Trabajo de Fin de Grado

Sistema inteligente de recomendación de restaurantes

Intelligent restaurant recommendation system

Marta Julia González Padrón

La Laguna, 7 de julio de 2022

D. **Christopher Expósito Izquierdo**, con N.I.F. 78.851.649-J profesor Ayudante Doctor adscrito al Departamento Ingeniería Informática y de Sistemas de la Universidad de La Laguna como tutor.

D. **Israel López Plata**, con N.I.F. 42.193.801-W profesor Ayudante Doctor adscrito al Departamento Ingeniería Informática y de Sistemas de la Universidad de La Laguna como cotutor.

C E R T I F I C A N

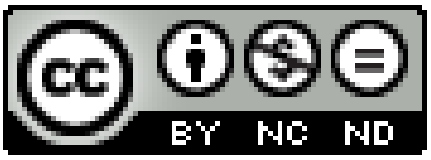
Que la presente memoria titulada:

”*Sistema inteligente de recomendación de restaurantes*”

ha sido realizada bajo su dirección por Doña **Marta Julia González Padrón**, con N.I.F. 79.098.982-G.

Y para que así conste, en cumplimiento de la legislación vigente y a los efectos oportunos firman la presente en La Laguna a 7 de julio de 2022

Licencia



Agradecimientos

A mis tutores, Christopher e Israel, por haberme guiado y por su paciencia a la hora de tutorizar este trabajo.

A mis padres, familiares, pareja y amistades, por apoyarme y confiar en mí.

A mis amigos Vanessa y Óscar, que comenzaron siendo compañeros de clase, pero que tras tantas horas de trabajo juntos se han convertido en pilares de mi trayectoria en la carrera.

A mis compañeros de trabajo, por hacer estos meses de duro trabajo más amenos e inspirarme a conseguir mis objetivos profesionales.

A las personas que he conocido a lo largo de la carrera, que me han aportado conocimiento y ratos felices, por haber formado parte de este capítulo tan importante que tan deseadamente llega a su fin.

Resumen

Los sistemas de recomendación ayudan a las personas a encontrar ítems de calidad que cumplan unos requisitos previos, que pueden estar ligados a los gustos del usuario o a sus preferencias en ese momento. Esto lo hacen intentando predecir, con la ayuda de diferentes técnicas, cuáles de los ítems disponibles son de mayor interés para el usuario.

El desarrollo de este Trabajo de Fin de Grado nace del interés por aportar una oferta de mayor calidad a la demanda que tienen a día de hoy los sistemas de recomendación en el ámbito gastronómico. Puesto que el demandante cambia de requisitos y gustos constantemente influenciados por su entorno, se quiere priorizar la creación de un sistema flexible capaz de adaptarse a los cambios del usuario. Para ello, se ha implementado un sistema que utiliza un filtrado basado en contenido [13] con componentes totalmente desacoplados para poder así ir adaptando el mismo a las demandas de los usuarios, como pueden ser nuevas características como entrada del algoritmo o cambios directos en el mismo.

Los resultados recopilados en la experimentación muestran como se ha cumplido el objetivo principal de crear una aplicación capaz de realizar una recomendación personalizada de restaurantes al usuario. Adicionalmente, se plantea la posibilidad de utilizar otros filtrados para así poder complementar y mejorar el presente.

Palabras clave: recomendación, restaurantes, filtrado basado en contenido, aplicación, sistema

Abstract

Recommender systems help people find quality items that meet certain prerequisites, which may be linked to the user's tastes or preferences at the time. They do this by trying to predict, with the help of different techniques, which of the available items are most interesting for the user.

The development of this final degree project arises from the interest in providing a higher quality offer to the demand that recommendation systems have nowadays in the gastronomic field. Since the demander's requirements and tastes are constantly changing and influenced by their environment, the aim is to prioritize the implementation of a flexible system capable of adapting to the user's changes. For this purpose, a system using content-based filtering [13] with fully decoupled components has been implemented in order to adapt the system to user demands, such as new features as the algorithm's input or direct changes to the algorithm.

The results collected in the experimentation show that the main goal of creating an application capable of making a personalized recommendation of restaurants to the user has been achieved. In addition, the possibility of using other filtering methods to complement and improve the existing one is also considered.

Keywords: *recommendation, restaurants, content-based filtering, application, system*

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

A día de hoy, gracias a Internet se tiene acceso a infinidad de información en cuestión de segundos [33]. Es por ello, que muchas veces el usuario confía en la recomendación de terceros a la hora de realizar una acción, como pueden ser las valoraciones de un producto publicadas en la red. No obstante, dichas recomendaciones no tienen por qué ser objetivas y es de ahí que nace la necesidad de la incorporación de sistemas de recomendación en Internet.

Hoy en día, los sistemas de información se han integrado en todas las áreas comerciales. Por ejemplo, en el sector turístico, y concretamente en el de la industria gastronómica, la información generada a partir de las recomendaciones y valoraciones es muy importante, ya que en principalmente es generada por los propios clientes. Por lo tanto, la correcta gestión de esta información resulta fundamental tanto para los usuarios finales, como para los establecimientos de restauración. En este último caso, las buenas críticas pueden suponer un aumento muy importante de sus ingresos, por lo que deben prestar atención a la calidad de su servicio.

Debido a que la cantidad de información disponible en las plataformas de reseñas es muy elevada, es decisivo disponer de sistemas que recopilen, evalúen y sinteticen la información para que sea mostrada de forma accesible al usuario. Los estándares de accesibilidad[1] incluyen adaptar lo que se muestra a los gustos del usuario y descartar elementos que son de poca utilidad para el usuario. Por otra parte, las personas tienen la necesidad de satisfacer sus gustos y no se conforman con fuentes extensas, lo que apoya lo establecido en los estándares de accesibilidad. Debido a la demanda que existe para las personas de consumir ítems, nace la oferta de los sistemas recomendadores, que caza perfectamente con esta problemática.

Los sistemas de recomendación son aplicaciones que permiten a los usuarios obtener información sobre productos de interés en función de su perfil y otros factores. Este tipo de software utiliza algoritmos de inteligencia artificial que ofrecen las mejores opciones disponibles y muestran cierto grado de idoneidad para cada una de ellas, en función de una variedad de características evaluables. Estos sistemas de recomendación se pueden aplicar a una variedad de temas, como recomendaciones de series/películas, regalos, libros y música, entre otros. Este Trabajo de Fin de Grado se centra en la creación de un sistema inteligente de recomendación de restaurantes para satisfacer la necesidad de las personas a la hora de planificar qué establecimiento gastronómico visitar.

[1] *Estándares de Accesibilidad*: `www.w3.org/WAI/standards-guidelines/wcag`

1.1. Tipos de sistemas de recomendación

Como ya fue mencionado con anterioridad, los sistemas de recomendación tienen como objetivo proporcionar información de un producto que sea de interés para el usuario según su perfil u otros factores. Se ha de destacar que dependiendo del problema a tratar, estos sistemas pueden variar y filtrar la información de diferente manera, pero en todos los casos se va a tener al usuario al que se le desea realizar la recomendación, un conjunto de ítems[2] y una forma de recoger las valoraciones que le hacen los usuarios a los ítems. Esta recolección de datos puede ser implícita, recogiendo las interacciones que hace el usuario con los mismos, o explícita, dando una valoración directa a los ítems.

Algunos de los tipos de sistemas de recomendación más conocidos son los siguientes:

- *Filtrado basado en popularidad (figura 1.1).* Esta técnica busca filtrar los ítems según la popularidad de los mismos respecto a una variable, que usualmente suele ser el número de valoraciones o el número de ventas de un artículo, y son expuestos a todos los usuarios de forma genérica. Tienen como ventaja que son sistemas bastante sencillos, pero no incluyen la personalización según el usuario.

Ítem	Número de valoraciones
A	100
B	20

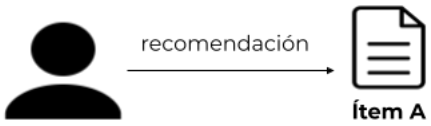


Figura 1.1: Filtrado basado en popularidad

- *Filtrado basado en contenido (figura 1.2).* Este tipo de filtrado se basa en la experiencia del usuario, ya que recomienda un ítem según las características de los ítems que les ha gustado al usuario previamente. Alguna de las ventajas que ofrece este tipo de recomendación, es que si se añaden nuevos ítems al sistema, el filtrado los sigue teniendo en cuenta, ya que estos tienen unas características fijas. Por otro lado, aparece el problema del nuevo usuario, ya que al ser nuevo no tiene valorados ningún ítem y el sistema es incapaz de saber sus gustos. Además, al usuario se le recomiendan siempre ítems similares y no se tienen en cuenta otros ítems que quizás puedan ser interesantes para él.

[2] Ítem: Unidad de un conjunto.

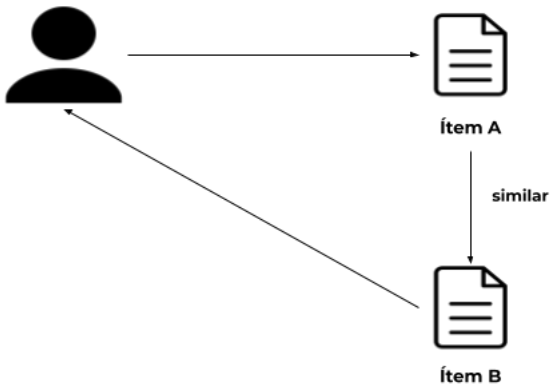


Figura 1.2: Filtrado basado en contenido

- *Filtrado colaborativo (figura 1.3).* Como bien indica su nombre, este filtrado se basa en la similitud con otros perfiles de usuario para obtener los ítems que puedan ser de interés para el usuario activo. Esto quiere decir que para recomendar un ítem se consultan otros usuarios que tengan gustos parecidos al usuario actual, y se le recomiendan los otros ítems que son de agrado para los otros usuarios pero desconocidos para el usuario actual. Este sistema aporta variedad en las recomendaciones, pero, sin embargo, los ítems nuevos no son incluidos en la recomendación porque no han sido consumidos por ningún usuario y se necesitan vectores de valoraciones muy grandes.

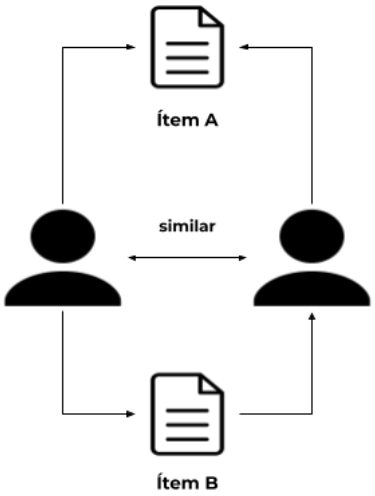


Figura 1.3: Filtrado colaborativo

- *Filtrado basado en conocimiento (figura 1.4).* Esta recomendación recoge las preferencias del usuario en ese momento y retorna los ítems que cumplen los requisitos. Este filtrado es muy útil cuando los datos del sistema donde se integran son propensos a cambiar, pero tienen como desventaja que el usuario debe interactuar más con la interfaz gráfica para que el sistema pueda recoger los datos de entrada.

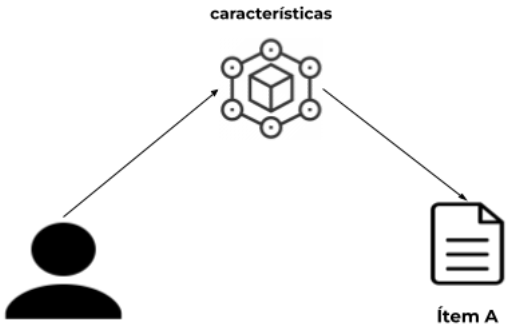


Figura 1.4: Filtrado basado en conocimiento

- *Sistemas híbridos*. Este tipo de sistemas utiliza diferentes tipos de filtrado en uno solo. Esta alternativa suele dar los mejores resultados, ya que combina diferentes tipos de filtrado para cubrir las carencias que tienen por separado y así poder realizar una recomendación más precisa.

1.1.1. Algoritmos recomendadores

Existen múltiples algoritmos para solucionar un problema que requiera una recomendación. Un enfoque son los filtrados colaborativos. Dentro de estos, existen dos diferentes estrategias, según [13]: *basado en vecinos* y *basado en modelos*.

Un algoritmo basado en memoria o vecinos construye una matriz de similitud entre usuarios según los ítems consumidos de cada uno de ellos. Una técnica que se suele usar bastante para ello es la técnica de los *K-vecinos* [28], que considera, como bien indica su nombre, los *K* vecinos con más gustos similares al usuario en cuestión. Para calcular dicha similitud, existen diferentes métricas [4]:

- *Similitud del coseno*. Calcula cuán similar son dos vectores matemáticamente en un espacio de producto interno[3]. Esto se calcula dividiendo el producto escalar de ambos vectores entre la magnitud de los mismos. El resultado de este debe estar entre 0 y 1. Cuanto más cercano a 1, más similares son los vectores.

$$similitud(x,y) = \frac{\sum_{i \in B_{x,y}} (r_{x,i} \cdot r_{y,i})}{\sqrt{\sum_{i \in B_{x,y}} (r_{x,i})^2} \cdot \sqrt{\sum_{i \in B_{x,y}} (r_{y,i})^2}}$$

$B_{x,y}$ es el conjunto de ítem calificados por los usuario *x* e *y*,
$r_{x,i}$ es la calificación del usuario *x* con respecto al ítem *i*.

- *Correlación de Pearson*. Mide la relación entre dos variables siempre que sean aleatorias cuantitativas.

[3] *Espacio de Producto Interno*: `www.hmong.es/wiki/Inner_product_space`

$$similitud(x,y) = \frac{\sum_{i \in B_{x,y}} (r_{x,i} - \overline{r}_x) \cdot (r_{y,i} - \overline{r}_y)}{\sqrt{\sum_{i \in B_{x,y}} (r_{x,i} - \overline{r}_x)^2 \cdot \sum_{i \in B_{x,y}} (r_{y,i} - \overline{r}_y)^2}}$$

$B_{x,y}$ es el conjunto de ítem calificados por los usuarios *x* e *y*,
r_x es la media de las calificaciones del usuario *x*, y
$r_{x,i}$ es la calificación del usuario *x* con respecto al ítem *i*.

- *Diferencia cuadrática media.* Calcula la diferencia Euclídea entre dos vectores de valoraciones.

$$similitud(x,y) = 1 - \frac{1}{\#B_{x,y}} \sum_{i \in I_u} \left(\frac{r_{x,i} - r_{y,i}}{max - min} \right)^2 \in [0,1]$$

$\#B_{x,y}$ es el cardinal del conjunto de ítems
calificados tanto por el usuario *x* como el usuario *y*.

Dichas métricas reciben como entrada una matriz con los votos de los usuarios a diferentes ítems y calculan cuánto se asemejan los unos con los otros. Tras obtener la similitud, se obtienen los *K* vecinos más similares y se predice que valoración le daría el usuario en cuestión a los ítems valorados por los vecinos. Esta predicción se puede hacer de diferentes maneras como pueden ser las medias aritmética y ponderada [25]:

- *Media aritmética.*

$$p_{u,i} = \frac{1}{\#G_{u,i}} \cdot \sum_{n \in G_{u,i}} r_{n,i}$$

  $p_{u,i}$ es la predicción calculada teniendo en cuenta las puntuaciones de los usuarios similares al actual *u*, en el ítem *i*.

  $G_{u,i}$ es el conjunto de valoraciones hechas por los usuarios vecinos del usuario activo, sobre el ítem del que se desea crear la predicción.

- *Media ponderada.* Es similar a la media aritmética, pero en este caso, cada uno de los valores se multiplica por un peso y la suma de todos ellos se divide entre la suma de todos los pesos. Esta métrica es de gran utilidad cuando los datos dentro del conjunto tienen importancias relativas al resto de datos.

$$p_{u,i} = \frac{\sum_{n \in G_{u,i}} sim(u,n) \cdot r_{n,i}}{\sum_{n \in G_{u,i}} sim(u,n)}$$

Con toda esta información, solo quedaría escoger cuantos ítems se le van a recomendar al usuario, basándose en la predicción anteriormente calculada.

Cabe destacar sobre los filtrados colaborativos basados en modelos, que a partir de la matriz de valoraciones, crea modelos de aprendizaje automático usados para realizar una

recomendación. Dentro de estos se pueden utilizar redes neuronales [7] y *Single Value Decomposition* (SVD) [34], entre otros.

Por otro lado, otro tipo de filtrado que se usa en los sistemas de recomendación es el basado en contenido. Este se utiliza mayoritariamente cuando se desea obtener resultados similares a los consumidos anteriormente por el usuario. Es por ello que este tipo de sistema ha sido el implementado en este Trabajo de Fin de Grado, cuya explicación en detalle se puede consultar en el apartado 3.2.1.

El proceso de recomendación en un filtrado basado en contenido se puede dividir en tres pasos:

1. *Analizar y obtener el contenido.* Primeramente, se recopila la información expuesta, en documentos, páginas o características de un ítem, para poder procesarla y estructurarla como sea necesario para poder ser usada.

2. *Aprender del perfil del usuario activo.* En este paso se recopilan las preferencias del usuario de su perfil, para poder así saber cuáles son las características que le gustan al usuario activo.

3. *Filtrar el contenido.* Finalmente, se filtra el contenido del primer paso calculando la similitud que tienen dichos elementos con el perfil del usuario basándose en la información extraída en el segundo paso, lo que resulta en una lista de ítems posiblemente relevantes para el usuario activo. Dicha similitud puede ser calculada con las métricas expuestas al principio del apartado 1.1.1.

Según [13], el algoritmo que parece predominar más a día de hoy es el *SVD*. Esto se debe a que tiene unos resultados muy buenos y basta con almacenar los vectores de valoraciones, por lo que elimina bastante ruido del algoritmo con respecto a otros. No obstante, otras investigaciones, como [5], indican que el comportamiento de los algoritmos depende mucho del contexto donde se apliquen. Es por ello que recomiendan estudiar los diferentes algoritmos que existen en el ámbito en cuestión y se analicen los resultados.

1.2. Objetivos

El objetivo principal de este Trabajo de Fin de Grado es la creación de una aplicación que, con la ayuda de algoritmos de Inteligencia Artificial, sea capaz de realizar una recomendación personalizada de restaurantes al usuario.

