UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería Informática

CVESPORTS: Plataforma para gestionar el servicio de deportes de ayuntamientos de la provincia de Valencia

Trabajo Fin de Grado

Grado en Ingeniería Informática

AUTOR/A: López Moine, Paul

Tutor/a: Albert Albiol, Manuela

Cotutor/a: Torres Bosch, María Victoria

CURSO ACADÉMICO: 2022/2023

Agradecimientos

Antes de empezar me gustaría agradecer a todas aquellas personas que han formado parte de este proyecto, tanto de manera directa como indirecta, que termina hoy con la presentación de este Trabajo de Fin de Grado.

En primer lugar, quiero agradecer a mis padres por haberme apoyado en todo momento. Gracias por vuestra paciencia, por ayudarme a seguir el camino correcto en mis momentos más difíciles, y, sobre todo, por hacerme ver que el que trabaja duro termina recibiendo su recompensa. Agradecer a mi hermana por hacerme creer en mis capacidades cuando no lo hacía y por ser mi apoyo.

En segundo lugar, quiero mencionar a mis amigos de la carrera y agradecerles por estar ahí cuando más nos necesitamos. Sin vosotros gran parte de este camino habría sido diferente y se habría hecho cuesta arriba. Durante estos meses nos hemos estado ayudando entre nosotros y eso se ha hecho ver tanto en nuestro desempeño profesional, como estudiantil y personal.

También quiero agradecer a mis tutoras Manuela y Victoria por guiarme y aconsejarme a lo largo de este camino. Gracias por darme ideas para ir más allá de lo que yo tenía planeado y aumentar mi interés en buscar más soluciones a implementar, y, sobre todo, a aprender. Vuestros consejos han sido un factor clave para terminar este proyecto.

Por último, quiero dar las gracias a mi equipo durante el periodo de prácticas en NTT Data Valencia. Gracias por haberme acogido y enseñado vuestra forma de trabajar, con esto habéis conseguido que me interese aún más en aprender tecnologías que antes ni habría contemplado mirar. Vuestros conocimientos, experiencia y valores han sido clave para mejorarme como profesional y no perder el rumbo tanto en el proyecto de la empresa como en este Trabajo de Fin de Grado.

En resumen, gracias a todos por formar parte de este proyecto y haber aportado lo mejor de vosotros para ayudarme en lo posible. Mi gratitud más profunda a cada uno de vosotros.

Resumen

El deporte es uno de los pilares fundamentales de la salud. La práctica de algunos deportes requiere de instalaciones específicas; normalmente la ciudadanía cuenta con entidades privadas o públicas que ofrecen dichas instalaciones. Las instalaciones municipales son un recurso muy utilizado y valorados por los practicantes de algunos de estos deportes, ya que habitualmente su uso suele ser gratuito o a un precio bajo. Sin embargo, no suele existir una forma sencilla de que los usuarios puedan gestionar las reservas de estas instalaciones. El principal objetivo de "CVESPORTS" es centralizar y agilizar el proceso de gestión de las infraestructuras deportivas públicas de los municipios de la provincia de Valencia. Para ello, se desarrollará una plataforma web que facilite a los usuarios la consulta y reserva de las instalaciones ofrecidas por cada municipio. El TFG pretende ofrecer a la ciudadanía una plataforma que mejore la experiencia de usuario en la reserva de instalaciones y que permita a los propios municipios una mejora en uno de los servicios públicos más demandado. La plataforma se podría expandir a más provincias, incluyendo los servicios que ofrezcan sus respectivas instituciones locales. El TFG se desarrollará utilizando TypeScript con Angular y CSS para front-end, y SpringBoot con Java para back-end. En cuanto a la base de datos se utilizará SQL en MySQL.

Palabras clave: Servicio de deportes; gestión de instalaciones; gestión de reservas; plataforma web; TypeScript; SpringBoot; experiencia de usuario.

Abstract

Sport is one of the fundamental pillars of health. The practice of some sports requires specific facilities; Normally, citizens have private or public entities that offer these facilities. Municipal facilities are a resource widely used and valued by practitioners of some of these sports, since their use is usually free or at a low price. However, there is usually no easy way for users to manage reservations for these facilities. The main objective of "CVESPORTS" is to centralize and streamline the management process of public sports infrastructures of the municipalities of the province of Valencia. To this end, a web platform will be developed to make it easier for users to consult and reserve the facilities offered by each municipality. The TFG aims to offer citizens a platform that improves the user experience in the reservation of facilities and that allows the municipalities themselves to improve one of the most demanded public services. The platform could be expanded to more provinces, including the services offered by their respective local institutions. The TFG will be developed using TypeScript with Angular and CSS for front-end, and SpringBoot with Java for back-end. As for the database, SQL Oracle will be used.

Keywords: sports service; facility management; Gestion of reservs; Web platform; typescript; SpringBoot; user experience.

Tabla de contenidos

Índice de figuras

1. Introducción

En este primer capítulo a modo de introducción se comentan las principales razones para llevar a cabo la idea propuesta. También se marcan los objetivos que la plataforma pretende abordar y el impacto que se espera que tenga sobre la sociedad. En último lugar se explica la estructura de la memoria, descomponiéndola por capítulos con sus respectivas descripciones.

1.1 Motivación

Hoy en día es prácticamente indispensable disponer de una plataforma que automatice los procesos de gestión de los servicios deportivos de entidades públicas [1]. Tradicionalmente, estas entidades permitían a los ciudadanos acceder a sus servicios a través de llamadas telefónicas o de forma presencial en las oficinas puestas a disposición para ello. Sin embargo, la utilización de plataformas web permiten que el cliente pueda gestionar sus reservas en cualquier momento y lugar, esto es sinónimo de comodidad. Viéndolo desde el punto de vista de los empleados que gestionan estos servicios deportivos, se reduce en gran medida la carga de trabajo, ya que se automatizan todos los procesos de gestión. También mejora la comunicación entre el usuario y la entidad pública y, para evitar conflictos de reserva, proporciona información de la disponibilidad de las instalaciones en tiempo real. Además, la incorporación de pago integrados directamente en la plataforma ofrece un nivel adicional de conveniencia y seguridad. Los usuarios pueden realizar y gestionar sus pagos de manera segura y sin complicaciones desde cualquier lugar y en cualquier momento [2].

Actualmente existen diferentes plataformas web de entidades públicas que permiten a los ciudadanos gestionar las reservas de instalaciones deportivas. Sin embargo, estas plataformas no son fácilmente accesibles y su diseño tampoco es intuitivo. Además, después de haber investigado no se ha hallado ninguna plataforma existente que ofrezca la capacidad de gestionar de manera eficiente y unificada las reservas de los servicios deportivos públicos en los municipios. Como usuario frecuente estos servicios, consideraba que la manera utilizada para reservar no era óptima.

Sería una gran ventaja disponer de una plataforma que permita compartir los servicios ofrecidos de cada municipio. Los usuarios se benefician poniéndose en forma y mejorando sus habilidades según el deporte, los administradores tienen una forma más sencilla de gestionar a dichos usuarios y el municipio dispone de una herramienta que centraliza toda la información.

De ahí surge la idea de desarrollar una plataforma ágil que mejore la experiencia de usuario a la hora de hacer uso de los servicios deportivos públicos. Para conseguirlo se ha optado por elaborar una aplicación web, que sea apta tanto para escritorio como para dispositivos móviles. Esta aplicación, a la que hemos llamado CVESPORTS,

supondría una mejora para agilizar los procesos comentados anteriormente y ofrecer una buena experiencia de usuario.

1.2 Objetivos

El principal objetivo del Trabajo Final de Grado (TFG) es desarrollar una plataforma web que permita gestionar las reservas de los servicios deportivos públicos de los municipios de la provincia de Valencia. Esta plataforma será accesible por los ciudadanos y por el administrador de la entidad pública mediante una página web con el nombre "CVESPORTS".

Dentro del objetivo principal se pueden definir los siguientes subobjetivos:

- Asegurar la confidencialidad de la información personal.
- Diseñar una interfaz intuitiva y fácil de usar, que ofrezca una buena experiencia de usuario.
- Disponer de un sistema de notificaciones.
- Contar con medidas de seguridad para proteger la plataforma web de ataques.

1.3 Impacto esperado

La plataforma web va a suponer una mejora en la calidad del flujo del proceso de gestión. Disponer de una herramienta que centralice todos los procesos de una reserva ofrece muchas ventajas tanto para el usuario que quiera hacer uso de los servicios deportivos públicos, como para los administradores que tienen más fácil diligenciar todos los trámites. Esta propuesta espera generar un impacto positivo en toda la provincia de Valencia y que en un futuro pueda expandirse hasta incluir municipios de toda la Comunitat Valenciana e incluso de otras comunidades autónomas.

1.4 Estructura de la memoria

Para la memoria se ha seguido la siguiente estructura:

- Capítulo 1 – Introducción: introduce y pone en contexto la temática del proyecto, describe los objetivos que se pretenden cumplir y el impacto a generar deseado.
- Capítulo 2 – Estado del arte: revisa las diferentes ofertas que existen en el mercado con un objetivo parecido o igual que "CVESPORTS" y compara las ventajas y desventajas de cada una.
- Capítulo 3 – Análisis del problema: se realiza un planteamiento previo para tener claro los agentes que van a interactuar, las acciones que van a llevar a cabo para interactuar con la web y cómo las van a realizar.

- Capítulo 4 – Diseño de la solución: se enfoca en la arquitectura utilizada para el proyecto.
- Capítulo 5 – Desarrollo de la solución: se comenta el proceso llevado a cabo durante el desarrollo para alcanzar el producto deseado, haciendo énfasis en los aspectos más complejos.
- Capítulo 6 – Pruebas: expone los resultados de las pruebas realizadas tanto a usuarios como técnicas. Además, muestra desde el punto de vista del cliente final cómo de buena es la plataforma.
- Capítulo 7 – Conclusiones: en este último capítulo se hace una valoración objetiva del proyecto, comentando los diferentes problemas y limitaciones que se han ido encontrando, y mejoras que se pueden aplicar en un futuro para perfeccionar la web.

2. Estado del arte

En este segundo capítulo se realiza una revisión de las aplicaciones más recientes cuyos objetivos son similares a los de CVESPORTS. Proporciona un contexto teórico e identifica tendencias actuales con la finalidad de justificar la necesidad de la investigación.

2.1 Contexto tecnológico

En la actualidad existen diversas herramientas con un propósito similar al de CVESPORTS, organizar y gestionar servicios deportivos. La principal diferencia entre ellas es el tipo de usuario al que van enfocados, mientras unas van dirigidas a personal de dirección de los servicios deportivos, otras se centran en los usuarios de estos servicios.

Además, en los últimos años este tipo de aplicaciones busca adentrarse en un ámbito más social, es decir, acercarlas a plataformas como pueden ser Instagram[1] o Facebook[2], incluyendo funcionalidades para compartir con otros usuarios información acerca del deporte y de los intereses de cada uno.

A continuación, se comentan algunas herramientas con funciones parecidas a las propuestas en CVESPORTS.

2.1.1. Playtomic

Playtomic [3] es una plataforma que gestiona las reservas de diferentes instalaciones (pistas de pádel, de tenis, de squash, de fútbol, etc.) de clubes deportivos privados. Permite a los usuarios organizar partidas para jugar tanto con tus amigos como con otra gente de la comunidad. El usuario también puede ver su historial de partidas y según sus resultados modificar su nivel de juego, ya que éste es el factor más importante a la hora de unirse a una partida. La aplicación también cuenta con un mecanismo estilo Instagram, es decir, los usuarios pueden seguir a otros usuarios según sus intereses comunes.

[1] Página web oficial de Instagram: https://www.instagram.com/
[2] Página web oficial de Facebook: https://es-es.facebook.com/
[3] Página web oficial de Playtomic: https://playtomic.io/

Figura 1. Interfaz de la aplicación Playtomic [3]

Observando la Figura 1, se puede apreciar cómo los usuarios se pueden apuntar a partidas abiertas creadas por otros y visualizar los resultados de los partidos organizados a través de la aplicación.

2.1.2. TPC Matchpoint

TPC Matchpoint [4] es un software de pago de terceros que gestiona los centros deportivos de manera integral. Los clubes pueden escoger qué tipo de funciones desean según el enfoque que tengan, por ejemplo, para un club de pádel o tenis está disponible la reserva de pistas, y para un gimnasio la gestión de las cuotas de los socios. Es muy conocida y utilizada por los clubes privados, sobre todo de pádel, ya que organiza de manera muy eficiente toda la información relacionada con jugadores, torneos, partidos y estadísticas.

Ofrece una personalización limitada, hay que ajustarse a las características que ofrece, por lo que no permite al usuario customizar la plataforma a su gusto. Como cualquier software, hay que integrarlo con el modelo de datos y los servicios de la plataforma, lo que puede llevar un tiempo familiarizarse con todas las características y funciones del software. Otro aspecto negativo es la dependencia tecnológica, y es que su uso, en caso de haber errores o interrupciones, puede afectar gravemente la disponibilidad y funcionalidad de la plataforma.

[4] Página web oficial de TPC Matchpoint: https://tpcmatchpoint.com/index.html

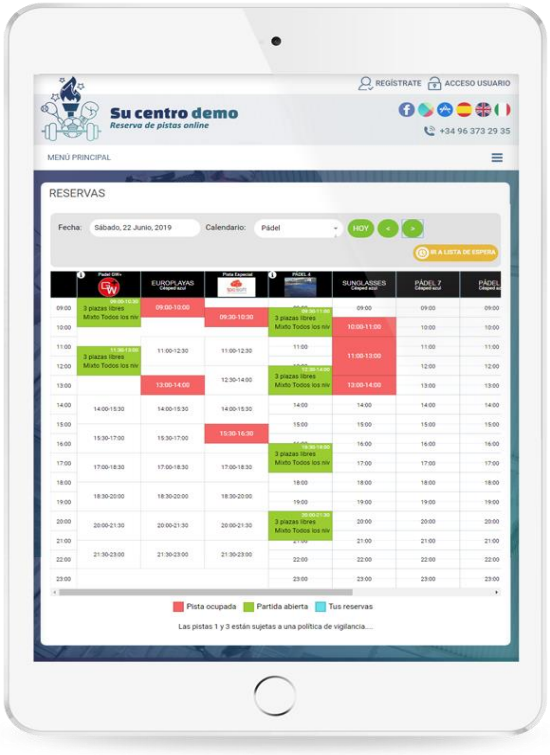

Figura 2. Interfaz de reserva de pistas de pádel en TPC Matchpoint [4]

En la Figura 2 se muestra un ejemplo de cómo el software gestiona y organiza las partidas de un club de pádel. Vemos que diferencia bien las partidas libres de las reservadas y de las abiertas. Para estas últimas indica el número de jugadores para completar la partida y su rango de nivel.

2.1.3. Playtomic Manager

Playtomic Manager [5] es una aplicación cuyo cometido es muy parecido al de Playtomic. Lo que las diferencia es su enfoque, y es que como su propio nombre indica, la versión Manager se centra en proporcionar las herramientas necesarias a los dueños y administradores de los clubes para gestionar, mantener y personalizar la interacción del usuario con la web del club.

La aplicación permite gestionar las clases ofrecidas por los monitores, facilitando a los jugadores inscribirse en ellas desde sus dispositivos móviles. Dispone de un modo liga, en el cual se puede publicar y gestionar de manera automatizada las ligas por fases, grupos y resultados. Organiza torneos de la forma más sencilla, gestionando la inscripción y todos los participantes, crea reservas recurrentes para clientes fidelizados, etc.

[5] Página web oficial de Playtomic Manager: https://playtomic.com/es/playtomic-manager/

Playtomic Manager ofrece una gran variedad de opciones a parte de las comentadas anteriormente, pero su complejidad hace que la curva de aprendizaje sea más elevada. Para los clubes que buscan un estudio más exhaustivo de su infraestructura, es una buena opción gracias a la especificación de información. Pero a diferencia de CVESPORTS, ésta busca la simplicidad y compactar la funcionalidad del usuario con la del administrador.

