universidad de león

Escuela de Ingenierías

Industrial, Informática y Aeroespacial

GRADO EN INGENIERÍA INFORMÁTICA

SISTEMA RECOMENDADOR

NBA FANTAZY

17 DE DICIEMBRE 2023

Autor: Aitor Vizcaya Ardura

Contenido

1 INTRODUCCION

En esta memoria se proporcionará una explicación detallada acerca del desarrollo, funcionamiento, tecnologías empleadas, problemas abordados, posibles mejoras futuras, desafíos surgidos durante el progreso de la aplicación y las soluciones adoptadas.

La aplicación en cuestión se llama NBA Fantazy y es una página web orientada a la NBA que puede servir como herramienta de asistencia para cualquier participante de juegos de estilo Fantasy, simuladores de crear tu quinteto inicial ideal y competir contra tus amigos u otras personas, e incluso podría llegarse a utilizar en apuestas deportivas del estilo que jugador será el MVP del campeonato, el mejor jugador defensivo, rookie del año etc, es decir, apuestas que se basan especialmente en las estadísticas puras de un jugador.

La función principal de la aplicación será orientar a los usuarios sobre qué jugadores podrían fichar para su Fantasy o, como se ha dicho anteriormente, a cuáles apostar en determinados ámbitos.

Una vez entrado al sitio web, este contará con un Inicio de Sesión y luego un Recomendador que contará con varias secciones, siendo la más destacada el apartado de Jugadores Recomendados Para Ti, además de un buscador de Jugadores con diversos filtros para facilitar la búsqueda, una función de buscar Jugadores Similares a un Jugador determinado y también una lista con tus Jugadores Favoritos, la cuál será muy importante ya que sobre ella se basará los Recomendados Para Ti. También cabe destacar, que se podrá visualizar los datos de un jugador pulsando sobre su botón de INFO. Esto es importante ya que el Buscador de Jugadores está simplificado con las estadísticas más importantes de los Jugadores como pueden ser puntos o asistencias, dejando de lado aquellas que son más profundas y suelen ser combinaciones de otras, como efectividad en tiros de campo, ya que al fin y al cabo, se querrá buscar jugadores simplemente.

Cabe destacar que la aplicación está diseñada para ser accesible para cualquier usuario, con una paleta de colores agradable y una estructura clara de navegación, sin importar sus conocimientos sobre la NBA

Además, se podría considerar la posibilidad de ampliar este proyecto en el futuro para incluir más información de otras temporadas de la NBA, e incluso poder dar un salto a ligas domésticas como podrían ser la ACB o la Euroliga.

2 DESCRIPCIÓN DEL PROBLEMA

El baloncesto, como deporte de gran popularidad a nivel mundial, ha generado un entorno competitivo y apasionante tanto en las canchas reales como en el ámbito virtual de las ligas Fantasy. Sin embargo, este entorno no está exento de desafíos y obstáculos que los participantes enfrentan al intentar conformar equipos exitosos en sus ligas Fantasy. A continuación, se detallan algunos de los problemas recurrentes que motivan la necesidad de una aplicación especializada en recomendar jugadores para mejorar la experiencia de los usuarios.

- **Abundancia de Datos**: La vasta cantidad de datos estadísticos, históricos y de rendimiento de los jugadores de baloncesto puede resultar abrumadora para los participantes de las ligas Fantasy. Se presenta un desafío para los usuarios analizar eficientemente esta información para tomar decisiones informadas sobre sus elecciones de jugadores.
- **Complejidad en la Selección de Jugadores**: Elegir el conjunto óptimo de jugadores para formar un equipo competitivo implica considerar diversas variables, como lesiones, rachas de rendimiento, enfrentamientos y estrategias de juego. En nuestro caso solo nos basaremos en estadísticas puras y duras. Este proceso puede ser complicado y consumir tiempo, especialmente para aquellos usuarios que no poseen un profundo conocimiento del baloncesto y las ligas Fantasy.

- **Cambio Dinámico en el Rendimiento de Jugadores**: Lesiones repentinas, cambios en el estado físico de los jugadores y otras variables impredecibles pueden tener un impacto significativo en el rendimiento de un jugador. Los participantes necesitan estar al tanto de estas dinámicas para ajustar rápidamente sus estrategias, una tarea desafiante sin el apoyo de herramientas especializadas.
- **Necesidad de Recomendaciones Personalizadas**: Cada usuario tiene un enfoque único y preferencias al construir su equipo Fantasy. Una aplicación efectiva debería ser capaz de proporcionar recomendaciones personalizadas que se alineen con el estilo y objetivos individuales de cada participante.

La existencia de estos desafíos resalta la importancia de una solución tecnológica que simplifique y optimice el proceso de selección de jugadores en las ligas Fantasy de baloncesto, proporcionando a los usuarios recomendaciones inteligentes y adaptadas a sus necesidades específicas.

3 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, por ello se ha elegido el modelo-vista-controlador (MVC), un patrón arquitectónico utilizado en el diseño de software, especialmente en el desarrollo de aplicaciones web. El objetivo principal del patrón MVC es separar la lógica de presentación de la lógica de negocio y la gestión de eventos. Esta separación facilita la modificación y mantenimiento del código, ya que los distintos componentes del sistema tienen responsabilidades bien definidas.

A continuación, se explica cada componente del patrón MVC:

3.1 Modelo

3.1.1 Responsabilidad

Representa la lógica de negocio y los datos de la aplicación. Esto incluye la gestión de datos, reglas de negocio, lógica de cálculos y cualquier otra manipulación de datos.

3.1.2 Características

- El modelo es independiente de la interfaz de usuario. No tiene conocimiento directo de cómo se presenta la información al usuario.
- Responde a las solicitudes del controlador y notifica cualquier cambio a las vistas asociadas.

- Puede haber varios modelos en una aplicación, cada uno representando una parte específica de la lógica de negocio.

3.2 Vista

3.2.1 Responsabilidad

Es responsable de la presentación de la información al usuario y de la interacción con él. Muestra los datos provenientes del modelo y envía las interacciones del usuario al controlador.

3.2.2 Características

- La vista tiene conocimiento del modelo, pero no de la lógica de negocio subyacente.
- Puede haber múltiples vistas que presentan los datos de manera diferente, según las necesidades del usuario.
- Las vistas reciben actualizaciones desde el modelo cuando los datos cambian y también envían comandos al controlador cuando el usuario interactúa.

3.3 Controlador

3.3.1 Responsabilidad

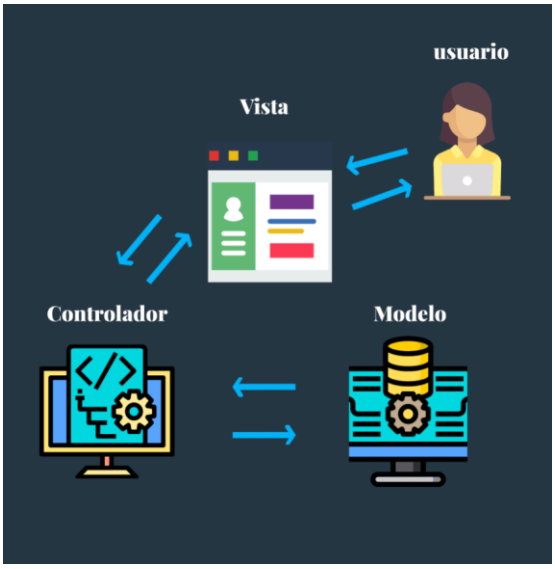
Maneja las interacciones del usuario y actúa como intermediario entre el modelo y la vista. Recibe las entradas del usuario, actualiza el modelo según sea necesario y actualiza la vista para reflejar los cambios.

