Universidad Zaragoza
1542

Trabajo Fin de Grado

Desarrollo y Evaluación de un Sistema de Recomendación Basado en una Aproximación Push

Development and Evaluation of a Push-Based Recommendation System

Autora

Irene Fumanal Lacoma

Director

Sergio Ilarri Artigas

ESCUELA DE INGENIERÍA Y ARQUITECTURA

2021

AGRADECIMIENTOS

En primer lugar, me gustaría agradecer la labor de Sergio Ilarri, director del proyecto, por su implicación en él, la ayuda proporcionada en todo momento y las indicaciones que han hecho posibles la finalización de este proyecto. También destacar el interés mostrado y la propuesta de presentar un artículo al *MoMM 2021*.

Agradecer a mi madre, por su apoyo en los momentos más difíciles y estresantes vividos durante la elaboración del trabajo y el grado. A mi padre, por su constante motivación para que no me hundiera incluso cuando veía todo demasiado complicado. Por supuesto, a mi hermana, por ser ese apoyo en el que desahogarme y buscar ánimos cuando más agobiada me sentía. Gracias también al resto de mi familia que ha estado apoyándome a lo largo del trabajo.

Gracias a mis amigos y amigas de la universidad, especialmente a los compañeros y compañeras del grado. Hemos compartido muchas experiencias, buenas y malas, a lo largo de estos años. Sin vuestro apoyo todo habría sido mil veces más difícil.

Gracias a todos mis amigos y amigas de siempre. Vuestra presencia, vuestro apoyo y vuestros ánimos han sido clave sobre todo en los momentos más duros, tanto durante el grado como durante la realización de este TFG.

Finalmente, también es relevante agradecer el apoyo a esta investigación por parte del Gobierno de Aragón - proyecto de cooperación transfronteriza Nueva Aquitania-Aragón PASEO 2.0 (AQ-8), del proyecto PID2020-113037RBI00 / AEI / 10.13039/501100011033 (proyecto NEAT-AMBIENCE), y del Departamento de Ciencia, Universidad y Sociedad del Conocimiento del Gobierno de Aragón (Gobierno de Aragón: referencia de grupo T64_20R, grupo COSMOS).

Desarrollo y Evaluación de un Sistema de Recomendación Basado en una Aproximación Push

RESUMEN

El exceso de información disponible en la actualidad puede desbordar a los usuarios cuando tienen que tomar una decisión y escoger entre diversas opciones. Los sistemas de recomendación, (*RS Recommender Systems*) tienen como objetivo sugerir ítems a los usuarios según sus preferencias y circunstancias, y resultan de gran interés en el campo de la investigación y en el mercado. Estos abordan una perspectiva bidimensional basada en *Usuario-Ítem*. Existen también los sistemas de recomendación dependientes del contexto (*CARS, Context-Aware Recommender Systems*), cuya perspectiva es tridimensional *Usuario-Ítem-Contexto*, en la que incorporan el contexto del usuario en el proceso de recomendación para adaptar mejor las recomendaciones proporcionadas.

El sistema desarrollado consiste en un sistema de recomendación dependiente del contexto y proactivo (que recomienda ítems sin que el usuario intervenga) para usuarios móviles. Este proyecto se ha realizado sobre el prototipo desarrollado en el Trabajo de Fin de Grado *Desarrollo de un Prototipo de Aplicación Móvil para Sistemas de Recomendación Proactivos*, que trabaja sobre un modelo de arquitectura de un sistema de recomendación proactivo y que incluye un prototipo móvil capaz de recibir recomendaciones, gestionar las actividades recomendadas de diversas categorías y comunicarse con el encargado de realizar recomendaciones a los usuarios (el gestor de entorno). El foco de este proyecto previo no era la parte de recomendación.

Este proyecto amplía el prototipo móvil anterior haciéndolo capaz de detectar qué tipos de recomendación deben activarse sin la intervención del usuario. Esto permite tomar este tipo de decisiones en el propio dispositivo y beneficiar la privacidad del usuario, ya que, si este lo desea, puede decidir no compartir información del contexto con el recomendador externo. Además, ofrece la posibilidad de definir reglas personalizadas para decidir qué tipos de recomendación deben activarse según su contexto. Para ello, se ha realizado una búsqueda de tecnologías que permitan la definición de estas reglas y puedan funcionar en un dispositivo Android. Esto, junto con la posibilidad de establecer prioridades entre los distintos tipos de recomendación, hace que el sistema sea capaz de ofrecer una experiencia más personalizada al usuario. Para evaluar el prototipo se ha implementado un *gestor de entorno* de prueba, se ha definido un escenario real y se han comprobado sus resultados. Además, se completaron pruebas de rendimiento de la tecnología de reglas en un dispositivo móvil.

Índice

Capítulo 1

Introducción

En la Sección 1.1 se introducen los sistemas de recomendación y la motivación del proyecto. En la Sección 1.2 se mencionan los objetivos y el alcance del proyecto. Por último, en la Sección 1.3 la organización del proyecto y la estructura de la memoria.

1.1. Sistemas de recomendación y motivación del proyecto

El objetivo de los sistemas de recomendación es proporcionar ítems o actividades a los usuarios según sus intereses y gustos. Los sistemas de recomendación (RS) sugieren ítems teniendo en cuenta el ítem y el usuario. Los ***CARS*** *(Context-Aware Recommender Systems)* son un tipo de RS que extienden el modelo clásico de usuario-ítem, añadiendo el contexto al proceso de recomendación, y así intentar sugerir actividades más apropiadas. El contexto de un usuario es algo que cambia constantemente. Por este motivo, un sistema de recomendación proactivo, es decir, que sea capaz de responder sin que sea necesario que el usuario lo solicite, es preferible frente a uno reactivo, donde el usuario debe de hacer una petición explícita para activar el proceso de recomendación.
En los sistemas de recomendación proactivos es necesario un módulo capaz de detectar cuándo es necesario activar el proceso de recomendación a partir del contexto y de qué tipo de ítem (por ejemplo, si es hora de comer y el usuario no está en casa puede resultar interesante recomendar restaurantes). En este proyecto hemos implementado este módulo mediante una tecnología que permita definir reglas y tomar este tipo de decisiones en el dispositivo móvil. Además, se mejora la privacidad del usuario porque ya no es necesario compartir la información del contexto con el gestor de entorno. El sistema diseñado se evalúa mediante un escenario real y unos datos de prueba.

1.2. Objetivos y alcance del proyecto

El objetivo principal de este proyecto es implementar una arquitectura que permita proporcionar recomendaciones a los usuarios sin que este las solicite previamente. Para

esto se ha desarrollado un módulo capaz de decidir cuándo hay que activar el proceso de recomendación de nuevos ítems al usuario y de qué tipo, dependiendo de su **contexto**. El proyecto se ha realizado sobre una arquitectura implementada previamente en [1]. Esta implementación contaba con una aplicación móvil y el diseño de un gestor de entorno que se encargaba de proporcionar recomendaciones a la aplicación, sin ser esto último el foco del trabajo. Una de las dificultades del proyecto ha sido incorporar el módulo mencionado al proyecto ya existente. Además, se añade la posibilidad de que el usuario **personalice** sus propias reglas para que dependiendo de su contexto se recomienden actividades de un tipo o de otro. En el planteamiento del proyecto se busca preservar la **privacidad** del usuario intentando que las decisiones de activación de una recomendación se realicen en el dispositivo móvil. Finalmente se evalúa la propuesta implementada en un escenario relacionado con el turismo, donde las recomendaciones de ítems/actividades pueden resultar muy útiles para los usuarios.

1.3. Organización del proyecto y contenido de la memoria

El desarrollo del proyecto se ha dividido en varias fases. La primera fue la lectura y documentación del estado del arte, de la arquitectura propuesta y del prototipo ya existente. Así como el análisis de la implementación previa. En segundo lugar, la búsqueda y elección de tecnología para crear el módulo de detección de reglas capaz de funcionar sobre un dispositivo móvil. En tercer lugar, la integración con el prototipo previo implementado de aplicación móvil. La cuarta fase fue el diseño e implementación de las nuevas pantallas relacionadas con las reglas en la aplicación. La quinta fue la traducción de las reglas definidas por los usuarios mediante las interfaces de la aplicación a las reglas con la sintaxis de la tecnología. Finalmente, se desarrolló la parte de pruebas, en la que se utilizaron recomendadores y datos de prueba para probar la arquitectura implementada en un escenario ficticio concreto.
Durante el desarrollo de cada una de las partes anteriores se han ido realizando pruebas y revisiones constantes con el objetivo de solventar posibles errores conforme se completaba parte del trabajo.

La memoria está estructurada en las siguientes partes: en el Capítulo 2 se hace una introducción a los sistemas de recomendación, la arquitectura y prototipo previos y el tipo de tecnología que se va a utilizar (*procesamiento de eventos complejos*). En el Capítulo 3 se presenta el diseño y planteamiento de la arquitectura desarrollada, junto a la búsqueda y selección de tecnología para el módulo de detección de reglas. El

Capítulo 4 presenta la implementación de las nuevas funcionalidades de la aplicación móvil y la integración de la tecnología a la aplicación móvil y la traducción de las reglas a su sintaxis. El Capítulo 5 explica el gestor de entorno de prueba utilizado como recomendador. El Capítulo 6 incluye el planteamiento y ejecución de las pruebas y sus resultados. Finalmente, en el Capítulo 7 se explican las conclusiones y el trabajo futuro. Además, hay varios anexos que complementan la memoria principal. En el Anexo A se incluyen los requisitos de la aplicación. En el Anexo B se presentan el mapa de navegación, las pantallas implementadas y el diagrama de paquetes de la aplicación. El Anexo C incluye ejemplos de reglas definidas con Siddhi. El Anexo D la encuesta realizada durante la elaboración del trabajo. El Anexo E incluye manuales de instalación y de uso de la aplicación. En el Anexo F se presentan fragmentos de código importantes. En el Anexo G se explica el diseño del gestor de entorno. Finalmente, en el Capítulo I se adjunta el artículo presentado en el MoMM 2021.

Capítulo 2

Conceptos básicos y trabajo previo

En la Sección 2.1 explica los fundamentos de la arquitectura base del trabajo. La Sección 2.2 presenta los aspectos clave del prototipo previo. Por último, la Sección 2.3 introduce el tipo de tecnología utilizada para detectar cuando activar una recomendación.

2.1. Sistema de recomendación basado en una aproximación push

En la Sección 1.1 se ha hecho una introducción a los sistemas de recomendación tradicionales y los dependientes en el contexto. En el artículo [2] se propone una arquitectura para un sistema que ofrece recomendaciones al usuario de manera proactiva. Por lo tanto, el sistema sigue una aproximación *push*, en la que no es necesaria una solicitud explícita por parte del usuario para que este reciba recomendaciones. Este artículo es uno de los puntos de partida del trabajo.
La idea principal del modelo que sigue la arquitectura es captar la dinámica del contexto del usuario para realizar recomendaciones más precisas [2]. Está formada por los siguientes elementos:

- El **contexto** nos permite delimitar el alcance y el propósito de la recomendación.
- Un **entorno** es un área común donde los usuarios coexisten para realizar un conjunto de actividades bajo una serie de restricciones que nos ayudan a delimitar el entorno y que pueden ser físicas o virtuales (la geolocalización o el tiempo son algunos ejemplos de restricciones). Además, un usuario puede estar activo en varios entornos a la vez.
- El **gestor de entorno** es un agente asociado a un entorno concreto y que gestiona los usuarios que pertenecen a éste y la comunicación.
- Los **usuarios** son los agentes que están interesados en las recomendaciones en un entorno.
- Las **actividades** equivalen a los *items*(por ejemplo, productos o lugares) cuando se habla de sistemas de recomendación, pudiendo incorporar también acciones

como actividades (por ejemplo, visitar un museo).

La arquitectura está diseñada para entornos móviles donde el contexto del usuario cambiará constantemente. Para seguir un comportamiento proactivo, se proponen las siguientes etapas en el proceso de recomendación:

- **Fase de activación del proceso de recomendación.** En esta etapa se decide si el proceso de recomendación debe empezar o no.
- **Fase de pre-filtrado** para eliminar las actividades o tipos de actividades que se quedan fuera del contexto del usuario.
- **Algoritmo de recomendación** que seleccionará las actividades según el contexto del usuario.
- **Fase de post-filtrado** donde se eliminarán las coincidencias entre las recomendaciones obtenidas por diferentes entornos y se tendrán en cuenta aspectos relacionados con las preferencias privadas del usuario, que no ha querido compartir con el gestor externo.
- **Presentación** de las recomendaciones al usuario.

Algunas de las etapas anteriores serán ejecutadas en el dispositivo móvil del usuario y otras en un gestor entorno. La Figura 1 se pueden observar estas etapas.

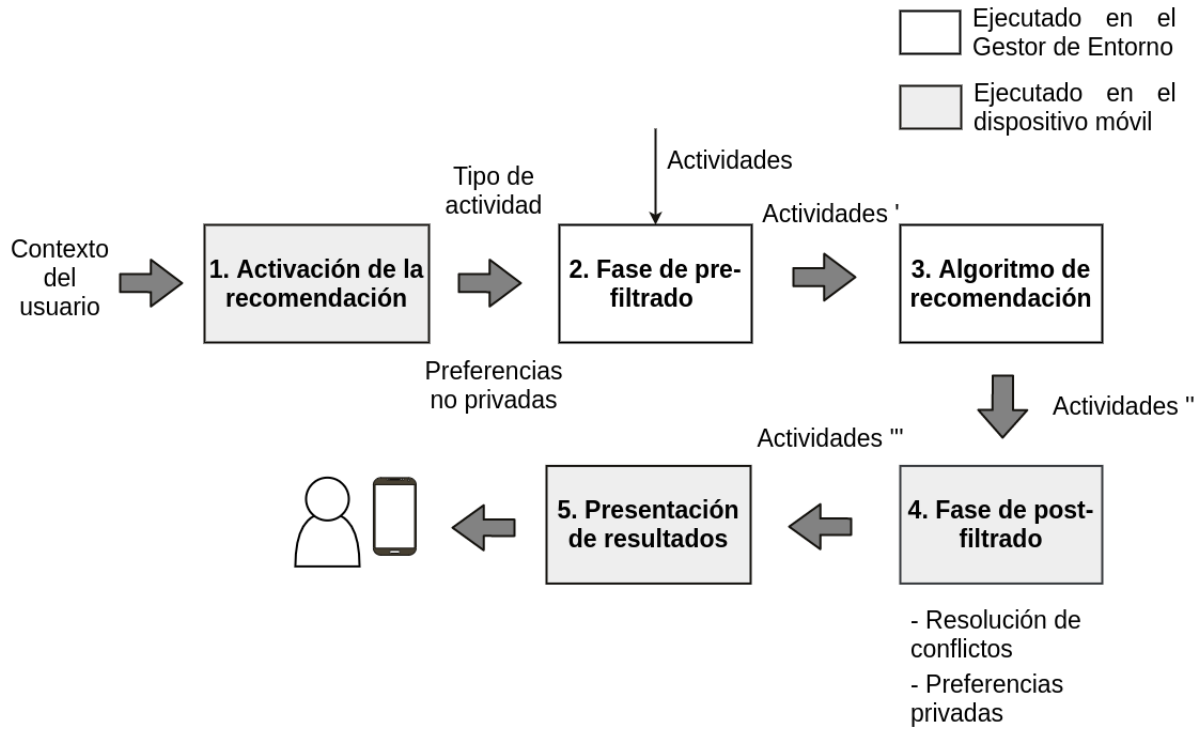


Figura 1: Fases del proceso de recomendación. Adaptada de [2]

2.2. Arquitectura previa y prototipo existente

En el Trabajo de Final de Grado [1] se comenzó la implementación de la arquitectura mencionada anteriormente. En el se desarrolló un prototipo móvil capaz de gestionar las actividades recomendadas y la comunicación con el gestor de entorno, sin centrarse en el proceso de recomendación.

En primer lugar, se desarrolló una aplicación móvil con ***React Native*** (`https://reactnative.dev/`) que capturaba y enviaba periódicamente el contexto del usuario al gestor de entorno y mostraba las actividades obtenidas como resultado. Además, el usuario podía realizar dos configuraciones clave. La primera, relacionada con la privacidad, permitía seleccionar aquella información del contexto que no quería compartir con el gestor. La segunda, consistía en elegir los tipos de actividades que eran de su interés para que sólo estas fueran mostradas al usuario.

En segundo lugar se implementó un gestor de entorno de prueba para realizar las recomendaciones. Se definió una API para el gestor de entorno y se diseñó una base de datos con la información de las actividades y toda la necesaria para abordar el proceso de recomendación.

El sistema también permitía que los usuarios guardaran, descartaran o puntuaran las actividades, información que resultaría útil en la tarea de recomendación.

La arquitectura mencionada seguía las siguientes fases (puede verse en la Figura 1):

1. Se captura el nuevo contexto del usuario a partir de los sensores del dispositivo móvil (por ejemplo, el GPS) o de peticiones a servicios externos (por ejemplo: para obtener el tiempo atmosférico). La información que se captura en el prototipo es la localización, los eventos del calendario, el tiempo atmosférico e información básica del usuario.
2. Se envía al gestor de entorno aquella información del contexto que el usuario ha permitido que sea compartida, respetando así su privacidad.
3. El gestor de entorno realiza el proceso de recomendación, obtiene las actividades que se ajusten al contexto del usuario y se envían al dispositivo móvil.
4. Una vez el dispositivo móvil ha recibido las recomendaciones enviadas por el gestor de entorno, se descartan aquellas actividades cuyo tipo no resulta de interés al usuario según la configuración que éste ha establecido en la aplicación.
5. Se muestran los resultados en la aplicación móvil.

El prototipo móvil se desarrolló con *React Native* pero sólo funciona en Android

debido a la necesidad de incorporar código nativo para la ejecución de tareas en segundo plano (para la comunicación con el gestor de entorno) utilizando la librería *Headless JS* (`https://reactnative.dev/docs/headless-js-android`), que solo ofrece una guía para Android. El gestor de entorno de prueba se desarrolló con *Spring Boot* y para el proceso de recomendación se utilizó *Apache Spark*. El proyecto del gestor de entorno no ha sido localizado y por lo tanto solo se disponía de la API definida y el diseño de la base de datos.

2.3. Procesamiento de eventos complejos (CEP)

El contexto del usuario cambia de manera dinámica y constante y por lo tanto es fundamental saber cuándo debe activarse el proceso de recomendación y cuándo no. Los dispositivos móviles incluyen diferentes sensores que capturan datos sobre el contexto del usuario y además permiten obtener información haciendo peticiones a servicios externos. Toda esta información puede ser útil para tomar esa decisión mediante la definición de una serie de reglas.

En el artículo [3] se presentan varios casos de uso y algunos ejemplos de reglas que pueden utilizarse para decidir qué tipos de recomendaciones deben activarse. Para ello proponen una implementación utilizando reglas *SWRL* (*Semantic Web Rule Language*) definidas sobre la ontología *SNN* (*Semantic Sensor Network*). Durante el desarrollo del trabajo se ha utilizado tecnología de Procesamiento de Eventos Complejos (*Complex Event Processing* o *CEP*) para la definición e implementación de esas reglas. Esta tecnología nos permite analizar y procesar información que llega en forma de eventos y encontrar aquellos que son significativos (oportunidades o amenazas) en tiempo real para responder de la manera más rápida posible. Para identificar aquellos eventos que son relevantes se definen una serie de reglas. CEP permite registrar los eventos y encontrar patrones entre ellos. Además, permite manejar restricciones basadas en el tiempo, algo muy importante si queremos detectar la *no ocurrencia de eventos*. En la Sección 3.3 se mencionan distintos ejemplos de tecnologías cuya base es CEP y que han sido consideradas para el desarrollo del proyecto.

Capítulo 3

Diseño y planteamiento de la arquitectura

En este capítulo presentamos la arquitectura que hemos implementado. La Sección 3.1 incluye la arquitectura propuesta, en la Sección 3.2 el análisis de tecnologías CEP y en la Sección 3.3 la prueba de las tecnologías CEP capaces de funcionar en Android.

3.1. Arquitectura propuesta

A continuación presentamos la arquitectura propuesta en alto nivel para el desarrollo del sistema de recomendación que proporciona recomendaciones proactivas a usuarios con dispositivos móviles. El objetivo de las novedades introducidas es la implementación de la **fase de activación del proceso de recomendación**, para que el dispositivo móvil sea capaz de decidir qué tipo de recomendaciones debe activar dependiendo del contexto del usuario y de las reglas definidas por este, y comunicarle al *gestor de entorno* su decisión. Los tres aspectos clave de nuestra nueva propuesta son los siguientes:

1. La fase de activación del proceso de recomendación se ejecuta en el dispositivo móvil. Las decisiones tomadas se comunicarán posteriormente al gestor de entorno. Para poder realizar esto, se necesita una tecnología basada en CEP capaz de ejecutarse en un dispositivo móvil.
2. Garantizamos la privacidad del usuario, ya que la decisión sobre el tipo de recomendación que debe activarse se realiza en el dispositivo móvil y no es necesario enviar datos sobre el contexto del usuario al gestor de entorno si el usuario no lo desea.
3. Permitimos al usuario que participe y personalice la activación de la recomendación según sus intereses. Para ello, el usuario puede definir dos tipos de reglas:
   - ***Context Rules.*** Permiten la definición de situaciones que pueden ser relevantes para el usuario. En nuestra implementación hemos incorporado

cuatro tipos: basadas en la hora, basadas en el calendario, basadas en la localización y basadas en el tiempo atmosférico. Por ejemplo, el usuario puede definir una regla basada en la localización con nombre *AtHome* y las coordenadas geográficas de su casa para determinar si el usuario está en su casa o no. Otro ejemplo es la basada en la hora llamada *Time4Lunch*, en la que el usuario indica la hora a la que suele comer(por ejemplo, de 13:00 a 15:00).

- ***Recommendation Triggering Rules.*** Están formadas por dos o más *context rules* y activan un tipo concreto de recomendación. Por ejemplo, si tenemos las context rules *AtHome* y *LunchTime*, podemos definir una regla que indique que si NO estamos en casa y es hora de comer active una recomendación de restaurantes.

Además, el usuario puede definir ***Exclusion Sets*** para indicar que no quiere recibir recomendaciones de dos o más tipos de recomendación a la vez y que quiere dar más prioridad a una que a otras. Un ejemplo sería un *exclusion set* llamado *FirstEatThenWatch* en el que se dé más prioridad a las recomendaciones de restaurantes que a las de cines en caso de que las reglas definidas activaran ambos tipos.

Los dos tipos de reglas definidas anteriormente deben gestionarse desde el motor CEP que funciona en el dispositivo móvil. Por lo tanto, el proceso ilustrado en la Figura 2 y que se realiza para obtener ítems recomendados es el siguiente:

1. El dispositivo móvil captura el contexto del usuario y se lo envía al motor CEP.
2. El motor procesa el contexto del usuario y decide qué tipos de recomendación deben activarse según las *context rules* y *triggering rules* definidas.
3. Se comprueban los *exclusion sets* y se gestionan los tipos de recomendación obtenidos por el motor según los intereses del usuario.
4. Se envía una petición al *gestor de entorno* para que active el proceso de recomendación de actividades de los tipos obtenidos.
5. El gestor de entorno completa el proceso de recomendación y devuelve los resultados al dispositivo móvil.
6. El dispositivo móvil filtra las actividades que ha obtenido del gestor de entorno quedándose exclusivamente con aquellas que resultan de interés para el usuario en ese momento y que el usuario puede configurar desde los ajustes de la aplicación.

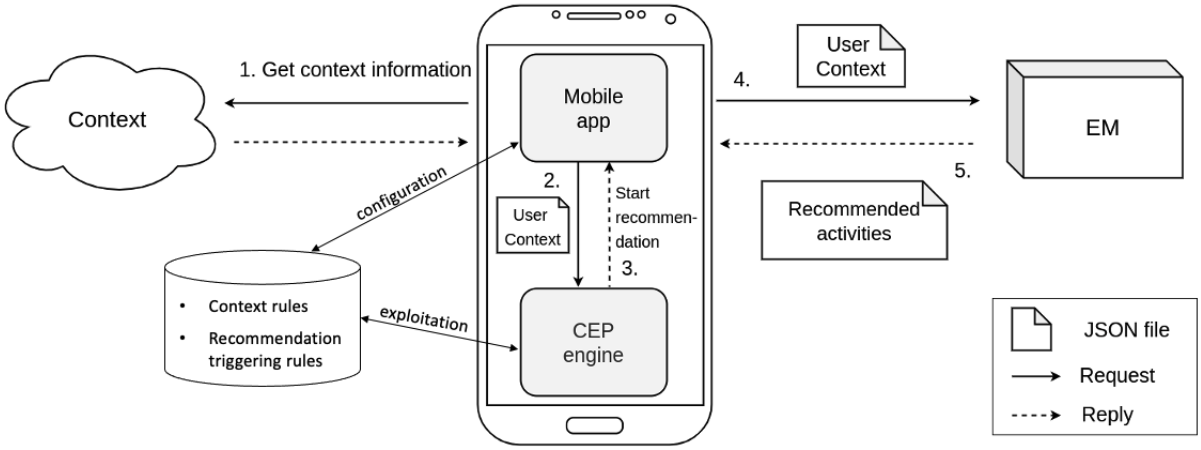


Figura 2: Arquitectura en alto nivel centrada en la activación de las recomendaciones

En la Tabla 1 se reflejan las diferencias entre el prototipo previo y el prototipo final desarrollado.

Característica	**Prototipo previo**	**Prototipo actual**
Captura del conexto	✓	✓
Gestión de las actividades obtenidas como recomendación	✓	✓
Valoración de las actividades	✓	✓
Seleccionar qué tipos de recomendación resultan de interés	✓	✓
Seleccionar qué información se comparte con el gestor de entorno	✓	✓
Solicitar el inicio del proceso de recomendación sólo cuando es necesario	✗	✓
Detección de cuándo es necesario activar un tipo de recomendación	✗	✓
Definir reglas que personalizadas para decidir cuando hacer la activación	✗	✓
Establecer prioridades entre los tipos de recomendaciones que deben activarse	✗	✓
Estudio de los tipos de recomendación que pueden ser de interés para los usuarios mediante la realización de una encuesta	✗	✓
Evaluación de toda la arquitectura con un escenario de prueba	✗	✓

Tabla 1: Comparación de las características del proyecto previo y el actual

El aspecto clave de la propuesta anterior es la necesidad de una tecnología CEP que pueda ejecutarse en un dispositivo móvil. Debido a esta restricción, se propuso una arquitectura alternativa en la que la toma de decisiones sobre qué tipos de

recomendaciones debían activarse tendrían que ejecutarse en el gestor de entorno. En el dispositivo móvil podría definirse un módulo inteligente cuyo objetivo sería definir reglas muy sencillas para tomar decisiones más pequeñas. Esta aproximación introduce varios problemas relacionados con el incremento del número de mensajes que deberían transmitirse por la red, necesitando un protocolo de comunicación más complejo, y agotarían la batería del dispositivo de manera más rápida debido a un mayor consumo de energía.

3.2. Análisis y selección de tecnología CEP

En este apartado presentamos las tecnologías CEP que se consideraron durante la búsqueda. La búsqueda y comparación de tecnologías se ha realizado teniendo en cuenta los artículos y trabajos de investigación [4], [5] y [6].

3.2.1. Esper

Esper (`https://www.espertech.com/esper/`) es una de las tecnologías CEP más conocidas, de código abierto y disponible para Java y .NET. Está bien documentada y aunque apareció en 2006, nuevas versiones siguen siendo lazandas. Su lenguaje de definición de reglas es similar a SQL, llamado Event Processing Language (EPL), lo que facilita la curva de aprendizaje. En 2013 fue adaptada para que pudiera utilizarse en Android, mediante el proyecto Asper (https://github.com/mobile-event-processing/Asper), solucionando algunos conflictos debidos a dependencias y paquetes Java no disponibles en Android. Este proyecto no ha sido actualizado desde entonces, siendo un gran inconveniente, ya que está basado en una versión muy antigua de Esper, la 4.8.0.

3.2.2. Siddhi

Siddhi (`https://siddhi.io/`) es otra de las tecnologías CEP más conocidas y de código abierto. El proyecto está bien documentado y para definir las reglas cuenta con su propio lenguaje SiddhiQL, similar también a SQL. Puede utilizarse como librería de Java o Python o como un microservicio en Docker o Kubernetes. Hay disponible una herramienta de escritorio que permite desarrollar aplicaciones Siddhi y probarlas de manera sencilla, Streaming Integrator Tooling desarrollado por WSO2. Siddhi puede ejecutarse en un dispositivo Android. Por ejemplo, en el artículo [7] se utiliza Siddhi para monitorizar pacientes en el ámbito de la salud y donde el motor CEP Siddhi funciona en el dispositivo móvil.

3.2.3. Apache Flink

Flink (`https://flink.apache.org/`) es una tecnología de procesamiento de flujos de eventos. Tiene una biblioteca especial para procesamiento de eventos complejos llamada FlinkCEP. Se centra en el procesamiento de datos distribuido y no ha sido diseñado para ejecutarse en dispositivos móviles. No hemos encontrado ningún caso en el que se intente utilizar Apache Flink en Android o se intente portar a la plataforma.

3.2.4. Drools

Drools (`https://www.drools.org/`) es otra alternativa CEP basada en Java. Tiene su propio lenguaje para definir las reglas, llamado DRL (Drools Rule Language), en ficheros .drl de texto. En [4] se indica que Drools es peor que Siddhi o Esper en cuanto a la utilización de recursos. No es posible ejecutar Drools directamente en Android. Sin embargo, en [8], se menciona la utilización de una versión de Drools adaptada a Android. Esto parece ser una solución ad-hoc porque no se ha encontrado la publicación de esa versión.

3.2.5. Microsoft StreamInsight

Microsoft StreamInsight (`https://docs.microsoft.com/en-us/archive/msdn-magazine/2012/march/microsoft-streaminsight-building-the-internet-of-things`) es una tecnología CEP comercial. Esto ha sido uno de los motivos para descartar la tecnología, ya que no es de código abierto y se necesita una licencia de SQL Server para utilizarlo. Las reglas deben definirse como consultas utlizando .LINQ (.NET Language-Integrated Query). No se ha encontrado ningún ejemplo de su utilización en Android.

3.3. Prueba de tecnologías CEP en dispositivos móviles y selección final

Nuestra principal necesidad es una tecnología CEP capaz de funcionar en un dispositivo móvil, requisito que solo lo satisfacen Asper y Siddhi. Para escoger entre Siddhi y Asper, desarrollamos una pequeña aplicación en Android con tres botones: uno para iniciar el motor CEP, otro para pararlo y otro para enviar un evento y que sea procesado por el motor. Los resultados de las pruebas fueron los siguientes:

- Asper funcionó correctamente en la aplicación mencionada. Para ello se añadió la librería al proyecto de Android Studio. Decidimos descartar esta opción

porque Asper utilizaba una versión muy antigua de Esper. Para solucionar este inconveniente, se intentaron utilizar directamente versiones más recientes de Esper (Esper 7.1 y 8.7) para ver si eran compatibles con Android. No obtuvimos éxito en la prueba ya que detectamos problemas con dependencias, clases necesarias que no estaban disponibles en Android.

- Siddhi también funcionó correctamente. Se añadieron todas las dependencias Siddhi al proyecto Android (disponibles como bibliotecas de Java). Se realizaron pruebas con las versiones 4 y la 5.1. Existía una extensión, *Siddhi Script JS*, que permitía definir algunas cosas en Javascript, algo que resultaría útil para cálculos como la comparación de coordenadas geográficas. No era compatible con Android y por lo tanto hubo que buscar una alternativa, que se explica en el Anexo C.2.

Después de estos intentos y las razones presentadas anteriormente, se decidió optar por Siddhi como la tecnología CEP que utilizaríamos en el proyecto.

Capítulo 4

Implementación de la aplicación móvil

En este capítulo abordamos las funcionalidades y las decisiones tomadas para el desarrollo de la aplicación móvil. En la Sección 4.1 presentamos los nuevos requisitos de la aplicación. En la Sección 4.2 mencionamos el diseño y la implementación de la interfaz. En la Sección 4.3 presentamos la incorporación de la tecnología en el proyecto, el envío del contexto del usuario al motor CEP, la recepción del resultado. En la Sección 4.4 explicamos la implementación de las reglas definidas por el usuario en la interfaz y su traducción a la sintaxis de Siddhi.

4.1. Análisis de requisitos de la aplicación

La aplicación desarrollada es una ampliación de la desarrollada en el Trabajo de Final de Grado [1] y que incluía el envío del contexto y la comunicación con el gestor de entorno, y la gestión de las actividades recomendadas por este (mostrar su información, descartar, marcar como favorita o puntuar del 1 al 5). Presentaba una pantalla de ajustes donde el usuario podía seleccionar sus intereses entre las diferentes categorías y otra de perfil donde el usuario podía decidir qué información quería compartir con el gestor de entorno y cuál no, respetando así su **privacidad**.

La aplicación debía incorporar las siguientes nuevas funcionalidades:

- Creación, edición, borrado y listado de las *context rules*. Habrá cuatro tipos posibles de este tipo de reglas que podrán gestionarse: basadas en la hora, en el calendario, en la localización y en el tiempo atmosférico.
- Creación, edición, borrado y listado de las *recommendation triggering rules*. Cada triggering rule debe estar formada por dos o más context rules ya existentes y activará un tipo de recomendación.
- Creación, edición, borrado y listado de los *exclusion sets*.
- Ordenación de los *exclusion sets* según la prioridad que le interese al usuario (por ejemplo: si el usuario define un exclusion set donde las recomendaciones de

restaurantes tienen más prioridad que las de cines, y en otro exclusion set define lo contrario, podrá decidir cuál de los dos tiene prioridad estableciendo un orden que podrá cambiar cuando desee).

– La aplicación tendrá integrado el motor CEP con el que se comunicará para enviar el contexto de manera periódica y obtener el resultado. Además de la comunicación con el gestor de entorno cuando sea necesario. Esta parte de la aplicación carece de interfaz gráfica.

La aplicación original se desarrolló con React Native con el objetivo de obtener una aplicación compatible con Android e iOS. Finalmente la aplicación solo estaba disponible para Android porque fue necesario utilizar la librería Headless.js para poder ejecutar operaciones en segundo plano (comunicación periódica con el EM) y no era compatible con iOS. La aplicación desarrollada por lo tanto mantiene este comportamiento y actualmente está destinada a usuarios con dispositivos Android.

En el Anexo A se incluye el listado de requisitos funcionales y no funcionales de la aplicación.

4.2. Diseño e implementación de la interfaz

Como el proyecto a desarrollar es la continuación de uno existente, se seguirá el diseño de la interfaz, añadiendo exclusivamente las nuevas vistas necesarias para implementar las nuevas funcionalidades. En el Anexo B.1 presentamos el mapa de navegación de las pantallas donde las pantallas grises son las nuevas incorporadas a la aplicación y en el Anexo B.2 las pantallas finales implementadas. Estas nuevas vistas son:

– **Creación de *context rule***. Hay cuatro vistas similares de este estilo que corresponden a los cuatro tipos de context rule que hay. Su implementación puede verse en las Figuras B.6a, B.7a, B.8a y B.9a.

