Proyecto de Sistemas Informáticos. Sistema de Recomendación de Actividades Turísticas: Madrid Live

MEMORIA DEL PROYECTO

Belén Agudo Calvo
Jessica del Valle Corral
Jose Luis Jorro Aragoneses

———————

Dirigido por: Belén Díaz Agudo
Codirigido por: Juan A. Recio García

Departamento de Ingeniería del Software e Inteligencia Artificial
Facultad de Informática
Universidad Complutense de Madrid

Junio 2013

Documento maquetado con TEXIS v.1.0+.

Proyecto de Sistemas Informáticos. Sistema de Recomendación de Actividades Turísticas: Madrid Live

Memoría de Proyecto de Sistemas Informáticos

Ingeniería del Software e Inteligencia Artificial

Departamento de Ingeniería del Software e Inteligencia Artificial

Facultad de Informática

Universidad Complutense de Madrid

Junio 2013

Copyright © Belén Agudo Calvo, Jessica del Valle Corral y Jose Luis Jorro Aragonenes

Autorización

Belén Agudo Calvo, Jessica del Valle Corral y Jose Luis Jorro Aragoneses, alumnos matriculados en la asignatura de Sistemas Informáticos, autorizan, mediante el presente documento, a la Universidad Complutense de Madrid (UCM), a difundir y utilizar con fines académicos, no comerciales, y mencionando expresamente a sus autores, tanto la propia memoria,como el código, la documentación y/o el prototipo desarrollado. Todo ello realizado durante el curso académico 2012-2013 bajo la dirección de Belén Díaz Agudo y Juan Antonio Recio, profesores del Departamento de Ingeniería del Software e Inteligencia Artificial.

Belén Agudo Jessica del Valle Jose Luis Jorro

Agradecimientos

Queremos aprovechar estas líneas para agradecer el cariño de todas aquellas personas que nos han ayudado y apoyado a lo largo de todos estos años de carrera.

En primer lugar, queremos dar nuestro más sincero agradecimiento a Belén Díaz Agudo y Juan Antonio Recio, por habernos dirigido en nuestro proyecto de fin de carrera. Gracias por todas las atenciones que hemos recibido por vuestra parte y por el tiempo que habéis perdido con nosotros, resolviendo cada duda o problema siempre con una sonrisa. Pero sobre todo gracias por la confianza que habéis depositado en nosotros.

También queremos agradecer a nuestros compañeros de clase y amigos que hayan estado con nosotros en los momentos más duros de la carrera, motivándonos a hacerlo lo mejor posible y ayudándonos cuando estábamos perdidos. Gracias por hacer de estos duros años algo inolvidable. Además, queremos agradecer a todos los usuarios que han participado en la fase de pruebas, ya que sin ellos no podriamos haber mejorado la aplicación.

No queremos pasar por alto la oportunidad de agradecer a todos los profesores que hemos tenido durante nuestra vida académica, no sólo en esta facultad, sino también desde pequeños, porque entre todos han formado la base para que hoy podamos ser lo que somos.

Y por último, el agradecimiento más importante, aquel que va dedicado a nuestras familias y parejas. En especial, a nuestros padres por ser las personas con las que hemos contado, contamos y contaremos incondicionalmente durante toda nuestra vida. Gracias por estar siempre a nuestro lado, apoyándonos y aconsejándonos.

A todos vosotros, que habéis sufrido con nosotros cada fracaso y celebrado cada logro, por pequeño que fuese, queremos dedicaros nuestro proyecto final de carrera.

Resumen

Hoy en día en el mundo del comercio online casi todos los servicios Web están provistos de un sistema recomendador que guía al usuario en la búsqueda del producto adecuado para él. En concreto el sector turístico se ha beneficiado de estos progresos tecnológicos e Internet proporciona numerosas páginas Web informando de distintas posibilidades de visitas por distintas ciudades españolas. Sólo en Madrid, en Agosto de 2012 recibimos 581.920 viajeros que se alojaron en los establecimientos hoteleros de la ciudad[1]. Madrid ofrece una gran variedad de posibilidades para sus visitantes lo que si se dispone de poco tiempo hace muy útil cualquier recomendación.

El objetivo de este proyecto, denominado Madrid Live, es el diseño y desarrollo de un sistema de recomendación de actividades de ocio en Madrid para individuos o grupos. La meta de este sistema es ayudar a resolver la sobrecarga de información que hay en la Web sobre actividades de ocio en Madrid y facilitar la búsqueda y decisión a sus usuarios.

Para este proyecto hemos implementado un prototipo con cuatro tipos de actividades que son museos, restaurantes, parques y paseos. Pero Madrid Live no está limitado a estas actividades, ya que es muy extensible y cuenta con un recomendador genérico de planes donde cada actividad se elige con otros recomendadores específicos. La aplicación también tiene un catálogo dinámico de actividades siempre actualizado. Esto se debe a la conexión del sistema en tiempo real con distintas APIs de las cuales se descargan la información necesaria.

Madrid Live tiene como finalidad encontrar un plan, combinando los diferentes tipos de actividades, adecuado a las restricciones horarias y a las preferencias del usuario. Para ello hemos utilizado distintas técnicas de recomendación. De esta forma diseñamos planes de una manera más fácil de decidir para los usuarios y que resulten de su interés. Por último, destacar el componente social del proyecto, ya que tiene la posibilidad de recomendar

[1]Datos obtenidos del Área de Gobierno de Economía y Participación Ciudadana de Madrid, observatorio económico, http://www.esmadrid.com

planes a un grupo de usuarios respetando los gustos de cada uno para que todos estén satisfechos con el plan obtenido. La aplicación está disponible via Web[2] para ser utilizada.

Palabras clave : Sistemas de recomendación, razonamiento basado en casos, ontologías, turismo, filtrado colaborativo, funciones agregación.

[2]http://ssii13.fdi.ucm.es

Abstract

Nowadays inside the online commerce all web services are provided by a recommender system, which leads the user while searching the most suitable product for him. Specifically the tourist sector has been profiting of this technological progress and the Internet generally provides numerous web sites with information of various posibilities for visiting different Spanish cities. In our city, during August 2012, we have hosted 581.920 visitors which stayed in the hotels of the city[3].

The main goal of this Final Project called Madrid Live, has been the design and development of a recommender system of leisure activities for individuals or groups in Madrid. The target of this system is to cope with the abundance of information about activities that exists on the Internet in Madrid. Furthermore it provides an easy way of searching and deciding for the users.

The system is able to recommend four kinds of activities such as museums, restaurants, parks and touristic walks. Madrid Live is really expandable and not limited only to those activities because it works with a general recommender of plans for the time schedule and with specific recommenders, which choose each activity independently. In addition, the application also has a dynamic list of activities always updated. That is caused by the connection of the system in real time with different APIs from which it downloads the necessary information.

The objective of Madrid Live is to find a plan, combining the different type of activities, suitable for the time restrictions and the preferences of the user. For that propose we have used reasoning techniques based on cases, "Case Based Reasoning". In this way we are designing plans to simplify the decision process of the users, which considers simultaneously their interests. Finally we have to highlight the social factor of the project, which provides the possibility of recomemnd plans for a group of people respecting the pre-

[3]Data acquired from the government of economy of Madrid, http://www.esmadrid.com

ferences of every person. The application is available on Internet[4] and ready to be used.

Key words: Recommender system, case-based reasoning, ontologies, tourism, collaborative filtering and aggregation functions.

[4] http://ssii13.fdi.ucm.es

Índice

Índice de figuras

Índice de Tablas

Capítulo 1

Motivación y Objetivos

En la ciudad de Madrid tenemos más de 50 museos, casi 60 monumentos turísticos, más de 10.000 bares y restaurantes, más de 50 salas de cine y teatros. Ésto nos da una idea de todas las actividades de ocio que ofrece la ciudad de Madrid a los numerosos visitantes que cada año llenan la ciudad. Por este motivo es necesario una herramienta que permita al usuario saber qué sitios le pueden interesar más. Madrid Live busca el objetivo de encontrar al usuario las actividades que mejor se adapten a sus preferencias.

Existen muchas páginas orientadas a mostrar la información turística de Madrid, por ejemplo, la guía del ocio[1], TripAdvisor, etc. Lo que hacen estas webs es mostrar formularios de búsqueda que dan como resultado listados con actividades que se ofrecen en Madrid, proporcionándole la información que necesita para comprar entradas, reservar una mesa o para desplazarse a ese lugar. Al usuario toda esa información puede no interesarle y dificultarle la búsqueda de una actividad atractiva para él, llevándole incluso a sentirse abrumado con tanta información y deje de buscar.

Por ello en Madrid Live se analizan las preferencias del usuario y se le filtran aquellos que le resulten más interesante. En un mundo en que el que el factor social está siendo fundamental, el paso siguiente que se quiere cubrir es que no solo una persona pueda obtener un plan acorde con sus gustos, sino que un grupo de personas pueda hacer un plan y que todos los usuarios de ese grupo queden satisfechos con las actividades que han realizado. Esta es la novedad que aporta nuestro prototipo con respecto a las anteriores páginas Web.

Por último, indicar que Madrid Live es la base de un proyecto que puede evolucionar incluyendo más actividades (la aplicación está diseñada para facilitar la incorporación de otros tipos de actividades), extendiéndolo a otras

[1] http://www.guiadelocio.com

Figura 1.1: Logo de la aplicación

ciudades e incluso sincronizándolo con alguna red social para que la recomendación a grupos de usuarios pueda ser más efectiva.

1.1. Objetivos

Nuestro principal objetivo es diseñar una aplicación útil y lo más versátil posible para planear visitas por Madrid. La aplicación está orientada a que turistas o ciudadanos de Madrid puedan hacer un plan de ocio en la ciudad de forma sencilla. Para ello no nos centramos en un solo tipo de plan y se posibilita la inclusión de actividades de distinta índole para poder crear planes variados. También se pretende que las recomendaciones sean lo más ajustadas a las preferencias de los usuarios, por ello se ha hecho especial hincapié en cómo recoger esta información y guardarla en un perfil de usuario para poder ajustar las recomendaciones que se le hagan.

Para Madrid Live se presentan las siguientes metas:

- Sugerir planes con actividades que satisfagan lo más posible las preferencias del usuario.
- Evitar repeticiones en las recomendaciones a un usuario llevando un histórico actualizado de las actividades ya propuestas.
- Realizar recomendaciones grupales intentando satisfacer a todos los miembros del grupo.
- Recoger las preferencias del usuario de la manera más sencilla intentando evitar formularios inmensos de preguntas.

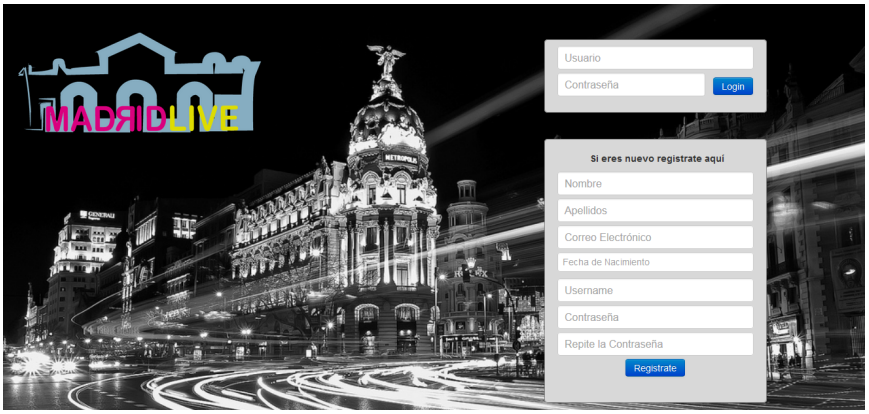

Figura 1.2: Interfaz Web del login.

- Utilizar distintas maneras de realizar los test de preferencias para que sea más dinámica la recogida de información.
- Incluir un sistema de valoración para recoger la satisfacción del usuario a la recomendación sugerida.
- Inclusión de un catálogo de actividades que se recogen de forma online, teniendo la información de las actividades siempre actualizada.
- Interfaz Web intuitiva y fácil de usar.
- Crear un arquitectura en capas que separe la lógica de la interfaz, permitiendo que otros dispositivos como móviles o tablets tener sus propias interfaces.
- Llevar a cabo un diseño modular para facilitar la integración de nuevos tipos de actividades en el sistema.
- Facilitar la inclusión de nuevos tipos de actividades, por ejemplo cines, con el uso de plantillas abstractas que usan los tipos de actividades para diseñar los planes.

1.2. Estructura de la memoria

El contenido de esta memoria está organizado de la siguiente manera:

1. En este primer Capítulo se ha expuesto la motivación y objetivos que se tienen para desarrollar un tipo de proyecto como éste.
2. En el Capítulo 2 se describe el estado del arte de los sistemas de recomendación para individuos y para grupos explicando sus tres variantes:

sistemas de recomendación basados en contenido, ricos en conocimiento y basados en la opinión de otros usuarios (colaborativos). En este capítulo también se describen las tecnologías usadas para desarrollar la implementación de este proyecto.

3. A continuación, en el Capítulo 3, se presenta de manera más concreta cómo se ha desarrollado nuestro sistema hablando sobre el marco teórico, las fuentes de conocimiento de donde se obtiene la información y la implementación del sistema comentando su arquitectura.

4. El Capítulo 4 se enumeran las conclusiones a las que se ha llegado tras realizar el trabajo, así como las líneas de trabajo futuro que puede tener nuestro proyecto.

5. Para finalizar se incorporan varios apéndices donde se encontrará información adicional sobre nuestro proyecto.

Capítulo 2

Estado del Arte

2.1. Sistemas Recomendadores

Como se muestra en la Figura 2.1 [1] el crecimiento acelerado de Internet en los últimos años, ha dado lugar a una cantidad inmensa de datos sobre cualquier tema en la red. Todo esto, origina una importante sobrecarga de información hacia los usuarios finales, quienes necesitan demasiado tiempo para hacer una búsqueda y a pesar de ello, en la mayoría de las ocasiones, los resultados que obtienen son poco exitosos.

Con el fin de dar un paso adelante en el contexto de la recuperación de información nacen los *sistemas de recomendación*Adomavicius y Tuzhilin (2005). Estos sistemas consisten en un tipo específico de filtro de información cuyo propósito es el de ayudar al usuario a seleccionar elementos de entre una gran cantidad de opciones. Actúan sugiriendo elementos que resulten de interés para los usuarios.

Una de las páginas web que utilizan sistemas de recomendación y además esta dentro de nuestro dominio, las actividades en una ciudad, es TripAdvisor. Como se puede ver en la Figura[2] 2.2 esta página web no sólo te da la posibilidad de buscar actividades de un tipo en una ciudad, sino que además le sugiere las actividades y los hoteles más valoradas por el usuarios registrados. Esto significa que utiliza técnicas de filtrado colaborativo. Consiste en que las personas registradas en la aplicación, que han consultado previamente esta web, escriben su opinión sobre los sitios y los valoran poniendo una puntuación. TripAdvisor utiliza esta información para posteriormente mostrar a futuros usuarios los sitios más populares y asegurar que las actividades sugeridas en la página principal, por lo general, son del agrado de la

[1] "Usuarios de Internet por cada 100 habitantes entre 2001 y 2011", International Telecommunications Union, Geneva. Dato obtenido el 4 de Abril de 2012

[2] http://www.tripadvisor.es/Attractions-g187514-Activities-c25-Madrid.html

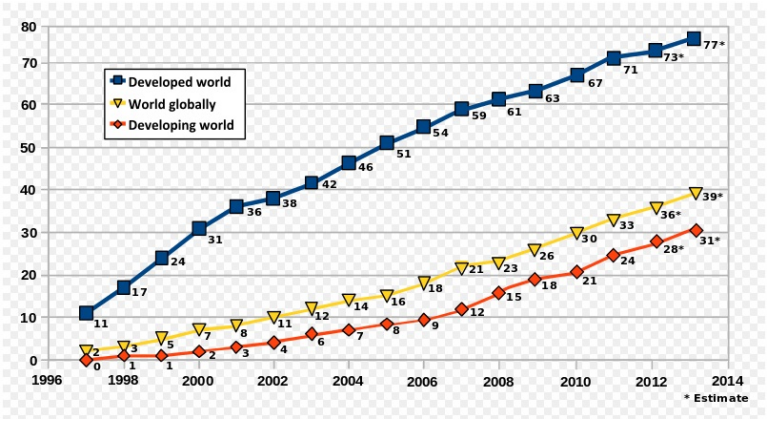

Figura 2.1: Crecimiento de Internet en los últimos años.

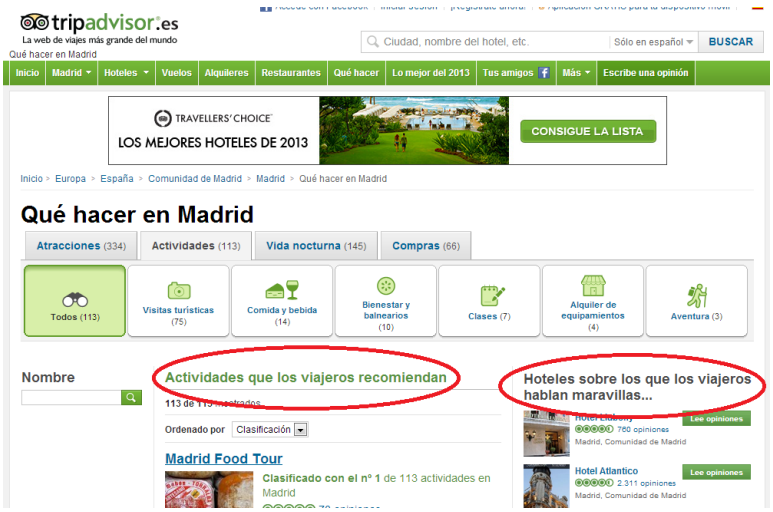

Figura 2.2: Página principal de tripAdvisor.

gente. Otro ejemplo de servicio web que incluye sistemas de recomendadores colaborativo es Amazon. Cuando un usuario consulta un producto de una categoría en concreto, el sistema muestra artículos que otros clientes que vieron este producto también compraron.

La conclusión es que estos sistemas comparan los productos en base al perfil del usuario o a la consulta concreta realizada y ofrecen más opciones similares para dar variedad en la recomendación y así tener más éxito.

Estos intereses del usuario de los que hablábamos antes, pueden ser introducidos de forma explícita o implícita.

- **Explícita**: cuando el sistema obtiene las preferencias de forma explícita, ofrece al usuario la posibilidad de indicar sus gustos mediante, por

ejemplo, un formulario.

- **Implícita**: el sistema obtiene información a través de la interacción del usuario con el propio sistema sin que éste lo note.

Ejemplos de recolección de preferencias del usuario:

- Que el usuario califique algún tema en concreto.
- Que ordene un conjunto de temas de una lista.
- Presentarle dos temas y solicitarle que seleccione el que más le guste.

De forma implícita:

- Guardar los temas que el usuario ha seleccionado o visualizado.
- Contabilizar el número de visitas que tiene un producto concreto.
- Analizar las redes sociales del usuario para conocer sus gustos.

A continuación, explicaremos en mayor detalle en qué consisten estos sistemas de recomendación (Figura 2.3), qué tipos hay, sus diferencias y algunas de las herramientas que nos permiten implementarlos.

2.1.1. Recomendadores Individuales

Los recomendadores individuales, sugieren elementos (o productos) que resulten de interés para un único usuario. Existen dos formas muy distintas de recomendar un producto a un único individuo, basándonos en su perfil de usuario y/o en sus preferencias actuales o en su entorno social. Según la procedencia de estas características, podemos distinguir dos grandes grupos de sistemas de recomendación:

1. Si la recomendación se basa en la comparación de las características del producto con los gustos del usuario: Sistemas de Recomendación por aproximación *basada en contenido*.
2. Si la recomendación se basa en las valoraciones de los usuarios: Sistemas de Recomendación *basada en filtrado colaborativo*.

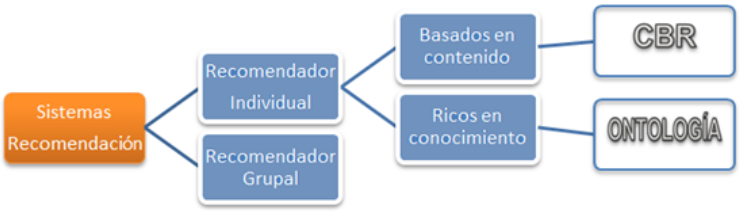

Figura 2.3: Sistemas Recomendadores

2.1.1.1. Sistemas de recomendación basados en contenido

Los sistemas recomendadores basados en contenido utilizan una base de elementos con descripción (características) y un perfil con asignaciones de importancia a estas características. Estas descripciones se utilizan para emparejar aquellos elementos que mejor cumplan las preferencias del usuario. En otras palabras, su objetivo no es otro que el de encontrar productos similares a aquellos que al usuario le han gustado en función del contenido del mismo.

Los sistemas de recomendación basados en contenido tienen un amplio paralelismo con el razonamiento basado en casos (CBR, del inglés Case Based Reasoning)(Pazzani y Billsus (2007)), uno de los paradigmas de resolución de problemas más exitosos y utilizados dentro de la Inteligencia Artificial. En este apartado hablaremos tanto de los Sistemas Basados en Conocimiento como del ciclo CBR.

Podemos definir un *Sistema Basado en Conocimiento* (SBC) como un sistema informático que utiliza conocimiento sobre un dominio específico de aplicación para obtener una solución a un problema de este dominio. En nuestro proyecto, un claro ejemplo de dominio para SBC es el de los distintos museos de Madrid (Sección 4.1.3.3).

Por otro lado, los *Recomendadores Basados en Casos*, cuentan con una base de casos compuesta por problemas resueltos anteriormente junto con la solución que se tomó en ese momento para los mismos. De este modo los nuevos problemas que vayan surgiendo, se irán resolviendo a través de la adaptación de las soluciones pasadas (las cuales fueron capaces de resolver problemas similares al actual).