3.3.2 Características

- El controlador interpreta las acciones del usuario y traduce esas acciones en operaciones en el modelo.
- No tiene conocimiento de cómo se presenta la información al usuario, pero sí sabe qué vista se debe actualizar.
- Puede haber varios controladores en una aplicación, cada uno manejando diferentes partes de la interfaz de usuario o funcionalidades específicas.

3.4 Flujo de Trabajo en el Patrón MVC

1. El usuario interactúa con la interfaz de usuario, generando una acción.
2. El controlador recibe la acción y actualiza el modelo según sea necesario.
3. El modelo notifica a las vistas asociadas sobre cualquier cambio.
4. Las vistas consultan el modelo y actualizan la interfaz de usuario según los cambios.
5. Este ciclo se repite cuando el usuario realiza nuevas interacciones.



El desarrollo de la aplicación se ha realizado en un Windows 10, y el código en Visual Studio Code.

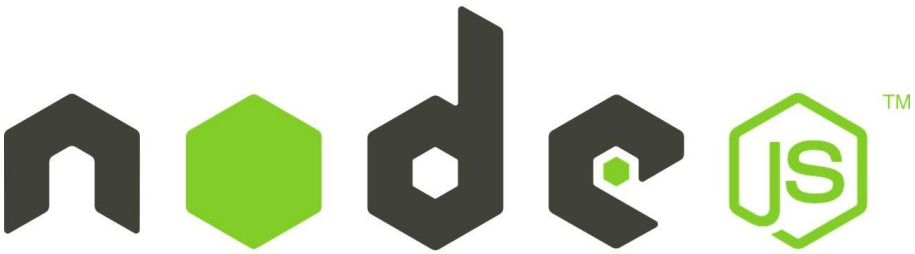
3.5 Backend

Se ha empleado Node.js para el backend, una tecnología que ya he utilizado en otras asignaturas.

Para iniciar un proyecto con Node.js, solo hay que descargarlo desde la página oficial, instalarlo y luego utilizar el comando 'npm init' en la terminal.

A continuación, el sistema solicitará información relacionada con nuestro proyecto. Una vez completado, podemos configurarlo para que funcione como un servidor web, y luego iniciarlo mediante 'npm start'.

En este punto, podemos agregar diferentes módulos a nuestro proyecto para facilitar el desarrollo de la aplicación. En mi caso, he empleado los siguientes; Nodemon, Cors, Express y Neo4j-driver.



3.6 Frontend

Para el desarrollo del frontend se ha utilizado el framework React. React es una biblioteca de JavaScript creada por Facebook para construir interfaces de usuario interactivas. Su enfoque principal es la creación de componentes reutilizables, bloques de construcción independientes que representan partes específicas de la interfaz de usuario.

Utiliza JSX, una extensión de la sintaxis de JavaScript que permite escribir código similar a XML/HTML dentro de archivos JavaScript. Esto facilita la creación y comprensión de la estructura de los componentes.

React emplea un Virtual DOM, una representación en memoria del DOM real, para mejorar la eficiencia de las actualizaciones en la interfaz de usuario. En lugar de actualizar directamente el DOM, React compara las diferencias entre el Virtual DOM y el DOM real antes de realizar cambios, lo que minimiza la manipulación directa del DOM y mejora el rendimiento.

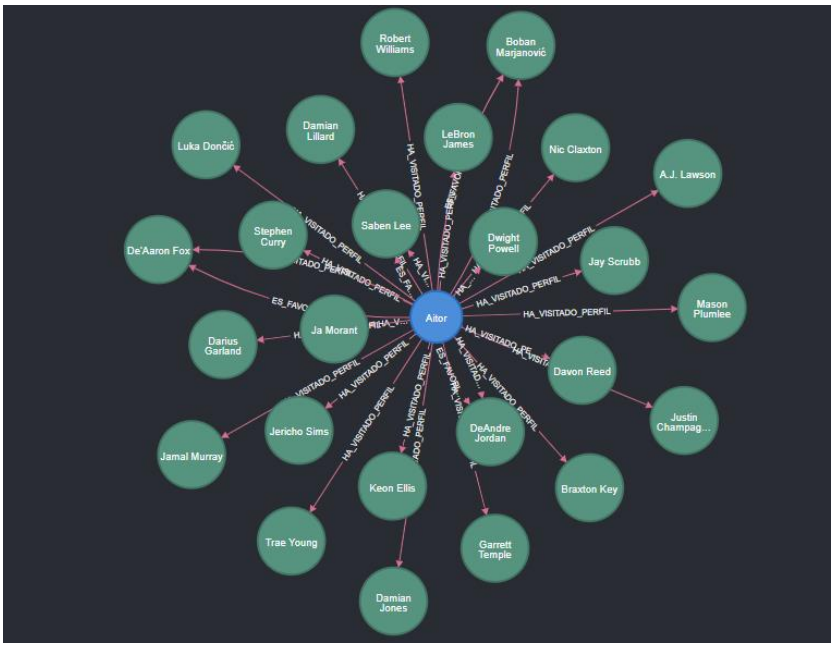
Además, React sigue un flujo unidireccional de datos, lo que significa que los datos fluyen en una dirección, desde el componente principal hacia los componentes secundarios. Esto facilita el mantenimiento del estado de la aplicación y reduce la posibilidad de errores.

Por último al contar con una comunidad activa de desarrolladores y una amplia variedad de recursos, incluyendo tutoriales, documentación oficial y bibliotecas de terceros, he podido solucionar varias dudas que tenido durante el desarrollo de la aplicación.

3.7 Base De Datos

Como base de datos se utilizará Neo4j.Neo4j es una base de datos orientada a grafos que organiza la información en nodos, relaciones y propiedades. Utiliza el lenguaje de consulta Cypher y mantiene transacciones ACID para garantizar la integridad de los datos.

Es escalable, flexible y se adapta bien a escenarios donde las relaciones son clave, como redes sociales o análisis de red. Neo4j tiene una comunidad activa, soporte empresarial y se utiliza en diversas aplicaciones, desde redes sociales hasta análisis de fraude y sistemas de recomendación. Su capacidad para gestionar datos altamente interconectados lo hace valioso en entornos donde las relaciones son fundamentales.



3.8 Almacenamiento

Para el almacenamiento de todo el proyecto se utilizará Github, porque facilita la colaboración en el desarrollo de software, controla versiones de código, organiza proyectos, permite trabajo distribuido, posibilita despliegues continuos, actúa como portafolio profesional, integra herramientas externas, ofrece seguridad y promueve la colaboración abierta, impulsando la eficiencia y éxito en el desarrollo de software.

