Trabajo Fin de Grado

Desarrollo de un Prototipo de Aplicación Móvil para Sistemas de Recomendación Proactivos

Development of a Mobile App Prototype for Proactive Recommendation Systems

Autor

Manuel Herrero Gajón

Director

Sergio Ilarri Artigas

ESCUELA DE INGENIERÍA Y ARQUITECTURA
2019

TRABAJOS DE FIN DE GRADO / FIN DE MÁSTER

Escuela de Ingeniería y Arquitectura Universidad Zaragoza

DECLARACIÓN DE AUTORÍA Y ORIGINALIDAD

(Este documento debe acompañar al Trabajo Fin de Grado (TFG)/Trabajo Fin de Máster (TFM) cuando sea depositado para su evaluación).

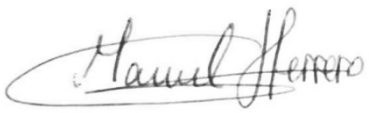
D./Dª. Manuel Herrero Gajón,

con nº de DNI 25203207Z en aplicación de lo dispuesto en el art. 14 (Derechos de autor) del Acuerdo de 11 de septiembre de 2014, del Consejo de Gobierno, por el que se aprueba el Reglamento de los TFG y TFM de la Universidad de Zaragoza,

Declaro que el presente Trabajo de Fin de (Grado/Máster) Grado, (Título del Trabajo) Desarrollo de un Prototipo de Aplicación Móvil para Sistemas de Recomendación Proactivos,

es de mi autoría y es original, no habiéndose utilizado fuente sin ser citada debidamente.

Zaragoza, 24 de enero de 2019



Fdo: Manuel Herrero Gajón

AGRADECIMIENTOS

Tras un largo período de trabajo y aprendizaje me gustaría agradecer a Sergio Ilarri, director de este proyecto por su ayuda y colaboración durante todo el proceso de investigación y elaboración del trabajo, ofreciendo una guía en el desarrollo de la temática y resolviendo todas las dudas surgidas de forma rápida y precisa. También es relevante agradecer a los proyectos TIN2016-78011-C4-3-R (AEI/FEDER, UE), el proyecto Aragón-Aquitania del Gobierno de Aragón PASEO 1.0 (AQ-8), y DGA-FSE (grupo de investigación COS2MOS), que han servido de apoyo durante el trabajo.

Agradezco también a los docentes de la Universidad de Zaragoza por su trabajo durante mi formación y a mis compañeros, que me han ayudado durante estos años de estudio, tanto en los estudios como en lo personal haciendo estos años más sencillos.

Por último, agradezco también el apoyo de mi familia y amigos. Especialmente a mis padres, por su respaldo y confianza durante estos años.

Desarrollo de un Prototipo de Aplicación Móvil para Sistemas de Recomendación Proactivos

RESUMEN

En la actualidad, el exceso de información se puede volver un problema durante el uso de la tecnología. La capacidad de abordar y manejar grandes cantidades de información de las personas es limitada, lo que unido al exceso de contenido irrelevante empaña la usabilidad del sistema. Los sistemas de recomendación juegan un papel clave en esta problemática. Estos sistemas ayudan a los usuarios a elegir entre una gran variedad de diferentes opciones facilitando su toma de decisiones. En los últimos años, gracias a los avances obtenidos en la tecnología móvil ha crecido el interés por los sistemas de recomendaciones para estos usuarios móviles: los llamados *Context-Aware Recommender Systems (CARS)*, pasan a considerar no solo las preferencias del usuario sino también las condiciones de su contexto.

El objetivo del proyecto es continuar esta tendencia, explotando las opciones que ofrece la captura y análisis del contexto para diseñar un sistema proactivo, en el cual el usuario esté totalmente liberado en las tareas de ofrecer una entrada al recomendador y sea automáticamente y gracias a la información de su contexto como se generen recomendaciones interesantes para el mismo. Para ello se utiliza el modelo definido en investigaciones previas realizadas por investigadores de la Universidad de Zaragoza siendo el artículo titulado "Push-Based Recommendations in Mobile Computing Using a Multi-Layer Context" y publicado en el congreso MoMM 2015 donde se propone una arquitectura basada en el concepto de entornos y gestiona el impacto de eventos dinámicos y los actores envueltos en el proceso de recomendaciones.

Una vez analizada la literatura sobre la que se establece el proyecto y visiones auxiliares, se ofrece la arquitectura sobre la que se implementará el proyecto. El trabajo se centra en el desarrollo de un prototipo de aplicación móvil que permite conocer y profundizar sobre el concepto estudiado. Permite conocer las limitaciones y carencias del modelo en la tecnología actual, pero también las ventajas obtenidas en la experiencia de usuario. Además de la aplicación móvil, se realiza el desarrollo del otro componente fundamental del sistema, el gestor de entorno, que incluye el sistema de recomendaciones. Estas implementaciones tienen como objetivo verificar la arquitectura realizada y conocer con mayor profundidad los desafíos actuales en los sistemas de recomendaciones. Asimismo, todo el desarrollo realizado es utilizado también para conocer las diferentes tecnologías disponibles, tanto en el desarrollo de aplicaciones móviles, como en el de sistemas de recomendación.

El proyecto propone una arquitectura sobre la que trabajar con el prototipo de aplicación móvil desarrollado. Ésta aplicación se espera que pueda servir de herramienta para futuras investigaciones orientadas al desarrollo de sistemas de recomendaciones para usuarios móviles, permitiendo conocer el contexto de cada usuario para poder ofrecer recomendaciones de mayor precisión.

Índice general

Índice de figuras

Capítulo 1
Introducción

En este capítulo se ofrece una breve introducción a los sistemas de recomendación que ayuda a comprender la motivación de la realización del proyecto. Además, se reflejan los objetivos y la metodología a seguir durante su transcurso.

1.1 Sistemas de recomendación

Los sistemas de recomendación tienen el objetivo de generar recomendaciones significativas sobre una serie de ítems o productos para un grupo de usuarios para los que puedan resultar interesantes. Son sistemas que a partir de una serie de ítems son capaces de reconocer los intereses de los usuarios para ofrecerles los artículos más recomendables para cada uno. Netflix como recomendador de series o películas, es uno de los sistemas de recomendación más reconocibles dentro de la industria actual [2].

El desarrollo tecnológico, especialmente los avances producidos en la Web como medio de transacciones electrónicas, ha servido como catalizador para facilitar a los usuarios un canal a través del cual pueden expresar su opinión acerca de sus gustos.

El principio de las recomendaciones recae sobre la existente dependencia entre usuarios y acciones centradas en un ítem. Así un usuario interesado en documentales históricos es probable que esté más interesado en documentales sobre naturaleza que en películas de terror. En ocasiones los ítems pueden agruparse en un conjunto como categorías. El objetivo del sistema de recomendación es encontrar las dependencias entre los diferentes artículos a través de los datos recogidos, con el fin de obtener un modelo capaz de realizar predicciones para cada usuario objetivo. Es decir, acumular información para poder inferir los gustos de cada usuario.

El análisis de las recomendaciones se basa en las interacciones pasadas entre usuarios e ítems como indicadores para futuras elecciones [3]. En dicha interacción entre usuario e ítem es importante conocer la opinión del usuario, también conocida como *feedback* o retroalimentación. Esta puede ser darse de forma explícita mediante un sistema de evaluación basado en una puntuación (“me gusta” o “no me gusta” o basado en un sistema de estrellas). Puesto que obtener esta valoración no siempre es posible, el sistema también puede interpretar el *feedback* implícito que obtiene a través de la navegación y acciones realizadas por el usuario.

1.2 Objetivos y alcance del proyecto

Este proyecto aborda el desarrollo de una aplicación para dispositivos móviles que pueda interaccionar con un sistema de recomendación proactivo dependiente del contexto, el trabajo se realiza en colaboración con los proyectos TIN2016-78011-C4-3-R y PASEO 1.0, así como el grupo de investigación COS2MOS.

Se toma como base el estudio “Recomendaciones proactivas en computación móvil utilizando una aproximación contextual multi-capa” [1]: en él se establece una base teórica sobre la que implementar un sistema de recomendación. Este TFG muestra la implementación conjunta de un sistema centrado en dos componentes principales los gestores de entorno y los dispositivos móviles, pero realizando especial énfasis en el componente móvil, las diferentes posibilidades que otorga la computación ubicua y cómo ésta puede aplicarse en el contexto de los sistemas de recomendaciones.

El objetivo del proyecto es obtener una segunda visión sobre el estudio citado y realizar una implementación que permita conocer más en profundidad las limitaciones que tiene la tecnología actual a la hora de poner en práctica la arquitectura propuesta. Este proyecto, aunque resulte enfocado al entorno móvil, estudia una solución para la arquitectura global del sistema con el fin de analizar fielmente las debilidades y fortalezas del diseño e intentar extraer una conclusión convincente que sirva como base para posibles investigaciones posteriores.

1.3 Metodología y organización del proyecto

El proyecto se encuentra divido en cuatro fases diferenciadas: En primer lugar, se realiza un estudio de la literatura en el campo de los sistemas de recomendación enfocados al entorno móvil. La siguiente fase del proyecto se orienta a asociar la literatura estudiada junto al estudio realizado por el grupo de investigación en el que se enmarca el desarrollo de este trabajo para diseñar y plantear la arquitectura que va a tomarse como referencia a lo largo del desarrollo.

La tercera fase se centra en el análisis y la implementación de la aplicación móvil, el análisis de las ventajas de analizar el entorno a través de los dispositivos, las diferentes tecnologías de desarrollo en la industria actual, los requisitos básicos de la aplicación, y la implementación de esta.

La siguiente fase se encuentra centrada en el diseño de un gestor de entorno que incluye un sistema de recomendación básico, cuyo objetivo es completar el sistema, con el fin analizar en mayor profundidad tanto la estructura y diseño anterior como la comunicación entre los componentes. Por último, se espera que estas implementaciones pueden ser utilizadas como un entorno sobre el que realizar pruebas que puedan utilizarse como base para trabajos futuros de los investigadores de la Universidad de Zaragoza que están trabajando en este ámbito.

Individualmente cada parte de la guía anterior será revisada conforme el ejercicio del trabajo requiera actualizar o definir nuevos requisitos del proyecto. Es importante recalcar que en cada fase del proyecto se utilizarán prototipos, de modo que el trabajo pueda estar en desarrollo constante obteniendo un desarrollo ágil del producto final.

La memoria sigue la siguiente estructura: en primer lugar se introducen los sistemas de recomendación, sus características y diseños clásicos para poder entender qué son los sistemas de recomendación basados en el contexto. Una vez introducido el tema, el siguiente capítulo sirve para conocer la propuesta original de arquitectura del artículo [1] y cómo es adaptada para la realización de este TFG. Los siguientes capítulos contienen la información asociada a la implementación de la aplicación móvil y el Gestor de Entorno. Finalmente se muestran las conclusiones obtenidas y el posible trabajo futuro.

Los anexos incluidos se utilizan de apoyo a la memoria complementando la información presentada. En cada anexo se muestra información relevante mencionada durante el transcurso de la memoria, que se considera de relevancia para comprender con mayor profundidad el trabajo realizado.

Capítulo 2
Estudios y casos en sistemas de recomendaciones

A continuación, se describe en mayor profundidad qué son los sistemas de recomendación y a qué necesidades responden, cuáles son sus objetivos a nivel técnico y operacional y cómo se pueden implementar.

2.1 Objetivos y propiedades de un sistema de recomendación

El objetivo principal de los sistemas de recomendación es acercar o dar a conocer a los usuarios un producto acorde a sus gustos o necesidades. Este principio no es tan obvio como puede parecer; a continuación, se describen los objetivos más concretos [16]:

- Importancia o precisión: el objetivo más evidente e importante del sistema es realizar recomendaciones relevantes para el usuario; sin embargo, no es el único objetivo del sistema.

- Novedad: los sistemas de recomendación sirven al usuario como forma de descubrimiento de artículos. Un sistema de recomendaciones poco repetitivo puede conseguir ampliar la diversidad de ventas de una empresa y permitir a los usuarios descubrir nuevos gustos.

- *Serendipity*: se define como el descubrimiento de algo positivo o útil de forma inesperada. Este punto resulta de cierta ambigüedad dada la relación entre el nivel en el que un ítem es inesperado y es útil para el usuario. El término a diferencia de la novedad remarca el objetivo de conseguir una recomendación sorprendente para el usuario [4].

- Incrementar la diversidad de las recomendaciones: los sistemas de recomendaciones típicamente sugieren una lista de los k mejores ítems. Existe el riesgo de que estos ítems sean demasiado similares entre sí, y que no agraden al usuario. Aumentando la diversidad de estos ítems aumentan las posibilidades de que al menos alguna recomendación encaje con el usuario en cada momento.

- Relevancia del usuario: dentro de un sistema se debe tener en cuenta si todas las opiniones de los usuarios van a tener el mismo peso y en caso de no ser así cómo deben calcularse los pesos de cada uno.

- Confianza: es relevante tanto la confianza del sistema en sus predicciones como la confianza que otorgan los usuarios al sistema. En la primera, el sistema debe ser capaz de ofrecer resultados fieles a la realidad conforme aumentan las valoraciones de los usuarios y por tanto aumenta la información que puede analizar. La segunda resulta relevante para el usuario cuando comprueba que el sistema ofrece recomendaciones razonables para él. Esto lleva a que el usuario típicamente sigue con más frecuencia las recomendaciones propuestas [7].

- Arranque en frío: una consecuencia directa de confiar en un modelo dependiente del *feedback* de los usuarios es que el sistema inicialmente no posee información suficiente para realizar predicciones precisas.

- Riesgo: en algunas ocasiones, realizar una recomendación puede suponer un riesgo potencial. Un ejemplo de esto es que en un *e-commerce* puede resultar contraproducente recomendar un producto cuyo stock pueda agotarse, pudiendo ser más recomendable para el sistema ofrecer un producto sin riesgo a tener una rotura de stock.

- Robustez: definida como la estabilidad del sistema frente a la presencia de información falsa, típicamente introducida con el fin de influir en las recomendaciones. El sistema debe ser capaz de reaccionar contra ataques cuyo fin sea alterar el resultado de las predicciones. Además, puede utilizarse un ataque equivalente con el fin de conocer información privada de los usuarios, vulnerando así su privacidad. En general el sistema debe desarrollar un sistema de defensa frente a inyecciones de información maliciosa.

- **Adaptabilidad**: un sistema real opera con una colección de ítems en constante cambio. Cuando un evento exterior inesperado ocurre, el sistema debería ser capaz de adaptar sus predicciones a las tendencias del momento. Esto supone que ciertas predicciones solo resulten útiles a los usuarios durante un breve periodo de tiempo, volviéndose obsoletas posteriormente.

- **Escalabilidad**: como en cualquier sistema de información la escalabilidad es un punto clave, el sistema debe poder ser capaz de procesar y adaptar su funcionamiento al incremento tanto de información como de peticiones por parte de los usuarios, mientras mantiene las propiedades anteriores.

Idealmente, el diseño de un sistema de recomendaciones debe contemplar todas estas propiedades, ya que cada una de ellas ofrece un comportamiento deseable en el sistema. Sin embargo, en ocasiones, debido a la amplitud de objetivos, no es posible alcanzar todos estos objetivos. Es importante establecer un objetivo en el diseño del sistema que permita conseguir un resultado que satisfaga los puntos más críticos enumerados.

2.2 Algoritmos y métodos en sistemas de recomendación clásicos

- **Filtrado basado en contenido**: los sistemas de recomendación trabajan con perfiles, que guardan información acerca del usuario y sus gustos. Durante el proceso de recomendación, el sistema compara los ítems valorados positivamente por el usuario tratando de recomendar otros similares. La dificultad de este método radica en cómo valorar la similaridad entre distintos ítems. Uno de los algoritmos utilizados en este método trata de representar tanto a usuarios como ítems como vectores en un espacio vectorial, de forma que cuanto menor es el ángulo formado entre dos vectores más similares son. Para ello cada ítem tiene asociados una serie de atributos con un valor que permiten generar el vector que lo representará. Cuanto mejor descrito esté cada ítem por sus atributos mayor será la precisión de las recomendaciones [5]. La siguiente figura ejemplifica este paradigma, en ella se muestran dos géneros entre los que se sitúan los vectores de dos películas y el vector asociado con las preferencias del usuario.

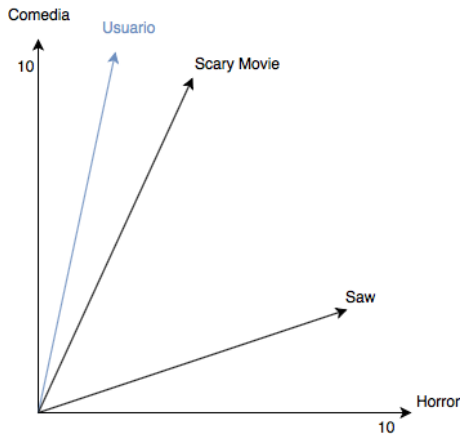


Figura 1. Vectores de similaridad en un plano. Adaptado de [5].

- **Filtros colaborativos:** basa sus predicciones en la opinión y comportamiento de otros usuarios en el sistema. Asume que las acciones de otros usuarios son una base razonable para realizar una recomendación. Es decir, si un grupo de usuarios están de acuerdo en la importancia de ciertos ítems, probablemente lo estarán también para otros distintos.

En general los sistemas de recomendación que utilizan filtros colaborativos generan primero un conjunto de recomendaciones para las preferencias del usuario, para posteriormente organizarlas según las valoraciones de los usuarios. En ocasiones esta organización es diferente para cada usuario, de forma que la clasificación generada tiene mayor validez para el usuario objetivo.

Existen dos métodos tradicionales de filtros colaborativos, filtros colaborativos usuario-usuario y filtros colaborativos ítem-ítem.

- Los filtros colaborativos usuario-usuario fueron los primeros en implementarse. Parten de la premisa de que usuarios con evaluaciones similares pasadas es probable que se comporten de forma similar en el futuro. Requiere componer una función de similaridad entre los usuarios para generar nuevas predicciones. Este método se considera efectivo, aunque poco eficiente y escalable.

- Los filtros colaborativo ítem-ítem tratan de solventar este problema de escalabilidad. En este caso se utiliza la similaridad entre los patrones de valoraciones de los ítems. El método trata de deducir la similaridad de los ítems mediante los patrones de preferencia de los usuarios como ocurre en los sistemas de recomendación basados en contenido. El algoritmo es altamente sensible cuando se disponen de pocas valoraciones de los ítems, puesto que las valoraciones tienen un peso relativo al número de estas.

En base a estos dos métodos surgen nuevas aproximaciones, entre ellas la reducción de dimensionalidad, que combinar tanto el dominio referente a usuarios como a ítems mientras trata de eliminar dimensiones redundantes. Para ello considera que es posible considerar tanto a ítems como a usuarios como conjuntos divisibles de similares características [6].

- **Aproximaciones híbridas:** para obtener mejores resultados algunos sistemas de recomendación utilizan aproximaciones que tratan de aprovechar las ventajas de ambos filtros comentados anteriormente [5].

2.3 Sistemas de recomendación basados en el contexto

Los sistemas de recomendación tradicionales ignoran el hecho de que los usuarios interaccionan con el sistema en un determinado contexto y las preferencias del usuario pueden ser diferentes dependiendo del contexto.

El término “contexto” hace referencia a la existencia de factores contextuales como el tiempo, la localización, etc. Para analizar el contexto de un usuario existen dos dimensiones a considerar: qué factores puede conocer el sistema del usuario y cómo estos cambian a lo largo del tiempo.

La evolución del contexto a lo largo del tiempo resulta un factor importante en el momento de realizar una recomendación. Cada factor tiene un tiempo de caducidad, que en ocasiones no es posible predecir, mientras que otros son fácilmente predecibles. En un mismo sistema se analiza, dentro del contexto, información asociada al usuario, como edad o género y el tiempo atmosférico, temperatura, precipitaciones, etc. Mientras que el primero es improbable que varíe en un tiempo razonablemente breve, en el segundo es posible que en el proceso desde que se realiza la medición hasta que se procesa haya cambiado drásticamente, y sin datos suficientes no es posible asegurar el contexto en el momento de realizar la predicción. Este problema divide el contexto en dimensiones estáticas y dinámicas. La monitorización del contexto debe tener en cuenta la variabilidad de cada dimensión para obtener un contexto realista manteniendo la eficiencia, especialmente importante en entornos móviles [8].

Una vez obtenida la información deseada asociada al contexto, es necesario identificar qué se va a realizar con este contenido. La información puede ser tan dispersa que es necesario acotarla; por ejemplo, si disponemos de una variable que indica la categoría de un ítem, el conjunto de categorías para poder ser analizadas es recomendable que se encuentre acotada a un cierto número de elementos,

lo mismo ocurre con valores numéricos como puede ser el porcentaje de humedad, por los que resulte más interesante realizar agrupaciones y disponer de un rango menor de valores [8].

Técnicamente, una vez el conjunto de datos está bien definido existen tres formas diferentes de construir un algoritmo de recomendación haciendo uso de información contextual (ver figura 2).

