ALGORITMO DE RECOMENDACIÓN DE RESTAURANTES PARA GRUPOS BASADO EN BASES DE DATOS CON GRAFOS

UNIVERSIDAD POLITÉCNICA DE MADRID

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA DE SISTEMAS INFORMÁTICOS

Autor: Jesús Calzas Bedoya

Tutor: Ángel Panizo Lledot

Madrid, Julio 2024

Agradecimientos

Lo primero de todo, me gustaría agradecer a mi tutor, Ángel, por haberme guiado de la mejor forma posible en este proyecto. Ha sido un proyecto que me ha llenado mucho realizar, y que, sin ninguna duda, sin sus directrices e ideas, jamás hubiese llegado a realizar.

En especial, me gustaría agradecer a mi familia el apoyo que me han dado a lo largo de todos estos años, desde el colegio hasta la universidad. Sin ellos no podría haber llegado a este momento, en especial a mis padres, quienes me han estado ahí siempre que lo he necesitado. A continuación, me gustaría agradecerle a mi pareja por haberme apoyado y estar ahí disfrutando los momentos buenos e impulsándome en los no tan buenos; a mis amigos más cercanos por haberme ayudado y apoyado cuando más lo necesitaba, y en general a todas las personas con las que he compartido este camino y que han hecho que la universidad se convierta en una etapa que jamás olvidaré creando muchos recuerdos y buenos momentos.

Resumen

Este Trabajo de Fin de Grado se centra en el **desarrollo de un sistema de recomendación de restaurantes para grupos de personas.** Siendo más específicos, en generar recomendaciones de restaurantes enfocados en las **experiencias positivas del grupo** en cuestión y de sus integrantes. Para ello se hace uso de las experiencias previas que tienen tanto en común como por separado los integrantes del grupo, enfocándose en las que han satisfecho a los usuarios, y descartando los restaurantes que no lo han hecho.

Para obtener los restaurantes, el algoritmo se enfoca en un **filtro de recomendación social**, obteniendo a los usuarios con los gustos más similares al grupo y generando una predicción de los restaurantes que más satisfarían a los integrantes. Con esto se busca cumplir los objetivos de satisfacer a un grupo entero **generando las recomendaciones de la manera más sencilla para ellos**.

A diferencia del resto de sistemas de recomendación, este se enfoca en recomendar cumpliendo las expectativas y gustos de más de una persona. Hasta el momento **no existe ningún sistema que tenga este enfoque.**

Para demostrar esto, he utilizado el dataset de la popular aplicación americana **Yelp**, que cuenta con los datos necesarios para poder llevar a cabo el proyecto y poner a prueba el algoritmo diseñado.

El código desarrollado para este Trabajo de Fin de Grado se encuentra alojado en un repositorio de GitHub, al cual se puede acceder mediante el siguiente enlace: https://github.com/jesuscalzasbedoya/TFGRestaurantes.git

Abstract

This Final Degree Project focuses on the **development of a restaurant recommendation system for groups of people**. More specifically, to generate restaurant recommendations focused on the **positive experiences of the group** in question and its members. To do this, it makes use of the previous experiences that the members of the group have both in common and separately, focusing on those that have satisfied the users, and discarding restaurants that have not.

To obtain the restaurants, the algorithm focuses on a **social recommendation filter**, obtaining the users with the most similar tastes to the group and generating a prediction of the restaurants that would most satisfy the members. This seeks to meet the objectives of satisfying an entire group **by generating recommendations in the easiest way for them**.

Unlike other recommender systems, this one focuses on making recommendations that meet the expectations and tastes of more than one person. So far **there is no other system that has this approach**.

To demonstrate this, I have used the dataset of the popular American application **Yelp**, which has the necessary data to carry out the project and test the designed algorithm.

The code developed for this Final Degree Project is hosted in a GitHub repository, which can be accessed through the following link:

https://github.com/jesuscalzasbedoya/TFGRestaurantes.git

Índice

1. Introducción

1.1. Contexto y problemas

Una de las principales actividades a la hora de relacionarnos con otras personas independientemente de la hora es la de ir a desayunar, comer, cenar o simplemente ir a tomar algo a un bar, restaurante, cafetería… Esto suele tener diferentes motivos según el contexto y las personas que formen el grupo:

- Socialización o amistad: Se trata de una oportunidad para disfrutar de amigos, familiares o seres queridos en un entorno relajado y agradable. Ayuda a desconectar de los problemas diarios y abstraernos de ellos durante unas horas.
- Celebración: es un plan bastante común ir a un restaurante debido a un evento como cumpleaños, aniversarios u otros hitos importantes para festejarlo junto a ciertas personas importantes para el protagonista de la celebración.
- Relaciones profesionales: Las comidas de negocios o reuniones en un tono más informal son muy comunes. Ayudan a establecer contactos, discutir proyectos de trabajo o simplemente fortalecer lazos con tus compañeros de trabajo. Esta es una práctica común entre diferentes empresas, sin ir más lejos, la cena de empresa cuando se acercan las navidades.
- Probar nuevas experiencias culinarias: Cada vez es más común aprovechar que los restaurantes ofrezcan platos y sabores diferentes para explorar y descubrir nuevos sabores a los que no estamos habituados a disfrutar en nuestro día a día.

A pesar de lo gratificante que puede significar esta acción, tiene puntos negativos que pueden ser tediosos al practicarla. Uno de esos puntos, y al cual está enfocado este proyecto, es el de elegir el restaurante al que asistir. A veces esto puede complicarse en un grupo por diferentes motivos. Algunos de ellos pueden ser:

- Preferencias y gustos personales: Hoy en día, vivimos en una sociedad extremadamente globalizada y, sin salir de una misma ciudad, podemos probar diferentes sabores procedentes de innumerables culturas. Cada integrante del grupo tiene sus propios gustos en cuanto a tipo de comida, estilo de restaurante o ambiente, y puede ser complicado llegar a un consenso.
- Ubicación y accesibilidad: La ubicación es esencial si los diferentes miembros del grupo viven en lugares diferentes. Encontrar un lugar que agrade a todos puede ser algo tedioso y complicado ya que es complicado que un restaurante no esté demasiado alejado de ninguno de los integrantes. Como problema añadido está la accesibilidad si uno de los miembros sufre de movilidad reducida o algún otro problema que haga que aumente la dificultad para acceder a lugares que se encuentren fuera de su rutina diaria.
- Presupuesto: Este se trata de un problema bastante común ya que es un factor importante a considerar para algunas personas. Cada miembro del grupo puede tener restricciones presupuestarias diferentes.

- Diversidad cultural: Como mencioné en el primer punto, actualmente vivimos en una sociedad extremadamente globalizada. Esto influye no solo en la oferta de restaurantes, sino también en los propios demandantes. Cada persona puede tener diferentes preferencias basadas en su cultura.
- Experiencias previas: Dicho punto es el centro de este proyecto. Engloba todo lo mencionado anteriormente ya que, de forma indirecta, todos influyen en los restaurantes visitados anteriormente. A su vez, este acaba influyendo en los anteriores debido a las experiencias vividas en ellos, alimentando unos gustos y haciendo que otros pierdan fuerza con el paso del tiempo. Algunos miembros del grupo pueden preferir un restaurante específico debido a una experiencia positiva anterior en un restaurante de características similares. En el otro lado de la balanza nos encontramos lo contrario. Ciertos integrantes del grupo pueden tener una mala experiencia en algún restaurante, por lo que rechazan asistir a un restaurante de características similares a este. Hay veces que incluso, los restaurantes con buenas y malas experiencias pueden llegar a ser los mismos, alimentando la confrontación entre los integrantes del grupo e incrementando la dificultad para elegir un nuevo restaurante.

Para ayudarnos a solventar y minimizar estos problemas, podemos encontrar múltiples aplicaciones o webs que nos pueden ser de ayuda a la hora de conocer lugares nuevos que sean de agrado o no de todos los individuos del grupo. Estas, suelen estar enfocadas a la recomendación hacia una persona, y después, el grupo debe ponerse de acuerdo con dichas recomendaciones y elegir entre ellas. Estas aplicaciones, como “Tripadvisor”, “The Fork” o “Yelp”, se basan en algoritmos y sistemas de recomendación basándose en búsquedas y filtros que introduce el usuario, información de reseñas y calificaciones anteriores para así afinar los gustos del usuario contratándolos con las características de los restaurantes que tienen alojados en su base de datos.

De forma no tan generalizada, existen los sistemas de recomendación grupales. A diferencia de los sistemas de recomendación convencionales, se centran en las preferencias compartidas de un grupo de personas, en vez de una en particular. Su principal objetivo es proporcionar recomendaciones que satisfagan las preferencias y necesidades del grupo. Para ello, pueden enfocarse en diferentes métodos como la votación o ponderación grupal, donde los miembros del grupo dan valoraciones a las diferentes opciones recomendadas, quedándose con las que mayor valoración tengan; optimización de la diversidad, donde se dan recomendaciones que satisfagan los diferentes gustos que existen dentro del grupo; o los perfiles grupales, que utilizan la creación de un perfil nuevo en base a los perfiles de los individuos del grupo.

1.2. Qué es Group Eat

Group Eat es el sistema en el que está centrado este proyecto. Siguiendo con la idea mencionada en el apartado anterior, este proyecto se trata de un sistema de recomendación grupal enfocada a la recomendación de restaurantes. Para ello, haré uso del sistema de perfil grupal. En base a los usuarios que formen el grupo, se creará un perfil que reúna los gustos de todos ellos. El corazón de estas recomendaciones serán las reseñas previas que han realizado los propios usuarios de forma individual. Se tendrán en cuenta las reseñas que mejor valoración tengan se comenzará a buscar a partir de los restaurantes relacionados.

Siguiendo un sistema de recomendación con filtrado colaborativo, se obtendrán restaurantes afines a los gustos de los usuarios del grupo. Estos resultados podrán ser visibles en una página web.

1.3. Dataset

En este proyecto, los datos son la parte principal ya que, gracias a ellos, son posibles las recomendaciones. Para ello, se ha hecho uso del *dataset* educativo de "Yelp" [1].

Yelp es una plataforma que alberga negocios como restaurantes, tiendas y demás tipos de servicios que pueden ser consumidos por usuarios. Fue fundada en 2004 y desde entonces ha ido creciendo hasta ser una de las principales plataformas de reseñas. Dada su longevidad, constan con una gran base de datos de usuarios, negocios y reseñas. Aquí en España lo podemos comparar con "Tripadvisor", ya que sería el equivalente.

Yelp tiene una particularidad que a nivel estudiantil lo convierte en muy interesante. Esta plataforma cuenta con un *dataset* académico de uso libre en su página web. Este *dataset* es extraído de su base de datos real, por lo que contiene datos reales. Es un *dataset* extremadamente completo que consta de 5 documentos separados y uno adicional de imágenes. Para este proyecto solo es necesario el conjunto principal de 5 documentos. Este conjunto está formado por: *users*, *reviews*, *business*, *checkin* y *tip*. Todos ellos son de tipo Json.

- El primer archivo contiene información de los usuarios (*user_id, name, review_count, yelping_since, friends, useful, funny, cool, fans, elite, average_stars, compliment_hot, compliment_more, compliment_profile, compliment_cute, compliment_list, compliment_note, compliment_plain, compliment_cool, compliment_funny, compliment_writer, compliment_photos*) en forma de atributos. De estos datos, únicamente nos serán interesantes el *user_id* que nos servirá de identificador único, *name* que contiene el nombre del usuario, *friends* que es una lista con identificadores de otros usuarios que tiene guardado como amigos. El resto de información podría servir a nivel informativo para cumplimentar información, pero el proyecto no se centra en eso.

- El segundo archivo contiene la información relacionada con las reviews (*review_id, user_id, business_id, stars, date, text, useful, funny, cool*) en forma de atributos. Este archivo relaciona los usuarios con los restaurantes visitados. Los atributos más interesantes para el proyecto es el *user_id* que sirve para identificar el propietario de la review, el *business_id* representa el restaurante, el campo *stars* se trata de la valoración dada en la reseña, y *review_id* es el identificador único de la reseña.
- El tercer archivo contiene la información relacionada con los restaurantes y demás negocios. Este archivo contiene (*business_id, name, address, city, state, postal code, latitude, longitude, stars, review_count, is_open, attributes, categories, hours*). Group Eat únicamente utilizará el *business_id*, *name* y *address* para su funcionamiento.

Estos 2 siguientes archivos no serán utilizados en el proyecto, por lo que los mencionaré superficialmente:

- *Checkin* únicamente contiene el id del negocio y una cadena de fechas
- Tip contiene consejos escritos por usuarios sobre negocios

Como hemos podido ver, se trata de un conjunto de datos muy completo, lo que lo hace perfecto para este proyecto, poniendo solución a posibles problemas que pudiesen surgir relacionados con la falta de datos.

En el otro lado de la balanza, podemos ver que contiene datos que igual no son de nuestro interés en un principio. Más adelante veremos cómo ha sido tratado hasta llegar a tener la base de datos completamente operativa partiendo de esta base.

1.4. Objetivos

Este proyecto se ve regido por unos objetivos un tanto marcados:

- En primer lugar, se busca la indagación en el mundo de las recomendaciones grupales. Se trata de un ámbito que no es tan conocido y explotado como será en un futuro. Hoy en día vemos recomendaciones en prácticamente cualquier sistema que utilicemos, pero todas de carácter individual. En este caso se tratará de un sistema de recomendación grupal de restaurantes.
- Aplicación y explotación de las bases de datos con grafos en este ámbito debido a su gran potencia a la hora de relacionar nodos haciéndolas idóneas.
- En segundo plano queda la creación de una web. Su finalidad es la de utilización del sistema.
- Como objetivo global, al tratarse de un TFG, se busca demostrar los conocimientos obtenidos a lo largo de la carrera.

1.5. Motivación

El principal motivo que promueve este Trabajo de Fin de Grado es investigar el mundo de las recomendaciones e indagar en él. Hoy en día se encuentran presentes en todos los ámbitos de nuestra vida cotidiana, y han llegado para quedarse. Con este proyecto se busca aprender de ellos y, sobre todo, intentar abrir nuevos caminos en base a las investigaciones que realice.

Finalmente, la creación de un nuevo algoritmo es un reto perfecto para asumir en un proyecto como este. Actualmente se presenta a diario el problema de elegir un restaurante al que ir cuando se junta un grupo de personas. Con este proyecto se busca encontrar una solución a dicho problema, facilitando esta tarea a los usuarios que lo utilicen. Para ello es imprescindible crear un algoritmo específico para el sistema.

Por otro lado, es importante impulsar cualquier algoritmo con una base de datos potente que provea los datos necesarios para que pueda funcionar este. En este caso, se ha optado por utilizar una base de datos basada en grafos. Este no es el principal tipo de base de datos que se utiliza normalmente. Con este proyecto se busca impulsar su utilización y gestionar una integración con las tecnologías encargadas de las recomendaciones.

1.6. Estructura de la memoria

En el capítulo 1, tras una breve introducción, nos encontramos información explicativa de lo más básico del proyecto, como el *dataset* utilizado, motivaciones u objetivos.

En el capítulo 2, en el estado del arte se explican los diferentes enfoques que puede tener un sistema de recomendación grupal, haciendo especial hincapié en el utilizado en este proyecto.

En el capítulo 3, nos adentramos en la parte principal del proyecto, el desarrollo del proyecto. Comienzo explicando los requisitos por los que se rige este proyecto, seguido de la metodología que se ha seguido en el proyecto. El diseño del algoritmo junto a las pruebas elegidas serán lo siguiente que se encuentran en este documento. En ese apartado se explicará en que me he regido para diseñar y el algoritmo de este sistema junto con las pruebas seguidas para ver su funcionamiento. A continuación, se explica que tecnología he utilizado y porqué en todas las etapas de este proyecto. Finalmente, se explica la arquitectura utilizada y los diferentes sprints en los que se ha dividido el trabajo.

