UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería Informática

Desarrollo de una aplicación para Scouting de jugadores en el mundo del fútbol: Optimización del proceso de identificación de talentos

Trabajo Fin de Grado

Grado en Ingeniería Informática

AUTOR/A: Muñoz Montoro, Francisco

Tutor/a: Rebollo Pedruelo, Miguel

CURSO ACADÉMICO: 2022/2023

Resumen

El TFG tiene como objetivo principal el desarrollo de una aplicación de *scouting* de jugadores para fútbol, con el fin de optimizar el proceso de identificación de talentos. La propuesta se basa en la implementación de una herramienta innovadora que permita a los *scouter* buscar y descubrir jóvenes promesas de manera más eficiente y precisa.

La aplicación está centrada en la configuración de alarmas, que notificarán a los usuarios sobre jugadores según unos parámetros que hayan preestablecido con anterioridad. Esta función exclusiva brinda a los s*couter* una ventaja estratégica al recibir información inmediata y relevante sobre potenciales talentos emergentes. Además de las alarmas personalizadas, la aplicación también proporcionará otras funcionalidades complementarias, como estadísticas detalladas de jugadores, registro de observaciones y seguimiento de la progresión a lo largo del tiempo. Estas características adicionales fortalecerán la capacidad de los *scouter* para evaluar y monitorear a los jugadores de interés.

Palabras clave: scouter, scouting

Abstract

The main objective of the TFG is the development of a soccer player scouting application, in order to optimize the talent identification process. The proposal is based on the implementation of an innovative tool that allows scouts to search for and discover promising youngsters more efficiently and accurately.

The application is focused on the configuration of alarms, which will notify users about players according to parameters that they have previously established. This unique feature gives scouts a strategic advantage by receiving immediate and relevant information on potential emerging talent. In addition to custom alarms, the app will also provide other complementary features such as detailed player statistics, observation logging, and progression tracking over time. These additional features will strengthen the scouts' ability to evaluate and monitor players of interest.

Keywords : scout, scouting

Resum

El TFG té com a objectiu principal el desenvolupament d'una aplicació de *scouting* de jugadors per a futbol, amb la finalitat d'optimitzar el procés d'identificació de talents. La proposta es basa en la implementació d'una eina innovadora que permeta als *scouters* buscar i descobrir joves promeses de manera més eficient i precisa.

L'aplicació està centrada en la configuració d'alarmes, que notificaran als usuaris sobre jugadors segons uns paràmetres que hagen preestablit amb anterioritat. Aquesta funció exclusiva brinda als *scouters* un avantatge estratègic en rebre informació immediata i rellevant sobre potencials talents emergents. A més de les alarmes personalitzades, l'aplicació també proporcionaran altres funcionalitats complementàries, com a estadístiques detallades de jugadors, registre d'observacions i seguiment de la progressió al llarg del temps. Aquestes característiques addicionals enfortiran la capacitat dels *scouters* per a avaluar i monitorar als jugadors d'interés.

Paraules clau: scouter, scouting

Tabla de contenidos

1. Introducción

El *scouting*, en el ámbito del deporte, es una práctica fundamental para la identificación y evaluación de talentos. Consiste en el proceso de búsqueda, seguimiento y análisis de jugadores con el objetivo de descubrir aquellos con habilidades excepcionales y potencial para destacar en su disciplina deportiva. El *scouting* desempeña un papel crucial en el reclutamiento de jugadores para equipos profesionales, selecciones nacionales y programas de desarrollo deportivo. A través de la observación meticulosa, el análisis de datos y el seguimiento continuo, los *scouters* buscan identificar a futuros talentos y tomar decisiones fundamentadas en la formación de equipos competitivos. En este contexto, el presente trabajo se centra en el desarrollo de una aplicación de *scouting* de jugadores para el fútbol, con el propósito de mejorar y optimizar este proceso de identificación de talentos en la industria deportiva.

1.1. Motivación

La motivación que impulsa este proyecto se basa en la creciente importancia del *scouting* en el mundo del fútbol y la necesidad de herramientas innovadoras que faciliten y optimicen este proceso. El fútbol es un deporte altamente competitivo y en constante evolución, donde descubrir talentos emergentes es crucial para el éxito de los equipos y el crecimiento de los jugadores. La identificación temprana de futuros talentos es una tarea desafiante y requiere de métodos eficientes para garantizar la precisión y la objetividad en la evaluación de jugadores.

Además, el avance de la tecnología ha abierto nuevas posibilidades en el ámbito del *scouting*, brindando herramientas más sofisticadas para el análisis y seguimiento de jugadores. Esta evolución tecnológica es la que nos ha permitido contar con datos muy precisos de cada jornada con la que poder seguir creando herramientas que destaquen y sean útiles para el *scouter*, que al final es el objetivo que tengo. Por ello creo que una herramienta que sea capaz de filtrar las toneladas de información que se recaban en cada jornada y pueda brindarle al *scouter* una herramienta efectiva para identificar y evaluar talentos emergentes puede ser diferenciadora.

1.2. Objetivos

1.2.1. Objetivo general

El objetivo principal de este trabajo es desarrollar una aplicación de *scouting* de jugadores para fútbol que permita a los *scouter* suscribirse a alarmas que permitan filtrar la información recopilada en las jornadas de juego. Esta herramienta busca facilitar la identificación de talentos de manera más eficiente, proporcionando a los *scouter* una forma sencilla y precisa de encontrar jugadores prometedores. La aplicación contará con una interfaz intuitiva, un sistema de búsqueda avanzado y funciones de análisis y seguimiento de jugadores, brindando a los usuarios una herramienta integral para descubrir y evaluar talentos emergentes en el mundo del fútbol.

1.2.2. Objetivos específicos

1. Diseñar y desarrollar una interfaz intuitiva y amigable que permita a los *scouter* tener una herramienta de trabajo con la que descubrir nuevos talentos y poder hacer un cierto seguimiento de estos.

2. Tener un sistema de búsqueda en nuestra base de datos que te permita encontrar cualquier jugador en activo de cualquiera de las principales ligas del mundo, así como poder consultar su ficha.

3. Tener un sistema de guardado de jugadores en listas creadas por el propio usuario, llamadas WatchLists, de manera que el usuario pueda almacenar a aquellos jugadores que les quiera hacer un seguimiento más detallado.

4. Tener un sistema de alarmas a las que los usuarios se puedan suscribir de manera que reciban semanalmente una actualización con los nuevos jugadores.

1.3. Impacto esperado

Se han identificado dos tipos de usuarios sobre los que tiene impacto la aplicación: usuario y administrador. Además, también tiene impacto sobre dos de los diecisiete objetivos para el desarrollo sostenible.

1.3.1. Usuario

Los usuarios serán capaces de poder guardarse jugadores de cualquier liga del mundo en listas personalizables de manera que puedan hacerles un seguimiento mucho más exhaustivo además de poder tener acceso a alarmas sobre diferentes valores o situaciones de interés.

1.3.2. Administrador

El administrador podrá tener acceso a toda una serie de información acerca de que jugadores están agregando los usuarios a sus listas o sobre que alarmas están siendo más suscritas o con más notificaciones activadas. Además, tendrán estadísticas en tiempo real del uso de la aplicación.

1.3.3. Objetivos para el Desarrollo Sostenible

Además de los usuarios, la aplicación también puede contribuir a varios Objetivos de Desarrollo Sostenible (**ODS**) establecidos por las Naciones Unidas. Algunos de los ODS que podrían estar relacionados con este proyecto son los siguientes:

ODS 9: Industria, Innovación e Infraestructura: La creación de una aplicación de *scouting* moderna y eficiente utilizando tecnologías innovadoras como React Native y Node.js demuestra una apuesta por la innovación y el desarrollo de infraestructuras tecnológicas que puedan beneficiar tanto a los *scouter* como a los jugadores.

ODS 10: Reducción de las desigualdades: Al proporcionar una plataforma accesible para la detección de talentos en el fútbol, la aplicación de *scouting* podría ayudar a reducir las desigualdades al ofrecer oportunidades a jóvenes promesas que, de otro modo, podrían no ser detectadas o reconocidas.

ODS 16: Paz, justicia e instituciones sólidas: Al proporcionar una plataforma transparente y objetiva para la detección de talentos en el fútbol, la aplicación de *scouting* puede contribuir a la promoción de prácticas justas y éticas en el ámbito deportivo.

Es importante tener en cuenta que el impacto de un proyecto en los Objetivos de Desarrollo Sostenible puede variar dependiendo del contexto y del alcance de este. Sin embargo, la aplicación de *scouting*, al facilitar la detección de talentos y la mejora en la toma de decisiones en el mundo del fútbol, podría tener un efecto positivo en muchas áreas.

1.4. Estructura

En los siguientes puntos de este documento se expondrá todo el proceso de desarrollo de nuestro proyecto. En el punto número dos, se analizará el estado del arte. En este punto veremos las principales aplicaciones existentes, además, haremos una comparativa de estas con nuestra aplicación, con el fin de ver el hueco que ocuparemos en el mercado. En el punto tres, haremos un análisis en profundidad del problema. En este punto buscaremos soluciones posibles para nuestro proyecto, haremos una especificación de requisitos y crearemos un plan de trabajo. En el punto cuatro, pasaremos al diseño de la solución, se expondrá la arquitectura que conformará el sistema, junto con diagramas del flujo de datos, además de las tecnologías utilizadas para ello. En el punto cinco, se presentará el desarrollo de la solución, junto con particularidades y problemas que hayan ocurrido durante el desarrollo. En el punto seis, expondremos un apartado de pruebas realizadas al proyecto. Se mostrarán las pruebas hechas a usuarios y las pruebas unitarias llevadas a cabo en el código. En el punto siete, haremos un resumen de las conclusiones obtenidas. En el punto ocho expondremos las funcionalidades que se desarrollaran a futuro. Por último, en el punto nueve se expondrán las referencias bibliográficas que han sido necesarias para la elaboración de este documento.

2. Estado del arte

El *Scouting [1-2-3]* de jugadores en el mundo del fútbol se remonta a los primeros años del siglo XX. A medida que el fútbol se volvía un deporte más popular, los clubes comenzaban a darle importancia a identificar y reclutar futuras promesas. En esa época el proceso de *scouting* implicaba enviar ojeadores, conocidos como *scouter* a partidos y competiciones para que pudieran evaluar el rendimiento de los jugadores.

En sus inicios todo se realizaba de manera manual. Los *scouter* asistían a los partidos en persona y se dedicaban a tomar notas detalladas sobre las características y habilidades de los jugadores. Los informes se realizaban posteriormente por los miembros del cuerpo técnico y estos utilizaban la información recabada para tomar decisiones sobre los futuros fichajes que realizaría el equipo y el desarrollo que tendrían los jugadores.

En aquel entonces, la tecnología no estaba tan avanzada como lo está en la actualidad. Por lo tanto, los *scouter* dependían de su experiencia y conocimientos futbolísticos para identificar el potencial de los jugadores. No existían las herramientas digitales y tecnológicas que se utilizan en la actualidad, lo que hacía que el proceso de *scouting* fuera más laborioso y basado en la observación directa.

2.1. ¿Cómo se realiza el Scouting?

El proceso de *scouting* en el contexto del fútbol es esencial para identificar y seleccionar a jugadores prometedores. A continuación, se destacan cinco puntos principales sobre cómo se realiza el *scouting* en el mundo del fútbol:

1. Definición de criterios de búsqueda: Antes de iniciar el proceso de *scouting*, es crucial establecer los criterios específicos que se utilizarán para evaluar a los jugadores. Estos criterios pueden incluir aspectos técnicos, tácticos, físicos y mentales, como habilidades con el balón, visión de juego, velocidad, fortaleza física y capacidad de toma de decisiones. Definir estos criterios ayuda a guiar y enfocar la búsqueda de talentos.

2. Observación en diferentes contextos: El *scouting* implica observar a los jugadores en una variedad de contextos, como partidos oficiales, entrenamientos y torneos. Esto permite evaluar su desempeño en situaciones competitivas y no competitivas, así como su consistencia en diferentes escenarios. La observación se realiza tanto en persona como a través de análisis de vídeo, lo que amplía las oportunidades de evaluación.

3. Análisis y seguimiento de datos: En el proceso de *scouting*, se recopilan datos relevantes sobre los jugadores para un análisis más profundo. Esto puede incluir estadísticas de partidos, mediciones físicas, registros de lesiones y otros indicadores clave. El seguimiento de datos a lo largo del tiempo ayuda a identificar patrones de rendimiento, fortalezas y debilidades, y evaluar el progreso de los jugadores a lo largo de su desarrollo. Cuando se trata de estadísticas de partidos, se recopilan y registran meticulosamente una variedad de datos cuantitativos, tales como goles, asistencias, pases completados, intercepciones, despejes y otras métricas relacionadas con el desempeño. Estos números objetivos proporcionan una visión cuantitativa del rendimiento de los jugadores y sirven como una base sólida para comparaciones y análisis en el proceso de *scouting*. Además de las estadísticas de rendimiento, también se tiene en cuenta la recopilación de mediciones físicas precisas y confiables. Esto implica evaluar aspectos como la velocidad, la fuerza, la resistencia y otras capacidades físicas relevantes. Estos datos físicos brindan información valiosa sobre la condición atlética de los jugadores, su capacidad para enfrentar desafíos físicos y su potencial de rendimiento a largo plazo.

4. Evaluación comparativa: Una parte fundamental del *scouting* es la comparación de jugadores. Los *scouter* evalúan a múltiples jugadores en relación con los criterios establecidos, permitiendo la identificación de las cualidades y habilidades más destacadas en cada jugador. La evaluación comparativa ayuda a determinar la posición relativa de los jugadores en el panorama general y a tomar decisiones informadas sobre su potencial y encaje en un equipo.

5. Informes y recomendaciones: Después de completar la evaluación de los jugadores, los *scouter* deben elaborar informes detallados que resuman sus observaciones y conclusiones. Estos informes son presentados a los entrenadores, directores técnicos y otros miembros del equipo de toma de decisiones. Las recomendaciones basadas en el *scouting* influyen en la toma de decisiones sobre fichajes, contrataciones y desarrollo de los jugadores.

2.2. Nuevas tecnologías en el Scouting

En el ámbito del *scouting [4]*, el avance tecnológico ha traído consigo una serie de herramientas y aplicaciones innovadoras que han revolucionado la forma en que se lleva a cabo este proceso. A continuación, se presentan algunas de las nuevas tecnologías utilizadas en el *scouting*:

1. El análisis de vídeo avanzado ha revolucionado la forma en que los *scouter* examinan el desempeño de los jugadores en el fútbol. Gracias a los avances en algoritmos de reconocimiento de patrones y aprendizaje automático, ahora es posible desglosar y evaluar minuciosamente cada acción y movimiento durante un partido. Estos sistemas automatizados permiten identificar acciones específicas, como pases precisos, regates efectivos o intercepciones clave. Además de identificar acciones individuales, el análisis de vídeo avanzado también puede evaluar tácticas y estrategias empleadas por los equipos. Al aplicar algoritmos de aprendizaje automático a grandes volúmenes de datos, los sistemas pueden detectar patrones tácticos comunes, entender las dinámicas de juego y destacar aspectos relevantes del desempeño colectivo. Esta tecnología ofrece una visión detallada del juego, lo que permite a los *scouter* obtener una perspectiva más profunda y objetiva. Pueden examinar las decisiones tomadas por los jugadores en diferentes situaciones de juego, analizar su posicionamiento en el campo, identificar fortalezas y debilidades en su juego y evaluar su capacidad para adaptarse a diferentes escenarios.

2. Sistemas de seguimiento GPS: Los sistemas de seguimiento GPS se han convertido en una herramienta fundamental en el *scouting* moderno. Estos dispositivos se colocan en los jugadores y registran datos precisos sobre su rendimiento físico durante los partidos y entrenamientos. Los *scouter* pueden acceder a información detallada sobre la distancia recorrida, la velocidad, la aceleración, las zonas de mayor actividad y otros parámetros relevantes. Estos datos ayudan a evaluar la resistencia, la velocidad y la capacidad atlética de los jugadores, así como a identificar patrones de rendimiento.

2.3. Crítica al estado del arte

En el ámbito del *scouting* de jugadores en el fútbol, existen diversas aplicaciones y herramientas disponibles en la actualidad que han facilitado algunas de las tareas necesarias para llevar a cabo este proceso. Sin embargo, si bien estas aplicaciones ofrecen una variedad de funcionalidades, ninguna de ellas aborda todas las necesidades de manera integral. Aun así, encontramos cierta similitud entre ellas, enfocándose todas en el mismo público objetivo, creando entre ellas una rivalidad directa. Es en este contexto donde nuestra propuesta, la aplicación de *scouting* de jugadores que estamos desarrollando busca llenar un vacío dejado por las demás aplicaciones existentes. Para cubrir este vacío nos enfocaremos plenamente en el desarrollo de una aplicación móvil que, si bien posiblemente no cubra todas las necesidades del *scouter*, sea una herramienta indispensable para facilitarle su trabajo.

En adelante, se llevará a cabo un análisis detallado de las características de las tres aplicaciones más relevantes y similares a la nuestra. A través de este análisis comparativo, se identificarán las fortalezas y debilidades de cada una, y se destacará cómo nuestra aplicación aborda de manera efectiva algunos de los puntos débiles identificados en las aplicaciones existentes.

1. Wyscout:

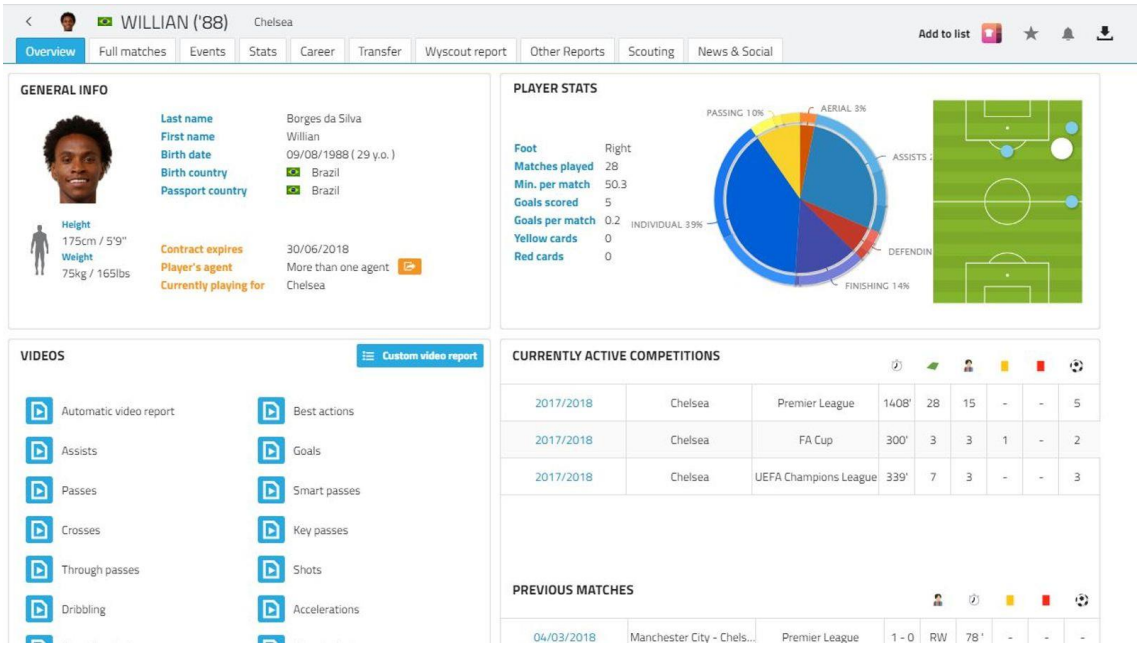


Figura 1 Captura Aplicación Wyscout

Wyscout [5-6] es la plataforma líder en el mundo del fútbol que proporciona un conjunto completo de herramientas para la observación, análisis y seguimiento de jugadores y equipos. Se destaca por ofrecer una extensa base de datos que abarca una amplia gama de competiciones y jugadores a nivel mundial. Su enfoque se centra en brindar a los usuarios acceso a estadísticas detalladas, videos de partidos y análisis tácticos, lo que permite a los cazatalentos,

entrenadores y clubes tomar decisiones informadas y estratégicas en el proceso de *scouting*.

Wyscout permite a los usuarios filtrar y buscar jugadores en función de diversas métricas y criterios, como edad, posición, rendimiento en partidos específicos y habilidades clave. Además, ofrece herramientas de análisis de vídeo avanzadas que utilizan tecnologías de reconocimiento de patrones y aprendizaje automático para evaluar acciones específicas y tácticas de juego. Esto proporciona una visión más profunda de las fortalezas y debilidades de los jugadores, lo que resulta en una selección más precisa de talentos emergentes.

La plataforma también permite a los usuarios seguir el progreso de los jugadores a lo largo del tiempo, lo que facilita el análisis de su desarrollo y progresión en el tiempo. Wyscout ha establecido una sólida reputación en el mundo del fútbol gracias a su amplia gama de funcionalidades, su enfoque en el análisis de vídeo avanzado y su extensa base de datos.

2. Scout7:

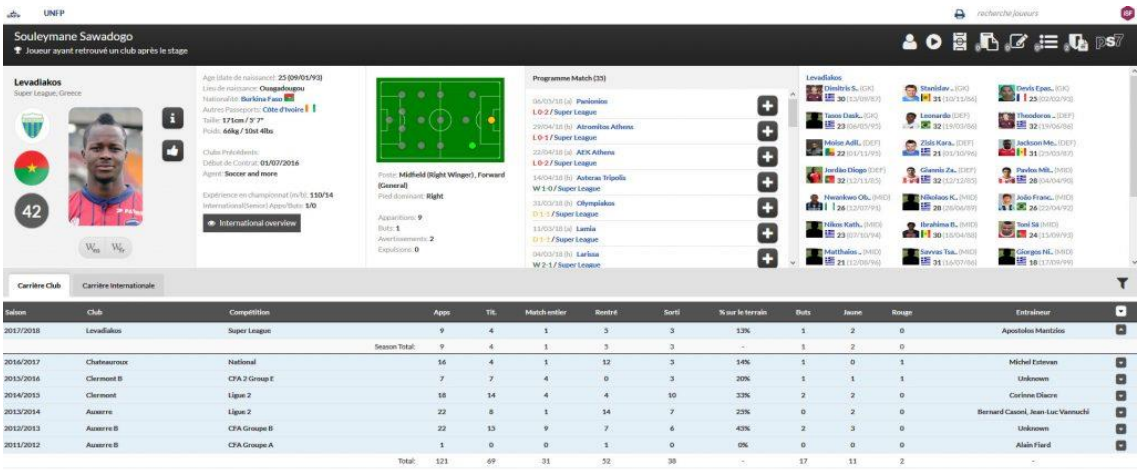


Figura 2 Captura Aplicación Scout7

Scout7 [7] es otra plataforma de referencia en el mundo del fútbol, que se especializa en proporcionar herramientas avanzadas de *scouting* y análisis para clubes, entrenadores y cazatalentos. Al igual que Wyscout, Scout7 ofrece una amplia base de datos que cubre una diversidad de competiciones y jugadores a nivel global.