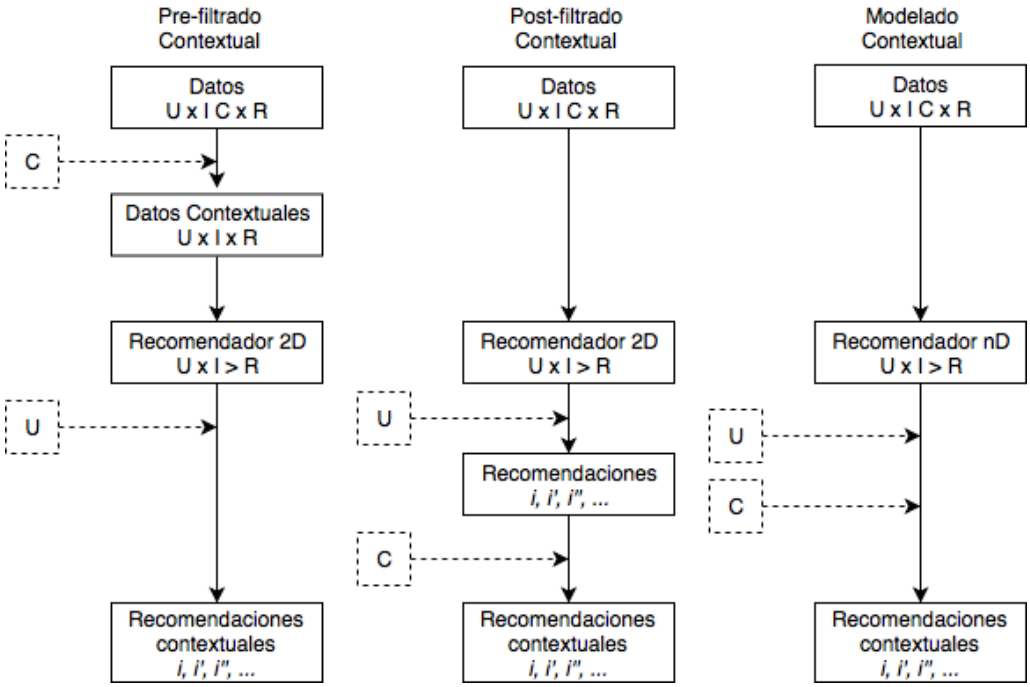


Figura 2. Paradigmas en sistemas de recomendación dependientes del contexto. Adaptado de [9].

- Pre-filtrado contextual: consiste en aplicar una fase de filtrado inicial capaz de suprimir, a través del contexto los pares usuario-ítem de menor relevancia para generar recomendaciones con información contextual de mayor similitud. En el caso de aplicar esta técnica es relevante comprobar si existen casos en los que reducir la información a la hora de entrenar el recomendador supone un peor rendimiento de este. Puede darse este caso cuando no hay suficiente información para un contexto determinado. Como se comenta anteriormente, el modelado del contexto resulta de gran relevancia a la hora de aplicar estas técnicas.

- Post-filtrado contextual: esta metodología, al igual que los sistemas clásicos, produce inicialmente recomendaciones ignorando el contexto. Una vez generadas las recomendaciones hace uso del contexto filtrando recomendaciones que son irrelevantes en determinados contextos o ajustando la valoración de estas recomendaciones. Esta técnica al igual que el pre-filtrado contextual tienen la ventaja de poder utilizar técnicas clásicas de sistemas de recomendación.

- Modelado contextual: este enfoque integra directamente la información contextual en la función de recomendación. El modelado sustituye el par usuario-ítem por la tupla usuario-ítem-contexto. El recomendador sustituye las dos dimensiones clásicas por un modelo n-dimensional.

Capítulo 3
Diseño y planteamiento de la arquitectura a utilizar

Esta sección muestra el diseño de la arquitectura del sistema a diseñar, base tomada de la propuesta teórica expuesta en el artículo de investifación *Push-Based Recommendations in Mobile Computing Using a Multi-Layer Contextual Approach* [1].

Se muestran también los requisitos que implica seguir esta aproximación y cómo desde una vista global la arquitectura expuesta resuelve y distribuye las tareas del sistema objetivo.

3.1 Recomendaciones proactivas en computación móvil utilizando una aproximación contextual multi-capa

El artículo [1] sirve como punto de partida de este proyecto. En él se presenta una aproximación efectiva de recomendación de ítems en entornos móviles que ejerza proactivamente, cuya mayor particularidad reside en el concepto de entorno y el tratamiento del impacto de eventos dinámicos y actores implicados en el proceso de recomendación.

La idea principal del modelo consiste en dividir la dinámica del usuario en unidades llamadas contextos. El contexto delimita el alcance o propósito de las recomendaciones con el objetivo de incrementar la precisión de las predicciones.

El entorno es otro concepto básico en el modelo: un entorno encapsula el proceso de recomendación y su información contextual y la comunicación entre las diferentes entidades. Un entorno se define como un área común en la que coexisten usuarios que comparten constantes físicas o virtuales y un conjunto de acciones realizables que lo delimitan. Estos conceptos implican que un mismo usuario puede resultar activo en entornos superpuestos.

Por ejemplo, un usuario se encuentra en el recinto de un museo pero, a su vez, se encuentra cercano a un restaurante a la hora de la comida. En este caso, un mismo usuario se encuentra activo en dos entornos diferentes al mismo tiempo, el modelo propuesto debe ser capaz de gestionar esta situación y evitar los conflictos.

El modelo propone dos tipos de agentes en el sistema: usuarios y Gestores de Entorno o *Environment Managers* (*EMs*). Los usuarios conforman las partes interesadas en las recomendaciones, mientras que un *EM* es una entidad encargada de proveer información en entornos específicos.

Generar recomendaciones para los usuarios de forma proactiva es el objetivo final del sistema, de esta forma el usuario no necesita pedir explícitamente recomendaciones acerca de un tipo de actividad que esté interesado, sino que el sistema automáticamente toma la iniciativa y creará recomendaciones adecuadas liberando al usuario de tareas manuales. Para conseguir este propósito se sigue un proceso de cinco fases ejecutadas tanto por el *EM* como por la aplicación móvil:

1. El dispositivo del usuario de forma transparente decide cuándo activar el proceso de recomendación.
2. Fase de pre-filtrado en el *EM* para descartar las actividades que se encuentren fuera del alcance el contexto y/o intereses del usuario.
3. Algoritmo de recomendación que realizará la predicción sobre las actividades.
4. Fase de post-filtrado ejecutada en el dispositivo móvil que resuelva conflictos entre diferentes actividades. Durante esta fase se utilizan preferencias privadas que ayudan a mejorar la

experiencia de usuario, el objetivo es compartir la mínima de información posible con los gestores para que el sistema funcione adecuadamente.

5. Presentación de resultados al usuario.

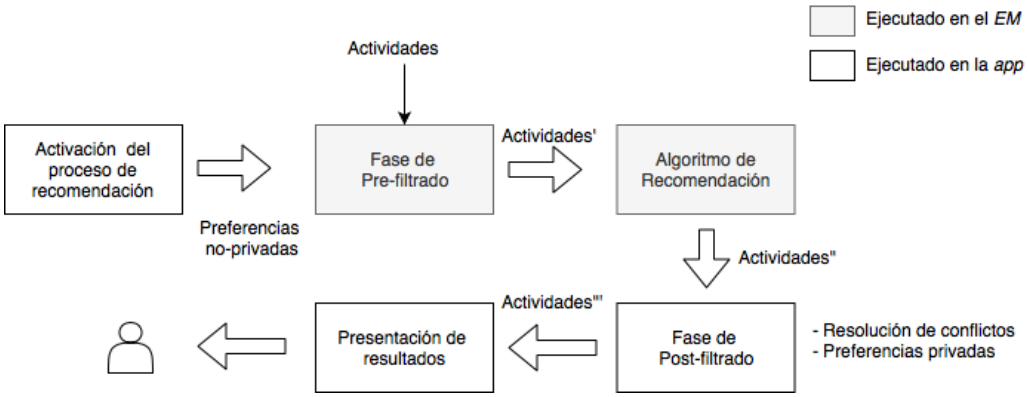


Figura 3. Flujo del sistema de recomendación. Adaptado de [1].

En las siguientes secciones se relata cómo este marco ha servido de referencia para implementar un sistema completo adaptado a la tecnología actual.

3.2 Visión global del sistema y resolución de conflictos

Una vez detallados los conceptos básicos de proyecto y el objeto de éste, podemos plantear la arquitectura que servirá de referencia para el trabajo.

El sistema, compuesto por dos agentes principales, se puede entender a partir del modelo cliente-servidor. Dos tipos agentes, la aplicación móvil que ejerce como cliente y el *EM*, que toma el papel de servidor, tendrá la misión de proveer información a cada cliente que realice peticiones.

La particularidad del sistema reside en el aspecto en el que pueden coexistir diferentes *EMs* independientes entre sí, originando un sistema global descentralizado. El cliente no dependerá de un servidor central para funcionar, en lugar de eso cada vez que éste se encuentre en un contexto compatible con algún *EM* podrá recibir nueva información.

En el anexo I. Documentación API CARS se propone una interfaz compatible con la aplicación a implementar para cada gestor de entorno que desee participar en el sistema. Dicha interfaz establece una serie de operaciones que permiten comunicar ambas partes adecuadamente, además de definir tareas necesarias para la gestión del servicio entre ellas, se definen operaciones como una precarga de información recopilada por otros gestores, con el fin de minimizar problemáticas como el arranque en frío del recomendador.

En la figura 4 podemos ver un diagrama que representa la arquitectura de alto nivel propuesta para el sistema. En él se refleja cómo el sistema trabaja con múltiples nodos iguales de cada tipo formando un sistema descentralizado. Este diseño completamente distribuido supone una serie riesgos en la implementación.

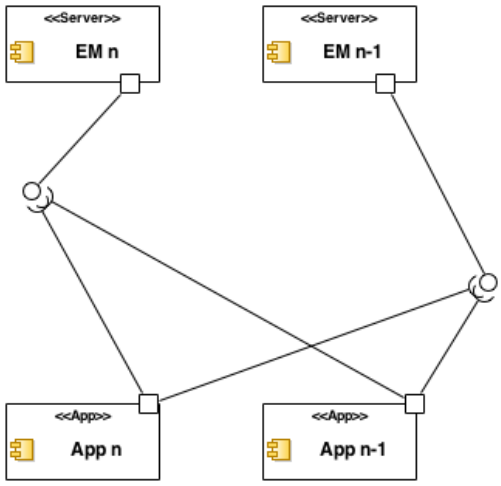


Figura 4. Diagrama de alto nivel del sistema

El sistema diseñado idealmente resulta totalmente descentralizado, de forma que cada componente forma parte de un conjunto de iguales. Resulta relevante considerar que, dentro del sistema, mientras que los servidores pueden considerarse como un componente estático, los diferentes usuarios utilizan teléfonos móviles, herramientas que con relativa brevedad son sustituidos por otros nuevos e incluso nuevos usuarios pueden utilizar teléfonos de antiguos usuarios, lo que dificulta la tarea de poder asociar una identidad a cada dispositivo del sistema. Esta premisa obliga a recurrir a un servicio externo con el cual sea posible asociar una identificación inequívoca a cada usuario; de esta forma dos usuarios diferentes no pueden presentarse a un mismo *EM* con una misma identidad. Asimismo, la arquitectura requiere una forma de conectar servidor y cliente. En el Capítulo 4 se detallan con mayor precisión este tipo de problemáticas y su implicación directa con la privacidad del usuario.

Esta arquitectura propuesta genera un desafío, la privacidad del usuario. La información del usuario que comparte con los *EMs* no debe mantener ningún tipo de carácter personal que pueda servir para identificar al usuario. Así, cada *EM* podrá almacenar el contexto asociado a la valoración que realiza un usuario sobre un ítem, pero en ningún momento tendrá acceso a la identidad del usuario, más allá de su identificador.

3.3 Integración entre aplicación y Gestor de Entorno

La arquitectura propuesta divide el sistema en dos tipos de entidades, la aplicación móvil y el gestor de entorno. Cada una cuenta con una serie de tareas individuales. Igualmente es necesario una forma de coordinar y conectar ambos para compartir información y que el sistema pueda completarse correctamente. A continuación, se muestra un diagrama de comunicación entre ambas partes, cada paso de la comunicación se encuentra enlazada respecto al flujo del sistema recomendación ilustrado la Figura 3.

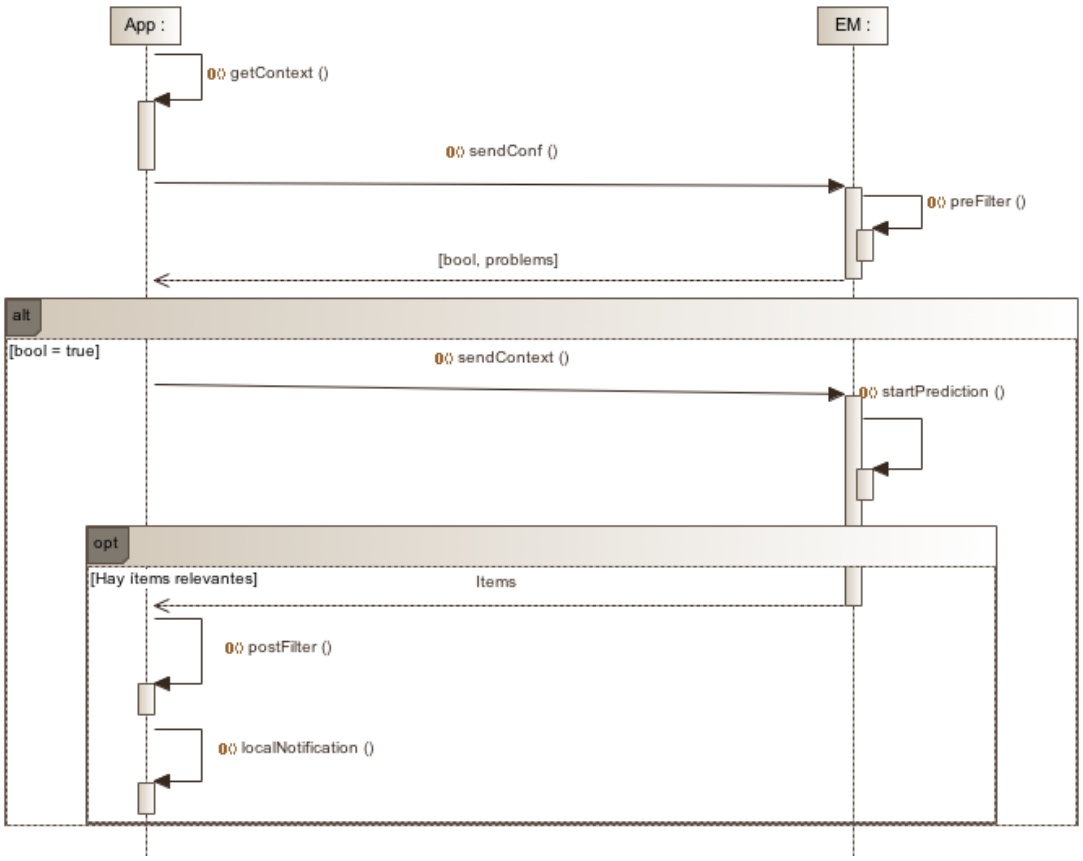


Figura 5. Diagrama de comunicación

En el diagrama encontramos el reparto de tareas descrito en el flujo de recomendación. Esta interacción realizada de forma periódica comienza con una toma de la información y eventos del entorno del usuario realizada transparentemente para el mismo.

Una vez se tiene información renovada se contacta con cada *EM* para informar acerca de ciertas propiedades tanto del perfil de usuario como de su privacidad. Con esta información el *EM* debe comunicar al usuario si su configuración pública le hace apto para recibir recomendaciones.

Si el usuario es apto el proceso continúa, la información del contexto recogida anteriormente se formatea adecuadamente y se envía al gestor, que iniciará el algoritmo de recomendación para proporcionar al usuario los ítems que pueden ser de su agrado.

Finalmente, la aplicación realizada otra etapa de filtrado con la información recibida que le permite con mayor conocimiento descartar actividades que no sean de interés, además de proceder a notificar al usuario de la llegada de nueva información y según la configuración privada del usuario ordenarlas para su posterior visualización.

El desarrollo de dos componentes individuales, en ocasiones se produce como proyectos separados que pueden o no conocerse. El foco de este proyecto es la implementación de la parte móvil; por tanto, para poder desarrollar un gestor de entornos compatible, se ha diseñado una propuesta de una interfaz de programación de aplicaciones o *API* que establezca un contrato entre las partes a seguir para garantizar las comunicaciones. Esta interfaz ha sido especificada utilizando el lenguaje *RESTful API Modeling Language (RAML)* [15], *RAML* facilita y estandariza la exposición de la *API* y ayuda a compartirla, diversas herramientas permiten generar vistas web a partir de la

definición en este lenguaje. En el Anexo 1. Además, se encuentran más detalles acerca de la interfaz, así como su definición.

La interfaz propuesta incluye tres métodos necesarios para la comunicación entre gestor y móvil: dos operaciones ya mostradas en el diagrama de comunicación en la Figura 5, que permiten recibir la configuración y datos del usuario y la información de su contexto, una tercera que interpreta cómo se debe enviar el *feedback* de un usuario, ya sea implícito o explícito.

Igualmente, la interfaz propone un modelo de autenticación y gestión del *EM* como servicio, que permita tramitar información tanto actividades como usuarios, de forma masiva o individualizada, con el objetivo de poder gestionar el contenido y la información de forma segura.

Capítulo 4
Implementación de la aplicación móvil

Esta sección muestra el proceso y la toma de decisiones realizada durante la implementación de la aplicación. Además, relata los problemas encontrados, las soluciones propuestas y las decisiones tomadas.

Este proceso implica gran parte del desarrollo del proyecto, siendo el mayor punto de énfasis a implementar dentro del sistema formulado. En este apartado se acentúan la funcionalidad de la aplicación de forma individual, incluyendo el prototipado y capturas de la propia interfaz gráfica.

4.1 Análisis de requisitos y planificación

El artículo [1] propone un *framework* genérico sobre el que desarrollar una arquitectura. Aunque no ahonda en los requisitos que necesita la aplicación móvil, podemos determinar ciertas obligaciones.

La captura del contexto del usuario es el requisito más importante y más complejo. La tecnología disponible en los teléfonos móviles permite conocer multitud de detalles tanto físicos como virtuales, i.e. posición geográfica, velocidad, dirección o eventos en la agenda, contactos e incluso a través de terceros, información personal, redes sociales, información meteorológica, etc. Toda esta información, a pesar de que tecnológicamente es recopilable, genera dos problemas. El primero es que si la información no es relevante incrementa el volumen de información con la que trabajar y la carga de trabajo, disminuyendo la eficiencia del sistema. El segundo implica la privacidad del usuario, este punto resulta crítico, por lo que el usuario debe decidir qué información va a compartirse, e incluso en el caso de la posición geográfica, de gran interés en el sistema, la comparte con un nivel de ruido dentro de un rango seleccionado.

El resto de los requisitos están relacionados con la interacción del usuario con las actividades, visualización, borrado, almacenamiento en favoritos, configuración de preferencias y de sesión.

Antes de comenzar a desarrollar la aplicación es necesario conocer las tendencias actuales en el mercado de los dispositivos móviles, en el primer cuatrimestre de 2018 sumaban entre iOS y Android en España en una tendencia ascendente el 99,7% de cuota de mercado [16]. Este hecho muestra la importancia de explorar las tecnologías no solo nativas, sino alternativas híbridas cuyo fin es unificar al máximo la implementación, minimizando costes de desarrollo. Este proyecto ha aprovechado la oportunidad de conocer de primera mano estas tecnologías para ampliar el conocimiento de sus beneficios y desventajas frente a soluciones nativas. React Native ha sido la tecnología escogida para esta tarea. Basada en tecnologías web, además de ofrecer diversas librerías permite incluir código nativo que solvente posibles deficiencias. El propósito de utilizar este enfoque es conocer si es posible obtener una aplicación 'nativa' funcional en ambas plataformas, utilizando un lenguaje ya conocido como JavaScript. Anteriormente, se exploró la posibilidad de implementar una aplicación embebida en el navegador concretamente utilizando PhoneGap, pero en una fase temprana del desarrollo se detectaron inconvenientes suficientes como para abandonarla. La falta de librerías y el rendimiento fueron algunos de los inconvenientes detectados.

El desarrollo de la aplicación se encuentra planificado en 4 fases: prototipado, análisis y recogida del contexto, interacción usuario-ítems y configuración e interacción *app-EM*. La primera ayuda a concretar detalles necesarios para la implementación de las consiguientes, pero además la recogida del contexto es analizada desde la perspectiva de la capacidad para obtener la información y su

complejidad computacional, la utilidad real de la información y la privacidad del usuario. En el Anexo IV.I se enumeran los requisitos funcionales y no funcionales recogidos para la aplicación.

4.2 Diseño de la aplicación, interfaz y experiencia de usuario.

La aplicación se ha diseñado conforme a los requisitos definidos en la sección anterior y con el objetivo de facilitar al usuario su interacción con los ítems propuestos por los *EMs*. Encontramos seis vistas diferentes:

- Inicio de sesión: tiene como función permitir a un usuario iniciar sesión para acceder a la aplicación.

- *Home* o inicio: es la pantalla inicial, desde la que se puede consultar una lista con todos los ítems propuestos. Las vistas de favoritos o históricos son equivalentes a ésta, la diferencia reside en qué información muestran. En 'favoritos' se visualizan actividades guardadas explícitamente por el usuario. 'Históricos' permite visualizar temporalmente ítems cuyo contexto ya no coincide con el contexto actual. Estas pantallas permiten, sin acceder a la actividad marcar (o desmarcar) cada actividad para guardarla o eliminarla del dispositivo.

