TRABAJOS FIN DE GRADO

Universidad de Jaén

Escuela Politécnica Superior de Jaén

Sistema de gestión de reservas y participación para pádel y deportes similares

Autor: **Eduardo Lomas Recena**

Grado: **Ingeniería Informática**

Fecha: 09/09/2024

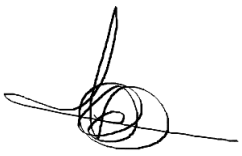
Universidad de Jaén

Departamento de Informática

Don Antonio Jesús Rueda Ruíz, Tutor del Trabajo Fin de Grado titulado: '**Sistema de gestión de reservas y participación para pádel y deportes similares**', que presenta Don Eduardo Lomas Recena, otorga el visto bueno para su entrega y defensa en la Escuela Politécnica Superior de Jaén.

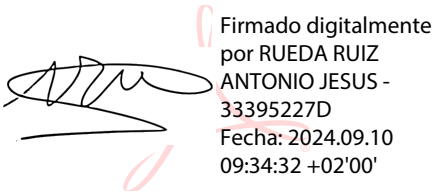
Jaén, 09 de 2024

El alumno:



Eduardo Lomas Recena

El tutor:



Antonio Jesús Rueda Ruíz

Agradecimientos

Quiero expresar mi más sincero agradecimiento a todas las personas que, de una u otra manera, han hecho posible la realización de este Trabajo de Fin de Grado.

En primer lugar, desea agradecer a mi tutor, Antonio Jesús Rueda Ruíz, por su orientación y paciencia a lo largo del proceso. Su experiencia y conocimientos han sido fundamentales para la culminación exitosa de este proyecto.

Agradezco también a mis profesores y a la EPSJ por haberme proporcionado las herramientas necesarias para desarrollarme académica y profesionalmente durante estos años. Cada lección aprendida ha contribuido de manera significativa a mi formación.

A mi familia, especialmente a mis padres, por su amor incondicional, comprensión y motivación constante. Gracias por siempre creer en mí y confiar en cada decisión que he tomado.

Finalmente, quiero agradecer a mis amigos y seres queridos por su compañía, comprensión y ánimo durante este periodo. Su apoyo ha sido una fuente de energía y motivación en los momentos más difíciles.

FICHA DEL TRABAJO FIN DE TÍTULO	
Titulación	Grado en Ingeniería informática
Modalidad	Proyecto de Ingeniería
Especialidad	General
Mención	General
Idioma	Español
Tipo	Específico
TFT en equipo	No
Autor/a	Eduardo Lomas Recena
Fecha de asignación	16/04/2024

Descripción corta	Este TFG plantea una sencilla aplicación que permita la reserva de instalaciones de pádel y deportes de equipo similares que aporte valor añadido como facilitar la organización de partidos, establecer mecanismos de penalización para usuarios que incumplan las reglas de participación, generar estadísticas de uso de las instalaciones, ranking de usuarios, ofertas y premios para jugadores frecuentes, etc.

NORMAS APLICADAS EN ESTE DOCUMENTO	
LOCALES	
TFT-UJA:2017	Normativa de Trabajos Fin de Grado, Fin de Máster y otros Trabajos Fin de Título de la Universidad de Jaén (Normativa marco UJA aprobada en Consejo de Gobierno)
TFT-EPSJ:2017	Normativa sobre Trabajos Fin de Grado y Fin de Máster en la Escuela Politécnica Superior de Jaén (Normativa EPSJ aprobada en Junta de Escuela)
TFT-EPSJ	Criterios de evaluación y normas de estilo para TFG y TFM de la Escuela Politécnica Superior de Jaén
NACIONALES E INTERNACIONALES	
ISO 2145:1978	Documentación - Numeración de divisiones y subdivisiones en documentos escritos
UNE 50132:1994	Traducción de la ISO 2145
APA 6ª edición	Estilo de referencias y citas de APA (American Psychological Association)

NORMAS UTILIZADAS COMO BASE O REFERENCIA	
NACIONALES	
UNE 157001:2014	Criterios generales para la elaboración formal de los documentos que constituyen un proyecto técnico
UNE 157801:2007	Criterios generales para la elaboración de proyectos de sistemas de información
Estas normas se han utilizado como base o referencia para la inclusión de algunos contenidos y definiciones sobre elaboración de proyectos, entendiendo como proyecto la documentación consensuada entre una empresa y un cliente, que da lugar al perfeccionamiento de un contrato para la elaboración de una obra o la prestación de un servicio. Por consiguiente, no debe esperarse la aplicación de estas normas en cuanto a la completitud de los contenidos ni a la organización de los mismos.	

Contenido

Tabla de contenido

Índice de ilustraciones

Índice de tablas

1 ESPECIFICACIÓN DEL TRABAJO

En este capítulo se presenta la especificación del trabajo, con una estructura y contenidos **inspirados** en los criterios y recomendaciones que establece la norma UNE 157801:2007 - "*Criterios Generales para la elaboración de proyectos de Sistemas de Información*".

A lo largo del documento se utilizarán términos y acrónimos cuya descripción aparecen en el apartado 8 (Definiciones y abreviaturas).

1.1 Introducción

Este Trabajo de Fin de Grado se enfoca en el desarrollo de una aplicación web diseñada para facilitar la reserva de pistas de pádel. La iniciativa surge en respuesta a la creciente demanda de herramientas tecnológicas que simplifiquen la gestión de espacios deportivos y mejoren la experiencia de los usuarios.

En un contexto donde la tecnología ha experimentado un crecimiento acelerado en los últimos años, la creación de aplicaciones web se ha convertido en una necesidad imperativa. La accesibilidad y versatilidad que ofrecen estas plataformas las hacen ideales para abordar una amplia gama de problemas, desde la gestión empresarial hasta a la interacción social y, en este caso, la reserva de instalaciones deportivas.

La necesidad de crear aplicaciones web se ha vuelto cada vez más evidente en un mundo donde la conectividad y la accesibilidad son aspectos fundamentales. Según datos de un informe de Statista sobre el mercado de aplicaciones web en 2023, se estima que el número de aplicaciones web activas ha aumentado en un 25% en los últimos dos años, reflejando la creciente demanda y adopción de este tipo de soluciones.

El objetivo principal del proyecto es desarrollar una solución eficiente y accesible que responda a las necesidades específicas de los aficionados al pádel,

optimizando el proceso de reserva de pistas y mejorando la experiencia general de los usuarios. En las siguientes secciones, se profundizará en los fundamentos teóricos y técnicos que sustentan esta iniciativa, así como en las estrategias y metodologías empleadas para su implementación.

El documento se estructura en torno a varios apartados que abarcan desde la justificación de la relevancia del proyecto hasta el análisis detallado de sus resultados y conclusiones. A lo largo del mismo, se examinarán los desafíos inherentes al desarrollo de la aplicación, así como las oportunidades y beneficios que ofrece su implementación en el ámbito deportivo.

En resumen, este trabajo presenta un esfuerzo por aprovechar el potencial de la tecnología web para mejorar la gestión y accesibilidad de los recursos deportivos, contribuyendo así a enriquecer la experiencia de los usuarios y promover la práctica del pádel de manera más eficiente y satisfactoria.

1.2 Antecedentes y estado del arte

1.2.1 Introducción al pádel

El pádel es un deporte de raqueta que ha ganado una inmensa popularidad en las últimas décadas. Fue creado en México en la década de 1960 por Enrique Corcuera y se caracteriza por ser una combinación de tenis y squash, jugándose en una pista cerrada con paredes que forman parte del juego. Este deporte es especialmente popular en España y América Latina, aunque su presencia se ha expandido a nivel mundial, con miles de pistas y clubes dedicados a su práctica.

Uno de los factores clave del crecimiento del pádel es su accesibilidad. Es un deporte fácil de aprender y jugar, adecuado para personas de todas las edades y niveles de habilidad. Además, su naturaleza social, al jugarse en parejas, fomenta la interacción y el trabajo en equipo, haciendo del pádel una actividad recreativa y competitiva ideal. La Federación Española de Pádel (FEP) y el World Padel Tour

(WPT) han sido fundamentales en la profesionalización y promoción del deporte, organizando torneos que atraen a jugadores y espectadores de todo el mundo.

1.2.2 Contexto de la gestión de reservas

Con el auge del pádel también ha crecido la demanda de servicios que faciliten su práctica. Tradicionalmente, la reserva de pistas de pádel se realizaba mediante métodos manuales, como llamadas telefónicas o visitas en persona a los clubes. Aunque estos métodos fueron funcionales durante un tiempo, presentaban varias limitaciones:

- Errores humanos: La gestión manual de reservas era susceptible a errores, como la doble reserva de pistas o la pérdida de información.
- Falta de disponibilidad en tiempo real: Los jugadores no podían verificar la disponibilidad de pistas en tiempo real, lo que ocasionaba frustraciones y confusiones.
- Ineficiencia y falta de convencimiento: El proceso de reserva manual podía ser tedioso y consumir mucho tiempo tanto para los jugadores como para el personal de los clubes.

La necesidad de una gestión más eficiente llevó al desarrollo de soluciones digitales. Inicialmente centradas en deportes como el tenis y el fútbol, estas soluciones se adaptaron rápidamente al pádel debido a la creciente demanda. Las aplicaciones y plataformas web permitieron a los jugadores reservar pistas de manera más eficiente y cómoda, ofreciendo funcionalidades como la gestión de pagos online, la organización de partidos y torneos, y la consulta de la disponibilidad de pistas en tiempo real.

1.2.3 Evolución de la tecnología en la gestión deportiva

La digitalización ha transformado radicalmente la gestión deportiva en los últimos años. Los primeros sistemas de gestión deportiva eran simples bases de datos que facilitaban la administración de las reservas y los calendarios. Sin

embargo, con el avance de la tecnología, estas soluciones se han vuelto más sofisticadas e integradas, ofreciendo una gama amplia de funcionalidades.

1. **Aplicaciones móviles y plataformas web:** La aparición de aplicaciones móviles y plataformas web han permitido a los usuarios realizar reservas de manera instantánea y desde cualquier lugar. Estas herramientas no solo facilitan la reserva, sino que también proporcionan información en tiempo real sobre la disponibilidad de las pistas y permiten gestionar pagos y organizar eventos.
2. **Sistemas de gestión integral:** Los sistemas modernos de gestión deportiva integran múltiples funcionalidades en una sola plataforma. Estas soluciones permiten la gestión de reservas, el control de accesos, la administración de socios y la organización de torneos, entre otras cosas. Además, pueden integrarse con otros sistemas como el de gestión financiera y CRM (Customer Relationship Manager).
3. **Inteligencia artificial y análisis de datos:** La incorporación de la inteligencia artificial (IA) y el análisis de datos ha abierto nuevas posibilidades para la gestión deportiva. La IA puede predecir la demanda de reservas, optimizar el uso de instalaciones y ofrecer recomendaciones personalizadas para los usuarios. El análisis de datos permite a los administradores de clubes tomar decisiones informadas basadas en el comportamiento y las preferencias de los usuarios.
4. **Internet de las cosas (IoT):** el IoT permite el monitoreo en tiempo real de las instalaciones deportivas. Sensores y dispositivos conectados pueden proporcionar datos sobre la ocupación de las pistas, el estado del equipamiento y el uso de las instalaciones, lo que facilita una gestión más eficiente y el mantenimiento preventivo.
5. **Experiencias personalizadas:** La tecnología ha permitido ofrecer experiencias más personalizadas a los usuarios. Desde aplicaciones que sugieren partidos y compañeros de juego basados en el nivel y las preferencias, hasta programas de fidelización que recompensan la lealtad de los usuarios, la personalización se ha convertido en un componente clave de la gestión deportiva moderna.

La evolución de la tecnología en la gestión deportiva no solo ha mejorado la eficiencia operativa, sino que también ha enriquecido la experiencia de los usuarios, facilitando el acceso y fomentando la participación en actividades deportivas. Estos avances son fundamentales para comprender las soluciones tecnológicas actuales y futuras en la gestión de reservas de pistas de pádel.

1.2.4 Estado actual de las aplicaciones de reserva de pistas de pádel

En los últimos años, el mercado de las aplicaciones de reserva de pistas de pádel ha experimentado un crecimiento significativo, reflejando la popularidad creciente del deporte y la demanda de soluciones eficientes para la gestión de reservas. A continuación, se presentan algunas de las aplicaciones más destacadas en este ámbito, sus características y un análisis comparativo de sus funcionalidades.

- Playtomic
    - Descripción: Playtomic es una de las aplicaciones líderes en la gestión de reservas de pistas de pádel. Además de ofrecer reservas de pistas, Playtomic permite a los usuarios encontrar y unirse a partidos, organizar torneos y conectar con otros jugadores.
    - Características principales:
        - Reserva de pistas en tiempo real.
        - Organización de partidos y torneos.
        - Pagos online y gestión de cuotas.
        - Perfiles de usuario con estadísticas y niveles de habilidad.
        - Integración con clubes para la gestión de instalaciones y socios.
- Pádel Manager
    - Descripción: Pádel Manager es otra aplicación popular que se centra en proporcionar una plataforma integral para jugadores y

clubes de pádel. Ofrece herramientas tanto para la gestión de reservas como para la organización de competiciones.

- Características principales:
  - Reserva de pistas y gestión de disponibilidad.
  - Creación y gestión de ligas y torneos.
  - Pagos online seguros.
  - Sistema de puntuación y clasificación de jugadores.
  - Comunicación directa entre jugadores y clubes.

- Reservaplay
  - Descripción: Reservaplay es una plataforma que permite la reserva de instalaciones deportivas, incluyendo pistas de pádel, a través de una interfaz intuitiva y fácil de usar.
  - Características principales:
    - Reserva online de pistas.
    - Gestión de usuarios y socios.
    - Pagos online y control de accesos.
    - Notificaciones y recordatorios automáticos.
    - Informes y análisis de uso de las instalaciones.

La evolución y el estado actual de las aplicaciones de reserva de pistas de pádel reflejan un mercado dinámico y en crecimiento, con un enfoque constante en mejorar la experiencia del usuario y la eficiencia operativa de los clubes. Sin embargo, la innovación continua y la atención a las necesidades específicas de los usuarios seguirán siendo clave para el desarrollo futuro de estas soluciones tecnológicas.

1.3 Descripción de la situación de partida

Tras tener unos conocimientos previos del estado del arte y contexto del proyecto vamos a comentar la situación de partida. Actuamos como una empresa ficticia encargada de desarrollar una aplicación web para la gestión de reservas de pistas en un club de pádel. En este escenario, hemos simulado una contratación por parte de un club de pádel que necesita una solución tecnológica para mejorar su gestión diaria. El club enfrenta varios desafíos importantes que queremos resolver con nuestra aplicación. Entre estos desafíos se encuentran la gestión fluida de reservas de pistas. Actualmente, el club utiliza una combinación de métodos manuales y herramientas digitales básicas, lo que resulta en ineficiencias operativas y experiencia del usuario mejorable.

Esta iniciativa representa una oportunidad ideal para desarrollar una aplicación que no solo satisfaga las necesidades específicas de este club de pádel, sino que también se convierta en una solución genérica aplicable a otros clubes e instituciones similares. Nuestra meta es crear una plataforma robusta y flexible que pueda ser adaptada y escalada según las necesidades de diferentes clientes en el sector deportivo.

1.3.1 Descripción del entorno actual

El club de pádel que ha contratado nuestros servicios presenta una infraestructura y organización que se detalla a continuación, con el fin de comprender mejor los aspectos que se verán afectados por la nueva aplicación.

Infraestructura Tecnológica Actual

1. Sistemas de Reserva Manuales y Digitales Básicos
    - Métodos manuales: El club utiliza libros de registros físicos y llamadas telefónicas para gestionar las reservas de pistas, lo cual es propenso a errores humanos y dobles reservas.

- Herramientas digitales básicas: Algunos registros se llevan en hojas de cálculo y aplicaciones de calendario sin integración, lo que dificulta la actualización en tiempo real y la coordinación efectiva

Experiencia de Usuario

1. Socios del Club
    - Proceso de reserva: Los socios deben contactar al club por teléfono o en persona para reservar pistas, lo cual puede ser inconveniente y lento.
    - Acceso a Información: Limitado acceso a horarios de disponibilidad de pistas.
2. Administradores
    - Carga administrativa: Elevada debido a la necesidad de gestionar manualmente las reservas y coordinar las actividades del club.
    - Comunicación: Dificultades para mantener una comunicación efectiva y actualizada con los usuarios.

1.3.2 Resumen de las deficiencias y carencias identificadas

La principal deficiencia que se va a superar con el desarrollo de la aplicación web es la gestión manual que se hace de los miembros del club de pádel y de las reservas de las pistas lo que puede llegar a malentendidos y confusiones y a la pérdida de datos de gran importancia.

Por otro lado, los usuarios de la aplicación tendrán mayor facilidad para visualizar en todo momento la disponibilidad de las pistas y poder realizar las reservas con la mayor comodidad y eficiencia posible, incrementando la satisfacción de los usuarios.

Otra carencia que tienen los sistemas de reservas actuales es que necesitas obligatoriamente un grupo de cuatro personas para jugar, por lo que, si falta alguna persona, todos se quedan sin jugar. Con la implementación de eventos no privados

queremos fomentar la participación de personas a jugar con otras para conocer gente y poder jugar sin tener un grupo de cuatro personas disponibles desde el principio.

1.4 Objetivos del trabajo

El presente trabajo tiene como objetivo principal el desarrollo de una aplicación web de reserva de pistas, enfocada en ser altamente escalable y desarrollada utilizando metodologías ágiles para asegurar un proceso iterativo eficiente y adaptable a las necesidades del usuario.

En cuanto a los objetivos técnicos del proyecto nos encontramos con los siguientes:

1. **Desarrollo del backend con Spring Framework:** La elección de un framework es debido a que hoy día son fundamentales para el desarrollo de aplicaciones web ya que nos ofrecen una estructura preconstruida formada por distintos componentes que podemos usar en el lado del servidor, contribuyendo a un mejor rendimiento, escalabilidad y seguridad en la aplicación que vayamos a construir. El uso de Spring Framework en el backend asegura una implementación de componentes robustos y bien estructurados, facilitando la escalabilidad y mantenibilidad del sistema. Además, debido a una previa experiencia con el framework, creo que es la mejor opción para el desarrollo de mi aplicación web.
2. **Desarrollo del frontend con React:** React es una biblioteca de JavaScript desarrollada por Facebook que se utiliza para construir interfaces de usuario interactivas y de una sola página (Single Page Applications). Utiliza un enfoque basado en componentes que facilita la creación de interfaces de usuario reutilizables. Los beneficios clave de React incluyen el rendimiento optimizado debido a su enfoque en el DOM virtual, la facilidad de mantenimiento gracias a su modularidad, y una

curva de aprendizaje relativamente baja en comparación con otros frameworks. El desarrollo del frontend con React permitirá al usuario una experiencia dinámica y eficiente, con actualizaciones rápidas y visualmente atractivas.

3. **Comunicación entre backend y frontend mediante una API:** La comunicación entre el backend y el frontend se llevará a cabo a través de una API RESTful (Representational State Transfer). Una API RESTful es un conjunto de principios de arquitectura que utiliza HTTP como un protocolo de comunicación y está basado en recursos que pueden ser accedidos y manipulados mediante operaciones estándar HTTP (GET, POST, PUT, DELETE). Esta arquitectura facilita la separación clara entre backend y frontend, permitiendo una integración flexible y eficiente. La API RESTful asegurará que las solicitudes del frontend se gestionen de manera segura y eficaz, garantizando una experiencia de usuario fluida y consistente.

1.5 Requisitos iniciales

Para cumplir con el objetivo del proyecto la aplicación web tiene que satisfacer los siguientes requisitos:

A nivel de aplicación:

- La aplicación debe ser responsive, es decir, debe adaptarse correctamente a diferentes tamaños de pantallas y dispositivos.
- La aplicación debe incluir un sistema de registro y autenticación de usuarios.
- La aplicación debe tener definido un sistema de roles para diferenciar un usuario normal de un administrador.

A nivel de usuario:

- El usuario debe poder visualizar la disponibilidad de las pistas.
- El usuario debe poder reservar una pista que esté libre.
- El usuario debe poder unirse a un evento que no sea privado.
- El usuario debe poder solicitar la unión a un evento que sea privado

- El usuario debe poder cancelar la reserva de la pista cumpliendo con un plazo de tiempo.
- El usuario debe cancelar la solicitud de unión a un evento.
- El usuario debe poder aceptar las solicitudes a un evento privado siempre y cuando este sea el anfitrión del evento.
- El usuario debe poder visualizar las personas que hay en el evento.
- El usuario debe poder acceder a los perfiles de otros usuarios.
- El usuario debe poder editar los datos de su propio perfil.

A nivel de administrador:

- El administrador debe poder crear, editar y eliminar las pistas que estarán disponibles para reservar.
- El administrador debe poder generar un horario semanal con la disponibilidad para cada día de la semana y en cada pista independientemente.
- El administrador debería poder sancionar a un usuario que no ha acudido a un evento.

1.6 Alcance

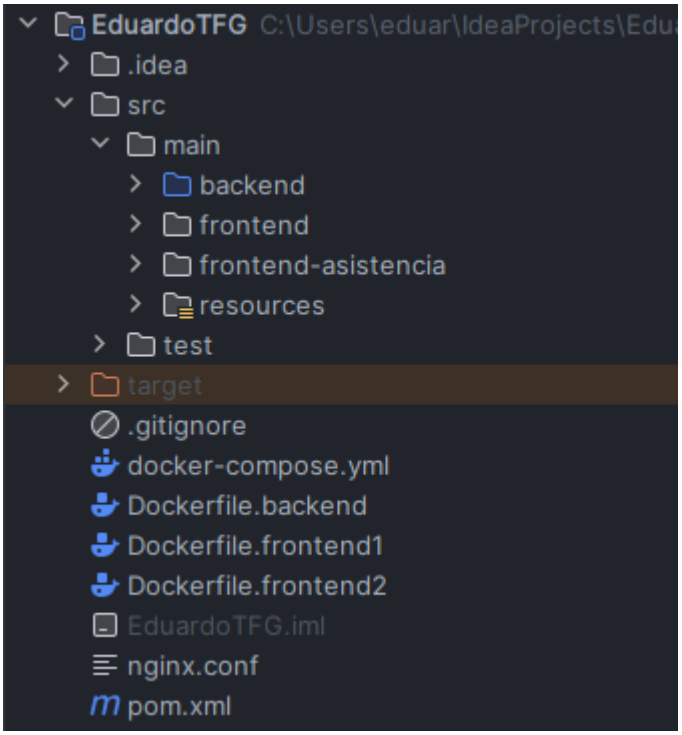
En el siguiente apartado se definirá hasta dónde llega el trabajo, describiendo claramente lo que se incluye en los resultados:

Esta aplicación está pensada para pequeñas instalaciones que tengan varias pistas que estén disponible para reserva para los usuarios en el día a día, pero esto puede ser ampliable a varios deportes y mucha cantidad de pistas. Además, se podría incluir un apartado para creación de torneos.

Debido al tiempo limitado que disponemos para la realización del proyecto no se pueden incluir ese tipo de funcionalidades. No obstante, nuestro proyecto es una aplicación intuitiva para el usuario que incluye la operativa básica necesaria para

hacer reservas de pistas y unirse a eventos de otra gente. Por otra parte, el administrador tendrá su propio panel de control en el que manejar todas las entidades que influyen en la aplicación y que intentará mejorar el orden y la experiencia de los usuarios. También hay otro frontend para que los participantes puedan confirmar su asistencia a los eventos, y que está pensada para una tablet o dispositivo similar disponible en las propias instalaciones. De esta manera el administrador podrá hacer el seguimiento del uso de las instalaciones y en su caso, si alguien no ha asistido, imponer las sanciones correspondientes.

1. **Proyecto:** Código fuente de toda la aplicación web desarrollada en los que destacan las siguientes carpetas dentro de src/main/:
    a. frontend: Código desarrollado para el frontend de la aplicación general.
    b. frontend-asistencia: Código desarrollado para el frontend de la aplicación para registrar las asistencias a los eventos.
    c. backend: Código desarrollado para el backend de la aplicación.



2. **Documentación:** Es el propio documento, en el que se encuentra todo sobre el trabajo desarrollado.

1.7 Hipótesis y restricciones

El TFG se define como una asignatura de 12 créditos, lo que supone que la duración total del proyecto será de 300 horas, incluyendo todas las etapas del ciclo de vida, con la excepción del mantenimiento. Por consiguiente, la principal restricción aplicable es la limitación de la duración del trabajo.

1.8 Descripción de la solución propuesta

La aplicación web propuesta permite abarcar todas las funcionalidades esenciales en una aplicación de este tipo, lo que permite satisfacer todas las necesidades del usuario para una gran cantidad organizaciones con características similares. Las características significativas de esta solución son las siguientes:

- Gestión de reservas de pistas: La aplicación permitirá a los usuarios visualizar la disponibilidad de las pistas y la creación y cancelación de los eventos.
- Herramientas de gestión para administradores: Los administradores dispondrán de una interfaz en la que podrán gestionar los usuarios, visualizar estadísticas, modificar horarios de disponibilidad de las pistas, crear/eliminar/modificar nuevas pistas y tarifas.
- Diseño responsive: La aplicación deberá tener un diseño responsive para conseguir una buena experiencia de usuario tanto en dispositivos móviles como en ordenadores.
- Diseño compuesto: Diferenciaremos la separación de los componentes que forman la aplicación. Tanto frontend como backend están diferenciados, lo que permite una mejor organización.
- Sistema de roles: Se implementa un sistema de roles que permitirá separar funcionalidades entre usuarios y administradores.
- Sistema de confirmación de asistencia: Se dispone de un frontend distinto al principal que servirá como método de registrar la asistencia de los usuarios a los eventos reservados.

En conclusión, la aplicación web propuesta ofrece una solución integral que cubre todas las funcionalidades esenciales para la gestión eficiente de reserva de pistas de pádel, lo que la convierte en una herramienta valiosa para organizaciones que requieren este tipo de sistema. Al incorporar herramientas de gestión para administradores, un diseño responsive, una arquitectura compuesta, un sistema de roles y un mecanismo de confirmación de asistencia, la aplicación garantiza una experiencia de usuario optimizada y una administración efectiva.

1.9 Tecnologías utilizadas

Vamos a describir las tecnologías utilizadas en el proyecto según a la parte que pertenecen.

1.9.1 Tecnologías frontend

Para el desarrollo del frontend apostamos por usar un conjunto de tecnologías que permitiesen crear una interfaz de usuario sencilla y eficiente. El conjunto de tecnologías para el frontend es:

- **React:** Su elección se debe que permite la reutilización de componentes, su eficiencia en la gestión del DOM ya que minimiza las operaciones de actualización, lo que se traduce en una interfaz más rápida y reactiva. Otra de las ventajas es que cuenta con una gran comunidad, por lo que hay una gran cantidad de bibliotecas y herramientas complementarias que ayudan con la resolución de algunos problemas.
- **JavaScript:** Al ser el lenguaje nativo de los navegadores web, garantiza una excelente compatibilidad y rendimiento en cualquier plataforma. Además, JavaScript permite la manipulación directa del DOM, facilitando la creación de experiencias de usuario fluidas y receptivas. Gracias a su versatilidad, JavaScript se integra perfectamente con React, permitiendo una interacción rápida y eficiente entre los componentes de la interfaz y la lógica de la aplicación.

- **React Router DOM:** Librería de React usada para la gestión de la navegación y las rutas dentro de la aplicación, permitiendo una navegación fluida entre diferentes vistas.
- **Booststrap / React Booststrap:** Es un framework de CSS utilizado con su versión adaptada para React que nos permite crear interfaces con un estilo moderno y responsive.
- **Chart.js / React Chart.js 2:** Son librerías utilizadas para la creación de gráficos interactivos en la vista del administrador, lo que facilita la visualización de datos dentro de la aplicación.
- **Axios:** Es una biblioteca para manejar solicitudes HTTP en aplicaciones React debido a su interfaz intuitiva y su capacidad para simplificar el manejo de peticiones y respuestas. Ofrece un manejo de errores más robusto que “fetch”, soporte para interceptores y la capacidad de cancelar solicitudes en curso, lo que mejora la eficiencia y la robustez de la aplicación. Además, Axios permite configuraciones globales que facilitan la integración de la autenticación y configuraciones comunes, haciendo que el código sea más limpio y mantenible.
- **Otros:** Otras bibliotecas y componentes complementarios que me han ayudado al desarrollo del frontend son:
    - React Datepicker: Selector de fecha, útil para formularios y gestión de eventos.
    - Date-fns: Biblioteca para la manipulación y formateo de fechas en JavaScript, conocida por su rendimiento y facilidad de uso.
    - FontAwesome: Utilizado para integrar iconos escalables en la interfaz de usuario, mejorando la experiencia visual.

1.9.2 Tecnologías backend

Para el desarrollo del backend, optamos por utilizar un conjunto de tecnologías que nos permitieran construir una infraestructura robusta, escalable y segura. Las tecnologías usadas en el backend son:

- **Spring Boot:** Debido a su configuración simplificada, su amplio ecosistema que proporciona acceso a una gran cantidad de herramientas y bibliotecas para tareas comunes como seguridad y acceso a bases de datos. También se tiene en cuenta su despliegue simplificado en cualquier entorno, su gran comunidad con una amplia documentación y su integración con otras tecnologías populares como Hibernate.
- **Java:** Es una elección ideal para el desarrollo del backend ya que es un lenguaje maduro y ampliamente utilizado en la industria, lo que garantiza una base sólida y confiable para el desarrollo de aplicaciones empresariales. Además, es el lenguaje nativo de Spring Boot, lo que asegura una integración perfecta y un aprovechamiento completo de las características de este marco. Por otro lado, ofrece un rendimiento robusto y es altamente escalable, lo que lo hace adecuado para aplicaciones de gran envergadura y alta demanda.
- **Spring Security:** Usado por su capacidad para proporcionar un sistema de autenticación y autorización robusto y altamente configurable. Además, además su integración fluida con Spring Boot y la posibilidad de personalizar cada aspecto del sistema de seguridad permiten una gestión precisa de los permisos y accesos, asegurando que solo los usuarios autorizados puedan interactuar con nuestros servicios críticos.
- **MySQL:** Elegida debido a su fiabilidad, rendimiento y facilidad de uso, características que lo convierten en una opción ideal para aplicaciones empresariales. Es altamente compatible con Spring Boot y Spring Data JPA, lo que facilita su integración en nuestro marco tecnológico, permitiendo un manejo eficiente de las transacciones y consultas.
- **API REST:** Permitiendo la comunicación entre backend y frontend. Esta arquitectura facilita la interacción entre ambos componentes a través de peticiones HTTP, permitiendo al frontend consumir los servicios del backend de forma sencilla y escalable.

1.9.3 Otras tecnologías

Este apartado está reservado para aquellas tecnologías que han sido utilizadas para el desarrollo del producto pero que no tienen nada que ver con el backend y frontend.

- **Git:** Es un sistema de control de versiones que facilita la gestión de cambios en el código, asegurando que cada modificación esté documentada y pueda ser revertida si es necesario, lo que minimiza riesgos y errores.
- **GitHub:** Actúa como un repositorio remoto para almacenar y versionar nuestro código con Git. Además de permitir almacenar el código en la nube, proporciona herramientas integradas para revisión de código, manejo de issues y gestión de proyectos, lo que mejora la calidad del desarrollo.
- **Docker:** Docker es una plataforma de software que permite crear, desplegar y gestionar aplicaciones dentro de contenedores. Un contenedor es una unidad estandarizada de software que empaqueta el código de una aplicación junto con todas sus dependencias, bibliotecas y configuraciones necesarias para que funcione de manera consistente en cualquier entorno.
- **Trello:** Es una herramienta de gestión de proyectos y tareas basada en la web que utiliza un enfoque visual para organizar el trabajo. Está diseñada para ayudar a individuos y equipos a planificar, coordinar y realizar un seguimiento de sus tareas y proyectos de manera eficiente.
- **IntelliJ IDEA:** Es un entorno de desarrollo integrado (IDE) para programación. Es ampliamente utilizado para el desarrollo de aplicaciones en varios lenguajes de programación, pero es especialmente conocido por su soporte robusto para java.

1.10 Metodología de desarrollo de software

En la actualidad, el uso de metodologías de desarrollo ágil se ha consolidado como una práctica estándar en la gestión de proyectos de software. Este enfoque

responde a la necesidad de adaptarse rápidamente a los cambios y entregar valor de manera incremental y continua. Las metodologías ágiles permiten una mejor respuesta a los cambios en los requisitos y una gestión más dinámica del proyecto, lo que es crucial en un entorno variable como el desarrollo de software.

