ESCUELA POLITÉCNICA NACIONAL

FACULTAD DE INGENIERÍA ELÉCTRICA Y ELECTRÓNICA

IMPLEMENTACIÓN DE UN PROTOTIPO DE SISTEMA DE RECOMENDACIÓN PARA PUBLICIDAD INTELIGENTE

PROYECTO DE TITULACIÓN PREVIO A LA OBTENCIÓN DEL TÍTULO DE INGENIERO EN "ELECTRÓNICA Y REDES DE INFORMACIÓN"

ERICK AMIR PEÑAFIEL MASSÓN

erick.penafiel@epn.edu.ec

PROYECTO DE TITULACIÓN PREVIO A LA OBTENCIÓN DEL TÍTULO DE INGENIERO EN "ELECTRÓNICA Y TELECOMUNICACIONES"

JAIME ORLANDO TAMAYO PORTERO

jaime.tamayo01@epn.edu.ec

DIRECTOR: M.Sc. RAÚL DAVID MEJÍA NAVARRETE

david.mejia@epn.edu.ec

Quito, agosto 2018

AVAL

Certifico que el presente trabajo fue desarrollado por Erick Amir Peñafiel Massón y Jaime Orlando Tamayo Portero, bajo mi supervisión.

M.Sc. RAÚL DAVID MEJÍA NAVARRETE
DIRECTOR DEL TRABAJO DE TITULACIÓN

DECLARACIÓN DE AUTORÍA

Nosotros, Erick Amir Peñafiel Massón y Jaime Orlando Tamayo Portero, declaramos bajo juramento que el trabajo aquí descrito es de nuestra autoría; que no ha sido previamente presentado para ningún grado o calificación profesional; y, que hemos consultado las referencias bibliográficas que se incluyen en este documento.

A través de la presente declaración cedemos nuestros derechos de propiedad intelectual correspondientes a este trabajo, a la Escuela Politécnica Nacional, según lo establecido por la Ley de Propiedad Intelectual, por su Reglamento y por la normatividad institucional vigente.

ERICK AMIR PEÑAFIEL MASSÓN

JAIME ORLANDO TAMAYO PORTERO

DEDICATORIA

Dedico este trabajo de titulación a mi familia la cual ha sido mi más grande inspiración para cumplir esta meta en mi vida, a mis amigos quienes caminaron junto a mí durante este trayecto, y cada profesor, maestro del cual he tenido la oportunidad de aprender.

Dedico esto a las futuras generaciones, para que sirva de inspiración en la búsqueda de conocimiento.

Erick Amir Peñafiel Massón

DEDICATORIA

Este trabajo de titulación está dedicado las personas e instituciones que más han influenciado en mi vida, dándome los mejores consejos, guiándome y formándome a ser una persona de bien. Dedico esto a: mis padres y profesores que compartieron los años más importantes de mi vida.

Jaime Orlando Tamayo Portero

AGRADECIMIENTO

Agradezco a mi madre, por su gran sacrificio, y por su amor incondicional que me impulsó a cumplir esta meta.

A mi padre, quien me ha acompañado en cada etapa de mi vida, cuyo amor, consejos y guía me han forjado como ser humano.

A mis abuelos, cuya memoria y enseñanzas vivirán a través de mí.

A mi tía, cuyo apoyo y ejemplo fueron parte fundamental para conseguir esta meta.

A mis amigos, con quienes compartimos risas, victorias y derrotas, cuya amistad valoro cada día.

A la Escuela Politécnica Nacional, por brindar una fuente de conocimientos y sabiduría.

Erick Amir Peñafiel Massón

AGRADECIMIENTO

A quienes me han heredado el tesoro más valioso que pueden dárselo a un hijo: amor. A quienes sin escatimar esfuerzo alguno han sacrificado gran parte de su vida para formarme y educarme. A quienes nunca podré pagar todos sus desvelos ni aún con las riquezas más grandes del mundo. Por eso y más Gracias.

A Dios y a la Virgen Santísima

A mis padres: Margot y Enrique.

A mi hermana Carol.

Jaime Orlando Tamayo Portero

ÍNDICE DE CONTENIDO

ÍNDICE DE FIGURAS

ÍNDICE DE ECUACIONES

ÍNDICE DE TABLAS

ÍNDICE DE CÓDIGOS

RESUMEN

En la actualidad, la publicidad tiene un alcance tan grande que se encuentra en todos los medios de comunicación y no está orientada a un segmento en particular. Por lo tanto, se tiene una gran cantidad de publicidad que puede resultar abrumante para el consumidor. Por lo mencionado, se plantea desarrollar un prototipo de sistema de recomendaciones para publicidad inteligente. Este prototipo permitirá demostrar los beneficios de la publicidad inteligente y las ventajas que podría traer para consumidores. El presente documento consta de cuatro capítulos:

El primer capítulo consta de una breve introducción sobre sistemas de recomendaciones, modelos y algoritmos básicos. Posteriormente se presenta características relevantes a la implementación del prototipo, como el desarrollo de aplicaciones Android, MySQL, y servicios WCF.

En el segundo capítulo se presenta la metodología empleada para el desarrollo de este proyecto técnico. La metodología está dividida en dos fases: diseño, e implementación. La fase de diseño consta de un análisis de sistemas de recomendaciones existentes, establecimiento de requerimientos de usuario, y diseño de los componentes. En la fase de implementación se presenta la codificación de los componentes.

En el tercer capítulo se presenta resultados y discusión sobre las pruebas realizadas a los distintos componentes del prototipo. También se presenta los resultados sobre encuestas de validación a diferentes usuarios.

El cuarto capítulo contiene conclusiones y recomendaciones que se han obtenido al realizar el presente proyecto técnico.

Finalmente se han incluido los anexos: I) Encuestas de requerimientos de usuario, II) Scripts para la creación de tablas, III) Solución del algoritmo de recomendaciones (Código Fuente), IV) Solución del servicio WCF (Código Fuente), V) Solución del cliente Android (Código Fuente), VI) Resultados de recomendaciones del cliente Android, VII) Calificaciones de clientes para pruebas del algoritmo de recomendaciones, VIII) Encuestas de validación, IX) Manual de usuario y X) Diagramas de clases detallados de los componentes del prototipo.

PALABRAS CLAVE: sistema de recomendación, publicidad inteligente, cliente Android, servicio WCF, base de datos SQL, algoritmo de recomendación.

ABSTRACT

Nowadays, publicity has such a long range that is present in every communication media and it is not aimed to a particular segment. Hence, there is a large amount of publicity that can be overwhelming for the consumer. For the aforementioned, the development of a recommender system prototype for smart publicity is proposed. This prototype will allow to demonstrate de benefits of smart publicity and the advantages that could bring to the consumers.

The present document consists of four chapters:

The first chapter consists of a brief introduction about recommender systems, models and basic algorithms. Later, relevant characteristics to the implementation of the prototype are presented, like the development of Android applications, MySQL, and WCF services.

In the second chapter, the methodology used in the development of this technical study is presented. The methodology is divided in two phases: design and implementation. The design phase consists of the analysis of existent recommender systems, establishment of user requirements, and design of the components. In the implementation phase, the coding of the components is presented.

In the third chapter, results and discussion about tests made to the different components of the prototype are presented. Also, results about validation polls made to different users are presented.

The fourth chapter contains conclusions and recommendations obtained during the realization of the present technical study.

Finally, the next appendixes have been included: I) User requirement polls, II) Table scripts, III) Recommender algorithm solution (Source code), IV) WCF service solution, V) Android client solution (Source Code), VI) Android client recommendation's results, VII) Customer ratings for tests of the recommendation algorithm VIII) Validation polls, IX) User guide and X) Detailed prototype components class diagrams.

KEY WORDS: recommender system, smart publicity, Android client, WCF service, SQL database, recommendation algorithm.

1. INTRODUCCIÓN

En la actualidad, la publicidad tiene un alcance tan grande que se encuentra en todos los medios de comunicación, como televisión, radio, Internet, prensa y en las calles; en la mayoría de casos se difunde de manera general y no está orientada a un segmento en particular. Por lo tanto, se tiene una gran cantidad de publicidad que puede resultar abrumante para el consumidor [1], [2].

El uso de sistemas de recomendaciones puede integrarse a la difusión de la publicidad para solucionar el inconveniente de tener anuncios no orientados a un segmento en particular. De esta manera los consumidores recibirán anuncios sobre productos o servicios relevantes a sus intereses. Los sistemas de recomendaciones han ayudado a las personas a manejar y controlar el mundo de la información abundante y sobrecargada [3].

Los beneficios de esta tecnología incluyen: incrementar el número de ventas, incrementar la satisfacción del usuario, incrementar la felicidad del usuario y comprender mejor lo que el usuario quiere [4].

El enfoque de este proyecto técnico es desarrollar un prototipo de sistema de recomendaciones para publicidad inteligente el cual permitirá demostrar los beneficios de la publicidad inteligente y las ventajas que podría traer para consumidores el disponer de recomendaciones provistas con base en sus preferencias.

1.1 Objetivos

El objetivo general de este proyecto técnico es: desarrollar un prototipo de sistema de recomendación que presente publicidad de acuerdo a la localización y preferencias del usuario.

Los objetivos específicos de este proyecto técnico son:

- Analizar los algoritmos empleados en los sistemas de recomendaciones, el desarrollo de clientes Android, las bases de datos y el servicio WCF[1].
- Diseñar los componentes que forman parte del sistema de recomendaciones: aplicación Android, servicio WCF, base de datos, y algoritmo de recomendaciones.
- Implementar los componentes del sistema de recomendaciones.
- Analizar los resultados de las pruebas ejecutadas en el sistema de recomendaciones.

[1] WCF (*Windows Communication Foundation*): es un *framework* para la creación de aplicaciones orientadas a servicios.

1.2 Alcance

El prototipo de Sistema de Recomendaciones para Publicidad Inteligente estará compuesto por cuatro componentes: cliente Android, servicio WCF, base de datos y un algoritmo de recomendaciones. Una descripción general del sistema se muestra en la Figura 1.1.

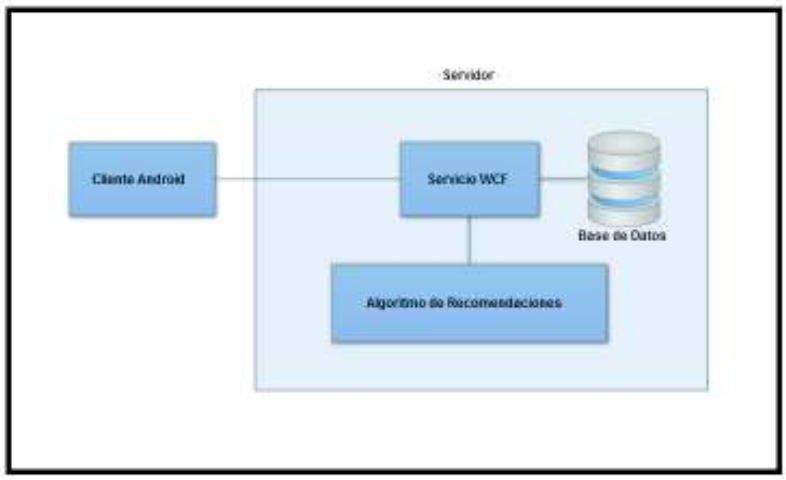


Figura 1.1 Descripción general del sistema

El Cliente Android será responsable de recolectar y enviar información del dispositivo, como la ubicación. Esta información será almacenada en la base de datos en conjunto con la publicidad de los locales comerciales. El cliente Android realizará peticiones de recomendaciones a través del servicio WCF. El algoritmo de recomendaciones usará como entradas la información recolectada por la Aplicación Android, así como la información de los locales comerciales, para que sean procesadas y devolver como salidas las recomendaciones.

Se utilizará el lenguaje de programación C# para el desarrollo del algoritmo de recomendaciones y para resolver tareas básicas en el sistema de recomendación como la clasificación de predicciones y la categorización de ítems [5]. La codificación del servicio WCF también se realizará con el lenguaje de programación C#.

Para establecer los requerimientos de usuarios se realizarán encuestas a 50 personas. De la misma forma, para validar la satisfacción del usuario se realizarán encuestas a 50 personas en la etapa de pruebas.

1.3 Marco Teórico

El objetivo principal de los Sistemas de Recomendaciones (SR) es ayudar a los usuarios en la toma de decisiones. En términos simples un sistema de recomendaciones es un software el cual provee sugerencias para los usuarios [6].

Principalmente los sistemas de recomendaciones cumplen con su cometido cuando se trata de publicar y vender productos. Por ejemplo, un electrodoméstico específico puede ser anunciado como el primer elemento de una lista en una página web debido a que un sistema de recomendación entró en juego y se basó en el enunciado "clientes que compraron esto también comprarán aquello".

La idea puede ir más lejos y se puede utilizar los intereses del usuario de manera que cuando el usuario entre a la tienda, una lista de electrodomésticos aparezca exclusivamente de acuerdo a sus gustos. Por lo tanto, cada cliente observará una lista diferente dependiendo de sus preferencias [7]. En otras palabras, se habla de recomendaciones personalizadas o también llamado publicidad inteligente. La publicidad inteligente puede ser generada de diferentes formas, pero en sí depende del modelo básico del SR empleado.

Dentro de un SR existen dos entidades: usuario e ítem. "Usuario" es la persona que hace uso del SR al proveer su opinión sobre ítems y recibe recomendaciones sobre nuevos ítems del sistema. "Ítem" es el término general usado para denotar qué es lo que se recomienda a los usuarios. Los sistemas de recomendaciones están asociados con un conjunto de ítems $i = \{i_1, i_2, \dots, i_n\}$ y su objetivo es recomendar a los usuarios ítems i que les puedan resultar de interés [8].

Normalmente un SR se enfoca en un tipo específico de ítem (por ejemplo: libros, películas, publicidad, etc.) y acorde a su diseño, a su interfaz gráfica y a su algoritmo de recomendación, las recomendaciones son personalizadas para proveer sugerencias útiles y efectivas para un tipo específico de ítem [9].

1.3.1. Modelos Básicos de los Sistemas de Recomendaciones

Un SR depende de los datos del usuario para generar las recomendaciones, estos datos se clasifican en dos tipos: datos de interacciones usuario-ítem, tales como calificaciones o comportamientos de compra, y datos de información de atributos sobre el usuario e ítems, como perfiles de usuario o palabras relevantes [10]. Según los tipos de datos, los Sistemas de Recomendaciones se clasifican en modelos básicos, los cuales son analizados a continuación.

a. Collaborative Filtering

Esencialmente *Collaborative Filtering* (filtrado colaborativo o CF) hace uso de la colaboración de las calificaciones de múltiples usuarios para identificar aquellos usuarios cuyos gustos sean similares a los gustos del usuario para el cual se va a generar la recomendación.

Por ejemplo, si un usuario *A* y un usuario *B* tienen un historial de compra muy parecidos, y *A* realiza la compra de un producto, el cual no ha sido visto por *B* aún, el razonamiento básico es proponer este producto también a *B* [7]. Por lo tanto los usuarios colaboran entre sí implícitamente para obtener recomendaciones.

Por cada usuario se establece un conjunto de usuarios "vecinos" encontrados a partir de la correlación existente entre sus historiales de calificación de productos pasados. Los nuevos productos a ser recomendados son predichos basándose en una combinación de calificaciones conocidas a partir de los usuarios "vecinos" [11].

Este modelo no requiere de ningún conocimiento sobre los ítems, por lo tanto, la ventaja de emplear CF como modelo básico en un sistema de recomendaciones, es que los atributos de los ítems no tienen que ser ingresados al sistema y mucho menos mantenidos.

b. Content-Based Recommendation

Este modelo de sistema de recomendación cumple su trabajo solo si se cuenta con dos tipos de información: el contenido de la información disponible sobre los ítems y las calificaciones o comportamiento de compra de los usuarios. La tarea de este modelo consiste en determinar aquellos ítems que encajan mejor con las preferencias del usuario. Las recomendaciones se generan a partir del cómputo de similitudes o usando aproximaciones probabilísticas.

Esta técnica brinda independencia de usuario debido a que se explota las calificaciones solamente del usuario para construir su perfil. Ofrece transparencia debido a que se puede entender que un ítem haya sido recomendado analizando sus características o descripciones que fueron parte del proceso de recomendación [12].

Sin embargo, presenta inconvenientes, como el hecho de que el número de características que un ítem puede tener, podría ser limitado, considerando la capacidad de almacenamiento que tenga el sistema, o que la técnica no puede recomendar algo inesperado dado que depende de perfiles y por lo tanto siempre va a recomendar algo similar a lo que se haya calificado por otros usuarios. Un usuario nuevo debe recolectar suficientes calificaciones para que la técnica pueda entender las preferencias de usuario y dar recomendaciones adecuadas [12], [13].

c. Knowledge Based Recommendation

Si se considera a aquellos consumidores que realizan compras esporádicamente, no se puede hacer uso de un historial de compra, lo que es un pre-requisito para los enfoques de

filtrado colaborativo y basados en contenido. Por lo tanto es necesario un enfoque más detallado y estructurado.

Al tener un consumidor que, por ejemplo, realiza la compra de un producto cada varios años, el sistema de recomendaciones no puede proponer un producto con base en lo que otros usuarios han adquirido, lo que resultaría en únicamente vender los productos más vendidos. Por lo tanto, se requiere de un sistema que haga uso del conocimiento que se tenga sobre las características de los ítems para satisfacer las necesidades y preferencias de los usuarios, y finalmente determinar cuán útil es el ítem para los usuarios [7].

El enfoque de este modelo es emplear conocimiento detallado sobre las características de un producto, además se pueden usar restricciones explícitas para describir el contexto en el cual ciertas características son relevantes para los consumidores. Sin embargo, el hecho de presentar productos que únicamente cumplan con un conjunto de requisitos dados no es suficiente, pues se carece del componente de personalización, y cada usuario que tenga el mismo conjunto de requisitos, puede terminar con un mismo conjunto de recomendaciones. Por lo tanto, es necesario que el sistema también mantenga un perfil de usuario.

d. Hybrid Recommendation

Cada modelo utiliza diferentes tipos de información para brindar una recomendación adecuada al usuario. Sin embargo, ninguno de los modelos básicos anteriores es capaz de explotar toda la información. Como consecuencia, se deben construir sistemas que combinen la fuerza de diferentes algoritmos para resolver problemas [7].