– **Vista de edición y vista con la información de la *context rule*.** Estas dos son muy similares. Las Figuras B.6b, B.7b, B.8b y B.9b son las pantallas de visualización y las Figuras B.6c, B.7c, B.8c y B.9c las de edición.

– **Listado de *context rules*.** Desde ella se puede acceder a las anteriores y se puede eliminar las reglas. Puede consultarse en la Figura B.5c.

– **Creación de *triggering rule***. Se pueden seleccionar dos o más *context rules* y el tipo de recomendación que se va a activar. Las Figuras B.10a, B.10b y B.10c muestran ejemplos de la pantalla implementada.

- **Vista de edición y vista con la información de la *triggering rule*.** Las Figuras B.11a y B.11b representan las pantallas de visualización y de edición respectivamente.
- **Listado de *triggering rules*.** Desde ella se puede acceder a las anteriores, se pueden eliminar las reglas y desactivar las que se desee. Puede consultarse en la Figura B.11c.
- **Creación de *exclusión set***. Se pueden seleccionar dos o más tipos de recomendación y establecer un orden de prioridad entre ellos. La Figura B.4c representa la pantalla implementada.
- **Vista de edición y vista con la información del *exclusion set*.** Las Figuras B.5a y B.5b representan las pantallas de visualización y edición respectivamente.
- **Listado de *exclusion sets***. Desde ella se puede acceder a las anteriores, se pueden eliminar y ordenar los *exclusion sets* según el usuario desee. Puede verse en la Figura B.4b.

Para definir los tipos de reglas que podían ser útiles para los usuarios y los tipos de recomendación que estos esperan en la aplicación elaboramos una encuesta cuya información puede consultarse en el Anexo D. Se decidió que los tipos de recomendación que podían activarse eran: restaurantes, alojamientos, lugares de interés, museos, edificios de espectáculos (auditorios,cines), actividades de ocio y lugares de entretenimiento (bares, discotecas, pubs) y tiendas.

La aplicación previa ya tenía implementadas las pantallas de listado de actividades (Figuras B.2a, B.2b y B.2c) y la pantalla donde podía consultarse la información de una actividad (Figura B.3a). Además, también tenía la pantalla de menú, que ha sido modificada para añadir las opciones de navegación a las pantallas de las reglas (Figura B.4a) y las pantallas de perfil del usuario y ajustes (Figuras B.4b y B.4c).

La aplicación inicial aprovechaba la característica de *React* para crear interfaces a partir de componentes y así permitir y facilitar la reutilización de componentes y código durante el desarrollo de las pantallas. Se ha tratado de mantener este método de desarrollo en la creación de las nuevas pantallas añadidas. Para el desarrollo ha sido necesario incorporar alguna biblioteca como por ejemplo *react-native-modal-datetime-picker* para poder utilizar selectores de hora y fechas en el calendario. Con respecto a las bibliotecas presentes en el prototipo previo, destacar:

- **Realm Database**. Para la base de datos del dispositivo móvil. Se trata de una

base de datos NoSQL que permite definir los esquemas de diferentes objetos. Se ha ampliado el modelo de datos original, incorporando los esquemas de las *context rules*, *triggering rules* y *exclusion sets* y realizando pequeñas modificaciones para adaptarlo a las nuevas funcionalidades.

– **Facebook SDK**. Se utilizaba exclusivamente para la autenticación del usuario en el sistema, ya que permitía identificar al usuario de manera anónima y única. En nuestro proyecto el inicio de sesión de Facebook se mostraba en un navegador de Android. En agosto de 2021 decidieron retirar este método de inicio de sesión y añadir otros más seguros (con notificación push o completar el inicio de sesión en un navegador Chrome) y para ello era necesario actualizar a la versión 8.2 del SDK, no compatible con la versión de React Native de nuestro proyecto. Por ahora ha sido desactivado el inicio de sesión y queda como trabajo futuro solucionar el problema.

Todas las pantallas implementadas pueden verse en el Anexo B.2. Además, en el Anexo E pueden consultarse manuales de instalación y de usuario para poder probar el prototipo.

4.3. Integración de Siddhi en el prototipo existente

La aplicación está desarrollada sobre el prototipo previo donde la mayor parte de la lógica de su código estaba elaborada con Javascript. Se decidió trabajar sobre este prototipo, ya que contaba con las pantallas y la lógica necesarias para la gestión y visualización de las actividades recomendadas y la gestión de los intereses del usuario y su perfil y ajustes. Además, ya contaba con una pequeña parte desarrollada con código nativo Android (Java) para el envío periódico del contexto al gestor de entorno. Por este motivo se decidió ampliar ese prototipo y no elaborar un nuevo. Para añadir Siddhi al proyecto hemos incorporado más código nativo para utilizar las bibliotecas Java necesarias que permiten lanzar, detener y gestionar el motor CEP. Las cuatro principales son: *Siddhi Core*, *Siddhi QueryAPI*, *Siddhi Query Compiler* y *Siddhi Annotation* (`https://siddhi.io/en/v5.1/download/#siddhi-libs`).

Una de las mayores dificultades del proyecto fue tener por un lado desarrollada la lógica de la aplicación y las interfaces con Javascript y por otro la gestión del motor CEP con código nativo, ya que de alguna manera era necesaria la comunicación entre ambas partes. El código nativo era necesario porque las dependencias de Siddhi eran bibliotecas Java. A continuación mostraremos cómo se realizó este trabajo.

Siddhi como servicio Android

Para integrar Siddhi en la aplicación se ha incorporado como un **Servicio de Android** (clase *Service*). Un servicio es un componente que permite ejecutar operaciones de larga duración en segundo plano y que no proporciona interfaz al usuario. A continuación mencionamos los tipos de servicios que existen:

- **Iniciados**. Una vez iniciados se ejecutan de forma indefinida.
- **Enlazados**. Permite una interacción cliente/servidor entre el componente que se enlaza y el servicio.
- **Híbridos**. Combina cualidades de los dos.

Siddhi ha sido incorporado como un **servicio híbrido** entre los dos, que se inicia de manera indefinida y a la vez permite una interacción cliente/servidor con los componentes. Para resolver la dificultad de comunicar el código JavaScript con el código nativo y poder lanzar, detener, enviar eventos y recoger el resultado obtenido por Siddhi, hemos creado un Android Native Module (`https://reactnative.dev/docs/native-modules-android`), llamado *SiddhiClientModule*, que actúa como un cliente del motor CEP (invoca sus operaciones). Este expone una serie de métodos que son utilizados desde el código JavaScript. Concretamente, para poder comunicar el código JavaScript con Siddhi, hemos implementado tres clases nativas: 1) *SiddhiAppManager* gestiona la comunicación entre la aplicación y el motor Siddhi, 2) *SiddhiService* implementa la lógica del servicio Android y utiliza la clase anterior, y 3) *SiddhiClienteModule* sirve como intermediario entre JavaScript y el servicio Android.

La aplicación móvil recogerá el contexto del usuario periódicamente (en el caso del proyecto, cada 30 segundos) y lo enviará a Siddhi para que decida qué tipos de recomendación deben activarse. Por otro lado, la aplicación tiene que esperar a los resultados obtenidos por el motor CEP. Para implementar ambas tareas se ha utilizado la librería *Headless JS* (`https://reactnative.dev/docs/headless-js-android`), que permite ejecutar tareas mientras la aplicación está en segundo plano.

Envío del contexto

Con un tiempo determinado de actualización del contexto, la aplicación recogerá el contexto y lo enviará a Siddhi. Para implementarlo se ha definido una tarea de envío de contexto (*SendContextTask*) utilizando la librería *Headless JS*. La clase extiende a *HeadlessJsTaskService* y cada vez que se quiera ejecutar la tarea se debe invocar

a *startService*. Para que esta tarea se lance periódicamente hemos implementado un Servicio Android iniciado que inicia la tarea definida cada vez que se cumple el tiempo especificado (por ejemplo: si el tiempo de actualización del contexto es 30 segundos, lanza la tarea cada 30 segundos). En la Figura 3 se pude ver la implementación del envío del contexto.

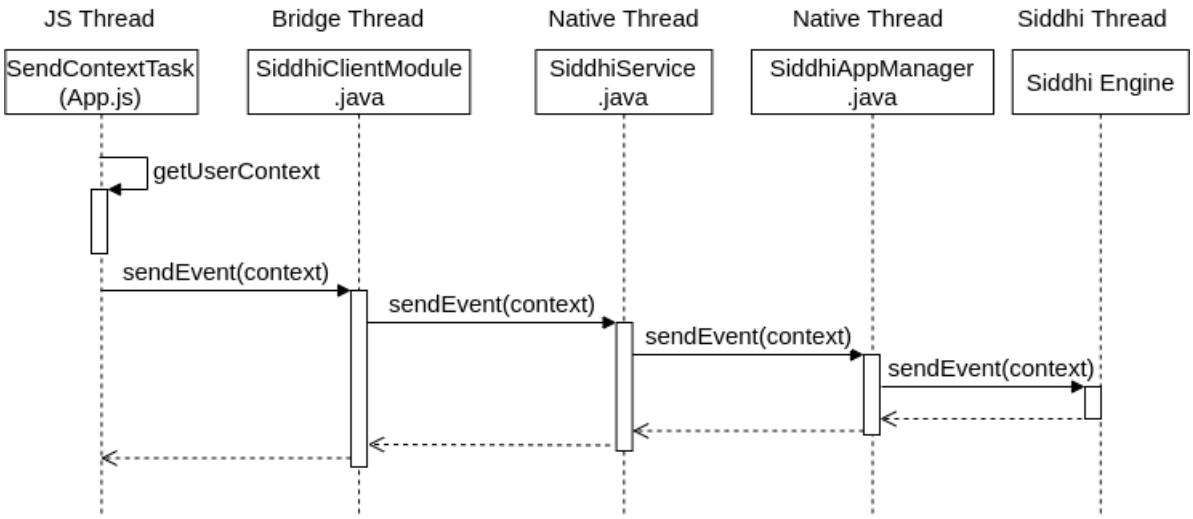


Figura 3: Diagrama de secuencia del envío del contexto desde JavaScript a Siddhi

A continuación mostramos el formato del contexto que enviamos a Siddhi. Destacar que en el vector de *Observations*, habrá tantos elementos cuyo *observedProperty* sea *Location* como *context rules* de ese tipo haya definido el usuario. Esto es porque la comparación de coordenadas geográficas se realiza antes de enviar el contexto, y a Siddhi se le envía la distancia de dicho lugar con la ubicación actual.

```
{    "UserContext": {
         "contextId": "id del contexto",
         "date": "dd/MM/YYYY",
         "time": "hh:mm:ss"
     },
     "Observations": [
         { "observedProperty": "Weather",
           "optionalField": "temperatura",
           "observationValue": "{Clear, Clouds...}" },
         { "observedProperty": "Location",
           "optionalField": "nombre",
           "observationValue": "distanciaCalculada" }
         ]
}
```

Obtención del resultado

Cada regla en Siddhi se define como una *consulta*. Las consultas consumen eventos y almacenan los resultados en lo que llamamos *streams*. La recuperación de los tipos de recomendación obtenidos por Siddhi se realiza registrando un *callback* que recibe los resultados una vez los eventos han sido procesados. Hay dos tipos de *callback*: 1) *Query callback* que se suscribe a una consulta de Siddhi (regla), 2) *Stream callback*, que se suscribe a un stream/evento. En nuestro sistema utilizamos la primera. Sólo se reciben los resultados de aquellos eventos que cumplen las reglas, por lo tanto, solo se obtendrán los tipos de recomendación que se deben activar. En nuestra implementación hemos creado un *stream* llamado *Result* en el que se acumulan los tipos de recomendaciones que deben activarse. Se ha creado una consulta que consume los eventos de tipo *Result* y agrupa los resultados en otro *stream* llamado *FinalResult*; así se notifican todos los tipos de recomendación en el mismo callback con el formato *<contextId, {tipo de recomendación 1, tipo de recomendación 2, ...}>*. Para poder unificar el resultado como hemos mencionado y agrupar los tips de recomendación activados, necesitamos esperar un tiempo determinado, ya que el procesamiento de Siddhi se realiza en tiempo real. Hemos utilizado una ventana temporal de tipo *batch* para recolectar los resultados cada X tiempo determinado (cada 5 segundos se devuelven nuevos resultados al *stream Result*). De esta manera recuperamos desde el código nativo los resultados obtenidos por el motor CEP. Todo lo anterior se encuentra en la clase ***SiddhiAppManager***.
La consulta de Siddhi que recoge todos los tipos de recomendación que deben activarse y que utiliza la ventana temporal podemos verla a continuación.

```
@info(name = 'finalResults')
from Results#window.timeBatch(7 sec)
select contextId, str:groupConcat(recommendation)
    as recommendation group by contextId
insert into FinalResults;
```

La obtención del resultado desde JavaScript se ha realizado definiendo una tarea con *Headless JS* llamada *ListenRecommendationResultTask.* Esta tarea se ejecuta continuamente y realiza las siguientes acciones: 1) comprueba si el usuario ha iniciado sesión y si *Siddhi* ha sido iniciado, 2) espera a que un nuevo resultado sea notificado, 3) obtiene el resultado y comprueba los *exclusion sets*, 4) solicita al gestor de entorno nuevas recomendaciones con los tipos de recomendación obtenidos y el contexto del usuario. Para obtener el resultado utiliza el método de ***SiddhiClientModule***, que utilizará los métodos de ***SiddhiService*** y este los de ***SiddhiAppManager***.

SiddhiClientModule es un módulo nativo y no permite que sus métodos devuelvan resultados. Para poder recuperarlo hemos utilizado un *callback* de JavaScript. La operación de obtener el resultado del módulo nativo lanzará un nuevo hilo de ejecución en el que esperará a que se le notifique un nuevo resultado. Ha sido necesario crear este hilo para no ejecutar tareas largas o de espera en el principal de *SiddhiClientModule*, ya que la aplicación se vería afectada y podría bloquearse. En el nuevo, esperamos a que se nos notifique un nuevo resultado, gracias al mecanismo ***wait-notify*** de los threads en Java. En la Figura 4 podemos observar este proceso, suponiendo que se realiza el *result.notify()* desde el *SiddhiAppManager* cuando un nuevo resultado ha sido recogido. Posteriormente se comprobarán los *exclusion sets* y se comunicará al gestor de entorno las decisiones tomadas.

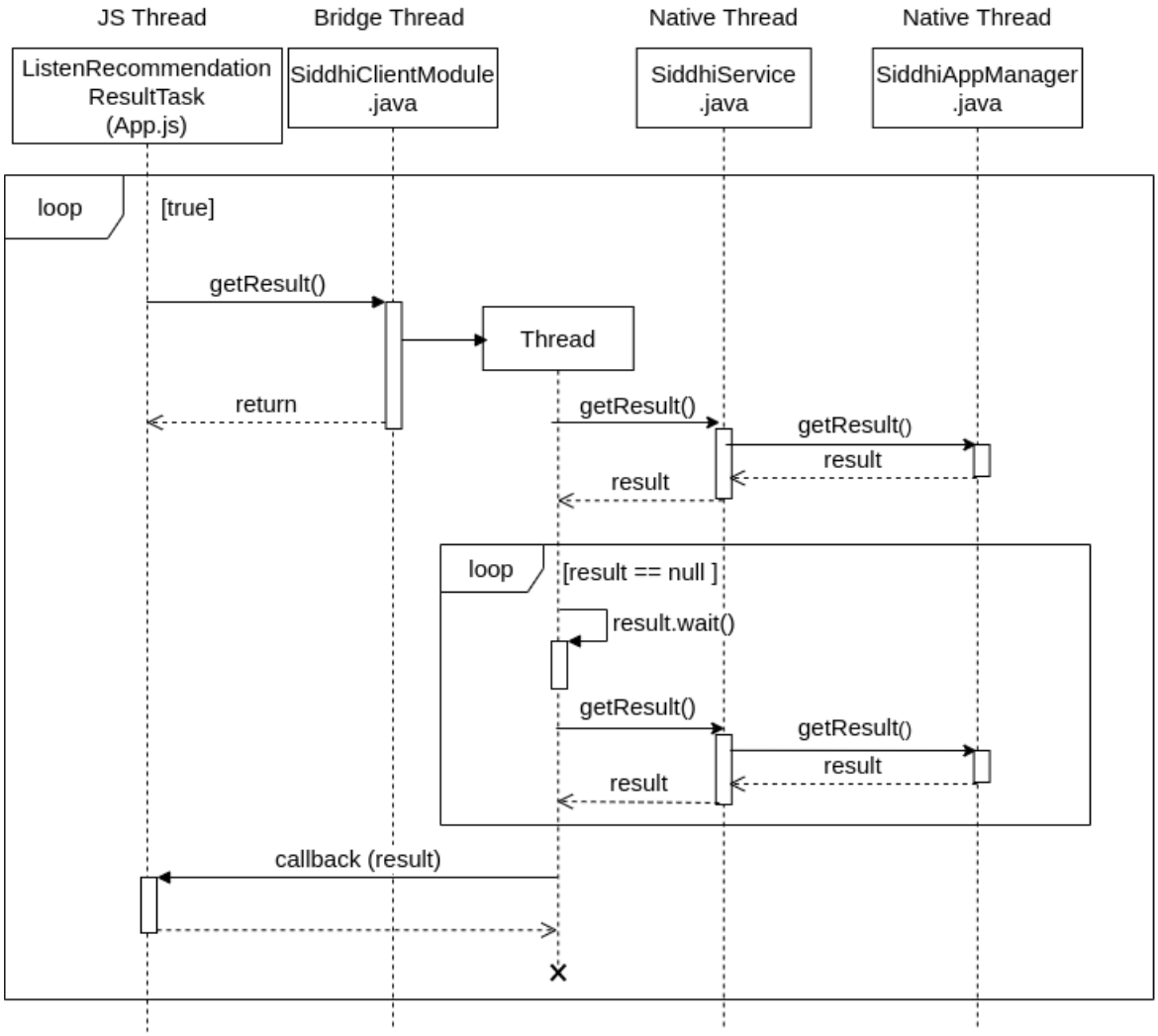


Figura 4: Diagrama de secuencia de la obtención del resultado de Siddhi desde JavaScript

4.4. Integración de las reglas de Siddhi con la interfaz de usuario

Después de tener implementadas las interfaces gráficas donde el usuario puede definir las reglas que le interesan, y de tener integrado Siddhi en el sistema, necesitamos la lógica que conecta ambas partes traduciendo las reglas definidas por el usuario mediante las interfaces a la sintaxis de Siddhi. A continuación, explicamos cómo se generan las reglas, cómo se actualizan cuando un usuario añade, modifica o elimina alguna y cómo se declaran en Siddhi. También mostramos un diagrama de paquetes general de la aplicación móvil.

Actualización de la aplicación Siddhi y reglas

Cuando el usuario define una regla, los datos introducidos se almacenan en la base de datos del dispositivo (la base de datos está implementada con la tecnología *Realm*), y cuando se modifica una regla existente o se elimina, se actualizan los datos. Hemos definido dos nuevos esquemas en la base de datos local, uno para las *context rules* y otro para las *triggering rules*. Se decidió agrupar todos los tipos de *context rules* en un mismo esquema porque así la recuperación de todas las reglas para su listado era más sencilla. Hemos desarrollado un módulo (*createSiddhiApp*) que se encarga de leer de la base de datos local las reglas almacenadas y traducirlas a la sintaxis de Siddhi. Generamos una cadena con todas las reglas definidas y se la enviamos a Siddhi para posteriormente iniciar la aplicación. Para iniciar una nueva aplicación Siddhi y su conjunto de reglas, la biblioteca Java del motor CEP lo hace en forma de cadena de caracteres.
El motor Siddhi se detiene y se vuelve a iniciar cada vez que se hace una modificación en las reglas definidas. El protocolo diseñado para gestionar el conjunto de reglas de Siddhi (aplicación Siddhi) es el siguiente: 1) una *context rule* es editada o eliminada o una *triggering rule* es creada, modificada o eliminada por el usuario, 2) se genera la cadena de caracteres con las todas las reglas definidas (incluidas las *context rules*), 3) se detiene el motor CEP, y 4) inicia de nuevo el motor CEP con las nuevas reglas definidas. Es importante destacar que la aplicación Siddhi solo está en funcionamiento cuando hay *triggering rules* que el usuario ha seleccionado como activas.

Implementación de reglas

La sintaxis de Siddhi es muy similar a SQL. Las reglas se han definido declarando consultas que consumen *streams* de entrada y emiten eventos a los *streams* de

salida. Cuando Siddhi recibe el contexto con el formato que hemos indicado anteriormente, lo primero que debemos hacer es leerlo y separar los campos para que pueda ser procesado. El contexto tiene dos partes: *UserContext* y *Observations*. El primer paso será separar el contexto y emitir los eventos a los *streams* con los mismos nombres. En el caso de *Observations* cada elemento del vector será un evento en el *stream*. Para poder procesar el contexto de entrada se ha utilizado la extensión *Siddhi Execution JSON* (`https://siddhi-io.github.io/siddhi-execution-json/api/latest/`). También se han utilizado otras extensiones: *Siddhi Execution Time* (`https://siddhi-io.github.io/siddhi-execution-time/`) para poder trabajar con las horas y las fechas adecuadamente y *Siddhi Execution String* (`https://siddhi-io.github.io/siddhi-execution-string/`) para trabajar con *strings* en las reglas.

Cada tipo de regla tiene su propia implementación y además las *context rules* pueden negarse. Para permitir esto, en la *context rule* que esté negada se añadirá otra regla donse se invertirán las condiciones de su regla normal. En el Anexo C presentamos ejemplos de la implementación de todas las reglas.

De esta manera, el prototipo móvil está finalizado y en la siguiente sección mostramos un diagrama de paquetes del proyecto que recoge todo lo mencionado.

Diagrama de paquetes del proyecto

Después de haber desarrollado el prototipo móvil, la estructura del proyecto resultante puede verse en la Figura 5, y en la Figura B.12 el diagrama completo, presente en el Anexo B.3 junto a su correspondiente descripción.

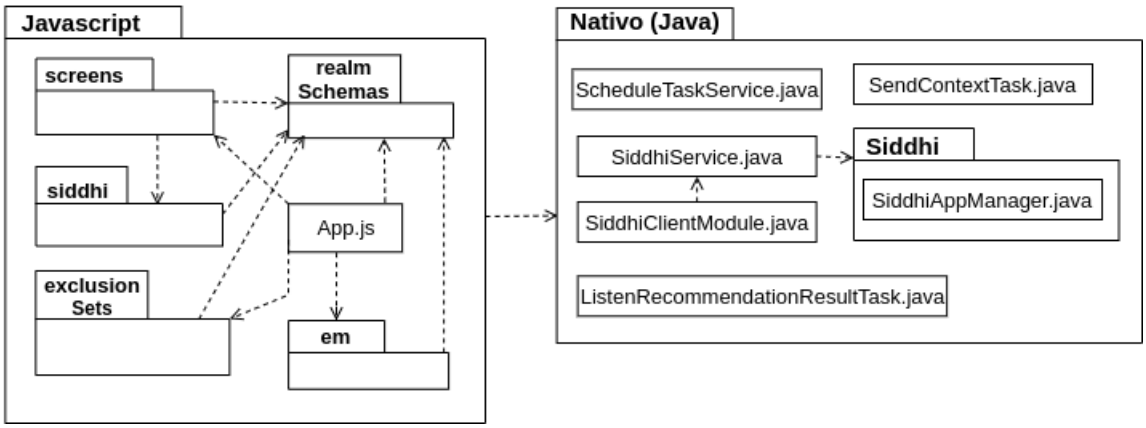


Figura 5: Diagrama simple de paquetes de la aplicación móvil

Capítulo 5

Implementación del Gestor de Entorno de prueba

El *Gestor de Entorno* es el componente de la arquitectura cuya tarea principal es ofrecer recomendaciones a los usuarios que se encuentran dentro su entorno. Para ello debe tener un registro de posibles actividades que serán recomendadas y, por tanto, otra de sus tareas es gestionar dichas actividades. El prototipo móvil se comunicará con este componente para identificar al usuario y solicitar nuevas actividades. Para esto último, el prototipo enviará al *Gestor de Entorno* los tipos de recomendaciones de los que debe buscar actividades para el usuario y que han sido calculados por el dipositivo móvil gracias a *Siddhi*, y la información del contexto del usuario que este haya querido compartir. El *Gestor de Entorno* deberá permitir la gestión de las actividades (creación de nuevas, modificación, consulta y borrado).
En la Sección 5.1 explicamos cómo se ha implementado y qué tecnologías se han utilizado. En la Sección 5.2 mencionamos los dos tipos de recomendadores con los que se ha probado la arquitectura.

5.1. Implementación y tecnologías

En el trabajo presentado en [1] se realizó una implementación del *Gestor de Entorno* pero el proyecto de dicho componente no fue localizado por lo que en hemos podido reutilizarlo. Sin embargo, sí que había documentación sobre cómo se había realizado. Se desarrolló como un servidor web y la API de las operaciones que debía exponer, su descripción, y la documentación relacionada con el esquema de la base de datos desarrollada estaban disponibles. Debido a que la aplicación móvil se desarrolló sobre el prototipo ya existente, donde la gestión de las actividades y la comunicación con *el Gestor de Entorno* ya estaban implementadas, se decidió reutilizar la propuesta de implementación mencionada.

El servidor web se ha desarrollado utilizando ***Spring Boot*** (`https://spring.io/projects/spring-boot`), al igual que en la propuesta original. Este framework de Java

permite desarrollar y lanzar aplicaciones web de manera muy sencilla e implementar una *API REST*, y facilita la utilización de bibliotecas de aprendizaje automático como ***Apache Mahout*** (`https://mahout.apache.org/`), biblioteca utilizada en el desarrollo de este proyecto para la tarea de recomendación. Esta biblioteca ha sido elegida porque contiene algoritmos de recomendación ya implementados y su utilización es muy sencilla. Para documentar la API del servidor se ha añadido al proyecto de *Spring Boot* la tecnología ***Swagger*** (`https://swagger.io/`) con la especificación *OpenAPI 3.0*, tal y como se indica en [9]. La base de datos planteada en la propuesta anterior era una base de datos relacional. Para implementarla se ha escogido el sistema gestor de bases de datos ***PostgreSQL*** (`https://www.postgresql.org/`), ya que es una tecnología de código abierto y su instalación y uso son gratuitos. El servidor web y la base de datos durante el desarrollo se desplegaron de manera remota utilizando ***Heroku*** (`https://id.heroku.com`) para eliminar carga al equipo donde se realizó el proyecto. Durante las pruebas, la necesidad de leer un fichero estático (un CSV) para el proceso de recomendación con Apache Mahout dificultó el despliegue en remoto y para las pruebas finales se decidió tener el servidor web en local junto a la base de datos lanzada en un contenedor ***Docker***.

5.2. Recomendadores

Para la tarea de recommendación se han definido dos tipos de recomendadores: 1) recomendador aleatorio y 2) recomendador con *Apache Mahout*. La operación del servidor que inicia el proceso de recomendación y obtiene las actividades como resultado, envía al servidor el identificador del usuario, los tipos de recomendaciones que deben proporcionarse al usuario e información relacionada con el contexto (aquella que el usuario ha decidido compartir). En nuestro proyecto el recomendador no utiliza esta información de contexto, ya que para probar nuestro prototipo nos interesa enfocarnos que se devuelvan actividades de los tipos detectados en el dispositivo móvil, donde si que utilizamos los datos de contexto. En una futura ampliación del recomendador debería valorarse si utilizar esa información para realizar las recomendaciones (por ejemplo la localización). En el prototipo actual, la información del contexto es importante en las decisiones tomadas por el dispositivo móvil.

Recomendador aleatorio

Este recomendador se desarrolló para realizar pruebas más sencillas ya que su implementación no requiere de ninguna tecnología externa. Es un recomendador básico que lo que hace es recuperar aquellas actividades que no han sido compartidas todavía

al usuario, filtrar aquellas que son de los tipos solicitados por el dispositivo móvil, y devolver un número aleatorio (entre 0 al 10) de esas actividades, sin tener en ningún momento en cuenta las preferencias del usuario.

Recomendador con Apache Mahout

El objetivo de utilizar *Apache Mahout* en el proyecto es simular un proceso de recomendación más real al usuario que con el recomendador aleatorio (donde no se tienen en cuenta los gustos del usuario). Los diferentes algoritmos de recomendación existentes se describen con detalle en el Anexo G.3. Debido a los datos utilizados y que presentamos en la Sección 6.3, el algoritmo que hemos utilizado con *Apache Mahout* es de filtrado colaborativo ítem-ítem, que busca recomendar ítems según la similitud con otros, teniendo en cuenta las valoraciones de los ítems por parte de los usuarios.

Capítulo 6

Evaluación experimental

Se han realizado dos tipos de pruebas: pruebas de rendimiento y pruebas de resultados. En la Sección 6.1 se explican una serie de experimentos realizados para evaluar el rendimiento de Siddhi. En la Sección 6.2 presentamos un escenario ficticio, en la Sección 6.3 hablamos sobre los datos de prueba utilizados sobre ese escenario y en la Sección 6.4 sobre los resultados obtenidos.

6.1. Pruebas de rendimiento

Estas pruebas se realizaron antes de implementar las pantallas y las reglas del prototipo actual con el objetivo de probar el rendimiento de Siddhi en un dispositivo Android. Las pruebas fueron realizadas sobre un dispositivo Android con un Qualcomm Snapdragon 626 (octacore A53, 2,2 GHz) de procesador, 4 GB RAM y Android 8.1.0. Se establecieron 7 tipos de *context rules* y se elaboró un script con Python que definía *triggering rules* de manera aleatoria combinando esas *context rules* (evitando combinar reglas contradictorias, por ejemplo, hora=13:30 AND hora=14:27). Las *triggering rules* resultantes de estas combinaciones podían contener desde 1 *context rule* hasta 7. Se crearon un total de 300 *triggering rules*. Por otro lado, se generaron 21 contextos de prueba que entendía Siddhi mediante otro script con Python. Para realizar los experimentos, añadimos todas las *context rules* a una aplicación de Android sencilla y que tenía Siddhi integrado, y las n primeras *triggering rules* de las generadas (este número cambiaba en cada ejecución hasta llegar a 200). Cada 20 segundos se enviaba un contexto nuevo de los generados, y se medía la latencia de activación de una *triggering rule* con cada cambio de contexto, siendo el tiempo entre el cambio de contexto y el momento en el que se detecta la activación de un tipo de recomendación. A partir de un mismo contexto pueden activarse varios tipos de recomendación y por eso se registraba la latencia de cada activación.

A continuación se explican los resultados obtenidos. En la gráfica de la izquierda de la Figura 6 se muestra la latencia de activación media, la mínima (la de la primera regla activada) y la máxima (la de la última regla activada). Esta última incrementa considerablemente con el número de *triggering rules*. Sin embargo, la latencia media

incrementa lentamente y la mínima no se ve afectada por el número de *triggering rules* definidas. Esto se debe a que Siddhi procesa las reglas una a una, por lo tanto debe ser similar entre cada experimento. Como conclusión, el número de triggering rules definidas es muy alto, mientras la latencia en el peor caso no supera los 220 milisegundos, un tiempo muy pequeño. Además se considera que es improbable que el usuario defina tantas *triggering rules* y si lo hiciera, las latencias serían igualmente aceptables. La gráfica de la derecha refleja el número de *triggering rules* activadas frente al número de definidas. Como es lógico, cuanto mayor es el número de reglas definidas, mayor es el número de reglas que pueden ser activadas por un cambio de contexto.

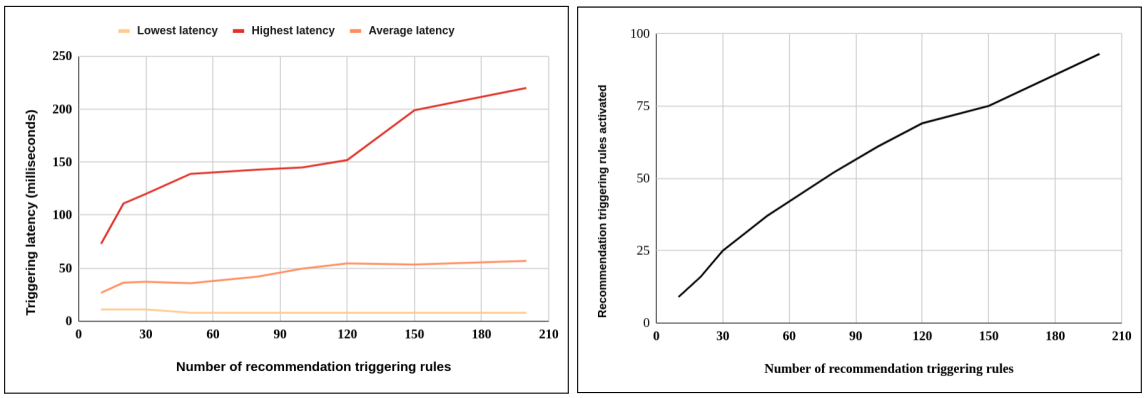


Figura 6: Evaluación de la latencia de activación (izquierda) y el número de reglas activada (derecha)

Para una explicación más detallada de los experimentos puede consultarse el artículo presentado en el Anexo I.

6.2. Descripción breve del escenario

El perfil del usuario que se ha escogido de prueba es una usuaria llamada Alicia que realiza sola un viaje de interés turístico a Madrid el fin de semana del 26, 27 y 28 de noviembre. Alicia ya estado otras veces en Madrid. Se aloja en un apartamento por la zona de Malasaña y se mueve por las zonas de alrededor (Chueca, la zona del Palacio Real, etc). Alicia está interesada en todas las categorías de recomendaciones, excepto en la de alojamientos. Ha definido 11 *context rules*, 7 *triggering rules* (cada una activa una categoría distinta), y 3 *exclusion sets*. En el Anexo H.1 aparece una descripción más detallada del escenario.

El objetivo principal de la evaluación con este escenario es comprobar el correcto funcionamiento del módulo de detección de los tipos de recomendación que hay que activar.