- Perfil y Ajustes: ambas pantallas se utilizan para personalizar la aplicación, establecer filtros utilizados para las fases de pre y post filtrado y presentación.

- Actividad: vista que ofrece toda la información asociada a una actividad concreta. En ella se puede valorar la misma o acceder a una nueva pantalla con un **mapa** en el que se visualiza su posición geográfica y la posición del usuario.

En la Figura 6 se muestra el resultado del mapa de navegación que se ha utilizado como prototipo para el diseño de la interfaz gráfica de la aplicación móvil. En el mapa aparecen las pantallas descritas anteriormente y el proceso de interacción que debe realizar el usuario para navegar entre cada pantalla.

Adicionalmente, en el Anexo II puede consultarse en detalle, el resultado final del diseño de cada vista y ver una captura de estas, así como una descripción de las interacciones que pueden realizarse desde cada vista y el efecto que producen en el sistema.

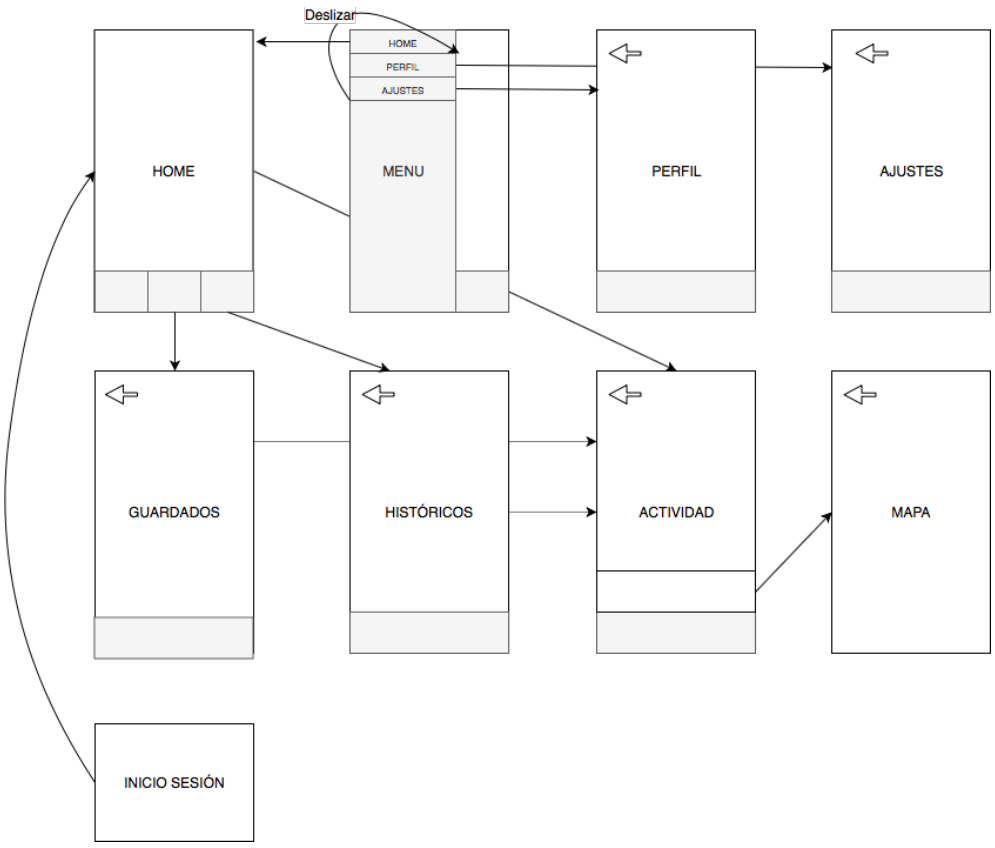


Figura 6. Mapa de navegación de la aplicación

El diseño de la interfaz de la aplicación trata de seguir las guías de estilo propuestas por Apple para su plataforma iOS [19], en ella proponen seis principios que debe cumplir una aplicación para tener un buen nivel de usabilidad: integridad estética, que es la coherencia entre la acción a realizar y su apariencia; consistencia, que consiste en hacer el diseño familiar al usuario; manipulación directa, que supone manipular los elementos de la aplicación facilita el entendimiento de las acciones, ver instantáneamente el resultado de una acción refuerza la confianza del usuario; f*eedback*, significa que el sistema debe emitir una respuesta para cada acción del usuario; metáforas, que trata de crear semejanzas entre posibles acciones dentro de la aplicación y la vida real; control de usuario, que establece la necesidad entre crear un diseño que permita al usuario tener el control de la aplicación siempre y cuando no habilite la posibilidad de tomar acciones peligrosas.

La finalidad de seguir esta guía es adaptar y entender la implicación de principios básicos en el diseño para conseguir una aplicación que proponga una interacción familiar para el usuario, pudiendo adaptar la estética en función de las necesidades (diferencias entre plataformas o versiones) sin que esto pueda afectar al comportamiento de la aplicación.

A continuación, se muestra una captura del *home*, cuyo diseño trata de trasladar las recomendaciones anteriores al prototipo de la aplicación. El objetivo es, además de cumplir con los requisitos funcionales, ofrecer otros no funcionalidades, orientados a ofrecer una aplicación con la que el usuario pueda sentirse cómodo, que sea sencilla, pero a la vez ofrezca una experiencia agradable. En el Anexo IV se incluyen estos requisitos no funcionales.

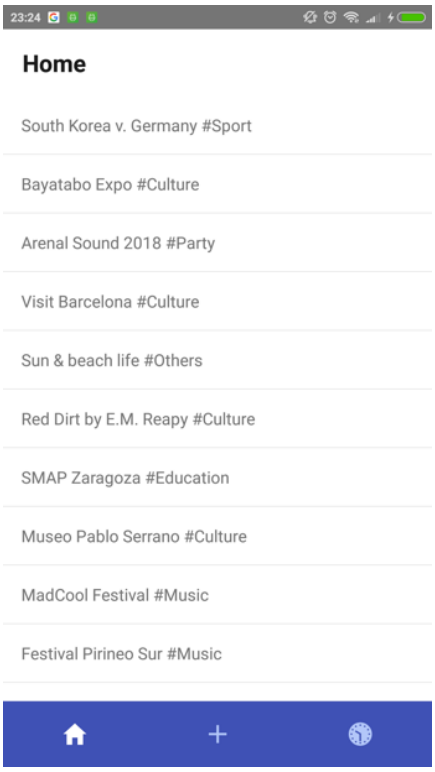


Figura 7. Captura de la pantalla inicio de la aplicación móvil

Todas las pantallas que componen la aplicación han sido diseñadas siguiendo el prototipo propuesto. En el Anexo II pueden consultarse cada una de ellas, junto con el detalle de las opciones que ofrece cada una.

4.3 Implementación de la aplicación

React se encuentra construido para generar interfaces de usuario a partir de componentes. Esta pauta ha sido seguida durante la implementación de la aplicación, se ha buscado reutilizar los componentes que en diferentes pantallas realizan la misma tarea, de forma que cada componente se construye sobre otros componentes generando finalmente cada pantalla de la aplicación. Así se consigue un diseño sencillo en el cada componente encapsula su lógica y vista, y al mismo tiempo se comparte un modelo de datos único. Este modelo, inicialmente diseñado junto al prototipo conforme a los requisitos del sistema se ha ido adaptando para poder satisfacer la lógica de cada componente.

Adicionalmente a [17], se ha recurrido al uso de ciertas librerías externas para complementar el desarrollo, la base de datos utilizada, el sistema de autenticación, la generación de notificaciones y la obtención del contexto se ayudan de las librerías que se mencionan a continuación:

- Realm Database [20]: esta librería ofrece una base de datos NoSQL ideada para simplificar el desarrollo de aplicaciones. Partiendo de la definición de esquemas de diferentes objetos, permite interactuar con los datos mediante objetos, agiliza el desarrollo en proyectos cuyas consultas no tengan un nivel de complejidad alto, pero que recurran frecuentemente a consultas simples y actualizaciones de datos. El enfoque de esta base de datos permite realizar un diseño orientado a interactuar directamente con la lógica de cada componente a través de objetos.

La base de datos se encuentra diseñada a partir de cinco objetos: usuarios, actividades, ajustes y un objeto genérico utilizado para componentes del contexto. La simplicidad de este paradigma permite asociar de forma sencilla el prototipo y los requisitos en la capa de persistencia de la aplicación.

- Facebook SDK [21]: se utiliza únicamente su servicio de autenticación de la red social, con el fin de identificar de forma anónima e inequívoca a los usuarios de la aplicación. Además, ofrece servicios para obtener información sobre los usuarios que podría considerarse para ampliar el contexto del usuario, aunque puede comprometer la privacidad del usuario.

En el caso de querer ampliar la información que puede obtener por defecto esta librería es necesario solicitar a Facebook permisos razonando para qué va a ser utilizada dicha información.

- React Notifications CE [22]: es una librería que ofrece una interfaz para trabajar con notificaciones locales y remotas, es compatible con Firebase Cloud Messaging [25], tecnología de Google utilizada en el envío de notificaciones a dispositivos iOS y Android y también permite utilizar Google Cloud Messaging, obsoleta desde el 10 de abril de 2018.

- Geolocation API [23] y **React Native Calendar** [24]: han sido utilizadas para capturar el contexto del usuario. La primera informa de las coordenadas geográficas en las que se encuentra el dispositivo y la velocidad a la que se está moviendo. La segunda permite acceder a la información de los calendarios del dispositivo y sus eventos detallados.

Gracias tanto a las herramientas ofrecidas de serie, como a las librerías de terceros ya citadas, se ha podido desarrollar la aplicación con cierta agilidad, pudiendo centrar los esfuerzos en decisiones de lógica y diseño. Sin embargo, una parte de la aplicación incluye código nativo, necesaria para programar tareas mientras la aplicación se encuentra en segundo plano. Dicha parte ha sido realizada utilizando Headless JS [27], una librería para Android que permite ejecutar tareas en JavaScript mientras la aplicación está en segundo plano. Para ejecutar estas tareas es necesario utilizar un servicio nativo que inicie el proceso; por ello es necesario implementar código nativo que, defina el servicio que inicia el código JavaScript y, la forma de iniciar este servicio, como puede ser de forma periódica (el caso de este TFG) o como respuesta a un evento del sistema.

4.4 Interacción con los Gestores de Entornos

La arquitectura del sistema ha sido diseñada teniendo en mente que una implementación única de esta aplicación pueda interaccionar a través de una interfaz con cualquier gestor de entorno que la implemente. Este principio supone liberalizar la implementación de los gestores de los que podemos conocer o no su comportamiento. Por ello es vital asumir que la aplicación debe asegurar una buena experiencia de usuario independientemente de las intenciones del gestor y las estrategias que puede seguir para promocionar su contenido por delante de otros gestores.

Una consecuencia adicional del diseño distribuido y la falta de una entidad centralizada es la falta de un repositorio que permita a los gestores conocer a los usuarios y viceversa. Sin embargo, puesto que la implementación de la aplicación sí está estandarizada y gracias al soporte que ofrece las plataformas para recibir actualizaciones en las aplicaciones, se puede implementar este repositorio dentro de los documentos de la aplicación de modo que cuando un nuevo *EM* entre a formar parte del sistema se emita una actualización para que los usuarios comiencen a interactuar con él. En este documento se incluye información acerca del *EM* que permita a falta de confirmación por el propio

gestor, entender a la aplicación que se encuentra en un entorno compatible, compartiendo así la mínima información necesaria con los gestores, que a su vez evitarán sobrecargas en su sistema.

La implementación del gestor de entorno, explicada en el Capítulo 5, incluye métodos para entender la interacción con el usuario, si un usuario borra un ítem para impedir su visualización se entiende que no desea que aparezca más, esta acción produce retroalimentación que se en principio se podría enviar al gestor y este debería evitar enviar este ítem otra vez al usuario, sin embargo, no existe la certeza de que esto ocurra. Para evitar que el usuario vuelva a recibir este contenido e incluso genere una notificación en su terminal, afectando directamente a su experiencia la aplicación ha sido diseñada de modo que las actividades no se eliminan completamente, sino que se marcan como borradas en la base de datos: de este modo se pueden diferenciar las actividades que el usuario no desea ver y además durante la fase de post-filtrado pueden descartarse. Caso parecido ocurre si el gestor envía en repetidas ocasiones un ítem; esto puede darse por malas prácticas del *EM* o simplemente porque su información asociada haya sido actualizada. En ambos casos, la fase de post-filtrado es clave, la aplicación debe detectar y actualizar la información que se actualiza, incluso llegando a notificar al usuario si el cambio se considera lo suficientemente relevante.

La aplicación debe estar preparada para cualquier comunicación con el *EM* siempre que siga el formato especificado en la documentación. La comunicación sabiendo que es el punto más crítico del sistema se ha diseñado de la forma más sencilla posible con la intención de evitar errores y comportamientos inesperados en ambas partes. Esto reduce el abanico de escenarios que deben estudiarse y el comportamiento que puede o debe realizar cada componente.

4.5 Resolución de problemas y posibles conflictos

Cualquier proyecto con cierta complejidad tiene una gran posibilidad de sufrir problemas o conflictos durante su desarrollo. Este caso no ha sido una excepción y durante el desarrollo han surgido tanto dificultades derivadas del planteamiento como derivadas de limitaciones técnicas.

En el plano del diseño, como se ha repetido a lo largo del proyecto las principales dificultades vienen dadas por la descentralización del sistema. Utilizar un servidor a través del cual cuando un usuario entre a la aplicación le permita registrarse e identificarse soluciona el problema de conseguir que cada usuario se identifique de forma inequívoca para cada *EM.* Para solucionar esta restricción se ha hecho uso de un servicio de autenticación de terceros, en este caso se ha implementado un sistema de *login* con Facebook, aunque se podría haber utilizado cualquier otro que asegure que nunca se ofrecerá el mismo *token* de autenticación a distintos usuarios dentro de la aplicación; la base de datos se encuentra adaptada de forma que sea capaz de trabajar con *tokens* de distintos proveedores. La elección de Facebook como servicio de autenticación viene motivada debido a pesar de que en este proyecto no se ha tomado información personal de la red social, es una forma sencilla de enriquecer el contexto del usuario a costa de su privacidad. Como posibles avances futuros siguiendo la filosofía de permitir al usuario qué información privada es compartida podría hacerse uso de dichos datos.

Un problema añadido, y que comparte similitudes con el problema anterior, se debe a la imposibilidad de asociar un dispositivo físico a un único usuario. Un usuario *U* comienza utilizando la aplicación con un dispositivo *D,* por diversos temas el dispositivo *D* pasa a manos de un nuevo usuario *U'*, y posteriormente de vuelta a su usuario original *U*. Para lidiar con estas situaciones la base de datos de la aplicación está adaptada para mantener información de cada uno de los usuarios que han iniciado sesión en el mismo dispositivo, sin perder información. Desafortunadamente, si el usuario *U* pasa a utilizar un nuevo dispositivo *D'* perdería toda la información almacenada, dado que

no hay un sistema de copia de seguridad en la nube. A pesar de ello, para los *EMs* no se trata de un nuevo usuario y mantendrán su precisión en las nuevas predicciones.

La privacidad del usuario ha sido el principal conflicto durante el diseño y la implementación de la aplicación; decidir qué información del usuario capturar y con quién debe compartirse es un aspecto crítico. Para solventar esta problemática se ha optado por permitir al usuario decidir qué será compartido, con qué frecuencia e incluso el ruido que se añaden a ciertas medidas como la posición geográfica. Para decidir si es necesario informar con mayor detalle del contexto o si se informa actualmente de información irrelevante son necesarias pruebas del sistema que clarifiquen qué dirección puede tomar la aplicación en posibles futuros avances.

La planificación de tareas en segundo plano ha sido el mayor problema técnico encontrado durante la implementación de la aplicación. React Native, por defecto recomienda utilizar el framework Expo [26] cuyo objetivo es facilitar el desarrollo de la aplicación, hecho corroborado durante este desarrollo. Desafortunadamente, aunque aseguran que están trabajando en ello, en su versión 31.0.0 última hasta la fecha, no dispone de herramientas para ejecutar tareas en segundo plano, perdiendo la proactividad del sistema. La capacidad de ejecutar estas tareas sin que el usuario tenga que abrir la aplicación permite: en primer lugar, monitorizar el contexto actual del usuario, y comenzar la comunicación con los gestores de entorno. Así, la aplicación toma la iniciativa en el proceso sin depender de la interacción del usuario.

Este punto hizo que el desarrollo de la aplicación no solo tuviera que migrar para eliminar el framework y sus dependencias, sino que React Native solo dispone de una herramienta [27] para ejecutar tareas en JavaScript en segundo plano exclusiva para Android, perdiendo la compatibilidad entre ambas plataformas. Para solucionar esta problemática se estudiaron librerías de terceros, pero las limitaciones de estas hicieron que finalmente se decidiera progresar exclusivamente en Android.

Las limitaciones y problemas encontrados durante el desarrollo han podido ser subsanados buscando un enfoque alternativo al planificado inicialmente. No obstante, tanto por parte de diseño como por aspectos técnicos las soluciones no son ideales. Respecto al diseño del sistema es necesario hacer un balance entre la privacidad del usuario y qué información y con qué servicios se va a compartir, dar la posibilidad de elegir al usuario que información se comparte o añadir ruido a las medidas aumentan dicha privacidad en detrimento de la calidad de la información compartida.

Técnicamente también se han encontrado dificultades, React Native no es una tecnología lo suficiente madura como para soportar todas las necesidades del proyecto y que éste pueda seguir siendo compatible para Android y iOS, perdiendo una de las principales ventajas de utilizar un *framework* de desarrollo híbrido.

En el Capítulo 6 se explican conclusiones acerca de estos problemas encontrados. Además, se incluye una sección en la que se comentan las limitaciones de la aplicación móvil y, cómo en trabajo futuro se pueden incluir nuevos requisitos que mejoren la experiencia del usuario en el uso del sistema.

Capítulo 5
Implementación y bases de un Gestor de Entorno (EM)

En este capítulo se va a explicar con mayor detalle el rol que ejerce el *EM* en el sistema mostrado en la Figura 4, las necesidades de este más allá de su propia interacción con la aplicación. Se propone una implementación sobre un servidor web, debido a la compatibilidad que ofrece con la tecnología móvil utilizada.

5.1 Definición de tareas y servicios a proveer

La definición de Gestor de Entorno dentro del sistema hace referencia a un agente especial asociado a un entorno concreto, cuya tarea es consiste en realizar recomendaciones a los usuarios que se encuentren dentro de este entorno. El segundo objetivo es ofrecer la posibilidad, no sólo de distribuir actividades entre los distintos usuarios sino la de almacenar y gestionar dichas actividades a través de una interfaz. Por tanto, dentro del *EM* deben desarrollarse de forma paralela dos roles diferenciados, como recomendador y comunicador de actividades a los diferentes usuarios y como gestor de su propio contenido. A continuación, se exponen los diferentes requisitos según el rol de *EM* y sus implicaciones en el desarrollo de este.

5.1.1 Gestor de Entorno como sistema de recomendación

Como se ha comentado en secciones anteriores los sistemas de recomendación tienen el objetivo de generar recomendaciones significativas sobre una serie de ítems o productos para un grupo de usuarios a los que les puedan resultar interesantes. Concretamente este sistema aprovecha la capacidad de conocer el contexto del usuario en el momento de la recomendación. Uniendo la definición clásica del sistema de recomendación junto al conocimiento del entorno es posible definir la base de la funcionalidad que debe ofrecer:

El sistema debe ser capaz de gestionar qué usuarios son de su interés (a través de los datos de su entorno) y monitorizar los datos provistos por cada uno de los usuarios. Como sistema de recomendaciones debe ser capaz de generar un modelo a través del cual predecir si un ítem puede ser de interés para un usuario en un contexto concreto. Para lograr proactividad en el sistema desde la aplicación móvil se genera una notificación, por lo que será importante evitar saturar al dispositivo del usuario buscando suscitar el mayor interés posible al recibir una nueva notificación. Por tanto, reflejando esto con métricas comúnmente utilizadas en sistema de recomendaciones, en este caso se buscará premiar la precisión del sistema frente al *recall.*

$$Precisión = \frac{Recomendados \cap Relevantes}{Recomendados} \qquad \text{Recall} = \frac{\text{Recomendados} \cap \text{Relevantes}}{\text{Relevantes}}$$

5.1.2 Gestor de Entorno como gestor de contenido.

En el sistema diseñado el sistema de recomendación es una importante parte del *EM*, pero, éste desempeña funciones alternativas que permitirán adquirir nueva información. Este componente del sistema ofrecerá también la posibilidad de gestionar los ítems que puede recomendar. De esta forma, al igual que contacta con la aplicación para adquirir nuevas predicciones otro tipo de usuario con rol de administrador accederá, esta vez para realizar operaciones básicas sobre los ítems que se almacenarán en su base de datos. Crear, actualizar, consultar y eliminar son las operaciones básicas

necesarias para proveer de actividades al sistema de recomendaciones. En el Anexo I se encuentran detallados estos métodos junto a los necesarios para la comunicación *EM-app*.

5.2 Propuesta de implementación de un *EM*

El papel del *EM* en el sistema puede entenderse, simplificando el concepto de Gestor de Entorno como un servidor que proporciona de información a los usuarios en forma de recomendaciones en ciertas ocasiones. Sobre esta afirmación se ha realizado el diseño de un componente que permite la entrada, gestión y almacenamiento de actividades, así como de usuarios y sus interacciones con estas actividades.

