UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería Informática

Desarrollo de una aplicación híbrida para la reserva y gestión de instalaciones deportivas: desarrollo del back-end.

Trabajo Fin de Grado

Grado en Ingeniería Informática

AUTOR/A: Montalvo Tercero, Tomás

Tutor/a: Pelechano Ferragud, Vicente

CURSO ACADÉMICO: 2021/2022

Agradecimientos

A los profesores que me han acompañado a lo largo de mis estudios, en especial a aquellos que me han transmitido pasión por la informática y por el desarrollo de software, que han hecho más amena mi etapa universitaria y tantas cosas he aprendido de ellos.

A mi familia y amigos que han sido un apoyo en mi etapa universitaria y me han permitido que mi experiencia en valencia sea algo inolvidable. En especial a mi compañero y amigo Javier Bellot el cual ha hecho posible que hoy esté aquí, por tantas tardes de estudio y motivación para seguir adelante, por ser mi fiel amigo estos 4 años. También a mi compañero de clase y de TFG Rubén Gordo por haber despertado en mi la inquietud de desarrollar una idea de negocio, por enseñarme lo que significa el esfuerzo continuo y no rendirse nunca.

Resumen

Hoy en día y más aún con la situación que vivimos de pandemia mucha gente se ha lanzado a las pistas deportivas para poder despejar su mente y poder respirar aire puro, algo que hasta hace un par de años no habíamos echado de menos, pero que ahora tanta importancia le damos.

Appuntate es un trabajo de emprendimiento que pretende facilitar la reserva de instalaciones deportivas, ya sea para practicar deporte con amigos o para competir en eventos organizados y recibir clases y cursillos.

El trabajo constará de dos partes, la primera, realizada junto con un compañero, consistirá en la elaboración de la idea de negocio, especificación de requisitos, puesta en marcha y trabajo futuro, de esta manera se definirá en conjunto todo el proceso de emprendimiento que no esté ligado a una parte técnica, esta parte repesenta aproximadamente el 40% del TFG.

La segunda parte tratará el trabajo específico e individual desarrollado, me centraré sobre todo el proceso y metodología seguida para el desarrollo del back, basado en un modelo API REST que hace uso de JPA e Hibernate para conectar con la base de datos. También se mostrará la arquitectura y estructura del proyecto. Como parte adicional el trabajo contendrá pequeños aspectos a tener en cuenta a la hora de desarrollar un api (endpoints, mensajes de error, pruebas con Postman, etc).

Para el desarrollo de la capa de persistencia se utilizará como base de datos PostgreSQL administrada con pgAdmin y para el back-end y capa lógica se hará uso de spring frameworks y el lenguaje Java en el entorno VScode.

Palabras clave: emprendimiento, back-end, java, spring, hibernate, jpa, maven, API REST

Abstract

Nowadays, and even more so with the pandemic situation we live in, many people have dived into sports courts to clear their minds and breathe fresh air, something we had never missed until a couple of years ago, but we now give so much importance.

Appuntate is an entrepreneurial project that aims to ease the reservation of sports facilities, either to practice sports with friends, as for to compete in organized events and to receive lessons and courses. The project will consist of two parts, the first one, carried out together with a partner, will consist of the elaboration of the business idea, requirement specification, start-up and future work to be done, thus defining together the whole process of entrepreneurship which isn't linked to any technical part, this section represents aproximately 40% of this FDP.

The second part will deal with the specific and individual work developed, I will focus on the whole process and methodology followed for the development of the back end, based on a REST API model that makes use of JPA and Hibernate to connect to the database. I will also show the architecture and structure of the project. As an additional part, the work will contain small aspects to take into account when developing an API (endpoints, error messages, tests with Postman, etc).

For the development of the persistence layer we will use PostgreSQL database, managed with pgAdmin, and for the back-end and logic layer we will use spring frameworks and the Java programming language in the VScode environment.

Keywords: entrepreneurship, back-end, java, spring, hibernate, jpa, maven, API REST

Índice

Índice de ilustraciones

Índice de tablas

1. Introducción

1.1 Motivación

En un mundo donde la mayor parte de las empresas ya habían pasado por un proceso de informatización, la pandemia mundial vivida recientemente ha contribuido a que las empresas que todavía no lo habían hecho se vieran obligadas a adaptarse para no perder fuerza y seguir siendo relevantes en su sector. Además de esto, el COVID-19 también ha abierto las puertas a nuevas formas de trabajo y al surgimiento de nuevas empresas que han revolucionado sus sectores o han experimentado un gran crecimiento.

Dentro de este contexto la sociedad también ha experimentado un cambio en sus hábitos. Este cambio ha significado un mayor acercamiento al mundo de la tecnología sustituyendo muchas de las tareas cotidianas del día a día que realizamos de manera presencial a un modelo remoto o sin salir de casa. Como por ejemplo hacer la compra, pedir comida a domicilio en ocasiones en las que hubiésemos ido al restaurante, reuniones con amigos y familia por videollamada y hasta un modelo de teletrabajo.

Aunque la mayoría de los sectores se han adaptado a la situación que hemos vivido, en el ámbito de las instalaciones deportivas no hemos visto el surgimiento o el crecimiento de un software de gestión de reservas que haya dado respuesta a las nuevas necesidades de la sociedad. Por esta razón creemos que Appuntate puede satisfacer dichas necesidades.

Impulsados por estas ideas pensamos que Appuntate es la propuesta ideal para hacerse un hueco en el mercado modernizando el concepto existente del software de gestión de instalaciones deportivas y creando un entorno de competencia saludable en un mercado dominado por una aplicación que es claramente superior al resto de las alternativas existentes y que analizaremos más en detalle a lo largo de este TFG.

1.2 Objetivos

Este trabajo tiene como objetivo el desarrollo de un proyecto de emprendimiento que facilite la reserva y gestión de instalaciones deportivas. Favoreciendo la práctica de deporte en la sociedad.
Este objetivo puede dividirse en 3 apartados distintos:

En este apartado se cubren los objetivos relacionados con el emprendimiento y el desarrollo de un modelo de negocio.

Emprendimiento

- Establecer nuestra idea de negocio
- Realizar un estudio de mercado
- Generar una proyección económica
- Presentar la propuesta a posibles clientes con experiencia en el sector

En este apartado se cubren las características con las que cuenta Appuntate.

Producto
- Facilitar la reserva de instalaciones y actividades deportivas
- Facilitar la gestión de instalaciones deportivas
- Unificar la reserva y gestión en una misma aplicación
- Ofrecer una interfaz de usuario sencilla e intuitiva
- Dar mayor visibilidad y promover el deporte

Estos objetivos están enfocados al desarrollo de la aplicación.

Tecnología
- Desarrollar una API REST con código limpio y mantenible
- Seguir buenas prácticas de programación en java

1.3 Estructura de la memoria

Este trabajo realizado en conjunto presenta la evaluación y puesta en marcha de una idea de negocio seguida del desarrollo y estudio para una posible comercialización. El proyecto seguirá la siguiente estructura.

- **Capítulo 1:** Establecer la motivación y los objetivos de la propuesta.
- **Capítulo 2:** Evaluar la idea de negocio haciendo un análisis y un estudio exhaustivo de todos los aspectos relacionados con el emprendimiento: análisis de competidores, proyección económica, etc.
- **Capítulo 3:** Generar una ERS (Especificación de Requisitos) sobre la cual se fundamentará todo el desarrollo de la aplicación.
- **Capítulo 4:** Desarrollar experimentos con posibles clientes con experiencia en el sector.
- **Capítulo 5:** Exponer distintos aspectos técnicos como la tecnología y herramientas utilizadas para el desarrollo.
- **Capítulo 6:** Desarrollar la API REST, mostrando casos prácticos y distintos retos que se han surgido en la implementación.
- **Capítulo 7:** Mostrar cómo se ha llevado a cabo el control de versiones y despliegue de la API REST.
- **Capítulo 8:** Exponer las conclusiones del trabajo de emprendimiento.
- **Capítulo 9:** Plantear posibles desarrollos e ideas.

2. Evaluación de la idea de negocio

2.1 Resumen de Appuntate

Appuntate es una aplicación híbrida que pretende unificar en un mismo paquete software la gestión de instalaciones deportivas, así como la reserva de pistas e inscripciones a eventos y cursos. Para lograr este objetivo la aplicación cuenta con dos apartados, uno dedicado a la gestión, utilizado por la administración de los centros deportivos, y otro dedicado al usuario, donde podrá realizar reservas e inscripciones.

El apartado reservado para el usuario ofrece una gran variedad de opciones y filtros dinámicos para encontrar la pista perfecta de forma rápida e intuitiva, consultar sus reservas, organizar su calendario, etc, mientras que el apartado dedicado a la gestión ofrece a los administradores un panel de control desde el cual tienen la capacidad de gestionar sus instalaciones, registrar pistas, publicar eventos, publicar clases y cursos, gestionar las reservas de los usuarios, etc.

La naturaleza híbrida implica que la aplicación sea desarrollada con la intención de ser una aplicación multiplataforma, lo que permite que sea utilizada desde un navegador web, un teléfono móvil o una tablet.

Este paquete software será distribuido mediante licencias de renovación anual que contarán con un pago mensual y variarán en precio según el número de pistas que quiera registrar el centro deportivo.

2.2 Idea de negocio

El objetivo de cualquier idea de negocio es obtener un beneficio monetario, pero antes de comenzar con el desarrollo y puesta en marcha hay que hacer distintos estudios y análisis previos que determinarán si la idea es plausible y tiene un hueco en el mercado.

Los estudios y análisis que hemos realizado para evaluar nuestra idea de negocio recogen tanto el aspecto económico, como fortalezas y debilidades de Appuntate en el mercado.

2.2.1 Estudio de mercado

El primer paso que hemos tomado para la evaluación de la idea de negocio es un estudio de mercado. Este estudio de mercado consiste en un análisis de competidores y una tabla comparativa que pretende exponer las características de Appuntate frente a la competencia.

2.2.1.1 Análisis de competidores

Dentro del ámbito de aplicaciones de gestión de instalaciones deportivas hemos identificado a las que serían nuestras competidoras principales y hemos realizado un análisis individual en el que hemos expuesto las fortalezas, debilidades y público objetivo de cada una de ellas.

Playtomic

El primer competidor que hemos analizado es Playtomic.

Playtomic es una plataforma dedicada a la reserva y gestión de pistas. Esta plataforma ofrece tanto una aplicación móvil como una página web desde las cuales los usuarios pueden realizar distintas operaciones. También ofrece un panel de control desde el cual el centro deportivo puede gestionar las reservas de los usuarios.

Playtomic funciona basándose en el software de gestión, Syltek, un software diseñado para la gestión de centros deportivos distribuido como un "SaaS" o "Software as a service" que permite a Playtomic obtener todas las funcionalidades que se mencionan más adelante.

Al igual que Appuntate, esta plataforma se divide en dos partes, una dirigida al usuario que quiere realizar reservas y otra dedicada a la administración.

Es una aplicación totalmente completa que no solo ofrece al usuario un espacio desde el cual poder realizar reservas, sino que también es una red social donde los usuarios pueden interactuar y compartir publicaciones sobre todo lo relacionado con el deporte.

El apartado dedicado a la gestión tiene nombre propio y se llama "Playtomic manager" y aunque no hemos podido probarlo de primera mano debido a que no está disponible de manera abierta. Sin embargo, su página web ofrece mucha información sobre las funcionalidades que ofrece, además de contar con distintos videos en YouTube donde se explican y se exponen dichas funcionalidades.

A parte de todas estas características con las que cuenta Playtomic, también cabe destacar el éxito que ha tenido durante los últimos años, éxito que solo ha aumentado a raíz de la pandemia y que la ha convertido en la plataforma más grande de España para reservar pistas de tenis y de pádel, llegando a ser patrocinadora oficial del World Padel Tour durante las temporadas 2022 y 2023.

Como podemos ver por el éxito que ha tenido Playtomic en el sector deportivo y la larga y completa lista de funcionalidades que ofrecen tanto la aplicación de gestión "Playtomic Manager" como la aplicación de usuario, consideramos que Playtomic es nuestro mayor competidor y un ejemplo para seguir en cuanto al desarrollo de Appuntate, por esta razón, Playtomic es la aplicación a la que más tiempo de estudio le hemos dedicado.

Sporttia

Esta plataforma de reserva y gestión de instalaciones deportivas ofrece un portal web y una aplicación móvil desde los cuales los centros pueden gestionar sus instalaciones y eventos y los usuarios realizar reservas e inscripciones. Este sistema de gestión y reservas se centra en el sector público, ya que sus clientes son en su totalidad ayuntamientos y centros municipales.

Tras un estudio de la aplicación, haberla probado, y haber preguntado a algunos usuarios habituales y a un administrador de un centro con dicha aplicación, mencionados en los experimentos que expondremos más adelante, llegamos a la conclusión de que es una aplicación con bastantes carencias tanto en diseño de interfaz de usuario como en el aspecto funcional.

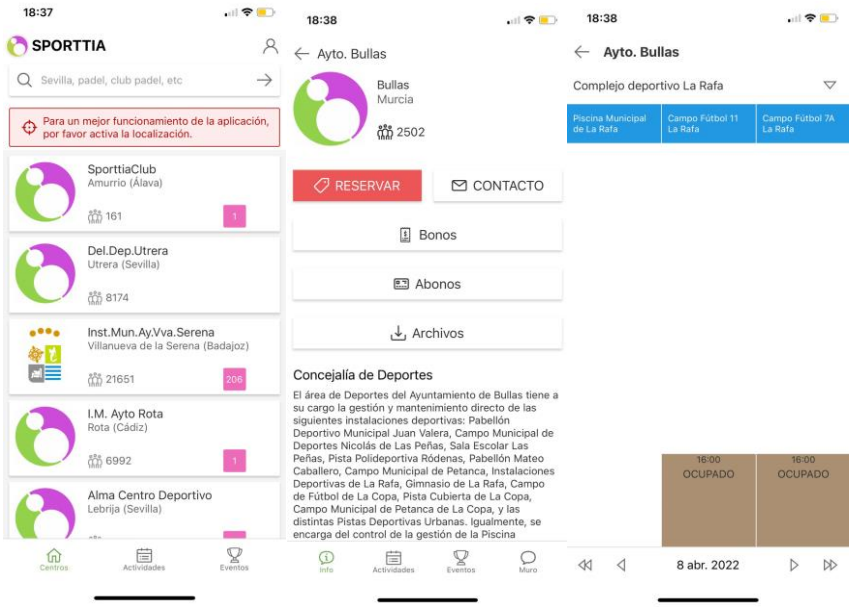

Ilustración 1 - Aplicación Sporttia

Como se puede observar en la imagen 1 el diseño de la interfaz de usuario es poco intuitivo, anticuado. Tiene carencias en el diseño en cuanto a las imágenes de los ayuntamientos, existen una gran cantidad de ayuntamientos que no cuentan con foto, además la visualización del horario para realizar una reserva, dando la sensación de que la tabla no está correctamente formada, ya que cuando la pista está libre se muestran cuadrados en verde, cuando está ocupada cuadrados en marrón sin embargo en ocasiones también aparecen cuadrados en gris y en blanco en los cuales no queda claro cuál es el estado de la pista. Tampoco se adapta correctamente a todos los tamaños de pantalla, no es responsive[1]. También tiene bastantes fallos funcionales pero el más destacable es que cuando pretendes pagar con tarjeta es muy probable que el pago falle y la pista quede bloqueada.

Centrándonos en el apartado de la gestión de centros que proporciona Sporttia. Basándonos en la experiencia que había tenido una de las personas que participó en uno de los experimentos y que era usuaria de este apartado de gestión, podemos concluir que el apartado de gestión no cuenta con toda la funcionalidad necesaria para cumplir con los requisitos del centro que administraba y que además era difícil de manejar.

Pese a la mala puntuación de la aplicación en play store y apple store, y las malas experiencias desde la parte de la administración, esta plataforma junto con la de Omesa (que comentaremos a continuación) dominan el sector público.

MyTurn

Este sistema de gestión y reservas de instalaciones cuenta con un apartado web para la administración y una aplicación móvil para los usuarios. Se trata de una aplicación muy completa y visualmente moderna que pretende facilitar la gestión a los centros deportivos.

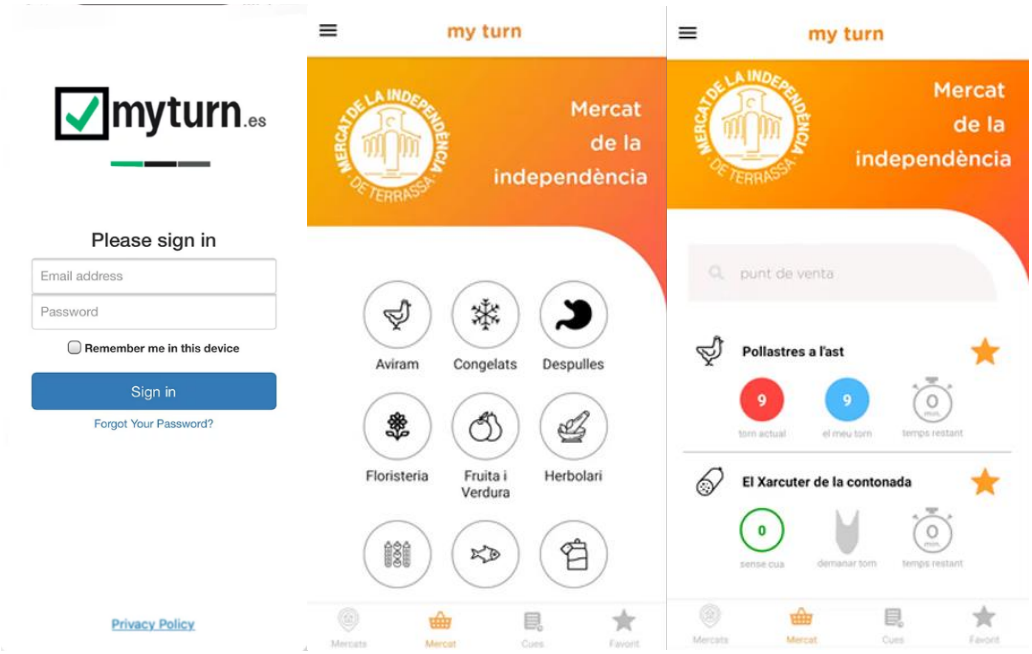

Ilustración 2 - Aplicación MyTurn

Está dirigida principalmente al sector privado, centros pequeños que quieran invertir en digitalización, permitiendo la gestión de gimnasios, negocios físicos, hoteles, campings e instalaciones deportivas que pertenezcan a una comunidad de vecinos, urbanizaciones, etc.

Clicac

Una plataforma web y móvil dirigida a instalaciones deportivas, ayuntamientos, centros privados y comunidades de propietarios, que permite el fácil acceso a las instalaciones mediante un sistema de verificación por NFC.

A pesar de sus altos precios de alta y de renovación de licencia, este software lleva varios años sin lanzar ninguna actualización provocando que en las versiones más recientes de los sistemas operativos iOS y Android la aplicación no funcione además mantiene una interfaz poco intuitiva y anticuada.

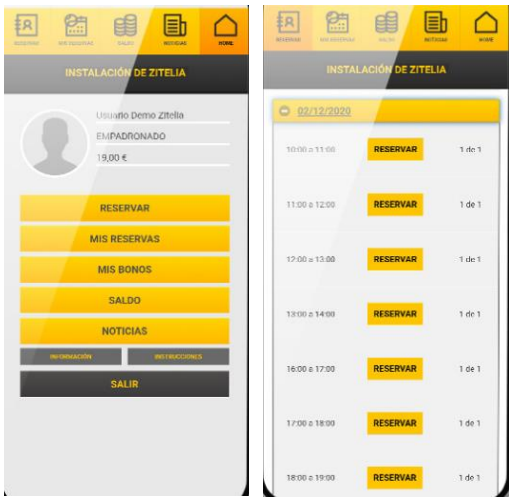

Ilustración 3 - Aplicación Clicac

Omesa

Omesa informática es una empresa fundada en 1988 especializada en el desarrollo de software de gestión de instalaciones deportivas, control de acceso, control de presencia y automoción.

Dentro del catálogo de productos con el que cuenta Omesa, nosotros nos centraremos en, Omesa BeSport, su software dedicado a la gestión deportiva y de gimnasios.

Este software no ofrece una solución integral si no que ofrece distintos módulos que se pueden contratar por separado para ajustarse a las necesidades de cada instalación

A diferencia de otros competidores como Playtomic, Omesa BeSport no se solapa completamente con Appuntate ya que ofrece una serie de servicios que quedan fuera de los objetivos de Appuntate como los terminales de venta automática, el control de accesos, el control de presencia, taquilla, etc.

Omesa ofrece toda la funcionalidad que ofrece Appuntate, sin embargo, su integración modular lo convierte un sistema algo complejo como nos han comentado en los experimentos que se expondrán más adelante, esta complejidad se agrava aún más teniendo en cuenta que la interfaz de usuario, aunque funcional, ha quedado anticuada y está muy lejos de ser sencilla y fácil de usar.

Aunque Omesa cuenta con toda la funcionalidad necesaria en un software de Gestión de instalaciones deportivas, consideramos que no es un fuerte competidor, ya que aunque ofrezca más servicios que Appuntate, como por ejemplo servicios que requieren de una infraestructura física como pueden ser los terminales de auto venta, el ecosistema de Omesa BeSport se centra sobre todo en la parte de gestión y no tanto en la parte de usuario, mientras que Appuntate pretende ser un fuerte competidor en ambas partes, ofreciendo un sistema de gestión completo, sencillo y fácil de usar, además de una aplicación de usuario también completa, sencilla y fácil de usar. Todo en un mismo paquete, eliminando la complejidad de seleccionar los módulos necesarios para cada centro, simplificando enormemente el proceso de implementación de un nuevo sistema de gestión.

2.2.1.2 Tabla comparativa

	Playtomic	Sporttia	MyTurn	Clicac	Omesa informática	Appuntate
Filtros dinámicos	✓	×	✓	×	✓	✓
Gestión de eventos	✓	✓	×	×	✓	✓
Gestión de clases	×	×	✓	×	×	✓
Pago con tarjeta	✓	✓	×	×	✓	✓
Pagar en efectivo	×	×	×	×	×	✓
Reservas grupales	✓	×	×	×	×	✓
Búsqueda por geolocalización	✓	✓	×	×	×	✓
Redirección a google maps para obtener las rutas hacia las instalaciones	✓	×	×	×	×	✓
Integración con google maps con la ubicación de los centros mostrados en chinchetas	✓	×	×	×	×	✓
Gestión de perfil de usuario	✓	✓	×	×	✓	✓
Vista de calendario de reservas/inscripciones (Usuario)	×	×	×	×	×	✓
Reservas rápidas	×	×	×	×	×	✓
Foto de perfil	✓	✓	×	×	×	✓
Galería de fotos de las instalaciones	✓	×	×	×	✓	✓
Realizar reservas en varios deportes	✓	✓	×	×	✓	✓
Filtrar por valoración	×	×	×	×	×	✓
Sistema de reseñas	×	×	×	×	×	✓
Aplicación web (Administracion)	✓	✓	✓	✓	✓	✓
Aplicación móvil (Usuarios&Administración)	✓	✓	✓	×	✓	✓
Modificación de reservas	×	×	×	×	×	✓

Cancelación de reservas*	×	×	×	×	×	✓

Tabla 1 - Comparativa de caracteristicas frente aplicaciones competidoras

*El centro puede cancelar las reservas realizadas por los usuarios y efectuar un reembolso en caso de que lo crean pertinente. Esto es útil cuando por razones como la lluvia un usuario no puede hacer uso de las instalaciones que ha reservado, aunque puede hacerse en cualquier otra situación si el centro lo decide así.

2.2.2 Modelo de negocio

Appuntate es un proyecto de emprendimiento y como para cualquier otra empresa, es necesario elaborar un modelo de negocio.

Este modelo de negocio nos permite definir qué tipo de empresa va a ser Appuntate, definiendo la manera en que vamos a introducir el producto en el mercado además de establecer cómo y desde donde vamos a conseguir nuestros ingresos.

Habiendo realizado un estudio de competidores, conociendo la naturaleza de Appuntate y habiendo concluido que es la única alternativa viable, hemos decidido optar por un modelo SaaS (Software as a Service). Este modelo nos permite distribuir nuestro producto desde la nube (web y tiendas de aplicaciones) y proporcionárselo a nuestros clientes mediante el uso de licencias.

Software as a Service o SaaS, es un modelo de distribución de software basado en la nube que consiste en que el proveedor del software aloje en la nube su producto y los clientes puedan acceder a él desde cualquier lugar. Funciona con un sistema de licencias que permite al cliente pagar por el uso que le dé al producto que se ofrece. Además de hacer que el software sea accesible desde cualquier lugar, el modelo SaaS también trae consigo otras ventajas como la reducción de costes. Esto se consigue debido a que el coste de alojamiento e infraestructura recae sobre el proveedor del software en vez de en el cliente, como ocurre con enfoques más tradicionales en los cuales se suele pagar una vez por el producto y el cliente es el encargado de invertir en la infraestructura necesaria para que el software funcione, además de mantenerla y tener que pagar una cuota de mantenimiento o pagar por nuevas versiones del producto que se lancen en el futuro.

Ilustración 4 - Modelo SaS

Como hemos mencionado anteriormente, una de las razones por las cuales hemos decidido elaborar un modelo SaaS es porque dada la naturaleza de Appuntate, no tiene sentido distribuir el software de manera que se aloje en un servidor del cliente. Vender el sistema de manera que el cliente tuviese que alojarlo el mismo, incrementaría muchísimo el coste de mantenimiento de este software, ya que el cliente no solo tendría que asumir el coste de alojamiento, sino que también, Appuntate como empresa, tendría que cambiar la manera en la que se distribuye el software y la manera en la que genera ingresos, ya que tendría que vender el sistema de gestión como un paquete con un pago inicial y una cuota de mantenimiento, o en su defecto, cobrar por las actualizaciones del sistema. Todo esto empeoraría notoriamente la calidad del producto y lo haría mucho menos atractivo para los clientes ya que además de aumentar los costes, sería más difícil distribuir nuevas versiones del sistema para implementar mejoras o solucionar posibles problemas.

Un enfoque "tradicional" tendría más sentido si por ejemplo, Appuntate, fuese un software hecho a medida para un cliente en concreto y no pudiese ser utilizado por otros clientes. En este caso, sí que sería coherente que el sistema no estuviese alojado en la nube ya que solo accedería a él ese cliente en concreto y además, no existiría la posibilidad de distribuirlo a otros clientes porque al ser un software a medida, no se adaptaría a las necesidades de otros centros.

De esta manera, ofrecemos Appuntate mediante una serie de licencias. Estas licencias variaran en precio dependiendo de las necesidades del cliente siguiendo el modelo SaaS, aplicando un enfoque distinto dependiendo del tipo de centro, instalaciones privadas y ayuntamientos. Decidimos hacer esta distinción debido a que las necesidades de estos dos tipos de organizaciones varían de forma significativa y nos presentan distintos tipos de oportunidades. El precio de las licencias será determinado, en el caso de los centros privados, por la cantidad de pistas que quieran gestionar y en el caso de los ayuntamientos, dependerá de la cantidad de centros en los que se vaya a utilizar el sistema.

2.2.3 Proyección económica

A continuación, vamos a exponer la proyección económica de esta idea de negocio en vista a los próximos 5 años. Para calcular los gastos y beneficios que se van a mostrar a continuación hemos intentado ser fieles a la realidad, utilizando precios reales de los servicios que utilizamos y determinando el precio de nuestro producto según las conclusiones sacadas en las distintas reuniones con ayuntamientos y observando el precio de los competidores.

Como hemos expuesto anteriormente, distribuiremos el producto mediante una renovación anual de la licencia y con pagos mensuales. A continuación, se muestran los precios de las distintas licencias disponibles.

	Pistas	***Precio***
Basic	7	39,99€
Standard	14	74,99€
Premium	25	119,99€
Premium+	25+	119,99€ + 9,99€ / pista extra
Ayuntamiento	Ilimitadas	89

Tabla 2 - Precio mensual de Appuntate

Para el desarrollo de esta proyección económica hemos determinado que nuestros ingresos se dividirán entre las distintas licencias de la siguiente manera.

