UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería Informática

Desarrollo de una aplicación web para la promoción del deporte

Trabajo Fin de Grado

Grado en Ingeniería Informática

AUTOR/A: Herrando Cuenca, Sergio

Tutor/a: Vidal Oriola, Germán Francisco

CURSO ACADÉMICO: 2023/2024

Resumen

El deporte es uno de los principales pasatiempos de la sociedad actual. Además de su práctica como actividad recreativa, también ofrece múltiples beneficios para la salud. Sin embargo, uno de los principales problemas a la hora de practicar un deporte es encontrar compañeros con los que poder realizar la actividad en cuestión. Gracias a las nuevas tecnologías, podemos hacer esta tarea más accesible, acortando las distancias entre las personas (y entre los centros deportivos), fomentando así la práctica del deporte en el momento y lugar deseado. Más concretamente, la aplicación que se propone pretende dotar a los usuarios de una serie de funciones para facilitar la práctica de su deporte favorito, como son la búsqueda de partidos ya organizados o la creación de nuevos partidos que el resto de los usuarios podrá ver y, si son de su interés, apuntarse. De este modo, permitirá crear vínculos entre deportistas, aumentando la frecuencia del ejercicio deportivo y, en consecuencia, su salud y bienestar personal. Además, los centros deportivos podrán acercarse aún más a los usuarios ofertando sus servicios dentro de la aplicación con el objetivo de aumentar el uso de sus instalaciones.

En esta memoria se recogen todas las fases que han constituido el desarrollo de esta aplicación. Desde un primer punto de contacto investigando aplicaciones relacionadas hasta el análisis de requisitos, el diseño de la solución, su desarrollo y pruebas.

Palabras clave: deporte, búsqueda de compañeros, desarrollo web

Abstract

Sports are one of the main pastimes of today's society. In addition to being practiced as a recreational activity, it also offers multiple health benefits. However, one of the main problems when practicing a sport is finding companions with whom to carry out the activity in question. Thanks to new technologies, we can make this task more accessible, shortening the distances between people (and between sports centres), thus promoting the practice of sports at the desired time and place. More specifically, the proposed application aims to provide users with a series of functions to facilitate the practice of their favourite sport, such as searching for already organized matches or creating new matches that other users can see and join if they are interested. In this way, it will allow the creation of bonds between athletes, increasing the frequency of physical exercise and, consequently, their health and personal well-being. Additionally, sports centres will be able to get even closer to users by offering their services within the application with the aim of increasing the use of their facilities.

This report covers all the phases that have constituted the development of this application. From an initial point of contact by researching related applications to the requirements analysis, solution design, development, and testing.

Keywords: sports, finding companions, web development

Índice de contenidos

1. Introducción 8
1. Motivación 8
2. Objetivo 8
3. Estructura 8
2. Estado del arte 10
2.1 SportPartner 10
2.2. Meetup 11
2.2 Telegram 13
2.3 Comparación de las aplicaciones 13
2.4 Crítica al estado del arte 14
3. Especificación de Requisitos Software 15
3.1 Introducción 15
3.1.1 Propósito 15
3.1.2 Alcance 15
3.1.3 Definiciones, siglas y abreviaciones 16
3.1.4 Apreciación global 16
3.2 Descripción global 16
3.2.1 Perspectiva del producto 16
3.2.2 Funciones del producto 16
3.2.2.1 Casos de uso 17
3.2.2.1.1 Módulo usuarios 17
3.2.2.1.1.1 Módulo autenticación 20
3.2.2.1.2 Módulo actividades 21
3.2.2.1.3 Módulo centros 24
3.2.3 Características del usuario 25
3.2.4 Restricciones 25
3.2.4.1 Restricciones funcionales 25
3.3 Requisitos específicos 25
3.3.1 Atributos del sistema software 25
4. Diseño de la solución 26
4.1 Tecnologías utilizadas 26
4.1.1 Tecnologías del cliente 26
4.1.1.1 Vue.js 26
4.1.1.2 Vuetify 26
4.1.1.3 VeeValidate 27

4.1.1.4 Vue-i18n 27
4.1.2 Tecnologías del servidor 28
4.1.2.1 Spring Boot 28
4.1.2.2 Hibernate y JPA 28
4.1.2.3 Flyway 28
4.1.2.4 Argon2 29
4.2 Arquitectura de la aplicación 29
4.2.1 Arquitectura del cliente 29
4.2.1.1 Arquitectura basada en componentes 30
4.2.1.2 Single Page Application 30
4.2.2 Arquitectura del servidor 31
4.2.2.1 Arquitectura en capas 31
4.2.2.2 Base de datos 31
5. Desarrollo de la solución 34
5.1 Desarrollo del cliente 34
5.1.1 Estructura 34
5.1.2 Reutilización de componentes 34
5.1.2.1 Componentes personalizables 35
5.1.2.2 Herencia 35
5.1.2.3 Mixins 35
5.1.3 Internacionalización 36
5.1.4 VeeValidate 36
5.2 Desarrollo del servidor 37
5.2.1 Arquitectura en capas 37
5.2.1.1 Desarrollo de la capa de presentación 37
5.2.1.2 Desarrollo de la capa de servicio 38
5.2.1.3 Desarrollo de la capa de persistencia 38
5.2.1.4 Desarrollo de la capa de modelo 39
5.2.2 Data Transfer Objects 40
5.2.3 Almacenamiento de contraseñas 40
5.2.4 Gestión de errores 41
5.3 Metodología empleada en el desarrollo 42
5.3.1 JIRA 42
5.3.2 Organización en *sprints* 43
5.3.3 Git 44
Convenciones 44
5.4 Resultado final 45

6. Pruebas 50
6.1 Pruebas unitarias 50
6.2 Pruebas de integración 51
6.2.1 Problemas encontrados: CORS 51
6.3 Análisis estático de código 52
6.4 Pruebas de aceptación 52
7. Conclusiones 54
7.1 Relación con los estudios cursados 55
7.1.1 Competencias transversales 55
7.2 Trabajo futuro 55
Bibliografía 57
8. Pruebas de aceptación 59
8.1 Sesión 1 59
8.2 Sesión 2 60
9. Objetivos de Desarrollo Sostenible 61

Índice de ilustraciones

Ilustración 1: Capturas de pantalla SportPartner – Actividades....10
Ilustración 2: Capturas de pantalla SportPartner – Miembros 11
Ilustración 3: Capturas de pantalla Meetup – Registro 11
Ilustración 4: Capturas de pantalla Meetup – Actividades....12
Ilustración 5: Captura de pantalla Meetup – Conexiones....12
Ilustración 6: Capturas de pantalla Telegram - Grupos y chats....13
Ilustración 7: Diagrama de dominio con los conceptos principales del proyecto.... 17
Ilustración 8: Diagrama de casos de uso de usuarios 17
Ilustración 9: Diagrama de casos de uso de autenticación 20
Ilustración 10: Diagrama de casos de uso de actividades 21
Ilustración 11: Diagrama de casos de uso de centros 24
Ilustración 12: Ciclo de vida de un componente Vue. Fuente: https: vuejs.org 27
Ilustración 13: Arquitectura de la aplicación 29
Ilustración 14: Arquitectura del cliente.... 30
Ilustración 15: Arquitectura del servidor 31
Ilustración 16: Diagrama entidad relación de la base de datos.... 33
Ilustración 17: Estructura de la aplicación Vue.... 34
Ilustración 18: Ejemplo de mixin 35
Ilustración 19: Archivo JSON con traducciones.... 36
Ilustración 20: Componente ValidationSpan.... 36
Ilustración 21: Estructura de controladores....37
Ilustración 22: Estructura de servicios.... 38
Ilustración 23: Estructura de repositorios 38
Ilustración 24: Estructura de modelos.... 39
Ilustración 25: Entidad Player.... 40
Ilustración 26: Código de registro e inicio de sesión....41
Ilustración 27: Clase con los códigos de error....41
Ilustración 28: Historia con subtareas en JIRA.... 42
Ilustración 29: Cronograma con sprints 44
Ilustración 30: Estructura de ramas Git.... 44
Ilustración 31: Estructura de un commit en Git.... 45
Ilustración 32: Proceso de registro en SportMate.... 46
Ilustración 33: Creación de actividad en SportMate.... 46
Ilustración 34: Calendario de actividades en SportMate 47
Ilustración 35: Buscador de centros en SportMate.... 48
Ilustración 36: Página de centros en SportMate.... 48
Ilustración 37: Página de centro con usuarios en Sport Mate.... 49
Ilustración 38: Prueba unitaria del método registerPlayer.... 50
Ilustración 39: Configuración del CORS en Spring Boot 51
Ilustración 40: Advertencia de error de SonarLint 52
Ilustración 41: Controlador refactorizado con DTO.... 52

Índice de tablas

Tabla 1: Comparación de aplicaciones 13
Tabla 2: Caso de uso - Buscar jugador 18
Tabla 3: Caso de uso - Ver jugador 18
Tabla 4: Caso de uso - Editar perfil de usuario 18
Tabla 5: Caso de uso - Enviar solicitud de amistad 19
Tabla 6: Caso de uso - Ver solicitudes de amistad 19
Tabla 7: Caso de uso - Aceptar solicitud de amistad 19
Tabla 8: Caso de uso - Rechazar solicitud de amistad 19
Tabla 9: Caso de uso - Registro 20
Tabla 10: Caso de uso - Inicio de sesión 21
Tabla 11: Caso de uso - Cerrar sesión 21
Tabla 12: Caso de uso - Crear partido 22
Tabla 13: Caso de uso - Editar partido 22
Tabla 14: Caso de uso - Unirse a partido 22
Tabla 15: Caso de uso - Abandonar partido 23
Tabla 16: Caso de uso - Buscar partido 23
Tabla 17: Caso de uso - Ver partido 23
Tabla 18: Caso de uso - Buscar centro 24
Tabla 19: Caso de uso - Ver centro 24
Tabla 20: Caso de uso - Seguir centro 25

CAPÍTULO 1
Introducción

1. Motivación

Con el paso de los años, la sociedad evoluciona y cambian hábitos de la vida cuotidiana. Es innegable que el deporte es una de esas facetas que las personas han decidido incluir en su vida diaria como una forma de alcanzar el bienestar y mejorar su salud. Sin embargo, pese a que los deportes son ya un pilar en la vida de mucha gente, muchos de ellos requieren de más personas para poder practicarlos. En algunas ocasiones, no encontrar a un compañero supone no poder ejercitarse ese día, perder el hábito y, en ocasiones, abandonar la práctica habitual.

Por otro lado, la comunicación entre deportistas a la hora de organizar un evento suele realizarse a través de aplicaciones de mensajería instantánea tales como *WhatsApp* y *Telegram*, suponiendo la mayoría de las veces pérdidas de tiempo o malentendidos. Además, las personas no siempre son conocedoras de la oferta deportiva que existe en los lugares donde habitan. Con el *boom* del deporte, no son pocos los nuevos establecimientos para la práctica deportiva tales como gimnasios, rocódromos, … Incluso las propias instalaciones municipales que suelen ser desconocidas para los nuevos habitantes.

2. Objetivo

La aplicación a desarrollar propone acortar distancias entre deportistas, facilitar la práctica del deporte y fomentarlo en los distintos establecimientos deportivos. Más concretamente, la aplicación deberá ser capaz de ofrecer a los usuarios la posibilidad de crear partidas de cualquier deporte, especificando hora y lugar, de forma que otros usuarios puedan unirse a ella y practicar deporte juntos. También los propios negocios deportivos podrán darse a conocer dentro de la aplicación con secciones dedicadas a sus centros, pudiendo ofertar sus instalaciones, así como ofrecer descuentos exclusivos. Además, el proyecto se apoya sobre el tercero de los Objetivos de Desarrollo Sostenible (ODS) de la OMS, el de Bienestar y Salud, fomentando la práctica del deporte entre las personas para que puedan beneficiarse de sus múltiples beneficios.

3. Estructura

El documento está dividido en varios apartados que constituyen la memoria completa del TFG. Cada uno aborda un aspecto fundamental del mismo y sigue una estructura común al resto. Además del presente capítulo de introducción, el resto de los capítulos que componen la memoria son los siguientes:

2. **Estado del arte**
   En este apartado se recoge un estudio de aplicaciones en el mercado similares a la propuesta en este proyecto, analizando sus puntos fuertes y débiles y comenzando a definir cuáles deben ser las características deseadas de nuestra aplicación.
3. **Especificación de Requisitos Software**
   En este capítulo se detallarán los requisitos que deberá cumplir la aplicación una vez su desarrollo se dé por concluido acompañándolo de diagramas de dominio y diagramas de casos de uso.

4. **Diseño de la solución**
   Este capítulo describe con detalle temas relacionados con la arquitectura y tecnologías de la aplicación. Argumenta porqué se ha optado por cada una de ellas y se introducen algunas ilustraciones que complementen la explicación.
5. **Desarrollo de la solución**
   A lo largo de este capítulo se hace un repaso de la fase de implementación abordando cada uno de los elementos que la componen. Se detallan algunos de los conceptos más importantes del sistema y cómo se han desarrollado. Por último, se explica la metodología seguida para la consecución del proyecto.
6. **Pruebas**
   En este capítulo se hace mención a los tipos de pruebas realizadas durante el proyecto junto a algunos ejemplos reales y la solución que se le ha dado a los problemas encontrados.
7. **Conclusiones**
   El último capítulo hace un repaso de las distintas fases del proyecto y evalúa la consecución de los objetivos marcados al comienzo de este. Se relaciona con los estudios cursados y se presentan los trabajos a realizar en el futuro.

Además de los capítulos, al final de la memoria se encuentran dos anexos:

- Anexo 1: se recogen los formularios de las pruebas de aceptación.
- Anexo 2: relaciona este trabajo con los Objetivos de Desarrollo Sostenible (ODS).

CAPÍTULO 2
Estado del arte

A medida que avanzan los años la oferta de aplicaciones se vuelve más grande y el mercado más competitivo. Dentro de un mismo tema podemos encontrar gran variedad de posibilidades, desde la más general a una más específica.

En el caso de las aplicaciones deportivas, el mercado está copado por un reducido grupo. Sin embargo, este grupo ofrece generalmente aplicaciones enfocadas a la medición de datos deportivos y su posterior publicación en redes sociales propias de la aplicación o de terceros como *Twitter* o *Instagram*. Ese enfoque es muy distinto al que se da en este proyecto. La principal diferencia es que esta aplicación pretende facilitar la práctica de deporte con otras personas.

Acotando pues las aplicaciones del mercado que buscan ofrecer a los usuarios mayor facilidad a la hora de practicar deporte junto a más personas, vamos a analizar algunas de ellas:

2.1 SportPartner

SportPartner[1] es una aplicación para smartphone disponible en *iOS* y *Android* que permite publicar actividades para que otros usuarios se unan. Sigue un concepto muy similar al de una red social, permitiendo a los usuarios crear perfiles con imágenes, biografías y deportes favoritos.

Al registrarse por primera vez en la aplicación se solicita una ubicación y un listado de los deportes favoritos del usuario. Estos datos se utilizarán para ofrecer actividades de interés cerca de la zona del usuario (ver Ilustración 1). Además, *SportPartner* cuenta con un módulo llamado *Miembros*. En este, aparecen todos los usuarios relacionados con el perfil del usuario teniendo en cuenta sus preferencias e intereses (ver Ilustración 2). De esta manera es posible entablar amistades y conocer gente que practique el mismo deporte de una forma fácil y sencilla.

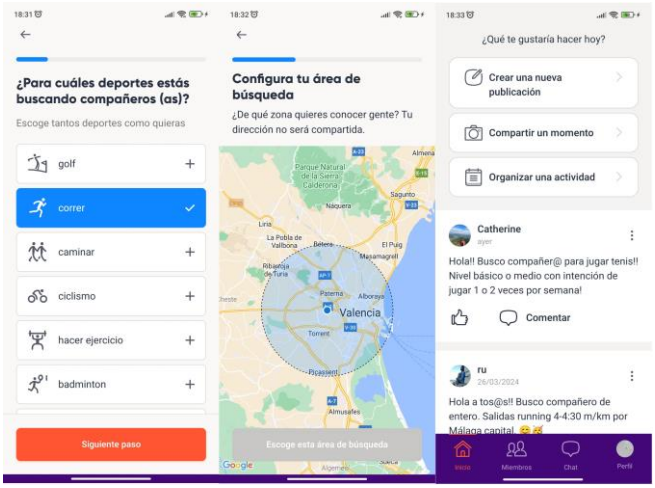

Ilustración 1: Capturas de pantalla SportPartner – Actividades

[1] https://www.sportpartner.com/es

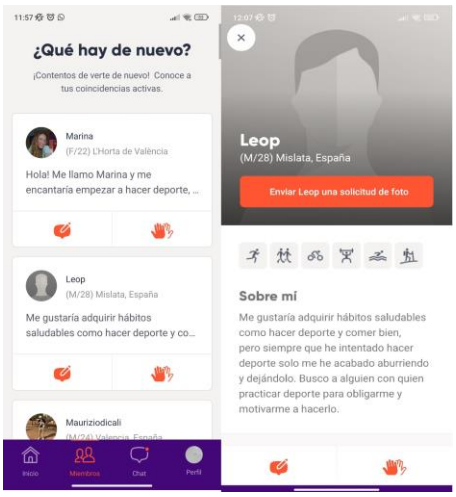

Ilustración 2: Capturas de pantalla SportPartner – Miembros

2.2. Meetup

Meetup[2] es una aplicación web y móvil con la que encontrar actividades en una zona. La variedad de estas actividades es muy grande y no solo comprende deportes sino eventos culturales, gastronómicos, formativos, etc.

Durante la fase de registro se debe rellenar algunos campos con información relativa al propio usuario, además de su localización e intereses (ver Ilustración 3).

Esta información será muy útil a la hora de mostrar una de las funcionalidades más interesantes de la aplicación: *Explorar*. Dentro de este módulo, los usuarios son capaces de descubrir qué eventos existen cerca de ellos. El uso de un mapa facilita mucho buscar actividades de interés. Una vez se selecciona una, se muestra una interfaz muy sencilla e intuitiva que facilita a los usuarios los datos necesarios para la actividad (ver Ilustración 4).