Comparación de Metodologías Ágiles

Existen diversas metodologías ágiles que se han popularizado en la industria, entre las que destacan Scrum y Kanban. A continuación, se realiza una comparación detallada entre ambas.

- Estructura
    - Scrum: Se organiza en ciclos de trabajo llamados “sprints”, que suelen tener una duración fija de entre 2 y 4 semanas. Cada sprint culmina con la entrega de un incremento funcional del producto.
    - Kanban: Se basa en el flujo continuo de trabajo sin ciclos predefinidos, utilizando un tablero visual para gestionar las tareas a través de diferentes estados del flujo de trabajo.
- Roles
    - Scrum: Define roles específicos como el Scrum Master, el Product Owner y el equipo de desarrollo. Estos roles son esenciales para mantener la estructura y el flujo de trabajo dentro de los sprints.
    - Kanban: No requiere roles específicos, lo que proporciona una mayor flexibilidad en la gestión del trabajo. Todos los miembros del equipo pueden contribuir de manera equitativa en la gestión de tareas.
- Ventajas
    - Scrum: Proporciona una estructura clara con procesos bien definidos, lo que facilita la coordinación en equipos y la planificación a corto plazo.

- Kanban: Ofrece una visualización clara y constante del estado de las tareas, lo que facilita la identificación de cuellos de botella y permite una adaptación rápida a los cambios.

- Desventajas
  - Scrum: Puede resultar rígido y difícil de implementar en proyectos pequeños individuales, debido a la necesidad de cumplir con las ceremonias y roles establecidos.
  - Kanban: Al carecer de una estructura formal y de iteraciones, puede no ser adecuado para proyectos que requieren una planificación temporal más rigurosa y predeterminada.

Justificación de la metodología utilizada

Tras analizar ambas metodologías, se ha decidido adoptar una combinación de características de Scrum y Kanban para el desarrollo de este TFG. La elección de esta metodología mixta se basa en las siguientes razones:

1. Estructura y Flexibilidad: Los principios de Scrum se utilizarán para dividir el proyecto en iteraciones, lo que facilitará la planificación y permitirá establecer metas claras a corto plazo. Kanban complementará esta estructura proporcionando una herramienta visual que facilite la gestión diaria entre tareas y la adaptación a cambios en los requisitos.
2. Visualización del trabajo: El tablero Kanban permitirá una visualización clara del estado del trabajo en todo momento, asegurando que las tareas prioritarias se aborden lo antes posible.
3. Adaptación rápida a cambios: La flexibilidad de Kanban permitirá realizar ajustes en el plan de trabajo durante cada sprint, adaptándose rápidamente a cualquier cambio en los requisitos o prioridades.
4. Optimización del Flujo de Trabajo: Al combinar la estructura iterativa de Scrum con los límites de trabajo en progreso de Kanban, se busca evitar la sobrecarga y garantizar que las tareas se completen de manera eficiente antes de iniciar nuevas actividades.

Justificación de no usar una metodología clásica

Las metodologías de desarrollo de software tradicionales, como el modelo en cascada, han sido ampliamente utilizadas en el pasado. Sin embargo, para un proyecto como el TFG, presentan varias limitaciones que las hacen menos adecuadas en comparación con las metodologías ágiles:

1. Rigidez en la planificación: El modelo en cascada sigue un enfoque secuencial, donde cada fase del desarrollo debe completarse antes de avanzar a la siguiente. Esto limita la capacidad de adaptarse a cambios en los requisitos una vez que se ha avanzado en el desarrollo, lo que puede ser problemático en un proyecto donde las necesidades pueden evolucionar.
2. Larga duración entre entregas: En las metodologías clásicas, la entrega de un producto funcional suele ocurrir al final del ciclo de desarrollo, lo que retrasa la posibilidad de realizar ajustes. En un TFG, es crucial recibir una retroalimentación constante para asegurar que el proyecto de alinee con los objetos definidos.
3. Poca flexibilidad: Las metodologías tradicionales no facilitan la incorporación de cambios sobre la marcha, ya que cualquier modificación en los requisitos pueden requerir la revisión de fases anteriores del proyecto.
4. Riesgo del fracaso: Debido a la falta de iteraciones y entregas incrementales, los problemas pueden detectarse en etapas muy avanzadas del desarrollo, cuando ya es demasiado tarde para realizar correcciones efectivas sin comprometer el proyecto.

Conclusión

La adopción de una metodología ágil mixta que combine elementos de Scrum y Kanban proporciona la estructura necesaria para planificar y ejecutar el proyecto de manera eficiente, al tiempo que permite la flexibilidad para adaptarse a cambios y optimizar el flujo del trabajo. Este enfoque es claramente más adecuado para un proyecto individual como el TFG, donde la gestión efectiva del tiempo y la

adaptabilidad son esenciales, en contraste con las limitaciones impuestas por las metodologías clásicas.

1.11 Análisis de riesgos

Los posibles riesgos que hay en el proyecto y que podrían afectar al desarrollo del mismo son los siguientes:

1. **Problemas de integración entre el frontend y backend**
   1.1. Descripción: La integración entre el frontend y el backend podría no ser tan sencilla como se esperaba, lo que podría llevar a problemas de comunicación entre ambos componentes.
   1.2. Probabilidad: Media
   1.3. Impacto: Alto
   1.4. Plan de contingencia: Realizar pruebas de integración tempranas y frecuentes. Utilizar herramientas de testing automatizadas para verificar la comunicación entre componentes.
2. **Dependencia de librerías de terceros**
   2.1. Descripción: El uso de librerías o frameworks que podrían no cumplir con las expectativas o no ser compatibles con futuras actualizaciones de React o Spring.
   2.2. Probabilidad: Media
   2.3. Impacto: Medio
   2.4. Plan de contingencia: Evaluar múltiples alternativas antes de elegir una librería. Mantenerse actualizado con las versiones de librerías y planificar tiempo para posibles migraciones.
3. **Problemas de rendimiento**
   3.1. Descripción: La aplicación web podría tener problemas de rendimiento, como tiempos de carga elevados o ineficiencias en la gestión de datos.

3.2. Probabilidad: Baja

3.3. Impacto: Alto

3.4. Plan de contingencia: Realizar pruebas de carga desde etapas tempranas. Optimizar el código y la base de datos.

4. Inexperiencia en alguna tecnología

4.1. Descripción: Una posible falta de experiencia en alguna tecnología crítica (React, Spring, bases de datos), podría resultar en una curva de aprendizaje mucho más empinada lo que conlleva un progreso más lento y más errores.

4.2. Probabilidad: Alta

4.3. Impacto: Alto

4.4. Plan de contingencia: Dedicar tiempo a formación y aprendizaje antes de abordar las tareas más complejas. Búsqueda de ayuda en foros, comunidades y tutoriales.

1.12 Estimación del tamaño y esfuerzo

Ya que el presente proyecto es un TFG, no existen restricciones de tipo económico, sino de tipo temporal (un número aproximado de horas). Por consiguiente, los cálculos de tamaño del proyecto están supeditados el tiempo disponible. En cuanto al esfuerzo, se dispone de tan un solo efectivo (la persona autora del trabajo).

1.13 Planificación temporal

En esta sección se presenta la planificación temporal del proyecto, teniendo en cuenta que el desarrollo sigue un enfoque ágil utilizando Kanban. A diferencia de otras metodologías ágiles como Scrum, Kanban no se basa en sprints o ciclos de tiempo fijo, sino en un flujo continuo donde las tareas se manejan de manera flexible y adaptable según las prioridades.

El desarrollo del proyecto abarca desde el mes de febrero hasta el mes de septiembre. Hay que tener en cuenta que durante el mes de junio no se trabajó por tema de exámenes. La primera fase del proyecto fue la de análisis de requisitos y diseño en la cual se definieron todos los requisitos mínimos que debería tener el proyecto. La fase de desarrollo del proyecto se realizó prácticamente al mismo tiempo que la fase de pruebas para ir comprobando que todo estaba bajo control. De la misma manera la documentación se realizó progresivamente con la fase de desarrollo y pruebas.

Una vez conocidas las fases que componen la realización del proyecto, vamos a delimitar las franjas de tiempo en las que se han ido realizando cada una de las fases para poder organizarlo y componer un diagrama de Gantt para visualizarlo con mayor claridad.

- Análisis de requisitos y diseño: 01/02/2024 – 29/02/2024
- Desarrollo: 01/03/2024 – 30/08/2024
- Pruebas: 15/03/2024 – 30/08/2024
- Documentación: 01/02/2024 – 30/08/2024

Aquí se muestra un diagrama de Gantt en el que se puede apreciar de manera más visual la planificación temporal seguida para el proyecto.

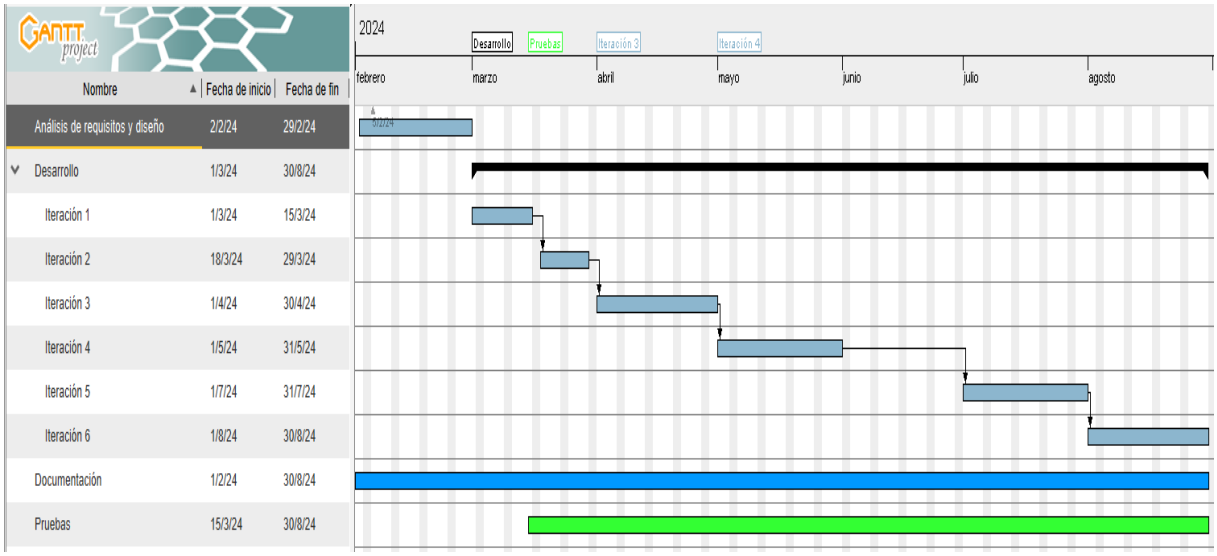


Ilustración 1: Ilustración Diagrama de Gantt

1.14 Presupuesto

Como cualquier otro proyecto, este necesita de un presupuesto para estimar el coste que va a tener. Para ello vamos a tener dividirlos en apartados

1.14.1 Hardware

El equipo usado para el desarrollo del proyecto es un MSI Prestige 15 que tuvo un coste de 1350€. Teniendo en cuenta que el tiempo de vida medio para un portátil es de 4 años y que el proyecto dura unos 7 meses, se debe calcular el coste del portátil en proporción a la duración del proyecto. Teniendo en cuenta esto, y haciendo el siguiente cálculo:

Coste = Precio total * Duración del proyecto/Vida útil del portátil.

Coste = 1350 * 7 / 48 = 196,88€

Por tanto, el uso del portátil para el desarrollo del proyecto costaría unos 196,88€

1.14.2 Sueldos

En primera instancia se necesitará un analista de software, cuyo sueldo medio en España es de 32.270€ al año, lo que equivale a 18,87€ por hora. En el proyecto el analista ha participado en la primera parte del proyecto que duraba un mes, en la que suponemos un total de 40 horas * 18,87€/hora da un total de 754,8€.

Para todo el desarrollo del proyecto se necesitará un desarrollador fullstack, cuyo sueldo medio en España es de 28.500€ al año, lo que equivale a 16,41€ por hora. Dado que en el proyecto trabaja durante todo el desarrollo se estima que trabaja durante unas 60 horas al mes durante los 6 meses que está desarrollándose la aplicación nos sale un total de 60h * 6 meses * 16,41€ = 5.907,60€

Para hacer las pruebas al software se necesita un tester cuyo sueldo medio en España es de 27.500€ al año, que equivalen a 15,55€ la hora. Dado que trabaja durante 5 meses y medio con unas 20 horas al mes, calculamos de la siguiente manera: 20h * 5,5 meses ^15,55€ = 1.710,5€.

1.14.3 Licencias

En cuanto a licencias se ha necesitado una licencia de Windows 11 pro cuyo coste es de 89,98€ y licencia para IntelliJ IDEA cuyo precio es de 16,90€ al mes, por lo que el coste total es de 101,4€.

Como conclusión se expone una tabla en la que se resumen todos los gastos que definen el presupuesto del proyecto. Se agrega un 10% del coste total para afrontar costes indirectos y no previstos en un principio, por lo que el presupuesto queda así:

Medios	Coste (€)
Hardware	196,88
Sueldos	754,8 + 5.907,6 + 1.710,5 = 8.372,9
Licencias	89,98 + 101,4 = 191,38
Costes indirectos	876,12
Total	9.637,28

Una vez analizados todos los costes a tener en cuenta para el desarrollo del proyecto nos queda un presupuesto total de 9.637,28€.

1.14.4 Mecanismos de control de calidad

El aseguramiento de la calidad en la redacción del trabajo implica la verificación de la completitud (falta de omisiones), la integridad de la documentación, así como una redacción clara, concisa y entendible por todos los participantes e interesados en dicho trabajo.

Al estar el trabajo desarrollado por una sola persona y verificado por un/a tutor/a, no es necesario llevar un documento de control de edición y revisión de la documentación. De esta forma, se han utilizado mecanismos básicos para la verificación de la integridad y completitud, que incluyen un control de versiones a nivel de documento y un control sencillo de la trazabilidad de especificaciones y requisitos.

2 DISEÑO INICIAL

El objetivo de este apartado es definir nuestra idea principal de la aplicación, incluyendo sus principales requisitos. Esto abre el camino que deberemos seguir para llegar a nuestro objetivo final y será nuestra guía. Cabe recordar que seguiremos una metodología de desarrollo ágil por lo que nuestro proyecto irá evolucionando con el paso del tiempo y las nuevas ideas y cambios que vayan apareciendo durante el desarrollo.

Como idea general del proyecto, queremos queríamos hacer una aplicación web de reserva de pistas de pádel que tuviese los siguientes requisitos generales:

- Una interfaz sencilla e intuitiva que permitiese al usuario registrarse, iniciar sesión y visualizar las reservas de las correspondientes pistas de pádel.
- Una vista para el administrador que le permitiese controlar todos los eventos y usuarios de la aplicación, así como poner horarios, tarifas y todo lo correspondiente con las pistas.

2.1 Especificaciones del sistema

Una vez visto el diseño inicial de nuestro proyecto, en este apartado especificaremos detalladamente los requisitos tanto funcionales como no funcionales obtenidos.

Para ello se listarán los requisitos y entre paréntesis se indicará la prioridad que tiene dicho requisito. Las diferentes prioridades son las siguientes:

- **Alta:** Estos requisitos son críticos para el éxito del proyecto y deben ser implementados y cumplidos antes que cualquier otro.
- **Media:** Estos requisitos son importantes, pero no críticos. Mejoran la experiencia del usuario o añaden funcionalidades adicionales que son valiosas, pero no esenciales para la operación básica del sistema.

- **Baja:** Estos requisitos son deseables, pero no esenciales para la funcionalidad principal del sistema. Son características que agregarían valor o mejorarán la usabilidad, pero no son necesarias para el funcionamiento básico.

2.1.1 Requisitos funcionales

Los requisitos funcionales con especificaciones que definen las funciones y comportamientos que un sistema debe cumplir para satisfacer las necesidades del usuario. Los requisitos funcionales de nuestro proyecto son los siguientes:

- **Registrar usuario (Alta):** El usuario podrá registrarse en la aplicación para poder tener acceso a todas las funcionalidades de la aplicación.
- **Iniciar sesión (Alta):** El usuario podrá iniciar sesión en la aplicación con unas credenciales previamente registradas que le darán acceso a la aplicación.
- **Visualizar pistas (Alta):** El usuario podrá ver la disponibilidad de las pistas en el menú principal de la aplicación.
- **Reservar pista (Alta):** El usuario podrá reservar una pista que este libre un día y hora determinado.
- **Unirse a evento (Media):** Un usuario podrá unirse a un evento que previamente ha sido reservado por otra persona.
- **Enviar solicitud (Baja):** Un usuario podrá enviar una solicitud a un evento que se ha creado de manera privada para poder unirse.
- **Cancelar solicitud (Baja):** Un usuario que ha realizado una solicitud puede cancelarla con un tiempo de antelación predefinido.
- **Aceptar solicitud (Baja):** El anfitrión del evento privado podrá aceptar solicitudes para que otros usuarios se unan al evento.
- **Cancelar reserva (Media):** Un usuario podrá cancelar la reserva hecha para salirse del evento con una antelación determinada en la aplicación.

- **Ver usuarios (Baja):** Un usuario podrá ver los usuarios que componen el evento y acceder a su perfil.
- **Ver perfil (Media):** Un usuario podrá acceder a su perfil para ver sus datos.
- **Editar perfil (Media):** Un usuario podrá editar datos de su perfil que hayan cambiado o quiera modificar.
- **Mostrar historial (Baja):** Un usuario podrá acceder a su historial de partidos mostrando cuál finalmente se ha jugado y cuál queda por jugar.
- **Sistema de roles (Alta):** Se debe implementar un sistema de roles para que un administrador tenga acceso privado a su panel de control.
- **Mostrar estadísticas (Baja):** El administrador tendrá una vista para visualizar algunas estadísticas interesantes.
- **Sancionar usuario (Media):** El administrador podrá sancionar a un usuario sin jugar por un período de tiempo.
- **Manejo de pistas (Media):** El administrador podrá crear/editar/eliminar las pistas que hay en el sistema.
- **Manejo de horarios (Media):** El administrador podrá editar los horarios en los que cada pista estará disponible.
- **Manejo de tarifas (Baja):** El administrador podrá crear tarifas para según el tipo de pista, turno de juego y durante periodos de tiempo.
- **Manejo de eventos (Media):** El administrador tendrá acceso a las reservas que se realizan y podrá eliminar eventos creados por usuarios por error.
- **Manejo de usuarios (Baja):** El administrador tendrá acceso a los perfiles de los usuarios y podrá modificar sus datos.
- **Otorgar bonos (Baja):** El administrador tendrá la capacidad de otorgar bonos que el usuario podrá utilizar para realizar reservas gratuitamente. Esto está pensado para que los usuarios que más reservas realicen

durante un mes sean premiados de alguna manera y haya incentivos por jugar a parte de por diversión.

- **Confirmar asistencia (Baja):** El usuario podrá confirmar su asistencia al evento a la hora de entrar a las instalaciones mediante un simple frontend introduciendo su pin.

2.1.2 Requisitos no funcionales

Los requisitos no funcionales especifican las características cualitativas y las restricciones del sistema, como el rendimiento, la seguridad, la usabilidad y la escalabilidad. Los requisitos no funcionales determinan cómo debe comportarse el sistema en términos de calidad y eficiencia.

- **Seguridad (Alta):** La aplicación tendrá un sistema de autenticación seguro y los datos sensibles como la contraseña serán encriptados.
- **Rendimiento (Alta):** La aplicación debe tener un tiempo de respuesta válido para que la experiencia del usuario no se vea degradada.
- **Usabilidad (Media):** La interfaz debe ser simple e intuitiva para cualquier tipo de usuario, permitiendo la reserva de pistas y su visualización de manera rápida y sencilla. Además, tiene que tener un diseño responsive para poder ser usada en gran variedad de dispositivos.

2.2 Análisis y diseño del sistema

Una vez detallados todos los requisitos iniciales, incluido el diseño preliminar de clases, se procederá a presentar el análisis y diseño del sistema, incluyendo diagramas que complementen y ayuden a entender su funcionamiento.

2.2.1 Análisis del sistema

En el apartado de análisis del sistema se elaboran los diagramas que representan el comportamiento del sistema, las interacciones entre sus

componentes proporcionando una visión integral de cómo el sistema debe funcionar en su entorno real.

A continuación, se muestra el diagrama de casos del sistema con los distintos actores y sus interacciones con el sistema.

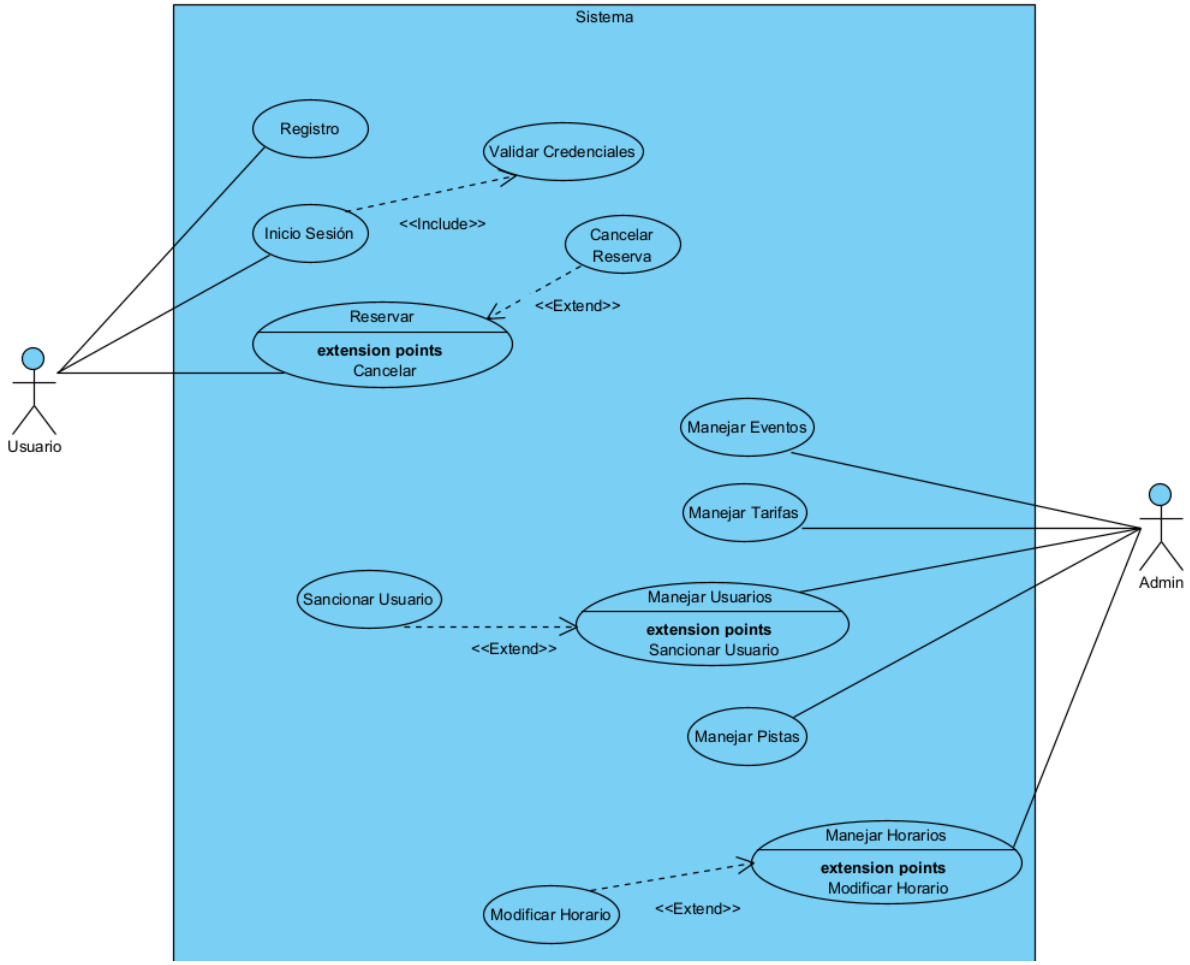


Ilustración 2: Ilustración Diagrama Casos de Uso

Continuaremos con la definición de cada caso de uso mediante las siguientes tablas descritas a continuación:

Campo	Descripción
ID	Identificador único del caso de uso (ej: CU01)
Nombre	Nombre del caso de uso (ej: Reservar pista)
Actor(es)	Actores involucrados (ej: Usuario)
Descripción	Breve descripción de la funcionalidad (ej: Permite al usuario reservar una pista)

Flujo Principal	Pasos principales del proceso (ej: 1. Seleccionar pista, 2. Confirmar pista)
Excepciones	Manejo de errores (ej: Mostrar mensaje de error si la reserva falla)
Requisitos Especiales	Requisitos adicionales (ej: Reserva en menos de 5 segundos)

Una vez explicados los campos que contiene la tabla procedemos con la exposición de cada caso de uso con su correspondiente diagrama de secuencia para representar la interacción entre los distintos componentes del sistema.

Registro

Campo	Descripción
ID	CU01
Nombre	Registro
Actor(es)	Usuario
Descripción	Permite al usuario registrarse en el sistema
Flujo Principal	1. El usuario accede al formulario de registro 2. El usuario rellena los campos requeridos 3. El usuario confirma el registro
Excepciones	Mostrar mensaje de error si el usuario ya está registrado o si los datos no son válidos
Requisitos Especiales	La contraseña debe cumplir con los requisitos mínimos de seguridad

Tabla 1: Caso de Uso Registro

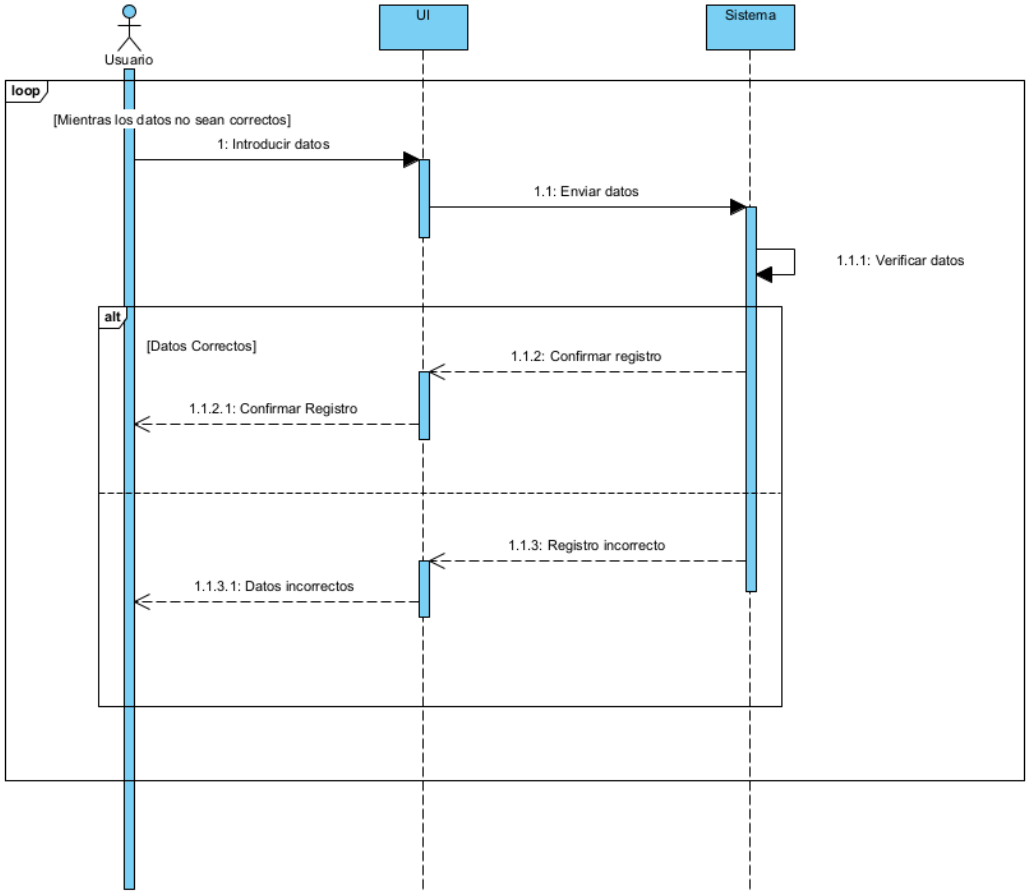


Ilustración 3: Ilustración Diagrama de Secuencia Registro

Inicio Sesión

Campo	Descripción
ID	CU02
Nombre	Inicio Sesión
Actor(es)	Usuario
Descripción	Permite a un usuario registrado iniciar sesión en la aplicación utilizando sus credenciales
Flujo Principal	1. El usuario accede a la página de inicio de sesión 2. El usuario introduce sus credenciales 3. El sistema verifica las credenciales 4. Si las credenciales son correctas, el usuario es autenticado y redirigido a la página principal
Excepciones	Mostrar mensaje de error si las credenciales son

	incorrectas o si el usuario no está registrado
Requisitos Especiales	La sesión debe expirar después de un tiempo de inactividad o al cerrar el navegador

Tabla 2: Caso de Uso Inicio Sesión

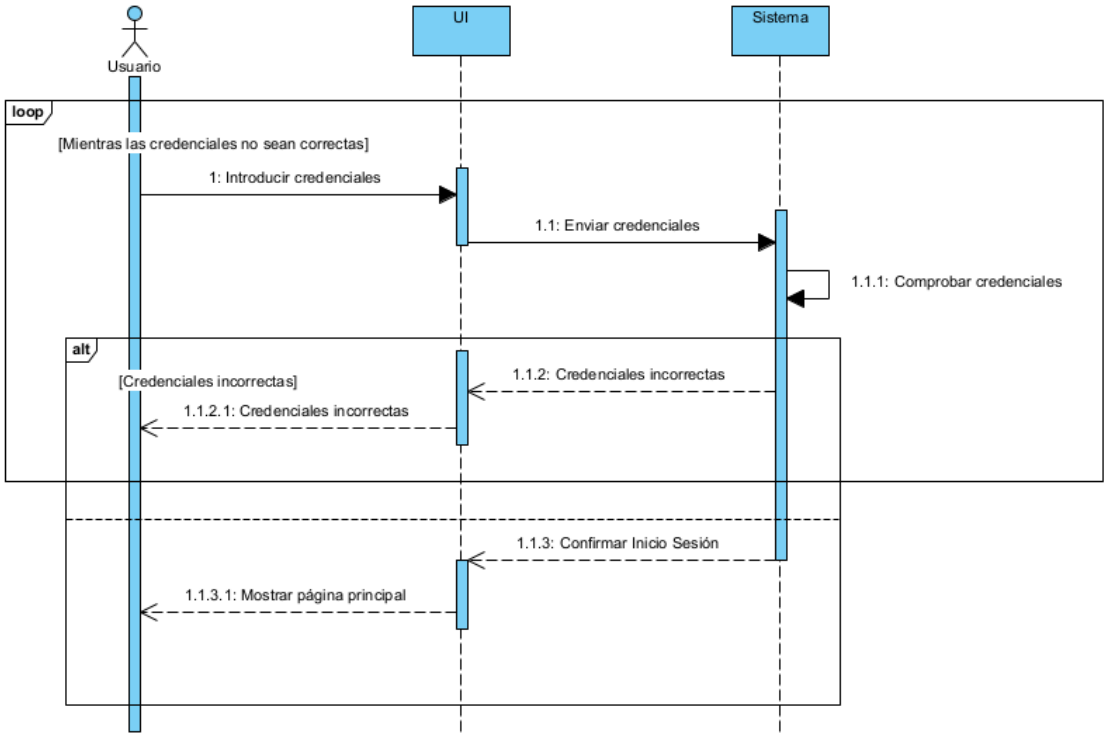


Ilustración 4: Ilustración Diagrama de Secuencia Inicio Sesión

Reservar

Campo	Descripción
ID	CU03
Nombre	Reservar
Actor(es)	Usuario
Descripción	Permite a un usuario registrado reservar una pista de pádel para una fecha y hora específicas
Flujo Principal	1. El usuario accede al sistema de reservas 2. Selecciona la fecha, hora y pista deseada 3. Confirma la reserva 4. El sistema verifica la disponibilidad y confirma la reserva al usuario
Excepciones	Mostrar mensaje de error si la pista ya está reservada

	o si hay problemas en el sistema
Requisitos Especiales	La reserva debe confirmarse en tiempo real y actualizar la disponibilidad de inmediato