Los objetivos específicos son los siguientes:

- Se podrán gestionar perfiles de usuario.
- Se podrán consultar los restaurantes populares cerca del usuario.
- Se recomendarán restaurantes según una zona especificada.
- Se crearán itinerarios gastronómicos dadas las diferentes zonas donde se encontrará el usuario.
- Un usuario podrá escoger sus preferencias a la hora de planificar un itinerario gastronómico.

- Se mostrarán los resultados de forma geolocalizada.
- Un usuario podrá introducir reseñas con el fin de mejorar futuras recomendaciones.

Capítulo 2

Motivación y estado del arte

2.1. Estado del arte

Dado el fuerte impacto que tienen los sistemas de recomendación en el mercado actual, hoy en día existen diferentes alternativas comerciales que ayudan a la recomendación de restaurantes, entre los que se encuentran TripAdvisor[1], Yelp[2], Google Maps[3] o Foursquare[4]. No obstante, aunque estos sistemas son bastante utilizados, simplemente filtran los resultados según los parámetros que le indica el usuario en ese preciso momento, por lo que carecen de una personalización que puede ser interesante para el usuario. Tampoco buscan la automatización de la recomendación, mostrando todas las opciones disponibles y delegando enteramente en el usuario la decisión sobre qué restaurante elegir en cada momento. Sin embargo, este tipo de sistemas proporcionan APIs *(Application Programming Interfaces)* donde se recoge la información de cada uno de los restaurantes, las cuales pueden utilizarse como base para la realización de recomendaciones en el prototipo a desarrollar.

A diferencia de muchos otros campos, el origen de los sistemas de recomendación surge pocos años atrás, hace tres décadas, cuando en el Centro de Investigación de Palo Alto *Xerox*, quisieron entrar en detalle con respecto a la cantidad de correos electrónicos que eran recibidos. Ese interés causó la implementación de *Tapestry* [32], un sistema que filtra los documentos juntando las interacciones pasadas de diferentes perfiles, mejorando así los resultados. Este sistema recolecta la interacción que tenía cada usuario con los correos electrónicos para poder determinar cuáles eran de interés. Cabe destacar que *Tapestry* fue diseñado de tal manera que podía usarse para filtrar cualquier conjunto de documentos electrónicos.

A día de hoy, hay escasez de sistemas de recomendación únicamente de restaurantes que personalicen el resultado de la recomendación según diferentes datos de entrada y construyan a partir de ahí itinerarios para restaurantes. No obstante, si existen algunos sistemas relacionados con la gastronomía. Algunos de ellos se describen a continuación, tras haber hecho una revisión detallada del estado actual de los sistemas recomendadores de restaurantes:

REJA [21] es un sistema recomendador híbrido implementado en Jaén, España, que utiliza una recomendación con filtrado colaborativo y una basada en conocimiento para

[1]TripAdvisor: `https://www.tripadvisor.es/`
[2]Yelp: `https://www.yelp.es/`
[3]Google Maps: `https://www.google.es/maps/`
[4]Foursquare: `https://foursquare.com/`

poder proporcionar una recomendación de restaurantes al usuario.

Un ejemplo de sistema inteligente de recomendación de restaurantes fue uno implementado por un antiguo alumno de la Universidad de Barcelona para su Trabajo de Fin de Máster [15]. Se trata de integrar un sistema de recomendación a una aplicación ya existente llamada *Velada*[5]. El trabajo consiste en estudiar los datos de los restaurantes y los usuarios de la aplicación para poder así aplicar el algoritmo que mejor se comportase con esos datos. Se experimentó con un algoritmo colaborativo y con otro basado en contenido, con diferentes configuraciones iniciales, como pueden ser diferentes métricas base para calcular las recomendaciones.

Otra posibilidad para crear recomendaciones de restaurantes es la propuesta en [19], un sistema que genera recomendaciones usando el algoritmo colaborativo de los K-vecinos. Este se basa en las búsquedas pasadas del usuario para estudiar sus intereses con respecto a los restaurantes en el pasado.

También existen sistemas recomendadores de restaurantes relacionados con el turismo. El recomendador creado por [10] aporta unos resultados pensados para turistas, ya que este tipo de usuario busca mayoritariamente lugares donde poder vivir una experiencia culinaria típica de la región visitada. La recomendación se hace basándose en las preferencias del usuario.

Una perspectiva interesante para los sistemas recomendadores, es el sistema recomendador propuesto por [6]. La innovación en este proyecto es que utiliza información visual como pueden ser fotografías tomadas por los usuarios o las publicadas en redes sociales y procesándola con ayuda de una red neuronal convolucional. La idea nace de que los restaurantes similares suelen tener una apariencia parecida también. Para llevar a cabo este sistema recomendador se ha usado un filtrado colaborativo y uno basado en contenido.

Una alternativa relacionada con los restaurantes es la presentada por [22], una plataforma que se centra en recomendar sitios para comer basados en los platos y no en el restaurante como tal. La iniciativa nace de promover el turismo gastronómico y utiliza una recomendación basada en ítems y dependiente del contexto, que necesita entradas contextuales para poder mejorar la precisión de la recomendación a realizar. Este proyecto depende estrechamente de las reseñas y valoraciones de los usuarios. Es por ello que utiliza herramientas de gamificación, incorporando minijuegos, clasificaciones para el usuario o insignias, entre otros, para que el usuario se vea motivado a seguir aportando información a la aplicación.

Por otro lado, existe también un proyecto [16] que presenta un sistema de recomendación de dietas saludables que utilizan la información de los usuarios como las preferencias, datos demográficos, hábitos de ejercicio y situación de salud, para clasificarlos en categorías. Dicha categoría es utilizada por el sistema para suministrar al usuario paquetes de comida cada semana. No obstante, aunque el resultado final es una recomendación de dieta, este es realmente un problema de optimización que trata de maximizar el número de nutrientes que consume el usuario para cada una de sus comidas al día, teniendo en cuenta algunas restricciones como es el consumo de calorías diarias.

Aunque no haya gran cantidad de artículos científicos ni proyectos sobre recomendadores de restaurantes aplicados a un plan de comidas, existen miles de fuentes de información de restaurantes en la red. Entre las disponibles, una de las más conocidas es *TripAdvisor*[6], una aplicación web donde se recoge y se muestran valoraciones de negocios

[5] *Velada*: `www.appvelada.com`

[6] TripAdvisor: `www.tripadvisor.es`

relacionados con el sector turístico. Aunque es cierto que hay un apartado de restaurantes *(figura 2.1)*, este sistema carece de personalización y simplemente se limita a filtrar los restaurantes según las opciones seleccionadas por el usuario.

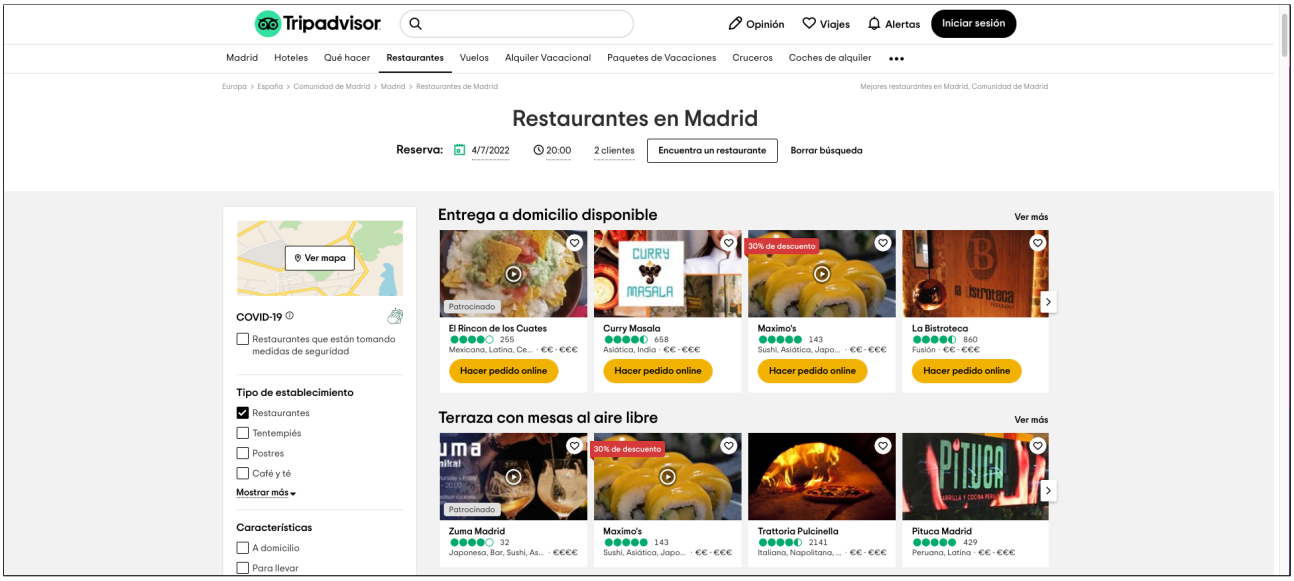


Figura 2.1: Pantalla de filtrado de restaurantes de TripAdvisor

Adicionalmente, una sub-compañía de *TripAdvisor* es *The Fork*[7] que realmente es un portal de reservas para restaurantes, aunque es una buena alternativa si se desea visualizar los restaurantes de una zona con sus respectivas valoraciones y otra información.

Otra de las alternativas más comunes a la hora de querer buscar restaurantes en una zona en específico es usando *Google Maps*[8]. Aunque realmente es un servidor de aplicaciones en la web, con ayuda del *Perfil de Empresa de Google*[9] esta plataforma muestra información general sobre los negocios, entre ellos restaurantes, y sus respectivas reseñas realizadas por diferentes usuarios.

Por último, existe un grupo de plataformas que están dentro de las aplicaciones de búsqueda comercial de restaurantes, que son las aplicaciones de servicio a domicilio y comida para llevar. Estas aplicaciones contienen información de restaurantes como el menú, localización, horario, precios e incluso reseñas. Algunos ejemplos de estas plataformas son: *Uber Eats*[10], *Glovo*[11] y *Just Eat*[12].

2.1.1. Otros sistemas recomendadores

Sistemas recomendadores en comercios electrónicos

En los años noventa, se comenzó a plantear la importancia de realizar un cambio en la producción de artículos para poder así satisfacer la amplia demanda de los consumidores ofertando variedad de productos [26]. Con ayuda del comercio electrónico, las empresas han podido solucionar esa problemática. No obstante, esto también ha causado que el

[7] The Fork: `https://www.thefork.es/`
[8] Google Maps: `www.google.es/maps`
[9] *Perfil de Empresa de Google*: `www.google.com/intl/es_es/business`
[10] *Uber Eats*: `www.ubereats.cm`
[11] *Glovo*: `www.glovoapp.com`
[12] *Just Eat*: `www.just-eat.es`

usuario deba manejar una mayor cantidad de información antes de poder adquirir el producto que cubre sus necesidades. Adicionalmente, los sistemas recomendadores han aportado diferentes beneficios al comercio electrónico que hacen que su incorporación en ellos sea esencial. Uno de ellos, es incentivar a los visitantes de las páginas a comprar productos. Según [2], solo el 3 % de los visitantes de los comercios electrónicos compran productos, por lo que realizar una buena recomendación para estos usuarios puede aumentar el número de ventas. Otro beneficio de los sistemas de recomendación es incentivar a los clientes, con ayuda de sistemas recomendadores, a comprar otros productos que puedan ser de su interés.

Algunas empresas que son ejemplos de comercio electrónico pueden ser *Amazon*[13] y *eBay*[14]. Ambas plataformas cuentan con perfiles de usuarios para rastrear sus gustos, valoraciones y diferentes filtros para encontrar los productos que se ajusten a las necesidades del usuario.

Sistemas recomendadores de películas

Las películas y series han sido muy relevantes en el desarrollo de los sistemas recomendadores. Por un lado, en los años 90, surgió el proyecto *GroupLens*[15] por el Instituto Tecnológico de Massachusetts y la Universidad de Minnesota, que también querían solventar el problema de la sobrecarga de información. En este caso llegaron a la conclusión de que si había usuarios que habían valorado positivamente algunos artículos, lo más probable sería que sus gustos coincidiesen en otros productos también. Desde entonces, GroupLens ha creado diferentes proyectos pioneros en la industria. Uno de los productos de *GroupLens* es *MovieLens*[16], un recomendador personalizado. Inicialmente, sus recomendaciones eran basadas en usuarios, pero a día de hoy lo son basadas en ítems.

Otro de los acontecimientos que han jugado un gran papel en la historia de los sistemas recomendadores sucedió en 2009. La compañía de transmisión de vídeo en línea *Netflix*, preparó una competición de algoritmos colaborativos, llamada *Netflix Prize* [1], con el fin de mejorar su algoritmo, llamado *Cinematch*. El ganador de esta competición fue el equipo *BellKor's Pragmatic Chaos*, que fue capaz de mejorar la predicción un 10.6 %. Desde entonces, la compañía ha seguido mejorando el algoritmo, incluyendo una clasificación personalizada y adaptación de la experiencia de usuario.

Sistemas recomendadores turísticos

Un tipo de sistema de recomendación de gran utilidad es el sistema de planificación turística. Este tipo de sistema puede generar recomendaciones a partir de la información proporcionada por varios subsistemas, como recomendaciones de vuelos, alojamiento, turismo y restaurantes. Uno de los sistemas de recomendación más relacionados con este Trabajo de Fin de Grado es el problema de diseño de planificación turística, o también conocido como *Tourist Trip Design Problem* (*TTDP*) [9]. Aunque originalmente *TTDP* es un problema de optimización, está estrechamente relacionado con los sistemas de recomendación. Este problema trata la planificación de un itinerario turístico para visitar diferentes puntos de interés, teniendo en cuenta la situación y preferencia del usuario. El *TTDP* ejerce un papel imprescindible en el desarrollo de la experiencia turística de los

[13] *Amazon*: `www.amazon.com`
[14] *eBay*:`www.ebay.com`
[15] GroupLens: `https://grouplens.org/`
[16] *MovieLens:* `https://movielens.org/`

turistas y por ello el beneficio económico para dichas atracciones turísticas. A raíz de esto, se han realizado diversas investigaciones a lo largo de los años para poder solucionar el problema.

El *TTDP* se define [30] por un conjunto de puntos de interés, unos tiempos en recorrer las distancias de un punto a otro teniendo en cuenta los diferentes medios de transportes, una puntuación para dicho punto de interés, el tiempo de visita para el mismo, el tiempo que desea el usuario realizar turismo al día y cuánto va a durar la estancia del turista. Con toda esa información, el problema busca una solución que haga eficaz el itinerario turístico del usuario.

Otro sistema recomendador que entra dentro de la categoría de recomendadores turísticos, son los sistemas recomendadores de hospedaje. Existen diferentes artículos al respecto, como [29], que estudian las preferencias de los usuarios. También se han implementado sistemas híbridos que mezclan recomendaciones basadas en contenido con recomendaciones colaborativas [35] [36]. Estos sistemas son de gran utilidad porque cada vez que una persona desea planificar un viaje, un elemento esencial es escoger en que lugar se van a hospedar, por lo que el uso de estos sistemas simplificaría y mejoraría este proceso a los usuarios.

2.2. Motivación

Tras el impacto que ha tenido la pandemia por COVID-19, muchos sectores como el turístico y el hostelero se han visto gravemente perjudicados. Además, se han visto en una tesitura donde la digitalización de su negocio ha sido primordial. Por otro lado, la cantidad de información que reside sobre restaurantes en internet es masiva. Es por ello que me ha llamado la atención este Trabajo de Fin de Grado, ya que une el campo turístico y el hostelero, generando una recomendación de sitios a partir de los gustos del usuario, las reseñas, el presupuesto y la zona donde vaya a estar el usuario, destacando así con respecto a otros sistemas ya existentes. Además, como ya se ha visto al principio de este capítulo 2, no existen sistemas conocidos que cumplan el objetivo de este Trabajo de Fin de Grado, por lo que me satisface crear y aportar un proyecto novedoso al mercado.

Otro aspecto que me ha incentivado a realizar este trabajo, es el aprender tecnologías nuevas y conectarlas entre sí, potenciando así lo aprendido a lo largo del grado y mejorando mi formación académica.

Capítulo 3

Sistema de recomendación de restaurantes

3.1. Descripción general

El aplicativo implementado en este Trabajo de Fin de Grado es un sistema inteligente de recomendación de restaurantes en forma de aplicación *Fullstack*, que le proporciona al usuario una serie de establecimientos para ir a comer según sus gustos y preferencias, como pueden ser el presupuesto que poseen o el tipo de comidas. En la siguiente imagen *(figura 3.1)* se pueden observar los diferentes casos de uso del aplicativo.