Se han identificado dos problemas cuando se utilizan los sistemas de recomendación convencionales, el problema del inicio en frío (*cold-start*) y el problema de ataque-resistencia.

El problema de inicio en frío hace referencia a que los sistemas de recomendaciones inicialmente cuentan con una cantidad de calificaciones relativamente baja y resulta difícil aplicar modelos como *collaborative recommendation system* o *content-based recommendation* [10]. El problema de ataque-resistencia trata sobre el impacto que tiene un SR sobre la venta de productos y servicios, y cómo las calificaciones de los ítems pueden ser modificadas para beneficiar a un vendedor específico.

Estos problemas se solucionan con la recomendación híbrida (*hybrid recommendation*). La metodología para crear el modelo híbrido más común es combinar diferentes modelos, por ejemplo, mezclar *collaborative recommendation* con *content-based recommendation*. Sin

embargo, también es posible mezclar modelos del mismo tipo, pero con grupos de datos diferentes [14].

1.3.2. Algoritmos de los Sistemas de Recomendaciones

Existen varios enfoques que pueden tomar los modelos básicos de Sistemas de Recomendaciones. A continuación se realiza un análisis de los siguientes algoritmos: *Knowledge Discovery in Databases* (KDD), *Web Mining* y *Nearest Neightbour Algorithm*.

a. Knowledge Discovery in Databases (KDD)

Los sistemas de recomendaciones pueden ser vistos como una aplicación de KDD [15]. Este algoritmo ayuda a cumplir dos objetivos: el ahorro de dinero al descubrir el potencial de competencia y el hacer dinero al descubrir formas para vender más productos a consumidores [16].

La definición de *Knowledge Discovery* es la extracción de información no trivial, implícita, previamente desconocida y potencialmente útil de datos. KDD exhibe cuatro características importantes [17]:

- El descubrimiento de conocimiento es representado con un lenguaje de alto nivel.
- Los descubrimientos representan con precisión el contenido de la base de datos.
- El descubrimiento de conocimiento es interesante acorde a los estándares definidos por los usuarios. Que sea interesante implica que la información es potencialmente útil y el proceso de descubrimiento, no trivial.
- El proceso de descubrimiento es eficiente.

En la Figura 1.2 se muestra de forma resumida el proceso KDD. Éste es un proceso interactivo e iterativo, incluyendo varios pasos que se describen a continuación:

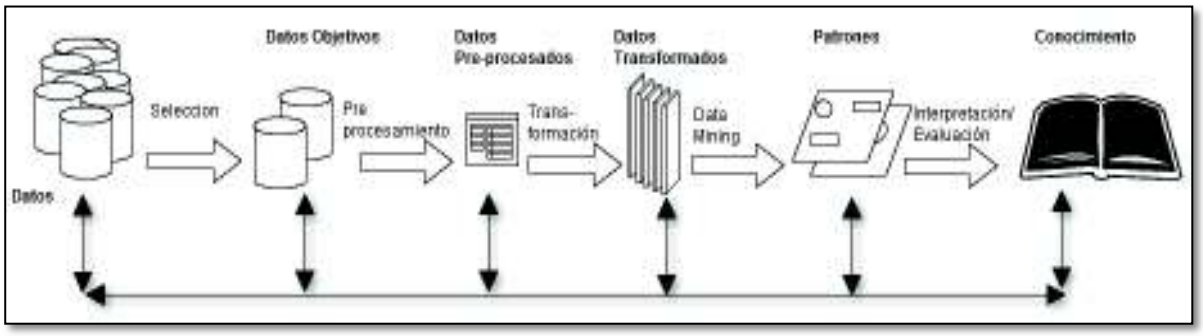


Figura 1.2 Pasos del proceso de KDD [15]

Datos: se adquiere conocimiento relevante sobre el área de estudio y meta a conseguirse.

Selección y datos objetivos: se selecciona un conjunto de datos, un subconjunto de variables o muestras de datos, los cuales serán la fuente de conocimiento.

Pre-procesamiento: incluye operaciones básicas sobre los datos, como la eliminación de ruido o la aislación de partes de ser necesario. Otras tareas son la derivación de nuevos atributos, donde se crean campos con relaciones entre los atributos conocidos, y agrupaciones, donde las relaciones uno a muchos de una base de datos, se pueden convertir en relaciones uno a uno y agregar un campo de conteo o suma, para contabilizar todos los registros de la relación [18].

Transformación: incluye encontrar características útiles para representar los datos, según la meta elegida, y empleando métodos de transformación se reduce el número de variables para encontrar una representación para los datos. Los métodos de transformación se encargan de la discretización y relleno de datos faltantes [19].

Elección de la función de minería de datos: incluye la decisión del propósito del modelo derivado de un algoritmo de *Data Mining.*

Elección del algoritmo de *data mining*: se elige los métodos a ser usados para buscar patrones dentro de los datos, como la decisión de que modelos y parámetros que pueden ser apropiados.

Data Mining: es la búsqueda de patrones de interés en una forma representacional particular o en un conjunto de presentaciones, incluyendo reglas de clasificación, regresiones, *clustering*, secuencias, dependencias o línea de análisis.

Interpretación/Evaluación: consiste en la interpretación de los patrones descubiertos y posiblemente pueda llevar a alguno de los pasos anteriores. También permite la posible visualización de los patrones extraídos, al remover patrones redundantes o irrelevantes, y traducir aquellos que son útiles en términos entendibles por los usuarios.

Conocimiento: uso del conocimiento descubierto. La aplicación de los patrones extraídos puede seguir los siguientes puntos [18]: descripción, predicción e intervención.

b. Web Usage Mining

Web usage mining se refiere al descubrimiento automático y análisis de patrones de secuencia de clics (*clickstream*) y datos asociados, recolectados o generados como resultado de interacciones de usuarios con recursos web en uno o más sitios web. El objetivo es capturar, modelar, y analizar patrones de comportamiento y los perfiles de los usuarios que interactúan con un sitio web. Los patrones descubiertos son usualmente

representados como colecciones de páginas, objetos o recursos que son accedidos por grupos de usuarios con necesidades o intereses comunes [20].

A diario se generan y recolectan cantidades enormes de volúmenes de datos. La mayoría de esta información es usualmente generada por servidores web, y recolectada en forma de *logs*. Otras fuentes de información de usuarios incluyen *logs* de referencias que contienen información acerca de registro de usuarios, datos de encuestas, etc.

Los *logs* contienen enormes cantidades de información útil como las URL[2], direcciones IP[3], fechas, entre otros. El analizar estos datos puede ayudar a encontrar más consumidores en potencia y ofrecer un servicio de calidad. Por ejemplo, cuando las personas visitan un sitio web, dejan cierta información como su dirección IP, páginas visitadas, fecha de visita, etc. *Web usage mining* recolectará, analizará y procesará esa información para posteriormente establecer el comportamiento de los usuarios [20], [21].

El proceso de *usage mining* puede ser dividido en tres etapas independientes como se observa en la Figura 1.3: recolección de datos, pre-procesamiento de datos y descubrimiento de conocimiento [21].

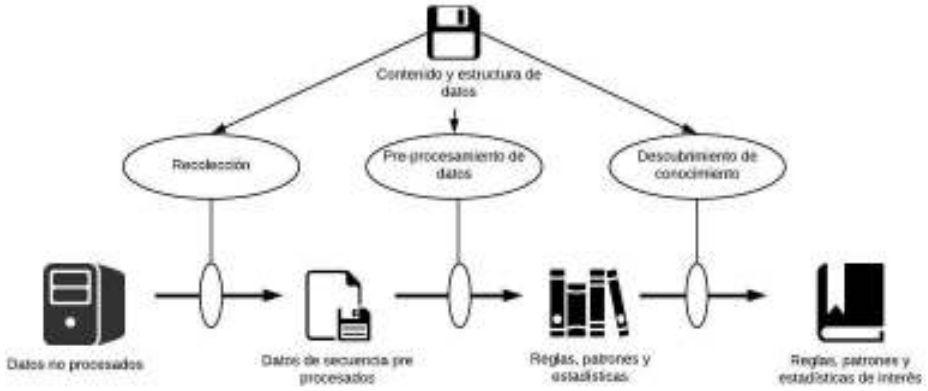


Figura 1.3. El proceso de *Web usage mining* [21]

Las etapas del proceso *Web usage mining* se describe a continuación:

- Recolección de datos: es el primer paso de *web usage mining*, la autenticidad e integridad de datos afectarán las siguientes etapas de manera leve y será de importancia para determinar las características de calidad de un servicio.
- Pre-procesamiento de datos: algunas bases de datos son insuficientes, inconsistentes e incluyen ruido. El pre-tratamiento de datos consiste en llevar a cabo una transformación y unificación para aquellas bases de datos. El resultado es que

[2] URL (*Uniform Resource Locator*): especifica la ubicación de un recurso web en una red, y el mecanismo para obtenerlo.
[3] IP (*Internet Protocol*): es un protocolo para enviar datagramas a través de Internet u otra red.

la base de datos llegará a ser íntegra y consistente. En el trabajo de pre-procesamiento, principalmente se incluyen limpieza de datos, identificación de usuarios, identificación de sesión y complementación de patrones.

- Descubrimiento de conocimiento: en esta etapa se usan métodos estadísticos para llevar a cabo el análisis y la minería de los datos pre procesados. Se puede descubrir los intereses de los usuarios o la comunidad del usuario y luego construir el modelo de interés.

c. Algorithm k-Nearest Neighbor

El algoritmo *k-Nearest Neighbor* (k-NN) es un algoritmo no paramétrico y de aprendizaje perezoso (*lazy*). Al decir no paramétrico, significa que no se realiza ninguna suposición en los datos principales, esto es de gran utilidad ya que, en el mundo real, la mayoría de datos no obedecen ninguna suposición teórica [22]. Aprendizaje perezoso se refiere al hecho de que el algoritmo no construye un modelo hasta el momento en el que la predicción es requerida, se dice que es perezoso porque solamente trabaja al último segundo [23]. El algoritmo se basa en una lista ordenada de los usuarios más similares al usuario al que se desea entregar una recomendación [24]. Esto conlleva a realizar una comparación de un usuario versus el resto de usuarios.

Para explicar el funcionamiento del algoritmo se consideran dos conjuntos de datos, un Conjunto de Entrenamiento (CE) y un Conjunto de Validación (CV). Las muestras que forman parte del CE contienen datos que pertenecen a diferentes clases. Mientras que para las muestras del CV estas clases resultan desconocidas. El algoritmo k-NN calcula la distancia entre cada muestra del CE y todas las muestras del CV. Se pueden emplean diferentes funciones para calcular la distancia, una de ellas es el coeficiente de correlación de Pearson. Una vez realizado el cálculo de la distancia, el algoritmo k-NN toma las k muestras más cercanas en el CE, clasificándolas en orden ascendente y etiquetando estas muestras con el nombre de la clase a la que pertenecen [25].

El valor de k afecta de forma significativa al rendimiento del algoritmo k-NN así como la función usada para calcular la distancia. Cuando se cuenta con muestras de datos que no son uniformemente distribuidos, la elección de k se vuelve difícil. Según [26] se toma el conjunto de entrenamiento y lo convierte a un dominio más pequeño llamado modelo k-NN, en este modelo las muestras del CE son agrupadas con base en la similitud que existe entre ellas. La salida de este modelo está formado por tuplas, que contienen la clase del grupo, la similitud entre la muestra más distante y la muestra central del grupo, y el número de muestras en ese grupo. En este caso no existe necesidad de escoger el mejor valor de k, dado que el número de muestras en cada grupo puede ser visto como el valor óptimo de

k y la razón de esto es que cada grupo tiene sus propias características. Este proceso reduce el tamaño del CE y remueve la necesidad de escoger un valor de k [26].

d. Coeficiente de Correlación de Pearson

El coeficiente de correlación de Pearson (CCP) es utilizado para medir la dependencia o relación lineal de 2 variables. Su resultado se encuentra en el rango de -1 a 1 en donde un valor entre 0 y 1 indica una relación positiva un valor entre 0 y -1 indica una relación negativa y un resultado de 0 indica que no existe relación entre las variables. Con datos reales, no se espera encontrar valores exactos de -1, 0 o 1 [27]. El valor del CCP se calcula con la expresión presentada en la Ecuación 1.1.

$$r = \frac{\sum XY - \frac{(\sum X)(\sum Y)}{n}}{\sqrt{\left(\sum X^2 - \frac{(\sum X)^2}{n}\right)\left(\sum Y^2 - \frac{(\sum Y)^2}{n}\right)}}$$

Ecuación 1.1. Coeficiente de Correlación de Pearson

Donde r es el coeficiente de correlación de Pearson, X y Y son las variables a las cuales se desea encontrar su relación y n es la cantidad de duplas (X, Y) a comparar. Una relación positiva lineal como se muestra en la Figura 1.4, se refiere a que si X incrementa Y también lo hace, de la misma forma si X disminuye Y también.

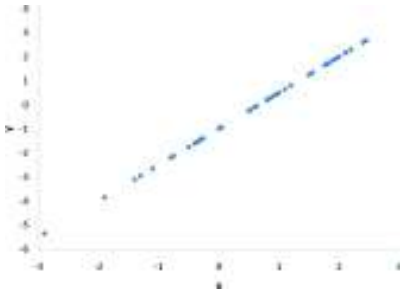


Figura 1.4. Relación positiva lineal

En una relación negativa lineal, como se muestra en la Figura 1.5, si X incrementa, Y disminuye, y si X disminuye, Y aumenta.

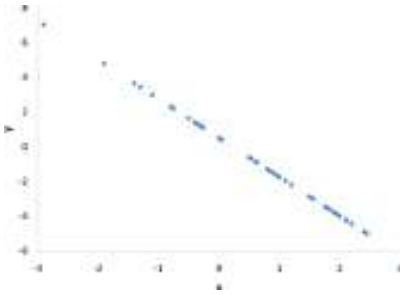


Figura 1.5. Relación negativa lineal

En la Figura 1.6 se observa que gráficamente no se puede establecer ninguna relación entre las variables X y Y.

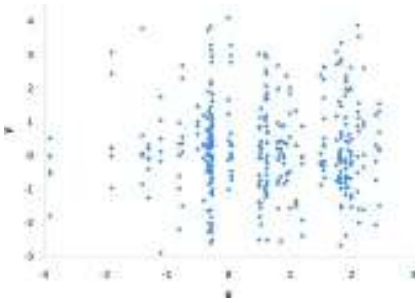


Figura 1.6. Sin relación

1.3.3. Desarrollo de aplicaciones Android

Para el desarrollo de aplicaciones Android para dispositivos, se requiere de un SDK (*Software Development Kit*), el cual contiene una serie de utilidades como herramientas de programación, compilación y depuración, y un emulador Android para probar la aplicación.

a. Android SDK

Android SDK es un conjunto de herramientas de desarrollo usado para el desarrollo de aplicaciones para plataformas Android.

Cada vez que Google libera una nueva versión de Android, un correspondiente SDK es también lanzado. Para que haya la posibilidad de escribir programas con las últimas características, los desarrolladores deben descargar e instalar cada versión de SDK para un teléfono en particular.

Las plataformas de desarrollo que son compatibles con SDK incluyen sistemas operativos como Windows (XP o superiores), Linux (cualquier versión reciente) y Mac OS X (10.4.9 o superior).

A pesar de que el SDK puede ser usado para escribir programas Android con el símbolo de sistema (*command prompt*), el método más común es usando un entorno de desarrollo integrado. El IDE[4] más empleado es Eclipse con el *plug-in* de las Herramientas de Desarrollo de Android (ADT).

Sin embargo, existen otros IDE, como NetBeans o IntelliJ que también pueden usados. La mayoría de estos IDE proveen una interfaz gráfica permitiendo a los desarrolladores realizar tareas de desarrollo eficientemente.

[4] IDE (*Integrated Development Enviroment*): software el cual provee de un ambiente de programación para optimizar el desarrollo y depuración de software

b. Android Studio

Android Studio es el IDE oficial desarrollado por Google, basado en IntelliJ y presenta las siguientes características:

- Compilación basada en *Gradle*
- Tiene un emulador rápido
- Consta de un entorno unificado en el que se pueden desarrollar aplicaciones para cualquier dispositivo Android
- *Instant Run* que permite realizar cambios en tiempo de ejecución
- Incluye una vasta cantidad de herramientas y *frameworks* de prueba

Los proyectos creados en Android Studio contienen uno o más módulos con archivos fuente o archivos de recursos. Estos módulos son:

- Módulos de aplicaciones para Android
- Módulos de bibliotecas
- Módulos de *Google App Engine*

Cada módulo de la aplicación contiene las siguientes carpetas:

- `manifests`: contiene el archivo `AndroidManifest.xml`.
- `java`: contiene los archivos de código fuente de Java.
- `res`: Contiene todos los recursos, como archivos XML[5] (*Layouts*), imágenes de mapa de bits.

A continuación se definen algunos términos importantes que se emplean en el desarrollo de aplicaciones con Android Studio:

Gradle: es una herramienta para automatizar el proceso de compilación de proyectos. Google creó este sistema de manera amigable para los desarrolladores de Android de tal forma que estos se enfoquen en realizar sus proyectos, en vez de preocuparse en la construcción del ambiente de desarrollo [28]. *Gradle* en Android Studio facilita la inclusión de librerías a la compilación en forma de dependencias. Estas dependencias pueden ubicarse dentro de la máquina en donde se realiza la compilación o en un repositorio remoto. Cuando se emplean dependencias de un repositorio remoto, se debe añadir en el archivo `build.gradle` el repositorio que se va a emplear, por ejemplo *Maven* o *Ivy* según se lo requiera.

[5] XML (*Extensible Markup Lenguage*): es un lenguaje que define un conjunto de reglas para la codificación de documentos en un formato que puede ser leído por humanos y maquinas.

Actividades: una actividad representa una pantalla donde el usuario puede realizar acciones de forma interactiva, como ingresar información, observar un mapa, tomar una foto, etc. Por lo general una aplicación está conformada por varias actividades, en donde se tiene una actividad principal la cual se presenta por primera vez cuando el usuario inicie la aplicación. El orden en el que son presentadas las actividades depende de cómo esté codificada la lógica de la aplicación. Además es posible pasar datos de una actividad a otra según se requiera.

