidente (`Actividad 2`) mediante la invocación de su método `onCreate()` transmitiéndolo al `Intent`.

## Layouts

Los Layouts (diseños) [17] definen la estructura visual de una interfaz de usuario de una actividad o Fragment. Android permite crear diseños declarando los elementos de la interfaz en un archivo XML o creando instancias de elementos del diseño en tiempo de ejecución programáticamente.

Actúa como un contenedor de *Views* para establecer un orden visual que facilite la interacción del usuario con la interfaz.

Para cargar un diseño en una actividad una vez se ha definido el archivo XML se utiliza el método *setContentView()* dentro del método *onCreate()*.

Algunos de los diseños más populares son:

- **LinearLayout:** Es un *viewGroup* que distribuye los elementos en una sola dimensión establecida. Es decir, en una sola columna en vertical o en una sola fila en horizontal. Ver figura 7.2.1.2.

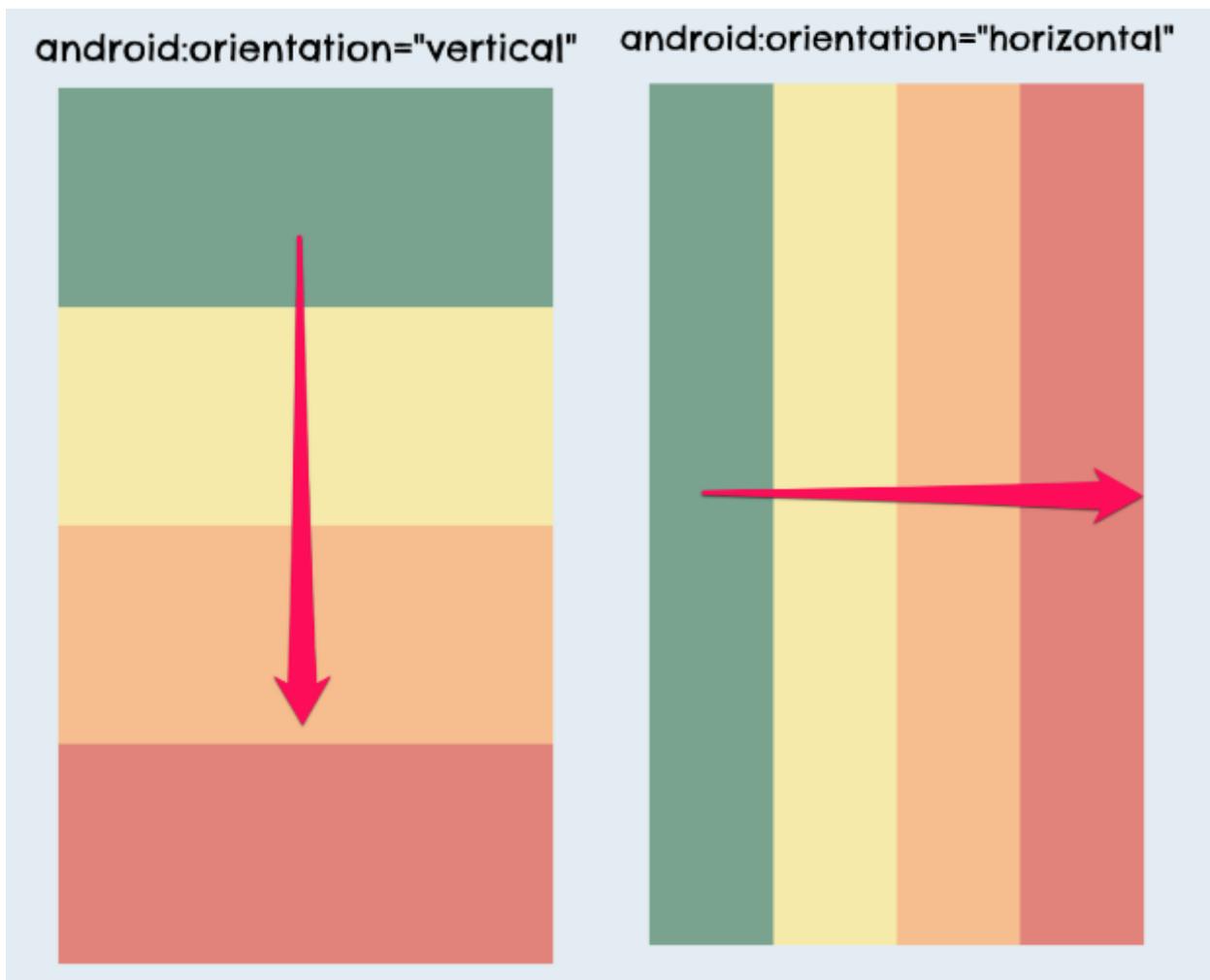


Figura 7.2.1.2: LinearLayout

- **FrameLayout:** Es un *viewGroup* creado para mostrar un solo elemento en pantalla. No obstante, se pueden añadir varios hijos con el fin de superponerlos. Para alinear cada elemento dentro del diseño se usa el parámetro *Android:layout\_gravity*. Ver figura 7.2.1.3.

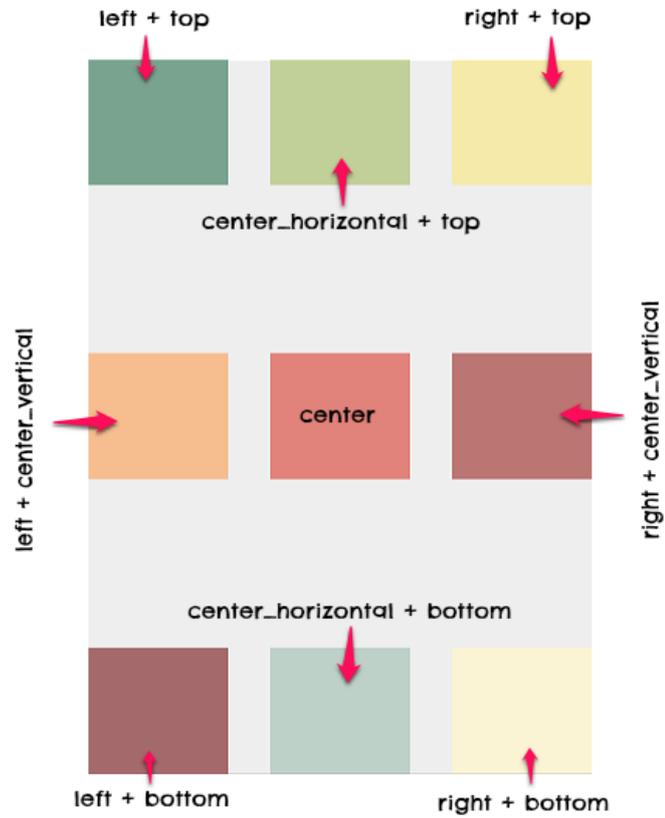


Figura 7.2.1.3: FrameLayout

- **RelativeLayout**: Permite alinear cada hijo con referencias subjetivas de cada elemento hermano. Es el diseño más flexible y elaborado. Ver figura 7.2.1.4.

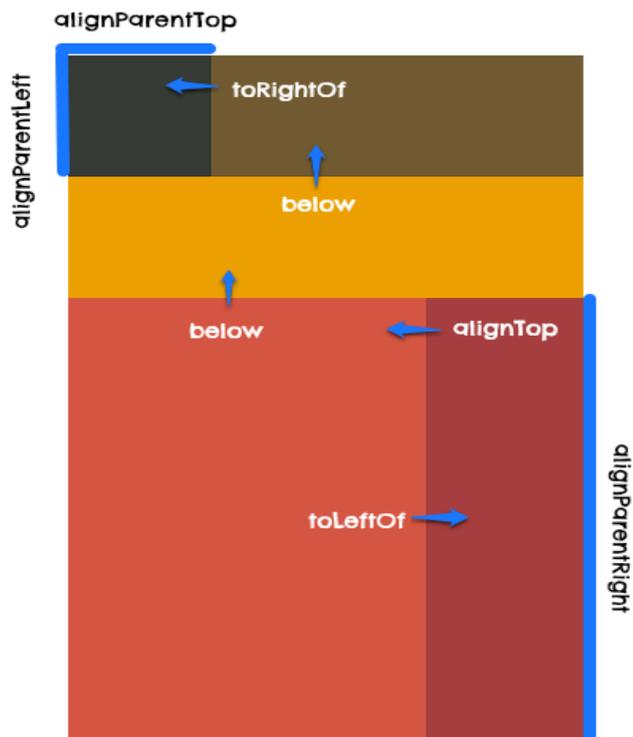


Figura 2.2.1.4: RelativeLayout

- **ConstraintLayout:** Permite crear diseños grandes y complejos simplificando las interfaces en anidamiento. Este layout, similar al RelativeLayout, nos permite establecer relaciones entre todos los elementos y la propia vista padre, permitiendo así ser mas flexible que la resta de diseños. En la versión 2.2 de Android Studio se introdujo el nuevo *LayoutEditor* que nos permite utilizar las constraintLayouts. En la figura 7.2.1.5 se muestra la interfaz del nuevo editor de Android Studio para crear y administrar diseños con ConstraintLayout. Mencionar que la mayoría de diseños utilizados en la aplicación han sido diseñados utilizando ConstraintLayout.

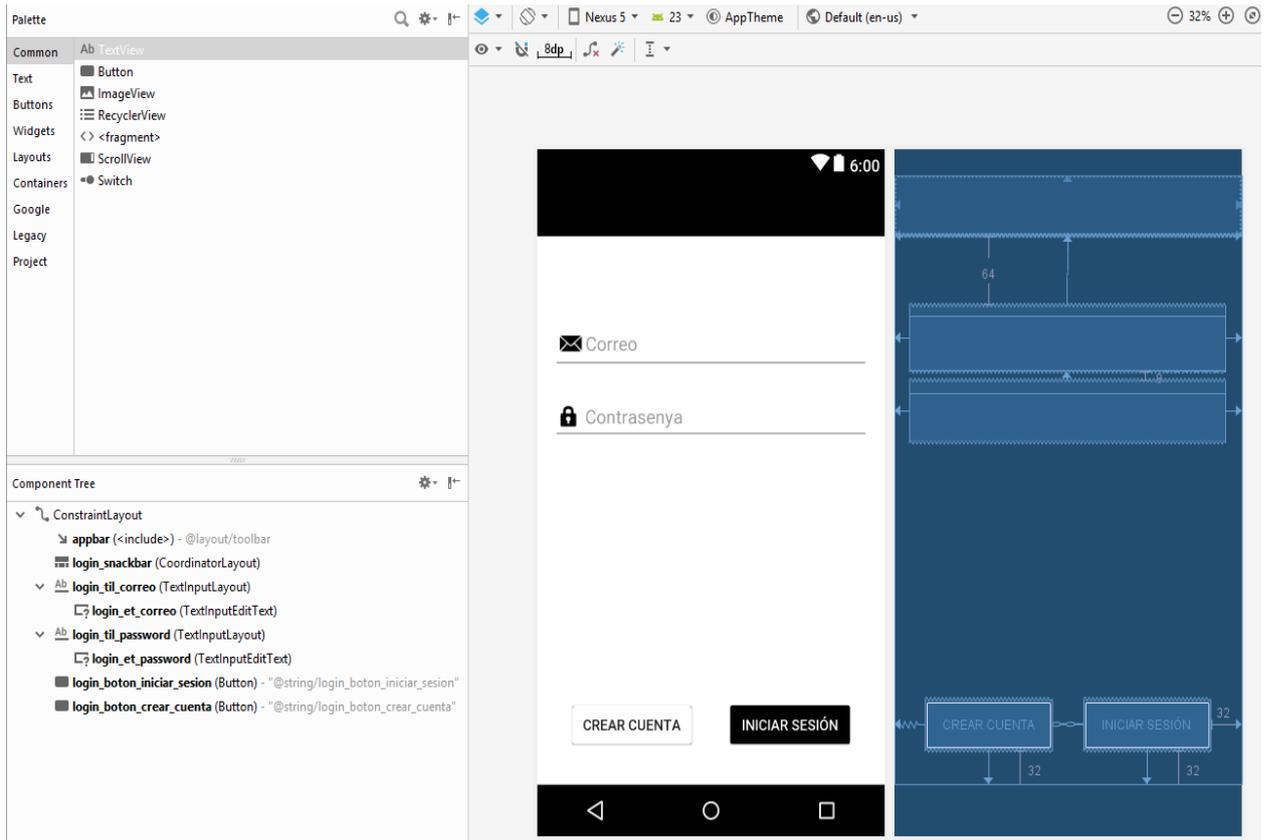


Figura 7.2.1.5: Editor Android Studio para crear diseños con ConstraintLayout

## Ciclo de vida

Es fundamental la implementación de métodos para administrar el ciclo de vida de las actividades de una aplicación Android. El ciclo de vida de una actividad puede verse alterado por la asociación entre diferentes actividades y por la pila de actividades.

Una actividad se puede encontrar en 3 estados:

- **Detenida:** La actividad se encuentra en segundo plano. Permanece el objeto Activity en memoria y se mantiene toda la información de estado, pero no se visualiza en pantalla. En caso de que el sistema necesite más memoria puede eliminarla.
- **Pausada:** La actividad es visible pero otra actividad se encuentra en primer plano y tiene el foco del usuario. El objeto Activity se conserva en memoria y se mantiene toda la información de estado. Puede ser eliminada de memoria en casos extremos de baja memoria.

- **En ejecución:** La actividad se encuentra en primer plano y tiene el foco del usuario.

Invocando el método *finish()* podemos forzar a eliminar actividades de memoria. Durante el ciclo de vida de una actividad los cambios se notifican a través de diferentes métodos Callback. Disponemos de los siguientes métodos:

<b>Método</b>	<b>Descripción</b>
<b>onCreate()</b>	Se utiliza cuando se crea la actividad por primera vez.
<b>onRestart()</b>	Se utiliza justo antes de volverse a iniciar una actividad que estaba detenida.
<b>onStart()</b>	Se utiliza justo antes de que la actividad se vuelva visible para el usuario.
<b>onResume()</b>	Se utiliza justo antes de que la actividad comience a interactuar con el usuario.
<b>onPause()</b>	Se utiliza cuando el sistema está a punto de reanudar una actividad. Su finalidad es confirmar datos sin guardar o detener servicios que consumen mucha CPU.
<b>onStop()</b>	Se utiliza cuando la actividad ya no es visible para el usuario.
<b>onDestroy()</b>	Se utiliza justo antes de que una actividad sea completamente eliminada.

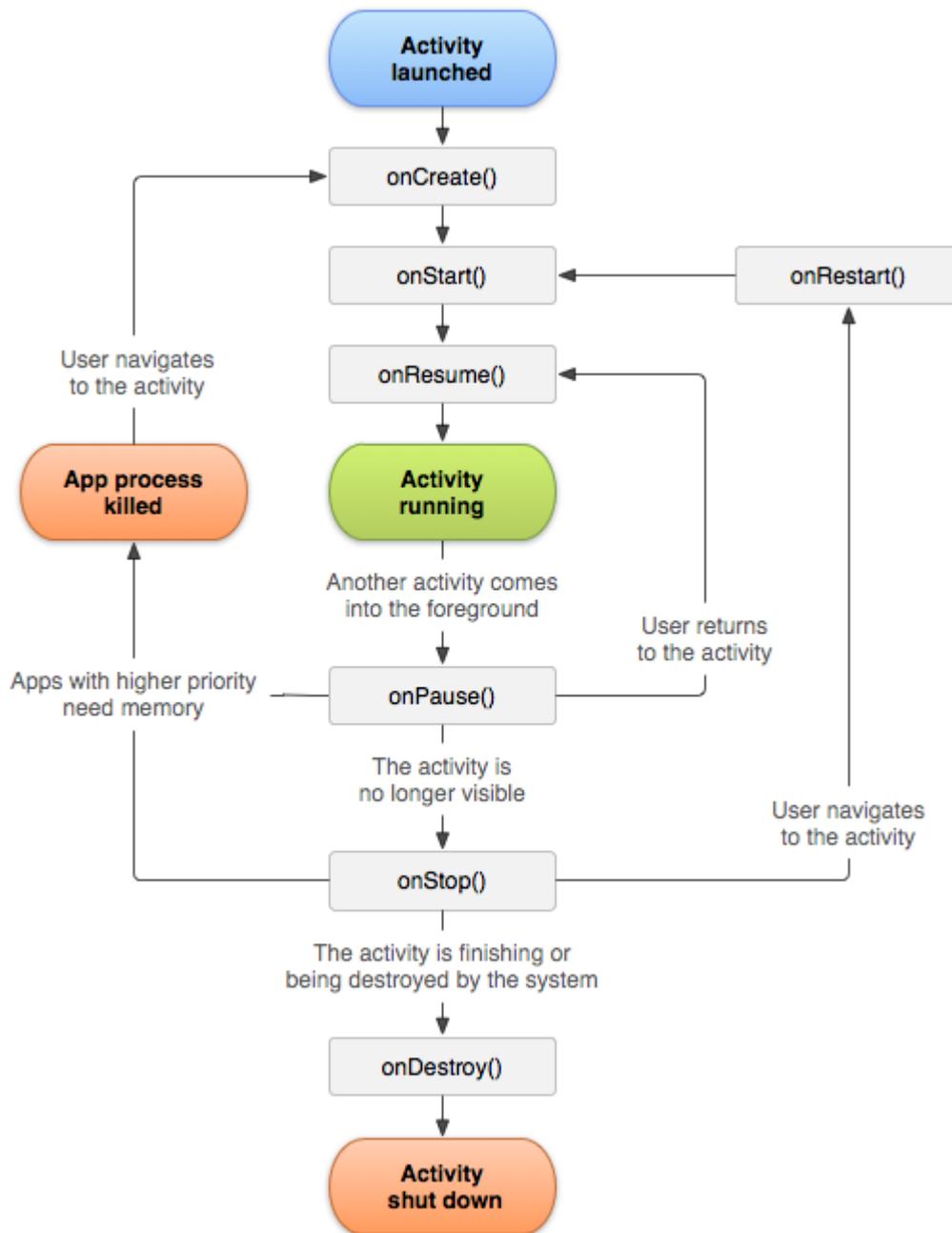


Figura 7.2.1.6: Ciclo de vida de una actividad

## 7.2.2 Android Studio

Para el desarrollo de la aplicación móvil se ha utilizado el entorno de desarrollo integrado (IDE) Android Studio [18], ya que se trata del IDE oficial para el desarrollo de aplicaciones para Android.

Se encuentra basado en IntelliJ IDEA, disponiendo de su potente editor de códigos y herramientas para desarrolladores de IntelliJ. Además, ofrece una serie de funcionalidades adicionales destinadas para el desarrollo de aplicaciones basadas en Android:

- Sistema de compilación basado en Gradle.
- Emulador de dispositivos móviles.

- Entorno unificado en el cual desarrollar para diferentes versiones de Android y distintos dispositivos.
- Instant Run mediante la cual se pueden realizar cambios en la aplicación sin necesidad de compilar un nuevo APK.
- Herramientas Lint para detectar problemas de rendimiento, usabilidad, compatibilidad de versiones, etc.

En concreto se ha utilizado la versión 3.1.2 de Android Studio.

Android Studio dispone de Gradle, un sistema de compilación basado en JVM (Java Virtual Machine). Se trata de un plugin, por lo tanto, permite su actualización y exportación de un proyecto a otro.

Facilita la tarea de configuración y personalización de la compilación del proyecto, gestiona las dependencias de forma cómoda (basado en Maven) y nos facilita la creación de diferentes versiones de la aplicación, como por ejemplo realizar múltiples versiones para distintos dispositivos móviles.

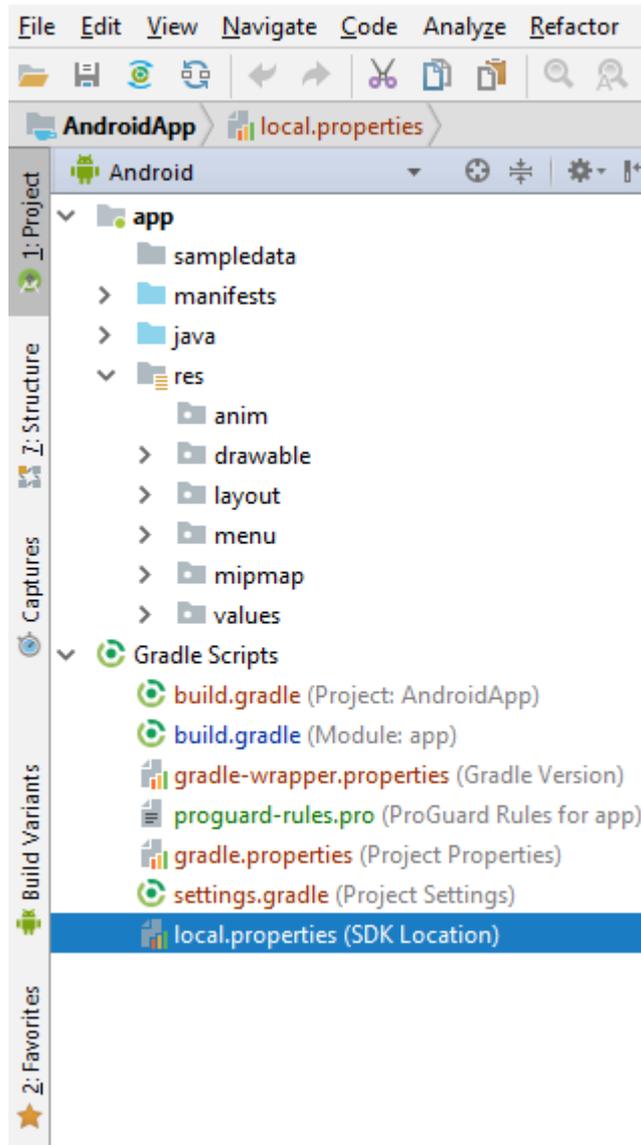


Figura 7.2.2.1: Estructura del proyecto en Android Studio

## 7.2.3 Retrofit + GSON

Para realizar las comunicaciones entre la aplicación móvil Android y el servicio Web basado en el estándar REST desarrollado en la aplicación servidor se ha utilizado la librería Retrofit. [19] Retrofit es un cliente REST para Android y Java, desarrollado por Square.

Permite agregar convertidores personalizados para mapear los datos obtenidos desde una API REST en formato XML o JSON en un objeto de una clase personalizada mediante un deserializador.

Para configurar la librería Retrofit en nuestro proyecto en primer lugar debemos incluir la dependencia de Retrofit en el archivo Gradle del proyecto de Android Studio:

```
implementation 'com.squareup.retrofit2:retrofit:2.3.0'
```

Tal y como se ha mencionado en la sección 7.1.12, el formato de datos a utilizado durante las comunicaciones entre cliente y servidor es JSON. Por lo tanto, se ha añadido una dependencia más que nos facilita el trabajo a la hora de trabajar con los JSON devueltos por el servidor.

GSON se trata de una librería de Java de serialización/deserialización para convertir objetos a JSON y viceversa:

```
implementation 'com.squareup.retrofit2:converter-gson:2.3.0'
```

Mediante Retrofit podemos aislar el manejador de peticiones en una interfaz en la cual se definen todas las llamadas a realizar en la aplicación. Utiliza anotaciones para definir los componentes de la API:

```
public interface ApiRest {  
  
    @POST("usuario/login")  
    Call<UsuarioLoginRespuesta> iniciarSesion(@Body UsuarioLoginPeticion  
        datosPeticion);  
  
    @GET("pais")  
    Call<List<Pais>> paises();  
  
    @Multipart  
    @POST("imagen/usuario")  
    Call<Imagen> subirImagenUsuario(@Part MultipartBody.Part file);  
  
    @DELETE("usuario/logout/{id}")  
    Call<GenericId> logout(@Path("id") Long id);  
  
    @PUT("usuario/{idUsuario}")  
    Call<GenericId> modificarPerfil(@Body UsuarioModificarPerfil  
        datosModificarPerfil, @Path("idUsuario") Long idUsuario);  
  
    . . .  
}
```

Otra funcionalidad que nos aporta Retrofit es el envío de las peticiones de forma asíncrona:

```
Call<List<Pais>> peticionRestPaises = apiRest.paises();
peticionRestPaises.enqueue(new Callback<List<Pais>>() {
    @Override
    public void onResponse(Call<List<Pais>> call, Response<List<Pais>> response)
    {
        . . .
    }
    @Override
    public void onFailure(Call<List<Pais>> call, Throwable t) {
        . . .
    }
});
```

Retrofit realiza la función de *wrapper* del cliente OkHttp. En realidad, se podría utilizar solo OkHttp para realizar las peticiones, pero esto implicaría que tengamos que crearnos las instancias AsyncTask o crear nuestro propio hilo de ejecución. Por otro lado, tampoco sería necesario utilizar GSON ya que disponemos de las clases nativas JSON Object y JSON Array de Java.

Pero el hecho de utilizar Retrofit + GSON nos realiza una gran simplificación de código y nos proporciona mayores facilidades para gestionar y crear las peticiones REST motivo por el cual se decidió a su estudio y utilización para el manejo de las peticiones a realizar a la API REST.

## 7.2.4 Glide

Glide [20] es una librería de Android de código abierto destinada a cargar imágenes, videos y GIFs animados.

Glide se encarga de descargar y guardar las imágenes en memoria cache, además aporta la funcionalidad de poder editar las imágenes tanto en medidas como en tamaño en disco.

El hecho de conservar las imágenes en cache proporciona un menor tiempo de respuesta a la hora de visualizarlas en la aplicación.

Para configurar Glide en el proyecto tan solo tenemos que añadir la dependencia en el archivo Gradle del proyecto Android Studio:

```
implementation 'com.github.bumptech.glide:glide:4.7.1'
```

A continuación, se muestra un ejemplo de la sencillez a la hora de obtener una imagen del servidor y cargarla en un componente mediante Glide:

```
RequestOptions options = new RequestOptions().centerCrop();

Glide.with(getContext())
    .load(Global.BASE_URL + Global.IMAGE_USER +
        UsuarioActual.getInstance().getId().toString() + "/" +
```

```
        imagenNombre)
    .apply(options)
    .into(imagenPerfil);
```

En el método load se indica la URL de la API REST encargada de proporcionar las imágenes. Con el método apply podemos definir una serie de características a la hora de mostrar la imagen en el componente como por ejemplo la posición de esta respecto al componente. Finalmente, en el método into introducimos el componente en el cual queremos que se visualice la imagen obtenida.

Uno de los motivos de la utilización de la librería Glide, es que crear una funcionalidad propia encargada de obtener y mostrar medios puede comportar mucho trabajo a la hora de desarrollarla. Hay que tener en cuenta aspectos como el cacheo de los medios obtenidos, decodificación, administración de las peticiones, etc.

Glide proporciona una serie de métodos que nos automatiza o facilita la gestión de ciertas funcionalidades.

## 7.2.5 Firebase

Firebase [21] es una plataforma destinada al desarrollo de aplicaciones móviles y aplicaciones web desarrollada en 2011. En el año 2014 fue adquirida por Google.

Google disponía de una serie de servicios los cuales cada uno de ellos tenía un SDK diferente. Como medida de integración respecto a esta divergencia nace Firebase, la integración de todos estos servicios a través de un único SDK, con el objetivo de facilitar el desarrollo y mantenimiento de las aplicaciones.

Podemos dividir los servicios que nos aporta esta plataforma en 3 categorías:

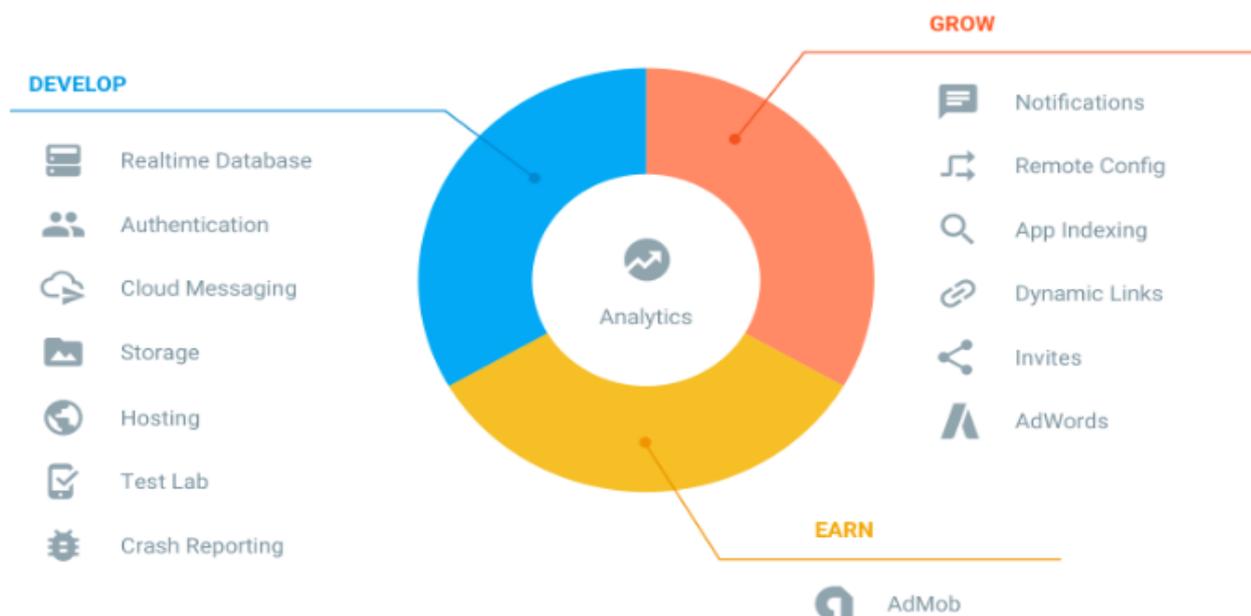


Figura 7.2.5.1: Servicios de la plataforma Firebase

- **Desarrollo (*develop*):** Firebase nos proporciona una gran variedad de servicios destinados a la creación de aplicaciones de una forma más ágil.
  - ***Real Time Database:*** Bases de datos en tiempo real que permiten almacenar y sincronizar los datos a través de una base de datos NoSQL alojada en la nube. Realiza la sincronización entre distintos dispositivos conectados.
  - ***Authentication:*** Servicio para gestionar el acceso a toda la infraestructura. Permite la administración de los usuarios de manera simple y segura mediante métodos de autenticación como correo electrónico y contraseña o proveedores externos como Google o Facebook.
  - ***Cloud messaging:*** Permite el envío de mensajes y notificaciones a los usuarios en distintas plataformas (Android, IOS y web) de forma gratuita. Los mensajes se pueden enviar de forma individual a un dispositivo, a un grupo de dispositivos o a segmentos de usuarios.
  - ***Storage:*** Almacenamiento para desarrolladores que necesitan guardar y servir contenido como imágenes, audio, video, etc.
  - ***Hosting:*** Proporciona un alojamiento estático y seguro de la aplicación.
  - ***Test Lab:*** Servicio que permite testear aplicaciones Android mediante pruebas automáticas y personalizadas en dispositivos virtuales y físicos alojados en el centro de datos de Google.
  - ***Crash Reporting:*** proporciona información en tiempo real para ayudar a diagnosticar y solucionar problemas en la aplicación.
  
- **Crecimiento de los usuarios (*grow*):** Firebase facilita el crecimiento y la fidelidad de los usuarios mediante distintos servicios.
  - ***Notifications:*** Mediante la consola de Firebase se pueden diseñar y enviar a los usuarios notificaciones con el objetivo de fomentar el uso de la aplicación.
  - ***Remote Config:*** Permite personalizar la manera en que la aplicación se presenta para cada usuario. Como por ejemplo mostrar contenido personalizado a determinados usuarios.
  - ***App indexing:*** Servicio destinado a atraer a los usuarios a la aplicación mediante la indexación de la aplicación con el objetivo de que cuando busquen contenido relacionado les aparezca la posibilidad de utilizar o instalar la aplicación.
  - ***Dynamic Links:*** Vínculos directos para brindar una experiencia del usuario personalizada.
  - ***Invites:*** Envío de invitaciones por correo electrónicos o SMS compartiendo la aplicación directamente con los contactos destinatarios.
  
- **Monetización de la aplicación (*Earn*):** Servicio que utiliza tecnologías AdMob con la finalidad de monetizar y rentabilizar las aplicaciones mediante anuncios.
  
- **Firestore Analytics:** Herramienta que ocupa el lugar central en Firestore. Mediante Analytics podemos obtener información de uso por parte de los usuarios de forma gratuita.

Mediante Android Studio se ha realizado la configuración de Firebase en la aplicación Android desarrollada. El IDE nos ofrece el servicio *Firebase Assistant* que nos ayuda a conectar Firebase al proyecto. El asistente nos genera automáticamente las dependencias en el archivo Gradle de nuestro proyecto y nos proporciona una interface para configurar el proyecto en la consola de Firebase. En la figura 7.2.5.2 podemos visualizar la aplicación Android desarrollada integrada en Firebase. [22]



Figura 7.2.5.2: Proyecto Android integrado en Firebase

Para el desarrollo de la aplicación se ha utilizado el servicio de Firebase Cloud Messaging para el manejo y envío de notificaciones push y el servicio de base de datos en tiempo real para gestionar los mensajes en los foros de cada evento deportivo.

## 7.2.6 Firebase Realtime Database

Con este servicio de Firebase [23] podemos almacenar y sincronizar datos con una base de datos NoSQL alojada en la red. Los datos se sincronizan con todos los dispositivos clientes en tiempo real y se mantienen disponibles incluso cuando la aplicación no tiene conexión.

Los datos se almacenan en formato JSON. Todos los clientes comparten una instancia de Realtime Database y reciben actualizaciones automáticas con los datos más recientes.

En la aplicación móvil, por cada evento deportivo se crea un foro el cual puede ser privado o público. Si el foro del evento es público cualquier usuario registrado en la aplicación que acceda al detalle del evento puede visualizar y participar en el foro. En cambio, si el foro se crea como privado, sólo los usuarios registrados en el evento pueden visualizar y participar en éste.

En un principio, se quiso manejar el almacenado y envío de los mensajes del foro mediante la base de datos relacional MySQL. Pero, finalmente, se decidió almacenar la información de los mensajes de los foros en la base de datos que nos proporciona Firebase.

El principal motivo de este cambio es la sincronización de datos en tiempo real que nos proporciona el servicio Realtime Database que, de forma sencilla, se encarga de mantener el contenido del foro actualizado en todo momento. De otro modo, se hubiesen tenido que desarrollar otras alternativas como el uso de webSockets para mantener el contenido del foro actualizado en todo momento.

En la consola de Firebase podemos consultar y manipular los diferentes datos persistidos de los foros de los eventos tal y como muestra la siguiente figura.

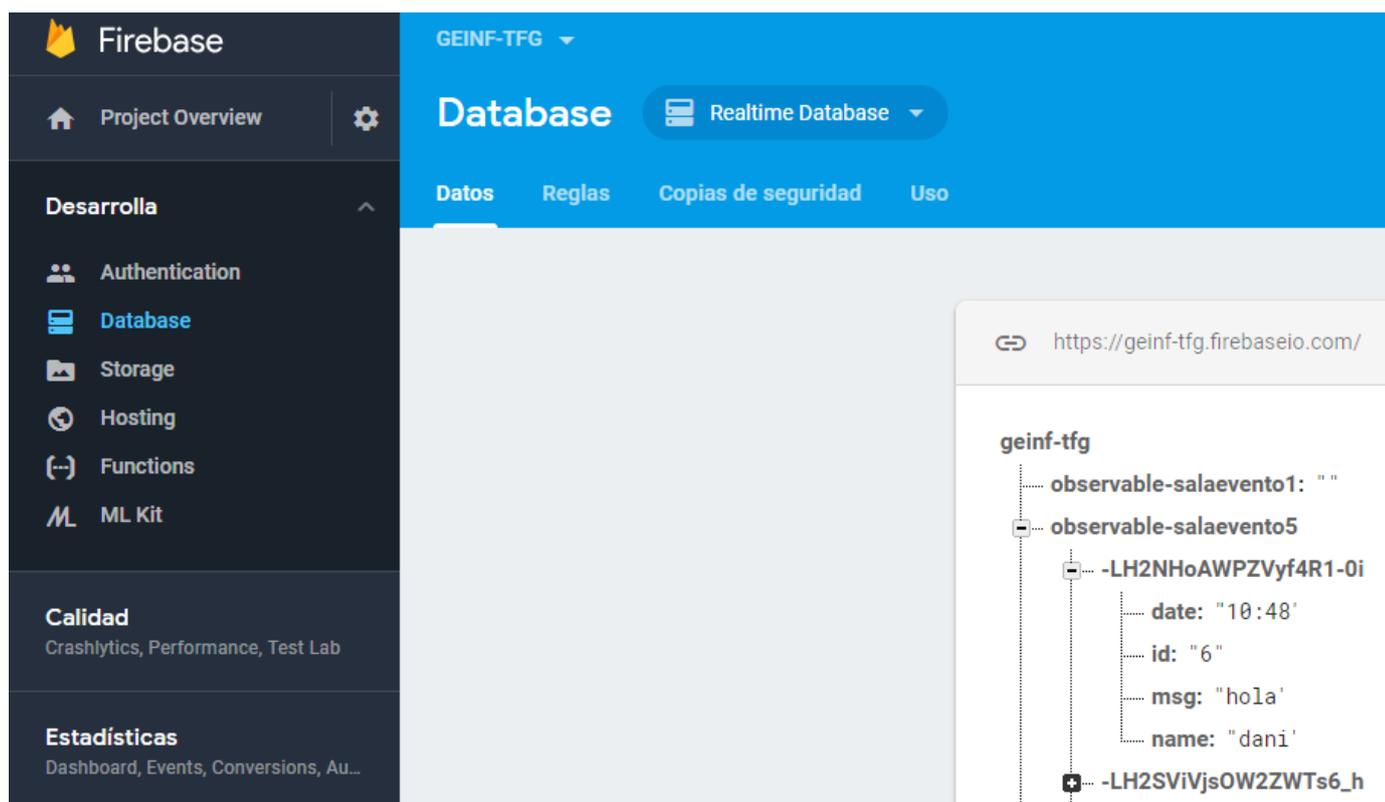


Figura 7.2.6.1: Base de datos de Firebase para el almacenado del contenido de los foros

## 7.2.7 Firebase Cloud Messaging

Firebase Cloud Messaging (FCM) [24] es una solución de mensajería multiplataforma que nos permite enviar mensajes de forma gratuita y segura. Ofrece una amplia variedad de funciones y opciones de mensajería.

Con FCM podemos enviar dos tipos de mensajes a los clientes:

- **Mensajes de notificación:** FCM muestra automáticamente el mensaje en los dispositivos del usuario final en nombre de la aplicación cliente.
- **Mensajes de datos:** La aplicación cliente es la responsable de procesar los mensajes recibidos.

En este proyecto se han utilizado los mensajes de notificación para enviar notificaciones push a los dispositivos móviles informando de ciertos cambios que puedan interesar al usuario sin necesidad de tener que acceder a la aplicación.

En el momento en que un usuario inicia sesión o se registra a la aplicación desde un dispositivo, Firebase genera y registra un *token* único que identifica el dispositivo físico. Este registro lo almacenamos en una base de datos ligado al usuario.

Posteriormente, cuando sucede un evento en el cual hay que notificar a los usuarios afectados, en la aplicación servidor se genera una petición POST mediante un archivo JSON informando al servicio de Firebase Cloud Messaging el token/s de los destinatarios. A continuación, se muestra un ejemplo del contenido de una petición de envío de una notificación:

```
{
  "message": {
    "token": "bk3RNwTe3H0:CI2k_HHwgIpoDKCIZvvDMExUdFQ3P1...",
    "notification": {
      "title": "Título de la notificación",
      "body": "Contenido de la notificación"
    }
  }
}
```

Finalmente, el servicio de google se encarga de enviar los mensajes de notificación a los usuarios finales. En la Figura 7.2.7.1 se muestra el proceso de envío de mensajes de notificación utilizado:

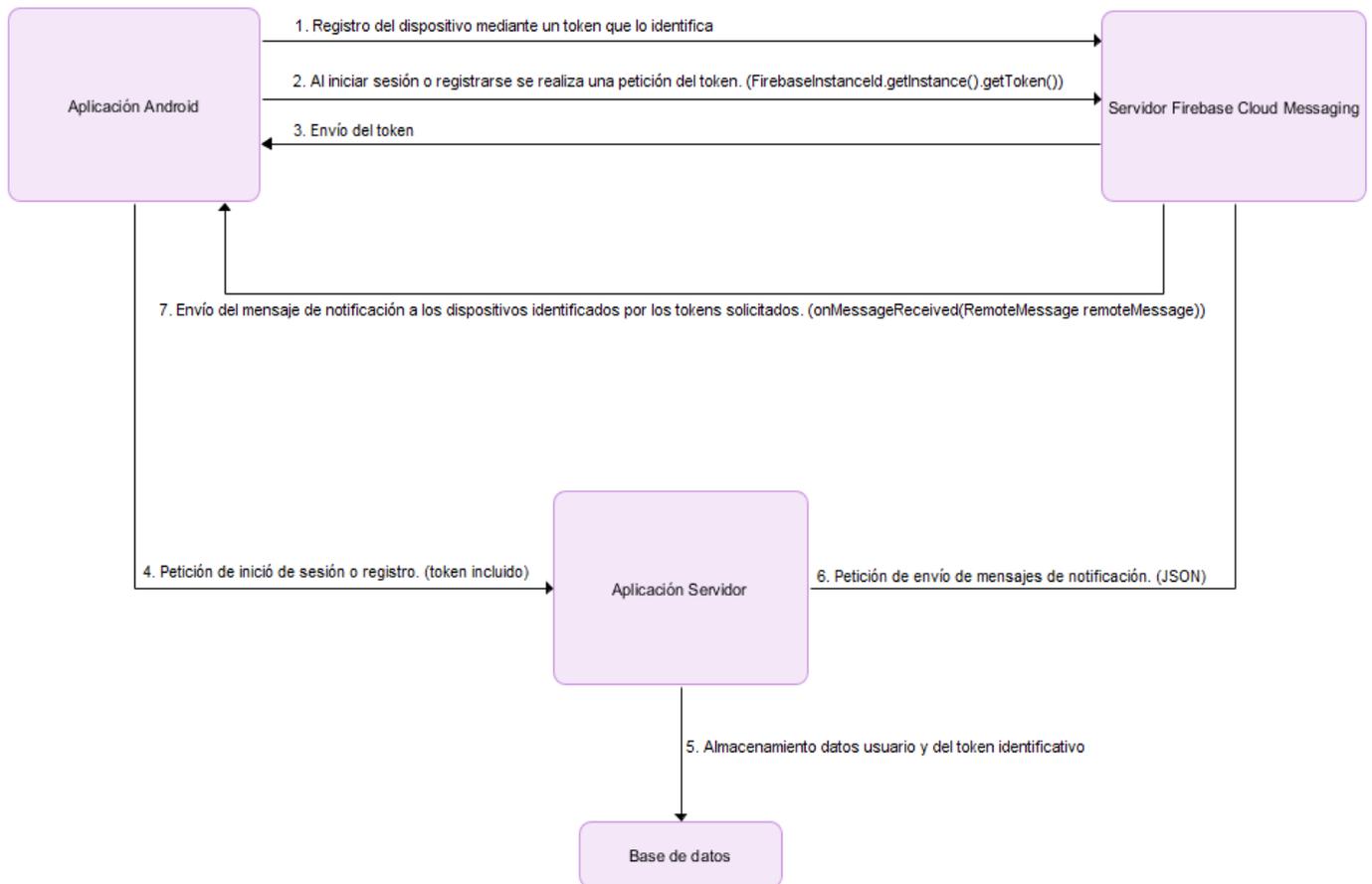


Figura 7.2.7.1: Arquitectura Firebase Cloud Messaging

Mediante el método `FireBaseInstanceId.getInstance.getToken()` obtenemos el token que identifica al dispositivo físico sobre el cual hemos iniciado sesión en la aplicación.

Por otro lado, a la hora de recibir los mensajes de notificación en la aplicación cliente Android podemos definir el comportamiento y el estilo visual de las notificaciones sobrescribiendo el método `onMessageReceived(RemoteMessage message)` en una clase que extienda de la clase `FireBaseMessagingService`.

## 7.3 Otros

En este apartado se mencionan y definen los programas utilizados para la elaboración de la documentación del proyecto.

### 7.3.1 GanttProject



Figura 7.3.1.1: Logo GanttProject

Se trata de un programa código abierto con licencia GPL desarrollado en el lenguaje de programación Java. Su principal funcionalidad es la administración de proyectos mediante el uso de diagramas de Gantt. Mediante este tipo de diagramas se permite visualizar el desglose de tareas o actividades programadas durante el desarrollo de un proyecto de forma cronológica junto con su duración.

En el desarrollo de la memoria se ha utilizado este software para plasmar en diagramas la planificación inicial y la planificación final de las tareas del proyecto tal y como se puede ver en el *capítulo 4*.

Se ha utilizado este software ya que es totalmente gratuito, dispone de una interface fácil de utilizar y está exclusivamente dedicado a la creación de diagramas de Gantt.

### 7.3.2 Visual Paradigm



Figura 7.3.2.1: Logo Visual Paradigm

Visual Paradigm se trata de una herramienta UML CASE (Ingeniería del software asistida por computación.)

Ha sido desarrollada para soportar el ciclo de vida completo del proceso de desarrollo de un software mediante la representación de todo tipo de diagramas.

Se ha utilizado la versión Community Edition que es gratuita para usos no comerciales.

Se ha escogido esta herramienta ya que dispone de los recursos para generar un gran número de diagramas como diagramas de clase, diagramas de caso de usos, diagrama de secuencias, diagrama de comunicaciones, diagrama de actividades, etc.

En conclusión, en un mismo software disponemos de todos los tipos de diagramas a diseñar en la memoria del proyecto.

### 7.3.4 Microsoft Word



Figura 7.3.4.1: Logo Microsoft Word

Software creado por Microsoft integrado en el paquete de ofimática denominado Microsoft Office.

Está orientado al procesamiento de textos y actualmente es el procesador de texto más popular mundialmente.

Se ha utilizado para para escribir la memoria del proyecto ya que se trata de un software el cual ya había sido utilizado y tiene todas las funcionalidades que se necesitan para la elaboración de la memoria.

---

## 8. Análisis y diseño del sistema

---

En este capítulo, se realiza un análisis de los requisitos funcionales mediante diagramas de caso de uso y sus fichas correspondientes. Por otra parte, se presenta el diseño realizado para cumplir dichos requerimientos. Tanto el análisis como el diseño, siguen la metodología definida en *capítulo 3* de dividir las tareas a realizar en partes diferenciadas.

### 8.1 Análisis de los requisitos del sistema

Una vez detectados y definidos los requisitos funcionales (*Capítulo 6*), mediante diagramas de casos de uso, mostraremos de forma gráfica la funcionalidad del sistema y como el usuario interacciona con el mismo. Adicionalmente, para una mayor comprensión del diagrama de casos de uso se complementa con su ficha de caso de uso.

Mediante los diagramas de casos de uso, se define la secuencia de interacciones que se desarrollan entre el sistema y el actor principal en respuesta a un evento iniciado por el actor sobre el sistema.

#### 8.1.1 Actores del sistema

Se identifica un actor como cualquier entidad externa que interactúa con éste y le requiere una funcionalidad. Esto incluye a operadores humanos, sistemas externos o entidades abstractas como el tiempo.

En el caso de la aplicación desarrollada, sólo existe un tipo de Actor que corresponde a cualquier usuario que disponga la aplicación y la ejecute. La principal diferenciación entre los usuarios a la hora de interactuar con el sistema es si han iniciado sesión o no en la aplicación.



Figura 3.1.1.1: Actores del sistema

#### 8.1.2 Partes del sistema

Una vez identificados los requisitos funcionales y estudiadas las herramientas a utilizar en el desarrollo del proyecto, se analizan las diferentes partes que tendrá la aplicación durante el proceso de implementación tal y como se comenta en la metodología utilizada.

Cuando un usuario inicia la aplicación en su dispositivo móvil, dispone de dos opciones, iniciar sesión o registrarse.

Para un usuario que ha iniciado sesión en el sistema dispone de una serie de pantallas sobre las cuales puede interactuar:

- **Principal:** Pantalla de inicio a la cual se accede una vez un usuario inicia sesión o se da de alta en el sistema.  
En esta pantalla, se muestran, si los hay, los eventos deportivos que correspondan al municipio y categorías deportivas favoritas introducidos en el momento en que se registró en la aplicación, pudiendo acceder al detalle de los mismos.  
En esta pantalla se dispone de un menú lateral que nos permite crear un evento deportivo, acceder al buscador de eventos deportivos, acceder a la información del perfil, visualizar los eventos del usuario, acceder al menú de configuración, cerrar sesión y modificar el perfil.
- **Buscador:** En esta pantalla se muestra un formulario que permite aplicar ciertos filtros para realizar una búsqueda personalizada entre todos los eventos deportivos dados de alta en la aplicación. Una vez se aplican los filtros, se muestra la pantalla principal listando los eventos resultantes de la búsqueda.
- **Crear evento:** Pantalla que muestra un formulario que permite crear un evento deportivo personalizado.
- **Perfil usuario:** En esta pantalla se muestran los datos introducidos en el momento en que se realizó el registro en la aplicación. Además, muestra un listado de los eventos creados por el usuario con la posibilidad de acceder a su detalle o suspenderlos.
- **Modificar perfil usuario:** Se permite modificar los datos introducidos en el momento en el que el usuario se dio de alta en el sistema.
- **Mis eventos:** Permite acceder a las pantallas que muestran los eventos creados por el usuario pudiendo suspenderlos o acceder a su detalle, eventos en el que el usuario está participando permitiendo acceder a su detalle y darse de baja, y a los eventos en que el usuario está en cola para participar pudiendo acceder a su detalle o darse de baja de la cola.
- **Configuración notificaciones:** En esta pantalla se permiten activar/desactivar las notificaciones disponibles en la aplicación.
- **Detalle evento:** permite acceder a las pantallas que muestran el detalle de un evento deportivo. Estas pantallas son los datos principales del evento, la lista de los usuarios que participan, lista de los usuarios en cola, foro del evento y un mapa que muestra el lugar en donde se celebra el evento.  
En caso de ser el propietario del evento se permite dar de baja a los usuarios registrado o que estén en cola. En caso de no ser el propietario del evento se permite darse de baja o de alta en el evento. En ambos casos se permite acceder al perfil de los usuarios.

### 8.1.3 Diagramas y fichas de casos de uso

En este apartado se muestran los diagramas y fichas de casos de uso de las partes en que se ha dividido el proyecto.