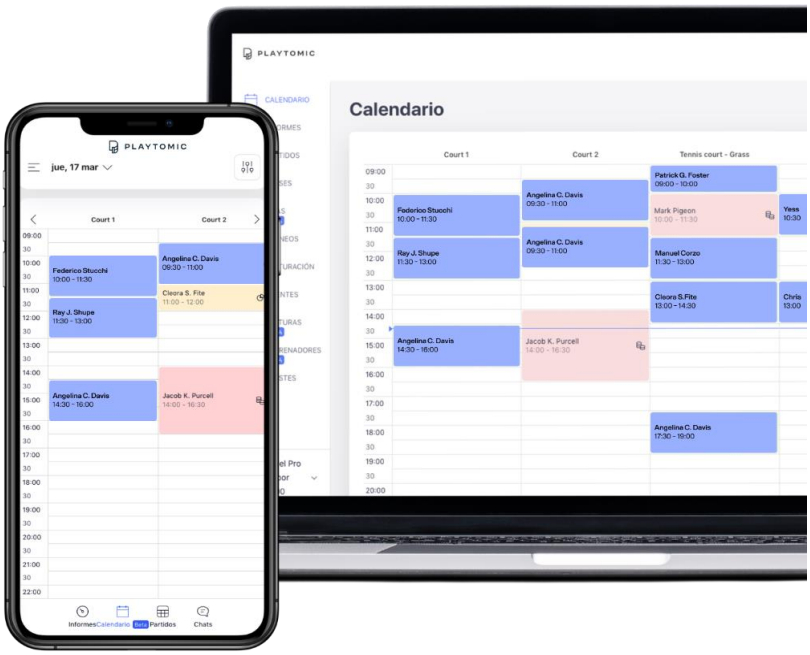

Figura 3. Interfaz de Playtomic Manager [5]

En la Figura 3 se muestra una captura de la organización de los eventos en el calendario de la aplicación.

Ambas plataformas ofrecen mecanismos de gestión y reserva bastante parecidos, aunque, como se ha dicho antes, CVESPORTS prima por facilitar al usuario el manejo de la plataforma, aunque se ha de tener en cuenta que Playtomic Manager y la propuesta apuntan hacia diferentes tipos de usuarios, por lo tanto, la complejidad de una u otra depende completamente de lo que éstos buscan.

2.2 Propuesta

Comparando CVESPORTS con las plataformas mencionadas anteriormente, se pueden identificar varios argumentos que respaldan el desarrollo del proyecto.

- En primer lugar, está pensado que CVESPORTS no esté compuesto por productos software de terceros ni que sea una extensión de una aplicación ya existente, como las ya mencionadas.

- Otro punto a favor es la personalización ya que los municipios pueden solicitar una adaptación del estilo o estructura de las interfaces para adecuarlas a sus intereses.
- CVESPORTS se centra únicamente en gestionar reservas, así como Playtomic y Playtomic Manager, pero la principal diferencia es que la propuesta combina y simplifica los objetivos de ambas aplicaciones. Sí que es cierto que éstas incluyen más funciones como ver estadísticas o la interacción de unos usuarios con otros, pero el cometido de CVESPORTS es permitir la gestión tanto a usuarios como administradores de unos servicios concretos en una única plataforma web.

- Por último, y referente a las distintas funcionalidades extra que ofrecen las aplicaciones comentadas anteriormente, lo que CVESPORTS busca cubrir son las necesidades básicas que requiere una plataforma de gestión de reservas deportivas. La idea es que todas estas funciones adicionales sean añadidas en un futuro para ofrecer más variedad de información y una mejor experiencia. Todo esto se comenta en el apartado de “Trabajo futuro” en el **Capítulo 7**.

3. Análisis del problema

En este tercer capítulo se identifican los problemas que la plataforma debe resolver. Involucra comprender las necesidades y requisitos del usuario, definir los objetivos y perfilar las funcionalidades y características que se deben incluir para abordar el problema.

3.1 Especificación de requisitos

En este apartado se introducen los requisitos que se desean cubrir, acompañados de sus casos de uso.

3.1.1 Requisitos funcionales

Los requisitos funcionales son las declaraciones de cómo debe funcionar el sistema.

La aplicación tiene los siguientes requisitos:

1. **Cuenta de usuario**
   - CU1. Iniciar sesión
   - CU2. Registrarse
   - CU3. Modificar cuenta
   - CU4. Eliminar cuenta
   - CU5. Cerrar sesión

2. **Reserva**
   - CU6. Ver servicios ofrecidos
   - CU7. Realizar una reserva
   - CU8. Cancelar una reserva
   - CU9. Pagar una reserva

3. **Comunicación**
   - CU10. Enviar notificaciones

4. **Administrador**
   - CU11. Eliminar reserva
   - CU12. Añadir servicio
   - CU13. Ver usuarios registrados
   - CU14. Modificar datos del municipio
   - CU15. Eliminar usuario
   - CU16. Ver detalles de una reserva

A continuación, se muestra el diagrama de casos de uso (Figura 4) representado los requisitos comentados anteriormente, con el objetivo de obtener una representación más visual de los diferentes actores y sus respectivas funciones.

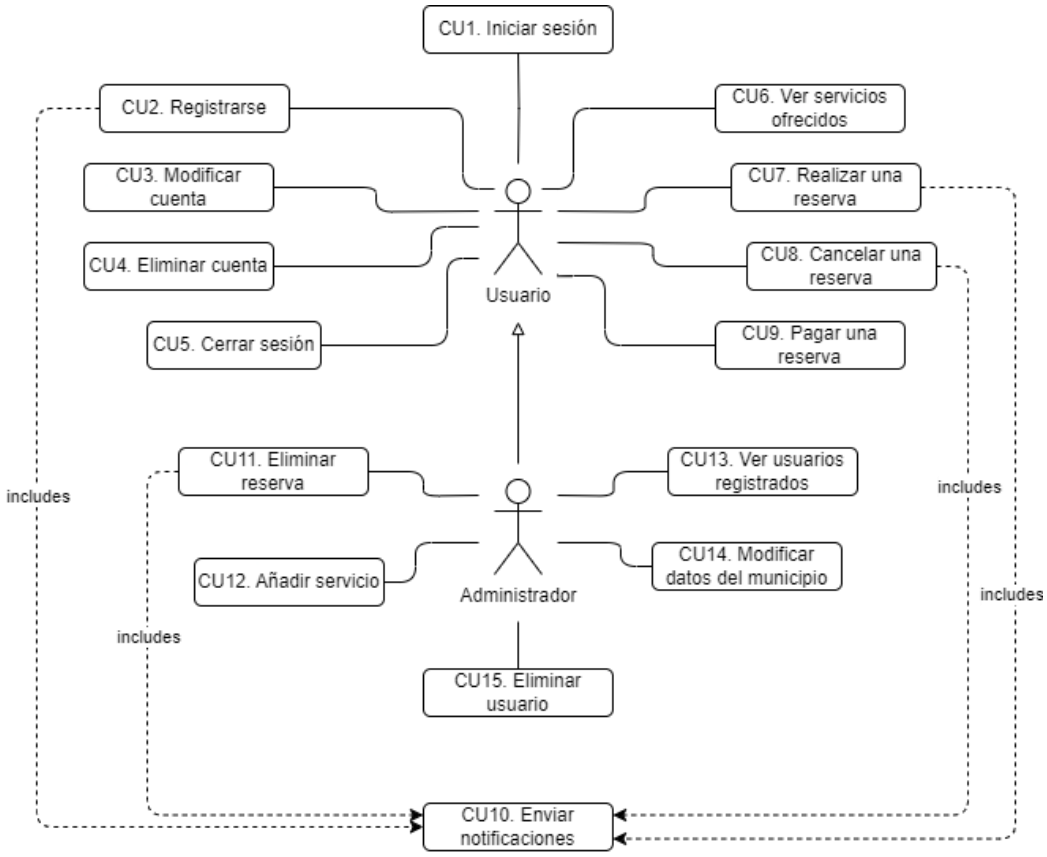

Figura 4. Diagrama de casos de uso

La descripción de cada caso de uso es la siguiente:

Nombre	CU1. Iniciar sesión
Actores involucrados	Usuario y Administrador.
Objetivo	Iniciar sesión en la plataforma.
Precondiciones	El usuario debe tener una cuenta registrada en la plataforma.
Escenario principal	1. El usuario selecciona la opción “Iniciar Sesión” en el menú superior. 2. Introduce el email y contraseña de su cuenta y hace click en “Iniciar sesión.”

	3. La plataforma comprueba que los datos corresponden correctamente con la cuenta.
Postcondiciones	Redirección a la página del perfil de usuario y se muestra en el menú la sección “Mis Reservas”.
Rutas alternativas	3a. En caso de que el email no exista o no está escrito correctamente, se muestra un mensaje de error en la parte inferior del campo de texto del email indicando que el email introducido no es correcto o no existe. 3b. En caso de que la contraseña no sea correcta se muestra un mensaje de error en la parte inferior del campo de texto de la contraseña indicando que la contraseña introducida no es correcta.

Nombre	CU2. Registrarse
Actores involucrados	Usuario.
Objetivo	Registrarse en la plataforma.
Precondiciones	El usuario no tiene que estar ya registrado.
Escenario principal	1. El usuario selecciona la opción “Iniciar Sesión” en el menú superior. 2. Selecciona la opción de registrarse. 3. Introduce los datos personales que le pide el sistema. 4. Completa el registro. 5. Recibe un correo confirmando el registro.
Postcondiciones	+ Redirección a la página del perfil de usuario y se muestra en el menú la sección “Mis Reservas”. + El sistema enviará un correo al usuario confirmando el alta en la plataforma.
Rutas alternativas	3a. En caso de que el email ya esté registrado o no esté escrito correctamente, se muestra un mensaje de error en la parte inferior del campo de texto del email indicando que el email introducido no es correcto o no existe. 3b. En caso de que los datos introducidos en los campos no sean correctos, se muestra un mensaje de error en la parte inferior de los respectivos campos.

Nombre	CU3. Modificar cuenta
Actores involucrados	Usuario.
Objetivo	Realizar cambios en una cuenta.
Precondiciones	El usuario debe haber iniciado sesión.
Escenario principal	1. El usuario inicia sesión. 2. Entra en la sección “Mi Cuenta” 3. Modifica la información que desee y pulsa “Confirmar cambios”. 4. El sistema actualiza la información y redirige al usuario a la sección “Mi cuenta”.
Postcondiciones	La cuenta del usuario ha sido modificada.
Rutas alternativas	4a. En caso de que el usuario introduzca información incorrecta, el sistema mostrará un aviso.

Nombre	CU4. Eliminar cuenta
Actores involucrados	Usuario.
Objetivo	Eliminar una cuenta de la plataforma.
Precondiciones	El usuario debe haber iniciado sesión.
Escenario principal	1. El usuario inicia sesión. 2. Entra la sección “Mi Cuenta”. 3. Pulsa el botón “Eliminar cuenta”. 4. La aplicación preguntará si se desea eliminar la cuenta del sistema. 5. Una vez eliminada la cuenta, el usuario será redirigido a la página principal.
Postcondiciones	La cuenta del usuario ha sido eliminada del sistema y las reservas a su nombre también serán eliminadas.
Rutas alternativas	Ø

Nombre	CU5. Cerrar sesión
Actores involucrados	Usuario y Administrador.
Objetivo	Cerrar sesión en la plataforma.
Precondiciones	El usuario debe haber iniciado sesión.
Escenario principal	1. El usuario inicia sesión en la aplicación. 2. Pulsa el botón “Cerrar sesión”. 3. El sistema pregunta si se desea cerrar sesión. 4. El usuario es redirigido a la página principal.
Postcondiciones	El usuario ha cerrado sesión de la aplicación.
Rutas alternativas	Ø

Nombre	CU6. Ver servicios ofrecidos
Actores involucrados	Usuario y administrador.
Objetivo	Ver los servicios disponibles para reservar.
Precondiciones	Ø
Escenario principal	1. El usuario selecciona la opción “Municipios” del menú superior y busca el municipio del cual quiere consultar sus servicios. 2. El usuario accede a la página específica del municipio y puede consultar tanto información específica del municipio como de los servicios que ofrece.
Postcondiciones	El usuario ha visto los servicios ofrecidos por el municipio.
Rutas alternativas	Si no existe el municipio que se busca, el usuario no obtendrá resultado en la búsqueda.

Nombre	CU7. Realizar una reserva
Actores involucrados	Usuario y administrador.
Objetivo	Realizar una reserva de un servicio.
Precondiciones	El usuario debe haber iniciado sesión.
Escenario principal	1. El usuario inicia sesión en la aplicación. 2. El usuario selecciona la opción “Municipios” del menú superior y busca el municipio del cual quiere consultar sus servicios. 3. El usuario accede a la página específica del municipio. 4. Selecciona uno de los servicios ofrecidos (por ejemplo, Pista 2 de pádel de 17:30-19:00). 5. El sistema preguntará si se desea confirmar la reserva. 6. La reserva se ha completado y se puede consultar en “Mis reservas”.
Postcondiciones	+ El usuario realiza la reserva y ésta pasa a estar en estado “No pagada”. + El sistema enviará un correo al usuario confirmando la realización de la reserva. + La reserva pasará a estar en estado “Reservada”.
Rutas alternativas	4a. En caso de estar ya reservado, el sistema no dará la opción al usuario de seleccionar ese servicio para reservar.

Nombre	CU8. Cancelar una reserva
Actores involucrados	Usuario.
Objetivo	Cancelar una reserva realizada.
Precondiciones	El usuario debe haber iniciado sesión y realizado una reserva.
Escenario principal	1. El usuario inicia sesión en la aplicación. 2. El usuario accede a la página de “Mis Reservas”. 3. En la lista de reservas realizadas, selecciona una y pulsa el botón “Cancelar reserva”. 4. El sistema preguntará si se desea confirmar la cancelación de la reserva.

Postcondiciones	+ La reserva ya no estará a nombre del usuario. + La reserva estará disponible para otros usuarios. + El sistema enviará un correo al usuario confirmando la cancelación de la reserva. + La reserva pasará a estar en estado “Libre”.
Rutas alternativas	Ø

Nombre	CU9. Pagar una reserva
Actores involucrados	Usuario.
Objetivo	Realizar el pago de una reserva confirmada.
Precondiciones	El usuario debe haber iniciado sesión y realizado una reserva.
Escenario principal	1. El usuario inicia sesión en la aplicación. 2. El usuario accede a la sección “Mi cuenta”. 3. Si dispone de alguna reserva por pagar, selecciona la opción “Pagar reserva”. 4. Se abre una ventana modal que da 3 opciones de método de pago. 5. El usuario elige su método de pago y sigue las instrucciones indicadas.
Postcondiciones	El pago de la reserva se habrá realizado.
Rutas alternativas	Ø

Nombre	CU10. Enviar notificaciones
Actores involucrados	Sistema.
Objetivo	Enviar notificaciones a los usuarios.
Precondiciones	El usuario debe tener una dirección de correo registrada en la aplicación.

Escenario principal	1. El sistema detecta que es necesario enviar una notificación vía correo electrónico al usuario.
Postcondiciones	El usuario recibirá una notificación vía email.
Rutas alternativas	1a. Si la dirección de correo no está registrada en el sistema, no se enviará ninguna notificación.

Nombre	CU11. Eliminar reserva
Actores involucrados	Administrador.
Objetivo	El administrador elimina una reserva existente.
Precondiciones	+ El administrador debe haber iniciado sesión. + Debe haber una reserva realizada.
Escenario principal	1. El administrador inicia sesión en la aplicación. 2. Entra en la sección “Admin” del menú superior. 3. Selecciona la pestaña de “Reservas”. 4. Pulsa el botón “Eliminar Reserva” de la reserva que se desea eliminar. 5. El sistema preguntará si se desea eliminar la reserva.
Postcondiciones	+ El administrador elimina una reserva realizada por un usuario. + El sistema enviará un correo al usuario confirmando la eliminación de su reserva.
Rutas alternativas	Si no se ha realizado una reserva, el administrador no tendrá disponible la opción de “Eliminar” para ese preciso servicio.

Nombre	CU12. Añadir servicio
Actores involucrados	Administrador.
Objetivo	El administrador crea un nuevo servicio para su municipio.
Precondiciones	+ El administrador debe haber iniciado sesión.
Escenario principal	1. El administrador inicia sesión en la aplicación. 2. Entra en la sección “Admin” del menú superior. 3. Selecciona la pestaña de “Crear servicio”. 4. Rellena el formulario y pulsa el botón “Crear nuevo servicio”.
Postcondiciones	El sistema añade un nuevo servicio al municipio.
Rutas alternativas	Ø

Nombre	CU13. Ver usuarios registrados
Actores involucrados	Administrador.
Objetivo	El administrador consulta la lista de los usuarios dados de alta en la plataforma.
Precondiciones	+ El administrador debe haber iniciado sesión.
Escenario principal	1. El administrador inicia sesión en la aplicación. 2. Entra en la sección “Admin” del menú superior. 3. Selecciona la pestaña de “Usuarios”.
Postcondiciones	El sistema muestra al administrador la lista con todos los usuarios.
Rutas alternativas	Ø

Nombre	CU14. Modificar datos del municipio
Actores involucrados	Administrador.
Objetivo	El administrador modifica la información del municipio que administra.
Precondiciones	+ El administrador debe haber iniciado sesión. + Debe haber una reserva realizada.
Escenario principal	1. El administrador inicia sesión en la aplicación. 2. Entra en la sección “Admin” del menú superior. 3. Modifica los datos en el formulario que hay disponible en el dashboard del administrador. 4. Pulsa el botón de “Modificar”.
Postcondiciones	El sistema modifica la información del municipio.
Rutas alternativas	4a. El sistema informa en caso de que algún campo del formulario no cumpla con los requisitos.

