Meet&Share:
APLICACIÓN WEB PARA HACER AMIGOS COMPARTIENDO GUSTOS Y AFICIONES

FERNANDO LÓPEZ CARRIÓN

TRABAJO DE FIN DE GRADO EN INGENIERÍA DEL SOFTWARE
FACULTAD DE INFORMÁTICA
UNIVERSIDAD COMPLUTENSE DE MADRID

Curso 2018/2019

Director: ANTONIO SARASA CABEZUELO

Agradecimientos

En primer lugar, me gustaría agradecerle a mi tutor, Antonio Sarasa Cabezuelo, el tiempo dedicado y la atención prestada. Sus consejos y anotaciones me han orientado desde el principio y me ha sido más fácil progresar estos meses en el trabajo.

También quiero agradecer a mis padres, mis hermanas y mi familia, la paciencia que han tenido conmigo en estos años de carrera, de esfuerzo y de superación. Su aliento ha sido fundamental en momentos de caída, y la confianza que depositan en mí y en mi trabajo, me han hecho llegar a conseguir este objetivo que me encuentro defendiendo.

Vine a esta casa con un sueño, y espero irme con un sueño cumplido.

Índice general

Resumen 1
Abstract 2
1. Introducción 3
1.1. Motivación 3
1.2. Objetivos 3
1.3. Estructura 4
1. Introduction 6
1.1. Motivation 6
1.2. Objectives 6
1.3. Structure 6
2. Estado del arte 8
2.1. Meetup 8
2.2. Fever 9
2.3. Geokeda 10
2.4. Timpik 11
3. Tecnologías empleadas 13
3.1. Herramientas para el cliente 13
3.1.1. HTML5 13
3.1.2. Bootstrap 13
3.1.3. CSS 13
3.1.4. Librerías de iconos 13
3.1.5. Javascript 14
3.1.6. JQuery 14
3.1.7. AJAX 14
3.1.8. Bootstrap Datatables 15
3.1.9. Moment.js 15
3.1.10. Mapbox 15
3.2. Herramientas de la parte del servidor 15
3.2.1. Spring 15
3.2.2. Maven 15
3.2.3. Thymeleaf 16
3.3. Base de datos: Firebase 16

3.4. Otras herramientas auxiliares 16
3.4.1. Git 16
3.4.2. Herramientas de desarrollo 17
3.4.3. Heroku 18
4. Especificación de requisitos 19
4.1. Diagramas de casos de uso 19
4.2. Casos de uso 21
4.2.1. CU Usuario no identificado 21
4.2.2. CU Usuario registrado 22
5. Modelo de datos 42
5.1. Modelo entidad-relación 42
5.2. Base de datos 42
5.2.1. Información de la aplicación 42
5.2.2. Usuarios 43
5.2.3. Actividades 47
5.2.4. Notificaciones 48
5.2.5. Mensajes 48
5.3. Contenido Multimedia 49
5.3.1. Usuarios 49
5.3.2. Actividades 50
5.4. Autenticación 50
6. Arquitectura de la aplicación 52
6.1. Capa de usuarios 52
6.2. Capa de presentación 53
6.3. Capa de negocio 53
6.4. Capa de integración 54
6.5. Capa de Recursos 54
7. Diseño de la aplicación 55
7.1. Aspectos de diseño 55
7.1.1. Visibilidad del estado del sistema 55
7.1.2. Adecuación entre el sistema y el mundo real 56
7.1.3. Libertad y control por el usuario 56
7.1.4. Consistencia y estándares 57

7.1.5. Prevención de errores 57
7.1.6. Reconocer mejor que recordar 59
7.1.7. Flexibilidad y eficiencia de uso 59
7.1.8. Estética y diseño minimalista 60
7.1.9. Ayudar a los usuarios a reconocer, diagnosticar y solucionar los errores 60
7.1.10. Ayuda y documentación 61
7.2. Funcionalidad principal 62
7.2.1. Inicio de sesión 62
7.2.2. Creación de usuarios 64
7.2.3. Buscar usuarios / actividades 67
7.2.4. Modificar perfil 69
7.2.5. Gestionar usuarios 75
7.2.6. Amigos 76
7.2.7. Crear actividades 79
7.2.8. Modificar información de la actividad 81
7.2.9. Inscribirse / desinscribirse en una actividad 83
7.2.10. Mapa de actividades 85
8. Conclusiones y trabajo futuro 87
8.1. Conclusiones 87
8.2. Trabajo futuro 87
8. Conclusions and future work 89
8.1. Conclusions 89
8.2. Future work 89
1. Anexo I: Guía de uso 94
1.1. Usuarios no identificados 94
1.1.1. Página principal 94
1.1.2. Registro 95
1.1.3. Iniciar sesión 96
1.2. Usuarios registrados 96
1.2.1. Buscador 96
1.2.2. Ver Perfil 97
1.2.3. Modificar perfil 98
1.2.4. Cerrar sesión 99

1.3. Administradores 99
1.3.1. Activar / desactivar usuarios 99
1.3.2. Dar de alta nuevos administradores 100
1.4. Personas y Empresas 100
1.4.1. Crear actividad 101
1.4.2. Modificar actividad 102
1.4.3. Cancelar actividad 103
1.4.4. Inscribirse / desinscribirse en una actividad 104
1.4.5. Valoraciones 104
1.4.6. Amigos 106
1.4.7. Compartir actividades 106
1.4.8. Notificaciones 107
1.4.9. Chat 108
2. Anexo II: Guía de instalación 109
2.1. Crear una nueva carpeta 109
2.2. Descomprimir el archivo comprimido 109
2.3. Comprobar puerto 8080 109
2.4. Ejecutar el proyecto 109
2.5. Despliegue de la aplicación 110
2.6. Advertencias 110

Índice de Figuras

Figura 1. Página principal de Meetup 9
Figura 2. Página principal de Fever 10
Figura 3. Página principal de Geokeda 11
Figura 4. Página de eventos de Timpik 12
Figura 5. Casos de uso de la aplicación 20
Figura 6. Casos de uso Usuario no autenticado 21
Figura 7. Casos de uso Usuario registrado 23
Figura 8. Casos de uso Usuario Adminstrador 27
Figura 9. Casos de uso Personas y Empresas 30
Figura 10. Modelo entidad-relación 42
Figura 11. Base de datos de la aplicación en Firebase Realtime Database 43
Figura 12. Ejemplo de colección de usuarios en Firebase Realtime Database 44
Figura 13. Estructura de Firebase Cloud Storage 49
Figura 14. Estructura de la carpeta "users" de Firebase Cloud Storage 49
Figura 15. Estrucutra de la carpeta de un usuario en Firebase Cloud Storage 49
Figura 16. Contenido de la carpeta "profilePicture" de un usuario en Firebase Cloud Storage 50
Figura 17. Estructura de la carpeta “activities” de Firebase Cloud Storage 50
Figura 18. Archivos almacenados en una actividad de Firebase Cloud Storage 50
Figura 19. Arquitectura de la aplicación 52
Figura 20. Arquitectura interna de la Capa de Presentación 53
Figura 21. Arquitectura interna de la Capa de Recursos 54
Figura 22. Vista de Inicio de sesión con la opción resaltada 55
Figura 23. Vista de una actividad que no corresponde con ningún menú 56
Figura 24. Categorías de las actividades identificadas mediante iconos 56
Figura 25. Mensaje de confirmación de baja junto con las opciones de confirmar y cancelar 57
Figura 26. Barra principal de la aplicación mostrando el logo, las notificaciones y el perfil 57
Figura 27. Barra principal de la aplicación visualizada en un dispositivo móvil 57
Figura 28. Validaciones al crear una actividad 58
Figura 29. Vista de ver actividad una vez ésta ha empezado 58
Figura 30. Filtro del mapa con diferentes iconos en función de la opción 59
Figura 31. Notificaciones y lista de actividades con enlaces a las entidades 59
Figura 32. Entidades con información relevante 60
Figura 33. Vista con iconos identificativos sin soporte escrito 60
Figura 34. Vista de mis actividades con texto de ayuda 61
Figura 35. Diagrama de clases de la aplicación 62
Figura 36. Página de inicio de sesión 63
Figura 37. Página de registro para Personas y Empresas 65
Figura 38. Página de registro para Administradores 65

Figura 39. Página de búsqueda de usuarios para Administradores 68
Figura 40. Página de búsqueda de usuarios y actividades para Personas y Empresas ... 68
Figura 41. Página para modificar el perfil 70
Figura 42. Modal para cambiar la imagen de perfil 72
Figura 43. Modal para cambiar la contraseña 73
Figura 44. Interfaz para cancelar la cuenta 74
Figura 45. Página para la gestión de usuarios de los Administradores 75
Figura 46. Modal para añadir un nuevo amigo 77
Figura 47. Página para aceptar / rechazar peticiones de amistad 78
Figura 48. Página para eliminar amigos 79
Figura 49. Página para crear una actividad 80
Figura 50. Página para modificar una actividad 82
Figura 51. Modal para añadir una imagen a una actividad 82
Figura 52. Página imagen añadida a una actividad 82
Figura 53. Página de una actividad a la que inscribirse 84
Figura 54. Página principal con el mapa de la aplicación 86
Figura 55. Página imagen añadida a una actividad 103

Índice de Tablas

Tabla 1. Atributos de un Administrador en Firebase Database 45
Tabla 2. Atributos de una Persona en Firebase Database 46
Tabla 3. Atributos de una Empresa en Firebase Database 47
Tabla 4. Atributos de una Actividad en Firebase Database 48
Tabla 5. Atributos de una Notificación en Firebase Database 48
Tabla 6. Atributos de un Mensaje en Firebase Database 49
Tabla 7. Atributos de Usuario en Firebase Auth 51
Tabla 8. Atributos pertenecientes a un usuario registrado 66
Tabla 9. Atributos pertenecientes una actividad 80

Índice de Capturas

Captura 1. Interfaz de la página principal 94
Captura 2. Interfaz de registro de personas y empresas 95
Captura 3. Interfaz de inicio de sesión 96
Captura 4. Interfaz de búsqueda de usuarios por parte de un Administrador 96
Captura 5. Interfaz búsqueda de usuarios realizada por una Persona o Empresa 97
Captura 6. Interfaz ver perfil de usuario 97
Captura 7. Interfaz para cambiar la imagen de perfil 98
Captura 8. Interfaz para modificar el perfil de un usuario 98
Captura 9. Interfaz para modificar la contraseña 99
Captura 10. Interfaz para cancelar la cuenta 99
Captura 11. Interfaz para cerrar la sesión 99
Captura 12. Interfaz para la gestión de usuarios de los Administradores 100
Captura 13. Interfaz de creación de nuevos administradores 100
Captura 14. Interfaz que muestra el botón para crear una actividad 101
Captura 15. Interfaz de creación de actividades 101
Captura 16. Interfaz para modificar una actividad 102
Captura 17. Interfaz para añadir una imagen a una actividad 103
Captura 18. Interfaz para cancelar una actividad propia 103
Captura 19. Interfaz para inscribirse / desinscribirse de una actividad 104
Captura 20. Interfaz donde se muestra el historial de actividades 104
Captura 21. Interfaz para valorar una actividad y a su administrador 105
Captura 22. Interfaz que muestra el ranking de actividades y usuarios 105
Captura 23. Interfaz para añadir un amigo 106
Captura 24. Interfaz para resolver una petición de amistad 106
Captura 25. Interfaz que muestra la lista de actividades creadas y registradas por un usuario 107
Captura 26. Interfaz para compartir actividades con otros usuarios 107
Captura 27. Interfaz de notificaciones 107
Captura 28. Interfaz que muestra cómo abrir una nueva conversación con un amigo . 108
Captura 29. Interfaz que muestra como enviar un mensaje 108
Captura 30. Estructura del proyecto ejecutable entregado 109
Captura 31. Gestionar permisos de la aplicación 110

Índice de Códigos

Código 1. firebase-auth.js Inicio de sesión 64
Código 2. MainController.java Registro 67
Código 3. UserServiceImp.java Buscar usuarios 69
Código 4. UserController.java Editar perfil 71
Código 5. editProfile.js Cambiar imagen de perfil 72
Código 6. UserController.java Modificar contraseña 73
Código 7. UserController.java Cancelar cuenta 74
Código 8. AdminController.java Reactivar usuarios 76
Código 9. FriendController.java Enviar petición de amistad 77
Código 10. FriendController.java Resolver petición de amistad 78
Código 11. FriendServiceImp.java Eliminar amigo 79
Código 12. ActivityController.java Crear actividad 81
Código 13. ActivityController.java Editar actividad 83
Código 14. ActivityController.java Inscribirse en una actividad 85

Resumen

En este trabajo se ha desarrollado una aplicación web para conectar a gente con gustos e intereses comunes que buscan compartir su tiempo libre y divertirse a través de eventos y actividades geolocalizadas.

Los usuarios registrados pueden crear e inscribirse en las diferentes actividades que ellos mismos proponen, encontrándose estas clasificadas en categorías para que los usuarios pueden filtrar en base a sus intereses, pudiendo de esta manera organizar eventos sociales, aprovechar ofertas de viajes, descubrir nuevos planes en la ciudad, hacer amigos y valorar su experiencia una vez hayan participado en estas actividades.

De esta manera, la aplicación ofrece a los usuarios una herramienta online para organizar su tiempo libre, permitiendo encontrar nuevos planes y eventos próximos a su ubicación.

Palabras clave

Aplicación web, Aplicación multiplataforma, Red social, Spring, Firebase

Abstract

This work has been developed as a web application to connect people with the same likes and common interests that seeks to share their free time and have fun through events and geolocated activities.

Registered users can create and register in activities that other users propose. These activities are classified into categories so that users can filter them based on their interests. This project allows users to organize social events, take advantage of travel offers, discover new plans in the city, make friends and rate the experience once they have participated in these activities.

By this way, the application offers users an online tool to organize their free time, allowing them to find new plans and events near their location.

Keywords

Web application, Multiplatform application, Social network, Spring, Firebase

1.Introducción

En este capítulo se explicará la motivación para desarrollar el trabajo, los objetivos planteados, y la estructura de la memoria.

1.1. Motivación

La realidad a la que se enfrentan las personas adultas (mayores de 18 años), es que dedican la mayor parte del tiempo a cumplir con sus obligaciones profesionales, dejando el tiempo restante que suele ser mínimo para desarrollar su vida social.

En este sentido, la misión de este proyecto pasa por entender que el tiempo es limitado y que el ser humano se siente en la necesidad de administrarlo; siendo una de las claves del éxito personal la capacidad de organización y gestión del mismo.

Por otro lado, la visión se enfocará en asegurarnos que el tiempo libre sea de calidad, ya que éste influye en el bienestar físico y mental de las personas.

Por estos motivos, queda claro que las personas necesitan organizar su tiempo libre, cumpliendo con la necesidad de sentirse realizados a la vez que comparten estos momentos con los demás.

1.2. Objetivos

Teniendo en cuenta lo anterior, se plantea como objetivo general el desarrollo de una herramienta que permita a los usuarios administrar su tiempo libre a la vez que interaccionan con otras personas.

Este objetivo se refina en los siguientes subobjetivos específicos:

- Utilizar las actividades o eventos como mecanismos de unión entre los usuarios
- Fomentar este mecanismo permitiendo a los propios usuarios definir las actividades que se organizan
- Clasificar estos planes según los intereses comunes de un grupo de personas
- Situar las actividades en una localización concreta, permitiendo a los usuarios especificar un punto de reunión
- Facilitar mecanismos de comunicación entre los usuarios que acudirán a estos eventos para que puedan mantenerse en contacto
- Proporcionar un sistema de feedback a los usuarios, para que puedan valorar y filtrar los eventos organizados

1.3. Estructura

En este capítulo se resumirá la estructura de la memoria, exponiendo las diferentes secciones en las que se divide el proyecto que trata de resolver el problema.

1. **Introducción**
   En esta sección se detalla la motivación que ha llevado a desarrollar este trabajo, los objetivos que se plantean, y la estructura de la propia memoria.

2. **Estado del arte**
   En esta sección se presentan las herramientas y aplicaciones que cumplen con una funcionalidad similar a la planteada.

3. **Tecnologías empleadas**
   En este capítulo se explican las tecnologías y herramientas que se utilizan en el trabajo, así como las librerías y frameworks empleados.

4. **Especificación de requisitos**
   En este capítulo se presentan los diferentes casos de uso que ofrece la aplicación, organizándolos según la funcionalidad de la que dispone cada uno de los actores.

5. **Modelo de datos**
   En este capítulo se explica el diseño y la organización de la base de datos, así como la información que contiene.

6. **Arquitectura de la aplicación**
   En este capítulo se expone la arquitectura del proyecto web, describiendo las diferentes capas en las que se divide.

7. **Diseño de la aplicación**
   En este capítulo se aborda el diseño de la herramienta, asociando cada interfaz gráfica con la funcionalidad que implementa.

8. **Conclusiones y trabajo futuro**
   En este capítulo se enumeran las conclusiones derivadas del desarrollo del proyecto, así como las diferentes mejoras y cambios que se podrían desarrollar en un futuro.

   Bibliografía
   En este capítulo se recogen los diferentes recursos externos a la aplicación que se han utilizado.

   Anexo I: Guía de uso
   En esta primera sección adicional, se desarrollan una serie de pautas a seguir para cumplir con la funcionalidad principal de la aplicación.

   Anexo II: Guía de instalación

En este segundo añadido, se detallan los diferentes pasos requeridos para la instalación y correcto funcionamiento de la herramienta.

1.Introduction

In this chapter will be explained the motivation to develop the project, the objectives, and the structure of the memory.

1.1. Motivation

Currently, adults (over 18 years) face the reality of spending most of their time satisfying their professional obligations, leaving the rest for improving their social life.

In this sense, the mission of this project is to understand that time is limited, and the human feels the need of administrate it; being organization skills one of the keys to reach their personal success.

On the other hand, the vision will focus on making sure that free time means quality, since it influences the physical and mental well-being of people.

For these reasons, it is clear that people need to organize their free time while feeling realized and sharing these moments with others.

1.2. Objectives

Taking into account the above, the aim is to develop an application that allows users to manage their free time while interacting with other people.

This objective is refined in the following sub-objectives:

- Use activities or events as mechanisms of union between users
- Promote this mechanism allowing the users themselves to define the activities that are organized
- Classify these plans according to the common interests of a group of people
- Place the activities in a specific location, allowing users to specify a meeting point
- Ease communication mechanisms among the users who will attend these events, so they can keep in touch
- Provide feedback system to users, so they can assess and filter the organized events

1.3. Structure

This chapter will summarize the structure of memory, exposing the different sections in which is divided.

1. **Introduction**

This section details the motivation that has led to develop this work, the objectives that are proposed, and the structure of the memory itself.

2. **State of art**
   Presents the tools and applications that fulfill a functionality similar to the one proposed.

3. **Used technologies**
   This chapter explains the technologies and tools used in the work, as well as the libraries and frameworks employed.

4. **Requirements specification**
   In this section the different cases of use offered by the application are exposed, organizing them according to the functionality available to each actor.

5. **Data model**
   This chapter explains the design and organization of the database, as well as the information it contains.

6. **Architecture of the application**
   In this section the architecture of the web project is exposed, describing the different layers in which it is divided.

7. **Application design**
   In this chapter the design of the tool is approached, associating each graphic interface with the functionality it implements.

8. **Conclusions and future work**
   This section lists the conclusions derived from the project's development, as well as the different improvements and changes that could be implemented in the future.

Bibliography
This chapter sum up the different external application's resources that have been employed.

Annex I: User guide
In this first additional section, a series of guidelines are been exposed in order to describe the main functionality of the application.

Annex II: Installation guide
In this second addition, the steps required for the installation and correct tools' operation are detailed.

2. Estado del arte

En este capítulo, se describen algunas de las herramientas que implementan funcionalidades parecidas a las desarrolladas en este trabajo.

2.1. Meetup

Es una red social multiplataforma que permite a sus miembros unirse a grupos creados por organizadores, con el objetivo de conectar personas con intereses en común [1] [2], teniendo un gran impacto en la sociedad [3].

La aplicación está centrada en la creación de grupos o *"meetups"*, cuyos administradores son los encargados de organizar los eventos, moderar las conversaciones y foros, realizar encuestas, etc. De esta manera, los usuarios pueden unirse a estos grupos en base a sus gustos y categorías. Esto puede verse en la página web oficial [4] mostrada en la Figura 1.

La herramienta cuenta a su vez con un sistema de amigos, por el cual los usuarios pueden ponerse en contacto con otra gente sin que sea necesario tener una actividad en común.

Pese a que el sitio web y las apps móviles son gratuitas, y no contienen publicidad; la empresa obtiene ingresos cobrando una membresía mensual a los organizadores de los eventos, oscilando estas entre los 8 y 15 $ (USD) dependiendo de las ofertas e impuestos aplicados.

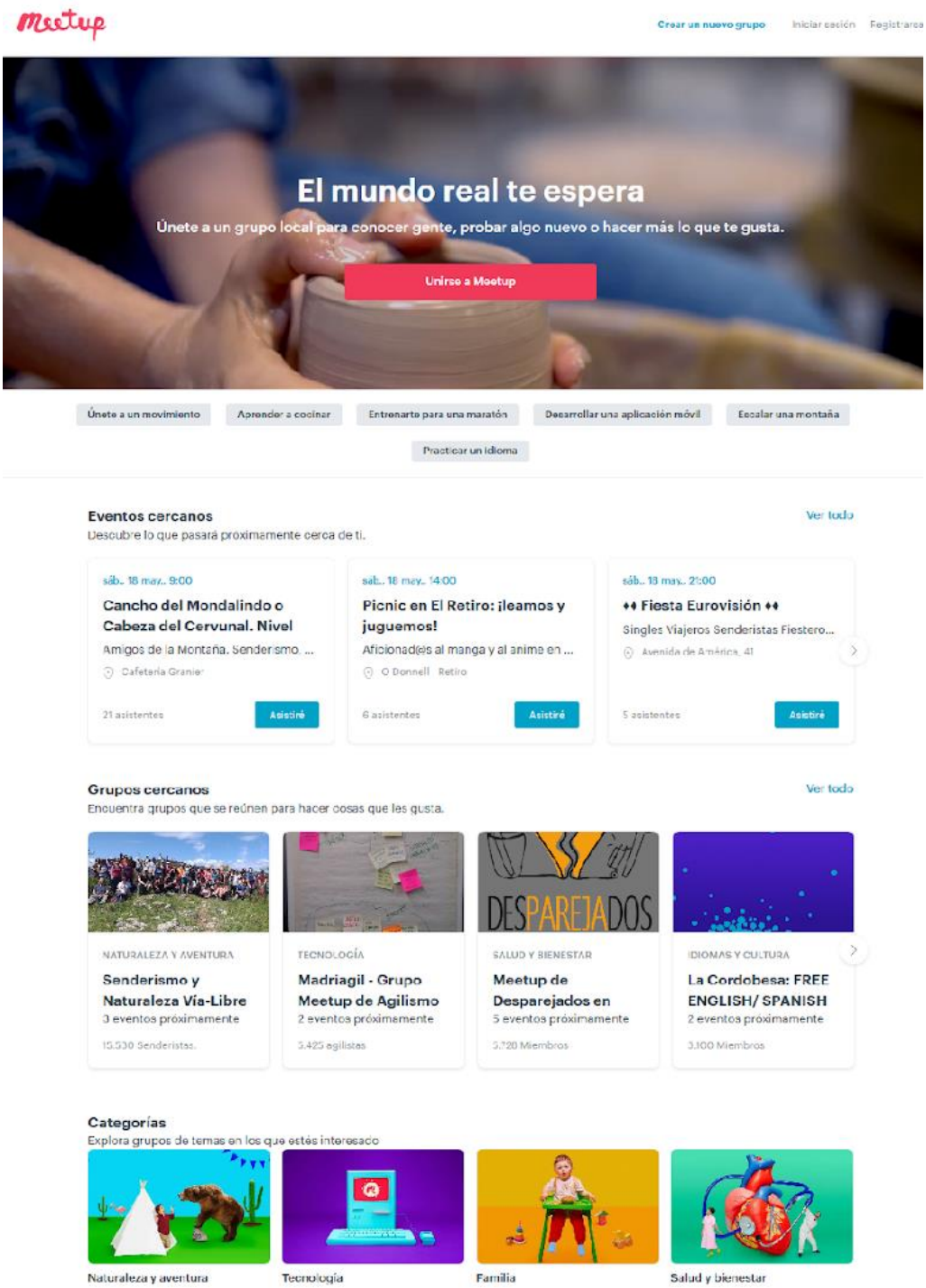


Figura 1. Página principal de Meetup

2.2. Fever

Se trata de una plataforma social cuyo objetivo principal es que los usuarios puedan aprovechar las diferentes ofertas, tickets y entradas que se publicitan; siendo líder en este sector [5] [6], pese a que requiere un gasto monetario por parte del usuario.

Esta aplicación como se puede apreciar en el sitio web oficial [7] de la Figura 2, destaca por su sencillez, eliminando la interacción entre los usuarios, permitiendo descubrir nuevos planes cercanos a una ubicación dada, y ofreciendo un gran catálogo de eventos categorizados y ordenados por fecha.

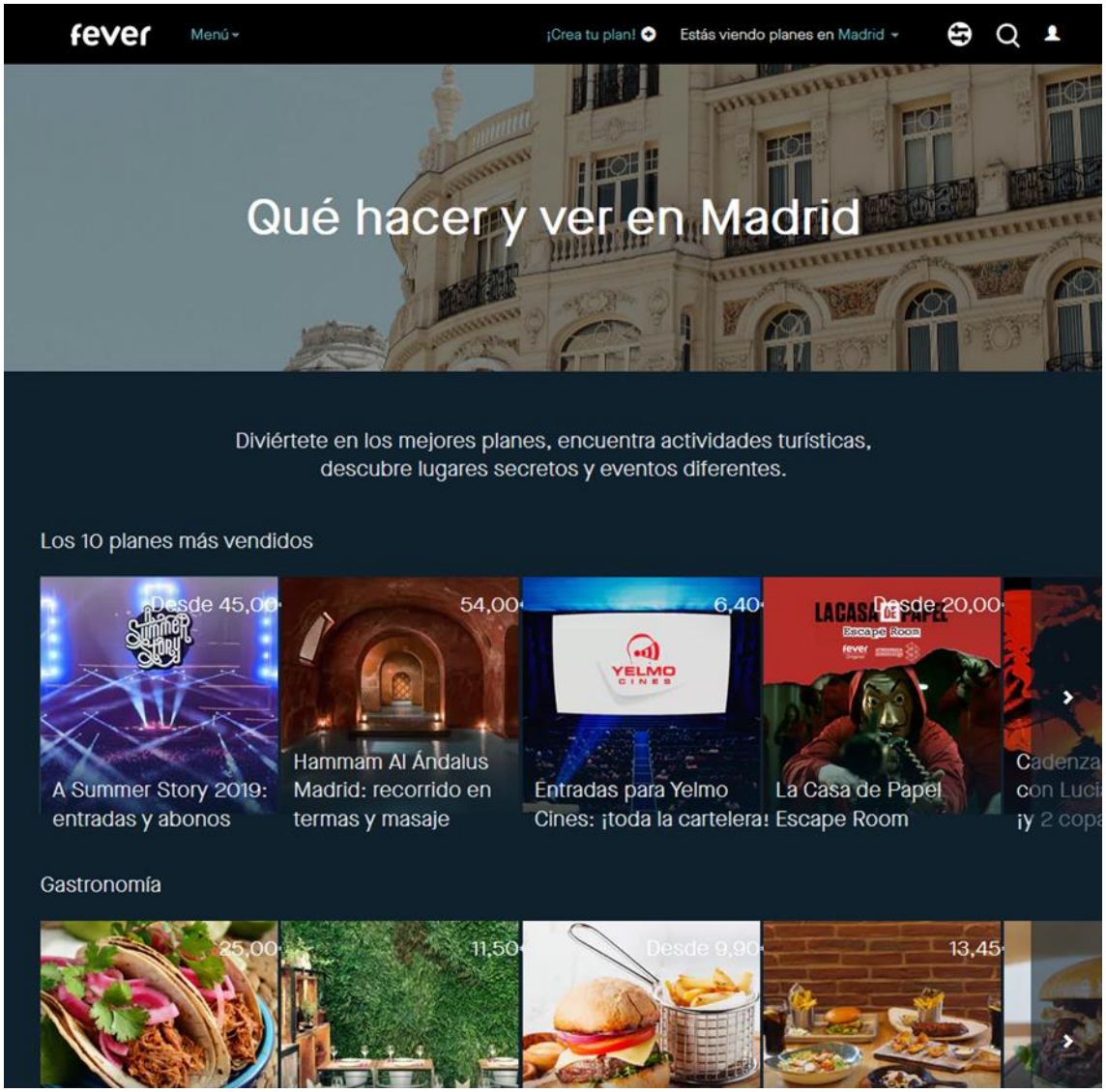


Figura 2. Página principal de Fever

2.3. Geokeda

Es una aplicación gratuita multiplataforma española orientada a reunir a personas para que compartan experiencias deportivas y culturales. El objetivo de la aplicación es facilitar la compartición de sus hobbies y aficiones entre los usuarios.

La plataforma permite crear eventos públicos conocidos como *"geokedadas"* que aparecerán en el calendario principal, y actividades exclusivas creadas por grupos donde solo pueden inscribirse sus integrantes. También ofrece tablones con

actividades populares, foros de iniciación, eventos informativos y guías de ayuda para hacer sentir lo más cómodo posible a los usuarios más nuevos o reservados.

Para conocer más acerca de la aplicación se puede consultar la siguiente reseña [8] o visitar el sitio oficial cuya página principal se muestra en la Figura 3 [9].

Figura 3. Página principal de Geokeda

2.4. Timpik

Es una red social [10] orientada a países de habla Hispana con un carácter profesional, cuyo objetivo es unir a deportistas y profesionales del deporte, y que actualmente funciona como la herramienta principal para organizar eventos deportivos en Madrid [11] [12].

Los deportistas pueden buscar partidos y eventos cercanos en un horario específico, y que además incluye una lista de espera por si se producen bajas de última hora (véase Figura 4). También incluye un foro para poder comentar el evento, y la opción de pagar online en caso de que sea necesario.

Así mismo, pueden añadirse nuevos amigos, consultar los logros obtenidos, o crear y participar en clubs para formar parte de pequeñas comunidades deportivas.

Los profesionales del deporte son los encargados de organizar los eventos que tendrán lugar en la aplicación, cuyo perfil está orientado a empresas y clubes de

deportes que necesiten una herramienta para gestionar los eventos que tendrán lugar en sus instalaciones.

Por último, cabe destacar que la empresa genera beneficios mediante el cobro de una comisión a los jugadores y clubes que participan en estos eventos, llegando a acuerdos comerciales con estos locales.