Ilustración 3: Capturas de pantalla Meetup – Registro

[2] https://www.meetup.com/es-ES/

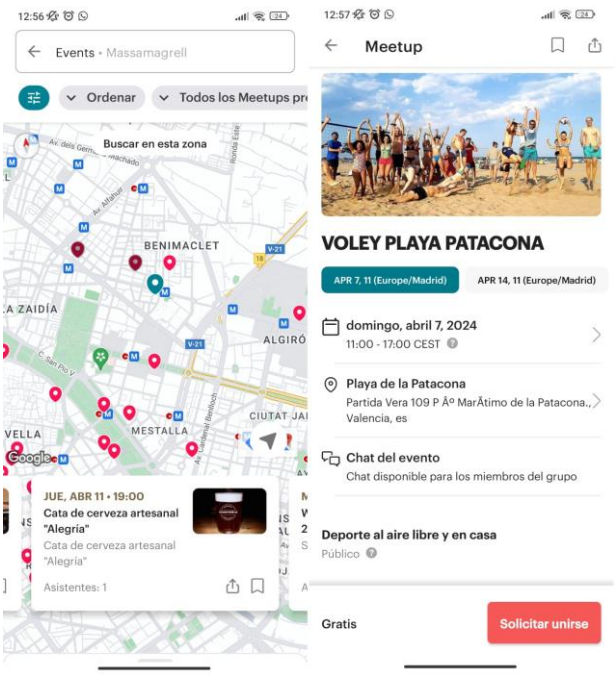

Ilustración 4: Capturas de pantalla Meetup – Actividades

Por último, *Meetup* ofrece a los usuarios agregarse como amistades, pero bajo la condición de haber asistido a un mismo evento previamente. Durante las siguientes veinticuatro horas a la finalización de una actividad, ambos usuarios pueden decidir si añadirse a la lista de amigos dentro de la aplicación (ver Ilustración 5).

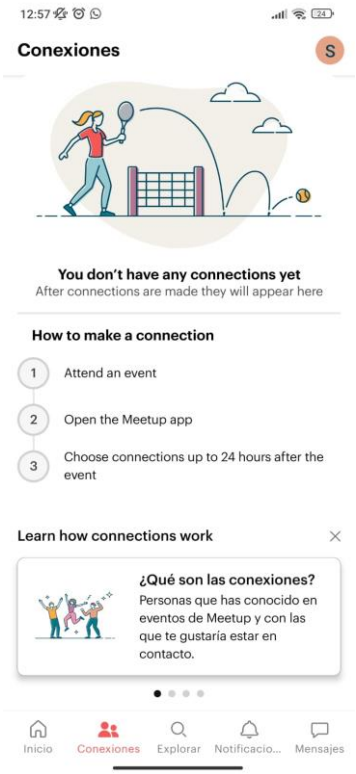

Ilustración 5: Captura de pantalla Meetup – Conexiones

A pesar del gran abanico de posibilidades que ofrece la aplicación, *Meetup* restringe algunas de ellas a los usuarios con suscripción de pago. Estas suscripciones pueden ser mensuales o anuales.

2.2 Telegram

Las aplicaciones de mensajería, como es el caso de *Telegram*[3], son una solución muy utilizada por los usuarios a la hora de encontrar compañeros de deporte. Entre sus principales características destaca la comodidad y la gran cantidad de usuarios que la utiliza.

La aplicación ofrece los denominados *Grupos*. Son chats comunitarios en los que el acceso puede ser público o privado y donde los usuarios ingresan para establecer relaciones entre ellos. De esta forma, se crean grupos deportivos donde las personas organizan actividades (ver Ilustración 6).

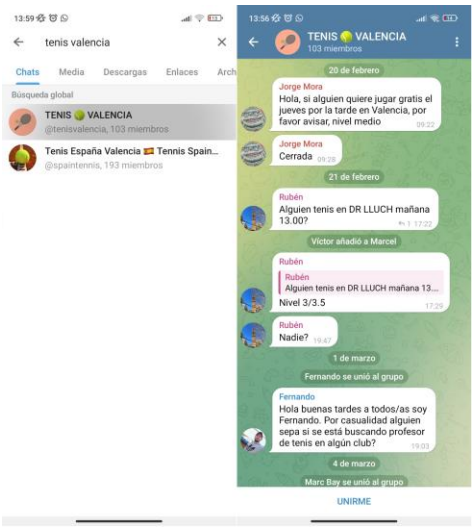

Ilustración 6: Capturas de pantalla Telegram - Grupos y chats

2.3 Comparación de las aplicaciones

A continuación, a modo de resumen, la Tabla 1 muestra las principales funcionalidades de cada aplicación.

Tabla 1: Comparación de aplicaciones

	Aplicaciones		
Funcionalidad	**SportPartner**	**Meetup**	**Telegram**
Registro obligatorio	✓	✓	✓
Creación y búsqueda de actividad	✓	✓	Vía mensaje de texto
Perfil de usuario personalizable	✓	✓	Reducido

[3] https://telegram.org/

Actividades ofrecidas según gustos e intereses	✓	✓	✗
Compatibilidad	Android, iOS	Web, Android, iOS	Web, Android, iOS
Gratuito	✓	Parcial	✓

2.4 Crítica al estado del arte

Telegram puede ser una opción para los usuarios que quieran cierta inmediatez, evitando descargar aplicaciones o creando perfiles dentro de las mismas. A pesar de ello, unirse a grupos con gente desconocida es desaconsejable por motivos de privacidad, *spam*, etc.

Las aplicaciones analizadas en los apartados 2.1 y 2.2 muestran características comunes entre ellas. En primer lugar, implementan un tipo de red social. Los usuarios tienen sus propios perfiles personalizables que permiten a otros usuarios saber con quién interactúan. Pueden agregar a otras personas a su lista de confianza incluso dejar comentarios y realizar invitaciones privadas a sus actividades.

En segundo lugar, los módulos de creación y búsqueda de actividades son muy ricos y suelen cumplir con todas las necesidades que el usuario necesita. Además, poseen interfaces intuitivas y sencillas que facilitan la tarea a los usuarios.

Como aspecto negativo, si solamente dejamos en manos del usuario la creación de actividades y, en consecuencia, del flujo principal de la aplicación, podemos estar perdiendo la oportunidad de fomentar al máximo la utilización. Por esto mismo, en este TFG se pretende introducir un nuevo factor, totalmente externo a los propios usuarios, que cree y promocione actividades, aumentando así los eventos, las conexiones entre usuarios y el uso de la aplicación. Esta misión la llevarán a cabo los centros deportivos, que podrán ofertar sus centros para realizar actividades. Además, *Meetup* no es totalmente gratuita. Muchas de sus funciones requieren de pagos que tiene que asumir el usuario. Mediante la propuesta que ofrece este proyecto, el coste y mantenimiento de la aplicación sería a cargo de los centros deportivos que deseen publicitarse. Así pues, los centros deberían abonar un importe para que el resto de los usuarios conozcan sus instalaciones y su oferta deportiva.

En definitiva, la aplicación seguirá el estándar del mismo tipo de aplicaciones en el mercado en concepto de creación y búsqueda de actividades, así como el formato de red social, pero reformulará la monetización brindando a las propias empresas externas la posibilidad de publicitarse a cambio del pago de tasas.

CAPÍTULO 3
Especificación de Requisitos Software

3.1 Introducción

A lo largo de esta sección se detallarán todos los requisitos que debe cumplir la aplicación una vez haya sido implementada. Dadas las características del proyecto, se seguirá el estándar de Especificación de Requisitos de Software (*ERS*) IEEE-STD-830-1998 [1].

El desarrollo de la aplicación se divide en tres módulos:

1. Módulo usuarios: comprende toda la funcionalidad relacionada con el registro, creación de perfiles, inicio de sesión y amistades.
2. Módulo actividades: permitirá crear, buscar o eliminar actividades de interés.
3. Módulo centros: destacará a los centros deportivos dentro de las actividades de la aplicación.

3.1.1 Propósito

El propósito de la Especificación de Requisitos de Software no es otro que detallar cuales son los requerimientos que la aplicación debe cumplir para facilitar el futuro desarrollo, así como acotar los distintos comportamientos de la aplicación. Está dirigido a los desarrolladores del sistema.

3.1.2 Alcance

La aplicación recibirá el nombre de *SportMate*, haciendo referencia en inglés a su principal objetivo: encontrar compañeros de deporte.

El sistema será capaz de ofrecer diferentes actividades a los usuarios, ya sean creadas por otros usuarios u ofrecidas por centros deportivos. Dichas actividades deben ofrecer una serie de datos básicos tales como el tipo de deporte que se va a practicar, el nivel, los jugadores necesarios, la fecha, la hora y el lugar. También aparecerá la lista de jugadores que ya se han apuntado al evento.

Cada usuario tendrá un perfil público, accesible para el resto de los usuarios, con datos referentes a él tales como una pequeña descripción, una foto, los deportes que le gustan y el nivel que tiene en cada uno. Será posible agregar a otro usuario a la lista de amistades y será útil a la hora de crear eventos privados, pues solo los amigos podrán unirse. Además, las actividades finalizadas de los amigos de un usuario aparecerán en un tablón para incentivar el uso de la aplicación.

Finalmente, la aplicación ofrecerá un espacio donde los centros deportivos puedan actualizar su información. Dicha información será visible para todos los usuarios y supondrá una forma de publicitar centros. Será posible seguir a estos centros, siendo accesibles con mayor facilidad y apareciendo como primeras opciones al crear actividades. El pago de tasas de los centros no se incluirá como una funcionalidad del sistema.

3.1.3 Definiciones, siglas y abreviaciones

- **ERS**: especificación de requisitos de software.
- **Framework**: marco de trabajo que tiene como objetivo facilitar la solución de problemas al desarrollar. Es una estructura en la que apoyarse para construir software.
- **Back end**: compone la parte del servidor de la aplicación software. Guarda y procesa información para poder devolverla a la parte del cliente.
- **Front end**: compone la parte del cliente en la aplicación software. Es el punto de interacción entre el usuario y el sistema. Representa la información procedente del back end.
- **Protocolo HTTP**: del inglés *HyperText Transfer Protocol*, constituye el conjunto de normas y reglas para la comunicación entre cliente y servidor.

3.1.4 Apreciación global

Esta *ERS* está dividida en los siguientes apartados:

Primeramente, se hará una descripción global del proyecto, con el objetivo de explicar su relación con otros sistemas, sus funciones, las características de los usuarios que lo utilizarán, sus restricciones y un último apartado con el trabajo a realizar en un futuro. Para definir con mayor exactitud las funciones de la aplicación, se utilizan diagramas y tablas de casos de uso, así como un diagrama de dominio para lograr el entendimiento de los conceptos y relaciones de la aplicación.

Para terminar, se enuncian algunos atributos de calidad del sistema que la aplicación debe cumplir al término del desarrollo de esta.

3.2 Descripción global

3.2.1 Perspectiva del producto

El producto software no tiene relación con ningún otro y constituirá un único elemento independiente.

Para facilitar el entendimiento de la aplicación, de sus conceptos y de las relaciones entre ellos se muestra un diagrama de dominio (ver Ilustración 7). Un diagrama de dominio representa las entidades principales y sus relaciones dentro de un dominio específico. En nuestro caso, aparecen los pilares principales de nuestro proyecto, siendo el deporte, la actividad, el jugador y el centro los cuatro principales. Un jugador podrá practicar el número de deportes que desee además de unirse o crear cualquier cantidad de actividades. Por otra parte, los centros también podrán ofrecer actividades y ser seguidos por los jugadores. Además, estos centros ofrecen como mínimo una modalidad deportiva para practicar en sus instalaciones y tendrán la posibilidad de obsequiarles con promociones.

3.2.2 Funciones del producto

Estas son las distintas funcionalidades que debe ofrecer la aplicación:

- Módulo de usuarios: todo lo relacionado con los perfiles de usuario y listas de amistad.
    - Autenticación: dentro de este módulo se incluye el registro de usuario, inicio y cierre de sesión.

- Módulo de actividades: corresponde a la creación, búsqueda, unión y abandono de actividades y su visualización junto a la información necesaria.
- Módulo centros: funcionalidades relativas a la visualización de centros, su información y sus promociones.

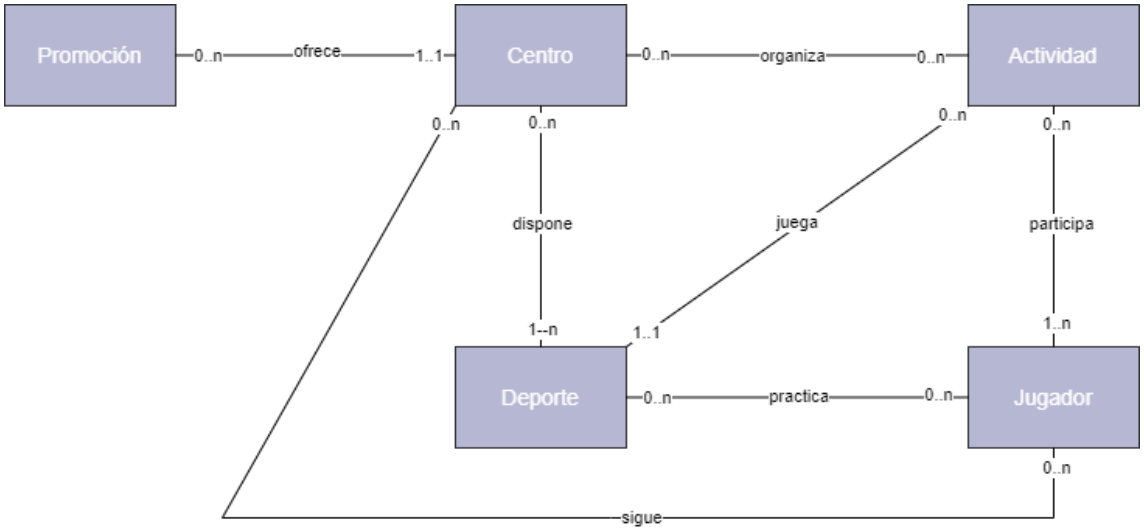

Ilustración 7: Diagrama de dominio con los conceptos principales del proyecto

3.2.2.1 Casos de uso

Vamos a detallar los distintos casos de uso haciendo una separación por módulos y siguiendo la enumeración de funciones del apartado 3.2.2. Cada descripción de módulo irá acompañada de un diagrama de casos de uso, así como de las correspondientes tablas.

3.2.2.1.1 Módulo usuarios

A continuación, se muestra el diagrama de casos de uso en el módulo de usuarios (ver Ilustración 8)

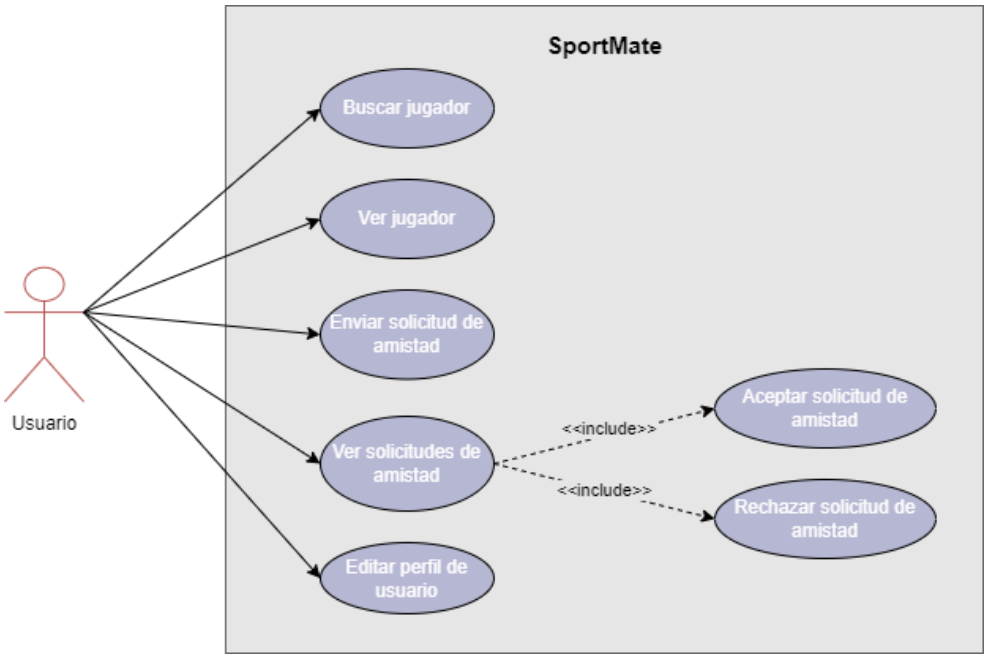

Ilustración 8: Diagrama de casos de uso de usuarios

Tabla 2: Caso de uso - Buscar jugador

Nombre	Buscar jugador
Descripción	Permite buscar a un jugador.
Precondición	No
Secuencia principal	1. El usuario introduce un texto de búsqueda en el buscador de usuarios. 2. La aplicación muestra una lista todos los usuarios relacionados con el parámetro de búsqueda.
Actor	Usuario
Postcondición	No
Notas	--

Tabla 3: Caso de uso - Ver jugador

Nombre	Ver jugador
Descripción	Permite al usuario ver la información de un jugador.
Precondición	No
Secuencia principal	1. El usuario pulsa sobre una tarjeta de jugador. 2. La aplicación muestra la ficha de jugador y la siguiente información: • Nombre • Apellidos • Nombre de usuario • Deportes practicados
Actor	Usuario
Postcondición	No
Notas	--

Tabla 4: Caso de uso - Editar perfil de usuario

Nombre	Editar perfil de usuario
Descripción	Permite editar el perfil personal del usuario.
Precondición	Inicio de sesión
Secuencia principal	1. El usuario selecciona el botón “Editar perfil” 2. La aplicación redirige a la página de edición de perfil. En esta, aparecen los siguientes datos del usuario: • Nombre • Apellidos • Nombre de usuario • Fecha de nacimiento • Correo electrónico • Foto de perfil • Ciudad • Deportes practicados y nivel 3. El usuario modifica los campos. 4. El usuario pulsa el botón “Guardar cambios”.
Actor	Usuario
Postcondición	La aplicación modifica la información del usuario y la almacena.
Notas	--

Tabla 5: Caso de uso - Enviar solicitud de amistad

Nombre	Enviar solicitud de amistad
Descripción	Permite añadir a la lista de amigos a un usuario.
Precondición	Inicio de sesión
Secuencia principal	1. El usuario pulsa el botón "Añadir amigo" en la ficha de jugador de otro usuario.
Actor	Usuario
Postcondición	La aplicación notifica al otro usuario de la solicitud de amistad.
Notas	--

Tabla 6: Caso de uso - Ver solicitudes de amistad

Nombre	Ver solicitudes de amistad
Descripción	Permite visualizar la lista de solicitudes de amistad pendientes de confirmación.
Precondición	Inicio de sesión
Secuencia principal	1. El usuario pulsa sobre el botón "Solicitudes de amistad". 2. La aplicación muestra la página con la lista de solicitudes de amistad pendientes. Cada solicitud consta de los siguientes campos: • Nombre del usuario solicitante • Foto de perfil del usuario solicitante • Botón de aceptar / rechazar
Actor	Usuario
Postcondición	No
Notas	--

Tabla 7: Caso de uso - Aceptar solicitud de amistad

Nombre	Aceptar solicitud de amistad
Descripción	Permite añadir a la lista de amigos de un usuario a otro usuario.
Precondición	Recibir solicitud de amistad por otro usuario
Secuencia principal	1. El usuario accede a la página de solicitudes de amistad. 2. El usuario pulsa el botón "Añadir" de la solicitud de amistad.
Actor	Usuario
Postcondición	La aplicación añade a ambos usuarios a sus respectivas listas de amigos.
Notas	--

Tabla 8: Caso de uso - Rechazar solicitud de amistad

Nombre	Rechazar solicitud de amistad
Descripción	Permite rechazar la petición de amistad.
Precondición	Recibir solicitud de amistad por otro usuario
Secuencia principal	1. El usuario entra en la página de solicitudes de amistad. 2. El usuario pulsa el botón "Rechazar" de la solicitud de amistad.
Actor	Usuario
Postcondición	La aplicación elimina la solicitud de la lista de solicitudes de amistad pendientes de confirmación del usuario.
Notas	--

3.2.2.1.1.1 Módulo autenticación

Los distintos casos de uso para el módulo de autenticación (ver Ilustración 9) se enumeran en este apartado.