A partir de la interfaz propuesta en el Anexo I y los requisitos recogidos en el Anexo IV. se ha diseñado un esquema entidad-relación y posteriormente la base de datos SQL que utiliza el *EM*. En la Figura 8 se muestra el diseño de la base de datos que permite explicar el modelo de datos implementado.

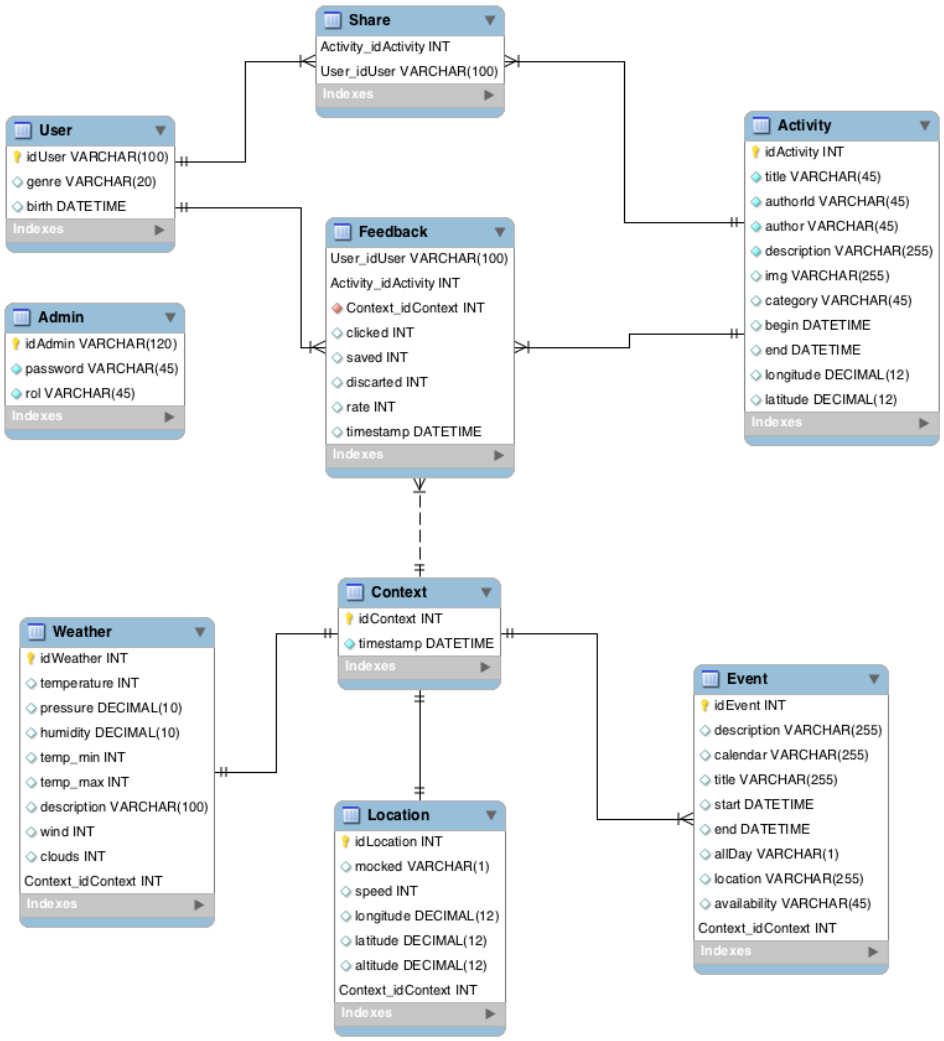


Figura 8. Esquema relacional implementado en el EM

En el esquema se distinguen los tres tipos de entidad principales, usuarios, actividades y contexto. En este caso se han definido con los mismos campos especificados en la *API* propuesta, se ha tomado esta decisión para no restringir la información a analizar, aunque es posible que, tras realizar pruebas sobre el sistema eliminar información de menor relevancia ayude a mejorar la eficiencia.

La relación entre usuario y actividad *share* indica que una actividad ya se ha compartido con un usuario, de esta forma es posible enviar actividades repetidamente a la aplicación. Aunque la aplicación en su fase de post-filtrado, ya posee un mecanismo para evitar esta problemática si se detecta antes de ser enviado se consigue reducir la cantidad de datos consumidos por el dispositivo móvil en la comunicación.

La segunda característica relevante a comentar en el modelo propuesto se encuentra en la relación entre usuario, actividad y contexto, *Feedback.* De cara a la posterior implementación del sistema de recomendación y puesto que la aplicación lo permite, se añadió la posibilidad de considerar la retroalimentación implícita cuando un usuario realiza un clic, guarda o marca para borrado una actividad. La decisión relevante dentro de esta propuesta recae en qué contexto será almacenado cuando un usuario valora una actividad, para ello se sigue la siguiente regla: si el *feedback* es explícito se almacenará el contexto del usuario en el momento de proporcionar la valoración; si es implícito ya sea guardar o borrar y no tiene valoración se almacenará el contexto del momento en el que se produce la acción del usuario; finalmente en caso de que el usuario solo haya realizado clic en la actividad se guardará el contexto del momento del último clic.

El tipo de entidad *Admin* se utiliza con el fin de definir usuarios que tengan permiso para acceder, crear, modificar o eliminar las actividades y otras tareas de mantenimiento dentro del gestor.

Sobre las decisiones tomadas en este diseño y los requisitos definidos en la interfaz se ha construido el servidor del *EM*. Para la implementación de un ejemplo de *EM* se ha utilizado el lenguaje Java utilizando el *framework* Spring Boot [28], el motivo de esta elección reside en la combinación de la facilidad que ofrece para construir la *REST API* propuesta junto a la compatibilidad con librerías de aprendizaje automático como Apache Mahout [29] o Apache Spark MLlib [30] que puedan utilizarse para implementar el algoritmo de recomendación.

La arquitectura del servidor se encuentra dividida en tres capas de forma similar a un esquema modelo-vista-controlador, en el cual la parte relativa a la vista se sustituye por una capa que define la interfaz a través de la que se comunica con los distintos usuarios. La implementación del servidor, más allá del recomendador se limita ofrecer operaciones *CRUD*, con ciertas limitaciones sobre actividades y usuarios que pueden consultarse en el Anexo I.

5.3 Implementación de un sistema sencillo de recomendaciones de ejemplo

En la implementación de un Gestor de Entorno reside uno de los puntos más críticos en el sistema completo: el diseño del sistema de recomendaciones. Las posibilidades que aporta una descripción tan detallada del entorno del usuario pueden ser aprovechadas de diferentes formas, encontrar el enfoque más preciso no es sencillo y es un tema de interés en investigaciones actuales.

Este tema es de especial complejidad y sobrepasa los límites de este TFG, el cual se encuentra centrado especialmente en la parte móvil del sistema. Por ello, en este trabajo se implementa un sistema de recomendaciones dependiente del contexto simplificado, en el Capítulo 2, se comentan tres formas de implementar el sistema de recomendaciones; de ellas se ha tomado la alternativa que propone realizar un pre-filtrado en base al contexto para construir un recomendador con datos de mayor calidad.

Para implementar este sistema de recomendaciones se ha optado por explorar la solución Apache Spark [31]. El objetivo de este *framework* es dar una solución rápida y de propósito general a la computación en grupos, posible solución en la implementación de un gestor cuyo concepto implica dar soporte a usuarios de forma concurrente con los altos costes de computación que puede conllevar la generación de recomendaciones. Spark incluye la librería de *machine learning* Spark MLlib [30] requerida en la implementación del algoritmo de recomendación.

El algoritmo implementado comienza en el filtrado de información. En primer paso se recuperan todas las valoraciones disponibles, generando una estructura con los datos usuario-ítem-contexto-valoración por cada valoración. Es posible además utilizar el *feedback* implícito que el sistema es capaz de recoger para sustituir o complementar la valoración explícita.

La forma de filtrar el contexto elegida ha sido utilizar vectores de similaridad como se utilizan en sistemas de recomendaciones con filtros basados en contenido. Esto exige transformar cada valor del contexto en un valor nominal, preferiblemente en el que dos valores similares entre sí sean transformados en valores nominales próximos reflejando esa similaridad. La función que transforme cada característica del contexto en un valor afecta a la construcción del vector que será utilizado en el filtro. En este proyecto se ha optado por una solución más trivial que no considera la proximidad entre valores; reflejar que dos valores de un dato no estructurado, como puede ser una descripción, son similares entre sí, no es una tarea sencilla y depende de cada conjunto de valores. Una vez obtenemos un vector con la información del contexto elegimos las valoraciones en las cuales el coseno del vector de su contexto y el contexto actual del usuario es menor. Este filtro es necesario que se ajuste a la cantidad de datos de los que disponemos, especialmente en el contexto actual, puesto que es necesario un mínimo de información para poder generar un recomendador eficaz.

Una vez disponemos de las valoraciones realizadas en un contexto similar al actual podemos generar un recomendador con la información usuario-ítem-valoración. El recomendador será capaz de: dado un usuario, predecir la valoración que éste puede dar a cada ítem. En este caso se utiliza un filtro colaborativo, una de las técnicas más extendidas en sistemas de recomendaciones. El objetivo de esta aproximación es completar las entradas vacías en la matriz de asociación usuario-ítem encontrado variables latentes que puedan utilizarse para predecir esta información. El algoritmo utilizado es el propuesto en [30], *Alternating Least Squares* (ALS).

Las propuestas de factorización de matrices asocian usuarios e ítems a un espacio de factores latentes de dimensión *f*. Cada ítem *i* está asociado a un vector $q_i \in \mathbb{R}^f$ y cada usuario *u* a un vector $p_u \in \mathbb{R}^f$. El resultado del producto de ambos captura la interacción entre usuario e ítem, aproxima la valoración de forma estimada $\hat{r}_{ui} = q_i^T p_u$. La mayor dificultad computacional reside en estimar dichos vectores, ya que se necesita utilizar un algoritmo de fuerza bruta poco eficiente para el sistema. ALS propone una aproximación más eficiente:

Entendemos que q_i y p_u son desconocidos, pero si conociésemos una de las incógnitas la optimización del problema se convierte en cuadrática, siendo resoluble computacionalmente. La técnica ALS consiste en iterar ajustando ambos vectores hasta poder calcular p_u de forma individual, para después calcular q_i, proceso que termina convergiendo en un mínimo local [10].

El beneficio de este algoritmo se encuentra en la capacidad de calcular de forma paralela cada q_i de los demás vectores de ítems y cada p_u de los demás vectores de usuario, aprovechando la capacidad de computación paralela que podemos conseguir con Spark.

La computación de cada modelo con el que realizar las recomendaciones al usuario es una tarea costosa que ralentiza el proceso de recomendación, en este proyecto cada vez que llega una petición

el proceso se realiza desde cero, sin embargo, sabiendo que el sistema puede recibir peticiones de forma continua es necesario reflexionar sobre la eficiencia de éste, incluso a costa de reducir su eficacia.

Definir un grupo acotado de contextos en los que ya se haya calculado el modelo permite asignar a cada petición un modelo generado previamente con un conjunto de valoraciones en un contexto similar, de forma que el proceso de recomendación evita los cálculos de mayor coste computacional a cambio de perder cierta precisión. Estas variantes deben definirse y someterse a pruebas en un entorno real en el que comprobar las ventajas e inconvenientes de cada solución su implicación en el conjunto del sistema.

Una vez disponemos de un modelo que permita realizar recomendaciones, la biblioteca utilizada permite generar la *n*-mejores recomendaciones para un usuario, el criterio elegido para realizar recomendaciones ha sido, establecer un valor fijo para *n* una vez tenemos las mejores recomendaciones para el usuario en su contexto actual serán enviadas las *m* valoraciones que no se hayan enviado antes y cuya estimación sea superior a tres en una escala de valoraciones entre cero y cinco. El objetivo es enviar ítems cuya valoración estimada sea 'buena' para el usuario intentado no saturar de información su dispositivo con el deseo de maximizar la atención del usuario en estas sugerencias.

5.4 Pruebas sobre el sistema de recomendaciones

El punto crítico del *EM* reside en el sistema de recomendaciones. Una de las mayores dificultades encontradas en el desarrollo del proyecto reside en la capacidad de realizar pruebas sobre el sistema completo. Para ello se necesita un conjunto de datos, que permitan conocer si la interacción entre aplicación y gestor es coherente y si el algoritmo del sistema de recomendaciones ofrece unas predicciones con acierto suficiente.

Los *datasets* o conjuntos de datos pueden ser utilizados con el fin de comprobar la fiabilidad de un sistema de recomendaciones. El artículo [11] se trata la problemática de obtener y analizar los diferentes conjuntos de datos que pueden ser utilizados para evaluar un sistema de recomendación basado en el contexto. En él se puede comprobar cómo la disponibilidad de esta información es reducida y generalmente se encuentra descompensada, de forma que los que conjuntos expuestos que contienen amplia información sobre el contexto tienen habitualmente un alto porcentaje de valores desconocidos. Se resalta LDOS-CoMoDa [12] un *dataset* con recomendaciones de películas y su contexto. Este conjunto finalmente ha sido el elegido y adaptado para validar el algoritmo construido para el recomendador. La elección de éste conjunto se debe a que ofrece hasta doce atributos contextuales y sólo el 4% de ellos son desconocidos.

Una vez elegido el conjunto de datos, es necesario adaptar la estructura de la información que contiene a la estructura del sistema. Durante este proceso, aunque se trata de perder el mínimo de información, resulta complejo asociar ambas estructuras puesto que, mientras la fuente de datos está orientada a películas, el sistema diseñado es de ámbito generalista y en la mayoría de los casos la información esta descrita como un texto. Para realizar estos procesos de transformación de datos se ha utilizado el software Knime [32] que permite generar a través del fichero de datos original uno adaptado al sistema que pueda cargarse con las operaciones de carga masiva definidas en la interfaz del *EM*.

El algoritmo de recomendaciones asocia a cada entrada del par usuario ítem una estimación de la valoración. Para comprobar la efectividad del recomendador se ha medido el error de esta estimación en dos pruebas diferentes, en la primera se ha suprimido el concepto de contexto aplicando

el algoritmo de recomendación sobre todos los datos, en la segunda prueba se ha añadido la fase de pre-filtrado con el objetivo de conocer cómo afecta el contexto disponible en la precisión de las estimaciones.

Las predicciones en las pruebas se han realizado tras entrenar aleatoriamente con aproximadamente el 80% de los datos del conjunto, mientras el 20% restante se ha utilizado para realizar las pruebas de validación del algoritmo. La configuración del modelo es la recomendada por defecto por los creadores de la librería, el único parámetro especificado viene en el número de iteraciones que debe realizar el algoritmo hasta converger (según la documentación, debería converger entre 10 y 20).

La medida utilizada para estimar el error del algoritmo es la raíz del error cuadrático medio (*MSE*), mostrada a continuación:

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(f_i - y_i)^2$$

Donde N es el número de estimaciones, f_i el valor de la estimación y y_i el valor real de para la valoración i. Las valoraciones se encuentran cuantificadas entre cero y cinco, por tanto, el error determina la diferencia de valoración entre la realidad y la estimación.

Tomando 10 iteraciones en la primera prueba, obviando el contexto de las recomendaciones la raíz del error cuadrático medio obtiene un valor de 2,87, reduciéndose hasta 1,49 si elevamos el número de iteraciones a 20.

En la segunda prueba antes de generar el modelo se realiza la fase de pre-filtrado con el objetivo de eliminar los contextos más diferentes. En esta prueba no se han apreciado diferencias al variar el número de iteraciones, a diferencia de la prueba anterior, posiblemente debido a la gran reducción que sufre el conjunto de datos tras el filtrado. Como resultado se ha obtenido un error medio de 1,53.

Reflexionando sobre estos resultados se pueden obtener diversas conclusiones, en primer lugar, aunque el error es de aproximadamente punto y medio se debe a que el algoritmo no produce estimaciones enteras, es decir siempre va a existir cierto error, aunque la estimación aplicando un redondeo pueda considerarse acertada. También existe una tendencia en los usuarios a variar su criterio a lo largo del tiempo, un usuario inicialmente asocia una puntuación a cierta exigencia, pero esta no siempre es igual; al tratarse de un criterio subjetivo es el concepto de contexto el que trata de aplicar un balance entre los diversos factores que pueden afectar al criterio del usuario y su valoración explícita. Este sencillo estudio preliminar no muestra apenas diferencia entre la eficacia aplicando o no el contexto. Esta prueba puede demostrar que los factores seleccionados no permiten segmentar con acierto las opiniones. La complejidad de estas pruebas es elevada debido a la escasez de *datasets* estudiados que además poseen una estructura diferenciada de la capturada a través de la aplicación desarrollada.

Capítulo 6
Conclusiones y trabajo futuro

En este capítulo, se extraen conclusiones tanto personales como técnicas acerca del proyecto. Asimismo, se proponen ciertas guías para orientar trabajo futuro que pueda aprovechar la investigación y el trabajo realizado.

6.1 Evaluación personal

Este trabajo es para mí la primera experiencia en un proyecto de investigación, experiencia muy diferente a otros trabajos realizados durante el grado. Este proyecto me ha permitido no sólo conocer y profundizar una temática específica, sino también poner en práctica conocimientos adquiridos de diferentes áreas. El proyecto parte del análisis de un artículo [1] a partir del cual se trata de conseguir una implementación de un modelo teórico propuesto, la investigación realizada aunque parte de este artículo, abarca desde qué es un sistema de recomendación y cómo funcionaban en sus inicios hasta obtener un conocimiento más específico, desde los sistemas de recomendación basados en el contexto y qué algoritmos utilizan a desarrollo de aplicaciones móviles, limitaciones y ventajas de las diferentes tecnologías, etc. Todo ello vinculando esta información al proceso de desarrollo del proyecto, ahondando en diferentes investigaciones y soluciones a cada problema encontrado para adaptar este conocimiento al propósito del proyecto.

La complejidad de este proyecto, además de servir como experiencia en el campo de la investigación también es una experiencia en el diseño y gestión de proyectos. La planificación en el proyecto es clave, por ello desde el comienzo se ha planificado el desarrollo de este siguiendo un diagrama de Gantt; sin embargo, la dificultad al asignar un horario continuado al proyecto ha hecho que las fechas asignadas a cada tarea debían ser revisadas conforme avanzaba el desarrollo del proyecto. No solo la falta de tiempo personal, sino las dificultades encontradas durante el mismo desarrollo, complican la tarea de organización del proyecto. Esta dificultad ha sido la más compleja de tratar durante el trabajo, una mejor planificación hubiera facilitado y agilizado las fechas de progreso.

En la Figura 9, se muestra en forma de gráfico el conjunto de tareas y el tiempo en horas dedicado a cada una de ellas.

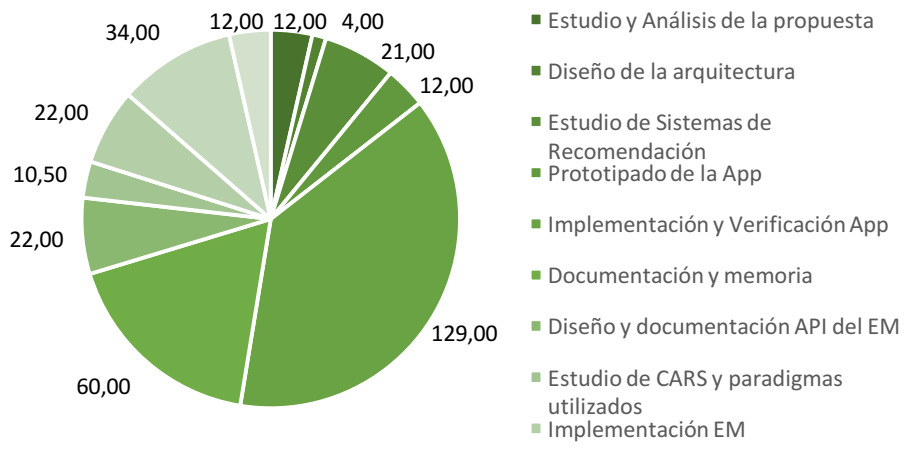


Figura 9. Gráfico de distribución de horas por tarea

El siguiente diagrama, es un diagrama de Gantt que expone el tiempo dedicado a cada tarea, pero en este caso no con relación a las horas dedicadas sino a las fechas que abarca desde el inicio al fin de la tarea. Como se ha comentado previamente debido a mi relación de disponibilidad, el desarrollo del trabajo ha sufrido diversas interrupciones.