Layout: un *layout* define la estructura visual para una interfaz de usuario. Un *layout* se puede declarar de dos maneras: declararlo en XML, ya que Android provee un extenso vocabulario XML o instanciar *layouts* durante el tiempo de ejecución. La ventaja de declarar la interfaz de usuario en XML es que abre la posibilidad de separar de mejor manera la presentación de la aplicación del código, lo que significa que el momento que se desee modificar la presentación no hay necesidad de modificar el código fuente y volver a compilar. Usar esta manera de declaración también ayuda a depurar problemas más fácilmente.

La estructura de la interfaz de usuario es definida en archivos XML que se encuentran en el directorio `res/layout`. Cada ventana tiene su propio código XML, escrito en un archivo. Estos archivos representan un *layout* (una ventana) y cada *layout* puede contener otros elementos [29]. Una actividad está asociada con un *layout*.

Vistas: una vista es un objeto de la clase `android.view.View`, la cual contiene información sobre el área de la ventana, y permite establecer el *layout*. Las vistas son útiles para el uso de *widgets*[6], que son subclases ya implementadas que dibujan los elementos en las ventanas. Entre la lista de *widgets* que se pueden implementar se tienen elementos `TextView`, `EditText`, `Button`, `CheckBox` [29].

Adaptadores: un adaptador es una clase que actúa como puente entre un elemento de la interfaz gráfica y la fuente de los datos, para llenar con datos el elemento gráfico. Un adaptador provee de acceso a los datos de los elementos. Un adaptador mantiene los datos y los envía a una vista adaptador, para mostrar la información en diferentes vistas como `GridView`, `ListView`, `Spinner`, etc. Una ventaja de los adaptadores es que pueden ser personalizados [30].

1.3.4. MySQL

MySQL es la base de datos de código abierto más popular [31]. Una base de datos es una aplicación separada que almacena una colección de datos. Cada base de datos tiene una

[6] *Widgets*: son elementos de interacción como un botón, o una barra de desplazamiento.

o más API (*Application Programming Interface*) para crear, acceder, administrar, buscar y replicar los datos que mantiene. Otros tipos de almacenamiento de datos pueden ser usados, como sistemas de archivos o grandes tablas en memoria, pero la recuperación y escritura no sería tan fácil y rápida en estos sistemas [32].

A continuación se definen algunos términos usados en MySQL [32]:

- Base de datos: es una colección de tablas, con datos relacionados.
- Tabla: es una matriz con datos.
- Columna: contiene datos de un único tipo.
- Fila: es un grupo de datos relacionados.
- Llave Primaria: es un campo o conjunto de campos que identifica de manera única a cada fila de una tabla. El valor de una llave no puede aparecer dos veces en una tabla. Con una llave, se puede encontrar una sola fila.
- Llave Foránea: es un campo o conjunto de campos en una tabla que identifica de manera única las filas de otra tabla.
- Integridad Referencial: la integridad referencial asegura que un valor de llave foránea siempre apunte a una fila existente.

Actualmente se emplean Sistemas de Gestión de Bases de Datos Relacionales (RDBMS), para almacenar y administrar grandes volúmenes de datos. Se dice que son relacionales, dado que los datos son almacenados en diferentes tablas, y se establecen relaciones entre ellas usando llaves primarias u otras llaves conocidas como llaves foráneas. Una RDBMS es un software que permite implementar una base de datos y garantizar la integridad referencial entre filas de varias tablas, interpretar peticiones SQL y combinar la información de varias tablas [32].

Las bases de datos son ampliamente usadas en varias aplicaciones para mantener la información de los clientes, sus preferencias, e historiales. MySQL se ha convertido en la opción líder en términos de bases de datos, para aplicaciones web, incluyendo Facebook, Twitter, YouTube [31], y en varias aplicaciones en donde se requieren bases de datos embebidas.

1.3.5. SQLite

SQLite es una librería en proceso que implementa un motor de base de datos SQL autónomo, sin servidor, con cero configuraciones y transaccional [33]. SQLite es un motor de base de datos SQL embebido. El código para SQLite es de dominio público y es libre para uso de cualquier propósito, ya sea comercial o privado.

A diferencia de la mayoría de bases de datos SQL, SQLite no tiene el proceso servidor por separado. SQLite lee y escribe directamente en discos de archivos ordinarios. SQLite es una base de datos SQL completa con múltiples tablas, contenida en un solo disco de archivos.

Algunas características de SQLite pueden ser omitidas, permitiendo reducir aún más el tamaño de esta librería, de tal forma que SQLite puede representar la mejor opción de motor de base de datos para un dispositivo de memoria limitada como celulares, PDA[7], y reproductores MP3[8].

SQLite ha sido cuidadosamente probado antes de cada lanzamiento y tiene una reputación de ser muy confiable. Los desarrolladores continúan ampliando las capacidades de SQLite y aumentando su confiabilidad y desempeño manteniendo la compatibilidad hacia atrás con la especificación de interfaz publicada, la sintaxis SQL y el formato de archivo de la base de datos.

Entre las características más notables de SQLite se tienen [34]:

- Cero configuraciones: SQLite no necesita ser instalado antes de su uso, no hay la necesidad de arrancar, detener o configurar ningún proceso servidor.
- Archivo de base de datos único: una base de datos SQLite es un único archivo de disco duro que puede ubicarse en cualquier parte de la jerarquía de directorios.
- Archivo de base de datos multiplataforma estable: el formato de archivo SQLite es multiplataforma. Un archivo de base de datos escrito en una máquina se puede copiar y utilizar en una máquina diferente con una arquitectura diferente.

1.3.6. Servicio WCF

El servicio WCF (*Windows Communication Foundation*) es un programa que expone una colección de *endpoints*. Cada *endpoint* es un portal para la comunicación con los clientes finales. Diferentes *endpoints* pueden ser creados con un puerto diferente el archivo de configuración del lado del cliente. Todas las comunicaciones WCF toman lugar a través de los *endpoints*. Un *endpoint* está constituido por 3 elementos [35]:

- *Address:* dirección del servicio. Esta es la dirección a la cual los mensajes deben ser enviados por los clientes, de tal forma que sean hospedados por el servicio.

[7] PDA (*Personal Digital Assistant*): es un dispositivo móvil que funciona como un administrador personal de información.
[8] MP3: es un formato de codificación de audio para audio digital.

- *Binding:* define el canal usado para comunicarse con un *endpoint*. El elemento de nivel más bajo que tiene es el protocolo de transporte, el cual se encarga de pasar los mensajes dentro de una aplicación WCF. Ejemplos de protocolos son: TCP[9], HTTP[10], *Named Pipes*[11], y MSMQ[12].
- *Contract:* define la capacidad, o conjunto de características ofrecidas por un *endpoint*. El contrato define las operaciones que un *endpoint* expone al cliente para aprovechar el servicio y el formato de los mensajes que es requerido para las operaciones. Las operaciones de contrato usan métodos de una clase, que son implementados por el *endpoint,* incluyendo la firma de parámetros pasados adentro y afuera de los métodos.

a. Message Exchange Patterns (MEP)

WCF ofrece varias maneras de intercambiar mensajes entre un cliente y un servicio, frecuentemente denominadas Patrones de Intercambio de Mensajes (MEP). El tipo de operación utilizada para comunicarse con el servicio es parte del servicio, un MEP específico puede colocar algunas restricciones en los vínculos permitidos ya que no todos los enlaces WCF realmente apoyan a todos los MEP disponibles. A continuación, se describen algunos MEP que WCF soporta:

Request – Reply: es el modo de operación por defecto de WCF. Usando este MEP el cliente realiza una llamada al servicio en forma de mensaje y se bloquea hasta que se obtenga una respuesta. Si el servicio no responde dentro de un rango de tiempo determinado, el cliente recibirá un mensaje de "*Timeout-Exception*". El uso de la operación *Request – Reply* es muy simple y se asemeja a la programación bajo el modelo de cliente servidor.

One Way: en el caso de que una operación no tenga un valor de retorno y al cliente no le importe acerca del éxito o fallo de la llamada, WCF ofrece operaciones de un solo sentido o una sola vía para apoyar este tipo de llamadas. A diferencia de llamadas del tipo *Request – Reply*, las llamadas de una sola vía usualmente bloquean el cliente solo por el momento que dure enviar la llamada. Como no se genera un mensaje de respuesta por WCF, los valores no regresan al cliente.

---

[9] TCP *Transmission Control Protocol* es un protocolo de capa transporte, confiable, orientado a la conexión.
[10] HTTP *Hypertext Transfer Protocol* es un protocolo de capa aplicación, para transmitir datos entre Servidores y clientes Web.
[11] *Named Pipes*: es un protocolo desarrollado para redes de área local. Permite pasar información de un proceso de una computadora, a otros procesos.
[12] MSMQ *Microsoft Message Queuing* es un protocolo de mensajería que permite correr aplicaciones en procesos o servidores separados para una comunicación prueba de fallas.

Callback – Duplex: usar comunicaciones WCF dúplex permite al servicio llamar de regreso a sus clientes e invocar un método de cliente. Las operaciones de llamadas de vuelta son especialmente útiles cuando se producen eventos y se necesita notificar al cliente de la ocurrencia de los mismos. Sin embargo, para habilitar este servicio en el lado del cliente, se tiene que conocer el *endpoint* del cliente. Por lo tanto, es necesario que el cliente haga un llamado a un método de servicio primero, que reserva el canal de llamada de retorno al *endpoint* del cliente para uso posterior. A través de ese canal es posible que el servicio envíe mensajes al cliente y permite invocar métodos específicos.

b. Streaming

Cuando el cliente y el servicio intercambian mensajes, estos se almacenan en el lado del receptor y son entregados solo cuando el mensaje entero haya sido recibido. Esto significa, que el cliente se desbloquea solo si el mensaje de petición (*request*) y el mensaje de respuesta (*reply*) del servicio hayan sido enviados y recibidos completamente. Esto funciona perfectamente para mensajes pequeños. Sin embargo, para mensajes grandes sería impráctico el bloquear hasta que el mensaje completo haya sido recibido. Por lo tanto, WCF habilita el lado del receptor para empezar a procesar los datos recibidos mientras el mensaje continúa siendo recibido. Este tipo de operación se denomina modo de transferencia de tipo *streaming*, y mejora el rendimiento y la capacidad de respuesta de mensajes con grandes *payloads*. Esta comunicación se realiza a través de llamadas: asíncronas y en cola.

Llamadas asíncronas: con las llamadas asíncronas el cliente no se bloqueará y controlará las respuestas inmediatamente después de que se haya emitido la solicitud. El servicio luego ejecuta la operación en segundo plano. Tan pronto como la operación se complete, el cliente es provisto con resultados de la ejecución del mismo. Las llamadas asíncronas mejoran las respuestas y disponibilidad del cliente.

Llamadas en cola: los mensajes en cola usan *Microsoft Message Queue* (MSMQ). WCF encapsula cada mensaje del tipo SOAP[13] en un mensaje MSMQ y lo coloca en la cola designada. Además, el cliente no se comunica directamente con un servicio sino con la cola del mensaje. Como resultado, todas las llamadas son inherentemente asíncronas y desconectadas. En el lado del servicio, los mensajes en cola son detectados por el oyente de la cola, que toma secuencialmente los mensajes de la cola y los envía a una instancia de servicio.

[13] SOAP (*Simple Object Access Protocol*): es un protocolo para intercambiar información estructurada en la implementación de servicios Web en una red de computadoras.

c. Hosting

Para correr y acceder a un servicio WCF es necesario, que el servicio sea hospedado en un ambiente apropiado denominado *hosting* [35]. Este ambiente consiste de un proceso llamado proceso *host*. Un único proceso *host* puede hospedar múltiples servidores y un mismo tipo de servicio puede ser hospedado en múltiples procesos *host*. El único requerimiento del *host* es soportar dominios de aplicación .NET[14]. Por esta razón se consideran cuatro métodos de *hosting*:

- IIS: dado que el Servicio de Información de Internet (*Internet Information Services*) es usado principalmente como servidor web, únicamente soporta HTTP como protocolo de transporte. La ventaja de usar IIS es que posee varias características de administración [36] como: monitoreo de estado, modo reposo, reciclado de procesamiento, entre otras.
- *Self-hosting:* cuando un servicio se hospeda en una aplicación administrada, se denomina *self-hosting*. las aplicaciones administradas son escritas por el desarrollador, esto incluye aplicaciones graficas como *Windows Forms* y aplicaciones de consola.
- WAS: el Servicio de Activación de Windows (WAS[15]) tiene las mismas características de IIS, pero al contrario de IIS, WAS soporta todo tipo de protocolo de transporte, incluyendo HTTP, TCP, *Named Pipes* y MSMQ.
- Servicio de Windows: para clientes locales resulta mejor el hospedar un servicio WCF como un servicio Windows [36]. Todas las versiones de Windows soportan este tipo de *hosting*, y es mantenido por el Administrador de Control de Servicios (SCM[16]).

d. Patrón Suscripción - Publicación

El Patrón Suscripción - Publicación es un patrón de diseño frecuentemente usado en conexión con WFC de tal manera que habilite publicaciones a clientes basadas en eventos a través de la red. Esto permite a un servicio informar a sus clientes que algo ha pasado en él. Es necesario las comunicaciones MEP dúplex para que este patrón funcione. Las llamadas de operación de una sola vía también se pueden utilizar, pero son opcionales [37].

La Figura 1.7 (paso 1) muestra el patrón se basa en clientes que se suscriben al servicio. Luego el servicio almacena cada canal de respuesta de llamada de cada cliente, en una

[14] .NET es un *framework* gratuitito, multiplataforma, para el desarrollo de varios tipos de aplicaciones.
[15] WAS *Windows Activation Service* administra procesos que contienen aplicaciones que albergan servicios WCF.
[16] SCM *Service Control Manager* es un proceso el cual inicia, detiene e interactúa con procesos de Windows.

lista. Cuando un evento ocurre, el servicio publica a todos los clientes que se encuentran en la lista (Ver Figura 1.7, paso 3). Sin embargo, en la mayoría de casos el origen de eventos con gran frecuencia no es el mismo servicio sino otro cliente o un servicio externo. En esta situación, éste patrón opera como un distribuidor. El cliente que causó el evento publica que un evento sucedió al distribuidor (Ver Figura 1.7, paso 2) y el distribuidor informa al resto de clientes. Si un cliente ya no está interesado en recibir notificaciones simplemente deja de suscribirse al servicio, que significa eliminar al cliente de la lista de llamadas de retorno [37].

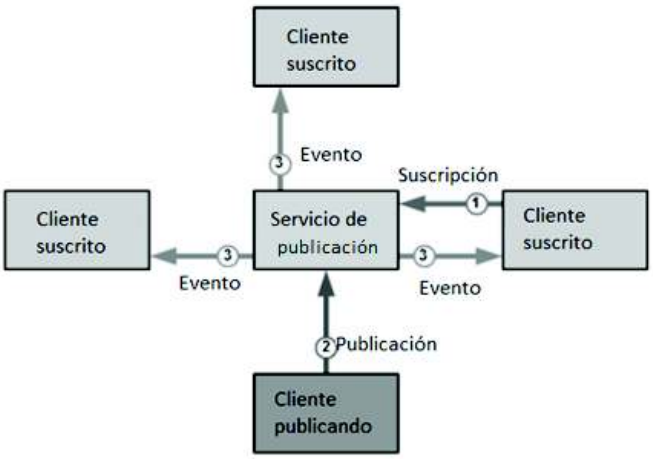


Figura 1.7. Esquema básico del patrón suscripción - publicación [37]

e. Descubrimiento del servicio WCF

A fin de que un servicio pueda ser accedido, es necesario que el cliente conozca su dirección. Una forma, sería incluir todas las direcciones relevantes del servicio en el código del cliente o en los archivos de configuración. Sin embargo, esto requiere que se trabaje estrechamente entre el cliente y el servicio. Para evitar esto, se han desarrollado varios métodos que permiten al cliente encontrar de forma dinámica al servicio y su dirección. Uno de estos métodos es *WS*[17]*-Discovery*. *WS-Discovery* es una especificación de WS de bajo peso, para encontrar servicios en la red. *WS-Discovery* usa SOAP y UDP[18] *multicast*[19] para encontrar servicios en la red.

Existen cuatro escenarios en los que actúa *WS-Discovery* y cada uno tiene su propio mensaje [37]:

- *Hello:* en cuanto el servicio arranca, este envía un mensaje *hello* usando UDP *multicast* para informar a otros participantes sobre su disponibilidad.

[17] WS (*Web Service*): es un servicio ofrecido por un dispositivo electrónico a otro, a través de Internet.
[18] UDP (*User Datagram Protocol*): es un protocolo de capa transporte, no confiable, no orientado a la conexión.
[19] *Multicast*: es una comunicación entre un solo transmisor y un grupo de receptores en una red.

- *Bye*: cuando se va a detener el servicio, este envía un mensaje *bye* para indicar que ya no se encuentra disponible.
- *Probe*: un cliente interesado en un servicio en específico, envía un mensaje *probe,* que contiene información acerca del tipo de servicio del cual quiere hacer uso.
- *Resolve:* para localizar un *endpoint* y su dirección, el cliente envía un mensaje *Resolve* que contiene el nombre del servicio.

f. Peticiones HTTP

El cliente realiza un pedido HTTP al servidor. El servidor realiza diferentes actividades según la petición del cliente, y envía de regreso un mensaje como respuesta. Esta respuesta contendrá un estado completo de la solicitud del cliente y también lo que el cliente solicitó dentro del cuerpo de la respuesta. Este proceso se lleva a cabo gracias a peticiones HTTP. El conjunto común de métodos de peticiones de HTTP es: GET, POST, PUT, DELETE, entre otros.

GET sirve para obtener información de un servidor, es decir para transportar los datos que están en el servidor al cliente. POST envía información desde el cliente para que sea procesada y realice cambios en la información en el servidor.

Ambos métodos solicitan una respuesta del servidor y ahí es donde parece que los conceptos son iguales ya que con ambos se podría lograr los mismos objetivos. Se puede enviar por GET ciertos datos en la URL y colocar información en una base de datos, pero eso le correspondería al método POST. De la misma manera se podría solicitar una página diferente por medio de POST y simplemente mostrarla como respuesta, aunque eso debería ser a través de una llamada GET.