Nombre	CU15. Eliminar usuario
Actores involucrados	Administrador.
Objetivo	El administrador elimina un usuario del sistema.
Precondiciones	+ El administrador debe haber iniciado sesión.
Escenario principal	1. El administrador inicia sesión en la aplicación. 2. Entra en la sección “Admin” del menú superior. 3. Selecciona la pestaña de “Usuarios”. 4. Pulsa el botón “Eliminar usuario” del usuario que se quiera eliminar. 5. El sistema preguntará si se desea eliminar el usuario.
Postcondiciones	La cuenta del usuario ha sido eliminada del sistema y las reservas a su nombre también serán eliminadas.
Rutas alternativas	Ø

3.1.2 Requisitos no funcionales

Los requisitos no funcionales definen cómo debería ser un sistema en lugar de qué debería hacer. Explican los aspectos de calidad del software y, de manera contraria a los requisitos funcionales, se incorporan de manera incremental.

Los requisitos no funcionales de CVESPORTS son los siguientes:

Seguridad: la web debe estar protegida contra el acceso no autorizado. Para asegurar que esté protegida, se implementa una robusta estrategia de seguridad basada en tres pilares fundamentales: integridad, autenticación y autorización. Existen dos roles, usuario y administrador. El usuario tiene acceso a todas las vistas, excepto la del administrador, la cual solamente se puede acceder iniciando sesión con la clave del administrador.

- Integridad: garantiza que la información no sea alterada de forma no autorizada.
- Autenticación: permite comprobar que los datos del usuario corresponden con su rol, otorgando así el acceso correspondiente al sistema.
- Autorización: asignar los diferentes tipos de acceso según el rol del usuario.

Rendimiento: la web debe permitir que un número determinado de usuarios puedan interactuar con ésta sin que haya ninguna bajada de rendimiento. Este objetivo se sostiene en dos pilares clave: eficiencia y tiempo de respuesta.

- Eficiencia: debe realizar cada una de las funciones con el menor consumo y tiempo posible. Esto no solo significa que las operaciones se realicen rápidamente, sino que también se utilicen de manera eficiente los recursos disponibles.
- Tiempo de respuesta: debe ser rápido, no sobrepasando el umbral de 1 segundo. Un tiempo de respuesta rápido asegura una experiencia de usuario fluida.

Escalabilidad: la web debe ser capaz de escalar hacia arriba o hacia abajo según sea necesario, para adaptarse a los cambios en la demanda

- Capacidad de carga: debe poder gestionar una capacidad de peticiones alta sin reducir drásticamente el rendimiento. Se espera que la aplicación web sea utilizada por un promedio de 40 personas en un día normal y 150 en una hora pico.

Mantenibilidad: la web debe ser fácil de mantener y actualizar.

- Legibilidad del código: debe tener un código claro, que sea entendible.
- Gestión de errores: debe tener un sistema de detección de fallos para gestionar los posibles errores o excepciones que puedan aparecer durante la ejecución.

- Flexibilidad: debe ser adaptable a cambios que se realicen en el futuro.
- Modularidad: debe estar dividido en diferentes componentes para facilitar la modificación de uno específico.

Fiabilidad: la web debe ser confiable, proporcionando un servicio constante y cumpliendo con los requisitos y expectativas del usuario de manera continua.

- Tolerancia a fallos: debe poder funcionar sin errores ni interrupciones.
- Disponibilidad: debe estar disponible cuando sea necesario.

Usabilidad: la web debe ser intuitiva y ha de permitir que los usuarios nuevos puedan llegar a todas las secciones de manera directa.

- Facilidad de aprendizaje: debe ser fácil de aprender para los usuarios menos experimentados.
- Facilidad de memorizar: debe ser fácil de memorizar a la hora de llevar a cabo una función.
- Retroalimentación: debe mantener informado al usuario de lo que hace cuando interactúa con el sistema.
- Atractivo visual: debe contar un diseño bien estructurado e intuitivo para que los usuarios interactúen de manera más fácil.

Interoperabilidad: la web debe ser compatible con otros sistemas, permitiendo la interacción y el intercambio de datos de manera eficiente y efectiva. Esto implica ajustarse a los estándares y protocolos comunes, y ser capaz de integrarse con una variedad de sistemas y tecnologías software.

3.2 Plan de trabajo

Para el desarrollo del proyecto se han aplicado algunas prácticas derivadas de la metodología ágil SCRUM. SCRUM es un marco de trabajo de gestión de proyectos ágil que ayuda a los equipos a estructurar y gestionar su trabajo a través de un conjunto de principios y prácticas [6]. Esta metodología propone una serie de roles de entre los cuales en el presente proyecto se han utilizado los siguientes:

- Desarrollador: únicamente el proyectando.
- Product Owner: aunque no hay estrictamente ninguna persona que ejerza este rol, sí que tenemos al propio desarrollador, que abarca la toma de decisiones a la hora de priorizar tareas, cómo abordarlas, cuáles se consideran MVP (Minimum Viable Product) y su testeo o revisión, y las tutoras del proyecto. Después de cada sprint se ha realizado una revisión donde se le muestra el estado del proyecto, se analizan posibles mejoras y reorganiza el sprint posterior

con las tareas más prioritarias y aquellas que no ha sido posible realizar en el sprint correspondiente.

Como propone SCRUM, la carga de trabajo ha sido almacenada en el **backlog** (pila de trabajo) y se ha ido organizando a medida que se realizaban las tareas, siguiendo así un orden de prioridad. Ha servido mucho para posponer tareas más complejas que no han podido ser terminadas, y para identificar posibles bloqueos en unidades de trabajo más prioritarias que opacaban el desarrollo de otras con menos prioridad.

Según la metodología ágil, hay que poblar el backlog de las tareas, las cuales están agrupadas en categorías. Cada una de estas categorías trata una temática diferente, y supone grandes beneficios en los siguientes aspectos:

- Organización: ayuda a comprender mejor el backlog, aportando mayor visibilidad agrupando las tareas por categorías.
- Priorización: permite priorizar las tareas dentro de cada categoría, que facilita la planificación de cada sprint.
- Enfoque: es útil a la hora de centrarse en una tarea o componente específico.

A continuación, se definen las categorías y unidades de trabajo del backlog:

Categoría	**Unidades de trabajo (UTs)**
Listado de partidas	+ Mostrar en formato de tabla todas las partidas que ofrece un municipio + Consumir y mapear lista desde la API
Listado de usuarios	+ Mostrar en formato de tabla todos los usuarios dados de alta en el sistema + Consumir y mapear lista desde la API
Perfil de usuario	+ Modificar información de perfil + Vista del perfil del usuario + Eliminar cuenta del sistema + Cerrar sesión
Registro y autenticación	+ Iniciar sesión + Registrarse en la plataforma + Recuperar contraseña
Notificaciones	+ Notificar vía correo electrónico de la confirmación de una reserva + Notificar vía correo electrónico de la cancelación de una reserva + Notificar vía correo electrónico de la creación de una cuenta.

Reservas	+ Completar una reserva + Cancelar una reserva + Visualizar las reservas de partidas de un municipio en específico + Visualizar las reservas de un usuario + Visualizar historial de reservas + Integrar servicios de pago
Gestión de reservas	+ Eliminar una reserva + Ver detalles de una reserva
Gestión de municipios	+ Insertar/modificar datos de un municipio + Crear nuevos servicios para un municipio
Gestión de usuarios	+ Eliminar un usuario del sistema
Navegación	+ Navegación de tipo jerárquica + Mantener las propiedades y contextos entre vistas

Siguiendo la metodología SCRUM el proyecto se ha dividido en **6 sprints** con una duración de **dos semanas** cada uno. En ellos se ha realizado tanto el análisis previo, como el desarrollo y la realización de la memoria. Se ha empezado por la parte de análisis, en la que se investiga acerca de la temática escogida y qué tecnologías utilizar. Una vez terminado este proceso se pasa al desarrollo y documentación. Para tener la información lo más actualizada posible lo que se ha hecho ha sido redactar a medida que se iban realizando tareas, así se van apuntando los cambios que van a afectar en un futuro al desarrollo y se tiene una información más clara en el apartado de desarrollo.

A continuación, se describe lo realizado en cada sprint, indicando fecha de inicio y de fin:

Sprint 0 (10/04/2023 – 24/04/2023)

El primer sprint se centra en la investigación del contexto del proyecto y las tecnologías a utilizar. Es de los más importantes ya que define el alcance del proyecto e introduce las bases de este. También se estableció el primer contacto con el tutor para acordar la idea del proyecto y los objetivos a abordar. En cuanto a tareas llevadas a cabo se encuentran la creación del esquema de la arquitectura con las tecnologías a utilizar, el esquema de la base de datos y de las vistas que tendrá la aplicación en el punto de partida.

Sprint 1 (24/04/2023 – 08/05/2023)

En el segundo sprint se creó la primera versión de las interfaces, así como la estructura del back-end, organizando el proyecto en los paquetes correspondientes para posteriormente implementar su funcionalidad. También se trabajó en la navegación de la aplicación, y es que uno de los retos principales de este sprint fue que cada una de las vistas estuviera bien conectada para evitar que el usuario sea redirigido a un recurso que no existe.

Sprint 2 (08/05/2023 – 22/05/2023)

En el tercer sprint se empezaron a crear los primeros controladores de la API para que el front-end de la aplicación pudiese consumirla y trabajar con la información de la base de datos. Se implementó la visualización tanto de los servicios ofrecidos por los municipios como la lista de reservas de un usuario. También se implementó la eliminación de cuenta del usuario en el sistema y la modificación de sus datos personales.

Sprint 3 (22/05/2023 – 05/06/2023)

En el cuarto sprint se arreglaron los bugs de las funcionalidades implementadas en el sprint anterior. También se implementaron las funcionalidades de reservar, cancelar reservas, proceder al pago y gestionar las reservas de un usuario. Por último, se implementó el sistema de notificaciones vía correo electrónico para notificar la confirmación o cancelación de las reservas.

Sprint 4 (05/06/2023 – 19/06/2023)

El sprint comenzó con la finalización de las vistas, agregándoles estilos y acciones a sus controles. Se trabajó en la implementación de la seguridad de la aplicación web, implementando un servicio de autenticación mediante tokens. También se realizaron pruebas unitarias y de integración para asegurar el correcto funcionamiento de los controladores.

Sprint 5 (19/06/2023 – Presente)

En este último sprint (hasta una futura continuación del proyecto) se añadió la vista del administrador junto a sus funcionalidades. También se incluyeron varias mejoras en la funcionalidad de la aplicación web y se realizaron más pruebas unitarias y de integración para comprobar que las modificaciones funcionen correctamente. Por último, se realizaron pruebas de usabilidad para tener feedback de usuarios reales y tomar nota de posibles mejoras a tener en cuenta para el futuro.

4. Diseño de la solución

La fase de diseño en el desarrollo de software es una etapa esencial en la que se define y organiza la estrategia para resolver el problema. En este capítulo se detalla la arquitectura del sistema. En particular, se aborda el enfoque adoptado para estructurar el software, incluyendo las elecciones tecnológicas.

4.1 Arquitectura

Para el desarrollo de CVESPORTS se opta por una arquitectura MVC (Modelo-Vista-Controlador) debido a varios requisitos no funcionales clave: seguridad, rendimiento, escalabilidad, mantenibilidad, fiabilidad e interoperabilidad [7].

La arquitectura MVC consiste en separar la lógica de negocio, la presentación y la entrada del usuario en tres componentes, lo que aporta varios beneficios significativos. Por un lado, mejora la seguridad, ya que separa la vista del usuario de la lógica de negocio, lo que dificulta la manipulación no autorizada de los datos. Mejora el rendimiento al permitir que diferentes componentes se procesen de manera independiente. Esto hace que la aplicación maneje una mayor cantidad de usuarios sin experimentar una bajada de rendimiento. También aumenta la escalabilidad, ya que permite que los componentes se modifiquen sin afectar a los demás.

Por otro lado, la arquitectura favorece la mantenibilidad y reutilización del código, lo que simplifica las tareas de actualización y mejora del software a lo largo del tiempo [8]. También proporciona una mayor fiabilidad al permitir la realización de pruebas unitarias, lo que mejora la calidad del código. Además, esta arquitectura promueve la interoperabilidad, ya que facilita la integración de la aplicación con otras tecnologías.

A continuación, en la Figura 5 se muestra el esquema de la arquitectura MVC:

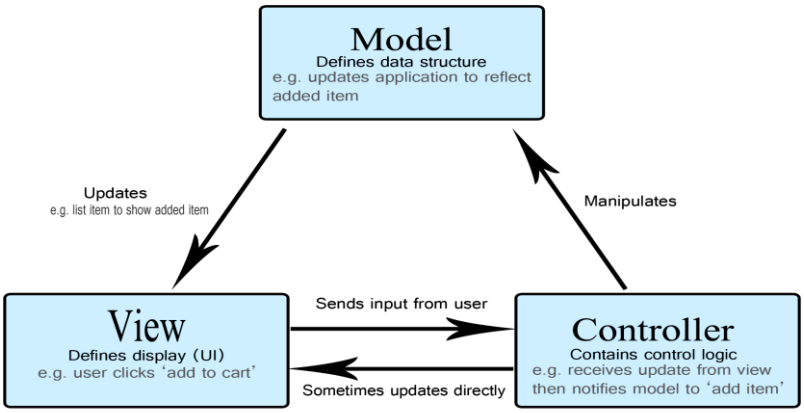

Figura 5. Arquitectura de la aplicación [9]

4.1.1 Capa de Modelo

El modelo representa la capa de la aplicación que implementa la lógica de negocio, lo que significa que es responsable de manipular la información proveniente de una base de datos, una API o un objeto JSON, convirtiéndola en conceptos significativos para la aplicación, procesarla, entre otras tareas relativas a la manipulación de datos. [10]

Los objetos del modelo (entidades) representan los principales conceptos del dominio del sistema. Permiten el almacenamiento, manipulación e intercambio de datos entre las capas de la aplicación.

La Figura 6 muestra el diagrama de clases con el tipo de objetos que se utiliza en la aplicación:

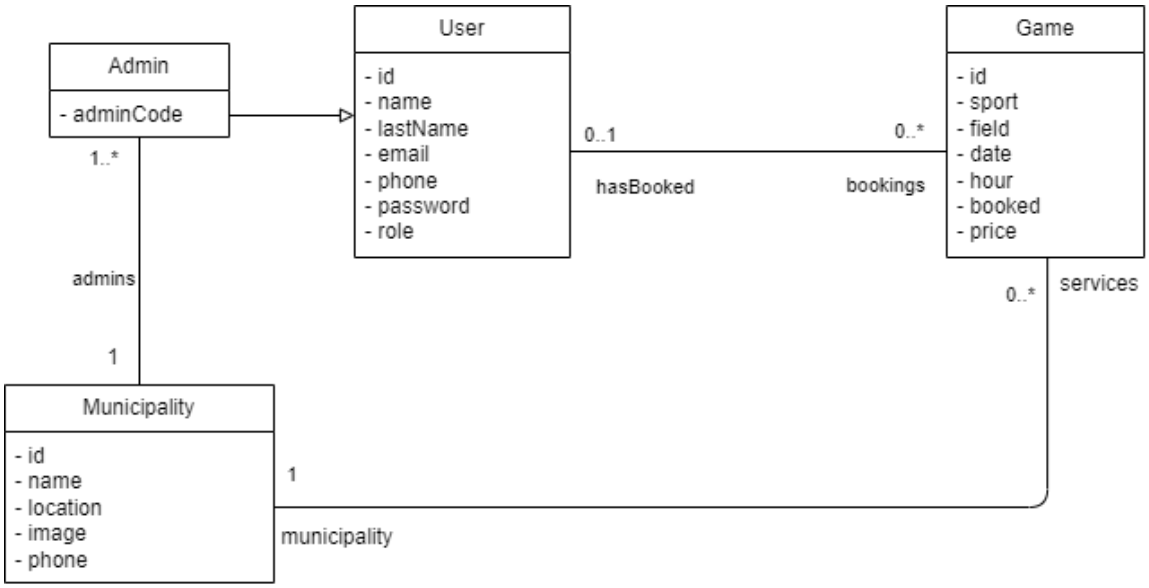

Figura 6. Diagrama de clases de la aplicación

El diagrama de clases representa una relación entre Usuario y Game de 0 a muchos. Esto es porque un usuario puede tener más de una reserva (un usuario tiene una lista de reservas). La relación entre Municipality y Game es también es de 0 a muchos. Un municipio puede ofrecer una lista de partidas. En cuanto a la relación Municipality y Administrador, es de 1 a muchos porque un administrador solo puede moderar un solo municipio.

La Figura 7 muestra el diagrama Entidad-Relación de la base de datos. Nótese que las clases que tienen una relación tienen un atributo del mismo tipo que la clave primaria de la tabla destino.