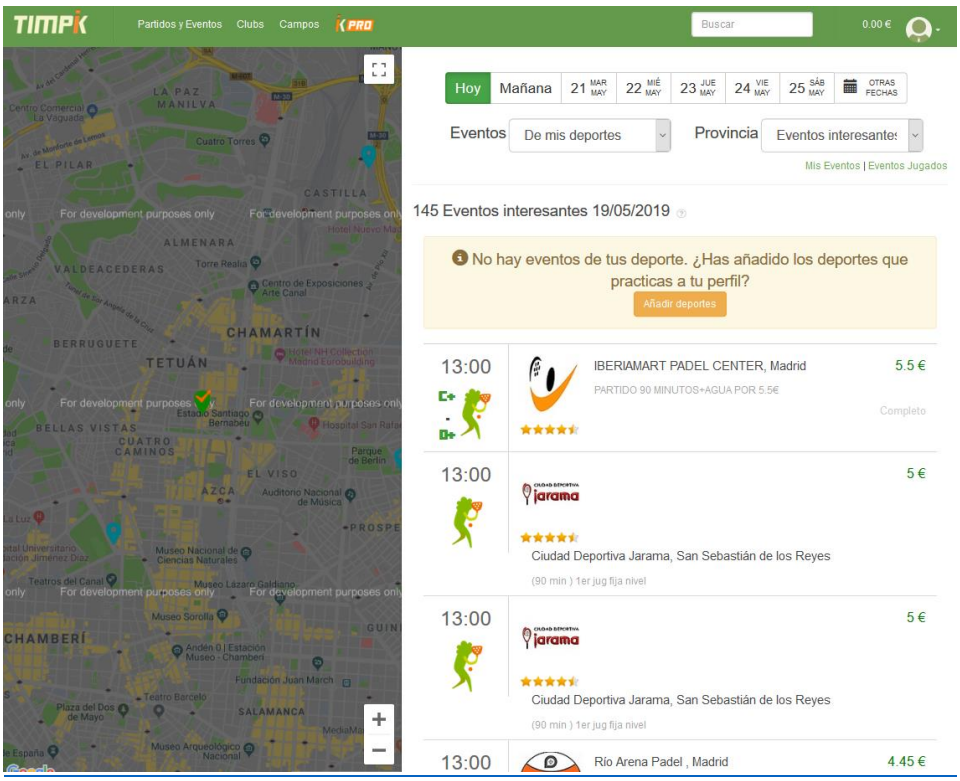


Figura 4. Página de eventos de Timpik

3. Tecnologías empleadas

En este capítulo se describen las tecnologías utilizadas para desarrollar el proyecto. La presentación se va a estructurar exponiendo las tecnologías empleadas en el lado del cliente, en el lado del servidor, y la base de datos. Así mismo, se comentarán otras herramientas adicionales utilizadas.

3.1. Herramientas para el cliente

3.1.1. HTML5

En el lado del cliente, se ha utilizado HTML5 [13] [14], el lenguaje de marcado empleado mundialmente para el desarrollo de páginas web. A través de este, se implementan los contenedores y contenidos de nuestra aplicación, utilizando etiquetas que denotan el comienzo y final de estos elementos.

3.1.2. Bootstrap

Es una biblioteca de herramientas web de código abierto ampliamente extendida, compatible con la mayoría de navegadores, y que contiene plantillas de diseño para el desarrollo de interfaces de usuario. Este framework [15] permite desarrollar aplicaciones web responsive que se adaptan a prácticamente cualquier tipo de dispositivo, aunque principalmente está orientado a la visualización en terminales móviles (mobile-first). Además, cuenta con un gran repertorio de ejemplos y documentación que ayudan al desarrollador a entender rápidamente el uso de la herramienta.

3.1.3. CSS

Es un lenguaje de diseño gráfico que permite crear hojas de estilo en cascada (Cascade Style Sheets [16]) que contienen reglas o propiedades de estilo (colores, fuentes, márgenes, etc.) aplicables al elemento especificado. Este lenguaje está diseñado para separar el contenido de la página web (escrito en HTML) de la forma en la que se presenta, reduciendo la complejidad, repetición de código, y permitiendo adaptar los documentos al dispositivo en el que se estén visualizando.

3.1.4. Librerías de iconos

Con el objetivo de que la aplicación sea los más amigable posible con el usuario, se han utilizado bibliotecas de iconos.

La primera es Font-Awesome [17], un framework de iconos compatible con la mayoría de navegadores y dispositivos que emplea imágenes vectoriales para representar objetos, símbolos y figuras. Está diseñado para funcionar junto con Bootstrap ofreciendo más de 300 iconos a incluir en sitios web mediante una simple etiqueta HTML.

En segundo lugar, se ha utilizado Material-Icons, una librería de iconos creada por Google bajo el concepto de Material-Design [18]. Esta forma de diseño ofrece una serie de pautas y normas que prometen resultados claros, sencillos, ordenados, llamativos, y cuya jerarquía se basa en el uso de sombras para indicar la posición en la que se encuentran los elementos en la interfaz.

3.1.5. Javascript

Se trata de un lenguaje de programación interpretado y orientado a objetos, utilizado ampliamente tanto en el lado del cliente como en el lado del servidor. Javascript [19] está basado en prototipos, que se utilizan para construir webs dinámicas, incorporar efectos, manejar eventos, procesar formularios, etc [20]. Constituye junto con HTML y CSS, los tres pilares fundamentales sobre los que se construyen la gran mayoría de sitios web en Internet.

3.1.6. JQuery

JQuery [21] [22] es una biblioteca multiplataforma de software libre y código abierto destinada principalmente a interactuar y manipular los elementos HTML DOM (representación en forma de árbol de todos los elementos de un sitio web). La razón de ser de esta herramienta es separar el código Javascript de los documentos HTML, ofreciendo funcionalidad de una manera breve, clara y compatible con todos los navegadores. También proporciona manejadores de eventos, animaciones y peticiones AJAX.

3.1.7. AJAX

AJAX [23] (Asynchronous Javascript And Xml) es una técnica empleada para desarrollar sitios web y aplicaciones interactivas, ofreciendo dinamismo entre las comunicaciones de cliente y servidor. Esta tecnología permite que una página web ya renderizada pueda solicitar nueva información al servidor en segundo plano, de manera asíncrona, y sin necesidad de recargarse, permitiendo implementar acciones complejas o workflows sin que se produzcan interrupciones.

3.1.8. Bootstrap Datatables

Se trata de un plug-in o complemento para Bootstrap y JQuery que permite extender de forma fácil estas herramientas, dando un nuevo soporte y funcionalidad a las tablas incorporadas en la aplicación. A través de esta librería [24] se consigue organizar eficazmente la información, filtrarla, paginarla, y hacerla seleccionable y ordenable.

3.1.9. Moment.js

Moment.js [25] es una librería Javascript compatible con navegadores y servidores, que a través de una sencilla interfaz facilita la lectura, validación, localización, manipulación y visualización de fechas en prácticamente cualquier tipo de formato.

3.1.10. Mapbox

Mapbox [26] es un framework multiplataforma de código abierto que integra mapas vectoriales interactivos y personalizables, siendo una de las principales alternativas a Google Maps al contar con gran parte de su funcionalidad accesible de manera gratuita. Esta herramienta permite además geolocalizar a los usuarios de la aplicación, realizar trazos, clusterizar ubicaciones, crear mapas de calor, etc.

3.2. Herramientas de la parte del servidor

3.2.1. Spring

Es un framework muy popular en el ámbito empresarial, de código abierto, e implementado en Java. Spring [27] tiene una estructura modular que proporciona las herramientas necesarias para construir sistemas web flexibles, escalables y seguros.

Las principales características que ofrece Spring son la implementación del patrón MVC orientado a POJOs (Plain Old Java Objects), la gestión de transacciones, el procesamiento de datos por lotes, el acceso a datos a través de JPA (Java Persistence API), y la gestión de dependencias a través de Maven.

3.2.2. Maven

Es una herramienta de código abierto perteneciente a Apache [28] e integrada con Spring, que proporciona una interfaz común para gestionar dependencias y construir proyectos Java configurables a través de ficheros XML [29]. Esta tecnología utiliza un POM (Project Object Model) para

estructurar el proyecto, describir sus dependencias, módulos, componentes, librerías, etc. Es capaz de gestionar proyectos software completos a través de ciclos de build que validan, compilan, testean, empaquetan, verifican, instalan y despliegan el código.

3.2.3. Thymeleaf

Es un gestor de plantillas [30] para Spring que permite modificar la interfaz que se devuelve al cliente, inyectar objetos Java como elementos DOM y Javascript, manipularlos, e incluso operar sobre ellos a través de una serie de interfaces que se proporcionan.

3.3. Base de datos: Firebase

Firebase es una plataforma de desarrollo multidispositivo basada en la nube y gestionada por Google [31] que proporciona a los desarrolladores una serie de APIs a partir de las cuáles éstos puedan organizar, guardar y sincronizar la información de las aplicaciones en tiempo real.

La herramienta ofrece un plan gratuito y una amplia documentación, permitiendo gestionar cualquier tipo de información o contenido, y proponiendo soluciones escalables (pues se implementa a través de una base de datos no relacional), dándole incluso al creador la posibilidad de generar dinero a través de los anuncios que la aplicación integre.

3.4. Otras herramientas auxiliares

A continuación, se detallarán las diferentes herramientas o plataformas que se han empleado para realizar el control de versiones, la implementación de la aplicación, y su correspondiente despliegue.

3.4.1. Git

Es uno de los principales sistemas de control de versiones más extendidos en el mundo, constituyendo un software libre muy potente orientado a proyectos con un gran número de archivos y fuentes de código. Fue creado con el objetivo de registrar los cambios en el software y coordinar el trabajo entre las personas que participan en los repositorios que alojan a las aplicaciones. Las principales características de Git [32] son la rapidez, el sistema de trabajo por branches o ramas, la gestión distribuida y el realmacenamiento periódico.

3.4.2. Herramientas de desarrollo

En esta subsección, se explicarán las diferentes herramientas utilizadas para realizar el proyecto:

3.4.2.1. Spring Tool Suite 3 (STS 3)

Se trata de un entorno de desarrollo integrado (IDE) de aplicaciones Spring gratuito basado en Eclipse [33], que ofrece soluciones documentadas, prácticas comunes utilizadas, soporte completo para aplicaciones web, integración con herramientas distribuidas y sistemas de control de versiones, mecanismos de comunicación asíncronos, gestión de beans, control de autorización y autenticación, así como asistencia para la creación y organización de estos proyectos software a través de Maven.

3.4.2.1. Visual Studio Code

Es un editor de código fuente gratuito y de código abierto creado por Microsoft [34], y es empleado fundamentalmente para desarrollar herramientas y aplicaciones web. Incluye soporte para la depuración, integración con Git y sistemas de control de versiones, resaltado de sintaxis y análisis inteligente del código. La herramienta a su vez cuenta con un sinfín de plug-ins y extensiones, siendo ampliamente personalizable, permitiendo cambiar el tema, asignar atajos de teclado, ajustar preferencias de visualización, etc. preferencias.

3.4.2.2. Android Studio

Es un IDE oficial para la integración y desarrollo de aplicaciones Android [35]. Se encuentra basado en Gradle (sistema de control de dependencias) e IntelliJ IDEA (editor de código similar a Eclipse), cuyo lenguaje de programación principal es Java.

Esta herramienta ofrece servicios de localización, optimización, refactorización, plantillas de diseño para interfaces de usuario, soporte con Google Cloud Messaging, además de permitir el diseño y emulación de apps móviles sobre una gran variedad de dispositivos.

3.4.2.3. Consola de Firebase

Se trata de una aplicación web online [36] que sirve de base y configuración para la plataforma Firebase antes mencionada. El

propósito de la herramienta consiste en administrar los diferentes complementos que proporciona la plataforma, ofreciendo interfaces y servicios que implementan la gestión de cuentas de usuario, autenticación, almacenamiento de datos y organización de cualquier tipo de contenido.

3.4.3. Heroku

Heroku [37] [38] es una plataforma como servicio (PaaS) que permite desplegar aplicaciones programadas en diferentes lenguajes de programación, y que pueden ser compiladas, ejecutadas y montadas a partir un repositorio Git. De esta manera, cada vez que se realiza un commit sobre la rama configurada, el sistema es capaz de automáticamente relanzar el proyecto para que los cambios tengan efecto en la versión desplegada. Esto permite a la aplicación, mantenerse siempre actualizada y accesible a través de cualquier navegador, sin necesidad de tener que configurar un servidor específico para poder alojarla.

4. Especificación de requisitos

En este capítulo, abordaremos los Casos de uso (CU) presentes, que se agruparán dependiendo de los actores.

4.1. Diagramas de casos de uso

Para presentar los casos de uso de forma organizada, se describirán primero los diferentes actores que interactúan con la aplicación. Éstos, crean todo el contenido que se sirve y definen los tipos de usuario que interactúan con la misma:

- **Usuario no autenticado**: persona anónima no registrada que carece de una cuenta para acceder a la funcionalidad completa que se ofrece.
- **Usuario registrado**: usuario inscrito en la aplicación que se encuentra con una sesión iniciada, habiendo introducido previamente un email y contraseña correspondientes con un usuario activo de base de datos.

Los usuarios registrados son clasificados a su vez en función del rol que éstos desempeñan en la aplicación, cumpliendo uno de los siguientes perfiles:

- Administrador: usuarios destinados a gestionar las cuentas de usuario registradas en la aplicación. Sobre ellos recae toda la responsabilidad, pues tienen la capacidad de activar y desactivar a los usuarios, pudiendo a su vez dar de alta a nuevos administradores.
- Persona: Público objetivo destinatario de la aplicación. Su actividad se centra en generar el contenido y dar vida a la aplicación a través de la interacción con otros usuarios, y la inscripción y creación de nuevas actividades.
- Empresa: Grupo de usuarios cuyo objetivo principal está basado en la gestión y organización de nuevas de actividades, buscando que éstas llegan a la mayor cantidad de público posible, y compitiendo a su vez por los puestos más altos de los rankings. De esta forma las empresas conseguirán mostrar al conjunto de posibles clientes la calidad de los servicios que éstas ofretan.

A continuación, se especifican todos los casos de uso que componen la funcionalidad completa de la aplicación, agrupados según el tipo de usuario (véase en la Figura 5).

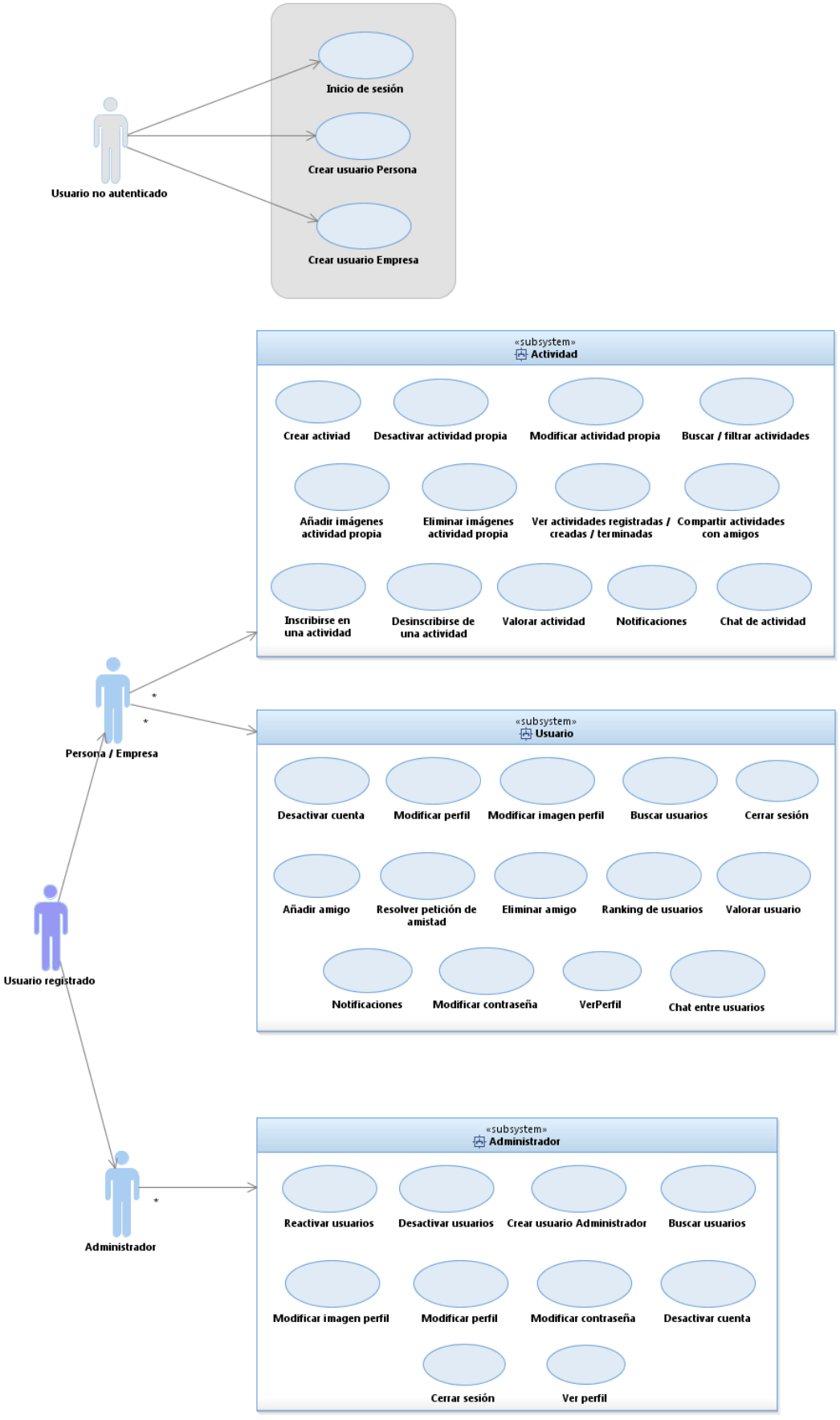


Figura 5. Casos de uso de la aplicación

4.2. Casos de uso

En esta subsección se abordarán con detalle cada uno de los casos de uso, agrupados según los actores que intervienen y que componen la aplicación.

4.2.1. CU Usuario no identificado

Estos son los usuarios anónimo no registrados que por tanto no pueden acceder a la funcionalidad completa de la herramienta, limitándose a los casos de uso descritos en la Figura 6.

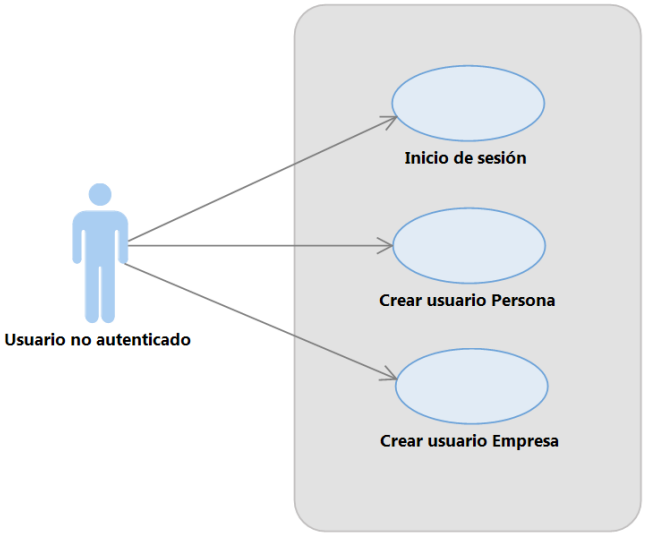


Figura 6. Casos de uso Usuario no autenticado

Caso de uso	Descripción
Identificador	InicioSesion
Nombre	Inicio de sesión
Actores	Usuario no identificado
Visión general	Permite al usuario acceder a la aplicación
Condiciones previas	El usuario debe encontrarse activo y registrado en la aplicación
Resultado	Redirige a la página de inicio del usuario
CU Asociados	CrearUsuarioPersona, CrearUsuarioEmpresa, CrearUsuarioAdministrador, CerrarSesion
Entradas	Email y contraseña del usuario
Salida	Notificación de inicio de sesión
Índice de utilización	Alto

Caso de uso	Descripción
Identificador	CrearUsuarioPersona
Nombre	Crear un usuario con el rol de persona
Actores	Usuario no identificado
Visión general	Permite al usuario crear una cuenta en el sistema
Condiciones previas	El usuario no debe encontrarse ya registrado
Resultado	Crea la cuenta e inicia sesión en la aplicación
CU Asociados	IniciarSesion, CrearUsuarioEmpresa, CrearUsuarioAdministrador, DesactivarCuenta, CerrarSesion
Entradas	Datos de la persona a dar de alta
Salida	Notificación de inicio de sesión
Índice de utilización	Medio

Caso de uso	Descripción
Identificador	CrearUsuarioEmpresa
Nombre	Crear un usuario con el rol de empresa
Actores	Usuario no identificado
Visión general	Permite al usuario crear una cuenta en el sistema
Condiciones previas	El usuario no debe encontrarse ya registrado
Resultado	Crea la cuenta e inicia sesión en la aplicación
CU Asociados	IniciarSesion, CrearUsuarioPersona, CrearUsuarioAdministrador, DesactivarCuenta, CerrarSesion
Entradas	Datos de la empresa a dar de alta
Salida	Notificación de inicio de sesión
Índice de utilización	Medio

4.2.2. CU Usuario registrado

En el caso de los usuarios registrados, ya forman parte de la aplicación, por lo que pueden acceder a la funcionalidad completa que se recoge en la Figura 7.

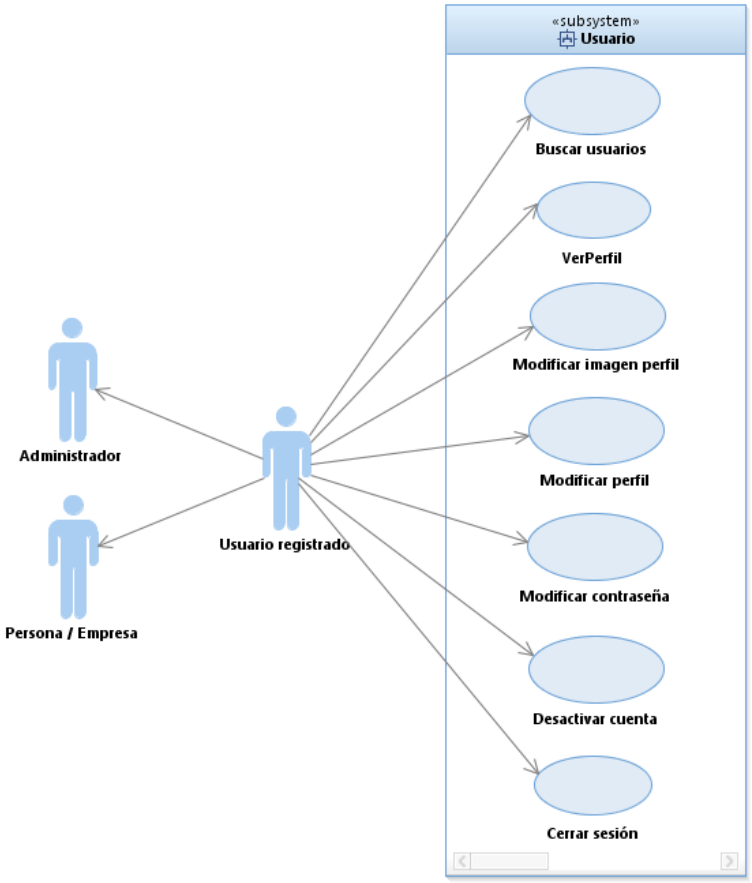


Figura 7. Casos de uso Usuario registrado

Caso de uso	Descripción
Identificador	BuscarUsuarios
Nombre	Buscar usuarios
Actores	Usuario registrado
Visión general	Permite encontrar al usuario cuyo email o nombre sea similar al texto especificado
Condiciones previas	El texto de búsqueda no puede encontrarse vacío
Resultado	Devuelve los resultados encontrados
CU Asociados	BuscarActividad
Entradas	Texto de búsqueda
Salida	Resultados
Índice de utilización	Muy alto

Caso de uso	Descripción
Identificador	VerPerfil
Nombre	Ver perfil
Actores	Usuario registrado
Visión general	Permite al usuario visualizar su información de perfil
Condiciones previas	Ninguna
Resultado	Muestra los datos del usuario
CU Asociados	ModificarImagenPerfil , ModificarPerfil, ModificarContraseña
Entradas	Identificador del usuario
Salida	Ninguna
Índice de utilización	Muy alto

Caso de uso	Descripción
Identificador	ModificarImagenPerfil
Nombre	Modificar imagen de perfil
Actores	Usuario registrado
Visión general	Permite al usuario modificar la imagen de perfil actual
Condiciones previas	Ninguna
Resultado	Modifica la imagen de perfil actual
CU Asociados	VerPerfil, ModificarPerfil, ModificarContraseña
Entradas	Imagen nueva
Salida	Se actualiza la foto de perfil dinámicamente
Índice de utilización	Medio

Caso de uso	Descripción
Identificador	ModificarPerfil
Nombre	Modificar perfil
Actores	Usuario registrado
Visión general	Permite al usuario modificar la información de perfil
Condiciones previas	Ninguna
Resultado	Modifica los datos del usuario
CU Asociados	VerPerfil, ModificarImagenPerfil, ModificarContraseña
Entradas	Datos del usuario a modificar
Salida	Notificación de cambios guardados
Índice de utilización	Medio

Caso de uso	Descripción
Identificador	ModificarContraseña
Nombre	Modificar contraseña
Actores	Usuario registrado
Visión general	Permite al usuario modificar la contraseña de inicio de sesión
Condiciones previas	Ninguna
Resultado	Modifica la contraseña del usuario
CU Asociados	VerPerfil, ModificarImagenPerfil, ModificarPerfil
Entradas	Contraseña antigua y contraseña nueva
Salida	Notificación de contraseña cambiada
Índice de utilización	Bajo

Caso de uso	Descripción
Identificador	DesactivarCuenta
Nombre	Desactivar cuenta
Actores	Usuario registrado
Visión general	Permite al usuario cancelar la cuenta y cerrar la sesión
Condiciones previas	Ninguna
Resultado	Desactiva la cuenta y cierra la sesión
CU Asociados	CrearUsuarioAdministrador, CrearUsuarioPersona, CrearUsuarioEmpresa, CerrarSesion
Entradas	Confirmación escrita del usuario para desactivar la cuenta
Salida	Notificación de cuenta desactivada
Índice de utilización	Bajo

Caso de uso	Descripción
Identificador	CerrarSesion
Nombre	Cerrar sesión
Actores	Usuario registrado
Visión general	Permite al usuario cerrar la sesión y abandonar la aplicación
Condiciones previas	El usuario debe encontrarse con la sesión iniciada
Resultado	Redirige a la página de principal de la aplicación
CU Asociados	IniciarSesion
Entradas	Ninguna
Salida	Notificación de cierre de sesión
Índice de utilización	Medio

4.2.2.1. CU Administrador

Los administradores son los usuarios registrados encargados de gestionar las cuentas de usuario de la aplicación a través de las funcionalidades presentadas en la Figura 8.

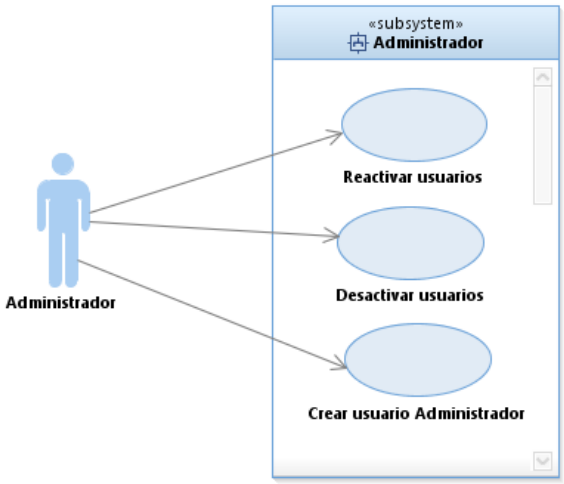


Figura 8. Casos de uso Usuario Adminstrador

Caso de uso	Descripción
Identificador	ReactivarUsuarios
Nombre	Reactivar usuarios
Actores	Adminstrador
Visión general	Permite al administrador volver a activar las cuentas de usuario seleccionadas
Condiciones previas	Los usuarios seleccionados deben encontrarse desactivados
Resultado	Vuelve a activar las cuentas de usuario seleccionadas, permitiendo de nuevo a éstos el acceso a la aplicación
CU Asociados	DesactivarUsuarios
Entradas	Lista de identificadores de los usuarios seleccionados
Salida	Tabla de usuarios actualizada con los usuarios reactivados
Índice de utilización	Bajo

Caso de uso	Descripción
Identificador	DesactivarUsuarios
Nombre	Desactivar usuarios
Actores	Adminstrador
Visión general	Permite al administrador desactivar las cuentas de usuario seleccionadas
Condiciones previas	Los usuarios seleccionados deben encontrarse activados
Resultado	Da de baja las cuentas de usuario seleccionadas, impidiendo a estos volver a iniciar sesión en la aplicación
CU Asociados	ReactivarUsuarios
Entradas	Lista de identificadores de los usuarios seleccionados
Salida	Tabla de usuarios actualizada con los usuarios desactivados
Índice de utilización	Bajo

Caso de uso	Descripción
Identificador	CrearUsuarioAdministrador
Nombre	Crear un usuario con el rol de administrador
Actores	Administrador
Visión general	Permite al usuario dar de alta una nueva cuenta de administrador en el sistema
Condiciones previas	El usuario no debe encontrarse ya registrado
Resultado	Crea la cuenta de usuario
CU Asociados	Iniciar Sesion, CrearUsuarioPersona, CrearUsuarioEmpresa, DesactivarCuenta, CerrarSesion
Entradas	Datos del administrador a dar de alta
Salida	Notificación de administrador creado
Índice de utilización	Bajo

4.2.2.2. CU Persona y Empresa

Estos actores comparten funcionalidad, siendo distinguibles a través de los atributos guardados sobre cada uno en la base datos.

A continuación, se muestra el diagrama de casos de uso que indica la funcionalidad accesible para estos actores (Figura 9).