Generalmente se emplean enlaces para ejecutar llamadas GET ya que la idea del enlace es simplemente solicitar una información (página) al servidor y que sea devuelta como una respuesta. Mientras que cuando se usan formularios para actualizar datos de productos, clientes, noticias, etc. se puede enviar más cantidad de datos empleando el método POST que por GET.

1.3.7. Distancia entre dos puntos dados por latitud y longitud

La distancia entre dos puntos dados por latitud y longitud es calculada mediante la fórmula de Haversine presentada en la Ecuación 1.2.

$$Distancia = R * c$$

Ecuación 1.2. Fórmula de Haversine

Donde R es el radio de la Tierra en kilómetros y su valor es 6373, y c se calcula con la expresión Ecuación 1.3.

$$c = 2 * \arctan\left(\frac{\sqrt{A}}{\sqrt{1-A}}\right)$$

Ecuación 1.3. Cálculo de $\boldsymbol{c}$

Donde A se calcula con la Ecuación 1.4.

$$A = \sin^2\left(\frac{\Delta Latitud}{2}\right) + \cos\left(Longitud_{final}\right) * \cos\left(Longitud_{origen}\right) + \sin^2\left(\frac{\Delta Longitud}{2}\right)$$

Ecuación 1.4. Cálculo de $\boldsymbol{A}$

En donde $\Delta Latitud$ y $\Delta Longitud$ se calcula con las Ecuaciones 1.5 y 1.6 respectivamente.

$$\Delta Latitud = Latitud_{final} - Latitud_{origen}$$

Ecuación 1.5. Cálculo de $\Delta\boldsymbol{Latitud}$

$$\Delta Longitud = Longitud_{final} - Longitud_{origen}$$

Ecuación 1.6. Cálculo de $\Delta\boldsymbol{Longitud}$

2. METODOLOGÍA

El presente proyecto técnico se desarrollará empleando una investigación de tipo aplicada, en la cual se emplearán los conocimientos adquiridos respecto al desarrollo de aplicaciones Android, servicios WCF, bases de datos y algoritmos de recomendaciones para el desarrollo de los componentes del prototipo, el cual presentará recomendaciones de publicidad de acuerdo a la localización y preferencias del usuario.

Se analizarán sistemas de recomendaciones existentes para determinar el modelo básico que se empleará. Se recolectará información de personas mediante encuestas para determinar los requerimientos de usuario, y mediante la observación de características de locales comerciales se determinará el tipo de información que se recogerá sobre los mismos, con lo cual se establecerán los requerimientos de usuario.

La metodología de desarrollo de software del prototipo se conforma de dos fases, una fase de diseño, y una fase de implementación.

2.1. Diseño del prototipo

Para el diseño del prototipo se empezará con el estudio de sistemas de recomendaciones usados por Amazon, Netflix, Google y Facebook [10]. Se establecerán los requerimientos de usuario mediante 50 encuestas y se determinará la información de publicidad que se almacenará en la base de datos. Parte de la etapa de diseño será la realización de *sketches* de las interfaces graficas del cliente Android, así como diagramas de clases, diagramas de casos de uso, diagramas de entidad-relación, y diagramas de interacción. Los diagramas de clases de cada uno de los componentes del prototipo se encuentran con mayor detalle en el Anexo X.

2.1.1. Recomendaciones de amistad de Facebook

Los algoritmos de recomendaciones empleados por Facebook son privados, sin embargo, según [38], de entre los varios métodos empleados para realizar una recomendación, uno de ellos es amigos de amigos (*Collaborative Filtering*). Este método presenta una fácil implementación y una gran efectividad.

El método amigos de amigos se basa en el hecho de que, si dos usuarios comparten una gran cantidad de amigos en común, estos poseen un gran potencial de convertirse en amigos también. Este método también es conocido como vecinos comunes y a medida que las redes sociales han ido creciendo, se han ido desarrollado varios algoritmos basados en este método.

Las redes sociales definen las relaciones entre los usuarios usando la figura G [39], por lo que una red social puede representarse por $G(V, E)$, siendo V los vértices o nodos, y E el conjunto de conexiones, enlaces o bordes, tal como se muestra en la Figura 2.1.

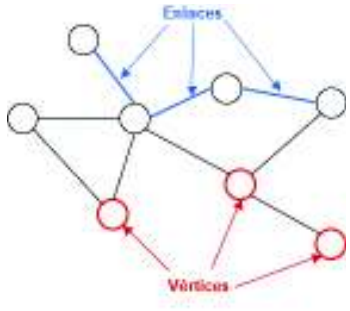


Figura 2.1. Figura G de una red social

Dos vértices son llamados vecinos si se encuentran conectados por un enlace. En una red social, se puede asignar un peso W a cada enlace, donde W representa la fuerza de la conexión entre dos vértices. El peso representa una parte crucial en una red social; un peso bajo, representa un usuario cuyo número de interacciones con otros usuarios es relativamente bajo en contraste con otros usuarios, de tal forma que este usuario no tendría una percepción completa de los beneficios que ofrece la recomendación de amigos en una red social. A continuación se presentan tres diferentes formas de alcanzar la recomendación de amistad [39]:

a) Un algoritmo basado en contenido que considera: atributos demográficos como edad, sexo, trabajo, residencia, intereses del usuario como películas, series, música. y contexto como ubicación GPS[20], hora, clima, etc. Según estos atributos se puede establecer un nivel de similitud entre usuarios de tal forma que se pueda realizar una recomendación de interés.
b) Los algoritmos basados en *Collaborative Filtering* pueden ser usados para calcular la similitud entre usuarios que compartan mismos intereses.
c) Se pueden calcular recomendaciones basadas en la figura G, de tal manera que es posible calcular los pesos en función de la cantidad de amigos comunes.

2.1.2. Sistema de recomendaciones de Netflix

Netflix es el proveedor líder de servicio por suscripción para películas y series de TV, maneja más de 23 millones de suscriptores en Estados Unidos y Canadá, y puede realizar *streaming* de video en calidad HD (*High Definition*) a una tasa promedio de 3.6 Mbps[21] [40].

[20] GPS (*Global Positioning System*): es un sistema de navegación basado en satélites capaz de proveer de la información necesaria para que un dispositivo GPS determine su ubicación geográfica.
[21] Mbps (*Megabits per second*): es una unidad de medición del *throughput* de una red.

El servicio de vídeo en *streaming* utiliza algoritmos de *machine learning* que tienen la finalidad de ir más allá de las ideas preconcebidas de los usuarios para poder ofrecerles alternativas que, aunque no habían sido considerados al principio, podrían ser de su agrado. Cabe mencionar que Netflix usa varios algoritmos que en conjunto forman el sistema de recomendaciones como tal. Mediante este proceso, la plataforma etiqueta a cada espectador en una categoría de gustos entre las 2.000 que tienen tipificados, y en función de esta categoría, el sistema de recomendaciones muestra unos contenidos u otros.

Un elemento importante en la personalización de Netflix es la consciencia, es decir, se requiere que los miembros estén conscientes que el sistema se adapta a sus gustos. Esto no solo incrementa la confianza en el sistema, pero también promueve a los usuarios a dar una retro alimentación, lo que mejorará las recomendaciones. Otra forma de mejorar la confianza, es a través de explicaciones al por qué se decide recomendar una película o serie al usuario [41].

Con respecto a la clasificación de recomendaciones, esta se basa en la popularidad de ítems, y la razón para usar este esquema es simple, en promedio, es altamente probable que un usuario vea lo que la mayoría de usuarios también ve. Sin embargo, la popularidad es lo opuesto a la personalización, en este escenario se mostraría una misma lista de ítems para cada usuario. Por lo tanto, se busca encontrar una función de clasificación personalizada, de tal forma que cada usuario se vea satisfecho con las recomendaciones. Un enfoque para alcanzar esto es el uso de clasificaciones predichas en conjunto a la popularidad de ítems.

2.1.3. Sistema de recomendaciones de Amazon

Amazon es conocido por ser el centro comercial más grande del Internet y como tal posee un catálogo de productos enorme. Por ende, es necesario un sistema de recomendaciones que ayude a los usuarios a encontrar productos que de otro modo probablemente no encontrarían. Amazon cuenta con un algoritmo personalizado de recomendaciones denominado filtrado colaborativo ítem a ítem (*item-item Collaborative Filtering*). En resumen, este algoritmo asocia cada producto adquirido por un usuario con una lista de productos similares que se obtiene en función de los elementos que hayan sido adquiridos en un mismo pedido, añadidos al carro de compras o al conocido *wish list*.

El filtrado colaborativo ítem a ítem, se enfoca en encontrar ítems similares, no clientes similares. Por cada ítem escogido o valorado por el usuario, el algoritmo intenta encontrar ítems similares. Luego agrega ítems similares y los recomienda. Un algoritmo de *Collaborative Filtering* tradicional representa un usuario como un vector de ítems N-

dimensional, dónde N es el número de diferentes catálogos de ítems. Los componentes del vector son positivos cuando son escogidos para una compra o cuando el cliente da una valoración positiva y son negativos cuando el cliente da una valoración negativa.

Para compensar los artículos más comercializados, el algoritmo multiplica los componentes del vector por la frecuencia inversa (la inversa del número de clientes que han comprado o calificado el artículo), haciendo que los artículos menos conocidos sean mucho más relevantes.

El algoritmo genera recomendaciones basado en pocos clientes que son lo más similares posibles al usuario. Comúnmente se puede medir cuan similar son dos usuarios A y B midiendo el coseno del ángulo entre dos vectores. El algoritmo puede seleccionar recomendaciones de clientes similares usando diferentes métodos, pero una técnica comúnmente usada es valorar cada ítem de acuerdo a la cantidad de clientes que escogieron dicho ítem [46].

Pero el usar *Collaborative Filtering* para generar recomendaciones es demasiado caro en términos de cómputo. Es aquí donde el filtrado colaborativo basado en ítems entra en juego. *Collaborative Filtering* era basado en el usuario, lo que quiere decir que el primer paso era buscar entre otros usuarios para encontrar personas con los mismos intereses, luego buscar de los ítems que esas personas similares encontraron, aquellos ítems que el usuario no ha encontrado aún. No obstante, el filtrado colaborativo ítem a ítem empieza por encontrar ítems que se relacionan entre sí dentro de cada catálogo. Relación en este caso que se define como: “personas que compraron un ítem probablemente pueden comprar otro”. De tal forma que para cada ítem, se busca otros ítems que fueron escogidos con una frecuencia inusualmente alta por personas que compraron el primer ítem.

Después de que este vector de ítems relacionados se construye, se generan recomendaciones rápidamente como una serie de búsquedas. Para cada artículo que es parte del contexto actual de este cliente y sus intereses previos, se buscan artículos relacionados, se combinan para obtener los elementos de interés más probables, se filtran los artículos ya vistos o comprados, y luego quedan los artículos para recomendar. Adicional a esto, la mayoría del cómputo se realiza *offline* y el cálculo de la recomendación se realiza en tiempo real a partir de una serie de búsquedas [6], [10].

2.1.4. Sistema de recomendaciones de noticias de Google

El desafío del sistema de recomendación de noticias de Google es ayudar a los usuarios a encontrar artículos que son interesantes para leer, debido al gran volumen de artículos

existentes en todo el mundo. Google *News* utiliza *Collaborative Filtering* para realizar las recomendaciones a sus usuarios.

Las noticias de Google es un sitio web de noticias generadas por computadora que agrega encabezados a partir de fuentes a nivel mundial. Clasifica los sinnúmeros de artículos de acuerdo a diferentes categorías, por ejemplo, deportes, entretenimiento, entre otros y los muestra en sus correspondientes secciones. Cada sección contiene los 3 primeros encabezados de cada categoría. Cuando se ingresa como usuario con una cuenta de Google, se presenta una opción para almacenar el historial de búsquedas y clics realizados de tal manera que luego se pueda acceder más fácilmente a dichas noticias. Además, se presenta una opción en la que se recomiendan noticias que han sido recomendadas para el usuario, basadas en dicho historial.

El Internet nunca tiene escasez de contenido. El desafío es encontrar el contenido adecuado para cada usuario, que pueda estar interesado en leer. Sin embargo, en muchas de las ocasiones ni el mismo usuario sabe lo que busca. Este es el caso de noticias, películas, entre otros, y es por esto que el usuario termina buscando en sitios como `news.google.com` cosas que pueda ser de interés con la idea "Muéstrame algo interesante". Este es el caso ideal en el que se presentan recomendaciones a un usuario basado en sus intereses y sus actividades pasadas de sitios relevantes.

El uso de *Collaborative Filtering* debe acoplarse de tal forma que cumpla los requerimientos de Google *News* como escalabilidad ya que diariamente el sitio web es visitado por millones de personas y el número de noticias es del orden de millones de artículos. O considerar la rotación de artículos, ya que los ítems que se tienen que recomendar van cambiando en cuestión de minutos, dado que las historias que son de interés son aquellas que sucedieron hace un par de horas. Por lo tanto, cualquier modelo viejo de hace muchas horas atrás puede ya no ser de interés. Por esto, el sistema de recomendaciones de noticias de Google es una mezcla de algoritmos (PLSI[22], MinHash[23] e *item covisitation*[24]) que pueden solucionar estos problemas para generar recomendaciones.

[22] PLSI (*Probabilistic Latent Semantic Indexing*): Es un algoritmo basado en técnicas de *clustering* para mejorar el desempeño de *Collaborative Filtering.* El modelo usa usuarios e ítems como variables aleatorias y la relación entre ellos es aprendida modelando una distribución conjunta de usuarios e ítems como una distribución mezclada.

[23] MinHash: Es un algoritmo basado en técnicas de *clustering*. Asigna a un par de usuarios a un mismo *cluster* con una probabilidad proporcional a la superposición entre el conjunto de ítems que este par de usuarios han calificado.

[24] *Item covisitation:* se define como dos ítems elegidos por el mismo usuario dentro de un intervalo de tiempo. Se almacena la información como una gráfica y esta se actualiza cada vez que el usuario tiene actividad.

2.1.5. Elección del modelo de recomendación

Para la elección del modelo de recomendación, se escogerá el modelo que mejor encaje con el concepto de brindar una recomendación que presente a los usuarios publicidad de locales comerciales de diferentes categorías, tomando en cuenta su localización y preferencias. Con respecto a los sistemas de recomendaciones estudiados: Facebook, Amazon y Google *News*, emplean *Collaborative Filtering*, o una variación de este modelo para adaptarlo a sus necesidades, ya sea por el alto número de usuarios que manejan o por el tipo de ítems que deben recomendar. En ellos se refleja la utilidad que CF tiene a la hora de recomendar ítems basados en las preferencias de los usuarios y las similitudes entre ellos. Netflix, en cambio emplea *machine learning* para realizar su recomendación, por dos factores: realiza una recomendación con base en las actividades que el usuario realiza sobre la plataforma de Netflix, y el hecho de que tiene más de 23 millones de subscriptores y más de 2.000 categorías de gustos.

De acuerdo a las características de *Knowledge-Based Recommendation,* se aplica en escenarios complejos en donde los ítems no son seleccionados muy frecuentemente y por ende usa el conocimiento que se tenga sobre las características de los ítems para satisfacer las necesidades y preferencias de los usuarios. Es por esto, que este modelo no es apto para el prototipo, debido a que no se almacena ninguna información de los ítems a recomendar, ni tampoco se utilizan ítems que son seleccionados con muy poca frecuencia.

Content-Based Recommendation cumple su trabajo solo si se cuenta con dos tipos de información: el contenido de la información disponible sobre los ítems y las calificaciones o comportamiento de compra de los usuarios. Su tarea consiste en determinar aquellos ítems que encajan mejor con las preferencias del usuario. El modelo tampoco encaja con el prototipo debido a que no se manejará información alguna de los ítems.

Finalmente, *Collaborative Filtering* hace uso de la colaboración de las calificaciones de múltiples usuarios para identificar aquellos cuyos gustos sean similares a los gustos del usuario para el cual se va a generar la recomendación. Además, es muy versátil y puede ser aplicada a cualquier dominio y ayuda a recomendar al usuario ítems que son inesperados pero que pueden terminar gustándole. El prototipo encaja bien con el uso de este modelo y es por eso que se escoge a *Collaborative Filtering* como el modelo de recomendación a ser implementado en el prototipo.

2.1.6. Elección del Algoritmo de Recomendación

Con respecto al Algoritmo de Recomendación, se escogerá aquel que trabaje de mejor manera con el modelo *Collaborative Filtering*. KDD se encarga del descubrimiento de

información no trivial, implícita, previamente desconocida y útil de datos. Sin embargo, CF no desea extraer información sino utilizar la información solo de usuarios existentes en la base de datos para brindar información. Es por eso que KDD no se escoge como el algoritmo para el prototipo.

El algoritmo *web usage mining* se encarga del descubrimiento automático y análisis de patrones de secuencia de clics y datos asociados, recolectados o generados como resultado de interacciones de usuarios con recursos web en uno o más sitios web. La mayoría de esta información es usualmente generada por servidores web, y recolectada en forma de *logs*. Como CF no tiene por objeto tratar de recoger información, ni interactuar con sitios web, *web usage mining* no es el algoritmo más adecuado para este caso.

K-Nearest Neighbor se basa en una lista ordenada de los usuarios más similares al usuario al que se desea entregar una recomendación [24]. Esto conlleva a realizar una comparación de un usuario versus el resto de usuarios. CF puede usar este algoritmo dado que el modelo conoce las preferencias de los usuarios y el algoritmo puede usar esta información para conocer la similitud entre ellos y así brindar una recomendación. Por lo tanto, se escoge el algoritmo k-NN para el prototipo. Para encontrar la similitud entre usuarios se utilizará el coeficiente de Correlación de Pearson.

Adicionalmente, según [26] se puede tomar el conjunto de entrenamiento (CE) y convertirlo a un dominio más pequeño llamado modelo k-NN, en donde las muestras del CE son agrupadas con base en la similitud que existe entre ellas. La salida de este modelo está formada por la clase del grupo, la similitud entre la muestra más distante y la muestra central del grupo, y el número de muestras en ese grupo. En este caso no existe necesidad de escoger el mejor valor de k, dado que el número de muestras en cada grupo puede ser visto como el valor óptimo de k y la razón de esto es que cada grupo tiene sus propias características. Por esta razón, al usuario se le mostrará una lista de categorías de locales comerciales, que vendrían a representar los grupos a los que pertenecen las muestras del conjunto de entrenamiento.