En el capítulo 4 se encuentran los resultados obtenidos al ejecutar las pruebas mencionadas anteriormente, junto a los problemas encontrados a lo largo del desarrollo del proyecto.

En el capítulo 5 se muestran las conclusiones. Estas incluirán trabajo futuro que se recomienda a la hora de continuar con el proyecto, el impacto que puede generar Group

Eat y unas conclusiones personales de cómo me he sentido a lo largo de este proceso y qué pienso sobre este proyecto.

El capítulo 6 contiene la bibliografía que contiene las fuentes utilizadas a lo largo del proyecto.

Finalmente, se encuentra el Anexo A, que contiene los datos de las pruebas realizadas al sistema.

2. Estado del arte

2.1. Enfoques en sistemas de recomendación

Un filtro de recomendación es una técnica perteneciente a los sistemas de recomendación. Este sirve para seleccionar elementos que pueden ser del gusto del recomendado. Su objetivo principal es el de reducir las opciones y presentar al usuario las más relevantes en cuanto a sus preferencias. Esta técnica es comúnmente utilizada en múltiples sitios web dedicados al *e-commerce* que consideramos cotidianos como *Amazon, Netflix* o *Spotify* [2].

Para obtener las recomendaciones finales se pueden tener en cuenta diversos factores. En base a estos factores, el filtro de recomendación utilizado puede ser de diferentes tipos:

- Filtrado basado en contenido: Se genera una recomendación cuyos productos resultantes sean similares a los ya consumidos por el usuario.
- Filtrado colaborativo: El resultado de la recomendación viene dado a partir de productos que han gustado a otros usuarios similares al usuario principal.
- Filtrado híbrido: Este filtro es una mezcla de los 2 anteriores. El resultado procede de productos similares a los que le ha gustado al usuario y productos que les han gustado a los usuarios similares.
- Filtrado en conocimiento: Se enfoca en generar recomendaciones basadas en reglas de experto. Es muy interesante para temas muy específicos donde el usuario no suele cambiar sus preferencias.

2.2. Sistemas de recomendación con filtrado basado en contenido

El algoritmo de filtrado se basa en las características de los productos consumidos por el usuario para obtener otros productos con características similares a estos. Se trata de un modelo sencillo que únicamente necesita la información de los productos para ponerse en funcionamiento. El filtro analiza las características de los elementos recomendables (texto, imágenes, atributos, metadatos…) determinando así su relevancia para el usuario [3].

Este filtro suele seguir unos ciertos pasos de forma genérica:

- Se construye un perfil del usuario en función a sus preferencias y comportamientos del usuario. Este puede incluir productos consumidos o con los que haya interactuado anteriormente (productos vistos, leídos, comprados…)

- A continuación, se extraen las características relevantes de los productos candidatos a ser recomendados al usuario. Estos se representan en forma de atributos relevantes que los identifican frente a otros.
- Las características de los productos candidatos se comparan con las características de los productos asociados al perfil del usuario mencionados en el primer paso mediante algún algoritmo de comparación o similitud.
- Finalmente se generan las recomendaciones en base a las coincidencias de las características resultantes del paso anterior.

Como puntos a favor de este tipo de filtro podemos ver su capacidad para ofrecer recomendaciones personalizadas y relevantes incluso con poca información de otros usuarios. Únicamente se depende del usuario principal.

En cambio, podemos encontrar algunos puntos negativos que pueden ser travas con el tiempo. Este filtro puede no captar preferencias complejas o cambios de interés en el usuario. La diversidad de recomendaciones se ve mermada al ofrecer recomendaciones únicamente dentro de una categoría o tipo de elementos, enfrascando al usuario únicamente en ellos impidiendo su desarrollo o descubrir nuevas recomendaciones que también pueden serle de interés.

Para explicar mejor este filtro voy a dar un ejemplo: Un usuario ha comprado recientemente una alfombrilla de ratón, por lo que el sistema generará una recomendación que dará como resultado: periféricos.

2.3. Sistemas de recomendación con filtrado colaborativo

El filtro colaborativo se basa en las calificaciones otorgadas por los diferentes usuarios registrados en el sistema analizando sus preferencias y comportamientos [5]. En otras palabras, se tienen en cuenta las valoraciones registradas en la base de datos, se buscan a usuarios con perfiles parecidos al del usuario para sugerir otros productos en base a sus preferencias.

La base de este filtro es la idea de que, si dos usuarios tienen reseñas parecidas en el pasado, sus gustos serán similares. Por lo tanto, lo que le ha gustado a uno que no es conocido por el otro, puede gustarle a este segundo.

Podemos distinguir 2 enfoques de este filtro:

- Basado en la vecindad
    - Se basa en la similitud entre usuarios. Se construye una matriz de similitud entre usuarios basada en comportamientos pasados como calificaciones a productos, y vecinos o usuarios similares [3].
    - Estos vecinos nos proporcionarán sus preferencias o experiencias pasadas para generar recomendaciones para nuestro usuario con una

predicción de la valoración que puede resultar si finalmente lo califica este usuario.

- Basado en modelos
    - Se construye un modelo estadístico o matemático que representa las preferencias de los usuarios y los elementos recomendables.
    - El modelo es entrenado utilizando datos históricos de calificaciones. Finalmente es utilizado para predecir las valoraciones que le podría dar el usuario en caso de probarlo, generando las recomendaciones a partir de ellas.

Como puntos a favor encontramos la capacidad de descubrimiento de nuevas recomendaciones y la habilidad para ofrecer recomendaciones precisas y relevantes sin la necesidad de tener una gran cantidad de información de los productos recomendables [6].

En cambio, podemos encontrar ciertos puntos negativos como el problema del *Cold Start*

- Se trata de una limitación originada por una baja cantidad de usuarios o elementos registrados.
- *Cold Start* de usuario: Ocurre cuando el usuario se acaba de registrar en el sistema y no se tiene información sobre sus referencias pasadas (no hay ninguna registrada).
- *Cold Start* de sistema: Se observa cuando se implementa un nuevo sistema de recomendación sin datos previos o poca información histórica.

Dadas las características del filtro, se trata del más adecuado para *Group Eat*. El problema del *Cold Start* se acaba solucionando gracias al *dataset* elegido. Al tener una gran cantidad de información, se puede operar con él sin miedo a quedarnos faltos de datos.

Continuando con el ejemplo del filtro anterior, un usuario ha comprado una alfombrilla de ratón. En este caso, el sistema le recomendará objetos cualesquiera que hayan comprado también usuarios que hayan comprado alfombrillas de ratón. Estos objetos pueden ser del tipo que sea, no tienen por qué ser periféricos.

2.4. Sistemas de recomendación con filtrado híbrido

Un filtro de recomendación híbrido mezcla las características de los anteriores 2 filtros. Esto lo hace un filtro muy completo ya que mezcla lo mejor de ambos, evitando ciertos problemas que pueden surgirles por separado [7]. Este filtro procesa tanto las características del producto o servicio como las interacciones realizadas.

Esta técnica presenta ventajas como la capacidad de manejar el problema del *Cold Start* y la capacidad de proporcionar recomendaciones más precisas para los usuarios del sistema.

En cambio, el hecho de combinar y gestionar diferentes fuentes de información hace que la complejidad del algoritmo se incremente a la par que el costo computacional causando la necesidad de optimizar los componentes y recursos del sistema.

2.5. Sistemas de recomendación basados en conocimiento

Este filtro de recomendación es un enfoque que utiliza información y reglas de conocimiento experto. Se basa en el análisis de datos históricos o en el comportamiento de usuarios similares, apoyándose en información explícita y estructurada sobre los elementos recomendables y las preferencias de los usuarios.

Este sistema de recomendación tiene como ventaja la capacidad para ofrecer recomendaciones precisas y adaptadas a las necesidades de los usuarios sin la necesidad de tener mucha información del usuario [8]. Además, proporcionan explicaciones claras sobre la generación de las recomendaciones por el conocimiento y reglas explícitas usadas.

En la otra mano, podemos ver la dificultad que supone el mantenimiento de la base de datos y la participación de los expertos. Además, no se pueden predecir los cambios de preferencias de los usuarios.

A diferencia del filtrado basado en contenido, este filtro es utilizado en dominios donde se necesita un resultado más profundo y detallado, mientras que el filtrado basado en contenido es mejor para adaptarse a cambios de gustos de los usuarios y para recomendar productos más variados. Para explicarlo mejor utilizaré un ejemplo: Un usuario se encuentra en una plataforma de *streaming* de películas. Si solo se dedicase esa plataforma a un género, lo mejor sería el filtrado basado en conocimiento. Si es una plataforma genérica, que tiene contenido de muchos géneros, lo mejor es un filtrado basado en conocimiento.

3. Desarrollo del proyecto

3.1. Requisitos del proyecto

En esta sección, se verán los requisitos por los que se han regido este proyecto. Podemos diferenciar por un lado los requisitos funcionales y por otro los no funcionales.

3.1.1. Requisitos funcionales

ID	NOMBRE	DESCRIPCIÓN
RF1	Validación del usuario	El sistema debe ser capaz de comprobar que el usuario pertenece al sistema mediante su identificador de usuario.
RF2	Obtención de amigos	El sistema debe ser capaz de obtener y almacenar a los integrantes del grupo.
RF3	Obtención de ciudad	El sistema debe ser capaz de obtener y almacenar la ciudad donde debe realizarse la recomendación.
RF4	Generar perfil de grupo	El sistema debe ser capaz de generar un perfil de grupo a partir de 1 o más usuarios.
RF5	Generar recomendación	El sistema debe ser capaz de generar una recomendación devolviendo los 5 restaurantes con mayor predicción.
RF6	Mostrar los resultados	El sistema debe ser capaz de mostrar los 5 restaurantes con mejor predicción indicando el nombre, dirección y valoración.
RF7	Repetir recomendación	El sistema debe ser capaz de permitir al usuario volver al paso de selección de amigos
RF8	Interfaz gráfica	El sistema debe disponer de una aplicación web como interfaz gráfica

3.1.2. Requisitos no funcionales

ID	NOMBRE	DESCRIPCIÓN
RNF1	Accesibilidad	El sistema debe poder ser utilizada por gente que sufre cualquier tipo de daltonismo.
RNF2	Compatibilidad	El sistema debe ser compatible con una interfaz web. Además, debe estar preparada para que futuras tecnologías de mostrado de datos puedan acceder al sistema de forma sencilla
RNF3	Eficiencia	El sistema debe poder generar una recomendación en menos de 5 segundos una vez introducidos todos los parámetros.
RNF4	Escalabilidad	El sistema debe poder soportar a 1000 usuarios realizando peticiones de forma simultánea.

RNF5	Fiabilidad	La base de datos debe mantenerse intacta tras realizar una recomendación.
RNF6	Precisión	El sistema debe tener al menos una precisión media del 65% de acierto en las recomendaciones.
RNF7	Robustez	El sistema debe ser capaz de denegar el acceso cuando un usuario intente acceder a una URL del sistema sin seguir los pasos estipulados por el sistema.
RNF8	Usabilidad	La página web debe ser sencilla y fácilmente entendible, generando una recomendación en menos de 4 pasos.

3.2. Metodología del trabajo

3.2.1. Desglose del trabajo

Para la planificación de este proyecto, he utilizado la metodología más utilizada en el mundo de las metodologías ágiles y que se encuentra expandiéndose hacia diferentes sectores: SCRUM [9]. Esta metodología se basa en la entrega de productos de forma eficiente y creativa buscando el valor máximo en cada entrega [10].

En SCRUM, las personas que forman parte se encuentran marcadas claramente mediante diferentes roles que, al tratarse de un proyecto unipersonal como es este, me ha sido complicado establecer.

- *Product owner*: como representante del producto se encuentra el tutor de este TFG, Ángel Panizo. La función de un *product owner* es la de mostrar claridad sobre los requisitos, determinar los siguientes pasos a seguir según la planificación y priorización de estos, y finalmente, verificar si se cumplen en cada entrega.
- Equipo Scrum: dentro de un equipo Scrum se destacan el *Scrum Master* (representante y líder del equipo) y el resto de miembros, como pueden ser desarrolladores. En este caso, el único miembro soy yo, Jesús Calzas Bedoya, por lo que me he encargado de llevar a cabo el desarrollo del sistema, realizar las estimaciones de los requisitos, solucionar los problemas ocurridos, preservar la utilización de las metodologías ágiles y mostrar el producto resultante al final de cada *sprint*.

En cuanto a la división del trabajo, se ha optado por dividir el trabajo en *sprints* como avanzaba en el párrafo anterior. Siguiendo un modelo en cascada con retroalimentación, cada *sprint* se ha basado en el desarrollo de una de las partes del sistema, dando como resultado, un total de 4 *sprints*:

- *Sprint* 1: Desarrollo de la base de datos. En ella se incluye la creación de la base de datos, introducción del *dataset* y adaptación a las necesidades del producto.

Dando como resultado una base de datos completamente funcional y operativa para cualquier *query* que pueda necesitarse en el proyecto.

- *Sprint* 2: Desarrollo de la lógica del sistema. Dando como resultado un sistema completamente funcional, pero sin interfaz gráfica para el usuario.
- *Sprint* 3: Creación de interfaz web para la visualización de los datos y fácil manejo del sistema. Dando como resultado el producto completamente funcional cumplimentando el resultado del *sprint* anterior con una interfaz gráfica.
- *Sprint* 4: Creación del sistema de pruebas para comprobar la veracidad de los resultados del sistema.

En el apartado 3.7 se explicará en más detenimiento el contenido de los *sprints*.

Para la visualización del trabajo pendiente, se ha utilizado un tablero Kanban. Un tablero Kanban está basado en diferentes columnas, que determinan el estado en el que se encuentran cada una de las tareas e historias de usuario asociadas al *sprint* en el que se encuentre el proyecto en ese momento [11]. En particular, en este proyecto se han instaurado 4 columnas: Listo para hacer (comúnmente conocido como "*to do*"), en curso, esperando revisión (también conocido como terminado), listo o cerrado (a este paso se llega una vez que se dé por verificada la funcionalidad de la *historia de usuario*) en cuestión.

Los requisitos funcionales se han especificado como historias de usuario de cara a la planificación de los *sprints* y su visualización en el tablero Kanban:

ID	NOMBRE	DESCRIPCIÓN
US1	Validación del usuario	Cómo usuario de Group Eat quiero poder validar mis credenciales de usuario del sistema para poder acceder a él.
US2	Seleccionar amigos	Cómo usuario del sistema, quiero poder seleccionar amigos de mi lista de amigos, para determinar los integrantes del grupo.
US3	Seleccionar ciudad	Como usuario del sistema, quiero poder seleccionar una ciudad, para recibir recomendaciones enfocadas a esa ciudad.
US4	Visualización del resultado	Como usuario del sistema, quiero poder visualizar los restaurantes recomendados, para poder conocer cuáles son los restaurantes y la valoración estimada.
US5	Volver a generar recomendación	Como usuario del sistema, quiero poder volver a recibir una recomendación, para cambiar alguno de los parámetros indicados anteriormente.