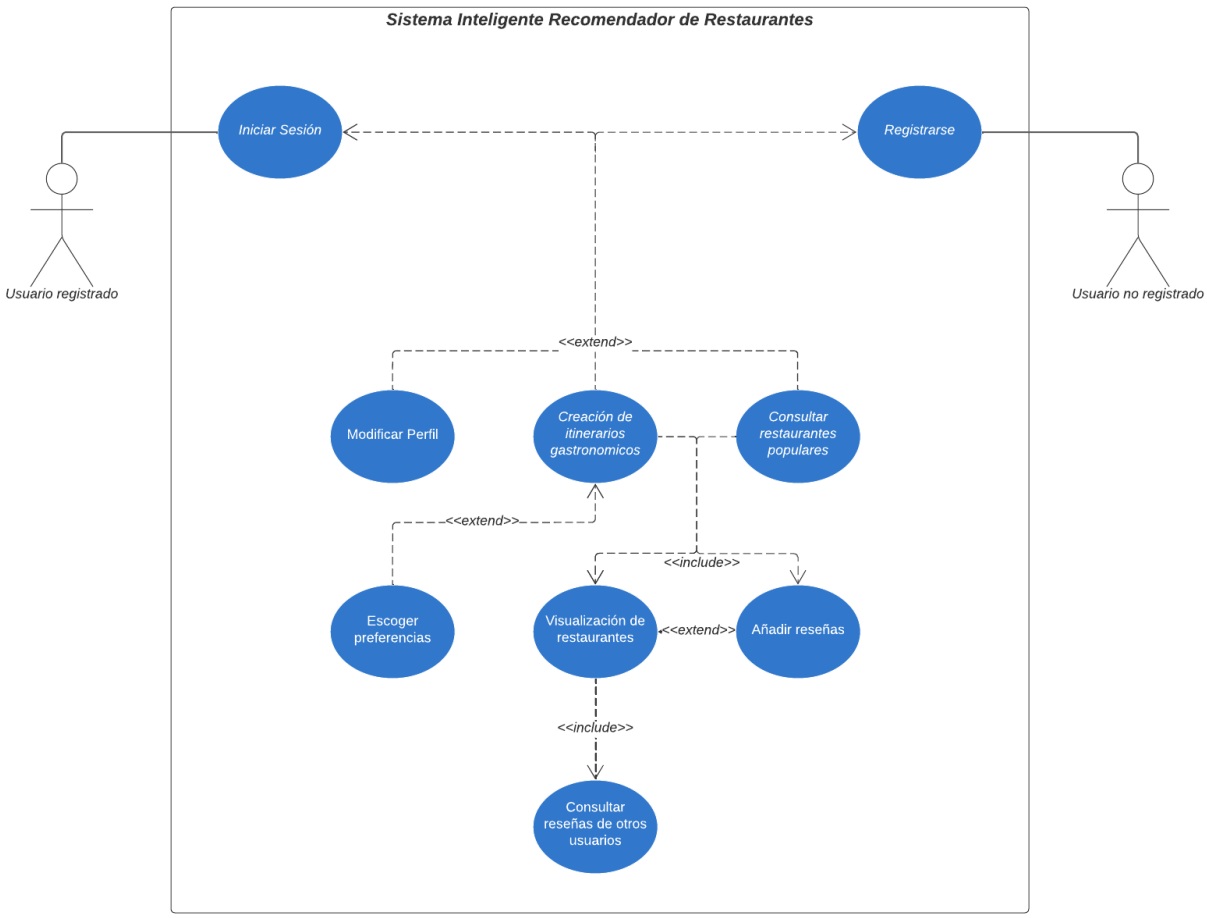


Figura 3.1: Diagrama de casos de uso

Para implementar esto se necesita que los usuarios se registren para poder empezar a almacenar sus preferencias y realizar una recomendación basada en su perfil. Por otro lado, la información de los restaurantes es obtenida de una API externa y proporciona la siguiente información:

- Identificador del restaurante
- Nombre
- Valoración
- Categorías
- Imágenes
- Conjunto de reseñas

Con esta información se crea un algoritmo de recomendación con un filtrado basado en contenido que utiliza la similitud del coseno para facilitarle al usuario diferentes locales de restauración de su interés. Dicho algoritmo se ha implementado completamente desacoplado del resto de la aplicación y recibe como entrada:

- Los ítems que han sido valorados positivamente por el usuario activo.
- Las preferencias del usuario activo.
- Todos los ítems almacenados.

La salida del algoritmo consiste en un conjunto de pares donde el primer elemento es un número decimal que representa la similitud calculada y el segundo elemento es el ítem al que le corresponde esa similitud.

Para el caso de este Trabajo de Fin de Grado, se han filtrado los posibles ítems que cumplen las restricciones del usuario activo, teniendo en cuenta la zona donde el usuario se va a encontrar, si la comida que va a realizar es el desayuno, almuerzo o cena, el tipo de restaurante de preferencia para el usuario y el presupuesto que dispone. Tras ejecutar el algoritmo con estos parámetros, los resultados se envía de nuevo a la interfaz de usuario para ser visualizados.

3.2. Tecnología y arquitectura del software

Este proyecto *Fullstack* tiene diferentes componentes, como son el *Frontend*, el *Backend* y la base de datos. Cada uno de ellos tiene cierto nivel de complejidad, es por ello que es muy importante una buena elección de las tecnologías a emplear. Como se puede ver en la *figura 3.2*, se ha utilizado el patrón de arquitectura del software *Modelo-Vista-Controlador* (MVC) [14], que separa el aplicativo en 3 componentes: las estructuras de datos (modelos), la interfaz de usuario (vista) y el objeto que controla la comunicación entre el modelo y la vista (controlador). Esto permite separar de forma clara la lógica de la aplicación, facilitando la escalabilidad y el mantenimiento de la misma. Además, el *Frontend* de la aplicación hace peticiones a una API externa para poder obtener los datos de los restaurantes. Adicionalmente, se ha implementado el recomendador de forma aislada y es consumido como servicio desde el *Backend*.

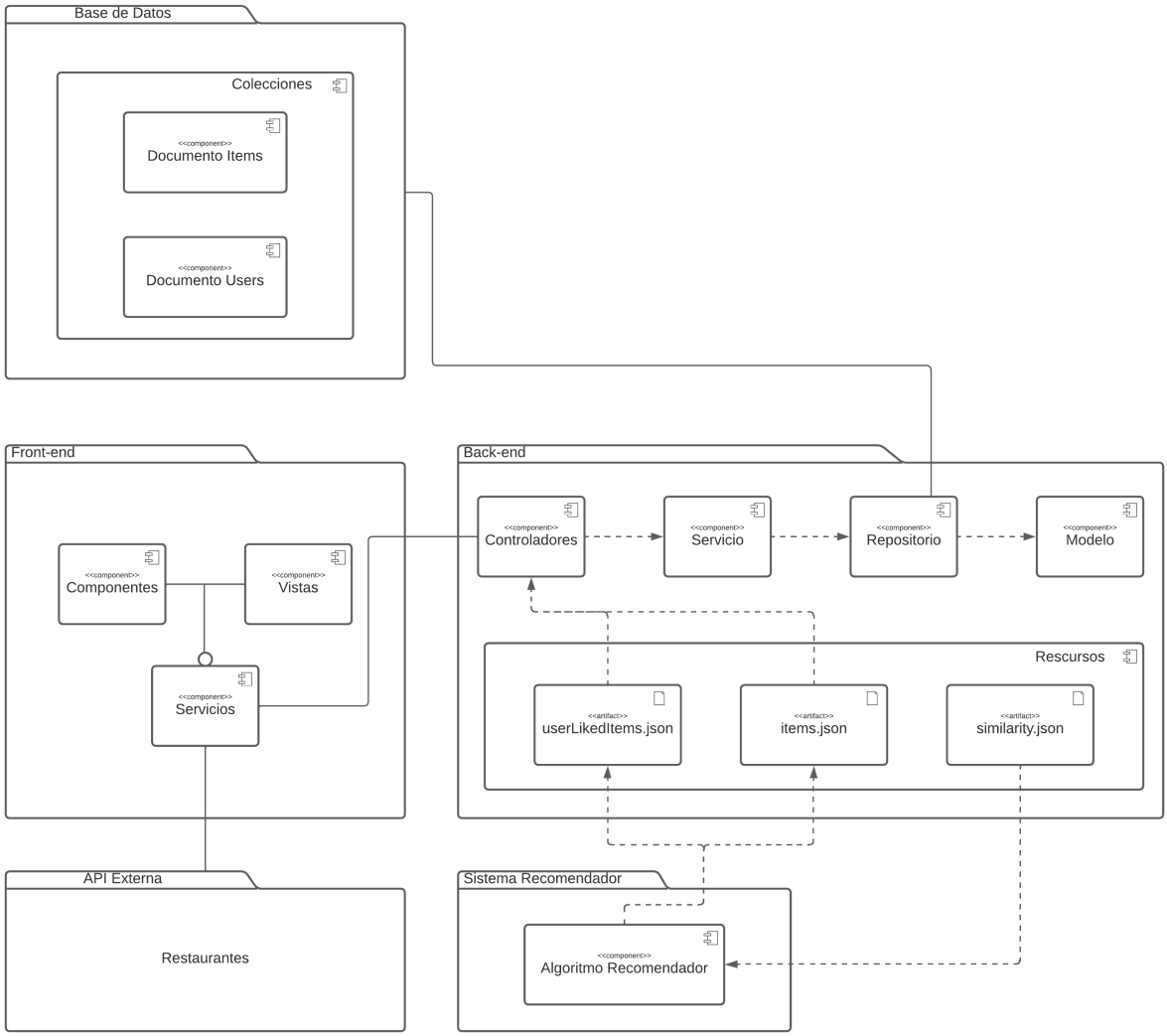


Figura 3.2: Diagrama de componentes

3.2.1. Algoritmo Recomendador

El algoritmo de recomendación de ítems a usuarios ha sido implementado como un proyecto aparte con una estructura de datos de entrada y salida predefinida. Al desacoplar el algoritmo del aplicativo, tanto el proyecto como el aplicativo se vuelve más flexible, escalable y mantenible, ya que se puede cambiar el algoritmo rápidamente o este puede ser utilizado en otros proyectos sin influir en el resto de la aplicación actual. El lenguaje de programación utilizado ha sido *Java* [8], un lenguaje de programación orientado a objetos, muy utilizado y por ende con una gran documentación. Adicionalmente, se ha usado *Maven*[1], una herramienta para la gestión de la estructura de proyectos implementados en *Java*. *Maven* necesita un fichero POM *(Project Object Model)* donde se especifican las dependencias y configuraciones del proyecto en formato XML *(Extensible Markup Language)*. El algoritmo recibe 3 parámetros:

- *items.json*: Un fichero en formato JSON que contiene el conjunto de ítems candidatos a ser recomendados.

[1] *Maven*: `https://maven.apache.org/`

- *userLikedItems.json*: Un fichero en formato JSON que contiene los ítems que han sido valorados positivamente por el usuario activo.
- *similarity.json*: Un fichero en formato JSON donde almacenar el resultado del algoritmo.

Cada ítem debe tener la siguiente estructura:

- *"id":* El identificador del ítem actual.
- *"name":* El nombre del ítem actual.
- *"description":* Una descripción del ítem actual.
- *rating":* Un número del 0 al 5 que represente la valoración del ítem actual. 0 es la puntuación más baja y 5 la más alta.
- *"latitude":* La latitud de la coordenada donde se encuentra el ítem actual.
- *"longitude":* La longitud de la coordenada donde se encuentra el ítem actual.
- *çategories":* Un conjunto de categorías que indiquen el tipo del restaurante actual. Por ejemplo, si se habla de restaurantes, una posible característica podría ser *"sushi"* o *"mexican"*.
- *"price":* Un número entero del 1 al 4 que represente el nivel adquisitivo del ítem actual. 1 representa que el ítem es muy barato y 4 indica que el ítem es muy caro.
- *"images":* Un conjunto de enlaces a imágenes del ítem actual.
- *reviews":* Un conjunto de reseñas del ítem actual.

Por otro lado, cada reseña *(review)* tiene esta estructura:

- *"id":* El identificador de la reseña actual.
- *"itemId":* El identificador del ítem que se está valorando en la reseña actual.
- *üserId":* El identificador del usuario que escribe la reseña actual.
- *rating":* Un número del 0 al 5 que represente la valoración que el usuario le da al ítem. 0 es la puntuación más baja y 5 la más alta.
- *"text":* Una valoración escrita por el usuario.
- *"timestamp":* Una marca temporal que indica cuando se realizó la reseña. El formato aceptado es *YYYY-MM-DD HH:mm:ss*.

Teniendo esta información, el algoritmo utiliza la similitud del coseno *(1.1.1)* para calcular que ítems podrían ser de interés para el usuario. Para conseguir el resultado, se siguen internamente estos pasos:

1. **Eliminar las palabras vacías de las reseñas de los ítems.** Debido a que las reseñas son cadenas de texto, lo normal es que contengan palabras que no aportan información, también llamadas palabras vacías *(stop words)*. La eliminación de estas es muy importante, puesto que reduce el número de palabras y por ende la indexación de las mismas es más rápida. Además, los resultados suelen ser más acertados, ya que la selección de palabras es más cercana a un conjunto de palabras clave.

2. **Tokenizar**[2] **la información que va a ser evaluada.** En este caso se tokenizan las categorías y las reseñas de los usuarios para cada ítem y luego son insertadas en un vector de cadenas *(strings)*. Este será el vector de información importante para cada ítem.

3. **Crear el vocabulario.** El vocabulario de nuestro algoritmo va a ser el conjunto de todos los *tokens* únicos de reseñas y categorías, por lo que es representado por un vector de cadenas *(strings)*.

4. **Crear matriz con la frecuencia de cada token para cada ítem.** Para realizar esto se ha implementado un *Count Vectorizer (figura 3.3)*, una función que recorre cada uno de los vectores de información importante de ítems y para cada una de las palabras del vocabulario se cuenta cuantas veces aparecen cada una de ellas en cada ítem.

	Palabras del Vocabulario			
Ítems	Palabra 1	Palabra 2	...	Palabra n
Ítem 1	0	1	6	2
Ítem 2	1	4	9	4
...	3	2	0	8
Ítem m	0	5	1	0

Figura 3.3: Ejemplo de *Count Vectorizer*

5. **Calcular similitud.** Como fue explicado en el apartado 1.1.1, la fórmula de la similitud es utilizada en vectores. En este caso se tiene una matriz con frecuencias de palabras, por lo que para poder aplicarla hay que recorrer cada una de las filas de la matriz y para cada una de ellas, calcular el ítem actual con todas las filas nuevamente. El resultado de esto es una matriz *(figura 3.4)* donde la diagonal va a ser 1 y el resto serán números decimales comprendidos entre 0 y 1. Se sabe de antemano que la diagonal va a ser igual a 1, porque el resultado de cada una de esas celdas viene del cálculo de la similitud coseno de un ítem consigo mismo. Cabe destacar que en esta matriz los primeros k elementos se corresponden los

[2] *Tokenizar*: Dividir el texto en las unidades que lo conforman.

posibles ítems a recomendar y en los últimos l están los ítems que han sido valorados positivamente por el usuario, por lo que se sabe que la matriz es cuadrada y tiene un rango igual a $k \times l$.

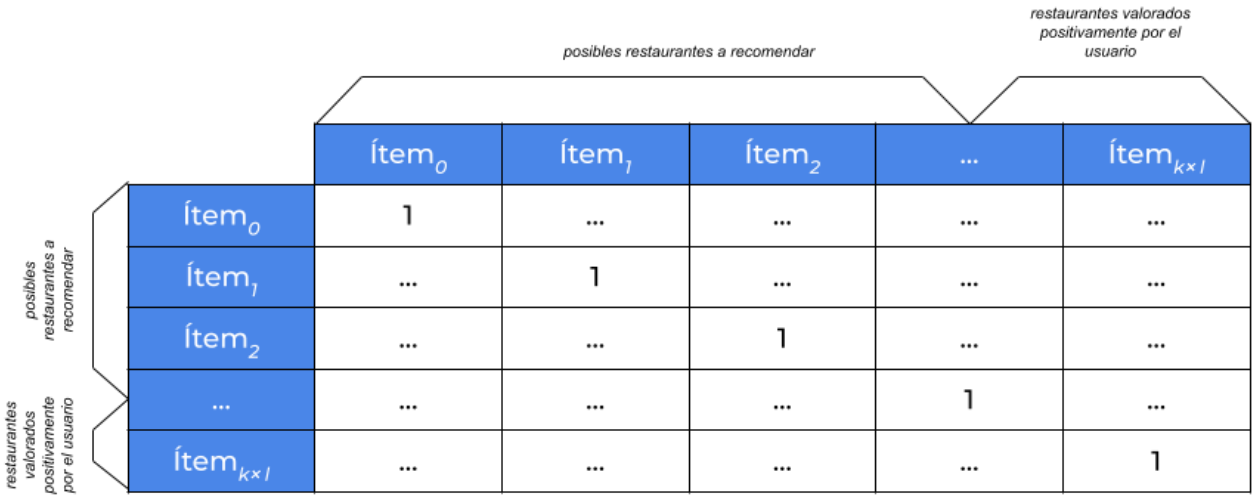


Figura 3.4: Ejemplo de matriz con similitud calculada

6. **Obtener los ítems que van a ser recomendados *(figura 3.5)*.** Una vez se tiene la matriz calculada, se quiere obtener que ítems pueden ser de interés para el usuario activo. Como se tiene almacenados los ítems que el usuario ha valorado positivamente, interesa saber que otros ítems de los disponibles son similares a ellos. Para esto se va a recorrer las últimas l filas que representan los ítems que le han gustado al usuario para luego recorrer las primeras k columnas y almacenar en cada posición de un vector un par con el ítem y su similitud correspondiente.

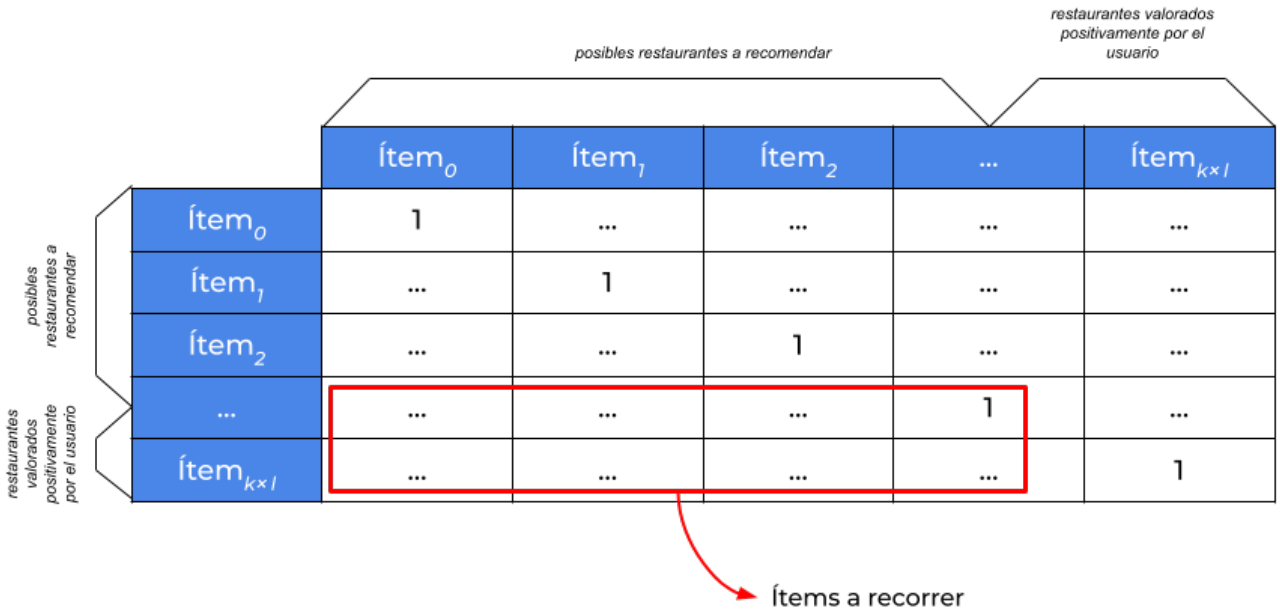


Figura 3.5: Ejemplo de obtención de ítems a recomendar

Si el usuario ha valorado más de un ítem con interioridad, al recorrer las columnas en las siguientes iteraciones, el valor ya va a estar almacenado, pero en este caso

se actualiza la similitud si la nueva es mayor a la que ya se había almacenado. Al terminar este proceso, solamente se ordena este vector por la similitud de manera descendente para que el primer par de similitud - ítem sea el que mayor similitud tiene con el perfil del usuario activo y en última posición esté el par menos similar. Puede llegar un punto en el que los últimos elementos tengan una similitud nula o igual a cero. En esos casos se ordenan estos ítems de forma descendente según su puntuación global.

