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 front-end.

Trabajo Fin de Grado

Grado en Ingeniería Informática

AUTOR/A: Gordo Gil, Rubén

Tutor/a: Pelechano Ferragud, Vicente

CURSO ACADÉMICO: 2021/2022

Agradecimientos

A mis compañeros de clase, los que se han convertido en mis amigos y han sido una parte fundamental de mi vida durante estos años de carrera, han servido de inspiración, de apoyo, y me han ayudado a llegar a ser quien soy hoy. Ellos, con los que me he divertido, he atravesado momentos difíciles y de los que he aprendido. En especial a mis compañeros Javier, Julio, Daniel y por supuesto, Tomás, con el que he tenido el placer de realizar este trabajo.

A mis profesores, que me han guiado durante esta etapa académica y me han dado a conocer un mundo tan apasionante como es del desarrollo de software. Gracias a ellos he descubierto mi verdadera vocación.

A mi familia, por estar siempre ahí, por darme un apoyo incondicional, por saber guiarme hacia el camino correcto para poder alcanzar mi máximo potencial. No podría haber deseado una familia mejor que la que tengo.

Resumen

En una época post-pandemica el conjunto de la población se ha acostumbrado a realizar hasta las tareas más cotidianas como hacer la compra, a través de internet y la reserva de pistas en centros deportivos no es una excepción. Aunque existen herramientas que permiten realizar estas reservas a través de internet, existe un claro margen de mejora. Este trabajo detalla el desarrollo de la idea de negocio de una solución única para la reserva y gestión de instalaciones deportivas además del proceso de implementación de esta solución en forma de aplicación hibrida, utilizando las tecnologías de desarrollo web utilizadas por las grandes empresas en la actualidad.

Palabras clave: Aplicación híbrida, emprendimiento, Angular, Ionic, testing, deporte, móvil, gestión

Abstract

In a post-pandemic era, the population has become accustomed to carrying out even the most mundane daily tasks, such as shopping, online. Making a reservation for a court in a sport center is no exception. Although there are tools already in use that allow these reservations to be made online, there is a lot of room for improvement. This work details the development of a business idea that introduces a unique solution that includes a sport center management system that also allows clients to make court reservations, it also illustrates the implementation process of this solution in the form of a hybrid application, using the latest web development technologies used by large companies.

Keywords: hybrid application, entrepreneurship, Angular, Ionic, testing, sport, mobile, management

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 cambiando muchas de las tareas cotidianas del día a día que se realizaban de manera presencial a un modelo remoto. El cambio de hábitos mencionado es fácilmente apreciable en situaciones como hacer la compra online en vez de ir al supermercado, pedir comida a domicilio en ocasiones en las que hubiésemos ido al restaurante, reuniones con amigos y familia por videollamada y hasta en el trabajo con un modelo remoto.

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 de 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.

En este apartado se exponen 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 exponen los objetivos que Appuntate tiene que cumplir como producto.

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

En este apartado se exponen los objetivos relacionados con el desarrollo de la apliación.

Tecnología
- Desarrollar una aplicación híbrida capaz de ejecutarse en múltiples plataformas
- Establecer estrategias de testing que garanticen la calidad del código
- Seguir las buenas prácticas establecidas en **Angular**

1.3 Estructura de la memoria

El trabajo es un trabajo realizado en conjunto con Tomás Montalvo Tercero. La idea de negocio ha sido desarrollada de manera colaborativa y el apartado técnico ha sido desarrollado de manera individual. El apartado individual incluye todos los capítulos desde el 5, estando incluido, en adelante.

El trabajo en su parte conjunta presenta la evaluación y puesta en marcha de una idea de negocio seguida por la parte individual donde se detalla todo el desarrollo técnico de la aplicación.

La memoria 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:** Introducir el concepto de aplicación híbrida y establecer las tecnologías y el entorno de desarrollo utilizado
- **Capítulo 6:** Exponer la estrategia establecida para el control de versiones
- **Capítulo 7:** Detallar la estructura y la arquitectura utilizada para el desarrollo
- **Capítulo 8:** Detallar el uso de los componentes que proporciona **Angular** y su implementación en Appuntate

- **Capítulo 9:** Ilustrar los métodos utilizados para establecer la conexión con el back-end
- **Capítulo 10:** Comentar la estrategia seguida para el diseño de interfaces de usuario y exponer los prototipos de las interfaces de usuario
- **Capítulo 11:** Exponer la importancia des testing y las tecnologías utilizadas
- **Capítulo 12:** Detallar las estrategias de implantación y despliegue
- **Capítulo 13:** Establecer los trabajos a realizar en las siguientes fases de desarrollo
- **Capítulo 14:** Se presentan las conclusiones y la relación con los estudios cursados
- **Capítulo 15**: Se exponen las Referencias bibliográficas

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 las 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. El estudio de mercado realizado 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 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 [1].

Playtomic es una plataforma dedicada a la reserva y gestión de pistas que ofrece tanto una aplicación móvil como una página web desde las cuales los usuarios pueden realizar distintas operaciones. La plataforma 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, se llama "Playtomic manager" y aunque no hemos podido probarlo de primera mano debido a que no está disponible de manera abierta, 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 a seguir en cuanto al desarrollo de Appuntate, por esta razón, Playtomic es la aplicación a la que más tiempo de estudio le hemos dedicado.

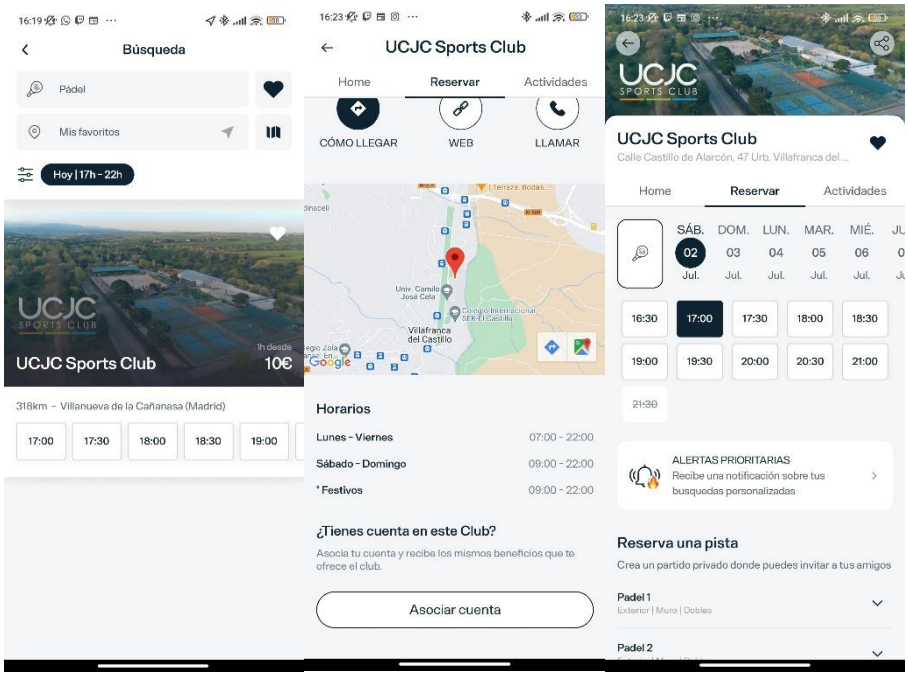

Ilustración 1 - Aplicación Playtomic

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 [2]. Este sistema de gestión y reservas se centra en el sector público, todos sus clientes son ayuntamientos y centros municipales.

Tras un estudio de la aplicación, haberla probado, y haber preguntado a algunos usuarios habituales además de a un administrador de un centro con dicha aplicación, mencionado 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.

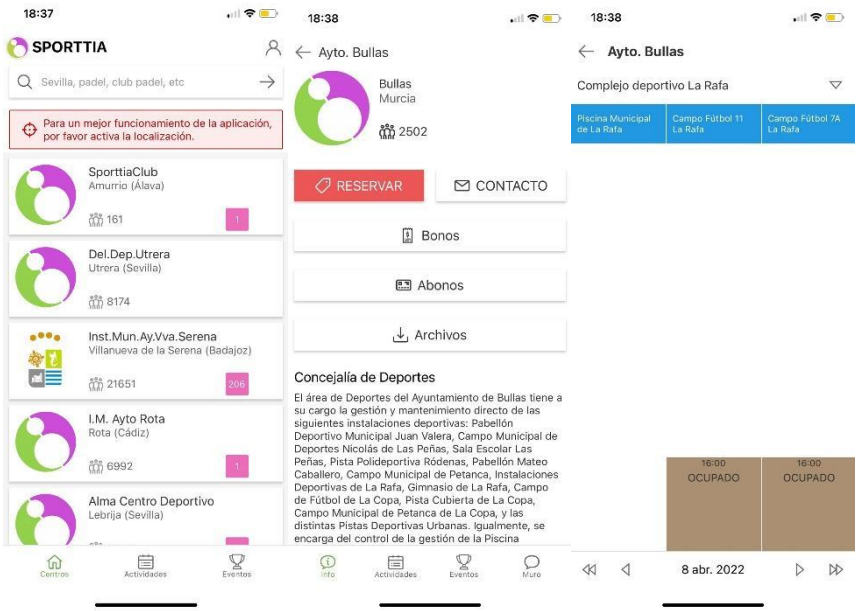

Ilustración 2 - Aplicación Sporttia

Como se puede observar en la ilustración 2, el diseño de la interfaz de usuario es poco intuitivo y anticuado. La aplicación 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 y, 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. La interfaz tampoco se adapta correctamente a todos los tamaños de pantalla, no sigue un diseño “responsive”. 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 la Google Play Store y Apple App store, 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 [3]. Se trata de una aplicación muy completa y visualmente moderna que pretende facilitar la gestión a los centros deportivos.

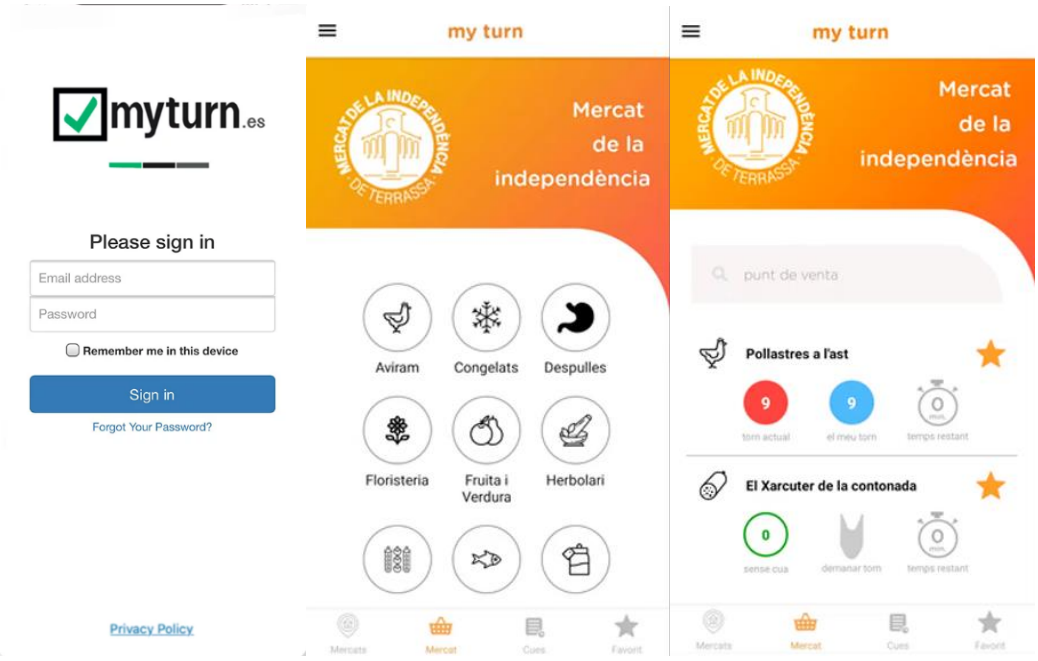

Ilustración 3 - Aplicación MyTurn

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

Clicac es 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 [4].

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.

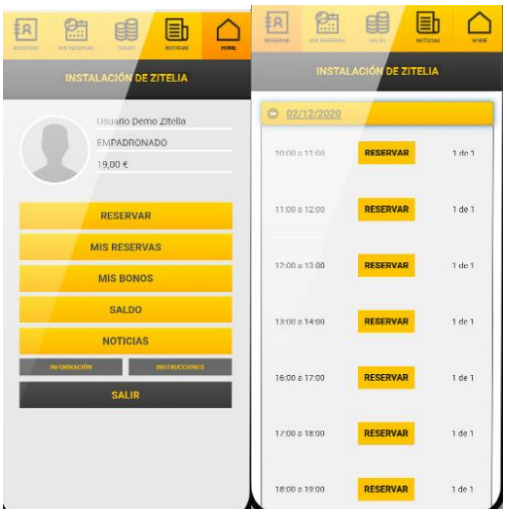

Ilustración 4 - Aplicación Clicac

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 [5].

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. Los módulos disponibles son los siguientes:

1. Abonados
2. Escuelas y actividades
3. Reservas y alquiler de espacios
4. Taquilla
5. Terminales de venta automática
6. Servicios web
7. Control de accesos
8. Control de presencia
9. Competiciones
10. Administración y control de caja
11. App hecha a medida para cada centro deportivo
12. Mantenimiento de instalaciones
13. Padrón
14. Servicios médicos
15. Procesos automáticos
16. Parking

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., por esta razón, nos hemos centrado solamente en los siguientes módulos y sus funcionalidades principales:

1. Abonados
    a. Funcionalidades:
        i. Base de datos de abonados
        ii. Tratamiento de beneficiarios o familiares
        iii. Gestión de cuotas y formas de pago
        iv. Emisión de recibos
2. Escuelas y actividades
    a. Funcionalidades:
        i. Base de datos de usuarios
        ii. Emisión de carné de usuario
        iii. Gestión de bonos
3. Reservas y alquiler de espacios
    a. Funcionalidades:
        i. Gestión de las reservas de las instalaciones (Creación, modificación)
4. Servicios web
    a. Funcionalidades:
        i. Aplicación con funcionalidad para el usuario de las instalaciones
            1. Consultar información de la entidad
            2. Descarga de documentos
            3. Realización de reservas y alquileres
            4. Solicitud de plazas
            5. Darse de alta
5. Competiciones
    a. Funcionalidades:
        i. Gestión de ligas a una o dos vueltas
        ii. Ranking
        iii. Gestión de pruebas de “Cross” y natación
6. Administración y control de caja
    a. Funcionalidades:
        i. Gestión de compras y pedidos
        ii. Control de almacén
        iii. Control de caja
        iv. Control de gestores
        v. Control de terminales de auto venta

Como podemos ver, Omesa ofrece toda la funcionalidad que ofrece Appuntate, sin embargo, su integración modular lo convierte un sistema algo intrincado y 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 (administración)	✓	✓	✓	✓	✓	✓

Aplicación móvil (Usuarios & Administración)	✓	✓	✓	×	✓	✓
Modificación de reservas	×	×	×	×	×	✓
Cancelación de reservas [1]	×	×	×	×	×	✓

Tabla 1 - Comparativa de características frente aplicaciones competidoras

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 [6].

[1] La característica de cancelación de reservas hace referencia a que 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.

Ilustración 5 - Modelo SaaS

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 lo mencionado 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 6 - Beneficios trimestrales al cabo de 5 años

En la ilustración 6 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 7 - Gastos trimestrales al cabo de 5 años

Como podemos observar en la ilustración 7 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 8 - 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 9 - Beneficio de inversores

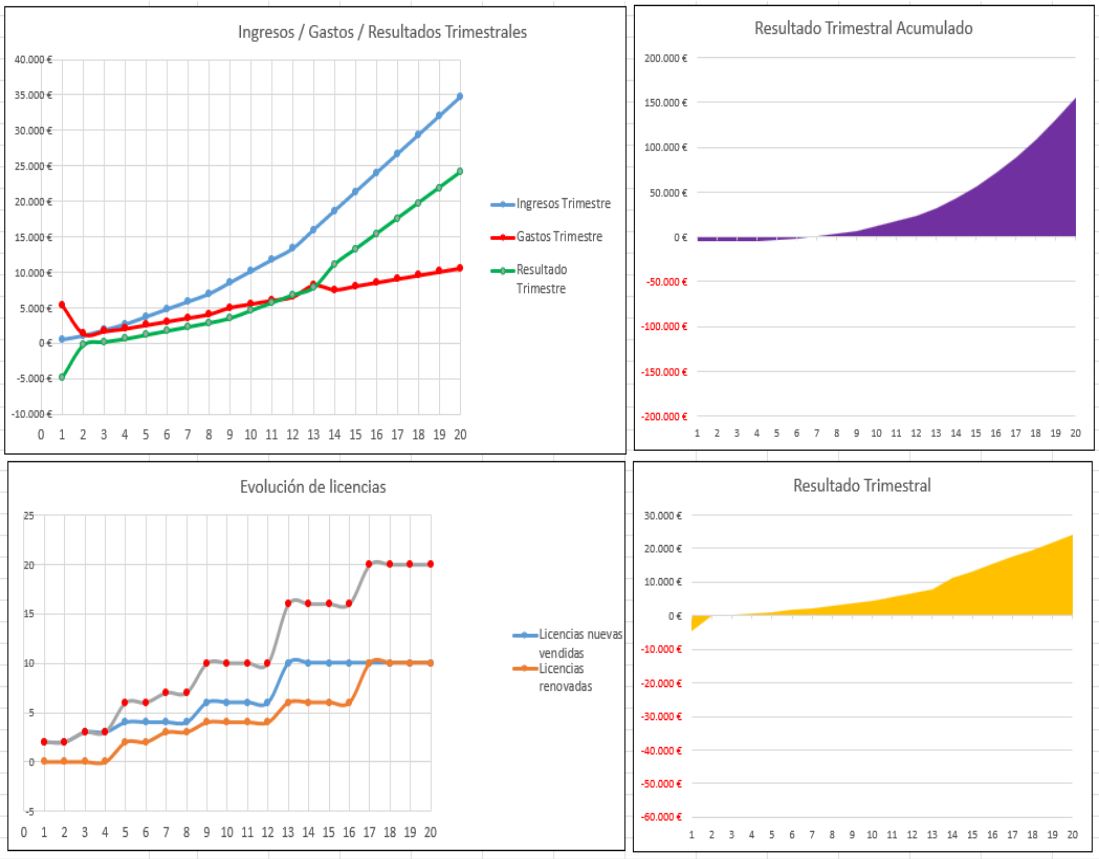

Ilustración 10 - Gráficas de gastos y beneficios

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 11 - 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, a pesar de existir competencia, nos centramos en un sector concreto del mercado que está en auge y utilizamos tecnologías moderas además de ofrecer 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 [7].

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 esta.

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.

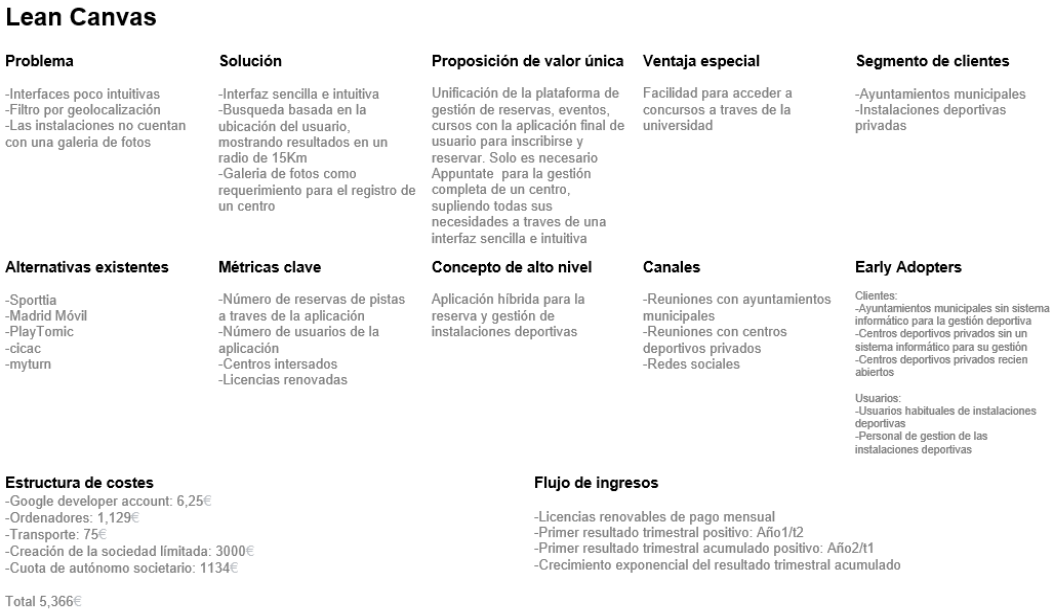

Ilustración 12 - Lean Canvas

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 de 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, permitiendo 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. Los filtros pueden hacer uso de la localización del usuario para mostrar los centros más cercanos, en un radio de 15km y 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 una 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. Los usuarios 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. En el modelo se muestran 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.

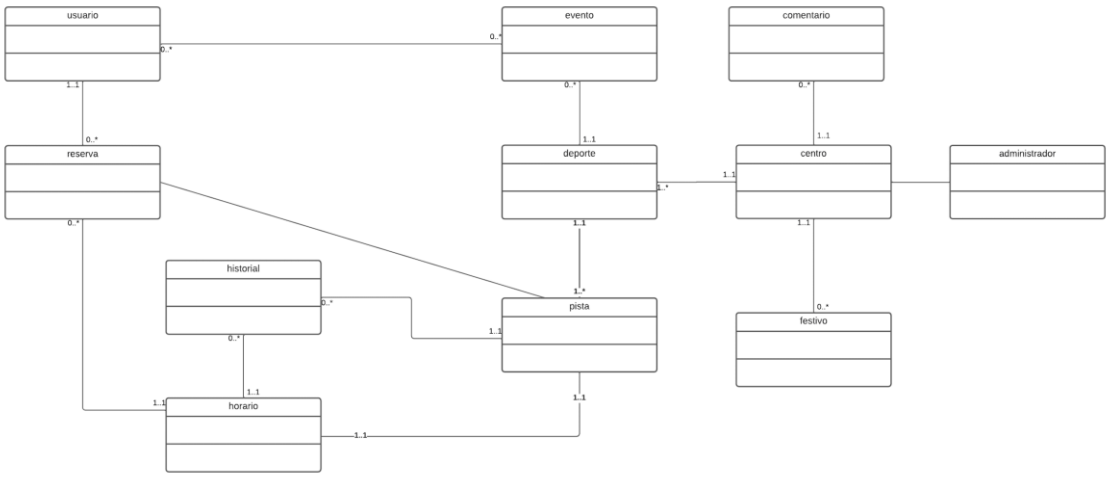

Ilustración 13 - Modelo de dominio

Como se muestra en la ilustración 13 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.

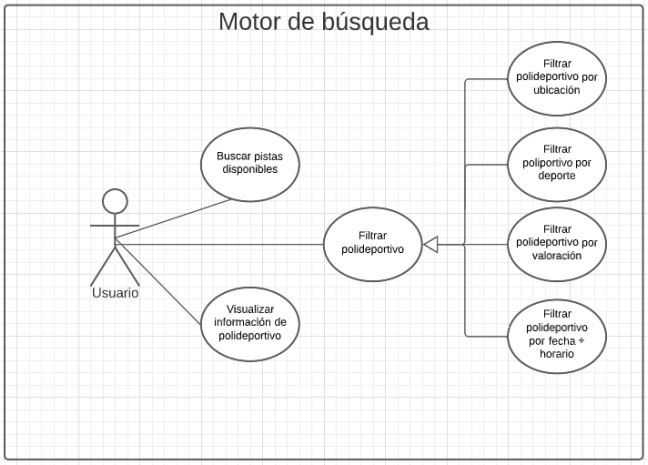

Ilustración 14 - Casos de uso para la característica: Motor de búsqueda

- **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.

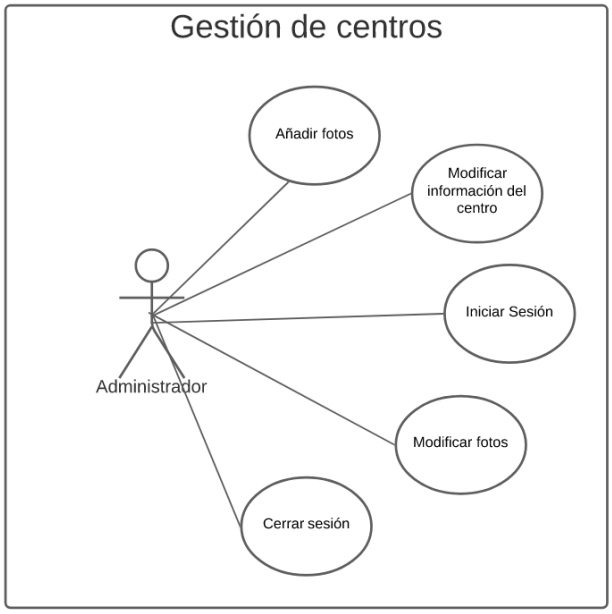

Ilustración 15 - Casos de uso para la característica: Gestión de centros

- **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.

