UNIVERSIDAD DE MÁLAGA

E.T.S. INGENIERÍA INFORMÁTICA

UNIVERSIDAD DE MÁLAGA

Grado en Ingeniería Informática

Computación científica

Plataforma de Búsqueda y Análisis de Sucesión de Eventos en Partidos de Fútbol

Platform for searching and analyzing event sequences in football matches

Realizado por

Ángel Zorrilla Martínez

Tutorizado por

Dr. Javier Ferrer Urbano

Co-tutorizado por

Dr. Christian Cintrano López

Departamento

Departamento de Lenguajes y Ciencias de la Computación

Málaga, November 25, 2025

UNIVERSIDAD DE MÁLAGA

E.T.S. INGENIERÍA INFORMÁTICA
UNIVERSIDAD DE MÁLAGA

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA

GRADO EN INGENIERÍA INFORMÁTICA

Computación científica

Plataforma de Búsqueda y Análisis de Sucesión de Eventos en Partidos de Fútbol

Platform for searching and analyzing event sequences in football matches

Realizado por

Ángel Zorrilla Martínez

Tutorizado por

Dr. Javier Ferrer Urbano

Co-tutorizado por

Dr. Christian Cintrano López

Departamento

Departamento de Lenguajes y Ciencias de la Computación

UNIVERSIDAD DE MÁLAGA

MÁLAGA, NOVEMBER 25, 2025

Fecha defensa: diciembre de 2025

Resumen

En los últimos años, el fútbol profesional ha incorporado el análisis de datos para apoyar la toma de decisiones tácticas y de *scouting*. Sin embargo, las herramientas actuales se centran en estadísticas aisladas de equipos o jugadores y no permiten explorar las secuencias de eventos que hay detrás de cada estadística. Este Trabajo Fin de Grado propone el desarrollo de una plataforma web que facilita la búsqueda y análisis de secuencias de eventos en partidos de fútbol profesional a partir de datos abiertos.

La solución se construye sobre la base de datos de *StatsBomb Open Data*, para la que se diseña un proceso completo de ingesta, normalización y carga en un modelo relacional capaz de soportar consultas complejas manteniendo el contexto temporal y espacial de las jugadas. Sobre esta base de datos se implementa un *backend* que expone una API REST para lanzar búsquedas de patrones, calcular métricas agregadas y generar *rankings* de jugadores por rol, apoyado en un motor de búsqueda secuencial que interpreta patrones definidos por el usuario. El *frontend*, desarrollado como aplicación web, permite definir jugadas directamente sobre el campo, configurar filtros y explorar los resultados mediante resúmenes, *rankings* y estadísticas avanzadas por jugador.

La herramienta se valida mediante pruebas funcionales y dos casos de estudio: la búsqueda de un extremo derecho con un perfil táctico específico y el análisis de un patrón de centros laterales al segundo palo. Estos ejemplos muestran que la plataforma es capaz de recuperar patrones complejos de juego y ofrecer información útil para el análisis táctico y el *scouting*. Esto demuestra que es posible construir, a partir de datos abiertos y tecnologías accesibles, una herramienta que acerca el análisis secuencial de jugadas al trabajo cotidiano de clubes, analistas y proyectos académicos.

Palabras clave: fútbol, patrones, secuencia de eventos, análisis táctico, ojeadores, plataforma web, datos abiertos.

Abstract

In recent years, professional football has increasingly incorporated data analysis to support tactical decision-making and scouting. However, many of the tools currently available focus on isolated statistics for teams or players and do not allow analysts to explore the sequences of events that lie behind each metric. This Final Degree Project proposes the development of a web platform that facilitates the search and analysis of event sequences in professional football matches using open data.

The solution is built on top of the StatsBomb Open Data, for which a complete ingestion, normalisation and loading process has been designed into a relational model capable of supporting complex queries while preserving the temporal and spatial context of each play. On this database, a backend exposes a REST API to launch pattern searches, compute aggregated metrics and generate player rankings by role, relying on a sequential search engine that interprets patterns defined by the user. The frontend, developed as a web application, allows the user to define plays directly on the pitch, configure filters and explore the results through summaries, rankings and advanced per-player statistics.

The tool is validated through functional tests and two case studies: the search for a right winger with a specific tactical profile and the analysis of a pattern of wide crosses to the far post. These examples show that the platform is capable of retrieving complex patterns of play and providing useful information for tactical analysis and scouting. This project demonstrates that it is possible to build, from open data and accessible technologies, a tool that brings sequence-based analysis of plays closer to the everyday work of clubs, analysts and academic projects.

Keywords: football, patterns, event sequences, tactical analysis , scouting, open data, web platform.

Agradecimientos

En primer lugar, quiero agradecer a mi familia todo lo que ha hecho por mí. Siempre me habéis exigido dar lo mejor, incluso cuando yo mismo dudaba, y al mismo tiempo habéis confiado siempre en mí para sacar las cosas adelante. Gracias por todo el apoyo y cariño que he recibido, esto ha sido el motor que me ha acompañado durante todos estos años y la razón por la que hoy puedo estar cerrando esta etapa.

También quiero dar las gracias a mi novia y mis amigos, los de dentro y fuera de la universidad, por hacerme disfrutar de cada momento de la vida. Gracias por estar ahí cuando tocaba desconectar, por sacarme una sonrisa con cualquier tontería, pero sobre todo por hacerme la vida mucho más bonita incluso en los momentos más difíciles de cada uno.

A mis compañeros y compañeras de carrera, quiero decirles que sin ellos este camino no habría sido igual ni, probablemente, habría llegado hasta aquí. La cantidad de horas que hemos pasado en la biblioteca, haciendo del estudio y de los agobios algo mucho más ameno sabiendo que juntos podíamos con cualquier examen.

Por último, quiero agradecer a la educación pública. Desde la escuela hasta la universidad, he tenido la oportunidad de aprender y crecer gracias al esfuerzo de muchas personas que creen en una educación accesible y de calidad. A todas ellas, mi reconocimiento y mi agradecimiento sincero por haber hecho posible que hoy pueda presentar este Trabajo Fin de Grado.

Nomenclatura

API	*Application Programming Interface*. Interfaz de programación de aplicaciones.
BD	Base de datos.
CORS	*Cross-Origin Resource Sharing*. Mecanismo de control de acceso entre orígenes en aplicaciones web.
CRUD	*Create, Read, Update, Delete*. Conjunto de operaciones básicas sobre datos persistentes.
CSS	*Cascading Style Sheets*. Lenguaje de estilos para documentos HTML.
CSV	*Comma-Separated Values*. Formato de fichero de texto para datos tabulares.
ETL	*Extract, Transform, Load*. Proceso de extracción, transformación y carga de datos.
HTTP	*HyperText Transfer Protocol*. Protocolo de comunicación para la web.
HTTPS	*HyperText Transfer Protocol Secure*. Versión segura de HTTP con cifrado TLS.
JSON	*JavaScript Object Notation*. Formato de intercambio de datos basado en texto.
PIB	Producto Interior Bruto.
REST	*Representational State Transfer*. Estilo de arquitectura para servicios web.
SPA	*Single Page Application*. Aplicación web de página única que se carga de forma dinámica.
SQL	*Structured Query Language*. Lenguaje de consultas a bases de datos relacionales.
TFG	Trabajo Fin de Grado.
TL	Transformación y limpieza de datos utilizada en el proceso de preparación del *dataset*.
UI	*User Interface*. Interfaz de usuario de la aplicación.
URL	*Uniform Resource Locator*. Identificador de un recurso accesible en la web.
Vite	Herramienta de construcción y servidor de desarrollo para aplicaciones web en *JavaScript*/*TypeScript*.
xG	*expected goals*. Métrica que estima la probabilidad de que un tiro termine en gol.

1 Introducción

1.1 Motivación

El fútbol es más que un deporte: su capacidad de reunir personas más allá de su género, edad, clase social y religión lo convierten en un fenómeno cultural que trasciende fronteras y despierta pasiones por todo el mundo. Además, el fútbol tiene un impacto económico a nivel mundial. Solo en España, según un informe presentado por LaLiga[8] "la industria del fútbol profesional en España cuantifica su impacto socioeconómico en 194.381 empleos, 8.390 millones de euros en impuestos y una facturación equivalente al 1,44 % del PIB español". Esto muestra que el fútbol no solo es un deporte, sino también un motor económico, social y cultural.

En los últimos años, el fútbol ha vivido una revolución tecnológica que ha cambiado la forma de jugar, entrenar, ojear, arbitrar e incluso de ver el fútbol. Entre las innovaciones más importantes se encuentran el videoarbitraje, tecnologías como el ojo de halcón y el fuera de juego semiautomático, la monitorización GPS de los jugadores para el posterior análisis de su rendimiento físico y el desarrollo de métricas avanzadas para el análisis de partidos. Sin ir más lejos, equipos muy conocidos como el Manchester City o el FC Barcelona han adoptado estas herramientas e incluso han ido más allá. El primero, según Innovación Digital[6] "ha implementado herramientas de análisis predictivo de jugadas e Inteligencia Artificial tanto para las sesiones técnicas como para prevenir riesgos de lesiones". El segundo, según fuentes como Barça Innovation Hub[1] "es pionero en usar Inteligencia Artificial para el scouting, e incluso ha explorado el uso de herramientas de realidad virtual para simular situaciones de juego en los entrenamientos". Esto no solo demuestra la importancia de la tecnología en el fútbol de alto nivel, sino que apoya la idea de que si quieres competir por ser de los mejores, tienes que adaptarte al fútbol moderno y adoptar las nuevas corrientes tecnológicas.

En este contexto de crecimiento, en un deporte donde cada detalle es importante y cualquier error puede representar grandes pérdidas, los clubes buscan cada vez más mejorar el rendimiento de sus jugadores y equipos mediante herramientas tecnológicas. Para el análisis táctico se ha vuelto imprescindible el uso de tecnología de análisis de datos. Esto ha generado la necesidad de herramientas capaces de analizar grandes volúmenes de datos relacionados con el fútbol. La mayoría de ellos trata la información de miles de partidos y competiciones, por lo que sería imposible gestionarlos sin algoritmos de Inteligencia Artificial. Normalmente, estas herramientas están centradas en estadísticas específicas, como la distancia recorrida o los pases

completados, así como en otras métricas más avanzadas como estimaciones de goles esperados (xG) o la presión ejercida. Todo ello permite a los entrenadores tomar decisiones informadas y estratégicas.

En este proyecto, aunque nos situamos dentro de esta tendencia, nos alejamos de los enfoques que se quedan solo en estadísticas aisladas. Partimos de la idea de que el fútbol es un deporte de acciones encadenadas, donde cada acción no se entiende por sí sola, sino en función del contexto en el que ocurre. Por eso, nos centramos en analizar la sucesión de eventos que se dan durante los partidos y en cómo el contexto de cada acción influye en el desarrollo de la jugada. A través de la identificación de patrones de juego, buscamos ofrecer una herramienta útil para el análisis táctico de los entrenadores y para el *scouting*.

1.2 Objetivos

Objetivo General

El objetivo de este proyecto es diseñar y desarrollar una plataforma web capaz de identificar secuencias de eventos en partidos de fútbol profesional. Una plataforma que ofrecerá una interfaz web desde la que usuarios, mayormente entrenadores y ojeadores, puedan definir jugadas como una secuencia de eventos sobre un campo de fútbol y obtener como resultado todas las coincidencias detectadas en la base de datos de partidos de fútbol.

Un ejemplo fácil de entender sería imaginar a un ojeador que quiere buscar un jugador con un perfil parecido al de Lamine Yamal. Para ello, puedes utilizar la aplicación y solo tienes que definir sobre el campo de fútbol virtual un jugador que recibe el balón en banda derecha, regatea y sigue con el balón hacia la frontal del área donde dispara a puerta. La aplicación devolverá una lista con los jugadores que más realizan este tipo de acciones.

Subobjetivos

A partir del objetivo general del proyecto, se plantean los siguientes subobjetivos:

- Obtener y procesar los datos de partidos de fútbol a partir de las fuentes disponibles.
- Definir el modelo y la estructura de los datos, dando lugar a una base de datos capaz de procesar consultas complejas.
- Implementar un motor de búsqueda que permita encontrar patrones de eventos en la base de datos.
- Calcular métricas que midan el rendimiento de las jugadas y de los jugadores, como el porcentaje de tiros a puerta o de goles marcados, y así poder evaluar el impacto de los jugadores y de las jugadas.
- Desarrollar una plataforma web intuitiva para que el usuario pueda realizar las consultas y analizar los resultados obtenidos.

Resultados esperados

Como resultado de este proyecto se espera obtener una plataforma web funcional, fácil e intuitiva. Una base de datos sólida y óptima para consultas complejas. Un sistema de búsqueda adecuado a la dimensión de la base de datos y la complejidad de las consultas. Un manual de instrucciones de la aplicación. Una guía de instalación para cualquier persona. Y por último, una memoria que documente el desarrollo, implementación y validación del sistema.

1.3 Metodología del trabajo

La metodología de trabajo adoptada es Scrum[13]. Este enfoque iterativo se centra en entregar valor verificable de forma temprana, combinando diseño, implementación y pruebas en ciclos cortos. Esto permite detectar errores pronto y ajustar prioridades según salgan errores, aparezca nueva información o dado el *feedback* del cliente (los tutores en este caso). Esta elección responde a la necesidad de mantener los requisitos bajo control y asegurar el rumbo del proyecto.

La cadencia se desarrolla en *sprints* de dos semanas. En la que cada iteración empieza con una sesión de planificación, continúa con un seguimiento del proyecto y termina con una revisión donde se demuestra el incremento funcional y se plantea una retrospectiva para mejorar el proceso. Todo esto ayuda a conseguir un ritmo constante y un seguimiento cercano por parte de los tutores.

Los tutores actúan como *stakeholders*, aportando en las revisiones de sprint criterios y validación. Cada historia queda terminada cuando se cumple lo que se tenía previsto, con un código versionado, pruebas básicas y evidencias de funcionamiento. La memoria recoge esto con *sprints* dedicados a "procesamiento de datos", "sistema de consultas" o "diseño del *frontend*", donde *Scrum* ayudó a ajustar objetivos sin comprometer calidad.

Scrum se ajusta a las fases previstas en el anteproyecto: investigación del estado del arte, recopilación de datos y análisis de su estructura original, etc. Pero en vez de abordarlas en cascada, se cierran con verticales utilizables por *sprint*, un ejemplo sería un ETL mínimo con su primer conjunto de consultas y una vista básica de resultados. Esta organización se combina con la temporización planeada y con el objetivo de dejar incrementos verificables con cada iteración. Cada *sprint* deja un incremento ejecutable y documentado, de modo que la validación puede hacerse de forma continua, tal como se describe en la memoria.

1.4 Estructura de la memoria

La memoria se estructura en distintos capítulos que reflejan de forma secuencial el desarrollo del proyecto. Se parte desde la definición del problema y su motivación, hasta la evaluación final de los resultados obtenidos.

A continuación, se resume brevemente el contenido de cada capítulo:

- **Introducción.** En ella se expone la motivación del proyecto, los objetivos perseguidos, el enfoque tecnológico y la propia estructura del proyecto.
- **Estado del arte.** En este apartado se analizan herramientas actuales de análisis de fútbol y las limitaciones de las soluciones existentes y las bases de datos de dónde se obtiene la información.
- **Diseño del sistema.** Se detallan los requisitos del sistema, los casos de uso, la arquitectura por capas y el modelo de datos que sirve de base al motor de búsqueda y a la API.

- **Proceso de TL**. Se describe el flujo completo de obtención, limpieza, normalización y carga de los datos de StatsBomb Open Data hasta su almacenamiento en la base de datos relacional, junto con los mecanismos de refresco de tablas derivadas.
- **Implementación de la solución**. Se explica cómo se ha materializado el diseño en código: implementación del backend (API, motor de búsqueda y servicios de análisis) y del frontend (cliente web).
- **Validación y casos de uso.** Se presentan las pruebas realizadas, el análisis de los resultados obtenidos y las limitaciones que tiene el sistema.
- **Conclusiones y trabajo futuro.** Se exponen las conclusiones finales del proyecto, las dificultades encontradas y futuras líneas en las que se puede seguir trabajando.

2

Estado del arte

En este capítulo se va a revisar el estado del arte del análisis de datos orientado al fútbol. El objetivo es conocer las herramientas, técnicas y fuentes de datos actuales que utilizan los grandes equipos y ojeadores para el estudio de rendimiento táctico y búsqueda de patrones de juego en equipos y/o jugadores.

Esta revisión nos permite saber el contexto del problema, las soluciones que existen hoy en día y justificar la herramienta propuesta en este TFG, una plataforma para el análisis de secuencias en partidos de fútbol.

2.1 Herramientas actuales de análisis en fútbol

Como ya hemos visto antes, el fútbol es un deporte que se está adaptando a la tecnología y a los nuevos cambios que ésta produce. Es por ello que en los últimos años han salido nuevas plataformas y herramientas orientadas al análisis táctico de fútbol. La mayoría de ellas están enfocadas en asistir a los equipos, poniendo el foco en el uso por parte de los entrenadores o de los ojeadores. En este apartado vamos a analizar las más relevantes a día de hoy.

WyScout

WyScout es una de las plataformas de análisis táctico y *scouting* más usadas en el ámbito profesional [4]. Ofrece una base de datos muy extensa que, según Hudl [3], constituye la mayor biblioteca de vídeos y datos de fútbol del mundo, ya que recoge información de más de 600 competiciones a nivel global. Con esto permiten al usuario visualizar clips de videos de estos partidos etiquetados por eventos, como pueden ser pases, tiros y saques de esquina. Además, incorpora otras herramientas como estadísticas generales de los partidos, equipos y jugadores, así como funciones, entre las que destacan los mapas de calor, las redes de pases y los gráficos de rendimiento. Esto la hace una herramienta estándar en clubes y agencias. A pesar de su potencia y usabilidad, presenta una serie de limitaciones, como: no permitir al usuario buscar ni detectar patrones o secuencias complejas en el juego y que al tratarse de una herramienta de pago, restringe su accesibilidad al público.

InStat

InStat es otra plataforma muy utilizada por los clubes de fútbol [7]. Según cuenta Dani Pérez en su artículo dedicado a InStat [12], clubes como el Atlético de Madrid han apostado fuerte por estas herramientas.

Esta aplicación ofrece una serie de herramientas que facilitan el análisis táctico y el *scouting*, dando estadísticas de equipos, jugadores y acceso a clips de vídeo categorizados por tipo de acción en los partidos, en esto se parece a *WyScout*. Una de sus grandes virtudes es que tiene un sistema que puntúa a los jugadores tanto en los partidos como por su rendimiento general. Una de las herramientas más importantes que ofrece es *InStat Scout*, que permite buscar jugadores mediante filtrado de datos dando una serie de parámetros como la posición, edad, rendimiento, altura y origen. Aunque se distancie un poco más de *WyScout* por la última herramienta mencionada, comparten las mismas limitaciones: son de pago y no tienen la posibilidad de filtrar por secuencias de eventos en los partidos de fútbol.

2.1.1 Metrica Sports

Estoy seguro de que casi todo el mundo que seguimos fútbol en redes sociales hemos visto alguna vez un vídeo realizado con *Metrica Sports*. Aquí un ejemplo:`https://www.youtube.com/shorts/tgaKc1Ie0jc`

Metrica Sports es una plataforma centrada en el análisis visual de jugadas a partir de datos de seguimiento (*tracking*) y eventos [10]. Destaca por la posibilidad de generar animaciones tácticas, que permiten ver con claridad la posición de los jugadores, la trayectoria del balón y la evolución de las jugadas con el tiempo. Esta herramienta es utilizada por clubes y analistas, sobre todo para estudiar la gestión del espacio, las líneas defensivas y otros factores que suelen depender de la posición de los jugadores y el balón, como la presión coordinada.

Aunque esta herramienta puede ser realmente útil para el análisis y presentación visual de jugadas, Metrica Sports no está orientada en la búsqueda de patrones complejos durante el juego. Tampoco permite definir consultas a partir de secuencias de eventos, ya que su enfoque principal es la visualización.

2.1.2 *StatsBomb IQ*

StatsBomb ofrece una versión de datos abiertos, *StatsBomb Open Data* [15], la fuente que vamos a utilizar en este proyecto. En este apartado vamos a tratar *StatsBomb IQ*, que se enfoca en el análisis centrado en los datos.

StatsBomb IQ es una herramienta dirigida a clubes profesionales, analistas y cuerpos técnicos, que ofrece una gran variedad de información a partir de la anotación detallada de los eventos. Llega a ofrecer información de más de 3400 eventos por partido y cubre aproximadamente 140 competiciones [5]. Destaca que ofrece métricas avanzadas como los *expected goals (xG)*, que estima la probabilidad de que una jugada acabe en gol. Además de *On-Ball Value*, que puntúa el impacto de las acciones que realiza un jugador con balón. Sin embargo, no ofrece la posibilidad de filtrar por secuencia de eventos. En este contexto, se hace notable la falta de una solución que permita trabajar con secuencias definidas de eventos.

2.2 Técnicas y enfoques

El análisis de datos aplicado al fútbol ha evolucionado mucho estos últimos años, permitiendo que se estudie el rendimiento de jugadores y equipos de distintas maneras. Desde el análisis más básico, como el recuento de tiros, porcentaje de posesión o pases completados, hasta métricas más avanzadas como los goles esperados o redes de pases. Estas herramientas cumplen su función y son muy útiles, permitiendo día a día basar la toma de decisiones de los entrenadores en datos fiables.

Sin embargo, la mayoría de estas técnicas se centran en eventos individuales o datos aislados, sin tener en cuenta algo fundamental como el contexto o la secuencia completa de acciones que han participado en la jugada. En el fútbol, una acción no se entiende por sí sola, sino que ha sido la reacción de múltiples eventos en cadena, un regate que consigue el espacio suficiente para generar un centro que es rematado y acaba en gol. Estas estructuras constituyen una secuencia y su análisis permite detectar patrones de juego, comprender el juego colectivo y valorar la calidad de la toma de decisiones según su contexto. Esto aporta una perspectiva nueva, dejando de ver el fútbol como estadísticas simples y empezándolo a ver como un deporte de equipo, donde se juzga cada acción por su contexto y no por simples métricas.

Algunas plataformas ofrecen visualización de secuencias típicas de juego, pero no permiten al usuario definir sus propios patrones y explorar un *dataset* de partidos con estas secuencias configurables. Además, hacer esto último no es fácil, requiere combinar la información espacial y temporal, lo que implica un gasto mayor de procesamiento. Ante este problema, este TFG plantea una herramienta que permite buscar secuencias de juego definidas por el usuario sobre una base de datos real. Abriendo las puertas para el futuro del análisis táctico y automatizado.

2.3 Fuentes de datos abiertas

El análisis de datos orientado al fútbol requiere el acceso a fuentes de información detalladas y bien estructuradas, que representen y permitan representar lo que está pasando en el partido. El problema está en que la mayoría de grandes empresas que recopilan estos datos de eventos y seguimientos, como Statsbomb, ofrecen servicios a equipos y agencias profesionales, lo que limita considerablemente el uso de estos datos para fines académicos o por parte de desarrolladores independientes. La existencia de fuentes de datos abiertas y de calidad es algo importante para impulsar la investigación y nuevas técnicas como la que se ofrece en este TFG.

Una de las iniciativas recientes que impulsó la creación de este TFG es *StatsBomb Open Data*, un proyecto que pone a disposición pública una gran serie de datos sobre partidos de fútbol profesionales. Esta fuente proporciona archivos `.json` que contienen información estructurada sobre competiciones, partidos, alineaciones, eventos y las posiciones de los jugadores respecto al campo. Los datos de eventos describen cada acción del partido (una falta, un regate, un pase...), incluyendo el tipo de evento, la ubicación en el campo, jugadores implicados entre otros datos útiles. Este nivel de complejidad al describir cada acción permite realizar análisis tácticos a partir de la reconstrucción de jugadas y detección de patrones.

Existen otras fuentes abiertas de información de partidos de fútbol. Plataformas como *Kaggle* albergan bases de datos relacionados con resultados y estadísticas. Otras plataformas como *Metrica Sports*, antes mencionada, ofrecen pequeñas muestras gratuitas de datos de *tracking* y eventos. En general, estas alternativas no permiten el análisis secuencial que necesita este proyecto, ya que carecen de información detallada y un gran volumen de datos estructurados. Es por eso que *StatsBomb Open Data* ha sido considerada la mejor opción para la implementación y desarrollo de la herramienta propuesta.

2.4 Conclusión del estado del arte

Existen herramientas orientadas al análisis de fútbol, como *WyScout* o *InStat*, que son fundamentales para cualquier equipo profesional, ya que facilitan el trabajo de entrenadores y ojeadores. Sin embargo, todas comparten los mismos problemas: son herramientas cerradas y de pago, cuyo acceso está restringido normalmente a clubes o agencias de ojeadores.

Independientemente de las limitaciones de acceso, las plataformas actuales se centran en estadísticas individuales y objetivas basadas en la descripción puntual de las acciones. Si bien es cierto que este enfoque permite analizar el rendimiento individual de los jugadores o generar resúmenes gracias a las estadísticas, no proporciona una visión completa del partido. El análisis de secuencias de eventos implica ver el transcurso de juego en un partido como una cadena de acciones interdependientes entre ellas, lo que permite analizar la actuación colectiva de un equipo. Sin embargo, este tipo de análisis requiere de recursos y herramientas para estructurar y consultar datos, cosa que no está al alcance de cualquier usuario.

Conociendo esto, cobra gran importancia la disponibilidad de fuentes abiertas como lo es StatsBomb Open Data, esta fuente ofrece un conjunto de datos con un nivel de detalle que permite reconstruir jugadas. Pero no existe una herramienta que permita consultar dichos datos definiendo patrones personalizados. Por esto, este Trabajo Fin de Grado propone una solución para llenar este vacío, una plataforma web que aprovecha datos abiertos, proporcionados por *StatsBomb Open Data*, para facilitar el análisis de secuencias de eventos en partidos de fútbol, permitiendo así un enfoque de exploración no tan explotado como puede ser la identificación de patrones y estrategias dirigido tanto a profesionales como a investigadores u ojeadores.

3

Diseño del sistema

3.1 Requisitos

A continuación, se redactan los requisitos finales que casan con la implementación actual, se señalan como opcionales los que no están cerrados aún.

Requisitos funcionales

Los requisitos funcionales expuestos abajo especifican las funcionalidades y el comportamiento que debe cumplir la plataforma de análisis de secuencia de eventos para satisfacer las necesidades del usuario, siguiendo la idea del producto.

Requisitos funcionales del usuario

U.1 **Definir un patrón de jugada** como una secuencia ordenada de eventos definidos en un campo de fútbol.

U.2 **Aplicar filtros** por competición, temporada, equipo, jugador, tipo de evento, minuto y zona del campo al lanzar una consulta.

U.3 **Ejecutar la búsqueda** de jugadas que cumplan el patrón definido y los filtros establecidos.

U.4 **Visualizar los detalles de cada jugada encontrada**, incluyendo orden de eventos, minuto, jugadores y club, posiciones o zonas en el campo.

U.5 **Refinar una búsqueda ya ejecutada**, modificando filtros y criterios de ordenación para obtener una nueva lista de resultados sobre los nuevos parámetros.

U.6 **Visualizar estadísticas avanzadas por patrón**, como frecuencia de aparición, porcentaje de tiros a puerta o goles...

U.7 **Guardar patrones** y recuperarlos para su reutilización.

U.8 **Exportar resultados** de una búsqueda y compartirla con enlaces reproducibles.

U.9 **Autenticación**: registrarse, iniciar y cerrar sesión y recuperar contraseña

Requisitos funcionales del administrador

A.1 **Ingesta de datos**. Importar y actualizar la base de datos con archivos (JSON) mediante *scripts* o herramientas de *back-office*.

A.2 **Verificación de integridad**. Comprobar consistencia de los resultados y detectar anomalías en la base de datos.

A.3 **Gestión de usuarios y roles**. Asignar rol de administrador, bloquear/desbloquear usuarios e incluso forzar el reinicio de credenciales.

Requisitos funcionales del sistema

S.1 Proveer API para la **autenticación y gestión de sesión** segura.

S.2 Mantener un modelo de **roles** (usuario y administrador).

S.3 **Modelar y almacenar eventos** con índices (partido, equipo, jugador, minuto...) preservando el orden temporal.

S.4 **Motor de búsqueda** de patrones secuenciales y con tolerancias de tiempo/espacio configurables.

S.5 El sistema deberá **validar el formato y la integridad** de los archivos cargados.

S.6 **Entrega de resultados completa** y re-ejecución correcta cuando cambien patrón o filtros.

S.7 Propocionar un **CRUD de patrones guardados** por usuario.

S.8 Permitir la **exportación de resultados** y parámetros de búsqueda.

Requisitos no funcionales

Además de las funcionalidades anteriores, el sistema debe cumplir una serie de requisitos no funcionales relacionados con la calidad del servicio, la experiencia de uso y la operación de la plataforma.

RNF.1 **Usabilidad:** La interfaz debe ser intuitiva, permitiendo que usuarios inexpertos puedan definir un patrón, aplicar filtros y lanzar una búsqueda sin necesidad de formación previa.

RNF.2 **Rendimiento en consultas**. El tiempo de respuesta de las consultas no debe superar los 30 segundos. Permitiendo que el usuario pueda iterar sobre sus búsquedas sin esperas excesivas.

RNF.3 **Rendimiento y escalabilidad de la base de datos**. Tras los procesos de ingesta de datos (A.1) se deben crear o actualizar los índices sobre la base de datos manteniendo un rendimiento en consultas acorde al requisito anterior.

RNF.4 **Portabilidad:** El sistema debe ser accesible desde los navegadores web más usados (*Google Chrome*, *Mozilla Firefox*, *Microsoft Edge*) sin requerir instalaciones.

RNF.5 **Mantenibilidad y coherencia de la API**. El código debe estar modularizado y documentado de manera que sea posible realizar actualizaciones o añadir nuevas funcionalidades con un impacto acotado. La API debe exponer *endpoints* homogéneos, validar los parámetros de entrada y devolver códigos y formatos de error consistentes, facilitando la depuración y la integración con otras herramientas.