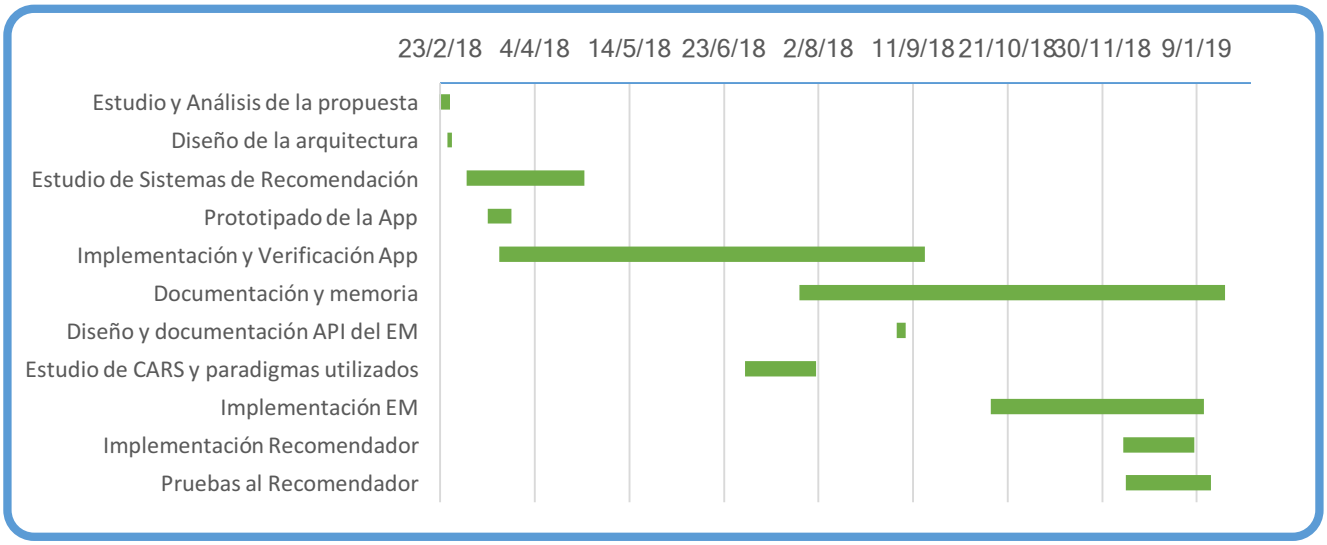


Figura 10. Diagrama de Gantt del proyecto

El gráfico muestra la duración en días de cada tarea. Este gráfico complementa al gráfico de distribución de horas por tarea. Se puede observar que el tiempo de desarrollo del proyecto a pesar de extenderse excesivamente a lo largo del tiempo las tareas de mayor extensión lo son también en horas dedicadas.

Este proyecto, pese a las dificultades encontradas durante su progreso, me ha permitido adquirir nuevas experiencias acerca de la planificación y desarrollo de proyectos. Asimismo, he podido elegir una temática en la que adquirir un conocimiento específico que me ha permitido, en colaboración con Sergio Ilarri, contribuir al artículo [13] complementando la experiencia con este trabajo en el ámbito de la investigación; en el Anexo V se puede consultar dicho artículo.

6.2 Conclusiones acerca del proyecto

A lo largo de este proyecto se ha diseñado un sistema de recomendaciones basadas en el contexto. El diseño de este sistema, basado en la propuesta "Push-Based Recommendations in Mobile Computing Using a Multi-Layer Context" [1], trata de obtener un modelo que poder implementar manteniendo los mismos objetivos que en la propuesta. Del diseño de este sistema se obtienen diferentes conclusiones. Suprimir una entidad central en el sistema aumenta la complejidad de éste, ya que con la ayuda de una entidad centralizada se facilitaría el registro de nuevos gestores de entorno, se podría evitar o complementar el uso de servicios de terceros para iniciar sesión, e incluso podría existir un sistema *cloud* a través del cual se almacenen los datos de cada usuario, de forma que si éste cambia de terminal toda su información pueda ser transmitida a este nuevo dispositivo e incluso, podría ofrecer servicios con esta información a los diferentes gestores. Pero especialmente podría utilizarse como conector entre aplicación y gestor. En este sistema propuesto es la aplicación la que conoce la dirección de los gestores e inicia la comunicación, la entidad central podría ejercer este rol, su función sería, a través del contexto del usuario indicar a la aplicación qué gestores son compatibles. Esta opción permite añadir nuevos *EMs* al sistema sin tener que actualizar la aplicación y establecer un mecanismo de mayor complejidad que conecte al usuario con el gestor sin afectar al rendimiento y tamaño de la aplicación. Sin embargo, el diseño descentralizado también aporta ventajas como la reducción del coste de mantenimiento del sistema o el aumento de la tolerancia a errores.

Centrándonos en el prototipo de la aplicación móvil, en el ámbito técnico la tecnología elegida, React Native, muestra *pros y contras* respecto a una implementación nativa, sin entrar en términos de rendimiento puesto que la aplicación actualmente no realiza tareas de alta complejidad computacional. El objetivo de utilizar esta tecnología era conocer de primera mano el mundo de las aplicaciones híbridas y conocer si es posible con un único desarrollo ofrecer una aplicación compatible con iOS y Android. La conclusión al respecto muestra que cuando la aplicación es relativamente compleja mediante librerías externas puede obtenerse un producto que satisfaga la mayoría de necesidades; sin embargo, este proyecto tiene como requisito la proactividad de las recomendaciones y para conseguirlo es necesario que la aplicación sea capaz de programar procesos en fondo que se comuniquen con el gestor, para realizar esta tarea la tecnología no está suficiente desarrollada y solo ofrece una solución para Android, para ejecutar tareas programadas con JavaScript. En iOS obliga a re-implementar la tarea en código nativo, perdiendo así la compatibilidad. Existen librerías como react-native-background-task [33] que afirma programar tareas en segundo plano para ambas plataformas con la limitación de ejecutarla con un intervalo mínimo de 15 minutos, limitación que obligó a descartarla. Sin embargo, el resultado final obtenido es satisfactorio. La reutilización de componentes y la facilidad para implementar interfaces ayuda a agilizar el desarrollo notablemente. Además, es una tecnología cambiante que recibe actualizaciones semanales, por lo que poco a poco ofrece nuevas posibilidades, reduciendo la fragmentación que puede existir en el desarrollo.

El conjunto del proyecto permite indagar más en el concepto presentado en [1], partiendo de un esquema teórico se genera un desarrollo genérico que puede servir como base para futuras investigaciones. El objetivo era conocer qué aspectos propuestos son realizables, en términos de

efectividad y eficiencia, pero sin dejar de lado la privacidad del usuario y qué limitaciones obligan a proponer una alternativa. Por ejemplo, la metodología utilizada para conocer si un usuario puede estar activo en un entorno se encuentra redefinida por la dificultad que tiene el gestor de conocer a nuevos usuarios en un entorno en el que la información de estos no se encuentra centralizada en ningún nodo conocido.

En el aspecto asociado al paradigma de recomendaciones se puede concluir que la falta de fuentes de datos supone un obstáculo y dificultad para establecer conclusiones sólidas; sin embargo, esta propuesta ofrece la posibilidad de capturar el contexto de forma automática, objetiva y transparente al usuario, que puede suponer un gran avance a la hora de conocer la situación del usuario en el momento de su valoración.

En [11] se listan una serie de conjuntos de datos de con valoraciones contextuales de diferentes fuentes entre las que se incluyen DePaulMovie y LDOS-CoMoDa sobre películas, InCarMusic sobre música para pasajeros de coche, STS y TripAdvisor para turismo y Frappé de ámbito genérico. Estos conjuntos, aunque ofrecen información acerca del contexto tienen importantes limitaciones, a excepción de LDOS-CoMoDa los *datasets* ofrecen o poca información sobre el contexto o tienen un gran número de valores desconocidos en él. En la Figura 11 se informa con detalle de esta situación.

Dataset	# context attributes	# users	# items	# ratings	% unknown context values
DePaulMovie	3	97	79	5043	28.71%
InCarMusic	7	42	139	4012	90.54%
Frappé	6	957	4082	96203	23.09%
STS	14	325	249	2534	89.37%
TripAdvisor	1	1202, 2371	1890, 2269	14175, 28350	0%, 0%
LDOS-CoMoDa	12	121	1232	2296	4.00%

Figura 11. Tabla de estadísticas de los conjuntos de datos. Adaptado de [11].

La aplicación desarrollada puede llegar a obtener la localización del usuario (y si esta es simulada por algún tipo de aplicación que permita falsear las coordenadas que sitúan al usuario), velocidad, atributos atmosféricos (entre ellos temperatura, nubosidad, humedad, visibilidad o velocidad del viento) y atributos acerca de eventos en el calendario como el calendario al que pertenecen (en Android por ejemplo, se incluye por defecto en España un calendario con los días festivos), el título del evento, localización, su inicio y final o si el usuario estará o no ocupado. En total suman 24 atributos que salvo decisión del usuario pueden ser recogidos y enviados a los gestores junto a las valoraciones de los ítems. Disponer de esta herramienta alivia la tarea de recogida de información y futuros avances en el sistema pueden disponer de datos más precisos con los que desarrollar un recomendador más eficaz. La aplicación móvil desarrollada es el inicio de la puesta en práctica de la arquitectura presentada en [1], el resultado de este desarrollo ha sido obtener una forma recopilar de automáticamente una descripción detallada del contexto que permita continuar la investigación acerca de los sistemas de recomendación dependientes del contexto con datos fieles a la realidad.

6.3 Avances futuros

Se considera que el desarrollo realizado resulta de utilidad para continuar con la investigación en sistemas de recomendación que se está realizando en el Departamento de Informática e Ingeniería de Sistemas de la Universidad de Zaragoza. En este sentido, está previsto que el equipo investigador con el que se ha colaborado en este TFG continúe refinando y ampliando el prototipo desarrollando nuevas funcionalidades.

El prototipo ofrece una base que adaptar a nuevos proyectos de investigación. La capacidad que tiene para capturar el contexto puede resultar de utilidad para solventar la dificultad que poseen algunos sistemas a la hora de capturar el contexto del usuario. Además ofrece la posibilidad de distribuir ítems a diferentes usuarios con el fin de entender la implicación del contexto del usuario a la hora aceptar o rechazar las propuestas que recibe.

En el ámbito funcional de la aplicación el objetivo era satisfacer una serie de requisitos básicos para que el usuario pueda recibir e interaccionar con diferentes actividades. Estas interacciones han sido diseñadas pensando en el gestor, de forma que reciba el *feedback* que el usuario produce con sus acciones, pero es interesante pensar, desde el punto de vista del usuario, un sistema de valoración con comentarios o un desglose de las valoraciones (no sólo la media de estas) como utilizan páginas como TripAdvisor [34] que ofrece al usuario la capacidad de conocer mejor las experiencias de otros usuarios y qué esperar de dicha actividad. Asimismo, el usuario puede que quiera conocer más acerca de cada gestor de entorno, puede darse el caso que la temática concreta de un gestor no resulte interesante para él, por tanto, sería interesante ofrecer la posibilidad no solo de conocer cierta información acerca de este sino de poder no compartir información con él. El prototipo desarrollado tiene como objetivo ser una prueba el concepto, pero de cara a una solución comercial podría ser interesante ampliar sus capacidades con estos requisitos descritos con el objetivo de mejorar la experiencia del usuario en el sistema.

Todos los avances futuros que se proponen deben seguir manteniendo una de las metas del proyecto, mantener la privacidad del usuario. Además, cualquier futuro cambio debería realizarse con el objetivo no solo de mantener esta privacidad, sino cumplir la normativa europea de protección de datos.

Paralelamente a este proyecto se están realizando estudios de temática similar en este ámbito. El rango de ítems de los sistemas de recomendaciones a veces puede resultar demasiado amplio y una estructura genérica puede ser excesivamente compleja o simplemente no ser efectiva para cualquier situación y en ocasiones es necesario centrarse en un objetivo para el que proponer una solución más específica.

PASEO 1.0 (Profile generation And content Suggestion on E-Tourism) [14] es un proyecto actual, orientado a producir una aplicación que recomiende actividades a turistas. El sistema que propone es un recomendador proactivo basado en el contexto. Investiga la posibilidad del uso de tecnologías semánticas, que permitan representar el conocimiento disponible para generar recomendaciones. Este proyecto se está desarrollando desde la Universidad de Zaragoza y aunque su desarrollo es complementario a este estudio comparte una base y objetivo común, comprender y estudiar las capacidades y posibilidades de los sistemas de recomendaciones proactivos.

Referencias

[1] Ramón Hermoso, Sergio Ilarri, Raquel Trillo, María del Carmen Rodríguez-Hernández, "Push-Based Recommendations in Mobile Computing Using a Multi-Layer Contextual Approach", 13th International Conference on Advances in Mobile Computing & Multimedia (MoMM 2015), Brussels (Belgium), ACM Press, ISBN 978-1-4503-3493-8, pp. 149-158, December 2015.

[2] Prem Melville y Vikas Sindhwani, "Recommender Systems", en: Sammut C., Webb G.I. (eds) Encyclopedia of Machine Learning and Data Mining. Springer, Boston, MA.

[3] Francesco Ricci, "Knowledge-Based Recommender Systems", eCommerce and Tourism Research Laboratory, Automated Reasoning Systems Division ITC-irst, Trento, Italy https://www.ics.uci.edu/~welling/teaching/CS77Bwinter12/presentations/course_Ricci/15-KnowledgeBased.pdf, último acceso: 25 de julio 2018.

[4] Suboojitha Sridharan, "Introducing Serendipity In Recommender Systems Through Collaborative Methods", Open Access Master's Theses, paper 453. University of Rhode Island, 2014.

[5] Daniar Asanov, "Algorithms and Methods in Recommender Systems", Berlin Institute of Technology, 2011.

[6] Michael D. Ekstrand, John T. Riedl y Joseph A. Konstan , "Collaborative Filtering Recommender Systems", Foundations and Trends in Human–Computer Interaction, Vol. 4, No. 2 (2010) 81–173.

[7] Guy Shani y Asela Gunawardana, "Evaluating Recommendation Systems", en: Ricci F., Rokach L., Shapira B., Kantor P. (eds) Recommender Systems Handbook. Springer, Boston, MA, 2011.

[8] Gediminas Adomavicius, Bamshad Mobasher, Francesco Ricci y Alex Tuzhilin, "Context-Aware Recommender Systems", Association for the Advancement of Articial Intelligence, AI Magazine Vol 32 No. 3: Fall 2011.

[9] Yong Zheng, "Tutorial: Context In Recommender System", Proceedings of the 31st ACM SIGAPP Symposium on Applied Computing (ACM SAC 2016), Pisa, Italy, April 2016.

[10] Yehuda Koren, Yahoo Research, Robert Bell y Chris Volinsky, AT&T Labs-Research, "Matrix Factorization Techniques for Recommender Systems", IEEE Computer Society, pp. 42-49, Agosto 2009.

[11] Sergio Ilarri, Raquel Trillo-Lado, Ramón Hermoso, "Datasets for Context-Aware Recommender Systems: Current Context and Possible Directions", First Workshop on Context in Analytics (CiA 2018), in conjunction with the 34th International Conference on Data Engineering (ICDE 2018), Paris (France), IEEE Computer Society, Electronic ISBN 978-1-5386-6306-6, Print on Demand (PoD) ISBN 978-1-5386-6307-3, ISSN 2473-3490, pp. 25-28, Abril 2018.

[12] Andrej Ošir, Ante Odić, Matevž Kunaver, Marko Tkalčič, F. Jurij Tasič, “Database for contextual personalization”, Elektrotehniški vestnik. [English print ed.], 2011, vol. 78, no. 5, pp. 270-274.

[13] Sergio Ilarri y Manuel Herrero, "Towards the Implementation of a Push-Based Recommendation Architecture", 13th International Workshop on Semantic and Social Media Adaptation and Personalization (SMAP 2018), Zaragoza (Spain), IEEE, Electronic ISBN 978-1-5386-8225-8, USB ISBN 978-1-5386-8224-1, Print on Demand (PoD) ISBN 978-1-5386-8226-5, pp. 87-92, September 2018.

[14] Ramón Hermoso Traba, “Inteligencia Artificial y turismo: una oportunidad para las AA.PP.”, Universidad de Zaragoza, http://www.itainnova.es/wp-content/uploads/2018/11/CIVITIA2018_MesaRetos-Ram%C3%B3n-Hermoso.pdf último acceso: 21/01/2019.

[15] RAML Org., “RAML Version 1.0: RESTful API Modeling Language”, https://raml.org, último acceso: 28 enero 2019.

[16] Kantar Inc. “Ventas de smartphones: las marcas chinas toman Europa”, https://es.kantar.com/tech/móvil/2018/mayo-2018-cuota-de-mercado-de-smartphones-en-españa-primer-trimestre-2018/, último acceso: 14 diciembre 2018.

[17] Facebook Inc. “React Native. A framework for building native apps using React”, https://facebook.github.io/react-native/, último acceso: 28 enero 2019.

[18] Adobe Systems Inc., “Adobe PhoneGap,” https://phonegap.com/, último acceso: 26 mayo 2018.

[19] Apple Inc., “Human Interface Guidelines”, https://developer.apple.com/design/human-interface-guidelines/ios/overview/themes/, último acceso: 16 diciembre 2018

[20] Realm Inc. “Realm Database”, https://realm.io/products/realm-database, último acceso: 28 enero 2019.

[21] Facebook Inc., “Facebook for developers”, https://developers.facebook.com, último acceso: 29 enero 2019.

[22] Carlos Alcazar, “React Native Push Notifications Community Edition”, https://github.com/calcazar/react-native-push-notification-CE, último acceso: 29 enero 2019.

[23] Facebook Inc., “Geolocation”, https://facebook.github.io/react-native/docs/geolocation, último acceso: 29 enero 2019.

[24] Wix Inc., “React Native Calendar Components”, https://github.com/wix/react-native-calendars, último acceso: 29 enero 2019.

[25] Google Inc., “Firebase Cloud Messaging”, https://firebase.google.com/docs/cloud-messaging/?hl=es-419, último acceso 29 enero 2019.

[26] 650 Industries, Inc., “Expo”, https://expo.io, último acceso: 29 enero 2019.

[27] Facebook Inc., “Headless JS”, https://facebook.github.io/react-native/docs/headless-js-android, último acceso: 29 enero 2019.

[28] Pivotal Software Inc., “Spring Boot”, http://spring.io/projects/spring-boot, último acceso 29 enero 2019.

[29] The Apache Software Foundation, “Mahout”, https://mahout.apache.org, último acceso 29 enero 2019.

[30] The Apache Software Foundation, “Apache Spark MLlib”, https://spark.apache.org/mllib/, último acceso 27 enero 2019.

[31] The Apache Software Foundation, “Apache Spark”, https://spark.apache.org, último acceso 27 enero 2019.

[32] Knime GmbH, “KNIME – Open for Innovation”, https://www.knime.com, último acceso 29 enero 2019.

[33] James Isaac, “react-native-background-task”, https://github.com/jamesisaac/react-native-background-task, último acceso: 29 enero 2019.

[34] TripAdvisor Inc., “TripAdvisor España”, https://www.tripadvisor.es, último acceso: 29 enero 2019.

Anexos

Anexo I

Documentación del API del Gestor de Entornos

La propuesta presentada en este proyecto incluye una arquitectura en la que dos componentes principales interactúan entre sí. La necesidad de interacción entre componentes cuya implementación puede ser independiente exige un contrato que ambos cumplan para llegar al entendimiento. Este contrato se establece a través de una RESTful API que, haciendo uso del estándar HTTP define una serie de operaciones necesarias para acceder al Gestor de Entorno.

La definición de la documentación está descrita en el lenguaje RESTful API Modeling Language (RAML). Este lenguaje está diseñado específicamente para documentar interfaces, incluye la definición de tipos, operaciones o seguridad, herramientas necesarias para este documento. Además, utilizando el *plugin* raml2html[1] se pueden generar páginas web con un formato más legible e interactivo para comprender la documentación como se aprecia en la figura I.1.

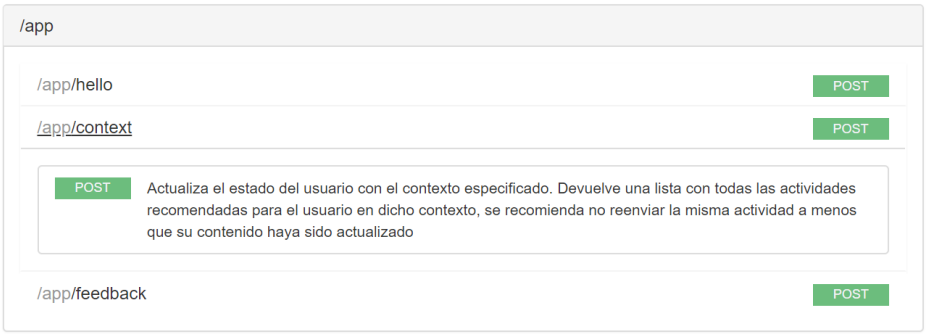


Figura I.1. Vista web generada automáticamente con raml2html

Las operaciones definidas en el documento se dividen en *session, activity, users, schema* y *app*:

- *Session* incluye operaciones de inicio y fin de sesión para obtener credenciales que permitan acceder a ciertas operaciones potencialmente peligrosas si las usara un usuario malintencionado.
- *Activity* y *Users* engloban operaciones de gestión de actividades y usuarios respectivamente.
- *Schema* ofrece métodos para importar y exportar información del gestor con la finalidad de gestionar los datos directamente de la base de datos, puede ayudar a tratar con análisis de información o evitar arranque en frío.
- *App* estas operaciones son las únicas obligatorias para comunicar aplicación y EM. Definen los métodos necesarios para cumplir el protocolo de comunicación establecido en el proyecto.