	%	*Ingresos / 100 licencias*
Basic	10	3,999€
Standard	10	7,499€
Premium	10	11,999€
Premium+	5 (3 pistas extra)	7,4935€
Ayuntamiento	65	57,85€
		Total = 88,84€

Tabla 3 - Distribución de licencias

Apoyándonos en los cálculos mostrados en la tabla 3, el precio medio por licencia siguiendo esta distribución es de 88,84€/mes.

Número de Trimestre	1	2	3	4	5	6	7	8	9	10
Tipos de licencias acumuladas	**Año 1 / T1**	**Año 1 / T2**	**Año 1 / T3**	**Año 1 / T4**	**Año 2 / T1**	**Año 2 / T2**	**Año 2 / T3**	**Año 2 / T4**	**Año 3 / T1**	**Año 3 / T2**
Licencias nuevas vendidas	2	2	3	3	4	4	4	4	6	6
Licencias renovadas después de un año	0	0	0	0	2	2	3	3	4	4
Total de licencias nuevas vendidas y renovadas	2	2	3	3	6	6	7	7	10	10
Total de licencias Activas	2	4	7	10	14	18	22	26	32	38
Ingresos Trimestrales										
(Ventas nuevas + Renovaciones) * Precio	533 €	533 €	800 €	800 €	1.599 €	1.599 €	1.866 €	1.866 €	2.665 €	2.665 €
Total Ingresos	533 €	1.066 €	1.866 €	2.665 €	3.731 €	4.797 €	5.863 €	6.930 €	8.529 €	10.128 €

Número de Trimestre	11	12	13	14	15	16	17	18	19	20
Tipos de licencias acumuladas	**Año 3 /T3**	**Año 3 / T4**	**Año 4 / T1**	**Año 4 / T2**	**Año 4 / T3**	**Año 4 / T4**	**Año 5 / T1**	**Año 5 / T2**	**Año 5 / T3**	**Año 5 / T4**
Licencias nuevas vendidas	6	6	10	10	10	10	10	10	10	10
Licencias renovadas después de un año	4	4	6	6	6	6	10	10	10	10
Total de licencias nuevas vendidas y renovadas	10	10	16	16	16	16	20	20	20	20
Total de licencias Activas	44	50	60	70	80	90	100	110	120	130
Ingresos Trimestrales										
(Ventas nuevas + Renovaciones) * Precio	2.665 €	2.665 €	4.264 €	4.264 €	4.264 €	4.264 €	5.330 €	5.330 €	5.330 €	5.330
Total Ingresos	11.727 €	13.326 €	15.991 €	18.656 €	21.322 €	23.987 €	26.652 €	29.317 €	31.982 €	34.648

Ilustración 5 - Beneficios trimestrales al cabo de 5 años

En la imagen 5 se muestran los beneficios obtenidos en los próximos 5 años desglosados trimestralmente. Para ello se ha tenido en cuenta el número de licencias nuevas vendidas y renovadas de forma trimestral.

Para el calcular los gastos se han utilizado los precios reales de los servicios utilizados, así como cuotas de autónomos y otros, con el fin de ser lo más fieles posibles a la realidad.

Número de Trimestre	1	2	3	4	5	6	7	8	9	10
Tipos de licencias acumuladas	**Año 1 / T1**	**Año 1 / T2**	**Año 1 / T3**	**Año 1 / T4**	**Año 2 / T1**	**Año 2 / T2**	**Año 2 / T3**	**Año 2 / T4**	**Año 3 / T1**	**Año 3 / T2**
Gastos Anuales										
Alojamiento cloud de la base de datos https://aw	0 €	100 €	150 €	150 €	150 €	150 €	150 €	150 €	150 €	150 €
Google developer account	6,25 €	6,25 €	6,25 €	6,25 €	6,25 €	6,25 €	6,25 €	6,25 €	6,25 €	6,25 €
Apple developer account	0 €	24,75 €	24,75 €	24,75 €	24,75 €	24,75 €	24,75 €	24,75 €	24,75 €	24,75 €
Ordenadores	1.129 €	0 €	0 €	0 €	0 €	0 €	0 €	0 €	0 €	0 €
Transporte	75 €	75 €	75 €	75 €	75 €	75 €	75 €	75 €	75 €	75 €
Telefono	23 €	23 €	23 €	23 €	23 €	23 €	23 €	23 €	23 €	23 €
Sociedad Limitada	3.000 €	0 €	0 €	0 €	0 €	0 €	0 €	0 €	0 €	0 €
Autonomo Societario	1.134 €	1.134 €	1.134 €	1.134 €	1.134 €	1.134 €	1.134 €	1.134 €	1.134 €	1.134 €
Community Manager	0 €	0 €	0 €	0 €	0 €	0 €	0 €	0 €	450 €	450 €
Google API	0 €	0 €	282	660 €	1.164 €	1.668 €	2.172 €	2.676 €	3.180 €	3.684 €
Total Gastos	5.366 €	1.362 €	1.694 €	2.072 €	2.576 €	3.080 €	3.584 €	4.088 €	5.042 €	5.546 €

Número de Trimestre	11	12	13	14	15	16	17	18	19	20
Tipos de licencias acumuladas	Año 3 /T3	Año 3 / T4	Año 4 / T1	Año 4 / T2	Año 4 / T3	Año 4 / T4	Año 5 / T1	Año 5 / T2	Año 5 / T3	Año 5 / T4
Gastos Anuales										
Alojamiento cloud de la base de datos https	150 €	150 €	150 €	150 €	150 €	150 €	150 €	150 €	150 €	150 €
Google developer account	6,25 €	6,25 €	6,25 €	6,25 €	6,25 €	6,25 €	6,25 €	6,25 €	6,25 €	6,25 €
Apple developer account	24,75 €	24,75 €	24,75 €	24,75 €	24,75 €	24,75 €	24,75 €	24,75 €	24,75 €	24,75 €
Ordenadores	0 €		1.129 €	0 €	0 €	0 €	0 €	0 €	0 €	0 €
Transporte	75 €	75 €	75 €	75 €	75 €	75 €	75 €	75 €	75 €	75 €
Telefono	23 €	23 €	23 €	23 €	23 €	23 €	23 €	23 €	23 €	23 €
Sociedad Limitada	0 €	0 €	0 €	0 €	0 €	0 €	0 €	0 €	0 €	0 €
Autonomo Societario	1.134 €	1.134 €	1.134 €	1.134 €	1.134 €	1.134 €	1.134 €	1.134 €	1.134 €	1.134 €
Community Manager	450 €	450 €	450 €	450 €	450 €	450 €	450 €	450 €	450 €	450 €
Google API	4.188 €	4.692 €	5.196 €	5.700 €	6.204 €	6.708 €	7.212 €	7.716 €	8.220 €	8.724 €
Total Gastos	6.050 €	6.554 €	8.187 €	7.562 €	8.066 €	8.570 €	9.074 €	9.578 €	10.082 €	10.586 €

Ilustración 6 - Gastos trimestrales al cabo de 5 años

Como podemos observar en la imagen 6 vamos a necesitar servicios de despliegue en Apple y Google y alojamiento cloud de la base de datos, así como, las cuotas de autónomos y de sociedad limitada.

A medida que pase el tiempo y aumenten los ingresos, serán necesarios algunos materiales como teléfonos y ordenadores e incurriremos en gastos de transporte para visitar a los clientes.

Número de Trimestre	1	2	3	4	5	6	7	8	9	10
Tipos de licencias acumuladas	Año 1 / T1	Año 1 / T2	Año 1 / T3	Año 1 / T4	Año 2 / T1	Año 2 / T2	Año 2 / T3	Año 2 / T4	Año 3 / T1	Año 3 / T2
Resultado Trimestral	-4.833 €	-296 €	172 €	593 €	1.155 €	1.717 €	2.279 €	2.841 €	3.487 €	4.582 €
Resultado Trimestral Acumulado	-4.833 €	-5.129 €	-4.958 €	-4.365 €	-3.210 €	-1.492 €	787 €	3.628 €	7.115 €	11.697 €

Número de Trimestre	11	12	13	14	15	16	17	18	19	20
Tipos de licencias acumuladas	Año 3 /T3	Año 3 / T4	Año 4 / T1	Año 4 / T2	Año 4 / T3	Año 4 / T4	Año 5 / T1	Año 5 / T2	Año 5 / T3	Año 5 / T4
Resultado Trimestral	5.677 €	6.772 €	7.804 €	11.094 €	13.255 €	15.417 €	17.578 €	19.739 €	21.900 €	24.061 €
Resultado Trimestral Acumulado	17.373 €	24.145 €	31.949 €	43.044 €	56.299 €	71.716 €	89.294 €	109.033 €	130.933 €	154.995 €

Ilustración 7 - Beneficios netos trimestrales al cabo de 5 años

De esta forma, teniendo en cuenta los ingresos y los gastos generados trimestralmente obtenemos estos resultados, en los que a partir del tercer trimestre del segundo año empezamos a obtener beneficios.

Uno de los principales pilares sobre los que se sustenta la obtención de beneficios en una etapa tan temprana es que seremos nosotros los únicos socios y trabajadores. Planteamos la primera incorporación en el tercer año con un Community Manager para servir de apoyo en el ámbito del marketing y la publicidad.

En caso de que un inversor realizase una entrada con 7.366€ en el caso de que quisiese liquidar su inversión en el tercer año recibiría un retorno sobre su inversión de 434%.

Inversión mínima solicitada	7.366 €
Punto de equilibrio alcanzado en trimestre	5
Recuperación de inversión en trimestre	9
Valor de empresa en año 3 (EBITDA x 10)	212.506 €
Porcentaje de participación asociado a inversión	30%
Multiplicación de inversión con exit en el año 3	1,7605

Ilustración 8 - Beneficio de inversores

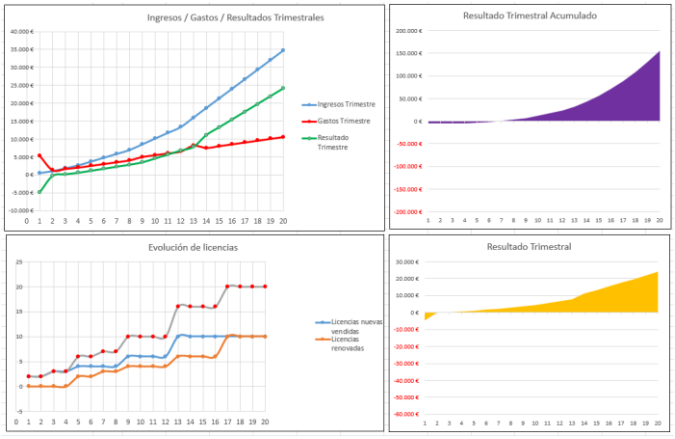

Ilustración 9 - Graficas de gastos y beneficios

2.2.4 Análisis DAFO

A la hora de analizar cuáles son los puntos fuertes y débiles de nuestra idea de negocio y poder enfocar mejor el proyecto hemos realizado un análisis DAFO en el que se muestran las fortalezas y debilidades divididas según si estás son externas o internas.

	INTERNO	EXTERNO
NEGATIVO	**DEBILIDADES** • El equipo de desarrollo cuenta con solo dos integrantes • Poca experiencia en el sector • No contamos con experiencia en otros campos más allá del desarrollo del software (marketing, gestión de instalaciones deportivas…)	**AMENAZAS** • El desarrollo de aplicaciones que van a ser usadas por los ayuntamientos suele hacerse por grandes empresas • Ya existen otras aplicaciones que intentan suplir las mismas necesidades que Appuntate • Los ayuntamientos suelen querer software exclusivo y a medida con lo cual puede ser complicado vender el software a varios ayuntamientos
POSITIVO	**FORTALEZAS** • Uso de ultimas tecnologías (Permite desarrollar aplicaciones multiplataforma) • Proyecto de bajo coste • Proyecto viable para desarrollarlo entre dos personas • Contamos con funcionalidad que no ofrece la competencia	**OPORTUNIDADES** • Nos centramos en un nicho de mercado muy concreto • Poca competencia en el sector público • Gran escalabilidad • Sector de la salud y el deporte está en auge (sobre todo después de la pandemia)

Ilustración 10 - Análisis DAFO

Como se puede observar en el análisis DAFO el mayor problema de esta idea de negocio y su puesta en marcha, es la poca experiencia y el desconocimiento de sectores importantes, como el marketing.

A su vez contamos con puntos muy importantes a nuestro favor y es que, a pesar de existir competencia, nos centramos en un sector concreto del mercado, que está en auge y utilizamos tecnologías moderas además de precios más asequibles que la competencia.

2.2.5 Lean canvas

El Lean Canvas es una adaptación del Business Model Canvas tradicional que forma parte de la metodología Lean Startup.

El lean Canvas es un documento que pretende recoger el plan de negocio en una sola página, exponiendo todos los aspectos clave de la idea de negocio que determinarán el éxito o fracaso de la misma.

La creación de un Lean Canvas es sumamente importante en un proyecto de emprendimiento y en consecuencia, también es muy importante para el desarrollo de Appuntate, ya que nos permite poner en perspectiva el producto que desarrollamos y nos aseguramos de que no vamos a lanzar al mercado un producto que no recibirá una buena aceptación, ya sea por no disponer de una ventaja especial sobre el resto de competidores o porque simplemente no existe un público objetivo interesado en este tipo de producto.

Lean Canvas

Problema

-Interfaces poco intuitivas
-Filtro por geolocalización
-Las instalaciones no cuentan con una galeria de fotos

Solución

-Interfaz sencilla e intuitiva
-Busqueda basada en la ubicación del usuario, mostrando resultados en un radio de 15Km
-Galeria de fotos como requerimiento para el registro de un centro

Proposición de valor única

Unificación de la plataforma de gestión de reservas, eventos, cursos con la aplicación final de usuario para inscribirse y reservar. Solo es necesario Appuntate para la gestión completa de un centro, supliendo todas sus necesidades a traves de una interfaz sencilla e intuitiva

Ventaja especial

Facilidad para acceder a concursos a traves de la universidad

Segmento de clientes

-Ayuntamientos municipales
-Instalaciones deportivas privadas

Alternativas existentes

-Sporttia
-Madrid Móvil
-PlayTomic
-cicac
-myturn

Métricas clave

-Número de reservas de pistas a traves de la aplicación
-Número de usuarios de la aplicación
-Centros intersados
-Licencias renovadas

Concepto de alto nivel

Aplicación híbrida para la reserva y gestión de instalaciones deportivas

Canales

-Reuniones con ayuntamientos municipales
-Reuniones con centros deportivos privados
-Redes sociales

Early Adopters

Clientes:
-Ayuntamientos municipales sin sistema informático para la gestión deportiva
-Centros deportivos privados sin un sistema informático para su gestión
-Centros deportivos privados recien abiertos

Usuarios:
-Usuarios habituales de instalaciones deportivas
-Personal de gestion de las instalaciones deportivas

Estructura de costes

-Google developer account: 6,25€
-Ordenadores: 1,129€
-Transporte: 75€
-Creación de la sociedad límitada: 3000€
-Cuota de autónomo societario: 1134€

Total 5,366€

Flujo de ingresos

-Licencias renovables de pago mensual
-Primer resultado trimestral positivo: Año1/t2
-Primer resultado trimestral acumulado positivo: Año2/t1
-Crecimiento exponencial del resultado trimestral acumulado

Ilustración 11 - Lean Canvas

2.2.6 Conclusiones de la evaluación

Como hemos podido observar en este análisis de la idea de negocio, en el sector privado hay un gran rival al que por presupuesto y madurez en el mercado no podemos hacer frente, Playtomic es una aplicación moderna, activa y que ha tenido una muy buena recepción dentro del mercado consolidando todavía más su éxito en estos tiempos postpandemia. Sin embargo, en el ámbito público encontramos aplicaciones poco actualizadas, con fallos funcionales, visuales y caras, lo que produce una mala experiencia tanto en los usuarios como en los administradores de las instalaciones. Por este motivo creemos que la mejor opción es centrarnos en el sector público ya que la competencia es más floja y podemos hacernos un hueco en el mercado poco a poco.

3. Especificación de requisitos

3.1 Introducción

Appuntate es una herramienta diseñada para la gestión y reserva de instalaciones deportivas.

Este sistema consta de dos partes, una parte, accesible por la administración del centro deportivo, que está disponible tanto en aplicación móvil como en aplicación web, y otra parte, accesible al usuario de las instalaciones.

Tanto los administradores del centro deportivo como los usuarios que hacen uso de las instalaciones accederán a la aplicación iniciando sesión. Para poder iniciar sesión, los usuarios deben registrarse previamente, indicando nombre de usuario, correo electrónico, número de teléfono, nombre, apellidos y contraseña. Los administradores podrán acceder a la aplicación con la cuenta que se les proporcionará una vez se inscriba el centro deportivo en la aplicación.

Para la inscripción de un centro, se solicitará que proporcione una dirección, una o varias imágenes de las instalaciones, los deportes que se pueden practicar y la información de contacto que consta de una dirección de correo electrónico y un teléfono. El equipo de Appuntate se encargará de dar de alta al centro y le proporcionará una cuenta desde la cual los administradores puedan acceder a la aplicación.

Una vez inscrito el centro, el personal de administración podrá introducir las distintas pistas que tienen disponibles, pudiendo especificar el deporte, el horario, la duración de las reservas y el precio de reserva. Además, desde la aplicación el administrador del centro será capaz de consultar todas las reservas que se han realizado a través de la aplicación para cada pista y también podrá introducir, cancelar o modificar reservas. De esta manera podrá introducir en la aplicación las reservas que se realicen por teléfono, en el centro o a través de cualquier otro medio. Esto permite que Appuntate sea la única herramienta que necesite un centro para la gestión de reservas. Desde la aplicación también podrá acceder al histórico de reservas de cada pista.

Además de la gestión de reservas, los centros también pueden publicar eventos, torneos con generación automática de cuadros, ligas y clases que aparecen en el tablón principal de la aplicación desde el cual los usuarios pueden consultar toda la información sobre el evento e inscribirse en él.

Al crear un evento, el administrador, puede seleccionar los campos necesarios para adaptar el formulario de inscripción según los datos que necesite el centro para ese evento determinado.

Para el uso de la aplicación, los usuarios deberán registrarse proporcionando nombre, apellidos, contraseña, nombre de usuario, email y teléfono. También podrán subir una foto de perfil para mejorar la experiencia social dentro de la aplicación.

Una vez iniciada la sesión pueden buscar pistas haciendo uso de filtros. Pueden filtrar por deporte, horario, valoración de las pistas y ubicación. Estos filtros pueden hacer uso de la localización del usuario para mostrar los centros más cercanos, en un radio de 15km.
El resultado de la búsqueda que se muestra tendrá dos vistas distintas, una en forma de lista y otra en la que se mostrarán los centros sobre un mapa.

Una vez encontrada la pista que quiere reservar, el usuario puede seleccionar la hora de la reserva y un método de pago (tarjeta de crédito o efectivo).
Al realizar la reserva, el usuario puede valorar las instalaciones dándoles una valoración basada en estrellas, de 1-5 estrellas, siendo 1 la peor valoración y 5 la mejor valoración además de poder dejar un pequeño comentario.
Al igual que los administradores, los usuarios también pueden acceder a su propio registro de reservas teniendo disponibles dos vistas, una en forma de lista y otra en forma de calendario, pudiendo cancelar o modificar cualquiera de las reservas. También tendrán un historial en el que consultar las reservas realizadas el último mes.

Para facilitar el desplazamiento a las instalaciones reservadas se proporcionará una redirección a Google Maps que indique la ruta desde la ubicación actual hasta el centro.

3.2 Modelo de dominio

En la etapa de especificación de requisitos es necesario desarrollar un modelo de dominio que tiene principal función dar forma y una base al diseño de los objetos software. Se muestras las clases conceptuales al dominio del problema con la finalidad de dar una solución estructurada, indicando relaciones relevantes, vocabulario, etc.

Para realizar este modelo de dominio se ha utilizado UML (Unified Modeling Language), un lenguaje de modelado respaldado por el grupo OMG, que nos permite representar las clases del modelo de dominio.

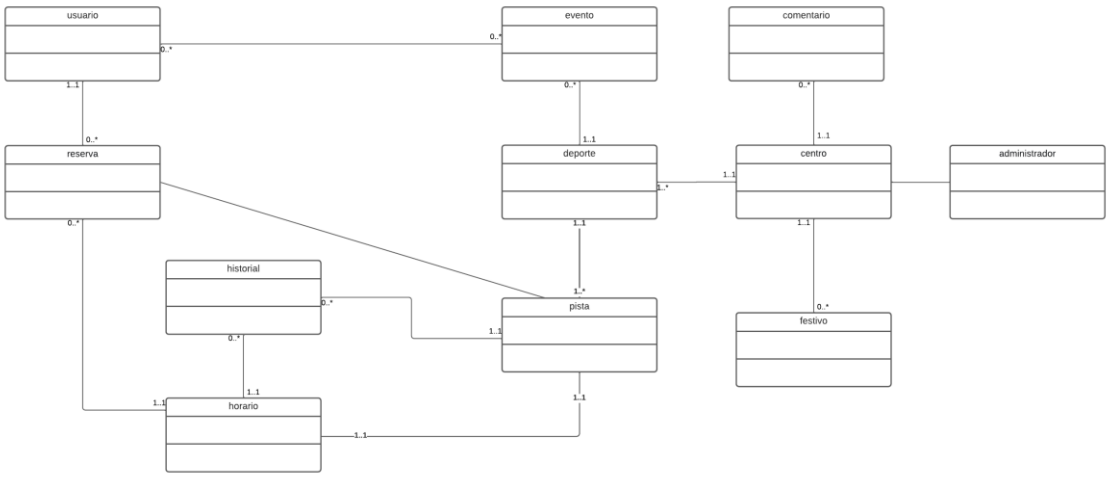

Ilustración 12 - Modelo de dominio

Como se muestra en la imagen 12 en este modelo de dominio se representa, se puede observar como las distintas entidades se relacionan entre sí para dar una solución gráfica al problema software planteado.

Un centro, que tiene un administrador, cuenta con días festivos y comentarios que dejan los usuarios. Un centro ofrece disponibilidad para practicar distintos deportes, por tanto, el centro debe ofrecer pistas y horarios para practicar dicho deporte, estas pistas las pueden reservar los usuarios de Appuntate. El centro también pude organizar eventos a los cuales se pueden inscribir los usuarios.

3.3 Casos de uso

- **Motor de búsqueda:** Para ofrecer una buena solución a la búsqueda de pistas para reservar se ofrece la característica de un motor de búsqueda.
    - **Buscar pistas disponibles:** Se muestra como resultado solo los centros con pistas disponibles.
    - **Filtrar polideportivos por ubicación (mapa + geolocalización):** Se ofrece un filtro por geolocalización en un radio de 15km y la posibilidad de ver los resultados en un mapa
    - **Filtrar polideportivos por deporte:** Se debe poder filtrar por deporte.
    - **Filtrar pistas por valoración:** Para una mejor experiencia del usuario se puede filtrar por valoración de pistas, encontrando las pistas mejor valoradas.
    - **Filtrar polideportivos por fecha + horario:** Se debe poder filtrar por fecha y hora en la que se desea jugar.
    - **Visualizar información de polideportivo (usuario) (incluidas reseñas):** De los resultados encontrados se debe mostrar información relevante como reseñas.

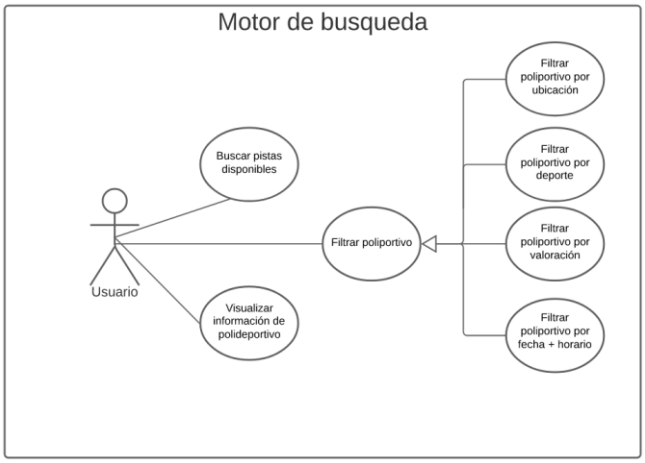

Ilustración 13 - Casos de uso para la característica: Motor de búsqueda

- **Gestión de centros:** Se necesita una lógica de gestión de centros en los que los administradores puedan personalizar su polideportivo.
    - **Añadir foto de polideportivo:** Los administradores pueden añadir fotos a la información del polideportivo.
    - **Modificar fotos de polideportivo:** También deben poder modificar las fotos ya añadidas.
    - **Iniciar sesión:** Los administradores deben iniciar sesión para realizar las acciones comentadas.
    - **Cerrar sesión:** Deben poder cerrar sesión.

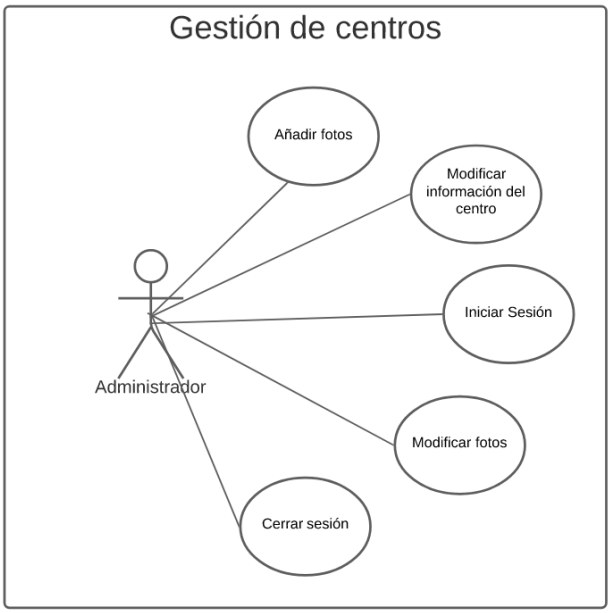

Ilustración 14 - Casos de uso para la característica: Gestión de centros

- **Gestión de instalaciones:** Se debe ofrecer a los centros la opción de modificar la información de sus instalaciones.
    - **Modificar horarios de la pista:** El centro debe poder modificar los horarios de una pista.
    - **Modificar nombre de la pista:** También debe tener la opción de modificar los nombres de las pistas.
    - **Dar de baja una pista:** Debe poder eliminar una pista de la aplicación.
    - **Dar de alta una pista:** Los administradores tienen la opción de añadir nuevas pistas a sus instalaciones.