Tabla 3: Caso de Uso Reservar

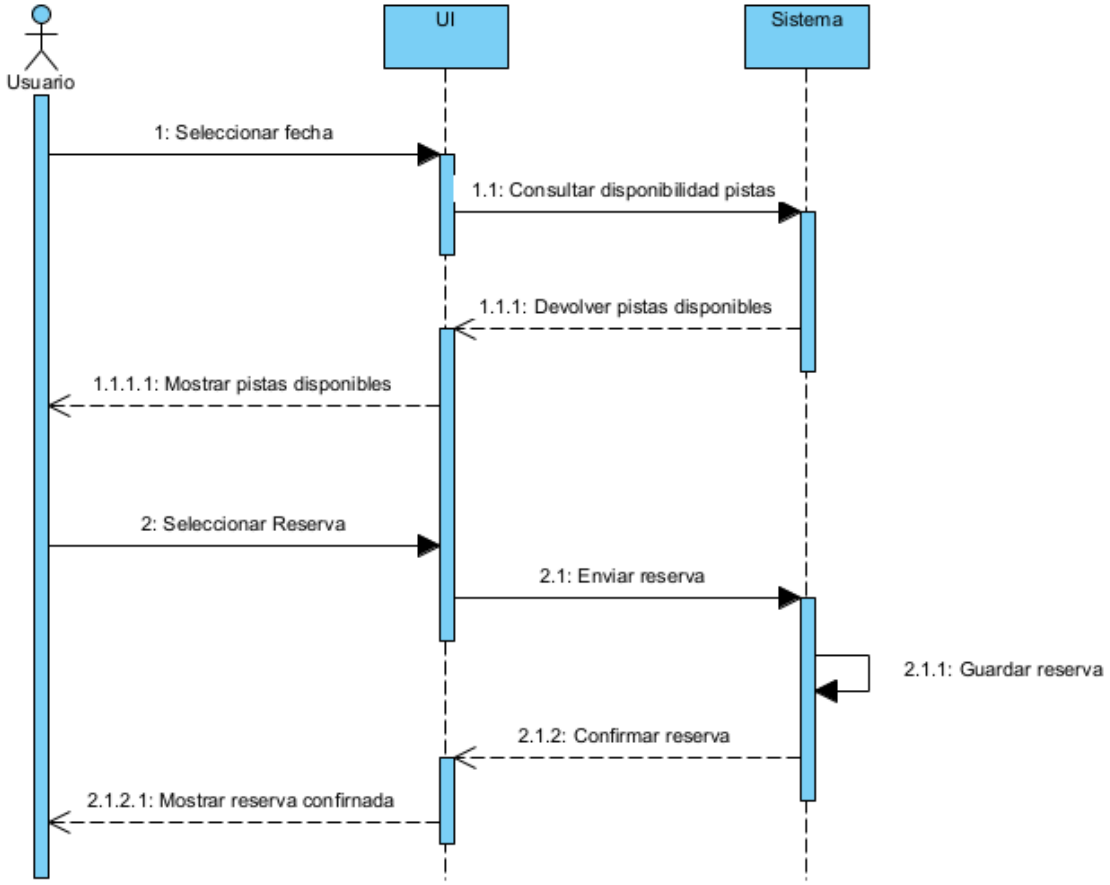


Ilustración 5: Ilustración Diagrama de Secuencia Reservar

Cancelar Reserva

Campo	Descripción
ID	CU04
Nombre	Cancelar Reserva
Actor(es)	Usuario
Descripción	Permite a un usuario registrado cancelar una reserva de pista previamente realizada

Flujo Principal	1. El usuario accede a la sección de reservas 2. Selecciona la reserva que desea cancelar 3. Confirma la cancelación 4. El sistema procesa la cancelación y actualiza la disponibilidad de la pista
Excepciones	Mostrar mensaje de error si la cancelación no es posible
Requisitos Especiales	La cancelación debe ser reflejada inmediatamente en el sistema para permitir nuevas reservas

Tabla 4: Caso de Uso Cancelar Reserva

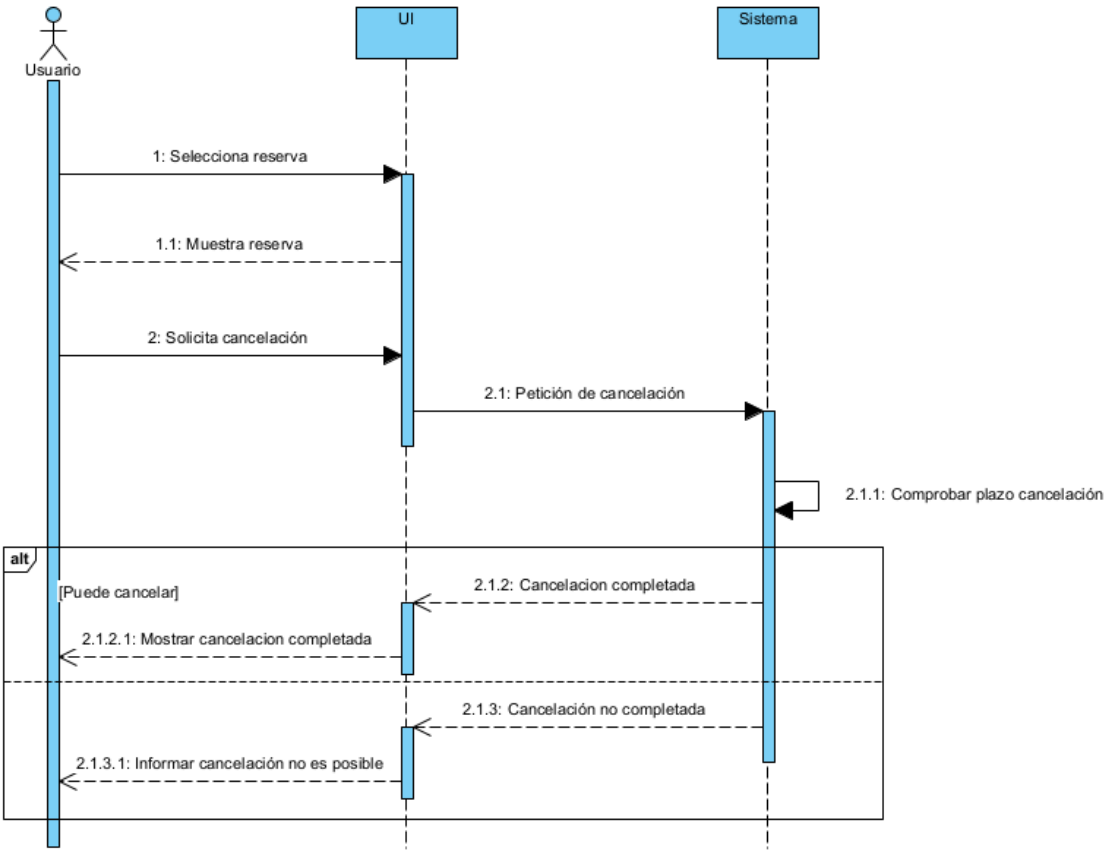


Ilustración 6: Ilustración Diagrama de Secuencia Cancelar Reserva

Manejar Eventos

Campo	Descripción
ID	CU05
Nombre	Manejar Eventos

Actor(es)	Administrador
Descripción	Permite al administrador eliminar y editar eventos en el sistema
Flujo Principal	1. El administrador accede al panel de gestión de eventos 2. Selecciona el evento 3. Confirma la acción 4. El sistema guarda los cambios y actualiza la lista de eventos
Excepciones	Mostrar mensaje de error si los datos son inválidos o si no se puede eliminar un evento debido a reservas existentes
Requisitos Especiales	Las acciones deben actualizarse inmediatamente en el sistema

Tabla 5: Caso de Uso Manejar Eventos

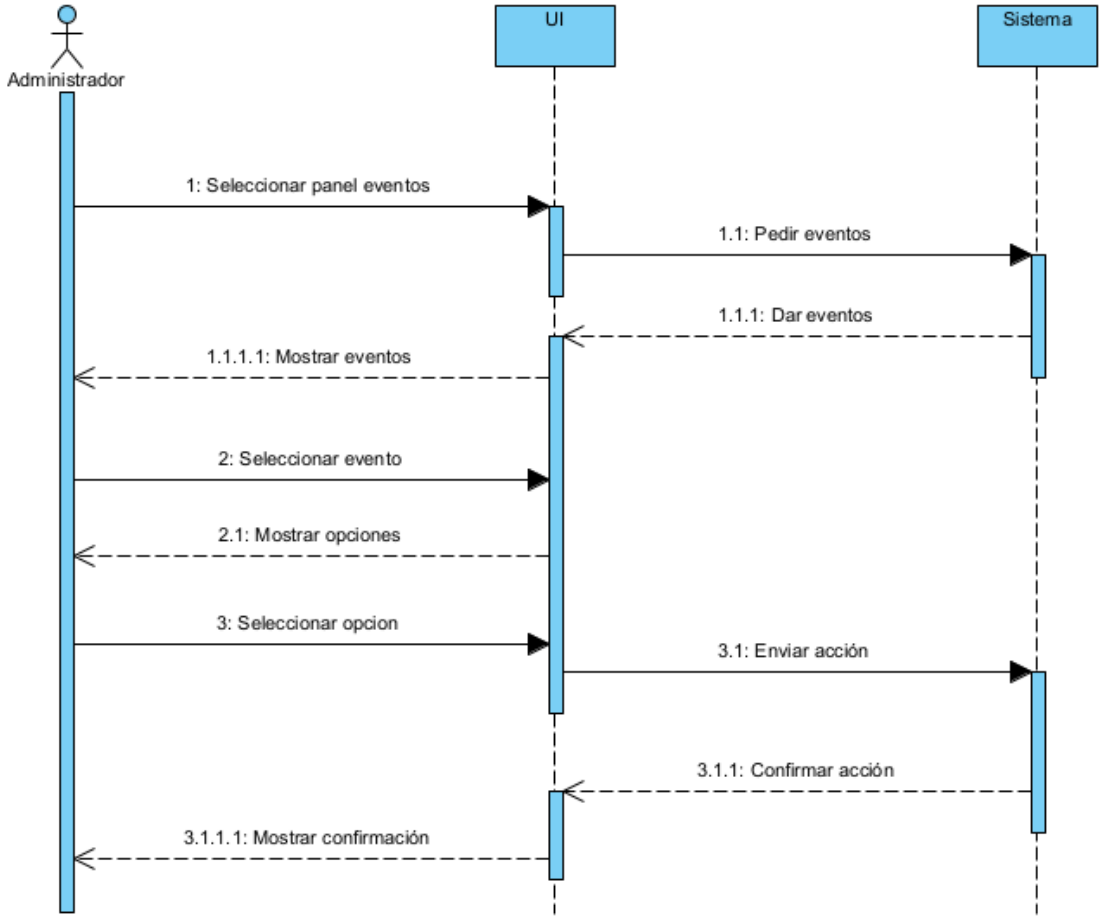


Ilustración 7: Ilustración Diagrama de Secuencia Manejar Eventos

Manejar Tarifas

Campo	Descripción
ID	CU06
Nombre	Manejar Tarifas
Actor(es)	Administrador
Descripción	Permite a un administrador editar, crear y eliminar tarifas de las reservas
Flujo Principal	1. El administrador accede al panel de gestión de tarifas 2. Selecciona una tarifa existente para editar o eliminar, o bien crear una nueva tarifa 3. Confirma la acción 4. El sistema guarda los cambios y actualiza la lista de tarifas
Excepciones	Mostrar mensaje de error si los datos de la tarifa son inválidos
Requisitos Especiales	Las modificaciones de tarifas deben aplicarse de inmediato a las reservas futuras

Tabla 6: Caso de Uso Manejar Tarifas

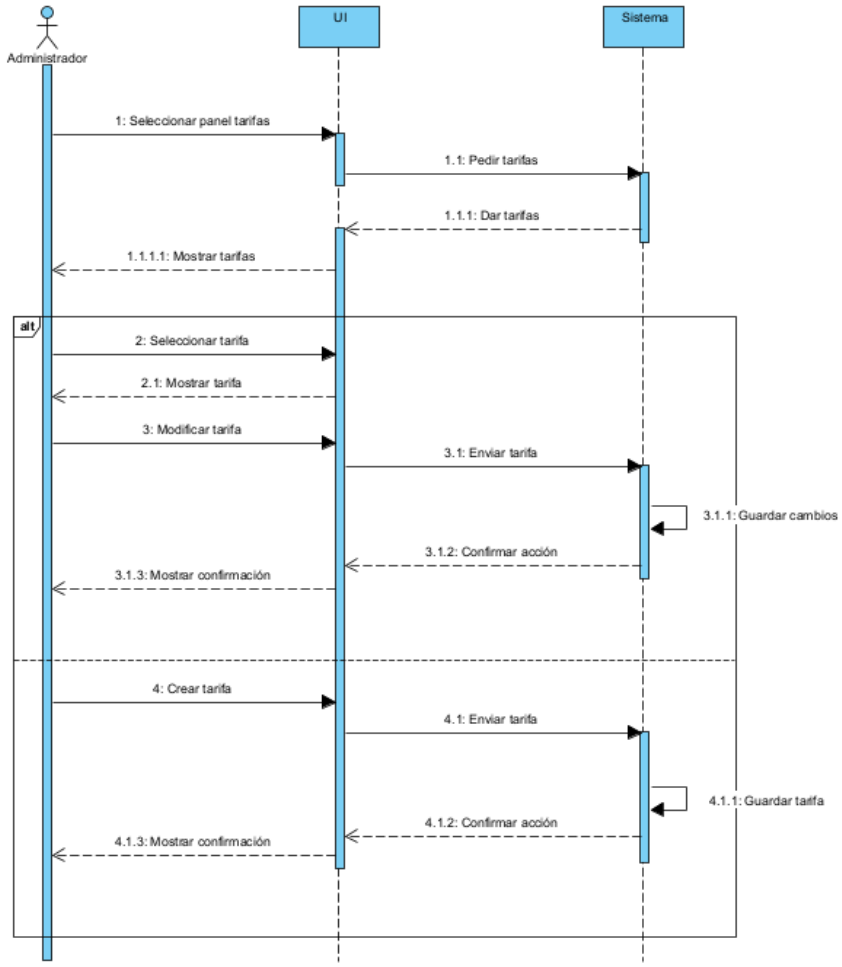


Ilustración 8: Ilustración Diagrama de Secuencia Manejar Tarifas

Manejar Usuarios

Campo	Descripción
ID	CU07
Nombre	Manejar Usuarios
Actor(es)	Administrador
Descripción	Permite al administrador gestionar los usuarios registrados, editando o eliminando
Flujo Principal	1. Accede al panel de gestión de usuarios 2. Selecciona, edita o elimina un usuario 3. Modifica detalles del usuario 4. Confirma la acción 5. El sistema guarda los cambios y actualiza la lista de usuarios

Excepciones	Mostrar error si los datos del usuario son inválidos.
Requisitos Especiales	Cambios deben ser reflejados inmediatamente

Tabla 7: Caso de Uso Manejar Usuarios

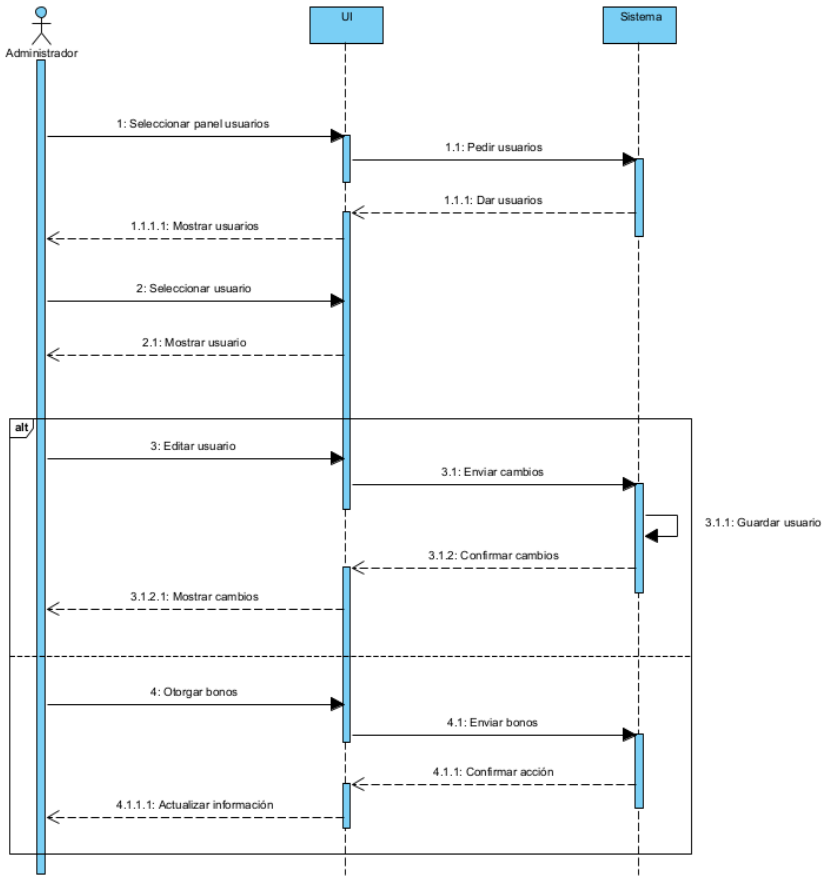


Ilustración 9: Ilustración Diagrama de Secuencia Manejar Usuarios

Sancionar Usuario

Campo	Descripción
ID	CU08
Nombre	Sancionar Usuario
Actor(es)	Administrador
Descripción	Permite al administrador aplicar sanciones a usuarios por incumplimientos de las normas
Flujo Principal	1. El administrador accede al perfil del usuario 2. Selecciona la opción de sancionar 3. Introduce los detalles de la sanción (motivo, duración)

	4. Confirma la sanción 5. El sistema registra la sanción y notifica al usuario sancionado
Excepciones	Mostrar mensaje de error si los datos de la sanción son inválidos
Requisitos Especiales	La sanción debe estar registrada y reflejarse de inmediato en el sistema

Tabla 8: Caso de Uso Sancionar Usuario

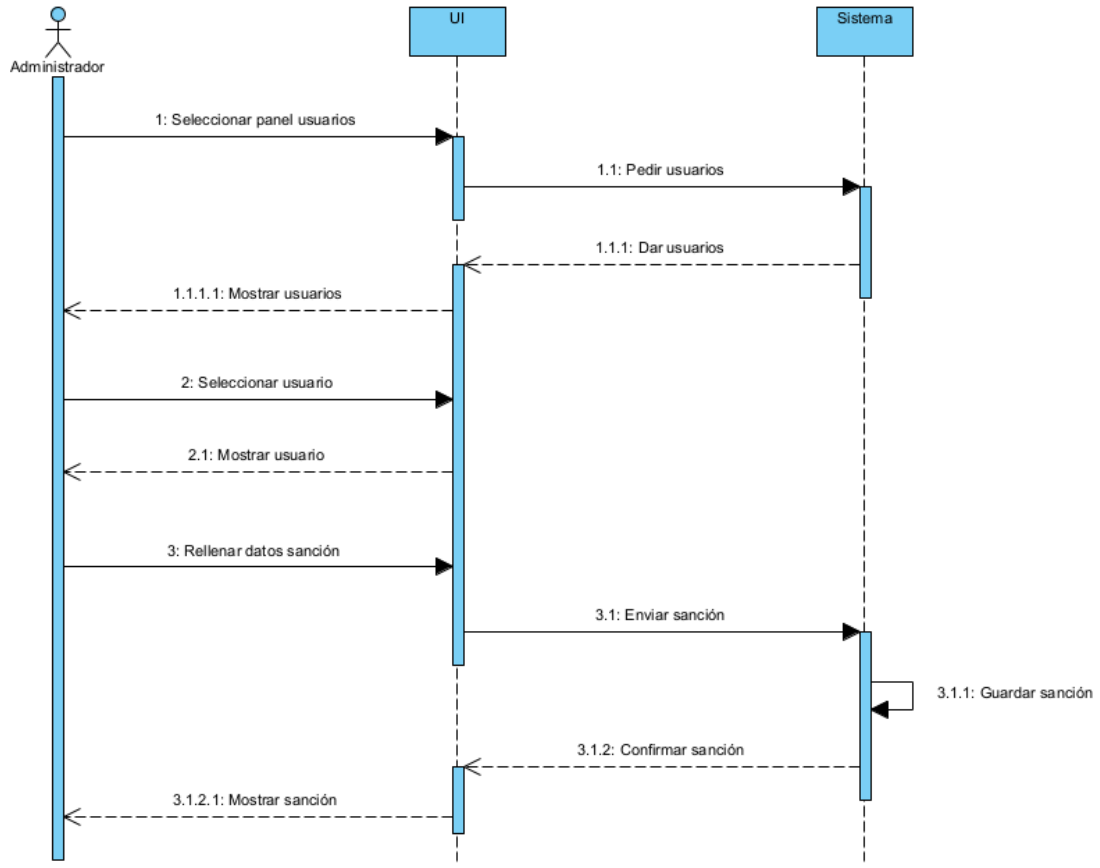


Ilustración 10: Ilustración Diagrama de Secuencia Sancionar Usuario

Manejar Pistas

Campo	Descripción
ID	CU09
Nombre	Manejar Pistas
Actor(es)	Administrador
Descripción	Permite al administrador crear, editar y eliminar las

	pistas de pádel disponibles en el sistema
Flujo Principal	1. Accede al panel de gestión de pistas 2. Selecciona una pista existente para editar o eliminar, o bien crea una nueva pista 3. Introduce o modifica los detalles de la pista 4. Confirma la acción 5. El sistema guarda los cambios actualiza la lista de pistas
Excepciones	Mostrar mensaje de error si los datos de la pista son inválidos
Requisitos Especiales	Los cambios en las pistas deben reflejarse de inmediato en el sistema para evitar conflictos

Tabla 9: Caso de Uso Manejar Pistas

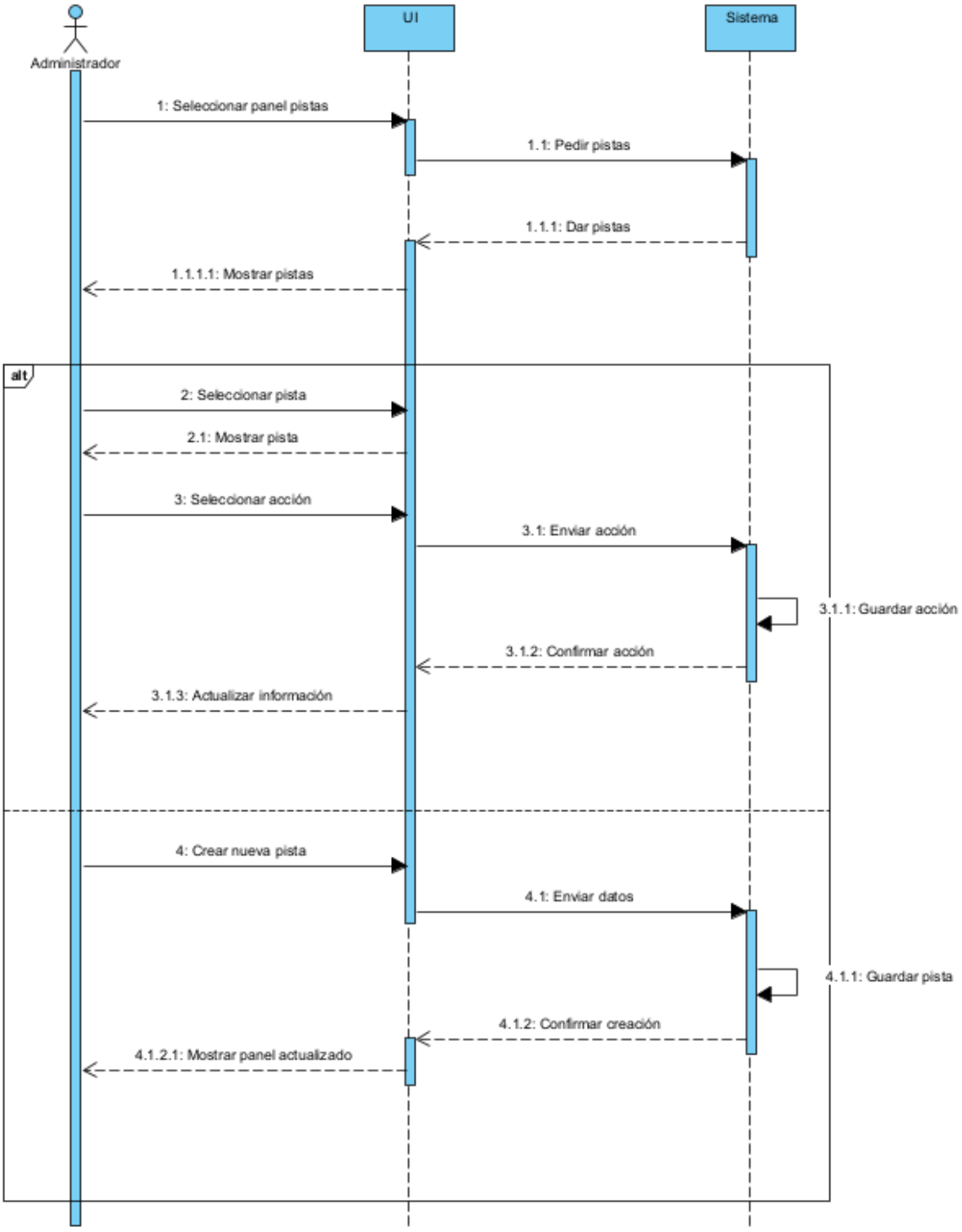


Ilustración 11: Ilustración Diagrama de Secuencia Manejar Pistas

Manejar Horarios

Campo	Descripción
ID	CU10
Nombre	Manejar Horarios
Actor(es)	Administrador
Descripción	Permite al administrador modificar y gestionar los horarios de disponibilidad para las pistas de pádel
Flujo Principal	1. Accede al panel de gestión de horarios 2. Selecciona una pista para mostrar su horario 3. Introduce o modifica los detalles del horario (día, horas) 4. Confirma la acción 5. El sistema guarda los cambios y actualiza los horarios disponibles
Excepciones	Mostrar mensaje de error si los datos del horario son inválidos
Requisitos Especiales	Los cambios en los horarios deben reflejarse de inmediato en el sistema para evitar conflictos

Tabla 10: Caso de Uso Manejar Usuarios

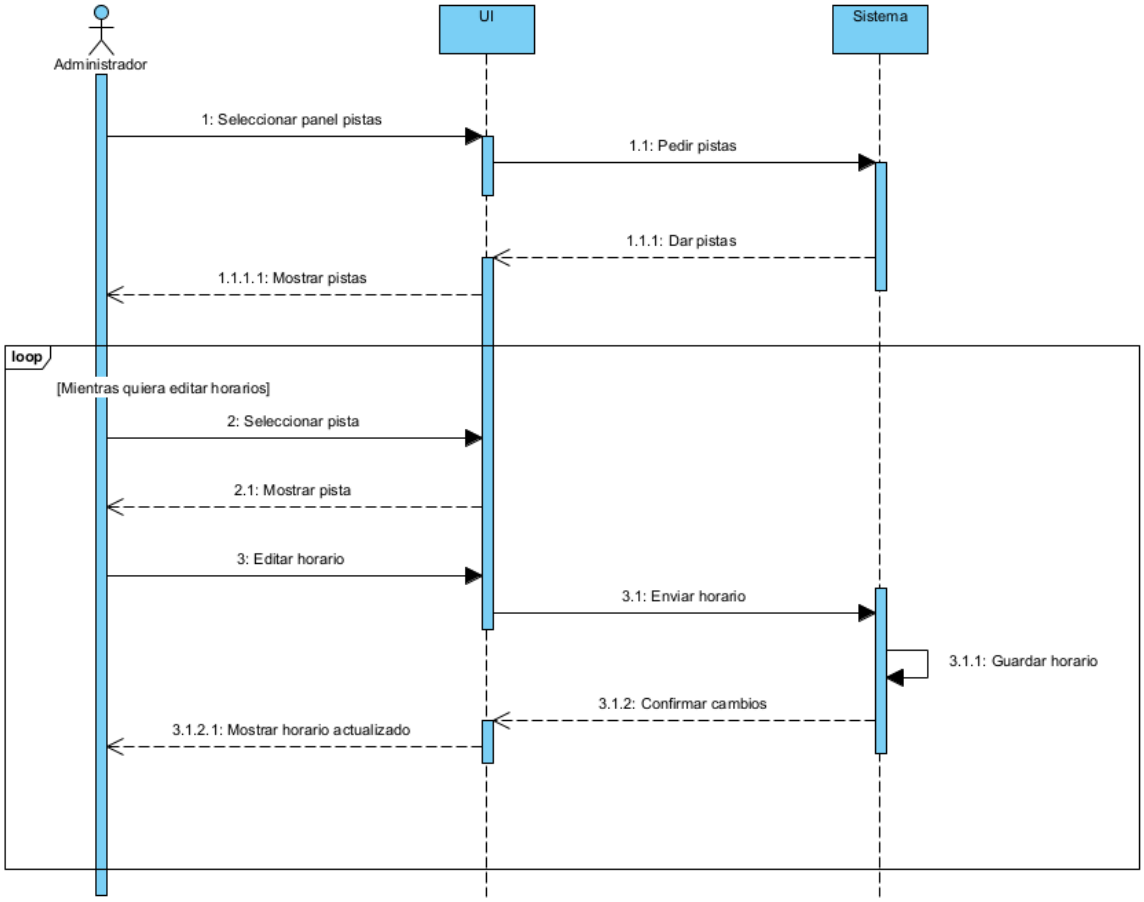


Ilustración 12: Ilustración Diagrama de Secuencia Manejar Horarios

1.1.1 Diseño del sistema

A continuación, se detallará la arquitectura y la comunicación entre los distintos componentes de la aplicación. La arquitectura del sistema se basa en una estructura de tres capas: frontend, backend y base de datos, cada una con un papel específico y crítico en el funcionamiento global del sistema.

Frontend: Desarrollado con React, proporciona la interfaz de usuario interactiva desde la que los usuarios pueden realizar reservas, gestionar sus perfiles y acceder a su historial.

Backend: Implementado con Spring, maneja la lógica de negocio y las operaciones del sistema. El frontend se comunica con este a través de peticiones a una API RESTful. El backend recibe las solicitudes del frontend, procesa la información y, si es necesario, interactúa con la base de datos MySQL para obtener

o modificar los datos. En resumen, actúa como intermediario entre el frontend y la base de datos, asegurando que las solicitudes del usuario sean procesadas adecuadamente y que los datos se mantengan consistentes y actualizados.

Base de datos: Almacena toda la información crítica del sistema, como los usuarios, reservas, horarios, etc.

Este diseño asegura una separación clara de responsabilidades, optimiza la gestión de datos y facilita la escalabilidad y mantenimiento del sistema. Aquí muestro una imagen para apreciar visualmente la arquitectura del sistema y la interacción entre sus capas.