Esta herramienta se distingue por su enfoque en la gestión y el seguimiento completo del proceso de *scouting*. Proporciona un sistema integral para capturar, almacenar y analizar datos relacionados con jugadores y equipos. Los usuarios pueden acceder a estadísticas detalladas, videos de partidos y análisis tácticos para evaluar el rendimiento y las habilidades de los jugadores en diversos contextos.

Una de las características destacadas de Scout7 es su capacidad para realizar comparativas entre jugadores y equipos, lo que facilita la toma de decisiones informadas. Además, ofrece informes personalizados y herramientas para el seguimiento del desarrollo de los jugadores, permitiendo una evaluación continua de su progreso y evolución en el tiempo.

Scout7 también brinda a los usuarios la posibilidad de personalizar sus búsquedas y filtros. La herramienta ha sido ampliamente adoptada en el ámbito del fútbol profesional y se ha ganado una sólida reputación gracias a su enfoque en la gestión integral.

3. InStat Scout:

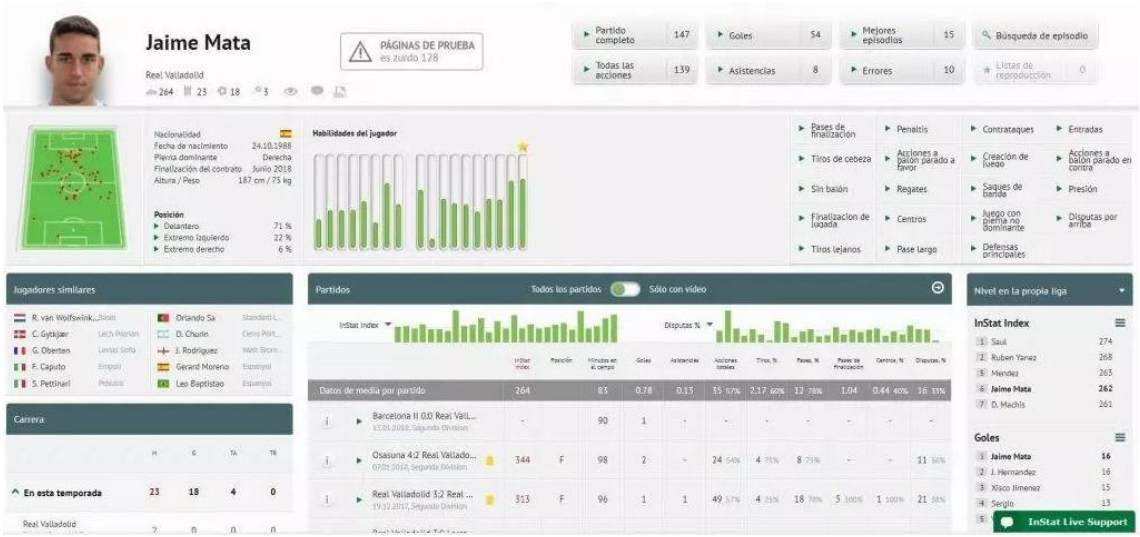


Figura 3 Captura Aplicación InStat Scout

InStat Scout [8-9] es otra destacada plataforma de *scouting* y análisis en el ámbito del fútbol. Al igual que Wyscout y Scout7, InStat Scout ofrece un conjunto completo de herramientas para la observación y seguimiento de jugadores y equipos. Su enfoque se centra en brindar análisis de vídeo avanzado y estadísticas precisas, lo que permite a los cazatalentos, entrenadores y clubes evaluar el rendimiento de los jugadores de manera detallada.

La plataforma proporciona una amplia base de datos que cubre una gran variedad de competiciones y jugadores en todo el mundo. Los usuarios pueden acceder a vídeos de partidos de alta calidad y análisis tácticos que utilizan tecnologías de reconocimiento de patrones y aprendizaje automático para identificar acciones específicas en el juego.

InStat Scout ofrece características útiles para filtrar y buscar jugadores según distintos criterios, como edad, posición, habilidades destacadas y rendimiento en partidos específicos. Además, brinda informes detallados y estadísticas completas que facilitan la comparación y evaluación de jugadores y equipos.

Una de las ventajas de InStat Scout es su interfaz de usuario intuitiva y fácil de usar, lo que agiliza el acceso a la información relevante y mejora la experiencia del usuario. También ofrece la posibilidad de realizar análisis comparativos entre jugadores y equipos, lo que ayuda en la toma de decisiones informadas.

	Wyscout	Scout7	InStat Scout	ScoutItApp
CAR1: Búsqueda y filtrado de jugadores				
- CAR1.1: Búsqueda por posición, edad, habilidades, etc.	Sí	Sí	Sí	SI
- CAR1.2: Filtros avanzados según criterios específicos	Sí	Sí	Sí	NO
CAR2: Visualización de estadísticas de partidos				
- CAR2.1: Estadísticas individuales y de equipo	Sí	Sí	Sí	NO
- CAR2.2: Detalles de goles, asistencias, pases, etc.	Sí	Sí	Sí	SI
CAR3: Seguimiento de jugadores en tiempo real				
- CAR3.1: Actualización en vivo de acciones y eventos	Sí	Sí	Sí	NO
- CAR3.2: Notificaciones de eventos importantes	Sí	Sí	Sí	NO
CAR4: Análisis táctico y estratégico				
- CAR4.1: Desglose de tácticas y formaciones	Sí	Sí	Sí	NO
- CAR4.2: Análisis de movimientos y posiciones	Sí	Sí	Sí	NO
CAR5: Comparativa de jugadores				
- CAR5.1: Comparación estadística y de habilidades	Sí	Sí	Sí	SI
- CAR5.2: Evaluación de fortalezas y debilidades	Sí	Sí	Sí	SI
CAR6: Generación de informes y presentaciones				
- CAR6.1: Creación de informes personalizados	Sí	Sí	Sí	NO
- CAR6.2: Generación de presentaciones visuales	Sí	Sí	Sí	NO
CAR7: Integración con otras herramientas de análisis				
- CAR7.1: Conexión con sistemas de videoanálisis	Sí	Sí	Sí	NO
- CAR7.2: Integración con bases de datos externas	Sí	Sí	Sí	NO
CAR8: Gestión de perfiles de jugadores				
- CAR8.1: Registro y actualización de datos personales	Sí	Sí	Sí	SI
- CAR8.2: Organización y categorización de jugadores	Sí	Sí	Sí	SI
CAR9: Alarmas y notificaciones configurables				
- CAR9.1: Configuración de alarmas personalizadas	No	No	No	SI
- CAR9.2: Notificaciones de jugadores que cumplen criterios específicos	No	No	No	SI
- CAR9.3: Alarmas basadas en criterios de rendimiento	No	No	No	SI
- CAR9.4: Alarmas interactivas con los mejores 11 de la semana mostrados sobre el campo	No	No	No	SI

Figura 4 Comparativa de aplicaciones

2.4. Propuesta

Como se ha podido observar en el resto de las aplicaciones que existen en el mercado, todas son muy parecidas entre sí, siendo todas potentes *suites* de herramientas con el objetivo de que los *scouters* tengan una gestión integral de su trabajo.

Nuestra propuesta ScoutIt está diseñada para abordar un nicho específico en el mercado del fútbol. A diferencia de las otras aplicaciones que se centran en proporcionar herramientas generales de observación y análisis de jugadores, ScoutIt se centra por su funcionalidad única de suscribirte a alarmas semanales de jugadores, ninguna de las aplicaciones de la competencia ha explotado este campo en cuestión.

Como hemos dicho, la principal característica de ScoutIt es la posibilidad de suscribirte a alarmas de jugadores. Un ejemplo de alarma sería "jugadores sub-21 que hayan jugado más de 90 minutos con su selección". De esta manera, cuando el usuario se suscribe a esta alarma, cada lunes, si se dan las condiciones para que haya una actualización de la alarma, el usuario recibiría en su teléfono móvil una lista actualizada con los nuevos jugadores que han cumplido esta característica. En este caso, la condición sería que se hayan jugado partidos de selecciones.

Una vez el usuario reciba los nuevos datos podrá decidir si algún jugador le resulta de interés con el objetivo de guardarlo en listas que el mismo usuario tiene la capacidad de crear y gestionar. Estas listas las llamamos WatchList, estas WatchList pueden contener los jugadores que el usuario crea que tienen mayor relevancia para él o para su equipo de trabajo, ya que estas podrán ser compartidas con otros usuarios o equipos de trabajo. De esta manera conseguimos que el *scouter* pueda en unos pocos minutos recibir información de su interés y si descubre algún jugador que pueda llegar a tener cierta relevancia para su trabajo guardarlo en una WatchList para poder hacer un mejor seguimiento de éste en el tiempo.

Además, ScoutIt contará con un servicio de creación de alarmas a petición del cliente de manera que el usuario pueda con una serie de valores tales como edad, posición, rendimiento en partidos específicos y otras características clave configurar sus propias alarmas personalizadas con el fin de eliminar trabajo mecánico de búsqueda en las aplicaciones de la competencia.

Podemos asegurar que ScoutIt se basará en una base de datos actualizada y completa de jugadores y equipos de diferentes competiciones, lo que garantizará que los usuarios obtengan datos precisos y confiables para su análisis. Y aunque ScoutIt no tiene muchas de las herramientas que ofrece la competencia, la aplicación se enfoca en brindar otro tipo de servicio que creemos que esta por explotar aún en el mercado. De esta manera, creemos que con nuestras alarmas los *scouter* podrán agilizar el proceso de detección de talentos y facilitar la identificación de jóvenes promesas que se ajusten a sus necesidades específicas, pudiendo también así contribuir al crecimiento y desarrollo de jóvenes futbolistas en la industria deportiva.

3. Análisis del problema

3.1. Identificación y análisis de posibles soluciones.

Como es de esperar, cuando se quiere realizar una aplicación, existe una fase de diseño donde se identifican sus problemas y necesidades, así que se tienen que tomar una serie de decisiones sobre la misma.

Para empezar, hay que remarcar que una de las necesidades que plateaba este proyecto, a parte de la creación de la aplicación de *scouting*, era la de que los administradores depusieran de un panel de gestión. Para ello, diferenciaremos siempre entre la aplicación destinada a los usuarios y la aplicación web destinada a que los administradores puedan llevar un control.

Al crear estas 2 aplicaciones, surge una necesidad más, que es la de crear una API que nos permita tener una conexión fluida que sirva de nexo común entre en el panel de administración, la aplicación destinada a los usuarios y la base de datos.

3.1.1. Plataforma App Usuarios

La plataforma escogida estaba bastante clara desde los inicios. Aunque se barajaron diversas opciones.

En primer lugar, se pensó en realizar una aplicación web siguiendo un poco la estela de las demás aplicaciones de *scouting* disponibles hoy en día en el mercado. Pero una vez definido, que la principal funcionalidad de la aplicación iba a ser las alarmas. Es decir, que el *scouter* pudiera recibir eventualmente cuando se actualizarán los datos de nuestra base, una alerta con los jugadores que han entrado o salido de sus alarmas. Se llegó a la conclusión que la mejor plataforma para llevar a cabo esto serían los dispositivos móviles.

El problema que nos encontramos aquí es que el desarrollo de aplicaciones nativas para las dos principales plataformas de telefonía móvil hoy en día (Apple y Android) difiere mucho una de otra, sin contar el coste de tiempo y recursos de tener que desarrollar la misma aplicación para dos plataformas distintas.

Por ello, la única solución viable para el desarrollo de una aplicación móvil con un rendimiento lo más parecido a una aplicación nativa y con una base de datos compartida por los usuarios, era utilizar un *framework* que nos acerque lo máximo posible a las características nativas de estas plataformas.

3.1.2. Compatibilidad App Usuarios

La compatibilidad resulta un aspecto fundamental de la aplicación. Dada la gran cantidad de dispositivos móviles que existen en la actualidad, con diferentes tipos de hardware, diferentes tipos de resolución de pantalla y distintos tipos de sistema operativo, resulta fundamental desarrollar la aplicación sobre una base que nos permita que la aplicación se ejecute de una manera óptima en cualquier tipo de dispositivo móvil. Esto es debido a que la filosofía del producto mínimo viable (PMV) y el marco de tiempo de un TFG no nos permite poder desarrollar una aplicación distinta para cada uno de los sistemas operativos móviles actuales, ni contamos con la posibilidad de poder ejecutar nuestra aplicación en un gran número de dispositivos para comprobar que todo se dispone y ajusta a las resoluciones y tamaños de pantalla de los distintos dispositivos del mercado. Por todo esto, se decide trabajar sobre un *framework* capaz de desplegar la aplicación en los distintos sistemas operativos y que nos permita programar la aplicación de una manera *responsive* con el objetivo de que se acople a la infinidad de tipos de pantalla que existen en los dispositivos móviles a día de hoy.

3.1.3. Solución propuesta App Usuarios.

Una vez decidida que la plataforma en la que desarrollaría la aplicación iba a ser los dispositivos móviles y analizados los distintos problemas que planteaba esta plataforma. Se decidió que se iba a utilizar un *framework* que nos brindara una aproximación lo más cercana posible a aplicaciones nativas tanto de Android como de IOS y que el mayor porcentaje de código de la aplicación fuera igual en ambos sistemas. Para ello se decidió utilizar el *framework* React Native.

React Native [13-14] es un *framework* de desarrollo de aplicaciones móviles que permite crear aplicaciones nativas utilizando JavaScript como lenguaje de programación.

La principal ventaja de React Native es que permite a los desarrolladores escribir una vez el código y luego ejecutarlo en múltiples plataformas, evitando tener que desarrollar y mantener dos aplicaciones separadas. Esto se debe a que React Native utiliza una combinación de componentes nativos de los sistemas además de componentes de React para construir la interfaz de usuario, lo que proporciona un rendimiento y una apariencia nativa en cada plataforma.

Otra de las principales ventajas de utilizar React Native es su capacidad para reutilizar código mediante componentes. Los componentes son bloques de construcción de la interfaz de usuario que encapsulan la lógica y la apariencia de una determinada funcionalidad o elemento visual.

En React Native, los componentes son altamente reutilizables. Esto significa que un componente creado para una parte de la aplicación puede ser utilizado en diferentes partes o incluso en otras aplicaciones sin tener que volver a escribir el código desde

cero. Esto ahorra tiempo y esfuerzo, ya que se evita la duplicación de código y se promueve la consistencia en la aplicación. Esto fomenta la modularidad y la escalabilidad del desarrollo de aplicaciones móviles. Al tener componentes bien estructurados y separados, es más fácil realizar cambios y mejoras en la aplicación sin afectar a otras partes.

3.1.4. Plataforma Panel de Gestión

La plataforma para el panel de gestión que utilizaran los administradores también estaba clara de un primer momento, ya que, si se tiene que hacer alguna gestión en la aplicación, la persona encargada casi con total seguridad estará trabajando en un ordenador. Aunque sí que es verdad que si surge algún problema y ningún administrador se encuentra delante del ordenador debería poderse acceder desde el móvil. Por todo esto se decidió que la manera de gestionar la aplicación seria mediante una aplicación web, con un diseño orientado al trabajo desde un ordenador, pero con la posibilidad de acceder desde un dispositivo móvil.

3.1.5. Compatibilidad *Panel de Gestión*

La compatibilidad resulta un aspecto fundamental también en el panel de gestión, ya que tiene que poder ejecutarse en una gran variedad de navegadores tanto de escritorio como en su versión móvil. Por ello se decidió hacer como en la aplicación de usuarios y optar por un *framework* que nos ofreciera la facilidad de crear una aplicación web pudiendo abstraer la compatibilidad con dispositivos y pudiendo centrar el esfuerzo en el desarrollo.

3.1.6. Solución propuesta Panel de Gestión.

En esta ocasión debido a las necesidades de compatibilidad y con la experiencia de trabajo que se tenía con la aplicación de usuarios se decidió por optar por la variante de React [15] para el desarrollo de aplicaciones web React.js.

React.js es una biblioteca de JavaScript desarrollada por Facebook que se utiliza para construir interfaces de usuario interactivas y reactivas. Se basa en un enfoque declarativo, lo que significa que los desarrolladores describen cómo debería verse y comportarse la interfaz de usuario, y React se encarga de manejar eficientemente los cambios y actualizaciones necesarios.

La principal característica de React.js al igual que en React Native es la utilización de componentes reutilizables. Estos componentes se pueden combinar para formar

aplicaciones complejas, lo que nos facilita la estructuración y el mantenimiento del código. Además, React.js es altamente escalable y flexible.

3.1.7. Comunicación entre las distintas Aplicaciones.

Como es de esperar al tener más de una aplicación en el proyecto, tener una gestión de usuarios y tener que enviar y recibir información desde cada una de ellas, se necesitaba de un sistema que nos sirviera de centro de intercomunicación entre todas las partes implicadas, tanto las aplicaciones como la base de datos.

Por esto se decide que lo mejor es tener un servidor con una API con la cual mediante el protocolo HTTP nos permita hacer peticiones y recibir respuestas para interactuar con los datos de manera eficiente y segura. El uso de una API permite una comunicación fluida y estructurada entre el servidor y la aplicación cliente, garantizando la integridad y la seguridad de los datos.

El protocolo HTTP (Hypertext Transfer Protocol) es utilizado ampliamente en la comunicación entre el cliente y el servidor. Mediante solicitudes HTTP, como GET, POST, PUT o DELETE, se pueden enviar datos al servidor, realizar consultas y manipular recursos.

Al tener un servidor con una API, se puede establecer una arquitectura de cliente-servidor donde la aplicación cliente (en este caso, la aplicación de *scouting* o el panel de gestión) puede enviar solicitudes a la API para obtener, modificar o almacenar datos. Esto permite una mayor flexibilidad en el manejo de la información y facilita la expansión y escalabilidad del sistema.

Además, al utilizar una API, se pueden implementar medidas de autenticación y autorización para garantizar que solo los usuarios autorizados puedan acceder y manipular los datos. Esto brinda un nivel adicional de seguridad y control sobre la aplicación.

3.1.7. Solución propuesta para la comunicación.

Para poder crear nuestra API, se ha decidido que lo mejor es crear un servidor que la aloje y gestione las peticiones. Este servidor estará basado en Node.js y Express.

Node.js es un entorno de ejecución de JavaScript del lado del servidor que permite desarrollar aplicaciones web escalables y de alto rendimiento. Express, por otro lado, es un *framework* web rápido y minimalista para Node.js que facilita la creación de API y el manejo de rutas y solicitudes HTTP. La elección de Node.js y Express como tecnologías de servidor se debe a su eficiencia, escalabilidad y su capacidad para manejar grandes volúmenes de solicitudes. Estas tecnologías son ampliamente utilizadas en el desarrollo de aplicaciones web y son especialmente adecuadas para construir API robustas y flexibles.

En cuanto a la base de datos, se utilizará MongoDB, que es una base de datos NoSQL orientada a documentos. MongoDB se caracteriza por su flexibilidad y escalabilidad, lo que lo hace ideal para el almacenamiento de datos en aplicaciones que manejan una gran cantidad de información. Su estructura basada en documentos permite almacenar y consultar datos de forma eficiente y sin una estructura fija predefinida, lo que brinda flexibilidad en la evolución del esquema de la base de datos.

Para la conexión y el modelado de datos en MongoDB, se utilizará Mongoose. Mongoose es una biblioteca de modelado de objetos de MongoDB para Node.js que proporciona una interfaz sencilla y intuitiva para interactuar con la base de datos. Con Mongoose, se pueden definir esquemas de datos, realizar operaciones de CRUD (crear, leer, actualizar y eliminar) de manera sencilla y establecer validaciones y relaciones entre los datos.

Con todo esto tendremos un servidor que nos servirá de nodo de comunicación entre las distintas aplicaciones de manera que podemos enviar cualquier tipo de petición a éste, abstrayendo a las aplicaciones de gestionar cualquier tipo de comunicación con la base de datos. De esta manera, podremos garantizar la integridad.

También con esto nos aseguramos de que la aplicación pueda tener una escalabilidad adecuada. Al no tener una estimación del volumen de usuarios que tendremos hasta que no salga al mercado, se considera que es necesario garantizar una escalabilidad sencilla desde los primeros pasos de la aplicación para no vernos en problemas si esta crece en volumen una vez esté publicada.

3.3. Especificación de requisitos.

3.3.1. Requisitos funcionales

A continuación, se procede a describir los requisitos funcionales que deben formar parte de la aplicación:

RF1 – Gestionar alarmas en la aplicación, este requisito engloba todas las funcionalidades relacionades con la visualización e interacción con las alarmas.

RF2 - Gestión alarmas en el Panel de Control, este requisito engloba las funcionalidades relacionadas con operaciones CRUD y actualización de alarmas.

RF3 - Gestión de jugadores en la APP, este requisito engloba las funcionalidades relacionadas con la visualización de los jugadores e interacción con ellos, así como la búsqueda de estos.

RF4 - Gestión de WatchLists en la APP, este requisito engloba las funcionalidades relacionadas con la interacción del usuario con estas listas personalizadas.