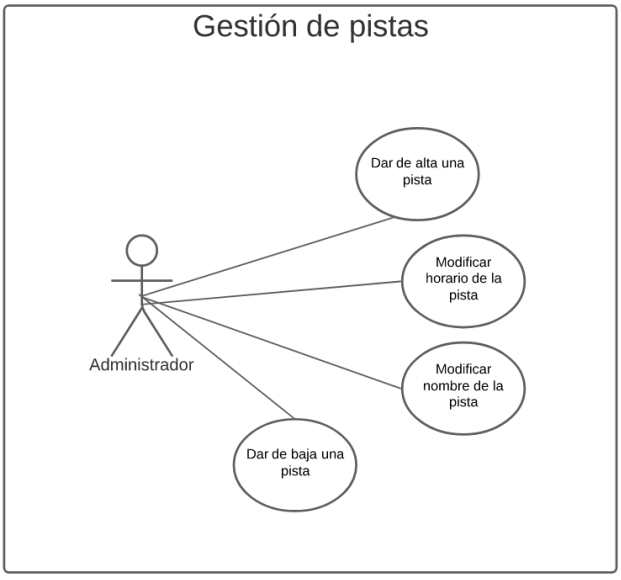

Ilustración 16 - Casos de uso para la característica: Gestión de pistas

- **Gestión de reservas:** Esta característica ofrece toda la funcionalidad necesaria para gestionar las reservas tanto de los usuarios de las instalaciones como de los administradores de estas.
    - **Reservar una pista:** Reservar una pista para un día 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

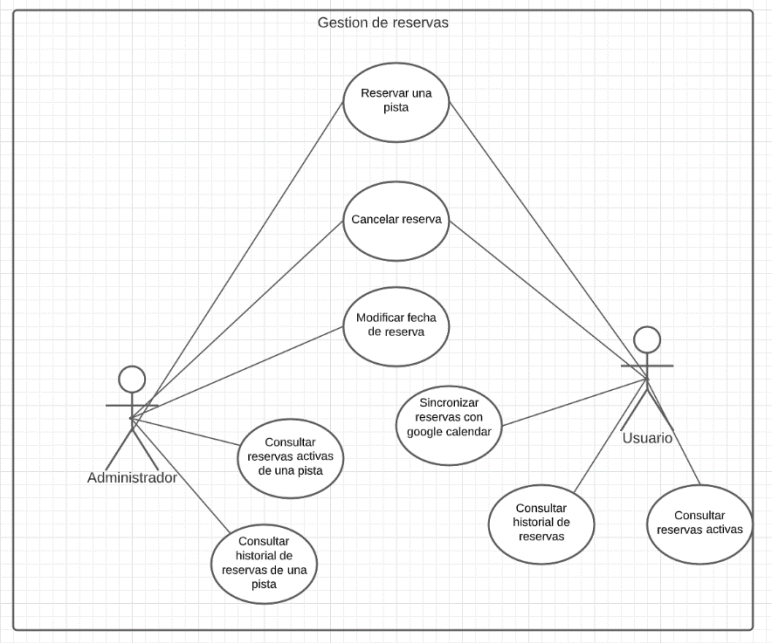

Ilustración 17 - Casos de uso para la característica: Gestión de reservas

- **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 las 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

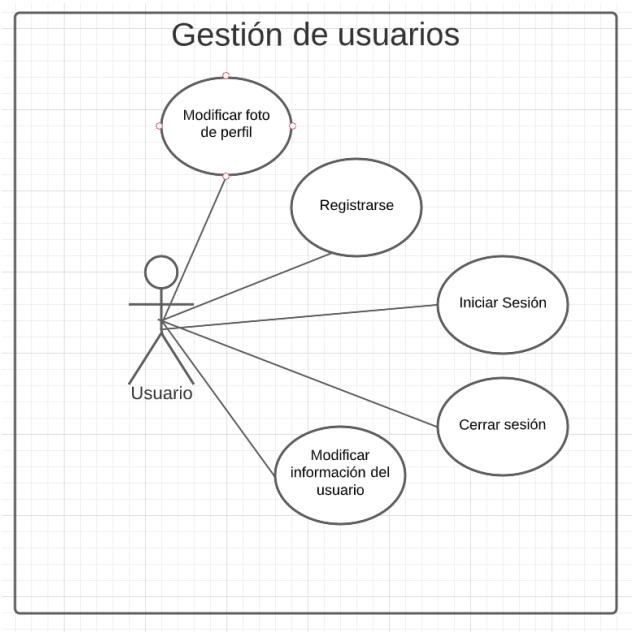

Ilustración 18 - Casos de uso para la característica: Gestión de usuario

- **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

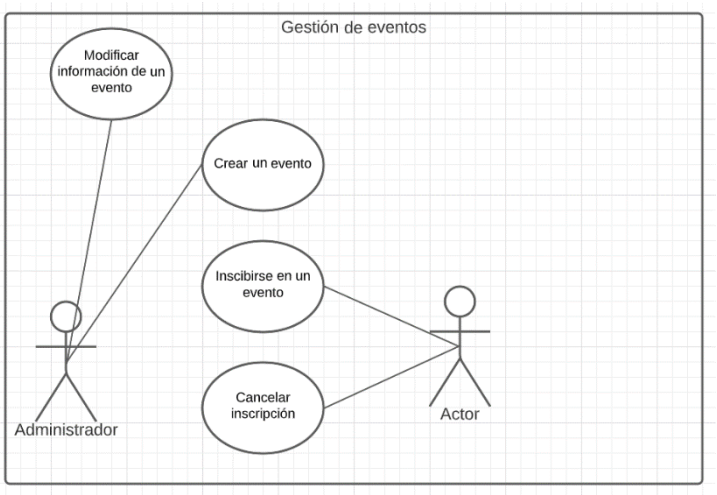

Ilustración 19 - Casos de uso para la característica: Gestión de eventos

- **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 los 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

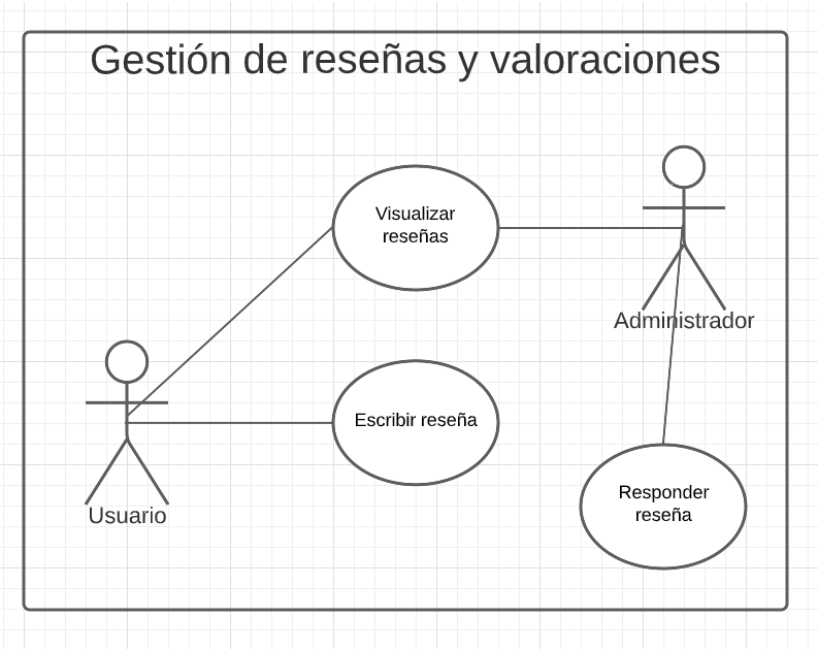

Ilustración 20 - Casos de uso para la característica: Gestión de reseñas y valoraciones

- **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

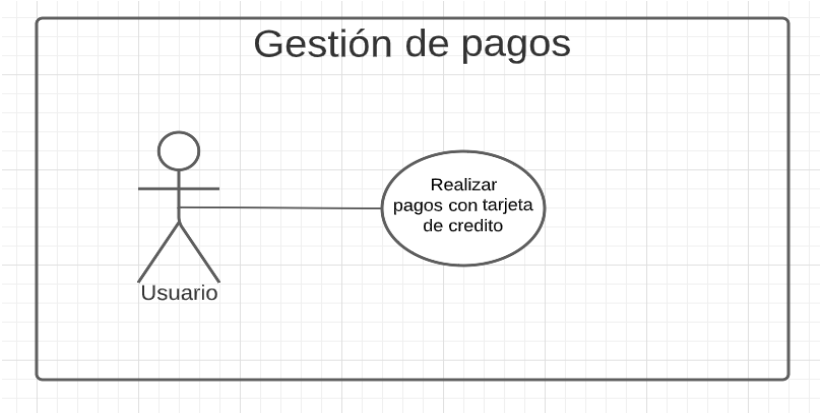

Ilustración 21- Casos de uso para la característica: Gestión de pagos

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 en castellano, 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 primer 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, en segunda instancia, en el segundo MVP, 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.

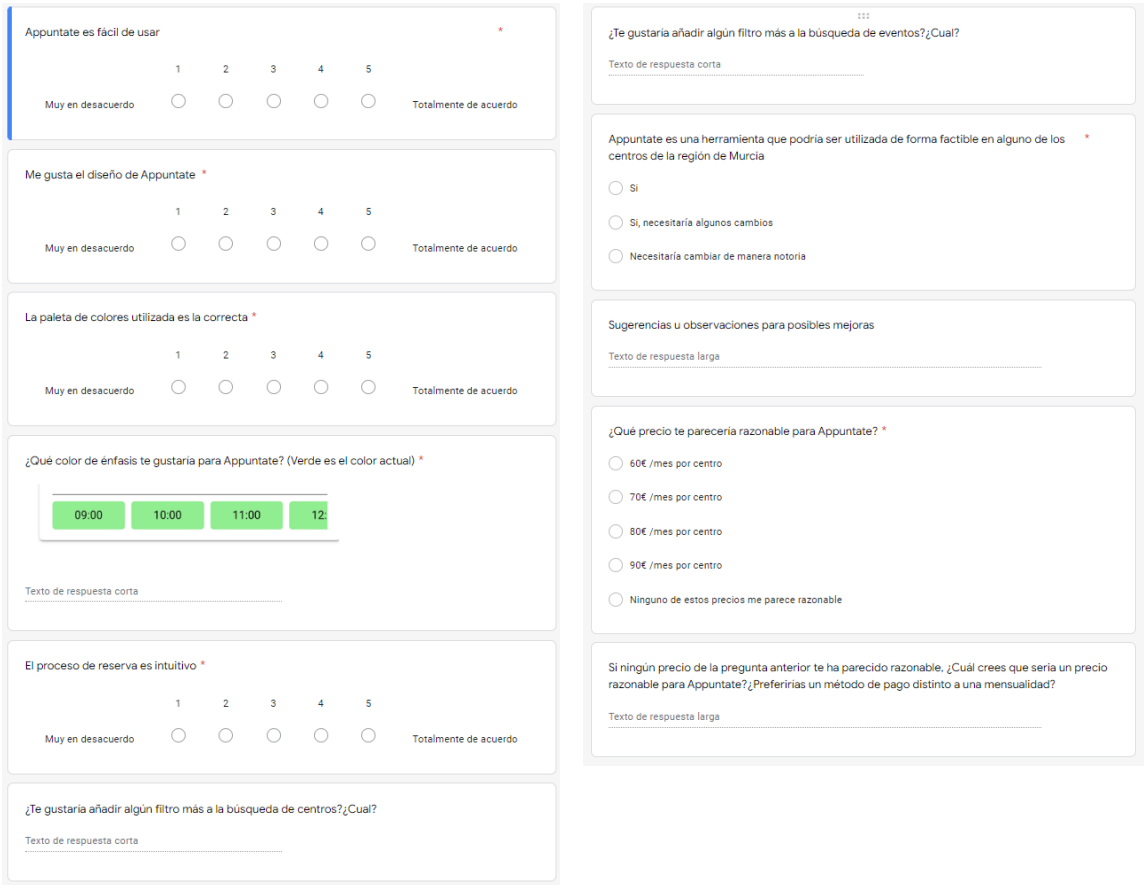

Ilustración 22 - Encuesta del primer MVP

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.

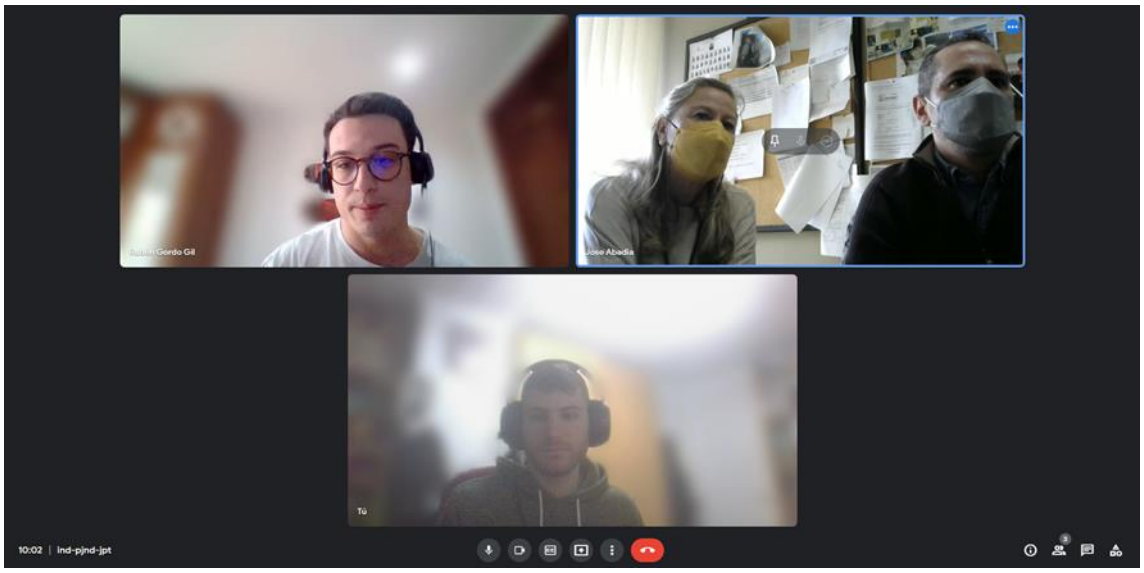

Ilustración 23 - Entrevista con el Ayuntamiento de Murcia

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 los 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.

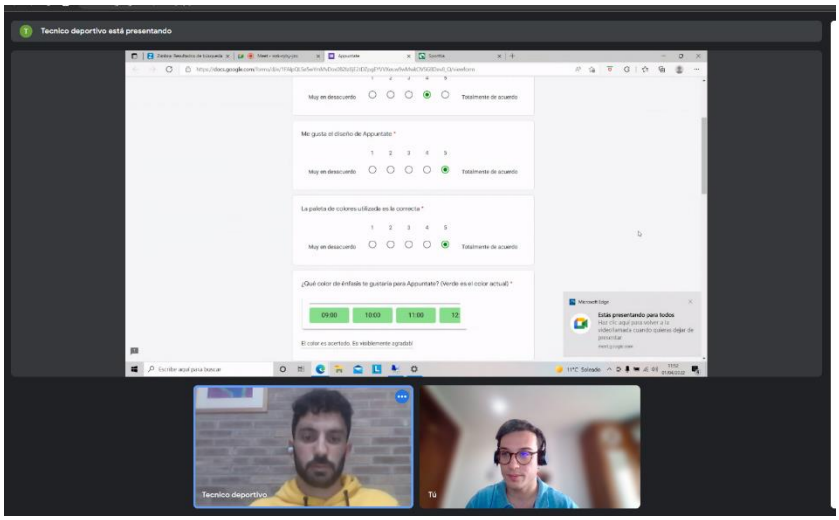

Ilustración 24 - Entrevista con el ayuntamiento de la Pobla de Farnals

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.

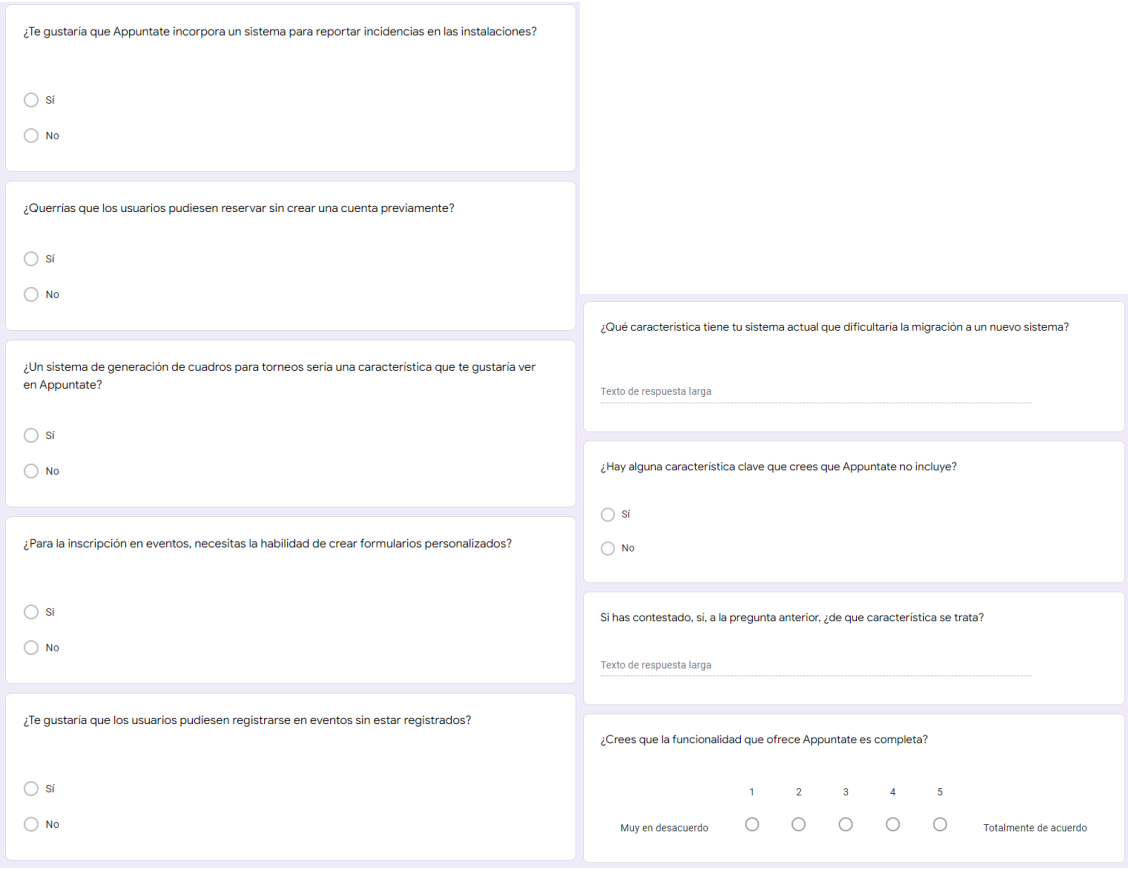

Ilustración 25 - Encuesta primer MVP

Para la realización de este experimento conseguimos contactar con el ayuntamiento de San Pedro, un municipio ubicado en Murcia.

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 resaltó acerca de Appuntate por encima del sistema que acaban de adquirir, era 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 que 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.

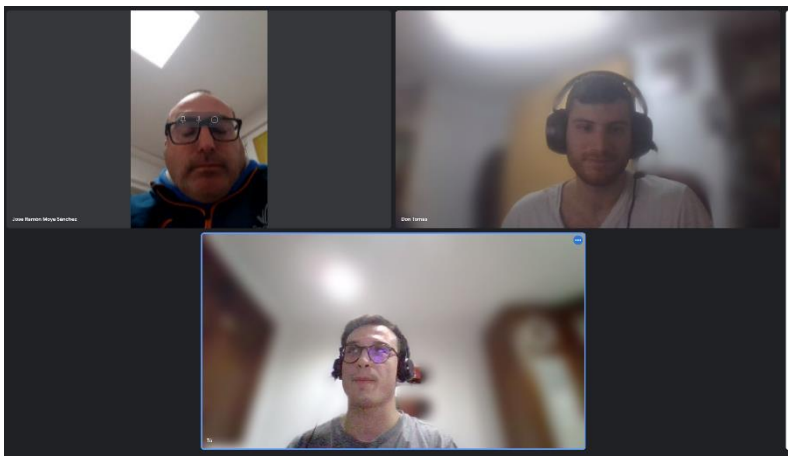

Ilustración 26 - Entrevista con el Ayuntamiento de San Pedro

5. Contexto tecnológico

El objetivo principal de la parte individual de este TFG es el desarrollo de una aplicación hibrida.

Las aplicaciones hibridas son aquellas que pueden correr en múltiples plataformas sin la necesidad de un desarrollo completo y específico para cada una de ellas, esto quiere decir que estas aplicaciones pueden correr tanto en la web como en sistemas operativos como **Android** e **iOS** de la misma manera que lo harían aplicaciones desarrolladas específicamente para estas plataformas. Este tipo de aplicaciones combinan elementos tanto de la web como de las plataformas en las que van a poder correr.

La gran ventaja de las aplicaciones hibridas sobre las aplicaciones nativas es, como se menciona anteriormente, que eliminan la necesidad de realizar un desarrollo específico para cada una de las plataformas en las que se quiere que lanzar la aplicación. El mismo código fuente se comparte entre la versión web y las versiones específicas de cada plataforma, esto no solo se traduce en que solo hay que realizar un único desarrollo, sino que también hay que mantener una única fuente de código, lo que hace que las aplicaciones hibridas sean, por norma general, más eficientes en cuanto a costes que las aplicaciones nativas, además de también ser más eficientes en cuanto al tiempo de desarrollo ya que desarrollar una única aplicación en lugar de varias, una para cada plataforma, implica un tiempo de desarrollo más corto.

Aunque las aplicaciones hibridas cuentan con muchas ventajas sobre las aplicaciones nativas, también traen consigo una serie de limitaciones, una de ellas siendo que una aplicación hibrida puede que no tenga acceso a ciertas características de los sistemas operativos o de los dispositivos donde va a correr debido a que puede que no existan herramientas que lo permitan. Otro inconveniente de las aplicaciones hibridas es que la interfaz de usuario es compartida a través de los distintos dispositivos y sistemas operativos, lo que puede ofrecer una experiencia menos inmersiva dentro del ecosistema de estos. Sumando a esta lista de limitaciones, también cabe mencionar que las aplicaciones nativas tienen la capacidad de obtener un rendimiento superior al de las aplicaciones hibridas, ya que no existe una capa intermedia entre el sistema operativo en el que se están ejecutando y la aplicación en sí.

Si bien es cierto que los inconvenientes mencionados son problemas reales, no son los únicos y sería posible listar más, la realidad es que estas limitaciones son cada vez menos significativas ya que conforme la tecnología avanza surgen herramientas como **Ionic** y **Capacitor**, en las que se profundizará más adelante, que dejan atrás muchos de estos inconvenientes mediante distintas soluciones como la oferta de una gran cantidad de plugins, que permiten a las aplicaciones híbridas acceder a características específicas de cada plataforma, la oferta de librerías de componentes con diseños que se adaptan a las plataformas y la optimización suficiente para obtener un muy alto rendimiento.

El hecho de que las ventajas que ofrecen las aplicaciones híbridas superen con creces sus inconvenientes ha hecho que las aplicaciones hibridas sean cada vez más populares.

5.1 Tecnologías utilizadas

En este aparatado se expone el conjunto de tecnologías utilizadas para el desarrollo del frontal de Appuntate explicando su uso y justificando su elección. La elección de las tecnologías utilizadas es muy importante en cualquier tipo de desarrollo de software ya que no todas las tecnologías son las óptimas para llevar a cabo cualquier tipo de proyecto.

Teniendo en cuenta la naturaleza de Appuntate el enfoque más coherente a la hora de elegir las tecnologías que se van a utilizar para el front-end de la aplicación es un enfoque basado en la web. Esto se debe a que Appuntate no va a ser instalada en la infraestructura del cliente ni mantenida por él. De esta manera para distribuir el software siguiendo el modelo de negocio elaborado, una aplicación que pueda ser accedida desde la web, además de distribuida a través de las tiendas de aplicaciones móviles, es lo ideal para llevar a cabo la iniciativa de Appuntate.

Esto cierra la puerta a tecnologías como Xamarin, una plataforma basada en .NET y C# que permite el desarrollo de aplicaciones multiplataforma que no están enfocadas a la web, y se la abre a librerías y frameworks como **Angular**, **Vue.js** o **React.js**, que junto a tecnologías como **Ionic**, **Capacitor** y **Cordova** permiten el desarrollo de aplicaciones hibridas basadas en la web.

De entre los frameworks y librerías disponibles para el desarrollo de aplicaciones híbridas basadas en la web los más relevantes en los últimos años han sido **React.js**, **Angular** y **Vue.js**. Cada uno de ellos con características diferentes que los hacen apropiados para tipos de proyectos específicos.

5.1.1 React.js

En primer lugar, **React.js** [8]. **React.js** es una librería de **JavaScript** mantenida por el equipo de Meta Open Source, antiguamente Facebook Open Source, para el desarrollo de aplicaciones web basado en la construcción de componentes individuales que pueden ser luego utilizados para componer una interfaz de usuario. Tiene un enfoque declarativo lo que significa que hay un desacoplamiento entre la lógica de la aplicación y el DOM (Document Object Model). El DOM representa la estructura de la página web en forma de árbol donde las etiquetas **HTML** que la conforman son los nodos de este y tienen sus correspondientes nodos padres e hijos. Este desacoplamiento significa que el DOM no se manipula directamente, sino que utilizando **React.js** el desarrollador declara como quiere sea el estado final de la página y es **React.js** el que se encarga de realizar las manipulaciones necesarias en el DOM para llegar al resultado deseado, un enfoque muy distinto al tomado por librerías más antiguas como jQuery que están basadas en la manipulación directa del DOM. Este enfoque hace que **React.js** sea más sencilla de utilizar resultando en un código más sencillo, más legible y fácil de mantener.

React.js también cuenta con **React Native**, un framework que permite desarrollar aplicaciones móviles para **iOS** y **Android** utilizando código escrito en **React.js**. **React native** se encarga de renderizar en código nativo de la plataforma el código escrito en **JavaScript**.

5.1.2 Vue.js

En segundo lugar, **Vue.js** [9]. A diferencia de **React.js**, **Vue.js** no es solo una librería, sino que es un ecosistema que puede adoptarse de manera incremental, es decir, puedes utilizar las distintas partes de este ecosistema según las necesidades del proyecto. No es mandatorio adoptar todos estos componentes para el desarrollo de un proyecto con **Vue.js**.

De la misma manera en la que **React.js** adopta un enfoque declarativo, **Vue.js** también lo hace. Cuenta con renderizado declarativo lo que significa que permite describir el estado final del **DOM** basándose en el estado en el que se encuentre el código escrito en **JavaScript**. Esto significa que **Vue.js** es reactivo ya que es capaz de detectar los cambios de estado en el código **JavaScript** y actualizar el estado del DOM cuando ocurren estos cambios.

Vue.js también está centrado en torno a la creación de componentes que después son utilizados para construir una interfaz de usuario. Cuenta con características como "*single-file-components*", o componentes en un solo archivo, que encapsulan la plantilla **HTML**, el código **JavaScript** y los estilos **CSS** en un solo archivo con extensión *.vue. Está es una característica que diferencia a **Vue.js** de **Angular** y **React.js** y trae ventajas como una sintaxis más sencilla y un proceso de compilado mejor optimizado, ventajas que pueden ser la razón por la cual ciertos proyectos opten por **Vue.js** en lugar de cualquier otro framework.

Al igual que **React.js**, **Vue.js** también contaba con una plataforma propia para el desarrollo de aplicaciones nativas llamada **Vue Native**, pero este framework ha sido deprecado. En su lugar, hoy en día, se utilizan tecnologías como **Ionic** y **Capacitor** junto a **Vue.js** para el desarrollo de aplicaciones nativas y **PWA**s (*Progressive Web Applications*). **Ionic** incluso cuenta con una versión construida específicamente para proyectos **Vue.js** que permite utilizar la CLI (*Command Line Interface*) de **Vue.js** en lugar de la CLI propia de **Ionic**, lo cual es una ventaja por encima de **React.js** y **Angular**.

5.1.3 Angular

En tercer lugar, **Angular** [10]. **Angular** es un framework diseñado para el desarrollo web mantenido por Google. **Angular**, al igual que **Vue.js** y **React.js** también tiene un enfoque declarativo y basado en la construcción de componentes individuales que más tarde pueden ser utilizados para la construcción de interfaces de usuario.

A diferencia de **React.js** y **Vue.js**, **Angular** está basado en **TypeScript,** y aunque es cierto que tanto **React.js** como **Vue.js** soportan el uso de **TypeScript**, es necesaria una configuración adicional y en algunos casos el uso de paquetes o librerías de terceros para que todas las características que trae **TypeScript** funcionen, es decir, en **Angular**, **TypeScript** funciona "*right out of the box*", una expresión en ingles que significa que funciona desde el primer momento sin ningún tipo de configuración extra, mientras que en **React.js** y **Vue.js** no es así. Esto es una ventaja que trae **Angular** por encima de **React.js** y **Vue.js** que al igual que los "*single-file-components*" en **Vue.js**, puede que sea la razón por la cual se decida utilizar **Angular** para un proyecto determinado por encima de otros frameworks. Esta integración permite que empezar un proyecto basado en **TypeScript** sea mucho más sencillo y rápido con **Angular** que con **Vue.js** o **React.js**.

Otra característica diferenciadora que comparten tanto **Angular** como **Vue.js** es el "*two-way-binding*", o enlazado bidireccional en castellano, una característica que permite que la comunicación entre dos ficheros sea mucho más sencilla ya que, cuando se utiliza este enlazado bidireccional, ambos ficheros que comparten datos pueden ver los cambios que ha realizado el otro de manera inmediata. Esto no es posible en **React.js** sin tener que realizar una implementación más compleja ya que, los datos solo fluyen en una dirección entre los ficheros, desde el fichero que tiene los datos hacia el fichero que los recibe.

Aunque cada una de estas plataformas tiene sus ventajas e inconvenientes, cualquiera de ellas podría ser utilizada para el desarrollo de Appuntate ya que todas permiten el desarrollo de **PWA**s y aplicaciones nativas con la ayuda de tecnologías como **Ionic**, **Capacitor** y **React Native** entre otras.

