UNIVERSIDAD DE MÁLAGA

E.T.S. INGENIERÍA INFORMÁTICA

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA
GRADO EN INGENIERÍA INFORMÁTICA

MENCIÓN EN SISTEMAS DE INFORMACIÓN

DESARROLLO DE UN SISTEMA DE RECOMENDACIÓN GRUPAL TURÍSTICO

DEVELOPMENT OF A TOURISTIC GROUPAL RECOMMENDER SYSTEM

Realizado por
Ángel Jiménez Góngora

Tutorizado por
Carlos Rossi Jiménez

Departamento
Lenguajes y Ciencias de la Computación

UNIVERSIDAD DE MÁLAGA

MÁLAGA, SEPTIEMBRE DE 2019

Fdo. El/la Secretario/a del Tribunal

Fecha defensa:

Resumen

El turismo, al igual que las tecnologías web, se ha convertido en una de las principales opciones de ocio a nivel mundial. Esto ha propiciado la aparición de los sistemas de recomendación, capaces de filtrar las opciones para que los viajeros escogieran más fácil y rápidamente, automatizando el proceso de decisión. En este trabajo profundizaremos sobre una de las alternativas a los sistemas de recomendación clásicos, enfocados en un usuario en concreto, a otro donde las recomendaciones se filtrarán en función de un grupo de usuarios que vayan a viajar juntos al mismo destino, apoyándonos en un motor de recomendación basado en implicaciones lógicas con la raíz teórica del Análisis de Conceptos Formales (FCA). La aplicación, que ha sido desarrollada en fases incrementales siguiendo metodologías ágiles de gestión de proyectos como Scrum, es adaptable, escalable y construida utilizando Python y JavaScript como lenguajes principales, permitirá además de la funcionalidad principal anteriormente descrita, funcionalidades de registro y autenticación de usuarios, la configuración del motor de recomendación, así como el establecimiento de valoraciones y preferencias por parte de los usuarios.

Palabras clave: Sistema de recomendación grupal, recomendaciones orientadas al turismo, aplicación web

Abstract

In parallel to web technologies, tourism activities have grown to become one of the main options worldwide when it comes to spend leisure time. This fact opened the emergence of recommendation systems, capable of filtering over the options that travelers were being offered in order to make their choices easier and, fastly, automatizing part of the decision process. In this project, we will delve into one of the main alternatives to classic recommendation systems, commonly focused in a single user, towards a different one in which different users will be travelling to the same location joined as a group, using a recommendation motor based on logic implications with its theoretical roots coming from the Formal Concept Analysis. This web application, developed through incremental stages following agile methodologies such as Scrum, is adaptable to future changes, scalable and built using Python and JavaScript as primary languages, upon deployed will present the features such as registration and authentication of users, the possibility to configure the recommendation engine, as well as the management of valorations and preferences introduced by the users.

Keywords: Groupal recommendation system, touristic oriented recommendations, web application

Índice

1

Introducción

1.1 Motivación

En la actualidad, la era de la información, vivimos abrumados por la inmensa cantidad de datos a los que podemos tener acceso, gracias a los avances tecnológicos al respecto y al trabajo continuado de miles de personas en los últimos años. El ámbito de las ofertas de opciones turísticas no es una excepción a esta regla, por lo que también ha visto incrementado el volumen de información disponible.

Hemos tomado en cuenta un estudio estadístico de datos turísticos de la Organización Mundial del Turismo (UNWTO por las siglas en inglés), que nos indica que los números globales de viajes turísticos siguen en alza año a año, experimentando el mayor crecimiento bruto en números desde 2010, (+7%) y siendo España el segundo destino turístico más solicitado, solo por detrás de Francia, aun teniendo en cuenta los desafíos de seguridad nacional que se han tenido que afrontar estos últimos años [1].

Teniendo en cuenta todo lo anterior, es comprensible entender la necesidad de la industria enfocada a la comercialización de estas ofertas a través de Internet en aplicaciones webs a los navegantes, que elegirán aquellas que más se adecúen a sus

necesidades o preferencias iniciando una actividad comercial o un intercambio entre ambas partes.

Al aplicarlo a una situación real, el usuario está limitado por ciertos factores, como el tiempo del que dispone en el viaje turístico, el estatus económico, sus propias inclinaciones y gustos… Además, como hecho clave, lo más usual es que las personas no viajen solas y sí lo hagan en grupos (especialmente en familias, donde todavía existe un gran mercado), como factor clave en el desarrollo de este estudio. Estos hechos, unidos a la creciente cantidad de ofertas disponibles, propicia el surgimiento de métodos en el procesamiento de la información que hagan un filtrado de las opciones acorde a las características antes mencionadas, que aporten utilidad y conduzcan a la satisfacción de los usuarios en el uso de los sistemas de recomendación turísticos.

1.2 Objetivos

Se marca como objetivo principal para el desarrollo de este proyecto la consecución de una aplicación web capaz de, partiendo de un grupo de usuarios contenido dentro de todos los existentes en el sistema, obtener una selección de ítems turísticos con unas características y atributos concretos que serán acordes a las preferencias del anteriormente citado grupo.

Dicha aplicación interactuará con la API del motor de recomendación desarrollado por el grupo de investigación del SICUMA [2] para el proceso de generación de una recomendación. Este motor deberá ser configurable por un tipo de usuario especial, los administradores.

El motor de recomendación, basado en resultados de investigación [4] científica del grupo SICUMA, es una herramienta en sí misma para la generación de resultados de recomendación a partir de una serie de datos de entrada. Éstos son:

1. Los ítems u objetos a recomendar por el sistema, ligados a determinados atributos con pesos o grados entre 0 y 1.Las reglas o relaciones lógicas que incluirán ambas

partes en la configuración, llamadas implicaciones. Relacionan una causa con un efecto.

2. Los denominados segmentos contextuales (conjuntos de implicaciones), representando el contexto de los usuarios solicitantes.

Estas implicaciones, así como los contextos a los que se aplican, son generadas a partir de conjuntos de datos de entrada, usando como base teórica la Lógica y el Análisis de Conceptos Formales.

Para entender el funcionamiento de la generación de una recomendación, se debe asumir que el agente o perfil para el que se solicitará la recomendación deberá de especificar un valor contextual previo, deben encontrarse ítems suficientes en el sistema para hacer interesante la selección y las reglas deben haberse configurado con anterioridad a la solicitud de recomendación para que el resultado final de esta tenga relevancia.

Además, los usuarios deberán ser capaces de ver en tiempo real y tras la ejecución del proceso los ítems filtrados, además de poder acceder a detalles descriptivos de los mismos, así como establecer valoraciones y comentarios.

El desarrollo del proyecto se realizará siguiendo la metodología Scrum, una metodología ágil en la que se modularizan las distintas tareas existentes y se reparten en secciones temporales preestablecidas o sprints que suelen durar de 1 a 2 semanas. La principal ventaja de este enfoque es que nos permitirá realizar entregas de forma continua, agilizando el ciclo continuo de feedback.

Los procesos de análisis y diseño se realizarán gracias a la herramienta de modelado de datos **MagicDraw**, que permite la creación, gestión y ordenación de estructuras tales como requisitos, casos de uso y diagramas de todo tipo. La maquetación se realizará en la aplicación web de **Balsamiq**, con un acceso completo proporcionado por la UMA.

En relación a la arquitectura de la aplicación, se seguirá el patrón de **Modelo-Vista-Controlador** (MVC) que nos permite la separación de las lógicas en distintas capas

cohesionadas que, a su vez, reducirán el acoplamiento entre los distintos componentes. En este patrón, la capa de modelo hace referencia a la base de datos generalmente y/o a una representación de la misma en la capa lógica que la gestionará, la capa de vista generalmente es el primer punto de acceso de los usuarios con la aplicación y comunicará sus acciones con el resto y la capa de controlador agrupará las lógicas internas y de comunicación entre las distintas partes.

Para el stack tecnológico se utilizarán herramientas *open source* en su totalidad, que cuenten con un gran apoyo de la comunidad de desarrolladores global. La parte de back-end estará desarrollada en **Django** [13] en su mayoría, la parte de front-end utilizará **JavaScript** [16] basado en el framework de **Vue.js** [11] y la infraestructura que intervendrá en tareas de despliegue y comunicación de servicios será **Docker** [12].

1.3 Estructura de la memoria

La idea principal sobre la que trabajaremos será constructiva y partirá de unas bases concretas como es el contexto y estado actual del arte del problema sometido a estudio.

La descripción general del sistema abrirá el siguiente apartado, para ir abordando todas las especificaciones que han tomado parte en el desarrollo del proyecto, desde la definición de los requisitos, los casos de uso, pasando por los diagramas de representación del modelo físico de la aplicación, así como las vistas y controladores asociados. También haremos mención a la maquetación inicial prevista para la aplicación y a la arquitectura en la que está cimentada, para terminar entrando de lleno a las funcionalidades y a explicaciones del código fuente resultado del desarrollo, en parte de back-end y parte de front-end así como las interacciones entre los servicios existentes.

Finalmente realizaremos una evaluación del funcionamiento del sistema de recomendación grupal desarrollado y estableceremos unas conclusiones fruto de la reflexión personal sobre los resultados.

2

Contexto

2.1 Introducción a las recomendaciones turísticas grupales

En el mundo actual podemos observar que el sector turístico está en clara alza como una actividad comercial y de consumo global. El número de turistas es cada vez mayor debido a la facilidad con la que hoy en día se puede viajar, a la diversificación de posibilidades totales y diferentes tipos de turismo, reducción de los costes y los tiempos de viaje y la visión compartida de un mundo más globalizado e interconectado. También existen cada vez más ciudades en las que se apuesta por la innovación en un sector con mucha competitividad, así como por el objetivo de un desarrollo turístico sostenible a largo plazo para evitar la devaluación del interés de los destinos y la preservación de los bienes culturales y naturales.

Originalmente los primeros métodos de recomendación turística en los tiempos modernos han venido dados por las llamadas guías turísticas o de viaje. Éstas son libros o magazines enfocados sobre una región, una ciudad o un destino turístico concreto que contienen detalles tales como direcciones, precios, incluso a veces valoraciones críticas de hoteles, hostales, restaurantes; además de itinerarios de interés recomendados o útiles para el viajero. Más tarde, las empresas conocidas como agencias de viajes tomarían el

relevo ofertando paquetes de viajes y organizando el viaje tanto de grupos de turistas como de viajeros individuales.

Más adelante, con la aparición de los avances tecnológicos basados en la digitalización de la información, surgieron nuevas alternativas como los foros de discusión en los que intercambiar opiniones o valoraciones y la distribución electrónica de los mismos contenidos de las guías que antes se imprimían en papel, estando disponibles en forma de descarga o acceso web, y permitiendo mantener más al día cualquier información relevante contenida en estos formatos.

Los métodos más actuales comprenden aplicaciones webs personalizadas y apps móviles que proporcionan todo tipo de información sobre los destinos y las actividades a realizar dentro de ellos, además de comprobaciones de disponibilidad online a tiempo real, valoraciones de expertos y usuarios, integraciones de métodos de pago e incluso recomendaciones paralelas (sobre servicios o productos complementarios a otro objetivo principal).

Encontramos algunos ejemplos de recomendación turística en servicios como los ofrecidos por Google Trips o TripAdvisor, aunque no hay muchas soluciones concretas actualmente en el mercado que solucionen esta problemática. Nos centraremos por el momento en explicar las características de este último:

- Opción más usada y valorada globalmente en este ámbito, con alrededor de 455 millones de visitantes únicos mensuales, más de 600 millones de reseñas y opiniones de usuarios registradas [3].
- Creación de rutas e itinerarios personalizados.
- Reservas, comprobación de disponibilidad y pagos a través de la aplicación, además de gestión de tickets.
- Sugerencias categorizadas de actividades en la sección *Qué hacer* (Figura 1).

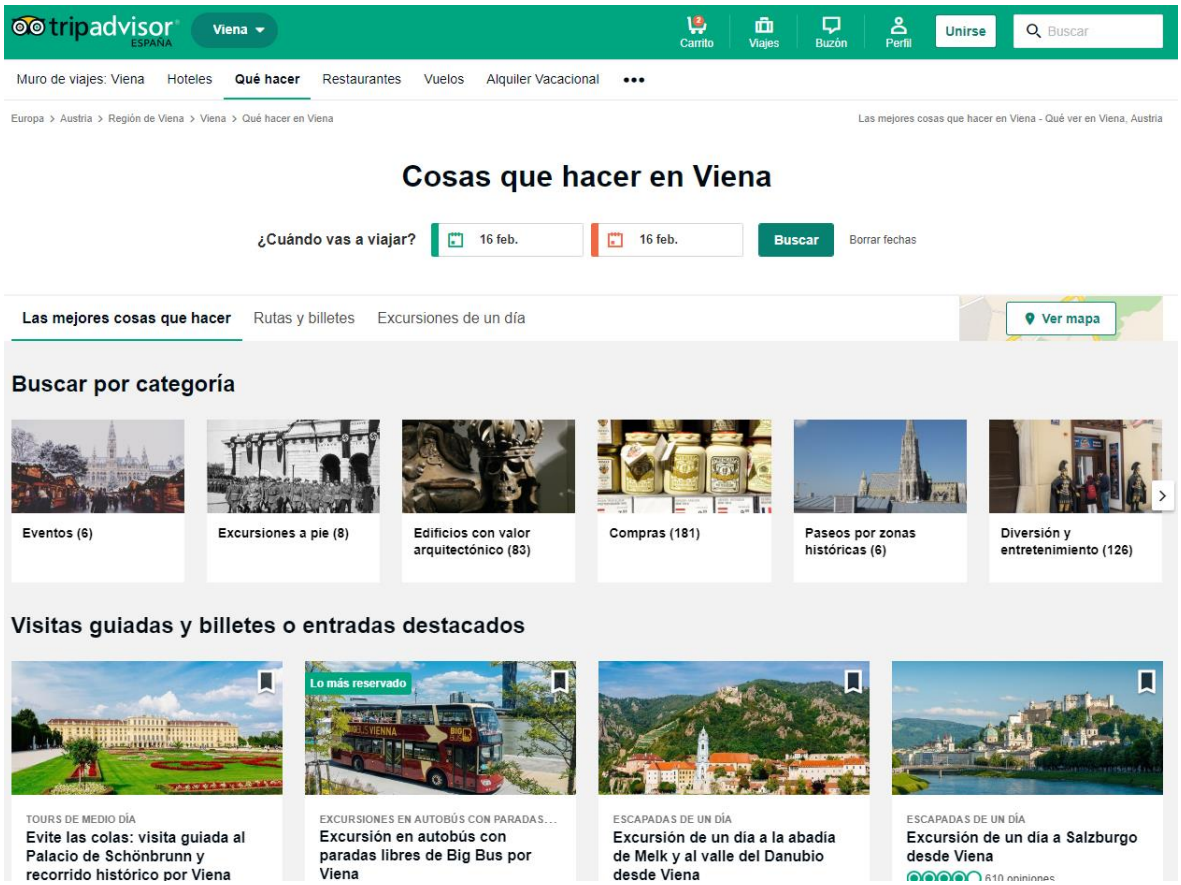

Figura 1: Qué hacer, en TripAdvisor

Esta opción es la más parecida a la metodología que llevaremos a cabo en el proyecto de este trabajo.

En octubre de 2014 TripAdvisor también presentó una nueva función, *Solo para ti*, que ofrece recomendaciones personalizadas de hoteles basadas en las preferencias del usuario y el historial de búsquedas en el sitio, pero que resultó ser un fracaso tras generar rechazo por la mayoría de usuarios ya que tuvo un enfoque excesivamente comercial.

Aunque se basa esencialmente en el uso de reseñas y valoraciones introducidas por los usuarios, se sirve además de un algoritmo de ordenamiento por popularidad y afinidad a la hora de ofrecer recomendaciones de puntos de visita a los usuarios.

2.2 Estado del arte de la recomendación turística

Definimos los sistemas de recomendación como todos aquellos que, en relación a una dimensión de ítems de información, son capaces de seleccionar y sugerir un conjunto reducido de los mismos para un usuario concreto usualmente con objeto de reducir el tiempo que el usuario pasa cavilando sobre una elección. Esta definición, que constituye el principal paradigma de avances en este campo, tradicionalmente se ha venido mostrando con un particular enfoque individualista, es decir, en relación a usuarios concretos con sus características.

Dichos sistemas, dirigidos en el ámbito comercial a clientes potenciales como navegantes en páginas web de empresas, han demostrado su eficacia a la hora de satisfacer a los clientes y aumentar sus beneficios económicos con un mayor número de transacciones e interacciones de forma sencilla al facilitarles el proceso de selección, por lo que cada vez más empresas están apostando por esta manera de captar clientes, aumentando de igual forma la red de usuarios e, inherentemente, añadiendo valor a dichos sistemas (mayor número de valoraciones y feedbacks) y refinándolos.

Debemos ser capaces de reconocer las variables que manejamos en estos sistemas. Cuando hablamos de ítem, debemos saber que se trata de un punto de interés, una actividad turística que puede ser recomendada o valorada por un usuario, definida usualmente por un punto geográfico o localización, una fecha concreta u horario y encasillada en un tipo o categoría previamente definido. Estas categorías serán vitales en el proceso de generación de las recomendaciones, determinadas por un grado de pertenencia.

También existen los que en ciertos trabajos científicos [4] se denominan contextos, como pueden ser las condiciones meteorológicas, los horarios disponibles, la preferencia de mañanas, tardes o noches, o el precio, que se muestra usualmente como uno de los factores más determinantes a juzgar por los usuarios. Dependiendo de esta dimensión particular, podemos centrarnos en tres tipos diferentes de sistemas de recomendación contextuales:

- Pre-filtrado contextual (o contextualización de la entrada de datos a recomendar), donde los valores contextuales guían la selección de ítems restringidos a los que guardan relación directa con los anteriores. Después, los datos serán la entrada de un sistema de recomendación de dos dimensiones (usuarios, ítems).
- Post-filtrado contextual (o contextualización de la salida de datos a recomendar), las valoraciones serán predichas usando cualquier sistema de recomendación tradicional de dos dimensiones, para después aplicar los valores contextuales a fin de reducir el número de ítems posibles.
- Modelado contextual (o contextualización de funciones de recomendación), donde la información contextual es usada directamente en la técnica de modelado como parte de la estimación de la valoración.

También citan, en [4], que el más indicado para un destino con un gran número de puntos de interés debería basarse en el pre-filtrado principalmente para reducir el número de datos de entrada a procesar, reduciendo el tiempo de ejecución del algoritmo de recomendación. Por otro lado, esto no reduce las posibilidades de aplicar cualquiera de los otros paradigmas.

Los ítems recomendados aparecerán en una lista de la que el usuario deberá seleccionar los más adecuadas para su viaje turístico. Esta lista será la recomendación del usuario y es el resultado del proceso anterior.

Cuando hablamos de recomendación grupal, el principal enfoque de este proyecto, es necesario ajustarse a unas condiciones y aplicar unas técnicas específicas. Ya no se trabaja con un solo usuario, sino con una lista de ellos y con las interacciones y relaciones que se crean entre los mismos. Esto hace la tarea más difícil que con el enfoque clásico, por lo que las técnicas que actualmente se proponen podrían clasificarse, según el artículo de Ludovico Boratto [5], en:

- Tipificación de grupos: A cada grupo se le asignará previamente la pertenencia a una categoría en concreto, y se generarán recomendaciones para ese tipo de grupo en concreto con ligeras modificaciones.
- Adquisición de preferencias: Un sistema de recomendación podrá adquirir las preferencias en consideración con las expresadas por cada usuario o permitiendo a los grupos la expresión de las mismas.
- Modelado de grupos: Es el proceso adoptado para combinar las preferencias individuales en un modelo único que represente al grupo.
- Predicción valorativa: Aquí se intentará dar valoraciones predictivas a los ítems para individuos o, específicamente, para grupos antes de que las mismas se produzcan, ayudando a la recomendación.
- Consenso de grupo: En consideración con cada individuo, se intentarán mostrar ítems que sean aceptados por todos los individuos, evitando el rechazo individual por encima de todo.
- Explicación recomendativa: Cada ítem irá acompañado de una justificación de pertenencia a la recomendación del grupo.

2.3 Implementación de sistemas de recomendación grupales en otras aplicaciones o servicios

En este apartado se analizarán distintas aplicaciones que hayan implementado sistemas de recomendación en un contexto relativamente actual, estén o no relacionadas con el ámbito turístico en el que nos enfocaremos a lo largo de este proyecto, pero que nos ayudarán a entender la manera en que desde otros grupos de trabajo se han afrontado estos desarrollos.

La primera que vamos a evaluar será **Spotify**, una aplicación multiplataforma empleada para la reproducción de música vía streaming, en la que cada usuario puede guardar sus propias listas de reproducción con sus canciones favoritas. Aunque nos permite encontrar cualquier canción o tema a través de su barra de búsqueda, una de sus características estrella es *Descubrimiento semanal* (Figura 2), una lista de reproducción que

agrupa treinta canciones fuera del ámbito y estilo de escucha habitual que considera que serán del agrado del mismo. Esta lista se genera partiendo del historial de reproducción del usuario, así como a partir de la aplicación de técnicas de filtrado colaborativo, a través de modelos que analizan el comportamiento de usuarios concretos que son comparados entre sí.