Ciclo CBR El razonamiento basado en casos se explica mediante el ciclo CBR (Figura 2.4). Este ciclo consta de 4 fases muy diferenciadas (conocidas como las 4 Rs) que explicaremos a continuación:

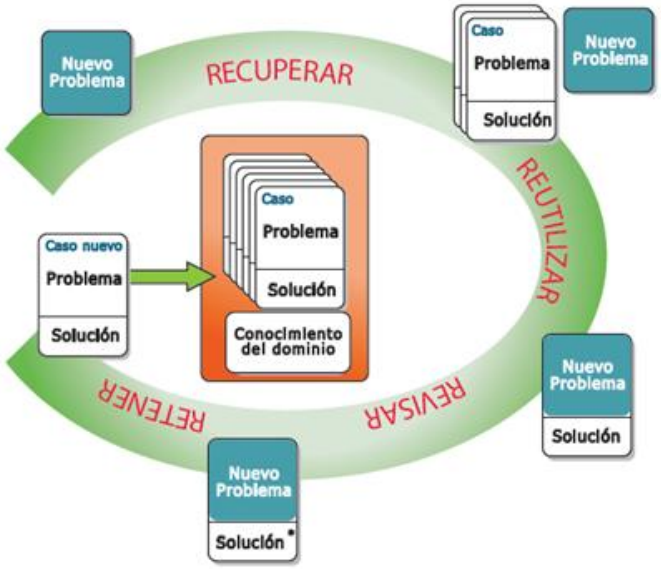

Figura 2.4: Ciclo CBR

1. **Recuperar**: en esta fase recuperamos los casos más relevantes (es decir los que presentan una mayor similitud) para una consulta dada.
2. **Reutilizar**: durante esta fase, extraemos la solución del caso seleccionado para utilizarla.
3. **Revisar**: debemos revisar dicha solución y modificar parte de ésta con el fin de que se ajuste mejor al caso que queremos resolver.
4. **Recordar**: por último, después de haber aplicado la solución con éxito, es importante recordar la solución final para añadirla, si aporta algo nuevo, como un nuevo caso a la base de casos.

Para entenderlo mejor explicaremos un ejemplo de CBR implementado en Madrid Live. Cogeremos como ejemplo de CBR el Recomendador de Museos. Este recomendador tiene una base de casos compuesta por todos los museos que hay en la ciudad de Madrid. En la información que se guarda para cada museo tenemos los datos generales (dirección, página web, etc.) y los tipos de exposiciones que tienen dichos museos, por ejemplo, el Museo del Prado tiene exposiciones fijas de Pintura y Escultura, por lo que en la ficha del Museo del Prado Figurarán a 1 el tipo Pintura y Escultura, y 0 el resto de tipos, ya que estos tipos no se encuentran en dicho Museo.

A partir de esa base de casos tenemos que buscar un museo para un

determinado usuario. Para ver los tipos de museos que le gustan a los usuarios se les hace un test de preferencias (Sección 2.3). Entonces valoramos cada tipo de museo que le gusta al usuario con un valor entre 0 (no le gusta nada) y 1 (le gusta mucho). Imaginemos que el usuario tiene como valores 1 en Ciencias, y 0 en el resto de campos (esto no es exactamente así ya que los demás tipos de museos tendrán algún valor y puede que la máxima preferencia del usuario sea a más de un tipo de museo, pero sirve para simplificar el ejemplo). Entonces procedemos a hacer el ciclo CBR:

1. **Recuperar**: Insertamos la consulta con las preferencias del usuario y buscamos un museo cuyo valor en Ciencias sea 1 y el resto de campos sea 0. Una vez que el recomendador ha buscado los museos, devuelve una lista con los museos que más se adaptan a la solución, además nos devuelve un valor de satisfacción entre 0 y 1. Este valor indica cuánto de próximo es la solución con la consulta hecha.
2. **Reutilizar**: Encontramos que el primer museo que nos propone para el usuario es el Museo de Ciencias Naturales, en este caso el museo cumple con todo lo que hemos pedido en la consulta, por lo que su nivel de satisfacción es 1. Usaremos esa solución para mostrar al usuario.
3. **Revisar**: Al obtener ese museo, adaptamos el horario que se nos solicita a la solución, ya que cuando obtenemos la solución ésta carece de horario. Con esto ya tendríamos la solución completa para devolverla al usuario.
4. **Recordar**: En nuestro caso no recordamos las soluciones, ya que el conjunto de soluciones posibles lo tenemos ya incluido en nuestra base de casos, y no depende de la adaptación de una solución al usuario. Si, por ejemplo, guardáramos los casos con el horario, podríamos guardar la solución devuelta al usuario, ya que hemos modificado el horario. Así la próxima vez que tuviésemos un usuario similar y en ese mismo horario se le devolvería directamente el Museo de Ciencias Naturales, sin necesidad de adaptarla en el horario porque ya la tendríamos incluida en nuestra base de casos.

Otras características por las que se define un recomendador CBR es la forma en la que interactua con el usuario. Según esta característica podemos definir dos tipos:

- *Single-shot:* Esta forma muestra al usuario un conjunto de soluciones que se han recuperado, y éste elige el que prefiere de entre todos los mostrados y puede elegir las recomendaciones que no le

agradan. Se puede usar este método para basar el recomendador en un perfil de usuario, en el que modificaríamos las preferencias del perfil según sus elecciones. Otra opción sería basarla en recomendadores colaborativos, haciendo que usuarios que hayan seleccionado elementos parecidos obtengan como resultados elementos seleccionados por el.

- *Conversacional:* Aquí el sistema interactua más con el usuario para obtener el resultado final. Consiste en que el sistema le propone una recomendación al usuario, y éste transmite al sistema que opinión tiene sobre la recomendación, por ejemplo, puede seleccionar una de esas recomendaciones y obtener datos parecidos al que ha elegido. El sistema vuelve a hacer la consulta modificando las preferencias del usuario en su elección y vuelve a mostrar otras soluciones. Así se consigue refinar el perfil del usuario con las características comunes que tienen los elementos seleccionados por él.

2.1.1.2. Sistemas de recomendación ricos en conocimiento. Ontologías

Aunque son muchas las definiciones que podemos encontrar sobre el concepto de *Ontología*, una de las más aceptadas es la de Thomas Gruber (cita: Gruber (1993)), quien define una Ontología como una '*especificación formal y explícita de una conceptualización compartida*'. A continuación elaboraremos dicha definición.

- El término *conceptualización* implica que toda ontología desarrolla un modelo abstracto del dominio que representa (basado conceptos, atributos, valores y relaciones).

- Con *especificación explícita* indicamos que una ontología supone la descripción y representación de un dominio concreto mediante conceptos, atributos, valores, relaciones, funciones, etc., definidas explícitamente.

- El término *formal* implica que todas las representaciones deben expresarse en una ontología mediante un formalismo idéntico.

- Y por último, el término *compartida*. En efecto, una ontología lo es cuando dicha conceptualización y su representación formal y explícita ha sido favorablemente acogida por todos los usuarios dela misma.

Las ontologías recogen la información de un dominio de un modo genérico y proporciona un manera consensuada de representar el conocimiento. De esta manera puede ser reutilizada y compartida por distintos sistemas.

El objetivo de las ontologías es crear un vocabulario común que defina el significado de los conceptos y las relaciones entre ellos. En Madrid Live, hemos creado una ontología de Restaurantes (ver Apéndice A).

2.1.1.3. Filtrado Colaborativo

La abrumadora cantidad de datos presente en Internet, ha hecho necesario el desarrollo de mecanismos de filtrado de información. Una de las técnicas utilizadas por algunos sistemas recomendadores para hacer frente a este problema se denomina *filtrado colaborativo*.

Un sistema de recomendación colaborativo es aquel en el que las recomendaciones se realizan basándose en los términos de similitud entre los usuarios. Es decir, recomiendan elementos que son del gusto de otros usuarios de intereses similares (se basan en el concepto del boca a boca entre usuarios) en lugar de recomendar elementos similares a los que le gustaban a dicho usuario en un pasado, tal y como sucedía con los basados en contenido.

El flujo de trabajo de un sistema de filtrado colaborativo es el que se muestra a continuación:

1. Se recogen las valoraciones que cada usuario da a los distintos elementos (en nuestro caso, a las distintas actividades que ofrecemos: museos, restaurantes,...). Hay dos formas de recoger estas valoraciones:

   - *De forma explícita*: el usuario asigna una puntuación a cada elemento.
   - *De forma implícita*: el sistema extrae la información pertinente de las acciones del usuario (el tiempo que pasa leyendo una determinada página web, los enlaces que sigue, los elementos que selecciona,...).

2. El sistema compara las valoraciones de este usuario con la de todos los demás y encuentra las personas que tienen gustos similares a los suyos.

3. El sistema recomienda elementos que los usuarios similares han calificado "de su agrado", teniendo en cuenta que el usuario al que se le va a realizar la recomendación todavía no haya realizado esta actividad.

Podemos distinguir 2 tipos de algoritmos de filtrado colaborativo:

- *Algoritmos de filtrado colaborativo basados en memoria, o algoritmos de vecinos cercanos (Nearest Neighbour) (Jannach et al. (2011))*: Estos algoritmos utilizan la base de datos completa para generar una

predicción. Primeramente emplean técnicas estadísticas para encontrar a vecinos, es decir usuarios con un historial de valoraciones sobre los elementos similar al usuario actual y posteriormente se utilizan una serie de algoritmos que combinan las preferencias de esta vecindad para realizar las predicciones y recomendaciones.

- *Algoritmos de filtrado colaborativo basados en Modelo* (Galán Niet (2006), Albin Rodriguez (2009)): Estos algoritmos proporcionan recomendaciones de productos desarrollando primero un modelo de las puntuaciones de los usuarios sobre los ítems. No se utilizan técnicas estadísticas sino una aproximación probabilística que calcula el valor esperado de una predicción del usuario para cada ítem en función de los ratings anteriores. Es decir, calculan como de similar son estas puntuaciones con respecto al ítem activo con el fin de realizar una predicción para el mismo. Para ello se utilizan distintos algoritmos de aprendizaje Clustering o redes neuronales como las Redes de Funciones de Base Radial (RBFN) (ver Isasi Viñuela y Galvan Leon (2004)).

2.1.2. Recomendadores Grupales

Hasta ahora hemos estado hablando de recomendaciones individuales, pero uno de los objetivos de Madrid Live era lograr hacer recomendaciones de planes para grupos.

Es ahí cuando entran en juego los ***Recomendadores grupales***, los cuales consisten en la generación de una recomendación para un grupo a partir de la información sobre las preferencias individuales de cada uno de los miembros que lo componen (ver: Jameson y Smith (2007)).

Hay diversas soluciones que permiten combinar las distintas recomendaciones individuales generadas para dar lugar a una recomendación grupal. Algunas de ellas son:

1. **Mezclar las recomendaciones individuales**
2. **Crear una agregación** de las puntuaciones individuales para un conjunto determinado de productos.
3. **Tratar de manera diferente** a cada miembro del grupo.
4. **Crear un modelo del grupo** como si fuera un individuo y hacer una recomendación individual al modelo.

Tras valorar las distintas posibilidades, optamos por crear una agregación de las puntuaciones individuales para un conjunto determinado de ítems.

Para ello, lo primero que hacemos es realizar una predicción de la puntuación que asignaría cada usuario miembro del grupo a cualquier actividad y recomendaríamos el conjunto de actividades que hayan obtenido un mayor valor en la 'valoración agregada'.

Existen distintas funciones de agregación (ver Gartrell et al. (2010)) para calcular la valoración agregada. En Madrid Live hemos utilizado las tres que explicamos a continuación:

1. **Maximizar la satisfacción media**:

   Intentar que el grupo quede lo más satisfecho posible con la recomendación.

   $R_i = media(r_{ij}) = \frac{1}{n}\sum_{j=1}^{n} r_{ij}$

1. **Minimizar miserias**:

   Intenta que ningún miembro quede muy insatisfecho.

   $R_i = MAX(min_j r_{ij})$

1. **Máxima satisfacción**:

   Intenta que el usuario más satisfecho con una actividad sea el que prevalezca sobre todos los demás.

   $R_i = MAX(max_j r_{ij})$

Para entender mejor todo lo que hemos explicado hasta ahora, veremos un ejemplo para cada función de agregación. Supongamos que tenemos 3 usuarios (user1, user2 y user3) y 3 recomendaciones propuestas para cada uno de ellos (rec1, rec2, rec3). La siguiente tabla muestra el nivel de satisfacción (variable entre el 0 y el 1) que obtienen tras realizar una recomendación individual.

	User1	**User2**	**User3**
Rec1	0.8	0.4	0.9
Rec2	0.3	0.7	0.4
Rec1	0.4	0.6	1.0

1. **Maximizar la satisfacción media:** $R_i = media(r_{ij}) = \frac{1}{n}\sum_{j=1}^{n} r_{ij}$

   - $MediaRec1 = 1/3(0{,}9 + 0{,}4 + 0{,}9) = 0{,}733$
   - $MediaRec2 = 1/3(0{,}3 + 0{,}7 + 0{,}3) = 0{,}433$

- $MediaRec3 = 1/3(0{,}2 + 0{,}6 + 1{,}0) = 0{,}6$

La recomendación propuesta para el grupo es Rec1, puesto que posee la media más alta (0.733).

2. **Mínima miseria:** $R_i = MAX(min_j r_{ij})$

- Valor mínimo para User1 = 0.2 (Rec3).
- Valor mínimo para User2 = 0.4 (Rec1).
- Valor mínimo para User3 = 0.3 (Rec2).

El valor máximo de entre los mínimos es 0.4, por lo que devolvemos Rec2.

3. **Máxima satisfacción:** $R_i = MAX(max_j r_{ij})$

- Valor máximo para User1 = 0.9 (Rec1).
- Valor máximo para User2 = 0.7 (Rec2).
- Valor máximo para User3 = 1.0 (Rec3).

El valor máximo de entre los máximos es 1.0, por lo que devolvemos Rec3.

Por último, indicar que tal y como se ha podido comprobar, en todos los casos anteriores tratamos a todos los miembros del grupo de igual manera. Sin embargo, se pueden aplicar distintos pesos y filtros a cada miembro del grupo, logrando así que las opiniones de unos sean más determinantes que las de otros. Por ejemplo, establecer pesos distintos para un padre y su hijo, haciendo que la opinión de uno se tenga mucho más en cuenta que la del otro.

Aunque esta opción no está implementada en nuestra aplicación es una posible ampliación que se podría llevar a cabo de forma sencilla, más adelante en el apartado de trabajo futuro (Sección 5.1) se explicará más en detalle.

2.2. Tecnologías usadas

En esta sección describiremos las principales tecnologías que hemos ido empleando a lo largo de todo el desarrollo de nuestro proyecto, Madrid Live.

Figura 2.5: Tecnologías usadas

2.2.1. Java

Java fue diseñado por James Gosling en 1990 como software para dispositivos electrónicos de consumo. Las principales razones que llevaron a la creación de este lenguaje, fueron:

- Creciente necesidad de interfaces más cómodas e intuitivas.
- Fiabilidad del código y facilidad de desarrollo.
- La enorme diversidad de controladores electrónicos.

Se trata de un lenguaje de programación de propósito general, concurrente, basado en clases, y orientado a objetos. Fue diseñado con la intención de permitir a los desarrolladores escribir un programa una única vez y ejecutarlo en cualquier dispositivo. Esto es gracias a la **máquina virtual de Java** (en inglés Java Virtual Machine, JVM).

La JVM es una de las piezas fundamentales de la plataforma Java. Se trata de una máquina virtual de proceso nativo, es decir, ejecutable en una plataforma específica, capaz de interpretar y ejecutar instrucciones expresadas en un código binario especial, el cual es generado por el compilador del lenguaje Java.

La gran ventaja de la máquina virtual java es aportar portabilidad al lenguaje. De ahí el famoso axioma que sigue a Java: "Write once, run anywhere".

Hoy en día, Java es uno de los lenguajes de programación más populares en uso. Actualmente cuenta con unos 10 millones de usuarios reportados.

2.2.2. HTML y CSS

HTML (HyperText Markup Language) («lenguaje de marcado hipertextual»), hace referencia al lenguaje de marcado predominante para la elaboración de páginas web.

Se trata de un lenguaje muy sencillo que permite escribir texto de forma estructurada y clara, además de complementarlo con diferentes objetos, como son las imágenes o animaciones. Está compuesto por etiquetas (rodeadas por corchetes angulares (<,>)), que marcan el inicio y el fin de cada elemento del documento. En cuanto a la versión de HTML utilizada, comentar que utilizamos HTML5 para desarrollar toda la interfaz web que verá y manejará el usuario que haga uso de nuestra aplicación.

Con el fin de separar la estructura de nuestras páginas web del diseño elegido para las mismas, vamos a emplear las hojas de estilo en cascada o CSS (en inglés, CascadingStyle Sheets). Por lo tanto, podríamos definir el CSS es un lenguaje de estilo que define la presentación de los documentos HTML. Con estas hojas de estilo, abarcamos cuestiones relativas a fuentes, colores, márgenes, líneas, altura, anchura, imágenes de fondo o posicionamiento avanzado (entre otras muchas cosas).

2.2.3. JQuery

jQuery es una biblioteca de JavaScript creada por John Resig. Se trata de unas librerías de código que contienen procesos o rutinas ya listos para usar.

Una de sus finalidades es lograr que los desarrolladores no tengan que realizar las tareas más básicas, si no que utilicen las implementaciones ya existentes en el propio framework, las cuales están probadas y se garantiza que funcionan correctamente.

jQuery permite crear sitios web interactivos moviendo los elementos HTML alrededor, creando animaciones personalizadas, y logrando que los usuarios puedan afectar a un sitio web haciendo clic en el ratón o escribiendo desde el teclado. jQuery, gracias a su flexibilidad y extensibilidad, ha cambiado la forma en que millones de personas escriben JavaScript.

2.2.4. J2EE

La plataforma J2EE permite el desarrollo de servicios en Internet mediante el lenguaje Java. Mediante JSP (un lenguaje de programación de páginas web dinámicas como PHP o ASP) y Servlets (explicados en mayor detalle a continuación) se pueden desarrollar sitios Web bajo la tecnología Java.

En nuestro caso, usamos J2EE para implementar los servlets. Un servlet es un pequeño programa Java que se ejecuta dentro de un servidor Web y permite extender la funcionalidad del mismo gestionando los mensajes

transmitidos entre el cliente y el servidor. Por lo general, estas solicitudes de petición se llevan a cabo a través del protocolo de transferencia de hipertexto (HTTP).

La finalidad de los servlets es la de generar todas las páginas de nuestra web de forma dinámica a partir de los parámetros que obtiene de la petición.

Hay 4 puntos que destacar durante la vida de un Servelt:

1. El cliente solicita una petición a un servidor a través de una URL.
2. El servidor recibe dicha petición. Una vez que la ha recibido, pueden darse dos casos:
   - Si es la primera, el servlet se construye y se procede a llamar al método `init()`.
   - Si ya está iniciado, cualquier otra petición dará lugar a un nuevo hilo.
3. Se procesa la petición y se devuelve el resultado al cliente a través del método `service()`, quien averigua si la petición es un GET o un POST e invoca al método correspondiente `doGet() , doPost(), doPut(), doDelete()`.
4. Si estima que ya no es necesario, el contenedor invoca el método `destroy()` del Servlet y lo retira (destruyéndolo y liberando todos los recursos abiertos).

2.2.5. SQL

SQL (por sus siglas en inglés Structured Query Language, en español, Lenguaje de Consulta Estructurado) es un lenguaje de programación declarativo de acceso a bases de datos relacionales que permite especificar diversos tipos de operaciones en ellas. En nuestro caso, el sistema de gestión de bases de datos que hemos utilizado es MySQL.

Este interpretador del lenguaje SQL permite crear base de datos y tablas, insertar datos, modificarlos, eliminarlos, ordenarlos, hacer consultas y realizar otras muchas operaciones. En definitiva, podemos decir que nos permite administrar nuestra base de datos.

Actualmente se ha convertido en un estándar de lenguaje de bases de datos, y la mayoría de los sistemas de bases de datos lo soportan.

```
JSON generado...
{
"valorado" :"Museo Nacional del Ferrocarril",
"visitado" : [
"Museo Nacional del Ferrocarril",
"Museo Naval"
],
"tamanyo" :"2",
"status" : "OK"
}
```

Figura 2.6: Formato JSON

2.2.6. JSON

Para la comunicación entre la interfaz web y la API de servicios de nuestro sistema hemos utilizado JSON[3] (JavaScript Object Notation). JSON es un sistema se intercambio de datos ligero que facilita a las personas la comprensión de la información que se desea comunicar y es fácil de parsear y generar para las máquinas. Esto hace que sea una herramienta muy usada en el mundo web.

Es un formato de texto que el cual es un lenguaje completamente independiente pero que tiene muchas convenciones y es familiar para los programadores de lenguajes como C, C++, Java, JavaScript y muchos otros. Estas propiedad hacen que JSON sea el método ideal para el intercambio de información.

Como podemos ver en la Figura 2.6 el formato del JSON es una colección de pares clave/valor que pueden estar agrupados en listas. Estas colecciones se convertirán en listas de objetos que podrán ser registros, tablas hash, etc. Esto nos ha facilitado mucho la comunicación entra las distintas partes de nuestro trabajo.

2.2.7. jCOLIBRI

jCOLIBRI [4] (ver: Recio García (2008)) es una plataforma creada en 2005 para desarrollar proyectos basados en el razonamiento basado en casos. Su arquitectura ha sido diseñada para ser extensible y reutilizada por los diferentes dominios y familias CBR. Ofrece muchos de los elementos necesarios para el desarrollo de sistemas CBR y provee al programador de numerosos ejemplos que son útiles para testear el framework y son parte de su documentación. Dispone también de 5 estrategias diferentes de métodos de

[3] http://www.json.org/
[4] http://gaia.fdi.ucm.es/research/colibri/jcolibri

Figura 2.7: Logo jCOLIBRI

recuperación con 7 métodos de selección y 30 tipos diferentes de métricas de similitud.

2.3. Mecanismos para la obtención de preferencias del usuario

El sistema de puntuación Elo es un método matemático, basado en cálculo estadístico usado para calcular la habilidad relativa de los jugadores de ajedrez. En Madrid Live, lo emplearemos para obtener un listado ordenado de los museos según las preferencias de cada usuario. Es decir, utilizando y adaptando el algoritmo original, construiremos su ranking de preferencias para este tipo de actividad.