RNF.6 **Seguridad:** La importación de datos deberá validar el formato y evitar inyecciones de código y otros errores que puedan poner en riesgo la integridad el sistema. Las credenciales de usuario deben transmitirse mediante una conexión cifrada y almacenarse con un algoritmo de *hash* robusto.

RNF.7 **Disponibilidad y recuperación**. Deben existir copias de seguridad de la base de datos y procedimientos de restauración que permitan devolver el sistema a un estado consistente en caso de fallo..

RNF.8 **Observabilidad y trazabilidad**. El sistema debe generar registros (*logs*) de las operaciones, incluyendo las peticiones de búsqueda, las ingestas de datos y las acciones de administración. Además, debe mantener métricas de uso que permitan detectar errores recurrentes y orientar futuras optimizaciones.

3.2 Casos de uso

Los casos de uso son una descripción de cómo el usuario o sistema (actor) interactúan con otro sistema para lograr un objetivo específico.

3.2.1 Actores

Nuestro sistema tiene dos tipos de actores:

- **Usuario** (entrenador/analista): diseña patrones, busca, analiza resultados, (opcional) guarda/exporta.
- **Administrador** (*back-office*): importa/actualiza la base de datos, reconstruye índices, verifica integridad y (opcional) gestiona roles.

3.2.2 Descripción detallada de los casos de uso

CU-U01: Autenticación

Actor principal	Usuario
Objetivo	Acceder de forma segura (registro, iniciar/cerrar sesión, recuperar contraseña)
Precondiciones	
Postcondiciones	El usuario tiene sesión activa y accede a la pantalla de inicio.
Flujo principal	1. Introduce credenciales 2. El sistema valida y crea sesión 3. Acceso a las funcionalidades.
Requisitos que cumple	U.9, S.1, S.2

CU-U02: Diseñar patrón

Actor principal	Usuario
Objetivo	Configurar un patrón de eventos sobre un campo de fútbol
Precondiciones	CU-U01
Postcondiciones	El patrón queda listo para una posible consulta.
Flujo principal	1. El usuario accede al apartado de diseño de patrones. 2. Selecciona los eventos en el orden y espacio deseado. 3. Validación mínima de coherencia. 3. Confirmar patrón (buscar/guardar)
Requisitos que cumple	U.1, U.2, S.3, S.4

CU-U03: Búsqueda de patrón definido

Actor principal	Usuario
Objetivo	Obtener resultados paginados que cumplen patrón + filtros
Precondiciones	CU-U02
Postcondiciones	Resultados disponibles
Flujo principal	1. Pulsa **buscar** 2. *Backend* ejecuta búsqueda secuencial con filtros. 2. Devuelve primera página y total estimado. 3. La UI muestra la lista
Requisitos que cumple	U.2, U.3, S.3, S.4, S.6

CU-U04: Visualizar jugada

Actor principal	Usuario
Objetivo	Visualizar los detalles de de una jugada seleccionada.
Precondiciones	CU-U03
Postcondiciones	Detalles del patrón disponibles.
Flujo principal	1. Realizar una búsqueda. 2. Mostrar detalles. 3. Volver o refinar.
Requisitos que cumple	U.4, U.3, S.3, S.4

CU-U05: Refinar resultados

Actor principal	Usuario
Objetivo	Ajustar filtros/ordenación en la lista y re-ejecutar la consulta
Precondiciones	CU-U03
Postcondiciones	Resultados refinados disponibles
Flujo principal	1. Cambiar filtros/orden. 2. Realizar otra búsqueda. 3. Mostrar lista actualizada.
Requisitos que cumple	U.5, U.2, U.3, S.4, S.6

CU-U06: Visualizar estadísticas avanzadas del patrón

Actor principal	Usuario
Objetivo	Consultar estadísticas avanzadas del patrón .
Precondiciones	CU-U03
Postcondiciones	Métricas del patrón disponibles
Flujo principal	1. Seleccionar una jugada de la lista. 2. Mostrar detalle. 3. Volver o refinar.
Requisitos que cumple	U.6, U.3, S.4, S.6

CU-U07: Guardar patrón

Actor principal	Usuario
Objetivo	Guardar patrones personales, listarlos, editarlos y reutilizarlos.
Precondiciones	CU-U02
Postcondiciones	La búsqueda queda disponible para reutilización.
Flujo principal	1. Diseñar patrón 2. Ejecutar la opción guardar patrón 3. Realizar búsqueda o editar
Requisitos que cumple	U.7, U.1, S.7

CU-U08: Exportar/compartir resultados

Actor principal	Usuario
Objetivo	Exportar resultados como CSV/JSON o generar enlace reproducible
Precondiciones	CU-U03
Postcondiciones	Archivo o enlace disponible
Flujo principal	1. Realiza una búsqueda con éxito. 2. Elegir la opción exportar resultados. 3. Obtener un archivo con los resultados o un enlace reproducible.
Requisitos que cumple	U.8, U.3, S.8

CA-A01: Importar/actualizar base de datos

Actor principal	Administrador
Objetivo	Ingerir JSON con validación e idempotencia, actualizar índices.
Precondiciones	CU-U01
Postcondiciones	Nuevos datos son incluidos en el *dataset*
Flujo principal	1. El administrador selecciona uno o varios archivos JSON con datos de partidos. 2. El sistema valida la estructura de los archivos. 3. Ingerir nuevos datos sin duplicados. 4. El sistema transforma los datos al modelo interno y los inserta sin crear duplicados. 5. El sistema actualiza los índices necesarios y registra la operación de ingesta.
Requisitos que cumple	A.1, A.2, S.3, S.5

CA-A02: Verificación de integridad

Actor principal	Administrador
Objetivo	Reconstruir índices, verificar integridad, revisar *logs* y métricas, ejecutar backup-s/restauración
Precondiciones	CA-A01
Requisitos que cumple	A.2, S.3, S.5

CA-A03: Gestión de roles

Actor principal	Administrador
Objetivo	Asignar rol administrador, bloquear/desbloquear, forzar reseteo.
Precondiciones	CU-U01.
Postcondiciones	Actualizaciones en los roles.
Flujo principal	1. Accede a gestión de usuarios. 2. Selecciona el usuario objetivo 3. Realiza la acción (cambiar rol, bloquear/desbloquear). 4. El sistema confirma la operación.
Requisitos que cumple	A.3, U.9, S.1, S.2

3.2.3 Diagrama de casos de uso

El diagrama general de casos de uso del sistema se recoge en la Figura 3.1. Muestra la interacción entre el usuario y el sistema, ilustrando las funciones que tiene que hacer el sistema desde el punto de vista del usuario.

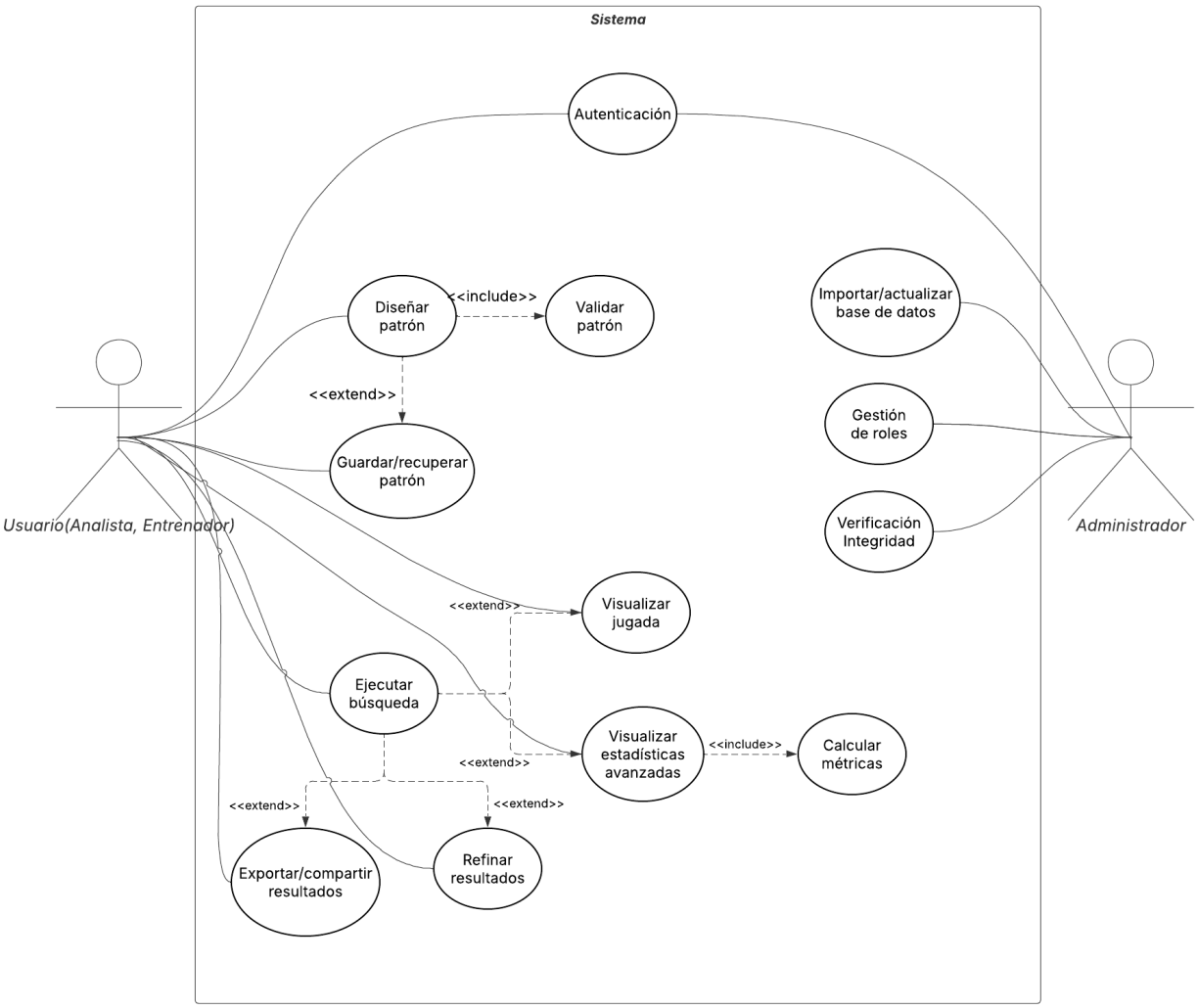


Figura 3.1. *Diagrama de casos de uso del sistema*

3.2.4 Criterios de aceptación

Los criterios de aceptación son un conjunto de condiciones específicas y medibles que han de cumplirse para que una funcionalidad o historia de usuario sea considerada completa y aceptada por los interesados. Definen el sistema, siendo así una guía para los equipos de desarrollo y una base sólida en las que basar las pruebas.

3.2.4.1 Flujos de Usuario

CA-U01 — Autenticación correcta (U.9, S.1, S.2, RNF.6)

Dado un usuario registrado, cuando inicia sesión con credenciales válidas, accede a la aplicación. Si usa credenciales inválidas, se muestra un claro error y se abre un flujo donde se le permite cambiar la contraseña.

CA-U02 — Diseño de patrón válido (U.1, U.2, S.3, RNF.1)

Dado un diseño sobre el campo, si el usuario define una secuencia ordenada de eventos sobre el campo, el sistema valida la coherencia de la operación y se le permite buscar o guardar.

CA-U03 — Búsqueda completa y dentro de tiempo (U.3, S.4, S.6, RNF.2)

Dado un patrón y filtros, cuando el usuario pulsa "Buscar", los resultados han de llegar en menos de 30 segundos.

CA-U04 — Re-ejecución (U.5, S.6, RNF.1)

Dado un listado de resultados, cuando el usuario cambia el patrón o los filtros y vuelve a "Buscar", la nueva lista refleja exclusivamente los resultados de los nuevos parámetros.

CA-U05 — Detalles de una jugada (U.4, S.3, S.4, RNF.1)

Dado un resultado, el usuario ve la secuencia en orden, los jugadores que más realizan esa jugada y cuantas veces se ha encontrado. Y la reproducción sobre el campo sigue exactamente esa secuencia.

CA-U06 — Estadísticas consistentes (U.6, S.3, RNF.2)

Dado un conjunto de resultados, cuando se muestran distintas métricas como porcentaje tiro a puerta y porcentaje de gol, dichas métricas coinciden con la realidad.

CA-U07 — Guardado y recuperación de patrones (U.7, S.7, RNF.1, RNF.7)

Dado un patrón nombrado, cuando el usuario lo guarda, entonces aparece en su lista personal. Y cuando lo recupera, entonces se rellena el diseñador con los valores del guardado.

CA-U08 — Exportar/compartir (U.8, S.8, RNF.4)

Dado una lista filtrada, cuando se exporta a CSV/JSON, entonces el fichero contiene como mínimo, los máximos participantes de esa jugada y cuántas veces se ha hecho.

3.2.4.2 Administración/back-office

CA-A01 — Ingesta idempotente y segura (A.1, S.3, S.5, RNF.2, RNF.6)

Dado un JSON válido, cuando se ejecuta la importación, entonces se validan esquema y claves, se normalizan coordenadas y no se crean duplicados; y cuando se re-importa el mismo archivo, entonces el estado de la base de datos no cambia. Y si falla la validación, entonces la base de datos queda intacta y se registra el error.

CA-A02 — Rendimiento tras ingesta (A.1, RNF.2, RNF.3)

Dado una importación exitosa, cuando se reconstruyen/analizan índices, entonces las consultas mantienen el objetivo de latencia en patrones típicos.

CA-A03 — Integridad referencial (A.2, S.3, S.5, RNF.6)

Dado la BD poblada, cuando se ejecuta la verificación, entonces no hay jugadores asociados a equipos/temporadas erróneos ni claves huérfanas.

CA-A04 — Observabilidad y trazas (A.1, A.2, RNF.8)

Dado el sistema en uso, cuando se consulta el panel /logs, entonces existen métricas básicas y *logs* por petición.

CA-A05 — Backups y restauración (RNF.7)

Dado una copia de seguridad, cuando se restaura, entonces la BD y sus índices regresan a un estado consistente y utilizable.

CA-A06 — Gestión de roles (A.3, S.2, RNF.6) — Opcional

Dado un usuario, cuando se le asigna rol admin, entonces puede ejecutar ingestas/mantenimiento; usuarios sin rol admin no pueden.

3.2.4.3 Calidad transversal (RNF)

CA-Q01 — Portabilidad (RNF.4)

El sistema funciona en las versiones soportadas de Chrome, Firefox y Edge sin instalaciones adicionales.

CA-Q02 — Mantenibilidad (RNF.5)

La API está documentada y existen *scripts* ETL versionados.

CA-Q03 — Seguridad mínima (RNF.6)

El despliegue usa HTTPS. Contraseñas guardadas con hash robusto. Validación estricta de JSON de importación.

3.2.5 Trazabilidad requisitos

La Tabla 3.1 recoge la matriz de trazabilidad entre requisitos, casos de uso y criterios de aceptación, mostrando qué pruebas validan cada requisito funcional y no funcional.

Tabla 3.1. *Matriz de trazabilidad Requisitos – Casos de uso – Criterios de aceptación*

Requisito	Casos de uso	Criterios de aceptación / Pruebas
U.1 Patrón de jugada	CU-U02	CA-U02 (validación coherencia, patrón buscar/guardar)
U.2 Aplicar filtros	CU-U02, CU-U03, CU-05	CA-U02 (definición de patrón y filtros), CA-U03 (búsqueda en tiempo), CA-U04 (re-ejecución)
U.3 Ejecutar búsqueda	CU-U03	CA-U03 (entrega completa en $\leq$ 30 s)
U.4 Detalle de jugada	CU-U04	CA-U05 (orden, minuto, jugadores, zonas, reproducción fiel)
U.5 Refinar búsqueda	CU-U05	CA-U04 (resultados reflejan parámetros)
U.6 Estadísticas del patrón	CU-U06	CA-U06 (coherencia de métricas con lista)
U.7 Guardar patrón	CU-U07	CA-U07 (persistencia y recuperación)
U.8 Exportar/compartir	CU-U08	CA-U08 (estructura CSV/JSON, enlace)
U.9 Autenticación	CU-U01	CA-U01 (login/errores/recuperación)
A.1 Ingesta de datos	CA-A01	CA-A01 (validación, normalización y rollback en error)
A.3 Verificación de integridad	CA-A02	CA-A03 (sin claves huérfanas; jugadores en equipos/temporadas correctos)
S.1 API de autenticación	CU-U01, CA-A03	CA-U01 (sesión segura), CA-A06 (autorización admin)
S.2 Roles (usuario/admin)	CU-U01, CA-A03	CA-U01 (solo admin funciones admin), CA-U06 (diferenciación usuario/admin)
S.3 Modelar y almacenar eventos	CU-U02, CU-U04, CA-A01, CA-A03	CA-U02 (patrón definido sobre eventos reales), CA-U05 (detalle de jugadas), CA-A01 (datos insertados correctamente), CA-A03 (integridad referencial)
S.4 Motor secuencial	CU-U03, CU-U04, CU-U05	CA-U03 (tiempos), CA-U04 (re-ejecución), CA-U05 (secuencia coherente)
S.5 Validar formato	CA-A01, CA-A03	CA-A01 (rechazo de JSON inválidos), CA-A03 (integridad tras importación)
S.6 Entrega completa / re-ejecución correcta	CU-U03, CU-U05	CA-U03 (todos los resultados cumplen patrón y filtros), CA-U04 (no se reutilizan resultados al cambiar parámetros)
S.7 CRUD patrones	CU-U07	CA-U07 (guardar y recuperar patrones)
S.8 Exportación	CU-U08	CA-U08 (exportación de resultados)
RNF.1 Usabilidad	CU-U02, CU-U04, CU-U05, CU-U07	CA-U02 , CA-U04 (refinado), CA-U05, CA-U07 (gestión de patrones)
RNF.2 Rendimiento en consultas	CU-U03, CU-U05	CA-U03 (tiempos de respuesta), CA-U06 (sin esperas)
RNF.3 Rendimiento y escalabilidad BD	CA-A01, CA-A02	CA-A01 (ingesta eficiente), CA-A02 (mantienen tiempos)
RNF.4 Portabilidad	—	CA-Q01 (funciona en Chrome y Firefox sin instalaciones)
RNF.5 Mantenibilidad	—	CA-Q02 (API documentada)
RNF.6 Seguridad	CU-U01, CA-A01, CA-A03, CA-A06	CA-U01 (HTTPS), CA-A01 (validación de importación), CA-A03 (integridad), CA-A06 (control de acceso), CA-Q03
RNF.7 Disponibilidad y recuperación	CU-U07, CA-A05	CA-U07 (patrones disponibles tras incidencias), CA-A05
RNF.8 Observabilidad y trazabilidad	CA-A01, CA-A02, CA-A04	CA-A04 (*logs* operaciones), CA-A01/A02 (traza ingestas)

3.3 Arquitectura global

En este apartado se describe la arquitectura del sistema. La solución elegida para el proyecto trata de un modelo **cliente-servidor** organizado por un **patrón por capas**. Se ha elegido este patrón debido a que facilita el desarrollo, la gestión y el mantenimiento del sistema software. El objetivo de la solución es permitir al usuario diseñar jugadas sobre un campo, ejecutar búsquedas de secuencias de eventos y analizar el conjunto devuelto mediante estadísticas del patrón y de jugadores específicos. Todo ello, manteniendo tiempos de respuesta e integridad de datos.

3.3.1 Vista lógica por capas

La arquitectura software se organiza en cuatro capas y un conjunto de procesos de operación:

1. **Capa de presentación (UI)**. Recoge la intención del usuario (diseño del patrón, filtros...) y muestra tanto la lista de jugadas encontradas, las estadísticas del patrón (jugadas encontradas, equipos involucrados, tiempos medio entre acciones, etc.) y los detalles de cada una de las jugadas (participantes, secuencia ordenada, porcentaje tiro a puerta/gol, etc.). La interfaz es la responsable de no mezclar parámetros entre diferentes consultas y de forzar la reejecución cuando los parámetros cambian.
2. **Servicios de aplicación (API)**. Esta capa se encarga de orquestar la operación, valida los parámetros de entrada, interactúa con el motor de búsqueda, compone respuestas homogéneas y estandariza los errores.
3. **Dominio (motor de búsqueda)**. Implementa el intérprete del patrón y el buscador de secuencias de eventos que preserva el orden temporal y aplica tolerancias. También calcula métricas del patrón. Esta capa garantiza el orden coherente de los eventos.
4. **Persistencia**. Almacena toda la información sobre competiciones, partidos, eventos y todos los metadatos relacionados. Mantiene índices para las dimensiones claves y preserva la integridad referencial.
5. ***Back-office* (operación)**. Proceso de ingesta idempotente de ficheros, reconstrucción de índices, verificación de integridad y restauración. No forma parte del flujo del usuario, pero cumplen una función muy importante para la calidad y disponibilidad de datos.

La estructura global del sistema, organizada en estas capas lógicas, se resume en la Figura 3.2, que muestra la comunicación entre la interfaz de usuario, la API, el motor de búsqueda y la capa de persistencia.

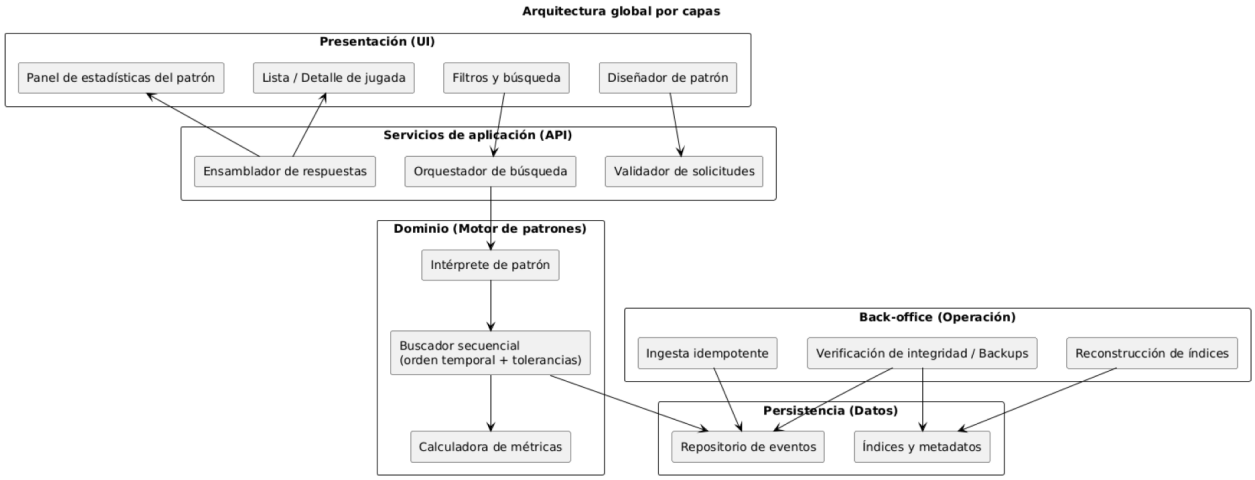


Figura 3.2. *Arquitectura global por capas*

3.3.2 Escenarios clave

Ingesta y mantenimiento

El administrador incorpora nuevos archivos y el *back-office* valida el formato, verifica la idempotencia, reconstruye los índices y comprueba la integridad de los datos. En caso de alguna incidencia, el administrador ejecuta la restauración para devolver la base de datos y sus índices a un estado previo consistente. Esto asegura rendimiento de la aplicación y calidad de los datos.

Consulta de patrón (Flujo principal del usuario)

El usuario define un patrón de jugada como una secuencia ordenada de eventos sobre un campo de fútbol y establece los filtros (competición, equipo, jugador, *outcomes*, etc.), junto con las tolerancias espacio-temporales. La UI construye la solicitud y genera un identificador de consulta derivado de todos los parámetros de la búsqueda. La capa de servicios (API) valida la petición y delega al motor de dominio, que interpreta la secuencia recibida y encuentra todas las coincidencias que cumplan las condiciones y preserven el orden y lugar de los eventos. La respuesta devuelve el conjunto completo de coincidencias. Con esos datos la UI presenta la lista, permite abrir el detalle de cada jugada (participantes, porcentajes de tiro/gol, etc.) y muestra las estadísticas del patrón calculadas sobre el conjunto devuelto.

El flujo completo de diseño de un patrón, aplicación de filtros y ejecución de la búsqueda puede verse de forma resumida en la Figura 3.3, donde se ilustra cómo interactúa el usuario con la aplicación.

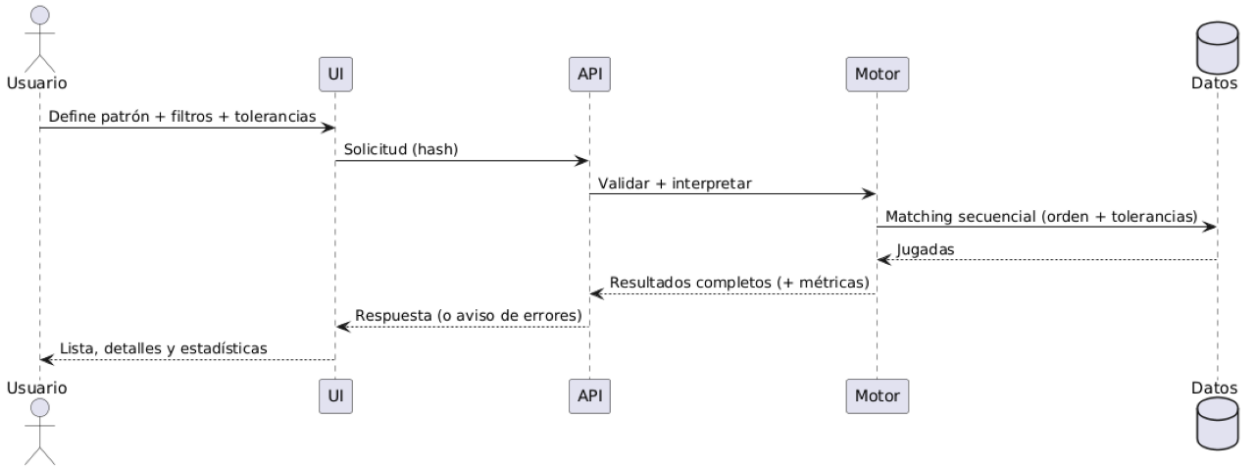


Figura 3.3. *Consulta de patrón*

Refinado y re-ejecución

Una vez obtenida la lista de resultados, el usuario puede ajustar filtros o parámetros de la búsqueda. Esto provoca que la UI invalide el identificador de consulta anterior y recomponga la solicitud. Que la API valide la nueva petición, y vuelva a delegar en el dominio, que de nuevo encuentra todas las coincidencias que se ajusten únicamente a los nuevos criterios. El objetivo de esta operación es garantizar que los resultados obtenidos reflejen los parámetros vigentes sin arrastrar ningún resultado de la operación anterior.

3.3.3 Diseño del servidor (*Backend*)

Este apartado describe la estructura lógica de negocio del servidor, centrada en los contratos funcionales que definen cómo se comunican los componentes, el flujo de ejecución y las políticas que garantizan coherencia y rendimiento en el sistema.

Contratos y responsabilidades

La capa de servicios actúa como enlace entre el cliente y el dominio, a través de una serie de operaciones que engloban la lógica de negocio, atendiendo las peticiones del usuario y devolviendo resultados consistentes. Cada contrato establece qué datos espera recibir, qué procesamiento va a realizar internamente y qué información va a devolver, garantizando que las interacciones sean trazables. La API ofrece una interfaz intuitiva y estable para la aplicación cliente.

A continuación, se detallan las principales operaciones del sistema:

- **Búsqueda de secuencias** (`/buscar`). Es el principal flujo de interacción con el usuario. El servidor recibe un patrón de jugada como secuencia ordenada de eventos, filtros y tolerancias espacio-temporales, y ejecuta el proceso completo de búsqueda. La operación valida los parámetros de entrada, interpreta el patrón y busca coincidencias en la base de datos identificando todas las secuencias de eventos que respetan el orden, las condiciones y las relaciones definidas por el usuario.

  El servicio devuelve una respuesta completa que incluye: Un **resumen global** (*summary*) con el número total de secuencias encontradas, el tiempo medio entre eventos y el número de equipos encontrados que han realizado ese patrón. Un ***ranking* por rol** (*ranking*) que mide la participación de los jugadores diferenciados por su función en la jugada (tirador, asistente, pasador previo, recuperador, etc.). Una selección de **ejemplos** de secuencias reales. Un **identificador único de consulta** (`query_id`) que permite recuperar posteriormente estadísticas adicionales sin reejecutar la búsqueda completa.

  Esta operación es idempotente, reproducible y eficiente. Cada combinación de parámetros genera un identificador único, facilitando la trazabilidad de las búsquedas y la reutilización de resultados en consultas posteriores.

- **Estadísticas por jugador y rol** (`/player-insights`). A partir del identificador de consulta (`query_id`) generado en la búsqueda principal, esta operación accede únicamente a las secuencias de dicha consulta. Sobre ese subconjunto, calcula métricas agregadas y distribuciones de participación por rol: quiénes fueron los principales tiradores, asistentes, pasadores previos, recuperadores o porteros involucrados. Ayuda al usuario a profundizar en el rendimiento individual dentro del contexto del patrón definido, sin necesidad de volver a ejecutar el motor de búsqueda completo. Además, también ofrece ejemplos de jugadas en las que cada jugador participó, permitiendo contrastar los datos con evidencias reales.

- **Catálogos y soporte**(`/outcomes`, `/competitions`, `/seasons`, `/teams`, `/players`). Estos servicios proporcionan información para alimentar los menús y filtros de la interfaz. Permiten validar los parámetros introducidos por el usuario y ofrecer una experiencia más fluida. Además, sirven para interpretar correctamente los valores de resultado (*outcome*) de cada tipo de evento, garantizando que las consultas sean válidas.