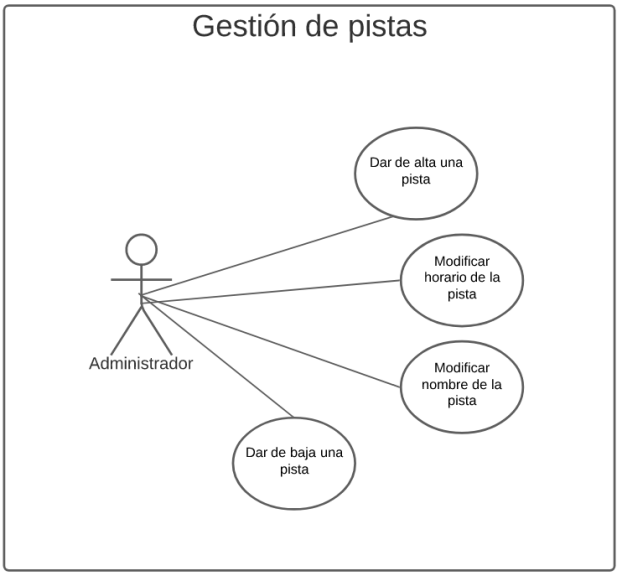

Ilustración 15 - Casos de uso para la característica: Gestión de pistas

- **Gestion de reservas:** Esta caracteristica ofrece toda la funcionalidad necesaria para gestionar las reservas tanto de los usuarios de las instalaciones como de los administradores de las mismas
    - **Reservar una pista:** Reservar una pista para un dia determinado a una hora determinado
    - **Cancelar reserva:** Cancelar una reserva existente
    - **Modificar fecha de reserva:** Modificar la fecha de una reserva en una pista determinada a otra fecha en la que también haya disponibilidad
    - **Consultar reservas activas de una pista:** El administrador puede ver toda la lista de reservas existentes de una pista determinada que tengan una fecha posterior a la actual
    - **Sincronizar reservas con Google Calendar:** Exportar las reservas mediante la API de Google a Google Calendar
    - **Consultar historial de reservas de una pista:** Visualizar las reservas de una pista determinada que tengan una fecha anterior a la actual.
    - **Consultar historial de reservas:** El usuario de las instalaciones puede visualizar todas las reservas que ha realizado con fecha anterior a la actual
    - **Consultar reservas activas:** El usuario de las instalaciones puede visualizar las reservas que haya realizado con una fecha posterior a la actual

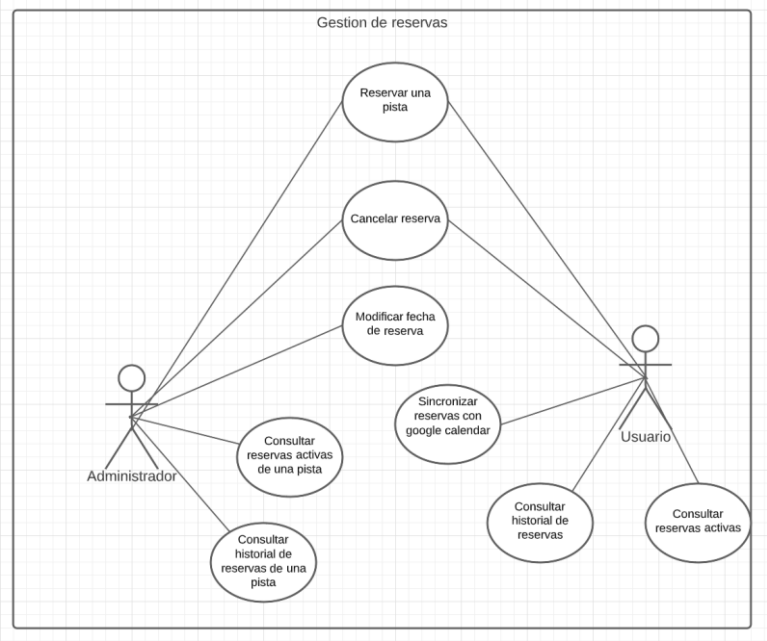

Ilustración 16 - Caso de uso Gestión de reservas

- **Gestión de usuario:** Proporciona la posibilidad de modificar los datos de un usuario
  - **Modificar foto de perfil:** Modificar la foto de perfil ya sea subiendo una foto desde la galería o utilizando la cámara del dispositivo
  - **Registrarse:** Registrarse en la aplicación para poder acceder a ella
  - **Iniciar sesión:** Introducir los credenciales del usuario para acceder a la aplicación
  - **Cerrar sesión:** Cerrar la sesión de usuario actual de la aplicación
  - **Modificar información de usuario:** Modificar cualquier dato que haya introducido el usuario dentro de la aplicación

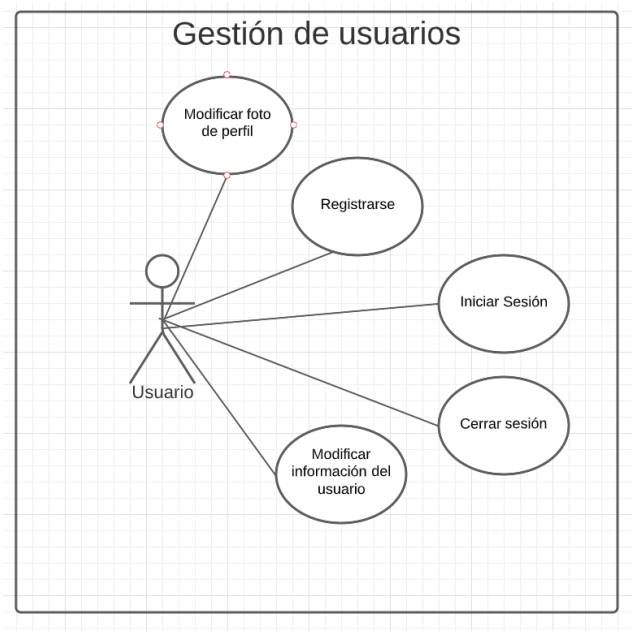

Ilustración 17 - Caso de uso Gestión de usuario

- **Gestión de eventos:** Ofrece toda la funcionalidad necesaria para gestionar la creación e inscripción a eventos
    - **Crear un evento:** El administrador crea un evento añadiendo los datos correspondientes a ese evento
    - **Modificar información de un evento:** El administrador es capaz de modificar cualquier aspecto de un evento que ha creado previamente
    - **Inscribirse en un evento:** El usuario de las instalaciones es capaz de inscribirse a un evento publicado por un centro
    - **Cancelar inscripción:** El usuario cancela la inscripción a un evento al que se ha inscrito previamente

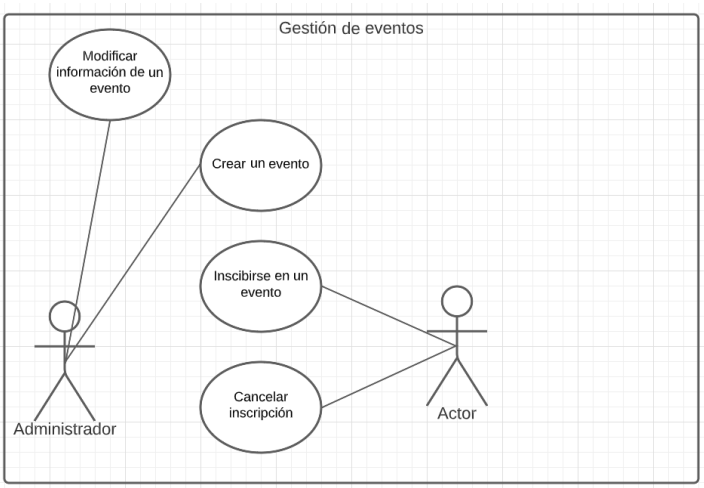

Ilustración 18 - - Caso de uso Gestión de eventos

- **Gestión de reseñas y valoraciones:**
    - **Visualizar reseñas:** Tanto el administrador como el usuario de las instalaciones es capaz de visualizar las reseñas que han dejado el resto de usuarios o ellos mismos
    - **Escribir reseña:** Posteriormente al uso de una instalación, lo que implica la existencia de una reserva, el usuario de las instalaciones es capaz de dejar una reseña sobre la pista que ha utilizado
    - **Responder reseña:** El administrador del centro responde a una reseña que ha dejado un usuario sobre una de sus pistas

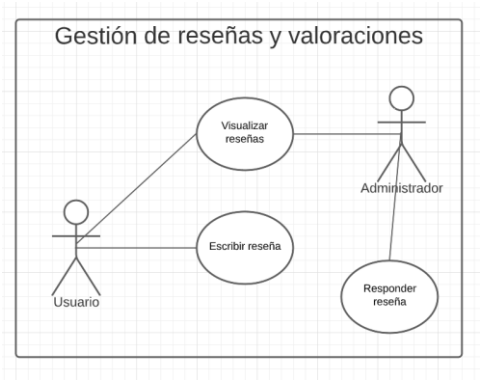

Ilustración 19 - Caso de uso Gestión de reseñas y valoraciones

- **Gestión de pagos:**
  - **Realizar pagos con tarjeta de crédito:** El usuario introduce los datos de su tarjeta y realiza el pago de la reserva que quiere realizar

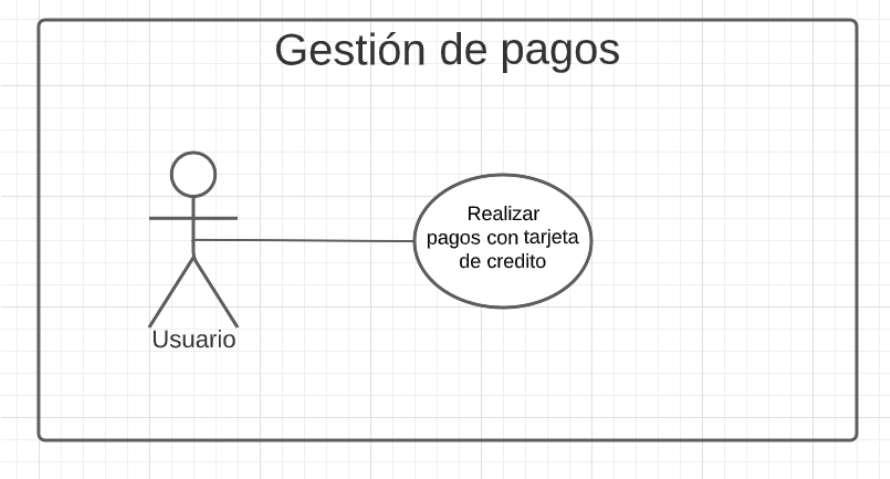

Ilustración 20 - Caso de uso Gestión de pagos

4. Desarrollo

En este apartado se detallan tanto el concepto de MVP como los dos experimentos que se han realizado durante el desarrollo de Appuntate.

En primer lugar, un MVP (Minimum Viable Product), o producto mínimo viable, es una versión de la aplicación que contiene el conjunto de funcionalidades mínimas para poder hacer uso de la aplicación. El desarrollo de estas versiones tiene como objetivo proporcionar un paquete software que pueda ser usado en una fecha previa al final del desarrollo.

En el caso de Appuntate se ha planteado un primer MVP que propone una solución de cara al usuario final de las instalaciones. Este MVP, teniendo en cuenta que Appuntate es un producto dirigido a instalaciones deportivas, no podría considerarse un MVP per se, ya que un centro deportivo no podría implantar la aplicación como sistema de gestión con las funcionalidades ofrecidas en el MVP. Sin embargo, debido al desacoplamiento entre el apartado de la aplicación dirigida a los usuarios de las instalaciones y el apartado dirigido a la administración de las mismas, se ha decidido presentar en una primera instancia, una versión de la aplicación que cubre todas las funcionalidades necesarias para ser utilizada por el usuario de las pistas y una segunda versión en la que se incluyen, esas mismas funcionalidades en adición a algunas, nuevas, y la funcionalidad necesaria, esta vez sí, para que el centro pudiese implementar Appuntate como sistema de gestión de las instalaciones.

Para la realización de estos MVPs nos hemos puesto en contacto con diversos ayuntamientos y centros deportivos para realizar entrevistas acompañadas de una demo y una encuesta.

4.1 Desarrollo del primer MVP

Para el desarrollo de este primer MVP, conseguimos contactar con el ayuntamiento de Murcia y el ayuntamiento de San Pedro.

Durante las entrevistas, se introducía a los representantes del ayuntamiento a la idea de negocio de Appuntate y a su concepto como solución para ofrecer un único producto que fuese capaz de gestionar sus instalaciones deportivas sin la necesidad de otras herramientas.

La demo consistía en un tour por la aplicación, mostrando como un usuario sería capaz de acceder a la aplicación y realizar acciones como reservar una pista, modificar una reserva, buscar pistas, inscribirse en un evento, etc.

En cuanto a la encuesta, se trataba de una encuesta realizada a través de Google Forms en la que se preguntaba por aspectos relacionados a la interfaz de usuario y al concepto de Appuntate como sistema de gestión.

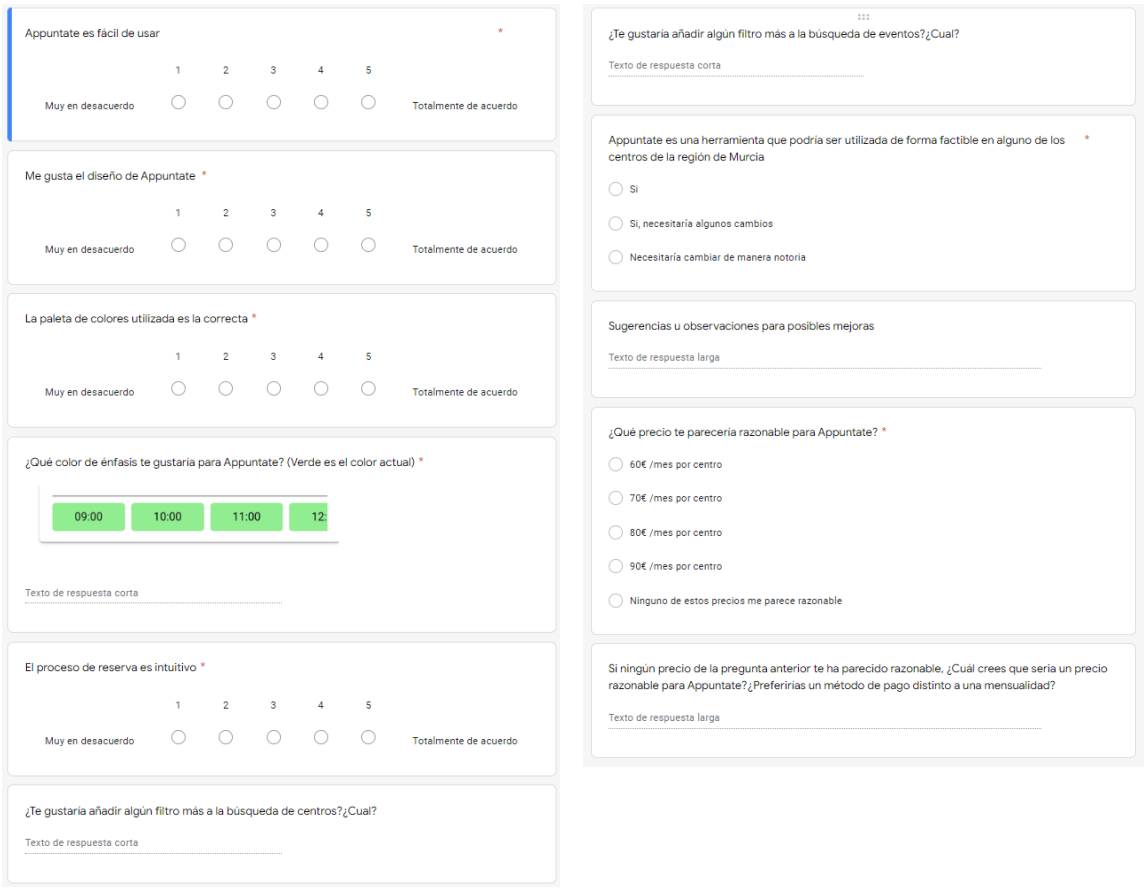

Ilustración 21 - Encuesta del primer MVP

4.1.1 Ayuntamiento de Murcia

Durante esta primera entrevista, los representantes del centro nos comentaron que justo en el momento de realizar la entrevista, estaban a punto de entrar en el proceso de búsqueda de una nueva herramienta de gestión para sus instalaciones deportivas ya que se les acababa el contrato con la que tenían en la actualidad así que estaban especialmente interesados en escuchar la propuesta.

Nos comentaron que Appuntate les pareció una aplicación muy agradable visualmente y que, como concepto, ofrecería una solución prácticamente a medida para sus centros teniendo en cuenta sus necesidades. Sin embargo, también nos comentaron un aspecto clave que ellos necesitarían en un sistema de gestión que Appuntate no contemplaba.

En este caso en particular el ayuntamiento tenía como requisito que cualquier persona, registrada o no en la aplicación, pudiese acceder a ella y consultar las pistas, los horarios, los eventos y la información del centro, pudiendo incluso realizar una reserva sin la necesidad de crear una cuenta.

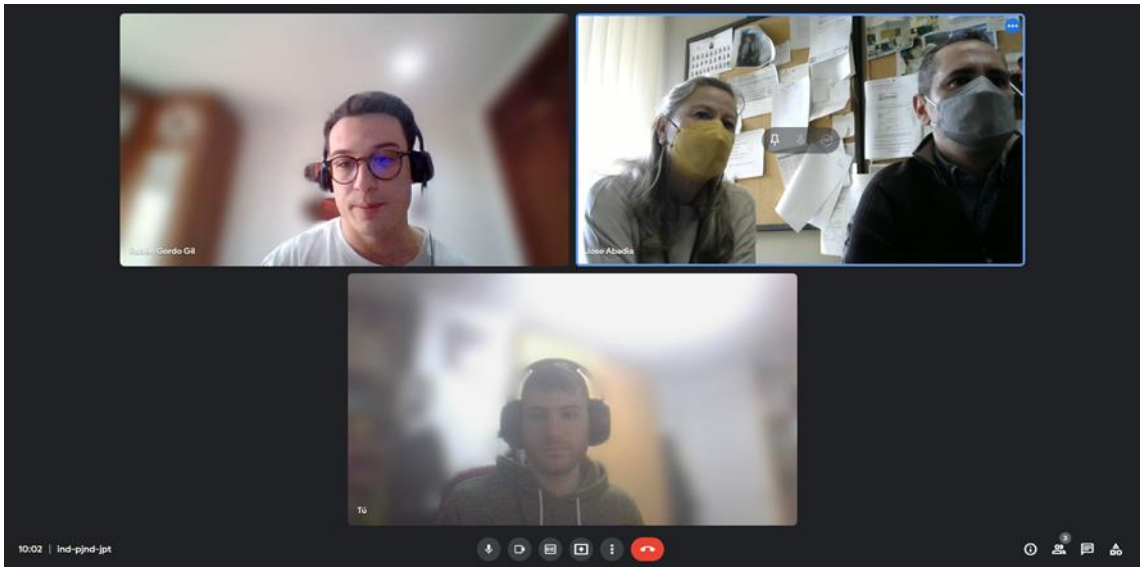

Ilustración 22 - Entrevista con el Ayuntamiento de Murcia

4.1.2 Ayuntamiento de la Pobla de Farnals

La segunda entrevista realizada fue con el responsable de las instalaciones deportivas del ayuntamiento de la Pobla de Farnals, una pequeña localidad de Valencia.

Durante esta entrevista, el feedback recibido fue muy parecido al de la primera entrevista, consideraron Appuntate como una solución muy viable. Al tratarse de una localidad pequeña, todas sus necesidades quedaban prácticamente cubiertas.

Además, el entrevistado nos comentó ciertas funcionalidades adicionales que ayudarían a Appuntate a ganar una ventaja sobre el resto de competidores llegada la hora de sustituir el sistema de gestión que utilizaban actualmente en ese ayuntamiento. Estas funcionalidades

estaban relacionadas con la gestión de eventos y eran la generación automática de cuadros y horarios.

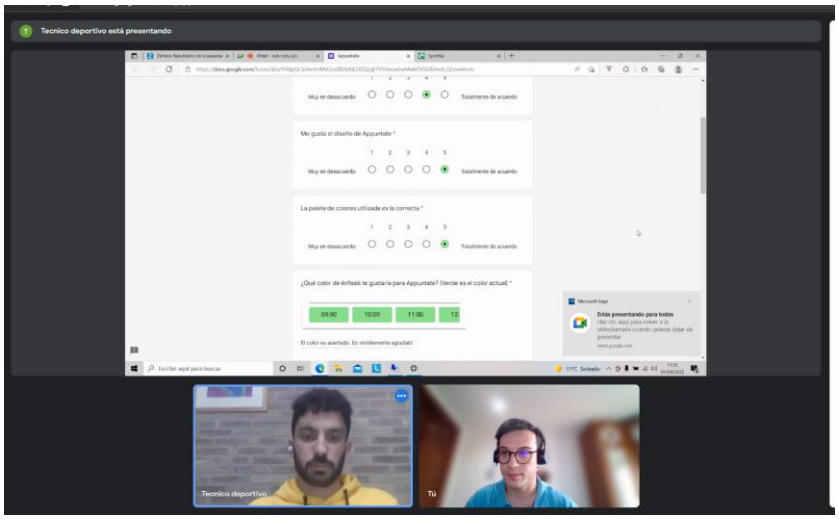

Ilustración 23 - Entrevista con el ayuntamiento de la Pobla de Farnals

4.2 Desarrollo del segundo MVP

Para el desarrollo de este segundo MVP, el planteamiento fue prácticamente el mismo que para el primero. La entrevista consistía en una presentación de la idea de negocio de Appuntate y del concepto de Appuntate como solución única para la gestión de instalaciones deportivas, una demostración, esta vez más extensa y un breve formulario, distinto al utilizado en el primer MVP.

La demostración, en este caso, era más extensa, ya que la aplicación incluía tanto el apartado abierto al usuario de las instalaciones como el apartado de gestión de estas. Siendo el segundo el más interesante para los administradores de los centros.

Esta vez, la encuesta trataba menos sobre el diseño y el concepto de Appuntate y más en funcionalidades concretas extraídas de las entrevistas realizadas durante el primer MVP que podrían resultar interesantes para los centros y que podrían ser implementadas en Appuntate en un futuro.

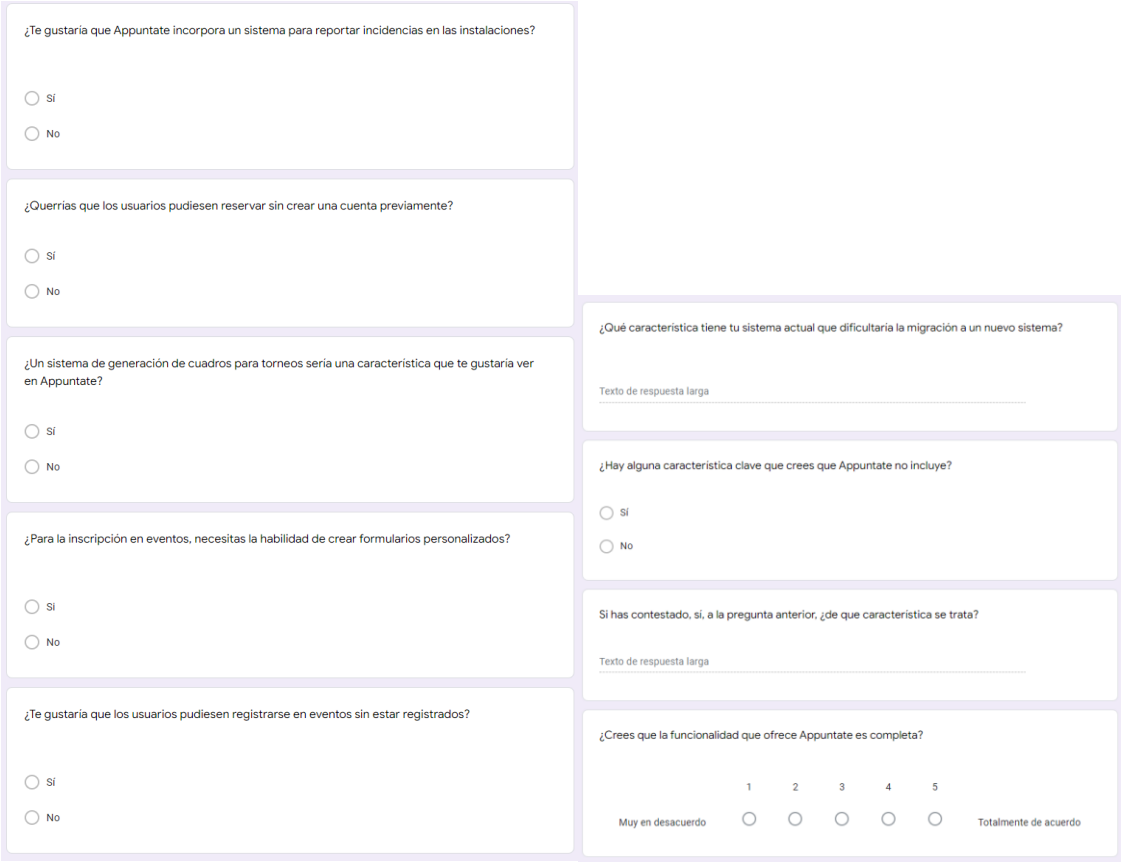

Ilustración 24 - Encuesta primer MVP

4.2.1 Ayuntamiento de San Pedro

Durante esta entrevista el representante dejo claro que en estos momentos no se planteaba un cambio de sistema de gestión de instalaciones deportivas debido a que su centro había adquirido recientemente el sistema de Sporttia, uno de los competidores que hemos analizado. Sin embargo, ya que el objetivo de la entrevista no era vender Appuntate sino presentarlo, nos dio sus opiniones sobre la solución que planteamos.

En primer lugar, un aspecto que resalto por encima del sistema que acaban de adquirir es la sustantiva mejora en el ámbito visual y estético que presentaba Appuntate por encima del sistema de Sporttia, un aspecto que nos comentó que le preocupaba desde el primer momento en el que decidieron adoptar ese sistema.

Además del apartado estético, el representante nos comentó que en cuanto a la funcionalidad que ofrecía el sistema para la gestión de pistas, las necesidades estaban quedando cubiertas y los empleados eran capaces de utilizar el sistema sin ningún tipo de problema, sin embargo, estaba empezando a recibir quejas por parte de los usuarios ya que la interfaz de la aplicación enfocada al público no era intuitiva y estaba provocando que, sobre todo, personas mayores, tuviesen problemas para reservar pistas a través del móvil, lo que resultaba en que acabasen reservándolas mediante los medios tradicionales, ya sea llamando por teléfono o acudiendo al centro. Este aspecto nos comentó que probablemente quedaría solventado si la aplicación tuviese una interfaz parecida a la de Appuntate ya que le pareció intuitiva y fácil de usar.

En cuanto al apartado de gestión de instalaciones de Appuntate, el representante nos comentó que el hecho de que estuviese disponible tanto en versión móvil como en versión web incluyendo exactamente las mismas funcionalidades era una característica clave que le gustaría tener en su sistema actual.

Un aspecto que sí resultó ser un factor decisivo es el hecho de que Appuntate como solución no plantee en primera instancia una infraestructura física, ya que la ventaja y la razón que ofrecía Sporttia por la cual se decantaron en ese centro por ese sistema es el hecho de que aparte de ofrecer una solución software, también ofrecen una infraestructura de sistemas de riego, luces y puertas que facilitaba mucho el trabajo de los empleados del centro.

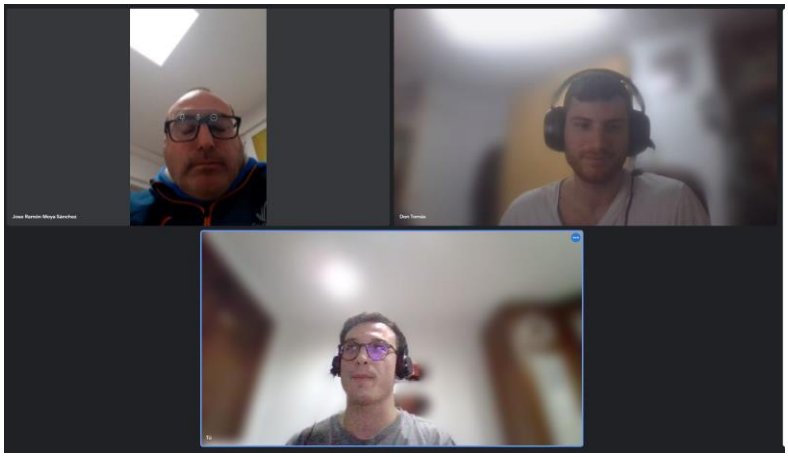

Ilustración 25 - Entrevista con el Ayuntamiento de San Pedro

5. Aspectos técnicos

Este es el primer apartado del TFG que se ha realizado de forma individual y trata sobre las herramientas y entornos de desarrollo utilizados en la API REST.

5.1 Entornos de desarrollo y herramientas

5.1.1 Visual Studio Code[2]

Visual Studio Code es un editor de código desarrollado por Microsoft que permite programar y escribir código en una gran variedad de lenguajes y utilizar distintos frameworks. También cuenta con una gran variedad de extensiones para ayudar a tener un código limpio y ordenado, para ser más productivo y muchas otras utilidades. A pesar de que no es uno de sus puntos fuertes, este entorno de desarrollo también nos permite depurar el código de una forma sencilla con la ayuda de una de sus extensiones (Debugger for Java[3]).

Las razones por las que se ha utilizado este editor de código es que cuenta con una interfaz moderna, minimalista y sencilla de utilizar en la que se reduce el espacio ocupado por las herramientas de trabajo y la consola para dar más protagonismo al código. Permite el uso de Spring Boot (framework que se ha utilizado para el desarrollo), y contiene una gran variedad de extensiones, que facilitan y ayudan a la programación.

Una de las extensiones destacadas que se ha utilizado para aumentar la productividad en el desarrollo de este trabajo es Vim[4], la incorporación de comandos para poder editar código de una forma más rápida y eficiente.

5.1.2 pgAdmin[5]

pgAdmin es un administrador de bases de datos PostgreSQL, a pesar de no ser un muy utilizada en el mundo empresarial es una gran opción para administrar bases de datos PostgreSQL ya que ofrece una interfaz sencilla, en la que se pueden realizar muchas de las acciones sin necesidad de ejecutar scripts SQL y da cierta agilidad a tareas pequeñas como hacer un insert de una sola fila o un select sencillo.

