universidad de león

Escuela de Ingenierías

Industrial, Informática y Aeroespacial

GRADO EN INGENIERÍA INFORMÁTICA

Memoria

ULTRA FANTASY

Autor: Alonso Prada Merayo

27 de diciembre del 2022

Introducción

En este documento se explicará de forma detallada todo lo relacionado con el desarrollo, funcionamiento, tecnologías que se utilizan, problemas que resuelve, posibles mejoras futuras, problemas que han ido apareciendo en el progreso de la aplicación y las soluciones que se han ido tomando, entre otra mucha información.

La aplicación es una página web que servirá como herramienta de ayuda a cualquier participante de juegos estilo Fantasy (un simulador de hacerte tu propio equipo para conseguir el mayor número de puntos posibles), dándote ayudas sobre qué jugador de La Liga deberías fichar. La aplicación tendrá varias secciones, siendo la más importante el apartado de búsqueda de jugadores, el cual permite al usuario primero hacer una búsqueda de en qué jugadores está interesado, aplicando los diferentes filtros. Una vez realizada la búsqueda se podrá acceder al perfil de cualquiera de los jugadores encontrados, y dentro de éste, se te recomendarán algunos jugadores semejantes.

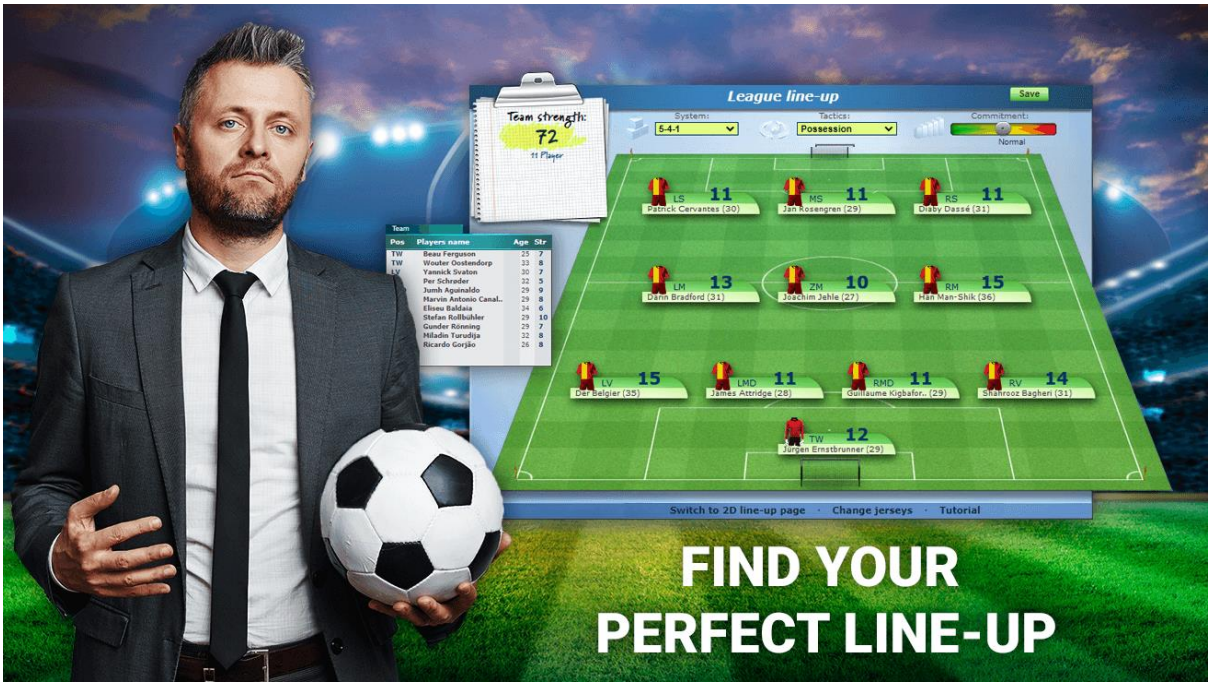
Nuestra aplicación está orientada a cualquier usuario promedio, independientemente de sus conocimientos de la liga española, y de cómo sea su Fantasy particular. También en el futuro si se continuase este proyecto se podría extender a otras ligas importantes y competiciones.

Descripción del problema

El deporte y por ende el mercado del fútbol es uno de los más importantes y duraderos de toda la historia moderna. Ya sea porque eres un fan de un equipo, de un jugador o simplemente es algo de lo que casi cualquier persona sabe un mínimo, mucha gente se ve motivada a hacer algo relacionado con este deporte. Entre esas cosas están los Fantasy: simuladores de una liga interna, ya sea pública o con tus amigos, intentando conseguir la mayor cantidad de puntos posibles de cada jugador. Sin embargo en los Fantasy no hay excesivos recursos informativos a los usuarios, y muchos tienden a ser incluso “engañosos”, intentando vender jugadores promesa a la mínima que tienen 2 partidos decentes, acabando la mayoría de vuelta al banquillo.

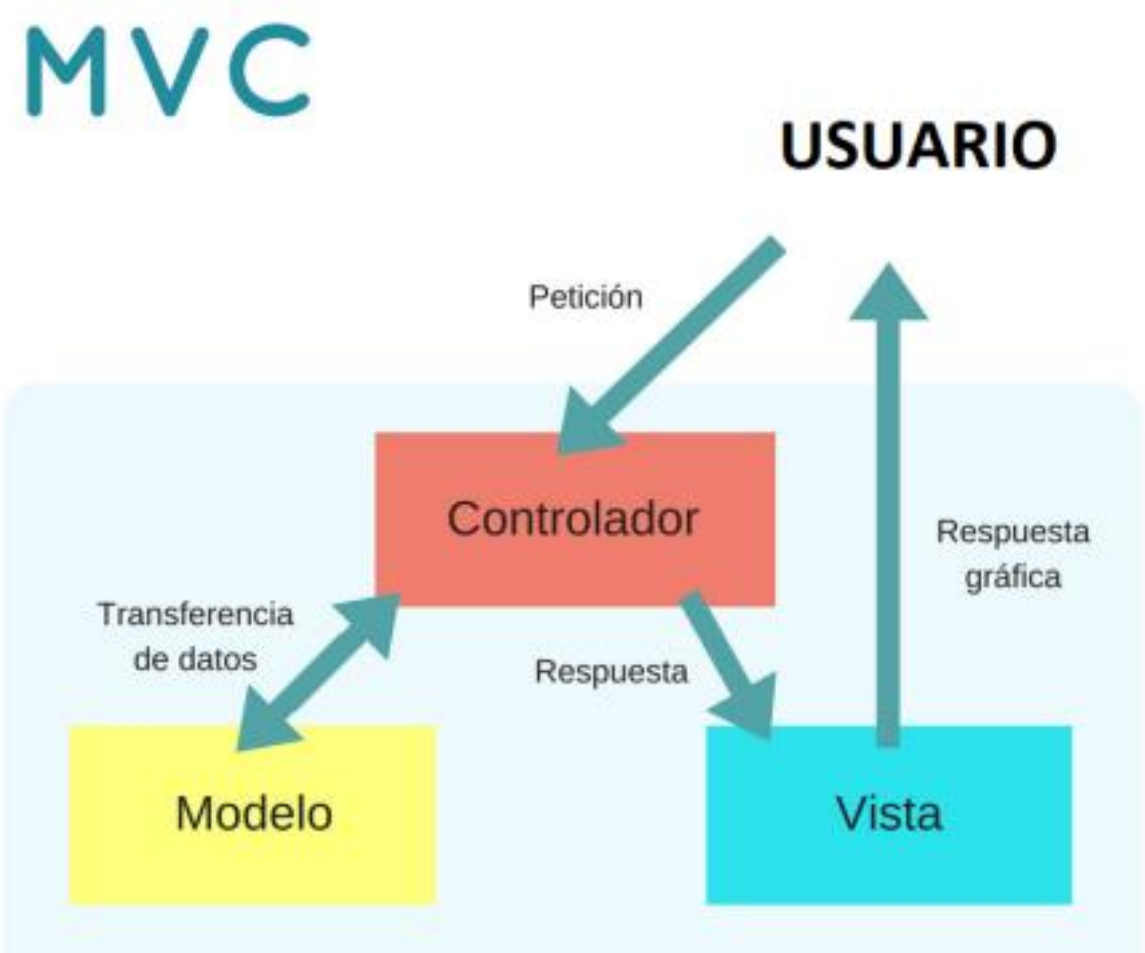
Otro factor es la cantidad de jugadores y ligas profesionales que hay en la liga, seguir la pista a un jugador se vuelve una tarea difícil, más si éste no es un jugador muy popular.