RF5 - Gestión de WatchLists en el Panel de Control, este requisito engloba las funcionalidades relacionadas con la visualización de información importante y gestión de las WatchList por parte de los administradores

Requisitos funcionales	Casos de uso
RF1	CU 1
RF2	CU2
RF3	CU3
RF4	CU4
RF5	CU5

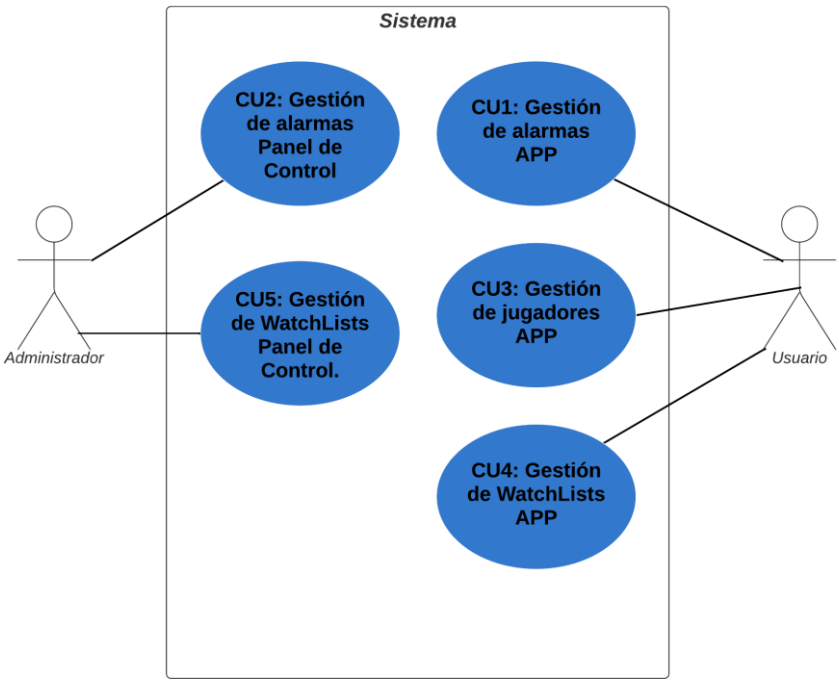


Figura 5 Casos de uso del sistema

CU1: Gestión de alarmas APP

CU1.1: Ver una Alarma

Descripción	El usuario tiene que ser capaz de poder visualizar los jugadores que hay dentro de una alarma.
Precondición	Haber iniciado sesión en una cuenta, tener alarmas suscritas en la lista del usuario.
Secuencia principal	1. El usuario entra en su lista de alarmas. 2. Selecciona una y entra dentro de ella pulsando en cualquier parte de ella (título, imagen). 3. Visualiza una lista de cartas de jugadores si la alarma es tipo lista, o un terreno de juego con los jugadores colocados en sus respectivas posiciones si la alarma es tipo terreno de juego.
Alternativas/Errores	
Post condición	

CU1.2: Activar notificaciones de una Alarma

Descripción	El usuario tiene que ser capaz de poder activar las notificaciones *push* de una alarma
Precondición	Haber iniciado sesión en una cuenta, tener alarmas suscritas en la lista del usuario.
Secuencia principal	1. El usuario entra en su lista de alarmas.

	2. Selecciona una y entra dentro de ella pulsando en cualquier parte de ella (título, imagen). 3. Presiona el botón con la imagen de una campana para activar las notificaciones. 4. El sistema notificará al usuario con una notificación *push* en su smartphone cuando esta alarma reciba una actualización. 5. El botón de la alarma cambia de color para indicar que está activada
Alternativas/Errores	
Post condición	El sistema activa las notificaciones *push* para ese usuario en esa alarma en concreto.

CU1.3: Desactivar notificaciones de una Alarma

Descripción	El usuario tiene que ser capaz de poder desactivar las notificaciones *push* de una alarma
Precondición	Haber iniciado sesión en una cuenta, tener alarmas suscritas en la lista del usuario.
Secuencia principal	1. El usuario entra en su lista de alarmas. 2. Selecciona una y entra dentro de ella pulsando en cualquier parte de ella (título, imagen). 3. Presiona el botón con la imagen de una campana para desactivar las notificaciones. 4. El sistema dejara de notificar al usuario con una notificación *push* en su smartphone cuando esta alarma reciba una actualización. 5. El botón de la alarma cambia de color para indicar que está desactivada
Alternativas/Errores	
Post condición	El sistema desactiva las notificaciones *push* para ese usuario en esa alarma en concreto.

CU2: Gestión de alarmas Panel de Control

CU2.1: Crear una Alarma de manera manual.

Descripción	El administrador debe de ser capaz de crear una alarma manualmente añadiendo nombre, jugadores, usuarios que la tienen activa y el tipo de alarma
Precondición	Haber iniciado sesión en una cuenta de administrador en la aplicación web
Secuencia principal	1. El administrador selecciona el panel de creación de alarmas 2. Selecciona los jugadores que contendrá esta alarma 3. Selecciona que tipo de alarma será si de tipo lista o de tipo terreno de juego. 4. Selecciona los usuarios que tendrán esta alarma

	5. Establece un nombre a la alarma 6. Pulsa el botón de crear alarma
Alternativas/Errores	2.1 Si la lista de jugadores que contendrá la alarma se encuentra vacía el sistema lanza un error con un aviso de que la alarma tiene que contener al menos a un jugador. 3.1 Si el tipo de alarma se encuentra vacío el sistema lanza un error con un aviso de que la alarma debe tener seleccionado un tipo. 4.1 Si el campo nombre de alarma se encuentra vacío el sistema lanza un error con un aviso de que la alarma debe tener un nombre.
Post condición	El sistema agrega la Alarma a la lista de alarmas.

CU2.2: Crear una Alarma de manera automática.

Descripción	El administrador debe de ser capaz de crear una alarma automáticamente adjuntando un archivo csv exportado desde un programa de cálculo estadístico como R Studio. Aun así debe de ser posible la modificación del nombre, jugadores, usuarios que la tienen activa y el tipo de alarma
Precondición	Haber iniciado sesión en una cuenta de administrador en la aplicación web
Secuencia principal	1. El administrador selecciona el panel de creación de alarmas 2. Presiona el botón importar fichero. 3. Selecciona el archivo en su explorador de archivos, 4. El sistema carga todos los valores automáticamente con los valores contenidos dentro del archivo CSV.
Alternativas/Errores	2.1 Si la lista de jugadores que contendrá la alarma se encuentra vacía el sistema lanza un error con un aviso de que la alarma tiene que contener al menos a un jugador. 3.1 Si el tipo de alarma se encuentra vacío el sistema lanza un error con un aviso de que la alarma debe tener seleccionado un tipo. 4.1 Si el campo nombre de alarma se encuentra vacío el sistema lanza un error con un aviso de que la alarma debe tener un nombre.
Post condición	El sistema agrega la Alarma a la lista de alarmas.

CU2.3: Suscribir a un usuario a una Alarma de manera manual.

Descripción	El administrador debe de ser capaz de poder suscribir a un usuario a una alarma.
Precondición	Haber iniciado sesión en una cuenta de administrador en la aplicación web
Secuencia principal	1. El administrador selecciona el panel de visualización de alarma. 2. Selecciona la alarma deseada de la lista de posibles. 3. Pulsa el botón ver más información. 4. Pulsa el botón modificar usuarios. 5. En el campo de búsqueda de usuarios, escribe el nombre del usuario. 6. En la lista de resultados marca la casilla correspondiente para suscribir al usuario.
Alternativas/Errores	2.1 Si el usuario no existe o está mal escrito la lista de usuarios aparecerá vacía.
Post condición	El sistema suscribe al usuario a la Alarma.

CU2.4: Eliminar la suscripción de un usuario a una Alarma de manera manual.

Descripción	El administrador debe de ser capaz de poder eliminar la suscripción de un usuario a una alarma.
Precondición	Haber iniciado sesión en una cuenta de administrador en la aplicación web
Secuencia principal	1. El administrador selecciona el panel de visualización de alarma. 2. Selecciona la alarma deseada de la lista de posibles. 3. Pulsa el botón ver más información. 4. Pulsa el botón modificar usuarios. 5. En el campo de búsqueda de usuarios, escribe el nombre del usuario. 6. En la lista de resultados desmarca la casilla correspondiente para eliminar la suscripción del usuario.
Alternativas/Errores	2.1 Si el usuario no existe o está mal escrito la lista de usuarios aparecerá vacía.
Post condición	El sistema elimina la suscripción del usuario a la Alarma.

CU3: Gestión de jugadores APP

CU3.1 Buscar jugadores.

Descripción	El usuario tiene que ser capaz de poder buscar jugadores insertando algunos parámetros deseados como (liga, equipo, edad o posición)
Precondición	Haber iniciado sesión en una cuenta
Secuencia principal	1. El usuario se dirige al apartado de búsqueda 2. Introduce los filtros deseados 3. El sistema devuelve una lista con las cartas de los jugadores que concuerdan con los parámetros de búsqueda.
Alternativas/Errores	Si los parámetros de búsqueda dan como resultado una búsqueda que no obtiene resultado la lista se muestra vacía
Post condición	El sistema muestra los jugadores encontrados.

CU3.2 Ver más información del jugador

Descripción	El usuario tiene que ser capaz de poder ver información más detallada del jugador.
Precondición	Haber iniciado sesión en una cuenta
Secuencia principal	1. El usuario tiene delante una carta de jugador en cualquier parte de la aplicación donde se renderiza este componente. 2. Pulsa en el botón ver más información 3. El sistema devuelve una pantalla con información detallada de su rendimiento deportivo de la temporada.
Alternativas/Errores	
Post condición	El sistema abre una ventana modal con la información del jugador.

CU3.3 Añadir a un jugador a la WatchList

Descripción	El usuario tiene que ser capaz de poder añadir a su WatchList a cualquier jugador
Precondición	Haber iniciado sesión en una cuenta
Secuencia principal	1. El usuario tiene delante una carta de jugador en cualquier parte de la app donde se renderiza este componente. 2. Pulsa en el botón Like 3. El sistema devuelve una pantalla con las diferentes WatchList que tiene el usuario pudiendo agregar al jugador a la que le sea más conveniente.

Alternativas/Errores	
Post condición	El sistema añade al jugador a la WatchList que el usuario ha designado.

CU3.4 Eliminar a un jugador de la WatchList

Descripción	El usuario tiene que ser capaz de poder eliminar de sus WatchList a cualquier jugador.
Precondición	Haber iniciado sesión en una cuenta
Secuencia principal	1. El usuario tiene delante una carta de jugador en cualquier parte de la aplicación donde se renderiza este componente. 2. Pulsa en el botón Like 3. El sistema devuelve una pantalla con las diferentes WatchList que tiene el usuario pudiendo eliminar al jugador de la WathList si es que este se encuentra en alguna.
Alternativas/Errores	
Post condición	El sistema borra al jugador a la WatchList que el usuario ha designado.

CU4: Gestión de WatchLists APP

CU4.1 Crear WatchLists.

Descripción	El usuario tiene que ser capaz de poder crear nuevas WatchList desde la aplicación
Precondición	Haber iniciado sesión en una cuenta
Secuencia principal	1. El usuario se dirige al apartado de WatchList 2. Le da al botón de crear nueva WatchList 3. El sistema abre una ventana modal. 4. El usuario rellena el nombre de la WatchList 5. El usuario le da al botón crear. 6. El sistema crea una nueva WatchList y la asocia al usuario.
Alternativas/Errores	Si el campo nombre se queda vacío al intentar crear la WatchList salta un error avisando al usuario que el campo nombre no puede estar vacío.
Post condición	El sistema crea la WatchList y la asocia al usuario.

CU4.2 Borrar WatchLists.

Descripción	El usuario tiene que ser capaz de poder borrar WatchLists desde la aplicación
Precondición	Haber iniciado sesión en una cuenta
Secuencia principal	1. El usuario se dirige al apartado de WatchList 2. Selecciona una WatchList de la lista. 3. El usuario le da al botón borrar. 4. El sistema lanza una advertencia de la WatchList va a ser borrada 5. El sistema elimina la WatchList del usuario.
Alternativas/Errores	
Post condición	El sistema elimina la WatchList del usuario.

CU4.3 Borrar Jugadores de la WatchList.

Descripción	El usuario tiene que ser capaz de poder borrar jugadores de la WatchLists desde la aplicación
Precondición	Haber iniciado sesión en una cuenta
Secuencia principal	1. El usuario se dirige al apartado de WatchList 2. Selecciona una WatchList de la lista. 3. El usuario le da al botón borrar dentro de cada carta de jugador. 4. El sistema lanza una advertencia de que el jugador va a ser borrado 5. El sistema elimina al jugador de la WatchList del usuario.
Alternativas/Errores	
Post condición	El sistema elimina al jugador de la WatchList del usuario.

CU5: Gestión de WatchLists Panel de Control.

CU5.1 Consultar WatchLists en la base de datos.

Descripción	El administrador debe de ser capaz de visualizar todas las WatchList que hay en la base de datos y poder filtrarlas
Precondición	Haber iniciado sesión en una cuenta de administrador en la aplicación web
Secuencia principal	1. El administrador selecciona el panel de WatchLists 2. El sistema muestra una lista con todas las WatchLists 3. El administrador puede filtrar por jugadores 4. El administrador puede filtrar por usuario.
Alternativas/Errores	

Post condición	

CU5.2 Consultar información más detallada de las WatchLists en la base de datos.

Descripción	El administrador debe de ser capaz de visualizar todos las que jugadores tiene la WatchList y a que usuario pertenece
Precondición	Haber iniciado sesión en una cuenta de administrador en la aplicación web
Secuencia principal	1. El administrador selecciona el panel de WatchLists 2. El sistema muestra una lista con todas las WatchLists 3. El administrador selecciona una de ellas. 4. El sistema muestra los jugadores que hay dentro de ella junto con el usuario propietario de esta.
Alternativas/Errores	
Post condición	

CU5.3Consultar nuevos jugadores añadidos a las WatchLists en la base de datos.

Descripción	El administrador debe de ser capaz de visualizar que jugadores se han añadido y a que WatchList
Precondición	Haber iniciado sesión en una cuenta de administrador en la aplicación web
Secuencia principal	1. El administrador selecciona el panel de Logs 2. El sistema muestra una lista con todas los Logs 3. El administrador selecciona una de ellas. 4. El sistema muestra el jugador contiene el *log*. 5. El sistema muestra a qué hora se ha añadido el jugador 6. El sistema muestra otros jugadores que contiene esa WatchList 7. El sistema muestra el nombre de la WatchList
Alternativas/Errores	

3.3.2 Requisitos no funcionales

RNF1: Interfaz intuitiva

Descripción	La interfaz debe ser lo bastante intuitiva como para que un usuario nuevo pueda utilizarla sin necesidad de consultar ninguna guía.

RNF2: Interfaz responsive

Descripción	La interfaz debe ajustarse adecuadamente a la pantalla del dispositivo sin importar su resolución de pantalla de manera que los elementos de la interfaz se ajusten de manera responsive

RNF3: Disponibilidad ininterrumpida

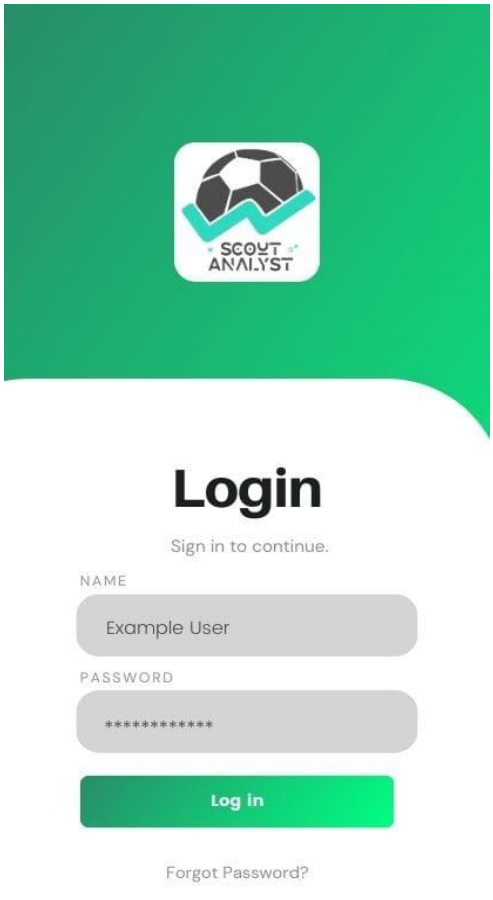
Descripción	El sistema tiene que estar disponible la mayor cantidad de tiempo, los *scouters* de jugadores trabajan en una gran cantidad de franjas horarias por lo que la disponibilidad de los servidores que proveen la información a la aplicación tiene que ser total.

RNF4: Confirmación de acciones críticas

Descripción	Cualquier acción que conlleve la eliminación de información o conlleve algún tipo de acción irreversible o de difícil restauración tiene que venir acompañadas de una confirmación por parte del usuario

3.4. Prototipos

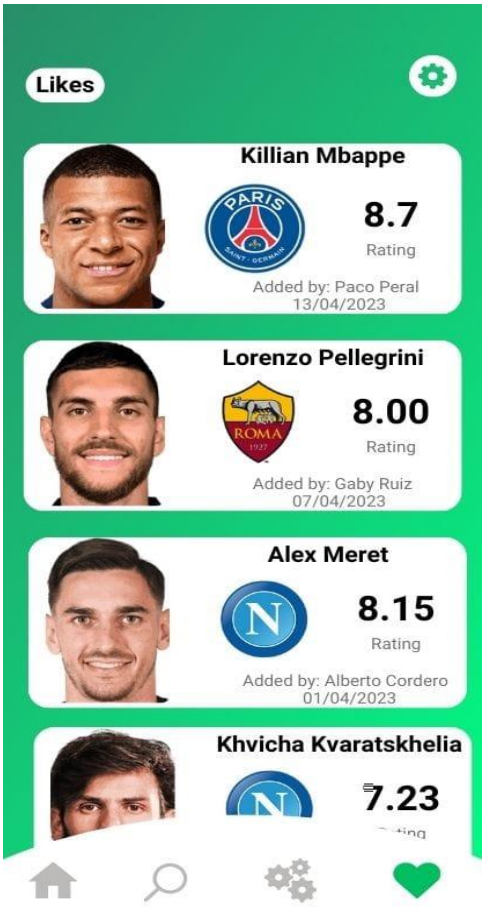
Durante las primeras reuniones que se hicieron con la empresa que tiene pensado utilizar esta aplicación se intentó hacer un pequeño borrador antes de poder realizar un documento de especificación de requisitos. Con este documento se intentaba plasmar de manera un poco general el rumbo que debía tomar el proyecto y hacia donde queríamos ir. En él se dejó claro que lo más idóneo seria crear una herramienta que los *scouter* pudieran tener siempre encima por lo que la plataforma sobre la cual se desarrollaría la aplicación quedo clara rápidamente. También en este documento, quedó claro que los tres pilares fundamentales de la aplicación serían, la búsqueda de jugadores, la gestión de éstos mediante listas personalizables WatchList y la suscripción a Alarmas. Con toda la información recabada en estas reuniones se diseñó un primer prototipo en Canva Estudio, que sirvió de guía para el posterior diseño dentro de React.



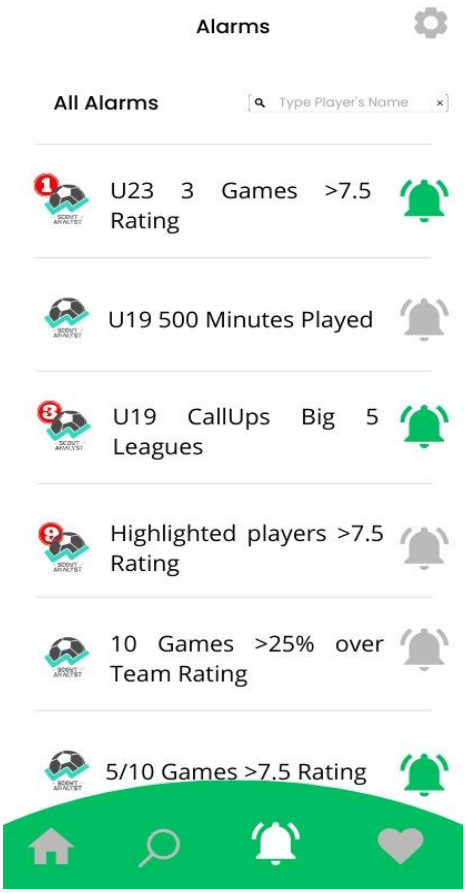
Mockup 1 Pantalla Inicio de Sesión

Mockup 2 Pantalla Principal

Mockup 3 Pantalla Búsqueda de Jugadores



Mockup 4 Pantalla WatchList



Mockup 5 Pantalla Alarmas

Aunque desde los primeros prototipos la aplicación final ha ido evolucionando y cambiando en algunos aspectos, se puede ver ya cual es el flujo de trabajo con la aplicación.

Lo primero que se encuentran los usuarios es la pantalla de inicio de sesión (Mockup 1), una vez introducen su usuario y contraseña la aplicación les redirige a la pantalla principal o home de la aplicación.

Como podemos ver en el menú de abajo tenemos 4 botones. Estos se encargan al completo de la navegación de la aplicación ya que el resto de las pantallas que aparecen se abren de manera modal cuando estamos en alguno de los 4 paneles principales. Como se puede observar el primer botón es el botón Home que nos redirigiría al Mockup 1. En esta pantalla principal en primer lugar el usuario puede ver un breve resumen de las estadísticas de uso de la aplicación que ha tenido, seguido de algunas WatchLists que el usuario tiene creadas y por último una pequeña carta de jugador donde la aplicación muestra un jugador que ha tenido especial relevancia en el día.

El segundo botón es el botón de búsqueda que nos redirigiría al panel de búsqueda de jugadores en el cual mediante la introducción de una serie de simples parámetros de búsqueda nos devolvería una lista de jugadores que cumpliría con los criterios de búsqueda.

El tercer botón, en un primer lugar, iba a estar destinado a la configuración que el usuario pudiera hacer de su perfil, pero a los pocos días de las primeras reuniones se decidió que lo mejor era que este botón estuviera destinado al panel de alarmas por lo que lo hemos incluido en esta primera versión de Mockups de la aplicación. Este panel de alarmas muestra todas las alarmas existentes, con un pequeño filtro en la parte superior de manera que el usuario pueda buscar alarmas. Además, al lado de cada alarma existe un botón en forma de campana que habilita al usuario a decidir que alarmas quiere que le sean notificadas cuando estas reciban una actualización.

Por otra parte, el ultimo botón de la navegación nos conduce al apartado de WatchLists. En este apartado el usuario puede ver que jugadores ha guardado en su WatchList para poder tener un seguimiento más cercano de algún jugador que le sea de especial interés.

3.5. Plan de trabajo

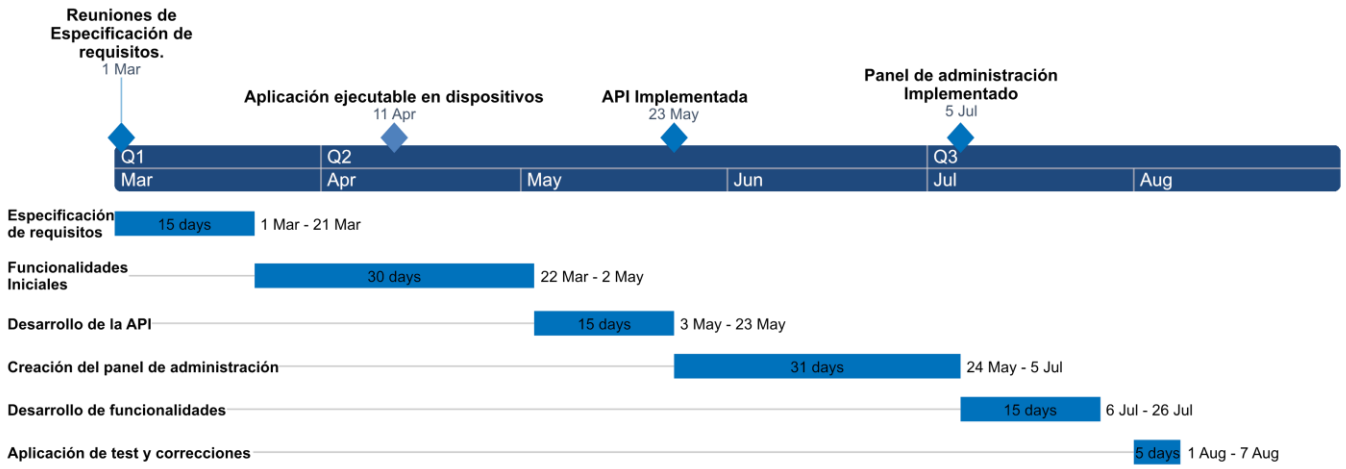


Figura 6 tablero Kanban

Para el desarrollo del proyecto se ha planteado el uso de la metodología ágil basada en *sprints* de 3 semanas. Para ello en un tablero *Kanban* se han definido todas las unidades de trabajo en las que se subdividió el proyecto según los documentos de especificación de requisitos iniciales. A estas unidades de trabajo se les han definido una estimación en horas y el resultado de esta estimación ha sido el diagrama de Gantt que podemos observar en la figura 6.