Flujo y políticas de ejecución

El procesamiento de una búsqueda sigue un flujo compuesto por tres fases principales, diseñado para garantizar la coherencia de los resultados, la eficiencia en la ejecución y la trazabilidad de cada operación.

1. **Preprocesado del patrón.** El sistema interpreta la secuencia de la jugada enviada por el cliente, expandiendo atajos a sus equivalentes (por ejemplo, *Recovery* se descompone en acciones que implican recuperar el balón como *Ball Recovery*, *Interception* o *Duel* exitosos con cambio de posesión). También se aplican las tolerancias espacio-temporales, las restricciones por zona del campo y las combinaciones de acciones predefinidas (como la secuencia *Pass-Shot*). El objetivo es normalizar la estructura del patrón antes de su evaluación.
2. **Ejecución del motor.** Una vez iniciada la búsqueda del patrón, el motor de búsqueda busca las coincidencias en la base de datos aplicando los filtros seleccionados (competición, temporada, equipo, jugador, tipo de evento, *outcomes*, etc.), manteniendo el orden temporal de los eventos, la posición y el contexto. Este proceso identifica todas las secuencias que cumplen la secuencia definida, garantizando que cada coincidencia respete el orden espacio-temporal y la lógica de la posesión entre las acciones. La búsqueda está optimizada, minimiza accesos redundantes y aprovecha los índices de la base de datos.
3. **Composición de respuesta.** El servidor construye una respuesta completa y útil englobando los resultados obtenidos. Esta incluye un **resumen global** con el número total de secuencias encontradas, tiempo medio entre acciones y el número de equipos o partidos en los que aparece el patrón. Un ***ranking* por rol** que muestra la participación de los jugadores según su función (tirador, asistente, pasador previo, recuperador, etc.). Un conjunto de **repeticiones** que permiten identificar a los conjuntos de jugadores que más ocurrencias tienen del patrón buscado y un **identificador de consulta** (`query_id`) que asocia los resultados con el contexto de búsqueda, facilitando la obtención posterior de estadísticas complementarias sin reejecutar todo el proceso.

Además, para mejorar la experiencia, cada consulta genera un identificador de consulta asociado al resultado en memoria. Esto permite recuperar las estadísticas adicionales sin reejecutar la búsqueda completa, manteniendo el servidor sin memoria a corto plazo y escalable. Para el control de carga, se limitan el número de consultas en caché y se encapsulan las respuestas (sólo máximos participantes). La observabilidad se trata mediante trazas por identificador de consulta y contadores que ayudan a diagnosticar tiempos extraños o cuellos de botella.

3.3.4 Diseño del cliente (*Frontend*)

El cliente forma parte de la capa de presentación y actúa como intermediario entre el usuario y la lógica de negocio del servidor. El objetivo es darle al usuario una interfaz en la que las acciones principales, como dibujar una jugada, aplicar filtros o explorar resultados, se muestren de forma clara y sencilla. El diseño del cliente también tiene la responsabilidad de mantener la coherencia entre búsquedas, validar los parámetros y representar los resultados de una manera comprensible.

Contratos de interacción

El diseño del cliente se forma alrededor de tres responsabilidades:

- **Expresar intención.** El usuario diseña el patrón como secuencia de eventos (zona o coordenadas, tolerancias, equipo/jugador por paso, *outcomes*, *switch_possesion*, *optional*, *goal/success*) y fija filtros globales (competición, temporada, equipo, jugador).
- **Comprender el resultado.** Se muestran resumen, lista de jugadas, *ranking* por rol y patrones repetidos con métricas. El usuario puede abrir un panel lateral con *insights* de jugador en el contexto de la última búsqueda.
- **Refinar y reejecutar.** Cambios en parámetros invalidan la consulta anterior. Entonces, la UI recrea la petición y actualiza el estado. Se conserva el `query_id`.

Comportamientos clave de la interfaz

El comportamiento de la UI se apoya en varios módulos:

- **Diseñador de jugadas en campo.** A través de un campo virtual, se pueden definir eventos en zonas predefinidas o posiciones sobre el campo con un simple clic. Al definir varios eventos, se trazan líneas entre los eventos seguidos unos a otros, distintos iconos para cada tipo de eventos y un resaltado para saber adecuadamente a qué zona del campo te estás refiriendo. Además de una ayuda visual (una flecha) para saber en qué lado del campo se ataca.
- **Catálogos de contexto.** La UI consulta outcomes por evento (con *fallback* local si es necesario). Y carga lista de competiciones, equipos y de jugadores condicionados por equipo/temporada para acotar la selección del usuario en cada paso.
- **Resultados.** Tras la búsqueda, el usuario dispone de *summary*, *ranking*, repeticiones (*repeats*) y ejemplos. El usuario puede abrir un apartado de *insights* por rol/jugador.

En conjunto, el diseño del cliente busca funcionalidad y simplicidad, permitiendo al usuario expresar consultas complejas mediante un proceso fácil en una interfaz visual directa e intuitiva.

3.4 Modelo de datos

3.4.1 Diagrama entidad-relación

La Figura 3.4 muestra el diagrama entidad–relación obtenido directamente de la base de datos a través de la aplicación *DBeaver* [2]. Este diagrama permite visualizar la estructura general del sistema y cómo se conectan entre sí las entidades principales y las tablas de apoyo utilizadas.

En la parte superior del diagrama se observa la relación entre `competitions` y `matches`. A partir de ahí, la tabla `events` aparece como el núcleo central del modelo, ya que forma la secuencia completa de eventos del encuentro y se enlaza con la información específica de cada tipo de evento (pases, disparos, regates, duelos, conducciones, etc.).

El diagrama también muestra las tablas derivadas (`player_match_team`, `player_team_season`, `team_comp_season`, `comp_season`), que permiten organizar de forma eficiente la relación entre jugadores, equipos, temporadas y competiciones sin la necesidad de definir tablas independientes para estas entidades. Estas tablas no aportan información nueva, sino que reorganizan la existente para agilizar consultas, filtros y la carga de datos en la interfaz.

Por último, aparecen las tablas opcionales asociadas a los datos *StatsBomb 360* (`three_sixty` y `freeze_frame`), que detallan la información espacial para algunos eventos. Aunque no se utilizan en esta versión del motor de búsqueda, su presencia en el esquema muestra que el modelo está preparado para futuras líneas de análisis más centradas en el posicionamiento y la estructura táctica.

En conjunto, el diagrama resume un modelo claro, compacto y orientado al análisis, donde la tabla `events` es el eje principal y el resto de entidades complementan la información de manera coherente y eficiente.

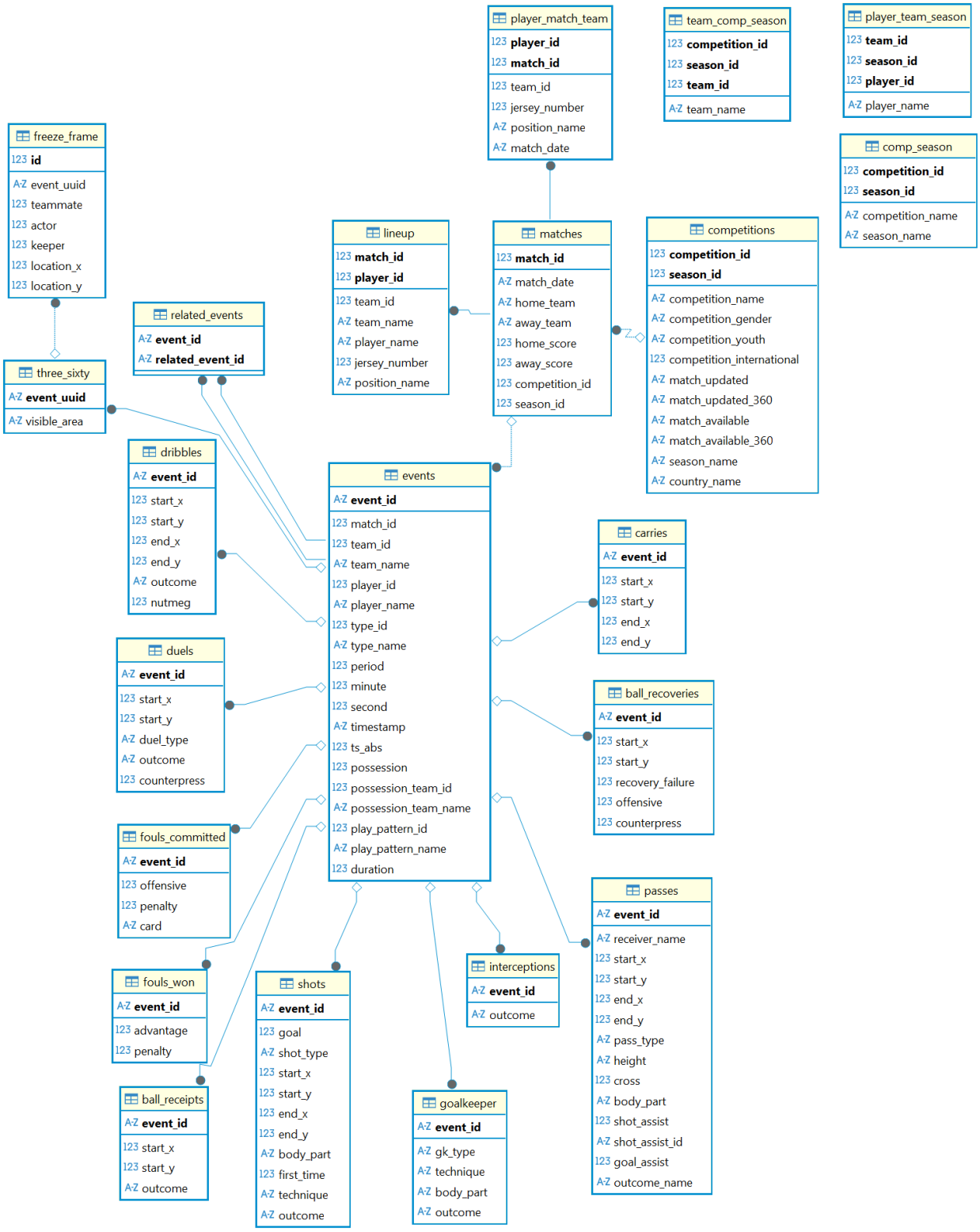


Figura 3.4. *Diagrama entidad–relación extraído mediante DBeaver.*

A partir de esta visión global, en los apartados siguientes se detalla la estructura del modelo, la jerarquía de tablas y el papel que desempeña cada una de ellas dentro del sistema.

3.4.2 Estructura general

La estructura general del sistema se basa en un esquema relacional diseñado para mantener la integridad de los datos, evitar redundancias y ofrecer un acceso eficiente a la información durante la ejecución de consultas y su posterior análisis.

El conjunto se almacena localmente en una base de datos SQLite [14], una implementación de un motor de datos relacional que aporta ligereza, portabilidad y rendimiento al sistema. Su estructura es genérica y portable, por lo que sería posible migrar fácilmente a gestores relacionales más robustos (como *PostgreSQL* o *MySQL*) sin necesidad de alterar las relaciones o las claves existentes.

Jerarquía principal.

La estructura se organiza en torno a las tablas principales que forman el núcleo del modelo y reflejan de forma directa los ficheros de *StatsBomb Open Data*:

- **Competitions:** Representa las diferentes competiciones incluidas en el sistema.

  Contiene los atributos para identificar la competición y contextualizar sus temporadas (`competition_id`, `competition_name`, `country_name` y `season_id`).

  Además de campos adicionales como fechas de actualización, indicadores de disponibilidad u otras informaciones sobre el tipo de competición que resultan útiles en partes del proyecto.

- **Matches:** Contiene la información de cada partido disputado almacenado en Stats Bomb Open Data.

  Incluye identificadores tanto del partido como de la competición y temporada, fecha y los nombres de los equipos.

  (`match_id`, `competition_id`, `season_id`, `match_date`, `home_team_id`, `away_team_id`).

  Se almacena también los marcadores y otra información contextual.

- **Lineup:** Almacena la alineación utilizada por cada equipo en cada partido.

  Incluye identificadores tanto del partido como del equipo y jugadores. Nombres de los equipos y jugadores

  (`match_id`, `team_id`, `player_id`, `team_name`, `player_name`).

  Además incluye información adicional de los jugadores y la posición que representan en el campo.

  (`jersey_number`, `position_name`

- **Events:** Es la tabla central del sistema y almacena cada acción del juego. Su diseño tiene dos objetivos principales: (i) representar cualquier acción del juego (pases, disparos, recuperaciones, conducciones, etc.), y (ii) proporcionar el contexto espacio–temporal para reconstruir la secuencia de un partido y ejecutar búsquedas de patrones. Para ello, cada registro incluye:

  Identificadores: (`event_id`, `match_id`, `team_id`, `team_name`, `player_id`, `player_name`.

  Contexto temporal: `period`, `minute`, `second`.

  Contexto espacial: `x`, `y`, `end_x`, `end_y` (posiciones registradas de los eventos).

  Y tipología y contextos: `type`, `subtype`, `play_pattern_name`)

  La estructura general permite que todos los eventos compartan una base común y, posteriormente, se amplíen mediante tablas específicas para capturar los atributos propios de cada tipo de acción.

La información específica de cada tipo de evento se almacena en tablas auxiliares, estas se enlazan con la tabla **Events** a través de **event_id**, esto permite almacenar atributos particulares sin saturar la estructura principal.

- **Pases (passes):** Contiene la información de un pase.
  - **outcome_name**: completado, fallo, bloqueo...
  - **receiver_name**: jugador que recibe el pase.
  - **height**, **cross**, **length**: detalles del pase.
  - **shot_assist**, **goal_assist**: indica si el pase genera una ocasión de disparo, o es asistencia de gol.
- **Disparos (shots):** Contiene la información de los remates realizados.
  - **goal**: si el disparo acabó en gol.
  - **outcome**: En qué resultó el disparo: tiro a puerta, fuera, bloqueado.
  - **body_part**: parte del cuerpo utilizada.

    Además de otras características cómo la técnica del disparo o si fue al primer toque.
- **Regates (dribbles):** Contiene la información de las acciones que realizan un desborde individual.
  - **outcome**: éxito o fallo.
  - **nutmeg**: si fue un regate entre las piernas, parámetro adicional.
- **Conducciones (carries):** Detalla las acciones en las que un jugador progresa con el balón controlado.
  - **start_x**, **start_y**
  - **end_x**, **end_y**
- **Duelos (duels):** Acción en la que se disputa la posesión del balón.
  - **outcome**: éxito o pérdida.
  - **duel_type**: tipo de duelo.
  - **counterpress**: si la acción se realiza cuando hay un apresión tras pérdida.
- **Recepciones de balón (ball_receipts):** Acción en la que un jugador controla un pase recibido.
  - **outcome**: si la acción ha resultado exitosa o fallida.
- **Intercepciones (interceptions):** Acciones que cortan el juego rival.
  - **outcome**: Completada, pérdida, acaba fuera...
- **Recuperaciones (ball_recoveries):** Acciones que cambian la posesión del balón.
  - **recovery_failure**: si falla la recuperación y se vuelve a perder la posesión.
  - **counterpress**, **offensive**: Si la recuperación viene de presión o en una acción en ofensiva.
- **Acciones de portero (goalkeeper):** Describe las intervenciones del guardameta.
  - **technique**: técnica empleada.
  - **outcome**: resultado de la intervención: gol, despeje fuera, balón controlado...
- **Faltas cometidas (fouls_committed) y recibidas (fouls_won):**
  - **card**: tarjeta mostrada
  - **advantage**, **penalty**: indicadores adicionales.

Tablas auxiliares.

Además de las entidades principales, el modelo incorpora varias tablas auxiliares cuyo objetivo es optimizar las consultas más frecuentes y evitar redundancias en la base de datos. Estas tablas actúan como estructuras de apoyo generadas o como catálogos derivados de los ficheros originales de StatsBomb.

- **player_match_teams:** Relaciona jugadores con los equipos y partidos en los que ha jugado. Se construye combinando `lineup` y `events`, y permite generar listados de jugadores filtrados por competición o temporada.
- **player_team_season:** Indica en qué equipo jugó cada jugador durante cada temporada. Esta tabla ayuda a la construcción de menús y filtros y evita fallos en jugadores con varios equipos a lo largo del año.
- **team_comp_season:** Relaciona los equipos con su competición por temporada. Se deriva de los eventos realmente presentes en el *dataset*, garantizando que solo se incluyan equipos participantes.
- **comp_season:** Catálogo que combina competición y temporada que permite construir selectores rápidos para la interfaz.
- **event_types:** Catálogo de tipos de evento detectados en el *dataset*. Facilita la construcción de filtros en la interfaz.
- **outcomes:** Tabla con los posibles resultados de cualquier evento. Por ejemplo: (*success*, *fail*, *blocked*, etc.). Se utiliza para filtros.

Datos adicionales

El proceso de ingesta también incorpora los datos de *StatsBomb 360*, datos que proporcionan información de posicionamiento de todos los jugadores sobre el campo en los instantes de cada evento. Aunque estas tablas no se utilizan en esta versión del motor de búsqueda, se mantienen en el modelo para permitir futuras líneas de trabajo. El sistema almacena estos datos en dos tablas:

- **three_sixty:** que contiene el identificador del evento (`event_uuid`) y el área visible en ese momento (`visible_area`).
- **freeze_frame:** registra los datos de la posición de cada jugador en el momento del evento, incluyendo para cada uno de los jugadores si es compañero, rival, portero o actor principal, junto con sus coordenadas.

3.4.3 Relaciones y cardinalidades

El modelo relacional del sistema sigue la jerarquía del fútbol (competiciones, partidos, equipos, jugadores y eventos) e incluye relaciones derivadas de estas para optimizar las búsquedas de patrones. Todas las entidades se conectan mediante claves primarias y foráneas que garantizan coherencia, trazabilidad y reconstrucción temporal de los partidos.

Relaciones principales (jerarquía del dominio)

- **Competition** $1 \rightarrow N$ **Season:** Donde cada competición puede contener varias temporadas.
  `competitions.competition_id → seasons.competition_id`.
- **Season** $1 \rightarrow N$ **Match:** Donde cada temporada contiene múltiples partidos.
  `seasons.season_id → matches.season_id`.
- **Match** $1 \rightarrow N$ **Event:** Donde cada partido contiene miles de eventos que describen todas las acciones ocurridas.
  `matches.match_id → events.match_id`.

Estas relaciones forman la columna vertebral del modelo, en la que se basa todo el sistema.

$$\texttt{Competition.(competition\_id, season\_id)} \rightarrow \texttt{matches.match\_id} \rightarrow \texttt{events.event\_id}$$

Relaciones entre eventos y sus tipos

La tabla **events** actúa como base común para cualquier acción del juego, esta almacena contexto del partido, el equipo, el jugador, el tiempo exacto y las coordenadas del evento. Pero cuando se necesita información específica de un tipo de acción, se recurre a tablas específicas que están enlazadas por **event_id**.

- En todos estos casos la relación es de 1 a 1, entre la tabla **event** y la tabla específica: (**events.event_id** ▯ **tabla_específica.event_id**).
- Donde las tablas específicas pueden ser cualquiera de todas estas, antes ya vistas. (**passes, shots, dribbles, carries, duels, interceptions, goalkeeper, fouls_committed, fouls_won, ball_receipts, ball_recoveries**)

Esto permite mantener **events** ligera, y acceder a las tablas de detalle solamente cuando se requiera. (Por ejemplo, filtrar por tiros a puerta o buscar los pases que asisten a un disparo).

Capas 360º y posicionamiento

La base de datos introduce información adicional de visibilidad y posicionamiento para un subconjunto de eventos. En nuestro modelo del sistema, esta estructura se representa mediante las tablas **three_sixty** y **freeze_frame**, ambas enlazadas con **events** a través del identificador del evento (**event_id**). La relación entre estas tablas las definimos de esta forma:

- **Events** $1 \rightarrow 0..1$ **Three_Sixty.** No todos los eventos del partido disponen de datos 360º. Pero cuando están presentes, existe solo un registro en **three_sixty** asociado al evento con **event_uuid**. Un evento puede no tener datos 360º o tener un único registro asociado.
- **Three_Sixty** $1 \rightarrow N$ **Freeze_Frame.** Cada registro de **three_sixty** tiene información sobre el evento (como el área visible), mientras que **freeze_frame** indica la posición de cada jugador en la jugada. Un evento con datos 360º genera varios registros en **freeze_frame** (uno por jugador observado), dando lugar a una relación de uno a muchos.

En conjunto, la estructura relacional de los datos 360 queda como:

$$\texttt{events}\ 1 \rightarrow 0..1\ \texttt{three\_sixty}\ 1 \rightarrow N\ \texttt{freeze\_frame}$$

Aunque la versión actual del motor de búsqueda no utiliza estos datos, su inclusión en el modelo da lugar a que en el futuro se puedan incorporar al sistema, el análisis táctico teniendo en cuenta la estructura posicional de cada equipo.

3.4.4 Normalización y optimización

Como el sistema debe ser capaz de ejecutar búsquedas complejas de secuencias de eventos, era necesario mantener un esquema normalizado para evitar inconsistencias, pero sin introducir soluciones costosas para no sobrecargar en cada consulta al motor. Esto dio lugar a un modelo, estructurado alrededor de la tabla principal **events** pero apoyado en tablas derivadas y catálogos permitiendo acceder rápidamente a la información más frecuente.

Desnormalización intencionada

El sistema incluye varias desnormalizaciones deliberadas cuyo objetivo es optimizar el rendimiento:

- **Tablas específicas para cada tipo de evento.** En lugar de una tabla `events` con cientos de columnas opcionales, los detalles de cada tipo de evento se especifican en tablas como `passes`, `shots`, `dribbles`, etc. Esto mantiene `events` ligera y mejora el rendimiento del motor de búsqueda.
- **Uso de tablas derivadas en lugar de una tabla de equipos o jugadores.** La información asociada a jugadores y equipos se construye desde `matches`, `lineup` y las tablas derivadas. Ya que mantener entidades independientes para jugadores o equipos no aporta ventajas en este proyecto y solo habría añadido complejidad innecesaria.
- **Duplicación controlada de nombres de jugador y equipo en `events`.** Dado que el motor necesita determinar el equipo y el jugador asociado a cada evento en tiempo real, mantener estos campos directamente en la tabla `events` evita uniones constantes y reduce significativamente el tiempo de consulta.

Todas estas desnormalizaciones se realizan con un objetivo: permitir búsquedas complejas de secuencias de eventos sin comprometer la eficiencia.

Optimización de consultas

Para garantizar un comportamiento relativamente rápido con una base de datos voluminosa, se añadieron optimizaciones orientadas al acceso rápido de información de la base de datos local, para ello se incorporarían índices a nuestra base de datos:

- **Índices en campos de filtrado habitual:** `match_id`, `player_id`, `team_id`, `type_name`, `outcome_name`. Estos índices reducen drásticamente el coste de las consultas del motor.
- **Índices compuestos temporales:** `(match_id, ts_abs)` permite recorrer un partido en orden sin operaciones adicionales.

4

Proceso de TL

En este capítulo se describe el proceso de tratamiento de los datos que da soporte al sistema. Partimos de los ficheros originales de *StatsBomb Open Data* y explicamos cómo se descargan, se verifican y se almacenan localmente. A continuación, se detallan los pasos de normalización y carga en el modelo relacional definido en el capítulo anterior, así como las tareas de mantenimiento y refresco de tablas derivadas. El objetivo es garantizar que la base de datos sobre la que trabaja el motor de búsqueda sea consistente, reproducible y lo bastante eficiente como para soportar consultas complejas sobre secuencias de eventos.

4.1 Ingesta, mantenimiento y validación de datos

El primer paso del desarrollo fue realizar un proceso sólido de obtención y carga de datos. El sistema utiliza datos de la fuente *StatsBomb Open Data*, un conjunto de datos de acceso público que contiene información detallada de competiciones, partidos, eventos y alineaciones. Además, incluye datos *tracking* con las posiciones de los jugadores en algunos partidos, aunque este último no esté incluido en el proyecto, puede resultar útil para futuras líneas de trabajo.

4.1.1 Fuente de datos: *StatsBomb Open Data*

StatsBomb Open Data proporciona los datos a través de un repositorio Github, donde alberga la información en formato JSON, organizada en distintos niveles:

- ***Competitions*:** contiene el listado de competiciones disponibles, con identificadores, nombres, país y tipo de torneo.
- ***Matches*:** agrupa los partidos asociados a cada competición y temporada, indicando fecha, equipos, estadio y árbitro.
- ***Lineups*:** Informa sobre las alineaciones de cada equipo en un partido, detallando los jugadores convocados, su posición, dorsal y rol inicial (titular o suplente).
- ***Events*:** incluye todos los eventos registrados en un partido, cada uno con su tipo (pase, disparo, recuperación, duelo, regate, etc.), coordenadas de inicio y fin, jugador implicado, equipo, resultado y momento exacto del juego.
- ***Three-sixty*:** guarda información extra de posicionamiento de los jugadores en el instante de cada evento (no utilizada en esta versión, pero útil para trabajos futuros).

Cada uno de estos niveles nos aporta un nivel de granularidad diferente, lo que convierte a StatsBomb en una fuente bastante completa para proyectos de análisis deportivo. Además, no se limita a estadísticas finales de partido, sino que registra cada acción individual con su contexto, permitiendo reconstruir el flujo del partido con precisión. Este enfoque nos permite desarrollar un sistema como el de este trabajo, capaz de detectar secuencias de jugadas y patrones de comportamiento a partir de datos reales.

Sin embargo, esta riqueza informativa también plantea ciertos desafíos. Debido a la cantidad de ficheros JSON y el volumen de estos archivos, que contienen tanto estructuras anidadas como variables diferentes dependiendo del tipo de evento, procesarlo directamente no sería eficiente para tareas analíticas. Dado este contexto, nos fue necesario diseñar un proceso de transformación que recogiera la información relevante y la pasara a un modelo relacional, sin perder detalle.

Dado que el volumen de datos es considerable, cada partido puede contener entre 2.000 y 4.000 eventos, lo que se traduce en varios cientos de miles de registros una vez procesadas todas las competiciones. Es por esto, que se optó por diseñar una estructura de almacenamiento local. Los ficheros descargados se almacenan en `data/` siguiendo esta estructura:

```
data/
    competitions.json
    matches/<competition_id>/<season_id>.json
    lineups/<match_id>.json
    events/<match_id>.json
    three-sixty/<match_id>.json
```

En conclusión, trabajar con *StatsBomb Open Data* nos ha permitido desarrollar una infraestructura de ingesta y validación de datos, además de sentar las bases tanto del presente como de futuras líneas del proyecto, especialmente en ámbitos relacionados con el posicionamiento de jugadores y el análisis táctico avanzado.

4.1.2 Descarga e integración automatizada

Una vez definida la estructura base de almacenamiento, el siguiente paso fue desarrollar la automatización de la descarga y organización de los datos. Con el objetivo de evitar tareas manuales, que implican la posibilidad de errores humanos, y para garantizar que el conjunto de datos estuviese sincronizado con el repositorio de *StatsBomb Open Data*. Se implementó un *script* en `Python` encargado de `descargar_datos.py`, la obtención de los ficheros desde el repositorio público de GitHub, gestionando su almacenamiento en local `data/`, validando la integridad de cada archivo descargado.

El proceso de descarga tiene el siguiente orden: primero obtiene la lista de competiciones, descarga los partidos asociados a cada una y, finalmente, descarga las alineaciones y los eventos correspondientes a cada partido. De esta forma se mantiene la coherencia entre los diferentes niveles de la base de datos y se evita la existencia de ficheros inconsistentes. El proceso completo de descarga y validación automática de los datos se resume en la Figura 4.1, donde se detallan las etapas desde la obtención de los ficheros JSON hasta su validación y almacenamiento local.

Además, el *script* está diseñado para ser flexible y reutilizable. Acepta diferentes argumentos que permiten personalizar el proceso según las necesidades de cada ejecución, como se muestra a continuación:

- `--data-dir <ruta>`: define el directorio donde se almacenarán los datos (por defecto `./data`).
- `--competition <id> <id>`: descarga una o varias competiciones específicas, utilizando los identificadores definidos en `competitions.json`.
- `--pair <comp:season> <comp:season>`: permite definir la descarga a una competición y temporada concreta.
- `--overwrite`: fuerza la reescritura de archivos ya existentes, útil en caso de descargas interrumpidas o actualizaciones del repositorio.

Ejemplos de uso real durante el desarrollo:

```
# Descarga completa del dataset en ./data
    python descargar_datos.py --data-dir ./data

# Descarga de dos competiciones (ejemplo: Liga y Premier)
    python descargar_datos.py --competition 11 12

# Descarga específica de temporadas en una competición
    python descargar_datos.py --pair 11:1 11:2

# Forzar reescritura de archivos dañados o incompletos
    python descargar_datos.py --overwrite
```

Durante la ejecución, el *script* realiza comprobaciones para garantizar que los ficheros sean correctos. Primero, verifica que el archivo exista y no esté vacío. Segundo, comprueba que el tamaño sea superior a un umbral mínimo esperado. Y por último, realizar una chequeo al JSON completo para asegurar que la estructura es válida. En caso de errores o interrupciones, el proceso los notifica en consola y marca los archivos afectados para su reintento en una ejecución posterior.

El sistema es idempotente, lo que significa que volver a ejecutar la descarga no genera duplicados ni sobrescribe datos válidos. Una vez completado el proceso, se genera un informe resumen que indica el número de competiciones, partidos y eventos descargados, así como los archivos ausentes o duplicados detectados. Todo el proceso esta reflejado en la Figura 4.1.