Este sistema fue elaborado por el profesor de física húngaro Arpad Emrick Elo, un gran aficionado al ajedrez que desarrolló este método de cálculo de rankings para los jugadores de ajedrez considerando su nivel.

Este dato es muy importante ya que la fórmula empleada en el cálculo pondera en función de la diferencia de nivel entre los dos jugadores. Esto es, asigna menos puntuación al contrincante ganador, si es el que tiene más nivel, o por el contrario, asigna una mayor puntuación,si es el jugador de menor nivel el que gana la partida.

En 1970, la FIDE (Fédération Internationale des Échecs) acordó adoptar el sistema de puntuación Elo para clasificar a los mejores ajedrecistas de todo el mundo. Este sistema es ampliamente usado también para otros juegos pero todos de competición por parejas.

Años más tarde (Octubre de 2003) Mark Zuckerberg utilizó este siste-

ma para realizar un sitio web llamado "Facemash" que precedió al conocido "Facebook". El propósito de esta página web era clasificar a las chicas de la universidad de Hardvard según su atractivo. El programa mostraba dos imágenes de diferentes chicas y el usuario elegía cual le gustaba más. Mark Zuckerberg utilizaba el sistema de puntuación Elo para, según las respuestas de los usuarios, hacer un ranking de popularidad de las chicas.

A continuación, vamos a explicar en detalle cómo funciona este sistema y cómo lo hemos adaptado a nuestro caso (Sección 3.3.1.1).

2.3.1. Sistema de puntuación Elo

Todo el cálculo matemático del sistema de puntuación Elo (ver Elo (1978)), se basa en las fórmulas que se muestran a continuación:

$$E_A = \frac{1}{1 + 10^{(R_B - R_A)/400}} \quad E_B = \frac{1}{1 + 10^{(R_A - R_B)/400}}$$

Siendo:

- **EA:** ranking esperado para el jugador A (estimación)
- **RA:** Puntuación previa (puntos acumulados hasta el momento) del jugador A.
- **EB:** ranking esperado para el jugador B (estimación)
- **RB:** Puntuación previa (puntos acumulados hasta el momento) del jugador B.

Sin embargo esto sólo permite obtener el valor del ranking esperado, por lo que posteriormente debe ser procesado junto con los resultados en una segunda fórmula que es continuación de la anterior.

La fórmula para generar la nueva puntuación es la siguiente:

$$Rn = Ro + C * (S - Se)$$

Dónde:

- **Rn:** es la nueva puntuación para el producto N.
- **Ro:** es la puntuación anterior (que se usó con Ra y Rb).
- **C:** constante que varía de acuerdo al tipo de cálculo.
- **S:** especifica el resultado de la elección (Tabla 2.1)

Resultado	Valor de S
Gana	1
Empata	0.5
Pierde	0

Tabla 2.1: Tabla con los distintos valores de S

- **Se:** es el ranking esperado (valor que obtuvimos en la primera fórmula).

Es importante destacar que el algoritmo explicado anteriormente, considera el "nivel" de cada jugador, pues, como habíamos comentado anteriormente, no se puede comparar del mismo modo una victoria o derrota (en nuestro caso) de un museo que tiene un nivel muy alto contra otro que posee una puntuación muy baja.

Si así fuese, la fórmula pondera en función de la diferencia, asignando menos puntuación al que tenía más si es que gana, o un mayor puntuación al que derrotó a uno de alto nivel.

2.4. Conclusiones

En este capítulo se hace un repaso de las técnicas utilizadas. Se comienza por los recomendadores individuales explicando los que son basados en contenido, los ricos en conocimiento y el filtrado colaborativo. A continuación se explica el funcionamiento de los recomendadores grupales haciendo hincapié en las funciones de agregación utilizadas. Después se describen las técnologias utilizadas en el prototipo de las que se puede destacar JQuery y JCOLIBRI. Por último se explican los mecanismos para la obtención de preferencias de usuario.

Capítulo 3

Madrid Live: un Recomendador Social de Ocio

En este capítulo hablaremos de como funciona Madrid Live, y como está diseñado para ser un recomendador de actividades de Ocio en Madrid. Se empezará hablando de como es el diseño del sistema, el siguiente punto será saber donde Madrid Live consigue el catálogo de actividades y por último hablaremos de como se consiguen las preferencias de los usuarios.

3.1. Diseño del Sistema

A continuación explicaremos el diseño teórico de nuestra aplicación, Madrid Live. Veremos cómo es su arquitectura, esto es, los distintos módulos de los que está compuesta la aplicación, y las diversas funcionalidades que ofrece al usuario.

3.1.1. Arquitectura Modular

El diseño de Madrid Live lo hemos dividido en 3 módulos (Figura 3.1):

1. ***Núcleo:*** Es la parte más importante de Madrid Live ya que contiene todos los recomendadores desarrollados para ofrecer plantillas de planes adaptadas a los gustos de cada persona y la gestión de los distintos test de preferencias.

   El núcleo está conectado con la base de datos del servidor, de la cual obtendremos los datos necesarios para las recomendaciones: tanto los relativos a los usuarios (datos personales, pesos,...) como los que hacen referencia al catálogo estático de actividades. Las fuentes de datos utilizadas se detallan en 3.2.

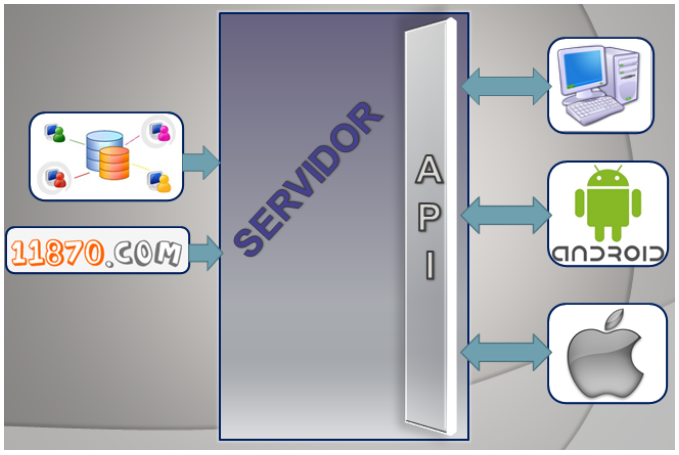

Figura 3.1: Diseño de la arquitectura

2. ***Servicios:*** Esta capa es la que ofrece los servicios del núcleo a las distintas interfaces que se implementen. Está diseñada con J2EE (Sección 2.2.4) y tiene una función de web service por cada funcionalidad que ofrecemos.

   Esta capa recibe una petición de la interfaz y hace llegar al núcleo la solicitud. Cuando el núcleo ya ha ejecutado la acción, por ejemplo, pedir la recomendación de una plantilla para un usuario, el núcleo envía el resultado al web service para que éste se encargue de enviarlo a la interfaz, en formato Json (Sección 2.2.6), que ha solicitado ese servicio.

3. ***Interfaz:*** Es importante destacar que la aplicación está preparada para distintas plataformas (web, iOs, Android...). Para hacer la interfaz de usuario nosotros hemos optado por hacer una aplicación web, en la que se pueden ejecutar todos los servicios que proporciona la herramienta, como puede ser recomendar una plantilla o valorar una actividad, entre otras muchas. Ésta aplicación web está diseñada con HTML5 (Sección 2.2.2) y jQuery (Sección 2.2.3).

3.1.2. Descripción funcional

Antes de explicar las funcionalidades del sistema, debemos explicar el elemento principal que usamos para poder hacer las recomendaciones.

Madrid Live es un *recomendador de plantillas de actividades*. Estas plantillas son creadas por el sistema o por el usuario indicando en ellas los tipos de actividades que desean incluir en su plan (por ejemplo, Museo-Restaurante-Parque) y el horario en el que va a transcurrir dicha plantilla.

Figura 3.2: Ejemplo de una plantilla abstracta

Figura 3.3: Ejemplo de una plantilla concreta

Una vez que tenemos la plantilla, a la cual llamamos *Plantilla Abstracta* (Figura 3.2), la aplicación se encargará de insertar las distintas actividades concretas asignadas a cada tipo. Por ejemplo, para la plantilla anterior la aplicación podría proponer el Museo del Prado como museo, el Retiro como parque, etc.

Cuando una Plantilla Abstracta ya tiene las actividades concretas asignadas, pasa a denominarse *Plantilla Concreta* (Figura 3.3).

Ahora sí, una vez aclarados ambos conceptos, podemos proceder a explicar la descripción funcional de nuestro proyecto. En Madrid Live hemos implementado numerosos servicios, pero muchos de ellos son muy triviales por lo que no se explicaran en este apartado (ejemplo de ello, podrían ser: login, logout,...). A continuación se explicarán los servicios que cumplen los distintos objetivos (Sección 1.1) que queríamos cubrir con nuestra aplicación:

3.1.2.1. Test de Preferencias

Con el fin de que nuestra aplicación tenga unos datos iniciales con los que poder realizar una recomendación, es necesario llevar a cabo un test de preferencias para conocer algo acerca de los gustos de un determinado usuario.

En Madrid Live, se realizan test de preferencias sobre dos actividades distintas: museos y restaurantes (ya que son las actividades incluidas en nuestros planes que dependen en mayor grado de los gustos de cada usuario), por lo que es condición necesaria que el usuario complete ambos test antes de realizar cualquier recomendación.

- **Precondición**: El usuario debe estar identificado en el sistema.

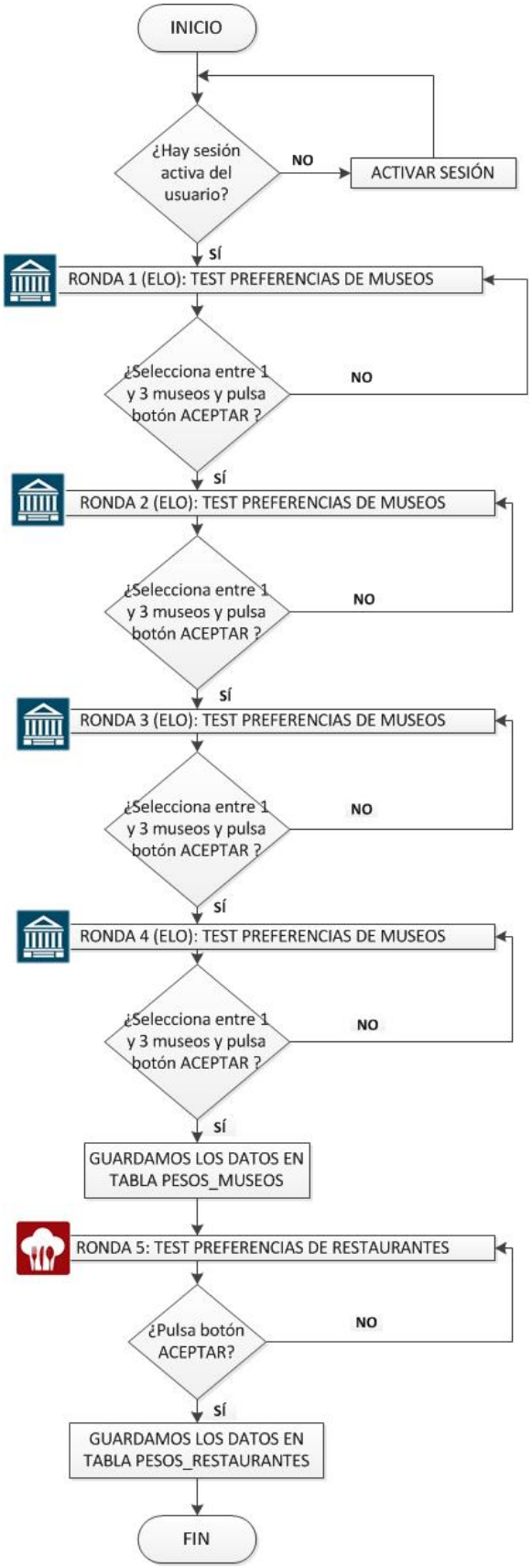

Figura 3.4: Diagrama de flujo normal: Test de preferencias

- **Postcondición**: El sistema contendrá los pesos del usuarios sobre los tipos de museos y restaurantes.
- **Flujo Normal**: (Figura 3.4) El sistema proporcionara al usuario 5 rondas para conocer sus gustos sobre los distintos tipos de museos y restaurantes. Las 4 primeras rondas se corresponden con el test Elo (preferencia de museos) mientras que la última hace referencia a los gustos de los usuarios sobre los tipos de restaurantes.
  - ***Paso 1: Test Elo(1).*** El sistema proporciona al usuario 6 tipos de museos elegidos aleatoriamente de los cuales el usuario debe elegir un máximo de 3. Una vez que el usuario haya seleccionado los tipos de museos que más le gusten, deberá pulsar sobre el botón Aceptar.
  - ***Paso 2: Test Elo(2).*** El sistema calcula el resultado de la selección de los distintos tipos de museos que se han seleccionado en la ronda anterior. Una vez que el sistema ha hecho dichos cálculos muestra al usuarios otros 6 tipos de museos, donde 3 se corresponden con el grupo de tipos de museos más valorados y los otros 3 con el segundo grupo más valorado por el usuario.
  - ***Paso 3: Test Elo(3).*** El sistema calcula el resultado de la selección de los tipos de museos que ha seleccionado el usuario en la ronda anterior. Una vez que el sistema ha hecho dichos cálculos muestra al usuario otros 6 tipos de museos, donde 3 se corresponden con el grupo de tipos de museos más valorado y los otros 3 con el grupo de tipos de museos que no han salido en la ronda anterior.
  - ***Paso 4: Test Elo(4).*** El sistema calcula el resultado de la selección de los tipos de museos que se han seleccionado en la ronda anterior. Una vez que el sistema ha hecho dichos cálculos muestra al usuario la última ronda de 6 museos. Éstos se corresponden con aquellos que hayan obtenido una mayor puntuación en el ranking de Elo.
  - ***Paso 5: Test Restaurantes.*** El sistema calcula el resultado de la selección de los tipos de museos que ha seleccionado el usuario en la ronda anterior y concluye el calculo de los pesos del usuario con los tipos de museos. A continuación la aplicación muestra una lista con todos los tipos de restaurantes, el usuario debe ordenar esa lista de restaurantes ordenándolos de mayor a menor según sea le guste o no el tipo de restaurante al usuario.
  - ***Paso 6: Guardar Resultado Restaurante.*** El sistema guarda los resultados del test de restaurantes. Una vez el sistema ha hecho ésto, el test de preferencias se da por terminado.

3.1.2.2. Valoración de la Plantilla

Cuando un usuario ha realizado el plan que se le ha recomendado, éste puede valorar las distintas actividades que estaban incluidas en él, con el fin de mejorar los pesos de sus gustos. De esta forma, la aplicación perfecciona las futuras recomendaciones para ese usuario concreto.

- **Precondición:** El usuario debe estar identificado en el sistema y haber aceptado alguna recomendación.
- **Postcondición:** El sistema guarda las valoraciones de las actividades y ajusta los pesos de los gustos según las nuevas valoraciones proporcionadas por el usuario.
- **Flujo Normal:** (Figura 3.5) El sistema mostrará al usuario todas aquellas actividades que ha realizado y aún no han sido valoradas y ajustará los pesos de dichas actividades.
  - ***Paso 1: Mostrar Actividades.*** El sistema muestra todas las actividades realizadas por el usuario y que aun no han sido valoradas. El usuario valorará una actividad con un número comprendido entre el 1 y el 5 y pulsará el botón "Valorar" para que se guarde el resultado de la valoración.
  - ***Paso 2: Guardar Valoración.*** El sistema guarda la valoración de esa actividad, y dependiendo del tipo de actividad además debe hacer lo siguiente:
    - **Restaurante o Museo**: Ajusta el peso del tipo de restaurante/museo al que pertenece esa actividad según la valoración que ha dado el usuario.
    - **Parque o Paseo**: Actualiza la media de valoraciones del parque/paseo para que sirva en otras recomendaciones.

    Si esa actividad es la última que faltaba por valorar en el plan, se calcula la valoración media del plan. Una vez hecho esto, la plantilla ya puede estar disponible para el Recomendador de Plantillas Concretas.

3.1.2.3. Obtención de la Recomendación

Este servicio busca una plantilla teniendo en cuenta las restricciones que pone el usuario (horario, localización,...) y adapta esa plantilla, si fuese necesario, poniendo actividades concretas que se corresponden con los gustos del usuario.

- **Precondición:** El usuario debe estar identificado en el sistema.