I.I. CARS versión 1.0.1

```
#%RAML 1.0
---
title: CARS
baseUri: http://localhost:8080/{version}
version: 1.0.1
```

[1] Raml2html: https://github.com/raml2html/raml2html

```yaml
documentation:
  - title: Resumen
    content: |
        CARS API presenta la interfaz mínima recomendada que debe implementar un
        Environment Manager para trabajar adecuadamente con la aplicación móvil.
        Además, considera diferentes métodos que deberían utilizarse para
gestionar
        y conocer el contenido del servidor para ofrecer una experiencia adecuada
        a los usuarios del mismo
mediaType: [ application/json ]
securitySchemes:
    passthrough:
        description: Autorización necesaria para el acceso a ciertas operaciones
de la API
        type: Pass Through
        describedBy:
            headers:
                api_key:
                    description: Utilizada para enviar un token de acceso válido
                    type: string
            responses:
                401:
                    description: Token inválido o expirado. Es necesario re-
autenticar el usuario
                403:
                    description: Petición denegada
types:
    DeviceToken:
        type: string
        description: Identificador utilizado por Google Firebase Cloud Messaging
para enviar notificaciones exclusivamente al terminal indicado
    UserId:
        type: string
        description: Identificador de usuario
    ActivityId:
        type: integer
        description: Identificador de actividad
    Img:
        type: string
        description: URL de la imagen
    Begin:
        type: date-only
        description: Fecha de inicio
    End:
        type: date-only
        description: Fecha de finalización
    Genre:
        type: string
```

```
        description: Género
    Birth:
        type: date-only
        description: Fecha de nacimiento
    Title:
        type: string
        description: Título
    Description:
        type: string
        description: Descripción
    AuthorId:
        type: string
        description: Identificador del EM emisor
    Author:
        type: string
        description: Descripción del EM emisor
    Category:
        type: string
        description: Categoría de la actividad
    Mocked:
        type: boolean
        description: Localización simulada
    Temperature:
        type: integer
        description: Temperatura en grados Kelvin
    Temp_min:
        type: integer
        description: Temperatura mínima en grados Kelvin
    Temp_max:
        type: integer
        description: Temperatura máxima en grados Kelvin
    Wind:
        type: integer
        description: Velocidad del viento en m/s
    Clouds:
        type: integer
        description: Porcentaje de cielo cubierto de nubes
    AllDay:
        type: integer
        description: ¿El evento abarca todo el día?
    Address:
        type: string
        description: Localización (dirección) del evento
    Availability:
        type: string
        description: Disponibilidad
```

```
   Calendar:
        type: string
        description: Nombre del calendario que incluye el evento
    Longitude:
        type: number
        format: double
        description: Longitud
    Latitude:
        type: number
        format: double
        description: Latitud
    Altitude:
        type: number
        format: double
        description: Altitud
    Speed:
        type: number
        format: double
        description: Velocidad en m/s
    Pressure:
        type: number
        format: double
        description: Presión atmosférica en hPa
    Humidity:
        type: integer
        description: Porcentaje de humedad

    Clicked:
        type: boolean
        description: Actividad visitada
    Saved:
        type: boolean
        description: Actividad marcada como favorita
    Discarded:
        type: boolean
        description: Actividad descartada
    Value:
        type: number
        minimum: 0
        maximum: 5
        description: Valoración de la actividad, si es 0 la actividad no ha sido
valorada
    Token:
        type: string
        description: Token de acceso de sesión
    Password:
        type: string
        description: Constraseña
```

```
UserSetting:
        type: boolean
        description: Indicada si los datos del perfil serán compartidos
    LocationSetting:
        type: boolean
        description: Indica si la localización será compartida
    LocationAccurate:
        type: boolean
        description: Indica si la localización es exacta (true) o incorpora ruido
(false)
    WeatherSetting:
        type: boolean
        description: Indica si la información meteorológica será compartida
    CalendarSetting:
        type: boolean
        description: Indica si la información del calendario del usuario será
compartida
    Cause:
        type: string
        description: Reason to exclude user
    Admitted:
        type: boolean
        description: Indica si el usuario ha sido admitido (true) o denegado
(false)

   User:
        properties:
            id: UserId
            genre?: Genre
            birth?: Birth
    Activity:
        properties:
            id?: ActivityId
            title: Title
            description: Description
            authorid: AuthorId
            author: Author
            img?: Img
            category: Category
            begin?: Begin
            end?: End
            longitude?: Longitude
            latitude?: Latitude


   Location:
        properties:
            mocked?: Mocked
```

```
            longitude: Longitude
            latitude: Latitude
            altitude?: Altitude
            speed?: Speed

    Weather:
        properties:
            temperature: Temperature
            pressure?: Pressure
            humidity: Humidity
            temp_min?: Temp_min
            temp_max?: Temp_max
            description?: Description
            wind?: Wind
            clouds?: Clouds
    Event:
        properties:
            title: Title
            description: Description
            begin?: Begin
            end?: End
            allDay?: AllDay
            location?: Address
            availability?: Availability
            calendar?: Calendar
    Context:
        properties:
            weather?: Weather
            location?: Location
            events?: Event[]
            timestamp: datetime
    Feedback:
        properties:
            user: UserId
            activity: ActivityId
            clicked: Clicked
            saved: Saved
            discarded: Discarded
            rate: Value
            context: Context
    LocationProfile:
        properties:
            setting: LocationSetting
            accuracy: LocationAccurate
    Settings:
        properties:
            user: UserSetting
            location: LocationProfile
```

```
            weather: WeatherSetting
            calendar: CalendarSetting
    HelloAnswer:
        properties:
            result: Admitted
            reson?: Cause
    SQL:
        type: file
        fileTypes: ['*/*']
        description: Fichero con el contenido la base de datos

/session:
    /login:
        post:
            body:
                application/json:
                    type: object
                    properties:
                        user: UserId
                        pass: Password
            description: Inicio de sesión. Devuelve el token de acceso válido
            responses:
                200:
                    body: Token

    /logout:
        get:
            securedBy: [passthrough]
            description: Devuelve true en caso de cerrar la sesión del usuario,
false en otro caso
            responses:
                200:
                    body:
                        type: boolean
/activity:
    /retrieve:
        /{id}:
            uriParameters:
                id: ActivityId
            get:
                description: Devuelve el objeto actividad asociado al
identificador especificado

                responses:
                    200:
                        body: Activity
        /active:
            get:
```

```
                description: Devuelve una lista de las actividades almacenadas
cuya fecha de finalización sea igual a posterior a la actual
                responses:
                    200:
                        body: Activity[]
        /all:
            get:
                description: Devuelve una lista de todas las actividades
almacenadas
                responses:
                    200:
                        body: Activity[]
    /store:
        /{id}:
            uriParameters:
                id: ActivityId
            post:
                body: Activity
                securedBy: [passthrough]
                description: Si existe un usuario con el id especificado en el
objeto User actualiza la información del mismo y devuelve el mismo valor de su
clave, en caso contrario crea un nuevo usuario en la base de datos y devuelve el
valor de la nueva clave del usuario
                responses:
                    200:
                        body: string
    /delete:
        /{id}:
            uriParameters:
                id: ActivityId
            post:
                securedBy: [passthrough]
                description: Elimina la actividad cuyo identificador sea igual al
indicado y su feedback asociado. Devuelve true si es eliminado, false en caso
contrario
                responses:
                    200:
                        body: boolean
        /all:
            post:
                securedBy: [passthrough]
                description: Elimina todas las actividades almacenadas y su
feedback. Devuelve true si se elimina algún dato, false en caso contrario
                responses:
                    200:
                        body: boolean

/schema:
```

```
    /import:
        post:
            body: SQL
            securedBy: [passthrough]
            description: Incorpora toda la información indicada a la base de
datos
            responses:
                200:
                    body: boolean
/export:
        get:
            securedBy: [passthrough]
            description: Exporta toda la información almacenada en la base datos
            responses:
                200:
                    body: SQL

/users:
    /retrieve:
        /{id}:
            uriParameters:
                id: UserId
            get:
                securedBy: [passthrough]
                description: Devuelve la información asociada a un usuario con el
identificador especificado
                responses:
                    200:
                        body: User
        /all:
            get:
                securedBy: [passthrough]
                description: Devuelve una lista con la información asociada a los
usuarios almacenados en la base de datos
                responses:
                    200:
                        body: User[]
    /delete:
        /{id}:
            uriParameters:
                id: UserId
            post:
                securedBy: [passthrough]
                description: Elimina la información asociada al usuario con el
identificador indicado. Devuelve true si elimina al usuario indicado, false en
caso contrario
                responses:
                    200:
```

```
                        body: boolean
        /all:
            post:
                securedBy: [passthrough]
                description: Devuelve una lista con la información asociada a los
usuarios almacenados en la base de datos. Devuelve true si elimina datos, fasle
en caso contrario
                responses:
                    200:
                        body: User[]
/app:
    /hello:
        post:
            body:
                application/json:
                    type: object
                    properties:
                        user: User
                        settings: Settings
            description: Recibe la información de ajustes de un nuevo usuario, en
caso de que sea apto para el EM devuelve true, en caso contrario false
            responses:
                200:
                    body: HelloAnswer
    /context:
        post:
            body:
                application/json:
                    type: object
                    properties:
                        user: UserId
                        context: Context
            description: Actualiza el estado del usuario con el contexto
especificado. Devuelve una lista con todas las actividades recomendadas para el
usuario en dicho contexto, se recomienda no reenviar la misma actividad a menos
que su contenido haya sido actualizado
            responses:
                200:
                    body: Activity[]
/feedback:
        post:
            body:
                application/json:
                    type: Feedback
            description: Almacena o actualiza el feedback indicado en la base de
datos. En caso de almacenar o modificar algún dato devuelve true, false en caso
contrario
            responses:
```

```
            200:
                body: boolean
```

Esta interfaz sirve como contrato entre la aplicación móvil y los Gestores de Entorno con el fin de garantizar la viabilidad en las comunicaciones entre ellos. Sobre esta documentación se ha realizado el diseño del Gestor de Entorno de este TFG con el objetivo de garantizar y agilizar la compatibilidad de ambos componentes.

Se recomienda, siempre que sea posible, transformar esta documentación a una versión web con el *plugin* citado para facilitar la legibilidad.

Anexo II

Diseño de la aplicación móvil

Este anexo relata el diseño de la interfaz de la aplicación móvil, los criterios que se han seguido para diseñar gráficamente cada pantalla y la interacción que permite al usuario.

II.I. Objetivos del diseño

El diseño de una aplicación ofrece la posibilidad de convertir una aplicación en un producto de gran calidad, en la que las expectativas en su funcionalidad se reflejen en la experiencia de usuario final. La interfaz gráfica de la aplicación se ha diseñado con las guías de diseño de Apple para iOS *Human Interface Guidelines*[1] como referencia. Esta guía basa el éxito de una buena experiencia en seis principios que deben cumplirse:

- Integridad estética: representa el acierto entre el diseño de la apariencia de la aplicación y su integración con el comportamiento esperado. Busca la coherencia entre el diseño y las expectativas del usuario en su acción.
- Consistencia: una aplicación consistente implementa una interfaz familiar para el usuario, iconos, textos o terminología utilizada en otras aplicaciones y sea considerada como 'estándar'.
- Manipulación directa: hacer uso de gestos del usuario, para que sienta que interactúa con los objetos de la interfaz como si fueran objetos físicos.
- *Feedback*: o retroalimentación, sirve como refuerzo para que el usuario comprenda la implicación de sus acciones.
- Metáforas: vincular objetos virtuales con acciones u objetos cotidianos, facilitan la comprensión del sistema. La papelera como forma de eliminar contenido es un ejemplo de ello.
- Control del usuario: el usuario y no la aplicación debe tener el control. Las aplicaciones pueden realizar sugerencias o avisos pero, el usuario debe tener la sensación de control sobre la aplicación.

El mapa de navegación del prototipo incluido en el proyecto es la base de la que parte el diseño de la interfaz y la interacción con el usuario. El objetivo del prototipo es ofrecer una manera rápida y sencilla de conocer todas las recomendaciones actuales disponibles, y que el número de gestos para consultar, valorar, marcar como favorito o eliminar un ítem sea mínimo. El diseño se encuentra centrado en el usuario, de forma que siguiendo la filosofía *push* del sistema el usuario pueda liberar su tiempo para disfrutar de estas recomendaciones, sin tener que pedirlas explícitamente. En el anexo II.II se muestran capturas de la aplicación reflejadas en el mapa de navegación y ampliadas de forma individual.

[1] Apple Human Interface Guidelines: https://developer.apple.com/design/human-interface-guidelines/ios/overview/themes/

II.II. GUI de la aplicación

En esta sección se muestra una comparativa entre el prototipo del mapa de navegación y el mapa de navegación obtenido. Además se muestran capturas individuales de cada pantalla con mayor detalle.

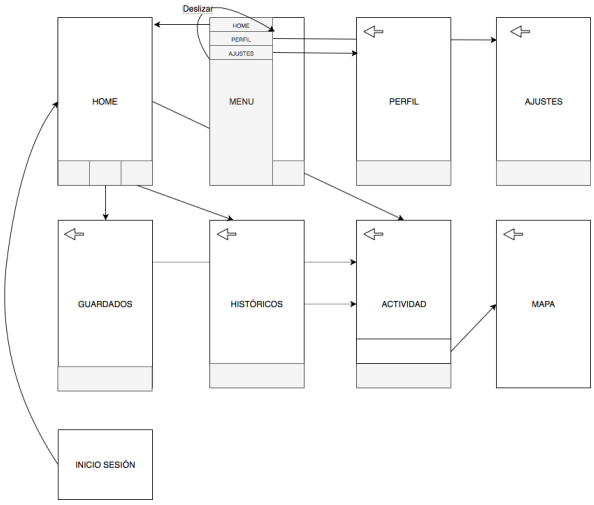


Figura 1. Prototipo del mapa de navegación

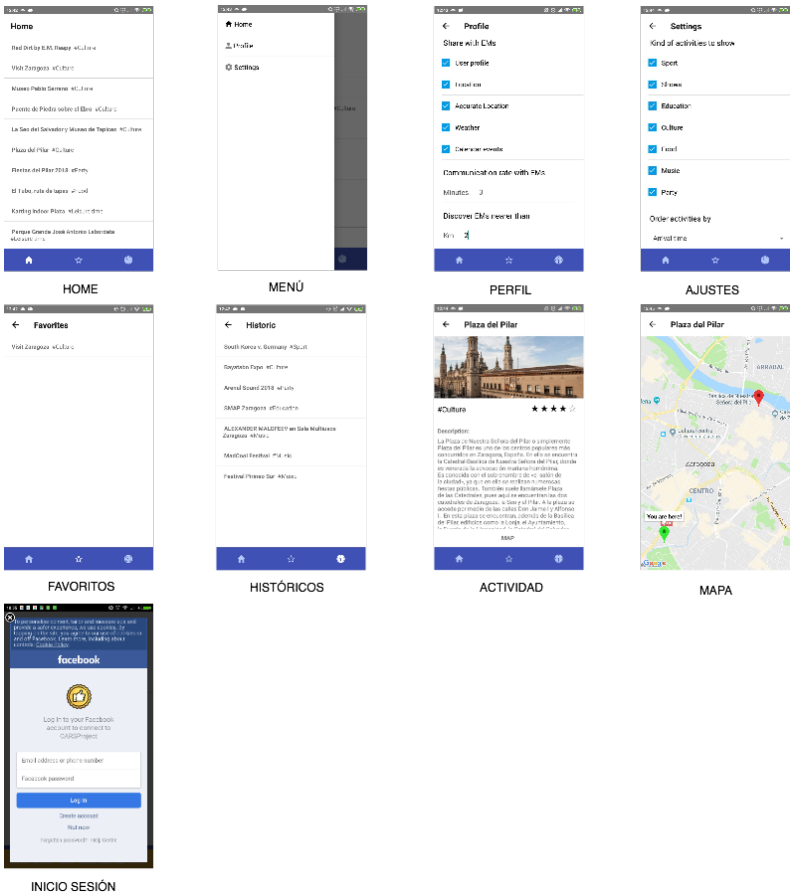
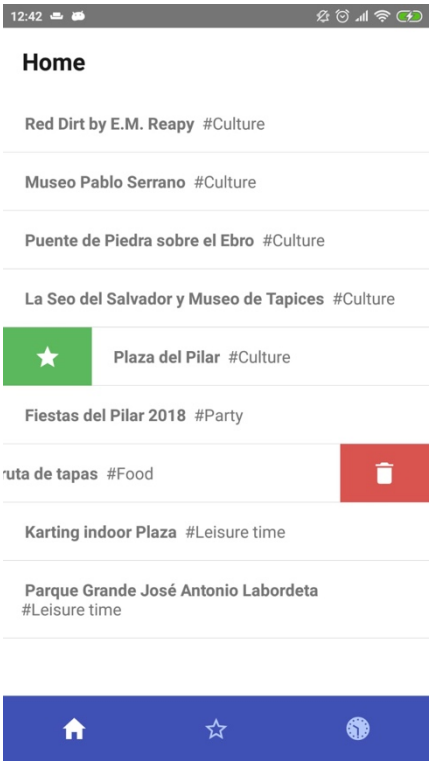
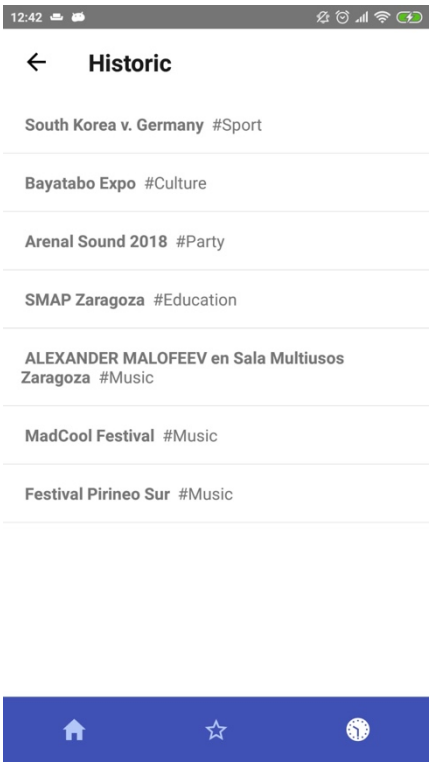


Figura 2. Pantallas del mapa de navegación

Home:

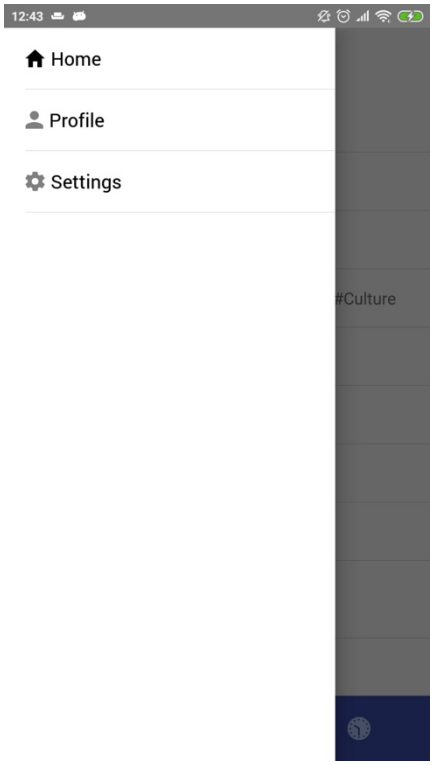


Favoritos e históricos:

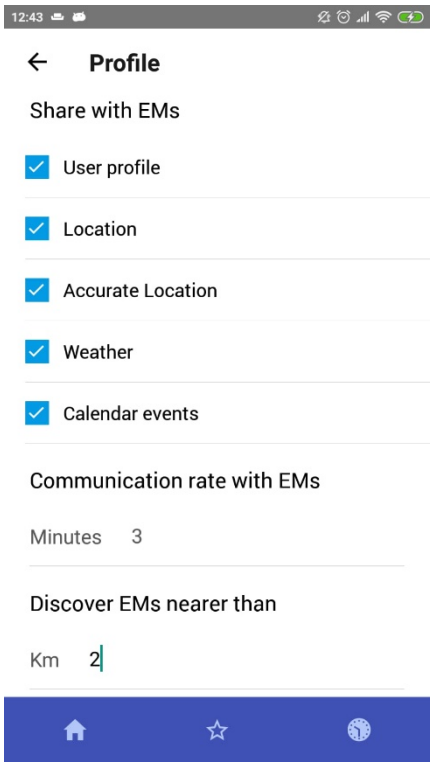


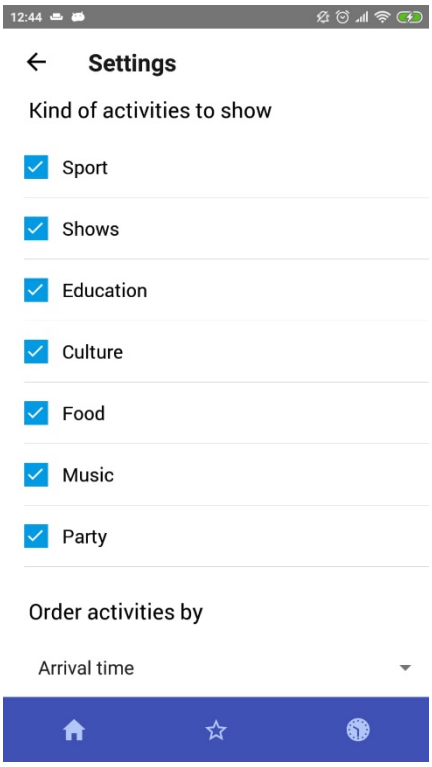
Actividad y su mapa:

Menú de navegación:

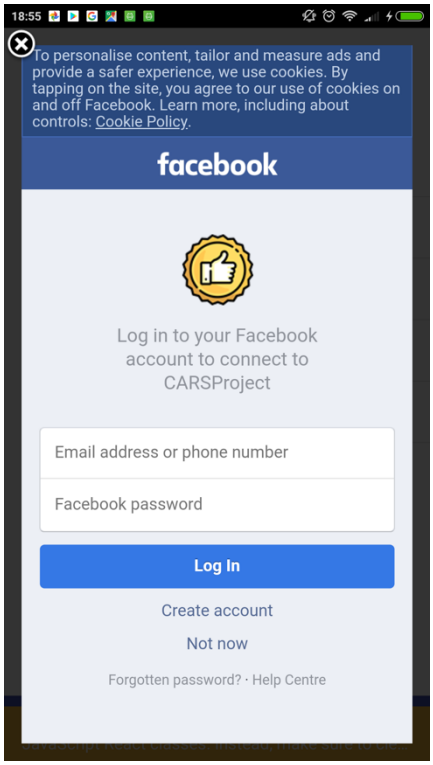


Perfil y ajustes:





Inicio de sesión:



II.III. Manual de usuario de la aplicación

La aplicación desarrollada ofrece un conjunto de acciones para que el usuario interaccione con cada ítem recibido. Estos ítems hacen referencia a diferentes tipos de actividades. Las actividades se componen: un título, una categoría, una descripción y opcionalmente: una imagen, una posición geográfica y fechas de inicio y final. La intención de cada actividad es describir una tarea que podría realizar el usuario.