Angular ha sido el framework elegido debido a que está basado en **TypeScript** y cuenta, al igual que **Vue.js**, con enlazado de datos bidireccional, dos características que agilizan el desarrollo y traen las ventajas mencionadas anteriormente.

5.1.3.1 HTML

En **Angular**, a los archivos **HTML** se les llama "*Templates*", plantillas en castellano, y sirven para estructurar la vista de los componentes y de las interfaces de usuario que serán compuestas por esos componentes. Esta estructura se conforma mediante el uso de etiquetas, donde las etiquetas conforman un árbol dentro del archivo **HTML,** como se ha mencionado anteriormente, siendo estas etiquetas los nodos del árbol. Las etiquetas pueden tener etiquetas padre y etiquetas hijo.

Existen una gran variedad de etiquetas para encapsular distintos elementos dentro de la vista, como por ejemplo, la etiqueta <p> para texto, <img /> para imágenes y , que es utilizada para dividir las distintas secciones dentro de la vista. Estas etiquetas sirven como contenedores y en su mayoría, exceptuando etiquetas como la de imagen, se usan como apertura y cierre de distintas secciones de la vista. La etiqueta de apertura no cuenta con una barra, , y la de cierre cuenta con una barra antes del nombre de la etiqueta, . Todas las etiquetas que existan entre las etiquetas de apertura y cierre pasan a ser hijas de la etiqueta que las rodea conformando así el árbol.

Al crear un componente en **Angular** se genera de manera automática una etiqueta para ese componente que más tarde puede ser usada en las plantillas de otros componentes para componer la interfaz de usuario. Esta etiqueta se genera con el prefijo "app" seguido del nombre del componente, <app-mi-componente></ app-mi-componente>.

5.1.3.2 Sass

Sass, entre otras opciones que proporciona **Angular**, es un lenguaje utilizado para dar estilos a los archivos **HTML**.

Para conseguir esto el lenguaje Sass es compilado a **CSS**, otro lenguaje utilizado para dar estilos a los archivos **HTML**, en este aspecto, Sass, es muy parecido a **TypeScript**, el cual se compila a **JavaScript**.

Sass proporciona todas las funcionalidades que ofrece **CSS,** compartiendo su sintaxis y añadiendo a la misma proporcionando características que no están disponibles en **CSS**. Algunas de estas características son la adición de indentaciones en su sintaxis haciendo el código más legible, tiempos de carga más rápidos, declaración de variables para todo tipo de datos como fuentes y colores, además otras muchas características que hacen que Sass tenga toda la potencia de **CSS** sumando herramientas que facilitan su desarrollo y que no están presentes en **CSS**.

5.1.3.3 TypeScript

TypeScript es el lenguaje sobre el cual está basado **Angular**. Fue desarrollado por Microsoft, lanzado en 2012 y sigue siendo mantenido Microsoft.

TypeScript es un lenguaje basado en **JavaScript** que es compilado a **JavaScript** en lugar de ser interpretado directamente por el navegador. **TypeScript** añade una sintaxis para los tipos. Mientras que **JavaScript** es un lenguaje con tipado dinámico, es decir, el tipo de las variables se infiere en tiempo de ejecución, **TypeScript** cuenta con un tipado estático, lo que quiere decir que el tipo de las variables se sabe en tiempo de compilación. Esto se consigue ya que es el programador es el que se encarga de establecer el tipo que tienen las variables en el código, por ejemplo, para declarar una constante que contendrá una cadena de caracteres la sintaxis sería la que se muestra en la ilustración 27:

```
const testString: string = 'cadena de texto';
```

Ilustración 27 - Sintaxis TypeScript

Sin embargo, a diferencia de lenguajes como Java, que **TypeScript** cuente con tipado estático no significa que sea necesario declarar de manera explícita el tipo de las variables ya que **TypeScript** es capaz de inferir tipos. En el siguiente ejemplo no se declara de manera explícita el tipo de la constante pero **TypeScript** infiere que la constante es de tipo “string”.

```
const testString = 'cadena de texto';
```

Ilustración 28 – Inferencia de tipos en TypeScript

Esto ocurre de igual manera con variables que no sean de tipos primitivos como cadenas y booleanos, por ejemplo, si declaramos una constante a la que asignamos un objeto que contiene atributos de tipos primitivos, una función que tiene como parámetro de entrada una cadena e intentamos llamar a la función con la constante, **TypeScript** inferirá el tipo de la constante, comprobara que no se trata de una cadena de texto y mostrara un error.

```typescript
const testCar = {
    model: 'modelo del coche',
    numberOfDoors: 5,
    convertible: false,
    extras: ['asientos calefactables', 'ventanillas tintadas']
};

// tipo inferido del coche de prueba
//     TestCarType{
//         model: string;
//         numberOfDoors: number;
//         convertible: boolean;
//         extras: string[];
//     }

// función que toma como parametro una cadena
function getCarManufacturer(model: string) {
    return 'fabricante'
}

// llamada a función con el coche de prueba
getCarManufacturer(testCar);
```

Ilustración 29 – Inferencia de tipos de objetos en TypeScript

Esto es una gran ventaja sobre **JavaScript** ya que facilita la identificación de bugs relacionados con los tipos. En código **JavaScript** esta inconsistencia con los datos no generaría ningún tipo de error hasta que el código se ejecute, sin embargo, en **TypeScript** el error se puede identificar en el mismo momento en el que se escribe el código.

TypeScript permite la creación de tipos mediante el uso de interfaces. Los tipos pueden usarse para declarar variables y asegurarse de que el valor que toman esas variables es consistente con el tipo con el que han sido declaradas, utilizando el ejemplo anterior, en la ilustración 30 se puede ver la definición de la interfaz del tipo Car y como **TypeScript** se asegura de que el valor que toman las variables de tipo Car sea consistente con la interfaz.

```typescript
interface Car{
    model: string;
    numberOfDoors: number;
    convertible: boolean;
    extras: string[];
}

const testCar: Car = {
    model: 'modelo del coche',
    numberOfDoors: 5,
    convertible: false,
    extras: ['asientos calefactables', 'ventanillas tintadas']
};

const fakeCar: Car = {
    model: 'modelo del coche',
    numberOfDoors: '5',
    convertible: false,
    extras: ['asientos calefactables', 'ventanillas tintadas']
}
```

Ilustración 30 - Uso de interfaces en TypeScript

Gracias a esta inferencia de tipos cualquier código escrito en **JavaScript** también es código escrito en **TypeScript,** lo que significa que todo código que pueda ser escrito en **JavaScript** también puede ser escrito en **TypeScript**, otorgando a **TypeScript** toda la potencia de **JavaScript** sumada a la sintaxis de tipos, lo que ha convertido a **TypeScript** en uno de los lenguajes de programación más populares en los últimos años [11].

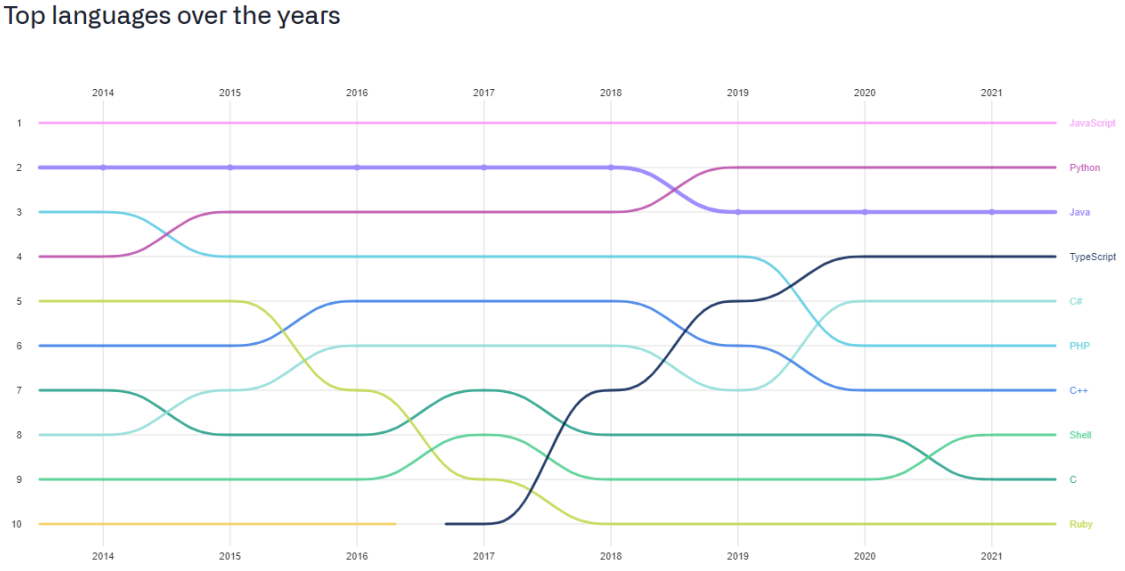

Ilustración 31 - Popularidad de lenguajes de programación en los últimos años

Todas estas características son algunas de las razones por las cuales se ha elegido **Angular** por encima de **React.js** o **Vue.js**. Disponer de **TypeScript** sin la necesidad de ningún tipo de configuración adicional o el uso de paquetes o librerías de terceros agiliza el proceso de desarrollo y ayuda en la detección de errores en el código.

5.1.4 Ionic

Para generar una **PWA** además de las versiones nativas para **Android** e **iOS** de Appuntate se ha elegido **Ionic** [12].

Ionic es un conjunto de herramientas diseñadas para el desarrollo de aplicaciones hibridas que permite desplegar aplicaciones desarrolladas con **Ionic** tanto en la web en forma de **PWA** como en sus versiones nativas de **iOS** y **Android**. **Ionic** consigue esto a través del uso de SDKs nativos de cada plataforma que permiten renderizar aplicaciones desarrolladas con **Ionic** a través de una vista web, sin embargo, esto no significa que una aplicación desarrollada en **Ionic** sea una página web envuelta dentro de una aplicación nativa. Las aplicaciones desarrolladas en **Ionic** tienen acceso a características nativas de cada una de las plataformas utilizando herramientas como **Capacitor**, sobre la cual se profundizará en el siguiente apartado, o **Cordova**, características que van desde el acceso a la cámara, acceso al GPS del dispositivo, notificaciones “push”, hasta el acceso a sensores con los que cuente el dispositivo, por ejemplo, el giroscopio o el acelerómetro.

Ionic cuenta con un paquete de herramientas específicas para su uso con **Angular** llamado “angular-toolkit” que proporciona todo lo necesario para integrar **Ionic** en una aplicación **Angular**, por ejemplo, integra la CLI de **Angular** dentro de la CLI de **Ionic** para proporcionar características específicas para **Angular**.

Ionic, además de proporcionar las herramientas necesarias para poder desplegar una aplicación **Angular** como una **PWA** o una aplicación nativa, también proporciona una extensa librería de componentes que pueden ser utilizados desde **Angular** para construir las interfaces de usuario. La librería de componentes de **Ionic** es extensa y cuenta con una documentación excelente, haciendo que el uso de estos componentes dentro de una aplicación **Angular** sea muy sencillo, en adición, **Ionic** tiene la capacidad de detectar en que plataforma está siendo ejecutada la aplicación y puede adaptar el diseño de los componentes de **Ionic** con un diseño específico para esa plataforma, diseñado para que se integre mejor con esta. Esto significa que el desarrollador no tiene que preocuparse de maquetar sus propios componentes para cada una de las plataformas.

Ionic cuenta con un rendimiento excelente ya que utiliza herramientas como la aceleración por hardware que le permite utilizar recursos de la GPU, en lugar de la CPU, para el renderizado de imágenes y transiciones, logrando una mayor fluidez y velocidad. El uso de este tipo de herramientas logra que una aplicación de **Ionic** pueda obtener un rendimiento muy similar al de una aplicación nativa.

5.1.5 Capacitor

Capacitor es una plataforma que da acceso a SDKs nativos a través de una librería de plugins [13].

Para que **Ionic** pueda acceder a las funcionalidades nativas de la plataforma en la que se está ejecutando la aplicación, es necesaria una capa de compatibilidad, **Capacitor** proporciona esta capa de compatibilidad. **Capacitor** se encarga de envolver la aplicación web dentro de un "WebView" [14], una ventana que permite renderizar la aplicación web dentro de una aplicación nativa, este "WebView" no solo permite renderizar la aplicación, sino que también actúa como un puente entre la aplicación web, el hardware y el sistema operativo.

Para poder acceder al hardware y a las propiedades específicas de cada plataforma, **Capacitor** proporciona una extensa librería de plugins, estos plugins proporcionan acceso APIs nativas, necesarias para acceder a funcionalidades hardware y funcionalidades específicas de cada plataforma, envolviendo estas APIs nativas y proporcionando nuevas APIs accesibles desde **JavaScript** que dan acceso a las APIs nativas.

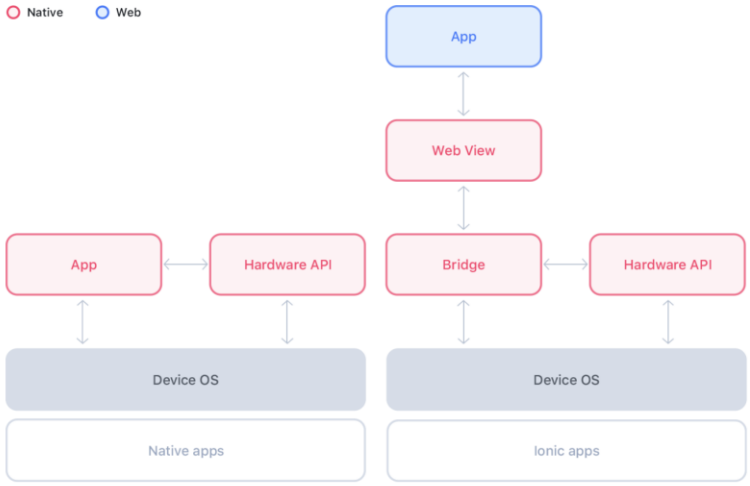

Ilustración 32 - Arquitectura de Capacitor

5.2 Desarrollo multiplataforma

En este apartado se detallan las distintas plataformas a las cuales ira dirigido el desarrollo de Appuntate además de explicar las pautas necesarias para el desarrollo en cada una de ellas

Partiendo del objetivo de que Appuntate se distribuya a través de la web y de las tiendas de aplicaciones, Appuntate se ha desarrollado pensando en tres plataformas, la web, **Android** e **iOS**.

Para el despliegue en la web se ha decidido desarrollar una versión **PWA** (Progressive Web App). Las **PWA**s son aplicaciones web diseñadas para ofrecer al usuario una experiencia nativa, esto lo consiguen mediante una serie de diferencias con aplicaciones web corrientes. A diferencia de las aplicaciones web corrientes, las **PWA**s pueden instalarse en un dispositivo, están diseñadas para que sean "responsive", es decir, la interfaz de la aplicación se adapta al tamaño de la pantalla o de la ventana en la que se está ejecutando, y pueden funcionar de manera online.

Además de ofrecer ventajas sobre aplicaciones web convencionales, las **PWA**s también ofrecen ventajas sobre aplicaciones nativas. A diferencia de las aplicaciones nativas, la instalación de una **PWA** es muy sencilla, consistiendo en dos clics desde el navegador, no son necesarios instaladores ni el acceso a tiendas de aplicaciones. Utilizando el navegador Chrome, al acceder a una aplicación web que es una **PWA**, para instalarla solo es necesario hacer clic en el botón de descarga que aparece a la derecha de la barra de navegación.

Más allá de la sencillez de la instalación, las **PWA**s también tienen como ventaja que, por lo general, su tamaño suele ser menor que el de aplicaciones nativas, esto significa que en el caso de Appuntate la implantación mediante una **PWA** sea muy sencilla por parte del cliente, incluso más si contamos con que las **PWA**s pueden ser accedidas desde la web, a través de cualquier dispositivo, y que gracias a su enfoque, mencionado anteriormente, "responsive", se adaptan al dispositivo en el que se están ejecutando.

En cuanto a los dispositivos móviles, el desarrollo se ha realizado teniendo en mente dos plataformas, **Android** e **iOS**. Aunque una **PWA** puede servir para distribuir la aplicación y que pueda ser accedida desde cualquier dispositivo a través de la web, lanzar una versión para **Android** e **iOS** permite acceder a funcionalidades propias de los dispositivos y ofrecer una experiencia móvil más completa. En adición, existen dos tiendas de aplicaciones, la Google Play Store y la Apple App Store que son increíblemente populares, contando con 111,3 mil millones de descargas en el caso de la Google Play Store [15] y en el caso de la Apple App Store, 32,3 mil millones en 2021 [16], estas tiendas son la principal fuente desde la cual los usuarios de dispositivos **Android** e **iOS** descargan sus aplicaciones, así que desde un punto de vista meramente empresarial, dejando a un lado los aspectos técnicos, existen muchas ventajas al proporcionar una aplicación a través de estas tiendas.

Después de haber expuesto las distintas plataformas hacia las cuales está enfocado el desarrollo, en los siguientes apartados se detallan las pautas técnicas necesarias para el desarrollo en cada una de las plataformas.

5.2.1 PWA

Para convertir una aplicación web en una **PWA** existen una serie de requerimientos que tienen que cumplirse para que obtenga todas las funcionalidades propias de una **PWA** como, por ejemplo, ser instalable y poder ejecutarse sin necesidad de conexión a internet [17].

5.2.1.1 Web Manifest

En primer lugar, para convertir una aplicación web en una **PWA** es necesario un "Web manifest", un archivo que indica al navegador todo lo que necesita saber en cuanto a cómo ha de comportarse la aplicación una vez esta es instalada por el usuario.

El archivo "Web manifest" puede tener formato JSON o .webmanifest y tiene el aspecto que se muestra en la ilustración 33.

Ilustración 33 - Web Manifest

En este archivo se describen características clave de la aplicación que necesita el navegador para poder instalarla, se declaran el nombre, “name”, el cual tomara la aplicación cuando se instale, un nombre más corto, “short_name”, que aparecerá en las partes de la aplicación en las que por espacio en pantalla no se pueda mostrar el nombre, el color del tema, “theme_color”, que es el color que tomara la ventana de la aplicación, el color de fondo, “background_color”, que es el color que tomara la aplicación mientras carga cuando se abre en dispositivos móviles, “display”, propiedad que define como se muestra la interfaz propia del navegador, es decir, la barra de búsqueda, los botones de navegación, etc., además de permitir que la aplicación se inicie en modo pantalla completa, el conjunto de URLs que forman parte de la aplicación, definido en la propiedad “scope” y la URL base sobre la cual empezara la ejecución de la aplicación, definida

como “start_url”. En este archivo también se define un vector de imágenes en la propiedad “icons” que serán utilizadas por la aplicación para mostrarlos como el icono de la aplicación, el icono que aparece en la ventana en la que se ejecuta, el icono que toma en la barra de herramientas, etc. Todos los campos mencionados son necesarios para convertir la aplicación web en una **PWA**.

5.2.1.2 Service Worker

Otro de los requerimientos de una **PWA** es un “Service worker”. Un “Service worker” es un tipo de “Web worker”, que no es más que código **JavaScript** que se ejecuta en segundo plano [18]. Esto se consigue ejecutando este código **JavaScript** en un hilo de ejecución distinto al hilo de la aplicación, solventando algunos inconvenientes que trae el hecho de **JavaScript** se ejecute en un solo hilo de ejecución, como por ejemplo la asincronía. **JavaScript** se ejecuta en un único hilo de ejecución, lo que significa que todas las instrucciones se ejecutan de manera secuencial sin ningún tipo de paralelismo.

Los “Web workers” son una manera de realizar operaciones asíncronas como llamadas a una API REST, ya que se ejecutan en un hilo distinto, esto consigue que al realizar una llamada que puede llevar un tiempo no sea necesario bloquear la aplicación y que el usuario pueda seguir interactuando con ella. También pueden utilizarse para mejorar el rendimiento de ciertas aplicaciones web que requieran de una potencia computacional muy grande, como aplicaciones que renderizan imágenes CAD, repartiendo la carga computacional entre varios hilos.

En concreto el “Service worker” necesario para una **PWA** se va a encargar de una serie de tareas, una de ellas siendo interceptar cualquier tipo de petición que realice la aplicación. La utilidad detrás de esto es que el “Service worker” tiene acceso a la memoria cache, memoria que puede utilizar para guardar archivos, peticiones y respuestas HTTP entre otras cosas, de esta manera el "Service worker" puede ser utilizado para almacenar todos los recursos necesarios para que la aplicación funciona de manera offline en la memoria cache. También puede interceptar peticiones HTTP y devolver una respuesta concreta si la aplicación se encuentra offline, o almacenar respuestas a peticiones determinadas para devolverlas directamente al usuario sin la necesidad de esperar a la respuesta del servidor. El "Service worker" también es el encargado de mostrar notificaciones incluso cuando la aplicación no se esté ejecutando en primer plano, siempre y cuando el navegador este corriendo, de esta manera la **PWA** puede recibir notificaciones en todo momento [19].

La necesidad de incorporar un “Service worker” para poder obtener una **PWA** trae consigo una implicación, las llamadas realizadas al servidor desde la aplicación tienen que estar servidas por HTTPS, esto es debido a que, por razones de seguridad, los “Service workers” no funcionan a través de HTTP.

5.2.1.3 Construcción

Gracias a **Angular** e **Ionic** el cumplimiento de todos estos requerimientos es muy sencillo y se pueden completar siguiendo unos pocos pasos.

Para convertir Appuntate en una **PWA** se ha utilizado un comando proporcionado por la CLI de **Angular** que añade un “Service worker” y toda la configuración necesaria para que funcione, además del archivo “manifest” al proyecto. El comando es “ng add @angular/**pwa**”.

Una vez realizados estos pasos es posible realizar una "*build*" de la aplicación, tras la "*build*", cómo podemos observar en la ilustración 34, ya es posible descargar Appuntate desde un navegador.

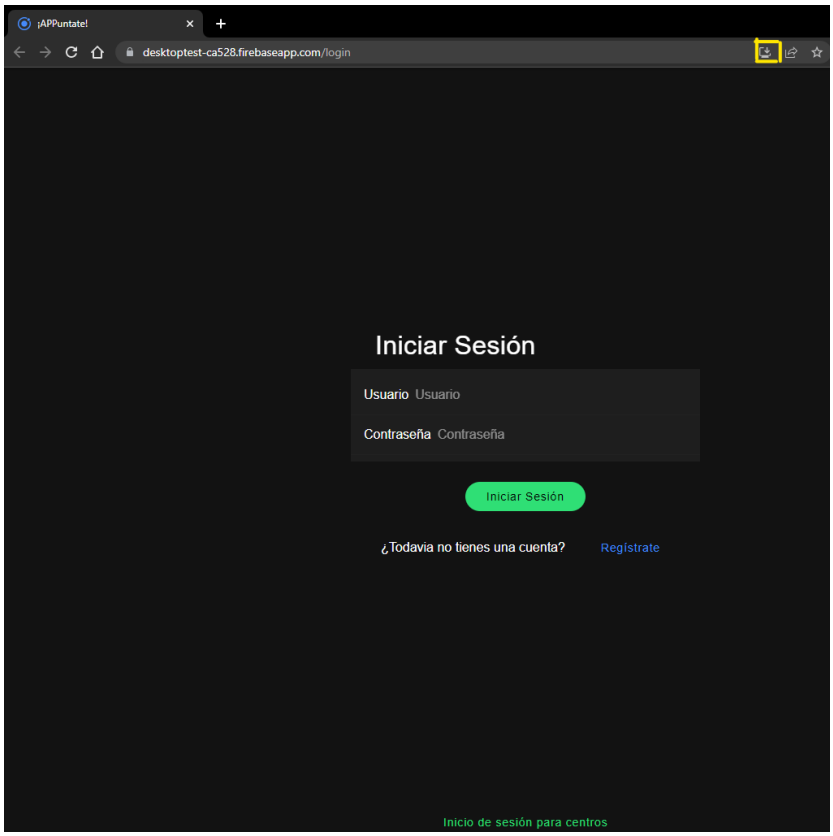

Ilustración 34 - Descarga de Appuntate como PWA

5.2.1 Android

En el caso de las versiones nativas de Appuntate, en este caso la de **Android**, **Ionic** y sobre todo **Capacitor** toman el papel principal ya que son las dos herramientas que permiten generar las aplicaciones nativas.

A través de la CLI de **Ionic** podemos utilizar **Capacitor** para generar la versión **Android** de Appuntate.

Los comandos son:

- ionic capacitor add android
    - crea el proyecto **Android** con todos los archivos necesarios
- ionic capacitor build android
    - construye una versión ejecutable del proyecto **Android**

Una vez ejecutados estos comandos sobre el directorio raíz de Appuntate se obtiene la versión **Android** de la aplicación, esta versión puede ser ejecutada tanto en emuladores como en dispositivos físicos y será posteriormente utilizada para el lanzamiento en la Google Play Store.

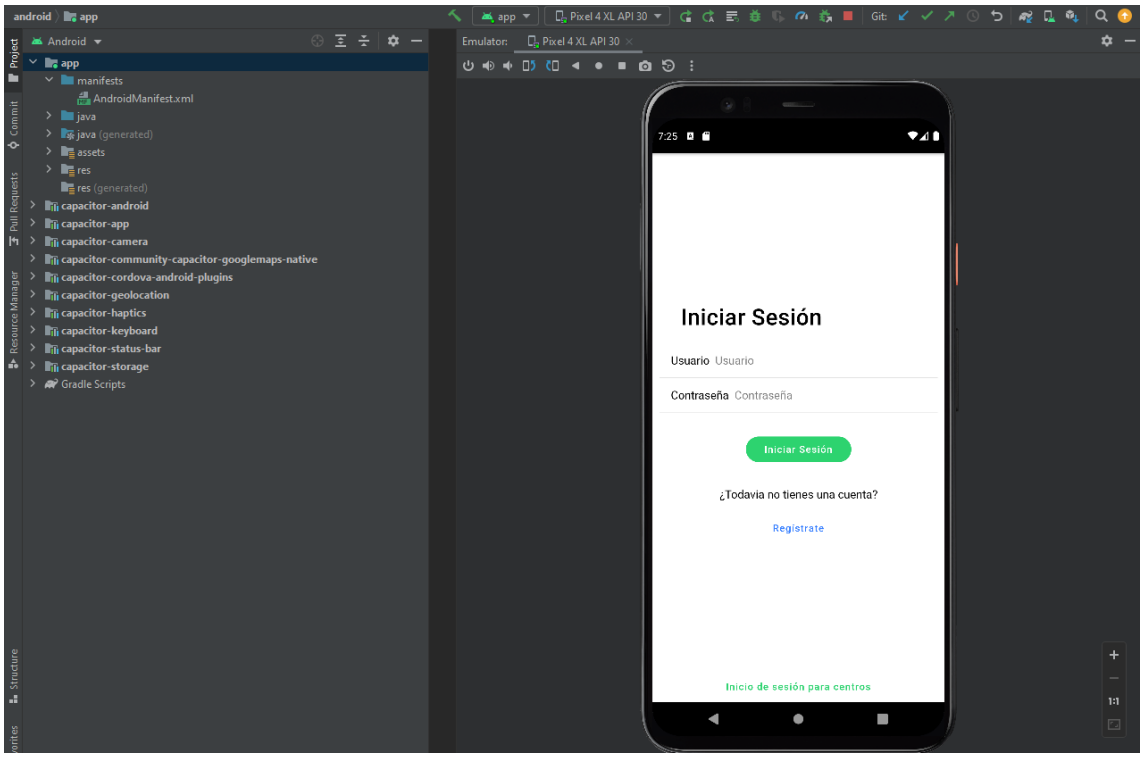

Ilustración 35 - Versión Android de Appuntate ejecutándose en un emulador desde Android Studio

5.2.3 iOS

En el caso de la versión para **iOS** el proceso es idéntico al de **Android**. Se hace uso de la CLI de **Ionic** para construir un proyecto nativo con **Capacitor**. Solo es necesario ejecutar dos comandos.