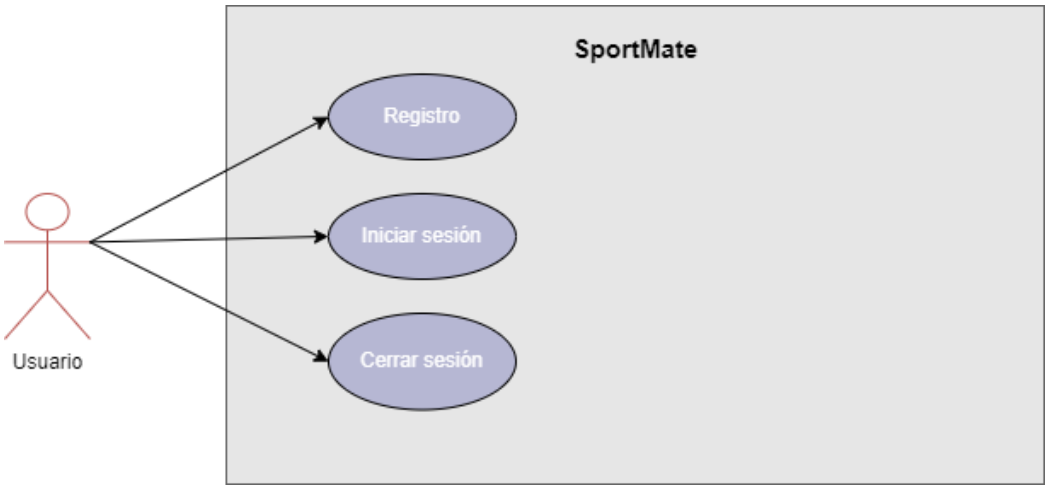

Ilustración 9: Diagrama de casos de uso de autenticación

Tabla 9: Caso de uso - Registro

Nombre	Registro
Descripción	Proceso de registro de nuevo usuario en la aplicación
Precondición	No
Secuencia principal	1. El usuario pulsa el botón “Regístrate” en la pantalla principal. 2. Se redirecciona a la página de registro donde se solicitan los siguientes datos: • Nombre • Apellidos • Nombre de usuario • Fecha de nacimiento • Correo electrónico • Contraseña • Confirmación de contraseña • Foto de perfil • Ciudad • Deportes practicados y nivel 3. El usuario introduce la información requerida. 4. El usuario pulsa el botón “Registrarme”.
Actor	Usuario
Postcondición	El nuevo usuario queda registrado en la aplicación.
Notas	--

Tabla 10: Caso de uso - Inicio de sesión

Nombre	Inicio de sesión
Descripción	Proceso de inicio de sesión de usuario. Permite el acceso a las principales funcionalidades de la aplicación.
Precondición	El usuario debe de estar previamente registrado.
Secuencia principal	1. El usuario selecciona el botón “Iniciar sesión”. 2. La aplicación desplegará un panel con un formulario de inicio de sesión consistente en: • Correo electrónico • Contraseña 3. El usuario introduce sus credenciales 4. El usuario pulsa el botón “Iniciar sesión”
Actor	Usuario
Postcondición	El usuario accede a la aplicación.
Notas	--

Tabla 11: Caso de uso - Cerrar sesión

Nombre	Cerrar sesión
Descripción	Permite al usuario cerrar sesión.
Precondición	Inicio de sesión
Secuencia principal	1. El usuario selecciona el botón “Cerrar sesión”. 2. La aplicación muestra la página principal.
Actor	Usuario
Postcondición	Se cierra la sesión del usuario.
Notas	--

3.2.2.1.2 Módulo actividades

A continuación, los casos de uso relacionados con actividades (ver Ilustración 10).

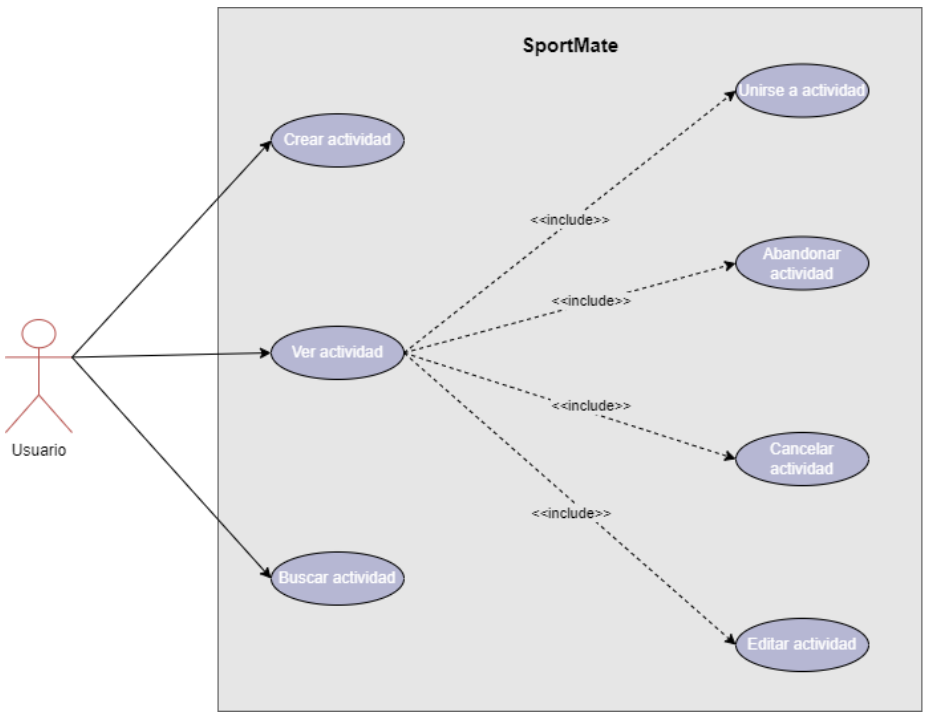

Ilustración 10: Diagrama de casos de uso de actividades

Tabla 12: Caso de uso - Crear partido

Nombre	Crear partido
Descripción	Proceso de creación de un partido. Permite al usuario crear un nuevo partido con distintos parámetros.
Precondición	Inicio de sesión
Secuencia principal	1. El usuario pulsa el botón "Crear Partida". 2. La aplicación muestra un formulario con los siguientes campos referentes a atributos de la partida: • Deporte • Descripción • Número de jugadores • Nivel • Lugar • Fecha (día y hora) • Privacidad 3. El usuario rellena los campos del formulario. 4. El usuario pulsa el botón "Crear partido".
Actor	Usuario
Postcondición	La aplicación almacena el partido.
Notas	--

Tabla 13: Caso de uso - Editar partido

Nombre	Editar partido
Descripción	Permite editar algunos campos de un partido.
Precondición	Usuario creador del partido.
Secuencia principal	1. El usuario pulsa sobre el botón "Editar partido" en la ficha de partido. • La aplicación muestra un formulario con los siguientes campos referentes a atributos del partido: • Descripción • Número de jugadores • Nivel • Lugar • Fecha (día y hora) • Privacidad 2. El usuario modifica los campos del formulario. 3. El usuario pulsa el botón "Guardar".
Actor	Usuario
Postcondición	La aplicación almacena el partido con los nuevos valores.
Notas	--

Tabla 14: Caso de uso - Unirse a partido

Nombre	Unirse a partido
Descripción	Permite al usuario unirse a un partido.
Precondición	Inicio de sesión. El partido no puede estar lleno.
Secuencia principal	1. El usuario pulsa sobre una tarjeta de partido. 2. La aplicación muestra la ficha de partido. 3. El usuario pulsa el botón "Unirme".
Actor	Usuario
Postcondición	La aplicación asocia al usuario con el partido.
Notas	--

Tabla 15: Caso de uso - Abandonar partido

Nombre	Abandonar partido
Descripción	Permite al usuario abandonar un partido.
Precondición	Pertenecer al partido
Secuencia principal	1. El usuario selecciona el partido que desea abandonar. 2. La aplicación muestra la modal de ficha de partida. 3. El usuario pulsa el botón “Abandonar partido”. 4. La aplicación muestra un modal con un mensaje de advertencia solicitando al usuario confirmación. 5. El usuario pulsa el botón “Abandonar” en la modal de advertencia.
Actor	Usuario
Postcondición	La aplicación elimina al usuario de la lista de jugadores del partido.
Notas	--

Tabla 16: Caso de uso - Buscar partido

Nombre	Buscar partido
Descripción	Permite al usuario buscar un partido según parámetros.
Precondición	No
Secuencia principal	1. El usuario pulsa el botón “Buscar partido” en la página de selección de partido. 2. La aplicación muestra un formulario con distintos campos a rellenar según las preferencias del usuario: • Lugar • Hora • Nivel • Deporte 3. El usuario introduce los parámetros de búsqueda. 4. El usuario pulsa el botón “Buscar”. 5. La aplicación muestra los partidos que coincidan con los parámetros introducidos.
Actor	Usuario
Postcondición	No
Notas	--

Tabla 17: Caso de uso - Ver partido

Nombre	Ver partido
Descripción	Permite visualizar la información correspondiente a un partido.
Precondición	No
Secuencia principal	1. El usuario pulsa sobre una tarjeta de partido. 2. La aplicación muestra la ficha de partido con la siguiente información: • Deporte • Jugadores unidos • Número de jugadores necesarios para completar partido • Lugar • Hora • Nivel • Descripción

	• Privacidad
Actor	Usuario
Postcondición	No
Notas	--

3.2.2.1.3 Módulo centros

Los casos de uso para el módulo de centros son los siguientes (ver Ilustración 11).

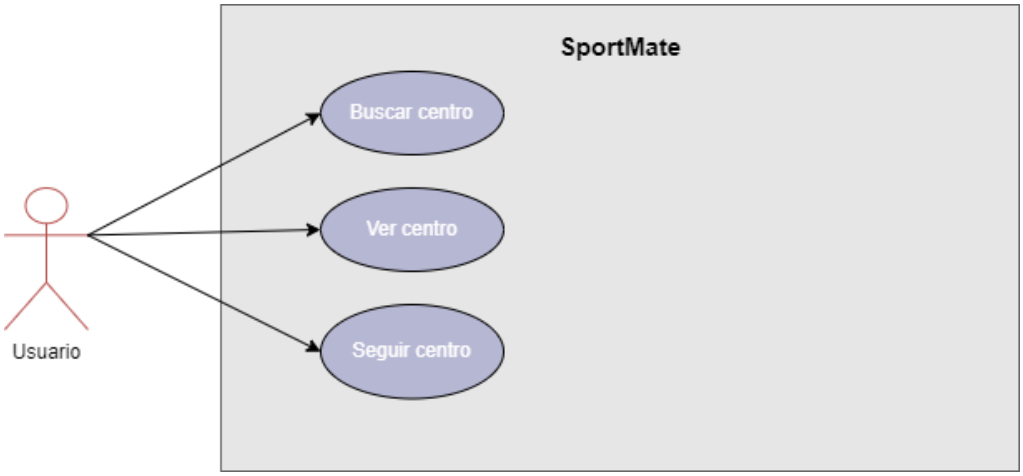

Ilustración 11: Diagrama de casos de uso de centros

Tabla 18: Caso de uso - Buscar centro

Nombre	Buscar centro
Descripción	Permite ver un centro y su información
Precondición	No
Secuencia principal	1. El usuario pulsa el botón "Buscar centros" 2. El usuario introduce un texto de búsqueda y/o un deporte. 3. La aplicación muestra una lista de todos los centros relacionados con los parámetros de búsqueda.
Actor	Usuario
Postcondición	No
Notas	--

Tabla 19: Caso de uso - Ver centro

Nombre	Ver centro
Descripción	Permite ver un centro y su información
Precondición	No
Secuencia principal	1. El usuario pulsa sobre una tarjeta de centro. 2. Se abre una página con información acerca del centro seleccionado donde aparecerá: • Nombre • Dirección • Imágenes • Promociones • Descripción • Lista de jugadores que lo siguen
Actor	Usuario
Postcondición	No
Notas	--

Tabla 20: Caso de uso - Seguir centro

Nombre	Seguir centro
Descripción	Permite al usuario seguir a un centro
Precondición	No
Secuencia principal	1. El usuario pulsa sobre el botón "Seguir" en la página del centro.
Actor	Usuario
Postcondición	La aplicación registra al centro de la lista de centros seguidos por el usuario.
Notas	--

3.2.3 Características del usuario

La diversidad de usuarios a los que está orientada la aplicación es muy grande. Cualquier persona con acceso a internet puede utilizar *SportMate*. Esto comprende rangos de edad muy grandes, diferente sexo, diferentes gustos y niveles deportivos variados. Los usuarios podrán navegar por la aplicación valiéndose por su experiencia en aplicaciones web y móviles. No será necesario que tengan ningún conocimiento técnico adicional.

3.2.4 Restricciones

Estas son las restricciones del proyecto que se deben tener en cuenta durante la fase de desarrollo:

3.2.4.1 Restricciones funcionales

- Disponer de un dispositivo con navegador web.
- La aplicación debe ser accesible desde cualquier sistema operativo.
- Es necesario el acceso a internet.
- La interfaz de usuario debe ser intuitiva, haciéndole fácil la experiencia al usuario.

3.3 Requisitos específicos

3.3.1 Atributos del sistema software

Estos son los requisitos de calidad que deberán satisfacerse:

- **Disponibilidad:** La aplicación deberá ofrecer todos sus servicios a los usuarios que deseen acceder y dispongan de conexión a internet, siendo esta necesaria, en cualquier caso.
- **Estética:** La aplicación debe presentar un aspecto sencillo, agradable y limpio, priorizando la simplicidad y el orden de los elementos que componen la aplicación.
- **Usabilidad:** La aplicación debe ser sencilla de usar para el usuario. Se debe perseguir la rapidez de aprendizaje, la memorización de patrones de funcionalidades y la simplicidad de los procesos.
- **Integridad:** La aplicación debe asegurar la integridad de los datos tanto en inserciones como en consultas no autorizadas.
- **Autenticidad:** La aplicación debe disponer de un sistema de autenticación para los usuarios.
- **Confidencialidad:** La aplicación debe garantizar la seguridad de sus datos e información ante consultas ilegítimas.

CAPÍTULO 4
Diseño de la solución

En el ciclo de vida de un proyecto software la fase de diseño juega un papel fundamental. Esta fase transforma los requerimientos previamente definidos en la especificación de requisitos en una arquitectura detallada que servirá de modelo y guía en la fase de implementación. En este capítulo definiremos las tecnologías y la arquitectura a utilizar en nuestra aplicación distinguiendo entre dos componentes: el cliente y el servidor.

4.1 Tecnologías utilizadas

4.1.1 Tecnologías del cliente

4.1.1.1 Vue.js

Vue.js[4] es un *framework* de JavaScript [2] de código abierto utilizado para construir interfaces de usuario y aplicaciones de página única (SPA). Vue.js se basa en un sistema de reactividad declarativa que permite que el DOM, o *Document Object Model*, [3] se actualice automáticamente cuando los datos cambian. Utiliza una estructura basada en componentes y un sistema de enlace de datos bidireccional que facilita la sincronización entre el modelo y la vista.

Los componentes constituyen la base del *framework*. Cada uno de ellos representa una instancia de Vue con unas opciones predefinidas. Permiten dividir la interfaz de usuario en piezas reutilizables. Atraviesan un ciclo de vida desde su creación hasta su destrucción (ver Ilustración 12). Durante estas fases, Vue.js proporciona *hooks* o ganchos que permiten ejecutar código en momentos específicos. Esto supone de gran ayuda a la hora de personalizar el comportamiento del componente.

Para conseguir que el DOM se actualice de forma automática, Vue.js utiliza un sistema de Virtual DOM para realizar actualizaciones sobre el DOM real. El Virtual DOM es una instancia en memoria del DOM real y permite detectar diferencias entre ambos con el fin de actualizar únicamente los elementos que lo necesiten.

Para definir gráficamente las interfaces, se hace uso de plantillas basadas en HTML que permiten combinarlo con directivas especiales, expresiones JavaScript y estilos CSS. Estas plantillas son declarativas y de fácil utilización.

4.1.1.2 Vuetify

Vuetify[5] es una biblioteca de componentes de interfaz de usuario para Vue.js basada en el diseño de Material Design de Google [4]. Facilita una amplia gama de componentes prediseñados. Incluye elementos comunes tales como botones, formularios, tarjetas, modales, tablas o menús; todos altamente personalizables.

Esta librería forma parte del ecosistema de Vue.js, lo que le hace beneficiarse de las herramientas proporcionadas por el *framework*. Goza de una integración fluida y se ajusta perfectamente a las convenciones y buenas prácticas de Vue.