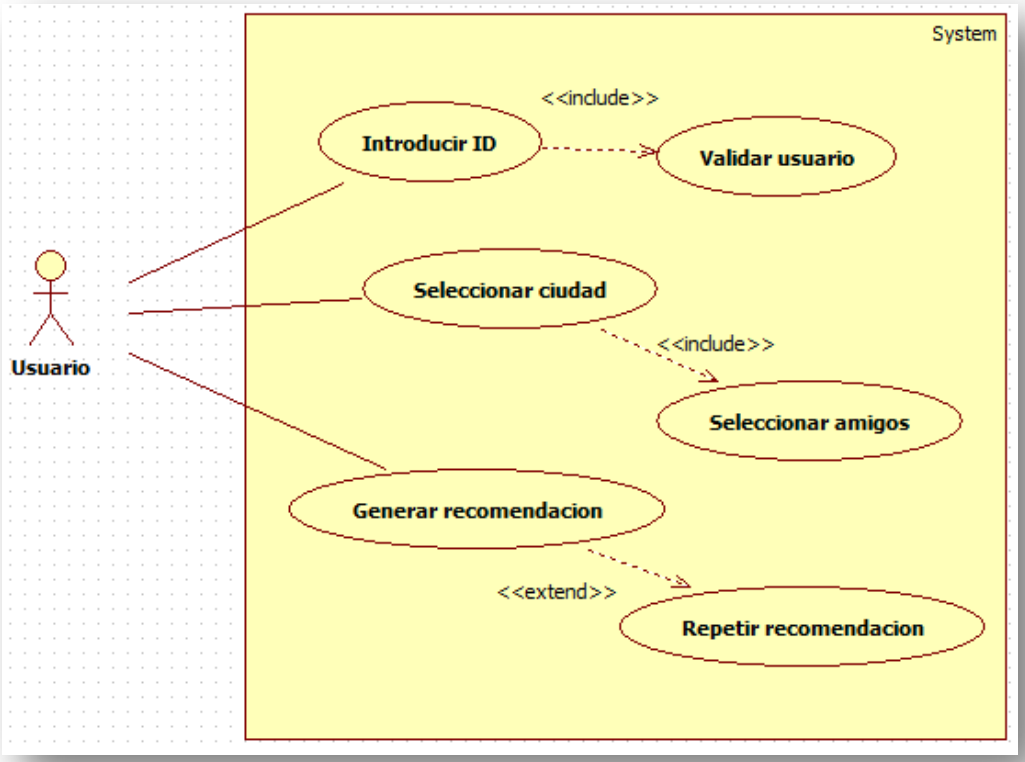


Diagrama de casos de uso del sistema.

3.2.2. Priorización

En el apartado anterior quedan reflejados los requisitos por los que se rige el proyecto. Todos tienen su importancia, pero es importante determinar cuáles son las prioridades del proyecto, y cuales deben ser cumplidos al 100%. Para ello, se va a utilizar una técnica de priorización típica de las metodologías ágiles: la técnica MoSCoW [12]. Esta técnica clasifica normalmente tareas o historias de usuario, según la importancia que tengan de cara al sprint que vaya a comenzar. En este caso, lo voy a utilizar para priorizar los requisitos e historias de usuario mencionadas anteriormente:

- Requisitos no funcionales:
  - *Must*/Obligatorio:
    - Eficiencia, accesibilidad, fiabilidad y precisión, son los requisitos que deben cumplirse obligatoriamente.
  - *Should*/Debería estar
    - Compatibilidad, robustez y usabilidad son características importantes que deben ser cumplidas.
  - *Could*/Podría esta
    - Escalabilidad es el requisito menos importante de los impuestos a este proyecto debido a las características de este.

- Requisitos funcionales
    - Los requisitos funcionales son todos imprescindibles ya que se trata del funcionamiento básico del sistema
- Historias de usuario
    - Al igual que los requisitos funcionales son imprescindibles todas y no deben faltar ninguna. En el caso de que no se pueda por falta de tiempo, la menos imprescindible es la US5.
    - De ellas se hará un análisis más en profundidad en el apartado 3.7, donde se irá una por una priorizando las tareas necesarias para implementarlas.

3.3. Base de datos

En este apartado se describirá cómo está formada y cómo se ha pasado del *dataset* inicial a la base de datos final. Para la base de datos he utilizado una de tipo NoSQL, más concretamente, basada en grafos. Neo4j es la tecnología idónea para este sistema de recomendación ya que aporta una gran ordenación de datos junto a la eficiencia esperada para la obtención de peticiones.

Una base de datos basada en grafos se estructura principalmente con 2 tipos de objetos: los nodos y las relaciones. Los nodos albergan los principales datos de cada objeto. Estos están formados por atributos y se distinguen entre nodos de diferentes tipos. Por otro lado, nos encontramos con las relaciones. Como su propio nombre indica, se encargan de relacionar los nodos entre sí. Al igual que los nodos, las relaciones tienen sus propios atributos y se distinguen entre ellos en base al tipo de relación que sean.

La base de datos de este proyecto se ha estructurado de la siguiente forma:

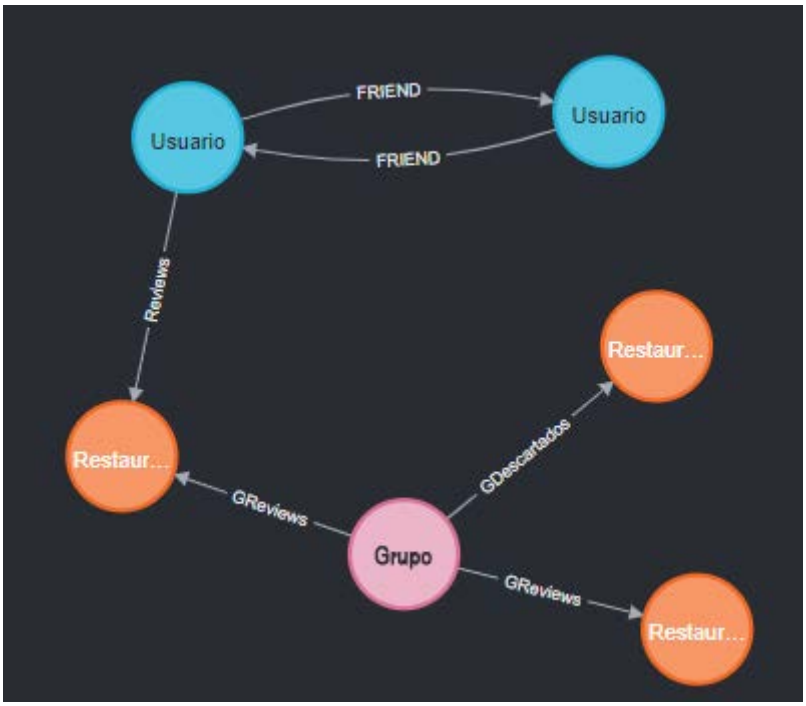


Diagrama que ilustra los nodos y relaciones que forman parte de la base de datos

3.3.1. Nodos

Los nodos de la base de datos se dividen en 3 tipos:

- Usuario:
  - Se trata de toda la información registrada de los usuarios.
  - Atributos:
    - *User_id*: Identificador único de cada usuario
    - *Nombre*: Nombre del usuario
- Restaurante:
  - Información relacionada con los restaurantes.
  - Atributos:
    - *Business_id*: Identificador único de cada restaurante.
    - *Nombre*: Nombre del restaurante.
    - *Ciudad*: Nombre de la ciudad en la que se ubica el restaurante.
    - *Dirección*: Dirección en la que se encuentra el restaurante.
- Grupo:
  - En estos nodos se guarda una información básica de los grupos.
  - Se trata de nodos volátiles que no perdurarán en la base de datos más allá de la recomendación.
  - Se crean una vez se determinen los integrantes del grupo, y se eliminan una vez se haya realizado la recomendación.
  - Atributos:
    - *Grupo_id*: identificador único de cada grupo. Se crea a partir de la concatenación de los *user_id* de los integrantes del grupo.

3.3.2. Relaciones

Las relaciones de la base de datos se dividen en 5 tipos:

- *Reviews*
  - Relaciona un usuario y un restaurante.
  - Almacena como atributo la valoración dada por el usuario entre 1 y 5 estrellas.
  - Está formado a partir del fichero de *Reviews* del *dataset*. Inicialmente se introdujeron los datos como nodos debido al gran volumen de datos proporcionado, y luego fue transformado a una relación entre ambos nodos participantes.
- *Friends:*
  - Relaciona un usuario con otro usuario.
  - No almacena ningún atributo.
  - Indica los amigos que tiene un usuario.
  - En el *dataset* inicial se encontraba embebido dentro de un atributo, pero fue modificado transformándose en una relación para eliminar datos

vacíos debido a que el *dataset* no es completo y hay *user_id* de usuarios inexistentes.

- *GReviews*:
  - Relaciona un grupo con un restaurante.
  - Almacena la valoración media de todos los integrantes del grupo sobre un restaurante siempre que sea positiva.
  - Es temporal ya que, al terminar la recomendación, se elimina a la vez que el grupo.
  - Su fin es el de marcar los restaurantes positivos del grupo para obtener los usuarios afines necesarios del algoritmo y no aparecer en una futura recomendación.
- *GDescartados*:
  - Relaciona un grupo con un restaurante.
  - No almacena ningún atributo.
  - Es temporal ya que, al terminar la recomendación, se elimina a la vez que el grupo.
  - Su fin es el de marcar los restaurantes negativos del grupo para no tenerlos en cuenta para el resto del algoritmo, salvo para no ser recomendado.

3.4. Algoritmo de recomendación

En este apartado se describirá cómo está diseñado el algoritmo del sistema. Se trata de la parte principal de este proyecto ya que, haciendo uso de la base de datos previamente descrita, se encargará de generar las recomendaciones de los restaurantes.

En el primer apartado se hará una descripción detallada de cómo está estructurado el algoritmo. Se hará referencia a las funciones base que han sido tomado como referencia para llevarse a cabo su creación.

A continuación, en el segundo apartado, se hará referencia al sistema dedicado a las pruebas del algoritmo principal.

3.4.1. Diseño del algoritmo

El algoritmo se basa en gran parte de información obtenida a partir de la base de datos y de *querys* que van a realizar la función de este. Esto último se debe a que la base de datos posee una gran potencia para la gestión de la información y es de gran interés para cumplir el requisito de rendimiento.

A continuación, adjunto una muestra del pseudocódigo seguido en el algoritmo:

```
grupo = generarGrupo(usuario, amigos)
usuariosAfines = obtenerUsuariosAfines(grupo)
establecerSimilitud(usuariosAfines, Grupo)
sesgarUsuarios(usuariosAfines)
restaurantes = generarPrediccion(grupo, usuariosAfines)
listaFinal = sanearRestaurantes(restaurantes)
return listaFinal
```

El algoritmo se estructura en 3 partes principales:

- Usuarios afines:
  - Para comenzar el funcionamiento del algoritmo, tras la formalización del grupo, se requiere la obtención de usuarios afines al propio grupo.
  - La selección está basada en el índice de Jaccard dando prioridad a los usuarios con más restaurantes en común con el grupo [13]:

$$Jaccard = \frac{Restaurantes\ comunes}{Restaurantes\ totales}$$

  - Estos usuarios se obtienen a partir de las valoraciones positivas que han realizado de los restaurantes. Mediante la siguiente *query* se obtienen los 20 usuarios con mayor valoración media de sus reseñas propias, de esta forma, podremos obtener los usuarios que hayan visitado los mejores restaurantes, o que por lo menos hayan resultado más satisfechos, a partir de restaurantes con valoraciones positivas en común.
  - Cabe destacar que se obtiene en la consulta la media de las *reviews* de los usuarios a pesar de que existía un atributo que la almacenaba en el *dataset* inicial ya que no se trata de un dato real. El *dataset* no es completo al 100% y no se encuentran todas las *reviews* realizadas.
  - A continuación, adjunto la consulta realizada a la base de datos para obtener los usuarios afines.

```
MATCH (gr:Grupo{grupo_id: 'grupo_id'})-[:GDescartados]-
>(rd:Restaurante)

WITH collect(r.business_id) AS aceptados, collect(rd.business_id)
AS descartados

MATCH (r1:Restaurante)<-[:Reviews]-(u)-[:Reviews]-
>(r2:Restaurante)

WHERE r1.business_id IN aceptados AND (NOT r2.business_id IN
aceptados AND NOT r2.business_id IN descartados)

WITH u.user_id AS uid, count(DISTINCT r2) AS rDif, aceptados,
descartados

MATCH (u:Usuario)-[:Reviews]->(r:Restaurante)

WHERE u.user_id IN uid AND (r.business_id IN aceptados OR
r.business_id IN descartados)

WITH u.user_id AS userId, count(DISTINCT r) AS rComun, rDif

MATCH (u:Usuario)-[rev:Reviews]->(r:Restaurante)

WHERE u.user_id IN userId

WITH u.user_id AS uid, avg(rev.stars) AS media, rComun, rDif
ORDER BY rComun DESC, rDif DESC LIMIT 20

RETURN uid, media
```

- Similitud
    - Mide la similitud entre el grupo y los usuarios afines.
    - Para ello, se hace uso de la siguiente ecuación [14]:

$$Similitud(grupo, usuario) = \frac{\sum(vGrupoRestaurante_i - vMediaGrupo) \cdot (vUsuarioRestaurante_i - vMediaUsuario)}{\sqrt{(\sum(vGrupoRestaurante_i - vMediaGrupo))^2} \cdot \sqrt{(\sum \cdot (vUsuarioRestaurante_i - vMediaUsuario))^2}}$$

    - En esta ecuación se compara las valoraciones que ha dado el grupo sobre los restaurantes en común y la valoración media de todas las reseñas del grupo, con las valoraciones que ha dado el usuario afín y su valoración media propia.
    - El numerador está formado por un sumatorio del producto de la valoración que le ha dado el grupo al restaurante en común i (*vGrupoRestaurante*$_i$) menos la valoración media del grupo (*vMediaGrupo*) por la valoración del usuario afín dada al restaurante en común i (*vUsuarioRestaurante*$_i$) menos la valoración media del usuario (*vMediaUsuario*). Sea i el restaurante i-ésimo entre 1 y n siendo n el número de restaurantes en común.
    - El denominador se trata del producto de 2 raíces cuadradas. Por un lado, tenemos el sumatorio de la valoración que le ha dado el grupo al restaurante en común i (*vGrupoRestaurante*$_i$) menos la valoración media del grupo (*vMediaGrupo*), todo ello elevado al cuadrado. Por otro lado,

tenemos el sumatorio de la valoración del usuario afín dada al restaurante en común i ($vUsuarioRestaurante_i$) menos la valoración media del usuario (*vMediaUsuario*), todo ello elevado al cuadrado. Sea i el restaurante i-ésimo entre 1 y n siendo n el número de restaurantes en común.

- Finalmente se obtiene un grado de similitud que será utilizado en las predicciones.
- A continuación, se muestra cómo se obtienen los datos necesarios para generar el grado se similitud. Esta *query* devolvería la valoración media del grupo y del usuario afín.

```
MATCH (g:Grupo{grupo_id: 'grupo_id'})-[grev:GReviews]-
>(gr:Restaurante)

MATCH (u:Usuario{user_id'user_id'})-[urev:Reviews]-
>(ur:Restaurante)

WHERE ur.business_id = gr.business_id

RETURN grev.stars as vGrupo, urev.stars as vUsuario
```

- Matemáticamente presenta un caso de error esta ecuación. Supongamos que únicamente hay una reseña en común. Si esta valoración coincide con la valoración media, se generaría un 0 en el numerador y/o en el denominador pudiendo provocar resultados erróneos como $\frac{0}{x} = 0$, o indeterminaciones como $\frac{x}{0}$ y $\frac{0}{0}$ siendo x cualquier número mayor que 0.
- Para solucionar dicho error, se ha propuesto la siguiente solución:

```
si (la diferencia de la valoracion y la valoracion media del
grupo = 0)

entonces

      diferenciaGrupo = 0.01

si (la diferencia de la valoracion y la valoracion media del
usuario = 0)

entonces

      diferenciaUsuario = 0.01

sumar diferenciaGrupo*diferenciaUsuario
```

- Predicción
  - La función de predicción se encarga de obtener una estimación de la valoración que daría el grupo en el caso de visitar finalmente el restaurante propuesto [14].