#### 8.1.3.1 Registro

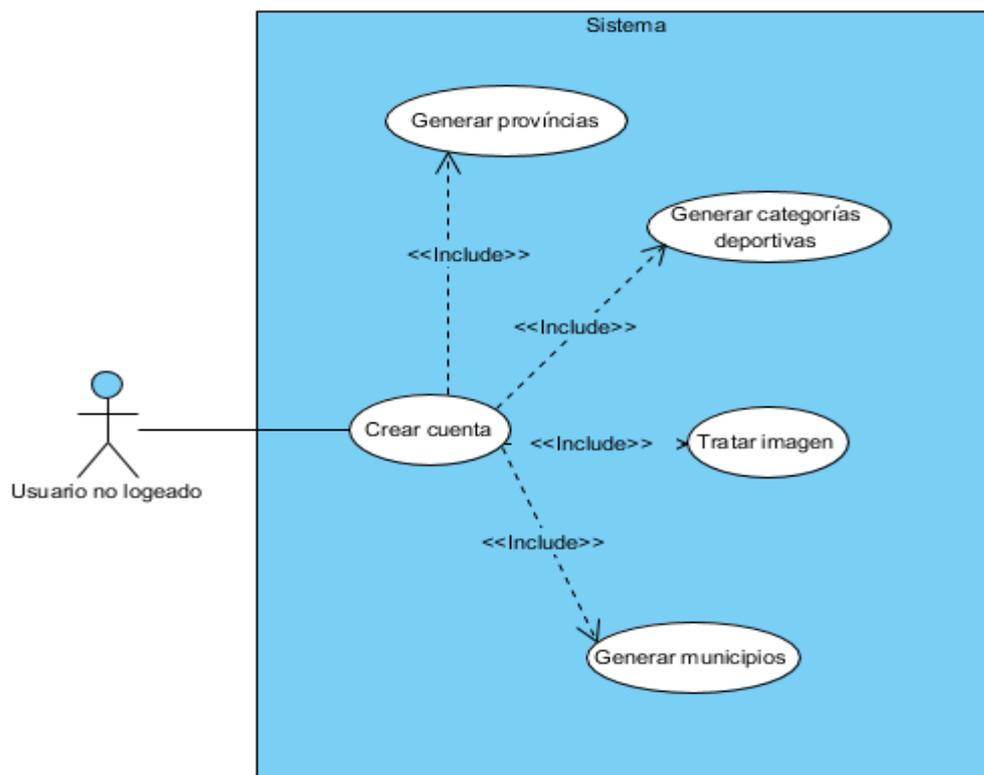


Figura 8.1.3.1.1: Diagrama caso de uso registro

Caso de uso Registrarse	
Descripción	Usuario puede registrarse en la aplicación.
Actor	Usuario no logeado.
Precondición	Usuario no ha iniciado sesión.
Flujo principal	<ol style="list-style-type: none"> <li>1. El usuario abre la aplicación.</li> <li>2. Toca el botón crear cuenta.</li> <li>3. El sistema muestra el formulario de registro listando provincias, municipios y categorías deportivas.</li> <li>4. Usuario rellena el formulario introduciendo nombre, apellidos, Nick, correo electrónico, contraseña, provincia, municipio, categorías deportivas e imagen de perfil.</li> <li>5. Usuario toca el botón crear.</li> <li>6. El sistema verifica los datos introducidos.</li> <li>7. Se redirige al usuario a la página principal.</li> </ol>
Flujo alternativo	En caso de que algún campo del formulario esté sin rellenar o la cuenta de correo o el Nick ya están registrados, se muestra el mensaje de error en el formulario.
Postcondición	Usuario registrado correctamente en base de datos con contraseña cifrada, token de Firebase generado e imagen de perfil guardada.

### 8.1.3.2 Imagen de perfil

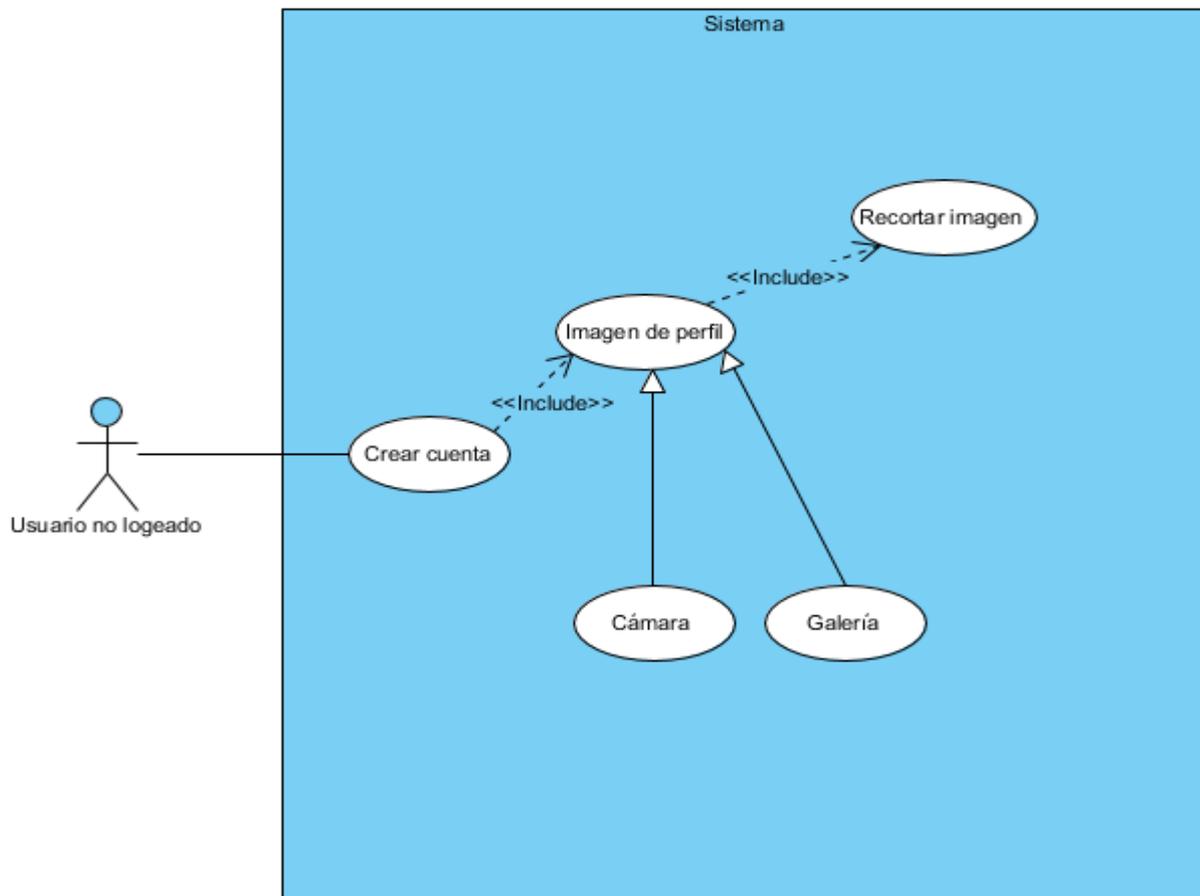


Figura 8.1.3.2.1: Diagrama caso de uso Imagen de perfil

Caso de uso		Imagen de perfil
Descripción	Usuario puede subir una imagen de perfil.	
Actor	Usuario no logeado.	
Precondición	Usuario no ha iniciado sesión.	
Flujo principal	<ol style="list-style-type: none"> <li>1. El usuario abre la aplicación.</li> <li>2. Toca el botón crear cuenta.</li> <li>3. El sistema muestra el formulario de registro.</li> <li>4. Usuario toca imagen de perfil por defecto.</li> <li>5. Sistema muestra menú para escoger de donde subir imagen de perfil (cámara o galería).</li> <li>6. Sistema verifica si tiene permisos para acceder a la galería del dispositivo o a la cámara.</li> <li>7. Usuario selecciona una imagen.</li> <li>8. El sistema permite recortar la imagen.</li> <li>9. El sistema muestra la imagen de perfil seleccionada.</li> </ol>	
Flujo alternativo	En caso de que no se tengan permisos para acceder a la cámara o a la galería del dispositivo se muestra la imagen de perfil por defecto.	
Postcondición	Se muestra en el formulario de registro la imagen seleccionada por el usuario.	

### 8.1.3.3 Iniciar sesión

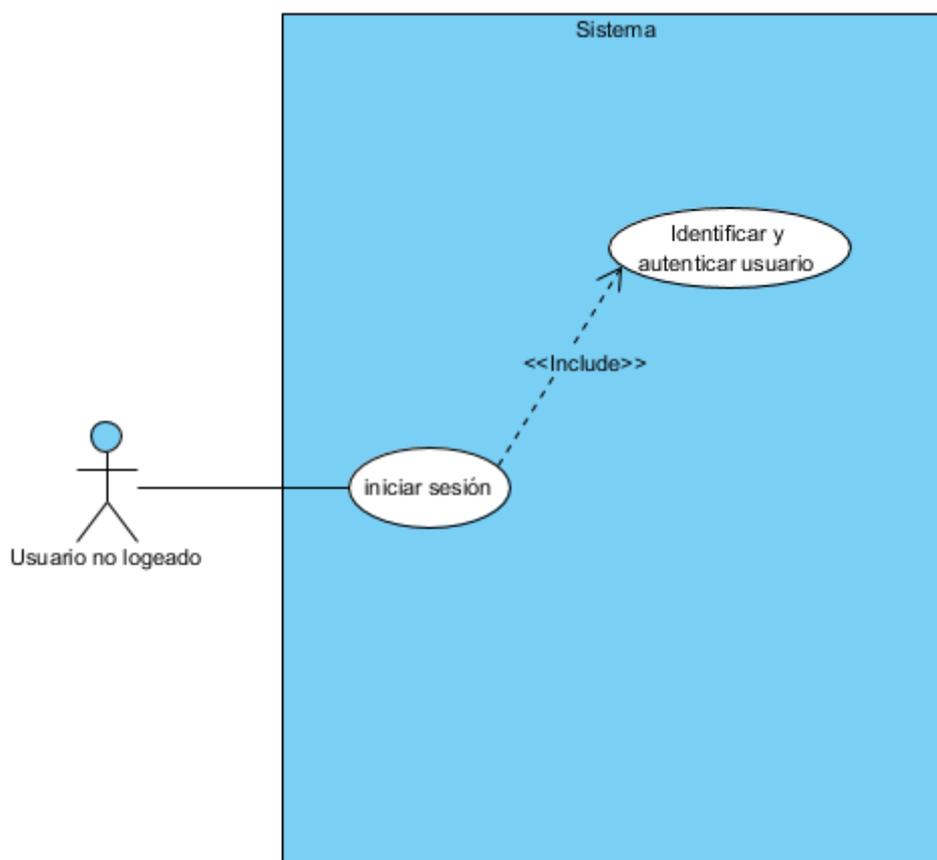


Figura 8.1.3.3.1: Diagrama caso de uso iniciar sesión

Caso de uso		Iniciar sesión
Descripción	Usuario puede iniciar sesión para acceder a la aplicación	
Actor	Usuario no logeado.	
Precondición	Usuario no ha iniciado sesión.	
Flujo principal	<ol style="list-style-type: none"> <li>1. El usuario abre la aplicación.</li> <li>2. Usuario introduce dirección de correo electrónico y contraseña</li> <li>3. Usuario toca el botón iniciar sesión.</li> <li>4. El sistema verifica los datos.</li> <li>5. Se redirige al usuario a la página principal.</li> </ol>	
Flujo alternativo	En caso de que algún campo del formulario esté sin rellenar o la cuenta de correo o la contraseña sean incorrectos, se muestra el mensaje de error.	
Postcondición	Se genera token identificativo de Firebase y usuario tiene acceso a la aplicación.	

### 8.1.3.4 Principal

El caso de uso Principal, corresponde a la funcionalidad que ejecuta el sistema en el momento posterior a que un usuario inicia sesión o se registra en la aplicación. Corresponde a la interfaz gráfica principal de la aplicación. Contiene el menú lateral con la resta de funcionalidades a realizar en la aplicación. En la figura 8.1.3.3.1 se muestra el diagrama de casos de uso que corresponde a las acciones que lleva el sistema una vez autenticados en la aplicación.

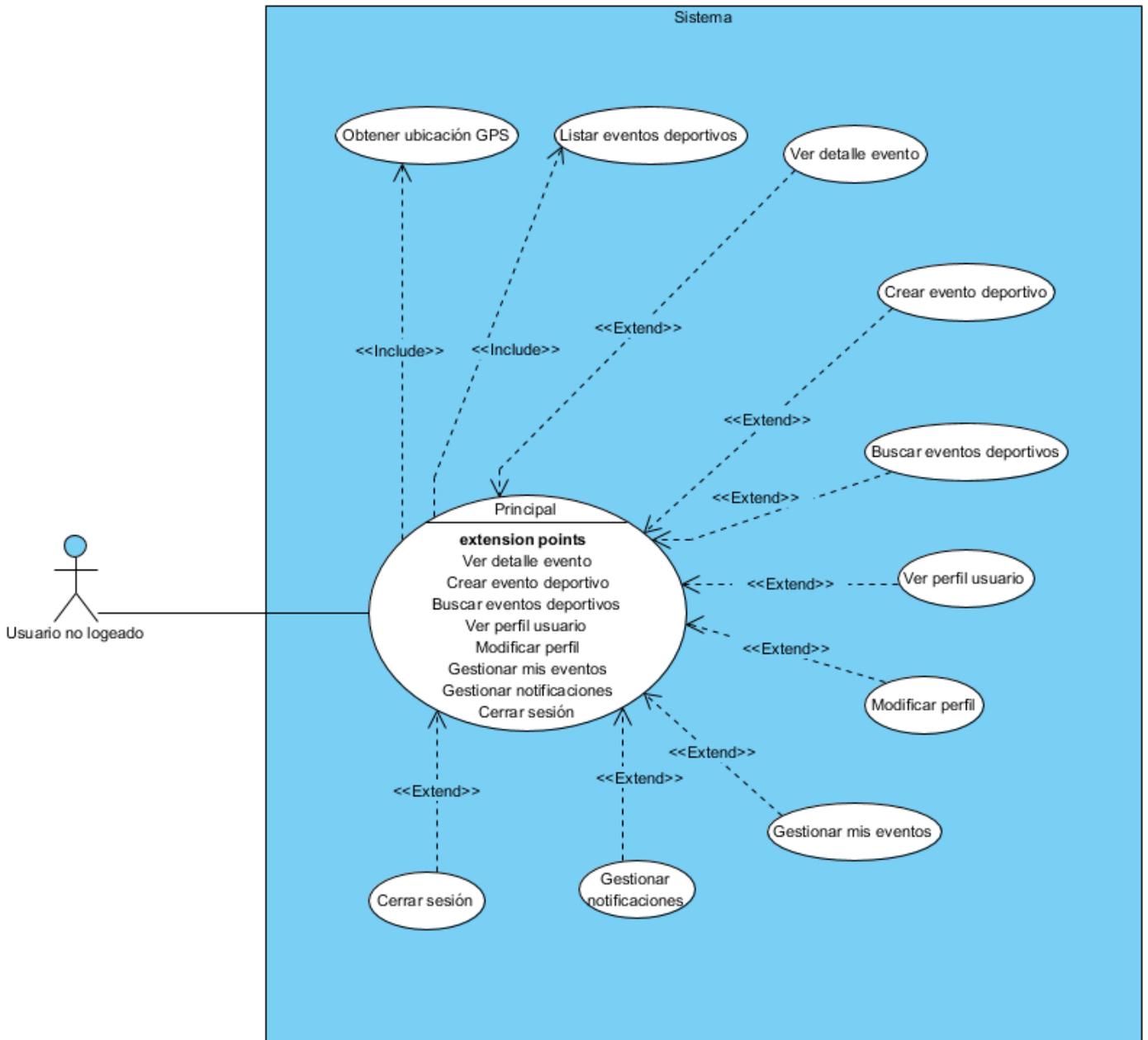


Figura 8.1.3.4.1: Diagrama caso de uso principal

Caso de uso Principal	
Descripción	Usuario puede visualizar los eventos deportivos listados y acceder a su detalle, publicar un evento deportivo, buscar eventos, ver su perfil, modificar su perfil, cerrar sesión, ver sus eventos y administrar las notificaciones.
Actor	Usuario logeado.
Precondición	Usuario ha iniciado sesión en la aplicación.
Flujo principal	<ol style="list-style-type: none"> <li>1. El sistema revisa si tiene permisos para acceder al GPS del dispositivo móvil: <ol style="list-style-type: none"> <li>a. Si tiene permisos y el dispositivo tiene el sensor GPS activado registra la ubicación GPS.</li> <li>b. Si tiene permisos y el dispositivo no tiene el sensor GPS recomienda activarlo.</li> <li>c. Si no tiene permisos, solicita al usuario permiso para utilizar el sensor GPS <ol style="list-style-type: none"> <li>i. Si se conceden permisos y el sensor GPS está activado se registra la ubicación GPS.</li> <li>ii. Si se conceden permisos y el sensor GPS está desactivado se recomienda activarlo.</li> <li>iii. Si se deniega el permiso no se registra ubicación GPS.</li> </ol> </li> </ol> </li> <li>2. El sistema busca los eventos deportivos, no finalizados y no suspendidos, que coincidan con el municipio del usuario y con sus deportes favoritos seleccionados: <ol style="list-style-type: none"> <li>a. Si hay resultados, se muestra un listado con los eventos deportivos resultantes</li> <li>b. Si no hay resultados, se muestra un mensaje informando de que se pueden modificar sus preferencias por defecto.</li> </ol> </li> </ol>
Flujo alternativo	Error al acceder a la aplicación, se cierra sesión y se para la ejecución de la aplicación.
Postcondición	Ubicación registrada en base de datos y eventos deportivos listados.

Las diferentes funcionalidades extendidas del caso de uso Principal se definen en los siguientes apartados.

### 8.1.3.5 Cerrar sesión

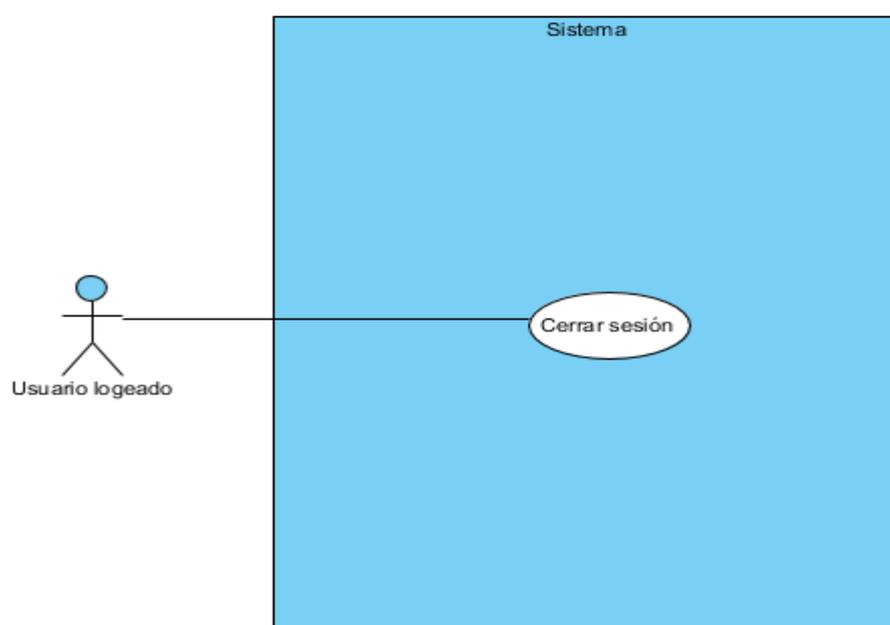


Figura 8.1.3.5.1: Diagrama caso de uso cerrar sesión

Caso de uso		Cerrar sesión
Descripción	Usuario puede cerrar sesión de la aplicación.	
Actor	Usuario logeado.	
Precondición	Usuario ha iniciado sesión.	
Flujo principal	<ol style="list-style-type: none"> <li>1. El usuario abre el menú lateral.</li> <li>2. El usuario selecciona la opción cerrar sesión.</li> <li>3. El sistema elimina la sesión del usuario.</li> <li>4. El sistema redirige al usuario a la página de inicio de sesión.</li> </ol>	
Flujo alternativo	Ninguno	
Postcondición	El sistema elimina la sesión actual del usuario. Posteriormente, redirige al usuario a la pantalla de inicio de sesión.	

### 8.1.3.6 Crear evento deportivo

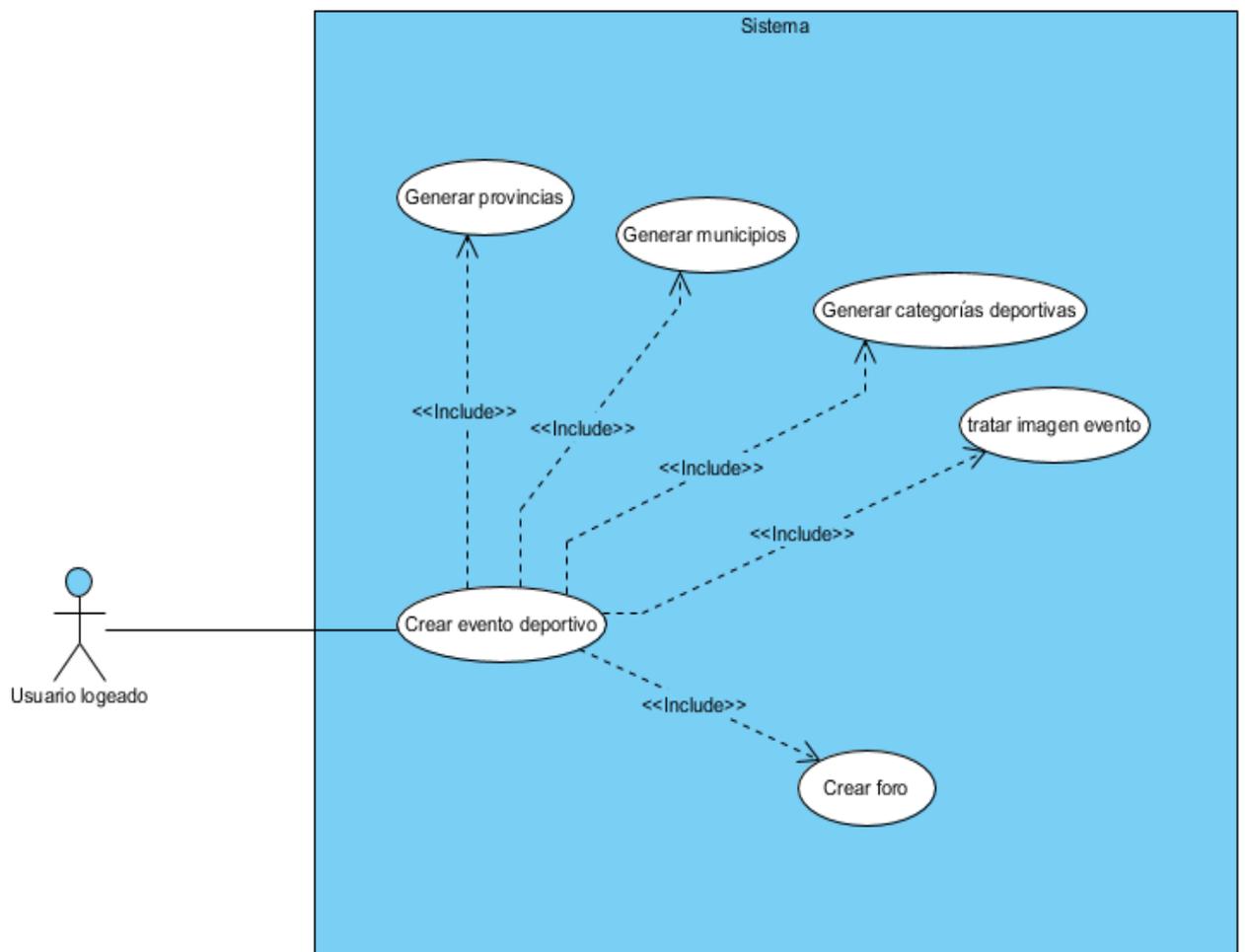


Figura 4.1.3.6.1: Diagrama caso de uso crear evento deportivo

Caso de uso		Crear evento deportivo
Descripción	Usuario crea un evento deportivo.	
Actor	Usuario logeado.	
Precondición	Usuario ha iniciado sesión.	
Flujo principal	<ol style="list-style-type: none"> <li>1. El usuario abre el menú lateral.</li> <li>2. El usuario selecciona la opción crear evento.</li> <li>3. El sistema muestra el formulario para crear un evento deportivo cargando el listado de categorías deportivas, provincias y municipios.</li> <li>4. El usuario debe insertar los datos: título del evento, descripción del evento, seleccionar categoría deportiva, número de participantes, duración estimada, privacidad del foro, fecha y hora de celebración, imagen del evento y ubicación del evento.</li> <li>5. Usuario selecciona la ubicación del evento: <ol style="list-style-type: none"> <li>a. Si selecciona el ícono Ubicación el sistema muestra un mapa de Google con posibilidad de seleccionar ubicación exacta.</li> <li>b. Si selecciona el ícono Municipio el sistema lista las provincias y municipios generados al iniciar formulario.</li> </ol> </li> <li>6. Usuario pulsa el botón crear.</li> <li>7. El sistema valida los datos introducidos.</li> <li>8. El sistema crea el foro en la base de datos de Firebase.</li> <li>9. El sistema redirige al usuario a la pantalla Principal.</li> </ol>	
Flujo alternativo	Si algún campo del formulario está vacío muestra un mensaje de error informando.	
Postcondición	El sistema crea la tabla del foro del evento en la base de datos externa de Firebase, registra en la base de datos local los datos del evento creado y su imagen asociada.	

### 8.1.3.7 Buscar eventos deportivos

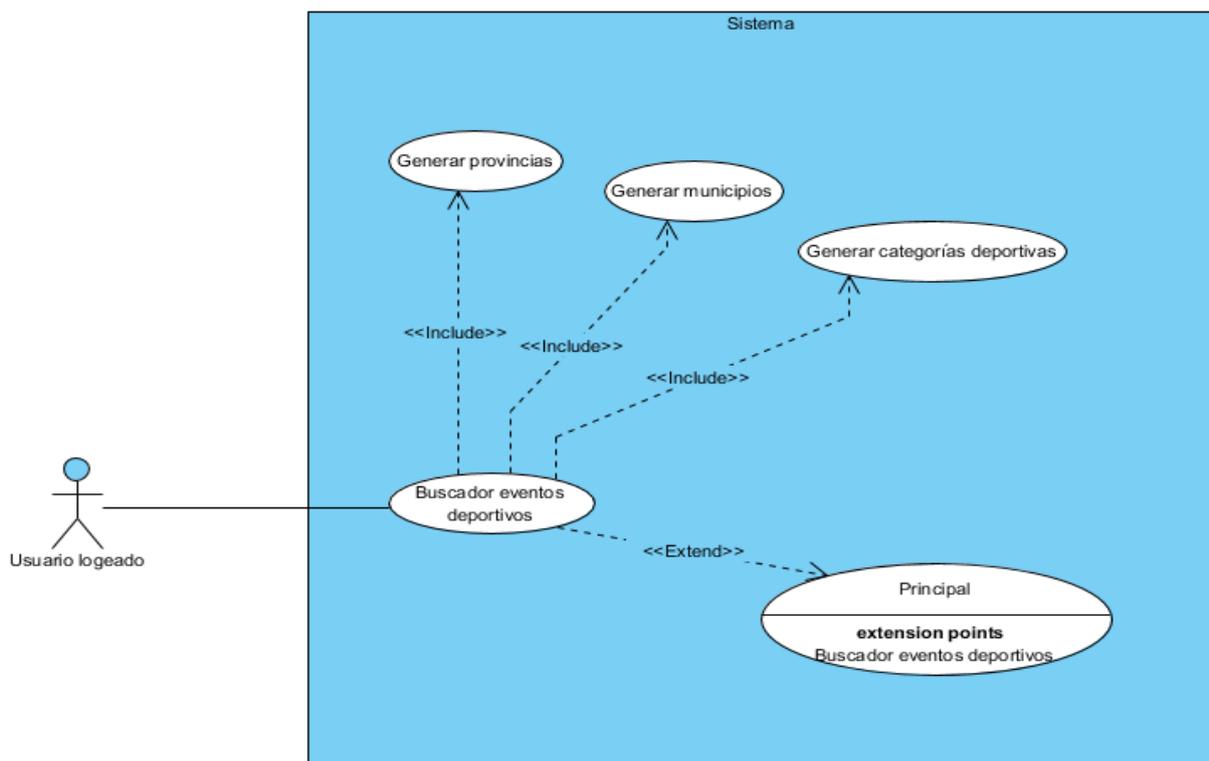


Figura 5.1.3.7.1: Diagrama caso de uso buscar eventos deportivos

<b>Caso de uso</b>		<b>Buscar eventos deportivos</b>
Descripción	Usuario puede buscar eventos deportivos.	
Actor	Usuario logeado.	
Precondición	Usuario ha iniciado sesión.	
Flujo principal	<ol style="list-style-type: none"> <li>1. El usuario toca la lupa de la pantalla principal o accede a la opción buscar eventos del menú lateral en la pantalla Principal.</li> <li>2. El sistema muestra el formulario filtrar la búsqueda de eventos deportivos generando el listado de categorías deportivas, provincias y municipios.</li> <li>3. El usuario puede insertar los datos: título del evento, categorías deportivas a buscar, fecha de celebración del evento y ubicación del evento.</li> <li>4. Usuario selecciona ubicación del evento: <ol style="list-style-type: none"> <li>a. Si selecciona el ícono Cerca de Mi puede definir una distancia máxima según su ubicación actual, registrada mediante el dispositivo GPS al iniciar sesión.</li> <li>b. Si selecciona el ícono Ubicación el sistema lista las provincias y municipios generados al iniciar formulario.</li> </ol> </li> <li>5. Usuario pulsa el botón aplicar filtros.</li> <li>6. El sistema realiza la consulta de los eventos deportivos que cumplan los filtros seleccionados por el usuario.</li> <li>7. El sistema redirige al usuario a la pantalla principal, listando los eventos deportivos resultantes de la búsqueda.</li> </ol>	
Flujo alternativo	<p>El usuario pulsa el botón atrás, por lo tanto, no se realiza la búsqueda y se redirige al usuario a la pantalla principal listando los eventos resultantes de la búsqueda por defecto realizada al iniciar sesión.</p> <p>En caso de que se realice la búsqueda sin resultados se muestra un mensaje informando al usuario.</p>	
Postcondición	El sistema genera un listado en la página Principal con los eventos deportivos resultantes de la búsqueda ordenada por fecha de celebración.	

### 8.1.3.8 Ver mis eventos

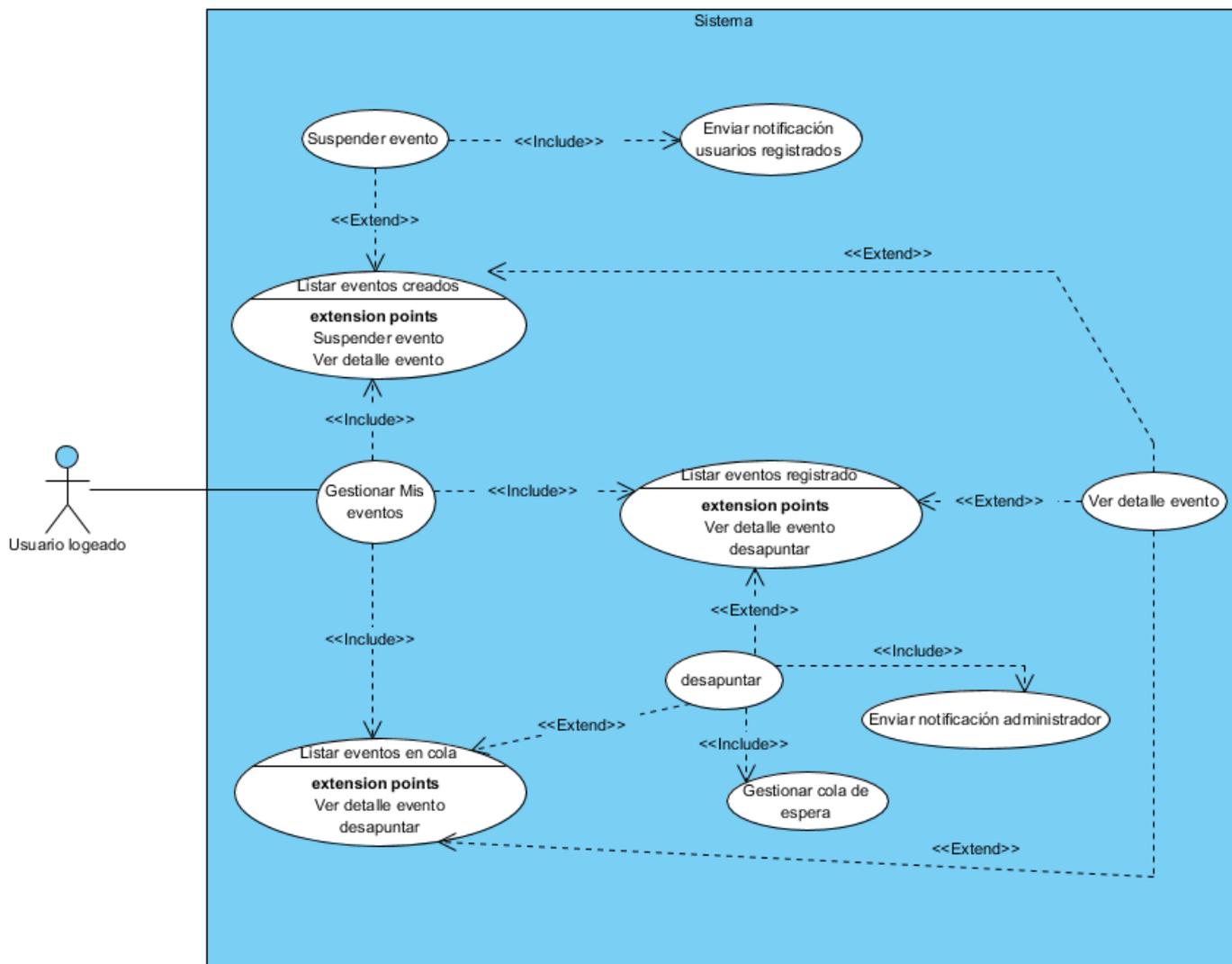


Figura 8.1.3.8.1: Diagrama caso de uso mis eventos

Caso de uso	Mis eventos
Descripción	Usuario puede visualizar sus eventos creados, eventos registrado y eventos en cola.
Actor	Usuario logeado.
Precondición	Usuario ha iniciado sesión.
Flujo principal	<ol style="list-style-type: none"> <li>1. El usuario abre el menú lateral.</li> <li>2. El usuario pulsa la opción Mis eventos.</li> <li>3. El sistema genera 3 pantallas:                             <ol style="list-style-type: none"> <li>a. Eventos creados por el usuario ordenados por fecha y estado del evento.</li> <li>b. Eventos en que el usuario está apuntado.</li> <li>c. Eventos en que el usuario está en cola de espera.</li> </ol> </li> <li>4. El sistema muestra en pantalla la pestaña con los eventos creados.</li> </ol>
Flujo alternativo	En caso de que en algún listado no haya eventos no muestra ningún elemento.
Postcondición	Se genera una pantalla con 3 pestañas.

<b>Caso de uso</b>		<b>Suspender evento</b>
Descripción	Usuario puede suspender un evento creado por el mismo.	
Actor	Usuario logeado.	
Precondición	Usuario ha iniciado sesión y es propietario del evento deportivo.	
Flujo principal	<ol style="list-style-type: none"> <li>1. El usuario abre el menú lateral.</li> <li>2. El usuario pulsa la opción Mis eventos.</li> <li>3. El sistema genera 3 pantallas: <ol style="list-style-type: none"> <li>a. Eventos creados por el usuario ordenados por fecha y estado del evento.</li> <li>b. Eventos en que el usuario está apuntado.</li> <li>c. Eventos en que el usuario está en cola de espera.</li> </ol> </li> <li>4. El usuario escoge la pestaña eventos creados.</li> <li>5. El usuario pulsa cancelar evento en uno de los eventos listados.</li> <li>6. El sistema verifica si realmente desea eliminar.</li> <li>7. Usuario acepta cancelar el evento.</li> <li>8. El sistema reordena los eventos creados de la pantalla.</li> <li>9. El sistema notifica a los usuarios apuntados en el evento de que el evento ha sido suspendido.</li> </ol>	
Flujo alternativo	<p>No hay eventos deportivos creados por el usuario.</p> <p>Hay eventos creados, pero están todos cancelados o finalizados, por lo tanto, no se habilita la opción cancelar evento.</p> <p>Usuario no acepta cancelar el evento y el sistema no realiza ninguna acción.</p>	
Postcondición	Usuario ha suspendido el evento y se ha notificado a todos los participantes.	

<b>Caso de uso</b>		<b>Desapuntar de un evento registrado</b>
Descripción	Usuario puede desapuntarse de un evento en el que está registrado como participante.	
Actor	Usuario logeado.	
Precondición	Usuario ha iniciado sesión y es participante del evento deportivo.	
Flujo principal	<ol style="list-style-type: none"> <li>1. El usuario abre el menú lateral.</li> <li>2. El usuario pulsa la opción Mis eventos.</li> <li>3. El sistema genera 3 pantallas: <ol style="list-style-type: none"> <li>a. Eventos creados por el usuario ordenados por fecha y estado del evento.</li> <li>b. Eventos en que el usuario está apuntado.</li> <li>c. Eventos en que el usuario está en cola de espera.</li> </ol> </li> <li>4. El usuario escoge la pestaña eventos registrado.</li> <li>5. El usuario pulsa Desapuntarme en uno de los eventos listados.</li> <li>6. El sistema verifica si realmente desea desapuntarse.</li> <li>7. Usuario acepta desapuntarse del evento.</li> <li>8. El sistema reordena los eventos registrado de la pantalla.</li> <li>9. El sistema notifica al propietario del evento de que el usuario se ha desapuntado.</li> <li>10. El sistema comprueba si hay usuarios en cola para participar en el evento: <ol style="list-style-type: none"> <li>a. Si hay usuarios en cola, el primero de la lista pasa automáticamente a ser participante del evento. El sistema notifica al usuario que ha sido apuntado al evento.</li> <li>b. Si no hay usuario en cola el sistema no hace nada.</li> </ol> </li> <li>11. El sistema refresca el listado de eventos registrados eliminando el evento en que se ha desapuntado el usuario.</li> </ol>	
Flujo alternativo	No hay eventos deportivos en el que el usuario participe.	
Postcondición	Usuario se ha desapuntado de un evento y se ha notificado al propietario.	

### 8.1.3.9 Ver detalle evento deportivo

En la aplicación, se permite entrar al detalle de un evento deportivo por diferentes vías:

- Listado de eventos generado en la pantalla principal al iniciar sesión o al realizar una búsqueda mediante los filtros del buscador.
- Accediendo al menú lateral a la opción Mis eventos se genera una pantalla con 3 pestañas como hemos visto en el apartado anterior.
- Accediendo al perfil de un usuario o de uno mismo, se muestra un listado de los eventos creados.

Por lo tanto, a la hora de definir el caso de uso de ver el detalle de un evento, supondremos que se ha accedido mediante una de las 3 vías comentadas. Se tendrá en cuenta si el usuario que accede al detalle de un evento es el propietario o no de éste.

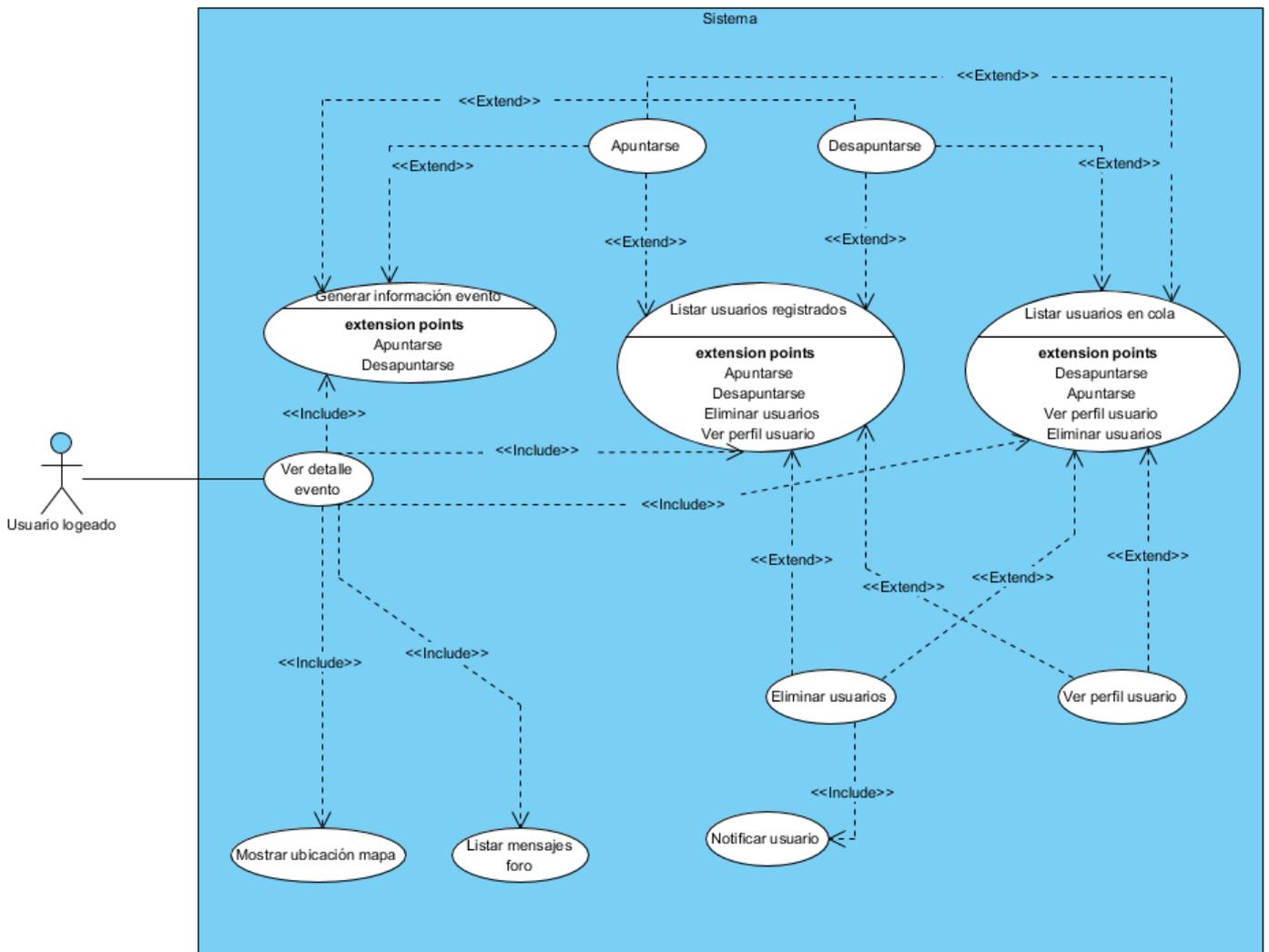


Figura 8.1.3.9.1: Diagrama caso de uso ver detalle evento.

<b>Caso de uso</b>		<b>Ver detalle evento deportivo</b>
Descripción	Usuario puede visualizar el detalle de un evento deportivo.	
Actor	Usuario logeado.	
Precondición	Usuario ha iniciado sesión.	
Flujo principal	<ol style="list-style-type: none"> <li>1. El usuario accede al detalle de un evento deportivo.</li> <li>2. El sistema genera 5 pantallas: <ol style="list-style-type: none"> <li>a. Pestaña con la imagen e información del evento deportivo. <ol style="list-style-type: none"> <li>i. Si el usuario actual no es el propietario del evento <ol style="list-style-type: none"> <li>1. Si está apuntado en el evento como participante o en cola se muestra la opción desapuntarse.</li> <li>2. Si no está registrado en el evento se muestra la opción apuntarse.</li> </ol> </li> <li>ii. Si el usuario actual es propietario no se muestran las opciones apuntarse/desapuntarse.</li> </ol> </li> <li>b. Listado de los usuarios que participan en el evento deportivo. <ol style="list-style-type: none"> <li>i. Si el usuario actual no es el propietario del evento <ol style="list-style-type: none"> <li>1. Si está apuntado en el evento como participante o en cola se muestra la opción desapuntarse.</li> <li>2. Si no está registrado en el evento se muestra la opción apuntarse.</li> </ol> </li> <li>ii. Si el usuario actual es propietario se muestra la opción de eliminar participantes del evento.</li> </ol> </li> <li>c. Listado de los usuarios en cola de espera para participar en el evento. <ol style="list-style-type: none"> <li>i. Si el usuario actual no es el propietario del evento <ol style="list-style-type: none"> <li>1. Si está apuntado en el evento como participante o en cola se muestra la opción desapuntarse.</li> <li>2. Si no está registrado en el evento se muestra la opción apuntarse.</li> </ol> </li> <li>ii. Si el usuario actual es propietario se muestra la opción de eliminar participantes en cola del evento.</li> </ol> </li> <li>d. Listado con los mensajes del foro del evento. <ol style="list-style-type: none"> <li>i. Si el foro es privado <ol style="list-style-type: none"> <li>1. Si el usuario es propietario o esta apuntado en el evento se muestra el contenido del foro y se puede participar.</li> <li>2. Si el usuario no es propietario y no está registrado en el evento no puede visualizar ni participar en el foro.</li> </ol> </li> <li>ii. Si el foro es público cualquier usuario de la aplicación puede visualizar y participar en el foro.</li> </ol> </li> <li>e. Mapa indicando la ubicación en donde se celebrará el evento.</li> </ol> </li> <li>3. El sistema muestra en pantalla la pestaña información del evento.</li> </ol>	
Flujo alternativo		
Postcondición	Se genera una pantalla con 5 pestañas detallando la información del evento.	

<b>Caso de uso</b>		<b>Apuntarse a un evento deportivo</b>
Descripción	Usuario puede apuntarse a un evento deportivo.	
Actor	Usuario logeado.	
Precondición	Usuario ha iniciado sesión y no es participante del evento deportivo.	
Flujo principal	<ol style="list-style-type: none"> <li>1. El usuario accede al detalle de un evento del cual no es propietario.</li> <li>2. El usuario pulsa el botón apuntarse. <ol style="list-style-type: none"> <li>a. Si el evento está completo se añade al usuario a la lista de espera en orden de entrada.</li> <li>b. Si el evento no está completo o tiene aforo ilimitado se registra al usuario como participante.</li> </ol> </li> <li>3. Sistema actualiza pestañas de usuarios registrados y usuarios en cola.</li> <li>4. El sistema envía una notificación al administrador informándole del nuevo participante.</li> <li>5. El sistema actualiza el listado de participantes del evento.</li> </ol>	
Flujo alternativo	Si el evento está completo el usuario se registra en la lista de espera y no como participante.	
Postcondición	El usuario es nuevo participante del evento deportivo.	

<b>Caso de uso</b>		<b>Eliminar participante evento deportivo</b>
Descripción	Usuario puede eliminar participantes de un evento deportivo del cual es propietario.	
Actor	Usuario logeado.	
Precondición	Usuario ha iniciado sesión y es propietario del evento deportivo.	
Flujo principal	<ol style="list-style-type: none"> <li>1. El usuario accede al detalle de un evento del cual es propietario.</li> <li>2. El usuario accede a la lista de participantes o la lista de usuarios en cola.</li> <li>3. El usuario pulsa el botón con el ícono de una papelera.</li> <li>4. El sistema verifica si realmente desea eliminar al usuario:</li> <li>5. El usuario confirma la eliminación.</li> <li>6. Si se elimina un participante registrado. <ol style="list-style-type: none"> <li>a. Si el evento estaba completo y hay participantes en cola, el primero de la lista pasa a ser participante registrado. El sistema envía una notificación al usuario registrado.</li> </ol> </li> <li>7. El sistema envía una notificación al usuario eliminado.</li> <li>8. El sistema actualiza los listados de usuarios registrados y usuarios en cola.</li> </ol>	
Flujo alternativo	No hay usuarios registrados en el evento.	
Postcondición	Se ha eliminado a un usuario del evento deportivo y enviado una notificación informándole.	

### 8.1.3.10 Modificar evento

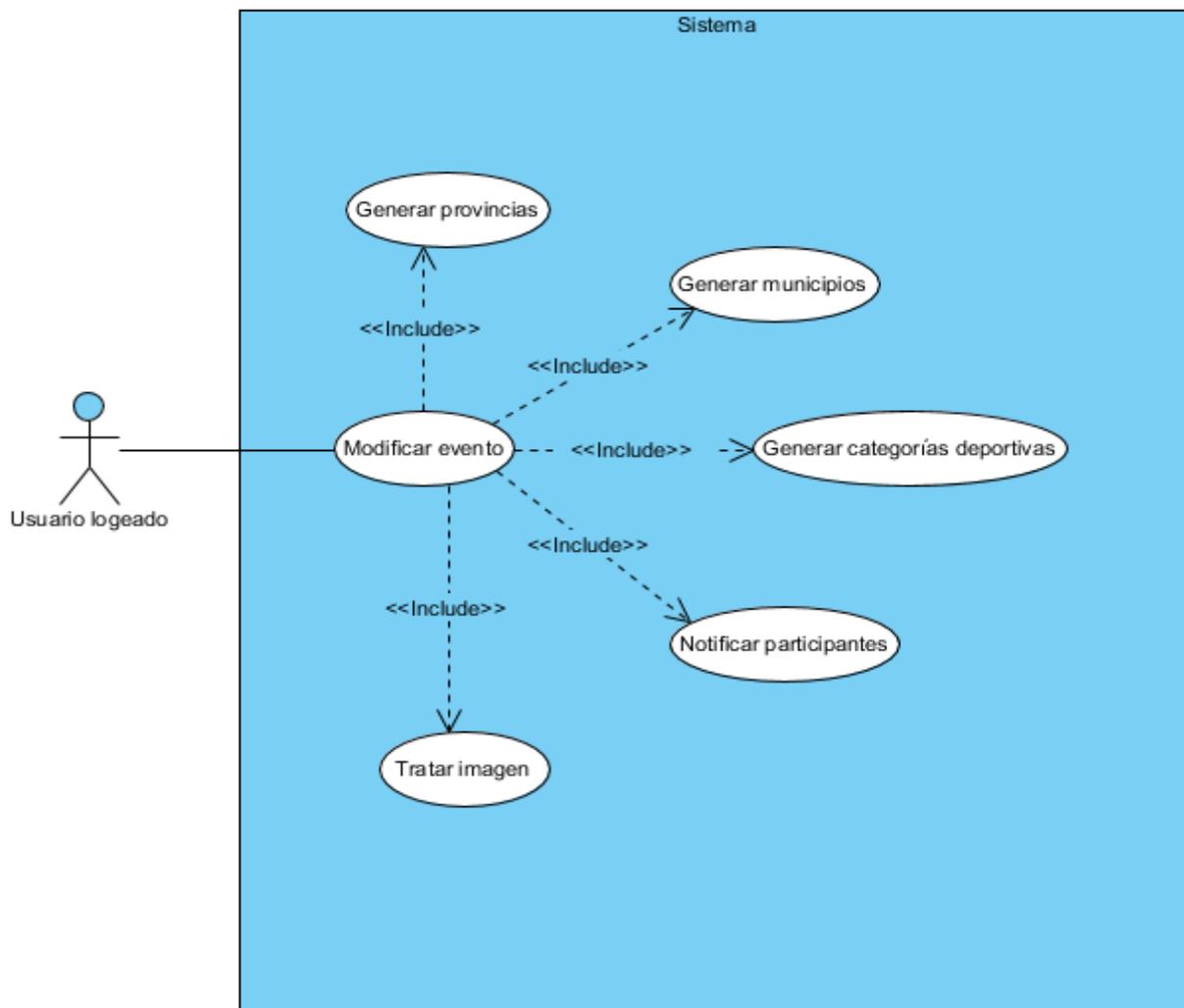


Figura 8.1.3.10.1: Diagrama caso de uso modificar evento.