[4] https://vuejs.org/
[5] https://vuetifyjs.com/en/

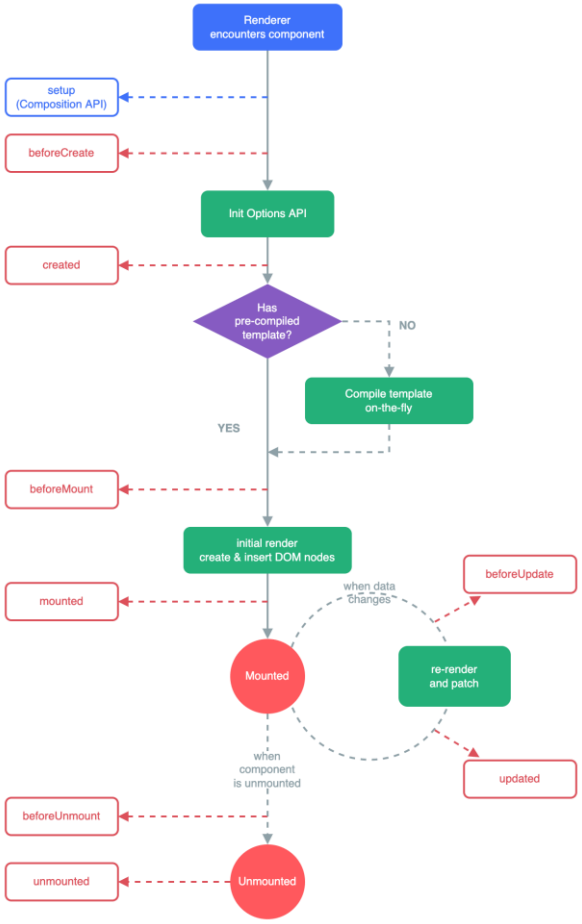

Ilustración 12: Ciclo de vida de un componente Vue. Fuente: https: vuejs.org

El uso de esta librería asegura una consistencia visual a lo largo de toda la aplicación, pues todos sus elementos siguen un estilo. Además, al ser componentes ya definidos pueden ahorrar mucho tiempo en el diseño e implementación de la interfaz de usuario.

4.1.1.3 VeeValidate

VeeValidate[6] es una biblioteca de validación de formularios para Vue.js diseñada para facilitar la validación de datos de entrada. Permite definir reglas de validación de forma declarativa para diferentes campos y validarlos automáticamente. Cuenta con una característica de validación asíncrona muy útil a la hora de realizar validaciones basadas en llamadas a API. Los mensajes de error son personalizables por cada regla y campo del formulario además de ofrecer soporte para su internacionalización.

4.1.1.4 Vue-i18n

La librería Vue-i18n[7] es una biblioteca que facilita la internacionalización, i18n [5], en aplicaciones Vue.js. Permite traducir el contenido adaptándolo a distintos idiomas y regiones. Cuenta con una integración muy sencilla y con funcionalidades interesantes como la selección dinámica de idiomas dentro de la aplicación.

[6] https://vee-validate.logaretm.com/v4/
[7] https://vue-i18n.intlify.dev/

4.1.2 Tecnologías del servidor

4.1.2.1 Spring Boot

Spring Boot[8] es un framework de desarrollo en Java que simplifica la creación de aplicaciones. Es una extensión de Spring Framework[9] que se centra en reducir el desarrollo al ofrecer autoconfiguración y un despliegue simplificado.

La Inversión del Control (IoC) y la Inyección de Dependencias (DI) son conceptos fundamentales en el ecosistema Spring Boot. IoC es un patrón de diseño en el cual la ejecución de un programa se invierte o delega a un contenedor o *framework*. Es este último el que se encarga de crear y administrar los componentes promoviendo la separación de responsabilidades y la creación de aplicaciones más flexibles y mantenibles. Por otro lado, DI es un patrón relacionado con IoC. Más bien se refiere a la técnica mediante la cual a los objetos se les proporciona sus dependencias. En lugar de que estos busquen o creen sus dependencias, estas son inyectadas desde el exterior.

Para implementar este par de conceptos Spring Boot hace uso del contenedor *ApplicationContext* que gestiona todos los objetos y sus dependencias. Un objeto, también llamado *Bean*, se indica en el propio código con anotaciones como *@Controller*, *@Service* o *@Repository* [6]. Además, la inyección de dependencias puede hacerse en el código de distintas maneras: mediante constructor, *setter* o la anotación *@Autowired*.

4.1.2.2 Hibernate y JPA

Para la gestión de los datos dentro de la aplicación y su persistencia en la base de datos se hará uso de las tecnologías Hibernate[10] y JPA.

Hibernate es un framework de mapeo objecto-relacional (ORM) para Java. Facilita el mapeo de objetos Java a tablas en una base de datos relacional y viceversa. Proporciona un lenguaje de consulta llamado HQL (*Hibernate Query Language*) que es orientado a objetos y facilita la manipulación de datos. Además, Hibernate maneja las transacciones de manera transparente para el desarrollador y simplifica la interacción con la base de datos.

JPA (*Java Persistance API*) es una especificación estándar de Java para la persistencia. Hibernate implementa JPA y utiliza sus interfaces y anotaciones para interactuar con la base de datos. Al igual que Spring Boot, JPA también ofrece anotaciones con las que especificar mapeos objeto-relacional y operaciones CRUD (*Create, Read, Update, Delete*). De forma similar a HQL en Hibernate, JPA define JPQL (*Java Persistance Query Language*) para realizar consultas orientadas a objetos.

4.1.2.3 Flyway

Flyway[11] es una herramienta diseñada para facilitar y gestionar la migración de esquemas de bases de datos de manera controlada y automatizada. Organiza las migraciones en versiones secuenciales donde cada migración representa un único script que describe los cambios necesarios. Estos *scripts* están numerados y Flyway los

[8] https://spring.io/projects/spring-boot
[9] https://spring.io/
[10] https://hibernate.org/
[11] https://flywaydb.org/

ejecuta de manera ascendente asegurando que los cambios se apliquen de forma controlada y predecible. Además, Flyway proporciona un mecanismo de *rollback* o vuelta atrás en el caso de fallo durante la migración.

4.1.2.4 Argon2

Argon2 es un algoritmo hash diseñado para el almacenamiento de contraseñas de forma segura. Fue ganador del concurso PHC[12] (*Password Hashing Competition*) en 2015, organizado por expertos en seguridad informática.

El algoritmo está diseñado para resistir ataques a fuerza bruta, donde el atacante trata de adivinar contraseñas probando con un gran número de combinaciones. Cuenta con parámetros de configuración como coste de tiempo, tamaño en memoria o paralelismo que permiten ajustar el algoritmo a las necesidades de seguridad. También protege ante tipos de ataque que buscan patrones débiles en las contraseñas como ataques de diccionario [7] o de tablas arcoíris.

4.2 Arquitectura de la aplicación

La arquitectura seguida para la implementación del proyecto es la de cliente-servidor [8]. El cliente solicita recursos (solicitud), el servidor procesa la petición (procesamiento), devuelve los recursos (respuesta) y el cliente los presenta al usuario final mediante una interfaz gráfica, permitiéndole enviar nuevas solicitudes (interacción).

En nuestro sistema diferenciamos el cliente y el servidor en dos aplicaciones independientes. Por un lado, tenemos una aplicación Vue, encargada de gestionar la parte del cliente. Por otro lado, una API REST Spring Boot actúa como servidor y hace uso de una base de datos H2 (ver Ilustración 13). Así pues, esto constituye una arquitectura de aplicaciones desacopladas, donde cada componente se desarrolla, despliega, escala y mantiene por separado.

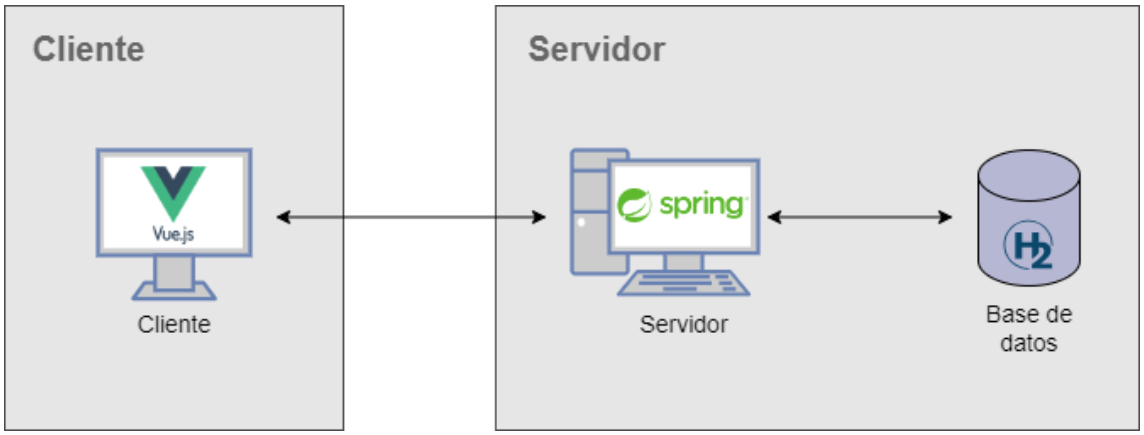

Ilustración 13: Arquitectura de la aplicación

4.2.1 Arquitectura del cliente

La parte del cliente está desarrollada con el framework Vue.js y algunas librerías externas como Vuetify o Axios. En conjunto, estos elementos permiten gestionar la vista y lógica de la interfaz con el fin de implementar una aplicación web SPA.

[12] https://www.password-hashing.net/

4.2.1.1 Arquitectura basada en componentes

La arquitectura basada en componentes se enfoca en la reutilización y la modularidad del código. El componente es la unidad fundamental y representa una funcionalidad relacionada. Algunas de sus ventajas son:

- Reutilización de código: cada componente puede ser reutilizado en cualquier parte de la aplicación. Al tratarse del desarrollo de la interfaz de usuario esto presenta una clarísima ventaja en los casos que se desee reutilizar alguna funcionalidad previamente implementada.
- Modularidad: permite dividir la aplicación en piezas más pequeñas. Así pues, un diseño complejo puede dividirse en componentes más pequeños, autónomos y desarrollados independientemente.
- Mantenimiento: al encapsular el código relacionado en componentes es más fácil depurar y corregir errores.
- Escalabilidad: es más sencillo introducir nuevas funcionalidades puesto que apenas hay dependencias entre los componentes ya existentes.

En nuestra aplicación haremos uso de los componentes de Vue.js. Cada uno encapsula su propia estructura HTML y CSS (*template*) y su lógica JavaScript (*script*). Estos componentes podrán ser reutilizables a lo largo de todo el aplicativo y tendrán acceso a la lógica común del cliente que son los servicios. Los servicios serán los encargados de comunicar los componentes Vue con el servidor haciendo uso de la librería Axios (ver Ilustración 14).

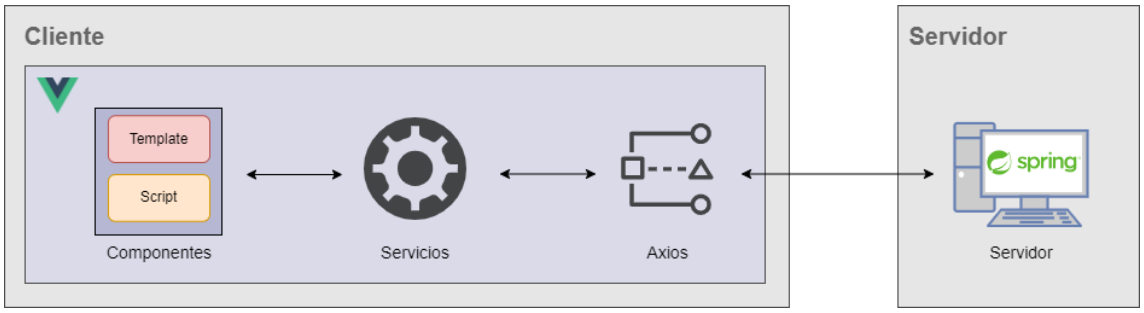

Ilustración 14: Arquitectura del cliente

4.2.1.2 Single Page Application

Single Page Application (Aplicación de Página Única) o SPA es una metodología de desarrollo web donde se carga una única página HTML y actualiza dinámicamente el contenido sin necesidad de recargar la página entera. Algunas ventajas de este modelo son:

- Experiencia de usuario mejorada: la navegación rápida y la capacidad de respuesta hacen que la experiencia de usuario sea fluida y sin interrupciones.
- Mejor rendimiento: al no tener que recargar la página completamente, las SPA pueden ofrecer un desempeño óptimo, haciéndose más notable en páginas con mucha interactividad y actualizaciones frecuentes.
- Reducción de carga en el servidor: al manejar más lógica en el cliente se reduce la cantidad de solicitudes al servidor y la cantidad de datos enviados mejorando la eficiencia del sistema.

4.2.2 Arquitectura del servidor

En el servidor se hace uso de una aplicación Spring Boot, encargada de procesar todas las peticiones provenientes del cliente.

4.2.2.1 Arquitectura en capas

La aplicación se construirá siguiendo una arquitectura en capas. Esta arquitectura define una forma de diseñar y organizar las funcionalidades del sistema separando cada una según su propósito. Podemos diferenciar entre:

- Capa de presentación: maneja la interacción con el usuario. Se encarga de recibir solicitudes HTTP y coordinar el flujo de datos entre la vista y los servicios.
- Capa de servicio: contiene la lógica de negocio de la aplicación.
- Capa de persistencia: se encarga de la interacción con la base de datos.
- Capa de modelo: contiene las clases que representan los datos de la aplicación y cómo se relacionan entre sí.

A nivel de Spring Boot, en la capa de presentación encontraremos los controladores, que manejarán todas las peticiones desde el cliente. En la capa de servicio se hallarán distintos servicios diferenciados por caso de uso. Por la parte de la capa de persistencia tendremos los repositorios, que interactuarán con las clases de dominio, representantes de la capa de modelo (ver Ilustración 15).

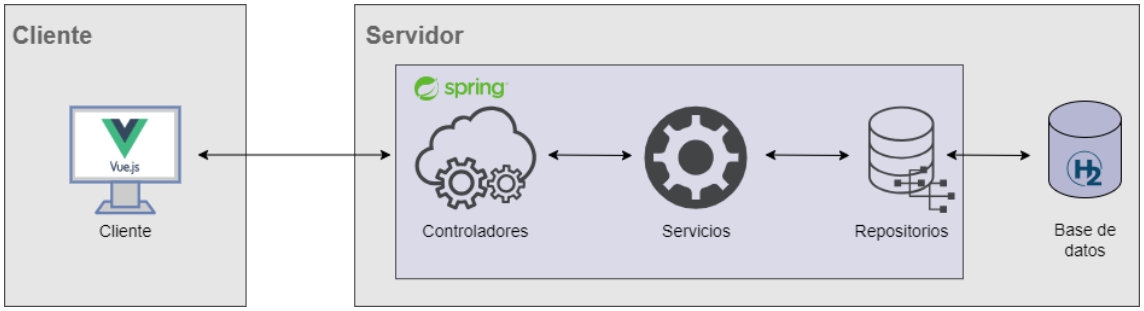

Ilustración 15: Arquitectura del servidor

4.2.2.2 Base de datos

La base de datos será H2. Se crearán distintas tablas y relaciones que expresen los datos y conceptos siguiendo el diagrama entidad-relación de la Ilustración 16. Este tipo de diagrama es comúnmente utilizado para representar las relaciones entre los distintos elementos de la base de datos. Se ha generado mediante la herramienta de gestión de bases de datos DBeaver.

En el diagrama entidad-relación se puede observar las entidades de nuestro proyecto. El elemento con mayor número de relaciones es la tabla *GAME*, la cual hace referencia a las actividades. Podemos ver que algunas columnas de esta tabla son claves foráneas a tablas como *GAME_STATUS*, *CITY* o *LEVEL*. Así conseguimos evitar redundancias, mantener la consistencia de los datos y lograr un mantenimiento más sencillo. Por otro lado, también es muy común en este diseño las tablas intermedias. Estas tablas las utilizaremos para representar en base de datos las relaciones Muchos a Muchos. Por ejemplo, los elementos *GAME* y *PLAYER* representan este tipo de relación, pues una actividad puede involucrar a muchos jugadores y a la vez un jugador puede estar

involucrado en muchas actividades. De esta forma aparecería la tabla *GAME_PLAYER* con las columnas *GAP_PLY_ID y GAP_GAM_ID* que hacen referencia a la columna jugador y actividad respectivamente. Otros ejemplos pueden ser la tabla *CENTER_PLAYER*, *CENTER_SPORT* o *PLAYER_SPORT*.

Cabe destacar el diseño de la tabla *PLAYER_FRIEND*. Esta tabla será la encargada de almacenar todas las solicitudes de amistad y su estado entre jugadores. Las columnas *PLF_PLY_SENDER* y *PLF_PLY_RECEIVER* contendrán claves foráneas a la columna *PLY_ID* de la tabla *PLAYER* y representarán al emisor y receptor de la solicitud. Estas dos columnas también formarán la clave primaria de la tabla, evitando más de una solicitud de amistad entre jugadores. Además, con el fin de asegurar la integridad de los datos, se creará un *check* que impide persistir un emisor y un receptor idénticos, pues por lógica de negocio un jugador no puede enviarse una solicitud de amistad a sí mismo.

Respecto a la autenticación, se hace uso de la tabla *AUTHENTICATION*. En ella se almacena una clave *hash* junto al identificador del jugador. De esta forma se aumenta la seguridad separando los datos en diferentes tablas y permitiendo así ser más flexibles en un futuro si se decide cambiar el método de autenticación.

La tabla *CENTER_IMAGE* guarda rutas de imágenes en bases de datos. Estas imágenes van asociadas a centros y cada una tiene una tipología según el lugar que vayan a ocupar. Por ejemplo, con la columna *CNI_TYPE* podemos indicar que esa es la imagen principal de un centro, o que esa imagen debe cargarse en la zona de *banners* de la página del centro. Esto facilita al cliente a la hora de cargar las imágenes en su lugar correspondiente.

Por último, la tabla *flyway_schema_history* es una tabla autogenerada por la librería Flyway. En esta se almacenan todas las versiones de migración que se han ejecutado en base de datos. Así bien, cuando un fichero de migración es ejecutado con éxito, en la tabla queda almacenado el registro evitando que en un futuro se vuelva a migrar.