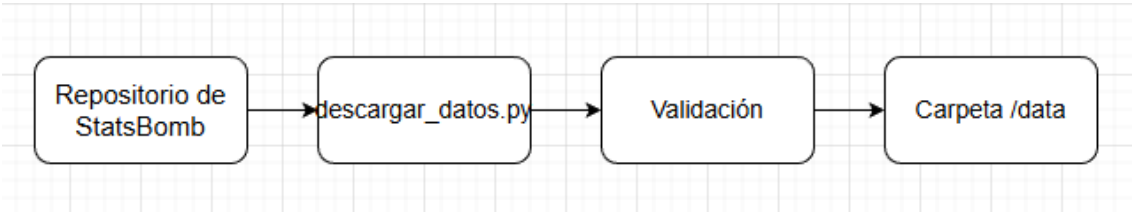


Figura 4.1. *Flujo general de descarga y validación automática de datos mediante* ***descargar_datos.py****.*

El resultado es un conjunto de datos en local completamente estructurado y validado, listo para su procesamiento. Cada ejecución del *script* garantiza que la carpeta **data/** mantenga una estructura coherente con el modelo lógico de la base de datos, siguiendo la siguiente estructura de almacenamiento:

```
data/
    competitions.json
    matches/<competition_id>/<season_id>.json
    lineups/<match_id>.json
    events/<match_id>.json
    three-sixty/<match_id>.json
```

Gracias a esta automatización, la preparación de datos se convierte en un proceso repetible, rápido y seguro. Esto permite centrarse en el desarrollo del sistema analítico sin preocuparse por inconsistencias en los datos de origen.

4.1.3 Carga estructurada y reconstrucción de índices

El siguiente paso al completar la fase de descarga y validación consistió en transformar los ficheros JSON en una base de datos relacional optimizada para la consulta. Por esto se desarrolló el *script* `database.py`, encargado de crear la base de datos local, definir la estructura de tablas y relaciones, y cargar los registros de forma ordenada y coherente.

El proceso parte de la carpeta `data/`, donde se encuentran los archivos descargados y validados por el *script* `descargar_datos.py`. Desde ahí, `database.py` recorre todos los archivos de la base de datos local, generando una nueva base de datos llamada `futbol.db`, que construye las tablas de soporte al sistema: *competitions, seasons, teams, players, matches, lineups* y *events*. Cada una de ellas corresponde con un nivel de información de la base de datos original de *StatsBomb Open Data*, adaptado al modelo relacional interno.

Diseño del proceso de carga.

Durante el desarrollo se definieron tres fases principales que garantizan la consistencia y el rendimiento del proceso:

1. **Normalización y limpieza inicial:** Cada archivo JSON es procesado y se extraen de él únicamente los campos necesarios para el análisis. Se eliminan valores nulos o duplicados y se asignan identificadores consistentes en toda la base. Se respetan los identificadores originales de StatsBomb para mantener la trazabilidad del dato.
2. **Validación referencial:** Antes de insertar un evento, se comprueba que los identificadores de jugador, equipo y partido existen en las tablas correspondientes. De esta forma se evita la inserción de registros inconsistentes. Las alineaciones (`lineups`) desempeñan un papel importante, ya que actúan como control para asegurar que los jugadores referenciados en los eventos pertenecen al encuentro en cuestión.
3. **Carga incremental e índices:** Una vez validados los datos, se realiza la inserción por lotes para mejorar el rendimiento y evitar bloqueos en SQLite. Tras completar de datos cada tabla, se crean los índices principales sobre los campos más utilizados por el motor de búsqueda: `match_id`, `team_id`, `player_id`, `type` y `outcome`. Estos índices reducen de forma significativa los tiempos de respuesta, sobre todo en consultas largas o utilizando filtros por jugador.

El proceso finaliza mostrando en consola un resumen con el número total de filas insertadas por tabla, el tiempo de inserción y el tamaño final de la base de datos. Además, el *script* permite reconstruir los índices en cualquier momento, lo que facilita el mantenimiento cuando se añaden nuevas competiciones o temporadas.

La Figura ilustra el flujo de carga, normalización e indexación de los datos en la base de datos relacional, mostrando cómo se transforman los JSON originales en tablas normalizadas con índices preparados para la búsqueda.

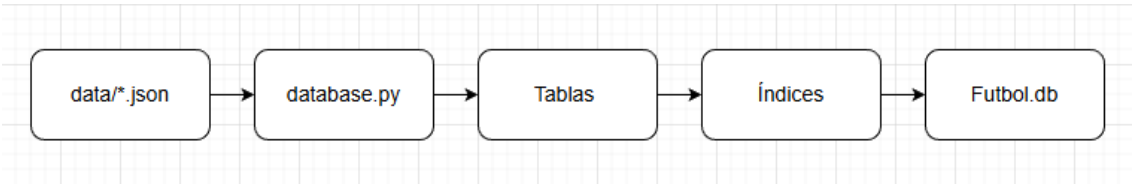


Figura 4.2. *Proceso de carga, normalización e indexación de la base de datos mediante* `database.py`.

El resultado final de esta fase es una base de datos local completamente estructurada, que combina integridad referencial con un buen rendimiento de consulta. Este modelo forma parte del núcleo del sistema analítico y es la base sobre la que opera el motor de búsqueda, los módulos de *ranking* e *insights* y las consultas avanzadas que veremos a continuación.

4.1.4 Verificación y controles de calidad

Terminada la fase anterior, carga estructurada y construcción de índices, el siguiente paso es establecer mecanismos para el control de calidad, para poder asegurar la consistencia de la base de datos. La información proviene de ficheros externos, estos controles son importantes para detectar a tiempo posibles errores de origen, prevenir duplicidades y garantizar los resultados obtenidos por el motor de búsqueda.

Comprobaciones estructurales.

El primer bloque de validaciones pone el foco en la estructura de los archivos descargados y en la correspondencia entre las diferentes capas de la base de datos. Durante la fase de descarga, el *script* `descargar_datos.py` verifica automáticamente que todos los ficheros existen y que no presentan errores de formato.

En paralelo a la descarga, se realiza una comprobación de correspondencias:

- Cada partido en `matches/` debe tener su correspondiente archivo de `events/` y de `lineups/`.
- Se valida que no haya identificadores duplicados en competiciones, equipos o jugadores.

Estos chequeos garantizan que la estructura de la base de datos sea íntegra y que las relaciones entre entidades se respeten incluso antes de la importación a la base de datos.

Validación referencial en la base de datos.

En la carga de la base de datos, mediante el uso del *script* (`database.py`), se realizan unas comprobaciones adicionales que aseguran la coherencia del modelo relacional:

- Se revisa que cada `player_id`, `team_id` y `match_id` presente en la tabla `events` exista previamente en sus respectivas tablas de referencia.
- Se bloquea la inserción de registros huérfanos mediante claves foráneas y comprobaciones previas de integridad.

Estos controles permiten detener la importación ante errores y mantener la trazabilidad de los problemas detectados.

Verificación de duplicados y coherencia temporal.

En el proceso de construir la base de datos se realizaron medidas para evitar duplicidades y garantizar la coherencia temporal de los eventos. Los identificadores naturales de cada entidad (como `competition_id`, `match_id`, `player_id` o `event_id`) actúan como claves primarias en la base de datos, lo que impide la inserción de registros repetidos y asegura que cada elemento se almacene una sola vez. Además, antes de insertar un nuevo registro, el *script* `database.py` verifica si la clave ya existe en la tabla correspondiente, omitiendo duplicaciones en caso de reconstrucciones.

En la inserción de eventos, para mantener la coherencia temporal, se aplicó un control que garantiza el orden cronológico, a través de la inserción de jugadas mediante los valores de `minute` y `second` del evento. Esto garantiza que las secuencias puedan reproducirse correctamente y que los algoritmos del motor de búsqueda operen sobre datos temporalmente consistentes.

Finalmente, en las pruebas realizadas no se detectaron duplicados significativos, y el sistema fue capaz de reimportar competiciones completas sin generar inconsistencias.

Revisión estadística y validación de contenido.

Aparte de los controles de estructura y referencialidad, se realizaron comprobaciones estadísticas para validar la calidad del contenido cargado. Estas pruebas se llevaron a cabo tras completar la ingesta de los datos y consistieron en comparar las cifras totales de cada tipo de entidad con los valores esperados según las competiciones descargadas.

Por ejemplo, se contrastaron los siguientes aspectos:

- Número total de partidos por competición frente al listado oficial de `matches.json`.
- Coincidencia entre los equipos y competiciones registrados y las relaciones establecidas en las tablas auxiliares.

Estas comprobaciones se ejecutaron directamente mediante consultas SQL, lo que permitió confirmar que los datos cargados mantenían una estructura sin pérdidas y coherente. Aunque el sistema no dispone aún de una verificación estadística automática, las pruebas aseguraron que los datos de los registros de origen se correspondían a los datos finales de la base de datos.

Monitorización del rendimiento y estabilidad.

En las fases de carga, se estudiaron los tiempos de ejecución de los principales *script*, y el comportamiento de la base de datos bajo diferentes volúmenes de información. Estos ensayos permitieron ajustar parámetros y optimizar el rendimiento general del sistema. En las primeras ejecuciones, el proceso de carga completo de todas las competiciones tardaba en torno a 45 minutos, mientras que las recargas parciales o las reconstrucciones de índices se completaban en menos de 5 minutos.

También hubo mejoras notables en la velocidad de las consultas una vez creados los índices en `match_id`, `team_id`, `player_id` y `type`, llegando a reducir el tiempo de respuesta hasta la mitad en búsquedas complejas. Esto confirma que el diseño de índices y la estructura relacional resultan adecuados para el volumen de datos manejado. Es más, las pruebas demostraron que la base de datos se mantiene íntegra aun cuando se realizan múltiples ejecuciones de carga, y que la memoria utilizada por los *scripts* se mantiene constante. Esto confirma que el sistema opera de forma estable.

Conclusión.

Gracias a estos controles, el proceso de carga asegura la integridad, la coherencia de los datos, y la fiabilidad del sistema a largo plazo. Este sistema resulta capaz de detectar errores, recuperarse de fallos en la descarga, y mantenerse íntegro tras actualizaciones de la base de datos. Estos controles son la base sobre la que operan el modelo de datos y los módulos desarrollados en los siguientes apartados.

5

Implementación de la solución

5.1 Implementación del *backend* (API y motor de búsqueda)

En este apartado describimos cómo hemos materializado el diseño de la lógica del sistema en una implementación del *backend*. Empezando por la arquitectura en capas, ya definida en el capítulo anterior, la trasladamos a una API REST en *Python*, que ejecuta el motor de búsqueda sobre nuestra base de datos con los datos de *StatsBomb Open Data* y calcula métricas para su explotación en el cliente. El objetivo de esta implementación es optimizar los tiempos de respuesta al usuario, mantener la integridad de los datos y facilitar la posible futura evolución del sistema.

5.1.1 Tecnologías y entorno de ejecución

El lenguaje al que optamos para la implementación del *backend* fue **Python**, ya que ofrece un sistema completo para el tratamiento de datos y la construcción de APIs web. Esto lo convierte en una opción adecuada para un proyecto de análisis de eventos futbolísticos, que necesitamos combinar consultas a base de datos y cálculo de estadísticas sobre estas consultas. Sobre Python utilizamos el *framework* FastAPI para la capa de servicios. Este *framework* nos permite definir los *endpoints* de la API REST, asociando a cada ruta una función de Python que implementa la lógica correspondiente. Además, FastApi viene con el sistema de modelos *Pydantic*, esto es importante, ya que utilizamos este sistema para definir las estructuras de entrada y salida, como la estructura que representa la consulta de búsqueda completa. Esto nos aporta validación automática de tipos y serialización de las respuestas, reduciendo código y facilitando la detección de errores.

Utilizamos un servidor ASGI, *Uvicorn*, que nos permite la comunicación asíncrona entre el servidor y FastAPI. En concreto, *Uvicorn* se encarga de ejecutar la aplicación y gestionar los eventos, atendiendo a las peticiones HTTP y delegando en FastAPI la gestión de cada ruta. Esta arquitectura nos permite manejar múltiples consultas al motor de búsqueda con poco consumo de recurso, manteniendo tiempos de respuesta adecuados, incluso cuando se trata de consultas complejas o que abarcan muchos partidos.

Respecto al almacenamiento, como sistema gestor de base de datos elegimos SQLite. Como trabajamos sobre un conjunto cerrado de datos procedentes de los ficheros JSON de *StatsBomb Open Data*. Una base de datos integrada directamente dentro del software es suficiente, además simplifica la instalación y el despliegue. El módulo de base de datos se encarga de crear las tablas e índices necesarios, así

como de cargar y transformar los datos a un modelo relacional optimizado para las búsquedas de secuencias de eventos.

Además de estos componentes, el *backend* utiliza librerías de Python para diferentes tareas, manejo de ficheros y JSON, gestión de fechas, tiempos y tipado. Tanto para la descarga inicial como para la preparación de datos, empleamos *scripts* que se ejecutan independientemente del servidor API, pero que comparten el mismo entorno de ejecución. Esto está organizado en una estructura de carpetas en el directorio `backend/` que agrupa el código de la API, el motor de búsqueda y los servicios de análisis, mientras que los datos procesados se almacenan en un fichero de base de datos (*futbol.db*) que es accesible por los distintos módulos del sistema.

5.1.2 Organización de módulos del backend

En la implementación del *backend* organizamos el código en varios módulos, siguiendo la arquitectura por capas definida en el diseño. Cada módulo tiene una responsabilidad y se comunica con el resto de módulos a través de interfaces, lo que facilita el mantenimiento.

En primer lugar, el módulo principal del *backend* es `main_api.py`. En este fichero definimos la aplicación de FastAPI [**FASTApi**], configuramos el *middleware* de CORS y definimos las estructuras de entrada y salida de datos a través de *Pydantic*. Además, `main_api.py` implementa los distintos *endpoints* de la API REST: el endpoint de búsqueda de secuencias, los endpoints de obtención de *rankings*, de *player insights* y otras rutas relacionadas con la obtención de opciones de filtrado (catálogos). Este módulo sirve como entrada a la capa de servicios, organizando las llamadas, tanto al motor de búsqueda como a los servicios de analítica, componiendo la respuesta final que consume el cliente web.

La gestión de almacenamiento de datos está en el módulo `database.py`. Definimos la conexión a la base de datos en SQLite, la creación de tablas e índices y el proceso de carga de los ficheros JSON de *StatsBomb Open Data* hacia un modelo relacional. Este módulo se encarga de transformar los datos de competiciones, partidos y eventos en tablas normalizadas, y de crear índices para optimizar las consultas frecuentes del motor de búsqueda.

Sobre esta capa construimos el módulo `motorbusqueda.py`, que implementa el motor de búsqueda de secuencias de eventos. Este módulo se encarga de la lógica del sistema, empezando por la traducción del patrón definido por el usuario a una consulta para la base de datos, siguiendo por aplicar el algoritmo de búsqueda secuencial que comprueba si una secuencia de eventos cumple las condiciones del patrón (tipo de evento, zonas del campo, equipo, jugador, resultados de la acción, cambios de posesión, márgenes temporales, etc.). Dentro de `motorbusqueda.py` también definimos los parámetros de las zonas del campo y las coordenadas sobre este, esto se utiliza en algunas partes del algoritmo.

Para mantener el código más claro, el motor de búsqueda se apoya en el módulo `helpers.py`. Este fichero se encarga de definir funciones auxiliares, como las normalizaciones de cadenas, detección de acciones exitosas y la identificación de cambios de posesión. Al establecer esto en un módulo aparte, garantizamos un criterio uniforme para interpretar los eventos, estableciendo este mismo criterio para futuras extensiones.

Una vez obtenidas las secuencias de jugadas devueltas por el motor de búsqueda, construimos sobre estas una capa de servicios de análisis, implementada en varios módulos. El módulo `summary.py` genera un resumen global de los resultados (número total de jugadas encontradas, distribución por partidos y equipos, tiempos medios entre eventos, etc.), mientras que `ranking.py` calcula la clasificación de jugadores según distintos roles (tiradores, asistentes, pasadores previos, recuperadores, regateadores o porteros) a partir de su participación en las jugadas que cumplen el patrón. El módulo `player_insights.py` va más allá de estos resultados ofreciendo información detallada y ejemplos de jugadas para un jugador y rol concretos, y el módulo `repeats.py` agrupa jugadas parecidas entre sí y las considera como un mismo patrón, identificando qué secuencias de eventos se repiten en distintos partidos. En definitiva, estos módulos forman la capa de servicios en la parte de análisis, cogen las jugadas encontradas por el motor de búsqueda y añaden resúmenes, *rankings* y estadísticas adicionales que ayudan a la interpretación de los resultados por parte del usuario.

Para mejorar el rendimiento, incorporamos un módulo de caché en memoria, `cache.py`. Este módulo implementa una estructura sencilla

que asocia un identificador de consulta **query_id** con la lista de jugadas obtenidas de esa búsqueda. Cuando el usuario solicita, por ejemplo, las estadísticas avanzadas de un jugador concreto en una jugada concreta o explora *rankings* derivados de una búsqueda ya realizada, la API puede recuperar las jugadas desde esta caché en lugar de reejecutar el motor de búsqueda, reduciendo así el tiempo de respuesta y la carga sobre la base de datos.

Por último, añadimos la parte *online* del *backend* con un módulo de preparación de datos, **descargar_datos.py**, que se ejecuta de forma independiente al servidor de la API. Este *script* descarga automáticamente los datos de *StatsBomb Open Data*, y se encarga de repetir la descarga si se produce algún fallo, y guardar los ficheros en la estructura de directorios definida para el proyecto. Además, verifica que se han obtenido todos los partidos y eventos necesarios antes de construir la base de datos. Resulta fundamental para que el *backend* trabaje siempre sobre un conjunto de datos coherente y actualizado.

En conclusión, esta organización en módulos permite alinearnos con la arquitectura por capas planteada anteriormente en el diseño. **Main_api.py** representa la capa de servicios, **motorbusqueda.py** y los módulos de análisis forman el núcleo de dominio, **database.py** representa la capa de persistencia y **descargar_datos.py** incorpora los procesos de carga inicial de datos. Esta separación de responsabilidades facilita encontrar errores y la posibilidad de incorporar nuevas líneas de desarrollo al proyecto sin introducir acoplamientos entre componentes.

5.1.3 Implementación de la API REST

En esta subsección describimos cómo implementamos la API REST que actúa como punto de entrada al *backend*. Esta API es la encargada de recibir las peticiones del cliente, validar los parámetros y llamar al motor de búsqueda de secuencias, y devolver una respuesta que el *frontend* pueda representar.

5.1.3.1 Modelos de datos y contratos

Para definir los contratos de la API, utilizamos *Pydantic*. El modelo principal de entrada es `SearchRequest`, que representa una consulta de búsqueda completa. Este modelo agrupa:

- La lista de pasos del patrón , donde cada paso se modela mediante la clase `EventFilter`.
- Los parámetros globales de búsqueda (tolerancia y margen_tiempo), que controlan el margen espacial y temporal permitido entre eventos consecutivos.
- Los filtros contextuales (`competition_id, season_id, team_id, player_id, match_id`), que nos permiten acotar la búsqueda a una competición, temporada, equipo o jugador concreto.

Cada `EventFilter` describe la serie de condiciones que debe cumplir cada evento dentro de la secuencia: tipo de evento, patrón de juego, zona o coordenadas de inicio, lista de resultados admitidos para el evento (`outcome / outcomes`), posibles cambios de posesión (`switch_possession`) e indicaciones sobre el éxito de la acción (`success, goal`). De esta forma, la estructura del patrón que envía el cliente encaja con la estructura que maneja el motor de búsqueda.

Por otro lado, respecto a la salida, el *endpoint* de búsqueda devuelve un objeto JSON que incluye, entre otros campos, el resumen de resultados, los *rankings* por rol, información adicional de los partidos implicados, las agrupaciones de jugadas repetidas y un identificador de consulta (`query_id`) que utilizamos para recuperar esas mismas jugadas desde la caché.

5.1.3.2 Endpoints principales

Los *endpoints* están definidos en `main_api.py`. El más sencillo es `(GET /status)`, que utilizamos como comprobación de la disponibilidad del *backend* y devuelve un objeto con un indicador de éxito.

El núcleo de la API lo forma el *endpoint* `POST /buscar`, que realiza el caso de uso de una ejecución de búsqueda de secuencias de eventos. Este *endpoint* recibe en el cuerpo de la petición un `SearchRequest`. Primero construimos una estructura de filtros a partir de los identificadores de competición, temporada, equipo, jugador o partido que pueda haber indicado el usuario.

A continuación, transformamos la lista de `EventFilter` en una representación más sencilla, un diccionario sin campos nulos. Y normalizamos algunos casos, si el cliente envía varios *outcomes* para un mismo evento se construye una lista con esos *outcomes*. Una vez tengamos la secuencia procesada, junto a los filtros contextuales y los parámetros espacio-temporales, se pasa al motor de búsqueda como entrada. La entrada acaba siendo una lista de jugadas, donde cada jugada es una secuencia de eventos con la información necesaria para su análisis.

Cuando ya tenemos las jugadas, el *endpoint* `(POST /buscar)` llama a los servicios de análisis, que construyen un resumen de los resultados, calculan los *rankings* de jugadores por rol y generan las agrupaciones de jugadas repetidas. Además, se genera un identificador de consulta (`query_id`) único. Este identificador se asocia a la lista completa de jugadas mediante la caché, de manera en que pueda reutilizarse en peticiones posteriores sin repetir la búsqueda. Y, finalmente, devolvemos al cliente un objeto JSON que integra todos lo anterior.

El segundo *endpoint* relevante es `(GET /player-insights)`, que permite ir más allá de los resultados de una búsqueda ya realizada. Este endpoint recibe como parámetros el rol a analizar, el identificador del jugador y el `query_id` de la búsqueda. A partir del `query_id` recuperamos las jugadas desde la caché, si no encontramos la consulta, devolvemos un error HTTP indicando que el identificador no es

válido. Una vez encontrada la consulta, sobre las jugadas almacenadas, filtramos aquellas en las que el jugador aparece con el rol solicitado y calculamos estadísticas específicas de ese rol (volumen de acciones, eficacia, contexto de las jugadas, etc.), acompañadas de un conjunto de ejemplos. La respuesta permite al cliente tener una vista detallada del rendimiento del jugador dentro del patrón analizado.

Completamos la API con varios *endpoints* auxiliares. El *endpoint* **`(GET /outcomes)`** devuelve, dado un tipo de evento, la lista de resultados (**`outcomes`**) distintos posibles que pueden tener. Para ello utilizamos un mapeo interno entre tipos de eventos y columnas, así el nombre del evento dado por el usuario nunca se inserta directamente en la consulta SQL, sino que pasa un proceso previo de traducción a valores controlados por el sistema.

Finalmente, los *endpoints* **`(GET /options/competitions), (GET /options/seasons), (GET /options/teams), (GET /options/matches) y (GET /options/players)`** proporcionan las listas de competiciones, temporadas, equipos, partidos y jugadores disponibles en la base de datos. Estos servicios se apoyan en el módulo **`database.py`** y sirven para rellenar los desplegables de filtros del cliente web, garantizando que el usuario solo pueda seleccionar valores que encajan con los datos realmente cargados.

5.1.3.3 Gestión de errores y CORS

Para permitir que el *frontend*, ejecutado en un origen distinto, **`http://localhost:5173`**, pueda consumir la API sin problemas de seguridad del navegador, configuramos en **`main_api.py`** un *middleware* de CORS sobre la aplicación de FastAPI. Esta configuración indica el origen permitido, las cabeceras y métodos aceptados, así como la posibilidad de enviar credenciales si en un futuro se incorpora autenticación.

En cuanto al tratamiento de errores, el *endpoint* **`POST /buscar`** tiene su lógica principal en un bloque *try/except*. Si se produjese alguna excepción inesperada durante la construcción de los filtros, la ejecución del motor de búsqueda o las fases posteriores, registramos el error en consola, facilitando el diagnóstico, y devolvemos al cliente un objeto JSON con un mensaje de error controlado. En el *endpoint* **`GET /player-insights,`** además de las excepciones **`HTTP`** propias de FastAPI, también tratamos cuando el **`query_id`** no se corresponde con ninguna búsqueda almacenada en la caché.

Con todo esto, la API REST del *backend* ofrece una interfaz clara y estable entre el cliente y el sistema, que encaja con los contratos definidos en el diseño y preparada en todo caso para evolucionar si se incorporan nuevos servicios de búsqueda o de análisis.

5.1.4 Motor de búsqueda de secuencias

El motor de búsqueda forma parte del núcleo del *backend*, ya que se encarga de determinar si en los datos de *StatsBomb Open Data* existen secuencias de eventos que cumplan con el patrón definido por el usuario. En esta subsección describimos cómo pasamos de un patrón a un conjunto de jugadas concretas, desde el preprocesamiento del patrón y la consulta a la base de datos hasta el algoritmo de búsqueda secuencial.

Preprocesamiento del patrón

Antes de buscar sobre la base de datos, el patrón recibido desde la API es transformado en una estructura más manejable. Esta tarea de preprocesamiento se centra en una función que recorre la lista de pasos **`EventFilter`** y aplica normalizaciones:

- **Limpieza y normalización de campos.** Eliminamos los campos nulos, unificamos nombres y nos aseguramos de que las opciones de equipo y de posesión (**`switch_possession`**) tengan siempre valores coherentes. Por ejemplo, si el usuario indica que el equipo debe ser "el mismo" para dos eventos consecutivos, registramos esa relación en la estructura interna.
- **Preparación de zonas y coordenadas.** Cuando el patrón es definido mediante zonas del campo, pasamos esas zonas del campo a rangos de coordenadas, con los que podremos trabajar en el algoritmo de búsqueda. Si el usuario define el patrón con coordenadas concretas, las mantenemos, pero le añadimos una tolerancia espacial configurable que permitirá aceptar pequeñas variaciones.

- **Expansión de tipos compuestos.** Algunos valores del tipo de evento, como por ejemplo una recuperación de balón genérica, se traducen a uno o varios tipos de eventos concretos de *StatsBomb Open Data* (recuperaciones, intercepciones, duelos, etc.), ajustando los campos de los resultados de la acción y los cambios de posesión para expresar correctamente el de la acción.

Durante este proceso también construimos una lista con los tipos de evento utilizados en el patrón. Esto nos sirve más adelante para limitar las tablas implicadas en la consulta SQL, evitando cargar datos innecesarios y reduciendo así la cantidad de eventos que el motor tiene que procesar en memoria.

Consulta a la base de datos

Con el patrón ya preprocesado, el motor de búsqueda realiza una consulta sobre la base de datos SQLite que devuelve solo los eventos candidatos a formar parte de las secuencias buscadas. El objetivo es que el algoritmo de búsqueda trabaje en memoria sobre un conjunto de eventos ya filtrado, ya sea por contexto (partidos relevantes) o por el tipo de acción (solo los tipos de evento que aparecen en el patrón). El proceso se puede dividir en varios pasos:

1. **Selección de partidos relevantes.** Se reduce el conjunto de partidos sobre los que vamos a realizar la búsqueda. Esta selección no se hace directamente en `motorbusqueda.py`, sino que viene preparada desde la interfaz, a partir de los filtros de competición, temporada, equipo, jugador o partido indicados por el usuario, construimos la lista de `match_ids` válidos. Para ello se consultan las tablas de competiciones, temporadas y partidos.

   De este modo, cuando el motor de búsqueda construye la consulta SQL dispone de una lista de partidos acotada si el usuario ha utilizado filtros por competición, temporada, partido o jugador. En caso de que el usuario no haya utilizado ningún filtro, el motor de búsqueda realizara la consulta sobre todos los partidos de la base de datos.

   Esta lista de partidos se traduce en una condición `WHERE e.match_id IN (...)` que acota desde el principio los eventos que la base de datos devuelve.
2. **Filtrado por tipos de evento del patrón**. Utilizamos la información extraída en el preprocesado del patrón, la lista de distintos tipos de evento que aparecen en el patrón del usuario. Estos tipos se usan en la condición `WHERE LOWER(e.type_name) IN (:tipos)`. Así, aunque la tabla principal de eventos contenga todo el histórico de acciones de los partidos seleccionados, la consulta acota aún más restringiendo a los eventos cuyo tipo coincide con los tipos implicados en el patrón.

   Este filtrado ayuda a reducir significativamente el número de filas que el motor tiene que procesar en memoria, especialmente cuando el patrón se centra en unos pocos tipos de evento.
3. **Combinación con tablas específicas de eventos.** La base de datos sigue la estructura de *StatsBomb Open Data* y separa cierta información específica en tablas auxiliares. Hay una tabla principal de eventos (con el orden temporal, los equipos, los jugadores y el tipo de cada acción) y las tablas auxiliares que amplian esos eventos con información detallada.

Además, la base de datos cuenta con índices específicos sobre columnas como `match_id, team_id, player_id` y tipo de evento, creados en el módulo de base de datos. Estos índices reducen el tiempo de ejecución de las consultas del motor, especialmente cuando el patrón es restrictivo y el número de eventos candidatos decrece de forma significativa.