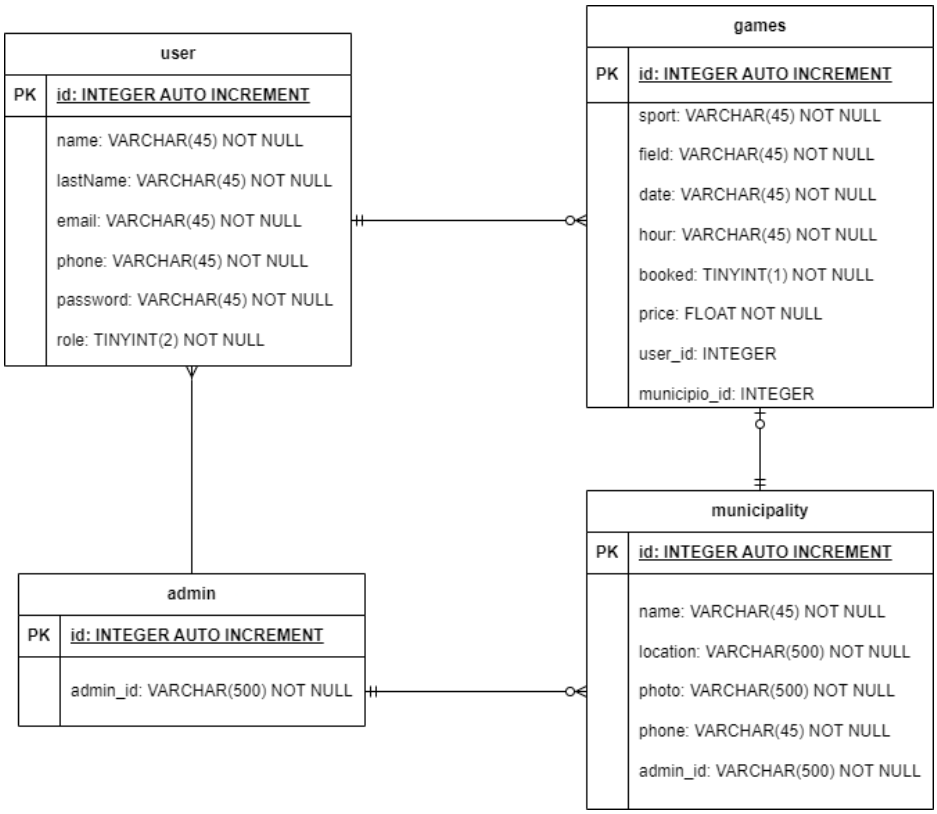

Figura 7. Diagrama Entidad Relación de la aplicación

4.1.2 Capa de Vista

La capa de vista se compone de los elementos con los que interactúa el usuario, es decir, de la interfaz. Se encarga de representar visualmente la aplicación web y actualizarla según los datos que se solicitan al controlador, con el cual se comunica a través de eventos. La parte visual captura eventos como puede ser el click de un botón o dejar el cursor encima de un elemento concreto. Estos eventos disparan funciones que están declaradas en el controlador y se comunican con el modelo, y dependiendo del tipo de petición recogerán o actualizarán datos.

En el desarrollo de CVESPORTS, los mock-ups juegan un papel crucial en esta capa de vista. Los mock-ups son prototipos que permiten obtener una visión previa de cómo se verá la aplicación. Ayudan a visualizar la distribución de los elementos de la interfaz, la interacción del usuario y el flujo de la aplicación. Fueron esenciales en la etapa de diseño para garantizar que la interfaz sea intuitiva y fácil de usar, tanto para los usuarios que buscan municipios y servicios como para los administradores que gestionan la información.

A continuación, se muestran algunos de los mock-ups que se utilizaron en esta fase de diseño:

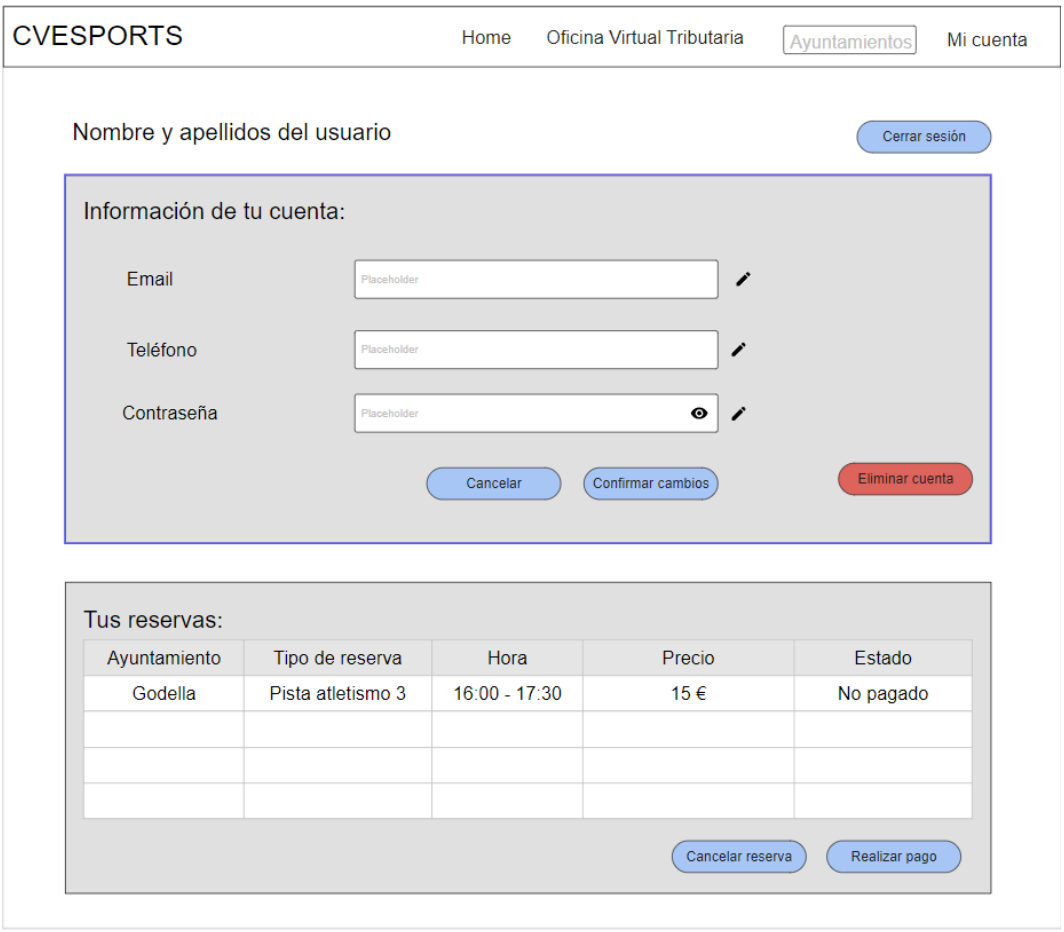

Figura 8. Mock-up del perfil de usuario

En la Figura 8 se muestra la vista de la cuenta del usuario, donde puede modificar información o eliminar la propia cuenta. La interfaz permite cambiar el correo electrónico, el teléfono y la contraseña. Para acceder a esta vista primero hay que iniciar sesión cuando se selecciona “Mi cuenta” en el menú superior. Una vez dentro, se permite al usuario elegir entre iniciar sesión o registrarse, y una vez cumplido uno de los dos casos de uso nombrados previamente, ya se accede a la vista de “Modificar cuenta”, donde se pueden visualizar las reservas que fueron realizadas a nombre de la cuenta, cancelarlas, proceder al pago, cerrar la sesión y eliminar la cuenta del sistema. Para ello se selecciona cualquier fila de la tabla y según el botón apretado redirige al usuario a una vista u otra.

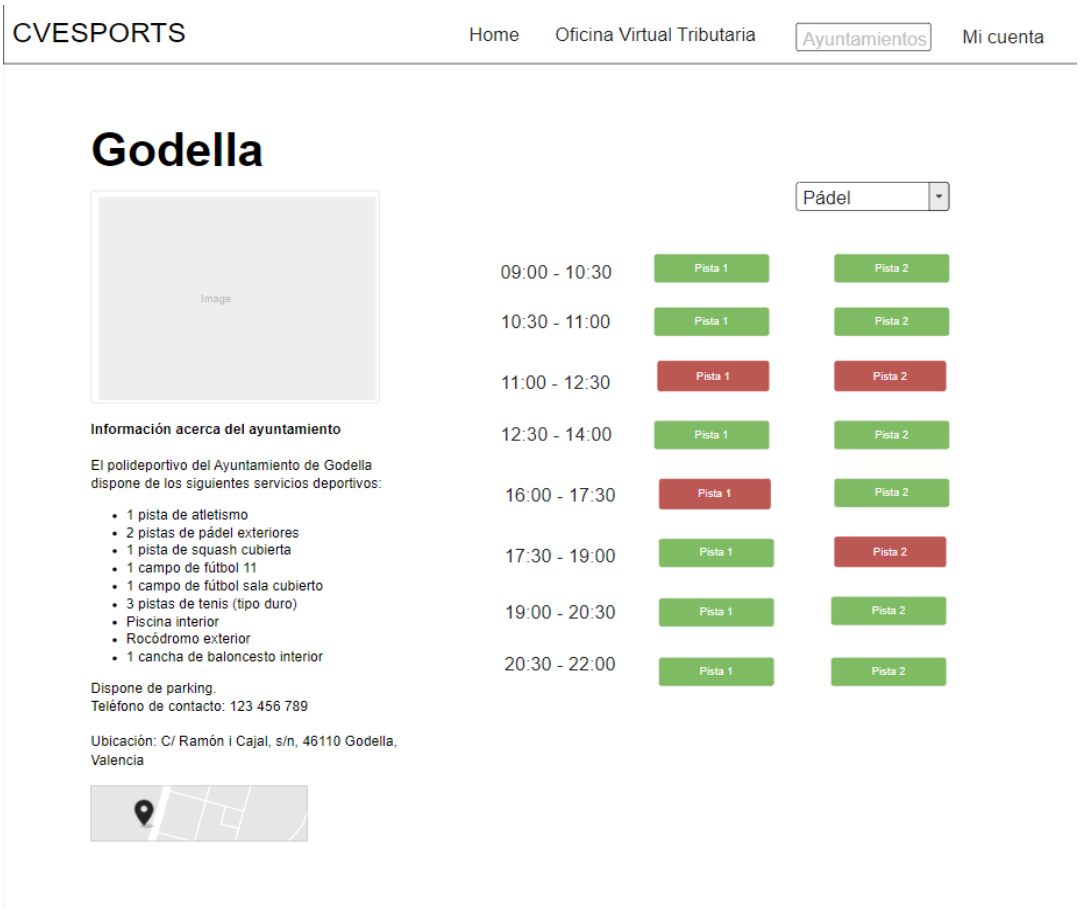

Figura 9. Mock-up de la vista para consultar servicios

La Figura 9 muestra la vista del usuario para interactuar con los servicios que ofrecen los municipios. En el menú superior el usuario busca el municipio deseado y accede a su página de servicios, donde puede consultar información como qué tipos de pistas ofrece, el teléfono de contacto, la ubicación e información extra que puede ser útil. Para reservas una pista, en este caso una de pádel, sólo hay que pulsar el botón de una pista verde, ya que las rojas no están disponibles. Eso abre un *pop-up* que avisa al usuario si está seguro de que quiere confirmar la reserva. En caso afirmativo esa pista pasa a estar en estado *reservada no pagada*. Para aclarar esto último, se muestra a continuación en la Figura 10 un diagrama de estados.

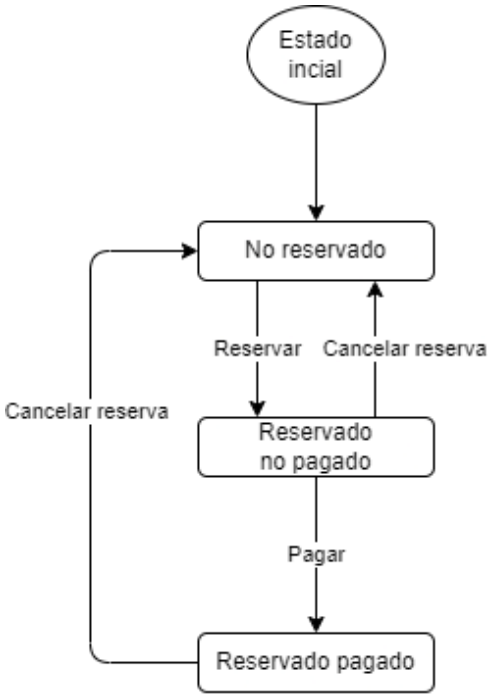

Figura 10. Diagrama de estado de reservas

Es importante remarcar que la parte visual de la aplicación debe ser independiente a la lógica de negocio y de control, para así promover la modularidad y facilitar el mantenimiento de la aplicación. Aquí es donde el front-end desempeña un papel fundamental. Suele estar construido con librerías como React o Angular y se corresponde directamente con la capa de Vista.

En CVESPORTS esta capa se encarga de que el usuario pueda interactuar con la web de manera que sea capaz de buscar un municipio y acceder a sus servicios para después proceder a reservar y realizar el pago conveniente, mientras que el administrador debe poder gestionar toda esa información y modificarla.

4.1.3 Capa de Controlador

La capa de controlador es la encargada de hacer de intermediario entre el usuario (vista) y el sistema (modelo). Recibe solicitudes de la parte visual y las procesa para llevar a cabo las acciones correspondientes, ya sea almacenar o recuperar datos. Realiza la transformación de datos para hacer que dichas capas se entiendan, por lo que traducirá la información que envía la interfaz a través de protocolos HTTP a los objetos del modelo. De la otra forma, el controlador toma la información del modelo y la adapta a la estructura de la Vista. En este sentido, el controlador actúa como puente entre el front-end y el back-end, garantizando que ambas se puedan comunicar de manera efectiva. Su funcionamiento es primordial debido a que gestiona toda la lógica del flujo de la aplicación, haciendo más fácil la experiencia de usuario y el mantenimiento del código.

4.2 Stack tecnológico

En esta fase de diseño también se ha decidido qué tecnologías se van a utilizar para el desarrollo, contemplando tanto la compatibilidad entre ellas como la versión más eficiente.

En primer lugar, para el front-end se ha utilizado Angular. Bien es cierto que hay otros frameworks de JavaScript como React, React Native y Vue.js, pero los conocimientos ya adquiridos de la tecnología escogida eran la razón de su inclusión para el desarrollo.

Pasando a la parte back-end, se eligió Java 11 porque es la última versión LTS (Long Time Service) existente, esto significa que esta versión dispone de un mayor soporte y numerosas actualizaciones. Para terminar de componer el back-end se utiliza Spring Boot con la versión 2.7.11, ya que es la última versión más estable y compatible con Java 11. En la web que ofrece Spring para crear un proyecto se podían elegir diferentes versiones, desde la ya nombrada hasta la 3.1.

Terminando con la base de datos, en un primer momento se decidió trabajar con una base de datos Oracle por su facilidad de uso y la familiarización con las bases de datos relacionales. En este punto se encuentra uno de los primeros problemas en el planteamiento del proyecto, y es que es necesario disponer de una licencia oficial para poder hacer uso de dicha base de datos, por lo tanto, se terminó optando por MySQL.

A continuación, en la Figura 11 se presenta un esquema que detalla todas estas tecnologías organizadas de acuerdo con la arquitectura de la aplicación:

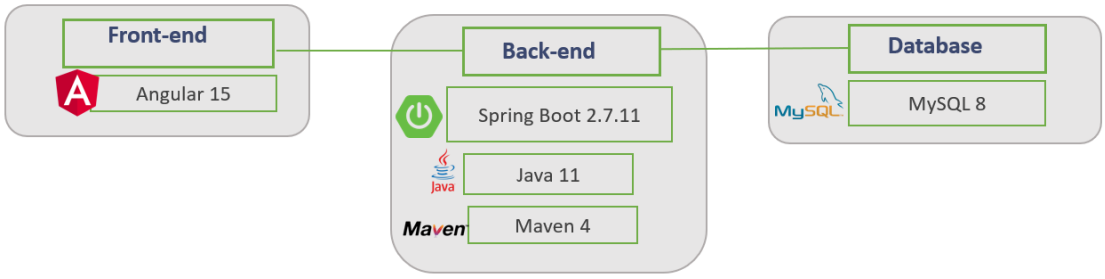

Figura 11. Stack tecnológico

5. Desarrollo de la solución

Una vez definida la arquitectura y las tecnologías que se van a utilizar se puede pasar a describir el desarrollo llevado a cabo para obtener el MVP (Minimum Viable Product). Este capítulo se desglosa en dos apartados, front-end y back-end, en los que se explica en profundidad las tecnologías nombradas en el **Capítulo 4** y las implementaciones que se han considerado más complejas.

5.1. Front-end

Para realizar la parte visual de la aplicación, lo primero que se ha realizado ha sido un estudio previo de las vistas a implementar. La aplicación tendrá 6 vistas: la vista Home, Municipios, Mis reservas, la parte de autenticación en Mi cuenta, la vista del perfil del usuario y la vista del administrador.