4 APLICACIÓN

4.1 Frontend

La aplicación se divide en 2 partes:

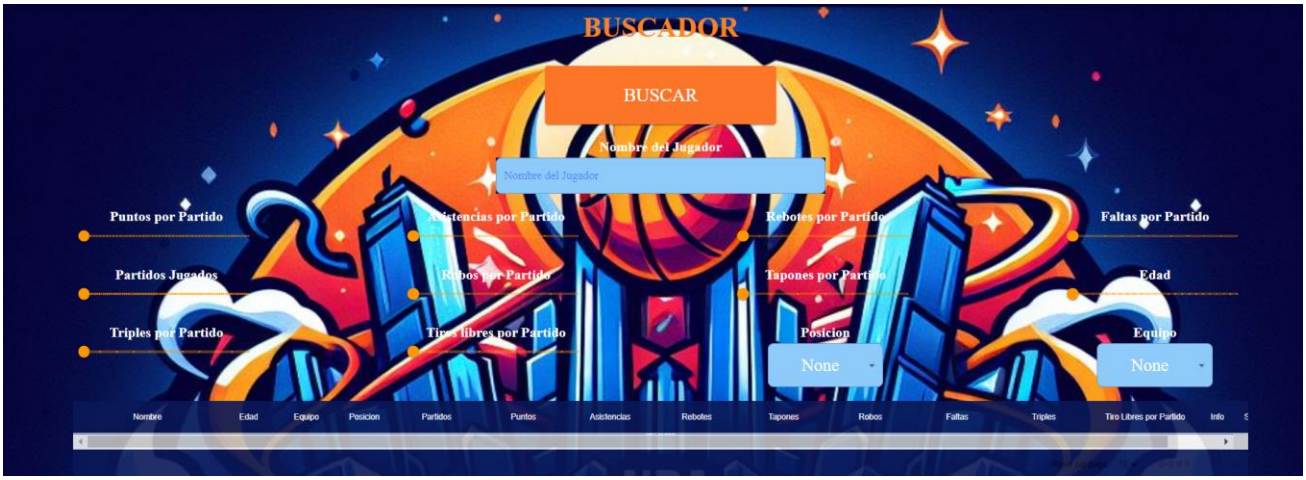
4.1.1 Inicio de Sesión

En la primera pestaña que veremos al entrar será un Login que nos permitirá iniciar sesión o crear un usuario para poder acceder a la aplicación web.

4.1.2 Home

Una vez iniciada sesión podremos ver la página principal del Usuario, es decir, el Recomendador.

Lo primero que veremos será un buscador con filtros para buscar aquellos jugadores que queramos.



Este buscador cuenta con filtros como Nombre, Puntos por Partido, Asistencias por Partido, entre otros, pero que no entraré en detalle sobre que significa cada uno porque queda suficientemente claro que es cada filtro.

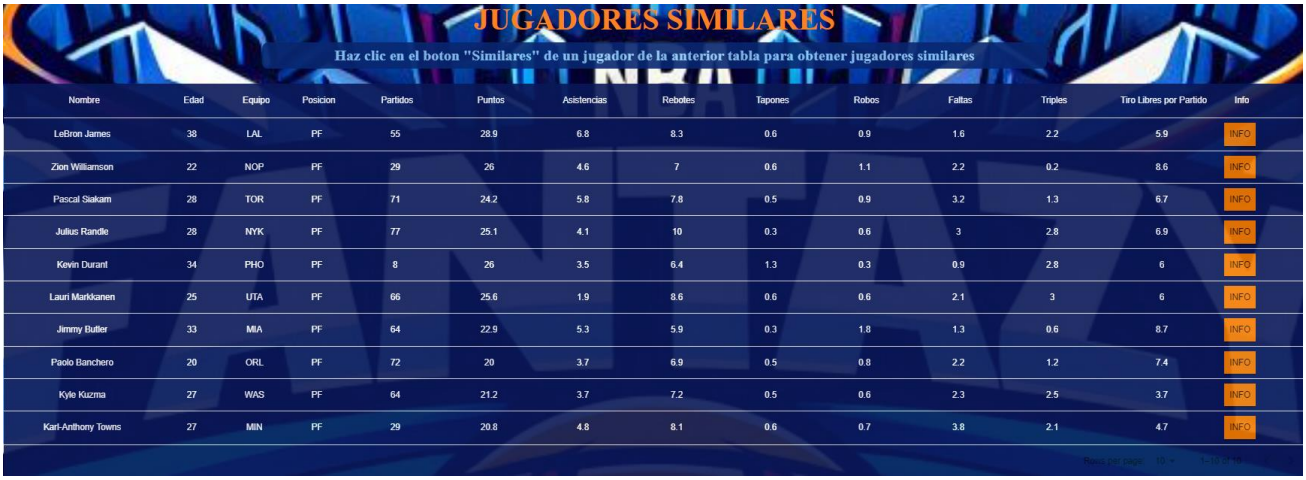
Por ejemplo si decidiéramos buscar un jugador con un mínimo de 10 puntos por partido y 5 asistencias nos saldrían los siguientes resultados:

Nombre	Edad	Equipo	Posicion	Partidos	Puntos	Asistencias	Rebotes	Tapones	Robos	Faltas	Triples	Tiro Libres por Partido	Info
Nikola Jokić	27	DEN	C	69	24.5	9.8	11.8	0.7	1.3	2.5	0.8	6	INFO
Domantas Sabonis	26	SAC	C	79	19.1	7.3	12.3	0.5	0.8	3.5	0.4	5.5	INFO
Giannis Antetokounmpo	28	MIL	PF	63	31.1	5.7	11.8	0.8	0.8	3.1	0.7	12.3	INFO
Jimmy Butler	33	MIA	PF	64	22.9	5.3	5.9	0.3	1.8	1.3	0.6	8.7	INFO
Kris Dunn	28	UTA	PG	22	13.2	5.6	4.5	0.5	1.1	2.6	0.8	2.4	INFO
Markelle Fultz	24	ORL	PG	60	14	5.7	3.9	0.4	1.5	2.2	0.5	2.4	INFO
De'Aaron Fox	25	SAC	PG	73	25	6.1	4.2	0.3	1.1	2.4	1.6	6	INFO
Kyrie Irving	30	DAL	SG	20	27	6	5	0.6	1.3	2.8	2.9	4.7	INFO
Shai Gilgeous-Alexander	24	OKC	PG	68	31.4	5.5	4.8	1	1.6	2.8	0.9	10.9	INFO
Bradley Beal	29	WAS	SG	50	23.2	5.4	3.9	0.7	0.9	2.1	1.6	4.6	INFO

Si pulsamos sobre el botón de info de cada jugador(esto es así para todas las tablas) se nos mostraría toda la información detallada sobre dicho jugador:

Cada vez que pulsemos sobre la info de un jugador se nos actualizará el contador de veces visitado dicho jugador, y además, en esta pestaña podremos marcar el jugador como favorito pulsando el icono de corazón en la parte inferior derecha.