- Cada restaurante es obtenido de la lista de restaurantes del usuario afín con una valoración positiva.

$$Predicción(restaurante, grupo, usuario)$$
$$= vGrupoMedia + Similitud(grupo, usuario) \cdot (vUsuarioRestaurante - vMediaUsuario)$$

  - A la valoración media del grupo (*vGrupoMedia*) se le suma el producto del grado de similitud entre el grupo y el usuario por la resta de la valoración del usuario al restaurante (*vUsuarioRestaurante*) menos la valoración media del usuario (*vMediaUsuario*).
  - Se realiza a la base de datos la siguiente consulta. De ella obtendremos los restaurantes que ha visitado el usuario y no el grupo, junto a la valoración que le ha dado el usuario.

```
MATCH (g:Grupo{grupo_id : 'grupo_id'})-[:GReviews]-
>(ra:Restaurante)

MATCH (g:Grupo{grupo_id : 'grupo_id'})-[:GDescartados]-
>(rd:Restaurante)

WITH COLLECT(ra.business_id) AS rAceptados,
COLLECT(rd.business_id) AS rDescartados

MATCH (c:Usuario{user_id: 'user_id'})-[rev:Reviews]-
>(r:Restaurante)

WHERE NOT r.business_id IN rAceptados AND NOT r.business_id IN
rDescartados

RETURN r.business_id as restauranteId, rev.stars as stars
```

Una vez explicadas las funciones principales del algoritmo, a continuación, se presentará la estructura completa del algoritmo. Este no se encuentra formado únicamente de las funciones mencionadas anteriormente. Tanto al inicio, como entre medias y al final, se realizan obtenciones de datos, cribados de los resultados y demás procedimientos cuyo fin es el de presentar resultados con las predicciones más altas sin interferencias de datos superfluos que pueden ralentizar el funcionamiento del algoritmo o enturbiar los resultados. Con esto se busca cumplir el RNF-3, relacionado con la eficiencia del sistema; RNF-4 que gestiona la escalabilidad (buscando no saturar la memoria del servidor para poder gestionar un mayor volumen de peticiones); y el RNF-6 que mira por la precisión de las recomendaciones.

- Obtención de datos
  - El funcionamiento del sistema comienza obteniendo los datos necesarios para el correcto funcionamiento.
  - De inicio se necesita reconocer al usuario que está utilizando el sistema. Esto se realiza comprobando si existe el *user_id* introducido en la base de datos. En caso de que no ser así, se solicita de nuevo al usuario.

    - Una vez se verifica, se solicitan tanto los integrantes del grupo como la ciudad donde se desea que se encuentre el restaurante. Ambos parámetros se proporcionan en una lista cerrada de opciones, por lo que no da paso a errores humanos más allá de que el usuario se confunda a la hora de seleccionar dentro de las opciones dadas.
- Formación del grupo
    - A continuación, se procede a formar el grupo uniendo a los amigos seleccionados con el usuario principal.
    - Se crea en la base de datos el grupo de forma temporal.
    - Al grupo se le establecen las relaciones con los restaurantes teniendo en cuenta las valoraciones de los integrantes.
- Obtener usuarios afines
    - Se obtiene una lista con todos los usuarios que pueden ser afines al grupo en base a la función explicada anteriormente.
    - La lista está formada por el *user_id* de cada usuario y su valoración media.
- Similitud
    - Se genera un grado de similitud para cada candidato a ser un usuario afín.
    - La lista obtenida se sesga obteniendo los 10 usuarios con mayor grado de similitud con el grupo.
- Predicción
    - En este punto se genera la lista con los restaurantes que son candidatos a ser recomendados.
    - Cada restaurante que forme parte de la lista será identificado mediante su *business_id* y se guardará la predicción determinada.
- Saneamiento
    - Una vez obtenida una lista con todos los restaurantes y sus predicciones, se procede a sanear dicha lista dejando como resultado únicamente los 5 restaurantes con predicción más alta.
    - Para el saneamiento, primero se eliminan los restaurantes repetidos, haciendo una media de todas las predicciones, resultando una única aparición con la valoración media calculada. A continuación, se seleccionan los únicamente los restaurantes que se encuentran en la ciudad indicada por el usuario, descartando el resto. Finalmente, se ordenan de mayor a menor en función a la predicción, devolviendo máximo 5 restaurantes.
    - Los restaurantes resultantes se mostrarán al usuario como resultado de la predicción.

A continuación, se mostrará un diagrama de secuencia que servirá para clarificar el comportamiento del sistema y la interacción de los diferentes actores entre ellos:

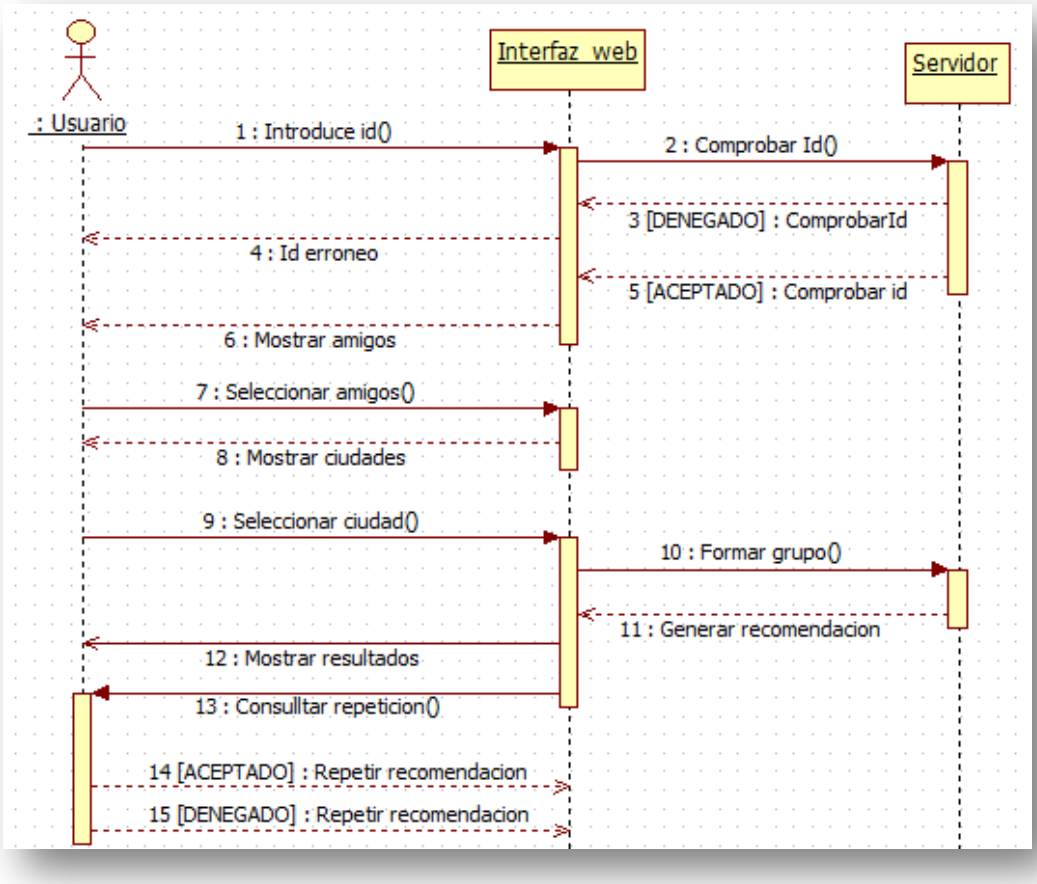


Diagrama de secuencia seguido por el sistema

3.4.2. Pruebas del algoritmo

Una parte fundamental de un sistema de recomendación son las pruebas para comprobar la veracidad de los resultados. En este caso se busca obtener al menos un 65% de acierto en los resultados según lo planificado en el RNF-6.

Para probar el sistema de recomendación, he optado por utilizar el conocido método de pruebas *Leave-One-Out Cross-Validation* [15] y adaptarlo a este caso. Este método consiste en obtener un conjunto de datos, entrenarlos todos menos uno y utilizar este último como conjunto de pruebas, repitiéndose con cada punto de datos. Esto proporciona un elevado número de iteraciones y computacionalmente es considerado como intensivo, pero proporciona una evaluación exhaustiva del modelo. Es utilizado principalmente para pruebas relacionadas con *Machine Learning* e inteligencia artificial.

En el caso de este sistema de recomendación, el conjunto de datos estará formado por los restaurantes que han sido reseñados por parte del grupo. Se obtendrán los restaurantes dejando 1 de ellos fuera de los datos, se ejecutará el algoritmo y se comprobará si finalmente forma parte de la recomendación obtenida.

El conjunto de datos es obtenido a partir de un conjunto de usuarios que es formado de manera aleatoria. De cada usuario se forma el grupo con todos los usuarios con los que tiene una relación de amistad.

Para llevar el sistema de pruebas a cabo, se ha creado un fichero aparte del código, que conecta con este. En él se ha establecido todo lo necesario para simular una ejecución de forma automática de forma repetida. Esto puede verse en el fichero *SistemaPruebas.py* del repositorio de Github.

El sistema de pruebas está estructurado de la siguiente forma:

```
usuarios = obtenerUsuariosAleatorios()
mientras (haya más usuarios) {
	grupo = formarGrupo()
	restaurantes = obtenerRestaurantes()
	usuariosAfines = obtenerUsuariosAfines()
	mientras (haya restaurantes por probar) {
		eliminarRelacion()
		generarRecomendacion()
		si (restauranteEsRecomendado())
			acierto + 1
		restaurarRelacion()
	}
}
calcularMediaFinal()
```

- A partir del conjunto de usuarios aleatorios, se forman los grupos de la manera mencionada anteriormente. De cada grupo se obtienen los restaurantes y los usuarios afines utilizando las funciones, de forma independiente, utilizadas en el algoritmo principal del sistema.
- Una vez localizados los usuarios afines, ya se puede comenzar a extraer restaurantes de la lista de restaurantes reseñados. Es importante obtenerlos al inicio ya que, al ser un algoritmo determinista, podía darse el caso de que fuese imposible que apareciese el restaurante extraído debido a que no habría ningún usuario afín que lo pudiese recomendar.
- Una vez extraído el restaurante, se ejecuta el algoritmo de recomendación. Al finalizar, se comprueba que el restaurante aparezca en el conjunto de restaurantes resultado (5 restaurantes con una predicción más alta).
- En el caso de aparecer, se sumaría al contador de casos de éxito. Finalmente, se calcula el porcentaje de casos de éxito en cada ronda de 50 restaurantes

comprobados. En un inicio se pensó tomar cada usuario/grupo de forma atómica y que las rondas fuesen de 50 o más restaurantes. Pero fue descartada ya que el algoritmo no determina sus resultados en función del número de restaurantes que tiene relacionados cada grupo. Los resultados se comprueban en grupos de 50 restaurantes, y si a mitad de un grupo se realiza ese corte, el próximo restaurante formará parte de la siguiente ronda. La única ronda que puede contar con menos de 50 restaurantes es la ronda final debido a que se acaben los usuarios del conjunto inicial aleatorio.

- Una vez finalizado el conjunto inicial aleatorio de usuarios, se realizará una media con todos los grupos de evaluaciones en un total de 50 restaurantes, obteniendo un grado final de acierto del sistema.
- Cabe destacar que cada vez que se comprueba si un restaurante es recomendado, se vuelve a añadir a la lista de restaurantes dicho restaurante, y se restaura la relación con la valoración que se dio en un inicio.

3.5. Elección de la tecnología

En este apartado se explicarán los motivos de la elección de cada tecnología utilizada en el sistema. Dentro del sistema podemos destacar 3 partes principalmente: la base de datos, el *backend* (donde se encuentra el algoritmo) y el *frontend*.

Cada parte tiene su objetivo particular y debe cumplir unos u otros requisitos. Es por ello que se ha utilizado diferentes tecnologías enfocadas a cada uno de ellos de forma individualizada:

3.5.1. Elección de una base de datos

La base de datos es una de las partes del sistema principales del sistema como se ha reflejado a lo largo de todo el documento. Su principal función es la de almacenar y gestionar de forma eficaz los datos preservando su integridad.

Dentro de todo el abanico de posibilidades, la tecnología utilizada es Neo4J. Esta se trata de un tipo de base de datos del tipo NoSQL, para ser más precisos, basada en grafos. Este tipo de bases de datos presentan estas ventajas:

- Que sea basada en grafos ayuda a cumplir el requisito RNF-3 procesando de forma extremadamente rápida las consultas. Esto se debe a que la velocidad de este tipo de bases de datos depende únicamente del número de relaciones de los nodos implicados, no del tamaño del conjunto de datos. Una base de datos relacional degrada el rendimiento a medida que aumenta la complejidad de las consultas.
- Por otro lado, este tipo de bases de datos presenta una ventaja a la hora del diseño ya que en mucho más intuitivo que una base de datos SQL.

- En el caso de querer modificar la estructura de la base de datos, no es para nada costosa debido a su flexibilidad.
- Facilidad para el aumento de nodos. Al tratarse de una base de datos basada en reseñas, los usuarios pueden crear cuantas quieran y sin necesidad de afectar negativamente a la eficiencia de la base de datos.
- En comparación con una base de datos relacional, a la hora de incluirse relaciones es mucho más sencillo ya que se pueden ahorrar una gran cantidad de *joins* (repercutiendo en el espacio que ocupa y la complejidad que puede representar).
- A la hora de realizar consultas, la anidación entre ellas es mucho más sencilla debido a la propia sintaxis del lenguaje. En cypher (lenguaje en el que está basado Neo4J) las consultas se establecen siguiendo un orden lógico utilizando en la siguiente consulta lo devuelto en la anterior.
- A la hora de analizar una red o realizar recomendaciones, como puede ser la de este sistema, un grafo es identificado como la estructura de datos idónea.

3.5.2. Elección de la tecnología para el backend del algoritmo

Para la creación del *backend*, he optado por utilizar Python. Python es un lenguaje de alto nivel que se ha vuelto muy popular en los últimos años. Esto se debe a que cuenta con unas características particulares que se adaptan a multitud de proyectos. En mi caso, me ha aportado lo siguiente:

- Alta compatibilidad con diferentes tecnologías. Python cuenta con una gran compatibilidad con Neo4J y con diferentes tecnologías que pueden ser utilizadas para la creación de un *frontend*. En este caso se utilizará Flask[1], que se trata de una extensión de Python. En el apartado 3.5.3 hablaré sobre ella.
- Multitud de bibliotecas disponibles. Para la conexión con el resto de tecnologías, se pueden encontrar diferentes librerías, lo que lo hace muy interesante como intermediario entre ellas mientras desempeña las funciones de *backend* y gestión del algoritmo. Además, la biblioteca estándar de Python es muy amplia cubriendo una gran cantidad de tareas básicas, que son las necesarias en este proyecto.
- Sintaxis legible. Python, al ser de alto nivel, tiene un lenguaje fácilmente entendible para una persona. Además, es de fácil entendimiento y rápido de aprender. En mi caso, es la primera vez que utilizo este lenguaje, y ha sido muy sencillo de aprender a utilizarlo y he encontrado muchas facilidades a la hora de encontrar documentación y explicaciones de cómo utilizarlo.

[1] https://flask.palletsprojects.com/en/3.0.x/

- Libre y de código abierto. Esto alimenta lo escrito en el anterior punto ya que es fácil encontrar documentación sobre el lenguaje. Por otro lado, se presenta la ventaja de no requerir licencia para nada de lo que he necesitado utilizar. Al tratarse de un proyecto personal de un alcance limitado, no se destinan más recursos de los que tengo en mi poder actualmente.

Como desventaja destaca la velocidad del procesado y la gestión de la memoria, ya que es uno de los puntos débiles de este lenguaje. Aun así, se ve suplida con la utilización de las librerías correctas, además de las funciones básicas que integra el propio lenguaje, y la tecnología utilizada en la base de datos.

3.5.3. Elección de la tecnología para fronted

La última parte de este proyecto es el *frontend*. Para su creación se ha optado por utilizar Flask. Flask es un *microframework* web de Python caracterizado por su sencillez de utilización y aprendizaje. Este es uno de los motivos por los cuales he decidido utilizarlo en este proyecto ya que no me considero ningún experto en *frontend*. Flask tiene diferentes ventajas por las cuales.