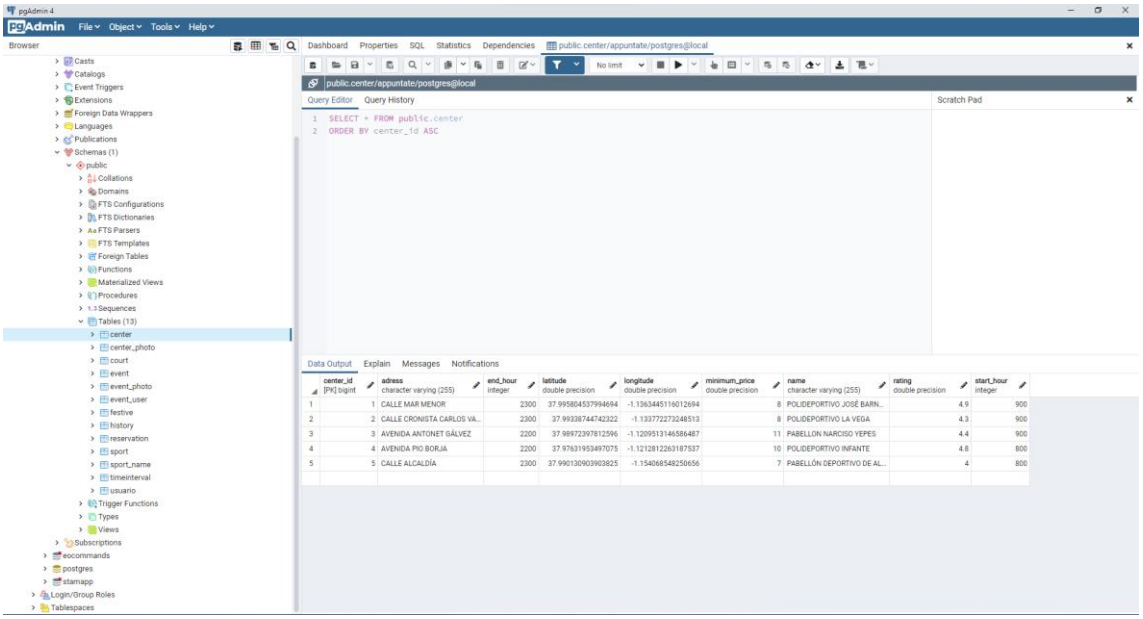

Ilustración 26 - Interfaz pgAdmin

En la imagen 21 se puede observar en la parte izquierda todas las tablas de la base de datos, en la ventana principal un pequeño script select y en la parte inferior el resultado de la búsqueda.

5.1.3 Postman[6]

Postman es una aplicación web y escritorio que nos permite realizar pruebas sobre una API mediante llamadas HTTP. De esta forma se pueden hacer tests de los endpoints ofrecidos por la API sin necesidad de desarrollar un front.

Esta herramienta nos da la facilidad de añadir distintas configuraciones a nuestras llamadas de forma sencilla, pudiendo añadir tipo de autorización, headers y body. También nos ofrece la posibilidad de recibir la respuesta de la API en formato JSON.

A demás cuenta con un sistema de colecciones para poder guardar llamadas a los distintos endpoints y compartirlas, lo que nos ha servido de gran utilidad para que mi compañero (que se encargaba del front) viera cómo se comporta la API antes de empezar a realizar las llamadas desde el front.

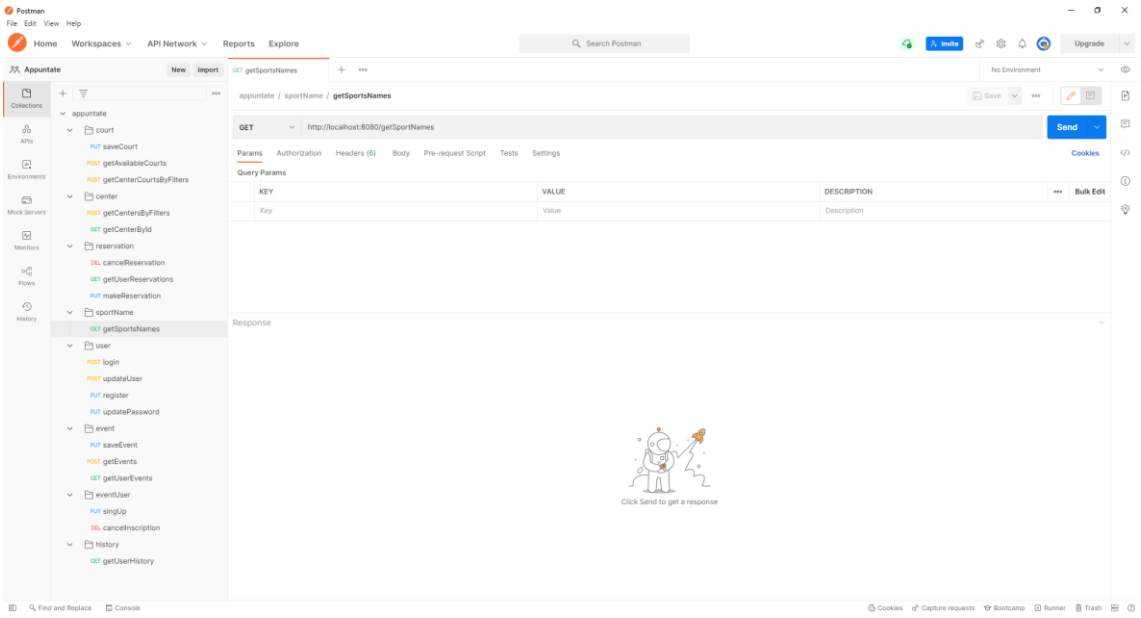

Ilustración 27 - Interfaz Postman

En la figura 22 se puede observar la interfaz de Postman y la colección creada con todas sus carpetas desplegadas para mostrar las distintas llamadas realizadas a los endpoints de la API.

5.2 Tecnologías utilizadas

5.2.1 Spring Boot[7]

Spring Boot es un framework diseñado por Spring Foundation y que usa la licencia Apache License 2.0. Este framework nos permite crear aplicaciones enfocándonos únicamente en el desarrollo, ya que se encarga de las labores de configuración de dependencias y despliegue. Para esto utiliza un servidor de aplicaciones embebido, Tomcat.[8]

Uno de los putos centrales de Spring son los contenedores o más conocidos con el nombre de Beans, estos se encargan de crear los objetos, conectarlos, configurarlos y controlar su ciclo de vida mediante la inyección de dependencias.

El patrón de inyección de dependencias tiene como finalidad conseguir un código más desacoplado que nos facilitará hacer tests, combinar distintas partes del sistema y cambiar las

dependencias en tiempo de ejecución. La inyección de dependencias se podrá hacer mediante un constructor de dependencias en la clase deseada o mediante la anotación @Autowired.[9]

Spring nos ofrece Spring MVC para facilitar la creación de aplicaciones webs siguiendo el patrón Modelo-Vista-Controlador, que es el patrón más utilizado en el mundo del desarrollo web.

Este framework es muy utilizado en el mundo del desarrollo web ya que es gratis, permite el desarrollo de webs y microservicios, funciona sobre JVM lo que facilita la migración de cualquier proyecto ya desarrollado en Java, es rápido y ofrece grandes facilidades a la hora de configuración, despliegue y conexión con bases de datos. Esto hace que grandes empresas en el mundo del software como Netflix, Amazon, eBay, etc lo utilicen para sus desarrollos.

Decidí utilizar este framework ya que aparte de todas las ventajas que ofrece contaba con experiencia previa y es la tecnología con la que estoy actualmente trabajando.

5.2.1.1 Maven[10]

Maven no es simplemente una herramienta para simplificar los procesos de compilación y generación de ejecutables a partir de código fuente. Nos brinda una gran ayuda en todas las etapas del desarrollo de software.

- Valida que un proyecto sea correcto.
- Compila.
- Prueba el código con pruebas unitarias.
- Empaqueta el código compilado y lo transforma a formato .jar o .war.
- Verifica que el código empaquetado cumple los criterios de calidad.
- Facilita el uso de otras dependencias.
- Ayuda a desplegar código en un entrono.

Toda esta configuración de Maven se encuentra en el archivo pom.xml del proyecto Spring.[11]

5.2.1.2 Hibernate

Hibernate es una herramienta ORM (mapeo objeto-relacional) diseñada por Red Hat que facilita el mapeo de las entidades de una base de datos y los modelos de objetos de una aplicación. Por tanto, es la herramienta que nos facilita el desarrollo de la persistencia en nuestras aplicaciones Java. De esta forma, Hibernate nos ayuda a guardar y recuperar datos de una base de datos, facilitando el almacenamiento de información útil que deseamos recuperar en otro momento.

Hibernate ORM se ocupa de la persistencia en bases de datos relacionales como es el caso de la utilizada en este proyecto. Esta tecnología también implementa herencia, polimorfismo, asociación y composición sin la necesidad de interfaces ni clases base.

Ya que fue diseñado para funcionar en un clúster de servidores de aplicaciones nos ofrece una arquitectura altamente escalable en cualquier entorno lo que es de gran ayuda ya que está preparada para soportar peticiones de miles de usuarios.[12]

5.2.1.3 JPA

JPA (Java Persistence API) es una propuesta estándar que ofrece Java para implementar un ORM, es decir nos permite realizar consultas a una base de datos en código Java. La decisión de utilizar JPA frente a JDBC en este proyecto fue que la mayoría de los proyectos de desarrollo web en el mundo empresarial utilizar este estándar.

La diferencia entre estas dos opciones que nos ofrece Java es que cuando hacemos uso de JDBC las consultas a la base de datos debemos hacerlas en lenguaje SQL nativo mientras que con el uso de JPA podemos hacer uso de una nomenclatura Java que se proporciona y en el peor de los casos hacer uso de HQL[13] o SQL nativo.[14]

5.2.2 SQL

SQL (Structured Query Language) es un lenguaje de programación que permite acceder a datos de una base de datos, pudiendo realizar distintas acciones como crear, eliminar, insertar, etc.

Debido a que tenía conocimiento previo de este lenguaje de programación y es el más utilizado para bases de datos relacionales decidí utilizarlo.[15]

5.2.2.1 postgreSQL[16]

PostgresSQL es un sistema de gestión de bases de datos relacionales de código abierto y uso gratuito. Es dirigido por una comunidad de desarrolladores que trabajan sin ánimo de lucro y están apoyados por organizaciones comerciales.

Utiliza MVCC (Acceso concurrente multiversión) para tener una alta concurrencia lo que permite que mientras un proceso actúa sobre una tabla otros procesos puedan acceder a la misma tabla sin producir bloqueos.[17]

Cuenta con un gestor de bases de datos que solo es compatible con postgres, desarrollado también por esta comunidad de desarrolladores, llamado pgAdmin, el cual nos facilita mucho el trabajo SQL.

6. Desarrollo del API REST

6.1 API[18] REST[19]

Una API es un mecanismo que permite la comunicación e intercambio de información entre sistemas, y ofrece un conjunto de funcionalidades sobre un servidor que pueden ser utilizadas por aplicaciones cliente.

Se utilizan para interactuar con un servicio sin necesidad de conocer su arquitectura interna ni tecnología utilizada. Permite aplicar diferentes operaciones durante el acceso y recuperación de los datos facilitando la obtención de la información requerida, es muy común en las APIs el filtrado de datos y la selección del formato de salida.

Las APIs se deben implementar siguiendo algún modelo arquitectónico que permite definir el acceso al servicio, en este caso se ha hecho uso de REST (Transferencia de estado representacional).

REST es una interfaz para conectar varios sistemas basados en el protocolo HTTP, aunque también se pueden usar otros estándares, que nos permite obtener y generar datos en formatos específicos como XML y JSON. Su funcionamiento es simple ya que tan solo consiste en hacer peticiones HTTP indicando el request necesario con el propósito de recibir un response, respuesta de la API.

REST es una interfaz para conectar varios sistemas basados en el protocolo HTTP, aunque también se pueden usar otros estándares, y que se puede desarrollar en varios lenguajes como Java, next.js, Phyton y muchos más.

Las APIs que utilizan el protocolo HTTP se apoya nen los métodos básicos de http que son:

- **Get:** Obtiene información de un recurso.
- **Post:** Crea nuevos recursos.
- **Put:** Modifica los recursos ya existentes.
- **Patch:** Modificar parcialmente un recurso ya existente.
- **Delete:** Eliminar un recurso.

La API REST se diseña para proporcionar acceso a objetos, datos, servicios a través de recursos y que el cliente pueda acceder a ellos. Un recurso tiene un URI o endpoint que identifica de forma única ese recurso y mediante el que lo clientes interactúan mediante el intercambio de información.

Los consumidores del servicio intercambian información en formato XML o JSON enviando y recibiendo datos en estos formatos.

En 2008 Leonard Richardson propuso un modelo de madurez para las API web:

- **Nivel 0:** Definir un endpoint y todas las operaciones como métodos POST.
- **Nivel 1:** Crear distintos endpoints para los distintos recursos.
- **Nivel 2:** Usar métodos HTTP para definir las operaciones.
- **Nivel 3:** Usar hiperdemia, dando soporte en nuestro servicio a recursos como audios, videos y documentos, estaríamos hablando de una API RESTfull.

6.2 Modelo de la base de datos

Antes de comenzar con el desarrollo del back, es necesario diseñar el modelo de la base de datos. Para ello, teniendo en cuenta la lógica de negocio y los requisitos que debe cumplir la aplicación, elaboramos un modelo que satisficiera nuestras necesidades.

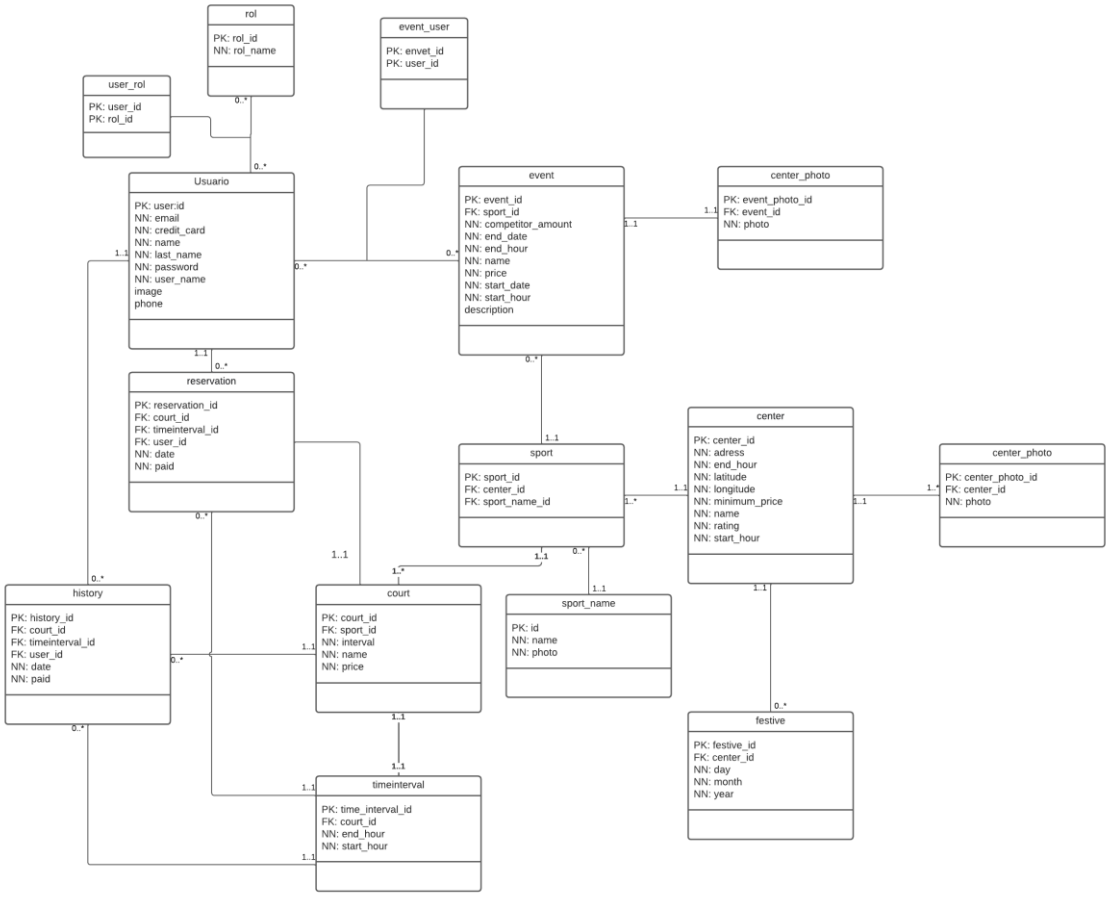

Ilustración 28 - Modelo de la base de datos

En esta imagen se muestra el modelo de la base de datos, donde PK = clave primaria, FK = clave ajena, NN = no nulo, U = único.

Se ha intentado hacer un modelo en tercera forma normal y lo más desacoplado posible para facilitar el crecimiento del modelo de datos y la aplicación en el futuro. Un ejemplo es la separación de usuario y rol, de esta forma cuando sea necesario se pueden añadir nuevos roles.

Usuarios

Contiene la información de los usuarios, así como sus permisos, tiene relación con las reservas y el historial de usuario, así como con los eventos a los que está suscrito.

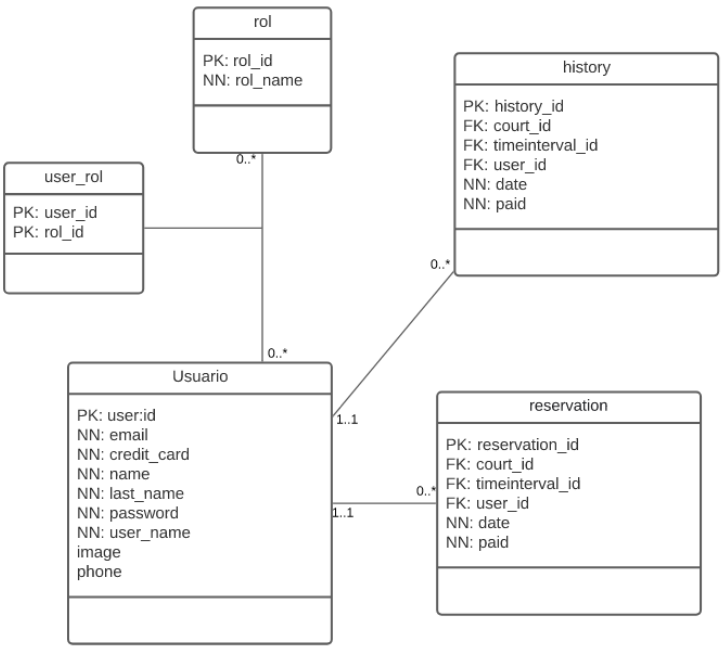

Ilustración 29 - Modelo asociado a los usuarios

- **Usuario:** Contiene toda la información de usuario necesaria para su registro, así como los eventos a los que está inscrito, las reservas realizadas y el historial de reservas.
- **Rol:** Es un aspecto muy importante de la aplicación ya que determinará si un usuario tiene permisos o no para hacer las distintas acciones dentro de la aplicación.
- **User_Rol:** Determina la relación entre usuarios y roles, ya que un usuario puede tener varios roles y un mismo rol lo pueden tener varios usuarios.
- **Reservation:** Almacena toda la información necesaria sobre las reservas activas del usuario.
- **History:** Contiene el histórico de reservas realizadas por el usuario.

Centros

Los centros tienen información al respecto de su ubicación, contacto, nombre y son los encargados de gestionar y crear las pistas con las que cuentan sus instalaciones, así como de organizar eventos.

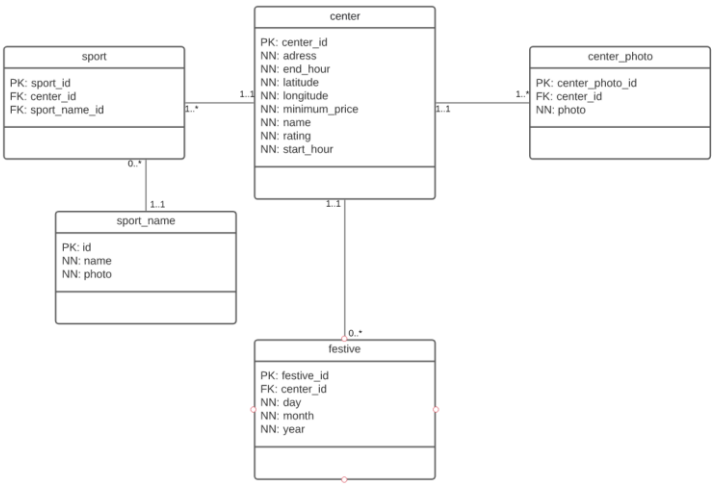

Ilustración 30 - Modelo asociado a los centros

- **Center:** Guarda la información relevante del centro como su ubicación, horario, nombre, y adicionalmente se le asigna un precio mínimo según se van creando pistas y se le van asignando precio y un rating determinado por la votación media de los usuarios.
- **Sport:** Un centro tendrá asignado los distintos deportes que se pueden practicar en él.
- **Sport_Name:** Cada deporte tiene un nombre, se separa en una tabla diferente para conseguir la tercera forma normal, y evitar nombre repetidos en la tabla sport.
- **Festive:** Almacena los días festivos de un centro que se tendrán en cuenta a la hora de consultar pistas disponibles.
- **Center_Photo:** Contiene las fotos de los distintos centros.

Court

Cada pista debe pertenecer obligatoriamente a un centro y está tiene información y tener un deporte asignado, ya que se presupone que en cada pista solo se puede practicar un deporte.

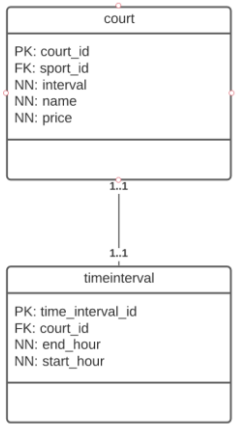

Ilustración 31 - Modelo asociado a las pistas

- **Court:** Cada pista cuenta con un nombre y un precio a parte de tener un deporte.
- **Timeinterval:** Esta tabla almacena el horario de cada pista, el horario se define como los intervalos de tiempo posibles a reservar desde la hora de apertura hasta la hora de cierre. Por ejemplo, en caso de ser una pista que abre a las 8 de la mañana y cierra a las 9 de la noche y el tiempo de alquiler es de 1h en esta tabla se almacenan las horas desde la apertura hasta el cierre, 8:00 - 9:00, 9:00 - 10:00…

Event

Los eventos deben estar asignados a un deporte que se pueda practicar en el centro que publica dicho evento y contiene información útil para el usuario, así como las condiciones para la inscripción.

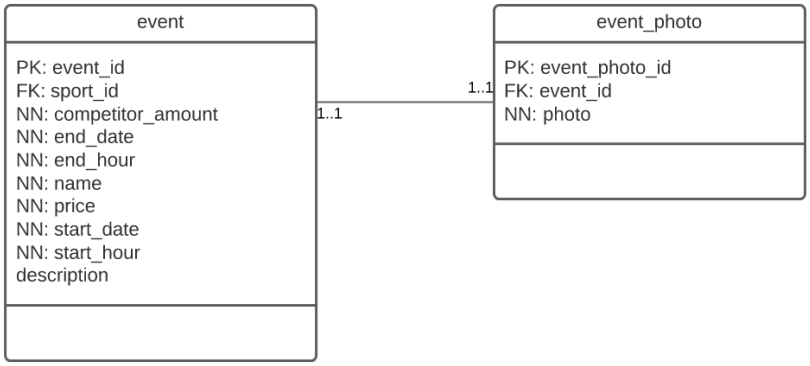

Ilustración 32 - Modelo asociado a los eventos

- **Event:** Contiene información útil del evento como el día de inicio y fin, horario, descripción, precio, participantes y el deporte que se va a practicar.
- **Event_Photo:** Cada evento tendrá sus fotos y se guardarán en esta tabla.

6.3 Estructura y arquitectura del proyecto

La arquitectura del proyecto viene definida por la tecnología comentada en el apartado anterior. Se ha desarrollado un backend envuelto en un servidor Tomcat proporcionado por Maven, en este servidor se desarrollará la aplicación, que con la ayuda del estándar JPA e Hibernate se accede a la base de datos. Todo esto que compone el backend debe ser accesible por los consumidores, en este caso un front.

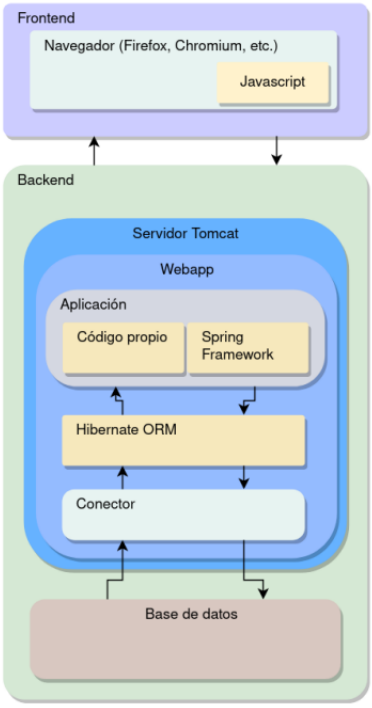

Ilustración 33 - Arquitectura de Appuntate

En cuanto a la arquitectura funcional del proyecto se ha optado por un modelo sencillo en el que se utiliza Hibernate para acceder a la base de datos, los servicios para la lógica de negocio y los controladores para la conexión con el front, como se muestra en la figura 24.

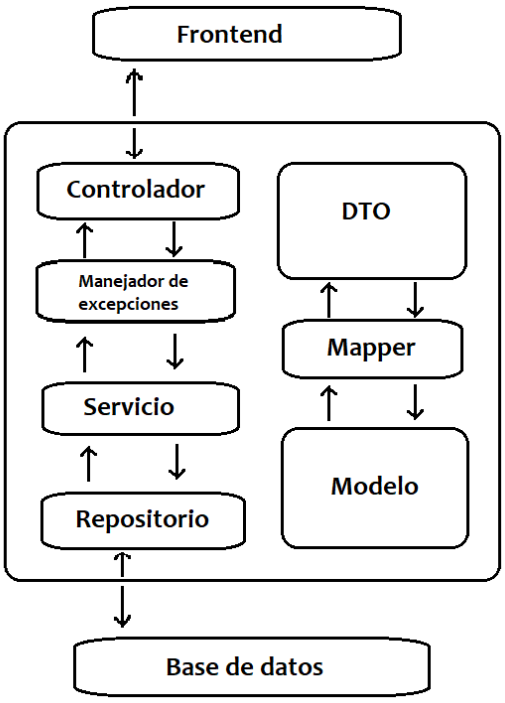

Ilustración 34 - Arquitectura funcional de Appuntate

- **Frontend:** Representa a los consumidores de la api, en este caso será una aplicación de ionic.
- **Controlador:** Es la parte que expone a la API para ser consumida mediante los distintos endpoints y a la cual llegan las peticiones HTTP. El controlador devuelve información útil para el consumidor en formato JSON.
- **Manejador de excepciones:** Se utiliza para manejar los errores HTTP, evitando que la aplicación se quedé bloqueada por una excepción que no se esperaba y para evitar posibles errores a la hora de consumir la api y que estos errores estén bien informados.
- **Servicio:** Los servicios implementan toda la lógica de negocio, es la parte más compleja y trabajan con los DTOs. También se encarga de hacer uso del manejador de excepciones y de los mappers.
- **Repositorio:** Es el encargado de proporcionar acceso a los datos, proporcionando acceso a la base de datos tanto para consultar como para insertar y editar.
- **Base de datos:** Lugar de almacenamiento de información estructurado en tablas y estas en columnas, a las que se puede acceder para consultar datos.
- **Modelo:** Es una representación en una clase Java de una tabla de la base de datos con sus columnas. A estos modelos también se les añade información como las relaciones entre tablas, valores no nulos, únicos y la clave primaria.
- **DTO:** Data Transfer Object, son clases Java sin ninguna lógica que tienen el fin de replicar los modelos de una forma sencilla con la que trabajan los servicios. También se utilizan para recibir y enviar información a los consumidores.
- **Mapper:** Es el encargado de llevar a cabo la transformación entre modelos y DTOs, está transformación se debe poder hacer en ambos sentidos.