Si no decidiéramos ver la info del jugador, podríamos pulsar en el botón de Similares para cargar en la siguiente tabla aquellos jugadores más similares en esa posición con dicho jugador. Para Giannis Antetokoumpo serían los siguientes:



Si comparamos a Giannis con los jugadores similares, podemos ver que son bastantes similares en cuestión de estadísticas, ya que los jugadores que se han mostrado son los 10 jugadores más parecidos a él mediante la utilización de la distancia euclídea para comparar las estadísticas de todos los jugadores con Giannis, eligiendo aquellos 10 con menor distancia.

En la siguiente tabla podremos ver aquellos jugadores que el sistema nos recomienda según nuestras visitas y jugadores favoritos. Por ejemplo estos serían algunos jugadores recomendados para mí:

Nombre	Edad	Equipo	Posicion	Partidos	Puntos	Asistencias	Rebotes	Tapones	Robos	Faltas	Triples	Tiro Libres por Partido	Info
JD Davison	20	BOS	SG	12	1.6	0.9	0.8	0.2	0.2	0.4	0.2	0.2	INFO
Tyrese Martin	23	ATL	SG	16	1.3	0.1	0.8	0	0.1	0.1	0.1	0.1	INFO
JD Davison	20	BOS	SG	12	1.6	0.9	0.8	0.2	0.2	0.4	0.2	0.2	INFO
Dalen Terry	20	CHI	SG	38	2.2	0.6	1	0.1	0.3	0.6	0.2	0.6	INFO
Boban Marjanović	34	HOU	C	31	3.3	0.3	1.9	0.1	0.2	0.3	0	0.9	INFO
Bruno Fernando	24	ATL	C	8	3.4	0.1	1.9	0.4	0	0.8	0	0.8	INFO
Wendell Moore Jr.	21	MIN	SG	29	1.4	0.6	0.6	0.2	0.3	0.4	0.1	0.2	INFO
Garrett Temple	36	NOP	SG	25	2	0.5	0.7	0.1	0.4	0.6	0.4	0.2	INFO
Richaun Holmes	29	SAC	C	42	3.1	0.2	1.9	0.3	0.1	1.2	0.1	0.5	INFO
Dwight Powell	31	DAL	C	76	6.7	0.9	4.1	0.3	0.6	2.8	0	2.2	INFO

Aquí podemos ver Pivots y escoltas, que concuerdan con algunas ultimas visitas y jugadores favoritos como los que tenía añadidos, buscando aquellos jugadores que más se podrían parecer a mis últimas búsquedas:

Nombre	Edad	Equipo	Posicion	Partidos	Puntos	Asistencias	Rebotes	Tapones	Robos	Faltas	Triples	Tiro Libres por Partido	Info
Giannis Antetokounmpo	28	MIL	PF	63	31.1	5.7	11.8	0.8	0.8	3.1	0.7	12.3	INFO
Duncan Robinson	28	MIA	SF	42	6.4	1.1	1.6	0	0.3	1.8	1.5	0.8	INFO
De'Aaron Fox	25	SAC	PG	73	25	6.1	4.2	0.3	1.1	2.4	1.6	6	INFO
Wendell Moore Jr.	21	MIN	SG	29	1.4	0.6	0.6	0.2	0.3	0.4	0.1	0.2	INFO
Boban Marjanović	34	HOU	C	31	3.3	0.3	1.9	0.1	0.2	0.3	0	0.9	INFO
Saben Lee	23	PHI	PG	2	3	1	0	0	0.5	0.5	0	0	INFO
DeAndre Jordan	34	DEN	C	39	5.1	0.9	5.2	0.6	0.3	1.8	0	1.2	INFO

4.2 Backend

En este apartado mostraré las rutas del backend que nos permitirá obtener la información de neo4j.

Se ha configurado para que el backend esté escuchando las peticiones que se envían desde el frontend en el puerto 5000 como se puede ver en la siguiente imagen:

```
/**
 * Importamos la conexión a la base de datos
 */
const driver = require("./conexionDB");

/**
 * Configuramos el puerto del servidor
 */
app.set("port", 5000);
```

4.2.1 /inicioSesion

```
/**
 * Endpoint para el inicio de usuarios
 */
app.post("/inicioSesion", async (req, res) => {
  const { usuario, contrasenia } = req.body;

  const query = `MATCH (u:Usuario {nombre: $usuario, contrasenia: $contrasenia}) RETURN u`;

  let sesion = driver.session();

  await sesion
    .run(query, { usuario: usuario, contrasenia: contrasenia })
    .then((result) => {
      if (result.records.length === 0) {
        console.log("Usuario no encontrado");
        res.status(404).send({ message: "Usuario no encontrado" });
      } else {
        console.log("Usuario encontrado");
        res.status(200).send({ message: "Usuario encontrado" });
      }
    })
    .catch((error) => {
      console.error(error);
      res.status(500).send({ message: "Internal server error" });      You, el mes pasado • Añ
    })
    .then(async () => await sesion.close());
});
```

Este endpoint nos servirá para buscar si existe un usuario con dicho nombre y contraseña en el sistema, y si los encuentra permitirá al sistema iniciar la sesión.

4.2.2 /registro

```
/**
 * Endpoint para el registro de usuarios
 */
app.post("/registro", (req, res) => {
  const { usuario, contrasenia } = req.body;

  const query = `MERGE (u:Usuario {nombre: $usuario, contrasenia: $contrasenia}) RETURN u`;

  let sesion = driver.session();

  sesion
    .run(query, { usuario: usuario, contrasenia: contrasenia })
    .then((result) => {
      if (result.records.length === 0) {
        res.status(404).send({ message: "Usuario ya registrado" });
      } else {
        res.status(200).send({ message: "Usuario registrado" });
      }
    })
    .catch((error) => {
      console.error(error);
      res.status(500).send({ message: "Internal server error" });
    })
    .then(async () => await sesion.close());       You, hace 2 semanas • Cosas
});
```

Este endpoint sirve para registrar en neo4j un usuario con una contraseña determinada. Si ya existe un usuario con dicho nombre, el sistema avisará al usuario de que ese nombre de usuario no está disponible.