2.1.7. Arquitectura del prototipo

Los componentes que conformarán el prototipo del sistema son: un cliente Android, un servicio WCF, un algoritmo de recomendaciones y una base de datos SQL. La comunicación entre los componentes se la realizará mediante el protocolo HTTP. La arquitectura que tendrá el prototipo es del tipo cliente-servidor, como se observa en la Figura 2.2.

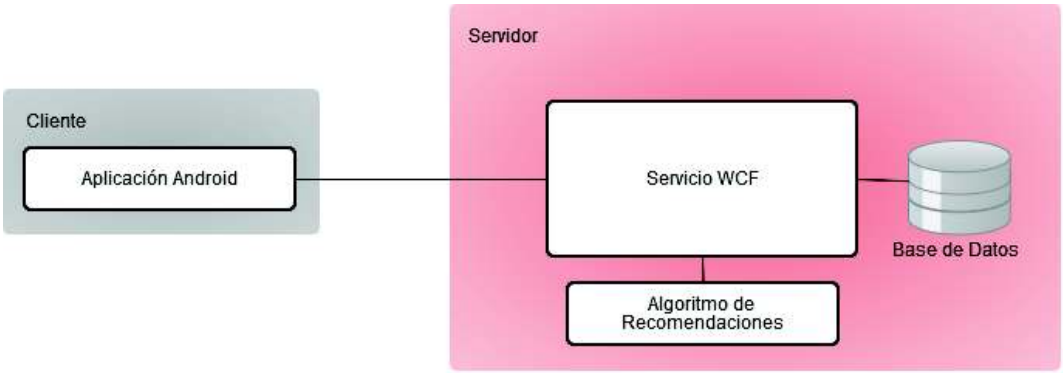


Figura 2.2. Arquitectura del prototipo

2.1.8. Requerimientos de usuario

Considerando que el modelo de recomendaciones que se implementará en el prototipo es *Colaborative Filtering* con el algoritmo k-NN, al usuario se le mostrará una lista de categorías de locales comerciales, de tal forma que al seleccionar una o varias categorías que desea buscar, se le muestre a continuación una lista de comerciales para que el usuario califique. Para determinar qué categorías se deben mostrar al usuario, y cómo hacerlo, se realizó una encuesta a los usuarios. A continuación se muestran los resultados obtenidos de esta encuesta. La encuesta se encuentra en el Anexo I.

La primera pregunta permite determinar las categorías de publicidad que se van a mostrar al usuario. En la Figura 2.3 se observa que las cuatro categorías más escogidas son: Comida, Ropa, Zapatos y Entretenimiento.

Figura 2.3. Respuestas a la primera pregunta de la encuesta

La segunda pregunta permite conocer si los usuarios desean calificar los locales comerciales una vez han adquirido el producto o servicio. Los resultados muestran que un 89.6% de los encuestados sí quisiera poder calificar locales comerciales considerando sus experiencias previas.

La tercera pregunta permite conocer si el usuario está de acuerdo en almacenar y mostrar el historial de las calificaciones que hizo a locales comerciales. Los resultados muestran

que a un 86% de los encuestados les gustaría visualizar las calificaciones que han realizado a locales comerciales.

La cuarta pregunta permite conocer la cantidad de publicidad recomendada que los usuarios desean observar, de tal forma que no sea abrumante ni escasa. La Figura 2.4 indica que la mayoría de usuarios encuestados (el 50% de usuarios) piensa que 10 es una cantidad de recomendaciones apropiadas.

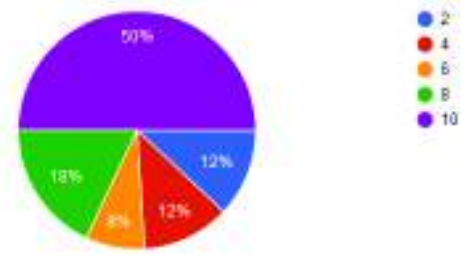


Figura 2.4. Respuestas a la cuarta pregunta de la encuesta

Las recomendaciones que se le mostrarán al usuario serán basadas en la ubicación, para lo cual se requiere un radio de búsqueda para determinar los locales comerciales que se encuentren cerca del usuario. Para determinar la máxima distancia del radio se realiza la quinta pregunta. La Figura 2.5 refleja que el mayor porcentaje de usuarios encuestados (54%) piensa que 100 metros es la distancia máxima a la que se debería ajustar el radio de búsqueda de locales comerciales.

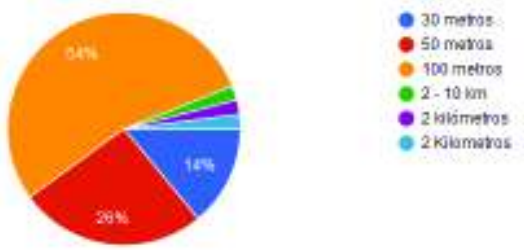


Figura 2.5. Respuestas a la quinta pregunta de la encuesta

Al usuario se le presentará una lista de locales comerciales. Los locales comerciales que han sido considerados para este proyecto se encuentran en la zona norte del Distrito Metropolitano de Quito, y se recolectó información de 91 locales. Para que un usuario pueda identificar a un local comercial se ha recolectado la siguiente información:

- Nombre del local.
- Categoría pertenece el local.
- Imagen que represente al local.
- Ubicación (latitud y longitud).
- Publicidad que ofrece el local comercial de sus productos.

De los resultados de la encuesta se han obtenido los siguientes requerimientos que serán plasmados en el prototipo:

- Las categorías que se presentarán al usuario son Comida, Ropa, Zapatos y Entretenimiento.
- Se permitirá calificar locales comerciales con base en sus experiencias previas.
- Se le mostrará al usuario las calificaciones que ha realizado previamente a locales comerciales.
- La cantidad de recomendaciones máxima que se mostrará al usuario es 10.
- La máxima distancia que cubre el radio será de 100 metros.

Considerando que es necesario identificar cada usuario que haga uso del prototipo, es necesario que un usuario se registre para ingresar al sistema y obtenga recomendaciones.

2.1.9. Diagrama de flujo del sistema

El diagrama de flujo que se muestra en la Figura 2.6 describe los pasos que se ejecutarán para la obtención de recomendaciones. Primero se pregunta si el usuario está registrado en el sistema; si no lo está, se procede a realizar el registro del usuario, y si el usuario está registrado, se procede al ingreso al sistema.

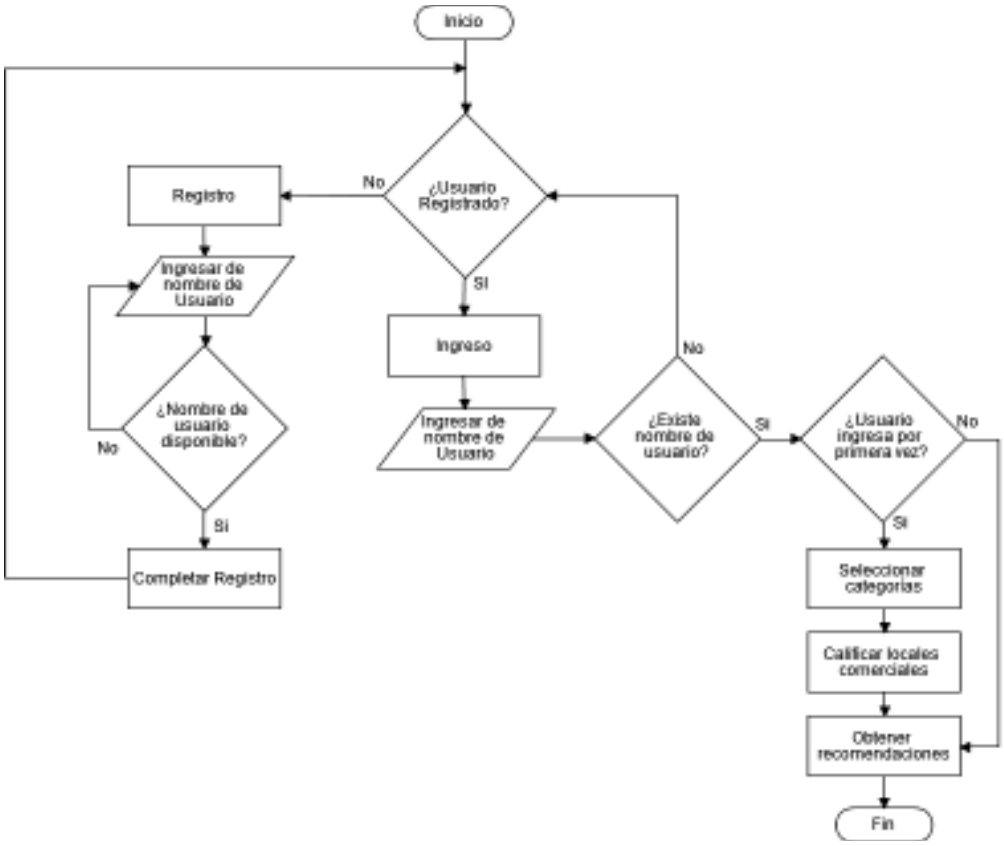


Figura 2.6. Diagrama de flujo del sistema

Para que el usuario pueda registrarse, primero ingresará su nombre que lo identificará dentro del sistema, luego se tendrá que comprobar que ese nombre esté disponible y no haya sido usado por algún otro usuario, si no está disponible, se tendrá que ingresar otro nombre de usuario, y si está disponible, se completará el registro.

En el ingreso, el usuario ingresará su nombre y se comprobará que este exista en el sistema, si no existe en el sistema tendrá que registrarse. Si el usuario existe en el sistema, se comprueba si es la primera vez que ingresa al sistema. Si es la primera vez, se procederá seleccionar las categorías de locales comerciales y a continuación a calificar los locales comerciales para que finalmente el usuario obtenga recomendaciones de publicidad. Si no es la primera vez que ingresa, el usuario obtiene directamente las recomendaciones de publicidad generadas a partir de las calificaciones que ha realizado anteriormente.

2.1.10. Diagrama de Casos de Uso

Después del análisis realizado en los requerimientos de usuario, solamente se requiere un tipo de actor, el cual se va a llamar Usuario. El actor Usuario utilizará el prototipo para obtener recomendaciones de publicidad. Los casos de uso principales son Registro e Ingreso, como muestra la Figura 2.7.

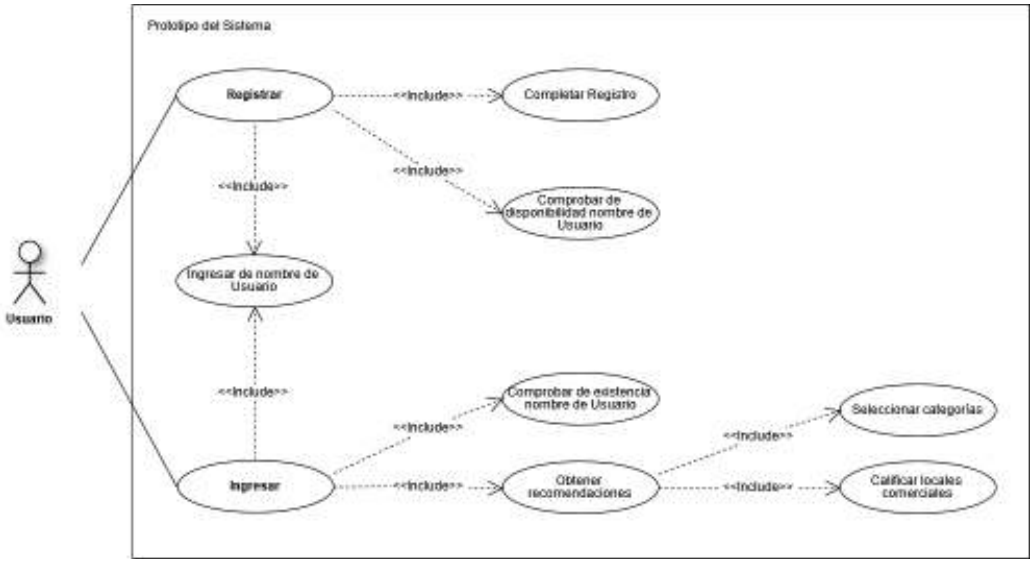


Figura 2.7. Diagrama de Casos de Uso

Registrar, permitirá que el usuario pueda registrarse en el sistema. Registrar incluye los casos de uso: Ingresar nombre de Usuario, Comprobar disponibilidad de nombre de usuario y Completar Registro.

Ingresar nombre de Usuario, permitirá al usuario ingresar su nombre; Comprobar disponibilidad de nombre de usuario, permitirá verificar que no exista otro usuario con dicho

nombre en el sistema; y, Completar Registro, permitirá al usuario finalizar el proceso de registro.

Ingresar, permitirá que el usuario pueda ingresar al sistema. Ingresar incluye: Ingresar nombre de Usuario, Comprobar existencia de nombre de usuario y Obtener recomendaciones.

Comprobar existencia de nombre usuario, permitirá al usuario verificar que está registrado en el sistema y Obtener recomendaciones, permitirá al usuario conseguir recomendaciones de publicidad. Obtener recomendaciones incluye: Seleccionar categorías, en donde el usuario podrá seleccionar las categorías de publicidad, y Calificar locales comerciales, donde el usuario podrá calificar locales comerciales.

2.1.11. Diagrama de Entidad – Relación

La base de datos es el componente que se encargará de almacenar toda la información del prototipo, y con base en los requerimientos de usuarios establecidos, se realizó el diagrama de Entidad – Relación de la base de datos, el cual se muestra en la Figura 2.8.

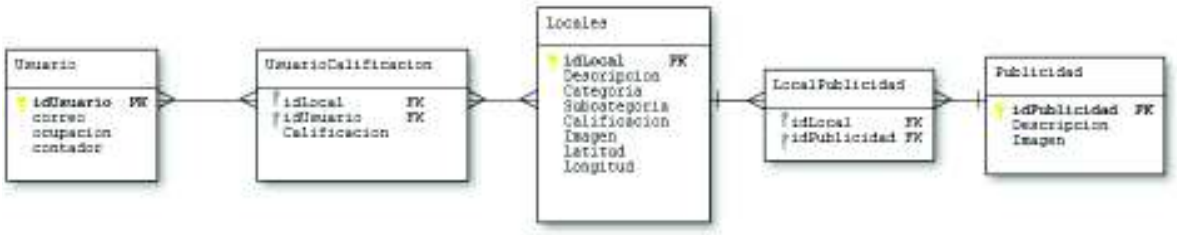


Figura 2.8. Diagrama de Entidad – Relación

La tabla `Usuario` se encargará de almacenar información de los usuarios, como su correo y ocupación. Un usuario tendrá un identificador único `idUsuario`, que corresponderá al nombre de usuario. El campo `contador` permitirá distinguir a los usuarios que ingresen por primera vez al sistema.

La tabla `Locales` contendrá información respecto a los locales comerciales. Cada local tendrá un identificador único `idLocal` y otros campos que describen a un local comercial, como una breve descripción del local, categoría y subcategoría a la que pertenece un local, calificación dada por un usuario, imagen del local, y la ubicación en coordenadas de latitud y longitud.

La tabla `Publicidad` contendrá información respecto a la publicidad de los locales comerciales. Cada publicidad tendrá un identificador único `idPublicidad`, una breve descripción y una imagen de la publicidad.

La tabla `UsuarioCalificacion` es una tabla de juntura que va a permitir relacionar la tabla `Usuario` y la tabla `Locales`, de tal forma que un usuario podrá calificar a varios locales comerciales, y un local comercial podrá tener varias calificaciones.

La tabla `LocalPublicidad` es una tabla de juntura que permitirá identificar las publicidades correspondientes a los locales comerciales.

2.1.12. Diagrama de clases del servicio WCF

El servicio WCF ofrecerá métodos que reflejen los procedimientos establecidos en el diagrama de casos de uso. El servicio WCF consta principalmente de una interfaz y una clase que implementa dicha interfaz como se observa en la Figura 2.9. Las clases `Usuario`, `Locales`, `Publicidad` y `UsuarioCalificacion` son utilizadas para asociar la información almacenada en las tablas de la base de datos. La clase `Categorias` se utiliza para mantener las categorías seleccionadas. La interfaz `IOperaciones`, define todos los métodos los cuales serán consumidos por un usuario a través de peticiones HTTP. La clase `Operaciones` implementa las operaciones definidas en la interfaz y otros métodos requeridos para realizar consultas en la base de datos o utilizar el algoritmo de recomendaciones. A continuación se describen estos métodos:

- `EnviarCalificaciones`: retorna una lista de recomendaciones de publicidad, generada a partir de las calificaciones realizadas por un usuario, que deben ser proporcionadas como argumento.
- `Intereses`: retorna una lista de los locales comerciales que corresponden a los intereses seleccionados por un usuario, que son proporcionados como argumento.
- `Login`: retorna las propiedades de un usuario registrado en la base de datos, usando su identificador como argumento.
- `ObtenerHistorial`: retorna una lista de locales comerciales que un usuario ha calificado.
- `ObtenerRecomendaciones`: retorna una lista de recomendaciones de publicidad generada a partir de los intereses de un usuario. El nombre de usuario será proporcionado como argumento.
- `ObtenerUsuarios`: retorna una lista de los usuarios almacenados en la base de datos.
- `Registrar`: recibe como argumentos la información de un usuario y lo almacena en la base de datos.
- `VerificarLogin`: recibe como argumento el nombre de un usuario y retorna como resultado un entero, que indica si el usuario existe en la base de datos.