3.2.2. Base de Datos

La base de datos juega un papel crucial de una aplicación, puesto que es la responsable de almacenar los datos. En este caso, se ha usado la base de datos *NoSQL* [31] MongoDB[3]. Esta base de datos orientada a documentos almacena los datos en una estructura de datos *BSON*[4], en lugar de en tablas, como hacen las clásicas bases de datos relacionales. Al estar implementado en el lenguaje de programación *C++*, esta base de datos aprovecha mucho los recursos de la máquina donde se ejecuta. Algunas características[3] de *MongoDB son:*

- Los datos son adaptados directamente a los tipos de datos nativos de la aplicación para la mayoría de lenguajes.
- Velocidad y manejo de los datos.
- Facilidad para escalar el proyecto, tanto horizontal como verticalmente.
- Es muy flexible, ya que no es necesario tener una estructura de datos predefinida y se adapta a cualquier tipo de datos. Esto es, de gran utilidad cuando se desconoce los datos de entrada o no son fijos.
- Facilita el tratamiento masivo de datos, ya que al tratarse de una base de datos basada en documentos, las consultas son mucho más rápidas y permite un mayor volumen de dato con respecto a bases de datos relacionales.
- Es una tecnología muy conocida por lo que tiene una comunidad y una gran cantidad de documentación.
- Un buen soporte por parte de los creadores, lo que hace que la tecnología vaya mejorando y este siempre lo más actualizada posible.

Algunas de estas características son comunes para las bases de datos *NoSQL*.

Los documentos previamente descritos son almacenados en una colección. Un ejemplo sería: Si se tiene un documento con atributos como: Nombre, Apellido, Edad, Residencia... La colección en la que puede estar almacenado es en la llamada *Persona*.

Son las características mencionadas anteriormente y la facilidad de integración que tiene con la tecnología utilizada para el *Backend* los motivos por lo que se ha elegido utilizar *MongoDB* como base de datos de este proyecto.

[3] *Mongo*: `www.mongodb.com`
[4] *BSON*: Representación binaria de *JSON*

3.2.3. Backend

En el lado del *Backend* las tecnologías utilizadas han sido *Java* [8], el cual se apoya *Spring Boot*[5]. Esta herramienta permite desarrollar arquitecturas basadas en servicios y micro-servicios sin tener que realizar toda la configuración que eso conllevaría normalmente.

Para implementar el *Backend* de una aplicación, hay diferentes opciones [17], como pueden ser *ExpressJS*[6] y *Django*[7]. No obstante, aunque *Django* y *ExpressJS* puedan llegar a ser más fáciles de comprender, *Spring Boot* ofrece un *Backend* muy robusto que es fácil de configurar, iniciar e integrar, tiene una alta seguridad, puede manejar varias peticiones a la vez y a día de hoy, es muy demandado [24].

Al ser *Spring Boot* un *framework* que utiliza *Java* como lenguaje de programación, al crear el proyecto de cero se necesita una herramienta para ello. Se puede usar *Gradle*[8] o *Maven*. La mayor diferencia entre estas es que *Gradle* es una herramienta para la automatización de la compilación, mientras que *Maven* es una herramienta de gestión y comprensión de proyectos. Al igual que para el proyecto que contiene el sistema recomendador [3.2.1], se ha utilizado la herramienta *Maven* para mantener la misma estructura de ficheros a lo largo del Trabajo de Fin de Grado.

El Backend de esta aplicación contiene una API REST [20] que se encarga de que los datos puedan ser accedidos desde la interfaz de usuario, de que la base de datos pueda almacenar los datos que son manipulados desde la interfaz de usuario y de poder ejecutar el algoritmo recomendador. Una API REST contiene un conjunto de reglas o *endpoints* que definen como pueden comunicarse y conectarse el proveedor de información y el usuario.

La estructura de ficheros es bastante extensa. Como se puede ver en el diagrama de componentes *Figura 3.2* se cuenta con diferentes componentes importantes: *recursos, modelos, repositorios, servicios y controladores*.

En primer lugar, se encuentran los recursos o *resources*. Ahí es donde se almacenan los recursos que van a ser utilizados en el proyecto. Entre ellos se encuentran el ejecutable que contiene el algoritmo recomendador *[3.2.1]*, los ficheros *JSON* que serán pasados como entrada al algoritmo y donde se almacenará la salida del mismo.

Por un lado, uno de los componentes más importante de un proyecto en *Spring Boot* es el modelo. Un modelo es una clase que representa un objeto con sus respectivos atributos y métodos. En este proyecto, como se puede apreciar en la siguiente imagen *(figura 3.6)*, se han implementado tres modelos: *Item, User* y *Feedback*.

[5] *Spring Boot*: `www.spring.io/projects/spring-boot`
[6] *ExpressJS*: `https://expressjs.com/es/`
[7] *Django*: `https://www.djangoproject.com/`
[8] *Gradle*: `https://gradle.org/`

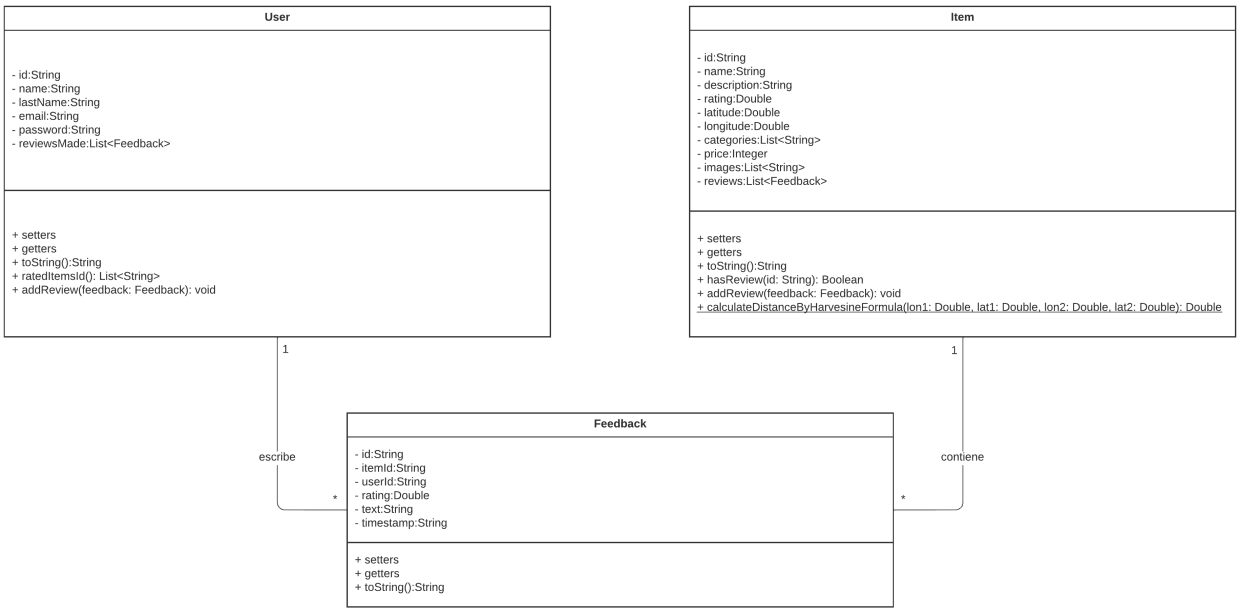


Figura 3.6: Diagrama de Clases

- Clase *Item*: hace referencia a la colección *Item* de la base de datos. Tiene diferentes validadores como que el identificador debe ser único, el nombre debe tener entre 1 y 100 caracteres, la descripción tiene un máximo de 2000 caracteres y los campos de longitud y latitud no pueden estar vacíos. Si estos requisitos no se cumplen, el objeto se considera inválido.

- Clase *User*: hace referencia a la colección *User* de la base de datos. Tiene diferentes validadores como que el identificador debe ser único, el nombre y apellidos deben tener entre 1 y 100 caracteres cada uno, el email debe ser único, su campo no puede estar vacío y tiene que tener un formato de correo electrónico válido. Además, su contraseña debe tener entre 8 y 30 caracteres, debe tener mínimo una letra en mayúscula y otra en minúscula, al menos un dígito y no acepta espacios. Si estos requisitos no se cumplen, el objeto se considera inválido.

- Clase *Feedback*: No hace referencia a ninguna colección de la base de datos, pero forma parte de todos los documentos de ambas colecciones.

Por otro lado, para que estos modelos funcionen como es de esperar, es necesario crear un repositorio para cada uno de ellos. Los repositorios son componentes esenciales dentro de la capa de persistencia de la aplicación, la encargada de guardar y recuperar las entidades necesarias. Es por ello que los repositorios extienden la interfaz de *MongoRepository* y están conectados directamente con nuestra base de datos, *MongoDB*. Una gran ventaja que tiene Spring Boot es que cuenta con la clase *MongoTemplate* y con la interfaz *MongoRepository*, por lo que la integración del framework con la base de datos es automática sin requerir mayor configuración.

Para modificar el contenido de los repositorios se requiere un servicio para cada uno de ellos. Como bien indica el nombre del componente, estos pertenecen a la capa de servicios. Los servicios son los encargados de resolver la lógica de negocio del aplicativo.

Una vez se han creado los servicios, estos son consumidos desde los controladores, que pertenecen a la capa de controladores. El aplicativo contiene tres controladores:

- **Item Controller:** Controlador para gestionar los datos relacionados con los ítems. Este controlador tiene como base o prefijo para los *endpoints* declarados en él el valor *"/items"*.
- **User Controller:** Controlador para gestionar los datos relacionados con los usuarios. Este controlador tiene como base o prefijo para los *endpoints* declarados en él el valor *"/users"*.
- **Recommendation Controller:** Controlador para gestionar la recomendación. Tiene como base o prefijo para los *endpoints* declarados en él el valor *"/recommendation"*.

Estos son capaces de recibir peticiones HTTP *(Hypertext Transfer Protocol)* y devolver contenidos de la capa de servicios. Una petición HTTP es un mensaje enviado por un cliente para iniciar una acción en el servidor. Los diferentes métodos HTTP o verbos implementados son los siguientes:

- **GET**: Solicita la consulta datos.
- **POST**: Solicita la inserción de datos.
- **DELETE**: Solicita la eliminación de datos.

Cabe destacar que para cada uno de los controladores, se ha habilitado el protocolo de recursos cruzados CORS[9] *(Cross Origin Resource Sharing)* para el origen `http://localhost:8081/`, que es donde está desplegado el *Frontend* del proyecto y donde se van a solicitar los datos. De esta manera, se restringen cualquier otro tipo de peticiones que no vengan de ese origen.

Los *"endpoints"* implementados en esta *API REST* se presentan a continuación:

- **Endpoint *"/items"* -** Acepta las siguientes peticiones HTTP:
  - Petición *GET*: Retorna todos los ítems almacenados junto con el código de estado satisfactorio *HTTP 200 OK*.
  - Petición *POST* a *"/items"*: Recibe un cuerpo con la estructura predefinida en *[3.2.1]* para un ítem. Si el cuerpo es correcto y el identificador del ítem no está ya asignado a otro ítem, se almacena el nuevo ítem y se retorna el código de estado satisfactorio *HTTP 200 OK*. Si al contrario el cuerpo está bien definido y el identificador ya existe, este será actualizado con la nueva información proporcionada y se retornará el código de estado satisfactorio *HTTP 200 OK*. Si el cuerpo no está bien definido, no se inserta el ítem y se retorna el código de estado *HTTP 400 Bad Request*.
- **Endpoint *"/items/{id}"* -** Recibe la variable *id* que representa un identificador de un ítem y acepta las siguientes peticiones HTTP:
  - Petición *GET*: Si existe un ítem con el *id* recibido, se retorna el ítem correspondiente junto con el código de estado satisfactorio *HTTP 200 OK*. Si no existe, se retorna el código de estado *HTTP 404 Not Found*.

[9] *CORS*: `https://developer.mozilla.org/es/docs/Web/HTTP/CORS`

- Petición *POST*: Recibe un cuerpo con la estructura predefinida en *[3.2.1]* para un ítem y un *id* del ítem que se desea modificar. Si el cuerpo es correcto y el identificador proporcionado coincide con el de otro ítem, este se actualiza con la nueva información proporcionada y se retorna el código de estado satisfactorio *HTTP 200 OK*. Si al contrario el cuerpo no está bien definido o el identificador no se corresponde con el de ningún ítem, se retorna el código de estado *HTTP 400 Bad Request* o el código de estado *HTTP 404 Not Found*, respectivamente.
- Petición *DELETE*: Recibe un *id* del ítem que se desea eliminar. Si el identificador proporcionado coincide con el de otro ítem, este es eliminado de la base de datos y se retorna el código de estado satisfactorio *HTTP 200 OK*. Si no, se retorna el código de estado *HTTP 404 Not Found*.

- **Endpoint *"/items/feedback/{id}"* -** Recibe la variable *id* que representa un identificador de un ítem y acepta las siguientes peticiones HTTP:
  - Petición *POST*: Recibe un cuerpo con la estructura predefinida en *[3.2.1]* para una reseña y un *id* del ítem al que se le desea hacer la reseña. Si el cuerpo es correcto y el identificador proporcionado coincide con el de otro ítem, a este se le añade la reseña proporcionada y se retorna el código de estado satisfactorio *HTTP 200 OK*. Si al contrario el cuerpo no está bien definido o el identificador no se corresponde con el de ningún ítem, se retorna el código de estado *HTTP 400 Bad Request* o el código de estado *HTTP 404 Not Found*, respectivamente.
- **Endpoint *"/users"* -** Acepta las siguientes peticiones HTTP:
  - Petición *GET*: Retorna todos los usuarios almacenados junto con el código de estado satisfactorio *HTTP 200 OK*.
  - Petición *POST*: Recibe un cuerpo con la estructura predefinida en *(figura 3.6)* para un usuario. Si el cuerpo es correcto y el identificador del usuario y el email proporcionado no están ya asignados a otro usuario, se almacena el nuevo usuario, junto al código de estado satisfactorio *HTTP 200 OK*. Si al contrario el cuerpo está bien definido y el identificador ya existe, este será actualizado con la nueva información proporcionada y se retorna junto al código de estado satisfactorio *HTTP 200 OK*. Si el cuerpo no está bien definido, no se inserta el usuario y se retorna el código de estado *HTTP 400 Bad Request*.
- **Endpoint *"/users/{id}"* -** Recibe la variable *id* que representa un identificador de un usuario y acepta las siguientes peticiones HTTP:
  - Petición *GET*: Si existe un usuario con el *id* recibido, se retorna el mismo junto al código de estado satisfactorio *HTTP 200 OK*. Si no existe, se retorna el código de estado *HTTP 404 Not Found*.
  - Petición *POST*: Recibe un cuerpo con la estructura predefinida en *(figura 3.6)* para un usuario y un *id* del usuario que se desea modificar. Si el cuerpo es correcto y el identificador proporcionado coincide con el de otro usuario, este se actualiza con la nueva información proporcionada y se retorna el código de estado satisfactorio *HTTP 200 OK*. Si al contrario el cuerpo no está bien definido o el identificador no se corresponde con el de ningún usuario, se retorna el código de estado *HTTP 400 Bad Request* o el código de estado *HTTP 404 Not Found*, respectivamente.

- Petición *DELETE*: Recibe un *id* del usuario que se desea eliminar. Si el identificador proporcionado coincide con el de otro usuario, este es eliminado y se retorna el código de estado satisfactorio *HTTP 200 OK*. Si no, se retorna el código de estado *HTTP 404 Not Found*.

- **Endpoint *"/users/feedback/{id}"* -** Recibe la variable *id* que representa un identificador de un ítem y acepta las siguientes peticiones HTTP:

  - Petición *POST*: Recibe un cuerpo con la estructura predefinida en *(figura 3.6)* para una reseña y un *id* del usuario que escribió la reseña. Si el cuerpo es correcto y el identificador proporcionado coincide con el de otro usuario de la base de datos, a este se le añade la reseña proporcionada y se retorna el código de estado satisfactorio *HTTP 200 OK*. Si al contrario el cuerpo no está bien definido o el identificador no se corresponde con el de ningún usuario, se retorna el código de estado *HTTP 400 Bad Request* o el código de estado *HTTP 404 Not Found*, respectivamente.

- **Endpoint *"/users/login"* -** Acepta las siguientes peticiones HTTP:

  - Petición *GET*: Recibe dos parámetros, el email del usuario y la contraseña del mismo. Si existe algún usuario con esa combinación de email y contraseña, se retorna al usuario en cuestión junto al código de estado satisfactorio *HTTP 200 OK*. Si no, se retorna el código de estado *HTTP 400 Bad Request*.

- **Endpoint *"/recommendation"* -** Acepta las siguientes peticiones HTTP:

  - Peticion *GET*: Recibe como parámetros diferente información necesaria para obtener la recomendación del usuario:
    - *radius:* El radio en metros de la zona donde se desea realizar la recomendación.
    - *longitude*: La longitud de la coordenada en grados decimales del punto central de la zona donde se desea realizar la recomendación.
    - *latitude*: La latitud de la coordenada en grados decimales del punto central de la zona donde se desea realizar la recomendación.
    - *priceRange*: Una lista de dos valores entre 1 y 4, que representan la cota superior e inferior del presupuesto del usuario, siendo 1 restaurantes baratos y 4 restaurantes de lujo.
    - *categories*: Una lista donde cada elemento es una categoría.
    - *userId*: El identificador del usuario al que se le desea realizar la recomendación.