3.6. Presupuesto

Al inicio del desarrollo del proyecto de la aplicación de *scouting*, se realizó un plan de costes proyectados para contar con una estimación aproximada de lo que podía llegar a costar desarrollar el proyecto. Como desarrollador del proyecto se han asumido los costes asociados con el hardware, la electricidad y otros gastos operativos, lo que ha permitido reducir significativamente los costes totales del proyecto.

En cuanto al salario del desarrollador, se ha establecido un salario limpio de 1500 euros mensuales. Teniendo en cuenta los impuestos y retenciones aplicados en España, el coste bruto mensual del desarrollador es de aproximadamente 1 968,83 euros. A lo largo de los siete meses de desarrollo del proyecto, este coste salarial representa una parte significativa del presupuesto total.

En cuanto al costo del servidor, se ha optado por utilizar un servidor VPS que tiene un coste de 49,90 euros al mes. Este servidor es utilizado para alojar la base de datos MongoDB y la API, asegurando una comunicación rápida y segura entre la aplicación y el servidor.

En resumen, el coste total del proyecto de la aplicación de *scouting*, considerando el salario del desarrollador y el costo del servidor durante los siete meses de desarrollo, es de aproximadamente 14 131,11 euros.

3.7. Análisis de riesgos

Durante la etapa inicial del proyecto, se realizó un análisis exhaustivo de los posibles riesgos que podrían afectar el desarrollo de la aplicación de *scouting*. A continuación, se presentan los riesgos identificados:

1. **Retrasos en el desarrollo**: Existe la posibilidad de que algunos aspectos del proyecto puedan requerir más tiempo del previsto inicialmente, lo que podría resultar en retrasos en la entrega del producto final.

2. **Problemas de compatibilidad**: Dado que el proyecto involucra el uso de diferentes tecnologías y plataformas, puede surgir la dificultad de asegurar la compatibilidad adecuada entre ellas.

3. **Seguridad de datos**: La seguridad de los datos de los usuarios es un factor crítico en una aplicación como esta. La posibilidad de brechas de seguridad o acceso no autorizado a la información debe ser considerada. En el mundo en el que se centra esta aplicación, el interés de un club o un *scouter* por un jugador puede significar muchos miles de euros en juego si se enterase otro club competidor, por lo que esta aplicación, si llega a escalar lo suficiente, puede ser susceptible de intentos de acceso no autorizado a información valiosa.

4. **Cambios en los requisitos: durante el desarrollo**: Los requisitos pueden cambiar o surgir nuevas funcionalidades solicitadas por el cliente, lo que podría afectar a la planificación y a los recursos, de manera que el tiempo de desarrollo se vea afectado.

5. **Falta de experiencia en tecnologías específicas**: La falta de experiencia o conocimientos en algunas tecnologías utilizadas en el proyecto podría ralentizar el desarrollo o afectar a la calidad del producto final. Al final, en este TFG todas las tecnologías implementadas son de nuevo conocimiento para mí, por lo que la posibilidad de una implementación con defectos es una posibilidad real.

6. **Problemas de escalabilidad:** Si la aplicación no está diseñada para manejar grandes cantidades de usuarios o datos, podría enfrentar problemas de rendimiento y escalabilidad en el futuro.

7. **Falta de retroalimentación del cliente:** La falta de una retroalimentación o comunicación constante con el cliente podría llevar a malentendidos o dificultades para cumplir con las expectativas del cliente.

Para mitigar estos riesgos, se implementaron diversas estrategias a lo largo del desarrollo del proyecto. Por ejemplo, se asignaron tiempos adicionales en la planificación para afrontar posibles retrasos y cambios en los requisitos. Se realizó una investigación exhaustiva de las tecnologías utilizadas, y en caso de falta de experiencia en alguna de ellas, se dedicó tiempo para capacitarse adecuadamente. Además, se implementaron medidas de seguridad de acuerdo con los estándares para proteger los datos de los usuarios y se estableció una comunicación constante con la empresa interesada en el desarrollo, para asegurar una retroalimentación oportuna y una comprensión clara de sus necesidades.

4. Diseño de la solución

4.1 Arquitectura del sistema

Durante el desarrollo del proyecto se implementará una arquitectura del sistema diseñada para asegurar la eficiencia, escalabilidad y facilidad de mantenimiento.

Para ello se ha optado por una arquitectura cliente-servidor con API. Esta arquitectura sigue un modelo de diseño en el desarrollo de software que se basa en la interacción entre dos componentes clave: el cliente y el servidor, conectados a través de una API. Esta arquitectura se utiliza ampliamente para crear sistemas y aplicaciones que requieren comunicación y colaboración entre diferentes partes, optimizando la organización, la escalabilidad y la eficiencia en el intercambio de datos y servicios.

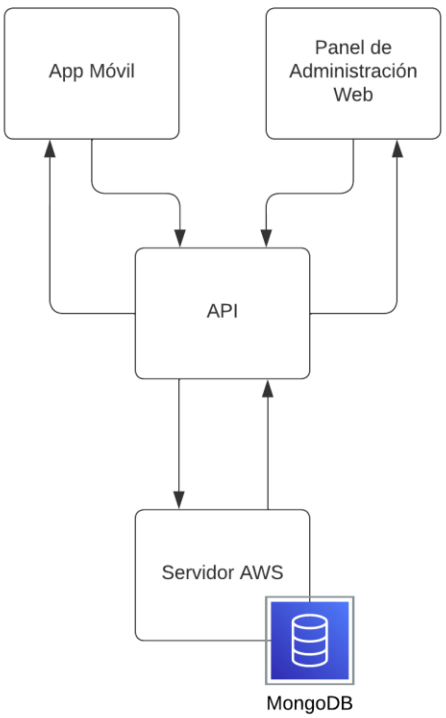


Figura 7 Arquitectura del Sistema

En esta arquitectura, contamos con **4 partes principales**:

- **Aplicación Móvil (Cliente):** Esta aplicación se ejecuta en dispositivos móviles y se conecta a la API para obtener y enviar datos. Actúa como un cliente

que solicita información específica y realiza acciones desde dispositivos móviles, como smartphones o tabletas.

- **Panel de Administración Web (Cliente):** El panel de administración web también funciona como un cliente. Los administradores utilizan esta interfaz web para interactuar con el sistema, acceder a informes y realizar acciones de gestión. Al igual que la aplicación móvil, se conecta a la API para obtener y enviar datos.

- **API (Servidor):** La API se encuentra en el lado del servidor y actúa como un intermediario entre las aplicaciones cliente y la base de datos. Recibe las solicitudes de las aplicaciones cliente, procesa las solicitudes y devuelve las respuestas correspondientes. La API proporciona una forma estructurada de comunicación y define cómo las aplicaciones cliente pueden interactuar con el servidor.

- **Base de Datos MongoDB en AWS (Servidor):** La base de datos se encuentra en un entorno de servidor, alojada en AWS. Almacena y organiza los datos que las aplicaciones cliente necesitan. La API se conecta a esta base de datos para recuperar y almacenar datos según lo solicitado por las aplicaciones cliente.

Ventajas de la Arquitectura Cliente-Servidor con API:

- **Separación de Preocupaciones:** Las aplicaciones cliente (móvil y web) se centran en la presentación y la interacción, mientras que el servidor (API y base de datos) se encarga del almacenamiento y la gestión de datos.

- **Acceso Controlado a los Datos:** Las aplicaciones cliente solo acceden a los datos a través de la API, lo que permite un control más preciso y seguro sobre el acceso a la información.

- **Flexibilidad en la Implementación**: Las aplicaciones cliente pueden evolucionar y mejorarse por separado sin afectar al otro componente. Los cambios en la API pueden gestionarse independientemente.

- **Escalabilidad Adecuada:** Puedes escalar el servidor (API y base de datos) según las necesidades de rendimiento, manteniendo un equilibrio en la distribución de recursos.

4.2 Diseño detallado

4.2.1. Diseño del flujo de datos

Dentro de la estructura del proyecto el flujo de datos es un parte esencial que permite una comunicación coherente y eficiente entre las diversas entidades involucradas. La interacción entre las aplicaciones *frontend* (la aplicación móvil y el panel de administración web) y el servidor *backend* se fundamenta en el uso de peticiones HTTP, las cuales facilitan el diálogo entre los distintos componentes. HTTP se selecciona por su capacidad de definir cómo los mensajes de solicitud y respuesta deben estructurarse y ser transmitidos. Además HTTP, con sus distintos métodos de solicitud (GET, POST, PUT, DELETE, entre otros) y códigos de estado (200 OK, 404 Not Found, 500 Internal Server Error), proporciona un abanico de herramientas para gestionar la comunicación y mantener la integridad del flujo de datos.

Además, la información intercambiada se presenta y encapsula en el formato JSON (JavaScript Object Notation), lo que contribuye a la uniformidad y legibilidad del intercambio de datos.

Enlace entre Aplicaciones Frontend y Servidor Backend:

- 1 . Iniciación de la Comunicación: El proceso comenzará cuando una aplicación *frontend*, ya sea la aplicación móvil o el panel de administración web, inicie una comunicación. Esta comunicación implicara una solicitud específica, como obtener datos de jugadores, crear alarmas o realizar cualquier acción en el sistema.

- 2. Transmisión de la Solicitud HTTP: Para transmitir esta solicitud al servidor *backend*, la aplicación *frontend* enviara una solicitud HTTP.

- 3. Procesamiento en el servidor Backend: Cuando el servidor *backend*, en este caso la API, reciba la solicitud HTTP, entra en acción. Analizará los detalles de la solicitud para entender la naturaleza de la acción solicitada y determina la mejor manera de responder.

- **4. Acceso a la Base de Datos:** La API según el caso procederá a recuperar, insertar o modificar la información almacenada en la base de datos MongoDB alojada en AWS. Para ello, gracias a la abstracción que proporciona Mongoose, haciendo uso de métodos y funciones que simplifican la creación y ejecución de consultas, se realizarán las operaciones pertinentes

-**5. Generación de Respuesta en Formato JSON**: Una vez que la API tenga la información necesaria, generará una respuesta estructurada en formato JSON. JSON es un formato de intercambio de datos liviano y altamente legible,

lo que facilitará la interpretación y el procesamiento por parte de la aplicación *frontend*.

-6. Retorno de la Respuesta al Cliente: La respuesta JSON se enviará de vuelta a la aplicación *frontend* que realizó la solicitud inicial. Esta respuesta contiene la información solicitada o el resultado de la acción.

7- Interacción en la Interfaz de Usuario: La aplicación *frontend* recibirá la respuesta JSON y la procesará según sea necesario. Puede utilizar los datos recibidos para actualizar la interfaz de usuario o presentar información de los cambios que ha habido en la base de datos.

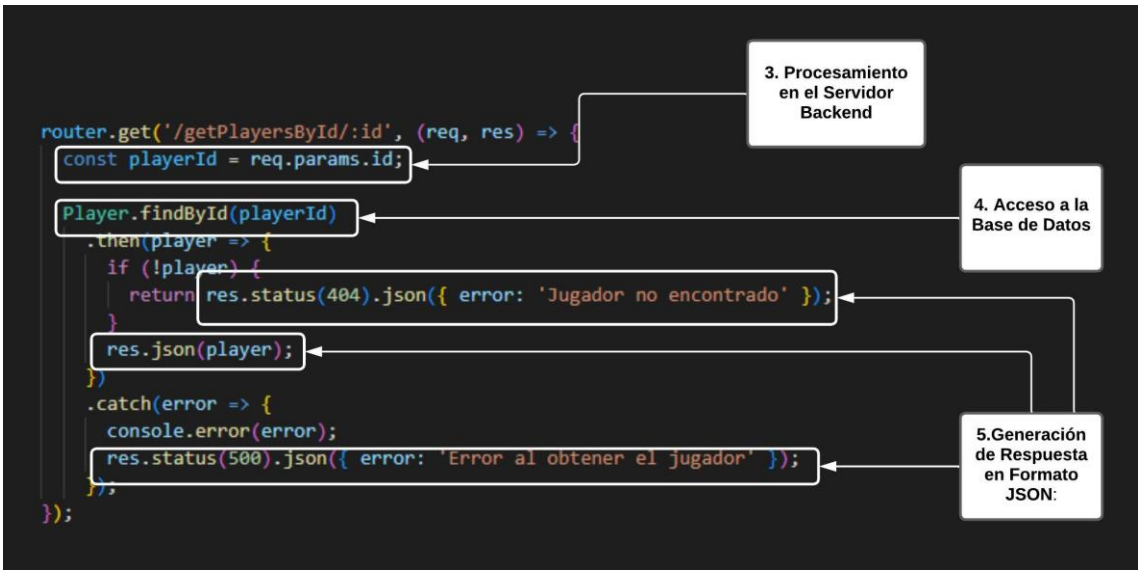


Figura 8 Método de la API

En la figura 8 se muestra un método de la API de jugadores en el cual la API recibe una solicitud de búsqueda de un jugador. Para ello recibe como parámetro la id única del jugador. En primer lugar, en el paso 3, el método se encarga de guardar en una constante el valor que recibe de la llamada HTTP. Posteriormente, en el paso 4, establece una comunicación con la base de datos pidiendo el objeto que contenga la id única del jugador. Por último, en el paso 5, dependiendo de la contestación de la base de datos, el método genera un archivo JSON para responder a la consulta HTTP.

4.2.2. Diseño de la base de datos

En la concepción de la base de datos de este proyecto, basada en MongoDB, se ha buscado un enfoque que combine la flexibilidad inherente a una base de datos NoSQL con la necesidad de establecer relaciones entre distintos esquemas. Si bien MongoDB promueve la de normalización y la independencia de esquemas, se ha considerado necesario introducir relaciones para atender requisitos específicos y optimizar el acceso y la organización de los datos.

Modelado Documental:

MongoDB es una base de datos NoSQL que adopta un enfoque documental, almacenando la información en documentos BSON (Binary JSON). Esta estructura adaptable permite almacenar datos heterogéneos en un mismo conjunto de datos, reduciendo la complejidad de las operaciones de consulta y manipulación.

Relaciones en el Contexto NoSQL:

A pesar de esta naturaleza de almacenamiento, el diseño de la base de datos ha incorporado relaciones en situaciones particulares. Aunque las relaciones en MongoDB difieren de las relaciones en bases de datos relacionales, se han implementado usando métodos como la referenciación y la inclusión de subdocumentos.

Tomemos el escenario en el que se requieren almacenar datos de Usuarios y sus correspondientes WatchList, estas son propias de cada usuario y se utilizan para guardar jugadores en listas privadas para su posterior seguimiento y análisis. En lugar de incluir las WatchList, dentro de los documentos de usuario, se ha optado por la referenciación. Cada documento de usuario almacena información propia del usuario, mientras que las WatchList, se guardan en documentos independientes, haciendo referencia al usuario al que están asociados mediante un identificador único.

Ventajas del Enfoque Relacional:

- **Eficiencia en Consultas**: Mediante referencias o subdocumentos, se permite la ejecución de consultas más eficientes, evitando la necesidad de recorrer todos los documentos.
- **Optimización de Espacio**: Al evitar la redundancia de datos, se logra una utilización eficiente del espacio en la base de datos, especialmente en relaciones uno a muchos.
- **Gestión de Actualizaciones**: Modificaciones en datos compartidos se realizan en un único punto y se propagan automáticamente a todas las referencias, simplificando la gestión de la integridad.

Diagrama de la base de datos.

En la figura 9, se muestra el diagrama de la base datos MongoDb. Como vemos, la tabla más importante y donde recae la mayor cantidad de información es la tabla que hace referencia a los jugadores. Estos serán la piedra angular del proyecto y la información de éstos se irá actualizando cada semana conforme se jueguen nuevos partidos. Los jugadores podrán estar dentro de las listas personalizadas de los usuarios WatchList o dentro de las alarmas que recibirían los usuarios cada semana. Por esto mismo existe una relación entre la tabla *player* y la tabla alarmas, ya que los jugadores están incluidos en las alarmas lo mismo pasa en el caso de las WatchList.

Por otro lado, los usuarios podrán estar suscritos a alarmas, por lo que existe una relación entre la tabla usuarios y la tabla alarmas. Por otra parte, los usuarios también pueden crear sus propias WatchList y añadir jugadores, por lo que existe una relación entre la tabla usuario y la tabla WatchList.

Por último, la tabla log es de uso exclusivo del panel de administración. Este se generará cada vez que un usuario añada un jugador a su WatchList. De manera que la tabla log tiene una relación con el jugador que se está añadiendo además de otra relación con la WatchList a la que se añade y una última relación con el usuario que está añadiendo a ese jugador.

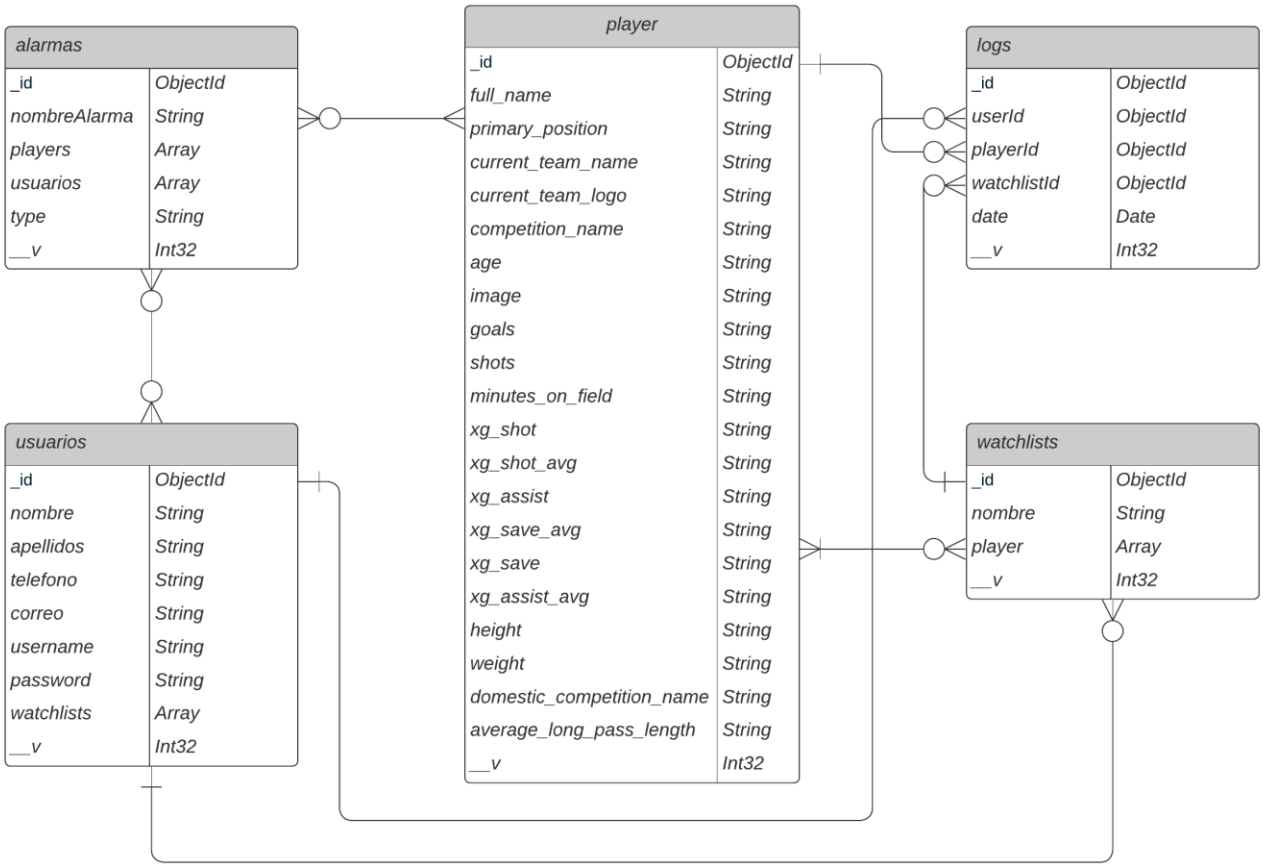
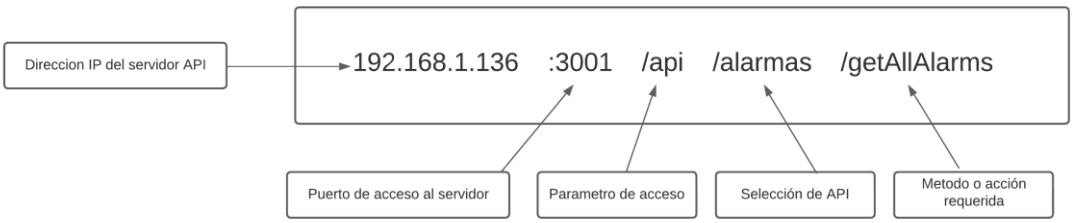


Figura 9 Diagrama de la base de datos

4.2.3. Diseño de la API

Dadas las necesidades del proyecto, se ha determinado que harán falta grupos de métodos dentro de la API de manera que se configurara una estructura de enrutamiento que se asemeje a unas sub-API o enrutadores específicos. Esta técnica se utiliza comúnmente para modularizar y organizar el código en aplicaciones Node.js y Express, especialmente cuando la API tiene múltiples recursos y rutas.

De esta manera, se creará una clase principal index que actuará como un punto de entrada para configurar las rutas principales de la API.



Para ello en el parámetro de selección de API necesitaremos 5 posibles enrutadores, uno por cada tabla de la base de datos.

-1 (/alarmas): Contendrá todos los métodos relacionados con la creación modificaciones o consulta de alarmas.

-2 (/usuarios): Contendrá todos los métodos relacionados con la creación modificaciones o consulta de usuarios.