- `Verificar`: recibe como argumento el nombre de un usuario y retorna como resultado un entero, que indica si el nombre de usuario está disponible para ser registrado.
- `CalcularDistancia`: retorna la distancia que existe entre un usuario y un local comercial. Las coordenadas de latitud y longitud del usuario y del local serán proporcionadas como argumentos
- `EjecutarSentencia`: permite especificar cualquier sentencia SQL *non query* como argumento, y retorna como resultado el número de filas afectadas.
- `IngresarCalificaciones`: recibe como argumento las calificaciones realizadas por un usuario y las almacena en la base de datos.
- `ObtenerLocales`: retorna una lista de los locales comerciales almacenados en la base de datos.
- `ObtenerPublicidad`: retorna una lista de la publicidad almacenada en la base de datos.
- `ObtenerPublicidadCercana`: retorna una lista de todos los locales comerciales que están dentro de un radio de distancia especificado por el usuario, que se proporciona como argumento.

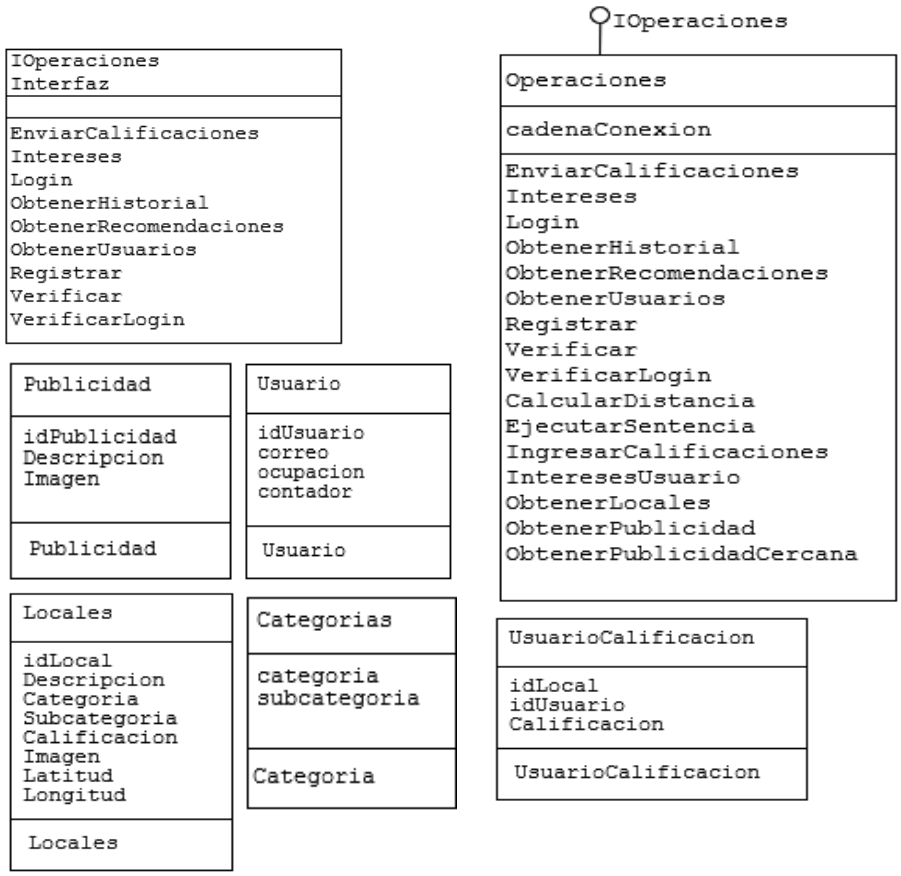


Figura 2.9. Diagrama de clases del servicio WCF

2.1.13. Diagrama de clases del algoritmo de recomendación

El algoritmo de recomendación será instanciado desde el Servicio WCF. La Figura 2.10 muestra el diagrama de clases del algoritmo de recomendación. El algoritmo de recomendación ofrecerá métodos para el cálculo de la recomendación. El servicio WCF instanciará al algoritmo de recomendación llamando a la clase `Formato`. Esta clase contiene el siguiente método:

- `ListaUso`: retorna una lista ordenada con las recomendaciones de locales comerciales con su respectiva valoración. Utiliza un diccionario que contiene información de Identificación del usuario (`IdUsuario`), identificación de local comercial (`IdPublicidad`) y calificación del usuario (`Calificacion`). También verifica que dentro del diccionario haya información del usuario al cuál se brindará la recomendación.

La clase `BaseDeRecomendacion` es utilizada para generar recomendaciones sin ordenar como resultado de la comparación de los usuarios con el usuario al cuál se entregará la recomendación. Esta clase dispone de los siguientes métodos:

- `CalculoRecomendacion`: retorna una lista no ordenada de recomendaciones de locales comerciales. Realiza una verificación del valor de similitud calculada entre el usuario al que se brinda la recomendación y el resto de usuarios. Si el valor de similitud, obtenido a partir del coeficiente de correlación de Pearson, es cero o menor que cero, no existe similitud entre usuarios, se ignora este usuario y sus locales comerciales de la recomendación y se procede a comparar con otro usuario. Si la similitud es mayor que cero, se usa las calificaciones de locales comerciales de este usuario para la recomendación.

- `ObtenerVectorElementosSimilares`: retorna un vector resultado de la comparación del usuario al que se brinda la recomendación y otro usuario. El vector contiene las identificaciones de locales comerciales, con la respectiva valoración de los dos usuarios coincidentes.

- `ObtenerVectorElementosNoSimilares`: retorna un vector resultado de comparar el usuario al que se brinda la recomendación y otro usuario. Este vector se usa si después de la comparación con cada uno de los usuarios no hay elementos similares; y al no haber elementos similares la recomendación como tal no es posible al no existir ningún tipo de relación. Es por ello que se usa este vector que tiene elementos aún no valorados por el usuario al que se brinda la recomendación.

La clase `Distancia` se utiliza para obtener el valor de similitud calculado a partir del coeficiente de correlación de Pearson utilizando las calificaciones de locales comerciales similares. Contiene el siguiente método:

- `CorrelacionPearson`: retorna el coeficiente de correlación de Pearson utilizando los valores obtenidos a partir del método `ObtenerVectorElementosSimilares`.

La clase `Ordenar` se utiliza para obtener una lista ordenada que contiene solo las identificaciones de locales comerciales (sin valoración) que va ser usada por el servicio WCF. La clase `ItemARecomendar` es utilizada para crear un objeto que almacenará la identificación de local comercial y la calificación del usuario a ese local.

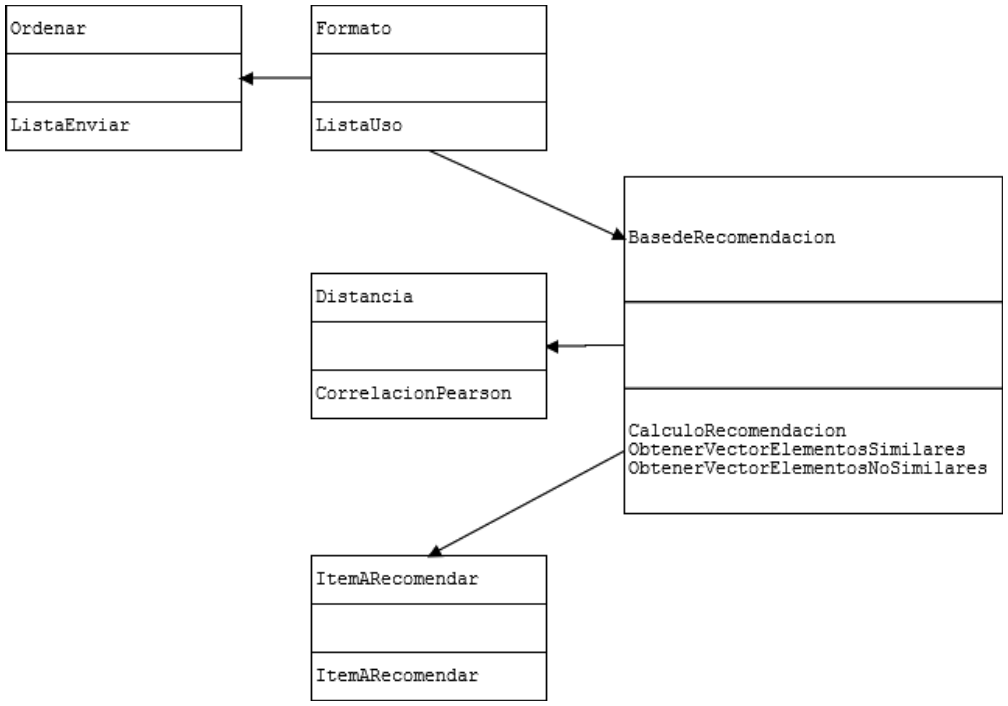


Figura 2.10. Diagrama de clases del algoritmo de recomendación

2.1.14. Diagrama de clases del cliente Android

En la Figura 2.11 se presenta una descripción del cliente Android representada por un diagrama de clases. En este diagrama se definen las clases para el cliente Android.

Cada actividad tiene que derivar de `AppCompatActivity`[25], por lo que deberán implementar el método `onCreate`, donde se ejecutará la configuración básica de la actividad, como la declaración de variables, y la declaración y configuración de la interfaz gráfica.

[25] `AppCompatActivity`: es una clase base para actividades que hacen uso de las funciones de la barra de acción de la biblioteca de soporte de Android Studio.

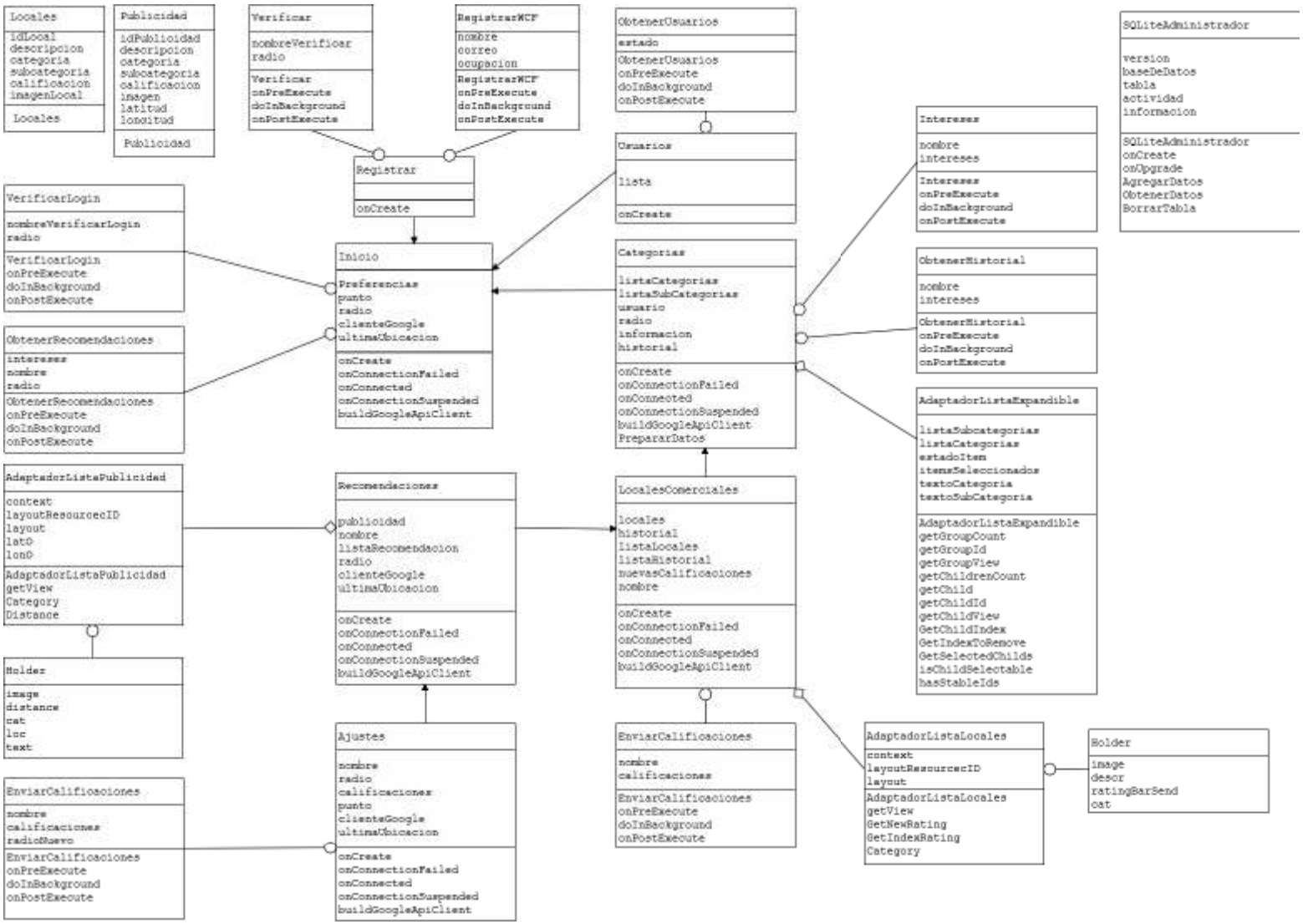


Figura 2.11. Diagrama de clases del cliente Android

A continuación se describen las actividades del cliente Android:

- `Inicio`: actividad principal del cliente Android. Aquí un usuario podrá ingresar al sistema y en caso de no estar registrado, dirigirse a la actividad de registro.
- `Registrar`: actividad en donde un usuario podrá ingresar sus datos para registrarse en el sistema.
- `Usuarios`: actividad en donde se podrán visualizar los usuarios registrados en el sistema.
- `Categorias`: actividad en donde un usuario podrá seleccionar sus intereses de una lista de categorías de publicidad.
- `LocalesComerciales`: actividad en donde se mostrará al usuario una lista de locales comerciales con base en su selección de intereses. Aquí el usuario podrá calificar locales comerciales según sus experiencias previas.
- `Recomendaciones`: actividad en donde se mostrará al usuario una lista de recomendaciones de publicidad con base en las calificaciones que realizó a locales comerciales.
- `Ajustes`: actividad en donde el usuario podrá ajustar el radio de búsqueda de locales comerciales.

Las actividades `Inicio`, `Categorias`, `Recomendaciones` y `Ajustes`, van a implementar las clases `OnConnectionFailedListener` y `ConnectionCallbacks` para hacer uso de la API de *Google Play Services*[26] y obtener la ubicación del dispositivo Android. Por lo tanto, estas clases deberán implementar los métodos `onConnectionFailed`, `onConnected` y `onConnectionSuspened`.

El método `onConnectionFailed` se emplea para recibir errores de conexión con la API de Google. `onConnected` se utiliza para saber cuándo se ha establecido una conexión y `onConnectionSuspened` para saber cuándo se ha suspendido una conexión. Para establecer una conexión con la API de Google se emplea el método `buildGoogleApiClient`.

En las actividades se usan clases internas para realizar peticiones HTTP al servicio WCF. Una clase interna es una clase anidada no estática. Las clases internas son derivadas de `AsyncTask` [27] para realizar peticiones HTTP en segundo plano, de tal forma que no

[26] *Google Play Services*: es un servicio propietario que trabaja en segundo plano, usado para el desarrollo de aplicaciones Android.
[27] `AsyncTask`: es una clase que contiene métodos para realizar operaciones en segundo plano y mostrar resultados en la interfaz gráfica, sin la necesidad de manipular *threads* y o *handlers*.

interfieran con los procesos de la interfaz gráfica de las actividades. Por lo tanto estas clases deberán implementar los métodos `onPreExecute`, `doInBackground` y `onPostExecute`.

El método `onPreExecute` es el primero en ser invocado cuando se ejecuta la tarea, y se emplea para configurar la interfaz gráfica. `doInBackground` se emplea para realizar la petición HTTP y recibir su respuesta. `onPostExecute` se ejecuta al finalizar `doInBackground` y se encarga de procesar la información recibida en la respuesta de la petición HTTP.

A continuación se describen las clases internas que realizan peticiones HTTP:

- `VerificarLogin`: es una clase interna de `Inicio` usada para verificar si un usuario existe en la base de datos.
- `ObtenerRecomendaciones`: es una clase interna de `Inicio` usada para obtener recomendaciones de publicidad de un usuario con base a sus intereses.
- `ObtenerUsuarios`: es una clase interna de `Inicio` usada para obtener los usuarios en la base de datos.
- `Verificar`: es una clase interna de `Registro` usada para verificar si un nombre de usuario está disponible para ser registrado en el sistema.
- `RegistrarWCF`: es una clase interna de `Registro` usada almacenar un usuario.
- `Intereses`: es una clase interna de `Categorias` usada para obtener una lista de locales comerciales con base en los intereses del usuario.
- `ObtenerHistorial`: es una clase interna de `Categorias` usada para obtener los locales comerciales que ha calificado el usuario.
- `EnviarCalificaciones`: es una clase interna de `LocalesComerciales` usada para obtener recomendaciones de publicidad con base en las calificaciones que realizó el usuario.
- `EnviarCalificaciones`: es una clase interna de `Ajustes` usada para obtener recomendaciones de publicidad con base en las calificaciones que realizó el usuario y el nuevo radio que escogió el usuario.

Las clases `Locales` y `Publicidad` son utilizadas para representar la información recibida en las respuestas a las peticiones HTTP.

Para que el usuario pueda elegir sus intereses de publicidad, se le presentará una lista expandible de categorías; cada categoría con sus respectivas subcategorías de publicidad.

Esta lista de categorías se mostrará en un `ExpandableListView`[28] para lo cual define el adaptador `AdaptadorListaExpandible`, que se deriva de `BaseExpandableList` por lo que se deberán implementar los siguientes métodos:

- `getChildView`: retorna una vista que representa una subcategoría, para ser mostrada en la interfaz gráfica.
- `getGroupView`: retorna una vista que representa a una categoría, para ser mostrada en la interfaz gráfica.
- `getChild`: retorna la información correspondiente a una subcategoría.
- `getGroup`: retorna la información correspondiente a una categoría.
- `getChildrenCount`: retorna el número de subcategorías dentro de una categoría.
- `getGroupCount`: retorna el número total de categorías.
- `getGroupId`: retorna un valor que identifica una categoría.
- `getChildId`: retorna un valor que identifica una subcategoría.
- `isChildSelectable`: este método permite definir si una subcategoría es elegible o no.

Las respuestas a peticiones HTTP que contenga listas de locales comerciales, historiales de calificaciones o listas de recomendaciones de publicidad, se mostrarán como listas en la interfaz gráfica a través del *widget* `ListView`[29]. Para lo cual, se definen adaptadores derivados de `ArrayAdapter`, y por lo tanto deberán implementar el método `getView`. Este método retorna una vista que representa un ítem de una lista para ser presentado en la interfaz gráfica.