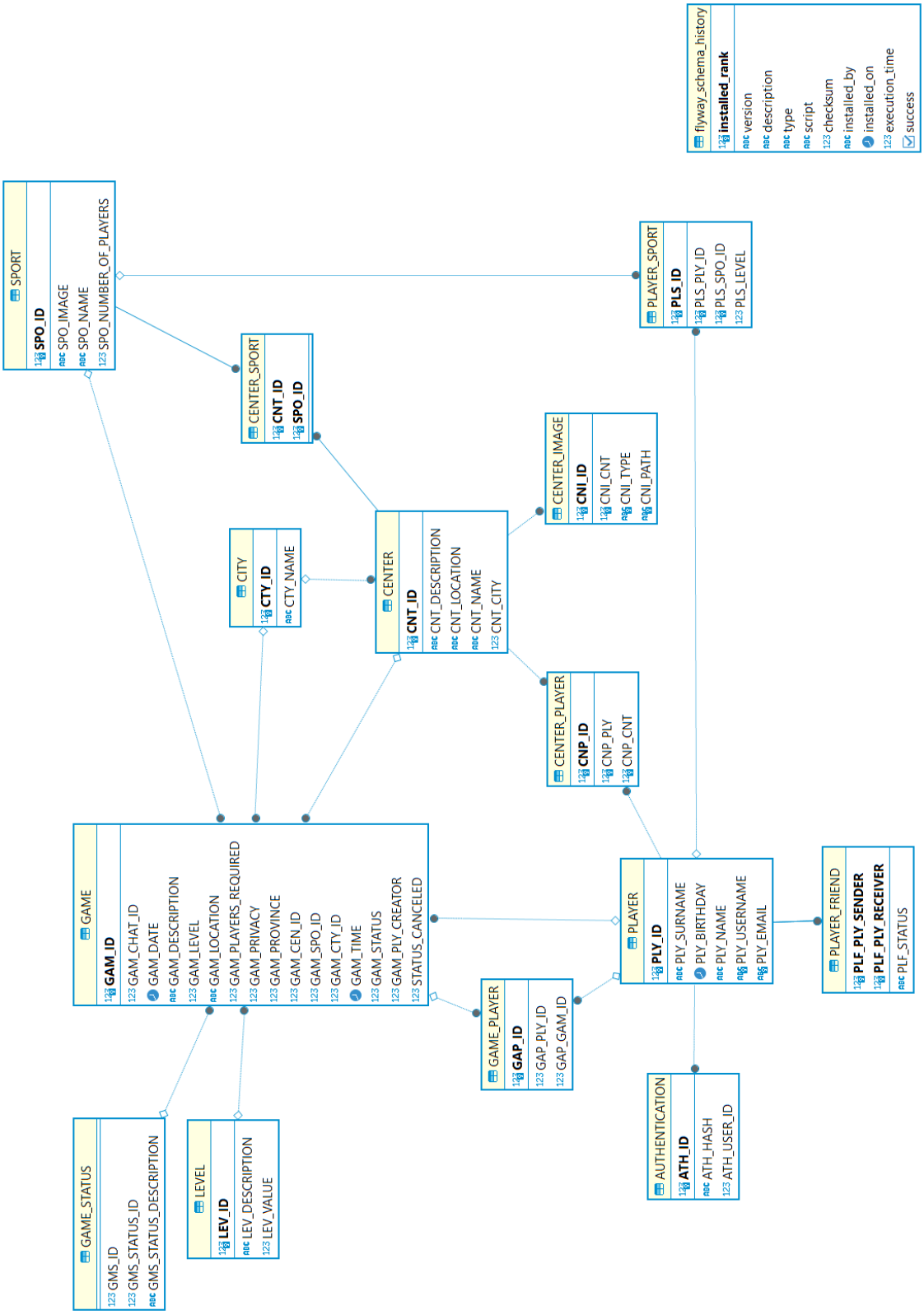

Ilustración 16: Diagrama entidad relación de la base de datos

CAPÍTULO 5
Desarrollo de la solución

En este capítulo se explican los aspectos más importantes de la implementación del aplicativo. Se mostrará la estructura y técnicas utilizadas para cumplir con la especificación de requisitos definida en el capítulo 3 siguiendo siempre la arquitectura y tecnología pautada en el capítulo 4. En primer lugar, se detallará la aplicación cliente y en segundo lugar la aplicación servidor.

5.1 Desarrollo del cliente

La aplicación cliente se basa en el *framework* de Vue.js. Como se explicó en el capítulo 4, utiliza una arquitectura basada en los componentes, focalizándose en la reutilización de ellos a lo largo de toda la aplicación.

5.1.1 Estructura

Los componentes se han dividido siguiendo una distinción de funcionalidad. Las carpetas *cards*, *structure* y *util* contienen componentes reutilizables en toda la aplicación mientras que el resto de las carpetas agrupan componentes según su caso de uso. Los servicios se agrupan también por caso de uso como se muestra en la Ilustración 17.

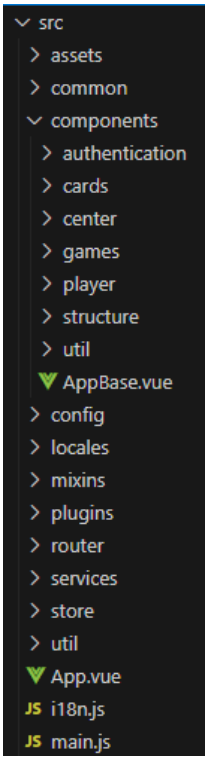

Ilustración 17: Estructura de la aplicación Vue

5.1.2 Reutilización de componentes

Con el fin de reducir el tiempo de desarrollo y para conseguir una aplicación más fácilmente mantenible, se ha seguido una metodología estricta de reutilización de componentes. Se han aplicado diferentes técnicas y se han aprovechado funcionalidades propias de Vue.js para ello.

5.1.2.1 Componentes personalizables

Pese a que los componentes puedan ser reutilizados a lo largo de la aplicación, en ocasiones nos interesa que algunas de sus características sean configurables para adaptarlos al caso de uso específico. Por ejemplo, el componente *CenterCard.vue* declara un botón al final de su plantilla donde se da la posibilidad al usuario de ver más información sobre el centro. No siempre querremos que esa funcionalidad sea visible. Por ello, se hace uso de los *props* [9], datos que el componente padre manda al componente hijo.

El componente *CenterCard.vue* es utilizado para mostrar la información de un centro. La *prop pShowDetails* de tipo *Boolean* puede ser pasada por el componente padre para indicar que el botón de mostrar más detalles no debe visualizarse.

5.1.2.2 Herencia

Durante el desarrollo se ha observado que muchos métodos estaban siendo duplicados a medida que se creaban componentes. Para definirlos todos en un único lugar y evitar la duplicidad se emplea la cláusula *extends*. De esta forma, mediante herencia, el componente hijo hereda todos los métodos del componente padre. Muy útil a la hora de centralizar en un solo lugar las peticiones de datos al servidor.

5.1.2.3 Mixins

Los *mixins* en Vue.js [10] son una técnica de reusabilidad de código que permite definir un conjunto de opciones reutilizables con el propósito de ser incluidas en múltiples componentes. Un *mixin* puede ser definido de forma local o global a todos los componentes de la aplicación. Durante el desarrollo se quiso centralizar la implementación de los métodos relacionados con la obtención y conversión de fechas mediante un *mixin* global. Podemos ver en la Ilustración 18 algunos de los métodos definidos. En la función *mxFormatDate* dado un parámetro *date* de tipo fecha, utilizando la función *format* de la librería *date-fns* se le da un formato *dd/MM/yyyy*. En la segunda, *mxFormatHour*, dada una hora se devuelve un *String* con la hora.

JS mxDate.js

src > mixins > JS mxDate.js > ...

You, 4 weeks ago | 1 author (You)

```
1  import { format } from 'date-fns';
2
3  export const mxDate = {
4    methods: {
5      mxFormatDate(date) {
6        return format(date, 'dd/MM/yyyy')
7      },
8      mxFormatHour(time) {
9        const parts = time.split(":");
10       return parts[0] + ":" + parts[1];
11     }
12   }
13 }
```

Ilustración 18: Ejemplo de mixin

5.1.3 Internacionalización

La aplicación está preparada para traducir sus pantallas a cualquier idioma que se defina. Esto se consigue mediante la librería Vue-i18n. Esta librería hace uso de archivos JSON con las traducciones para cada idioma. Desde las plantillas de los componentes se referencian los elementos del JSON con las traducciones (ver ilustración 22). Por ejemplo, en la clase *CreateGame.vue* se definen los textos de los *steppers* utilizando Vue-i18n (ver Ilustración 19).

```
const messagesi18 = {
  es: {
    createGame: {
      createGame: 'Crear partido',
      sport: 'Deporte',
      selectSport: 'Selecciona un deporte',
      dateAndLocation: 'Fecha y lugar',
      selectDate: 'Selecciona una fecha',
      selectHour: 'Selecciona una hora',
      selectDateRange: 'Selecciona una fecha o un rango',
      selectLocation: 'Elige un lugar',
      selectNumberPlayers: '¿Cuántos jugadores necesitas?',
      selectLevel: 'Indica el nivel de tu partido',
      numberOfPlayers: 'Número de jugadores',
      playersNeeded: '¿Cuántos jugadores buscas?',
      gameDescription: 'Descripción',
      privateGame: 'Solo mis amigos pueden unirse',
      created: '¡Has creado un partido!',
      addComment: 'Añade algún comentario sobre tu partida',
      addProvince: 'Selecciona una provincia',
      addCity: 'Selecciona una ciudad',
      addLocation: 'Indica una dirección',
      addCenter: 'Selecciona un centro'
    },
    general: {
      optional: '(Opcional)'
    },
    btn: {
      accept: 'Aceptar',
```

Ilustración 19: Archivo JSON con traducciones

```
<template>
    <div>
      <p class="mb-0 mt-1" :class="errorClass" v-if="errors.has(fieldName)">
        {{errors.first(fieldName)}}
      </p>
    </div>
  </template>

  <script>
  export default {
    inject: ['$validator'],
    props: {
      errorClass: {
        type: String,
        required: false,
        default: 'validationError'
      },
      fieldName: {
        type: String,
        required: true
      }
    }
  }
  </script>
```

Ilustración 20: Componente ValidationSpan

5.1.4 VeeValidate

Para las validaciones de campos se utiliza la biblioteca de VeeValidate. Para estas validaciones se reutiliza el componente *ValidationSpan.vue*. Como se mencionaba en el apartado de componentes personalizables, este componente hace uso de props para

mostrar el nombre del campo que da el error y los mensajes de advertencia con un estilo personalizado (ver Ilustración 20). También se puede apreciar cómo se inyecta el validador de VeeValidate con la cláusula *inject*. Esto le permite al componente acceder a las funcionalidades de validación proporcionadas por la librería. La propia plantilla define en HTML un texto el cual puede aplicarse una clase si tiene errores con la cláusula *v-if*. En el caso que los haya, el texto aparecerá en color rojo.

5.2 Desarrollo del servidor

En el lado del servidor se implementa una aplicación Spring Boot. Se hace uso de una arquitectura por capas para distinguir cada una según su responsabilidad específica.

5.2.1 Arquitectura en capas

5.2.1.1 Desarrollo de la capa de presentación

La capa de presentación es la encargada de manejar todas las llamadas a la API que expone la aplicación. La estructura utilizada se corresponde con la Ilustración 21.

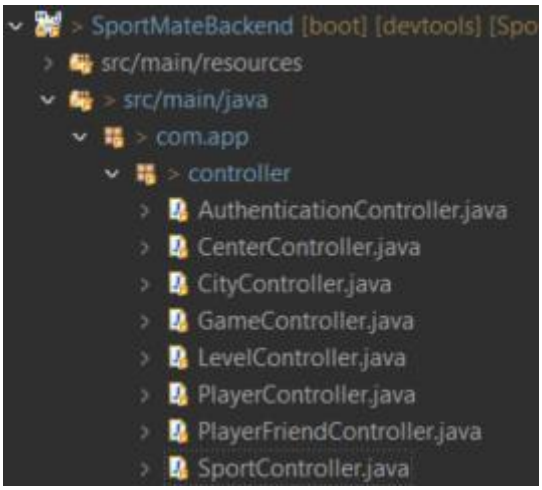

Ilustración 21: Estructura de controladores

- *AuthenticationController.java*: controlador de llamadas HTTP relacionadas con el registro e inicio de sesión así cómo métodos que comprueban usuarios o emails existentes a la hora de crear usuarios.
- *CenterController.java*: controlador de llamadas HTTP relacionadas con la obtención de centros y seguimiento de ellos.
- *CityController.java*: controlador de llamadas HTTP relacionadas con la obtención de las diferentes ciudades disponibles en la aplicación.
- *GameController.java*: controlador de llamadas HTTP relacionadas con la obtención y creación de centros.
- *LevelController.java*: controlador de llamadas HTTP relacionadas con la obtención de los diferentes niveles deportivos disponibles en la aplicación.
- *PlayerController.java*: controlador de llamadas HTTP relacionadas con la obtención y búsqueda de jugadores.
- *PlayerFriendController.java*: controlador de llamadas HTTP relacionadas con solicitudes de amistad por parte de los usuarios.
- *SportController*.java*:* controlador de llamadas HTTP relacionadas con la obtención de deportes.

5.2.1.2 Desarrollo de la capa de servicio

En la parte de implementación de los servicios, se ha seguido un enfoque de clase-interfaz. Todos los servicios declaran una interfaz con los métodos disponibles y una clase que los implementa (ver Ilustración 22). Además de seguir varios principios SOLID [11] como la Inversión de Dependencias (DIP) y el Principio de Responsabilidad Única (SRP), este diseño permite a los controladores de la capa de presentación despreocuparse por la implementación específica del servicio haciendo el código modular y fácil de cambiar.

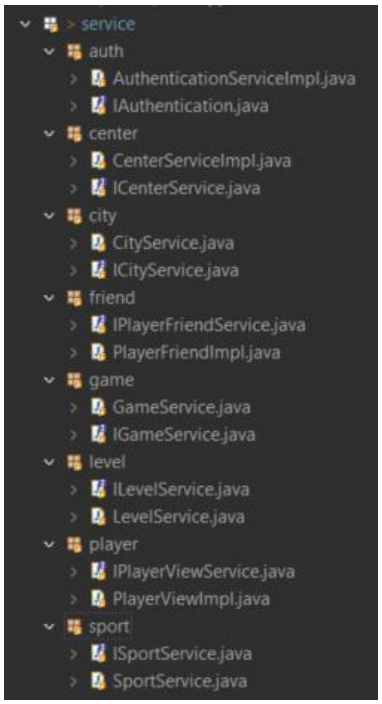

Ilustración 22: Estructura de servicios

5.2.1.3 Desarrollo de la capa de persistencia

La capa de persistencia conecta los servicios con la base de datos. En el proyecto se ha creado un repositorio por cada modelo de datos (ver Ilustración 23). Para ello, se extiende de la interfaz predefinida por JPA con los métodos CRUD [12]. Nótese el uso de la anotación *@Repository* para indicar a Spring que la interfaz debe ser tratada como un *Bean* de tipo repositorio. Con esto, el *framework* será capaz de inyectarlo en otros componentes automáticamente.

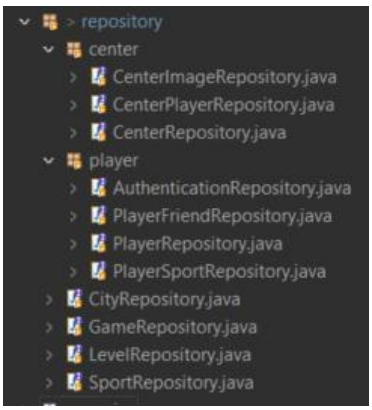

Ilustración 23: Estructura de repositorios

5.2.1.4 Desarrollo de la capa de modelo

En esta capa se definen las clases que representan los modelos de base de datos. Se define una clase por cada modelo (ver Ilustración 24).

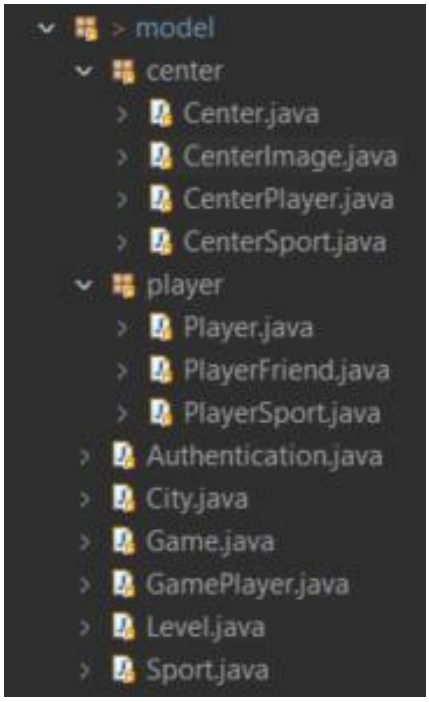

Ilustración 24: Estructura de modelos

En la Ilustración 25 podemos ver cómo se utilizan las anotaciones ofrecidas por Spring para relacionar correctamente las entidades.

- *@Entity*: declara la clase como un *Bean* reconocible para Spring Boot.
- *@Data*: utilizado por la librería Lombock[13] para implementar automáticamente los métodos *setter* y *getter* de la clase.
- *@Table*: se indica el nombre de la tabla de la base de datos que se quiere mapear.
- *@Id*: marca la columna como clave primaria de la tabla.
- *@Column*: nombre de la columna.
- *@NonNull*: indica que la columna no puede ser nula.
- *@ManyToMany*: modela relaciones muchos a muchos.
- *@JoinTable*: declara las FK que apuntan a la tabla intermedia.