- El principal motivo es su simplicidad y facilidad de uso. Se trata de un framework minimalista y muy ligero en cuanto a procesamiento, por lo que es perfecto para mi proyecto. Con esta parte, únicamente busco el mostrado de los datos y crear una interfaz gráfica para el usuario que sea sencilla y fácil de utilizar.
- Es rápido de desarrollar debido a su simplicidad. El sprint dedicado a esta parte del proyecto se caracteriza frente al resto en cuanto al corto tiempo disponible, por lo que no se puede dedicar demasiado esfuerzo en aprender una tecnología tediosa o más completa. Este tema se desarrollará más en el apartado 3.7.3 .
- Además, para la depuración proporciona una gran cantidad de mensajes detallados con los errores y una consola interactiva. Como buen principiante que soy en este tema, que proporcione facilidades en este tema es una gran ayuda.

3.6. Arquitectura de la aplicación

Para la arquitectura del código, se ha optado por ordenar la información siguiendo un Modelo-Vista-Controlador (MVC). Al estar utilizando el lenguaje de programación Python, es complicado de llevar a cabo debido a la inexistencia de las interfaces y que no se trata de un lenguaje orientado a objetos al uso. El código del proyecto está distribuido entre diferentes archivos, distribuidos en 2 grandes módulos:

- *Backend*
    - En esta sección encontramos todos los archivos relacionados con la parte que se encarga del algoritmo y la lógica del sistema. En este módulo nos encontramos con los diferentes archivos:

        - Algoritmos.py
            - En este archivo se encuentra el corazón del sistema. Se guardan los métodos encargados de generar las predicciones, similitud y obtener los usuarios afines
        - DataExchange.py
            - Se encarga de albergar los métodos necesarios para el paso de información con la web o cualquier otro método que esté relacionado con el *frontend.*
        - Grupo.py
            - Almacena la clase Grupo y se encarga de dar soporte a las funcionalidades básicas de estos. Se asemeja a una clase *model* en un sistema con arquitectura *Model-View-Controller*
        - mainCodigo.py
            - Se trataba de la clase principal cuando no existía una interfaz de usuario. Se encargaba de dar cuerpo al sistema y arrancarlo. Actualmente almacena algún método que da base al funcionamiento del programa. Se asemeja a una clase *controller* en un sistema con arquitectura *Model-View-Controller*.
        - Restaurante.py
            - Almacena la clase Restaurante y se encarga de dar soporte a las funcionalidades básicas de estos. Se asemeja a una clase *model* en un sistema con arquitectura *Model-View-Controller*
        - SistemaPruebas.py
            - En ella se encuentra todo lo relacionado con el sistema de pruebas automatizado. Almacena los métodos tanto para generar las recomendaciones accediendo al resto de ficheros, como para imprimir y guardar los resultados.
        - Usuario.py
            - Almacena la clase Usuario y se encarga de dar soporte a las funcionalidades básicas de estos. Se asemeja a una clase *model* en un sistema con arquitectura *Model-View-Controller*
        - viewMain.py
            - Clase relacionada con la interfaz gráfica de usuario. Pertenece al siguiente módulo, pero se encuentra alojada en este por motivos de facilidad de comunicación entre ambos. Se asemeja a una clase *controller* en un sistema con arquitectura *Model-View-Controller*.
- Web
    - Este módulo contiene todo lo relacionado con la interfaz de gráfica de usuario. Los ficheros que se encuentran aquí son los siguientes:

- layout.css
  - Almacena los estilos y colores utilizados en la interfaz.
- *Templates*
  - Dentro de esta carpeta, encontramos las plantillas que determinan como están estructurados las páginas de la web. Estas son las vistas del sistema.

NOMBRE	DESCRIPCIÓN
404.html	Da forma a la ventana que se es redirigido en caso de presentar un error en la URL introducida o intentar acceder a una de forma incorrecta.
AmigosCiudad.html	En ella se encuentra la ventana en la cual se elige a los integrantes del grupo y la ciudad en la que debe estar el restaurante recomendado.
Index.html	Muestra la ventana inicial del sistema, donde se debe identificar el usuario.
IndexIdErroneo.html	Igual que el anterior, pero denota que el usuario ha introducido un id erróneo.
Layout.html	Plantilla que sirve al resto para regirse y seguir el mismo estilo en todo el sistema.
MostrarResultados.html	En él se estructura la ventana donde se muestran los resultados de la recomendación. Además, alberga el botón para volver al inicio y empezar de nuevo la recomendación.

- Index.py
  - Comunica la web con el *backend*, además de leer las url que son accedidas lanzando los html correspondientes.

3.7. Desarrollo

Como mencioné en el apartado 3.2, para la planificación de este proyecto se ha optado por seguir una metodología ágil basada en Scrum. Para dividir los periodos de trabajo he decidido realizar 4 sprints marcados por lo desarrollado en cada uno. Dado que se trata de un proyecto especial en el que el tiempo a dedicar es diferente a un proyecto al uso, los tiempos dedicados a cada sprint son muy amplios y para nada lineales. En los siguientes apartados se explicará de forma clara como se ha dividido el trabajo, las tareas dedicadas a el desarrollo de cada *User Story* y la adaptación que se ha conseguido para adaptar los objetivos y el trabajo a desarrollar con el tiempo como mencionaba anteriormente.

En cuanto a las tareas, todas las tareas han sido diseñadas y estimadas al inicio de cada sprint por el equipo de desarrollo como indican los principios de Scrum. Para la estimación se ha tomado el método "Días-hombre disponibles y productividad diaria". Pienso que es el más indicado ya que en él se tiene en cuenta que en un día de trabajo se realizan más tareas aparte de las dedicadas al proyecto. Dadas las características del proyecto, este método encaja perfectamente a la situación de este proyecto, aunque no fuese ese el motivo por el cual fue diseñado. Dependiendo del momento en el que se ubique cada sprint se ha podido dedicar más días al proyecto, pero en todos se ha dedicado una media de 2h/día al desarrollo del proyecto, contando con que cada día se trabaja un total de 3 horas. El resto de las horas se han dedicado a la investigación necesaria para el propio proyecto.

- Para la estimación se ha utilizado un sistema de "*Poker Game*" donde las posibilidades son: 1, 2, 3, 5, 8 y 13.
- La priorización de las tareas se realizará en base a "Alto", "Medio" y "Bajo".

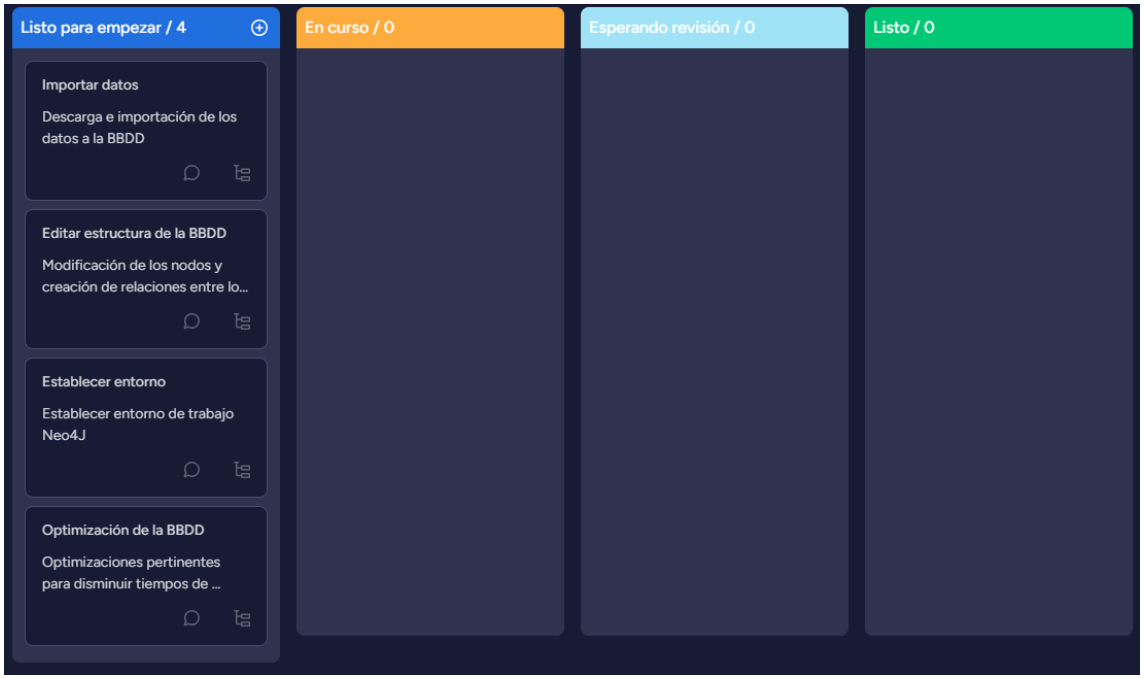
En cuanto al sistema de "*Poker Game*", se trata de un sistema de estimación de tiempo que necesita cada tarea. Este se realiza antes de comenzar cada sprint, con las tareas del propio sprint, por el equipo de desarrollo. En este caso, lo he realizado yo, como único integrante.

3.7.1. Sprint 1

En el primer sprint se realizaron las tareas relacionadas con la creación y la puesta a punto de la base de datos. El tiempo dedicado a este sprint fue de 2 días por semana. Dado que no hay ninguna historia de usuario dedicado a ello, se han creado un paquete de tareas dedicadas a ello. Es un paso esencial del proyecto, por lo que se requiere un sprint únicamente para él. Las tareas requeridas para ello son las siguientes:

- Establecer entorno
    - Descripción: Establecer el entorno de trabajo Neo4J donde se establecerá la base de datos y desde donde se manejará de forma directa.
    - Estimación: 1 día
    - Prioridad: Alta
- Importar datos
    - Descripción: Importación de los datos del *dataset* a la base de datos.
    - Estimación: 3 días
    - Prioridad: Alta
- Editar estructura de la BBDD
    - Descripción: Tratamiento de los datos introducidos y modificación de la estructura de los nodos y relaciones.
    - Estimación: 3 días

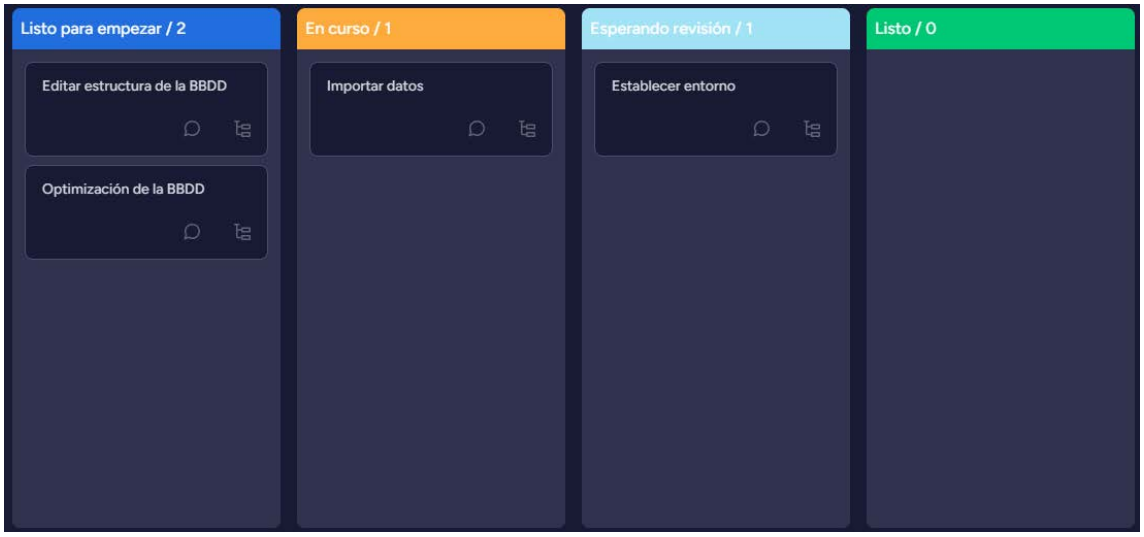
- Prioridad: Media



Kanban inicial Sprint 1

En cuanto a la tarea "Importar datos" modificó toda la planificación teniendo que ampliar los plazos del sprint. Al tratarse de un conjunto de datos tan completo, trajo consigo una gran cantidad de problemas para la importación de los datos.

- Primero se consideró el particionado de los datos, pero finalmente se optó por la inserción de estos mediante consola. Llegar hasta este punto y llevarlo a cabo acabó ampliando el tiempo dedicado de 3 días como estaba estimado a 1 mes de trabajo, por lo que fueron finalmente 10 días.



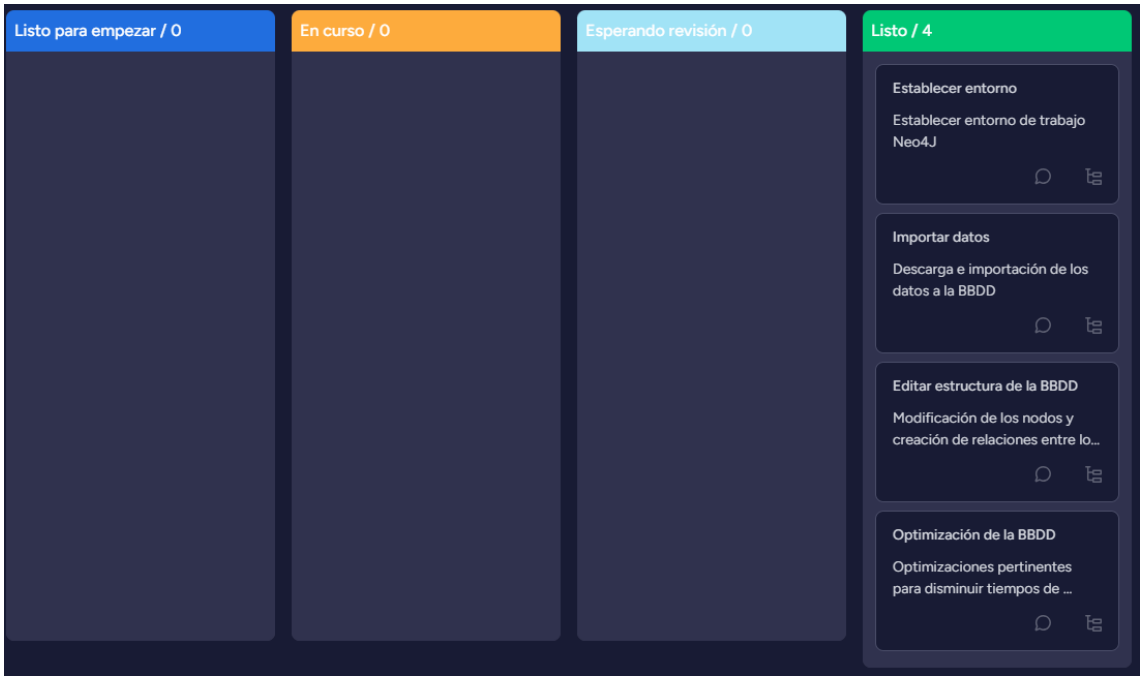
Kanban intermedio Sprint 1

En cuanto a la última tarea, abarca tanto la modificación de la disposición de los nodos y la creación de las relaciones, como las técnicas empleadas para la optimización de la base de datos y las consultas.