6.3. Datos de prueba utilizados

Para evaluar la arquitectura necesitábamos datos sobre **actividades** para recomendar, y también **valoraciones de usuarios** a esas actividades y poder utilizar un tipo de recomendador disponible en Apache Mahout. La primera opción que valoramos fue la utilización de conjuntos de datos reales. Buscamos conjuntos de datos de actividades, valoraciones y usuarios reales. En [10] se presentan y se analizan varios conjuntos de datos para los *Context Aware Recommender Systems*, entre los cuales se menciona ***TripAdvisor***. Ese trabajo incluye conjuntos de datos de *TripAdvisor* disponibles en [11], pero esta opción fue descartada porque sólo había datos sobre hoteles y queríamos más categorías. Como alternativa valoramos utilizar los conjuntos de datos disponibles en el proyecto ***Tourpedia*** (`http://tourpedia.org/`) presentado en [12]. Los datos disponibles tenían información de las actividades y valoraciones sobre ellas, pero no aparecía la información de los usuarios que aportaban las valoraciones; por lo tanto, descartamos también esta opción. La siguiente alternativa fue obtener datos reales sobre las actividades del portal de datos abiertos de alguna ciudad y generar datos falsos de valoraciones y usuarios con la herramienta ***AUTO-DataGenCARS*** [13]. Finalmente no pudimos utilizar tampoco esta aproximación porque la herramienta no podía generar los datos para el caso concreto que nosotros necesitábamos (datos de valoraciones y usuarios de unos ítems ya existentes). Finalmente, la decisión tomada fue extraer los datos del portal de datos abiertos de Madrid y generar los datos de valoraciones a esos usuarios mediante un script, tratando de emular algo realista. Todos los scripts implementados para la gestión de los datos de prueba han sido desarrollados con *Python*.

Datos de actividades

Los datos de posibles actividades y lugares para recomendar al usuario se buscaron en los portales de datos abiertos de varias ciudades. Se consultaron los de Zaragoza, Barcelona y Madrid y finalmente se optó por los datos de este último, porque se adaptaban más a nuestras necesidades. Estaban disponibles en portal de datos abiertos de Madrid (`https://datos.madrid.es/portal/site/egob`). En el Anexo H.2 se incluyen las fuentes de las que se han extraído estos datos. La información que contiene cada actividad es: 1) ***id***, identificador generado por nuestro sistema; 2) ***title***, nombre de la actividad; 3) ***author***, nombre del gestor de entorno; 4) ***authorId***, identificador alternativo proporcionado por el autor; 5) ***description***, pequeña descripción de la actividad; 6) ***img***, enlace a una imagen de la actividad (es opcional); 7) ***longitude y latitude***, longitud y latitud de las coordenadas terresres; 8) ***begin y ending*** fecha de

inicio y de finzalización (opcionales); 9) ***category***, categoría de actividad de nuestra aplicación; y 10) ***subcategories***, subcategorías a las que pertenece. Para cada una de las categorías se ha exportado su información en un CSV y posteriormente se ha insertado en la base de datos mediante un script.

Datos de valoraciones de usuarios

Los datos de valoraciones por parte de usuarios a las actividades mencionadas en el apartado anterior se han obtenido generándolas artificialmente mediante un script. Usando el script se han generado un total de 12249 valoraciones a diferentes actividades y de diferentes usuarios. El script genera un fichero con el formato *<id-usuario, id-actividad, valoración (0 al 5)>*. Los datos han sido elaborados para el escenario propuesto en la Sección 6.2. Cada usuario queda representado por un identificador numérico. Primero se creaban algunas valoraciones positivas (4 o 5) correspondientes al usuario de prueba (usuario que sigue la descripción de 6.2) a actividades de las subcategorías que resultan de su interés y que se presentan en la descripción del escenario. Posteriormente se crean grupos de un número determinado de usuarios que valoran aquellas subcategorías de una categoría que el usuario tiene interés de manera positiva (4 o 5) y actividades que no son de esas categorías con una negativa (0 al 3). De esta manera, al utilizar un algoritmo de ***filtro colaborativo basado en ítems*** con *Apache Mahout*, al usuario de prueba se le recomendarán actividades que tengan valoraciones parecidas a las que él tiene interés.

6.4. Ejecución de las pruebas y resultados

Las pruebas se han desarrollado con los mismos datos y escenario con los dos recomendadores mencionados. Se creó un JSON que contenía distintos contextos del usuario (adaptados al escenario) y con un intervalo determinado (por ejemplo, 30 segundos) se leía uno nuevo y se enviaba a Siddhi, en lugar del capturado con el dispositivo. En el Anexo H.3 puede consultarse algún ejemplo concreto de resultado del procesamiento de alguno de los contextos de prueba.

La Figura 7 representa un dibujo de la ejecución del escenario definido con los contextos de prueba generados. El objetivo de esta ejecución es evaluar el sistema, verificando que la detección de tipos de recomendación que hay que activar y los *exclusion sets* funcionan adecuadamente. En la figura vemos la línea temporal (divida en dos), los números del centro representan el identificador del contexto. Las etiquetas de arriba incluyen los cambios de contexto que son relevantes y las acciones del usuario con las *triggering rules* (activarlas/desactivarlas). Además, la etiqueta de color gris

representa la edición de la *triggering rule VisitMuseums* (elimina la *context rule* que condiciona el tiempo atmosférico)). Las etiquetas inferiores y unidas con una flecha, representan los tipos de recomendación que Siddhi ha detectado que hay que activar. Aquellos que están subrayados, son los que han decidido los *exclusion sets* que hay que activar y son comunicados al gestor de entorno. Algún aspecto destacable:

- Con el contexto 4, Siddhi decide que según las reglas definidas (consultar Anexo H.1) deben activarse recomendaciones de tipo *PlacesOfInterest* y *Restaurants*. Finalmente los *exclusion sets* deciden que la recomendación de este último tiene prioridad y por lo tanto solo se recomiendan restaurantes. Este resultado es el esperado.
- Con los contextos 5, 14, 15 y 17 el sistema ha decidido que no debe activarse ningún tipo de recomendación. Este comportamiento es el esperado.
- En el contexto 2 se activan los tipos *Leisure* y *Museums*. En el contexto 3 no hay ningún cambio de contexto relevante que pueda cambiar el resultado. Sin embargo, al desactivar la regla que activa *Leisure*, en este caso sólo se activa la recomendación de tipo *Museums*.
- Antes de procesarse el contexto 17 se modifica la regla que activa los museos y se le elimina la *context rule* que condiciona la situación meteorológica. Esto ha causado que se recomienden museos cuando está nublado, algo que no hubiera ocurrido sin la modificación de la regla.
- Antes del procesamiento del contexto 9 se añade un nuevo *exclusion set* que prioriza los espectáculos frente a las actividades de ocio. Como resultado de esto, el *exclusion set* decide que solo se activan recomendaciones de *ShowsHalls*.

Todos los resultados representados en la Figura 7 son correctos y por lo tanto se considera que el sistema se comporta adecuadamente. Con la ejecución de este escenario se han probado las reglas con diferentes cambios en ellas y contextos.

Por último, se indican algunas diferencias en los tipos de actividades recomendadas por cada recomendador. En la Figura 8 se puede ver un ejemplo de las recomendaciones de ambos recomendadores para el contexto 8. Con este contexto podrían recomendarse de los tipos: *Leisure, PlacesOfInterest, Museums*. El aleatorio ha devuelto 3 ítems como resultado y entre ellos hay 2 actividades de tipo *PlacesOfInterest* cuya subcategoría son *Instalaciones culturales* y una de *Leisure*, cuya subcategoría es *Exposiciones*. Según el escenario, estas actividades no resultarían en un principio de interés para el usuario, ya que las preferencias presentadas en el Anexo H.1 no coinciden. Sin embargo, el implementado con Apache Mahout ha devuelto 8 ítems de la categoría *PlacesOfInterest* y subcategoría *Edificios y monumentos*, que sí que forma parte de las preferencias

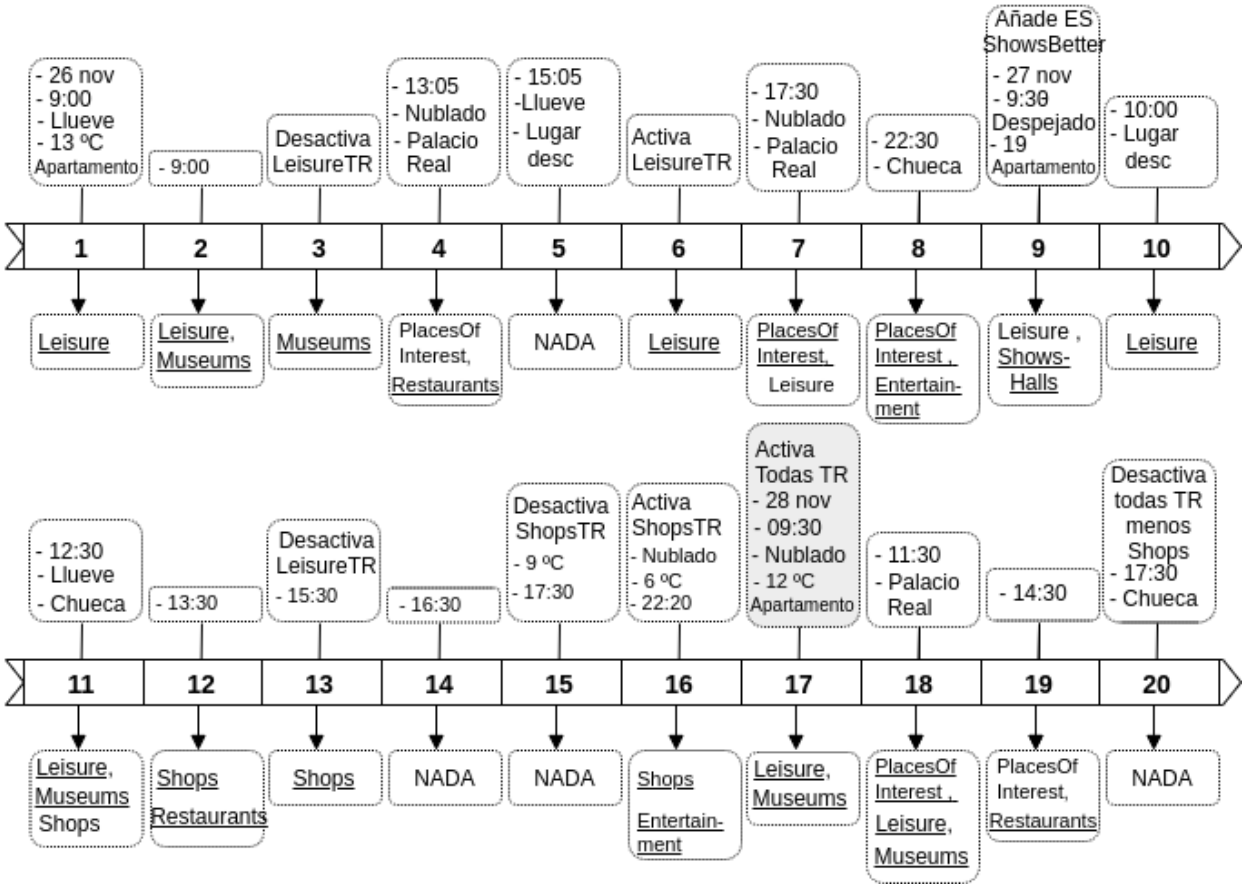


Figura 7: Arquitectura en alto nivel centrada en la activación de las recomendaciones

del usuario. Todos los resultados obtenidos son actividades que están presentes en el fichero de valoraciones que utiliza Apache Mahout como entrada para hacer las recomendaciones. Comparando ambos resultados podemos intuir que los resultados que proporciona el recomendador aleatorio no son tan reales como los de un recomendador más complejo. Lo ideal para ejecutar las pruebas habría sido contar con datos de valoraciones e ítems reales y además en mayor cantidad, ya que los algoritmos de filtrado colaborativo ítem-ítem tienen peores resultados si el número de valoraciones no es muy grande. El objetivo de tener un recomendador con *Apache Mahout* era probar la arquitectura con un recomendador que se aproximara más a alguno real. Trabajar de manera más profunda con el recomendador formaría parte del trabajo futuro del proyecto.

```
LOG: 4- Tipos de recomendación activadas por Siddhi para el contexto: 18
LOG: leisure,museums,placesOfInterest,
LOG: 5- Tipos de recomendación activadas filtradas por los EXCLUSION SETS:
LOG: placesOfInterest,museums,leisure,
LOG: 6- Items recomendados por el gestor de entorno:
LOG: Number of recommended items: 3
LOG: 2922; Teatro Amaya; PlacesOfInterest; Instalaciones culturales,
LOG: 601; Madrid 1862-1920: Galdós, relato de un nuevo paisaje urbano. Prorrogada hasta el 14 de nov
LOG: 2405; Galer&iacute,a &Eacute,boli; PlacesOfInterest; Instalaciones culturales,
```

(a) *Actividades recomendadas con el recomendador aleatorio*

```
LOG: 4- Tipos de recomendación activadas por Siddhi para el contexto: 18
LOG: leisure,museums,placesOfInterest,
LOG: 5- Tipos de recomendación activadas filtradas por los EXCLUSION SETS:
LOG: placesOfInterest,museums,leisure,
LOG: 6- Items recomendados por el gestor de entorno:
LOG: Number of recommended items: 7
LOG: 2344; Edificio Castellana 81 (Torre BBVA); PlacesOfInterest; Edificios y monumentos,
LOG: 2743;  CaixaForum Madrid; PlacesOfInterest; Edificios y monumentos Instalaciones culturales,
LOG: 2722; Puerta de Hierro; PlacesOfInterest; Edificios y monumentos,
LOG: 2807; Bolsa de Comercio de Madrid; PlacesOfInterest; Edificios y monumentos,
LOG: 2790; Palacio de Fern&aacute,n N&uacute,&ntilde,ez; PlacesOfInterest; Edificios y monumentos,
LOG: 3044;  Teatro Real; PlacesOfInterest; Edificios y monumentos,Instalaciones culturales,
LOG: 2461; Torre PwC; PlacesOfInterest; Edificios y monumentos,
```

(b) *Actividades recomendadas con el recomendador de filtrado colaborativo ítem-ítem*

Figura 8: Ejemplos de registros recogidos durante la ejecución con el recomendador de Apache Mahout

Capítulo 7

Conclusiones y trabajo futuro

Este capítulo es el último y se incluye lo siguiente: la Sección 7.1 explica los esfuerzos realizados y la conclusión personal; la Sección 7.2 presenta las conclusiones sobre el proyecto; y la Sección 7.3 el trabajo futuro que debería realizarse si se continuara.

7.1. Trabajo realizado y conclusión personal

En mi opinión, la elaboración de este proyecto no ha sido algo sencillo. El proyecto constaba de varias etapas diferentes entre sí que luego formaban un prototipo común. En cada una de estas etapas se han tenido que afrontar diferentes problemas, ya sea solucionando los problemas directamente o buscando alguna alternativa. Los obstáculos más importantes que se han afrontado han sido la puesta en marcha del prototipo previo, la implementación del prototipo del *Gestor de Entorno*, ya que no se localizó su proyecto y tuvo que rehacerse, y la integración de Siddhi con el prototipo previo. En la parte final del proyecto surgió el problema con el inicio de sesión mencionado en la Sección 4.3. A pesar de los problemas encontrados, considero que se han cumplido los objetivos del trabajo.

Este ha sido el primer proyecto de investigación que he realizado, además de ser el primer trabajo de esta extensión que he realizado sola. He aprendido aspectos sobre el desarrollo de aplicaciones en Android que desconocía; he conocido tecnologías nuevas para mí (*React Native, Siddhi y Apache Mahout*) y he ampliado mis conocimientos con otras ya conocía; he mejorado mis capacidades de búsqueda de información y comparación de tecnologías; y finalmente, he aprendido conceptos totalmente desconocidos para mí (algunos ejemplos son, *RS y CARS*, algoritmos de recomendación, procesamiento de eventos complejos). Además, desarrollar este trabajo me ha permitido colaborar por primera vez en un artículo de investigación. En la Tabla 2 se presentan las horas dedicadas aproximadas a cada tarea del proyecto y en la Figura 9 el cronograma.

Tarea	**Horas aproximadas**
Estudio previo	15 h
Prueba del prototipo móvil existente	30 h
Implementación gestor de entorno y base de datos	45 h

Búsqueda y selección de tecnología CEP	80 h
Integración de Siddhi en el prototipo existente	105 h
Implementación de nuevas pantallas de la aplicación	80 h
Integración de las reglas entre interfaces y Siddhi	70 h
Implementación de los *exclusion sets*	25 h
Datos, escenarios y pruebas	80 h
Memoria y documentación	80 h
Total	**610 h**

Tabla 2: Horas dedicadas al proyecto (aproximadas)

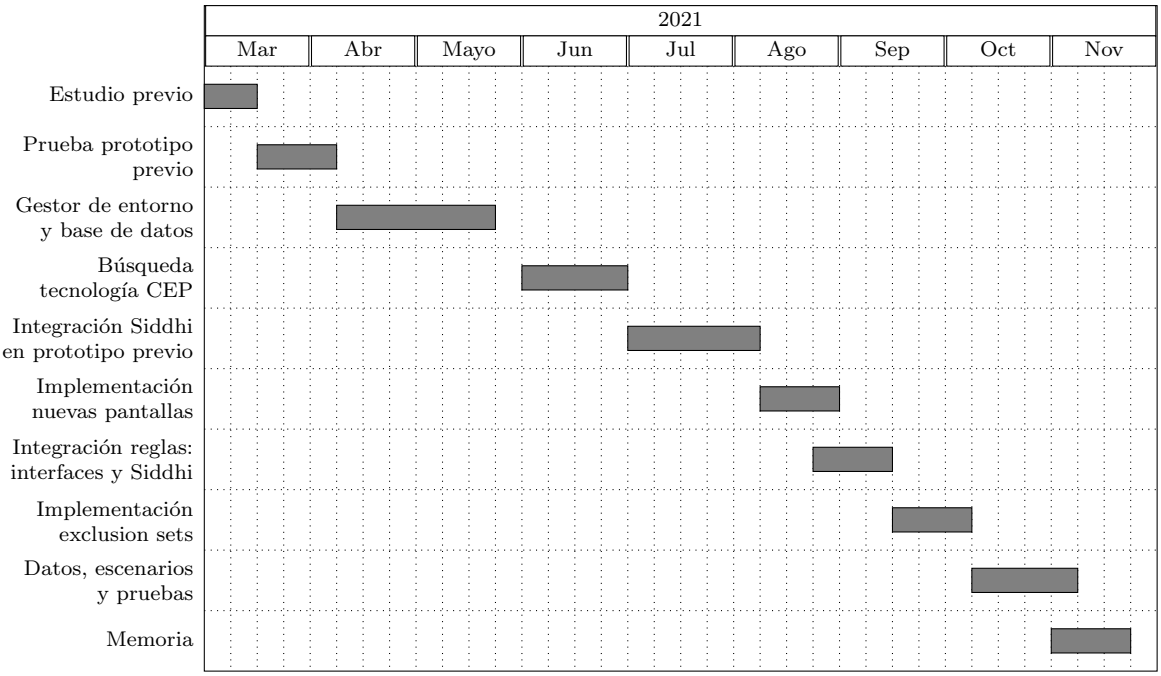


Figura 9: Diagrama de Gantt

7.2. Conclusiones del proyecto

El proyecto tenía como objetivo principal ampliar una arquitectura ya existente y evaluar su utilización con algún escenario real, además de permitir al usuario personalizar el proceso de recomendación y mejorar su privacidad. Como resultados globales del proyecto podemos mencionar:

- Se ha logrado implementar un módulo inteligente que detecta qué tipo o tipos de recomendación deben activarse según el contexto del usuario y las reglas definidas, y cuándo activarse. Esto se ha podido realizar con tecnología de procesamiento de eventos complejos, tal y como se proponía.

- El usuario puede personalizar el proceso de recomendación gracias a las *context*

rules, *triggering rules* y *exclusion sets*. Además, estos últimos permiten establecer prioridades entre tipos de recomendación.

- La decisión sobre qué tipos de recomendaciones deben activarse se completa en el dispositivo móvil con la información de contexto recogida. Esto es un gran avance con respecto el prototipo previo, ya que las decisiones se toman en el dispositivo y se comunican al gestor. Este podrá recibir (o no) la información del contexto que el usuario desee compartir, mejorando así la privacidad del usuario.

- El proyecto final está formado por una gran variedad de tecnologías que de alguna manera deben comunicarse entre sí: *Siddhi, React Native, Java y Realm* en la aplicación móvil; *Spring Boot, PostgreSQL, Apache Mahout y Swagger* para el gestor de entorno; y se han utilizado otras tecnologías como *Heroku* o *Docker* para desplegar el sistema.

Algo que debe destacarse es la elaboración del artículo presentado en el Anexo I y que ha sido aceptado en ***MoMM 2021****: The 19th International Conference on Advances in Mobile Computing & Multimedia*.

7.3. Trabajo futuro

La arquitectura implementada durante el proyecto podrá seguir siendo mejorada en un futuro. Algunas de las posibles tareas a realizar son:

1. Actualización de las dependencias y las versiones del proyecto de la aplicación móvil, con el objetivo de solventar posibles conflictos como el del inicio de sesión.

2. En relación a la anterior, otro posible trabajo a realizar sería la mejora de las interfaces con elementos y acciones más cómodos como por ejemplo el *drag and drop* para ordenar una lista de elementos.

3. Incorporación de nuevos tipos de *context rules* diferentes a los cuatro que se presentan en este proyecto.

4. Modificación de las *triggering rules* con el objetivo de que su creación sea más flexible y admita algunos aspectos como combinar *context rules* de manera que se presentan varias opciones que pueden cumplirse (similar a la conjunción *o*). Por ejemplo, *si estoy en la oficina* ***o*** *estoy en el centro de la ciudad, y hace buen tiempo, que me recomiende restaurantes*. Esto actualmente puede realizarse pero habría que definir dos *triggering rules* distintas, una para si estás en la oficina y hace buen tiempo, y otra para si estás en el centro y hace buen tiempo.

5. Mejora y ampliación del *gestor de entorno* y los recomendadores, ya que en este proyecto ese componente ha sido fundamental para realizar las pruebas pero no ha sido el foco del trabajo realizado.

Se cree que el código del prototipo desarrollado puede continuar mejorándose y extendiéndose con nuevas funcionalidades, para obtener un demostrador y un prototipo de interés para la investigación en sistemas de recomendación, por parte de los investigadores con los que he colaborado e incluso en el futuro por parte de otros grupos de investigación.

Bibliografía

[1] Manuel Herrero Gajón and Sergio Ilarri Artigas. Desarrollo de un prototipo de aplicación móvil para sistemas de recomendación proactivos. 2019.

[2] Ramón Hermoso, Sergio Ilarri, Raquel Trillo, and María del Carmen Rodríguez-Hernández. Push-based recommendations in mobile computing using a multi-layer contextual approach. In *Proceedings of the 13th International Conference on Advances in Mobile Computing and Multimedia*, pages 149–158, New York, NY, USA, 2015. Association for Computing Machinery.

[3] Ramón Hermoso, Sergio Ilarri, and Raquel Trillo-Lado. Proactive mobile cars in action: A first step towards making sense of context rules. In *2018 13th International Workshop on Semantic and Social Media Adaptation and Personalization (SMAP)*, pages 69–74, 2018.

[4] Kenny Warszawski. *Complex Event Processing for Internet of Things: Open-Source Frameworks Analysis*. PhD thesis, 2020.

[5] Miyuru Dayarathna and Srinath Perera. Recent advancements in event processing. 51(2), 2018.

[6] K. Tawsif, Jakir Hossen, Joseph Emerson Raja, Jesmeen Hoque, and Md Hossain. A review on Complex Event Processing Systems for Big Data. pages 1–6, 03 2018.

[7] Amarjit Singh Dhillon, Shikharesh Majumdar, Marc St-Hilaire, and Ali El-Haraki. A mobile complex event processing system for remote patient monitoring. In *2018 IEEE International Congress on Internet of Things (ICIOT)*, pages 180–183, 2018.

[8] Markus Schinle, Johannes Schneider, Timon Blöcher, Jochen Zimmermann, Sebastian Chiriac, and Wilhelm Stork. A modular approach for smart home system architectures based on android applications. In *2017 5th IEEE International Conference on Mobile Cloud Computing, Services, and Engineering (MobileCloud)*, pages 153–156, 2017.

[9] baeldung. Documenting a spring rest api using openapi 3.0, 2021. Available at `https://www.baeldung.com/spring-rest-openapi-documentation`, Accedido por última vez 26/11/2021.

[10] Sergio Ilarri, Raquel Trillo-Lado, and Ramon Hermoso. Datasets for context-aware recommender systems: Current context and possible directions. In *2018 IEEE 34th International Conference on Data Engineering Workshops (ICDEW)*, pages 25–28, 2018.

[11] Yong Zheng. *CARSK*it context-aware_data_sets. `https://github.com/irecsys/CARSKit/tree/master/context-aware_data_sets`, Accedido por última vez 26/11/2021.

[12] Stefano Cresci, Andrea D'Errico, Davide Gazzè, Angelica Lo Duca, Andrea Marchetti, and Maurizio Tesconi. Tour-pedia: a web application for sentiment visualization in tourism domain. *Come Hack with OpeNER! Workshop Programme*, page 18, 01 2014.

[13] Sergio Ilarri, María del Carmen Rodríguez, Raquel Trillo-Lado, Ramon Hermoso, and Ignacio Palacios. Auto-datagencars: Advanced user oriented tool *D*ata*G*en*CARS*. `http://webdiis.unizar.es/~silarri/AUTO-DataGenCARS/`. Accedido por última vez 26/11/2021.

[14] D. A. Asanov. Algorithms and methods in recommender systems. 2011.

[15] Catálogo de datos abiertos Madrid. Restaurantes con perfil turístico de la ciudad de madrid. `https://datos.madrid.es/portal/site/egob/menuitem.c05c1f754a33a9fbe4b2e4b284f1a5a0/?vgnextoid=ce33a73970504510VgnVCM2000001f4a900aRCRD&vgnextchannel=374512b9ace9f310VgnVCM100000171f5a0aRCRD&vgnextfmt=default.` , Accedido por última vez 26/11/2021.

[16] Catálogo de datos abiertos Madrid. Alojamientos de la ciudad de madrid. `https://datos.madrid.es/portal/site/egob/menuitem.c05c1f754a33a9fbe4b2e4b284f1a5a0/?vgnextoid=df42a73970504510VgnVCM2000001f4a900aRCRD&vgnextchannel=374512b9ace9f310VgnVCM100000171f5a0aRCRD&vgnextfmt=default.` , Accedido por última vez 26/11/2021.

[17] Catálogo de datos abiertos Madrid. Tiendas, comercios y mercados con perfil turístico de la ciudad de madrid. `https://datos.madrid.es/portal/site/egob/menuitem.c05c1f754a33a9fbe4b2e4b284f1a5a0/?vgnextoid=86e3a73970504510VgnVCM2000001f4a900aRCRD&vgnextchannel=374512b9ace9f310VgnVCM100000171f5a0aRCRD&vgnextfmt=default.` , Accedido por última vez 26/11/2021.

[18] Catálogo de datos abiertos Madrid. Locales de diversión y entretenimiento con perfil turístico de la ciudad de madrid. https://datos.madrid.es/portal/site/egob/menuitem.c05c1f754a33a9fbe4b2e4b284f1a5a0/?vgnextoid=54d4a73970504510VgnVCM2000001f4a900aRCRD&vgnextchannel=374512b9ace9f310VgnVCM100000171f5a0aRCRD&vgnextfmt=default. , Accedido por última vez 26/11/2021.

[19] Catálogo de datos abiertos Madrid. Puntos de interés turístico de la ciudad de madrid. https://datos.madrid.es/portal/site/egob/menuitem.c05c1f754a33a9fbe4b2e4b284f1a5a0/?vgnextoid=3b70a73970504510VgnVCM2000001f4a900aRCRD&vgnextchannel=374512b9ace9f310VgnVCM100000171f5a0aRCRD&vgnextfmt=default. , Accedido por última vez 26/11/2021.

[20] Catálogo de datos abiertos Madrid. Museos de la ciudad de madrid. https://datos.madrid.es/portal/site/egob/menuitem.c05c1f754a33a9fbe4b2e4b284f1a5a0/?vgnextoid=118f2fdbecc63410VgnVCM1000000b205a0aRCRD&vgnextchannel=374512b9ace9f310VgnVCM100000171f5a0aRCRD&vgnextfmt=default. , Accedido por última vez 26/11/2021.

[21] Catálogo de datos abiertos Madrid. Api. https://datos.madrid.es/portal/site/egob/menuitem.214413fe61bdd68a53318ba0a8a409a0/?vgnextoid=b07e0f7c5ff9e510VgnVCM1000008a4a900aRCRD&vgnextchannel=b07e0f7c5ff9e510VgnVCM1000008a4a900aRCRD&vgnextfmt=default. , Accedido por última vez 26/11/2021.

Lista de Figuras

Lista de Tablas

Anexos

Anexos A

Análisis de requisitos

En este anexo presentamos el análisis de requisitos de la aplicación desarrollada, incluyendo los requisitos funcionales y los no funcionales.

A.1. Requisitos funcionales

RF	Descripción
RF1	El usuario podrá identificarse en la aplicación y cerrar sesión cuando desee.
RF2	El usuario podrá seleccionar los tipos de recomendación que son interesantes para él.
RF3	El usuario podrá seleccionar qué información desea compartir con los gestores de entorno.
RF4	El usuario podrá consultar el listado de actividades que han sido recomendadas, marcadas como favoritas o su fecha ha vencido (históricas).
RF5	El usuario podrá ver la información de una actividad.
RF6	El usuario podrá descartar, puntuar del 1 al 5 y marcar como favorita cada actividad (así como desmarcarla si dejara de ser favorita).
RF7	El usuario podrá añadir nuevas ***context rules*** de los tipos: *time-based, calendar-based, location* o *weather*.
RF8	El usuario podrá consultar el listado de las ***context rules*** que haya definido previamente.
RF9	El usuario podrá eliminar, editar y consultar la información de las ***context rules*** que haya definido previamente.
RF10	El usuario podrá añadir nuevas ***triggering rules*** combinando dos o más *context rules* definidas previamente (y negarlas si lo desea) y el tipo de recomendación que desee que active la regla.
RF11	El usuario podrá consultar el listado de las ***triggering rules*** que haya definido previamente.
RF12	El usuario podrá eliminar, editar y consultar la información de las ***triggering rules*** que haya definido previamente.
RF13	El usuario podrá activar y desactivar las ***triggering rules*** que ha definido cuando desee.
RF14	El usuario no podrá eliminar una ***context rule*** que sea utilizada por una o más *triggering rules* directamente. En caso de que lo intentara, el usuario será informado de las *triggering rules* que utilizan la *context rule* que desea borrar.

RF15	El usuario podrá añadir nuevos ***exclusion set*** combinando dos o más tipos de recomendación e indicando un orden entre ellos para establecer prioridad entre los tipos.
RF16	El usuario podrá consultar el listado de los ***exclusion sets*** que haya definido previamente.
RF17	El usuario podrá ordenar el listado de ***exclusion sets*** para establecer prioridad entre ellos en caso de conflicto.
RF18	El usuario podrá eliminar, editar y consultar la información de los ***exclusion sets*** que haya definido previamente.

Tabla 3: Requisitos funcionales

A.2. Requisitos no funcionales

RNF	**Descripción**
RNF1	La aplicación será compatible para dispositivos móviles Android 6.0 o superior (API Level 23).
RNF2	La aplicación necesitará que el idioma del dispositivo sea *Inglés*.
RNF3	La aplicación necesitará que los permisos de ubicación y calendario del dispositivo se autoricen para su correcto funcionamiento.
RNF4	La aplicación necesitará tener conexión a Internet para recibir nuevas recomendaciones.

Tabla 4: Requisitos no funcionales

Anexos B

Diseño de la aplicación

En el Anexo B.1 se presenta el mapa de navegación correspondiente a la aplicación. El Anexo B.2 se incluyen las pantallas implementadas de la apliación mócil. En el Anexo B.3 se incluye el diagrama de paquetes completo del proyecto de la apliación móvil.

B.1. Mapa de navegación de la aplicación

La Figura B.1 representa el mapa de navegación de la aplicación móvil. Se pueden diferenciar las pantallas antiguas de las pantallas nuevas o las que han sido modificadas. Para navegar a las nuevas vistas de *exclusion sets, context rules* y *triggering rules* debe hacerse desde el menú de la aplicación. El menú se despliega arrastrando la pantalla de izquierda a derecha desde cualquier vista. A las pantallas de inicio, guardados e históricos se puede navegar mediante el menú o la barra de navegación inferior, y a las del perfil o ajustes desde el menú. Por otro lado, para observar la información de una actividad debe hacerse pulsándola desde cualquier listado de actividades. Se ha intentado mantener la consistencia en la forma de navegar de las nuevas pantallas con las del prototipo antiguo.