Los comandos son:

- ionic capacitor add ios
    - crea el proyecto **iOS** con todos los archivos necesarios
- ionic capacitor build ios
    - construye una versión ejecutable del proyecto **iOS**

Una vez ejecutados estos comandos sobre el directorio raíz de Appuntate la aplicación está lista para ejecutarse en un emulador o un dispositivo **iOS** físico. Antes de poder ejecutarla habrá que configurar un certificado de firma para el cual será necesario un ID de Apple.

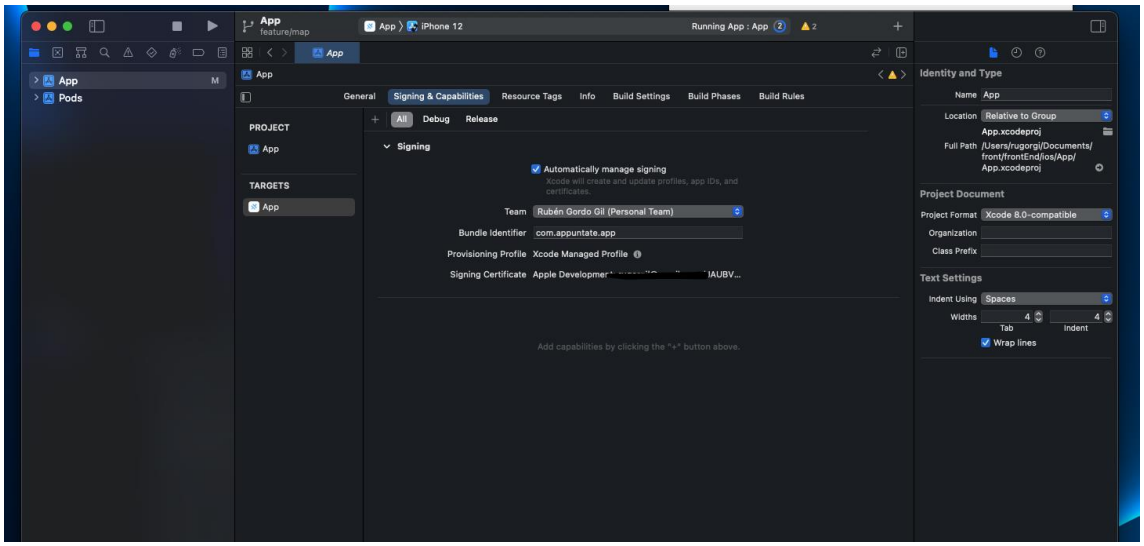

Ilustración 36 - Configuración del certificado de firma

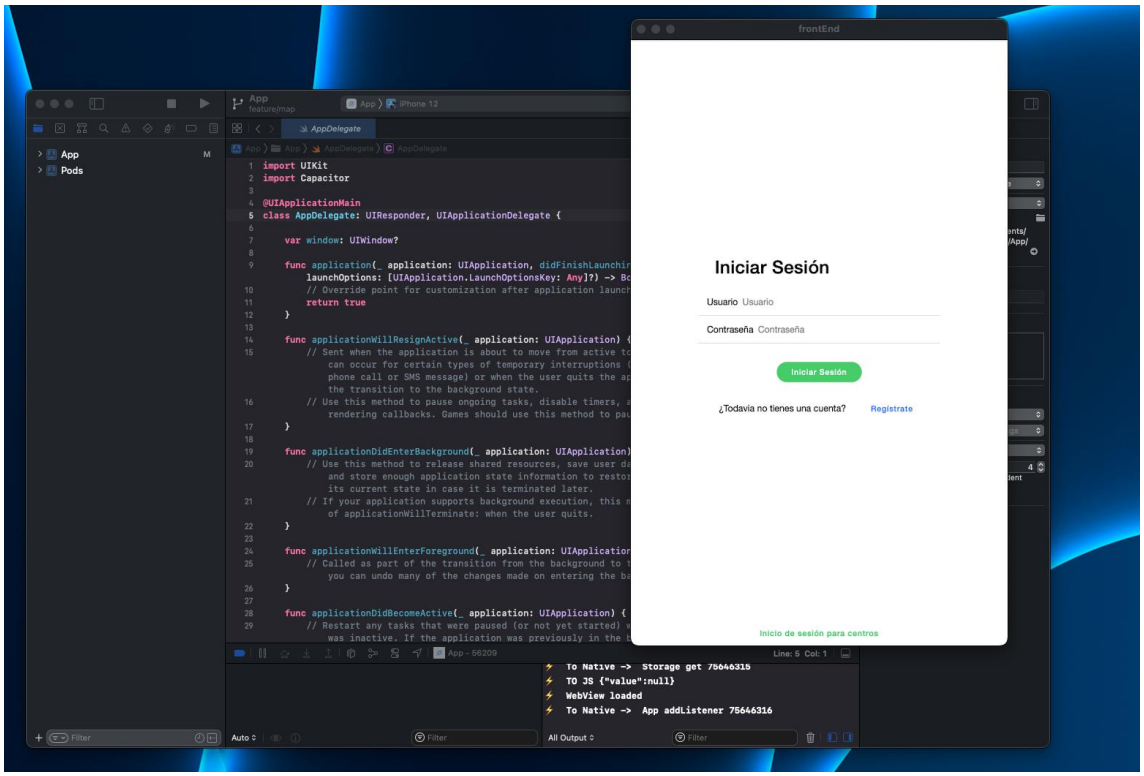

Ilustración 37 - Versión iOS de Appuntate ejecutándose en un emulador en XCode

5.3 Entorno de desarrollo

En este apartado se exponen todas las herramientas que se han utilizado para el desarrollo de Appuntate, se explica el funcionamiento y el porqué de la elección de estas.

En cualquier tipo de proyecto de desarrollo de software el entorno de desarrollo es de vital importancia. La elección de las herramientas utilizadas puede actuar a favor o en detrimento del proyecto así que es muy importante analizar las herramientas que se van a utilizar para completar el desarrollo.

5.3.1 Visual Studio Code

En primer lugar, el editor de texto utilizado para escribir el código fuente de Appuntate, Visual Studio Code.

Visual Studio Code es un editor de texto desarrollado por Microsoft que fue lanzado en 2015 y que cuenta con características propias de un IDE, *Integrated Development Enviroment*, o en castellano, Entorno de Desarrollo Integrado, y que puede ser considerado un IDE en sí mismo. Algunas de estas características son, su "*debugger*" o depurador, que permite depurar el código directamente desde el editor, la integración con Git y el acceso a una gran librería de extensiones que permiten expandir la funcionalidad del editor.

Para el desarrollo de Appuntate no se han utilizado las herramientas de depuración que incorpora Visual Studio Code pero sí se ha hecho uso de una gran cantidad de extensiones que se de detallan a continuación.

- Angular Language Service
    - Proporciona una lista de sugerencias, funcionalidad de autocompletado y mensajes de diagnóstico AOT(C), *Ahead of Time (Compilation)*. Estos mensajes utilizan la característica AOT(C) de **Angular** que permite compilar el código antes de ejecutarlo en el navegador, de esta manera esta extensión puede proporcionar mensajes de error que aparecerían durante el tiempo de compilación mientras se desarrolla la aplicación, ahorrando mucho tiempo de depuración ya que esta funcionalidad es capaz de detectar errores en el momento en el que se está escribiendo el código [20].

- Material Icon Theme
    - Ofrece una serie de iconos para las carpetas y archivos dentro de la barra del explorador de archivos de Visual Studio Code. Dependiendo del nombre de la carpeta y de la extensión de los archivos, se aplica un icono u otro, haciendo que la estructura de archivos sea más visual y haciendo más fácil localizar archivos y carpetas además de permitir visualizar la estructura de directorios del proyecto de una forma más clara [21].

- Prettier
    - Formateador de código. Permite formatear el código de manera que este tenga un formato consistente basándose en una serie de reglas, como por ejemplo, la longitud máxima de una línea, espaciado, tabulado, etc., esto permite que todo el código de la aplicación tenga el mismo formato, lo cual es muy útil cuando participan varias personas en un mismo proyecto ya que independientemente de quien haya escrito el código, este tendrá el mismo formato [22].

- ESLint
    - Herramienta que permite detectar patrones problemáticos dentro del código. Ayuda en la detección de errores sin la necesidad de ejecutar el código. Está basado en reglas que pueden ser modificadas, añadidas o eliminadas [23].

- npm scripts
    - Permite ejecutar los scripts definidos en el archivo "package.json" directamente desde Visual Studio Code sin la necesidad de una consola. Facilita y hace más rápido el proceso de ejecutar la aplicación, ejecutar pruebas, construir la aplicación, etc. [24].

- Thunder Client
    - Herramienta que permite realizar peticiones HTTP directamente desde Visual Studio Code. Permite realizar cualquier tipo de peticiones permitiendo editar el cuerpo, las cabeceras y los parámetros de la petición. Una vez realizada la llamada permite visualizar la respuesta del servidor mostrando el estado de la petición, el tamaño de la respuesta, el tiempo que ha tardado el servidor en responder, etc. Es especialmente útil para realizar pruebas rápidas, aunque no es tan potente como herramientas como Postman. Agiliza el proceso de prueba ya que es accesible desde el editor de código [25].

- CodeMetrics
    - Extensión que calcula la complejidad de las funciones en tiempo real y muestra un mensaje encima de cada función con su complejidad y un comentario diciendo que, o bien, la complejidad de la función está dentro de un rango aceptable o si la función es demasiado compleja y debería considerarse una refactorización. Haciendo clic en el comentario muestra todo el cálculo realizado para llegar a la complejidad que se muestra encima de la función [26].

El acceso a todas estas extensiones son la razón por la cual se ha escogido Visual Studio Code. La cantidad de funcionalidad extra accesible desde estos plugins convierte a Visual Studio Code en uno de los editores de texto más potentes para el desarrollo de aplicaciones web. Aunque es cierto que trae consigo ciertos inconvenientes como un uso de memoria mucho más elevado que otros editores de texto como Sublime Text, la agilidad que proporcionan las extensiones en cuanto al desarrollo es más que suficiente para justificar este tipo de desventajas.

5.3.2 Android Studio

Android Studio es el IDE oficial ofrecido por Google para el desarrollo de aplicaciones Android, está basado en la plataforma IntelliJ, una plataforma desarrollada para la creación de herramientas para desarrolladores e IDEs, esto se debe a que Android Studio ha sido desarrollado por Google en conjunto con JetBrains, la compañía responsable de la plataforma IntelliJ.

Este IDE permite el desarrollo de aplicaciones **Android** escritas tanto en Java como en Kotlin, sin embargo, en el caso de Appuntate, Android Studio no se ha usado como una herramienta de desarrollo, sino de depuración ya que **Capacitor** se encarga de generar todo el proyecto **Android** sin tener que escribir ni una sola línea de código en Java o Kotlin.

Aunque Android Studio no es el único IDE disponible que permite el desarrollo de aplicaciones **Android**, Android Studio es el más indicado para el desarrollo de Appuntate ya que no va a ser utilizado para escribir código, lo que elimina la necesidad de que ofrezca funcionalidades avanzadas que pueden ser necesarias en proyectos escritos en Java o Kotlin, y trae consigo, sin la necesidad de una configuración demasiado compleja, todas las herramientas de depuración necesarias para Appuntate.

La primera de estas herramientas es la integración de emuladores dentro del IDE. Desde el mismo IDE es posible crear emuladores de dispositivos **Android** con una gran variedad de características, pudiendo elegir desde el modelo de dispositivo que va a ser emulado, la versión de **Android** instalada en el emulador, hasta la posibilidad de acceder a funciones hardware como sensores biométricos, giroscopio, acelerómetro y GPS. Estos emuladores permiten ejecutar la aplicación como si estuviese instalada en un dispositivo real para realizar tareas de depuración.

Además de permitir ejecutar Appuntate en un emulador, Android Studio también permite depurar la aplicación en un dispositivo real, para hacer esto el dispositivo debe tener habilitadas las opciones de desarrollador y tener activada, o bien, la depuración a través de USB o la depuración inalámbrica.

Para instalar la aplicación a través de USB el dispositivo tiene que estar conectado al ordenador a través de un cable USB, para instalarlo de manera inalámbrica, tanto el ordenador como el dispositivo móvil tienen que estar conectados a la misma red Wi-Fi.

Ejecutando Appuntate en emuladores además de en un dispositivo real se ha comprobado el correcto funcionamiento de Appuntate en la plataforma **Android**, de entre todos los dispositivos emulados y físicos en los que se ha probado Appuntate, no se ha detectado ninguna incidencia, desde el primer momento el proyecto **Android** generado por **Capacitor** ha funcionado correctamente sin la necesidad de configuración adicional.

5.3.3 XCode

XCode es un IDE desarrollado por Apple que puede decirse que es el homologo a Android Studio para desarrollo de aplicaciones en **iOS**. El IDE es utilizado para el desarrollo de aplicaciones **iOS**, MacOS, iPadOS y el resto de los sistemas operativos de Apple como el utilizado en sus relojes

inteligentes llamado watchOS. En el contexto de Appuntate solo es interesante el desarrollo de aplicaciones **iOS**.

A diferencia de Android Studio, no existen alternativas a XCode para el desarrollo de aplicaciones **iOS**, aunque el código de la aplicación que se esté desarrollando pueda editarse desde otros IDEs o editores de texto, XCode es necesario para generar el paquete ".ipa" necesario para distribuir y lanzar una aplicación **iOS**, ya sea a través de la interfaz gráfica o de su línea de comandos. Más allá de que no exista una alternativa, XCode solo está disponible en MacOS, lo que implica que es necesario una maquina corriendo MacOS para poder desarrollar una aplicación para **iOS**.

El IDE, en su última versión, ofrece emuladores que permiten emular todos los dispositivos con todas las versiones de **iOS** que siguen recibiendo soporte por parte de Apple, para emular dispositivos o versiones más antiguas de **iOS** es necesario utilizar una versión anterior del IDE. En el contexto de Appuntate la versión utilizada de **Capacitor** soporta desde la versión 12 de **iOS** en adelante, así que no ha sido necesario utilizar una versión más antigua de XCode ya que, en este caso, la limitación queda establecida por **Capacitor** y no por XCode.

El uso que se le ha dado a XCode ha sido el mismo que a Android Studio, emular dispositivos para comprobar el correcto funcionamiento de Appuntate en la plataforma y generar el paquete ".ipa"

5.3.4 Google Chrome

Ya que Appuntate es una aplicación web, es necesario un navegador para ejecutar la aplicación y realizar tareas de depuración durante el desarrollo.

Google Chrome es un navegador multiplataforma desarrollado por Google lanzado en 2008. Chrome, además de ser veloz, ofrece herramientas avanzadas para desarrolladores y acceso a un "*market place*" de extensiones que permiten añadir funcionalidad al navegador de una manera muy parecida a las extensiones de Visual Studio Code que se han mencionado anteriormente.

Las herramientas enfocadas a los desarrolladores que incluye Chrome se llaman Chrome DevTools y estas diseñadas para agilizar el proceso de desarrollo de aplicaciones web ofreciendo una gran variedad de funcionalidades. Este conjunto de herramientas son la razón por la cual se ha utilizado Chrome como navegador principal para el desarrollo de Appuntate.

En primer lugar, una de las características que ofrece Google Chrome que más se han utilizado en el desarrollo de Appuntate es la posibilidad de simular dispositivos móviles desde el navegador. La simulación no funciona de la misma manera que un emulador ya que Google Chrome no es capaz de emular un dispositivo, lo que permite la simulación es mostrar cómo se vería la aplicación si se ejecutase en un dispositivo determinado, de esta manera podemos comprobar cómo se ve Appuntate en una gran variedad de dispositivos sin la necesidad de arrancar un emulador o instalar la aplicación en un dispositivo, lo cual consumiría muchísimo más tiempo que simplemente refrescar la página en el navegador.

Chrome ofrece la simulación de una gran variedad de dispositivos de varias plataformas, en concreto, para Appuntate, son interesantes los dispositivos que tienen como sistema operativo **Android** e **iOS**. Para simular un dispositivo desde Google Chrome hay que acceder a las

herramientas para desarrolladores pulsando F12 o accediendo desde el menú que se encuentra a la derecha de la barra de navegación. Una vez dentro de las herramientas para desarrolladores hay que hacer clic en el icono de dispositivo.

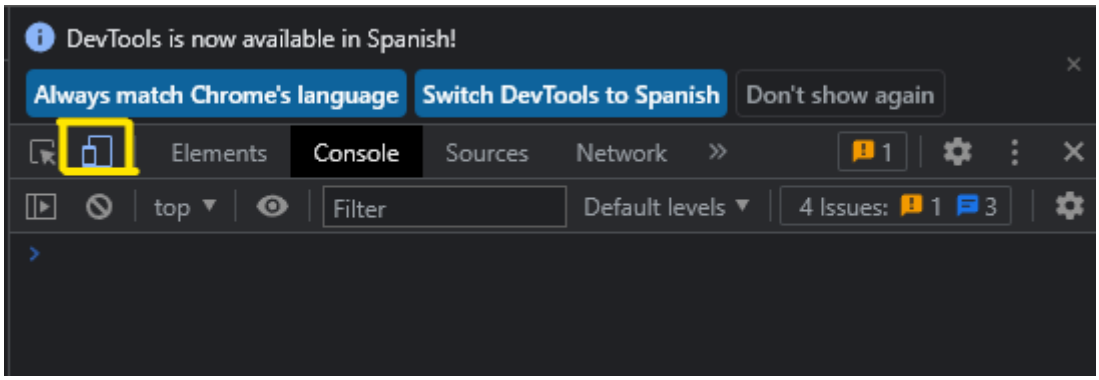

Ilustración 38 - Simulación de dispositivos en Chrome

Después de hacer clic sobre ese icono la vista de la página web cambia y aparece la barra de herramientas que se muestra en la ilustración 39.

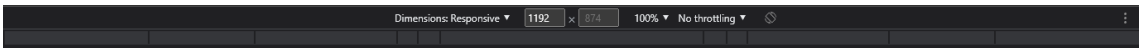

Ilustración 39 - Barra de herramientas en Chrome

Esta barra de herramientas nos permite cambiar el dispositivo a simular, las dimensiones de la pantalla, el escalado de la pantalla, la rotación de la pantalla, establecer limitaciones en cuanto al hardware del dispositivo y poner en modo offline la aplicación.

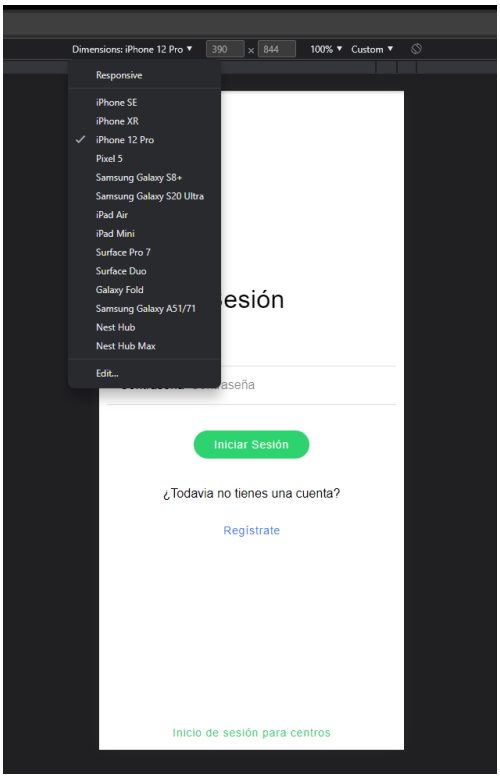

Ilustración 40 - Dispositivos disponibles para emular por defecto en Chrome

En la ilustración 40 se muestran los dispositivos disponibles por defecto, pero se puede editar esa lista añadiendo cualquiera de los dispositivos de la ilustración 41.

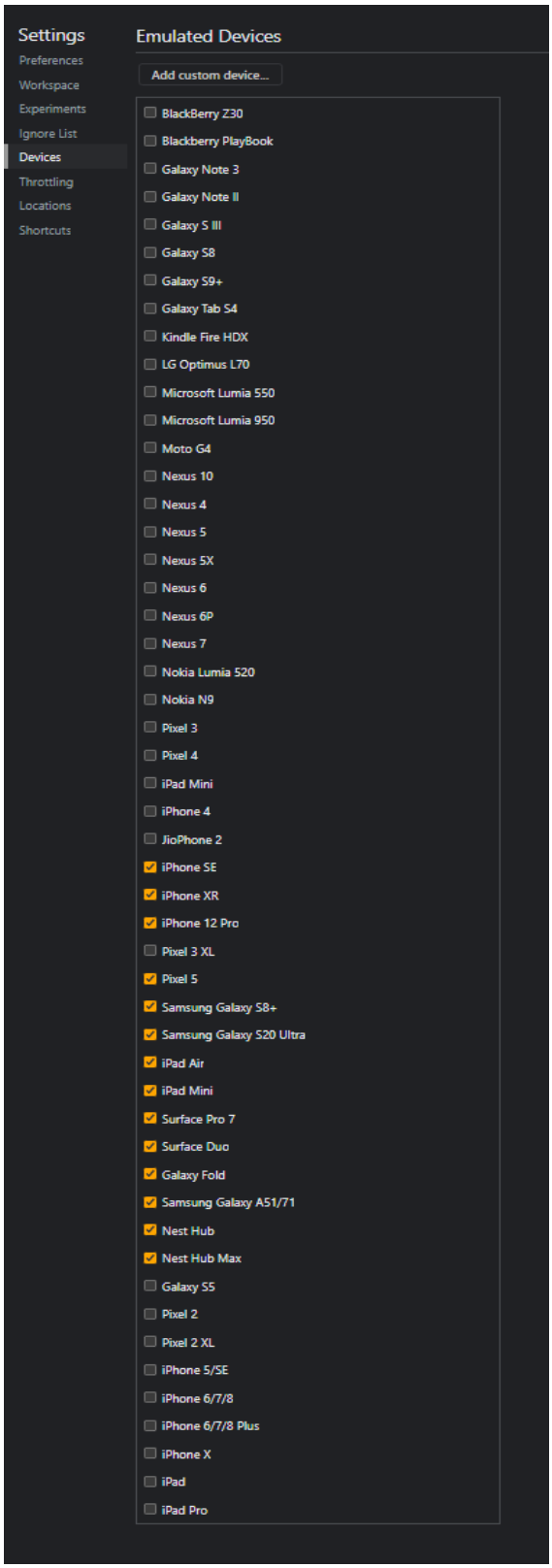

Ilustración 41 - Lista completa de dispositivos disponibles para emular en Chrome

Entre toda la selección de dispositivos también se encuentra la opción responsive, al seleccionar esta opción aparecen dos barras alrededor de la ventana que permiten ajustar el tamaño de la ventana arrastrándolas con el ratón, lo cual es particularmente útil cuando se están desarrollando vistas distintas para distintos tamaños de pantalla como en el caso de Appuntate ya que es posible

ver el cambio de la vista de un tamaño de una pantalla a otro simplemente arrastrando las barras con el ratón.

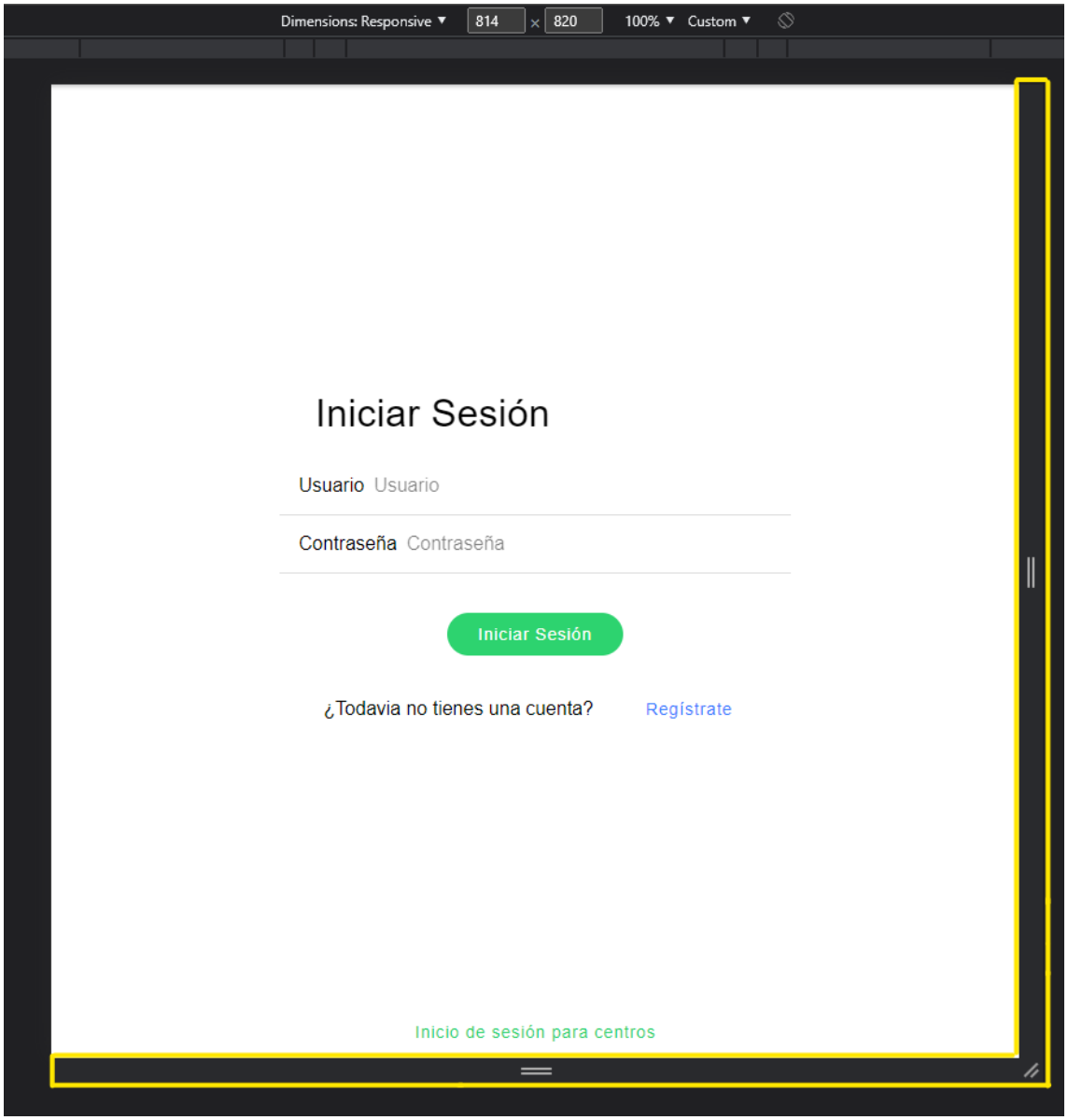

Ilustración 42 - Barras de redimensionamiento

Esta funcionalidad ha sido clave para el desarrollo de Appuntate ya que durante todo el proceso de desarrollo se ha comprobado que la interfaz de Appuntate se mostrase de manera correcta en dispositivos tanto **Android** como **iOS** con distintos tamaños de pantalla.

Además de la simulación de dispositivos, Google Chrome también ofrece herramientas de manipulación del DOM, una consola, un visor para los archivos que ha cargado la aplicación que permite editarlos y realizar tareas de depuración, un inspector de red, herramientas para medir el rendimiento de la aplicación, acceso al local storage y herramientas específicas para la depuración de **PWA**s. Todas ellas han sido utilizadas para el desarrollo de Appuntate y se detallan a continuación. Estas herramientas son accesibles desde las herramientas para desarrolladores.