Estas son las razones por las que decidí crear un sistema de recomendación para fichajes, y que las personas puedan utilizarlo de manera simple y sin gastos económicos. La aplicación mediante una selección de varios filtros, y un par de clicks no sólo mostrará los jugadores que están dentro de esos rangos, sino que también te recomendará jugadores similares basándose en tus búsquedas anteriores o jugadores similares basándose en el jugador que estés ojeando en ese mismo momento, permitiendo así al usuario ver una gran cantidad de jugadores similares en un breve periodo de tiempo.



Herramientas de desarrollo

El desarrollo se ha orientado a hacer una aplicación web, que permita la conexión a la misma desde cualquier dispositivo ya sea móvil u ordenador, la aplicación sigue el patrón de diseño de **Modelo-Vista-Controlador** conocido como Model-View-Controller en inglés (MVC), por el que cada componente tendrá la siguiente funcionalidad.



- **Vista:** es la parte visual de la aplicación donde se mostrará la información

que se solicite.

- **Controlador:** es donde está la lógica del sistema, se reciben las peticiones del usuario desde la vista, se analizan y se ejecuta las diferentes funciones para enviar una respuesta.

- **Modelo:** es donde se almacena toda la información de la aplicación, y recibirá peticiones desde el controlador según la información solicitada.

En este caso la vista corresponde con el frontend, el controlador sería el backend y el modelo es la base de datos.

Todo el desarrollo se ha hecho bajo un ordenador con distribución linux, más concretamente con ubuntu (**versión 20.04**) que es un sistema operativo de software libre y código abierto, y las herramientas para programadores que ofrece este tipo de sistema operativo frente a otros, facilitan mucho el trabajo de desarrollo.

Y por último para la creación de la propia aplicación se han utilizado las siguientes herramientas:

Para el **backend** se ha utilizado **Node.js**, tecnología que he tenido que aprender a utilizar este cuatrimestre, aunque ya estuve interesado en usarla en algún proyecto en el pasado. Si hay algo que me ha sorprendido es la comodidad para usarlo y facilidad de hacer una base para la web.

Para iniciar un nuevo proyecto mediante Node.js, lo único que tendríamos que hacer es descargarlo desde la página oficial, una vez descargado lo instalamos y por último, utilizaremos en un terminal el comando **npm init.**

Después, nos pedirá diferente información relacionada con nuestro proyecto y una vez acabado, ya podemos configurarlo para que actúe como un **servidor web** e iniciarlo mediante el comando **npm start.**

Aquí ya podemos añadir a nuestro proyecto **diferentes módulos** para que nos ayuden en el desarrollo de la aplicación, mediante el comando:

En mi caso se han utilizado los siguientes módulos:

- **Nodemon**, monitorea los cambios en el código fuente que se está desarrollando y automáticamente reinicia el servidor.

- **Cors**, nos permite el intercambio de recursos entre los diferentes componentes, en nuestro caso entre el frontend y el backend.

- **Express**, framework web más popular de Node, y es la librería subyacente para un gran número de otros frameworks

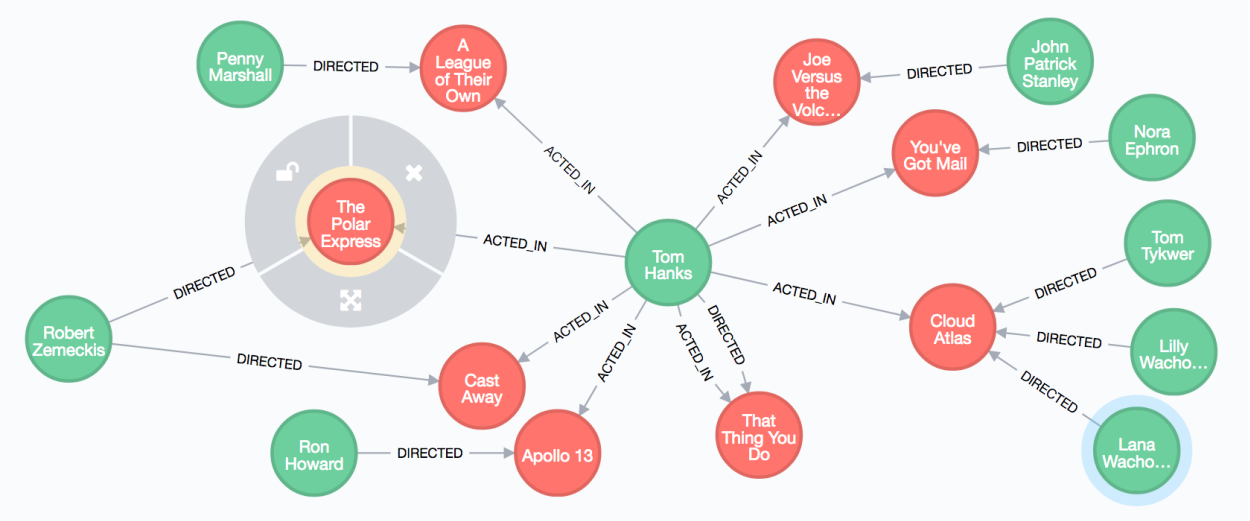
- **Neo4j-driver**, nos permite comunicarnos desde el backend a la base de datos, abriendo una sesión para realizar consultas.

En el área del **frontend** se ha utilizado **Vue.js**, que es un framework que conocía de prácticas externas pero nunca he usado desde 0 en un proyecto. Como apoyo para dar una mayor calidad a los elementos visuales se ha utilizado **Boostrap** que es un framework de css que me permitirá crear los estilos de los elementos de la página de una forma sencilla y también se usará **FontAwesome** que es un conjunto de iconos que me servirá para decorar la aplicación.
Al iniciar Vue en nuestro proyecto viene con lo básico para que puedas elegir qué es lo que quieres y lo que no y modeles el proyecto a tu gusto, aunque sin ningún tipo de duda lo mejor de usar Vue es la capacidad de poder trabajar en la página y ver los cambios en tiempo real, sin necesidad de recargar la página ni reabrir el servidor.
Para crear nuestro proyecto con Vue tenemos que utilizar el comando **vue create (nombre del proyecto)**
Seleccionamos la configuración que deseemos y ya podemos empezar a trabajar.

Por último la base de datos, en la cual se utilizará la tecnología **Neo4j,** es una base de datos orientada a grafos, por lo que podremos ver cómo está almacenada la información de una manera fácil y detallada. Neo4j utiliza **Cypher** como lenguaje propio para realizar consultas, similar a SQL pero relativamente más cómodo de usar, especialmente con Neo4j de backend.

Así se vería la representación de la información de una base de datos creada

mediante Neo4j.



Los círculos son los diferentes nodos existentes y las flechas uniéndolos las relaciones, tanto los nodos como las relaciones pueden tener propiedades lo que es muy útil para tener una mayor y mejor información de toda nuestra base de datos.

En mi caso al estar utilizado ubuntu como sistema operativo para poder acceder a trabajar con la base de datos necesitaremos abrir un terminal de cypher mediante el comando **cypher-shell** o ir en tu navegador a http://localhost:7474.

Una vez dentro de la consola hay que loguearse y quitar la contraseña genérica inicial neo4j, sustituyéndola por otra. Con *:exit* en la terminal podemos salir en cualquier momento. Lo más importante de neo4j para nosotros es la posibilidad de hacer consultas ya sea para crear o buscar información almacenada.

La aplicación quedaría definida así con las tecnologías y componentes explicados anteriormente.

Aunque se ha utilizado, el uso de Github ha sido muy secundario al trabajar con un único ordenador. Al solo requerir tener una copia de seguridad de la misma yo he utilizado para almacenar mis archivos temporalmente Google Drive.