Caso de uso		Modificar evento deportivo
Descripción	Usuario puede modificar los datos de un evento del cual es propietario.	
Actor	Usuario logeado.	
Precondición	Usuario ha accedido al detalle de un evento deportivo del cual es propietario.	
Flujo principal	<ol style="list-style-type: none"> <li>1. El usuario accede al detalle de un evento del cual es propietario.</li> <li>2. El sistema habilita la opción de modificar el evento.</li> <li>3. El usuario pulsa sobre botón de modificar evento.</li> <li>4. El sistema carga un formulario editable con la información del evento. Se permite modificar la imagen del evento, el título del evento, la descripción, la categoría deportiva, el número de participantes, la duración, la privacidad del foro, día y hora y la ubicación.</li> <li>5. El usuario pulsa guardar.</li> <li>6. El sistema verifica si realmente desea modificar el evento.</li> <li>7. El usuario confirma los cambios.</li> <li>8. El sistema redirige al usuario a la página anterior y envía una notificación a los participantes del evento informándoles del cambio.</li> </ol>	
Flujo alternativo	Algún campo del formulario está vacío y se muestra un mensaje de error.	
Postcondición	Se han modificado los datos de un evento deportivo y notificado a los participantes.	

### 8.1.3.11 Ver perfil usuario

Se puede acceder mediante 2 vías al perfil propio de un usuario o al perfil de otro usuario. Mediante el menú lateral en la opción mi perfil accedemos al perfil del usuario autenticado. Por otro lado, al acceder al detalle de un evento en las pestañas en donde se listan los usuarios registrados y los usuarios en cola, también podemos acceder al perfil de los usuarios.

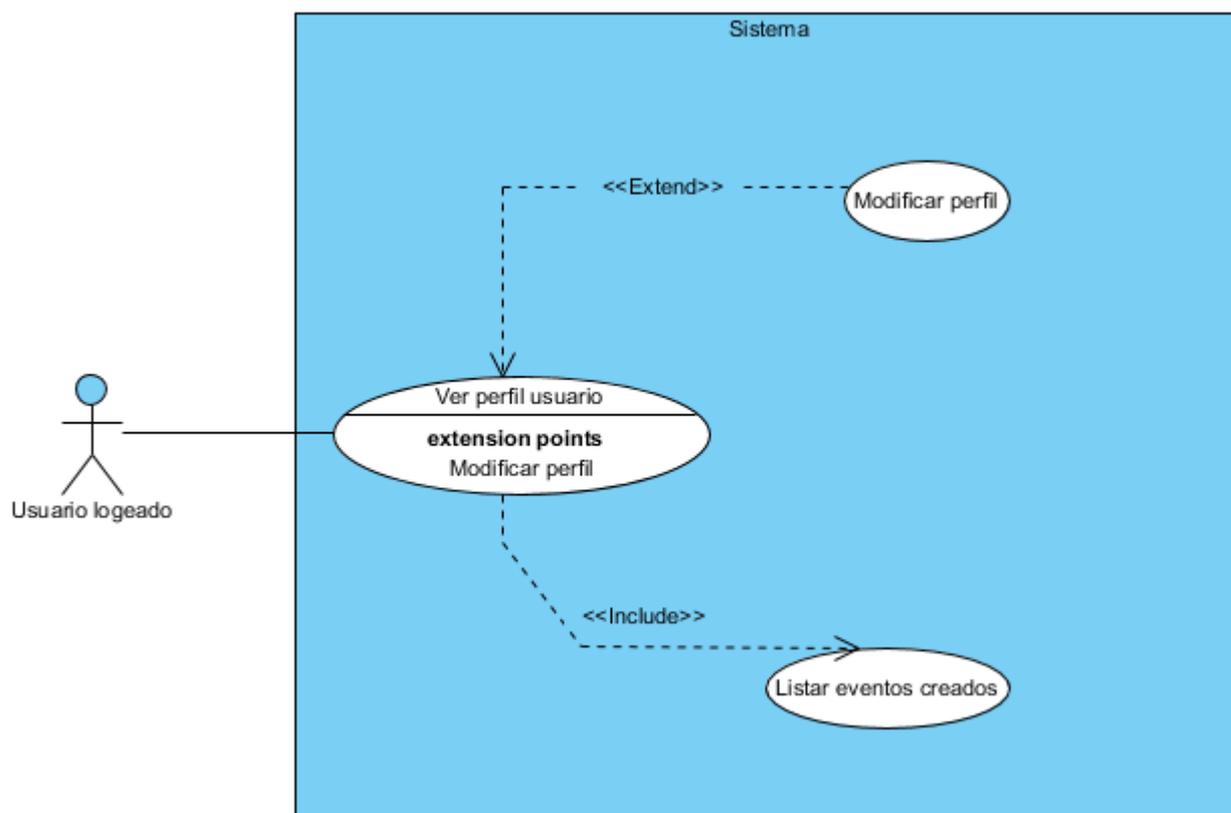


Figura 8.1.3.11.1: Diagrama caso de uso ver perfil usuario.

Caso de uso	Ver perfil usuario
Descripción	Usuario puede visualizar el perfil.
Actor	Usuario logeado.
Precondición	Usuario ha accedido al detalle de un usuario.
Flujo principal	<ol style="list-style-type: none"> <li>1. El usuario accede al perfil de un usuario.</li> <li>2. El sistema muestra una pantalla en donde se muestra la imagen de perfil, los datos del usuario y los eventos creados por el usuario.</li> <li>3. Si el perfil es el del mismo usuario, se habilita una opción para modificar el perfil.</li> <li>4. Si el perfil es el del mismo usuario, se habilita la opción de suspender los eventos creados.</li> </ol>
Flujo alternativo	
Postcondición	Se muestra el perfil del usuario por pantalla.

### 8.1.3.12 Modificar perfil

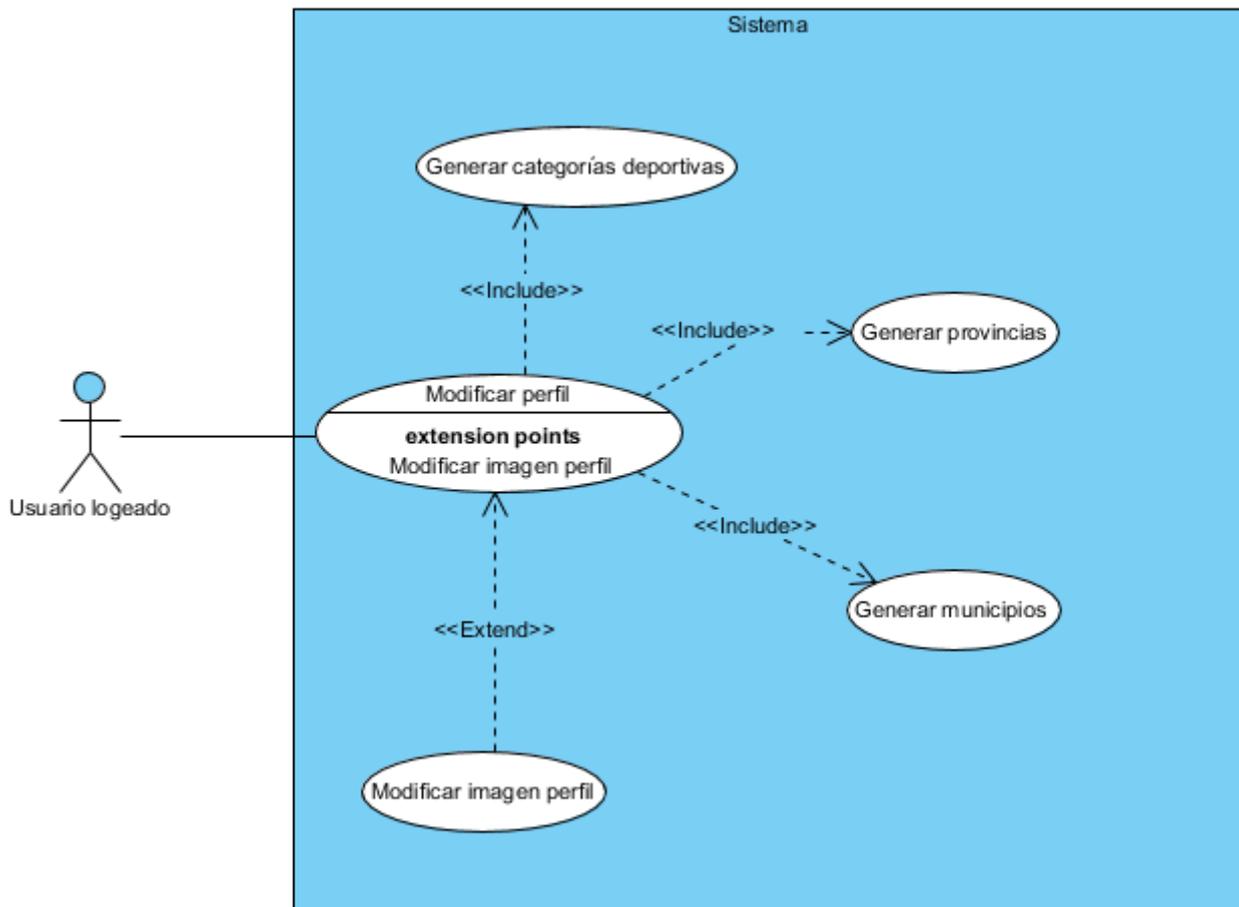


Figura 8.1.3.13.1: Diagrama caso de uso modificar perfil

Caso de uso	Modificar evento deportivo
Descripción	Usuario puede modificar los datos del perfil.
Actor	Usuario logeado.
Precondición	Usuario ha accedido al detalle de su perfil y ha seleccionado modificar perfil,
Flujo principal	<ol style="list-style-type: none"> <li>1. El usuario accede al detalle de su perfil.</li> <li>2. El usuario toca la opción modificar perfil.</li> <li>3. El sistema carga un formulario editable con la información del perfil. Se permite modificar la imagen del perfil, el nombre, los apellidos, el Nick, las categorías deportivas favoritas, la provincia y el municipio de residencia.</li> <li>4. El usuario pulsa guardar.</li> <li>5. El sistema verifica si realmente desea modificar los datos del perfil.</li> <li>6. El usuario confirma los cambios.</li> <li>7. El sistema redirige al usuario a la página anterior.</li> </ol>
Flujo alternativo	Algún campo del formulario está vacío y se muestra un mensaje de error.
Postcondición	Se han modificado los datos del perfil del usuario.

### 8.1.3.13 Administrar notificaciones

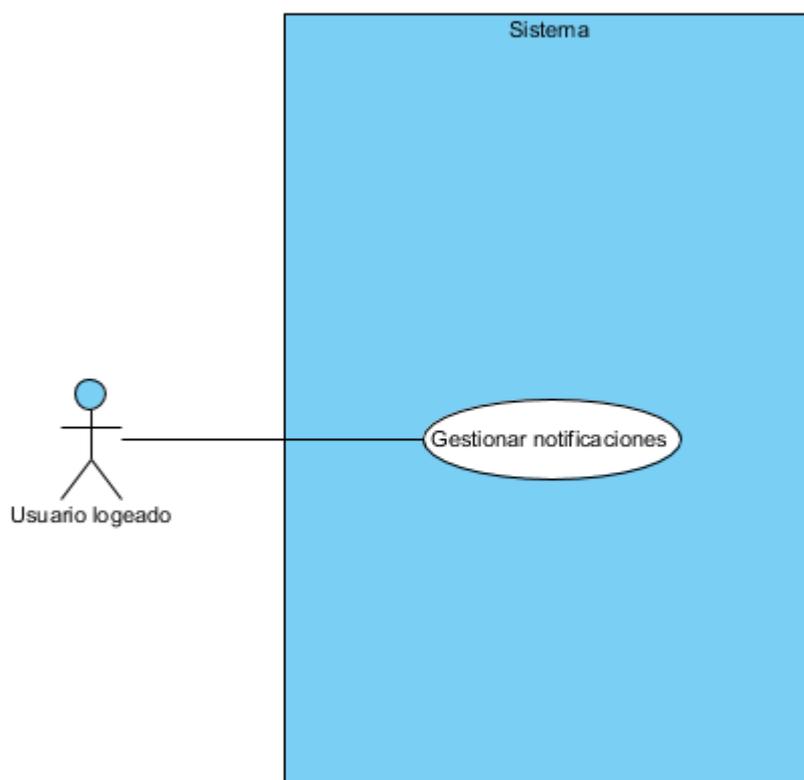


Figura 8.1.3.13.1: Diagrama caso de uso administrar notificaciones

<b>Caso de uso Administrar notificaciones</b>	
Descripción	Usuario puede activar/desactivar las notificaciones.
Actor	Usuario logeado.
Precondición	Usuario ha iniciado sesión.
Flujo principal	<ol style="list-style-type: none"><li>1. El usuario abre el menú lateral.</li><li>2. El usuario selecciona la notificación.</li><li>3. El sistema lista las notificaciones disponibles en la aplicación mostrando si se encuentran activadas o desactivadas.</li><li>4. El usuario puede activar/desactivar cada tipo de notificación.</li><li>5. El sistema guarda automáticamente al modificar el estado de una notificación.</li></ol>
Flujo alternativo	Ninguno
Postcondición	El usuario gestiona si desea activar o desactivar los diferentes tipos de notificaciones y el sistema lo registra.

## 8.2 Diseño del sistema

En este apartado se muestra el diseño utilizado para el desarrollo del sistema.

### 8.2.1 Modelo entidad-relación

Mediante el modelo entidad-relación se modela la representación de las entidades utilizadas en el sistema de información, así como sus interrelaciones y propiedades. En la figura 8.2.1.1 se muestra el modelo entidad-relación diseñado en una etapa inicial.

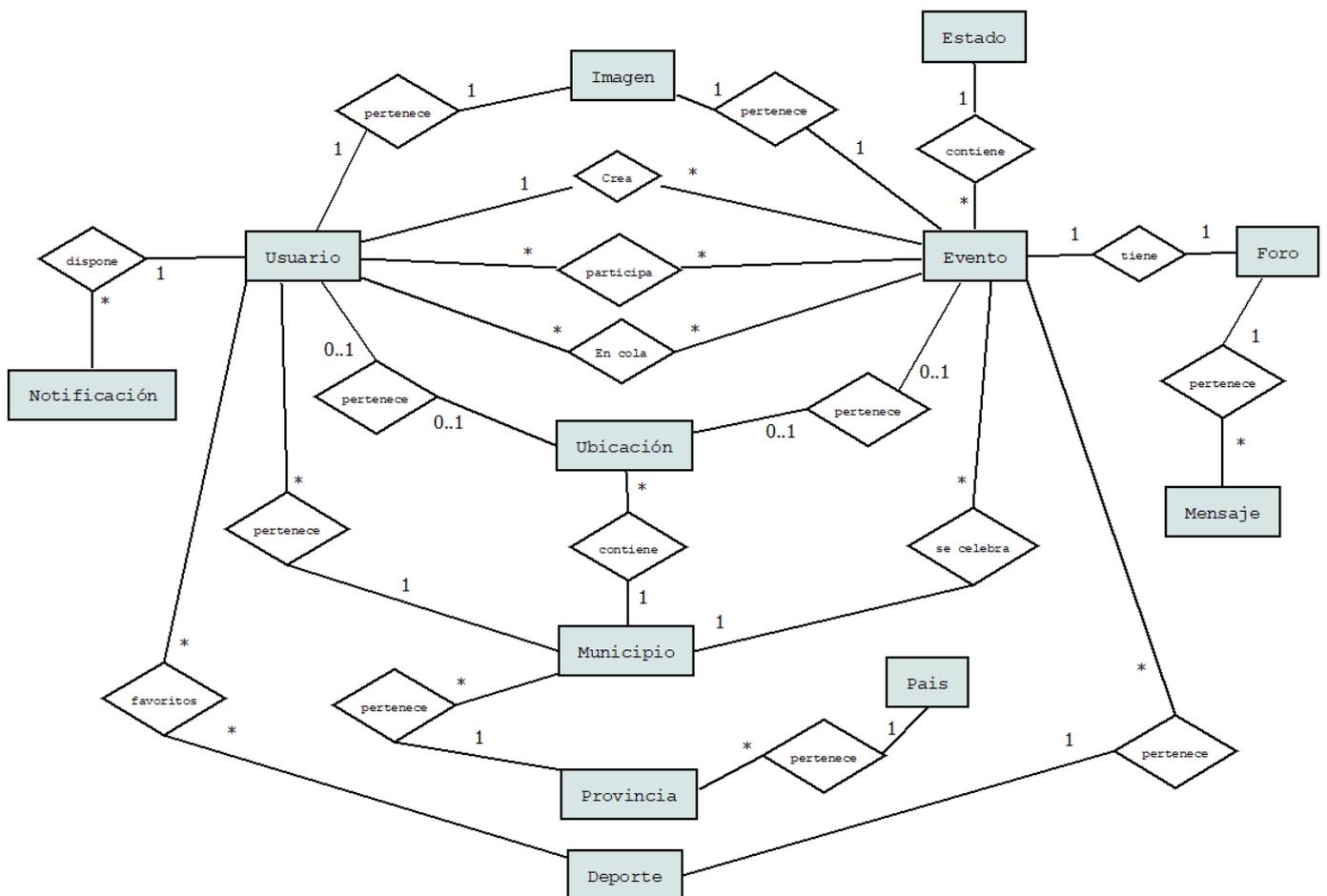


Figura 6.2.1.1: Modelo entidad-relación diseño inicial

En un inicio, se realizó un estudio para determinar las entidades y relaciones que se necesitarían para el desarrollo de la aplicación. Respecto al diseño final (figura 8.2.1.2), la principal diferencia la observamos en las entidades Foro y Mensaje que estaban destinadas a gestionar el contenido de los foros de los eventos deportivos. Finalmente, como se ha explicado en el *apartado 7.2.6* esta funcionalidad ha sido externalizada en la base de datos de Firebase.



- usuarioEliminado: indica se desea recibir una notificación cuando el administrador del evento elimina al usuario del evento en el que está registrado.
- **Usuario:** Guarda la información de los diferentes usuarios registrados en la aplicación. Contiene la relación a la imagen de perfil, a la configuración de las notificaciones, a la ubicación GPS registrada, a los eventos creados, eventos registrado, eventos en cola, municipio, país y provincia de residencia y a las categorías deportivas favoritas.
- **Evento:** Guarda la información de los diferentes eventos deportivos creados por los usuarios registrados en la aplicación. Contiene la relación a los usuarios registrados en el evento, los usuarios en cola, el estado del evento, imagen del evento, municipio o ubicación GPS, a la categoría deportiva y al administrador del evento.
- **Estado:** Contiene los valores de estado en que un evento puede estar:
  - Abierto: el evento no se ha celebrado y hay espacio disponible para que los usuarios se apunten.
  - Completo: el evento no se ha celebrado pero el aforo está completo.
  - Finalizado: el evento ya se ha celebrado.
  - Suspendido: el administrador del evento ha decidido cancelar el evento deportivo.
- **Deporte:** Contiene el listado de las diferentes categorías deportivas disponibles en la aplicación.
- **Usuario\_deporte:** Contiene los datos que nos indican las categorías deportivas favoritas de cada usuario de la aplicación.
- **Eventos\_registrado:** Contiene los datos que nos indica los eventos en que participa cada usuario de la aplicación.
- **Eventos\_en\_cola:** Contiene los datos que nos indica los eventos en que un usuario está en lista de espera.
- **Imagen:** Contiene la ruta al sistema de ficheros de las imágenes de los usuarios y eventos deportivos creados en la aplicación.
- **Ubicación:** Guarda la información de la ubicación GPS registrada de los dispositivos móviles de los usuarios y de los eventos si se ha seleccionada a través del mapa de Google.
- **País:** Contiene los países disponibles en la aplicación.
- **Provincias:** Contiene las provincias disponibles en la aplicación.



## 8.2.3 Diseño interfaz usuario

Para realizar el diseño de la interfaz gráfica de las diferentes pantallas de la aplicación móvil se ha utilizado NinjaMocks [25], que se trata de una aplicación web que nos permite crear maquetas gráficas. Nos proporciona la funcionalidad de organizar y diseñar las maquetas utilizando una potente herramienta de tipo *drag&drop*, mediante la cual podemos arrastrar y mover los elementos dentro de la pantalla de la maqueta. Fue escogida esta aplicación ya que dispone de los componentes propios de Android simulando de una forma casi idéntica el futuro diseño a realizar.

La interfaz se ha diseñado con el propósito de ser intuitiva y simple, pero sin olvidar que sea potente y cubra todos los requisitos necesarios para cumplir con los objetivos. A continuación, en las siguientes figuras podemos visualizar las maquetas realizadas en la fase de análisis y diseño de la aplicación.

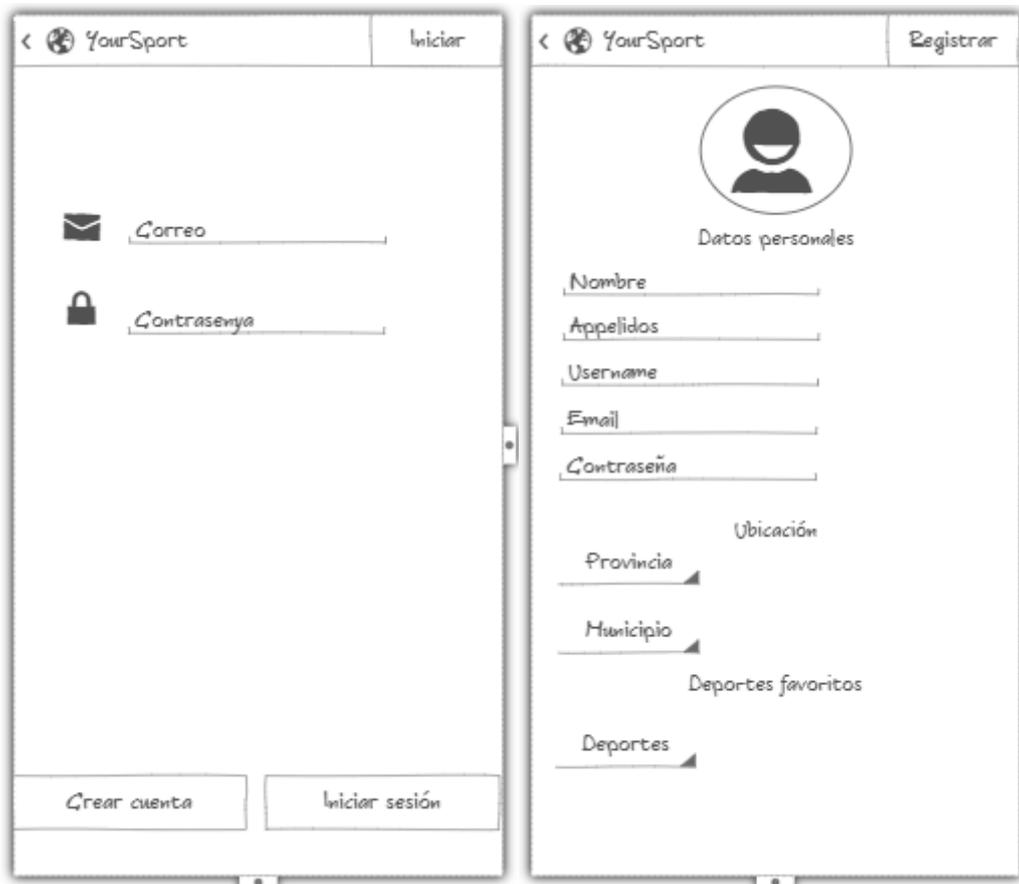


Figura 8.2.3.1.1: Pantallas de inicio de sesión y crear una cuenta

En la figura 8.2.3.1.1 se muestran las maquetas de las pantallas de inicio de sesión y el formulario para darse de alta en la aplicación. La primera pantalla corresponde a la pantalla principal de la aplicación, es decir, la pantalla que aparecerá siempre al iniciar la aplicación. En caso de no estar dado de alta en la aplicación y, por lo tanto, no poder autenticarse, se dispone de un formulario para registrarse como nuevo usuario en la misma.

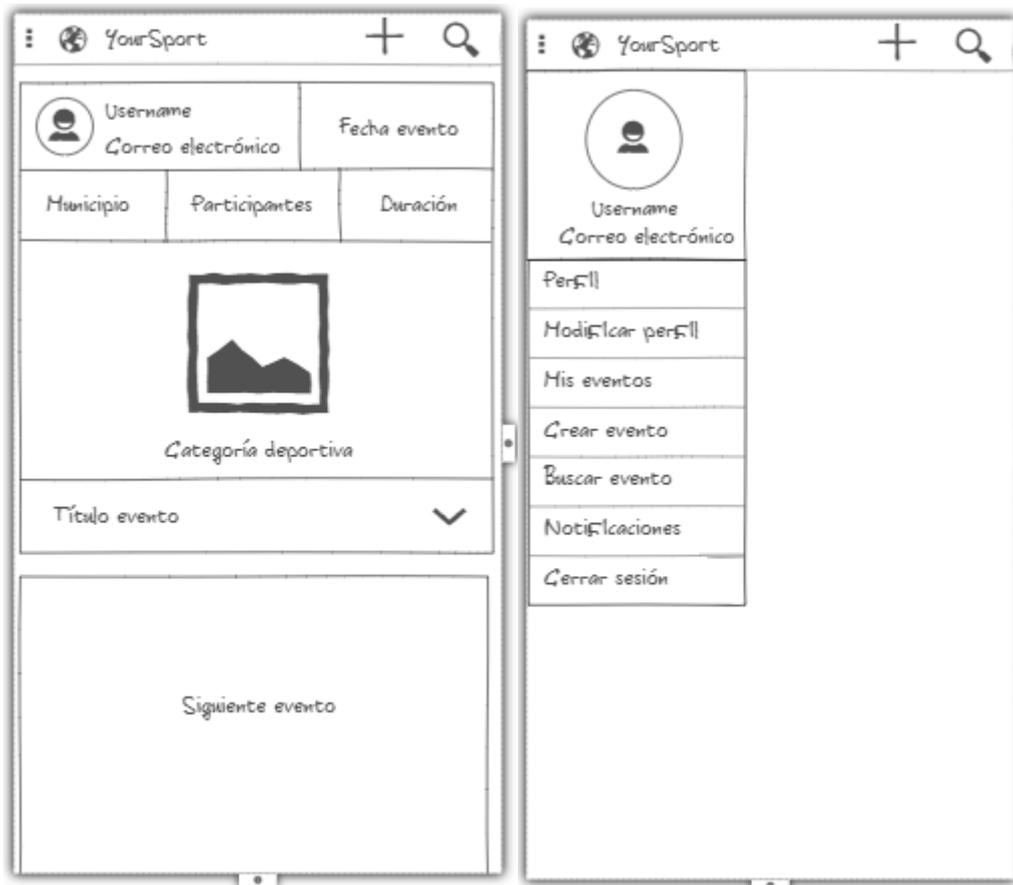


Figura 8.2.3.1.2: Pantallas principal y menú lateral

En la figura 8.2.3.1.2 se muestra la maqueta de la pantalla principal que se puede visualizar inmediatamente que un usuario inicia sesión o se registra en el sistema. Se considera la pantalla principal porque permite navegar a las distintas funcionalidades disponibles en la aplicación a través del menú lateral o de la barra superior.

Como cuerpo de esta pantalla se genera un listado de eventos que coincide con la búsqueda por defecto de cada usuario según sus preferencias o al resultado de realizar una búsqueda personalizada.

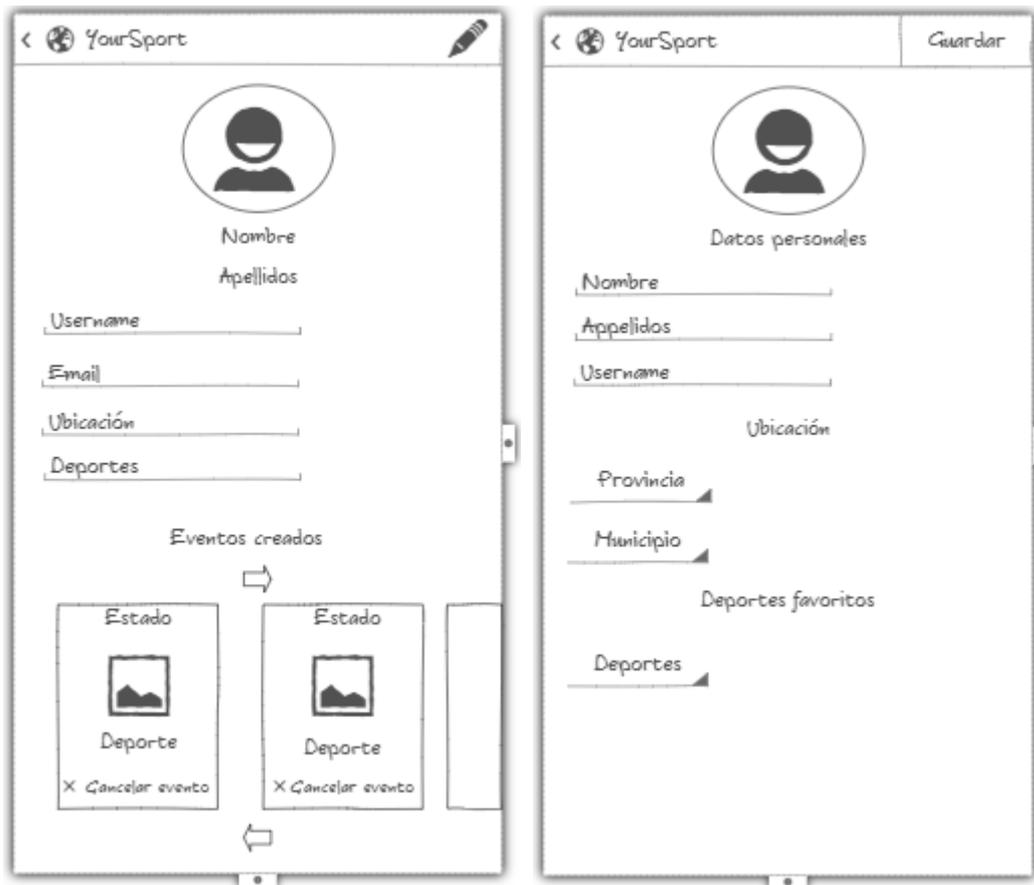


Figura 8.2.3.1.3: Pantallas perfil usuario y modificar perfil

En la figura 8.2.3.1.3 se muestran las pantallas referentes a la información de perfil de un usuario y al formulario para modificar dicha información. Dentro de la pantalla de la información de perfil de cada usuario además de su información básica se muestra un listado con orientación horizontal de todos sus eventos creados.

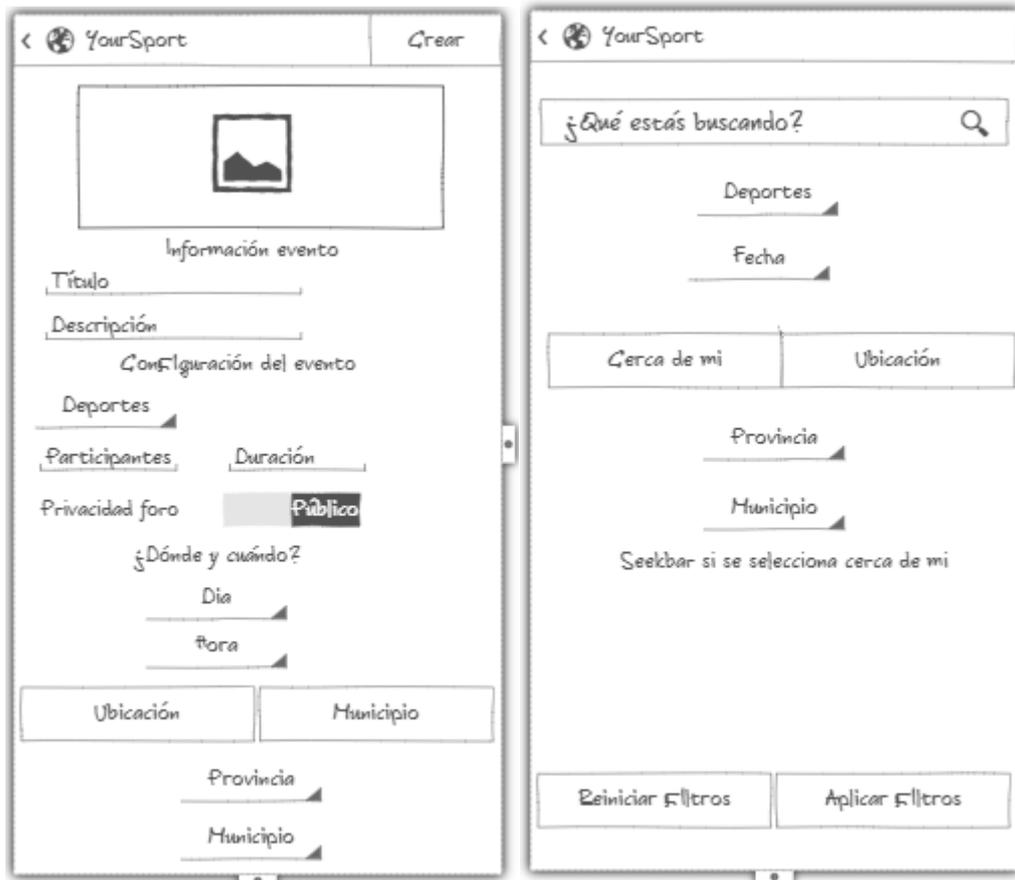


Figura 8.2.3.1.4: Pantallas crear evento deportivo y buscador de eventos

En la figura 8.2.3.1.4 se muestran las pantallas referentes a la creación de un nuevo evento deportivo y al formulario disponible para realizar búsquedas personalizadas.

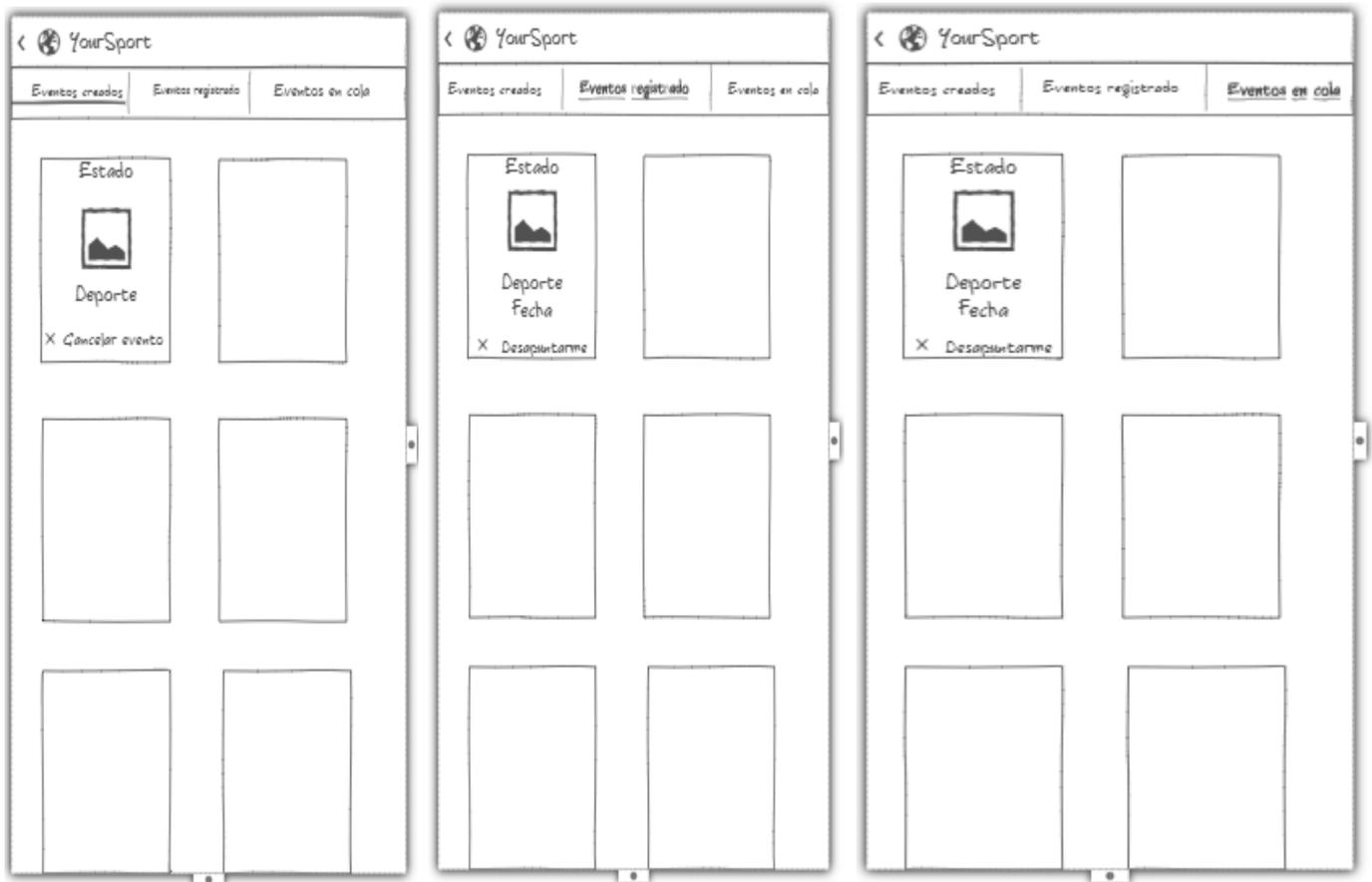


Figura 8.2.3.1.5: Pantallas de la opción mis eventos

En la figura 8.2.3.1.5 se muestran las pantallas disponibles al seleccionar la opción mis eventos en el menú lateral de la pantalla principal. Como se puede observar, se listan en diferentes pestañas los eventos creados por el usuario, los eventos en que el usuario participa y los eventos en que el usuario está en lista de espera.

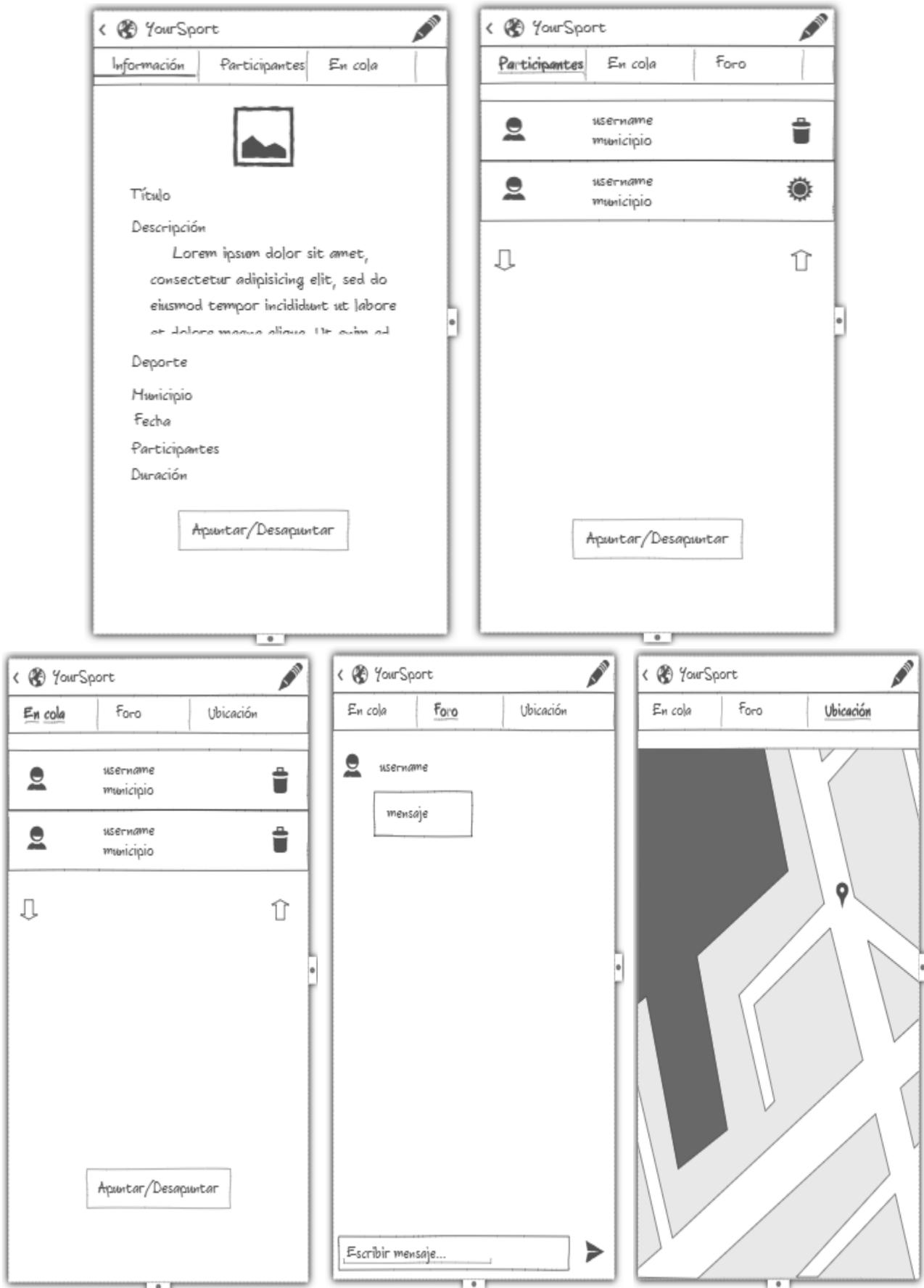


Figura 8.2.3.1.6: Pantallas detalle de un evento

En la figura 8.2.3.1.6 se muestran las pantallas disponibles al acceder al detalle de un evento deportivo. Como se puede ver se muestra en diferentes pestañas la información básica del evento, la lista de usuarios registrados, la lista de usuarios en lista de espera, el contenido del foro del evento y la ubicación del evento representada en un mapa.

The screenshot shows a mobile application interface for 'YourSport'. At the top, there is a back arrow, the app name 'YourSport', and a 'Guardar' button. Below this is a placeholder image for the event. The main content is organized into sections: 'Información evento' with input fields for 'Título' and 'Descripción'; 'Configuración del evento' with a 'Deportes' dropdown menu, 'Participantes' and 'Duración' input fields, and a 'Privacidad foro' toggle switch currently set to 'Público'; and '¿Dónde y cuándo?' with 'Dia' and 'hora' dropdown menus. At the bottom, there are input fields for 'Ubicación' and 'Municipio', and a 'Provincia' dropdown menu.

Figura 8.2.3.1.7: Pantallas detalle de un evento

En la figura 8.2.3.1.7 se muestra la pantalla que contiene el formulario para modificar los datos de un evento ya creado.



Figura 8.2.3.1.8: Pantallas detalle de un evento

En la figura 8.2.3.1.8 se muestra la pantalla que contiene el formulario para activar/desactivar las diferentes notificaciones disponibles en la aplicación.

Finalmente, en la figura 8.2.3.1.9 se muestra un diagrama de pantallas de manera que se pueda interpretar la navegación entre pantallas existente en la aplicación.

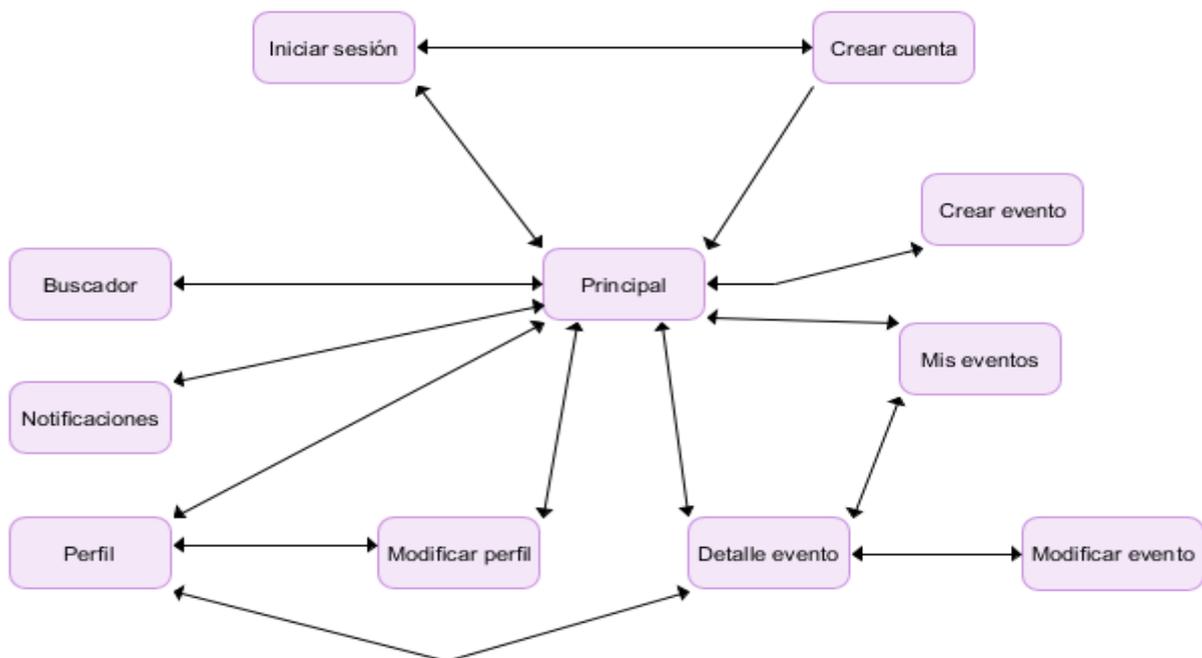


Figura 8.2.3.1.9: Diagrama de pantallas

## 8.3 Diseño API REST

Mediante la API REST el servidor de aplicaciones proporciona toda la información necesaria a los clientes y les permite realizar acciones. Como el servicio Web está basado en el estándar REST nos permite enviar peticiones HTTP REST (GET, POST, DELETE, PUT) a una dirección en concreto. El formato de datos utilizado para realizar las peticiones y respuestas es JSON.

Se ha diseñado la API REST en diferentes recursos:

- Recurso usuario
- Recurso ubicación
- Recurso país
- Recurso provincia
- Recurso municipio
- Recurso deporte
- Recurso evento
- Recurso participante
- Recurso notificación
- Recurso imagen

A continuación, se definen las diferentes operaciones a realizar sobre los recursos disponibles.