Algoritmo de búsqueda secuencial

Con los eventos candidatos ya cargados en memoria, el motor continua ejecutando un algoritmo de búsqueda secuencial que recorre los eventos de cada partido en orden cronológico y busca secuencias que coincidan con el patrón definido por el usuario. La idea es comprobar si existe una secuencia de eventos que respete todo lo definido en el patrón. El funcionamiento puede dividirse en estas fases:

1. **Procesamiento partido a partido.** Antes que nada, agrupamos los eventos devueltos por la consulta según su identificador de partido `match_id`. Dentro de cada partido ordenamos los eventos temporalmente, teniendo en cuenta cuando se ha producido cada evento (periodo, minuto, segundo, etc.). Así, para cada partido tenemos una secuencia cronológica que representa ordenadamente lo que sucedió en el campo de fútbol, cosa que nos permite comprobar si aparece en él, el patrón definido por el usuario con el orden y las restricciones establecidas.
2. **Búsqueda de puntos de inicio**. Una vez tengamos la secuencia ordenada de un partido, recorremos los eventos buscando posibles puntos de inicio de una jugada que cumpla con el patrón definido. Para cada evento candidato, consideramos que ese puede ser el punto de partida del patrón, iniciamos una secuencia con ese evento e intentamos buscar el segundo evento definido en el patrón con los eventos posteriores a este en el partido. Es posible que el evento inicial no cumpla las condiciones del primer paso, entonces lo descartamos y seguimos buscando el punto de inicio.
3. **Comparación paso a paso con el patrón.** Para la construcción de una jugada válida, utilizamos una función que, dado un evento y un paso del patrón, confirma si ese evento satisface todas las condiciones especificadas en ese paso del patrón. Para ello, en cada paso se comprueba:
    - **El tipo de evento.** Comprobamos que el nombre del evento, coincide con el tipo de evento declarado en el patrón, o pertenece a alguno de los equivalentes definidos en el preprocesado, como por ejemplo una recuperación genérica.
    - **La zona o coordenadas.** Si el paso del patrón especifica una zona, verificamos que las coordenadas de inicio del evento están dentro del rango de esa zona, teniendo en cuenta la tolerancia espacial definida en el patrón. En cambio, si en el paso del patrón están especificadas unas coordenadas, se compara el punto del evento con el punto objetivo del patrón, sin dejar de tener en cuenta la tolerancia.
    - **El equipo y el jugador.** Comprobamos para cada evento se cumplan las reglas de posesión del equipo definida en el patrón, que puede ser la misma, contraria o cualquiera con respecto al evento anterior. Y también para cada uno de los eventos comprobamos que si en el patrón está definido un jugador concreto, coincida con el del evento actual.
    - **El resultado de la acción.** Comprobamos que el resultado del evento actual esté entre los resultados permitidos en el paso del patrón. Además, tenemos en cuenta si el patrón define acciones exitosas o fallidas y las comprobamos.
    - **Los cambios de posesión.** Si el paso especifica que debe haber un cambio de posesión, recuperación tras pérdida, comprobamos que la posesión del evento actual es distinta a la del equipo que tenía la posesión en el evento anterior, ya actualizamos para el sistema qué equipo tenemos como poseedor del balón.
    - **Las restricciones temporales.** Calculamos el tiempo entre el evento actual y el evento anterior de la secuencia encontrada y si ese tiempo supera el margen temporal permitido, dejamos de buscar sobre esa secuencia ya que consideramos que la jugada se ha roto.

    Si el evento cumple todas estas condiciones en el paso actual, lo añadimos a la secuencia parcial y avanzamos al siguiente paso del patrón. En caso contrario, lo descartamos y seguimos buscando a partir de los eventos posteriores.
4. **Construcción y registro de jugadas válidas.** A la vez que recorremos los eventos de un partido y vamos avanzando en el patrón, cuando hayamos conseguido una secuencia de cumpla todos los pasos del patrón, registramos esa secuencia como una jugada válida. Cada una de estas jugadas se guarda detalladamente, en ella se indica: identificadores de competición, temporada y partido, equipo que realiza la acción, jugadores que hayan participado y lugar, momento y resultado de cada evento. Una vez este registrada la jugada, el algoritmo sigue recorriendo la secuencia de eventos del partido buscando otras posibles coincidencias, manejando incluso secuencias que se solapan pero que cumplen el patrón.

Al finalizar este proceso para todos los partidos seleccionados, el motor devuelve una lista de jugadas válidas, donde cada elemento representa una instancia del patrón en un partido. Esta lista es la que después utilizaran los módulos de resumen, clasificación, "*player-insights*" y repeticiones, que calculan estadísticas y extraen información detallada a partir de las jugadas.

5.1.5 Servicios de análisis

Cuando el motor de búsqueda ya ha identificado todas las secuencias que cumplen el patrón definido por el usuario, transformamos esas secuencias en información fácil de interpretar. Para ello, construimos una capa de servicios de análisis, implementada en los módulos `summary.py, ranking.py, player_insights.py y repeats.py.` Esta capa toma como entrada la lista de jugadas devuelta por el motor y genera resúmenes, *rankings*, estadísticas individuales y patrones repetidos, que son los que se muestran en la interfaz final.

Resumen de resultados

En primer lugar `summary.py` construye un resumen global de los resultados de la búsqueda. A partir de la lista de jugadas recibidas, calculamos el número total de secuencias que cumplen el patrón, el número de equipos y partidos distintos en los que aparece y el tiempo medio transcurrido entre eventos dentro de las jugadas.

Para ello recorremos todas las jugadas devueltas por el motor de búsqueda y guardamos, los identificadores de equipos y partidos implicados, y por otro lado, los intervalos de tiempo entre eventos consecutivos dentro de cada secuencia. Con estos datos conseguimos saber cuántas veces se repite el patrón en la base de datos filtrada, cuántos equipos lo ejecutan y si se trata de una combinación muy rápida (pocos segundos entre acciones) o más lenta. Este resumen es parte de la respuesta del *endpoint* `/buscar` y se muestra en la interfaz como "Resumen del patrón", dando al usuario una visión global de los resultados.

Ranking por rol

A continuación implementamos el módulo `ranking.py`, que construye una clasificación de jugadores en función de su participación en las jugadas encontradas. El objetivo es identificar qué futbolistas aparecen con más frecuencia en cada rol del patrón. Cada uno de estos roles están definidos como tirador, asistente, pasador previo, recuperador, regateador o portero.

Para ello recorremos de nuevo todas las jugadas y, dentro de cada secuencia, analizamos los eventos de los que está compuesto. En función del tipo de evento y de su posición en la jugada asignamos roles a los jugadores implicados, por ejemplo, dada una secuencia "pase - tiro" consideramos tirador al que ejecuto el disparo y asistente al jugador que le dio el pase. En patrones que incluyen recuperaciones, marcamos como recuperador al jugador que gana de nuevo la posesión. Estas asignaciones se agregan en contadores por jugador y rol, así somos capaces de ordenar a los jugadores por número de participaciones en cada función.

El resultado se muestra como una clasificación, donde para cada rol listamos a los jugadores más destacados junto con su número de apariciones. Esta información se representa en la interfaz a través del panel "*Ranking* por rol" , desde el que el usuario puede identificar rápidamente qué futbolistas son más importantes en la ejecución del patrón analizado.

Estadísticas individuales

Para ir más allá de las estadísticas, queríamos profundizar sobre comportamiento de cualquier jugador dentro del patrón, para ello definimos un servicio, implementado en `player_insights.py` y expuesto a través del *endpoint* `/player-insights`. Este servicio no vuelve a ejecutar el motor de búsqueda, sino que parte del identificador de consulta (`query_id`) generado en la búsqueda principal. Esto hace que recuperamos desde la caché las jugadas y trabajamos sobre ese subconjunto de secuencias.

Sobre las jugadas recuperadas filtramos aquellas en las que el jugador indicado participa con el rol establecido (tirador, asistente, pasador previo, etc.). Calculamos métricas como el volumen de acciones, proporción de jugadas en las que interviene, número de goles o tiros a puerta generados, partidos distintos en los que aparece, entre otras. Además, seleccionamos un conjunto de ejemplos representativos, enriquecidos con datos necesarios para su visualización (partido, minuto, equipos, coordenadas de los eventos, descripción de la jugada).

El resultado devuelve tanto estas estadísticas agregadas como la lista de ejemplos, y se les muestra al en el panel lateral de *insights* individuales por jugador. Así, el usuario puede pasar de una visión global del patrón a un análisis detallado del rendimiento de un jugador concreto sin volver a ejecutar la búsqueda ni conocer la estructura de la base de datos.

Agrupación de repeticiones de patrones

Por último, el módulo `repeats.py` se encarga de detectar y agrupar patrones que se repiten entre las jugadas devueltas por el motor. La idea es identificar secuencias que comparten la misma estructura básica, por ejemplo, un mismo pase entre dos jugadores seguido de un disparo, o una combinación típica entre varios futbolistas de un equipo.

Para ello, para cada jugada generamos una firma simplificada que resume su estructura clave. Esta firma se construye a partir de información como el tipo de evento y los jugadores implicados en cada paso. Agrupamos las jugadas por firma, donde cada grupo representa un patrón que se repite, e incluimos tanto el número de ocurrencias como algunas métricas básicas (por ejemplo, cuántas de esas jugadas terminan en gol o en tiro a puerta). Para evitar respuestas grandes, limitamos el número de grupos devueltos y el número de jugadas por grupo, priorizando aquellos patrones que aparecen con mayor frecuencia.

Estos grupos se devuelven en la respuesta del *endpoint* `/buscar` en el bloque de "Repeticiones", que se le muestra al cliente en forma de lista de patrones recurrentes junto con sus ocurrencias. Esto ayuda al usuario a descubrir combinaciones de jugadores y patrones de juego que se repiten, aportando un análisis más allá del simple conteo de jugadas.

5.1.6 Caché y rendimiento

Para mejorar el rendimiento en los casos en el que el usuario realiza varias consultas sobre una misma búsqueda, incorporamos una caché en memoria, implementada en el módulo `cache.py`. Este módulo se encarga de asociar un identificador de consulta `query_id` con la lista de jugadas devueltas por el motor de búsqueda. Para ello, cada vez que el *endpoint* `/buscar` termina de ejecutar la búsqueda, genera un `query_id` aleatorio y almacena en caché las secuencias de eventos obtenidas. Esto provoca que cuando el usuario quiera acceder a estadísticas detalladas a través del *endpoint* `/player-insights`, recuperamos a través de la caché las jugadas y no hace falta ejecutar de nuevo la búsqueda.

5.1.7 Conclusiones sobre la implementación del *backend*

La implementación del *backend* incorpora la arquitectura por capas definida en el diseño y forma todo lo necesarios para ejecutar búsquedas de patrones sobre los datos de *StatsBomb Open Data*. A través del módulo principal `main_api.py` exponemos una API REST, basada en FastAPI y modelos de *Pydantic*, que realizan las operaciones clave del sistema: ejecución del motor de búsqueda, obtención de *rankings* y generación de *player_insights*.

En el núcleo del *backend* se encuentra el motor de búsqueda implementado en el *script* de Python `motorbusqueda.py`, que combina un preprocesamiento del patrón, consultas SQL para el filtrado sobre la base de datos SQLite y un algoritmo de búsqueda secuencial. Este motor localizaa secuencias de eventos que respetan las condiciones espaciales, temporales y de posesión definidas por el usuario.

Sobre las jugadas devueltas por el motor construimos una capa de servicios de análisis, implementada sobre los módulos `summary.py`, `ranking.py`, `player_insights.py` y `repeats.py`. Estos servicios transforman el conjunto de jugadas devueltas en resúmenes globales,

rankings por rol, estadísticas detalladas por jugador y patrones recurrentes. Por último, utilizamos una pequeña capa de caché en memoria (`cache.py`) que permite reutilizar los resultados de una búsqueda cuando se solicitan las estadísticas avanzadas sobre un jugador y rol. Además el *backend* ofrece *scripts* de descarga y preparación de datos que aseguran que siempre se trabaje sobre un conjunto de información coherente.

5.2 Implementación del *frontend* (interfaz de usuario)

En esta sección describimos cómo hemos realizado la capa de presentación del sistema con una aplicación web de una sola página. El objetivo del *frontend* es permitir al usuario formar secuencias de eventos de manera visual, configurar filtros de contexto sobre las competiciones y temporadas disponibles, realizar búsquedas en la API y visualizar los resultados a través de resúmenes, *rankings*, ejemplos de jugadas y paneles de *player-insights*. Esto se ha diseñado teniendo en cuenta los contratos definidos en el *backend*, manteniendo una correspondencia entre los modelos de datos de la API y los componentes de React.

5.2.1 Implementación del *frontend* interfaz de usuario

Para la implementación del *frontend* elegimos desarrollar una *Single Page Application (SPA)* utilizando React [9] y TypeScript [11]. React nos proporciona un modelo para construir la interfaz a partir de componentes reutilizables, mientras que TypeScript nos permite tipar las estructuras de datos de las comunicaciones con el *backend*, como los filtros de eventos, la consulta de búsqueda completa o los resúmenes y *rankings* devueltos. Esta combinación de tecnologías ayuda a mantener alineados los modelos del servidor con los del cliente. Además ayuda a reducir errores derivados de tipos incorrectos.

La aplicación se arranca desde el archivo `main.tsx`, donde creamos la raíz de React y renderizamos el componente principal `App` dentro del elemento `root` del documento HTML. También se importa la hoja de estilos `index.css`, que activa la configuración de Tailwind CSS para el proyecto. Tailwind permite aplicar estilos mediante clases directamente en el marcado JSX, esto nos ayuda a definir márgenes, colores de texto, o sombras, esto ayuda a que el diseño de la interfaz se vea limpio sin escribir grandes cantidades de código CSS.

El acceso a la API del *backend* se realiza en el módulo `api.ts`. En este fichero configuramos un cliente HTTP en Axios con una `baseURL` obtenida del entorno `import.meta.env.VITE_API_URL` y definimos funciones para cada contrato del *backend*: comprobación del estado del servidor (`apiStatus`), ejecución de búsquedas (buscar), obtención de `player-insights` (`getPlayerInsights`) y obtención de los catálogos para competiciones, temporadas, equipos, jugadores, partidos y outcomes.

Los tipos de datos que la interfaz maneja se declaran en el *script* `types.ts`. En este módulo definimos: el tipo `EventFilter` para representar cada paso del patrón, el tipo `SearchRequest` para modelar la consulta de búsqueda completa, el tipo `SearchResponse` para la respuesta del *endpoint* `/buscar` y la estructura de las respuestas de *player-insights*. También se define el tipo `Option`, utilizado en la interfaz para representar elementos seleccionables, como competiciones o temporadas, mediante un identificador numérico y una etiqueta legible. Utilizar estos tipos nos permite que el compilador nos avise cuando algún componente intenta acceder a campos que no existen.

El núcleo del *frontend* se forma en el componente `SearchPage`, la pantalla principal de la aplicación. Este componente maneja el estado de la búsqueda, lanza las peticiones al *backend* con las funciones definidas en el *script* `api.ts` y distribuye los datos recibidos en otros subcomponentes:

- `BarraFiltros`, implementa la selección en cascada de competición, temporada, equipo y jugador.
- `PlayDesigner`, permite construir el patrón de eventos sobre un campo de fútbol, para ello se apoya en el componente `Campo` para la representación gráfica.
- `InsightsDrawer`, un panel lateral deslizante en el que se muestran las estadísticas avanzadas de los jugadores cuando el usuario hace *click* sobre un jugador del *ranking*.

Además, utilizamos otros módulos: `Zonas.ts`, que define las zonas y segmentos del campo, `eventIcons.ts`, que para cada tipo de evento devuelve un icono gráfico y `OutcomesFallback.ts`, que proporciona listas de *outcomes* para cada tipo de evento, por si el *backend* no puede devolverlos. Estos módulos separan la lógica de presentación (componentes) de la configuración visual y los dominios (zonas del campo, iconos, resultados posibles), haciendo que la interfaz sea fácil de mantener y escalar.

5.2.2 Pantalla principal de búsqueda (`SearchPage`)

El núcleo del cliente es el componente `SearchPage`, la pantalla principal de la aplicación. Donde el usuario configura los filtros (competición, temporada, equipo, jugador), diseña la secuencia de eventos sobre el campo, lanza la búsqueda y visualiza los resultados, incluyendo el panel de *player-insights*.

El componente mantiene en su estado, tres bloques de información clave:

- **Estado de la consulta**. Lista de pasos del patrón, parámetros globales y filtros contextuales.
- **Estado de la respuesta** del *backend*. Objeto `SearchResponse` que contiene el resumen del patrón, el *ranking por rol*, las repeticiones de jugadas y una selección de ejemplos. Además almacena el `query_id` devuelto por la API, utilizado para solicitar `player-insights`.
- **Estado de la interfaz**. Indicadores de carga, posibles mensajes de error, disponibilidad del *backend* y la información para controlar el panel lateral de `player-insights`.

El envío de la búsqueda se realiza al ejecutar la función *submit*, que se dispara al pulsar el botón "Buscar" que envia el formulario. Esta función limpia cualquier error, activa el indicador de carga y construye el objeto `SearchRequest` a partir del estado actual de la consulta. Con este objeto llamamos a la función `/buscar` del módulo `api.ts`, que realiza la petición POST al *endpoint* `/buscar`. En caso de que la llamada sea exitosa, actualizamos el estado de respuesta con el `SearchResponse` devuelto y desactivamos el indicador de carga. En caso de error, capturamos el mensaje, lo mostramos en la interfaz y también marcamos el fin de la carga.

`SearchPage` forma la pantalla principal mediante cuatro zonas distintas. En la parte superior se sitúa la barra de filtros (`BarraFiltros`), que permite fijar el contexto de la consulta. En el centro se ubica el diseñador de jugadas (`PlayDesigner`), con la estructura para ajustar los parámetros y el botón de búsqueda. En la parte inferior se muestran los resultados: el resumen del patrón, el *ranking* por rol, las repeticiones de jugadas y la lista de ejemplos. Y por último en la parte derecha de la pantalla se abre el panel lateral `InsightsDrawer`, cuando el usuario selecciona un jugador en el *ranking*.

5.2.3 Barra de filtros (`BarraFiltros`)

El componente `BarraFiltros` implementa la selección en cascada de competición, temporada, equipo y jugador. Se encarga de ayudar al usuario a acotar el contexto antes de lanzar la búsqueda, reduciendo el volumen de datos sobre el que trabaja el motor.

`BarraFiltros` mantiene su estado con cuatro identificadores opcionales (`competition_id, season_id, team_id, player_id`). Estas opciones tienen el tipo `Option` definido en `types.ts`, que encapsula un identificador y una etiqueta . Al formar el componenete `BarraFiltros`, se llama a las funciones `getCompetitions, getSeasons, getTeams y getPlayers` del módulo `api.ts`. Donde `getCompetitions` carga el listado general de competiciones, `getSeasons(competition_id)` carga el listado de temporadas disponibles a partir de la competición seleccionada, y de forma similar se carga el listado de equipos y jugadores, se realiza en función de la competición y temporada seleccionadas, en el caso de los jugadores también equipo.

Cada vez que cambia uno de los filtros, el componente actualiza su estado y notifica al componente padre (`SearchPage`) los nuevos valores, de manera que la interfaz se mantiene actualizada. Para evitar inconsistencias, `BarraFiltros` incluye una función de ayuda (`ensureValid`) que comprueba si el valor seleccionado sigue existiendo en la lista de devuelta por la API. Por tanto, si un filtro deja de ser

válido porque ha cambiado el contexto (por ejemplo, al cambiar de competición desaparece un equipo), se reinicia automáticamente su valor. Con esto se evita que el usuario tenga seleccionados identificadores que no tienen sentido.

Visualmente, el componente se presenta como una fila de *selects* estilizados mediante Tailwind, presentando un orden lógico de izquierda a derecha: competición, temporada, equipo y por último jugador. Cada desplegable incluye una opción vacía que permite no fijar ese filtro y así poder analizar un contexto más amplio si resulta necesario.

5.2.4 Diseñador de jugadas (`PlayDesigner, Campo`)

El componente **`PlayDesigner`** forma el centro interactivo de la interfaz. Su función es permitir que el usuario defina el patrón de jugada como una secuencia de eventos en un campo virtual, especificando para cada evento: el tipo de acción (pase, tiro, regate, recuperación, etc.) su ubicación y parámetros adicionales (equipo, éxito, gol, cambio de posesión, resultados aceptados).

Para ello, **`PlayDesigner`** mantiene dos tipos de estado:

- **El patrón de la jugada**. Se recibe desde **`SearchPage`** o se construye como una lista de **`EventFilter`**. Cada elemento representa un paso del patrón e incluye para cada paso: el tipo de evento, las reglas de equipo, la zona o las coordenadas, la lista de outcomes seleccionados, los indicadores de éxito y las banderas como **`switch_possession`** u optional, si procede.
- **Estado de la edición en curso**. Recoge la configuración del "evento en curso" que el usuario está preparando antes de añadirlo al patrón. Incluyendo: el tipo de evento, la lista de *outcomes* seleccionados, la regla de equipo, el modo de posicionamiento (por zonas o por coordenadas) y, cuando procede, el jugador al que se quiere asociar ese paso concreto.

Cuando el usuario selecciona un tipo de evento, **`PlayDesigner`** consulta los posibles resultados para ese tipo mediante la función **`getOutcomes`** del módulo **`api.ts`**. Si la llamada no devuelve datos, el componente recurre a un listado local de respaldo definido en **`OutcomesFallback.ts`**. Este listado cubre los *outcomes* habituales por tipo de evento y garantiza que el usuario disponga de opciones, incluso si el servidor no puede proporcionarlas.

La selección y visualización de los eventos se realiza en el componente **`Campo`**. **`PlayDesigner`** le pasa a **`Campo`** la lista actual de pasos del patrón, el índice del paso seleccionado y un *callback* que se ejecuta cuando el usuario hace clic sobre cualquier zona del campo. A partir de ese clic, el componente **`Campo`** traduce las coordenadas de la interfaz a las coordenadas del campo definido de *StatsBomb Open data* y, en función del modo activo, determina la zona o las coordenadas que tiene que guardar. Además, el componente **`Campo`** también dibuja el campo de fútbol, los iconos de cada evento, las líneas que conectan eventos consecutivos y el resaltado de la zona bajo el ratón, esto ayuda al usuario a entender en qué zona del campo está definiendo el patrón.

Cuando el usuario está satisfecho con la configuración de un evento, **`PlayDesigner`** lo añade a la lista de pasos del patrón, la herramienta entonces definir el siguiente evento o seleccionar un paso ya definido para editarlo o eliminarlo. En todo momento la representación visual (iconos y líneas en Campo) se mantiene sincronizada con la estructura de datos que después se envía al *backend* como parte de la petición **`SearchRequest`**.

5.2.5 Panel de resultados e (`InsightsDrawer`)

Cuando el usuario ha definido el patrón y ha realizado la búsqueda, **`SearchPage`** recibe un objeto **`SearchResponse`** con la respuesta. **`SearchPage`** divide esta entre distintos componentes encargados de la visualización. En primer lugar, el resumen global del patrón se muestra en un bloque donde se indica el número total de jugadas encontradas, el número de equipos y partidos cubiertos y el tiempo medio entre acciones. A continuación, el *ranking* por rol se representa como una tabla, donde para cada rol se listan los jugadores junto con su número de intervenciones. Por último, las repeticiones se muestran como una lista de patrones recurrentes, cada uno acompañado de información sobre cuántas veces se repite y ejemplos representativos.

Cuando el usuario hace clic sobre un jugador dentro del *ranking*, El componente `SearchPage` actualiza el estado del panel lateral, guardando el rol y el identificador del jugador seleccionados y manteniendo el `query_id` de la búsqueda. Con esta información se actualiza el componente `InsightsDrawer`, que se abre como un panel deslizante desde el lateral derecho de la pantalla.

El componente `InsightsDrawer` recibe como propiedades el estado de apertura, el callback de cierre, el rol, el identificador del jugador y el `query_id`. Cada vez que se abre ejecuta una función que llama a `getPlayerInsights` en `api.ts`. Esta función envía una petición al *endpoint* `/player-insights` del *backend*, que utiliza el `query_id` para recuperar las jugadas de la última búsqueda desde la caché y calcula las estadísticas avanzadas para el jugador y rol indicados.

Cuando la respuesta llega correctamente, `InsightsDrawer` actualiza su estado interno con un objeto `PlayerInsightsResponse`. Este objeto se compone de el nombre del jugador, estadísticas asociadas (acciones realizadas, goles, tiros a puerta, partidos implicados, etc.) y una lista de ejemplos. El panel muestra estas estadísticas y, a continuación, lista los ejemplos los cuales llevan información sobre su minuto de juego, una breve descripción y, cuando está disponible, una búsqueda de *YouTube* que facilita localizar el vídeo del partido correspondiente.

Con esto, la implementación del *frontend* cumple con los contratos definidos en el diseño, permitiendo expresar la intención del usuario mediante un patrón visual, aplicar filtros contextuales, visualizar los resultados a través de resúmenes y *rankings*, y profundizar en el rendimiento individual de los jugadores dentro de las jugadas encontradas.

5.2.6 Conclusiones sobre la implementación del *frontend*

La implementación del cliente materializa la capa de presentación definida en el diseño y actúa como intermediario entre el usuario y los servicios del *backend*. A través del componente principal `SearchPage` manejamos el estado de la consulta, las llamadas a la API y la distribución de los resultados entre los distintos componentes,

Módulos como `api.ts` y `types.ts` se centran en el acceso al *backend* y la definición de tipos compartidos, mientras que componentes como `BarraFiltros`, `PlayDesigner`, `Campo` e `InsightsDrawer` se encargan del contexto de análisis, de la construcción visual del patrón, la representación sobre el campo y la visualización detallada de los resultados.

El usuario, recibe una barra de filtros en cascada, un diseñador de jugadas interactivo sobre el campo y los resultados divididos en distintos bloques como resumen global, *ranking* por rol y repeticiones, junto a las estadísticas avanzadas si clicas sobre cualquier jugador del *ranking*. Esto facilita que el usuario encuentre lo qué está buscando.

En conjunto, la implementación del *frontend* completa la arquitectura propuesta: una base de datos relacional y un motor de búsqueda especializado. Dando pie a una aplicación que ofrece una interfaz web interactiva que hace accesible al usuario la definición de patrones, la ejecución de búsquedas complejas y el análisis de resultados sin necesidad de interactuar directamente con consultas SQL o con los datos sin procesar de *StatsBomb Open Data*.

6

Validación y casos de uso

Una vez diseñada e implementada la solución, es necesario comprobar que el sistema cumple con los requisitos funcionales y no funcionales planteados en los capítulos anteriores. Para ello, hemos establecido una estrategia de validación que comprueba principalmente: la calidad de los datos, el funcionamiento del *backend* y la experiencia de que proporciona la interfaz de usuario.

En primer lugar, se ha validado la integridad y consistencia de los datos obtenidos a partir de *StatsBomb Open Data*. El proceso de descarga e inserción en la base de datos, implementado en los *scripts* de ingesta, asegura que:

- El número de competiciones, temporadas, partidos, equipos, jugadores y eventos coincide con el publicado por la fuente original.
- No se producen pérdidas de información durante la importación.
- Las claves primarias y foráneas evitan registros huérfanos o relaciones incoherentes entre tablas.

A continuación, se ha abordado la validación del *backend*. Probando que:

- El **motor de búsqueda de secuencias de eventos** se encarga de recorrer el conjunto de eventos y detectar las secuencias que cumplen las condiciones establecidas por el usuario.
- Los **servicios de análisis**, construidos sobre los resultados de la búsqueda, se encargan de la construcción de resúmenes de las jugadas devueltas, creación de *rankings* de jugadores por rol, elaboración de estadísticas avanzadas por jugador y rol, y de encontrar los patrones de jugadas repetidas.
- La **API** actúa como intermediaria entre la interfaz web y el motor de búsqueda, exponiendo una serie de *endpoints* que permiten lanzar búsquedas de patrones, recuperar los detalles de las jugadas y obtener estadísticas avanzadas, gestionando además los códigos de respuesta.

Por último, se ha validado interfaz de usuario. Comprobando, que las llamadas al *backend* se realizan correctamente y que la experiencia de uso es correcta. Para esto último, se comprueba que:

- Los filtros se actualizan en cascada y sólo permiten combinaciones válidas.
- El diseñador de jugadas permite construir patrones válidos y los representa correctamente sobre el campo.
- Las búsquedas responden en tiempos razonables para el volumen de datos considerado.
- La visualización de resultados es clara y comprensible para un analista.

Para comprobar todo lo descrito se han implementado diferentes tipos de pruebas:

- **Pruebas de caja blanca** sobre componentes internos del *backend*, verificando el comportamiento de funciones del motor de búsqueda y de los servicios de análisis con entradas controladas.
- **Pruebas de caja negra** sobre la API y la interfaz, comprobando que, dada una entrada, las respuestas obtenidas son resultados coherentes sin necesidad de conocer el detalle de la implementación.
- **Pruebas de integración** entre la base de datos, el motor de búsqueda, la API y el *frontend*, asegurando que desde la definición de un patrón hasta la visualización de resultados se realiza sin errores.
- **Pruebas de aceptación** que simulan consultas reales que podría realizar un analista táctico como patrones de centros laterales o combinaciones de pase y tiro en zonas del campo.

Además, se han definido **criterios de aceptación** para cada caso de prueba:

- Se considera que un test es satisfactorio cuando el comportamiento observado del sistema coincide con el comportamiento definido en sus criterios de aceptación (entradas, salidas, mensajes de error, tiempos de respuesta, etc.).
- En el caso concreto de las pruebas sobre el motor de búsqueda, un test es satisfactorio cuando todos los resultados devueltos cumplen las restricciones impuestas por el patrón y los filtros configurados.
- Las métricas derivadas de los resultados (*rankings*, porcentajes de éxito, etc.) son coherentes con los datos y pueden reproducirse a partir de una muestra de jugadas inspeccionada manualmente.
- No se producen errores durante la ejecución y, en caso de fallo, se proporcionan mensajes comprensibles y adecuados al usuario.

Todas las validaciones se han realizado utilizando datos reales obtenidos del conjunto de datos de *StatsBomb Open Data*, esto significa que las pruebas se han realizado sobre contextos reales y no sobre conjuntos simplificados. La combinación de estos niveles y tipos de prueba permite obtener una visión global del comportamiento del sistema y detectar posibles errores en las distintas partes de la implementación.

Las pruebas comentadas en este apartado están disponibles en el repositorio de GitHub del proyecto, dentro de la carpeta `tests`. Tener este conjunto de pruebas bien montado hace que sea mucho más sencillo mantener y seguir desarrollando el sistema en el futuro, porque permite detectar rápido si algún cambio introduce errores.

6.1 Pruebas funcionales

6.1.1 Pruebas del *backend*

Las pruebas sobre *backend* comprueban que tanto el motor de búsqueda como los servicios de análisis realizan correctamente su tarea y la API expone estos servicios de forma consistente.