Descripción de la aplicación

General

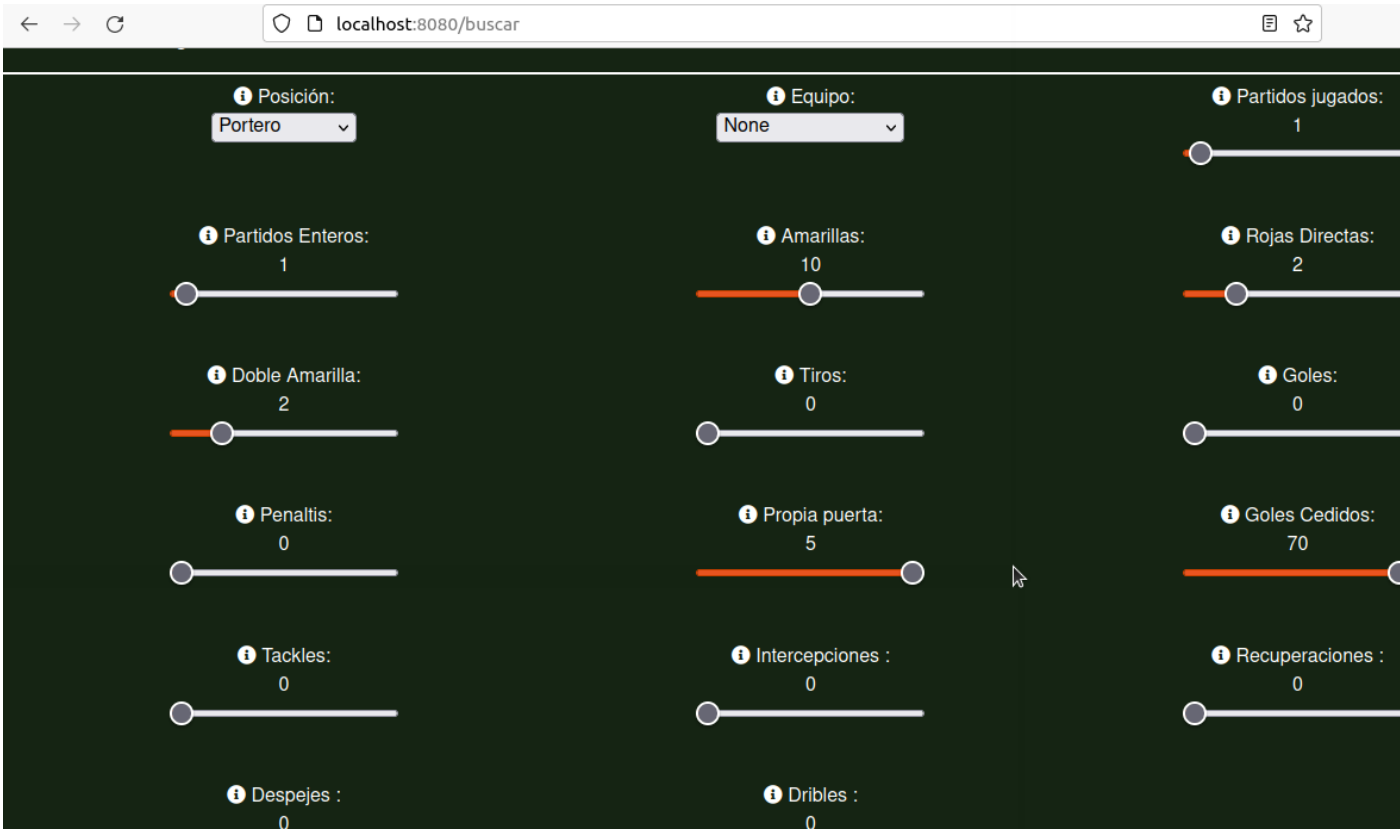
Nuestra aplicación se divide en tres partes:

- Una página de inicio breve introduciendo al usuario a la web misma
- Una página de buscar jugadores, donde está toda la lógica del recomendador.
- La página del perfil de usuario.

Frontend

Aquí nos vamos a centrar en el área de **búsqueda de jugadores** que es donde se realiza la mayoría de acciones de la aplicación, en esta sección lo primero que nos encontramos es la selección de diferentes filtros, los cuales nos ayudarán a buscar el jugador que queramos.

Cada filtro tiene a su lado un icono con una ***i*** que contiene una breve explicación del mismo, como podemos ver en la siguiente imagen:



El único filtro obligatorio que debemos seleccionar es el de la posición, todos los demás son opcionales que podemos modificar para obtener mejores resultados.

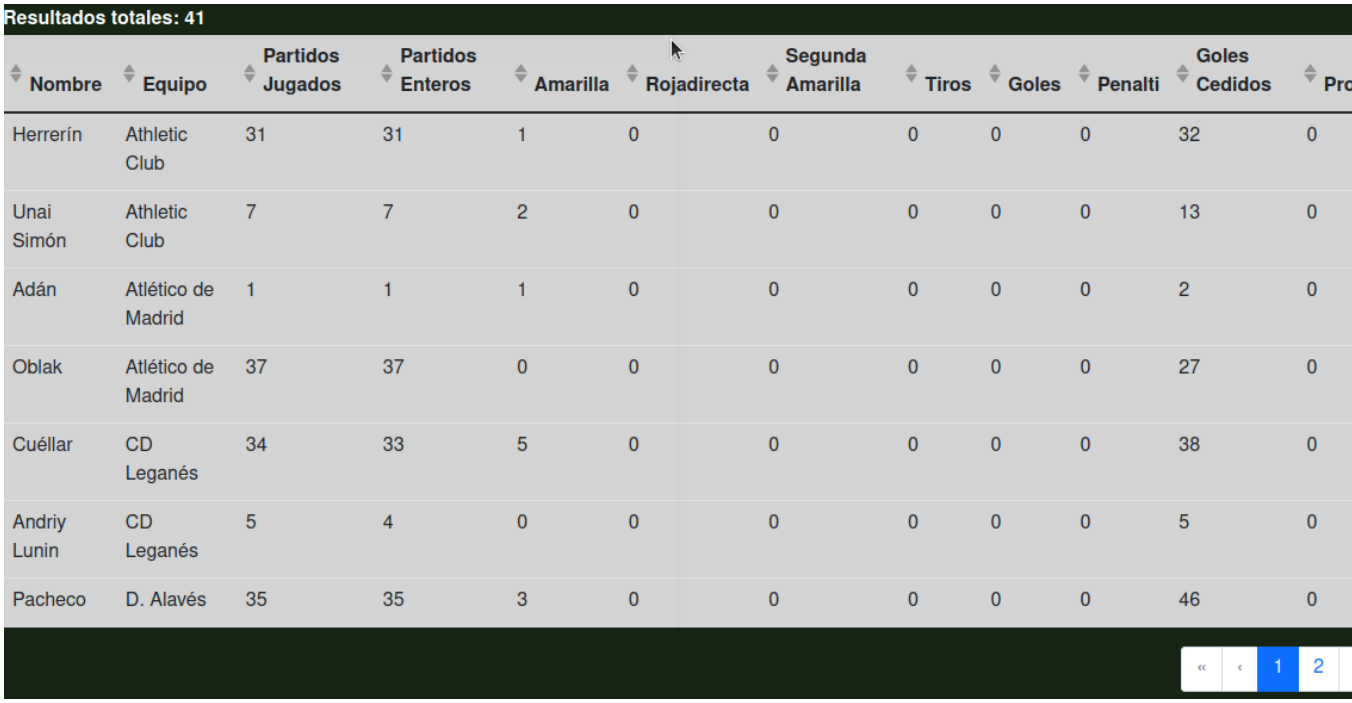
La aplicación consta de los siguientes filtros:

- **Posición** del jugador
- **Equipo** del mismo
- **Partidos Jugados y Partidos Enteros** entre 0 y la jornada actual (como estoy usando una base de datos de otros años es el máximo 38)
- **Amarillas, Rojas Directas y Dobles Amarillas**, en todos los casos el número máximo de tarjetas que la persona acepta que sus jugadores tengan.
- **Tiros** realizados, da igual que sean a puerta o no.
- **Goles** marcados, algo vital para la puntuación de los fantasy.
- **Penaltis** separados de los propios goles al ser un gol más “seguro”
- **Goles en propia puerta,** muy perjudiciales si los haces para un fantasy.
- **Goles recibidos mientras estás en el campo**, como medida de cómo de buenos son los porteros.
- **Segadas** para evaluar defensas y medios
- **Intercepciones** para evaluar defensas
- **Recuperaciones de balón** para evaluar defensas y medios
- **Despejes** para evaluar defensas
- **Dribles** para evaluar medios y delanteros

Después de los filtros encontraremos 3 botones.