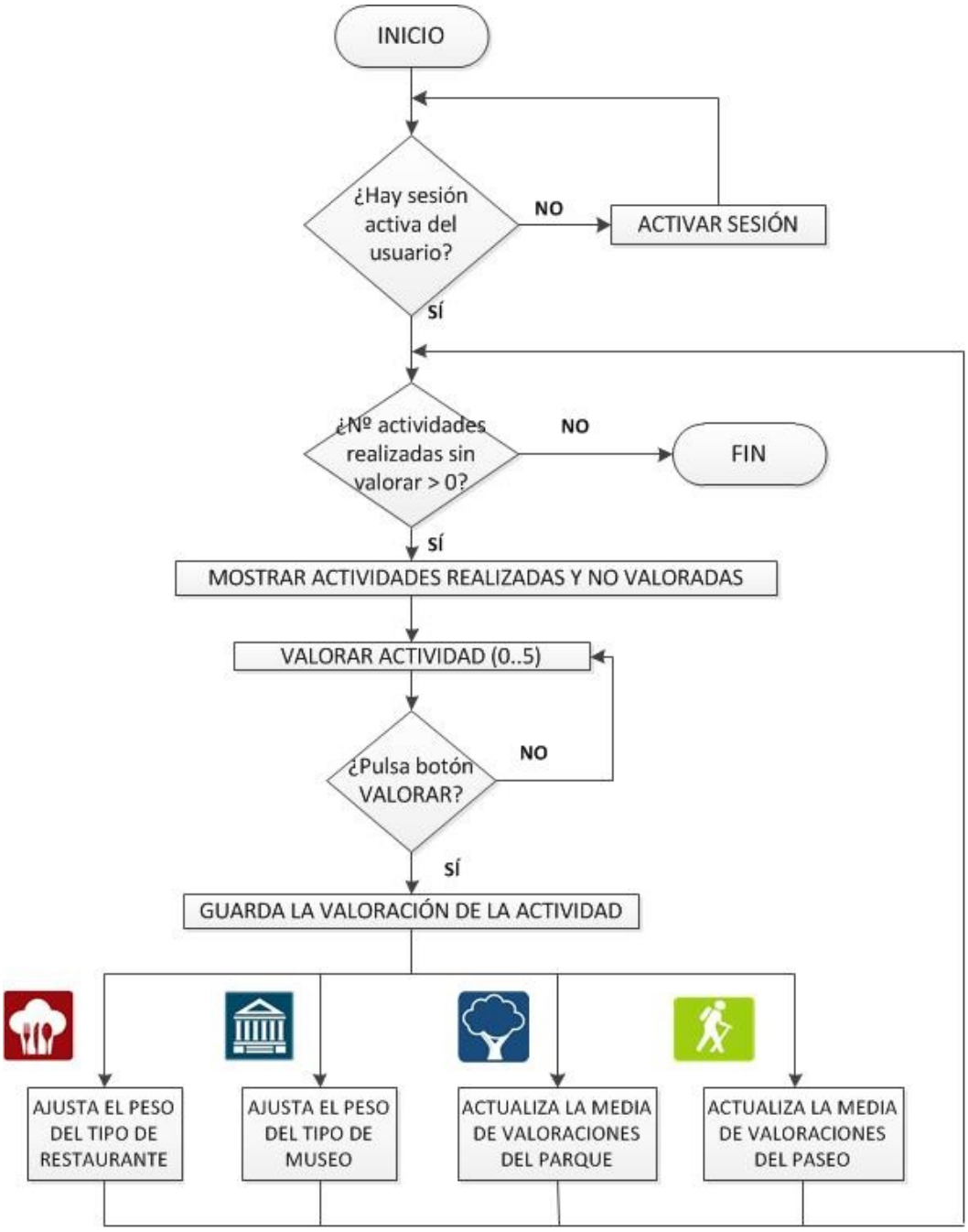

Figura 3.5: Diagrama de flujo normal: Valorar plantilla

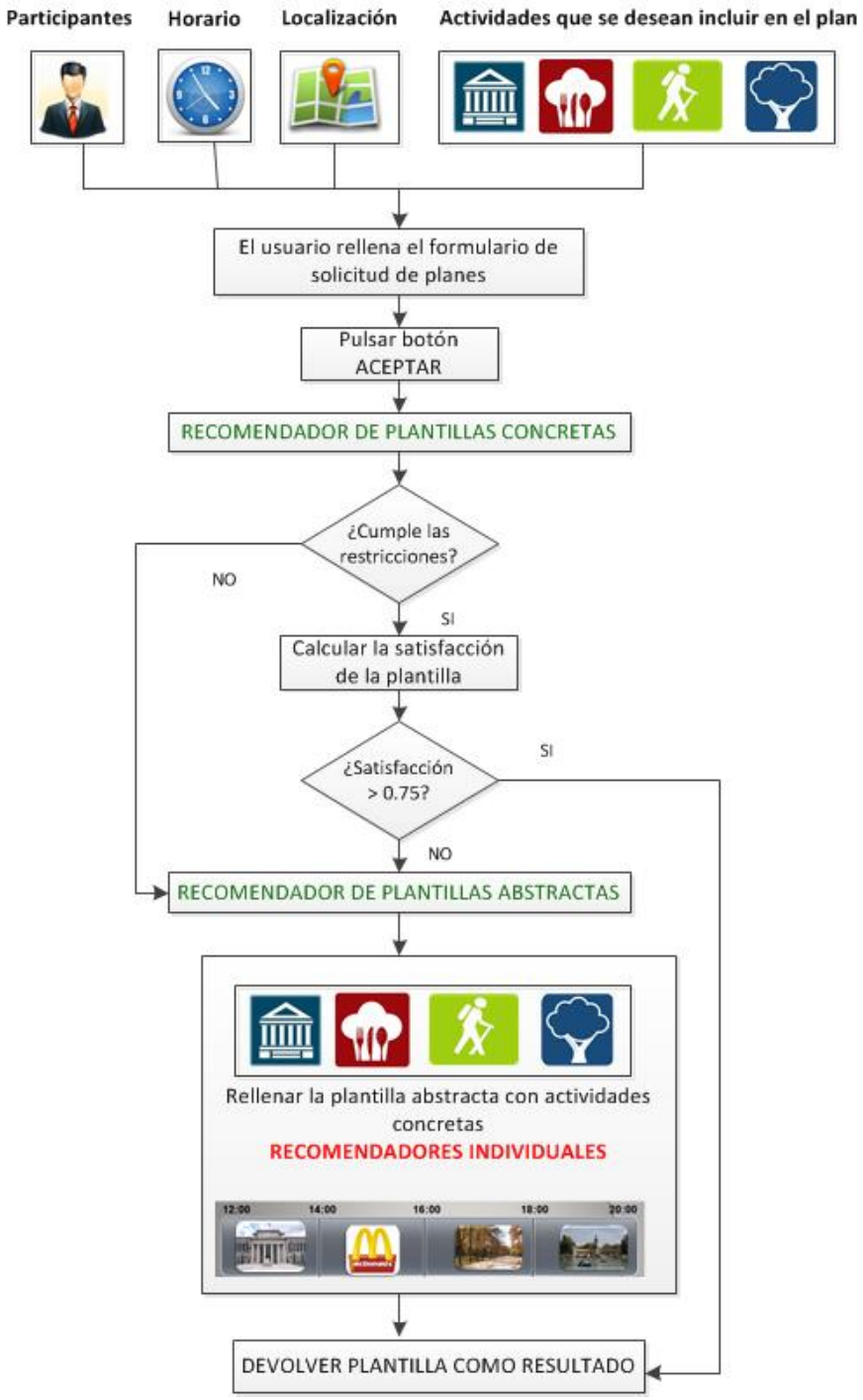

Figura 3.6: Diagrama de flujo normal: Obtención de la recomendación

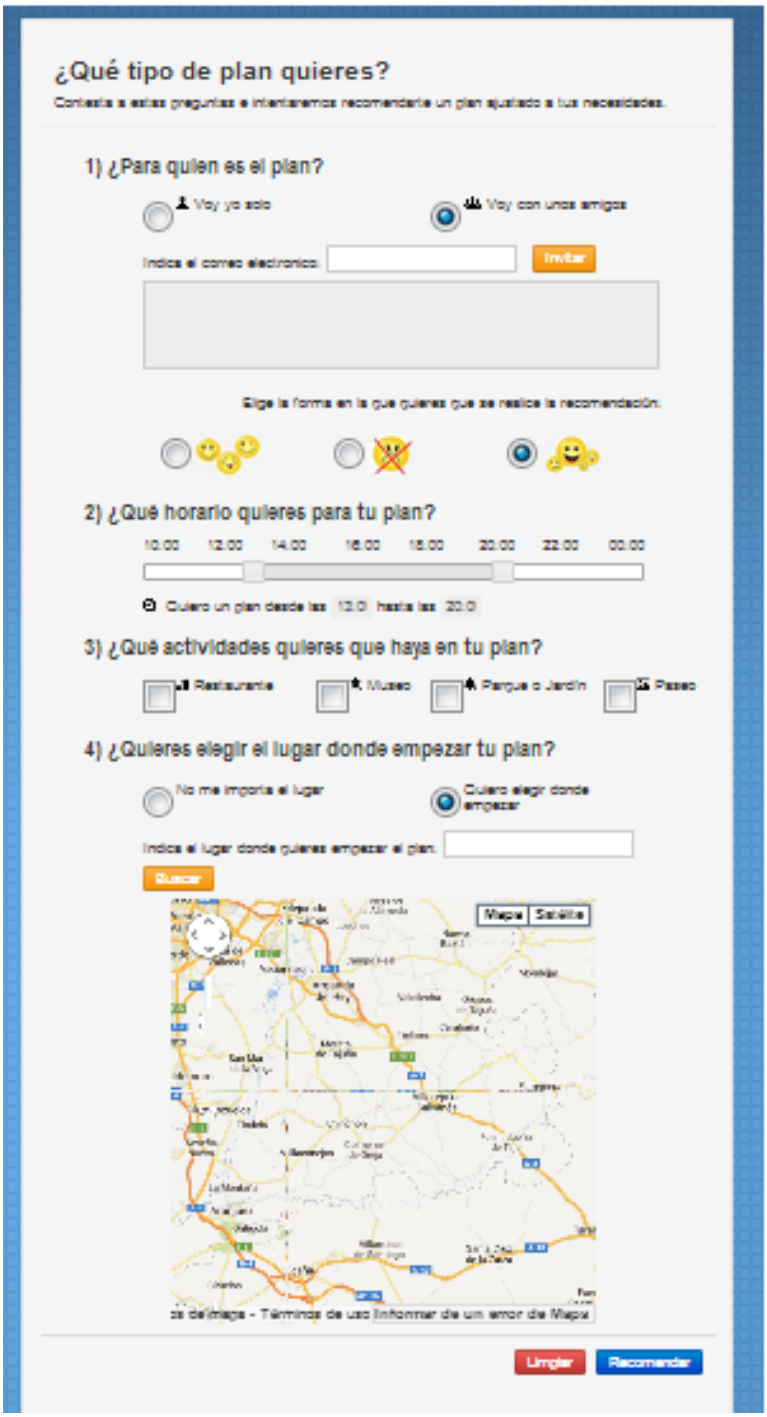

Figura 3.7: Formulario para solicitar la recomendación

- **Postcondición:** El sistema devuelve una plantilla rellena con las actividades que se adaptan las preferencias del usuario.
- **Flujo Normal:** (Figura 3.6) El sistema buscará una plantilla que se adapte a las restricciones del usuario y cuyas actividades se ajusten al perfil del usuario.
  - ***Paso 1: Rellenar Formulario.*** El usuario rellena un formulario donde pone las condiciones que tiene que tener la plantilla (ver Figura 3.7). Las opciones que debe rellenar son:
    - Personas que va a participar en el plan. Si no marca nada es que el usuario hará el plan solo.
    - Horario en el que va a transcurrir el plan. Obligatorio.
    - Tipos de actividades que quiere que haya en su plan.
    - Localización del lugar donde comenzar el plan.

    Una vez que el usuario ha indicado los parámetros de la consulta pulsa el botón Recomendar.
  - ***Paso 2: Recomendador de Plantillas Concretas.*** El sistema busca en la base de datos, plantillas realizadas por otros usuarios. Debe buscar las plantillas y cumplir lo siguiente:
    - Restricciones del formulario rellenado por el usuario, es decir, horario, localización inicial y tipos de actividades que quieren que aparezcan (pueden aparecer más tipos de actividades).
    - Obtener la satisfacción de ese usuario para la plantilla. El sistema calcula dicha satisfacción como el resultado de hacer la media de la satisfacción de ese usuario con respecto a cada actividad del plan. En caso de que la recomendación se haga para más de una persona la aplicación calculará la satisfacción de cada actividad según la función de agregación que haya seleccionado el usuario que ha solicitado la recomendación. Una vez que ya tiene el resultado de satisfacción de cada actividad se calcula la media entre todas las soluciones de la función de la agregación hecha a cada actividad del plan como hemos indicado anteriormente.

    Si se recupera alguna plantilla cuya satisfacción media para el usuario (o usuarios) sea mayor de 0,75, entonces se devuelve dicha plantilla como solución, sino existe ninguna plantilla que cumpla ésto se realizará el paso siguiente.
  - ***Paso 3: Recomendador de Plantillas Abstractas.*** El sistema busca en la base de casos de Plantillas Abstractas una que cumpla las restricciones puestas por el usuario en el formulario de búsqueda, es decir, horario, localización inicial y tipos de actividades que quieren que aparezcan. Una vez que el sistema ha

devuelto una plantilla se procede a rellenar la plantilla con actividades. Por cada tipo de actividad que haya se ejecuta el Paso 4.

- ***Paso 4: Recomendadores de Actividades.*** El sistema va llamando a cada recomendador según el tipo de actividad que haya que rellenar, y cada recomendador devolverá una actividad concreta, esta actividad concreta dependerá de:
  - El plan es para una sola persona: entonces se devuelve la actividad que mas satisfacción tenga para ese usuario y que el usuario no haya realizado dicha actividad en los últimos 6 meses.
  - El plan es para un grupo de personas: por cada usuario se devuelve las 10 actividades que más satisfacción tengan para cada usuario y se unen a la función de agregación, cuando ya se ha realizado este proceso para todos los usuarios se devuelve la mejor actividad según la función de similitud elegida, por ejemplo, si la función de agregación escogida es la media se devuelve la actividad que tiene mejor satisfacción media entre todos los usuarios que van a participar en el plan.

  Una vez que se ha rellenado toda la plantilla se envía el resultado al usuario.
- ***Paso 5: Respuesta del Usuario.*** El usuario indicará si le gusta o no le gusta el plan:
  - Si le gusta el plan, esa plantilla se guardará en la base de datos y el usuario podrá valorarla cuando haya realizado el plan.
  - Si no le gusta el plan, el sistema volverá a repetir el proceso pero desechando los resultado del plan rechazado por el usuario.

3.2. Fuentes de conocimiento

Sin duda una de las partes más importantes de nuestro proyecto radica en las fuentes de información utilizadas. Antes de realizar cualquier recomendación, necesitamos conocer todos los datos necesarios acerca de la actividad que vamos a recomendar. Por ejemplo, si vamos a recomendar un restaurante, necesitaremos saber con antelación su nombre, su ubicación, el tipo de comida que ofrece e incluso la valoración que tiene dicho restaurante para otros usuarios. Para ello, necesitamos buscar fuentes que nos proporcionen toda esa información.

Figura 3.8: Catálogos de Recomendación

En nuestro caso, haremos una importante distinción entre 2 tipos de fuentes de información. La primera de ellas hace referencia al catálogo de actividades que vamos a recomendar en nuestra aplicación web, y la segunda a toda la información referente al usuario que hará uso de la misma. A continuación explicaremos en mayor detalle cada una de ellas.

3.2.1. Catálogo de recomendación

El catálogo de recomendación va a contener toda la información necesaria acerca de las distintas actividades que vayamos a recomendar. Como por ejemplo: Nombre, dirección, teléfono de contacto, página web, tipo de comida que ofrece (en el caso de restaurantes) o la duración total de la actividad (en el caso de un paseo), entre otras cosas.

Con el fin de hacer más modular la aplicación y ofrecer una amplia variedad de recursos, hemos optado por utilizar dos tipos de catálogos: estáticos y dinámicos (Figura 3.8).

- **Catálogos estáticos**: Los empleados para almacenar la información de los museos, los parques, monumentos o paseos. En este caso, lo que hacemos es descargar una única vez todos los datos necesarios y los almacenamos en nuestras bases de datos, accediendo a éstas siempre que tengamos que hacer alguna consulta.

- **Catálogos dinámicos**: Los empleados para almacenar la información de los restaurantes. En este caso, no nos interesa descargarnos toda la información relativa a todos los restaurantes de Madrid, puesto que la cantidad de datos sería excesiva. Por tanto, lo que haremos será descargar en cada consulta, la información necesaria de todos los restaurantes que estén a un 1 km a la redonda de un punto concreto.

3.2.1.1. Origen de los datos

En cuanto al origen de los datos, indicar que toda la información necesaria la obtenemos de dos web distintas, 11870.com y Google Places. A continuación explicamos en mayor detalle cada una de ellos.

11870.com Se trata de una página web donde los usuarios pueden guardar y compartir los sitios y servicios que más le gusten en cualquier parte del mundo. Todo esto lo hacen a través de opiniones, fotos y/o vídeos.

El usuario entra en 11870.com y, a través de un buscador, puede encontrar todo tipo de locales. Además, una vez que encuentra lo que buscaba, por ejemplo restaurantes en Chamartín, puede ver fotografías, vídeos y comentarios que le ayuden a elegir el que más se ajuste a sus necesidades. También, 11870.com permite crear tarjeteros virtuales donde organizar los sitios y servicios que más nos gustan, además de descubrir numerosos sitios y servicios nuevos a través de nuestros amigos y/o recomendarles aquellos que más nos gusten. Con 11870.com, podemos explorar una ciudad antes de visitarla.

Google Places Google Places ofrece crear un registro gratuito de comercios en los mapas de Google (Google Maps), representando así una mejora y una simplificación enorme para que el responsable de un negocio pueda promocionarse fácilmente y de forma gratuita por Internet. Además le permite obtener facilidades y estadísticas (de dónde vienen sus clientes, qué buscan en Google para llegar a la web de su negocio, etc.) que le ayudarán a tomar decisiones para seguir creciendo.

Cualquier búsqueda que de productos que realicemos a través de Internet tiene un embudo de afinación, el cual normalmente funciona así:

1. Búsqueda genérica (p.ej. comprar coches)
2. Segmentación por producto (p.ej. comprar Seat Ibiza)
3. Segmentación geolocalizada (p.ej. comprar Seat Ibiza en Madrid)

Google Place pretende eliminar el tercer paso y que para encontrar un establecimiento cerca de su casa simplemente poniendo el servicio o bien el producto que necesita tendrá un mapa con las empresas más próximas, cómo llegar hasta ellas y cualquier otro tipo de información añadida.

Podemos resumir las ventajas de este servicio como:

- Llega a millones de usuarios de forma rápida y gratuita. Hoy en día, son miles las personas que usan Google Maps para decidir dónde comprar, dónde ir o en qué lugar encontrar lo que buscan, por lo que es importante asegurarse de que su ficha de empresa se pueda encontrar fácilmente. Google Places permite crear una ficha de empresa de calidad en solo unos minutos y de manera totalmente gratuita.
- Práctico y fácil de administrar.
- Opciones de primera calidad completamente gratuitas.

A continuación, mostramos una tabla en la que se puede observar el origen de los datos de cada actividad y el tipo de catálogo empleado en cada caso (Tabla 3.1).

ACTIVIDAD	ORIGEN DE LOS DATOS	CATÁLOGO
Museos	Google Place	Estático
Restaurantes	11870.com	Dinámico
Parques	11870.com	Estático
Paseos	11870.com	Estático

Tabla 3.1: Origen de los datos y tipo de catálogo para cada actividad

3.3. Modelado del usuario

Para que un recomendador tenga éxito es muy importante obtener los gustos del usuario de la forma más precisa posible. En este punto es donde entra el modelado del usuario, que es el que nos permite sacar una referencia del usuario con sus preferencias y poder realizar las consultas a los distintos recomendadores basándonos en dichos ellos. A continuación explicamos como modelamos los usuarios en nuestra aplicación.

3.3.1. Test de Preferencias

Nuestra aplicación ofrece dos test de preferencias: museos y restaurantes. Con el fin de hacer más dinámico y ameno el hecho de rellenar ambos test, hemos decidido emplear dos técnicas distintas: algoritmo de Elo para museos y ordenación de los tipos de comida para restaurantes. En cambio, para los paseos y parques no es necesario que el usuario realice un test de preferencias, ya que los recomendamos basándonos en las valoraciones del

otros usuarios y la localización en la que se encuentre el usuario.

Puesto que la aplicación ofrece muchos tipos diferentes de museos, sería realmente difícil para el usuario ordenarlos según sus gustos, por ese motivo, hemos considerado que el algoritmo de Elo podría facilitar al usuario la elaboración de su ranking para este tipo de actividad. Este algoritmo permite que el usuario pueda ordenar todos los museos seleccionando al menos 3 de los 6 museos que le mostramos en cada ronda. En el caso de restaurantes, bastaría con ordenar los 6 tipos básicos de restaurantes que componen nuestra ontología. A continuación explicamos en detalle cada uno de ellos.

3.3.1.1. Test de Preferencias de Museos: Algoritmo de Elo

Una vez explicado cómo funciona el algoritmo original Elo para clasificar jugadores de ajedrez (Sección 2.3), vamos a ver cómo lo vamos a ir aplicando y adaptando a nuestras necesidades con el fin de hacer nuestra clasificación de preferencias para museos. Para empezar, dividiremos los 12 tipos de museos en 3 grandes grupos como se puede ver en la tabla 3.2.

ARTE	CIENCIA	OTROS
Pintura	Ciencias	Infantil
Arquitectura	Historia	Transporte
Escultura		Ropa

Tabla 3.2: División de grupos por tipos de museos

Decidimos dividir los 12 museos en 3 categorías con el fin de poder hacer un seguimiento más próximo de los museos mostrados en las 4 rondas que explicaremos a continuación. De esta forma también procuramos que el 90 % de los museos se muestren al menos una vez para ser valorados por el usuario.

Cabe destacar también que en un principio todas las puntuaciones de los museos estarán inicializadas a 0 y se moverán en un rango del 0 al 5, siendo esta última la puntuación más alta que puede tener un determinado museo.

- ***Ronda 1*: Aleatoria**

  Se muestran 6 museos aleatorios (durante esta ronda no tendremos en cuenta la clasificación de grupos anterior), de los cuales el usuario podrá elegir hasta un máximo de 3 (Figura 3.9).

Figura 3.9: Test de preferencias para museos

Una vez hecha la elección, se recalculan las nuevas puntuaciones siguiendo el algoritmo de Elo[1].

- ***Ronda 2*: Los 2 grupos con mayor puntuación**

  Una vez que hemos sacado las nuevas puntuaciones tras hacer la primera ronda, entran en juego los 3 grupos (Arte, Ciencia y Otros).

  Lo que pretendemos hacer a partir de ahora, es que los museos que mostremos no sean totalmente aleatorios, sino que podamos decidir qué grupos queremos mostrar, con el fin de hacer más fácil el descarte y encaminar en función de los gustos del usuario las siguientes rondas.

  En la ronda 2, mostraremos museos pertenecientes a los 2 grupos que más puntos obtuvieron en la ronda anterior. Es decir, se suman las puntuaciones que tienen hasta el momento todos los museos que pertenecen a cada grupo y elegimos los dos grupos más elegidos.

- ***Ronda 3*: Mas puntuado R2 vs Menos puntuado R1**

  Esta ronda también va a depender de la anterior, ya que volveremos a sumar las puntuaciones parciales de cada grupo para elegir el más elegido por el usuario y mostraremos museos de ese grupo junto con museos del grupo que no salió en la ronda anterior, esto es, el menos elegido en la primera ronda.

[1] El enfrentamiento se produce por cada par de museos, esto es, cada museo se enfrenta a todos los demás. Consideramos que un museo seleccionado gana sobre uno no elegido y empata si ambos son elegidos o no lo son.

- ***Ronda 4*: MIS FAVORITOS**

  Es la última ronda, y en ella mostraremos los museos que más le han gustado al usuario, es decir aquellos que hayan obtenido las puntuaciones más altas (sus favoritos).

Una vez finalizadas las 4 rondas, tendremos un listado ordenado de los museos según las preferencias de cada usuario. Es decir, habremos construido su ranking de preferencias.

3.3.1.2. Test de Preferencias de Restaurantes

Una vez más, antes de llevar a cabo una recomendación, es importante obtener el mayor conocimiento posible acerca de los gustos que tiene cada usuario sobre el "restaurante" que se le va a recomendar.

Para clasificar los tipos de restaurantes se ha diseñado una ontología. Ésta nos permite tener los restaurantes separados por cada tipo de comida que ofrecen y poder recuperar aquellos que sean favorables a las preferencias del usuario. El diseño de la ontología se puede ver completamente en el Apéndice A.

Para conocer las preferencias de un usuario, vamos a realizar un test sencillo, el que únicamente tendrá que ordenar las 6 categorías básicas de tipos de alimentos que hemos diferenciado en la Ontología empleada, situando en primer lugar su favorita y en la última posición aquel tipo de comida que más le disguste (Figura 3.10).
Estas 6 categorías son: Carne, verduras, pescados y mariscos, guisos, exóticos y especias y picantes.

De esta forma tendremos organizadas nuestras categorías y será más sencillo realizar una recomendación que se ajuste a lo que busca el usuario que hace uso de nuestra aplicación.

***Mejorar un perfil a través de los test de preferencias*:**

El usuario tiene la opción de mejorar su perfil tantas veces como desee. Es importante hacerle saber que cuanto más sepamos de él, mejor serán las recomendaciones que le hagamos, por esa razón, siempre que el usuario lo desee podrá realizar nuevos test de preferencias, proporcionándonos así nuevos conocimientos acerca de sus gustos.

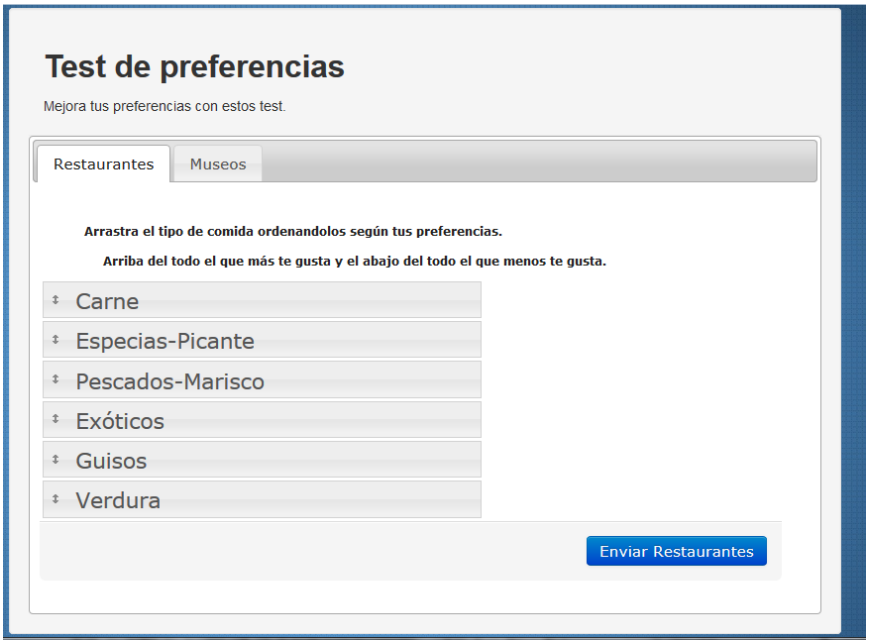

Figura 3.10: Test de preferencias para restaurantes

3.3.2. Valoraciones

Para llevar a cabo buenas recomendaciones es muy importante tener toda la información posible acerca de los gustos o preferencias de cada usuario para cada uno de los productos que vayamos a recomendar (ya sean museos, restaurantes...).