- La transformación más importante es la de las reseñas. Cuando se introdujeron en la base de datos, se hizo como nodos. En un inicio, la base de datos estaba formado únicamente por los nodos: "Usuario", "Restaurante" y "*Review*". Para una mayor facilidad y reducir las relaciones intermedias entre los nodos Usuario y Restaurante, los nodos *Review* se han convertido en relaciones "*Reviews*". Esta nueva relación conecta a los usuarios y los restaurantes. Se forma a partir de que un nodo cualquiera *Review* contiene como atributos *user_id* (que indica qué usuario ha realizado la reseña) y un *business_id* (que hace referencia al restaurante reseñado).
  Para el funcionamiento del sistema únicamente nos interesan esos 2 atributos y el atributo "*stars*" que hace referencia a la valoración dada por el usuario siendo 5 la valoración máxima y 1 la mínima. Como inicialmente el atributo *stars* estaba definido como una cadena de caracteres, es necesario transformarlo a un *integer*.
- Se crea la relación *Friends* que determina la relación de amistad entre 2 nodos Usuario. Para formar esta relación, se parte del atributo "*Friends*" que tiene cada Usuario.
  Para obtener la lista de forma correcta para operar con ella se hace uso del plugin APOC. De todas las funcionalidades que tiene el plugin, únicamente vamos a utilizar la que tiene para desenrollar listas, para la lista de strings en este caso. Una vez se haya desenrollado, se crea la relación que, a diferencia de la anterior relación, esta no tiene ningún atributo. En el momento que se ha ido a realizar esta transformación, se ha observado que era inviable esto debido al tiempo que se tardaba en realizar la instrucción. Para poder llevarlo a cabo, he utilizado una desnormalización: todos los nodos tienen un id propio interno del sistema, por lo que lo he guardado como atributo del nodo. A continuación, a este nuevo atributo le he creado un índice sobre el que trabajar. Con esto se ha conseguido reducir el tiempo de cómputo de prácticamente 1 minuto por cada 10 usuarios a aplicarse a todos los nodos en casi 2 minutos. La consulta utilizada es la siguiente:

```
MATCH (u:Usuario)
UNWIND apoc.text.split(u.friends, ",") AS f_id
MATCH (f:Usuario {user_id: trim(f_id)})
CREATE (u)-[:FRIEND]->(f)
```

- El índice anterior no es el único creado para la base de datos. Con el fin de optimizar las consultas, se han creado índices en los atributos "*user_id*" y *"business_id*" de los nodos Usuario y Restaurante respectivamente. En los nodos Grupo no se ha realizado debido a que se trata de un nodo temporal y que no se espera tener una gran cantidad de ellos de forma simultánea. En el caso de que el tráfico de la consulta sea mayor del esperado, se puede considerar crear uno sobre el atributo identificador. No haría falta crear uno cada vez que se instancie un nodo, se mantiene, aunque no haya ningún grupo en la base de datos.



Kanban final Sprint 1

Finalmente, tras una revisión y visto bueno del *Product Owner*, las tareas, que se encontraban como terminadas una vez se finalizasen, pasan a estar cerradas ya que dejan como resultado una base de datos completamente funcional y preparada para dar soporte al sistema.

3.7.2. Sprint 2

El segundo sprint está marcado por la creación del algoritmo y el funcionamiento interno del sistema. En él se desarrollarán las historias de usuario marcadas en el apartado 3.2. Al igual que en el apartado anterior, se han establecido unas tareas marcadas por el equipo de desarrollo para conseguir el objetivo del sprint. En este caso si están

relacionadas con las historias de usuario en cuestión. Las tareas que se llevarán a cabo son:

- US4 – Visualización de los resultados
  - Generación de proyecto
    - Descripción: Al ser la primera tarea, también se debe crear el proyecto en Python y establecer lo necesario para realizar la conexión con la base de datos.
    - Estimación: 3
    - Priorización: Alta
  - Creación y borrado de perfil de grupo
    - Descripción: Creación de grupo con los integrantes, obtener reseñas del grupo y borrado del mismo una vez finalizada la recomendación.
    - Estimación: 1
    - Priorización: Alta
  - Obtención de usuarios afines
    - Descripción: Generación de función para obtener los usuarios afines al grupo. Únicamente se debe quedar en la lista los 20 más afines.
    - Estimación: 2
    - Priorización: Alta
  - Asignar grado de similitud
    - Descripción: Formalizar función de similitud mediante consulta a la base de datos y asignársela a los usuarios afines únicamente resultando los 10 más similares.
    - Estimación: 3
    - Priorización: Alta
  - Generar predicción y mostrado de datos
    - Descripción: Formalizar la función de predicción mediante consulta a la base de datos y depurar la lista generada devolviendo los 5 restaurantes con mayor predicción.
    - Estimación: 5
    - Priorización: Alta
- US5 – Volver a generar recomendación
  - Devolver al usuario al paso inicial
    - Descripción: Dar la posibilidad de volver a la generación del grupo al usuario o finalizar la recomendación.
    - Estimación: 1
    - Priorización: Baja

Kanban inicial Sprint 2

Tras la estimación, se ha determinado que la duración del sprint es de 15 días efectivos, lo que equivale a poco más de 1 mes de días naturales ya que para este sprint he podido trabajar en el proyecto 3 días/semana. A continuación, se detalla el proceso seguido en cada historia de usuario y como de fiel ha sido la estimación con la realidad:

- US4 – Visualización de los resultados
  - Esta historia de usuario es la más larga de este sprint por lo que abarcará la mayor parte del tiempo de la implementación.
  - En esta historia de usuario se creó el proyecto de 0 estableciendo la conexión con la base de datos. Se dio una estimación amplia ya que se trata del primer proyecto que realizo en Python. Tras completar la tarea asignada, se observó que la estimación fue correcta completándose en el tiempo esperado.
    En esta tarea además se estableció todo lo relacionado con las estructuras de datos y clases necesarias para el correcto funcionamiento del sistema, como puede ser una clase Usuario, Restaurante o el fichero dedicado al algoritmo.
  - En la primera tarea se abarca todo lo relacionado con la gestión de los usuarios. Se precisa de la creación del grupo, con su identificador único y las relaciones “GReviews” y “GDescartados”. Se comparan todas las reseñas realizadas por los integrantes del grupo. A los restaurantes que tienen todas las reseñas positivas se les crea una reseña “GReviews” con la media de estrellas, y a los que no, una “GDescartados”.
    Al final de la recomendación se realiza el borrado del grupo mediante una simple consulta a la base de datos. De esta forma prevalece la integridad de la base de datos como se requiere. Al borrar el grupo, la base de datos queda exactamente igual a como estaba antes de la

recomendación. La implementación se ajustó correctamente al día estimado para esta tarea.

Al comenzar la siguiente historia tarea, el tablero Kanban se encuentra de la siguiente forma:

Kanban intermedio Sprint 2

Cómo podemos observar, se encuentran terminadas las tareas ya implementadas, al igual que las historias de usuario que tienen asociadas. En el lado contrario se encuentran en la columna inicial las que aún no han sido implementadas. La tarea que se encuentra en progreso se trata de la siguiente:

- Para la obtención de usuarios afines se implementó la consulta mencionada en el apartado 3.4. A pesar de la simplicidad del Índice de Jaccard, la obtención de la *query* es más compleja. Una vez obtenido, únicamente nos quedamos con los 20 que más restaurantes en común comparten y los almacenamos en una estructura de datos para su futuro uso. La tarea se realizó en el tiempo estimado.
- Esta es una de las tareas críticas del algoritmo. Aquí se obtienen cuanto similares son los usuarios afines y el grupo de amigos. En el momento en el que se estaba probando, se observó el problema mencionado cuando se habló de esta función. Tras observar las diferentes soluciones, se llegó a la conclusión de que lo más optimo y sencillo de implementar es la solución aportada de sustituir los 0's problemáticos por 0.01. De esta forma se consiguió cumplir con los plazos determinados en la estimación. Finalmente se almacenan los 10 usuarios más similares al grupo en la estructura de datos pertinente.

- Finalmente, en esta tarea se implementa la función de predicción mencionada. No solo se formalizaron las *querys* necesarias si no que se manejó el cribado para obtener los restaurantes con las valoraciones más altas. El manejo de esto supuso un tiempo mayor de lo esperado alargándose en 2 días el tiempo estimado al comienzo del sprint. Probar que no se devolviesen restaurantes repetidos y cómo manejarlos, manejar las estructuras de datos con los restaurantes y modificar las consultas a medida que lo necesite el algoritmo fue más tedioso de lo esperado.

- US5 – Volver a generar recomendación
  - Esta tarea es muy sencilla y se completó sin mayor problema. Recoge el proceso entero en un bucle. La tarea se cumplió en el tiempo esperado.

Una vez finalizado el sprint se revisó dando el visto bueno a las tareas finalizadas. El tiempo esperado total era de 15 días y al final se cumplió en 17 días. En cualquier proyecto hubiese supuesto dejar alguna tarea sin realizar o a medias, pero dada las condiciones de este, se pudo realizar sin problemas. Al tener más flexibilidad en los días de trabajo, me he podido permitir ese lujo.

El resultado del Kanban tras el visto bueno de las tareas es el siguiente:

Kanban final Sprint 2

3.7.3. Sprint 3

El tercer sprint de este proyecto tiene la finalidad de finalizar la implementación de las funciones básicas del sistema e implementar una interfaz de usuario con la que manejarlo. Al ser la primera vez que realizaba una, se optó por ser pesimistas a la hora

de estimar los plazos de las tareas de frontend. Las primeras tareas se han estimado con plazos más largos que los últimos ya que se espera que al final tuviese una mayor soltura con la tecnología aplicada. Las tareas se han establecido de la siguiente forma:

- US1 – Validación del usuario
    - Comprobación del usuario mediante el identificador
        - Descripción: El usuario debe ser capaz de introducir su identificador personal y el sistema debe ser capaz de comprobarlo. En el caso de pertenecer al sistema, se dará por validado. En caso contrario se denegará el acceso al sistema y se le dará la oportunidad de introducir el identificador de nuevo indicándole el error.
        - Estimación: 2
        - Priorización: Alta
- US2 – Selección de amigos
    - Mostrado de lista de amigos y registro de seleccionados
        - Descripción: Obtención de la lista de amigos de la base de datos mediante consulta y mostrado de ella al usuario. Se debe permitir una selección múltiple.
        - Estimación: 1
        - Priorización: Alta
- US3 – Selección de ciudad
    - Mostrado de ciudades disponibles y registro de la seleccionada
        - Descripción: Obtención de lista de ciudades que pueden ser recomendadas al grupo de usuarios y mostrado de ella al usuario. Se debe permitir una única elección.
        - Estimación: 1
        - Priorización: Media
- Creación de plantilla base
    - Descripción: Crear una plantilla base en la que se van a basar las diferentes pestañas de las que está formada la web.
    - Estimación: 2
    - Priorización: Alta
- Generación de interfaces
    - Descripción: Generación del diseño de cada una de las páginas web.
    - Estimación: 5
    - Priorización: Alta
- Establecer paso de información
    - Descripción: Guardado de la información proporcionada por el usuario para el funcionamiento del sistema, comunicación con el *backend* y envío de los resultados a la interfaz.
    - Estimación: 3
    - Priorización: Alta
- Pruebas

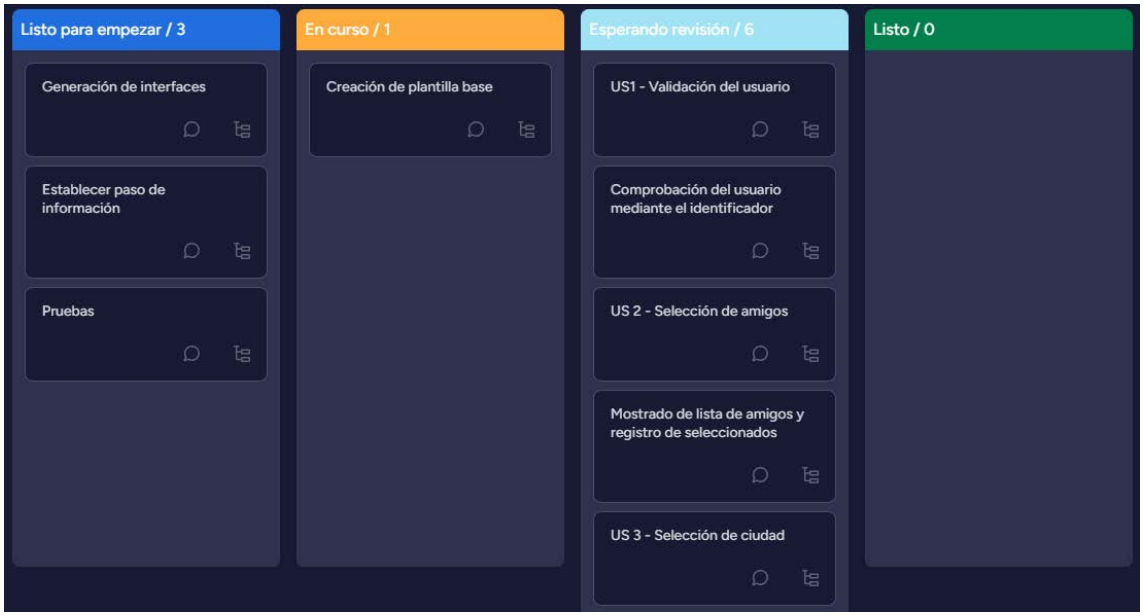
- Descripción: Pruebas de funcionamiento de la web y tiempo reservado para cambios.
- Estimación: 3
- Priorización: Alta

Kanban inicial Sprint 3

Una vez establecidas las tareas a realizar en el sprint, se comienza la implementación de ellas. Se espera que el tiempo empleado sea de 17 días, en torno a 1 mes natural de trabajo.

- US1 – Validación del usuario
  - Para la esta tarea se fue demasiado pesimista ya que se tardó 1 día menos de lo estimado. Al crear el objeto de tipo Usuario, se obtiene el nombre del usuario. Al realizar la llamada al método encargado de indicar si existe el usuario o no, comprueba que el nombre no sea nulo. En caso de contener un nombre, se entiende que el usuario existe en la base de datos, por lo que se acepta. En caso de que sea nulo, se deniega el acceso al sistema.
- US2 – Selección de amigos
  - Para la tarea destinada a esta historia de usuario se empleó el tiempo estimado. Primero se obtuvo todos los amigos que tiene el usuario mediante las relaciones "*Friend*". Aunque se identifique cada usuario por su *user_id*, para una mayor usabilidad, se muestra el nombre que tiene almacenado cada usuario. Una vez seleccionados, se almacenan guardando su id. Más adelante se utilizarán para la formación del grupo.
- US3 – Selección de ciudad

- Para esta historia de usuario, se aprovechó la soltura obtenida al realizar una tarea similar a otra ya hecha (la anterior). Es mostrar únicamente las ciudades que pueden tener restaurantes recomendables ya que, si no, se mostrará una lista con cientos de ciudades (la mayoría inútiles). De esta forma es mucho más sencillo para el usuario encontrar la ciudad que desean. Finalmente se cumplió el plazo estimado para su implementación.



Kanban intermedio Sprint 3

- La creación de la plantilla base fue más sencillo de lo pensado, por lo que se realizó en 1 día en vez de 2. El soporte que se puede encontrar sobre Flask es muy amplia y ayuda a acelerar los plazos de trabajo.
- La siguiente tarea se encuentra basada en la anterior. Se crearon las diferentes páginas web basándose en un estilo único para facilitar tanto la usabilidad del sistema como la implementación. Cada página es distinta a la anterior, requiriendo diferentes objetos que utilizar en cada una. Aprender a utilizar html fue un desafío al principio, pero finalmente se pudo dominar. Además, al utilizar flask, se necesitaba mezclar con Python.
- Una vez finalizada la implementación del diseño de cada una, se empleó un día para refinar los diseños y hacer que queden más atractivas.
- Una vez finalizado el diseño, se realizó la comunicación con el *backend* y cómo se conseguían manejar los datos. Flask hace que parezca sencillo este paso y se consiguió realizar la tarea en los plazos establecidos. Por otro lado, fue necesario modificar mínimamente la estructura del código principal para realizar la comunicación de los módulos. Se generó el fichero *DataExchange.py*

que contenía las funciones necesarias para la comunicación con cada una de las páginas manteniendo la lógica del sistema.

- Finalmente se reservó un tiempo de pruebas y posibles cambios. De esta tarea solo se necesitó 1 día ya que las pruebas salieron según lo esperado y los cambios que hubo que realizar fueron mínimos.