Estas vistas se han implementado utilizando el framework Angular. A continuación, se detallan algunos aspectos de interés de este framework y los elementos que se han utilizado en el proyecto.

5.1.1. Principales características de Angular

Angular[6] es un framework de desarrollo de aplicaciones web basado en TypeScript (un superconjunto de JavaScript que añade tipado estático y objetos basados en clases) que permite a los desarrolladores crear aplicaciones web SPA (Single Page Applications) de gran calidad y alto rendimiento.

Para entrar en contexto una SPA es una web que incluye todo el contenido en una sola página. Carga solamente un archivo HTML y todo se produce dentro de ese archivo. Esta es una forma de ofrecer una experiencia más rápida y fluida.

Siguiendo con Angular, utiliza un patrón de arquitectura MVC para permitir una separación de las responsabilidades de la aplicación y ayudar a mantener el código organizado y modular. Soporta la reutilización de componentes, es decir, Angular usa el concepto de componentes, los cuales son bloques de código reutilizables que se pueden utilizar en diferentes partes de la aplicación. Por lo que además de la organización y la modularidad, también mejora la eficiencia del código. La forma que tiene de comunicarse con el modelo es mediante el enlace de datos bidireccional, lo que significa que cualquier cambio realizado en la interfaz se va a reflejar en la capa de modelo y viceversa. La capa del modelo está en las clases TypeScript y la vista se muestra mediante los archivos HTML [11].

[6] Página web oficial de Angular: https://angular.io/

El CLI (Command Line Interface) de Angular está construido a partir de Node.js, y emplea NPM (Node Package Manager) para instalar y gestionar todas las dependencias del proyecto.

Teniendo en mente las ventajas comentadas anteriormente, Angular es una muy buena opción para llevar a cabo el desarrollo de la parte visible de la web. Se puede crear una interfaz personalizada y optimizada compatible con la gran mayoría de navegadores web. Esto último es muy importante porque lo que se busca con CVESPORTS es que esté disponible para escritorio y para móvil.

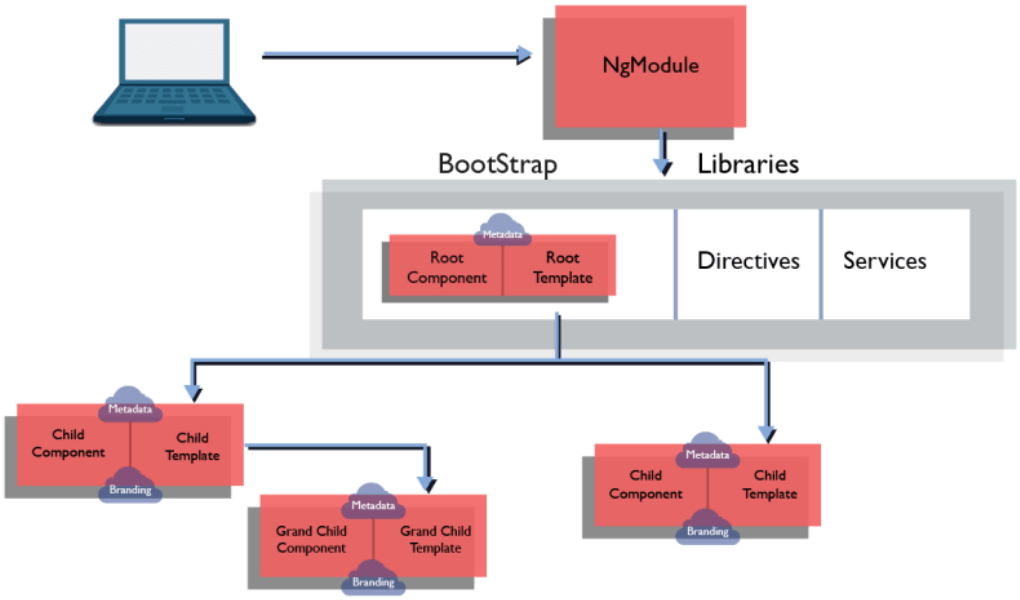

Figura 12. Arquitectura de Angular [12]

En la Figura 12 se observa un esquema de la arquitectura de Angular. NgModule representa el concepto de módulo, que es un contenedor que agrupa componentes y servicios. Cada componente tiene una plantilla HTML para definir el estilo y una clase TypeScript para controlar su lógica. Mediante el módulo de enrutamiento se permite la navegación entre componentes sean del mismo módulo o no. Los servicios se encargan de la manipulación de datos y ayudan a evitar que haya duplicidad en el código. En cuanto a las directivas, alteran los elementos del DOM, por ejemplo, si disponemos de una lista de 'x' elementos y queremos introducirla en una tabla, mediante la directiva **ngFor** se realiza la inserción. Todo esto es debido a que las plantillas de Angular son dinámicas, cuando se renderizan modifican el DOM según las directivas que hayamos introducido [13]. Existen dos tipos de directivas, las estructurales y las de atributo. El ejemplo anterior es una directiva estructural, ya que altera elementos del DOM (añadiéndolos, eliminándolos o reemplazándolos). Las de atributo alteran el estado de un elemento existente. Aquí se utiliza la directiva **ngModel** para hacer posible el *two-way data binding*. Un ejemplo para almacenar en una variable los datos introducidos en un campo de texto es *<input [(ngModel)]="user.name">*.

5.1.2. Paquetes de angular utilizados

En el desarrollo de una aplicación en Angular se hace uso de una serie de paquetes, los cuales aportan funcionalidades extra al proyecto sin añadir ningún tipo de dependencias. Se instalan mediante el ya mencionado NPM y se agregan directamente en la aplicación, en concreto en el archivo "package.json". Desde ese archivo se pueden ver todos los paquetes que están instalados en la aplicación y su respectiva versión.

Existen dos tipos diferentes de paquetes, los de la siguiente lista se conocen como "dependencies", las cuales se utilizan durante la ejecución de la aplicación y permiten que funcione correctamente en un entorno de producción.

- **@angular/animations:** este paquete proporciona animaciones y efectos visuales a la aplicación.
- **@angular/common:** contiene utilidades comúnmente utilizadas como los pipe o algunas directivas, y también proporciona servicios.
- **@angular/compiler:** es el compilador de la aplicación, se encarga de transformar las plantillas HTML a código JavaScript para que pueda ser interpretado por el navegador.
- **@angular/core:** es el paquete más importante de Angular, contiene todos los elementos necesarios para desarrollar una aplicación, como los decoradores, módulos, inyección de dependencias, entre otros.
- **@angular/forms:** este paquete contiene la funcionalidad necesaria para crear formularios reactivos.
- **@angular/platform-browser:** renderiza el lado del cliente en el navegador. Proporciona servicios para lanzar una aplicación en el navegador.
- **@angular/platform-browser-dynamic:** este paquete proporciona los métodos esenciales para compilar la aplicación de manera dinámica.
- **@angular/router:** este paquete permite la navegación entre los diferentes componentes de la aplicación.
- **@popperjs/core:** permite crear pop-ups en la aplicación.
- **jquery:** ayuda en el manejo de eventos y en la manipulación del DOM mediante una API.
- **ngx-paypal:** este paquete integra la funcionalidad de PayPal en la aplicación.
- **rxjs:** es un paquete diseñado para la programación reactiva con objetos de tipo Observable. Facilita la composición de código asíncrono.
- **tslib:** contiene funciones auxiliares para TypeScript.
- **ngx-pagination:** este paquete permite manejar la paginación de manera fácil y eficiente.
- **jwt-decode:** proporciona una manera de decodificar los JWT (JSON Web Tokens) en el lado del cliente. Es útil para acceder a cierta información que está almacenada en el token.

En cuanto a la siguiente lista, las "devDependencies son necesarias para el desarrollo de la aplicación, pero no para que se ejecute en producción. Incluyen herramientas de desarrollo, marcos de prueba, entre otros.

- **@angular-devkit/build-angular:** este paquete se utiliza en las herramientas de líneas de comandos para llevar a cabo la optimización y la compilación JIT (Just-in-Time).
- **@angular/cli:** es la interfaz de comandos de Angular.
- **@angular/compiler-cli:** es el compilador de Angular basado en la línea de comandos.
- **autoprefixer:** este paquete añade prefijos a las propiedades de CSS y ayuda a los navegadores a soportar cualquier sintaxis de CSS.
- **postcss:** permite transformar CSS con JavaScript, y es útil para la compatibilidad con navegadores antiguos.
- **tailwindcss:** este framework de CSS permite construir nuestros propios componentes desde el HTML, a diferencia de *Bootstrap* que ya los proporciona de manera predefinida. Se puede personalizar cualquier elemento de la interfaz e incorpora un sistema de diseño que permite adaptar la vista para cualquier dispositivo. En las figuras 13 y 14 se compara la sintaxis de Tailwind con su equivalente en CSS.

```
<div class="max-w-sm rounded overflow-hidden shadow-lg ml-8 hover:scale-95 duration-300">
```

Figura 13. Sintaxis en TailwindCSS

```
.shadow-lg {
    --tw-shadow: 0 10px 15px -3px rgb(0 0 0 / 0.1), 0 4px 6px -4px rgb(0 0 0 / 0.1);
    --tw-shadow-colored: 0 10px 15px -3px var(--tw-shadow-color), 0 4px 6px -4px var(--tw-shadow-
color);
    box-shadow: var(--tw-ring-offset-shadow, 0 0 #0000), var(--tw-ring-shadow, 0 0 #0000), var(--tw-
shadow);
}
```

Figura 14. Sintaxis en CSS

Ambas secciones se instalan en la carpeta de node_modules y permiten un control más preciso y evitan incluir las dependencias de desarrollo en el entorno de producción.

5.1.3 Virtual DOM

En Angular, el DOM (Document Object Model) es una representación del documento HTML en la memoria del navegador, que se puede manipular mediante TypeScript y las directivas de Angular. A diferencia de otras tecnologías web, aquí el DOM se actualiza de manera eficiente y automática gracias al uso del sistema de detección de cambios que tiene Angular.

Las plantillas HTML se combinan con los componentes TypeScript para formar las vistas, y las vistas a su vez se combinan para formar la aplicación. Cada componente tiene su propio virtual DOM. El virtual DOM es una representación ligera y eficiente del DOM real, y que se utiliza para realizar las actualizaciones y cambios de manera más rápida y eficiente [14].

Angular utiliza en enlace de datos bidireccional, lo que hace que el manejo del DOM sea más eficiente y fácil de manejar para los desarrolladores.

5.2. Back-end

A continuación, en este apartado se comentarán los aspectos más técnicos en la parte trasera de la aplicación. El back-end se ha implementado como una API REST encargada de interactuar con la base de datos.

La API está organizada en torno a los recursos, que son los objetos del sistema. Cada recurso tiene su propio endpoint, por ejemplo, Usuario tendrá *'/user'*. Estos endpoints se utilizan junto con métodos HTTP (GET, POST, PUT, DELETE) para llevar a cabo operaciones CRUD (Create, Read, Update y Delete) sobre los recursos. Por ejemplo, si quiero obtener un usuario con un id en concreto, puedo usar **GET /users/{id}**. Cada petición debe incluir toda la información necesaria para que la API pueda procesarla.

En las URL se indica la versión de la API a medida que va evolucionando. Para la primera versión quedaría tal que **/v1/users/{id}**. También se ha incluido un sistema de manejo de errores. Para ello se ha creado una clase *GlobalExceptionManager* que se aplica todos los controladores del sistema y que dentro se pueden construir el manejo de una excepción concreta. En la Figura 15 se muestra el manejo de errores en la introducción de datos erróneos en un formulario. Si detecta que algún campo no es correcto, almacena el error en una lista de mensajes y al final de la solicitud devuelve un código de respuesta 400 (Bad Request) junto a la lista.

```
@ExceptionHandler(MethodArgumentNotValidException.class)
public ResponseEntity<List<String>> handleMethodArgumentNotValidException(MethodArgumentNotValidException ex) {
    List<String> errorMessages = getNonValidFieldMessages(ex);
    return ResponseEntity
            .badRequest()
            .body(errorMessages);
}
1 usage
private List<String> getNonValidFieldMessages(MethodArgumentNotValidException ex) {
    BindingResult result = ex.getBindingResult();
    List<String> errorMessages = new ArrayList<>();
    result.getFieldErrors().forEach((fieldError) -> {
        errorMessages.add(fieldError.getField() + ": " + fieldError.getDefaultMessage());
    });
    return errorMessages;
}
```

Figura 15. Manejo de errores de la API

La API ha sido construida utilizando Spring Boot. A continuación, se comentan los aspectos más relevantes de este framework.

5.2.1. Principales características de Spring Boot

Spring Boot [7]es un framework de código abierto que utiliza la tecnología Java. Está basado en Spring, una herramienta prácticamente idéntica. Lo que la diferencia de la nombrada en primer lugar es su flexibilidad y mejor control del desarrollo, mientras que Spring Boot, se centra en simplificarlo y automatizarlo. Se utiliza para realizar el Back-end, la parte que la aplicación web que comunica la interfaz de usuario con el modelo de datos.

Una de las grandes ventajas es la inyección de dependencias, uno de los principios fundamentales en el desarrollo software. En la POO (Programación Orientada a Objetos) hay objetos que dependen de otros, por ejemplo, un objeto cliente depende de un objeto servicio. La inyección de dependencias pasa el servicio al cliente, así elimina la necesidad de que el cliente especifique qué servicio quiere usar. La inyección es la que se encarga de hacer esa especificación por el cliente.

Además, también provee varios gestores de dependencias, en CVESPORTS se utilizará Maven.

5.2.2. Comunicación con la base de datos

Dentro de Spring hay varias herramientas que facilitan la interacción con una base de datos, como JdbcTemplate, JPA y CrudRepository. Para este proyecto se ha utilizado JPA (Java Persistence API), que es mucho más sencillo y ofrece más funcionalidades que los otros dos enfoques. JPA permite interactuar objetos Java con tablas de bases de datos relacionales y proporciona un alto nivel de abstracción para las operaciones comunes de la base de datos.

A continuación, se muestra un ejemplo de cómo un método del servicio recibe un objeto Usuario mediante la llamada al repositorio:

```
@Override
public Optional<User> getUserById(Long id) {
    return Optional.ofNullable(userRepository.findById(id).orElseThrow(() -> new NoSuchElementException("No User found with id " + id)));
}
```

Figura 16. Uso de interfaces JPA

[7] Página web oficial de Spring Boot: https://spring.io/projects/spring-boot

En este caso, para que el repositorio *userRepository* pueda utilizar las interfaces que proporciona JPA (findById, saveAndFlush, delete, etc) solamente hay que extenderlo del repositorio *JpaRepository*, e indicarle el tipo de objeto que tiene que mapear junto al tipo de su clave primaria.

Cabe mencionar que CrudRepository también es una opción viable por sus métodos basados en operaciones CRUD, pero a diferencia de JPA, este no te permite personalizar tus propios métodos, como por ejemplo:

```
@Query(value = "SELECT * FROM partida p WHERE p.user_id = :user_id", nativeQuery = true)
List<Partida> findPartidasByUserId(@Param("user_id") Long user_id);
```

Figura 17. Personalización de métodos con JPA

Con la anotación *@Query* JPA permite escribir queries nativas o utilizando el lenguaje de consulta JPQL (Java Persistence Query Language). Para este caso la query será ejecutada cuando se llamé al método *findPartidasById*. Este es uno de los mayores beneficios que aporta el trabajar con un enfoque ORM (Object-Relational Mapping).

5.2.3. Dependencias Spring Boot

En Spring Boot pasa exactamente igual que con Angular, para trabajar con ciertos recursos hay que importar librerías o paquetes extra que nos ayudan a realizar una tarea determinada, aunque en este caso se llaman "dependencies". Estas dependencias están incluidas en un archivo *pom.xml,* el cual además contiene información del proyecto, plugins y perfiles. Para este proyecto se han utilizado las siguientes dependencias:

- **spring-boot-starter-data-jpa:** simplifica la configuración a la hora de utilizar JPA para interactuar con la persistencia de datos.
- **spring-boot-starter-web:** esta dependencia proporciona las herramientas necesarias para construir APIs RESTful.
- **spring-boot-starter-test:** proporciona lo necesario para realizar el testing de la aplicación, incluyendo Junit y Mockito entre otras bibliotecas.
- **spring-boot-starter-mail:** esta dependencia se utiliza para el envío de correos electrónicos mediante SMTP.
- **mysql-connector-j:** driver que conecta la aplicación con la base de datos.
- **lombok:** es una biblioteca que ayuda a reducir el boilerplate en el código (bloques de código que se escriben de manera recurrente).
- **mapstruct:** ayuda en el mapeo de objetos Java y simplifica la conversión entre diferentes tipos de beans.
- **maven-compiler-plugin:** este plugin se utiliza para compilar el código del proyecto Java.
- **springdoc-openapi-ui:** esta dependencia se utiliza para generar la documentación de la API utilizando OpenAPI.