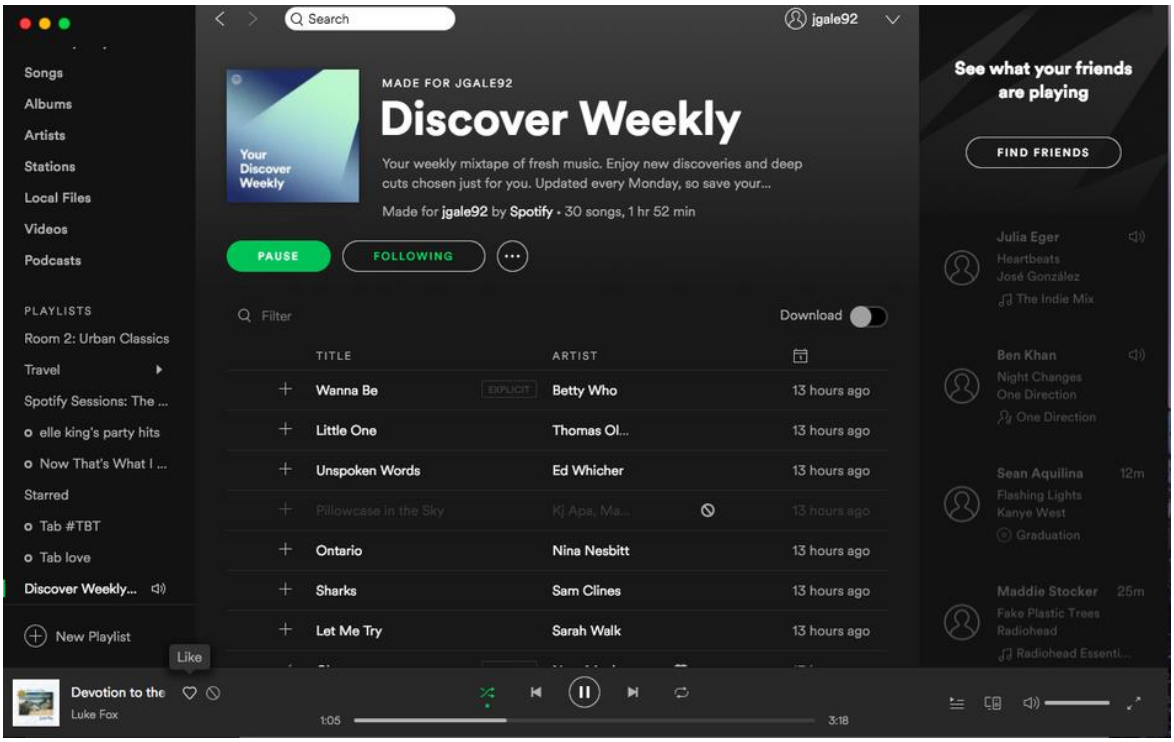

Figura 2: Descubrimiento semanal, *en* ***Spotify***

El motor de recomendación de **Spotify** ha incrementado de forma significativa la ventaja de esta compañía en su sector de negocio debido al alto nivel de éxito que es capaz de cosechar en sus recomendaciones [6]. El enfoque característico de recomendación grupal que resaltaremos aquí será el de las listas de reproducción colaborativas, en las que varios usuarios podrán añadir canciones de manera simultánea. Al final de esta lista, existirá una sección llamada *Canciones recomendadas* (Figura 3: Canciones recomendadas, en SpotifyFigura 3) que son seleccionadas automáticamente en base a las canciones existentes en la lista, además de en las características de cada usuario que las introdujo.

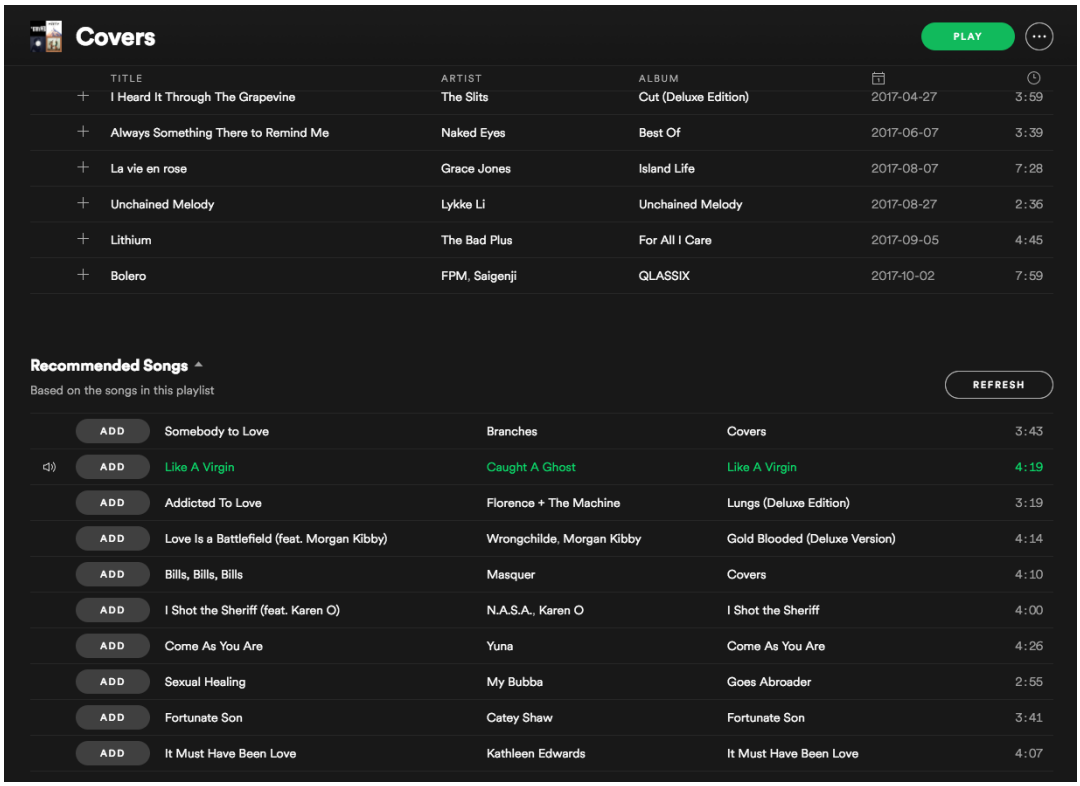

Figura 3: Canciones recomendadas, en Spotify

También a través de **Spotify** se están enfocando esfuerzos en desarrollar una prestación muy novedosa llamada *Escucha social* [7], en la que se permite a múltiples usuarios añadir canciones a una cola de reproducción que puede ser escuchada y manipulada de manera simultánea a través del escaneo de un código QR. Esto da una muestra de la dirección que se está tomando, dando un papel cada vez más protagonista al uso social de la aplicación.

Otra aplicación a evaluar **PlanChat** (Figura 4)**,** una herramienta centralizada comunicativa que permite a los usuarios descubrir, planificar y coordinar viajes en grupos [8]. Dedicada a hacer la planificación más llevadera y sencilla integrando herramientas comunicativas como mensajes de texto y correo electrónico, nos centraremos en su particular servicio de *descubrimiento*, en el cual se le presentarán a los usuarios sugerencias personalizadas de viajes y planificaciones, teniendo en cuenta su historial y sus características como grupo, dando la oportunidad de añadir las sugerencias elegidas a una lista de deseos, de la cual se monitorizarán los precios y notificarán cuando es el momento perfecto para reservarlo todo.

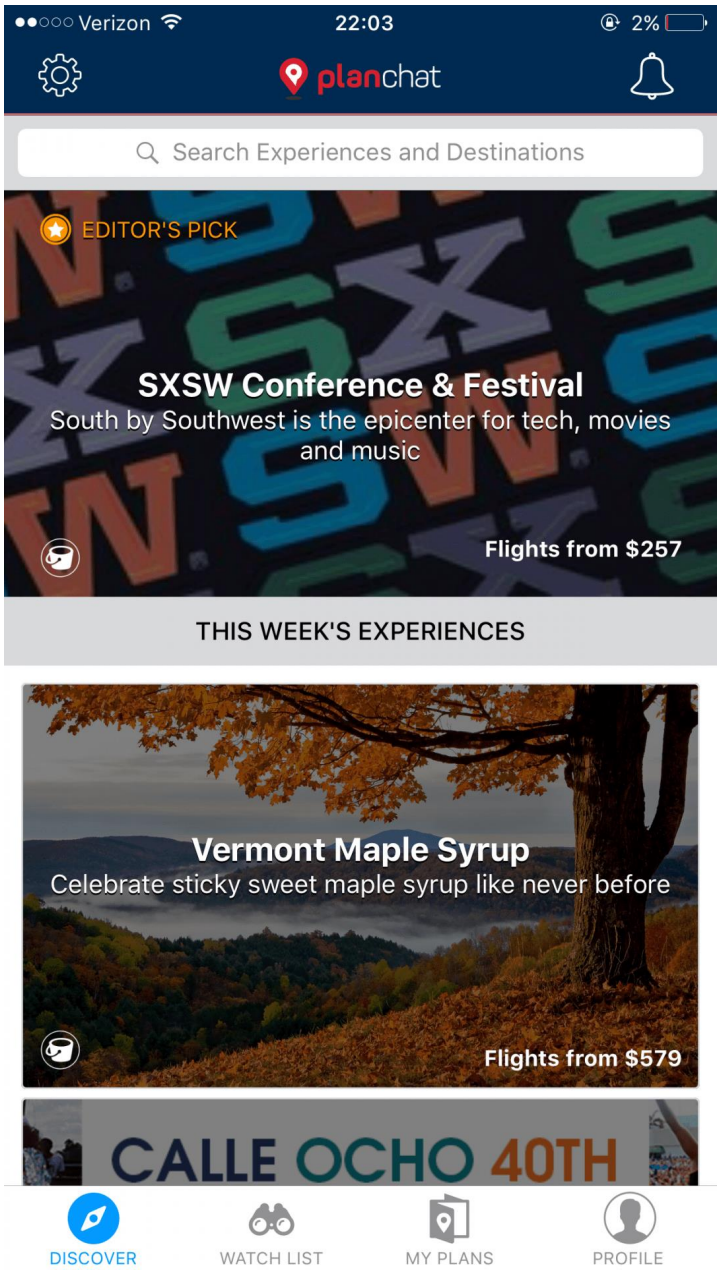

Figura 4: Aplicación iOS de PlanChat

Otra de las aplicaciones que destacaremos es **DateNightMovies** [9], un servicio en la web que nos ayudará a resolver el dilema de elección de película entre una pareja, dando una solución media que intentará agradar a ambos usuarios. Para ello, bastará con que cada uno seleccione la película que desearía ver, finalizando en una serie de resultados tras la aplicación del proceso de recomendación (Figura 5).

Figura 5: Recomendación final de DateNightMovies

3

Descripción del sistema

3.1 Descripción general

La aplicación web desarrollada consiste en un sistema de recomendación turístico orientado a la oferta automatizada de productos o recursos turísticos disponibles en una ciudad y adaptados a las preferencias de grupos de personas que vayan a viajar juntas.

Estas recomendaciones, dirigidas a los grupos, comprenderán listas de los denominados recursos turísticos que serán guardadas en los perfiles de cada grupo de usuarios, accesibles por los mismos a través de la aplicación, así como podrán acceder a los detalles específicos de cada oportunidad.

Además de la funcionalidad principal, se espera que el sistema sea capaz de alojar comentarios sobre las productos y valoraciones de los usuarios, así como de otras funcionalidades básicas como pueden ser el acceso al sistema de manera segura y autenticada mediante el uso de tokens, el registro de nuevos usuarios y la posibilidad de recuperar una contraseña olvidada.

Los productos o recursos turísticos y sus características serán introducidas por un administrador del sistema, que además tendrá la capacidad de poder configurar el motor de recomendación de la API del motor de recomendación, proporcionada por el grupo de investigación SICUMA. Esta configuración es la que definirá la generación de recomendaciones mediante las implicaciones conocidas como reglas y la incidencia de valores de los atributos sobre ellas. Más adelante explicaremos a fondo en qué consiste esta configuración.

Los administradores tendrán un control completo sobre la información alojada dentro del sistema gracias al panel de administración, teniendo en cuenta que podrán también consultar y modificar información de los productos o recursos turísticos, de sus atributos e incluso de los propios usuarios. Todos estos recursos también pueden ser borrados si los administradores lo estiman necesario.

La comunicación con el motor de recomendación por parte de la aplicación, sección que está aparte de la configuración, está completamente automatizada a través de servicios, de manera que es invisible para el usuario final, y facilitará la gestión del administrador al tratar los elementos de la API del motor de recomendación directamente como si fuesen objetos programáticos o instancias de clases dentro del propio sistema. Por ejemplo, la creación de *Implicaciones* vendrá simplificada por la relación entre uno o varios *Antecedentes* y uno o varios *Consecuentes*.

Ésta es una de las razones por las que se ha escogido el stack tecnológico empleado en este proyecto. Todas las tecnologías usadas aquí son open source, o lo que es lo mismo, de acceso gratuito y con un desarrollo y mantenimiento a cargo de una comunidad de desarrolladores. El back-end está cimentado usando el lenguaje de programación Python y el popular framework de desarrollo web Django, que permite las prácticas ORM (mapeo objeto-relacional) o de puente entre una base de datos relacional como MySQL y un servidor web. El front-end ha sido construido a partir del lenguaje JavaScript con el framework orientado a componentes de **Vue.js** como principal apoyo. Todos estos elementos compartirán una base común como la que brinda la “dockerización”, una técnica

de **Docker** que permite la reproducción de entornos virtualizados mediante el uso de contenedores que podrán ser activados o desactivados según la necesidad.

Desde el punto de vista de un nuevo usuario que entre a la plataforma, podrá registrarse, recordar su contraseña o iniciar sesión. Si el usuario todavía no está registrado, deberá hacerlo antes de poder disfrutar de la mayoría de las funcionalidades disponibles en la aplicación. Las funciones principalmente serán la de solicitar una recomendación para un grupo de viajeros a través del formulario en la página principal, la de consulta y valoración de las recomendaciones actuales o pasadas, la de establecimiento de sus propias preferencias, creación de grupos y gestión de usuarios dentro de los mismos, la de consultar y valorar opciones turísticas y la de consultar, modificar y borrar sus propias valoraciones.

Las recomendaciones son generadas gracias al motor de recomendación, al cual el sistema accede y gestiona a través de la API, y que ha sido desarrollado por el grupo de investigación SICUMA [2]. Para entender este funcionamiento, debemos entender que existen, dentro de la aplicación, cuatro entidades principales que pasaremos a describir y cuya existencia está representada en la base de datos del sistema y, a su vez, en el sistema del propio motor de forma paralela:

- Ítems: Elementos que serán objeto de las listas de recomendaciones en relación a un destino y sobre los que se podrán definir atributos y valores. En este proyecto son los productos o recursos turísticos. Un ejemplo de ítem sería por ejemplo el Museo Picasso en el destino de la ciudad de Málaga.
- Atributos: Características de los ítems que participan de las implicaciones y que toman un valor concreto (entre 0.0 y 1.0) para cada uno. Para el anterior ejemplo de ítem, atributos válidos serían el valor cultural, la concurrencia o el precio.
- Implicaciones: Reglas lógicas de aplicación del sistema, que estarán íntimamente ligadas con los segmentos contextuales. Siguiendo el curso de ejemplos para una implicación válida, dado para el Museo Picasso un valor cultural de 0.9, la concurrencia será de 0.6.

- Segmentos contextuales: Llamados “profiles” o perfiles dentro del motor de recomendación, regirán el modo de aplicación de las reglas y estarán relacionados con los usuarios y con los factores circunstanciales que les afectan principalmente. Otro ejemplo aquí podría ser el interés cultural de un grupo.

El motor de recomendación implementa funciones algorítmicas basadas en las raíces del Análisis Formal de Conceptos, y más en concreto, en la idea de implicación. Este concepto fundamental lógico, representado en su manera más simplificada por la fórmula A → B pero que puede presentarse en formas más complejas, viene a dar a entender en lenguaje natural que “si observamos que un objeto tiene el atributo A, entonces deducimos que también presentará el atributo B”. En la representación, A hace de la parte llamada antecedente o hecho observado con precedencia, y B es un consecuente o hecho asumido por consecuencia. Y más concretamente, dentro del motor cada antecedente o consecuente irá acompañado de un peso concreto, entre 0.0 y 1.0.

Este tipo de modelos son muy útiles para interpretar de una manera correcta y ordenada los flujos de conversión de la información que, de forma natural, se realizan para obtener conocimiento útil. Para nuestro proyecto en concreto, el fin último a modo de conocimiento útil o al menos aproximado será la obtención de recomendaciones relevantes de ítems turísticos para los grupos de usuarios en la plataforma.

Para ello, dentro de la configuración del motor se definirán las reglas o implicaciones que se aplicarán a cada uno de los segmentos contextuales presentes. En el contexto de uso planteado en este TFG, se asume que será un experto turístico quien defina las implicaciones, así como los atributos (y sus pesos) de cada producto o recurso turístico. Estableciendo sus valores de forma adecuada podremos obtener resultados bastante relevantes, como por ejemplo teniendo en cuenta el tipo de viaje del que se esperan recomendaciones (familiar, laboral, ocio...) o el objetivo o principal razón efectuado del viaje. Así mismo, las implicaciones referidas a ello tendrán sentido también desde un punto de vista lógico y harán referencia a las características directas de cada ítem u opción recomendable a ojos del motor de recomendación, como pueden ser el precio, la cercanía al centro de la localidad o la afluencia de gente o concurrencia del mismo. Tanto el peso o

grado de un segmento contextual como el grado de cada atributo de un ítem tendrán valores comprendidos entre 0.0 y 1.0, para representar el nivel de identificación de la entidad con la característica que se le atribuye. Tras la definición de estos valores, y en base a las implicaciones asignadas al segmento contextual, se asumirá que para el motor de recomendación su descripción se ha realizada de forma satisfactoria.

Terminada la configuración de los segmentos contextuales, las implicaciones o reglas asociadas, la definición de ítems, atributos y la asignación de los grados de pertenencia entre ambos, estará listo el sistema para que le sean solicitadas las recomendaciones.

4

Requisitos y casos de uso

En este apartado se analizarán los requisitos recabados en las fases preliminares del desarrollo de este proyecto, identificando los requerimientos mínimos que se necesitarían para considerar exitosa la consecución de este.

4.1 Requisitos funcionales

- ❑RF01 - Iniciar sesión en la aplicación web
  - ❑ Todo tipo de usuario debe estar habilitado para hacer login de forma segura.
- ❑RF02 - Modificar perfil
  - ❑ Todo usuario debe estar habilitado para gestionar las preferencias y los contextos de su perfil, así como sus valoraciones y comentarios.
- ❑RF03 - Registro de usuario
  - ❑ El sistema debe ser capaz de habilitar el registro de nuevos usuarios.
- ❑RF04 - Recordar contraseña
  - ❑ El sistema debe de proveer algún método alternativo para recordar contraseña cuando los usuarios no se acuerden de la misma.

❏RF06 - Guardar recomendación

- ❏ Todos los usuarios deben de ser capaces de guardar en la base de datos una recomendación generada para un grupo al que pertenezca.

❏RF07 - Recuperar recomendaciones

- ❏ Todos los usuarios deben de ser capaces de recuperar las recomendaciones generadas para un grupo al que pertenezca.

❏RF08 - Introducción de valores contextuales

- ❏ El usuario deberá ser capaz de introducir valores contextuales que sean imposibles de recabar por el propio motor de recomendación, como por ejemplo las fechas de visita o la localización.

❏RF09 - Introducción de preferencias individuales

- ❏ El usuario deberá ser capaz de elegir dentro de una serie de valores concretos, sus preferencias concretas respecto a las posibilidades que se le ofrecerán como resultado.

❏RF10 - CRUD grupo

- ❏ Los usuarios deben ser capaces de crear y gestionar grupos.

❏RF11 - CRUD ítem

- ❏ Los administradores deben ser capaces de crear y gestionar ítems.

❏RF12 - CRUD usuario

- ❏ Los administradores deben ser capaces de crear y gestionar usuarios.

❏RF13 - CRUD valoración

- ❏ Todo usuario debe ser capaz de introducir y gestionar valoraciones sobre los resultados del motor de recomendación y sobre los ítems.

❏RF14 - Importar un conjunto de ítems

- ❏ Los administradores deben ser capaces de importar un conjunto de ítems en un formato previamente especificado.

❏RF15 - Generación de gráficas e informes

- ❏ El sistema deberá ser capaz de generar gráficas e informes relativos a la efectividad del motor de recomendación y la actividad de los usuarios.

❏RF16 - Configuración del motor de recomendación

- ❏ Los administradores deben ser capaces de configurar el motor de recomendación de manera adecuada.

4.2 Requisitos no funcionales

- ❑RNF01 - Usabilidad
  - ❑ La interfaz debe ser sencilla y intuitiva, de manera que un usuario carente de experiencia pueda aprender a utilizarla desde el principio.
- ❑RNF02 - Estabilidad
  - ❑ La aplicación debe ser capaz registrar más de 10000 usuarios, y un uso simultáneo de al menos 50 sin problemas.
- ❑RNF03 - Seguridad
  - ❑ La gestión de los datos personales de los usuarios debe cumplir fielmente las normas establecidas de la RGPD. Los usuarios se protegerán mediante un sistema de autenticación tokenizado.
- ❑RNF04 - Compatibilidad
  - ❑ La aplicación debe ser accesible desde cualquier navegador moderno (mínimo desde Microsoft Edge, Google Chrome, Opera, Mozilla Firefox y Vivaldi)
- ❑RNF05 - Navegabilidad
  - ❑ La velocidad de respuesta de la aplicación no debe de superar los 3 segundos, salvo en la generación de recomendaciones, que deberá ser en menos de 30 segundos.
- ❑RNF06 - Falibilidad
  - ❑ La aplicación no deberá de producir ningún error grave, y los errores leves se mostrarán al usuario a través de la interfaz a modo de feedback.