-3 (/logs): Contendrá todos los métodos relacionados con la creación modificaciones o consulta de logs.

-4 (/players): Contendrá todos los métodos relacionados con la creación modificaciones o consulta de jugadores.

-5 (/watchlists): Contendrá todos los métodos relacionados con la creación modificaciones o consulta de Watchlists.

Una vez seleccionada la sub-API, el siguiente parámetro de la consulta HTTP será la llamada al método que queramos que actúe.

4.3 Tecnología utilizada

A continuación, se enumerarán y explicarán las diferentes tecnologías y herramientas que se han sido necesarias para el desarrollo del proyecto.

4.3.1. Lenguajes de programación, frameworks y librerías.

React JS para el Panel de Administración:

React JS, un *framework* de código abierto desarrollado por Facebook, ha sido el elemento principal en la creación del panel de administración web. React JS facilita la construcción de interfaces de usuario interactivas y de alto rendimiento mediante el uso de componentes reutilizables. Los componentes, que representan piezas individuales de la interfaz de usuario, pueden ser diseñados y modificados de manera independiente, lo que simplifica la gestión y el mantenimiento del código

React JS se caracteriza por su enfoque en la programación declarativa y la representación virtual del DOM. Esta abstracción permite que los cambios en los datos se reflejen de manera eficiente en la interfaz, optimizando la velocidad y la adaptabilidad de la aplicación.

React Native para la Aplicación Móvil:

Para el desarrollo de la aplicación móvil, se ha optado por React Native [13-14], una extensión de React JS que permite crear aplicaciones móviles nativas utilizando un único código base. React Native ofrece la ventaja de desarrollar aplicaciones móviles multiplataforma, lo que significa que gran parte del código puede ser compartido entre las versiones de iOS y Android.

Al aprovechar React Native, se construyen componentes que se traducen directamente en elementos nativos de la interfaz de usuario, proporcionando una experiencia de usuario auténtica y de alto rendimiento. La programación declarativa y la actualización eficiente del DOM virtual, características centrales de React, también están presentes en React Native, lo que permite cambios rápidos y actualizaciones sin problemas.

JavaScript (JS) en React JS y la API:

JavaScript es el lenguaje base que se ha empleado tanto en la construcción de la interfaz de usuario en React JS [15] como en la implementación de la API.

JavaScript es el lenguaje de programación central en el ecosistema de React JS. Los componentes de React JS, que representan partes individuales de la interfaz de usuario, se definen y construyen utilizando JavaScript. Las funcionalidades de manipulación de datos, eventos y dinamismo de la interfaz se implementan a través de JavaScript.

React JS utiliza JavaScript para interactuar con el DOM virtual, gestionar el estado y los ciclos de vida de los componentes, y manejar eventos de usuario. El uso de JavaScript proporciona flexibilidad y poder en la creación de componentes dinámicos y reactivos.

En el contexto de la API, JavaScript se usa para definir rutas, controladores y lógica de servidor en general. Esto permite que el servidor interprete y responda a las solicitudes HTTP provenientes de las aplicaciones frontend.

JSX en React Native:

JSX (JavaScript XML) [25] es una extensión de sintaxis que combina elementos de JavaScript y XML (Extensible Markup Language) para describir la estructura y el diseño de la interfaz de usuario. En React Native, JSX se convierte en elementos nativos en la plataforma móvil. Aunque JSX se parece a HTML, en realidad es una representación de JavaScript que permite definir la jerarquía de componentes de manera más legible e intuitiva.

JSX simplifica la creación de elementos de interfaz de usuario en React Native. Por ejemplo, en lugar de crear elementos de manera imperativa mediante JavaScript puro, puedes utilizar JSX para describir componentes de una manera similar a la escritura de elementos HTML. Luego, React Native se encarga de convertir esos elementos JSX en elementos nativos en tiempo de ejecución.

Uso de Node.js y Express para Crear el Servidor API

Para la creación del servidor API, se ha empleado Node.js junto con el framework Express

Node.js:

Node.js [15] es un entorno de tiempo de ejecución de JavaScript basado en el motor V8 de Chrome. A diferencia de la ejecución tradicional de JavaScript en el navegador, Node.js permite ejecutar código JavaScript en el lado del servidor. Esta capacidad brinda numerosos beneficios, como la posibilidad de construir aplicaciones de red y servidores altamente escalables y eficientes. La elección de Node.js para crear el servidor API se debe a su enfoque en el manejo asíncrono y no bloqueante, lo que resulta esencial para atender múltiples solicitudes en paralelo sin degradar el rendimiento. Node.js también se destaca por su ecosistema de paquetes npm, que ofrece una amplia variedad de módulos y bibliotecas que agilizan el desarrollo y la implementación.

Express:

Express [19-20] es un *framework* minimalista y flexible para Node.js que simplifica la creación de servidores web y APIs. Ofrece una abstracción sobre las operaciones HTTP y proporciona una serie de utilidades para definir rutas, manejar solicitudes y enviar respuestas. La simplicidad de Express permite centrarse en la lógica de la aplicación en lugar de preocuparse por detalles de bajo nivel. La elección de Express como *framework* para el servidor API se basa en su facilidad de uso y su capacidad para acelerar el proceso de desarrollo. Proporciona métodos y middleware que facilitan la validación de datos, la autenticación, el manejo de sesiones y otras tareas comunes en la creación de API.

Uso de Mongoose para la Conexión a MongoDB

Mongoose es una biblioteca de modelado de objetos para Node.js que se ha empleado para facilitar la conexión y la interacción con la base de datos MongoDB. Esta herramienta proporciona una capa de abstracción sobre las operaciones en la base de datos, permitiendo definir esquemas, crear modelos y ejecutar consultas de manera más sencilla y estructurada.

4.3.2. Herramientas

Uso de Visual Studio Code (VSCode)

Visual Studio Code [26], ampliamente conocido como VSCode, es un entorno de desarrollo integrado (IDE) altamente popular y versátil desarrollado por Microsoft. Su adopción en el ámbito de la programación se debe a su interfaz de usuario amigable que lo hacen atractivo para desarrolladores de diferentes niveles de experiencia y la posibilidad de emplearlo para diversos lenguajes de programación.

La integración de extensiones permite adaptarse a las necesidades específicas del proyecto, agregando características como la validación de sintaxis, el resaltado de código y la depuración en tiempo real.

Uso de Canva para Mockups

Canva ha sido esencial en la fase de diseño al permitir la creación eficiente de mockups visuales sin necesidad de experiencia en diseño. Su interfaz sencilla facilita la elaboración de representaciones visuales atractivas de las interfaces de usuario.

Uso de Lucidchart para Diagramas

Se ha utilizado Lucidchart para la visualización clara de conceptos complejos durante el diseño y desarrollo de la aplicación. Su ha empleado en la creación de diversos tipos de diagramas, como diagramas de la arquitectura o en diagramas de clase.

5. Desarrollo de la solución

5.1. Organización

Durante el desarrollo del proyecto de la aplicación se decidió utilizar la metodología ágil para una gestión eficiente del proyecto, permitiendo una mayor flexibilidad y adaptabilidad ante los posibles desafíos que pudieran surgir durante el desarrollo. Cada sprint de tres semanas se enfocó en objetivos específicos y se dividió en tareas concretas para lograr avances significativos en el desarrollo de la aplicación, la API y el panel de administración.

Sprint 1 (1 de marzo - 21 de marzo):

Durante este sprint inicial, se llevó a cabo una fase de planificación detallada del proyecto. Se realizaron reuniones con la empresa interesada en el desarrollo de la aplicación como guía para definir los objetivos y requisitos del proyecto. También se identificaron las funcionalidades clave que se implementarían en la aplicación de *scouting*, la API y el panel de administración. Además, se llevó a cabo una investigación exhaustiva para seleccionar las tecnologías adecuadas para el desarrollo, optando por React Native para la aplicación móvil, Node.js y Express para el servidor, y MongoDB para la base de datos. Se configuró el entorno de desarrollo y se estableció una hoja de ruta para los siguientes *sprint*.

Sprint 2 (22 de marzo - 11 de abril):

En este segundo *sprint*, se dio inicio al desarrollo incremental de la aplicación, la API y el panel de administración. Se trabajó en la implementación del esquema de la base de datos en MongoDB, así como en la funcionalidad básica para el registro y autenticación de usuarios en la aplicación. Se crearon las rutas iniciales para la API y se desarrollaron las funciones para la autenticación y autorización de usuarios en el servidor. Paralelamente, se avanzó con la mejora de la interfaz de usuario básica, permitiendo la visualización de jugadores en la aplicación.

Sprint 3 (12 de abril - 2 de mayo):

Durante este tercer *sprint*, se enfocó en implementar funcionalidades clave para permitir a los usuarios buscar y filtrar jugadores según criterios específicos. También se realizaron mejoras en la interfaz de usuario, mejorando la experiencia del usuario. Además, se llevaron a cabo pruebas iniciales para identificar y corregir posibles errores en la aplicación.

Sprint 4 (3 de mayo - 23 de mayo):

En este cuarto *sprint*, se trabajó en el desarrollo de la API para gestionar cualquier solicitud tanto de la aplicación como del futuro panel de administración. Se implementó la funcionalidad para administrar jugadores de manera que pudieras añadirlos a las WatchList y eliminarlos en cualquier parte de la aplicación que se

renderizaran las cartas de los jugadores. Así como la creación y eliminación de WatchList.

Sprint 5 (24 de mayo - 13 de junio):

Durante este quinto *sprint*, se implementó toda la funcionalidad relacionada con las alarmas, es decir la visualización de listas de alarmas dentro de la aplicación, la posibilidad de entrar en los dos tipos posibles de alarmas. Alarmas tipo listas de jugadores y alarmas tipo terreno de juego, y que pudieras interactuar con los jugadores que ahí aparecían agregándolos a las respectivas WatchList de interés. Se implementó la posibilidad de suscribirte a las alarmas teniendo en cuenta si se tienen suficientes créditos para ello además de la posibilidad de activar las notificaciones push de estas.

Sprint 6 (14 de junio – 5 de julio):

Durante este quinto *sprint*, se implementó la base del panel de administración, configurando toda la navegación y conexión de éste con la API, así como la posibilidad de crear nuevas alarmas de manera manual o mediante un sistema que fuera capaz de leer archivos JSON y cargar automáticamente la información de la alarma.

Sprint 7 (6 de julio - 26 de julio):

Durante este séptimo *sprint*, se centró en la implementación de la funcionalidad de monitoreo de alarmas y acceso a estadísticas relevantes en el panel de administración. Además de implementar la funcionalidad para administrar jugadores y usuarios a través del panel de administración, permitiendo a los usuarios con los permisos adecuados realizar tareas de gestión de contenidos, se realizaron ajustes finales y mejoras de rendimiento en la aplicación, la API y el panel de administración. Se llevaron a cabo pruebas exhaustivas para detectar y corregir posibles errores, asegurando la calidad del producto final.

Sprint 8 (1 de septiembre – 7 de septiembre):

La última semana del proyecto, se preparó la entrega final, incluyendo la documentación necesaria y la presentación del trabajo realizado. Se realizaron todos los test unitarios de nuevo por si se pudieran haber hecho cambios que hubieran afectado a las primeras funcionalidades y se corrigieron los errores detectados.

5.2. Problemas e implementaciones relevantes

Como se ha mencionado en diversas ocasiones, al comienzo del desarrollo, tanto React Native como React.js o la implementación de API REST eran tecnologías y metodologías completamente nuevas y desconocidas para mí, por lo que los problemas y las dificultades técnicas que se han sufrido son incontables. Sin embargo, a continuación, se expondrán únicamente las que más relevancia han tenido en este proyecto:

5.2.1 Búsqueda dinámica de jugadores

Un aspecto que se definió como fundamental en alguna de las reuniones con la empresa interesada en el desarrollo de la aplicación era que la búsqueda de jugadores fuera dinámica. Es decir, que cada vez que se cambiara un parámetro de alguno de los filtros la lista generada fruto del resultado debía cambiar también automáticamente sin necesidad de pulsar un botón buscar. Esto añade bastante complejidad al código y sobre todo bastante carga a al servidor, ya que tiene que estar haciendo constantes peticiones cada vez que se cambia un valor. Además de que la lista no puede quedarse vacía en ningún momento si algún parámetro se queda sin seleccionar.

En primer lugar, la búsqueda cuenta con cuatro selectores principales, un selector de posición, un selector de edad, un selector de liga y un selector de equipo.

Aquí nos encontramos otro problema en la implementación y es que al tener la aplicación todos los equipos de las principales ligas mundiales, no podemos hacer un selector de equipo con todos los equipos de la aplicación. El selector de equipo debe aparecer únicamente cuando se seleccione primero una liga y contener los equipos pertenecientes a dicha liga.

Para ello contamos con el atributo onValueChange utilizado en algunos componentes de React Native para controlar los cambios de valor y capturar los eventos cuando el usuario interactúa con estos componentes.

Cuando hay algún cambio de valor en el selector de liga el método handleLeagueChange entra en acción. En ese instante, ocurren tres cosas principales.

La primera de todas actualiza el valor de la liga seleccionada para poder hacer la consulta, después inicializa el valor del selector de equipo para que este aparezca ya con la liga seleccionada. De esta forma, sólo se mostrarán los equipos de la liga seleccionada y por último para conseguir que la lista de jugadores se vuelva a cargar y se haga una nueva petición a Mongo fuerza al componente a volver a renderizarse con el fin de conseguir que la búsqueda sea dinámica.

```
const handleLeagueChange = (value) => {

  setSelectedLeague(value);
  setSelectedTeam('');
  setForceRenderKey((prevKey) => prevKey + 1);

};
```

Figura 10 Código de un manejador de evento

Dado que es posible por ejemplo que queramos buscar jugadores de toda una liga sin seleccionar un equipo en específico, había que buscar una manera de poder hacer la consulta a mongo con algún parámetro vacío.

Para ello, en la URL de consulta, si algún selector se quiere dejar vacío se definió el carácter "_" como distintivo de que ese parámetro se quedaba vacío.

```
router.get('/:primary_position/:current_team_name/:domestic_competition_name/:age', (req, res) => {
  const primary_position = req.params.primary_position;
  const current_team_name = req.params.current_team_name;
  const domestic_competition_name = req.params.domestic_competition_name;
  const age = req.params.age;

  let query = {};

  if (primary_position && primary_position !== '_') {
    query.primary_position = primary_position;
  }

  if (current_team_name && current_team_name !== '_') {
    query.current_team_name = current_team_name;
  }

  if (domestic_competition_name && domestic_competition_name !== '_') {
    query.domestic_competition_name = domestic_competition_name;
  }

  if (age && age !== '_') {
    query.age = age;
  }

  Player.find(query)
    .then(players => {
      res.json(players);
    })
    .catch(error => {
      console.error(error);
      res.status(500).json({ error: 'Error al obtener los jugadores' });
    });
});
```

Figura 11 Código de un método de la API

De esta manera, la API cuando recibe todos los parámetros en la URL, puede comprobar que si el parámetro está vacío o cuenta con el carácter "_" este no tiene que ser añadido a la consulta que se hará a MongoDB.

5.2.2 Búsqueda parcial de nombres de jugadores y usuarios.

En el panel de administración de la aplicación existe un apartado de *logs*. Estos *logs* se generan cuando el usuario añade a algún jugador a alguna WatchList. Era un requisito fundamental que se pudieran filtrar los *logs* por nombre de jugador y por no nombre de usuario o haciendo un filtrado mezclando ambos parámetros. El problema es que muchas veces en el uso diario de una aplicación, no sabemos el nombre del usuario o del jugador tal y como aparece en la base de datos, por lo que se deben de poder hacer búsquedas parciales en la que se busquen coincidencias entre el texto insertado y los

datos almacenados en la base de datos. Además de que la búsqueda pueda ser cruzada con usuario y jugador.

```
179  router.get('/getLogsByPlayerAndUserName/:fullname/:name', async (req, res) => {
180    try {
181      const full_name = req.params.fullname;
182      const nombre = req.params.name;
183      console.log(full_name)
184      const playerQuery = full_name ? { full_name: { $regex: full_name, $options: 'i' } } : {};
185      const userQuery = nombre ? { nombre: { $regex: nombre, $options: 'i' } } : {};
186
187      let playerIds = [];
188      if (Object.keys(playerQuery).length > 0) {
189        const players = await Player.find(playerQuery);
190        playerIds = players.map((player) => player._id);
191      }
192
193      let userIds = [];
194      if (Object.keys(userQuery).length > 0) {
195        const users = await Usuario.find(userQuery);
196        userIds = users.map((user) => user._id);
197      }
198
199      let query = {};
200
201    if (full_name !== "_" && nombre !== "_") {
202      query = {
203        $and: [
204          { jugadorId: { $in: playerIds } },
205          { userId: { $in: userIds } }
206        ]
207      };
208    } else if (full_name == "_") {
209      query = {
210        $or: [
211          { jugadorId: { $in: playerIds } },
212          { userId: { $in: userIds } }
213        ]
214      };
215    } else if (nombre == "_") {
216      query = {
217        $or: [
218          { jugadorId: { $in: playerIds } },
219          { userId: { $in: userIds } }
220        ]
221      };
222    }
223
224    const logs = await Log.find(query);
225
226      res.json(logs);
227    } catch (error) {
228      console.log('Error al obtener los logs:', error);
229      res.status(500).json({ error: 'Ocurrió un error al obtener los logs' });
230    }
231  });
232
```

Figura 12 Código de un método de la API

La figura 12 representa el código del método getLogsByPlayerAndUserName de la sub-API de *logs*. En este método se recibirán los textos introducidos en los campos nombre de usuario y jugador del apartado logs del panel de administración. Como es uno de los métodos más complejos de la API, creo que hay que prestar atención a algunos detalles.

Como mencionábamos anteriormente, la necesidad de poder hacer búsquedas cruzadas y parciales, las líneas 184 y 185 de código cobran bastante importancia. Vamos a explicar por ejemplo la línea 184:

```
const playerQuery = full_name ? { full_name: { $regex: full_name,
$options: 'i' } } : {};
```

La línea de código se encarga de construir una consulta para buscar jugadores en la base de datos MongoDB. La consulta se basa en el valor del parámetro full_name, que se pasa como parte de la URL de la solicitud en la ruta de la API.

- **full_name:** Este es el valor del parámetro fullname que se recibe en la solicitud. Representa el nombre completo del jugador que se está buscando en la base de datos.

- **playerQuery:** Esta variable es un objeto que se utilizará como parte de la consulta a la base de datos para buscar jugadores cuyo nombre completo coincida con el valor de full_name.

- **?:** Este es un operador ternario que evalúa si full_name tiene un valor. Si full_name tiene un valor (es decir, no es nulo ni una cadena vacía), el operador ternario ejecutará la primera parte de la expresión después del ?.

- **{ full_name: { $regex: full_name, $options: 'i' } }:** En caso de que full_name tenga un valor, se construye un objeto que define la consulta para buscar jugadores cuyo nombre completo coincida de manera parcial con full_name, y se utiliza la opción $regex para realizar una búsqueda basada en expresiones regulares. La opción $options: 'i' especifica que la búsqueda debe ser insensible a mayúsculas y minúsculas.

- **: {}:** La segunda parte del operador ternario después del : indica qué sucede si full_name no tiene un valor (es decir, es nulo o una cadena vacía). En este caso, simplemente se asigna un objeto vacío {} a playerQuery, lo que significa que no se aplicará ningún filtro en la consulta.

Una vez tenemos construidas las consultas, tanto para los jugadores como para los usuarios, el método se encarga de buscar todos los usuarios y jugadores que coinciden con los parámetros de búsqueda. Como puede haber muchos usuarios o jugadores que tengan coincidencias, utilizamos la función map para extraer el _id de cada usuario y crear un *array* con esos valores en las líneas 190 y 196 del código.

```
if (full_name !== "_" && nombre !== "_") {
    query = {
      $and: [
        { jugadorId: { $in: playerIds } },
        { userId: { $in: userIds } }
      ]
```

```
    };
  } else if (full_name == "_") {
    query = {
      $or: [
        { jugadorId: { $in: playerIds } },
        { userId: { $in: userIds } }
      ]
    };
  } else if (nombre == "_") {
    query = {
      $or: [
        { jugadorId: { $in: playerIds } },
        { userId: { $in: userIds } }
      ]
    };
  }
```

Por último, el método pasa a construir la consulta de los *logs*. Para ello dependiendo si los campos de la consulta están vacíos o no construye una consulta cruzada o no. Para ello utilizamos los operadores **"$and"**, que indica que ambas condiciones deben cumplirse para que un registro sea seleccionado o los operadores **"$or"**, que indican que basta con que una de las condiciones deba cumplirse para que un registro sea seleccionado. Por otro lado, utilizamos el operador **"$in"** para verificar si el parámetro del *log* cuando se haga la consulta está incluido en el *array* que previamente se ha verificado que coincide total o parcialmente con el parámetro de entrada.

5.3 Ejemplo de funcionamiento

Para mostrar el funcionamiento de la aplicación de ScoutIt se van a presentar una serie de capturas de la interfaz de usuario en la que se irán explicando las funcionalidades de cada pantalla. Como hemos indicado en distintas partes de esta memoria, ScoutIt tiene dos partes principales. La primera de todas sería la aplicación móvil a la cual acceden los usuarios de la aplicación y la otra parte sería la aplicación web a la cual acceden los administradores cuando quieren realizar alguna gestión o seguimiento de la aplicación.

Aplicación móvil nativa para dispositivos Android e IOS

Home

La pantalla que se muestra a la derecha es la pantalla principal de la aplicación, pantalla a la que se accede una vez el usuario inicia sesión.

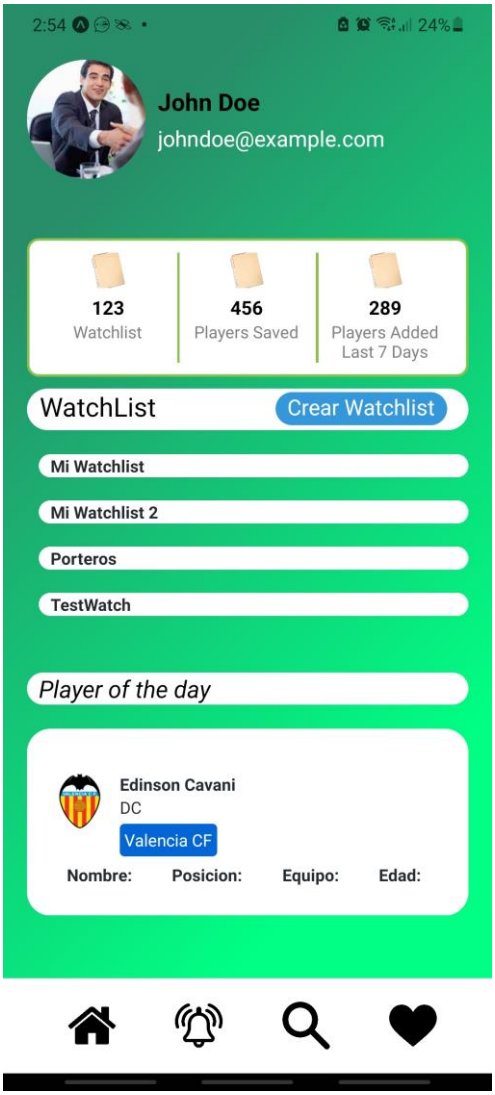


Figura 13 Pantalla principal de la aplicación movil

Como se puede observar está divida en cuatro bloques principales de información.