- **mockito-core:** se utiliza para simular el comportamiento de objetos reales y aislar el código que se prueba en los tests.
- **spring-boot-starter-security:** esta dependencia agrega el marco de seguridad Spring Security, que proporciona las características de seguridad a la aplicación.
- **jsonwebtoken:** este plugin permite definir una forma segura de transmitir información entre las diferentes partes de la aplicación.

5.2.4. Apache Maven

Apache Maven[8] es una herramienta open source que se utiliza para gestionar y construir proyectos de software Java. Permite facilitar y automatizar ciertas tareas en la gestión de un proyecto de Java.

Utiliza un archivo XML llamado POM (Project Object Model) para describir el software que se debe construir, así como sus dependencias, componentes, configuración de compilación, pruebas y otros aspectos del proyecto.

Está construido utilizando una arquitectura basada en plugins

Unas de las funcionalidades que ofrece son las siguientes:

- Compilación y despliegue de aplicaciones Java (JAR, WAR).
- Gestión de las librerías requeridas por la aplicación.
- Ejecución de pruebas unitarias.
- Generación de documentación del proyecto.
- Integración con diferentes IDE (Eclipse, etc).

5.2.5. Apache Tomcat

Tomcat[9] es un servidor web y contenedor de servlets. Proporciona un entorno de ejecución para las aplicaciones web desarrolladas en Java. Se encarga de recibir las solicitudes HTTP, procesarlas y enviar las respuestas correspondientes. Es ampliamente utilizado como servidor de aplicaciones para desplegar aplicaciones web en entornos de producción.

Es el servidor web predeterminado de Spring Boot, y se inicia automáticamente cuando se lanza la aplicación.

[8] Página web oficial de Apache Maven: https://maven.apache.org/
[9] Página web oficial de Apache Tomcat: https://tomcat.apache.org/

5.2.6. Implementación de las notificaciones

La aplicación requiere de un servicio de notificaciones que permita informar a los usuarios de cualquier modificación dentro de la aplicación, para ello se ha implementado una configuración que permita utilizar el protocolo SMTP (Simple Mail Transfer Protocol) con Gmail. Lo primero que se ha hecho es solicitar una clave, generada por Google, que permita el uso de dicho protocolo en aplicaciones de terceros. Después se configura en el archivo *application.properties* del proyecto Java de la siguiente manera:

```
9  # Mail Configuration
10 spring.mail.host=smtp.gmail.com
11 spring.mail.port=587
12 spring.mail.username=tu_correo_electronico
13 spring.mail.password=clave_generada
14 spring.mail.properties.mail.smtp.auth=true
15 spring.mail.properties.mail.smtp.starttls.enable=true
```

Figura 18. Configuración SMTP

Una de las cosas que se han tenido en cuenta es qué puerto elegir. El SMTP simple utiliza el puerto 25, el problema es que la gran mayoría de proveedores de Internet lo bloquean debido a su uso abusivo para el spam. También se dispone de los puertos 465 (SMTPS) y 587 (STARTTLS), cuya diferencia es la utilidad que se les da hoy en día. Mientras que el 465 está deprecado, el 587 es el estándar. Y lo bueno que tiene es la capacidad de degradarse a una conexión no segura en caso de que sea necesario. Esto permite que el usuario pueda comunicarse con el servidor, aunque la conexión no haya sido cifrada y la acción de degradarse implica que la comunicación pueda proceder sin ser bloqueado por un firewall u otro dispositivo de red. Además, el protocolo STARTTLS también puede actualizar a una conexión más segura.

5.2.7. Implementación de la seguridad

Dado que la propuesta implica que el dispositivo del usuario interactúe con el sistema, es muy importante tener en mente mantener una API asegurada. Una API dañada o expuesta es la principal causa de las vulneraciones de la seguridad de los datos, ya que la información queda a disposición del público [15]. Para fortalecer la seguridad en nuestra API se pueden siguen diversas prácticas:

- Uso de tokens: permite autenticar de manera segura a los usuarios y limitar el acceso a los recursos a los usuarios que no han sido autenticados. El punto negativo es que los tokens pueden ser robados y si no se manejan de manera segura, eso implica que permitiría el acceso no autorizado a los recursos.

- Uso de cifrados y firmas: protege la confidencialidad e integridad de los datos y permite verificar la identidad de los usuarios para que solo los usuarios adecuados puedan descifrar sus datos.

- Uso de cupos y límites: consisten en establecer un cupo con la frecuencia con la que se puede consumir la API, por lo tanto, ante un gran número de solicitudes a la API puede mostrar un abuso. Por eso son muy útiles para prevenir ataques DDOS (denegación de servicio).

En este apartado se centra en mostrar las medidas implementadas para mantener la integridad y confidencialidad de los datos del usuario. Para ello se ha empleado JWT (JSON Web Tokens), una manera compacta para compartir información entre las diversas partes de la plataforma. Estos tokens (una cadena de caracteres) se firman digitalmente con una clave secreta para asegurar que provienen de una fuente confiable y que no han sido manipulados durante su transmisión. Si alguien intenta alterar los datos del token, la firma ya no será válida [16].

En la Figura 19 se muestra la interacción entre cliente-servidor mediante tokens. En primer lugar, el cliente envía una petición POST para autentificarse en la plataforma con sus credenciales. El servidor recibe la petición, valida los datos introducidos y genera el token a partir de la clave secreta que tenga configurada. Es importante remarcar que cuantos más bits (128, 192, 256 o 512) tenga la clave secreta más segura es, pero también requiere de más recursos computacionales, ya que la encriptación y desencriptación son procesos bastantes costosos. Una vez generado el token, el servidor lo envía al cliente para que posteriormente lo pueda utilizar para acceder a recursos de la API que requieran estar autenticado. Cuando el cliente realiza la petición utilizando el token generado por el servidor, se valida utilizando la clave secreta y se procede a realizar la llamada del endpoint correspondiente.

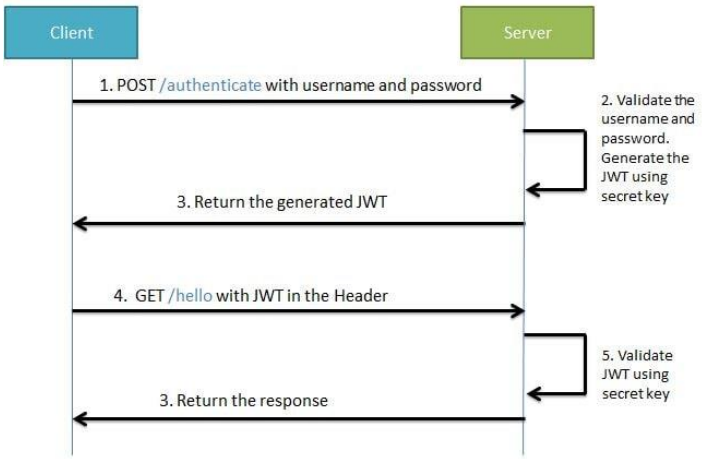

Figura 19. Flujo de trabajo del uso de tokens

Para implementar lo comentado anteriormente se han seguido una serie de pasos, empezando por la creación de un filtro de seguridad (*SecurityFilterChain*), en el cual se indican las rutas de la API que requieren que el usuario esté autenticado [17].

```
public SecurityFilterChain securityFilterChain(HttpSecurity http)
throws Exception {
    http
            .csrf()
            .disable()
            .authorizeHttpRequests()
                .antMatchers("/api/v1/auth/**")
                .permitAll()
            .and()
            .authorizeHttpRequests()
                .antMatchers("/api/v1/ayuntamientos/**")
                .permitAll()
            .anyRequest()
            .authenticated()
            .and()
            .sessionManagement()
            .sessionCreationPolicy(SessionCreationPolicy.STATELESS)
            .and()
            .authenticationProvider(authenticationProvider)
            .addFilterBefore(jwtAuthFilter,
UsernamePasswordAuthenticationFilter.class);
    return http.build();
}
```

Figura 20. Filtro de seguridad

En la Figura 20 se puede apreciar que para la ruta */api/v1/auth/*** el filtro de seguridad admite cualquier tipo de petición HTTP y no requiere que el usuario esté autenticado, indicándolo con *.permitAll()*. En las líneas *.anyRequest().authenticated()*, se indica que para cualquier otro tipo de petición el usuario debe estar autenticado.

```
public AuthenticationResponse authenticate(AuthenticationRequest
request) {
    try {
        authenticationManager.authenticate(
                new UsernamePasswordAuthenticationToken(
                        request.getEmail(),
                        request.getPassword()));
    }
        ...
    }
    User user = userRepository.findByEmail(request.getEmail())
            .orElseThrow();
    var jwtToken = jwtService.generateToken(user);
    return AuthenticationResponse.builder()
            .token(jwtToken)
            .build();
}
```

Figura 21. Captura de la request

Para que el usuario inicie sesión en la plataforma debe realizar una petición que proporcione sus credenciales, las cuales se comprueban en la base de datos para garantizar que el usuario existe. En la Figura 21 se puede observar que, una vez hecha esa comprobación, el servicio genera el token y lo retorna al usuario.

El cuerpo o payload de un token es la parte central del token que contiene las afirmaciones sobre un usuario. Existen tres tipos de afirmaciones: registradas, públicas y privadas. En las afirmaciones registradas existen nombres ya predefinidos que son reconocidos y procesados por el protocolo JWT. Las públicas pueden ser definidas a voluntad, pero no deben provocar conflictos con las afirmaciones privadas y registradas. En cuanto a las privadas, se crean para compartir información entre partes del sistema que vayan a utilizarlas.

Este podría ser un ejemplo de payload de un token decodificado: ***{ "sub": "1234567890", "name": "John Doe", "iat": 1516239022 }.*** En este ejemplo, "sub" representa el sujeto del token e "iat" define su fecha de emisión, mientras que "name" es una afirmación privada.

Para generar el token se deben establecer a las afirmaciones del payload los valores del *UserDetails*, objeto el cual contiene al usuario que ha realizado la petición de inicio de sesión. El sujeto tendrá el valor del correo electrónico, el "iat" el tiempo actual y la fecha de expiración del token será 1 hora después del tiempo actual.

```
public String generateToken(
        Map<String, Object> extraClaims,
        UserDetails userDetails
) {
    return Jwts
            .builder()
            .setClaims(extraClaims)
            .setSubject(userDetails.getUsername())
            .setIssuedAt(new Date(System.currentTimeMillis()))
            .setExpiration(new Date(System.currentTimeMillis() + 1000
* 60 * 60))
            .signWith(getSignInKey(), SignatureAlgorithm.HS256)
            .compact();
}
```

Figura 22. Generación del token

Basándonos en la arquitectura de la aplicación web, cuando el usuario introduce las credenciales, la interfaz en el lado del cliente realiza la petición HTTP que devuelve el token generado. Al recibir la respuesta, la interfaz de usuario almacena este token en el **localStorage** (Figura 23), el almacenamiento local del navegador.

```
public logIn(credentials: {email: string, password:string}):
Observable<{token: string}> {
    return this.http.post<{token:
string}>(`${this.apiAuth}/authenticate`, credentials).pipe(
      tap((response: {token: string}) => {
        localStorage.setItem('jwt_token', response.token);
      } )
    );
  }
```

Figura 23. Gestión del token en el front-end

Para poder hacer uso de los datos del token, es necesario decodificarlo. Se ha hecho uso de la librería *jwt_decode*, que proporciona las herramientas necesarias para extraer la información del token. En la Figura 24 se muestra la función creada para decodificar el token y almacenar en un objeto **User** el usuario que ha iniciado sesión, para poder trabajar con sus datos. Primero se almacena en una constante el token almacenado en localStorage. Se decodifica haciendo uso de la función *jwtDecode* y almacena el token decodificado en otra constante. Finalmente utiliza el sujeto del token decodificado, el cual representa el correo electrónico del usuario para hacer una petición GET y obtener el usuario.

```
getCurrentUser(): User | null {
    const token = localStorage.getItem('jwt_token');
    if(!token) {
      return null;
    }
    const decodedToken = jwtDecode<DecodedToken>(token);
    const user: User = this.getLogedUser(decodedToken.sub);
    return user;
  }
```

Figura 24. Decodificación del token

5.2.8. Documentación de la API

Documentar una API es un paso fundamental del desarrollo que facilita la comprensión del código por parte de los miembros de un equipo o de cualquier otra persona que vaya a interactuar con la API. La documentación proporciona claridad sobre las operaciones que se llevan a cabo y agrega un contexto que favorece la mantenibilidad a largo plazo del proyecto.

Existen herramientas muy conocidas para esta tarea, como JavaDoc[10], Swagger[11], TypeDoc[12], etc. Aquí se ha optado por Swagger para documentar la API REST. Swagger es una herramienta que permite describir la estructura de tu API para que posteriormente el programa lo lea. Como resultado final proporciona una interfaz web gráfica donde se pueden probar los endpoints mientras se consulta la API. Es de gran ayuda para desarrolladores y usuarios no tan experimentados.

Figura 25. Documentación del método de un controlador mediante Swagger

Swagger muestra todas las llamadas a la API separándolas en sus respectivos controladores. En la Figura 25 se observa un método HTPP DELETE en el que se pasa como parámetro el id del usuario que se quiere eliminar de la base de datos.

5.2.9. Refactoring

Se han realizado varias tareas de refactoring durante el desarrollo del proyecto. A continuación, se describen algunas de las refactorizaciones más aplicadas.

La refactorización *Extract Method* consiste en extraer una sección de código de un método ya existente en un método aparte, con la finalidad de poder reutilizarlo en otros puntos de la aplicación. Con esta estrategia se evitan repeticiones de código, reducir complejidad y aumentar la modularidad.

[10] Página web oficial de JavaDoc: https://docs.oracle.com/javase/8/docs/technotes/tools/windows/javadoc.html
[11] Página web oficial de Swagger: https://swagger.io/solutions/api-documentation/
[12] Página web oficial de TypeDoc: https://typedoc.org/guides/doccomments/

```
private readonly PAYPAL_CLIENT_ID = "example_ID";

ngOnInit(): void {

  this.configurePaypal();
}

private configurePaypal(): void {
  this.payPalConfig = {
    currency: "EUR",
    clientId: this.PAYPAL_CLIENT_ID,
```

Figura 26. Servicio de PayPal

Como se muestra en la Figura 26, el código que maneja la configuración del servicio de PayPal para la realización de pagos ha sido refactorizado. Este código ha sido encapsulado en un método independiente y reutilizable, con el objetivo de mejorar la modularidad y evitar la duplicación innecesaria de código en otros componentes.

Otra refactorización que se ha utilizado es *Replace Magic Number with Symbolic Constant*, con el objetivo de facilitar la mantenibilidad del código. En las figuras Figura 26 y Figura 27 se muestran dos casos que han requerido esta refactorización. En el código de la Figura 26 se ha creado una constante que almacene el identificador del para que pueda ser utilizada sin tener que codificarla.

Lo mismo sucede en el código de la Figura 27, para las rutas de navegación de la aplicación se ha optado por crear un archivo que almacene en constantes sus respectivas rutas de acceso, así es más fácil indicar en el código por dónde se navega.

```
export class AppUrls {
    public static readonly HOME_ROUTE = 'home';

    public static readonly ERROR_ROUTE = 'error-page';

    public static readonly NO_PERMISSION_ROUTE = 'no-permission';

    public static readonly PROFILE_ROUTE = 'myProfile';

    public static readonly AYUNTAMIENTO_ROUTE = 'reservaPistas';
}
```

Figura 27. Rutas de la aplicación

5.2.10. Patrón Repository

El patrón repositorio es un patrón de diseño destinado a mantener las preocupaciones de persistencia fuera del modelo de dominio del sistema. Una o más abstracciones de persistencia (interfaces) se definen en el modelo de dominio, y estas abstracciones tienen implementaciones en forma de adaptadores específicos a persistencia definidos en otra parte de la aplicación [18].

Las implementaciones del repositorio son clases que encapsulan la lógica necesaria para acceder a las fuentes de datos. Centralizan la funcionalidad de acceso a datos proporcionando una mejor mantenibilidad y desacoplando la infraestructura utilizada para acceder a la base de datos [19].

En la Figura 28 se puede ver la interfaz del repositorio de la clase User. Cuando se realiza una llamada desde el servicio, esta interfaz es la que se encarga de hacer los mapeos entre el objeto User y la base de datos.

```
@Repository
public interface UserRepository extends JpaRepository<User, Long> {
    @Query(value = "SELECT * FROM user u WHERE u.email = :email",
nativeQuery = true)
    Optional<User> findByEmail(@Param("email") String email);

}
```

Figura 28. Repositorio de Usuario

6. Pruebas

En este penúltimo capítulo se comentan los distintos tipos de pruebas realizadas que verifican la funcionalidad de la aplicación web, así como su rendimiento, para asegurar que cumple con las expectativas y necesidades del usuario.