4.3 Requisitos de información

- ❑RI01 - Grupo
  - ❑ Conjunto de usuarios, comportados como ítems al que pueden ser asignadas preferencias y contextos, y que pueden recibir recomendaciones.
- ❑RI02 - Ítem
  - ❑ Todo aquello calificable, susceptible de recibir una valoración por los usuarios y que forma parte de un resultado en un sistema de búsqueda. Datos utilizados por el algoritmo de recomendación.
- ❑RI03 - Dominio
  - ❑ Categoría en la que se encasilla cada ítem.
- ❑RI04 - Usuario
  - ❑ Aquel en la aplicación que, formando grupos, puede manejar su información, introducir contextos, valoraciones...
- ❑RI05 - Perfil
  - ❑ Donde reside la información referente al usuario o grupo, relacionada con sus preferencias y sus elecciones de contexto.
- ❑RI06 - Preferencia
  - ❑ Inclinación favorable o negativa sobre un ítem formalizada por un usuario o grupo.
- ❑RI07 - Atributo
  - ❑ Aspecto valorable de un ítem.
- ❑RI08 - Contexto
  - ❑ Condiciones o circunstancias que afectarán a los resultados del motor de recomendación agregadas por el usuario, el administrador o recogidas automáticamente.
- ❑RI09 - Resultado
  - ❑ Finalidad del motor de recomendación después de la aplicación del algoritmo.
- ❑RI10 - Valoración
  - ❑ Asignación de valor y/o utilidad a un ítem, atributo o grupo por parte de un usuario o grupo.
- ❑RI11 - Comentario

- ❑ Valoración en forma de texto introducido por un usuario o grupo.

❑RI12 - Puntuación

- ❑ Valoración numérica introducida por un usuario o grupo.

❑RI13 - Implicación

- ❑ Relación conclusiva entre las valoraciones de atributos y las recomendaciones.

❑RI14 - Administrador

- ❑ Aquel que, haciendo uso de la aplicación podrá introducir nuevos ítems, gestionarlos y configurar el motor de recomendación.

❑RI15 - Datos de campo

- ❑ Conjunto de datos que el sistema debe poseer antes de que el proceso de recomendación comience.

❑RI16 - Datos de entrada

- ❑ Conjunto de datos que el usuario comunica al sistema para generar una recomendación.

4.4 Jerarquía de actores

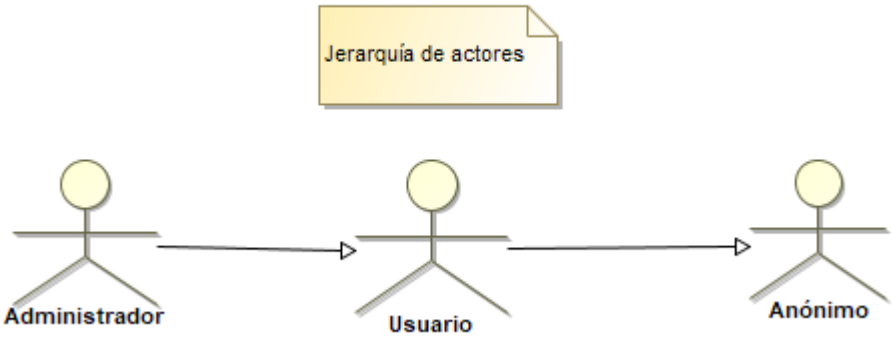

Figura 6: Jerarquía de actores

En la Figura 6 entendemos que, a nivel de permisos, los usuarios superiores serán capaces de hacer todas las acciones de los niveles inferiores, excepto casos de uso especiales como el 'Registro de usuario'.

4.5 Casos de uso de usuario anónimo

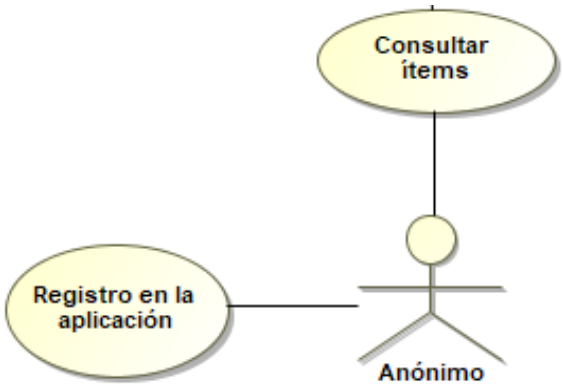

Figura 7: Casos de uso de usuario anónimo

- Consultar ítems
  - Escenario de éxito 1 - Consulta de ítems:
    - El usuario solicita el consultar los ítems existentes en el sistema.
    - El sistema muestra toda la información, paginada, de los ítems disponibles en el sistema.
  - Escenario alternativo en paso 2:
    - No existen ítems en el sistema.
    - El sistema informa al usuario o anónimo de que la lista está vacía, y de que contacte con los administradores para poder interactuar con el sistema correctamente.

- Registro en la aplicación
  - Escenario de éxito 1 - Registro:
    - Un anónimo solicita el registro al sistema.
    - El sistema muestra al anónimo un formulario indicando las informaciones pertinentes para registrar un nuevo usuario.
    - El anónimo introduce las informaciones necesarias y confirma la acción.
    - El sistema crea el usuario relativo al anterior anónimo.
  - Escenario alternativo en paso 3:

- El anónimo introduce una dirección de correo electrónico ya existente en el sistema.
- El sistema informa al anónimo de lo ocurrido y le aconseja acometer el proceso de recordar contraseña.

4.6 Casos de uso de usuario

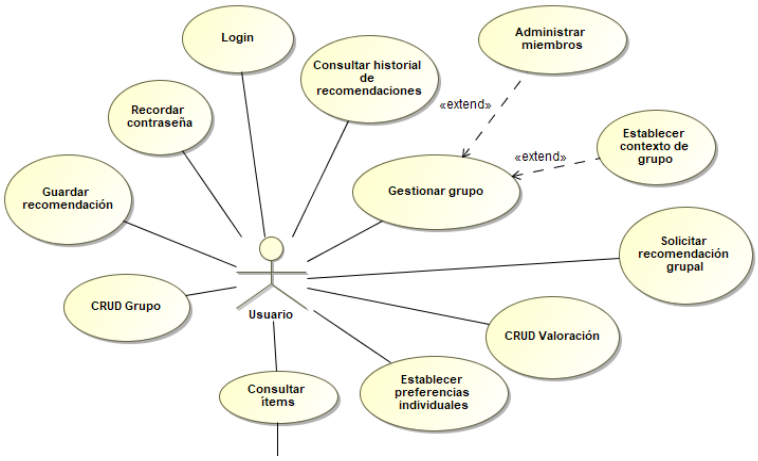

Figura 8: Casos de uso de usuario registrado

- Login
  - Escenario de éxito 1 - Login:
    - El usuario solicita el acceso al sistema.
    - El sistema muestra al usuario un formulario indicando que debe de rellenar sus credenciales para garantizar el acceso.
    - El usuario introduce su dirección de correo electrónico y su contraseña correctamente y pide comprobación.
    - El sistema comprueba que dichas credenciales son correctas y garantiza el acceso.
  - Escenario alternativo en paso 3:
    - El usuario introduce una contraseña errónea en las credenciales.
    - El sistema informa al usuario de que ha cometido un error y de que las credenciales introducidas no son válidas para garantizar el acceso al mismo.

- Recordar contraseña
  - Escenario de éxito 1 - Recordar contraseña con éxito:
    - El usuario, que es incapaz de acordarse de su contraseña, solicita recordar la contraseña al sistema.
    - El sistema solicita al usuario su dirección de correo electrónico, con la que se registró.
    - El usuario introduce su dirección de correo electrónico correctamente y completa la solicitud.
    - El sistema envía un correo electrónico a dicha dirección con la contraseña visible.
  - Escenario alternativo en paso 3:
    - El usuario introduce una dirección de correo electrónico inexistente.
    - El sistema informa al usuario de que ha cometido un error y de que esa dirección de correo electrónico todavía no ha sido registrada en él.

- CRUD grupo
  - Escenario de éxito 1 - Creación de grupo:
    - El usuario solicita al sistema la creación de un nuevo grupo desde su panel.
    - El sistema pide que añada uno por uno, desde sus contactos, a los demás usuarios que conformarán el grupo.
    - El usuario completa los integrantes del grupo y finaliza su solicitud.
    - El sistema crea el grupo conformado por esos usuarios además del solicitante.
  - Escenario de éxito 2 - Eliminación de grupo:
    - El sistema muestra al usuario los grupos a los que pertenece y a los que es administrador.
    - El usuario solicita la eliminación de uno de los que es administrador.

- El sistema elimina el grupo y todas las informaciones relativas al mismo.

- Solicitar recomendación grupal
  - Escenario de éxito 1 - Recomendación grupal conseguida:
    - El sistema muestra todas las posibilidades de establecimiento de valores contextuales previos a la solicitud de recomendación, así como la selección de grupo de entre los contactos del usuario.
    - El usuario indica valores para cada una de las opciones, además de seleccionar un grupo.
    - El sistema habilita el proceso de solicitud de recomendación.
    - El usuario solicita al sistema la generación de una recomendación grupal.
    - El sistema realiza el proceso, guarda la recomendación en el historial y la muestra al usuario.
  - Escenario alternativo en paso 2:
    - Se indica una fecha anterior al día de hoy.
    - El sistema informa al usuario de que con ese valor será imposible completar el proceso de recomendación.
  - Escenario alternativo en paso 2:
    - El usuario no elige a ningún contacto para viajar y solicita recomendación.
    - El sistema aconseja al usuario de que debe elegir algún contacto como acompañante para continuar el proceso de recomendación.
  - Escenario alternativo en paso 5:
    - El sistema no es capaz de generar una recomendación con éxito.
    - El sistema muestra debidamente cuál es la razón del fallo, e indica a contactar con soporte.

- Consultar historial de recomendaciones
  - Escenario de éxito 1 - Consulta del historial de recomendaciones:
    - El usuario solicita consultar su historial de recomendaciones.

- El sistema muestra toda la información, paginada, de las recomendaciones previamente generadas por el usuario en el historial.

- Escenario alternativo en paso 2:
  - No existen recomendaciones guardadas en el historial.
  - El sistema informa al usuario de que debe de solicitar nuevas recomendaciones antes de poder ver ninguna en esta sección.

- Guardar recomendación
  - Escenario de éxito 1 - Guardado de recomendación:
    - El sistema muestra la recomendación al usuario, además de un botón de guardado.
    - El usuario solicita al sistema, clicando en ese botón, el guardado.
    - El sistema guarda la recomendación, relacionándola con el propio usuario.
  - Escenario alternativo en paso 2:
    - La recomendación ya fue guardada por el usuario.
    - El sistema muestra la recomendación al usuario, además de un botón para dejar de guardar la recomendación.
    - El usuario solicita dejar de guardar la recomendación.
    - El sistema borra la relación del guardado, y ahora vuelve a mostrar el botón de guardado.

- Establecer preferencias individuales
  - Escenario de éxito 1 - Establecer preferencias individuales:
    - El sistema muestra todas las posibilidades de establecimiento de preferencias.
    - El usuario da valores acordes a su pensamiento y no a los del grupo.
    - El sistema modifica cada preferencia de forma acorde.
    - El usuario solicita guardar esas preferencias.
    - El sistema guarda esos valores indicados por el usuario, que tomarán parte en el proceso de recomendación.

- Escenario alternativo en paso 4:
  - Se intentan guardar valores erróneos en las preferencias.
  - El sistema informa al usuario o anónimo de que dichos valores no serán aceptados por el sistema, y aconseja qué formato o qué tipo de valores son los esperados.

- CRUD Valoración
  - Escenario de éxito 1 - Dar valoración a recomendación:
    - El usuario solicita al sistema la valoración de una recomendación generada.
    - El sistema guarda el valor de dicha recomendación.
  - Escenario de éxito 2 - Modificación de valoración de recomendación:
    - El sistema muestra al usuario las recomendaciones generadas.
    - El usuario solicita la modificación de una de las valoraciones dadas por otro valor.
    - El sistema efectúa la modificación requerida.
  - Escenario de éxito 3 - Eliminar valoración de recomendación:
    - El sistema muestra al usuario las recomendaciones generadas.
    - El usuario solicita la eliminación de una de las valoraciones dadas.
    - El sistema efectúa la eliminación y la recomendación queda sin valorar.

- Consultar ítems
  - Escenario de éxito 1 - Consulta de ítems:
    - El usuario solicita el consultar los ítems existentes en el sistema.
    - El sistema muestra toda la información, paginada, de los ítems disponibles en el sistema.
  - Escenario alternativo en paso 2:
    - No existen ítems en el sistema.
    - El sistema informa al usuario o anónimo de que la lista está vacía, y de que contacte con los administradores para poder interactuar con el sistema correctamente.

- Gestionar grupo
  - Escenario de éxito 1 - Gestionar grupo:
    - El sistema muestra al usuario los grupos a los que pertenece y a los que es administrador.
    - El usuario solicita la gestión de un grupo el cual es administrador.
    - El sistema muestra el panel de gestión.
  - Escenario alternativo en paso 2:
    - El usuario solicita la gestión de un grupo el cual no es administrador.
    - El sistema informa al usuario de que no le está permitido realizar esa acción.

- Administrar miembros de grupo
  - Escenario de éxito 1 - Añadir miembros al grupo:
    - El sistema muestra al usuario los grupos a los que pertenece y a los que es administrador.
    - El usuario solicita la adición de nuevos miembros a un grupo el cual es administrador.
    - El sistema pide que añada a los usuarios deseados al grupo.
    - El usuario, cuando llegue a estar satisfecho, pide el guardado del grupo.
    - El sistema guarda el estado del grupo con todos los integrantes añadidos.
  - Escenario de éxito 2 - Eliminar miembros del grupo:
    - El sistema muestra al usuario los grupos a los que pertenece y a los que es administrador.
    - El usuario solicita la modificación de los miembros de un grupo el cual es administrador.
    - El sistema muestra los usuarios disponibles en el grupo.
    - El usuario procede a eliminar los usuario que considere del grupo, y solicita el guardado de estado del grupo.

- El sistema guarda el estado del grupo con todos los integrantes actualizados.

- Establecer contexto de grupo
  - Escenario de éxito 1 - Establecer contexto de grupo:
    - El sistema muestra al usuario los grupos a los que pertenece y a los que es administrador.
    - El usuario solicita el establecimiento de contexto a un grupo el cual es administrador.
    - El sistema pide que introduzca los valores contextuales deseados al grupo.
    - El usuario, cuando llegue a estar satisfecho, pide el guardado del grupo.
    - El sistema guarda el estado del grupo con los valores contextuales indicados.
  - Escenario alternativo en paso 3:
    - El usuario no indica ningún valor de los necesarios y solicita el guardado.
    - El sistema indica al usuario que no es posible realizar esa acción dado que faltan informaciones por completar.

4.7 Casos de uso de administrador

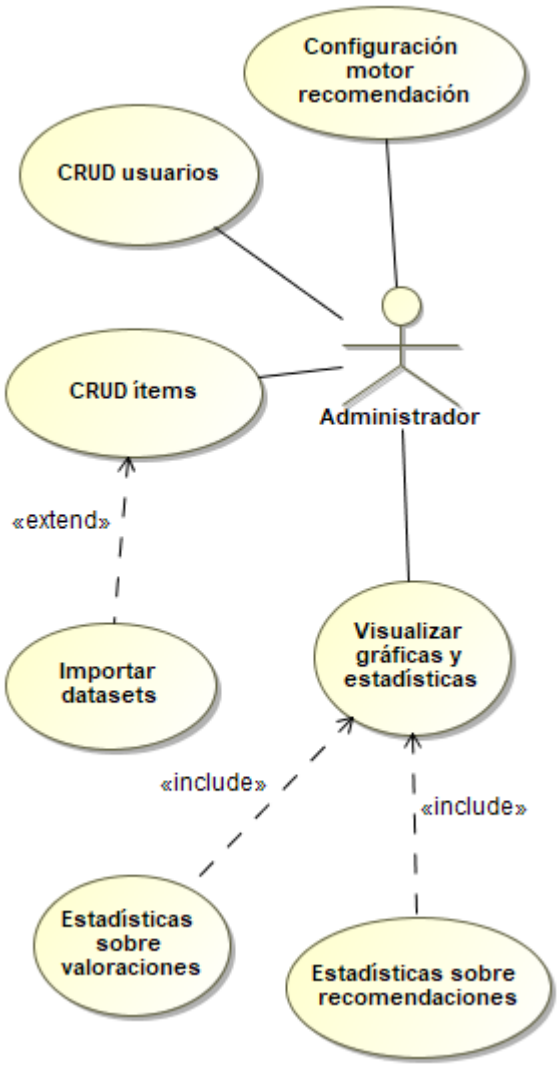

Figura 9: Casos de uso de usuario administrador

- CRUD Usuarios
  - Escenario de éxito 1 - Creación de usuario:
    - El administrador accede a la página de creación de usuarios mediante el botón de crear usuario.
    - El sistema muestra un formulario con todos los campos existentes en el modelo de usuario.
    - El administrador rellena todas las informaciones que considera necesarias.
    - El administrador clica en el botón de creación.
    - El sistema crea al usuario, que a partir de ese momento podrá ser utilizado.

- Escenario de éxito 2 - Cambio de email:
  - El administrador accede a la página de edición del usuario en concreto.
  - El sistema muestra un formulario con todos los campos existentes en el modelo de usuario, previamente rellenados con las informaciones actuales.
  - El administrador sustituye el email existente por el deseado.
  - El administrador clica en el botón de guardado.
  - El sistema actualiza al usuario manipulado por el administrador.
- Escenario alternativo en paso 3:
  - El administrador rellena todas las informaciones pertinentes, con un email en un formato incorrecto.
  - El administrador clica en el botón de guardado.
  - El sistema muestra un aviso en rojo, solicitando cambiar el email por uno válido y el usuario no es creado o actualizado.

- CRUD ítems
  - Escenario de éxito 1 - Creación de ítem:
    - El administrador accede a la página de creación de ítems mediante el botón de crear ítem.
    - El sistema muestra un formulario con todos los campos existentes en el modelo de ítem.
    - El administrador rellena todas las informaciones que considera necesarias.
    - El administrador clica en el botón de creación.
    - El sistema crea al ítem, que a partir de ese momento podrá ser utilizado.
  - Escenario de éxito 2 - Listado de ítems:
    - El administrador accede a la página de listado de ítems desde el panel de administrador.

- El sistema muestra toda la información, paginada, de los ítems disponibles en el sistema, así como de la posibilidad de editarlos o borrarlos.

- Escenario alternativo en paso 2:
  - El sistema muestra una lista vacía, no hay ítems.
  - El sistema sugiere el botón de crear ítem.

- Importar datasets
  - Escenario de éxito 1 - Importar datasets:
    - El administrador clica en el botón de 'Importar datasets'.
    - El administrador sube el archivo relacionado.
    - El sistema completa el proceso de importación del dataset.
    - El administrador, desde el panel de administración, será capaz de ver los datos importados.
  - Escenario alternativo en paso 2:
    - El administrador sube un archivo con un formato erróneo o inesperado.
    - El sistema mostrará un mensaje de error, y volverá a sugerir al usuario que reinicie el paso 2 con un archivo correcto.

- Visualizar gráficas y estadísticas
  - Escenario de éxito 1 - Visualizar principales gráficas:
    - El administrador accede a la página de 'Estadísticas'.
    - El sistema muestra todas las gráficas en conjunto.
    - El administrador visualizará más informaciones detalladas sobre las gráficas al pasar el cursor.
  - Escenario alternativo 1:
    - El administrador accede a la página de 'Estadísticas'.
    - El sistema muestra todas las gráficas en conjunto.
    - El administrador clica en 'Valoraciones'.
    - El sistema filtra solo las estadísticas sobre valoraciones.
  - Escenario alternativo 2:

- El administrador accede a la página de 'Estadísticas'.
- El sistema muestra todas las gráficas en conjunto.
- El administrador clica en 'Recomendaciones'.
- El sistema filtra solo las estadísticas sobre recomendaciones.

- Configurar motor de recomendación
  - Escenario de éxito 1 - Introducción de valores contextuales:
    - El administrador solicita al sistema comenzar el proceso de configurar el motor de recomendación.
    - El sistema muestra las implicaciones y contextos que existen en el sistema.
    - El administrador da valores a cada uno.
    - El administrador solicita guardar esos parámetros de ejecución para el motor.
    - El sistema guarda con éxito los valores introducidos.
  - Escenario alternativo en paso 2:
    - No hay contextos o implicaciones disponibles en el sistema.
    - El sistema muestra la lista vacía, a la vez que un botón para añadir contextos o implicaciones.
    - El administrador introducirá todos los contextos o implicaciones que considere necesarios.

5

Modelado estructural y de comportamiento

En este capítulo podremos visualizar cómo estructuramos según el patrón de arquitectura de software MVC (modelo, vista, controlador) la aplicación de recomendación grupal. Esto nos va a permitir establecer una separación lógica entre los distintos componentes internos, de modo que no exista un acoplamiento inherente en el que un cambio en una parte pueda afectar a otras zonas del código que no esperaríamos, respetando el principio de responsabilidad única de los componentes y facilitando los futuros desarrollos de la aplicación, así como la gestión y tratamiento de los posibles errores que puedan aparecer.

5.1 Diagrama de clases de entidad

Como observamos en la Figura 10, la clase *User* centraliza la mayor parte de las acciones, estando relacionada con la clase *Group* que es con la que la clase *Recommendation*, la más importante de nuestro entorno, se relaciona.

Una recomendación dentro de nuestro sistema será un conjunto de ítems, por lo que *Recommendation* vendrá relacionada de uno a muchos con la clase *Ítems*. Estos mismos ítems son los que se utilizarán para las recomendaciones en la interacción con el motor de recomendación, en un par junto con la otra clase *ItemAttribute* y en valor o peso dado por la otra clase *PertenanceGrade*. También se contendrán y reproducirán los valores contextuales a través de las clases *UserContext* y *ContextSegment* gracias a las clases *Implication*, *Antecedent* y *Consequent*, que representarán la capacidad de creación y mantenimiento de implicaciones, estableciendo las configuraciones del motor de recomendación.