Para tratar el *endpoint* **de búsqueda de patrones**, se plantean estos casos de prueba:

- **Patrón simple de un único evento**. Se define un patrón formado solamente por un tiro, sin ninguna condición más, ni de zona, equipo o jugador. El criterio de aceptación consiste en que todas las jugadas devueltas tengan únicamente un evento de tipo tiro y que el número de jugadas coincida, con el número de tiros registrados en el conjunto de datos.
- **Patrón de pase seguido de tiro**. Se define un patrón formado por un pase seguido de un tiro, acotando la búsqueda a una competición y temporada concretas. La prueba comprueba que las secuencias devueltas respetan el orden de los eventos, que ambos pertenecen al mismo equipo y que todos los partidos están dentro del contexto especificado.
- **Patrón con restricciones espaciales**. Se construye un patrón que exige que el primer evento (por ejemplo, un pase) se inicie en una zona concreta del campo y que el segundo evento (por ejemplo, un tiro) tenga lugar en otra zona específica. El objetivo es verificar que las coordenadas de los eventos se encuentran dentro de los rangos asociados a cada zona y que no devuelven jugadas que no cumplen las condiciones espaciales.
- **Patrón filtrado por jugador**. Se selecciona un jugador concreto y se define un patrón en el que participe en uno de los eventos. El criterio de aceptación consiste en que todas las jugadas devueltas aparezca este jugador en el rol indicado y que no se incluyan acciones de otros jugadores en esa posición.
- **Patrón sin resultados**. Se define un patrón restrictivo (por ejemplo, combinación de eventos poco frecuente en una zona muy concreta del campo). En este caso, el *backend* debe devolver una respuesta sin jugadas, pero sin producir errores, y la API debe informar de esta situación de forma controlada.

Además de las pruebas para el motor de búsqueda, se realizan comprobaciones de los **servicios de análisis**:

- Para el servicio de **resumen de jugadas**, se comprueba que el total de jugadas y la distribución por partidos, equipos y tipos de evento coinciden con lo observado. También se revisa que los porcentajes y métricas agregadas se calculan correctamente.
- Para la **clasificación de jugadores por rol**, se verifica que el orden de los jugadores muestra el número de apariciones en el patrón, así como otros indicadores diseñados. En una muestra reducida de jugadas se comprobará el *ranking* para asegurar que los valores se corresponden con los datos.
- Para los ***player-insights***, se comprueba que, dado un identificador de consulta, sólo se consideran las jugadas asociadas a esa búsqueda y que las estadísticas mostradas pertenecen al jugador y al rol analizado.
- Para el servicio de **jugadas repetidas**, se verifica que las agrupaciones comparten la misma estructura de eventos y que se identifican los patrones recurrentes cuando un equipo repite una misma jugada en diferentes partidos o momentos del encuentro. En este caso se ha contrastado con vídeo las jugadas para confirmar que la agrupación tiene sentido táctico.

Durante estas pruebas se contemplan también situaciones de error y casos como patrones demasiado restrictivos que no devuelven ninguna jugada. En estos casos se ha comprobado que el *backend* responde con mensajes de error adecuados, que la API devuelve códigos de estado HTTP coherentes y que no se provocan fallos inesperados en la ejecución.

6.1.2 Pruebas del *frontend*

Las pruebas sobre la interfaz de usuario se han diseñado para validar los caminos principales que seguirá el usuario final al utilizar la herramienta: selección del contexto de análisis, diseño del patrón, ejecución de la búsqueda e interpretación de resultados.

Los principales casos de prueba realizados son los siguientes:

- **Carga inicial de filtros**. Al iniciar la aplicación, se comprueba que se cargan correctamente los listados de competiciones, temporadas y equipos, y que los desplegables muestran solo las combinaciones válidas.
- **Filtrado en cascada**.Si se selecciona una competición, una temporada y un equipo. Se comprueba que el resto de filtros se actualizan en cascada, y que en la lista de jugadores disponibles solo deben salir jugadores del equipo y temporada seleccionados. Además, se comprueba que cualquier cambio en un filtro anterior invalida o reajusta los filtros dependientes.
- **Construcción de un patrón sencillo**. Utilizando el diseñador de jugadas sobre el campo, se añade un pequeño patrón seleccionando los tipos de evento, las zonas y el jugador implicado. Se comprueba que la representación visual en el campo refleja correctamente las elecciones del usuario y que el patrón se realiza adecuadamente de cara a ser enviado al *backend*.
- **Edición y eliminación de pasos del patrón**. Se añaden varios pasos al patrón y, posteriormente, se modifica o elimina alguno de ellos. Verificamos que el estado del patrón se actualiza correctamente y que la representación gráfica se mantiene coherente.
- **Ejecución de una búsqueda**. Una vez definido el patrón, se lanza la búsqueda desde la página principal. La prueba comprueba que los resultados se presentan correctamente. Además, se comprueba que el usuario puede lanzar nuevas búsquedas sin necesidad de recargar la página.
- **Exploración de *player insights***. A partir de los resultados de una búsqueda, se selecciona un jugador en el *ranking* por rol. La prueba verifica que se abre el panel con las estadísticas específicas de ese jugador en el contexto del patrón definido y que las jugadas de ejemplo se corresponden con las devueltas en la búsqueda original. También se comprueba que el usuario puede volver a la vista principal de resultados.

Para el seguimiento de las pruebas, se ha elaborado la tabla 6.1 de casos de prueba funcionales que recoge los escenarios más representativos.

La mayor parte de las pruebas descritas se han realizado de forma **manual**, interactuando con la interfaz web o mediante los *scripts* de ingesta y mantenimiento del *backend*. No se ha desarrollado una batería de tests automatizados, pero cada caso de prueba está descrito con sus pasos y resultado esperado, de modo que cualquier persona puede repetirlos y comprobar si el sistema sigue cumpliendo los requisitos.

Tabla 6.1. Casos de prueba funcionales del sistema (backend y frontend)

ID	Descripción	Módulo	Resultado esperado	Resultado
PF-01	Búsqueda de patrón simple sin filtros.	*Backend*	Resultados devueltos solo tienen un tipo de evento.	OK
PF-02	Búsqueda de patrón pase-tiro, competición y temporada acotada.	*Backend*	Resultados devueltos tienen un pase seguido de un tiro del mismo equipo en la competición y temporada seleccionada.	OK
PF-03	Patrón definido en zonas del campo.	*Backend*	Las coordenadas de los eventos de cada jugada coinciden dentro de las zonas del patrón.	OK
PF-04	Patrón filtrado por jugador.	*Backend*	En todas las jugadas de los resultados devueltos aparece el jugador y rol seleccionado.	OK
PF-05	Patrón restrictivo que no devuelve resultados.	*Backend*	La API responde sin errores.	OK
PF-06	Cálculo del resumen global de una consulta.	*Backend*	El número de jugadas y las estadísticas coinciden con una muestra revisada.	OK
PF-07	Cálculo del *ranking* de jugadores a partir de las jugadas devueltas.	*Backend*	El orden de los jugadores muestra su número de apariciones en el patrón, se comprueba con una muestra manual.	OK
PF-08	Generación de *player-insights* para jugador y rol en una consulta concreta.	*Backend*	Las estadísticas y jugadas de ejemplo se corresponden con el jugador y el rol analizado dentro de esa búsqueda.	OK
PF-09	Detección de repeticiones de patrones recurrentes.	*Backend*	Las agrupaciones comparten estructura y representan jugadas equivalentes en diferentes momentos o partidos.	OK
PF-10	Carga inicial de competiciones, temporadas y equipos en interfaz.	*Frontend*	Los desplegables muestran solo las opciones válidas.	OK
PF-11	Actualización en cascada de filtros (competición, temporada, equipo, jugador).	*Frontend*	Al cambiar un filtro de nivel superior se ajustan los filtros dependientes.	OK
PF-12	Construcción de un patrón (pase + tiro) sobre el campo.	*Frontend*	El patrón se representa gráficamente mostrando las elecciones del usuario.	OK
PF-13	Edición y eliminación de pasos ya definidos en el patrón.	*Frontend*	Al modificar o borrar un paso, el estado interno del patrón y su representación se actualizan.	OK
PF-14	Ejecución de búsqueda desde la interfaz con un patrón definido.	*Backend / Frontend*	La consulta se envía al *backend* y los resultados obtenidos son coherentes.	OK
PF-15	Apertura del panel de *player-insights* desde el *ranking* de jugadores.	*Frontend*	Se muestran estadísticas y ejemplos relacionados con el jugador y rol seleccionado en el patrón.	OK

6.2 Casos de estudio

6.2.1 Caso 1: búsqueda de un extremo derecho diferencial

En este primer caso de estudio se plantea un escenario de *scouting*: un club pierde a un jugador importante de su plantilla y necesitan encontrar un sustituto que se parezca, en la medida de lo posible, al jugador que han perdido, y por tanto este sea capaz de realizar una **jugada marca de la casa** de ese futbolista.

Se toma como referencia el perfil de un **extremo derecho zurdo**, similar al de jugadores actuales como Lamine Yamal o Yeremay. Una de las jugadas más características de un extremo zurdo jugando a pierna cambiada consiste en recibir muy abierto en banda derecha, encarar hacia dentro y finalizar con un tiro desde la frontal del área.

Con la herramienta desarrollada en este proyecto podemos definir este patrón de jugada y obtener un listado con jugadores que lo hayan realizado alguna vez. A partir de los resultados obtenidos, el analista puede identificar extremos con un perfil similar al de Lamine Yamal y evaluar si encajarían como posibles sustitutos.

Descripción del patrón táctico

La jugada a buscar se puede resumir de la siguiente forma:

- El extremo recibe el balón abierto en banda derecha y en zonas avanzadas.
- Tras la recepción, realiza una conducción hacia dentro que le llevan desde la banda al carril interior, acercándose a la frontal del área.
- La acción finaliza con un disparo del propio jugador, un tiro con su pierna izquierda desde la frontal.

El objetivo de este caso de estudio es utilizar este mismo patrón para identificar otros jugadores de otra competición que generen acciones similares y puedan ser candidatos a ocupar ese rol.

Configuración de la búsqueda en la herramienta

En este caso no se parte de un jugador concreto, sino que se establece la secuencia de eventos sobre el campo y se buscan todas las coincidencias. El procedimiento para reproducir la búsqueda definida anteriormente es el siguiente:

1. En la barra de filtros se seleccionan la competición y temporada de interés.
2. En el diseñador de jugadas sobre el campo se define un patrón compuesto por tres pasos:
    - **Paso 1**: Recepción del balón en banda derecha, en una zona avanzada del campo del rival. Esta situación se modela como un pase recibido o el inicio de una conducción en esa zona.
    - **Paso 2**: Conducción hacia dentro, desplazando la acción desde la banda al carril interior cercano a la frontal. Se configura un evento de conducción que mantienen la posesión en el mismo equipo.
    - **Paso 3**: Finalización del propio jugador. Este paso se configura como un evento de tiro desde las inmediaciones de la frontal del área rival. Opcionalmente se pueden restringir los *outcomes* a disparos a puerta o goles si se quiere analizar únicamente las acciones más peligrosas.
3. Se lanza la búsqueda y se obtienen todas las jugadas, en las que un jugador ejecuta este patrón.

La configuración de estos tres pasos en la herramienta se muestra en la Figura 6.1, donde se representan sobre el campo la recepción inicial en banda derecha, la conducción hacia dentro y el disparo final desde la frontal del área.

El patrón describe un comportamiento táctico, pero no fija de antemano quién lo ejecuta. Es el motor de búsqueda el que recorre los datos y encuentra qué jugadores, en qué equipos y en qué partidos, han realizado acciones de este tipo.

Resultados e interpretación

A partir del conjunto de jugadas devuelto por la búsqueda, la herramienta proporciona varias vistas útiles que ayudan el trabajo del ojeador:

- Un **resumen global** del patrón, que indica cuántas jugadas de este tipo aparecen en la muestra analizada, en qué competiciones y temporadas se concentran y cuántos equipos participan en estas acciones.
- Un ***ranking* de jugadores**, en el que se ordena cada futbolista según el número de veces que ha participado este patrón.
- Un panel de ***player insights***, que permite seleccionar un jugador concreto del *ranking* y ver sus estadísticas específicas dentro del patrón, junto con ejemplos de jugadas extraídas de la base de datos.

La Figura 6.2 recoge un ejemplo de la vista de resumen y del *ranking* de jugadores para este patrón. Se observa cómo la aplicación muestra el número total de secuencias encontradas, el tiempo medio entre eventos y la lista de futbolistas que más reproducen la jugada, lo que permite identificar rápidamente posibles candidatos.

A partir del *ranking*, el ojeador puede abrir el panel de *player insights* para un jugador concreto y analizar en detalle sus números dentro del patrón (apariciones, goles, porcentaje de conversión, etc.), además de revisar varias jugadas de ejemplo. La Figura 6.3 ilustra esta vista para uno de los candidatos encontrados.

Este caso de estudio muestra que la herramienta permite:

- **Capturar un perfil táctico concreto** de estos tipos de extremo a partir de una jugada característica del fútbol actual, aunque los jugadores de referencia no estén presentes en el conjunto de datos.
- **Explorar la competición en busca de jugadores que replican ese patrón**, mostrando un listado ordenado por volumen de jugadas y eficacia que sirve como punto de partida para el trabajo del ojeador.
- **Reducir el espacio de búsqueda**, apoyándose en datos objetivos para seleccionar un número de candidatos. A partir de los cuales ya se puede complementar el análisis con otros factores (edad, condiciones contractuales, informes cualitativos de *scouts*, etc.).

Aunque la decisión final de fichar a un jugador depende de muchos otros factores ajenos al modelo, este caso de estudio demuestra que la herramienta puede ser un apoyo útil en la fase de selección de perfiles, especialmente cuando se trata de sustituir a un futbolista con una jugada marca de la casa..

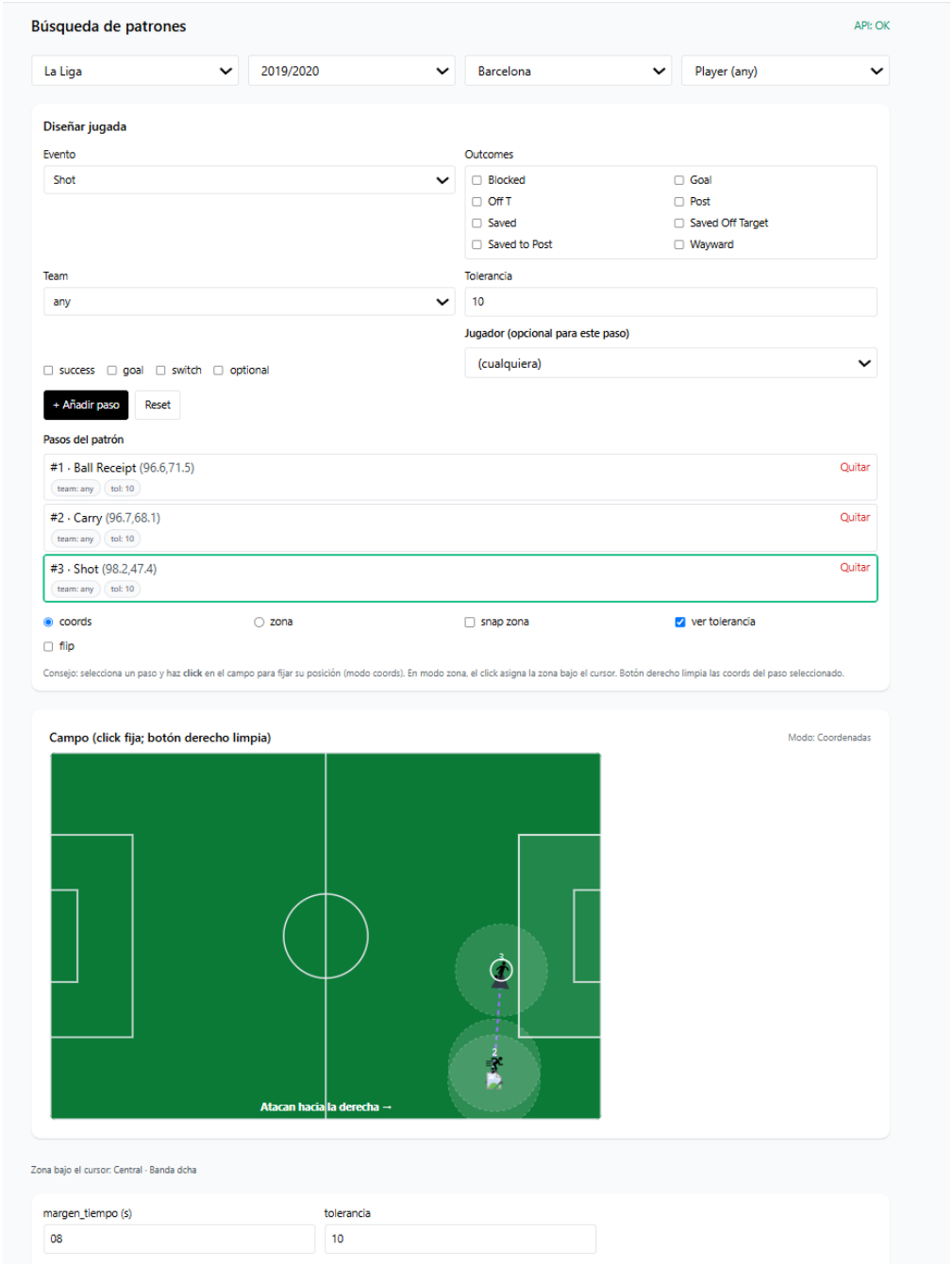


Figura 6.1. *Configuración del patrón de tres pasos en la herramienta.*

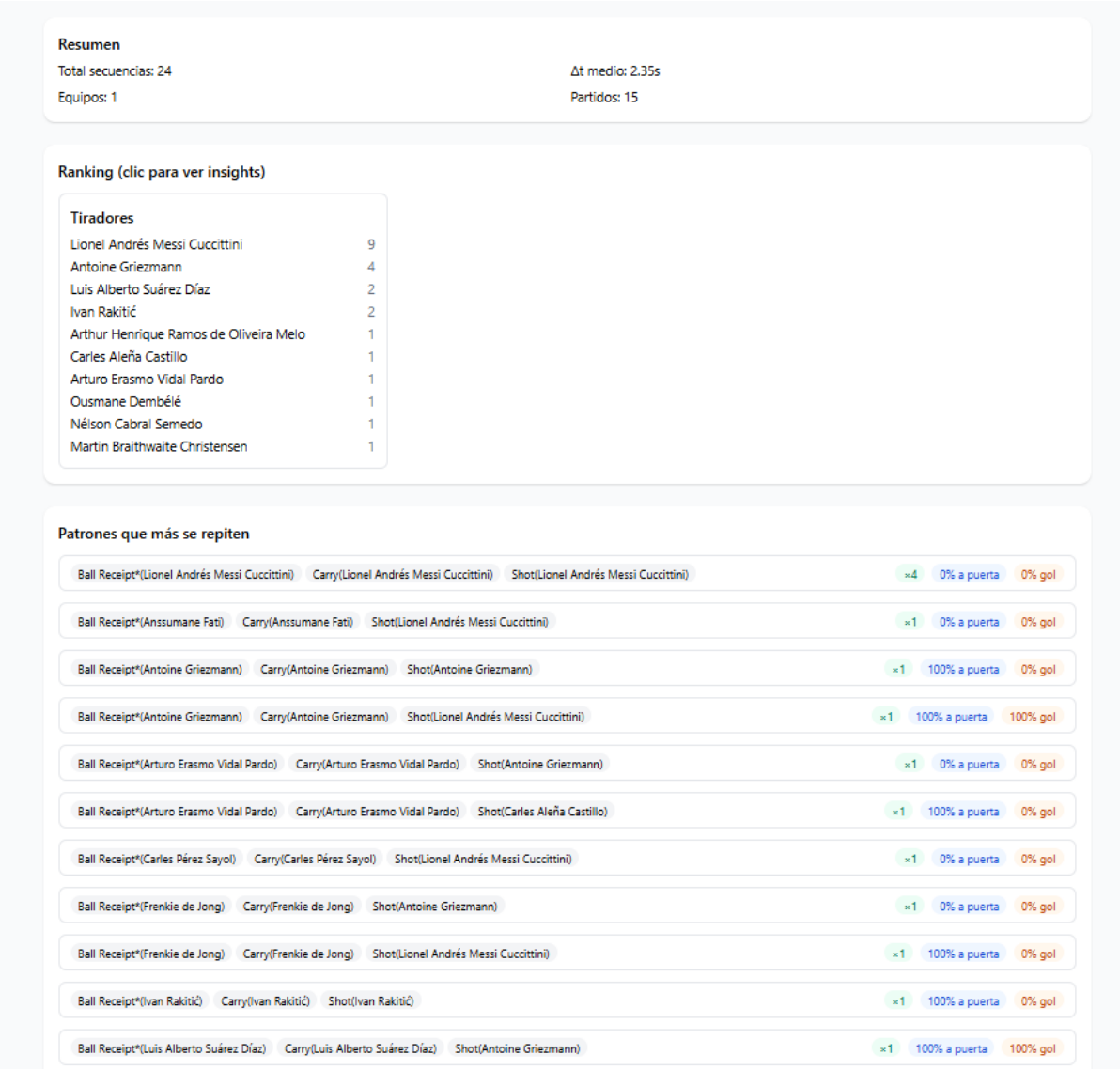


Figura 6.2. *Resumen de resultados y ranking de jugadores para el patrón tipo Lamine Yamal en el caso 1.*

Figura 6.3. *Ejemplo de panel de player insights para un tirador destacado en el patrón.*

6.2.2 Caso 2: análisis de un patrón de centros al segundo palo

El segundo caso de estudio que se plantea se hace desde la perspectiva de un **analista** que quiere entender cómo ataca su equipo en ciertas situaciones. En este ejemplo, se analiza una secuencia de ataque enfocada en los **centros laterales al segundo palo**, una forma de atacar en el fútbol, que busca centrar desde una banda y finalizar la jugada con la llegada de un rematador al segundo palo.

Descripción del patrón táctico

El patrón ofensivo que se quiere estudiar se describe de la siguiente forma:

- El equipo llega por una de las bandas mediante pases y/o conducciones hasta llegar a una zona en la banda en el último tercio.
- Desde esa zona, se ejecuta un **centro lateral** dirigido hacia el segundo palo.
- La jugada finaliza con un **remate** desde ese segundo palo, con una ocasión clara de gol.

El objetivo del caso de estudio es medir hasta qué punto este patrón aparece en los datos históricos, qué equipos lo utilizan con mayor frecuencia y eficacia, y qué jugadores destacan como asistentes y rematadores en estas situaciones.

Configuración de la búsqueda en la herramienta

De nuevo, el proceso comienza definiendo el patrón sobre el campo y dejando que sea el motor de búsqueda el que busque todas las coincidencias en los datos:

1. En la barra de filtros se seleccionan una competición y una temporada, según el análisis a realizar, se puede:
    - Restringir el filtro a un equipo concreto, si se quiere estudiar el patrón ofensivo de tu propio equipo.
2. En el diseñador de jugadas sobre el campo se define un patrón con dos pasos principales:

- **Paso 1**: Centro lateral hacia el área. Se configura un evento de pase (*pass*) cuyo punto de origen se encuentra en la banda y cuya zona de destino se sitúa en el área, cerca del segundo palo.
- **Paso 2**: Remate en el segundo palo. Se define un evento de tiro (*shot*) en la zona donde termina el pase establecido.

3. Además, se especifica que estos dos pasos deben producirse en la misma posesión y para el mismo equipo, de manera que no haya cambios de posesión entre el centro y el remate.
4. Se lanza la búsqueda y se obtienen todas las jugadas que cumplen estas condiciones.

El patrón no establece quién centra o remata, por lo que el motor de búsqueda detecta todas las combinaciones de jugadores que participan en estas jugadas dentro del contexto definido por los filtros.

La Figura 6.4 muestra el patrón configurado sobre el campo, donde se pueden ver las zonas de origen del centro y la posición aproximada del remate en el segundo palo.

Resultados e interpretación

Al igual que en el caso anterior, la herramienta genera automáticamente un resumen global del patrón, un *ranking* de jugadores y un panel de *player insights*. En este caso, además del número total de secuencias y del tiempo medio entre eventos, resulta interesante distinguir entre:

- Los **tiradores** o rematadores que más veces finalizan jugadas de este tipo.
- Los **asistentes** que más centros decisivos ponen hacia el segundo palo.
- Los **pasadores previos**, que participan en la fase de construcción antes del centro.

La Figura 6.5 recoge un ejemplo de esta vista de resultados para el patrón de centros al segundo palo, donde se aprecia el *ranking* separado por roles y los patrones que más se repiten.

A partir de estos rankings, el analista puede abrir el panel de *player insights* para profundizar en el comportamiento de asistentes y rematadores en este tipo de jugadas. La Figura 6.6 muestra, de forma comparada, un ejemplo de panel centrado en los asistentes (izquierda) y otro centrado en los rematadores (derecha).

Con esta información, el analista puede comprobar si el patrón se ajusta a la idea de juego del equipo, identificar qué jugadores participan más a menudo en estas situaciones y evaluar si la eficacia es la esperada (porcentaje de remates a puerta, goles, etc.), apoyando así la toma de decisiones tácticas y de planificación de plantilla. Todo esto muestra que la herramienta no sólo permite contar patrones recurrentes, sino también utilizarlos para preparar tácticamente tanto a equipos como partidos, encontrando débilidades para ciertas jugadas para equipos rivales o darse cuenta qué tipo de acciones son las que tu equipo mejor ejecuta.