El proceso de pruebas es fundamental para garantizar que el código funciona sin problemas y realizar las operaciones deseadas. Existen varios tipos de pruebas aplicables durante el ciclo de vida de un producto software que evalúan diferentes aspectos de su funcionamiento. En este proyecto se han realizado los siguientes tipos de pruebas: unitarias, de integración y de usabilidad.

6.1. Pruebas unitarias

Las pruebas unitarias consisten en evaluar de manera aislada una funcionalidad de código. Son de gran ayuda para identificar y corregir errores y permiten un desarrollo más rápido y de mayor calidad.

Para realizar estas pruebas se ha utilizado *Mockito*[13], un framework de pruebas unitarias para Java. La peculiaridad que tiene es que permite crear objetos "mock" que se pueden utilizar para probar unidades funcionales independientes, sin tener que afrontar problemas de dependencias que pueda tener con otra unidad funcional. Con el término "mock" nos referimos a la simulación de un objeto real para controlar el comportamiento de la unidad funcional. Esto ayuda a que solamente nos tengamos que focalizar en testear la funcionalidad del código en lugar de tener presente el impacto que puedan tener las dependencias sobre las pruebas.

```
@Test
void getUserById() {

when(userRepository.findById(user.getId())).thenReturn(Optional.of(use
r));

    Optional<User> returnedUser =
userService.getUserById(user.getId());

    assertTrue(returnedUser.isPresent());
    assertEquals(user.getId(), returnedUser.get().getId());
    verify(userRepository, times(1)).findById(user.getId());
}
```

Figura 29. Estructura de código de una prueba unitaria

[13] Página web oficial de Mockito: https://mvnrepository.com/artifact/org.mockito/mockito-core

En la Figura 29 se analiza el comportamiento de la función “getUserById”, que se encarga de devolver un Usuario dado un Id. Aquí se crea un mock del repositorio, el cual maneja todas las operaciones con la base de datos para la entidad User. La razón por la que se “mockea” es para no tener que depender de la base de datos real y poder controlar así su comportamiento. En la prueba se almacena en una variable “returnedUser” el usuario encontrado en la base de datos. Por último, se realizan tres aserciones que determinarán si la función del servicio ha pasado la prueba o no. En la primera se comprueba que el usuario no es nulo, en la segunda compara que el identificador del usuario devuelto coincide con el identificador proporcionado para la prueba, y en la última se verifica que ese identificador sea único en la base de datos.

6.2. Pruebas de integración

Las pruebas de integración implican la integración de diversos módulos de la aplicación y probar su comportamiento como una unidad combinada. El foco está en verificar que las unidades individuales se comunican correctamente entre sí. Para realizar estas pruebas se han utilizado MockMVC y Spring Test, que se integra con las tecnologías de pruebas unitarias. En este escenario son necesarias ya que hay que tratar como un conjunto a toda la lógica de negocio. Consiste en simular las peticiones que se realizan desde la capa de Vista para comprobar que las respuestas generadas son las correctas y no existe ningún error.

```
@Test
void getAllUsers() throws Exception {
    MvcResult mockMvcResult =
mockMvc.perform(MockMvcRequestBuilders.get(BASE_URL))
            .andReturn();
    assertEquals(200, mockMvcResult.getResponse().getStatus());
}
```

Figura 30. Estructura base de una prueba de integración

En la Figura 30 se puede ver la estructura de la prueba de integración de la función “getAllUsers”. En primer lugar, se almacena en un objeto MvcResult el resultado de la petición GET al endpoint que se quiere probar. Después se realiza una aserción que compara el resultado esperado (200 OK) con el estado de la respuesta la petición. Con esa línea se verifica que la solicitud se ha procesado correctamente.

También hay que tener en cuenta las respuestas erróneas, es decir, cuando la solicitud “falla exitosamente”. Es importante cubrir los casos erróneos para probar cómo se comporta el sistema ante entradas inválidas. Además, si algunos errores no se manejan correctamente, pueden provocar fallos graves en el sistema.

```
@Test
void userDoesNotExist() throws Exception {
    MvcResult mockMvcResult =
mockMvc.perform(MockMvcRequestBuilders.get(BASE_URL + "/" + 23))
            .andReturn();
    assertEquals(404, mockMvcResult.getResponse().getStatus());
}
```

Figura 31. Estructura de una prueba de integración con detección de fallos

En la Figura 31 se puede observar que la estructura de la prueba es igual a la de la Figura 30, la diferencia es la respuesta que espera recibir. El objetivo de esta prueba es cubrir el caso en el que se busque un usuario que no existe. Para este ejemplo, se utiliza un identificador aleatorio, por ejemplo 23, y se realiza la petición. En la aserción la respuesta debe coincidir con el estado 404 NOT_FOUND, para confirmar que el usuario no existe.

6.3. Pruebas de usabilidad

Las pruebas de usabilidad están enfocadas en evaluar la facilidad de uso de una aplicación web mediante usuarios reales. Los usuarios utilizan la aplicación para completar tareas específicas y, durante este proceso, los observadores analizan y toman nota de cómo interactúan con ella.

Para evaluar la usabilidad del MVP de este proyecto, se han utilizado cinco participantes, usuarios potenciales de la aplicación. Estos participantes han sido reclutados por el autor del trabajo. Los perfiles de estos usuarios son usuario y administrador. Entre los usuarios, se encuentran individuos con experiencia básica en el uso de aplicaciones digitales. Por otro lado, los administradores son profesionales con experiencia en la gestión de aplicaciones digitales, que están familiarizados con las tareas de supervisión y control.

Cada usuario ha utilizado la aplicación en una sesión presencial junto al autor del trabajo de aproximadamente 5 minutos. En esta sesión, el participante debía utilizar la aplicación y realizar las siguientes tareas relacionadas con el rol de usuario:

- Registrarse en la plataforma.
- Iniciar sesión.
- Iniciar sesión, entrar en la plataforma y cerrar sesión.
- Completar una reserva.
- Cancelar una reserva.
- Modificar sus datos personales.
- Eliminar su cuenta.

En cuanto a las tareas del administrador, el participante debía realizar las siguientes tareas:

- Consultar usuarios registrados.
- Consultar reservas de un municipio.
- Eliminar una reserva.
- Consultar información de la reserva.
- Crear un nuevo servicio.
- Eliminar un usuario del sistema.

El objetivo es identificar cualquier problema de usabilidad y determinar el nivel de satisfacción del usuario. Para este caso, se han utilizado formularios de Google donde se les pregunta a los usuarios la facilidad para hacer las diferentes tareas. Los participantes completarán estos formularios según su experiencia durante la prueba. La respuesta del usuario se realiza según la escala de Likert [20]. Atendiendo a esta escala, los participantes calificarán los enunciados de la encuesta desde un rango de opciones predefinidas. Este rango es de cinco puntos, donde el valor más bajo representa una respuesta negativa (*"Nada satisfecho"*) y el valor más alto indica una respuesta positiva (*"Totalmente satisfecho"*).

Una vez los participantes de la prueba hayan respondido a las encuestas, los datos recogidos se analizarán para entender mejor los problemas de usabilidad y el nivel de satisfacción del usuario. Este feedback ayudará a mejorar ciertos aspectos de la plataforma para proporcionar una experiencia de usuario más satisfactoria y eficiente.

A continuación, en la Figura 32 y Figura 33 se muestran los resultados de las pruebas de usabilidad. El valor más alto de la gráfica representa que el participante está *Totalmente satisfecho*, y el más bajo, que está *Nada satisfecho*.

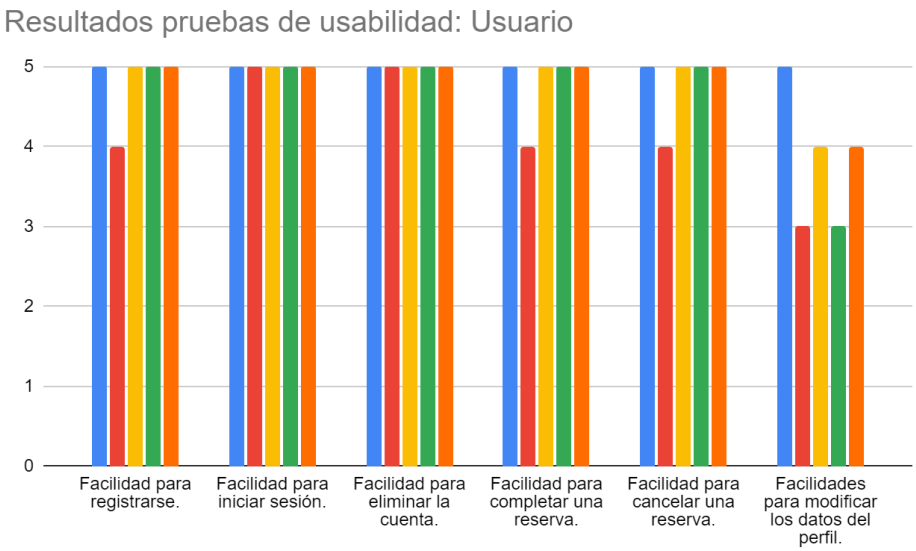

Figura 32. Resultados pruebas usabilidad de Usuario

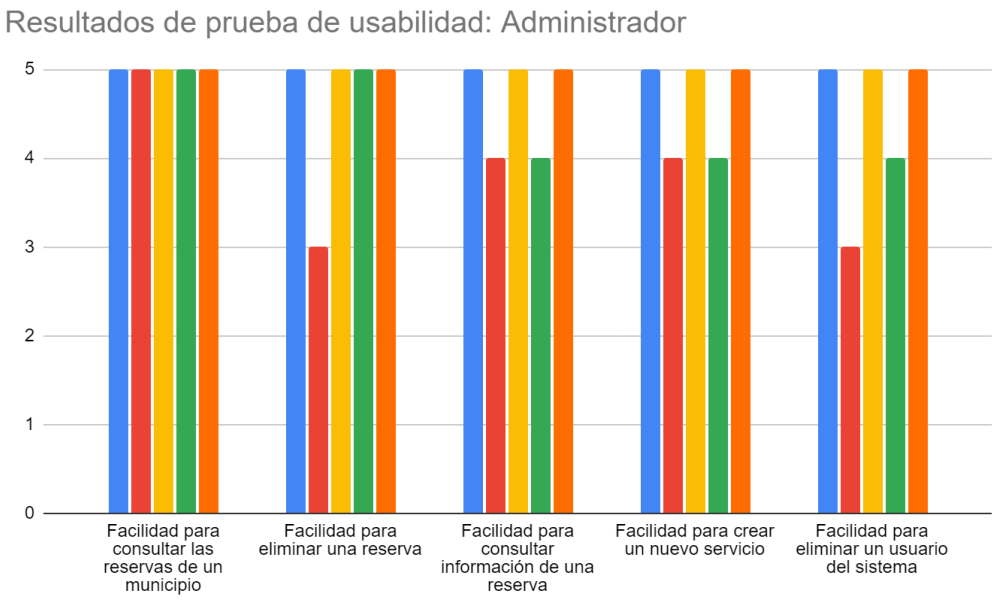

Figura 33. Resultados pruebas usabilidad de Administrador

7. Conclusiones

En este apartado se muestran algunas reflexiones finales, la relación del trabajo con asignaturas del grado de Ingeniería Informática, y el trabajo futuro.

7.1. Reflexiones finales

CVESPORTS es una idea que surgió a raíz de las necesidades que el autor de este trabajo detectó a la hora de interactuar con las páginas web de los centros deportivos públicos para reservas de pistas. Fueron numerosas las personas que compartían la misma opinión de que la interacción con las plataformas web no era práctica. A partir de ese momento comenzó un proceso de investigación y establecimiento de requisitos y objetivos para llevar a cabo el desarrollo de lo que sería este TFG. Todos esos objetivos marcados en la fase inicial se fueron cumpliendo de manera que se pudiera interactuar con la plataforma progresivamente. Hubo tareas fáciles y otras no tanto, las cuales requirieron de más tiempo, y por ello ha habido algunas que no se pudieron realizar o incluso pulir debido a la limitación de tiempo. Una de ellas es la de que la plataforma se encargue de hacer recomendaciones al usuario según sus preferencias, por ejemplo, si el usuario está principalmente interesado en jugar a pádel en Godella en horario de tardes, la plataforma debe ser capaz de notificarle vía correo electrónico aquellas partidas libres que hacen match con sus preferencias.

A lo largo del proceso de desarrollo he adquirido muchas aptitudes a nivel académico y profesional. Tanto en la fase de análisis como en la de desarrollo la curva de aprendizaje ha sido alta debido al desconocimiento para diseñar un proyecto de manera eficiente y de las tecnologías que se han utilizado.

7.2. Relación con asignaturas del Grado de Ingeniería Informática

Mis habilidades en el lenguaje de programación TypeScript, junto con lo aprendido en la asignatura Bases de Datos en la universidad, se han convertido en la base para mi trabajo en el desarrollo de un software viable y funcional. También asignaturas como Ingeniería del Software, Diseño de Software y Calidad de Software han sido de gran ayuda para establecer unas bases que servirían para mantener una interfaz limpia, simple y fácil de aprender, refactorizar código y ver la plataforma desde el punto de vista del cliente.

7.3. Trabajo futuro

Durante el desarrollo se han identificado algunas características que pueden ser incluidas en futuras actualizaciones de la plataforma. Se podría implementar un sistema de partidas abiertas, que el usuario pueda inscribirse en una partida con otras personas en caso de que no pueda reservar una partida para él solo. Este tipo de funcionalidad sería especialmente útil para deportes como el tenis o el pádel, donde es necesario contar con más de una persona.

Además, podría ser útil ofrecer al administrador un calendario interactivo para añadir eventos u otro tipo de información. Este tipo de herramienta ayudaría a los administradores a gestionar eficientemente los recursos y los horarios, y proporcionaría una visión clara y ordenada de las actividades y reservas del municipio.

Otra área de expansión potencial es la integración con las aplicaciones existentes que ya están siendo utilizadas por algunos municipios. Muchos municipios utilizan sistemas de gestión propios para llevar a cabo sus operaciones diarias. Por lo tanto, la posibilidad de integrar CVESPORTS con estos sistemas ya existentes permitiría una transición más fluida y una adopción más fácil de nuestra plataforma.

Cabe mencionar que este proyecto no queda aquí, está pensado mejorar aspectos para que sea la mejor solución para aplicar en el ámbito de las gestiones de reservas de los centros deportivos municipales. Se seguirán añadiendo nuevas funcionalidades a medida que se analizan las opiniones de los usuarios sobre la plataforma.