Para instalar la aplicación se ha generado un APK que permite instalar la aplicación en dispositivos Android 7.1 o superior.

El primer paso para acceder a la aplicación es la pantalla de inicio de sesión, esta pantalla obliga al usuario a iniciar sesión mediante su cuenta en Facebook[1], una vez inicia sesión es redireccionado automáticamente a la pantalla de inicio o *home.* El acceso desde Facebook otorga a la aplicación la única posibilidad de conocer el nombre del usuario, además de ofrecer un código válido únicamente para esta aplicación, con el que identificar al usuario

Una vez iniciada sesión el usuario en todo momento puede acceder a un menú desplazando el margen izquierdo de la pantalla hacia el derecho.

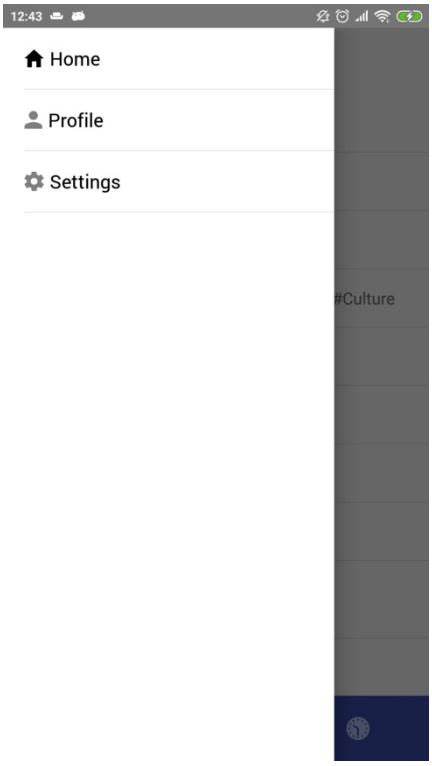


Figura 3. Menú de navegación

Este menú le permite acceder a las pantallas de *home*, perfil y ajustes. Asimismo, también puede navegar a las pantallas de *home*, favoritos e históricos con la barra inferior de navegación.

[1] Facebook: https://www.facebook.com/

Figura 4. Barra inferior de navegación

Además, puede volver a la pantalla anterior a la que se encuentre con la flecha situada en la esquina superior izquierda.

Las acciones realizables en cada pantalla son las siguientes:

- *Home*, favoritos e históricos: en estas pantallas se visualizan listados de las actividades accesibles, su título y categoría. Al pulsar sobre una actividad se redirigirá al usuario a una nueva pantalla con más detalle de esta actividad.

Desplazando de izquierda a derecha aparece un botón que permite marcar como favorita la actividad. Si el usuario se encuentra en la pantalla de favoritos el botón permite desmarcarla como favorita.

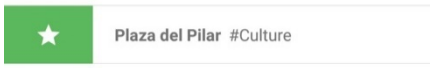


Figura 5. Botón de favoritos de actividad

Desplazando de derecha a izquierda sobre una actividad aparece un botón que permite eliminar la actividad, de forma que el usuario no la vuelva a visualizar en la aplicación.

Figura 6. Botón de borrado de actividad

La pantalla de *home* muestra las actividades recomendadas al usuario que hayan caducado, favoritos muestra la lista de las actividades que se marcaron como favoritos, e históricos muestra actividades cuya fecha de finalización es anterior a la actual.

- Actividad y mapa: en la pantalla de actividad podemos ver en detalle la actividad seleccionada. Inicialmente la valoración aparecerá sin ninguna estrella seleccionada hasta que el usuario la valore, y puede repetir esta acción de forma indefinida. Las actividades que tienen localización asociada mostrarán un botón que permite visualizar un mapa con su posición.

Figura 7. Botón de acceso al mapa

El mapa mostrado en pantalla se encuentra centrado en el punto de posición de la actividad, que se marca con un icono rojo. Además en el mapa habrá situado un icono verde que muestra la última posición registrada del usuario.

Figura 8. Mapa de posición de la actividad

- **Perfil y ajustes:** estas pantallas ofrecen una serie de parámetros al usuario que determinan el comportamiento de la aplicación. En ajustes se puede ajustar la información que se compartirá con los *EM*. Mientras que, en perfil se puede ajustar el comportamiento de la aplicación que percibe el usuario: qué categorías de actividades visualizar o qué criterio se debe seguir a la hora de organizar las actividades en la pantalla de inicio.

En la pantalla de perfil además de estos parámetros se sitúa un botón que permite cerrar la sesión del usuario.

Anexo III

Instalación y configuración de un gestor de entorno en el sistema

En este anexo se explican los pasos seguidos para implementar un gestor de entorno e introducirlo en el sistema.

III.I. Librerías utilizadas en la implementación

La implementación del *EM* de ejemplo se ha realizado utilizando el lenguaje Java en su versión 1.8, para asegurar su compatibilidad con las librerías utilizadas. Además de Java se ha utilizado el *framework* Spring Boot[1] y la herramienta Spring Tool Suite[2]. El objetivo de utilizar estos recursos es agilizar la producción del servidor que forma parte del gestor.

Una vez disponemos de un servidor web funcional, se puede pasar a instalar sobre él las librerías necesarias para realizar recomendaciones. La librería utilizada es Apache Spark 2.4.0[3]. Esta librería, además de aportar una serie de métodos que serán utilizados para generar predicciones es un *framework* para computación es clústers, factor importante en el caso de desarrollar un sistema recomendador, que maneja grandes cantidades de datos, en el *hardware* indicado.

Spark SQL[4] ofrece métodos que permiten procesar estructuras de datos. Junto a MlLib, librería orientada a *machine learning,* han sido las librerías utilizadas para implementar el recomendador. A continuación se muestra un extracto del fichero POM[5] (Project Object Model) que permite incluir estas librerías dentro del proyecto con Spring.

```
<dependency>
    <groupId>org.apache.spark</groupId>
    <artifactId>spark-core_2.11</artifactId>
    <version>2.4.0</version>
</dependency>

<dependency>
    <groupId>org.apache.spark</groupId>
    <artifactId>spark-sql_2.11</artifactId>
    <version>2.4.0</version>
</dependency>

<dependency>
    <groupId>org.apache.spark</groupId>
    <artifactId>spark-mllib_2.11</artifactId>
    <version>2.4.0</version>
```

1 Spring Boot: http://spring.io/projects/spring-boot

2 Spring Tool Suite: https://spring.io/tools

3 Apache Spark 2.4.0: https://spark.apache.org/docs/latest/

4 Apache Spark SQL: https://spark.apache.org/docs/latest/sql-programming-guide.html

5 Apache Maven Project. Introduction to the POM: https://maven.apache.org/guides/introduction/introduction-to-the-pom.html

```
</dependency>
```

En este trabajo se han utilizado estas tecnologías presentadas, pero no es necesario utilizar ninguna de ellas para asegurar el funcionamiento del gestor en el sistema; solo se trata de una sugerencia. El único requisito imprescindible es implementar la API presentada en el Anexo I, en lo que respecta al apartado *app* de la documentación.

III.II. Configuración del gestor en el sistema

El objetivo de la arquitectura expuesta a lo largo del trabajo es ofrecer un sistema descentralizado, en el que puedan coexistir diferentes implementaciones de gestores de entorno, mientras la implementación de la aplicación móvil se mantiene única. La eliminación de la entidad central se solventa introduciendo la información de todos los *EMs* en la misma aplicación.

La aplicación móvil desarrollada incluye un fichero tipo JSON que almacena información acerca del gestor necesaria para poder cumplir los requisitos del sistema. A continuación, se muestra un ejemplo de gestor registrado en el fichero:

```
{
    "address": "http://192.168.31.31:8080",
    "id": "DemoEM4",
    "name": "Demo EM 4",
    "coords": {
        "lat": 41.6,
        "lon": -0.9
    }
}
```

- Address*:* indica la dirección que la aplicación utilizará para realizar las peticiones.
- Id: identificador único del *EM*, que la aplicación lo utiliza para verificar la autenticidad de las recomendaciones.
- Name: descriptor del *EM*, su uso es solo informativo.
- Coords: optativo, indica las coordenadas físicas del *EM,* se utiliza para evitar que la aplicación realice peticiones a gestores que por lejanía se considerían incompatibles.

En el momento en el que un nuevo gestor desee formar parte del sistema se debe comunicar con el administrador de la aplicación e informar de los campos *addres, name* y *coords* si es necesario, cuando vaya a ser introducido al sistema, será informado con el identificador que debe utilizar. Una vez los usuarios de la aplicación actualicen su dispositivo podrán acceder al gestor.

Para realizar el despliegue del Gestor de Entorno es necesario un servidor, en este caso cuyo sistema operativo, sea compatible con Java 1.8. Además necesita una dirección a través de la cual la aplicación móvil pueda realizar peticiones HTTP. Además, se recomienda utilizar Spring Tool Suite para iniciar la aplicación y poder visualizar la traza de acciones de los usuarios en el sistema en tiempo real. En caso de no disponer de esta herramienta, se puede generar un ejecutable utilizando tanto Gradle 4 o superior, como Maven 3.2 o superior, como se muestra a continuación:

```
    $ ./gradlew build && java -jar build/libs/gs-spring-boot-0.1.0.jar (Gradle)
    $ mvn package && java -jar target/gs-spring-boot-0.1.0.jar (Maven)
```

Anexo IV

Requisitos del sistema

Este anexo enumera los requisitos detectados durante la fase de diseño del sistema que finalmente ha sido implementado. Se encuentra dividido en requisitos (funcionales y no funcionales) de la aplicación móvil y requisitos del Gestor de Entorno.

IV.I. Requisitos de la aplicación móvil

Requisitos funcionales:

- La aplicación debe capturar el contexto del usuario.
- La aplicación debe capturar la posición del usuario.
- La aplicación debe permitir iniciar y cerrar sesión del usuario.
- La aplicación debe permitir visualizar las actividades en forma de listado y detalladas.
- La aplicación debe permitir añadir actividades a favoritas.
- La aplicación debe permitir eliminar actividades como favoritas.
- La aplicación debe permitir ocultar actividades.
- La aplicación debe permitir valorar actividades.
- La aplicación debe permitir ajustar preferencias de privacidad del usuario, qué se comparte y cada con qué frecuencia.
- La aplicación debe permitir ajustar opciones de personalización en la visualización de actividades.
- La aplicación debe dejar de mostrar actividades obsoletas.
- La aplicación debe marcar actividades cuya fecha de expiración sea anterior a la actual como actividades históricas.
- La aplicación tener un registro de los *EMs* disponibles en el sistema.
- La aplicación debe poder comunicarse con los *EMs*.

Requisitos no funcionales:

- La aplicación debe permitir organizar las actividades por orden alfabético, fecha de llegada o cercanía.
- La aplicación debe permitir visualizar en un mapa la posición de la actividad.
- La aplicación debe ser sencilla y usable.
- La aplicación debe ser responsable en el gasto de batería.
- La identificación de cada usuario debe producir un identificador único para cada usuario.

IV.II. Requisitos del Gestor de Entorno (*EM)*

Requisitos funcionales:

- El *EM* debe permitir almacenar, consultar, editar y eliminar actividades.
- El *EM* debe permitir registrar la información asociada a los usuarios.
- El *EM* debe permitir recopilar el *feedback* de los usuarios.
- El *EM* debe ser capaz de generar recomendaciones dado un usuario y un contexto.

Requisitos no funcionales:

- El *EM* debe implementar mecanismos de seguridad para modificar sus datos.
- El recomendador debe generar recomendaciones con el objetivo buscar mayor precisión que *recall.*

Anexo V

Artículo “Towards the Implementation of a Push-Based Recommendation Architecture” publicado en el congreso SMAP 2018

Este anexo incluye el trabajo de investigación publicado en el *13th International Workshop on Semantic and Social Media Adaptation and Personalization (SMAP) 2018*. El artículo se elaboró utilizando los estudios elaborados en el desarrollo de este trabajo, en el artículo se muestra el avance de una implementación como prueba de concepto de una arquitectura que permita el desarrollo de sistemas de recomendación basado en el contexto para usuarios móviles, en el que el proceso de recomendación no es iniciado por el usuario sino por entidades externas de forma proactiva.

En el artículo se describe el proceso de implementación de la aplicación sobre la que se basa este TFG. Posteriormente al artículo, la aplicación ha sido refinada para lograr satisfacer los requisitos expuestos en el Anexo IV.

a Push-Based Recommendation Architecture", 13th International Workshop on Semantic and Social Media Adaptation and Personalization (SMAP 2018), Zaragoza (Spain), IEEE, Electronic ISBN 978-1-5386-8225-8, USB ISBN 978-1-5386-8224-1, Print on Demand (PoD) ISBN 978-1-5386-8226-5, pp. 87-92, September 2018. https://doi.org/10.1109/SMAP.2018.8501875

Towards the Implementation of a Push-Based Recommendation Architecture

Sergio Ilarri
University of Zaragoza, I3A
María de Luna, 1, Zaragoza (Spain)
Email: silarri@unizar.es

Manuel Herrero
University of Zaragoza
María de Luna, 1, Zaragoza (Spain)
Email: 681940@unizar.es

***Abstract*—Recommender systems play a key role for personalization. These systems help users when they need to choose among a variety of possible options, that may easily overwhelm them. In particular, in recent years, there has been increased interest in the development of recommender systems for mobile users. The so-called Context-Aware Recommender Systems (CARS) provide users with suitable suggestions by considering not only their preferences but also their specific context conditions.**

In this paper, we present our work in progress towards the development of a proof-of-concept implementation of a system architecture that can support the deployment of context-aware recommender systems for mobile users. We focus on scenarios where the recommendation process is not initiated by the user but by external entities that proactively trigger appropriate suggestions in different contexts.

I. Introduction

Recommender systems [1], [2] aid users to select suitable items (movies that they may like to watch, books to read, points of interest to visit, restaurants where they could have lunch, etc.) when the number of choices is large. Personalized recommender systems suggest items based on the preferences of each individual user, which are automatically learnt based on the information provided by the users (a typical feedback system is based on providing explicit ratings/scores). Moreover, it has been argued that, besides the user preferences, the context of the user is another important factor that should be considered when making recommendations. The intuition behind this idea is that context attributes can affect the relevance of certain items and types of items. Therefore, taking the context of the user into account will contribute to providing more suitable recommendations, which is important especially in scenarios of mobile computing, where the context of the user may change constantly. Thus, the traditional $User \times Item \rightarrow Rating$ two-dimensional recommendation model has given rise to a $User \times Item \times Context \rightarrow Rating$ three-dimensional model. These novel types of recommender systems are called *Context-Aware Recommender Systems* (*CARS*) [3], [4], [5], [6].

In mobile computing environments, the context of a mobile user is highly dynamic: due to his/her movements, his/her location as well as the surrounding environment is changing constantly. Therefore, CARS could be particularly relevant in those types of scenarios. The location of the user is such a key context attribute that the specific term *Location-Aware Recommender Systems* (*LARS*) [7], [8] has been proposed to denote those recommender systems that focus on considering the relevance of the location of the user to provide more suitable recommendations. Besides, in a typical mobile computing setting, an approach where the user receives recommendations without the need of investing effort in explicitly asking for them, could be more appropriate. That is, *push-based recommendations* (proactive recommender systems) would be the preferred option, as opposed to *pull-based recommendations* (reactive recommender systems), as long as the recommendations provided are relevant.

In [9], we presented a theoretical framework for push-based recommendations in mobile environments. In this paper, we report our work-in-progress towards the development of a prototype for mobile devices, that could serve as a proof of concept of our proposal. The rest of the paper is structured as follows. First, in Section II, we provide an overview of the proposed push-based architecture. Second, in Section III, we summarize some alternative technologies that could be used for the development of a prototype. Then, in Section IV, we provide some details about the development of the prototype and the experience acquired. Finally, in Section V, we summarize our conclusions and the current status of the development.

II. Basics of the Push-Based Recommendation Architecture

From an abstract viewpoint, the push-based recommendation architecture presented in [9] is composed of the following elements:

- *Contexts*. In the proposed model, we call context to the scope or purpose of a recommendation (e.g., research, tourism, leisure, etc.).
- *Environments*. An environment is a physical or virtual area where a number of users can perform different types of activities and receive information concerning that specific environment. A user may belong to several environments at the same time. For example, a user in a mall in a city may belong to an environment defined by the limits of the commercial center and, at the same time, to an environment defined by the area of the whole city. Besides, the user may have logged into a specific website using his/her mobile phone, and therefore may also belong to an environment defined by that website.
- *Agents*. Agents are entities that generate (communicative and/or physical) *events* and they can be *users* and

Environment Managers (*EMs*). Users can be people receiving recommendations (e.g., a tourist in a city, a person in a shopping center, etc.) as well as special users acting on behalf of a business or organization and offering information about certain items or activities that can be performed (e.g., a shop announcing special offers). An EM is in charge of a specific environment and handles the membership of users within the environment and the communication within the environment.

- *Activities*. We use the term *activity* to denote a certain activity on an item. As opposed to simply *items*, in the architecture proposed in [9], we consider activities as the basis for recommendation (e.g., a system can recommend going to a specific restaurant for lunch or for dinner, rather than just recommend a restaurant).

In this paper, we focus on the software that will be executed on the mobile devices of the users and on the architecture of EMs.

III. Technologies Considered

In this section, we describe the technologies that we have considered so far in the development of our prototype. First, we provide an overview of the three types of approaches that can be followed for the development of mobile apps. Then, we focus on React Native. Afterwards, we describe some functionalities of Expo.

A. Native Apps vs. Multi-platform Solutions

According to IDC, the market share of operating systems for mobile devices in the first quarter of 2017 was as follows [10]: 85.0% for Android, 14.7% for iOS, 0.1% for Windows, and 0.1% for other operating systems. Even though the market is not as fragmented as a few years back (e.g., besides Google, Microsoft and Apple, companies such as Nokia and RIM were also popular in the market of mobile operating systems around the year 2010), there are still at least three mobile platforms that companies usually target, in order to reach as many mobile users as possible. For the development of mobile apps for this market, several options can be considered:

- *Develop native apps* (platform-specific development). With this approach, we would need to develop independent implementations for each target operating system (e.g., Android and iOS). This option usually leads to maximum flexibility and performance, as it directly exploits the resources of each platform, but it also implies a higher cost. Besides, it implies the need of managing several platform-dependent versions that need to be updated individually.
- *Develop a mobile web app*. In this case, the idea is to exploit the functionalities of standard web technologies and web browsers, usually available on any mobile device, to ensure the interoperability. By using technologies such as HTML5, CSS and JavaScript, we achieve the portability objective: only one mobile app needs to be developed, which can then be run virtually on any mobile platform.
- *Develop hybrid apps with the support of a framework*. Hybrid apps are built using frameworks such as *Adobe PhoneGap* [11], *Apache Cordova* [12], *Ionic* [13], *React Native* [14], *Xamarin* [15], and *Appcelerator* [16]. They exploit web technologies but at the same time provide access to platform-dependent functionalities, such as sensors, and offer a native look-and-feel for the graphical user interface.

Nowadays, the use of frameworks to develop hybrid apps seems to be the most popular choice, as it considers the trade-off between flexibility and development cost.

B. React Native

Finally, our chosen technology to develop the prototype was *React Native* [14], which is used by many popular apps [17], such as Facebook, Instagram, Skype, and Pinterest. React Native facilitates the development of native mobile apps using JavaScript [18] and React [19] (a component-based JavaScript library to build user interfaces that follows the "learn once, write anywhere" principle, which emphasizes the relevance of developing apps for each platform but using a common framework). In the following, we summarize some interesting benefits of React Native:

- *JavaScript-based to build mobile apps*. React Native allows to build native mobile apps using only JavaScript (and the React library). The mobile apps developed with React Native are native apps, rather than for example mobile web apps based on the use of HTML5.
- *Hot reloading*. It supports reloading the app without recompiling (new versions of the modified files are injected while the app is running), even retaining the application state if needed. This leads to a quick and convenient development.
- *Wrapping of native code*. If needed (e.g., for performance reasons), it is possible to write native code and execute it as part of the React Native app. This may complicate the maintainability of the project, due to the introduction of platform-specific components, but on the other hand enables fine-tuning it for each target operating system.