[13] https://projectlombok.org/

```java
@Entity
@Data
@Table(name = "PLAYER")
public class Player {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "PLY_ID")
    private Long id;

    @NonNull
    @Column(name = "PLY_USERNAME", unique = true)
    private String username;

    @NonNull
    @Column(name = "PLY_EMAIL", unique = true)
    private String email;

    @NonNull
    @Column(name = "PLY_NAME")
    private String name;

    @NonNull
    @Column(name = "PLY_SURNAME")
    private String surname;

    @NonNull
    @Column(name = "PLY_BIRTHDAY")
    private Date birthday;

    @ManyToMany
    @JoinTable(
        name = "PLAYER_SPORT",
        joinColumns = @JoinColumn(name = "PLS_PLY_ID"),
        inverseJoinColumns = @JoinColumn(name = "PLS_SPO_ID")
    )
    private Set<Sport> sports;
}
```

Ilustración 25: Entidad Player

5.2.2 Data Transfer Objects

Data Transfer Objects o DTO es un patrón de diseño utilizado para transferir datos entre diferentes componentes del sistema. En esta aplicación juegan un papel fundamental, pues aportan múltiples beneficios:

- Protección de los datos: permiten controlar qué datos se transmiten entre los elementos del sistema. Por ejemplo, recibir un modelo de dominio directamente en los controladores podría provocar una inyección de datos maliciosos. Además, si los controladores devuelven modelos de dominio, podrían estar devolviendo datos sensibles de forma involuntaria.
- Separación de responsabilidades: ayudan a separar las preocupaciones entre las capas de la aplicación, mejorando la mantenibilidad del código.
- Flexibilidad: facilitan la adaptación de los datos según los requisitos específicos de cada componente o capa de aplicación.

5.2.3 Almacenamiento de contraseñas

La implementación del almacenamiento de contraseñas en el sistema merece que se profundice un poco más en su desarrollo. Ya que no seguiría las buenas prácticas almacenar la contraseña de un usuario directamente en una columna de base de datos, se optó por crear una tabla adicional que referenciara al usuario mediante una clave foránea. En esta nueva tabla, mediante el algoritmo Argon2, se guarda un *hash* que se utiliza cada vez que el usuario inicia sesión. De esta forma, cuando el usuario se registre se guardará este *hash* en base de datos. Cuando quiera iniciar sesión el servidor

recuperará ese *hash* y lo comparará con la contraseña proporcionada (ver Ilustración 26).

```java
@Override
public Player registerPlayer(RegisterRequest registerRequest) {
    Player requestPlayer = registerRequest.getPlayer();

    playerRepo.save(requestPlayer);

    try {
        String hashedPassword = argon2.hash(Constants.ARGON2_ITERATIONS,
                Constants.ARGON2_MEMORY, Constants.ARGON2_PARALELISM, registerRequest.getPassword());
        Authentication authRequest = new Authentication(requestPlayer.getId(), hashedPassword);
        authRepo.save(authRequest);
    } finally {
        argon2.wipeArray(registerRequest.getPassword().toCharArray());
    }

    for (SportRequest sportRequest : registerRequest.getSportList()) {
        PlayerSport playerSport = new PlayerSport(requestPlayer.getId(), sportRequest.getSport().getId(),
                sportRequest.getLevel().getId());
        playerSportRepo.save(playerSport);
    }

    return requestPlayer;
}

@Override
public Player login(String username, String password) {
    Optional<Player> player = playerRepo.findByUsername(username);
    if (!player.isPresent()) {
        throw new InternalException(ErrorCodes.USER_NOT_EXISTS, "El usuario no existe");
    }

    if (!checkLogin(player.get().getId(), password)) {
        throw new InternalException(ErrorCodes.WRONG_PASSWORD, "Contraseña incorrecta");
    }

    return player.get();
}
```

Ilustración 26: Código de registro e inicio de sesión

5.2.4 Gestión de errores

Para la correcta gestión de errores y una interpretación correcta del cliente, se definió una clase *InternalException* que proporciona un mensaje y código explicativos del error. Este código de error debe ser interpretable por el cliente en caso necesario. En la clase *ErrorCodes* se definen los errores devueltos por el servidor (ver Ilustración 27).

```java
public class ErrorCodes {

    // Auth
    public static final int EMAIL_ALREADY_EXISTS = 1000;
    public static final int USER_ALREADY_EXISTS = 1001;

    public static final int USER_NOT_EXISTS = 2000;
    public static final int WRONG_PASSWORD = 2001;


    // Game
    public static final int JOIN_PLAYER_ERROR = 3000;
    public static final int LEAVE_PLAYER_ERROR = 3001;
    public static final int CANCEL_PLAYER_ERROR = 3002;
    public static final int FINISHED_GAMES_ERROR = 3003;
    public static final int PLAYER_GAMES_ERROR = 3004;

    // Friends
    public static final int REPLY_FRIEND_REQUEST_ERROR = 4000;

    // Center
    public static final int CENTER_NOT_FOUND = 5000;
    public static final int CENTER_PLAYER_NOT_FOLLOWING = 5001;

    private ErrorCodes() {

    }
}
```

Ilustración 27: Clase con los códigos de error

5.3 Metodología empleada en el desarrollo

La metodología empleada a la hora de desarrollar el proyecto se basó en la combinación de dos enfoques ágiles conocidos: Scrum y Kanban [13]. ScrumBan combina elementos de ambos para adaptarse a ciertas necesidades y entornos. Adopta los eventos de Scrum como los *sprints* pero también la flexibilidad de Kanban en la gestión del flujo de trabajo [14].

5.3.1 JIRA

La herramienta para gestionar y organizar el trabajo fue JIRA[14]. Con este software se hizo uso de los *sprints* y el tablero Kanban, así como elementos básicos de JIRA tales como épicas, historias, tareas y errores.

En este proyecto se dividió la implementación de las distintas funcionalidades en cuatro épicas distintas. En JIRA, una épica representa una entidad que agrupa un conjunto de historias relacionadas para representar un objetivo común más grande. Proporcionan una visión a alto nivel de las características que se deben desarrollar. Así pues, en cada *sprint* se implementó una o varias épicas con el objetivo de garantizar un desarrollo estructurado y una funcionalidad incremental.

Respecto a las historias, se utilizaron los casos de uso definidos en el capítulo 3 de esta memoria como guía para el desarrollo de la funcionalidad (ver Ilustración 28). Dentro de una historia se puede encontrar un desglose en subtareas, comúnmente diferenciadas según si la implementación era necesaria en la aplicación cliente o servidor. Gracias a la integración de JIRA con Git, desde la propia historia o tarea se creaban las ramas para el desarrollo.

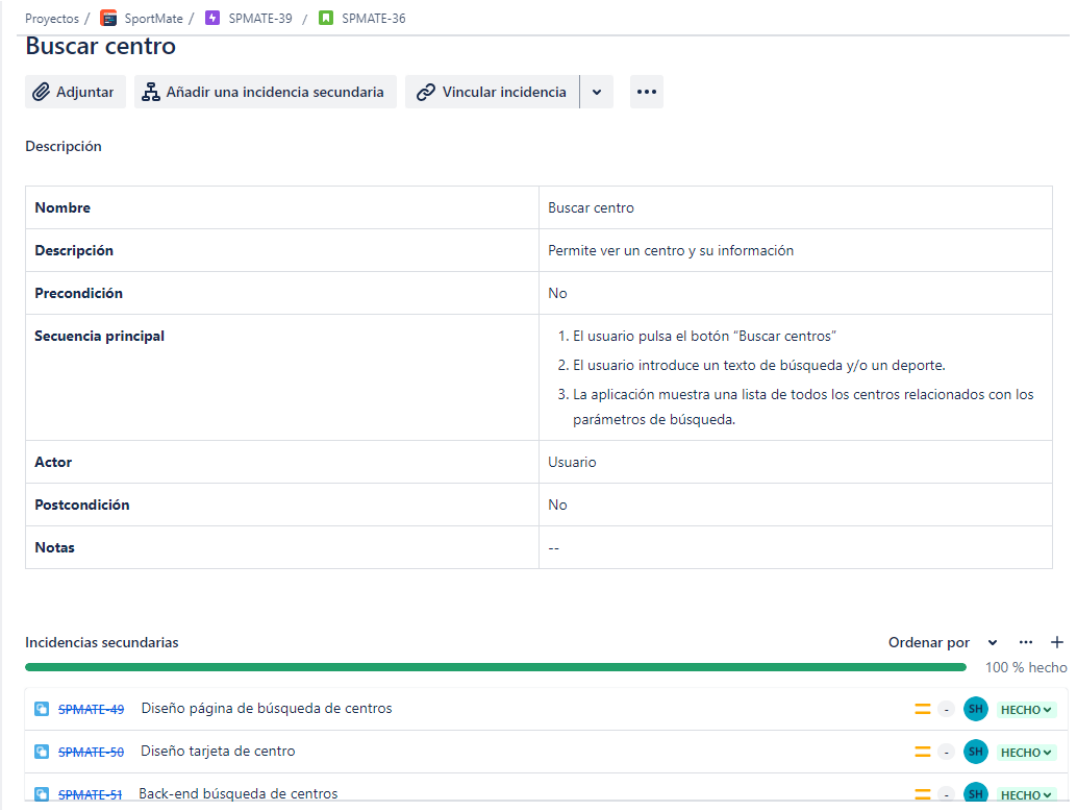

Ilustración 28: Historia con subtareas en JIRA

[14] https://www.atlassian.com/es/software/jira

5.3.2 Organización en *sprints*

Con el objetivo de definir objetivos claros y de priorizar tareas se dividió el trabajo en cuatro *sprints*.

Sprint 1

En el primer *sprint* se realizaron tareas relacionadas con el inicio del proyecto, tanto la búsqueda de las tecnologías como los primeros pasos con ellas. Al término de este se consiguió configurar una versión primaria de la aplicación, preparada para el comienzo de los desarrollos. En la parte del cliente se construyó una base de la aplicación sobre la que añadir nuevos diseños y pantallas mientras que en la parte del cliente se configuró el servidor y algunos temas de conexión entre ambas aplicaciones. En esta fase se realizaron algunas pruebas de integración, comentadas en el capítulo 6 de esta memoria.

Sprint 2

El objetivo del segundo *sprint* fue el de la implementación de los módulos de autenticación y de actividades. El desarrollo comenzó con el desarrollo de las tareas del módulo de actividades. Sin embargo, a la hora de desarrollar algunas funcionalidades se hizo evidente que era posible sin antes haber completado el módulo de autenticación. Por ejemplo, a la hora de crear una actividad era necesario indicar el creador de esta. La idea de dejar el desarrollo a mitad hasta que quedase finalizado el módulo de autenticación no parecía interesante, por lo que se cambió el rumbo y se comenzó con las tareas de registro, inicio de sesión y cierre de sesión.

Al cerrar el *sprint* quedaron por hacer varias tareas del módulo de actividades debido a la falta de tiempo: algunas configuraciones iniciales realizadas durante el primer *sprint* no funcionaron como se esperaba. Por ejemplo, en la parte del cliente se encontraron algunas incompatibilidades entre versiones de librerías y Vue.js. Debido al tiempo empleado en la búsqueda de soporte, su resolución y una estimación errónea de las horas de desarrollo, todas esas tareas pendientes se pasaron al siguiente *sprint* como prioritarias.

Sprint 3

El tercer *sprint* se compuso de tareas relacionadas con el módulo de usuarios, añadiendo tareas pendientes del *sprint* anterior. Durante el transcurso de este *sprint* se dieron cambios en varias tareas relacionadas con la implementación de algunos casos de uso. Esto fue causado por una apreciación que no se hizo durante la fase de recogida de requerimientos. En concreto, se hizo evidente ante la gran cantidad de actividades en las que un usuario podía formar parte que necesitaba un sitio donde poder consultar la fecha y el horario de cada una de ellas. Por esto, se añadió un nuevo caso de uso en el que se requería de la implementación de una nueva pantalla donde el usuario pudiera ver en un calendario sus partidos pasados y futuros. Esta funcionalidad era esencial para garantizar una buena experiencia al usuario final.

Por el contrario, este tiempo de desarrollo tuvo que retirarse de la implementación de otras tareas. La lista de actividades de amigos permitía a los usuarios ver los partidos privados de sus usuarios de confianza. Al no suponer un cambio negativo drástico para la aplicación se optó por eliminar esta funcionalidad para poder añadir la del visionado de actividades en el calendario.

Sprint 4

Por último, en el cuarto *sprint* se llevó a cabo el desarrollo del módulo de centros. Se hizo hincapié en el diseño de la página de los centros, pues suponía un elemento clave para el proyecto que esta página fuera atractiva. Fue el *sprint* más corto de los cuatro puesto que este módulo tenía menos tareas que el resto. Además, durante este *sprint* se corrigieron algunas incidencias reportadas a lo largo del desarrollo. Por ejemplo, con la modificación de algunos campos y refactorizaciones del módulo de actividades, el buscador implementado en el *sprint* 2 dejó de funcionar.

En la Ilustración 29 aparece el cronograma que proporciona JIRA para visualizar los *sprints* a lo largo del tiempo.

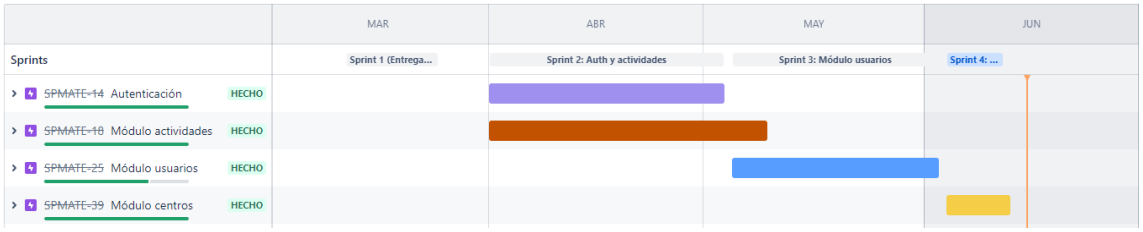

Ilustración 29: Cronograma con sprints

5.3.3 Git

Git[15] es un sistema de control de versiones muy famoso en la comunidad de desarrollo de software. Pese a que este proyecto fue implementado por un solo desarrollador, es ampliamente recomendado utilizar convenciones a la hora de emplear Git. Estas convenciones pueden suponer múltiples ventajes como la facilidad para revisar el código o la reducción de errores.

Convenciones

A lo largo del proyecto se han utilizado técnicas de nombramiento de ramas y *commits*. Todos los nombres y descripciones se realizan en inglés.

Estructura de ramas

Las ramas se han organizado según su tipología. En la carpeta *fix* se encuentran todas las ramas relacionadas con la corrección de errores. En la carpeta *feat* las ramas que implementan nuevas funcionalidades. Además, la rama *develop* actúa como *pre-release* al hacer *merge* de todo el nuevo código a medida que se termina. La rama *master* contiene la última versión de la aplicación y se actualiza al final de cada *sprint* (ver Ilustración 30).

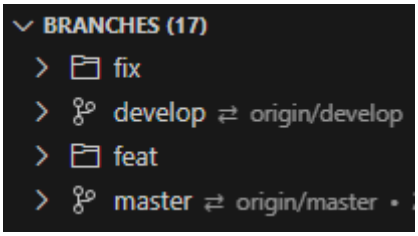

Ilustración 30: Estructura de ramas Git

[15] https://www.git-scm.com/

Nombre de ramas
El nombre de las ramas sigue la siguiente estructura: [Identificador de la incidencia de JIRA]-[nombre indicativo por parte del desarrollador]. Por ejemplo, para la subtarea -que aparece en la Ilustración 28, el nombre de la rama creada fue: SPMATE-49-search-center.

Commits
Los *commits* siguen una convención idéntica a las ramas. Para cada uno se indica su intención: añadir funcionalidad (*feat*) o corregir un error (*fix*). La estructura sería: [feat/fix]: [Mensaje descriptivo del *commit*] (ver Ilustración 31).

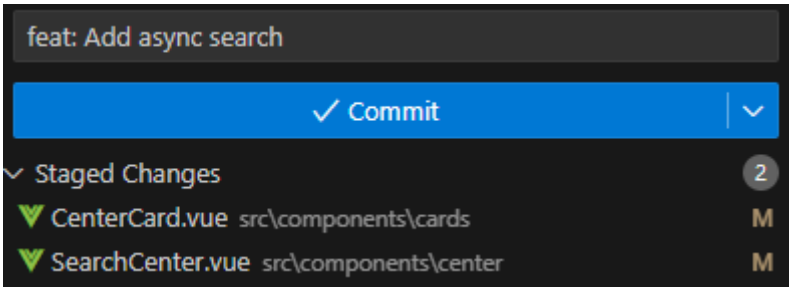

Ilustración 31: Estructura de un commit en Git

5.4 Resultado final

Para explicar el funcionamiento y valorar el resultado final de la aplicación, en este apartado se detallarán las distintas funcionalidades junto a algunas capturas de pantalla extraídas desde el navegador web *Google Chrome* [16].

Al entrar en la web se muestra la página de inicio. En toda la aplicación está disponible un menú superior y un menú lateral desplegable. En el menú superior se encuentra la opción de inicio o cierre de sesión según si el usuario ha accedido previamente. En el menú lateral volvemos a encontrar la opción de iniciar sesión o de registrarse. Justo debajo encontramos algunas funcionalidades que describiremos más adelante. Algunas de ellas solo estarán disponibles si iniciamos sesión.

Para registrar un perfil de usuario la aplicación nos guía a través de varias *slides* donde rellenar información (ver Ilustración 32). En la primera se solicita un correo y un nombre de usuario. Antes de pasar a la siguiente *slide* la aplicación comprueba que ninguno de los dos esté siendo utilizado por otra cuenta. En la segunda aparecen campos con datos como el nombre, apellido, fecha de nacimiento y contraseña, que debe coincidir en ambos casos. En la tercera se le pide al usuario que indique cuáles son sus deportes favoritos y qué nivel tiene en cada uno de ellos. Finalmente, si el proceso se completa adecuadamente se llega a la última *slide* confirmando que el registro ha sido satisfactorio. Durante la cumplimentación de todos los campos encontraremos validadores. Por ejemplo, en el caso que pongamos un correo mal escrito la aplicación nos mostrará un mensaje advirtiendo de ello y no nos permitirá pasar a la siguiente *slide*.

Una vez iniciada sesión, podemos crear una actividad desde la opción situada en el menú lateral. El diseño de la interfaz es parecido al del registro. A lo largo de varias

[16] https://www.google.com/intl/es_es/chrome/

slides se le pide al usuario información relativa a la actividad: el deporte, la fecha, la hora, el lugar, el nivel, el número de jugadores y una descripción opcional (ver Ilustración 33). De nuevo, si el proceso se completa adecuadamente aparecerá una *slide* advirtiendo de ello.