4.2.3 /jugadoresCar

```
/**
 * Endpoint para obtener todos los jugadores con un determinado número de puntos, rebotes y asistencias
 */
app.post("/jugadoresCar", (req, res) => {
  const {
    nombre,
    puntos,
    partidos,      You, la semana pasada • Busqueda con campos
    rebotes,
    asistencias,
    faltas,
    robos,
    triples,
    edad,
    tapones,
    posicion,
    equipo,
    tirosLibresPartido,
  } = req.body;

  let query = `MATCH (j:Jugador) WHERE j.puntosPartido >= $puntos AND j.partidosJugados >= $partidos AND j.rebotesTotalesPartido >= $rebotes AND j.asistenciasPartido >= $asistencias AND
  j.faltasPersonalesPartido >= $faltas AND j.robosPartido >= $robos AND j.triplesAcertadosPartido >= $triples AND j.edad >= $edad AND j.taponesPartido >= $tapones AND
  j.tirosLibresAcertadosPartido >= $tirosLibresPartido `;

  let json = {
    puntos: puntos,
    partidos: partidos,
    rebotes: rebotes,
    asistencias: asistencias,
    faltas: faltas,
    robos: robos,
    triples: triples,
    edad: edad,
    tapones: tapones,
    tirosLibresPartido: tirosLibresPartido,
  };
```

```
if (nombre !== "") {
  query += `AND j.nombre CONTAINS $nombre `;
  json.nombre = nombre;
}
if (equipo !== "") {
  query += `AND j.equipo CONTAINS $equipo `;
  json.equipo = equipo;
}
if (posicion !== "") {
  query += `AND j.posicion CONTAINS $posicion `;
  json.posicion = posicion;
}
query += `RETURN j,[(j)-[r:ES]->(n:Caracteristica) | n.valor] as caracteristicas`;

let sesion = driver.session();
//MIRAR PARAMETROS
sesion
  .run(query, json)
  .then((result) => {
    const jugadores = result.records.map((record) => {
      const jugador = record.get("j").properties;
      const caracteristicas = record.get("caracteristicas");
      jugador.caracteristicas = caracteristicas;
      return jugador;
    });
    // Modificar el parámetro deseado en todos los jugadores
    const jugadoresModificados = jugadores.map((jugador) => {
      jugador.edad = jugador.edad.low;
      jugador.partidosJugados = jugador.partidosJugados.low;
      jugador.partidosTitular = jugador.partidosTitular.low;
      return jugador;
    });
```

```
      const jugadoresTabla = jugadoresModificados.map((jugador, i) => ({
        id: i,
        nombre: jugador.nombre,
        puntos: jugador.puntosPartido,
        edad: jugador.edad,
        equipo: jugador.equipo,
        posicion: jugador.posicion,
        partidos: jugador.partidosJugados,
        asistencias: jugador.asistenciasPartido,
        rebotes: jugador.rebotesTotalesPartido,
        tapones: jugador.taponesPartido,
        robos: jugador.robosPartido,
        faltas: jugador.faltasPersonalesPartido,
        triples: jugador.triplesAcertadosPartido,
        tirosLibresPartido: jugador.tirosLibresAcertadosPartido,
      }));

      res.status(200).send({ jugadores: jugadoresModificados, jugadoresTabla });
    })
    .catch((error) => {
      console.error(error);
      res.status(500).send({ message: "Internal server error" });
    })
    .then(() => sesion.close());
});
```

Este endpoint permitirá devolver los jugadores en el sistema que cumplan los parámetros que se han introducido en los filtros.

4.2.4 /jugadoresSimilares/:nombre

```
app.get("/jugadoresSimilares/:nombre", async (req, res) => {      You, hace 4 días • Acabar similares
  const { nombre } = req.params;

  const query = `
    MATCH (j1:Jugador {nombre: $nombre})
    MATCH (j2:Jugador)
    WHERE j2 <> j1 AND j2.posicion = j1.posicion
    WITH j1, j2,
    gds.similarity.euclideanDistance([j1.asistenciasPartido, j1.canastasPartido, j1.efectividadEnTirosDeCampo, j1.faltasPersonalesPartido,  j1.perdidasPartido, j1.porcentajeDeTriple, j1.porcentajeTirosDe2Partido, j1.porcentajeTirosDeCampo, j1.porcentajeTiros
    , j1.puntosPartido, j1.rebotesDefensivosPartido, j1.rebotesOfensivosPartido, j1.rebotesTotalesPartido, j1.robosPartido, j1.taponesPartido, j1.tirosDe2AcertadosPartido, j1.tirosDe2Partido, j1.tirosLibresAcertadosPartido, j1.tirosLibresPartido, j1.tirosPor
      [j2.asistenciasPartido, j2.canastasPartido, j2.efectividadEnTirosDeCampo, j2.faltasPersonalesPartido,  j2.perdidasPartido, j2.porcentajeDeTriple, j2.porcentajeTirosDe2Partido, j2.porcentajeTirosDeCampo, j2.porcentajeTirosLibres
      , j2.puntosPartido, j2.rebotesDefensivosPartido, j2.rebotesOfensivosPartido, j2.rebotesTotalesPartido, j2.robosPartido, j2.taponesPartido, j2.tirosDe2AcertadosPartido, j2.tirosDe2Partido, j2.tirosLibresAcertadosPartido, j2.tirosLibresPartido, j2.tiro
    RETURN j2, distancia, [(j2)-[r:ES]->(n:Caracteristica) | n.valor] as caracteristicas
    ORDER BY distancia ASC LIMIT 10
  `;

  let sesion = driver.session();

  await sesion
    .run(query, { nombre: nombre })
    .then((result) => {
      const jugadores = result.records.map((record) => {
        const jugador = record.get("j2").properties;
        const caracteristicas = record.get("caracteristicas");
        jugador.caracteristicas = caracteristicas;
        return jugador;
      });
      const jugadoresModificados = jugadores.map((jugador) => {
        // Cambiar el parámetro deseado
        jugador.edad = jugador.edad.low;
        jugador.partidosJugados = jugador.partidosJugados.low;
        jugador.partidosTitular = jugador.partidosTitular.low;
        return jugador;
      });

      const jugadoresSimilaresTabla = jugadoresModificados.map(
        (jugador, i) => ({
          id: i,
          nombre: jugador.nombre,
          puntos: jugador.puntosPartido,
          edad: jugador.edad,
          equipo: jugador.equipo,
          posicion: jugador.posicion,
          partidos: jugador.partidosJugados,
          asistencias: jugador.asistenciasPartido,
          rebotes: jugador.rebotesTotalesPartido,
          tapones: jugador.taponesPartido,
          robos: jugador.robosPartido,
          faltas: jugador.faltasPersonalesPartido,
          triples: jugador.triplesAcertadosPartido,
          tirosLibresPartido: jugador.tirosLibresAcertadosPartido,
        })
      );
```

```
      res.status(200).send({
        jugadoresSimilares: jugadoresModificados,
        jugadoresSimilaresTabla,
      });
    })
    .catch((error) => {
      console.error(error);
      res.status(500).send({ message: "Internal server error" });
    })
    .then(() => sesion.close());
});
```

Este método será igual que el anterior pero solo buscará los jugadores similares a un jugador con un nombre determinado mediante el uso de la distancia euclídea de dichos jugadores.