Para finalizar el sprint, se realizó una *review* con el *Product Owner* de este proyecto mostrando el funcionamiento del producto finalizado. Tras dar el visto bueno, se dio por cerrada la implementación del sistema y se cerraron todas las tareas del sprint, con el punto positivo de haberlo hecho en menor tiempo del estimado. El resultado del Kanban fue el siguiente:

Kanban final Sprint 3

3.7.4. Sprint 4

El último sprint del proyecto está dedicado a la creación de un sistema de pruebas. El objetivo de este sprint es el de comprobar la eficacia de las recomendaciones del sistema resultado de los sprints anteriores. En este sprint se dispuso de los 7 días de la semana para la implementación. Para ello se plantearon 2 tareas:

- Generación de algoritmo de prueba
  - Descripción: Generar el código necesario para generar de forma automáticamente una recomendación para un caso aleatorio.
  - Estimación: 5
  - Priorización: Alta
- Automatizado y guardado de pruebas

- Descripción: Obtención de n casos aleatorios, siendo n el número de pruebas que se quiere realizar, y guardado de datos en un fichero de tipo Json.
- Estimación: 2
- Priorización: Media

Kanban inicial Sprint 4

Dado que se disponía de 7 días seguidos para los 7 días de implementación, se esperaba poder realizarlo en 1 semana.

- La primera tarea era la más crítica ya que de ella depende el sistema de pruebas completo. Sin ella no se podría probar el sistema prácticamente. Para implementarlo, se comenzó haciendo todas las llamadas posibles a los métodos que están incluidos en el sistema de forma interna. Pero al desarrollarlo se observó que había métodos que no se podían acceder ya que desencadenarían acciones que en ese momento no nos interesarían ya que para las pruebas se necesitan unas condiciones especiales, como la elección de los amigos.
  Se ha optado por seleccionar todos los amigos del usuario recibido y realizar una recomendación quitando uno de los restaurantes para posteriormente devolvérselo al grupo. Al quitarlo, se genera la recomendación y se comprueba si se encuentra dentro de los recomendados tal y como se ha indicado en el apartado 3.4.2.
  Finalmente, se ha conseguido realizar la tarea en únicamente 3 días en vez de los 5 estimados.
- Para la última tarea, se necesitó los 2 días estimados ya que únicamente era necesario generar el bucle y la forma de contabilizarlos aciertos. Finalmente se exportan los datos a un Json.

Finalmente se terminó la implementación acelerando los plazos estimados en 2 días.

Kanban final Sprint 4

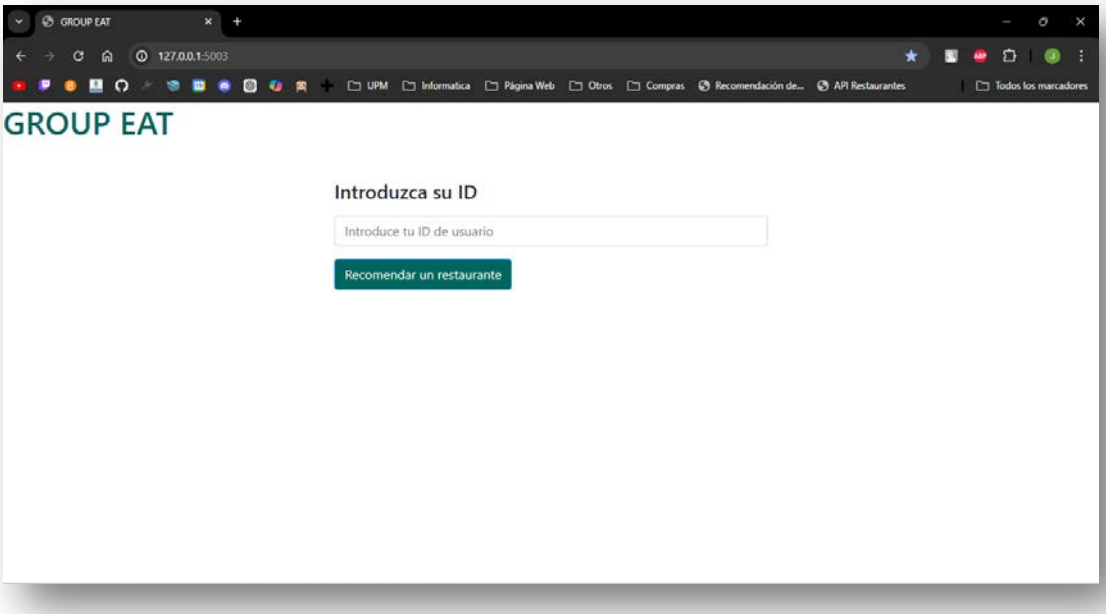
4. Resultados

4.1. Resultados obtenidos

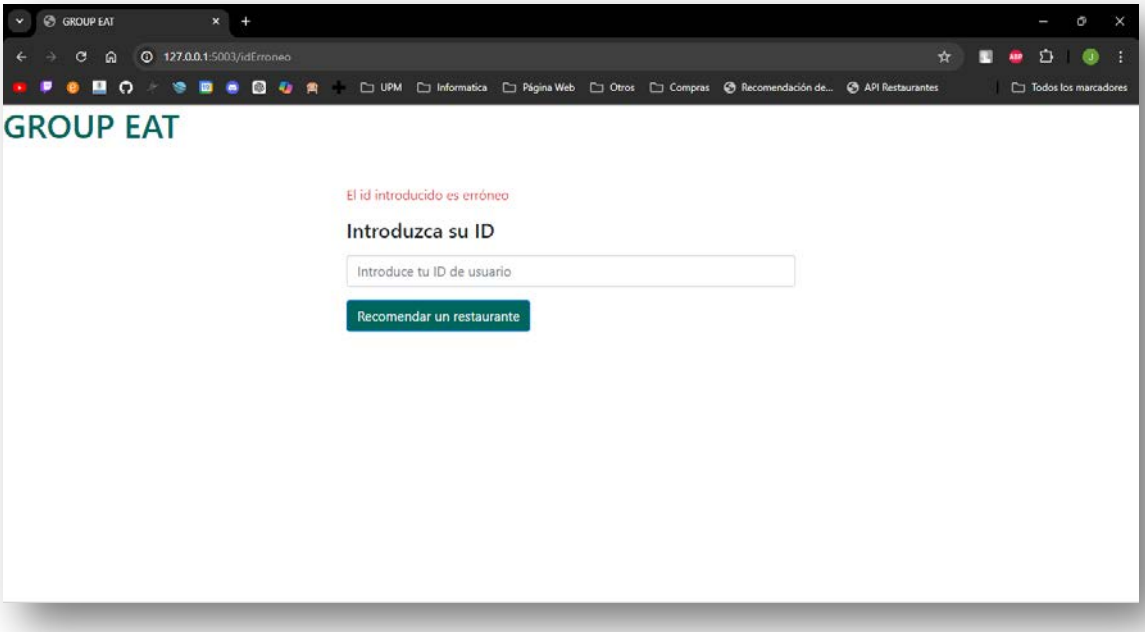
4.1.1. Sistema generado

Una vez finalizado el último sprint y por lo tanto la implementación del sistema, se ha obtenido un producto conforme a lo esperado. En las imágenes que se encuentran a continuación se visualiza un sistema sencillo y fácil de utilizar. En todas las ventanas se encuentra el mismo estilo y paleta de colores con una navegabilidad sencilla e intuitiva.

En primer lugar, se presenta la ventana de inicio. En la parte superior se muestra el logo del sistema. En el centro se muestra la puerta de entrada que da pie al sistema mediante el identificador de usuario. Se indica que es lo que debe introducir el usuario tanto en un texto encima del campo de escritura y dentro del campo de escritura con un texto predictivo. Una vez el usuario haya introducido su identificador personal, debe presionar el botón que encuentra debajo con el texto “Recomendar un restaurante” y este le redirigirá a la siguiente ventana. En caso de que se introduzca un identificador válido mostrará los amigos que dispone y las ciudades disponibles. En caso contrario se redirigirá de nuevo a esta pestaña.

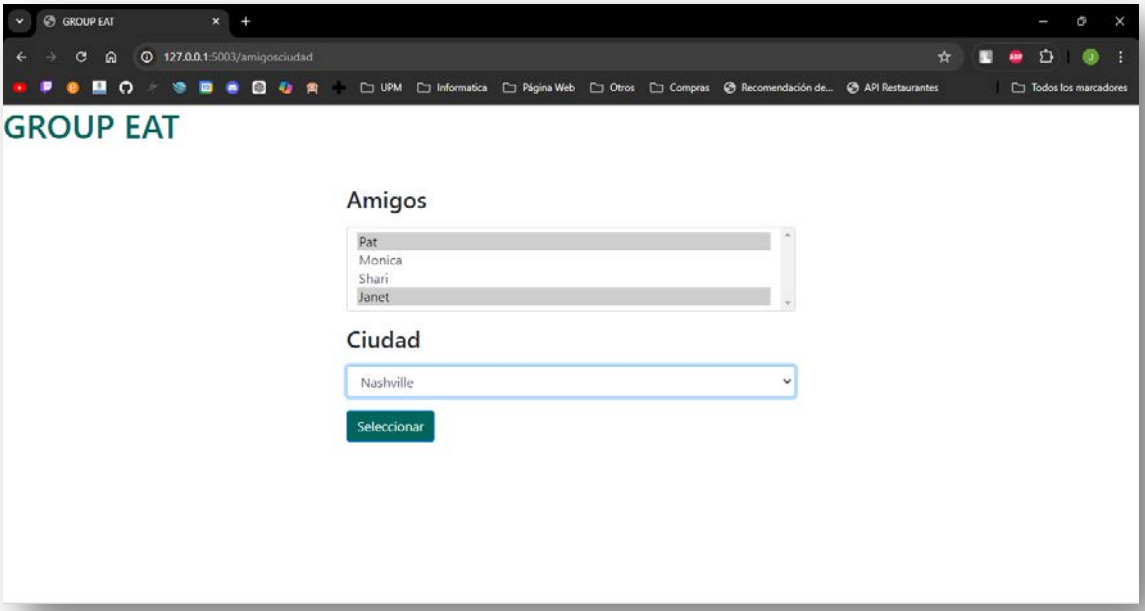


Antes de avanzar a la siguiente ventana, dado el caso de que se introduce un identificador incorrecto, se hará ver el error al usuario mostrando un texto en rojo como el siguiente. Este mensaje se encuentra en una página igual a la anterior, encima del texto indicativo de introducción del identificador de usuario.



Una vez introducido un identificador correcto avanzamos a la página en la que se va a formalizar el grupo y seleccionar la ciudad.

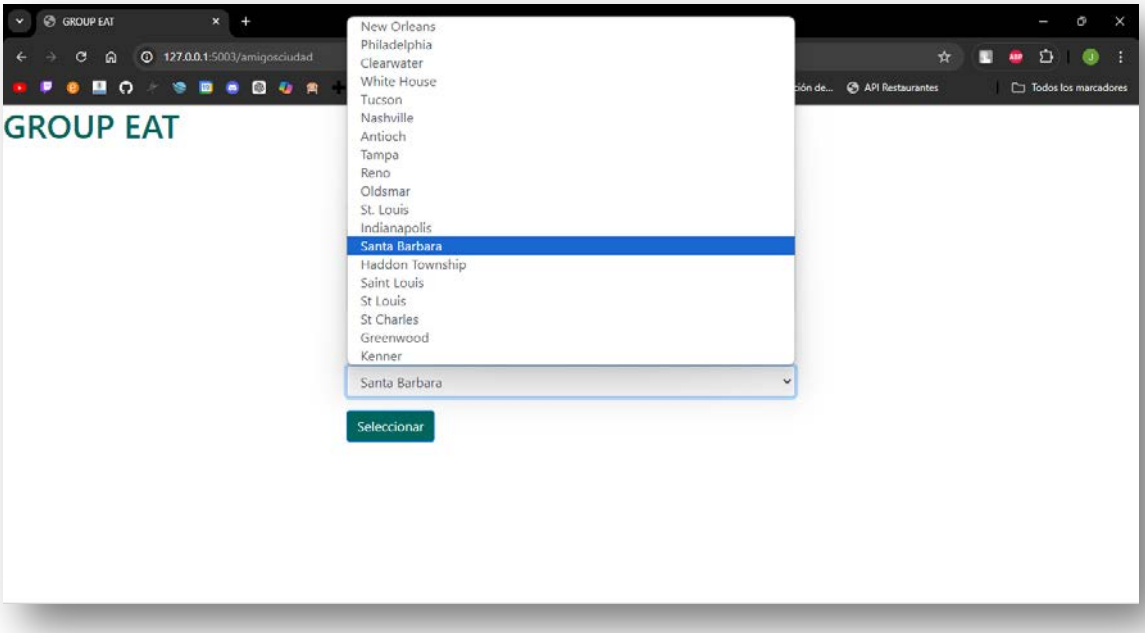
En primer lugar, se muestra un desplegable con todos los amigos. El desplegable es de múltiple selección, por lo que el usuario puede seleccionar cuantos amigos desee para formar el grupo de amigos.



Debajo del desplegable de amigos se presenta otro desplegable, pero esta vez muestra las ciudades disponibles para la recomendación. Este desplegable, a diferencia del

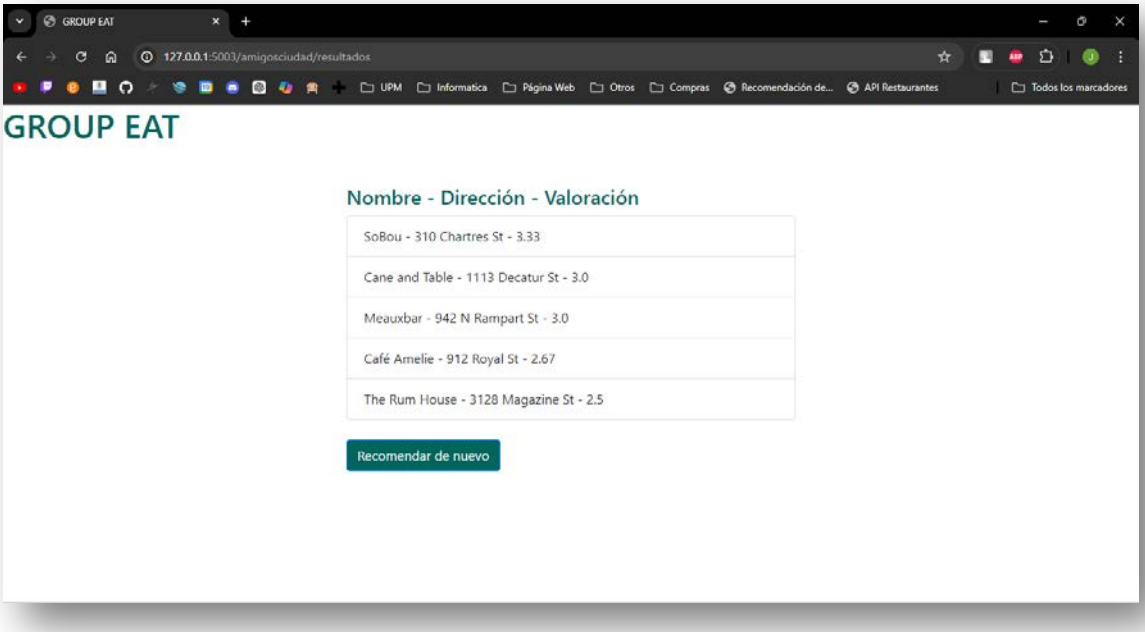
anterior solo nos da la posibilidad de seleccionar 1 opción. Una vez seleccionada, el desplegable se cierra y se guarda la elección.

Debajo de los desplegables se encuentra el botón que da paso a generar la recomendación. Este botón, que sigue con la estética del anterior, muestra el texto "Seleccionar" que realiza el guardado de los datos seleccionados y da paso a la siguiente ventana.



En la última ventana del sistema nos encontramos con la predicción generada. En ella se muestra un recuadro con los datos simplificados de cada restaurante. Se muestra de izquierda a derecha el nombre, la dirección donde se encuentra y la valoración generada, cómo se indica en la parte superior de la pantalla.

Al final, se encuentra otro botón que lo que hace es redireccionar al usuario a la página de selección de amigos y ciudad, para poder generar una recomendación para un nuevo grupo y/o ciudad distinta.