Otra funcionalidad interesante es la de ver las actividades en un calendario. Al navegar a esta página la aplicación ofrece todos los partidos a lo largo del tiempo independientemente de si han sido jugados con anterioridad (ver Ilustración 34). Para ver más información se puede hacer clic sobre alguno de ellos y la aplicación redirigirá a la página de visualización de la actividad. En esta página se pueden ver todos los datos del partido, una lista de los usuarios que se han unido y un botón para unirse, abandonar o cancelar la actividad. Además, si la actividad se desarrolla en un centro deportivo, se muestra en la parte derecha de la pantalla junto a una imagen y una breve descripción.

La aplicación dispone de un buscador donde pueden encontrarse actividades según las preferencias del usuario. En la parte izquierda de la pantalla aparecerán los distintos filtros que se pueden aplicar en la búsqueda: deporte, nivel, lugar y fecha. Si existen resultados se mostrarán en una lista a la derecha junto a algunos datos básicos. Haciendo clic sobre la tarjeta la aplicación redirigirá a la página de la actividad y podrá verse en detalle su información.

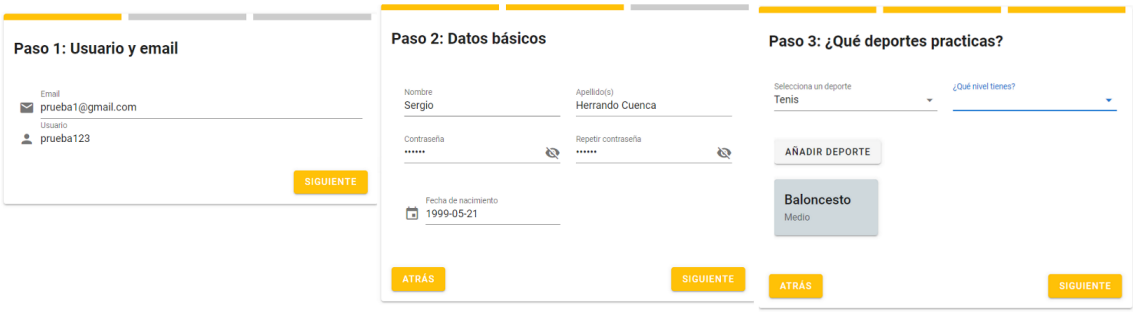

Ilustración 32: Proceso de registro en SportMate

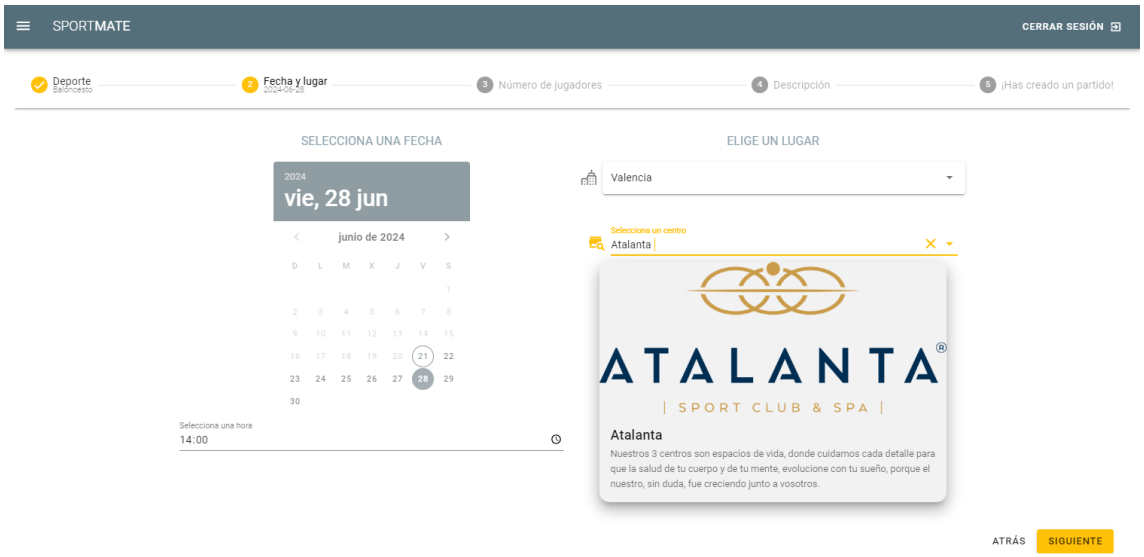

Ilustración 33: Creación de actividad en SportMate

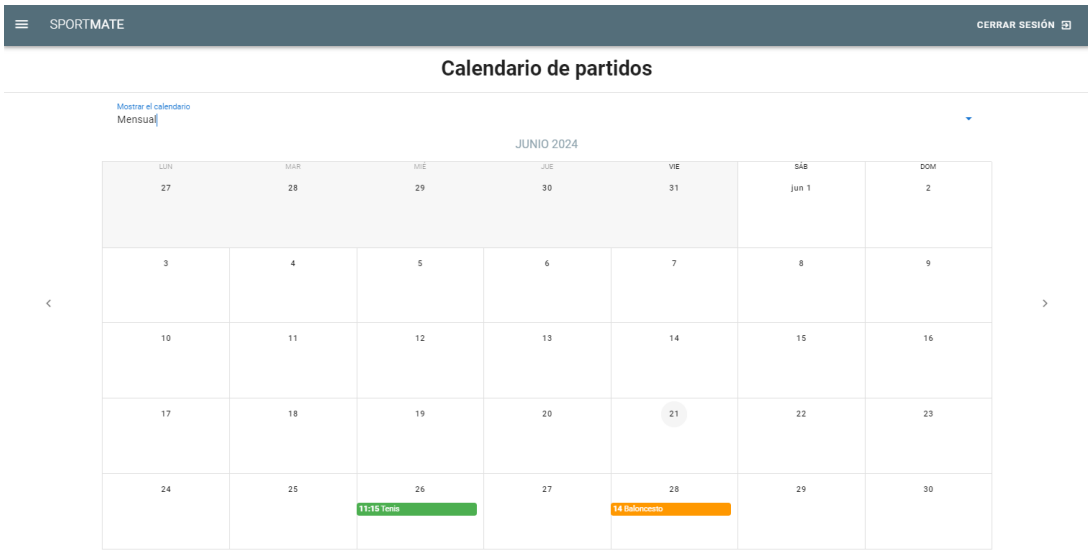

Ilustración 34: Calendario de actividades en SportMate

Además del buscador de actividades, también se encuentra disponible el buscador de jugadores. Este buscador encuentra jugadores con un nombre de usuario, nombre o apellidos que coincidan con el texto de búsqueda. Los resultados aparecen en una lista vertical y que el usuario podrá seleccionar para ver más detalles sobre él: nombre, apellidos, nombre de usuario y deportes practicados. En la vista del jugador también se tendrá la posibilidad de mandar una solicitud de amistad. En el caso que ya se haya enviado aparecerá un icono con un mensaje advirtiéndolo al igual que si ya ha sido aceptada y ambos jugadores son amigos.

Es posible consultar la lista de amistades desde una página dedicada a estas. Aquí, además de listar a todos los usuarios amigos, aparecerá una lista con las solicitudes de amistad pendientes de aprobar. Una vez se acepte o rechace la solicitud se actualizarán los datos de la página.

Si un jugador desea encontrar un centro deportivo donde practicar su deporte puede hacerlo desde el buscador de centros. Aquí pueden buscarse centros según su nombre, ciudad o deportes disponibles para practicar en sus instalaciones. Los resultados aparecerán en una lista debajo del cuadro con los filtros (ver Ilustración 35). En cada resultado aparece un botón donde hacer clic y navegar a la página del centro para obtener más información.

La página de centro contiene toda la información relativa al mismo. En la parte superior se encuentra un carrusel con fotos que van pasando cada cierto tiempo. Debajo, aparece el nombre del centro y un botón con el que seguirlo. Tras una descripción y la dirección, aparece de nuevo otro carrusel con todos los deportes ofrecidos. Finalmente, aparece una lista con los usuarios que siguen al centro (ver Ilustración 36 e Ilustración 37).

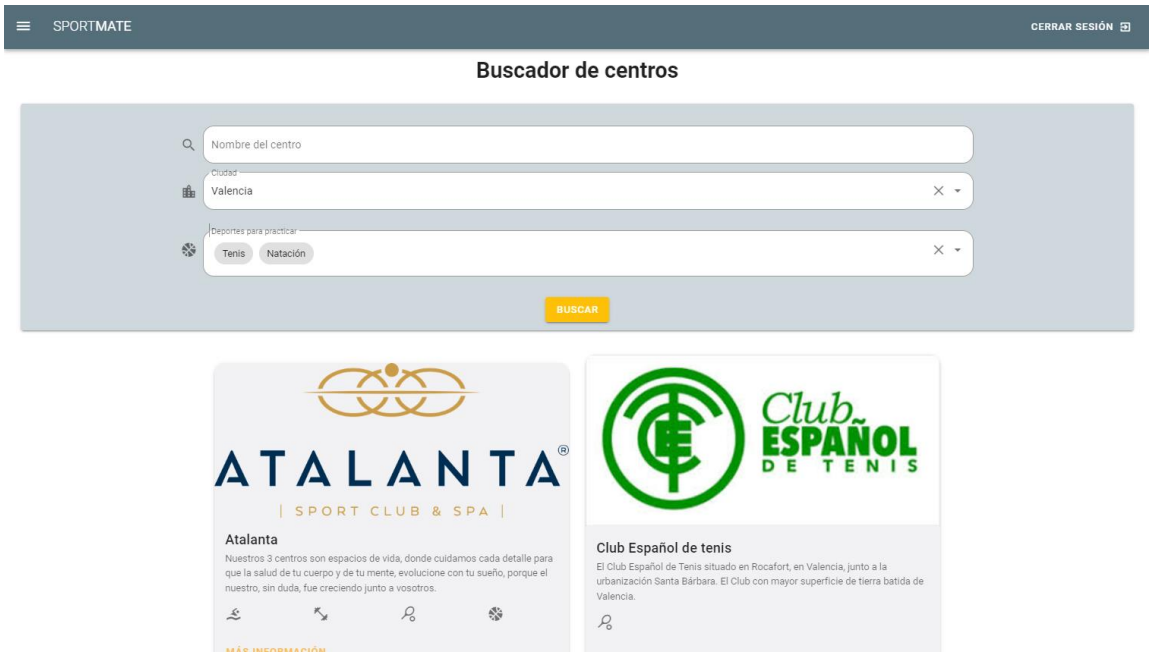

Ilustración 35: Buscador de centros en SportMate

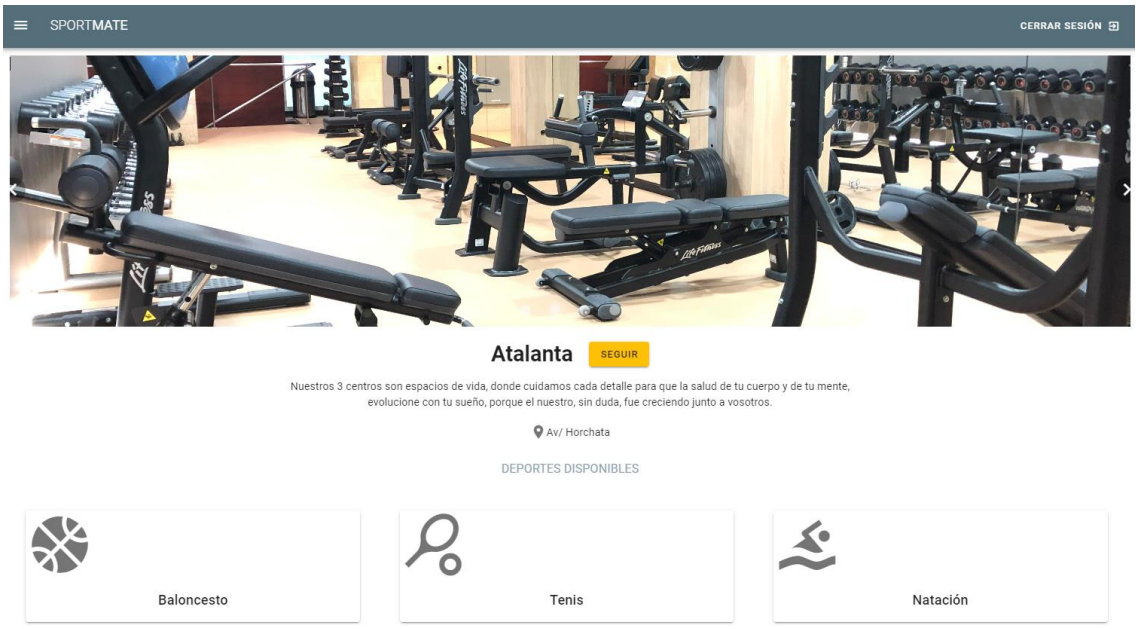

Ilustración 36: Página de centros en SportMate

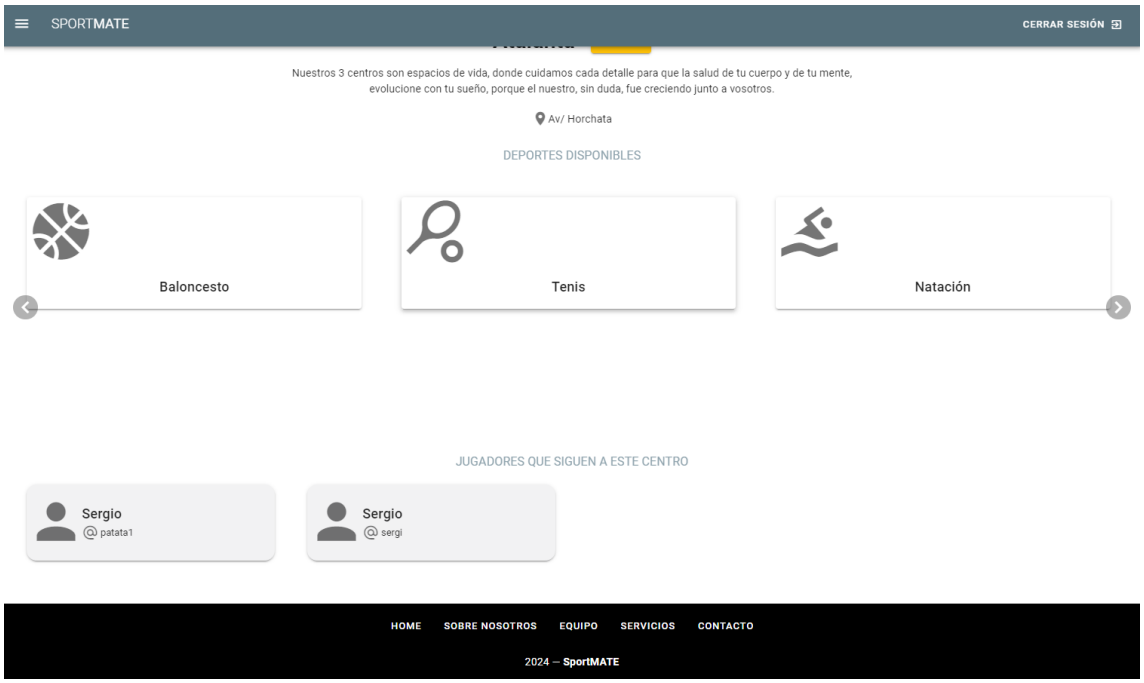

Ilustración 37: Página de centro con usuarios en Sport Mate

CAPÍTULO 6
Pruebas

La ejecución de pruebas en un proyecto software constituye una fase fundamental. Durante esta fase se asegura la calidad del código producido, detectando errores en las fases tempranas del desarrollo y reduciendo el coste de corrección de cara al futuro. Un buen plan de pruebas puede garantizar la calidad del producto final asegurando también el buen comportamiento de todas las piezas del sistema ante situaciones diversas. En este capítulo se detallan los dos tipos de prueba que se llevaron a cabo durante el proyecto: pruebas unitarias, pruebas de integración y pruebas de aceptación.

6.1 Pruebas unitarias

Las pruebas unitarias son un tipo de prueba de software que se centra en la evaluación de elementos individuales de un programa, también conocidos como unidades. Son partes más pequeñas de un código que pueden ser aislados y probados de manera independiente. El objetivo principal es validar que estas unidades de código cumplan su función de manera aislada, sin depender de otras, y se comprueba que cada unidad devuelve el resultado esperado.

En el proyecto se ha utilizado el marco de pruebas de JUnit[17] y Mockito[18]. JUnit proporciona anotaciones, inserciones y utilidades que facilitan la creación y ejecución de pruebas. Mockito es una biblioteca para la creación de objetos simulados (*mocks*). Estos *mocks* son objetos que imitan el comportamiento de otros componentes.

```
@BeforeEach
void setUp() {
    argon2 = Argon2Factory.create();
    authService.argon2 = argon2;
}

@Test
void testRegisterPlayer() {
    RegisterRequest registerRequest = mock(RegisterRequest.class);
    Player player = mock(Player.class);
    when(registerRequest.getPlayer()).thenReturn(player);
    when(registerRequest.getPassword()).thenReturn("password");

    SportRequest sportRequest1 = mock(SportRequest.class);
    SportRequest sportRequest2 = mock(SportRequest.class);
    when(registerRequest.getSportList()).thenReturn(List.of(sportRequest1, sportRequest2));

    when(playerRepo.save(any(Player.class))).thenReturn(player);

    authService.registerPlayer(registerRequest);

    verify(playerRepo, times(1)).save(player);
    verify(authRepo, times(1)).save(any(Authentication.class));
    verify(playerSportRepo, times(2)).save(any(PlayerSport.class));

    verify(registerRequest, times(1)).getPassword();
}
```

Ilustración 38: Prueba unitaria del método registerPlayer

[17] https://junit.org/junit5/
[18] https://site.mockito.org/

Como ejemplo, en la Ilustración 38 tenemos la prueba unitaria para el registro de jugadores con el método *testRegisterPlayer*. En primer lugar, *mockeamos* las dependencias de *PlayerRepository*, *PlayerSportRepository* y *AuthenticationRepository* a nivel de clase. Segundo, inyectamos estos *mocks* en una instancia de *AuthenticationServiceImpl* con la anotación *@InjectMocks*. Tercero, se configura un Argon2 simulado en el método *setup*. La anotación *@BeforeEach* hace que antes de cada prueba se ejecute el código de ese método. Cuarto, y ya dentro del método *testRegisterPlayer*, se crea un *RegisterRequest* y otros objetos necesarios como *mocks*. Quinto, configura los comportamientos esperados para todos los *mocks*. Sexto, llama al método *registerPlayer*. Por último, verifica que los métodos fueron llamados con los argumentos correctos.