4.2.5 /jugadoresRecomendados/:usuario

```
app.get("/jugadoresRecomendados/:usuario", async (req, res) => {
  const { usuario } = req.params;

  const query = `
  MATCH (u:Usuario {nombre: $usuario})-[:HA_VISITADO_PERFIL]->(j:Jugador)
  WITH COLLECT(j) AS jugadoresVisitados
  MATCH (u:Usuario {nombre: $usuario})-[:ES_FAVORITO]->(j:Jugador)
  WITH jugadoresVisitados, collect(j) AS jugadoresFavoritos
  WITH jugadoresVisitados + jugadoresFavoritos AS jugadoresEncontrados
  MATCH (j1:Jugador)
  WHERE j1 IN jugadoresEncontrados
  WITH jugadoresEncontrados, j1
  MATCH (j2:Jugador)
  WHERE j2 <> j1 AND j2.posicion = j1.posicion
  WITH j1, j2,
  gds.similarity.euclideanDistance([
    j1.asistenciasPartido, j1.canastasPartido, j1.efectividadEnTirosDeCampo, j1.faltasPersonalesPartido, j1.perdidasPartido, j1.porcentajeDeTriple, j1.porcentajeTirosDe2Partido, j1.porcentajeTirosDeCampo, j1.p
    j1.puntosPartido, j1.rebotesDefensivosPartido, j1.rebotesOfensivosPartido, j1.rebotesTotalesPartido, j1.robosPartido, j1.taponesPartido, j1.tirosDe2AcertadosPartido, j1.tirosDe2Partido, j1.tirosLibresAcert
  ], [
    j2.asistenciasPartido, j2.canastasPartido, j2.efectividadEnTirosDeCampo, j2.faltasPersonalesPartido, j2.perdidasPartido, j2.porcentajeDeTriple, j2.porcentajeTirosDe2Partido, j2.porcentajeTirosDeCampo, j2.p
    j2.puntosPartido, j2.rebotesDefensivosPartido, j2.rebotesOfensivosPartido, j2.rebotesTotalesPartido, j2.robosPartido, j2.taponesPartido, j2.tirosDe2AcertadosPartido, j2.tirosDe2Partido, j2.tirosLibresAcert
  ]) AS distanciadistancia
  RETURN j2, distancia,[(j2)-[r:ES]->(n:Caracteristica) | n.valor] as caracteristicas
  ORDER BY distancia ASC
  LIMIT 20`;

  let sesion = driver.session();

  await sesion
    .run(query, { usuario: usuario })
    .then((result) => {
      const jugadores = result.records.map((record) => {
        const jugador = record.get("j2").properties;
        const caracteristicas = record.get("caracteristicas");
        jugador.caracteristicas = caracteristicas;
        return jugador;
      });
      const jugadoresModificados = jugadores.map((jugador) => {
        // Cambiar el parámetro deseado
        jugador.edad = jugador.edad.low;
        jugador.partidosJugados = jugador.partidosJugados.low;
        jugador.partidosTitular = jugador.partidosTitular.low;
        return jugador;
      });
```

```
      const jugadoresRecomendadosTabla = jugadoresModificados.map(
        (jugador, i) => ({
          id: i,
          nombre: jugador.nombre,
          puntos: jugador.puntosPartido,
          edad: jugador.edad,
          equipo: jugador.equipo,
          posicion: jugador.posicion,
          partidos: jugador.partidosJugados,
          asistencias: jugador.asistenciasPartido,
          rebotes: jugador.rebotesTotalesPartido,
          tapones: jugador.taponesPartido,
          robos: jugador.robosPartido,
          faltas: jugador.faltasPersonalesPartido,
          triples: jugador.triplesAcertadosPartido,
          tirosLibresPartido: jugador.tirosLibresAcertadosPartido,
        })
      );

      res.status(200).send({
        jugadoresRecomendados: jugadoresModificados,
        jugadoresRecomendadosTabla,
      });
    })
    .catch((error) => {
      console.error(error);
      res.status(500).send({ message: "Internal server error" });
    })
    .finally(() => sesion.close());
});
```

En este endpoint se obtendrán los jugadores recomendados para un usuario según aquellos jugadores que haya visitado el usuario y sus favoritos.

4.2.6 /usuario/:nombre

```
app.get("/usuario/:nombre", async (req, res) => {        You, hace 2 sem
  const { nombre } = req.params;

  const query = `
    MATCH (u:Usuario {nombre: $nombre})
    RETURN u
  `;

  let sesion = driver.session();

  await sesion
    .run(query, { nombre: nombre })
    .then((result) => {
      const usuario = result.records.map(
        (record) => record.get("u").properties
      )[0]; // Obtener el primer elemento del array
      res.status(200).send(usuario);
    })
    .catch((error) => {
      console.error(error);
      res.status(500).send({ message: "Internal server error" });
    })
    .then(async () => await sesion.close());
});
```

Este endpoint devuelve la información de un usuario con un determinado nombre.

4.2.7 /visitarPerfil/:usuario

```
app.post("/visitarPerfil/:usuario", async (req, res) => {
  const { nombre } = req.body;
  const { usuario } = req.params;

  const query = `
    MATCH (u:Usuario {nombre: $usuario}), (j:Jugador {nombre: $jugador})
    MERGE (u)-[v:HA_VISITADO_PERFIL]->(j)
    ON CREATE SET v.numeroVisitas = 1
    ON MATCH SET v.numeroVisitas = v.numeroVisitas + 1
  `;

  let sesion = driver.session();

  await sesion
    .run(query, { usuario: usuario, jugador: nombre })
    .then(() => {
      res.status(200).send({ message: "Perfil visitado correctamente" });
    })
    .catch((error) => {
      console.error(error);
      res.status(500).send({ message: "Internal server error" });
    })
    .then(async () => await sesion.close());
});
```

Este endpoint servirá para actualizar el contador de visitas de un jugador para un usuario determinado.

4.2.8 /favoritos/:usuario

```
/**
 * Endpoint para obtener todos los jugadores favoritos de un usuario
 */
app.get("/favoritos/:usuario", async (req, res) => {
  const { usuario } = req.params;

  const query = `MATCH (u:Usuario {nombre: $usuario})-[:ES_FAVORITO]->(j:Jugador) RETURN j,[(j)-[r:ES]->(n:Caracteristica) | n.valor] as caracteristicas`;

  let sesion = driver.session();

  await sesion
    .run(query, { usuario: usuario })
    .then((result) => {
      const jugadores = result.records.map((record) => {
        const jugador = record.get("j").properties;
        const caracteristicas = record.get("caracteristicas");
        jugador.caracteristicas = caracteristicas;
        return jugador;
      });
      const jugadoresModificados = jugadores.map((jugador) => {
        // Cambiar el parámetro deseado
        jugador.edad = jugador.edad.low;
        jugador.partidosJugados = jugador.partidosJugados.low;
        jugador.partidosTitular = jugador.partidosTitular.low;
        return jugador;
      });

      const favoritosTabla = jugadoresModificados.map((jugador, i) => ({
        id: i,
        nombre: jugador.nombre,
        puntos: jugador.puntosPartido,
        edad: jugador.edad,
        equipo: jugador.equipo,
        posicion: jugador.posicion,
        partidos: jugador.partidosJugados,
        asistencias: jugador.asistenciasPartido,
        rebotes: jugador.rebotesTotalesPartido,
        tapones: jugador.taponesPartido,
        robos: jugador.robosPartido,
        faltas: jugador.faltasPersonalesPartido,
        triples: jugador.triplesAcertadosPartido,
        tirosLibresPartido: jugador.tirosLibresAcertadosPartido,
      }));
```

```
      res.status(200).send({
        favoritos: jugadoresModificados,
        favoritosTabla,
      });
    })
    .catch((error) => {
      console.error(error);
      res.status(500).send({ message: "Internal server error" });
    })
    .then(async () => await sesion.close());
});
```

Este endpoint sirve para devolver los jugadores favoritos de un usuario.