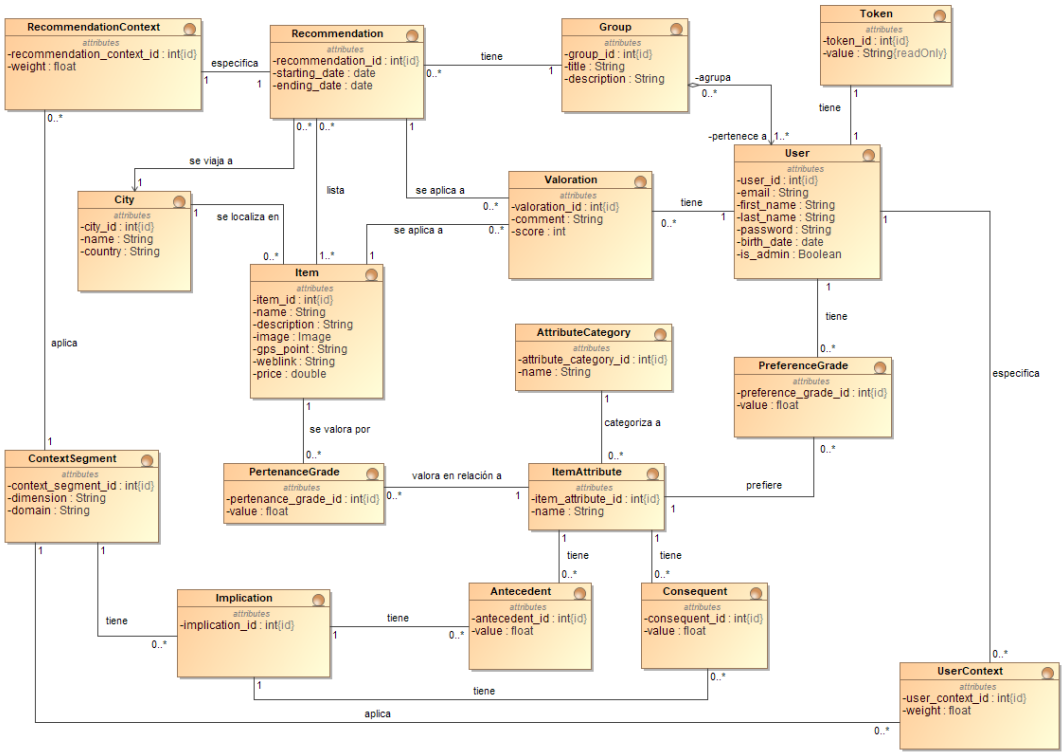

Figura 10: Diagrama de clases de entidad

No nos olvidamos de las preferencias, modeladas a través de la clase *PreferenceGrade*, además de la categorización de los atributos de los ítems, a través de la clase *AttributeCategory*.

5.2 Diagrama de clases de control

En esta capa de la arquitectura, representada en la Figura 11 van a residir las principales lógicas de la aplicación. Muchas veces actuará de intermediaria entre el modelo de datos físico, ejecutando los accesos y las escrituras en base de datos y entre las clases de interfaz que enseñarán los resultados al usuario.

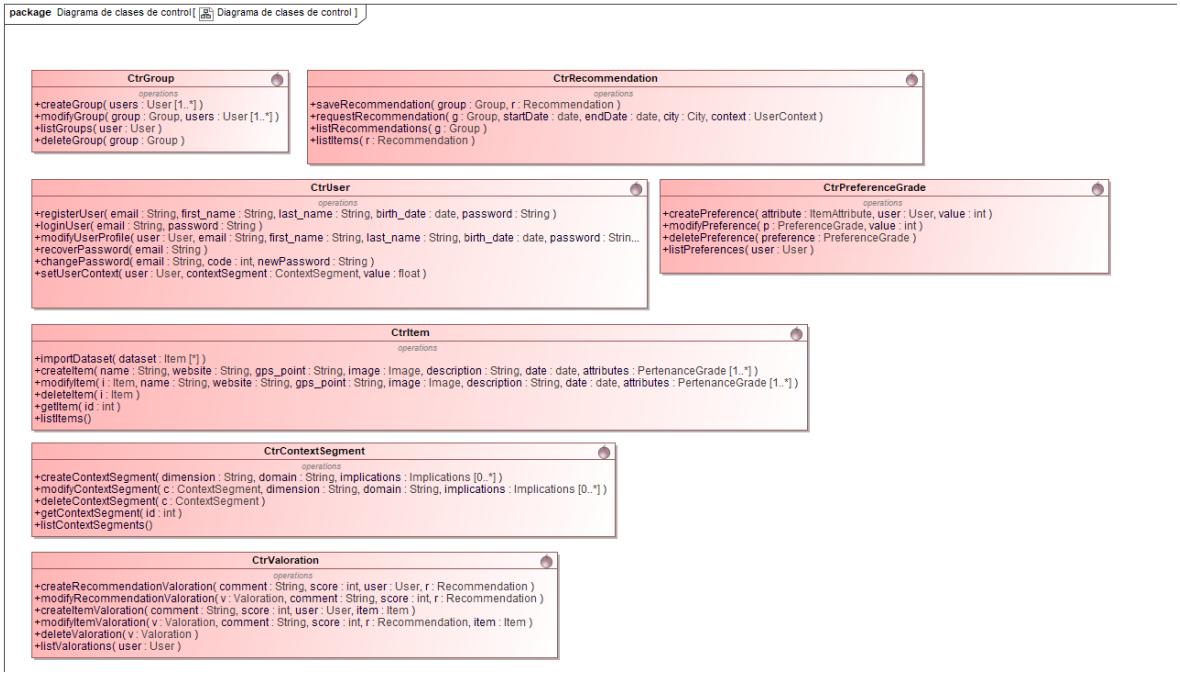

Figura 11: Diagrama de clases de control

En este caso, las clases de control van íntimamente ligadas a los modelos existentes en la aplicación terminando por ejecutar acciones sobre los mismos de manera separada. Un poco más especial es la clase de control de usuario, *CtrUser*, ya que interviene en procesos lógicos de autenticación y validación de usuarios además de la gestión de contraseñas. Del resto también a destacar *CtrRecommendation*, que contiene la puerta de entrada a la solicitud de recomendaciones en la interacción con el motor de recomendación externo y la gestión de las respuestas para construir una lista de ítems y ligarlos a dicha recomendación, y *CtrImplication* además de *CtrContextSegment* que intervendrán en la configuración del motor de recomendación a través de los procesos disponibles para los administradores.

CtrItem nos permitirá, además de gestionar los ítems, el import de un dataset existente. *CtrGroup* se encargará de lleno en de la gestión de grupos de usuarios para recomendaciones.

CtrValoration, *CtrPreferenceGrade* intervendrán en los procesos de los usuarios de crear valoraciones y preferencias sobre los ítems a recomendar y/o recomendados.

5.3 Diagrama de clases de interfaz

En este diagrama, representado en la Figura 12 se encontrarán las clases de interfaz que conformarán la capa intermedia con la que el usuario tendrá contacto con la aplicación. Acciones en botones y visualización de páginas serán las principales funciones de las que estas clases serán responsables, que podrán desencadenar otras acciones sobre las clases de control, tomando como información de entrada la introducida por el usuario, ya sea por formularios o de manera contextual a la entidad que estén haciendo referencia.

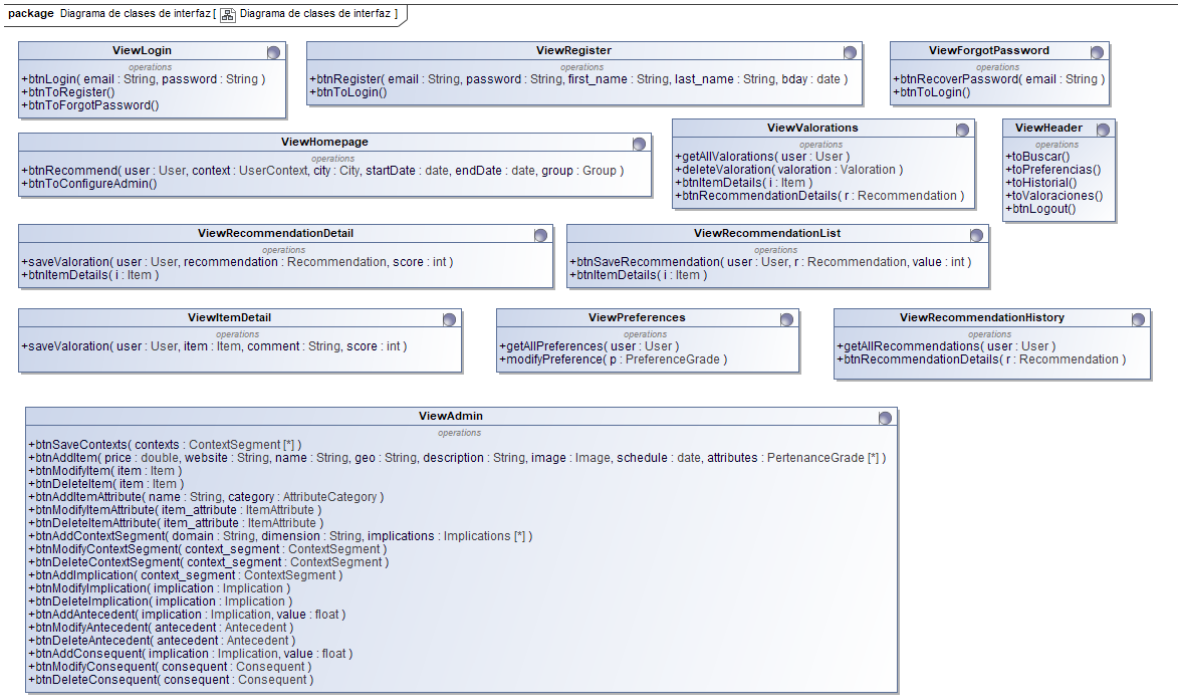

Figura 12: Diagrama de clases de interfaz

La vista de admin, *ViewAdmin* observamos que al tener control total sobre la aplicación y la entrada de datos tendrá la gestión de la totalidad de clases físicas de la aplicación. El encabezado de la página también tendrá su propia vista, *ViewHeader*, desde la que se facilitará la navegación por apartados.

Todas estas vistas están íntimamente ligadas con las maquetas de la aplicación, que serán presentadas en el siguiente apartado.

5.4 Modelo físico de datos

En este diagrama mostrado a través de la Figura 13 representaremos el mapeo de la base de datos a nivel físico en nuestra aplicación, y a través de qué campos se relacionan las tablas para ello. Todo esto se hace principalmente gracias a las claves primarias o IDs implementadas en las tablas, que identifican con un número autoincremental a cada fila de la base de datos o registro, y claves foráneas o FKs, que relacionan registros de objetos diferentes entre sí con una referencia a su ID desde tablas ajenas a ellos.

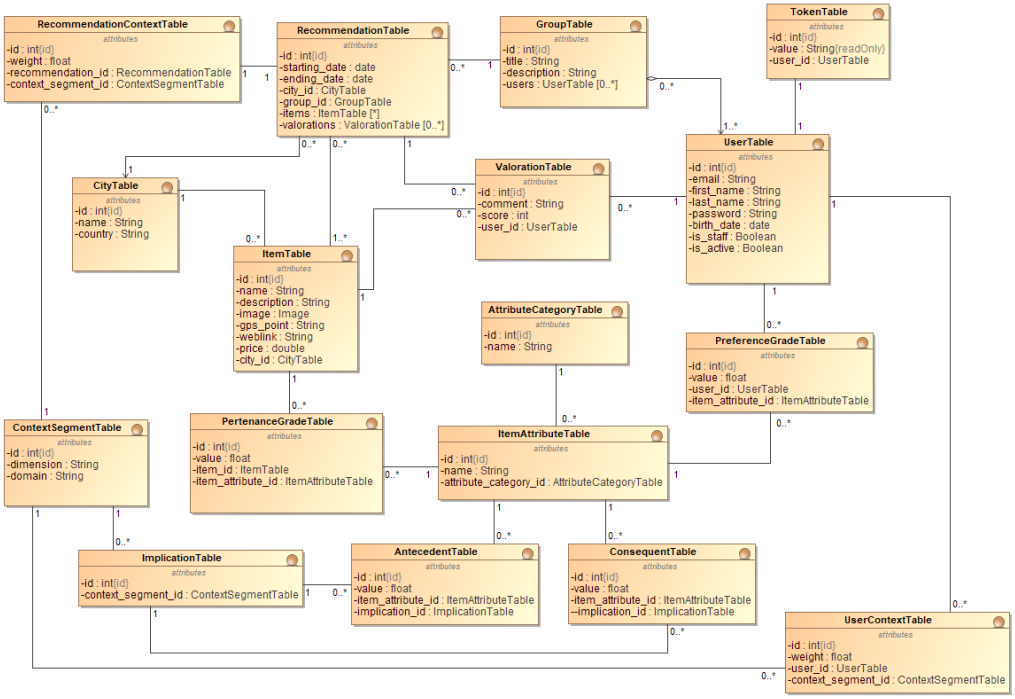

Figura 13: Modelo físico de datos

5.5 Diagramas de secuencia

Este tipo de diagramas son útiles para comprender el comportamiento en un contexto dinámico y sincronizado de los distintos componentes entre sí, para un determinado caso de uso y a través de un flujo secuencial de las interacciones entre los mismos. A continuación, se describen los diagramas de secuencia de los casos de uso más representativos del sistema.

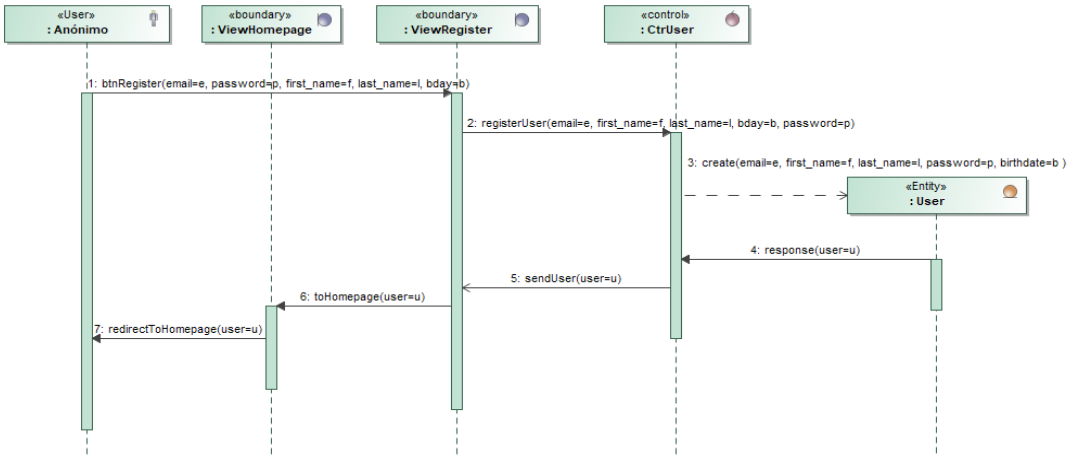

Figura 14: Diagrama de secuencia de registro de usuario

En la Figura 14, se describe el flujo normal de registro de usuario, en el que un usuario anónimo es capaz de crear un nuevo usuario dentro del sistema con los permisos básicos para habilitar su uso, y posibilitando la acción de acceso al sistema a través del mismo. La acción de clicar en el botón de registro pasa a través de la vista de *ViewRegister*, que a través del controlador *CtrUser* creará una nueva entidad de usuario en la base de datos con la información del formulario que ha ido comunicándose a través del flujo. Al final de la acción seguirá el curso normal de respuestas para que al final el propio actor *Anónimo* se convierta en un actor de tipo *Usuario normal*.

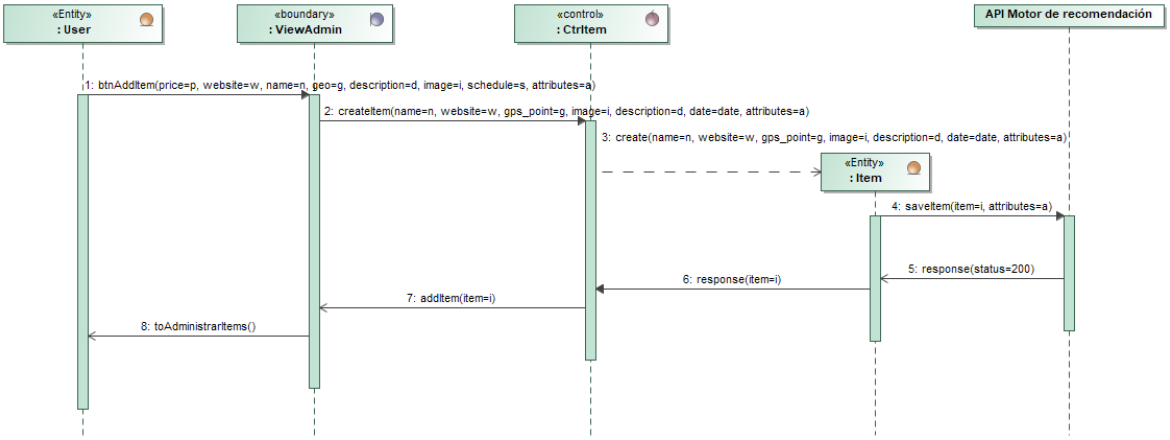

Figura 15: Diagrama de secuencia para añadir ítem

En el diagrama de la Figura 15 , un usuario autorizado con permisos de administrador es capaz de crear un nuevo ítem dentro del sistema que podrá ser utilizado para configurar el motor de recomendación y para intervenir en la generación de recomendaciones después de introducirlo en el motor. De nuevo interviene otro botón como suele ser habitual en las vistas, el *btnAddItem*, para que a través de la vista de *ViewAdmin*, comunique con la entidad de control *CtrItem* para crear un nuevo ítem en base de datos y, consecuentemente, en el motor de recomendación con la información pertinente del formulario que ha ido comunicándose a través del flujo. Al final del flujo de acciones del diagrama se proporcionará al usuario *Administrador* con las respuestas de que su acción se ha resuelto de forma satisfactoria.

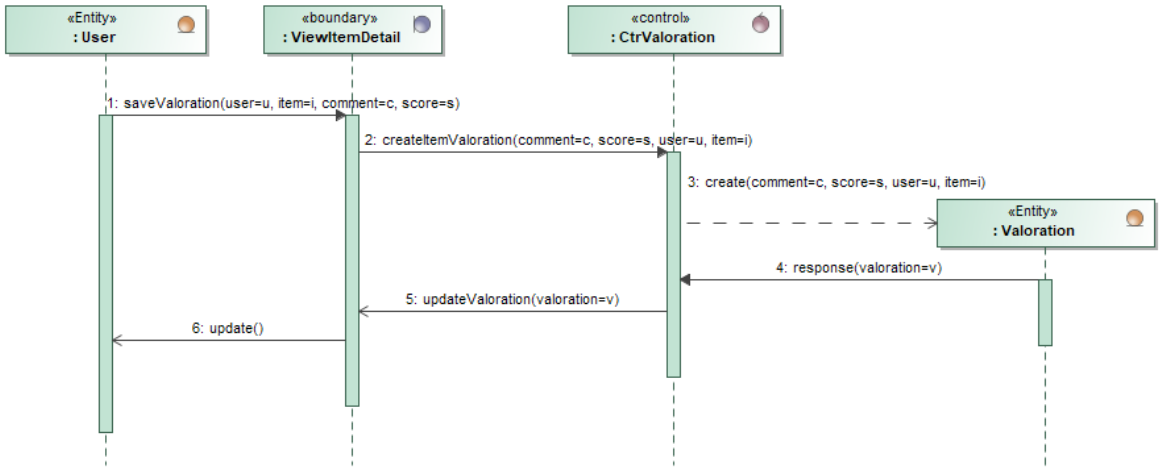

Figura 16: Diagrama de secuencia para añadir una valoración

En el diagrama de la Figura 16 un usuario normal de la aplicación creará una nueva valoración en la aplicación en relación a un ítem. Es importante recordar que este tipo de valoraciones también serán posibles para una recomendación. A través de la acción *saveValoration* comenzará el flujo de acciones, para luego por la vista de *ViewItemDetail* comunicar con la entidad de control *CtrIValoration* para crear una nueva valoración en base de datos que podrá ser vista por el resto de los usuarios de la aplicación. Al final del flujo de acciones del diagrama se proporcionarán al usuario las respuestas de que su acción se ha resuelto de forma satisfactoria.

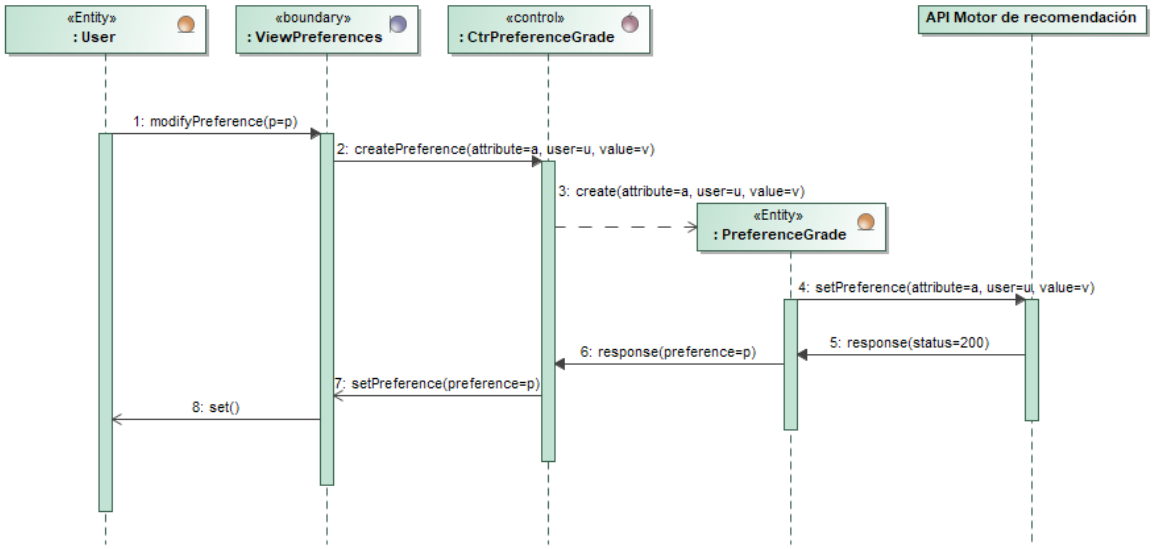

Figura 17: Diagrama de secuencia de establecer preferencia

En el diagrama de la Figura 17 un usuario normal de la aplicación establecerá una preferencia nueva o existente en la aplicación con relación a un tipo de ítem con un valor en concreto. Para ello se navegará desde la primera acción sobre el botón de *modifyPreference* en la vista *ViewPreference*, a través de la entidad de control de *CtrPreferenceGrade* para generar la entidad de clase de *PreferenceGrade* a partir de las informaciones específicas. Esta preferencia será compartida dentro del motor de recomendación también. El usuario que origina la acción recibirá también las respuestas acordes haciéndole saber del éxito de su petición.