- **Buscar**, nos permitirá buscar los jugadores según los filtros que hayamos aplicado.

● **Reiniciar**, para reiniciar todos los filtros, se colocarán por defecto.
● **Recomendar**, después de utilizar la aplicación varias veces y observar diferentes jugadores la aplicación podrá hacer una recomendación al usuario basándose en sus búsquedas anteriores, el funcionamiento se explicará más adelante en el apartado de explicación de algoritmos.


Resultados totales: 41

Nombre	Equipo	Partidos Jugados	Partidos Enteros	Amarilla	Rojadirecta	Segunda Amarilla	Tiros	Goles	Penalti	Goles Cedidos	Pro
Herrerín	Athletic Club	31	31	1	0	0	0	0	0	32	0
Unai Simón	Athletic Club	7	7	2	0	0	0	0	0	13	0
Adán	Atlético de Madrid	1	1	1	0	0	0	0	0	2	0
Oblak	Atlético de Madrid	37	37	0	0	0	0	0	0	27	0
Cuéllar	CD Leganés	34	33	5	0	0	0	0	0	38	0
Andriy Lunin	CD Leganés	5	4	0	0	0	0	0	0	5	0
Pacheco	D. Alavés	35	35	3	0	0	0	0	0	46	0

« ‹ 1 2

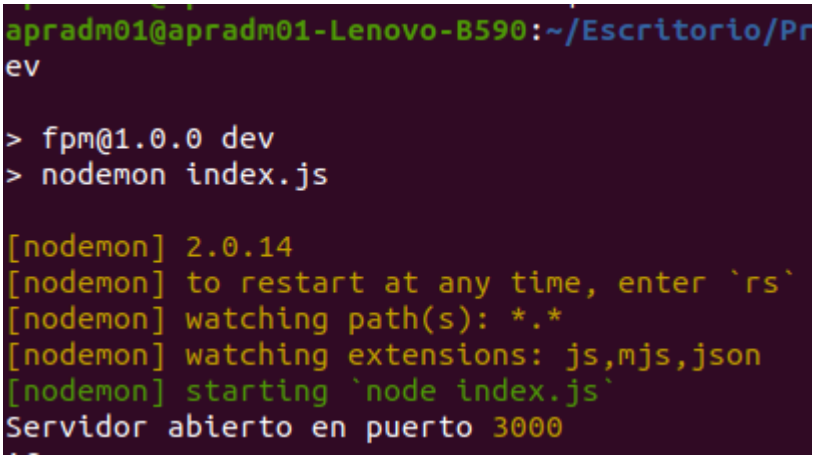
Después de darle al botón de buscar sale una tabla con todos los resultados, pudiendo filtrarlos según cualquier característica clickando sobre la misma. Pulsando sobre los jugadores se podría ir a su perfil.

Por último los jugadores recomendados del sistema, aparecen con una breve descripción de ellos mismos y una opción para ir al perfil.

Backend

En el backend vamos a realizar todo el apartado de lógica de la aplicación y el acceso a la base de datos.

Se ha configurado para que el backend esté escuchando las peticiones que se envían desde el frontend en el **puerto 3000** como podemos ver en la siguiente imagen:



Y aquí configuramos para que escuche en el puerto indicado.

```
11
12 const neo4j = require('./conexionDB.js');
13
14 // Configuraciones
15
16 app.set('port', 3000);
17
```

Por lo que todas las solicitudes tendrán que estar dirigidas hacía este puerto, para que sean recibidas correctamente y se realicen las diferentes operaciones. Todas las llamadas que se realizan desde el frontend están utilizando el método **POST,** este método permite enviar la información sin que el usuario la vea, ya que no se muestra en la URL como con el método GET sino que se almacena en el cuerpo de la petición.

Por lo que en la aplicación tenemos diferentes rutas, todas con este método pero cada una con funciones diferentes como veremos a continuación, aunque la estructura en todos es muy parecida.

En primer lugar tenemos la ruta de /buscarJugador, ésta es llamada cuando se

pulsa el botón de buscar en la sección de Búsqueda de jugadores.

Lo primero que vemos son las diferentes variables que corresponden a los

```
app.post('/buscarJugador', (req, res) => {

    // LEEMOS LOS VALORES

    var posicion = req.body.posicion;
    var equipo = req.body.equipo;
    var partidosJugados = req.body.partidosJugados;
    var partidosEnteros = req.body.partidosEnteros;
    var amarilla = req.body.amarilla;
    var rojadirecta = req.body.rojadirecta;
    var segundaAmarilla = req.body.segundaAmarilla;
    var tiros = req.body.tiros;
    var goles = req.body.goles;
    var penalti = req.body.penalti;
    var propiapuerta = req.body.propiapuerta;
    var golesCedidos = req.body.golesCedidos;
    var tackle = req.body.tackle;
    var inteceptions = req.body.inteceptions;
    var recoveries = req.body.recoveries;
    var clearances = req.body.clearances;
    var dribles = req.body.dribles;
```

diferentes filtros que se han seleccionado. A continuación se crea la petición a la base de datos.

```
var query = 'MATCH (j:Jugador) ';

query += 'WHERE ';

// POSICION
query += 'j.posicion = \''+posicion+ '\' ';

// PARTIDOS JUGADOS
query += ' AND j.partidosJugados >= '+partidosJugados+' ';

// EQUIPO
if(equipo != 'Empty') {
query += ' AND j.equipo = \''+equipo+ '\' ';
}

// PARTIDOS ENTEROS
query += ' AND j.partidosEnteros >= '+partidosEnteros+' ';

 // AMARILLAS
query += ' AND j.amarilla <= '+amarilla+' ';

// ROJA DIRECTA
query += ' AND j.rojadirecta <= '+rojadirecta+' ';

// SEGUNDA AMARILLA
query += ' AND j.segundaAmarilla <= '+segundaAmarilla+' ';
```

Por último recogemos y enviamos la información de estos jugadores para que el usuario pueda verla.

```
results.records.forEach(function(record) {

    var jugador = {

        // CARACTERÍSTICAS DE LOS JUGADORES

        nombre: record._fields[0].properties.nombre,
        equipo: record._fields[0].properties.equipo,
        partidosJugados: record._fields[0].properties.partidosJugados,
        partidosEnteros: record._fields[0].properties.partidosEnteros,
        amarilla: record._fields[0].properties.amarilla,
        rojadirecta: record._fields[0].properties.rojadirecta,
        segundaAmarilla: record._fields[0].properties.segundaAmarilla,
        tiros: record._fields[0].properties.tiros,
        goles: record._fields[0].properties.goles,
        penalti: record._fields[0].properties.penalti,
        propiapuerta: record._fields[0].properties.propiapuerta,
        golesCedidos: record._fields[0].properties.golesCedidos,
        tackle: record._fields[0].properties.tackle,
        inteceptions: record._fields[0].properties.inteceptions,
        recoveries: record._fields[0].properties.recoveries,
        clearances: record._fields[0].properties.clearances,
        dribles: record._fields[0].properties.dribles
    };
    jugadores.push(jugador);
})
```