4.2.9 /favoritos/:usuario

```
/**
 * Endpoint para añadir un jugador a favoritos
 */
app.post("/favoritos/:usuario", async (req, res) => {
  const { nombre } = req.body;
  const { usuario } = req.params;

  const query = `
    MATCH (u:Usuario {nombre: $usuario}), (j:Jugador {nombre: $nombre})
    MERGE (u)-[:ES_FAVORITO]->(j)
  `;

  let sesion = driver.session();

  await sesion
    .run(query, { usuario: usuario, nombre: nombre })
    .then(() => {
      res.status(200).send({ message: "Jugador añadido a favoritos" });
    })
    .catch((error) => {
      console.error(error);
      res.status(500).send({ message: "Internal server error" });
    })
    .then(async () => await sesion.close());
});
```

Este endpoint sirve para marcar como favorito a un jugador.

4.2.10 /favoritos/:usuario/:nombre

```
/**
 * Endpoint para eliminar un jugador de favoritos
 */
app.delete("/favoritos/:usuario/:nombre", async (req, res) => {
  const { usuario, nombre } = req.params;

  const query = `
    MATCH (u:Usuario {nombre: $usuario})-[r:ES_FAVORITO]->(j:Jugador {nombre: $nombre})
    DELETE r
  `;

  let sesion = driver.session();

  await sesion
    .run(query, { usuario: usuario, nombre: nombre })
    .then(() => {
      res.status(200).send({ message: "Relación eliminada correctamente" });
    })
    .catch((error) => {
      console.error(error);
      res.status(500).send({ message: "Internal server error" });
    })
    .then(async () => await sesion.close());
});
```

Este endpoint sirve para eliminar como favorito a un jugador determinado.

4.2.11 /jugadores

```
/**
 * Endpoint para obtener todos los jugadores
 */
app.get("/jugadores", async (req, res) => {
  /*const query = "MATCH (j:Jugador) RETURN j";*/

  const query =
    "MATCH (j:Jugador) RETURN j, [(j)-[r:ES]->(n:Caracteristica) | n.valor] as caracteristicas";

  let sesion = driver.session();

  await sesion
    .run(query)       You, hace 4 semanas • Avanzando
    .then((result) => {
      const jugadores = result.records.map((record) => {
        const jugador = record.get("j").properties;
        const caracteristicas = record.get("caracteristicas");
        jugador.caracteristicas = caracteristicas;
        return jugador;
      });

      // Modificar el parámetro deseado en todos los jugadores
      const jugadoresModificados = jugadores.map((jugador) => {
        // Cambiar el parámetro deseado
        jugador.edad = jugador.edad.low;
        jugador.partidosJugados = jugador.partidosJugados.low;
        jugador.partidosTitular = jugador.partidosTitular.low;
        return jugador;
      });

      const jugadoresTabla = jugadoresModificados.map((jugador, i) => ({
        id: i,
        nombre: jugador.nombre,
        puntos: jugador.puntosPartido,
        edad: jugador.edad,
        equipo: jugador.equipo,
        posicion: jugador.posicion,
        partidos: jugador.partidosJugados,
        asistencias: jugador.asistenciasPartido,
        rebotes: jugador.rebotesTotalesPartido,
        tapones: jugador.taponesPartido,
        robos: jugador.robosPartido,
        faltas: jugador.faltasPersonalesPartido,
        triples: jugador.triplesAcertadosPartido,
        tirosLibresPartido: jugador.tirosLibresAcertadosPartido,
      }));
```

```
      res.status(200).send({ jugadores: jugadoresModificados, jugadoresTabla });
    })
    .catch((error) => {
      console.error(error);
      res.status(500).send({ message: "Internal server error" });
    })
    .then(async () => await sesion.close());
});
```

Por último este endpoint sirve para obtener todos los jugadores del sistema.

4.3 Neo4J

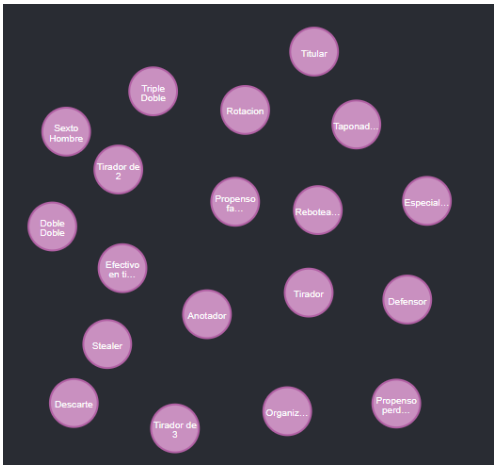
En este apartado hablaré sobre los nodos que se han creado para guardar los datos en neo4j.

4.3.1 Nodo de Jugadores

Este nodo servirá para guardar las estadísticas de los jugadores, las cuales son las que aparecen en la siguiente imagen:

4.3.2 Node de Características

En estos nodos se guardarán los valores de las características que poseerá cada jugador dependiendo sus estadísticas:

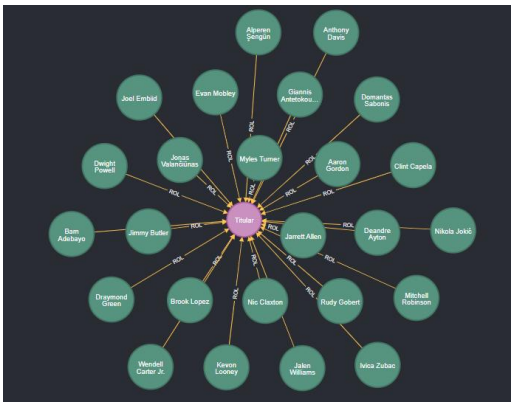


Estas características serán creadas de la siguiente manera siendo “j” un jugador:

- j.porcentajeTirosLibres > 0.70 AND c.valor = 'Especialista en tiros libres'
- j.efectividadEnTirosDeCampo > 0.6 AND c.valor = 'Efectivo en el tiro'
- j.taponesPartido > 1 AND c.valor = 'Taponador'
- j.rebotesTotalesPartido > 7 AND c.valor = 'Reboteador'
- j.robosPartido > 1 AND c.valor = 'Stealer'
- j.faltasPersonalesPartido > 2.75 AND c.valor = 'Propenso a hacer faltas'
- j.perdidasPartido > 2.5 AND c.valor = 'Propenso a perdidas'
- j.porcentajeTirosDe2Partido > 0.6 AND c.valor = 'Tirador de 2'
- j.porcentajeDeTriple > 0.4 AND c.valor = 'Tirador de 3'
- j.porcentajeDeTriple > 0.4 AND j.porcentajeTirosDe2Partido > 0.5 AND c.valor = 'Tirador'
- j.puntosPartido > 15 AND c.valor = 'Anotador'
- j.asistenciasPartido > 6 AND c.valor = 'Organizador' AND j.perdidasPartido < 7