A continuación se describen los adaptadores empleados:

- `AdaptadorListaLocales`: es un adaptador empleado para mostrar una lista de locales comerciales con una barra de calificaciones. Se utilizará este mismo adaptador para mostrar el historial de calificaciones de un usuario.
- `AdaptadorListaPublicidad`: es un adaptador empleado para mostrar una lista de recomendaciones de publicidad.

Considerando que la cantidad de datos que se puede pasar de una actividad a otra es aproximadamente 500 KB [42], [43], se define la clase `SQLiteAdminstrador`, que se utiliza para almacenar información de una actividad en la base de datos SQLite del dispositivo Android, para luego recuperarla en otra actividad.

[28] `ExpandableListView`: es un *widget* utilizado para mostrar grupos de datos en una lista por categorías.
[29] `ListView`: es un *widget* empleado para mostrar datos en una lista desplazable.

2.1.15. Sketches de la interfaz gráfica del cliente Android

A continuación se presentan los diseños de las interfaces gráficas de las actividades que conforman el cliente Android. En la Figura 2.12 se presenta el diseño de la interfaz gráfica de la actividad `Inicio`. Aquí gráficamente, el usuario podrá realizar el ingreso al sistema, dirigirse a la actividad que muestra todos los usuarios registrados, o dirigirse a la actividad de registro.

Figura 2.12. *Sketch* de la actividad `Inicio`

En la Tabla 2.1 se muestra la descripción de los elementos del *sketch* de la actividad `Inicio`.

Tabla 2.1. Descripción de la actividad `Inicio`

Identificador	Descripción	Control
1	Permite ingresar el nombre de usuario	`EditText`
2	Permite dirigirse realizar el ingreso al sistema	`Button`
3	Permite al usuario elegir la opción si desea que su nombre sea recordado al iniciar la aplicación	`CheckBox`
4	Permite dirigirse a la actividad `Usuarios`	`TextView`
5	Permite mostrar texto	`TextView`
6	Permite mostrar texto	`TextView`
7	Permite dirigirse a la actividad `Registro`	`TextView`

En la Figura 2.13 se presenta el diseño de la interfaz gráfica de la actividad `Usuarios`. Aquí se podrá observar todos los usuarios registrados en el sistema.

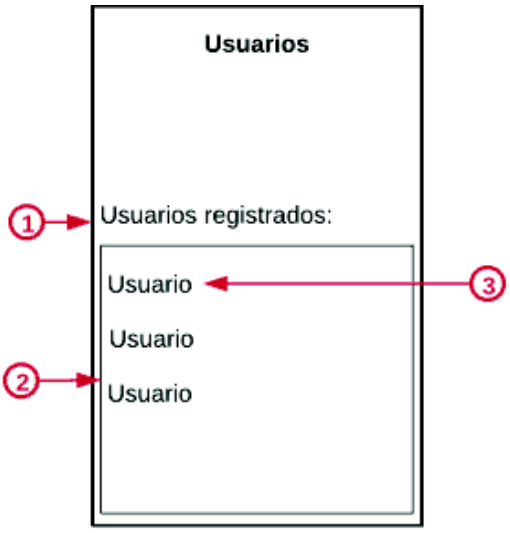


Figura 2.13. *Sketch* de la actividad `Usuarios`

En la Tabla 2.2 se muestra la descripción de los elementos del *sketch* de la actividad `Usuarios`.

Tabla 2.2. Descripción de la actividad `Usuarios`

Identificador	Descripción	Control
1	Permite mostrar texto	`TextView`
2	Permite mostrar una lista de usuarios	`ListView`
3	Permite mostrar en texto las propiedades de un usuario	`TextView`

En la Figura 2.14 se muestra el diseño de la interfaz gráfica de la actividad `Registro`. Aquí un usuario podrá ingresar su información para registrarse en el sistema.

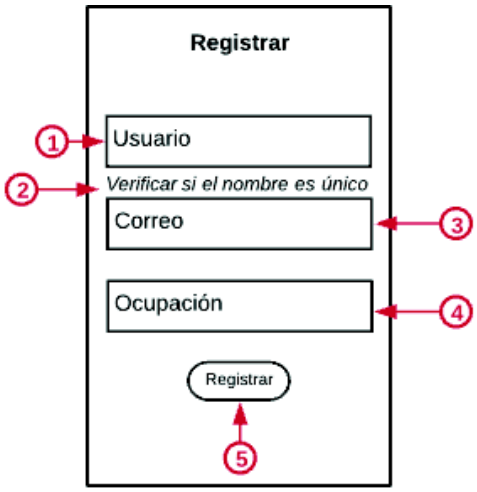


Figura 2.14. *Sketch* de la actividad `Registro`

En la Tabla 2.3 se muestra la descripción de los elementos del *sketch* de la actividad `Registro`.

Tabla 2.3. Descripción de la actividad `Registro`

Identificador	Descripción	Control
1	Permite ingresar el nombre de usuario	`EditText`
2	Permite verificar si el nombre ingresado está disponible para ser registrado	`TextView`
3	Permite ingresar el correo del usuario	`EditText`
4	Permite ingresar la ocupación del usuario	`EditText`
5	Permite finalizar el registro del usuario en el sistema	`Button`

En la Figura 2.15 se muestra el diseño de la interfaz gráfica de la actividad `Categorias`. Aquí un usuario podrá elegir sus intereses de publicidad de una lista de categorías.

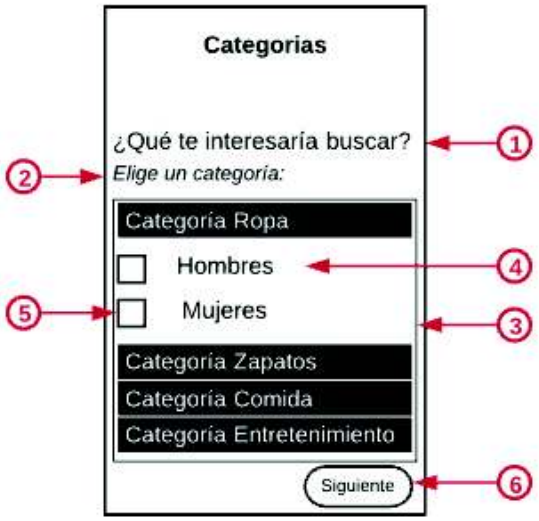


Figura 2.15. *Sketch* de la actividad `Categorias`

En la Tabla 2.4 se presenta la descripción de los elementos del *sketch* de la actividad `Categorias`.

Tabla 2.4. Descripción de la actividad `Categorias`

Identificador	Descripción	Control
1	Permite mostrar texto	`TextView`
2	Permite mostrar texto	`TextView`
3	Permite mostrar una lista expandible de categorías de publicidad	`ExpandableListView`
4	Permite mostrar una subcategoría	`TextView`
5	Permite al usuario elegir una subcategoría	`CheckBox`
6	Permite dirigirse a la actividad `LocalesComerciales`	`FloatingButton`

En la Figura 2.16 se muestra el diseño de la interfaz gráfica de la actividad `LocalesComerciales`. Aquí un usuario podrá calificar locales comerciales basados en su selección de categorías de publicidad.

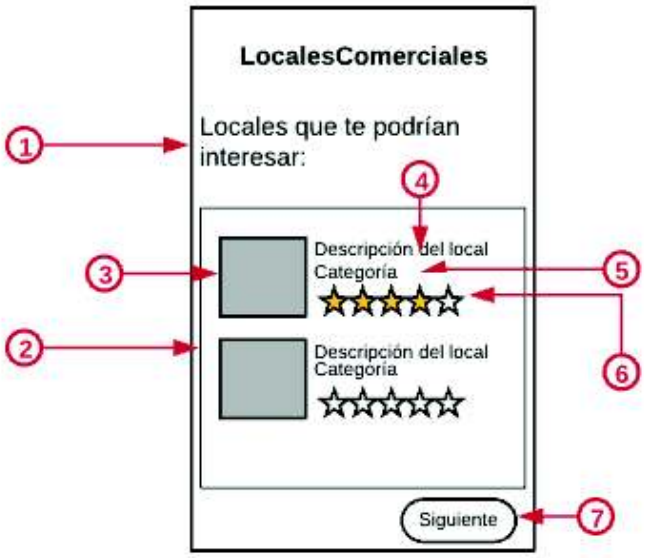


Figura 2.16. *Sketch* de la actividad `LocalesComerciales`

La Tabla 2.5 muestra la descripción de los elementos del *sketch* de la actividad `LocalesComerciales`.

Tabla 2.5. Descripción de la actividad `LocalesComerciales`

Identificador	Descripción	Control
1	Permite mostrar texto	`TextView`
2	Permite mostrar una lista de locales comerciales	`ListView`
3	Permite mostrar la imagen de un local comercial	`ImageView`
4	Permite mostrar la descripción de un local comercial	`TextView`
5	Permite mostrar la categoría a la que pertenece un local comercial	`TextView`
6	Permite al usuario calificar un local comercial	`RatingBar`
7	Permite dirigirse a la actividad `Recomendaciones`	`FloatingButton`

En la Figura 2.17 se muestra el diseño de la interfaz gráfica de la actividad `Recomendaciones`. Aquí se muestran las recomendaciones de publicidad a un usuario.

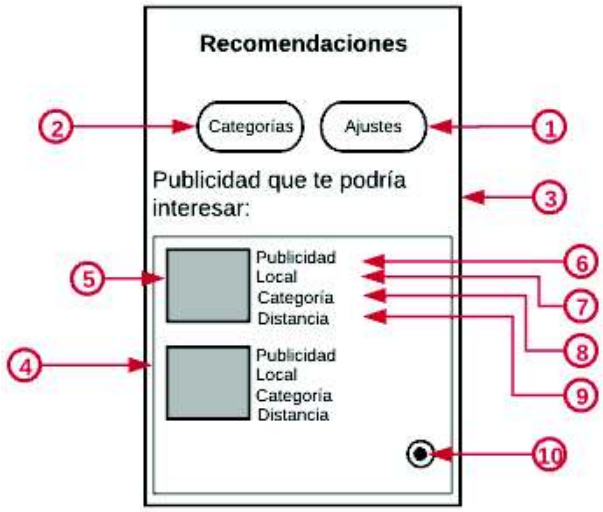


Figura 2.17. *Sketch* de la actividad `Recomendaciones`

En la Tabla 2.6 se muestra la descripción de los elementos del *sketch* de la actividad `Recomendaciones`.

Tabla 2.6. Descripción de la actividad `Recomendaciones`

Identificador	Descripción	Control
1	Permite dirigirse a la actividad `Ajustes`	`Button`
2	Permite dirigirse a la actividad `Categorias`	`Button`
3	Permite mostrar texto	`TextView`
4	Permite mostrar una lista de recomendaciones de publicidad	`ListView`
5	Permite mostrar una imagen de publicidad	`ImageView`
6	Permite mostrar una descripción de la publicidad	`TextView`
7	Permite mostrar a qué local pertenece la publicidad	`TextView`
8	Permite mostrar a qué categoría pertenece la publicidad	`TextView`
9	Permite mostrar a qué distancia del usuario se encuentra el local	`TextView`
10	Permite actualizar la ubicación del usuario	`FloatingButton`

En la Figura 2.18 se muestra el diseño de la interfaz gráfica de la actividad `Ajustes`, en la que el usuario podrá ajustar el radio de búsqueda de locales comerciales.

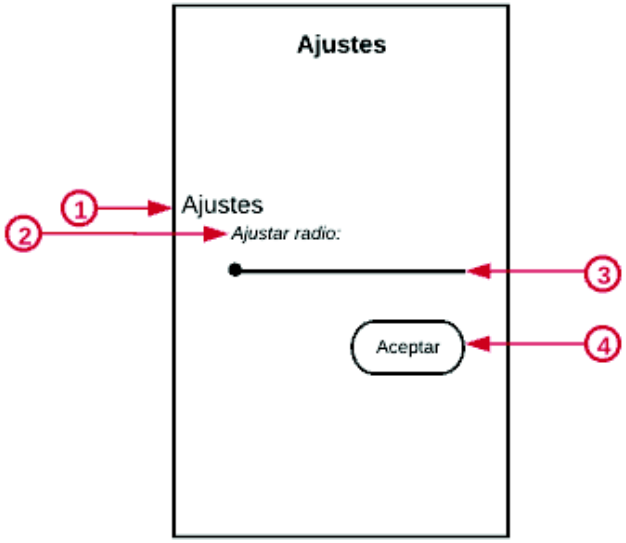


Figura 2.18. *Sketch* de la actividad `Ajustes`

En la Tabla 2.7 se muestra la descripción de los elementos del *sketch* de la actividad `Ajustes`.

Tabla 2.7. Descripción de la actividad `Ajustes`

Identificador	Descripción	Control
1	Permite mostrar texto	`TextView`
2	Permite mostrar texto	`TextView`
3	Permite al usuario ajustar el radio de búsqueda de locales	`SeekBar`
4	Permite regresar a la actividad `Recomendaciones`	`Button`

2.1.16. Diagramas de Interacción

Los diagramas de interacción describen cómo los componentes del prototipo, se van a comunicar entre sí para realizar las tareas establecidas en el diagrama de casos. A continuación se presentan los diagramas de interacción para el registro, ingreso de usuarios y obtención de recomendaciones.

a. Registro

El prototipo permitirá a usuarios registrarse para posteriormente poder acceder al mismo. La Figura 2.19 muestra cómo será la interacción de un usuario nuevo con el prototipo para registrarse.

El procedimiento de registro de usuarios es el siguiente: se mostrará la actividad de registro, en la que se permitirá el ingreso del nombre con que será identificado en el prototipo. Luego, se deberá comprobar que el nombre ingresado se encuentre disponible, para lo cual el cliente Android enviará una petición al servicio WCF, que le permita comprobar la disponibilidad del nombre del usuario nuevo.

El servicio WCF contará con un método que permita buscar en la base de datos el nombre de usuario ingresado para determinar si ya fue usado por otro usuario ya registrado en la base de datos.

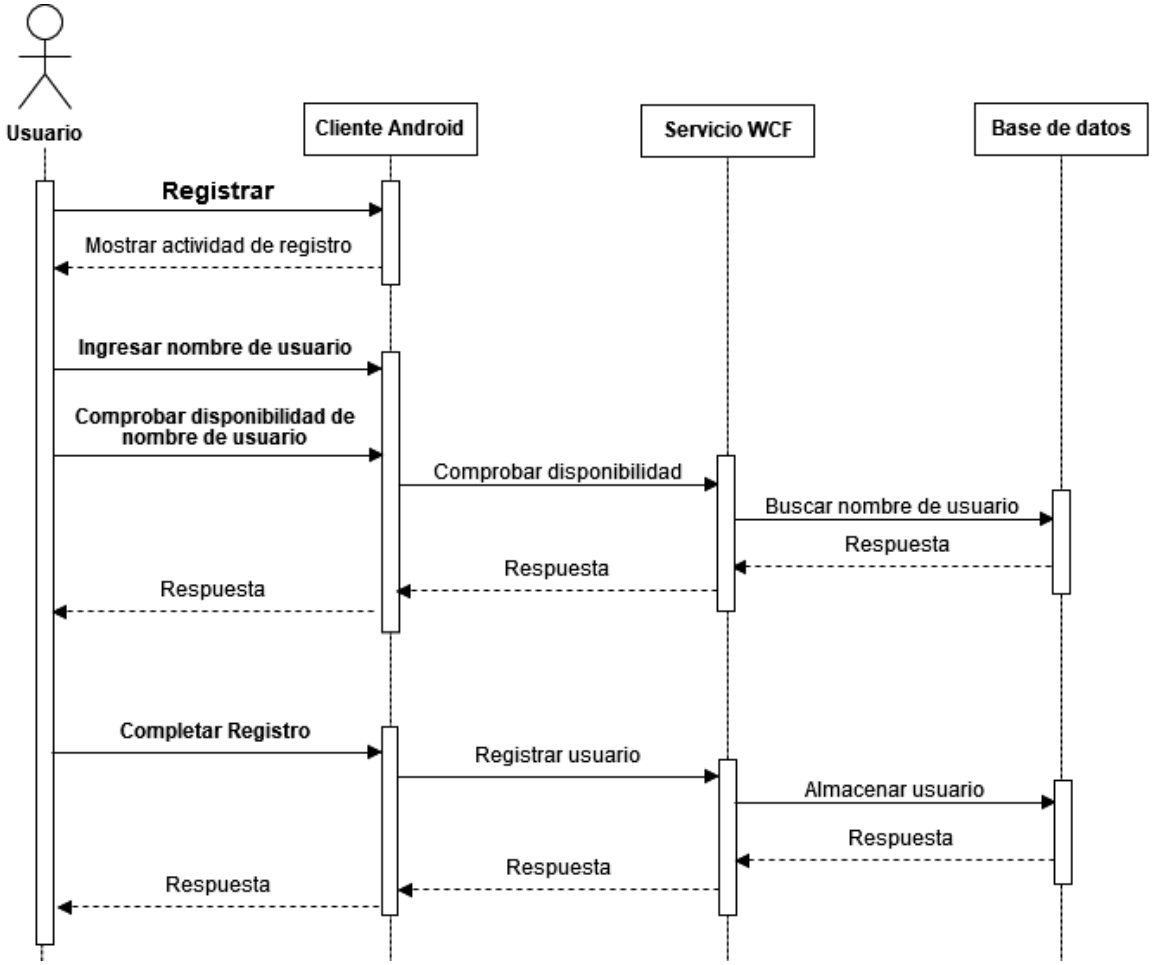


Figura 2.19. Diagrama de Interacción de Registro

b. Ingreso

Una vez que un usuario ha sido registrado en el prototipo, podrá acceder al mismo para obtener recomendaciones. Como muestra la Figura 2.20 un usuario primero ingresará su nombre en la actividad de inicio y procederá a comprobar si este existe en el prototipo, para lo cual el cliente Android enviará una petición al servicio WCF y se realizará una búsqueda en la base de datos de dicho nombre y se enviará la respuesta al usuario.