Referencias

[1] Sports Booker. (s.f.). Advantages of using an online sports and leisure booking system. Recuperado el 13 de mayo de 2023, de https://sports-booker.com/advantages-of-using-an-online-sports-and-leisure-booking-system/

[2] Bookteq. (2021). How to Maximise the Growth of Your Sports Facilities with Online Booking Software. Recuperado el 13 de mayo de 2023, de https://www.bookteq.com/how-to-maximise-the-growth-of-your-sports-facilities-with-online-booking-software/

[3] La Manzana Mordida. (2020). Playtomic App: Juega al pádel, tenis, fútbol y otros deportes. La Manzana Mordida. Recuperado el 21 de mayo de 2023, de https://lamanzanamordida.net/aplicaciones/recomendadas/playtomic-app-jugar-padel-tenis-futbol-otros-deportes/

[4] TPC Matchpoint. (s.f.). Software de gestión de clubs de pádel. Recuperado el 22 de mayo de 2023, de https://tpcmatchpoint.com/software-gestion-clubs-padel.html#prettyPhoto

[5] Playtomic. (s.f.). Playtomic Manager. Recuperado el 23 de mayo de 2023, de https://playtomic.com/es/playtomic-manager/

[6] Atlassian. (s.f.). Scrum. Recuperado el 28 de abril de 2023, de https://www.atlassian.com/agile/scrum

[7] DesarrolloWeb. (2020). Qué es MVC. DesarrolloWeb. Recuperado el 14 de mayo de 2023 de https://desarrolloweb.com/articulos/que-es-mvc.html

[8] Riahi, S. (2018). Modèle MVC. SlideShare. Recuperado el 14 de mayo de 2023, de https://fr.slideshare.net/SoulefRiahi/modele-mvc

[9] Mozilla Developer Network. (2022). MVC: Arquitectura de Software. Recuperado el 2 de julio de 2023, de https://developer.mozilla.org/es/docs/Glossary/MVC

[10] CakePHP. Understanding Model View Controller. Recuperado el 20 de junio de 2023, de https://book.cakephp.org/2/es/cakephp-overview/understanding-model-view-controller.html

[11] O'Hanlon, M. (2021). Angular Cookbook: Over 80+ hands-on recipes to architect performant applications and implement best practices in Angular. O'Reilly Media. Recuperado el 14 de mayo de 2023, de https://learning.oreilly.com/library/view/angular-cookbook/9781838989439/

[12] Marely, V. (2021). Arquitectura en Angular. Dev Community. Recuperado el 2 de julio de 2023, de https://dev.to/vanessamarely/arquitectura-en-angular-5c23

[13] Angular.io. (2022). Developer Guide: Architecture. Recuperado el 24 de junio de 2023, de https://v2.angular.io/docs/ts/latest/guide/architecture.html

[14] Knoldus Inc. (2022). Understanding Shadow and Virtual DOM [Blog post]. Recuperado el 20 de junio de 2023, de https://blog.knoldus.com/understanding-shadow-and-virtual-dom/#headThree

[15] Red Hat. (2019). Seguridad de las API: Qué es, por qué es importante y cómo implementarla. Recuperado el 21 de junio de 2023, de https://www.redhat.com/es/topics/security/api-security

[16] JavaInUse. (s.f.). Spring Boot + JSON Web Token (JWT) Hello World Example. Recuperado el 22 de junio de 2023, de https://www.javainuse.com/spring/boot-jwt

[17] Baeldung. (2022). Migrating off of Deprecated Spring Security's OAuth2 OAuth2ClientContext. Recuperado el 22 de junio de 2023, de https://www.baeldung.com/spring-deprecated-websecurityconfigureradapter

[18] Fowler, M. Repository. (s.f.). Martin Fowler. Recuperado el 28 de junio de 2023, de https://martinfowler.com/eaaCatalog/repository.html

[19] Microsoft. (2023). .NET Microservices. Architecture for Containerized .NET Applications | Infrastructure and Persistence Layer Design. Recuperado el 27 de junio de 2023, de https://learn.microsoft.com/en-us/dotnet/architecture/microservices/microservice-ddd-cqrs-patterns/infrastructure-persistence-layer-design

[20] ¿Qué es la escala de Likert y cómo utilizarla? (s.f.). QuestionPro. Recuperado el 26 de junio de 2023, de https://www.questionpro.com/blog/es/que-es-la-escala-de-likert-y-como-utilizarla/

Apéndice

A. Objetivos de Desarrollo Sostenible

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

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

CVESPORTS tiene una relación media con el Objetivo de Desarrollo Sostenible número 3, “Salud y Bienestar”, ya que la plataforma fomenta el deporte y mantener una vida saludable y activa para mejorar el bienestar físico y mental. Y es que el acceso a instalaciones deportivas es un factor clave para alcanzar este objetivo.

En cuanto al Objetivo de Desarrollo Sostenible 4, “Educación de Calidad”, mantiene una relación media con la plataforma, ya que puede ayudar a los profesionales a impartir su docencia, en este caso la que esté relacionada con el deporte, y a los alumnos a seguirla. En concreto, la meta 4a busca adecuar las instalaciones educativas

que tienen en cuenta las diferencias de cada persona, ya sean discapacidades u otras, para ofrecer un ecosistema de aprendizaje seguro, inclusivo y eficaz.

Siguiendo con el Objetivo de Desarrollo Sostenible 5, “Igualdad de género”, está ligeramente relacionada con la plataforma, ya que no se hace referencia a la igualdad de género. Sin embargo, es importante comentar que la aplicación ofrece las mismas posibilidades a la hora de reservar un servicio deportivo y utilizarlo, ya seas hombre o mujer. Para ser más específicos, en la meta 5b se busca, a través de las tecnologías, promover el empoderamiento de las mujeres.

La relación con el Objetivo de Desarrollo Sostenible 10, “Reducción de las desigualdades”, también es baja, ya que no se hace una referencia directa a este objetivo. Pero, al igual que se ha comentado en el párrafo anterior, la plataforma contribuye a la inclusión social y reducción de desigualdades al permitir que cualquier usuario tenga la capacidad de realizar un deporte específico.

Por último, el Objetivo de Desarrollo Sostenible 16, “Paz, justicia e instituciones”, mantiene una relación baja ya que la plataforma no hace referencia directa a la paz y justicia. Sin embargo, es importante recalcar que el deporte se puede utilizar como una herramienta para promover la paz y resolución de cualquier tipo de conflicto. Aquí entrarían en juego los valores deportivos, como el juego limpio, el respeto y la cooperación, estos pueden ayudar a mejorar las sociedades a unas más justas y pacificas. Un claro ejemplo es el problema que hay en la actualidad con el racismo. En todos los deportes se promueve el rechazo a todo comportamiento racista, se puede ver en cualquiera, hasta en las competiciones más famosas del mundo. Por lo tanto, lo que busca la plataforma es evitar todos aquellos comportamientos que inciten al odio y promover valores humanos para una convivencia sana.

B. Manual de usuario

Una vez dentro de la plataforma, la primera pantalla que ve el usuario es la de bienvenida. Se podría haber tenido en cuenta otra forma de hacerlo, como el típico inicio de sesión antes de acceder al dashboard, pero eso restaría usabilidad para el enfoque que se le quiere dar. Lo que se busca es que cualquier usuario pueda consultar la información esté registrado o no, y en caso de querer interactuar que deba darse de alta en el sistema.

Dentro de la vista principal se pueden acceder mediante el menú superior a las diferentes vistas.

Figura 34. Pantalla de inicio de CVESPORTS

En la Figura 35 se muestra al usuario la pantalla para iniciar sesión, que se puede alternar según si quiere registrarse o no. En caso de no recordar la contraseña, la plataforma da la opción de recuperarla a través del correo electrónico. Una vez completada la solicitud el usuario recibirá un correo con la contraseña que tenía configurada.

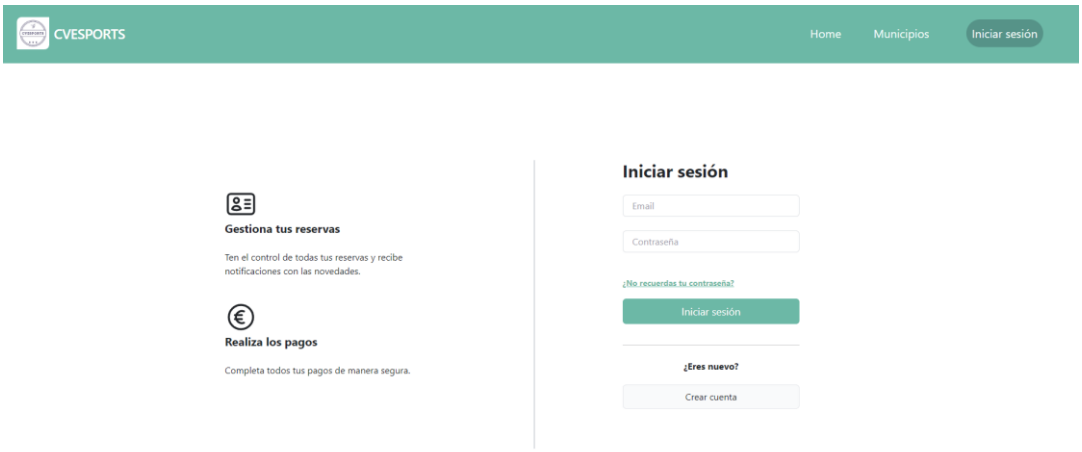

Figura 35. Pantalla de inicio de sesión y registrarse

Una vez iniciada la sesión, si el rol del usuario no es *ADMIN*, se le redirige a la pantalla de la Figura 36, donde puede ver toda la información de su cuenta mientras la sesión siga activa. Se puede modificar información del perfil, cerrar sesión y dar de baja del sistema. Además, también permite introducir información relacionada con su ubicación para facilitar la interacción con su municipio y los municipios cercanos. Todas estas acciones vienen acompañadas de un previo aviso en forma de pop-up en caso de que el usuario cambie de opinión o pulse un botón por error.

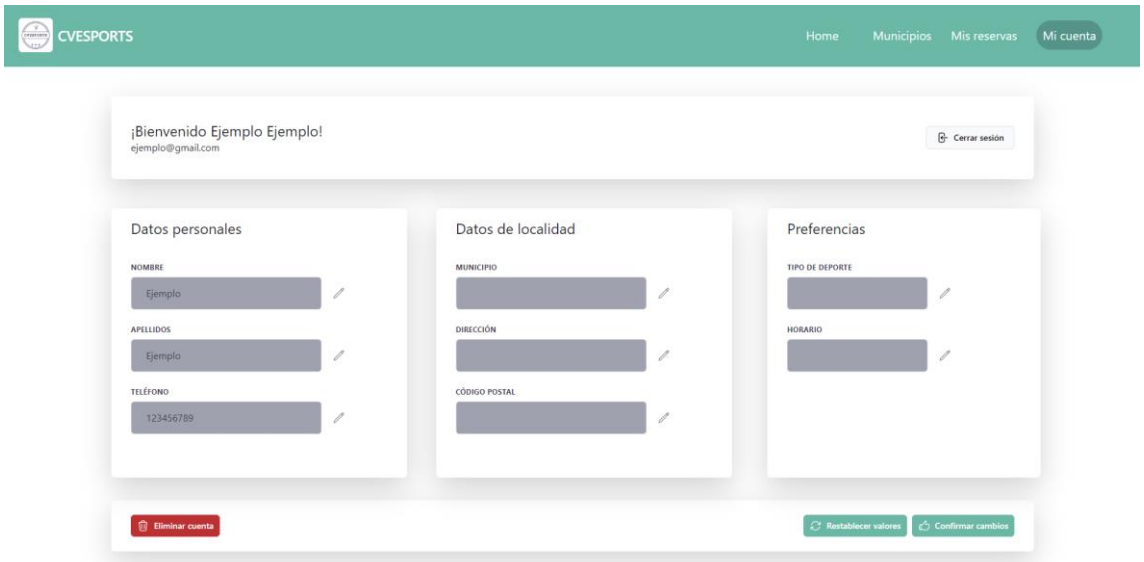

Figura 36. Dashboard del usuario

En esta pantalla de la Figura 37 se puede ver una lista de las reservas realizadas y caducadas. Estas últimas se destacan para que puedan diferenciarse de las que están aún activas y tener así un historial dentro de la lista. Para interactuar con ella se disponen de dos botones colocados en dos columnas distintas, uno para cancelar la reserva y otro para proceder al pago. La de cancelar la reserva abrirá un pop-up para confirmar si se quiere cancelar la reserva, y la de pagar abrirá otro pop-up para elegir entre tres métodos de pago diferentes, y según el elegido se redirigirá al usuario al servicio pertinente.

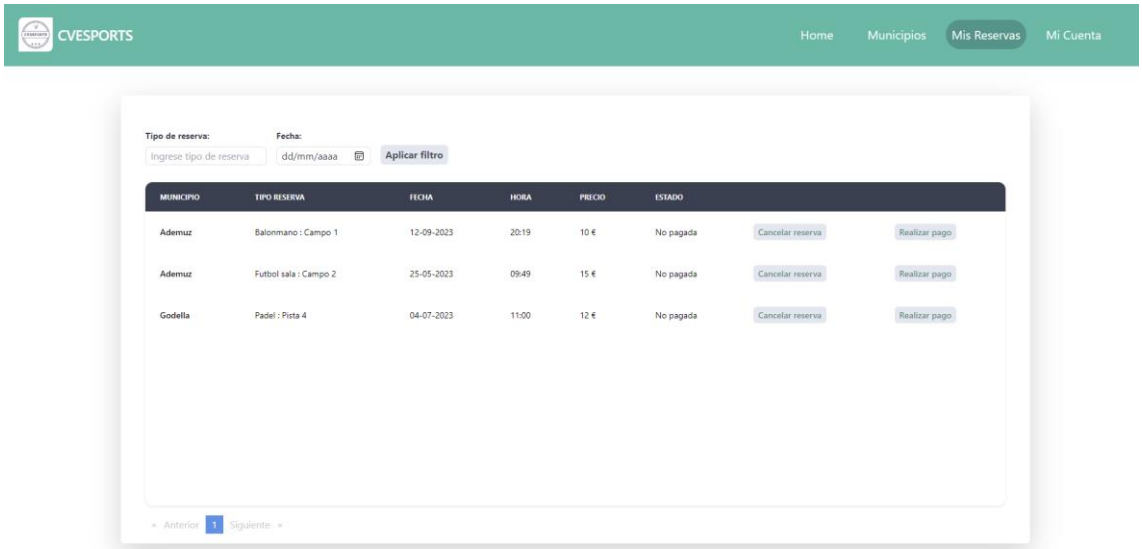

Figura 37. Vista "Mis Reservas"

Dentro la vista de municipios, en la Figura 38, se muestra toda la información del municipio dependiendo del que elija el usuario. Se muestran una imagen de los servicios deportivos de ese municipio, la ubicación y la lista con los servicios que ofrece, con la que se interactúa de la misma manera que con la lista de reservas, con una columna que da la opción a reservas en caso de que el estado de la pista sea *Libre*.

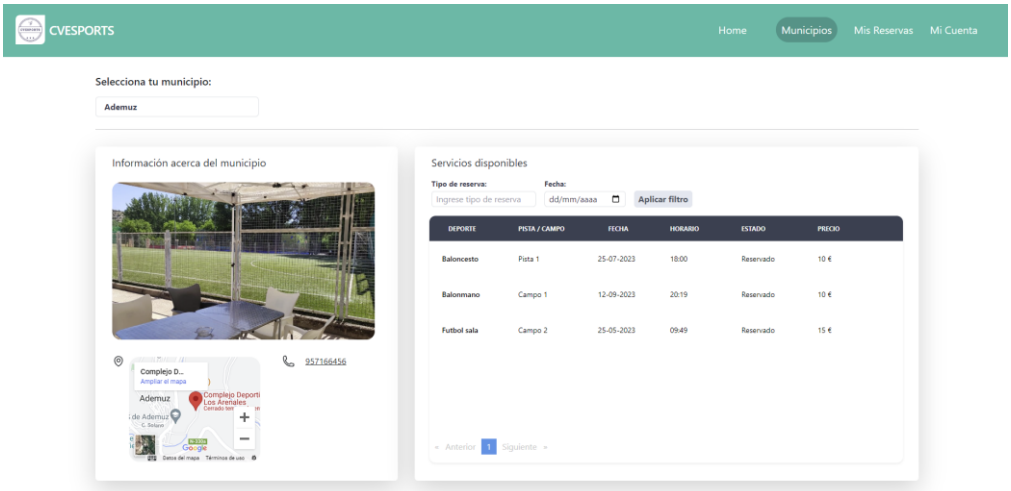

Figura 38. Vista "Municipios"

En cuanto a la parte del administrador de la Figura 39, este dispone de una vista exclusiva para editar información del municipio que gestiona, ver los usuarios registrados en la plataforma, ver su lista de servicios y manipularlos. Para acceder a ella el administrador debe acceder con una clave única.

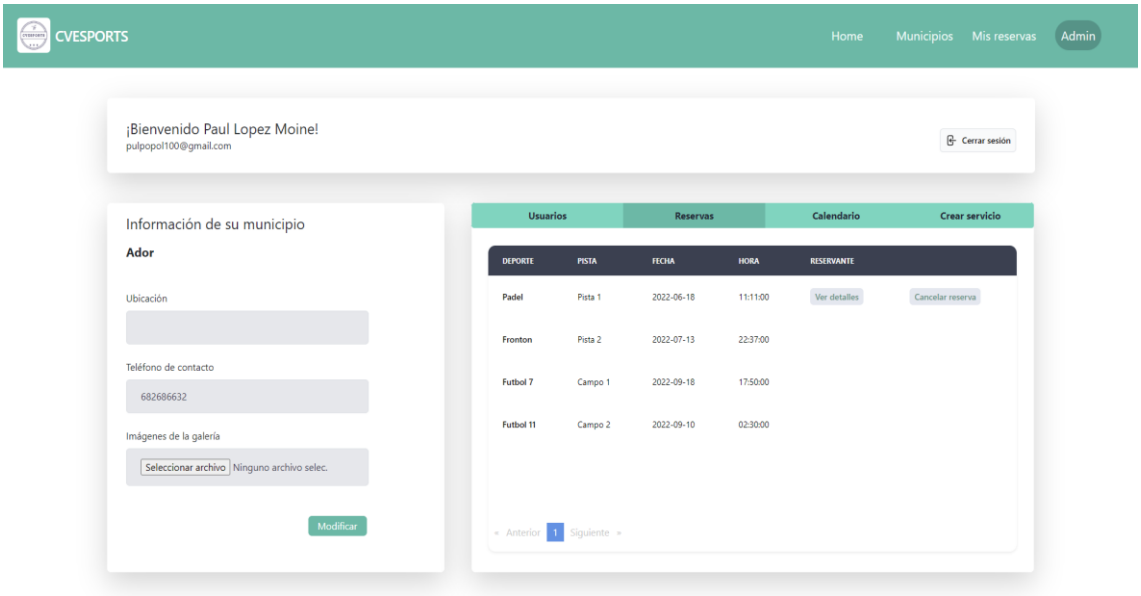

Figura 39. Dashboard del administrador