4.3.3 Node de Rol

Este rol servirá para determinar que rol tiene el jugador en el equipo:



Los roles para los jugadores serán creados de la siguiente manera:

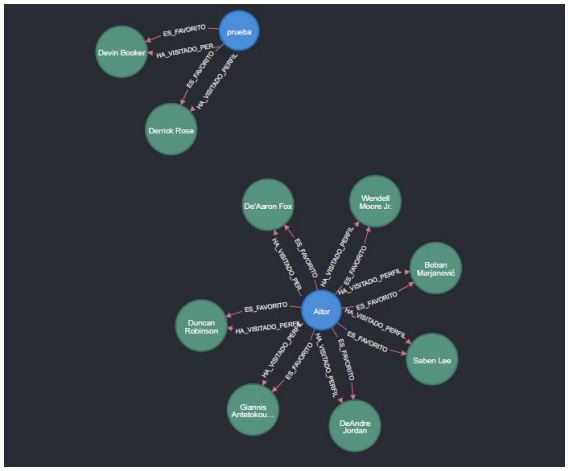
- j.partidosTitular > 50 AND c.valor = 'Titular'
- j.partidosTitular < 50 AND j.partidosTitular > 35 AND c.valor = 'Sexto Hombre'
- j.partidosTitular < 36 AND j.partidosTitular > 15 AND c.valor = 'Rotacion'
- j.partidosJugados < 10 AND c.valor = 'Descarte'

4.3.4 Nodo de Usuario

Este nodo servirá para almacenar los usuarios registrados en el sistema.

4.3.5 Relación ES_FAVORITO y HA_VISITADO_PERFIL

Esta relación permitirá crear una relación de los jugadores que son favoritos de un usuario y si ha sido visitado por este y cuántas veces.



4.3.6 Relación ES

Esta relación determina que característica tiene cada jugador.

5 ALGORITMO DE RECOMENDACIÓN

El algoritmo utilizado para la recomendación de jugadores similares a otros jugadores y los recomendados para un usuario en concreto es la distancia euclídea:

$$d_{12} = \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2 + (x_3 - y_3)^2},$$

Siendo X un jugador e Y otro jugador, siendo x1 una característica de un jugador e y1 la misma característica para otro jugador. Haciendo a la distancia euclídea de las características obtendremos unos valores que nos permitirán saber cuanto se parece un jugador con otro.

En código esta distancia euclídea se representaría de la siguiente forma:

```
WITH j1, j2,
gds.similarity.euclideanDistance([
  j1.asistenciasPartido, j1.canastasPartido, j1.efectividadEnTirosDeCampo, j1.faltasPersonalesPartido, j1.perdidasPartido, j1.porcentajeDeTriple, j1.porcentaj
  j1.puntosPartido, j1.rebotesDefensivosPartido, j1.rebotesOfensivosPartido, j1.rebotesTotalesPartido, j1.robosPartido, j1.taponesPartido, j1.tirosDe2Acertadc
], [
  j2.asistenciasPartido, j2.canastasPartido, j2.efectividadEnTirosDeCampo, j2.faltasPersonalesPartido, j2.perdidasPartido, j2.porcentajeDeTriple, j2.porcentaj
  j2.puntosPartido, j2.rebotesDefensivosPartido, j2.rebotesOfensivosPartido, j2.rebotesTotalesPartido, j2.robosPartido, j2.taponesPartido, j2.tirosDe2Acertadc
]) AS distanciaD
```

Siendo j1 y j2 2 jugadores distintos. Neo4j nos proporciona el uso de una función interna de distancia euclídea, por lo que nos ahorra el pesado hecho de hacer la fórmula anterior completa.

6 PROBLEMAS SURGIDOS

Durante el desarrollo del proyecto me han surgido esta serie de problemas:

- La conexión con Neo4j debe hacerse mediante un puerto del localhost, pero el puerto por defecto asociado a Neo4j ya estaba siendo ocupado en mi dispositivo, por lo que tarde un tiempo en darme cuenta de que no conseguía conectarme al neo4j porque ya estaba ocupado. La solución era cambiar el puerto por uno que estuviera libre.
- Neo4j en Javascript tiene problemas en la sincronización de las transacciones de las queries, por lo que tuve que implementar métodos de sincronización para evitar errores de conexión.

- La carga del archivo .csv donde se encuentra toda la base de datos me causó problemas ya tenía que ser importado dentro del propio Neo4j en una sección para imports, y no funcionaba rutas locales en el dispositivo.
- Por último, también tuve problemas con el uso de React ya que no me acordaba mucho de como se usaba, pero gracias a los cursos de la página oficial de React me fue más ameno volver a introducirme a este framework.

7 POSIBLES FUTURAS MEJORAS

Algunas mejoras que se podrían hacer en el proyecto, si se quisiera hacer un desarrollo más amplio del mismo podrían ser las siguientes:

- Obtención de una base de datos mucho más amplia con mayor cantidad de estadísticas sobre los jugadores ya que, aunque la del proyecto cuenta con una gran cantidad de características, el baloncesto es quizás el deporte con mayor número de estadísticas posibles de obtener, así que por ejemplo se podría buscar bases de datos que contasen con estadísticas por posesión, mapas de calor de los jugadores, porcentajes de tiro en determinados segmentos del partido, etc.
- Expansión del proyecto a otras ligas como podría ser la ACB o la Euroliga, así se podría contar con un sistema que pueda servir para todo tipo de necesidades.
- Mejora en los algoritmos de recomendación pudiendo utilizar algoritmos de computación neuronal, ya que ahora mismo el algoritmo utilizado es la distancia euclídea y quizás puede llegar a no ser del todo precisa.
- Mejora en la interfaz gráfica permitiendo una mejor claridad en los datos expuestos y los resultados de estos, incluso mejorando el sistema de filtrado de los jugadores.
- Se podría implementar una interconexión con otros usuarios. Por ejemplo, se podría añadir una nueva función que permitiera ver los jugadores recomendados generales para todos los usuarios, es decir, contar con todas las recomendaciones personales para cada usuario, y juntarlas todas, para hacer una recomendación mayor que se pueda interpretar como que jugadores son tendencias de los usuarios.
- Una mejora obvia sería el despliegue del sitio web, ya que ahora mismo solo está disponible en local.
- Por último, una mejora interesante sería añadir un sistema Recomendador que permitiera predecir resultados de los partidos, a su vez que posibles rendimientos de los jugadores en ellos. Esto podría ser conseguido mediante el uso de Python.

8 BIBLIOGRAFIA

- **Neo4j**: https://neo4j.com/
- **React**: https://es.react.dev/
- **Data NBA**: https://www.basketball-reference.com/leagues/NBA_2023_per_game.html
- **Distancia euclídea**: https://neo4j.com/docs/graph-data-science/current/alpha-algorithms/euclidean
- **Librería GDS Neo4j**: https://neo4j.com/download-center/#algorithms
- **Manual de ayuda de Cypher**: https://neo4j.com/docs/cypher-manual/current/
- **Mi Github**: https://github.com/avizca00