Bloque 1: En este bloque aparece la información del usuario que ha iniciado sesión en la aplicación. Aparece la foto de perfil que se ha subido en el momento del registro junto con el nombre de usuario y su correo electrónico.

Bloque 2: En este bloque aparecen estadísticas del uso que ha tenido el usuario en la aplicación, estas características están subdivididas en tres sub-bloques. El primero de todos muestra el número de WatchList que tiene el usuario creadas, el segundo indica el número de jugadores que hay guardados en esas WatchList. El último de todos muestra del total de número de jugadores añadidos a las WatchList, cuántos de ellos han sido añadidos en la última semana.

Bloque 3: En este bloque aparecen las últimas cuatro WatchList creadas por el usuario con sus respectivos nombres, ordenadas por antigüedad, siendo la más nueva la que más arriba aparece en la lista. Por otro lado, en este bloque también tenemos un botón que nos lleva a una ventada modal en la que podremos crear nuevas WatchLists.

Bloque 4: En este último bloque tenemos una recomendación de la aplicación de un jugador destacado del día de hoy, ya sea por su rendimiento deportivo o porque haya sido tendencia en algún medio deportivo.

Crear WatchList

Si pulsamos en el botón de Crear WatchList que teníamos en el bloque 3 de la pantalla principal de la aplicación se nos abre una ventana modal en la que podemos escribir el nombre que deseamos que tenga la WatchList, para posteriormente darle a botón de crear y que se nos añada nuestra lista para que podamos comenzar a añadir jugadores a la misma.

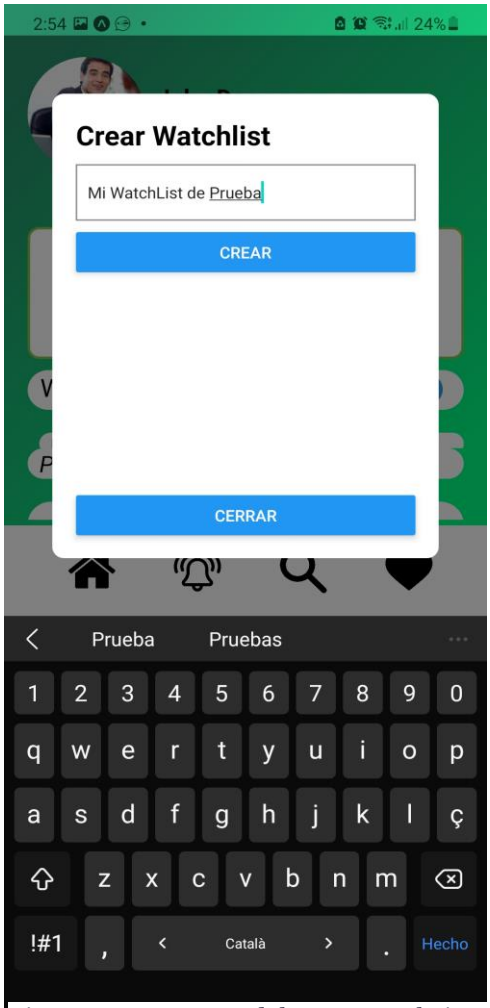


Figura 14 Ventana modal Crear WatchList

Navegación

Si nos fijamos en la navegación que tenemos en la parte de debajo de la aplicación, tenemos cuatro botones principales.

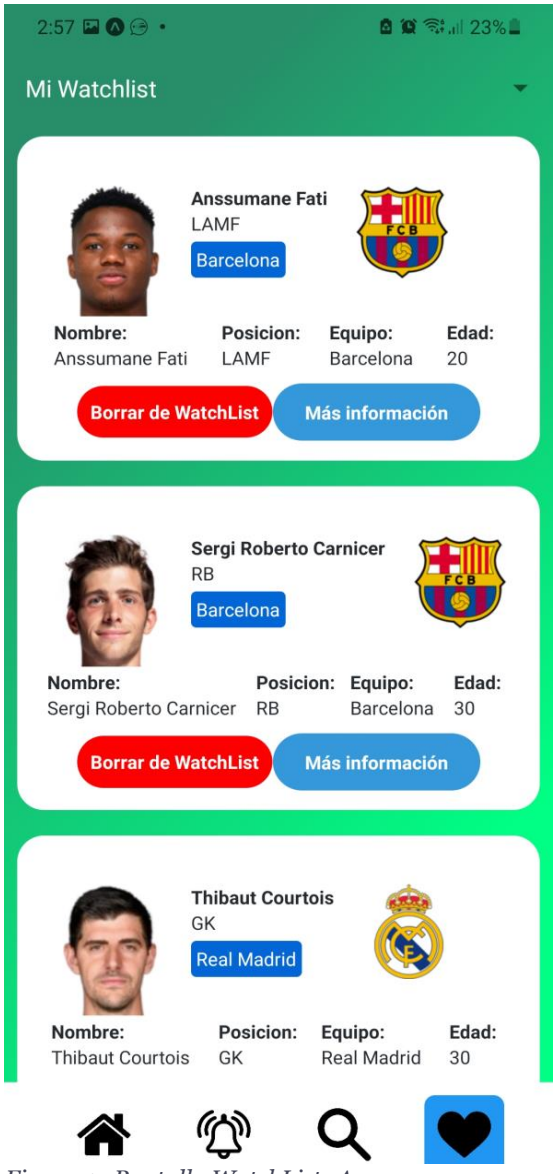


Figura 15 Pantalla WatchLists App

Vamos a explicar los botones de izquierda a derecha. El primero de todos, con un icono semejante a una casa, nos dirige a la pantalla principal de la aplicación o Home. Esta pantalla corresponde a la figura 13. El segundo botón en orden, con un icono semejante a una campana, corresponde a la pantalla de alarmas correspondiente a la figura 16. El tercer botón, con un icono que simula una lupa corresponde a la pantalla buscar de la aplicación, figura 23. Y por último tenemos un botón con un icono semejante a un corazón, el cual corresponde a la pantalla en la que podemos observar las WatchList del usuario que ha iniciado sesión. Podemos observar esta pantalla en las figuras 25 y 26. También, si nos fijamos en los botones, estos se resaltan en azul cuando se está en la pantalla que corresponde al botón para así facilitar al usuario la identificación de la pantalla en la que se encuentra.

Listado de Alarmas

La pantalla que tenemos a la derecha, figura 16, representa la pantalla correspondiente al listado de alarmas de la aplicación. En ésta el usuario puede ver las alarmas a las que está suscrito, así como entrar en ellas. La pantalla muestra un listado de las alarmas indicando el nombre de la misma, además del tipo de alarma. Las alarmas pueden ser tipo lista o tipo campo.

Si pulsamos por ejemplo en la primera alarma nos lleva a una alarma tipo lista que podemos ver en la pantalla de abajo, figura 17.

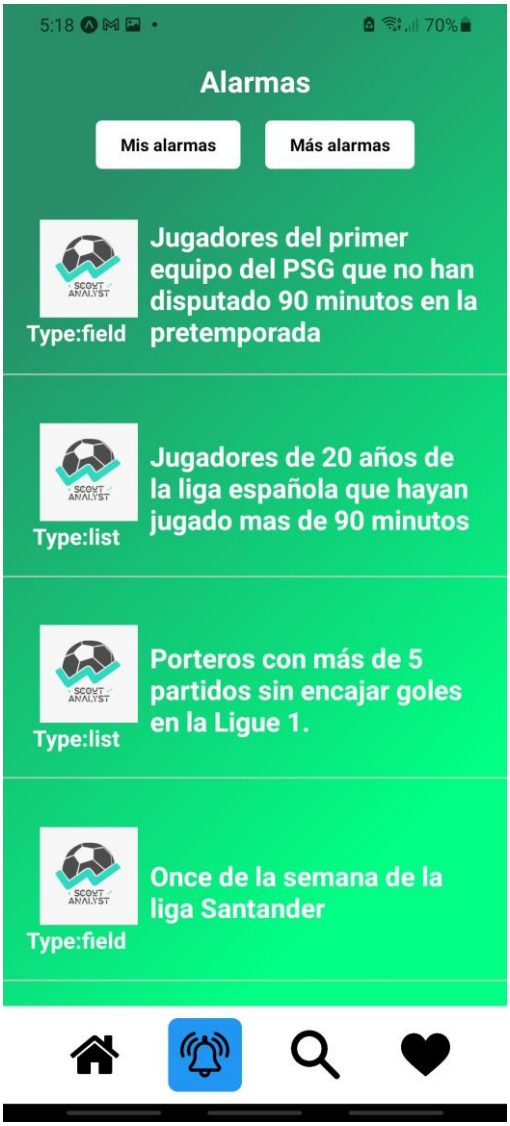


Figura 16 Pantalla Alarmas App

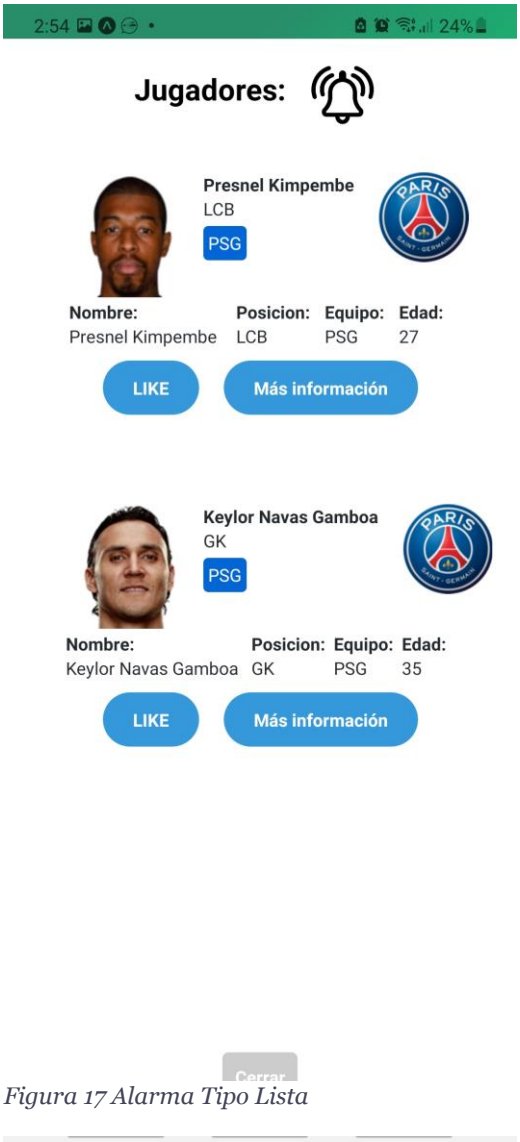


Figura 17 Alarma Tipo Lista

Primer Tipo, Alarmas Lista

En este tipo de alarmas los jugadores se representan mediante una lista. Como vemos en la figura 17, tenemos un botón en la parte superior representado por una campana, este botón activa las notificaciones *push* para esta alarma en concreto. De esta manera, si el usuario desea recibir una notificación cuando el contenido de ésta cambie, tiene la posibilidad de hacerlo.

Carta de Jugador

Los jugadores se representan mediante su carta. Este componente tiene bastante relevancia en la aplicación ya que gracias al funcionamiento de React, éste se renderiza en distintas partes de la aplicación favoreciendo la reutilización de código y evitando

redundancia. Esta carta de jugador nos presenta información relevante del mismo, como es su nombre, posición habitual en el terreno de juego, equipo y edad.

En estas cartas de jugadores tenemos varios botones, el primero de todos es el botón de LIKE.

Si este botón es pulsado, nos abre una ventana modal la cual podemos observar en la imagen de la derecha correspondiente a la figura 18. Esta ventana modal contiene una lista de las WatchList del usuario. De esta manera el usuario, mediante un selector tipo *switch,* puede marcar o desmarcar a que WatchList de las que tiene quiere añadirlo o eliminarlo. Si el jugador no está añadido a la WatchList en cuestión, el selector pasa a ser de color azul. Si por el contrario el jugador ya se encontraba en una WatchList y se presiona el selector, éste cambia a color gris simbolizando que el jugador ha sido eliminado.

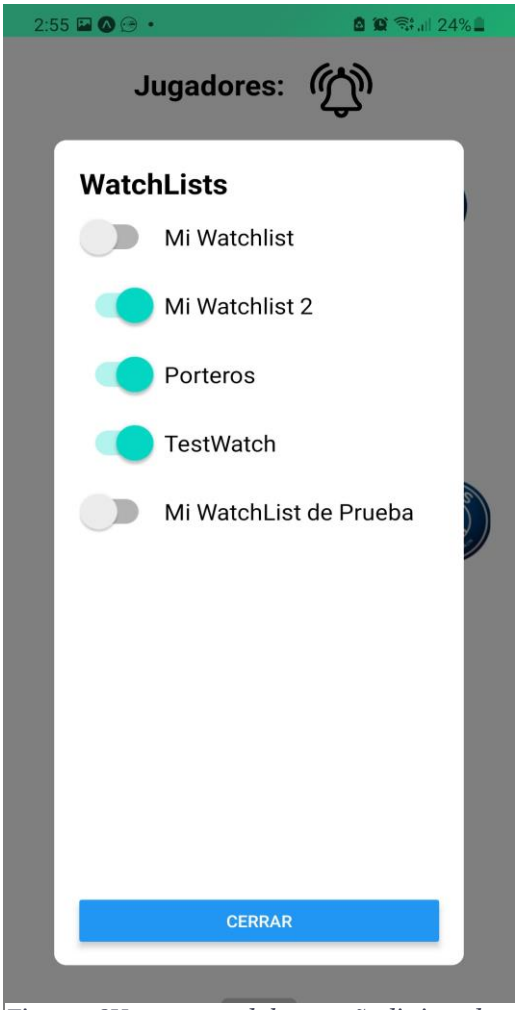


Figura 18Ventana modal para añadir jugadores a una WatchList

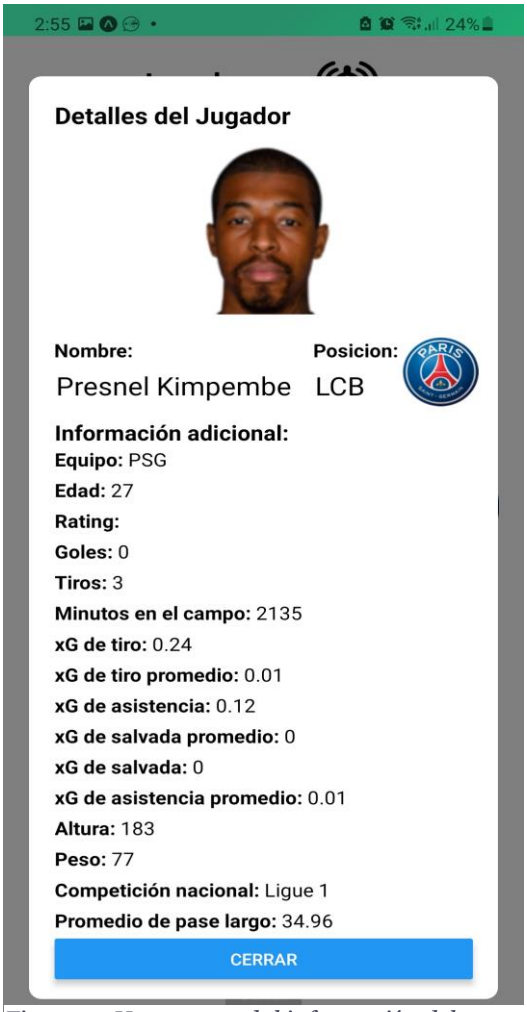


Figura 19 Ventana modal información del jugador

Es importante puntualizar que todas las listas de la aplicación cuentan con un *scroll* infinito, por lo que si hay demasiados elementos en la lista siempre se podrá seguir deslizando hacia abajo infinitamente. La carta de jugador tiene otro botón con el que podemos actuar. Si pulsamos el botón que dice Mas información se nos despliega otra ventana modal mostrada en la figura 19. Esta ventana modal nos presenta información más técnica y detallada del jugador. Esta información se actualiza cada jornada, por lo que los *scouter* pueden tener interés en revisar esta información cada semana, para ver la evolución de los jugadores que tienen guardados.

Segundo tipo, Alarmas Terreno de Juego.

Las alarmas a las que pueden estar suscritos los usuarios pueden ser de otro tipo, este tipo llamado terreno de juego se basa en disponer a los jugadores en un campo de fútbol en vez de una lista. Los jugadores se posicionan automáticamente en el terreno de juego según su posición habitual. Esto es especialmente útil en alarmas que representen equipos configurados a medida, o una métrica bastante común en el futbol llamada once ideal.

Como vemos en la figura 21, podemos ver representada una de estas alarmas y podemos interactuar con cada jugador de manera individual.

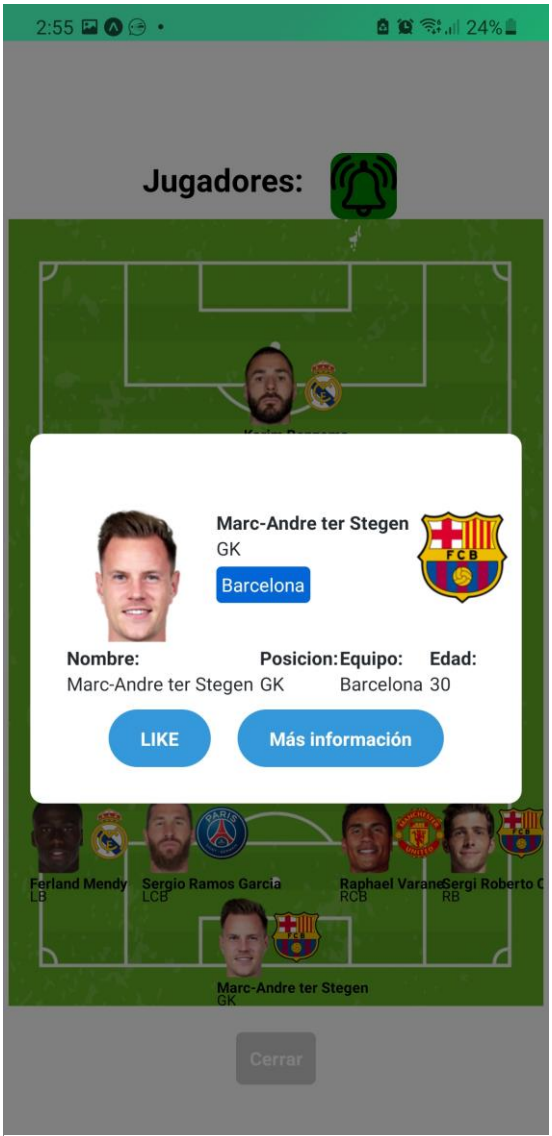


Figura 20 Carta de jugador en alarma tipo terreno de juego

Figura 21 Alarma Tipo Terreno de Juego

Si pulsamos en alguno de los jugadores que se muestran en el terreno de juego, se nos abrirá una ventana modal. Ésta nos muestra la carta de jugador del jugador que hemos presionado. Podemos observar esto en la pantalla de la izquierda, representada en la figura 20. Como vemos tenemos las mismas opciones que hemos explicado con anterioridad, botón LIKE para añadir al jugador a la WatchList que queramos. O el botón Más información para ver información más detallada del jugador.

Búsqueda de jugadores

En la pantalla de la derecha representada por la figura 23, podemos ver la pantalla correspondiente al tercer botón de navegación, la búsqueda de jugadores. En esta pantalla los usuarios pueden buscar a jugadores específicos que sean de su interés mediante cuatro filtros.

Estos filtros se basan en la posición habitual del jugador, liga en la que participa su equipo, equipo en el que juega y edad del jugador. Los filtros de posición, liga y equipo son un desplegable que los usuarios tienen que presionar para que se abra una ventana modal, en la cual seleccionar la opción que deseen.

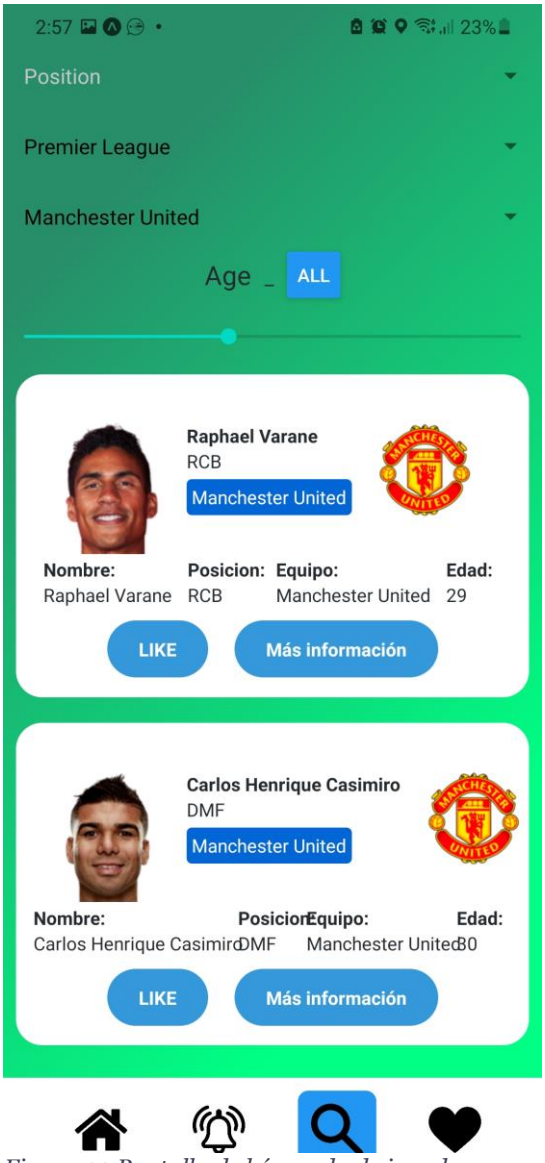


Figura 22 Pantalla de búsqueda de jugadores con filtros aplicados

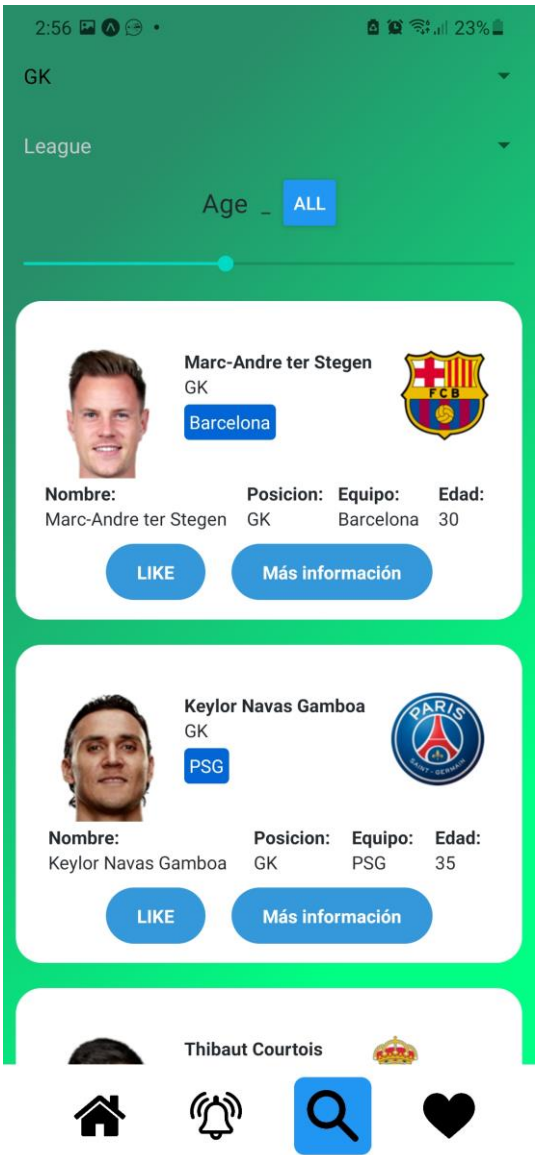


Figura 23 Pantalla de búsqueda de jugadores

Como se puede apreciar en la figura 23, el filtro de equipo no aparece. Esto es porque debido a la cantidad de equipos existentes, era imposible ponerlos en un solo filtro, por lo que al seleccionar una liga ya se muestra el filtro de equipos. En la figura 22 se puede observar el filtro. Por último, el filtro de edad está basado en una barra deslizable en la que los usuarios pueden mover la bola central hasta la edad que deseen o presionar el botón ALL, para hacer que el filtro muestre jugadores de todas las edades.