    Con esta información, se filtran los restaurantes que se ajusten a las propiedades recibidas.

    1. Primero se filtran los ítems de la base de datos que tengan una distancia entre él y las coordenadas recibidas por parámetro igual o menor al radio recibido. Esto se hace con ayuda de la fórmula de *Harvesine* [27], que calcula la distancia mínima entre dos puntos de un globo sabiendo su longitud y su latitud.
    2. Luego, se filtran los ítems que tengan al menos una de las categorías especificadas y que entren dentro del presupuesto del usuario, si se aplica.

3. El siguiente paso es obtener los ítems que han reseñado los usuarios positivamente, a partir del identificador proporcionado.
4. Se escriben los ficheros *userLikedItems.json* e *items.json* para poder ser pasados como entrada al algoritmo recomendador*[3.2.1]*.
5. Ejecutar el algoritmo recomendador, leer el fichero con la recomendación y retornarla.

3.2.4. Frontend

Por una parte, para el *Frontend* se ha utilizado el lenguaje de programación *JavaScript* [11], un lenguaje interpretado que es mayoritariamente utilizado para la programación *web*. Además, se ha desarrollado la interfaz del usuario con ayuda de *Vue.js*[10], un *framework* progresivo con una curva de aprendizaje bastante accesible.

Existen otros *frameworks* populares [18], como pueden ser *React*[11] y *Angular*[12]. No obstante, este Trabajo de Fin de Grado ha sido creado en Vue, puesto que a diferencia de los otros dos, es más ligero [18] y más sencillo de aprender. Esto es debido a aspectos como su baja complejidad al crear un proyecto y la sintaxis familiar, ya que separa su estructura en el fichero con una sección para el *HTML*[13] y otra para *JavaScript*. Estas tecnologías son muy utilizadas, por lo que si el desarrollador ya las conoce, no debería de haber ningún problema a la hora de aprender *Vue.js*.

Algunas de las librerías de *Vue.js* que se han utilizado en este Trabajo de Fin de Grado son:

- **Vuetify**[14]: Una librería de componentes.
- **Vuex**[15]:Una librería para el manejo de estados, lo cual permite almacenar algunas variables u objetos de manera centralizada, segura y accesible en todas las páginas de nuestra aplicación.
- **Vue Router**[16]: Una librería de enrutamiento que permite asignarle a las diferentes páginas de la aplicación las rutas URL *(Uniform Resource Locator)* correspondientes.

Por otro lado, también se ha utilizado *Mapbox GL JS*[17], una librería de *Javascript* que permite, entre otros, visualizar y animar mapas interactivos. Una ventaja que ofrece esta librería frente a otras, como la de *Google Maps*[18] es que Mapbox ofrece 50.000 cargas de mapas mensuales gratuitas sin necesidad de insertar método de pago. Entre esa ventaja y el diseño moderno que tienen los mapas, esta fue la librería que mejor se ajustaba a las necesidades del proyecto.

[10] *Vue.js*: `www.v2.vuejs.org/v2/guide`
[11] *React:* `https://reactjs.org/`
[12] *Angular:* `https://angular.io/`
[13] *HTML:* `https://developer.mozilla.org/es/docs/Web/HTML`
[14] *Vuetify:* `https://vuetifyjs.com/en/`
[15] *Vuex:* `https://vuex.vuejs.org/`
[16] *Vue Router*: `https://router.vuejs.org/`
[17] *Mapbox*: `https://www.mapbox.com/`
[18] *Google Maps*: `https://developers.google.com/maps?hl=es-419`

Para la obtención de datos de restaurantes, se han estudiado diferentes APIs *(figura 3.7)*: la de *TripAdvisor*[19], la de *Google Places*[20], la de *Foursquare*[21] y la de *Yelp*[22]. Las diferencias encontradas son las siguientes:

[19] *TripAdvisor API*: `https://www.tripadvisor.com/developers`
[20] *Google Places API*: `https://developers.google.com/maps/documentation/places/web-service`
[21] *Foursquare Places API*: `https://developer.foursquare.com/docs/places-api-overview`
[22] *Yelp Fusion API*: `https://www.yelp.com/developers/documentation/v3/get_started`

API	Contenido	Plan de uso
TripAdvisor API	• Nombre • Puntuación global • Número total de reseñas • Localización (ej.: ciudad) • Información de ranking • Categoría (Hoteles/Restaurante…) • Conteo de puntuaciones (Del 1 al 5) • Premios • Latitud y Longitud • Precio (barato € - muy caro €€€€) • Tipo de comida • Dirección • Número reseñas con fotos	Límite de 25,000 llamadas al día y 100 llamadas por segundo • Rellenar un formulario para solicitar ser un partner aprobado. • Una vez hecho eso se puede aplicar para recibir la clave. Se realizó el primer paso, pero se denegó la solicitud.

Foursquare Places API	• Buscar lugares cercanos
	• Buscar Lugares según…
	  ○ Categoría
	  ○ Dirección
	  ○ Latitud y Longitud
	  ○ Lugares relacionados
	• Detalles del sitio (dado un ID)
	• Fotos del sitio
	• Reseñas
		Para usar la API, en principio no debería haber problema. Cada mes se añaden 200 $ a cada cuenta que son empleados para pagar las peticiones.

Google Places API	• Búsqueda por texto
	• Busca lugares según un área
	• Buscar detalles del sitio
	  ○ Dirección
	  ○ Estado del local
	  ○ Número de teléfono
	  ○ Latitud y longitud
	  ○ Horas de apertura
	  ○ Fotos
	  ○ Precio (barato, caro, …)
	  ○ Puntuación
	  ○ Reseñas
	  ○ Tipo
	  ○ Página web del sitio
	  ○ Número de reseñas
		Hay que poner tarjeta de crédito, aunque se supone que mientras estés por debajo del límite de llamadas diarias no te cobran.
		Al mes dan 200 $ gratis y el precio de cada llamada es de aproximadamente 0.032 $.
		
		Máximo 100 llamadas por segundo.

Yelp Fusion API	• Retorna hasta 1000 establecimientos basados en el criterio establecido (tipo, localización, latitud y longitud, radio, categoría, está abierto o no…)
	• Dado un ID da detalles como
	  ○ Nombre
	  ○ Número de teléfono
	  ○ Si está abierto
	  ○ Número de reseñas
	  ○ Categorías
	  ○ Puntuación
	  ○ Localización
	  ○ Coordenadas
	  ○ Fotos
	  ○ Precio
	  ○ Horario por días
	  ○ Horario especial
		Gratuito. Límite de 5000 llamadas al día que se restablecen a las 00:00 UTC.
		
		Este límite puede aumentar contactando con el soporte, pero probablemente sea de pago.

Figura 3.7: Estudio APIs

Teniendo en cuenta el tamaño de este proyecto y que la aplicación no va a ser publicada inicialmente, dadas las características de la API de *Yelp,* que esta es gratuita y no requiere la inserción de métodos de pago, ha sido la elegida para la obtención de información de restaurantes.

La estructura de la parte del *front-end* del aplicativo está dividida en diferentes directorios y ficheros importantes:

- *“‘public/”:* Este directorio aloja los ficheros estáticos que no son procesados por *Vue.js*.
- *“‘src/apis/”:* Aquí se alojan todas las funciones que hacen llamadas a APIs, tanto a la API REST implementada [3.2.3] como a la API de *Yelp*.
- *“‘src/componets/”:* En esta carpeta se encuentran los componentes reutilizables de la aplicación, como puede ser un botón o un panel de navegación, entre otros. Un componente es un sub elemento de la página que contiene código *JavaScript*, *HTML* y *CSS*.
- *“‘src/views/”:* Las vistas son también componentes. No obstante, se diferencia en que estas muestra la estructura general de una página.
- *“‘src/plugins/”:* En este directorio se encuentran funcionalidades externas que se utilizan en la aplicación, como puede ser el módulo de *Vuetify* u otras utilidades.
- *“‘src/store/”:* Aquí se gestiona la información en una especie de almacén proporcionado por *Vuex*.
- *“‘src/router.js":* Este fichero contiene la configuración de las rutas de la aplicación y como van a ser utilizadas.
- *“‘src/main.js”:* El fichero *main.js* es el principal del proyecto, ya que se encarga de arrancarlo y es insertado en el *index.html* principal de la carpeta *“‘public/”*.

Vistas

Uno de los aspectos más importantes de este Trabajo de Fin de Grado es la implementación de las diferentes vistas para poder cumplir con los casos de uso definidos en *(figura 3.1)*. La primera vista que se encuentra el usuario es la de inicio de sesión *(figura 3.8)*.

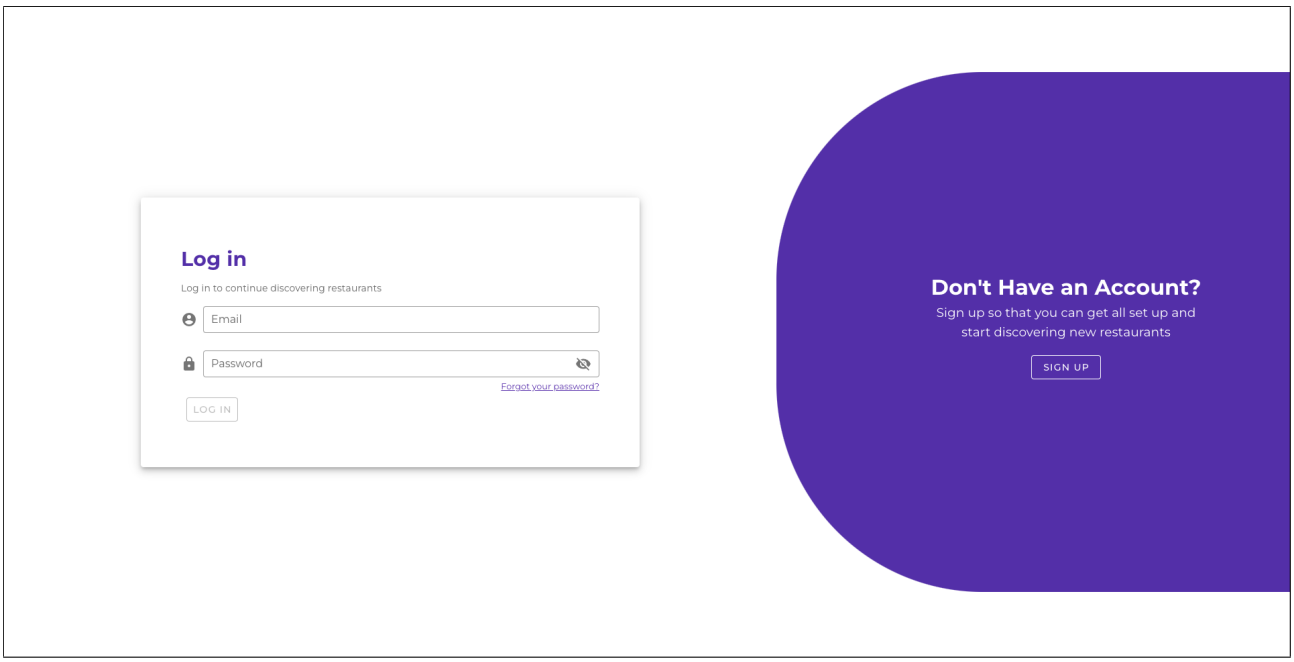


Figura 3.8: Vista de inicio de sesión

Esta vista habilita al usuario el iniciar sesión. Como se observa en la imagen, el botón para poder iniciar sesión está deshabilitado hasta que ambos campos estén rellenados. Una vez esto se haya completado, se hace una petición *GET* al endpoint *"/user/login"* de la *API REST* implementada para comprobar si el usuario está registrado y de ser así, le lleva a la página principal. Por otro lado, en el panel derecho de esta página aparece un botón para ser derivado a la página de registro *(figura 3.9)*.

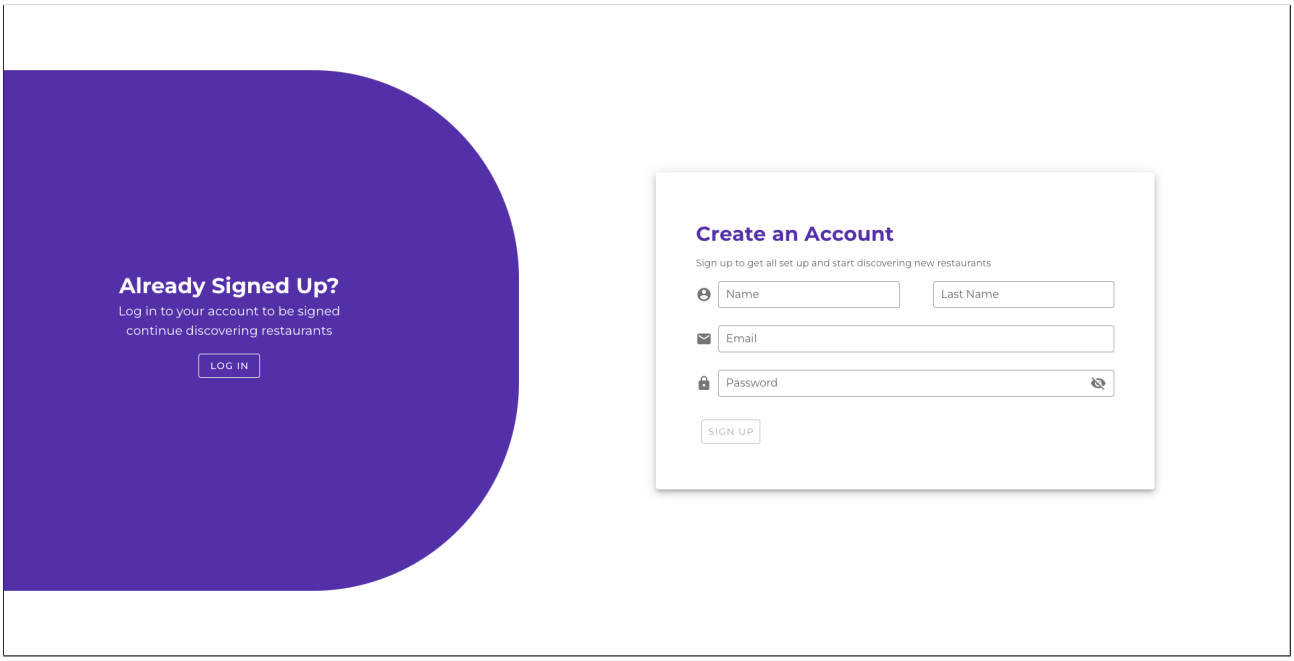


Figura 3.9: Vista de registro

Esta página es muy similar a la anterior, pero el usuario debe introducir sus datos para poder presionar el botón de registro. La contraseña debe tener entre 8 y 30 caracteres, tener mínimo una minúscula, una mayúscula y un dígito. el email debe tener un formato de correo electrónico correcto. Cuando estos campos hayan sido cumplimentados, se hace una petición *POST* al endpoint *"/users"* de la *API REST* implementada con la información proporcionada para poder crear el usuario. Si se pudo crear correctamente, se redirige al usuario a la página de inicio. Al igual que en la página anterior, en el panel izquierdo se le da la opción de redirigir al usuario a la página de inicio de sesión.

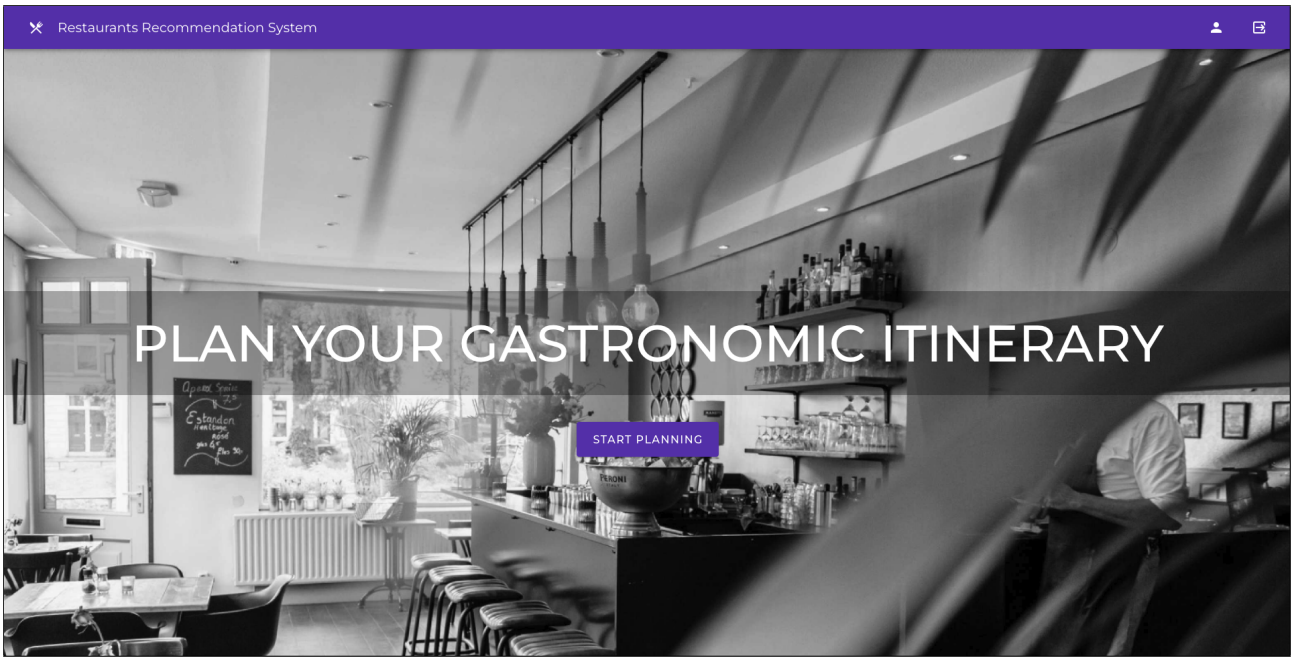


Figura 3.10: Vista principal

Una vez el usuario inicia sesión satisfactoriamente, se encuentra con la página principal de la aplicación *(figura 3.10)*. Podrá empezar a planear su itinerario gastronómico pulsando el botón *"START PLANNING"* en el centro de la pantalla, acceder a su perfil de usuario pulsando el icono de usuario en la esquina superior derecha o cerrar su sesión pulsando el icono de al lado. Estos dos últimos iconos forman parte del componente *Navigation*, que representa una barra de navegación que está disponible en todas las pantallas de acceso para usuarios que han iniciado sesión. Cabe destacar que si se pulsa el icono de cubiertos en la esquina superior izquierda, se volverá a la página principal. Si el usuario pulsa su el icono del perfil, se le llevará a la vista de perfil de usuario *(figura 3.11)*.