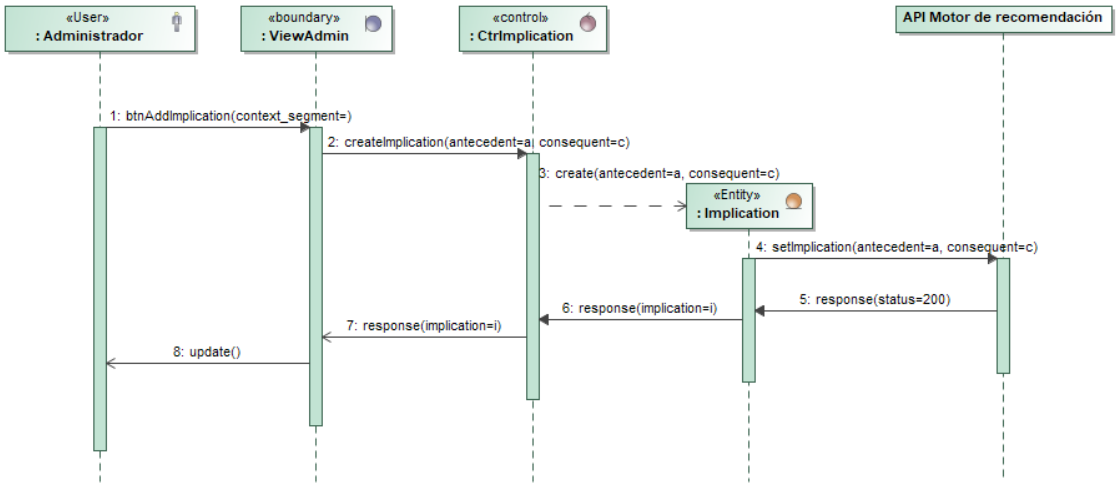

Figura 18: Diagrama de secuencia de configurar motor de recomendación

En el diagrama de la Figura 18 se representa la configuración de una regla de implicación dentro del motor de recomendación por un usuario de tipo *Administrador*. Para ello el *Administrador* tendrá que indicar el *Contexto* al que hará referencia dicha regla, además de los valores contenidos en la misma y sus antecedentes y consecuentes. Después de eso, al pasar a través de la vista *ViewAdmin* en el botón de *btnAddImplication* a partir de la clase de control *CtrImplication* se generará una entidad de tipo *Implication* que desencadenará en el contacto con la API del motor de recomendación para añadir la totalidad de la regla. De nuevo, el *Administrador* es notificado en las llamadas hacia atrás del éxito de su petición.

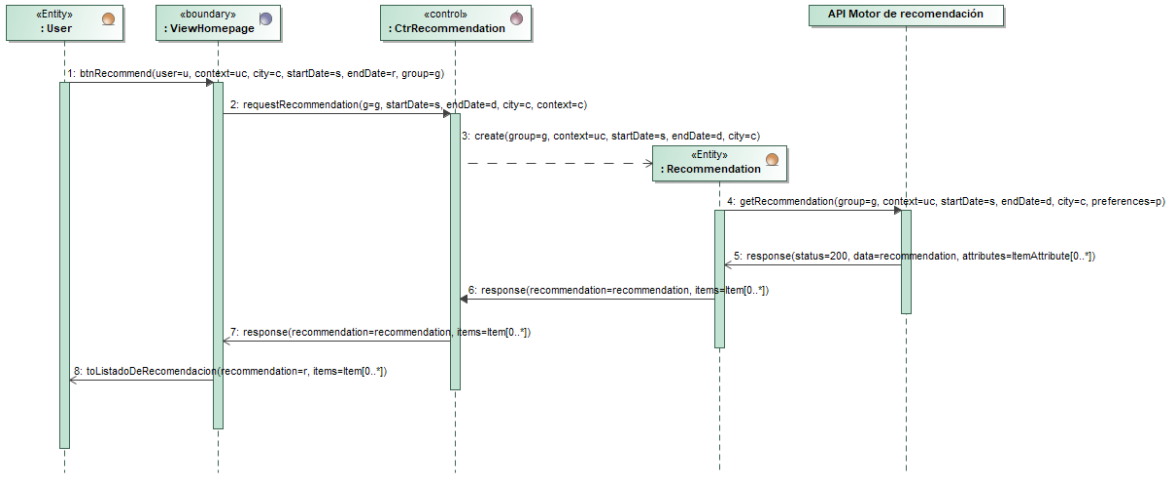

Figura 19: Diagrama de secuencia de generación de una recomendación

Llegamos al diagrama (Figura 19) más importante de todos en el que ocurre la acción principal referente a la razón de ser de esta aplicación. Vemos que un usuario cualquiera registrado en la aplicación es capaz de pedir una recomendación ya que, aunque vaya a ser una recomendación enfocada a un grupo en concreto, deberá ser un único usuario el que elija la solicite para sus acompañantes (que deben ser usuarios actualmente existentes en la aplicación).

Así mismo, deberá indicar también el valor contextual que habrán indicado previamente para poder generar la recomendación, además de la ciudad destino y las fechas a planear para el viaje antes de clicar en el botón de *btnRecommendation* que iniciará el proceso final en la vista *ViewHomepage*. Después, como viene siendo habitual contactará con su correspondiente clase de control, *CtrRecommendation* con esos mismos parámetros definidos con anterioridad para crear una entidad de tipo *Recommendation*, que a su vez efectuará de disparador de la función encargada de obtener la recomendación final en contacto con la API del motor, *getRecommendation*, que a su vez tendrá en cuenta también las preferencias de los usuarios.

La respuesta de la API del motor vendrá definida por un conjunto de ítems representados por sus *ids* que se recogerán en el controlador de nuevo y serán devueltos en las llamadas de vuelta para terminar por actualizar la página web actual de la aplicación por una nueva en la que se mostrará la lista de ítems obtenidos.

6

Modelado de interfaz de usuario

Previo al desarrollo de la parte front-end de aplicaciones web, es común y útil el uso de mockups o maquetación simple al menos para ir modelando como debería mostrarse la aplicación a nivel de interfaz de usuario (UI, User Interface) sin incidir en los detalles estéticos (UX, User eXperience), y prestando atención puramente la funcionalidad. También es posible hacer esta maquetación teniendo en cuenta diseño UX a nivel gráfico, aunque dejaremos ese enfoque para proyectos a nivel mucho más profesional y/o de empresa.

Para el maquetado usaremos una aplicación web, Balsamiq [10], haciendo uso de la licencia académica de la UMA. En esta aplicación seremos capaces de, intuitivamente, arrastrar y soltar componentes tales como campos de formularios, ventanas, alertas, cuadros… con una apariencia sencilla y personalizable. Se mostrarán a continuación las

imágenes de las maquetas creadas con tal fin, ordenadas por los flujos de acción temporales más lógicos en los que intervienen.

Figura 20: Acceso principal

Escenario de login clásico, con campos para poder rellenar las credenciales.

Figura 21: Registro de usuario

Formulario de registro con la información mínima para crear un usuario.

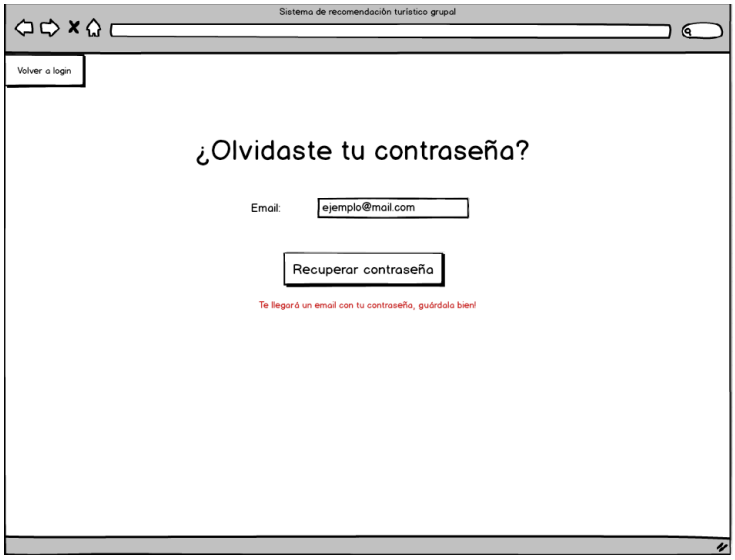

Figura 22: Recordar contraseña

Página en la que podrás recuperar tu contraseña, se mandará al correo indicado si encuentra al usuario.

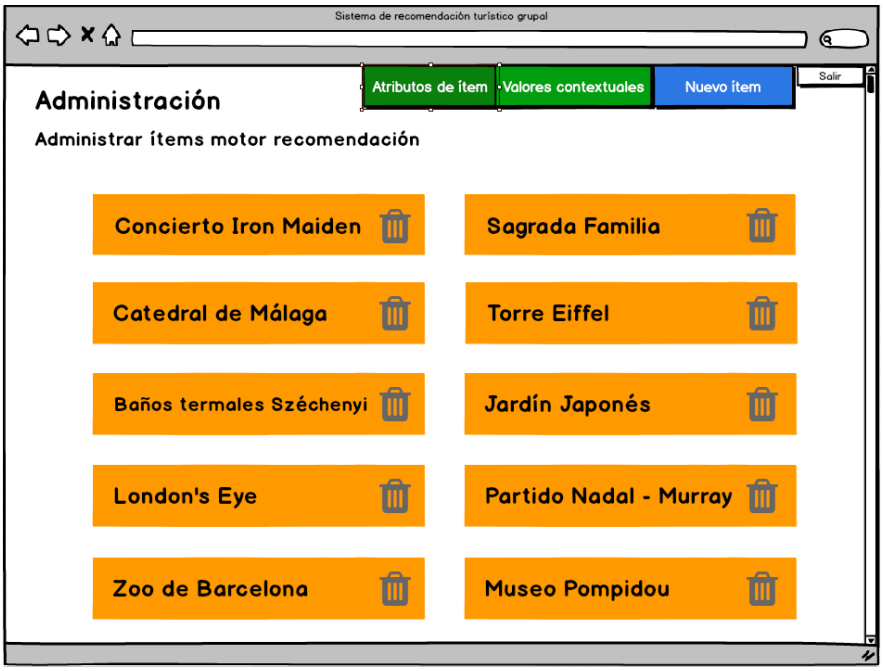

Figura 23: Administrar ítems

Panel de control del administrador, donde podremos gestionar los ítems del sistema.

Figura 24: Añadir o modificar ítem

Página de adición de un nuevo ítem o edición de uno existente en el sistema.

Figura 25: Administración de atributos de ítem

Panel de control del administrador, donde podremos gestionar los atributos de ítems del sistema.

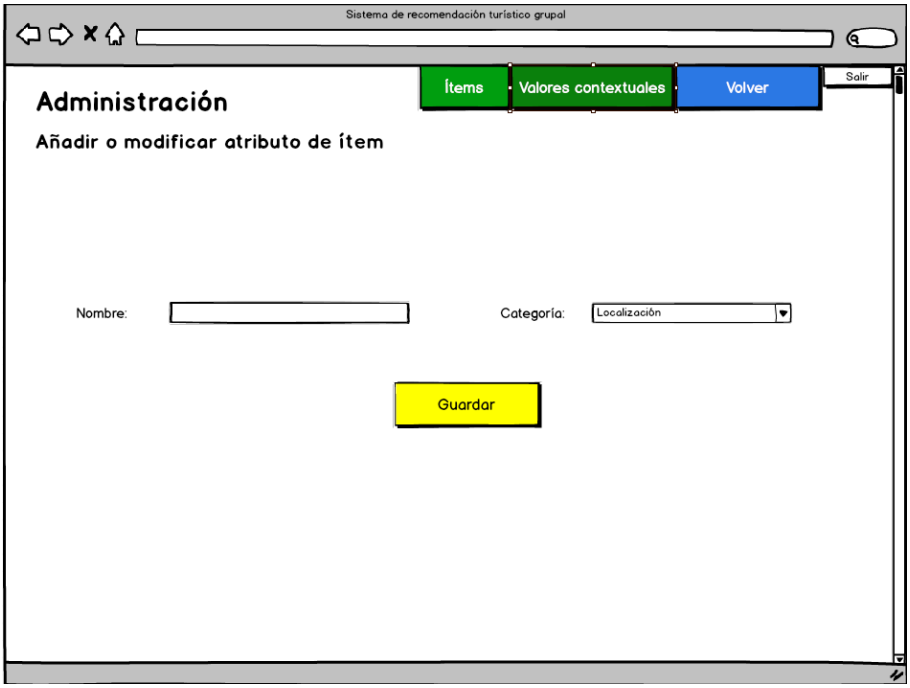

Figura 26: Añadir o modificar atributo de ítem

Página de adición de un nuevo atributo de ítem o edición de uno existente en el sistema.

Figura 27: Administración de valores contextuales

Panel de control del administrador, donde podremos gestionar los segmentos de contexto del sistema.

Figura 28: Añadir o modificar valor contextual

Página de adición de un nuevo segmento de contexto o edición de uno existente en el sistema.

Figura 29: Añadir o modificar implicación o regla

Página de adición de una nueva implicación o edición de una existente en el sistema.

Figura 30: Página principal o Homepage

Página a la que accederemos después del login o del registro, aquí podremos solicitar las recomendaciones para un grupo específico, en un contexto determinado y a un destino concreto.

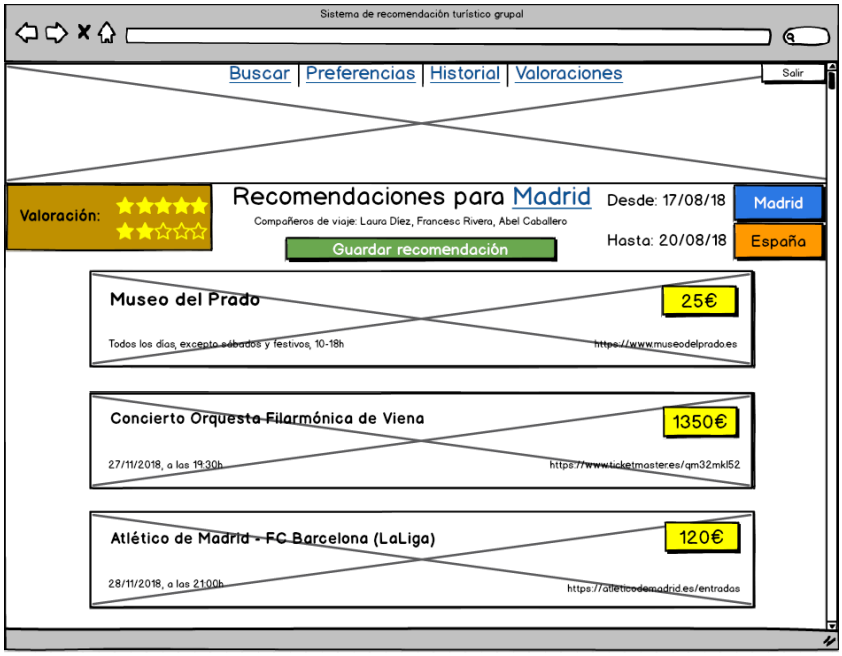

Figura 31: Listado de recomendación

Página a la que llegaremos después de solicitar una recomendación, donde se mostrarán los ítems recabados y dará la posibilidad de valorarla.

Figura 32: Detalles de ítem

Página específica de un ítem donde veremos más detalles del mismo.

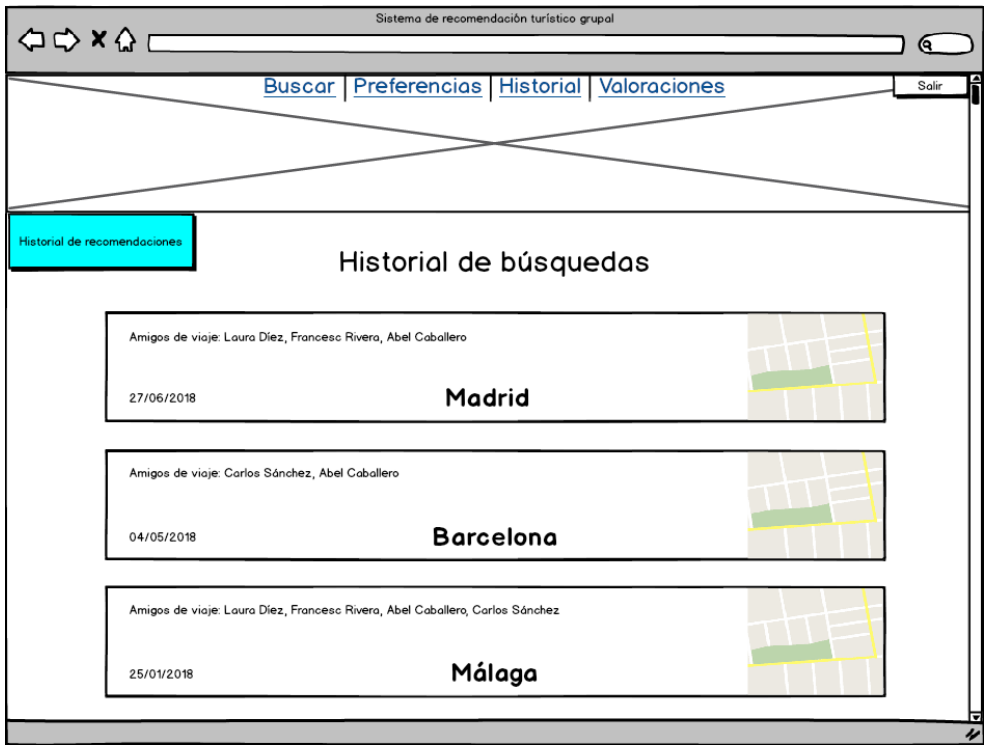

Figura 33: Historial de búsquedas de recomendación

La lista de recomendaciones en las que el usuario navegante se ha visto partícipe.

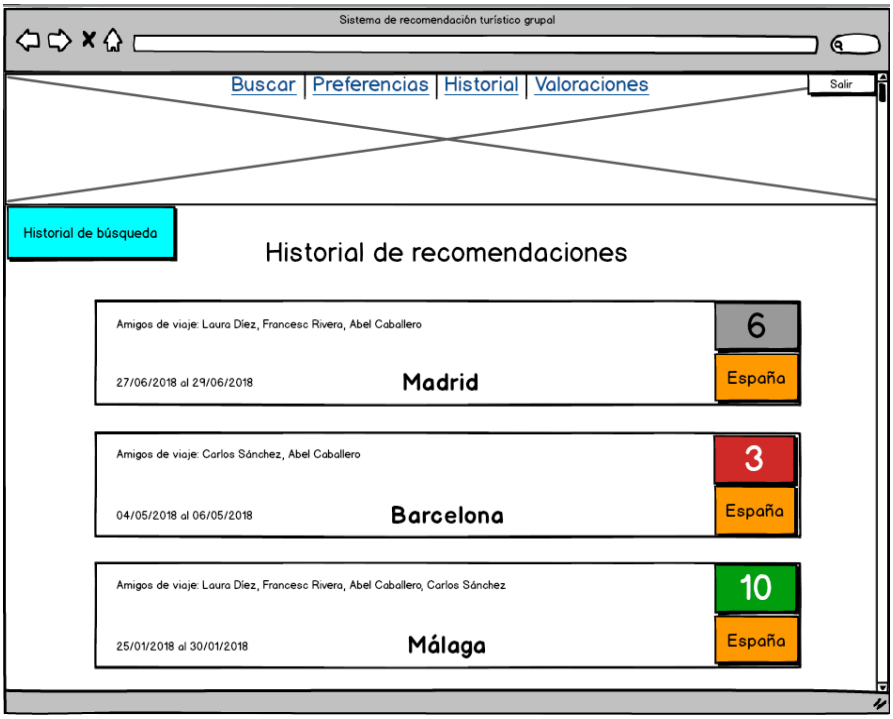

Figura 34: Historial de recomendaciones para el usuario

Misma página de antes, en la que veremos la valoración dada a cada recomendación si esta existe.