C. Expo

Besides, we initially considered also the use of *Expo* [20], which is a set of tools, libraries and services to build native iOS and Android apps by using JavaScript. Expo facilitates the development of React Native apps [21]. One of the most interesting features is the support for easy project management, as it can quickly push code updates to the mobile device with minimum effort from the developer and without the need to use specific development platforms such as Android Studio [22] (for Android devices) or Xcode [23] (for iOS devices). Besides, it provides several modules that offer interesting functionalities. Among those modules, we have tested and used the following ones in the first versions of the prototype:

- *Facebook* [24], which allows authenticating the user using his/her Facebook account. Once authenticated,

with the access token obtained, it is possible to use the *Graph API* [25] developed by Facebook to retrieve (if allowed by the user) basic information about his/her profile (such as his/her city and date of birth). It should be noted that in the current prototype it would be possible to switch from an authentication system based on Facebook to another one. It might also be possible even to go without authentication, although a user identifier (e.g., composed by a local alias plus a unique identifier for the mobile device) would be needed in order to associate the ratings provided by the user to a user ID.

- *Calendar* [26], which allows interacting with the device's system calendars in order to retrieve and query information about events and reminders.
- *Location* [27], which allows retrieving the current location of the mobile device and it also offers the possibility to subscribe to receive location updates.
- *SQLite* [28], which allows querying a local SQLite database, which can be used to store information locally on the mobile device (e.g., recommended items and context variables).

These modules facilitate the access of different types of data that may be relevant to the recommender system. For example, querying the user's calendar is as simple as calling the method *Expo.Calendar.getEventsAsync(calendarIds, startDate, endDate)*, which retrieves all the events included in the calendars identified by the identifiers provided (*calendarIds*) and considering the specified time period ([*startDate, endDate*]). An inconvenience found is that the behavior of this method is not independent of the operating system used: on iOS, it returns all the events whose time span overlaps with the temporal interval [*startDate, endDate*] provided as an argument; on Android, it only returns the events that start on or after *startDate* and end on or earlier than *endDate*. So, if the app developed is executing on Android, it is not possible to directly obtain, in an efficient way, all the events that overlap with a given period of time.

IV. Development of a Prototype

In this section, we provide some details about the development of our prototype. First, we present an overview of the workflow of the app. Second, we describe the context data that are retrieved automatically by using sensors and information provided by other apps. Finally, we summarize some difficulties found with the use of background tasks.

A. Overall Workflow of the Prototype

An overview of the workflow considered in the prototype implemented is shown in Figure 1:

1) First, the mobile device of the user captures some context data, by exploiting different sources, like local sensors, information provided by installed applications, and/or external services that may offer information about the area.
2) Second, part of those user context data may be sent to the EM. It should be noted that the user can configure, through the mobile app, which data he/she is willing to share with the EM as well as the context data that he/she allows to share with the local app. For example, the user may allow retrieving information about his/her age from his/her Facebook profile, but maybe he/she does not want this information to be transmitted to any EM: in that case, the age is not sent to the EM but could be used by the app later as a post-filtering step.
3) Finally, the EM sends a set of recommended activities to the mobile device of the user. The app in the mobile device could apply a post-filtering by considering private attributes not communicated to the EM as well as other user preferences.

As shown in the figure, the communication between the mobile app executing on the mobile device of the user and the EM is performed through the exchange of JSON files, which is convenient given that it is a basic common structure well supported by React Native.

It should be noted that, according to the workflow presented, there are two basic components that need to be implemented: the mobile app, that executes on the mobile user device, and the EM architecture, that will probably be executed on a fixed computer. For the moment, we have focused our attention on the implementation of the functionality that runs on the mobile device. Nevertheless, we have also designed a minimal interface (RESTful API) that any EM's implementation should provide:

- {*@EM*}*/add/user/*{*userID*}*/*{*devToken*} *POST* adds a user to the environment of the EM specified; if the user is already registered, then potential changes related to his/her registration will be recorded. In the parameters, *@EM* represents the address of the EM receiving the request, *userID* is a user token that identifies the user (e.g., obtained through an authentication based on a Facebook account), and *devToken* is a token that identifies the mobile device of the user. For example, in an implementation with Expo the *devToken* could be an *expoToken*, and *Firebase Cloud Messaging* [29] also works with a token that identifies the device. If the EM rejects the registration in the environment, then a key-value pair, with an error code and its description, is returned to the user. A JSON file can be sent along with the call to this method to provide information that can be used by the EM to decide about the request (e.g., the location of the user).
- {*@EM*}*/delete/user/*{*userID*} *POST* removes a user from the environment of the EM specified.
- {*@EM*}*/event/user/*{*userID*}*/*{*length*} *POST* sends a JSON file to the EM with information about the current context of the user. The new *length* parameter is an optional parameter that indicates the maximum number of recommendations that the device wants to retrieve for a given situation. Upon calling this method, the EM could potentially trigger a recommendation process to try to find activities that are relevant in the context provided. The app executing on the user device could call this method of the EM periodically, by following the required context update policy.

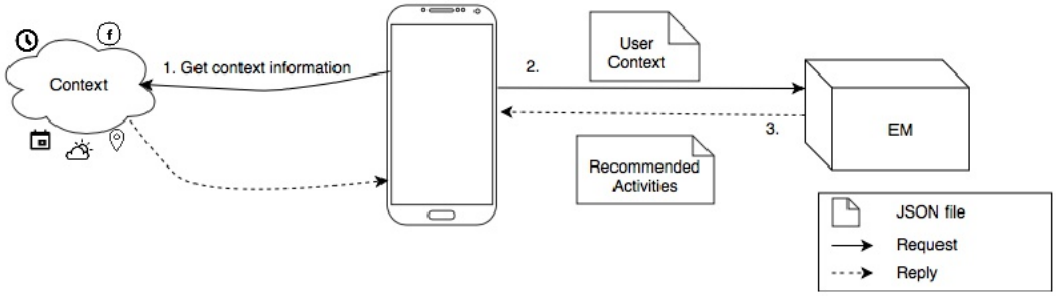


Fig. 1. General workflow in the implemented architecture

- {*@EM*}*/add/activity/ POST* adds an activity to the list of activities stored by the EM. This operation has not been defined for end users of the recommendation app; instead, it is constrained to be used by specific users (like administrators of EMs), that will be identified by the EM through an authentication token. The activity added is defined by a JSON file. Our prototype relies on a predefined catalog of types of activities, that is stored in a resource file associated to the app. If the catalog is expanded, a new version of the app with the resource file updated could be downloaded from the corresponding repository. Besides, if an EM defines a new own type of activity, it is added to a miscellaneous category ("Others"), which the developer of the mobile app could eventually integrate to the catalog, if there is a successful adoption of that type of activity by other EMs.
- {*@EM*}*/remove/activity/ POST* removes an activity from the list of activities stored by the EM. This operation acts as the counterpart of the previous one.
- {*@EM*}*/get/activity/*{*activityID*} *POST* obtains information about the activity identified by the parameter *activityID*, which should be in the list of activities stored by the EM.

By implementing this basic set of methods, any recommendation provider can participate and interact with the recommender mobile app running on the mobile device of the user. This structure offers high flexibility, as any provider is free to decide how to implement the services offered. We also envision the possibility of empowering the user with options in the app that enable him/her to opt in/out of the possibility of using certain EMs, which could be an effective measure to protect the user from potential abuses (like EMs flooding users with irrelevant information).

B. Context Data

In order to offer suitable recommendations to a mobile user, mobile CARS exploit context data to assess the potential relevance of different types of items/activities. Even though the user could be explicitly required to fill his/her context data when he/she provides a rating, by using a form, this can lead to a huge amount of attribute values not provided or left with default values (i.e., noise). For example, according to [30], the STS dataset [31], [32], which are data collected by the STS (South Tyrol Suggests) app, includes 14 context attributes but 89.37% of the context attribute values in the dataset are actually empty. Therefore, context data should be retrieved automatically, by using sensors of different types (physical, virtual, social, and/or human sensors [33]), if possible.

Currently, we are considering the following context data, as long as the user allows access to these data (see Listing 1 as an example of basic code that retrieves those data):

Listing 1. Basic code for retrieving context data when loading the app

```
async componentWillMount() {
   try {
      // Allow notifications
      myNotification._registerForPushNotifi-
          cationsAsync();
      // Get calendar events
      myCalendar._getCalendarAsync();
      // Get location and current weather
      this._getConditionsAsync();
      user = await AsyncStorage.getItem("user");
      token = await AsyncStorage.getItem("token");
      if (user !== null && token !== null) {
         // Go to Home
         this.props.navigation.navigate( "Home");
      } else {
         // No user available: go to Login
         this.props.navigation.navigate( "Login");
      }
   } catch (error) { // Error - repeat login
   this.props.navigation.navigate("Login");
}
```

- *Basic user profile attributes*. The mobile app can access the city and date of birth registered in the user's Facebook account (e.g., see Section III-C).
- *Location*. The GPS location of the mobile device is retrieved. For example, using the *Location* module of Expo (see Section III-C), the code shown in Listing 2 could be used.

Listing 2. Basic code for retrieving the current location

```
export async function _getLocationAsync() {
   try {
      // Ask the permission needed
      let status = await
          Permissions.askAsync(
          Permissions.LOCATION);
      if (status !== "granted") {
         throw "Permission to access the
             location was denied";
      }
      let location = await
          Location.getCurrentPositionA-
          sync({});
      // Save the location in the SQLite DB
      await AsyncStorage.setItem("location",
          JSON.stringify(location));
```

```
        await _getCurrentWeather(location);
    } catch (error) { console.log(error); }
}
```

- *Calendar data*. The calendars of the user are queried to obtain events that may affect the current date and/or time period (see Section III-C). Specifically, in our prototype, we are considering basic features of the events, like their time span (date/s and, if defined, also time), title, and description if available. Besides, we also obtain the name of the calendar where the event is registered, as it can reveal relevant information regarding the category of the event (e.g., the user may have calendars with names such as "Work", "Leisure", or even third-party calendars such as "National Festivities"). If the name of the calendar includes an email address, then the name is not retrieved, due to privacy concerns and the fact that an email address is usually not significant to characterize the event. An example code fragment is shown in Listing 3.

Listing 3. Basic code for retrieving calendar events for the next 7 days

```
export async function _getCalendarAsync() {
    try {
        // Ask the permission needed
        let {  status } = await
            Permissions.askAsync("calendar");
        if (status !== "granted") {
            throw "Permission to access
                calendar was denied";
        }
        // Gets an array of calendars
        let calendars = await
            Calendar.getCalendarsAsync();
        AsyncStorage.setItem("calendars",
            JSON.stringify(calendars));
        // Query the calendars
        let calendarIds = [];
        const end = new Date();
        const start = new Date();
        end.setDate(start.getDate() + 7);
        for (let index = 0; index <
            calendars.length; index++) {
            calendarIds.push(
                calendars[index].id + "");
        }
        let events = await
            Calendar.getEventsAsync(
            calendarIds, start, end);
        AsyncStorage.setItem("events",
            JSON.stringify(events));
    } catch (error) { console.log(error); }
}
```

- *Twitter data*. Trending topics in Twitter near the location of the user are obtained by using the Twitter API [34], thanks to the method *GET trends/place*, which retrieves the 50 trending topics for a specific WOEID (Yahoo! Where On Earth ID). To minimize the battery consumption of the mobile device and the amount of communications, and given that exploiting Twitter data may not be trivial, they can be obtained by the interested EM (rather than by the mobile device), in case information about the trending topics could be relevant to the recommendations.

- *Weather data*. Basic weather information in the current location of the user is obtained by using the OpenWeatherMap API [35] (see Listing 4).

Listing 4. Basic code for retrieving the current weather for a given locat

```
export async function
    _getCurrentWeather(location) {
    let lat = location.coords.latitude;
    let lon = location.coords.longitude;

    try {
        let response = await fetch(
            "http://api.openweathermap.org/" +
            "data/2.5/weather" + "?lat=" + lat
            + "&lon=" + lon + "&appid=" +
            oauth.openweathermap);
        let resJson = await response.json();
        AsyncStorage.setItem("weather",
            resJson);
    } catch (error) {
        ...
    }
}
```

As commented before (see Section IV-A), the values of these context attributes are initially stored on the user's mobile device and exploited locally. Additionally, if the user is willing to share these data, he/she can allow communicating them to the EMs, thus enabling not only a postfiltering of recommendations but also a prefiltering by the EMs themselves.

C. Difficulties Related to the Use of Background Tasks: Expo vs. React Native

Even though Expo boosts the development of mobile apps, thanks to the functionalities that it offers on top of React Native, we have found a major limitation. Particularly, the implementation of background tasks using that framework seems to be challenging. Running a task in the background would be needed in order to detect context changes, communicate them to the EMs where the user is currently active (if the user wants to share those context changes), and register the user into appropriate new environments as he/she moves around, even if the recommendation app is not executing on the foreground.

A possibility is the use of the library *react-native-background-task* [36], which supports the scheduling of a periodic task that is executed even if the app is executing on the background or even closed, as long as the execution period required is not higher than about 15 minutes. This means that, for example, a new environment could be detected with a delay of up to 15 minutes (i.e., if the user enters the limits of a new environment just after the background task has been executed for the last time and the user does not open the recommender app during that time lapse), even if the background task is programmed to be executed with the maximum frequency supported.

As an alternative, for the specific case of Android, *Headless JS* [37] could be used. The advantage of this library is that it does not impose any limitation regarding the frequency of execution of background tasks. The main disadvantage is that at the moment there is no port of Headless JS for iOS. Besides, it is not compatible with the use of Expo. Another existing library is *react-native-background-job* [38], which is based on React Native's HeadlessJS, but it also implies a minimum execution period of 15 minutes.

Even though we started the implementation of the prototype with Expo, we have finally decided not to use it, due to the aforementioned difficulties. Instead, we now directly

use React Native and Headless JS for the implementation of background tasks. Our preliminary tests show that Headless JS is a suitable approach for our problem. Therefore, we are currently re-implementing some parts of the code that relied on Expo's functionalities (mainly, tasks related to the capture of context data).

V. Conclusions and Future Work

In this paper, we have presented our current development of a prototype of a push-based recommendation architecture for mobile users. We have described the current status of the development along with some ideas about context data that could be exploited during the recommendation process by the EMs. Besides, we have mentioned several technologies that could be applied for the development. Our initial prototype was being developed using Expo, but now we are turning our attention to just using React Native, due to the limitations explained regarding the use of background tasks in Expo. The development of the prototype is still ongoing. Future immediate steps are refining and implementing the architecture of an EM, which includes defining the structure of the JSON files describing activities and the context of the user, among other tasks.

Acknowledgment

This work has been supported by the project TIN2016-78011-C4-3-R (AEI/FEDER, UE), the Government of Aragon - Aquitaine-Aragon project PASEO 1.0 (AQ-8), and DGA-FSE (COS2MOS research group).

References

[1] F. Ricci, L. Rokach, B. Shapira, and P. B. Kantor, *Recommender Systems Handbook*. Springer, 2011.

[2] J. Bobadilla, F. Ortega, A. Hernando, and A. Gutiérrez, "Recommender systems survey," *Knowledge-Based Systems*, vol. 46, pp. 109–132, 2013.

[3] G. Adomavicius and A. Tuzhilin, "Context-aware recommender systems," in *ACM Conference on Recommender Systems (RecSys)*. ACM, 2008, pp. 335–336.

[4] G. Adomavicius, B. Mobasher, F. Ricci, and A. Tuzhilin, "Context-aware recommender systems," *AI Magazine*, vol. 32, no. 3, pp. 67–80, 2011.

[5] Q. Liu, H. Ma, E. Chen, and H. Xiong, "A survey of context-aware mobile recommendations," *International Journal of Information Technology & Decision Making*, vol. 12, no. 1, pp. 139–172, 2013.

[6] M. del Carmen Rodríguez-Hernández and S. Ilarri, "Pull-based recommendations in mobile environments," *Computer Standards & Interfaces*, vol. 44, pp. 185–204, February 2016.

[7] J. J. Levandoski, M. Sarwat, A. Eldawy, and M. F. Mokbel, "LARS: A location-aware recommender system," in *28th International Conference on Data Engineering (ICDE)*. IEEE, April 2012, pp. 450–461.

[8] M. del Carmen Rodríguez-Hernández, S. Ilarri, R. Trillo-Lado, and R. Hermoso, "Location-aware recommendation systems: Where we are and where we recommend to go," in *International Workshop on Location-Aware Recommendations (LocalRec)*, vol. 1405. CEUR Workshop Proceedings, September 2015, pp. 1–8.

[9] R. Hermoso, S. Ilarri, R. Trillo-Lado, and M. del Carmen Rodríguez-Hernández, "Push-based recommendations in mobile computing using a multi-layer contextual approach," in *13th International Conference on Advances in Mobile Computing and Multimedia (MoMM)*. ACM, December 2015, pp. 149–158.

[10] IDC, "Smartphone OS," https://www.idc.com/promo/smartphone-market-share/os, May 2017, last Access: May 26, 2018.

[11] Adobe Systems Inc., "Adobe PhoneGap," https://phonegap.com/, last Access: May 26, 2018.

[12] The Apache Software Foundation, "Apache Cordova," https://cordova.apache.org/, last Access: May 26, 2018.

[13] Drifty Co. (d/b/a "Ionic"), "Ionic Framework," https://ionicframework.com/, last Access: May 26, 2018.

[14] Facebook Inc., "React Native – Build native mobile apps using JavaScript and React," https://facebook.github.io/react-native/, last Access: May 26, 2018.

[15] Xamarin Inc., "Xamarin," https://www.xamarin.com/, last Access: May 26, 2018.

[16] Axway, "Appcelerator," https://www.appcelerator.com/, last Access: May 26, 2018.

[17] Facebook Inc., "Who's using React Native?" https://facebook.github.io/react-native/showcase.html, last Access: May 26, 2018.

[18] Mozilla and individual contributors, "JavaScript guide," https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide, last Access: May 26, 2018.

[19] Facebook Inc., "React – A JavaScript library for building user interfaces," https://reactjs.org/, last Access: May 26, 2018.

[20] 650 Industries, Inc., "Expo," https://expo.io/, last Access: May 26, 2018.

[21] ——, "Expo – frequently asked questions: What is the difference between Expo and React Native?" https://docs.expo.io/versions/latest/introduction/faq#what-is-the-difference-between-expo-and, last Access: May 26, 2018.

[22] Google LLC and Open Handset Alliance, "Android Studio," https://developer.android.com/studio/, last Access: May 26, 2018.

[23] Apple Inc., "Xcode," https://developer.apple.com/xcode/, last Access: May 26, 2018.

[24] 650 Industries, Inc., "Expo – Facebook module," https://docs.expo.io/versions/latest/sdk/facebook, last Access: May 26, 2018.

[25] Facebook Inc., "Graph API – user," https://developers.facebook.com/docs/graph-api/reference/v3.0/user, last Access: May 26, 2018.

[26] 650 Industries, Inc., "Expo – Calendar module," https://docs.expo.io/versions/v27.0.0/sdk/calendar, last Access: May 26, 2018.

[27] ——, "Expo – Location module," https://docs.expo.io/versions/v27.0.0/sdk/location, last Access: May 26, 2018.

[28] ——, "Expo – SQLite module," https://docs.expo.io/versions/v27.0.0/sdk/sqlite, last Access: May 26, 2018.

[29] Google, "Firebase Cloud Messaging," https://firebase.google.com/docs/cloud-messaging/send-message?hl=es-419, last Access: May 26, 2018.

[30] S. Ilarri, R. Trillo-Lado, and R. Hermoso, "Datasets for context-aware recommender systems: Current context and possible directions," in *First Workshop on Context in Analytics (CiA), in conjunction with the IEEE 34th International Conference on Data Engineering (ICDE)*. IEEE, April 2018, pp. 25–28.

[31] M. Braunhofer, M. Elahi, and F. Ricci, "Context-aware dataset: STS - South Tyrol Suggests mobile app data," 2013, DOI: 10.13140/RG.2.2.34480.97281.

[32] ——, "STS: A context-aware mobile recommender system for places of interest," in *22nd International Conference on User Modeling, Adaptation, and Personalization (UMAP)—Posters, Demos, Late-breaking Results and Workshop*. CEUR Workshop Proceedings, 2014, pp. 75–80.

[33] S. Ilarri, O. Wolfson, and T. Delot, "Collaborative sensing for urban transportation," *IEEE Data Engineering Bulletin*, vol. 37, no. 4, pp. 3–14, December 2014, special Issue on Urban Informatics.

[34] Twitter, Inc., "Twitter API – Get trends near a location," https://developer.twitter.com/en/docs/trends/trends-for-location/api-reference/get-trends-place.html, last Access: May 26, 2018.

[35] OpenWeather, "OpenWeatherMap API," https://openweathermap.org/api, last Access: May 26, 2018.

[36] J. Isaac, "react-native-background-task," https://github.com/jamesisaac/react-native-background-task, last Access: May 26, 2018.

[37] Facebook Inc., "Headless JS," https://facebook.github.io/react-native/docs/headless-js-android.html, last Access: May 26, 2018.

[38] V. Eriksson, "react-native-background-job," https://github.com/vikeri/react-native-background-job, last Access: May 26, 2018.