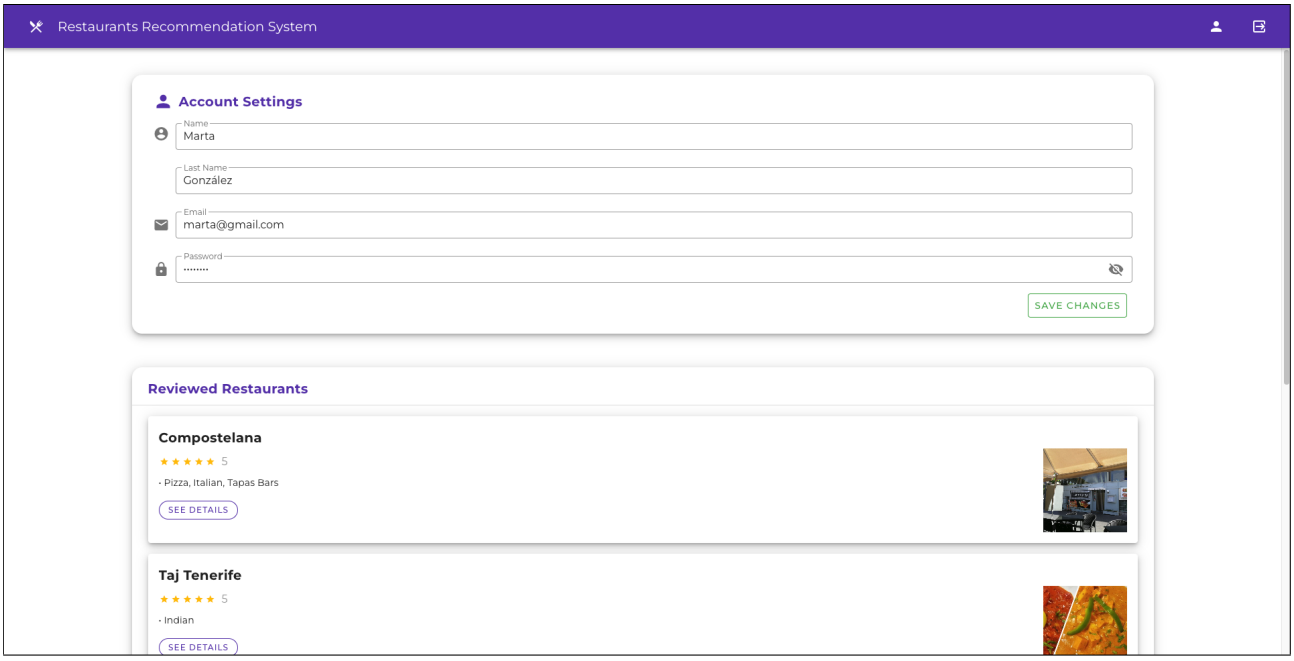


Figura 3.11: Vista de perfil de usuario

Como se puede observar en la imagen, esta vista se divide en dos componentes. El primero que se posiciona en la parte superior es el de la información del usuario, donde el mismo puede consultar la información, incluso modificarla siempre y cuando esta se adapte a las restricciones explicadas en las pantallas de inicio y registro. Si se satisfacen estos requisitos, se puede hacer una petición *POST* con los cambios al *endpoint "/users/{userId}"* de la *API REST* implementada pulsando el botón *"SAVE CHANGES"*.

El segundo componente está en la parte inferior y contiene una lista de los restaurantes reseñados por el usuario. Cada uno de estos elementos de la lista son componentes personalizados *Card* a los cuales se les pasa por parámetro toda la información del restaurante tras haber hecho una petición *GET* al endpoint *"/items/{itemId}"* de la *API REST* implementada para cada restaurante. Al pulsar sobre el botón *"SEE DETAILS"* se abre el componente *MoreDetails* con una vista detallada del restaurante en cuestión *(figura 3.12)*. En ella se podrán visualizar las reseñas que tiene, crear nuevas reseñas, consultar la puntuación general, la ubicación, una fotografía del lugar y el nombre.

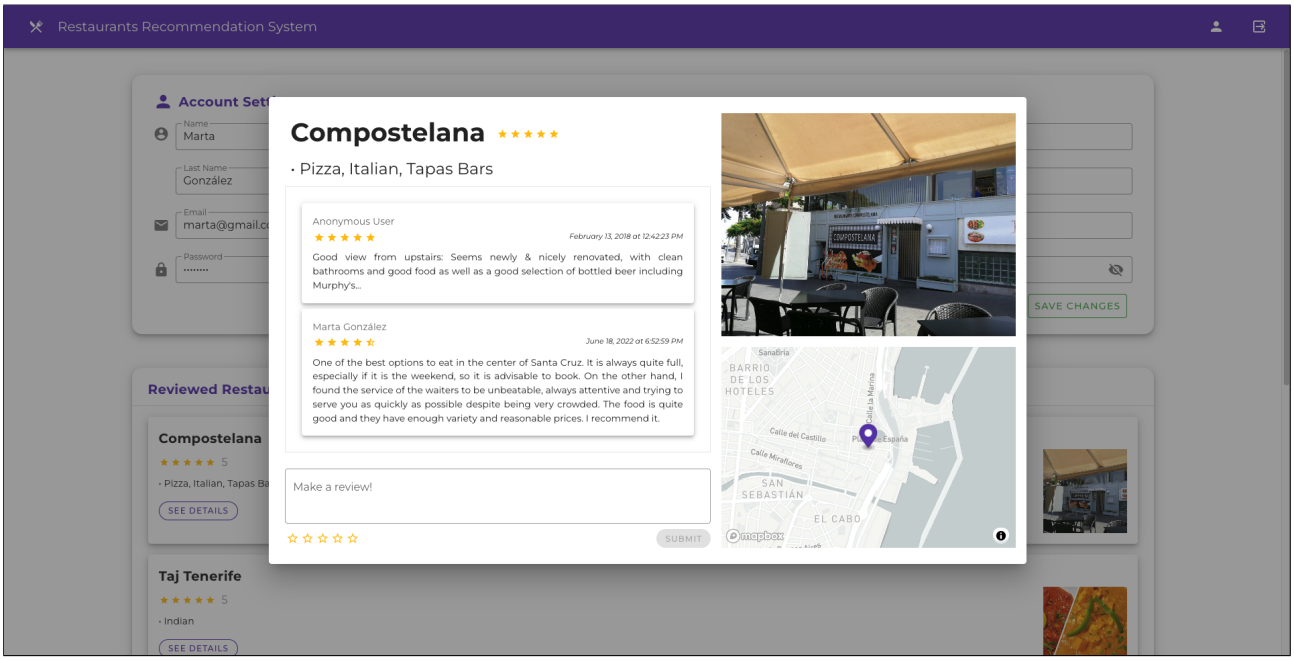


Figura 3.12: Vista detallada de un restaurante

Estos dos últimos componentes son recurrentes en la aplicación y se verán en vistas futuras.

Si se vuelve a la pantalla de inicio y se pulsa sobre el botón *"START PLANNING"*, se inicia el proceso de planificación gastronómica. Este proceso se encuentra en la vista de planificador, que contiene 5 sub-vistas. La primera *(figura 3.13)*, da la opción de escoger que comidas desea incluir el usuario en su itinerario gastronómico o visualizar los restaurantes mejor valorados cerca de su ubicación.

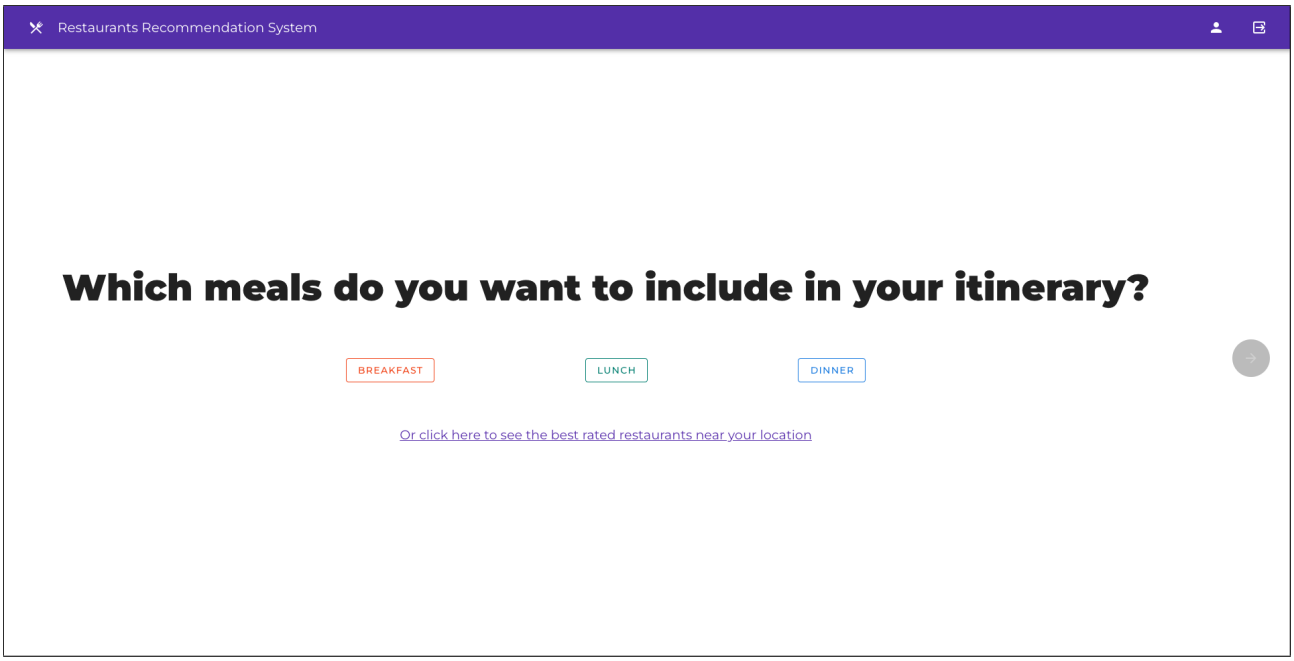


Figura 3.13: Vista principal del planificador

La opción de ver los restaurantes mejor valorados de la zona *(figura 3.14)* solo está disponible si el usuario ha concedido los permisos de ubicación necesarios. Si se pulsa sobre esa opción, se le deriva a la vista de restaurantes mejores valorados, que hace una petición *GET* a la API de *Yelp Fusion* con la ubicación del usuario por parámetro. Esta pantalla vuelve a mostrar el componente *Card* y *MoreDetails* explicados anteriormente.

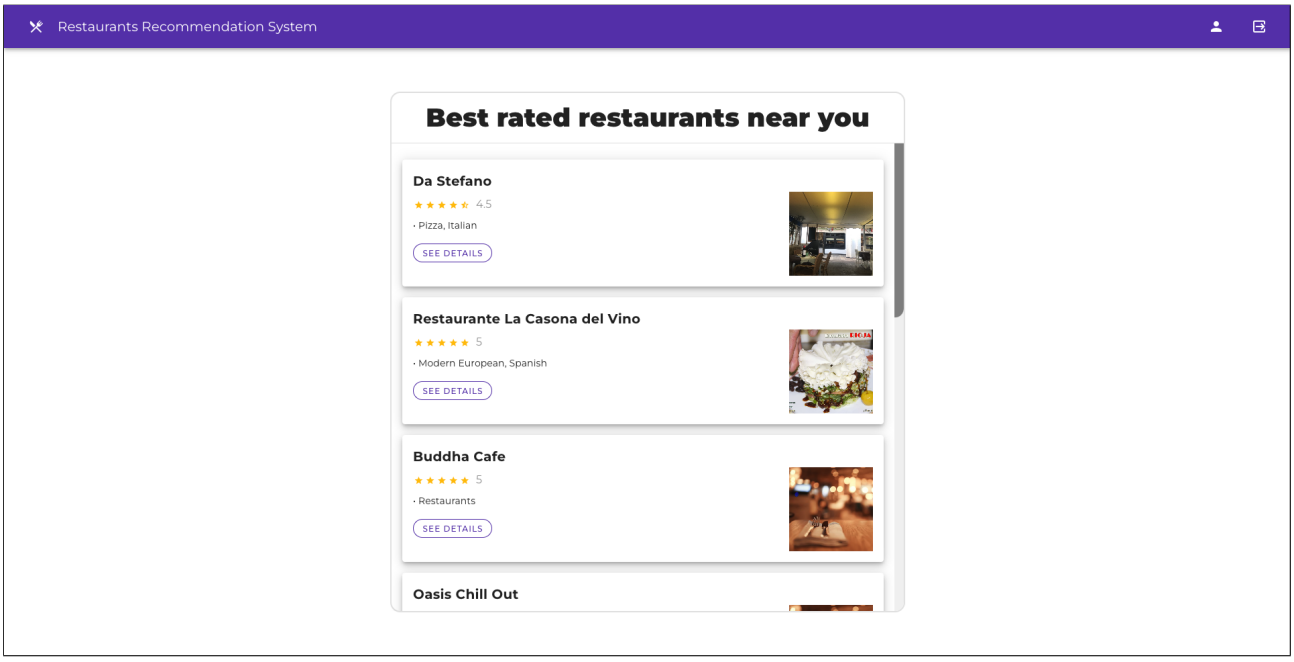


Figura 3.14: Vista de restaurantes mejor valorados

Si se vuelve a la primera vista del planificador *(figura 3.13)* se puede observar como la flecha de la derecha está desactivada. Esto es porque ninguna comida ha sido seleccionada. Desde que se seleccione al menos una, ya se podrá avanzar al siguiente apartado.

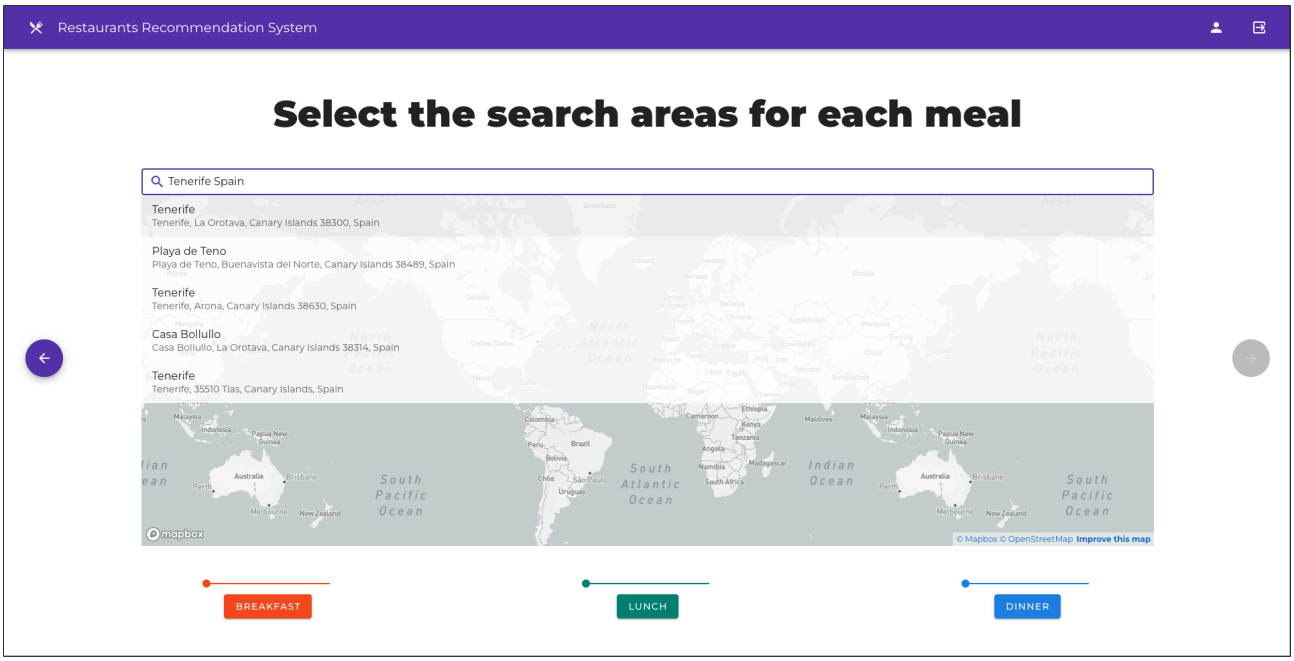


Figura 3.15: Vista de selección de zonas para cada comida seleccionada

En esta vista *(figura 3.15)* se pueden apreciar diferentes componentes. El primero de ellos es el mapa que se ve en el fondo, que es implementado con la librería de *Mapbox*. Si el usuario ha dado permiso para acceder a su ubicación, este mapa muestra una animación para mostrar más de cerca la localización del usuario. Este mapa es interactivo y el usuario debe interactuar con él para poder acceder al siguiente panel de configuración. Esto se hace seleccionando una un punto y zona en el mapa para cada una de las comidas seleccionadas en el apartado anterior. En la parte inferior de la pantalla se ve un botón y un deslizante para cada una de ellas. Se deberá elegir un punto en el mapa y modificar el radio de la zona en la que va a estar el usuario para esa comida en el deslizador del componente. Este radio está en metros. Por otro lado, encima del mapa se encuentra una barra de búsqueda. Este componente *Searchbar* consulta la API de *Mapbox* para obtener lugares en todo el mundo que cacen con el texto introducido por el usuario. Una vez se seleccione, el mapa muestra una animación para acercarse a esta posición.

Cuando se seleccionan los puntos, se desbloquea la flecha de la derecha para poder pasar a la siguiente vista y especificar más preferencias.