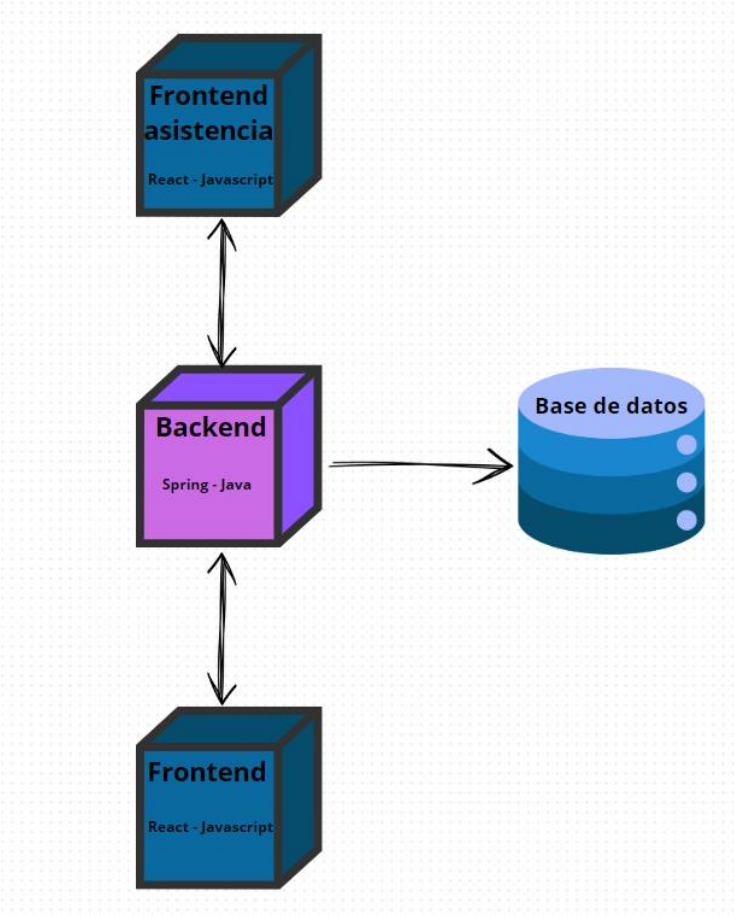


Ilustración 13: Ilustración Diseño del Sistema

El diagrama de clases que finalmente queda tras la resolución del proyecto que veremos cómo se va desarrollando en el siguiente apartado es el siguiente:

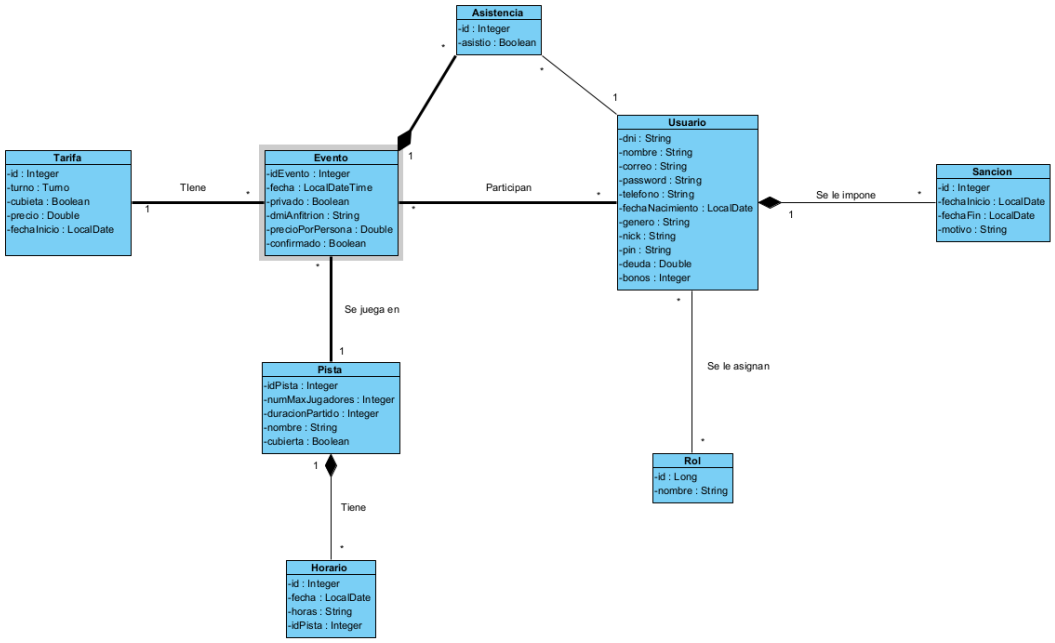


Ilustración 14: Ilustración Diagrama de Clases Final

Aunque más tarde describiremos más detalladamente las clases con la base de datos, aquí haremos una pequeña descripción de las relaciones entre ellas.

Al usuario, una de las clases fundamentales, se le asignan roles y se le pueden imponer sanciones. Por otro lado, también se relaciona con Evento, que es otra de las clases imprescindibles, ya que los usuarios participan en los eventos. Por cada usuario que se une a un evento, se crea una asistencia para dicho evento. Cada evento se asocia con una pista que a su vez se compone de unos horarios. Para cada evento existe una tarifa que se asocia a sus características como hora, tipo de pista, etc.

Una vez visto el diagrama de clases final, vamos a detallar el diseño de la base de datos, con sus diferentes entidades. La base de datos es un componente fundamental del sistema, ya que almacena y organiza toda la información necesaria para el funcionamiento de la aplicación web. El diseño de la base de datos se ha llevado a cabo cuidadosamente para garantizar la integridad, coherencia y eficiencia

en el manejo de los datos. Las entidades que forman parte de esta base de datos representan los principales objetos y relaciones del dominio del sistema.

Cada entidad corresponde a una tabla que almacena un conjunto de datos relacionados. Las relaciones entre estas entidades, como las asociaciones de unos a muchos o muchos a muchos, se han definido utilizando claves foráneas y tablas intermedias según sea necesario. Este diseño no solo asegura que la información esté bien estructurada, sino que también facilita las operaciones de consulto y manipulación de datos, permitiendo que el sistema responda eficientemente a las solicitudes. Aquí se expone un diagrama para ver las tablas que componen la base de datos y sus relaciones:

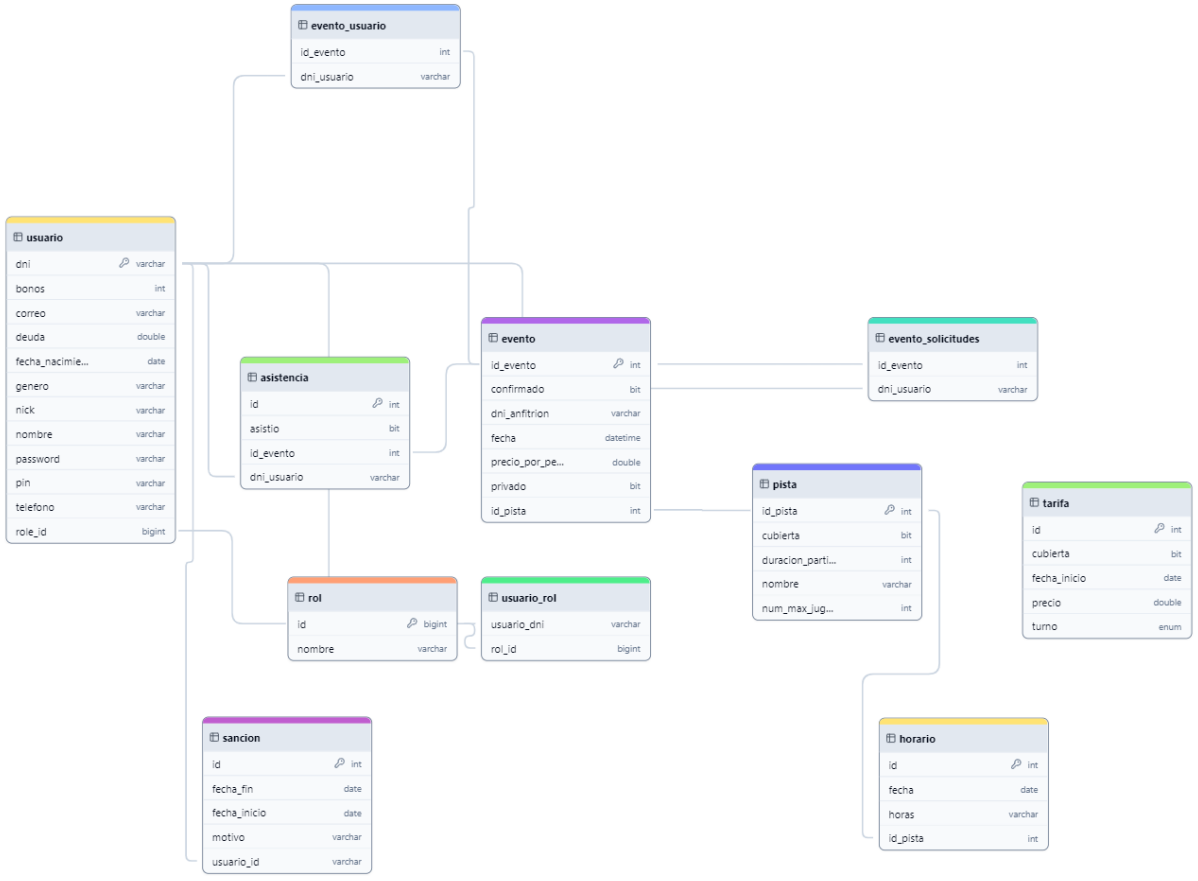


Ilustración 15: Ilustración Diagrama Diseño de Base de Datos

A continuación, procederemos a detallar cada una de las tablas que componen la base de datos.

Usuario

Campo	Tipo
Dni	VARCHAR
Bonos	INT
Correo	VARCHAR
Deuda	DOUBLE
Fecha_nacimiento	DATE
Genero	VARCHAR
Nick	VARCHAR
Nombre	VARCHAR
Password	VARCHAR
Pin	VARCHAR
Teléfono	VARCHAR

Tabla 11: Tabla Usuario Base de Datos

Esta tabla es uno de los componentes fundamentales para garantizar una experiencia de usuario personalizada y segura. Esta tabla se encarga de almacenar la información clave de cada usuario registrado en la plataforma, permitiendo gestionar sus credenciales y datos.

Evento

Campo	Tipo
Id_evento	INT
Confirmado	BIT
Dni_anfitrión	VARCHAR
Fecha	DATETIME
Precio_por_persona	DOUBLE
Privado	BIT
Id_pista	INT

Tabla 12: Tabla Evento Base de Datos

Esta tabla es otra de las fundamentales. Se encarga de guardar los eventos que hay con su correspondiente fecha y hora.

Evento_usuario

Campo	Tipo
Id_evento	INT
Dni_usuario	BIT

Tabla 13: Tabla Evento_usuario Base de Datos

En este sistema donde un usuario puede participar en múltiples eventos y, al mismo tiempo, un evento puede involucrar a varios usuarios, es fundamental contar con una estructura que gestione estas relaciones de manera eficiente. Por ello, la función principal de esta tabla es hacer de tabla intermedia que permite establecer una relación de “muchos a muchos” entre la tabla “usuario” y la tabla “evento”.

Rol

Campo	Tipo
Id	BIGINT
Nombre	VARCHAR

Tabla 14: Tabla Rol Base de Datos

Esta tabla simplemente guarda los distintos roles que puede tener los usuarios registrados en el sistema.

Usuario_rol

Campo	Tipo
Usuario_dni	VARCHAR
Rol_id	BIGINT

Tabla 15: Tabla Usuario_rol Base de Datos

Esta tabla tiene como función principal vincular a los usuarios con los roles específicos que pueden tener dentro del sistema.

Sancion

Campo	Tipo
Id	INT
Fecha_fin	DATE
Fecha_inicio	DATE
Motivo	VARCHAR
Usuario_id	VARCHAR

Tabla 16: Tabla Sanción Base de Datos

Tabla que contiene los datos asociados a la sanción a un usuario que ha sido sancionado incluyendo la fecha de inicio, de fin y el motivo de la sanción.

Asistencia

Campo	Tipo
Id	INT
Asistio	BIT
Id_evento	INT
Dni_usuario	VARCHAR

Tabla 17: Tabla Asistencia Base de Datos

Tabla que contiene los datos asociados a la asistencia de un usuario a un evento. Para ello tiene los atributos id_evento que hace referencia al evento a asistir y dni_usuario que hace referencia al usuario que debe asistir al evento.

Evento_solicitudes

Campo	Tipo
Id_evento	INT
Dni_usuario	VARCHAR

Tabla 18: Tabla Evento_solicitudes Base de Datos

Esta tabla contiene las solicitudes que los usuarios envían a un evento cuando este es privado para poder ser aceptados y, de esa manera, unirse al

evento. En consecuencia, id_evento hace referencia al evento al que quiere unirse y dni_usuario hace referencia al usuario que quiere unirse al evento.

Pista

Campo	Tipo
Id_pista	INT
Cubierta	BIT
Duracion_partido	INT
Nombre	VARCHAR
Num_max_jugadores	INT

Tabla 19: Tabla Pista Base de Datos

Esta tabla se encarga de almacenar toda la información relevante sobre cada una de las pistas disponibles en el sistema. Incluye características importantes como el campo “cubierta” para informar si la pista está en una zona cubierta o al aire libre.

Horario

Campo	Tipo
Id	INT
Fecha	DATE
Horas	VARCHAR
Id_pista	INT

Tabla 20: Tabla Horario Base de Datos

En esta tabla se almacena la información de los horarios que se asocian a una pista mediante el campo “id_pista” y que contiene las horas disponibles para una fecha en concreto.

Tarifa

Campo	Tipo
Id	INT
Cubierta	BIT
Fecha_inicio	DATE

Precio	DOUBLE
Turno	ENUM

Tabla 21: Tabla Tarifa Base de Datos

Esta tabla contiene los distintos campos variables y el precio que se le asignan a esos campos para las tarifas de las reservas. Esto permite que se pueda poner diferentes tarifas dependiendo de si la pista es cubierta o no, o dependiendo del turno en el que se juegue diferenciando entre turno de mañana o de tarde. Esto da más variedad y enriquece un poco la calidad de nuestra aplicación.

3 DESARROLLO

Como ya se ha explicado en apartados anteriores, se usará una metodología mixta entre Kanban y Scrum para el desarrollo de la aplicación. De esta manera el desarrollo se dividirá en iteraciones, cada una de las cuales abarcan unas historias de usuario, con su correspondiente diseñó, implementación y pruebas de funcionamiento que permitirán incrementar el valor del producto conforme se van completando las iteraciones y va avanzando el proyecto.

En cada iteración, para conseguir mostrar al cliente el incremento del valor del producto se va a desarrollar tanto la parte del backend, como del frontend que sea necesaria para completar las historias de usuario a realizar en dicha iteración.

A continuación, se va a proceder a detallar cada iteración en la que se ha dividido el desarrollo del proyecto, incluyendo las historias de usuario a implementar, detalles de la implementación y sus correspondientes pruebas, todo ello complementado con una serie de diagramas cuando sea necesario.

Cada iteración se dividirá en varios apartados para una mejor organización de su contenido. Dichos apartados son: Historias de usuario, implementación, pruebas y resultados.

3.1 Primera Iteración

En esta primera iteración vamos a implementar una parte esencial del proyecto como es la creación de usuarios, su correspondiente inicio de sesión y el sistema de roles. El usuario es crucial para el funcionamiento básico del sistema y, además, establece una base sólida y segura sobre la cual se implementarán funcionalidades en fases futuras.

3.1.1 Casos de uso

- Registro de usuario: El usuario podrá registrarse en la aplicación para poder tener acceso a todas las funcionalidades de la aplicación.
- Inicio de sesión: El usuario podrá iniciar sesión en la aplicación con unas credenciales previamente registradas que le darán acceso a la aplicación.
- Sistema de roles: Se debe implementar un sistema de roles para que un administrador tenga acceso privado a su panel de control.

En consecuencia, el tablero Kanban realizado con la aplicación Trello quedaría de la siguiente manera:

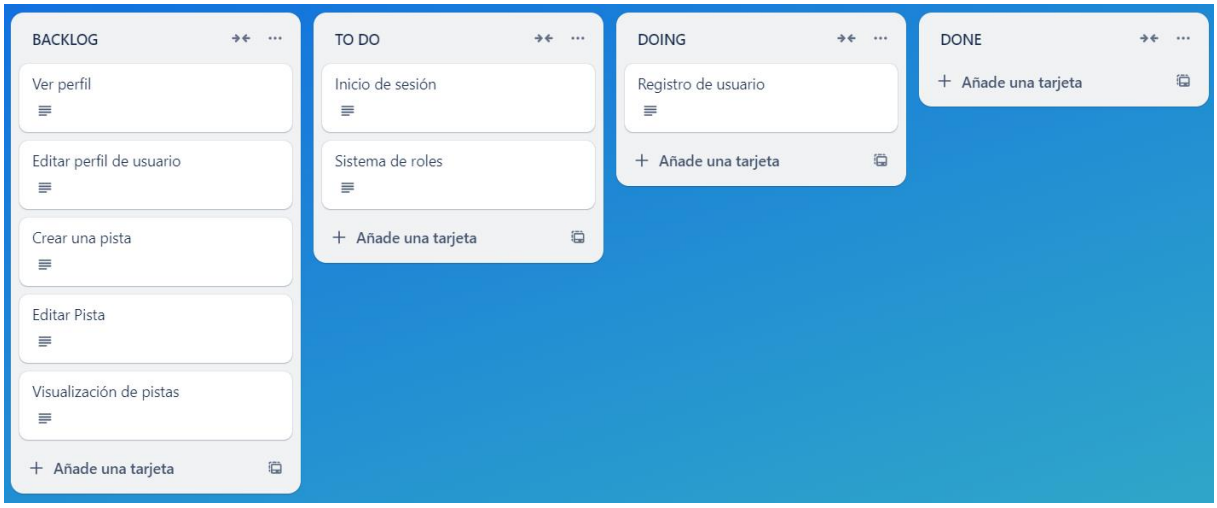


Ilustración 16: Ilustración Tablero Kanban Iteración 1

3.1.2 Implementación

Una vez creado el proyecto con Spring y tenerlo todo configurado y listo para empezar, comenzamos con la creación de la entidad “Usuario”, con sus atributos

básicos y que van a ser rellenados posteriormente en el formulario de registro creado en el frontend.

Tras haber creado la entidad, procedemos con la creación de la función en el servicio encargada de guardar un usuario en la base de datos, con las correspondientes comprobaciones para comprobar si ya existe un usuario con ese DNI o ese Nick y, teniendo en cuenta el aspecto de la seguridad, antes de guardar la contraseña en la base de datos se encripta usando un PasswordEncoder para que incluso si alguien accede a la base de datos, no pueda ver la contraseña en texto claro. Una vez creada la función en el servicio, procedemos con la implementación del endpoint que llamará el frontend para crear el nuevo usuario.

Tras la creación de las funciones en el backend se procede con el diseño del frontend, diseñando los componentes en React que contienen las vistas que permitirán a un usuario registrarse.

Tras completar el código relacionado con el registro de usuario, procedemos a la creación de un login, lo que implica configurar la seguridad del sistema. Para ello creamos una clase de configuración de Spring Security, añadiendo la autorización de solicitudes de la siguiente manera:

```
authorizeRequests
        .requestMatchers("/eduardotfg/registrar", "/eduardotfg/login","/eduardotfg/asistencias/**").permitAll()
        .requestMatchers("/eduardotfg/usuarios/**").hasAnyAuthority( ...authorities: "ADMIN", "USER")
        .requestMatchers("/eduardotfg/admin/**").hasAuthority("ADMIN")
```

Ilustración 17: Ilustración Filtros de seguridad

De esta manera cualquier usuario puede acceder a las rutas de registro y login, pero a las url que empiecen con ”/eduardotfg/usuarios/” deben estar autorizados con algún rol y los que empiecen con “/eduardotfg/admin/” deben tener el rol de admin. Para los roles hemos creado una entidad Rol que representa el tipo de rol de los usuarios. Cuando se registra un nuevo usuario, automáticamente se le asigna el rol “USER”.

Para la autenticación y autorización de la aplicación hemos optado por usar JWT por varias razones:

- Permite que el servidor sea “sin estado”, ya que toda la información del usuario está contenida en el token, lo que elimina la necesidad de mantener sesiones en el servidor.
- Los JWT son compactos y se pueden enviar fácilmente en encabezados HTTP o como parte de URLs, lo que los hace adecuados para aplicaciones móviles y web.
- Los JWT están firmados para garantizar que el contenido no ha sido manipulado. Además, pueden incluirse fechas de expiración para limitar su validez en el tiempo.

Para implementarlo en nuestro proyecto, primeramente, hemos creado una clase que genere un token con una Authentication que contiene unas credenciales de inicio de sesión. El token es firmado y se devuelve al frontend con los datos necesarios (Siempre y cuando el inicio de sesión sea correcto). Ahora cuando se quiera realizar una petición, se deberá enviar el token en los parámetros de la misma y este será validado por un filtro de autenticación que hemos creado en una clase.

En este filtro se valida que el token ha sido generado con nuestra firma, se obtienen los datos del usuario para validar que está en nuestra base de datos y se comprueba sus roles. Una vez se haya comprobado la validez del token se procesa la petición del frontend, teniendo en cuenta las restricciones de los roles y de CORS.

3.1.3 Pruebas

Para la realización de las pruebas hemos creado unos test para comprobar la eficacia de las funciones que intervienen en cada uno de las historias de usuario implementadas en la iteración. Para crearemos unas tablas que resuman el contenido de la prueba y su resultado.

Campo	Detalles
Id	PR01

Historia de usuario	Registro de usuario
Descripción	Registro de usuario con correo inválido
Datos de entrada	Usuario con uno de los datos no válidos
Acción	Enviar solicitud POST a /registrar con los datos del usuario
Resultado esperado	Respuesta HTTP 400
Estado	✓ testAltaUsuarioInvalido()

Tabla 22: Tabla Prueba 1 Registro de Usuario

Campo	Detalles
Id	PR02
Historia de usuario	Registro de usuario
Descripción	Verificar que un usuario se registra correctamente
Datos de entrada	Usuario con datos válidos
Acción	Enviar solicitud POST a /registrar con los datos del usuario
Resultado esperado	Respuesta HTTP 201 CREATED
Estado	✓ testAltaUsuario()

Tabla 23: Tabla Prueba 2 Registro de Usuario

Campo	Detalles
Id	PR03
Historia de usuario	Inicio de sesión
Descripción	Verificar que un usuario registrado puede iniciar sesión correctamente
Datos de entrada	Credenciales de un usuario registrado previamente
Acción	Enviar solicitud POST a /registrar con los datos del usuario. Luego, realizar solicitud POST a /login con las credenciales del usuario registrado
Resultado esperado	Respuesta HTTP 200 OK y el cuerpo de la respuesta debe contener un “accesToken” no vacío
Estado	✓ testLogin()

Tabla 24: Tabla Prueba 3 Inicio de Sesión

3.1.4 Resultados

A continuación, se muestran los resultados de todo lo implementado en esta iteración.

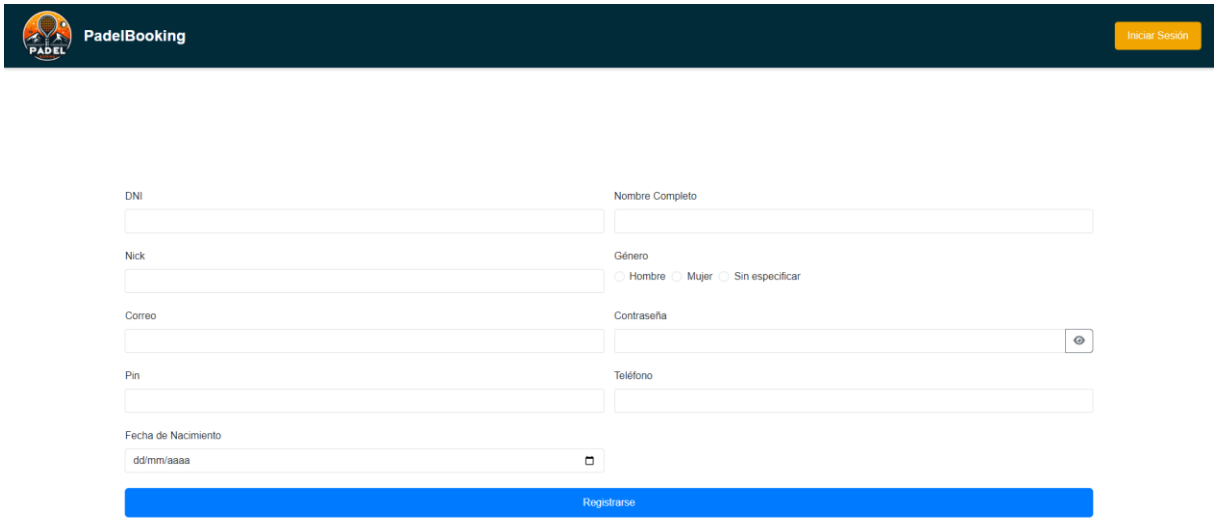


Ilustración 18: Ilustración Página de Registro

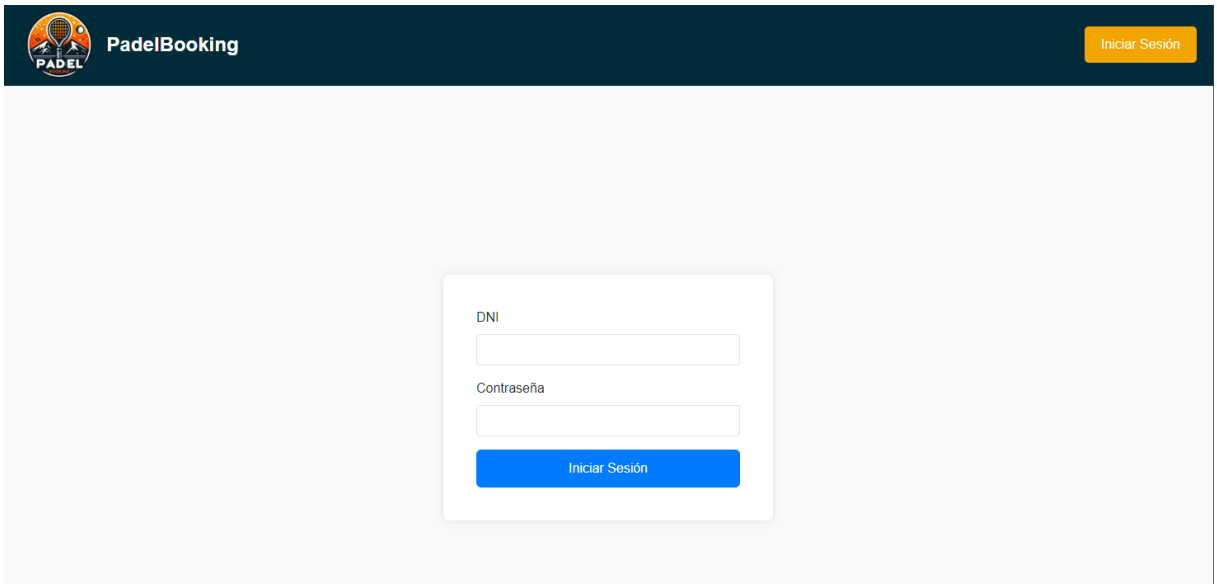


Ilustración 19: Ilustración Página de Inicio de Sesión

3.2 Segunda Iteración

En esta segunda iteración procederemos con la integración del menú principal de la aplicación, que es la visualización de las pistas. Para ello deberemos implementar la clase Pista y un menú para manejarlas. Además, seguiremos enriqueciendo la experiencia del usuario dando la posibilidad a una mayor personalización del perfil.

3.2.1 Casos de uso

- Ver perfil: Un usuario podrá acceder a su perfil para visualizar sus datos.
- Editar perfil: Un usuario podrá acceder a una vista de edición de perfil en la que modificar alguno de sus datos.
- Manejo de pistas: El administrador tendrá acceso a una vista donde poder crear/editar/eliminar las pistas del sistema.
- Visualizar pistas: Un usuario podrá ver el menú principal donde aparecen las pistas disponibles en la aplicación.

Justo al empezar la iteración el estado del tablero es el siguiente:

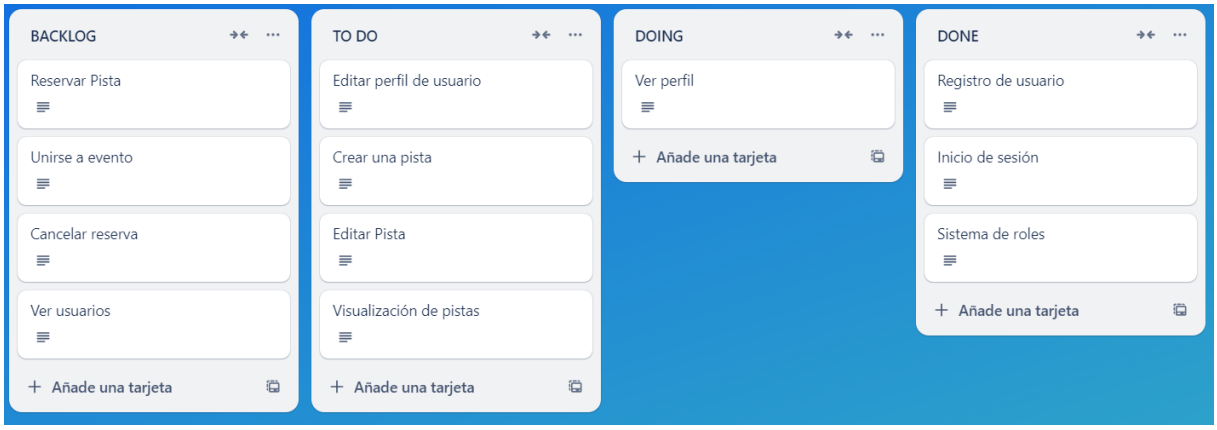


Ilustración 20: Ilustración Tablero Kanban Iteración 2

3.2.2 Implementación

Esta iteración la comenzamos con la creación de una vista en la que aparezca algunos de los datos de perfil del usuario para tener una información general. Para que aparezcan esos datos, primeramente, hay que obtenerlos. Para ello hemos

creado un endpoint al que solicitarle los datos de un usuario. De esta manera se solicitarán los datos de un usuario y se mostrarán en la vista los principales.

Debajo aparecerá un botón que permita acceder a un formulario para poder editar algunos de los datos del usuario como puede ser el número de teléfono, el nombre, la fecha de nacimiento, etc.

Si accedemos a la vista de editar perfil se cargará un formulario con todos los datos del usuario, algunos no modificables y que el usuario podrá cambiar (siempre cumpliendo con unas restricciones básicas) para posteriormente mandar el usuario editado a un endpoint creado que guarda los nuevos datos del usuario en la base de datos.

Una vez implementado todo lo relacionado con el perfil de usuario seguimos con la creación de un menú para el administrador en la que poder manejar las pistas del sistema. Para ello, crearemos la entidad Pista en el backend, con su correspondiente repositorio y su DTO para poder transferir los datos con el frontend.

Una vez tenemos la entidad Pista creada creamos un menú principal para el administrador en el que se visualice una lista con las pistas que ya existen y en el que aparezca un botón para poder eliminar la pista, además de otro botón que te lleve a una vista con un formulario donde crear la pista. Para que el menú siempre se mantenga actualizado con las pistas que hay en el sistema se usará useEffect para que cada vez que cambie la dependencia location, se obtengan todas las pistas que hay en el sistema.

```
useEffect( effect: () : void  ⇒ {
    fetchPistas();
}, deps: [location]);
```

Para la edición de la pista se hará de manera similar a la edición del perfil, obteniendo los datos de la pista y poniéndolos en un formulario que se editará y se

llamará a un endpoint para enviarlos y actualizar los datos de la pista en la base de datos. Para terminar con el manejo de pistas se crea un endpoint al que se le envía el id de la pista a eliminar y la borra de la base de datos.

Para terminar con las implementaciones de esta iteración crearemos una vista en la que se muestren las pistas del sistema en un horario predefinido que se mejorará en próximas iteraciones para poder hacer reservas de las mismas.

1.1.1 Pruebas

A continuación, mostramos las pruebas realizadas para las funciones usadas en esta iteración. A la hora de implementar estas pruebas se ha tenido que crear una clase nueva de Seguridad que solo se utiliza en los perfiles “test” para que no se necesite autorización a la hora de realizar las pruebas. Por tanto, la otra clase de Seguridad se ejecutará en los perfiles “!test”.