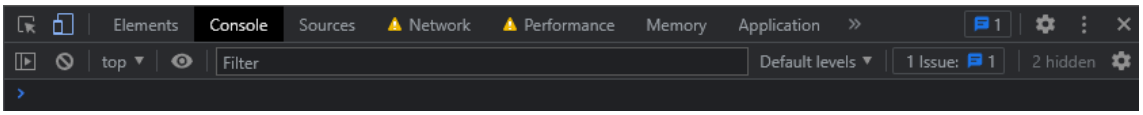

Ilustración 43 - Herramientas para desarrolladores en Chrome

- Herramientas para la manipulación del DOM
  - Chrome permite visualizar el DOM y editarlo en tiempo real, lo cual es muy útil cuando se están realizando tareas de maquetación ya que el desarrollador puede modificar el **HTML** y el **CSS** de la página directamente desde el navegador para poder visualizar cambios sin tener que editar el código fuente.

- Visor de archivos
  - Desde el visor de archivos el desarrollador puede seleccionar cualquier archivo que haya cargado la aplicación para visualizar y editarlo. Desde el editor se pueden establecer "*breackpoints*", marcas en el código que en el momento en el que el hilo de ejecución pasa por ellas, lo detienen, de esta manera el desarrollador puede ver en ese instante información que ofrece Google Chrome como el valor de las variables de ese archivo en ese momento y realizar acciones como reanudar el hilo de ejecución o ejecutar la siguiente instrucción o función.

- Inspector de red
  - El inspector de red permite ver toda la actividad de la red, lo cual resulta muy útil durante el desarrollo ya que facilita la detección de errores en cuanto al desarrollo de solicitudes al servidor. Desde el inspector se puede visualizar toda la información relacionada a una petición HTTP como el código de estado de la respuesta y los datos recibidos, esta información puede ser usada para comprobar si una petición a un "*endpoint*" de una API REST se está realizando de manera correcta.

- Herramientas para medir el rendimiento de la aplicación
  - Estas herramientas permiten analizar el rendimiento de la aplicación mientras se está ejecutando, permiten realizar una "grabación" de la aplicación mientras se está usando y proporcionan resultados de analíticas que muestran, por ejemplo, las FPS, franjas por segundo, de la aplicación, una métrica que indica si una página web se ejecuta de manera fluida o no. Además, también permiten simular el rendimiento de dispositivos con distintas CPUs y distintos tipos de red, esto permite realizar estas "grabaciones" y obtener las analíticas simulando que la aplicación se esté ejecutando en dispositivos con CPUs más o menos potentes y redes más o menos rápidas.

- Herramientas específicas para la depuración de **PWA**s

- Permiten modificar el archivo “manifest” directamente desde el navegador y simular eventos como que el usuario instale la aplicación para comprobar el comportamiento de esta en ese caso o eventos relacionados con el “Service worker” como insertar notificaciones push o simular que se encuentra en modo offline.

- Acceso al local Storage
  - Google Chrome permite acceder al local storage de la aplicación. El local storage es una propiedad de las aplicaciones web que permite almacenar datos en forma de clave valor y es utilizado para guardar datos que tienen que prevalecer aunque se cierre el navegador. Se suele utilizar, por ejemplo, para almacenar tokens de autenticación para mantener la sesión del usuario. Desde las herramientas para desarrolladores es posible añadir entradas al local storage, modificarlas o eliminarlas.

6. Control de versiones

El control de versiones es una pieza clave en el desarrollo de software, tanto por razones de seguridad como de control de calidad. Es muy importante almacenar el código en repositorios remotos y locales para evitar la pérdida de este.

Para Appuntate se ha optado por GitHub, entre el resto de los servicios que ofrece, es una herramienta basada en Git que permite la creación de repositorios remotos tanto privados como públicos.

Además de poder utilizar la línea de comandos de Git para realizar las acciones habituales como un "pull" o un "fetch", GitHub ofrece una interfaz gráfica sencilla e intuitiva llamada GitHub Desktop que permite realizar todas las acciones de Git y desde la cual se puede visualizar el estado actual del repositorio, visualizar archivos y comparar versiones.

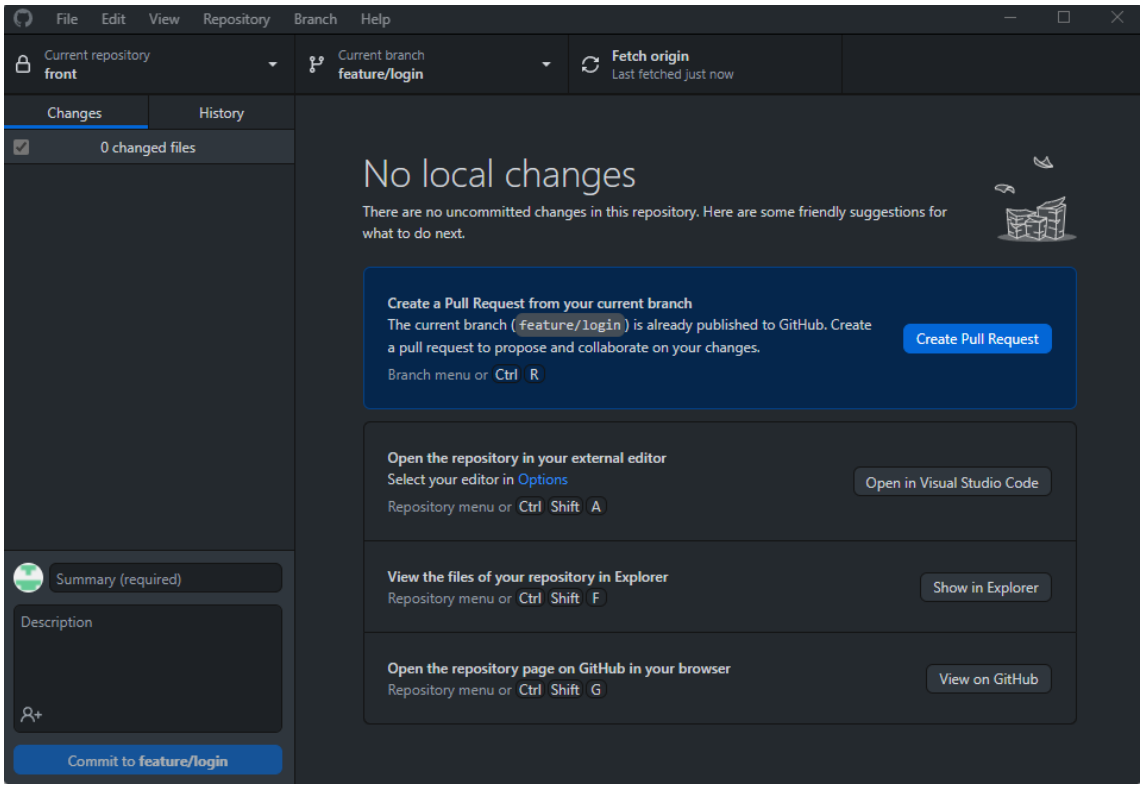

Ilustración 44 - GitHub Desktop

Durante el desarrollo de Appuntate se ha seguido un flujo de trabajo llamado Gitflow, este flujo consiste en crear tres ramas principales, una rama llamada master, otra llamada develop y otra llamada release.

En primer lugar, la rama master es la rama que contiene el código utilizado en la fase de producción, es decir la etapa en la que la aplicación va a ser lanzada, no se debe realizar ningún tipo de desarrollo sobre esta rama y el código se introduce en ella una vez se vaya a lanzar una nueva versión de la aplicación.

En segundo lugar, la rama develop es la rama en la cual se incluye el desarrollo realizado para nuevas funcionalidades, sin embargo, no se debe desarrollar sobre esta rama si no que se debe realizar el desarrollo en otro tipo de ramas e incluirlo en esta mediante el uso de "pull requests".

Las "pull request" son solicitudes para introducir un trozo de código en una rama determinada, estas solicitudes se asignan a un desarrollador distinto al que ha escrito el código que se quiere introducir y es aprobada o denegada, en el caso de que la solicitud sea aprobada, la rama que contenía el desarrollo se elimina y los cambios quedan en la rama develop, en el caso de que la solicitud sea denegada, el revisor puede incluir comentarios para que el desarrollador realice las modificaciones permitentes antes de que ese código se incluya en la rama develop.

El tipo de ramas mencionadas desde las cuales se realiza el desarrollo de una característica o funcionalidad determinada para después ser incluido en la rama develop se llaman ramas feature. Las ramas feature son ramas temporales que son eliminadas una vez el desarrollo realizado se incluye en develop y su nombre sigue el formato feature/nueva-funcionalidad.

En tercer lugar, la rama release es una rama en la cual se incluye el código contenido en la rama develop para distribuir una versión de la aplicación, estas versiones se suelen utilizar para ser distribuidas a las personas responsables de realizar pruebas sobre la aplicación desarrollada.

En el caso de que un fallo llegue a la rama master, existe un último tipo de rama temporal llamado hotfix, estas ramas se crean a partir de la rama master y se utilizan para corregir el fallo en cuestión, una vez resuelta la incidencia, se incluye la solución en la rama master y la rama hotfix se elimina. El nombre de estas ramas sigue el siguiente formato hotfix/error-a-solucionar.

Durante el desarrollo de Appuntate, al contar con únicamente un desarrollador, no ha sido posible realizar "pull requests" de manera apropiada ya que el código debe revisarse por una persona distinta a la que lo ha escrito, sin embargo, en cuanto el equipo de Appuntate crezca, se realizaran estas solicitudes de manera correcta.

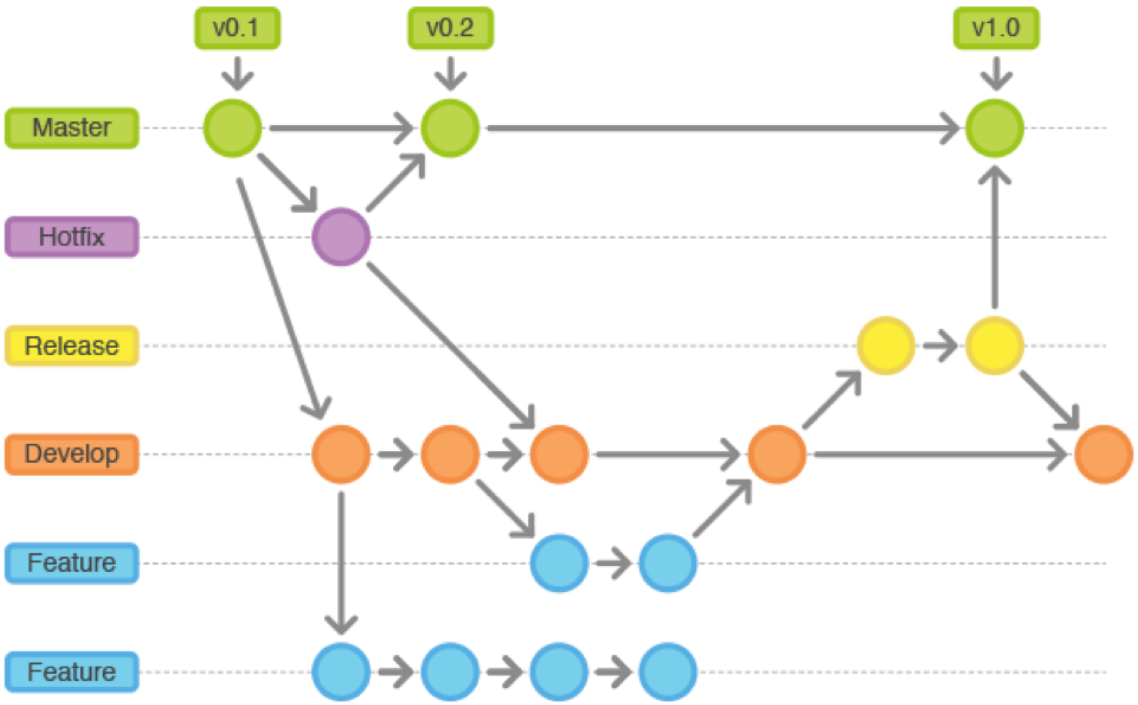

Ilustración 45 - Gitflow

7. Estructura y arquitectura del proyecto

En este apartado se detallan todas las pautas tomadas en cuanto a la arquitectura y estructura del front-end de Appuntate, se detalla tanto la arquitectura del front-end de Appuntate como la estructura de carpetas, que ha sido diseñada para conseguir que el proyecto sea escalable y mantenible.

7.1 Arquitectura

La arquitectura de Appuntate está formada por una aplicación **Angular** que hace uso de la librería de componentes de **Ionic** y plugins proporcionados por **Capacitor** que es incrustada en un web-view, o vista web, dentro de una aplicación nativa de **Android** e **iOS**.

La aplicación **Angular** está formada por módulos que establecen el contexto de un apartado de la aplicación, una funcionalidad o un flujo concreto, estos módulos declaran las piezas utilizadas dentro de su entorno.

Existe un módulo raíz que es utilizado para arrancar la aplicación y una serie de módulos dedicados a distintas funcionalidades, apartados y flujos, estos módulos declaran las distintas vistas, componentes y servicios asociados a sí mismos, además de declarar estas piezas, también son capaces de importarlas y exportarlas desde otros módulos, de esta manera, por ejemplo, un módulo puede hacer uso de un componente declarado por otro, si el segundo lo exporta.

Las vistas son componentes que cuentan con su propia lógica y hacen uso de otros componentes para conformar interfaces de usuario. La lógica y las plantillas de los componentes se enlazan mediante metadatos y se utilizan directivas para manipularlas de manera dinámica.

Los servicios son inyectados en los componentes mediante inyección de dependencias y son utilizados para compartir datos y funcionalidad entre componentes.

Para navegar entre vistas se utiliza el módulo de enrutamiento proporcionado por **Angular**, se utilizan módulos que importan este módulo para crear las rutas, existe un módulo de enrutamiento para el módulo raíz y un módulo de enrutamiento para cada módulo que tenga varias vistas. Appuntate hace uso del Lazy Loading, una estrategia de carga de módulos para optimizar el rendimiento de la aplicación.

En la ilustración 46 se muestran los componentes básicos de la arquitectura de la aplicación **Angular**.

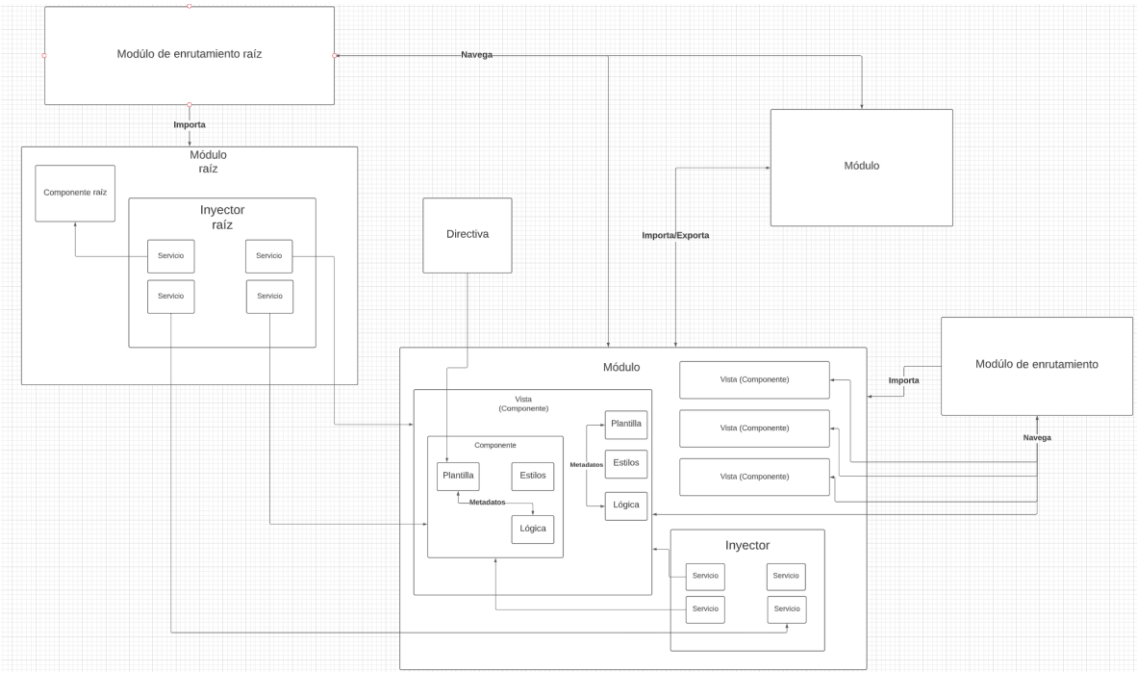

Ilustración 46 - Arquitectura de Angular en Appuntate

Para acceder a funcionalidades nativas de cada plataforma desde **Angular**, se hace uso de plugins proporcionados por **Capacitor** de la manera en la que se ha mencionado en el apartado de **Capacitor**.

La aplicación **Angular** se incrusta en un WebView a través de **Capacitor** y se genera una **PWA** mediante la CLI de **Angular** como se muestra en la ilustración 47.

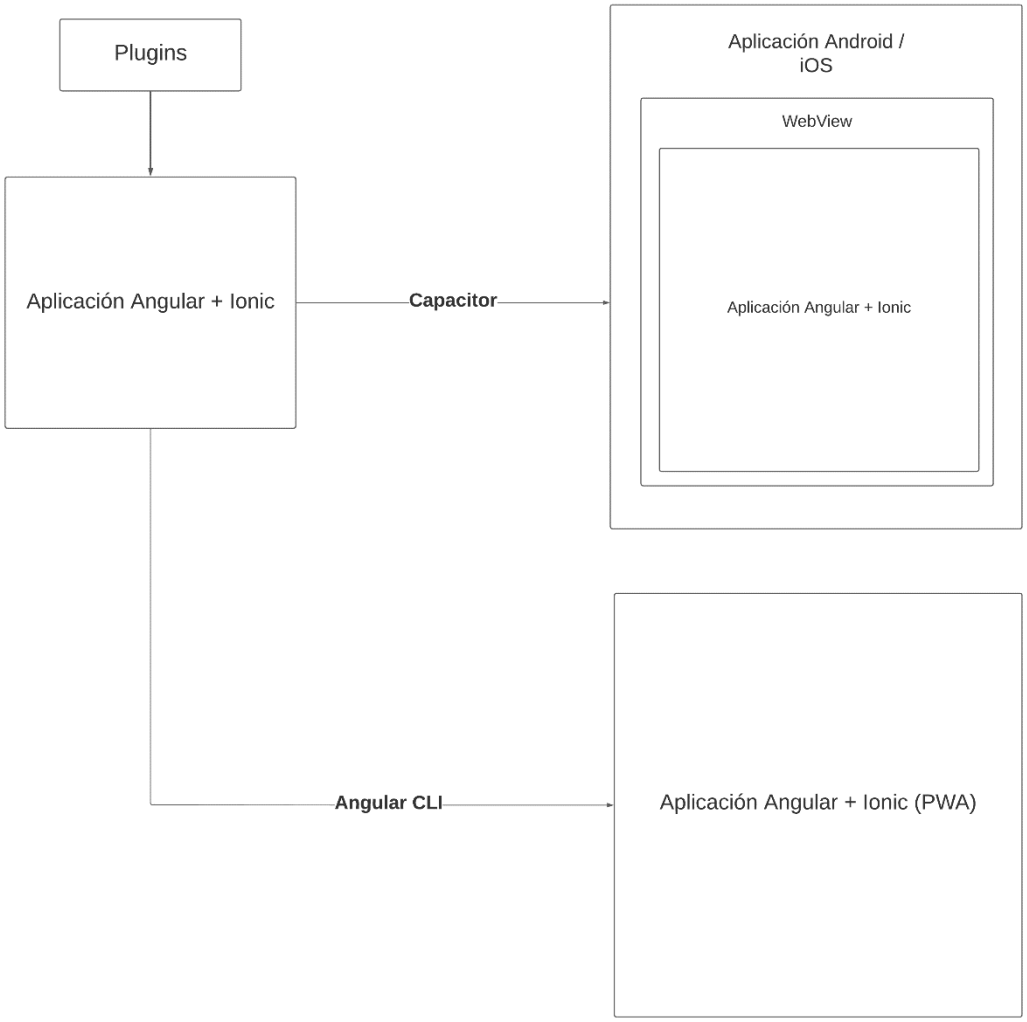

Ilustración 47 - Arquitectura de Appuntate

En los siguientes apartados, además de exponer la estructura de carpetas, también se describen de manera extensa todos los términos mencionados en este apartado que no han sido detallados.

7.2 Estructura de carpetas

En cuanto al diseño de la estructura de carpetas de Appuntate, se ha recurrido a la "***Angular** Coding Style Guide*", una guía que proporciona el equipo de Google que está detrás del desarrollo de **Angular** donde proporcionan recomendaciones y guías sobre cuales son y como usar buenas prácticas en el desarrollo de aplicaciones **Angular**.

Esta guía indica que, para seguir buenas prácticas, la estructura de carpetas de un proyecto **Angular** debe seguir los principios LIFT.

Los principios LIFT son los siguientes:

- *Locating our code is easy* (localizar nuestro código es fácil)
    - El código debe de ser fácil de localizar, es decir, la estructura de carpetas y archivos no debe ser un impedimento para poder encontrar un archivo. Los nombres de las carpetas deben ser descriptivos y coherentes con los archivos que contienen.

- *Identify code at a glance* (identificar el código en un vistazo)
    - Debe ser obvio que tipo de código contiene cada archivo, en **Angular** esto se consigue siguiendo la convención de sufijos para los nombres de los archivos, el nombre de los archivos debe contener un punto seguido del tipo de componente que contiene antes de la extensión del archivo, el patrón es el siguiente, "nombre-del-archivo.tipo.extension", por ejemplo:
        - "admin-calendar.component.ts"
        - "authentication.guard.ts"
        - "authentication.service.ts"

- *Flat structure as long as we can* (la estructura de archivos tiene que ser todo lo plana que sea posible)
    - La estructura de carpetas debe ser todo lo plana que sea posible, añadiendo carpetas solo cuando sea necesario, es decir, se debe considerar la creación de subcarpetas cuando el número de archivos que contenga ronde alrededor de los 10 archivos y solo si la creación de la carpeta aporta algún tipo de valor.

- *Try to stay DRY* (Intentar No Repetirnos)
    - Este principio hace referencia a que debemos intentar no declarar de manera explícita cosas que se pueden inferir de manera implícita, aplica al nombre de los archivos, por ejemplo, queda implícito que un archivo con extensión .scss contiene estilos, así que no es necesario que el nombre del archivo contenga la palabra "estilos" o "styles", un archivo con el siguiente nombre "global-styles.scss" no sigue el principio DRY mientras que un archivo con nombre "global.scss" sí que lo hace.

La estructura de carpetas de Appuntate se diseñó siguiendo estos principios y es la que se muestra en la ilustración 48.

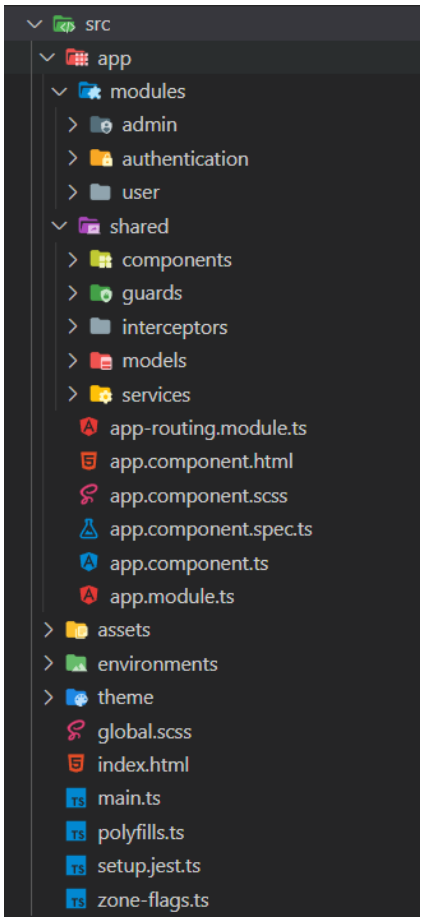

Ilustración 48 - Estructura de carpetas

A un primer nivel encontramos las siguientes carpetas:

- app
  - Contiene todo el código Fuente de la aplicación
- assets
  - Contiene las imágenes que utiliza la aplicación
- environments
  - En esta carpeta podemos encontrar dos archivos, un archivo para producción y otro para desarrollo. Estos archivos contienen una constante con la URL a la que apuntar para realizar llamadas HTTP, durante el desarrollo se apunta a un puerto local y en producción se apunta a la URL donde esté desplegado el back-end de la aplicación.
- theme
  - Esta carpeta contiene el archivo de estilos con todas las variables que se han utilizado en Appuntate

En la carpeta app se encuentra el código fuente de la aplicación y podemos encontrar dentro de esta carpeta 6 archivos, estos 6 archivos corresponden al módulo principal de la aplicación. El resto del código está separado en dos carpetas principales, la carpeta modules y la carpeta shared, la carpeta shared contiene todos los elementos del proyecto que son compartidos por dos o más módulos **Angular** y en la carpeta modules se encuentran todos los módulos **Angular** de la aplicación.

La carpeta shared contiene componentes, guardas, interceptores HTTP, modelos y servicios. Como se ha mencionado en el párrafo anterior, el criterio para estar dentro de esta carpeta es que el elemento se utilice en más de un módulo.

En la carpeta modules se encuentran los distintos módulos **Angular** con los que cuenta Appuntate y están divididos en tres carpetas, user, para los módulos correspondientes a la parte de la aplicación dedicado a los usuarios de las instalaciones deportivas, admin, para los módulos correspondientes al apartado de administración y authentication, para el módulo correspondiente al inicio de sesión.

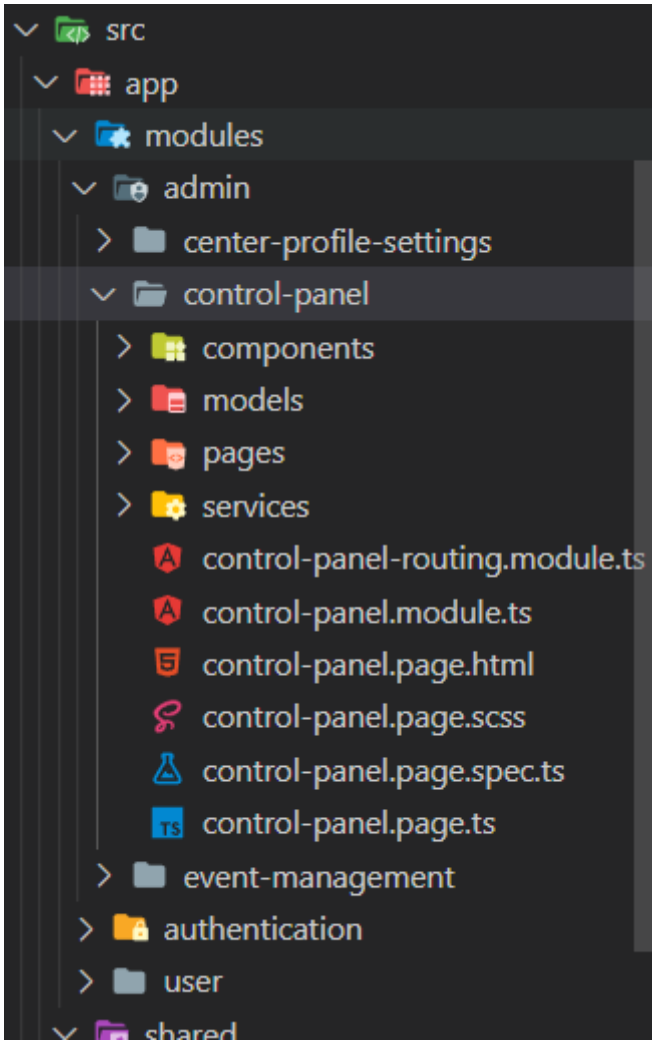

Ilustración 49 - Carpeta modules

Cada una de las carpetas dedicadas a un módulo puede contener dentro, además de los archivos correspondientes a ese modulo, carpetas que contengan componentes, servicios, páginas y modelos que, a diferencia de los que se encuentran dentro de la carpeta shared, solo se utilicen dentro de ese modulo.