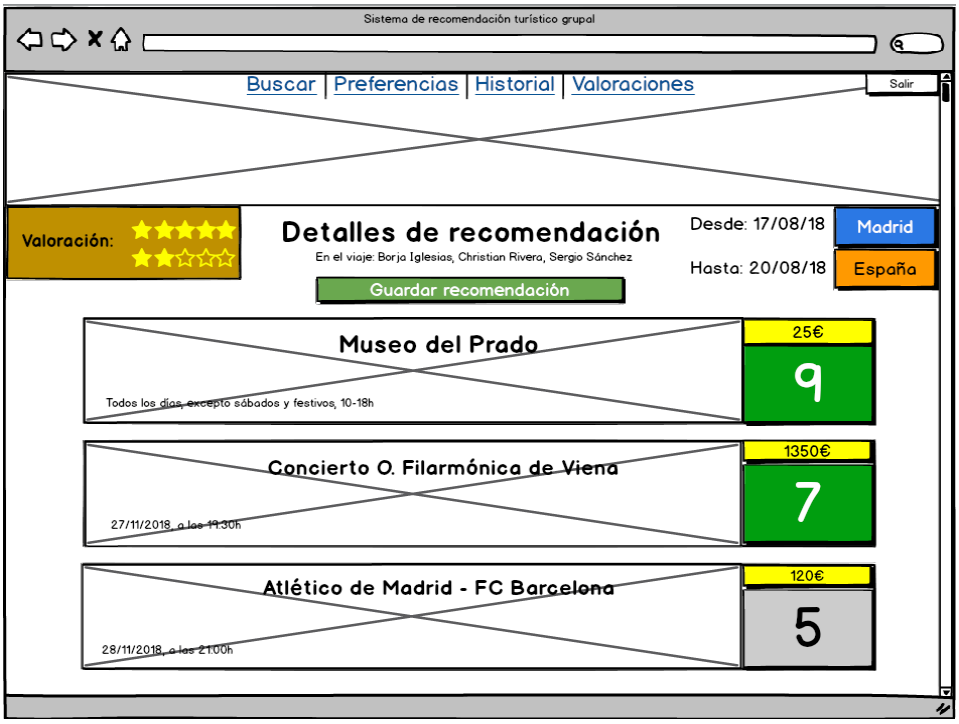

Figura 35: Detalles de recomendación guardada

Tras acceder a una valoración, podremos verla de nuevo desde el histórico.

Figura 36: Preferencias de usuario

Página en la que el usuario podrá dar definición a sus preferencias.

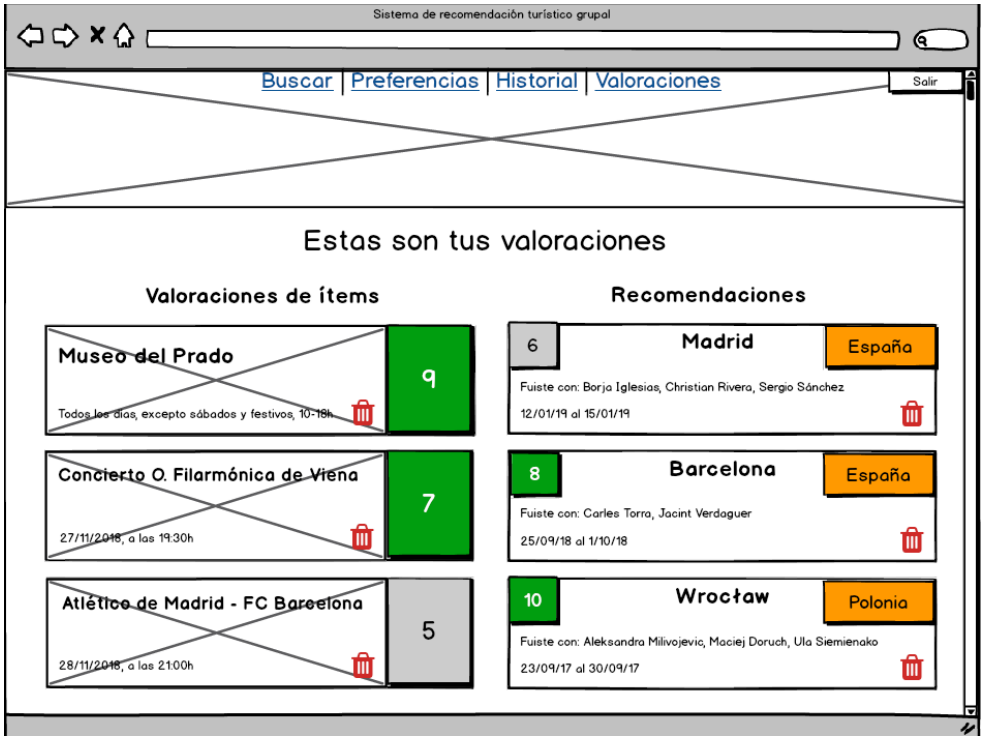

Figura 37: Valoraciones de usuario

Página donde agruparemos las valoraciones de todo tipo expresadas por el usuario.

7

Arquitectura

En este punto se aportarán detalles y se ahondará en explicaciones acerca de la arquitectura de la aplicación y de cómo se organizan a grandes rasgos sus capas lógicas. Al tratarse de una aplicación web, esta misma seguirá el stack de protocolos de redes, residiendo concretamente en la capa de aplicación. Entendiendo que el encapsulamiento se gestione a través de una swarm de **Docker** [12] (conjunto orquestado de nodos físicos que se encargarán de seleccionar destinos para los servicios o partes de la aplicación y estabilizar el ecosistema en base a los parámetros configurados), se propondrá que idealmente la aplicación habría de tener un dominio DNS concreto, con una redirección interna mediante un sistema de enrutamiento (pudiendo usar la herramienta open source **Traefik** [14]) que redireccione las peticiones recibidas y las comunicaciones entre partes del ecosistema de la aplicación.

Una de las grandes ventajas de **Docker** como sistema de desarrollo y puesta en producción que nos gustaría resaltar es la capacidad de facilitar un entorno reproducible con facilidad, haciendo olvidar el tedio que supondría la configuración inicial de un proyecto concreto para cualquier paso del proceso de implementación, además de la facilidad de desarrollar en consonancia distintos procesos y servicios como parte de un todo organizado.

A continuación (Figura 38), presentaremos una vista general de la arquitectura de la aplicación en un diagrama con los puntos más importantes:

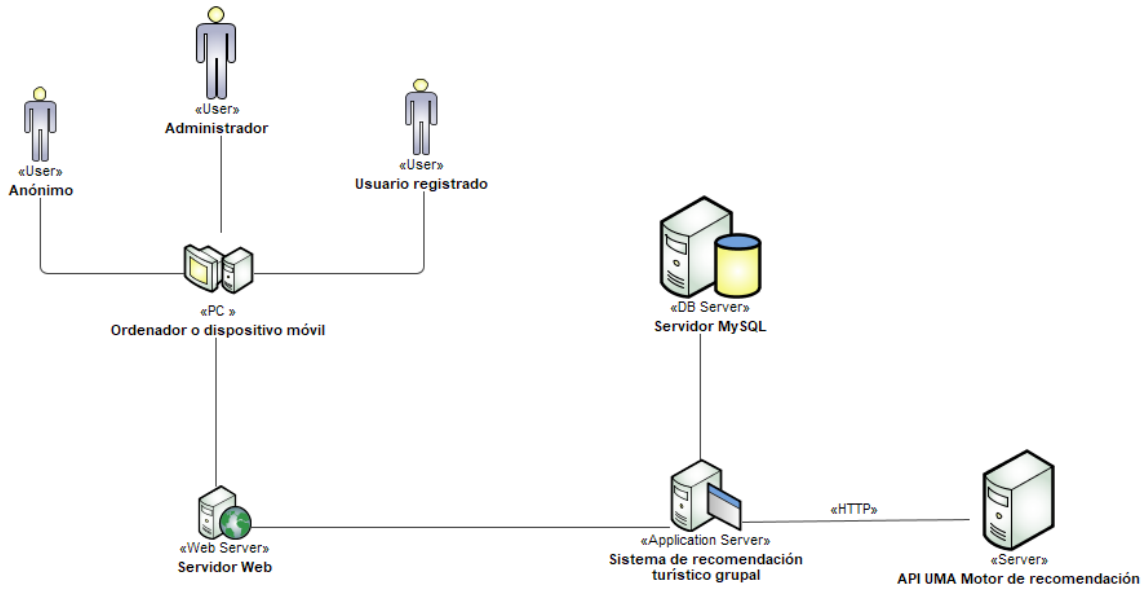

Figura 38: Arquitectura de la aplicación

Es importante reseñar que todos los actores van a utilizar la misma aplicación, y no haremos una diferenciación especial según el rol del mismo, solamente tendrán acceso separado a diferentes partes de ella. A través del dispositivo o navegador web será capaz de acceder al sistema, que a su vez ejerce su funcionamiento apoyándose en una base de datos **MySQL** [15] y que contacta asiduamente con la API de la UMA para establecer las configuraciones y devolver los resultados de las recomendaciones.

A nivel de stack tecnológico, al hacer un zoom en la parte desarrollada a fondo en este proyecto podemos presentar otro tipo de diagrama (Figura 39), más ilustrativo en cuanto a tecnologías usadas y que hace hincapié en la separación por partes con niveles de impacto distinto en un entorno orquestado en las aplicaciones modernas, el back-end y el front-end.

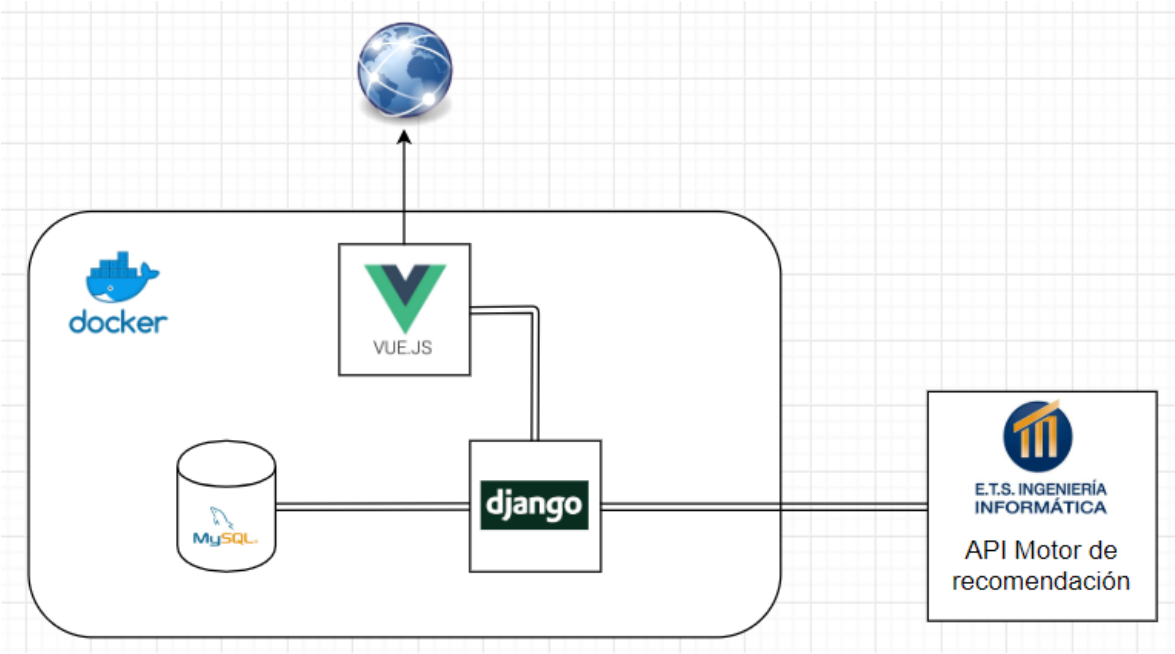

Figura 39: Stack tecnológico básico

En este caso la separación de la herramienta de "contenedorización" de **Docker** [12], antes explicada brevemente, nos permite organizar los servicios de esta manera. Principalmente serán tres, aunque la escalabilidad del sistema nos permitirá replicar cuantos sean necesarios en función de la carga recibida sin tener que hacer enfocarnos demasiado en la eficiencia del código (que también debería ser, a todas luces, importante en cualquier tipo de desarrollo software).

El front-end, desarrollado en **JavaScript** como código ejecutado en el ordenador del cliente [16], con el inestimable apoyo del framework **Vue.js** [11], que nos permitirá el desarrollo basándonos en componentes ligeros, reutilizables y cargados de poder a través de la inyección de valores desde las capas superiores de los mismos, reduciendo al mínimo la duplicidad de código.

El back-end, siendo la parte básica de la aplicación, contendrá todo desarrollo orientado puramente a gestionar las lógicas de negocio, la interacción con la base de datos, la comunicación con la API del motor de recomendación y donde presentará otra API REST que es la que se comunicará con el front-end para la introducción y recabado de información fruto del uso de la aplicación. La API REST es un tipo especial de API en la que

se realiza una representación de objetos o modelos de bases de datos en este caso, permitiendo las acciones de CRUD sobre ellos a través de una paralelización con los tipos de métodos del sistema HTTP:

- El método de POST que estará asociado a la creación de nuevos objetos.
- El método de GET que estará asociado a la recogida de objetos existentes en la base de datos.
- Los métodos PUT/PATCH que estarán asociados a la modificación de objetos en la base de datos.
- El método DELETE que estará asociado a la eliminación de objetos.

Toda esta especificación [17] de peticiones HTTP conlleva unos acuerdos específicos en términos de formato entre el front-end y el back-end, posibilitando la fácil serialización de datos desde las lógicas de este último.

Además, en una API REST no existe un estado de ejecución particular de la aplicación, simplemente se deben de tomar peticiones como órdenes desde el front-end siendo estas separadas entre sí, pudiendo existir una memoria caché para el rápido servido de resultados de peticiones idénticas en un corto periodo de tiempo. Se deja abierta la puerta a crear cualquier otro tipo de servicio incluso para repartir aún más la carga, como podría ser la separación del back-end en varios microservicios [18] además de dicha API REST.

8

Implementación

8.1 Introducción

Aquí nos centraremos en cómo hemos desarrollado este proyecto software desde el punto de vista de la implementación, ahondando en las partes del código relevantes para la explicación y explicando las estructuras y patrones seguidos. Se han seguido dos proyectos diferentes para el back-end y para el front-end. En esta primera parte, las explicaciones se centrarán en el proyecto back-end.

8.2 Estructura del proyecto back-end

Hemos seguido una estructura clásica de proyecto **Django** [13] separada por aplicaciones (Figura 40). La carpeta *recommender* contendrá la lógica específica, *settings* las configuraciones del proyecto específicas a cada entorno de despliegue. El resto de archivos nos permitirá el lanzamiento de la aplicación a través de especialmente los archivos *Dockerfile* y *deploy.yml*.

Es importante destacar también el papel del archivo *requirements.txt*, que será una manera de agrupar las propias dependencias de librerías y frameworks del proyecto, actualizable mediante la herramienta open source de control de versiones **Git** [18] y que se ejecutará de manera automática en cada despliegue, actualizando los mismos o manteniéndolos según la versión. *urls.py* es el último a destacar, que contiene los patrones iniciales de acceso a la API REST así como una documentación online del proyecto para desarrolladores front-end. *wsgi.py* nos permitirá el servido en un servidor web adaptado para ello. En el archivo *README.md* vendrá una pequeña guía con los comandos útiles.

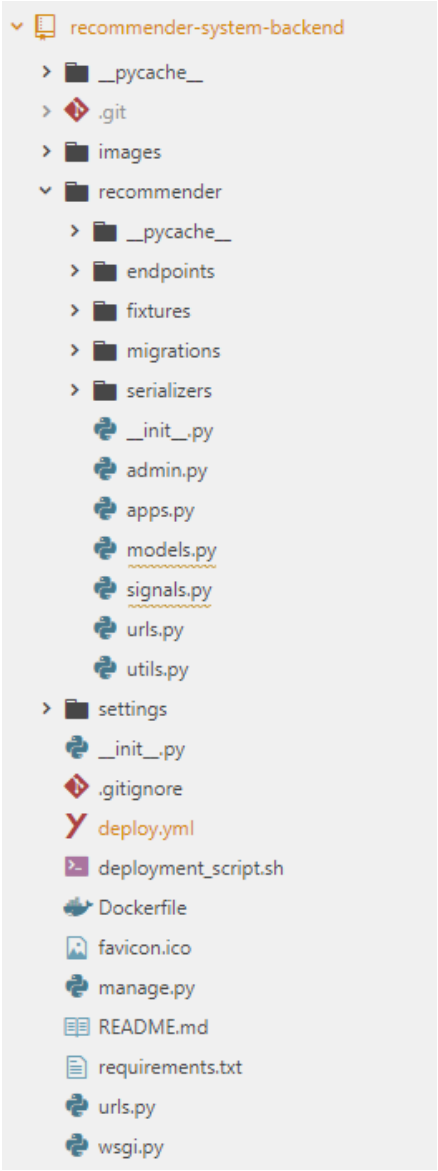

Figura 40: Estructura por archivos del proyecto back-end

Dentro de la carpeta de aplicación *recommender*, es importante comprender el papel del archivo *models.py*, que representará los objetos usados dentro de la aplicación (Figura 41) y los convertirá en tablas de la base de datos mediantes las migraciones, un proceso de generación automática de archivos que modificarán la base de datos para flexibilizarla en función del modelo cada vez que se produce un cambio nuevo en alguna de las clases o tablas.

```
class Recommendation(models.Model):

    start_date = models.DateField(null=True, blank=True, verbose_name='Start date')
    end_date = models.DateField(null=True, blank=True, verbose_name='End date')

    # Relations
    items = models.ManyToManyField('Item', blank=True, verbose_name='Items')
    context = models.OneToOneField('RecommendationContext', on_delete=models.CASCADE, related_name='recommendations', verbose_name='Context')
    group = models.ForeignKey('Group', on_delete=models.CASCADE, related_name='recommendations', verbose_name='Group')
    city = models.ForeignKey('City', on_delete=models.CASCADE, related_name='recommendations', verbose_name='City')

    def __str__(self):
        return str(self.id)


class Valoration(models.Model):

    comment = models.TextField(blank=True, max_length=1000, verbose_name='Comment')
    score = models.IntegerField(default=0, verbose_name='Score')

    # Relations
    user = models.ForeignKey('CustomUser', on_delete=models.CASCADE, related_name='valorations', verbose_name='User')
    item = models.ForeignKey('Item', null=True, on_delete=models.CASCADE, related_name='valorations', verbose_name='Valorated item')
    recommendation = models.ForeignKey('Recommendation', null=True, on_delete=models.CASCADE, related_name='valorations', verbose_name='Recommendation')

    def __str__(self):
        return str(self.id)
```

Figura 41: Ejemplos de modelos de Recommendation y Valoration

La carpeta *migrations* contendrá el histórico de cambios (Figura 42) sufridos a través de los scripts por la base de datos que explicarán su estado actual. Para hacer un cambio en alguno de los modelos, deberá después crearse migraciones nuevas con el comando "*python manage.py makemigrations"* y efectuarlas con el comando "*python manage.py migrate*".

```python
from django.db import migrations, models


class Migration(migrations.Migration):

    dependencies = [
        ('recommender', '0004_auto_20190617_1728'),
    ]

    operations = [
        migrations.AlterField(
            model_name='antecedent',
            name='value',
            field=models.DecimalField(decimal_places=2, max_digits=10, null=True, verbose_name='Value'),
        ),
        migrations.AlterField(
            model_name='consequent',
            name='value',
            field=models.DecimalField(decimal_places=2, max_digits=10, null=True, verbose_name='Value'),
        ),
        migrations.AlterField(
            model_name='pertenancegrade',
            name='value',
            field=models.DecimalField(decimal_places=2, max_digits=10, null=True, verbose_name='Value'),
        ),
        migrations.AlterField(
            model_name='usercontext',
            name='weight',
            field=models.DecimalField(decimal_places=2, max_digits=10, null=True, verbose_name='Weight'),
        ),
    ]
```

Figura 42: Ejemplos de operaciones en una migración

La carpeta *endpoints* contendrá las vistas de la aplicación que servirán los puntos de acceso REST para las aplicaciones (Figura 43). Todo esto es posible gracias a nuestra implementación adaptada de la librería *django-rest-framework* [20], que convertirá estas clases de vistas en adaptaciones REST para la gestión de nuestros modelos definidos en *models.py*. Estas vistas estarán presentes en *recommender/urls.py* (Figura 44), como el módulo de gestión interno de las URIs para cada uno de los modelos existentes. Es importante destacar también el papel de la carpeta *serializers*, que a su vez contendrá los formatos necesarios para la entrada o salida de datos de las vistas además de para otras operaciones internas como puede ser la formalización de un formato adecuado para la comunicación de información entre el back-end y la API del motor de recomendación.

```python
class ItemView(mixins.ListModelMixin, mixins.CreateModelMixin, generics.GenericAPIView):

    queryset = Item.objects.all()
    serializer_class = ItemSerializer

    def get(self, request, *args, **kwargs):
        return self.list(request, *args, **kwargs)

    def post(self, request, *args, **kwargs):
        return self.create(request, *args, **kwargs)


class ItemAttributeView(mixins.ListModelMixin, mixins.CreateModelMixin, generics.GenericAPIView):

    queryset = ItemAttribute.objects.all()
    serializer_class = ItemAttributeSerializer

    def get(self, request, *args, **kwargs):
        return self.list(request, *args, **kwargs)

    def post(self, request, *args, **kwargs):
        return self.create(request, *args, **kwargs)
```

Figura 43: Ejemplos de vistas en item.py

```python
api_urls = [
    # User
    url(r'^users$', UserView.as_view(), name='userFrontView'),
    url(r'^users/login$', csrf_exempt(LoginView.as_view()), name='loginFrontView'),
    url(r'^users/logout$', LogoutView.as_view(), name='logoutFrontView'),
    url(r'^users/register$', RegisterView.as_view(), name='registerFrontView'),
    url(r'^users/forgot_password$', ForgotPasswordView.as_view(), name='forgotPasswordFrontView'),
    url(r'^users/change_password$', ChangePasswordView.as_view(), name='changePasswordFrontView'),

    # Group
    url(r'^groups$', GroupView.as_view(), name='groupFrontView'),

    # Item
    url(r'^items$', ItemView.as_view(), name='itemFrontView'),
    url(r'^item_attributes$', ItemAttributeView.as_view(), name='itemAttributeFrontView'),
    url(r'^attribute_categories$', AttributeCategoryView.as_view(), name='attributeCategoryFrontView'),
    url(r'^pertenance_grades$', PertenanceGradeView.as_view(), name='pertenanceGradeFrontView'),
    url(r'^cities$', CityView.as_view(), name='cityFrontView'),

    # Recommendation
    url(r'^recommendations$', RecommendationView.as_view(), name='recommendationFrontView'),
    url(r'^recommendations/(?P<pk>.+)$', RecommendationSingleView.as_view(), name='recommendationFrontView'),

    # Context
    url(r'^context_segments$', ContextSegmentView.as_view(), name='contextSegmentFrontView'),
    url(r'^user_contexts$', UserContextView.as_view(), name='userContextFrontView'),
    url(r'^recommendation_contexts$', RecommendationContextView.as_view(), name='recommendationContextFrontView'),
    url(r'^implications$', ImplicationView.as_view(), name='implicationFrontView'),

    # Valoration
    url(r'^valorations$', ValorationView.as_view(), name='valorationFrontView'),

    # Choices
    url(r'^users/choices$', UserChoicesView.as_view(), name='userChoicesFrontView'),
    url(r'^cities/choices$', CityChoicesView.as_view(), name='cityChoicesFrontView'),
    url(r'^context_segments/choices$', ContextSegmentChoicesView.as_view(), name='contextSegmentChoicesFrontView'),
]
```