Campo	Detalles
Id	PR04
Historia de usuario	Ver perfil
Descripción	Verificar que se pueden obtener los datos de un usuario registrado
Datos de entrada	El DNI de un usuario previamente registrado
Acción	Enviar solicitud POST a /registrar con los datos del usuario. Luego, realizar solicitud GET a /usuarios/{dni}/obtener para obtener los datos del usuario
Resultado esperado	Respuesta HTTP 200 OK y el cuerpo de la respuesta debe contener los datos del usuario que ha sido registrado
Estado	✓ testObtenerDatosUsuario()

Tabla 25: Tabla Prueba 4 Ver Perfil

Campo	Detalles
Id	PR05
Historia de usuario	Editar perfil
Descripción	Verificar que se pueden editar los datos de un usuario existente
Datos de entrada	Datos cambiados de un usuario previamente registrado

Acción	Enviar solicitud POST a /registrar con los datos del usuario. Luego, cambiar algunos datos del usuario y enviar una solicitud PUT a /usuarios/{dni}/editar con el usuario modificado
Resultado esperado	Respuesta HTTP 200 OK y el cuerpo de la respuesta debe contener los datos del usuario modificado
Estado	✓ testEditarDatosUsuario()

Tabla 26: Tabla Prueba 5 Editar Perfil

Campo	Detalles
Id	PR06
Historia de usuario	Manejo de pistas
Descripción	Verificar que se puede crear una nueva pista correctamente
Datos de entrada	Datos de una entidad Pista
Acción	Enviar solicitud POST a /admin/pista para crear una pista con los datos proporcionados
Resultado esperado	Respuesta HTTP 201 CREATED y el cuerpo de la respuesta debe contener los datos de la pista creada
Estado	✓ testAltaPista()

Tabla 27: Tabla Prueba 6 Manejo de pistas

Campo	Detalles
Id	PR07
Historia de usuario	Manejo de pistas
Descripción	Verificar que se puede editar una pista existente correctamente
Datos de entrada	Datos de una entidad Pista editada
Acción	Enviar solicitud PUT a /admin/pista/{idPista}/editar para enviar los datos de la pista actualizada
Resultado esperado	Respuesta HTTP 200 OK y el cuerpo de la respuesta debe contener los datos de la pista editada
Estado	✓ testEditarPista()

Tabla 28: Tabla Prueba 7 Manejo de pistas

Campo	Detalles

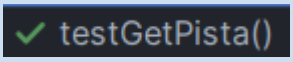
Id	PR08
Historia de usuario	Manejo de pistas
Descripción	Verificar que se pueden obtener correctamente los detalles de una pista existente
Datos de entrada	Datos de una entidad Pista
Acción	Enviar solicitud POST a /admin/pista para crear la pista y luego hacer GET a /admin/pista/{idPista} y obtener la pista creada
Resultado esperado	Respuesta HTTP 200 OK al obtener la pista y que los datos sean los mismos que los de la pista creada
Estado	

Tabla 29: Tabla Prueba 8 Manejo de pistas

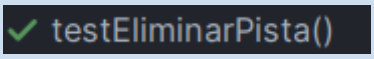
Campo	Detalles
Id	PR09
Historia de usuario	Manejo de pistas y Visualizar pistas
Descripción	Verificar que se puede eliminar una pista y que la eliminación es efectiva
Datos de entrada	Datos de una entidad Pista
Acción	Enviar solicitud DELETE a /admin/pista/{idPista}/eliminar para eliminar la pista y luego hacer GET a /usuarios/obtenerPistas y obtener una lista de pistas
Resultado esperado	Respuesta HTTP 200 OK al eliminar la pista y al obtener las pistas tener una lista de tamaño 0
Estado	

Tabla 30: Tabla Prueba 9 Manejo de pistas y Visualizar pistas

Campo	Detalles
Id	PR10
Historia de usuario	Visualizar pistas
Descripción	Verificar que se puede obtener todas las pistas registradas
Datos de entrada	Datos de tres pistas para ser creadas
Acción	Enviar solicitud GET a /usuarios/obtenerPistas para obtener todas las pistas del sistema
Resultado esperado	Respuesta HTTP 200 OK al obtener las pistas y obtener una lista con las pistas creadas

Estado	✔ testGetPistas()

Tabla 31: Tabla Prueba 10 Visualizar pistas

1.1.1 Resultados

Aquí se muestran los resultados de todo lo implementado durante esta iteración pudiendo ver un avance visual del proyecto.

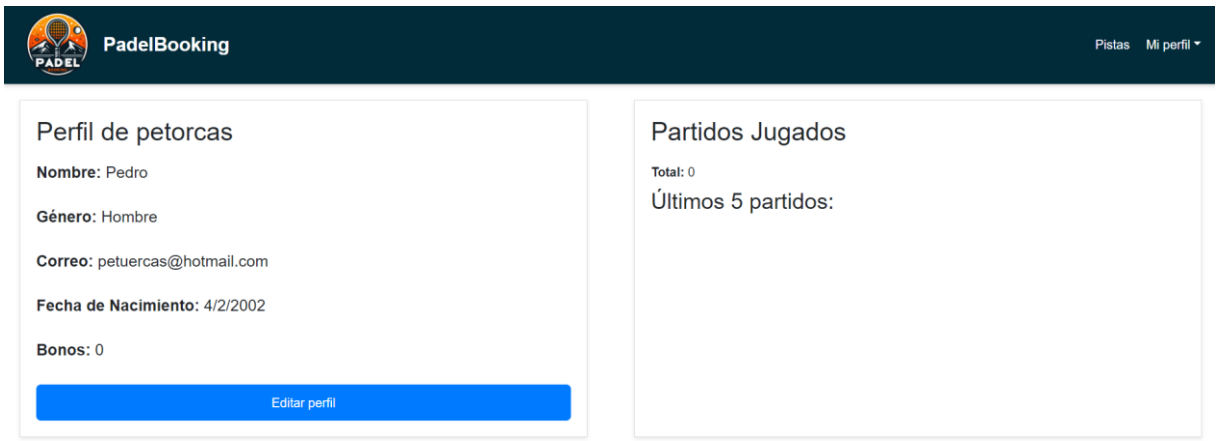


Ilustración 21: Ilustración Vista de Perfil de Usuario

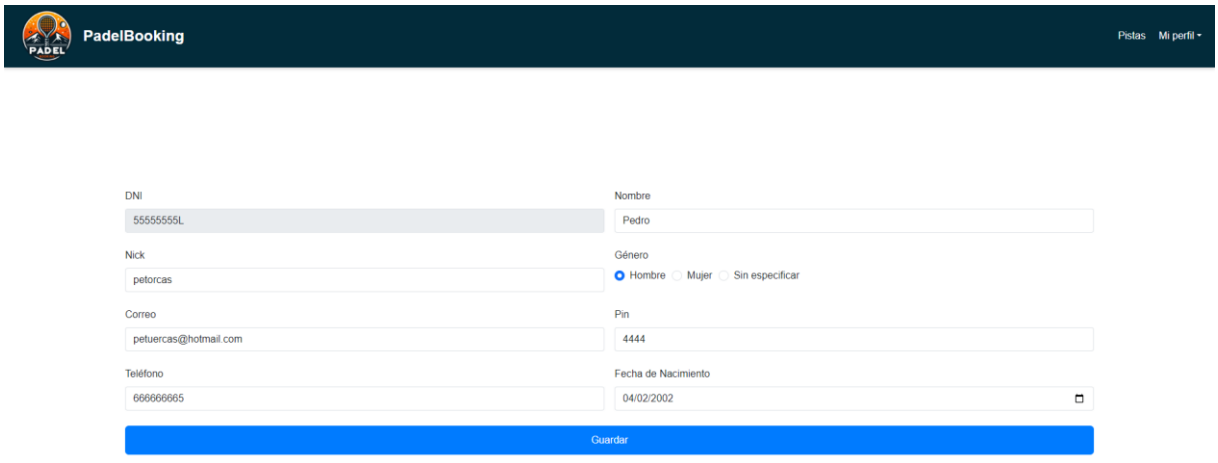


Ilustración 22: Ilustración Vista de Editar Perfil

Ilustración 23: Ilustración Vista de Menú Principal

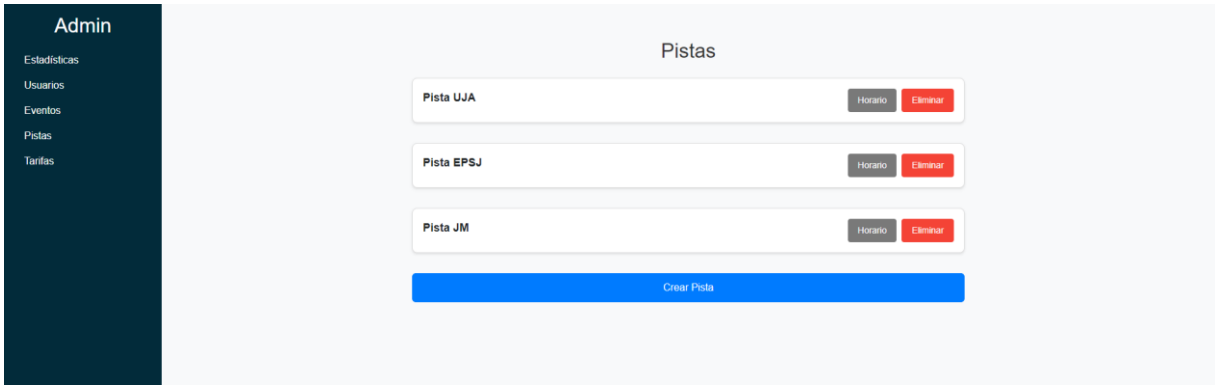


Ilustración 24: Ilustración Vista de Pistas Administrador

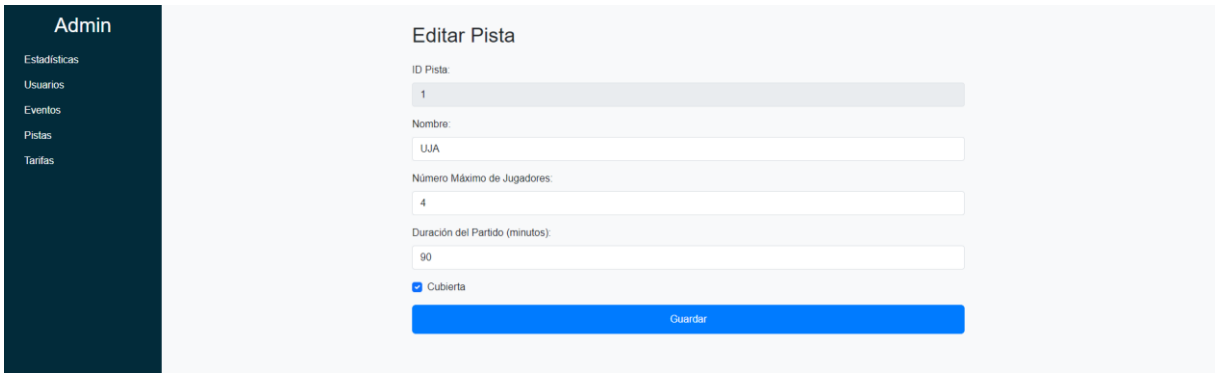


Ilustración 25: Ilustración Vista de Editar Pista Administrador

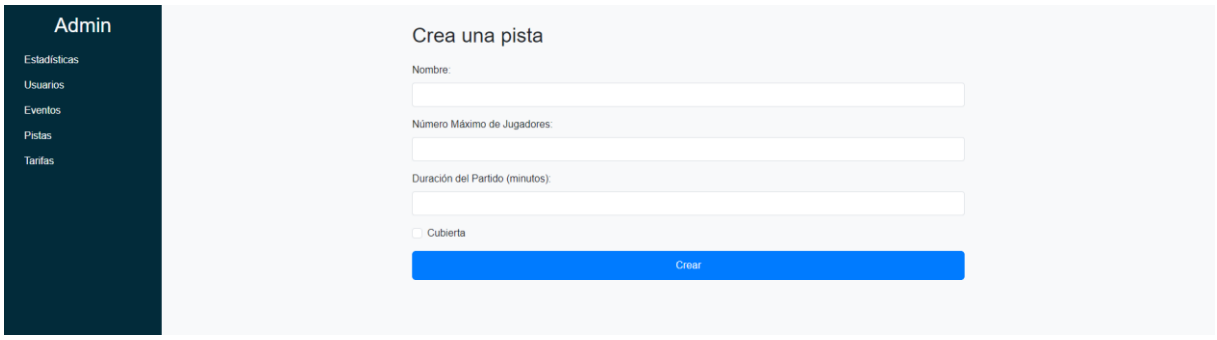


Ilustración 26: Ilustración Vista de Crear Pista Administrador

3.3 Tercera Iteración

En esta iteración, tras tener ya una base para poder visualizar las pistas, vamos a generar los horarios en los que podrán reservarse las pistas para poder empezar a realizar reservas.

3.3.1 Casos de uso

- Manejo de horarios: El administrador tendrá acceso a una vista en la que poder editar los horarios de disponibilidad de una pista.
- Reservar una pista: Un usuario podrá reservar una pista para comenzar un evento en una pista y a una hora disponibles.
- Unirse a evento: Un usuario podrá unirse a un evento que previamente haya sido creado por otro usuario.
- Cancelar evento: Un usuario podrá cancelar su unión al evento cumpliendo con un margen definido.
- Ver usuarios: Un usuario podrá ver los usuarios que están dentro del evento y acceder a su perfil.

Una vez finalizada la iteración anterior, al comenzar esta iteración el tablero Kanban queda de la siguiente manera:

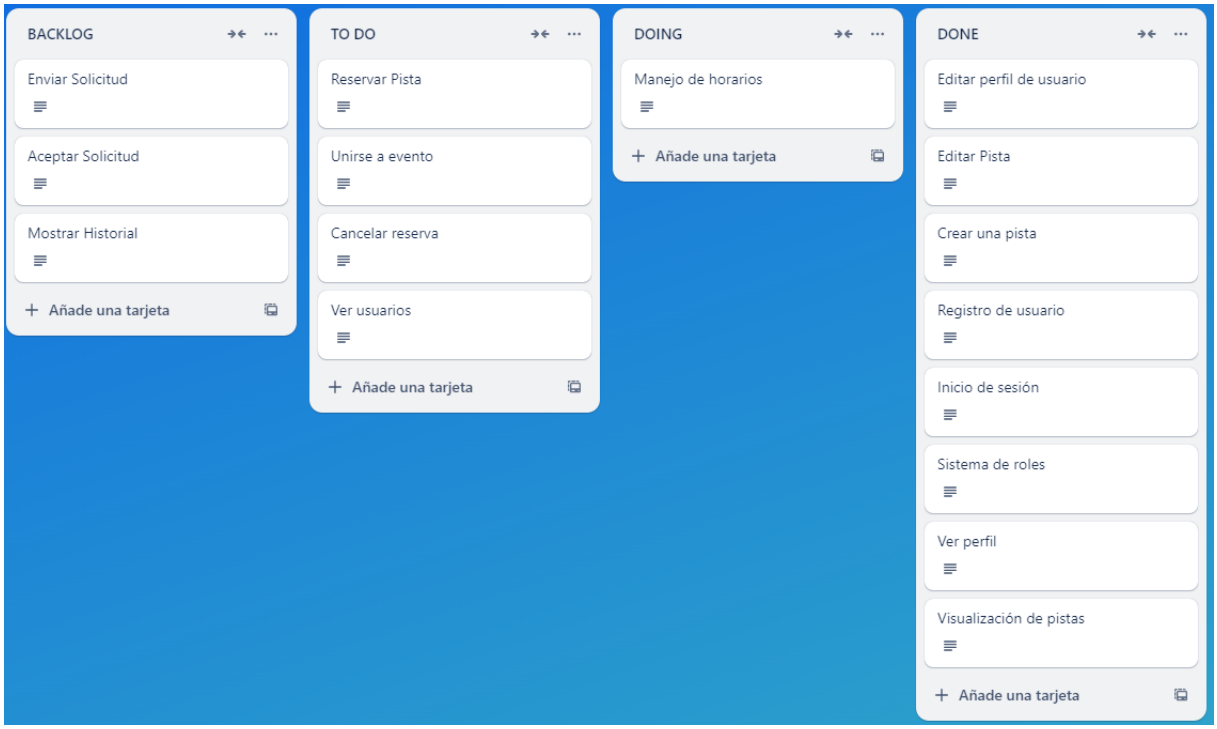


Ilustración 27: Ilustración Tablero Kanban Iteración 3

1.1.1 Implementación

Para comenzar con esta iteración creamos la entidad Horario que guarda las horas para un día y una pista que estará disponible para reservar.

Una vez creada la entidad Horario procedemos con la creación de una vista en la que poder manejar los horarios de las pistas. Para este diseño he pensado en seleccionar el horario de una pista que, por defecto, carga (si lo hay) el horario de la semana actual. Entonces el procedimiento es, seleccionar la semana que quieras modificar, debajo aparecerá un calendario de la semana. Ahora, el administrador podrá seleccionar día a día y hora a hora la disponibilidad o puede pulsar directamente en la columna para seleccionar directamente todas las horas de ese día. También hemos creado un checkbox debajo por si el administrador quiere que ese horario para esa semana se aplique a todas las pistas disponibles o solo a la pista actual.

Cuando se cambia la fecha de la que se quiere manejar los horarios de la pista, se obtienen los horarios de esa semana para, en el caso de que haya, se puedan modificar.

```
useEffect( effect: () : void  ⇒ {
    const fetchHorarios = async () : Promise<any>  ⇒ {

    };

    fetchHorarios();
}, deps: [startDate]);
```

Disponemos de unos días y horas predefinidas en el frontend que son las que estarán disponibles para los horarios.

```
const diasSemana : string[]  = ["Lunes", "Martes", "Miércoles", "Jueves", "Viernes", "Sábado", "Domingo"];
const horasDisponibles : string[]  = ["8:00", "9:30", "11:00", "12:30", "16:00", "17:30", "19:00", "20:30"];
```

A la hora de renderizar el horario, mapeamos en una tabla los días de la semana y las horas disponibles. Creamos el componente Hora por cada hora disponible en cada día que podrá ser seleccionado.

```
<div className="table-responsive">
    <table className="table table-bordered table-hover table-sm mx-auto">
        <thead className="thead-dark">
        <tr>
            <th scope="col"></th>
            {diasSemana.map((dia : string , index : number ) => (
                <th scope="col" key={dia} style={{ cursor: 'pointer' }}
                    onClick={() : void  => toggleAllHorasForDay(addDays(startOfSelectedWeek, index))}>
                    {dia} <br />
                    {format(addDays(startOfSelectedWeek, index), formatStr: 'dd/MM/yyyy')}
                </th>
            ))}
        </tr>
        </thead>
        <tbody >
        {horasDisponibles.map((hora : string ) => (
            <tr key={hora}>
                <th scope="row" style={{ whiteSpace: 'nowrap' }}>{hora}</th>
                {diasSemana.map((dia : string , index : number ) => {
                    const fecha : string  = format(addDays(startOfSelectedWeek, index), formatStr: 'yyyy-MM-dd');
                    const isFechaPasada : boolean  = isBefore(new Date(fecha), fechaActual) || isToday(new Date(fecha));

                    return (
                        <td key={`${dia}-${hora}`} style={{ padding: '0.25rem' }}>
                            <Hora
                                hora={hora}
                                isSelected={horasSeleccionadas[fecha] && horasSeleccionadas[fecha].includes(hora)}
                                onToggle={isFechaPasada ? undefined : () : void  => toggleHora(fecha, hora)}
                                disabled={isFechaPasada}
                            />
                        </td>
                    );
                })}
            </tr>
        ))}
        </tbody>
    </table>
</div>
```

Cada vez que se hace click sobre el componente Hora se llama a la siguiente función que añade/quita en el día la hora que ha sido seleccionada.

```
const toggleHora = (dia, hora) : void  => {  Show usages   Eduasso
    setHorasSeleccionadas( value: (prevSeleccionadas : {} ) : {}  => {
        const newSeleccionadas : {}  = { ...prevSeleccionadas };
        if (!newSeleccionadas[dia]) {
            newSeleccionadas[dia] = [];
        }
        if (newSeleccionadas[dia].includes(hora)) {
            newSeleccionadas[dia] = newSeleccionadas[dia].filter((h) : boolean  => h !== hora);
        } else {
            newSeleccionadas[dia].push(hora);
        }
        return newSeleccionadas;
    });
};
```

Una vez se ha ajustado el horario de la manera pertinente, se pulsa el botón de guardar que llama a una función que se encarga de, para cada día de la semana que haya horas seleccionadas, crea un DTO de horario y realiza una petición para guardarla en la base de datos. Si se ha activado el checkbox realiza lo anterior, pero para cada una de las pistas que haya. En el servidor si se envía un horario que ya existía, pero actualizado, lo comprueba y lo actualiza en la base de datos.

```
const guardarHoras = async () : Promise<void>  => {  Show usages   Eduasso
    const pistaIds : any[]  = applyToAll ? allPistaIds : [idPista];

    try {
        await Promise.all(
            pistaIds.flatMap( callback: pistaId =>
                Object.keys(horasSeleccionadas)
                    .filter(dia : string  => horasSeleccionadas[dia].length > 0)
                    .map(async dia : string  => {
                        const dtoHorario : {...}  = {
                            fecha: dia,
                            horas: horasSeleccionadas[dia].join('/'),
                            idPista: pistaId
                        };

                        try {
                            const response : AxiosResponse<any>  = await axios.post(
                                url: `http://localhost:8080/eduardotfg/admin/${pistaId}/horario`,
                                dtoHorario,
                                config: {
                                    headers: {
                                        'Authorization': `Bearer ${token}`,
                                        'Content-Type': 'application/json'
                                    }
                                }
                            );

                            console.log('Success:', response.data);
                        } catch (error) {
                            if (error.response) {
                                console.error('Error:', error.response.statusText);
                            } else {
                                console.error('Error:', error.message);
                            }
                        }
                    })
            )
        );
    } catch (error) {
        console.error('Error al guardar las horas:', error.message);
    }
};
```

```
public Horario altaHorario(@NotNull Horario horario){  6 usages  Eduasso
    Optional<Horario> horarioOptional = repositorioHorarios.buscarFechaPista(horario.getFecha(), horario.getIdPista());
    if(horarioOptional.isPresent()){
        Horario horarioExistente = horarioOptional.get();
        horarioExistente.setHoras(horario.getHoras());
        repositorioHorarios.actualizar(horarioExistente);
        return horarioExistente;
    }else{
        repositorioHorarios.guardar(horario);
        return horario;
    }
}
```

Hay que tener en cuenta que las horas se guardan en una cadena de la siguiente forma “9:30/11:00/16:00” de tal manera que si se selecciona una hora se añade a la cadena y si se deselecciona se quita, sin importar el orden de las horas.

Ahora, al cargar el menú principal se obtendrá el horario para el día seleccionado y para cada una de las pistas para que estén disponibles para el usuario.

Tras la implementación de los horarios seguimos con la creación de la entidad Evento, que es otra de las clases vitales de nuestra aplicación. A destacar de la entidad está la relación que tiene con Usuario, ya que por cada evento hay varios usuarios que serán los participantes.

```
@ManyToMany  7 usages
@JoinTable(
    name = "eventoUsuario",
    joinColumns = @JoinColumn(name = "idEvento"),
    inverseJoinColumns = @JoinColumn(name = "dniUsuario")
)
private List<Usuario> participantes;
```

Continuamos modificando el menú principal del usuario para que al renderizarse obtenga los horarios y eventos que haya para ese día. Por cada hora que haya en cada pista se crea un componente Square al que se le pasa un evento si existe para esa pista y para esa fecha y hora, además de los datos de la pista y si

está disponible. Al hacer click sobre el Square se muestran los datos del evento y el botón de Reservar/Unirse dependiendo si existe ya un evento.

Una función interesante para la mejor visualización de las pistas del usuario es la de setBackgroundColor() que, dependiendo de si hay evento, si el usuario está dentro del mismo o si está lleno el evento, se muestra el Square de diferente color. Para entender cada color hay una leyenda bajo las pistas para poder consultarlo.

```
const setBackgroundColor = () : string  ⇒ {  Show usages  Eduasso
    let backgroundcolor;
    if (!evento) {
        backgroundcolor = 'bg-verde';
    } else {
        if (evento.participantes.length === datos.maxJugadores) {
            if (evento.participantes.includes(datos.dniUsuario)) {
                backgroundcolor = 'bg-morado';
            } else {
                backgroundcolor = 'bg-rojo';
            }
        } else {
            if (evento.participantes.includes(datos.dniUsuario)) {
                backgroundcolor = 'bg-amarillo';
            } else {
                backgroundcolor = 'bg-naranja';
            }
        }
    }
    return backgroundcolor;
};
```

También está la función getButton() que devuelve el botón que debe aparecer en el popup dependiendo si hay evento o no y de si el usuario está dentro del evento.

```
if (evento) {
    if (evento.participantes.length < datos.maxJugadores && evento.privado === false
        && !evento.participantes.includes(datos.dniUsuario)) {
        return (
            <button type="submit" className="btn btn-primary"
                    onClick={handleSubmitUnirse} disabled={buttonDisabled}>
                Unirse
            </button>
        );
    } else if (evento.participantes.length < datos.maxJugadores && evento.privado === true
        && !evento.participantes.includes(datos.dniUsuario)) {
        if (evento.solicitudes.includes(datos.dniUsuario)) {
            return (
                <button type="submit" className="btn btn-primary"
                        onClick={handleSubmitCancelarSolicitud} disabled={buttonDisabled}>
                    Cancelar Solicitud
                </button>
            );
```

```
} else {
    return (
        <button type="submit" className="btn btn-primary" onClick={handleSubmitReservar} disabled={buttonDisabled}>
            Reservar
        </button>
    );
}
```

Para mostrar los usuarios que participan en el evento se ha creado un componente en forma de tarjeta en la que aparece el Nick del usuario y si se le hace click te envía a la vista del perfil del usuario de la misma manera en la que puedes visualizar tu propio perfil. Si un usuario intenta salirse de un evento al que le quedan menos de 24h para que empiece no podrá salirse y le saldrá un mensaje de aviso para que sepa que no puede cancelar la reserva.

1.1.1 Pruebas

Continuamos como cada iteración con la realización de los test para comprobar que todas las funciones implementadas funcionan correctamente.

Campo	Detalles
Id	PR11
Historia de usuario	Manejo de horarios
Descripción	Verificar que se puede crear correctamente un horario para

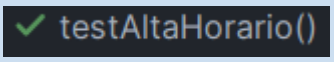
	una pista existente
Datos de entrada	Datos del horario para una pista previamente creada
Acción	Crear un horario para una pista enviando una solicitud POST a /admin/{idPista}/horario y enviando los datos del horario
Resultado esperado	Respuesta HTTP 201 CREATED al crear el horario y los datos del horario creado
Estado	

Tabla 32: Tabla Prueba 11 Manejo de horarios

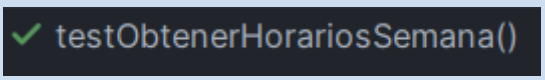
Campo	Detalles
Id	PR12
Historia de usuario	Manejo de horarios
Descripción	Verificar que se puede obtener correctamente los horarios de una pista para una semana específica
Datos de entrada	La pista creada y los horarios para distintos días
Acción	Crea una pista y varios horarios para luego enviar una solicitud GET a /admin/{idPista}/obtenerHorarioSemana pasando como parámetros la fecha de inicio y fin de la semana
Resultado esperado	Respuesta HTTP 200 OK al hacer la petición y la respuesta debe contener una lista con los horarios existentes para esa pista entre las fechas pasadas como parámetros
Estado	

Tabla 33: Tabla Prueba 12 Manejo de horarios

Campo	Detalles
Id	PR13
Historia de usuario	Manejo de horarios
Descripción	Verificar que se puede obtener el horario correcto para un día específico y una pista en concreto
Datos de entrada	Datos de la pista y del horario creados
Acción	Crea una pista y un horario para luego enviar una solicitud GET a /admin/{idPista}/obtenerHorario pasando como parámetro una fecha

Resultado esperado	Respuesta HTTP 200 OK al hacer la petición y la respuesta debe contener el horario creado para ese día y esa pista
Estado	✓ testObtenerHorario()

Tabla 34: Tabla Prueba 13 Manejo de horarios

Campo	Detalles
Id	PR14
Historia de usuario	Manejo de horarios
Descripción	Verificar que se puede obtener el horario correcto para un día específico y una pista en concreto
Datos de entrada	Datos de la pista y del horario creados
Acción	Crea una pista y un horario para luego enviar una solicitud GET a /admin/{idPista}/obtenerHorario pasando como parámetro una fecha
Resultado esperado	Respuesta HTTP 200 OK al hacer la petición y la respuesta debe contener el horario creado para ese día y esa pista
Estado	✓ testObtenerHorario()

Tabla 35: Tabla Prueba 14 Manejo de horarios

Campo	Detalles
Id	PR15
Historia de usuario	Reservar pista
Descripción	Verificar la creación de un evento con un usuario y una pista
Datos de entrada	Usuario y pista creadas y los datos del evento que se va a crear
Acción	Crea una pista y un usuario para luego enviar una solicitud POST a /usuarios/{dni}/evento pasando los datos del evento a crear
Resultado esperado	Respuesta HTTP 201 CREATED al hacer la petición y la respuesta debe contener el evento creado con el participante que ha hecho la reserva
Estado	✓ testAltaEvento()

Tabla 36: Tabla Prueba 15 Reservar pista

Campo	Detalles
Id	PR16
Historia de usuario	Unirse a evento
Descripción	Verificar que un usuario se pueda unir a un evento existente
Datos de entrada	Dos usuarios, una pista y un evento creado
Acción	Crea una pista, dos usuarios y un evento para luego enviar una solicitud POST a /usuarios/{dni}/eventos/{eventoId}/unirse pasando el DNI del usuario que quiere unirse y el id del evento al que quiere unirse
Resultado esperado	Respuesta HTTP 200 Ok al hacer la petición de unión al evento
Estado	✓ testUnirseEvento()

Tabla 37: Tabla Prueba 16 Unirse a evento

Campo	Detalles
Id	PR17
Historia de usuario	Cancelar Reserva
Descripción	Verificar que un usuario pueda salirse de un evento al que se ha unido
Datos de entrada	Dos usuarios, una pista y un evento creado
Acción	Crea una pista, dos usuarios y un evento al que se unen los usuarios para luego enviar una solicitud DELETE a /usuarios/{dni}/eventos/{eventoId}/salirse pasando el DNI del usuario que quiere salirse y el id del evento al que quiere salirse
Resultado esperado	Respuesta HTTP 200 Ok al hacer la petición de salirse del evento
Estado	✓ testSalirseEvento()

Tabla 38: Tabla Prueba 17 Cancelar reserva

1.1.2 Resultados

Ahora procedemos a mostrar los resultados de todo lo trabajado durante esta iteración.