8. Implementación

En los siguientes apartados se trata con detalle que son y con cuales cuenta Appuntate, los distintos componentes que proporciona **Angular**.

Cada uno de estos elementos que utilizados en conjunto constituyen un proyecto **Angular** son clases de **TypeScript** a las que se les añaden decoradores para hacer entender a **Angular** cual es la función del archivo.

8.1 Módulos

Appuntate cuenta con diversos módulos, divididos en tres carpetas como se ha mencionado en el apartado anterior. En las siguientes ilustraciones se muestran los módulos que contienen cada una de las tres carpetas.

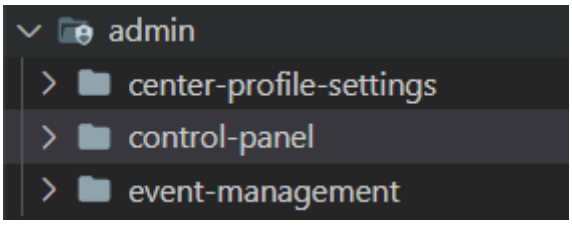

Ilustración 50 - Módulos de administración

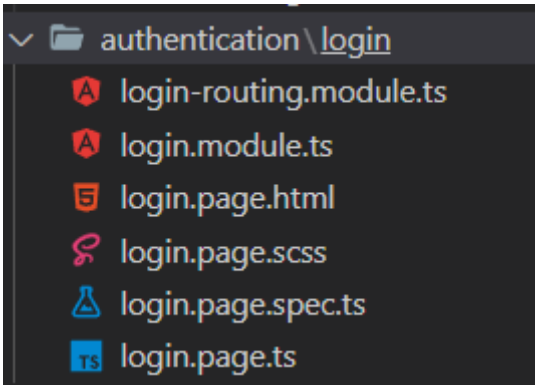

Ilustración 51 - Módulos de autenticación

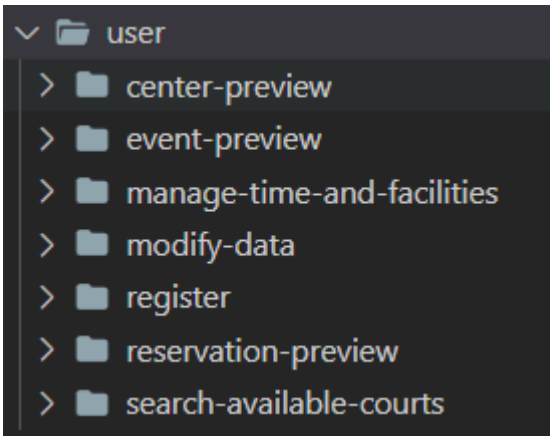

Ilustración 52 - Módulos de usuario

Los módulos se utilizan para encapsular bloques de código correspondientes a una cierta funcionalidad o un cierto flujo de la aplicación.

Los módulos se definen con el decorador @NgModule y dentro de este decorador hay que incluir un objeto que incluya las propiedades que definen al módulo en cuestión, las propiedades más importantes que hay que incluir en este objeto son las siguientes.

- declarations
  - Dentro de esta propiedad se incluyen los componentes, pipes y directivas que pertenecen al módulo.

- exports
  - Esta propiedad incluye cualquier elemento que pertenezca al módulo, es decir, que este dentro del atributo declarations que se quiera usar desde otros módulos

- imports
  - Dentro de esta propiedad se incluyen los módulos de los cuales se quieran usar elementos que tengan dentro del atributo exports. Incluir un módulo dentro de este atributo permite utilizar en ese mismo modulo los elementos que exporte el módulo importado, si un componente o un módulo que se quiera utilizar no se encuentra declarado dentro de este atributo, no podrá ser usado desde ese módulo.

- providers
  - En este atributo se incluyen los creadores de servicios que se quieran asociar a ese modulo, si un servicio está dentro del atributo providers de un módulo, solo será accesible desde otros módulos si estos importan ese modulo.

- bootstrap
    - Dentro de este atributo se declara el componente principal de la aplicación, esto significa que solo el módulo raíz debe contener este atributo.

Appuntate, como todas las aplicaciones **Angular**, tiene un módulo raíz, el cual es necesario y debe existir en cualquier aplicación **Angular** para que funcione. El módulo raíz suele llamarse AppModule y ese es el caso en Appuntate. El AppModule se encuentra dentro de la carpeta app y es el módulo desde el cual se arranca la aplicación declarando como componente principal de la misma el AppComponent que se encuentra en el mismo directorio.

app.module.ts M

src > app > app.module.ts > AppModule

```
1  import { NgModule } from '@angular/core';
2  import { BrowserModule } from '@angular/platform-browser';
3  import { RouteReuseStrategy } from '@angular/router';
4  import { CommonModule } from '@angular/common';
5  import { IonicModule, IonicRouteStrategy } from '@ionic/angular';
6  import { AppComponent } from './app.component';
7  import { AppRoutingModule } from './app-routing.module';
8  import { RouterModule } from '@angular/router';
9  import { HttpClientModule } from '@angular/common/http';
10 import { BrowserAnimationsModule } from '@angular/platform-browser/animations';
11 import { ComponentsModule } from './shared/components/components.module';
12
13 @NgModule({
14   declarations: [
15     AppComponent
16   ],
17   entryComponents: [],
18   imports: [
19     BrowserModule,
20     ComponentsModule,
21     IonicModule.forRoot({
22       mode: 'md'
23   }),
24     AppRoutingModule,
25     CommonModule,
26     RouterModule,
27     HttpClientModule,
28     ComponentsModule,
29     BrowserModule,
30     BrowserAnimationsModule
31   ],
32   providers: [{ provide: RouteReuseStrategy, useClass: IonicRouteStrategy }],
33   bootstrap: [AppComponent],
34 })
35 export class AppModule {}
36
```

Ilustración 53 - App Module

En la ilustración 53 podemos ver que dentro del atributo imports el AppModule de Appuntate importa el AppRoutingModule, este módulo es un módulo dedicado el enrutamiento de la aplicación, en él se definen las distintas rutas a las cuales se puede acceder desde ese modulo, cada una de esas rutas hace referencia a otro módulo para indicarle a **Angular** que modulo tiene que cargar cuando se accede a esa ruta. El AppRoutingModule no es el único modulo dedicado

al enrutamiento, otros paquetes de funcionalidad como el módulo control-panel también cuentan con un módulo dedicado al enrutamiento para cargar las distintas vistas.

control-panel-routing.module.ts

src > app > modules > admin > control-panel > control-panel-routing.module.ts > routes > loadChildren

```typescript
import { NgModule } from '@angular/core';
import { Routes, RouterModule } from '@angular/router';

import { ControlPanelPage } from './control-panel.page';

const routes: Routes = [
  {
    path: '',
    component: ControlPanelPage
  },
  {
    path: 'add-court',
    loadChildren: () => import('./pages/add-court/add-court.module').then( m => m.AddCourtPageModule)
  },
  {
    path: 'court/:id',
    loadChildren: () => import('./pages/court/court.module').then( m => m.CourtPageModule)
  }
];

@NgModule({
  imports: [RouterModule.forChild(routes)],
  exports: [RouterModule],
})
export class ControlPanelPageRoutingModule {}

```

Ilustración 54 - Control Panel Routing Module

```
  },
  {
    path: 'register',
    loadChildren: () =>
      import('./modules/user/register/register.module').then(
        (m) => m.RegisterPageModule
      ),
  },
  {
    path: 'control-panel',
    canActivate:[AuthenticationGuard],
    loadChildren: () =>
      import('./modules/admin/control-panel/control-panel.module').then(
        (m) => m.ControlPanelPageModule
      ),
  },
  {
    path: 'modify-data',
    canActivate: [NoDirectAccessGuard],
    loadChildren: () =>
      import('./modules/user/modify-data/modify-data.module').then(
        (m) => m.ModifyDataPageModule
      ),
  },
  {
    path: 'center-preview',
    canActivate: [NoDirectAccessGuard],
    loadChildren: () =>
      import('./modules/user/center-preview/center-preview.module').then(
        (m) => m.CenterPreviewPageModule
      ),
  },
  {
    path: 'event-preview',
    canActivate: [NoDirectAccessGuard],
    loadChildren: () =>
      import('./modules/user/event-preview/event-preview.module').then(
        (m) => m.EventPreviewPageModule
      ),
  },
  {
    path: 'event-management',
    canActivate: [NoDirectAccessGuard],
    loadChildren: () =>
      import('./modules/admin/event-management/event-management.module').then(
        (m) => m.EventManagementPageModule
      ),
  },
  {
    path: 'center-profile-settings',
    canActivate: [NoDirectAccessGuard],
    loadChildren: () =>
      import(
        './modules/admin/center-profile-settings/center-profile-settings.module'
      ).then((m) => m.CenterProfileSettingsPageModule),
  },
];

@NgModule({
  imports: [
    RouterModule.forRoot(routes, { preloadingStrategy: PreloadAllModules }),
  ],
  exports: [RouterModule],
})
export class AppRoutingModule {}
```

Ilustración 55 - App Routing Module

Dentro de la constante routes se definen todas las rutas y el módulo que ha de ser cargado desde cada una de ellas, sin embargo, para que el enrutamiento en **Angular** funcione no es necesario definir el módulo que ha de ser cargado al acceder a una ruta determinada ya que es posible que aplicaciones pequeñas no cuenten con más de un módulo, en ese caso sería suficiente con declarar la ruta como en la ilustración 56 [27].

```
const routes: Routes = [
  { path: 'heroes', component: HeroesComponent }
];
```

Ilustración 56 - Ruta en Angular

Appuntate cuenta con múltiples módulos, hace uso del Lazy Loading y por eso está declarado el módulo que ha de ser cargado al acceder a una ruta determinada.

El Lazy Loading es una característica con la que cuenta **Angular** diseñada para aplicaciones grandes. En una aplicación **Angular** que no haga uso del Lazy Loading todos los módulos de la aplicación son cargados cuando se inicia la aplicación, lo cual no supone ningún problema cuando la aplicación solo cuenta con uno o varios módulos, pero puede convertirse en uno cuando la aplicación cuenta con un número elevado de ellos ya que puede resultar en tiempos de carga muy elevados. El Lazy Loading permite que los módulos solo sean cargados cuando sea necesario, esto significa que hasta que no se accede a una ruta el módulo asociado a esa ruta no se carga, reduciendo el tiempo de inicio de la aplicación.

Esta característica es un arma de doble filo ya que hacer uso de ella reduce el tiempo de inicio de la aplicación pero aumenta el tiempo de carga cuando se accede a un módulo que todavía no ha sido cargado, teniendo esto en cuenta, en aplicaciones con un tamaño considerable, como es el caso de Appuntate, es preferible reducir el tiempo de inicio de la aplicación a costa de aumentar el tiempo de carga durante la navegación ya que el usuario probablemente no vaya a necesitar acceder a todos los módulos de la aplicación, lo que resulta en que el tiempo ahorrado en la carga inicial de la aplicación sea muy superior al perdido en la navegación.

8.2 Servicios

Los servicios en **Angular** se declaran utilizando el decorador @Injectable, a este decorador se le pasa un objeto con una única propiedad llamada providedIn que indica donde va a poder ser inyectado el servicio. El termino inyectado viene de la inyección de dependencias, la inyección de dependencias es una técnica que permite introducir una clase sobre la cual depende otra clase en la clase dependiente sin la necesidad de instanciarla, por ejemplo, si una clase depende de cierta funcionalidad que hay implementada en otra clase, la inyección de dependencias permite introducir esa clase en la clase dependiente sin la necesidad de que la clase que depende de la otra la instancie con el operador “new”. Para conseguir esto, en **Angular**, existen los inyectores, los inyectores se encargan de crear las instancias de las clases sobre las cuales dependen otras clases e inyectarlas en las clases dependientes. **Angular** proporciona inyectores durante el arranque de la aplicación así que no es necesario crear inyectores en **Angular**, **Angular** determina qué dependencias ha de inyectar comprobando el constructor de la clase, de esta manera, se deben declarar en él las dependencias.

```
constructor(
  private searchAvailableCourtsService: SearchAvailableCourtsService,
  private modalController: ModalController,
  private centerService: CenterService,
  private router: Router
) { }
```

Ilustración 57 - Dependencias a inyectar declaradas en el constructor

Para que un servicio solo pueda ser utilizado desde un módulo es necesario declarar ese módulo en la propiedad providedIn o dejar el decorador vacío y declarar el servicio dentro del módulo en el que se quiere usar en la propiedad providers del módulo, de esta manera, cada vez que el servicio aparezca dentro de la propiedad providers de un módulo, se creará una instancia de ese servicio que será solo accesible por los componentes de ese módulo, haciendo posible que existan tantas instancias de ese servicio como módulos que lo incluyan dentro de su propiedad providers. El hecho de que todos los componentes de un módulo puedan acceder a una misma instancia de un servicio los hace ideales para la comunicación entre componentes ya que si un componente de ese módulo cambia el valor de una variable y un componente del mismo módulo la lee después de que haya sido cambiada, el segundo componente será capaz de ver los cambios del primero ya que están accediendo a la misma instancia del servicio.

Los servicios, a través del atributo providedIn en el objeto que se introduce en el decorador, también pueden adoptar el patrón singleton, esto se consigue dándole el valor 'root' al atributo, de esta manera, a lo largo de toda la aplicación, solo se creará una instancia del servicio haciendo posible que componentes de distintos módulos accedan a esa misma instancia, permitiendo el intercambio de datos entre componentes de distintos módulos. Esto es útil en las situaciones en las que se quiere hacer accesible cierta funcionalidad o ciertos datos en toda la aplicación, lo que reduce enormemente la duplicidad de código y facilita la comunicación entre componentes. La manera en la que reduce la duplicidad de código es la siguiente, si existe cierta funcionalidad que tiene que ser accesible desde componentes declarados en distintos módulos, es posible extraer esa funcionalidad a un servicio y acceder a ella a través del servicio en lugar de implementarla múltiples veces en los distintos componentes.

En Appuntate todos los servicios que se encuentran dentro de la carpeta shared siguen el patrón singleton ya que tienen que ser accesibles desde múltiples módulos de la aplicación. El servicio de autenticación es un buen ejemplo de cómo utilizar los servicios para compartir datos entre componentes, ya sean de un mismo módulo o de distintos módulos, ya que, aunque en el caso del servicio de autenticación de Appuntate el servicio es accesible desde múltiples módulos, para compartir datos entre componentes de un mismo módulo, el código necesario es el mismo, lo único que habría que modificar es la propiedad providedIn del servicio o la propiedad providers del módulo.

En cuanto a evitar duplicidades en el código, el servicio de fechas de Appuntate es un buen ejemplo. El back-end de Appuntate almacena las fechas en un formato determinado, dd-mm-yyyy, así que en todas las llamadas en las cuales haya que enviar una fecha al servidor o se vaya a recibir una fecha del servidor hay que realizar una conversión entre el formato con el que trata las fechas el back-end de Appuntate y el objeto Date de **JavaScript**, para evitar que esa lógica se implemente en cada pantalla que realice una llamada de ese tipo, la lógica de conversión, o "parseo", se ha extraído al servicio de fechas "date.service.ts" que se encuentra en la carpeta shared, esto permite

acceder a la funcionalidad del servicio inyectándolo en el constructor de la clase desde la cual se quiera acceder a la funcionalidad.

```
Complexity is 3 Everything is cool!
@Injectable({
  providedIn: 'root'
})
export class DateService {

  constructor() { }

  toDateFormat(value: Date): string{
    const dateTime = value.toISOString();
    const index = dateTime.indexOf('T');
    const date = dateTime.substring( 0, index);
    const auxArray = date.split('-',3);
    return auxArray[2] + '-' + auxArray[1] + '-' + auxArray[0];
  }

  toHourFormat(value: Date): string{
    const dateTime = value.toISOString();
    const i = dateTime.indexOf('T');
    const hour = dateTime.substring(i + 1 ,i + 6);
    return hour;
  }

  Complexity is 3 Everything is cool!
  serverToISOFormat(date: string, hour: string){
    const dateArray = date.split('-');
    let day = dateArray[2];
    if(day.length < 2){
      day = '0' + day;
      date = dateArray[0] + '-' + dateArray[1] + '-' + day;
    }
    return date + 'T' + hour + ':00';
  }
}
```

Ilustración 58 - Date Service

```
start: this.dateService.serverToISOFormat(item.date, item.startHour),
```

Ilustración 59 - Ejemplo de uso del servicio de fechas

8.3 Interfaces

Las interfaces no son una característica de **Angular,** sino de **TypeScript** como se ha detallado en el apartado sobre **TypeScript**.

En proyectos **Angular** y en consecuencia en Appuntate, las interfaces de **TypeScript** se utilizan para modelar los datos que se envían y reciben desde y hacia el servidor mediante HTTP, esto no resta de que también se puedan utilizar fuera de este contexto sin incurrir en malas prácticas, no obstante, el uso más común de las interfaces **TypeScript** en proyectos **Angular** es el de modelar

los datos mencionados. Las interfaces suelen ubicarse dentro de una carpeta con el nombre "models" ya que su trabajo es modelar los datos que se van a enviar y recibir.

En Appuntate existen dos tipos de interfaces, las interfaces que se utilizan en un ámbito global, es decir, no solo se utilizan en un módulo de la aplicación, sino que se utilizan en dos o más módulos, y las interfaces que son propias de un módulo o componente, las primeras están ubicadas en la carpeta "shared" y las segundas están ubicadas dentro de las carpetas correspondientes al módulo o componente al que pertenecen y en el que son utilizadas.

Dentro de las capetas "models" correspondiente a un módulo o componente en concreto existen interfaces utilizadas para modelar objetos que ayuden a mostrar los datos por pantalla o que modelan peticiones y respuestas a servicios que solo se utilizan en ese modulo o pantalla. En la carpeta shared se encuentran los archivos que contienen las interfaces que modelan datos que se utilizan en múltiples módulos, por ejemplo, la interfaz que modela los deportes se utiliza tanto en la mayoría de los módulos de la carpeta user como de la carpeta admin.

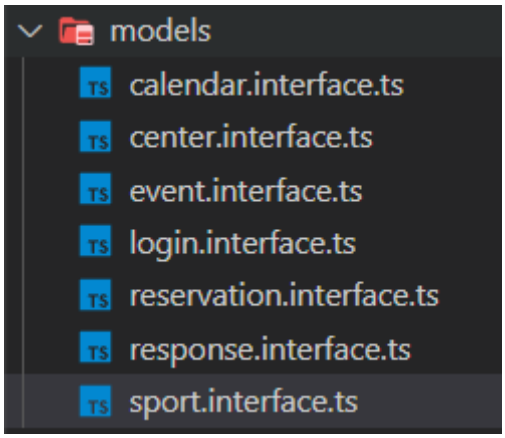

Ilustración 60 - Interfaces globales

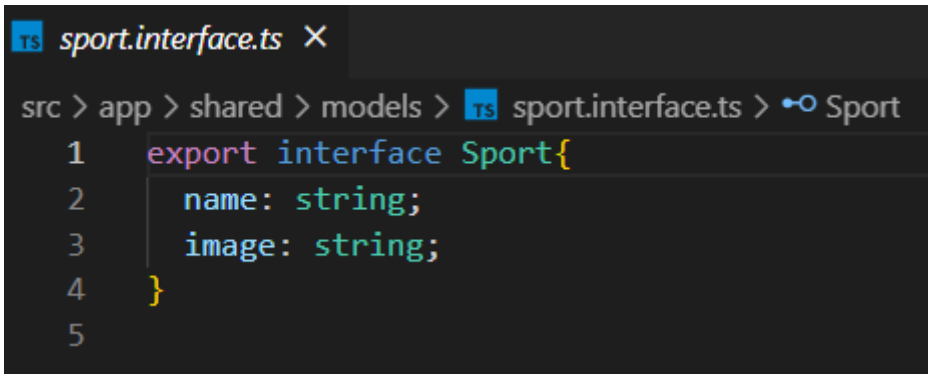

Ilustración 61 - Interfaz Sport

El uso de interfaces es necesario si se pretende maximizar la potencia que ofrece **TypeScript**. Al utilizar interfaces, no solo podemos detectar errores en el momento en que estamos escribiendo el código como se ha mencionado en el apartado sobre **TypeScript**, sino que también podemos acceder a herramientas del IDE o del editor de texto, en este caso Visual Studio Code, que muestran sugerencias de autocompletado basándose en el tipo de un objeto, lo cual agiliza el desarrollo.

8.4 Guardas

Las guardas son un mecanismo para proteger la navegación dentro de la aplicación, permiten bloquear la navegación hacia rutas determinadas dependiendo en unas condiciones establecidas.

Las guardas, al igual que los servicios, se definen en **Angular** como clases de **TypeScript** con el decorador @Injectable y en Appuntate todas cuentan con el atributo providedIn: 'root' para que sean accesibles desde toda la aplicación.

Estas clases además de contar con el decorador @Injectable, también implementan la interfaz "CanActivate", todas las clases que implementan la interfaz "CanActivate" cuentan con un método llamado "canActivate", dentro de este método es donde se introduce la lógica para determinar si se puede acceder a la ruta deseada o no.

Para hacer uso de las guardas se declaran dentro de la propiedad "canActivate" de las rutas en los módulos de enrutamiento.

En el caso de Appuntate las guardas son realmente importantes ya que al ser una aplicación donde se requiere un registro previo, lo que implica un inicio de sesión, el usuario no debe ser capaz de acceder a determinadas rutas si no ha iniciado sesión, además de esto, tanto el apartado de administración de los centros como el apartado ofrecido a los usuarios de las instalaciones deportivas están incluidos en una sola aplicación, lo que significa que, si no se establece un protocolo de seguridad mediante el uso de guardas, un usuario de las instalaciones deportivas podría intentar acceder a una ruta del apartado de administración y viceversa.

Appuntate cuenta con dos guardas para solventar este problema, una guarda que comprueba que el usuario haya iniciado sesión y una guarda que no permite el acceso directo a una ruta mediante la modificación de la URL en la barra del navegador, la primera guarda se utiliza en las rutas principales tanto del apartado de administración como en el apartado reservado para los usuarios de las pistas para asegurar que el usuario no pueda acceder a ninguno de los dos apartados sin haber iniciado sesión, la segunda bloquea la navegación mediante la modificación de la URL, lo que significa que el usuario solo puede navegar a través de la aplicación mediante la interfaz de usuario y, en consecuencia, como no se puede navegar mediante la interfaz desde el apartado de administración al apartado de usuario y viceversa, los usuarios, una vez iniciada sesión, no podrán salir de sus apartados respectivos.

8.5 Interceptores

En **Angular** los interceptores son herramientas que, como su propio nombre indica, son capaces de interceptar peticiones HTTP.

Se declaran mediante clases de **TypeScript** utilizando el decorador @Injectable e implementando la interfaz "HttpInterceptor", al implementar esta interfaz cuentan con un método llamado "intercept" que toma como parámetro la petición HTTP y permite realizar modificaciones sobre la misma antes de que esta sea enviada.

En el contexto de Appuntate, durante el desarrollo, no ha sido necesario el uso de interceptores HTTP, sin embargo, sí que se ha desarrollado un interceptor teniendo en cuenta que, en un futuro, el back-end de Appuntate soportara la autenticación mediante tokens JWT.

Los tokens JWT pueden ser utilizados desde el back-end para determinar si un usuario tiene permisos para realizar ciertas peticiones o para comprobar que la sesión del usuario sigue activa y no ha caducado, entre otras diversas aplicaciones.

Para que esto sea posible, cuando un usuario inicia sesión desde el front-end, el back-end envía en la respuesta a la petición de inicio de sesión, un token JWT, ese token suele almacenarse en el local storage del navegador y se introduce desde el front-end en los *headers*, cabeceras, de las peticiones HTTP que se realizan al back-end, de esta manera, el back-end realiza una serie de comprobaciones basándose en ese token y decide si servir o no esa petición.

El interceptor desarrollado hace exactamente lo que se ha mencionado en el párrafo anterior pero no entrara en uso hasta que el back-end de Appuntate implemente la autenticación mediante tokens JWT.

```
Complexity is 3 Everything is cool!
@Injectable()
export class JwtInterceptor implements HttpInterceptor {

  constructor(
    private authenticationService: AuthenticationService
  ) {}

  Complexity is 3 Everything is cool!
  intercept(request: HttpRequest<unknown>, next: HttpHandler): Observable<HttpEvent<unknown>> {
    const currentUser = this.authenticationService.getCurrentUserObject();
    const token = currentUser?.jwt;
    if(token){
      request = request.clone({
        // eslint-disable-next-line @typescript-eslint/naming-convention
        setHeaders: { Authorization: `Bearer ${currentUser.jwt}` }
    });
    }
    return next.handle(request);
  }
}
```

Ilustración 62 - Interceptor JWT

8.6 Componentes

Los componentes en **Angular** son la base sobre sobre la que se construyen las aplicaciones, estos componentes se definen con el decorador @Component que toma como parámetro un objeto que ha de contener tres atributos, selector, template y styles, el atributo selector define la etiqueta **HTML** que será utilizada para introducir ese elemento dentro de una interfaz, el atributo template contiene o bien la ruta a un archivo **HTML** que se utilizara para renderizar la vista del componente o directamente el código **HTML** y el atributo styles contiene, de la misma manera, o bien la ruta a un archivo **CSS** o directamente los estilos de la vista del componente.

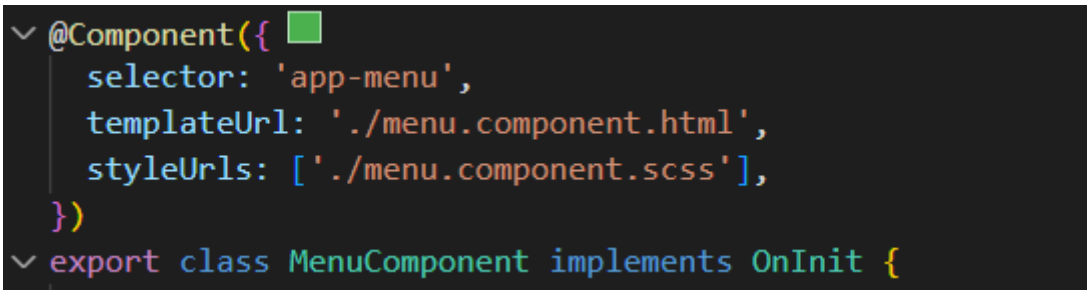

Ilustración 63 - Decorador Component

Angular proporciona herramientas para ayudar a crear dinamismo en los componentes, estas herramientas consiguen ese objetivo facilitando el control de la plantilla **HTML** con **TypeScript**. Las herramientas son el enlazado de atributos o, *attribute binding*, y la interpolación de texto o, *text interpolation*.