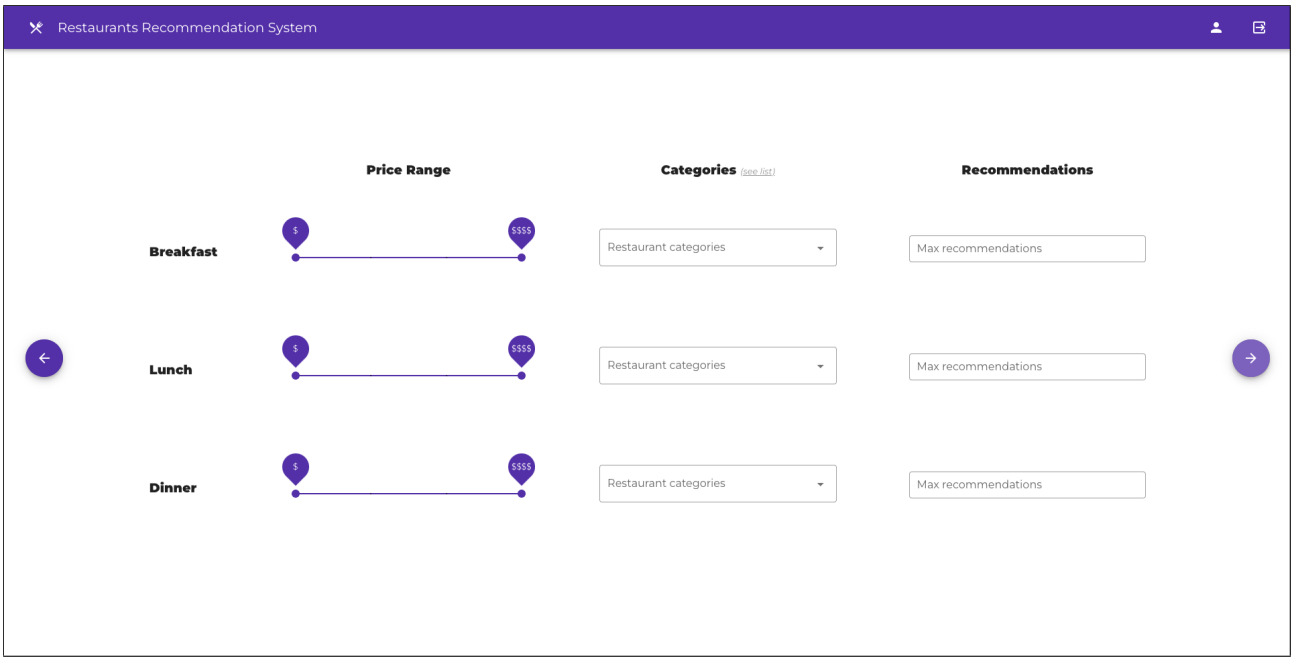


Figura 3.16: Vista de selección de preferencias

Esta sub-vista *(figura 3.16)* pretende darle al usuario la posibilidad de poder especificar diferentes preferencias para cada uno de las comidas seleccionadas: *presupuesto, categorías* y *número de recomendaciones*. Las categorías proporcionadas son las que vienen incluidas en la API de *Yelp Fusion* tras haber sido adaptadas y analizadas según las necesidades de la aplicación, ya que la lista original tiene más categorías, además de las de los restaurantes. si el usuario no especifica nada en esta vista se aplicarán las preferencias por defecto: restaurantes de cualquier rango salarial, restaurantes de todas las categorías posibles y cinco recomendaciones por comida. Cabe destacar que si la comida *"Desayuno"* fue seleccionada, se le aplicarán algunas categorías por defecto para que el tipo de restaurantes que recomiende sea típico para desayunar, cafetería, cafés, etcétera.

Al pulsar sobre la flecha de la derecha, se recoge la información de los restaurantes que están dentro de esas zonas haciendo peticiones *GET* a la API de *Yelp Fusion*, se almacenan en la base de datos si no existen previamente en ella y se envía una petición *GET* al endpoint *"/recommendation"* de la *API REST* implementada para cada una de las comidas. Cuando se realicen las recomendaciones se muestra la última vista del planificador *(figura 3.17)*:

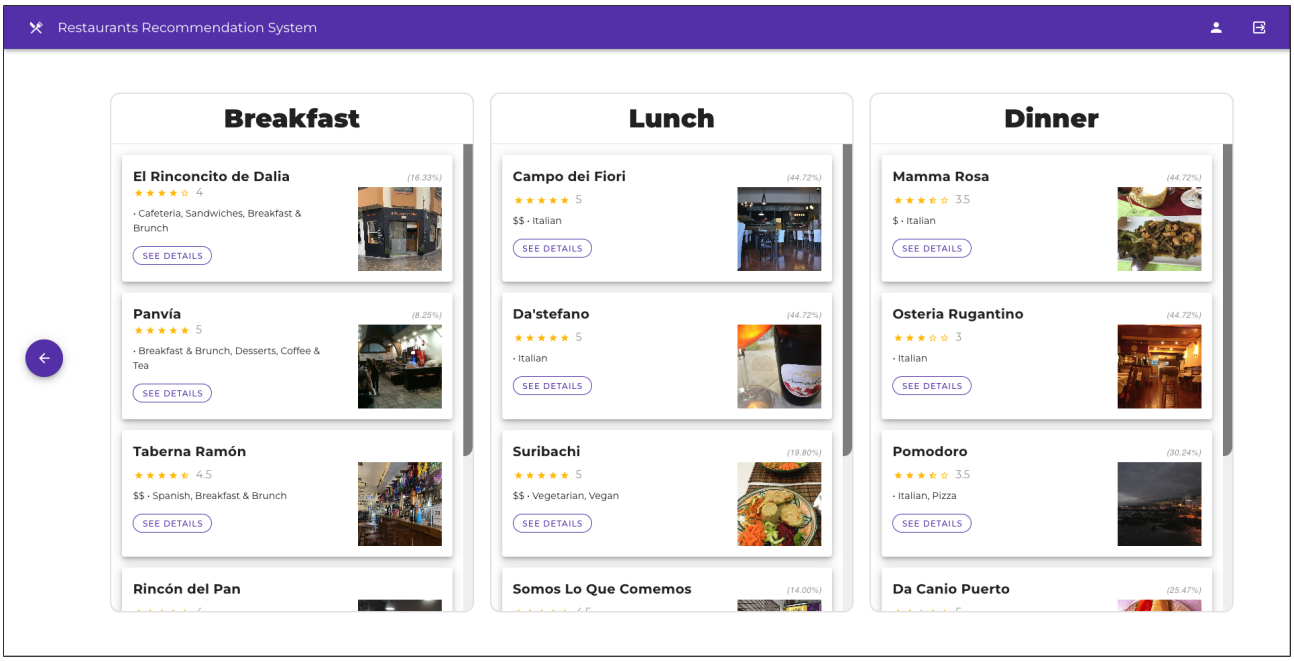


Figura 3.17: Vista de resultados de la recomendación

Esta vista es muy similar a las descritas anteriormente, ya que cada uno de los paneles para cada una de las comidas contiene una lista de componentes *Card* y *MoreDetails*. La única diferencia es que en este caso se muestra el porcentaje de similitud que tiene el restaurante con el perfil del usuario encima de la foto de cada restaurante.

Por último, existe la vista de error *(figura 3.18)*, que sirve para indicarle al usuario que ha intentado acceder a una página que no está definida.

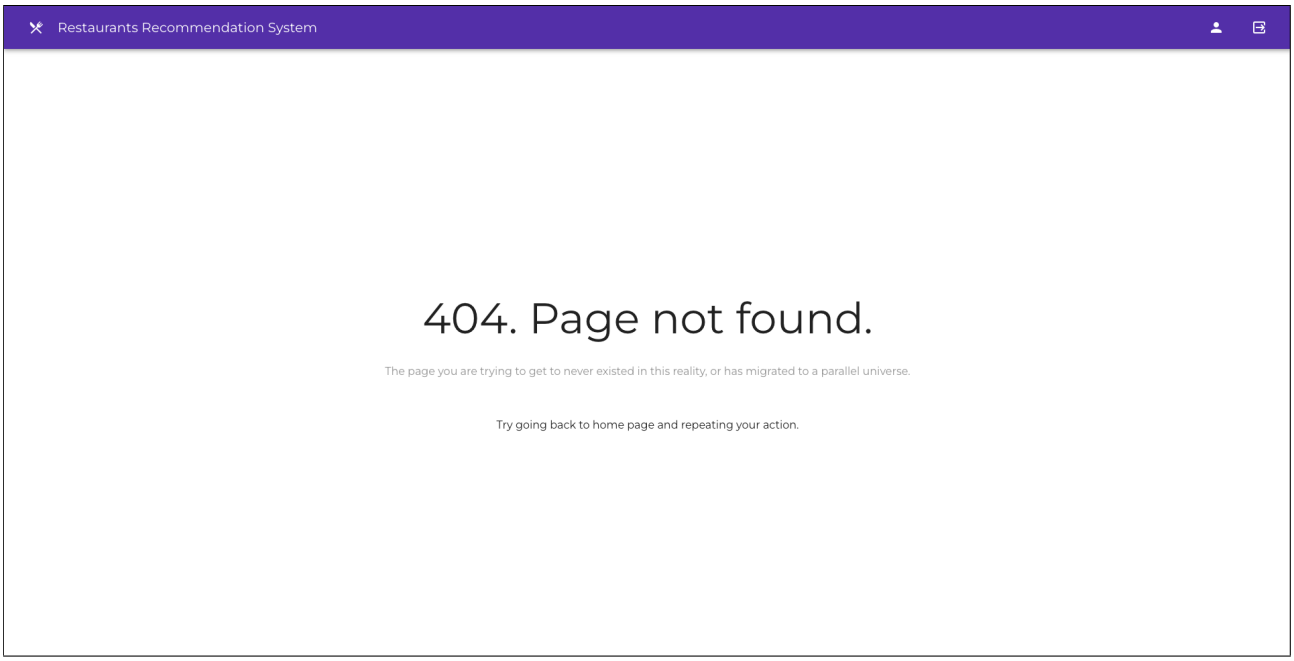


Figura 3.18: Vista de página no encontrada

Vuex

Para poder gestionar algunos aspectos fundamentales del aplicativo, se han implementado cuatro módulos:

- *Auth:* Este módulo almacena el usuario activo una vez ha iniciado sesión. Adicionalmente, se ha añadido en él un *plugin*[23] para poder almacenar este estado en el almacenamiento de la ventana. Esto implica que si el usuario refresca la ventana, el usuario seguirá con la sesión iniciada, a no ser que el usuario la finalice o cierre la pestaña.
- *Map:* Almacena el mapa principal para que se pueda acceder a él y a los puntos seleccionados para cada comida desde diferentes componentes de la aplicación.
- *Meals:* Guarda la información y preferencias para cada comida que va seleccionando el usuario a lo largo de la planificación gastronómica.
- *Places:* Este módulo almacena y gestiona los resultados de la barra de búsqueda del mapa.

Vue Router

Las únicas configuraciones adicionales que han recibido las rutas son:

1. Todo usuario que no esté registrado, es decir, cuya información no esté cargada en el módulo *Auth* de *Vuex*, será redirigido a la pantalla de inicio.
2. Toda ruta a la que se intente acceder que no esté definida, será redirigida a la ruta de error *(figura 3.18)*.

3.2.5. Despliegue

El aplicativo puede ser desplegado de forma local gracias a *Docker*[24]. Docker es un proyecto de código abierto que permite ejecutar aplicaciones dentro de unos contenedores donde se especifican previamente las dependencias con el fin de que, independientemente de en que máquina se ejecute, la configuración sea la misma. El proceso de creación de un contenedor es:

1. Se crea un Dockerfile donde se especifican las dependencias y la configuración del proyecto.
2. Se crea una imagen Docker que contiene tanto el código fuente como las librerías.
3. Se crea el contenedor donde se está ejecutando la aplicación.

Otra alternativa para cubrir estas necesidades es *RKT*[25]. Sus características son similares a las de Docker, no obstante, no se ha utilizado esta tecnología porque Docker ofrece una integración más sencilla, además de tener una comunidad mayor, por lo que la documentación es más amplia.

El despliegue tiene tres contenedores: uno para el *Frontend*, otro para el *Backend* y un último para *MongoDB*. Para poder ejecutarlos todos juntos, se debe especificar en un fichero de *Docker Compose*.

[23] *Plugin*: Complementos que añaden funcionalidades extra o mejoras a los programas.
[24] *Docker*: `https://www.docker.com/`
[25] *RKT*: `https://www.redhat.com/en/topics/containers/what-is-rkt`

Crear cada una de las imágenes cada vez que se realizan cambios puede ser una tarea piadosa. Es por ello que se ha automatizado esta tarea con la ayuda de *Github Actions*[26]. Esta es una plataforma de integración y despliegue continuo para poder automatizar la compilación, las pruebas y/o el despliegue.

El flujo de trabajo implementado es el siguiente *(figura 3.19)*.

1. Cuando se empujan cambios al repositorio, se inicia el flujo.
2. Se crean las imágenes para los proyectos del *Frontend* y/o *Backend*, según donde se hayan detectado cambios.
3. Se publican en *Docker Hub*[27], un repositorio público para distribuir imágenes Docker.
4. Desde el fichero de configuración del *Docker Compose*, se descargan las imágenes del *Frontend*, del *Backend* y la oficial de *Mongo*, para después poder ser ejecutadas en conjunto.

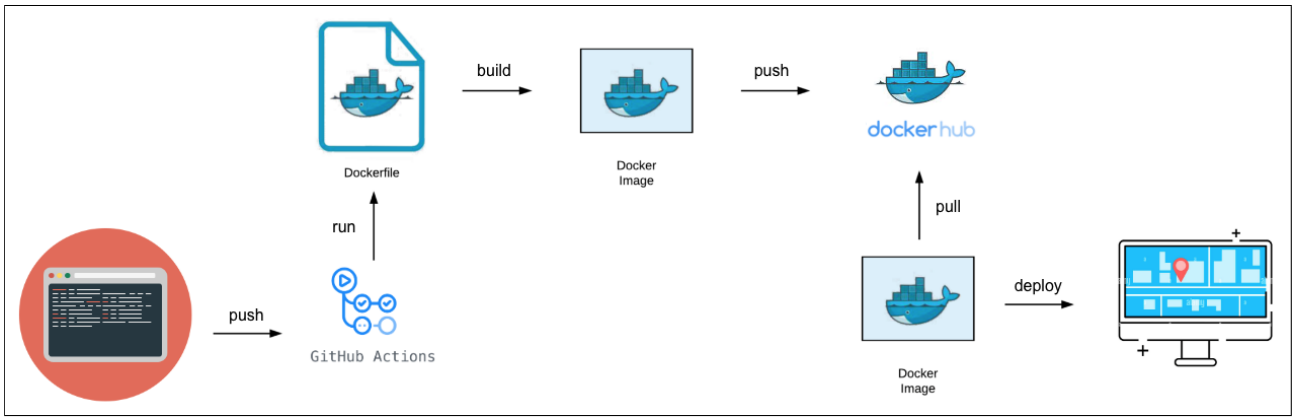


Figura 3.19: Diagrama de relación entre Docker y aplicativo

[26] *Github Actions*: `https://github.com/features/actions`
[27] *Docker Hub*: `https://hub.docker.com/`

Capítulo 4

Experimentación

Para comprobar el correcto funcionamiento del algoritmo recomendador, se ha puesto en evidencia diferentes escenarios donde para un mismo usuario con unos gustos marcados por sus reseñas realizadas, se crean diferentes recomendaciones en diferentes zonas. Debido a que el algoritmo solo va a recomendar restaurantes que estén dentro del área seleccionada, se ha planteado ver el comportamiento del algoritmo en tres diferentes zonas:

- Una zona con pocos restaurantes.
- Una zona con una cantidad suficiente de restaurantes.
- Una zona con muchos restaurantes.

El usuario creado ha reseñado 13 restaurantes donde muestra claramente su desinterés por los pescados, mariscos y los servicios lentos. Por otro lado, se muestra un gran interés por la cocina italiana, la hindú, la española, la japonesa, la china, la latina y los restaurantes de tapas, por lo que se espera que los restaurantes recomendados tengan alguna de esas especialidades. Para poder considerar el mayor número posible de restaurantes para cada zona, no se ha especificado ninguna categoría para pasar al algoritmo como dato de entrada y se han considerado restaurantes de cualquier nivel adquisitivo. Cabe destacar que estos datos son cogidos de una muestra y que los resultados pueden variar según la zona seleccionada y el perfil creado.

El cálculo de la similitud para cada una de las zonas se obtiene, según el capítulo 3, siguiendo una serie de pasos:

1. Se obtienen los restaurantes a partir de la *API REST* implementada que se encuentren en la zona especificada.
2. Procesar y tokenizar las categorías y reseñas de los restaurantes. A partir de ahí, se crea un vocabulario de palabras únicas aparecidas.
3. Crear una matriz de frecuencias con las palabras tokenizadas para cada restaurante de la zona.
4. Con ayuda del cálculo de la similitud de coseno, descrito en 1.1.1, se calcula cuán similar son los restaurantes que han sido valorados positivamente por el usuario al que se le va a hacer la recomendación con respecto a los disponibles en la zona.
5. Se obtienen los restaurantes con mayor similitud y se retornan al usuario.

Los resultados obtenidos son los siguientes:

Para la zona con pocos restaurantes (cuatro restaurantes) *(figura 4.1)*, los resultados generalmente no son muy buenos, puesto que no existen muchas opciones y si las características de los mismos no coinciden con los valorados por el usuario, la similitud entre ellos va a ser baja. Para este caso en concreto, los primeros tres resultados son los siguientes:

1. Restaurante de pizza y hamburguesas con valoración media de 4 estrellas y una única reseña. Similitud igual a *0,2236*.
2. Restaurante de comida española con valoración media de 4 estrellas y sin reseñas. Similitud igual a *0,1961*.
3. Restaurante italiano con valoración media de 4.5 estrellas y una única reseña. Similitud igual a *0,1944*.