### 8.3.1 Recurso usuario

URL	/rest/usuario/login
<b>Descripción</b>	Inicia sesión de un usuario en la aplicación si los datos enviados corresponden a un usuario registrado en el sistema y no tiene la sesión iniciada.
<b>Método</b>	POST
<b>Formato datos de entrada</b>	JSON
<b>Formato datos de salida</b>	JSON
<b>Parámetros URL</b>	-
<b>Parámetros cuerpo</b>	{ "email" : [String], "password" : [String], "tokenFireBase" : [String] }
<b>Respuesta correcta</b>	{ "id": [Long], "username": [String], "email": [String] }
<b>Respuesta incorrecta</b>	{ "message": "Error al obtener la sesión" } { "message": "Ya ha iniciado sesión" } { "message": "Email o contraseña incorrectos" }

URL	/rest/usuario
<b>Descripción</b>	Registra un nuevo usuario en la aplicación. El Nick y la cuenta de correo electrónico no existían previamente dadas de alta para otro usuario.
<b>Método</b>	POST
<b>Formato datos de entrada</b>	JSON
<b>Formato datos de salida</b>	JSON
<b>Parámetros URL</b>	-
<b>Parámetros cuerpo</b>	<pre>{   "email" : [String],   "username" : [String],   "password" : [String],   "tokenFireBase" : [String],   "nombre" : [String],   "apellidos" : [String],   "municipio" : [Long],   "deportesFavoritos" :   [     [Long]   ] }</pre>
<b>Respuesta correcta</b>	<pre>{   "id": [Long],   "username": [String],   "email": [String] }</pre>
<b>Respuesta incorrecta</b>	<pre>{   "message": "Error al obtener la sesión" }</pre> <pre>{   "message": "No se puede registrar un usuario logeado" }</pre> <pre>{   "message": "Ya existe un usuario con el email" + email }</pre> <pre>{   "message": "Ya existe un usuario con el Nick" + Nick }</pre>

URL	/rest/usuario/logout/{idUsuario}
<b>Descripción</b>	Cierra la sesión del usuario que realiza la petición.
<b>Método</b>	DELETE
<b>Formato datos de entrada</b>	-
<b>Formato datos de salida</b>	JSON
<b>Parámetros URL</b>	:idUsuario [Long] (Obligatorio)
<b>Parámetros cuerpo</b>	-
<b>Respuesta correcta</b>	{ "id": [Long] }
<b>Respuesta incorrecta</b>	{ "message": "Error al obtener la sesión" }
	{ "message": "No se puede cerrar la sesión de otros usuarios" }
	{ "message": "No se puede cerrar sesión si no está autenticado" }

URL	/rest/usuario/ {idUsuario}
<b>Descripción</b>	Obtiene la información del perfil del usuario indicado en el parámetro de la URL al realizar la petición.
<b>Método</b>	GET
<b>Formato datos de entrada</b>	-
<b>Formato datos de salida</b>	JSON
<b>Parámetros URL</b>	:idUsuario [Long] (Obligatorio)
<b>Parámetros cuerpo</b>	-
<b>Respuesta correcta</b>	<pre>{   "id": [Long],   "username": [String],   "nombre": [String],   "apellidos": [String],   "email": [String],   "fechaRegistro": [Date],   "municipio": {     "id": [Long],     "municipio": [String],     "latitudEstimada": [Double],     "longitudEstimada": [Double]   },   "provincia": {     "id": [Long],     "provincia": [String]   },   "pais": {     "id": [Long],     "pais": [String]   },   "deportesFavoritos": [     {       "id": [Long],       "deporte": [String]     }   ] }</pre>
<b>Respuesta incorrecta</b>	<pre>{   "message": "Error al obtener la sesión" } {   "message": "No ha iniciado sesión" } {   "message": "El usuario no existe" }</pre>

URL	/rest/usuario/{ idUsuario}
<b>Descripción</b>	Modifica los datos del usuario actual con sesión iniciada.
<b>Método</b>	PUT
<b>Formato datos de entrada</b>	JSON
<b>Formato datos de salida</b>	JSON
<b>Parámetros URL</b>	:idUsuario [Long] (Obligatorio)
<b>Parámetros cuerpo</b>	<pre>{   "username" : [String],   "nombre" : [String],   "apellidos" : [String],   "municipio" : [Long],   "deportesFavoritos" :     [       [Long]     ] }</pre>
<b>Respuesta correcta</b>	<pre>{   "id": [Long] }</pre>
<b>Respuesta incorrecta</b>	<pre>{   "message": "Error al obtener la sesión" }</pre>
	<pre>{   "message": "No se puede modificar datos de otros usuarios" }</pre>
	<pre>{   "message": "No ha iniciado sesión" }</pre>
	<pre>{   "message": "El Nick ya está registrado. Por favor escoja otro" }</pre>
	<pre>{   "message": "El usuario no existe" }</pre>

URL	/rest/usuario/ evento/{idUsuario}/{tipo}
<b>Descripción</b>	Obtiene los eventos de un usuario en función del tipo indicado por parámetro en la URL: Tipo (0) devuelve todos los eventos creados por el usuario. Tipo (1) devuelve todos los eventos en que participa el usuario. Tipo (2) devuelve todos los eventos en que el usuario está en cola.
<b>Método</b>	GET
<b>Formato datos de entrada</b>	-
<b>Formato datos de salida</b>	JSON
<b>Parámetros URL</b>	:idUsuario [Long] (Obligatorio) :tipo [Long] (Obligatorio)
<b>Parámetros cuerpo</b>	-
<b>Respuesta correcta</b>	[ <pre> {   "id": [Long],   "titulo": [String],   "descripcion": [String],   "duracion": [Long],   "numeroParticipantes": [Long],   "fechaEvento": [Date],   "deporte": {     "id": [Long],     "deporte": [String],   },   "estado": {     "id": [Long],     "estado": [String],   },   "participantesRegistrados": [Long],   "administrador": {     "id": [Long],     "username": [String],     "email": [String],   },   "municipio": {     "id": [Long],     "municipio": [String],     "latitudEstimada": [Double],     "longitudEstimada": [Double]   },   "provincia": {     "id": [Long],     "provincia": [String],   },   "foro": {     "id": [Long],     "esPublico": [Boolean],     "titulo": [String],   },   "ubicacion": { </pre>

	<pre>"latitud": [Double], "longitud": [Double], "direccion": [String], "municipio": [String], } } ]</pre>
<b>Respuesta incorrecta</b>	<pre>{   "message": "Error al obtener la sesión" } {   "message": "No ha iniciado sesión" } {   "message": "No existe el usuario" } {   "message": "El tipo no es válido. Introduzca (0) para recuperar sus eventos creados, (1) para recuperar los eventos en que está inscrito o (2) para los eventos en que está en cola." }</pre>

### 8.3.2 Recurso ubicación

URL	/rest/ubicacion/usuario
<b>Descripción</b>	Guarda en base de datos la ubicación GPS del dispositivo móvil que ha iniciado sesión.
<b>Método</b>	POST
<b>Formato datos de entrada</b>	JSON
<b>Formato datos de salida</b>	JSON
<b>Parámetros URL</b>	-
<b>Parámetros cuerpo</b>	<pre>{   "latitud" : [Double],   "longitud" : [Double],   "direccion" : [String],   "municipio" : [String] }</pre>
<b>Respuesta correcta</b>	<pre>{   "idUbicacion": [Long] }</pre>
	<pre>{   "message": "No ha iniciado sesión" } {   "message": "No existe el municipio indicado" } {   "message": "El usuario no está registrado" }</pre>

URL	/rest/ubicacion/{idEvento}
<b>Descripción</b>	Se obtiene la ubicación GPS de un evento deportivo.
<b>Método</b>	GET
<b>Formato datos de entrada</b>	-
<b>Formato datos de salida</b>	JSON
<b>Parámetros URL</b>	:idEvento [Long] (Obligatorio)
<b>Parámetros cuerpo</b>	-
<b>Respuesta correcta</b>	{ "latitud": [Double], "longitud": [Double], "direccion": [String], "municipio": [String], }
<b>Respuesta incorrecta</b>	{ "message": "Error al obtener la sesión" } { "message": "No ha iniciado sesión" } { "message": "No hay ubicación para el evento" } { "message": "No existe el evento" }

### 8.3.3 Recurso país

URL	/rest/pais
<b>Descripción</b>	Se obtienen los países disponibles en la aplicación.
<b>Método</b>	GET
<b>Formato datos de entrada</b>	-
<b>Formato datos de salida</b>	JSON
<b>Parámetros URL</b>	-
<b>Parámetros cuerpo</b>	-
<b>Respuesta correcta</b>	[ { "id": [Long], "pais": [String] } ]
<b>Respuesta incorrecta</b>	{ "message": "No hay países disponibles" }

### 8.3.4 Recurso provincia

URL	/rest/provincia/{idPais}
<b>Descripción</b>	Se obtienen las provincias disponibles del país indicado.
<b>Método</b>	GET
<b>Formato datos de entrada</b>	-
<b>Formato datos de salida</b>	JSON
<b>Parámetros URL</b>	:idPais [Long] (Obligatorio)
<b>Parámetros cuerpo</b>	-
<b>Respuesta correcta</b>	[ <pre>           {             "id": [Long],             "provincia": [String]           }           ]</pre>
<b>Respuesta incorrecta</b>	{ <pre>           "message": "No hay provincias disponibles"           }</pre>

### 8.3.5 Recurso municipio

URL	/rest/municipio/{idProvincia}
<b>Descripción</b>	Se obtienen los municipios disponibles de la provincia indicada.
<b>Método</b>	GET
<b>Formato datos de entrada</b>	-
<b>Formato datos de salida</b>	JSON
<b>Parámetros URL</b>	:idProvincia [Long] (Obligatorio)
<b>Parámetros cuerpo</b>	-
<b>Respuesta correcta</b>	[ <pre>           {             "id": [Long],             "provincia": [String],             "latitudEstimada": [Double],             "longitudEstimada": [Double]           }           ]</pre>
<b>Respuesta incorrecta</b>	{ <pre>           "message": "No hay municipios disponibles"           }</pre>

### 8.3.6 Recurso deporte

URL	/rest/deporte
<b>Descripción</b>	Se obtienen las categorías deportivas disponibles en la aplicación.
<b>Método</b>	GET
<b>Formato datos de entrada</b>	-
<b>Formato datos de salida</b>	JSON
<b>Parámetros URL</b>	-
<b>Parámetros cuerpo</b>	-
<b>Respuesta correcta</b>	[ <pre>           {             "id": [Long],             "deporte": [String]           }           ]           </pre>
<b>Respuesta incorrecta</b>	{ <pre>           "message": "No hay deportes disponibles"           </pre>

### 8.3.7 Recurso evento

URL	/rest/evento
<b>Descripción</b>	Permite crear un nuevo evento deportivo vinculado al usuario que realiza la petición.
<b>Método</b>	POST
<b>Formato datos de entrada</b>	JSON
<b>Formato datos de salida</b>	JSON
<b>Parámetros URL</b>	-
<b>Parámetros cuerpo</b>	<pre>           {             "titulo" : [String],             "descripcion" : [String],             "duracion" : [int],             "numeroParticipantes" : [int],             "esPublico" : [Boolean],             "deporte" : [Long],             "fechaEvento" : [String],             "ubicacionGPS": {               "latitud" : [Double],               "longitud" : [Double],               "direccion" : [String],               "municipio" : [String],             },             "tituloForo" : [String]           }           </pre> <p>* La ubicación del evento puede ser mediante ubicación GPS como en el</p>

	ejemplo anterior, o indicando el identificador de un municipio disponible en la aplicación. En este caso se sustituiría el campo “ubicacionGPS” por “municipio”: [Long].
<b>Respuesta correcta</b>	{ "id": [Long] }
<b>Respuesta incorrecta</b>	{ "message": “No ha iniciado sesión” }
	{ "message": “Error en la creación del evento” }
	{ "message": “No existe el municipio” }

URL	/rest/evento/{idUserario}/{idEvento}
<b>Descripción</b>	Permite crear un nuevo evento deportivo vinculado al usuario que realiza la petición.
<b>Método</b>	PUT
<b>Formato datos de entrada</b>	JSON
<b>Formato datos de salida</b>	JSON
<b>Parámetros URL</b>	:idUserario [Long] (Obligatorio) :idEvento [Long] (Obligatorio)
<b>Parámetros cuerpo</b>	<pre>{   "titulo" : [String],   "descripcion" : [String],   "duracion" : [int],   "numeroParticipantes" : [int],   "esPublico" : [Boolean],   "deporte" : [Long],   "fechaEvento" : [String],   "ubicacionGPS": {     "latitud" : [Double],     "longitud" : [Double],     "direccion" : [String],     "municipio" : [String],   },   "tituloForo" : [String] }</pre> <p>* La ubicación del evento puede ser mediante ubicación GPS como en el ejemplo anterior, o indicando el identificador de un municipio disponible en la aplicación. En este caso se sustituiría el campo “ubicacionGPS” por “municipio” : [Long].</p>
<b>Respuesta correcta</b>	{ "id": [Long] }

	}
<b>Respuesta incorrecta</b>	{ "message": "No ha iniciado sesión" }
	{ "message": "No puede modificar eventos de otros usuarios" }
	{ "message": "El evento no existe" }

URL	/rest/evento/{idEvento}
<b>Descripción</b>	Permite suspender un evento deportivo.
<b>Método</b>	DELETE
<b>Formato datos de entrada</b>	-
<b>Formato datos de salida</b>	JSON
<b>Parámetros URL</b>	:idEvento [Long] (Obligatorio)
<b>Parámetros cuerpo</b>	-
<b>Respuesta correcta</b>	{ "id": [Long] }
<b>Respuesta incorrecta</b>	{ "message": "No ha iniciado sesión" }
	{ "message": "El evento ya ha finalizado" }
	{ "message": "El evento ya ha sido suspendido" }
	{ "message": "No se puede cancelar el evento" }
	{ "message": "El evento no existe" }

URL	/rest/evento/{idEvento}
<b>Descripción</b>	Permite obtener la información del evento deportivo solicitado en la URL.
<b>Método</b>	GET
<b>Formato datos de entrada</b>	-
<b>Formato datos de salida</b>	JSON
<b>Parámetros URL</b>	:idEvento [Long] (Obligatorio)
<b>Parámetros cuerpo</b>	-
<b>Respuesta correcta</b>	<pre> {   "id": [Long],   "titulo": [String],   "descripcion": [String],   "duracion": [Long],   "numeroParticipantes": [Long],   "fechaEvento": [Date],   "deporte": {     "id": [Long],     "deporte": [String],   },   "estado": {     "id": [Long],     "estado": [String],   },   "participantesRegistrados": [Long],   "administrador": {     "id": [Long],     "username": [String],     "email": [String],   },   "municipio": {     "id": [Long],     "municipio": [String],     "latitudEstimada": [Double],     "longitudEstimada": [Double]   },   "provincia": {     "id": [Long],     "provincia": [String],   },   "foro": {     "id": [Long],     "esPublico": [Boolean],     "titulo": [String],   },   "ubicacion": {     "latitud": [Double],     "longitud": [Double],     "direccion": [String],     "municipio": [String], </pre>

	<pre>         }       }     }   } } </pre>
<b>Respuesta incorrecta</b>	<pre> {   "message": "No ha iniciado sesión" } </pre> <pre> {   "message": "El evento solicitado no existe" } </pre>

URL	/rest/evento
<b>Descripción</b>	Permite obtener eventos deportivos en función de los valores pasados por parámetro.
<b>Método</b>	GET
<b>Formato datos de entrada</b>	-
<b>Formato datos de salida</b>	JSON
<b>Parámetros URL</b>	<p>:offset [int] (Opcional) Posición del primer evento a devolver de la lista resultante a la petición realizada. Valor por defecto 10.</p> <p>:limite [int] (Opcional) Cantidad de resultados a devolver a partir del offset indicado. Valor por defecto 0.</p> <p>:titulo [String] (Opcional) Texto sobre el cual buscar coincidencias en los títulos de los evento creados. Valor por defecto "".</p> <p>:deportes Lista[Long] (Opcional) Identificadores de las categorías deportivas.</p> <p>:fechaEvento [Date] (Opcional) Fecha de celebración del evento.</p> <p>:distancia [Integer] (Opcional) Distancia en kilómetros desde la ubicación del evento hasta ubicación GPS registrada del usuario actual.</p> <p>:municipio [Long] (Opcional) Identificador del municipio en donde buscar los eventos deportivos.</p>
<b>Parámetros cuerpo</b>	-
<b>Respuesta correcta</b>	<pre> [   {     "id": [Long],     "titulo": [String],     "descripcion": [String],     "duracion": [Long],     "numeroParticipantes": [Long],     "fechaEvento": [Date],     "deporte": {       "id": [Long],       "deporte": [String],     },     "estado": {       "id": [Long], </pre>

	<pre> "estado": [String], }, "participantesRegistrados": [Long], "administrador": {   "id": [Long],   "username": [String],   "email": [String], }, "municipio": {   "id": [Long],   "municipio": [String],   "latitudEstimada": [Double],   "longitudEstimada": [Double] }, "provincia": {   "id": [Long],   "provincia": [String], }, "foro": {   "id": [Long],   "esPublico": [Boolean],   "titulo": [String], }, "ubicacion": {   "latitud": [Double],   "longitud": [Double],   "direccion": [String],   "municipio": [String], } } ] </pre>
<b>Respuesta incorrecta</b>	<pre> {   "message": "No ha iniciado sesión" } {   "message": "Error en la búsqueda de eventos" } </pre>

### 8.3.8 Recurso participante

URL	/rest/participante/{idEvento}
<b>Descripción</b>	Se registra un usuario como participante en un evento deportivo. Si el evento está completo, se le añade a la lista de espera.
<b>Método</b>	POST
<b>Formato datos de entrada</b>	-
<b>Formato datos de salida</b>	JSON
<b>Parámetros URL</b>	:idEvento [Long] (Obligatorio)
<b>Parámetros cuerpo</b>	-
<b>Respuesta correcta</b>	{ "id": [Long] }
<b>Respuesta incorrecta</b>	{ "message": "No ha iniciado sesión" }
	{ "message": "El evento ya ha finalizado" }
	{ "message": "El evento está suspendido" }
	{ "message": "Ya está en la cola del evento" }
	{ "message": "Ya está registrado en el evento" }
	{ "message": "El evento no existe" }

URL	/rest/ {idEvento}/{idUsuario}
<b>Descripción</b>	Se desapunta al usuario del evento, indicados por parámetro, ya sea como participante o de la lista de espera.
<b>Método</b>	DELETE
<b>Formato datos de entrada</b>	-
<b>Formato datos de salida</b>	JSON
<b>Parámetros URL</b>	:idEvento [Long] (Obligatorio) :idUsuario [Long] (Obligatorio)
<b>Parámetros cuerpo</b>	-
<b>Respuesta correcta</b>	{ "idEvento": [Long] }
<b>Respuesta incorrecta</b>	{ "message": "No ha iniciado sesión" } { "message": "No participas en el evento" } { "message": "No puede desregistrar otros usuarios de un evento" } { "message": "El usuario no existe" } { "message": "El evento no existe" }

URL	/rest/ {idEvento}/{tipo}
<b>Descripción</b>	Obtiene los participantes de un evento indicado por parámetro según el tipo especificado. Tipo (0) se obtienen los participantes del evento. Tipo (1) se obtienen los usuarios en lista de espera.
<b>Método</b>	GET
<b>Formato datos de entrada</b>	-
<b>Formato datos de salida</b>	JSON
<b>Parámetros URL</b>	:idEvento [Long] (Obligatorio) :tipo [Long] (Obligatorio)
<b>Parámetros cuerpo</b>	-
<b>Respuesta correcta</b>	[ { "id": [Long], "username": [String], "municipio": [String] }, ]
<b>Respuesta incorrecta</b>	{ "message": "No ha iniciado sesión" }

	} { "message": "El evento no existe" }
--	---

### 8.3.9 Recurso notificación

URL	/rest/notificacion
<b>Descripción</b>	Obtiene la configuración de las notificaciones del usuario actual.
<b>Método</b>	GET
<b>Formato datos de entrada</b>	-
<b>Formato datos de salida</b>	JSON
<b>Parámetros URL</b>	-
<b>Parámetros cuerpo</b>	-
<b>Respuesta correcta</b>	{ "altaUsuario": [Boolean], "bajaUsuario": [Boolean], "eventoCancelado": [Boolean], "datosModificados": [Boolean], "usuarioEliminado": [Boolean] }
<b>Respuesta incorrecta</b>	{ "message": "No ha iniciado sesión" }  { "message": "El usuario no está registrado" }

URL	/rest/notificacion
<b>Descripción</b>	Permite modificar la configuración de la notificaciones del usuario actual.
<b>Método</b>	PUT
<b>Formato datos de entrada</b>	-
<b>Formato datos de salida</b>	JSON
<b>Parámetros URL</b>	-
<b>Parámetros cuerpo</b>	{ "altaUsuario": [Boolean], "bajaUsuario": [Boolean], "eventoCancelado": [Boolean], "datosModificados": [Boolean], "usuarioEliminado": [Boolean] }
<b>Respuesta correcta</b>	{ "altaUsuario": [Boolean], "bajaUsuario": [Boolean], "eventoCancelado": [Boolean], "datosModificados": [Boolean],

	"usuarioEliminado": [Boolean] }
<b>Respuesta incorrecta</b>	{ "message": "No ha iniciado sesión" }
	{ "message": "El usuario no está registrado" }

### 8.3.10 Recurso imagen

URL	/rest/imagen/evento/{idEvento}
<b>Descripción</b>	Permite subir una imagen asociada al evento indicado por parámetro.
<b>Método</b>	POST
<b>Formato datos de entrada</b>	MULTIPART_FORM_DATA
<b>Formato datos de salida</b>	JSON
<b>Parámetros URL</b>	:idEvento [Long] (Obligatorio)
<b>Parámetros cuerpo</b>	-
<b>Respuesta correcta</b>	{ "id": [Long], "ruta": [String] }
<b>Respuesta incorrecta</b>	{ "message": "No ha iniciado sesión" }
	{ "message": "Error al guardar la imagen" }

URL	/rest/imagen/usuario
<b>Descripción</b>	Permite subir una imagen de perfil del usuario actual.
<b>Método</b>	POST
<b>Formato datos de entrada</b>	MULTIPART_FORM_DATA
<b>Formato datos de salida</b>	JSON
<b>Parámetros URL</b>	-
<b>Parámetros cuerpo</b>	-
<b>Respuesta correcta</b>	{ "id": [Long], "ruta": [String] }
<b>Respuesta incorrecta</b>	{ "message": "No ha iniciado sesión" }
	{ "message": "No se ha podido subir la imagen" }

URL	/rest/imagen/usuario/nombre/{idUserario}
<b>Descripción</b>	Se obtiene el nombre con el que se encuentra guardada la imagen de perfil del usuario indicado por parámetro.
<b>Método</b>	GET
<b>Formato datos de entrada</b>	-
<b>Formato datos de salida</b>	JSON
<b>Parámetros URL</b>	-
<b>Parámetros cuerpo</b>	-
<b>Respuesta correcta</b>	{ "nombreImagen": [String] }
<b>Respuesta incorrecta</b>	{ "message": "No ha iniciado sesión" }
	{ "message": "El usuario no existe" }

URL	/rest/imagen/evento/nombre/{idEvento}
<b>Descripción</b>	Se obtiene el nombre con el que se encuentra guardada la imagen de perfil del usuario indicado por parámetro.
<b>Método</b>	GET
<b>Formato datos de entrada</b>	-
<b>Formato datos de salida</b>	JSON
<b>Parámetros URL</b>	-
<b>Parámetros cuerpo</b>	-
<b>Respuesta correcta</b>	{ "nombreImagen": [String] }
<b>Respuesta incorrecta</b>	{ "message": "No ha iniciado sesión" }
	{ "message": "El evento no existe" }

URL	/rest/imagen/usuario/{idUserario}/{nombreImagen}
<b>Descripción</b>	Se obtiene la imagen de perfil del usuario indicado por parámetro.
<b>Método</b>	GET
<b>Formato datos de entrada</b>	-
<b>Formato datos de salida</b>	Image/png – image/jpg
<b>Parámetros URL</b>	:idUserario [Long] (Obligatorio) :nombreImagen [String] (Obligatorio)
<b>Parámetros cuerpo</b>	-
<b>Respuesta correcta</b>	-
<b>Respuesta incorrecta</b>	{ "message": "No ha iniciado sesión" } { "message": "El usuario no existe" }

URL	/rest/imagen/evento/{idEvento}/{nombreImagen}
<b>Descripción</b>	Se obtiene la imagen del evento indicado por parámetro.
<b>Método</b>	GET
<b>Formato datos de entrada</b>	-
<b>Formato datos de salida</b>	Image/png – image/jpg
<b>Parámetros URL</b>	:idEvento [Long] (Obligatorio) :nombreImagen [String] (Obligatorio)
<b>Parámetros cuerpo</b>	-
<b>Respuesta correcta</b>	-
<b>Respuesta incorrecta</b>	{ "message": "No ha iniciado sesión" } { "message": "El evento no existe" }

---

## 9. Implementación y pruebas

---

En este apartado se mostrará la implementación llevada a cabo, tanto en la aplicación servidor como en la aplicación cliente Android. Se detallará la implementación de los algoritmos y procesos más destacados del proyecto. Finalmente, se definirá la estrategia utilizada a la hora de testear las diferentes partes desarrolladas. El código fuente de ambas aplicaciones se encuentra almacenado en sus respectivos repositorios en la red. La aplicación servidor en la dirección:

[https://github.com/DanielBermudezRodriguez/TFG\\_APP\\_JAVAAEE.git](https://github.com/DanielBermudezRodriguez/TFG_APP_JAVAAEE.git)

Y la aplicación Android en la dirección:

[https://github.com/DanielBermudezRodriguez/TFG\\_APP\\_ANDROID.git](https://github.com/DanielBermudezRodriguez/TFG_APP_ANDROID.git)

### 9.1 Aplicación servidor

Este apartado se centra exclusivamente en detallar la implementación llevada a cabo para el desarrollo de la aplicación servidor JavaEE.

#### 9.1.1 Estructura del proyecto

En primer lugar, para entender mejor la implementación y funcionamiento de la aplicación servidor se explica de forma general la arquitectura de la plataforma desarrollada.

Para la gestión y construcción del proyecto se ha utilizado Maven [31]. Éste se trata de una herramienta software, *open-source*, destinada a la gestión y construcción de proyectos en Java creada en 2001. Maven define un modelo de configuración de construcción de proyectos de forma simple, basado en formato XML. Maven utiliza un *Project Object Model* (POM) para describir el proyecto de software a construir, sus dependencias y componentes externos, y el orden de construcción de los elementos. El archivo central de configuración de Maven es *pom.xml*, en el cual se indican los módulos que componen el proyecto o que librerías utiliza el software. El fichero contiene todo lo necesario para que, a la hora de generar el ejecutable, éste contenga todo lo necesario para su ejecución. A través del repositorio central de Maven (*Maven central*) podemos descargar las librerías sin necesidad de descargarlas a mano.

En el siguiente recuadro se muestra la dependencia utilizada en el archivo *pom.xml* para importar las dependencias que nos proporciona el servidor de aplicaciones utilizado.

```
<version.jboss.bom>9.0.2.Final</version.jboss.bom>
<dependency>
  <groupId>org.wildfly.bom</groupId>
  <artifactId>jboss-javaee-7.0-wildfly-with-tools</artifactId>
  <version>${version.jboss.bom}</version>
  <type>pom</type>
  <scope>import</scope>
</dependency>
```

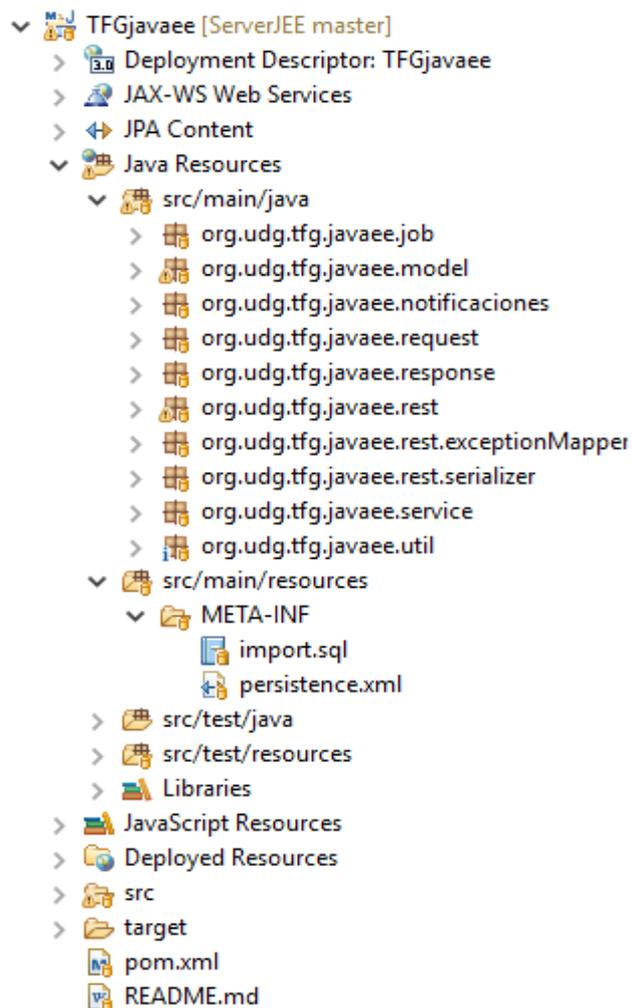


Figura 9.1.1.1: Estructura de la aplicación servidor en el IDE Eclipse

En el directorio padre se encuentra el nombre del proyecto Maven “TFGjavaee”. Se ha distribuido la implementación de la aplicación en 10 paquetes. El motivo de utilizar paquetes (*packages*) es agrupar las distintas partes del programa en clases que tienen una funcionalidad y elementos comunes. A continuación, se realiza una definición de la funcionalidad de cada paquete utilizado:

- **Org.udg.tfg.javaee.job:** Contiene las clases encargadas de ejecutar tareas asíncronas para el mantenimiento de la aplicación. Las clases utilizadas son las siguientes:
  - **ActualizarEstadoEvento.java:** Clase encargada de obtener los eventos deportivos existentes en la aplicación los cuales no se hayan celebrado y su estado no sea suspendido. Por cada evento obtenido compara con la fecha actual para determinar si el evento se ha celebrado y actualizar su estado como finalizado.



Figura 9.1.1.2: Clase ActualizarEventoEstado.java

El proceso se ejecuta de manera asíncrona cada minuto.

```

@Schedule(second = "*/60", minute = "*", hour = "*", persistent = false)
public void doWork() {
    List<Evento> eventos = eventoService.obtenerEventosNoFinalizados();
    log.log(Level.INFO, "Eventos recuperados: " + eventos.size());
    for (Evento evento : eventos) {
        eventoService.finalizarEvento(evento);
        log.log(Level.INFO, "El evento: " + evento.getTitulo() + " ha sido
        finalizado.");
    }
}
  
```

Los eventos no finalizados se obtienen a partir del Bean “eventoService” encargado de ejecutar la lógica de negocio de todas las acciones a realizar sobre los eventos deportivos. El contenido del método obtenerEventosNoFinalizados() se muestra a continuación.

```

public List<Evento> obtenerEventosNoFinalizados() {
    List<Evento> eventos = new ArrayList<>();
    Query q = em.createNamedQuery("@HQL_GET_EVENTOS_NO_FINALIZADOS");
    q.setParameter("fechaActual", new Date(), TemporalType.DATE);
    q.setParameter("idCancelado", Global.EVENTO_FINALIZADO);
    try {
        eventos = (List<Evento>) q.getResultList();
        return eventos;
    } catch (Exception e) {
        return eventos;
    }
}
  
```

Destacar la utilización de la funcionalidad *named queries* la cual nos permite agrupar las consultas HQL (Hibernate Query Language) en una clase. En este caso en la clase Evento.java, mapeada a la base de datos a través de Hibernate contiene la especificación de la consulta `@HQL_GET_EVENTOS_NO_FINALIZADOS`. Mediante la utilización de las consultas nombradas podemos utilizar una misma consulta en diferentes partes del código invocando solamente a su nombre que la identifica.

```

@Entity
@NamedQueries({
    @NamedQuery(name = "@HQL_GET_EVENTOS_NO_FINALIZADOS", query = "select e
    from Evento as e where e.fechaEvento <= :fechaActual and e.estado.id <>
    :idCancelado") })
public class Evento implements Serializable {
}
  
```

- **Org.udg.tfg.javaee.model:** Contiene las clases a ser mapeadas a la base de datos mediante el framework Hibernate. En el *apartado 8.2.2* se puede observar el diagrama de clases con las diferentes clases utilizadas para el desarrollo de la aplicación y sus relaciones.
- **Org.udg.tfg.javaee.service:** Contiene las clases encargadas de ejecutar la lógica de negocio a consecuencia de las peticiones recibidas mediante la API REST. Actúan como Beans, permitiendo a otras clases acceder a sus métodos, que interactúan con las clases del paquete *Org.udg.tfg.javaee.model* para obtener o persistir datos en la aplicación. La comunicación con dicho paquete se realiza mediante el objeto *EntityManager*. Se define una clase servicio en función del tipo de datos que queremos obtener o registrar. Las clases implementadas son las siguientes:
  - **DeporteService.java:** Bean local encargado de ejecutar la lógica de negocio de las peticiones realizadas sobre la entidad deporte definida en la clase *Deporte.java*.

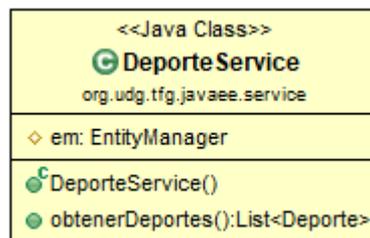


Figura 9.1.1.3: Clase DeporteService.java

DeporteService.java	
<b>Atributos</b>	<b>EntityManager:</b> Instancia encargada de gestionar las entidades de la aplicación. Mediante su utilización se realizan todas las transacciones disponibles contra la base de datos.
<b>Métodos</b>	<b>obtenerDeportes ():</b> Devuelve una lista con todas las categorías deportivas disponibles en la aplicación.

- **PaisService.java:** Bean local encargado de ejecutar la lógica de negocio de las peticiones realizadas sobre la entidad país definida en la clase *Pais.java*.

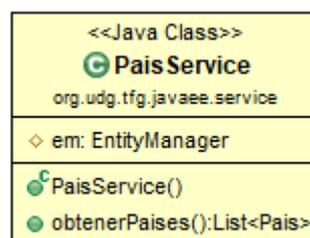


Figura 9.1.1.4: Clase PaisService.java

PaisService.java	
<b>Atributos</b>	<b>EntityManager:</b> Instancia encargada de gestionar las entidades de la aplicación. Mediante su utilización se realizan todas las transacciones disponibles contra la base de datos.
<b>Métodos</b>	<b>obtenerPaises ():</b> Devuelve una lista con todos los países disponibles en la aplicación.

- **ProvinciaService.java:** Bean local encargado de ejecutar la lógica de negocio de las peticiones realizadas sobre la entidad provincia definida en la clase Provincia.java.



Figura 9.1.1.5: Clase ProvinciaService.java

ProvinciaService.java	
<b>Atributos</b>	<b>EntityManager:</b> Instancia encargada de gestionar las entidades de la aplicación. Mediante su utilización se realizan todas las transacciones disponibles contra la base de datos.
<b>Métodos</b>	<b>obtenerProvinciasPais (Long):</b> Se indica por parámetro el identificador del país. Devuelve una lista con todas las provincias disponibles del país indicado por parámetro.

- **MunicipioService.java:** Bean local encargado de ejecutar la lógica de negocio de las peticiones realizadas sobre la entidad municipio definida en la clase Municipio.java.



Figura 9.1.1.6: Clase MunicipioService.java

MunicipioService.java	
<b>Atributos</b>	<b>EntityManager:</b> Instancia encargada de gestionar las entidades de la aplicación. Mediante su utilización se realizan todas las transacciones disponibles contra la base de datos.
<b>Métodos</b>	<b>obtenerMunicipioProvincia (Long):</b> Se indica por parámetro el identificador de una provincia. Devuelve una lista con todos los municipios disponibles de la provincia indicada por parámetro.

- **NotificacionService.java:** Bean local encargado de ejecutar la lógica de negocio de las peticiones realizadas sobre la entidad notificación definida en la clase Notificacion.java.



Figura 9.1.1.7: Clase NotificacionService.java

NotificacionService.java	
<b>Atributos</b>	<b>EntityManager:</b> Instancia encargada de gestionar las entidades de la aplicación. Mediante su utilización se realizan todas las transacciones disponibles contra la base de datos.
<b>Métodos</b>	<p><b>obtenerConfiguracionNotificaciones (Long):</b> Devuelve un objeto de la entidad notificación indicando la configuración de las notificaciones del usuario pasado por parámetro.</p> <p><b>modificarNotificaciones (Long, RequestModificarNotificacion):</b> Se indica por parámetro el identificador de un usuario y los datos de configuración de las notificaciones. Devuelve un objeto de la entidad notificación del usuario una vez se han realizado las modificaciones indicadas en el objeto de la clase RequestModificarNotificacion.</p>

- **UbicacionService.java:** Bean local encargado de ejecutar la lógica de negocio de las peticiones realizadas sobre la entidad ubicación definida en la clase Ubicacion.java.

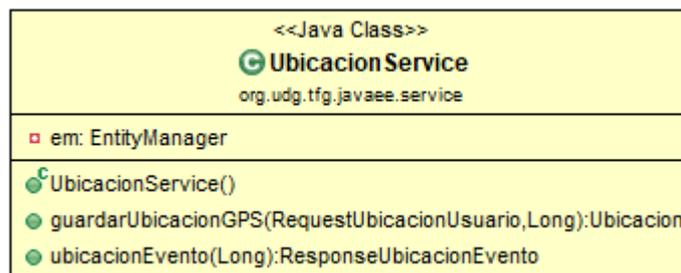


Figura 9.1.1.8: Clase UbicacionService.java

UbicacionService.java	
<b>Atributos</b>	<b>EntityManager:</b> Instancia encargada de gestionar las entidades de la aplicación. Mediante su utilización se realizan todas las transacciones disponibles contra la base de datos.
<b>Métodos</b>	<b>guardarUbicacionGPS (RequestUbicacionUsuario, Long):</b> Devuelve un objeto de la entidad ubicación una vez se ha registrado para el usuario, pasado por parámetro, los datos de la ubicación a registrar del objeto de la clase RequestUbicacionUsuario.

	<b>ubicacionEvento(Long):</b> Devuelve los datos de la ubicación del evento indicado por parámetro.
--	---

- **ParticipanteService.java:** Bean local encargado de ejecutar la lógica de negocio de las peticiones relacionadas con el registro o borrado de los usuarios en los eventos deportivos.



Figura 9.1.1.9: Clase ParticipanteService.java

ParticipanteService.java	
<b>Atributos</b>	<p><b>EntityManager:</b> Instancia encargada de gestionar las entidades de la aplicación. Mediante su utilización se realizan todas las transacciones disponibles contra la base de datos.</p> <p><b>Event&lt;NotificacionUsuarioEvento&gt;:</b> Evento de tipo NotificacionUsuarioEvento que guarda una instancia de la entidad usuario y evento. Permite disparar eventos con la finalidad de enviar una notificación antes de un suceso durante la lógica de negocio.</p>
<b>Métodos</b>	<p><b>addParticipanteEvento (Long, Long):</b> Se indica por parámetro el identificador de un usuario y el identificador de un evento deportivo. Se encarga de dar de alta a un usuario en un evento deportivo. Si el evento está completo, lo añade en la lista de espera. De lo contrario, lo añade como participante. Envía una notificación al propietario del evento si se añade correctamente al usuario.</p> <p><b>eliminarParticipanteEvento (Long, Long, Long):</b> Se indica por parámetro el identificador del usuario a eliminar, el identificador del evento deportivo y el identificador del usuario que realiza la petición. Se elimina el usuario del evento indicado, ya sea de la lista de espera o de participante. Si el administrador del evento elimina a otro usuario, se le envía una notificación informándole. En cambio, si un usuario se da de baja a si mismo se envía una notificación informando al administrador del evento.</p> <p><b>obtenerParticipantesEvento(Long, Long, Long):</b> Se indica por parámetro el identificador del evento, el identificador del usuario que realiza la petición y el tipo de listado a obtener. Devuelve una lista con los usuarios que pertenecen a un evento en función del tipo indicado. Si se indica 0 se devuelve los participantes del evento, si se indica 1 devuelve los usuarios en cola y si se indica 2 se devuelve una lista de ambos.</p>

- **UsuarioService.java:** Bean local encargado de ejecutar la lógica de negocio de las peticiones realizadas sobre la entidad usuario definida en la clase Usuario.java.

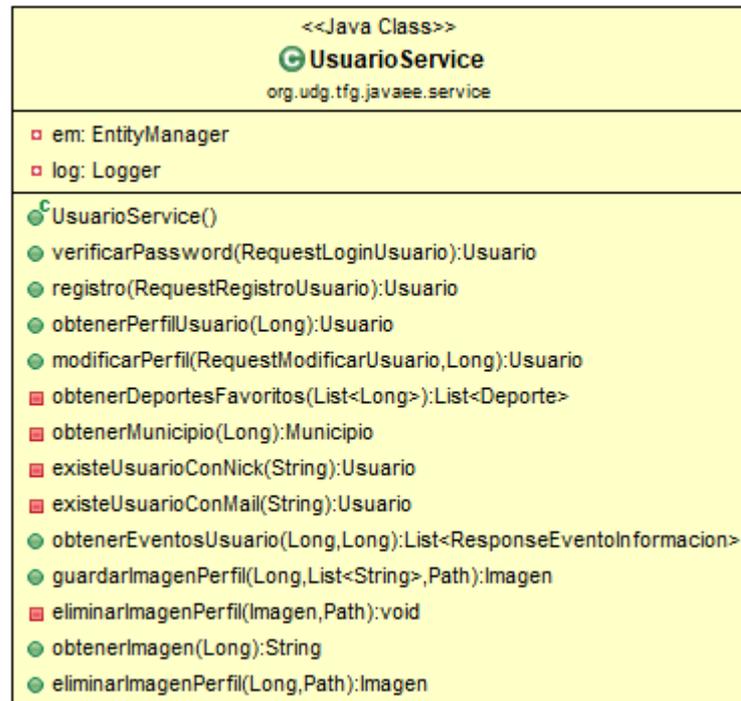


Figura 9.1.1.10: Clase UsuarioService.java

UsuarioService.java	
<b>Atributos</b>	<p><b>EntityManager:</b> Instancia encargada de gestionar las entidades de la aplicación. Mediante su utilización se realizan todas las transacciones disponibles contra la base de datos.</p> <p><b>Logger:</b> Permite registrar acciones llevadas a cabo durante los métodos.</p>
<b>Métodos</b>	<p><b>verificarPassword (RequestLoginUsuario):</b> Se indica por parámetro los datos recibidos ante una petición de inicio de sesión. Se encarga de verificar que la dirección de correo electrónica y la contraseña son válidas. Además, guarda el token que identifica el dispositivo móvil para enviar notificaciones.</p> <p><b>Registro (RequestRegistroUsuario):</b> Se indica por parámetro los datos recibidos ante una petición de registro de un nuevo usuario. Se encarga de validar y dar de alta un nuevo usuario en la aplicación. Además, registra el token que identifica el dispositivo móvil para enviar notificaciones.</p> <p><b>ObtenerPerfilUsuario (Long):</b> Se indica por parámetro el identificador del usuario del cual se quiere obtener la información de su perfil. Devuelve un objeto de la entidad usuario.</p> <p><b>ModificarPerfil (RequestModificarUsuario, Long):</b> Se indica por parámetro los datos a registrar en el perfil del usuario y el identificador del usuario al que hay que asignarle los datos. Devuelve un objeto de la entidad usuario.</p> <p><b>ObtenerEventosUsuario (Long, Long):</b> Se indica por parámetro el identificador del</p>

	<p>usuario y el tipo de eventos a obtener. Si se indica 0 se devuelven los eventos creados por el usuario, si se indica 1 se devuelven los eventos en el que el usuario es participante y si se indica 2 se devuelven los eventos en el que el usuario está en cola.</p> <p><b>GuardarImagenPerfil (Long, List&lt;String&gt;, Path):</b> Se indica por parámetro el identificador del usuario, el nombre de la imagen a subir y el directorio del sistema de ficheros utilizado en donde se guardan las imágenes. Se encarga de eliminar del sistema de ficheros la imagen anterior y de vincular al usuario la nueva imagen, guardando en la entidad imagen el nombre pasado por parámetro.</p> <p><b>ObtenerImagen (Long):</b> Se indica por parámetro el identificador del usuario. Se obtiene la ruta en el sistema de ficheros de la imagen del perfil del usuario.</p>
--	--

- **EventoService.java:** Bean local encargado de ejecutar la lógica de negocio de las peticiones realizadas sobre la entidad evento definida en la clase Evento.java.

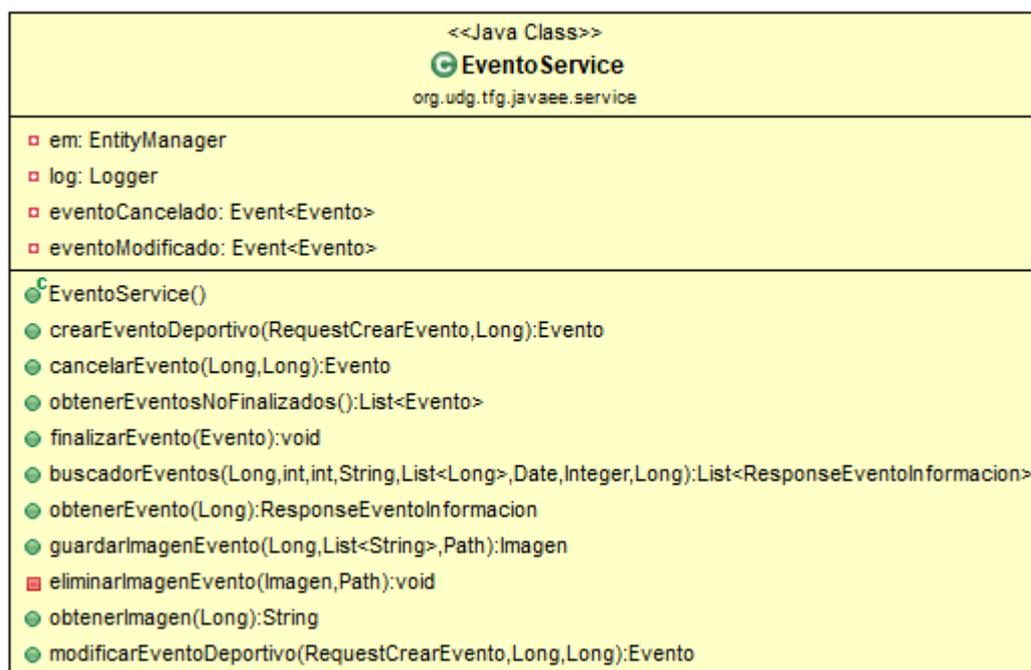


Figura 9.1.1.11: Clase EventoService.java

EventoService.java	
<b>Atributos</b>	<p><b>EntityManager:</b> Instancia encargada de gestionar las entidades de la aplicación. Mediante su utilización se realizan todas las transacciones disponibles contra la base de datos.</p> <p><b>Logger:</b> Permite registrar acciones llevadas a cabo durante los métodos.</p> <p><b>Event&lt;Evento&gt;:</b> Evento de tipo Evento que guarda una instancia de la entidad evento. Permite disparar eventos con la finalidad de enviar una notificación antes un suceso durante la lógica de negocio.</p>
<b>Métodos</b>	<p><b>CrearEventoDeportivo (RequestCrearEvento, Long):</b> Se indica por parámetro los datos recibidos ante una petición de creación de un nuevo evento deportivo y el identificado del usuario. Valida y crea en base de datos un nuevo evento deportivo</p>

	<p>asociado al usuario.</p> <p><b>CancelarEvento (Long, Long):</b> Se indica por parámetro el identificador del usuario y el identificador del evento deportivo. Verifica que el usuario sea el administrador del evento y, si no ha finalizado, modifica el estado del evento a suspendido. Notifica a los usuarios participantes del evento de su cancelación.</p> <p><b>BuscadorEventos (Long, int, int, String, List&lt;Long&gt;, Date, Integer, Long):</b> Se indica por parámetro el identificador del usuario, el límite de eventos a devolver, la posición a partir de la cual devolver eventos, el título, lista de identificadores de categorías deportivas, fecha del evento, distancia e identificador de un municipio. Según el valor de los parámetros realiza una consulta a base de datos y devuelve un listado con los eventos resultantes.</p> <p><b>ObtenerEvento (Long):</b> Se indica por parámetro el identificado del evento. Se devuelve un objeto con la información del evento solicitado.</p> <p><b>GuardarImagenEvento (Long, List&lt;String&gt;, Path):</b> Se indica por parámetro el identificador del evento, el nombre de la imagen a subir y el directorio del sistema de ficheros utilizado en donde se guardan las imágenes. Se encarga de eliminar del sistema de ficheros la imagen anterior y de vincular al evento la nueva imagen guardando en la entidad imagen el nombre pasado por parámetro.</p> <p><b>ObtenerImagen (Long):</b> Se indica por parámetro el identificador del evento. Se obtiene la ruta en el sistema de ficheros de la imagen del evento.</p> <p><b>ModificarEventoDeportivo (RequestCrearEvento, Long, Long):</b> Se indica por parámetro los datos del evento a modificar, el identificador del usuario que realiza la modificación y el identificador del evento a modificar. Valida y modifica los datos del evento deportivo. Notifica a los usuarios participantes del evento de que se ha realizado una modificación en el evento deportivo.</p>
--	--

- **Org.udg.tfg.javee.rest:** Contiene las clases que definen los puntos de entrada a la aplicación mediante la API REST. Especifica y procesa las peticiones disponibles utilizando los Beans definidos en las clases del paquete *Org.udg.tfg.javee.service* que ejecutan la lógica de negocio ante las peticiones recibidas. Finalmente, se encargan de enviar la respuesta a la petición recibida. Son las clases que definen la API REST.
  - **MyApplication.java:** Clase que hereda de *Application* y define el URI base de todos los recursos. En este caso el URI base es */rest*.

```

@ApplicationPath("/rest")
public class MyApplication extends Application {

}

```

- **DeporteRestService.java:** Clase encargada de definir las peticiones de la API REST relacionadas con la gestión de las categorías deportivas y devolver la respuesta obtenida del Bean DeporteService.java.



Figura 9.1.1.12: Clase DeporteRestService.java

- **PaisRestService.java:** Clase encargada de definir las peticiones de la API REST relacionadas con la gestión de los países y devolver la respuesta obtenida del Bean PaisService.java.

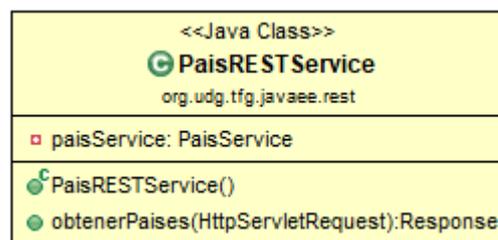


Figura 9.1.1.13: Clase PaisRestService.java

- **ProvinciaRestService.java:** Clase encargada de definir las peticiones de la API REST relacionadas con la gestión de las provincias y devolver la respuesta obtenida del Bean ProvinciaService.java.



Figura 9.1.1.14: Clase ProvinciaRestService.java

- **MunicipioRestService.java:** Clase encargada de definir las peticiones de la API REST relacionadas con la gestión de los municipios y devolver la respuesta obtenida del Bean MunicipioService.java.



Figura 9.1.1.15: Clase MunicipioRestService.java

- **UbicacionRestService.java:** Clase encargada de definir las peticiones de la API REST relacionadas con la gestión de la ubicación GPS de usuarios y eventos y devolver la respuesta obtenida del Bean UbicacionService.java.

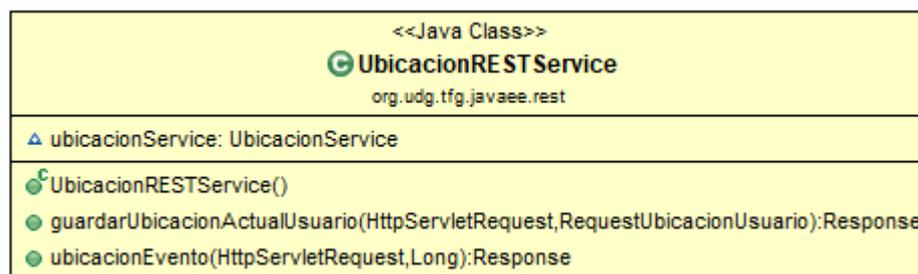


Figura 9.1.1.16: Clase UbicacionRestService.java

- **ParticipanteRestService.java:** Clase encargada de definir las peticiones de la API REST relacionadas con la gestión de las altas y bajas de usuarios en los eventos deportivos y devolver la respuesta obtenida del Bean ParticipanteService.java.



Figura 9.1.1.17: Clase ParticipanteRestService.java

- **ImagenRestService.java:** Clase encargada de definir las peticiones de la API REST relacionadas con la gestión de las imágenes de los usuarios y eventos deportivos y devolver las respuestas obtenidas de los Beans UsuarioService.java o EventoService.java.