El enlazado de atributos consiste en relacionar el valor de una variable u expresión de **TypeScript** a un atributo de un elemento **HTML**, esto se consigue encapsulando el atributo que se quiere enlazar entre corchetes, al realizar esto, se le podrá asignar una variable u expresión **TypeScript** a ese atributo. El enlazado de atributos no debe confundirse con el enlazado bidireccional ya que el enlazado de atributos solo funciona en una dirección, desde el **TypeScript** hasta el **HTML**, todos los cambios en el **TypeScript** se verán reflejados en el **HTML** pero no en la dirección contraria. La ilustración 64 muestra un ejemplo de enlazado de atributos utilizado en Appuntate.

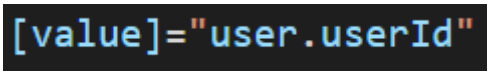

Ilustración 64 - Ejemplo de enlazado de atributos

Similarmente al enlazado de atributos, la interpolación de texto también nos permite introducir variables y expresiones **TypeScript** en la plantilla **HTML**. A diferencia del enlazado de atributos, la interpolación de texto se puede utilizar tanto en atributos **HTML** como en cualquier otra parte de la plantilla. La interpolación de texto hace uso de llaves dobles para contener la variable o expresión **TypeScript**, si se usan estas llaves en el valor de un atributo también puede utilizarse para asignar el valor de ese atributo a una variable o expresión **TypeScript,** pero además de esto, se puede utilizar en cualquier parte de la plantilla permitiendo combinar texto plano con **TypeScript**. La ilustración 65 muestra un ejemplo de la interpolación de texto utilizada en Appuntate.

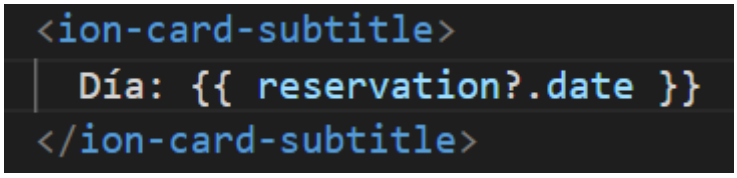

Ilustración 65 - Ejemplo de interpolación de texto

Los componentes cuentan con *two-way-binding*, esto se consigue a través del uso de una directiva, esa y el resto de las directivas que ofrece **Angular** permiten crear elementos dinámicos de la interfaz de usuario de una manera sencilla, a continuación, se detallan esas directivas.

Angular incorpora por defecto dos tipos de directivas, directivas que modifican los atributos o el comportamiento de los elementos del DOM y directivas estructurales que son capaces de modificar el DOM añadiendo o eliminando elementos.

Las directivas que proporciona para la modificación de atributos son las siguientes:

- NgClass
    - Esta directiva permite modificar los estilos de los elementos del DOM añadiendo o modificando sus clases **CSS**, esto lo hace a través de un ternario, dando un estilo u otro dependiendo del valor de la condición, true o false. Es especialmente útil si se quiere modificar la apariencia de un elemento de manera dinámica.
- NgStyle
    - Funciona de la misma manera que NgClass con la diferencia de que permite modificar atributos **CSS** en lugar de clases completas, de esta manera se puede modificar, por ejemplo, el tamaño de un bloque de texto sin tener que modificar el resto de los estilos que se le están aplicando a ese texto.
- NgModel
    - Esta directiva es la responsable del enlazado bidireccional, o *two-way data binding*. Permite enlazar el valor de una variable **TypeScript** con el valor que contiene un elemento **HTML**, de esta manera y como se ha mencionado ya anteriormente, cuando se produce un cambio en el **TypeScript**, este se verá reflejado en el **HTML** y cuando se produce un cambio en el **HTML**, este se verá reflejado en el **TypeScript**.

Tal y como se ha mencionado anteriormente, los componentes se utilizan para construir interfaces de usuario y en Appuntate existen dos tipos de componentes, componentes compartidos por varios módulos y componentes que son propios de un único módulo, en la ilustración 66 se puede apreciar el uso de estos dos tipos de componentes, el componente resaltado en azul siendo un componente compartido por varios módulos y el resaltado en amarillo que solo es utilizado en ese módulo.

Ilustración 66 - Pantalla haciendo uso de componentes globales y de su propio módulo

9. Integración con la API REST

En este apartado se describe como se ha realizado la integración con el back-end de Appuntate que implementa una API REST, detallando los mecanismos que nos proporcionan **Angular** y la librería RxJs para facilitar el desarrollo.

9.1 HTTP

El back-end de Appuntate ha sido desarrollado utilizando un framework basado en Java llamado Spring, este back-end ha sido diseñado para el tratamiento de los datos de la aplicación y ofrece una API REST accesible mediante el protocolo HTTP que proporciona los servicios necesarios para la comunicación con el front-end.

Esta API REST permite tanto el envío como la descarga de los datos que requiere o introduce el usuario exponiendo una serie de "*endpoints*" que actúan como la puerta de acceso a los datos. Para facilitar el acceso a los "*endpoints*", **Angular** proporciona una API para realizar peticiones HTTP.

Esta API esta implementada en un servicio que proporciona **Angular** y para hacer uso de ella hay que importar el módulo HttpClientModule en el módulo en el cual se quiere hacer uso de ella e inyectar el servicio HttpClient en la clase en la que se vayan a implementar las llamadas declarando el servicio en el constructor.

Una vez hecho esto, es posible realizar peticiones HTTP, en Appuntate se han utilizado llamadas de tipo POST, GET, PUT y DELETE, las llamadas POST para crear nuevas entradas de datos, las llamadas GET para solicitar datos, las llamadas PUT para modificar entradas de datos y las llamadas DELETE para eliminar entradas de datos.

Tanto los datos enviados como recibidos desde y al back-end están en formato JSON y están modelados haciendo uso de interfaces **TypeScript**.

Para crear las peticiones HTTP hay que invocar al método correspondiente que ofrece el servicio HttpClient para esa llamada, en el caso de una llamada de tipo GET hay que invocar al método get(), en el caso de una llamada de tipo POST hay que invocar al método post(), etc. En la ilustración 67 se muestra un ejemplo de una llamada de tipo GET dentro de Appuntate.

```
getCourtReservations(
  courtId: number
): Observable<Reservation[]>{
  return this.http.get<Reservation[]>(`${this.apiBaseUrl}/getCenterReservations/${courtId}`);
}
```

Ilustración 67 - Llamada GET

En el caso de que la petición requiera un cuerpo que contenga una serie de datos este se pasa como parámetro dentro del método correspondiente con forma de objeto **TypeScript**. En la

ilustración 67 se muestra una llamada que envía datos al servidor pasando esos datos como parámetro al método post().

```
getCenterCourtsByFilters(
  request: GetCenterCourtsRequest
){
  return this.http.post<Court[]>(`${this.apiBaseUrl}/getCenterCourtsByFilter` , request);
}
```

Ilustración 68 - Llamada POST

Debido a que las peticiones realizadas a los servicios tienen un tiempo de latencia y este tiempo de latencia puede variar debido a multitud de factores, todos estos métodos diseñados para realizar peticiones devuelven un observable con la respuesta que envía el back-end, este tipo de objetos se utiliza para realizar operaciones asíncronas como en este caso son las peticiones al back-end.

9.2 RxJs

En el anterior apartado se ha mencionado el uso de observables para tratar las respuestas del back-end debido a la naturaleza asíncrona de las peticiones HTTP. En este apartado se expone la manera en la que la librería RxJs proporciona los mecanismos necesarios para tratar la asincronía de las peticiones HTTP.

RxJs es una librería **JavaScript** para la creación de aplicaciones asíncronas basadas en eventos, esto lo consigue mediante un tipo de objeto llamado Observable acompañado de una serie de herramientas.

Un objeto de tipo Observable es un objeto que es capaz de emitir múltiples colecciones de valores. Para acceder a los valores que emiten los Observables es necesario suscribirse a ellos, estos objetos son capaces de emitir valores distintos en momentos distintos, estos valores una vez emitidos son recibidos por todos los puntos en la aplicación desde los cuales se haya creado una suscripción a ese Observable. La suscripción a un Observable funciona de la misma manera que una llamada a una función, si se llama a una función, esta, devolverá un valor, si existe una suscripción a un Observable, este, emitirá un valor, si no se invoca a una función, esta, no devolverá un valor y si no existe una suscripción a un observable, este, no emitirá un valor, la diferencia reside en que cuando se hace una llamada a una función, la función solo es capaz de devolver un valor, sin embargo, los Observables son capaces de emitir un número infinito de valores desde el momento en que se crea una suscripción, es decir, cuando se crea una suscripción, ese punto del programa queda a la escucha del valor que emite el Observable. Para conseguir que un observable emita un valor ha de invocarse el método next() sobre el observable con el valor que se quiere emitir, en un momento determinado del programa puede que se invoque al método next() con un valor determinado, el cual será recibido por un punto determinado del programa donde exista una suscripción, y en otro momento, el método next() puede ser invocado con un valor distinto y ese valor también será recibido por el mismo punto en el programa en el que existe la suscripción, esta característica los convierte en la herramienta ideal para almacenar valores recibidos mediante una respuesta HTTP.

El servicio HttpClient que proporciona **Angular** almacena los valores recibidos de las respuestas a las peticiones HTTP en objetos de tipo Observable debido al comportamiento mencionado. Al realizar una petición HTTP esta no ocurre de manera instantánea, existe un tiempo que transcurre desde que se envía la petición y se recibe la respuesta, de esta manera, al almacenarse en un observable, desde la parte de la aplicación desde la cual se quiera acceder a la respuesta, se puede crear una suscripción que expulsara el valor de la respuesta una vez sea recibida.

10. Diseño de la Interfaz de usuario

En cuanto al diseño de las pantallas se ha decidido utilizar componentes de la librería que proporciona **Ionic** intentando mantener las interfaces lo más sencillas e intuitivas posibles.

Al tratarse de una aplicación híbrida pensada para ejecutarse en distintas plataformas y en dispositivos con tamaños de pantalla distintos, se ha optado por desarrollar un diseño pensando en dos tipos de pantallas, la de un monitor de sobremesa y la de un teléfono móvil.

Para conseguir que las vistas de Appuntate se adapten tanto a la pantalla de un monitor como a la pantalla de un teléfono móvil, se han utilizado media queries.

Los media queries son una herramienta que proporciona **CSS** para establecer una condición sobre un bloque de código, de esta manera se ha establecido que ciertas propiedades **CSS** solo se utilicen cuando el tamaño de la pantalla cambia, esto permite modificar las propiedades como display-flex para modificar la manera en la que se organizan los elementos en la pantalla.

```
.container{
  display: flex;
  @media (max-width: 1230px) {
    flex-direction: column;
  }
```

Ilustración 69 - Media Query

Ya que el diseño se ha enfocado en dos tamaños de pantalla, la de un monitor y la de un teléfono móvil, se han desarrollado dos wireframes por pantalla, uno dedicado a la vista móvil y otro dedicado a la vista de un monitor.

Para la realización de estos wireframes se ha utilizado la herramienta MockFlow, esta herramienta proporciona una interfaz sencilla e intuitiva que permite realizar wireframes sin la necesidad de un entrenamiento extenso utilizando la herramienta, además de una interfaz sencilla, también proporciona un kit de componentes de **Ionic**, lo que ha permitido realizar wireframes con componentes que tienen el mismo diseño que los que se van a utilizar para el desarrollo.

A continuación, se muestran algunos de los wireframes desarrollados.

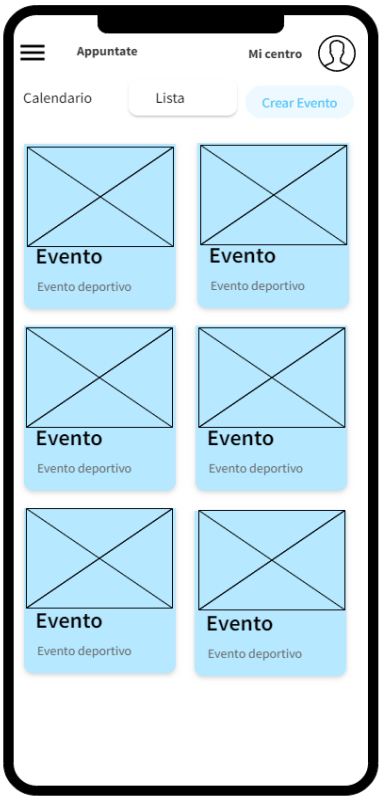

Ilustración 70 - Wireframe móvil 1

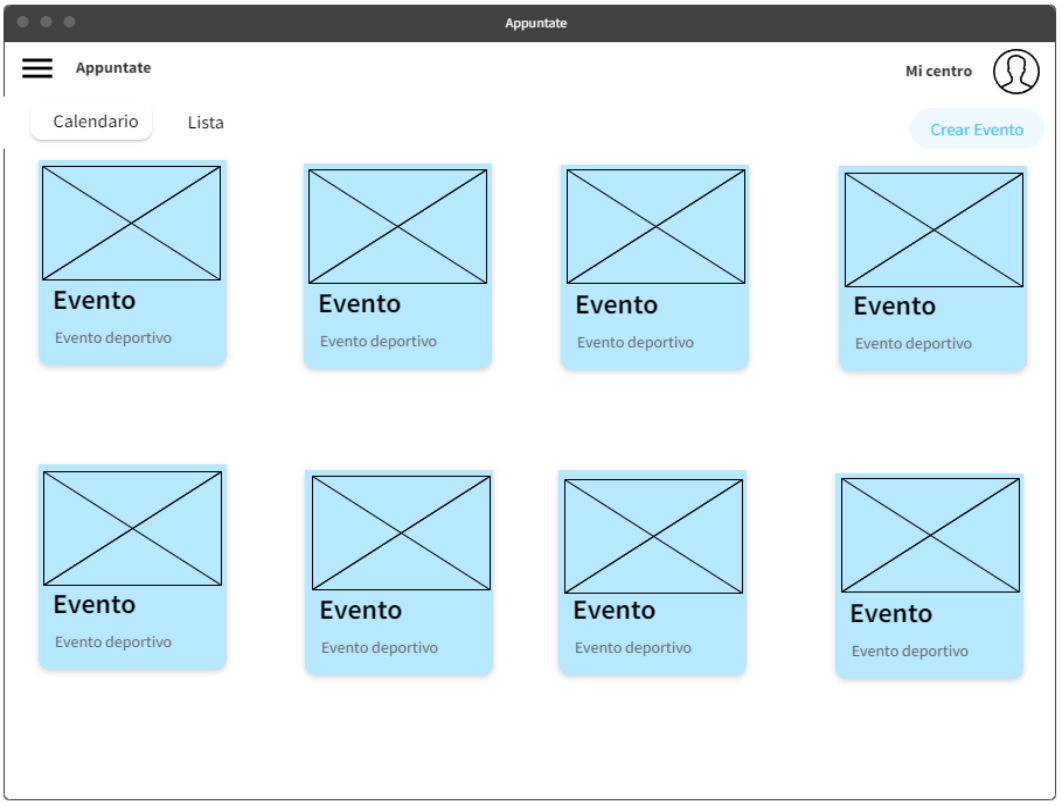

Ilustración 71 - Wireframe escritorio 1

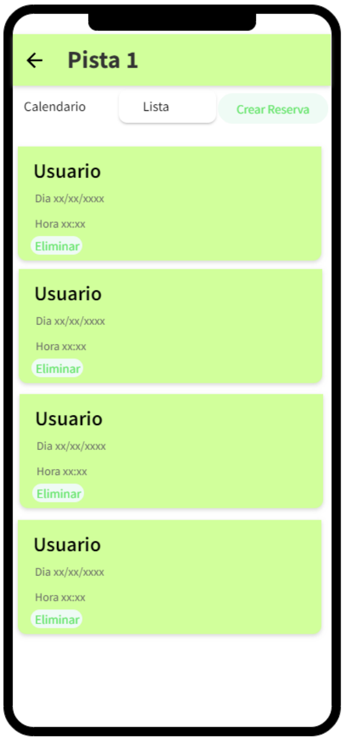

Ilustración 72 - Wireframe móvil 2

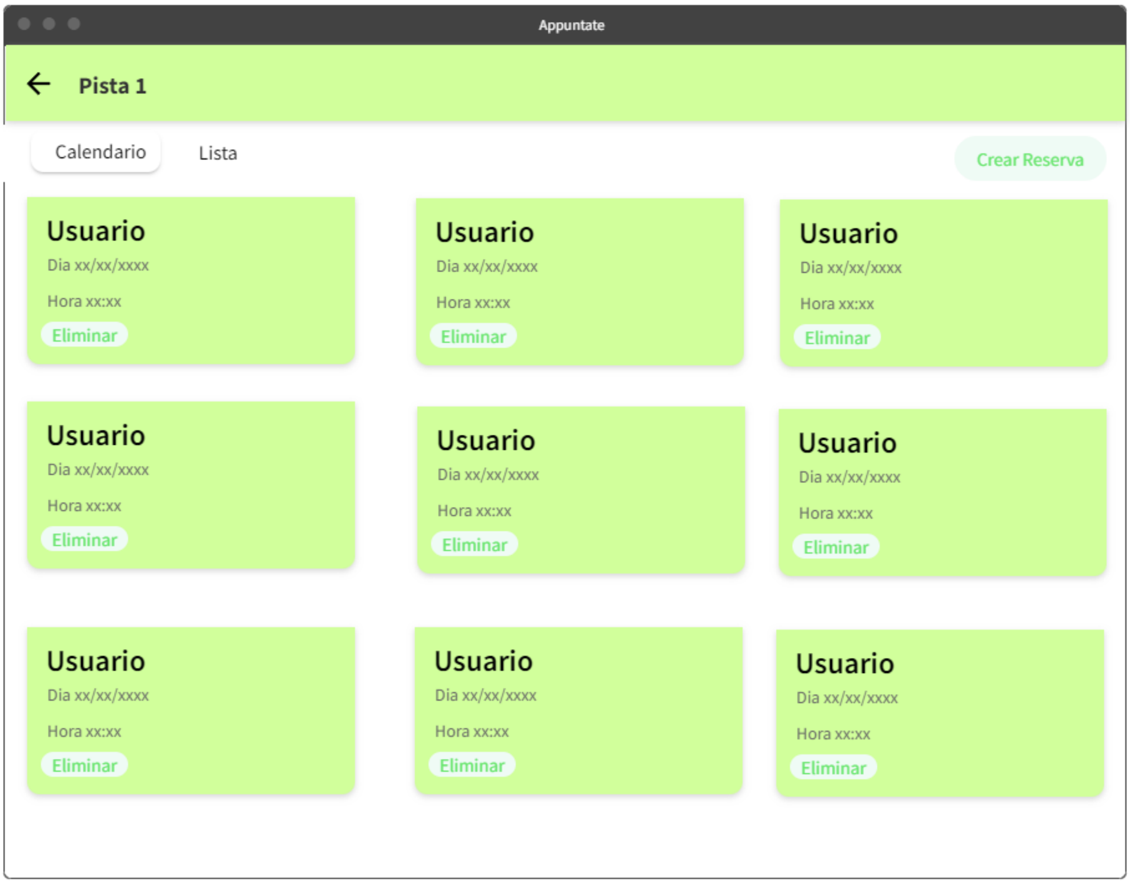

Ilustración 73 - Wireframe escritorio 2

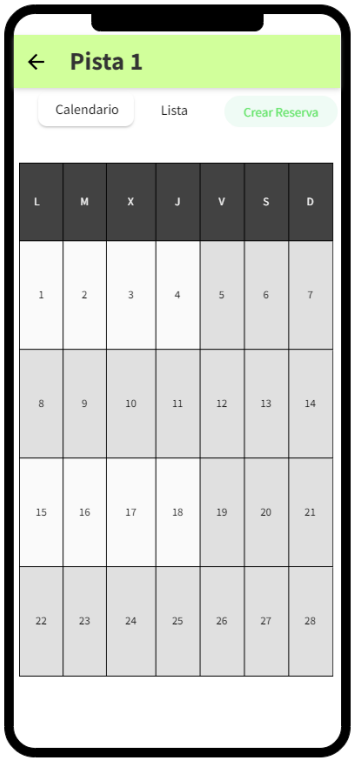

Ilustración 74 - Wireframe móvil 3

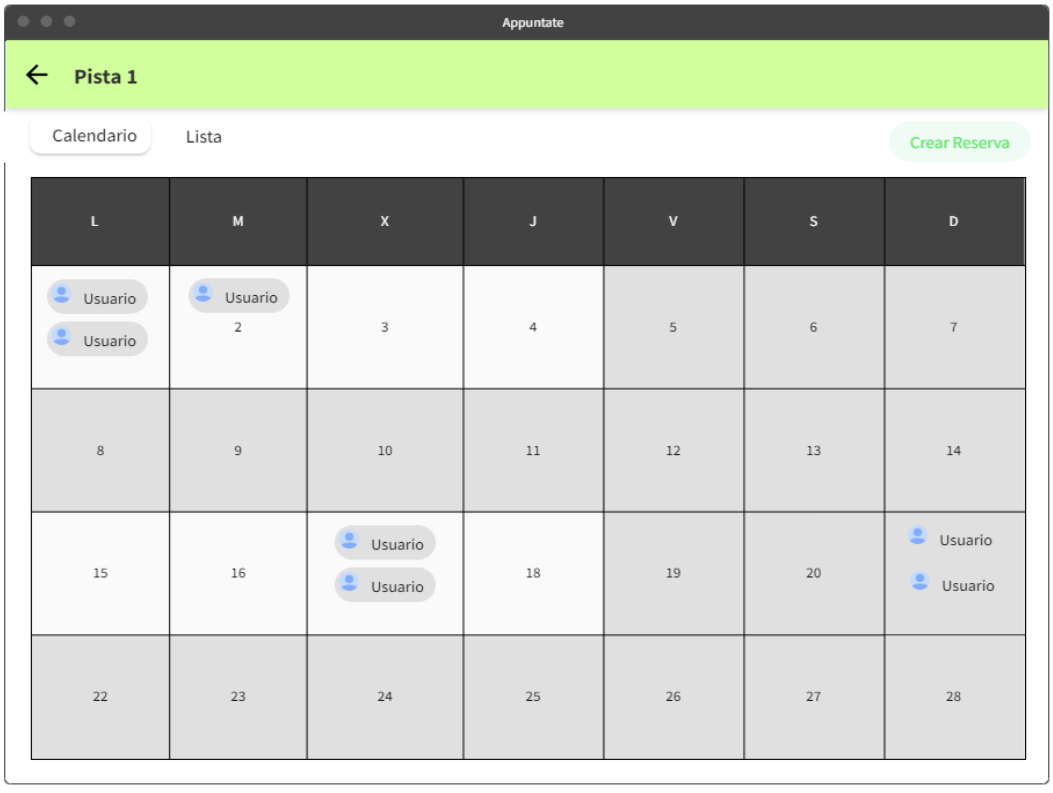

Ilustración 75 - Wireframe escritorio 3

11. Testing

El testing es un factor esencial en el desarrollo de software de calidad, es una herramienta que ayuda a comprobar la corrección del código desarrollado ayudando a detectar errores y a asegurarse de que el código hace lo que debe hacer.

La capacidad de poder detectar errores mediante el uso de tests reduce significativamente la posible deuda técnica acarreada durante el desarrollo ya que son una barrera para posibles errores que pueden pasar desapercibidos durante la fase de desarrollo, además de ser útiles en labores de refactorización.

Los tests son útiles en cuanto a labores de refactorización ya que no solo se pueden utilizar para comprobar que el código hace lo que debería hacer, sino que pueden ser utilizados para entender que debe hacer un trozo determinado de código. Cuando existe un equipo con varios desarrolladores los tests pueden servir como apoyo para entender cuál debe ser el funcionamiento del código reduciendo el tiempo necesario para solucionar posibles errores o realizar modificaciones, especialmente cuando el desarrollador encargado de estas tareas no es la misma persona que desarrollo ese código en primera instancia.

La reducción de la deuda técnica es especialmente importante en proyectos en los que trabajan varias personas y se utilizan metodologías agiles ya que cada desarrollador modifica pequeñas partes del código y las labores de refactorización suelen ser escasas. El problema con la deuda técnica es que, si no se solventa, actúa de una manera parecida al interés compuesto y conforme se avanza en el desarrollo se convierte en una bola de nieve cada vez más grande con un crecimiento exponencial, por esta razón, el testing es especialmente importante para Appuntate ya que, al ser un proyecto de emprendimiento, es importante intentar reducir los costes al máximo. Dado que el testing consigue reducir la deuda técnica asegurando la calidad del código desarrollado, el testing debe convertirse en una prioridad para Appuntate.

Durante el desarrollo se han desarrollado dos tipos de tests, tests unitarios y tests end-to-end, los cuales se detallan en los siguientes apartados.

11.1 Tests unitarios

Los tests unitarios son pruebas que se realizan sobre pequeños trozos de código para comprobar que se comportan de la manera esperada. Las pruebas unitarias se realizan desacoplando completamente el trozo de código que se quiere comprobar y estableciendo la salida esperada para un caso de prueba determinado [28].

Este tipo de pruebas son parte del *Test Driven Development*, TDD, o desarrollo dirigido por pruebas, este desarrollo consiste en desarrollar las pruebas unitarias antes de desarrollar el código, la idea detrás de esto es que el desarrollador, una vez creados los tests, escriba código que sea capaz de pasar esos tests, de esta manera la persona encargada de un desarrollo determinado se asegura de que su código haga exactamente lo que espera que haga, ya que, si no lo hace, el código no sería capaz de pasar los tests, una vez que el código desarrollado es capaz de pasar los tests, el

siguiente paso es refactorizar el código para optimizarlo. Los pasos mencionados forman parte del ciclo que se puede ver en la ilustración 76.

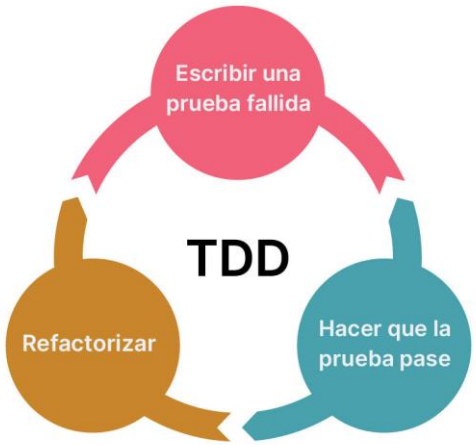

Ilustración 76 - Ciclo TDD

Aunque el desarrollo dirigido por pruebas es una buena manera de asegurar un código limpio y de calidad, no se ha utilizado para el desarrollo de Appuntate ya que es un proceso que tiene una cierta curva de aprendizaje y puede ser tedioso para desarrolladores que no tengan experiencia con el proceso, al tratarse de una idea de negocio, lo óptimo es que los desarrolladores que se incorporen al equipo en un futuro puedan entender el proceso de desarrollo y empezar a desarrollar en el menor tiempo posible, el TDD puede ser una carga que ralentice este proceso.

En el contexto de Appuntate las pruebas unitarias han sido utilizadas con un propósito distinto, el de comprobar el código una vez ha sido desarrollado, al escribir pruebas para código ya escrito, el desarrollador se ve obligado a repasar el código, comprobar que hace lo que debe hacer y si no es así, corregirlo.

Para la realización de los tests unitarios se ha escogido el framework **Jest** [29]**,** un framework dedicado al testing en **JavaScript** que es utilizado por empresas como Facebook, Twitter, Spotify y Airbnb. La decisión de utilizar **Jest** está basada en que es un framework que cuenta con una documentación rica y extensa, tiene una configuración sencilla y es capaz de ejecutar tests de manera notablemente más veloz que otras alternativas como Karma.

Los tests en **Jest** utilizan el método “expect” para establecer la condición que se tiene que dar para que una prueba pase o no, este método permite establecer todas las condiciones necesarias para probar cualquier trozo de código, se puede establecer que una expresión debe tener un valor determinado, que una variable booleana se evalúe a true o false, que se haya realizado una llamada a una función determinada, etc., la idea es ejecutar el código que se quiere probar y establecer con el método “expect” una condición que tiene que ser cierta y que determine que el código ha realizado la tarea para la que había sido diseñado.