Ilustración 28: Ilustración Vista de Horarios Administrador

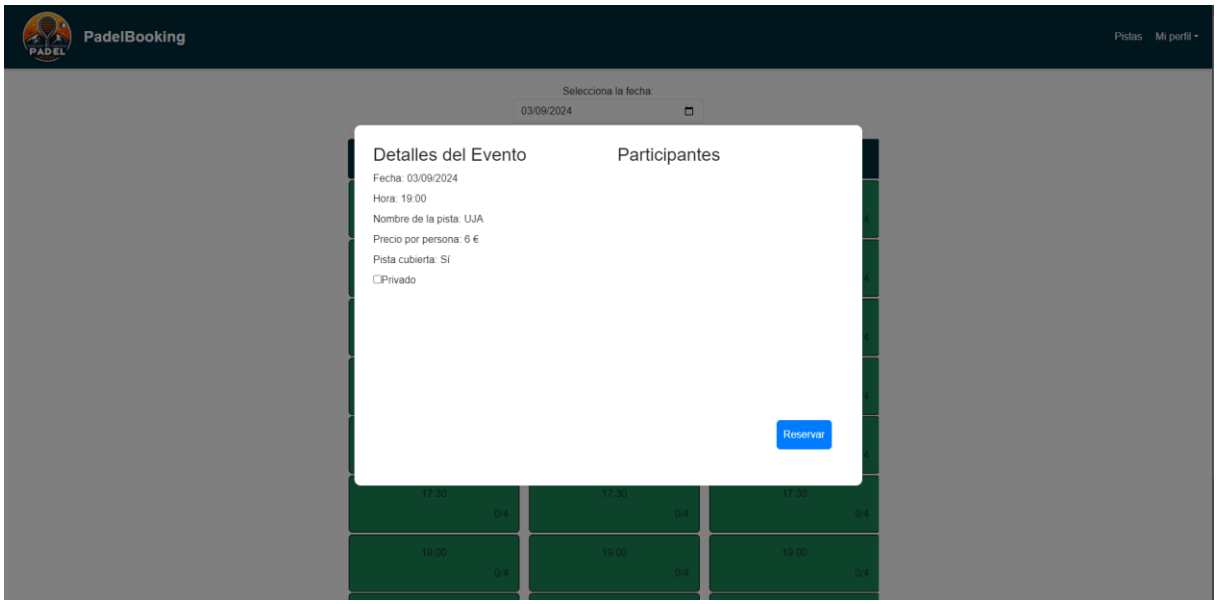


Ilustración 29: Ilustración Vista de Reservar

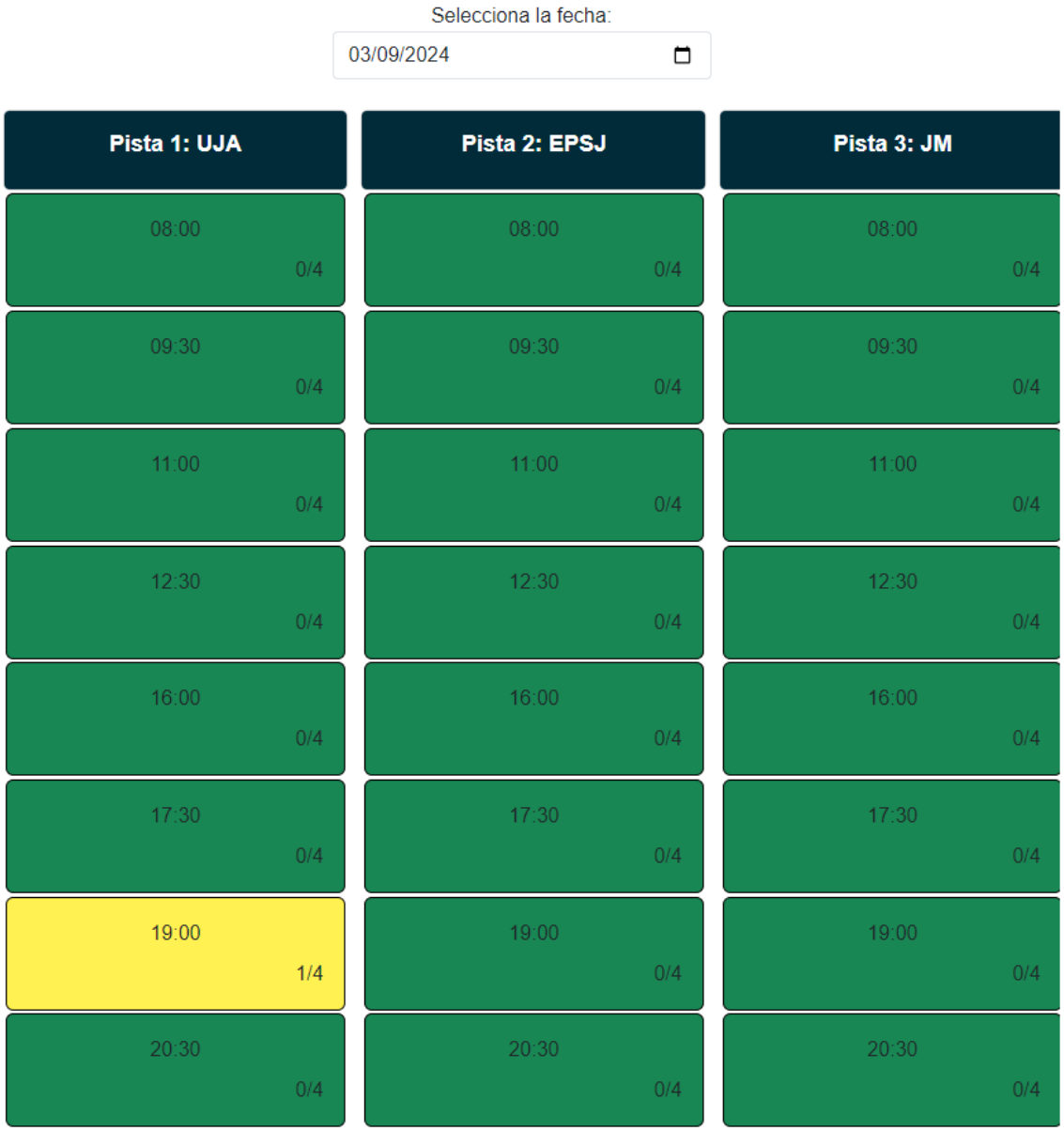


Ilustración 30: Ilustración Vista de Menú Principal con Reserva

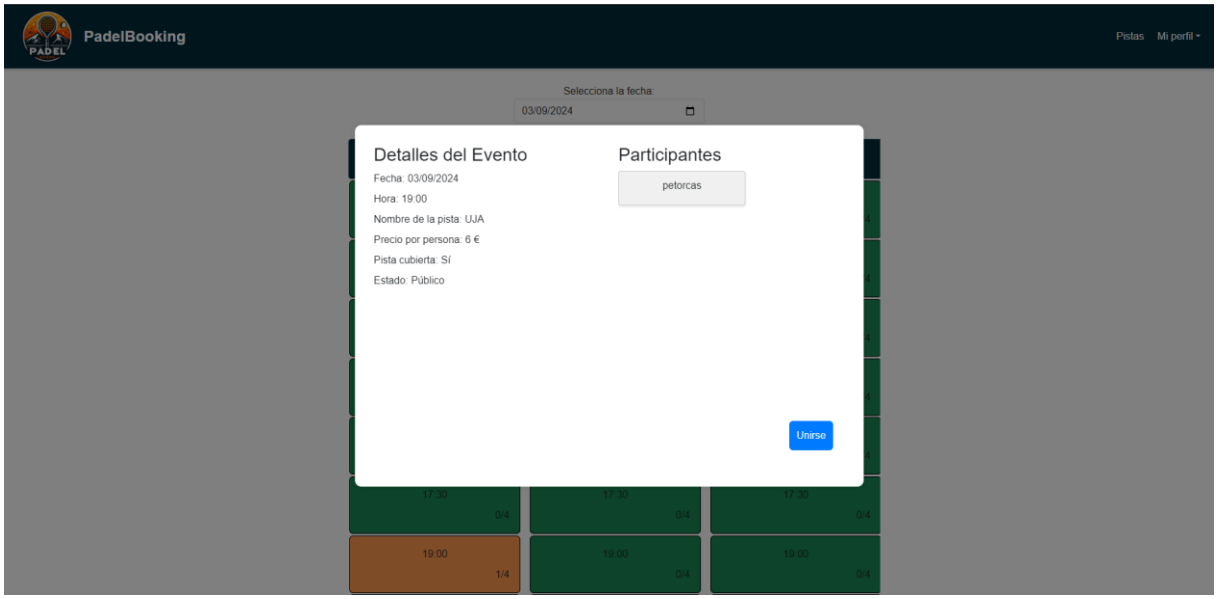


Ilustración 31: Ilustración Vista de Detalles del Evento

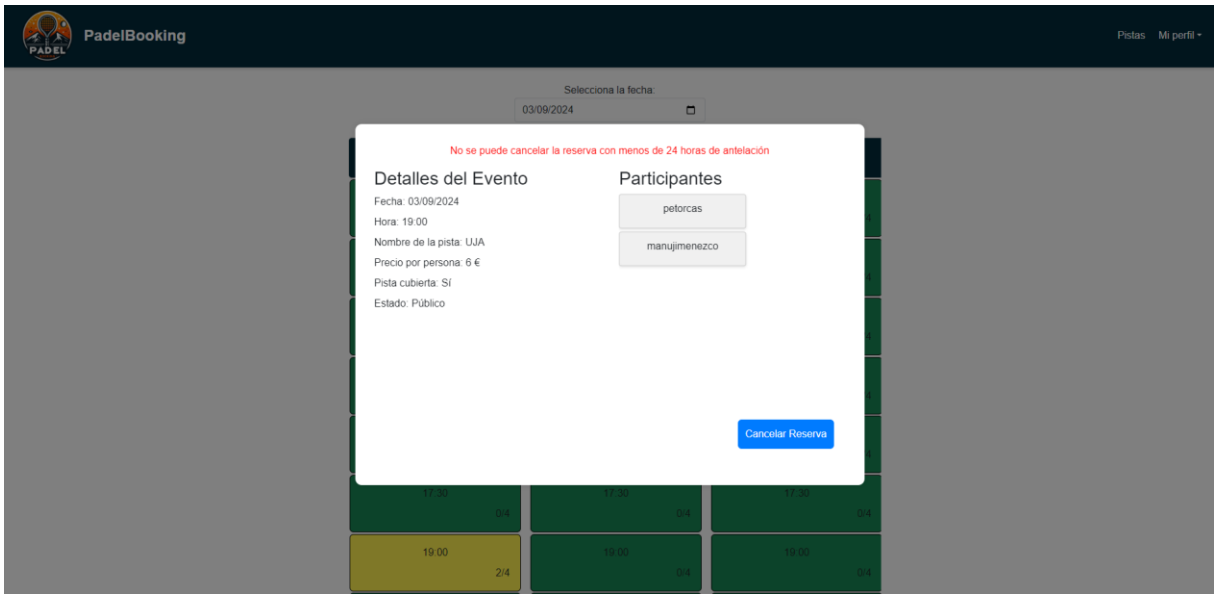


Ilustración 32: Ilustración Vista de Cancelar Reserva

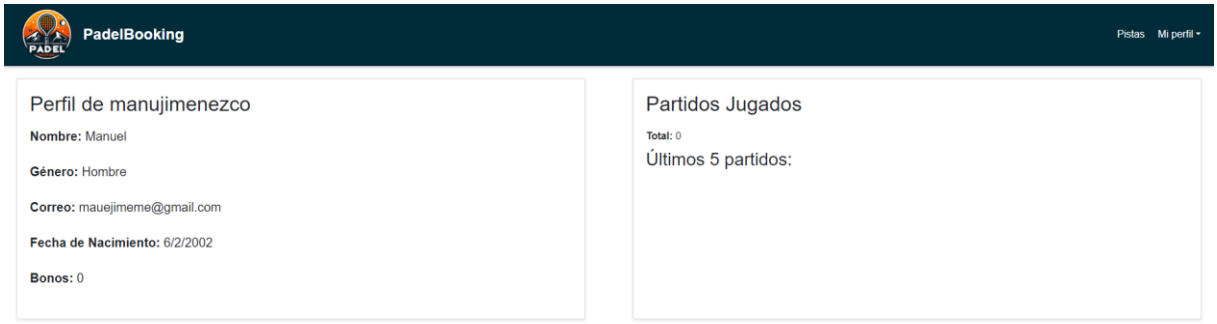


Ilustración 33: Ilustración Vista de Usuario Participante

3.4 Cuarta Iteración

Para la cuarta iteración vamos a enriquecer un poco más el proceso de reserva, pudiendo crear eventos privados a los que solo permita acceder solicitándolo y que el anfitrión sea el que tenga que aceptar las solicitudes. A parte, vamos a crear una vista para mostrar el historial de reservas del usuario en el que se puedan visualizar cuál está todavía por jugar y cual se ha jugado y cuál no se ha terminado de jugar.

3.4.1 Casos de uso

- Mostrar historial: Un usuario podrá acceder a su historial de partidos mostrando cuál finalmente se ha jugado y cuál queda por jugar.
- Enviar solicitud: Un usuario podrá enviar una solicitud a un evento que se ha creado de manera privada para poder unirse.
- Cancelar solicitud: Un usuario que ha realizado una solicitud puede cancelarla con un tiempo de antelación predefinido.
- Aceptar solicitud: El anfitrión del evento privado podrá aceptar solicitudes para que otros usuarios se unan al evento.

Tras la implementación de las historias de usuario de la iteración anterior y para comenzar esta se nos queda el siguiente tablero:

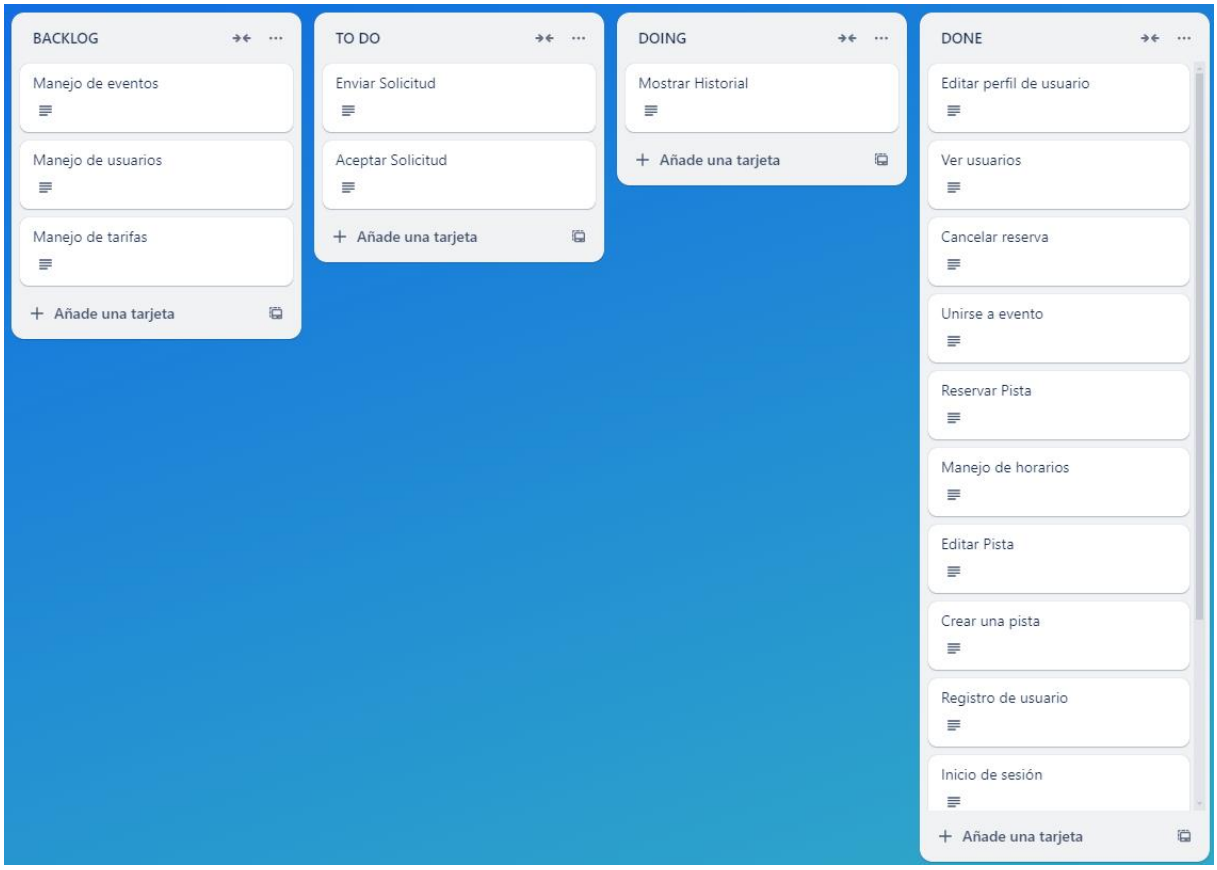


Ilustración 34: Ilustración Tablero Kanban Iteración 4

3.4.2 Implementación

Para empezar con la iteración se procede con el desarrollo de una vista para que el usuario pueda ver un historial de las reservas que ha realizado en la aplicación.

Para ello simplemente creamos en el servicio una consulta en la que se obtengan todos los eventos en los que ha participado el usuario y los mostramos en una tabla con algunos de sus datos ordenados por fecha. Para que el usuario pueda distinguir cuál de ellos se jugaron, no se jugaron o cuáles son las reservas futuras se ha creado una función para determinar el color de fondo de la fila de cada evento.

```
const obtenerEstiloFila = (evento) : {...} | {...}  ⇒ {  Show usages  Eduasso *
    const fechaEvento : Date  = new Date(evento.fecha);
    const fechaActual : Date  = new Date();

    if (fechaEvento < fechaActual && (evento.participantes.length < 4)) {
        return { backgroundColor: '#f1858f' };
    }

    if (fechaEvento > fechaActual && evento.participantes.length === 4) {
        return { backgroundColor: '#c390f1' };
    }

    return {};
};
```

De esta manera, si el evento es anterior a la fecha actual y el número de participantes es inferior a 4 el evento aparece en rojo porque no se consiguió llenar de participantes. Por otra parte, los eventos cuya fecha sea mayor a la actual se mostrarán en morado para saber que son reservas actuales y que tenemos que jugar. Finalmente, si el fondo es blanco es que la reserva se completó con normalidad.

Una vez desarrollado todo lo necesario para el historial de eventos del usuario, seguimos con la implementación de las solicitudes para los eventos privados. Para ello añadimos una relación entre Evento y Usuario que representan las solicitudes que hacen los usuarios a los eventos.

```
@ManyToMany  5 usages
@JoinTable(
        name = "eventoSolicitudes",
        joinColumns = @JoinColumn(name = "idEvento"),
        inverseJoinColumns = @JoinColumn(name = "dniUsuario")
)
private List<Usuario> solicitudes;
```

Para implementar las solicitudes debemos añadir a la función getButton() del Square para que si el evento es privado y todavía no está lleno devuelva el botón de Solicitar y si ya ha solicitado que devuelva el de cancelar solicitud.

```
} else if (evento.participantes.length < datos.maxJugadores && evento.privado === true && !evento.participantes.includes(datos.dniUsuario)) {
    if (evento.solicitudes.includes(datos.dniUsuario)) {
        return (
            <button type="submit" className="btn btn-primary" onClick={handleSubmitCancelarSolicitud} disabled={buttonDisabled}>
                Cancelar Solicitud
            </button>
        );
    } else {
        return (
            <button type="submit" className="btn btn-primary" onClick={handleSubmitSolicitar} disabled={buttonDisabled}>
                Solicitar
            </button>
        );
    }
```

Una vez hechas las solicitudes, estas le aparecerán dentro del evento al anfitrión, que es la persona que hizo la reserva. Si el anfitrión abandona el evento se pondría como anfitrión el siguiente usuario que se unió. Al lado de todas las solicitudes aparecerá un botón que permitirá al anfitrión aceptar la solicitud del usuario al evento.

3.4.3 Pruebas

Como en cada iteración, aquí están las pruebas realizadas para comprobar la funcionalidad de todas las novedades incorporadas.

Campo	Detalles
Id	PR18
Historia de usuario	Enviar solicitud
Descripción	Verificar que un usuario pueda solicitar unirse a un evento que fue creado por otro usuario
Datos de entrada	Dos usuarios, una pista y un evento creado
Acción	Crea una pista, dos usuarios y un evento al que un usuario envía una solicitud DELETE a /usuarios/{dni}/eventos/{eventoId}/solicitar pasando el DNI del usuario que hace la solicitud y el id del evento al que quiere solicitar
Resultado esperado	Respuesta HTTP 200 Ok al hacer la petición de solicitar unirse al evento y el evento tiene una solicitud y es del usuario que la ha realizado

Estado	✓ testSolicitarEvento()

Tabla 39: Tabla Prueba 18 Enviar solicitud

Campo	Detalles
Id	PR19
Historia de usuario	Cancelar solicitud
Descripción	Verificar que un usuario pueda cancelar una solicitud de unirse a un evento antes de que el anfitrión acepte la solicitud
Datos de entrada	Dos usuarios, una pista y un evento creado
Acción	Crea una pista, dos usuarios y un evento al que un usuario envía una solicitud. Posteriormente el mismo usuario envía una solicitud PUT a /usuarios/{dni}/eventos/{id}/cancelarSolicitud al que le pasa el DNI del usuario que cancela y el id del evento.
Resultado esperado	Respuesta HTTP 200 Ok al hacer la petición de cancelar la solicitud al evento y ahora el número de solicitudes del evento es 0
Estado	✓ testCancelarSolicitud()

Tabla 40: Tabla Prueba 19 Cancelar solicitud

Campo	Detalles
Id	PR20
Historia de usuario	Aceptar solicitud
Descripción	Verificar que el anfitrión de un evento pueda aceptar la solicitud de otro usuario para unirse al evento
Datos de entrada	Dos usuarios, una pista y un evento creado
Acción	Crea una pista, dos usuarios y un evento al que un usuario envía una solicitud. Posteriormente, el anfitrión hace una petición PUT a /usuarios/{dni}/eventos/{id}/aceptarSolicitud en el que se pasa el DNI del anfitrión como parámetro
Resultado esperado	Respuesta HTTP 200 Ok al hacer la petición de aceptar solicitud y comprobar que el usuario al que se ha aceptado ahora está dentro del evento
Estado	✓ testAceptarSolicitud()

Tabla 41: Tabla Prueba 20 Aceptar solicitud

3.4.4 Resultados

Una vez más, mostramos los avances en nuestra aplicación de esta iteración.

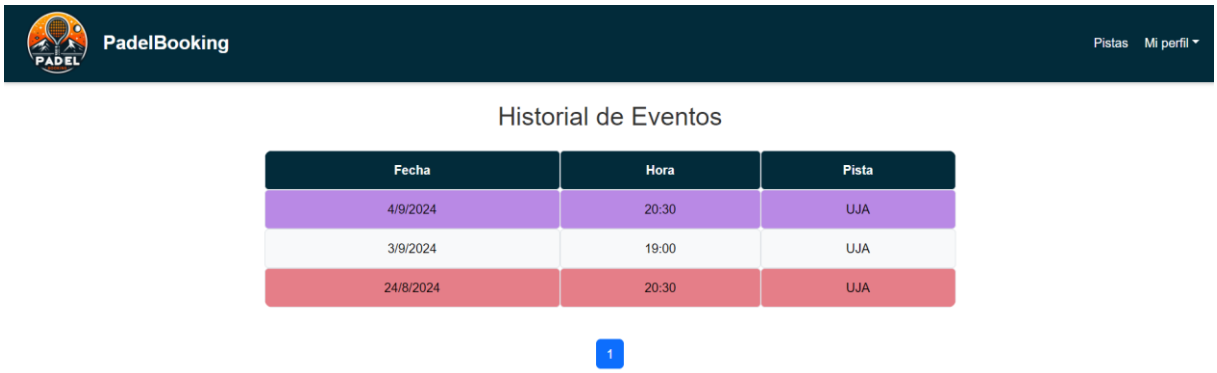


Ilustración 35: Ilustración Vista de Historial de Usuario

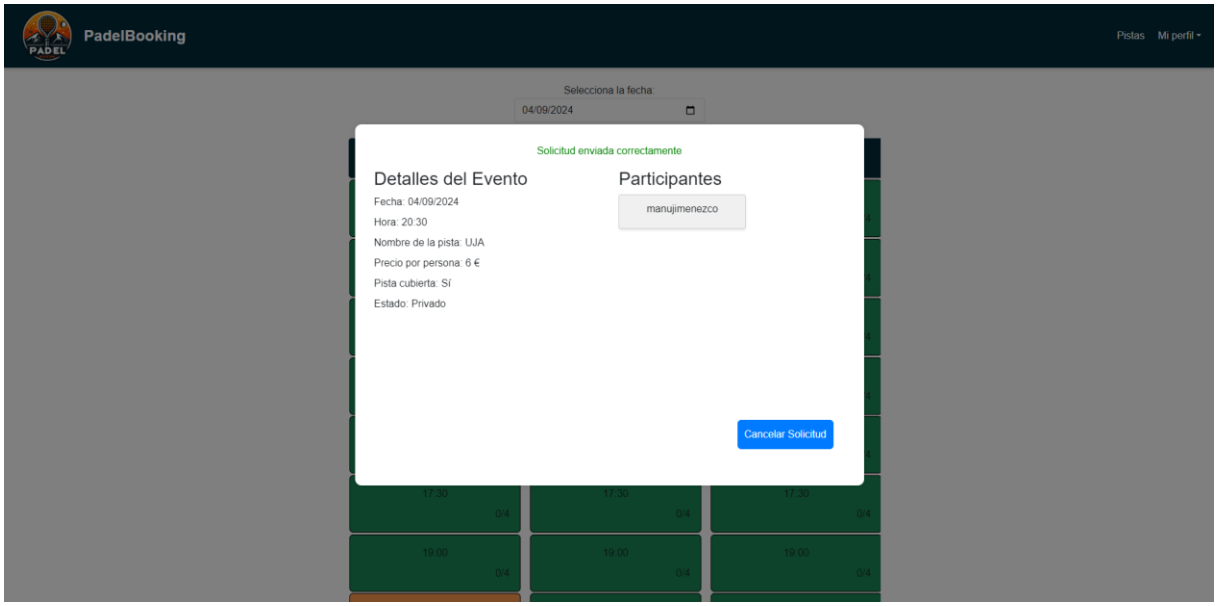


Ilustración 36: Ilustración Vista de Enviar Solicitud

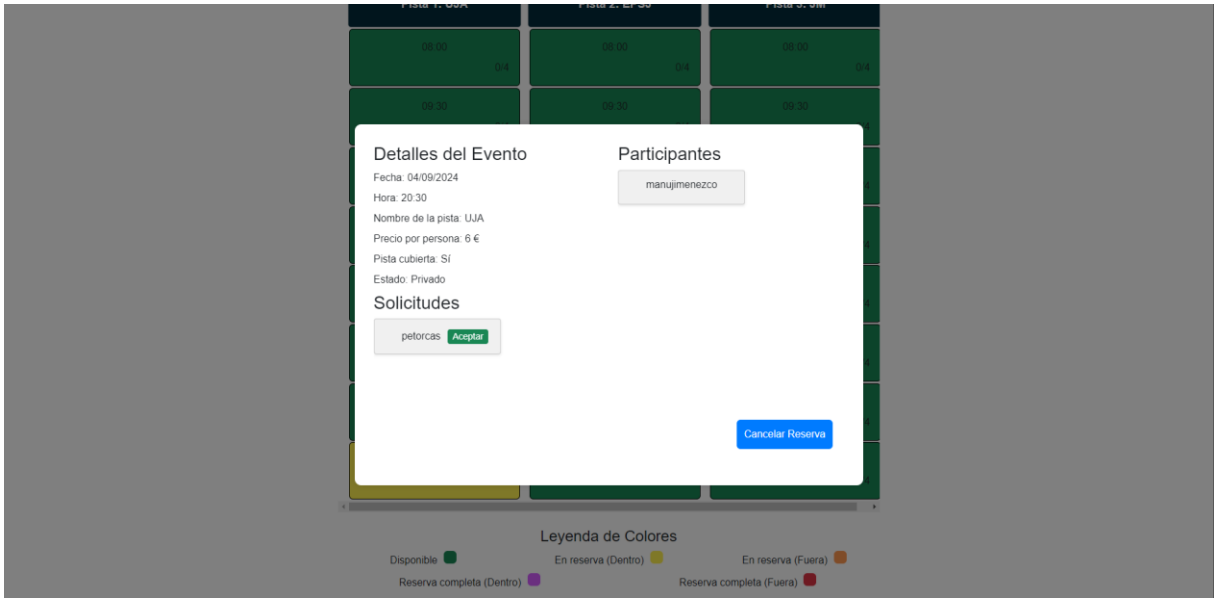


Ilustración 37: Ilustración Vista de Aceptar Solicitud

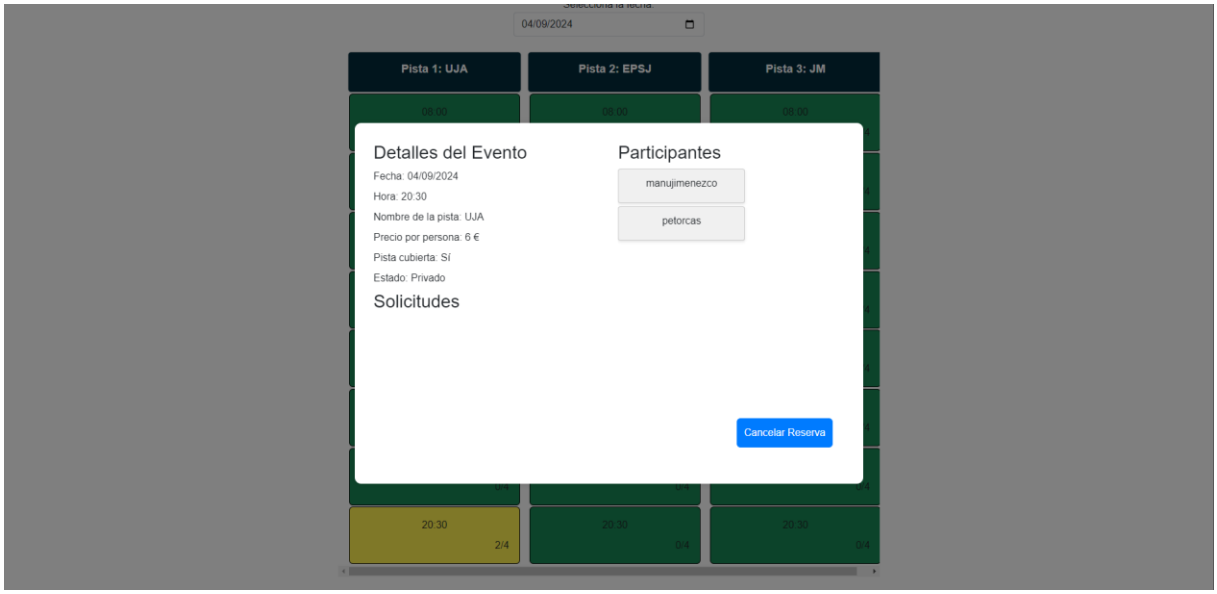


Ilustración 38: Ilustración Vista de Solicitud Aceptada

3.5 Quinta Iteración

En esta iteración vamos a implementar muchas funcionalidades para el administrador, como el manejo de eventos, el manejo de usuarios y el manejo de tarifas. De esta manera el administrador tendrá acceso y podrá manejar las entidades de la aplicación cuando lo crea conveniente.

3.5.1 Casos de uso

- Manejo de eventos: El administrador tendrá acceso a las reservas que se realizan y podrá eliminar eventos creados por error.
- Manejo de usuarios: El administrador tendrá acceso a los perfiles de los usuarios y podrá modificar sus datos.
- Manejo de tarifas: El administrador podrá crear tarifas para según el tipo de pista, turno de juego y durante periodos de tiempo.
- Sancionar usuario: El administrador podrá sancionar a un usuario sin jugar por un período de tiempo.
- Otorgar bonos: El administrador tendrá la capacidad de otorgar bonos que el usuario podrá utilizar para realizar reservas gratuitamente. Esto está pensado para que los usuarios que más reservas realicen durante un mes sean premiados de alguna manera y haya incentivos por jugar a parte de por diversión.

Nuevamente, con la finalización de la iteración anterior y el comienzo de esta, el tablero de Kanban queda de la siguiente manera.

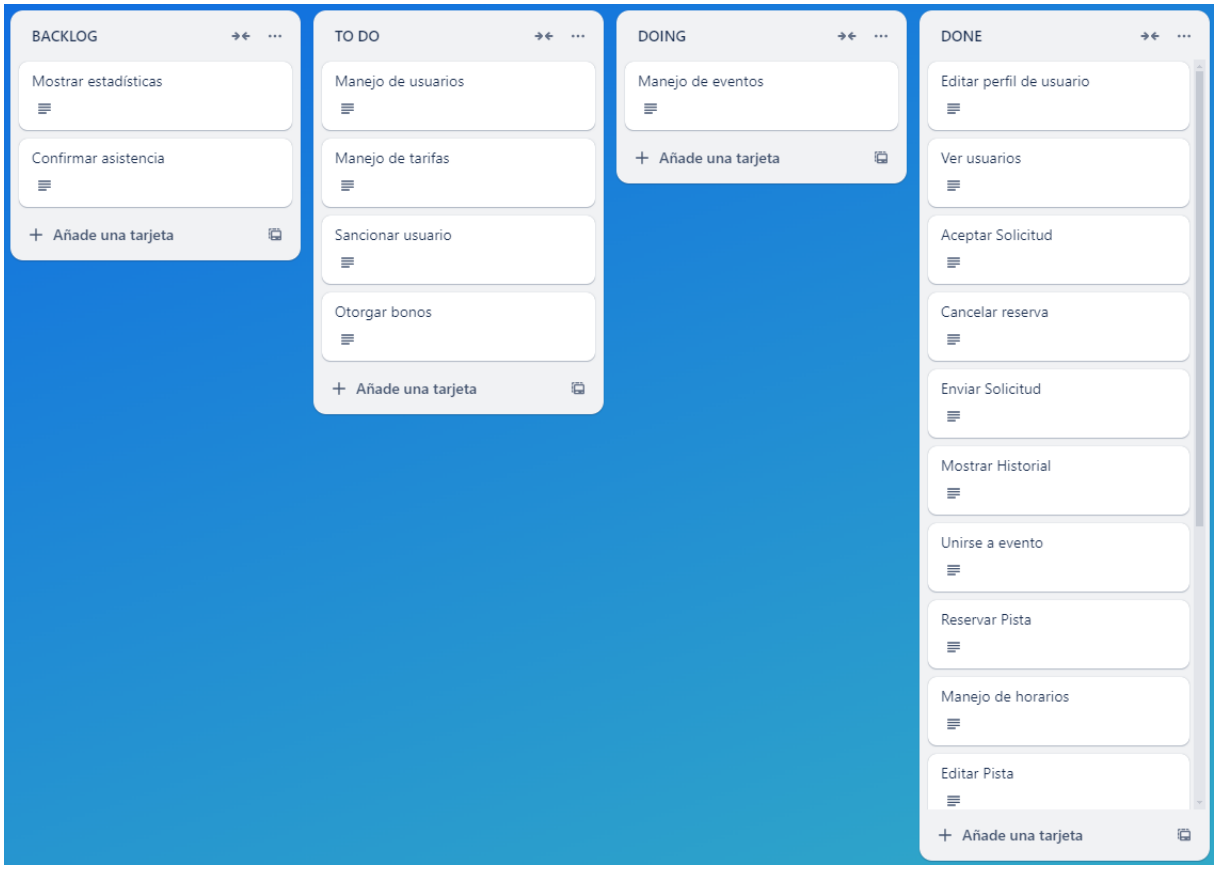


Ilustración 39: Ilustración Tablero Kanban Iteración 5

3.5.2 Implementación

Comenzamos con la implementación de una vista para el administrador que nos permite visualizar los eventos de una fecha diferenciando entre los que ya se han jugado y los que no. Para ello simplemente hacemos una petición que hace una consulta de los eventos para las pistas en la fecha seleccionada y se muestran comprobando la hora para saber si se han jugado o no. Disponemos de un botón de eliminar el evento en el caso de que un usuario lo haya reservado por erro y por tiempo no deje cancelarlo.