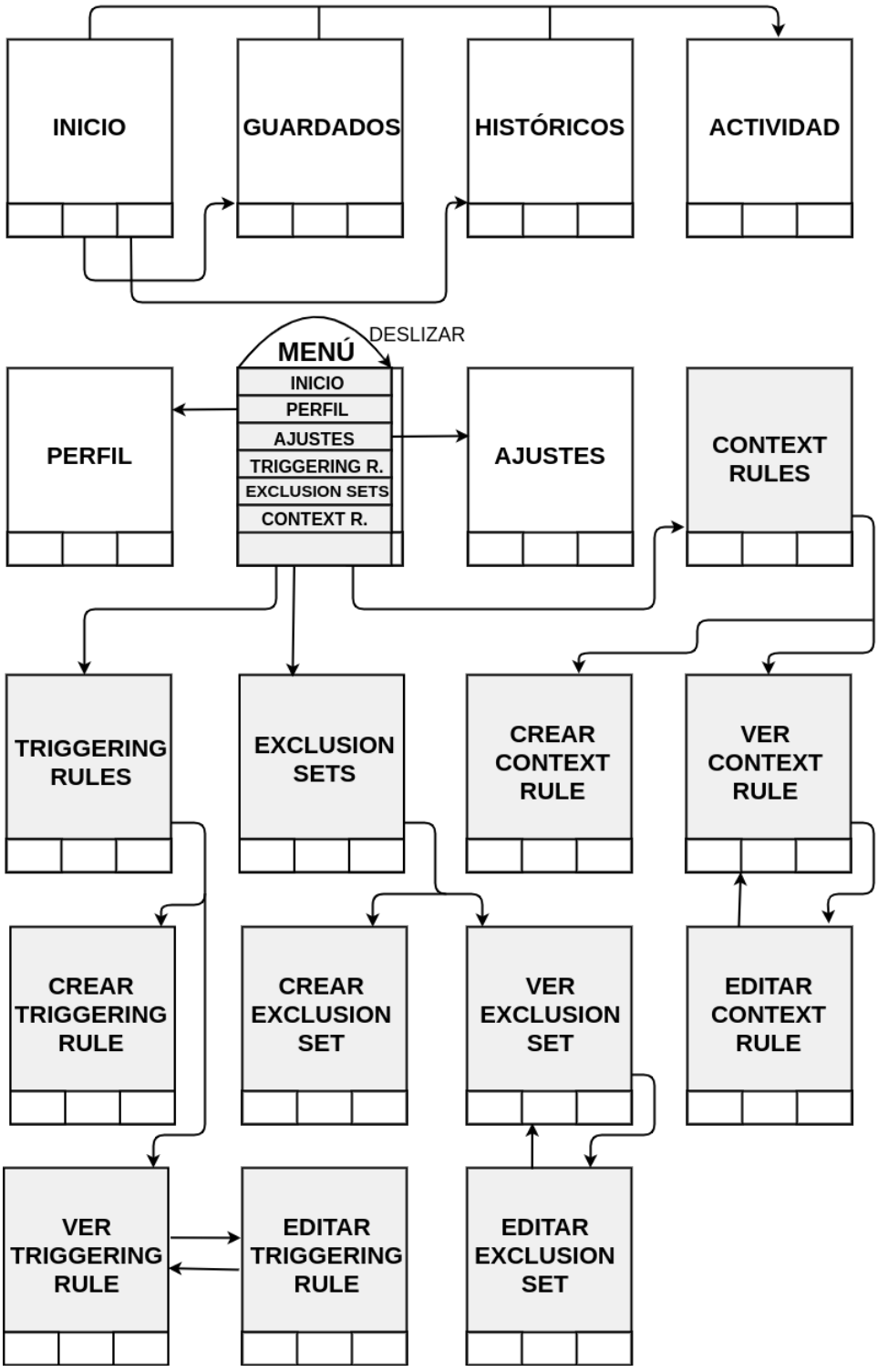
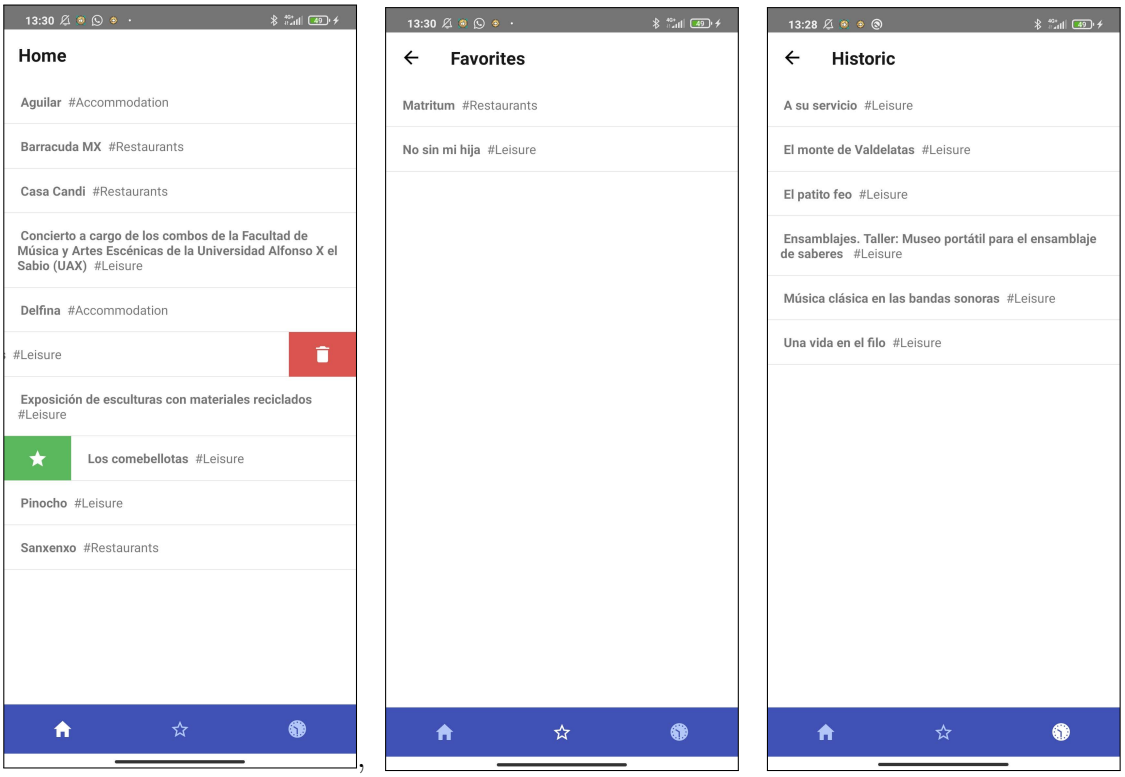


Figura B.1: Mapa de navegación. En gris, las nuevas pantallas incorporadas.

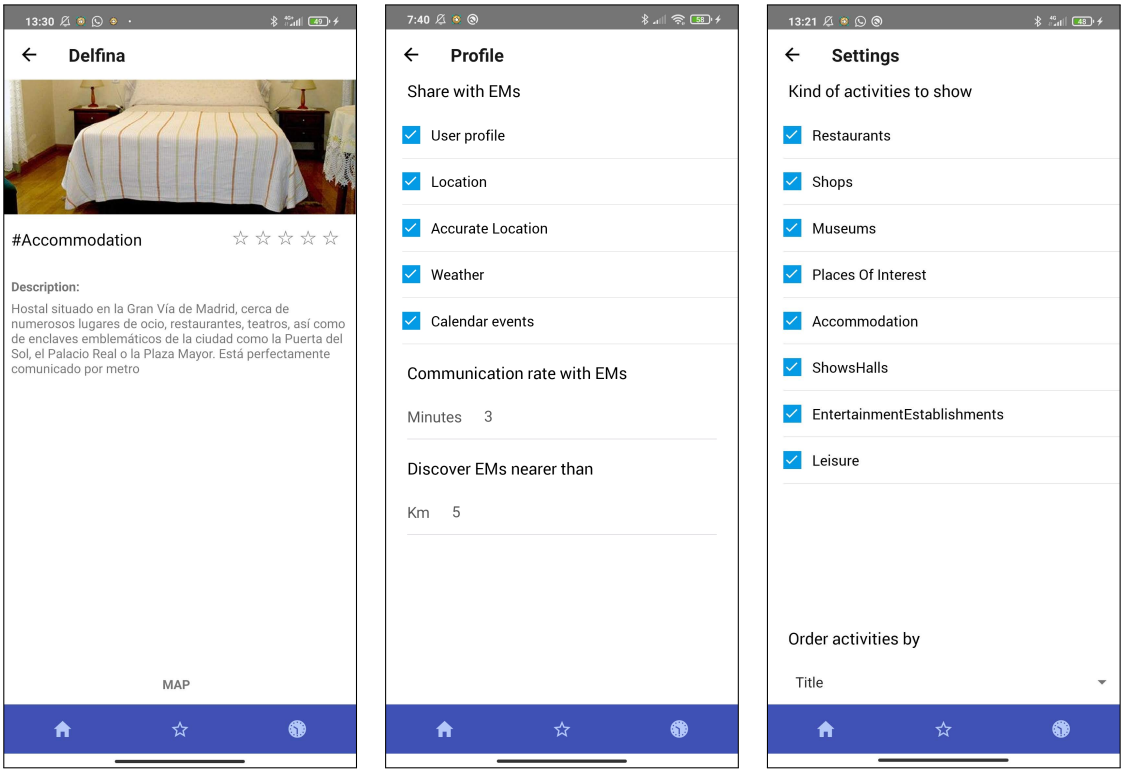
B.2. GUI de la aplicación

Las pantallas que muestran un listado de elementos tienen un comportamiento similar. En todas para el borrado de un item hay que deslizar de derecha a izquierda para que aparezca el botón rojo. Todas las pantallas relacionadas con la creación de reglas, edición o visualización también tienen un comportamiento similar, diferenciándoles únicamente el tipo de regla y sus correspondientes campos. Los nombres de las pantallas coinciden con los nombres de las vistas del mapa de navegación de la sección anterior.



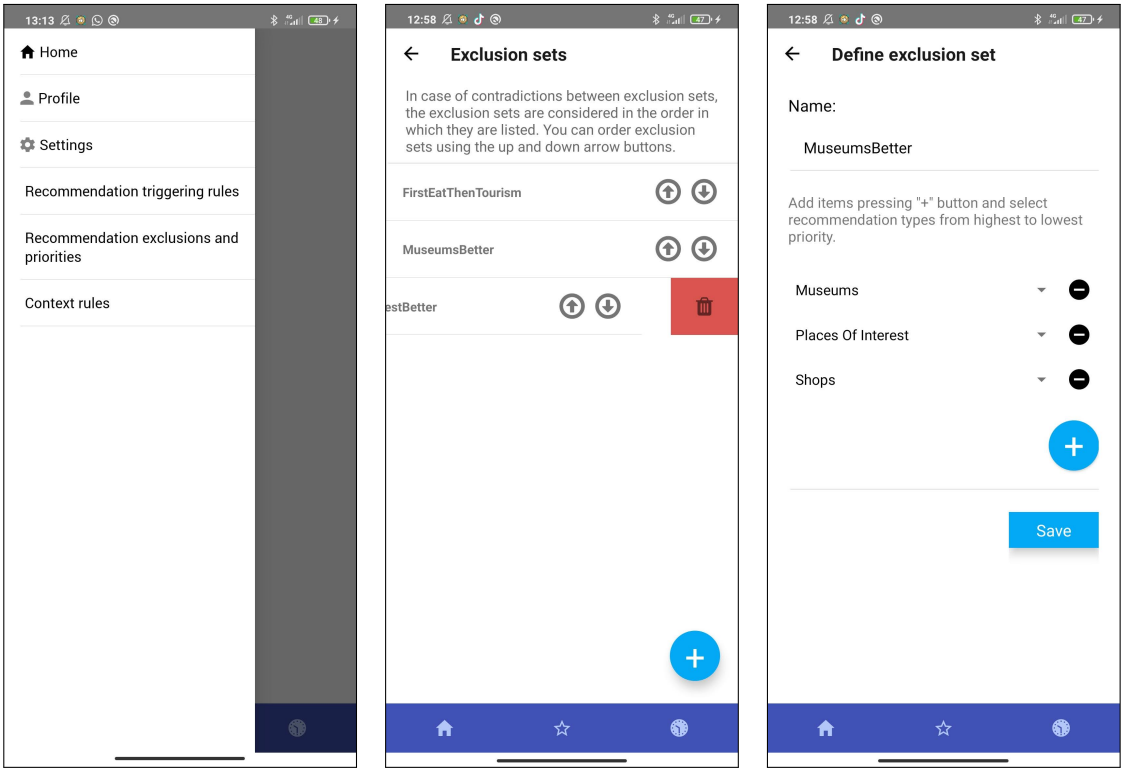
(a) *Inicio* **(b)** *Guardados* **(c)** *Prueba*

Figura B.2: Pantallas de listado de actividades



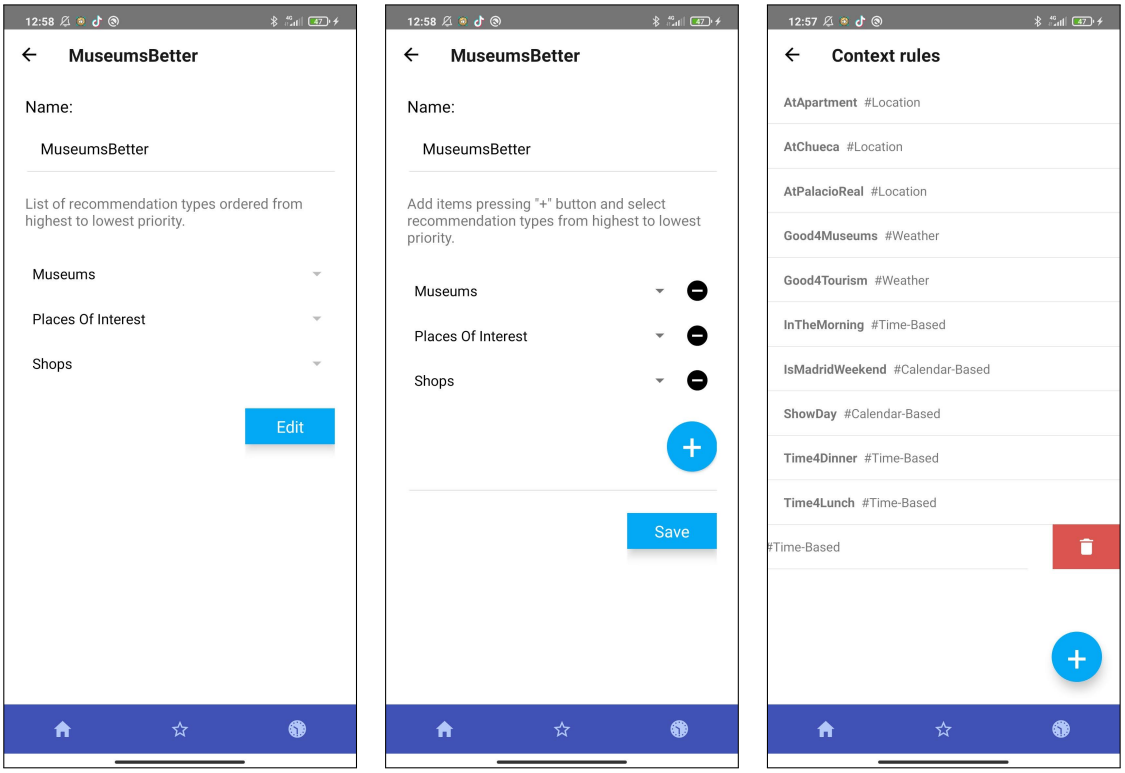
(a) *Actividad* **(b)** *Perfil* **(c)** *Ajustes*

Figura B.3: Pantalla de información de *Actividad*, de *Perfil* y de *Ajustes*



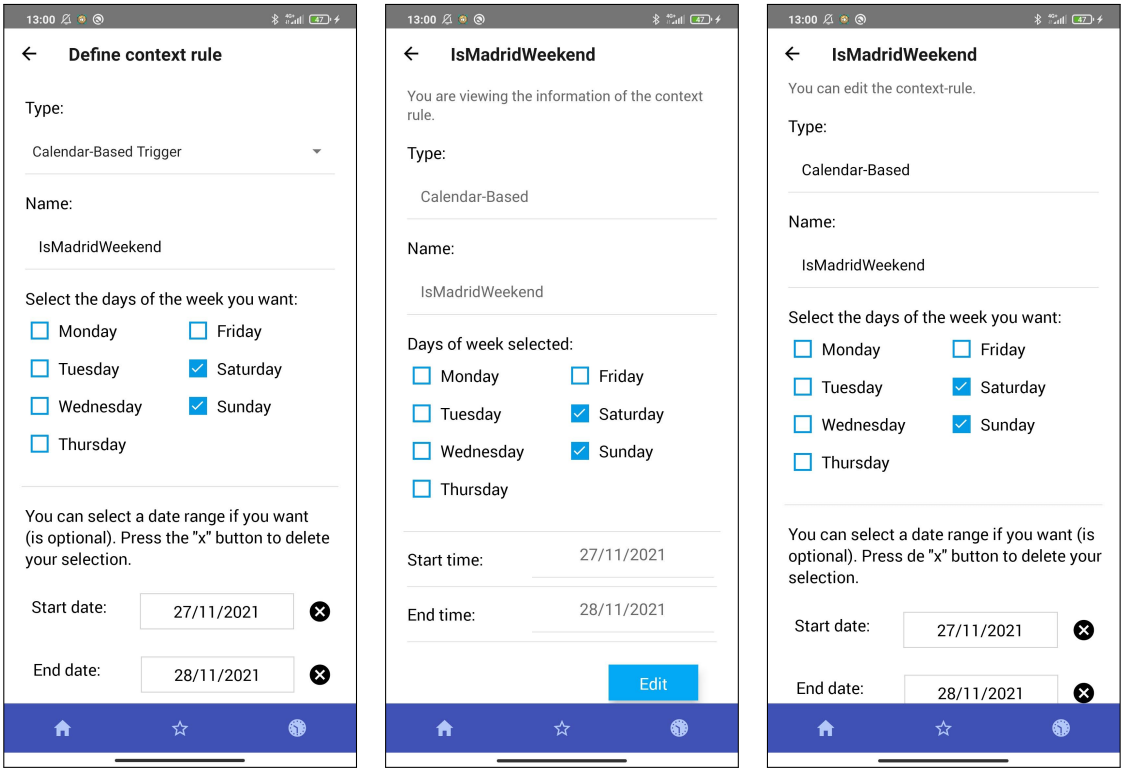
(a) *Menú* **(b)** *Exclusion Sets* **(c)** *Crear Exclusion Set*

Figura B.4: Pantalla de *Menú* y pantallas de *Exclusion Sets*



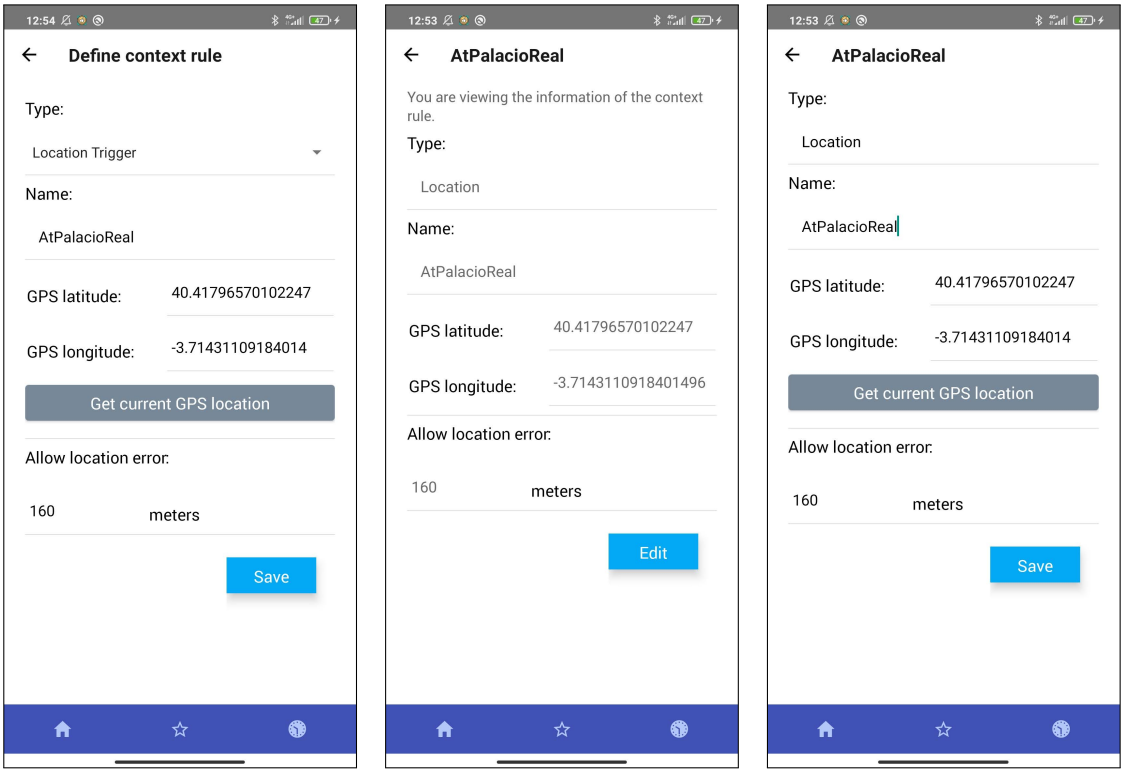
(a) *Ver Exclusion Set* **(b)** *Editar Exclusion Set* **(c)** *Context Rules*

Figura B.5: Pantallas *Exclusion Sets* y pantalla de listado de *Context Rules*



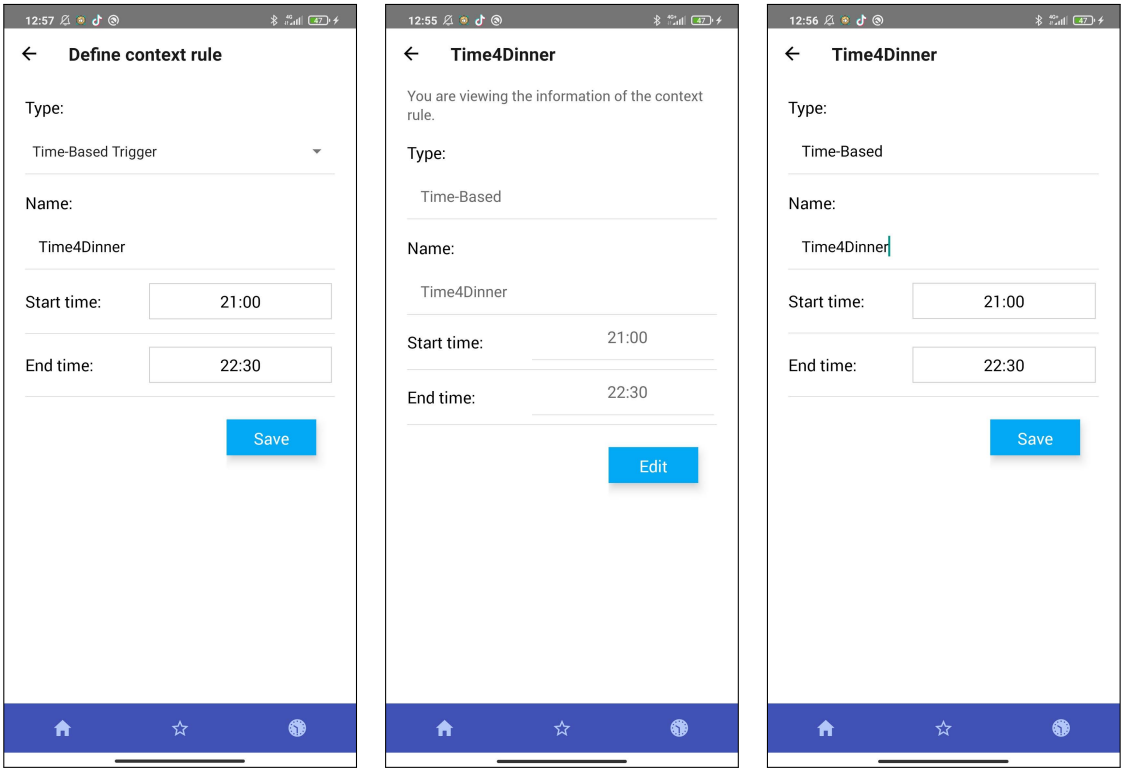
(a) *Crear* **(b)** *Ver* **(c)** *Editar*

Figura B.6: Pantallas de *Context Rules basadas en calendario*



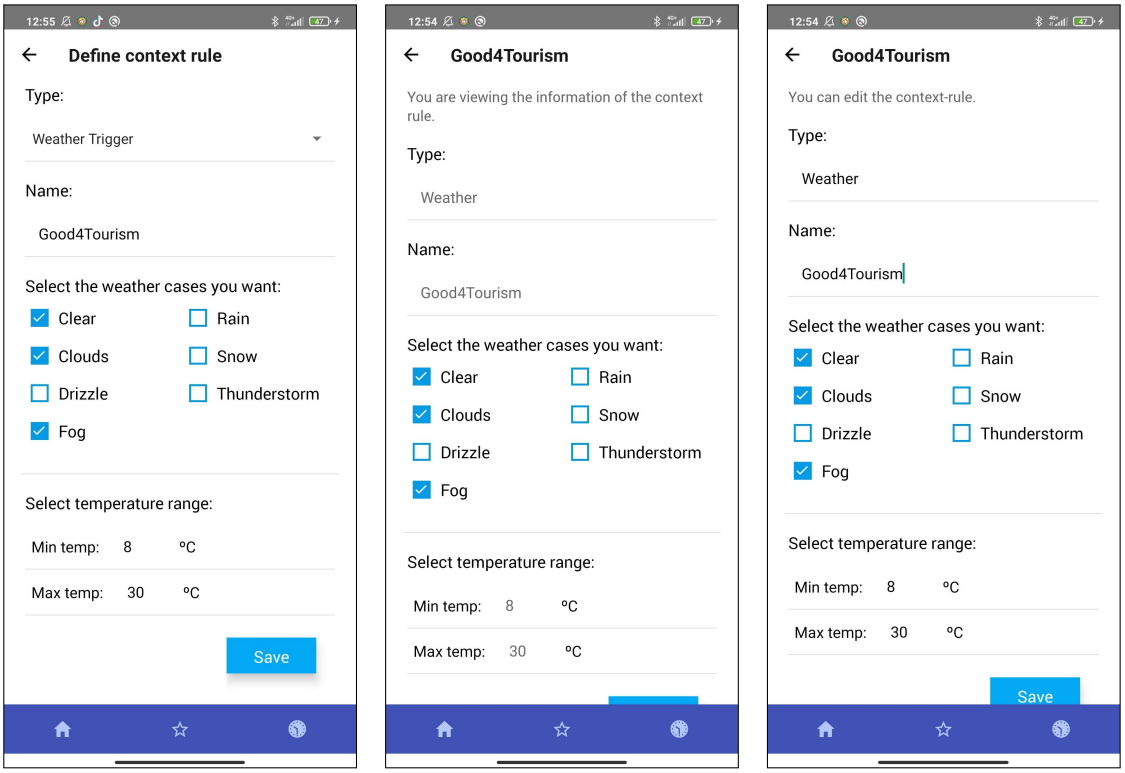
(a) *Crear* **(b)** *Ver* **(c)** *Editar*

Figura B.7: Pantallas de *Context Rules basadas en localización*



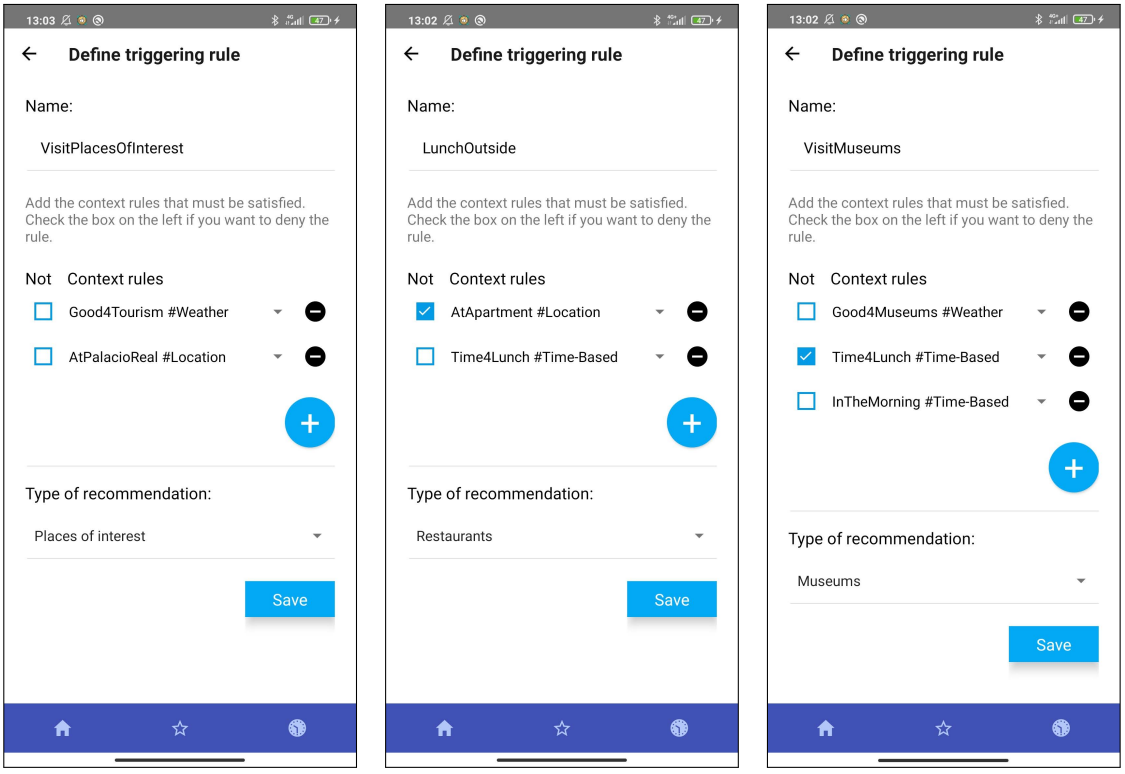
(a) *Crear* **(b)** *Ver* **(c)** *Editar*

Figura B.8: Pantallas de *Context Rules basadas en tiempo*



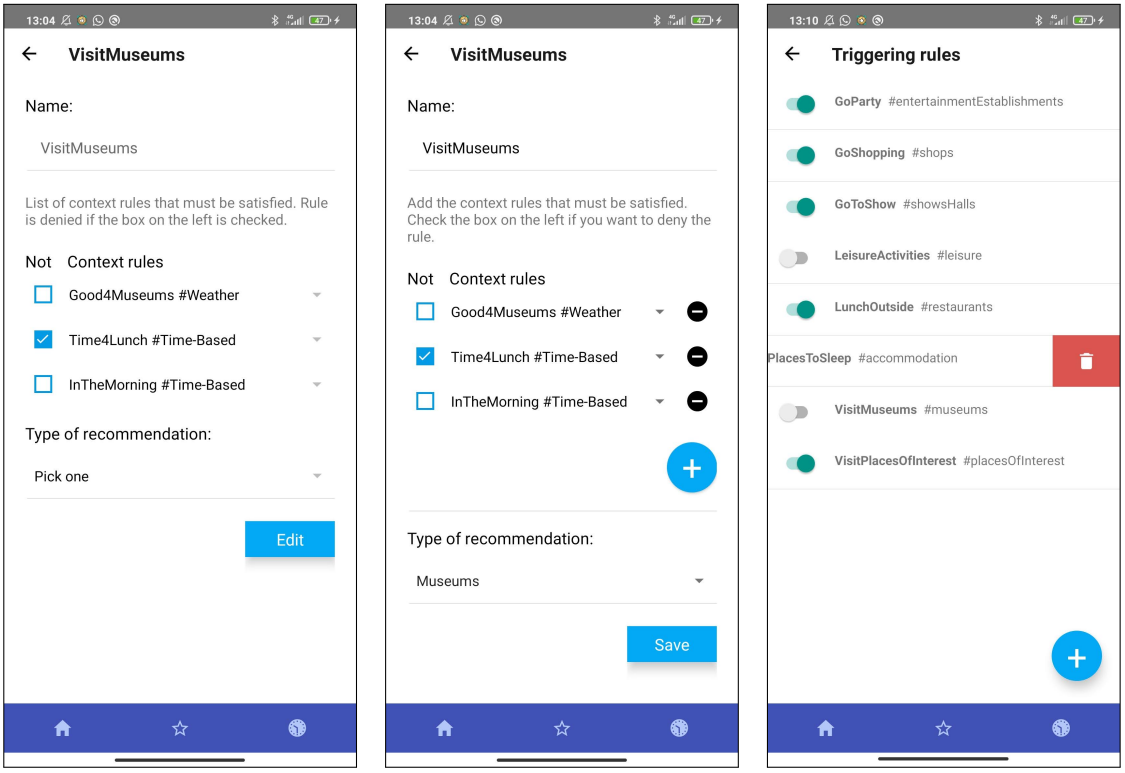
(a) *Crear* **(b)** *Ver* **(c)** *Editar*

Figura B.9: Pantallas de *Context Rules de tiempo atmosférico*



(a) *Dos Context Rules* **(b)** *Una Context Rule negada* **(c)** *Tres Context Rules, una negada*

Figura B.10: Ejemplos de pantallas de *Crear Triggering Rule*



(a) *Ver* **(b)** *Editar* **(c)** *Triggering Rules*

Figura B.11: Pantallas de *Editar Triggering Rules*

B.3. Diagrama de paquetes

Una de las principales dificultades encontradas en el desarrollo del proyecto ha sido tener parte del proyecto elaborada con código nativo y otra parte con Javascript. Para mostrar una visión general de la estructura del proyecto realizado para la aplicación móvil, se ha elaborado el diagrama de paquetes de la Figura B.12, donde podemos ver una clara separación entre ambas partes.

El paquete que representa la parte desarrollada con Javascript está formada por varios paquetes que se encargan de distintas tareas: 1) ***screens***, donde está la lógica relacionada con la creación de las interfaces y contiene otros paquetes más pequeños, separando las pantallas de las *context rules, triggering rules* y *exclusion sets*; 2) ***realmSchemas***, que contiene los esquemas de la base de datos y sus correspondientes operaciones; 3) ***em*** que se encarga de las operaciones relacionadas con la comunicación con el gestor de entorno; 4) ***siddhi*** cuya tarea es expresar las reglas definidas mediante la interfaz gráfica en reglas con la sintaxis de Siddhi; 5) ***exclusionSets*** que contiene la lógica necesaria para comprobar los exclusion sets; 6) ***event*** con operaciones relacionadas con el cálculo del contexto (localización, etc).

El paquete que representa el código nativo es más sencillo. Contiene los módulos necesarios para controlar y gestionar la comunicación con Siddhi y permitir la comunicación con el código Javascript. Tiene un paquete en su interior (Siddhi) que encapsula el control y la comunicación directa con el motor siddhi.

El paquete de Javascript utiliza el módulo *SiddhiClienteModule* para poder invocar operaciones en código nativo.

El símbolo * en la Figura B.12 indica que el paquete de *context rules* dentro del de *screens* en el de *Javascript*, contiene un módulo de crear, editar/ver para cada uno de los cuatro tipos de *context rules*.

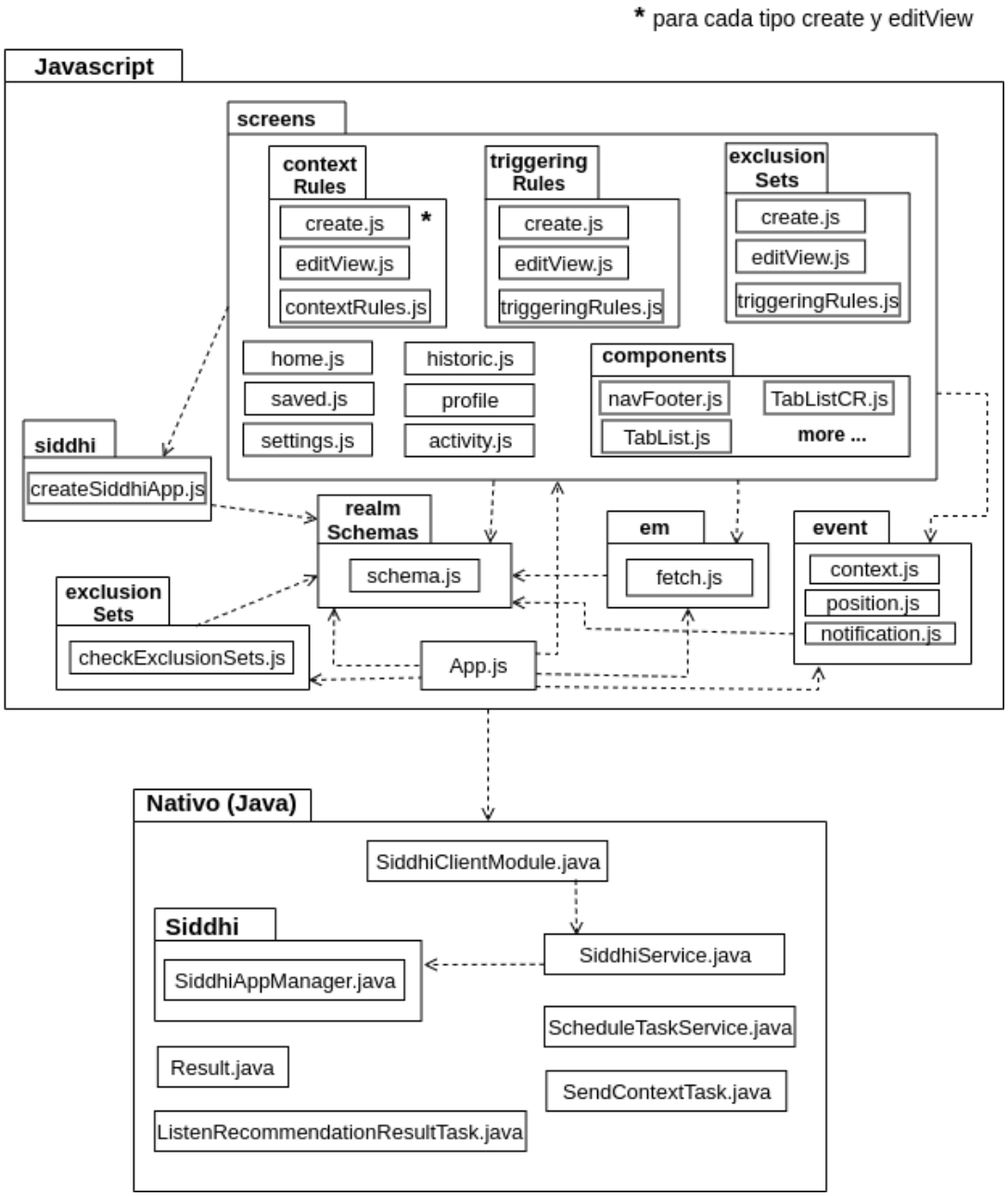


Figura B.12: Diagrama de paquetes completo de la aplicación móvil

Anexos C

Reglas definidas con la sintaxis de Siddhi

C.1. Definición de los tipos de reglas

En este capítulo presentamos la implementación de las *context rules* y las *triggering rules* con la sintaxis de Siddhi. En el caso de las primeras, incluimos también su definición cuando el usuario decide negarlas.