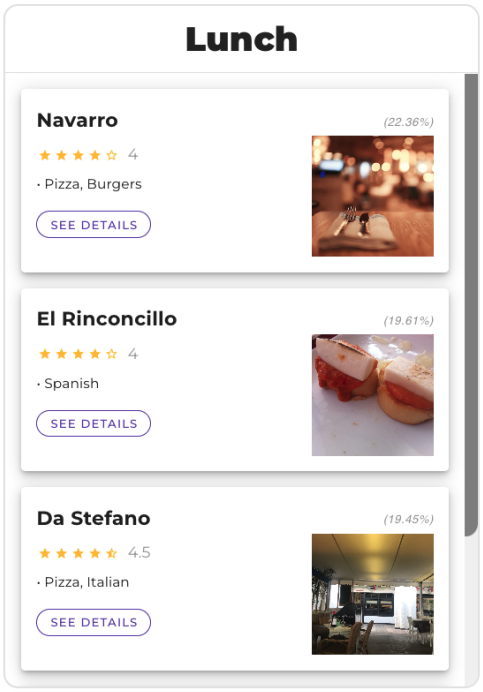


Figura 4.1: Recomendación generada en una zona con densidad baja de restaurantes

Aunque los tipos de restaurantes recomendados coinciden con los gustos del usuario, la similitud no es muy alta porque son restaurantes con máximo una única reseña que probablemente no contenga los gustos descritos por el usuario en su perfil.

Con respecto a las recomendaciones en la zona con una densidad de restaurantes media (ochenta y ocho restaurantes) *(figura 4.2)*, los resultados han mejorado. En este caso, los tres primeros resultados muestran lo siguiente:

1. Restaurante italiano con valoración media de 4 estrellas y sin reseñas. Similitud igual a *0,4472*.
2. Restaurante japonés con valoración media de 4 estrellas y sin reseñas. Similitud igual a *0,4472*.

3. Restaurante japonés y de sushi con valoración media de 5 estrellas y una única reseña. Similitud igual a *0,3380*.

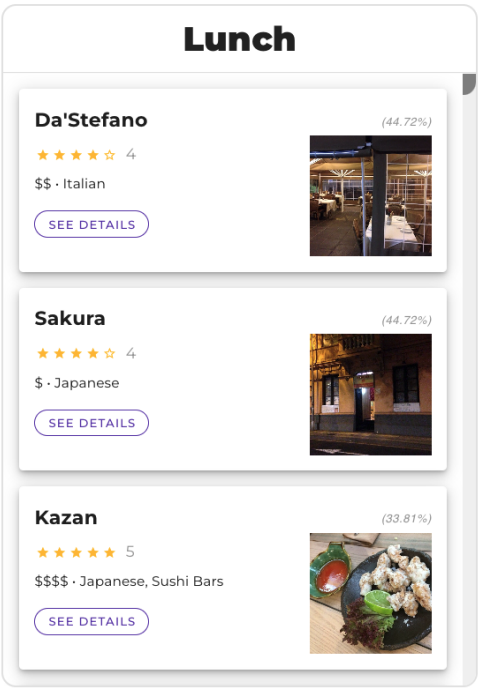


Figura 4.2: Recomendación generada en una zona con densidad media de restaurantes

El último escenario a estudiar es el de la zona con una densidad de restaurantes alta (doscientos setenta restaurantes) *(figura 4.3)*. En este caso, los tres primeros resultados muestran lo siguiente:

1. Restaurante japonés y de sushi con valoración media de 4.5 estrellas y sin reseñas. Similitud igual a *0,6324*.

2. Restaurante japonés con valoración media de 4 estrellas y sin reseñas. Similitud igual a *0,6324*.

3. Restaurante italiano con valoración media de 4 estrellas y una única reseña. Similitud igual a *0,4743*.

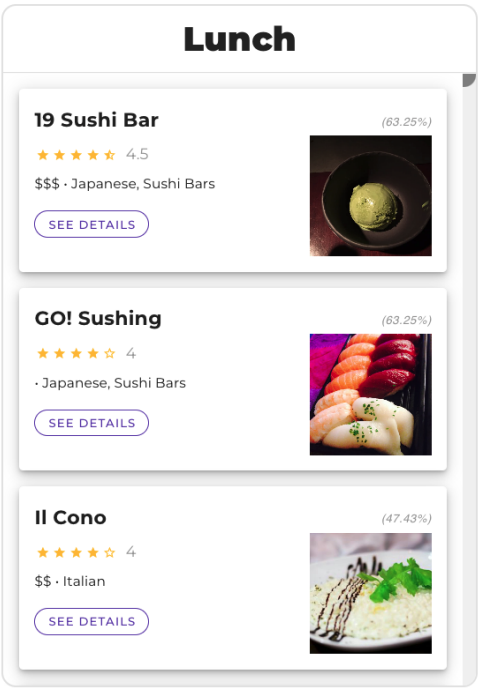


Figura 4.3: Recomendación generada en una zona con densidad alta de restaurantes

Como se puede observar en la siguiente gráfica *(figura 4.4)*, la similitud más alta de la zona ha ido incrementando junto con la densidad. Esto tiene sentido, ya que cuanto mayor es la densidad, más opciones hay para recomendar. Por otro lado, se puede observar como los restaurantes con mayor similitud no tienen apenas reseñas de otros usuarios. Esto se debe a que el texto de las reseñas de los restaurantes pueden no estar siendo tan representativos como se esperaba. No obstante, recoger las reseñas del usuario es necesario para poder saber que tipo de restaurantes son de interés para el mismo.

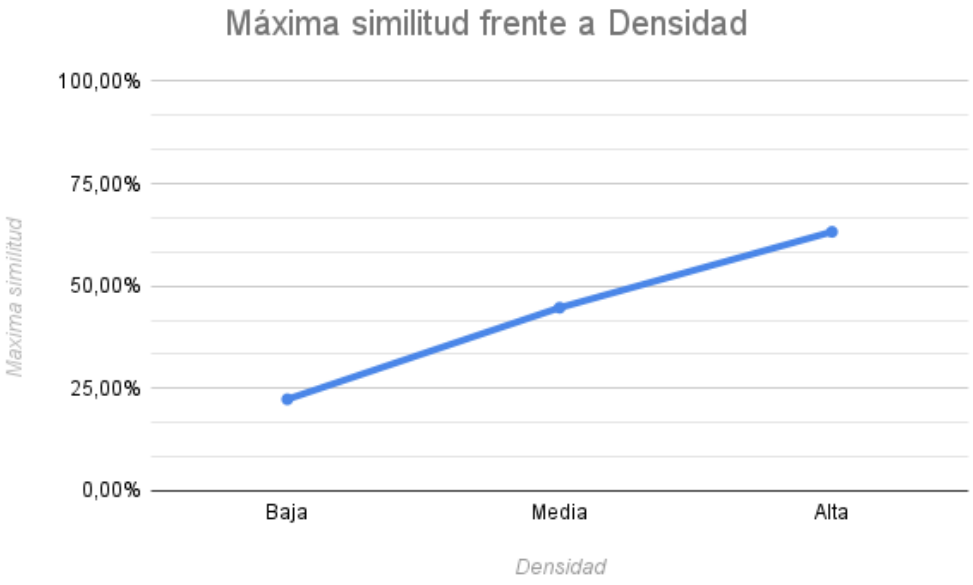


Figura 4.4: Gráfica para la experimentación del algoritmo recomendador

Capítulo 5

Presupuesto

El presupuesto para lo implementado en este Trabajo de Fin de Grado ha sido calculado dividiendo el trabajo en conceptos, especificando las horas empleadas para cada uno de ellos y un precio por hora fijo. El resultado del mismo puede ser consultado a continuación *(tabla 5.1)*.

Concepto	**Coste por hora**	**Horas**	**Coste total**
Estudio de las APIs	10€	10	100€
Estudio de las tecnologías a utilizar	10€	20	200€
Creación de la estructura del proyecto	20€	30	600€
Conexión entre los componentes	20€	10	200€
Creación de la interfaz de usuario	20€	120	2400€
Creación del algoritmo recomendador	25€	60	1500€
Mejoras y revisión del aplicativo	20€	10	200€
Redacción de la memoria	15€	60	900€
	Total	*320*	*6100€*

Tabla 5.1: Presupuesto del trabajo realizado

Capítulo 6

Conclusiones

Como se ha podido ver en este Trabajo de Fin de Grado, los sistemas de recomendación son y seguirán siendo de gran utilidad en diferentes ámbitos. En especial, gracias a la implementación de este proyecto, se proporciona una alternativa para que las personas puedan ahorrar tiempo a la hora de planear sus itinerarios gastronómicos. Los aspectos que más dificultad me han supuesto, han sido, por un lado, la creación y conexión de todos los componentes del proyecto, ya que todas las tecnologías eran nuevas para mí e inicialmente no conocía sus posibilidades. Por otro lado, pensar en el diseño de la aplicación teniendo en cuenta la mejor experiencia de usuario posible tambien ha sido todo un reto. Además, aunque la API de Yelp Fusion está muy bien para comenzar con el desarrollo de una aplicación debido a que es gratuita, puede convertirse en un obstáculo porque cada vez tiene menos soporte y los datos se están quedando obsoletos. Otro aspecto a destacar, como se ha visto en el capítulo 4, es que los textos de las reseñas de los usuarios parecen no ser importantes a la hora de obtener una buena similitud, lo que nos lleva a plantear las siguientes líneas futuras:

- Ampliación del algoritmo de recomendación, haciendo este un sistema híbrido que, por ejemplo, colabore con otros perfiles de usuario para así poder obtener recomendaciones más precisas.
- Inclusión de otros factores que influyen en la decisión del usuario, como puede ser el transporte tanto público como privado hasta los restaurantes. Esto aportaría información que hace las recomendaciones más precisas, por lo que el resultado se adecuaría más al usuario.
- Integración de este proyecto con otros proyectos relacionados con los sistemas de recomendación turísticos, como pueden ser sistemas de recomendación y reserva de hospedaje y generadores de planes de viaje. De esta manera, el sistema se enriquecería de diferentes opciones para los consumidores, lo que mejorará la experiencia de usuario.
- Adición de otras fuentes de información, como pueden ser otras APIs o conjuntos de datos, o incluso poder añadir los datos desde un perfil de administrador.

Capítulo 7

Conclusions

As we have seen in this final degree project, recommendation systems are and will continue to be very useful in different fields. In particular, thanks to the implementation of this project, an alternative has been provided so that people can save time when planning their gastronomic itineraries.

The most challenging aspects I had to deal with were, on one hand, the creation and connection of all the components of the project, considering that all the technologies were new to me and initially I didn't know their possibilities. On the other hand, thinking about the design of the application taking into account the best possible user experience has been difficult as well. Furthermore, although the Yelp Fusion API is very good to start with the development of an application because it is free, it can become an obstacle because of its decreasing support and therefore its data is becoming obsolete.

Another aspect to highlight, as seen in Chapter 4, is that the written users' reviews do not seem to be important when it comes to obtaining a good similarity, which leads us to propose the following future lines:

- Expansion of the recommendation algorithm, making it a hybrid system that, for example, collaborates with other user profiles in order to obtain more accurate recommendations.
- Inclusion of other factors that influence the user's decision, such as public and private transportation to restaurants. This would provide information that make the recommendations more accurate, resulting in a more accurate recommendation for the user.
- Integration of this system within other projects related to tourism recommendation systems, such as lodging recommendation and reservation systems and travel plan generators. In this way, the system would be enriched with different options for consumers, which will improve the user experience.
- Addition of other sources of information, such as other APIs or datasets, or even the ability to add data directly from an administrator profile.

Bibliografía

[1] James Bennett, Stan Lanning, et al. The netflix prize. In *Proceedings of KDD cup and workshop*, volume 2007, page 35. New York, 2007.

[2] Mitch Betts. Turning browsers into buyers. *MIT Sloan Management Review*, 42(2):8–8, 2001.

[3] Shannon Bradshaw, Kristina Chodorow, and Eoin Brazil. *MongoDB: The definitive guide: Powerful and scalable data storage*. O'Reilly Media, Inc., 2020.

[4] Juan C Brinez de Leon, Alejandro Restrepo Martinez, and Francisco E Lopez Giraldo. Similarity metrics applied to image analysis of photoelasticity. *Dyna*, 80(179):42–50, 2013.

[5] Laurent Candillier, Kris Jack, Françoise Fessant, and Frank Meyer. State-of-the-art recommender systems. In *Collaborative and Social Information Retrieval and Access: Techniques for Improved User Modeling*, pages 1–22. IGI Global, 2009.

[6] Wei-Ta Chu and Ya-Lun Tsai. A hybrid recommendation system considering visual information for predicting favorite restaurants. *World Wide Web*, 20(6):1313–1331, 2017.

[7] Moya Danilo, Tapia Liliana, Albán Mayra, and Rodríguez Gustavo. Un enfoque de machine learning en el desarrollo de sistema recomendadores para procesos de investigación. *Revista Ibérica de Sistemas e Tecnologias de Informação*, pages 816–827, 2020.

[8] Harvey M. Deitel and Paul J. Deitel. *Cómo programar en Java*. Pearson Educación, 2003.

[9] Julian Dibbelt, Charalampos Konstantopoulos, Dorothea Wagner, Damianos Gavalas, Spyros Kontogiannis, Christos Zaroliagis, Vlasios Kasapakis, and Grammati Pantziou. Multimodal route and tour planning in urban environments. In *2017 IEEE Symposium on Computers and Communications (ISCC)*, pages 214–219. IEEE, 2017.

[10] Faried Effendy, Barry Nuqoba, et al. Culinary recommendation application based on user preferences using fuzzy topsis. *IIUM Engineering Journal*, 20(2):163–175, 2019.

[11] Blaine T. Garfolo. *JavaScript*, pages 715–735. Elsevier, 2003.

[12] David Goldberg, David Nichols, Brian M. Oki, and Douglas Terry. Using collaborative filtering to weave an information tapestry. *Communications of the ACM*, 35(12):61–70, dec 1992.

[13] Oscar Escamilla González and Sergio Marcellin Jacques. Estado del arte en los sistemas de recomendación. *Res. Comput. Sci.*, 135:25–40, 2017.

[14] Yanette Díaz González and Yenisleidy Fernández Romero. Patrón modelo-vista-controlador. *Telemática*, 11(1):47–57, 2012.

[15] Pablo Granatiero. A restaurant recommender system for a new-born app-based gastronomic guide. Master's thesis, Universitat de Barcelona, 2021.

[16] Chen-Shie HO and Yu-Mei CHANG. Design and implementation of intelligent personalized dietary meal recommendation system. In *Proceedings of the International Conference on CCME*, pages 137–140, 2018.

[17] Marin Kaluža, Marijana Kalanj, and Bernard Vukelić. A comparison of back-end frameworks for web application development. *Zbornik veleučilišta u rijeci*, 7(1):317–332, 2019.

[18] Marin Kaluža, Krešimir Troskot, and Bernard Vukelić. Comparison of front-end frameworks for web applications development. *Zbornik Veleučilišta u Rijeci*, 6(1):261–282, 2018.

[19] K KesavaDasu, K Raj Kamal, and G Pranith. A cuisine based recommender system using k-nn and mapreduce approach. *International Journal of Recent Advances in Multidisciplinary Research*, 1(1):9–17, 2021.

[20] Li Li and Wu Chou. Design and describe rest api without violating rest: A petri net based approach. In *2011 IEEE International Conference on Web Services*, pages 508–515, 2011.

[21] Luis Martinez, Rosa M Rodriguez, and Macarena Espinilla. Reja: a georeferenced hybrid recommender system for restaurants. In *2009 IEEE/WIC/ACM International Joint Conference on Web Intelligence and Intelligent Agent Technology*, volume 3, pages 187–190. IEEE, 2009.

[22] Aimilia-Myriam Michail and Damianos Gavalas. Bucketfood: A crowdsourcing platform for promoting gastronomic tourism. In *2019 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops)*, pages 9–14. IEEE, 2019.

[23] Fabiola Ocampo-Botello, Federico Felipe-Durán, and Roberto de Luna-Caballero. Sistema de recomendación para el comercio electrónico aplicado a una tienda de libros. *Científica*, 18(2):55–62, 2014.

[24] Felipe Oliva Encabo et al. Desarrollo de una aplicación web con spring boot, vaadin y microservicios. Master's thesis, Universidad de Alcalá, 2021.

[25] Inga Karim Paz. Media aritmética simple. *Boletín electrónico*, 7:1–13, 2007.

[26] B Joseph Pine, Bart Victor, and Andrew C Boynton. Making mass customization work. *Harvard business review*, 71(5):108–11, 1993.

[27] Dwi Arman Prasetya, Phong Thanh Nguyen, Rinat Faizullin, Iswanto Iswanto, and Edmond Febrinicko Armay. Resolving the shortest path problem using the haversine algorithm. *Journal of critical reviews*, 7(1):62–64, 2020.

[28] Oscar Arley Riveros, Juan Guillermo Romero, and Jhon Francined Herrera. Implementación de la técnica de los k-vecinos en un algoritmo recomendador para un sistema de compras utilizando nfc y android. *Inge Cuc*, 13(1):9–18, 2017.

[29] Ryosuke Saga, Yoshihiro Hayashi, and Hiroshi Tsuji. Hotel recommender system based on user's preference transition. In *2008 IEEE International Conference on Systems, Man and Cybernetics*, pages 2437–2442. IEEE, 2008.

[30] Laura Sebastia and Eliseo Marzal. Extensions of the tourist travel design problem for different travel styles. *Procedia Computer Science*, 176:339–348, 2020.

[31] Michael Stonebraker. Sql databases v. nosql databases. *Communications of the ACM*, 53(4):10–11, 2010.

[32] R. Pell T. L. Saaty, P. Rogers. Portfolio selection through hierarchies. *Journal of Portafolio Management*, 6(3):16–21, 1988.

[33] M. Tascón. *Big Data y el Internet de las cosas: Qué hay detrás y cómo nos va a cambiar*. Los Libros de La Catarata, 2020.

[34] Manolis G Vozalis and Konstantinos G Margaritis. Applying svd on item-based filtering. In *5th International Conference on Intelligent Systems Design and Applications (ISDA'05)*, pages 464–469. IEEE, 2005.

[35] Kai Zhang, Keqiang Wang, Xiaoling Wang, Cheqing Jin, and Aoying Zhou. Hotel recommendation based on user preference analysis. In *2015 31st IEEE International Conference on Data Engineering Workshops*, pages 134–138. IEEE, 2015.

[36] Feng Zhao, Fengwei Yan, Hai Jin, Laurence T Yang, and Chen Yu. Personalized mobile searching approach based on combining content-based filtering and collaborative filtering. *IEEE Systems Journal*, 11(1):324–332, 2015.