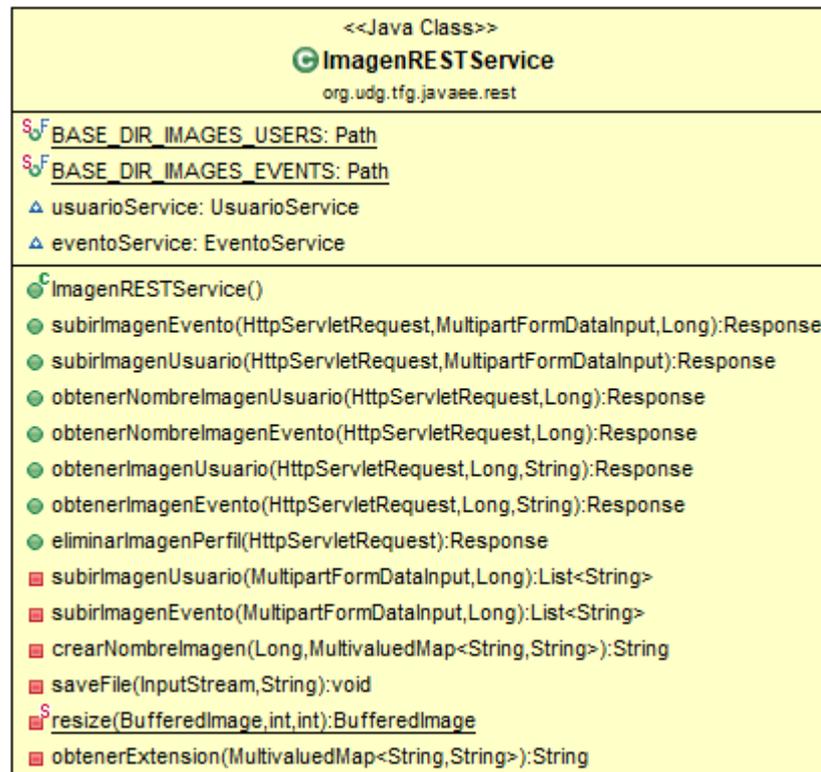


Figura 9.1.1.18: Clase ImagenRestService.java

Se declaran las rutas en el sistema de ficheros utilizado por el servidor para guardar las imágenes en función de si se trata de la imagen de un usuario o un evento deportivo.

```
public static final java.nio.file.Path BASE_DIR_IMAGES_USERS = Paths
    .get(System.getenv(Global.VARIABLE_ENTORNO_IMAGENES_USUARIOS));
public static final java.nio.file.Path BASE_DIR_IMAGES_EVENTS = Paths
    .get(System.getenv(Global.VARIABLE_ENTORNO_IMAGENES_EVENTOS));
```

- **UsuarioRestService.java:** Clase encargada de definir las peticiones de la API REST relacionadas con la gestión de los usuarios y devolver las respuestas obtenidas del Bean UsuarioService.java.



Figura 9.1.1.19: Clase UsuarioRestService.java

- **EventoRestService.java:** Clase encargada de definir las peticiones de la API REST relacionadas con la gestión de los eventos deportivos y devolver las respuestas obtenidas del Bean EventoService.java.

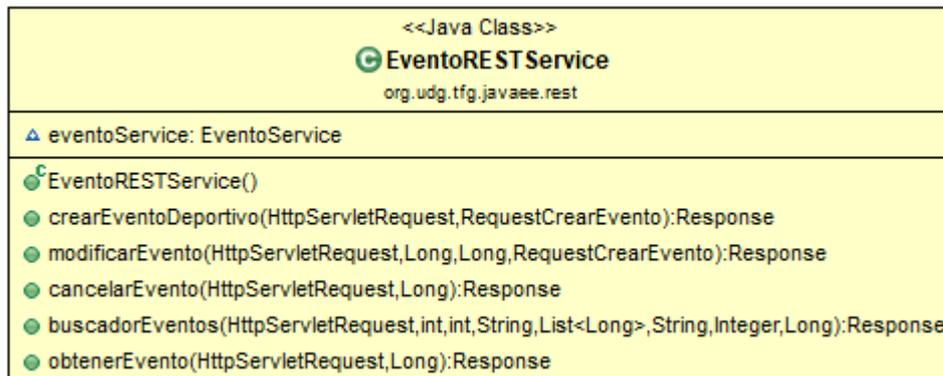


Figura 9.1.1.20: Clase EventoRestService.java

- **NotificacionRestService.java:** Clase encargada de definir las peticiones de la API REST relacionadas con la gestión de las notificaciones y devolver las respuestas obtenidas del Bean NotificacionService.java.



Figura 9.1.1.21: Clase NotificacionRestService.java

- **GenericRestService.java:** Clase que encapsula métodos destinados al control de las sesiones de los usuarios y a procesar las respuestas a devolver a las peticiones realizadas. Las clases del paquete *Org.udg.tfg.javee.rest*, heredan esta clase para utilizar dichos métodos.

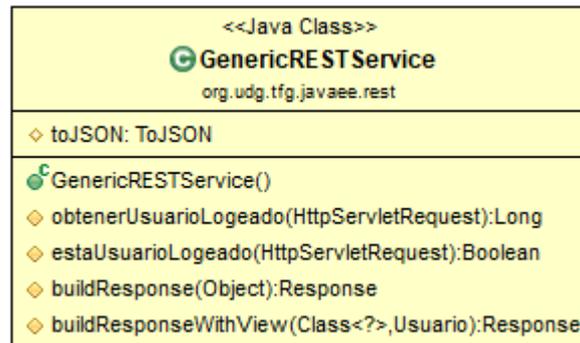


Figura 9.1.1.22: Clase GenericRestService.java

Mediante el método *obtenerUsuarioLogeado ()* podemos obtener el identificador de la sesión del usuario que realiza la petición:

```

protected Long obtenerUsuarioLogeado(@Context HttpServletRequest req) {
    HttpSession session = req.getSession();

    if (session == null) {
        throw new WebApplicationException("Error al obtener la sesión");
    }

    return (Long) session.getAttribute(Global.AUTH_ID);
}
  
```

Con el método *estaUsuarioLogeado ()* verificamos si el usuario que realiza la petición se encuentra logeado en el sistema:

```

protected Boolean estaUsuarioLogeado(@Context HttpServletRequest req) {
    HttpSession session = req.getSession();

    if (session == null) {
        throw new WebApplicationException("Error al obtener la sesión");
    }

    return ((Long) session.getAttribute(Global.AUTH_ID)) != null;
}
  
```

Con el método *buildResponse* () construimos la respuesta JSON mediante un objeto:

```
protected Response buildResponse(Object o) {
    try {
        return Response.ok(toJSON.Object(o)).build();
    } catch (IOException e) {
        throw new WebApplicationException("Error al serializar la respuesta");
    }
}
```

- **Org.udg.tfg.javee.request:** Contiene las clases encargadas de guardar los datos recibidos en las peticiones realizadas a la API REST.

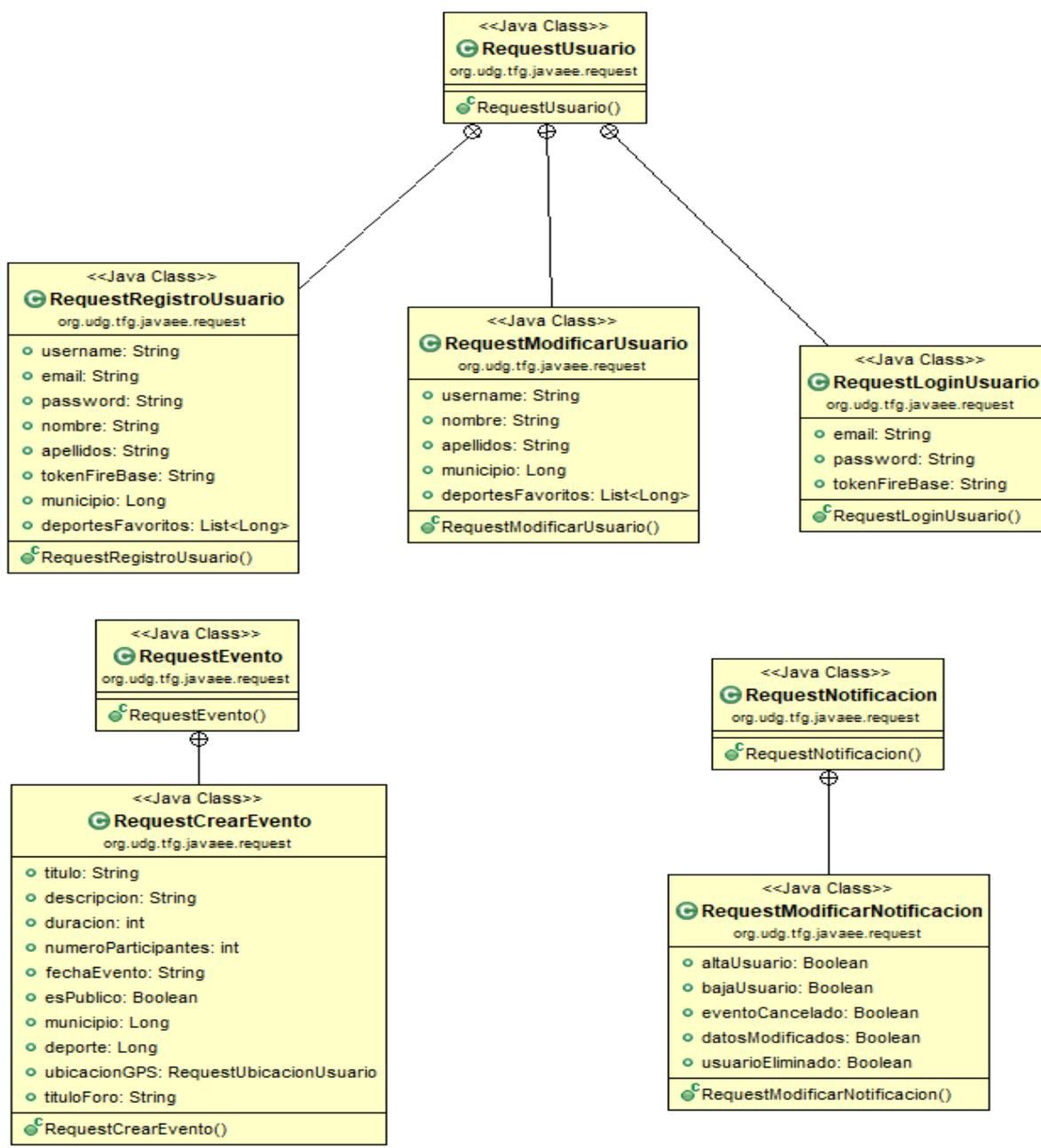


Figura 9.1.1.23: Clases del paquete **Org.udg.tfg.javee.request**

- **Org.udg.tfg.javae.response:** Contiene las clases encargadas de guardar los datos de respuesta a una petición realizada a la API REST.

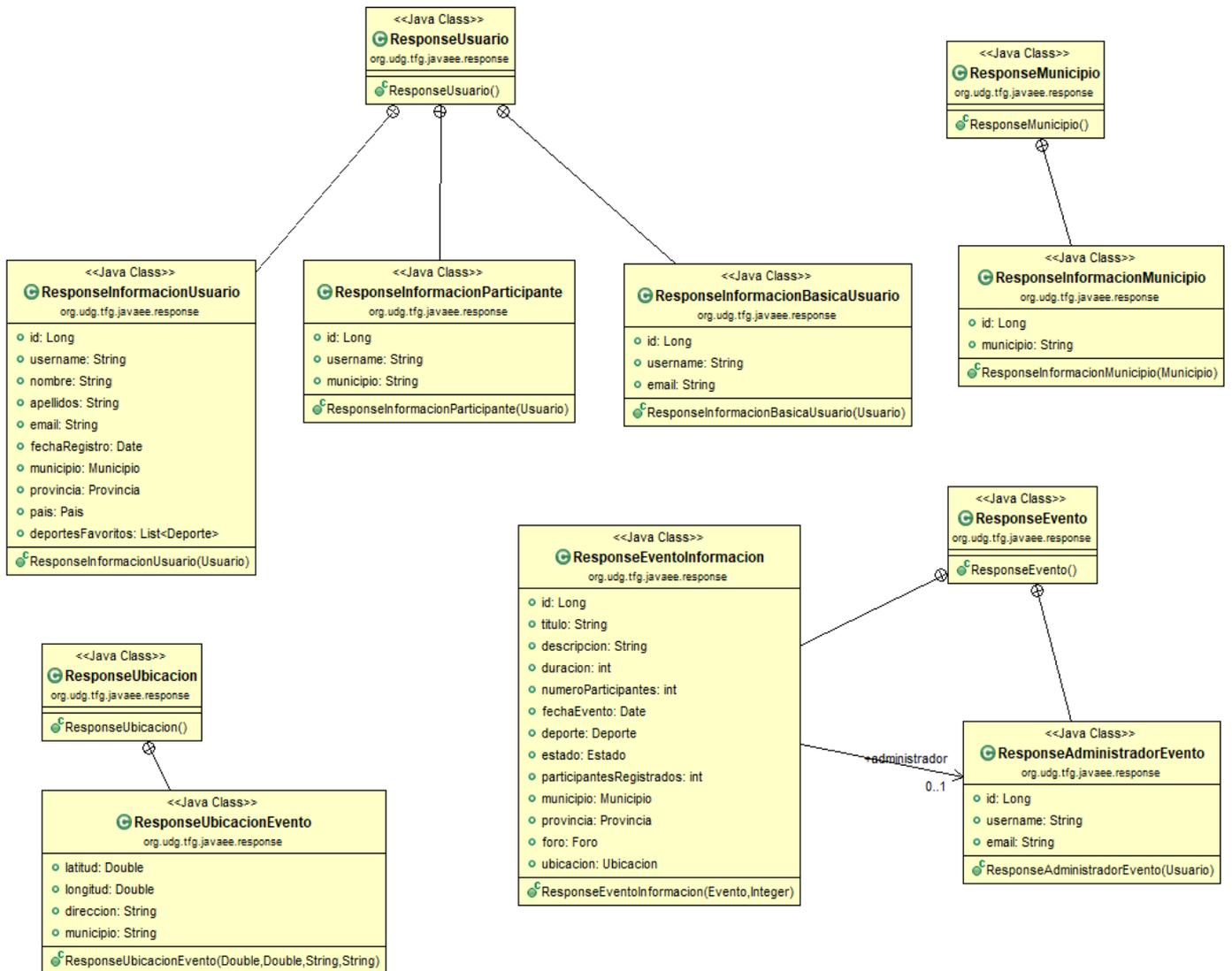


Figura 9.1.1.24: Clases del paquete **Org.udg.tfg.javae.response**

- **Org.udg.tfg.javee.response:** Contiene clases con diversas funcionalidades que pueden ser utilizadas varias veces en diferentes clases.

Destacar la clase HashPassword.java utilizada para cifrar las contraseñas de los usuarios de la aplicación. En esta clase se cifran y se validan las contraseñas de los usuarios mediante el algoritmo PBKDF2 [26] (*Password Based Key Derivation Function 2*). Los pasos para el cifrado de la contraseña son:

1. Generar una semilla o Salt diferente para cada usuario. En nuestro caso la dirección de correo electrónica.
2. Aplicar una función de cifrado tipo Hash al conjunto de contraseña más la semilla utilizada. En este caso se utiliza la función criptográfica SHA512.
3. Iterar el proceso un cierto número de veces. En este caso 1000 iteraciones.
4. Finalmente, se obtiene la contraseña cifrada a guardar en la base de datos con una longitud de 512 bits.

Para realizar la verificación de la contraseña repetiremos el mismo procedimiento y compararemos la contraseña resultante con la guardada en base de datos. En la figura 9.1.1.25 se muestra el código utilizado en la clase HashPassword.java.

```
public final class HashPassword {

    private static final int NUMERO_ITERACIONES = 1000;

    private static final int LONGITUD_HASH = 512;

    public static String passwordHash(String password, String salt) {
        try {
            char[] chars = password.toCharArray();
            PBEKeySpec spec = new PBEKeySpec(chars, salt.getBytes(), NUMERO_ITERACIONES, LONGITUD_HASH);
            SecretKeyFactory skf = SecretKeyFactory.getInstance("PBKDF2WithHmacSHA512");
            byte[] hash = skf.generateSecret(spec).getEncoded();
            return convertirAHexadecimal(hash);
        } catch (Exception e) {
            return null;
        }
    }

    public static boolean validarPassword(String originalPassword, String salt, String storedPassword) {
        return passwordHash(originalPassword, salt).equals(storedPassword);
    }

    private static String convertirAHexadecimal(byte[] array) {
        BigInteger bi = new BigInteger(1, array);
        String hex = bi.toString(16);
        int paddingLength = (array.length * 2) - hex.length();
        if (paddingLength > 0) {
            return String.format("%0" + paddingLength + "d", 0) + hex;
        } else {
            return hex;
        }
    }
}
```

Figura 9.1.1.25: Algoritmo cifrado y descifrado de contraseñas

- **Org.udg.tfg.javее.notification:** Contiene clases encargadas de procesar y enviar las notificaciones a los dispositivos móviles destinatarios. Durante la ejecución de los métodos de la lógica de negocio hay casuísticas definidas que disparan un evento con la finalidad de enviar a los destinatarios una notificación informativa.



Figura 9.1.1.26: Clases del paquete **Org.udg.tfg.javее.notification**

Para diferenciar los distintos tipos de notificaciones a enviar se han definido 5 nuevas anotaciones (**@Bajausuario**, **@DesapuntadoEvento**, **@EventoModificado**, **@EventoCancelado** y **@AltaUsuario**). Definiendo los eventos disparados con estas anotaciones, permitimos a la clase **EventoListener.java** diferenciar que tipo de notificación tiene que tratar, ya que en esta clase se interceptan los eventos y se construye el contenido de la notificación y los destinatarios mediante la clase **NotificacionListener.java**. Una vez se ha configurado la notificación a enviar la clase **EnviarNotificacion.java**, se encarga de realizar la petición al servidor Firebase de Google para que distribuya la notificación a los destinatarios indicados mediante los tokens.

Para entender mejor el procedimiento del envío de notificaciones seguiremos un ejemplo de un evento cancelado. Recordar que tanto en el inicio de sesión como en el registro de un usuario, se guarda un token identificativo por cada dispositivo móvil que se utilizará más adelante para enviar la notificación.

En primer lugar, en la clase EventoService.java se define un evento del tipo Evento.java mediante la anotación @EventoCancelado:

```
@Inject
@EventoCancelado
private Event<Evento> eventoCancelado;
```

En cuanto se recibe una petición a la API REST para cancelar un evento, en el método de la clase EventoService.java encargado de la lógica de negocio se dispara el evento pasándole por parámetro el objeto evento cancelado:

```
eventoCancelado.fire(evento);
```

Una vez se dispara el evento, la clase EventoListener.java intercepta el evento y determina su tipo mediante la anotación asignada. En este caso @EventoCancelado y procede a configurar la notificación creando el contenido de la notificación y rellenando los destinatarios:

```
// De forma asíncrona
// Varios eventos en paralelo
@Asynchronous
@Lock(LockType.READ)
public void notificarEventoCancelado(@Observes(during =
TransactionPhase.AFTER_SUCCESS) @EventoCancelado Evento e) {
    List<Usuario> participantes = new ArrayList<>();
    for (Usuario u : e.getParticipantes()) {
        // No es el administrador y el participante tiene activadas notificaciones
        // de eventos cancelados
        if (e.getAdministrador().getId() != u.getId() &&
            u.getNotificacion().getEventoCancelado())
            participantes.add(u);
    }
    if (participantes != null && !participantes.isEmpty()) {
        destinatarios.addAll(participantes);
        contenidoNotificacion.put("title", e.getTitulo());
        contenidoNotificacion.put("message", "El evento ha sido cancelado");
        contenidoNotificacion.put("notificationType",
            Global.NOTIFICACION_EVENTO_CANCELADO);
        contenidoNotificacion.put("idEvento", e.getId().toString());
        enviarNotificacion();
    }
}
```

Una vez configurada la notificación se utiliza el método `enviarNotificación()` de la clase `NotificationListener.java`. En este método se construye el objeto `Notificacion` rellenando los tokens del destinatario y asignando el contenido de la notificación:

```
protected void enviarNotificacion() {
    Notificacion notificacion = new Notificacion();

    for (Usuario u : destinatarios) {
        notificacion.addToken(u.getTokenFireBase());
    }

    notificacion.crearNotificacion(this.contenidoNotificacion);
    EnviarNotificacion.enviarNotificacion(notificacion);
}
```

Finalmente, se envía la petición de envío de notificación al servidor de Firebase mediante la clase `EnviarNotificación.java`:

```
public static void enviarNotificacion(Notificacion notificacion) {

    try {

        // Abrir conexión URL de google FireBase
        URL url = new URL(Global.URL_FIREBASE_GOOGLE);
        HttpURLConnection conn = (HttpURLConnection) url.openConnection();

        // Configurar parámetros petición
        conn.setRequestMethod("POST");
        conn.setRequestProperty("Content-Type", "application/json");
        conn.setRequestProperty("Authorization", "key=" + Global.API_KEY_FIREBASE);
        conn.setDoOutput(true);

        // Añadir contenido de la notificación en un JSON a la petición
        ObjectMapper mapper = new ObjectMapper();
        DataOutputStream wr = new DataOutputStream(conn.getOutputStream());
        mapper.writeValue(wr, notificacion);
        wr.flush();
        wr.close();

        // Procesar respuesta
        int responseCode = conn.getResponseCode();
        System.out.println("Enviando petición POST a la URL: " + url.toString());
        System.out.println("Código de respuesta: " + responseCode);

        BufferedReader in = new BufferedReader(new InputStreamReader(conn.getInputStream()));
        String inputLine;
        StringBuffer response = new StringBuffer();

        while ((inputLine = in.readLine()) != null) {
            response.append(inputLine);
        }
        in.close();
        System.out.println("Contenido respuesta: " + response.toString());

    } catch (Exception e) {
        System.out.println("ERROR al enviar la notificación");
    }

}
```

### 9.1.3 Modelo de clases

Una vez explicada la estructura del proyecto y la funcionalidad de los diversos paquetes de clases utilizados, en la figura 9.1.3.1 se observa el diagrama de clases de la aplicación servidor en función de los 3 paquetes que conforman el flujo principal de la aplicación.

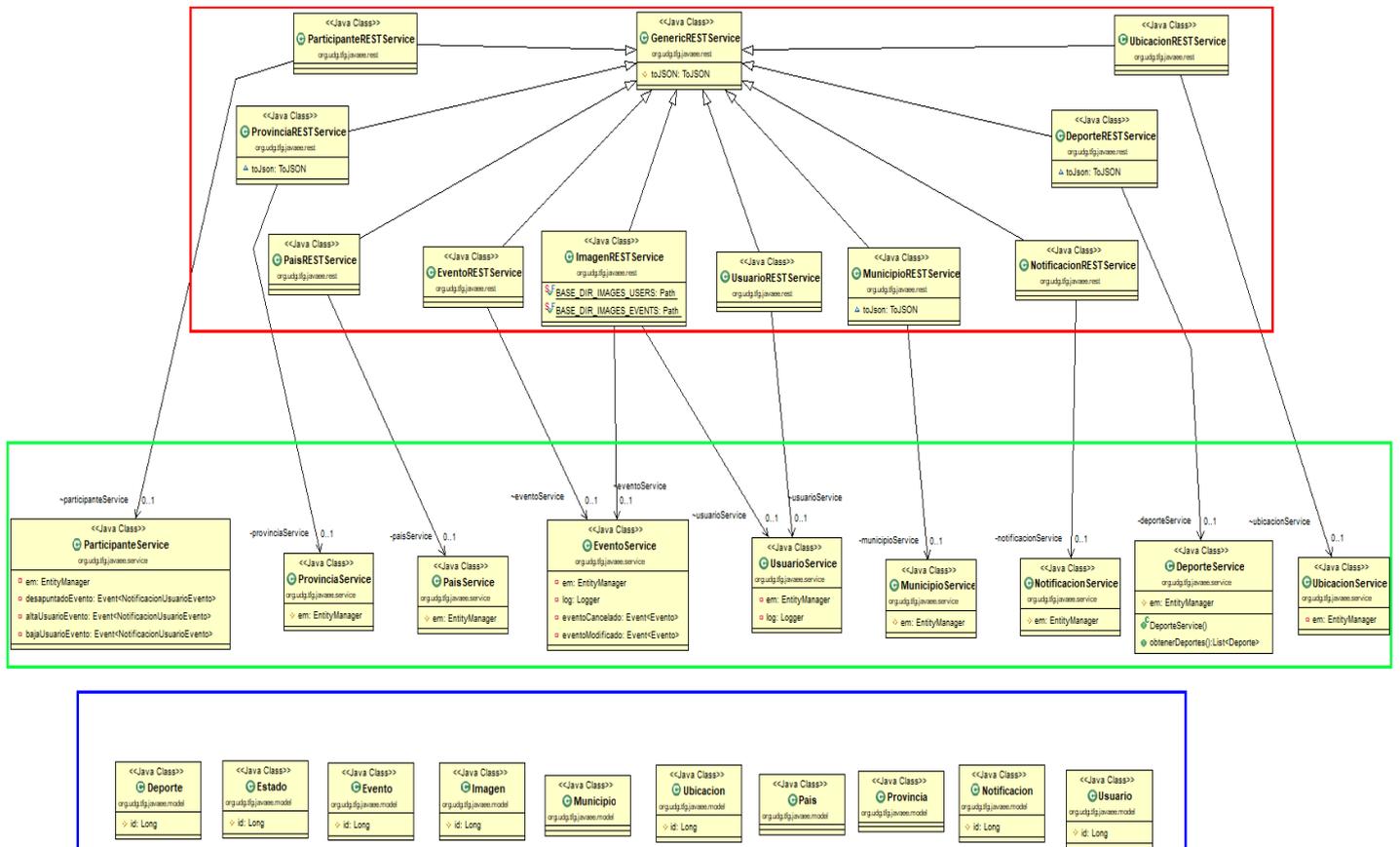


Figura 10.1.3.1: Diagrama de clases

La principal finalidad de la aplicación servidor es la de proporcionar un servicio Web basado en el estándar REST, recibir peticiones de los dispositivos clientes, procesar las peticiones recibidas, guardar los datos y generar una respuesta.

En el diagrama las clases del recuadro rojo conforman la definición de las llamadas disponibles a la API REST y se encargan de recibir las peticiones y servir la respuesta. Para llevar a cabo su funcionalidad utilizan las clases que se encargan de la lógica de negocio que son las clases representadas en el recuadro verde. Estas clases se encargan de ejecutar toda la lógica en función de las peticiones y datos proporcionados y de guardar los datos en las clases que conforman el modelo de datos representados en el recuadro azul. Mediante el objeto EntityManager realizan las operaciones con las entidades.

## 9.1.4 Buscador eventos

Una de las funcionalidades más importantes en la aplicación es la implementación del buscador de eventos. En la figura 9.1.4.1 se observa mediante el diagrama de clases el flujo que sigue una petición de búsqueda de eventos personalizada.

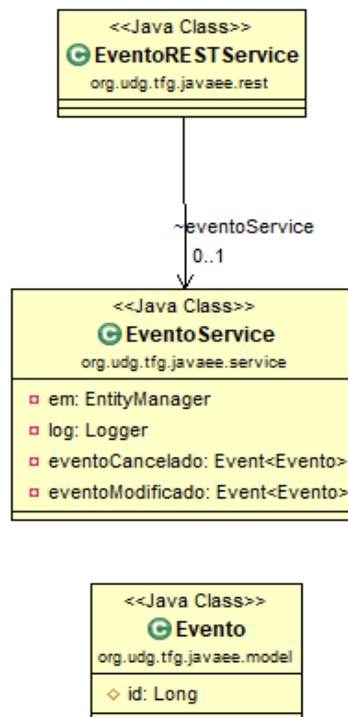


Figura 11.1.4.1: Diagrama clases buscador eventos

En primer lugar, se ha definido la llamada en la clase `EventoRestService.java`, definiendo el tipo de petición como GET y los diferentes parámetros que acepta:

```
@GET
@Produces(MediaType.APPLICATION_JSON)
public Response buscadorEventos(@Context HttpServletRequest req,
    @DefaultValue("10") @QueryParam("limite") int limite, // cantidad de resultados a partir del offset indicado
    @DefaultValue("0") @QueryParam("offset") int offset, // posición del primer evento en la lista obtenida
    @DefaultValue("") @QueryParam("titulo") String titulo, @QueryParam("deportes") final List<Long> deportes,
    @QueryParam("fechaEvento") String fechaEvento,
    @DefaultValue("-1") @QueryParam("distancia") Integer distancia,
    @DefaultValue("-1") @QueryParam("municipio") Long municipio) {
    if (estaUsuarioLogeado(req)) {
        SimpleDateFormat sdf = new SimpleDateFormat("dd/MM/yyyy");
        Date fecha = null;
        try {
            if (fechaEvento != null && !fechaEvento.isEmpty())
                fecha = sdf.parse(fechaEvento);
            List<ResponseEvento.ResponseEventoInformacion> eventos = eventoService.buscadorEventos(
                obtenerUsuarioLogeado(req), limite, offset, titulo, deportes, fecha, distancia, municipio);
            return buildResponse(eventos);
        } catch (ParseException e) {
            throw new WebApplicationException("Error en la búsqueda de eventos");
        }
    } else {
        throw new WebApplicationException("No ha iniciado sesión");
    }
}
```

Se observa cómo se aceptan 7 parámetros diferentes a la hora de realizar una búsqueda personalizada y sus valores por defecto en caso de no estar informados en la petición. Estos parámetros se incluyen en la URI de la petición GET. Un ejemplo de petición:

GET ▾	http://localhost:8080/rest/evento?limite=15&offset=0&deportes=1&deportes=6&titulo=ciclismo&distancia=10	Params
-------	---	--------

En este ejemplo, se solicitan los primeros 15 eventos a partir de la posición 0 del listado devuelto. Se buscarán los eventos tal que su categoría deportiva tenga como identificador en base de datos el 1 o el 6, que en su título contenga la palabra ciclismo y se encuentren como máximo a 10 kilómetros de distancia de la ubicación GPS registrada del dispositivo móvil que realiza la petición. Una vez se procesan los datos de entrada, se llama al método *BuscadorEventos* () de la clase *EventoService.java* que se ocupará de construir la consulta y devolver el listado de eventos deportivos resultantes.

En el proceso de crear la consulta, en primer lugar, se descartan los eventos administrados por el usuario que realiza la petición y los eventos que su estado sea cancelado o finalizado:

```
Query consulta;  
String consultaString = "select e , e.participantes.size from Evento e where e.administrador.id <> :idUserario ";  
consultaString += "and e.estado.id <> :idCancelado and e.estado.id <> :idFinalizado ";
```

A continuación, se construye la consulta en función de los parámetros recibidos en la URI:

```
if (titulo != null && !titulo.isEmpty()) {  
    consultaString += "and e.titulo like :titulo ";  
}  
  
if (deportes != null && !deportes.isEmpty()) {  
    for (int i = 0; i < deportes.size(); i++) {  
        if (i == 0)  
            consultaString += "and (e.deporte.id = :deporte" + i + " ";  
        else {  
            consultaString += "or e.deporte.id = :deporte" + i + " ";  
        }  
    }  
    consultaString += ") ";  
}  
  
if (fechaEvento != null) {  
    consultaString += "and year(e.fechaEvento) = :year and month(e.fechaEvento) = :month and day(e.fechaEvento) = :day ";  
}
```

En cuanto a la ubicación del evento se puede escoger entre dos opciones. Por un lado, se puede indicar el identificador de un municipio registrado en la base de datos o indicar una distancia máxima en kilómetros respecto a la ubicación GPS registrada del dispositivo móvil. En caso de que se indique una distancia y no se dispongan de coordenadas GPS registradas del dispositivo móvil se cogerán como referencia las coordenadas del municipio indicado en su perfil.

```

if (distancia != null && distancia >= 0) {
    if (distancia != 0) {
        if (usuario.getUbicacionGPS() != null) {
            latitud = usuario.getUbicacionGPS().getLatitud();
            longitud = usuario.getUbicacionGPS().getLongitud();
        } else {
            if (usuario.getMunicipio() != null) {
                latitud = usuario.getMunicipio().getLatitudEstimada();
                longitud = usuario.getMunicipio().getLongitudEstimada();
            }
        }
        consultaString += "and ((e.ubicacionGPS is not null and " + Global.FORMULA_DISTANCIA_GPS
            + " <= :distancia) or (e.ubicacionGPS is null and e.municipio is not null and "
            + Global.FORMULA_DISTANCIA_GPS_ESTIMADA + " <= :distancia )) ";
    }
}
// Escoger eventos por municipio seleccionado
else if (municipio != null && municipio != -1) {
    consultaString += "and e.municipio.id = :municipio ";
}
}

```

Para realizar el cálculo de distancias entre dos coordenadas se utilizada la siguiente fórmula definida en la constante *FORMULA\_DISTANCIA\_GPS\_ESTIMADA*:

```

" ( :km * acos ( cos ( radians(:latitud) ) * cos( radians( e.municipio.latitudEstimada ) ) * "
+ "cos( radians( e.municipio.longitudEstimada ) - radians(:longitud) ) + sin ( radians(:latitud) ) * "
+ "sin( radians( e.municipio.latitudEstimada ))) )";

```

Una vez construida la consulta y asignado el valor de los parámetros, se lanza la consulta mediante el objeto EntityManager que nos devolverá el listado de eventos resultantes ordenados por fecha de celebración. Este listado se devuelve a la clase EventoRestService.java encargada de parsear la lista de eventos en formato JSON.

### 9.1.5 Tratamiento imágenes

En la aplicación, tanto los usuarios como los eventos deportivos pueden tener una imagen asociada. Estas imágenes se guardan en el sistema de ficheros utilizado por el servidor. En la clase entidad Imagen.java se registra el nombre de la imagen en el sistema de ficheros.

En primer lugar, se han definido las variables de entorno que indican la ruta en el que se guardan las imágenes en el sistema de ficheros. En la figura 9.1.5.1 se observa la declaración de las variables en el sistema operativo.

Variables del sistema	
Variable	Valor
DIR_TFG_IMAGES_EVENTS	C:\Users\Dani\TFG\Imágenes\eventos\
DIR_TFG_IMAGES_USERS	C:\Users\Dani\TFG\Imágenes\usuarios\

Figura 9.1.5.1: Variables entorno imágenes

Para obtener el valor de las variables de entorno en la clase `ImagenRestService.java` se declaran las siguientes variables:

```
public static final java.nio.file.Path BASE_DIR_IMAGES_USERS = Paths
    .get(System.getenv(Global.VARIABLE_ENTORNO_IMAGENES_USUARIOS));
public static final java.nio.file.Path BASE_DIR_IMAGES_EVENTS = Paths
    .get(System.getenv(Global.VARIABLE_ENTORNO_IMAGENES_EVENTOS));
```

La clase `ImagenRestService.java` se encarga de definir la petición para subir una imagen asociada a un usuario o a un evento deportivo. En el siguiente código se muestra la implementación de la petición de subida de la imagen de perfil de un usuario:

```
@POST
@Path("/usuario")
@Produces(MediaType.APPLICATION_JSON)
@Consumes(MediaType.MULTIPART_FORM_DATA)
public Response subirImagenUsuario(@Context HttpServletRequest req, MultipartFormDataInput input) {
    if (estaUsuarioLogeado(req)) {
        List<String> imagenSubida = subirImagenUsuario(input, obtenerUsuarioLogeado(req));
        Imagen imagen = usuarioService.guardarImagenPerfil(obtenerUsuarioLogeado(req), imagenSubida,
            BASE_DIR_IMAGES_USERS);
        return buildResponse(imagen);
    } else {
        throw new WebApplicationException("No ha iniciado sesión");
    }
}
```

Se recibe una petición de tipo `MULTIPART_FORM_DATA` en la cual va incluida la imagen a subir. A continuación, el método `subirImagenUsuario ()` se encarga de obtener la imagen y subirla al sistema de ficheros. Para diferenciar las imágenes y que su nombre no coincida se utiliza el siguiente sistema para nombrar las imágenes antes de subirlas al sistema de ficheros:

```
private String crearNombreImagen(Long id, MultivaluedMap<String, String> headers) {
    String extension = obtenerExtension(headers);
    Calendar calendar = Calendar.getInstance();
    calendar.setTime(new Date());
    return "id" + id + "yourSport" + calendar.getTimeInMillis() + "." + extension;
}
```

Finalmente, una vez guardada la imagen en el sistema de ficheros en el objeto `Usuario.java` referente al usuario actual, se guarda un nuevo objeto `Imagen.java` indicando el nombre con el que se ha guardado la imagen:

```

public Imagen guardarImagenPerfil(Long idUsuario, List<String> imagenSubida, Path baseDir) {
    Usuario u = em.find(Usuario.class, idUsuario);
    if (u != null) {
        if (imagenSubida != null && !imagenSubida.isEmpty()) {
            Imagen imagen = u.getImagen();
            eliminarImagenPerfil(imagen, baseDir);
            imagen.setRuta(imagenSubida.get(0));
            return imagen;
        } else
            throw new EJBException("No se puede subir la imagen");
    } else
        throw new EJBException("El usuario no existe");
}

```

En caso de que ya hubiese una imagen de perfil anterior asociada esta se elimina del sistema de ficheros para liberar espacio:

```

private void eliminarImagenPerfil(Imagen imagen, Path baseDir) {
    if (!imagen.getRuta().equals(Global.NO_IMAGEN_PERFIL)) {
        File file = new File(baseDir.toString() + "\\" + imagen.getRuta());
        if (file.delete())
            log.log(Level.INFO, "Imagen eliminada correctamente");
        else
            log.log(Level.INFO, "No se pudo eliminar la imagen");
    }
}

```

## 9.2 Aplicación Android

Este apartado se centra exclusivamente en detallar la implementación llevada a cabo para el desarrollo de la aplicación cliente Android.

### 9.2.1 Estructura del proyecto

A la hora de organizar la implementación de la aplicación Android se ha realizado la siguiente clasificación:

- **AndroidManifest.xml:** se trata del fichero de configuración de la aplicación en donde definimos aspectos como el nombre de la aplicación, la versión, el icono principal, las actividades o servicios a ejecutar y los permisos necesarios para la ejecución de la aplicación.

```
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE" />
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
<uses-permission android:name="android.permission.CAMERA" />
```

Básicamente se solicitan permisos para poder realizar las peticiones a la API REST a través de internet, obtener imágenes de la galería del dispositivo o de la cámara y para poder obtener la ubicación GPS del dispositivo móvil a través del sensor GPS o de la red. En la siguiente captura se muestra la configuración realizada en cuanto al nombre de la aplicación, el icono a utilizar y el estilo base de la interfaz de usuario:

```
android:icon="@mipmap/ic_laurel"
android:label="YourSport"
android:theme="@style/AppTheme">
```

Para la declaración de las actividades en el fichero se utiliza la siguiente estructura:

```
<activity
    android:name=".activity.Registro"
    android:label="YourSport"
    android:screenOrientation="portrait"
    android:windowSoftInputMode="stateHidden|adjustPan" />
<activity
    android:name=".activity.Principal"
    android:label="YourSport"
    android:screenOrientation="portrait"
    android:windowSoftInputMode="stateHidden|adjustPan" />
```

- **Build.gradle:** se trata del fichero de configuración de la compilación. Se definen parámetros como la versión SDK actual y la versión mínima soportada.

```
defaultConfig {
    minSdkVersion 21
    targetSdkVersion 27
    versionCode 1
    versionName "1.0"
}
```

El sistema de compilación administra las dependencias del proyecto desde el sistema de archivos local y desde repositorios remotos. Declarando las dependencias en este fichero nos evitamos tener que descargar y copiar manualmente paquetes ejecutables de las dependencias utilizadas. Algunas dependencias declaradas en el fichero:

```
implementation 'com.squareup.retrofit2:retrofit:2.3.0'
implementation 'com.squareup.retrofit2:converter-gson:2.3.0'
implementation 'com.squareup.okhttp3:logging-interceptor:3.4.2'
```

- **Java:** Dentro del directorio java se implementa el paquete con las diferentes clases de la aplicación. En este caso se diferencia la funcionalidad de las clases en función del directorio en que se encuentran dentro del paquete. En la figura 9.2.1.1 se observan los directorios utilizados dentro del paquete de clases.

```
> activity
> adapter
> entity
> fragment
> service
> util
  MyFirebaseInstanceIdService
  MyFirebaseMessagingService
```

Figura 9.2.1.1: Estructura directorios proyecto

En la carpeta *activity* se encuentran las clases que actúan como las actividades de la aplicación. En la figura 9.2.1.2 se observan las diferentes actividades que conforman la aplicación.



Figura 9.2.1.2: Actividades del proyecto

En la carpeta *adapter* se encuentran las clases que implementan la interfaz *adapter*. Actúan como un enlace entre un conjunto de datos y un adaptador de vista. Son responsables de recuperar la información de un conjunto de datos y generar los objetos *view* mediante los datos. En la figura 9.2.1.3 se observan las clases de tipo adaptador implementadas.

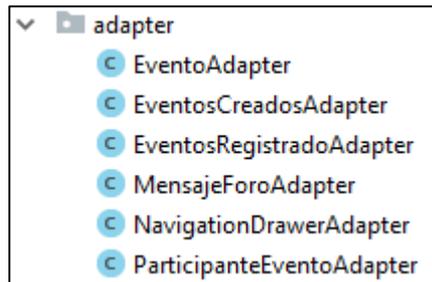


Figura 9.2.1.3: Adaptadores del proyecto

En la carpeta *Entity* se encuentran las clases destinadas a guardar los datos que se enviarán al realizar peticiones al servicio Web o de guardar los datos recibidos en las respuestas. En la figura 9.2.1.4 se observan las clases implementadas.

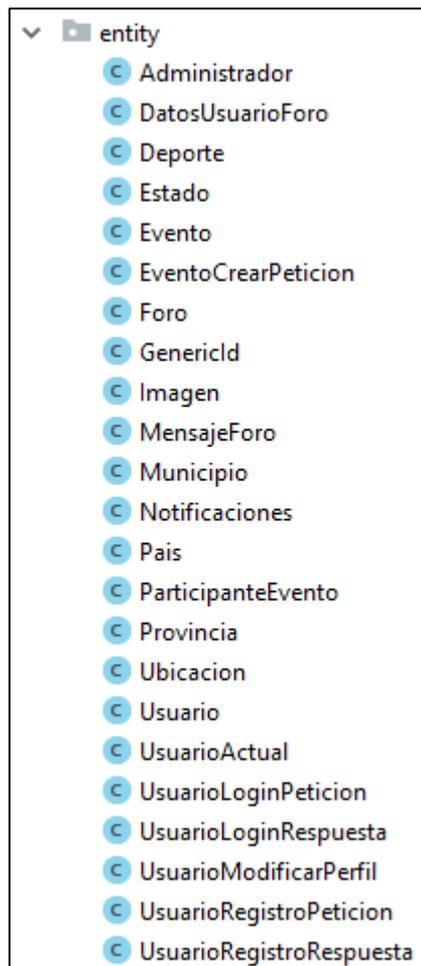


Figura 9.2.1.4: Entidades del proyecto

En la carpeta *fragment* se encuentran las clases que actúan como fragmentos en la aplicación. Estos pueden ser cargados y reutilizados por diferentes actividades. En la figura 9.2.1.5 se observan los fragmentos implementados.

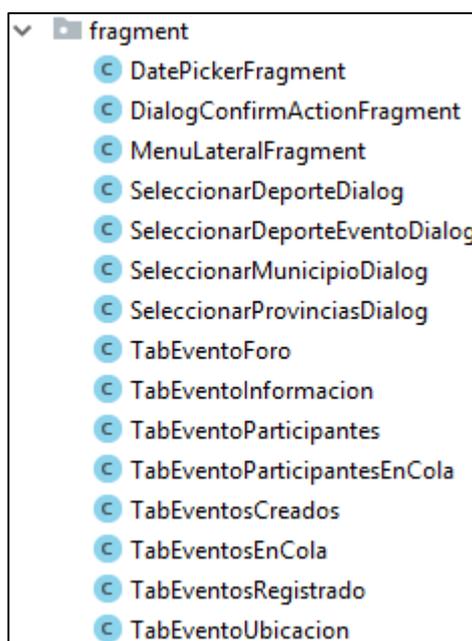


Figura 9.2.1.5: Fragmentos del proyecto

En la carpeta *util* se encuentran las clases auxiliares utilizadas en diferentes puntos de la aplicación. Encapsulan funcionalidades para ser reutilizadas en diversos momentos. En la figura 9.2.1.6 se observan las clases auxiliares implementados.

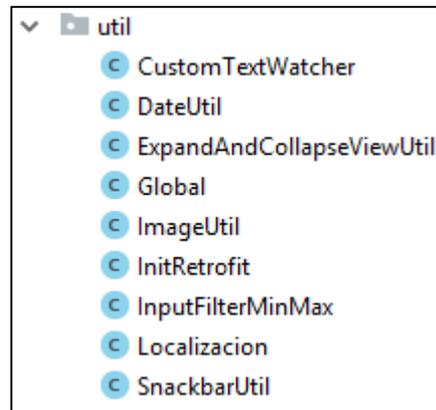


Figura 9.2.1.6: Fragmentos del proyecto

- **res:** Contiene todos los ficheros de recursos necesarios para el proyecto. Se divide en diversos subdirectorios.
  - **Res/drawable:** Contiene imágenes y elementos gráficos utilizados en la aplicación.
  - **Res/layout:** Contiene los ficheros XML que definen las interfaces gráficas de la aplicación.
  - **Res/menu:** Contiene los ficheros XML que definen las interfaces gráficas de los menús de la aplicación.
  - **Res/mipmap:** Contiene todos los íconos utilizados en la aplicación.
  - **Res/values:** Contiene los ficheros XML de recursos de la aplicación, como por ejemplo los literales utilizados (String.xml).

## 9.2.2 Implementación Retrofit

Tal y como se comenta en el *apartado 7.2.3* , se ha utilizado y configurado la librería Retrofit para realizar las peticiones a la API REST implementada en la aplicación servidor. Para poder utilizar dicha librería en la aplicación Android se han tenido que declarar las dependencias en el fichero *Build.gradle*:

```
implementation 'com.squareup.retrofit2:retrofit:2.3.0'  
implementation 'com.squareup.retrofit2:converter-gson:2.3.0'
```

Para gestionar la instancia de Retrofit durante el transcurso de la sesión de un usuario en la aplicación, se ha utilizado el patrón de diseño Singleton o instancia única. De esta manera se garantiza que una clase sólo tenga una instancia y proporcionamos un punto de acceso global a ella. Hablamos de la clase *InitRetrofit.java*, ubicada en el directorio *útil*, encargada de crear una instancia única inicializando el servicio de la librería a través de un método que crea una dicha instancia solo si todavía no existe alguna. Para proveer una única instancia global mediante el patrón Singleton, se consigue gracias a que:

1. La propia clase es responsable de crear la única instancia.
2. Permite el acceso global a la instancia mediante un método de clase.
3. Se declara el constructor de clase como privado para que no pueda ser instanciable directamente.

En la clase *InitRetrofit.java* se crea una instancia de la librería Retrofit y su asociación a la interface encargada de manejar las peticiones disponibles:

```
public static InitRetrofit getInstance() {  
    if (retrofit == null) {  
        retrofit = new InitRetrofit();  
    }  
    return retrofit;  
}
```

```

// Inicialización del objeto Retrofit (Patrón Singleton)
private InitRetrofit() {

    if (retrofit == null) {
        HttpLoggingInterceptor interceptor = new HttpLoggingInterceptor();
        interceptor.setLevel(HttpLoggingInterceptor.Level.BODY);

        CookieHandler cookieHandler = new CookieManager();

        OkHttpClient httpClient = new OkHttpClient.Builder()
            .addNetworkInterceptor(interceptor)
            .cookieJar(new JavaNetCookieJar(cookieHandler))
            .build();

        Gson gson = new GsonBuilder()
            .setLenient()
            .setDateFormat("dd/MM/yyyy HH:mm")
            .create();

        Retrofit retrofit = new Retrofit.Builder()
            .client(httpClient)
            .baseUrl(Global.BASE_URL)
            .addConverterFactory(GsonConverterFactory.create(gson))
            .build();
        apiRest = retrofit.create(ApiRest.class);
    }
}

```

En la última línea de código se muestra la asignación a la variable *ApiRest* del resultado del método *create()* del objeto Retrofit pasándole por parámetro la clase *ApiRest.class*.

Esta clase se trata de una interface que actúa como el manejador de las peticiones disponibles a realizar a través de la librería Retrofit, es decir, se definen y agrupan las llamadas que pueden ser realizadas en la aplicación Android. A continuación, se muestra el contenido de la definición de algunas peticiones implementadas:

```

public interface ApiRest {

    @POST("usuario/login")
    Call<UsuarioLoginRespuesta> iniciarSesion(@Body UsuarioLoginPeticion datosPeticion);

    @GET("pais")
    Call<List<Pais>> paises();

    @Multipart
    @POST("imagen/usuario")
    Call<Imagen> subirImagenUsuario(@Part MultipartBody.Part file);

    @DELETE("usuario/logout/{id}")
    Call<GenericId> logout(@Path("id") Long id);
}

```

Una vez tenemos configurada, definida e inicializada la instancia de Retrofit y su manejador de peticiones, ya podemos utilizar dicha instancia en cualquier parte del código de la aplicación para realizar peticiones síncronas o asíncronas a la API REST del servidor. A continuación, se muestra un ejemplo de cómo utilizar Retrofit para realizar una petición y procesar la respuesta obtenida:

```
// Inicializamos el servicio de APIRest de retrofit
apiRest = InitRetrofit.getInstance().getApiRest();
```

En primer lugar, obtenemos la instancia del manejador de peticiones para poder realizar las peticiones.

```
private void obtenerUbicacion() {
    Call<List<Pais>> peticionRestPaises = apiRest.paises();
    peticionRestPaises.enqueue(new Callback<List<Pais>>() {
        @Override
        public void onResponse(Call<List<Pais>> call, Response<List<Pais>> response) {
            if (response.raw().code() != Global.CODE_ERROR_RESPONSE_SERVER && response.isSuccessful()) {
                paises = response.body();
                paisActual = Global.DEFAULT_COUNTRY;
                obtenerProvincias();
            } else {
                try {
                    JSONObject jsonObjError = new JSONObject(Objects.requireNonNull(response.errorBody()).string());
                    SnackbarUtil.showSnackBar(findViewById(R.id.login_snackbar), jsonObjError.getString("message"),
                } catch (Exception e) {
                    Log.e("ERROR:", e.getMessage());
                }
            }
        }

        @Override
        public void onFailure(Call<List<Pais>> call, Throwable t) {
            Log.e("ERROR:", t.getMessage());
        }
    });
}
```

### 9.2.3 Listar eventos deportivos

Una de las funcionalidades principales de la aplicación es la de listar en la actividad principal los eventos resultantes a una búsqueda personalizada realizada por un usuario. Para ello, se ha implementado un *RecyclerView* [27] junto con *Cardviews* [28], es decir, una lista contenedora de tarjetas estilizadas en donde se proyectarán los eventos deportivos en una interfaz novedosa. Un *RecyclerView* es un contenedor de elementos en forma de lista como la clase *ListView*. Aunque tengan la misma funcionalidad, se ha decidido utilizar *RecyclerView* ya que permiten “reciclar” los ítems que ya no son visibles por el usuario debido al scrolling, proporcionando una mejora en el rendimiento a la hora de listar los elementos.