Otra ruta que tenemos es /recomendarSimilares, que entra en juego cuando le damos al botón de recomendar en la sección de Búsqueda de jugadores. La estructura es muy parecida a la ruta anterior, primero obtenemos las variables para configurar la petición y luego devolvemos la lista de jugadores. En esta parte está el primero de nuestros algoritmos.

```
app.post('/infoJugador', (req, res) => {

    // OBTENEMOS LOS VALORES DE LA PETICIÓN
    var jugador = req.body.nombre;
    var usuario = req.body.usuario;
    var posicion = req.body.posicion;
```

```
// OBTENEMOS LOS VALORES DE LA PETICIÓN
var jugador = req.body.nombre;
var usuario = req.body.usuario;
var posicion = req.body.posicion;

// BUSCAMOS EL USUARIO
var queryVisitaPerfil = 'MATCH (u:Usuario { nombre: \''+usuario+'\' }) ';

// BUSCAMOS EL JUGADOR
queryVisitaPerfil += 'MATCH (j:Jugador { nombre: \''+jugador+'\' }) ';

// CREAMOS RELACIÓN
queryVisitaPerfil += 'MERGE (u)-[v:VISITA_PERFIL]->(j) ';

// CREA LA RELACIÓN SI NO EXISTE
queryVisitaPerfil += 'ON CREATE SET v.veces = 1 ';

// ACTUALIZA LA RELACIÓN
queryVisitaPerfil += 'ON MATCH SET v.veces = v.veces + 1;';

// PROCESAMOS LA PETICIÓN
neo4j.run(queryVisitaPerfil)
```

Lo primero que hacemos es crear unas variables que almacenarán el jugador que queremos ver el perfil, el nombre del usuario que va a ver el perfil y la posición que juega este jugador.

```
// CONEXIÓN PARA VISITAR EL PERFIL DEL JUGADOR
var queryVerPerfil = 'MATCH (j:Jugador { nombre: \''+jugador+'\'}) RETURN j;';
```

Luego creamos una petición a la base de datos que creará una relación entre el usuario y el jugador con VISITA_PERFIL**,** si esta relación ya está creada se aumentará un contador de veces que se ha visitado el perfil de este jugador. Una vez hecho esto, crearemos otra petición que nos devuelva la información del jugador que queremos ojear.

Y por último, crearemos otra petición buscando jugadores similares al que vamos a ojear.

Ruta de /borrarBusquedas, esta ruta es llamada cuando pulsamos el botón de eliminar favoritos desde el perfil del usuario.
Primero guardamos en una variable el nombre del usuario al que le vamos a borrar sus favoritos, y después hacemos la petición borrando la relación entre VISITA_PERFIL y LIKES. Una vez hecho se informa al usuario de que ha sido realizado exitosamente.

```
app.post('/borrarBusquedas', (req, res) => {

    var usuario = req.body.usuario;
    var query = 'MATCH (u:Usuario { nombre: \''+usuario+'\' })-[v:VISITA_PERFIL]->(j:Jug
    query += 'DELETE v '
    query += 'WITH u ';
    query += 'MATCH (u)-[l:LIKES]->(j:Jugador) ';
    query += 'DELETE l;'
    neo4j.run(query)

    .then(results => {
        res.json( { correcto: true } );
    })
    .catch(error => {
        console.log(error);
    });
});
```

Ruta de /addFavorito, esta ruta se llama cuando pulsamos el botón de mostrar favoritos desde el perfil del usuario. Después de obtener el nombre del usuario y del jugador marcado se crea la petición para crear la relación LIKES entre ellos

```
app.post('/addFavorito', (req, res) => {
    var usuario = req.body.usuario;
    var jugador = req.body.jugador;
    var query = 'MATCH (u:Usuario { nombre: \''+usuario+'\' }) ';
    query += 'MATCH (j:Jugador { nombre: \''+jugador+'\' }) ';
    query += 'MERGE (u)-[l:LIKES]->(j) RETURN l;';
    neo4j.run(query)

    .then(results => {
        res.json( { correcto: true } );
    })
    .catch(error => {
        console.log(error);
    });
});
```

Base de datos

La base de datos inicial, utilizada en la aplicación ha sido obtenida en la página de kaggle y es “La Liga Player Stats 2019/20”. Ésta base de datos incluye a todos los jugadores que participaron en la misma ese año y sus contribuciones al equipo. Es de destacar que se intentó encontrar un dataset de este año actual, pero no se ha encontrado uno a mano cómodo con la información que necesito (aunque sé como conseguirlo, véase Mejoras Futuras).

Todas las características que estamos usando en la aplicación se mencionaron ya en la sección de Frontend, pero también hemos añadido algunas que diferencian a los jugadores (los números son dependientes de la jornada de la liga):

- **Goleador:**

Cuando el jugador ha marcado más de 20 goles.

● **Tirador de Penaltis:**
Cuando el jugador ha marcado más de 2 goles de penalti.

● **Porterazo:**
Cuando el jugador no ha encajado más de 30 goles, es portero y ha disputado un mínimo de 10 partidos.

● **Corta Jugadas:**
Cuando un jugador ha realizado más de 30 segadas y es defensa.

● **Regateador:**
Cuando el jugador ha realizado más de 30 dribles y es mediocentro.

● **Propenso a tarjetas:**
Cuando el jugador tiene más de 8 tarjetas amarillas.

● **Titular:**
Ha empezado en el campo la mayoría de partidos.

● **Regular:**
Ha jugado todos los minutos la mayoría de partidos.

● **Sustituto:**
Ha salido del banquillo o le han cambiado la mayoría de partidos.

Estas características ayudarán a la hora de implantar los algoritmos de recomendación del sistema, que se explicará su funcionamiento más adelante.

Como nuestra base de datos está en un archivo.csv lo primero que debemos hacer es exportar la información para crear la nueva base de datos en cypher, en donde creamos los siguientes nodos:

- **Nodo (Jugador)** este nodo será el principal de la aplicación ya que contendrá los atributos anteriormente definidos, su propiedad principal, la que lo identifique y lo diferencie con el resto será el nombre.

- **Nodo (Caracteristica)** este nodo contendrá diferentes características de los jugadores basándonos en los atributos que tenemos, facilitando así la recomendación de jugadores. Las diferentes características que existen fueron definidas anteriormente.

- **Nodo (Usuario)** este nodo contendrá a los usuarios que estarán usando la aplicación, se crearán diferentes relaciones con el **nodo Jugador** para los sistemas de recomendación de la aplicación.

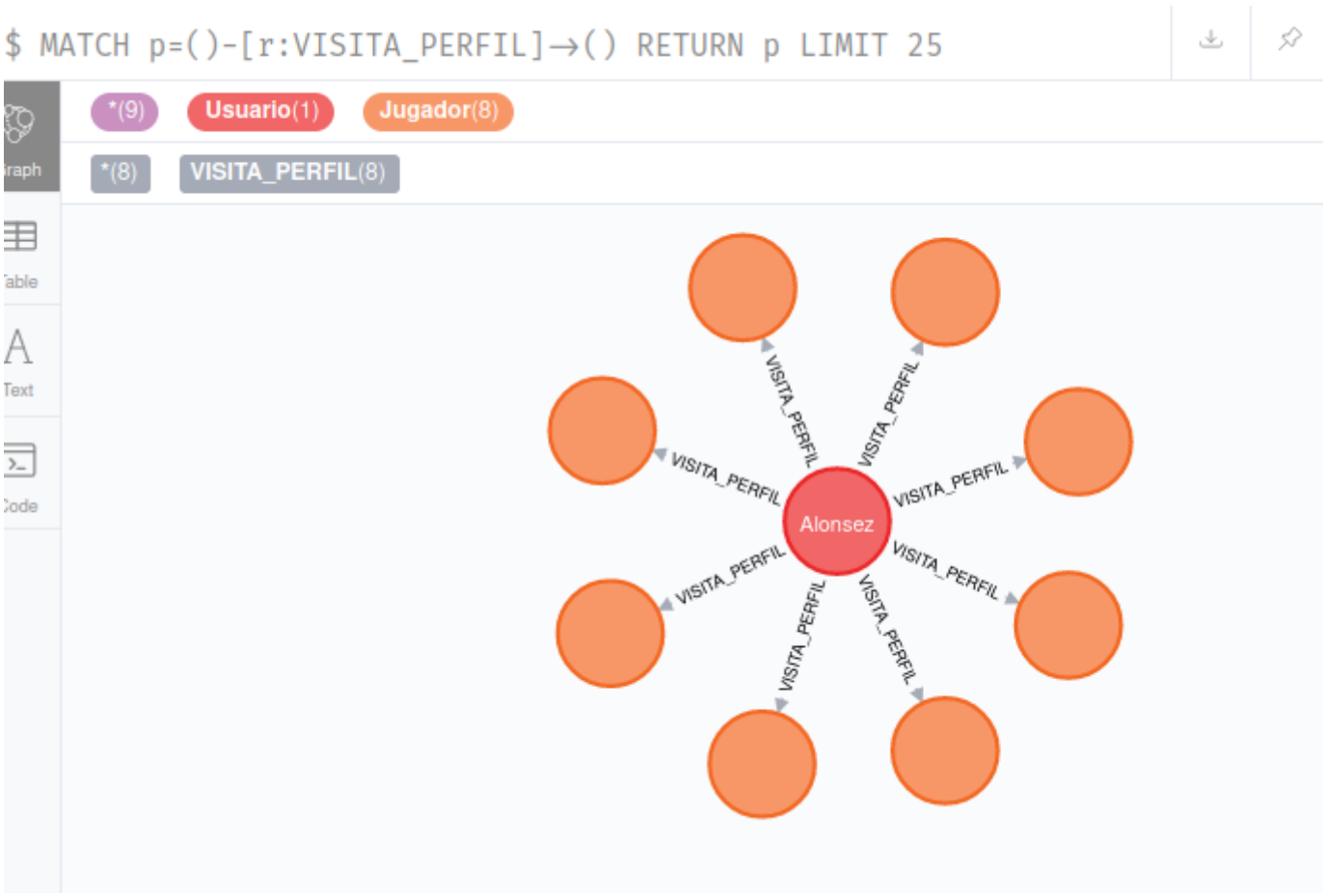
Y existirán las siguientes relaciones, que unirán los nodos y darán sentido a la base de datos:

- **Relación (VISITA_PERFIL)** entre el usuario y el jugador, esta relación también almacenará la cantidad de veces que el usuario visita el perfil de un determinado jugador, siendo muy útil para luego realizar recomendaciones.

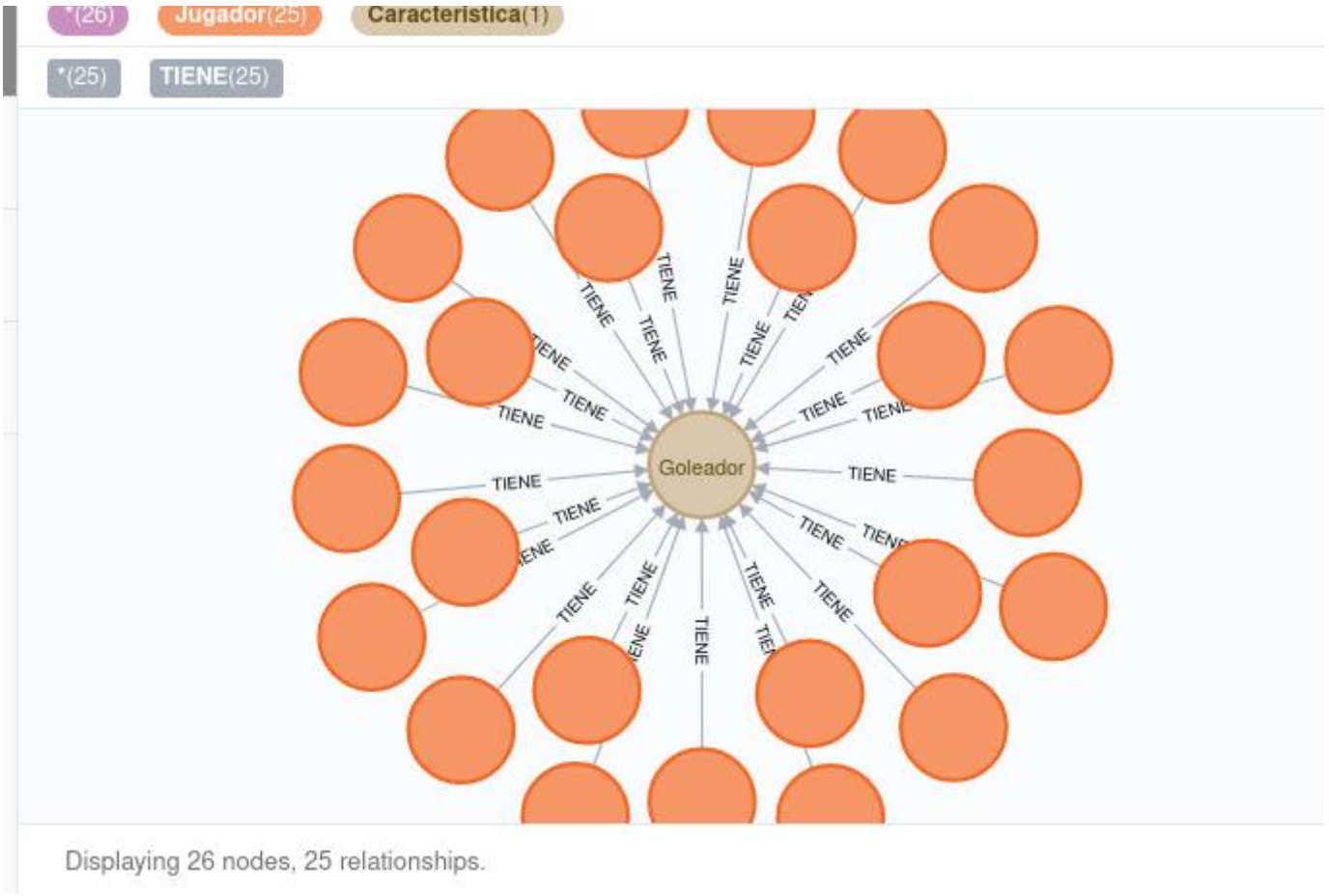
- **Relación (LIKES)** entre el usuario y el jugador, esta relación también almacenará cuando un usuario está interesado en un jugador y quiere guardarlo en una selección de favoritos, después los jugadores almacenados aquí aparecerán en el perfil del usuario.

- **Relación (TIENE)** entre el jugador y la característica, cuando un jugador tiene los atributos necesarios para tener una característica específica se creará esta relación.

Algunos ejemplos de los nodos con las relaciones: Visitas de Alonsez.



Relación Tiene con una de las características:



Explicación de los algoritmos

Aquí se explicarán de forma detallada los diferentes algoritmos que se han utilizado para los sistemas de recomendación que existen en la aplicación, pero lo primero que debemos hacer es instalar la librería de Neo4J Graph Data Science (gds) para poder trabajar con ellos, la debemos descargar desde la página de descargas de Neo4j.

Cuando esté descargada, para instalarla debemos añadirla en la siguiente ruta /var/lib/neo4j/plugins y una vez añadida reiniciamos el servicio de neo4j para aplicar los cambios con el comando service neo4j restart.

La aplicación tiene un buscador donde se aplican diferentes filtros para que los usuarios puedan hacer una búsqueda inicial de qué tipo de jugador está interesado, esto ayudará luego a que el sistema pueda realizar recomendaciones. Ambos algoritmos de recomendación están basados en la similitud entre los diferentes jugadores para hacer sugerencias.

El 1º algoritmo que se va a explicar es el que nos dará los jugadores similares al entrar al perfil de un jugador, este algoritmo está basado en el algoritmo de similitud de Jaccard, que mide la semejanza entre conjuntos y utiliza la siguiente fórmula:

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|} = \frac{|A \cap B|}{|A| + |B| - |A \cap B|}$$

Hemos modificado un poco el algoritmo para darle prioridad absoluta a la posición del jugador, pero una vez esto se mirará la similitud entre jugadores con los nodos Característica en común (a mayor cantidad en común y menos distintos mejor).

```
queryRecomendados = 'MATCH (j1:Jugador {nombre: \''+jugador.nombre+'\'})-[:TIENE]->(skill1) ';

yRecomendados += 'WITH j1, collect(id(skill1)) AS j1Skill ';
yRecomendados += 'MATCH (j2:Jugador) WHERE j2.posicion = \''+posicion+ '\' ';

yRecomendados += 'MATCH (j2)-[:TIENE]->(skill2) WHERE j1 <> j2 ';
yRecomendados += 'WITH j2, gds.alpha.similarity.jaccard(j1Skill, collect(id(skill2))) AS similitud ';
yRecomendados += 'RETURN j2 ORDER BY similitud DESC LIMIT 3;';
```

Finalmente se retornan los 3 resultados más relevantes.