Figura 44: URLs disponibles en la aplicación

Dicha comunicación residirá mayormente en el archivo de *signals.py* (Figura 45), en el que mediante el uso de señales el sistema será capaz de automáticamente configurar el motor de recomendación a través de su API gracias a las funciones contenidas aquí que se ejecutarán al producirse determinados escenarios como la creación de un ítem o una regla lógica de implicación. Apoyada también en parte por el archivo *utils.py* que formalizará otra de las capas lógicas de la comunicación con el motor (Figura 46), en este caso la parte final de manejo de peticiones HTTP.

```python
# Create or update an attribute-value pair
@receiver(models.signals.post_save, sender=PertenanceGrade)
def create_or_update_attribute_pertenance_pair(sender, instance, created=False, **kwargs):
    url = settings.RECOMMENDER_API + 'recommender/saveItem?'
    params = {
        "attribute": instance.item_attribute.name,
        "itemsset": instance.item.city.name,
        "name": instance.item.name,
        "system": "tfgangel",
        "value": instance.value,
    }

    url_params = add_query_params(url, params)

    RequestUtils.postRequest(url_params, {})


# Create or update an Antecedent
@receiver(models.signals.post_save, sender=Antecedent)
def create_or_update_antecedent(sender, instance, created=False, **kwargs):
    url = settings.RECOMMENDER_API + 'recommender/saveRule?'
    params = {
        "attribute": instance.item_attribute.name,
        "part": 'a',
        "ruledesc": 'Antecedente de ' + instance.implication.context_segment.domain + ' para ' + instance.item_attribute.name,
        'rulesset': instance.implication.context_segment.dimension,
        "system": "tfgangel",
        "value": instance.value,
    }

    url_params = add_query_params(url, params)

    RequestUtils.postRequest(url_params, {})
```

Figura 45: Ejemplos de funciones de señales

```python
class RequestUtils(object):

    @staticmethod
    def getRequest(url):
        headers = {"Content-Type": "application/json", "Accept": "application/json", "Authorization": "Bearer " + settings.RECOMMENDER_API_TOKEN}
        try:
            response = requests.get(url, "", headers=headers, timeout=60)
            return response
        except Exception as e:
            logging.error(str(e) + " : Get response")
            return Response({"error": "[ERROR IN GET]" + str(e)}, status=400)

    @staticmethod
    def postRequest(url, param):
        headers = {"Content-Type": "application/json", "Accept": "application/json", "Authorization": "Bearer " + settings.RECOMMENDER_API_TOKEN}
        try:
            response = requests.post(url, param, headers=headers)
            return response
        except Exception as e:
            logging.error(str(e) + " : Post response")
            return Response({"error": "[ERROR IN POST]" + str(e)}, status=400)

    @staticmethod
    def deleteRequest(url):
        headers = {"Content-Type": "application/json", "Accept": "application/json", "Authorization": "Bearer " + settings.RECOMMENDER_API_TOKEN}
        try:
            response = requests.delete(url, {}, headers=headers)
            return response
        except Exception as e:
            logging.error(str(e) + " : Delete response")
            return Response({"error": "[ERROR IN DELETE]" + str(e)}, status=400)
```

Figura 46: Archivo utils.py

Por último, destacar el archivo *admin.py* que habilitará el panel de mandos de los *Administradores*, accedido tras la autenticación, y que posibilitará una gestión total de todas las entidades existentes en la base de datos del sistema.

8.3 Estructura del proyecto front-end

Aquí la estructura a seguir cambia bastante en relación a la del proyecto back-end. La maquetación progresiva irá completamente dentro de la carpeta *src*, siendo la carpeta *static* sólo para situar determinados archivos de imagen que se espera que no cambien con asiduidad. La carpeta *config* contendrá diferentes archivos con variables para la configuración de entornos, relacionada íntimamente con la carpeta *build* en la que a través del paquete de módulos de **Webpack** [21] tendremos los scripts principales en los que desplegaremos la aplicación front-end. Dichos scripts serán ejecutados a través de la capa superior de infraestructura proporcionada por **Docker**, que vendrá definida en los archivos *Dockerfile, deployment_script.sh* y *deploy.yml*.

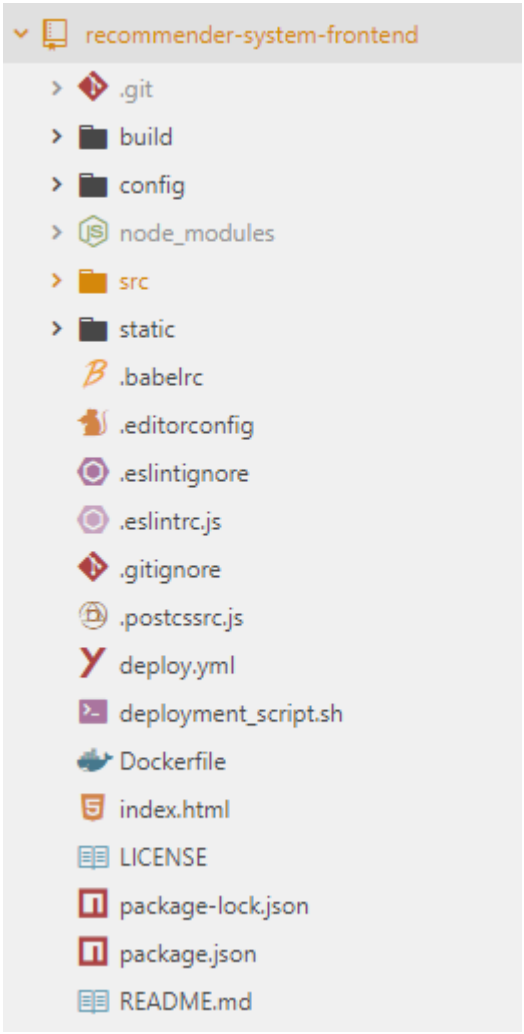

Figura 47: Estructura por archivos del proyecto front-end

También a destacar los archivos de *package-lock.json* y *package.json* que contendrán todas las dependencias necesarias para hacer funcionar el proyecto y que, de nuevo, estarán automatizadas dentro de los procesos específicos para hacer funcionar la aplicación. En el archivo *README.md* vendrá una pequeña guía con comandos útiles.

Indagando un poco más dentro de la carpeta de *src* (Figura 47) observamos cuatro piezas clave:

- *App.vue* y *main.js*, los archivos principales que serán parientes del resto. Esto es así porque la estructura de proyecto en **Vue.js** se asemeja a la estructura de datos del árbol, siendo *App.vue* el nodo padre superior. En este archivo se podrán situar

reglas CSS que sean comunes a todos los componentes de la aplicación, además de inicializar otras librerías menores y parámetros propios de **Vue.js**.

- La carpeta *store* contendrá la tienda, una memoria global de estado de la aplicación que actuará como una nube que será accesible e interaccionable no importa desde el componente en el que nos encontremos y en la que residirán valores como el token de autenticación, nuestro email de usuario o la URL base de la API con la que contactamos.
- *index.js* dentro de la carpeta *router*, aquí tendremos un navegador interactivo que contendrá las reglas de las páginas de la aplicación, señalando a donde debemos ir cada vez que tengamos que pasar por un cambio de página.
- La carpeta *components* que contendrá todos los componentes reutilizables dentro de la aplicación (Figura 48).

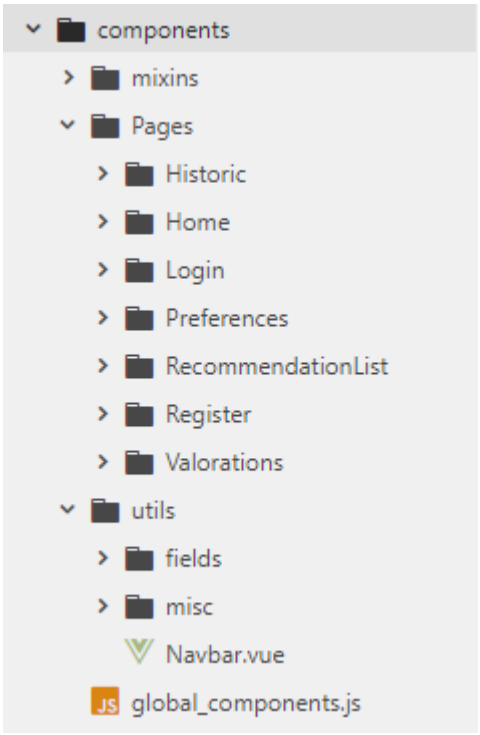

Figura 48: Estructura de la carpeta components

Aquí la carpeta *Pages* será la más importante, donde tendremos realmente todos los componentes específicos de cada página que muchas veces son una página en sí, y que serán gestionados por el *router* para decidir, según la URL a la que accedamos, qué archivos utilizar para mostrar en el navegador. Dentro de *utils* encontraremos la

definición de componentes que serán usados por toda la aplicación, así como en *global_components.js* encontraremos la inicialización de los mismos para poder ser usados en todos los componentes sin tener que declararlos previamente. La carpeta *mixins.js* contiene archivos de utilidades varios como funciones globales de formateo y los helpers utilizados para contactar con la API del back-end de la aplicación (Figura 49). Usaremos la implementación de **axios** [22] para ello por la facilidad de configuración que nos ofrece.

```
import axios from 'axios';

axios.defaults.xsrfCookieName = 'csrftoken'
axios.defaults.xsrfHeaderName = "X-CSRFTOKEN"

export default {
    data() {
        return {
            requestHandler: axios.create({
                baseURL: this.$store.state.api.domain,
                headers: {
                    'Authorization': 'Token ' + this.$store.state.api.token,
                    'X-CSRFToken': this.$store.state.api.csrftoken || "",
                    'Content-Type': 'application/json',
                    'Accept': 'application/json'
                }
            })
        }
    },
    methods: {
        async apiGet(url) {
            return await this.requestHandler.get(url)
        },
        async apiDel(url) {
            return await this.requestHandler.delete(url)
        },
        async apiPatch(url, data) {
            return await this.requestHandler.patch(url, data)
        },
        async apiPost(url, data) {
            return await this.requestHandler.post(url, data)
        },
    }
}
```

Figura 49: Interior del archivo requests.js

La estructura de un archivo en el framework de **Vue.js** es muy sencilla de entender. Consta de tres secciones, la sección HTML entre etiquetas *template*, en la que se define el esqueleto del componente, se inyectan variables, otros componentes y el las referencias a las clases CSS (Figura 50). La inyección de variables se hace escribiendo ':'

antes del parámetro en las opciones de la estructura HTML. Los componentes se utilizarán llamándolos por la etiqueta previamente definida para cada uno, y deberán estar definidos previamente en la siguiente parte o importados de forma global.

```
<template>
    <div class="login d-flex align-items-center">
        <b-container>
            <b-row><b-col class="title">Bienvenido</b-col></b-row>
            <b-container id="login-form">
                <b-form-group class="form-validation">
                    <b-row><b-col>
                        <input-field placeholder="alumno@ejemplo.com" v-model.trim="login_body.username" @keydown.enter.native="login" type="email"/>
                    </b-col></b-row>
                    <b-row><b-col>
                        <input-field placeholder="••••••••" v-model.trim="login_body.password" type="password" @keydown.enter.native="login"/>
                    </b-col></b-row>
                </b-form-group>
                <b-row v-show="this.feedback !== ''">
                    <b-col>
                        <p class="feedback-error">{{ this.feedback }}</p>
                    </b-col>
                </b-row>
                <b-row>
                    <b-col>
                        <btn size="big" color="green" @click="login()">Acceder</btn>
                    </b-col>
                    <b-col>
                        <btn size="big" color="green" @click="toRegister()">Registrarse</btn>
                    </b-col>
                </b-row>
                <b-row class="align-items-center justify-content-center">
                    <btn size="medium" color="blue" @click="toForgotPassword()">Olvidé mi contraseña...</btn>
                </b-row>
            </b-container>
        </b-container>
    </div>
</template>
```

Figura 50: Sección template de componente Login

En la parte intermedia o lógica, entre etiquetas *script* encontramos las importaciones locales de otros componentes, la definición de variables, y las funciones referentes al componente. Estas funciones podrán ser activadas debido a una acción concreta del usuario (click en un botón), a un evento concreto (generación de valores) o al curso de acciones preliminar o inicialización de componente (función *create* o *beforeCreate*) (Figura 51).

```vue
<script>
    export default {
        name: 'Login',
        data () {
            return {
                login_body: {},
                feedback: '',
            }
        },
        methods: {
            login() {
                this.apiPost('users/login', this.login_body).then(response => {
                    this.$store.dispatch('login', response.data)
                }, response => {
                    if (response.body !== undefined && response.body !== null) {
                        this.feedback = response.body.error;
                    } else {
                        this.feedback = 'Tus credenciales no son correctas!'
                    }
                });
            },
            toRegister() {
                this.$router.push({ name: 'Register' })
            },
            toForgotPassword() {
                this.$router.push({ name: 'ForgotPassword' })
            },
        },
        beforeCreate: function(state) {
            if (this.$route.query.logout !== undefined && this.$route.query.logout !== false && this.$store.state.logged_in) {
                this.$store.dispatch('logout');
            } else if (this.$store.state.logged_in) {
                this.$router.push({ name: 'Home' })
            } else {
                this.$store.dispatch('focusSection', "Login")
            }
        },
    }
</script>
```

Figura 51: Sección script de componente Login

En la parte final de apariencia, entre etiquetas *style* encontraremos las reglas CSS asociadas a este componente de forma local también. Definiremos las propias de la manera que más nos convenga (Figura 52).

```vue
<style lang="scss" scoped>
    @import 'src/assets/css/global.scss';

    .login {
        width: 100%;
        min-height: 100vh;
        background-color: $white;
        text-align: center;
        .title {
            text-align: center;
            margin-bottom: 10px;
        }
        #login-form {
            max-width: 400px;
        }
    }

</style>
```

Figura 52: Sección style de componente Login

Sería interesante desarrollar algo que nos permitiese la recogida del nuevo token por intervalo de tiempo, ya que este caduca y actualmente lo tenemos simplemente copiado y pegado en el archivo de configuración en el back-end, *settings/config.py.*

8.4 Pruebas de software

Todo proyecto de software debe ir aparejado de una serie de pruebas que certifiquen la calidad suficiente del desarrollo aportado. Normalmente, estas también sirven para el futuro desarrollo, guiando a los colaboradores y evitando que las nuevos incrementos introduzcan errores o funcionamiento inesperado sobre funcionalidades que ya presenta la aplicación sin que nos demos cuenta. Hemos desarrollado una serie de pruebas automatizadas para la parte de back-end, la más crítica de la aplicación y que interviene en las lógicas de comunicación con el motor de recomendación, que se encuentran en el archivo *recommender/tests.py*.

Estas pruebas se centran en las comunicaciones con la API del motor de recomendación y algunos otros procesos internos. Para realizar las pruebas automáticas, solo necesitamos ejecutar el comando dentro del contenedor que ejecuta el servidor back-end el comando '*python manage.py test*'.

9

Evaluación del sistema

En este apartado realizaremos una evaluación del desempeño del sistema en el objetivo principal de este proyecto, prestando atención a la relevancia de los resultados obtenidos para discernir si están al nivel adecuado.

El proceso de generación de recomendaciones de ítems turísticos sobre el que gira la aplicación se encuentra íntimamente ligado al motor de recomendaciones basado en implicaciones desarrollado por el grupo de investigación SICUMA, cuyo funcionamiento ha sido explicado durante diversos puntos a lo largo de esta memoria. Aun así, es preciso recordar que el paso de la configuración del sistema es un requerimiento esencial previo a cualquier tipo de uso de la aplicación. En la aplicación, cada segmento tendrá un conjunto de implicaciones definidas por un experto en la materia, que también definirá los atributos de los ítems y sus valores, estableciendo así dicha configuración.

Por lo tanto, el primer paso a seguir será el registro de un dataset de ítems turístico para uno de los dominios existentes o ciudades desde las que recomendar, referente a la

ciudad de Barcelona en nuestro caso. Además de esto se introducirán valores de 0.0 a 0.1 para cada uno de los ítems en los atributos que vamos a definir para este ejemplo, que serán:

1. *Precio*: El valor asignado será inversamente proporcional al precio del ítem, siendo más alto cuanto más barato sea.
2. *Valor cultural*: Se asignará un valor en función del nivel en que el ítem representa una referencia artística y/o social para el destino.
3. *Concurrencia*: El valor vendrá definido en función de la afluencia de gente que en promedio suele visitar el ítem.
4. *Centricidad*: El valor aquí será proporcional a la cercanía respecto al centro neurálgico de la ciudad en el que el ítem será situado.
5. *Ocio musical*: El valor asignado se definirá en función de lo interesante que sea el ítem a nivel musical o de divertimento relacionado con éste.
6. *Naturaleza*: El valor vendrá definido en función de la relación del mismo con la existencia de entidades animales o vegetales.
7. *Actividad deportiva*: El valor aquí irá en conjunción con lo relacionado que esté el ítem con cualquier evento o actividad deportiva.

El siguiente paso será definir los contextos que vamos a utilizar. Hemos elegido seis tipos distintos de contextos, diferentes entre unos y otros en el dominio que especifican, y agrupados por dimensión:

1. *Relax*: En este contexto se definirán los contextos cuya finalidad sea puramente de desconexión vacacional, buscando la liberación de estrés y la recarga de energías.
2. *Festivo*: En este contexto se definirá el carácter de ocio festivo que se tenga sobre el viaje, ya sea de celebración o de pura diversión.
3. *Familiar*: En este contexto se medirán en qué grado los viajes se producirán con un grupo de miembros allegados de la propia familia.
4. *Laboral*: En este contexto se medirán en qué grado los viajes se producirán con un grupo de compañeros de trabajo.

5. *Cultural*: En este contexto se medirán en qué grado el viaje tiene un objetivo de enriquecimiento o atracción cultural.
6. *Naturaleza*: En este contexto se evaluará a qué nivel se viaja para estar en contacto con la naturaleza.

Por último, quedará por definir un conjunto de reglas o implicaciones que se asignen en concreto a unos determinados valores de contexto. Estas reglas, definidas por antecedentes y consecuentes que llevarán asignado un valor en relación a un atributo de ítem, se establecen tomando en cuenta ciertos factores característicos a la situación a evaluar. Por ejemplo, si el contexto es festivo en alto grado (>0.6), no sería adecuado obtener ítems con un valor alto en el atributo de actividad deportiva, y en cambio lo que tendría sentido es que estemos favoreciendo la recomendación de ítems cuyos valores de atributo asignados para ocio musical sean altos. Definiendo los valores contextuales y las implicaciones para esta evaluación, tendremos:

1. *Familia* = 0.7. Pensada para obtener ítems que sean relacionables fácilmente con un contexto familiar o que incluya infantes. Una regla sería *Naturaleza* = 0.6 → *Precio* = 0.7
2. *Relax* = 0.8. Obtendremos ítems apacibles, que impliquen actividades sosegadas y sin mucho dinamismo. Una regla sería *Naturaleza* = 0.8 → *Valor cultural* = 0.7
3. *Festivo* = 0.9. Obtendremos mayormente ítems enfocados puramente en el ocio grupal. Una regla sería *Concurrencia* = 0.6, *Centricidad* = 0.5 → *Ocio musical* = 0.8
4. *Cultural* = 0.6. Los ítems se verán influenciados en la recomendación por su valor cultural, al haber un interés general en los mismos. Una regla sería ***Vacío*** → *Valor cultural* = 0.6

Con esto terminaríamos la configuración básica de nuestro sistema, listo para solicitar una recomendación adaptada al grupo que vayamos a elegir. Estos grupos se han

agrupado al final en la prueba en el contexto, entonces asumimos que una familia de cuatro personas viajando, siendo dos padres con dos hijos, tomaría el contexto de *Familia* arriba definido. Esto es lo que recibimos al solicitar al motor de recomendación una recomendación reducida de ítems para la ciudad de Barcelona:

Ítem	Actividad deportiva	Centricidad	Concurrencia	Naturaleza	Ocio musical	Precio	Valor cultural
Parque Güell	0.4	0.4	0.8	0.6	0.4	0.8	0.7
Playa de la Barceloneta	0.7	0.3	0.5	0.6	0.1	0.9	0.1
Zoo de Barcelona	0.3	0.6	0.5	0.9	0.3	0.7	0.2
Castillo de Montjuïc	0.4	0.4	0.4	0.6	0.2	0.7	0.8
Bunkers de Carmel	0.6	0.2	0.7	0.6	0.2	0.9	0.4

En total hemos recibido solamente 5 ítems, acortando bastante la amplísima oferta de la que dispone una ciudad tan turística mundialmente como Barcelona, aliviando a bastante a nuestra familia de ejemplo en la difícil tarea de escoger ítems o productos turísticos. Estos resultados se ajustan bastante a un contexto familiar, ya que todas tienen un buen valor en *Precio* además de en el atributo de *Naturaleza*. Creemos que esto se ajusta a la realidad turística de una familia promedio, que suelen preocuparse bastante por su economía familiar y que demandan el contacto con la *Naturaleza* debido a los hijos en edad infantil, de los que se prevé esa predisposición.