Para crear y administrar un *RecyclerView* se necesita de una fuente de datos que provea la información lógica de cada elemento y un adaptador que los lea, interprete e incruste. A diferencia de las *ListView* es necesario el elemento *LayoutManager*, encargado de añadir y reusar las vistas en el *RecyclerView*. Su función es calcular las posiciones fuera del foco del usuario, es decir no son visibles en pantalla, y así reemplazar el contenido de un elemento fuera de foco por el contenido de otro. Como se ha mencionado anteriormente, este hecho reduce los tiempos de ejecución.

Una *Cardview* es un contenedor capaz de presentar su aspecto con sombras y bordes redondeados. Su finalidad es la de ofrecer un elemento visual en forma de tarjetas de información. En la figura 9.2.3.1 se puede ver el diseño utilizado para las tarjetas de los eventos deportivos a listar.

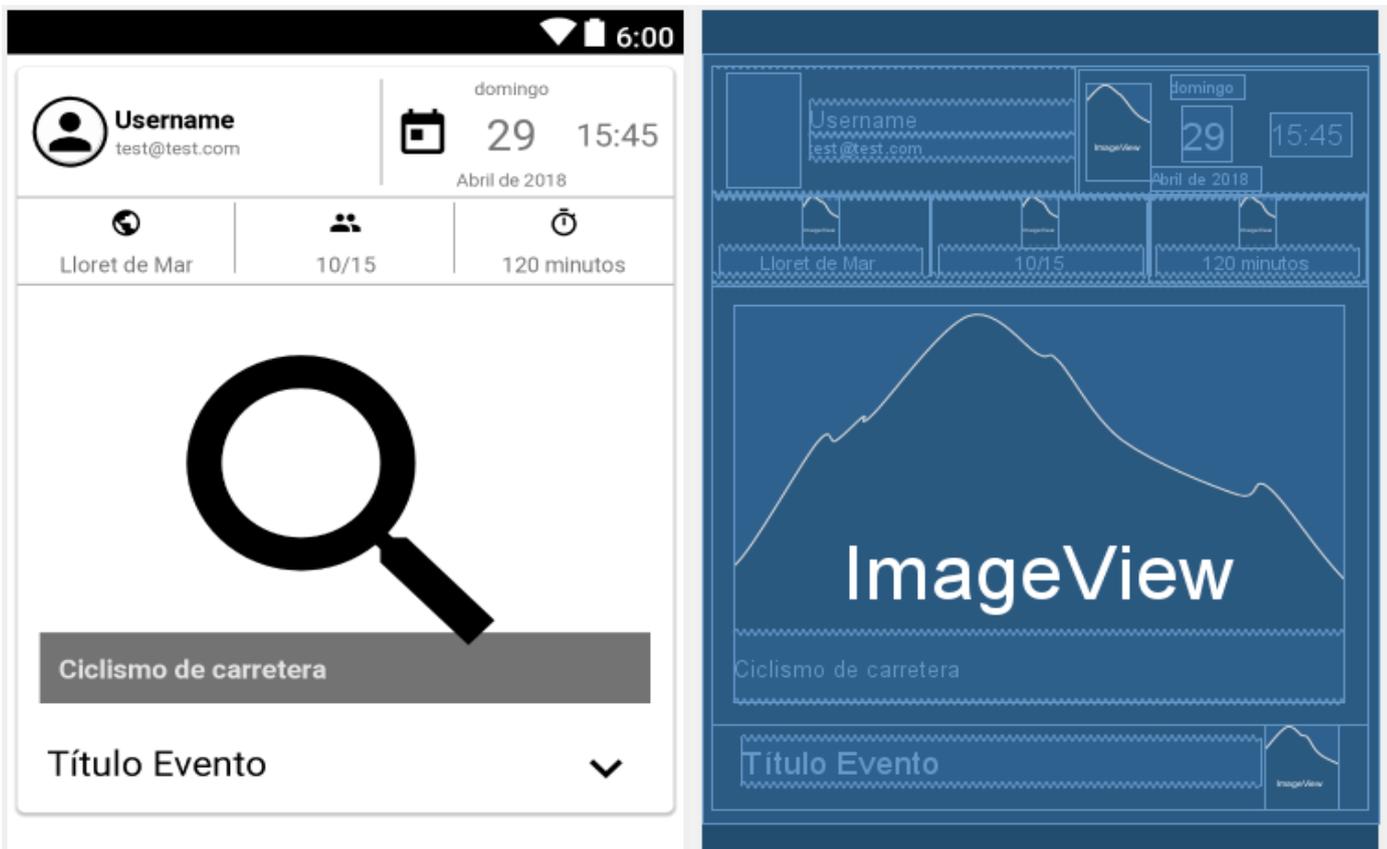


Figura 9.2.3.1: Tarjeta evento deportivo

En cuanto a la implementación, en primer lugar, definimos las dependencias en el fichero Build.gradle de las estructuras a utilizar:

```
implementation 'com.android.support:cardview-v7:27.1.1'  
implementation 'com.android.support:recyclerview-v7:27.1.1'
```

A continuación, ubicamos en el layout de la actividad Principal.java un elemento del tipo <Android.support.v7.widget.RecyclerView> para definir que se utilizará este tipo de listado en la interfaz de usuario.

```
<android.support.v4.widget.SwipeRefreshLayout  
    android:id="@+id/swipeLayout"  
    android:layout_width="match_parent"  
    android:layout_height="0dp"  
    app:layout_constraintBottom_toBottomOf="parent"  
    app:layout_constraintEnd_toEndOf="parent"  
    app:layout_constraintStart_toStartOf="parent"  
    app:layout_constraintTop_toBottomOf="@+id/appbar">  
  
    <android.support.v7.widget.RecyclerView xmlns:android="http://schemas.android.com/apk/res/android"  
        android:id="@+id/reciclador"  
        android:layout_width="match_parent"  
        android:layout_height="0dp"  
        android:padding="3dp"  
        android:scrollbars="vertical"  
        app:layout_constraintBottom_toBottomOf="parent"  
        app:layout_constraintEnd_toEndOf="parent"  
        app:layout_constraintStart_toStartOf="parent"  
        app:layout_constraintTop_toBottomOf="@+id/appbar">  
  
    </android.support.v7.widget.RecyclerView>  
</android.support.v4.widget.SwipeRefreshLayout>
```

En la implementación hemos incluido el *RecyclerView* en un elemento de tipo *SwipeRefreshLayout* [29] el cual nos proporciona un mecanismo para actualizar el contenido de un listado deslizándolo en su conjunto hacia abajo. De esta forma se regenera el listado ya presente realizando de nuevo la petición y actualizando el contenido de las tarjetas en caso de que se hayan producido cambios.

En cuanto a la fuente de datos, se ha utilizado la clase *Evento.java* ubicada en el directorio *Entity*. Esta clase contiene los atributos a mostrar en las tarjetas listadas y obtenidos en la respuesta del servidor a una petición de búsqueda de eventos deportivos. La clase contiene los siguientes atributos:

```

public class Evento implements Serializable {

    private Long id;

    private String titulo;

    private String descripcion;

    private int duracion;

    private int numeroParticipantes;

    private Date fechaEvento;

    private Deporte deporte;

    private Estado estado;

    private int participantesRegistrados;

    private Administrador administrador;

    private Municipio municipio;

    private Provincia provincia;

    private Foro foro;

    private Ubicacion ubicacion;
}

```

Los objetos de la clase `Evento.java` servirán de alimento para el adaptador. Para crear el adaptador, se realiza a través de la clase `RecyclerView.Adapter`. Los adaptadores para `RecyclerView` deben contener una clase interna que extienda de `RecyclerView.ViewHolder`. Un `ViewHolder` es un objeto que representa un elemento de la lista, el cual guarda las referencias a los `Views` dentro del layout para agilizar el acceso. Este objeto actúa como intermediario entre el `LayoutManager` y el adaptador, actuando como una especie de caché. En la implementación llevada a cabo, el adaptador y el objeto `ViewHolder` se definen en la clase `EventoAdapter.java` dentro del directorio `adapter`. Como vemos la clase hereda de `RecyclerView.Adapter`:

```

public class EventoAdapter extends RecyclerView.Adapter<EventoAdapter.EventoViewHolder> {

```

Dentro de la clase definimos la clase interna que hereda de `RecyclerView.ViewHolder`, en la cual definimos las referencias a los elementos del layout:

```

public static class EventoViewHoler extends RecyclerView.ViewHolder {
    CardView cardView;
    TextView usernameAdmin;
    TextView emailAdmin;
    ImageView imagenAdmin;
    ImageView imagenEvento;
    TextView deporteEvento;
    TextView participantesEvento;
    TextView municipioEvento;
    TextView duracionEvento;
    TextView tituloEvento;
    TextView descripcionEvento;
    ConstraintLayout layoutTitulo;
    ConstraintLayout layoutDescripcion;
    ImageView imagenTitulo;
    TextView diaMesEvento;
    TextView horaEvento;
    TextView mesEvento;
    TextView diaSemanaEvento;

    EventoViewHoler(View itemView) {
        super(itemView);
        cardView = itemView.findViewById(R.id.cardview);
        usernameAdmin = itemView.findViewById(R.id.cardview_username_administrador);
        emailAdmin = itemView.findViewById(R.id.cardview_email_administrador);
        imagenAdmin = itemView.findViewById(R.id.cardview_imagen_administrador);
        imagenEvento = itemView.findViewById(R.id.cardview_imagen_evento);
        deporteEvento = itemView.findViewById(R.id.cardview_deporte_evento);
        participantesEvento = itemView.findViewById(R.id.cardview_participantes_evento);
        municipioEvento = itemView.findViewById(R.id.cardview_evento_municipio);
        duracionEvento = itemView.findViewById(R.id.cardview_duracion_evento);
        tituloEvento = itemView.findViewById(R.id.cardview_titulo_evento);
        layoutTitulo = itemView.findViewById(R.id.constraintLayout14);
        layoutDescripcion = itemView.findViewById(R.id.constraintLayout15);
        imagenTitulo = itemView.findViewById(R.id.cardview_icono_descripcion);
        descripcionEvento = itemView.findViewById(R.id.cardview_descripcion_evento);
        diaMesEvento = itemView.findViewById(R.id.cardview_dia_mes_evento);
        horaEvento = itemView.findViewById(R.id.cardview_hora_evento);
        mesEvento = itemView.findViewById(R.id.cardview_mes_evento);
        diaSemanaEvento = itemView.findViewById(R.id.cardview_string_dia);
    }
}

```

La clase `EventoAdapter.java` extiende de `RecyclerView.Adapter` `<EventoAdapter.EventoViewHolder>`. Con esta declaración el `LayoutManager` ya sabrá a que tipo acudir para reducir el acceso por referencias.

Mediante el método `onCreateViewHolder()` se infla el contenido de los elementos a listar en la interfaz de las tarjetas. En el siguiente ejemplo se muestra como mediante el método `inflate` asignamos el layout `R.id.cardview_evento`.

```

@NonNull
@Override
public EventoAdapter.EventoViewHolder onCreateViewHolder(@NonNull ViewGroup viewGroup, int viewType) {
    View v = LayoutInflater.from(viewGroup.getContext()).inflate(R.layout.cardview_evento, viewGroup, attachToRoot: false);
    return new EventoViewHolder(v);
}

```

El método *onBindViewHolder ()* se encarga de realizar las modificaciones del contenido de cada tarjeta. Se recibe por parámetro el *ViewHolder* actual y su posición en la fuente de datos.

```

@Override
public void onBindViewHolder(@NonNull final EventoAdapter.EventoViewHolder holder, @SuppressWarnings("RecyclerView") final int position) {

    final Evento eventoActual = eventos.get(position);

    holder.usernameAdmin.setText(eventoActual.getAdministrador().getUsername());
    holder.emailAdmin.setText(eventoActual.getAdministrador().getEmail());
    holder.deporteEvento.setText(eventoActual.getDeporte().getDeporte());
}

```

Para crear la relación de las diferentes clases implementadas en la clase *Principal.java* declaramos las instancias de *RecyclerView*, *LayoutManager* y el adaptador de eventos:

```

// Lista de eventos
private RecyclerView recycler;
// Linear layout manager del recyclerView
private LinearLayoutManager lManager;
// Adaptador de eventos
private EventoAdapter adapter;

```

Finalmente, se obtiene la instancia de *RecyclerView*, se crea el *LayoutManager* y se crea el adaptador con la lista de evento deportivos a listar:

```

// Obtener el Recycler
recycler = findViewById(R.id.reciclador);
lManager = new LinearLayoutManager(context: this);
recycler.setLayoutManager(lManager);
// creamos adaptador con el evento click
adapter = new EventoAdapter(getApplicationContext(), eventos, new EventoAdapter.OnItemClickListener() {
    @Override
    public void onItemClick(Evento e, int position) {
        Intent i = new Intent(getApplicationContext(), EventoDetalle.class);
        i.putExtra(Global.KEY_SELECTED_EVENT, e);
        i.putExtra(Global.KEY_SELECTED_EVENT_IS_ADMIN, e.getAdministrador().getId().equals(UsuarioActual.getInstance().getId()));
        i.putExtra(Global.KEY_SELECTED_EVENT_POSITION, position);
        startActivityForResult(i, Global.REQUEST_CODE_EVENTO_DETALLE);
    }
});
recycler.setAdapter(adapter);

```

El *LayoutManager* fue instanciado con la subclase *LinearLayoutManager* indicando que el reciclador mostrará los elementos de forma vertical.

A la hora de crear el adaptador, se ha definido el evento *OnItemClickListener ()* que nos permite acceder al detalle de los eventos deportivos listados.

## 9.2.4 Implementación del foro

Como se ha mencionado en el *capítulo 7.2.6* para la creación y manejo de los mensajes del foro disponible en los eventos deportivos se ha utilizado el servicio Firebase Database Realtime de Google. En el momento en que un usuario crea un evento deportivo, la aplicación internamente crea una tabla en esta base de datos NoSQL alojada en la red de la siguiente forma:

```
// Creamos el foro en la base de datos de FireBase
if (idEvento != null) {
    DatabaseReference root = FirebaseDatabase.getInstance().getReference().getRoot();
    Map<String, Object> map = new HashMap<>();
    map.put(Global.PREFIJO_SALA_FORO_EVENTO + idEvento.getId().toString(), "");
    root.updateChildren(map);
}
```

Se obtiene una instancia de la base de datos de Firebase y creamos una nueva tabla utilizando un prefijo concatenado con el identificador del evento en la base de datos local. Cuando se accede al detalle de un evento y se selecciona la pestaña foro, la aplicación recupera una referencia al contenido de la tabla pasada por parámetro:

```
// Nombre de la sala
nombreSala = Global.PREFIJO_SALA_FORO_EVENTO + evento.getId().toString();
// Accedemos a la sala
salaForo = FirebaseDatabase.getInstance().getReference().child(nombreSala);
```

Cuando se introduce y envía un nuevo mensaje en el input del foro, se construye un mapa en donde se definen los campos a guardar de cada mensaje enviado. En concreto, se almacenará el nombre del usuario que realiza la petición, el contenido del mensaje, el identificador del usuario y la fecha en que se ha enviado. A continuación, se muestra la implementación del método encargado de enviar los mensajes a la base de datos de Firebase:

```

public void sendMessage(View view, String message) {
    if (message.length() > 0) {
        // Al hacer click botón enviar mensaje
        Map<String, Object> mapa = new HashMap<>();
        String tempKey = salaForo.push().getKey();
        salaForo.updateChildren(mapa);

        DatabaseReference mensajeSala = salaForo.child(tempKey);
        Calendar fecha = Calendar.getInstance();
        Map<String, Object> mapa2 = new HashMap<>();
        mapa2.put("name", username);
        mapa2.put("msg", message);
        mapa2.put("id", UsuarioActual.getInstance().getId().toString());
        mapa2.put("date", fecha.get(Calendar.HOUR_OF_DAY) + ":" + fecha.get(Calendar.MINUTE));
        mensajeSala.updateChildren(mapa2);

        Objects.requireNonNull(tilMensaje.getEditText()).setText("");
    }
    Objects.requireNonNull(tilMensaje.getEditText()).setText("");
}
}

```

Al objeto *DatabaseReference* se le asigna un evento que se dispara en el momento de enviar un nuevo mensaje o cuando se carga la pestaña en un primer momento. Su finalidad es la de presentar el contenido de los mensajes del foro almacenados en la tabla en la interfaz de pantalla. Se recorren todos los elementos mediante un iterador y se muestran en un *ListView* mediante un adaptador. La implementación del método es la siguiente:

```

childEventListener = salaForo.addChildEventListener(new ChildEventListener() {
    @Override
    public void onChildAdded(DataSnapshot dataSnapshot, String s) {

        Iterator i = dataSnapshot.getChildren().iterator();

        while (i.hasNext()) {
            String fecha = Objects.requireNonNull(((DataSnapshot) i.next()).getValue()).toString();
            Long idUsuario = Long.parseLong(Objects.requireNonNull(((DataSnapshot) i.next()).getValue()).toString());
            String mensajeUsuario = Objects.requireNonNull(((DataSnapshot) i.next()).getValue()).toString();
            String nombreUsuario = Objects.requireNonNull(((DataSnapshot) i.next()).getValue()).toString();

            DatosUsuarioForo datosUsuarioForo = new DatosUsuarioForo(nombreUsuario, idUsuario, fecha);
            boolean esUsuarioActual = nombreUsuario.equals(username);
            final MensajeForo mensajeForo = new MensajeForo(mensajeUsuario, datosUsuarioForo, esUsuarioActual);
            getActivity().runOnUiThread(new Runnable() {
                @Override
                public void run() {
                    mensajeAdapter.add(mensajeForo);
                    listaMensajes.setSelection(listaMensajes.getCount() - 1);
                }
            });
        }
    }
});
}
}

```

## 9.2.5 Manejo notificaciones

Como se ha mencionado en el *capítulo 7.2.7*, para gestión el envío y recibo de notificaciones push en dispositivos móviles se ha utilizado el servicio Firebase Cloud Messaging (FCM) de Firebase. Para utilizar este servicio se ha declarado la dependencia en el fichero Build.gradle:

```
implementation 'com.google.firebase:firebase-messaging:12.0.1'
```

Para identificar los distintos dispositivos móviles se realiza mediante un token asignado por el servidor de Firebase. Para ello se ha creado un servicio que extiende de *FirebaseInstanceIdService* que administra la creación, la rotación y actualización de los tokens de registro. Se tiene que definir el servicio en el fichero AndroidManifest.xml:

```
<service android:name=".MyFirebaseInstanceIdService"
  tools:ignore="ExportedService">
  <intent-filter>
    <action android:name="com.google.firebase.INSTANCE_ID_EVENT" />
  </intent-filter>
</service>
```

```
public class MyFirebaseInstanceIdService extends FirebaseInstanceIdService {

  private static final String LOGTAG = "FIREBASE-TOKEN";

  @Override
  public void onTokenRefresh() {

    String refreshedToken = FirebaseInstanceId.getInstance().getToken();
    Log.d(LOGTAG, "msg: Token actualizado: " + refreshedToken);

  }

}
```

El token de registro se almacena en la base de datos local en la tabla Usuario y se obtiene en el momento de iniciar sesión o registrarse en la aplicación mediante la siguiente sentencia:

```
String tokenFireBase = FirebaseInstanceId.getInstance().getToken();
```

Por otro lado, hay que controlar la recepción y tratamiento de las notificaciones recibidas. Para ello hay que crear un servicio que extienda de la clase *FirebaseMessagingService*. Este servicio debe estar declarado en el fichero AndroidManifest.xml:

```

<service android:name=".MyFirebaseMessagingService"
  tools:ignore="ExportedService">
  <intent-filter>
    <action android:name="com.google.firebase.MESSAGING_EVENT" />
  </intent-filter>
</service>

```

En la clase `MyFirebaseMessagingService.java` se implementa la lógica que intercepta las notificaciones recibidas, determina el tipo de notificación, obtiene el contenido y asigna la lógica de la notificación en el momento que se desea acceder a ellas. Para ello se sobrescribe el método `onMessageReceived()`:

```

@Override
public void onMessageReceived(RemoteMessage remoteMessage) {
    // Inicializamos el servicio de API Rest de retrofit
    apiRest = InitRetrofit.getInstance().getApiRest();

    Map<String, String> data = remoteMessage.getData();
    String notificationType = data.get("notificationType");
}

```

## 9.2.6 Ubicación GPS

En el momento en que un usuario inicia sesión, la aplicación intenta obtener la ubicación GPS del dispositivo móvil. El motivo de obtener la ubicación es la de poder realizar búsquedas de eventos deportivos en función a una distancia máxima indicada respecto a la ubicación actual. Para ello, los dispositivos Android disponen de dos maneras para localizar un dispositivo. Por un lado, es aprovechar el sensor de posicionamiento GPS (*Global Positioning System*) del dispositivo. Otra forma de obtener la localización es mediante la red, ya que el proveedor de red puede obtener la posición aproximada basándose en las antenas de telefonía móvil y Wifi. Como la localización del dispositivo es considerada información sensible, es necesario declarar los permisos en el archivo `AndroidManifest.xml` y solicitar en tiempo de ejecución los permisos al usuario.

```

<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />

```

```

// Pedir permisos para poder utilizar el GPS del dispositivo
private void obtenerUbicacionGPSActual() {
    // Pedir permisos para utilizar ubicación del dispositivo
    if (ActivityCompat.checkSelfPermission( context: this, Manifest.permission.ACCESS_FINE_LOCATION) != PackageManager.PERMISSION_GRANTED &&
        ActivityCompat.checkSelfPermission( context: this, Manifest.permission.ACCESS_COARSE_LOCATION) != PackageManager.PERMISSION_GRANTED) {
        // Si no hay permisos concedidos se piden:
        ActivityCompat.requestPermissions( activity: this, new String[]{Manifest.permission.ACCESS_FINE_LOCATION,}, Global.REQUEST_CODE_GPS_LOCATION);
    } else {
        // Si la aplicación está autorizada a la detección de localización:
        determinarUbicacion();
    }
}
}

```

En caso de que se autorice la obtención de la localización del dispositivo se ejecuta el método *determinarUbicacion* (). Mediante la clase *LocationManager* [30] de Android podemos obtener la ubicación del dispositivo ya sea por red o por el sensor GPS del dispositivo:

```
LocationManager mlocManager = (LocationManager) getSystemService(Context.LOCATION_SERVICE);
// Pedir actualizaciones de posición (cada minuto y notificar si distancia es > 1km respecto a la anterior)
if (mlocManager != null) {
    mlocManager.requestLocationUpdates(LocationManager.NETWORK_PROVIDER, Global.MINIMUM_TIME_UPDATE_LOCATION, Global.MINIMUM_DISTANCE_UPDATE_LOCATION, Local);
    mlocManager.requestLocationUpdates(LocationManager.GPS_PROVIDER, Global.MINIMUM_TIME_UPDATE_LOCATION, Global.MINIMUM_DISTANCE_UPDATE_LOCATION, Local);
}
```

Se ha definido que se realice una petición de localización cada minuto y solamente se modifique un cambio en la ubicación si la distancia actual es superior en 1 kilómetro a la última distancia registrada. En caso de que se notifique una nueva ubicación, esta se registra en la base de datos:

```
// Se recibe una nueva actualización de posicionamiento GPS
public void setLocation(Location localizacion) {
    if (localizacion != null && localizacion.getLatitude() != 0.0 && localizacion.getLongitude() != 0.0) {
        try {
            Geocoder geocoder = new Geocoder(context, Locale.getDefault());
            // Obtenemos una lista con las direcciones aproximadas. Nos quedamos solo la primera
            List<Address> list = geocoder.getFromLocation(
                localizacion.getLatitude(), localizacion.getLongitude(), maxResults: 1);
            if (!list.isEmpty()) {
                Address datosUbicacion = list.get(0);
                Ubicacion ubicacionActual = new Ubicacion(datosUbicacion.getLatitude(), datosUbicacion.getLongitude(),
                    datosUbicacion.getAddressLine(index: 0), datosUbicacion.getLocality());
                Call<GenericId> petitionRest = apiRest.guardarUbicacionActualUsuario(ubicacionActual);
                petitionRest.enqueue(new Callback<GenericId>() {
```

## 9.3 Pruebas

En cuanto a la implementación de aplicaciones software hay que prevenir que estas lleguen a manos de usuarios finales con errores. Por lo tanto, durante la etapa de desarrollo es imprescindible testear y definir pruebas de las diferentes funcionalidades. En este caso, acorde con la metodología utilizada, ver *capítulo 3*, se han realizado pruebas en tres circunstancias diferentes:

1. Testeo de la parte desarrollada.
2. Testeo de la unión de diferentes partes.
3. Testeo con diferentes modelos de ejemplos de la unión de las partes.

De esta forma se pretende minimizar la aparición de errores de funcionamiento de la aplicación. Lo ideal sería que las pruebas las realizaran personas diferentes a las que han programado el código como por ejemplo verificadores o analistas. En este caso este rol ha sido llevado a cabo por la misma persona, yo mismo.

En cuanto al testeo de una parte desarrollada nos centramos en la implementación del programa para escoger los casos de prueba. Lo ideal es testear todos los casos de prueba que recorran todos los caminos posibles en el flujo de control de la parte desarrollada. Técnicamente hablando, se testea que sucede en la parte al forzarla a pasar por cualquier condición o bucle.

Una vez integradas diferentes partes, hay que considerar que no es suficiente con validar las partes individualmente para asegurar el buen funcionamiento del programa. Por lo tanto, a medida que se van acoplando las partes entre si se testea de nuevo el funcionamiento del programa para asegurar que las partes que fueron testeadas individualmente continúan funcionando correctamente al integrarse.

Finalmente, una vez integrado todo el sistema y testeadas todas las partes de forma individual, y a medida que se iban acoplando entre sí, se realizan pruebas de la aplicación mediante modelos de ejemplos. En caso de detectar errores en cualquiera de las 3 fases de pruebas, una vez identificada la parte errónea, se procede a su corrección y posterior testeo siguiendo el flujo explicado.

---

## 10. Implantación y resultados

---

En este capítulo se muestra el grado de consecución de los objetivos mediante ejemplos del funcionamiento de la aplicación. Además, se expondrá la validez legal de la aplicación, especialmente la que hace referencia a la Ley Orgánica de Protección de Datos (LOPD).

### 10.1 Resultados en función de los objetivos

Para mostrar los resultados obtenidos en función de los objetivos, se irá recorriendo las diferentes pantallas de la aplicación mostrando capturas de pantalla del comportamiento de la aplicación en función de la interacción llevada a cabo con la interfaz de usuario.

Una vez la aplicación se encuentra instalada en nuestro dispositivo móvil se muestra el icono y nombre de la misma mediante la cual ejecutar la aplicación. En la figura 10.1.1 se muestra el icono de acceso a la aplicación.

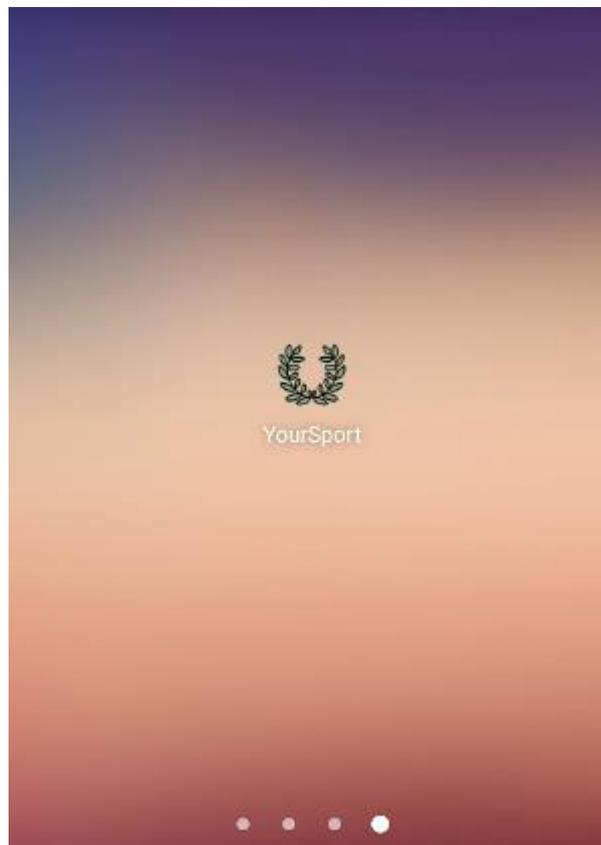


Figura 10.1.1: Icono de la aplicación

Como vemos en la figura 10.1.2, al acceder a la aplicación, se muestra el formulario en el cual se permite iniciar sesión o navegar hacia el formulario de alta.

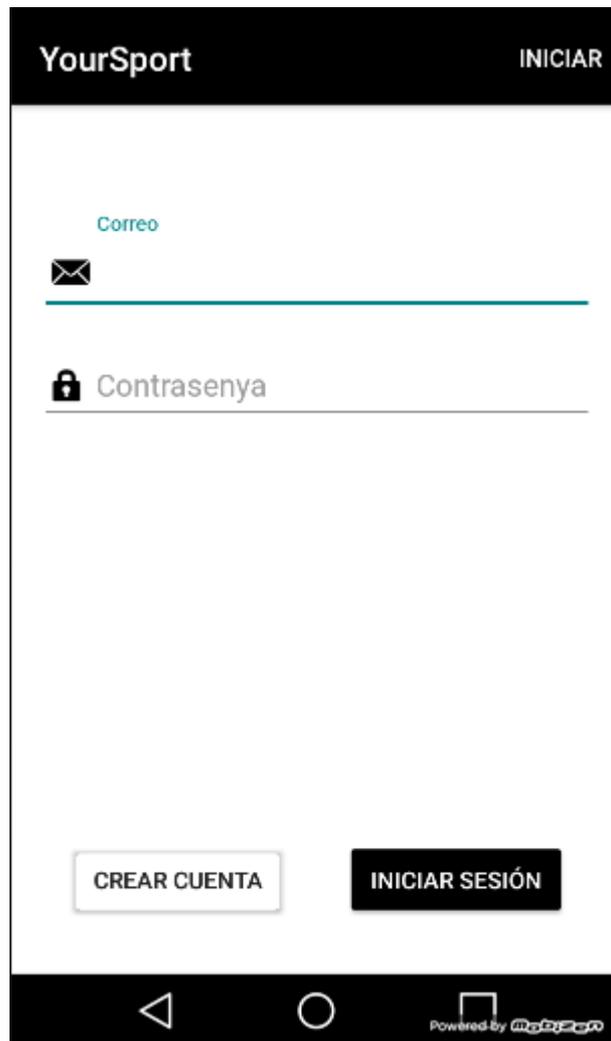


Figura 10.1.2: Interfaz inicio sesión

En la figura 10.1.3, se muestra el comportamiento del formulario si se trata de iniciar sesión dejando uno o ambos campos del formulario de acceso en blanco.

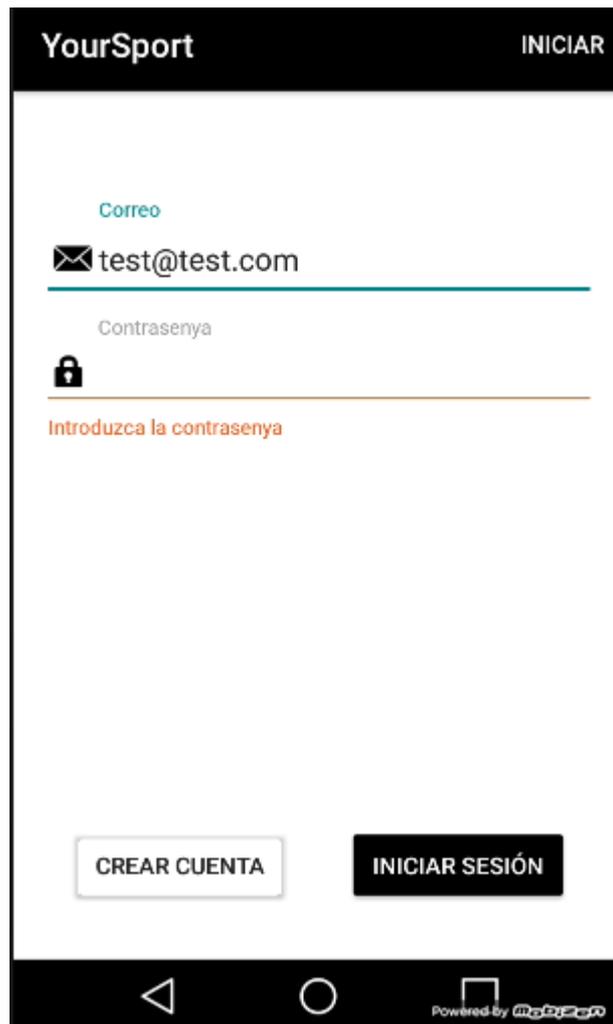


Figura 10.1.3: Interfaz inicio sesión campo del formulario en blanco

En la figura 10.1.4, se muestra el comportamiento del formulario si se trata de iniciar sesión indicando una dirección de correo electrónica o contraseñas erróneas o no dadas de alta en la aplicación.

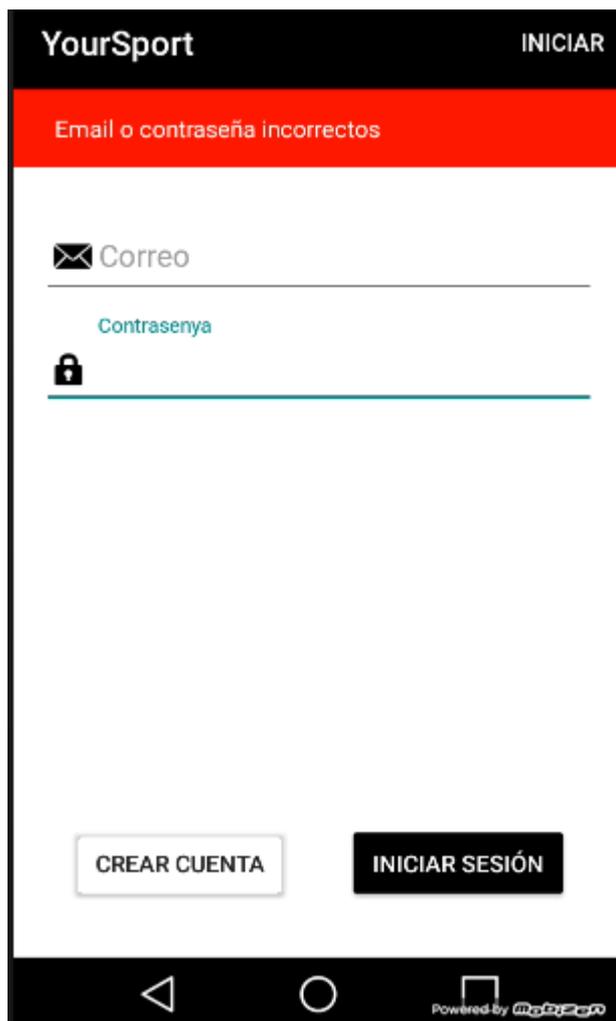


Figura 10.1.4: Interfaz inicio sesión credenciales erróneas

En caso de seleccionar el botón crear cuenta, la aplicación nos redirige al formulario de registro para darnos de alta en el sistema. La aplicación asigna a todos los usuarios una imagen por defecto, la cual puede ser modificada seleccionando una de la galería del dispositivo o de la cámara. Por otro lado, se cargan las provincias, municipios y categorías deportivas disponibles en base de datos.

En la figura 10.1.5, se muestra la interfaz gráfica del formulario de registro, la cual consiste en una vista scrollable excepto la imagen que se mantiene siempre fija.

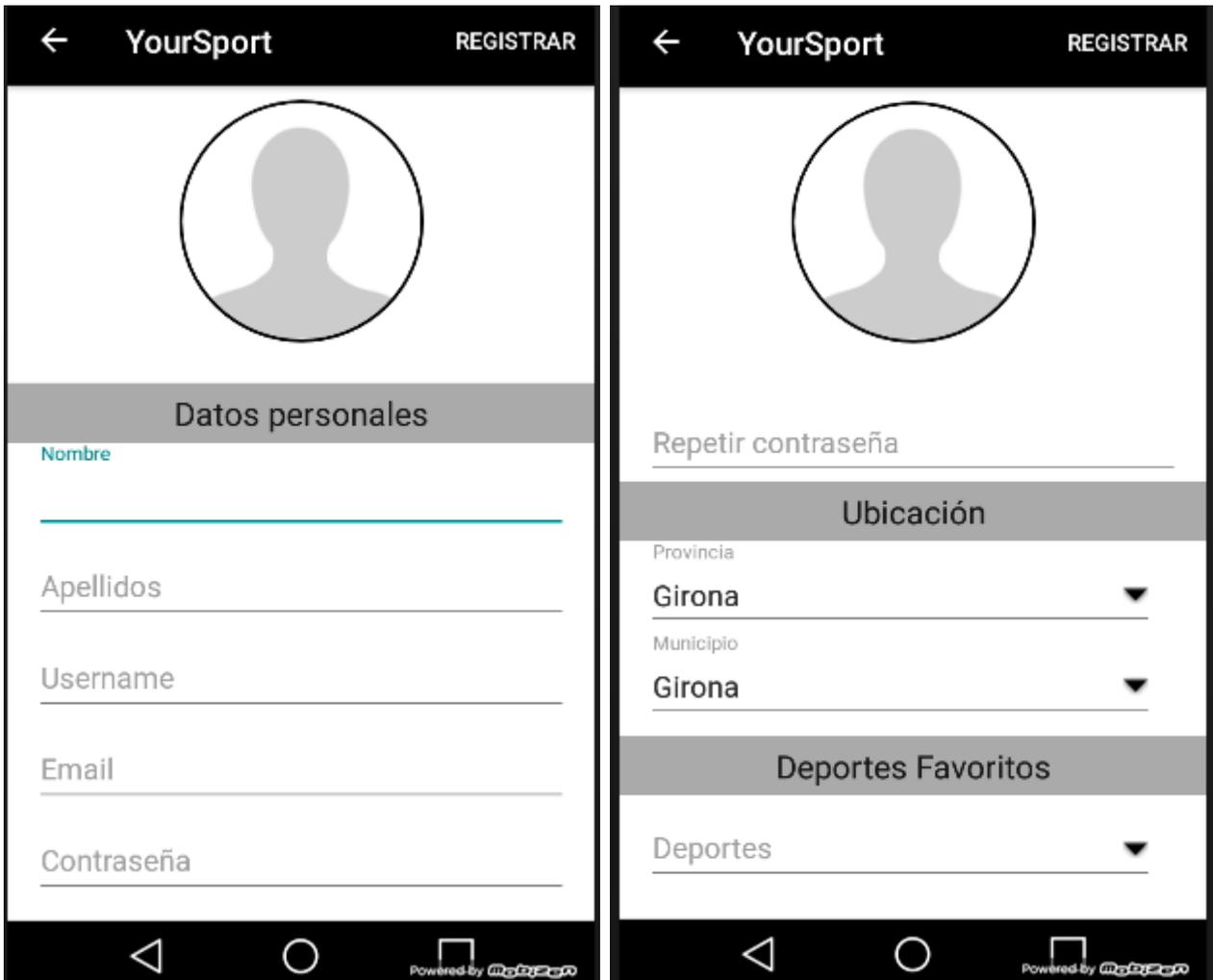


Figura 10.1.5: Interfaz inicio sesión credenciales erróneas

Si se pulsa sobre la imagen por defecto, se nos aparece un menú en donde podremos escoger de donde deseamos seleccionar una nueva imagen tal y como muestra la figura 10.1.6.

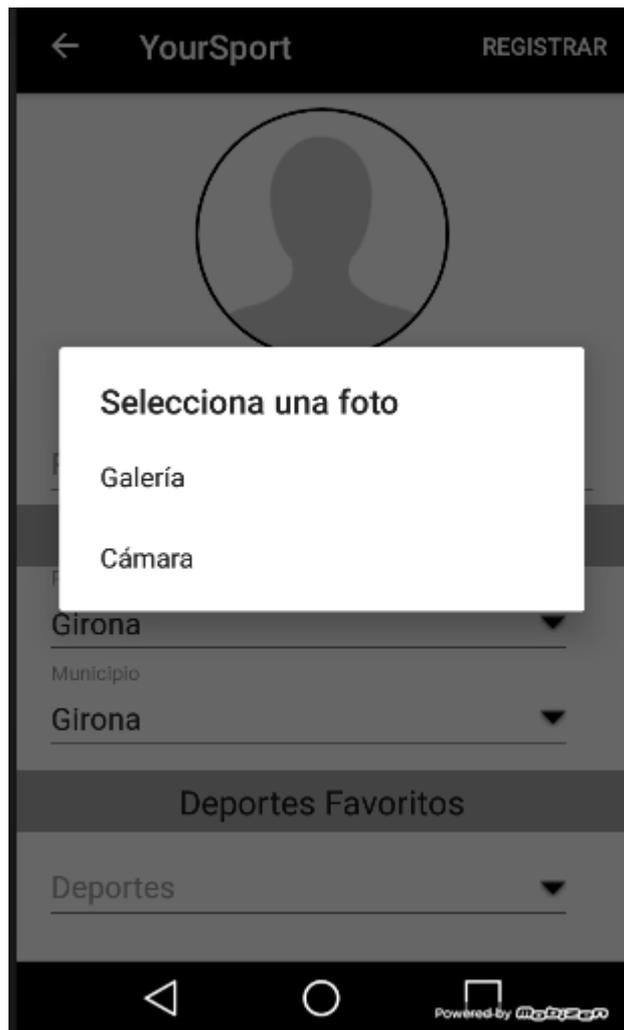


Figura 10.1.6: Interfaz menú seleccionar imagen

Al seleccionar una de las opciones la aplicación la primera vez que la ejecutemos nos pedirá permisos para acceder a la cámara o al sistema de ficheros del dispositivo móvil. Ver figura 10.1.7.

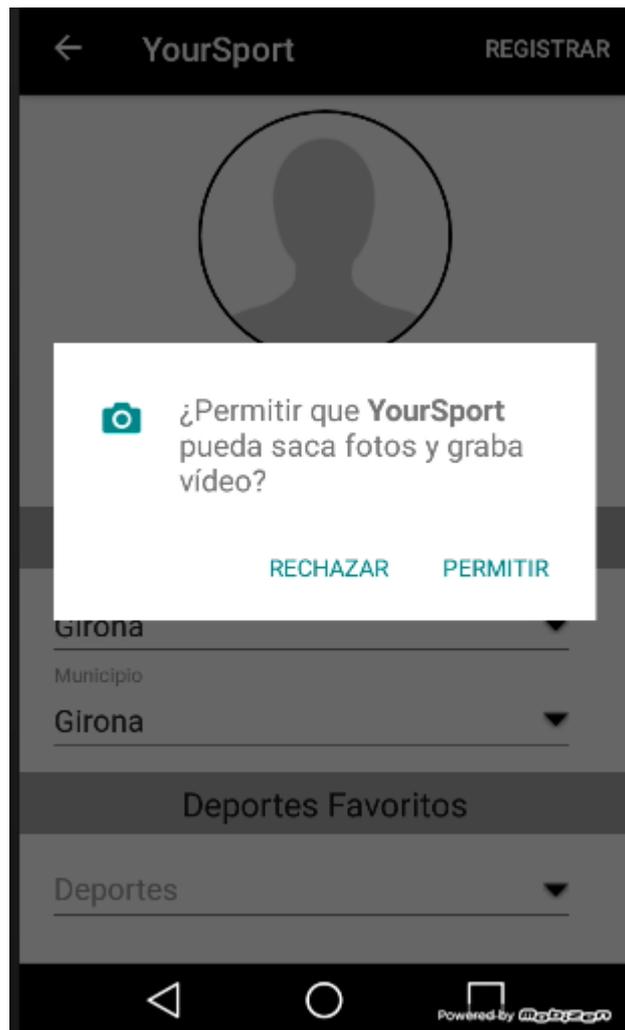


Figura 10.1.7: Interfaz permisos

En el momento en que se selecciona una imagen nueva, la aplicación consta de una librería que permite modificar la imagen antes de subirla al servidor. Como podemos observar en la figura 10.1.8 nos permite recortar la imagen entre otras opciones. Posteriormente, la imagen se visualizará en el formulario de registro.

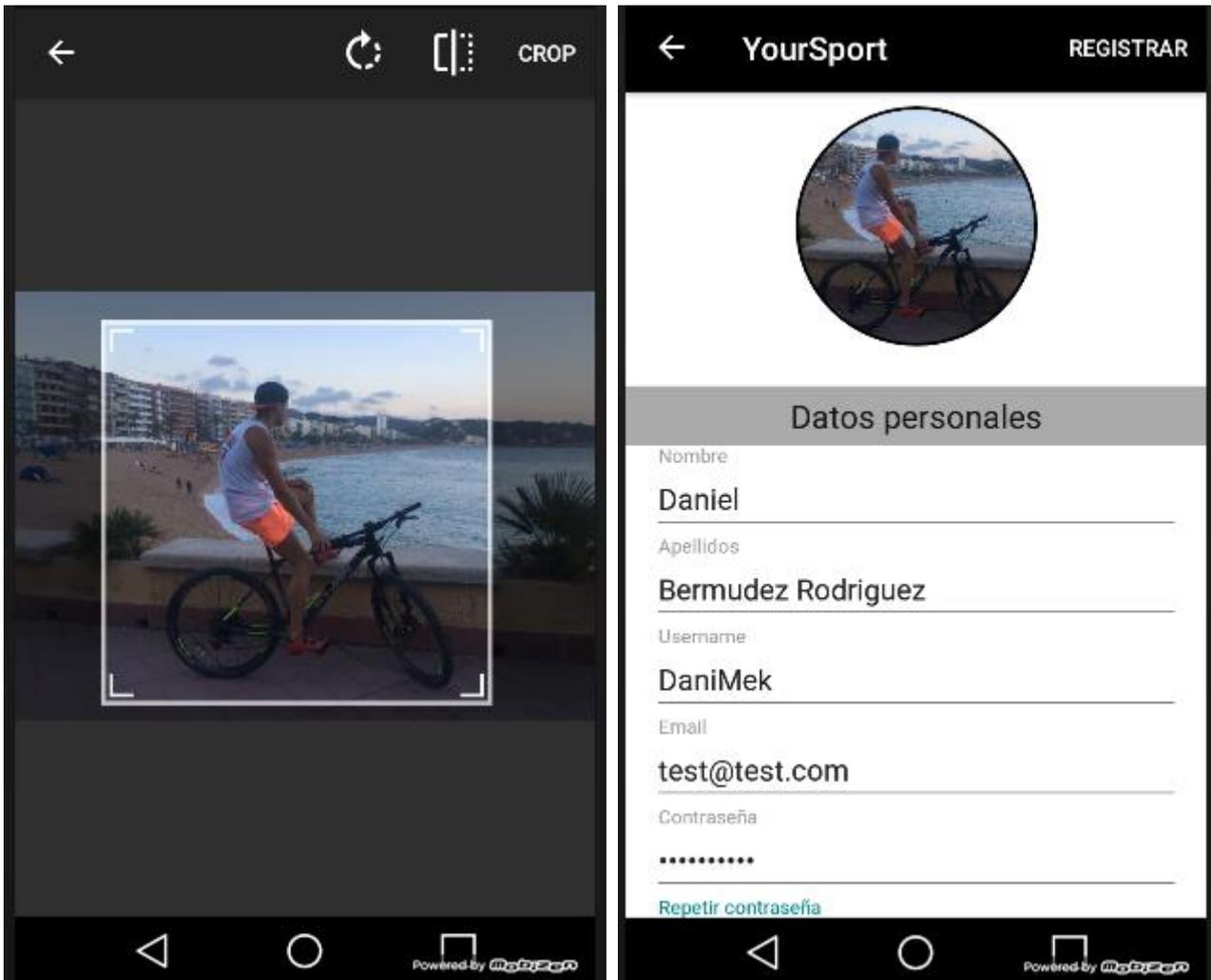


Figura 10.1.8: Interfaz recortar imagen

Para seleccionar nuestro lugar de residencia, la aplicación nos permite seleccionar de un desplegable las provincias y municipios de España. Los municipios se actualizan en función de la provincia seleccionada. Ver figura 10.1.9.