Time-Based context rule

Las *context rules* basadas en tiempo comprueban si la hora del contexto del usuario está entre el intervalo que se define en la pantalla de la regla. La pantalla donde el usuario define la regla puede verse en la Figura C.1 donde con una flecha se indican las entradas de texto y mediante un rectángulo los campos que deben pulsarse para que se pueda rellenar la información.

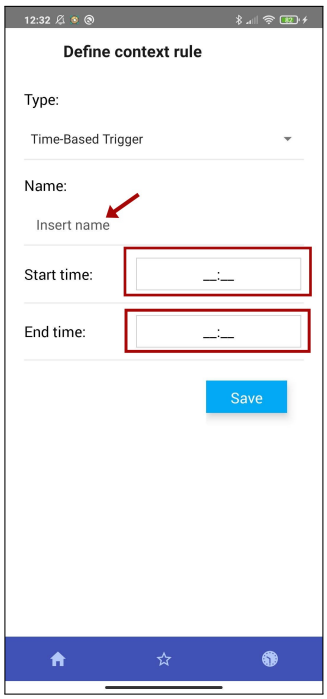


Figura C.1: Pantalla de definir regla basada la hora vacía

Con Siddhi calculamos ese tiempo en milisegundos y restamos el valor del tiempo actual y la fecha de comienzo, y el de la fecha de finalización y el tiempo actual. Comparamos esos valores con 0 para ver si está dentro del intervalo. Utilizamos una *and* para asegurarnos que la hora está en el rango definido por el usuario. En el ejemplo, incluimos una con nombre *Time4Lunch* y que define que el horario de comer del usuario es de 13:00 a 15:00.

```
@info(name='Time4LunchCR')
from UserContext[(
    (time:timestampInMilliseconds(time, 'HH:mm:ss') -
    time:timestampInMilliseconds('13:00:00', 'HH:mm:ss')) >= 0)
    and
    ((time:timestampInMilliseconds(time, 'HH:mm:ss') -
    time:timestampInMilliseconds('15:00:00', 'HH:mm:ss')) < 0)]
select contextId
insert into Time4Lunch;
```

Para negar la regla simplemente hace falta invertir los signos en las comparaciones y sustituir la *and* por una *or*.

```
@info(name='notTime4LunchCR')
from UserContext[(
    (time:timestampInMilliseconds(time, 'HH:mm:ss') -
    time:timestampInMilliseconds('13:00:00', 'HH:mm:ss')) < 0)
    or
    ((time:timestampInMilliseconds(time, 'HH:mm:ss') -
    time:timestampInMilliseconds('15:00:00', 'HH:mm:ss'))
    >= 0)]
select contextId
insert into notTime4Lunch;
```

Calendar-Based context rule

La *context rule* basada en calendario permite al usuario elegir los días de la semana que cumplen la regla y opcionalmente un periodo en el calendario (por ejemplo: del 1 de septiembre de 2021 al 1 de diciembre de 2021). El usuario define la regla en la pantalla de la Figura C.2.

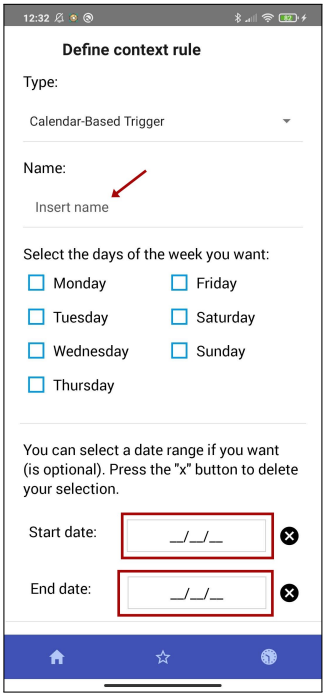


Figura C.2: Pantalla de definir regla basada en el calendario vacía

Gracias a la función de Siddhi de *dayOfWeek* obtenemos el día de la semana de la fecha del nuevo contexto y lo comprobamos con los que forman la regla. Después, si se ha incluido periodo de calendario, se comprueba si la fecha del contexto está en ese rango de tiempo utilizando *dateDiff* de Siddhi, y como en el caso anterior, lo comparamos con 0. La regla de ejemplo tiene de nombre *FreeMornings* y el usuario ha seleccionado lunes, martes o miércoles como días de la semana y del 1 de noviembre de 2021 al 31 de diciembre de 2021.

```
@info(name='FreeMorningsCR')
from UserContext[
    (time:dayOfWeek(date, 'dd/MM/yyyy') == 'Monday' or
    time:dayOfWeek(date, 'dd/MM/yyyy') == 'Tuesday' or
    time:dayOfWeek(date, 'dd/MM/yyyy') == 'Wednesday')
    and
    ((time:dateDiff(date, '01/11/2021',
        'dd/MM/yyyy', 'dd/MM/yyyy') >= 0 )
    and (time:dateDiff(date, '31/12/2021',
        'dd/MM/yyyy', 'dd/MM/yyyy') <= 0))]
select contextId
insert into FreeMornings;
```

Para negar la regla contradecimos su forma normal, asegurando que el día de la semana no es ninguno de los definidos por la regla (en este caso necesitamos *and*) y que la fecha no está en el periodo establecido, invirtiendo los signos de las comparaciones

anteriores y utilizando una *or*.

```
@info(name='notFreeMorningsCR')
from UserContext[
    (time:dayOfWeek(date, 'dd/MM/yyyy') != 'Monday' and
    time:dayOfWeek(date, 'dd/MM/yyyy') != 'Tuesday' and
    time:dayOfWeek(date, 'dd/MM/yyyy') != 'Wednesday')
    or
    ((time:dateDiff(date, '01/11/2021',
        'dd/MM/yyyy', 'dd/MM/yyyy') < 0 )
    or (time:dateDiff(date, '31/12/2021', '
        dd/MM/yyyy', 'dd/MM/yyyy') > 0))]
select contextId as contextId
insert into notFreeMornings;
```

Location context rule

El usuario crea las reglas de este tipo introduciendo un nombre y unas coordenadas de un lugar, que puede introducir de manera manual o capturar las coordenadas actuales pulsando un botón. Esta pantalla puede verse en la Figura C.3.

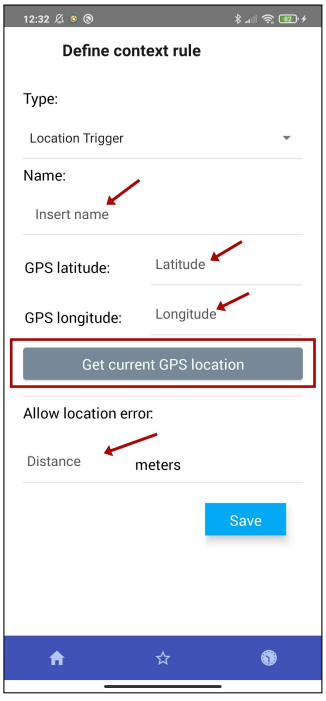


Figura C.3: Pantalla de definir regla de localización vacía

Para establecer la *context rule* relacionada con la ubicación del usuario es necesario hacer una comparación de coordenadas geográficas (latitud y longitud) en algún momento. En primer lugar intentamos utilizar una extensión de Siddhi que permitía definir funciones en JavaScript (`https://siddhi-io.github.io/siddhi-script-`

`js/`) para utilizarlo en las consultas. Esta extensión utilizaba dependencias de Java que no eran compatibles con Android y por lo tanto se descartó esta opción. Se tomó la decisión de realizar el cálculo de la distancia de las coordenadas antes de enviar el contexto a Siddhi y que la regla de Siddhi comprobara si esa distancia estaba comprendida en el límite que de error que el usuario puede elegir (por ejemplo, una distancia de 100 metros). En este caso el ejemplo representa la regla *AtHome* que comprueba si el usuario está en su casa o a una distancia de 100 metros como máximo.

```
@info(name='AtHomeCR')
from Observations[observedProperty =='Location' and
    optionalField =='AtHome'
    and
    convert(observationValue, 'int') <=100]
select contextId
insert into AtHome;
```

La negación de la regla anterior consiste en negar y cambiar el signo de comparación con la distancia máxima para que compruebe si es mayor que ésta.

```
@info(name='notAtHomeCR')
from Observations[observedProperty =='Location' and
    optionalField =='AtHome'
    and
    convert(observationValue, 'int') >100]
select contextId
insert into notAtHome;
```

Weather context rule

El usuario define la regla de tiempo introduciendo un nombre, seleccionando una serie de condiciones meteorológicas marcando una casilla y rellenando las temperaturas máximas y mínimas. En la Figura C.4 se muestra la pantalla desde donde el usuario puede completar dicha información.

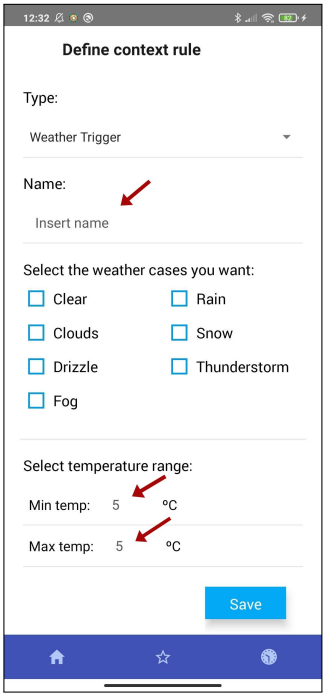


Figura C.4: Pantalla de definir regla de tiempo meteorológico vacía

A la hora de obtener el contexto, el tiempo atmosférico lo obtenemos haciendo una consulta a Open Weather (`https://openweathermap.org/api`) y seleccionando los datos que indican el estado del tiempo y la temperatura. La regla de ejemplo tiene de nombre *Good4Tourism* y el estado del tiempo puede ser despejado, nublado o con niebla. Además, la temperatura debe estar comprendida entre 7 y 35 grados ºC.

```
@info(name='Good4TourismCR')
from Observations[(observedProperty=='Weather') and
    (observationValue == 'Clear' or
    observationValue == 'Clouds' or
    observationValue == 'Fog') and
    (convert(optionalField, 'double') >=7) and
    (convert(optionalField, 'double') <=35)]
select contextId as contextId
insert into Good4Tourism;
```

Como en los ejemplos anteriores, para negar la regla hay que invertir los signos y reemplazar las *and* por *or* y viceversa.

```
@info(name='notGood4TourismCR')
from Observations[((observedProperty=='Weather') and
    (observationValue != 'Clear' and
    observationValue != 'Clouds' and
    observationValue != 'Fog')) or
```

```
    ((convert(optionalField, 'double') <7) or
    (convert(optionalField, 'double') >35))]
select contextId as contextId
insert into notGood4Tourism;
```

Triggering rule

Las *triggering rules* se definen a partir de dos o más *context rules* ya existentes (negadas o no). El usuario puede realizarlo mediante la pantalla incluida en la Figura C.5.

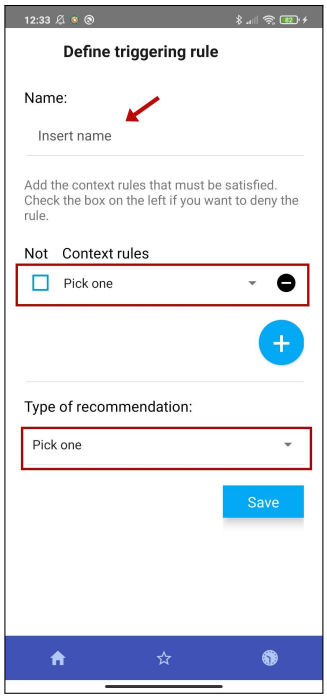


Figura C.5: Pantalla de definir *triggering rule* vacía

Para su implementación en Siddhi, cada *contextRule* emite un evento a un *stream* con su nombre (por ejemplo: notGood4Tourism en caso de una regla negada) y en la *triggering rule* se toman esos *stream* como entrada. Para esto se han utilizado los patrones que ofrece Siddhi (`https://siddhi.io/en/v4.x/docs/query-guide/#pattern`) y que permiten detectar patrones en los eventos a lo largo del tiempo. En el siguiente ejemplo observamos una *triggering rule* con nombre *LunchOutside* que consume los eventos de las *context rules* *Time4Lunch* y *AtHome* negada y activa el tipo de recomendación *restaurants*. El resultado se emite al *stream* de nombre *Resultado*.

```
@info(name='LunchOutsideTR')
from every e0 = Time4Lunch ->
```

```
    every e1 = notAtHome[e0.contextId == e1.contextId]
select e0.contextId, 'restaurants' as recommendation
insert into Results;
```

Otro ejemplo de *triggering rule* es la siguiente, cuyo nombre es *VisitPlacesOfInterest* y está formada por las *context rules* con nombre *FreeMornings*, *Time4Lunch* negada, *AtHome* y *Good4Tourism*, y activa los *placesOfInterest*.

```
@info(name='VisitPlacesOfInterestTR')
from every e0 = FreeMornings ->
    every e1 = notTime4Lunch[e0.contextId == e1.contextId] ->
    every e2 = AtHome[e0.contextId == e2.contextId] ->
    every e3 = Good4Tourism[e0.contextId == e3.contextId]
select e0.contextId, 'placesOfInterest' as recommendation
insert into Results;
```

El contexto del usuario enviado a Siddhi se divide en dos *streams* que tienen de nombre *UserContext* y *Observations*. Esta separación ha sido una decisión de diseño tomada para facilitar la definición de las reglas. Las *context* rules consumen eventos de entrada de estos *streams*: 1) *UserContext*, que contiene información como la fecha o la hora de envío del contexto, y 2) *Observations*, que contiene otros datos del contexto como el tiempo atmosférico o la localización. A Siddhi le enviamos las reglas en forma de cadena de caracteres, donde una regla está definida detrás de otra. Siddhi lanza las reglas en el orden de definición y en este caso la regla que filtra el contexto y emite eventos a *UserContext* está antes que la que emite a *Observations*. Por este motivo aquellas reglas que consumen eventos del primero sucederán antes. Las reglas basadas en la hora y en el calendario consumen de *UserContext* y las reglas que leen de *Observations* son las del tiempo atmosférico y la localización. Para implementar las *triggering rules* hemos tenido esto en cuenta, ya que los patrones de Siddhi que hemos comentado deben de ocurrir en el orden que se indica en la regla (primero e0, luego e1...). Por lo tanto, en las *triggering rules* las *context rules* se comprueban en el siguiente orden: 1) basadas en tiempo y calendario sin negar, 2) basadas en tiempo y calendario negadas, 3) de tiempo atmosférico y localización sin negar, y 4) de tiempo atmosférico y localización negadas. Además, estos subgrupos también se ordenan por orden alfabético, ya que es como se ordenan las *context rules* a la hora de definirlas para pasarlas a Siddhi.

C.2. Limitaciones de Siddhi en Android

Siddhi tiene varias extensiones que permiten ampliar las sentencias básicas de su sintaxis. Para el desarrollo del proyecto, la comparación de coordenadas geográficas es crucial, ya que es necesario para poder determinar si un usuario está en un lugar concreto o no (para las reglas de localización). Entre las extensiones de Siddhi actualmente no hay ninguna que permita realizar esta comparación de coordenadas directamente. Debido a esto, se planteó la posibilidad de utilizar la extensión *Siddhi Script JS* que permite definir funciones con JavaScript para luego utilizarse desde las reglas (consultas) de Siddhi como si fueran funciones de alguna extensión. La idea era definir una función que permitiera comparar las coordenadas actuales del usuario y que irían en su contexto, y las de la regla. Esto pudo realizarse con la herramienta de escritorio (*Siddhi Tooling*). Sin embargo, no fue posible en la aplicación debido a que algunas clases que utilizaba la extensión no eran compatibles con Android.

En el Código C.1 podemos ver la definición de la función que calcula la distancia entre coordenadas con la *Fórmula del Haversine* y un ejemplo de cómo se usaría con una regla de localización.

```
...

define function haversineDistance[JavaScript] return double {
    function toRad(x) {
        return x * Math.PI / 180;
    }
    var lon1 = data[0];
    var lat1 = data[1];
    var lon2 = data[2];
    var lat2 = data[3];
    var R = 6371; // km

    var x1 = lat2 - lat1;
    var dLat = toRad(x1);
    var x2 = lon2 - lon1;
    var dLon = toRad(x2)
    var a = Math.sin(dLat / 2) * Math.sin(dLat / 2) +
    Math.cos(toRad(lat1)) * Math.cos(toRad(lat2)) *
    Math.sin(dLon / 2) * Math.sin(dLon / 2);
    var c = 2 * Math.atan2(Math.sqrt(a), Math.sqrt(1 - a));
    var d = R * c;

    return d;
};
```

```
...

@info(name='AtHomeCR')
from Observations[observedProperty =='Location' and
    optionalField =='AtHome'
    and
    haversineDistance(userLong, ruleLong, userLat, ruleLat)
        <=100]
select contextId
insert into AtHome;
...
```

Código C.1: Formula de Harvesine con Javscript y consulta de Siddhi

Esto no ha sido posible por las razones mencionadas y por tanto, la solución adoptada ha sido calcular la distancia entre todas las coordenadas de las reglas definidas y la ubicación actual del usuario mediante la fórmula comentada, y enviar la distancia en el contexto de entrada a Siddhi para cada una de las reglas. El Código C.2 muestra el ejemplo de cómo se pasa la información de la distancia en el contexto a Siddhi.

```
{
    ... ,
    "Observations":[
        {
            "observedProperty":"Location",
            "optionalField":"AtHome",
            "observationValue":"2"
        },
        ...
    ]
}
```

Código C.2: Fragmento del contexto enviado a Siddhi

Anexos D

Encuesta sobre sistemas de recomendación

Después de la fase de integración de la tecnología en Android y antes de diseñar las nuevas pantallas de la aplicación relacionada con las reglas, elaboramos una encuesta para obtener información sobre qué tipo de reglas podían resultar interesantes al usuario y sobre qué tipos de actividades para recomendar esperaban. La encuesta se realizó con *Google Forms* y se envió a personas conocidas, al personal del Departamento de Informática e Ingeniería de Sistemas de la Universidad de Zaragoza y al estudiantado del Grado en Ingeniería Informática utilizando las redes sociales para que la completaran. La Figura D.1 se corresponde a la encuesta realizada. Podemos observar en la parte superior de la imagen que recibimos un total de **86 respuestas**. La encuesta consistía en una descripción breve con lo que se pedía y unos ejemplos prácticos para que después cada persona propusiese otros tres ejemplos similares y que le resultaran útiles.

En la Figura D.2 podemos observar algunos ejemplos de las respuestas obtenidas. Podemos ver que en alguna de ellas se le da bastante importancia a la localización del usuario (por ejemplo, *Si es por la mañana y estoy en la oficina, quiero que me informe de sitios cercanos para almorzar rápido*). En el ejemplo anterior, además de la localización se tiene en cuenta el momento del día que es (por la mañana). Por este motivo, y otras respuestas similares se decidió crear los tipos de *context rule* basados en tiempo (para poder establecer momentos del día) y de localización. Entre los ejemplos disponibles en la figura, otro es *Si estoy en una ciudad costera en verano y es un día soleado, quiero que me recomiende las mejores playas circundantes*, donde se le da importancia a la situación meteorológica del momento (*es un día soleado*) y también a la estación del año verano. Debido a esto y otras respuestas similares, se decidió incluir dos tipos más de *context rules* que serían el basado en calendario, para establecer periodos del año además de seleccionar los días de la semana (por ejemplo, es fin de semana) y las de tiempo atmosférico.

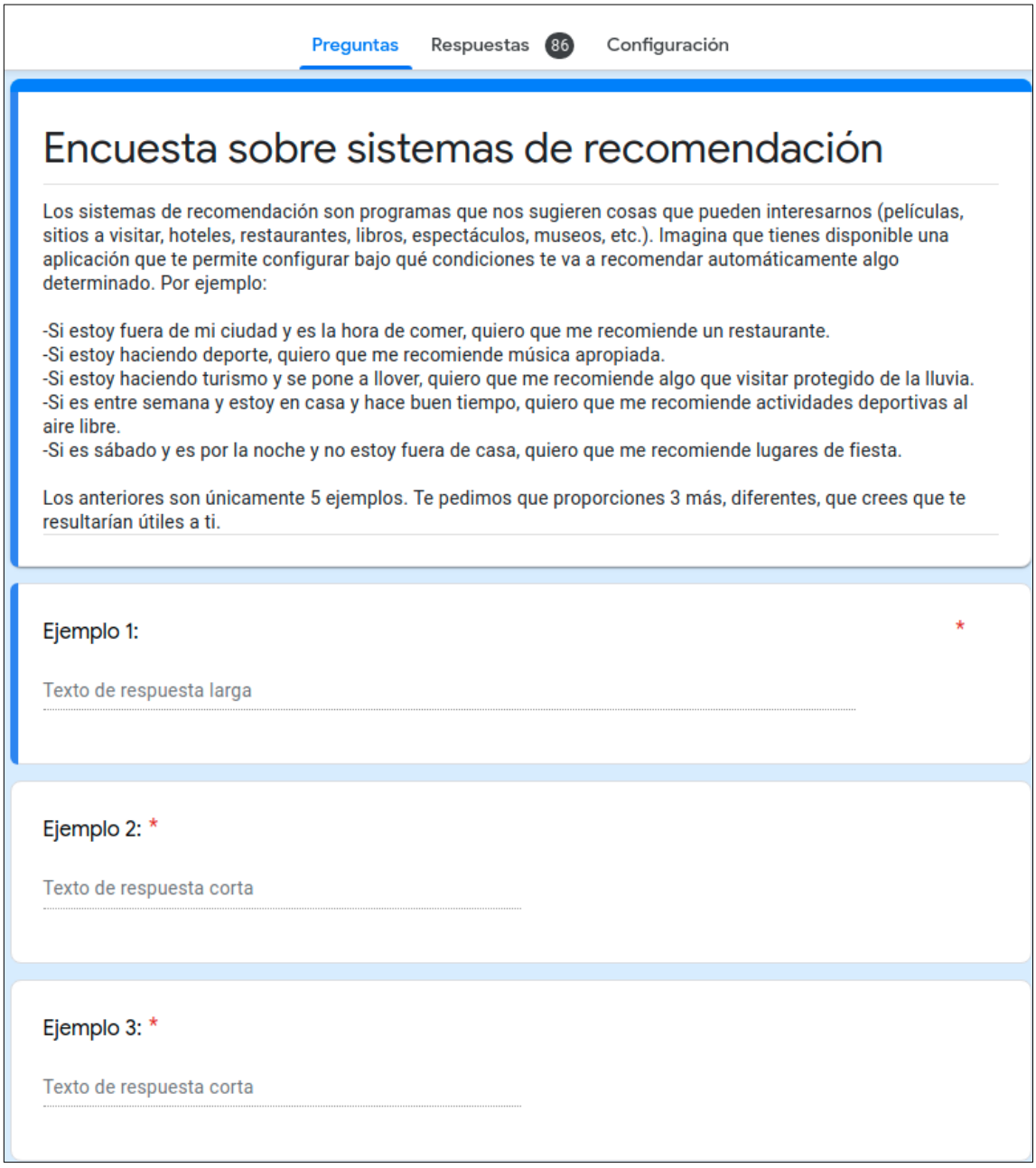


Figura D.1: Enunciado de la encuesta

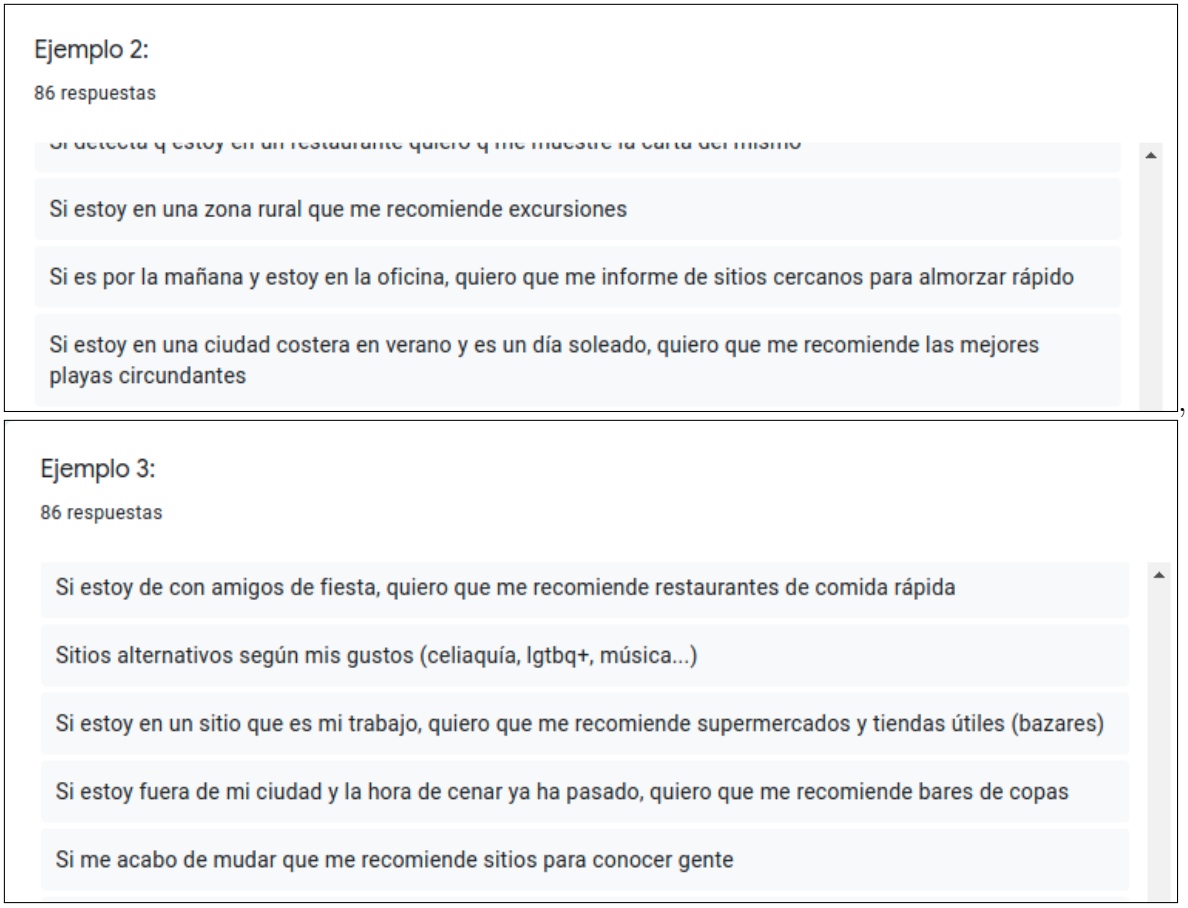


Figura D.2: Ejemplos de respuestas de la encuesta

De las 86 respuestas, una resultó ser inválida. Con esas respuestas se obtuvieron 255 ejemplos de los cuales uno estaba en blanco y otro se consideró inválido. La encuesta también nos permitió estudiar qué tipos de actividades esperaban los usuarios que se les recomendara. De las respuestas válidas obtuvimos las distintas categorías que se mencionaban y que podían ser de nuestro interés. La Tabla 5 presenta las categorías encontradas junto al número de veces que aparecen y el porcentaje total que representan en función de las personas que han respondido la encuesta de manera válida (85). A partir de esas categorías se escogieron aquellas que se consideraron más apropiadas para los datos de prueba y el escenario escogido: alojamientos, restaurantes, lugares de interés, museos, salas de espectáculos, actividades de ocio, lugares de entretenimiento y tiendas.

Categoría	**Número de apariciones**	**Porcentaje**
Rutas de senderismo/running	14	16,47 %
Lugares de interés turístico	12	14.12 %
Restaurantes	12	14.12 %
Tiendas	8	9.41 %
Bares/Pubs	8	9.41 %
Actividades de ocio y eventos	8	9.41 %
Espectáculos	7	8.24 %
Hoteles/Hostales	7	8.24 %
Playas/Calas	7	8.24 %
Cines	5	5.88 %
Museos	5	5.88 %
Rutas ciclistas	5	5.88 %
Transporte público	5	5.88 %
Actividades deportivas	5	5.88 %
Lugares tranquilos y apartados	3	3.53 %
Farmacias	3	3.53 %
Conciertos	3	3.53 %
Librerías	2	2.35 %
Piscinas	1	1.18 %
Actividades de aventura	1	1.18 %

Tabla 5: Posibles categorías de interés extraídas de la encuesta

Anexos E

Manuales del prototipo

Este capítulo incluye dos manuales de ayuda para utilizar el prototipo:el Anexo E.1 incluye el manual de instalación en Linux de la aplicación y del gestor de entorno; y el Anexo E.2 incluye el manual del usuario con algunas indicaciones útiles para facilitar su experiencia.

E.1. Manual de instalación

Aplicación

A continuación se explica cómo instalar la aplicación en un dispositivo móvil. En primer lugar, el proyecto ha sido desarrollado con React Native y el primer paso será instalar **React Native CLI** en Linux:

```
npm install -g react-native-cli
```

También es necesario tener el **SDK de Android** en el equipo desde donde se va a realizar la instalación. En equipos con Linux, añadir al fichero ***$HOME.bash_profile*** lo siguiente:

```
export ANDROID_HOME=$HOME/Library/Android/sdk
export PATH=$PATH:$ANDROID_HOME/emulator
export PATH=$PATH:$ANDROID_HOME/tools
export PATH=$PATH:$ANDROID_HOME/tools/bin
export PATH=$PATH:$ANDROID_HOME/platform-tools
```

El siguiente paso es descargar el proyecto y ejecutar las dependencias adecuadas. Para ello se recomienda la **versión 10 de Node.js** y ejecutar lo siguiente:

```
cd CARSProject
npm install
npx jetify
```

En una terminal dentro del directorio del proyecto ejecutamos lo siguiente para iniciar el empaquetador de React Native (Metro):

```
react-native start
```

Para instalar la aplicación en un dispositivo Android, instalar **Android Studio** (para el desarrollo del proyecto se ha utilizado la versión 4.1.2) y debe abrirse el proyecto CARS-Prototypes/android con la herramienta. Desde Android Studio se puede instalar y ejecutar la aplicación en el emulador de Android o en un dispositivo físico. Para este último, será necesario lo siguiente:

- El dispositivo móvil y el equipo deben tener conexión a internet.
- Ejecutar: *adb reverse tcp:8081 tcp:8081* (React-Native packager Metro).
- Ejecutar: *adb reverse tcp:8080 tcp:8080* (necesario si el gestor de entorno se ha lanzado como un servidor local que escucha en el puerto 8080).

Después de esto, la apliación debería ejecutarse correctamente en el dispositivo. Para ejecutar la aplicación móvil y poder explotar todas las funcionalidades es necesario que haya al menos un gestor de entorno activo y una base de datos con actividades. Para ello, en la siguiente sección se presenta como lanzarlos. En el archivo *ems.json* que contiene un listado con la información distintos gestores de entorno posibles, se debe incorporar la dirección del que se desea utilizar.

Gestor de entorno

El gestor de entorno está desarrollado con *Spring Boot* y puede desplegarse tanto en un servidor local como uno remoto (se deberá completar la información del fichero *ems.json* según esto). En este caso se explica cómo hacerlo de manera local. Se debe descargar el proyecto del gestor de entorno de su repositorio y tener una herramienta que permita lanzar aplicaciones web desarrolladas con Spring Boot (por ejemplo, *IntelliJ IDEA*).

En primer lugar, debemos crear la base de datos. En el proyecto se presenta la opción de desplegarla en un contenedor local con Docker y se incluye un fichero para poder hacerlo, *docker-compose.yml*. Posteriormente, se debe poblar la base de datos con las actividades, ejecutando el script cuyo nombre es *execute.sh*. Además, habrá que crear un usuario de prueba cuyos datos sean los siguientes:

```
{
  "id": 0,
  "email" : "test@gmail.com",
  "password" : "1234test5678",
```

```
  "genre": "female",
  "birth": "1999-10-20"
}
```

Una vez hecho lo anterior, ya puede ejecutarse el gestor de entorno de manera local y podrá recibir peticiones por parte de la aplicación móvil (debe comprobarse previamente que la comunicación entre la base de datos y el gestor de entorno sea buena, y lo mismo para el gestor y la aplicación).

E.2. Manual de usuario

A continuación se presenta un pequeño manual sobre cómo utilizar nuestra aplicación, especialmente la gestión de las reglas por parte del usuario. Para ello, se hace referencia a las imágenes incluidas en el Anexo B.2 y alguna nueva que se presenta en este apartado.

Al instalar la aplicación debería aparecer la pantalla de login para identificar al usuario pero, debido a los problemas que se exponen en la Sección 4.3, por ahora se inicia sesión automáticamente con un usuario de prueba, tal y cómo muestra la Figura E.1.