Encontramos dos flujos principales uno en el que el front pide datos y otro en el que el front guarda datos.

- **Flujo de solicitud de datos:** El front hace una petición al back, este la recibe por el controlador, y llama directamente al servicio que se encarga de hacer una petición a la capa de persistencia desarrollada con el estándar JPA y esta recupera los datos requeridos con la ayuda de Hibernate. Estos datos son enviados de nuevo al servicio el cual se encarga de modificar los datos con la ayuda del mapper y controlar las posibles excepciones que pueda haber, para finalmente pasar los datos al controlador y que este los envíe al front.

- **Flujo de guardado de datos:** El front envía los datos que desea guardar al back, el cual recibe la información en el controlador, este llama directamente al servicio que se encarga de transformar los datos enviados por el front a entidades que reconoce la base de datos a través del mapper y de controlar las posibles excepciones. Una vez conseguida una entidad preparada para ser guardada en la base de datos, el servicio hace uso de la capa de persistencia para guardarla.

Estos son los dos flujos principales, pero también se da el caso en que combina ambos flujos, ya que una vez guardados los datos (flujo de guardado de datos) se devuelve información al front haciendo uso del flujo de pedida de datos.

Para estructurar el proyecto se ha decidido dividirlo en cuatro paquetes principales, repository para el acceso a datos, service para el desarrollo de la lógica de negocio, model para definir los modelos y controller para ofrecer puntos de acceso a la API.

Esta estructura es la principal, y sobre la que en un principio se comenzó a desarrollar el proyecto, pero a lo largo que se avanzaba se requería de más paquetes para una mejor organización, quedándose una estructura final como se muestra en la siguiente imagen.

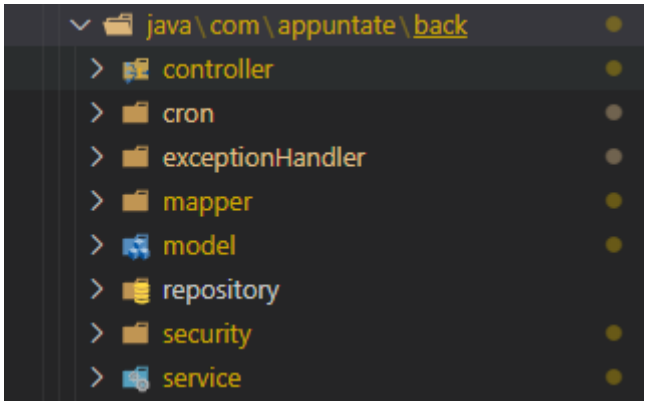

Ilustración 35 - Estructura de carpetas del proyecto

Como se puede observar en la imagen 26 se han creado más paquetes de los pensados en un principio y que se muestran en los siguientes apartados.

6.3.1 Entity

Para la creación de entidades se ha elaborado un diagrama de clases en el que queden representadas las distintas clases Java con sus atributos y las relaciones entre ellas. Gracias a Hibernate las clases Java se mapean directamente a tablas en la base de datos, por lo tanto el diagrama de clases y diagrama de la base de datos será muy similar, en el único caso en el que no se mapea una tabla por una entidad es en las relaciones muchos a muchos (*..*) ya que Hibernate crea una tabla relación que no queda reflejada en las clases Java.

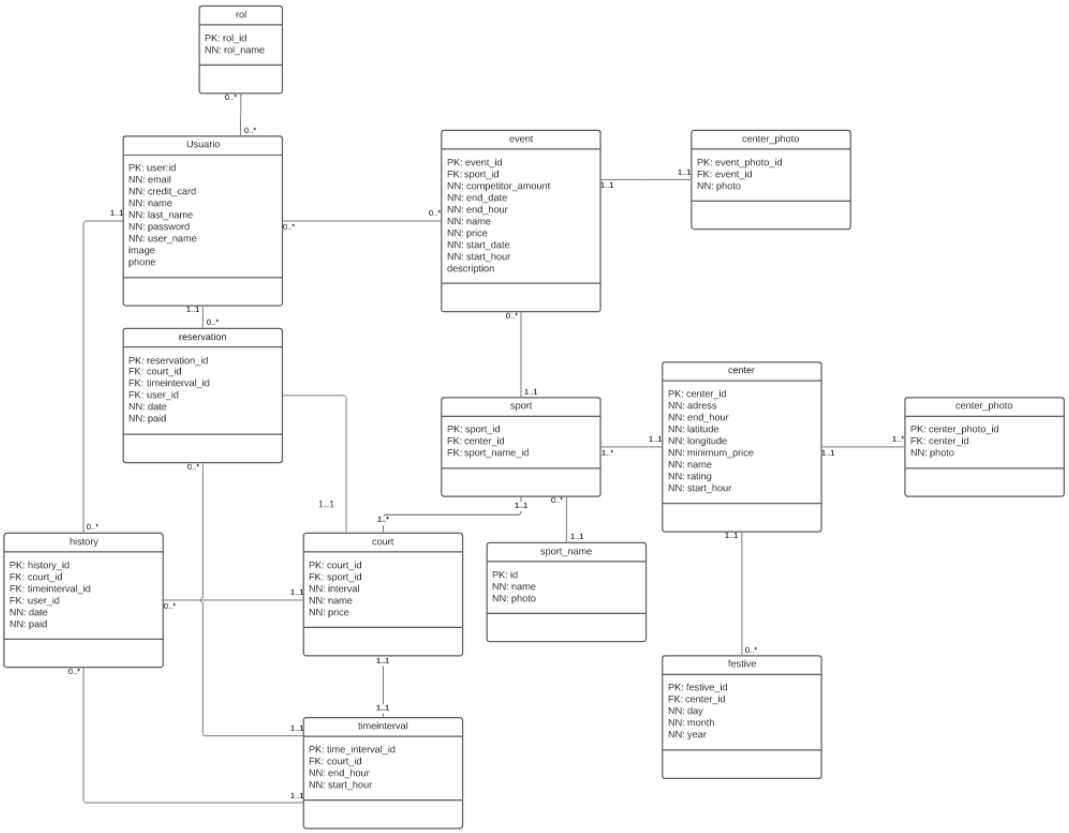

Ilustración 36 - Diagrama de clases

Para que se reconozca una clase Java como entidad a mapear en la base de datos es necesario incluir la anotación @Entity. Se ha optado por una generación automática de las claves primarias de las entidades @GeneratedValue(strategy=GenerationType.AUTO).

Para el caso especial comentado anteriormente de la relación *..* es necesario utilizar una notación @ManyToMany e indicar que se va a generar una tabla intermedia con dos columnas haciendo referencia a las claves primarias de las entidades que relaciona, de esta forma se genera una tabla relación que no queda representada en una clase Java.

```
@ManyToMany(fetch = FetchType.EAGER)
@JoinTable(name = "user_rol",
    joinColumns = @JoinColumn(name = "user_id"),
    inverseJoinColumns = @JoinColumn (name = "rol_id"))
private List<Rol> rols;
```

Ilustración 37 - Relación muchos a muchos

6.3.2 Repository

Los repositorios son los archivos java que se encargan de la persistencia y mantiene en contacto la base de datos con el resto de la aplicación. Para la comunicación con la base de datos se utiliza es estándar JPA por lo que todos los repositorios extienden de JpaRepository.

JpaRepository ofrece métodos básicos para el acceso a datos por clave primaria, pero para consultas y acciones más complejas hay que hacer uso de la creación de métodos propios bien con la ayuda del estándar JPA o en el caso de las más complejas recurriendo al lenguaje HQL, aunque también se puede hacer uso de SQL indicando que es una query nativa.

El estándar JPA nos permite generar métodos de una forma muy sencilla añadiendo en primer lugar la acción que se va a realizar find, update, delete seguido de los campos que se van a tener en cuenta para la búsqueda. Como funcionalidad adicional JPA nos proporciona una nomenclatura para acciones muy recurridas en las consultas a base de datos como pueden ser orderBy, distinct, ignoreCase, etc.

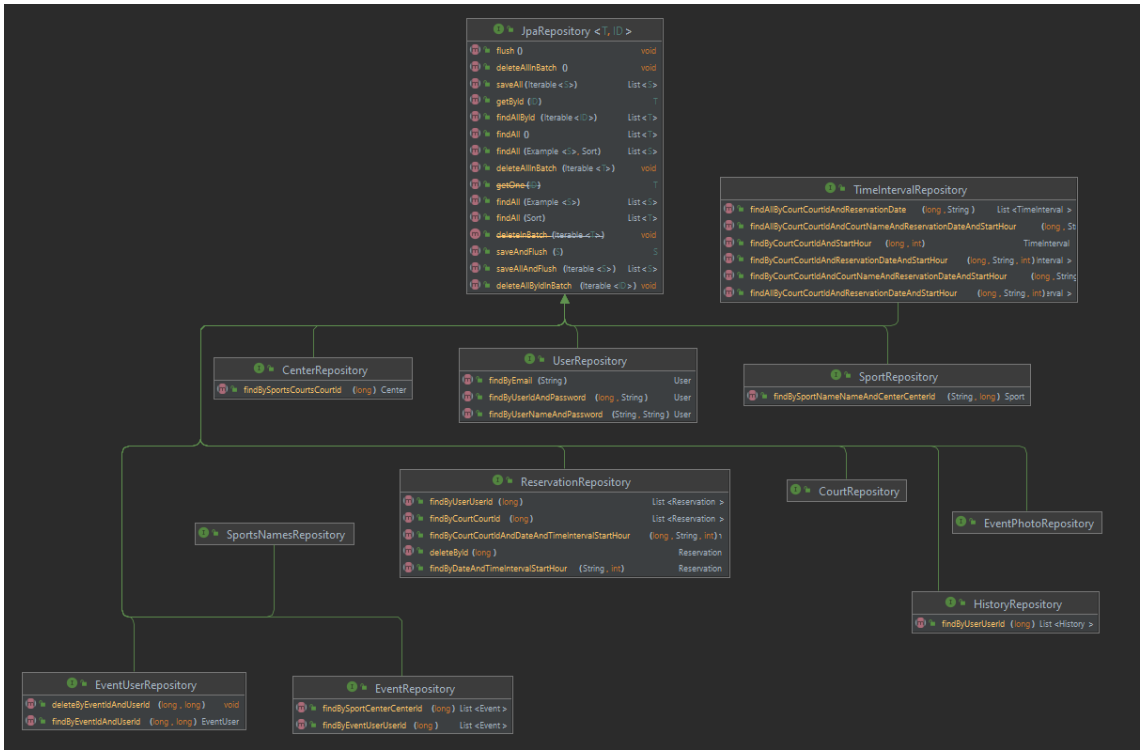

Ilustración 38 - Repositorios y sus dependencias

En la imagen 28 se muestra los distintos repositorios creados, con sus método y como estos extienden de la misma interfaz, JpaRepository.

6.3.3 Service

Los servicios son los encargados de que todo funcione correctamente siguiendo las reglas establecidas en el mundo real, es decir, implementan la lógica de negocio para que todo funcione según las especificaciones detalladas.

A la hora de desarrollar los servicios entran en juego muchos factores como estándares descritos en el libro clean code[20], como métodos que solo hacen una cosa, nomenclatura de métodos y variables, niveles de anidación en bucles y condiciones y muchos más. Estos estándares y consejos se pueden ver también en las demás partes del proyecto, pero en los servicios es donde más importancia toma y donde más importancia toma su uso.

Como añadido también es el lugar donde se hacen presentes el uso e implementación de patrones de diseño. En este proyecto se ha hecho uso del patrón fábrica[21] para la creación de pistas y eventos, aunque ahora mismo todas las pistas son iguales se ha hecho uso de este patrón para facilitar en un futuro el desarrollo de creación de instalaciones distintas como puede ser una piscina, y el patrón singleton[22] para la obtención de algunos datos como el listado de los nombres de los deportes.

Me gustaría añadir que los patrones de diseño y el código limpio y mantenible despertó en mi gran pasión, lo que me llevó a realizar varios cursos[23] que me han ayudado a desarrollar este proyecto con el uso de buenas prácticas.

Para favorecer una estructura limpia del proyecto se ha decidido clasificar los servicios en 3 partes.

- **Service:** Contienen la funcionalidad principal de la lógica de negocio y acceso a datos.

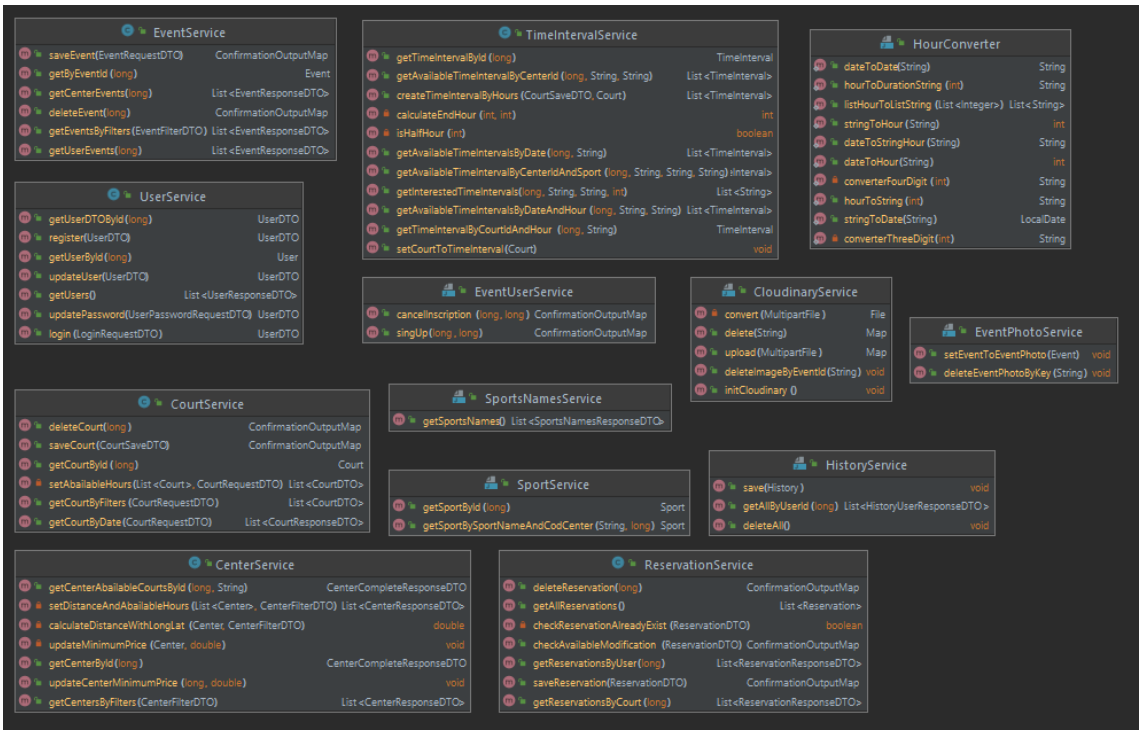

Ilustración 39 - Servicios y sus dependencias

- **CriteriaService:** Los utilizo para crear los criterios de búsqueda en los filtros dinámicos, lo que me permite realizar esta tarea de forma desacoplada y limpia apoyándome de una interfaz.

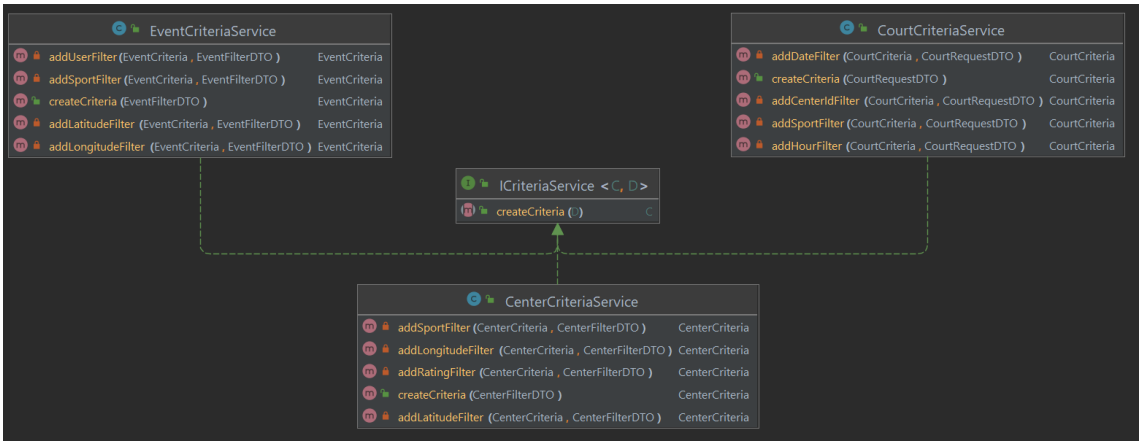

Ilustración 40 - Servicios criteria y sus dependencias

- **SpecificationService:** Me permiten crear las especificaciones con las que posteriormente haré la llamada a la base de datos, al fin y al cabo, me permite crear un consulta SQL con código Java.

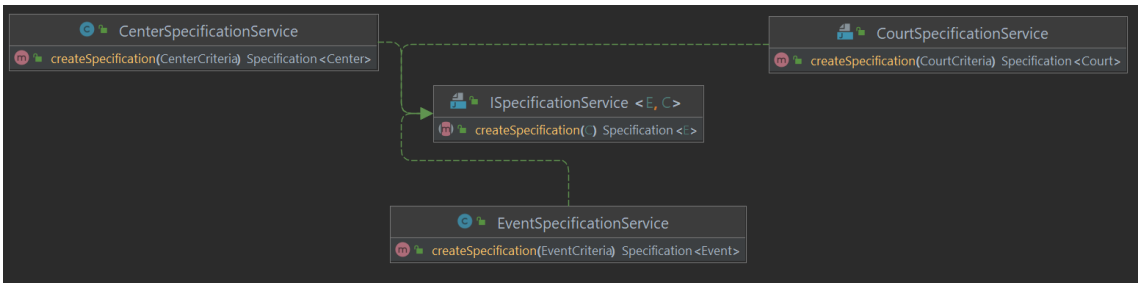

Ilustración 41 - Servicios specification y sus dependencias

6.3.4 DTO

El patrón DTO[24] se utilizan para transportar datos entre procesos reduciendo la cantidad de llamadas a métodos, estos DTOs no contienen ni forman parte de la lógica de negocio, simplemente es una ayuda para el tratamiento de objetos, es decir, para recibir objetos del front, tratarlos en el back y enviarlos de vuelta al front. Por lo que por cada entidad ha creado:

- **RequestDto:** Objeto que recibe el controlador con información enviada desde el front.
- **ResponseDto:** Objeto enviado a front con la información que solicita para crear las vistas.
- **Dto:** Objeto utilizado para desarrollo de la lógica de negocio, ya que por buenas practicas no se utiliza en la lógica de negocio directamente las entidades.

- **FilterDto:** Objetos utilizados para generar los filtros.

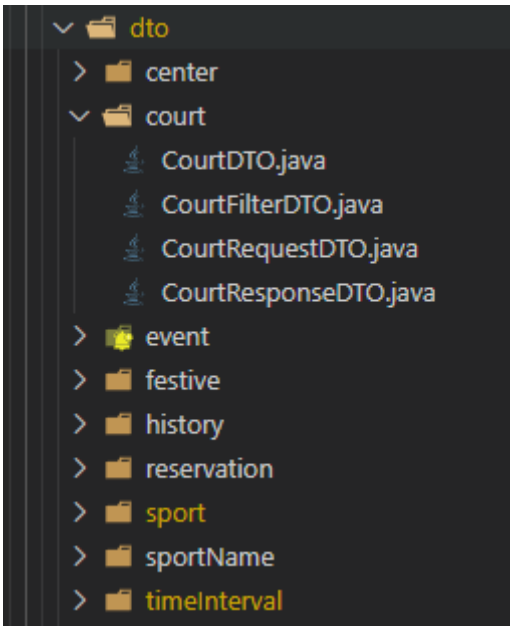

Ilustración 42 - Estructura de los DTO

6.3.5 Mapper

Los mappers se encargan únicamente de la conversión de entidades y DTOs, de esta forma una vez creado los mapper es muy fácil trabajar con los distintos objetos que se han expuesto en el apartado anterior.

En un principio pensé en utilizar la librería MapStruct que genera automáticamente estos mappers y ofrece gran facilidad, pero encontré ciertos problemas a la hora de desplegar el proyecto en AWS utilizando esta librería, por lo que finalmente los cree de forma manual.

Para favorecer el desacoplamiento y facilitar la creación de nuevos mappers pensé en crear una interfaz y que todos la implementaran.

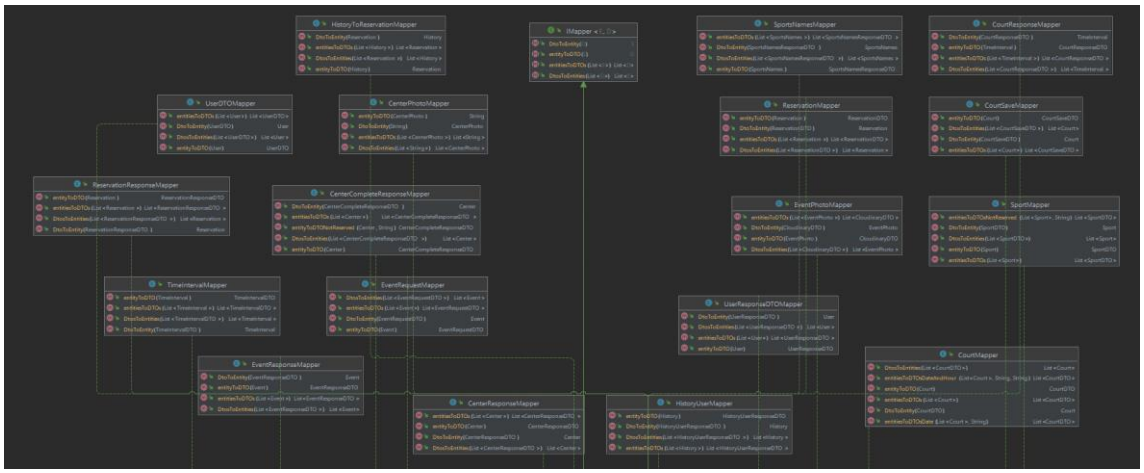

Ilustración 43 - Mapper y sus dependencias

Como se observa en la imagen 33 se ha creado una interfaz que define los métodos para los distintos casos de transformación que me he encontrado, transformaciones de entidades y listas de entidades a dtos y viceversa. De esta forma un mapper solo tiene que implementar esta interfaz y dar funcionalidad a los métodos.

6.3.6 Controller

Los controladores es la parte que se expone de la API y a la cual van a llegar las peticiones del front. En esta parte del proyecto se intentará incluir la menor lógica posible, simplemente hará de comunicador entre el front y los servicios.

Para el desarrollo de los controladores se ha tenido en cuenta dos aspectos importantes, los endpoints y los métodos HTTP.

- **Endpoint:** Es la ruta de la API mediante la que el consumidor, en este caso el front, puede conectarse para hacer las distintas llamadas. Para la nomenclatura de los endpoints se ha decidido utilizar sub-recursos y nombres descriptivos. Por ejemplo, para la autenticación de usuarios se ha utiliza el endpoint /auth/login.
- **Métodos HTTP:** Como se mencionaba en el apartado 6.1 API REST para cada tipo de operación HTTP nos aporta un método distinto.

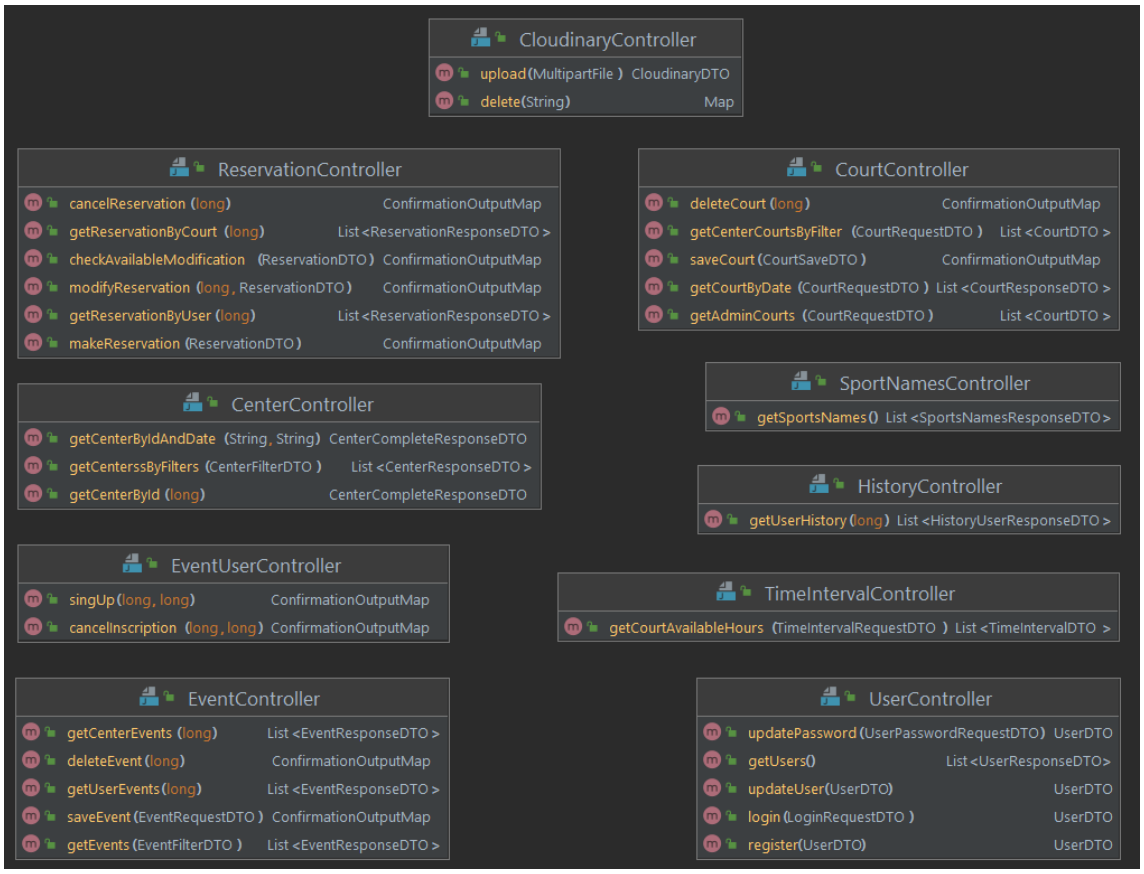

Ilustración 44 - Controller

Como se puede observar en la imagen 34 hay varios controladores pero cada uno de ellos se centra en una funcionalidad especifica.

- **CloudinaryController:** Lo utilizo para guardar y eliminar imágenes del repositorio de fotos que utilizo, Cloudinary[25].
- **ReservationController:** Gestiona todo lo relacionado con las reservas, desde realizar reservas, editarlas y eliminarlas hasta devolver las reservas de un usuario.
- **CourtController:** Nos permite crear, eliminar y editar pistas, también nos ofrece un listado de pistas disponibles para una fecha dada.
- **CenterController:** Se utiliza para obtener información de centros, hay 3 métodos disponibles para ser usados según convenga.
- **SportNamesController:** Ofrece un listado de los deportes.
- **EventUserController:** Permite inscribirse y cancelar inscripción a eventos.
- **TimeIntervalController:** Nos muestra los horarios disponibles de una pista determinada para una fecha determinada.
- **EventController:** Gestiona todo lo relacionado con los eventos, creación, edición, eliminación y obtención de eventos.
- **UserController:** Nos permite gestionar un usuario, haciendo el registro, el login y actualizando su información.

6.3.7 exceptionHandler

Para el manejo de errores HTTP se ha elegido hacerlo a partir de excepciones ya que spring nos facilita el manejo de excepciones y puedo sincronizarme a dicho manejador y completarlo según mi conveniencia. Se trata de un método sencillo y que nos aporta gran escalabilidad ya que es un desarrollo muy lineal y totalmente desacoplado.

Por tanto, se ha montado un manejador de excepciones que centraliza todas las excepciones y cada tipo va asociada a un error HTTP, con la intención, no de mandar al usuario final un mensaje de error, si no para informar al consumidor de la API que algo no está funcionando y aportar un mensaje significativo, no como hace por defecto el manejador de spring que envía por ejemplo “Bad Request”, facilitando así la consumición de la API.