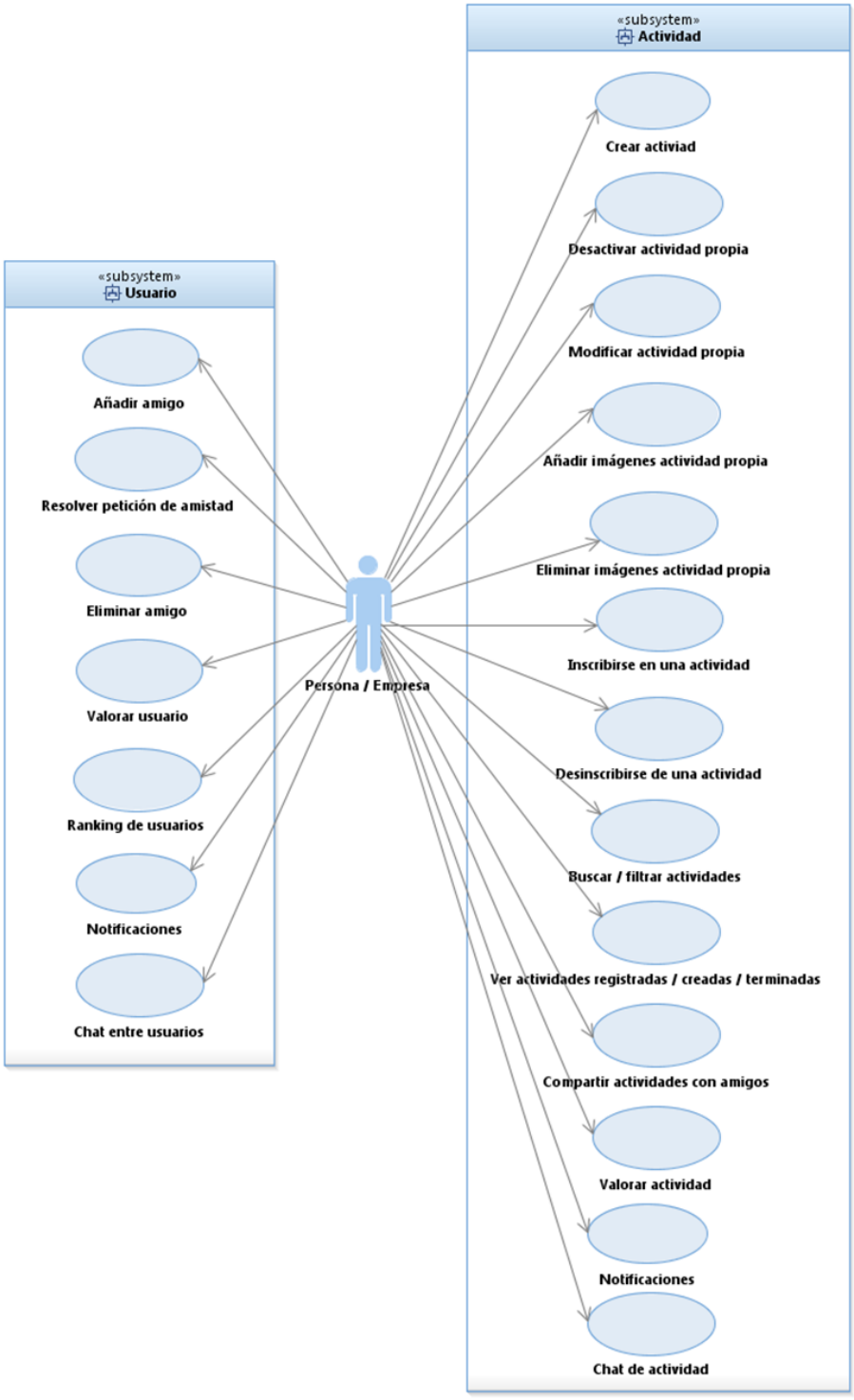


Figura 9. Casos de uso Personas y Empresas

4.2.2.2.1. Módulo de usuarios

En esta subsección se tratan los casos de uso relacionados con el propio usuario.

Caso de uso	Descripción
Identificador	AñadirAmigo
Nombre	Añadir amigo
Actores	Persona / Empresa
Visión general	Permite al usuario enviar una petición de amistad a otro usuario
Condiciones previas	Los usuarios no deben ser amigos
Resultado	Envía una petición de amistad con el texto indicado al usuario objetivo
CU Asociados	ResolverPeticionAmistad, EliminarAmigo
Entradas	Identificador del usuario al que se le envía la petición y mensaje opcional
Salida	Notificación petición de amistad enviada
Índice de utilización	Alto

Caso de uso	Descripción
Identificador	ResolverPeticionAmistad
Nombre	Resolver petición de amistad
Actores	Persona / Empresa
Visión general	Permite responder la petición de amistad enviada por un usuario
Condiciones previas	Los usuarios no deben ser amigos
Resultado	Elimina la petición del sistema, y en caso de que sea aceptada, añade a los usuarios a sus respectivas listas de amigos
CU Asociados	AñadirAmigo, EliminarAmigo
Entradas	Lista de identificadores de los usuarios seleccionados
Salida	En caso de aceptarla se notifica al usuario que envió la petición
Índice de utilización	Alto

Caso de uso	Descripción
Identificador	EliminarAmigo
Nombre	Eliminar amigo
Actores	Persona / Empresa
Visión general	Permite al usuario eliminar a un amigo
Condiciones previas	Los usuarios deben se amigos
Resultado	Elimina al usuario indicado de la lista de amigos
CU Asociados	AñadirAmigo, ResolverPeticionAmistad
Entradas	Identificador del usuario a eliminar
Salida	Notificación usuario eliminado
Índice de utilización	Bajo

Caso de uso	Descripción
Identificador	ValorarUsuario
Nombre	Valorar usuario
Actores	Persona / Empresa
Visión general	Permite al usuario valorar al administrador de una actividad
Condiciones previas	El usuario debe haber participado en al menos una actividad del usuario objetivo, pudiendo realizar la valoración siempre y cuando la actividad se encuentre activa y finalizada
Resultado	Permite al usuario valorar al administrador de una actividad una vez esta se encuentre finalizada, pudiendo dejar a su vez un comentario opcional
CU Asociados	ValorarActividad, RankingUsuarios, RankingActividades
Entradas	Identificador del administrador de la actividad, valoración y comentario redactado por el usuario
Salida	Muestra la valoración en la lista de valoraciones, accesible tanto al visualizar la actividad finalizada como al acceder al perfil del usuario
Índice de utilización	Medio

Caso de uso	Descripción
Identificador	RankingUsuarios
Nombre	Ranking de usuarios
Actores	Persona / Empresa
Visión general	Permite al usuario visualizar la lista de los 10 usuarios más votados de este momento
Condiciones previas	Ninguna
Resultado	Muestra el top 10 de usuarios más valorados en la aplicación
CU Asociados	ValorarUsuario, ValorarActividad, RankingActividades
Entradas	Ninguna
Salida	Top 10 de usuarios ordenados por valoración
Índice de utilización	Medio

Caso de uso	Descripción
Identificador	Notificaciones
Nombre	Notificaciones de usuario
Actores	Persona / Empresa
Visión general	Permite al usuario ver y eliminar las notificaciones que se producen por la acción de un usuario
Condiciones previas	Ninguna
Resultado	Muestra las notificaciones pendientes del usuario, pudiendo este a su vez marcarlas como leídas
CU Asociados	Ninguno
Entradas	Ninguna
Salida	Lista de notificaciones pendientes
Índice de utilización	Muy alto

Caso de uso	Descripción
Identificador	ChatUsuarios
Nombre	Chat entre usuarios
Actores	Persona / Empresa
Visión general	Permite al usuario interactuar con otros a través del envío de mensajes
Condiciones previas	Ninguna
Resultado	Muestra la conversación existente entre ambos usuarios, permitiendo el envío de nuevos mensajes
CU Asociados	ChatActividades
Entradas	Mensajes de texto
Salida	Conversación entre usuarios
Índice de utilización	Muy alto

4.2.2.2.2. Módulo de actividades

En este epígrafe se desarrollan los casos uso que involucran a una actividad con la que una persona o empresa interactúa.

Caso de uso	Descripción
Identificador	CrearActividad
Nombre	Crear Actividad
Actores	Persona / Empresa
Visión general	Permite al usuario crear una nueva actividad
Condiciones previas	Ninguna
Resultado	Crea la actividad y redirige a la página de Ver actividad
CU Asociados	DesactivarActividad, ModificarActividad
Entradas	Datos de la actividad a dar de alta
Salida	Notificación de actividad creada
Índice de utilización	Muy alto

Caso de uso	Descripción
Identificador	DesactivarActividad
Nombre	Desactivar Actividad propia
Actores	Persona / Empresa
Visión general	Permite al usuario desactivar una actividad creada
Condiciones previas	La actividad especificada debe pertenecer al usuario, estar activa y no haber empezado todavía
Resultado	Desactiva la actividad y notifica a los usuarios inscritos
CU Asociados	CrearActividad, ModificarActividad
Entradas	Identificador de la actividad a desactivar
Salida	Notificación de actividad desactivada
Índice de utilización	Medio

Caso de uso	Descripción
Identificador	ModificarActividad
Nombre	Modificar Actividad propia
Actores	Persona / Empresa
Visión general	Permite al usuario modificar una actividad creada
Condiciones previas	La actividad especificada debe pertenecer al usuario, estar activa, y no finalizada
Resultado	Modifica la actividad y notifica a los usuarios del cambio
CU Asociados	CrearActividad, DesactivarActividad
Entradas	Datos de la actividad a modificar
Salida	Notificación de actividad modificada
Índice de utilización	Medio

Caso de uso	Descripción
Identificador	AñadirImagenesActividad
Nombre	Añadir imágenes a una actividad propia
Actores	Persona / Empresa
Visión general	Permite al usuario añadir una imagen al carrusel de fotografías de la actividad seleccionada
Condiciones previas	La actividad debe pertenecer al usuario, encontrarse activa y no finalizada
Resultado	Añade la nueva imagen a la actividad
CU Asociados	ModificarImagenPerfil, EliminarImagenesActividad
Entradas	Imagen nueva
Salida	Imagen añadida al carrusel de fotos de la actividad
Índice de utilización	Alto

Caso de uso	Descripción
Identificador	EliminarImagenesActividad
Nombre	Eliminar imágenes en una actividad propia
Actores	Persona / Empresa
Visión general	Permite al usuario eliminar una imagen existente de una actividad creada
Condiciones previas	La actividad especificada debe pertenecer al usuario, estar activa, no haber finalizado, y la foto debe existir.
Resultado	Elimina la imagen indicada del carrusel de fotografías ded la actividad
CU Asociados	ModificarImagenPerfil, AñadirImagenesActividad
Entradas	Datos de la actividad a dar de alta
Salida	Notificación de actividad creada
Índice de utilización	Medio

Caso de uso	Descripción
Identificador	InscribirseActividad
Nombre	Inscribirse en una actividad
Actores	Persona / Empresa
Visión general	Permite al usuario inscribirse en una nueva actividad
Condiciones previas	El usuario no debe estar inscrito en la actividad, ni ser el administrador; y la actividad debe encontrarse activa y sin empezar
Resultado	Inscribe al usuario en la actividad indicada
CU Asociados	DesinscribirseActividad
Entradas	Identificador de la actividad a inscribirse
Salida	Notificación de inscripción
Índice de utilización	Alto

Caso de uso	Descripción
Identificador	DesinscribirseActividad
Nombre	Desinscribirse de una actividad
Actores	Persona / Empresa
Visión general	Permite al usuario desinscribirse de una actividad
Condiciones previas	El usuario debe estar inscrito en la actividad, no debe ser el administrador; y la actividad debe encontrarse activa y sin empezar
Resultado	Desinscribe al usuario de la actividad indicada
CU Asociados	InscribirseActividad
Entradas	Identificador de la actividad a desinscribirse
Salida	Notificación de desinscripción
Índice de utilización	Medio

Caso de uso	Descripción
Identificador	BuscarFiltrarActividades
Nombre	Buscar actividades / Filtrar actividades en el mapa
Actores	Persona / Empresa
Visión general	Permite al usuario buscar / filtrar las actividades mostradas
Condiciones previas	Ninguna
Resultado	Se muestran los resultados obtenidos a partir del criterio de búsqueda
CU Asociados	BuscarUsuarios
Entradas	Nombre de la actividad en el caso de la búsqueda. Fecha de inicio, radio en km, plazas libres o categorías seleccionadas en el caso de tratarse del filtro alojado en el mapa.
Salida	Notificación de actividad creada
Índice de utilización	Muy alto

Caso de uso	Descripción
Identificador	VerActividadesRegistradasCreadasTerminadas
Nombre	Ver actividades registradas / creadas / terminadas
Actores	Persona / Empresa
Visión general	Permite al usuario ver las actividades registradas, creadas o terminadas, siendo accesibles desde el panel de actividades o el menú de historial respectivamente
Condiciones previas	Ninguna
Resultado	Muestra la lista de actividades según el criterio seleccionado
CU Asociados	Ninguno
Entradas	Ninguna
Salida	Lista de actividades
Índice de utilización	Muy alto

Caso de uso	Descripción
Identificador	CompartirActividadesAmigos
Nombre	Compartir actividades con amigos
Actores	Persona / Empresa
Visión general	Permite al usuario compartir una actividad con los amigos seleccionados
Condiciones previas	Los usuarios seleccionados deben ser amigos de la persona o empresa que se encuentre con la sesión iniciada
Resultado	Comparte la actividad con los usuarios seleccionados
CU Asociados	Notificaciones
Entradas	Lista de identificadores de los usuarios, y actividad a compartir
Salida	Notificación actividad compartida
Índice de utilización	Alto

Caso de uso	Descripción
Identificador	ValorarActividad
Nombre	Valorar actividad
Actores	Persona / Empresa
Visión general	Permite al usuario valorar una actividad en la que haya participado
Condiciones previas	El usuario debe haber participado en la actividad, la actividad debe estar terminada, y el usuario no puede ser el administrador del evento.
Resultado	Permite al usuario valorar una actividad finalizada, pudiendo dejar a su vez un comentario
CU Asociados	ValorarUsuario, RankingUsuarios, RankingActividades
Entradas	Identificador del administrador de la actividad, valoración y comentario redactado por el usuario
Salida	Muestra la valoración en la lista de valoraciones, accesible cuando se ven los detalles de la actividad finalizada
Índice de utilización	Medio

Caso de uso	Descripción
Identificador	RankingActividades
Nombre	Ranking de actividades
Actores	Persona / Empresa
Visión general	Permite al usuario visualizar la lista de las 10 actividades más votadas de este momento
Condiciones previas	Ninguna
Resultado	Muestra el top 10 de actividades más valoradas en la aplicación
CU Asociados	ValorarUsuario, ValorarActividad, RankingActividades
Entradas	Ninguna
Salida	Top 10 de usuarios ordenados por valoración
Índice de utilización	Medio

Caso de uso	Descripción
Identificador	Notificaciones
Nombre	Notificaciones de actividad
Actores	Persona / Empresa
Visión general	Permite al usuario ver y eliminar las notificaciones que se producen por la acción de una actividad
Condiciones previas	Ninguna
Resultado	Muestra las notificaciones pendientes del usuario, pudiendo este a su vez marcarlas como leídas
CU Asociados	Ninguno
Entradas	Ninguna
Salida	Lista de notificaciones pendientes
Índice de utilización	Muy alto

Caso de uso	Descripción
Identificador	ChatActividades
Nombre	Chat entre usuarios de una actividad
Actores	Persona / Empresa
Visión general	Permite a los usuarios inscritos en una actividad interactuar entre sí.
Condiciones previas	Los usuarios deben estar inscritos en la actividad
Resultado	Muestra la conversación existente, permitiendo el envío de nuevos mensajes
CU Asociados	ChatUsuarios
Entradas	Mensajes de texto
Salida	Conversación entre usuarios
Índice de utilización	Muy alto

5.Modelo de datos

En este capítulo se explica la organización y estructura de la base de datos de la aplicación.

5.1. Modelo entidad-relación

Para llevar a cabo la persistencia, se adjunta un esquema conceptual de la base de datos, especificando las entidades que aparecen en la aplicación, y sus relaciones (Figura 10).

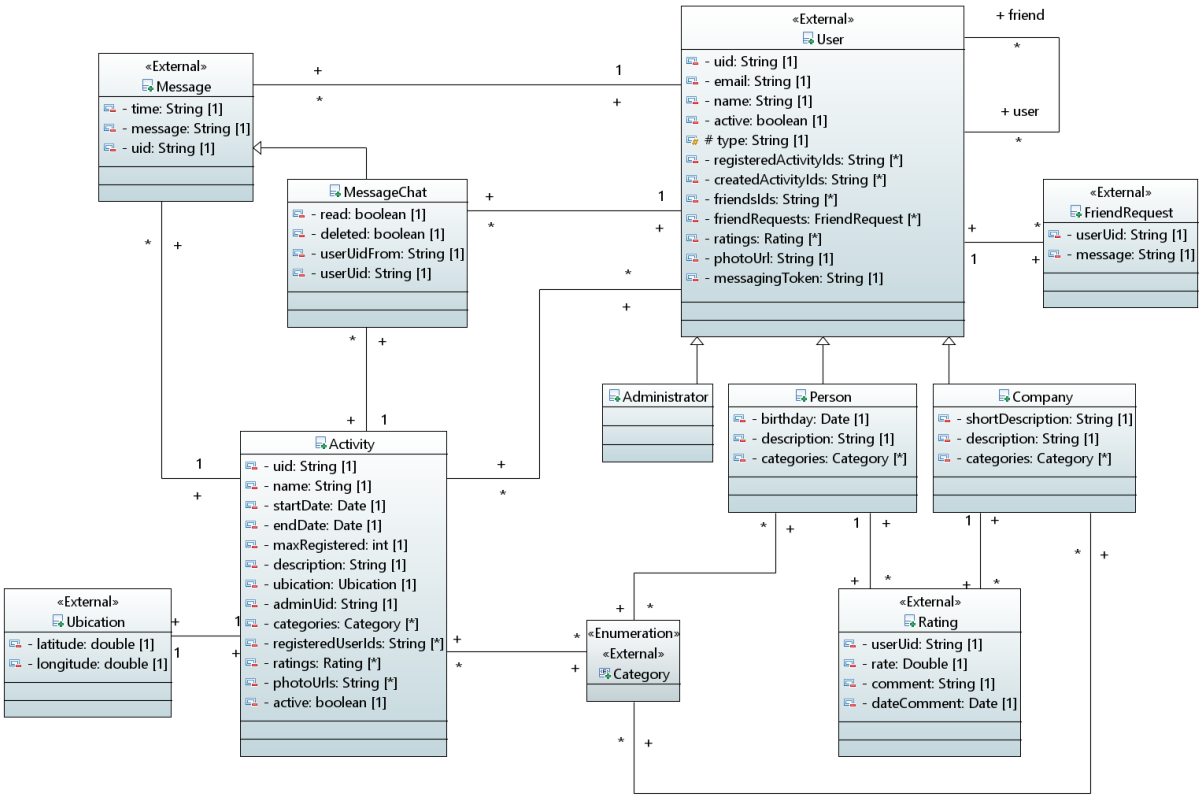


Figura 10. Modelo entidad-relación

5.2. Base de datos

Para implementar el modelo entidad relación se utilizarán diferentes clases de entidades, componiendo éstas la base de datos de la plataforma. Para ello se procederá a clasificar éstas en función del tipo de contenido que almacenan.

5.2.1. Información de la aplicación

Dado que se trata de una base de datos no relacional, no existen tablas al uso y el contenido de estas no tiene por qué tener una estructura fija.

Concretamente la base de datos se estructura mediante objetos en formato JSON.

Debido a esto, es el desarrollador es el encargado de diseñar la base de datos y mantenerla en un estado coherente, de manera que al realizar las diferentes consultas no se comprometa la información almacenada.

En este caso, se utiliza la herramienta Firebase Realtime Database para almacenar en colecciones la información referente a los usuarios, actividades, notificaciones y mensajes, siguiendo la estructura mostrada en la Figura 11.

Figura 11. Base de datos de la aplicación en Firebase Realtime Database

5.2.2. Usuarios

La aplicación distingue 3 tipos de usuarios: Administradores, Personas y Empresas.

Debido a que se trata de una base de datos no relacional, la información de cada usuario no sigue siempre una estructura de datos fija, sino que sólo se guardan los atributos correspondientes en función del tipo.

Colección de usuarios (“users”): Contiene la información referente a los distintas clases de usuarios que interactúan con la aplicación, encontrándose cada una de las entidades indexadas a través de un uid (identificador único). A continuación, en la Figura 12 se muestra un ejemplo de cómo se estructuran los atributos de cada uno de estos tipos de usuarios.

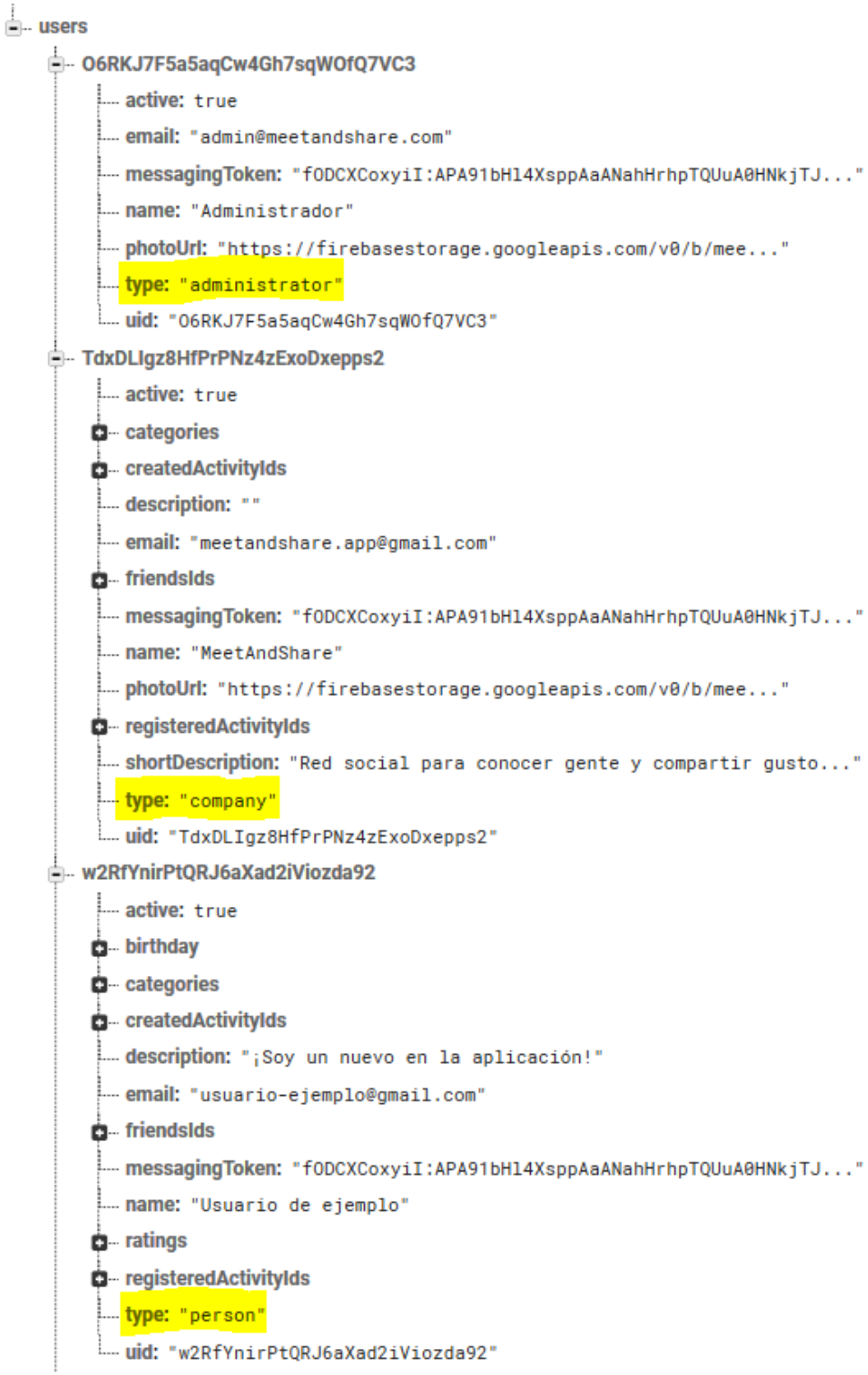


Figura 12. Ejemplo de colección de usuarios en Firebase Realtime Database

5.2.2.1. Administradores

Como se ha explicado anteriormente estos usuarios son los encargados de administrar el resto de cuentas, siendo los únicos capaces de dar de alta nuevos usuarios con este perfil.

A continuación, se muestra un ejemplo de esta entidad (véase Tabla 1), y se describe cada uno de sus atributos:

Atributos	Descripción
active	Campo que indica si el usuario está activo
email (único)	Correo electrónico del usuario
messagingToken	Token que habilita al usuario para poder enviar y recibir tanto notificaciones como mensajes.
name	Nombre del usuario
photoUrl	Imagen de perfil del usuario
type (*"administrator"*)	Campo que indica el tipo de usuario
uid	Identificador proporcionado por Firebase Auth al registrar un usuario

Tabla 1. Atributos de un Administrador en Firebase Database

5.2.2.2. Personas

Constituyen los usuarios registrados cuyo objetivo es conocer gente y compartir sus intereses a través de las actividades que se proponen. A continuación, se detallan los atributos de los que se componen (véase Tabla 2):

Atributos	Descripción
active	Campo booleano que indica si el usuario está activo
birthday	Fecha de nacimiento
categories	Categorías seleccionadas por el usuario, estas reflejan sus gustos y aficiones.
createdActivities	Lista con los uids de las actividades creadas por el usuario
description	Descripción del usuario
email (único)	Correo electrónico del usuario
friendIds	Lista con los uids de los amigos del usuario
friendRequests	Lista con las peticiones de amistad pendientes que el usuario ha recibido.

messagingToken	Token que habilita al usuario para poder enviar y recibir tanto notificaciones como mensajes.
name	Nombre del usuario
photoUrl	Imagen de perfil del usuario
ratings	Lista con las diferentes valoraciones escritas por los usuarios que se encuentran inscritos en alguna de las actividades propuestas por éste.
registeredActivities	Lista con los uids de las actividades en las que el usuario se ha inscrito (también aparecen las actividades creadas ya que los usuarios siempre se encuentran inscritos en sus propias actividades).
type (*"person"*)	Campo que indica el tipo de usuario
uid	Identificador proporcionado por Firebase Auth al registrar un usuario

Tabla 2. Atributos de una Persona en Firebase Database

5.2.2.3. Empresas

Son los usuarios registrados que buscan publicitar sus actividades a la mayor cantidad de público posible. A continuación se exponen las diferentes propiedades de las que están compuestas estas entidades (véase Tabla 3).

Atributos	Descripción
active	Campo booleano que indica si el usuario está activo
categories	Categorías seleccionadas por el usuario, estas reflejan sus gustos y aficiones.
createdActivities	Lista con los uids de las actividades creadas por el usuario
description	Descripción del usuario
email (único)	Correo electrónico del usuario
friendIds	Lista con los uids de los amigos del usuario
friendRequests	Lista con las peticiones de amistad pendientes que el usuario ha recibido.
messagingToken	Token que habilita al usuario para poder enviar y recibir tanto notificaciones como mensajes.
name	Nombre del usuario
photoUrl	Imagen de perfil del usuario

ratings	Lista con las diferentes valoraciones escritas por los usuarios que se encuentran inscritos en alguna de las actividades propuestas por éste.
registeredActivities	Lista con los uids de las actividades en las que el usuario se ha inscrito.
shortDescription	Breve frase que sirve de introducción para describir de manera pormenorizada a la empresa.
type (*"company"*)	Campo que indica el tipo de usuario
uid	Identificador proporcionado por Firebase Auth al registrar un usuario

Tabla 3. Atributos de una Empresa en Firebase Database

5.2.3. Actividades

Son los eventos organizados en la aplicación, estos son propuestos por personas o empresas, se encuentran localizados en el mapa y constituyen un punto a través del cual los usuarios inscritos pueden comenzar a conocerse.

Colección de actividades ("activities"): Guarda la información sobre las actividades propuestas por los usuarios. Cada entidad se indexa a través de un uid con los siguientes atributos (véase Tabla 4):

Atributos	Descripción
active	Campo booleano que indica si la actividad está activa
adminUid	Identificador del usuario que creó la actividad
categories	De manera análoga a los usuarios, alberga las categorías en las que se puede incluir la actividad.
description	Descripción de la actividad
endDate	Fecha y hora de fin de la actividad
maxRegistered	Número máximo de usuarios que pueden inscribirs**e** en la actividad.
name	Nombre de la actividad
photoUrls	Lista con las urls de las imágenes de la actividad
ratings	Lista con las diferentes valoraciones escritas por los usuarios que se encontraban inscritos en la actividad cuando esta finalizó
registeredUserIds	Lista con los uids de los usuarios inscritos en la actividad
startDate	Fecha y hora de inicio de la actividad

ubication	Sitio donde tiene lugar la actividad. Se encuentra expresado en coordenadas para poder ubicar la actividad en el mapa en todo momento.
uid	Identificador proporcionado por Firebase Realtime Database

Tabla 4. Atributos de una Actividad en Firebase Database

5.2.4. Notificaciones

Colección de notificaciones ("notifications"): Lista de notificaciones pendientes de cada usuario. Cada entidad se indexa a través del uid del usuario al que pertenecen estos avisos, albergando estos los siguientes datos (véase Tabla 5).

Atributos	Descripción
message	Texto que se mostrará al usuario cuando aparezca la notificación
uid	Identificador proporcionado por Firebase Realtime Database
time	Momento en el que se produjo el aviso

Tabla 5. Atributos de una Notificación en Firebase Database

5.2.5. Mensajes

Colección de mensajes ("messages"): Lista de los distintos mensajes enviados por los usuarios y que pueden tener como destino una actividad, u otro usuario. En el caso de los usuarios, se ubica la conversación bajo el uid concatenado de éstos, mientras que las actividades emplean directamente el identificador. A continuación, se detallan los atributos de los que se compone esta entidad (véase Tabla 6).

Atributos	Descripción
deleted	Campo booleano que indica si el mensaje ha sido borrado
message	Texto que se mostrará al cliente cuando aparezca la notificación.
read	Campo booleano que indica si el mensaje ha sido leído
time	Momento en el que se produjo el aviso
uid	Identificador proporcionado por Firebase Realtime Database
userUid	Identificador del usuario al que va dirigido el mensaje, en el caso de ser null se trata de un mensaje de actividad

userUidForm	Identificador del usuario emisor del mensaje

Tabla 6. Atributos de un Mensaje en Firebase Database

5.3. Contenido Multimedia

En este caso, se emplea Firebase Cloud Storage para almacenar todo el contenido que quiera publicar el usuario, bien sea modificar su foto de perfil, o subir nuevas imágenes a sus actividades.

En esto caso los ficheros son gestionados a través de carpetas, como podemos observar en la Figura 13. De esta manera existe una por cada entidad de la que se desea guardar la información.

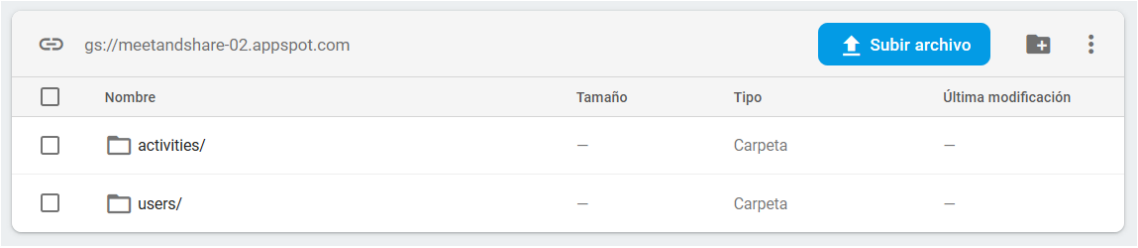


Figura 13. Estructura de Firebase Cloud Storage

5.3.1. Usuarios

El directorio de usuarios se organiza a través de subcarpetas que reciben como nombre el uid del usuario sobre el que se guardará la información (Figura 14). A su vez dentro de cada directorio, se almacena en la carpeta “profilePictures” la imagen de perfil del usuario correspondiente, distinguiendo ésta del resto de posibles archivos que fueran a alojarse en un futuro (véase Figuras 15 y 16).