El problema de todo esto, es que realizar numerosas preguntas al usuario con el fin de obtener la máxima información posible puede convertirse en una tarea tediosa para él, lo que podría llevarle incluso a no hacer uso de la aplicación. Por esta razón, hemos pensado que lo ideal sería hacer un test de preferencias mínimo, donde con pocas preguntas podamos saber tanto como sea posible.

Lo interesante de la aplicación es que siempre va a aprender cosas nuevas de cada usuario con el fin de ir mejorando en sus recomendaciones con el paso del tiempo.

Cada vez que la aplicación realice una recomendación, el usuario tendrá la posibilidad de valorar la actividad recomendada. Esta acción nos permitirá conocer aún más detalles sobre el usuario y nos ayudará a la hora de corregir errores futuros. En definitiva, esta valoración nos permitirá aprender del usuario e ir mejorando en sus próximas recomendaciones.

Figura 3.11: Rango de valoración para una actividad

Las valoraciones se realizan por medio de una escala Likert. Cada usuario al acceder a su perfil, podrá visualizar todas aquellas actividades que la aplicación le ha recomendado y será desde ahí, donde podrá valorar cada actividad por separado, tal y cómo se puede observar en la Figura 3.11, donde:

- **Puntuación de 5**: Significa que la actividad ha sido totalmente de su agrado.
- **Puntuación de 0**: La actividad propuesta no le ha gustado nada.

3.3.2.1. Valoración de un museo

Para cada usuario registrado en nuestra aplicación, tendremos almacenado un valor que nos indique cuánto le gusta o le disgusta un tipo de museo concreto. Veamos el ejemplo para entenderlo mejor:

Imaginemos que tenemos un usuario registrado con el nombre de "user1". Se le ha recomendado el Museo del Ferrocarril. Ahora bien, si tras visitar el museo, el usuario accede a su perfil y valora la actividad positivamente (por ejemplo con 3 puntos sobre 5) tendremos que incrementar el valor del peso de la etiqueta "Transporte" (en este caso, sumaríamos 0.1 al valor inicial). Por el contrario, si tras visitar el museo el usuario marca tan sólo un punto en su valoración, disminuiríamos el valor anterior (en este caso en un 0.2). El criterio a seguir se puede observar en la imagen 3.12

La Figura 4.13 muestra cómo un usuario de nuestra aplicación valoraría uno de los museos que ha visitado, en este caso, se trata de la Casa Museo José Padilla.

3.3.2.2. Valoración de un restaurante

En el caso de los restaurantes el proceso a seguir es exactamente el mismo que para museos. La única diferencia radica en las etiquetas, que esta vez harán referencia a las 6 clases básicas en las que dividimos nuestra Ontología:

- Carne

Figura 3.12: Criterio de valoración para museos y restaurantes

Figura 3.13: Valoración de un museo

- Especias y Picante
- Exóticos
- Guisos
- Pescado y Marisco
- Verduras

3.3.2.3. Valoración de un parque

Hasta este momento, valorábamos cada "elemento" a recomendar teniendo únicamente en cuenta las preferencias de ese usuario concreto, es decir, un usuario valoraba un restaurante exótico (por ejemplo) en función de sus gustos, sin tener en cuenta la opinión del resto de personas que han hecho uso de nuestra aplicación para ese tipo de restaurante concreto.

Sin embargo, en el caso de la valoración de un parque, el procedimiento a seguir es distinto al explicado para los dos casos anteriores. En este caso, cada parque tiene una única valoración, que se calcula haciendo la media de las valoraciones que los distintos usuarios le han dado a ese parque concreto.

Por otra parte, puesto que en un principio los usuarios no habrán realizado ninguna valoración, necesitamos inicializar cada parque con un valor, para ello, tomamos la valoración que tienen en la página 11870.com de donde sacamos el resto de información de interés para parques.

3.3.2.4. Valoración de un paseo

La valoración de paseos, sigue el mismo mecanismo que el explicado anteriormente para parques. En este caso, cada paseo también será puntuado con el valor resultante de hacer la media entre las valoraciones hechas por todos los usuarios.

La diferencia con el anterior, radica en que esta vez la inicialización de las valoraciones de cada paseo, se hará teniendo en cuenta la valoración que tiene cada "punto de interés" que se va a visitar a lo largo del mismo en la web citada anteriormente (11870.com).

3.3.3. Historial

A continuación vamos a explicar cómo hemos implementado un histórico de las actividades recomendadas en nuestro sistema.

El principal objetivo de ir guardando en la base de datos cada recomendación sugerida es *evitar repeticiones*. De esta manera no mostramos a un usuario la misma actividad dos veces. Cuando el sistema hace una recomendación filtra las actividades ya realizadas por ese usuario en los últimos 6 meses y así nos aseguramos de no repetir nuestra solución.

Otro de los motivos de implementar un historial con las actividades ya realizadas es para poder consultar las plantillas concretas ya realizadas por otros usuarios anteriormente. Al tener esta información guardada podemos *reutilizar* estas plantillas y presentarlas como nuevas recomendaciones al nuevo usuario que consulta el sistema.

Capítulo 4

Madrid Live: Implementación del prototipo

Ahora pasaremos a hablar de cómo hemos ido implementando los distintos aspectos que hemos estado explicando hasta este momento en nuestra memoria. Para contar cómo es la arquitectura de Madrid Live, hemos dividido éste punto en 3 secciones, una por cada módulo que compone la aplicación, tal y como se muestra en la Figura 3.1.

4.1. Núcleo

El núcleo es la parte más importante, ya que aquí es donde están implementadas todas las funcionalidades de Madrid Live (recomendar plantilla, valorar actividad, etc.). Este módulo está implementado con Java (Sección 2.2.1) y también usamos la librería jCOLIBRI (Sección 2.2.7) para implementar los recomendadores. Además se usan otras librerías que nos permiten implementar funciones para gestionar la base de datos y obtener datos dinámicos de APIs externas (p.e. 11870.com).

El núcleo cumple 3 objetivos principales: gestionar de datos, gestionar los test de preferencias y recomendar. A continuación iremos explicando cada objetivo en los distintos apartados.

4.1.1. Gestión de base de datos

Para que Madrid Live pueda hacer las recomendaciones correctamente necesita guardar distintos datos que luego reutilizará en futuras recomendaciones. Para ello el núcleo esta conectado con una base de datos de MySQL en la cual están implementadas las distintas tablas que se necesitan para guardar los datos.

En esta parte del núcleo hemos diseñado una clase por cada tabla de la base de datos, excepto para las tablas plantilla, plantilla-actividad y plantilla-usuario que al estar relacionadas entre sí se controla desde la clase `PlantillasSQL`. Estas clase contienen los métodos necesarios para poder hacer las distintas operaciones en las respectivas tablas como son insertar, modificar o recuperar distintos registros. Estas funciones son llamadas desde otros métodos de la aplicación, por lo que su funcionalidad es hacer una capa entre la base de datos y el resto del núcleo para hacer más sencillas las distintas consultas.

Además de estas clases que controlan cada tabla está la clase `MySQLConnector`, la cual contiene los datos de conexión con la base de datos (usuario, servidor, puerto, etc.). `MySQLConnector` proporciona la conexión de la base de datos con cada clase que controla su respectiva tabla, por lo que todas estas clases tienen que tener como atributo un elemento de conexión y el objeto `Connection`, para que puedan hacer las consultas SQL a la base de datos. Este objeto `Connection` se obtiene a partir de el objeto `MySQLConnector`.

Hemos aplicado el patrón Singleton para la clase `MySQLConnector`, la cual gestiona la base de datos. El motivo principal de utilizar el patrón es evitar que haya muchas conexiones a la base de datos al mismo tiempo y se quede bloqueada. De hecho sólo hace falta una conexión, ya que el núcleo es el único que necesita tener acceso a la base de datos, con el patrón controlamos que eso ocurra.

Dicho patrón lo que hace es diseñar la clase de tal manera que sólo exista una objeto de esa clase, es decir, durante toda la ejecución solo existe un objeto de `MySQLConnector`. Para aplicar dicho patrón se ha procedido a hacer privada la constructora, por lo que solo es accesible desde la propia clase `MySQLConnector`, y se llama una sola vez cuando se hace la primera referencia a la clase, cuando se hacen mas referencias a dicha clase el objeto que se devuelve es el mismo que se creo en la primera llamada.

Para que las distintas clases obtengan un objeto de la clase `Connection`, en la clase `MySQLConnector` hay un método estático llamado `initConnection` el cual devuelve la conexión a la base de datos con los datos que se han especificado al crear el objeto `MySQLConnector`. Al hacerlo de esta manera conseguimos que solo haya una conexión a la base de datos evitando que se esta se colapse.

4.1.2. Test de Preferencias

Como hemos explicado anteriormente, con los test de preferencias obtenemos los primeros datos para saber los gustos del usuario. El único test que

tiene la necesidad de estar implementado en una clase es el test de Elo, ya que hay que implementar los algoritmos para modificar las puntuaciones de los ranking de los museos y los métodos para ir obteniendo las distintas rondas. El test de preferencias de restaurantes lo controla directamente desde el módulo de interfaz, ajustando el peso a los tipos de restaurante según el orden seleccionado por el usuario y llama a la respectiva función de la Base de Datos para guardar los pesos de los tipos de restaurantes.

La clase que implementa el test de museos es `PreferenciasUsuario`. Esta clase proporciona los métodos necesarios para obtener los museos que hay que mostrar al usuario de una respectiva ronda y modificar los resultados de los pesos según la elección de tipos de museos hecha por el usuario. Para ello nos basamos en dos funciones que son:

- `getRonda`: Este método devuelve un vector con los tipos de museos que hay que mostrar al usuario en la siguiente ronda. Para ello hay que pasar como parámetro el identificador del usuario y la ronda del test en la que se encuentra. Además si la ronda que hay que mostrar es la tercera ronda, hay que pasarle los grupos que han salido en la ronda anterior.
- `puntuaRonda`: Aquí se procede a hacer las actualizaciones de todas las puntuaciones de los tipos de museos para este usuario, modificando el ranking de los gustos de los tipos de museos del usuario. Los parámetros que necesita este método son: el identificador del usuario al cual se le está realizando el test, un vector con los museos que ha seleccionado y otro vector con los museos que no ha seleccionado. Una vez que el método obtiene estos datos se procede a ejecutar el algoritmo de Elo que hemos adaptado para modificar las puntuaciones.

Como para la tercera ronda hay que decir que grupos han salido en la segunda ronda, se proporciona otro método `getGrupo`. A este método se le pasa por parámetro un tipo del museo y devolverá a qué grupo pertenece.

4.1.3. Recomendador

Madrid Live está compuesto por distintos recomendadores con el fin de obtener la plantilla que más pueda satisfacer a cada usuario. Como se explicó en la sección 3.1.2, una plantilla abstracta es una plantilla creada por el sistema o por el usuario indicando en ella los tipos de actividades que desea incluir en su plan y el horario en el que va a transcurrir dicha plantilla. Antes de ver y explicar cada recomendador veremos el funcionamiento general de la recomendación de plantillas (Figura 4.1). Para entender el funcionamiento de los recomendadores hay que explicar qué es el *valor de satisfacción*. Este valor está asociado a la actividad o plantilla que se devuelve como solución y

significa el grado de satisfacción que va a tener dicho usuario con la plantilla. En caso de las actividades este grado se calcula a partir de sus pesos sobre el tipo de actividad y las características de la actividad que se devuelve. En caso de las plantillas, dicho grado se corresponde con la media de grados de satisfacción de cada actividad que componen la plantilla.

Cuando un usuario solicita la recomendación de plantillas el recomendador que recibe dicha solicitud es el *recomendador de plantillas concretas*. Este recomendador busca en las plantillas de otros usuarios, una que pueda satisfacer los gustos del usuario y las restricciones que ha puesto en el formulario. Cuando ha hecho la recomendación se evalúa si la satisfacción de ese usuario con respecto a dicha plantilla es mayor de 0,75. Si cumple los requisitos y el valor es mayor de 0,75 se devuelve esa plantilla como solución, en caso de que no exista una plantilla que cumpla los requisitos del usuario o que el valor de la mejor plantilla sea menor o igual a 0,75 se pide la solución al *recomendador de plantillas abstractas*.

En caso de que no se haya obtenido una solución con el Recomendador de Plantillas Concretas entra en juego el Recomendador de Plantillas Abstractas. Dicho recomendador tiene una base de plantillas abstractas, es decir, plantillas que contienen los tipos de actividades, sin especificar cual es la actividad concreta que debe hacer el usuario. Por ejemplo en la Figura 4.2 podemos ver que la parte superior representa una plantilla abstracta con el horario y el tipo de actividad que tiene asignado. Este recomendador busca dentro de una base de casos las plantillas abstractas que cumplen con las restricciones que solicita el usuario, es decir, horario de la plantilla, tipos de actividades de la plantilla, etc. Una vez que el recomendador obtiene dicha plantilla, la recorre tipo por tipo e invoca al recomendador individual específico. Este último es el que devuelve un resultado en concreto, para cada tipo de actividad, que mayor satisfacción ofrezca a los usuarios que vayan a realizar el plan. Por ejemplo, en la Figura anterior 4.2 podemos ver en la parte inferior cómo cada recomendador individual ha calculado la mejor posibilidad para cada tipo. Por ejemplo: para museos, el Museo del Prado, para restaurantes McDonald y para parques el Retiro.

Además, si la recomendación es para un grupo de personas, es decir, el usuario ha invitado a otros usuarios (ver Figura 4.3) a unirse a su plan, hay que especificar la función de agregación que se va a usar para obtener la mejor actividad para el grupo. Se pueden implementar más tipos de funciones de agregación además de las que hay, todas ellas deben implementar la interfaz "Agregación". Dicha interfaz especifica que toda clase que implemente una función de agregación debe contener dos métodos:

- `addRecomendacion:` Esta función añade la solución de las actividades sugeridas para un usuario, y se añade al conjunto de actividades que se han recomendado ya a los demás usuarios. Para hacer esto, se pasa

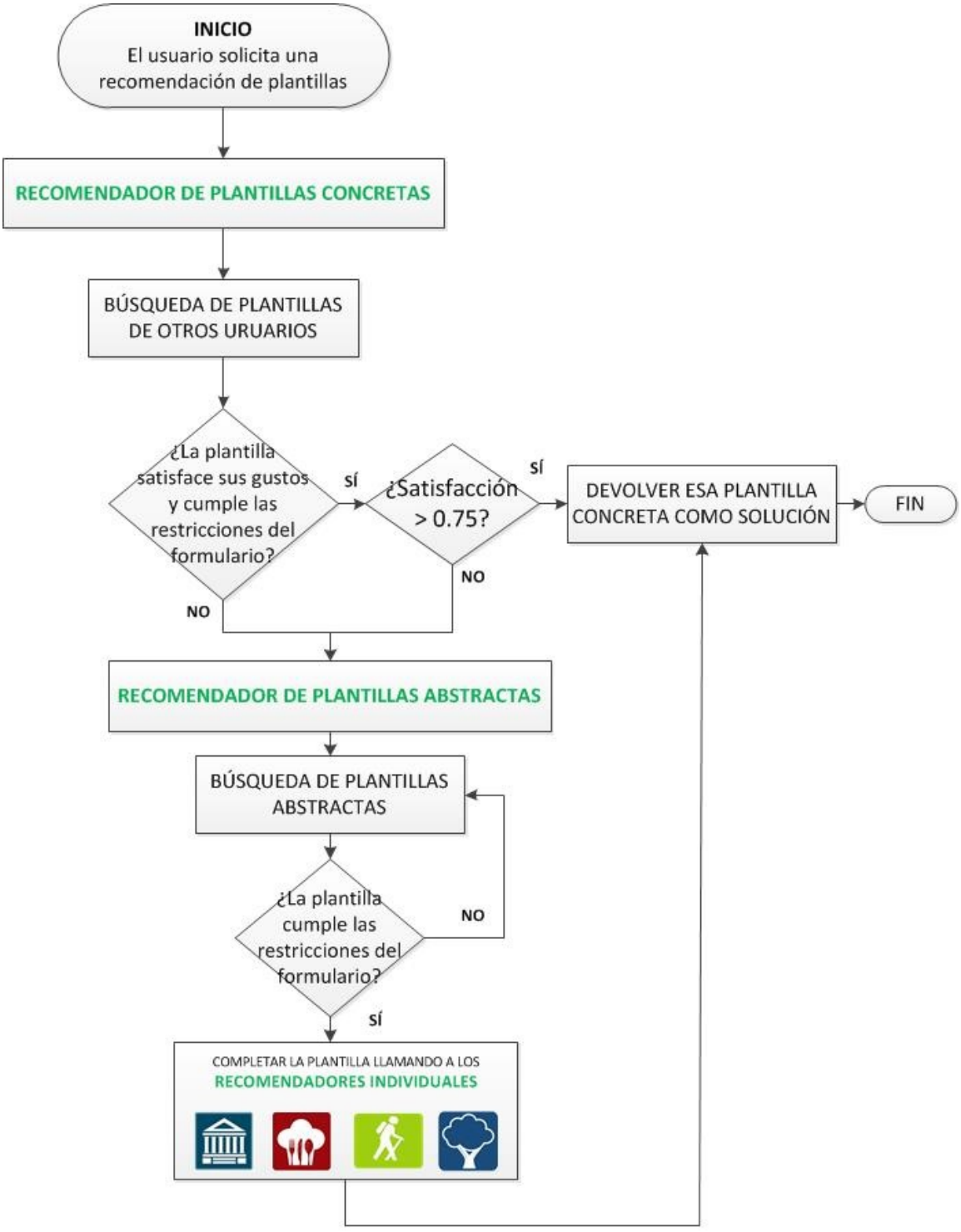

Figura 4.1: Diagrama de flujo normal: Comportamiento del Recomendador

Figura 4.2: Ejemplo recomendador plantilla abstracta.

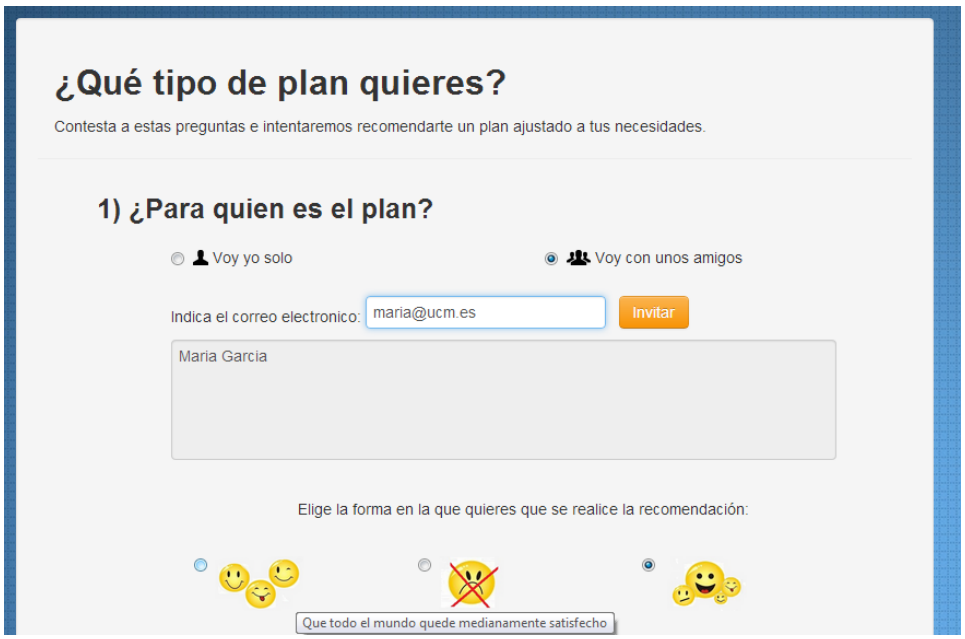

Figura 4.3: Añadir Usuario para plan de grupos

por parámetro la lista con las actividades recomendadas al usuario, y una lista con sus respectivos grados de satisfacción. Los grados de satisfacción servirán para hacer los cálculos de la satisfacción del grupo según los criterios que se usen.

- `getResult`: Esta función devuelve la mejor solución para el grupo después de aplicar la función correspondiente, y será esta actividad la que se devuelva como solución al recomendador de Plantillas Abstractas.

En Madrid Live hemos implementado tres tipos de función de agregación. Explicaremos en qué se basan para devolver la solución:

- ***Media:*** Cada vez que se agrega una actividad al conjunto de actividades recomendadas se va calculando la media, es decir se suma la satisfacción de cada usuario a cada actividad y se divide por el número total de actividades. Una vez que ya se han añadido todas las actividades se devuelve la que tenga la mayor media. Con esta función conseguimos que todos los usuarios queden satisfechos de la manera más igualada posible.

- ***Mínima Miseria:*** En este caso buscamos la actividad que menos disguste al grupo de usuarios. Para ello cada vez que añadimos una actividad al conjunto, guardamos el grado de satisfacción del usuario que menos satisfacción tenga en esa actividad. Cuando ya hemos añadido todas las actividades cogemos la actividad con el máximo de los grados de satisfacción guardados y se devuelve como solución. El objetivo de esta agregación es que ningún usuario quede totalmente descontento con el plan.

- ***Máximo:*** Esta función de agregación guarda el grado de satisfacción del usuario que tenga la máxima satisfacción por esa actividad. Una vez que hemos añadido todas las actividades devolvemos la actividad que tenga el mayor grado de satisfacción guardado. Ahora lo que se consigue es que un usuario en concreto este lo más satisfecho posible sin tener en cuenta el resto. Esto puede ser útil si este usuario es el más influyente del grupo.

A continuación explicaremos cómo ha sido la implementación de cada recomendador. Por cada uno de ellos explicaremos cómo obtiene los datos y cómo recupera la solución.