4.1.2. Pruebas del algoritmo

Tras la implementación del sistema, el siguiente paso es el de comprobar qué tan fiable es el sistema y sus recomendaciones. Para ello se ha generado un sistema de pruebas que genera casos de prueba de forma automática. Esto proporciona una métrica más o menos acertada que servirá para buscar mejoras de cara al futuro. Cuantas más pruebas se hagan, más real será el resultado obtenido.

El sistema de pruebas utiliza el algoritmo explicado en el apartado 3.4.2. Para aumentar la fiabilidad, se ha realizado el bucle de 50 restaurantes comprobados en un total de 30 iteraciones. De esta forma estamos comprobando un total de 1500 restaurantes en intervalos de 50.

Tras poner a prueba al algoritmo con esta prueba, se observa que el resultado final es de un 66% de acierto. Para comprobar los resultados obtenidos se adjunta un archivo Json en el Apéndice A.

4.2. Problemas y limitaciones

En la realización de este proyecto me he tenido que enfrentar a varios problemas y limitaciones que han hecho de este proyecto una experiencia más completa y me han aportado una motivación extra para realizarlo.

Para comenzar, el primer problema ha sido enfrentarse a tecnologías que no se dominaban. Esto es algo que se decidió ya que, tras investigar las recomendaciones de

Ángel, se llegó a la conclusión de que las tecnologías eran las idóneas para este proyecto. Vista esta situación, se ha tenido que resevar más tiempo de lo normal de lo que se haría normalmente para la investigación y el aprendizaje de cómo utilizarlas. Además de que requeriría más tiempo para implementar todo.

Nada más empezar la implementación, se presentó un problema en el *dataset* ya que este es bastante extenso. Al igual de que tiene puntos positivos como que nunca me van a faltar datos y que evito el problema del *Cold Start*; que sea tan extenso presenta el problema de que introducirlo en la base de datos es un quebradero de cabeza si no se ha hecho nunca. La opción de introducirlo por consola fue la acertada y conociendo las instrucciones necesarias, termina siendo una tarea más a realizar que no supone más problemas que otras. Por otro lado, el *dataset* no contiene los datos completos. Por ejemplo, hay reviews que no existe el restaurante o amistades de usuarios que tampoco forman parte del conjunto de datos.

Dado que el presupuesto es nulo, se encuentra la limitación de recursos disponibles. Esto afecta a lo siguiente:

- Imposibilidad de compra de licencias, teniendo que ceñirme a programas sin licencias. Esto supone que haya errores en las tecnologías utilizadas que podrían no aparecer si tuviesen un mayor soporte.
- A nivel hardware, dispongo de mi ordenador personal. Esto supone un problema a la hora de acceder a la interfaz web, ya que para poder utilizarla tiene hacerse desde el propio ordenador, o que al menos esté encendido y con el proceso arrancado, lo cual no puedo mantener.

Para terminar, se presenta la limitación de poder probar el sistema frente a varias recomendaciones simultáneas. Debido a esto, no se ha podido probar el RNF4.

A pesar de todo lo mencionado, el proyecto ha sido posible llevarlo a cabo solucionando cada una de las dificultades.

5. Conclusiones y trabajo futuro

5.1. Conclusiones

Una vez finalizado el proyecto, es momento de echar la vista atrás. En este proyecto, tras realizar una planificación de este, se ha creado un sistema de recomendación desde 0, formalizando una base de datos a partir de un *dataset* educativo. A partir de ella, se ha diseñado un algoritmo específico para el proyecto. Una vez implementado el *backend* completo del sistema, se ha diseñado una interfaz gráfica para poder utilizarlo. Finalmente, se ha terminado probando el sistema por completo para determinar su precisión y ver si finalmente se han cumplido todos los requisitos que se establecieron al inicio del proyecto.

En este punto haré una comparativa entre los requisitos que se pusieron al proyecto y los resultados finales explicados justo en el punto anterior.

Por un lado, todos los requisitos funcionales han sido completados. El único problema que se podía dar sería que alguna funcionalidad no se pudiese implementar por problema de tiempo, pero finalmente se han cumplido los plazos establecidos y no ha hecho falta dejar ninguno de ellos para el futuro.

En cuanto a los requisitos no funcionales haré un repaso más detallado:

- En cuanto a la accesibilidad y usabilidad, el sistema es lo suficientemente sencillo de entender y simple como para que no suponga ningún problema para ningún usuario hacer uso de este. En 3 pasos se obtiene una recomendación.
- El sistema es compatible con cualquier sistema de mostrado de datos ya que a partir del archivo *DataExchange.py* se puede hacer el cambio de datos sin ningún problema. Incluso Flask puede ayudar a ello generando el módulo que se necesite para comunicar con los métodos de ese fichero.
- Las recomendaciones se generan en torno a los 3-4 segundos mediante la interfaz gráfica, cumpliendo con el objetivo del RNF-3
- La base de datos queda intacta tras cada iteración, por lo que la fiabilidad es un objetivo cumplido.
- Tras las pruebas realizadas con el sistema de pruebas, podemos observar que el algoritmo de recomendación es preciso a un 66% con las características dadas.
- Cuando se intenta acceder a una URL de forma directa en la interfaz web, se maneja de tal forma que el usuario ve que está accediendo de la forma incorrecta y no se muestra ninguna recomendación ni información de ningún usuario.
- En cuanto al RNF-4 que está relacionado con la escalabilidad del sistema, ha sido imposible probarlo debido a la falta de recursos.

5.2. Trabajo futuro

Dadas las condiciones del proyecto hay ideas que no se han podido realizar y que a continuación mencionaré para poder realizarse en un futuro si se precisa.

1. Por un lado, encuentro el requisito no funcional RNF-4 que, como mencioné antes. Únicamente puedo acceder yo al sistema y no se puede probar que pasaría si accediesen varios usuarios de forma simultánea. A esto se le suma poder mantener de forma continua el servicio en funcionamiento y accesible desde cualquier punto.
2. Dadas las características del *dataset*, la identificación de los usuarios es muy precaria utilizando el identificador de usuario. A la hora de continuar con la usabilidad del sistema, se debería buscar un sistema de registro e inicio de sesión, de tal forma que cada usuario contase con un usuario único y una contraseña. Esto debería ser realizado siguiendo os protocolos de seguridad encriptando la contraseña. Adicionalmente, se debería añadir información a los usuarios como puede ser un correo electrónico.
3. Como tercer punto de trabajo futuro propongo aumentar la eficiencia del sistema o de las propias consultas, reducir las modificaciones de la base de datos o cualquier propuesta que haya a favor de cumplir y mejorar los requisitos establecidos al inicio de este documento.
4. Aumentar las funcionalidades de este sistema pudiendo filtrar los resultados, no solo por ciudad, si no por tipo de comida (italiana, mexicana, vegana…), tipo de restaurante, precio o cualquier filtro que pueda ser interesante y necesario.
5. Aumentar la seguridad de las conexiones y todo lo relacionado con la base de datos y el tratamiento de estos. Dado que se trata de un proyecto privado, con unos datos públicos y sin acceso al exterior, es un punto que no se ha tratado y queda fuera del alcance actualmente.

5.3. Impacto social

Finalizo el documento tal y como lo empiezo, haciendo referencia a las costumbres que tenemos arraigadas en nuestra sociedad. Muchas veces se utiliza como excusa quedar para tomar algo, comer, cenar etc. Y a su vez, muchas veces es un problema decidir a qué lugar ir, por lo que, con este proyecto se presenta una solución que puede aportar su grano de arena a esta sociedad.

Pienso que se trata de un gran proyecto que, con el tiempo y el apoyo requerido, puede convertirse en un sistema al que recurrir para solucionar un problema de una forma sencilla sin ser demasiado invasivos ya que permitiría a los usuarios seguir eligiendo.

5.4. Conclusiones personales

Para finalizar el proyecto, pienso que ha sido muy satisfactorio realizarlo tal y como lo he hecho. Me parece muy completo, se trata cualquier punto del ciclo de vida de un proyecto que pueda ser incluido en un proyecto software al uso. Al final, se trata de un trabajo final en el que se debe demostrar lo aprendido en toda la carrera. Este proyecto pienso que cumple con ello, tanto en las dotes de programación, como su particular guiño al mundo matemático en el algoritmo (aunque no sea muy complicado lo realizado), como la moraleja de no estancarnos en lo conocido y que debemos seguir avanzando siempre aprendiendo tecnologías y técnicas nuevas.

Ha sido un proyecto largo, pero en el cual no he dejado de aprender y recordar detalles que se acaban dejando en el olvido a lo largo de la carrera, ya que tras una asignatura viene otra y muchas veces lo nuevo acaba opacando lo anterior. Sobre todo, me siento muy satisfecho con el resultado que se ha obtenido.

Personalmente no se si en un futuro lo continuaré, más allá del segundo TFG que se me presenta a continuación y aprovecharé la oportunidad para continuarlo creando una nueva interfaz gráfica, este caso en una aplicación Android, pero me parece un proyecto con mucha proyección y que, sin ninguna duda, lo continuaría con el afán de completarlo.

6. Bibliografía

[1] Yelp. *Yelp.* 2022. https://www.yelp.com/dataset (último acceso: 10 de 12 de 2022).

[2] *ideasfrescas.com.mx.* 2024. https://ideasfrescas.com.mx/que-son-los-algoritmos-de-recomendacion/ (último acceso: 26 de 04 de 2024).

[3] Na, & Na., Sistemas de recomendación: Aprende Machine Learning, Aprende Machine Learning. (última consulta 26/3/2024) https://www.aprendemachinelearning.com/sistemas-de-recomendacion/

[4] Gramhagen. *Microsoft Learn.* s.f. rn.microsoft.com/es-es/azure/architecture/solution-ideas/articles/build-content-based-recommendation-system-using-recommender (último acceso: 28 de Abril de 2024).

[5] Martín, J. C. *Intef.* s.f. https://descargas.intef.es/recursos_educativos/ODES_SGOA/Bachillerato/TeI/7B6_SA_Recomendacion_a_tu_servicio/sistema_basado_en_filtrado_colaborativo.html (último acceso: 28 de Abril de 2024).

[6] Solusoft. Solusoft. s.f. https://www.solusoft.es/innovacion/sistemas-recomendacion-collaborative-filtering (último acceso: 28 de Abril de 2024).

[7] Perry, B. *FinePROXY.* 21 de Mayo de 2023. https://fineproxy.org/es/wiki/hybrid-recommender-systems/ (último acceso: 28 de Abril de 2024).

[8] Haro, Jose Felix de. *FutureSpace.* 17 de Septiembre de 2021. https://www.futurespace.es/sistemas-de-recomendacion-de-contenidos-adivina-que-piensan-tus-clientes/ (último acceso: 28 de Abril de 2024).

[9] Garzas, Javier. *javiergarzas.com.* 19 de Mayo de 2015. https://www.javiergarzas.com/2019/05/cual-es-el-estado-de-la-agilidad-a-nivel-mundial-13th-state-of-agile.html.

[10] *Proyectos Agiles.* 20 de Septiembre de 2021. https://proyectosagiles.org/que-es-scrum/ (último acceso: 4 de Mayo de 2024).

[11] *Kanban tool.* s.f. https://kanbantool.com/es/metodologia-kanban (último acceso: 4 de Mayo de 2024).

[12] *Miro.* s.f. https://miro.com/es/plantillas/matriz-moscow/ (último acceso: 27 de 06 de 2024).

[13] *GraphEverywhere.* s.f. https://www.grapheverywhere.com/algoritmo-de-similaridad-de-jaccard/ (último acceso: 27 de 06 de 2024).

[14] Alexander Felfernig, Ludovico Boratto, Martin Stettinger, Marko Tkalcic. *Group Recommender Systems.* Springer, 2018.

[15] Burrueco, Daniel. *Interactive Chaos.* s.f. https://interactivechaos.com/es/wiki/validacion-cruzada (último acceso: 12 de Junio de 2024).

[16] Oracle España. *Oracle.* s.f. https://www.oracle.com/es/autonomous-database/what-is-graph-database/ (último acceso: 12 de Junio de 2024).

Reza Zafarani, Mohammad Ali Abbasi, Huan Liu. *Social Media Mining.* Cambridge: Cambridge University Press, 2014.

APÉNDICE A: Resultados de prueba de precisión del algoritmo


```
{
    "Iteracion": "0",
    "Apariciones": "29/50",
    "Porcentaje de acierto": "58.0%"
}
{
    "Iteracion": "1",
    "Apariciones": "30/50",
    "Porcentaje de acierto": "60.0%"
}
{
    "Iteracion": "2",
    "Apariciones": "31/50",
    "Porcentaje de acierto": "62.0%"
}
{
    "Iteracion": "3",
    "Apariciones": "20/50",
    "Porcentaje de acierto": "40.0%"
}
{
    "Iteracion": "4",
    "Apariciones": "35/50",
    "Porcentaje de acierto": "70.0%"
}
{
    "Iteracion": "5",
    "Apariciones": "34/50",
    "Porcentaje de acierto": "68.0%"
}
```

```
{
    "Iteracion": "6",
    "Apariciones": "32/50",
    "Porcentaje de acierto": "64.0%"
}
{
    "Iteracion": "7",
    "Apariciones": "38/50",
    "Porcentaje de acierto": "76.0%"
}
{
    "Iteracion": "8",
    "Apariciones": "23/50",
    "Porcentaje de acierto": "46.0%"
}
{
    "Iteracion": "9",
    "Apariciones": "33/50",
    "Porcentaje de acierto": "66.0%"
}
{
    "Iteracion": "10",
    "Apariciones": "38/50",
    "Porcentaje de acierto": "76.0%"
}
{
    "Iteracion": "11",
    "Apariciones": "40/50",
    "Porcentaje de acierto": "80.0%"
}
{
    "Iteracion": "12",
    "Apariciones": "30/50",
    "Porcentaje de acierto": "60.0%"
}
```

```
{
    "Iteracion": "13",
    "Apariciones": "36/50",
    "Porcentaje de acierto": "72.0%"
}
{
    "Iteracion": "14",
    "Apariciones": "36/50",
    "Porcentaje de acierto": "72.0%"
}
{
    "Iteracion": "15",
    "Apariciones": "33/50",
    "Porcentaje de acierto": "66.0%"
}
{
    "Iteracion": "16",
    "Apariciones": "31/50",
    "Porcentaje de acierto": "62.0%"
}
{
    "Iteracion": "17",
    "Apariciones": "27/50",
    "Porcentaje de acierto": "54.0%"
}
{
    "Iteracion": "18",
    "Apariciones": "38/50",
    "Porcentaje de acierto": "76.0%"
}
{
    "Iteracion": "19",
    "Apariciones": "43/50",
    "Porcentaje de acierto": "86.0%"
}
```

```
{
    "Iteracion": "20",
    "Apariciones": "41/50",
    "Porcentaje de acierto": "82.0%"
}
{
    "Iteracion": "21",
    "Apariciones": "35/50",
    "Porcentaje de acierto": "70.0%"
}
{
    "Iteracion": "22",
    "Apariciones": "41/50",
    "Porcentaje de acierto": "82.0%"
}
{
    "Iteracion": "23",
    "Apariciones": "30/50",
    "Porcentaje de acierto": "60.0%"
}
{
    "Iteracion": "24",
    "Apariciones": "20/50",
    "Porcentaje de acierto": "40.0%"
}
{
    "Iteracion": "25",
    "Apariciones": "28/50",
    "Porcentaje de acierto": "56.0%"
}
{
    "Iteracion": "26",
    "Apariciones": "25/50",
    "Porcentaje de acierto": "50.0%"
}
```

```
{
    "Iteracion": "27",
    "Apariciones": "32/50",
    "Porcentaje de acierto": "64.0%"
}
{
    "Iteracion": "28",
    "Apariciones": "41/50",
    "Porcentaje de acierto": "82.0%"
}
{
    "Iteracion": "29",
    "Apariciones": "40/50",
    "Porcentaje de acierto": "80.0%"
}
{"Porcentaje total de acierto": "66.0%"}
```