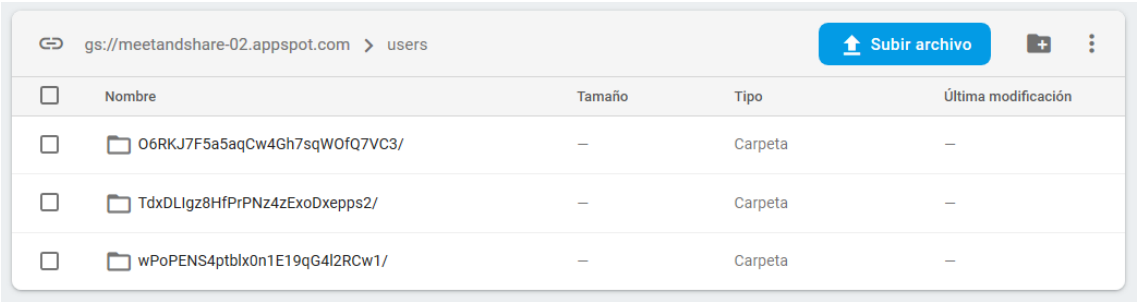


Figura 14. Estructura de la carpeta "users" de Firebase Cloud Storage

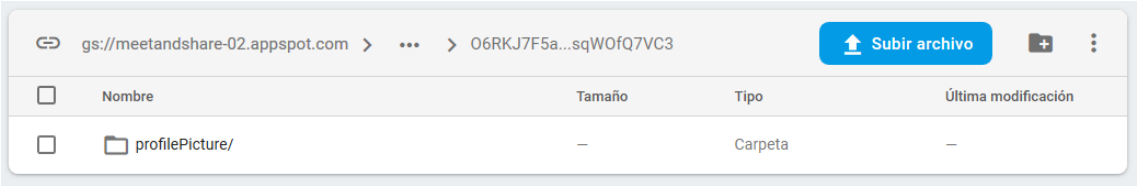


Figura 15. Estrucutra de la carpeta de un usuario en Firebase Cloud Storage

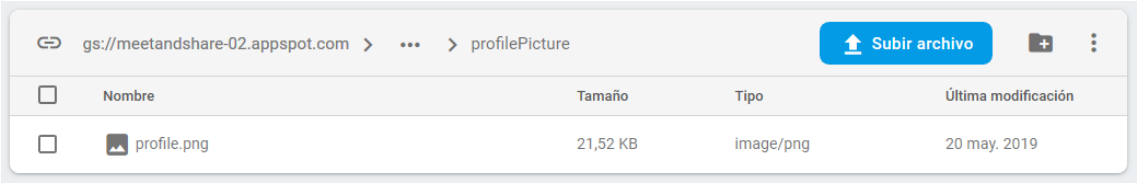


Figura 16. Contenido de la carpeta "profilePicture" de un usuario en Firebase Cloud Storage

5.3.2. Actividades

El contenido multimedia de las actividades se estructura de igual forma que el de usuarios. Como se aprecia en la Figura 17, se crea una carpeta por cada una de las actividades existentes, asignando a cada una el identificador del evento correspondiente. A su vez, las imágenes almacenadas reciben como nombre la fecha de subida (en formato "yyyy-mm-dd-hh:mm:ss:ms"), garantizando que nunca existirán dos fotos con el mismo nombre puesto que existe un cierto delay, y en ningún momento se ofrece la funcionalidad de subir más de una imagen a la vez (véase Figura 18).

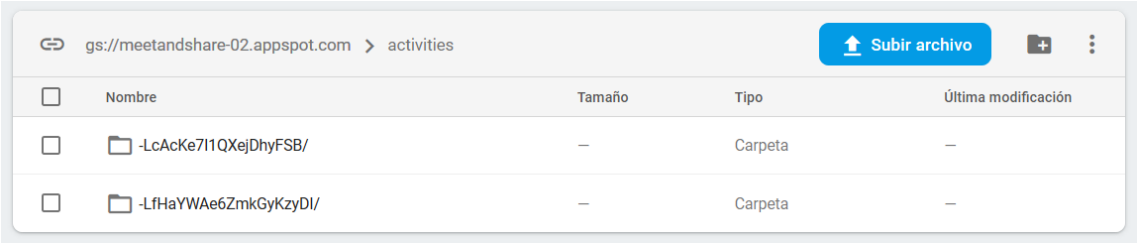


Figura 17. Estructura de la carpeta "activities" de Firebase Cloud Storage

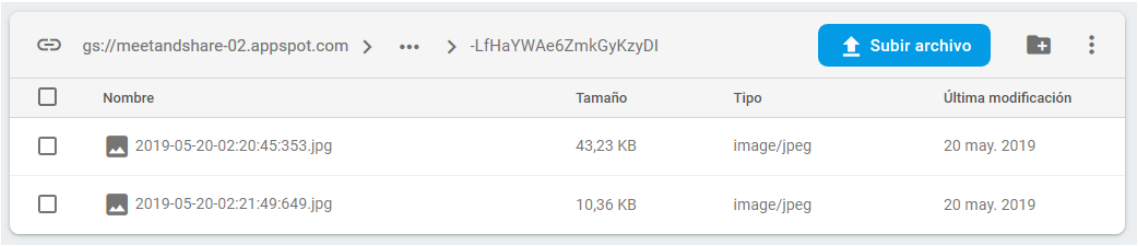


Figura 18. Archivos almacenados en una actividad de Firebase Cloud Storage

5.4. Autenticación

Implementada mediante el uso de Firebase Auth, esta herramienta nos permite crear una fila por cada usuario registrado, siendo distinguibles mediante un identificador único autogenerado, y un correo electrónico (véase Tabla 7).

Atributos	Descripción
Identificador (único)	Email del usuario

Proveedores	Entidades o elementos a través de los cuales se provee el inicio de sesión, en este caso se ha utilizado únicamente el proveedor de correo electrónico.
Fecha de creación (autogenerado)	Fecha en que el usuario se registró en la aplicación
Inicio de sesión (autogestionado)	Fecha de último inicio de sesión
Activo	Campo booleano interno que se gestiona a través de la API Java de Firebase Realtime Database y que indica si el usuario se encuentra habilitado y tiene permiso para acceder a la aplicación.
UID de usuario (autogenerado y único)	Clave primaria que corresponde con el uid empleado para mapear a los usuarios en la colección correspondiente.

Tabla 7. Atributos de Usuario en Firebase Auth

6. Arquitectura de la aplicación

En este capítulo se explica como ha sido desarrollada la aplicación, qué arquitectura sigue y de qué se encarga cada una de sus partes.

Para llevar a cabo este proyecto se ha utilizado una **arquitectura de cliente y servidor**. Siendo el cliente el encargado de servir la interfaz al usuario, renderizar los mapas, gestionar el contenido multimedia, autenticar al usuario, y mandar la información al servidor; y éste último necesario para procesar los datos e implementar la lógica de la aplicación.

La herramienta también sigue el esquema **MVC** (Modelo Vista Controlador). Este es un patrón de arquitectura software que separa la interfaz con la que interactúa el usuario, de los datos y lógica de negocio; proponiendo la división de la aplicación en estos tres componentes.

Como ya se menciona en el apartado de tecnologías, la aplicación ha sido desarrollada utilizando Spring a través de una estructura multicapa. Esta arquitectura es una de las más comunes y utilizadas en entornos empresariales, ya que divide el código en capas lo más independientes y desacopladas posibles (véase Figura 19), pero a su vez interconectadas, facilitando así el cambio y mantenibilidad de la aplicación.

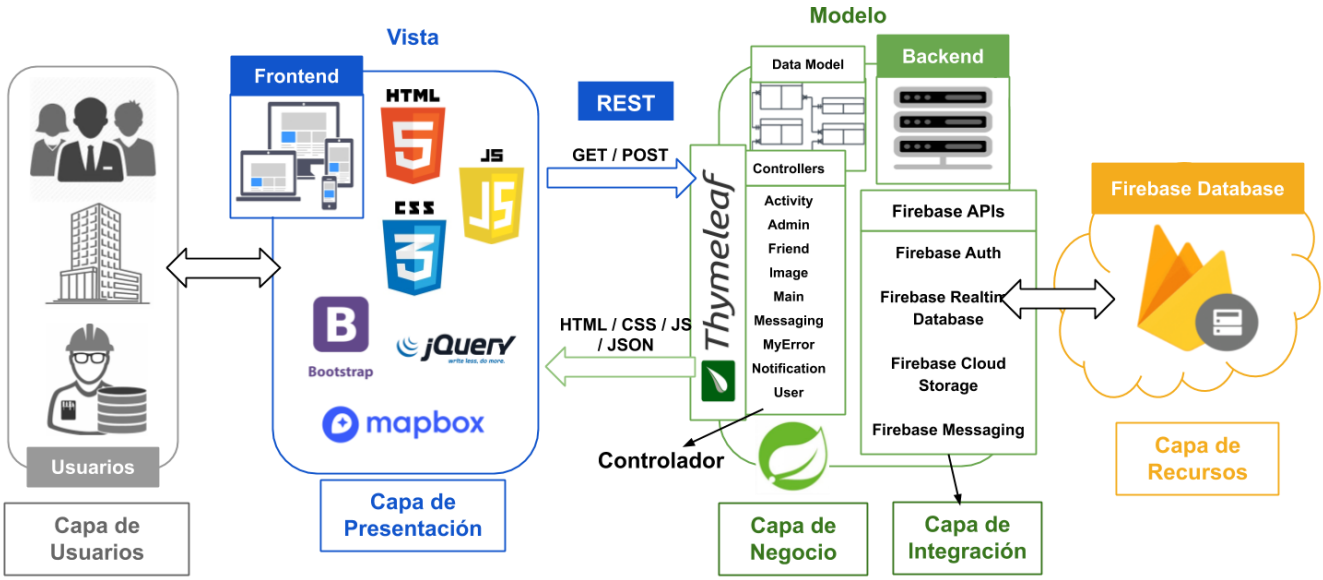


Figura 19. Arquitectura de la aplicación

6.1. Capa de usuarios

En primer lugar, se encuentra la capa conceptual de usuarios, que representa a los clientes que interactúan con la aplicación (Personas, Empresas y Administradores).

6.2. Capa de presentación

En segundo lugar, la capa de presentación o frontend se encuentra desarrollada mediante tecnologías web y constituye el cliente o interfaz sobre la que opera el usuario. Esta también constituye la Vista de nuestra aplicación, ofreciendo al cliente las diferentes interfaces de usuario que componen el proyecto, y proporcionando al servidor un punto de acceso e intercambio de datos a través de peticiones y respuestas.

Es la encargada de hacer accesible el modelo (la información) a la interfaz de usuario, transformando los datos que le llegan en un formato que el cliente sea capaz de entender y enviando las peticiones correspondientes al Controlador.

Este frontend está compuesto (Figura 20) por plantillas HTML implementadas con Bootstrap que contienen la estructura de la interfaz cuyo estilo se define mediante CSS para dar un aspecto más llamativo a la aplicación. A su vez se utiliza Javascript para dotar de dinamismo a las páginas web, utilizando JQuery para integrar las diferentes librerías Datatables, Moment.js y Mapbox descritas en la sección de tecnologías (véanse las subsecciones 3.1.8, 3.1.9 y 3.1.10 respectivamente).

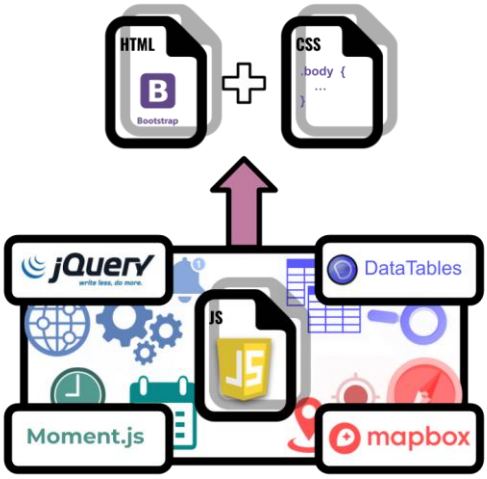


Figura 20. Arquitectura interna de la Capa de Presentación

6.3. Capa de negocio

La tercera columna representa al servidor o backend que encapsula la mayor parte de la funcionalidad. Como en la gran mayoría de proyectos, es el encargado de implementar la lógica de negocio, modelo de datos y control. A través de éstos se gestionan las peticiones que se reciben del frontend.

Es la capa donde se produce toda la interacción entre los elementos del MVC, pues la Vista envía las peticiones del usuario a través de los Controladores (clases anotadas mediante la etiqueta @Controller), encargados de invocar la distinta funcionalidad de la que se constituye el Modelo.

Este modelo representa través de las entidades de negocio (clases ubicadas en el paquete de domain) los datos solicitados que deben devolverse al controlador para que este los entregue de nuevo a la vista.

6.4. Capa de integración

También es implementada en el backend, siendo la encargada de intercomunicar el negocio con la base de datos. Para ello, el proyecto se ha desarrollado de manera que pueda seguirse incrementando su funcionalidad de forma fácil y sencilla, siendo el modelo el punto de acceso a la información, y utilizando con este fin servicios de aplicación (interfaces ubicadas en el paquete services que exponen la funcionalidad de la aplicación).

Éstos a su vez implementan las operaciones CRUD (crear, leer, actualizar y borrar) que integran y definen esta capa bajo una base de datos externa gestionada por Firebase. Utilizando esta plataforma el sistema es capaz de acceder a la información alojada, siendo esta gestionable a través de las APIs y librerías Java mencionadas en el capítulo 5.

6.5. Capa de Recursos

Por último, en la Figura 21 se observan los diferentes componentes que forman la Capa externa de Recursos. Esta alberga todos los datos persistentes de nuestra aplicación y nos proporciona una serie de métodos y operaciones asíncronas para poder gestionar y manipular nuestros datos: autenticación, usuarios, actividades, notificaciones, mensajes y contenido multimedia (véase la sección 5. Modelo de datos).

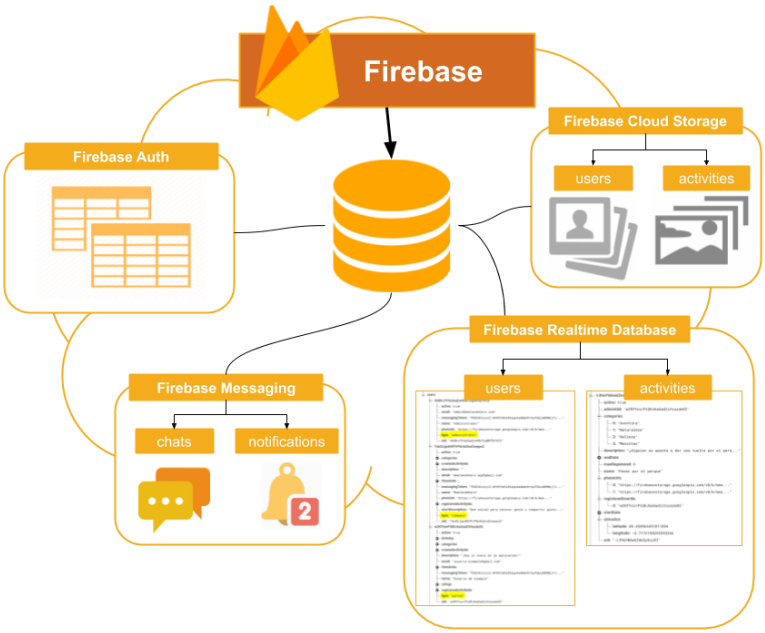


Figura 21. Arquitectura interna de la Capa de Recursos

7.Diseño de la aplicación

En esta sección se explicará cómo se ha implementado la funcionalidad más relevante de la aplicación según los actores que intervienen, tanto a nivel visual como a nivel de código.

7.1. Aspectos de diseño

En este apartado se indicarán las pautas y principios seguidos para el diseño visual de la aplicación y cómo éstos se han aplicado.

En primer lugar, definiremos lo que se entiende por usabilidad, siendo esta la facilidad que tienen los usuarios de interactuar y relacionarse con la interfaz de usuario de nuestra página.

Con el objetivo de crear interfaces usables que aumenten la eficiencia de las páginas web, reduzcan los costes y aumenten la fidelización de los usuarios, Jakob Nielsen, doctor en diseño de interfaces de usuario y ciencias de la computación por la Universidad Técnica de Dinamarca, definió los 10 principios de diseño basados en el usuario:

7.1.1. Visibilidad del estado del sistema

El usuario debe ser consciente en todo momento de lo que está pasando y en qué punto de la navegación se encuentra.

En este sentido la aplicación muestra en color resaltado el menú en el que se encuentra actualmente el usuario (Figura 22), y de no ser esta página un menú principal, se indica en negrita y con un gran tamaño de fuente el caso de uso con el que usuario se encuentra interactuando en ese momento (Figura 23).

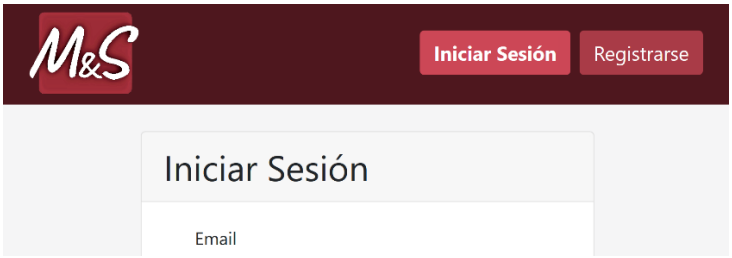


Figura 22. Vista de Inicio de sesión con la opción resaltada

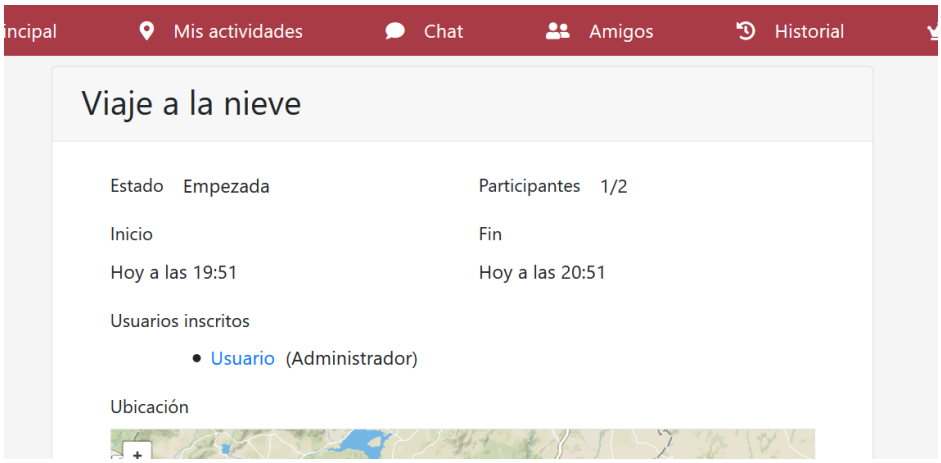


Figura 23. Vista de una actividad que no corresponde con ningún menú

7.1.2. Adecuación entre el sistema y el mundo real

La aplicación debe de mostrar su información y funcionalidad a través del mismo lenguaje que los usuarios.

Para ello, se emplean iconos identificables y que albergan relación con el mundo real, como es el caso de las categorías (Figura 23), la papelera de basura para eliminar una actividad ya terminada, representar el historial mediante un reloj, el chat mediante un bocadillo de texto, la página principal con una casa, etc.

Figura 24. Categorías de las actividades identificadas mediante iconos

7.1.3. Libertad y control por el usuario

Con el objetivo de dar más libertad y control al usuario, éste debe ser capaz de deshacer los cambios que recién haya realizado. Para ello se ofrecen las opciones de confirmar o cancelar los cambios en caso de que el usuario requiera realizar una modificación de una actividad o de su propio perfil.

También se realiza una verificación adicional en el caso de que el usuario desee dar de baja su perfil (Figura 25).

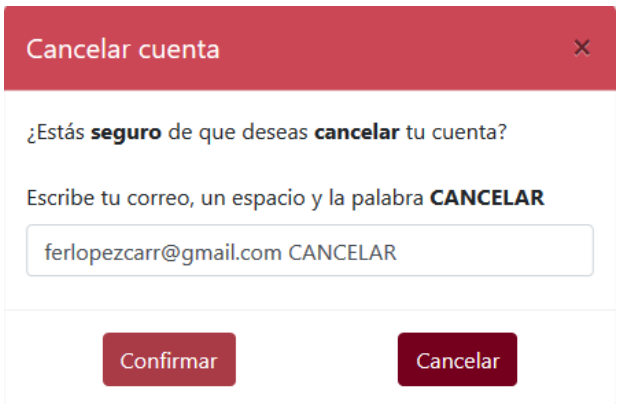


Figura 25. Mensaje de confirmación de baja junto con las opciones de confirmar y cancelar

7.1.4. Consistencia y estándares

Resulta conveniente seguir una serie de patrones y estándares ampliamente utilizados que el usuario ya identifica para no confundirlo.

Un ejemplo de esto es la existencia del logo en la parte superior izquierda de la página web (Figura 26) y el uso del icono de menú móvil (hamburguesa) en el caso de que el sitio sea consultado desde este dispositivo (Figura 27). De manera análoga, las notificaciones se encuentran cercanas a la identificación del usuario, y la selección del nombre del usuario en la parte superior derecha de la página conduce al usuario a su propio perfil.

Figura 26. Barra principal de la aplicación mostrando el logo, las notificaciones y el perfil

Figura 27. Barra principal de la aplicación visualizada en un dispositivo móvil

7.1.5. Prevención de errores

Con la finalidad de que el usuario no introduzca datos erróneos ni comprometa el sistema, las comprobaciones y validaciones de información se realizan tanto en el cliente como el servidor (Figura 28). Así mismo, al

usuario sólo se le ofrece la funcionalidad a la que puede acceder en cada momento, como puede ser el caso de no mostrar el botón de modificar o desinscribirse cuando la actividad ha empezado (Figura 29), y de sustituir los iconos de compartir por eliminar en la lista de actividades una vez estas han finalizado.

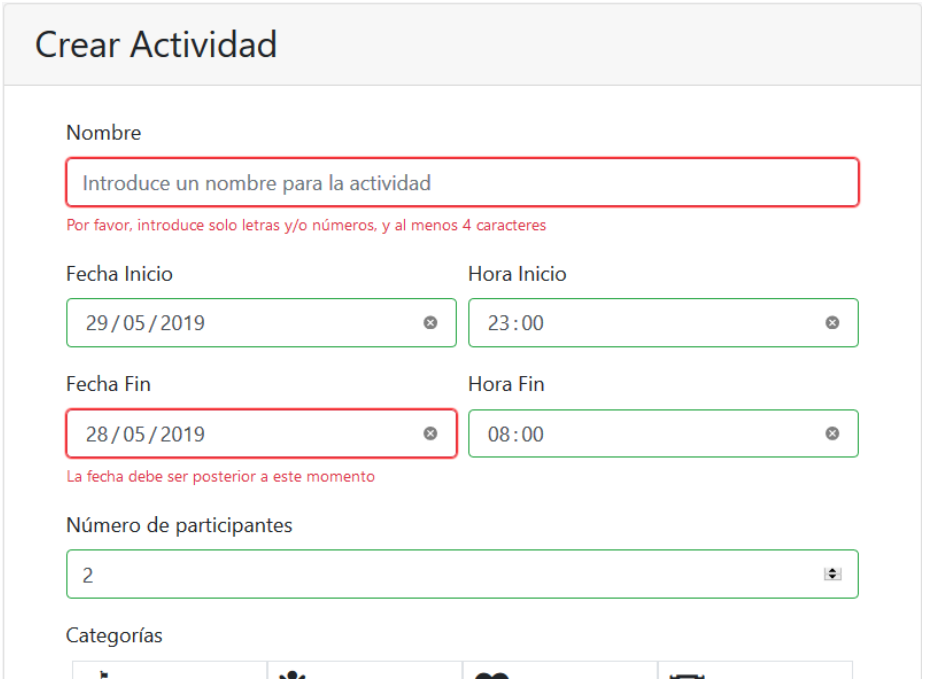


Figura 28. Validaciones al crear una actividad

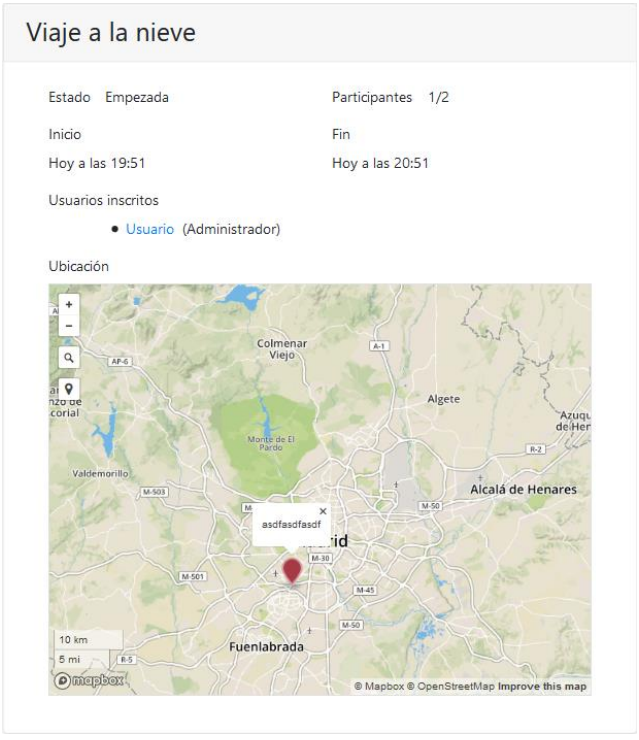


Figura 29. Vista de ver actividad una vez ésta ha empezado

7.1.6. Reconocer mejor que recordar

Para hacer las acciones más visibles y reconocibles, la aplicación utiliza la mayor cantidad de iconos diferentes e identificables dependiendo del caso de uso u opciones que se plantean (Figura 30), de tal manera que el usuario pueda relacionar estas figuras visuales con la funcionalidad que corresponda, aumentando la usabilidad y fluidez de la herramienta.

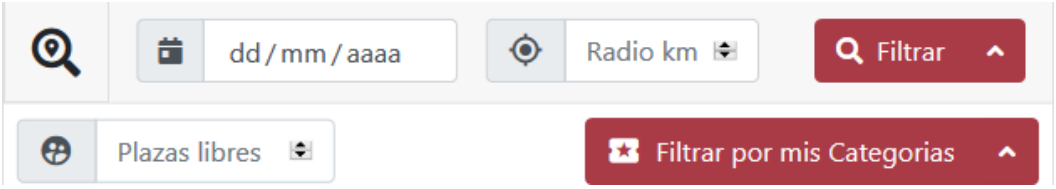


Figura 30. Filtro del mapa con diferentes iconos en función de la opción

También se emplea de manera sistemática el uso de cards (Figura 29), bloques de contenido que organizan la información ofreciendo el título de la sección en la parte superior y dando paso a la funcionalidad que se implementa.

7.1.7. Flexibilidad y eficiencia de uso

Se proporcionan diferentes puntos de acceso hacia una misma funcionalidad con el objetivo de ofrecer flexibilidad y adaptabilidad al usuario.

La aplicación ofrece siempre un enlace a la entidad que se referencia en cada caso de uso, siendo el caso de las notificaciones, usuarios inscritos en la actividad, las actividades situadas en lista que desplazan el mapa a la ubicación del evento seleccionado (véase Figura 31), etc.

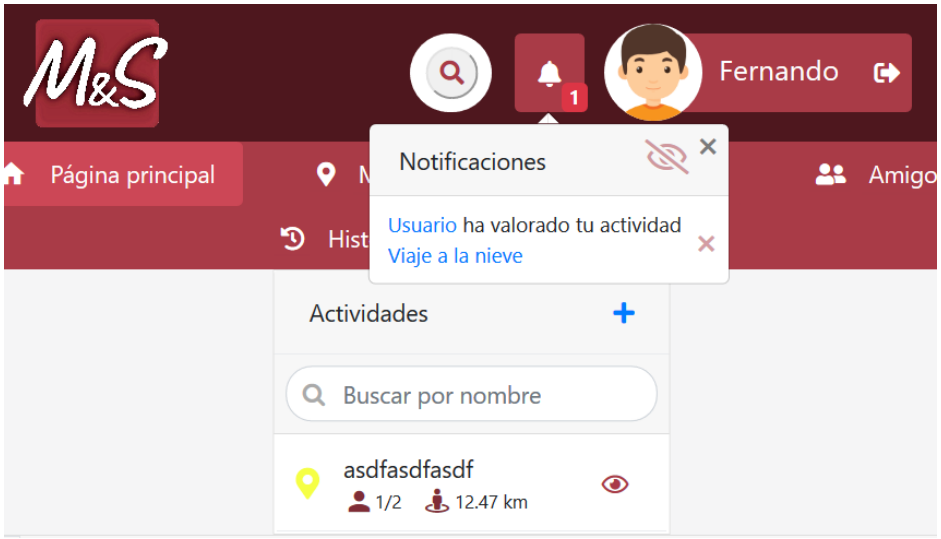


Figura 31. Notificaciones y lista de actividades con enlaces a las entidades

7.1.8. Estética y diseño minimalista

Con el objetivo de facilitar y simplificar el uso de la herramienta, se proporciona sólo la información más relevante en el caso de que las entidades se muestren en forma de lista (Figura 32), haciendo accesible el resto de información a través de un enlace a la vista correspondiente.

Adicionalmente, en caso de que en la interfaz no exista espacio suficiente como para mostrar la descripción de la funcionalidad, se utilizan iconos significativos que distinguen la acción seleccionada del resto; tal es el caso de la lista de actividades del mapa, el filtro, o la vista de amigos del usuario (véase Figura 33).

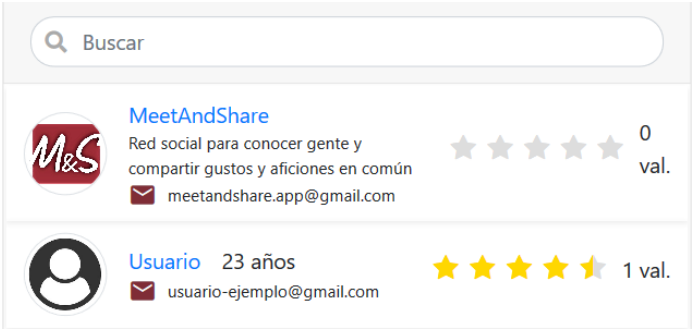


Figura 32. Entidades con información relevante

Figura 33. Vista con iconos identificativos sin soporte escrito

7.1.9. Ayudar a los usuarios a reconocer, diagnosticar y solucionar los errores

En el caso de que se produzca algún fallo en la aplicación, y este provenga de la interacción habitual del usuario, los mensajes de error se expresarán de manera que el usuario pueda identificar la causa del problema (Figura 28).