En la ilustración 77 se pueden ver los tests realizados para el servicio de fechas. Se puede apreciar que existe una prueba por cada uno de los métodos de la clase. El funcionamiento de los tests consiste en realizar una llamada al método que se quiere probar con un caso de la prueba para el que ya se sabe cuál debería ser la salida, después de realizar la llamada se establece a través del método “expect” cuál debe ser la salida.

```
describe('DateService', () => {
  let service: DateService;

  beforeEach(() => {
    TestBed.configureTestingModule({});
    service = TestBed.inject(DateService);
  });

  it('should be created', () => {
    expect(service).toBeTruthy();
  });

  it('should convert to date format', () => {
    const date: Date = new Date('2011-10-05T14:48:00.000Z');
    const parsedDate = service.toDateFormat(date);
    expect(parsedDate).toEqual(
      '05-10-2011'
    );
  });

  it('should convert from date to hour format', () => {
    const date: Date = new Date('2011-10-05T14:48:00.000Z');
    const parsedDate = service.toHourFormat(date);
    expect(parsedDate).toEqual(
      '14:48'
    );
  });

  it('should convert from date to hour format', () => {
    const date: Date = new Date('2011-10-05T14:48:00.000Z');
    const parsedDate = service.toHourFormat(date);
    expect(parsedDate).toEqual(
      '14:48'
    );
  });

  it('should convert from server to ISO format', () => {
    const date = '2011-10-05';
    const hour = '14:48';
    const parsedDate = service.serverToISOFormat(date, hour);
    expect(parsedDate).toEqual(
      '2011-10-05T14:48:00'
    );
  });

  it('should convert from server to ISO format with single digit day', () => {
    const date = '2011-10-5';
    const hour = '14:48';
    const parsedDate = service.serverToISOFormat(date, hour);
    expect(parsedDate).toEqual(
      '2011-10-05T14:48:00'
    );
  });
});

```

Ilustración 77 - Tests unitarios

11.2 Tests end-to-end

Los tests end-to-end son un tipo de pruebas que al contrario que los tests unitarios no prueban trozos individuales y aislados de código, sino que prueban el software desarrollado en toda su extensión.

Estas pruebas se suelen realizar sobre los distintos flujos que puede tener un usuario dentro la aplicación, por ejemplo, en el caso de Appuntate uno de los posibles flujos es que un usuario desde la página de búsqueda busque una pista y la reserve, la comprobación de que este flujo funcione correctamente puede ser una prueba end-to-end.

Este tipo de pruebas suelen ser realizadas por una persona encargada del *quality assurance* o garantía de calidad con acceso a la aplicación, esta persona sigue los flujos que seguiría un usuario normal dentro de la aplicación en busca de posibles errores, en el caso de que se detectase un error, este error debería ser resuelto por el equipo de desarrollo y la persona encargada de la garantía de calidad, una vez la incidencia estuviese solucionada, debería realizar la misma prueba con la cual detecto el error para comprobar que el error, efectivamente, ha sido corregido.

Sin embargo, también existen herramientas para la automatización de estas pruebas. Algunas de las ventajas que existen al automatizar este tipo de pruebas son la reducción en el coste de personal, ya que, en muchos casos, si se automatizan, estas pruebas son desarrolladas por los desarrolladores y no por las personas encargadas de la garantía de calidad, pero también hay excepciones, en algunos casos puede que las personas encargadas de *quality assurance* desarrollen estos tests, y que las pruebas automatizadas pueden ahorrar tiempo ya que una vez se desarrollan, si se detecta un error, el desarrollador es capaz de realizar modificaciones en el código y volver a lanzar la prueba para comprobar si el error ha quedado solucionado, evitando así el proceso de distribución a las personas que realizarían las pruebas de manera manual, testeo por la parte de estas personas y la comunicación al equipo de desarrollo de los posibles errores encontrados.

En el contexto de Appuntate tiene sentido automatizar este tipo de pruebas ya que supone una reducción considerable en el coste de llevarlas a cabo cuando se compara con el coste de contratar a personas que se dediquen únicamente a la realización de estas pruebas.

La herramienta seleccionada para realizar este tipo de pruebas es **Cypress** [30], esta herramienta es un framework que permite el desarrollo de pruebas end-to-end. Se ha decidido utilizar **Cypress** porque ofrece una documentación de calidad, es sencillo de configurar y escribir tests es igualmente fácil.

Este tipo de frameworks ofrecen la grabación de los tests, de esta manera cuando la cantidad de tests es considerable y el tiempo que requieren para ejecutarse es extenso, el desarrollador puede lanzarlos, dejar que se ejecuten y más tarde revisar cuales han fallado y en que paso visualizando los videos generados.

En la ilustración 78 se muestra una prueba desarrollada para el flujo en el cual un usuario hace clic en la tarjeta de un polideportivo y llama al centro a través de la aplicación.

```
describe('search-available-courts', () => {
  it('calls center', () => {
    cy.visit('http://localhost:8100/search-available-courts');
    cy.wait(500);
    // eslint-disable-next-line max-len
    cy.get(
      '#main > app-search-available-courts > ion-content > div:nth-child(2) > app-court-card'
    ).click();
    cy.wait(500);
    cy.get(
      // eslint-disable-next-line max-len
      '#main > app-center-preview > ion-content > div > div.info > ion-grid > ion-row:nth-child(6) > ion-col > div > ion-button:nth-child(1)'
    ).click();
  });
```

Ilustración 78 - Test end-to-end

El proceso consiste en indicarle a la prueba que debe navegar a una URL determinada y después mediante selectores **CSS** se le indica en que elementos debe hacer clic para llevar a cabo el flujo.

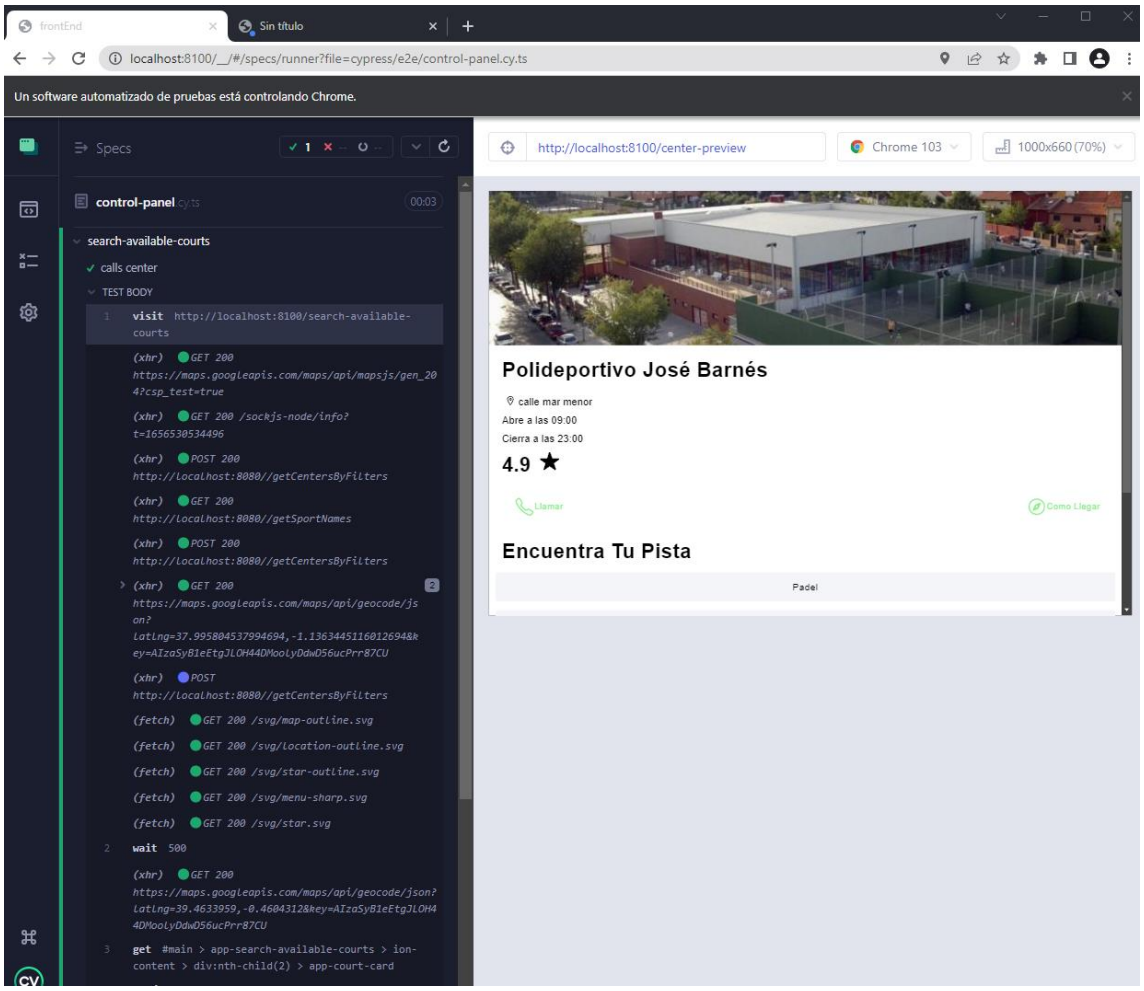

Ilustración 79 - Cypress

En la ilustración 79 se muestra la interfaz gráfica de **Cypress** donde se puede ver la vista de navegador y el sistema de logs que tiene en la parte izquierda donde muestra todas las acciones que se están realizando.

12. Implantación

Para la implantación de Appuntate se ha decidido utilizar una herramienta de automatización para facilitar el proceso de lanzamiento de versiones tanto al entorno de producción como al equipo que existirá en un futuro y estará encargado de realizar pruebas sobre Appuntate.

Para el lanzamiento de la versión **PWA** se ha decidido utilizar **Firebase** Hosting [31] ya que permite alojar páginas web y **PWA**s de manera sencilla además de que esa misma herramienta también permite realizar la distribución de versiones a futuros testers. En cuanto a las versiones móviles las plataformas de lanzamiento son las tiendas de aplicaciones de cada plataforma, la Google Play Store y la Apple App Store.

La herramienta elegida para realizar este proceso de automatización es **Fastlane** [32], una herramienta de código abierto dedicada al lanzamiento y la implantación de aplicaciones móviles. **Fastlane** inicialmente fue pensada para distribuir aplicaciones **iOS** pero en la actualidad la herramienta está siendo mantenida por Google y es capaz de automatizar el lanzamiento a entornos de prueba y de producción de tanto aplicaciones **iOS** como aplicaciones **Android**.

Fastlane permite automatizar estos procesos mediante un archivo llamado Fastfile en el que se pueden introducir unas herramientas llamadas lanes, estas lanes hacen uso de una serie de plugins que han sido desarrollados para **Fastlane** y que son capaces de realizar acciones como la compilación de una aplicación o la distribución de una versión a través de **Firebase**, dentro de estas lanes se pueden ejecutar tanto plugins como scripts .sh.

Para Appuntate se han creado diversas lanes para automatizar una serie de acciones. Una ventaja que trae **Fastlane** es que no solo sirve para automatizar el lanzamiento e implantación de versiones, sino que al ser capaz de ejecutar scripts .sh también sirve para automatizar tareas como inicializar el proyecto **Android** simplemente lanzando una lane, esto es particularmente interesante para el momento en el que el equipo de Appuntate crezca, los nuevos desarrolladores tan solo tendrán que lanzar una lane determinada para realizar acciones como crear los proyectos de **Android** y de **iOS** justo después de clonar el repositorio, lo cual agilizara el proceso de integración de nuevos miembros al equipo ya que configurar las aplicaciones nativas será tan sencillo como ejecutar un solo comando en la terminal.

En la ilustración 80 se puede ver el Fastfile de Appuntate.

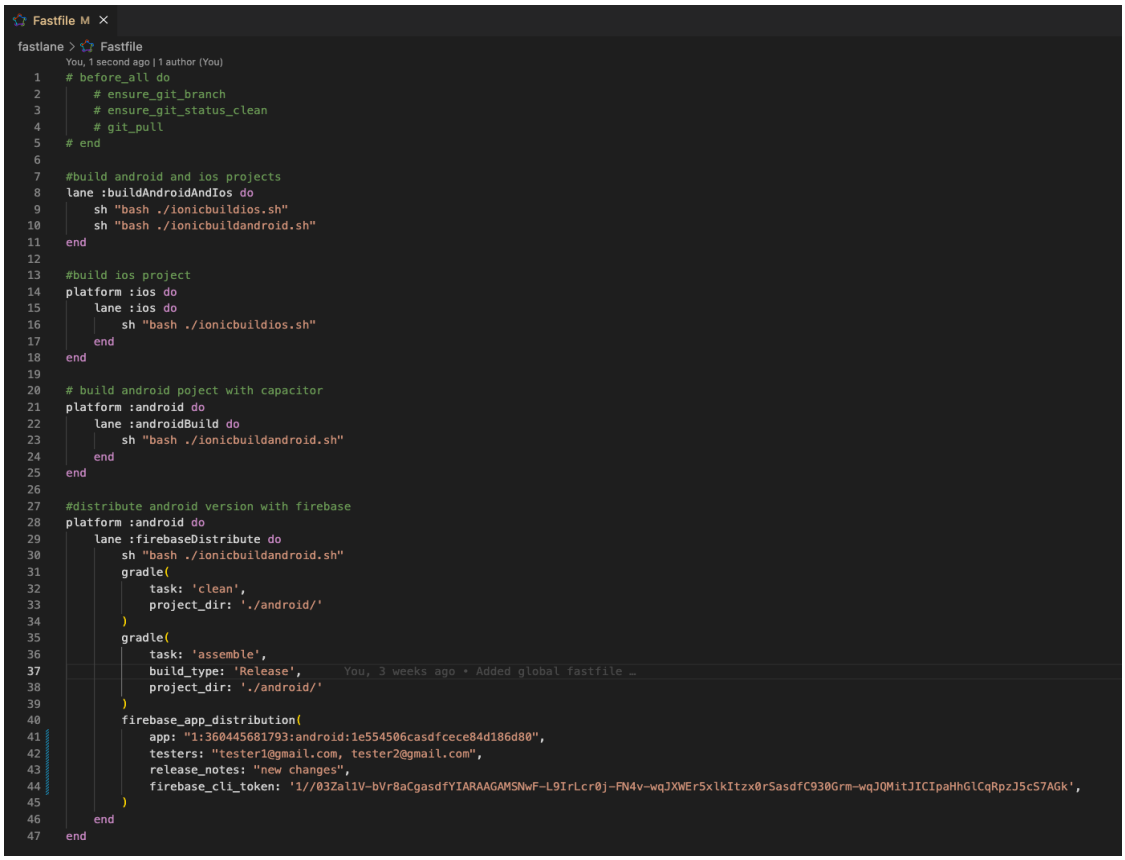

Ilustración 80 - Fastfile

La última de las lanes utiliza un plugin desarrollado para distribuir versiones a través de **Firebase**, para esto es necesario crear un proyecto desde la página web de **Firebase**, añadir una aplicación, ya sea **iOS** o **Android** al proyecto, obtener su ID y generar un token para la línea de comandos de **Firebase**, una vez realizados estos pasos, solo es necesarios incluirlos en el plugin y lanzar la lane.

Durante estas primeras fases del desarrollo de Appuntate no se ha distribuido una versión de prueba para la versión de **iOS** ya que es necesaria una cuenta de desarrollador de Apple, de la misma manera tampoco se ha hecho un despliegue a las tiendas de aplicaciones porque en ambos casos es necesaria también una cuenta de desarrollador, estas cuentas acarrean un coste económico y se obtendrían en fases futuras del desarrollo de Appuntate. En cuanto se obtengan las cuentas el procedimiento sería el mismo que para la distribución de versiones de prueba mediante **Firebase**, se crearían las lanes correspondientes para distribuir las aplicaciones a las tiendas.

13. Trabajos futuros

En las siguientes fases de Appuntate y conforme el equipo de Appuntate empezase crecer se tendrían en cuenta dos tipos de tareas a llevar a cabo durante el desarrollo futuro de Appuntate.

En primer lugar, se continuaría con el desarrollo de las funcionalidades incluidas en la especificación además de plantear la adición de funcionalidades propuestas por los entrevistados durante los experimentos realizados, estableciendo el foco sobre las funcionalidades dedicadas a eventos y torneos ya que han sido las más resaltadas por los administradores de las instalaciones, también se debería considerar la posibilidad de extender la solución que ofrece Appuntate incluyendo una infraestructura física de cámaras, sistemas de riego, sistemas de acceso, etc.

En segundo lugar, habría que aumentar la cobertura obtenida en los tests unitarios y desarrollar más tests end-to-end para cubrir todos los casos de uso de la aplicación y garantizar su correcto funcionamiento.

En tercer lugar, la búsqueda de feedback a través de entrevistas como las realizadas durante los experimentos debe tomar un papel principal ya que a través de estas entrevistas hemos podido extraer información que puede darnos una ventaja competitiva que nos permita ubicar a Appuntate en una mejor posición dentro del mercado donde podamos obtener un público objetivo mayor y como consecuencia un mayor número de clientes.

14. Conclusiones

El desarrollo de Appuntate ha sido una experiencia enriquecedora que ha servido tanto para realizar el espíritu emprendedor del equipo como para aprender sobre las tecnologías que están siendo utilizadas actualmente por las grandes empresas.

El desarrollo de una idea de negocio completa y la proyección económica que ello conlleva ha servido para conocer el contexto en el que se encuentra un emprendedor que quiere irrumpir en un mercado competitivo como es el de los sistemas de gestión de instalaciones deportivas, donde diferenciarse del resto productos y destacar es muy complicado y hay que buscar un hueco en el que el producto pueda encajar, lo que ha resultado ser una tarea para nada sencilla, esto es una lección que nos servirá en futuros emprendimientos fuera del contexto universitario y puede que sea una de las claves fundamentales para llevar el desarrollo de Appuntate más allá de este trabajo.

Las tecnologías utilizadas en este proyecto son las que están siendo utilizadas en la actualidad por las grandes empresas. Los dos miembros de Appuntate hemos tomado la decisión sobre las tecnologías a utilizar basándonos en las tecnologías utilizadas por las empresas en las que trabajamos actualmente, esto ha hecho posible que a través de este trabajo desarrollemos las habilidades que están siendo más cotizadas por las empresas en la actualidad, lo que ha servido para impulsar todavía más nuestra carrera profesional.

Además de las habilidades técnicas adquiridas durante el desarrollo, también se han trabajado aspectos como el trabajo en equipo, la comunicación constante entre el front-end y el back-end ha sido una pieza fundamental para el desarrollo, trabajar codo con codo y en armonía ha sido un aspecto clave para llevar adelante este proyecto. Las habilidades de comunicación han sido entrenadas a la vez que las de administración del tiempo y gestión de tareas.

El desarrollo de este proyecto también ha estado muy ligado con la materia impartida durante el grado. El trabajo ha consistido en la elaboración de un paquete software desde cero, empezando por la elección de las tecnologías a utilizar, continuando por conformar la arquitectura de la aplicación y todo lo que conlleva el proceso de desarrollo.

Se ha elaborado una especificación de requisitos para el desarrollo de un paquete software estableciendo las características que debe incluir, elaborando un modelo de dominio y estableciendo los distintos casos de uso utilizando conceptos vistos en la asignatura AER.

También se han expuesto patrones de diseño incluidos en la arquitectura como el patrón singleton y conceptos como la inyección de dependencias, tratados en la asignatura DDS.

Se ha tratado la importancia de la calidad de software estableciendo mecanismos para asegurarla y generar un proyecto que sea escalable y mantenible, estos mecanismos incluyen entre otros, el uso de GitFlow, las revisiones de código, las pruebas unitarias y las pruebas end-to-end, conceptos tratados en la asignatura MES.

Además de todo esto, también se han llevado a cabo tareas de programación utilizando una gran cantidad de conceptos tratados en distintas asignaturas a lo largo del grado.

15. Referencias bibliográficas

[1] Playtomic, «Playtomic.io,» [En línea]. Available: https://playtomic.io/. [Último acceso: 16 07 2022].

[2] Sporttia, «Sporttia,» [En línea]. Available: https://www.sporttia.com/en/home/. [Último acceso: 16 07 2022].

[3] M. Turn, «My Turn,» [En línea]. Available: https://www.myturn.es/en/home/. [Último acceso: 16 07 2022].

[4] Clicac, «Clicac,» [En línea]. Available: http://clicac.com/ClicAcWeb/. [Último acceso: 16 07 2022].

[5] O. Informática, «Omesa Informática,» [En línea]. Available: https://www.omesa.es/. [Último acceso: 16 07 2022].

[6] T. Yiu, «Medium,» [En línea]. Available: https://medium.com/alpha-beta-blog/business-models-saas-software-as-a-service-65a6a9a9dd14. [Último acceso: 16 07 2022].

[7] E. Ries, The Lean Startup: How Constant Innovation Creates Radically Successful Businesses, New York: Crown Business , 2011. [Último acceso: 16 07 2022]

[8] React.js, «React.js,» [En línea]. Available: https://reactjs.org/. [Último acceso: 16 07 2022].

[9] Vue.js, «Vue.js,» [En línea]. Available: https://vuejs.org/. [Último acceso: 16 07 2022].

[10] Google, «Angular.io,» [En línea]. Available: https://angular.io/. [Último acceso: 16 07 2022].

[11] Github, «Octoverse,» [En línea]. Available: https://octoverse.github.com/#try-a-new-practice-pull-request-wrangling. [Último acceso: 16 07 2022].

[12] Ionic, «Ionic Framework,» [En línea]. Available: https://ionicframework.com/. [Último acceso: 16 07 2022].

[13] Capacitor, «Capacitorjs,» [En línea]. Available: https://capacitorjs.com/docs. [Último acceso: 16 07 2022].

[14] Ionic, «Ionic Framework,» [En línea]. Available: https://ionicframework.com/docs/core-concepts/webview.

[15] Statistia, «Statista,» [En línea]. Available: https://www.statista.com/statistics/734332/google-play-app-installs-per-year/. [Último acceso: 16 07 2022].

[16] M. Igbal, «Business of Apps,» [En línea]. Available: https://www.businessofapps.com/data/app-statistics/#:~:text=143.6%20billion-,iOS%20App%20and%20game%20downloads,to%2032.3%20billion%20in%202021. [Último acceso: 16 07 2022].

[17] Mozilla, «Mozilla Developer,» [En línea]. Available: https://developer.mozilla.org/en-US/docs/Web/Progressive_web_apps/Installable_PWAs. [Último acceso: 16 07 2022].

[18] web.dev, «web.dev,» [En línea]. Available: https://web.dev/learn/pwa/service-workers/. [Último acceso: 16 07 2022].

[19] Google, «Angular.io,» [En línea]. Available: https://angular.io/guide/service-worker-getting-started. [Último acceso: 16 07 2022].

[20] vscode-ng-language-service, «Github,» [En línea]. Available: https://web.dev/learn/pwa/service-workers/. [Último acceso: 16 07 2022].

[21] vscode-material-icon-theme, «Github,» [En línea]. Available: https://github.com/PKief/vscode-material-icon-theme. [Último acceso: 16 07 2022].

[22] prettier-vscode, «Github,» [En línea]. Available: https://github.com/prettier/prettier-vscode. [Último acceso: 16 07 2022].

[23] vscode-eslint, «Github,» [En línea]. Available: https://github.com/Microsoft/vscode-eslint. [Último acceso: 16 07 2022].

[24] vscode-npm-scripts, «Github,» [En línea]. Available: https://github.com/Microsoft/vscode-npm-scripts. [Último acceso: 16 07 2022].

[25] thunder-client-support, «Github,» [En línea]. Available: https://github.com/rangav/thunder-client-support. [Último acceso: 16 07 2022].

[26] codemetrics, «Github,» [En línea]. Available: https://github.com/kisstkondoros/codemetrics. [Último acceso: 16 07 2022].

[27] Google, «Angular.io,» [En línea]. Available: https://angular.io/tutorial/toh-pt5. [Último acceso: 16 07 2022].

[28] Smartbear, «Smartbear,» [En línea]. Available: https://smartbear.com/learn/automated-testing/what-is-unit-testing/. [Último acceso: 16 07 2022].

[29] Jest, «Jestjs.io,» [En línea]. Available: https://jestjs.io/. [Último acceso: 16 07 2022].

[30] Cypress, «Cypress.io,» [En línea]. Available: https://www.cypress.io/. [Último acceso: 16 07 2022].

[31] Google, «Firebase,» [En línea]. Available: https://firebase.google.com/?gclid=Cj0KCQjw8O-VBhCpARIsACMvVLNMUgLp-s_bNO6CNJQ3iUnUvcwIuyq0hWwFBmjBeVrtE_rB02uJ4k4aAptfEALw_wcB&gclsrc=aw.ds. [Último acceso: 16 07 2022].

[32] Google, «Fastlane.tools,» [En línea]. Available: https://fastlane.tools/. [Último acceso: 16 07 2022].

ANEXO

OBJETIVOS DE DESARROLLO SOSTENIBLE

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

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

El desarrollo de Appuntate está relacionado con diversos objetivos de desarrollo sostenible. Siendo una propuesta de emprendimiento además de una solución completamente digital relacionada con el deporte, cuenta con una serie de características aplicables a la salud y bienestar, el trabajo decente y crecimiento económico y la ciudades y comunidades sostenibles.

Uno de los objetivos de Appuntate es impulsar la práctica de deporte en la sociedad. El incremento del ejercicio físico tiene un impacto positivo en la salud de las personas. Este fomenta un estilo de vida activo reduciendo hábitos sedentarios que pueden derivar en problemas como la obesidad y las enfermades relacionadas con la misma como problemas cardiovasculares, diabetes, hipertensión y muchas más.

Además de impulsar hábitos saludables, Appuntate es un proyecto de emprendimiento, lo que significa que, como empresa, Appuntate va a generar puestos de trabajo de calidad buscando expandir la plantilla conforme la empresa crece. La generación de estos puestos de trabajo servirá de ayuda a la economía local e incluso global ya que dada la naturaleza de los puestos de trabajo que se ofertarían como es el de desarrollador de software, estos pueden realizarse de manera remota. Esto significa que Appuntate sería capaz de generar trabajos en múltiples países.

Appuntate, como solución completamente digital también puede tener un impacto positivo en el ámbito de las ciudades y comunidades sostenibles. Con el incremento en la concentración de la población en ciudades, el uso de medios de transporte como los coches ha contribuido al empeoramiento de la calidad del aire. Appuntate ofrece un servicio que sustituye sistemas más tradicionales como pueden ser acercarse al centro deportivo para realizar una reserva. Con Appuntate los viajes en coche de esta naturaleza podrían eliminarse por completo, lo que contribuiría a la reducción de la contaminación producida por los coches.

Por último, en una época post-pandemica se han creado las infraestructuras necesarias para que la población pueda realizar tareas que antes se llevarían a cabo de manera presencial, de manera remota a través de internet. Appuntate propone una infraestructura de esta naturaleza, proporcionando las herramientas necesarias para que la reserva y gestión de instalaciones deportivas pueda realizarse de manera completamente remota. Creando una plataforma sobre la cual los centros deportivos puedan agilizar sus tareas de gestión incrementando su eficiencia y permitiendo de esta manera una mayor contribución al PIB.

Todo lo mencionado aporta a conseguir llevar a cabo los objetivos de desarrollo sostenible apostando por la creación de infraestructuras, la digitalización, hábitos de vida saludable y lucha contra la degradación medio ambiental. Propuestas como la de Appuntate son necesarias para avanzar hacia un mundo sostenible en el que la sociedad pueda prosperar sin dejar a nadie atrás.