A su vez este tratamiento de excepciones ayuda a proteger la API, es cierto que el desarrollador de front tiene que evitar que el usuario final haga peticiones no aceptadas como registrar dos veces un usuario, pero yo no puedo fiarme de que el desarrollador front lo vaya a hacer bien, por tanto, debo cubrirme las espaldas evitando que este tipo de peticiones se realicen.

Lo primero que hice fue crear la estructura de los errores que iba a manejar, por tanto, definí que un error va a tener la excepción lanzada por spring, un mensaje detallado del error y el path que ha provocado dicha excepción.

A continuación, hicé el manejador que asigna las excepciones a errores HTTP.

Para este proyecto solo se han necesitado de tipo

- **404 NOT_FOUND:** Cuando un recurso no se ha encontrado.
- **400 BAD_REQUEST:** Cuando se ha hecho una mala petición.
- **403 FORBIDDEN:** Cuando un usuario no tiene permisos para realizar una acción.
- **401 UNAUTHORIZED:** Cuando un usuario intenta hacer una acción que requiere login previo.

Lo último que hice fue crear las excepciones que iba necesitando y posteriormente las añadía al manejador. Es normal tener muchas excepciones, incluso cuantas más se tengan más protegido estará tu código y más ayuda tendrán los consumidores de la API.

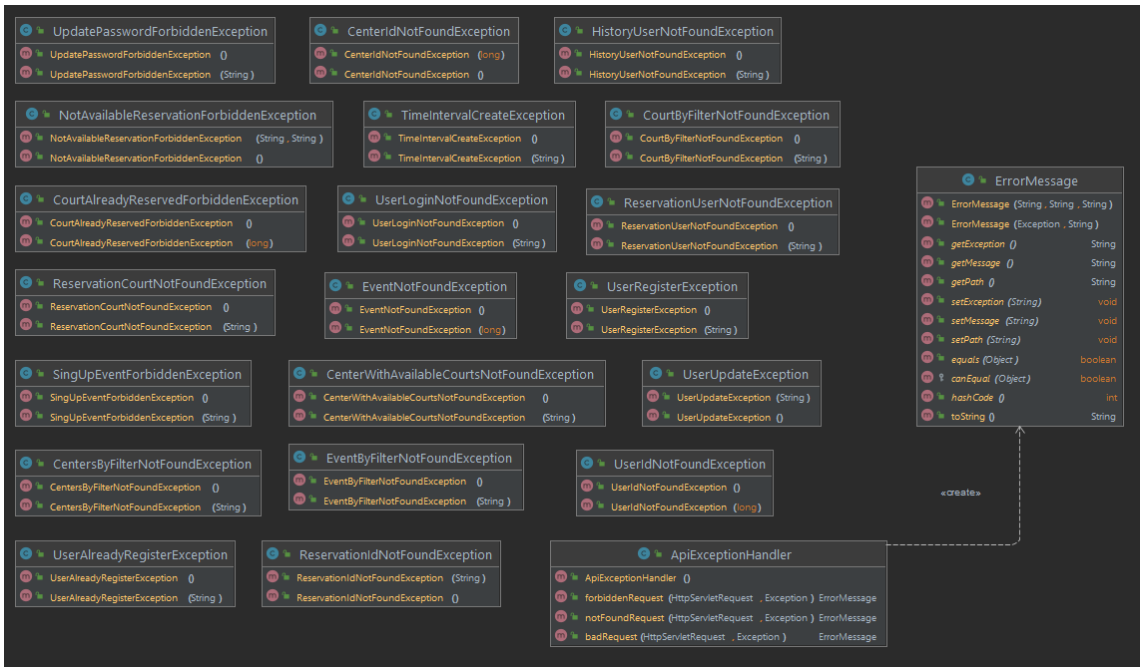

Ilustración 45 - Manejador de excepciones y sus dependencias

Para crear las excepciones es importante poner nombres significativos y siguiendo una estructura determinada como se muestra en la imagen 35. Primero se da una descripción de la excepción, después se indica el tipo y por último se indica que esa clase es una excepción.

6.4 Pruebas con Postman

Algo que ha cobrado gran importancia en este proyecto son las pruebas realizadas con Postman. Como comentaba en el apartado 5.1.3 Postman es una herramienta que nos permite probar nuestra API realizando llamadas HTTP.

Se ha creado una colección que contiene todas las llamadas a los diferentes endpoints de la API y de esta forma poder probar toda la funcionalidad que se ofrece en el servicio REST. Ha resultado de gran ayuda el uso de esta herramienta ya que se ha podido compartir la colección con mi compañero del front para que pudiera ver como realizar las distintas llamadas a los endpoints, teniendo en cuenta el header y el body.

También se ha utilizado a modo de tests de integración ya que cada vez que hacía un nuevo desarrollo ejecutaba todas las llamadas guardadas en la colección de Postman para verificar que seguían teniendo el comportamiento esperado.

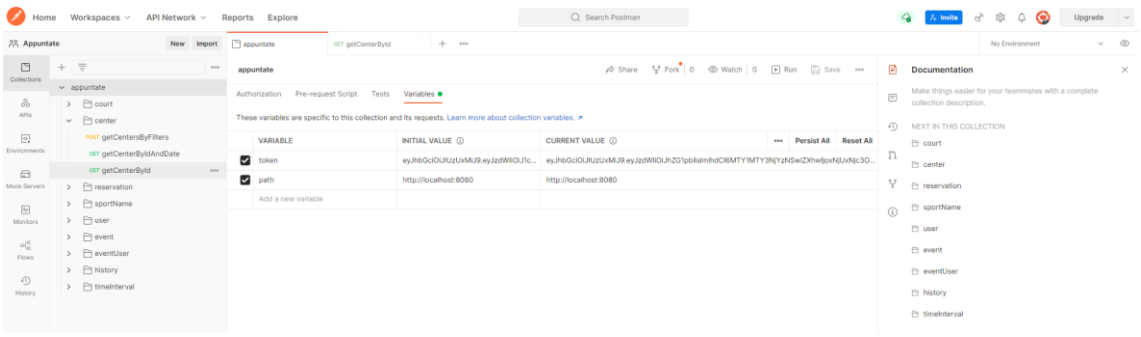

Ilustración 46 - Varialbes de la colección de Postman

Como se muestra en la imagen 38 se han creado dos variables en la colección, una para guardar el token JWT que se explicará en el apartado 6.6.3 Token JWT, y otra variable para almacenar el path al que se ataca de esta forma es muy cómodo por si se cambia el puerto local o se quiere apuntar al proyecto desplegado.

Antes de realizar cualquier llamada es necesario hacer register o login ya que son las llamadas que devuelven el token JWT que habilita autorización pare realizar las demás llamadas. Todas ellas deben contener el token JWT, unos headers y en algunos casos un body en el que el front pasa información necesaria para que el back pueda guardar datos, aplicar filtros, etc.

Como resultado de estas llamadas a los endpoints de la API se pueden obtener distintas respuestas, todas en formato JSON, que muestro a continuación.

- **Respuesta esperada:** Se obtiene una respuesta con información útil para el front que le ayude a mostrar los datos a los usuarios. En el caso de register y login se añade al cuerpo de la respuesta el token JWT.

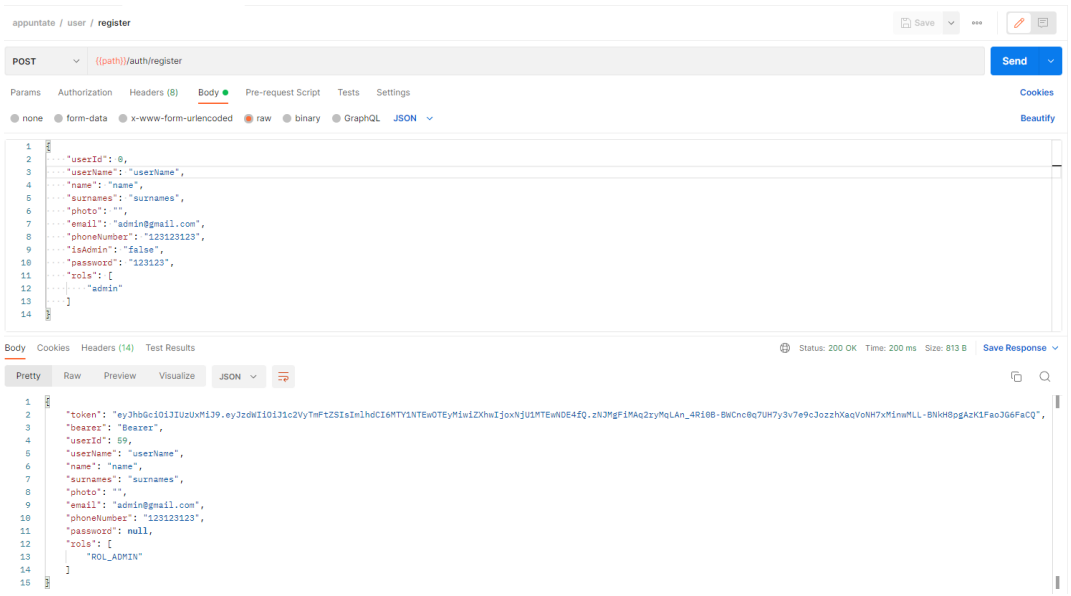

Ilustración 47 - Llamada register en Postman

- **Error autorización:** En caso de no incluir el token JWT al header de una llamada que no sea login o register, o que se intente hacer una operación que no está permitida para los roles asignados al usuario que intenta realizar dicha operación, se obtendrá una excepción indicando que el usuario no está autorizado.

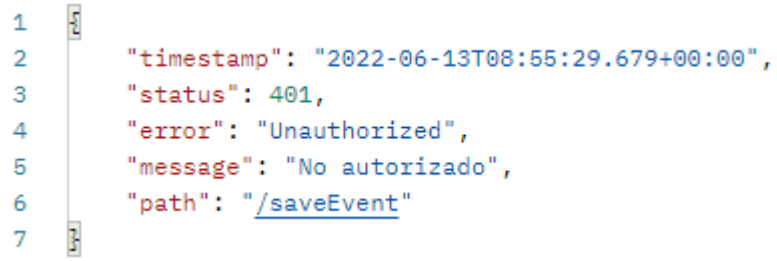

Ilustración 48 - Error autorización Postman

En la imagen 39 se muestra el resultado obtenido cuando un usuario con rol de usuario intenta crear un evento, acción reservada para usuarios con rol de administrador.

- **Excepción controlada:** Como se ha comentado en el apartado 6.3.7 exceptionHandler llevo a cabo un control de excepciones que se devuelven en formato JSON con la siguiente estructura.

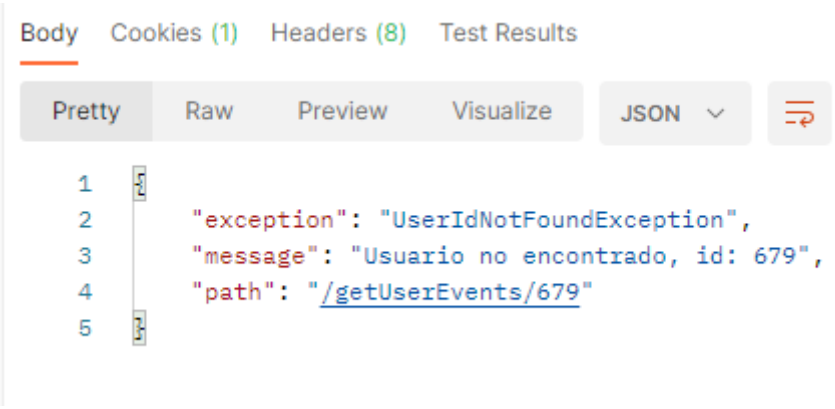

Ilustración 49 - Excepciones en Postman

La imagen 40 muestra la respuesta de una llamada que devuelve una excepción. Al intentar coger los eventos del usuario 679 se lanza una excepción UserIdNotFoundException que nos indica que el usuario con id 679 no ha sido encontrado.

6.5 Comportamiento del sistema

En este apartado voy a detallar con diagramas de secuencia el comportamiento que cumple el sistema para satisfacer todos los requisitos mencionados anteriormente y hacer posible que todo funcione como se espera.

Todos los diagramas parten del controlador ya que es el punto de acceso de la API y todos ello terminan también en el controlador, ya que siempre se devuelve una respuesta al consumidor. Se han omitido los métodos privados, constructores, getters/setters y las excepciones lanzadas para hacer una lectura más intuitiva y fácil de los diagramas.

6.5.1 Centros

Obtener un centro por Id

Esta característica permite al consumidor de la API obtener toda la información detallada de un centro dado su id. Como se observa en el diagrama de secuencia, una operación sencialla puede ser algo compleja ya que intervienen distintos mappers y conversores de fecha y hora.

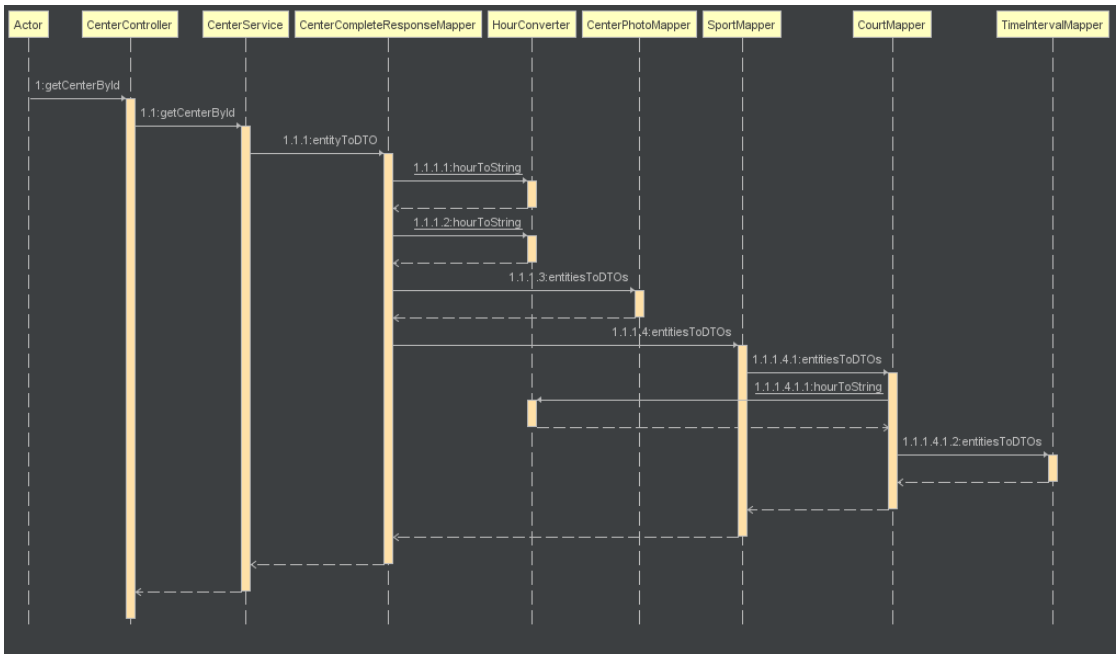

Ilustración 50 - Diagrama de secuencia - Obtener un centro por Id

Obtener centros mediante filtros

Esta funcionalidad es la que permite al consumidor dar la opción de que el usuario pueda realizar un búsqueda dinámica mostrando solo como resultado centros que cumplen los requisitos indicados según: deporte, día, hora, valoración y ubicación.

Para llevarla a cabo el servicio encargado de gestionar la lógica de los centros crea un criterio de búsqueda y a partir del mismo una especificación que finalmente se utiliza para llamar al repositorio, en una llamada incluida en la interfaz de JpaRepository findAll, que encuentra todos los centros según los filtros indicados.

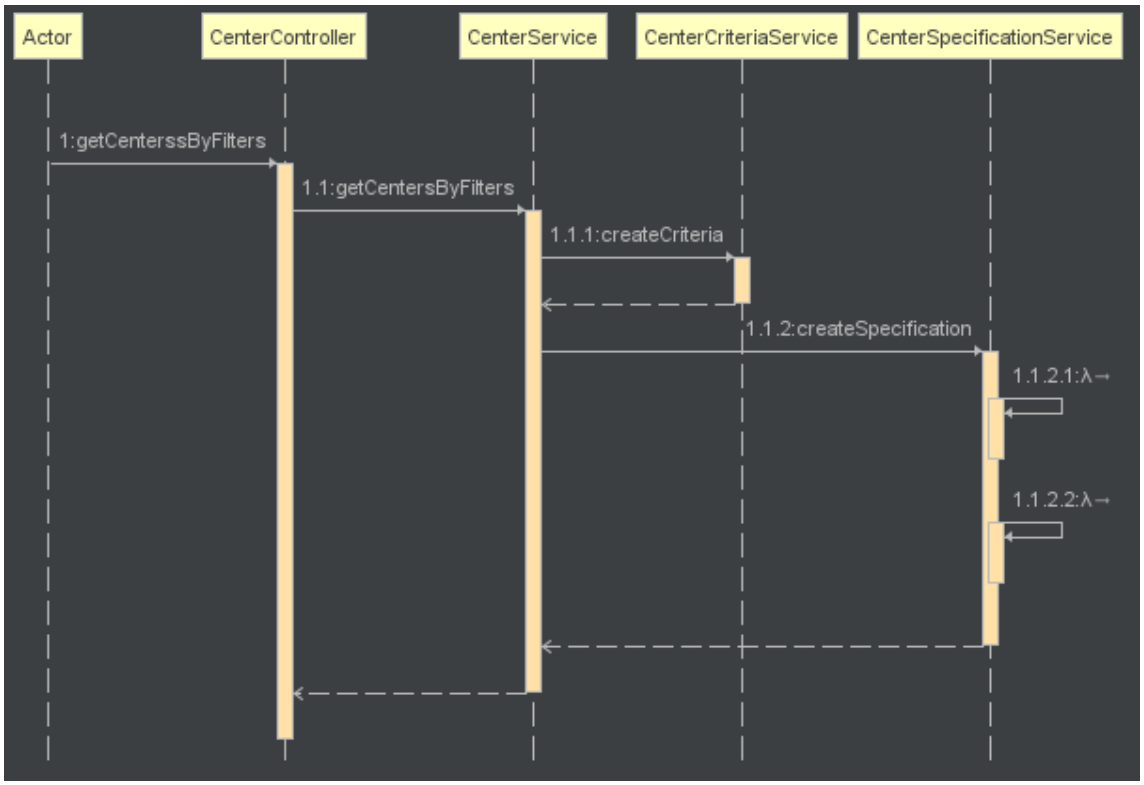

Ilustración 51 - Diagrama de secuencia - Obtener centros mediante filtros

Obtener información de un centro en una fecha dada

Este flujo permite al consumidor obtener toda la información relevante de un centro, incluyendo sus pistas disponibles para una fecha indicada.

Para ello primero busca el centro indicado y posteriormente obtiene solo las pistas disponibles para la fecha indicada.

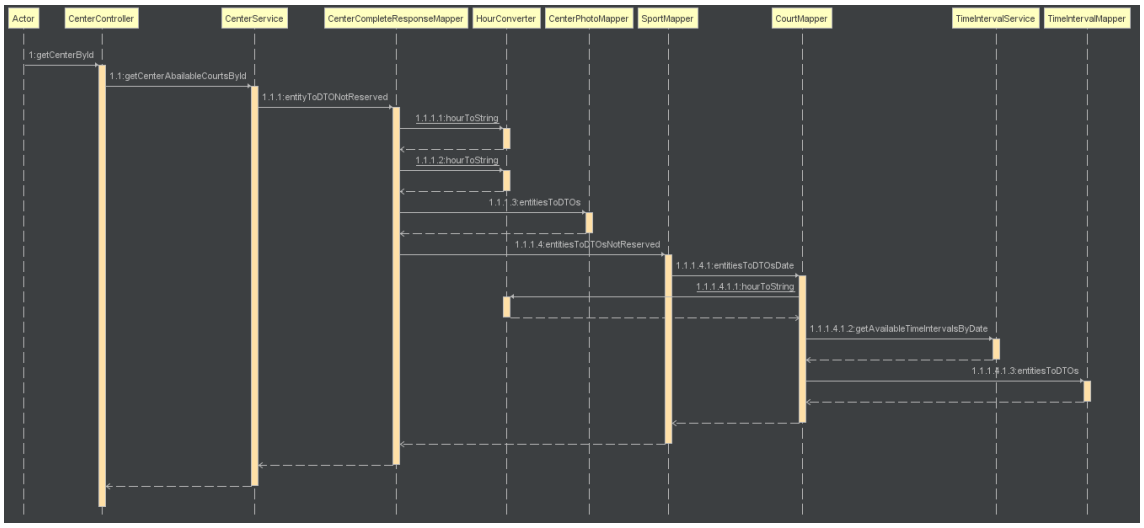

Ilustración 52 - Diagrama de secuencia - Obtener centros por fecha

6.5.2 Pistas

Guardar pista

Esta acción permite al front guardar una pista dando los datos necesarios para la creación. Esta es la forma la cual los centros pueden gestionar sus instalaciones.

Se puede apreciar en el diagrama de flujo como hay múltiples conversiones, principalmente de horas y fechas ya que estas se guardan en la base de datos como un string y de entidades, como se puede observar se hace uso de un mapper que transforma la información enviada por el front a una entidad que reconoce Hibernate para guardarla en base de datos. Tras el proceso de guardado se debe dar una respuesta u otra según si la operación ha sido realizada con éxito o no.

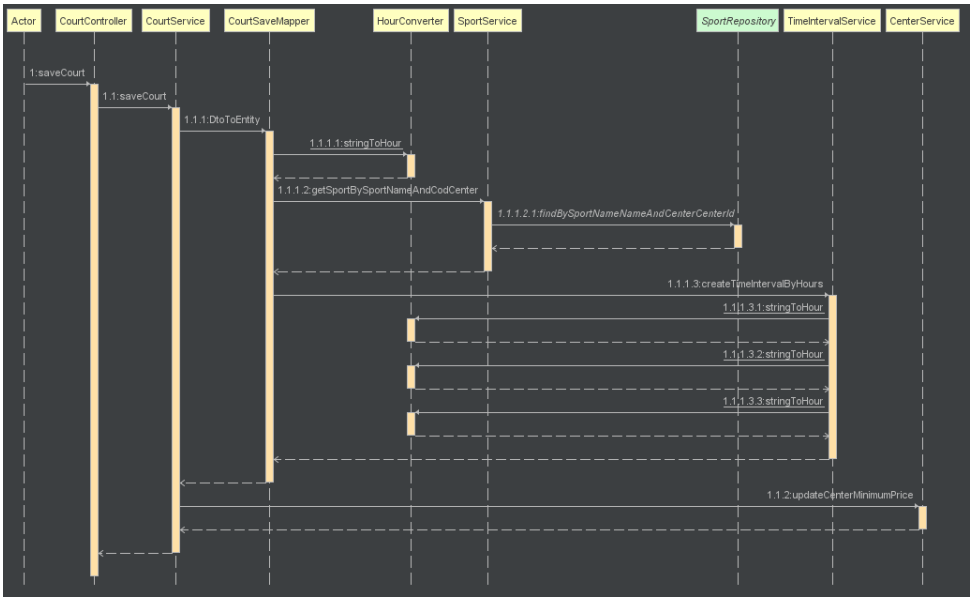

Ilustración 53 - Diagrama de secuencia - Guardar pista

Obtener pistas por fecha

Mediante este flujo el front puede obtener las pistas que tiene disponibles un centro para una fecha y una hora dada.

Es un flujo sencillo, de obtención de datos según unos criterios dados y mapeo de la entidad obtenida para devolverla al front.

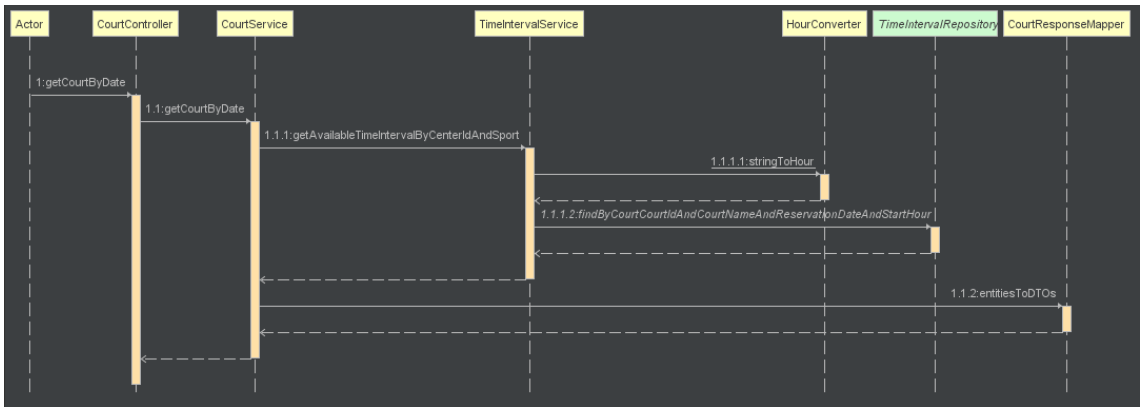

Ilustración 54 - Diagrama de secuencia - Obtener pista por fecha

Obtener pistas mediante filtros

Al igual que el consumidor requiere que se devuelvan centros según unos filtros dinámicos, también necesita poder obtener pistas de un centro determinado filtradas dinámicamente.

Es un diagrama muy similar a la obtención de centros mediante filtros ya que se ha implementado siguiendo la misma lógica.

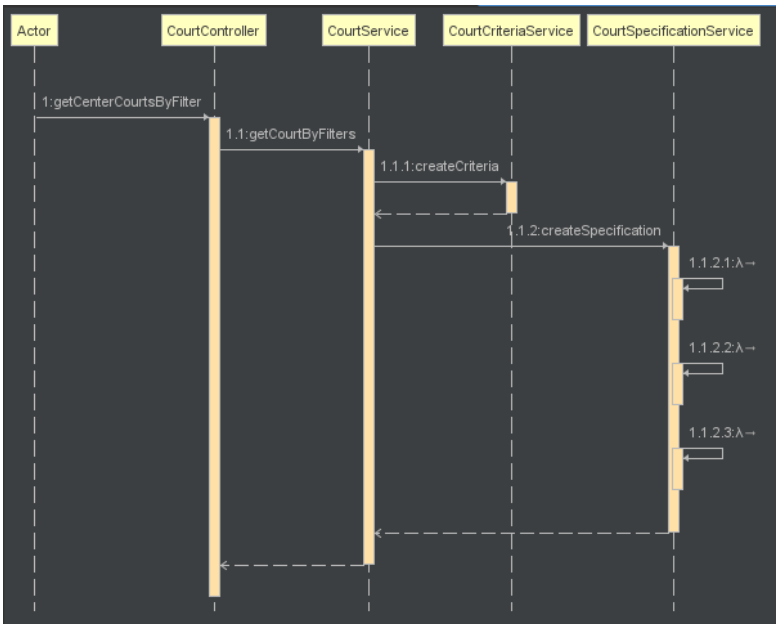

Ilustración 55 - Diagrama de secuencia - Obtener pistas mediante filtros

Eliminar pista

Un centro debe tener la posibilidad de eliminar una de sus pistas de las instalaciones, por ello se proporciona un endpoint que hace esto posible.

Es un flujo muy sencillo ya que la eliminación se hace por el id de la pista que se pasa como param en la request y no se necesita ningún mapper ni operación adicional más que el borrado en base de datos y para finalizar devolver una respuesta u otra según si se ha completado correctamente la operación o no.