A la hora de eliminar el evento hay que tener en cuenta el eliminar los participantes y solicitudes asociados al mismo. Para ello creamos en la clase Evento este @PreRemove.

```java
@PreRemove  Eduasso
private void preRemove() {
    for (Usuario usuario : participantes) {
        usuario.removeEvento(this);
    }
    participantes.clear();

    for (Usuario usuario : solicitudes){
        usuario.removeSolicitud( evento: this);
    }
    solicitudes.clear();
}
```

Para complementar esto, podemos pulsar sobre el evento para visualizar los datos del mismo y ver los participantes. Aprovechando que tenemos que implementar una vista para visualizar el perfil del usuario, pulsando en el participante podemos acceder a los datos del usuario con un formulario similar al de edición de perfil de los usuarios normales. Además, desde ahí podemos sancionar al usuario aportando unas fechas de inicio, fin y el motivo de la sanción. A parte, al sancionar al usuario se le añade una deuda a pagar con el importe que le ha costado esa reserva. Previamente a esto hemos creado la entidad Sanción que da bastante juego ya que si un jugador está sancionado no podrá hacer reservas durante ese período de tiempo.

Siguiendo con el manejo de usuarios hemos creado una vista para que el usuario le aparezca una tabla paginada con todos los usuarios a los que puede buscar por nick o DNI. Para ello se hace una consulta a la base de datos filtrando con la búsqueda realizada de la siguiente manera.

```
@Transactional(propagation = Propagation.SUPPORTS, readOnly = true) 1 usage  Eduasso *
public Page<Usuario> buscarUsuarios(String search, Pageable pageable) {
    String queryStr = "SELECT u FROM Usuario u WHERE LOWER(u.dni)" +
            " LIKE LOWER(:search) OR LOWER(u.nick) LIKE LOWER(:search)";
    TypedQuery<Usuario> query = em.createQuery(queryStr, Usuario.class);
    query.setParameter( s: "search",  o: "%" + search + "%");

    int totalRecords = ((Long) em.createQuery( s: "SELECT COUNT(u) FROM Usuario u " +
                    "WHERE LOWER(u.dni) LIKE LOWER(:search) OR LOWER(u.nick) LIKE LOWER(:search)")
            .setParameter( s: "search",  o: "%" + search + "%")
            .getSingleResult()).intValue();

    query.setFirstResult((int) pageable.getOffset());
    query.setMaxResults(pageable.getPageSize());

    List<Usuario> usuarios = query.getResultList();
    return new PageImpl<>(usuarios, pageable, totalRecords);
}
```

Pulsando sobre un usuario podemos acceder al perfil del mismo en el que implementaremos una función para otorgar bonos y otra para quitarle la deuda. También si el usuario está sancionado podrá quitársela desde el propio perfil.

Para darle un poco más de juego, se han definido unas tarifas. Creamos la entidad Tarifa que es bastante sencilla, las variaciones que existen son entre pista cubierta/no cubierta y el turno mañana/tarde. Creamos una vista para que pueda crear las tarifas aportando los datos pertinentes. Tenemos en cuenta que la tarifa que se usa para cada pista es la que cumple con la fecha de la misma, es decir, si una tarifa para el turno de mañana y en pista cubierta comienza el 17/09/2024 y otra para el mismo turno y tipo de pista el 19/09/2024 la tarifa que se aplicará el día 17 y 18 será la primera, pero a partir del 19 se aplica la siguiente. Para cumplir con el requisito de las fechas se hace la siguiente consulta:

```java
@Transactional(propagation = Propagation.SUPPORTS, readOnly = true) 3 usages  Eduasso *
public Optional<Tarifa> buscarTarifaMasRecientePorTurnoYCubierta(Tarifa.Turno turno, boolean cubierta, LocalDate fecha) {
    List<Tarifa> tarifas = em.createQuery(
                    s: "SELECT t FROM Tarifa t WHERE t.turno = :turno AND " +
                            "t.cubierta = :cubierta AND t.fechaInicio <= :fecha ORDER BY t.fechaInicio DESC",
                    Tarifa.class)
            .setParameter( s: "turno", turno)
            .setParameter( s: "cubierta", cubierta)
            .setParameter( s: "fecha", fecha)
            .setMaxResults(1)
            .getResultList();
    if (tarifas.isEmpty()) {
        return Optional.empty();
    } else {
        return Optional.of(tarifas.get(0));
    }
}
```

Además de todo esto, hemos implementado la posibilidad de poder editar y eliminar las tarifas ya existentes.

3.5.3 Pruebas

Seguimos con la verificación de todas las nuevas funciones creadas para el desarrollo de todas estas nuevas funcionalidades.

Campo	Detalles
Id	PR21
Historia de usuario	Manejo de eventos
Descripción	Verificar que un administrador pueda eliminar un evento y que el evento eliminado ya no aparezca en la lista de eventos
Datos de entrada	Un usuario, una pista y un evento creado
Acción	Crea una pista, un usuario y un evento. Posteriormente, se hace una petición DELETE a /admin/eventos/{id}/eliminar en el que se pasa el id del evento como parámetro
Resultado esperado	Respuesta HTTP 200 Ok al hacer la petición de eliminar el evento
Estado	✓ testEliminarEvento()

Tabla 42: Tabla Prueba 21 Manejo de eventos

Campo	Detalles
Id	PR22
Historia de usuario	Manejo de eventos

Descripción	Verificar que el sistema retorne correctamente los eventos asociados a una lista de pistas específicas en una fecha dada
Datos de entrada	Dos usuarios, dos pistas y dos eventos creado
Acción	Crea dos pistas, dos usuarios y dos eventos. Posteriormente, se hace una petición GET a /usuarios/eventos en el que se pasa los id de las pistas y la fecha como parámetros para obtener los eventos
Resultado esperado	Respuesta HTTP 200 Ok al hacer la petición de obtener los eventos y una lista con los eventos que coinciden con esas pistas y esa fecha
Estado	✓ testGetEventosPorPistaFecha()

Tabla 43: Tabla Prueba 22 Manejo de eventos

Campo	Detalles
Id	PR23
Historia de usuario	Sancionar usuario
Descripción	Verificar que el administrador puede sancionar al usuario y comprobar que su deuda se actualiza correctamente
Datos de entrada	Un usuario y la sanción que se le va a aplicar
Acción	Crea un usuario para después hacer una petición POST a /admin/sancionar/{dni} en el que se pasa el DNI del sancionado y la deuda que se le va a aplicar
Resultado esperado	Respuesta HTTP 201 CREATED al hacer la petición de sancionar y el usuario con la deuda actualizada
Estado	✓ testSancionarUsuario()

Tabla 44: Tabla Prueba 23 Sancionar usuario

Campo	Detalles
Id	PR24
Historia de usuario	Sancionar usuario
Descripción	Verificar que el administrador puede eliminar la sanción aplicada al usuario
Datos de entrada	Un usuario y la sanción que se le va a aplicar para posteriormente eliminarla
Acción	Crea un usuario para sancionarlo y finalmente hacer una petición DELETE a /admin/{dni}/sanción/eliminar en el que

	se pasa el DNI del sancionado y el id de la sanción
Resultado esperado	Respuesta HTTP 200 OK al hacer la petición de eliminar la sanción comprobar que el usuario no tiene sanción
Estado	✓ testEliminarSancion()

Tabla 45: Tabla Prueba 24 Sancionar usuario

Campo	Detalles
Id	PR25
Historia de usuario	Otorgar bonos
Descripción	Verificar que el administrador puede otorgar bonos a un usuario
Datos de entrada	Un usuario al que se le hace la prueba de los bonos
Acción	Crea un usuario para hacer una petición PUT a /admin/{dni}/otorgarBonos en el que se pasa la cantidad de bonos y el DNI del usuario al que se le va a aplicar
Resultado esperado	Comprobar que se obtienen los bonos correctamente según se otorgan y se quitan
Estado	✓ testFuncionamientoBonos()

Tabla 46: Tabla Prueba 25 Otorgar bonos

Campo	Detalles
Id	PR26
Historia de usuario	Manejar tarifas
Descripción	Verificar que el administrador puede crear una nueva tarifa
Datos de entrada	Una tarifa que va a ser creada
Acción	Hace una petición POST a /admin/tarifa con la tarifa a crear
Resultado esperado	Respuesta HTTP 201 CREATED y la tarifa que se ha creado
Estado	✓ testAltaTarifa()

Tabla 47: Tabla Prueba 26 Manejar tarifas

Campo	Detalles
Id	PR27
Historia de usuario	Manejar tarifas

Descripción	Verificar que el administrador obtiene las tarifas correctamente diferenciando entre turnos y tipos de pista
Datos de entrada	Cuatro tarifas con distintas propiedades entre sí
Acción	Tras crear las tarifas realiza petición GET a /admin/obtenerTarifas en el que se pasa una fecha
Resultado esperado	Respuesta HTTP 200 OK y una lista de las distintas tarifas válidas para esa fecha
Estado	✓ testGetTarifas()

Tabla 48: Tabla Prueba 27 Manejar tarifas

Campo	Detalles
Id	PR28
Historia de usuario	Manejar tarifas
Descripción	Verificar que el administrador edita una tarifa correctamente
Datos de entrada	Una tarifa que posteriormente será editada
Acción	Tras crear las tarifas realiza petición PUT a /admin/tarifa/{id}/editar en el que se pasa la tarifa editada con el id de la misma
Resultado esperado	Respuesta HTTP 200 OK y una lista de las distintas tarifas válidas para esa fecha que coincide con las características de la tarifa modificada
Estado	✓ testEditarTarifa()

Tabla 49: Tabla Prueba 28 Manejar tarifas

Campo	Detalles
Id	PR29
Historia de usuario	Manejar tarifas
Descripción	Verificar que el administrador elimina correctamente una tarifa
Datos de entrada	Dos tarifas con las distintas características
Acción	Tras crear las tarifas realiza petición DELETE a /admin/tarifa/{id}/eliminar en el que se pasa el id de la misma
Resultado esperado	Respuesta HTTP 200 OK y una lista de las distintas tarifas válidas para esa fecha que antes de eliminarla eran dos, pero ahora solo queda una creada y cumple con las

	características de la que no se ha eliminado
Estado	✓ testEliminarTarifa()

Tabla 50: Tabla Prueba 29 Manejar tarifas

Resultados

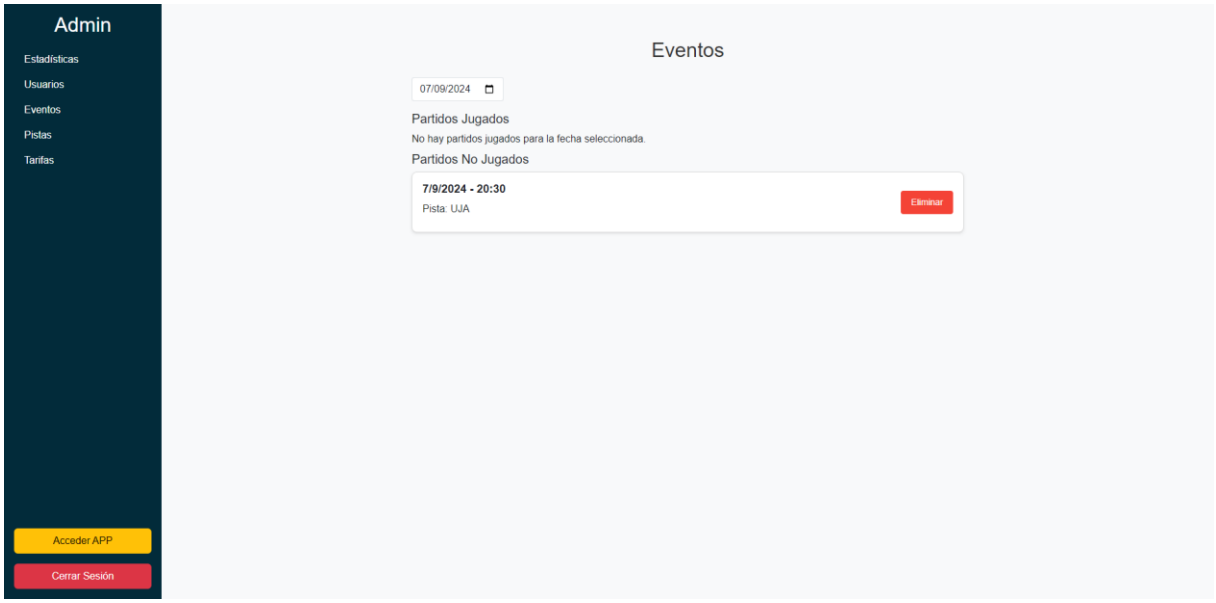


Ilustración 40: Ilustración Vista de Manejar Eventos Administrador

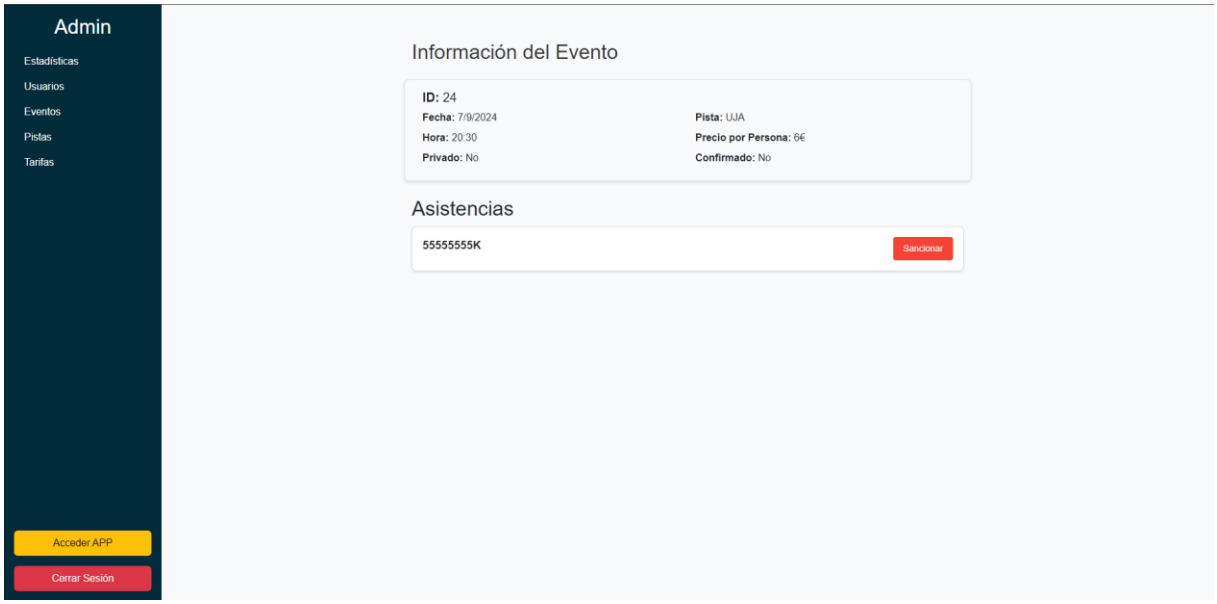


Ilustración 41: Ilustración Vista Información Evento Administrador

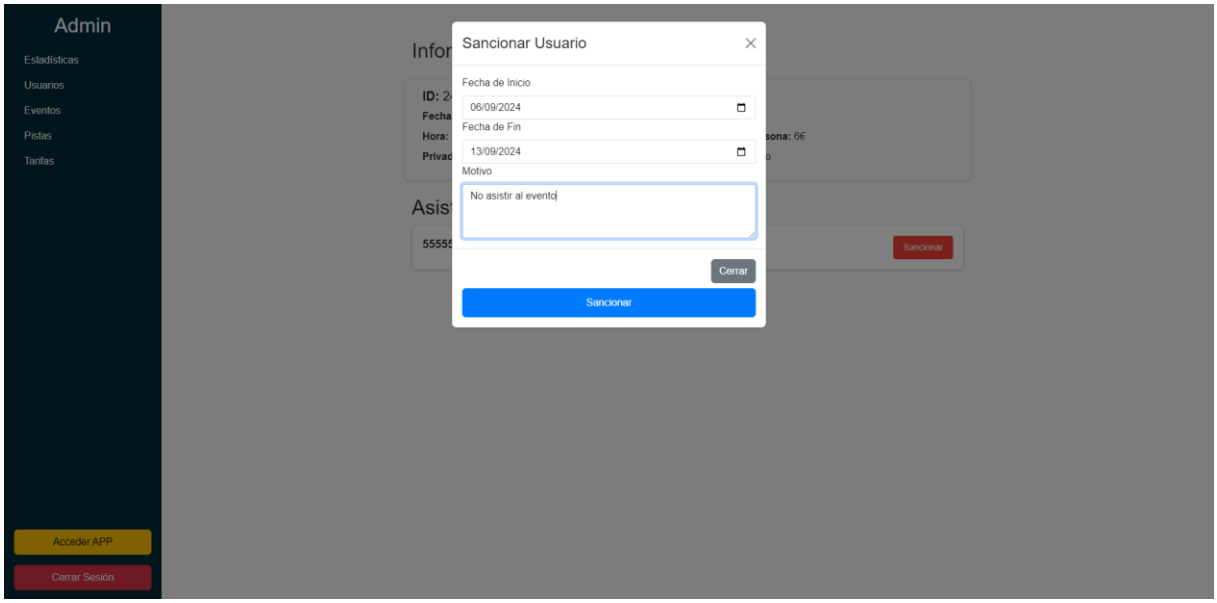


Ilustración 42: Ilustración Vista de Sancionar Usuario

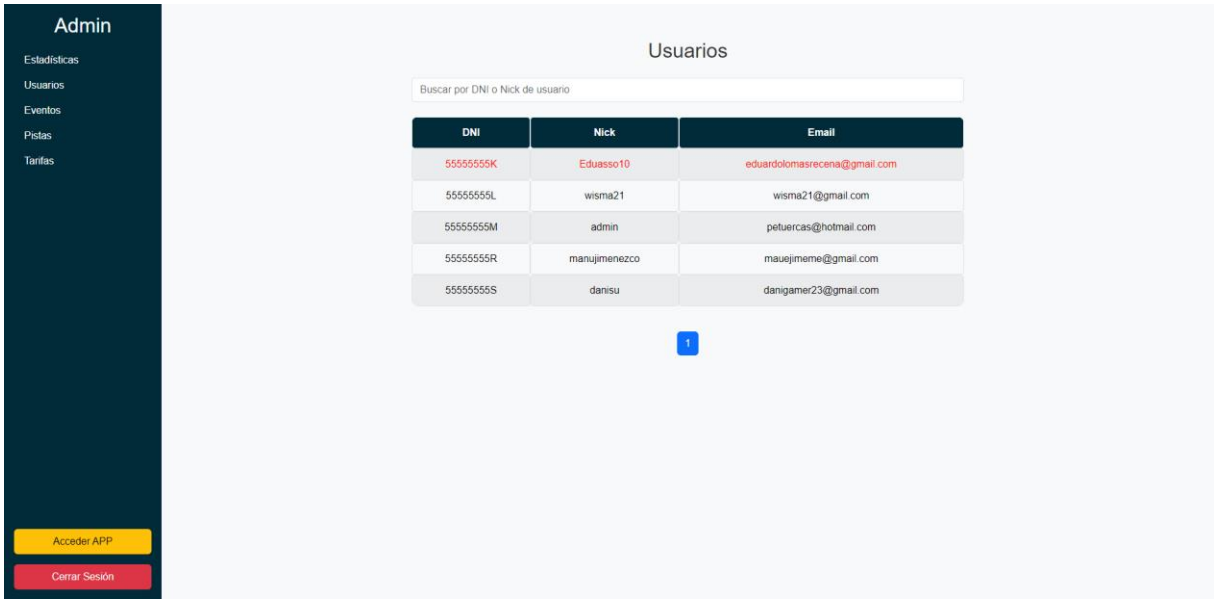


Ilustración 43: Ilustración Vista de Manejar Usuarios Administrador

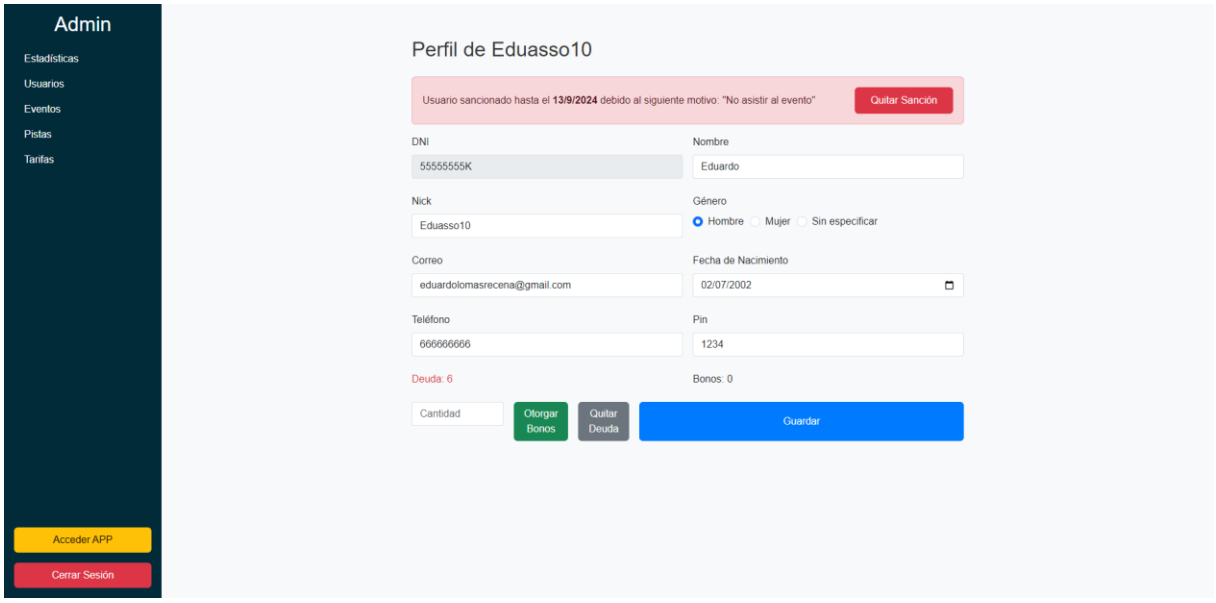


Ilustración 44: Ilustración Vista de Perfil Usuario Sancionado Administrador

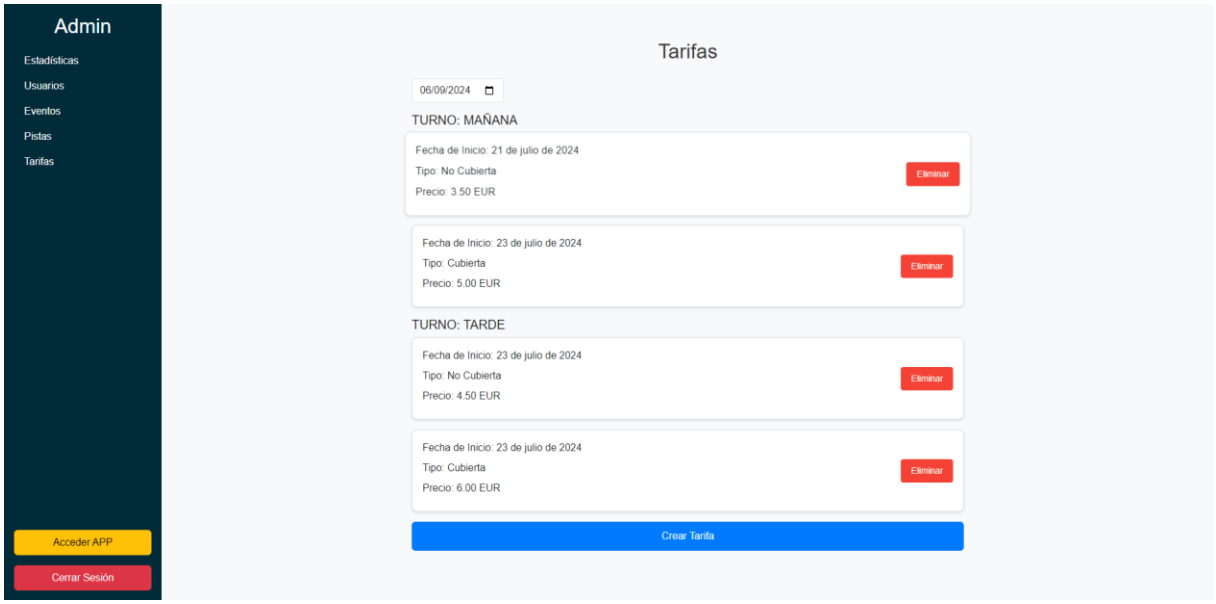


Ilustración 45: Ilustración Vista de Manejar Tarifas Administrador

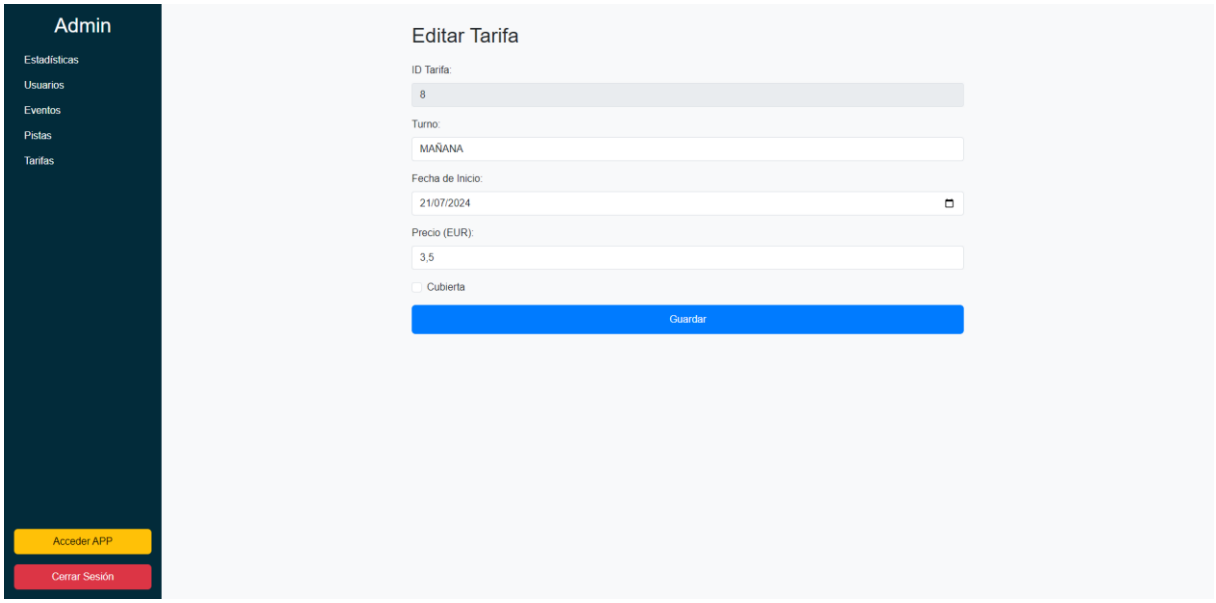


Ilustración 46: Ilustración Vista de Editar Tarifa Administrador

3.6 Sexta Iteración

Para esta última iteración vamos a implementar una vista de estadísticas para que el administrador tenga una información extra y pueda sacar conclusiones de la

misma y, además, vamos a crear un frontend muy simple que permita a los usuarios confirmar su asistencia al evento. Esto está pensado para que cuando se llegue a las instalaciones se confirme su asistencia al evento. De esta manera se podrá identificar más fácilmente si un usuario no ha acudido y se le podrá sancionar por ello.

3.6.1 Casos de uso

- Mostrar estadísticas: El administrador tendrá una vista para visualizar algunas estadísticas interesantes de los usuarios y los eventos,
- Confirmar asistencia: El usuario podrá confirmar su asistencia al evento a la hora de entrar a las instalaciones mediante un simple frontend introduciendo su pin.

Así quedaría nuestro tablero Kanban justo al comenzar la última iteración.

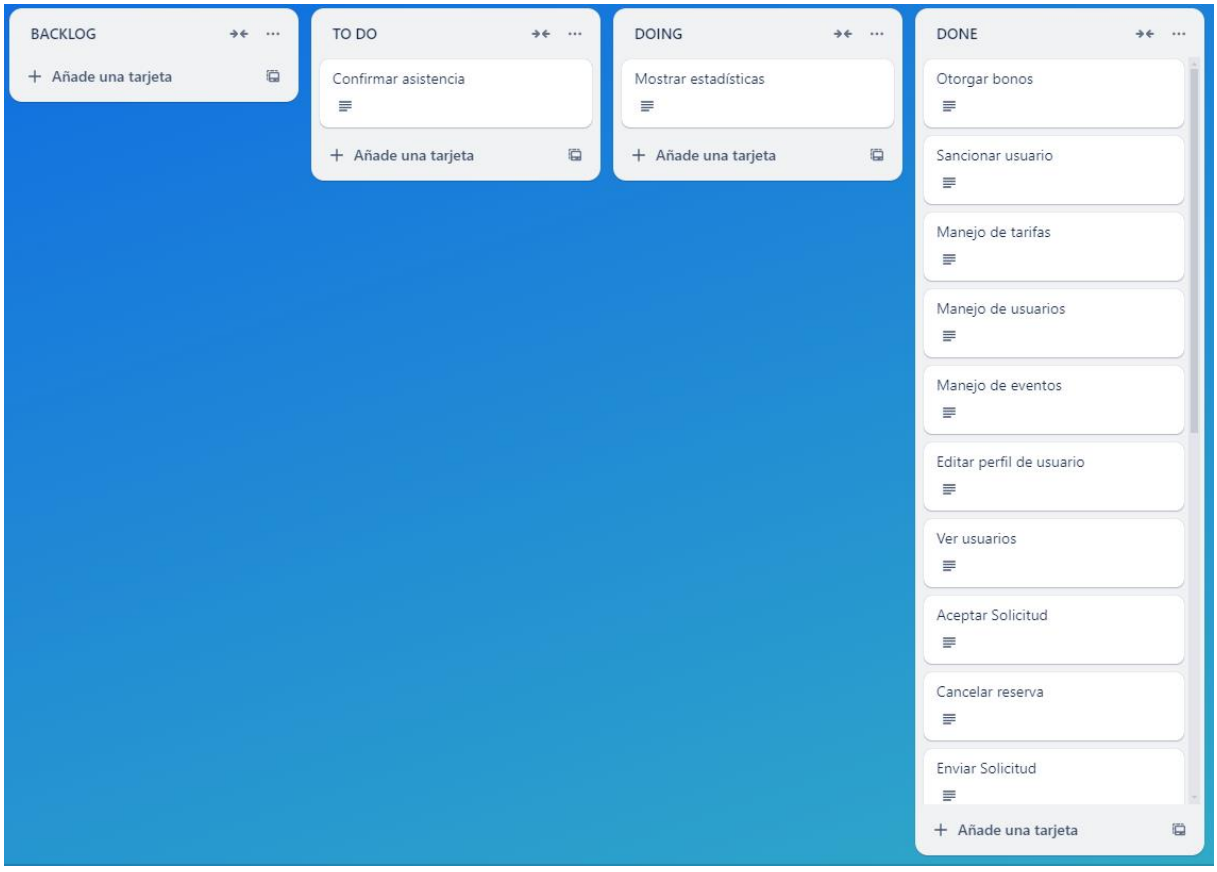


Ilustración 47: Ilustración Tablero Kanban Iteración 6

3.6.2 Implementación

Para implementar la visualización de estadísticas hemos hecho uso de “react-chartjs-2” que nos permite crear gráficos interactivos. Gracias a esto creamos los siguientes gráficos:

- Un gráfico de líneas para el número de partidos jugados durante el año.
- Un gráfico de barras para los usuarios con más partidos en un mes seleccionado.
- Un gráfico circular para las pistas más usadas en el mes seleccionado.