4.1.3.1. Recomendador Plantillas Concretas

Como hemos explicado anteriormente éste recomendador es el encargado de ver si existe alguna plantilla realizada por otro usuario que pueda cumplir

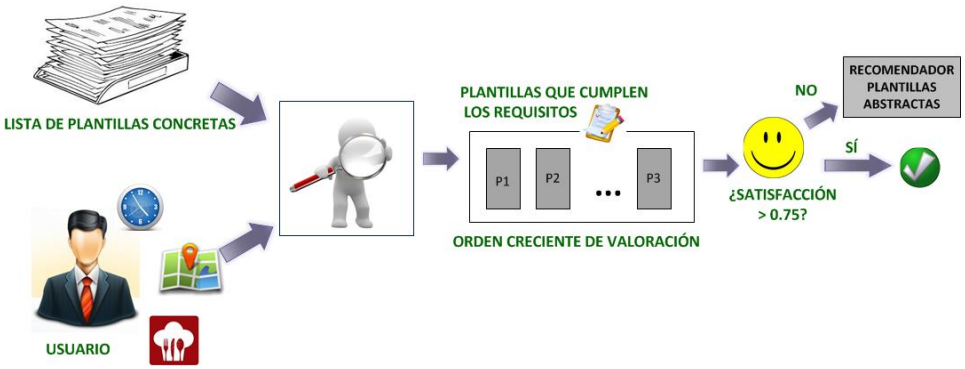

Figura 4.4: Esquema de Recuperación de la Solución en el Recomendador de Plantillas Concretas.

las restricciones y satisfacciones de éste. Es el primer recomendador que entra en funcionamiento y el que (si no encuentra la solución) se encarga de llamar al resto de recomendadores.

- Obtención de datos:
  Para obtener los casos en los cuales tiene que buscar la solución, este recomendador busca dentro de la base de datos todas las plantillas realizadas por otros usuarios. Para poder hacer esto y siguiendo la plantilla de recomendadores de jCOLIBRI se ha creado una clase que implementa la interfaz `Connector` de jCOLIBRI. La clase en cuestión es PlantillasConnector y se encarga de obtener los datos de la base de datos y crear objetos de las clases `PlantillaDescription` y `PlantillaSolution`. Una vez que tiene todas las plantillas se pasan a la base de casos del recomendador para que pueda ejecutar el algoritmo de CBR.

- Recuperación de Solución:
  La búsqueda de una plantilla concreta de otro usuario (Figura 4.4) se realiza en 2 fases: cumplimiento de las restricciones de la plantilla y satisfacción del usuario con esa plantilla. En la primera fase creamos una consulta a la base de casos con las restricciones que el usuario ha puesto (horario, localización y tipos actividades). Una vez que hemos obtenido dichas plantillas cogemos las que consideramos "candidatas" a ser solución, para ello cogemos aquellas plantillas cuyo indice de semejanza con la consulta sea del 0,90 o mayor. Además, si el usuario que va a hacer el plan ha hecho esa plantilla hace menos de 6 meses, esa plantilla se desecha. En caso de que no se encuentre ninguna plantilla "candidata", se pide la consulta al Recomendador de Plantillas Abstractas.

En caso de que exista alguna plantilla candidata, se pasa a la siguiente fase. Lo que se va a hacer es recorrer las actividades concretas que incluye el plan, y se va viendo que satisfacción tendrá el usuario con respecto a cada actividad del plan. Una vez que se ha calculado la satisfacción del usuario con cada actividad del plan se calcula la satisfacción del plan. Ésta se calcula con la media de satisfacciones de las actividades del plan. Cogeríamos la plantilla con la máxima satisfacción para el usuario, y si esta cumple que la satisfacción del plan para el usuario es de 0,75 o más se devuelve como solución, en caso contrario, se envía la consulta al Recomendador de Plantillas Abstractas.

Si la solicitud de recomendación es para un grupo de personas, se hacen las mismas fases. La primera fase es idéntica a la explicada anteriormente, ya que las restricciones son idénticas para todo el grupo de personas, es decir, el horario de la plantilla no cambia por cada usuario. La única diferencia es que en caso de alguno de los usuarios que participan en el plan haya realizado alguna de las plantillas "candidatas", ésta se desechara del grupo de candidatas.

La segunda fase sufre alguna modificación. Se calculan las satisfacciones de cada usuario con cada plantilla "candidata" como se ha explicado anteriormente. Y cada vez que se calcule esta satisfacción, se inserta en la función de agregación correspondiente. Una vez que se haya hecho ésto para cada usuario y cada plantilla, se le pide a la función de agregación que devuelva la mejor plantilla según el tipo de función de agregación seleccionado.

4.1.3.2. Recomendador Plantillas Abstractas

Este recomendador entrará en juego si el anterior no ha encontrado ninguna plantilla concreta en la base de datos de Madrid Live lo suficientemente adecuada para ese usuario. En este caso el objetivo de este recomendador es devolver una solución más especifica al usuario rellenando con actividades concretas de su agrado una plantilla abstracta que cumpla con las restricciones.

- Obtención de datos:
  Al ser otro recomendador basado en jCOLIBRI, necesita como el Recomendador de Plantillas Concretas, una clase que le devuelva todos los elementos de la base de casos. La clase `PlantillasAbstractasConnector` (que implementa la interfaz `Connector` de jCOLIBRI) obtiene de un archivo de texto todas la permutaciones de

Figura 4.5: Recuperación Solución Recomendador Plantillas Abstractas.

tipos de actividades que puede haber en un plan.

Carga todas estas plantillas en objetos de la clase `PlantillaDescription` y `PlantillaSolution`. Una vez que ha conseguido cargar del texto todas las plantillas abstractas las envía a la base de casos del recomendador.

- Recuperación de Solución:
  Como se ve en la Figura 4.5, este recomendador busca primero la plantilla abstracta que cumple con las restricciones que el usuario ha rellenado en el formulario, y devuelve la plantilla que más se asemeja a estas condiciones. Como el Recomendador de Plantillas Abstractas se basa en un recomendador CBR, este proceso se ejecuta en el ciclo del recomendador.

  Una vez que ha obtenido la plantilla que cumpla con las restricciones del usuario, el recomendador procede a adaptar la solución de la plantilla, esto es, a ir asignando una actividad concreta a cada tipo de actividad del plan. Para ello en cada tipo de actividad llama a cada recomendado individual pasándole las restricciones contextuales que tiene que tener en cuenta el recomendador y para que usuario es el plan. Una vez que ha asignado todas las actividades devuelve la plantilla al usuario.

Con esto tendríamos los recomendadores de plantillas. A continuación, entran en juego los recomendadores individuales. Estos recomendadores son específicos para un tipo de actividad concreta, por ejemplo, recomendador de actividades para museos. La aplicación está diseñada para que se puedan añadir más recomendadores sin que se tenga que modificar toda la aplicación, lo único que hay que hacer es diseñar el

recomendador individual para esa actividad que se quiere añadir y que dicho recomendador implemente la interfaz `RecomendadorIndividual`.

La interfaz `RecomendadorIndividual` obliga a implementar una función, la cual tiene que estar presente en el recomendador individual, ya que es la función que solicitará al Recomendador de Plantillas Abstractas para obtener las actividades de ese tipo de actividad. La función que deben implementar es recomendar. A esta función hay que pasarle por parámetro los identificadores de los usuarios que van a realizar el plan, la información del Contexto y la función de agregación (Sección 2.1.2) que se va a aplicar en caso de que se vaya a recomendar a un grupo de personas. Esta función devolverá una lista con las actividades concretas que recomienda, esta solución se devuelve a traves de una lista de objetos de `ActividadConcreta`, los cuales contienen todos los datos significativos de las actividades (nombre, dirección, web, etc.).

Como hemos visto, a los recomendadores individuales, necesitamos proporcionarles información del contexto, de esta forma, nos aseguraremos de que la actividad que nos devuelva cumpla las condiciones de localización, horario y lista de actividades abstractas. Esta información se pasa a través de un objeto del tipo `ContextInfo`, el cual contiene el horario que debe cumplir dicha actividad y la localización que tiene como referencia (puede ser la de la actividad anterior o la que especifique el usuario). Con estos datos conseguimos que cada actividad tenga una relación resto de ellas del plan basándonos en la localización.

Una vez que ya hemos explicado cuales son las clases que necesitan los recomendadores individuales, vamos a explicar como están implementados los distintos recomendadores.

4.1.3.3. Recomendador de Museos

Este recomendador busca los museos que pueden gustar más al usuario, partiendo de los pesos que tiene el usuario para cada tipo de museo.

- Obtención de datos:
  Este es otro recomendador que se basa en un recomendador de jCOLIBRI, pero en este caso usamos el `Connector` de base de datos que viene implementado en jCOLIBRI (`DataBaseConnector`). Con eso obtenemos el catálogo de museos que podemos recomendar. Estos museos están categorizados por tipos como por ejemplo (pintura, escultura, ciencia, etc.) y es con esos datos con los que

se obtiene la recomendación más adecuada.

Los datos de los museos que hay en la base de datos los cargamos en objetos de las clases `MuseoDescription` y `MuseoSolution`. Una vez que se hayan cargado todos los museos de la base de datos, se guardan en una lista para añadirlos a la base de casos del recomendador, el cual ya podrá ejecutar las acciones necesarias para obtener el museo que más grado de satisfacción tenga para el usuario.

- Recuperación de Solución:
  El recomendador busca los museos que estén categorizados por los tipos que le gusten a cada usuario (pintura, historia, ciencia...), evitando así que haya exposiciones que no le gusten. Además, si el museo debe de estar cerca de algún punto geográfico, se añade a la consulta y dándole a la posición más prioridad que a los tipos. Con eso conseguimos que devuelva el museo más cercano y que cumpla con las preferencias del usuario.

  Una vez que se ha ejecutado el ciclo, se filtran los resultados que ya están en el historial para evitar que el usuario visite un museo que ha visitado recientemente. El filtro consiste en eliminar del resultado los museos que ha visitado hace 6 meses o menos. Una vez que se han filtrado los resultados, se ordenan los museos de mayor a menor satisfacción del usuario y se devuelve el museo que mayor satisfacción tenga.

  En caso de que se esté buscando un museo para un grupo, en vez de devolver un único museo como resultado, se devuelve una lista con los 10 museos con mejor grado de satisfacción y se envían a la función de agregación correspondiente. Una vez que se ha realizado este proceso por cada usuario, se llama a la función `getResultado` de la función de agregación y se devuelve este museo como solución para el grupo.

4.1.3.4. Recomendador de Restaurantes

Aquí buscamos el restaurante, más cercano a la posición que nos indique el usuario, que más se adapte a sus gustos, los cuales obtenemos de su test específico de preferencias.

- Obtención de datos:
  Este es un recomendador dinámico, es decir, no tenemos los datos en la base de datos del sistema, sino que hay que obtenerlos de la

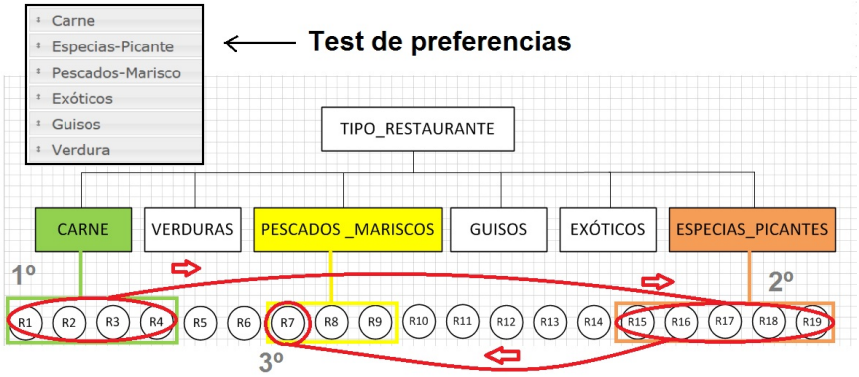

Figura 4.6: Ejemplo de selección de restaurantes

API de 11870.com. Esto se debe a que hay muchos restaurantes en la ciudad de Madrid y ocuparía mucho espacio en la base de datos.

Cuando se va a hacer una recomendación de restaurante al recomendador se le indica la posición desde la que se encuentra el usuario o donde se encuentra la actividad anterior de la plantilla. Para obtener el catálogo de restaurantes se obtienen los restaurantes que hay en el radio de 1 kilómetro desde la posición que nos han indicado y se cargan todos esos restaurantes en el recomendador.

- Recuperación de Solución:
  Los restaurantes que nos hemos descargado se meten en la ontología como instancias de ella, la podemos ver en detalle en el apéndice A. Esto nos permite clasificar los restaurantes por los tipos de comida que sirven. Una vez que hemos clasificado todos los restaurantes procedemos a extraer los restaurantes que más le gusten al usuario, es decir, el tipo de restaurante que mayor peso tenga para el usuario.

  Si con ese tipo de restaurante ya hemos obtenido 10 restaurantes, los devolvemos como solución, si no es así procedemos a relajar la condición y buscamos el segundo tipo de restaurantes con mayor peso para el usuario. Procedemos así sucesivamente hasta que obtenemos 10 restaurantes. En caso de que el recomendador sea para una única persona se devuelve el primer restaurante de la lista, ya que este será el que mejor grado de satisfacción tenga.

  En la Figura 4.6 se puede ver un claro ejemplo de cómo se relajan las restricciones a la hora de seleccionar los 10 restaurantes más

adecuados para cada usuario. En este caso, tras realizar el test de preferencias, el usuario sitúa la carne como su tipo de comida favorita, seguido de las especias y picantes, pescado y marisco, exóticos, guisos y por último, la que menos le gusta, la verdura.

Para escoger los 10 restaurantes más adecuados, seleccionamos primero todos aquellos que tengan la etiqueta "carne" dentro de nuestra ontología. Como en este caso tan sólo encontramos 4, necesitamos completar los 6 restantes con restaurantes de su segundo tipo favorito, en este caso, especias y picantes. Tras coger los 5 restaurantes existentes de este grupo, nos damos cuenta de que nos falta tan sólo 1 para completar la lista. Tal y como hicimos en el paso anterior, volvemos a relajar restricciones y cogemos el restaurante que nos queda del grupo de pescados y mariscos (situado en el tercer puesto de su ranking de preferencias).

La adaptación de la solución para el Recomendador de Restaurantes es la misma que la del Recomendador de Museos. Por cada usuario obtenemos una lista de 10 restaurantes, y se van agregando al conjunto de recomendaciones con la función de agregación correspondiente. Una vez que se ha terminado de agregar recomendaciones al conjunto, se llama a la función de agregación para que devuelva el restaurante que mejor se adapte al grupo según la implementación de la función de agregación escogida.

4.1.3.5. Recomendador de Parques

El parque viene determinado por la localización del usuario y la valoración que tiene el parque para todos los usuarios que lo han visitado.

- Obtención de datos:
  El Recomendador de Parques entra dentro de la categoría de recomendador estático, es decir, que el catálogo de parques que vamos a mostrar está dentro de la base de datos del sistema. Además es un recomendador basado en jCOLIBRI, por lo que usamos la clase `Connector` que viene por defecto.

  Como en los museos, cargamos la información de los parques en objetos de las clases `ParqueDescription` y `ParqueSolution`. Una vez que el sistema ha cargado en la base de casos toda la lista de parques que hay en la base de datos, se procede a obtener la recomendación del parque.

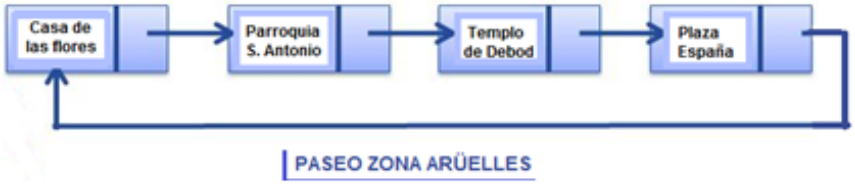

Figura 4.7: Ejemplo paseo circular.

- Recuperación de Solución:
  El parque que buscamos será uno que esté cerca de donde se encuentra el usuario y de entre los más cercanos, el más valorado por los usuarios. Para hacer la consulta se mete la localización del usuario o de la actividad anterior y se hace la búsqueda devolviendo el parque que mejor cumpla las condiciones.

  Para hacer una recomendación de un parque para grupos hacemos que dependa de la valoración que han dado otros usuarios al parque, por lo para un grupo de personas se busca el parque mejor valorado de los que tienen más cerca de su posición.

4.1.3.6. Recomendador de Paseos

Los paseos están compuestos por una lista circular de monumentos. Cada lista pertenece a una zona de Madrid diferente y por cada monumento tenemos la siguiente información: cuál es el punto siguiente a visitar y el tiempo que se tarda en llegar. Se puede ver en la Figura 4.7 la estructurada de un paseo como ejemplo.

- Obtención de datos:
  Los paseos están guardados en la base de datos. El recomendador indica un punto geográfico en el que se encuentran y se devuelve el paseo más cercano, es decir, el monumento más cercano con su posición en la lista, y la lista completa de ese paseo.
- Recuperación de Solución:
  Cuando se solicita la recomendación de un paseo, el recomendador solo tiene que pedir el punto geográfico donde se encuentra el usuario o la actividad anterior. Una vez que obtiene la lista añade monumentos a la solución de ese paseo siguiendo el orden de la lista de monumentos de ese paseo. El recomendador deja de insertar monumentos a ese paseo cuando ya se ha añadido todos los monumentos de la lista o se ha completado el tiempo que la plantilla ha asignado al paseo.

4.2. Servicios

Uno de los objetivos que queríamos conseguir es que Madrid Live fuera una aplicación multiplataforma, es decir, que sea accesible desde aplicaciones para PC, aplicaciones móviles, etc. Para ello se ha implementado una capa que proporciona funciones para obtener los distintos datos de Madrid Live y poder hacer recomendaciones desde, por ejemplo, un móvil Android.

A esta capa la llamamos la capa Servicios o API y ha sido implementada con J2EE. La API está compuesta por distintos servlets, cada uno de ellos proporciona un servicio o función. Para poder interactuar debe pasarse por url el nombre del servlet, por ejemplo, MadridLive/-login.do, e ir añadiendo los parámetros después de dicho nombre, por ejemplo, MadridLive/login.do?usuario=user&password=contraseña.

Cuando se ha solicitado dicho servicio, la API contestará en formato JSON 2.2.6, con la respuesta de la aplicación. Esta contestación puede ser un error (parámetros incorrectos, etc.), una respuesta de confirmación de que se ha realizdo la acción o un conjunto de datos obtenidos en caso de que se haya llamado a un servlet de consulta.

Los distintos servicios que se han implementado para Madrid Live se encuentran en el Apéndice C, donde se pone la url que hay que utilizar para cada servlet, los parámetros necesarios y el JSON que devuelve como solución.

4.3. Interfaz

Por último, hemos diseñado una interfaz para que los distintos usuarios puedan acceder a los servicios que ofrece Madrid Live. Para hacer dicha interfaz hemos optado por una aplicación web. Aunque como hemos explicado antes, la interfaz web no es la única posible a implementar, pero es la que creemos que es más visual.

Nuestra interfaz web (ver Figura 4.8) esta implementada con HTML5 y jQuery. Al usar HTML5, aunque es un lenguaje que aún no está estandarizado, proporciona distintos elementos que facilitan la programación, por ejemplo, los campos obligatorios en un formulario, y añaden nuevos tipos de inputs que ahorran trabajo a la hora del diseño.

Para que nuestra página web interactue con la API de Madrid Live y se convierta en la interfaz de ésta, hemos usado jQuery para comunicarse con las distintas llamadas a los servlet y el tratamiento de los JSON. Además al hacerlo de esta manera, todo este proceso queda camuflado para el usuario, y lo único que ve es que se le muestran los resultados directamente en la web.

A continuación, iremos ilustrando los diferentes servicios que ofrece nuestra interfaz.

Página Principal (Figura 4.8): Muestra la página principal de Madrid Live. En ella se encuentran los formularios de identificación (para usuarios registrados en la aplicación) y registro (para nuevos usuarios).

Test de preferencias (Figura 4.9): En ella se observa el test de preferencias inicial que todo usuario debe rellenar una vez se haya registrado. A través de este test conoceremos sus gustos y preferencias sobre los distintos tipos de museos y restaurantes.

Formulario de obtención de planes (Figura 4.10): Ilustra el formulario de obtención de planes. En él, el usuario especificará con quién irá, en qué horario, desde qué zona quiere comenzar y las distintas actividades que desea incluir en su plan.

Barra de progreso (Figura 4.11): Muestra la barra de progreso. Durante este tiempo la aplicación busca el plan más idóneo para el usuario, asegurándose de que cumpla las restricciones especificadas por el mismo en el formulario anterior.

Plan propuesto (Figura 4.12): En ella se puede observar el plan propuesto por la aplicación. Si nos fijamos, veremos que se han propuesto 4 actividades distintas y para cada una de ellas se han mostrado datos de interés (como pueden ser: el nombre, una fotografía, la dirección, la página Web e incluso una breve descripción del lugar).

Valoración de actividades (Figura 4.13): Muestra el procedimiento que debe seguir el usuario para valorar actividades ya realizadas. Tal y como se observa, ha de valorar cada actividad con una puntuación entre el 0 y el 5. A través de este valor, aprenderemos, y por tanto mejoraremos, las futuras recomendaciones para dicho

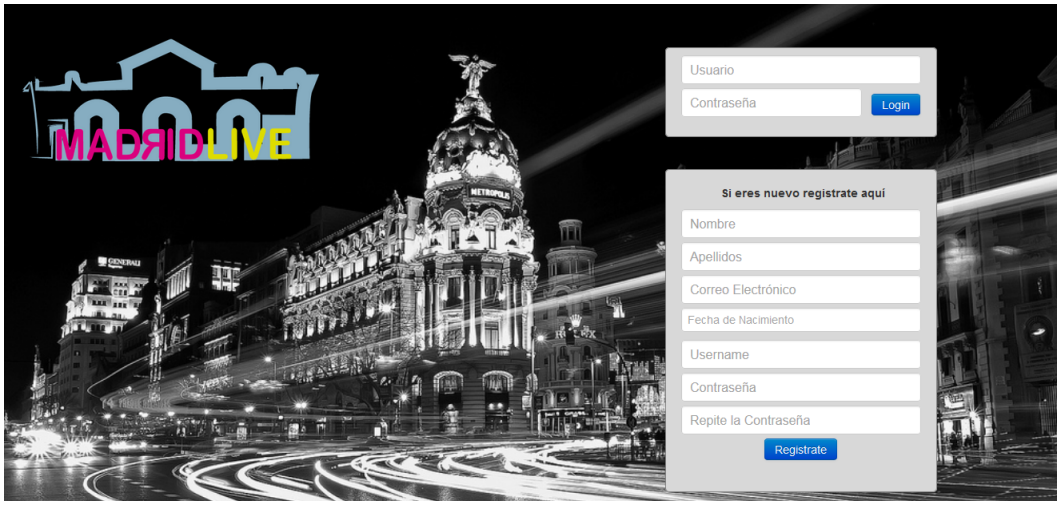

Figura 4.8: Página principal de Madrid Live

usuario.