Mientras el usuario no defina ninguna *triggering rule*, no recibirá ningún tipo de recomendación por parte del gestor de entorno. Para poder crear una *triggering rule*, tiene que crear previamente las *context rules* que desea que formen la *triggering rule*. Para ello debe navegar a la sección de *Context Rules* mediante el menú de la aplicación (Figura B.4a). De esta manera accederá al listado de *Context Rules* y podrá añadir nuevas pulsando el botón que contiene el símbolo *+*. Puede verse en la Figura B.5c. En todas las pantallas que contengan un botón con el mismo símbolo, éste tendrá la misma utilidad (añadir un elemento). Para navegar y añadir a las *Triggering Rules* y *Exclusion Sets* se hace de la misma manera (Figuras B.4c y B.11c).

Para acceder y editar los datos tanto de las reglas como de los *exclusion sets*, en el listado hay que pulsar sobre el elemento que se desea. De esta manera se accede a la pantalla de ver su información, desde donde se puede acceder a editarla. Para eliminar algún elemento de esos tipos, hay que arrastrar el elemento en el listado de derecha a izquierda.

La creación o edición de las *context rules* es especial, ya que dependiendo del tipo

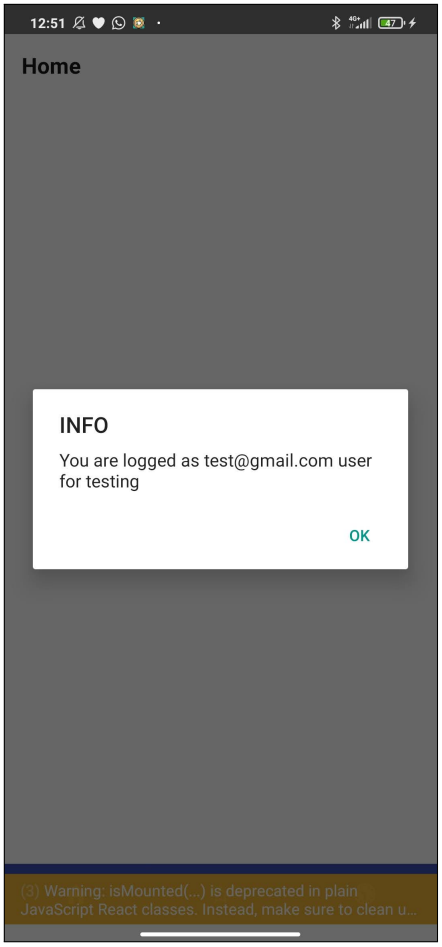
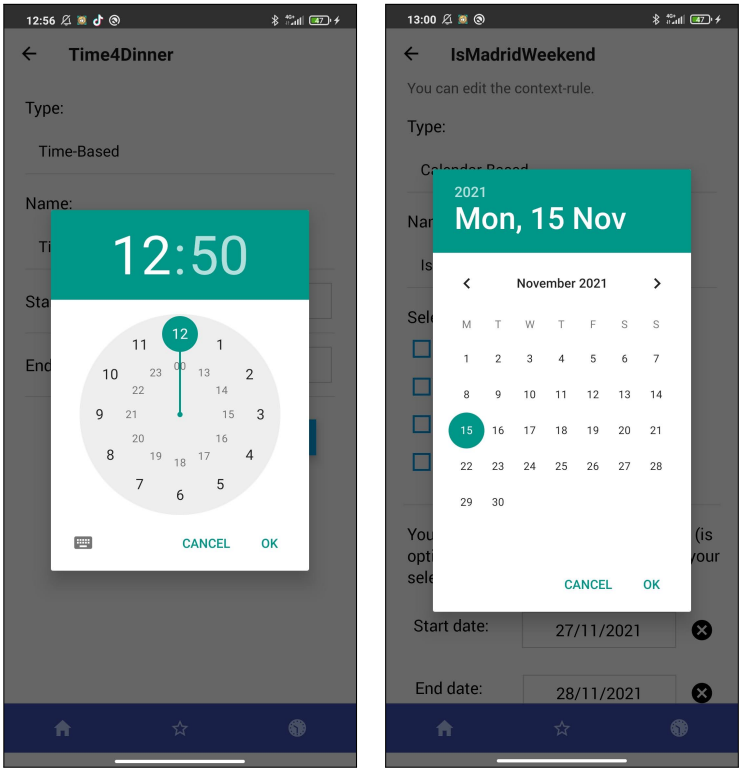


Figura E.1: Usuario de prueba identificado

que seleccionamos, aparecen unos campos a rellenar u otros. Aspectos importantes: 1) en la regla de localización pueden introducirse las coordenadas de manera manual o si se pulsa el botón *Get current GPS Location* se obtienen las coordenadas actuales usando el GPS del dispositivo; 2) en las basadas en tiempo, para introducir la hora se abre el seleccionador de horas de Android, que puede verse en la Figura E.2a; y 3) lo mismo con la fecha en las basadas en calendario, que puede verse en la Figura E.2b.

En la creación de las *Triggering Rules* hay que seleccionar un tipo de regla en cada uno de los elementos del listado. Puede verse algún ejemplo en las Figuras B.10a, B.10b y B.10c. Además, pueden eliminarse pulsando el símbolo **-** de la derecha y pueden negarse marcando la casilla de la izquierda (*checkbox*).

La creación de los *exclusion sets* es muy similar al de las *triggering rules*, pero se seleccionan tipos de recomendaciones en vez de *context rules*. El elemento que aparece primero en la lista tiene prioridad sobre el segundo, el segundo sobre el tercero, es decir, están ordenados de mayor a menor prioridad. Además, como la creación de *exclusion sets* es poco restrictiva, es posible que haya conflictos entre ellos. Por



(a) *Seleccionar hora* **(b)** *Seleccionar fecha*

Figura E.2: Pantallas con seleccionadores de fecha y hora

este motivo, aparecen en el listado por orden de prioridad (de mayor prioridad a menos) y pueden reordenarse mediante las flechas de la derecha (ver Figura B.4b). Un ejemplo de conflicto podría ser que el usuario haya definido un *exclusion set* con nombre *MuseumsBetter* formado por *Museums, PlacesOfInterest, Shops, Leisure* (en ese orden de prioridad) y otro *LeisureFirst* formado por *Leisure, Shops*. Si se pudieran activar al mismo tiempo los tipos de recomendación *Shops* y *Leisure*, no se sabría cuál de los dos debe activarse, debido a la existencia de esos dos elementos. Sin embargo, gracias a la ordenación de los exclusion sets podemos decidir cúal tiene prioridad: si *MuseumsBetter* aparece antes que *LeisureFirst*, tendrán preferencia las recomendaciones de *Shops*; si es al revés, tendrán preferencia las de *Leisure*.

Finalmente en los ajustes de la aplicación pueden seleccionarse las categorías que resultan de interés o no. Esto puede resultar interesante para indicar que no se desea que se muestren actividades de algún tipo de recomendación en algún momento. Además, en la pantalla de perfil se podrá seleccionar qué información se desea enviar al gestor de entorno y cuál no.

Anexos F

Fragmentos de código destacable

En este anexo se incluyen fragmentos del código que se consideran importantes de la implementación, a modo de ejemplo. Todos ellos están relacionados con la integración de Siddhi en el prototipo previo de Android.

Siddhi como Servicio

```
public class SiddhiService extends Service {

    private static final String TAG = "SIDDHI_SERVICE";
    private final IBinder binder = new SiddhiBinder();
    private Intent intent;
    private Boolean isStopped = true;

    // Hacemos que no sea necesario instanciar la clase
    private static SiddhiService siddhiService;
    private SiddhiAppManager siddhiAppManager =
        new SiddhiAppManager(this);

    // Binder para acceder a las operaciones del servicio
    public class SiddhiBinder extends Binder {
        SiddhiService getService() {
            // Return this instance of LocalService so clients
            // can call public methods
            return SiddhiService.this;
        }
    }

    @Override
    public IBinder onBind(Intent intent) {
        return binder;
    }

    /*
    * Metódos públicos para clientes
    */
    // Iniciar la aplicación Siddhi
    public String startSiddhiApp(String siddhiApp)
```

```
        throws RemoteException {
        if(isStopped){
            siddhiAppManager.startSiddhiApp(siddhiApp);
            isStopped = false;
            return siddhiApp;
        }else { return ""; }
    }

    // Detener la aplicacion Siddhi
    public void stopSiddhiApp(String siddhiAppName)
        throws RemoteException {
        if(!isStopped){
            isStopped = true;
            siddhiAppManager.stopSiddhiApp();
        }
    }

    // Enviar nuevo contexto
    public void sendEvent(String context){
        if(!isStopped){
            siddhiAppManager.sendEvent(context);
        }
    }

    // Obtener resultado
    public String getResult(){
        return siddhiAppManager.getResult();
    }

    // Obtener objeto resultado (compartido)
    public Result getResultObject() throws RemoteException {
        return siddhiAppManager.getResultObject();
    }

    // Comprobar si la aplicacion esta detenida
    public Boolean isStopped(){
        return isStopped;
    }

    ...
```

Código F.1: Fragmento de la clase *SiddhiService*

El Código F.1 corresponde a la implementación del servicio de Android que permite gestionar Siddhi. Es la parte correspondiente a los métodos públicos que luego pueden ser invocados para comunicarse con el servicio una vez lanzado.

Recogida del resultado del motor Siddhi

```
...

// Recibe los resultados de Siddhi: id_contexto,
// recomendacion1, recomendacion2...
siddhiAppRuntime.addCallback("finalResults",
    new QueryCallback() {

    @Override
    public void receive(long timestamp,
        Event[] inEvents, Event[] removeEvents) {

        // Obtiene el resultado
        String data = "";
        int len = inEvents[0].getData().length;
        for (int i = 0;i<len;i++){
            data = data + (String) inEvents[0].getData(i)
                + ",";
        }

        // Actualiza el resultado y lo notifica para que
        // sea recogido
        synchronized (result){
            result.setResult(data);
            result.notify();
        }

    }
});

...
```

Código F.2: Fragmento del método *startSiddhiApp* de la clase *SiddhiAppManager* correspondiente al *callback* utilizado para recoger los resultados del motor Siddhi

En el Código F.2 se muestra como se recoge el resultado calculado por Siddhi después de procesar el contexto enviado por las reglas definidas. Se añade un *callback* a una consulta, concretamente la que tiene de nombre *finalResults*, para obtener los tipos de resultados agrupados.

Recogida del resultado en SiddhiClientModule.java

```
...

private class Task implements Runnable {
        // Objeto resultado
        Result result;
        Callback callback;

        public Task(Callback callback) {
            // Guardamos el callback para luego
            this.callback = callback;
        }

        @Override
        public void run() {
            String r = "";
            try {
                // Obtenemos el objeto Result
                result = siddhiService.getResultObject();

                synchronized (result){
                    // Comprobamos el valor del resultado
                    r = result.getResult();
                    while(r.equals("")) {

                        // Esperamos a que llegue un resultado
                        result.wait();

                        // Leemos el nuevo resultado
                        r = result.getResult();
                    }
                }
            } catch (RemoteException | InterruptedException e){
                e.printStackTrace();
            }

            // Devolvemos el resultado al codigo Javascript
            callback.invoke(r);
        }
    }

...
```

Código F.3: Fragmento de la tarea que se lanza en el método *getResult()* de la clase *SiddhiClientModule*

En el Código F.3 anterior se muestra cómo se recoge el resultado calculado por

Siddhi después de procesar el contexto enviado por las reglas definidas. Se añade un *callback* a una consulta, concretamente la que tiene de nombre *finalResults*, para obtener los tipos de resultados agrupados.

SendContextTask: tarea periódica de envío de contexto

```
...

const SendContextTask = async () => {

  let user = Schemas.retrieveUser();

  // Se comprueba si existe ya un usuario identificado
  if(user != null){


    // Nos conectamos al SiddhiService
    SiddhiClientModule.connect();

    // Comprobamos si la apliacion Siddhi está detenida
    // (no hay reglas activas)
    let stopped = await isStopped("S");

    if(!stopped){

      // Código comentado: leer contexto de JSON, para pruebas
      // let context = jsonContextExample.shift();
      // Schemas.CreateContext("LOCATION",
        //JSON.stringify(context.Location));
      // Schemas.CreateContext("WEATHER",
        //JSON.stringify(context.Weather));
      // context = Event.buildSiddhiContextForTest
        //(context.UserContext);

      // Registrar localización y tiempo actual metereológico
      let pos = await myPosition._getLocation();
      Schemas.CreateContext("LOCATION", JSON.stringify(pos));
      let wea = await myPosition._getCurrentWeather(pos);

      // Obtener nuevo contexto para Siddhi
      let context = Event.buildSiddhiContext();

      if (context != null){
        context = JSON.stringify(context);
```

```
        // Enviamos el contexto a Siddhi
        SiddhiClientModule.sendEvent(context);
      }
    }
  }
}

...
```

Código F.4: Tarea de envío de contexto (Javascript) SendContextTask

En el Código F.4 se recoge y se envía periódicamente cada 30 segundos un nuevo contexto a Siddhi si el usuario ha iniciado sesión y la aplicación Siddhi está funcionando, es decir, hay *triggering rules* definidas.

ListenRecommendationResultTask: tarea de recepción de resultado

```
...

const ListenRecommendationResultTask = async () => {
    // Tiempo de espera mientras se inicia todo
    await sleep(4000);

    while(true){

      // Se comprueba si el usuario se ha identificado
      let user = Schemas.retrieveUser();
      if(user != null){

        // Comprobamos si la aplicación Siddhi no está detenida
        let stopped = await isStopped("L");
        if(!stopped){
          // Invocamos al metodo getResult de
          // SiddhiContextModule.java
          let result = await getResult();

          if(result != "start"){
            let r = result.split(",");
            let id = r.shift();

            // Comprobar los exclusion sets
            let recommendations = ExclusionSets
                .getRecommendationsWithExclusionSets(r);

            // iniciar recomendacion  con EM
            Communication.startRecommendation(recommendations);
```

```
            }
          }else{
              // Esperamos para volver a intentarlo
            await sleep(2000);
          }
        }else{
          // Si el usuario no se ha identificado esperamos
          //15 segundos y volvemos a intentar
          await sleep(15000);
        }
      }
}

...
```

Código F.5: Tarea de recepción del resultado (JavaScript) SendContextTask

En el Código F.5 anterior se espera a que se obtengan nuevos tipos de recomendación, se comprueban los *exclusion sets* y se filtran esos tipos de recomendación, y finalmente se comunican al gestor de entorno para que inicie el proceso de recomendación.

Anexos G

Diseño del gestor de entorno

En este apartado se amplía la documentación sobre el diseño del gestor de entorno. En el Anexo G.1 se introduce la API desarrollada para el gestor, y en el Anexo G.2 el esquema de la base de datos diseñado.

G.1. API del gestor de entorno

La API ha sido incorporada en el proyecto con la especificación *OpenAPI 3.0* y puede consultarse de manera visual gracias a *Swagger* una vez lanzado el proyecto. En las Figuras G.1 y G.2 puden verse las operaciones en la API de *Swagger*. Las funcionalidades del EM que se describen en las figuras abarcan lo siguiente:

- Permite gestionar las actividades.
- Permite gestionar los usuarios, incluyendo el inicio de sesión y el registro.
- Recomienda actividades según un usuario y los tipos de recomendación que deben activarse.
- Está protegida mediante el mecanismo *Basic-Auth* en el que se crea un token con el usuario y la contraseña para evitar que se acceda a sus datos.

Se han realizado algunos cambios con respecto a la propuesta inicial de la API del proyecto previo:

1. Todas las operaciones relacionadas con los usuarios se han unificado en un controlador (*Users)* cuya ruta empieza por */users*. En la propuesta anterior se tenía en dos distintos: *Session* y *Users*.
2. Se han añadido las operaciones */users/register* para completar el registro de un usuario, y */users/deleteAccount* para borrar la información de un usuario con su *token*.
3. En el controlador de actividades se ha añadido la operación */activity/store*.
4. La operación */app/context* ha sido modificada ligeramente y ahora el cuerpo de la petición debe incluir el campo *recommender* con los valores *random* o *mahout*

y un campo *categorías* con un vector con los tipos de recomendación que deben activarse.

5. Las operaciones del controlador */schema* se han omitido por ahora porque no se vio necesaria su utilidad.

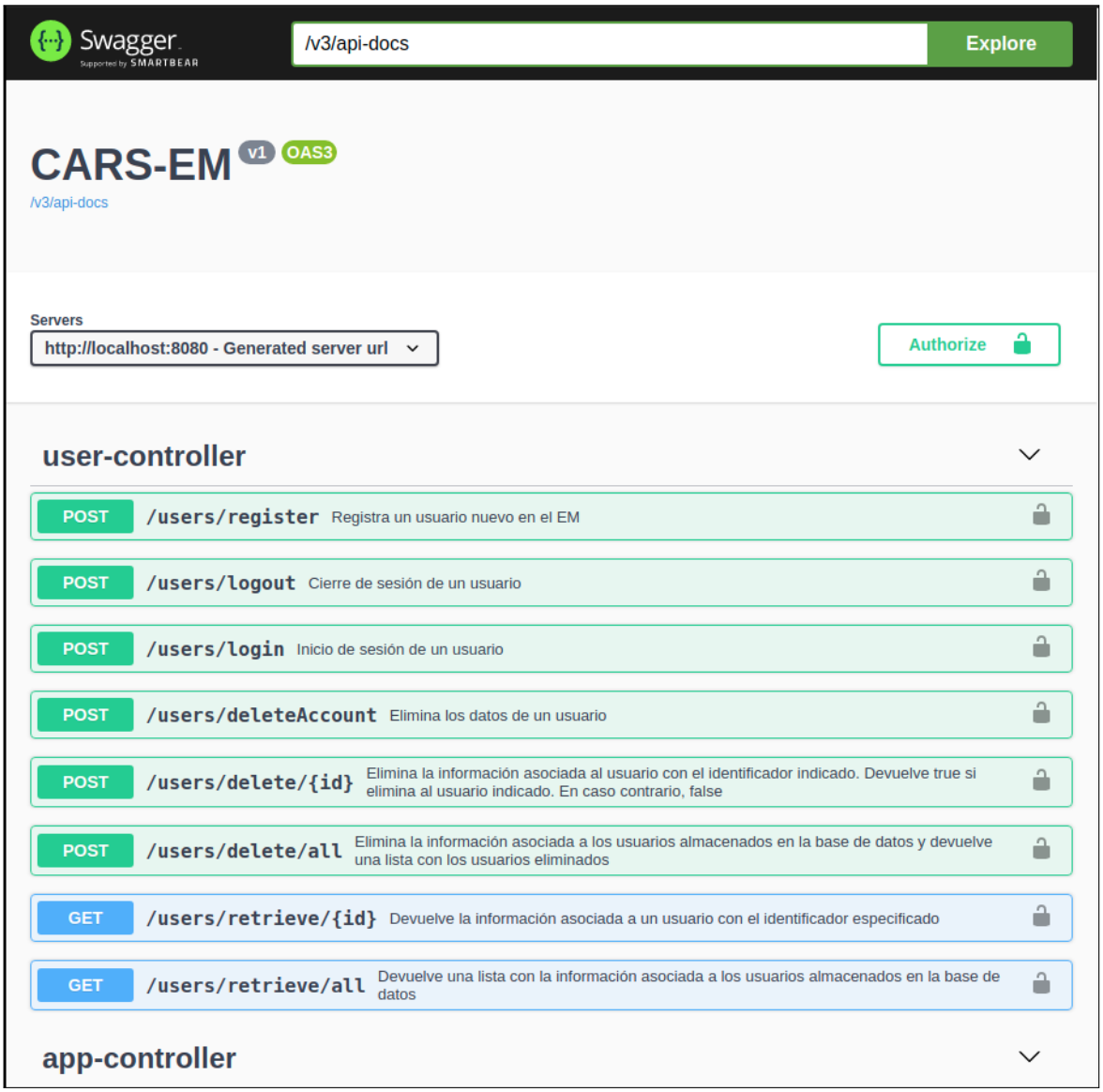


Figura G.1: Primer fragmento de la API con Swagger

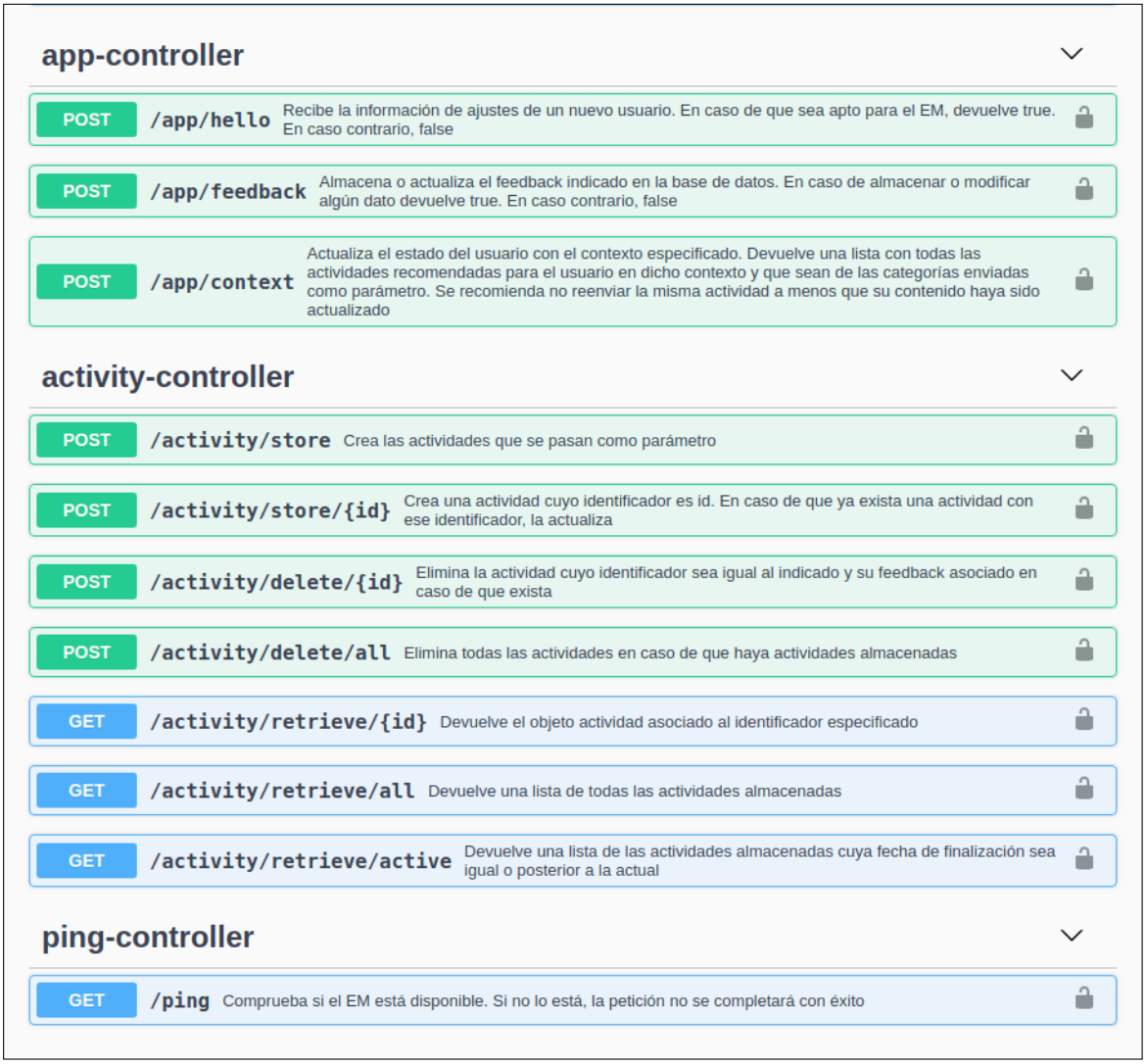


Figura G.2: Segundo fragmento de la API con Swagger

Como ejemplo, en la Figura G.3 se puede ver los detalles de una de las operaciones, la de registro, con Swagger. Incluye el formato del cuerpo de la petición y los códigos de respuesta según el resultado de la petición.

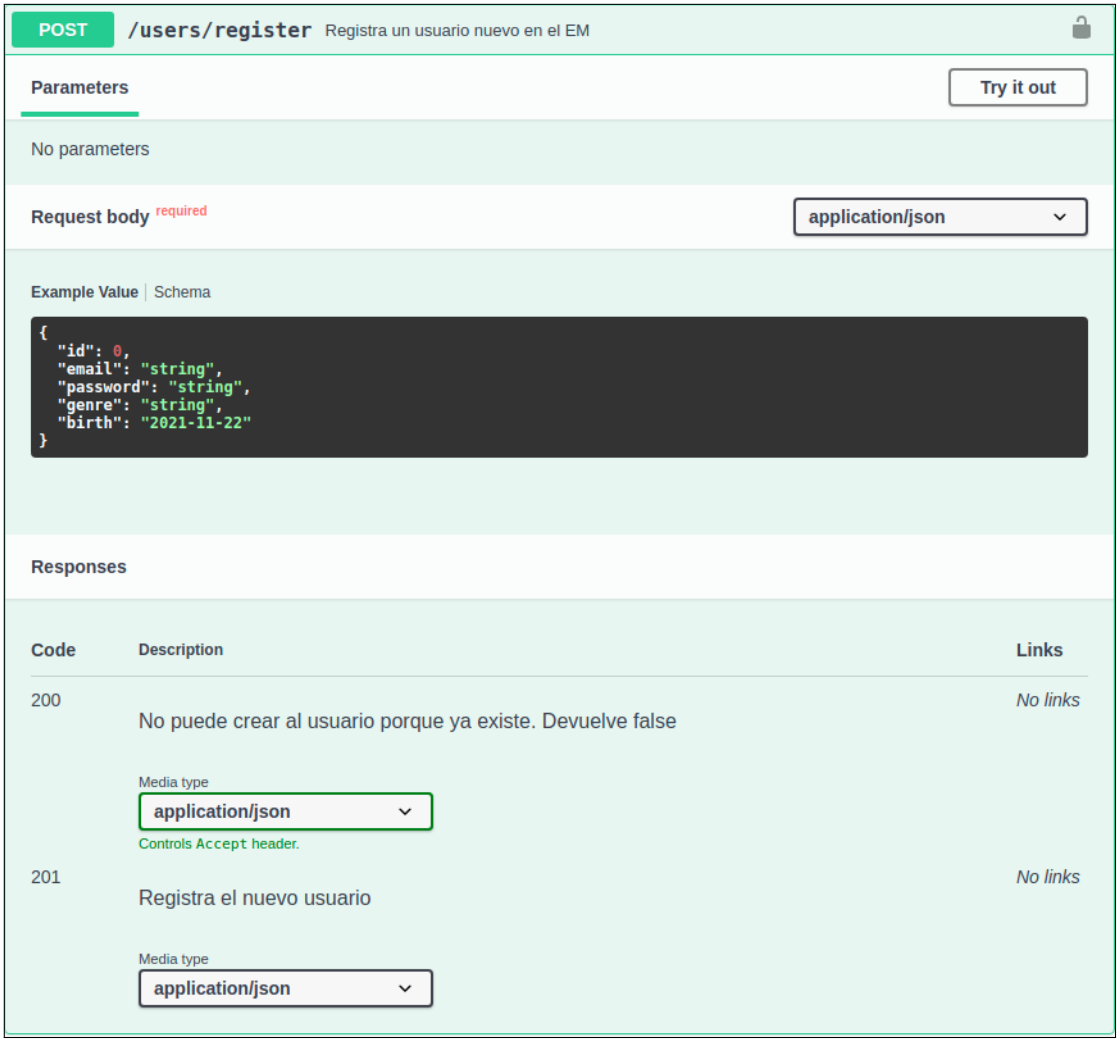


Figura G.3: Vista de la operación de registro en Swagger

A continuación presentamos la API completa descrita en un formato *YAML* y que ha sido obtenida gracias a *Swagger Editor* (`https://editor.swagger.io/`).

```yaml
openapi: 3.0.1
info:
  title: CARS-EM
  version: v1
servers:
  - url: http://localhost:8080
    description: Generated server url
security:
  - basicAuth: []
paths:
  /users/register:
    post:
      tags:
        - user-controller
      summary: Registra un usuario nuevo en el EM
      operationId: register
      requestBody:
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/User'
        required: true
      responses:
        '200':
          description: No puede crear al usuario porque ya existe.
          ↪ Devuelve false
          content:
            application/json: {}
        '201':
          description: Registra el nuevo usuario
          content:
            application/json: {}
  /users/logout:
    post:
      tags:
        - user-controller
      summary: Cierre de sesión de un usuario
      operationId: logout
      responses:
        '200':
          description: Se ha cerrado sesión correctamente
          content:
            application/json: {}
        '403':
          description: El token es inválido
```

```
        content:
          application/json: {}
  /users/login:
    post:
      tags:
        - user-controller
      summary: Inicio de sesión de un usuario
      operationId: login
      requestBody:
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/LoginRequest'
        required: true
      responses:
        '200':
          description: El inicio de sesión ha sido correcto
          content:
            application/json: {}
        '401':
          description: Login fallido. Usuario o contraseña incorrectos
          content:
            application/json: {}
        '404':
          description: No existe un usuario con ese email
          content:
            application/json: {}
  /users/deleteAccount:
    post:
      tags:
        - user-controller
      summary: Elimina los datos de un usuario
      operationId: deleteAccount
      responses:
        '200':
          description: Se ha eliminado el usuario correctamente
          content:
            application/json: {}
        '404':
          description: No puede eliminar la cuenta de un usuario
          ↪  porque no existe
          content:
            application/json: {}
  /users/delete/{id\}:
    post:
      tags:
        - user-controller
      summary: >-
```

```
92          Elimina la información asociada al usuario con el
            ↪ identificador
93          indicado. Devuelve true si elimina al usuario indicado. En
            ↪ caso
94          contrario, false
95        operationId: DeleteID
96        parameters:
97          - name: id
98            in: path
99            required: true
100           schema:
101             type: integer
102             format: int32
103       responses:
104         '200':
105           description: Devuelve true porque el usuario se ha eliminado
106           content:
107             application/json: {}
108         '404':
109           description: >-
110             El usuario especificado no existe y no se puede eliminar,
                ↪ devuelve
111             false
112           content:
113             application/json: {}
114   /users/delete/all:
115     post:
116       tags:
117         - user-controller
118       summary: >-
119         Elimina la información asociada a los usuarios almacenados en
            ↪ la base de
120         datos y devuelve una lista con los usuarios eliminados
121       operationId: DeleteAll
122       responses:
123         '200':
124           description: Devuelve una lista con todos los usuarios que
                ↪ ha eliminado,
125           content:
126             application/json: {}
127         '404':
128           description: No hay usuarios registrados
129           content:
130             application/json: {}
131   /app/hello:
132     post:
133       tags:
134         - app-controller
```

```yaml
    summary: |-
      Recibe la información de ajustes de un nuevo usuario. En
      caso de que sea apto para el EM, devuelve true. En caso
      ↪ contrario, false
    operationId: hello
    requestBody:
      content:
        application/json:
          schema:
            $ref: '#/components/schemas/AppHelloRequest'
      required: true
    responses:
      '200':
        description: El nuevo usuario es apto para el EM
        content:
          application/json: {}
      '400':
        description: EL body debe contar con un identificador de
        ↪ usuario y Settings
        content:
          application/json: {}
      '403':
        description: >-
          El token es inválido o el usuario no tiene acceso a esa
          ↪ información
          (token no corresponde a user)
        content:
          application/json: {}
      '404':
        description: No existe un usuario con ese identificador
        content:
          application/json: {}
  /app/feedback:
    post:
      tags:
        - app-controller
      summary: >-
        Almacena o actualiza el feedback indicado en la base de datos.
        En caso de almacenar o modificar algún dato devuelve true. En
        ↪ caso
        contrario, false
      operationId: feedback
      requestBody:
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/FeedbackRequest'
        required: true
```

```yaml
    responses:
      '200':
        description: Se almacena el feedback y el contexto de la actividad y el usuario
        content:
          application/json: {}
      '403':
        description: >-
          El token es inválido o el usuario no tiene acceso a esa información
          (token no corresponde a user)
        content:
          application/json: {}
      '404':
        description: La actividad y el usuario deben existir en el EM
        content:
          application/json: {}
/app/context:
  post:
    tags:
      - app-controller
    summary: >-
      Actualiza el estado del usuario con el contexto especificado. Devuelve
      una lista con todas las actividades recomendadas para el usuario en
      dicho contexto y que sean de las categorías enviadas como parámetro. Se
      recomienda no reenviar la misma actividad a menos que su contenido haya
      sido actualizado
    operationId: context
    requestBody:
      content:
        application/json:
          schema:
            $ref: '#/components/schemas/AppContextRequest'
      required: true
    responses:
      '200':
        description: >-
          Se actualiza el estado del usuario y se devuelve la lista de
          actividades que no han sido enviadas
        content:
          application/json: {}
      '403':
```

```
219           description: >-
220             El token es inválido o el usuario no tiene acceso a esa
                ↪ información
221             (token no corresponde a user)
222           content:
223             application/json: {}
224         '404':
225           description: >-
226             No existe un usuario con ese identificador o no existe ese
227             recomendador
228           content:
229             application/json: {}
230   /activity/store:
231     post:
232       tags:
233         - activity-controller
234       summary: Crea las actividades que se pasan como parámetro
235       operationId: Store
236       requestBody:
237         content:
238           application/json:
239             schema:
240               $ref: '#/components/schemas/ActivitiesRequest'
241         required: true
242       responses:
243         '200':
244           description: Devuelve la actividades creadas/actualizadas
245           content:
246             application/json: {}
247         '404':
248           description: El array de actividades no puede ser nulo
249           content:
250             application/json: {}
251   /activity/store/{id}:
252     post:
253       tags:
254         - activity-controller
255       summary: >-
256         Crea una actividad cuyo identificador es id. En caso de que ya
            ↪ exista
257         una actividad con ese identificador, la actualiza
258       operationId: StoreID
259       parameters:
260         - name: id
261           in: path
262           required: true
263           schema:
264             type: string
```

```
265       requestBody:
266         content:
267           application/json:
268             schema:
269               $ref: '#/components/schemas/Activity'
270         required: true
271       responses:
272         '200':
273           description: Devuelve la actividad creada/actualizada
274           content:
275             application/json: {}
276         '404':
277           description: El id de la actividad y id no coinciden
278           content:
279             application/json: {}
280   /activity/delete/{id}:
281     post:
282       tags:
283         - activity-controller
284       summary: >-
285         Elimina la actividad cuyo identificador sea igual al indicado
            ↪  y su
286         feedback asociado en caso de que exista
287       operationId: DeleteID_1
288       parameters:
289         - name: id
290           in: path
291           required: true
292           schema:
293             type: string
294       responses:
295         '200':
296           description: La actividad se ha borrado con éxito
297           content:
298             application/json: {}
299         '404':
300           description: No existe una actividad con ese id
301           content:
302             application/json: {}
303   /activity/delete/all:
304     post:
305       tags:
306         - activity-controller
307       summary: >-
308         Elimina todas las actividades en caso de que haya actividades
309         almacenadas
310       operationId: DeleteAll_1
311       responses:
```

```
312         '200':
313           description: Se han eliminado todas las actividades
              ↪ almacenadas
314           content:
315             application/json: {}
316         '404':
317           description: No hay ninguna actividad almacenada
318           content:
319             application/json: {}
320   /users/retrieve/{id}:
321     get:
322       tags:
323         - user-controller
324       summary: >-
325         Devuelve la información asociada a un usuario con el
            ↪ identificador especificado
326       operationId: retrieveID
327       parameters:
328         - name: id
329           in: path
330           required: true
331           schema:
332             type: integer
333             format: int32
334       responses:
335         '200':
336           description: Devuelve el usuario especificado
337           content:
338             application/json: {}
339         '403':
340           description: >-
341             El usuario especificado no coincide con el token y no
                ↪ tiene acceso a
342             los datos
343           content:
344             application/json: {}
345         '404':
346           description: El usuario especificado no existe
347           content:
348             application/json: {}
349   /users/retrieve/all:
350     get:
351       tags:
352         - user-controller
353       summary: >-
354         Devuelve una lista con la información asociada a los usuarios
355         almacenados en la base de datos
356       operationId: retrieveAll
```

```yaml
    responses:
      '200':
        description: Devuelve una lista con todos los usuarios
        content:
          application/json: {}
      '404':
        description: No hay usuarios registrados
        content:
          application/json: {}
/ping:
  get:
    tags:
      - ping-controller
    summary: >-
      Comprueba si el EM está disponible. Si no lo está, la petición no se
      completará con éxito
    operationId: ping
    responses:
      '200':
        description: El EM está disponible
        content:
          '*/*':
            schema:
              type: object
/activity/retrieve/{id}:
  get:
    tags:
      - activity-controller
    summary: Devuelve el objeto actividad asociado al identificador especificado
    operationId: retrieveID_1
    parameters:
      - name: id
        in: path
        required: true
        schema:
          type: string
    responses:
      '200':
        description: Devuelve la actividad correspondiente al id proporcionado
        content:
          application/json: {}
      '404':
        description: No existe ninguna actividad con ese id
        content:
          application/json: {}
```

```yaml
  /activity/retrieve/all:
    get:
      tags:
        - activity-controller
      summary: Devuelve una lista de todas las actividades almacenadas
      operationId: retrieveAll_1
      responses:
        '200':
          description: Devuelve la lista de las actividades
            ↪ almacenadas
          content:
            application/json: {}
        '404':
          description: No existen actividades almacenadas en este
            ↪ momento
          content:
            application/json: {}
  /activity/retrieve/active:
    get:
      tags:
        - activity-controller
      summary: >-
        Devuelve una lista de las actividades almacenadas cuya fecha
          ↪ de
        finalización sea igual o posterior a la actual
      operationId: retrieveActive
      responses:
        '200':
          description: >-
            Devuelve la lista de las actividades almacenadas con fecha
              ↪ igual o
            posterior a la actual
          content:
            application/json: {}
        '404':
          description: No existen actividades activas en este momento
          content:
            application/json: {}
components:
  schemas:
    User:
      type: object
      properties:
        id:
          type: integer
          format: int32
        email:
          type: string
```

```yaml
        password:
          type: string
        genre:
          type: string
        birth:
          type: string
          format: date
    LoginRequest:
      type: object
      properties:
        email:
          type: string
        password:
          type: string
    AppHelloRequest:
      type: object
      properties:
        user:
          type: integer
          format: int32
        settings:
          $ref: '#/components/schemas/Settings'
    Location:
      type: object
      properties:
        setting:
          type: boolean
        accuracy:
          type: boolean
    Settings:
      type: object
      properties:
        user:
          type: boolean
        location:
          $ref: '#/components/schemas/Location'
        weather:
          type: boolean
        calendar:
          type: boolean
    Activity:
      type: object
      properties:
        id:
          type: string
        authorId:
          type: string
        author:
```

```
          type: string
        title:
          type: string
        type:
          type: string
        description:
          type: string
        img:
          type: string
        begin:
          type: string
          format: date-time
        ending:
          type: string
          format: date-time
        latitude:
          type: number
          format: double
        longitude:
          type: number
          format: double
        subcategories:
          type: string
        category:
          type: string
          writeOnly: true
    Context:
      type: object
      properties:
        idcontext:
          type: integer
          format: int32
        timestamp:
          type: string
          format: date-time
        location:
          $ref: '#/components/schemas/Location'
        weather:
          $ref: '#/components/schemas/Weather'
        events:
          type: array
          items:
            $ref: '#/components/schemas/Event'
    Event:
      type: object
      properties:
        idevent:
          type: integer
```

```
          format: int32
        description:
          type: string
        calendar:
          type: string
        title:
          type: string
        startDate:
          type: string
          format: date-time
        endDate:
          type: string
          format: date-time
        allDay:
          type: boolean
        location:
          type: string
        availability:
          type: string
        context:
          $ref: '#/components/schemas/Context'
    FeedbackRequest:
      type: object
      properties:
        context:
          $ref: '#/components/schemas/Context'
        user:
          $ref: '#/components/schemas/User'
        activity:
          $ref: '#/components/schemas/Activity'
        clicked:
          type: boolean
        saved:
          type: boolean
        discarded:
          type: boolean
        rate:
          type: integer
          format: int32
    Weather:
      type: object
      properties:
        idweather:
          type: integer
          format: int32
        temp:
          type: number
          format: double
```

```
        pressure:
          type: integer
          format: int32
        humidity:
          type: integer
          format: int32
        temp_min:
          type: number
          format: double
        temp_max:
          type: number
          format: double
        description:
          type: string
        wind:
          type: number
          format: double
        clouds:
          type: integer
          format: int32
        context:
          $ref: '#/components/schemas/Context'
    AppContextRequest:
      type: object
      properties:
        context:
          $ref: '#/components/schemas/Context'
        categories:
          type: array
          items:
            type: string
        user:
          $ref: '#/components/schemas/User'
        recommender:
          type: string
    ActivitiesRequest:
      type: object
      properties:
        activities:
          type: array
          items:
            $ref: '#/components/schemas/Activity'
  securitySchemes:
    basicAuth:
      type: http
      scheme: basic
```