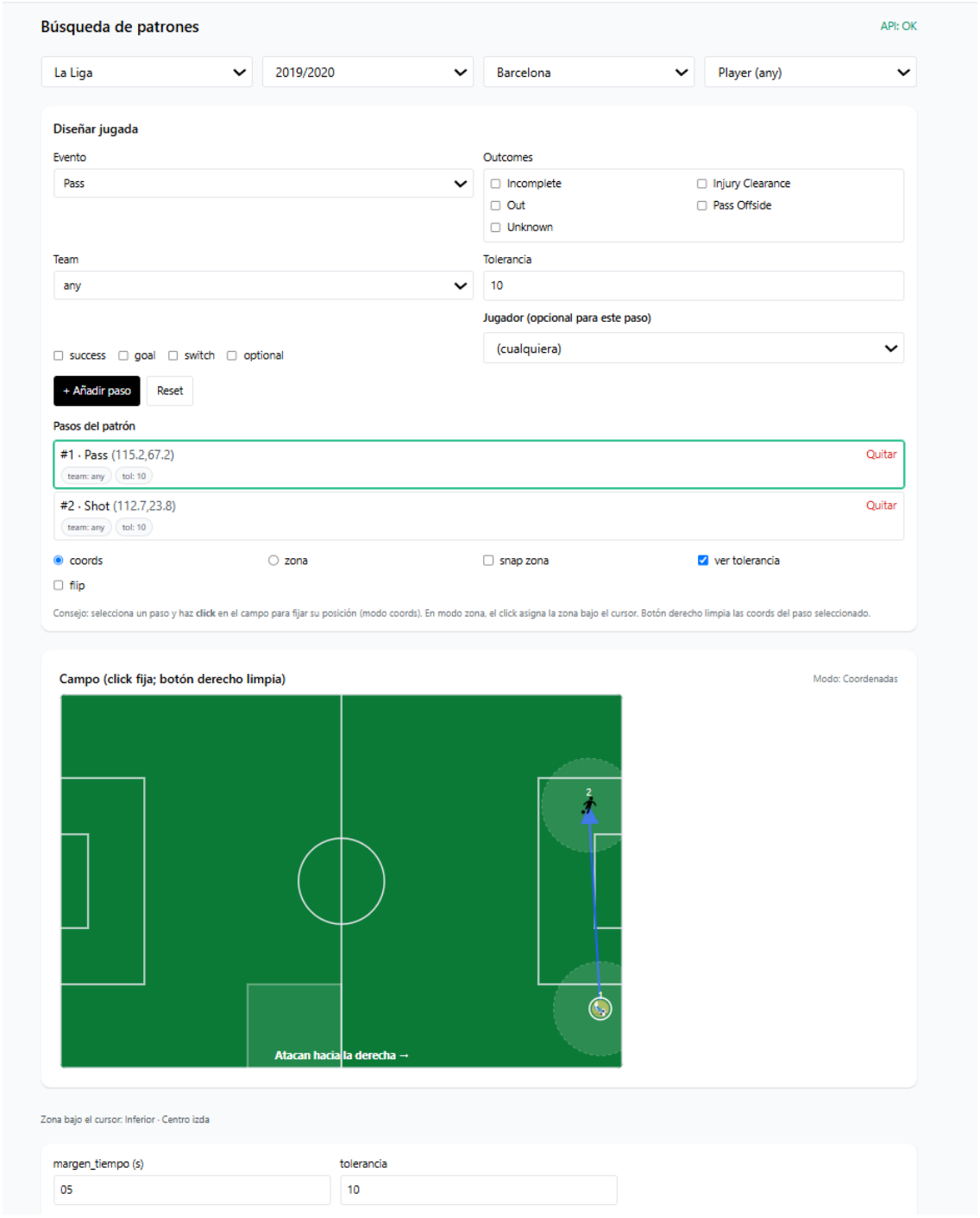


Figura 6.4. *Diseño del patrón de centros al segundo palo en el caso 2.*

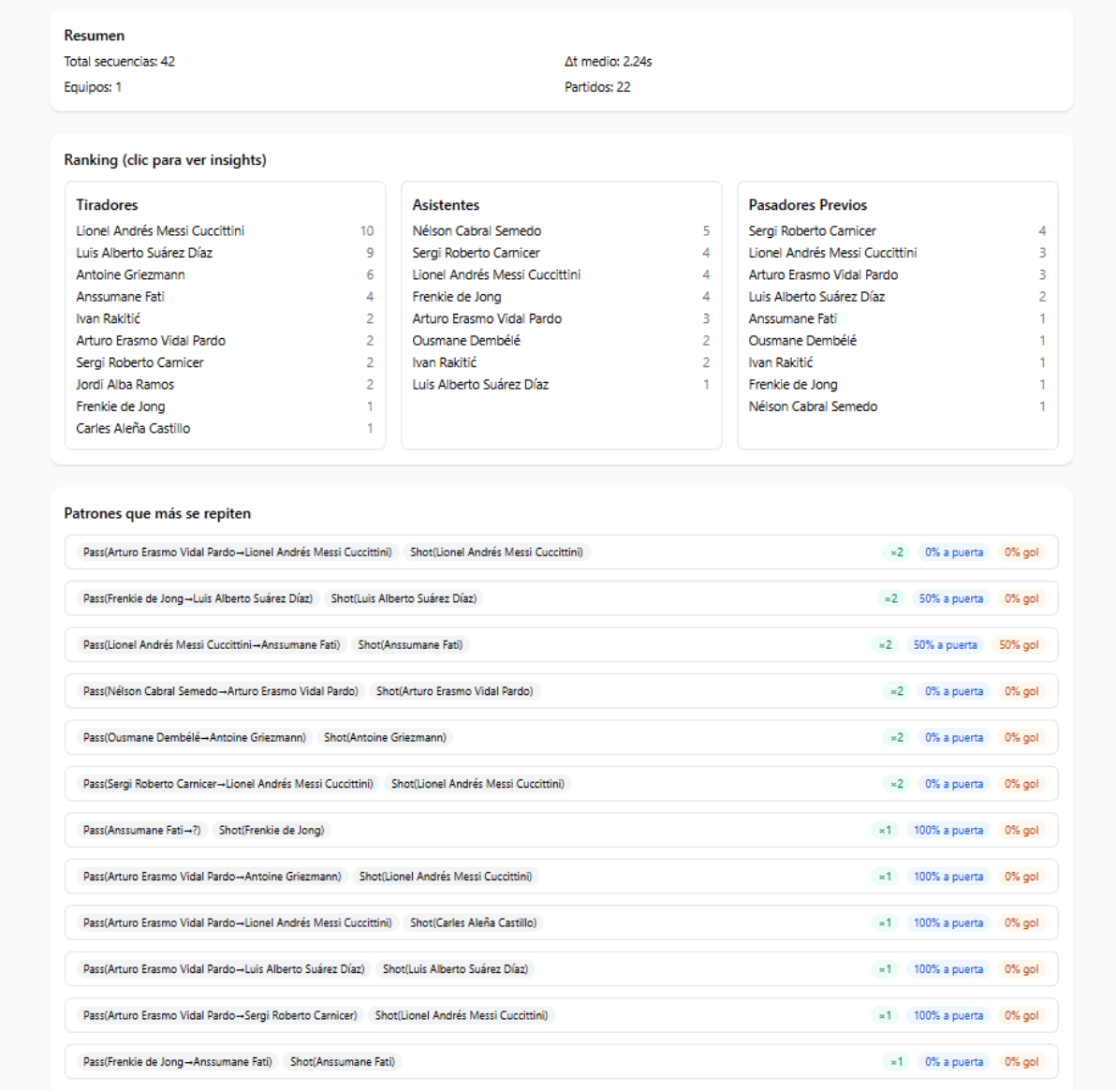
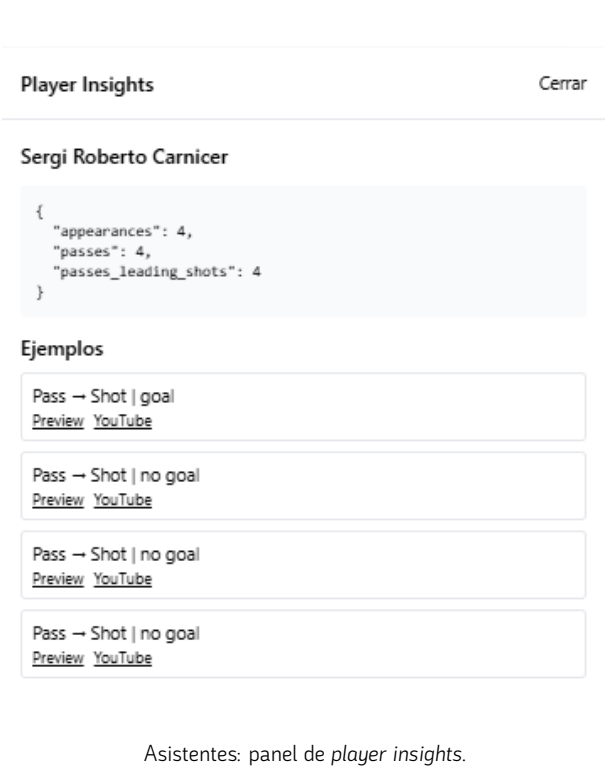
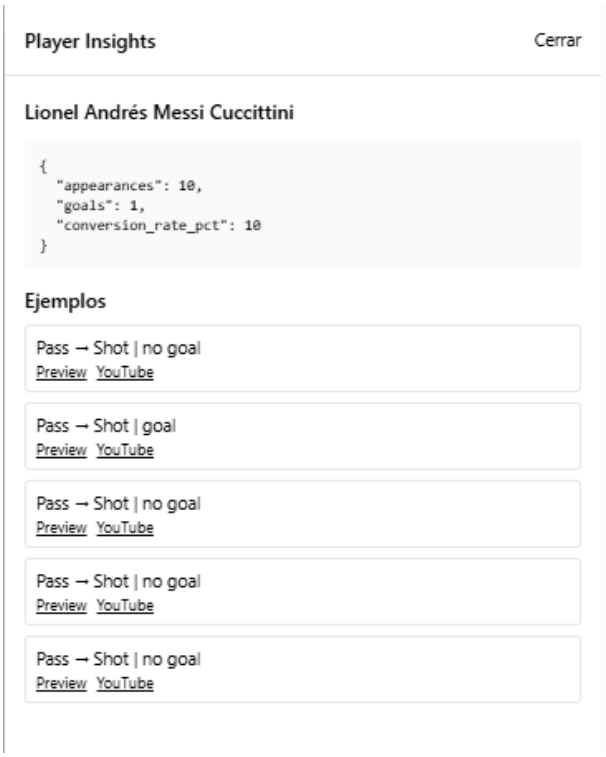


Figura 6.5. *Resumen de resultados y rankings por rol para el patrón de centros al segundo palo en el caso 2.*



Asistentes: panel de *player insights*.



Rematadores: panel de *player insights*.

Figura 6.6. *Comparativa de paneles de player insights para asistentes (izquierda) y rematadores (derecha) en el patrón de centros al segundo palo.*

6.3 Limitaciones encontradas

A pesar de los buenos resultados obtenidos tanto en las pruebas funcionales como en los casos de estudio, durante el desarrollo y validación de la herramienta se han identificado varias limitaciones. Algunas de ellas son relacionadas a los datos utilizados, otras con lo diseños del motor de búsqueda e interfaz, y otras están relacionadas con el rendimiento y la escalabilidad.

Las limitaciones comentadas no deben entenderse como restricciones, sino como puntos de partida para futuras líneas de trabajo. La dependencia de esta base de datos, la riqueza de la interfaz y la escalabilidad de la arquitectura son algunos de los aspectos que se pueden ir mejorando de manera iterativa.

Limitaciones de los datos y del modelo

En primer lugar, el sistema se apoya en un conjunto de datos acotado:

- La base de datos se ha construido a partir de *StatsBomb Open Data*, que proporciona un conjunto cerrado de partidos históricos para varias competiciones y temporadas. Esta elección encaja con los objetivos del TFG y evita depender de servicios comerciales. La misma arquitectura podría conectarse a un proveedor de datos de pago o en tiempo real simplemente adaptando los *scripts* de ingesta, por lo que la limitación afecta al alcance de la base de datos utilizado y no a la funcionalidad de la herramienta.
- El proyecto trabaja con datos de *eventos* (pases, tiros, conducciones, duelos, recuperaciones, faltas, etc.), que ofrecen una granularidad adecuada para estudiar patrones de eventos a nivel de jugada. No se han incorporado datos de *tracking*, que aportarían una granularidad más fina y permitirían extender el análisis a aspectos más detallados. En este TFG nos hemos centrado en el nivel de detalle que ofrecen los datos de evento, suficiente para definir patrones tácticos basados en secuencias de acciones con balón y explorar perfiles de jugadores que los ejecutan.
- Algunas decisiones del preprocesado de datos (normalización de nombres, mapeo de *outcomes*, criterios de éxito, etc.) se han tomado buscando un equilibrio entre fidelidad y simplicidad. Por ejemplo, en las acciones defensivas se han unificado distintos tipos de recuperaciones de los datos de *StatsBomb Open Data* (*ball recovery, interception, duels*, bajo una misma categoría de "recuperación" siempre que el equipo mantenga la posesión en la siguiente acción. De esta forma, los patrones que analizan dónde y cómo se recupera el balón pueden apoyarse en una definición más sencilla de interpretar. Estos criterios son razonables para un TFG, pero podrían refinarse en versiones futuras si se dispone de más tiempo o de información adicional sobre el contexto táctico.

Estas limitaciones condicionan el tipo de preguntas que se pueden responder con la herramienta y obligan a interpretar los resultados en base al contexto de la base de datos utilizada.

Limitaciones del motor de búsqueda y de los servicios de análisis

El motor de búsqueda trabaja sobre la base de datos relacional y recorre las secuencias de eventos para detectar patrones definidos por el usuario. Aunque permite expresar patrones complejos, tiene algunas restricciones:

- El lenguaje de patrones se basa en una **secuencia lineal de pasos**. Cada paso describe un tipo de evento, posibles restricciones de equipo, jugador, zonas del campo y *outcomes*. Esto cubre muchos casos interesantes, pero no permite modelar patrones con ramas alternativas, con múltiples pasos opcionales o estructuras condicionales complejas.
- La noción de tiempo se incorpora mediante el orden de los eventos y la continuidad de la posesión, pero en esta versión no hay reglas temporales como "antes del minuto 20 de partido". Tampoco se han incorporado, restricciones directas sobre el marcador o la fase del encuentro.

- El motor de búsqueda está optimizado para el volumen de datos que maneja este proyecto y se apoya en índices sobre las tablas de eventos. Al no haberse aplicado técnicas más avanzadas ni algoritmos para patrones muy largos, en consultas demasiado complejas, el coste computacional crece y los tiempos de respuesta aumentan.
- Los servicios de análisis construidos sobre los resultados de las búsquedas se basan en métricas sencillas como conteos, porcentajes y agrupaciones por jugador, equipo o partido. No se han integrado, métricas que supongan más complejidad como modelos de goles esperados para cada patrón.

En este sentido, la herramienta es muy útil para explorar patrones definidos y obtener una primera capa de análisis táctico, pero no pretende abarcar todo el campo de explotación avanzada de los datos.

Limitaciones de la interfaz y la experiencia de usuario

La interfaz web desarrollada permite diseñar patrones y explorar los resultados de manera interactiva, pero también presenta ciertas limitaciones:

- El diseñador de jugadas ofrece un modelo visual intuitivo, pero requiere que el usuario tenga un mínimo conocimiento sobre el modelo de datos (tipos de evento, estructura de posesiones, interpretación de zonas del campo).
- El flujo principal de la aplicación se centra en el diseño de un patrón, la ejecución de la búsqueda y la exploración de resultados, pero no incluye funcionalidades como el guardado y carga de patrones favoritos, la comparación entre dos patrones distintos o la exportación de los resultados para compartirlas con otros analistas.
- La herramienta está pensada para un uso individual desde un único navegador y, en su versión actual, no incorpora autenticación de usuarios, control de permisos.

Estas limitaciones no impiden el uso de la aplicación en un entorno de laboratorio o académico, pero serían aspectos a tener en cuenta por parte de un club o una empresa.

Limitaciones de rendimiento y escalabilidad

Por último, la arquitectura elegida condiciona el rendimiento y la escalabilidad de la solución:

- El *backend* se basa en una API desarrollada con un microframework web y una base de datos SQLite almacenada en un único fichero local. Esta combinación es muy adecuada para un entorno de desarrollo y para el volumen de datos manejado en el TFG, pero no es la más apropiada para escenarios con muchos usuarios.
- Se ha implementado una **caché en memoria** asociada a la propia API para ahorrar recalcular resultados de búsquedas recientes. Pero esta caché se pierde si se reinicia el servidor al no está distribuida entre varias instancias.
- No se dispone de métricas precisas sobre el comportamiento del sistema cuando se incrementa el número de partidos, de eventos o de usuarios concurrentes.

En resumen, la herramienta está bien adaptada a los objetivos de un Trabajo Fin de Grado y a un entorno de uso controlado, pero requeriría distintas mejoras para evolucionar hacia una solución dentro de una estructura profesional de club o de empresa de análisis de datos.

7

Conclusiones y trabajo futuro

En este capítulo recogemos las conclusiones finales del Trabajo Fin de Grado, junto con una reflexión sobre las principales dificultades que hemos encontrado y una propuesta de líneas de trabajo futuras.

7.1 Valoración final

El punto de partida de este proyecto fue la necesidad de disponer de una herramienta que permitiera ver más allá de las estadísticas aisladas de un jugador (por ejemplo, saber que un extremo ha realizado tres tiros en un partido) y trasladar ese análisis al nivel de la jugada completa. No sólo contar cuántos eventos realiza un jugador, sino poder ver y comparar cómo son las secuencias de eventos que terminan en esas acciones. Saber cómo recibe el balón, desde dónde progresa, qué decisiones toma y cómo finaliza la jugada.

Consideramos que este objetivo general se ha cumplido, ya que hemos construido sobre una base de datos real una plataforma capaz de identificar patrones de eventos definidos por el usuario y que devuelva resultados útiles y fiables tanto para el análisis táctico como para el *scouting*.

Primero, diseñamos e implantamos un **proceso completo de ingesta y preparación de datos** a partir de una base de datos abierta de *StatsBomb Open Data*. Este proceso lo forma la descarga automatizada de los ficheros originales, su normalización y la carga en un modelo relacional. Sobre esta base de datos se han construido índices y relaciones que permiten ejecutar consultas complejas manteniendo el orden temporal y el contexto de las acciones.

A continuación, desarrollamos el ***backend***. Donde la API expone contratos claros para lanzar búsquedas de patrones, obtener métricas agregadas y generar *rankings* de jugadores por rol. Además también establece metodos para construir catálogos de apoyo (competiciones, equipos, jugadores, tipos de *outcome*, etc.). El motor de búsqueda implementa un algoritmo, capaz de interpretar patrones definidos por el usuario y encontrar todas las jugadas que cumplen las condiciones seleccionadas. A partir de esas secuencias de eventos, los servicios de análisis construyen resúmenes globales, *rankings* y estadísticas detalladas.

Por último, hemos construido una **interfaz web** que permite al usuario definir un patrón, establecer filtros, lanzar la consulta y analizar los resultados mediante resúmenes, listados de jugadas, *rankings* por rol y ejemplos detallados. Los casos de estudio planteados muestran que la herramienta no se queda en un trabajo académico, sino que puede dar soporte a preguntas que se plantean entrenadores y analistas.

En definitiva, consideramos que el proyecto ha cumplido los objetivos planteados al inicio. Hemos pasado de una fuente de datos en bruto a una plataforma completa de búsqueda y análisis de secuencias de eventos, apoyada en una arquitectura por capas clara, con un motor de búsqueda funcional y una interfaz que hace accesible esa potencia al usuario final.

7.2 Dificultades encontradas

A lo largo del desarrollo han aparecido dificultades que han condicionado decisiones del diseño y el alcance final de la solución.

La primera dificultad estuvo relacionada con la **naturaleza de los datos**. Trabajar con ficheros en formato JSON, donde la información viene en estructuras anidadas que cambian según el tipo de evento, nos obligó a dedicar tiempo a estudiar el modelo original y a definir un esquema relacional manejable. La normalización de las tablas, el diseño de claves primarias y foráneas, y la construcción de índices requirieron varias iteraciones hasta encontrar la solución deseada.

La siguiente dificultad importante fue el **diseño del motor de búsqueda de patrones**. Para poder expresar patrones de forma razonable, fue necesario introducir un preprocesamiento del patrón y refinar el algoritmo de búsqueda hasta encontrar un equilibrio entre esta expresividad y tiempos de respuesta aceptables.

En la parte del ***backend***, surgieron dificultades relacionadas con la eficiencia y organización del código. Combinar filtros *SQL*, lógica de dominio y generación de estadísticas obligó a separar responsabilidades entre módulos (ingesta, motor de búsqueda, servicios de análisis, etc.) y a introducir la caché para no recalcular búsquedas costosas en cada petición de resúmenes o *player-insights*.

El principal reto del ***frontend*** fue conseguir una interfaz clara para el usuario. Representar el campo, definir eventos por zonas, gestionar los filtros en cascada y mantener la coherencia entre el estado del patrón, los filtros, los resultados y el panel de *insights* nos obligó a tener mucho cuidad con los componentes y las comunicaciones entre ellos. Además, tuvimos que renunciar a funcionalidades como la autenticación, el guardado de patrones y la exportación de resultados para poder centrar el esfuerzo en el flujo principal de búsqueda y análisis.

7.3 Futuras líneas de trabajo

A partir del sistema actual, existen varias líneas de trabajo que permitirían ampliar y mejorar la plataforma. La mayoría nacen de las limitaciones comentadas en el capítulo anterior.

La primera línea es la **ampliación y actualización de los datos**. Aunque el conjunto de datos utilizado ya proporciona un volumen considerable de partidos, se podría extender el número de competiciones y temporadas, así como automatizar periódicamente el proceso de descarga e ingesta manteniendo la base de datos actualizada.

En cuanto al **motor de búsqueda y los servicios de análisis**, una posible línea de mejora sería la posibilidad de incorporar restricciones temporales como duración total de la jugada o contexto de marcador y permitir ramas condicionales alternativas dentro del patrón. Además, aunque en el modelo de datos ya se han contemplado tablas para información de *tracking*, en este Trabajo Fin de Grado no se han explotado debido a su complejidad y al tiempo disponible. Una línea de trabajo relacionada sería aprovechar estos datos para describir patrones no sólo en términos de eventos de balón, sino también en función de la posición del resto de jugadores.

Respecto al ***frontend***, una posible mejora sería incorporar tanto sistemas autenticación de usuarios como de guardados de consulta y exportación de resultados. Esto permitiría que entrenadores y analistas mantuviesen una biblioteca personal de jugadas o compartiesen patrones dentro de un mismo cuerpo técnico. Además, se podrían añadir nuevas visualizaciones, como mapas de calor del patrón o integraciones más profundas con vídeo como enlazar los datos con cortes de partido si se dispone de acceso a ellos).

Por último, otra línea de trabajo sería estudiar un despliegue en un entorno escalable (contenedores, orquestación, base de datos gestionada en la nube, replicación, etc.). Esto permitiría evaluar el comportamiento de la plataforma con un mayor número de usuarios y acercarla al escenario de uso de un club profesional o de una empresa de análisis de datos.

7.4 Conclusión final

Este Trabajo Fin de Grado demuestra que, a partir de datos abiertos y tecnologías accesibles, es posible construir una herramienta capaz de identificar y analizar patrones de juego sobre datos de partidos reales, acercando el lenguaje de los datos al del cuerpo técnico. En definitiva, este proyecto pretende unir de forma práctica el análisis táctico y la ciencia de datos, dejando la puerta abierta a futuras evoluciones que permitan seguir entendiendo el fútbol no sólo como un conjunto de métricas vacías o individualidades, sino como un juego colectivo lleno de contextos, patrones e historias que los datos ayuden a revelar.

A

Manual de instalación

Este anexo proporciona una guía para la instalación, configuración y verificación necesaria del sistema. El objetivo es que cualquier usuario pueda reproducir la aplicación desde cero, independientemente del sistema operativo utilizado. Esta guía trata la instalación de los dos módulos principales del sistema: el **servidor (API)** y el **cliente (UI)**. Además, se incluye un proceso de **importación de datos de *StatsBomb Open Data***, utilizando el *script* `descargar_datos.py`, que descarga y prepara el conjunto de datos para su posterior carga en la base de datos del sistema.

A.1 Requisitos previos

Antes de empezar con la instalación, es necesario asegurar que el entorno cumple con ciertos requisitos mínimos de *software* y *hardware*. Por ello, en los siguientes apartados se definen las configuraciones recomendadas y los componentes necesarios para el funcionamiento del sistema.

Sistema operativo

El proyecto es multiplataforma y no hay inconveniente de utilizarse en los principales sistemas operativos de hoy en día.

- Si utilizas sistema operativo Windows, mínimo versión 10/11 (64-bit).
- Si utilizas sistema operativo Linux, recomendamos Ubuntu 22.04+/Debian 12+.
- Si utilizas sistema operativo macOS, mínimo versión 12+ (Apple Silicon o Intel).

Software

El programa requiere un conjunto de herramientas tanto para el *backend* como para el *frontend*.

- **Visual Studio Code** (VS Code), entorno de desarrollo integrado recomendado para editar, ejecutar y depurar el proyecto. Se aconseja instalar las extensiones oficiales de Python. Así como la extensión de SQLite, para consultas sobre la base de datos.
- Python 3.10–3.12 (recomendado 3.11). Se puede comprobar la versión que tienes con: `python --version`.
- **Node.js** LTS (18–22) y **npm** ó **pnpm**. Se puede comprobar la versión que tienes con: `node -v` o `npm -v`.
- **Git** para clonar el repositorio: https://github.com/AngelZM22/SequenceLabs.git
- SQLite (incluido con Python, usar la extensión mencionada anteriormente).

Recursos de datos

El funcionamiento de la aplicación depende de la disponibilidad de datos de eventos de partidos de fútbol en formato JSON. Para este proyecto se emplea el conjunto ***StatsBomb Open Data***, un *dataset* público que contiene información de competiciones, temporadas, partidos, eventos y datos espaciales. Enlace de este repositorio: https://github.com/statsbomb/open-data/tree/master/data Los ficheros se almacenan localmente en la carpeta `data/`, y son gestionados automáticamente por el *script* `descargar_datos.py`.

- Ficheros JSON: *competitions, matches, events, three-sixty*.
- Espacio en disco: $\geq$ 8 GB para datos + dependencias.

A.2 Estructura completa del proyecto

En este apartado se muestra la estructura que tiene el proyecto, con la organización de los módulos de *backend, frontend y data*. Esta estructura facilita la mantenibilidad y ayuda a comprender la función de cada componente del sistema.

```
SequenceLabs/
    backend/
        services/
            player_insights.py       # Estadísticas individuales por jugador
            ranking.py               # Rankings por rol y participación
            summary.py               # Resumen y agregaciones de búsquedas
        main_api.py              # Punto de entrada de la API
        database.py              # Carga de JSON, creación tablas e índices
        motorbusqueda.py         # Motor de búsqueda de secuencias
        cache.py                 # Caché por consulta (query_id)
        repeats.py               # Agrupación de jugadas repetidas
        helpers.py               # Funciones auxiliares de procesamiento
        descargar_datos.py       # Descarga automática de StatsBomb Open Data
        api/                     # (opcional) Carpeta con endpoints
        logs/                    # Carpeta opcional para trazas o registros
        __init__.py              # Inicialización del paquete Python

    frontend/
        src/
            main.tsx             # Punto de entrada del cliente (React)
            api.ts               # Comunicación con la API del backend
            types.ts             # Definición de tipos y modelos de datos

            components/          # Componentes para la interfaz
                Campo.tsx            # Campo de fútbol y eventos
                PatternBuilder.tsx   # Constructor de patrones de jugada
                PlayDesigner.tsx     # Editor de jugadas (interactivo)
```

```
            InsightsDrawer.tsx   # Panel lateral con estadísticas indiv
            BarraFiltros.tsx     # Filtros por competición, equipo, jugador
        pages/
            SearchPage.tsx       # Pantalla principal de búsqueda
        OutcomesFallback.ts  # Listado de Outcomes
        eventIcons.ts        # Iconos para los tipos de evento
        Zonas.ts             # Definición de las zonas del campo
        assets/              # Imágenes, íconos y estilos



    package.json             # Dependencias y scripts del cliente
    tsconfig.json            # Configuración de TypeScript
    vite.config.ts           # Configuración de Vite
    public/                  # Recursos estáticos servidos en producción
        icons/               # Almacen de imagenes para los iconos

data/
    competitions.json              # Listado de competiciones disponibles
    matches/                         # Partidos por competición/temporada
    <competition_id>/<season_id>.json
    events/                            # Eventos individuales por partido
    <match_id>.json
    three-sixty/                       # Datos espaciales (opcionales)
    <match_id>.json
    futbol.db       # Base de datos SQLite generada por descargar_datos.py

requirements.txt             # Dependencias del backend (Python)
README.md                    # Descripción general del proyecto
.env.development             # Variables de entorno del cliente (opcional)
.gitignore                   # Exclusiones para Git
```

La separación entre **backend/**, **frontend/** y **data/** permite trabajar de manera independiente sobre cada módulo. El *script* **descargar_datos.py** actúa como intermediario entre los datos externos de *StatsBomb Open Data* y el modelo de la base de datos. Los archivos en **frontend/src/components/** representan las distintas vistas y funcionalidades interactivas que permiten diseñar, ejecutar y analizar jugadas de forma visual.

A.3 Pasos de instalación

En este apartado se detalla el procedimiento para obtener el entorno del proyecto y poder ejecutar la aplicación. Las instrucciones van en orden, comenzando por la obtención del código fuente, la importación de los datos y la ejecución del servidor y del cliente.

A.3.1 Clonado del repositorio

El primer paso es obtener el código fuente del proyecto a través del repositorio oficial en GitHub y verificar su estructura:

1. En una terminal, ejecute:

```
git clone https://github.com/AngelZM22/SequenceLabs.git tfg-futbol
cd tfg-futbol
```

2. Verifique la estructura general del proyecto, validándola con la estructura antes vista.

A.3.2 Importación de datos (*StatsBomb Open Data*)

El proyecto incorpora un *script* diseñado para descargar, verificar y estructurar automáticamente la base de datos de *StatsBomb Open Data*. El *script* `descargar_datos.py` realiza comprobaciones de integridad y evita duplicados.

```
# Para la descarga completa
python descargar_datos.py --data-dir ./data
```

Este proceso puede demorarse. Al finalizar, el *script* muestra un resumen con el número de partidos procesados, archivos generados y posibles advertencias.

A.3.3 Carga y preparación de la base de datos

Una vez estén descargados los datos, es necesario generar la base de datos que servirá de soporte al motor de búsqueda. El *script* `database.py` importa los JSON procesados, crea las tablas, y construye los índices que optimizan las consultas. Para ello, ejecuta estos *scripts* en `/backend`:

1. Crea un entorno virtual, actívalo e instale las dependencias necesarias:

```
# Windows (PowerShell)
python -m venv .venv .venv\Scripts\Activate.ps1
# Linux/macOS
python3 -m venv .venv source .venv/bin/activate


# Instalación de dependencias
pip install --upgrade pip
pip install -r requirements.txt
```

2. Ejecute el *script* para construir la base de datos:

```
python database.py
```

3. Puede verificar si se ha generado correctamente de esta manera, deben salirte resultados:

```
sqlite3 data/futbol.db "SELECT COUNT(*) FROM matches;"
sqlite3 data/futbol.db "SELECT COUNT(*) FROM events;"
```

A.3.4 Arranque del servidor (API)

El servidor incorpora la lógica del sistema y expone los servicios de búsqueda de patrones, *ranking* de jugadores por rol y análisis avanzado de jugadores. Una vez configurada la base de datos, el arranque de la API puede realizarse con el siguiente comando:

```
python -m venv .venv .venv\Scripts\Activate.ps1
uvicorn main_api:app --reload --port 8000
```

Si todo es correcto, el servidor quedará disponible en la dirección `http://localhost:8000/`, lo puede consultar en el navegador. Para comprobar su estado, lo puede hacer mediante el navegador o con un script sencillo como:

```
 curl -X 'GET' \
'http://127.0.0.1:8000/status' \
-H 'accept: application/json'
```

A.3.5 Arranque del cliente (*Frontend*)

El cliente trata la capa de presentación del sistema, forma el diseño del programa y permite interactuar con las consultas y los resultados. Su despliegue se hace de forma local, con un servidor de desarrollo integrado. Es importante saber que, antes de inicializar el cliente, hay que inicializar el servidor, ya que el primero va a intentar comunicarse con el segundo. Para arrancar el servidor, ejecuta estos *scripts* en `/frontend`.

1. Instale las dependencias del cliente:

```
npm install
```

2. Configure la URL de la API en el archivo de entorno:

```
# .env.development VITE_API_URL=http://localhost:8000
```

3. Inicie el servidor de desarrollo:

```
npm run dev
```

4. Acceda desde el navegador a la dirección:

```
http://localhost:5173
```

A.4 Verificación funcional mínima

Cuando se tenga en ejecución el servidor y el cliente, se puede probar el funcionamiento del sistema. Para ello, el usuario debe acceder a la interfaz, aplicar filtros (competición, temporada y equipo) y diseñar un patrón básico (por ejemplo, *Pase* → *Disparo*). La búsqueda debe devolver un conjunto de jugadas coincidentes, estadísticas del patrón y los *rankings* por rol. Además, se debe poder abrir el panel lateral de *insights* de jugador sin reejecutar la búsqueda completa.

A.5 Errores comunes

Tanto en el proceso de instalación como al ejecutar el programa pueden darse errores en el entorno y en la manipulación de datos. A continuación se ven los fallos más habituales y sus posibles soluciones.

Datos no visibles en la aplicación

- **Causa:** La base de datos no se generó correctamente o faltan JSON descargados.
- **Solución:** Repetir el proceso de importación:

```
python descargar_datos.py --data-dir ./data
python database.py
```

Puertos ocupados

- **Síntoma:** Error `Address already in use`.
- **Solución:** Cambiar el puerto por otro libre.

```
uvicorn main_api:app --port 8001
npm run dev -- --port 5174
```

Bloqueos de SQLite

- **Causa:** Acceso simultáneo o apertura con otra herramienta.
- **Solución:** Cierre procesos que accedan a la BD y reinicie el servidor.

Versiones incompatibles de Python o dependencias

- **Solución:** Recrear un entorno limpio con Python 3.11.

```
deactivate
rm -rf .venv
python -m venv .venv
source .venv/bin/activate
pip install -r requirements.txt
```

Problemas de CORS o URL incorrecta

- **Causa:** Variable `VITE.API.URL` incorrecta o falta de permisos CORS.
- **Solución:** Revisar la URL y reiniciar el cliente.