El 2º algoritmo es el que entrará en acción cuando se pulse el botón de recomendar. Por ende, requiere primero haber buscado jugadores de forma normal para poder utilizarse.

Este algoritmo es más complejo que el anterior ya que para cada posición se definen cuales son los atributos más importantes, para compararlos con los demás jugadores y usará la distancia euclídea para determinar la similitud de los jugadores.

La distancia euclídea utiliza la siguiente fórmula:

$$d_{(p1,p2)} = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

Esto nos permitirá calcular la similitud entre los conjuntos de datos, primero se verá cuáles son los atributos utilizados para calcular la recomendación para cada posición y su explicación de la elección de esos atributos.

Para entender un poco las elecciones vamos a tener en cuenta ciertas características:

- En Mister (el Fantasy que yo conozco más) la mayoría de jugadores se valoran específicamente cuanto más, mejor. Da igual que falles 7 segadas y causes que te marquen 5 goles: si tienes muchos números positivos tienes una buena puntuación. Esto es muy evidente con equipos como el Real Madrid, que aunque ganan la mayoría de partidos sus jugadores no puntúan demasiado bien, especialmente sabiendo que son los más caros de conseguir. Esto es especialmente verdad con los goles: los goles en delanteros dan una bonificación de +3 a lo que puntúan, +4 a los mediocentros y +5 a defensas y porteros. Por eso es valioso ser el que tira los penaltis, porque es puntuación asegurada.
- Aunque parezca que el factor monetario es importante, este está muy sobrevalorado: tras unas pocas jornadas el dinero sobra dado que todas estas páginas dan como prioridad la satisfacción del usuario de irse consiguiendo jugadores según salen. Es más, ya en la primera jornada tienes dinero más que suficiente para conseguir buenos jugadores (Como referencia en Mister empiezas con 50 millones menos lo que valga tu equipo inicial generado aleatoriamente, que suele girar en torno a 20-25 millones).
- Del mismo modo que ir a por goles y lo más posible es lo importante, evitar a toda costa tarjetas rojas y que a tu portero le marquen gol también es vital: si a tu portero le marcan es muy probable que como mucho puntúe 3 puntos (siendo

8 lo “normal”), y si recibes una tarjeta roja no solo te pierdes el siguiente partido, lo cual fastidia tu disponibilidad, sino que en la propia jornada pierdes 6 puntos de sanción. Si tu jugador recibe una roja es a veces hasta mejor no llevar jugador y recibir “solo” un -4.

Con esto en mente hemos puesto estas características evaluando cada posición.

Portero:

- Partidos Jugados
- Partidos Completos
- Rojas Directas
- Goles en propia puerta
- Goles encajados

Defensa:

- Partidos Jugados
- Partidos Completos
- Rojas Directas
- Goles en propia puerta
- Amarillas
- Expulsión por doble amarilla
- Segadas
- Intercepciones
- Despejes
- Recuperaciones

Mediocentros:

- Partidos Jugados
- Partidos Completos
- Rojas Directas
- Goles en propia puerta
- Amarillas
- Expulsión por doble amarilla
- Segadas
- Goles
- Goles de Penalti
- Recuperaciones
- Dribles

Delanteros:

- Partidos Jugados
- Partidos Completos
- Rojas Directas
- Segunda Amarilla
- Goles

- Penaltis

Ahora que ya conocemos los atributos que se utilizarán para cada posición vamos a ver cómo funciona el algoritmo internamente.

```
var query = 'MATCH (j:Jugador) WHERE j.posicion = \''+posicion+ '\' ';
query += 'MATCH (u:Usuario)-[v:VISITA_PERFIL]->(j) WHERE u.nombre = \''+usuario+'\' ';
query += 'RETURN j.nombre ORDER BY v.veces DESC LIMIT 1;'
```

Primero necesitamos asegurarnos de que se han buscado jugadores anteriormente, y hacer una lista con las mismas, ordenado por número de veces.

Cuando conseguimos a los jugadores buscamos similares, y si los hay empezamos nuestro algoritmo, dependiendo de la posición del jugador.

```
if(posicion == 'Goalkeeper') { // PORTERO
    queryRecomendados += 'WITH j1, j2, \
    gds.alpha.similarity.euclideanDistance([j1.partidosJugados, j1.partidosEnteros, j1
    [j2.partidosJugados, j2.partidosEnteros, j2.rojadirecta, j2.propiapuerta, j2.goles

} else if(posicion == 'Defender') { // DEFENSA
    queryRecomendados += 'WITH j1, j2, \
    gds.alpha.similarity.euclideanDistance([j1.partidosJugados, j1.partidosEnteros, j1
        [j2.partidosJugados, j2.partidosEnteros, j2.rojadirecta, j2.propiapuerta, j2.a

} else if(posicion == 'Midfielder') { // MEDIOCENTRO
    queryRecomendados += 'WITH j1, j2, \
    gds.alpha.similarity.euclideanDistance([j1.partidosJugados, j1.partidosEnteros, j1
        [j2.partidosJugados, j2.partidosEnteros, j2.rojadirecta, j2.propiapuerta, j2.a

} else if(posicion == 'Forward') { // DELANTERO
    queryRecomendados += 'WITH j1, j2, \
    gds.alpha.similarity.euclideanDistance([j1.partidosJugados, j1.partidosEnteros, j1
        [j2.partidosJugados, j2.partidosEnteros, j2.rojadirecta, j2.segundaAmarilla, j
```

Por último ordenamos la lista y la enviamos al FrontEnd para enseñarla al usuario.

DAFO

El análisis DAFO se trata de una herramienta que permite evaluar de manera interna y externa las debilidades, amenazas, fortalezas y oportunidades de un proyecto y diseñar una estrategia para afrontar su futuro.

Consta de un análisis interno que se compone de las debilidades y fortalezas, donde se estudian las variables de producción, marketing, organización, recursos

humanos o personal y finanzas.

Y de un análisis externo que se compone de las amenazas y oportunidades, donde se estudian factores que no dependen directamente del proyecto pero que le afectan, donde entran variables como el mercado, el sector, la competencia y el entorno.

- **Debilidades:**
  La principal debilidad de la aplicación son los datos, ya que para crear un sistema de recomendación de fichajes avanzado se deberían tener información actualizada de los jugadores, actualizandolos cada vez que se use la aplicación ya que así tendríamos una mejor visión del estado actual del jugador en ese momento, no sólo basándonos en características que han sido definidas anteriormente según su trayectoria. Lo óptimo sería una combinación de datos actualizados y características del jugador.

- **Fortalezas:**
  La facilidad de uso es algo importante a tener en cuenta. Cuando un usuario se mete en la aplicación es cuestión de que seleccione lo que quiere y le de a buscar, y si le interesa puede ver jugadores similares. El no depender de dinero para poder enseñar jugadores también facilita que el usuario confíe más en el juicio de la aplicación, al evitar la falacia de “es muy barato es malo” o “es muy caro seguro que es buenísimo”.

- **Amenazas:**
  El ser una página externa a la posible página de Fantasy es una amenaza seria: si esas mismas páginas facilitan su propia base de datos para analizar cómo son los jugadores y adaptarte a ello, o si compran a alguien para que lo hagan por ellos podríamos perder buena parte del tráfico web.

- **Oportunidades:**
  Yendo un poco en línea con las amenazas, si somos nosotros quienes pactamos con las compañías de Fantasy para meter nuestra base de datos integrada en la misma podríamos obtener una ventaja competitiva con nuestra competencia.
  Aunque el fútbol es un mercado muy visto, el mundo de los Fantasy no tanto: es probable que las páginas que se asienten ahora tengan un muy buen futuro.

Problemas y soluciones en el desarrollo

Problema 1: En mi ordenador original no tenía acceso a root, lo cual impedía instalar los programas necesarios para realizar el proyecto. Aparte en su momento carecía de muchos conocimientos sobre programación en Windows.