G.2. Esquema de la base de datos del gestor de entorno

En este apartado se incluye de manera resumida el esquema de la base de datos del gestor de entorno. El esquema presentado es el mismo desarrollado en el proyecto [1], con pequeñas modificaciones. Se ha añadido un nuevo atributo a la tabla *Activity* con nombre *subcategories* para incorporar las subcategorías a las que pertenece cada actividad. La incorporación de este campo ha simplificado la ejecución de las pruebas. Los demás cambios realizados tienen que ver con algún tipo de dato, pero no son cambios realmente relevantes. En la Figura G.4 se incluye el esquema.

Se ha mantenido la propuesta de esquema del prototipo anterior porque de este modo apenas había que modificar nada en el prototipo móvil con respecto a las actividades, y también se facilita el trabajo si en un futuro se retoma el proyecto. El usuario que se planteaba como administrador en la propuesta anterior se ha incluido en la base de datos aunque ahora no se le ha dado especial importancia (se ha trabajado en todo momento con usuarios no administradores).

G.3. Tipos de recomendadores de Apache Mahout

Para la implementación del gestor hemos tenido que escoger entre uno de los tipos de algoritmos de recomendación tradicionales. Las distintos tipos de algoritmos están explicados en [14] y son:

- **Filtrado basado en contenido.** Trabajan con perfiles de usuarios y sus gustos con respecto a los items disponibles. Se comparan aquellos elementos que han sido valorados por el usuario con aquellos que todavía no, y busca las similitudes. Existen varias aproximaciones para calcular la similitud entre los items.
- **Filtrado colaborativo.** Trabaja con las opiniones de otros usuarios del sistema. Si dos usuarios proporcionan las mismas valoraciones (o similares) a un conjunto de elementos, tienen los mismos gustos. Un usuario recibe recomendaciones de actividades que no ha valorado todavía pero sí que lo han sido por otros usuarios. Existen varios tipos:
  - **Usuario-usuario**. Su idea principal es que usuarios con valoraciones similares se comportarán de manera parecida en el futuro. Necesitan calcular la similaridad entre usuarios.
  - **Ítem-ítem**. Busca la similitud de los ítems teniendo en cuenta las

valoraciones de los ítems por parte de los usuarios. A diferencia del anterior, no calcula una similitud entre usuarios, sino que busca la similitud entre los ítems.

- **Híbridos**. Combinan las ventajas de los dos anteriores.

En este proyecto se ha escogido un algoritmo de filtrado colaborativo de tipo ítem-ítem debido a que era el que mejor se adaptaba a nuestras necesidades.

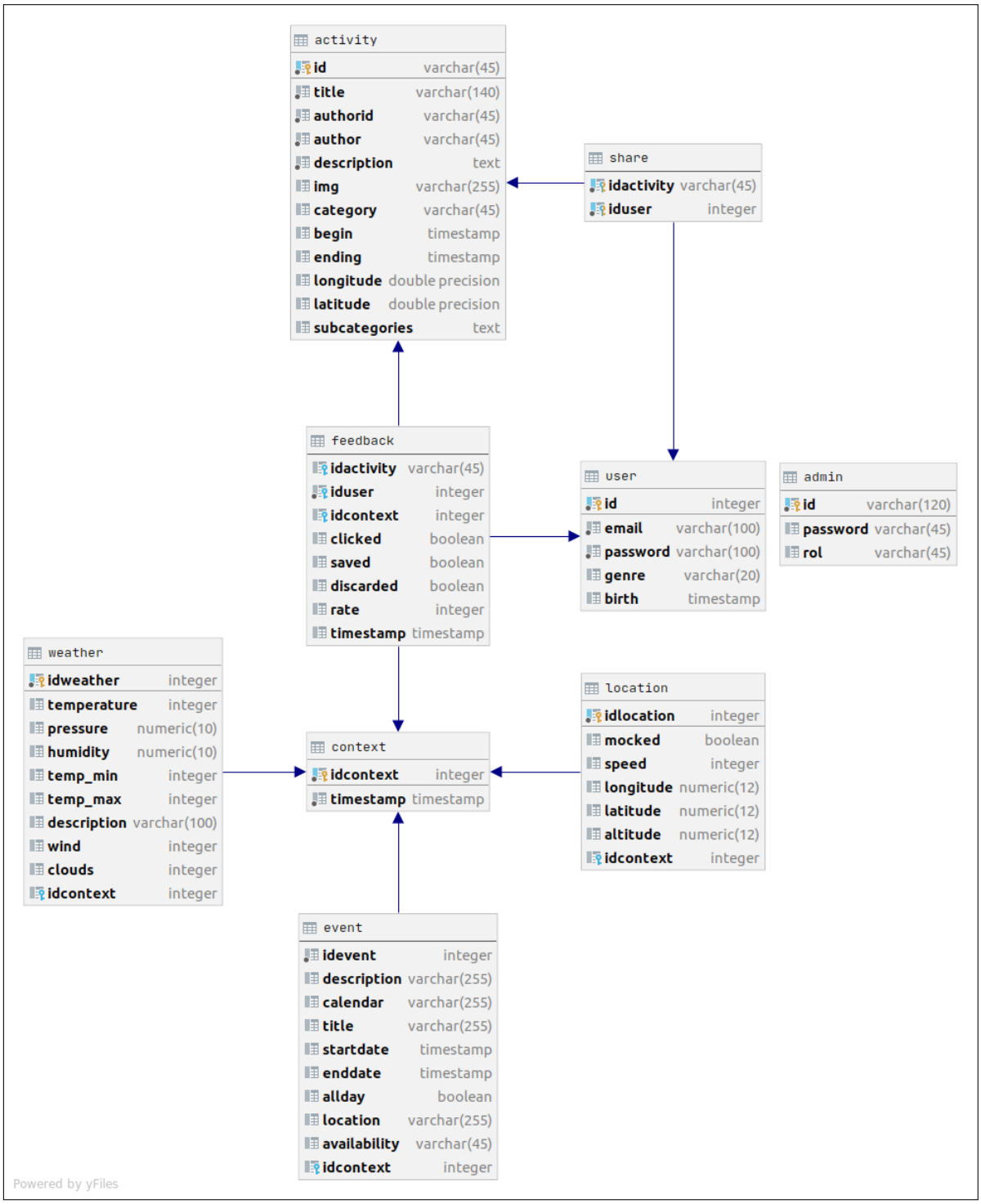


Figura G.4: Esquema relacional de la base de datos del EM (diseñado con *DataGrip*)

Anexos H

Documentación de las pruebas

En el Anexo H.1 se presenta la descripción detallada del escenario de prueba. En el Anexo H.2 las diferentes fuentes desde las que se han extraído los datos de prueba. En el Anexo H.3 se comentan algunos resultados de ejemplo de manera detallada.

H.1. Descripción completa del escenario

A continuación presentamos la descripción completa del escenario ficticio creado para elaborar las pruebas del proyecto.
Alicia está organizando un viaje de vacaciones para los días 26, 27 y 28 de noviembre de 2021. El destino que ha escogido ha sido Madrid, ciudad en la que ya ha estado otras veces. Se hospeda en un apartamento en Malasaña y espera hacer turismo y conocer sitios de alrededor, aunque no le importa coger el transporte para acudir a sitios más lejanos. Alicia va a utilizar nuestra aplicación para obtener recomendaciones sobre qué hacer en Madrid según su contexto y las context-rules, triggering-rules y exclusion sets que defina.

Algunas actividades de Madrid ya fueron valoradas por Alicia en sus viajes anteriores. Concretamente sus gustos según las categorías disponibles en la aplicación son: 1) veganos, vegetarianos, tapas y tabernas, de ***restaurantes****; 2) edificios y monumentos y escuelas de cocina y catas de vinos y aceites, de* ***lugares de interés****; 3) coctelerías y música en directo, de* ***lugares de entretenimiento****; 4) artesanía y regalo, hogar y decoración, de* ***tiendas****; 5) hostales y apartahoteles, de* ***alojamientos****; 6) museos, de* ***museos*** *(incluye otras subcategorías como planetarios o fundaciones); 7) auditorios y salas de conciertos, de* ***edificios para espectáculos****; y 8) actividades de cine y audiovisuales, circo y magia y baile, de* ***ocio****.*

Las context rules de tipo ***localización*** *definidas por Alicia son: 1)* ***AtApartment****, introduce las coordenadas del apartamento;* ***AtChueca****, donde introduce unas coordenadas del barrio de Chueca; y 3)* ***AtPalacioReal****, donde introduce las coordenadas del Palacio Real de Madrid. Las reglas* ***basadas en la hora*** *son: 1)* ***Time4Lunch****, su horario de comer es de 13:00 a 15:00; 2)* ***Time4Dinner****, su horario de cenar es de 21:00 a 22:30; 3)* ***Time4Party****, su horario de salir de fiesta es de*

22:00 a 23:59; y 4) ***InTheMorning****, de 9:30 a 14:00. Las reglas definidas* ***basadas en calendario*** *son: 1)* ***IsMadridWeekend****, donde ha seleccionado el sábado 27 y el domingo 28 que está de viaje; y 2)* ***ShowDay****, siendo el sábado el día que le gustaría ir a algún espectáculo. Finalmente, las reglas de* ***tiempo atmosférico*** *son: 1)* ***Good4Tourism****, si está nublado, despejado o con niebla y la temperatura está entre los 8 y los 30 ºC, considera que es adecuado hacer turismo; y 2)* ***Good4Museums****, si llueve, diluvia o hay tormenta y la temperatura está entre los 0 y los 15ºC, considera que prefiere visitar museos.*

Las triggering rules que ha definido han sido:

1. ***LunchOutside*** *que activa la recomendación de* ***restaurantes*** *si se cumplen las reglas AtApartment y Time4Lunch.*
2. ***GoToShow*** *que activa la recomendación de* ***edificios de espectáculos*** *y depende de las reglas AtApartment, Time4Lunch negada, Time4Dinner negada y GoToShow.*
3. ***GoShopping*** *que activa* ***tiendas*** *y depende de las reglas AtChueca, isMadridWeekend y Good4Tourism negada.*
4. ***VisitMuseums*** *activa la recomendación de* ***museos*** *y está formada por Good4Museums, Time4Lunch negada y InTheMorning.*
5. ***VisitPlacesOfInterest*** *que activa* ***lugares de interés*** *y depende de Good4Tourism y AtPalacioReal.*
6. ***GoParty*** *que activa* ***lugares de entretenimiento*** *y está formada por Time4Party y AtChueca.*
7. ***LeisureActivities*** *que activa actividades de* ***ocio*** *y depende de Time4Lunch negada y Time4Dinner negada.*

Algunas de esas reglas, como por ejemplo la que activa actividades de ocio, las desactivará puntualmente. Además, el domingo editará la regla que activa museos para que ya no dependa de la regla de Good4Museums.

Finalmente Alicia crea varios exclusion sets: 1) ***FirstEatThenTourism*** *que prioriza las recomendaciones de restaurantes frente a las de lugares de interés; 2)* ***MuseumsBetter****, que prioriza las de Museums, después las de lugares de interés y luego las de tiendas; 3)* ***PlacesOfInterestBetter*** *que prioriza los lugares de interés frente a las actividades de ocio; y 4)* ***ShowsBetter*** *que activará el sábado y que prioriza los espectáculos con respecto a las actividades de ocio y este en el listado lo situará por encima del anterior (le da prioridad).*

En el Anexo B.2 pueden consultarse algunos de los ejemplos anteriores definidos mediante las pantallas de la aplicación.

H.2. Obtención de los datos de actividades de Madrid

En la Tabla 6 se presentan las distintas fuentes de datos de las que se han extraído los datos de cada una de las categorías de actividades para recomendar del proyecto.

Categoría	Nombre del conjunto de datos	Fuente	Formato
Restaurantes	Restaurantes con perfil turístico de la ciudad de Madrid	[15]	XML
Alojamientos	Alojamientos de la ciudad de Madrid	[16]	XML
Tiendas	Tiendas, comercios y mercados con perfil turístico de la ciudad de Madrid	[17]	XML
Lugares de entretenimiento	Locales de diversión y entretenimiento con perfil turístico de la ciudad de Madrid	[18]	XML
Lugares de interés	Puntos de Interés turístico de la ciudad de Madrid	[19]	XML
Museos	Museos de la ciudad de Madrid	[20]	XML
Salas de espectáculos	Salas de espectáculos artísticos: teatros, cines, filmotecas, auditorios y salas de conciertos	[21]	JSON (from API)
Ocio	Actividades Culturales y de Ocio Municipal en los próximos 100 días	[21]	JSON (from API)

Tabla 6: Conjuntos de datos utilizados por categoría

H.3. Información detallada de algunos ejemplos de resultados

Durante la ejecución del escenario se registraron los siguientes eventos: el contexto leído del JSON, el enviado a Siddhi, las *triggering rules* activas, tipos activados por Siddhi, tipos activados por los *exclusion sets* y items recomendados por el gestor. A continuación mostramos algún ejemplo interesante. En la Figura H.1 aparecen tres contextos consecutivos de prueba, y vamos a analizar sus resultados.

En el escenario, antes del procesamiento del contexto 5, la *triggering rule Leisure Activities* ha sido desactivada por el usuario y por lo tanto no deben recomendarse actividades de ocio. Antes del procesamiento del contexto 6, esta regla será activada de nuevo por el usuario. Los resultados obtenidos para estos 3 contextos podemos verlos en la Figura H.2.

Algo destacable sobre el resultado del **contexto 5** (Figura H.2a) es que con ese contexto y las reglas definidas lo esperado es que no recomiende ningún tipo de

(a) *Contexto 5* **(b)** *Contexto 6* **(c)** *Contexto 7*

Figura H.1: Ejemplos de contextos de prueba

actividad. Esto es lo que ocurre, ya que puede comprobarse que inmediatamente después de mostrar las *triggering rules activas*, comienza el registro del resultado del siguiente contexto. Del resultado del **contexto 6** (Figura H.2b) destaca que en las *triggering rules* activas aparece la *LeisureActivities*, a diferencia del anterior, ya que la regla antes estaba desactivada. Además, los *exclusion sets* no influyen en las recomendaciones que Siddhi ha decidido que hay que activar (*leisure*). Como respuesta del gestor de entorno se han obtenido 3 nuevos ítems. Del resultado del **contexto 7** hay que destacar la diferencia entre el resultado de Siddhi y el de los *exclusion sets*. Siddhi ha decidido que con ese contexto hay que activar las recomendaciones de *Leisure, PlacesOfInterest*; sin embargo, el *exclusion set PlacesOfInterestBetter* prioriza los lugares de interés frente a las actividades de ocio. Por esto al gestor de entorno sólo se le piden actividades de este tipo. En los ítems recomendados, observamos que sólo hay del tipo *PlacesOfInterest*.

```
LOG: 1- Contexto de prueba
LOG: {"UserContext":{"contextId":"5","date":"26/11/2021","time":"15:05:00"},"Weather": ...
LOG: 2-Contexto enviado a Siddhi
LOG: {"UserContext":{"contextId":"5","date":"26/11/2021","time":"15:05:00"},"Observations": ...
LOG: 3-Triggering rules activas
LOG: GoParty,GoShopping,GoToShow,LunchOutside,VisitMuseums,VisitPlacesOfInterest
LOG: 1- Contexto de prueba
```

(a) *Resultado contexto 5*

```
LOG: 1- Contexto de prueba
LOG: {"UserContext":{"contextId":"6","date":"26/11/2021","time":"15:45:00"}, "Weather": ...
LOG: 2-Contexto enviado a Siddhi
LOG: {"UserContext":{"contextId":"6","date":"26/11/2021","time":"15:45:00"},"Observations": ...
LOG: 3-Triggering rules activas
LOG: GoParty,GoShopping,GoToShow,LeisureActivities,LunchOutside,VisitMuseums,VisitPlacesOfInterest
LOG: 4- Tipos de recomendación activadas por Siddhi para el contexto: 6
LOG: leisure,
LOG: 5- Tipos de recomendación activadas filtradas por los EXCLUSION SETS:
LOG: ,leisure
LOG: 6- Items recomendados por el gestor de entorno:
LOG: Number of recommended items: 3
LOG: 419; Frankie &amp;amp, Johnnie; Leisure; CineActividadesAudiovisuales
LOG: 973; Visita guiada a Madrid Río; Leisure; ItinerariosOtrasActividadesAmbientales
LOG: 12; A su servicio; Leisure; CircoMagia
```

(b) *Resultado contexto 6*

```
LOG: 3-Triggering rules activas
LOG: GoParty,GoShopping,GoToShow,LeisureActivities,LunchOutside,VisitMuseums,VisitPlacesOfInterest
LOG: 4- Tipos de recomendación activadas por Siddhi para el contexto: 7
LOG: leisure,placesOfInterest,
LOG: 5- Tipos de recomendación activadas filtradas por los EXCLUSION SETS:
LOG: placesOfInterest,
LOG: 6- Items recomendados por el gestor de entorno:
LOG: Number of recommended items: 8
LOG: 2810; Escuelas P&iacute,as de San Fernando (Sede UNED); PlacesOfInterest; Edificios y monumentos,
LOG: 2450; Puerto de los Leones; PlacesOfInterest; Otros,
LOG: 2861; Palacio de Liria; PlacesOfInterest; Edificios y monumentos,
LOG: 2735; Fuente de Orfeo; PlacesOfInterest; Edificios y monumentos,
LOG: 2242; Templete de la estaci&oacute,n de metro de Gran V&iacute,a; PlacesOfInterest; Edificios y monumentos,
LOG: 2291; Villa Sotera; PlacesOfInterest; Edificios y monumentos,
LOG: 2852; Iglesia Monasterio de las Benedictinas de San Pl&aacute,cido; PlacesOfInterest; Edificios y monumentos,
LOG: 2322; La abuela roquera; PlacesOfInterest; Edificios y monumentos,
```

(c) *Fragmento de resultado contexto 7*

Figura H.2: Ejemplos de registros recogidos durante la ejecución con el recomendador de Apache Mahout

Anexos I

Artículo MoMM 2021

Este anexo incluye el artículo presentado y admitido en *19th International Conference on Advances in Mobile Computing & Multimedia (MoMM 2021)*. La primera versión del artículo se elaboró para el 5 de agosto de 2021. Posteriormente, el artículo fue admitido y pudimos hacer alguna modificación para el envío de la versión final el 3 de octubre de 2021.

El artículo recoge el trabajo realizado hasta entonces, concretamente la presentación de la arquitectura implementada, la integración de Siddhi en el prototipo previo elaborado con React Native y además, algunos experimentos reaizados con Siddhi en Android para evaluar la tecnología. También se añadieron los prototipos realizados de las nuevas pantallas relacionadas con las *context rules*, *triggering rules* y *exclusion sets*.

An Experience with the Implementation of a Rule-Based Triggering Recommendation Approach for Mobile Devices

Sergio Ilarri
I3A, University of Zaragoza
Zaragoza, Spain
silarri@unizar.es

Irene Fumanal
University of Zaragoza
Zaragoza, Spain
758325@unizar.es

Raquel Trillo-Lado
I3A, University of Zaragoza
Zaragoza, Spain
raqueltl@unizar.es

ABSTRACT

In the current Big Data era, mobile context-aware recommender systems can play a key role to help citizens and tourists to make good decisions. Ideally, these systems should be proactive, able to detect the right moment and place to offer suggestions of a specific type of item or activity to the user. For this purpose, push-based recommender systems can be used, exploiting context rules to decide when a specific type of recommendation should be triggered.

However, experiences regarding the implementation of these types of systems are scarce. Motivated by this, in this paper, we describe our design and implementation efforts focusing on the ability to fire suitable recommendations, without user intervention, whenever it is required. In our proposal, the mobile user can activate, deactivate, parametrize, and define rules in an easy way, to obtain a better user personalization. Besides, the recommendation triggering is performed on the mobile device, which allows minimizing the amount of wireless communications and helps to protect the user's privacy (as context data is evaluated locally on the device, rather than by an external server). We have analyzed several technological options and evaluated the performance and scalability of our proposal, showing its feasibility.

CCS CONCEPTS

• **Information systems** → **Data management systems**; **Recommender systems**; • **Human-centered computing** → **Mobile computing**.

KEYWORDS

data management, mobile computing, mobile context-aware recommender systems, push-based recommendations, context rules, mobile devices

ACM Reference Format:
Sergio Ilarri, Irene Fumanal, and Raquel Trillo-Lado. 2021. An Experience with the Implementation of a Rule-Based Triggering Recommendation Approach for Mobile Devices. In *The 23rd International Conference on Information Integration and Web Intelligence (iiWAS2021), November 29-December 1, 2021, Linz, Austria.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3487664.3487806

iiWAS2021, November 29-December 1, 2021, Linz, Austria

ACM ISBN 978-1-4503-9556-4/21/11…$15.00
https://doi.org/10.1145/3487664.3487806

1 INTRODUCTION

Modern citizens are facing unprecedented challenges for which the development of suitable data management techniques is key [16]. One area where software systems can help is by facilitating decision-making through the pre-selection of options that could be relevant to a user, filtering among a usually-overwhelming set of possible choices. Recommender systems (RS) [22] are specialized in learning user preferences and suggesting items (e.g., points of interest, hotels, restaurants, etc.) that fit those preferences.

More specifically, *Context-Aware Recommender Systems* (*CARS*) [1] extend the classical 2D $User \times Item$ paradigm of RS to a three-dimensional space $User \times Item \times Context$. The key idea is that considering the context of the user (e.g., his/her location, time of the day, weather conditions, etc.) can lead to recommendations that are more appropriate. Besides, in a scenario where the users are moving and interacting with a mobile device, proactive recommender systems (offering push-based recommendations when appropriate, without the need of an explicit user request) [12] are usually preferred over reactive recommender systems (pull-based recommendations) [8].

A basic building block required to build a proactive recommender system is a decision module that determines, based on the context of the user, whether a recommendation of a specific type should be triggered or not. For example, for a certain tourist visiting a city, it may be appropriate to trigger a restaurant recommendation if the user is wandering the city streets and it is lunch time. Suitable context rules could be defined, and customized according to the preferences of the user, in order to detect situations where a given recommendation process should be activated. Several technological options could be exploited for this purpose but, as far as we know, no recent study compares the existing options and evaluates a proposal concerning the development of recommender systems. Moreover, rule-based engines are usually assumed to be executed on servers rather than on mobile devices.

Motivated by this, in this paper we describe our experience with the definition and implementation of a proactive rule-based recommendation architecture. The structure of the rest of the paper is as follows. Firstly, in Section 2, we present the architecture that we have defined to tackle the problem of providing rule-based push-based recommendations in mobile environments. Secondly, in Section 3, we provide an overview of an analysis of some alternative technologies that could be used for the development of a prototype. Thirdly, in Section 4, we describe our prototype. Fourthly, in Section 5, we evaluate the performance and scalability of the proposal. Fifthly, in Section 6, we discuss some related works. Finally, in Section 7, we summarize our conclusions and outline some lines of future work.

2 ARCHITECTURE OF THE SYSTEM

In this section, we present a high-level overview of the architecture proposed to deploy recommender systems that provide proactive (push-based) recommendations to mobile users when appropriate, without requiring any user intervention. The architecture is based on our previous proposal presented in [12] (whose preliminary implementation was described in [15]), that we have extended to support the definition and evaluation of rules. As shown in Figure 1, the main novelty is the focus on the detection of circumstances that should trigger a recommendation process, through the use of a *Complex Event Detection* (*CEP*) module executing on the mobile device. The mobile device captures environment data, through the use of sensors of different types, and sends them to the CEP engine. When the device decides, based on the CEP engine, that a specific type of recommendation should be triggered, then it communicates it to an *Entity Manager* (*EM*), which is a server in charge of providing recommendations, to start the recommendation process; in fact, the mobile device could interact with several EMs at the same time, to communicate the recommendation request to them, in case the user is simultaneously within several recommendation *environments* (areas of influence controlled by an EM). Finally, the EM will return a set of recommended activities that may be filtered on the mobile device based on private user's data not communicated to the EM. In the following, we remark three key aspects of the proposal.

Firstly, we highlight the *execution of the recommendation triggering phase on the mobile device*. The decision about the type of recommendation that should be triggered (if any) is taken on the mobile device. For this reason, a CEP engine that can be executed on mobile devices is needed. The alternative would be to host the CEP engine on a fixed server on the network (e.g., in the EM), but in that case the device should constantly send context data changes to that server, to re-evaluate the conditions that determine if any rule should be activated; this would lead to an increase of wireless communications and therefore drain the battery life of the device due to a higher energy consumption.

Secondly, we mention *privacy preservation*. As the recommendation triggering phase is executed on the mobile device, there is no need to send context data to a server in order to decide whether a specific type of recommendation should be activated or not. Moreover, once a recommendation of a certain type has been triggered, the user is always in control of the data that he/she is willing to share with an external recommendation server (the EM). Thus, for example, shareable non-sensitive preferences can be exploited by the EM (and its recommendation engine) to perform a prefiltering, but private user's data (such as sensitive user preferences and/or sensitive context data) will not leave the mobile device and will be used only as a postfiltering step that will be executed on the device itself.

Finally, a third key feature is the support for *customization of the triggering behavior*. The user can define and customize his/her preferences regarding the activation of recommendations of different types. For this purpose, on the mobile device of the user it is possible to define two different types of rules: 1) *context rules* specify context conditions/situations that may be relevant to the user, and 2) *recommendation triggering rules* (or, for brevity, just *triggering rules*) specify when a certain type of recommendation should be automatically activated (based on the context rules).

As an example of the first type of rules, the user could define a location-based context rule called "In Millennium Park" (that is satisfied whenever the user is in Millennium Park in Chicago, i.e., around the GPS location <41.882702, -87.619392>), a location-based context rule "At home" (which evaluates to true if the user is at his/her home), a time-based context rule "Time to wake up" (indicating the time interval when the user is expected to wake up), or a time-based context rule "Lunch time" (representing the expected time interval for lunch, which could be set depending on the preferences of the user and habits of the region/country where the user is located). As an example of triggering rule, the user could define a rule "Lunch outside", that activates the recommendation of restaurants if it is lunch time and the user is not at home. It is possible to define several recommendation triggering rules to activate the same type of recommendation (e.g., the user could define two rules, "Lunch outside" and "Dinner outside", to activate a recommendation of restaurants for lunch time and also for dinner time). Besides, if the user wants to avoid receiving some types of recommendations at the same time, he/she can define *exclusion sets* (sets of types of recommendations that should not be activated at the same time) and their priorities within the set; for example, the user could define an exclusion set "FirstEatThenWatch" to avoid the simultaneous recommendation of restaurants and museums and giving higher priority to the recommendation of restaurants.

The two types of rules mentioned above have to be managed by a CEP engine running on the mobile device. Basically, the goal of the CEP engine is to detect high-level events that should trigger specific types of recommendations. In the proposed architecture, all the event detection rules are processed on the mobile device, which requires a CEP engine that can be completely executed on the mobile device. In the absence of such a technology, we would need to execute the CEP engine on a fixed server, which would require an efficient communication policy between the mobile device and the remote CEP engine: simpler rules could be executed on the mobile device in order to decide is a specific context variable change should be communicated (i.e., it could potentially trigger a recommendation process) or not. Another possibility would be to implement appropriate context variable update policies. Update policies have been extensively studied in the case of location data [27] (e.g., dead-reckoning policies send an update when the error of the predicted location obtained by considering the last communicated location and speed exceeds a certain threshold); for general sensor data, approaches like the one proposed in [17], based on the use of prediction functions, could be applied.

3 TECHNOLOGIES CONSIDERED

In this section, we describe the technologies that we have considered as potentially useful for our prototype.

3.1 Mobile Development Platforms

As described in [15], there are three main approaches that can be followed for the development of mobile apps:

- Develop a native mobile app, considering a specific platform (e.g., Android or iOS). The main advantage of this solution is

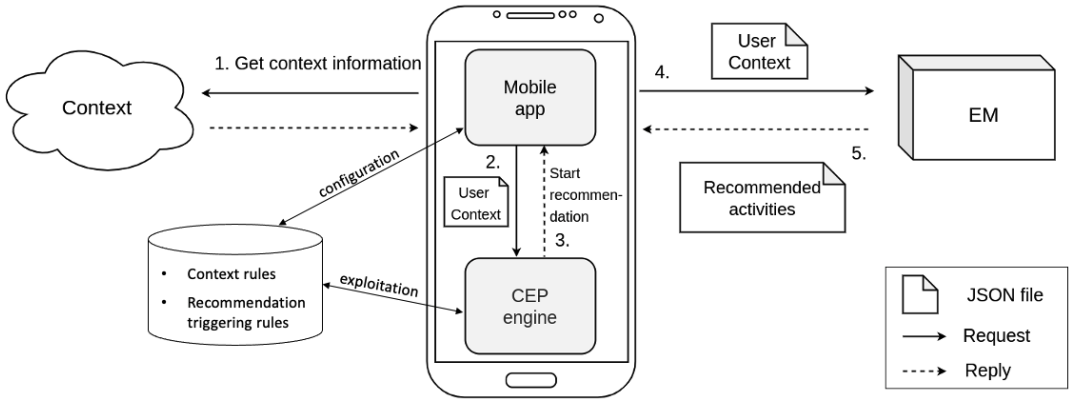


Figure 1: High-level architecture focused on the triggering of recommendations

that we can directly exploit the resources and functionalities of the target platform. The clear disadvantage is the need to develop a different implementation for each platform, that besides will need to be maintained independently.