Modificar perfil (Figura 4.14): La aplicación ofrece la posibilidad de cambiar algunos de nuestros datos personales presentes en nuestro perfil de usuario, como son: el correo electrónico o la contraseña de acceso.

Mejorar preferencias (Figura 4.15): Madrid Live permite volver a realizar los test de preferencias. Los gustos de un usuario pueden variar, por lo que consideremos importante que puedan modificar su ranking de preferencias tanto para los tipos de museos, como para los distintos tipos de restaurantes.

Figura 4.9: Test de preferencias inicial

Figura 4.10: Formulario para la obtención de planes.

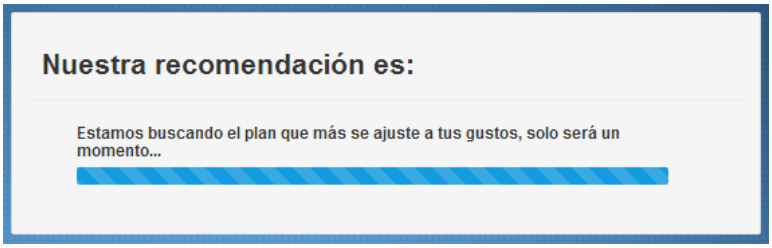

Figura 4.11: Búsqueda del plan más idóneo para el usuario

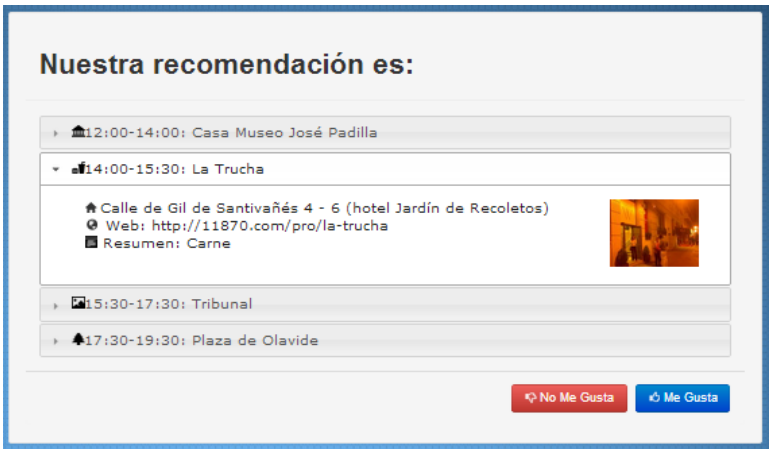

Figura 4.12: Plan propuesto por Madrid Live

Figura 4.13: Valoración de cada actividad realizada.

Figura 4.14: Modificación del perfil de usuario.

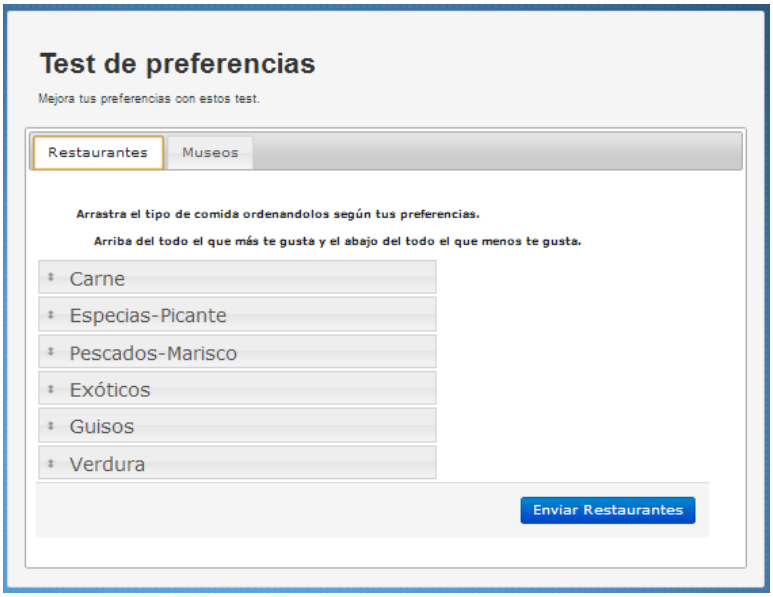

Figura 4.15: Mejorar las preferencias del usuario.

Capítulo 5

Conclusiones. Lineas de Trabajo Futuro

En este proyecto hemos presentado una aplicación web, Madrid Live, que consiste en un sistema recomendador de actividades de ocio en Madrid para individuos y grupos. Nuestra principal aportación es la implementación de un servicio web que contiene un catálogo variado de posibles actividades en nuestra ciudad y que al usuario le puede resultar muy útil a la hora de decidir qué quiere visitar en la ciudad. Con unos sencillos test descubrimos sus gustos y ajustamos la recomendación resultante lo más posible para sugerirle el plan de actividades más atractivo para él. También tenemos en cuenta factores del contexto, como por ejemplo: las restricciones temporales, la distancia de los lugares a visitar, etc.

Madrid Live es un prototipo fácilmente extensible debido a su diseño modular. Con mínimas modificaciones podría ser adaptado a otras interfaces, como por ejemplo móvil, adaptando el tamaño de las interfaces actuales a un dispositivo móvil. Otra posible extensión sería la inclusión de otros tipos de actividades en el recomendador de planes. Esto sería una tarea fácil ya que el prototipo esta preparado para ello con la capa API.

La aplicación ha sido desplegada en un servidor público[1]. Se ha hecho una fase de pruebas en la que se han registrado alrededor de 100 usuarios, los cuales han realizado cada una de las funcionalidades que se han desarrollado. Además, se ha grabado un vídeo tutorial de la aplicación para enseñar a los usuarios a usarla, este vídeo es accesible

[1] http://ssii13.fdi.ucm.es

desde la aplicación de Youtube[2].

A continuación repasaremos los objetivos que se habían planteado para esta aplicación (ver 1.1) y analizaremos como se han conseguido cumplir.

- Con la implementación de los perfiles de usuario hemos conseguido sugerir planes con actividades que satisfagan lo más posible las preferencias de los usuarios.
- Para dar variedad a las recomendaciones del usuario y evitar repeticiones, se ha añadido un historial de las actividades que el usuario ya ha realizado. Con ello conseguimos que el usuario no realice la misma actividad en un corto periodo de tiempo y que valore las actividades, permitiendo modificar su perfil y ajustar mejor las futuras recomendaciones, además de usarlas para filtrar las recomendaciones de otros usuarios.
- Hemos cerrado el sistema de recomendaciones añadiendo el factor grupal a la recomendación. Un usuario puede pedir una recomendación para un grupo de personas y el sistema consulta las preferencias de todos ellos para elegir el plan que más se adapta al grupo.
- Utilizando el sistema de puntuación de Elo y la ordenación de los tipos de restaurantes hemos conseguido recoger las preferencias del usuario de una manera sencilla y amena para el usuario.
- Utilizando la escala Likert conseguimos que los usuarios puedan valorar las actividades de la recomendación sugerida, y así las futuras recomendaciones se adaptaran mejor a su perfil.
- Para que nuestra aplicación pueda hacer buenas recomendaciones, es importante tener el catálogo de actividades siempre actualizado. Este objetivo se ha conseguido con la conexión del sistema en tiempo real con distintas APIs (11870.com, Google Places).
- Los usuarios que han realizado la fase de pruebas han destacado que la interfaz Web es muy intuitiva y fácil de usar.
- Destacamos el diseño modular que hemos mantenido para facilitar la posible ampliación del proyecto. La arquitectura está diseñada con diferentes capas que separan la lógica de la interfaz, permitiendo que la modificación o extensión de alguna de las partes no afecte al resto del proyecto.

Destacamos también que para conseguir estas recomendaciones resultantes hemos intentado usar diferentes métodos, como sistemas CBR,

[2]http://www.youtube.com/watch?v=r2IuPNcsAP4

ontologías, etc. para desarrollar un proyecto completo en diferentes técnicas de sistemas de recomendación.

5.1. Lineas de trabajo futuro

Nuestro trabajo esta diseñado de manera bastante modular con el propósito de facilitar la posible futura inclusión de nuevos tipos de actividades diseñando su recomendador individual e integrándolo en el sistema de generación de planes ya implementado. También se podría aumentar el dominio del sistema añadiendo otras ciudades de España. Esto no sería más que ampliar el catálogo de información con las actividades de la nueva ciudad y añadirlas a la base de casos.

El sistema ofrece recomendaciones para grupos de usuarios registrados previamente en la aplicación. Realizamos esta recomendación con la información que obtenemos en el test de preferencias del registro y con el feedback que pueden realizar después de recibir una recomendación. Esta área podría ser mejorada si se integrará la aplicación en una red social ya que se podría obtener muchos más datos de los usuarios del grupo y realizar una recomendación más ajustada para todos los integrantes, como por ejemplo: saber que relación tienen los usuarios entre sí, quien posee más liderazgo, etc.

Otra posible línea de trabajo futuro es la adaptación de la aplicación para asociarlo a otro tipo de interfaces como por ejemplo móvil. Esto no sería una tarea complicada ya que nuestra capa de servicios API implementada con Web Service permite a cualquier dispositivo obtener información del sistema.

Las rutas turísticas de nuestro sistema son paseos recorriendo monumentos que están predefinidos en el sistema y no pueden cambiar. Solamente será posible acortar el paseo debido a que es de mayor duración a la deseada por el usuario. Una posible mejora sería utilizar algoritmos de recorrido de grafos estableciendo los monumentos como los nodos y generar los paseos de manera dinámica teniendo en cuenta la distancia entre ellos y la valoración del usuario a esos monumentos.

Apéndice A

Ontología

En la ontología clasificamos los restaurantes que nos descargamos de la API 11870.com para poder hacer la recomendación al usuario. Cada restaurante pertenece a un tipo de restaurante, es decir, es una instancia de la clase Tipo_Cocina. Además cada restaurante tiene como propiedades la Ocasión y los Servicios.

A.1. Tipo de cocina

Cada restaurante que descargamos tiene un tipo de cocina. Este tipos de cocina es una etiqueta de la API 11870.com. En la ontología hemos hecho una clasificación de todos los tipos de cocina que nos proporciona la API en 7 grupos básicos. Los grupos son los que se le dice al usuario que ordene al hacer el test de tipos de restaurante, y tienen como subclases los tipos de cocina que tienen como propiedad principal la del grupo. La clasificación de los tipos de cocina queda como se ve en la figura A.1.

A.2. Ocasión

La ocasión es una propiedad que también nos proporciona la API de 11870.com. Clasifica los restaurantes en función del momento más adecuado para ir a comer allí. La clasificación se puede ver en la figura A.2.

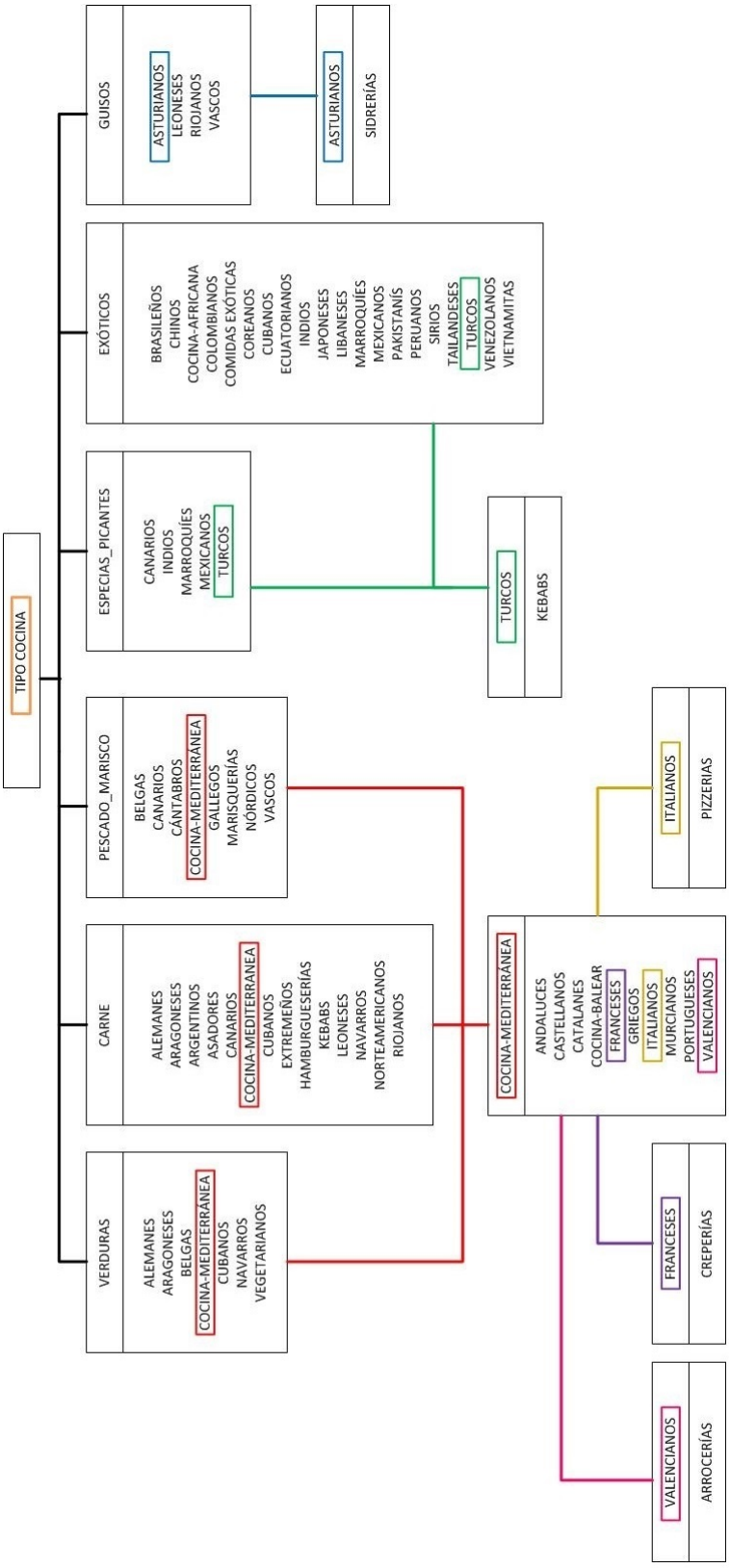

Figura A.1: Ontología: Tipo de cocina

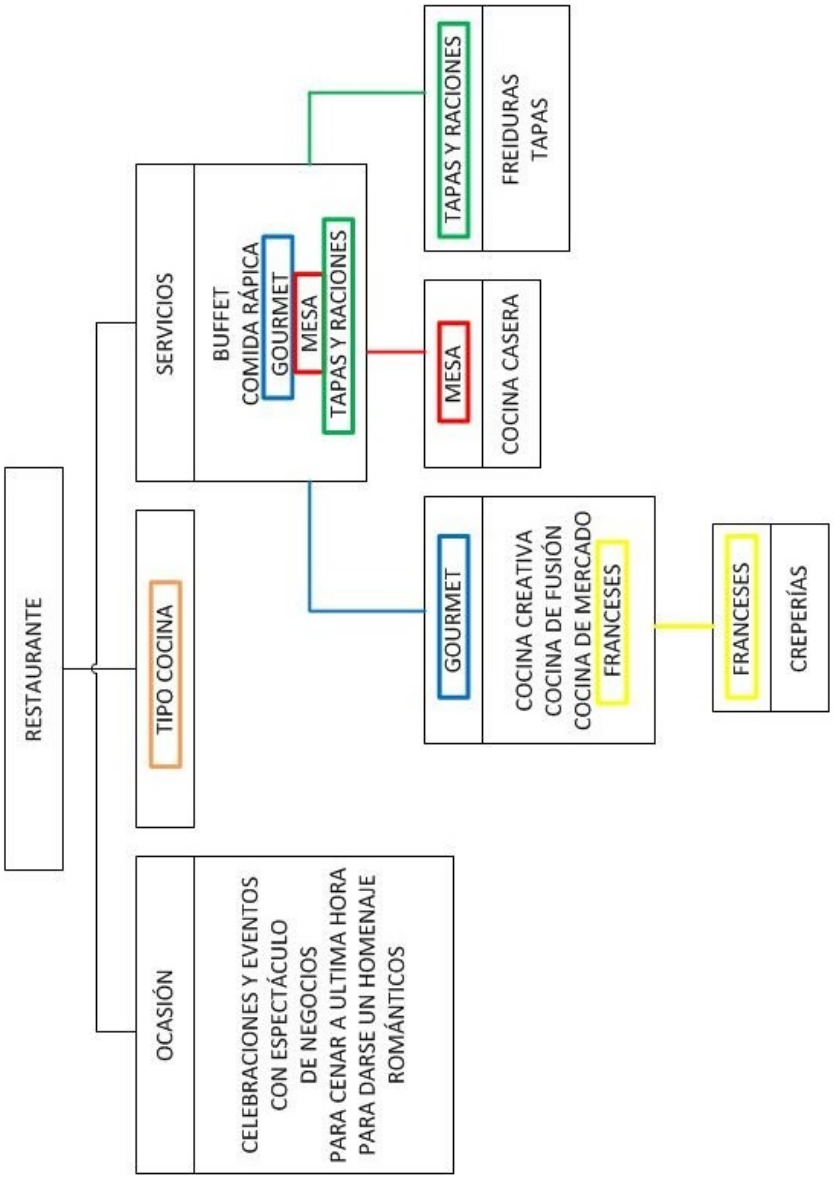

Figura A.2: Ontología: Restaurante

A.3. Servicios

Éstos son servicios adicionales que ofrece los restaurantes. Como los 2 grupos anteriores, estos servicios son etiquetas proporcionadas por la API de 11870.com y que están clasificados de la siguiente manera (Figura A.2).

Apéndice B

Base de Datos

En este apartado explicaremos como está implementada la base de datos del sistema. Comentaremos las tablas que componen dicha base de datos y su funcionalidad en el sistema. Para poder explicarlo dividiremos las tablas según su función: datos de usuario, catálogo de los recomendadores y plantillas de los usuarios.

B.1. Datos de los usuarios

Estas tablas guardan la información que necesitamos del usuario. Entre esa información se guarda los datos personales del usuario y los pesos de los de cada usuario con respecto los tipos de restaurantes y los tipos de museos. Las tablas que pertenecen a este grupo, ver figura B.1, son:

- `usuarios`: Esta tabla guarda los datos personales de cada usuario. Además cada usuario tendrá un identificador único con el que podremos asociar otros registros con ese usuario. La única condición que se debe cumplir para cada usuario es que el username, nombre de usuario con el que se registra en la aplicación, debe ser distinto al de cualquier otro usuario.
- `pesosmuseos`: Aquí guardamos las preferencias del usuario sobre cada tipo de museo. Se añade un registro por cada usuario con los resultados del test de Elo.
- `pesosrestaurantes`: El resultado del test de tipos de restaurantes que se le ha hecho al usuario, se guarda en esta tabla. Como en la tabla anterior cada usuario tiene un único registro para guardar sus preferencias sobre restaurantes.

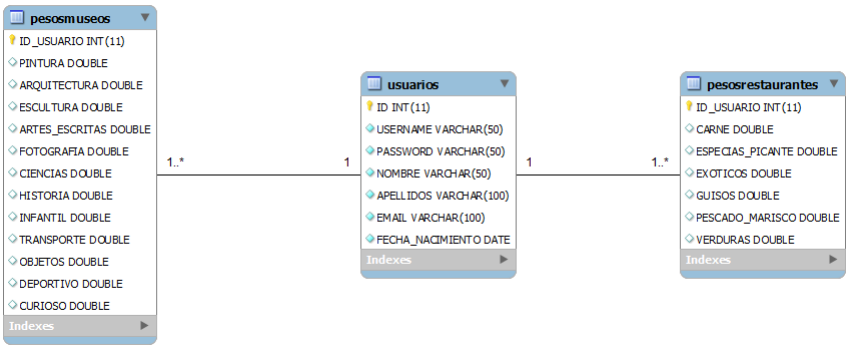

Figura B.1: Tablas de información de usuarios.

B.2. Catálogo de las recomendaciones

Como hemos explicado antes (ver 3.2.1), tenemos un catálogo estático cuyos registros se guardan en la base de datos. En este caso los datos que guardamos son los museos, parques y paseos (ver figura B.2). A continuación vemos las distintas tablas de catálogo:

- `museos`: En esta tabla guardamos la información principal de los museos (como son su dirección, página web, imagen, etc.) y los tipos de exposiciones que ofrece. Esos tipos de exposiciones no permite recomedar a un usuario un museo u otro.
- `parques`: Aquí se guardan los datos generales de los parques y las valoraciones totales de los usuario, con el número de usuarios que han visitado cada paque, para poder calcular la média. Con esa media conseguimos devolver el parque más valorado por los usuarios.
- `monumentos`: Esta tabla es la primera de las 3 que componen los datos referidos a paseos. Aquí tenemos toda la lista de monumentos que componen todos los paseos, con su información general. Estos son los nodos de la lista enlazada que forman los paseos.
- `paseos`: La tabla paseos define cada paseo dandole un nombre y un identificado único. Además guardamos los datos estadisticos de valoraciones y visitas de cada paseo.
- `paseos_monumentos`: La funcionalidad de esta tabla es relacionar a cada monumento, el paseo que le corresponde. Para ello en cada registro se tiene el identificador de monumento, el identificador del paseo al que pertenece, la posición que ocupa dentro del paseo y el tiempo que se tarda en llegar a ese monumento desde el anterior.