No obstante, en caso de que el usuario haya manipulado la información que se envía al servidor a través de alguna herramienta con el objetivo de romper el sistema o realizar acciones que en un principio no tendría

permitidas, la aplicación mostrará un error genérico o redireccionará directamente al usuario a la página principal.

Los errores inesperados o que no hayan sido tratados serán identificados y mostrados en el servidor, notificando al usuario de la situación.

7.1.10. Ayuda y documentación

La aplicación está destinada a cualquier clase de público, por lo que la funcionalidad e información se ofrece de una manera clara y organizada.

Pese a esto, en el caso de que no exista contenido en alguna sección de la actividad, se muestra al usuario un mensaje de ayuda que le informa de la situación (Figura 34). Tal es el caso de la búsqueda, la lista de actividades, el historial, y los rankings.

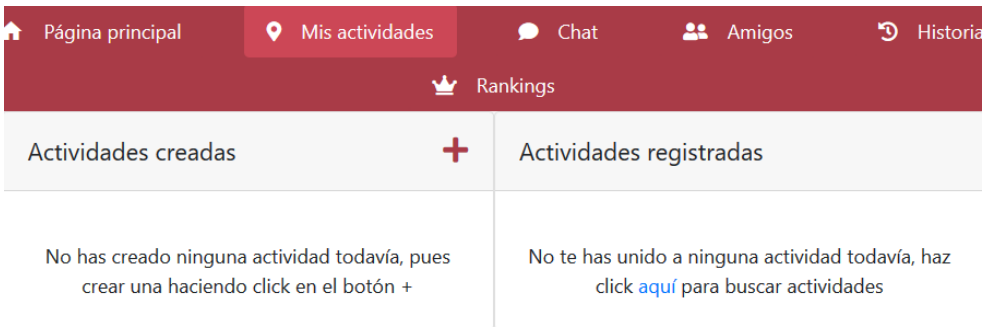


Figura 34. Vista de mis actividades con texto de ayuda

7.2. Funcionalidad principal

En primer lugar, se exponen las diferentes clases interfaces y paquetes que componen la aplicación (Figura 35).

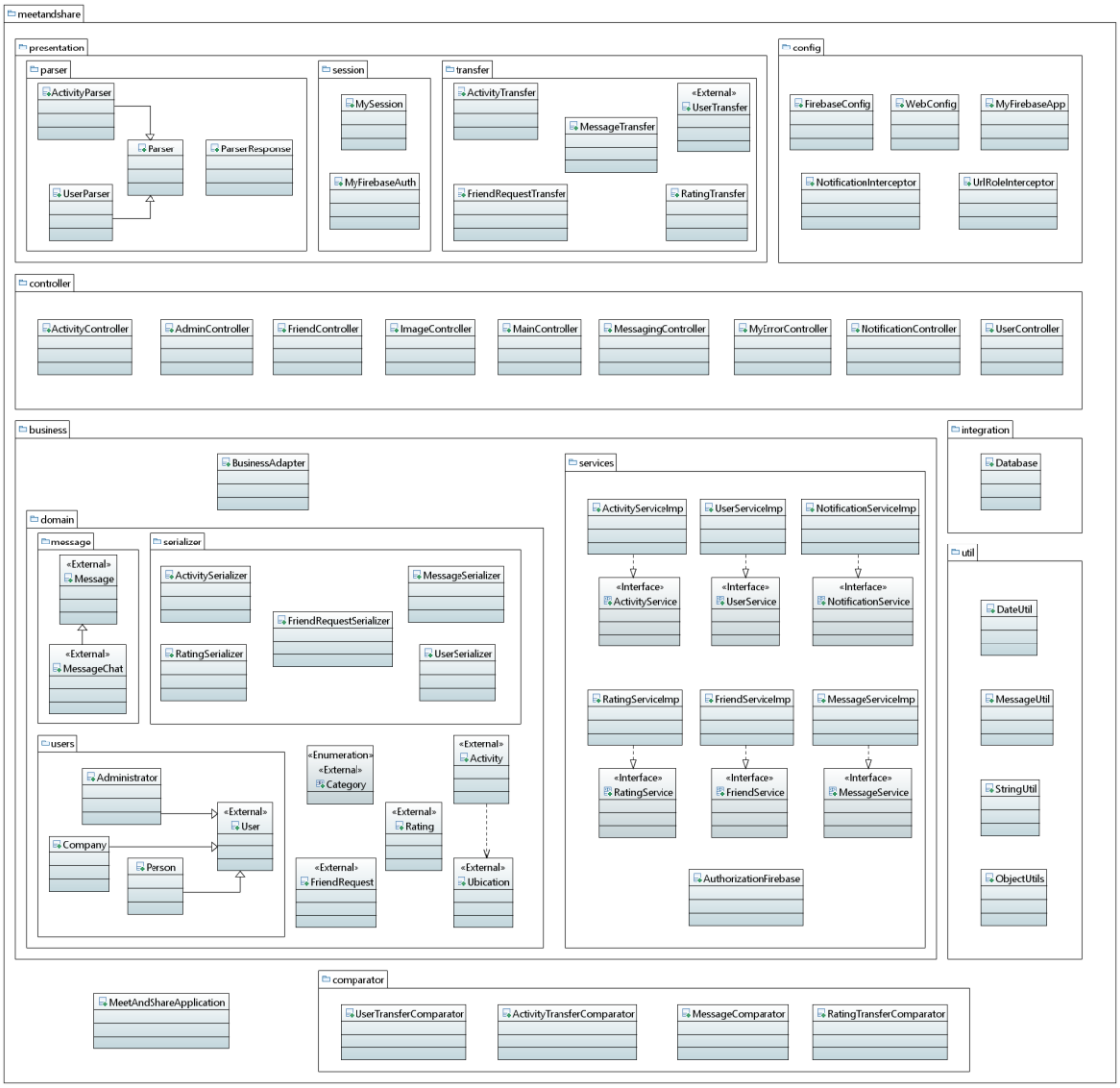


Figura 35. Diagrama de clases de la aplicación

Una vez se conoce la estructura del proyecto, se continúa desarrollando la funcionalidad principal que se presenta.

7.2.1. Inicio de sesión

Mediante el inicio de sesión, los usuarios registrados en la aplicación son capaces de acceder a los diferentes servicios que esta ofrece. Para ello, el usuario deberá introducir correctamente su correo electrónico y contraseña de acceso (véase Figura 36). Mientras se comprueban los datos enviados, ambas versiones mostrarán un icono de carga para informar al usuario que la petición ha sido recibida y se está tramitando.

Tras esto se iniciará sesión en el cliente, y en caso de que se hayan introducido los datos correctos, se lanzará la función *firebase.auth().onAuthStateChanged* (véase Código 1).

En esta, se obtendrá el token para poder enviar y recibir mensajes, que se enviará al servidor; y se recuperará el token de inicio de sesión que se mandará de nuevo al servidor para crear la sesión y recargar la página de inicio.

Al salir de la aplicación web (cerrando la pestaña o navegador), se mantiene la sesión guardada, de manera que, si el usuario no cierra la sesión, se redirigirá a la página de inicio la próxima vez que acceda al sitio.

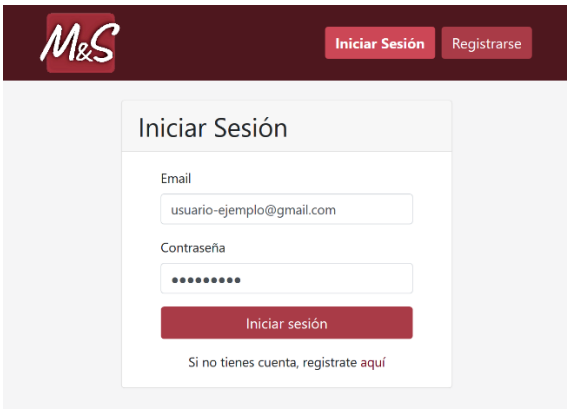


Figura 36. Página de inicio de sesión

```
firebase.auth().onAuthStateChanged(function(user) {
    console.log("on user found");
    if (user && (typeof logout == "undefined" || (typeof logout == "object" && !logout))) {
        // User is signed in.
        var emailUser = user.email;
        var photoURL = user.photoURL;
        var uid = user.uid;

        if(photoURL) {
            $("#profileImage").attr("src", photoURL);
            $("#profileImage").css({"background-image" : "none"});
            $("#profileImageView").attr("src", photoURL);
            $("#removeUserPhoto").attr('disabled', false);
        }

        firebase.messaging().getToken().then(function(currentToken) {
            if (currentToken) {
                if(uid !== null) {
                    $.ajax({
                        type: "POST",
                        url: "/token",
                        contentType: "application/json",
                        data: JSON.stringify({
                            "uid": uid,
                            "token": currentToken
                        }),
                        success: function(error) {
                            if(error === "")
                                console.log("Token sended successfully");
                            else {
                                $("#msgModal").find(".modal-title").text("Error");
                                $("#msgModal").find("#msg").text(error);
                            }
                        },
                        error: function (xhr, ajaxOptions, thrownError) {
                            console.log(xhr.status); console.log(thrownError);
                        }
                    });
                }
            } else
                console.log('No Instance ID token available. Request permission to generate one.');
        }).catch(function(err) {
              console.log('An error occurred while retrieving token. ', err);
        });

        firebase.auth().currentUser.getIdToken(/* forceRefresh */ true).then(function(idToken) {
            if($("#idToken"))
                $("#idToken").val(idToken);

            if(user && idToken && !userLogged) {//not already logged
                $("#msgModal").find(".modal-title").text("Iniciando sesión");
                $("#msgModal").find("#msg").text(emailUser);
                $("#msgModal").modal('toggle');

                $.ajax({
                    type: "POST",
                    url: "/login",
                    data: idToken,
                    success: function(error) {
                        if(error === "")
                            document.location.href = '/';
                        else {
                            $("#msgModal").find(".modal-title").text("Error");
                            $("#msgModal").find("#msg").text(error);
                        }
                    }
                    error: function (xhr, ajaxOptions, thrownError) {
                        console.log(xhr.status); console.log(thrownError);
                    }
                });
            }
        }).catch(function(error) {
            console.log(error);
        });
    }
});
```

Código 1. firebase-auth.js Inicio de sesión

7.2.2. Creación de usuarios

A través de esta funcionalidad, los potenciales usuarios podrán registrarse en la aplicación, bien como persona o como empresa (véase Figura 37). Cabe destacar, que los administradores son los únicos capaces de dar de alta nuevos usuarios de este tipo (Figura 38).

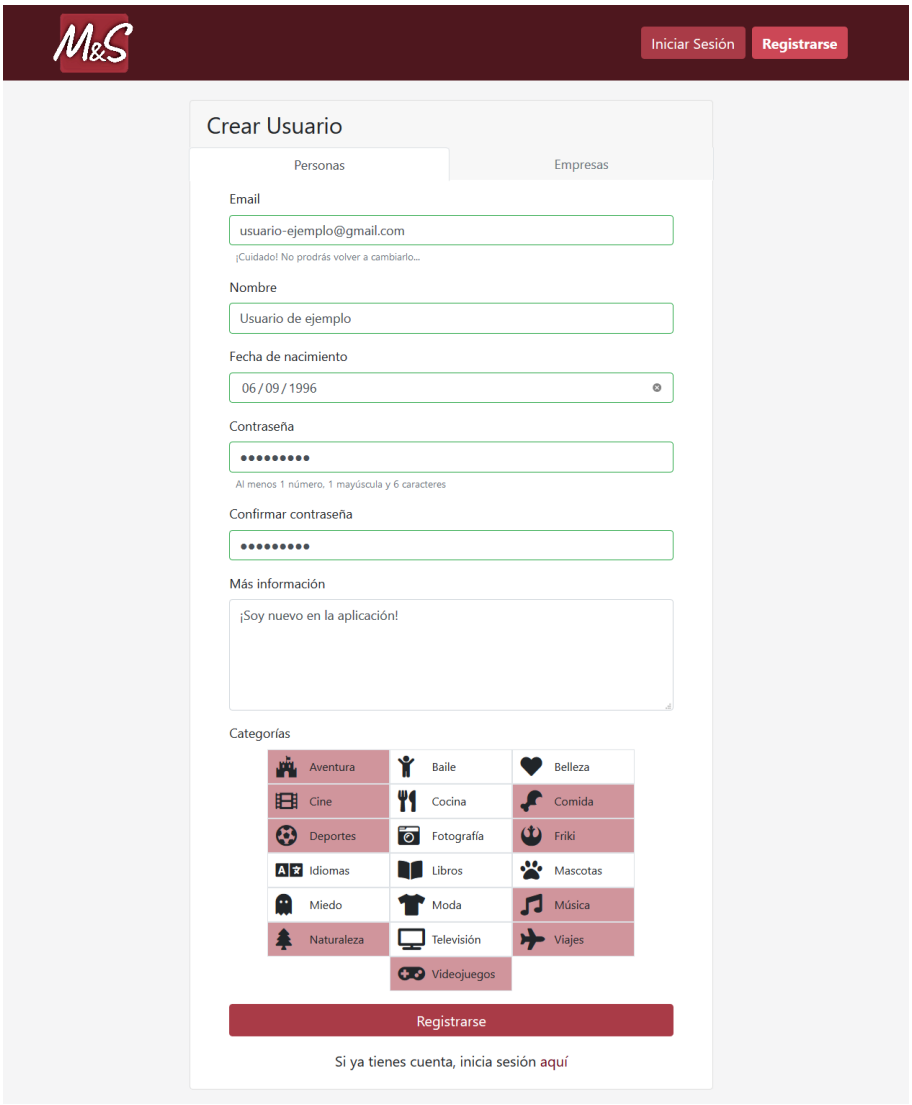


Figura 37. Página de registro para Personas y Empresas

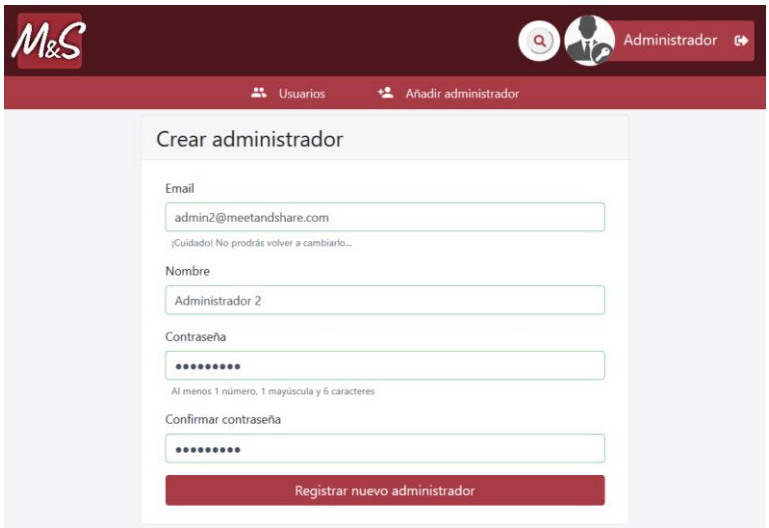


Figura 38. Página de registro para Administradores

Para crear un usuario se puede disponer de los siguientes atributos (Tabla 8):

Atributos	Descripción
Email*	Campo identificador del usuario que debe ser de la forma *usuario-guia@gmail.com* y no puede volver a cambiarse
Nombre*	Solo puede contener palabras con letras
Fecha de nacimiento* (exclusivo para personas)	Campo para identificar la edad del usuario, siendo necesario que sea mayor de edad
Contraseña*	Debe contener al menos 1 número, 1 mayúscula y debe contener al menos 6 caracteres
Confirmar contraseña*	Debe ser exactamente igual a la anterior
Descripción breve (exclusivo para empresas)	Información adicional que puede añadir la empresa y que aparecerá junto con el nombre de ésta
Más información	Descripción del usuario
Categorías	Lista con las categorías seleccionadas por el usuario que definirán sus gustos y aficiones
Los campos marcados con * son obligatorios	

Tabla 8. Atributos pertenecientes a un usuario registrado

Una vez el usuario ha rellenado correctamente el formulario, se procede a registrar al usuario en el servidor, siendo necesario comprobar si existe ya un usuario con el email especificado en la base de datos. En caso afirmativo, debe comprobarse el tipo de usuario a dar de alta, siendo necesario que un administrador se encuentre con la sesión activa para crear dicho usuario. Tras esto se registrará al usuario en Firebase Auth, Firebase Realtime Database, y por último en el caso de las personas y empresas se procederá a iniciar la sesión (véase Código 2).

```java
@PostMapping("/register")
public ModelAndView register(ModelAndView modelAndView, HttpSession session, SessionStatus status, @ModelAttribute UserTransfer userTransfer) {
    RedirectView redirectView = new RedirectView();
    String error = null;

    UserTransfer userLogged = null;
    if(session != null && session.getAttribute(MySession.userLoggedStr) != null)
        userLogged= MySession.getInstance().getUserLogged(modelAndView, session, status);

    boolean userIsLogged = (userLogged != null && userLogged.getType() != null && !userLogged.getType().isEmpty());
    boolean userIsNewAdmin = (userTransfer != null && userTransfer.getType() != null && userTransfer.getType().equalsIgnoreCase(Administrator.typeStr));
    boolean createUser = true;
    if(userIsNewAdmin)
        createUser = (userIsLogged && userLogged.getType().equalsIgnoreCase(Administrator.typeStr));

    UserRecord firebaseUser = null;

    if(UserParser.getInstance().parseUserRegister(redirectView, userTransfer)) {
        UserRecord userAuth = null;
        try {
            userAuth = MyFirebaseApp.getAuthenticationInstance().getUserByEmail(userTransfer.getEmail());
            if(userAuth != null) {
                error = "Ya existe un usuario con ese email";
            }
        } catch (FirebaseAuthException e) {
            if(createUser)
                firebaseUser = MySession.getInstance().createUser(userTransfer);
            else
                error = "No tienes permiso para crear nuevos administradores";
        }
    }

    if(error == null && firebaseUser != null) {
        userTransfer.setUid(firebaseUser.getUid());
        User user = UserSerializer.transferToDomainObj(userTransfer);
        user.setUid(firebaseUser.getUid());
        try {
            User userDatabase = null;

            if(createUser) {
                error = UserServiceImp.getInstance().create(user);
                userDatabase = UserServiceImp.getInstance().get(firebaseUser.getUid());
            }
            else
                error = "No tienes permiso para crear nuevos administradores";

            if(error == null && userDatabase != null && userDatabase.getActive()) {
                if(!userIsLogged)
                    MySession.getInstance().setUserLogged(session, userDatabase, userDatabase.getType());

                if(userDatabase.getType().equalsIgnoreCase(Person.typeStr))
                    modelAndView.setViewName("redirect:/map");
                else if(userDatabase.getType().equalsIgnoreCase(Company.typeStr))
                    modelAndView.setViewName("redirect:/map");
                else if(userDatabase.getType().equalsIgnoreCase(Administrator.typeStr))
                    modelAndView.setViewName("redirect:/users");
            }
        } catch (InterruptedException e) {
            error = MySession.databaseErrorStr;
            System.err.println(e.getMessage());
        }
    }

    if(error != null) {
        MySession.notifyUserModal(redirectView, null, error);
        modelAndView.addAllObjects(redirectView.getAttributesMap());
        modelAndView.setViewName("redirect:/register");
    }
    else
        modelAndView.setViewName("redirect:/");

    return modelAndView;
}
```

Código 2. MainController.java Registro

7.2.3. Buscar usuarios / actividades

En este caso la funcionalidad es la misma, pero se presenta de forma diferente.

Los administradores ya cuentan con la información de todos los usuarios de la aplicación (véase Figura 39), por lo que la búsqueda se realiza en el lado del cliente.

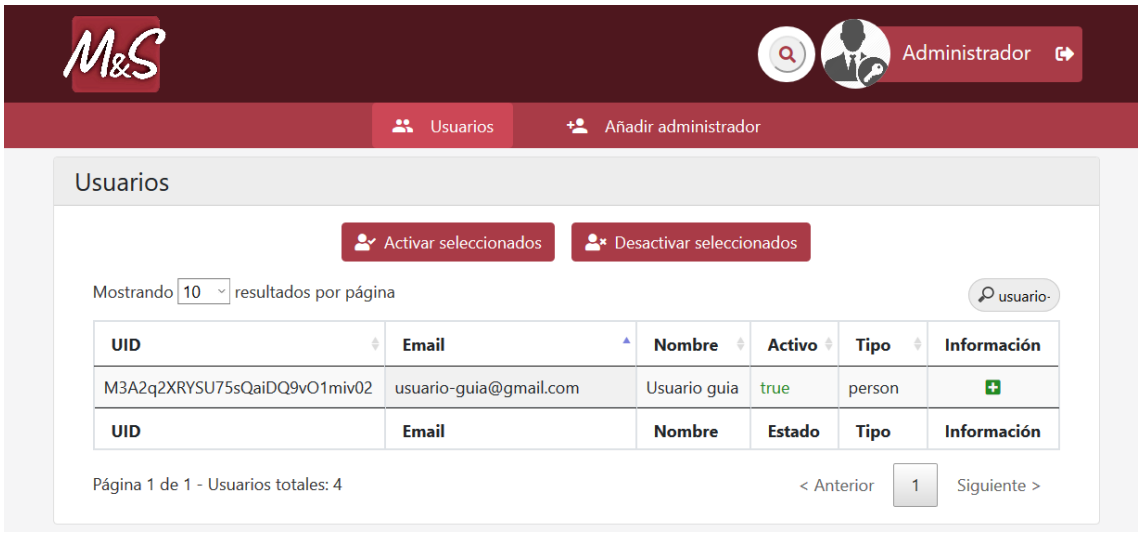


Figura 39. Página de búsqueda de usuarios para Administradores

Las personas o empresas por otro lado utilizarán la barra superior de búsqueda (véase Figura 40) para buscar tanto usuarios como actividades, siendo esta sensible a mayúsculas y minúsculas como se puede apreciar en el Código 3 (en el caso de las actividades se implementa de manera análoga). Esto es debido a que la API no contiene un método de texto contenido al uso, por lo que es necesario realizar una serie de cambios para que la búsqueda se realice correctamente.

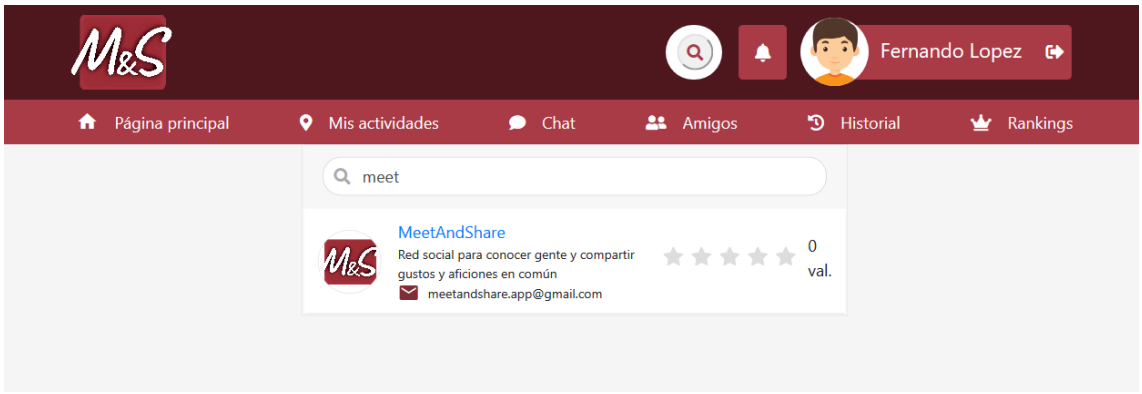


Figura 40. Página de búsqueda de usuarios y actividades para Personas y Empresas

```java
@Override
public List<User> findBy(String key, String searchText) throws InterruptedException {
    key = key.trim();
    searchText = searchText.trim();
    if(key == null || key.isEmpty() || searchText == null || searchText.isEmpty())
        return null;
    DatabaseReference reference = Database.getUsersDatabaseReference();

    List<User> users = new ArrayList<User>();

    Semaphore semaphore = new Semaphore(0);
    ValueEventListener userListener = new ValueEventListener(){
        @Override
        public void onDataChange(DataSnapshot dataSnapshot) {
            Iterable<DataSnapshot> dataSnapshotRoot = dataSnapshot.getChildren();

            for(DataSnapshot child : dataSnapshotRoot){
                try {
                    User user = BusinessAdapter.objectToUserEntity(child.getValue());
                    if(user != null && !users.contains(user))
                        users.add(user);
                } catch (IllegalArgumentException | IllegalAccessException e) {
                    log.error(e.getMessage());
                    semaphore.release();
                }
            }
            semaphore.release();
        }

        @Override
        public void onCancelled(DatabaseError error) {
            error.toException();
        }
    };

    if(searchText != null && !searchText.isEmpty() && searchText.length() > 1) {
        searchText = searchText.substring(0, searchText.length()-1);

        reference.orderByChild(key).startAt(searchText, key).endAt(searchText+"\uf8ff", key)
        .addValueEventListener(userListener);
        semaphore.acquire();
        reference.removeEventListener(userListener);
    }
    else {
        reference.orderByChild(key).startAt(searchText, key)
        .addValueEventListener(userListener);
        semaphore.acquire();
        reference.removeEventListener(userListener);
    }

    reference.orderByChild(key)
    .startAt(StringUtil.removeAccents(searchText), key).endAt(searchText+"\uf8ff", key)
    .addValueEventListener(userListener);
    semaphore.acquire();
    reference.removeEventListener(userListener);

    return (users.size() > 0 ? users : null);
}
```

Código 3. UserServiceImp.java Buscar usuarios

7.2.4. Modificar perfil

Los usuarios también son capaces de modificar (Figura 41) su nombre, fecha de nacimiento (en el caso de que sean personas), descripción breve (en el caso de que se trate de empresas), categorías, y descripción completa. Para ello se buscará la información del usuario, se comprobarán los datos, y se sustituirá con los cambios introducidos (véase Código 4).

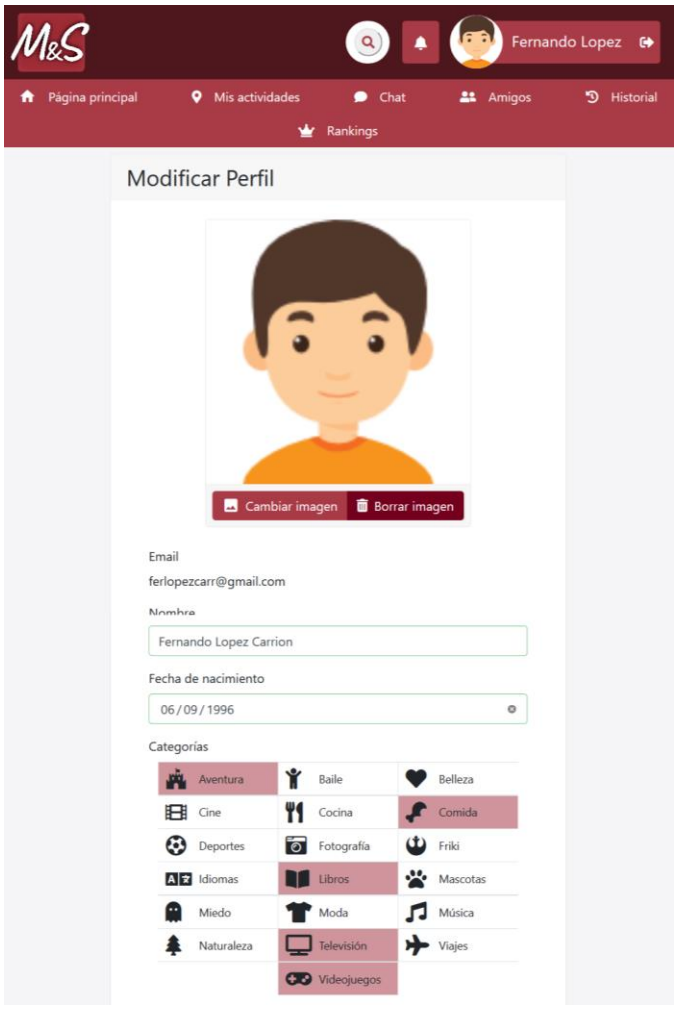


Figura 41. Página para modificar el perfil

```java
@PostMapping("/editProfile")
public RedirectView editProfilePost(ModelAndView modelAndView, HttpSession session, SessionStatus status,
        @ModelAttribute UserTransfer userTransfer){
    RedirectView redirectView = new RedirectView("/editProfile", true);
    String error = null;
    User userDatabase = null;
    UserTransfer userLogged = MySession.getInstance().getUserLogged(modelAndView, session, status);

    if(!UserParser.getInstance().parseUserModify(redirectView, userTransfer))
        userLogged = null;

    if(userLogged != null && userTransfer.getUid() != null && userTransfer.getUid().equalsIgnoreCase(userLogged.getUid())) {
        try {
            userTransfer.setUid(userLogged.getUid());
            User user = UserServiceImp.getInstance().get(userLogged.getUid());
            user.setName(userTransfer.getName());

            List<Category> categories = new ArrayList<Category>();
            JSONArray categoriesJSONArray = userTransfer.getCategoriesJSON();
            if(userTransfer.getCategoriesJSON() != null && userTransfer.getCategoriesJSON().length() > 0) {
                for(int i = 0; i < categoriesJSONArray.length(); i++) {
                    Category category = Category.getCategoryFromStringValue(categoriesJSONArray.getString(i));
                    if(category != null)
                        categories.add(category);
                }
            }

            if(user.getType().equalsIgnoreCase(Person.typeStr)) {
                Person person = (Person)user;
                person.setBirthday(userTransfer.getBirthday());
                person.setDescription(userTransfer.getDescription());
                person.setCategories(categories);
            }
            else if(user.getType().equalsIgnoreCase(Company.typeStr)) {
                Company company = (Company)user;
                company.setShortDescription(userTransfer.getShortDescription());
                company.setShortDescription(userTransfer.getShortDescription());
                company.setDescription(userTransfer.getDescription());
                company.setCategories(categories);
            }

            error = UserServiceImp.getInstance().save(user);

            if(error == null) {
                userDatabase = UserServiceImp.getInstance().get(userLogged.getUid());

                if(userDatabase == null)
                    error = MySession.userNotFoundErrorStr;
                else {
                    MySession.getInstance().setUserLogged(session, userDatabase, userDatabase.getType());
                    userTransfer = UserSerializer.domainObjToTransfer(userDatabase);
                }
            }
        } catch (InterruptedException | JSONException e) {
            error = MySession.databaseErrorStr;
            log.error(e.getMessage());
        }
    }
```

Código 4. UserController.java Editar perfil

De manera análoga se podrá cambiar la imagen de perfil presionando en el botón correspondiente, y adjuntando el archivo deseado como se observa en la Figura 42. Tras la subida se actualizará la foto, pudiendo en cualquier momento eliminarla mediante el botón de borrar imagen.