Para ello hemos creado unas funciones en el repositorio para obtener un Mapa en la que se guarden para cada mes el número de partidos jugados.

```
public List<Object[]> contarEventosPorMes(Year year) {  1 usage   Eduasso
    return em.createQuery(
                    s: "SELECT FUNCTION('MONTH', e.fecha) as mes, COUNT(e) as total " +
                            "FROM Evento e " +
                            "WHERE e.confirmado = true AND FUNCTION('YEAR', e.fecha) = :year " +
                            "GROUP BY FUNCTION('MONTH', e.fecha) " +
                            "ORDER BY FUNCTION('MONTH', e.fecha)",
                    Object[].class
            )
            .setParameter( s: "year", year.getValue())
            .getResultList();
}
```

También hemos hecho una consulta para obtener un mapa con los usuarios que más partidos han jugado en un mes seleccionado.

```
public List<Object[]> contarEventosPorUsuarioYMes(YearMonth month) {  1 usage   Eduasso
    return em.createQuery(
                    s: "SELECT p.dni, COUNT(e) as total " +
                            "FROM Evento e JOIN e.participantes p " +
                            "WHERE e.confirmado = true AND FUNCTION('YEAR', e.fecha) = :year AND FUNCTION('MONTH', e.fecha) = :month " +
                            "GROUP BY p.dni " +
                            "ORDER BY total DESC",
                    Object[].class
            )
            .setParameter( s: "year", month.getYear())
            .setParameter( s: "month", month.getMonthValue())
            .setMaxResults(10)
            .getResultList();
}
```

Por último, hemos realizado una consulta en la que se obtiene un mapa con las pistas y el número de partidos jugados por cada pista para el mes seleccionado.

```
public List<Object[]> contarEventosPorPistaYMes(YearMonth month) {  1 usage   Eduasso
    return em.createQuery(
                    s: "SELECT e.pista.idPista, COUNT(e) as total " +
                            "FROM Evento e " +
                            "WHERE e.confirmado = true AND FUNCTION('YEAR', e.fecha) = :year AND FUNCTION('MONTH', e.fecha) = :month " +
                            "GROUP BY e.pista.idPista " +
                            "ORDER BY total DESC",
                    Object[].class
            )
            .setParameter( s: "year", month.getYear())
            .setParameter( s: "month", month.getMonthValue())
            .getResultList();
}
```

Una vez creada la vista que permite al administrador visualizar estadísticas, procedemos a crear un simple frontend a parte del principal que está pensado para estar ejecutado en una Tablet en las instalaciones para que antes de jugar se pueda confirmar la asistencia de un usuario al evento. Para ello se muestran las pistas, el usuario tiene que seleccionar la pista en la que se juega y si hay un evento en esa pista, aparecerá para poder seleccionarlo. Debajo aparecerán los usuarios que participan en el evento, el usuario deberá seleccionarse e introducir su pin para confirmar la asistencia.

Hemos creado una entidad Asistencia que contiene un atributo booleano “asistio” que nos va a servir para identificar si un usuario ha confirmado su asistencia. Las asistencias se crean cada vez que un usuario se une a un evento y si el usuario se sale del evento, la asistencia se elimina.

Algo que hemos realizado para evitar “trampas” a la hora de confirmar la asistencia es mostrar el evento cuando queden 30 min antes de que empiece y 15 minutos después por si alguien llega tarde. De esta manera evitamos que un usuario acuda a las instalaciones por la mañana, confirme su asistencia al evento de las 20:00 y cuando llegue la hora de jugar no aparezca, pero si tenga la asistencia confirmada.

Si hay eventos, pero todavía no se pueden mostrar, se muestra un contador para ver el tiempo restante para que aparezca el siguiente evento. Si no hay eventos para ese día en esa pista, se muestra un mensaje que informa de que no hay eventos para esa pista.

3.6.3 Pruebas

De nuevo, se muestran las últimas pruebas relacionadas con la última iteración.

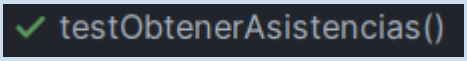
Campo	Detalles
Id	PR30
Historia de usuario	Confirmar asistencia
Descripción	Verificar que se obtienen las personas participantes del evento que necesitan confirmar su asistencia
Datos de entrada	Dos usuarios, una pista y un evento
Acción	Tras unirse los dos usuarios al evento se hace una petición GET a /asistencias/evento/asistencias pasando el id del evento para obtener las asistencias pertenecientes a dicho evento
Resultado esperado	Respuesta HTTP 200 OK y una lista de las asistencias que hay para ese evento
Estado	

Tabla 51: Tabla Prueba 30 Confirmar asistencia

Campo	Detalles
Id	PR31
Historia de usuario	Confirmar asistencia
Descripción	Verificar que un usuario puede confirmar su asistencia a un evento
Datos de entrada	Dos usuarios, una pista y un evento
Acción	Tras unirse los dos usuarios al evento se hace una petición PUT a /asistencias/evento/asistencia/confirmar en la que se pasa el id del evento, el DNI del usuario y el pin de confirmación

Resultado esperado	Respuesta HTTP 200 OK y la asistencia con el booleano en true para confirmar su validez
Estado	✓ testConfirmarAsistencia()

Tabla 52: Tabla Prueba 31 Confirmar asistencia

3.6.4 Resultados

Finalmente mostramos los resultados de las funcionalidades implementadas en esta última iteración.

Ilustración 48: Ilustración Vista de Estadísticas Administrador

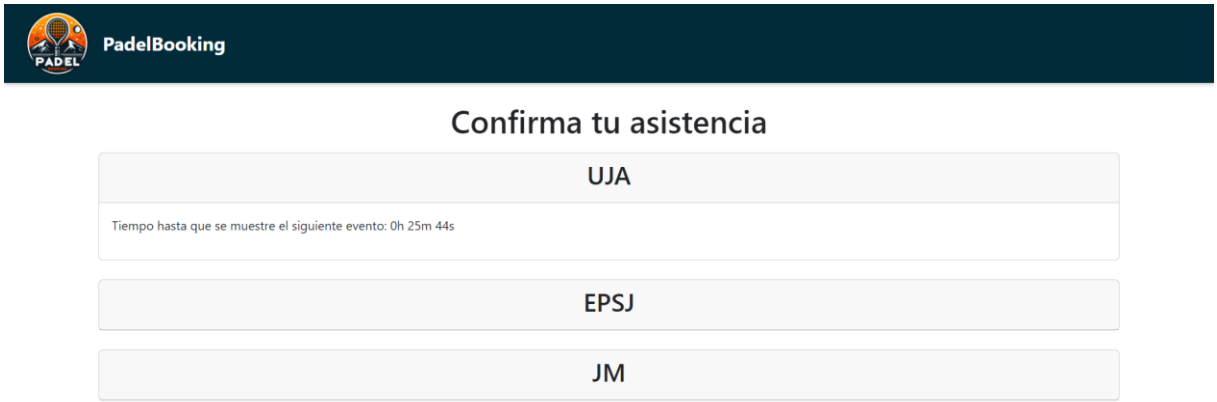


Ilustración 49: Ilustración Vista de Asistencias con eventos próximos

Ilustración 50: Ilustración Vista de Confirmar Asistencia

Ilustración 51: Ilustración Vista de Asistencia Confirmada

3.7 Pruebas finales

Para garantizar que la aplicación cumple con los requisitos y funciona correctamente se realizó un proceso de pruebas exhaustivo utilizando Docker.

Se utilizó Docker para crear un entorno controlado y reproducible para las pruebas. Se construyeron las imágenes utilizando los Dockerfile configurados para la aplicación y sus servicios independientes. Estas imágenes incluían todos los componentes necesarios, como la base de datos y el servidor de la aplicación.

Se definieron y configuraron los servicios en un archivo docker-compose.yml para gestionar múltiples contenedores y sus interacciones. Los contenedores fueron desplegados en un dispositivo diferente del entorno de desarrollo para simular un entorno de producción realista. Esto ayudó a identificar posibles problemas que podrían surgir al ejecutar la aplicación en un entorno distinto al de desarrollo.

Una vez que los contenedores estaban en funcionamiento, se realizaron pruebas para validar las funcionalidades de la aplicación.

A parte de esto, dentro de la aplicación se han programado unos test para el servicio y el controladorREST para validar la funcionalidad de las funciones programadas para la aplicación y que aseguran su correcto funcionamiento.

3.7.1 Validación de la usabilidad del sistema

Para el diseño de la aplicación se ha prestado especial atención a la usabilidad tanto en dispositivos móviles como en ordenadores de escritorio durante todo el desarrollo de las iteraciones aplicando estilos diferentes dependiendo del tamaño de la pantalla.

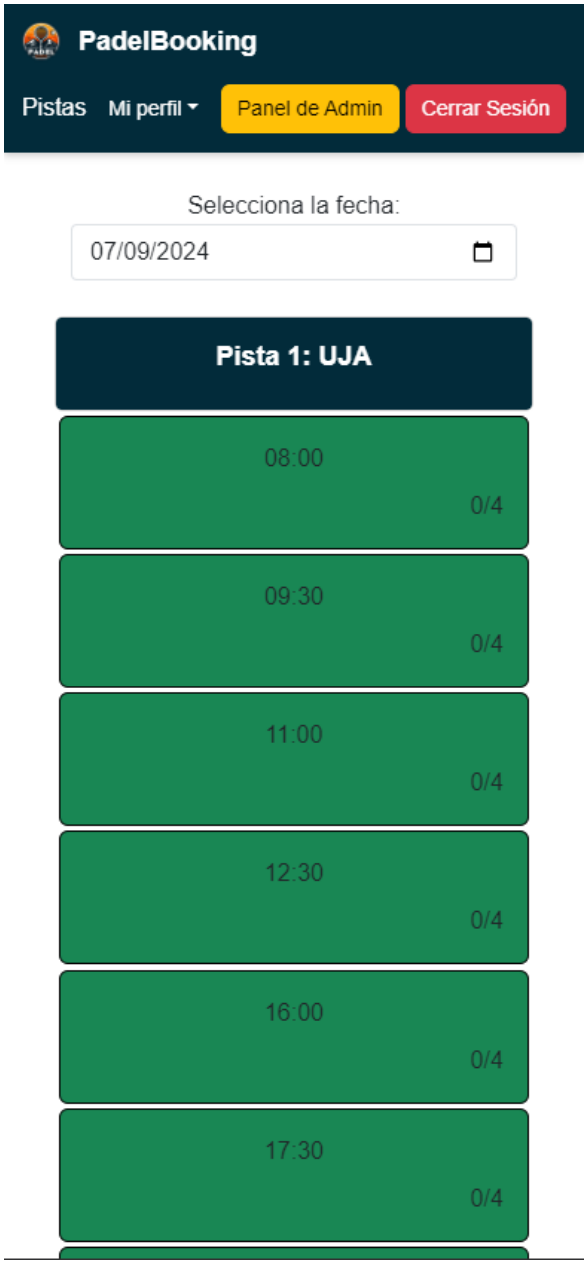


Ilustración 52: Ilustración Vista Móvil Menú Principal

La naturaleza de la aplicación requiere que los usuarios puedan consultar y gestionar sus reservas de manera rápida y eficiente. En situaciones cotidianas, como cuando están en movimiento o en un lugar donde no tienen acceso a un ordenador, los usuarios recurren a sus dispositivos móviles para acceder a la aplicación. Por lo tanto, una interfaz optimizada para móviles garantiza que puedan realizar reservas o revisar el estado de las mismas sin dificultad.

Ilustración 53: Ilustración Vista Móvil Historial

Una interfaz responsiva que se adapta a diferentes tamaños de pantalla es crucial para proporcionar una experiencia de usuario consistente y positiva. Aunque el administrador de la aplicación suele usar un ordenador de escritorio para gestionar y supervisar las reservas, se ha tomado en cuenta la usabilidad en dispositivos móviles también para estas vistas. Esto es debido a que, en situaciones de trabajo remoto, los administradores podrían necesitar acceder a la aplicación desde diferentes dispositivos.

Ilustración 54: Ilustración Vista Móvil Administrador

Para la barra lateral del administrador hemos puesto un botón que al pulsarlo aparece la barra con las opciones del administrado y al seleccionar una opción se cierra directamente para permitir la correcta visualización de las vistas.

Ilustración 55: Ilustración Vista Móvil Administrador Barra Lateral

4 CONCLUSIONES Y TRABAJOS FUTUROS

Tras la conclusión del proyecto hemos conseguido realizar una aplicación web con unas características importantes para el nivel de un TFG y en el que hemos podido exponer de manera personal lo que sería una aplicación fácil e intuitiva para el usuario basado en mi uso en aplicaciones de reservas de este tipo.

La aplicación está pensada para aquellas pequeñas empresas o clubes que tienen una cantidad pequeña de pistas para alquilar, pero esta puede ser escalable de amplia manera.

El desarrollo de este proyecto ha requerido el aprendizaje de nuevas tecnologías en el ámbito del desarrollo web de manera que he adquirido nuevos conocimientos que mejoran mi formación y que puede ser útil en el futuro laboral.

También he aprendido a priorizar del proyecto, teniendo que elegir en cada iteración que funcionalidades eran más convenientes para conseguir un avance en el proyecto que incrementase su valor.

Sobre los trabajos futuros, esta aplicación es fácilmente escalable y pueden añadirse muchísimas funcionalidades nuevas que incrementarían su valor y funcionalidades. Voy a comentar las que he pensado que serían más interesantes.

1. Múltiples deportes: Esta aplicación está pensada para reservas de pistas de pádel, pero se podrían añadir distintos deportes con sus correspondientes pistas y sus horarios como podría ser fútbol, baloncesto, tenis, etc. Esto sería ideal para unas instalaciones deportivas que tienen una variedad de pistas y deportes.
2. Gestión de torneos: La aplicación está pensada para la reserva de pistas del día a día, pero sería interesante añadirle una funcionalidad que permita organizar torneos formados por equipos y que según el perfil de usuario haga sorteos equitativos y permita ver los resultados de los distintos partidos
3. Tienda online: Este tipo de clubes suelen tener una tienda en sus instalaciones en la que venden material deportivo en relación al deporte que se practica, por tanto, sería interesante añadir un sistema de venta online para que pudieran comprar ese material desde casa.
4. Pasarela de pago: Actualmente la aplicación está pensada para que se haga la reserva y se pague en las instalaciones al encargado que esté presente, de manera física como con tarjeta, pero sería interesante incluir una manera de hacer el pago online en el momento de hacer la reserva.

5 APÉNDICES

5.1 Guía original del Trabajo Fin de Título

En este apartado se muestra la información acerca de la guía original del TFG publicada en la web.

UNIVERSIDAD DE JAÉN
Escuela Politécnica Superior de Jaén

PROPUESTA DE TRABAJO DE FIN DE GRADO

GRADO EN:
Grado en Ingeniería Informática

ESPECIALIDAD:
General

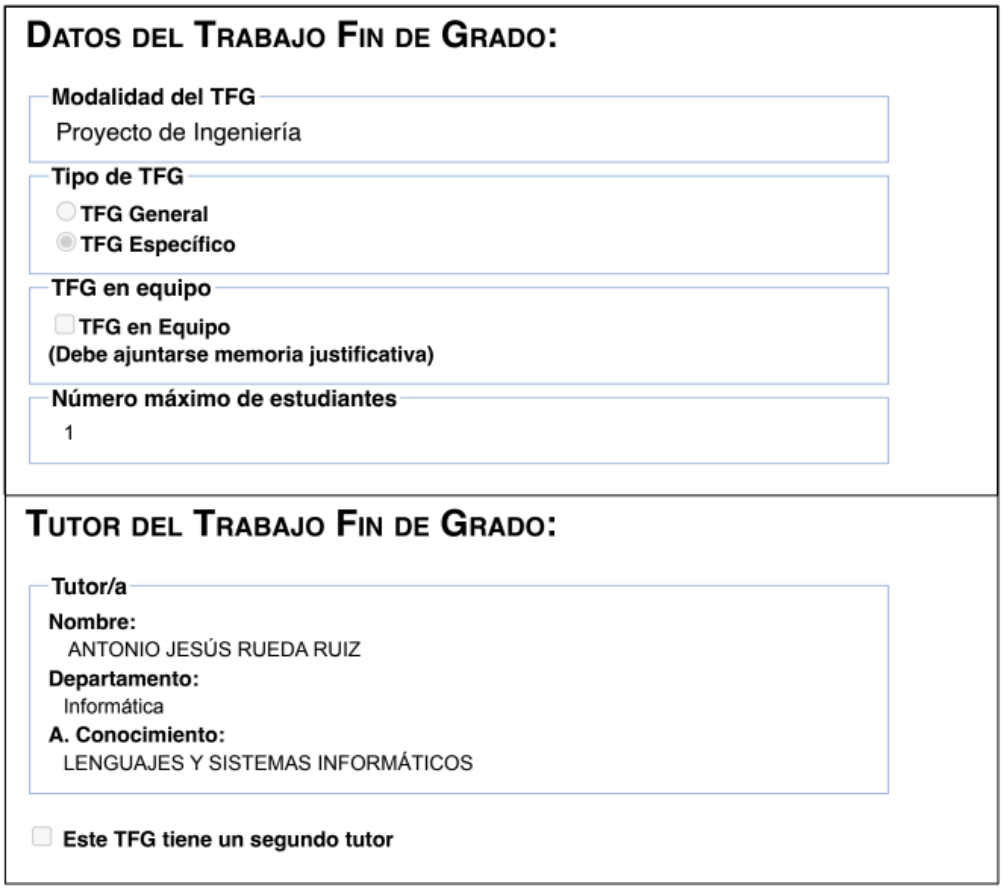
MENCIÓN:
General

TÍTULO DEL TRABAJO:
Sistema de gestión de reservas y participación para pádel y deportes similares

Idioma:
Castellano

DESCRIPCION CORTA DEL TFG:
Este TFG plantea una sencilla aplicación que permita la reserva de instalaciones de pádel y deportes de equipo similares que aporte valor añadido como facilitar la organización de partidos, establecer mecanismos de penalización para usuarios que incumplan la reglas de participación, generar estadísticas de uso de las instalaciones, ranking de usuarios, etc, ofertas y premios para jugadores frecuentes, etc.

A continuación, los datos del TFG y del tutor.



DATOS DEL TRABAJO FIN DE GRADO:

Modalidad del TFG
Proyecto de Ingeniería

Tipo de TFG
- ○ TFG General
- ◉ TFG Específico

TFG en equipo
- ☐ TFG en Equipo

(Debe ajuntarse memoria justificativa)

Número máximo de estudiantes
1

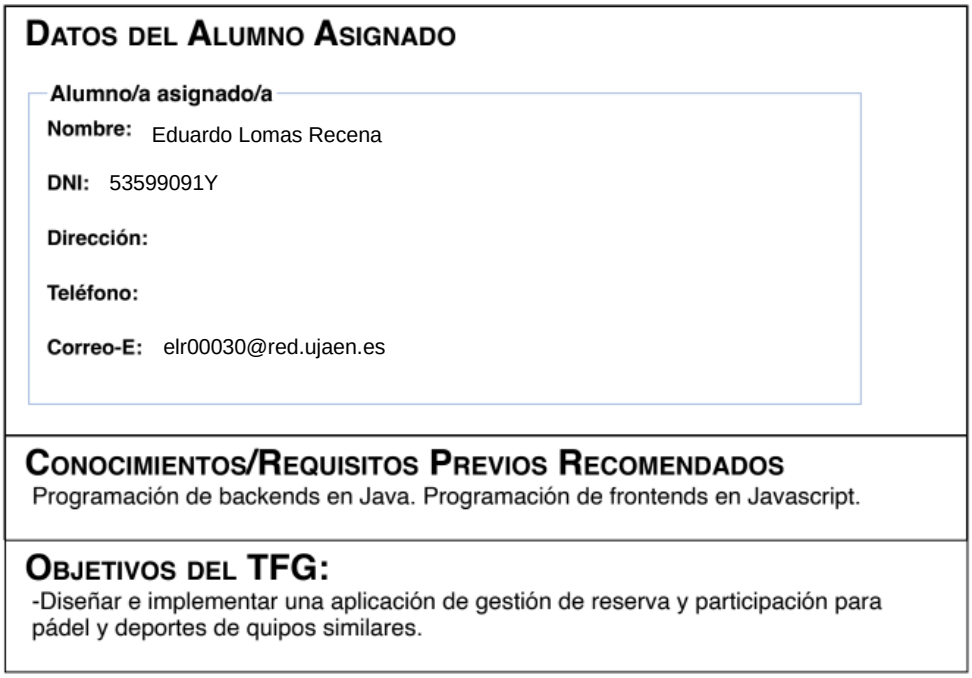
TUTOR DEL TRABAJO FIN DE GRADO:

Tutor/a

Nombre:
ANTONIO JESÚS RUEDA RUIZ
Departamento:
Informática
A. Conocimiento:
LENGUAJES Y SISTEMAS INFORMÁTICOS

☐ **Este TFG tiene un segundo tutor**

Posteriormente los datos del alumno.



DATOS DEL ALUMNO ASIGNADO

Alumno/a asignado/a

Nombre: Eduardo Lomas Recena

DNI: 53599091Y

Dirección:

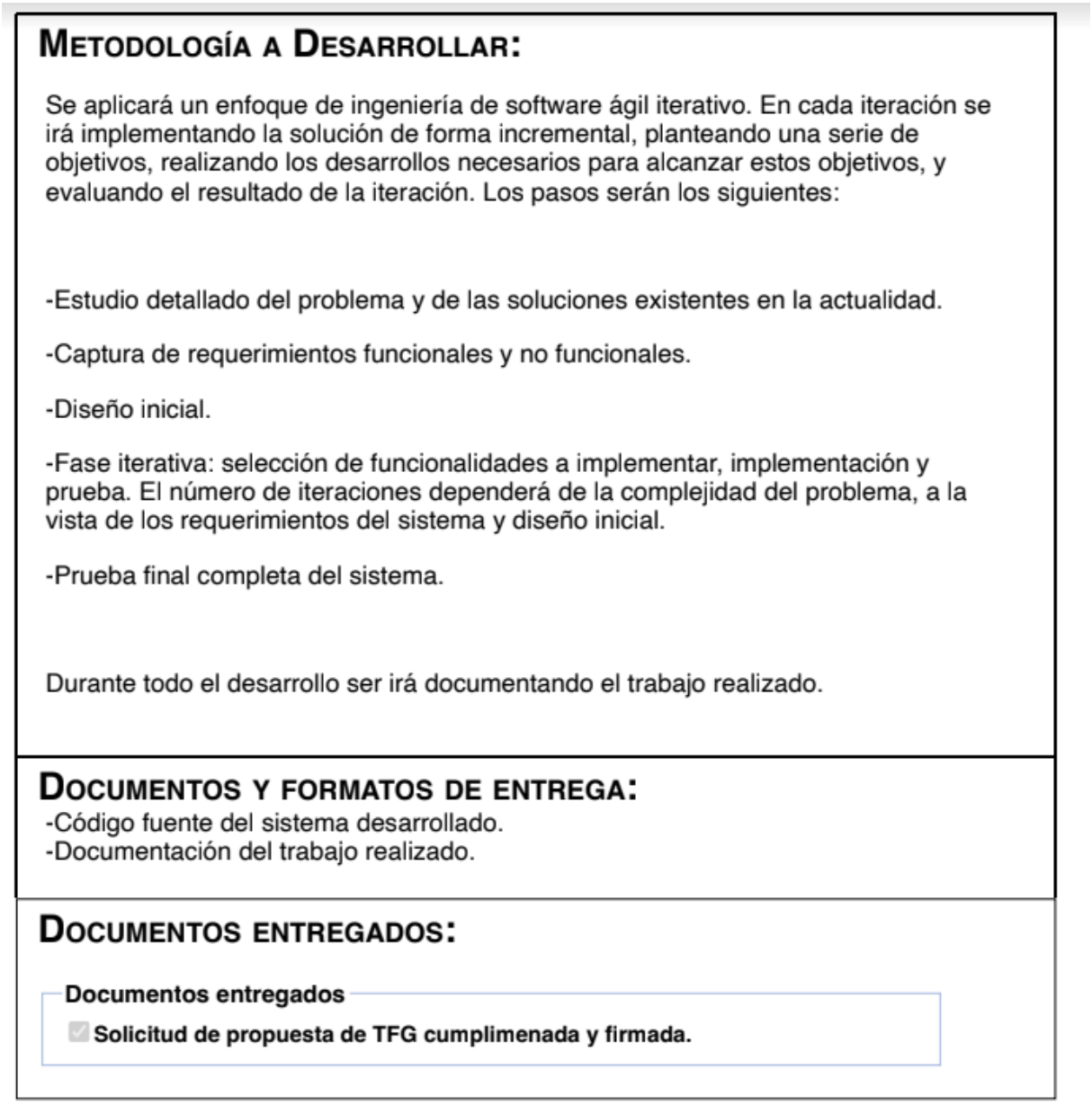
Teléfono:

Correo-E: elr00030@red.ujaen.es

CONOCIMIENTOS/REQUISITOS PREVIOS RECOMENDADOS
Programación de backends en Java. Programación de frontends en Javascript.

OBJETIVOS DEL TFG:
-Diseñar e implementar una aplicación de gestión de reserva y participación para pádel y deportes de quipos similares.

Por último, se adjunta la metodología a desarrollar y los documentos y formatos de entrega.



METODOLOGÍA A DESARROLLAR:

Se aplicará un enfoque de ingeniería de software ágil iterativo. En cada iteración se irá implementando la solución de forma incremental, planteando una serie de objetivos, realizando los desarrollos necesarios para alcanzar estos objetivos, y evaluando el resultado de la iteración. Los pasos serán los siguientes:

-Estudio detallado del problema y de las soluciones existentes en la actualidad.

-Captura de requerimientos funcionales y no funcionales.

-Diseño inicial.

-Fase iterativa: selección de funcionalidades a implementar, implementación y prueba. El número de iteraciones dependerá de la complejidad del problema, a la vista de los requerimientos del sistema y diseño inicial.

-Prueba final completa del sistema.

Durante todo el desarrollo ser irá documentando el trabajo realizado.

DOCUMENTOS Y FORMATOS DE ENTREGA:

-Código fuente del sistema desarrollado.
-Documentación del trabajo realizado.

DOCUMENTOS ENTREGADOS:

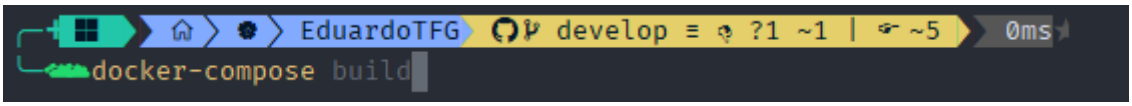
Documentos entregados

☑ Solicitud de propuesta de TFG cumplimenada y firmada.

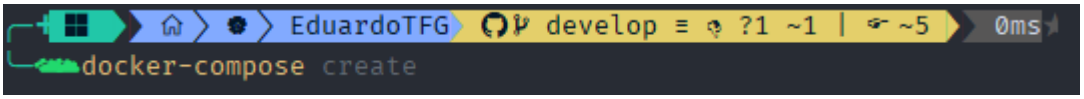
5.2 Instalación y configuración del sistema

Para la instalación se recomienda instalar Docker Desktop (https://www.docker.com/products/docker-desktop/) ya que con ello se instalará Docker y Docker-compose y además dispondrá una interfaz visual para ejecutarlo más fácilmente.

Para instalarlo, nos dirigimos desde la terminal al directorio raíz de la aplicación y ejecutamos el siguiente comando.



Con este comando crearemos las imágenes asociadas a cada servicio especificado en el docker-compose.yml. Si queremos crear los contenedores asociados a las imágenes, posteriormente de crearlas ejecutaremos este comando.



De esta manera se crearán los contenedores. Recomiendo ejecutar primeramente la base de datos y posteriormente el backend y los frontends para asegurarnos el funcionamiento correcto. Hay que tener en cuenta que no se estén en uso ninguno de los puertos configurados para la ejecución de los contenedores para que estos se ejecuten sin problema. En nuestro caso los puertos configurados son 3306 para la base de datos, 8080 para el backend y el 3000 y 3001 para los frontends.

Al iniciarse la aplicación hay unos datos insertados directamente ya en la base de datos para verla en acción. Estos son los usuarios que puedes utilizar directamente con sus credenciales:

- DNI: 55555555M/ Contraseña: sakjhfgjhsaghjf (Este es el único usuario que es administrador)
- DNI: 55555555K / Contraseña: KJSDGFKJDSHKFJHSD
- DNI: 55555555S / Contraseña: jdshgfjshdgjhfdsgsaas
- DNI: 55555555L / Contraseña: jkashdkjhsakjhdas

6 DEFINICIONES Y ABREVIATURAS

- **Frontend:** Es la parte de una aplicación web que interactúa directamente con el usuario. Es la capa visual y la interfaz con la que los usuarios interactúan,

incluyendo todos los elementos visibles en una página web, como botones, formularios, menús, gráficos. EL frontend se desarrolla utilizando tecnologías como HTML, CSS y JavaScript, y se enfoca en la experiencia del usuario y en la presentación de la información de manera clara y atractiva.

- **Backend:** Es la parte de una aplicación web que funciona detrás de escena y gestiona la lógica del negocio, las bases de datos, la autenticación de usuarios, y otras operaciones del servidor. No es visible para los usuarios, pero es fundamental para el funcionamiento de la aplicación. El backend se comunica con el frontend enviando y recibiendo datos a través de la red, generalmente mediante APIs. Se desarrolla usando lenguajes y frameworks como Java con Spring, Python con Django, o Node.js, y maneja tareas como el procesamiento de datos, el almacenamiento en base de datos, y la implementación de reglas de negocios.

- **API:** Interfaz de Programación de Aplicaciones es un conjunto de definiciones y protocolos que permiten que diferentes software o aplicaciones se comuniquen entre sí. En esencia, una API define cómo los diferentes componentes de software deben interactuar, proporcionando métodos y herramientas para que los desarrolladores puedan realizar peticiones y recibir respuestas de otros sistemas o servicios. Las APIs pueden ser utilizadas para acceder a datos, funciones o servicios de una aplicación externa y son esenciales para la integración de sistemas y la creación de aplicaciones que interactúan con otros servicios. Por ejemplo, una API de un servicio de pago permite a una aplicación procesar transacciones financieras mediante la llamada a las funciones ofrecidas por el servicio de pago.

- **Framework:** Es una estructura de soporte y un conjunto de herramientas y bibliotecas diseñadas para facilitar el desarrollo de aplicaciones de software. Proporciona una base de código reutilizable y una serie de convenciones y directrices que los desarrolladores deben seguir, lo que ayuda a estandarizar y agilizar el proceso de desarrollo. Los frameworks suelen incluir funcionalidades integradas para tareas comunes, como la gestión de base de datos, la autenticación de usuarios y la construcción de interfaces de usuario, lo que permite a los desarrolladores centrarse en la lógica específica de su aplicación en lugar de detalles técnicos subyacentes. Ejemplos de frameworks incluyen

React para el desarrollo frontend, Spring para aplicaciones Java en el backend, y Django para el desarrollo en Pyhton.

- **DOM:** Significa Modelo de Objeto de Documento. Es una interfaz de programación para documentos web. El DOM representa la estructura de un documento HTML o XML como una jerarquía de nodos, donde cada nodo corresponde a una parte del documento (como un elemento, un atributo o un texto). En esencia, el DOM permite a los programadores acceder y manipular el contenido, la estructura y el estilo de un documento web desde el código, usualmente mediante lenguajes de programación como JavaScript. Por ejemplo, con el DOM, un script puede cambiar el contenido de una página web, modificar su estructura, o agregar nuevos elementos sin necesidad de recargar la página completa.
- **RESTful:** Estilo de arquitectura para servicios web basado en los principios de REST (Representational State Transfer). Un servicio RESTful utiliza métodos HTTP estándar (como GET, POST, PUT, DELETE) para interactuar con recursos identificados por URLs, permitiendo una comunicación simple, escalable y sin estado entre cliente y servidor.
- **PasswordEncoder:** Interfaz que define métodos para la codificación y verificación de contraseñas. Su propósito principal es proporcionar una manera segura de almacenar contraseñas en una base de datos, evitando el almacenamiento de contraseñas en texto plano.

7 BIBLIOGRAFÍA

- Documentación React: https://react.dev/
- Documentación Spring: https://docs.spring.io/spring-framework/reference/index.html
- Documentación JWT: https://jwt.io/
- Documentación Spring Security: https://spring.io/projects/spring-security
- Documentación Booststrap: https://getbootstrap.com/docs/4.1/getting-started/introduction/
- Documentación JavaScript: https://getbootstrap.com/docs/4.1/getting-started/introduction/
- Documentación Docker: https://docs.docker.com/
- Documentación MySQL: https://dev.mysql.com/doc/
- React vs Angular: https://medium.com/notasdeangular/angular-vs-react-dos-herramientas-poderosas-en-la-web-d5e9abc568ab
- Cookies vs JWT: https://medium.com/@prashantramnyc/difference-between-session-cookies-vs-jwt-json-web-tokens-for-session-management-4be67d2f066e
- Django vs Spring Boot: https://www.geeksforgeeks.org/django-vs-spring-boot/
- JWT con Spring Security: https://www.toptal.com/spring/spring-security-tutorial
- Scrum: https://es.wikipedia.org/wiki/Scrum_(desarrollo_de_software)
- Kanban: https://www.atlassian.com/es/agile/kanban/boards#:~:text=%22Kanban%22%20es%20una%20palabra%20japonesa,el%20mundo%20trabaje%20en%20sinton%C3%ADa.