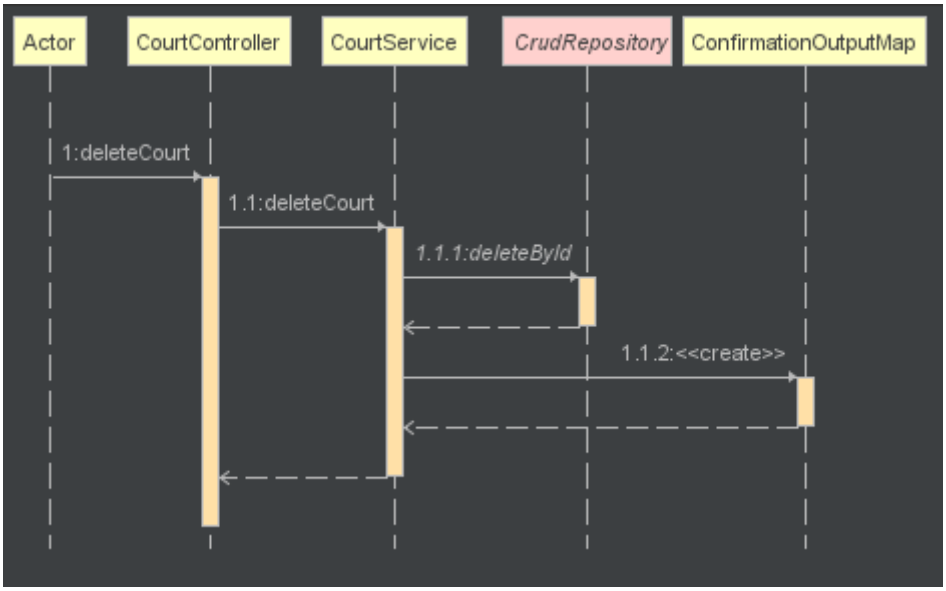

Ilustración 56 - Diagrama de secuencia - Eliminar pista

6.5.3 Eventos

Guardar evento

Los administradores de los centros organizan eventos y torneos y estos deben quedar reflejados en la aplicación para que los usuarios puedan ver información sobre el evento e inscribirse.

Para ello se ha generado un flujo sencillo de guardado de eventos que mapea lo que se recibe del front para posteriormente guardarlo en la base de datos y devolver una confirmación de evento creado en caso de que no haya saltado ninguna excepción.

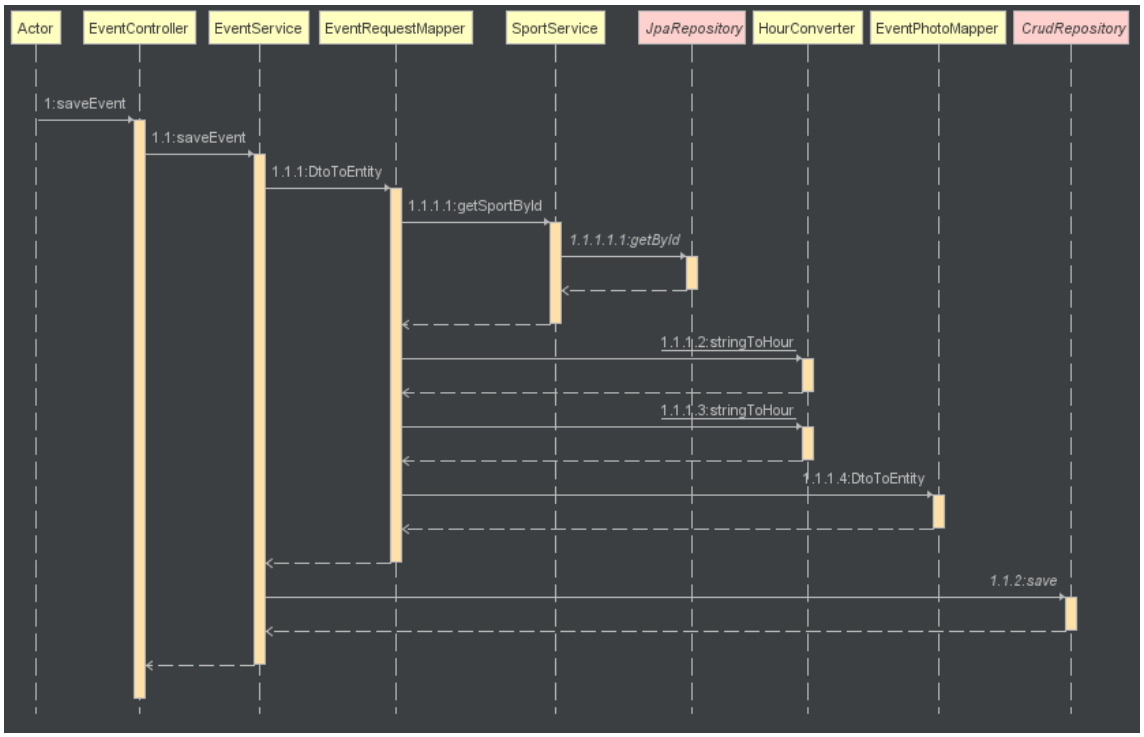

Ilustración 57 - Diagrama de secuencia - Guardar evento

Eliminar evento

Una vez pasada la fecha de un evento o si este ha sido cancelado debe ser eliminado, para ello se proporciona al consumidor este flujo que permite la eliminación de eventos de una forma sencilla.

Simplemente proporcionando el id del evento en el path de la request el evento se elimina de la base de datos sin necesidad de mapeos y otras operaciones.

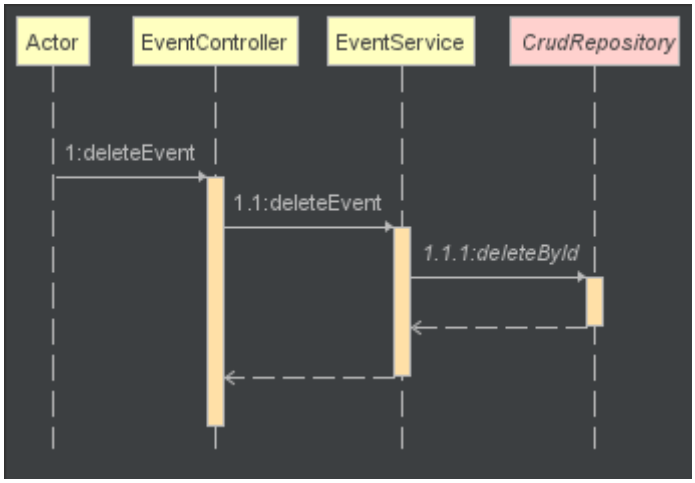

Ilustración 58 - Diagrama de secuencia - Eliminar evento

Obtener eventos mediante filtros

Para que un usuario pueda buscar eventos según sus intereses se ha creado un flujo mediante el cual el consumidor de la API puede ofrecer esta funcionalidad. Indicando el deporte que desea practicar el usuario y la posición actual del mismo este flujo devuelve todos los eventos encontrados según los criterios indicados en un radio de 30km.

Es un flujo similar al que se utiliza en la búsqueda de centros y pistas mediante filtros ya que se ha seguido la misma lógica. Se genera una llamada SQL en código java y se obtienen unos resultados que posteriormente se mapean a un objeto con información útil para el consumidor.

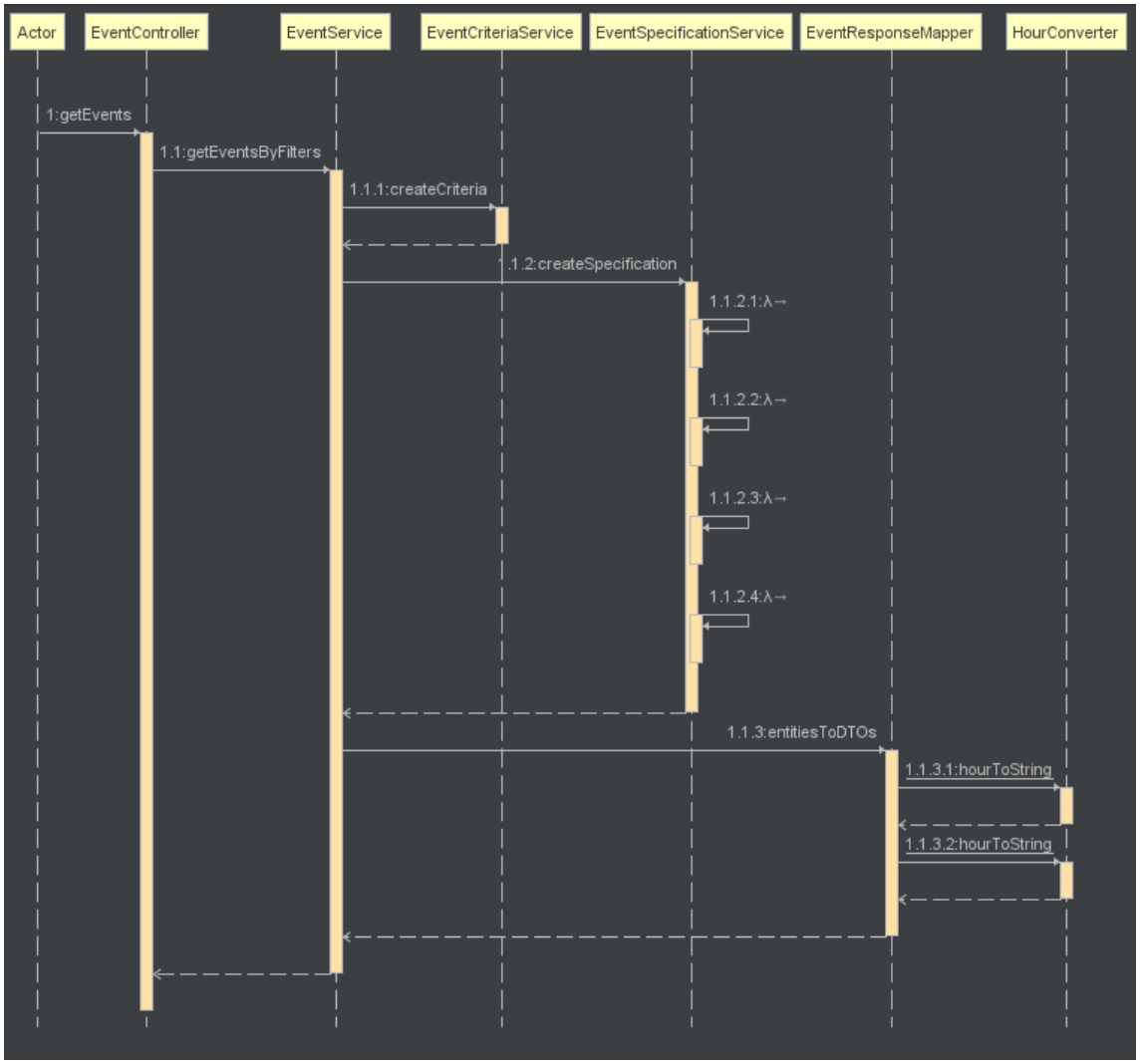

Ilustración 59 - Diagrama de flujo - Obtener eventos mediante filtros

Obtener eventos en los que está inscrito un usuario

Es importante que el usuario pueda ver en que eventos está inscrito, por ello se ofrece esta funcionalidad en la API.

Es una búsqueda sencilla, el consumidor tan solo tiene que proporcionar el id del usuario en el path de la request y gracias a la relación que existe entre los eventos y los usuarios a través de

la tabla event_user se pueden obtener todos los eventos a los que está inscrito un usuario con una simple consulta en el estándar JPA.

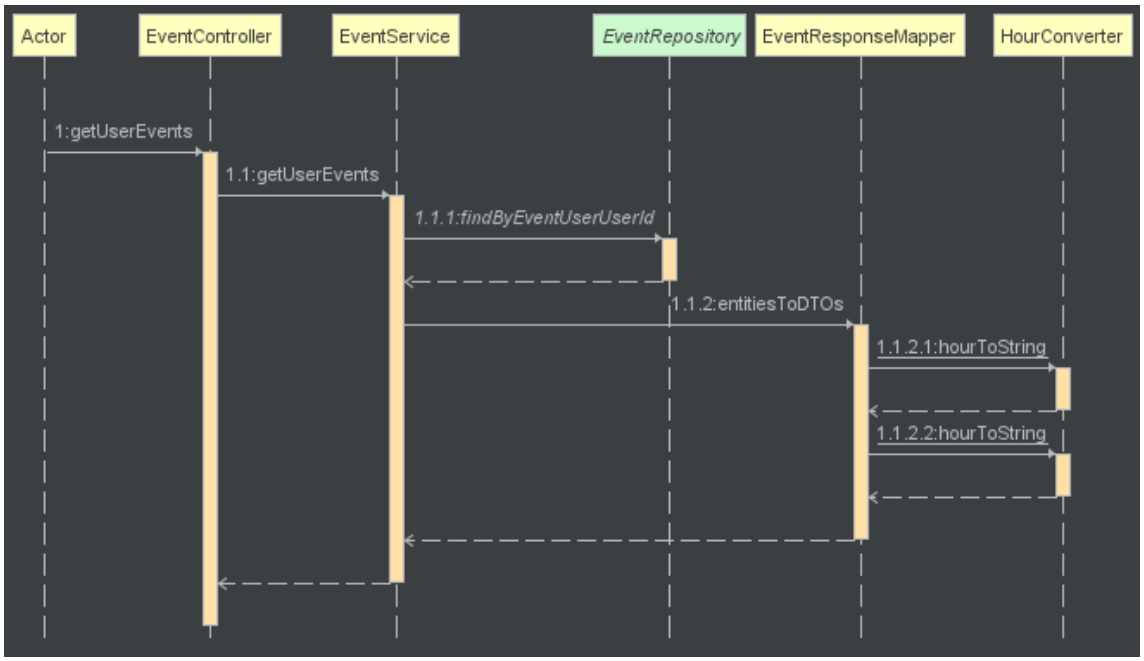

Ilustración 60 - Diagrama de secuencia – Obtener eventos en los que está inscrito un usuario

Obtener eventos publicados por un centro

También es importante que los centros sepan que eventos tienen publicados y que información ofrece cada uno de ellos, por tanto se proporciona al consumidor un endpoint para la obtención de eventos organizados por un centro en concreto.

Este flujo sigue exactamente la misma lógica que el explicado anteriormente y solo se pide al consumidor de la API que proporcione el id del centro en el path de la request.

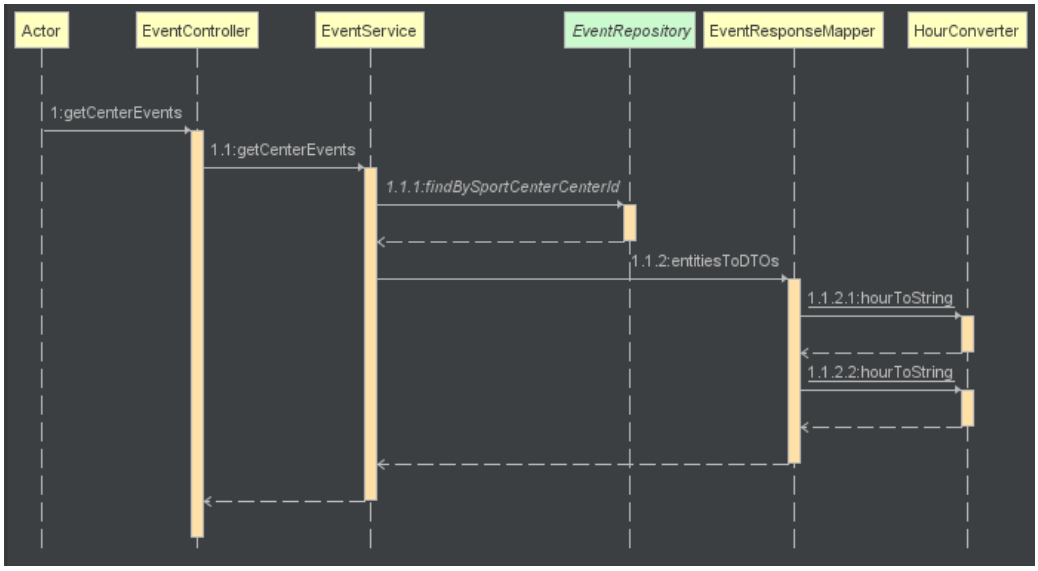

Ilustración 61 - Diagrama de secuencia - Obtener eventos publicados por un centro

6.5.4 Inscripciones

La tabla que se encarga de almacenar las inscripciones es la que relaciona los eventos con los usuarios, evento_user, es cierto que esta tabla se puede generar de forma automática como es en el caso de los usuarios y sus roles. Pero para las inscripciones se ha visto necesario generar la tabla a partir de una entidad para posteriormente poder acceder a ella con hibernate y cancelar las inscripciones eliminando la relación entre el usuario y el evento.

Tras mucho investigar no encontré otra solución viable ya que en todas ellas ocurría que para eliminar la relación debía también eliminar el evento, pero esta funcionalidad con concuerda con la lógica que debe seguir la aplicación.

Inscribirse en un evento

Al igual que los centros deben poder publicar sus eventos, los usuarios deben tener la posibilidad de inscribirse a ellos.

En este caso como se ha comentado anteriormente se ha generado una tabla asociada a una entidad por tanto para inscribirse en un evento tan solo se debe proporcionar a la API el id del evento y del usuario y estos dos datos son guardados en la tabla evento_user

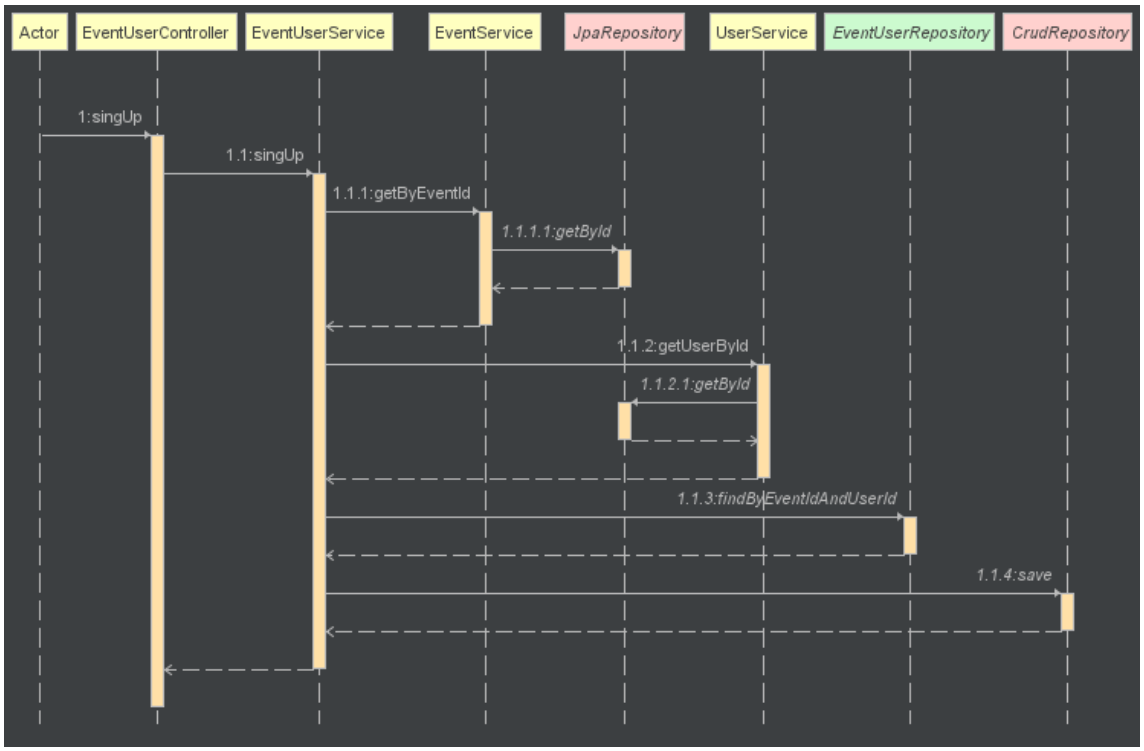

Ilustración 62 - Diagrama de secuencia - Inscribirse en un evento

Cancelar inscripción de un evento

El front debe dar la opción a los usuarios de cancelar la inscripción de un evento siempre y cuando esta acción se realice 24h antes del comienzo del evento. Por ello se implementa este flujo que permite a un usuario cancelar su inscripción.

Para ello el consumidor debe proporcionar el id del evento y del usuario en el path de la request y con dicha información se eliminará la relación de la base de datos, gracias el estándar JPA esto se puede llevar a cabo de una forma sencilla e intuitiva.

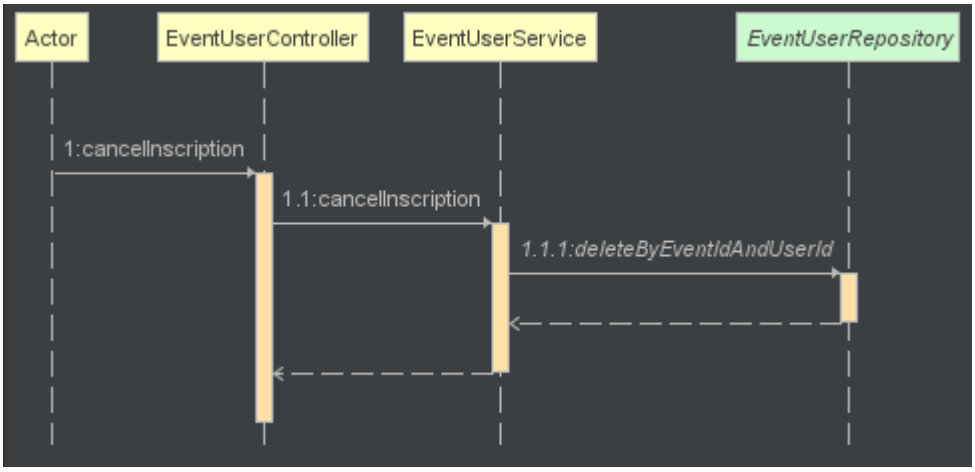

Ilustración 63 - Diagrama de secuencia - Cancelar inscripción de un evento

6.5.5 Historiales

Obtener historial de reservas

Es importante que el usuario pueda consultar un historial de reservas en la que aparezcan las reservas del último año, esto sirve tanto como justificante de la reserva ante cualquier problema con el centro o las instalaciones como registro en el que poder ver las instalaciones reservadas.

Al igual que ocurre al obtener los eventos en los que está inscrito un usuario, gracias a la relación que existe entre los usuarios y la tabla de historial con tan solo el id del usuario, proporcionado por el consumidor en el path de la request, se pueden obtener todas las reservas realizadas por dicho usuario que están almacenadas en la tabla de history.

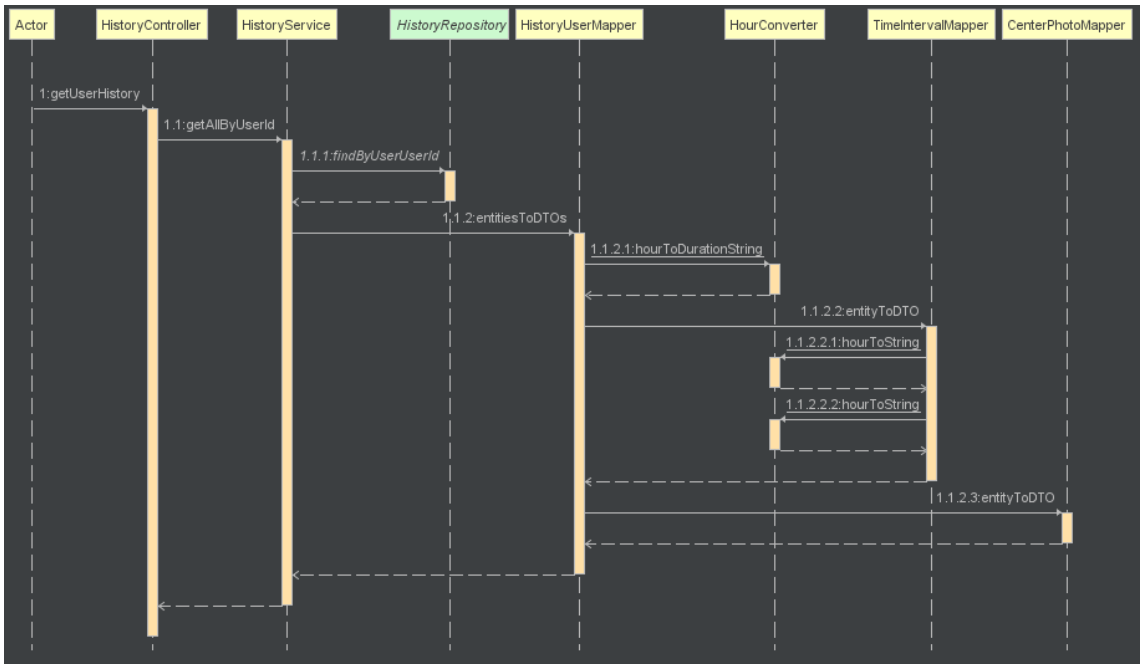

Ilustración 64 - Diagrama de secuencia – Obtener historial de reservas

6.5.6 Reservas

Reservar pista

Una de las funcionalidades principales es la reserva de pistas, ya que sin esta funcionalidad carecería de sentido la aplicación, por tanto, la API implementa un flujo, bastante complejo, que permite la reserva de pistas.

Se trata de un flujo complejo ya que se deben tener en cuenta varias cosas a la hora de reservar una pista, que el usuario exista, que la pista exista, que la pista pertenezca al centro, etc. Pero la principal es que esa pista no esté reservada a esa hora ese mismo día, en caso de que la haya se debe lanzar una excepción.

Todas estas comprobaciones se hacen simplemente por seguridad ya que en un principio el front muestra información coherente, pistas existentes, solo las horas y las pistas disponibles pero es cierto que puede haber fallos por parte de los consumidores de la API a la hora de desarrollar el front o incluso que la request enviada sea modificada, esto sería un caso de hackeo, por ello se deben hacer comprobaciones de seguridad en cada llamada.

Una vez hechas todas las comprobaciones se procede a un flujo normal de mapeo y guardado con una posterior respuesta de confirmación si no se lanza ninguna excepción.

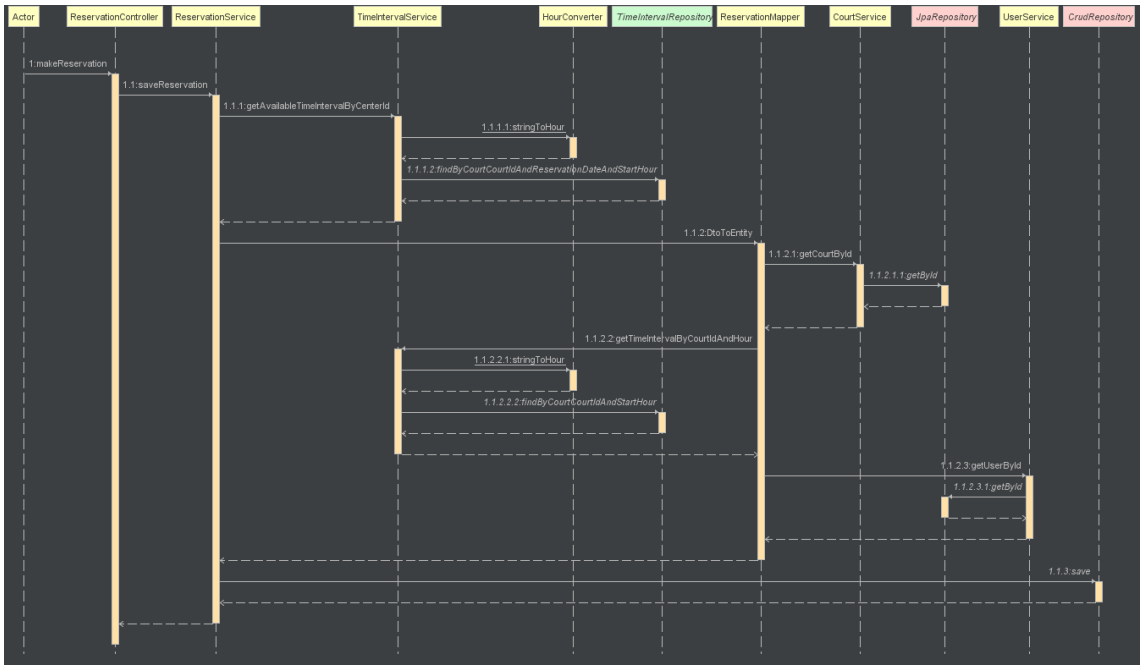

Ilustración 65 - Diagrama de secuencia - Reservar pista

Cancelar reserva

El usuario debe poder cancelar una reserva siempre y cuando queden más de 24h para su inicio, por ello en la API se ofrece un flujo que realiza esta acción.

El consumidor solo debe indicar el id de la reserva en el path de la request y tras comprobar que para dicha reserva quedan más de 24h se procede a su eliminación en la base de datos.