Cabe destacar que todo este proceso se realiza en el lado del cliente (véase Código 5), por lo que el servidor nunca recibirá archivos multimedia. Para cambiar la imagen se comprobará la extensión del fichero, y se buscará la imagen anterior. En caso de que se encuentre, se borrará el fichero antiguo y se sustituirá con el archivo seleccionado.

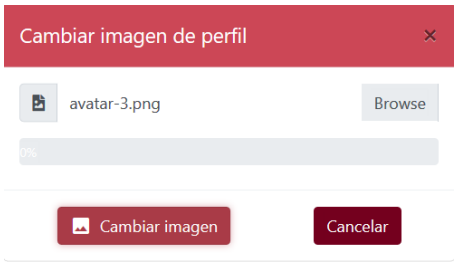


Figura 42. Modal para cambiar la imagen de perfil

```
$("#updatePhoto").click(function(event) {

    if(file && firebase.auth().currentUser) {

        let profilePictureUserRef = getProfilePictureUserRef();

        let fileExtensionSplit = file.name.split('.');
        let fileExtension = fileExtensionSplit[fileExtensionSplit.length-1];

        let isValid = isExtensionValid(fileExtension);
        if(isValid !== null) {
            var newImageUserReference = firebase.storage().ref(profilePictureUserRef + "profile."+fileExtension);

            let photoUrlOld = photoUrl;

            $("#progressBarImage").parent().removeClass("d-none");
            if(photoUrlOld && photoUrlOld !== "") {
                let urlFileName = getFileNameFromUrl(photoUrlOld);

                if(urlFileName && urlFileName !== "") {
                    let oldImageUserReference = firebase.storage().ref(profilePictureUserRef+urlFileName);

                    // Delete the file
                    oldImageUserReference.delete().then(function() {
                        uploadFile(newImageUserReference);
                    }).catch(function(error) {
                        console.log(error);
                    });
                }
            }
            else {
                uploadFile(newImageUserReference);
            }
        }
        else {
            let msg = "<div class='d-flex justify-content-center'>Imagen inválida, las extensiones soportadas son:</div>";
            //msg += avaliableImageFormats.toString().replace(new RegExp(',', 'g'), ", ");
            let formatTags = "<div class='d-flex justify-content-center'>";
            for(let i = 0; i < avaliableImageFormats.length; i++) {
                formatTags += "<div class='badge badge-secondary mx-1'>"+avaliableImageFormats[i]+"</div>";
            }
            formatTags += "</div>";
            //msg += "<li>Video: "+avaliableVideoFormats.toString()+"</li></ul>";
            $("#inputImageFile").addClass('is-invalid');
            invalidFeedback.html(msg+formatTags);
            invalidFeedback.show();
        }
    }

});
```

Código 5. editProfile.js Cambiar imagen de perfil

También es posible modificar la contraseña actual (Figura 41) adjuntando la contraseña anterior y una clave diferente. De esta manera (Código 6) se comprobará que los datos introducidos son correctos, y que el usuario existe y se encuentra activo. Tras esto se actualizará la contraseña en Firebase Auth y se notificará al usuario.

Figura 43. Modal para cambiar la contraseña

```
@PostMapping("/modifyPassword")
public RedirectView editProfilePost(ModelAndView modelAndView, HttpSession session, SessionStatus status,
        @ModelAttribute ("userUid") String userUid,
        @ModelAttribute ("password") String oldPassword,
        @ModelAttribute ("samePassword") String newPassword){
    RedirectView redirectView = new RedirectView("/editProfile", true);
    redirectView.addStaticAttribute("userUid", userUid);

    String error = null;

    UserTransfer userLogged = MySession.getInstance().getUserLogged(modelAndView, session, status);

    UserTransfer userParser = new UserTransfer();
    userParser.setPassword(oldPassword);
    userParser.setSamePassword(newPassword);

    if(UserParser.getInstance().parseModifyPassword(redirectView, userParser)) {
        if(userParser.getPassword().equalsIgnoreCase(userParser.getSamePassword()))
            error = "Debes introducir una nueva contraseña";
    }
    else
        userLogged = null;

    if(error == null && userLogged != null && userUid != null && !userUid.isEmpty() &&
            userLogged.getUid().equalsIgnoreCase(userUid)) {
        try {
            User user = UserServiceImp.getInstance().get(userLogged.getUid());
            UpdateRequest requestModifyUser = null;

            if(user != null && user.getActive()) {
                requestModifyUser = new UpdateRequest(userUid);
                if(requestModifyUser != null) {
                    UpdateRequest request = requestModifyUser.setPassword(newPassword);
                    UserRecord userRecord = MyFirebaseApp.getAuthenticationInstance().updateUser(request);
                    if(userRecord != null)
                        error = null;
                }
            }
            if(user == null || requestModifyUser == null)
                error = MySession.userNotFoundErrorStr;
            if(error != null && !error.isEmpty())
                modelAndView.addObject("userTransfer", UserSerializer.domainObjToTransfer(user));
        } catch (InterruptedException | FirebaseAuthException e) {
            error = MySession.databaseErrorStr;
            log.error(e.getMessage());
        }
    }

    if(error != null) {
        redirectView.addStaticAttribute("userUid", userUid);
        MySession.notifyUserModal(redirectView, null, error);
    }
    else {
        redirectView.setBeanName("/");
        MySession.notifyUserModal(redirectView, "Contraseña cambiada", "¡Tu contraseña se ha cambiado correctamente!");
    }

    return redirectView;
}
```

Código 6. UserController.java Modificar contraseña

Otra funcionalidad que ofrece la aplicación es la de cancelar la cuenta (Figura 44), escribiendo el email del usuario seguido de un espacio y la palabra CANCELAR. Para ello (véase Código 7) se comprobará que el texto adjuntado es el correcto, se desactivará la cuenta en Firebase Auth, después en Firebase Realtime Database, y por último se cerrará la sesión.

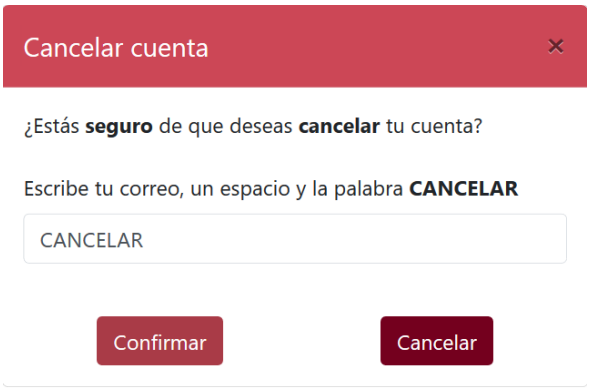


Figura 44. Interfaz para cancelar la cuenta

```
@PostMapping("/deactivateUser")
public RedirectView deactivateUser(ModelAndView modelAndView, HttpSession session, SessionStatus status,
        @ModelAttribute ("userUid") String userUid, @ModelAttribute ("textConfirmation") String textConfirmation){
    String error = null;
    UserTransfer userLogged = MySession.getInstance().getUserLogged(modelAndView, session, status);

    if(userLogged != null && userUid != null && !userUid.isEmpty() && userLogged.getUid().equalsIgnoreCase(userUid)) {

        String[] textConfirmationArray = (textConfirmation != null && !textConfirmation.isEmpty() ? textConfirmation.split(" ") : null);

        try {
            if(textConfirmationArray != null && textConfirmationArray.length == 2 &&
                    UserParser.isValidEmail(textConfirmationArray[0]) &&
                    textConfirmationArray[0].equalsIgnoreCase(userLogged.getEmail()) &&
                    textConfirmationArray[1].equalsIgnoreCase(MySession.cancelTextConfirmation))
                error = null;
            else {
                error = "Debes introducir tu correo, seguido de un espacio, y seguido de la palabra CANCELAR.";
                error += " Ejemplo: ejemplo@ejemplo.com CANCELAR";
            }
        } catch (ParseException e1) {
            error = e1.getMessage();
        }

        if(error == null) {
            User user = UserSerializer.transferToDomainObj(userLogged);
            user.setUid(userLogged.getUid());
            try {
                UpdateRequest request = new UpdateRequest(userUid);
                request = request.setDisabled(true);
                MyFirebaseApp.getAuthenticationInstance().updateUser(request);
                error = UserServiceImp.getInstance().delete(user);
            } catch (InterruptedException | FirebaseAuthException e) {
                error = MySession.databaseErrorStr;
                log.error(e.getMessage());
            }

            if(error == null) {
                UpdateRequest request = new UpdateRequest(userLogged.getUid()).setDisabled(true);
                UserRecord userRecord = null;
                try {
                    userRecord = MyFirebaseAuth.getInstance().updateUser(request);
                    MySession.getInstance().rejectUser(modelAndView, session, status);
                } catch (FirebaseAuthException e) {
                    error = MySession.databaseErrorStr;
                    log.error(e.getMessage());
                }
                if(userRecord == null)
                    error = MySession.databaseErrorStr;
            }
        }
    }

    RedirectView redirectView = null;
    if(error != null) {
        redirectView = new RedirectView("/editProfile", true);
        redirectView.addStaticAttribute("userUid", userUid);
        MySession.notifyUserModal(redirectView, null, error);
    }
```

Código 7. UserController.java Cancelar cuenta

7.2.5. Gestionar usuarios

Los administradores también son los encargados de gestionar las cuentas de las personas y las empresas. Para ello, éstos cuentan con una página (Figura 45) donde se cargan y paginan todos los usuarios de la aplicación, de modo que puedan encontrar a los usuarios que deseen dar de alta o baja.

Para ello, los administradores deberán ir seleccionando los usuarios sobre los que se quieran realizar las operaciones, y tras esto seleccionar la acción correspondiente.

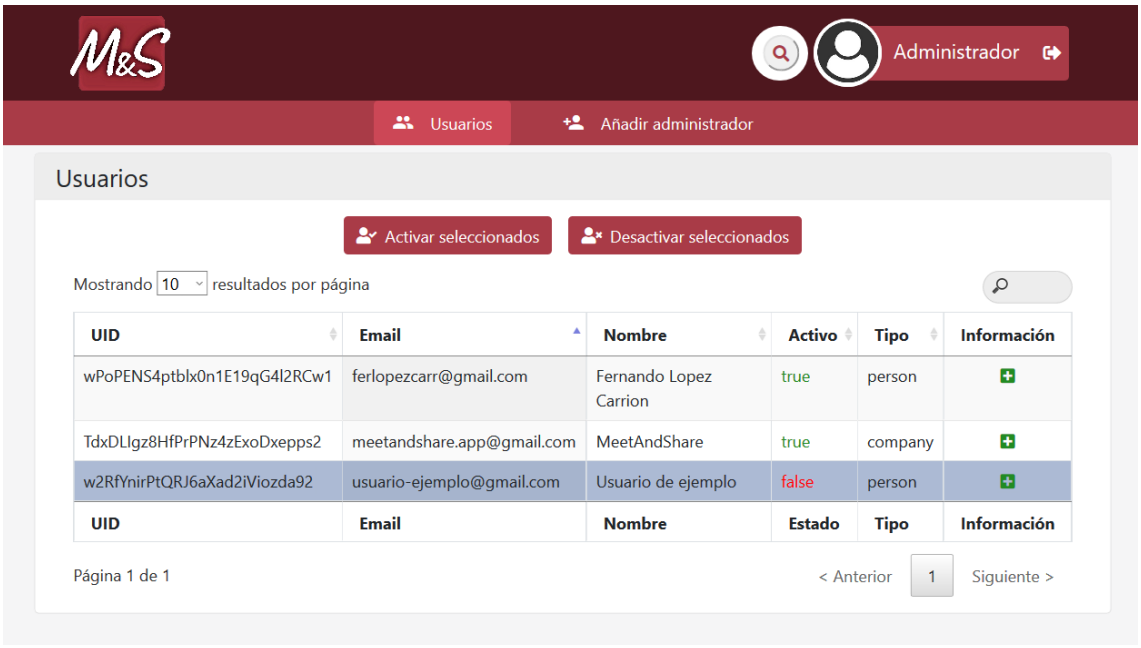


Figura 45. Página para la gestión de usuarios de los Administradores

Tras esto, se buscarán los usuarios seleccionados, y en caso de que existan, se activarán o desactivarán en función de lo elegido por el administrador (primero en Firebase Auth y después en Firebase Reltime Database) como puede verse en el Código 8.

```java
@PostMapping("/activateUsers")
public ModelAndView activateUsers(ModelAndView modelAndView, HttpSession session, SessionStatus status,
        @RequestBody String usersToActivateJson){
    String error = null;

    UserTransfer userLogged = MySession.getInstance().getUserLogged(modelAndView, session, status);

    JSONArray usersToActivateJsonArray = null;
    try {
        usersToActivateJsonArray = new JSONArray(usersToActivateJson);
    } catch (JSONException e) {
        log.error(e.getMessage());
    }

    if(userLogged != null) {
        if(usersToActivateJsonArray != null) {
            try {
                for(int i = 0; i < usersToActivateJsonArray.length(); i++) {
                    String userUid = usersToActivateJsonArray.getString(i);

                    User user = UserServiceImp.getInstance().get(userUid);
                    if(user != null && !user.getActive() && !user.getType().equalsIgnoreCase(Administrator.typeStr)) {
                        UpdateRequest request = new UpdateRequest(userUid);
                        request = request.setDisabled(false);
                        MyFirebaseApp.getAuthenticationInstance().updateUser(request);
                        user.setActive(true);
                        error = UserServiceImp.getInstance().save(user);
                    }
                }
            } catch (InterruptedException | JSONException | FirebaseAuthException e) {
                error = MySession.databaseErrorStr;
                log.error(e.getMessage());
            }
        }
    }

    RedirectView redirectView = new RedirectView();
    if(error != null) {
        MySession.notifyUserModal(redirectView, null, error);
        modelAndView.addAllObjects(redirectView.getAttributesMap());
    }

    modelAndView.setViewName("redirect:/");

    return modelAndView;
}
```

Código 8. AdminController.java Reactivar usuarios

7.2.6. Amigos

Con el objetivo de que los usuarios puedan conocerse e interactuar con otras personas, se ha implementado la funcionalidad de enviar peticiones de amistad.

Para ello, el usuario que desee enviar la petición debe buscar a otro usuario y enviar la solicitud con un mensaje opcional (a través del modal presente en la Figura 46). Tras esto, el sistema comprobará si ambos usuarios existen y en caso afirmativo enviará la solicitud de amistad (véase Código 9).

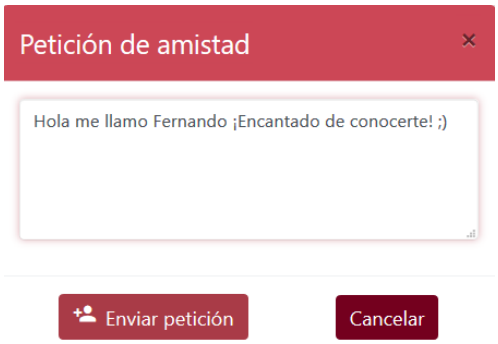


Figura 46. Modal para añadir un nuevo amigo

```
@PostMapping("/friendRequest")
public ModelAndView friendRequest(ModelAndView modelAndView, HttpSession session, SessionStatus status,
        @ModelAttribute ("friendRequest") FriendRequest friendRequest){
    String error = null;
    UserTransfer userLogged = MySession.getInstance().getUserLogged(modelAndView, session, status);

    if(userLogged != null) {

        if(friendRequest != null) {
            User userDatabase = null;
            try {
                String targetUserId = friendRequest.getUserUid();
                if(targetUserId != null && targetUserId != "") {
                    userDatabase = UserServiceImp.getInstance().get(userLogged.getUid());
                    User targetUser = UserServiceImp.getInstance().get(targetUserId);

                    if(userDatabase != null && targetUser != null &&
                            userDatabase.getActive() && targetUser.getActive()) {
                        friendRequest.setUserUid(userDatabase.getUid());
                        error = FriendServiceImp.getInstance().addFriendRequest(targetUserId, friendRequest);
                        if(error == null) {
                            List<String> registrationTokens = new ArrayList<String>();
                            registrationTokens.add(targetUser.getMessagingToken());
                            String msg = "";
                            msg += MessageUtil.getUserLink(userDatabase.getUid(), userDatabase.getName());
                            msg += " te ha enviado una <a href='/friends'>solicitud de amistad</a>";
                            List<String> uids = new ArrayList<String>();
                            uids.add(targetUserId);
                            MessageUtil.getInstance().sendNotification(Log, registrationTokens, msg, uids);
                        }
                    }
                    else
                        error = MySession.userNotFoundErrorStr;
                }
            } catch (InterruptedException e) {
                System.err.println(e.getMessage());
                error = MySession.databaseErrorStr;
            }

            if(userDatabase == null)
                error = MySession.databaseErrorStr;
        }
    }

    RedirectView redirectView = new RedirectView();
    if(error != null)
        MySession.notifyUserModal(redirectView, null, error);
    else
        MySession.notifyUserModal(redirectView, "Petición enviada", "Petición enviada con éxito");
    modelAndView.addAllObjects(redirectView.getAttributesMap());

    modelAndView.setViewName("redirect:/friends");

    return modelAndView;
}
```

Código 9. FriendController.java Enviar petición de amistad

De manera análoga, el usuario objetivo podrá resolver la petición de amistad, aceptándola o rechazándola (Figura 47). Para ello, la herramienta comprobará que los datos son correctos, que existen ambos usuarios, que la petición fue enviada, y en caso afirmativo resolverá la petición,

añadiendo a cada usuario a sus respectivas listas de amigos, y permitiéndoles mandarse mensajes entre sí (véase Código 10).

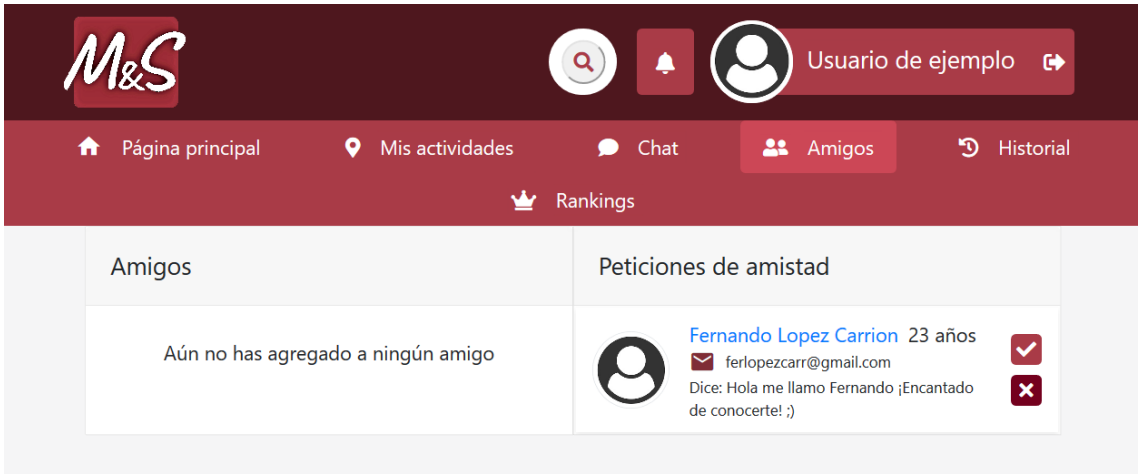


Figura 47. Página para aceptar / rechazar peticiones de amistad

```
@PostMapping("/friend")
public ModelAndView friendResponse(ModelAndView modelAndView, HttpSession session, SessionStatus status,
        @ModelAttribute ("friendRequest") FriendRequestTransfer friendRequest) {
    String error = null;
    UserTransfer userLogged = MySession.getInstance().getUserLogged(modelAndView, session, status);

    Boolean accepted = friendRequest.getAccepted();
    if(friendRequest.getAcceptedStr() != null && !friendRequest.getAcceptedStr().isEmpty()) {
        accepted = (friendRequest.getAcceptedStr().equalsIgnoreCase("true") ? true : false);
        friendRequest.setAccepted(accepted);
    }

    if(userLogged != null && friendRequest != null && userLogged.getUid() != null && friendRequest.getUid() != null && accepted != null) {
        User userLoggedDatabase = null;
        User userFriendRequest = null;
        try {
            userLoggedDatabase = UserServiceImp.getInstance().get(userLogged.getUid());
            userFriendRequest = UserServiceImp.getInstance().get(friendRequest.getUid());

            //User logged and user who ask for friendship exists
            if(userLoggedDatabase != null && userFriendRequest != null && userLoggedDatabase.getActive() && userFriendRequest.getActive()) {
                FriendRequest friendRequestObj = FriendRequestSerializer.transferToDomainObj(friendRequest);
                error = FriendServiceImp.getInstance().resolveFriendRequest(userFriendRequest, userLoggedDatabase, friendRequestObj, accepted);
                if(error == null && accepted != null && accepted) {
                    List<String> registrationTokens = new ArrayList<String>();
                    registrationTokens.add(userFriendRequest.getMessagingToken());
                    String msg = "";
                    msg += MessageUtil.getUserLink(userLoggedDatabase.getUid(), userLoggedDatabase.getName());
                    msg += " ha aceptado tu <a href='/friends'>solicitud de amistad</a>";
                    List<String> uids = new ArrayList<String>();
                    uids.add(friendRequest.getUid());
                    MessageUtil.getInstance().sendNotification(log, registrationTokens, msg, uids);
                }
            }

        } catch (InterruptedException e) {
            error = MySession.databaseErrorStr;
            System.err.println(e.getMessage());
        }

        if(userLoggedDatabase == null || userFriendRequest == null)
            error = MySession.databaseErrorStr;
    }

    RedirectView redirectView = new RedirectView();
    if(error != null)
        MySession.notifyUserModal(redirectView, null, error);
    else
        MySession.notifyUserModal(redirectView, "Amigo añadido", "La petición ha sido aceptada");
    modelAndView.addAllObjects(redirectView.getAttributesMap());

    modelAndView.setViewName("redirect:/friends");

    return modelAndView;
}
```

Código 10. FriendController.java Resolver petición de amistad

Por último, el sistema permitirá a los usuarios eliminar amigos (véase Figura 48), debiendo comprobar primero que ambos usuarios existen y tienen una relación de amistad (Código 11). Tras esto se deshabilitará el chat, y se eliminarán de sus respectivas listas.

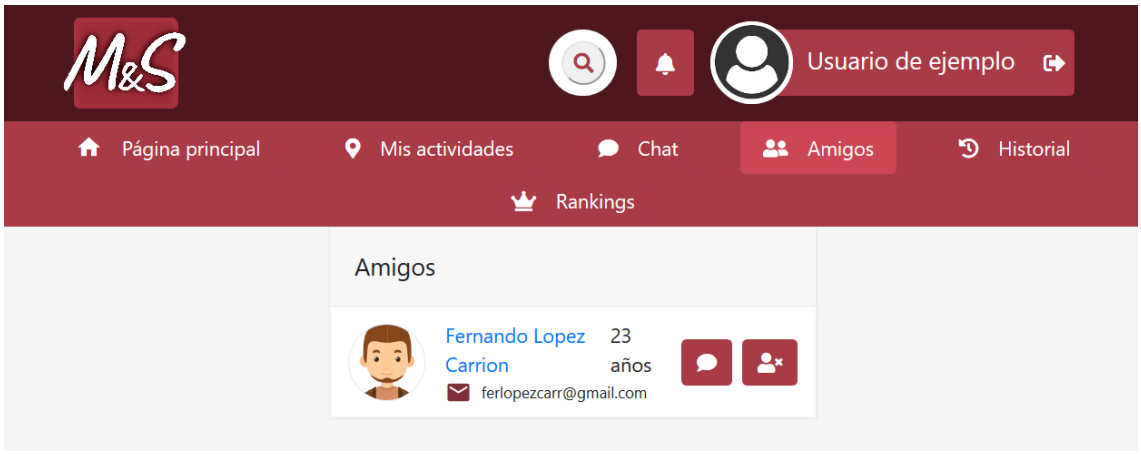


Figura 48. Página para eliminar amigos

```
@Override
public String removeFriend(String userUid, String friendUid) {
    String error = null;
    User userLoggedDatabase = null;
    User friend = null;

    try {
        userLoggedDatabase = UserServiceImp.getInstance().get(userUid);
        friend = UserServiceImp.getInstance().get(friendUid);

        if(friend != null && userLoggedDatabase != null) {

            if(userLoggedDatabase.getFriendsIds() != null && userLoggedDatabase.getFriendsIds().size() > 0) {
                if(userLoggedDatabase.removeFriend(friendUid)) {
                    error = UserServiceImp.getInstance().save(userLoggedDatabase);

                    List<String> tokens = new ArrayList<String>();
                    tokens.add(userLoggedDatabase.getMessagingToken());
                    TopicManagementResponse response = FirebaseMessaging.getInstance().unsubscribeFromTopic(tokens, friend.getUid());
                }
            }

            if((error == null || error.equalsIgnoreCase("")) && friend.getFriendsIds() != null && friend.getFriendsIds().size() > 0) {
                if(friend.removeFriend(userUid)) {
                    error = UserServiceImp.getInstance().save(friend);

                    List<String> tokens = new ArrayList<String>();
                    tokens.add(friend.getMessagingToken());
                    TopicManagementResponse response = FirebaseMessaging.getInstance().unsubscribeFromTopic(tokens, userLoggedDatabase.getUid());
                }
            }
        }
    } catch (InterruptedException | FirebaseMessagingException e) {
        error = MySession.databaseErrorStr;
        System.err.println(e.getMessage());
    }

    if(error == null && userLoggedDatabase == null || friend == null) {
        error = MySession.userNotFoundErrorStr;
    }

    return error;
}
```

Código 11. FriendServiceImp.java Eliminar amigo

7.2.7. Crear actividades

Tanto las personas como las empresas pueden proponer actividades para compartirlas con el resto de usuarios. Esto se hace a través del formulario ubicado en la Figura 49. Para ello los usuarios deben rellenar el formulario con los datos correctos, pudiendo incluir los datos recogidos en la siguiente tabla:

Atributos	Descripción
Nombre*	Solo puede contener palabras con letras
Fecha de Inicio*	Debe encontrarse en fomato “dd/mm/aaaa” y debe ser igual o posterior al día actual

Hora de Inicio*	Debe encontrarse en fomato “hh:mm” y debe ser posterior al momento actual
Fecha de Fin*	Debe encontrarse en el mismo formato que la fecha de inicio y debe ser posterior a esta
Hora de Fin*	Debe encontrarse en el mismo formato que la hora de inicio y debe ser posterior a la fecha y hora de inicio
Número de participantes*	Al menos debe haber dos participantes
Categorías	Lista con las categorías seleccionadas por el usuario que clasificarán la actividad
Descripción	Descripción de la actividad
Ubicación*	Sitio donde tendrá lugar la actividad
Los campos marcados con * son obligatorios	

Tabla 9. Atributos pertenecientes una actividad

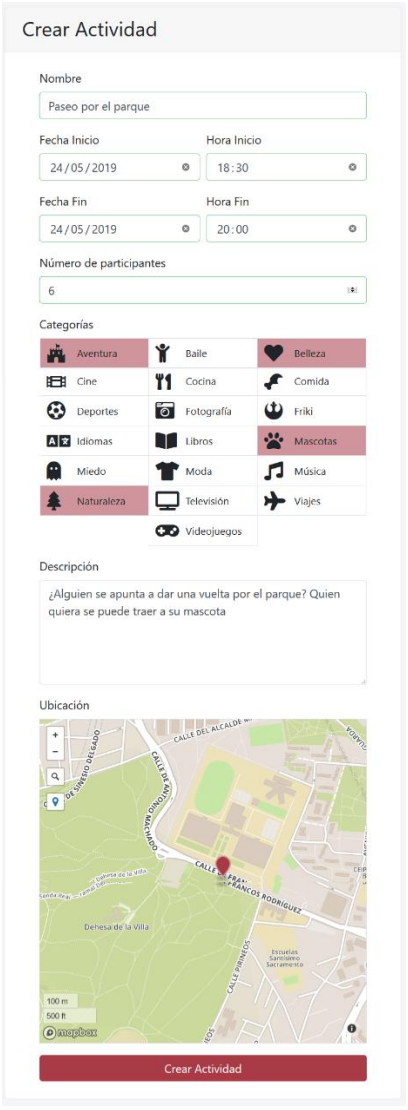


Figura 49. Página para crear una actividad

Una vez se encuentre relleno este formulario y con los datos correctos, la aplicación comprobará si existe el administrador de esta actividad, y en caso afirmativo creará la actividad registrando al usuario en el chat grupal (véase Código 12).

```
@PostMapping("/createActivity")
public ModelAndView createActivityPost(ModelAndView modelAndView, HttpSession session, SessionStatus status,
        @ModelAttribute ("activityTransfer") ActivityTransfer activityTransfer) {
    String error = null;
    UserTransfer userLogged = MySession.getInstance().getUserLogged(modelAndView, session, status);

    if(userLogged != null) {
        Activity activity = ActivitySerializer.transferToDomainObj(activityTransfer);

        User userDatabase = null;
        try {
            userDatabase = UserServiceImp.getInstance().get(userLogged.getUid());
            error = ActivityServiceImp.getInstance().create(userDatabase, activity);

            if(error == null || error.isEmpty()) {
                List<String> registrationTokens = new ArrayList<String>();
                registrationTokens.add(userDatabase.getMessagingToken());

                TopicManagementResponse response = FirebaseMessaging.getInstance().subscribeToTopic(
                    registrationTokens, activity.getUid());

                log.info(response.getSuccessCount() + " tokens were subscribed successfully");

                String actvityLink = MessageUtil.getActivityLink(activity.getUid(), activity.getName());
                List<String> userUids = new ArrayList<String>();
                userUids.add(userDatabase.getUid());
                MessageUtil.getInstance().sendNotification(log, registrationTokens, "Actividad "+actvityLink+" creada", userUids);
            }
        } catch (InterruptedException | FirebaseMessagingException e) {
            error = MySession.databaseErrorStr;
            log.error(e.getMessage());
        }

        if(userDatabase == null || error != null)
            modelAndView.setViewName("redirect:/");
        else {
            modelAndView.setViewName("redirect:/viewActivity");
            modelAndView.addObject("activityUid", activity.getUid());
        }
    }

    RedirectView redirectView = new RedirectView();
    if(error != null) {
        MySession.notifyUserModal(redirectView, null, error);
        modelAndView.addAllObjects(redirectView.getAttributesMap());
    }

    return modelAndView;
}
```

Código 12. ActivityController.java Crear actividad

7.2.8. Modificar información de la actividad

La herramienta permite a sus usuarios modificar las actividades creadas por éstos en el caso de que no hayan empezado (Figura 50). Esto permite a los usuarios rectificar o añadir cualquier dato que no hubiesen contemplado en un inicio. Además de esto se permiten añadir y eliminar imágenes (Figura 51 y 52), funcionando de manera muy similar al caso de uso modificar la imagen de perfil de un usuario.