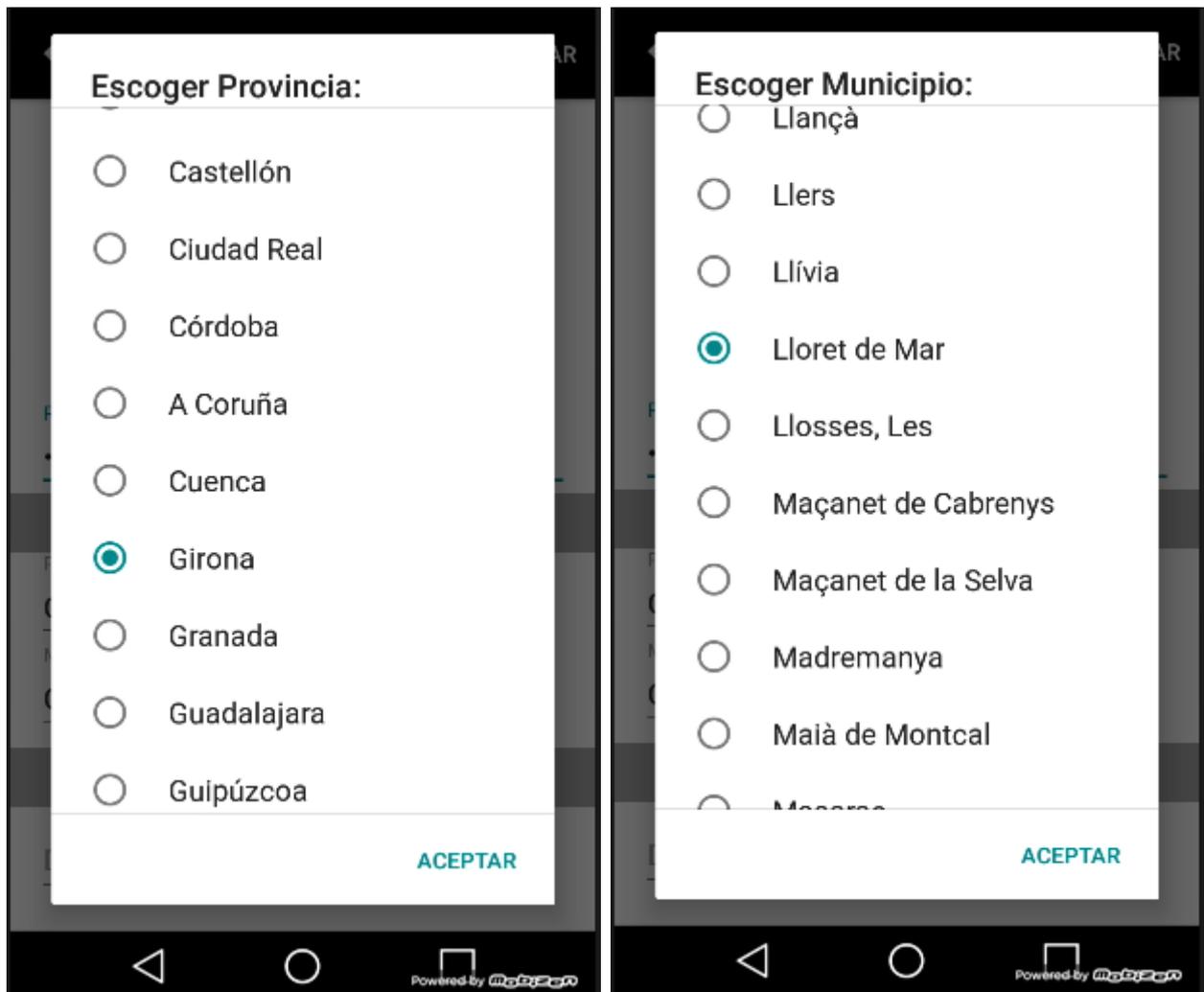


Figura 10.1.9: Interfaz provincias y municipios

En cuanto a las categorías deportivas disponibles la aplicación también muestra un desplegable con las diferentes opciones. En este caso podemos seleccionar más de una opción. Ver figura 10.1.10.

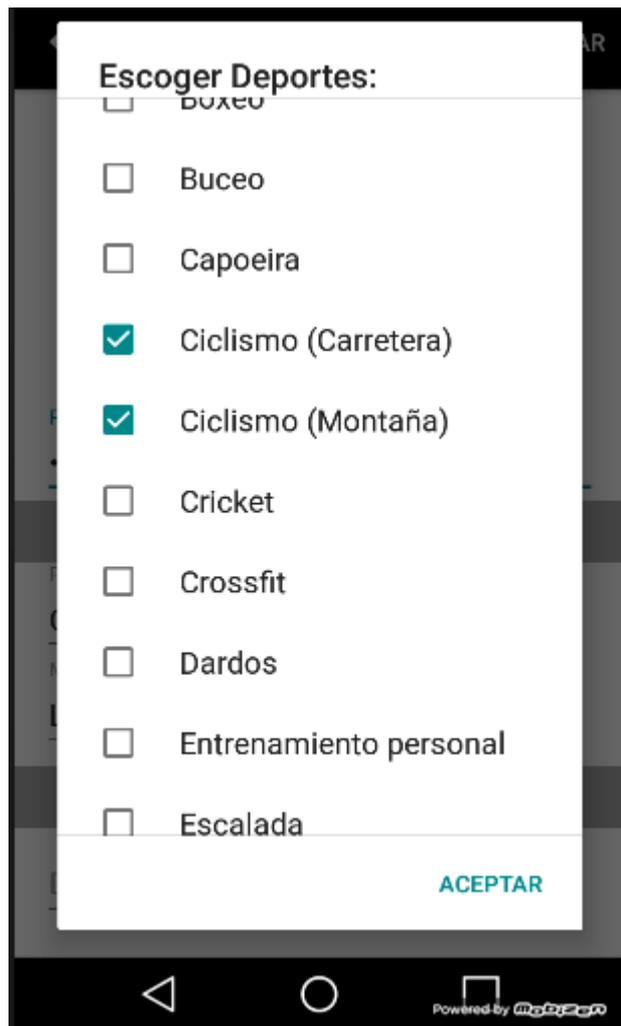


Figura 10.1.10: Interfaz categorías deportivas

En el formulario se utiliza un sistema de doble verificación obligando a introducir la contraseña deseada dos veces para evitar errores. En caso de que se introduzcan contraseñas diferentes y se seleccione la opción registrar, el formulario borra las contraseñas e indica que son incorrectas. Ver figura 10.1.11. También se realizan verificaciones comprobando que ningún campo del formulario está sin rellenar y que la dirección de correo electrónica no esté dada de alta ya en el sistema.

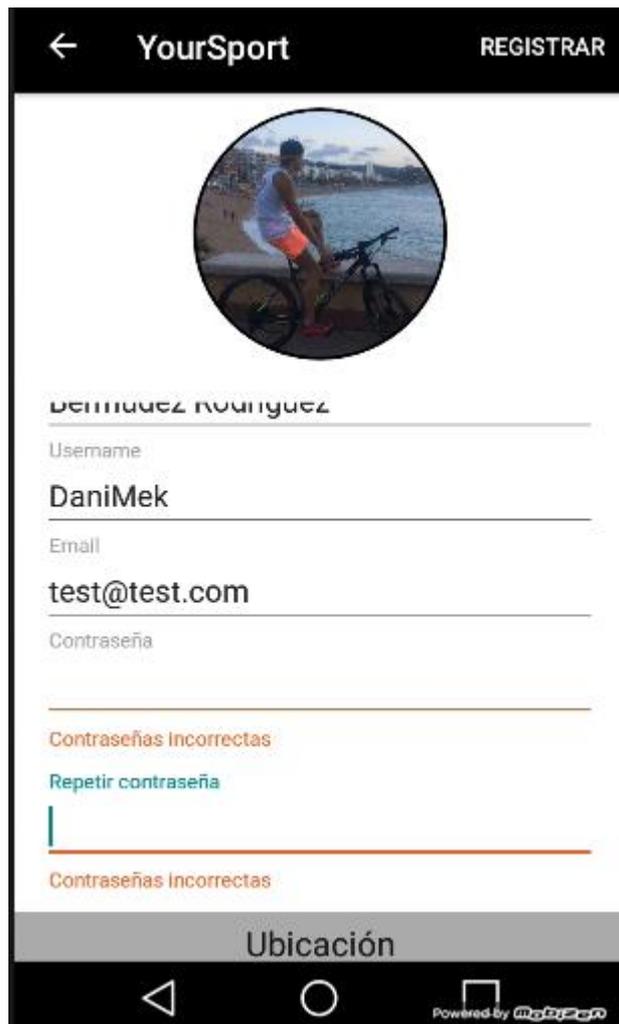


Figura 10.1.11: Verificación contraseña

Una vez registrado un nuevo usuario en el sistema se guarda en la base de datos la información introducida en el formulario. Entre ellos el token de identificación del dispositivo para las notificaciones y la contraseña cifrada. Ver figura 10.1.12.

password	tokenFireBase
1f8c0eff7a9d72f573e74464e21729551f89383bc958844...	cEXvhdcIZFU:APA91bGcN_MT02Prms6n...

Figura 10.1.12: Contraseña cifrada y token Firebase

Por otro lado, si se ha seleccionado una imagen de perfil esta se guarda en el sistema de ficheros del servidor y se asocia en la tabla imagen el nombre del fichero. Ver figura 10.1.13.

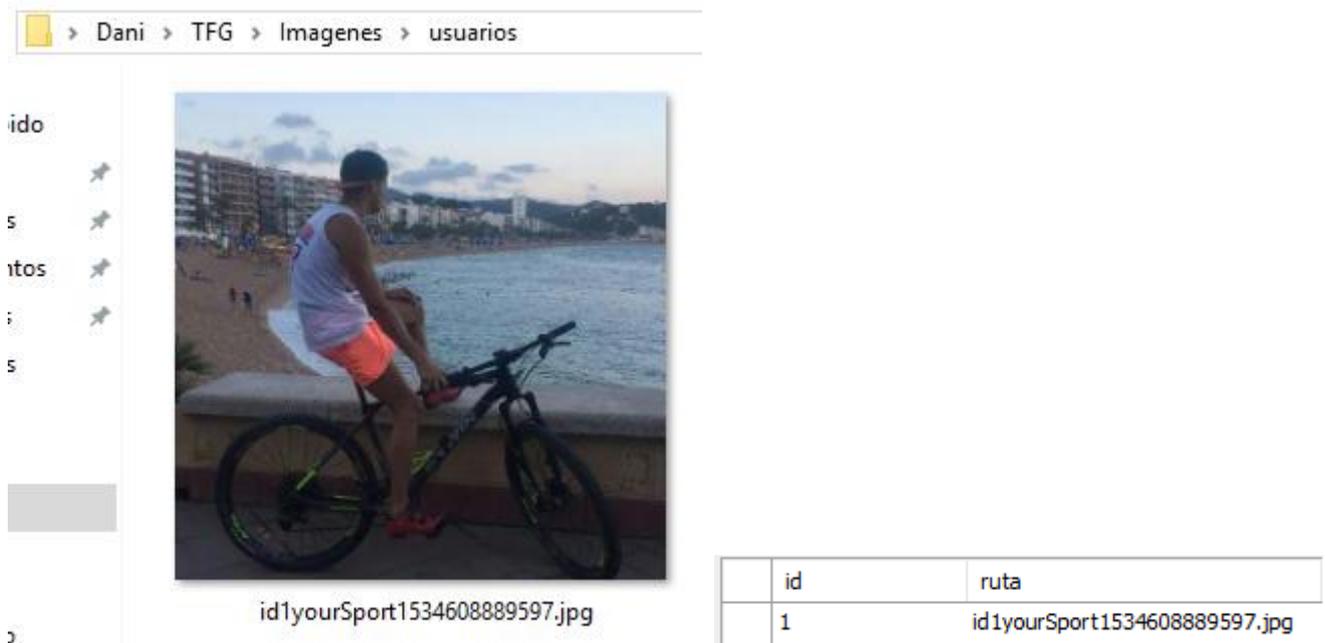


Figura 10.1.13: Imagen de perfil en el sistema de ficheros

Una vez un usuario se autentica o se da de alta en el sistema, automáticamente se inicia sesión redirigiendo la aplicación a la pantalla principal. En este punto el sistema comprueba si se tienen permisos para acceder a obtener la ubicación del dispositivo móvil. En caso de no tener los permisos asignados el sistema pregunta al usuario si permite acceder a su ubicación. Ver figura 10.1.14.

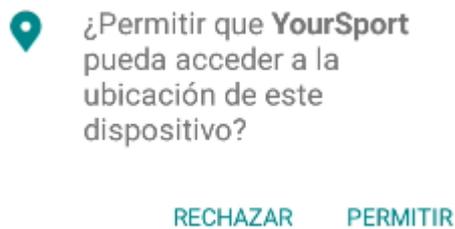


Figura 10.1.14: Pregunta permisos de acceso a la ubicación

Si se tienen los permisos y se obtiene de forma exitosa la ubicación del dispositivo esta se guarda en la base de datos. Ver figura 10.1.15.

id	direccion	latitud	longitud	municipio_id
1	Carrer Joan Baptista Lambert, 33, 17310 Lloret de Mar, Girona, España	41.7035586	2.8504655999999997	2542

Figura 10.1.15: Ubicación GPS registrada en base de datos

En la pantalla principal, en el momento en que se inicia sesión se listan los eventos deportivos que se celebren en el municipio de residencia del usuario y pertenezcan a sus categorías deportivas favoritas. En caso de no haber resultados se muestra un mensaje al usuario indicándole que puede utilizar el buscador. Ver figura 10.1.16.

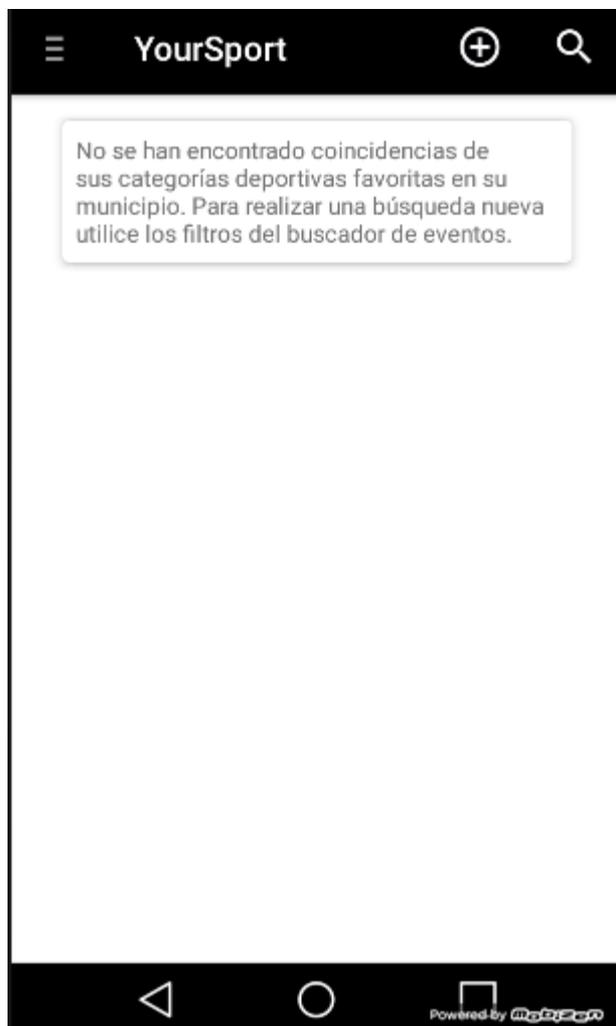


Figura 10.1.16: Interfaz principal sin eventos resultantes

En caso de que haya coincidencias con las preferencias por defecto se listan los eventos resultantes en una lista scrollable de tarjetas. Ver figura 10.1.17.

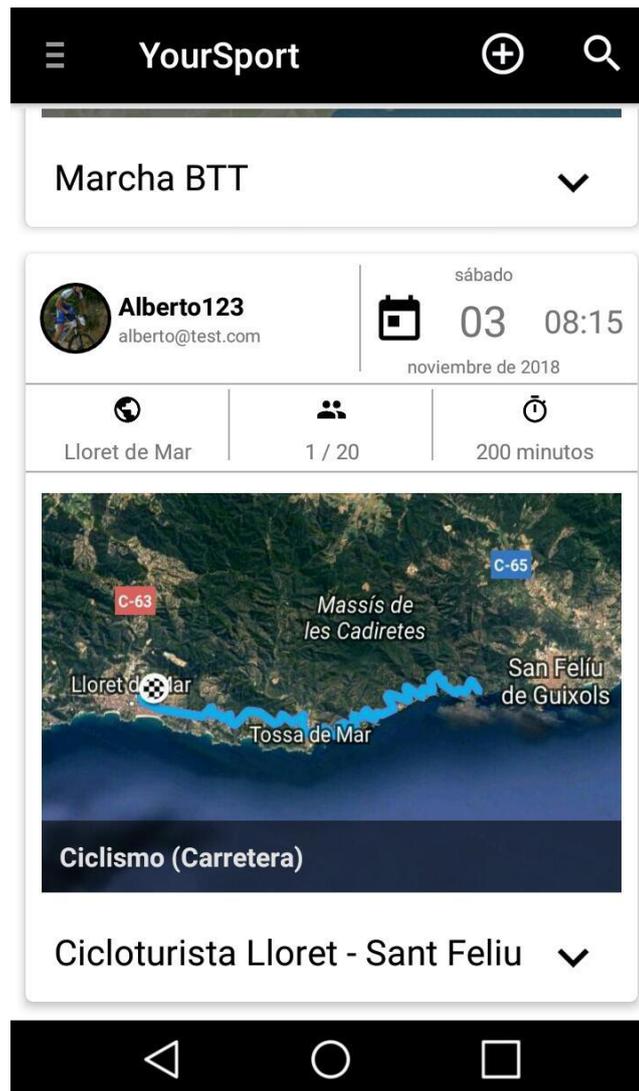


Figura 10.1.17: Interfaz principal listado eventos deportivos

En el margen superior izquierdo se encuentra el ícono que nos muestra el menú lateral de la aplicación. En este menú encontraremos todas las funcionalidades que nos permite realizar la aplicación. Por otro lado, en el margen derecho superior se encuentran también dos de las funcionalidades básicas de la aplicación, el buscador de eventos y el formulario para crear un nuevo evento deportivo. Ver figura 10.1.18.

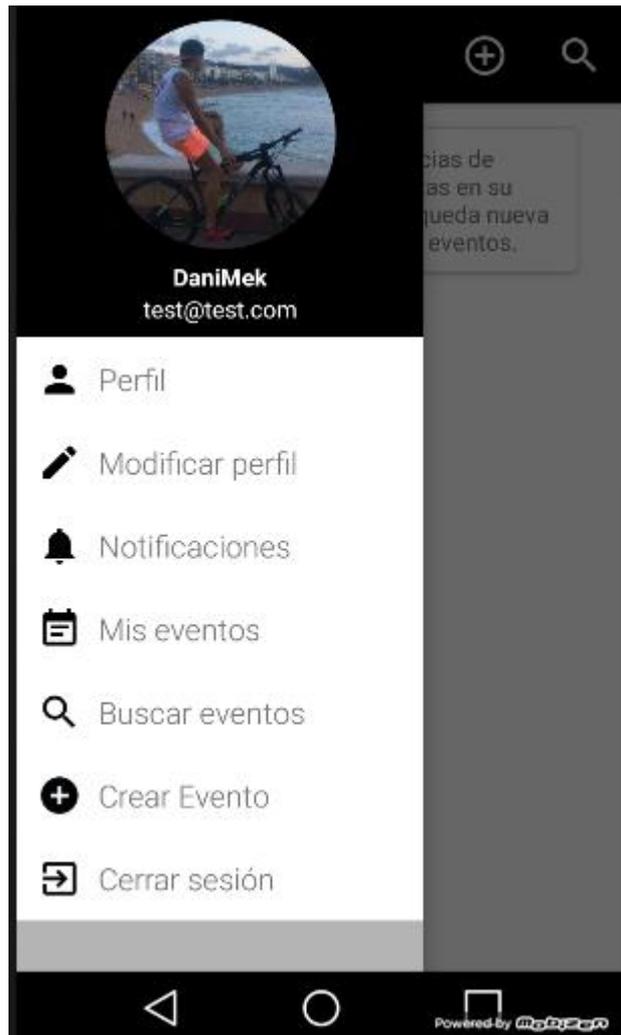


Figura 10.1.18: Interfaz principal menú lateral

Para realizar una búsqueda personalizada de eventos deportivos podemos acceder al formulario de búsqueda mediante el menú lateral, en la opción buscar eventos, o seleccionando el ícono de la lupa en la barra superior de la pantalla. Se podrán realizar búsquedas por coincidencias en el título de los eventos según el literal especificado, por categorías deportivas seleccionadas, por fecha de celebración y por ubicación. Ver figura 10.1.19.



Figura 10.1.19: Interfaz principal menú lateral

A la hora de seleccionar la fecha de celebración nos aparece un calendario en el cual podemos seleccionar la fecha deseada. Ver figura 10.1.20.

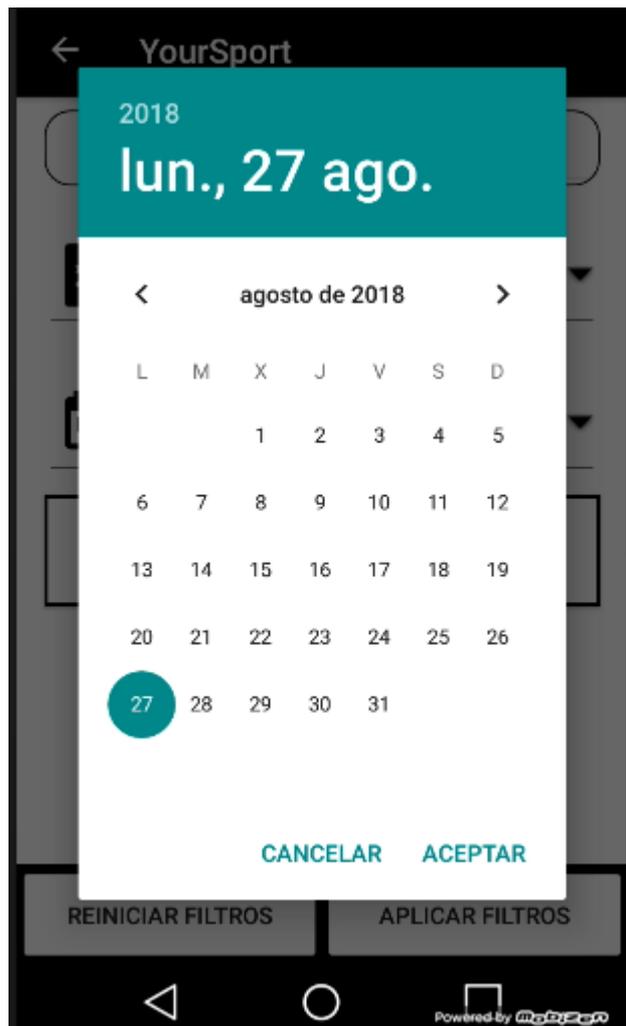


Figura 10.1.20: Interfaz principal menú lateral

Para seleccionar una ubicación determinada en que queremos buscar eventos deportivos, se disponen de dos opciones. Seleccionando la opción “cerca de mi” podemos seleccionar la distancia máxima a la que deseamos buscar eventos respecto a la ubicación GPS registrada del dispositivo móvil. En caso de seleccionar “ubicación” se muestran dos desplegables con las provincias y municipios disponibles en la aplicación. Ver figura 10.1.21.

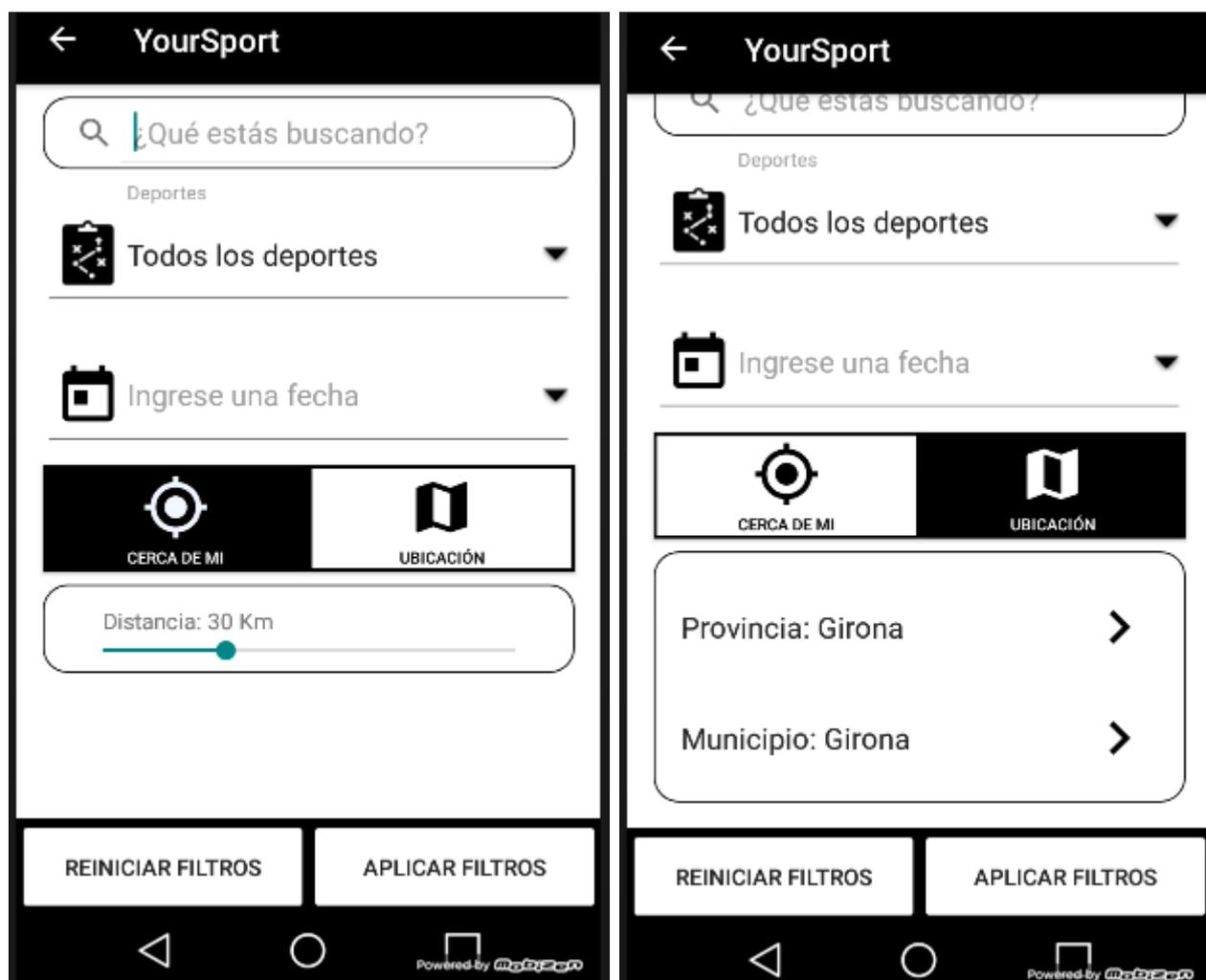


Figura 10.1.21: Seleccionar ubicación formulario búsquedas

Otra de las funcionalidades importantes de la aplicación es la de poder crear eventos deportivos. En la pantalla principal podemos acceder al formulario mediante el menú lateral seleccionando “crear evento” o seleccionando en la barra superior de la pantalla el icono de símbolo +. En este formulario podremos crear eventos deportivos adjuntando una imagen, informando de un título, descripción, categoría deportiva, número de participantes, duración estimada, privacidad del foro, día, hora y ubicación. Ver figura 10.1.22.

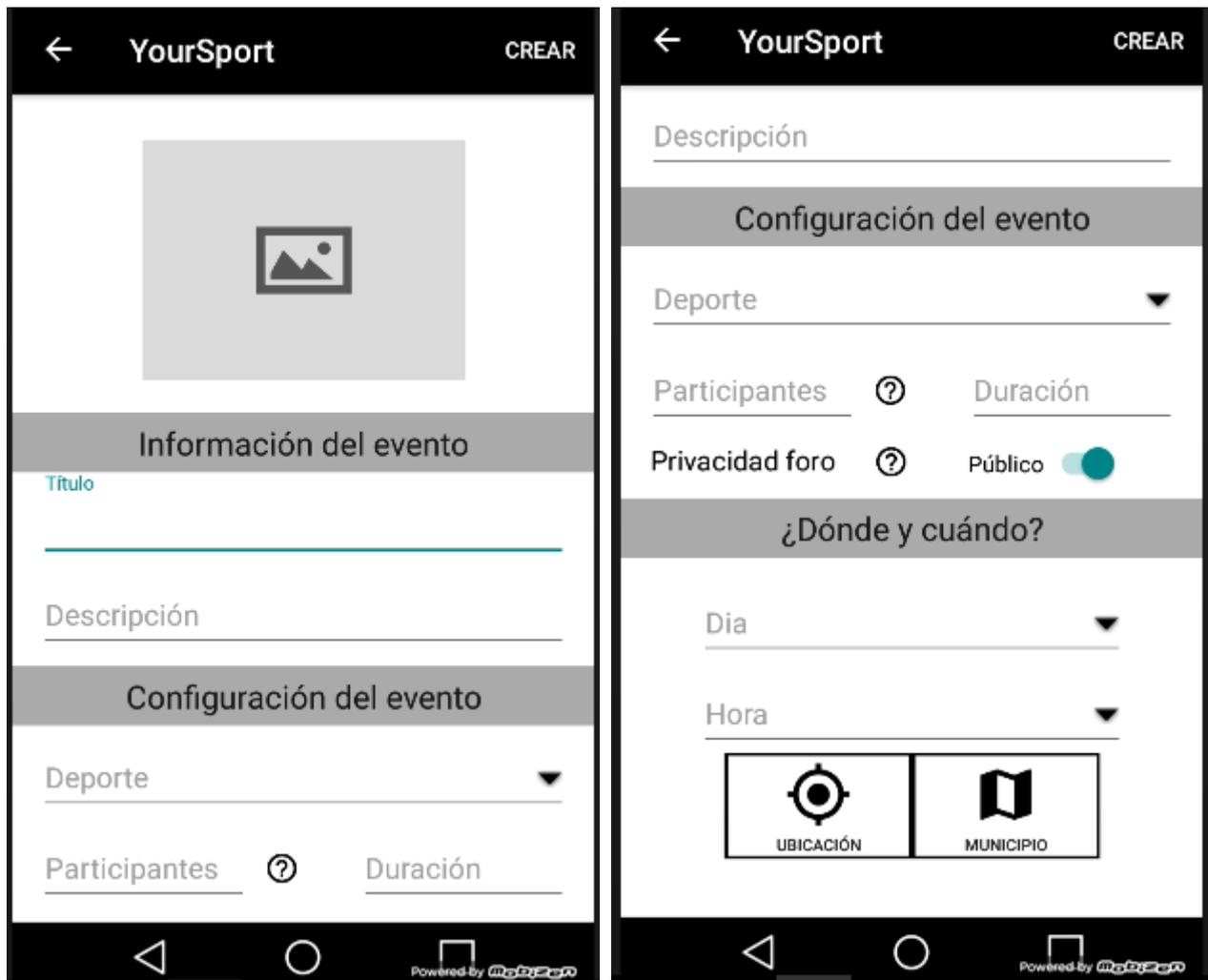


Figura 10.1.22: Seleccionar ubicación formulario búsquedas

El tratamiento de imágenes utiliza el mismo proceso que vimos anteriormente en el formulario de registro de usuarios. Para seleccionar la hora en la que se iniciará el evento de muestra un reloj en donde podemos seleccionar la hora deseada. Ver figura 10.1.22.

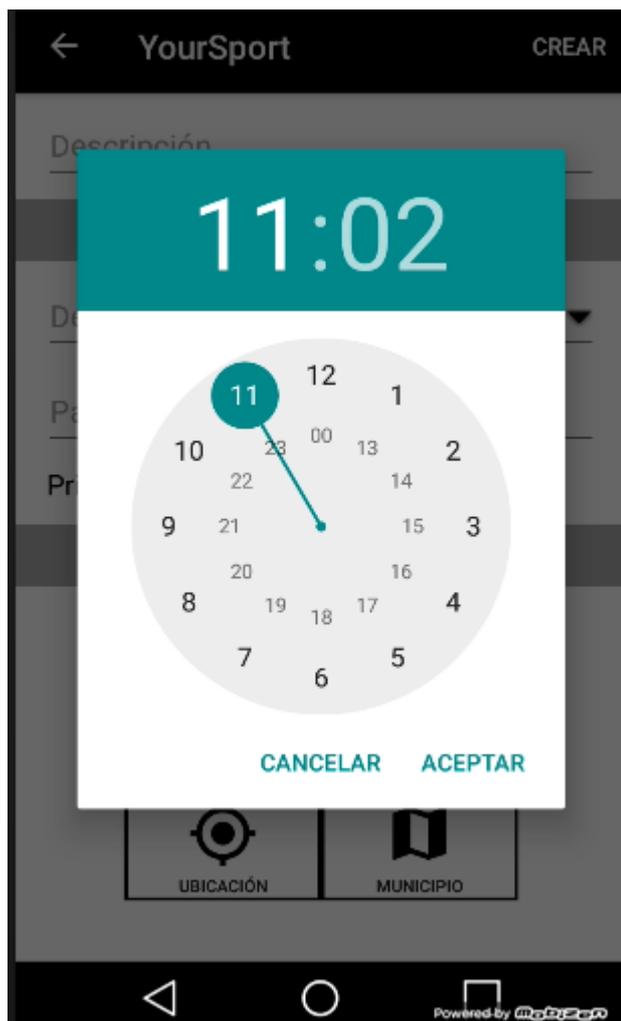


Figura 10.1.22: Seleccionar hora

A la hora de crear un evento deportivo debemos definir su ubicación geográfica. Disponemos de la opción “municipio” en la cual podremos seleccionar un municipio de los que nos proporciona la aplicación desde base de datos o seleccionando la opción “ubicación” nos muestra un mapa de Google en el cual podremos seleccionar una ubicación más precisa. Ver figura 10.1.23.

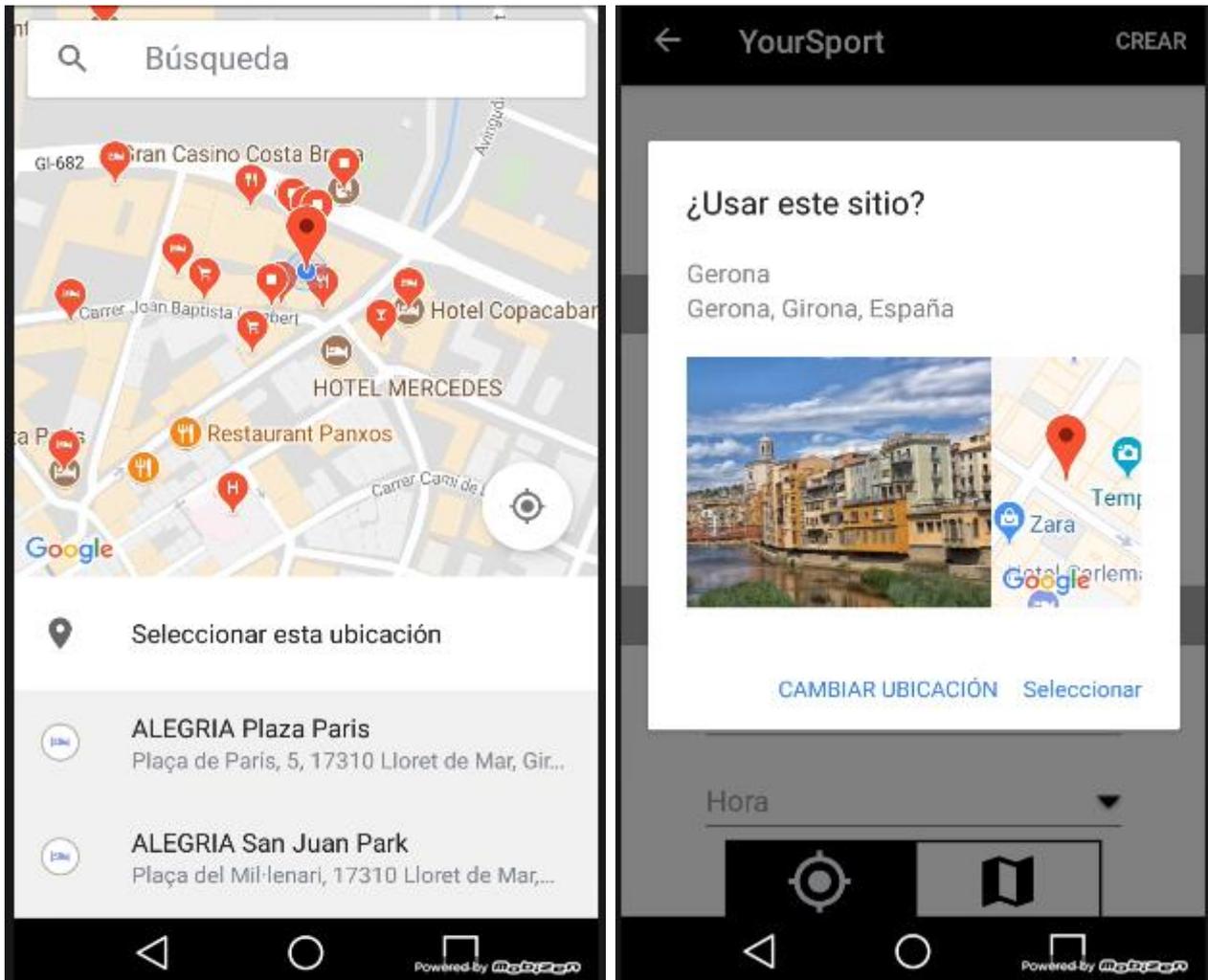


Figura 10.1.23: Seleccionar ubicación evento en mapa

A través del menú lateral podemos acceder a visualizar nuestros datos del perfil. En esta pantalla, a parte de la información del perfil, se muestra un listado scrollable horizontalmente con los eventos creados. Ver figura 10.1.24.

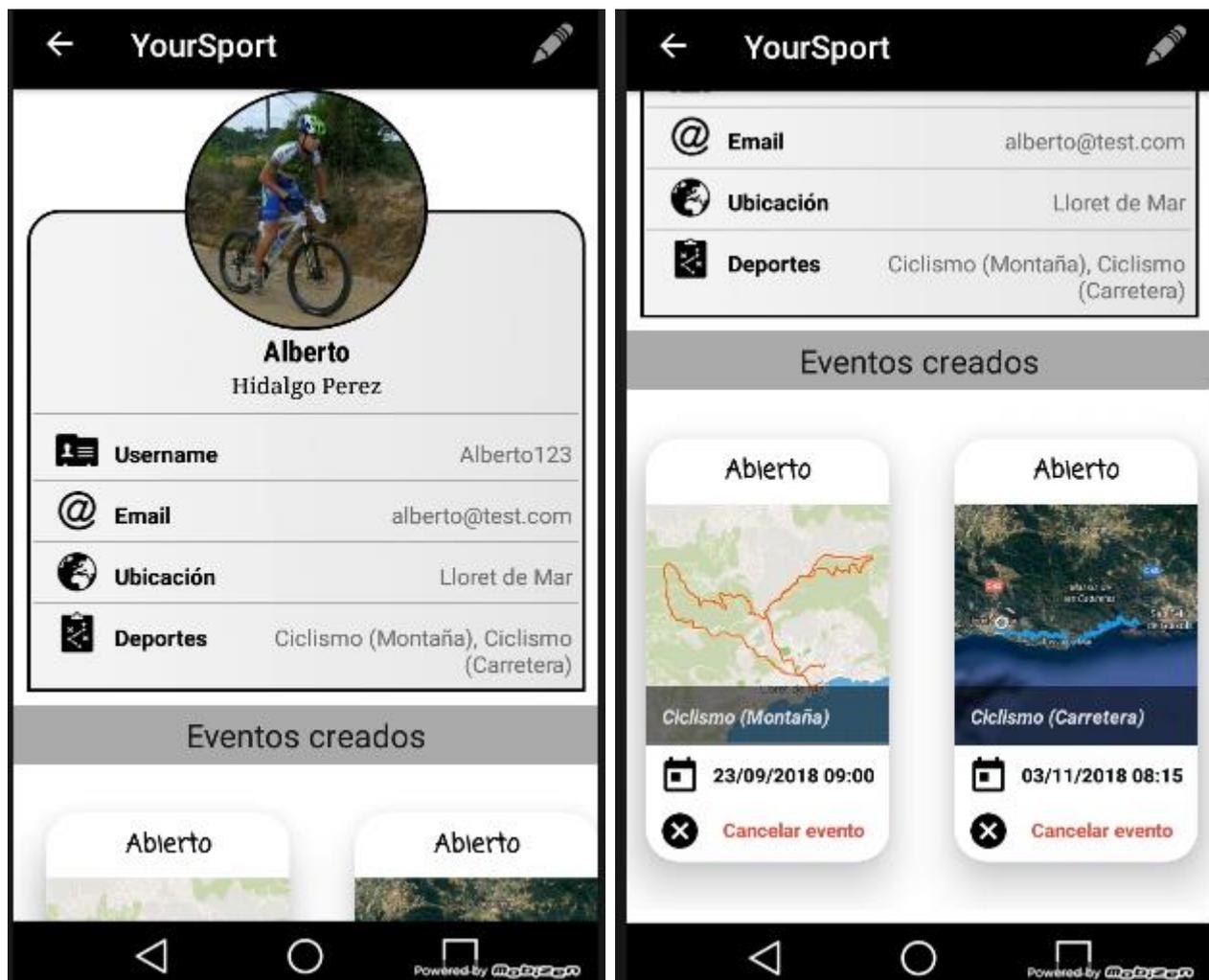


Figura 10.1.24: Pantalla información de perfil

Seleccionando la imagen de perfil del usuario está se mostrará en una nueva pantalla de forma individual. Ver figura 10.1.25.



Figura 10.1.25: Pantalla imagen

La aplicación permite modificar nuestros datos del perfil con los cuales el usuario se registró en la aplicación. Podemos actualizarlos accediendo a nuestro perfil y seleccionando el ícono de un lápiz en el margen superior derecho de la pantalla o a través de la opción del menú lateral “modificar perfil”. Se presenta un formulario editable relleno con los datos del usuario. Ver figura 10.1.26.

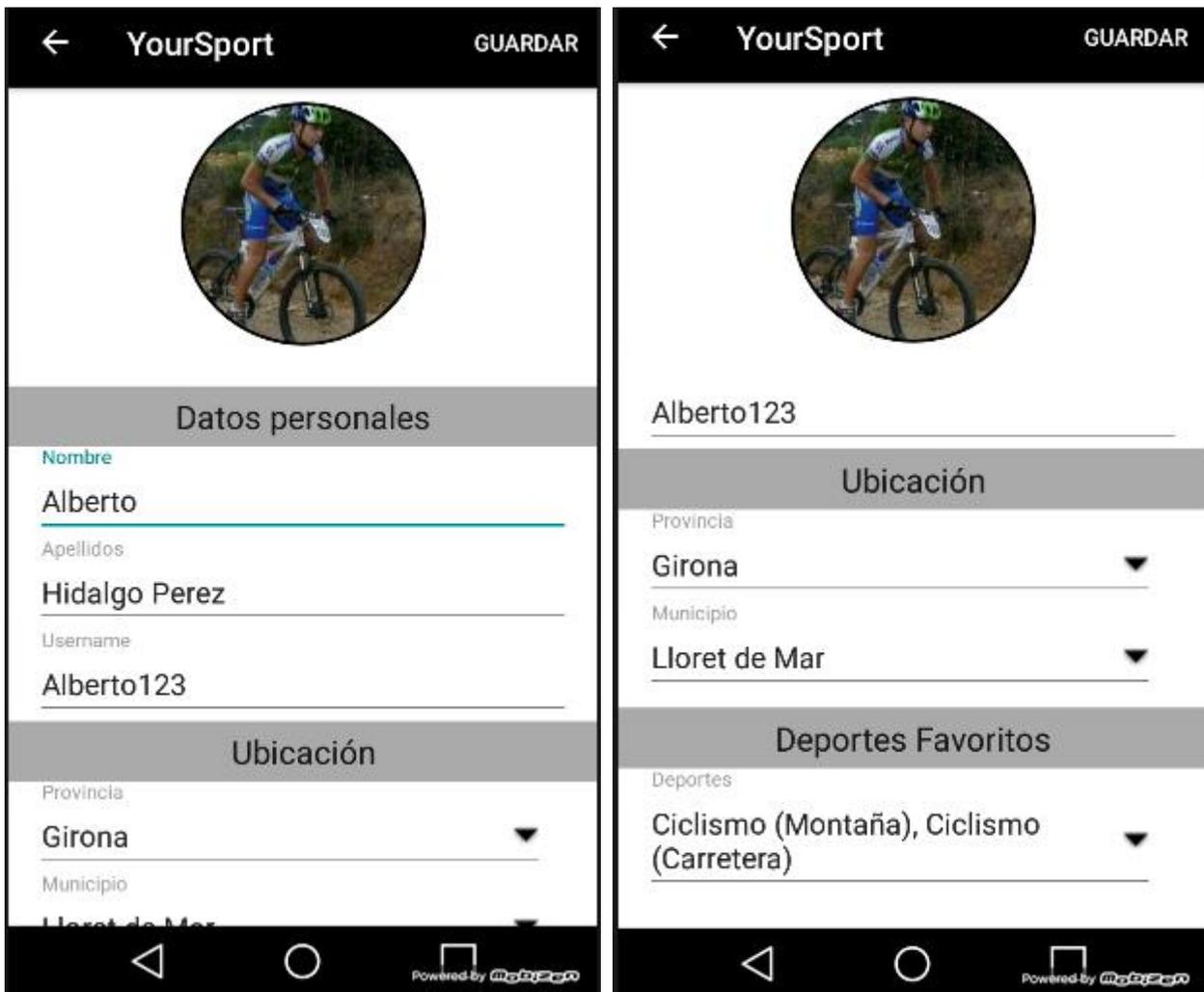


Figura 10.1.26: Formulario modificar perfil usuario

En el menú lateral los usuarios disponen de la opción “Mis eventos”. Al seleccionarla, la aplicación genera una pantalla con 3 pestañas que corresponden a los eventos creados por el usuario, eventos en el que el usuario está registrado como participante o los eventos en el que el usuario está en lista de espera. Ver figura 10.1.27.

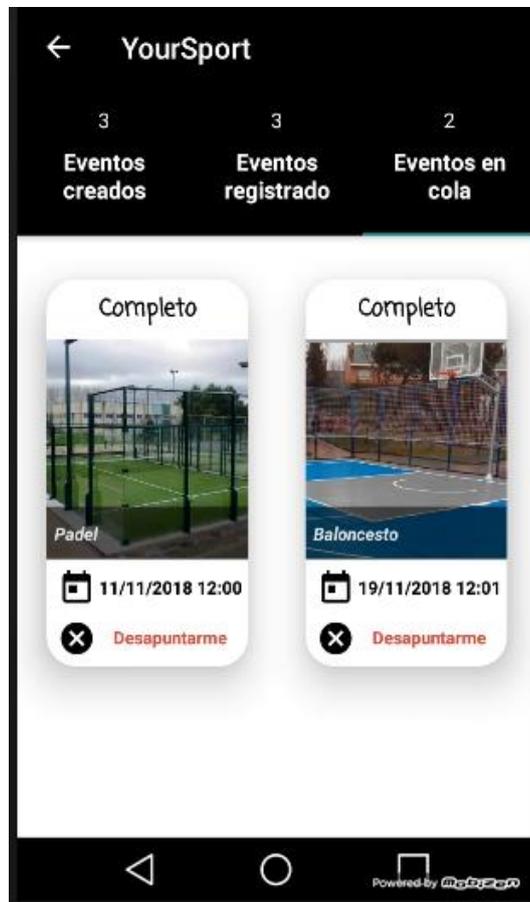
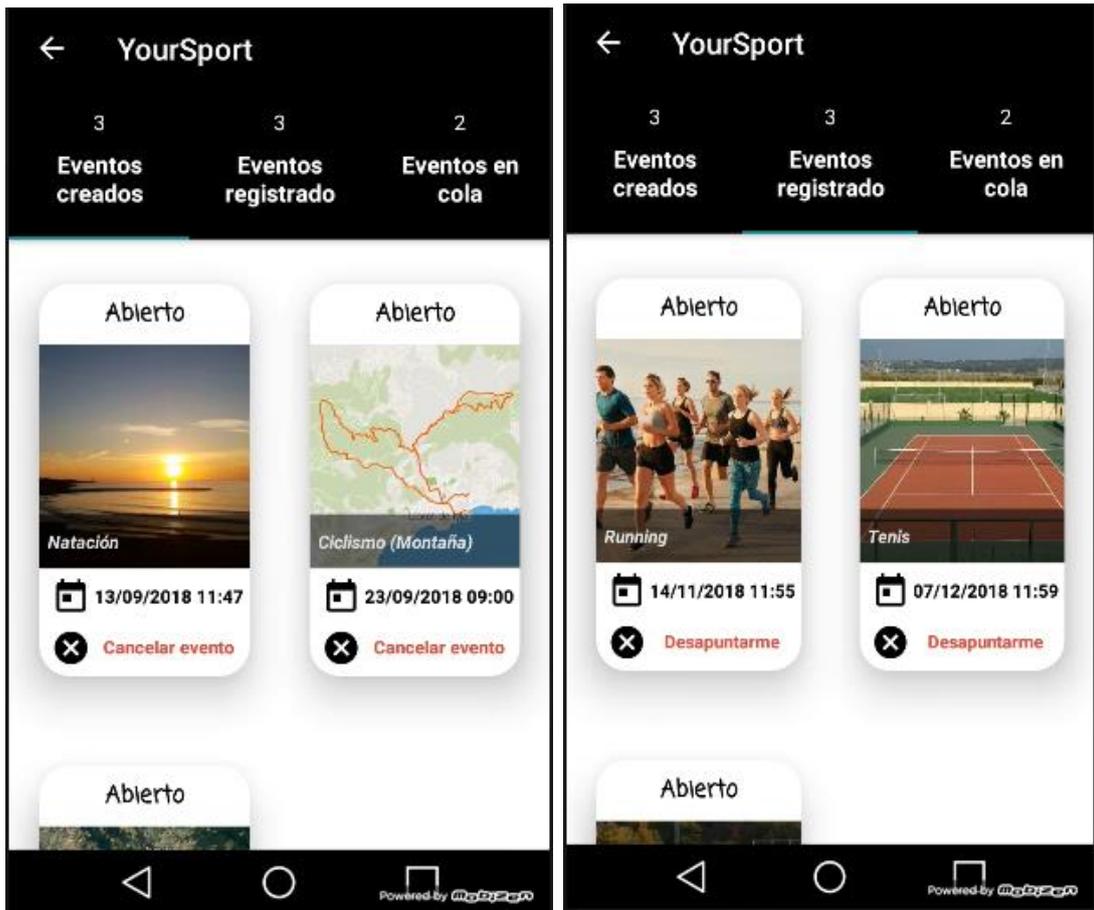


Figura 10.1.27: Pestañas eventos usuarios

En las 3 pestañas podemos acceder al detalle del evento pulsando sobre la tarjeta del evento que deseemos visualizar. En el listado de eventos creados, se dispone de la opción para suspender un evento. En los otros dos listados podemos desapuntarnos del evento ya sea como participante o de la lista de espera. En este caso, la tarjeta del evento desaparecería del listado y se actualizaría el contador.

A la hora de acceder al detalle de un evento, se puede acceder como propietario del evento o no. En caso de que el propietario del evento visualice el detalle de uno de sus eventos creados, dispondrá adicionalmente de la opción de modificar el evento y de expulsar usuarios de la lista de participantes y de la lista de espera. En caso de acceder como no administrador del evento, se dispondrán de las opciones de apuntarse o desapuntarse del evento.