En caso de tener otros errores o que estos persistan, se recomienda revisar los registros del servidor para identificar el `query_id` afectado, los tiempos de respuesta anómalos o posibles cuellos de botella en el motor de búsqueda.

B

Manual de usuario

Este anexo presenta una guía sobre el uso del sistema desde la perspectiva del usuario final. El objetivo es mostrar, de forma visual y práctica, cómo interactuar con la aplicación para diseñar jugadas, realizar búsquedas de patrones y analizar los resultados obtenidos.

B.1 Acceso e inicio del sistema

Una vez completado con éxito la instalación, el sistema puede iniciarse ejecutando simultaneamente estos *scripts*:

- En el servidor (en la carpeta `backend/`):

```
python -m venv .venv .venv\Scripts\Activate.ps1
uvicorn main_api:app --reload --port 8000
```

- En el cliente (en la carpeta `frontend/`):

```
npm run dev
```

- Desde el navegador, acceder a:

```
http://localhost:5173
```

Al acceder, se muestra la pantalla principal de la aplicación (Figura B.1), desde la cual se puede comenzar a construir un patrón de jugada.

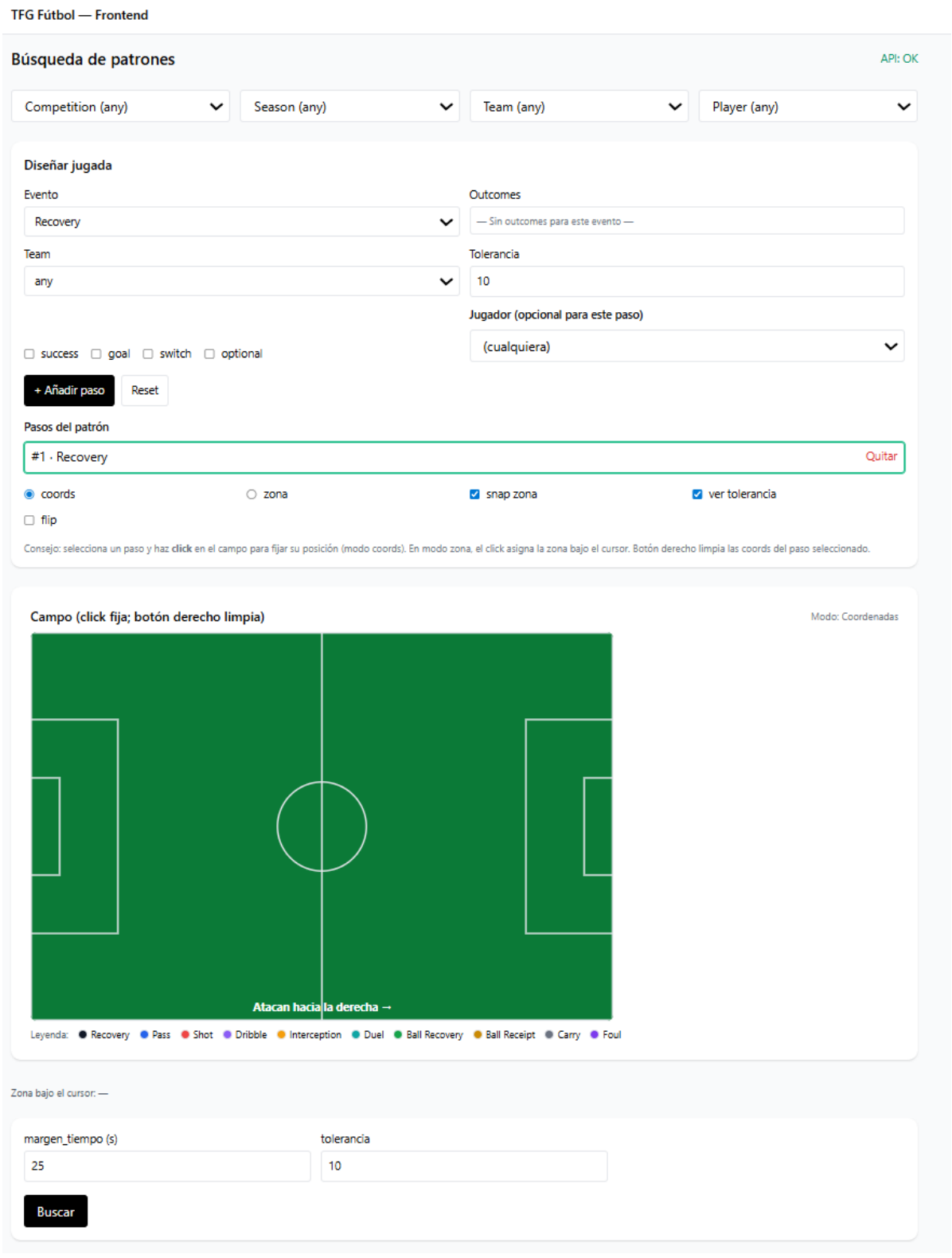


Figura B.1. *Pantalla principal de la aplicación al iniciar el cliente.*

B.2 Aplicación de filtros

Antes de definir la jugada, el usuario puede delimitar el contexto de análisis mediante el uso de la barra de filtros. (Figura B.2). Estos filtros permiten restringir la consulta a una competición, temporada, equipo o jugador específicos.

- **Competición:** liga o torneo sobre el que se desea analizar las jugadas.
- **Temporada:** año o edición del torneo seleccionado.
- **Equipo:** conjunto sobre el que se centrarán las jugadas.
- **Jugador:** permite filtrar por participación individual.

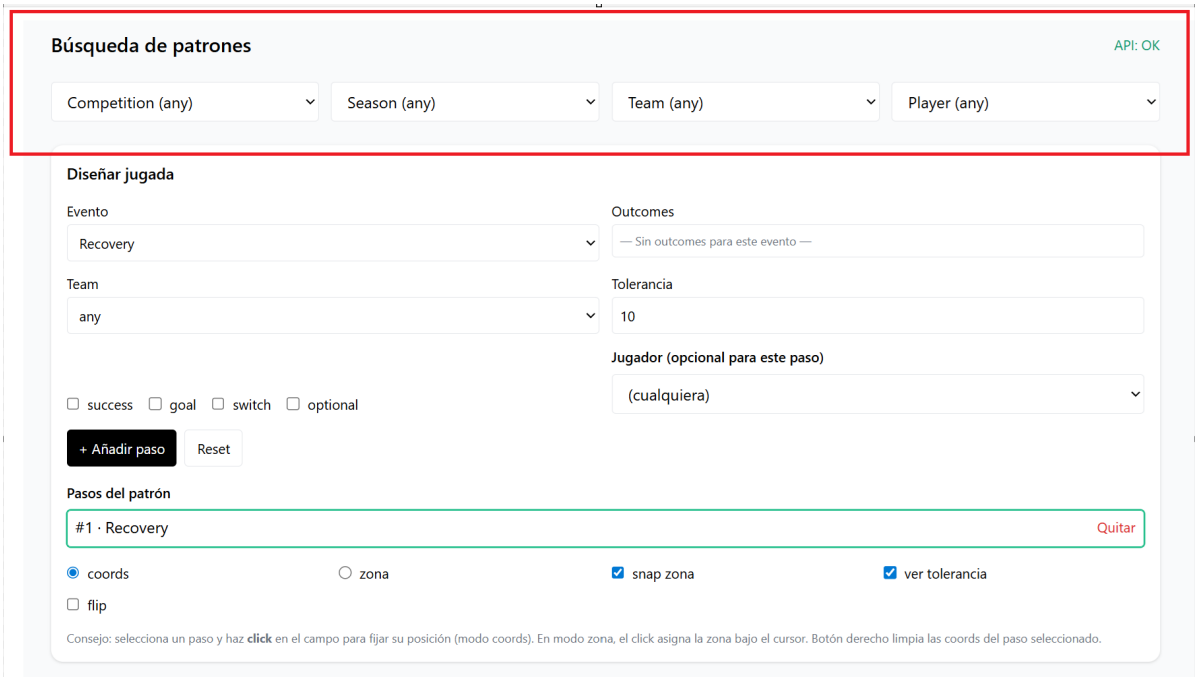


Figura B.2. *Barra de filtros con selección de competición, temporada, equipo y jugador.*

B.3 Diseño del patrón de jugada

El centro de la aplicación es el diseñador de jugadas, una interfaz interactiva que permite definir secuencias de eventos sobre un campo de fútbol. El usuario puede añadir eventos, establecer sus zonas o coordenadas, y ajustar tolerancias o condiciones de éxito.

1. Seleccione el tipo de evento en el panel lateral (pase, disparo, regate, recuperación, etc.).
2. Opcionalmente, elija los resultados esperados del evento *pase completado*, *tiro a puerta*, etc.
3. Dale al botón de agregar.
4. Haga clic sobre el campo para ubicar el punto de inicio o la zona del evento.
5. Ajuste atributos opcionales: *success*, *goal*, *switch*, *optional*.

La Figura B.3 muestra un ejemplo de patrón simple: un pase seguido de un disparo.

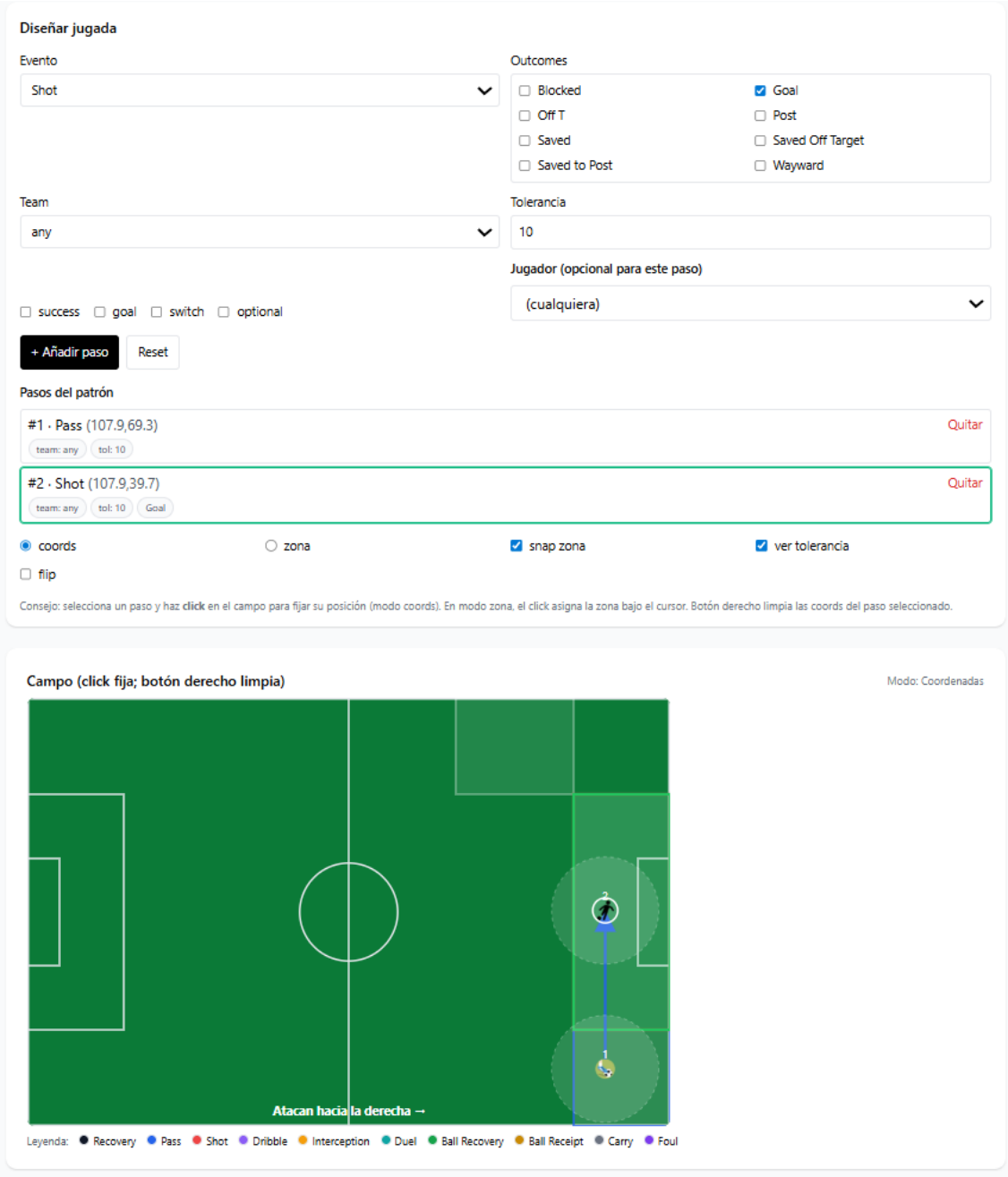


Figura B.3. *Diseño de un patrón de jugada (Pase → Disparo) sobre el campo.*

El usuario puede invertir el sentido del ataque, eliminar pasos, o modificar zonas específicas. Los iconos de color indican el tipo de evento y el estado de la acción (éxito, gol, pérdida, etc.).

B.4 Ejecución de la búsqueda

Una vez definido el patrón y configurados los filtros, el usuario puede iniciar la búsqueda pulsando el botón **"Buscar"**. El sistema envía la consulta a la API, que procesa los eventos en la base de datos y devuelve las secuencias coincidentes. Cuando la respuesta está lista, se actualizan los paneles de resultados y estadísticas. La Figura B.4 muestra la pantalla de resultados tras ejecutar una búsqueda.

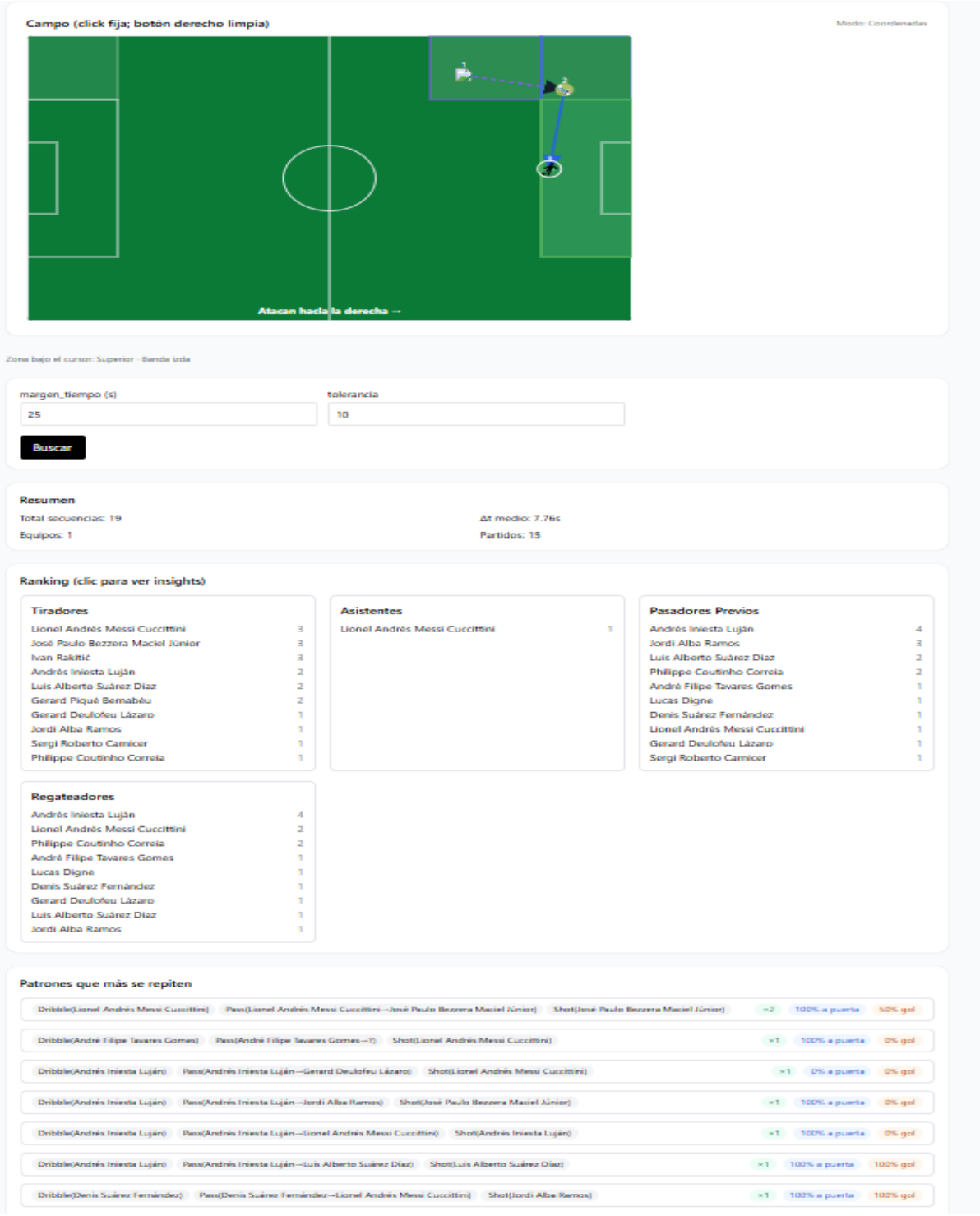


Figura B.4. *Visualización de resultados tras ejecutar la búsqueda.*

Cada resultado corresponde a una jugada real que cumple las condiciones definidas por el patrón. El usuario puede recorrer la lista, abrir jugadas concretas y visualizar al detalle.

B.5 Análisis de resultados y estadísticas

El sistema presenta distintos niveles de análisis para interpretar la información devuelta por la búsqueda:

1. **Resumen del patrón:** muestra la frecuencia total de aparición, el número de equipos involucrados y el tiempo medio entre acciones.
2. ***Ranking* por rol:** identifica a los jugadores con mayor participación según su función en la jugada (tirador, asistente, pasador previo, recuperador, etc.). Puede verse en la Figura B.5, donde se muestra el panel de ranking con los jugadores más destacados.
3. **Repeticiones (*repeats*):** agrupa jugadas similares, mostrando patrones recurrentes dentro de un mismo equipo o competición. Puede verse en la Figura B.7.
4. ***Player-Insights*:** El usuario puede pulsar sobre cualquier jugador del *ranking* para abrir un panel lateral con sus estadísticas individuales, como número de participaciones, partidos en los que aparece, y ejemplos visuales de jugadas donde intervino (Figura B.6).

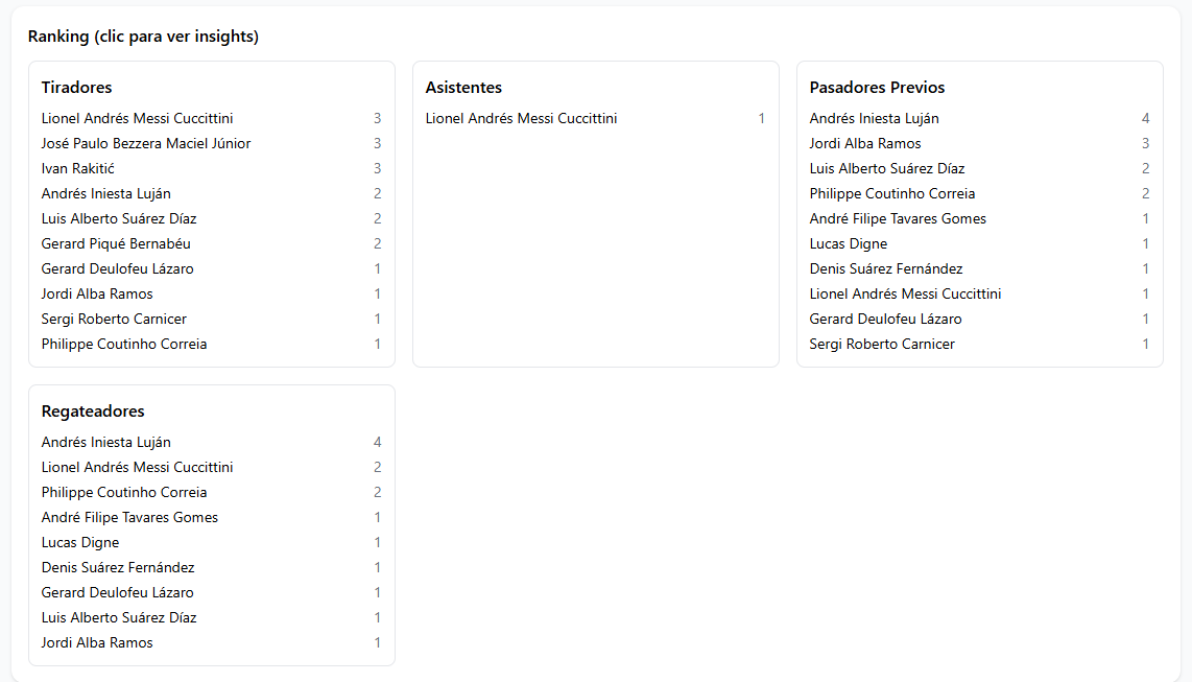


Figura B.5. *Panel de ranking con los jugadores más destacados en el patrón analizado.*

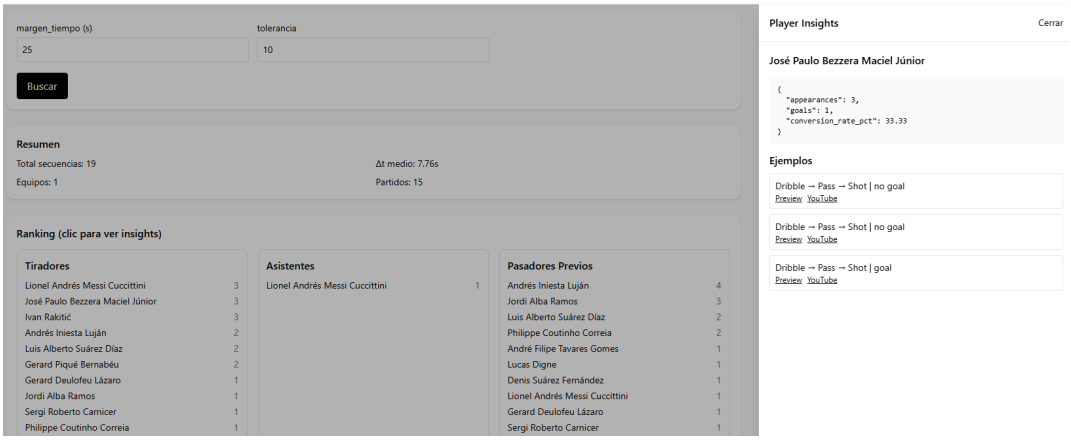


Figura B.6. *Panel lateral de insights individuales por jugador.*

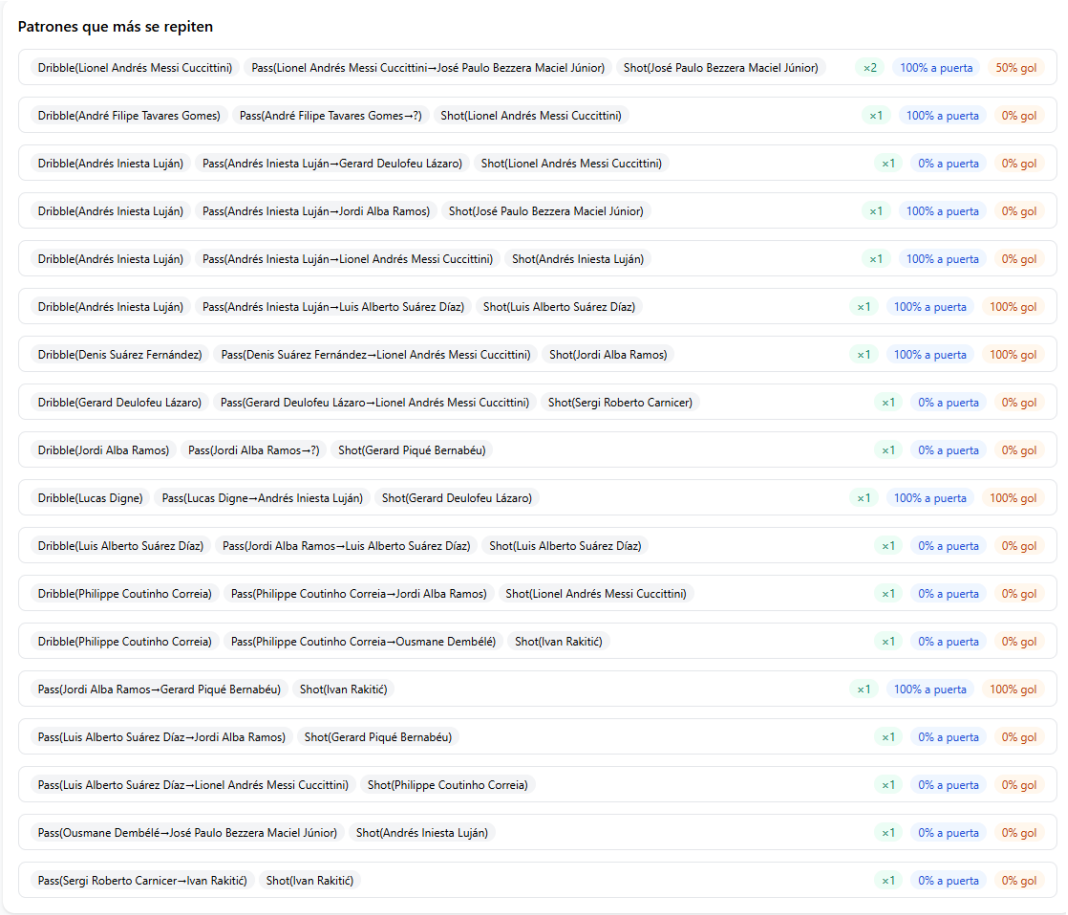


Figura B.7. *Repeticiones de patrones*

B.6 Refinado de consultas y nueva ejecución

Si el usuario desea modificar parámetros (por ejemplo, cambiar el jugador o ajustar tolerancias espaciales), puede hacerlo directamente sobre la interfaz. El sistema invalidará automáticamente la búsqueda anterior y generará un nuevo identificador de consulta, garantizando que los resultados reflejen únicamente los criterios vigentes.

Este mecanismo mejora la usabilidad y evita inconsistencias entre consultas.

B.7 Buenas prácticas de uso

- Utilizar los filtros de competición, temporada y equipo. Ayuda a reducir el tiempo de búsqueda.
- Evitar tolerancias excesivamente amplias, ya que incrementan el tiempo de búsqueda.
- Revisar las estadísticas del resumen para validar la coherencia de los resultados.
- Aprovechar el *ranking* para identificar jugadores con alta frecuencia en patrones específicos.
- Ejecutar periódicamente el *script* de **`descargar_datos.py`**, para mantener la base de datos actualizada.

El sistema está diseñado para permitir al cliente una experiencia fácil para un análisis completo, permitiendo a investigadores, analistas y entrenadores explorar patrones de forma interactiva, sin necesidad de tener conocimientos sobre la base de datos.

Bibliografía

[1] BarçaInnovationHub. "Mitos y certezas de la inteligencia artificial scouting". In: *https://barcainnovationhub.fcbarcelona.com/es/blog/mitos-y-certezas-de-la-inteligencia-artificial-scouting/* (2023).

[2] DBeaver Corp. *DBeaver Community*. Consultado el 25 de noviembre de 2025. 2025. URL: https://dbeaver.io.

[3] HUDL. "Wyscout". In: *https://www.hudl.com/en$_g$b/products/wyscout* (2025).

[4] Hudl. *Hudl Wyscout*. Consultado el 25 de noviembre de 2025. 2025. URL: https://www.hudl.com/en_gb/products/wyscout.

[5] Hudl StatsBomb. *Using StatsBomb IQ for Player Recruitment: Forwards*. Consultado el 25 de noviembre de 2025. 2021. URL: https://blogarchive.statsbomb.com/articles/soccer/using-statsbomb-iq-for-player-recruitment-forwards/.

[6] InnovacionDigital. "Tecnología en el fútbol: Cómo el análisis predictivo y la IA transforman las prácticas del Manchester City". In: *https://www.innovaciondigital360.com/i-a/tecnologia-en-el-futbol-como-el-analisis-predictivo-y-la-inteligencia-artificial-transforman-los-entrenamientos-del-manchester-city/* (2024).

[7] InStat Sport. *InStat Sport*. Consultado el 25 de noviembre de 2025. 2025. URL: https://instatsport.com.

[8] KPMG. "Impacto socio-económico del fútbol profesional en España". In: *https://assets.laliga.com/assets/2023/09/28/originals/39498f156b1e8f503e793b4b882e584c.pdf* (2023).

[9] Meta Platforms, Inc. *React Documentation*. Consultado el 25 de noviembre de 2025. 2025. URL: https://react.dev.

[10] Metrica Sports. *Metrica Sports*. Consultado el 25 de noviembre de 2025. 2025. URL: https://www.metrica-sports.com.

[11] Microsoft. *TypeScript: JavaScript With Syntax For Types*. Consultado el 25 de noviembre de 2025. 2025. URL: https://www.typescriptlang.org.

[12] Dani Pérez. "InStat – La plataforma de scouting que tu equipo necesita". In: *https://objetivoanalista.com/instat-plataforma-de-scouting/* (2025).

[13] Ken Schwaber and Jeff Sutherland. "La Guía Scrum. La guía definitiva de Scrum: las reglas del juego". In: *https://scrumguides.org/docs/scrumguide/v2020/2020-Scrum-Guide-Spanish-European.pdf* (2020).

[14] SQLite. *SQLite Home Page*. Consultado el 25 de noviembre de 2025. 2025. URL: https://sqlite.org/.

[15] StatsBomb. *StatsBomb Open Data*. Consultado el 25 de noviembre de 2025. 2025. URL: https://github.com/statsbomb/open-data.

UNIVERSIDAD
DE MÁLAGA

uma.es

E.T.S. de Ingeniería Informática
Bulevar Louis Pasteur, 35
Campus de Teatinos
29071 Málaga