Esto es lo que hemos obtenido con las reglas, los contextos y con los ítems definidos en el motor de recomendación según lo explicado arriba. De haber sido más suaves en los

valores de las implicaciones en nuestro contexto, habríamos obtenido un mayor número de resultados, haciendo la recomendación menos útil debido al filtrado menos exhaustivo que presentaría. Otra observación destacada es que a mayor número de reglas definidas en el sistema se perfeccionarían mucho más las recomendaciones a un contexto concreto. Es importante recordar que tampoco se trata de hacerlo al máximo, puesto que, si se llegan a definir implicaciones demasiado estrictas para un contexto, es bastante probable que no encontremos ningún ítem en la recomendación o que la cantidad de estos sea insuficiente para elegir.

10

Conclusiones

Después de la realización de un proyecto tan grande como éste no queda sino evaluar positivamente el aprendizaje que ha supuesto en el desarrollo e implementación de tecnologías de aplicaciones web para la recomendación de ítems turísticos, así como de la comprensión de técnicas y algoritmos actuales de recomendación turística.

En los primeros apartados exploramos las opciones existentes en el estado actual del arte de las recomendaciones, observando que las aplicaciones no se caracterizaban precisamente por ser muy abundantes y las que había eran bastante especializadas. Aun así, fue posible esclarecer desde la parte trabajada un camino sobre el que avanzar, combinando ciertas técnicas y enfoques para el caso de uso que nos afectaba en concreto, el de la recomendación turística grupal.

La siguiente fase del proyecto, la de análisis y diseño permitió configurar por completo cuáles serían las funcionalidades a través del modelado UML con la herramienta MagicDraw, llegando a un nivel de detalle bastante extenso en nuestro afán por cubrir el mayor número de escenarios y de posibilidades. En concreto, los casos de uso son una representación bastante fiel de lo que podríamos llamar “necesidades reales del usuario”, siendo los que mayoritariamente en un proyecto de software terminarán por evidenciarse

de manera más clara. También ha resultado interesante el enfoque dado por la maquetación, haciéndonos olvidar detalles triviales para nuestra parte como pueden ser los detalles a nivel de diseño.

En el desarrollo, la parte más abstracta de todo el proyecto se ha intentado dar una lógica de código limpio y clara modularización para dejar el proyecto en un estado fácilmente continuable gracias a la introducción de formatos y una estructuración clara, incluso a nivel de arquitectura. Enfocado en el uso de tecnologías open source, de fácil acceso para cualquier desarrollador y consolidadas gracias a la enorme comunidad que hay detrás de cada una y que se ocupa de mantenerlas y mejorarlas día a día. Creo que debería ser bueno que si se llegara a implementar de manera profesional se tenga a considerar la separación en microservicios para una mejor gestión de la carga de procesamiento. Tampoco debería olvidarse la posible implementación de un proxy si el número de servicios va en aumento, para facilitar una gestión transparente de la comunicación entre los mismos.

Toda esta planificación ha sido facilitada por el seguimiento de una metodología ágil (inspirada en Scrum), con el gestor de proyectos Taiga como soporte, una sencilla e intuitiva herramienta de gestión de proyectos a través de tareas gestionadas por sprints, dándonos una clara referencia continua en el tiempo del estado de cada uno de los pasos en los que nos encontrábamos y permitiendo de igual manera enfocarse en las cosas más críticas.

El uso de sistemas de recomendación está en continuo auge y mejora en gran parte de las aplicaciones que usa la sociedad en su día a día, generando una experiencia personalizada de forma cada vez más eficiente, seleccionando lo más relevante de un volumen mundial cada vez más grande de datos. Es por esto por lo que resulta de gran utilidad comprender realmente su funcionamiento y qué factores se toman en cuenta para generar una recomendación.

Por último, me gustaría remarcar que trabajar partiendo de resultados a nivel de investigación, con una base teórica fuerte como son las lógicas de implicaciones y el Análisis

Formal de Conceptos ha sido cuanto menos enriquecedor como experiencia, al ahondar más en un ámbito científico de las Ciencias de la Computación que nunca había visto tan de cerca y que me ha ayudado a entender conceptos y procedimientos que estoy seguro de que me serán útiles en mi futuro laboral.

Apéndice A

Manual de implantación

La implementación actual que se provee está pensada para ejecutarse en un entorno con servidores locales. Un posible futuro, si se desea continuar con el desarrollo o alojarla en un servidor dedicado no agregaría demasiada dificultad, al estar basada en contenedores de **Docker**.

Como se ha explicado anteriormente, hace falta tener instalado el programa de **Docker** en el sistema operativo sobre el que vayamos a lanzarlo como requisito principal. Éste ha de dotarse con los permisos adecuados de ejecución para su correcto funcionamiento. También habrá de provisionarse con unos recursos adecuados en función del uso que le vayamos a dar a la implementación. En el caso que presentamos valdrá con los mínimos aceptados.

La implementación a nivel de infraestructura está basada en la utilización de contenedores. Estos contenedores se caracterizan por aislar la capa de las dependencias de cada uno de los servicios gracias a los archivos llamados *Dockerfiles*, permitiendo implementarlos de manera ordenada en una pequeña instancia de un sistema operativo en concreto, Linux en nuestro caso.

Para que la aplicación funcione, tenemos que desplegar las dos partes del proyecto de desarrollo a la vez, el front-end y el back-end. Antes de comenzar con cualquiera de los

despliegues tenemos que asegurarnos de que **Docker** está activo en segundo plano, listo para trabajar en las tareas que le asignemos.

Despliegue del proyecto back-end

Para completar este proceso tenemos que ejecutar en la raíz del proyecto *recommender-system-backend*, con los permisos necesarios, el archivo denominado *deployment_script.sh*. Aunque antes de eso, deberemos editarlo y configurarlo con las variables que hay al principio del mismo. La variable ***WORKING_DIR*** es especialmente importante, pues de ella depende la correcta ejecución de esta tarea. Se deberá rellenar con el camino completo al directorio de la raíz del proyecto back-end.

La primera vez que se ejecute este comando tomará bastante tiempo, pero en los siguientes intentos el tiempo de despliegue se verá ampliamente reducido gracias a la memoria caché de las capas de **Docker**.

Despliegue del proyecto front-end

Se aplica el mismo proceso que el seguido anteriormente para el proyecto de back-end. Se debe de adaptar el archivo *deployment_script.sh* situado en la raíz del proyecto con las variables pertinentes, para a continuación ejecutarlo con los permisos necesarios.

Después de ello, seremos capaces de hacer uso e interaccionar con la aplicación servida en la URL *http://localhost:8080/*.

Manual de usuario

Un usuario que acaba de llegar a la aplicación y no dispone de cuenta, deberá clicar en el botón de registro y rellenar el formulario con todos los campos que aparecen de manera correcta, para registrar su usuario y poder comenzar a usar la aplicación.

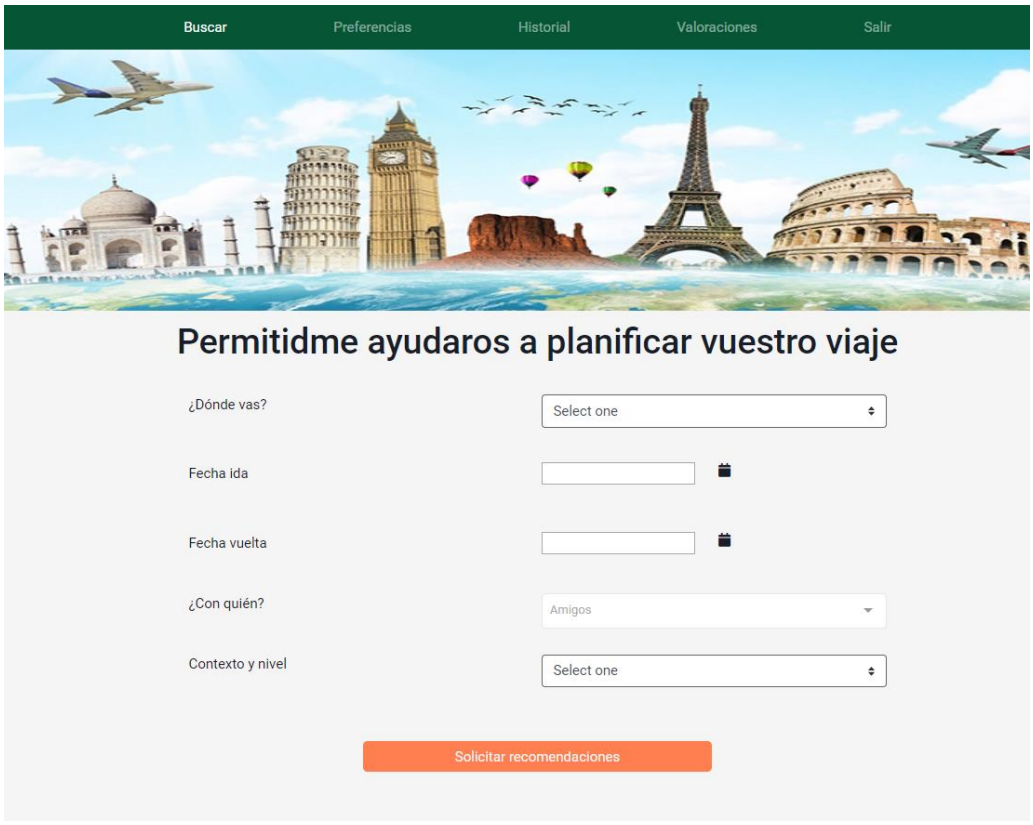

Figura 53: Página principal de la aplicación

En la figura, en la barra de navegación se ven diferenciados cinco apartados distintos: *Buscar*, *Preferencias*, *Historial*, *Valoraciones* y *Salir*. El primero hace referencia, como señala su marcado en la misma figura, a la página principal de búsqueda en la que nos encontramos.

En la página *Preferencias* se podrán gestionar las preferencias y el valor que le va a dar a cada una de las opciones que se le presentan disponibles. Volvemos a usar el patrón estrella como en la explicación anterior.

En la página *Historial* tendremos listadas todo el histórico de recomendaciones de las que se ha sido partícipe. Habrá una previsualización sobre la recomendación, pudiendo observar el grupo con el que se solicitó, el destino, las fechas y, de existir, se mostrará la valoración asignada.

En la página *Valoraciones* se podrán observar y modificar todas las valoraciones que se han formalizado a través del usuario que navega.

El apartado *Salir* ejercerá el proceso de cerrado de la cuenta actual en la aplicación y redirigirá a la página de *Login*.

Configuración del sistema

Para que el sistema llegue a ser útil, será absolutamente necesario que un experto en la materia con un usuario en el rol de administrador tenga acceso a la aplicación. El objetivo de este no es más que gestionar las diferentes entidades que dan sentido a la comunicación con la API del motor de recomendación.

En la Figura 54 mostramos como se verá el panel de control de la aplicación para un usuario administrador cualquiera. Estos usuarios tendrán un control total a nivel de gestión sobre los datos que van a persistir en cualquier ámbito, ya sean los propios ítems turísticos, sus atributos, sus asignaciones de valores, los contextos y las reglas.

La configuración del sistema, por lo tanto, se hará interaccionando con este panel en su mayoría. Cualquier acción que se realice aquí, como la creación o la modificación de una implicación, lanzará los disparadores internos de manera automática para establecer los cambios oportunos en el motor de recomendación.

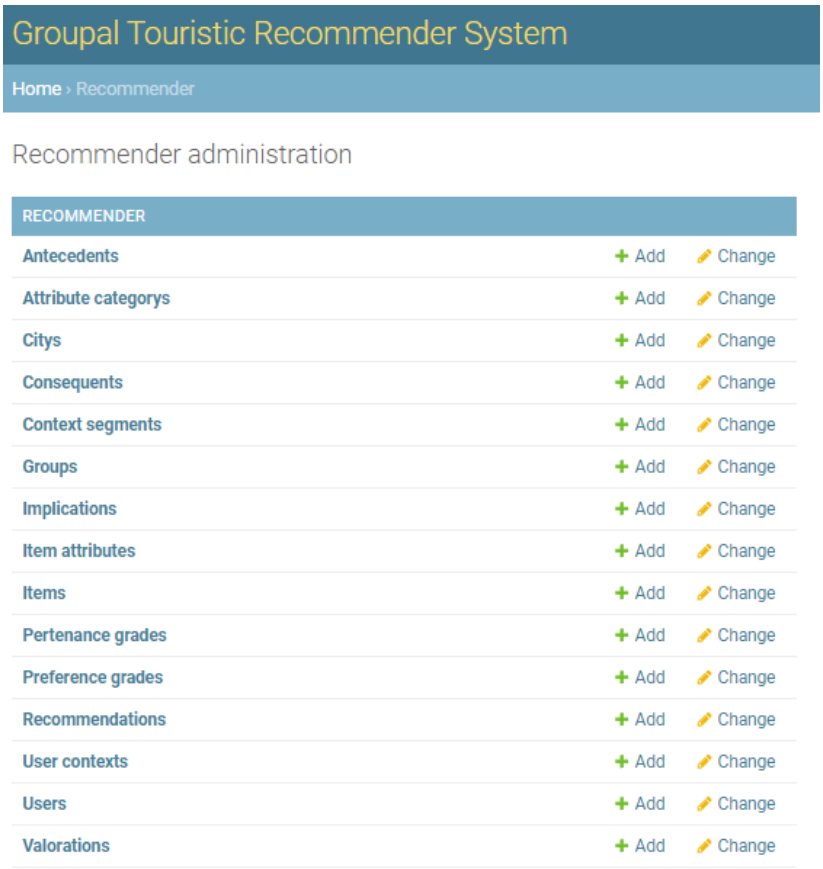

Figura 54: Panel de control de administrador

Dentro de cada tipo de entidad, al acceder a una cualquiera como en la Figura 55, veremos una lista de los elementos que existen en el sistema y que, por consiguiente, se han registrado también en el motor de recomendación.

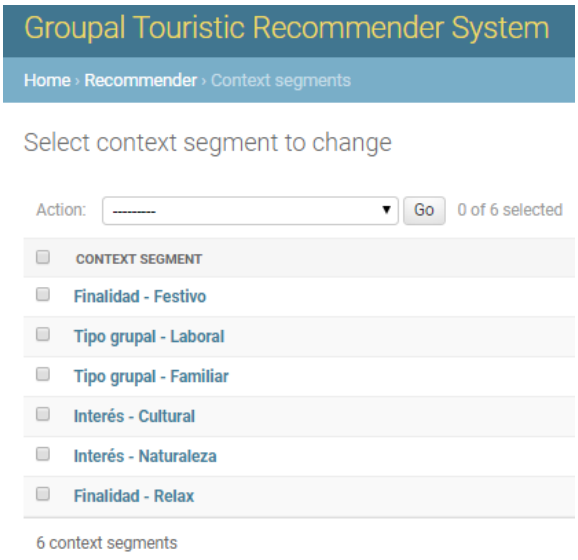

Figura 55: Lista de contextos

Si accedemos a uno de los registros de las entidades, por ejemplo, al de ítem en la Figura 56, observaremos las informaciones que actualmente están en ese tipo de entidad, sus relaciones y tendremos acciones disponibles para la modificación de ese tipo de entidad, la creación de uno nuevo, o el borrado.

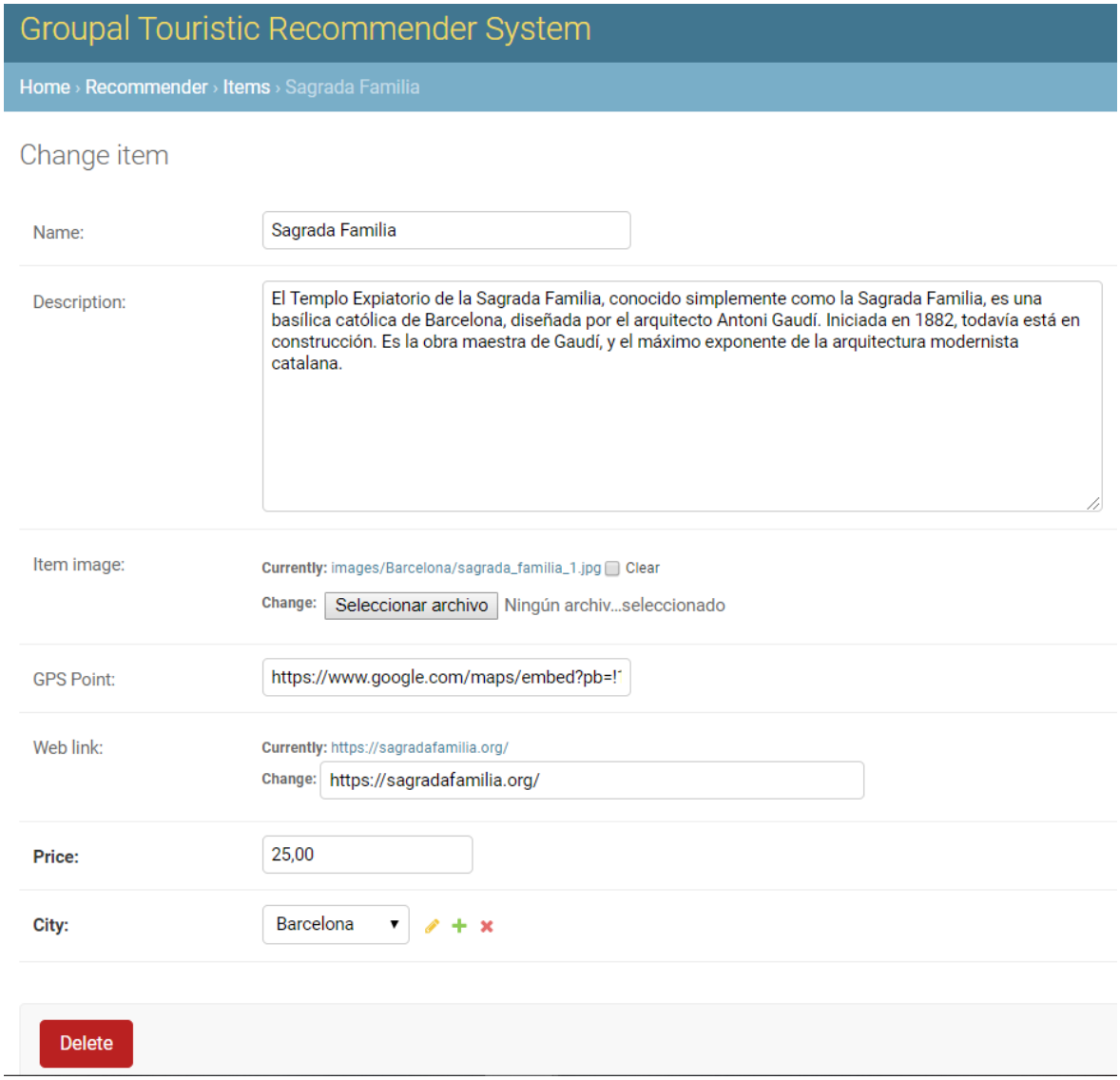

Figura 56: Información de ítem

Ahora que ya conoces las herramientas de las que dispones para usar el sistema y cómo usarlas, el siguiente paso será aplicar los conocimientos en el terreno.

Bibliografía

Referencias bibliográficas

[1] United Nations World Tourism Organization - Tourism Highlights 2018 Edition, https://www.e-unwto.org/doi/pdf/10.18111/9789284419876[3] TripAdvisor - Investor relations and metrics, https://ir.tripadvisor.com/
[4] J. L. Leiva, M. Enciso, C. Rossi, P. Cordero, Á. Mora and A. Guevara. 2013. Improving Recommender Systems with Simplification Logic to Manage Implications with Grades, pp. 292-293, ICSOFT (Selected Papers).
[5] Ludovico Boratto, Group Recommender Systems: State of the Art, Emerging Aspects and Techniques, and Research Challenges, ECIR 2016: 889-892.
[6] HBS Digital Initiative - How Spotify is Changing the Way We Consume Music, https://rctom.hbs.org/submission/discover-weekly-how-spotify-is-changing-the-way-we-consume-music/
[7] TechCrunch - Spotify is building shared-queue Social Listening, https://techcrunch.com/2019/05/31/spotify-social-listening/
[8] DigitalTrends - PlanChat is an all-in-one messaging app that simplifies vacation planning, https://www.digitaltrends.com/mobile/planchat-vacation-planner/
[17] RESTful API Designing guidelines - https://hackernoon.com/restful-api-designing-guidelines-the-best-practices-60e1d954e7c9
[18] How to break a monolith into microservices - https://martinfowler.com/articles/break-monolith-into-microservices.html

Referencias web

[2] SICUMA (Sistemas de Información Cooperativos de la Universidad de Málaga), http://www.sicuma.uma.es/es/pagina-ejemplo
[9] DateNightMovies - What should we watch tonight? https://datenightmovies.com/
[10] balsamiq.com - https://balsamiq.com/
[11] Vue.js, The Progressive JavaScript Framework - https://vuejs.org/
[12] Modernize your applications, accelerate innovation with Docker - https://docker.com
[13] Django web framework https://www.djangoproject.com/
[14] The Cloud Native Edge Router - https://traefik.io
[15] Why MySQL? - https://www.mysql.com/why-mysql/
[16] JavaScript main page - https://www.javascript.com/
[19] Local branching on the cheap - https://git-scm.com/
[20] Powerful and flexible toolkit for building REST APIs - https://www.django-rest-framework.org/
[21] What is Webpack - https://survivejs.com/webpack/what-is-webpack/
[22] Promise based HTTP client for browser and node.js - https://github.com/axios/axios