El detalle de un evento deportivo se muestra en una pantalla con 5 pestañas diferentes. En la primera pestaña se muestra la información e imagen del evento deportivo, en la segunda el listado de usuarios registrados como participantes, en la tercera el listado de usuarios en cola, en la cuarta el foro del evento y en la última la ubicación del evento en el mapa.

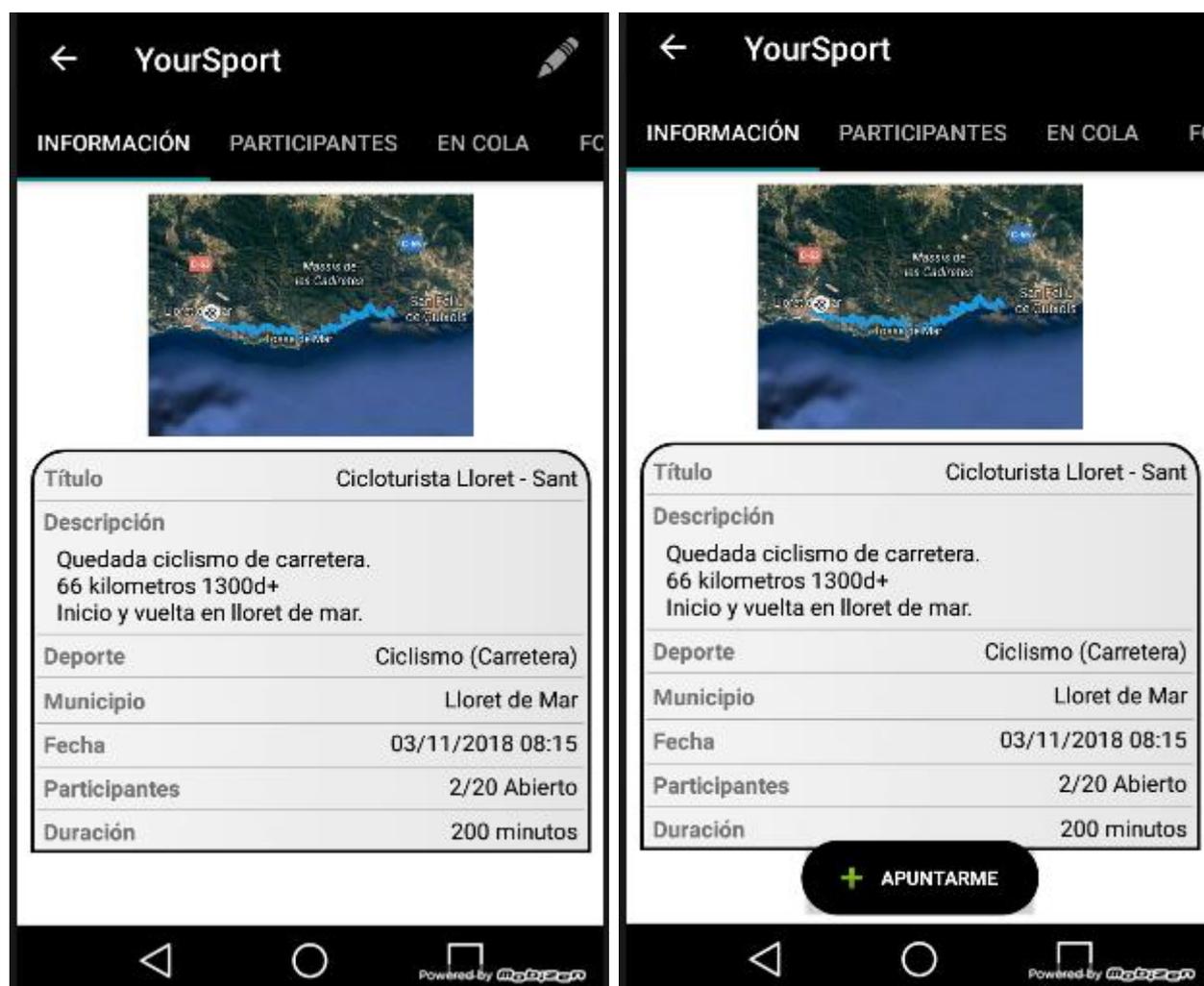


Figura 10.1.28: Pestaña información evento vista administrador y no administrador

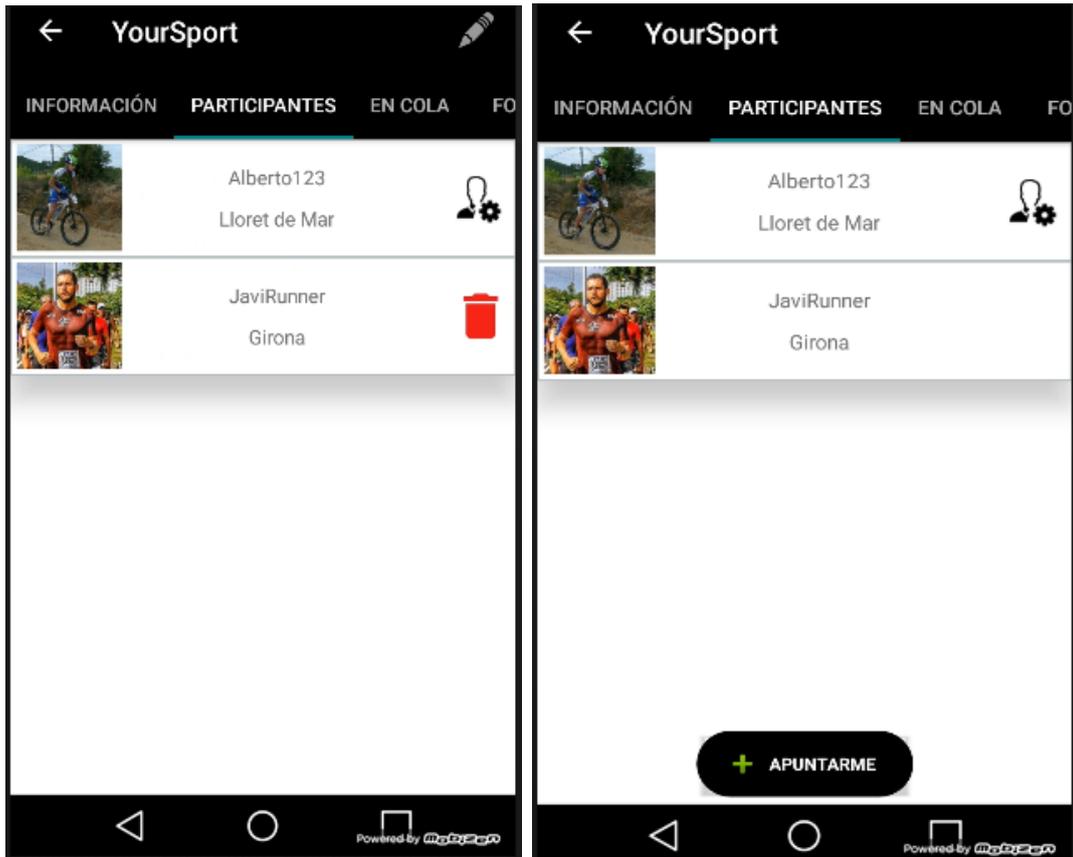


Figura 10.1.29: Pestaña participantes evento vista administrador y no administrador

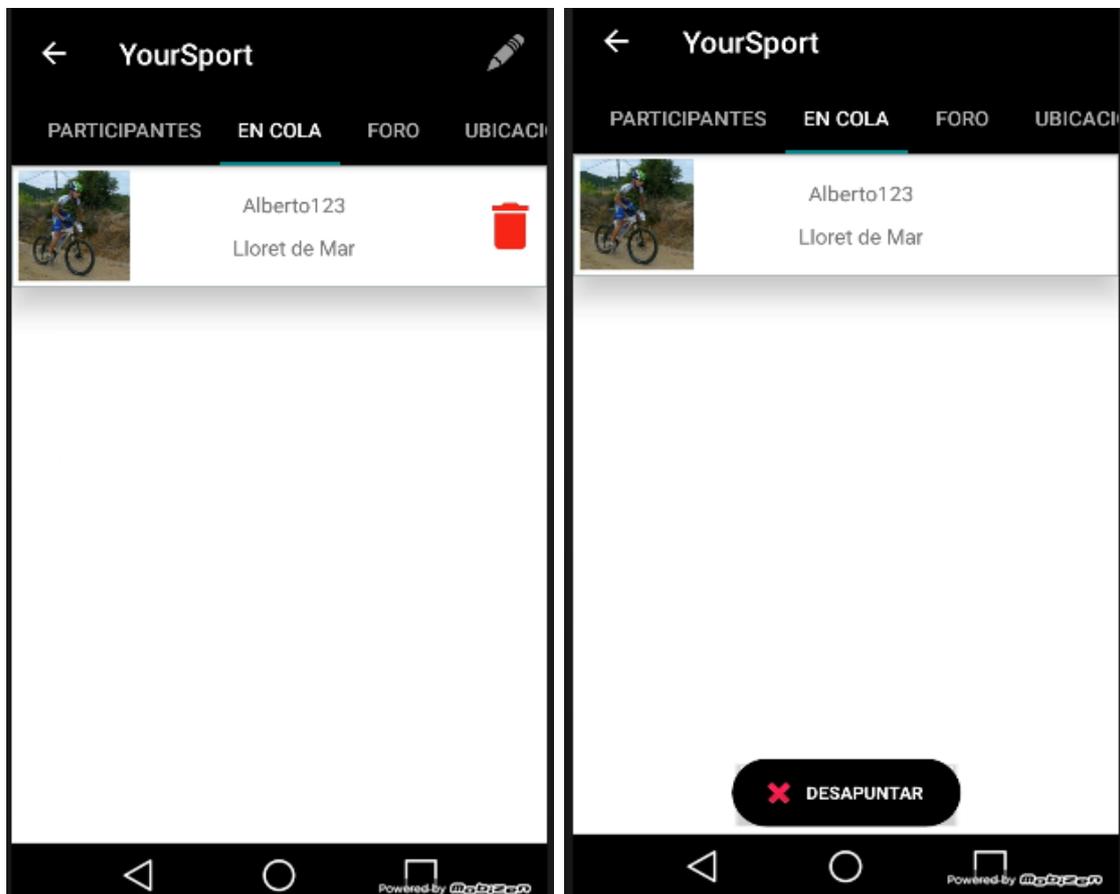


Figura 10.1.30: Pestaña participantes en cola vista administrador y no administrador

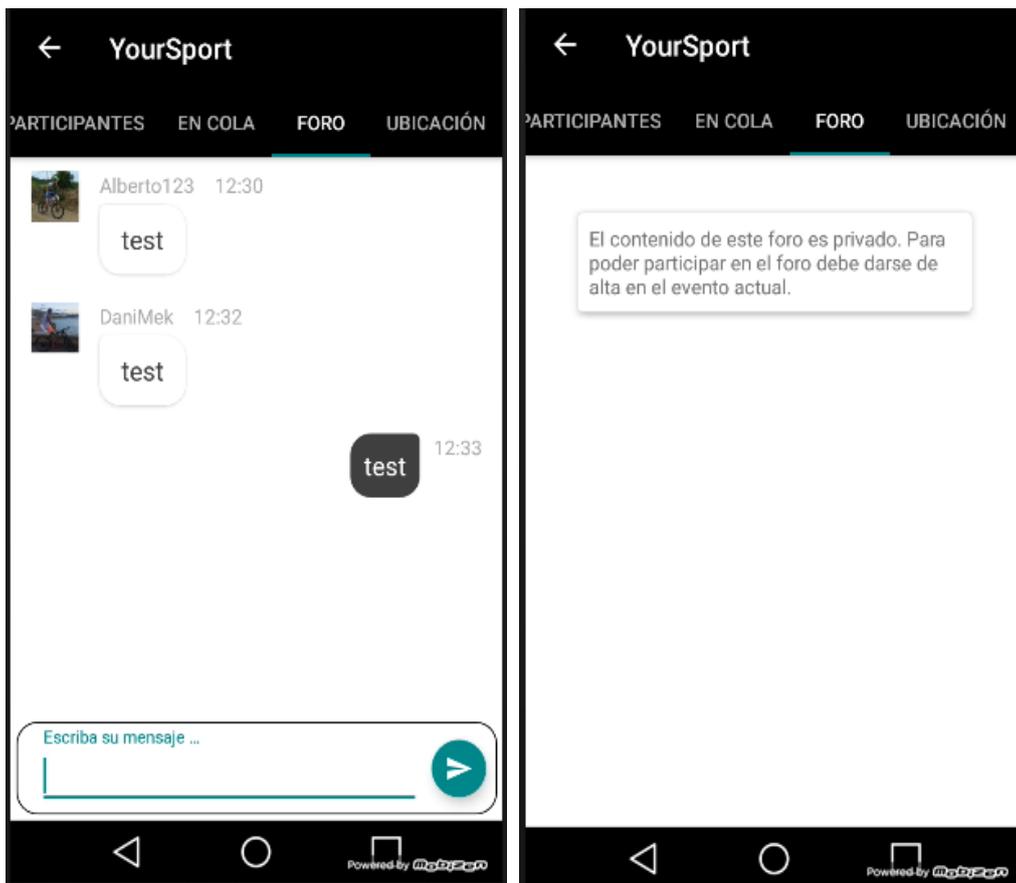


Figura 10.1.30: Pestaña foro evento público y privado



Figura 10.1.31: Pestaña ubicación evento

Vemos en las figuras 10.1.28 hasta la 10.1.31 que, al acceder como administrador a un evento, se dispone en la barra superior de un icono de un lápiz que nos permite modificar los datos del evento. Al seleccionarlo la aplicación nos redirige a un formulario editable con los datos del evento deportivo. Ver figura 10.1.32.

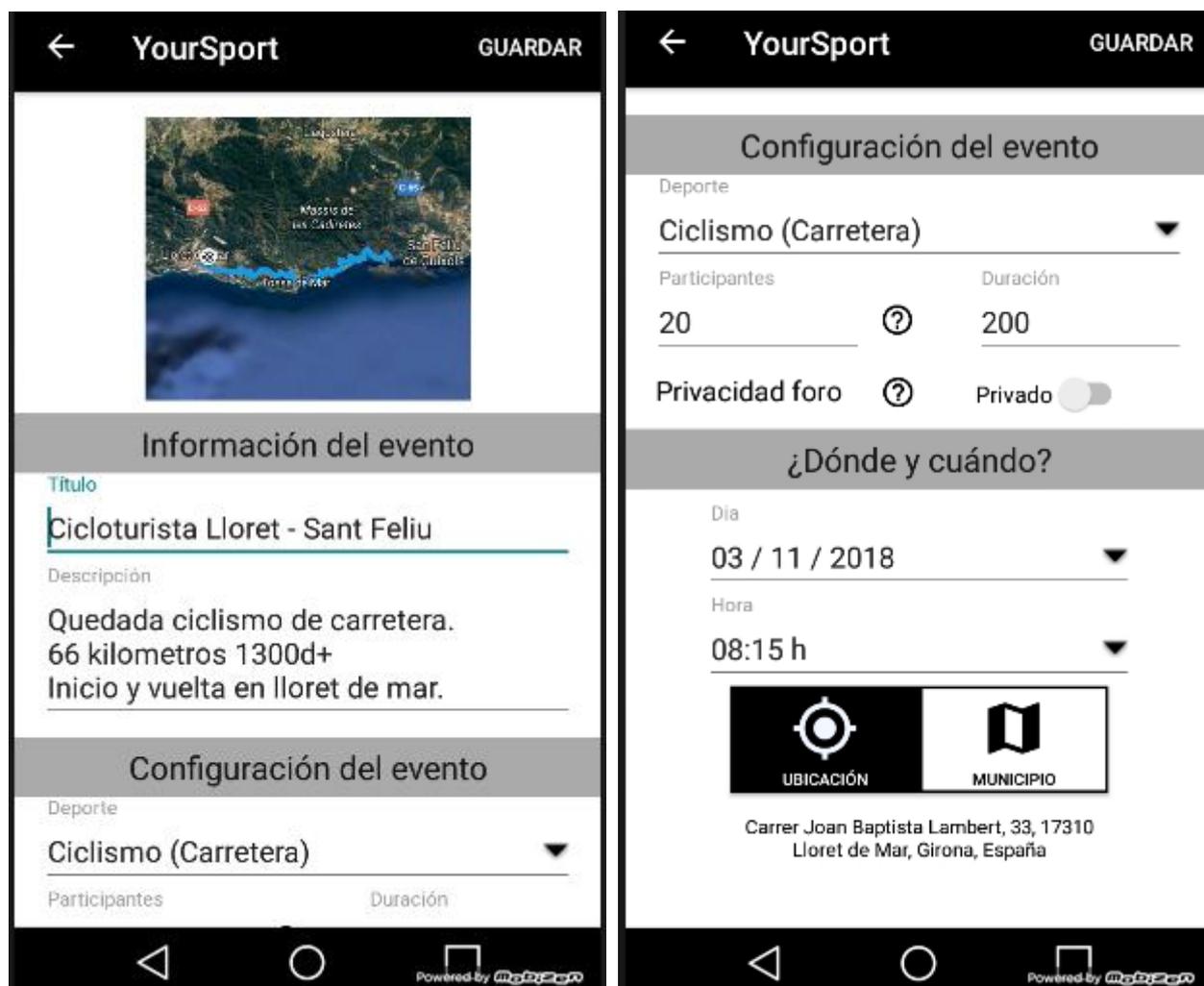


Figura 10.1.32: Interfaz modificar evento deportivo

Los usuarios pueden activar o desactivar las notificaciones disponibles en la aplicación. Para ello pueden acceder al menú de configuración de las notificaciones a través de la opción “Notificaciones” del menú lateral. Ver figura 10.1.33.

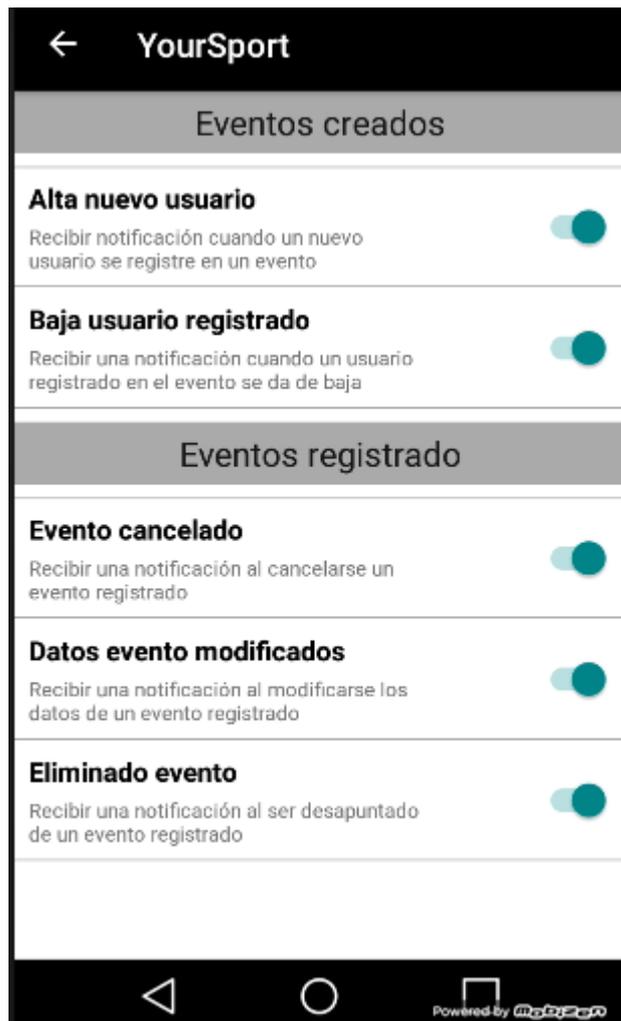


Figura 10.1.33: Interfaz configuración notificaciones



Figura 10.1.34: Ejemplo notificación recibida

## 10.2 Validez legal de la aplicación

Un aspecto muy importante a tener en cuenta a la hora de desarrollar aplicaciones es la validez legal de la aplicación. Debemos asegurar que la aplicación cumple con la legislación actual en cuanto a la Ley Orgánica de Protección de Datos (LOPD) y la Ley de Servicios de la Sociedad de la Información y Comercio Electrónico (LSSICE).

En primer lugar, es necesario recalcar que, por ahora la aplicación desarrollada no va a ser comercializada ni subida para el uso público, ya que se trata de una primera versión que abarca los inicios del proyecto.

Los datos de carácter personal, son datos que de alguna manera hacen referencia a una persona. La protección de datos es la capacidad que tienen los usuarios de disponer y decidir sobre todas las informaciones que le referencian.

Por lo tanto, en caso de que se decidiese publicar la aplicación, se tendrían que realizar todos los trámites para registrar los ficheros de seguridad, asignar un responsable de seguridad y tomar las precauciones necesarias sobre los datos almacenados. A los interesados a los cuales se les solicite datos personales tendrán que ser previamente informados de manera expresa, precisa e inequívoca de la existencia de un fichero o tratamiento de datos de carácter personal, de la finalidad de la recogida de los datos y de los destinatarios de la información. Por lo tanto, se debería informar a los usuarios de la recogida de datos y términos de condiciones y verificar si aceptan dichos términos en el momento de realizar el alta en la aplicación acogiéndose al derecho de información en la recogida de datos.

El responsable del fichero tendrá que adoptar las medidas de índole técnica y organizativas necesarias para garantizar la seguridad de los datos de carácter personal y evitar su alteración o pérdida.

La LOPD define 3 niveles de seguridad en función de la tipología de los datos (básico, medio y alto). En nuestro caso, los datos de carácter personal almacenados son el nombre, apellidos, correo electrónico, localización y lugar de residencia son considerados de nivel básico. El servidor en donde se alojase la aplicación y los datos, deberá cumplir las medidas de seguridad de nivel básico. Una de las medidas ya implementadas consiste en el almacenamiento ininteligible de las contraseñas.

Por otra banda, si nos centramos en el comercio electrónico, la aplicación desarrollada no incluye ningún tipo de servicio de pago. Por lo tanto, no aplica la Ley de Servicios de la Sociedad de la Información y Comercio Electrónico (LSSICE).

---

## 11. Conclusiones

---

Al iniciar este proyecto se plantearon varios objetivos. Una vez finalizado el proyecto se puede analizar su cumplimiento.

Antes de iniciar el proyecto, los conocimientos sobre el desarrollo de aplicaciones en Android eran muy básicos. Por lo tanto, en una primera fase de análisis y estudio de los lenguajes y herramientas, se realizó una formación básica para disponer de los conocimientos básicos para afrontar el desarrollo del proyecto. Posteriormente, durante la fase de implementación a través de las dudas y errores que fueron surgiendo, estos conocimientos se fueron fortaleciendo y asentando hasta el punto de haber adquirido el suficiente conocimiento como para desarrollar la aplicación diseñada.

Uno de los principales objetivos de la realización del proyecto era utilizar los conocimientos adquiridos a lo largo del grado en ingeniería informática, para crear una aplicación partiendo desde 0. De esta forma, se conoce y consolida el proceso completo a seguir para la realización de un proyecto de esta envergadura. Para el desarrollo de un proyecto se deben realizar tareas previas a la implementación como el estudio de la idea a desarrollar, estudio de mercado, escoger una metodología de trabajo adecuada, planificar el tiempo empleado en las tareas, deducir costes totales del desarrollo, definir requisitos funcionales y no funcionales, analizar y estudiar las diferentes herramientas a utilizar, implementar y documentar adecuadamente. Llevar a la práctica todo este ciclo para la realización de la aplicación me ha permitido enriquecer y consolidar conocimientos adquiridos a lo largo de mis estudios.

Respecto a los objetivos definidos en el *capítulo 1.4*, se han consolidado exitosamente, ya que finalmente se ha realizado la aplicación servidor basada en JavaEE, la aplicación cliente en Android, se ha diseñado e implementado el modelo de datos y la API REST para comunicar dichas aplicaciones entre sí, permitiendo crear una aplicación que proporcione a los usuarios una red social deportiva destinada principalmente a la práctica de deporte en grupo, mediante la gestión de eventos deportivos, adaptándose a las preferencias y disponibilidades horarias de los diferentes usuarios. Durante la etapa de implementación, incluso habiendo realizado un estudio y análisis previo de las herramientas a utilizar, han ido surgiendo diversas dificultades para realizar algunas tareas. Esto sumado a contratiempos por motivos laborales hicieron que tomara la decisión de retrasar la entrega del proyecto, tal y como se explica en el *capítulo 4*.

En cuanto a los requerimientos funcionales definidos en el *capítulo 6.1*, se han podido finalmente implementar, permitiendo así a los usuarios realizar todas las funcionalidades previstas en esta primera versión de la aplicación.

A nivel personal, considero que la elaboración de este proyecto aporta un gran grado de experiencia adquirida ya que ha sido un aprendizaje continuo. He aprendido sobre el funcionamiento de varios servicios y librerías de gran utilidad como Firebase, Hibernate, etc.

Siendo autocrítico, soy consciente de que el proyecto desarrollado no se puede considerar una versión final como para comercializar o publicar la aplicación en el mercado. Por lo tanto, se considera una potente primera versión que cumple con las funcionalidades básicas de una red social deportiva. Pero esto no exime que esté sujeta a mejoras o ampliaciones, tal y como se detalla en el *capítulo 12*.

---

## 12. Trabajo futuro

---

El proyecto realizado es el resultado de un trabajo final de grado y, por lo tanto, está sujeto a mejoras y ampliaciones de sus funcionalidades. La aplicación no se ha publicado ya que se considera una primera versión estable pero no una versión definitiva como para publicarla.

En cuanto a las posibles mejoras, destacaría la contratación de un diseñador gráfico profesional para re estilizar las interfaces de usuario. Actualmente, las interfaces han sido diseñadas de forma que cumplan los requisitos básicos para poder operar con la aplicación y satisfacer todas las funcionalidades descritas. Con la incorporación de un profesional, se pretende mejorar las interfaces de forma que sean más vistosas y dinámicas para los usuarios ya que, corresponden a la parte visual del proyecto y gran parte de la impresión de los usuarios sobre la aplicación depende de éstas.

Actualmente, la aplicación está desarrollada únicamente en lengua castellana y su alcance se limita al territorio español. Una posible ampliación sería la implementación de la aplicación multidioma. Se incorporaría la opción de escoger el idioma al acceder a la aplicación pudiendo seleccionar castellano, catalán o inglés. Todo y que se haya desarrollada para el territorio español, no debemos olvidar que en caso de que la aplicación estuviese publicada, en España hay una gran cantidad de turismo y el hecho de disponer de la aplicación en inglés incentivaría el uso a turistas de vacaciones en busca de realizar actividades deportivas en grupo.

Una posible funcionalidad a incorporar sería la de añadir un servicio de pago, lo cual podría incentivar a organizadores de carreras de cualquier categoría deportiva, que sean de pago, a gestionar las inscripciones y pagos a través de la aplicación.

La aplicación dispone de un buscador de eventos deportivos y de un foro por cada evento deportivo. En un futuro, se pretende añadir un buscador de usuarios y la posibilidad de chatear entre usuarios, sin la necesidad de que estén inscritos en el mismo evento deportivo. De esta forma, se pueden buscar eventos deportivos a través del buscador de usuarios y accediendo a su perfil podemos visualizar los eventos creados de dichos usuarios. Con el chat entre usuarios se pretende facilitar la comunicación entre los usuarios de la aplicación pudiendo organizar conjuntamente nuevos eventos deportivos.

---

## 13. Bibliografía

---

- [1] Ministerio de Educación, Cultura y deporte. *Anuario de Estadísticas deportivas 2018. Gráficos* [en línea]. <[https://www.mecd.gob.es/servicios-al-ciudadano-mecd/dms/mecd/servicios-al-ciudadano-mecd/estadisticas/deporte/anuario-deporte/AED-2018/Graficos\\_Anuario\\_de\\_Estadisticas\\_Deportivas\\_2018/Graficos\\_Anuario\\_de\\_Estad%ADsticas\\_Deportivas\\_2018.pdf](https://www.mecd.gob.es/servicios-al-ciudadano-mecd/dms/mecd/servicios-al-ciudadano-mecd/estadisticas/deporte/anuario-deporte/AED-2018/Graficos_Anuario_de_Estadisticas_Deportivas_2018/Graficos_Anuario_de_Estad%ADsticas_Deportivas_2018.pdf)>
- [2] Wikipedia. *Sistema operativo móvil* [en línea]. <[https://es.wikipedia.org/wiki/Sistema\\_operativo\\_m%C3%B3vil](https://es.wikipedia.org/wiki/Sistema_operativo_m%C3%B3vil)>
- [3] Wikipedia. *Windows 10 Mobile* [en línea]. <[https://es.wikipedia.org/wiki/Windows\\_10\\_Mobile](https://es.wikipedia.org/wiki/Windows_10_Mobile)>
- [4] Kantar Worldpanel. *Android vs iOS* [en línea]. <<https://www.kantarworldpanel.com/global/smartphone-os-market-share/>>
- [5] Android Developers. *Versiones de la plataforma* [en línea]. <<https://developer.android.com/about/dashboards/#Screens>>
- [6] WildFly. *About* [en línea]. <<http://wildfly.org/about/>>
- [7] Oracle. *Introduction to Java Platform, Enterprise Edition 7* [en línea]. <<https://www.oracle.com/technetwork/java/javaee/javaee7-whitepaper-1956203.pdf>>
- [8] JBoss. *RESTEasy JAX-RS* [en línea]. <[https://docs.jboss.org/resteasy/docs/3.0.9.Final/userguide/html\\_single/](https://docs.jboss.org/resteasy/docs/3.0.9.Final/userguide/html_single/)>
- [9] JBoss. *JBoss EJB3 Tutorials* [en línea]. <<http://docs.jboss.org/ejb3/docs/tutorial/1.0.7/html/index.html>>
- [10] JBoss. *Hibernate ORM Final User Guide* [en línea]. <[http://docs.jboss.org/hibernate/orm/5.3/userguide/html\\_single/Hibernate\\_User\\_Guide.html](http://docs.jboss.org/hibernate/orm/5.3/userguide/html_single/Hibernate_User_Guide.html)>
- [11] Eclipse. *Eclipse IDE for Java EE Developers* [en línea]. <<https://www.eclipse.org/downloads/packages/release/oxygen/3a/eclipse-ide-java-ee-developers>>

- [12] Postman. *Introduction* [en línea]. <<https://www.getpostman.com/docs/v6/>>
- [13] JSON. *Introducing JSON* [en línea]. <<https://www.json.org/>>
- [14] Android Developers. *Actividades* [en línea]. <<https://developer.android.com/guide/components/activities?hl=es-419>>
- [15] Android Developers. *Fragments* [en línea]. <<https://developer.android.com/guide/components/fragments?hl=es-419>>
- [16] Android Developers. *Intents y filtros de intents* [en línea]. <<https://developer.android.com/guide/components/intents-filters?hl=es-419>>
- [17] Android Developers. *Diseños* [en línea]. <<https://developer.android.com/guide/topics/ui/declaring-layout?hl=es-419>>
- [18] Android Developers. *Conoce Android Studio* [en línea]. <<https://developer.android.com/studio/intro/?hl=es-419>>
- [19] Retrofit. *A type-safe HTTP client for Android and Java* [en línea]. <<https://square.github.io/retrofit/>>
- [20] Glide v4. *Fast and efficient image loading for Android* [en línea]. <<https://bumptech.github.io/glide/>>
- [21] Firebase. *Documentación* [en línea]. <<https://firebase.google.com/docs/?hl=es-419>>
- [22] Firebase. *Agrega Firebase a tu proyecto Android* [en línea]. <<https://firebase.google.com/docs/android/setup?hl=es-419>>
- [23] Firebase. *Firebase Realtime Database* [en línea]. <<https://firebase.google.com/docs/database/?hl=es-419>>
- [24] Firebase. *Firebase Cloud Messaging* [en línea]. <<https://firebase.google.com/docs/cloud-messaging/?hl=es-419>>
- [25] NinjaMocks. *Online wireframe and mockup tool* [en línea]. <<https://ninjamock.com/>>
- [26] Wikipedia. *PBKDF2* [en línea]. <<https://en.wikipedia.org/wiki/PBKDF2>>
- [27] Android Developers. *Create a List with RecyclerView* [en línea]. <<https://developer.android.com/guide/topics/ui/layout/recyclerview>>
- [28] Android Developers. *Create a Card-Based Layout* [en línea]. <<https://developer.android.com/guide/topics/ui/layout/cardview>>

- [29] Android Developers. *Adding Swipe-to-Refresh To Your App* [en línea]. <<https://developer.android.com/training/swipe/add-swipe-interface>>
- [30] Android Developers. *LocationManager* [en línea]. <<https://developer.android.com/reference/android/location/LocationManager>>
- [31] Apache Maven Project. *Whats is Maven?* [en línea]. <<https://maven.apache.org/what-is-maven.html>>

---

## 14. Anexos

---

En este capítulo adjuntamos los scripts SQL utilizados para importar los distintos datos en la aplicación.

Estados de un evento deportivo:

```
INSERT INTO Estado VALUES (1, 'Abierto');
INSERT INTO Estado VALUES (2, 'Completo');
INSERT INTO Estado VALUES (3, 'Suspendido');
INSERT INTO Estado VALUES (4, 'Finalizado');
```

Categorías deportivas disponibles en la aplicación:

```
INSERT INTO Deporte (id, deporte) VALUES (1, 'Aerobic');
INSERT INTO Deporte (id, deporte) VALUES (2, 'Airsoft');
INSERT INTO Deporte (id, deporte) VALUES (3, 'Ajedrez');
INSERT INTO Deporte (id, deporte) VALUES (4, 'Badminton');
INSERT INTO Deporte (id, deporte) VALUES (5, 'Balón prisionero');
INSERT INTO Deporte (id, deporte) VALUES (6, 'Baloncesto');
INSERT INTO Deporte (id, deporte) VALUES (7, 'Balonmano');
INSERT INTO Deporte (id, deporte) VALUES (8, 'Béisbol');
INSERT INTO Deporte (id, deporte) VALUES (9, 'Billar');
INSERT INTO Deporte (id, deporte) VALUES (10, 'Bolos');
INSERT INTO Deporte (id, deporte) VALUES (11, 'Boxeo');
INSERT INTO Deporte (id, deporte) VALUES (12, 'Buceo');
INSERT INTO Deporte (id, deporte) VALUES (13, 'Capoeira');
INSERT INTO Deporte (id, deporte) VALUES (14, 'Ciclismo (Carretera)');
INSERT INTO Deporte (id, deporte) VALUES (15, 'Ciclismo (Montaña)');
INSERT INTO Deporte (id, deporte) VALUES (16, 'Cricket');
INSERT INTO Deporte (id, deporte) VALUES (17, 'Crossfit');
INSERT INTO Deporte (id, deporte) VALUES (18, 'Dardos');
INSERT INTO Deporte (id, deporte) VALUES (19, 'Entrenamiento personal');
INSERT INTO Deporte (id, deporte) VALUES (20, 'Escalada');
INSERT INTO Deporte (id, deporte) VALUES (21, 'Esgrima');
INSERT INTO Deporte (id, deporte) VALUES (22, 'Frontenis');
INSERT INTO Deporte (id, deporte) VALUES (23, 'Fútbol 7');
INSERT INTO Deporte (id, deporte) VALUES (24, 'Fútbol 11');
INSERT INTO Deporte (id, deporte) VALUES (25, 'Fútbol Sala');
INSERT INTO Deporte (id, deporte) VALUES (26, 'Fútbolín');
INSERT INTO Deporte (id, deporte) VALUES (27, 'Futbol Americano');
INSERT INTO Deporte (id, deporte) VALUES (28, 'Futbol Playa');
INSERT INTO Deporte (id, deporte) VALUES (29, 'Gimnasio');
INSERT INTO Deporte (id, deporte) VALUES (30, 'Golf');
INSERT INTO Deporte (id, deporte) VALUES (31, 'Hockey');
INSERT INTO Deporte (id, deporte) VALUES (32, 'Hípica');
INSERT INTO Deporte (id, deporte) VALUES (33, 'Inscripción Torneo');
INSERT INTO Deporte (id, deporte) VALUES (34, 'Judo');
INSERT INTO Deporte (id, deporte) VALUES (35, 'Karate');
INSERT INTO Deporte (id, deporte) VALUES (36, 'Karting');
INSERT INTO Deporte (id, deporte) VALUES (37, 'Kitesurf');
INSERT INTO Deporte (id, deporte) VALUES (38, 'Kendo');
INSERT INTO Deporte (id, deporte) VALUES (39, 'KickBoxing');
INSERT INTO Deporte (id, deporte) VALUES (40, 'Kung Fu');
INSERT INTO Deporte (id, deporte) VALUES (41, 'Laser-tag');
INSERT INTO Deporte (id, deporte) VALUES (42, 'MMA');
INSERT INTO Deporte (id, deporte) VALUES (43, 'Meditación');
INSERT INTO Deporte (id, deporte) VALUES (44, 'Motociclismo');
INSERT INTO Deporte (id, deporte) VALUES (45, 'Natación');
INSERT INTO Deporte (id, deporte) VALUES (46, 'Padel');
```

```

INSERT INTO Deporte (id, deporte) VALUES (47, 'Paintball');
INSERT INTO Deporte (id, deporte) VALUES (48, 'Patinaje');
INSERT INTO Deporte (id, deporte) VALUES (49, 'Petanca');
INSERT INTO Deporte (id, deporte) VALUES (50, 'Pilates');
INSERT INTO Deporte (id, deporte) VALUES (51, 'Pingpong');
INSERT INTO Deporte (id, deporte) VALUES (52, 'Piragüismo');
INSERT INTO Deporte (id, deporte) VALUES (53, 'Rugby');
INSERT INTO Deporte (id, deporte) VALUES (54, 'Running');
INSERT INTO Deporte (id, deporte) VALUES (55, 'Senderismo');
INSERT INTO Deporte (id, deporte) VALUES (56, 'Skateboard');
INSERT INTO Deporte (id, deporte) VALUES (57, 'Ski');
INSERT INTO Deporte (id, deporte) VALUES (58, 'Snowboard');
INSERT INTO Deporte (id, deporte) VALUES (59, 'Spinning');
INSERT INTO Deporte (id, deporte) VALUES (60, 'Squash');
INSERT INTO Deporte (id, deporte) VALUES (61, 'Surf');
INSERT INTO Deporte (id, deporte) VALUES (62, 'Taekwondo');
INSERT INTO Deporte (id, deporte) VALUES (63, 'Tenis');
INSERT INTO Deporte (id, deporte) VALUES (64, 'Volley Playa');
INSERT INTO Deporte (id, deporte) VALUES (65, 'Volleyball');
INSERT INTO Deporte (id, deporte) VALUES (66, 'Waterpolo');
INSERT INTO Deporte (id, deporte) VALUES (67, 'Windsurf');
INSERT INTO Deporte (id, deporte) VALUES (68, 'Yoga');
INSERT INTO Deporte (id, deporte) VALUES (69, 'Otros');

```

## Países disponibles en la aplicación:

```

INSERT INTO Pais VALUES (1, 'España');

```

## Provincias disponibles en la aplicación:

```

INSERT INTO Provincia (id,provincia,pais_id) VALUES (1, 'Álava', 1);
INSERT INTO Provincia (id,provincia,pais_id) VALUES (2, 'Albacete', 1);
INSERT INTO Provincia (id,provincia,pais_id) VALUES (3, 'Alicante', 1);
INSERT INTO Provincia (id,provincia,pais_id) VALUES (4, 'Almería', 1);
INSERT INTO Provincia (id,provincia,pais_id) VALUES (5, 'Ávila', 1);
INSERT INTO Provincia (id,provincia,pais_id) VALUES (6, 'Badajoz', 1);
INSERT INTO Provincia (id,provincia,pais_id) VALUES (7, 'Illes Balears', 1);
INSERT INTO Provincia (id,provincia,pais_id) VALUES (8, 'Barcelona', 1);
INSERT INTO Provincia (id,provincia,pais_id) VALUES (9, 'Burgos', 1);
INSERT INTO Provincia (id,provincia,pais_id) VALUES (10, 'Cáceres', 1);
INSERT INTO Provincia (id,provincia,pais_id) VALUES (11, 'Cádiz', 1);
INSERT INTO Provincia (id,provincia,pais_id) VALUES (12, 'Castellón', 1);
INSERT INTO Provincia (id,provincia,pais_id) VALUES (13, 'Ciudad Real', 1);
INSERT INTO Provincia (id,provincia,pais_id) VALUES (14, 'Córdoba', 1);
INSERT INTO Provincia (id,provincia,pais_id) VALUES (15, 'A Coruña', 1);
INSERT INTO Provincia (id,provincia,pais_id) VALUES (16, 'Cuenca', 1);
INSERT INTO Provincia (id,provincia,pais_id) VALUES (17, 'Girona', 1);
INSERT INTO Provincia (id,provincia,pais_id) VALUES (18, 'Granada', 1);
INSERT INTO Provincia (id,provincia,pais_id) VALUES (19, 'Guadalajara', 1);
INSERT INTO Provincia (id,provincia,pais_id) VALUES (20, 'Guipúzcoa', 1);
INSERT INTO Provincia (id,provincia,pais_id) VALUES (21, 'Huelva', 1);
INSERT INTO Provincia (id,provincia,pais_id) VALUES (22, 'Huesca', 1);
INSERT INTO Provincia (id,provincia,pais_id) VALUES (23, 'Jaén', 1);
INSERT INTO Provincia (id,provincia,pais_id) VALUES (24, 'León', 1);
INSERT INTO Provincia (id,provincia,pais_id) VALUES (25, 'Lleida', 1);
INSERT INTO Provincia (id,provincia,pais_id) VALUES (26, 'La Rioja', 1);
INSERT INTO Provincia (id,provincia,pais_id) VALUES (27, 'Lugo', 1);
INSERT INTO Provincia (id,provincia,pais_id) VALUES (28, 'Madrid', 1);
INSERT INTO Provincia (id,provincia,pais_id) VALUES (29, 'Málaga', 1);
INSERT INTO Provincia (id,provincia,pais_id) VALUES (30, 'Murcia', 1);
INSERT INTO Provincia (id,provincia,pais_id) VALUES (31, 'Navarra', 1);
INSERT INTO Provincia (id,provincia,pais_id) VALUES (32, 'Ourense', 1);
INSERT INTO Provincia (id,provincia,pais_id) VALUES (33, 'Asturias', 1);
INSERT INTO Provincia (id,provincia,pais_id) VALUES (34, 'Palencia', 1);
INSERT INTO Provincia (id,provincia,pais_id) VALUES (35, 'Las Palmas', 1);
INSERT INTO Provincia (id,provincia,pais_id) VALUES (36, 'Pontevedra', 1);
INSERT INTO Provincia (id,provincia,pais_id) VALUES (37, 'Salamanca', 1);
INSERT INTO Provincia (id,provincia,pais_id) VALUES (38, 'Santa Cruz De Tenerife', 1);
INSERT INTO Provincia (id,provincia,pais_id) VALUES (39, 'Cantabria', 1);
INSERT INTO Provincia (id,provincia,pais_id) VALUES (40, 'Segovia', 1);
INSERT INTO Provincia (id,provincia,pais_id) VALUES (41, 'Sevilla', 1);

```

```

INSERT INTO Provincia (id,provincia,pais_id) VALUES (42, 'Soria', 1);
INSERT INTO Provincia (id,provincia,pais_id) VALUES (43, 'Tarragona', 1);
INSERT INTO Provincia (id,provincia,pais_id) VALUES (44, 'Teruel', 1);
INSERT INTO Provincia (id,provincia,pais_id) VALUES (45, 'Toledo', 1);
INSERT INTO Provincia (id,provincia,pais_id) VALUES (46, 'Valencia', 1);
INSERT INTO Provincia (id,provincia,pais_id) VALUES (47, 'Valladolid', 1);
INSERT INTO Provincia (id,provincia,pais_id) VALUES (48, 'Vizcaya', 1);
INSERT INTO Provincia (id,provincia,pais_id) VALUES (49, 'Zamora', 1);
INSERT INTO Provincia (id,provincia,pais_id) VALUES (50, 'Zaragoza', 1);
INSERT INTO Provincia (id,provincia,pais_id) VALUES (51, 'Ceuta', 1);
INSERT INTO Provincia (id,provincia,pais_id) VALUES (52, 'Melilla', 1);

```

Municipios y su ubicación GPS estimada disponibles en la aplicación. No se adjuntan todas las sentencias ya que se dispone de 8116 municipios y esto provocaría un gran aumento en la memoria del proyecto. Adjuntamos algunos municipios de ejemplo:

```

INSERT INTO Municipio (provincia_id, municipio, id, slug, latitudEstimada, longitudEstimada) VALUES(1, 'Alegria-Dulantzi', 1, 'alegra-dulantzi', 42.841492, -2.5135074);
INSERT INTO Municipio (provincia_id, municipio, id, slug, latitudEstimada, longitudEstimada) VALUES(1, 'Amurrio', 2, 'amurrio', 43.0526489, -3.0010217);
INSERT INTO Municipio (provincia_id, municipio, id, slug, latitudEstimada, longitudEstimada) VALUES(1, 'Añana', 3, 'aana', 42.802352, -2.982607);
INSERT INTO Municipio (provincia_id, municipio, id, slug, latitudEstimada, longitudEstimada) VALUES(1, 'Aramaio', 4, 'aramaio', 43.054, -2.566);
INSERT INTO Municipio (provincia_id, municipio, id, slug, latitudEstimada, longitudEstimada) VALUES(1, 'Armiñón', 5, 'armin', 42.7230453, -2.872574);
INSERT INTO Municipio (provincia_id, municipio, id, slug, latitudEstimada, longitudEstimada) VALUES(1, 'Arraia-Maeztu', 6, 'arraia-maeztu', 42.7398149, -2.4459801);
INSERT INTO Municipio (provincia_id, municipio, id, slug, latitudEstimada, longitudEstimada) VALUES(1, 'Arrazua-Ubarrundia', 7, 'arrazua-ubarrundia', 42.8902778, -2.6391667);
INSERT INTO Municipio (provincia_id, municipio, id, slug, latitudEstimada, longitudEstimada) VALUES(1, 'Artziniega', 8, 'artziniega', 43.1222011, -3.1282091);
INSERT INTO Municipio (provincia_id, municipio, id, slug, latitudEstimada, longitudEstimada) VALUES(1, 'Asparrena', 9, 'asparrena', 42.8956667, -2.321);
INSERT INTO Municipio (provincia_id, municipio, id, slug, latitudEstimada, longitudEstimada) VALUES(1, 'Ayala/Aiara', 10, 'ayalaaiara', 43.0833333, -3.0630556);
INSERT INTO Municipio (provincia_id, municipio, id, slug, latitudEstimada, longitudEstimada) VALUES(1, 'Baños de Ebro/Mañueta', 11, 'baos-de-ebromaueta', 42.5302903, -2.6791438);
INSERT INTO Municipio (provincia_id, municipio, id, slug, latitudEstimada, longitudEstimada) VALUES(1, 'Barrundia', 12, 'barrundia', 42.9167, -2.49182);
INSERT INTO Municipio (provincia_id, municipio, id, slug, latitudEstimada, longitudEstimada) VALUES(1, 'Berantevilla', 13, 'berantevilla', 42.6824495, -2.8604015);
INSERT INTO Municipio (provincia_id, municipio, id, slug, latitudEstimada, longitudEstimada) VALUES(1, 'Bernedo', 14, 'bernedo', 42.6266842, -2.4975323);
INSERT INTO Municipio (provincia_id, municipio, id, slug, latitudEstimada, longitudEstimada) VALUES(1, 'Campezo/Kanpezu', 15, 'campezokanpezu', 42.6919, -2.37);
INSERT INTO Municipio (provincia_id, municipio, id, slug, latitudEstimada, longitudEstimada) VALUES(1, 'Elburgo/Burgelu', 16, 'elburgoburgelu', 42.849564, -2.5447617);
INSERT INTO Municipio (provincia_id, municipio, id, slug, latitudEstimada, longitudEstimada) VALUES(1, 'Elciego', 17, 'elciego', 42.5150725, -2.618277);
INSERT INTO Municipio (provincia_id, municipio, id, slug, latitudEstimada, longitudEstimada) VALUES(1, 'Elvillar/Bilar', 18, 'elvillarbilar', 42.5704973, -2.5449153);
INSERT INTO Municipio (provincia_id, municipio, id, slug, latitudEstimada, longitudEstimada) VALUES(1, 'Erriberagoitia/Ribera Alta', 19, 'erriberagoitiaribera-alta', 42.8072222, -2.9175);
INSERT INTO Municipio (provincia_id, municipio, id, slug, latitudEstimada, longitudEstimada) VALUES(1, 'Harana/Valle de Arana', 20, 'haranavalle-de-arana', 42.7595704, -2.3194022);
INSERT INTO Municipio (provincia_id, municipio, id, slug, latitudEstimada, longitudEstimada) VALUES(1, 'Iruña Oka/Iruña de Oca', 21, 'irua-okairua-de-oca', 42.8191667, -2.8077778);
INSERT INTO Municipio (provincia_id, municipio, id, slug, latitudEstimada, longitudEstimada) VALUES(1, 'Iruraz-Gauna', 22, 'iruraz-gauna', 42.8220723, -2.4950428);
INSERT INTO Municipio (provincia_id, municipio, id, slug, latitudEstimada, longitudEstimada) VALUES(1, 'Kripan', 23, 'kripan', 42.5916664, -2.5160786);
INSERT INTO Municipio (provincia_id, municipio, id, slug, latitudEstimada, longitudEstimada) VALUES(1, 'Kuartango', 24, 'kuartango', 42.8713889, -2.8855556);
INSERT INTO Municipio (provincia_id, municipio, id, slug, latitudEstimada, longitudEstimada) VALUES(1, 'Labastida/Bastida', 25, 'labastidabastida', 42.5904214, -2.7931662);
INSERT INTO Municipio (provincia_id, municipio, id, slug, latitudEstimada, longitudEstimada) VALUES(1, 'Lagrán', 26, 'lagrn', 42.6263954, -2.5838202);
INSERT INTO Municipio (provincia_id, municipio, id, slug, latitudEstimada, longitudEstimada) VALUES(1, 'Laguardia', 27, 'laguardia', 42.5542775, -2.5842371);
INSERT INTO Municipio (provincia_id, municipio, id, slug, latitudEstimada, longitudEstimada) VALUES(1, 'Lanciego/Lantziego', 28, 'lanciegolantziego', 42.5626187, -2.5132036);

```