6.2 Pruebas de integración

Las pruebas de integración se centran en verificar la correcta interacción entre los distintos componentes de un sistema. Una vez probados estos componentes individualmente con las pruebas unitarias, las pruebas de integración tratan de comprobar que interactúen correctamente entre ellos.

En nuestro proyecto, constituía de vital importancia que el cliente lograse conectar con el servidor para realizar peticiones. Se probó el correcto funcionamiento con diferentes llamadas *GET*, *POST*, *PUT* y *DELETE*. También se contrastaron las diferentes rutas de las llamadas y el manejo adecuado de estas por los controladores del servidor.

6.2.1 Problemas encontrados: CORS

Durante las primeras fases del proyecto, haciendo la configuración inicial del mismo, se ejecutaron varias pruebas de integración entre el cliente y el servidor. Al realizar llamadas desde el cliente, estas eran devueltas por el servidor bajo un mensaje de error. CORS o *Cross Origin Error Solutions* [15] es una política de seguridad de los navegadores web para proteger a los usuarios y sus datos al interactuar con recursos ubicados en dominios diferentes. Puesto que nuestro cliente y servidor se encuentran en puertos distintos, tuvimos que configurar CORS de manera global en la aplicación desde el servidor. En la Ilustración 39 se muestra un ejemplo de cómo se añadió un *Bean* de Spring Boot con la anotación *@Configuration* para admitir el origen del cliente y los métodos.

```
@Configuration
public class CorsConfig implements WebMvcConfigurer {
    @Override
    public void addCorsMappings(CorsRegistry registry) {
        registry.addMapping("/**")
                .allowedOrigins("http://localhost:8080")
                .allowedMethods("GET", "POST", "PUT", "DELETE")
                .allowedHeaders("*");
    }
}
```

Ilustración 39: Configuración del CORS en Spring Boot

6.3 Análisis estático de código

Durante el desarrollo se habilitó la herramienta SonarLint[19] en el entorno de desarrollo. Su propósito es detectar y corregir problemas mientras se escribe el código. Es compatible con múltiples lenguajes: en nuestro caso Java y JavaScript. Ante la detección de un error, la herramienta lo explica y en la mayoría de los casos proporciona ejemplos para solucionarlo.

Ilustración 40: Advertencia de error de SonarLint

En la Ilustración 40 se muestra un ejemplo de advertencia de SonarLint. En este controlador, estábamos implementando un método que gestionaba las llamadas con la ruta */game/joinGame*. En una primera fase, se declaró en los parámetros del método una clase *Entity*. Estas clases mapean objetos de Java con tablas de base de datos. Inmediatamente SonarLint lanzó la advertencia con la explicación del problema. Se refactorizó y pasó a utilizarse un DTO como se explica en el capítulo 5, durante el desarrollo del servidor (ver Ilustración 41).

```
@PostMapping(value = "/joinGame")
public ResponseEntity<GameDTO> joinGame(@RequestBody GamePlayerDTO game, HttpServletRequest request,
        HttpServletResponse response) {

    return new ResponseEntity<>(gameService.joinGame(game), HttpStatus.OK);
}
```

Ilustración 41: Controlador refactorizado con DTO

6.4 Pruebas de aceptación

Las pruebas de aceptación sirven para valorar el resultado final de un producto y determinar si está listo para entregarse al cliente o usuario final.

En las últimas fases de este proyecto se preparó un formulario con algunas preguntas para conocer la opinión de un potencial usuario de la aplicación. Además, se le pedía que comentase cualquier incidencia o sugerencia que le surgiera durante el uso de la web. Se concretaron dos sesiones en las que el usuario rellenaría el mismo formulario con el fin de evaluar la evolución de la aplicación siguiendo los comentarios de mejora aportados en la primera sesión.

El usuario que realizó las pruebas fue un joven de veinte años, muy acostumbrado a las interfaces web y móviles. Practica varios deportes entre ellos el tenis y el baloncesto. Su

[19] https://www.sonarsource.com/products/sonarlint

forma de organizar actividades deportivas es mediante la aplicación de mensajería WhatsApp[20]. La prueba la realizó en el navegador web Google Chrome.

Sesión 1

En esta sesión el usuario interactuó por primera vez con la aplicación. La satisfacción del usuario al término de la sesión fue muy positiva y valoró entre otras cosas el buen rendimiento.

Por otro lado, aparecieron algunos comportamientos no deseados durante las pruebas. El primero fue un mal funcionamiento del buscador de partidos en el que al añadir el nivel del partido a los filtros la llamada al servidor fallaba. Se solucionó cambiando la *query* de JPA que leía de la tabla de actividades. Otro fallo fue que al registrarse la aplicación no bloqueaba pasar al siguiente paso si no se introducía la fecha de nacimiento. Desde el cliente se optimizó el código y se bloqueó. Finalmente, en la página de actividad no se veía bien el número de jugadores. No estaba contando al creador, solo a los jugadores que se habían unido.

Sesión 2

Tras solucionar los problemas descritos en la primera sesión se volvió a realizar otra prueba para que el usuario evaluara si la experiencia había mejorado. En efecto, aunque el usuario solo cambiase la respuesta en una de las preguntas del formulario, en observaciones apenas comentó que algunas imágenes no se veían con toda la resolución posible, lo cual es problema del propio archivo de imagen. También comentó que al iniciar sesión la aplicación no advertía al usuario en ningún momento de la carga que estaba realizando en el momento, llevando a una confusión.

Conclusiones de las pruebas

Tras ambas sesiones se pudo evaluar que la aplicación está lista para su uso. Presenta una funcionalidad que cumple las expectativas del usuario y destaca por su gran rendimiento. Sin embargo, la aplicación debe transmitir mayor seguridad a sus usuarios, pues en ambas sesiones se indicó que la cesión de datos no generaba confianza.

[20] https://www.whatsapp.com/

CAPÍTULO 7
Conclusiones

En este capítulo de conclusiones se pretende hacer un recorrido por todas las fases del proyecto analizando el trabajo realizado y la consecución de los objetivos propuestos.

En primer lugar, se realizó una investigación sobre otras aplicaciones parecidas a la propuesta en esta memoria. Se detectaron los puntos fuertes y débiles de cada una ayudando a formar ideas para la futura fase de especificación. Fue durante esta primera fase de investigación donde aprendimos a analizar otras aplicaciones poniendo en valor sus funcionalidades, pero también siendo críticos con otros aspectos menos valiosos.

En segundo lugar, se definió la especificación de requisitos. Se detalló qué debía incluir la aplicación final y cuál debía ser su comportamiento con el estándar de especificación de requisitos IEEE-STD-830-1998. Además, después de la fase de desarrollo, se puso aún más en valor esta fase de especificación, pues dependió en gran medida de esta que el desarrollo fuera claro y con unas pautas marcadas. Aunque pueda considerarse que se realizó un buen trabajo, no fue perfecto y de cara a futuros proyectos se mejorarán algunos aspectos como por ejemplo la definición de algunos casos de uso en los que faltaba más detalle de la funcionalidad deseada.

En tercer lugar, se hizo el diseño de la solución. Aquí se buscaron y analizaron las tecnologías a emplear durante el desarrollo y se determinaron siguiendo las necesidades de la futura aplicación. También se eligió la arquitectura del cliente y del servidor utilizando patrones de arquitectura comunes en ingeniería del software. De esta fase tomamos como lección la importancia de un buen diseño para desarrollar un producto software de calidad, mantenible y escalable.

En cuarto lugar, se implementaron las funcionalidades de la aplicación en la fase de desarrollo. Fue el punto de encuentro de los capítulos anteriores donde todo lo detallado se plasmó en una aplicación software. Además, durante todo el desarrollo fueron de la mano las pruebas para asegurar una calidad y un cumplimiento de las funcionalidades.

La finalidad de este proyecto era desarrollar una aplicación web para promocionar el deporte y que los centros deportivos pudieran publicitarse dentro de ella. Con las funcionalidades de creación de actividades junto con la de la creación de perfiles de usuario se ha conseguido el primero de los propósitos. Los usuarios pueden conocer a otros mediante la aplicación buscando actividades de sus deportes favoritos. Además, otras funcionalidades como la vista de calendario aportan un gran valor y promocionan el uso de la aplicación fomentando así la práctica deportiva. Por otro lado, los centros deportivos tienen una gran visibilidad para los usuarios, permitiéndoles ofertar sus instalaciones y aumentando su visibilidad.

Además, se ha ganado un gran conocimiento en tecnologías actuales como Vue.js y Spring Boot, se ha aprendido a utilizar librerías y cómo combinarlas para obtener el mejor resultado, se han implementado patrones de diseño software y se han seguido convenciones y buenas prácticas durante el desarrollo.

En definitiva, los objetivos del proyecto se han cumplido con creces y el trabajo realizado ha aportado un gran conocimiento y lecciones de cara al futuro dentro del mundo de la ingeniería del software.

7.1 Relación con los estudios cursados

El grado en ingeniería informática y los conocimientos impartidos han jugado un papel fundamental en la consecución del proyecto.

La asignatura de *"Ingeniería del software"* nos ha ayudado a la hora de comprender la importancia de las metodologías ágiles en los proyectos, su arquitectura y sus pruebas. *"Interfaces Persona Computador"* ha servido de base a la hora de diseñar las interfaces de la aplicación. *"Bases de datos"* nos ha proporcionado unos conocimientos esenciales para poder diseñar la estructura de entidades y relaciones en nuestra base de datos. Además, durante mi estancia en la Universidad Politécnica de Breslavia con el programa Erasmus aprendí a desarrollar aplicaciones web con la asignatura *"Web Applications"*.

7.1.1 Competencias transversales

Algunas de las competencias transversales adquiridas durante el grado de Ingeniería Informática que han sido aplicadas durante el desarrollo de este proyecto son:

- Análisis y resolución de problemas: Durante todas las fases del proyecto se ha hecho frente a distintos problemas. La correcta resolución de ellos ha sido determinante para conseguir avanzar y no volver a encontrarlos en el futuro.
- Instrumental específica: Combinando la utilización de distintos *frameworks* de desarrollo, librerías y lenguajes de programación.
- Planificación y gestión del tiempo: Al compaginar la vida laboral y el desarrollo del proyecto al mismo tiempo la planificación del tiempo ha sido esencial.

7.2 Trabajo futuro

Pese a que la aplicación sea totalmente funcional y cumpla con los objetivos propuestos, aún quedan muchas facetas por mejorar en futuras versiones. Algunas pueden ser:

- Página para administrar centros: Como desarrollo prioritario, se debería implementar una vista desde donde un usuario administrador de centro pudiera cargar y editar la información de su centro. Esto ahorraría tiempo a los administradores de la aplicación ya que serían los propios centros los gestores de su propia información.
- Mejora del aspecto visual: Aunque con la librería de Vuetify se haya conseguido un aspecto moderno y limpio, la aplicación podría mejorar en algunas páginas como la de *Mis amigos* o *Creación de partida*.
- Envío de correos: De cara al futuro sería básico un módulo de envío de correos electrónicos. Se utilizaría principalmente en el registro de usuarios, como sistema de notificación de actividades y como posible vía de transmisión de información que los centros quieran comunicar a sus seguidores.

Bibliografía

1. Universidad de Cantabria. IEEE-STD-830-1998: ESPECIFICACIONES DE LOS REQUISITOS DEL SOFTWARE [Internet]. [citado 6 de abril de 2024]. Disponible en: https://www.ctr.unican.es/asignaturas/is1/ieee830_esp.pdf

2. JavaScript [Internet]. [citado 8 de junio de 2024]. Disponible en: https://developer.mozilla.org/es/docs/Web/JavaScript

3. ¿Qué es el DOM? [Internet]. ¿Qué es el DOM? [citado 8 de junio de 2024]. Disponible en: https://developer.mozilla.org/es/docs/Web/API/Document_Object_Model/Introduction

4. Material Design [Internet]. [citado 10 de junio de 2024]. Disponible en: https://m3.material.io/

5. Gabriel Fairman. ¿Qué es la internacionalización o i18n? [Internet]. ¿Qué es la internacionalización o i18n? [citado 10 de junio de 2024]. Disponible en: https://www.bureauworks.com/es/blog/que-es-internacionalizacion-o-i18n

6. Attila Fejér. Spring Boot annotations [Internet]. Spring Boot annotations. Disponible en: https://www.baeldung.com/spring-boot-annotations

7. ¿Qué es un ataque de diccionario? [Internet]. ¿Qué es un ataque de diccionario (si no se refiere a arrojar un libro a alguien)? 2019 [citado 11 de junio de 2024]. Disponible en: https://www.bbva.com/es/innovacion/que-es-un-ataque-de-diccionario-si-no-se-refiere-a-arrojar-un-libro-a-alguien/

8. Fernán García de Zúñiga. Todo sobre la arquitectura cliente-servidor [Internet]. Todo sobre la arquitectura cliente-servidor. 2024. Disponible en: https://www.arsys.es/blog/todo-sobre-la-arquitectura-cliente-servidor

9. Props [Internet]. Props. [citado 13 de junio de 2024]. Disponible en: https://vuejs.org/guide/components/props.html

10. Mixins [Internet]. Mixins. [citado 13 de junio de 2024]. Disponible en: https://vuejs.org/api/options-composition.html#mixins

11. Sam Millington. A Solid Guide to SOLID Principles [Internet]. A Solid Guide to SOLID Principles. [citado 13 de junio de 2024]. Disponible en: https://www.baeldung.com/solid-principles

12. CRUD [Internet]. CRUD. [citado 13 de junio de 2024]. Disponible en: https://developer.mozilla.org/es/docs/Glossary/CRUD

13. Atlassian. Kanban frente a scrum: ¿qué metodología ágil prefieres? [Internet]. Kanban frente a scrum: ¿qué metodología ágil prefieres? [citado 21 de junio de 2024]. Disponible en: https://www.atlassian.com/es/agile/kanban/kanban-vs-scrum

14. Atlassian. Scrumban: domina dos metodologías ágiles [Internet]. Scrumban: domina dos metodologías ágiles. [citado 18 de junio de 2024]. Disponible en: https://www.atlassian.com/es/agile/project-management/scrumban

15. CORS [Internet]. Errores CORS. [citado 13 de junio de 2024]. Disponible en: https://developer.mozilla.org/es/docs/Web/HTTP/CORS/Errors

ANEXO 1
Pruebas de aceptación

8.1 Sesión 1

Fecha: 19/06/2024

Usuario: Usuario 1

Responda en una escala de “Totalmente en desacuerdo” (1) a “Totalmente de acuerdo” (5):

	1	2	3	4	5
Facilidad de uso				x	
Las funcionalidades cumplen con mis expectativas				x	
La aplicación carga rápidamente y no presenta retrasos					x
El diseño de la aplicación es agradable				x	
Confío en que mis datos están seguros al registrarme		x			
Estoy satisfecho con mi experiencia general usando la aplicación				x	

Observaciones:

Al buscar un partido he aplicado el filtro de nivel, pero no me aparecían resultados. Si había ya algún resultado y aplicaba después el nivel nunca cambiaba.

En el proceso de registro he podido pasar al paso de los deportes sin haber puesto mi fecha de nacimiento.

En la página de partido no aparecen bien los usuarios. Sale uno menos.

8.2 Sesión 2

Fecha: 21/06/2024

Usuario: Usuario 1

Responda en una escala de “Totalmente en desacuerdo” (1) a “Totalmente de acuerdo” (5):

	1	2	3	4	5
Facilidad de uso				x	
Las funcionalidades cumplen con mis expectativas					x
La aplicación carga rápidamente y no presenta retrasos					x
El diseño de la aplicación es agradable				x	
Confío en que mis datos están seguros al registrarme		x			
Estoy satisfecho con mi experiencia general usando la aplicación				x	

Observaciones:

Algunas imágenes de los centros no se ven con buena resolución.

Cuando inicio sesión la página parece que no hace nada, pero enseguida carga. Me quedo sin saber si lo he hecho bien o si la página responde.

ANEXO 2

Objetivos de Desarrollo Sostenible

OBJETIVOS DE DESARROLLO SOSTENIBLE

Grado de relación del trabajo con los Objetivos de Desarrollo Sostenible (ODS).

Objetivos de Desarrollo Sostenibles	**Alto**	**Medio**	**Bajo**	**No Procede**
ODS 1. **Fin de la pobreza.**				**x**
ODS 2. **Hambre cero.**				**x**
ODS 3. **Salud y bienestar.**	**x**			
ODS 4. **Educación de calidad.**				**x**
ODS 5. **Igualdad de género.**				**x**
ODS 6. **Agua limpia y saneamiento.**				**x**
ODS 7. **Energía asequible y no contaminante.**				**x**
ODS 8. **Trabajo decente y crecimiento económico.**				**x**
ODS 9. **Industria, innovación e infraestructuras.**		**x**		
ODS 10. **Reducción de las desigualdades**.			**x**	
ODS 11. **Ciudades y comunidades sostenibles**.				**x**
ODS 12. **Producción y consumo responsables.**				**x**
ODS 13. **Acción por el clima.**				**x**
ODS 14. **Vida submarina.**				**x**
ODS 15. **Vida de ecosistemas terrestres.**				**x**
ODS 16. **Paz, justicia e instituciones sólidas.**				**x**
ODS 17. **Alianzas para lograr objetivos.**				**x**

Reflexión sobre la relación del TFG/TFM con los ODS y con el/los ODS más relacionados.

Tres de estos objetivos tienen relación con nuestra aplicación:

- **Salud y bienestar**: El deporte es un medio esencial para lograr mantener una vida sana y alcanzar el bienestar. Este proyecto promueve la práctica del deporte y fomenta el uso entre personas aumentando su práctica.
- **Industria, innovación e infraestructuras**: La aplicación hace uso de las últimas tecnologías y muestra como la utilización de estas puede emplearse en mejorar aspectos de la vida diaria. Además, mediante la inclusión de los centros en la aplicación se fomenta la industria deportiva al atraer a más usuarios, participantes y espectadores.
- **Reducción de las desigualdades**: La aplicación permite a cualquier persona de cualquier raza, edad y capacidad pueda organizar y participar en actividades. Acerca a personas independientemente de su procedencia fomentando la diversidad. Además, el uso de la aplicación es totalmente gratuito y está disponible para personas con limitaciones financieras.