En la pantalla de la derecha correspondiente a la figura 24, se puede observar la forma que tienen los filtros una vez pulsamos en ellos, para seleccionar la opción que se desee. En este caso, la figura 24, corresponde al filtro de equipo, una vez ha sido seleccionado en el filtro de liga “Premier League”.

Como se ha mencionado, algunas veces en la memoria, la búsqueda es dinámica por lo que no existe un botón físico de búsqueda. Cada vez que se modifica un parámetro de alguno de los cuatro filtros, la lista se actualiza automáticamente.

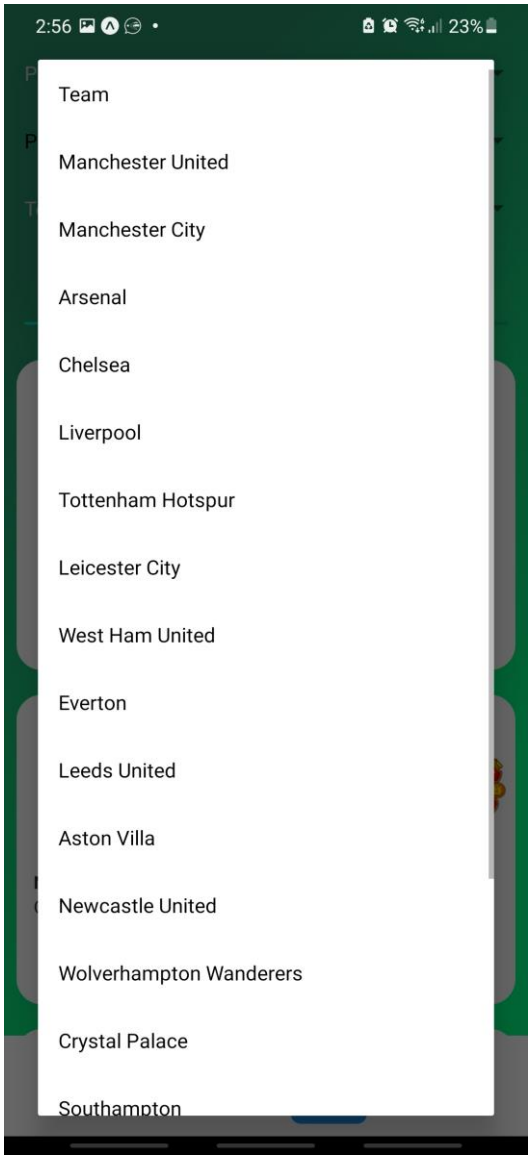


Figura 24 Pantalla de búsqueda de jugadores, filtro de equipos

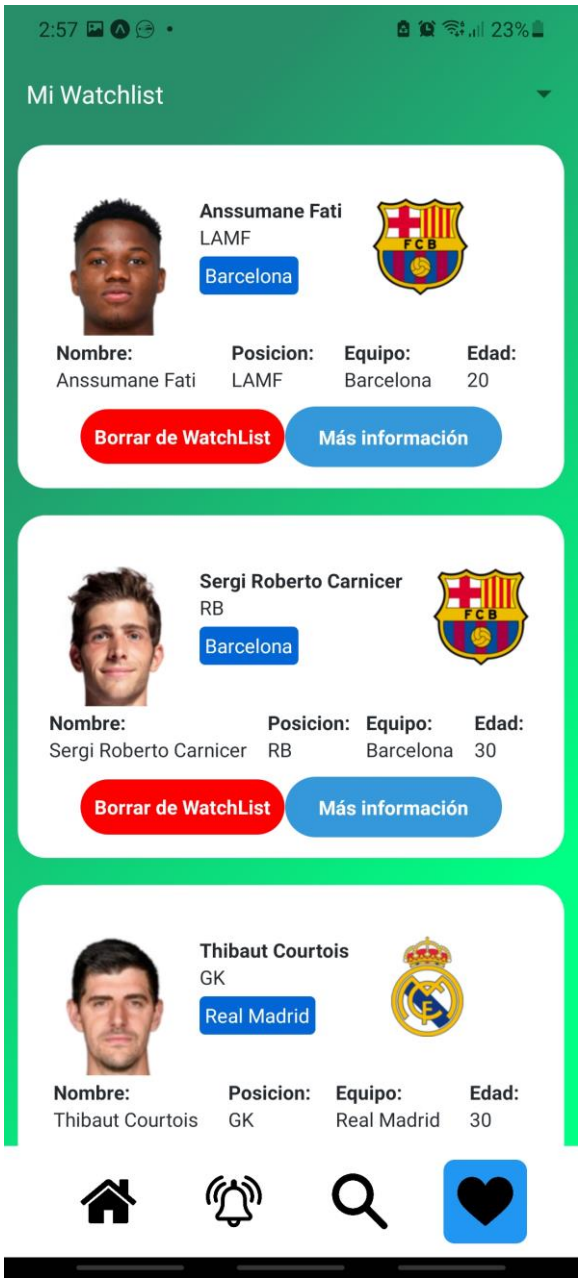


Figura 25 Pantalla de WatchLists App

WatchLists

En la pantalla de la izquierda, tenemos representada por la figura 25, la pantalla de visualización de las WatchList. Como se puede observar en esta pantalla, las cartas de jugadores han cambiado un poco, en esta pantalla el botón LIKE ha sido sustituido por el botón Borrar de WatchList. Este botón como su nombre indica borra el jugador de la WatchList directamente.

Por otro lado, en la parte de arriba de la pantalla tenemos el nombre de la WatchList tal y como se puede observar en la figura 26. Y si presionamos en el nombre se nos abre un desplegable, en el cual se puede seleccionar la WatchList que el usuario desee.

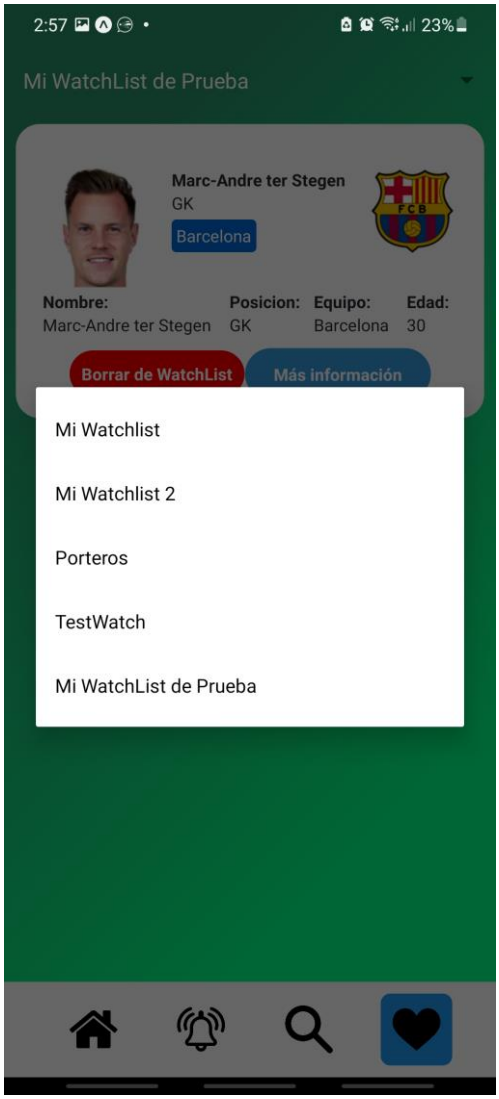


Figura 26 Ventana modal selector WatchList

Suscripción a Alarmas

En la pantalla de alarmas representada en la figura 27, tenemos la lista de alarmas disponibles a las que nos podemos suscribir. Podemos acceder a dicha lista pulsando el botón Mas alarmas que aparece en la parte superior de la pantalla. Una vez hemos accedido nos aparecen algunas alarmas preconfiguradas con sus nombres y tipo de alarma. Si queremos suscribirnos a ellas bastará con pulsar el botón de suscribirse para que se añadan. El botón cambiará a suscrito y la alarma en cuestión nos aparecerá en el apartado Mis alarmas.

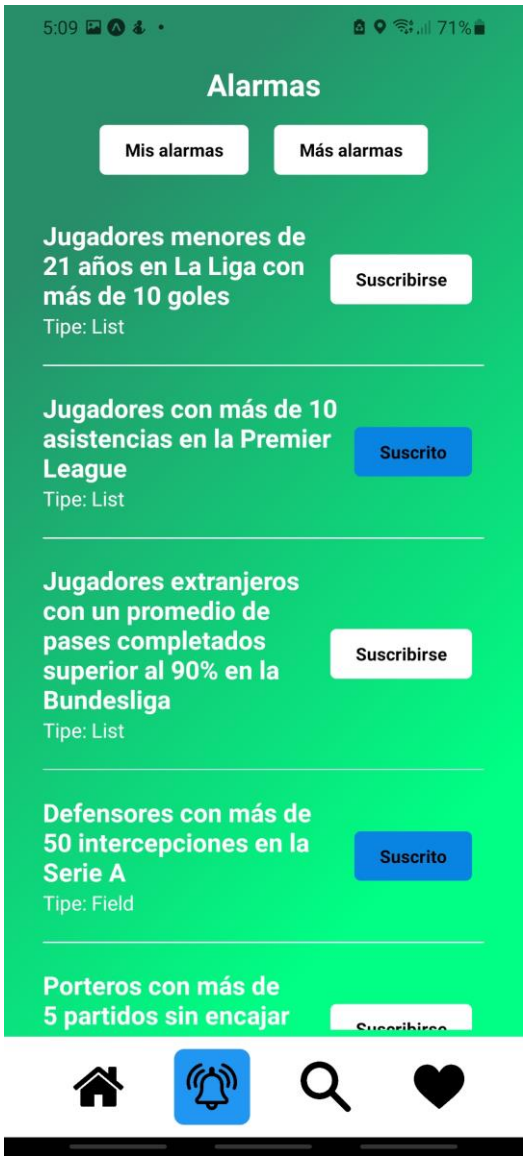


Figura 27 Pantalla de suscripción a alarmas

Panel de administración Web

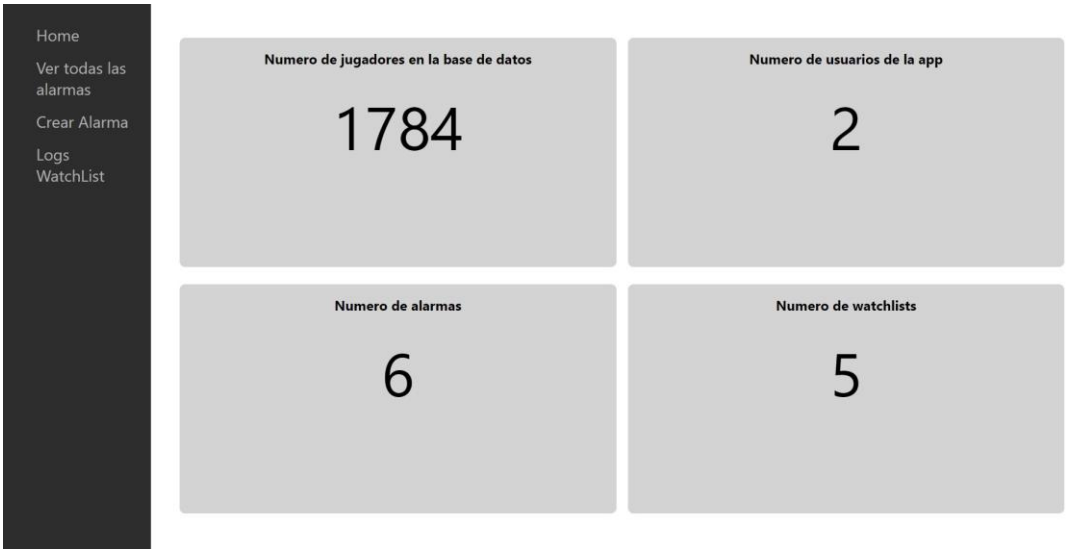


Figura 28 Pantalla Inicio Panel de Administración

La pantalla que se muestra arriba, representada por la figura 28, corresponde a la pantalla principal del panel de administración de la aplicación. En esta primera pantalla nos encontramos cuatro estadísticas de la aplicación para que se pueda seguir su evolución y crecimiento y sepamos en todo momento el tamaño actual de la aplicación.

Navegación

La navegación en el panel de administración está basada en cuatro botones desplegados en un menú fijo ubicado en la parte izquierda de la pantalla. Estos botones nos permitirán navegar por las distintas pantallas de la aplicación. Por ejemplo, el primer botón del menú corresponde a la pantalla home representada en la figura 28.

Listado de alarmas

El segundo botón de la navegación nos lleva a la pantalla representada por la figura 29. En esta pantalla, podemos observar todas las alarmas que hay ahora mismo activas en

Figura 29 Pantalla Alarmas Creadas Panel de Administración

la aplicación. Podemos ver el nombre de cada alarma, además de pulsar el botón de más información para ver información más detallada de cada alarma.

Figura 30 Ventana Modal Información de la Alarma

Si pulsamos el botón Más información, se nos abre una ventana modal en la que podemos ver que usuarios están suscritos a la alarma y que jugadores tiene incluidos dicha alarma.

Creación de nuevas alarmas.

Esta es una de las pantallas más importantes del panel de administración. Está representado por la figura 31. Esta pantalla está dividida en cuatro bloques principales, además de un campo nombre, en el cual el administrador introducirá el nombre de la alarma y de un botón Crear alarma.

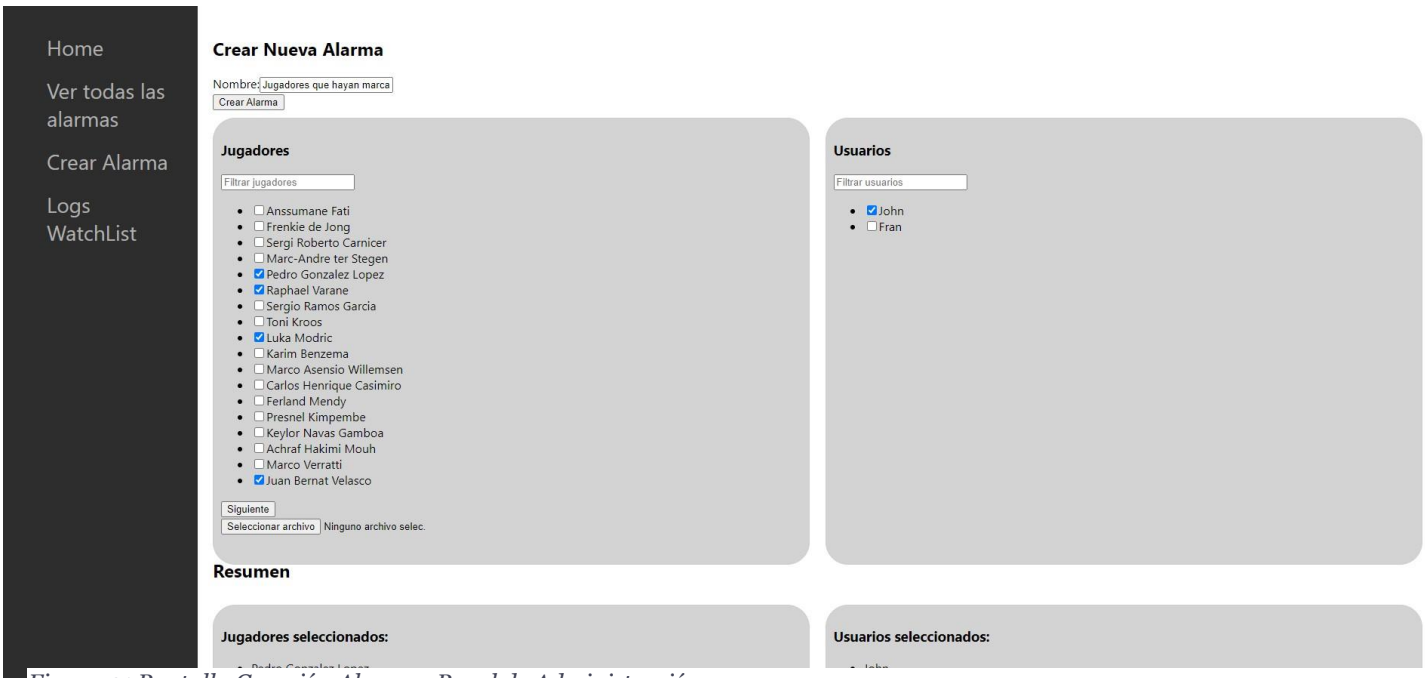


Figura 31 Pantalla Creación Alarmas Panel de Administración

El primer bloque, posicionado arriba a la izquierda, es el bloque destinado a elegir a los jugadores. Tenemos varias maneras de elegir que jugador va a estar incluido en la alarma, una manera manual y otra automática. La manera manual consiste en marcar las casillas de los jugadores que deseemos, podemos ir buscando a los jugadores mediante los botones de navegación, siguiente y atrás. O podemos filtrar directamente el jugador haciendo una búsqueda por su nombre en el campo habilitado para ello. La manera automática sería pulsar el botón seleccionar archivo y una vez se pulse este botón, el navegador abrirá nuestro explorador de archivos para que adjuntemos un archivo con extensión CSV. Este archivo debe tener una forma específica ya que el sistema buscará los id de los jugadores en la base de datos para automáticamente marcarlos como si se hubiera hecho a mano.

El segundo bloque, dispuesto arriba a la derecha, está destinado a suscribir manualmente a los usuarios que consideremos oportuno. Para ello tenemos un listado con los usuarios que hay ahora mismo registrados en la aplicación. Para suscribir a un usuario a la alarma, basta marcar la casilla correspondiente al usuario. También disponemos de la opción de filtrar a los usuarios por su nombre en la entrada de texto habilitada para ello.

El tercer bloque ubicado abajo a la izquierda es un bloque informativo. En este podemos recibir la información de qué jugadores tenemos marcados y que por consiguiente van a ser añadidos a la alarma. De esta manera, ya sea que marquemos a los jugadores de manera manual, automática o mixta, sabremos en todo momento que jugadores van a ser introducidos.

El último y cuarto bloque, ubicado abajo a la derecha, es también un bloque informativo. Éste cumple la misma función que el tercero, pero con los usuarios que van a ser suscritos. De esta manera, podemos saber de manera rápida, qué usuarios van a ser suscritos antes de crear la alarma.

Logs generados

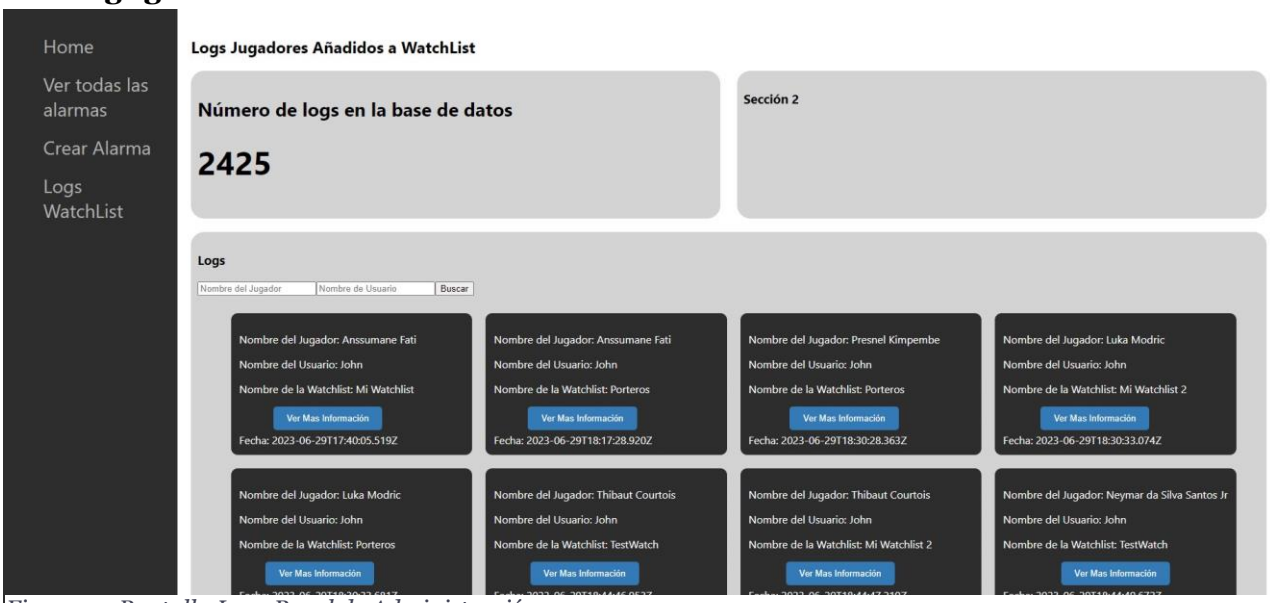


Figura 32 Pantalla Logs Panel de Administración

En esta pantalla representada en la figura 32, podemos observar la pantalla de *logs*. En esta pantalla aparecen todos los registros almacenados en la base de datos. Los *logs* o registros se crean cada vez que algún usuario añade a algún jugador a una WatchList. Como se puede observar, tenemos la opción de filtrar los registros por nombre de usuario, con el fin de poder ver qué jugadores está añadiendo algún usuario en particular. Podemos hacer una búsqueda por jugador para ver que usuarios están guardando al jugador o incluso hacer una búsqueda mixta de usuario y jugador.