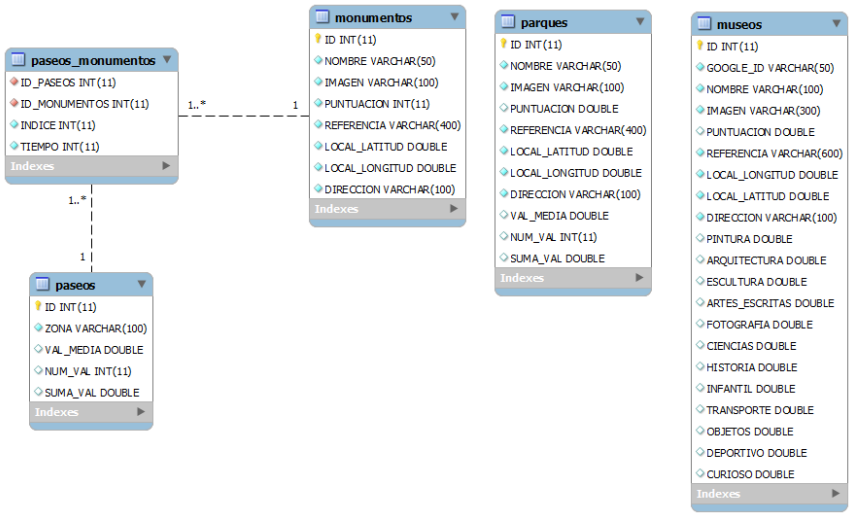

Figura B.2: Tablas de catálogo de recomendación.

B.3. Plantillas de usuarios

Todas las plantillas de los usuarios se guardan en la base de datos, ya que ésto nos permite que podemos devolver esas plantillas a otros usuarios. Las tablas que componen esta parte son las siguientes (ver figura B.3):

- `historial`: En esta tabla se guarda cada actividad que ha realizado un usuario, indicando que usuario lo ha hecho, el tipo de actividad, la información principal de dicha actividad y la valoración que le ha dado el usuario. Además guardamos la fecha de recomendación, eso nos permite hacer un filtro de algunas recomendaciones para haya variedad en las actividades que se le devuelve al usuario.
- `plantilla`: Aquí registramos cada plantilla que se ha recomendado en Madrid Live. La plantilla contiene los datos del usuario al que se le recomendao por primera vez, información para comparar con los requisitos que pide el usuario y la valoración de los usuarios a esa plantilla.
- `plantilla_actividad`: Esta tabla relaciona la plantilla con las respectivas actividades que contiene esa plantilla. Para ello se le asigna al identificador de cada actividad, el identificador de la plantilla al que pertenec (ese identificador es el identificador que pone el historial a la actividad). Además para saber el orden de las actividades se guarda dicho valor para cada actividad, y en

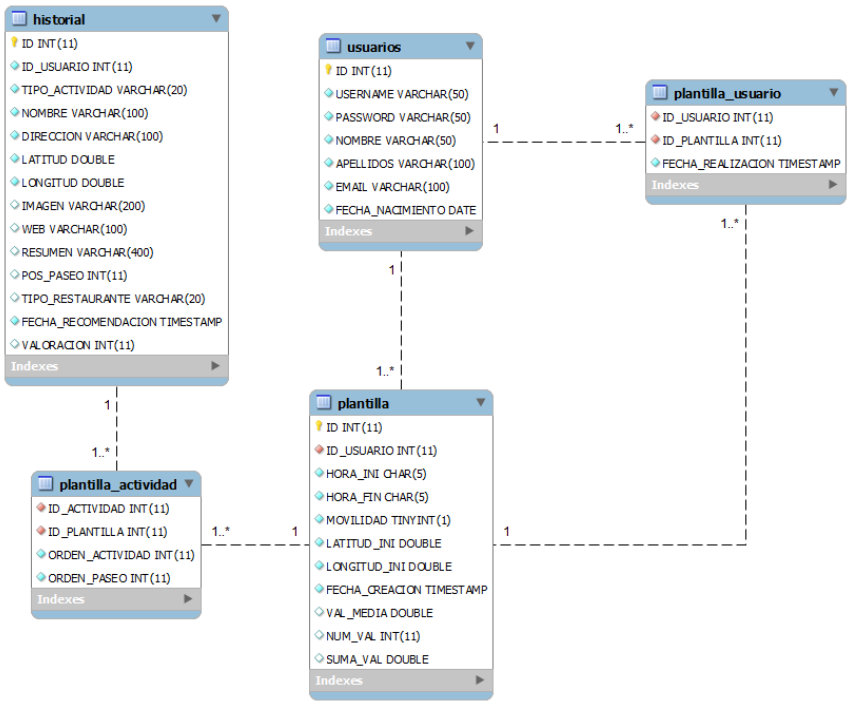

Figura B.3: Tablas de plantillas de actividades.

el caso de que fuera un paseo además se le añade la posición que ocupa en el paseo.

- `plantilla_usuario`: Por último, tenemos la tabla que relaciona cada plantilla con los usuarios que han hecho esa plantilla, es decir, la recomendación les ha devuelto esa plantilla y a los usuarios les ha gustado. Para ello relacionamos el identificador del usuario con el de la plantilla que ha hecho y la fecha en la que hizo el plan.

Apéndice C

API Madrid Live

En este apartado explicaremos los distintos servlets que componen la API, con los cuales se puede crear distintas interfaces.

C.1. Funciones de gestión de usuarios

A continuación explicamos en detalle las funciones de gestión de usuarios.

C.1.1. Login

Método que crea una sesión para el usuario. Para loguearse, el usuario debe de existir en la base de datos del sistema.

- **URL:**
  <servidor:puerto>/TurismoAPI/login.do
- **Parámetros:**
  - *username:* nombre de usuario en la aplicación.
  - *password:* contraseña del usuario para iniciar sesión en la aplicación.
- **Resultado:**
  - *mensaje:* mensaje mostrado por la API.
  - *status:* estado de la solicitud. "OK" si ha salido correctamente o "ERROR" si se ha producido algún error. En este caso, en el campo mensaje status mostrará dicho error.
- **Ejemplo de uso:**
  - *URL ejemplo:*
    ./login.do?username=usuario&password=contraseña

- *Resultado:*

```
{
‘‘mensaje’’ : ‘‘El usuario se ha logeado corr...’’,
‘‘status’’ : ‘‘OK’’
}
```

C.1.2. Logout

Método que cierra la sesión del usuario. El usuario debe de tener una sesión abierta.

- **URL:**
  <servidor:puerto>/TurismoAPI/logout.do
- **Parámetros:**
  Esta función no necesita parámetros
- **Resultado:**
  - *mensaje:* mensaje mostrado por la API.
  - *status:* estado de la solicitud. "OK" si ha salido correctamente o "ERROR" si se ha producido algún error. En caso de que se haya producido algún error, en el campo mensaje saldrá dicho error.
- **Ejemplo de uso:**
  - *URL ejemplo:*
    ./logout.do
  - *Resultado:*

```
{
‘‘mensaje’’ : ‘‘La sesión se ha cerrado corr....’’,
‘‘status’’ : ‘‘OK’’
}
```

C.1.3. Registro Usuario

Método que crea un nuevo usuario en la aplicación.

- **URL:**
  <servidor:puerto>/TurismoAPI/registro.do
- **Parámetros:**
  - *nombre:* Nombre del usuario.
  - *apellidos:* Apellidos del usuario.
  - *email:* Correo electrónico del usuario.

- *fechaNac:* Fecha de nacimiento del usuario.
- *username:* Nombre de usuario en la aplicación. Nombre con el que hará los login.
- *password1:* Contraseña del usuario.
- *password2:* Confirmación de la contraseña del usuario.

- **Resultado:**
  - *mensaje:* mensaje mostrado por la API.
  - *status:* estado de la solicitud. "OK" si ha salido correctamente o "ERROR" si se ha producido algún error. En caso de que se haya producido algún error, en el campo mensaje saldrá dicho error.
- **Ejemplo de uso:**
  - *URL ejemplo:*
    ./registro.do?nombre=Nombre&apellidos=Apellidos
    &email=user@correo.com&fechaNac=28/05/1986&username=user
    &password1=pass&password2=pass
  - *Resultado:*

```
{
''mensaje'' : ''El usuario se ha registrado corr....'',
''status'' : ''OK''
}
```

C.1.4. Buscar Usuario

Método que permite obtener el identificador y el nombre de un usuario a partir de su correo electrónico.

- **URL:**
  <servidor:puerto>/TurismoAPI/buscarusu.do
- **Parámetros:**
  - *email:* Correo electrónico del usuario buscado.
- **Resultado:**
  - *id:* identificador del usuario buscado.
  - *nombre:* nombre y apellidos del usuario buscado.
  - *status:* estado de la solicitud. "OK" si ha salido correctamente o "ERROR" si se ha producido algún error.
- **Ejemplo de uso:**
  - *URL ejemplo:*
    ./buscarusu.do?email=usuario@correo.com

- *Resultado:*

```
{
‘‘id’’ : ‘‘3’’,
‘‘nombre’’ : ‘‘usuario MadridLive’’,
‘‘status’’ : ‘‘OK’’
}
```

C.1.5. Modificar datos de Usuario

Método que permite modificar el correo electrónico o la contraseña del usuario. Para poder hacer dicha modificación, el usuario debe de tener una sesión abierta en el momento del cambio de datos.

- **URL:**
  <servidor:puerto>/TurismoAPI/modificar.do
- **Parámetros:**
  - *email:* Correo electrónico nuevo que deseamos modificar al usuario.
  - *password1:* Contraseña nueva que deseamos modificar al usuario.
  - *password2:* Confirmación de la nueva contraseña que se va a insertar.
- **Resultado:**
  - *mensaje:* mensaje mostrado por la API.
  - *status:* estado de la solicitud. “OK” si ha salido correctamente o “ERROR” si se ha producido algún error. En caso de que se haya producido algún error, en el campo mensaje saldrá dicho error.
- **Ejemplo de uso:**
  - *URL ejemplo:*
    ./modificar.do?email=usuario@nuevoCorreo.com
    &password1=nuevaPass&password2=nuevaPass
  - *Resultado:*

```
{
‘‘mensaje’’ : ‘‘El usuario se ha modificado corr....’’,
‘‘status’’ : ‘‘OK’’
}
```

C.1.6. Obtener perfil del Usuario

Esta función devuelve los datos del perfil del usuario que tiene la sesión abierta en ese momento.

- **URL:**
  <servidor:puerto>/TurismoAPI/perfil.do
- **Parámetros:**
  No es necesario pasar parámetros.
- **Resultado:**
  - *id:* identificador del usuario.
  - *username:* nombre de usuario con el que se ha registrado en Madrid Live.
  - *passwrod:* contraseña del usuario.
  - *nombre:* nombre del usuario.
  - *apellido:* apellidos del usuario.
  - *email:* correo electrónico del usuario.
  - *fecha:* fecha de nacimiento del usuario.
  - *status:* estado de la solicitud. "OK" si ha salido correctamente o "ERROR" si se ha producido algún error.
- **Ejemplo de uso:**
  - *URL ejemplo:*
    ./modificar.do?perfil.do
  - *Resultado:*

```
{
‘‘id’’ : ‘‘3’’,
‘‘username’’ : ‘‘user’’,
‘‘password’’ : ‘‘user’’,
‘‘nombre’’ : ‘‘usuario’’,
‘‘apellido’’ : ‘‘MadridLive’’,
‘‘mail’’ : ‘‘usuario@correo.com’’,
‘‘fecha’’ : ‘‘22/04/1980’’,
‘‘status’’ : ‘‘OK’’
}
```

C.2. Funciones de Recomendación

C.2.1. Recomendar plantilla

Esta función devuelve una plantilla recomendada para el usuario que tiene la sesión activa.

- **URL:**
  <servidor:puerto>/TurismoAPI/recomendador.do
- **Parámetros:**
  - *tipo:* tipo de plan que se quiere recomendar, es decir, el tipo de plantilla que se quiere obtener.
  - *horario:* horario en el que se quiere desarrollar el plan, debe indicarse la hora inicial y la hora final separadas por una coma.
  - *posición:* posición inicial del plan, se indicará la latitud y longitud de la posición separados por una coma.
  - *actividades:* tipos de actividades que debe de haber en el plan, estos tipos de actividades vendrán separados por comas.
  - *usuarios:* identificadores de los usuarios que van a participar en el plan, si viene vacío es que sólo participará en el plan el usuario que está solicitando la recomendación.
  - *tipoRecomendacion:* en caso de que sea una recomendación para un grupo de personas, se indica el tipo de función de agregación que se desea usar: máximo, media, minimaMiseria.
- **Resultado:**
  - *sol:* vector con todas las actividades del plan indicando su tipo y los datos de cada actividad.
  - *tamanyo:* número de actividades del plan.
  - *mensaje:* mensaje mostrado por la API, sólo aparece en caso de ERROR.
  - *status:* estado de la solicitud. "OK" si ha salido correctamente o "ERROR" si se ha producido algún error. En caso de que se haya producido algún error, en el campo mensaje saldrá dicho error.
- **Ejemplo de uso:**
  - *URL ejemplo:*
    ./recomendador.do?tipo=PlantillaConcreta
    &horario=12:00,16:00&actividades=Restaurantes,Museos
  - *Resultado:*

```
{
‘‘id’’ : ‘‘-1’’,
‘‘sol’’ : [
{
‘‘tipo’’ : ‘‘MUSEO’’,
‘‘actividad’’ : {
‘‘nombre’’ : ‘‘Museo Taurino’’,
```

```
‘‘direccion’’ : ‘‘Calle de Alcalá 237...’’,
‘‘resumen’’ : ‘‘Ubicado en el interior de la Mon...’’,
‘‘imagen’’ : ‘‘http://www.a2lenguas.com/images/w...’’,
‘‘web’’ : ‘‘ ’’,
‘‘horario’’ : ‘‘12:00-14:00’’,
‘‘posicion’’ : {‘‘latitud’’ : ‘‘40.43’’,‘‘longitud’’ : ‘‘-3.66’’},
‘‘paseo’’ : {
‘‘pasos’’ : [],
‘‘tamanyo’’ : ‘‘0’’
},
‘‘tipoRestaurante’’ : ‘‘ ’’
}
},{
‘‘tipo’’ : ‘‘RESTAURANTE’’,
‘‘actividad’’ : {
‘‘nombre’’ : ‘‘La Trucha’’,
‘‘direccion’’ : ‘‘Calle de Gil de Santivañés 4 - 6 (ho...)’’,
‘‘resumen’’ : ‘‘Carne’’,
‘‘imagen’’ : ‘‘http://11870.com/multimedia/imagenes/so_...’’,
‘‘web’’ : ‘‘http://11870.com/pro/la-trucha’’,
‘‘horario’’ : ‘‘14:00-15:30’’,
‘‘posicion’’ : {‘‘latitud’’ : ‘‘40.42’’,‘‘longitud’’ : ‘‘-3.68’’},
‘‘paseo’’ : {
‘‘pasos’’ : [],
‘‘tamanyo’’ : ‘‘0’’
},
‘‘tipoRestaurante’’ : ‘‘Carne’’
}
}
],
‘‘tamanyo’’ : ‘‘2’’,
‘‘usuarios’’ : [‘‘4’’],
‘‘status’’ : ‘‘OK’’
}
}
```


C.2.2. Historial de plantillas

Esta función inserta en la base de datos la plantilla, y la guarda dentro del historial de dicho usuario.

- **URL:**
  <servidor:puerto>/TurismoAPI/historial.do

- **Parámetros:**
  - *plantilla:* se inserta el jSon que se le había pasado al usuario como plantilla para hacer.
- **Resultado:**
  - *mensaje:* mensaje mostrado por la API indicando si se ha insertado correctamente la plantilla o no.
  - *status:* estado de la solicitud. "OK" si ha salido correctamente o "ERROR" si se ha producido algún error. En caso de que se haya producido algún error, en el campo mensaje saldrá dicho error.
- **Ejemplo de uso:**
  - *URL ejemplo:*
    ./historial.do?plantilla=jSonAnterior
  - *Resultado:*

```
{
''mensaje'' : ''El historial se ha guardado corr....'',
''status'' : ''OK''
}
```

C.3. Funciones para obtener Pesos de los Usuarios

C.3.1. Test de Preferencias

Esta función calcula las puntuaciones de cada ronda según la selección del usuario, y devuelve los datos que se deben mostrar en la siguiente ronda.

- **URL:**
  <servidor:puerto>/TurismoAPI/preferencias.do
- **Parámetros:**
  - *tipo:* tipo de actividad del test, Museos o Restaurantes.
  - *ronda:* ronda del test que se ha contestado.
  - *seleccionado:* tipos de museos seleccionados por el usuario o el orden de los tipos de restaurantes que ha puesto el usuario. Estos tipos deben ir separados por comas.
  - *noSeleccionado:* tipos de museos no seleccionados en la ronda, en caso de los restaurantes no se pone nada. Estos tipos deben ir separados por comas.
- **Resultado:**

- *sol:* vector con los tipos de museos o restaurantes que hay que mostrar en la siguiente ronda.
- *tamanyo:* tamaño del venctor solución.
- *ronda_siguiente:* número de la próxima ronda del test.
- *mensaje:* mensaje mostrado por la API, sólo aparece en caso de ERROR.
- *status:* estado de la solicitud. "OK" si ha salido correctamente o "ERROR" si se ha producido algún error. En caso de que se haya producido algún error, en el campo mensaje saldrá dicho error.

- **Ejemplo de uso:**
  - *URL ejemplo:*
    ./preferencias.do?tipo=Museos&ronda=2
    &seleccionado=CIENCIAS,PINTURA,HISTORIA
    &noSeleccionado=DEPORTIVO,ARTES_ESCRITAS,INFANTIL
  - *Resultado:*

```
{
''sol'' : [''CIENCIAS'', ''HISTORIA'',...],
''tamanyo'' : 6,
''ronda_siguiente'' : 3,
''status'' : ''OK''
}
```

C.3.2. Obtener actividades sin valorar

Esta función devuelve todas las actividades realizadas por el usuario y que todavía no ha valorado. Se obtienen las actividades sin valorar del usuario que tiene cargada la sesión.

- **URL:**
  <servidor:puerto>/TurismoAPI/sinvalorar.do
- **Parámetros:**
  Esta función no necesita parámetros.
- **Resultado:**
  - *tipoActividad:* vector con todas las actividades que aún no ha valorado.
  - *recomendacion:* número de actividades del vector sol.
  - *valoracion:* mensaje mostrado por la API, sólo aparece en caso de ERROR.
  - *status:* estado de la solicitud. "OK" si ha salido correctamente o "ERROR" si se ha producido algún error. En caso de que

se haya producido algún error, en el campo mensaje saldrá dicho error.

- **Ejemplo de uso:**
  - *URL ejemplo:*
    ./sinvalorar.do
  - *Resultado:*

```
{
‘‘sol’’ : [
{
‘‘tipo’’ : ‘‘MUSEO’’,
‘‘actividad’’ : {
‘‘nombre’’ : ‘‘Museo Taurino’’,
‘‘direccion’’ : ‘‘Calle de Alcalá 237, Madrid, España",
‘‘resumen’’ : ‘‘’’,
‘‘imagen’’ : ‘‘http://www.a2lenguas.com/images/words/...’’,
‘‘web’’ : ‘‘’’
}
}
],
‘‘tamanyo’’ : ‘‘1’’,
‘‘status’’ : ‘‘OK’’
}
```

C.3.3. Valorar una actividad

Esta función devuelve todas las actividades realizadas por el usuario y que todavía no ha valorado. Se obtienen las actividades sin valorar del usuario que tiene cargada la sesión.

- **URL:**
  <servidor:puerto>/TurismoAPI/valorar.do
- **Parámetros:**
  - *tipoActividad:* tipo de actividad que se ha valorado.
  - *recomendacion:* nombre de la actividad que se está valorando.
  - *valoracion:* valoración que le ha dado el usuario a esa activad, esta valoración es un número comprendido entre 0 y 5.
- **Resultado:**
  - *mensaje:* mensaje mostrado por la API.
  - *status:* estado de la solicitud. “OK” si ha salido correctamente o “ERROR” si se ha producido algún error. En caso de que se haya producido algún error, en el campo mensaje saldrá dicho error.

- **Ejemplo de uso:**
  - *URL ejemplo:*
    ./valorar.do?tipo=MUSEO&recomendacion=Museo del Prado &valoracion=4
  - *Resultado:*

```
{
‘‘mensaje’’ : ‘‘Se ha guardado la valoración corr...’’,
‘‘status’’ : ‘‘OK’’
}
```

Bibliografía

Adomavicius, G. y Tuzhilin, A. Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *IEEE Transactions on Knowledge and Data Engineering*, vol. 17(6), páginas 734–749, 2005.

Albin Rodriguez, A. P. Sistema de recomendación colaborativo basado en algoritmos de filtrado mejorados. *Escuela Politécnica Superior de Jaén*, 2009.

Elo, A. E. *The rating of chessplayers, past and present*. Arco Pub, 1978.

Galán Niet, S. M. Filtrado colaborativo y sistemas de recomendación. 2006.

Gartrell, M., Xing, X., Lv, Q., Beach, A., Han, R., Mishra, S. y Seada, K. Enhancing group recommendation by incorporating social relationship interactions. *Proceedings of the 16th ACM international conference on Supporting group work*, páginas 97–106, 2010.

Gruber, T. A translation approach to portable ontology specifications. *Academic Press*, 1993.

Isasi Viñuela, P. y Galvan Leon, I. M. Redes de neuronas artificiales un enfoque practico. *DSpace*, 2004.

Jameson, A. y Smith, B. *Recommendation to Groups*. The Adaptative Web, 2007.

Jannach, D., Zanker, M., Felferning, A. y Friedich, G. *User-based nearest neighbor recommendation*. An Introduction Recommender Systems, 2011.

Pazzani, M. J. y Billsus, D. *Content-Based Recommendation Systems*. The Adaptative Web, 2007.

Recio García, J. A. *Jcolibri. Una plataforma multi-nivel para la construcción y generación de sistemas de razonamiento basado en casos.*. Tesis Doctoral, Universidad Complutense de Madrid, 2008.