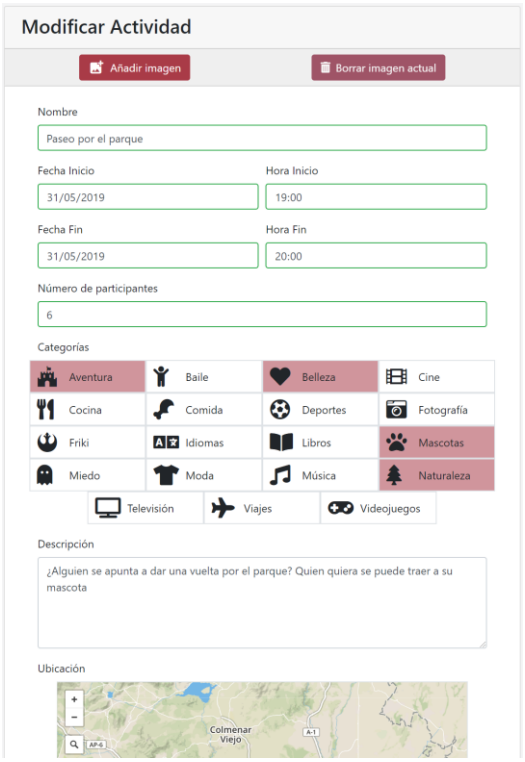


Figura 50. Página para modificar una actividad

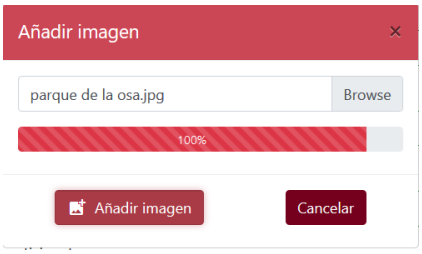


Figura 51. Modal para añadir una imagen a una actividad

Figura 52. Página imagen añadida a una actividad

Con el objetivo de editar los campos de la actividad, se debe rellenar el formulario con los nuevos datos a modificar, comprobar que éstos son

correctos, buscar la actividad, actualizar los campos, guardar los cambios y notificar a los usuarios (véase Código 13).

```
@PostMapping("/editActivity")
public ModelAndView editActivityPost(ModelAndView modelAndView, HttpSession session, SessionStatus status,
        @ModelAttribute ("activityTransfer") ActivityTransfer activityTransfer) {
    String error = null;
    UserTransfer userLogged = MySession.getInstance().getUserLogged(modelAndView, session, status);

    if(userLogged != null) {
        User userDatabase = null;
        try {
            userDatabase = UserServiceImp.getInstance().get(userLogged.getUid());

            if(userDatabase != null) {
                Activity activity = ActivityServiceImp.getInstance().get(activityTransfer.getUid(), true, false);

                if(activity != null && activity.getAdminUid() == userLogged.getUid()) {
                    activity.setName(activityTransfer.getName());

                    if(activityTransfer.getStartDate() == null)
                        activityTransfer.setStartDate(DateUtil.dateCorrectFormat(activityTransfer.getStartDateStr(), activityTransfer.getStartHourStr()));
                    activity.setStartDate(activityTransfer.getStartDate());

                    if(activityTransfer.getEndDate() == null)
                        activityTransfer.setEndDate(DateUtil.dateCorrectFormat(activityTransfer.getEndDateStr(), activityTransfer.getEndHourStr()));
                    activity.setEndDate(activityTransfer.getEndDate());

                    activity.setMaxRegistered(activityTransfer.getMaxRegistered());
                    activity.setDescription(activityTransfer.getDescription());

                    activityTransfer.setUbication(new Ubication(activityTransfer.getLatitude(), activityTransfer.getLongitude()));

                    activity.setUbication(activityTransfer.getUbication());

                    List<Category> categories = new ArrayList<Category>();
                    JSONArray categoriesJSONArray = activityTransfer.getCategoriesJSON();
                    if(activityTransfer.getCategoriesJSON() != null && activityTransfer.getCategoriesJSON().length() > 0) {
                        for(int i = 0; i < categoriesJSONArray.length(); i++) {
                            Category category = Category.getCategoryFromStringValue(categoriesJSONArray.getString(i));
                            if(category != null)
                                categories.add(category);
                        }
                    }
                    activity.setCategories(categories);

                    error = ActivityServiceImp.getInstance().save(userDatabase, activity);

                    if(error == null) {
                        String activityLink = MessageUtil.getActivityLink(activity.getUid(), activity.getName());
                        List<String> registrationTokens = new ArrayList<String>();
                        registrationTokens.add(userLogged.getMessagingToken());
                        List<String> uids = new ArrayList<String>();
                        uids.add(userDatabase.getUid());
                        MessageUtil.getInstance().sendNotification(log, registrationTokens, "Actividad "+activityLink+" modificada", uids);
                    }
                }
                else
                    error = MySession.activityNotFoundatabaseErrorStr;
            }
        } catch (InterruptedException | JSONException e) {
            error = MySession.databaseErrorStr;
            log.error(e.getMessage());
        }

        if(userDatabase == null || error != null)
            modelAndView.setViewName("redirect:/");
        else {
            modelAndView.setViewName("redirect:/viewActivity");
            modelAndView.addObject("activityUid", activityTransfer.getUid());
        }
    }

    RedirectView redirectView = new RedirectView();
    if(error != null)
        MySession.notifyUserModal(redirectView, null, error);
    else
        MySession.notifyUserModal(redirectView, "Actividad", "Cambios guardados satisfactoriamente");
    modelAndView.addAllObjects(redirectView.getAttributesMap());

    return modelAndView;
}
```

Código 13. ActivityController.java Editar actividad

7.2.9. Inscribirse / desinscribirse en una actividad

Siempre que la actividad no haya empezado, los usuario podrán inscribirse o desincribirse. Para ello el usuario deberá encontrar el evento en el que desea inscribirse (Figura 53).

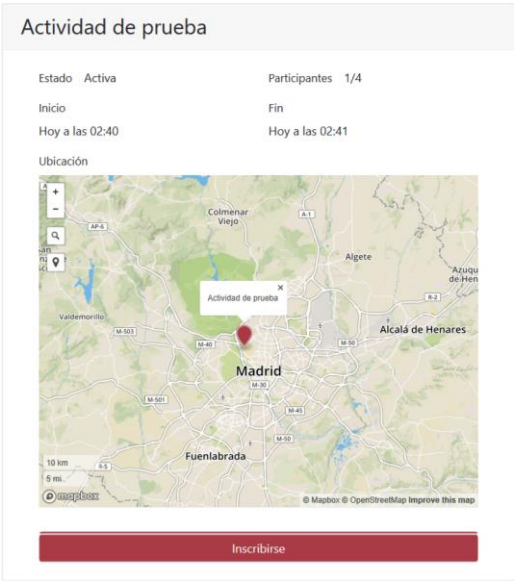


Figura 53. Página de una actividad a la que inscribirse

Una vez el usuario haya pulsado en el botón de registro, el sistema comprobará que el usuario y la actividad existen y se encuentran activos. Adicionalmente, la actividad no debe haber empezado, debe tener plazas disponibles, y el usuario no debe encontrarse inscrito previamente (véase Código 14). Tras esto, el usuario quedará inscrito en la actividad, siendo notificado de ello. Para desinscribirse del evento se sigue la misma fórmula.

```
userDatabase = UserServiceImp.getInstance().get(userLogged.getUid());
activity = ActivityServiceImp.getInstance().get(activityUid, true, false);
if(userDatabase != null) {
    if(activity != null && activity.getActive() && !activity.activityFinished()) {
        if(userDatabase.addRegisteredActivity(activity.getUid())) {
            Integer seats = activity.avaliableSeats();
            if(seats != null && seats > 0) {
                error = UserServiceImp.getInstance().save(userDatabase);

                if(error != null)
                    error = MySession.databaseErrorStr;
                else {
                    activity.addUser(userDatabase.getUid());
                    error = ActivityServiceImp.getInstance().save(userDatabase, activity);

                    if(error == null || error.isEmpty()) {
                        List<String> registrationTokens = new ArrayList<String>();
                        registrationTokens.add(userDatabase.getMessagingToken());

                        TopicManagementResponse response = FirebaseMessaging.getInstance().subscribeToTopic(
                            registrationTokens, activity.getUid());

                        String userText = MessageUtil.getUserLink(userDatabase.getUid(), userDatabase.getName());
                        String activityText = MessageUtil.getActivityLink(activity.getUid(), activity.getName());

                        log.info(response.getSuccessCount() + " tokens were subscribed successfully");

                        MessageUtil.getInstance().sendActivityMessage(log, registrationTokens,
                                "Se ha unido "+userDatabase.getName(), activityUid, userDatabase.getUid(), "null");

                        User admin = UserServiceImp.getInstance().get(userLogged.getUid());

                        if(admin != null) {
                            registrationTokens = new ArrayList<String>();
                            registrationTokens.add(admin.getMessagingToken());

                            List<String> uids = new ArrayList<String>();
                            uids.add(userDatabase.getUid());

                            MessageUtil.getInstance().sendNotification(log, registrationTokens,
                                    "El usuario "+userText+" se ha inscrito en la actividad "+activityText, uids);
                        }
                    }
                }
            }
            else
                error = "No hay sitios disponibles";
        }
        else
            error = "Ya estás inscrito a la actividad";
    }
    else
        error = MySession.activityNotFoundatabaseErrorStr;
}
else
    error = MySession.userNotFoundErrorStr;
```

Código 14. ActivityController.java Inscribirse en una actividad

7.2.10. Mapa de actividades

Mediante la interacción con el mapa (Figura 54), el usuario pordrá encontrar actividades cercanas a su ubicación, así como eventos que se ajusten a sus intereses. Esto es implementado mediante Mapbox (véase sección 3.1.10), utilizando una herramienta de gelocalización que en caso de que se hayan dado los permisos pertienentes, es capaz de situar al usuario en el mapa, calculando la distancia de éste con las actividades.

A su vez, se ofrece un filtro de actividades para adecuarse a las preferencias y necesidades de cualquier usuario. A través de estos, se permite especificar la fecha de creación, la distancia máxima en kilómetros, los sitios disponibles, y las categorías en las que se clasifiquen las actividades.

De manera análoga, en la interfaz se ubica el botón de crear actividades para permitir a los usuarios proponer nuevos planes.

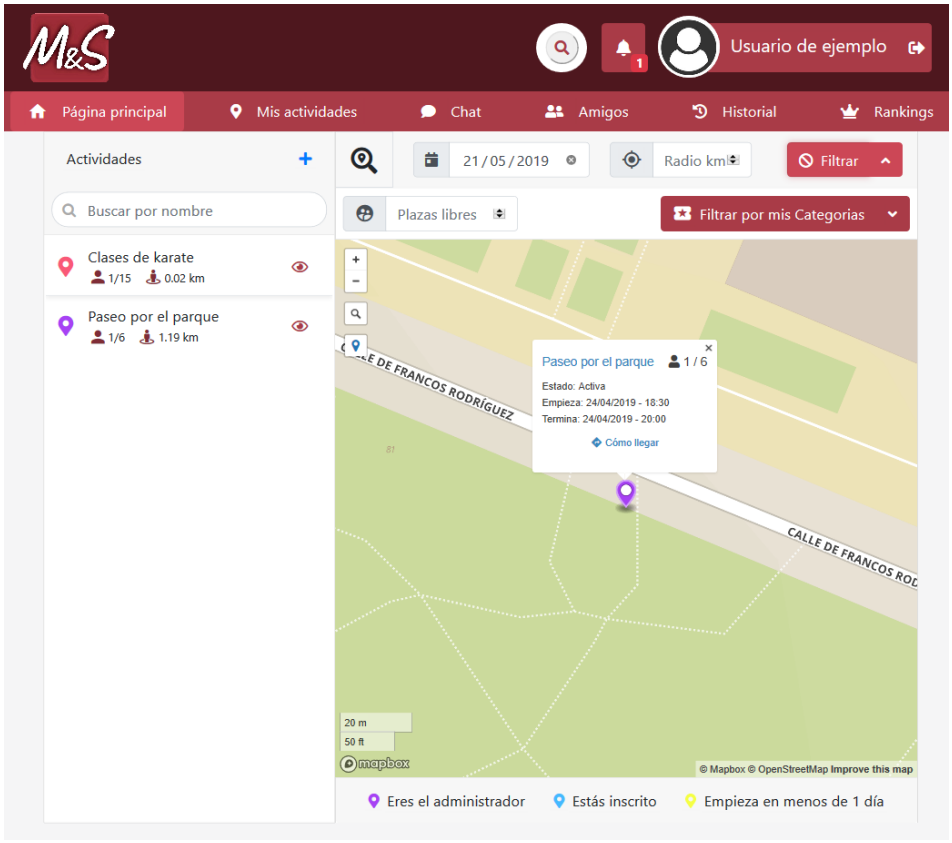


Figura 54. Página principal con el mapa de la aplicación

8. Conclusiones y trabajo futuro

En este capítulo se recogen las conclusiones extraídas del desarrollo de este proyecto, así como el trabajo y funcionalidad futura que queda pendiente de implementar.

8.1. Conclusiones

Las principales ventajas que caracterizan a la aplicación es que se trata de una plataforma gratuita, intuitiva, original, y que emplea tecnologías muy conocidas y extendidas, con una base de datos externa que consigue dar robustez al sistema. Al ser una herramienta desplegada en la nube presenta las ventajas e inconvenientes de este tipo de aplicaciones:

Ventajas:

- Acceso desde cualquier dispositivo y lugar, en cualquier momento
- Tecnología siempre actualizada, cumpliendo los más altos estándares de seguridad
- Ahorro en el mantenimiento técnico (no requiere de hardware, ni la configuración y mantenimiento del mismo)
- Capacidad de ampliación y crecimiento casi inmediata e ilimitada

Inconvenientes:

- Necesidad de conexión a internet
- Limitación en la gestión de la base de datos a las herramientas y plataformas de Firebase
- Capacidad máxima de las peticiones restringida al gasto que el dueño de la aplicación esté dispuesto a realizar, pues una vez la herramienta supere el número límite de peticiones, Google requerirá de una inversión monetaria para que el proyecto continúe funcionando

Como resultado final, se ha conseguido desarrollar una herramienta que consigue cumplir su objetivo, conectar a personas con intereses en común que buscan pasar parte de su tiempo libre desarrollando actividades culturales, deportivas, sociales, etc.

8.2. Trabajo futuro

A continuación, se detallan posibles mejoras o cambios que no han sido posibles de implementar, pero que aportarían un gran valor a la aplicación.

- Implementar versiones para IOS y Android que recojan el mismo nivel de funcionalidad que la desarrollada en el proyecto.

- Desarrollar una herramienta inteligente que permita sugerir nuevas actividades en base a los eventos en los que ha participado el usuario y sus categorías.
- Añadir captchas y confirmaciones de email que aseguren y verifiquen que los usuarios registrados son personas reales.
- Añadir visibilidad a los campos de los usuarios para que éstos puedan decidir qué información comparten con el resto.
- Implementar la creación de grupos de usuarios para que éstos puedan ponerse en contacto y puedan organizar nuevas actividades siendo una misma entidad.
- Modificar las actividades para que se permita añadir información adicional como el precio, condiciones de inscripción / desinscripción, eventos relacionados, etc.
- Añadir confirmaciones y cancelaciones de asistencia a las actividades de manera que se elimine o reduzca la incertidumbre existente al acudir a un evento y desconocer qué usuarios se presentarán.
- Mostrar el estado del usuario (Desconectado / En línea / Ocupado / Ausente), borrar mensajes enviados en los últimos 5 minutos, eliminar conversaciones antiguas, enviar mensajes con emoticonos, y adjuntar cualquier clase de contenido multimedia a los mismos con el objetivo de aumentar la riqueza del chat.
- Permitir reportar comportamientos fuera de lugar, contenido inadecuado y bloquear usuarios con el fin de que no se puedan recibir más mensajes, no puedan enviarse peticiones de amistad, ni puedan inscribirse a las actividades del usuario.

8. Conclusions and future work

This chapter includes the conclusions drawn from the development of this project, as well as the work and future functionality that remains to be implemented.

8.1. Conclusions

The application presents the advantage of being free, intuitive, and an original platform. It uses well-known and extended technologies, with an external database that gives robustness to the system, because if a failure occurs in the server, the application would still be accessible through the mobile application.

Being a tool deployed in the cloud presents the advantages and disadvantages of this type of applications:

Advantages:

- Acces from any device and place, at any time
- Technology always updated, meeting the highest safety standards
- Savings in technical maintenance (does not require hardware, configuration or maintenance)
- Capable of expanding and growing almost immediately and unlimited

Disadvantages:

- Internet connection needed
- Limited database management to Firebase tools and platforms
- Maximum capacity of requests restricted to the expense that the owner of the application is willing to make, because once the tool exceeds the limit number of requests, Google will require a monetary investment for the project to continue working

As a result, it has been possible to develop a tool that achieves its objective, connecting people with common interests who wants to spend part of their free time developing cultural, sports, social activities, etc.

8.2. Future work

Below are possible improvements or changes that have not been possible to implement, but they would bring great value to the application.

- Implement versions for IOS and Android that reflect the same level of functionality as that developed in the project.
- Develop an intelligent tool that allows suggesting new activities based on the events in which the user and its categories have participated.
- Add captchas and email confirmations that ensure and verify that the registered users are real people.

- Add visibility to users' fields so that they can decide what kind of information share with the others.
- Modify the activities data in order to allow adding new information such as price, registration / unregistration conditions, related events, etc.
- Add assistance checks to the activities in a way that delete or decrease the existing uncertainty when attending to an event and ignore which users will come.
- Show the user state (Disconnected, Online / Busy / Away), delete messages sent in the las 5 minutes, remove old conversations, send emojis and any kind of multimedia content in order to build a rich chat platform.
- Allow reporting misplaced behavior, inappropriate content, and block annoying users so that they can't send more friend requests, messages or inscribe in user's activities.

Bibliografía

[1] «freelancermap,» [En línea]. Available: https://www.freelancermap.com/freelancer-tips/es/12116-meetup-freelancers.

[2] J. Pastor. [En línea]. Available: https://www.javipastor.com/como-hacer-networking-en-tu-ciudad-con-meetup/.

[3] «El País,» [En línea]. Available: https://elpais.com/tecnologia/2004/12/27/actualidad/1104139679_850215.html.

[4] «Meetup,» [En línea]. Available: https://www.meetup.com/es-ES/.

[5] «ABC,» [En línea]. Available: https://www.abc.es/tecnologia/moviles/aplicaciones/abci-fever-espanola-vuelve-locos-fiesteros-londinenses-201807051116_noticia.html.

[6] «Expansion,» [En línea]. Available: http://www.expansion.com/economia-digital/companias/2018/08/01/5b603294ca4741af778b4642.html.

[7] «Ferver,» [En línea]. Available: https://feverup.com/madrid.

[8] «Android Spain,» [En línea]. Available: https://androidspain.es/geokeda-deporte-ocio-y-mucho-mas-con-gente-de-tu-ciudad/.

[9] «Geokeda,» [En línea]. Available: https://geokeda.es/.

[10] «Timpik,» [En línea]. Available: https://www.timpik.com/.

[11] «Xakata Android,» [En línea]. Available: https://www.xatakandroid.com/aplicaciones-android/aplicaciones-android-deportes-timpik.

[12] «Timpik,» [En línea]. Available: https://www.timpik.com/es/partidosProvinciaDeporte/28-madrid.

[13] Mozilla, «MDN web docs mozilla,» [En línea]. Available: https://developer.mozilla.org/es/docs/Web/HTML.

[14] «codigofacilito,» [En línea]. Available: https://codigofacilito.com/articulos/que-es-html.

[15] Bootstrap Team, «Bootstrap,» 1 5 2019. [En línea]. Available: https://getbootstrap.com/.

[16] W3C, «World Wide Web Consortium,» [En línea]. Available: https://www.w3.org/Style/CSS/.

[17] «FontAwesome,» [En línea]. Available: https://fontawesome.com/icons.

[18] Google, «Material Design,» [En línea]. Available: https://material.io/design/introduction/#principles.

[19] «MDN web docs mozilla,» [En línea]. Available: https://developer.mozilla.org/es/docs/Web/JavaScript.

[20] «w3schools,» [En línea]. Available: https://www.w3schools.com/jsref/default.asp.

[21] «JQuery,» [En línea]. Available: https://jquery.com/.

[22] «MDN web docs mozilla,» [En línea]. Available: https://developer.mozilla.org/es/docs/Glossary/jQuery.

[23] «w3schools,» [En línea]. Available: https://www.w3schools.com/jquery/jquery_ref_ajax.asp.

[24] «DataTables,» [En línea]. Available: https://datatables.net/.

[25] «Moment.js,» [En línea]. Available: https://momentjs.com/.

[26] «Mapbox,» [En línea]. Available: https://www.mapbox.com/.

[27] «Spring.io,» [En línea]. Available: https://spring.io/.

[28] «Apache Maven,» [En línea]. Available: https://maven.apache.org/.

[29] «Genbeta,» [En línea]. Available: https://www.genbeta.com/desarrollo/que-es-maven.

[30] «Thymeleaf.org,» [En línea]. Available: https://www.thymeleaf.org/.

[31] «Arpen Technologies,» [En línea]. Available: https://arpentechnologies.com/es/blog/aplicaciones-movil/que-es-firebase-y-que-nos-aporta/.

[32] «openwebinars,» [En línea]. Available: https://openwebinars.net/blog/que-es-git-y-para-que-sirve/.

[33] «Spring.io,» 20 2 2019. [En línea]. Available: https://spring.io/tools.

[34] «Visual Studio Code,» 7 4 2019. [En línea]. Available: https://code.visualstudio.com/.

[35] «Android Developers,» 07 04 2019. [En línea]. Available: https://developer.android.com/studio.

[36] «Firebase Docs,» 7 04 2019. [En línea]. Available: https://firebase.google.com/docs/?authuser=0.

[37] «Heroku,» [En línea]. Available: https://www.heroku.com/.

[38] «Platzi,» [En línea]. Available: https://platzi.com/blog/que-es-heroku-y-para-que-me-sirve/.

[39] «ElAndroidLibre,» [En línea]. Available: https://elandroidelibre.elespanol.com/2014/11/que-es-material-design.html.

1. Anexo I: Guía de uso

El propósito de esta sección en guiar al usuario a través de la aplicación, de manera que pueda explorar su funcionalidad principal. Debido a esto, la estructura de esta guía se basa en los actores que interactúen con la herramienta.

1.1. Usuarios no identificados

Esta subsección se enfoca a la funcionalidad accesible para los usuarios que no se encuentran registrados en la aplicación.

1.1.1. Página principal

Lo primero que ve el usuario es la página principal (Captura 1) de la aplicación donde se le informa del uso que esta tiene.

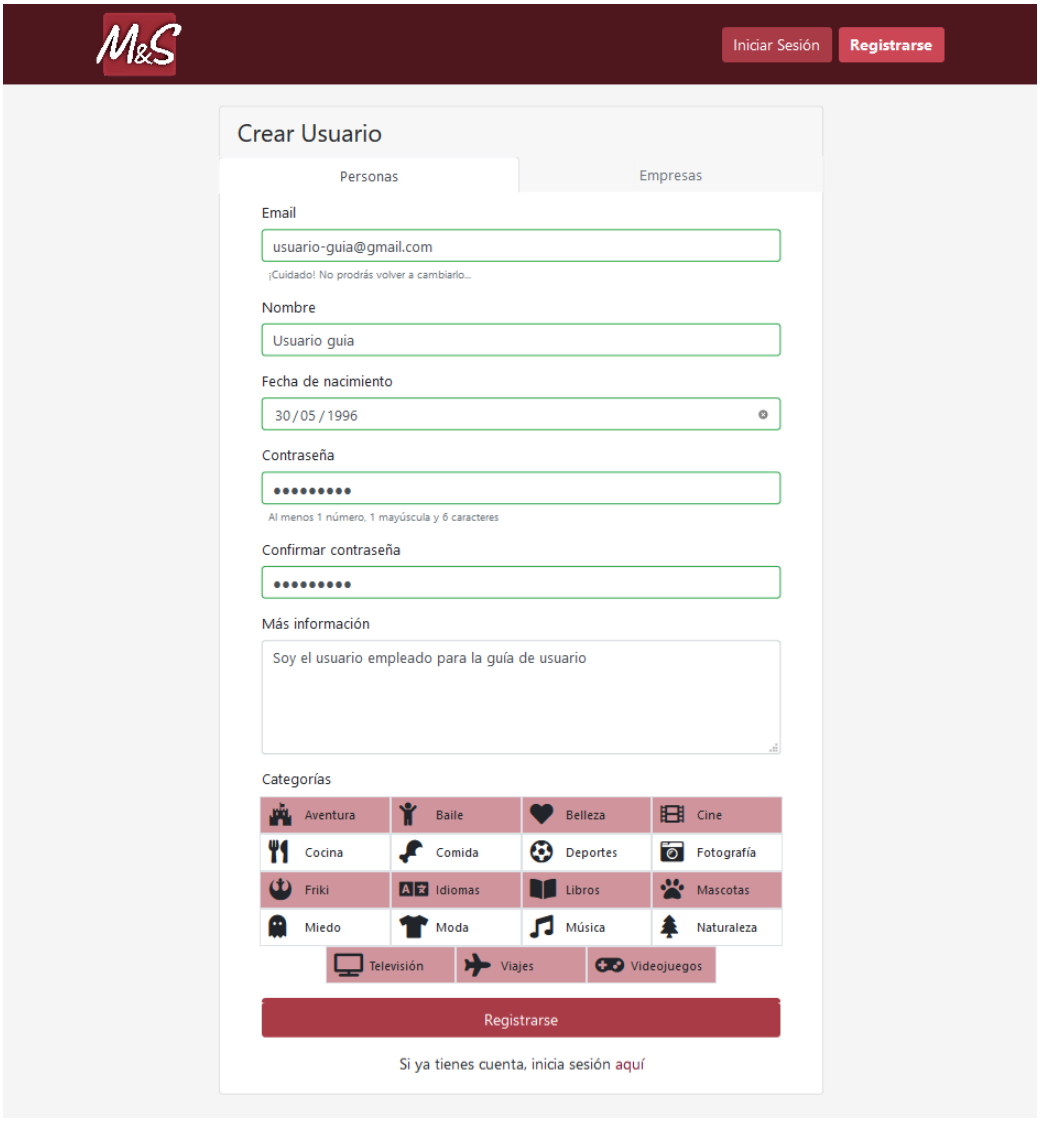
Captura 1. Interfaz de la página principal

En la parte inferior de la pantalla se encuentra el pie de página de la aplicación que aparecerá en cada una de las interfaces, este nos ofrece información sobre las redes sociales auxiliares de la aplicación, y un formulario de contacto con la herramienta. De manera análoga, nos encontramos un enlace al formulario de feedback de la aplicación que los usuarios testers han rellenado, así como la información del desarrollador de la aplicación.

1.1.2. Registro

Para poder acceder a la aplicación, es necesario que el usuario se registre a través de la página que aparece en la Captura 2.

Para ello el usuario debe rellenar los datos correctamente, y presionar el botón de registro.

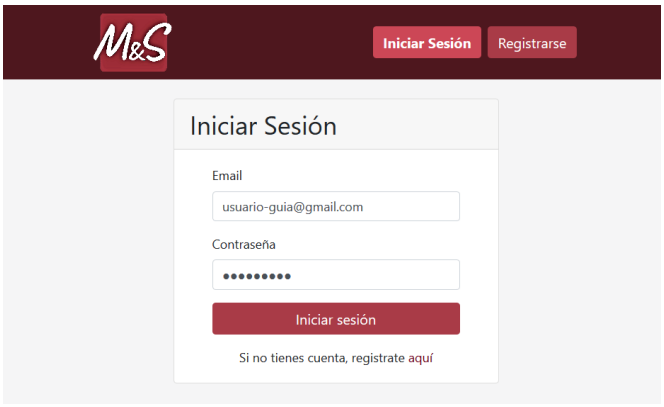


Captura 2. Interfaz de registro de personas y empresas

1.1.3. Iniciar sesión

Una vez el usuario se ha registrado con éxito, el sistema inicia la sesión automáticamente. No obstante, si el usuario ha cerrado la sesión, se ha perdido, o a cambiado de navegador; éste debe de introducir su email y contraseña correspondiente (véase Captura 3).

Tras esto se redirigirá a la página de inicio del usuario.



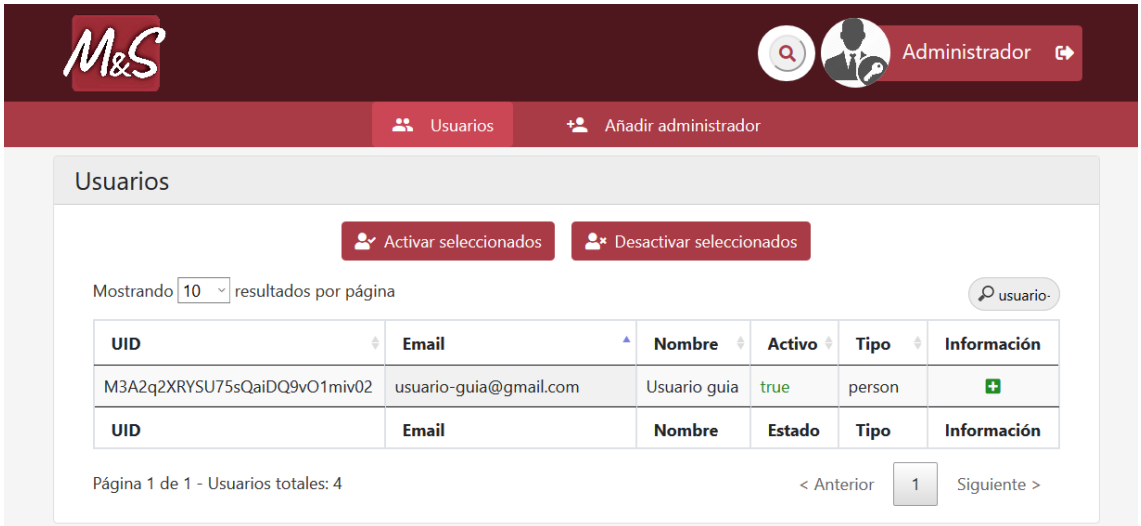
Captura 3. Interfaz de inicio de sesión

1.2. Usuarios registrados

Una vez se ha llevado a cabo el inicio de sesión, todos los usuarios registrados por el hecho de serlo comparten una misma funcionalidad:

1.2.1. Buscador

Los administradores tienen la capacidad de a los usuarios a través del buscador integrado en la tabla que se observa en la Captura 4.



Captura 4. Interfaz de búsqueda de usuarios por parte de un Administrador

Las personas o empresas podrán implementar la misma funcionalidad a través del buscador ubicado en la la barra superior, pudiendo obtener resultados tanto de actividades si se consulta el nombre, o usuarios si se introduce el nombre o email (véase Captura 5).