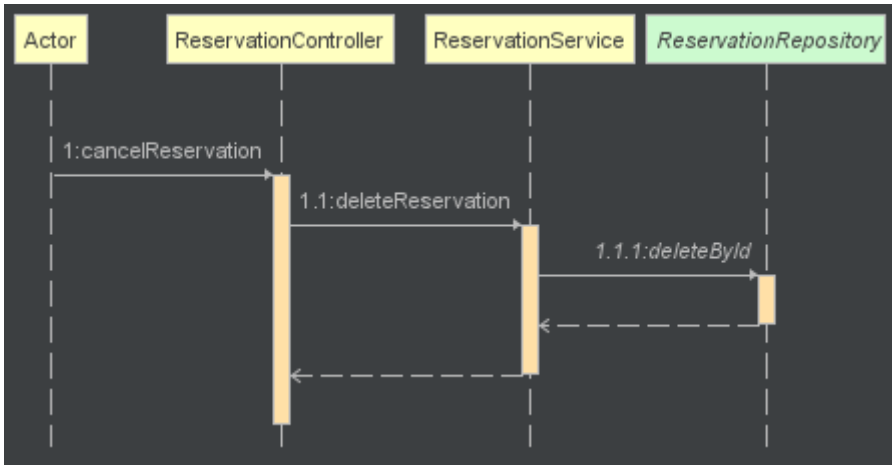

Ilustración 66 - Diagrama de secuencia - Cancelar reserva

Obtener reservas de un usuario

Se debe ofrecer al front la posibilidad de mostrar al usuario las reservas activas que tiene en un momento dado de tal forma que el usuario pueda consultar hora, pista e información adicional de la reserva.

Gracias a la relación que hay entre usuarios y reservas se pueden obtener las reservas activas de un usuario con tan solo el id del usuario que proporciona el consumidor en el path de la request. A partir de este id se puede realizar una simple búsqueda en base de datos con la ayuda del estándar JPA y con un posterior mapeo para enviar los datos al front.

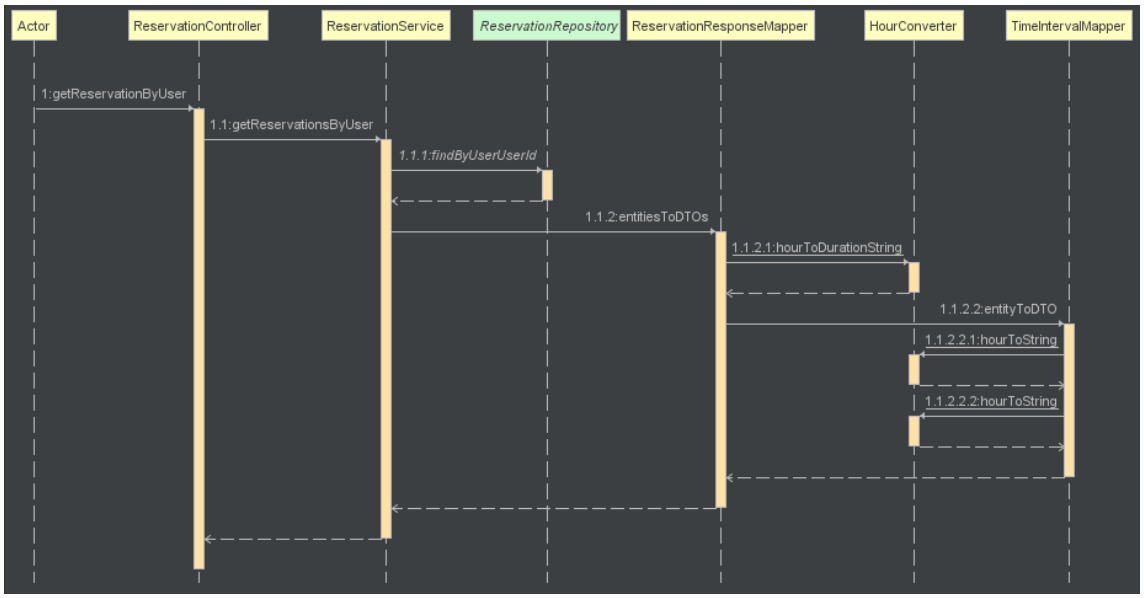

Ilustración 67 - Diagrama de secuencia - Obtener reservas de un usuario

Obtener reservas de un centro

Al igual que los usuario, es importante que los centros dispongan de un panel para visualizar las reservas de sus instalaciones, por ello la API implementa este flujo que ofrece dicha fucionalidad.

El flujo es similar al comentado anteriormente con la única diferencia que el consumidor de la API debe proporcionar en este caso el id del centro.

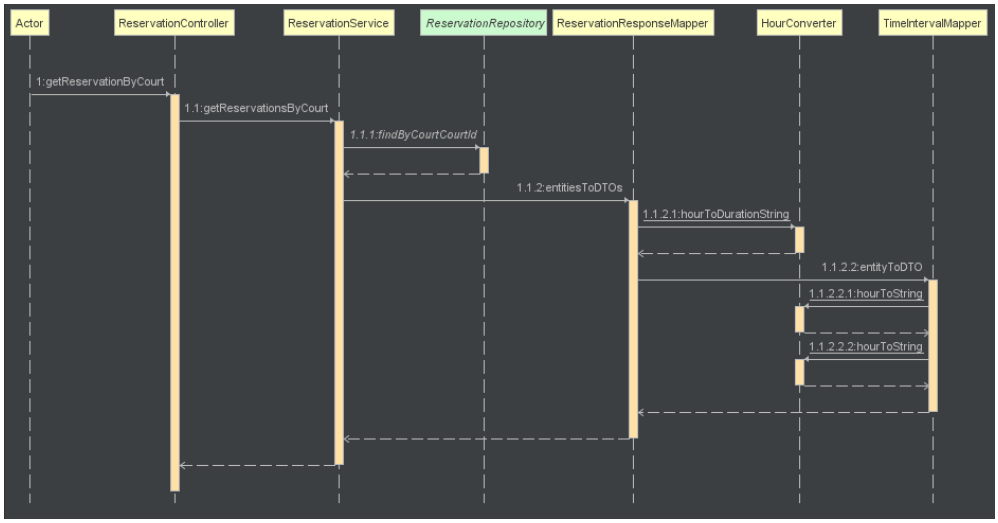

Ilustración 68 - Diagrama de secuencia - Obtener reservas de un centro

6.5.7 Deportes

Obtener nombres de deportes

En muchas ocasiones tanto los usuario como los administradores necesitan poder elegir un deporte, para que las opciones se limiten a deportes registrados en el sistema y sin errores de ortografía es interesante que el front muestre un listado con los posibles deportes a seleccionar.

Para ello se ha hecho una llamada sencilla a la base de datos que recupera todos los nombres de los deportes, pero ya que sería un caso repetitivo para obtener siempre el mismo resultado se ha decidido aplicar un patrón singleton que permita devolver una instancia creada anteriormente, ahorrando así una llamada a la base de datos.

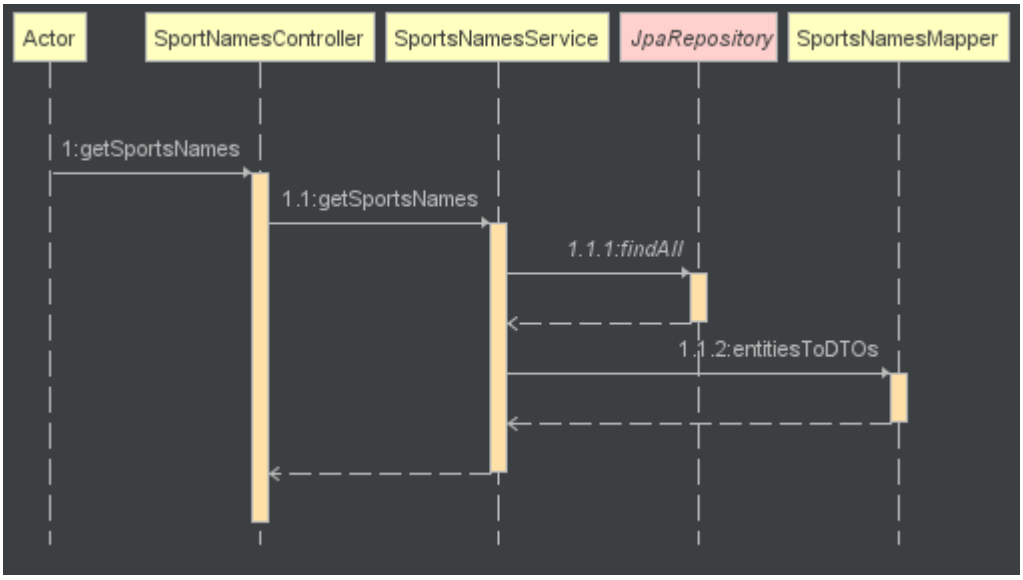

Ilustración 69 - Diagrama de secuencia - Obtener nombres de deportes

6.6 Desafíos de programación

A lo largo del desarrollo de la API Rest me he encontrado con ciertos retos, no necesariamente por que fuesen difíciles si no porque era la primera vez que plantaba dicho problema.

A continuación, se detallan los 3 desafíos más relevantes con los que se ha tenido que lidiar en este desarrollo para que la lógica de negocio se cumpliera según las especificaciones.

6.6.1 Filtros dinámicos

Desde un principio se decidió que un aspecto muy importante de Appuntate y lo cual hace que la búsqueda de un usuario sea rápida y eficaz es aplicar filtros para que el usuario pueda hacer

una búsqueda personalizada por deporte, ubicación, valoración de pistas, hora y día que se desea reservar.

Hasta ahora todo habían sido llamadas simples a la base de datos apoyadas en su gran mayoría del estándar JPA pero en este caso era un recurso que no servía.

¿Qué pasaría si al usuario solo le importa practicar un deporte en concreto, le da igual donde y cuando?
Para este caso solo se debería filtrar por deporte

¿Pero y si al usuario le importa la ubicación y el deporte y le da igual el día y la hora? En este caso se tendía que hacer un nuevo flujo que acabase en una llamada a la base de datos teniendo en cuenta sólo la ubicación y el deporte

Y esto seguiría ocurriendo con todas las posibles combinaciones, lo que es totalmente inviable ya que habría que hacer demasiados flujos para obtener una funcionalidad, poder filtrar centros según unos criterios dados.

Por lo que decidí investigar y ver la mejor forma de hacerlo, encontré una solución muy viable que permite generar una llamada SQL mediante código java y en caso de que alguno de los parámetros del filtro llegue nulo no se contempla, de esta forma se tienen en cuenta todas las opciones posibles.

Lo primero para llevar a cabo esta solución es crear con la ayuda de la librería jhipster un criterio de búsqueda. La librería jhipster nos proporciona nuevos tipos como StringFilter que nos da la opción de indicar una propiedad equals, in, notIn, etc. De esta forma por ejemplo para los deportes se ha añadido a un StringFilter la propiedad equeals con el nombre del deporte.

El caso mas complejo ha sido filtrar los centros en un radio de 15km, para ello se ha hecho uso de una fórmula para calcular una circunferencia con radio de 15km y posteriormente indicar con un IntegerFilter de la librería jhipster que la latitud de la posición actual del usuario debía ser menor o igual que la latitud mayor de la circunferencia calculada y la latitud de la posición actual del usuario mayor o igual que la latitud menor de la circunferencia calculada, para el caso de las longitudes se ha procedido de la misma manera. De esta forma obtenemos solo centros que su latitud y longitud están en la circunferencia calculada.

Una vez terminado de crear nuestro objeto criteria con se debe generar la consulta SQL. Para ello nuestro servicio debe extender de QueryService<T> y crear una especificación de la clase Specification<T>, a partir de aquí se emplea la misma lógica que en SQL, buscando por las columnas que nos interesa.

```java
@Service
public class CenterSpecificationService extends QueryService<Center> implements ISpecificationService<Center, CenterCriteria> {

    @Override
    public Specification<Center> createSpecification(CenterCriteria criteria) {

        Specification<Center> specification = (root, query, cb) -> { query.distinct(true); return null; };


        if(Objects.nonNull(criteria.getSport()))
            specification = specification.and(buildSpecification(criteria.getSport(), root -> root
                .join(Center_.sports, JoinType.LEFT)
                    .join(Sport_.sportName, JoinType.LEFT)
                        .get(SportsNames_.name)));

        if(Objects.nonNull(criteria.getLatitude()))
            specification = specification.and(buildRangeSpecification(criteria.getLatitude(), Center_.latitude));

        if(Objects.nonNull(criteria.getLongitude()))
            specification = specification.and(buildRangeSpecification(criteria.getLongitude(), Center_.longitude));

        if(Objects.nonNull(criteria.getRating()))
            specification = specification.and(buildRangeSpecification(criteria.getRating(), Center_.rating));


        return specification;
    }

}
```

Ilustración 70 - Creación de especificación para filtros dinámicos

Como se puede apreciar en la imagen 46 se hace uso reiterado de JoinType.LEFT que equivale a JOIN LEFT de SQL lo que nos permite buscar en otras tablas solo si la relación entre ambas no es nula.

Una vez se ha terminado de crear la especificación se llama a la base de datos con una llamada findAll del estándar JPA, al obtenerse los resultados se mapean y se envían a front.

6.6.2 Cron

Un cron es una parte de código del programa que se ejecuta automáticamente en un periodo de tiempo según se indique, se puede ejecutar cada unos cuantos segundos, minutos, horas, días, etc.

Se ha visto necesario la implementación de un cron tanto para poder eliminar automáticamente las reservas caducadas y pasarlas al historial de reservas como para eliminar el historial cada año.

Cada 5 minutos se comprueban las reservas activas y si se encuentra alguna en la que la fecha de la reserva es igual a la fecha de hoy y la hora finalización de la reserva es anterior a la hora actual esta reserva se elimina y pasa al historial.

De esta forma mantenemos en la tabla de reservas solo las reservas activas teniendo solo registros relevantes y reduciendo la carga de la tabla de reservas en base de datos.

6.6.3 Token JWT

JSON Web Token o JWT es un estándar abierto que se utiliza comúnmente en el desarrollo web para el intercambio de información segura entre las partes que componen un software. Los

tokens tienen información útil codificada en un objeto JSON, se firman digitalmente con clave pública o privada y se encriptan según un estándar elegido.

Los tokens constan de 3 partes fundamentales:

- **Header:** El encabezado contiene la lista de operaciones de criptografía que se aplican al token para encriptarlo.
- **Payload:** Contiene los datos útiles del JWT, datos reales que se transfieren a través del token.
- **Signature:** La firma se utiliza para verificar que el token no ha sido modificado y por tanto se trata de un token auténtico y no está siendo hackeado.

La función de el JWT en este proyecto es autorizador y validador, ya que será el encargado de comprobar si el usuario que está realizando una acción tiene autorización para llevarla a cabo y a su vez valida que el usuario exista y esté con la sesión iniciada.

Un token se genera cada vez que un usuario se registra o inicia sesión en la aplicación, para este proyecto se ha decidido crear un token sencillo que contiene la información esencial para su correcto funcionamiento, rol, nombre de usuario, fecha de creación del token y fecha de vencimiento. Para el encriptado se ha utilizado el algoritmo HS512 que se proporciona en la librería jsonwebtoken.

Este token se genera y se devuelve en el cuerpo de la respuesta cada vez que un usuario se registra o inicia sesión, de esta forma el consumidor dispone de este token que debe añadir en el header de todas las llamadas a la API, de esta forma se puede validar la existencia del usuario y que este ha iniciado sesión, así como si dicho usuario tiene autorización para llevar a cabo la acción realizada.

Se ha decidido poner 1 día de expiración para el token, de esta forma cada vez que pasen 24 desde el ultimo inicio de sesión se deberá volver a iniciar sesión para que se genere un nuevo token válido.

7. Control de versiones y despliegue en AWS

En todo proyecto de desarrollo de software es esencial tener un control de versiones ordenado y que permita el continuo desarrollo de código sin perder parte de este y que permita trabajar en paralelo a varios programadores.

Hay muchas herramientas que nos permiten realizar esta tarea, pero para este proyecto se ha decidió utilizar github ya que contaba con experiencia previa con esta plataforma.

7.1 GitHub

GitHub es un social coding que permite subir código y almacenarlo en repositorios utilizando el sistema de control de versiones de Git. Es gratuito y fácil de usar ya que cuenta con una gran compatibilidad con muchos entornos de desarrollo, en este caso existe un plugin para Visual Studio Code, y también cuenta con una aplicación de escritorio que facilita realizar todas las operaciones con una interfaz gráfica sin la necesidad de usar el terminal de comandos.

Es imprescindible tener una buena organización del repositorio, un buen control de las ramas y las distintas versiones para evitar que se pierda código o que se encuentre en producción un código que no funciona. Para ello es importante respetar el flujo de Git (Git Flow) y revisar correctamente el código que se va a subir al repositorio remoto.

En este proyecto se ha decidido tener 3 ramas principales a partir de las cuales salen las distintas ramas en las que se desarrolla código.

- **develop:** Es una rama de desarrollo a la cual se añade el código nuevo.
- **release:** Es una rama que contiene código en estado de prueba.
- **master:** Esta rama es la principal y siempre debe tener una versión estable del código.

Todo el código que se encuentre en entornos superiores debe estar también en entornos inferiores por tanto, develop siempre debe tener como mínimo el código que hay en reléase y master y en reléase siempre debe haber como mínimo el código que se encuentra en master. A partir de estas tres ramas principales nacen nuevas ramas según la intención y el origen.

- **feature:** Estas ramas siempre se crean a partir de la última versión de la rama develop y en este tipo de ramas es donde se realizarán los nuevos desarrollos.
- **bugfix:** Este tipo de ramas se generan a partir de la rama reléase y son para solucionar bugs detectados en el periodo de pruebas.
- **hotfix:** Las ramas hotfix son de máxima prioridad ya que se crean a partir de la rama master y se utilizan para arreglar bugs que ya están en producción, por tanto deben ser arreglados y mergeados lo antes posible, para tener cuanto antes un código estable en la rama master.

Una vez expuesto la metodología de trabajo mencionando las ramas principales y los distintos tipos de ramas que se han utilizado para el desarrollo voy a poner unos ejemplos para aclarar cualquier duda que pueda surgir.

- **Nuevo desarrollo:** Si deseo hacer un nuevo desarrollo, implementando un nuevo endpoint y de esta forma ampliar la lógica de negocio, debo crear una rama feature a partir de la ultima versión de la rama develop y desde esta rama feature realizar el desarrollo para posteriormente juntarlo con la rama develop.
- **Corregir un desarrollo en pruebas:** En caso de que se detecte algún error de la funcionalidad ya se por parte del consumidor o por mí, la forma correcta de proceder sería crear una nueva rama del tipo bugfix a partir de la versión más actualizada de la rama reléase, realizar los cambios oportunos para corregir el error en la rama bugfix y posteriormente actualizar la rama reléase y propagar los cambios a la rama develop.

- **Corregir un desarrollo en la rama master:** En este caso solo se sube un código a master cuando este ha sido probado y se va a hacer una demostración ante un centro. En caso de que se encuentre algún fallo ya sea antes o después de la demo se debe crear una rama hotfix a partir de la rama master y resolverse la incidencia en la rama hotfix con total prioridad y posteriormente juntarla con la rama master y propagar los cambios a las ramas reléase y develop.

7.2 Despliegue en AWS

Amazon Web Service (AWS) es una plataforma en la nube que nos ofrece amazon a los desarrolladores y que nos brinda de grandes oportunidades en el ámbito devops y de forma gratuita.

AWS dispone de múltiples servicios al alcance de los desarrolladores de forma gratuita. Son muchos los servicios que esta plataforma ofrece, pero para este proyecto solo han sido necesarios 2 de ellos.

Se ha hecho uso de AWS y los servicios que comento a continuación ya que tenía experiencia previa y es una forma sencilla y gratuita de disponer de la API de forma remota.

7.2.1 RDS

Servicio de bases de datos relacionales administrado, este es el resumen que nos ofrece amazon a los desarrolladores y el cual es por si mismo explicativo. RDS es un servicio que nos permite administrar nuestras bases de datos de forma remota pudiendo acceder desde cualquier parte.

Amazon nos ofrece una capa gratuita para este servicio, es cierto que es una capa muy limitada con una base de datos pequeña y lenta, pero son requisitos suficientes para este proyecto.

De esta forma se ha decidido almacenar la base de datos PostgreSQL en el servicio RDS de AWS haciendo que los datos almacenados no sean solo accesibles desde un ámbito local si no que se pueden acceder desde cualquier punto con internet permitiendo a mi compañero del front realizar pruebas con el móvil con datos reales.

7.2.2 Elastic BeanStalk

Este servicio de AWS nos permite crear entornos de nuestras aplicaciones como si de un contenedor Docker se tratara. De esta forma podemos tener la API levantada en remoto y se pueden hacer peticiones desde cualquier punto con internet.

Para el despliegue de esta API se ha creado un entorno Linux con Java 11 para mantener la aplicación en continuo funcionamiento. Al igual que cuando se lanza un proyecto en local este

servicio nos ofrece la posibilidad de ver los logs con la ayuda del servicio CloudWatch Logs pudiendo ver las trazas de los posibles errores que se produzcan en la aplicación desplegada.

8. Conclusiones

Appuntate es una idea de negocio que se ha desarrollado en conjunto a otro compañero y que intenta hacerse un hueco en un sector en auge a pesar de la situación de pandemia vivida. Esta propuesta de negocio no es solo una ilusión de dos alumnos de ultimo año de carrera si no que es una fuerte apuesta a un proyecto que creemos que tiene cabida en el mercado y que a la mayoría de los centros con los que hemos tratado le ha parecido una idea competente frente al mercado actual.

La aplicación web de Appuntate que es pieza fundamental de esta idea de negocio ha sido desarrollada con tecnología muy utilizada en el mercado del desarrollo de software a nivel profesional y empresarial.

En este trabajo se refleja el desarrollo de la API que da vida en forma de código a esta idea de negocio pudiendo hacer posible que la lógica de la aplicación funcione correctamente y según los requisitos definidos.

Para el desarrollo de la API se ha utilizado un modelo REST que ha sido realizado con relativa facilidad gracias a la configuración y ayuda ofrecida por Spring Boot, framework usado.

Pese a tener experiencia previa en programación de API REST con Spring Boot este proyecto me ha llevado a conocer y explorar rincones desconocidos de la programación y arquitectura software, sirviéndome de gran ayuda para aprender un poco más el uso de este framework, sus librerías y su compatibilidad con el estándar JPA e hibernate. Haciendo posible que poco a poco me acerque cada vez mas a lo que sería todo un sueño, ser un buen profesional.

Es cierto que ha sido una idea tediosa de llevar a cabo ya que a la vez que lidiaba con un la documentación y desarrollo del trabajo debía cumplir mi jornada laboral a tiempo completo. Sin embargo, estoy muy satisfecho con el resultado obtenido, es cierto que actualmente la aplicación desarrollada es un MVP, una versión muy pobre de lo que puede llegar a ser esta idea de negocio, pero creo que muestra el potencial de lo que puede ser un producto final de la aplicación.

9. Trabajo futuro

Aún queda mucho camino por delante, mucha funcionalidad que implementar y aspectos que mejorar, pero gracias a los experimentos y reuniones realizadas con ayuntamientos se han abierto bastantes puertas para Appuntate.

Por una parte, hemos aprendido que les gustaría tener a los ayuntamientos a la hora de gestionar sus instalaciones deportivas, lo que nos deja mucho trabajo por delante ya que estas entidades son muy específicas con sus requisitos.

Algunos aspectos para tener en cuenta para desarrollos futuros serían:

- Posibilidad de navegar por la aplicación sin iniciar sesión
- Generación automática de cuadros de enfrentamientos en los torneos y eventos.
- Ofrecer una generación de eventos dinámica, ya que cada centro y cada evento requerirá una información distinta.
- Gestión de las clases impartidas en el centro.
- Gestión de los usuarios por parte del centro.
- Incluir un mayor aspecto social en la aplicación con comentarios, valoraciones y reservas en grupo.
- Ofrecer una herramienta que ayude a organizar los horarios de partidos oficiales que se disputen en las instalaciones.
- Ofrecer descuentos según la situación familiar y personal.

Por otra parte, algunos ayuntamientos buscan un software a medida lo que no da paso a Appuntate, pero en estas reuniones los informáticos encargados de las instalaciones deportivas han mostrado interés en nuestra metodología de trabajo y resultados obtenidos abriendo una nueva puerta y es poder hacer un software a medida contando con un equipo de desarrollo más grande.

10. Anexos

Objetivos de desarrollo sostenible relacionados con el TFG

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

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

- **Salud y Bienestar:** Considero que la salud y el bienestar no solo está asociado a aspectos médicos y medicinales, si no que muy lejos de esto un aspecto imprescindible para la salud y el bienestar es la práctica de deporte, y uno de los objetivos principales de este trabajo es promocionar la práctica del deporte.

- **Igualdad de género:** Uno de los aspectos fundamentales en la sociedad de hoy en día es la aceptación del deporte femenino, que aun que se está luchando por conseguir la igualdad y el reconocimiento del mismo la realidad es que aún está muy lejos de ser así. Con la visibilidad del deporte a nivel amateur y facilitando que este se practique se pretende motivar a chicas jóvenes a propulsar sus sueños en el mundo del deporte.

- **Reducción de desigualdades:** Como se ha comentado en el objetivo anterior uno de los objetivos de este proyecto es acabar con la gran desigualdad que viven las mujeres en el mundo del deporte.

- **Trabajo docente y crecimiento económico:** A su vez, también creo que este proyecto promueve el crecimiento económico tanto de las ciudades y pueblos que adquieran este sistema para los centros municipales como de centros privados que busquen una forma de promocionarse y llegar a más gente.

11. Referencias

[1] Responsive: Diseño web adaptativo, mejor que adaptable o responsivo, es la traducción más ajustada en español de la expresión inglesa responsive web design. https://www.fundeu.es/dudas/palabra-clave/responsive/

[2] Visual Studio Code: https://code.visualstudio.com/

[3] Debugger for Java: https://marketplace.visualstudio.com/items?itemName=vscjava.vscode-java-debug

[4] Vim: https://www.vim.org/

[5] pgAdmin: https://www.pgadmin.org/

[6] Postman: https://www.postman.com/

[7] Spring Boot: https://spring.io/projects/spring-boot

[8] Spring Boot: https://blog.codmind.com/que-es-spring-boot/#:~:text=Spring%20Boot%20es%20una%20tecnolog%C3%ADa,servidor%20de%20aplicaciones%20y%20enfocarnos

[9] Inyección de dependencias Spring: https://www.adictosaltrabajo.com/2013/07/25/spring-container-inyeccion-dependencias/

[10] Maven: https://maven.apache.org/

[11] Maven: https://www.javiergarzas.com/2014/06/maven-en-10-min.html

[12] Hibernate: https://hibernate.org/orm/

[13] HQL: Hibernate Query Languaje https://docs.jboss.org/hibernate/orm/3.5/reference/es-ES/html/queryhql.html

[14] JPA: https://www.campusmvp.es/recursos/post/la-api-de-persistencia-de-java-que-es-jpa-jpa-vs-hibernate-vs-eclipselink-vs-spring-jpa.aspx

[15] SQL: https://datademia.es/blog/que-es-sql

[16] PostgreSQL: https://www.postgresql.org/

[17] PostgreSQL: https://es.wikipedia.org/wiki/PostgreSQL

[18] API: https://datos.gob.es/sites/default/files/doc/file/guia_publicacion_apis.pdf

[19] REST: https://docs.microsoft.com/es-es/azure/architecture/best-practices/api-design

[20] Clean code: libro que redacta varios estándares para mantener el código limpio, mantenible y reutilizable.

[21] Patrón fábrica: https://refactoring.guru/es/design-patterns/factory-method

[22] Patrón singleton: https://refactoring.guru/es/design-patterns/singleton

[23] Cursos: Todos los cursos los he realizado en la plataforma Pluralsight: https://www.pluralsight.com/
Design Pattern in Java: Creational: https://www.pluralsight.com/courses/design-patterns-java-creational

Design Pattern in Java: Structural: https://www.pluralsight.com/courses/design-patterns-java-structural
Design Pattern in Java: Behavioral: https://www.pluralsight.com/courses/design-patterns-java-behavioral
Java: Refactoring to Design Patterns: https://www.pluralsight.com/courses/java-refactoring-design-patterns
Java: Writing Readable and Maintainable Code: https://www.pluralsight.com/courses/java-writing-readable-maintainable-code
SOLID SoftwareDesign Principles in Java 8: https://www.pluralsight.com/courses/solid-software-design-principles-java

[24] https://www.baeldung.com/java-dto-pattern

[25] Cloudinary: https://cloudinary.com/