- Develop a mobile web app, by exploiting the functionalities of standard web technologies (HTML5, CSS, and JavaScript). In this way, the solution developed could be run on any mobile platform with a suitable web browser.
- Develop a hybrid mobile app, using a mobile development framework (e.g., Apache Cordova, Ionic, React Native, Xamarin, Flutter, Framework7, or Appcelerator Titanium). This approach tries to find a balance between flexibility and development cost. It exploits web technologies but also allows access to platform-dependent functionalities (e.g., easy access to sensors and native look-and-feel).

Nowadays, using a development framework is a popular choice. The existing mobile development platforms usually support an easy access to sensors embedded in the mobile device, which obviously play an important role in mobile context-aware recommendation systems [14], thus facilitating the capture of context data. In [5], a recent comparative study evaluating the performance of several cross-platform mobile development frameworks is presented. Among these platforms, we have selected React Native [10] for our project, which is one of the most popular frameworks and offers nice functionalities like convenient programming using JavaScript and the React library, hot reloading (possibility to reload the app without recompiling, even keeping the same app's state), and support for wrapping native code. In an online survey questionnaire described in [4], React Native achieved the highest score in terms of framework interest.

3.2 Rule-Based Context Detection

Complex Event Processing (CEP) systems [6, 7, 11] allow the specification of rules for the detection of patterns, including complex event patterns (defined through aggregation and composition of other events). Our goal is to use these kinds of systems in our architecture in order to decide if a specific context is appropriate to trigger the recommendation of a certain type of item, by defining suitable context rules. In the following, we briefly discuss some technologies that could be considered for our purposes.

3.2.1 Esper. Esper (https://www.espertech.com/esper/) is one of the most well-known CEP technologies. It is open source and available for Java and .NET. It is well-document and, even though it appeared in 2006 (Esper 0.7.0 Alpha was released on January 16, 2006), new versions are still being released (the latest version at the time of writing is 8.7.0, released on January 27, 2021). It supports a rule-definition language based on SQL, called *Event Processing Language* (*EPL*), which facilitates the learning curve and provides a rich and easy-to-use syntax. In 2013, Esper was adapted for Android mobile devices, through the project *Asper* –Esper for Android– (https://github.com/mobile-event-processing/Asper), by addressing dependency restrictions due to missing Java packages and classes in Android. However, Asper has not been updated since then, and the current Asper version is based upon Esper 4.8.0 (released in March, 2013). This is a major inconvenience, since using Esper on Android would require considering a quite old (and outdated) version of Esper.

3.2.2 Siddhi. With a shorter history than Esper, Siddhi (https://siddhi.io/) [25] is another relevant open-source CEP technology (Java library), under Apache License. The first version of Siddhi available was published on April 24, 2015 (Siddhi 2.2.0 Release) and the last current version on March 19, 2021 (Siddhi Core Release 5.1.19). The project is well-documented and integrated with the open-source WSO2 Enterprise Integrator (WSO2 EI) as part of its *Streaming Integrator Tooling*. It supports defining rules using the language *SiddhiQL*, which has a syntax similar to SQL. It can be used as a Java or Python library and also as a microservice in Docker or Kubernetes. Besides, it provides a desktop tool (*WSO2 Integration Studio*), developed by WSO2, that can be used to define and debug rules, allowing the simulation of events to check that they work correctly. Siddhi can be run on an Android application. As an example, an approach using Siddhi for monitoring patients in a health context was presented in [9]. This use case illustrates the use of "the edge" to perform complex event detection autonomously: rather than continuously communicate sensor data, a complex event is communicated to the remote IoT Hospital Server (IHS), using

the MQTT (Message Queuing Telemetry Transport) protocol, only when it is detected by the CEP engine running on the mobile device.

3.2.3 Apache Flink. Flink (https://flink.apache.org/) is an open-source data stream processing technology that includes a library to process complex events, called *Apache Flink CEP*. Flink is a very popular solution for the processing of data streams, including CEP support, but it is not specifically a CEP engine. Flink focuses on distributed data processing and has not been designed to be executed on mobile devices. As far as we know, it is not possible to execute it on Android devices. Besides, we are not aware of any approach trying to port Apache Flink CEP for Android.

3.2.4 Drools. Drools (https://www.drools.org/) is another popular Java-based open-source CEP technology or, as stated in its website, a Business Rules Management System (BRMS) solution. The language *DRL* (*Drools Rule Language*) is used to define the required business rules in .drl text files. As usual, rules are composed, at least, by conditions (when) and actions (then). According to [26], the use of resources by Drools is higher than in the case of Esper or Siddhi. Besides, it is not possible to directly execute Drools on an Android mobile device. However, in [23], which focuses on smart home systems, the authors mention the use of a ported version of Drools for Android; this seems to be an ad hoc solution applied by the authors and not publicly available. Porting issues with Drools, regarding the use of memory, are mentioned in [20].

3.2.5 Microsoft StreamInsight. Microsoft StreamInsight (https://docs.microsoft.com/en-us/archive/msdn-magazine/2012/march/microsoft-streaminsight-building-the-internet-of-things) is a commercial and well-known CEP technology. In this case, this solution is not open source and a SQL Server license is required to use it. Rather than defining rules, queries can be processed over the streaming data using the LINQ (.NET Language-Integrated Query) language. This technology is nowadays part of Microsoft SQL Server and cannot be executed directly on mobile devices.

3.2.6 Conclusions: Chosen CEP Engine. Our main goal is to have a CEP engine that can be executed on a mobile device. This is a requirement only satisfied by Asper and Siddhi (the other technological solutions could be used in case we adopted a different architecture where the rule detection could be performed on the EM, as briefly mentioned at the end of Section 2). To choose between Asper and Siddhi, we developed a simple mobile app in Android, with three simple buttons: one to start the CEP engine, another one to stop the CEP engine, and finally a third one to send an event to the CEP engine. We can note the following:

- The sample mobile app was successfully implemented by adding the Asper library to an Android Studio project. However, as mentioned in Section 3.2.1, Asper is based on Esper 4.8, which is very old. We tried to overcome this difficulty by testing if more recent versions of Esper (Esper 7.1 y 8.7) were directly compatible with Android (without using Asper). However, we noticed problems with dependencies (required classes that were not available in Android).
- The mobile app was also implemented, with no problems, using Siddhi. We tried with both the latest version of Siddhi and also with version 4. It would have been useful to be able to use and define custom functions in JavaScript to process some context data (e.g., to compare location coordinates and determine if they are equal), to be used as filters in the rules, but unfortunately Siddhi's JavaScript Extension, that allows to do this, is not well supported in Android (because some classes are not available in Android); therefore, we had to define the rules and the required computations using a more complex/verbose syntax using SiddhiQL.

There are some recent relevant surveys where other CEP technologies are analyzed [7, 11], but with no focus on their execution on mobile devices to be applied in the context of mobile recommender systems.

4 PROTOTYPE

In this section, we describe our prototype, focusing on the integration of the CEP engine, using Siddhi, and on the functionalities related with the definition and personalization of user rules. We have built our solution considering our previous work [15]. With the same spirit, we also use React Native (https://reactnative.dev/) [10] and the main code of the mobile app is written in JavaScript. Siddhi's engine can consume events coming from different sources and process them as required by the application. Once the events have been processed, it provides the results through different channels, such as HTTP, Kafka, or email. In our case, we use the Siddhi library to send and receive events using Java code.

We deployed Siddhi as an Android service, which allows executing short or long-term duration operations in the background, without affecting the Graphical User Interface (GUI) of the mobile app. There are three types of services in Android: foreground services, background services, and bound services. Specifically, we have used a bound service because it defines a client/server interface that supports interacting with the service.

The main code of our mobile app is written in JavaScript, but we need to use Android's native code to access Siddhi. To address this difficulty, we have created an Android Native Module (called *SiddhiClientModule*) operating as a client of the Siddhi engine: it invokes Siddhi's operations (e.g., start the engine, stop it, send an event, or retrieve a result) from the main code of the mobile app in JavaScript. More specifically, we distinguish three types of native classes: 1) *SiddhiAppManager* handles the communication between the app and Siddhi's engine, 2) *SiddhiService* implements the logic of the bound service, and 3) *SiddhiClientModule* is the native module that acts as a communication bridge between the Android's code and React Native's code (the operations described in this module can be called from the JavaScript code of the mobile app).

With a specified *context update period* (e.g., every 30 seconds), the mobile app will send the context information to Siddhi's engine (this communication could be avoided in case there is no context change). Besides, the mobile app also needs to listen to potential results from Siddhi, that could indicate the need of triggering a recommendation process of a certain type. For this purpose, we use *Headless JS* (which allows to run tasks in JavaScript while the app is executing in the background), supported by React Native, to define a service that periodically sends the user's context data to Siddhi (*SendContextTask*) and another service that listens to results provided by Siddhi (*ListRecommendationResultTask*).

Siddhi's engine only reports recommendation triggering rules that get activated. When a triggering rule is activated (i.e., when the current context matches the rule's conditions), an event is stored in a *Result* stream, which is retrieved by using a ReactNative's *callback*, as indicated in the official documentation of Siddhi concerning its use as a Java library. Instead of directly considering the type of recommendation associated to the rule that has been fired, the mobile app collects all the types of recommendations corresponding to recommendation triggering rules fired within a specific batch time window (e.g., 5 seconds by default in our current prototype). In this way, an example of batched output obtained could be <contextId, {restaurantRecommendation, museumRecommendation}>. Then, the batched output retrieved is processed to determine which of those types of recommendations should actually be initiated, based on the possible exclusion sets defined by the user and the priority of types of recommendations within each set (as explained in Section 2). To implement this batching functionality, we use the concept of *batch windows* provided by Siddhi, applying Siddhi's *timeBatch* function. An auxiliary thread is created by the *SiddhiClientModule* to obtain the results from Siddhi, as its executing thread should not be busy executing a task that could potentially require some time to complete.

We show some examples of screenshots in Figures 2 and 3. On the left of Figure 2, we show the screen that allows the definition, modification, deletion, and temporary deactivation of recommendation triggering rules. In the middle of Figure 2, we show an example of recommendation triggering rule defined by the user: he/she wants the recommendation of restaurants to be activated when it is lunch time and he/she is not at home. On the right of Figure 2, we show the screen that allows the definition and modification of context rules. These conditions correspond with context rules that the user can customize according to his/her specific needs, as shown in Figure 3; regarding the left screen in Figure 3, notice that using the button "Get current GPS location" may return an estimated location value (or the last known location) when the user is indoors (using positioning mechanisms other that the GPS, like the WiFi network the user is connected to). Another example of triggering rules supported (not shown in Figure 3) are calendar-based rules.

5 EXPERIMENTAL EVALUATION

We have performed some experiments to measure the performance of the recommendation triggering phase, which implies executing the Siddhi engine on a mobile device. Specifically, we have used a mobile device with a Qualcomm Snapdragon 626 processor (octa-core A53, 2,2 GHz), 4 GB RAM, and Android version 8.1.0.

First, we have defined a set of context rules based on 7 context variables: *dayOfTheWeek* (with possible values Monday, Tuesday, etc.), *season* (winter, spring, summer, or autumn), *partOfTheDay* (early morning, morning, afternoon, or night), *timeForDailyActivity* (time for lunch or time for dinner), *weatherStatus* (cloudy, clear, raining, etc.), *atHome* (with value true if the user is at home and false otherwise), and *onHolidays* (that indicates if the user is currently enjoying a holiday period). Then, we have defined a Python script that automatically defines recommendation triggering rules by randomly combining the previous context rules defined. The generation process avoids combining contradicting context rules in a single triggering rule (e.g., weatherStatus=clear AND weatherStatus=cloudy), as this would make no sense and the rule would be never satisfied. Each triggering rule has initially a single context rule; then, iteratively, with 25% probability, another context rule is added and with 75% probability no rule is added; this is repeated until it is not possible to add more non-contradicting context rules or the random decision taken at the last iteration does not add a new context rule. Following this process, 300 recommendation triggering rules have been synthetically generated for the experimental evaluation (each with a number of context rules between 1 and 7); in our experiments, the resulting average number of context rules per triggering rule is 3.36. Afterwards, by using another Python script, we have randomly generated a set of example contexts (that define values for each of the possible context variables that are used in the context rules) in JSON format, as required by Siddhi. 21 different contexts have been synthetically generated for the experiments.

Finally, we have performed the experiments. For each experiment, we define in the mobile device's app all the context rules, a subset of the recommendation triggering rules (the first n rules, depending on the number of recommendation triggering rules to consider in that particular experiment), and simulate context changes by sending to Siddhi each context defined (one context update every 20 seconds); then, we measure the *triggering latency* of each type of recommendation activated with each change of context, defined as the time elapsed between the context change and the time instant when the need to trigger a specific type of recommendation is detected. We measure this value using the module *SiddhiAppManager* (described in Section 4). As a change of context can activate several triggering rules at the same time, we collect the triggering latency of each single recommendation triggering rule activated by each change of context. In the following, we report the results obtained when considering the 21 context changes simulated.

On the left of Figure 4, we show the average triggering latency as well as the minimum triggering latency (triggering latency of the first rule activated by a context change) and the maximum triggering latency (triggering latency of the last rule activated by a context change). It can be seen that the highest triggering latency increases considerably with the number of recommendation triggering rules, as expected; however, the average triggering latency increases only very slowly; finally, the lowest latency is not affected by the number of recommendation triggering rules, as Siddhi checks the rules one by one, and therefore the cost of the first activation should be similar independently of the number of rules. It must be stressed that the latencies measured are always very small, not exceeding 220 milliseconds even in the worst case, that arises when the number of recommendation triggering rules is very high (200 recommendation triggering rules); we think that it is very unlikely that a mobile user will define such a high number of recommendation triggering rules and, even if he/she does it, the latencies will be anyway acceptable. On the right of Figure 4, we show the number of recommendation triggering rules activated depending on the number of recommendation triggering rules defined: obviously, the higher the number of recommendation triggering rules defined, the higher the expected number of rules that could be activated by a context change.

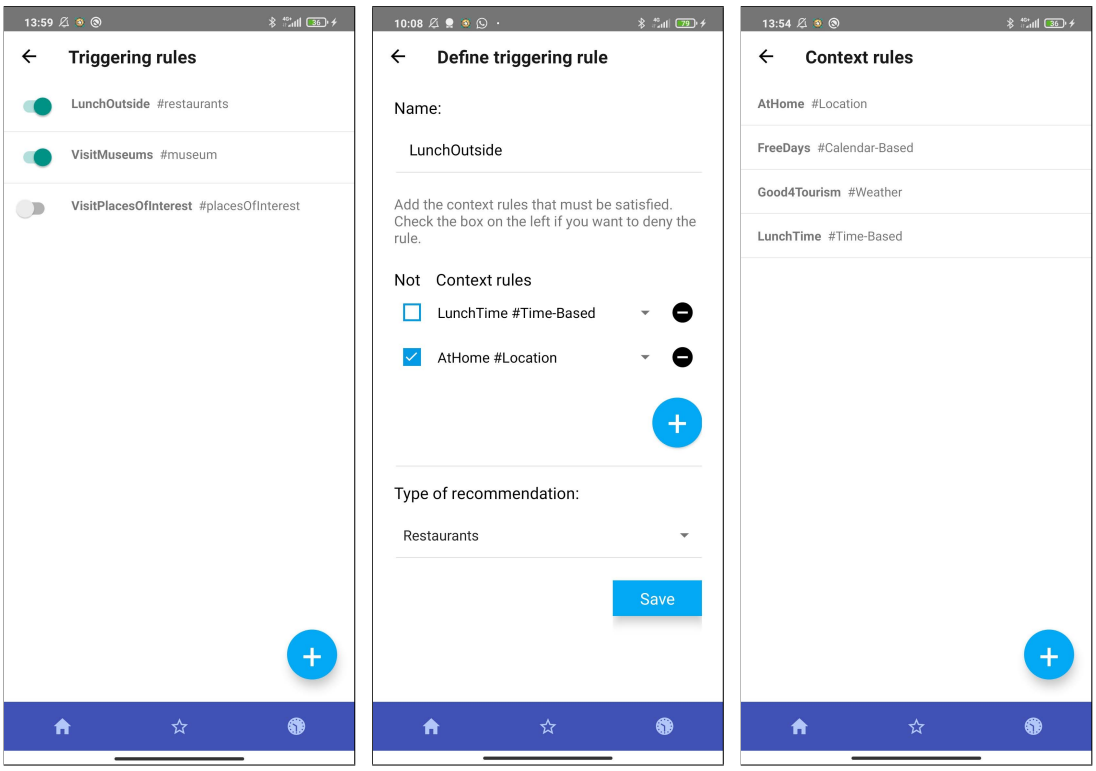


Figure 2: Screen to manage triggering rules (left), to define a new triggering rule (middle), and to manage context rules (right)

6 RELATED WORK

As described in Section 1, this work belongs to the field of recommender systems [22], and more specifically context-aware recommender systems [1]. Whereas a significant amount of research has been developed in the field of context-aware computing [2] and also in the field of context-aware recommender systems (CARS), it is more difficult to find practical experiences regarding the implementation of CARS using existing technologies and tools. The most related work is our previous proposal [15], based on the theoretical foundations described in [12], where we did not focus on the recommendation triggering phase, that is the core of this paper. Besides, in [13], we described some use case scenarios and presented some examples of context attributes and rules (with a SWRL-like syntax) that could be considered to trigger several types of recommendations; however, in [13] we did not carry out any practical implementation or testing.

In the proposal presented in this paper, we empower the user to be in control of the data he/she shares with external servers (thus protecting his/her privacy) and also to define and customize the context rules and recommendation triggering rules that are appropriate to him/her. The work presented in [24] also proposes the use of user-defined rules to activate recommendations (an example of rule shown in [24] is "If day is Monday then I would like to go to a Restaurant which is closer than 5 KM"): the users use a web form to specify the rule, which is then translated to a RuleML rule; however, that paper presents a model, which does not seem to be supported by an implementation/prototype.

The use of user-defined rules has also been proposed in other fields not related to recommender systems. For example, for the detection of network security situations, rules in SWRL are defined to compensate for the limited description ability of an ontology and improve the reasoning ability of the proposed model [28]. As another example, *SECE (Sense Everything, Control Everything)* [3] is a rule-based context-aware system that supports defining rules in a natural English-like formal language to compose different types of services based on events; an important difference with our work, besides belonging to a different research area, is that SECE is a web service (not something to be executed on a mobile device). As a final example, *LLA (Long-Life Application)* [18, 19] is a single context-aware distributed mobile application dedicated to everyday users; the idea is to inject the desired *situations* (context descriptions) and then matching situations to services in order to respond to contextual changes when it is suitable; appropriate situations may be injected by the user himself/herself, external providers (such as governments, business and private companies, and institutions/organizations/associations), or even a component of LLA that monitors the social environment of the user (e.g., tasks defined in Google Calendar) to suggest possible relevant situations.

There are also some rule-based end-user applications worth mentioning. For example, *IFTTT (If This, Then That)* [21] relies on a web service that allows automating some tasks and actions

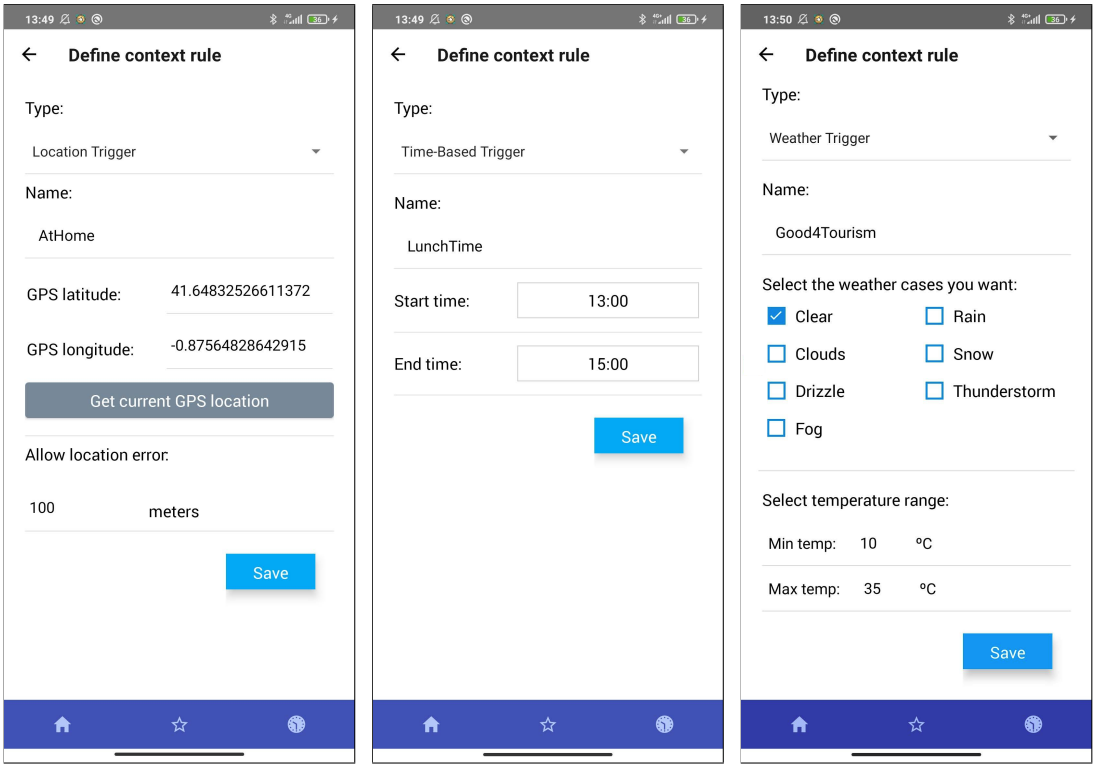


Figure 3: Screen to define a location-based context rule (left), a time-based context rule (middle), and a weather context rule (right)

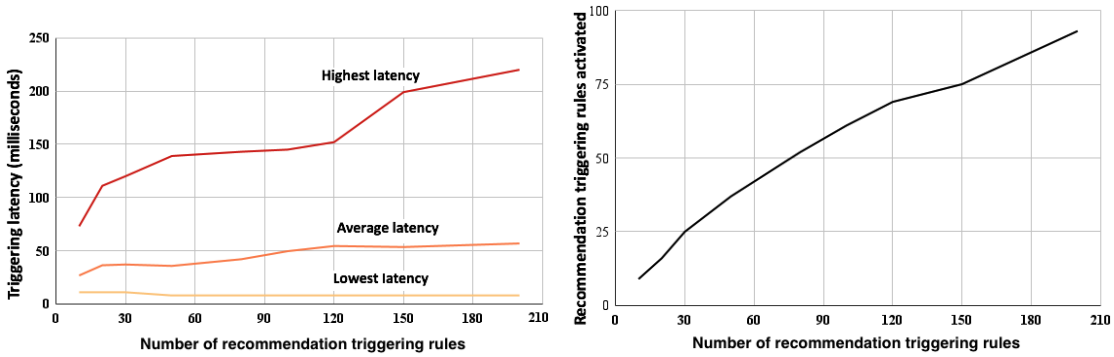


Figure 4: Evaluation of the triggering latency (left) and number of rules activated (right)

by connecting apps (e.g., the automatic publication in Twitter of a picture that the user has posted in Facebook, to relieve the user from the extra work of having to upload the same picture to several sites). The user can define those actions through the IFTTT's website or by using the IFTTT mobile app (available for Android and iOS). According to the web page of IFTTT, currently over 600 apps and devices work with IFTTT (e.g., apps like Twitter, Telegram, Google Drive, Gmail, and devices like Google Home, Alexa, etc.). Similarly, in the *Shortcuts* app (https://support.apple.com/en-gb/guide/shortcuts/), available for iOS devices, it is possible to define the conditions under which a specific task (shortcut) should run automatically (e.g., based on the time of the day or the location); specifically, four different types of triggers can be considered: event triggers (a specific time of the day, the use of an alarm on the user's phone, or an Apple Watch workout done by the user), travel triggers (when the user arrives in a location or leaves it, when the user connects or disconnects the CarPlay (https://www.apple.com/ios/carplay), or when a commute by the user is going to occur), communication triggers (reception of specific types of emails or messages), and setting triggers (related to modifications like activating the airplay

mode, changes in the battery level, connecting to a WiFi network, etc.).

Finally, as mentioned in Section 3.2, there are some surveys that analyze CEP technologies [7, 11], but with no focus on their execution on mobile devices to build proactive recommender systems. Thus, as far as we know, existing works mainly consider the execution of CEP engines on fixed servers, rather than on mobile devices. However, as explained in this paper (and particularly in Section 2), executing the recommendation triggering phase on the mobile device (rather than on a server) is expected to reduce the amount of wireless communications from the mobile device (communications of context updates) and thus the energy consumption; besides, this approach helps to keep the privacy of the users.

7 CONCLUSIONS AND FUTURE WORK

In this paper, we have described our experience with the development of a proactive push-based recommendation architecture, with an emphasis on the definition and triggering of context rules. A key distinct advantage of the proposed architecture is that the rule-based engine is executed on the mobile device, which improves the performance and maximizes the user's privacy (as the context is evaluated locally on the device). Besides, our flexible proposal gives full power to the user, by allowing him/her to define and customize his/her own rules. We have analyzed the technologies that could be used for this purpose and evaluated the performance and scalability of our proposal, showing its feasibility.

Concerning the prototype, we are currently finishing the integration of some functionalities of our previous prototype [15] (that did not focus on the recommendation triggering phase, as we do in this paper). Furthermore, we have recently carried out a survey with users to identify types of context rules and recommendation triggering rules that they would find useful (we have received 85 answers at the time of writing), which can help us to refine our prototype by defining a more complete customizable rule library. Finally, the possibility to develop a prototype that works on iOS could be analyzed in more detail in the future.

ACKNOWLEDGMENTS

Research supported by the Government of Aragon – Aragon-New Aquitaine project PASEO 2.0 (AQ-8), the project PID2020-113037RB-I00 / AEI / 10.13039/501100011033, and the *Departamento de Ciencia, Universidad y Sociedad del Conocimiento del Gobierno de Aragón* (Government of Aragon: group reference T64_20R, COSMOS research group).

REFERENCES

[1] Gediminas Adomavicius, Bamshad Mobasher, Francesco Ricci, and Alexander Tuzhilin. 2011. Context-aware recommender systems. *AI Magazine* 32, 3 (Fall 2011), 67–80. https://doi.org/10.1609/aimag.v32i3.2364

[2] Matthias Baldauf, Schahram Dustdar, and Florian Rosenberg. 2007. A Survey on Context-Aware Systems. *International Journal of Ad Hoc and Ubiquitous Computing* 2, 4 (June 2007), 263–277. https://doi.org/10.1504/IJAHUC.2007.014070

[3] Victoria Beltran, Knarig Arabshian, and Henning Schulzrinne. 2011. Ontology-Based User-Defined Rules and Context-Aware Service Composition System. In *Extended Semantic Web Conference (ESWC 2012) Workshops*. Lecture Notes in Computer Science (LNCS), Vol. 7117. Springer, Berlin, Heidelberg, 139–155. https://doi.org/10.1007/978-3-642-25953-1$_1$2

[4] Andreas Biørn-Hansen, Tor-Morten Grønli, Gheorghita Ghinea, and Sahel Alouneh. 2019. An Empirical Study of Cross-Platform Mobile Development in Industry. *Wireless Communications and Mobile Computing* 2019 (January 2019), 1–12. https://doi.org/10.1155/2019/5743892

[5] Andreas Biørn-Hansen, Christoph Rieger, Tor-Morten Grønli, Tim A. Majchrzak, and Gheorghita Ghinea. 2020. An empirical investigation of performance overhead in cross-platform mobile development frameworks. *Empirical Software Engineering* 25, 4 (June 2020), 2997–3040. https://doi.org/10.1007/s10664-020-09827-6

[6] Gianpaolo Cugola and Alessandro Margara. 2012. Processing Flows of Information: From Data Stream to Complex Event Processing. *Comput. Surveys* 44, 3, Article 15 (June 2012), 62 pages. https://doi.org/10.1145/2187671.2187677

[7] Miyuru Dayarathna and Srinath Perera. 2018. Recent Advancements in Event Processing. *Comput. Surveys* 51, 2, Article 33 (February 2018), 36 pages. https://doi.org/10.1145/3170432

[8] María del Carmen Rodríguez-Hernández and Sergio Ilarri. 2016. Pull-Based Recommendations in Mobile Environments. *Computer Standards & Interfaces* 44 (February 2016), 185–204. https://doi.org/10.1016/j.csi.2015.08.002

[9] Amarjit Singh Dhillon, Shikharesh Majumdar, Marc St-Hilaire, and Ali El-Haraki. 2018. A Mobile Complex Event Processing System for Remote Patient Monitoring. In *IEEE International Congress on Internet of Things (ICIOT 2018)*. IEEE, USA, 180–183. https://doi.org/10.1109/iciot.2018.00034

[10] Bonnie Eisenman. 2015. *Learning React Native: Building Native Mobile Apps with JavaScript*. O'Reilly Media, Sebastopol, California, USA.

[11] Nikos Giatrakos, Elias Alevizos, Alexander Artikis, Antonios Deligiannakis, and Minos Garofalakis. 2019. Complex event recognition in the Big Data era: a survey. *The VLDB Journal* 29, 1 (July 2019), 313–352. https://doi.org/10.1007/s00778-019-00557-w

[12] Ramón Hermoso, Sergio Ilarri, Raquel Trillo, and María del Carmen Rodríguez-Hernández. 2015. Push-based recommendations in mobile computing using a multi-layer contextual approach. In *13th International Conference on Advances in Mobile Computing and Multimedia (MoMM 2015)*. ACM, New York, NY, USA, 149–158. https://doi.org/10.1145/2837126.2837128

[13] Ramón Hermoso, Sergio Ilarri, and Raquel Trillo-Lado. 2018. Proactive Mobile CARS in Action: A First Step Towards Making Sense of Context Rules. In *13th International Workshop on Semantic and Social Media Adaptation and Personalization (SMAP 2018)*. IEEE, USA, 69–74. https://doi.org/10.1109/SMAP.2018.8501879

[14] Sergio Ilarri, Ramón Hermoso, Raquel Trillo-Lado, and María del Carmen Rodríguez-Hernández. 2015. A Review of the Role of Sensors in Mobile Context-Aware Recommendation Systems. *International Journal of Distributed Sensor Networks* 2015 (November 2015), 1–30. https://doi.org/10.1155/2015/489264

[15] Sergio Ilarri and Manuel Herrero. 2018. Towards the Implementation of a Push-Based Recommendation Architecture. In *13th International Workshop on Semantic and Social Media Adaptation and Personalization (SMAP 2018)*. IEEE, USA, 87–92. https://doi.org/10.1109/SMAP.2018.8501875

[16] Sergio Ilarri, Raquel Trillo-Lado, and Thierry Delot. 2020. Social-Distance Aware Data Management for Mobile Computing. In *18th International Conference on Advances in Mobile Computing & Multimedia (MoMM 2020)*. ACM, New York, NY, USA, 138–142. https://doi.org/10.1145/3428690.3429164

[17] Sergio Ilarri, Ouri Wolfson, Eduardo Mena, Arantza Illarramendi, and Prasad Sistla. 2009. A Query Processor for Prediction-Based Monitoring of Data Streams. In *12th International Conference on Extending Database Technologies (EDBT 2009)*, Vol. 360. ACM, New York, NY, USA, 415–426.

[18] Riadh Karchoud, Arantza Illarramendi, Sergio Ilarri, Philippe Roose, and Marc Dalmau. 2017. Long-Life Application – Situation Detection in a Context-Aware All-in-one Application. *Personal and Ubiquitous Computing* 21, 6 (December 2017), 1025–1037. https://doi.org/10.1007/s00779-017-1077-2

[19] Riadh Karchoud, Philippe Roose, Marc Dalmau, Arantza Illarramendi, and Sergio Ilarri. 2019. One App to Rule Them All: Collaborative Injection of Situations in an Adaptable Context-Aware Application. *Journal of Ambient Intelligence and Humanized Computing* 10 (December 2019), 4679–4692. Issue 12. https://doi.org/10.1007/s12652-018-0846-8

[20] Chinmoy Mukherjee. 2017. *Build Android-Based Smart Applications: Using Rules Engines, NLP and Automation Frameworks*. Apress, USA.

[21] Steven Ovadia. 2014. Automate the Internet With "If This Then That" (IFTTT). *Behavioral & Social Sciences Librarian* 33, 4 (2014), 208–211. https://doi.org/10.1080/01639269.2014.964593

[22] Francesco Ricci, Lior Rokach, Bracha Shapira, and Paul B. Kantor. 2011. *Recommender systems handbook*. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-0-387-85820-3

[23] Markus Schinle, Johannes Schneider, Timon Blöcher, Jochen Zimmermann, Sebastian Chiriac, and Wilhelm Stork. 2017. A Modular Approach for Smart Home System Architectures Based on Android Applications. In *Fifth IEEE International Conference on Mobile Cloud Computing, Services, and Engineering (MobileCloud 2017)*. IEEE, USA, 153–156. https://doi.org/10.1109/MobileCloud.2017.20

[24] Silky Sharma and Damandeep Kaur. 2015. Location based context aware recommender system through user defined rules. In *International Conference on Computing, Communication Automation (ICCCA 2015)*. IEEE, USA, 257–261. https://doi.org/10.1109/CCAA.2015.7148384

[25] Sriskandarajah Suhothayan, Kasun Gajasinghe, Isuru Loku Narangoda, Subash Chaturanga, Srinath Perera, and Vishaka Nanayakkara. 2011. Siddhi: A Second

Look at Complex Event Processing Architectures. In *2011 ACM Workshop on Gateway Computing Environments (GCE 2011)*. ACM, New York, NY, USA, 43–50. https://doi.org/10.1145/2110486.2110493

[26] Kenny Warszawski. 2020. *Complex Event Processing for Internet of Things: Open-Source Frameworks Analysis.* Master's thesis. University of Namur (Belgium). Available at https://researchportal.unamur.be/en/studentTheses/complex-event-processing-for-internet-of-things [Accessed: October 3, 2021].

[27] Ouri Wolfson, A. Prasad Sistla, Sam Chamberlain, and Yelena Yesha. 1999. Updating and Querying Databases that Track Mobile Units. *Distributed and Parallel Databases* 7, 3 (1999), 257–387. https://doi.org/10.1023/A:1008782710752

[28] Guangquan Xu, Yan Cao, Yuanyuan Ren, Xiaohong Li, and Zhiyong Feng. 2017. Network Security Situation Awareness Based on Semantic Ontology and User-Defined Rules for Internet of Things. *IEEE Access* 5 (2017), 21046–21056. https://doi.org/10.1109/ACCESS.2017.2734681