Solución: Se ha optado por instalar Ubuntu versión 20.04 en mi ordenador aparte, fragmentando el disco. Desde aquí podía crear mi propio usuario principal.

Problema 2: En la propia base de datos había varios valores imposibles de leer al tener espacios en su nombre (tarjetas amarillas).

Solución: Se ha añadido de forma manual el carácter _ en las cabeceras de la base de datos usada.

Problema 3: La versión de Java instalada en su momento (JAVA SE 9) no permitía el arranque del proyecto.

Solución: Se instaló JAVA SE 11, y cuando seguía existiendo el problema anterior, se eliminó completamente JAVA SE 9 y se puso el 11 como la versión predeterminada.

Problema 4: Con un proyecto descargado de clase para hacer pruebas y usar como base, no podía replicar su funcionamiento del frontend, al haber un error de inicialización.

Solución: export NODE_OPTIONS=--openssl-legacy-provider antes de iniciar el FRONTEND

Problema 5: Neo4j no dejaba realizar ningún tipo de operación inicialmente, al por error en la primera sesión no pedir autenticación.

Solución: La solución más simple: reinstalar Neo4j y asegurarse de poner una contraseña.

Problema 6: En las peticiones a Neo4j, específicamente con los equipos, había problemas a la hora de reconocer qué era nombre y qué era parte de Neo4j.

Solución: Este problema era causado al leer lo pedido: se leía del FrontEnd la palabra correcta pero era más de un nombre (Real Madrid) asique identificaba Madrid como algo que no debía existir. La solución es poner \ a ambos lados para que reconozca la cadena \Real Madrid\ como una sola.

Problema 7: Los atributos como propiedades del nodo Jugador dificultaban la implementación de los algoritmos para los sistemas de recomendación.

Solución: Se decidió crear otro nodo llamado Característica que cada jugador tuviera diferentes cualidades basándonos en los atributos iniciales.

Problema 8: Aunque creía que recordaba bien Vue de las prácticas de empresa me encontré con la realidad de que no sabía casi ni entender lo que estaba leyendo

Solución: Una serie de tutoriales de vue, obteniendo al menos un conocimiento básico de lo mismo.

Problema 9: La base de datos usada no es ideal: los datos son de la temporada 2019/2020.

“Solución”: Se empezó un proyecto personal para sacar una base de datos actualizada a día de hoy. La idea era pasar la ubicación de la base de datos a un excel de google drive, con los datos de la liga actualizados. Para ello

primero se tenían que tener los datos hasta entonces, para lo cual se había pensado introducirlos a mano de distintas webs que tienen los mismos pero no tienen forma de usarlos (la página de datos de SofaScore es bastante buena, si tan solo se pudiese conseguir). Una vez conseguidos los datos a día de hoy se prepararía un script que actualizaría los datos del excel según fuesen ocurriendo las jornadas.

Esta idea aunque prometedora me resultó imposible de aplicar en el trabajo, principalmente porque no fui capaz de adaptar un documento en google drive al trabajo en el portátil.

Mejoras futuras

La aplicación actualmente es bastante completa para el tiempo de desarrollo que se ha tenido, pero a lo largo del avance han surgido ideas para mejorarla los diferentes sistemas, pero la complejidad de algunas y la limitación de tiempo, no ha sido posible implementarlas. Algunas de las ideas son las siguientes.

1. Como ya he mencionado en el problema 9, conseguir una base de datos actual y actualizable en tiempo real sería el paso lógico y directo a este proyecto. Aparte de eso también habría que cambiar algunos números de la base de datos (como estoy lidiando con números estáticos no me he preocupado, pero por ejemplo el número de partidos jugados debería ser un porcentaje).

2. Añadir más atributos a los jugadores para identificarlos, no sólo de habilidad sino también en el sentido de estilo para darle mejor presentación a la página. (Imágenes de los jugadores, nacionalidad, edad actualizada, etc)

3. Un sistema de inicio de sesión: es una mejora que para este trabajo de por sí no ha sido prioritaria pero es necesario para poder tener varias personas usando la página. Dicho esto por lo que he investigado el proceso, sobre todo importándolo de proyectos externos, es más saber dónde buscar las referencias que hacerlo.

4. Usar diferentes algoritmos para calcular los jugadores. Cuando pienso en un Recomendador en mi mente tengo dos cosas: que me de más o menos lo que yo quiero y que no me de siempre lo mismo. El primer punto con la página actual lo conseguimos bastante bien, pero con solo 2 algoritmos que tenemos implementados en la página las recomendaciones tienden a ser “predecibles”. Me he planteado por encima la idea de tener varios algoritmos preparados y que con un generador aleatorio se elija cuál usar en ese momento, pero es algo que no se ha tenido tiempo.

Lecciones aprendidas

Esta asignatura es notablemente distinta al resto de las demás, dándote más

libertad para enfrentarte a un reto en concreto, sin necesidad de revisiones semanales o estudio para el examen de turno. Esto ha permitido aprender ciertos conocimientos que serían imposibles en cualquier otra asignatura. A continuación pongo varios ejemplos.

-Técnicas de resolución de problemas. Algo importante que ha definido en general este trabajo ha sido el tener que resolver diferentes obstáculos en la programación del mismo. Este trabajo ha permitido la creación de un procedimiento para resolver estos bloqueos (informarse del error, si no hay solución inmediata hay que revisar el propio método y las características derivadas. Si nada de esto funciona lo mejor es de hecho dar un paso atrás y hacer otra cosa para no cerrarse, y una vez estés listo de nuevo a revisar otra vez).

-Backend. Esta no es la primera asignatura que explica sobre la parte de atrás de la aplicación, pero si es la primera en la que de hecho yo he tenido que trabajar en el mismo (por puras coincidencias o no se ha hecho prácticas o hicieron la parte mis compañeros de equipo en su día). En esta asignatura he tenido que lidiar con la lógica de la aplicación y sobre todo aprender cómo se conecta el Frontend y el Backend.

-Neo4j, tecnología utilizada por primera vez en esta asignatura. Como persona que ha sufrido mucho utilizando SQL me he quedado gratamente sorprendido cuando he sido capaz de utilizar Neo4j sin demasiado esfuerzo. Las peticiones son rápidas de hacer y el que puedas meter la tecnología de grafos para hacer las recomendaciones casi desde el propio Neo4j es fantástico. Mi único arrepentimiento es no haber podido usar esto antes.

Bibliografía y referencias

- Base de datos:
  https://www.kaggle.com/datasets/thegreatcoder/laliga-player-stats

- Cursos de Neo4j:
  https://graphacademy.neo4j.com/

- Ubuntu:
  https://ubuntu.com/download

- Node.js
  https://nodejs.org/es/

- Neo4j
  https://neo4j.com

- Vue.js
  https://vuejs.org/

- Bootstrap
  https://getbootstrap.com/

- Font Awesome
  https://fontawesome.com/

- GitHub
  https://github.com/

- Manual de ayuda de Cypher
  https://neo4j.com/docs/cypher-manual/current/

- Página del Fantasy seleccionado como base del trabajo
  https://mister.mundodeportivo.com/

- Recursos para Bootstrap y Vue
  https://bootstrap-vue.org

- Instalación librerías en NEO4J
  https://neo4j.com/docs/graph-data-science/current/installation/neo4j-server/

- Librería gds de NEO4J
  https://neo4j.com/download-center/#algorithms

- Jaccard Similarity algorithm
  https://neo4j.com/docs/graph-data-science/current/alpha-algorithms/jaccard/

- Euclidean Distance
  https://neo4j.com/docs/graph-data-science/current/alpha-algorithms/euclidean /

- Reglas Mister (para los atributos importantes de cada posición)
  https://help.playmister.com/article/142-reglas-del-juego

- Patrón de diseño: Modelo-Vista-Controlador
  http://codingornot.com/mvc-modelo-vista-controlador-que-es-y-para-que-sirve /mvc-modelo-vista-controlador

- Explicación de DAFO:
  https://www.cerem.es/blog/claves-para-hacer-un-buen-dafo-o-foda