Captura 5. Interfaz búsqueda de usuarios realizada por una Persona o Empresa

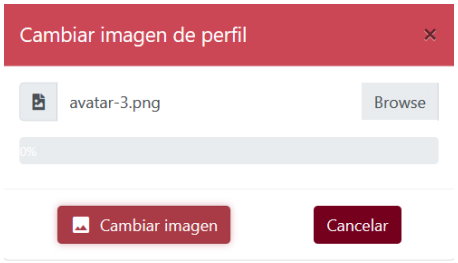
1.2.2. Ver Perfil

A través de la página mostrada en la Captura 6, puede consultarse el perfil de un usuario, mostrándose el botón de modificar en el caso de que se trate del mismo, o añadir amigo en el caso de que ambos usuarios no lo sean. De manera adicional, se podrán consultar las valoraciones realizadas por los usuarios inscritos en las actividades propuestas por las personas o empresas.

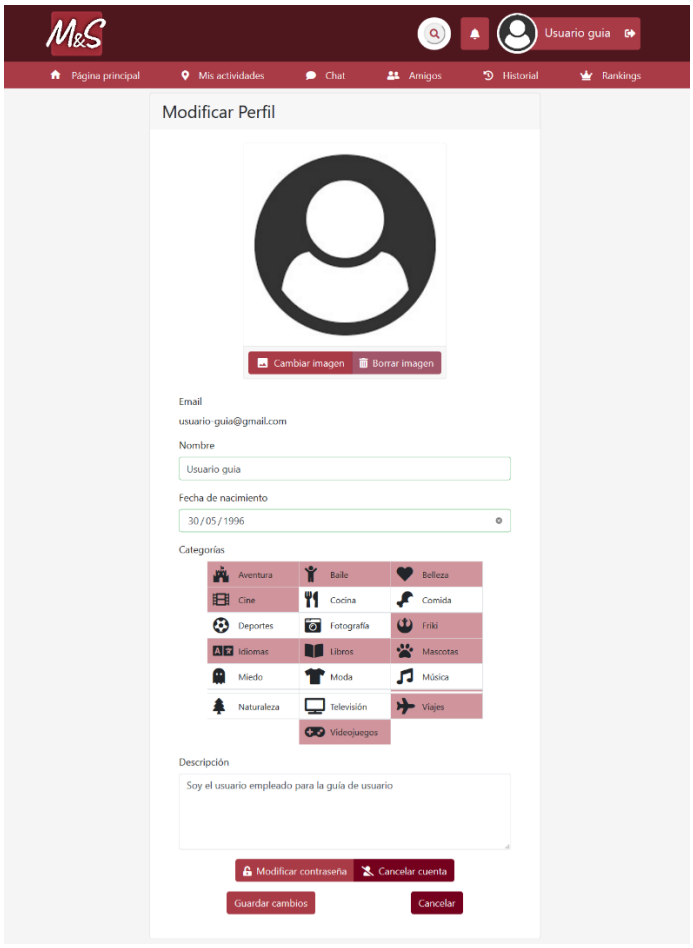
Captura 6. Interfaz ver perfil de usuario

1.2.3. Modificar perfil

Los usuarios registrados pueden modificar su foto (Captura 7), nombre, fecha de nacimiento, categorías y descripción (Captura 8). Una vez se hayan realizado dichos cambios, en necesario hacer click en el botón Guardar cambios para que éstos tengan efecto, en caso contrario pueden cancelarse.



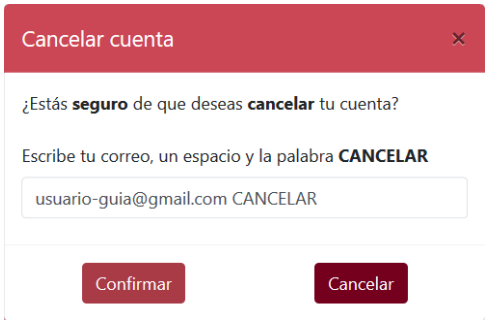
Captura 7. Interfaz para cambiar la imagen de perfil



Captura 8. Interfaz para modificar el perfil de un usuario

También se le permite al usuario modificar su contraseña introduciendo la contraseña anterior y una nueva, como puede apreciarse en la Captura 9. Por último el usuario es capaz de cancelar la cuenta (Captura 10) introduciendo su email seguido de un espacio y la palabra CANCELAR

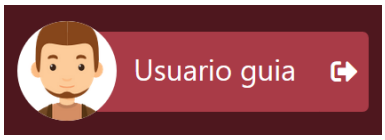
Captura 9. Interfaz para modificar la contraseña



Captura 10. Interfaz para cancelar la cuenta

1.2.4. Cerrar sesión

A través de esta funcionalidad el usuario podrá cerrar la sesión actual (véase Captura 11), eliminando al usuario de la misma, y redirigiendo a la página principal de la aplicación



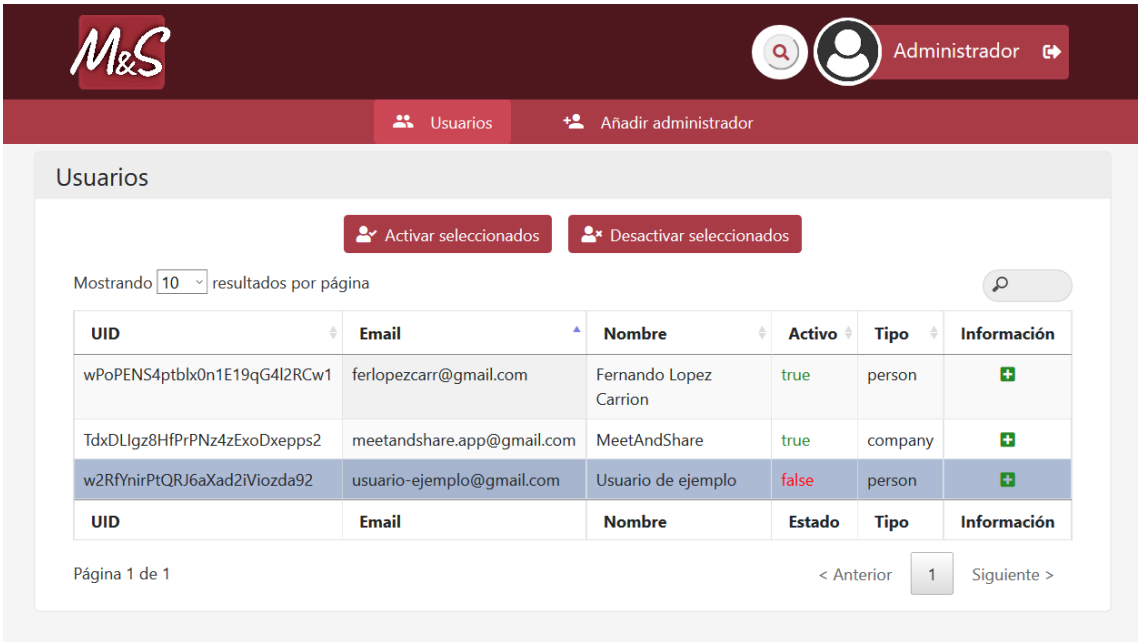
Captura 11. Interfaz para cerrar la sesión

1.3. Administradores

Los administradores son los encargados de gestionar las cuentas de usuario registradas en la aplicación, pudiendo ver los detalles de éstos, activarlos o desactivarlos a voluntad, y dar de alta nuevos administradores.

1.3.1. Activar / desactivar usuarios

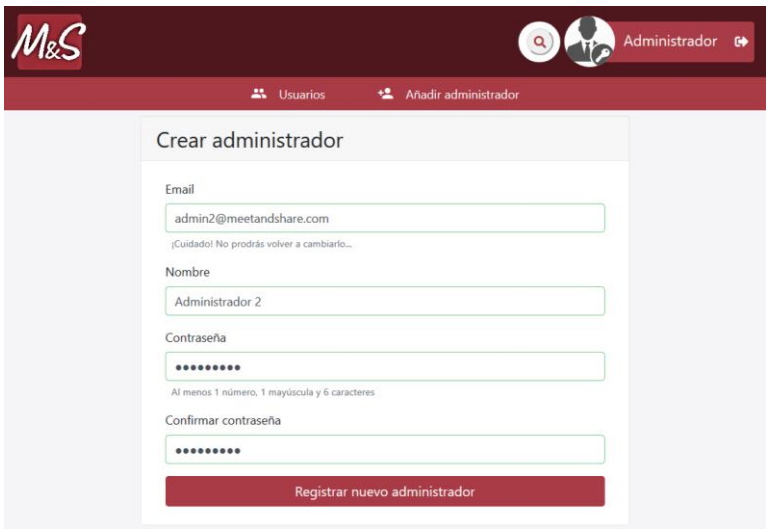
Para ello, se seleccionarán los usuarios afectados haciendo click en las filas correspondientes, y se ejecutará la operación indicada (véase Captura 12).



Captura 12. Interfaz para la gestión de usuarios de los Administradores

1.3.2. Dar de alta nuevos administradores

Con el objetivo de dar de alta a un nuevo administrador en la aplicación, los administradores deberán rellenar el formulario mostrado en la Captura 13, y hacer click en el botón de registro una vez se hayan adjuntado los datos correctos.



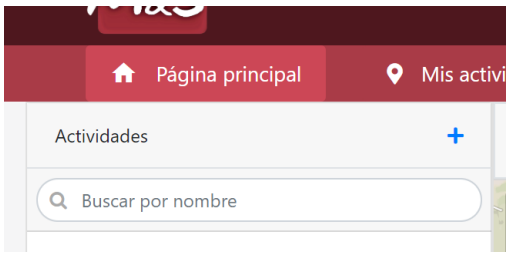
Captura 13. Interfaz de creación de nuevos administradores

1.4. Personas y Empresas

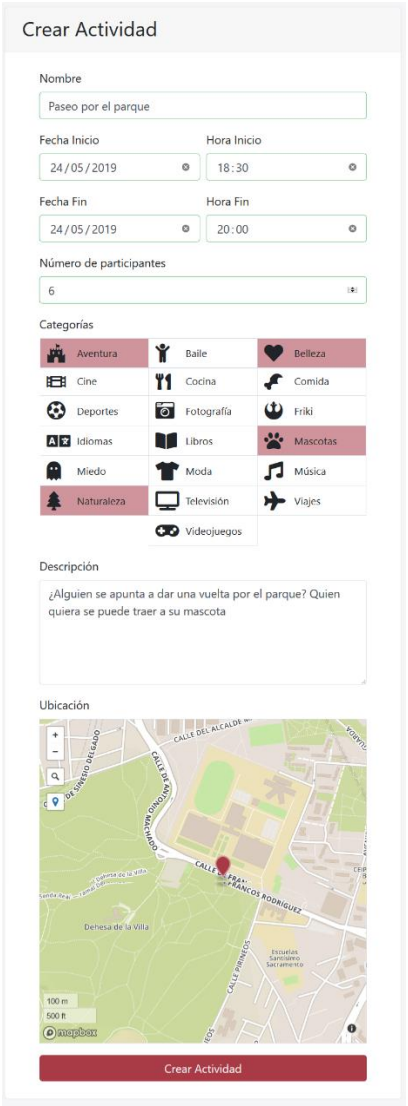
En esta subsección, se expondrá la funcionalidad accesible a estos usuarios.

1.4.1. Crear actividad

A través de la página principal mostrada en la Captura 14, las personas como las empresas son capaces de proponer actividades con el objetivo de divertirse y compartir su tiempo con otros usuarios. Para ello los usuarios deben rellenar el formulario correspondiente de la Captura 15 con la ubicación propuesta, el nombre, fecha de inicio y fin, participantes categorías y descripción.



Captura 14. Interfaz que muestra el botón para crear una actividad

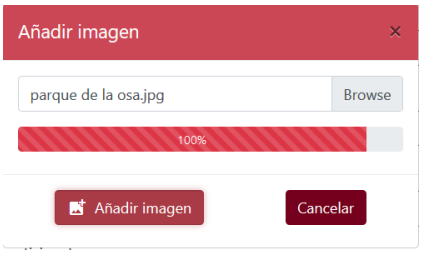


Captura 15. Interfaz de creación de actividades

1.4.2. Modificar actividad

Las personas y empresas podrán modificar sus actividades propuestas siempre y cuando éstas no hayan comenzado. Por un lado, el usuario podrá modificar todos los datos de la actividad (Captura 16), así como añadir imágenes adjuntando un archivo (como en la Captura 17), o seleccionando la imagen del carrousel a eliminar mediante el botón borrar imagen actual (Captura 17).

Captura 16. Interfaz para modificar una actividad

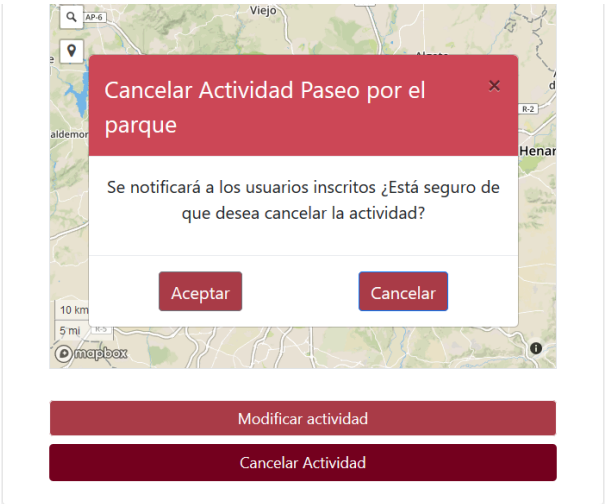


Captura 17. Interfaz para añadir una imagen a una actividad

Figura 55. Página imagen añadida a una actividad

1.4.3. Cancelar actividad

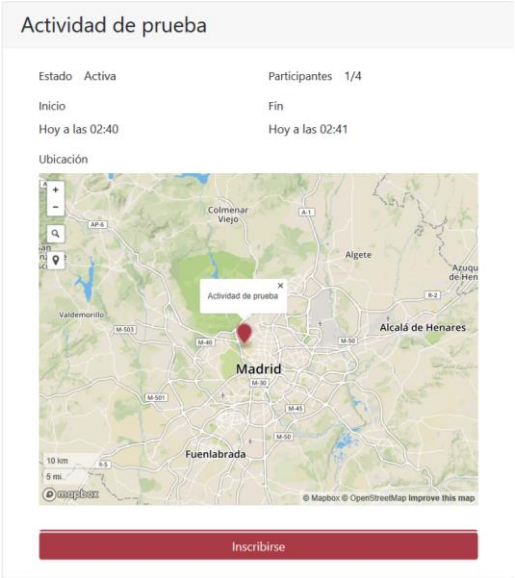
Si la actividad no ha comenzado el usuario puede cancelarla presionando en el botón de cancelar actividad situado al final de la visualización de una actividad propia, siendo necesario la confirmación de este proceso (véase Captura 18) e implicando que ésta decisión se notifique a los usuarios registrados.



Captura 18. Interfaz para cancelar una actividad propia

1.4.4. Inscribirse / desinscribirse en una actividad

Las personas y empresas también pueden inscribirse o desincribirse de las actividades propuestas por otros usuarios, siempre que estas no hayan comenzado. Para ello el usuario deberá seleccionar el botón correspondiente situado al final de la interfaz de visualización del evento.

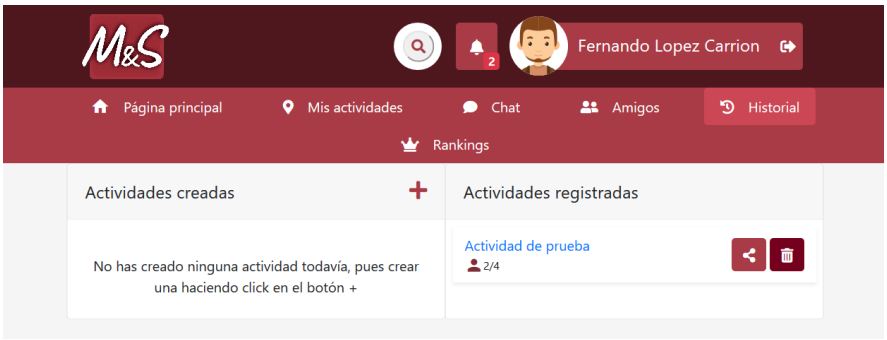


Captura 19. Interfaz para inscribirse / desinscribirse de una actividad

1.4.5. Valoraciones

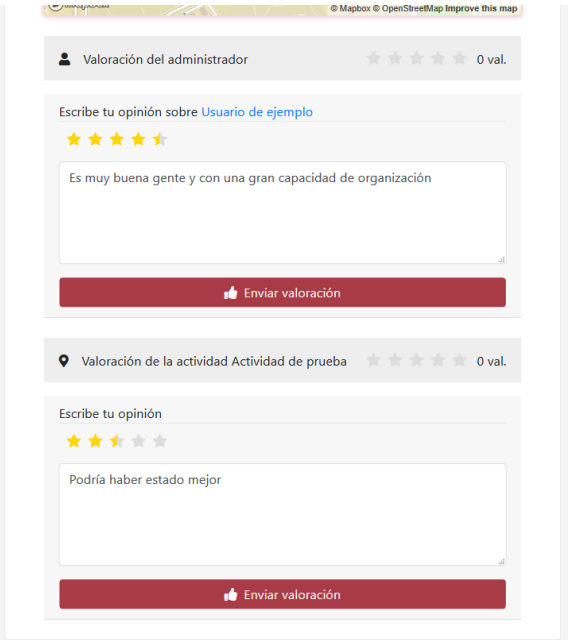
Una vez el usuario se ha inscrito en una actividad y ha participado, el sistema le permite dejar un comentario sobre la actividad y su administrador, de manera que los futuros usuarios puedan hacerse una idea previa de cómo serán los eventos propuestos por este administrador.

La forma más rápida es a través del historial, donde se muestran las actividades en las que el usuario ha participado (Captura 20).



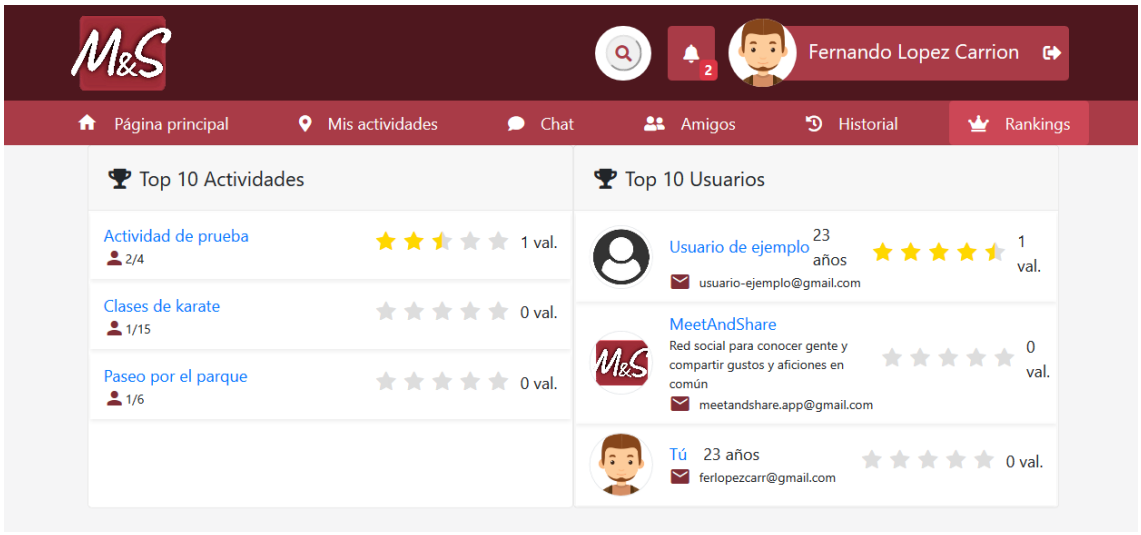
Captura 20. Interfaz donde se muestra el historial de actividades

Para valorar una actividad o administrador el usuario debe seleccionar la puntuación de la actividad, pudiendo dejar un comentario opcional, y hacer click en el botón enviar valoración (véase Captura 21).



Captura 21. Interfaz para valorar una actividad y a su administrador

Tras esto, el usuario podrá observar como su valoración afecta a la actividad y administrador a través del ranking (Captura 22).



Captura 22. Interfaz que muestra el ranking de actividades y usuarios

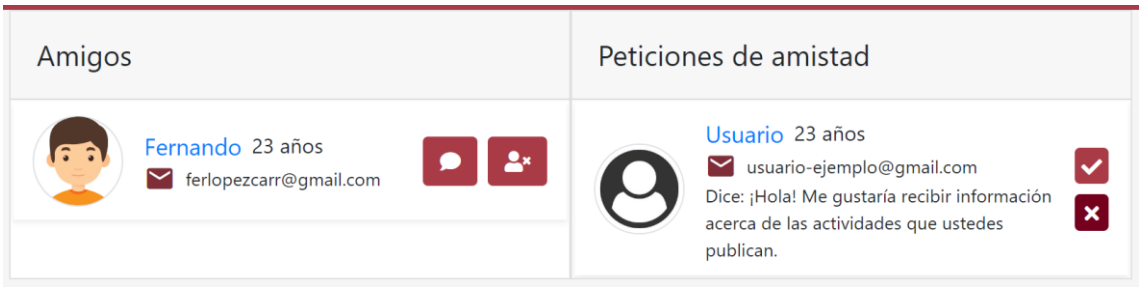
1.4.6. Amigos

La aplicación también permite a sus usuarios establecer relaciones de amistad para compartir las actividades que parezcan más interesantes y mantenerse en contacto.

Los usuarios serán capaces de añadir a otro como amigo a través de su perfil, o buscándolos directamente en la barra superior (ver sección 1.2.1 de este capítulo). Estos harán click el botón de añadir amigo pudiendo dejar adicionalmente un comentario (véase Captura 23).

Captura 23. Interfaz para añadir un amigo

Una vez enviada la petición, el usuario objetivo será el encargado de resolverla (Captura 24).



Captura 24. Interfaz para resolver una petición de amistad

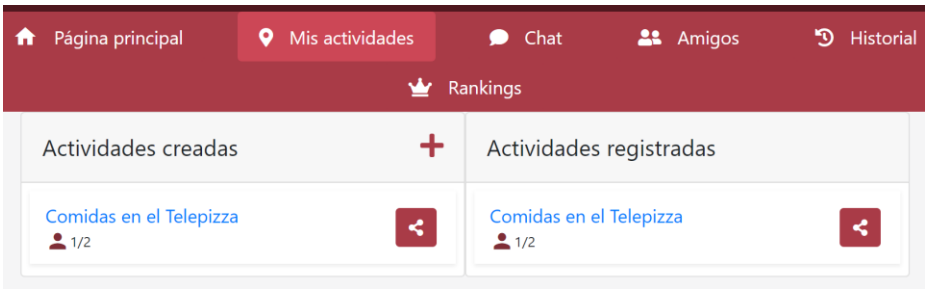
Tras eso, en caso de que se haya confirmado la petición, ambos usuarios aparecerán en sus respectivas listas de amigos.

1.4.7. Compartir actividades

Los usuarios podrán compartir actividades con sus amigos, de manera que éstos puedan unirse y asistir juntos al evento.

Para ello las personas o empresas deberán acceder a su lista de actividades (véase Captura 25). A través del botón de compartir, se podrán seleccionar

los usuarios a los que se les enviará la actividad (Captura 26), confirmando la operación mediante el botón de compartir actividad, y notificando a cada uno de ellos.

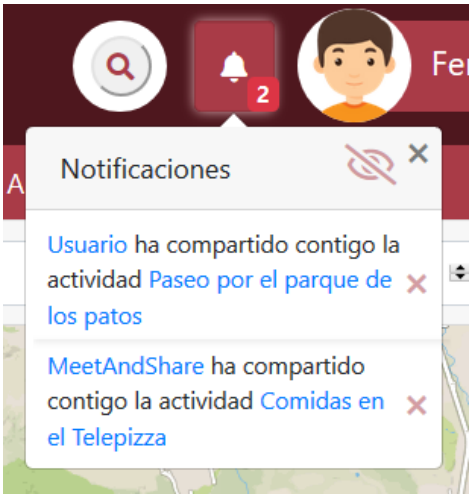


Captura 25. Interfaz que muestra la lista de actividades creadas y registradas por un usuario

Captura 26. Interfaz para compartir actividades con otros usuarios

1.4.8. Notificaciones

Los usuarios podrán ver las notificaciones recibidas a través del panel de avisos (Captura 27).



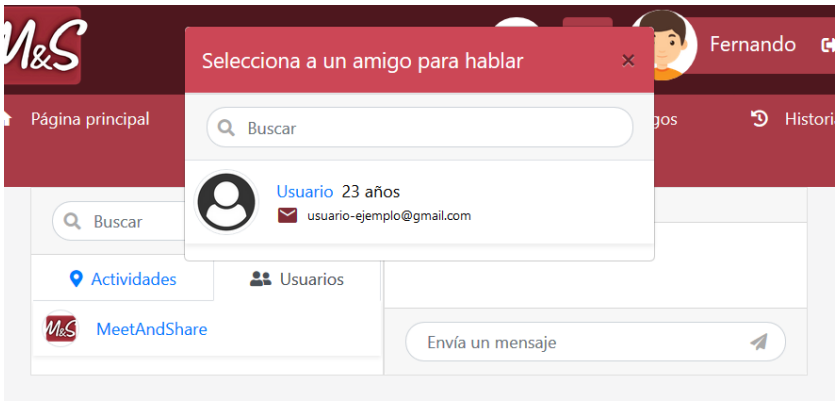
Captura 27. Interfaz de notificaciones

De forma adicional podrán eliminar las notificaciones pendientes haciendo click sobre el icono con forma de cruz para eliminar el aviso seleccionado, o mediante el botón representado con un ojo tachado para marcar todas como vistas.

1.4.9. Chat

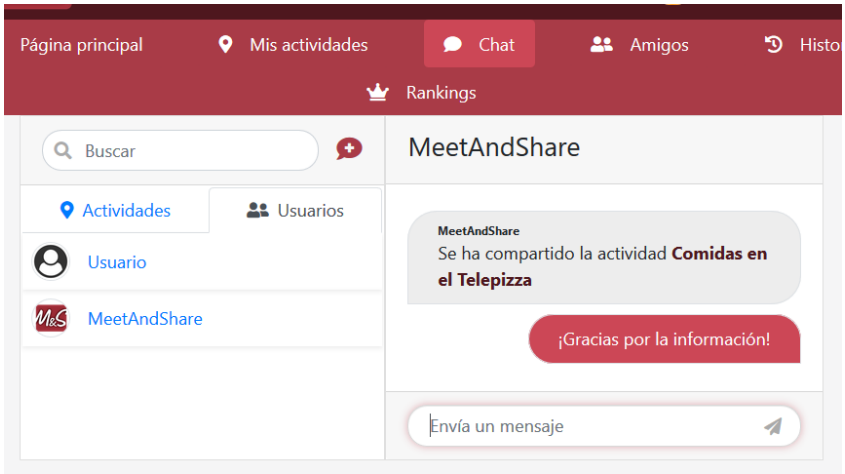
Por último, los usuarios pueden mantenerse en contacto a través del chat, donde aparecerán las actividades en las que se ha inscrito el usuario, así como los amigos con los que este ha conversado.

Para enviar un nuevo mensaje a un amigo puede seleccionarse la conversación directamente, o abrir una conversación a través del botón correspondiente (véase Captura 28).



Captura 28. Interfaz que muestra cómo abrir una nueva conversación con un amigo

Una vez se encuentre abierta la conversación, se le permitirá al usuario escribir un mensaje en el espacio habilitado y pulsar intro para enviarlo (véase Captura 29).



Captura 29. Interfaz que muestra como enviar un mensaje

2. Anexo II: Guía de instalación

En este capítulo adicional, se redacta un manual que describe paso a paso como realizar la instalación y despliegue de la aplicación.

No obstante, el código completo se encuentra accesible a través del siguiente repositorio:

https://github.com/ferlopezcarr/MeetAndShareTFG-Entrega

2.1. Crear una nueva carpeta

Para asegurar el correcto funcionamiento del proyecto, es necesario crear una nueva carpeta con un nombre descriptivo tal como "MeetAndShare-SpringApplication".

2.2. Descomprimir el archivo comprimido

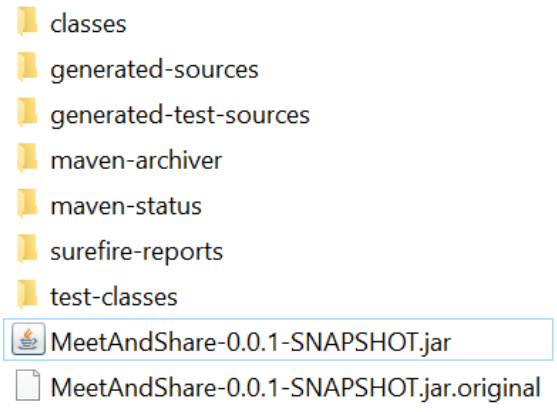
Tras crear la carpeta, debe volcarse el contenido del comprimido entregado en esta nueva ubicación.

2.3. Comprobar puerto 8080

Antes de ejecutar la aplicación, debemos comprobar que ninguna aplicación o servicio web se encuentra escuchando en el puerto indicado, ya que, de ser así, colisionaría con esta y no se conseguiría desplegar.

2.4. Ejecutar el proyecto

Una vez se encuentra instalada la aplicación y se ha comprobado el puerto, se debe hacer doble click en el fichero "MeetAndShare-0.0.1-SNAPSHOT.jar" como se muestra en la siguiente Captura:



Captura 30. Estructura del proyecto ejecutable entregado

2.5. Despliegue de la aplicación

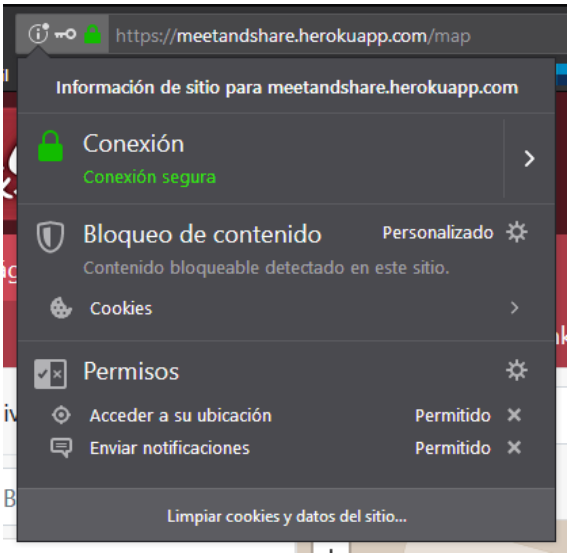
Tras esto, la aplicación queda accesible a través del navegador (Mozilla y Chrome) bajo las siguientes urls:

- Sitio recomendado: https://meetandshare.herokuapp.com/
- Aplicación local ejecutada: localhost:8080

2.6. Advertencias

Resulta necesario que el usuario conceda los permisos a la aplicación para acceder a las notificaciones y la ubicación, ya que, de otro modo, no se garantiza el correcto funcionamiento de los casos de uso involucrados.

En caso de que estos permisos no hayan sido solicitados, pueden configurarse haciendo click en el símbolo de información ubicado a la izquierda de la url del sitio mostrado en la siguiente Captura:



Captura 31. Gestionar permisos de la aplicación