En cada registro podemos observar alguna información de especial relevancia como es el nombre del jugador añadido, qué usuario ha añadido al jugador, así como a qué WatchList ha sido añadido además de la fecha y hora a la que ha ocurrido. Si pulsamos el botón Ver más información, se nos despliega una ventana modal con más información relevante del registro divida en tres bloques. El primer bloque, ordenado de izquierda a derecha, nos presenta información extra del jugador que ha sido añadido. El segundo bloque nos muestra la información del usuario y el ultimo nos muestra el nombre de la WatchList y otros jugadores que están dentro de la misma.

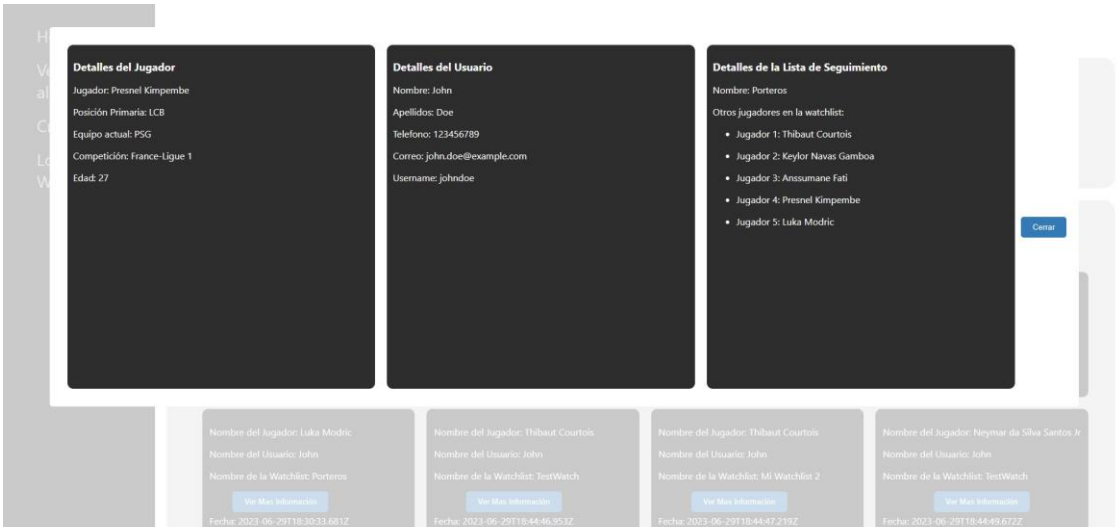


Figura 33 Ventana modal más información del Log

6. Pruebas

6.1. Pruebas unitarias

Vamos a realizar una serie de pruebas unitarias al código de nuestra API ya que es a la que más sentido tiene que se las realicemos, al ser la que se encarga de controlar el flujo de información, por lo que tenemos que asegurar la integridad de los datos y que no acepte valores de entrada erróneos.

Como nuestra API está desarrollada en JavaScript, utilizaremos una biblioteca de pruebas llamada Jest. Jest es una herramienta potente y ampliamente utilizada que simplifica el proceso de escribir, ejecutar y mantener pruebas unitarias. A través de Jest, el proceso de escritura de pruebas Jest implica la creación de pruebas específicas para cada unidad de código. Estas pruebas aíslan la funcionalidad de una unidad de código y evalúan su comportamiento en diferentes situaciones y con diversos datos de entrada. Es importante enfocar las pruebas unitarias en escenarios específicos y casos límite para garantizar una cobertura exhaustiva. Además, Jest ofrece una serie de funciones para comparar resultados esperados con resultados reales, como expect, toBe, toEqual y otras, que ayudan a validar la precisión de la implementación.

Las pruebas se realizaron a los métodos más críticos de las diferentes sub-Apis que hay en el proyecto.

Test1 : Recuperación de jugadores

Descripción	En esta prueba se va a comprobar mediante el método expect() que los métodos /getPlayersById y /getPlayersByName devuelven el JSON correspondiente.
Métodos	2

En la figura 33 se representa la forma que tienen los test unitarios. En este caso se explica el caso del test /getPlayersById, haciendo la consulta y comparando la respuesta obtenida con la esperada

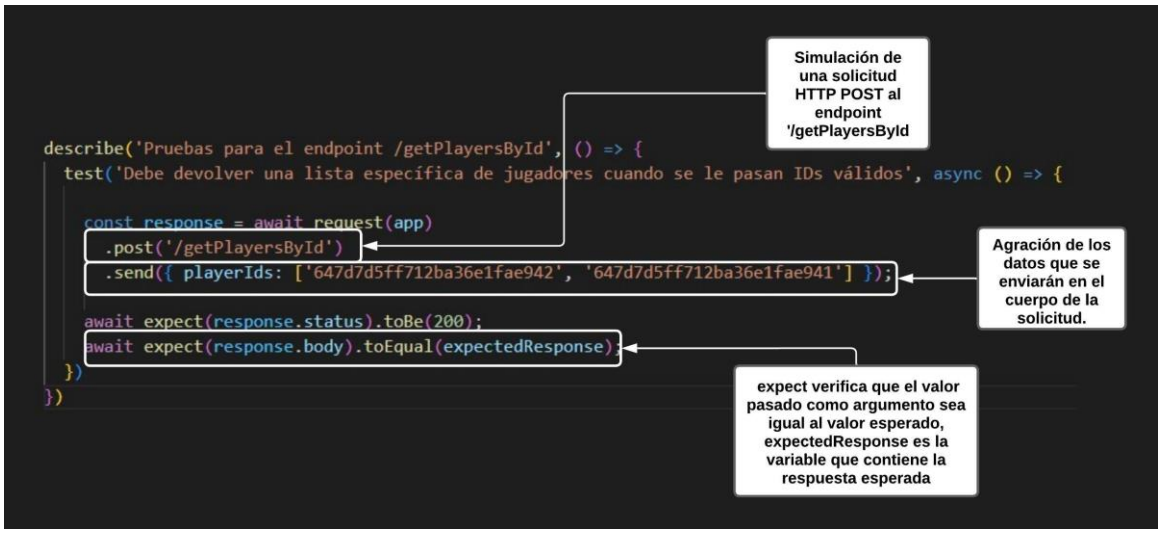


Figura 34 Muestra de código pruebas unitarias

Test2: Búsqueda de jugadores con parámetros.

Descripción	En esta prueba se va a comprobar mediante el método expect() que el método (/searchPlayers/:primary_position/:current_team_name /:domestic_competition_name/:age) recibe correctamente los parámetros de entrada y procesa una respuesta correcta con distintas combinaciones de parámetros de entrada incluyendo parámetros vacíos o nulos.
Métodos	1

Test3: Búsqueda de registros con parámetros.

Descripción	En esta prueba se va a comprobar mediante el método expect() que el método (/getLogsByPlayerAndUserName/:fullname/:name) recibe correctamente los parámetros de entrada y procesa una respuesta correcta con distintas combinaciones de parámetros de entrada incluyendo parámetros vacíos o nulos.
Métodos	1

Test4: Recuperación de usuarios

Descripción	En esta prueba se va a comprobar mediante el método expect() que el método /getUserById devuelve el JSON correspondiente.
Métodos	1

Test5: Recuperación de WatchList

Descripción	En esta prueba se va a comprobar mediante el método expect() que el método / getWatchListById devuelve el JSON correspondiente.
Métodos	1

Test6: Inserción de jugador a WatchList

Descripción	En esta prueba se va a comprobar mediante el método expect() que el método /addPlayerWatchList inserta el JSON correspondiente sin variaciones en la base de datos. Para ello vamos a insertar un jugador de prueba y recuperarlo para comprobar la integridad de los datos
Métodos	1

6.2. Validación con usuarios

Hemos implementado test unitarios con Jest para evaluar si la lógica de las clases de la API era correcta. Sin embargo, para las aplicaciones carece de sentido ya que lo que realmente cobra importancia es que cumplan el objetivo para el que fueron desarrolladas. Es decir, en este caso vamos a medir a la aceptación de los usuarios mediante cuestionarios para evaluar la usabilidad de ambas aplicaciones.

Para la evaluación de la usabilidad vamos a utilizar la métrica SUS (System Usability Scale). Esta métrica cuenta con 10 enunciados predefinidos. Estos enunciados son los siguientes:

1. Creo que me gustaría utilizar este sistema con frecuencia
2. Encontré el sistema innecesariamente complejo
3. Pensé que el sistema era fácil de usar
4. Creo que necesitaría el apoyo de un técnico para poder utilizar este sistema
5. Encontré que las diversas funciones de este sistema estaban bien integradas
6. Pensé que había demasiada inconsistencia en este sistema
7. Me imagino que la mayoría de la gente aprendería a utilizar este sistema muy rápidamente
8. Encontré el sistema muy complicado de usar
9. Me sentí muy seguro usando el sistema
10. Necesitaba aprender muchas cosas antes de empezar con este sistema

Cada enunciado se evalúa siguiendo la Escala de Likert. Esta escala es una herramienta de medición utilizada en investigaciones y encuestas para medir la actitud o nivel de acuerdo de las personas con respecto a una serie de afirmaciones o preguntas. Se compone de una serie de afirmaciones o declaraciones a las que los encuestados deben responder eligiendo uno de varios niveles de acuerdo.

1. Totalmente en desacuerdo
2. En desacuerdo
3. Ni de acuerdo ni en desacuerdo
4. De acuerdo
5. Totalmente de acuerdo

Para calcular el resultado después de que los usuarios hayan completado el cuestionario según la Escala de Likert haremos los siguientes cálculos.

1. Sumaremos las respuestas de los enunciados impares y después restaremos 5
2. Sumaremos las respuestas de los enunciados pares y restaremos a 25 ese total.
3. Por último, sumaremos ambos cálculos y los multiplicaremos por 2,5.
4. El resultado que obtengamos será inaceptable si está por debajo de 50, marginal si esta entre 50 y 68, y aceptable si está por encima de 68.

Figura 35 Escala SUS

	PREGUNTAS									
USUARIO	1	2	3	4	5	6	7	8	9	10
1	5	1	5	1	4	2	5	1	4	1
2	4	2	4	1	4	2	5	1	4	1
3	4	1	5	1	5	1	5	1	4	1
Total	13	4	14	3	13	5	15	3	12	3

Figura 36 Resultados de las pruebas con usuarios

Se ha pedido a varios posibles usuarios de la aplicación que evalúan su usabilidad mediante la métrica SUS que se ha expuesto antes. Si aplicamos el cálculo explicado anteriormente obtenemos lo siguiente.

1. Sumamos las respuestas de las preguntas impares, al ser 3 usuarios dividimos entre 3 y le restamos 5:
    a. (((13+14+13+15+12) / 3) – 5) = 17,33
2. Sumamos las respuestas de las preguntas pares, al ser 3 usuarios dividimos entre 3 y lo que nos dé se lo restamos a 25.
    a. (((25-((4+3+5+3+3)/3)=19
3. Sumos los dos resultados obtenidos y lo multiplicamos por 2,5. Con ello obtenemos lo siguiente.
    a. ((17,33 + 19) * 2,5) = 90,82

Nuestra medición de usabilidad con usuarios ha obtenido una puntuación de 90,82, por lo que según la figura 34 nuestra aplicación tiene una usabilidad Aceptable.

7. Conclusiones

Este proyecto se inició con un objetivo específico. Tener una aplicación en la que poder recibir alertas de ciertos parámetros del mundo del futbol de una manera clara y sencilla. El objetivo era poder automatizar a los técnicos del mundo del fútbol las búsquedas periódicas de nuevos talentos en un ámbito muy concreto. De esta manera cuando se cumpla el parámetro que desean, puedan recibir una notificación en su teléfono móvil y ver de manera clara qué jugador es el que ha cumplido el parámetro especificado, además de poder ver información actualizada del mismo. Además de esto, los técnicos podrían guardar los jugadores que sean de su interés en listas personalizadas con el fin de poder hacer un seguimiento periódico de sus estadísticas de juego jornada a jornada.

Como se ha podido observar a lo largo de la memoria, estos objetivos se han podido cumplir satisfactoriamente.

Cabe también recalcar que uno de los objetivos principales del proyecto era poder realizar un PMV con el cual probar la aceptación de un grupo reducido de personas antes de seguir trabajando en el producto que acabe llegando al mercado. Esto se debe a la propuesta de colaboración que se me presentó con trabajadores en activo del mundo del fútbol, gracias a mi tutor, el cual pudo ponerme en contacto con todos estos profesionales. De manera que, una vez finalizado este proyecto de fin de carrera, si la aceptación es buena, se pueda seguir trabajando en un proyecto serio y profesional.

Dado que hay un interés real de profesionales por el desarrollo de este producto se añadió también al desarrollo de la aplicación de manera paralela, el desarrollo de un panel de administración. El objetivo principal de este panel es el de poder observar y controlar que es lo que ocurre en la aplicación además de poder gestionarla.

Es bien sabido que hoy en día los datos cada vez están tomando mayor relevancia en el mundo, y el fútbol siendo una industria que mueve miles de millones a nivel mundial no es una excepción. Es por eso por lo que a nivel del desarrollo vemos, por ejemplo, el apartado de registros o *logs* en el panel de administración. Este panel a simple vista poco útil, brinda un información muy importante y estratégica para las personas que tengan acceso, ya que gracias a estos registros podemos observar que jugadores son interesantes para algunos usuarios o equipos y con ello extraer información valiosa de posibles estrategias de mercado que puedan seguir. Por lo que quien tenga esta información acaba poseyendo una posición ventajosa, dado el poder que brindan los datos a día de hoy.

A nivel personal, el desarrollo de este proyecto supone una gran oportunidad, no solo por el nivel de aprendizaje alcanzado por la cantidad de tecnologías aplicadas, si no por las posibles salidas profesionales.

Hoy en día el desarrollo de aplicaciones móvil y aplicaciones web se está convirtiendo en el foco principal de muchas empresas y he podido poner en práctica el desarrollo de éstas con tecnologías y métodos de desarrollo modernos. Ambas aplicaciones han sido desarrolladas con React, una tecnología que utilizan grandes empresas para sus aplicaciones como Facebook, Tesla, Microsoft, etc. He podido aprender desde cero cómo funciona esta tecnología y poder implementar aplicaciones funcionales que posiblemente tengan un uso real. Además de poder crear un sistema de comunicación entre ambas aplicaciones y la base de datos moderno y escalable según las necesidades del proyecto gracias a la implantación de una API REST. En definitiva, gracias a este proyecto he podido adquirir y poner en práctica habilidades de un desarrollador *full stack* que sin duda me serán de gran ayuda a nivel profesional en un futuro.

Aun así, este proyecto no habría tampoco sido posible sin el aprendizaje adquirido en dos asignaturas de la carrera que creo que tienen especial relevancia en el desarrollo de cualquier ingeniero:

-Análisis y Especificación de Requisitos: Gracias a esta asignatura puede adquirir los conocimientos necesarios para poder realizar una reunión en la que poder especificar y entender los requisitos que se requerían en el proyecto y poder realizar la documentación necesaria para el mismo.

-Proyecto de Ingeniería de Software: En esta asignatura pude aprender los principios de la metodología ágil y como llevarlos a cabo. De esta manera pude estimar la duración del proyecto y poder dividirlo en etapas o *sprints* para poder llevar un desarrollo ordenado y planificado del proyecto.

8. Trabajos futuros

Como es de esperar en este proyecto dado su limitación de tiempo se han tenido que escoger e implementar las funcionalidades más relevantes para lograr un producto mínimo viable. Pero en las reiteradas reuniones con los profesionales del sector se han detallado una lista de requisitos que conforman las futuras implementaciones que debería incluir una versión final del producto. Algunas de ellas son:

- Crear un repositorio de alarmas de distinta índole a la que los usuarios se puedan suscribir.

- Crear un sistema de creación de alarmas personalizadas para cada usuario.

- Crear un sistema de WatchList colaborativas en la que puedas compartirlas con otros usuarios o grupos de trabajo y que varios usuarios puedan añadir jugadores a las mismas.

- Añadir gráficos temporales a las cartas de los jugadores para poder ver de manera grafica la evolución de las estadísticas de éstos a lo largo de la temporada.

- Crear todo el sistema de facturación y monetización de la aplicación, este consistirá en un sistema de créditos. Con estos créditos los usuarios podrán suscribirse a alarmas. Se podrán comprar créditos de manera individual o recurrir a un sistema de pago mensual en el que dependiendo del tipo de suscripción tengas uno número distinto de posibles interacciones. Además, se añadirá la posibilidad de pago por alarmas personalizadas individualmente.

- Añadir cifrado de datos a las comunicaciones que reciba la API para añadir seguridad e integridad a los datos.

- Añadir estadísticas más detalladas del estado y tamaño de la aplicación en el panel de administración web

- Refactorizar el diseño de la aplicación con diseños de calidad para dar un estilo profesional a la aplicación.

9. Referencias

1. Scouting, Scout, Analista, Ojeador...: [03/2023] https://objetivoanalista.com/scouting-scout-analista-ojeador/
2. Scouting de fútbol: [03/2023] https://www.misentrenamientosdefutbol.com/diccionario/scouting-de-futbol
3. Scouting de fútbol: Historia y metodología en un equipo profesional [03/2023] https://objetivoanalista.com/scouting-futbol-historia-metodologia-equipo-profesional/
4. Scouting en fútbol, las nuevas tecnologías: [03/2023] https://www.futbollab.com/es/noticia/scouting-futbol-nuevas-tecnologias
5. Wyscout Platform: [03/2023] https://platform.wyscout.com/
6. Wyscout Wikipedia: [03/2023] https://en.wikipedia.org/wiki/Wyscout
7. Scout7, A BESPOKE SOFTWARE FOR SCOUTING [03/2023] https://www.sportperformanceanalysis.com/article/scout7-a-key-software-for-scouting
8. InStat – La Plataforma de Scouting Que Tu Equipo Necesita [03/2023] https://objetivoanalista.com/instat-plataforma-de-scouting/
9. Instatscout: [03/2023] https://instatscout.com/es/login
10. Objetivos y metas de desarrollo sostenible [04/2023] https://www.un.org/sustainabledevelopment/es/objetivos-de-desarrollo-sostenible/
11. Guía para la redacción de casos de uso: [04/2023] https://www.juntadeandalucia.es/servicios/madeja/contenido/recurso/416
12. React Native · Learn once, write anywhere [03/2023] https://reactnative.dev/
13. React Native - Wikipedia, la enciclopedia libre [04/2023] https://es.wikipedia.org/wiki/React_Native
14. React – Una biblioteca de JavaScript para construir interfaces [05/2023] https://es.react.dev/
15. Qué es NodeJS y para qué sirve [04/2023] https://openwebinars.net/blog/que-es-nodejs/
16. ¿Qué Es MongoDB? [04/2023] https://www.mongodb.com/es/what-is-mongodb
17. Qué es mongoose: [05/2023] https://codigofacilito.com/articulos/que-es-mongoose
18. Delightful JavaScript Testing: [06/2023] https://jestjs.io/es-ES/
19. Express - Node.js web application framework [05/2023] https://expressjs.com/
20. What Is Express.js? Everything You Should Know [04/2023] https://kinsta.com/knowledgebase/what-is-express-js/

21. REST API [04/2023]
https://www.redhat.com/es/topics/api/what-is-a-rest-api
22. ¿Qué es el protocolo HTTP? | KeepCoding Bootcamps [04/2023]
https://keepcoding.io/blog/que-es-el-protocolo-http
23. Qué es escala de Likert y cómo aplicar en 3 simples pasos: [08/2023]
https://www.zendesk.com.mx/blog/que-es-escala-de-likert/
24. ¿La usabilidad puede medirse? Escala SUS y test de usuario [08/2023]
https://www.flat101.es/blog/diseno-ux/la-usabilidad-puede-medirse-escala-sus-y-test-de-usuario/
25. Presentando JSX [05/2023]
https://es.legacy.reactjs.org/docs/introducing-jsx.html
26. Visual Studio Code [06/2023]
https://es.wikipedia.org/wiki/Visual_Studio_Code

ANEXO I

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

Como se ha indicado en la tabla superior, el desarrollo del proyecto podría tener relación con algunos de los Objetivos de Desarrollo Sostenible establecidos por las Naciones Unidas. Algunos de los ODS que podrían estar relacionados son los siguientes:

ODS 1 - Fin de la pobreza: Bajo

Nuestra aplicación podría potencialmente contribuir a la igualdad de oportunidades al permitir que los jóvenes talentos del fútbol sean descubiertos y evaluados, lo que podría beneficiar a aquellos que provienen de entornos desfavorecidos.

ODS 4 - Educación de calidad: Medio

Nuestra aplicación podría facilitar la identificación de jóvenes talentos y su acceso a oportunidades de desarrollo en el mundo del fútbol, con lo que consecuentemente obtendrían una educación de calidad.

ODS 5 - Igualdad de género: Bajo

Si nuestra aplicación se utiliza para descubrir talentos tanto masculinos como femeninos, podría contribuir a la igualdad de género en el deporte.

ODS 8 - Trabajo decente y crecimiento económico: Bajo

Si nuestra aplicación tiene éxito, podría contribuir a la generación de empleo en la industria del fútbol y el deporte.

ODS 9 - Industria, innovación e infraestructura: Medio

El desarrollo de una aplicación de este tipo implica innovación y podría mejorar la infraestructura del *scouting* deportivo.

ODS 10 - Reducción de las desigualdades: Alto

Nuestra aplicación podría ayudar a reducir las desigualdades al ofrecer a jóvenes talentos una oportunidad justa de ser descubiertos y evaluados.

ODS 16 - Paz, justicia e instituciones sólidas: Medio

Si nuestra aplicación contribuye a un proceso de selección más justo y transparente en el mundo del fútbol, podría estar relacionada con este ODS.

ODS 17 - Alianzas para lograr los objetivos: Bajo

La colaboración con equipos de fútbol, agentes deportivos y otros interesados podría ser crucial para el éxito de nuestro proyecto.