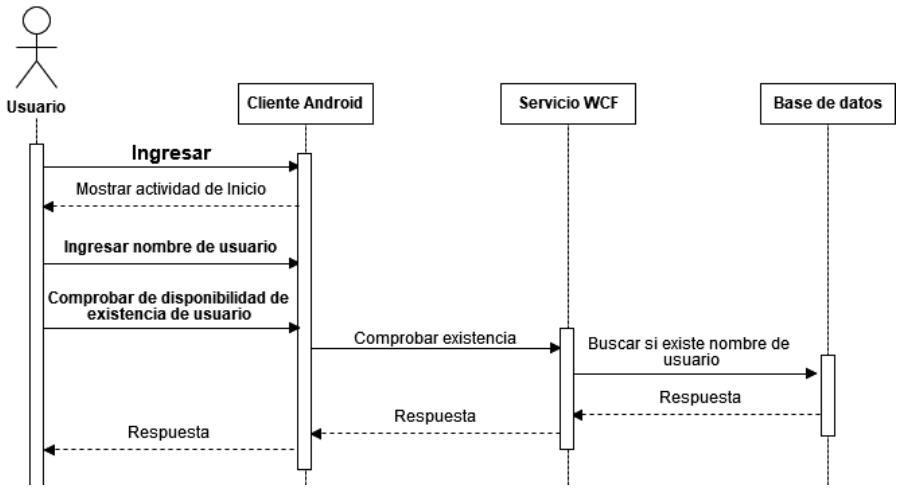


Figura 2.20. Diagrama de Interacción de Ingreso

c. Obtención de recomendaciones

Al ingresar un usuario al sistema, este procede a obtener recomendaciones de publicidad de locales comerciales de una categoría en particular. El proceso de obtención de recomendaciones se compone de la selección de categorías de publicidad y la calificación de locales comerciales.

En la Figura 2.21 se presenta la selección de categorías. Se inicia presentando al usuario una actividad con categorías de publicidad, de la cual el usuario seleccionará las categorías que le interesan y una vez completada esta selección, se enviará al servicio WCF en donde se realizará una búsqueda en la base de datos de todos los locales comerciales pertenecientes a las categorías seleccionadas por el usuario.

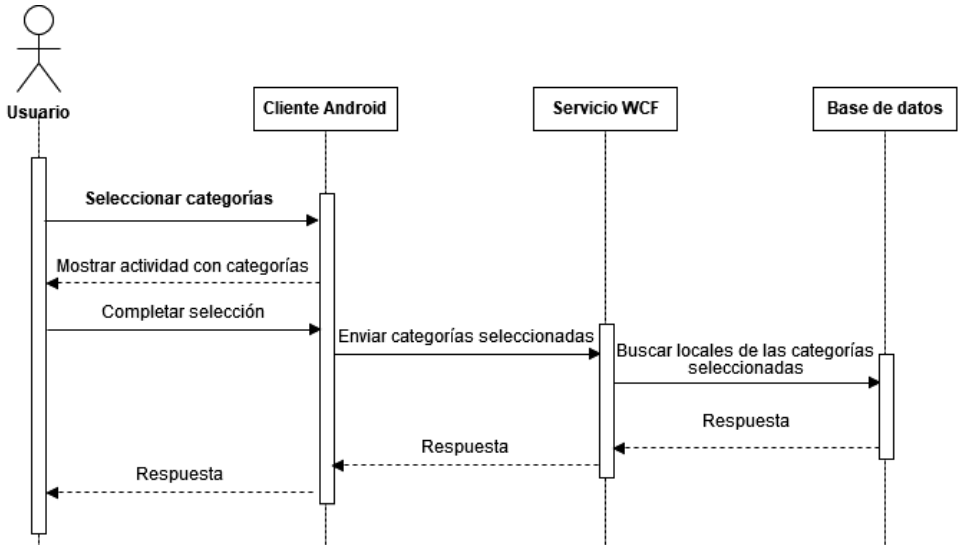


Figura 2.21. Diagrama de Interacción de selección de categorías

En la Figura 2.22 se presenta la calificación de locales comerciales. La respuesta de la selección de categorías será presentada al usuario en una actividad con locales comerciales, de tal forma que los pueda calificar. Una vez completada la calificación, será enviada al servicio WCF en donde se almacenarán estas calificaciones.

A continuación se obtienen de la base de datos, todos los locales comerciales relacionados a las categorías seleccionadas por el usuario y en conjunto con sus calificaciones, se procede a generar recomendaciones de locales comerciales. A partir de estos locales comerciales se obtiene de la base de datos, la publicidad correspondiente a cada local comercial y finalmente se envían las recomendaciones al usuario.

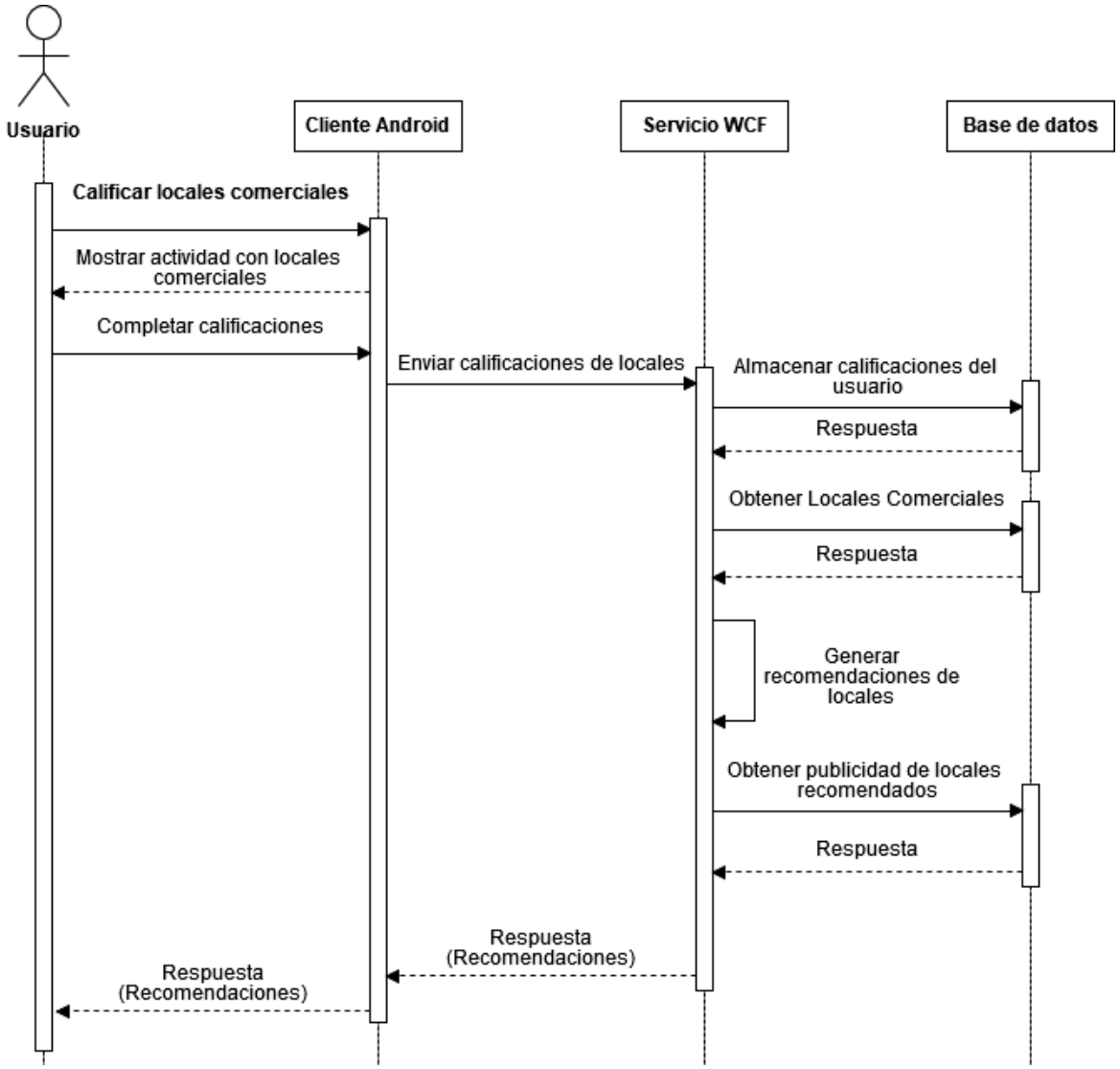


Figura 2.22. Diagrama de Interacción de calificación de locales comerciales

2.2. Implementación del prototipo

Para la implementación del prototipo se codificarán las tablas de la base de datos, el algoritmo de recomendaciones, el servicio WCF, la lógica del cliente Android y su interfaz

gráfica de usuario. Cada uno de los códigos se encuentran debidamente comentados en los Anexos III, IV y V. También se realizará la configuración del servidor y la configuración de una red inalámbrica para pruebas. A continuación se muestra un resumen de la implementación de cada componente.

2.2.1. Base de datos

Para el motor de la base de datos se utilizó SQL Server 2012 Express. A continuación se presenta un ejemplo de uno de los *scripts* usados para la generación de tablas, el resto de tablas fueron creadas con sentencias similares. Todos los *scripts* se encuentran en el Anexo II.

En el Código 2.1 se muestra un *script* para la creación de la tabla `Locales`, de la línea 1 a la 11 se define la sentencia `CREATE TABLE`. En la línea 2 se define la columna `IdLocal` que almacenará datos de tipo `INT`, que serán generados automáticamente con cada entrada nueva con un incremento de uno, no puede ser nula. Las líneas 3 y 4 definen columnas del tipo `VARCHAR` para la descripción y la categoría del local. La longitud máxima de estos campos es de 50 caracteres. En las líneas 5 y 6 se definen columnas del tipo `INT` para la subcategoría y calificación del local.

En la línea 7 se define una columna del tipo `VARCHAR` para almacenar una imagen transformada a una cadena de caracteres. La longitud de este campo es la máxima permitida para `VARCHAR`, cuyo valor es 65.535 bytes. Las líneas 8 y 9 definen columnas del tipo `FLOAT` para las coordenadas de latitud y longitud del local. En la línea 10 se especifica que la columna `IdLocal` es la llave primaria de esta tabla.

```
1   CREATE TABLE [dbo].[Locales] (
2       [IdLocal]      INT           IDENTITY (1, 1) NOT NULL,
3       [Descripcion]  VARCHAR (50)  NULL,
4       [Categoria]    VARCHAR (50)  NULL,
5       [Subcategoria] INT           NULL,
6       [Calificacion] INT           NULL,
7       [Imagen]       VARCHAR (MAX) NULL,
8       [Latitud]      FLOAT (53)    NULL,
9       [Longitud]     FLOAT (53)    NULL,
10      PRIMARY KEY CLUSTERED ([IdLocal] ASC)
11  );
```

Código 2.1. *Script* para la creación de la tabla `Locales`

2.2.2. Algoritmo de recomendaciones

Para la codificación del algoritmo de recomendaciones se empleó el lenguaje de programación C#. A continuación se muestran las secciones más importantes del código

del algoritmo de recomendaciones. El código completo se encuentra en el Anexo III. El Código 2.2 muestra parte de la clase `Formato`. Las líneas 10 a la 28 muestran el cambio de formato a diccionario anidado para que pueda ser interpretado por la lógica del algoritmo.

Las líneas 12 y 13 muestran el método `ListaUso`. Las líneas 15 y 16 crean el diccionario anidado `dic1` que es del tipo `Dictionary<string, Dictionary<string, double>>`. Dicha estructura contendrá la identificación del usuario en un objeto tipo `Key` del diccionario y como objeto tipo `Value` tendrá otro diccionario. Este segundo diccionario contendrá como objeto tipo `Key` la identificación del local, y como objeto tipo `Value` la calificación dada por el usuario a dicho local.

Las líneas 17 a la 19 crean vectores que van a contener temporalmente la información de identificación del usuario (`IdUsuario`), identificación de publicidad (`IdPublicidad`) y calificación (`Calificacion`) respectivamente. Las líneas 20 a 28 realizan el cambio de formato del tipo `List<UsuarioCalificacion>` a `Dictionary<string, Dictionary<string, double>>`.

```
10  public class Formato
11      {
12      public static List<int> ListaUso(List<UsuarioCalificacion>
13      ListaInicio, string ItemUser)
14          {
15          Dictionary<string, Dictionary<string, double>> dic1 = new
16          Dictionary<string, Dictionary<string, double>>();
17          string[] info = new string[1];
18          string[] info2 = new string[1];
19          double[] info3 = new double[1];
20          for (int i = 0; i < ListaInicio.Count; i++)
21          {
22           info[0] = ListaInicio[i].IdUsuario;
23           info2[0] = Convert.ToString(ListaInicio[i].IdPublicidad);
24           info3[0] = Convert.ToDouble(ListaInicio[i].Calificacion);
25           if (!dic1.ContainsKey(info[0]))
26            dic1.Add(info[0], new Dictionary<string, double>());
27            dic1[info[0]][info2[0]] = Convert.ToDouble(info3[0]);
28          }
```

Código 2.2. Clase `Formato`

Después del cambio de formato de los datos, se realiza una verificación, de tal forma que cuando exista un usuario que no haya calificado ninguna categoría aún, se retorne una lista con identificaciones de locales comerciales de la categoría que solicitó una recomendación. De esta forma, el usuario recibe una recomendación inesperada.

El Código 2.3 muestra el método `CalculoRecomendacion`. En las líneas 22 y 23 se realiza la comparación del usuario al que se brinda la recomendación con otro usuario

dentro del grupo del diccionario anidado `datos`. La línea 25 verifica que la comparación del usuario al que se brinda la recomendación no se realice con los datos de este mismo usuario. En las líneas 26 y 27 se llama al método `ObtenerVectorElementosSimilares`, para obtener ítems similares entre los usuarios que se comparan. La línea 28 comprueba si el vector resultante del método `ObtenerVectorElementosSimilares` tiene ítems, caso contrario, entre usuarios no hay similitud alguna y se procede a la siguiente comparación.

En las líneas 29 y 30 se llama al método `MedidaDistancia` para calcular la similitud existente entre usuarios a través del coeficiente de correlación de Pearson. La línea 31 permite saber si hay similitud entre usuarios por medio del valor calculado en la línea 30. Si el valor de similitud es cero o menor que cero, los usuarios no tienen relación y se continúa con la comparación con otro usuario. Si el valor es mayor que cero, hay similitud entre usuarios, y por tanto en la línea 32 se crea un diccionario que almacenará solo información del usuario al que se le brinda la recomendación.

Las líneas 33 a 47 asignarán una calificación a los ítems que no han sido calificados por el usuario al que se brinda la recomendación pero que sí han sido calificados por el otro usuario que tiene similitud.

```
19  public List<RecommenderItem> CalculoRecomendacion(Dictionary<
    string,Dictionary<string, double>> datos, string item,
    Func<double[],double[], double> DistanceMetric) {
      ...
22    foreach (KeyValuePair<string, Dictionary<string,
23    double>> pair in datos)
24    {
25    if (pair.Key == item) continue;
26    List<double[]> VectorElementosSimilares =
27    ObtenerVectorElementosSimilares(item,pair.Key,datos);
28    if (VectorElementosSimilares[0].Length == 0)continue;
29     double Similitud = MedidaDistancia(VectorElementosSimilares[0],
30     VectorElementosSimilares[1]);
31    if (Similitud <= 0) continue;
32     Dictionary<string, double> use = datos[item];
33    foreach (KeyValuePair<string, double> pairKeyValue in
34    pair.Value)
35    {...
47    }
```

Código 2.3. Método `CalculoRecomendacion` (parte 1 de 2)

Las líneas 49 y 50 crean una lista llamada `ListaRecomendacion` donde se agregarán los ítems que serán las recomendaciones. En las líneas 51 a 58 se realiza el cálculo del valor para cada ítem que se va a recomendar y se agrega este valor y el `IdPublicidad` en `ListaRecomendacion`. Esto sirve para poder organizar los ítems de mayor a menor

y recomendar en primer lugar aquellos que más similitud tienen con el usuario al que se brindará la recomendación.

```
49    List<RecommenderItem> ListaRecomendacion = new
50    List<RecommenderItem>();
51    for (int i = 0; i < diccionarioT1.Count; i++)
52    {
53     double calculoValor = diccionarioT1.ElementAt(i).Value /
54     diccionarioS1.ElementAt(i).Value;
55     diccionarioT1[diccionarioT1.ElementAt(i).Key]=calculoValor;
56     ListaRecomendacion.Add(new RecommenderItem(){Key =
57     diccionarioT1.ElementAt(i).Key,Value = calculoValor });
58    }
```

Código 2.3. Método `CalculoRecomendacion` (parte 2 de 2)

El Código 2.4 presenta parte del método `ObtenerVectorElementosSimilares` el cuál comparará los ítems valorados del usuario al que se le entregará la recomendación con cada uno del resto de usuarios dentro de la categoría. La línea 15 crea un vector que será retornado con ítems similares de los dos usuarios comparados.

Las líneas 16 a 19 crean dos diccionarios, el primero contendrá valores de ítems (con sus calificaciones) del usuario al que se brindará la recomendación y el segundo contendrá valores de ítems de otro usuario.

La línea 20 y 21 crean dos listas que sirven para agregar ítems para los dos usuarios que se comparan.

```
11  protected List<double[]> ObtenerVectorElementosSimilares(string
12  elemento1, string elemento2, Dictionary<string, Dictionary<string,
13  double>> dictionarydatos)
14          {
15          List<double[]> vectorEleCompartidos = new List<double[]>();
16          Dictionary<string, double> dictionaryelemento1 =
17          dictionarydatos[elemento1];
18          Dictionary<string, double> dictionaryelemento2 =
19          dictionarydatos[elemento2];
20            List<double> lstelemento1Vector = new List<double>();
21            List<double> lstelemento2Vector = new List<double>();
```

Código 2.4. Método `ObtenerVectorElementosSimilares` (parte 1 de 2)

Las líneas 22 a 30 realizan las comparaciones del usuario al que se le brindará la recomendación con otro usuario, comparando ítem por ítem y si coinciden, se agregan los ítems a los dos vectores instanciados en las líneas 16 a 19 (correspondientes a los dos usuarios comparados) con sus valores de calificación respectivos. La línea 31 agrega ítems del usuario al que se brindará la recomendación en el vector `vectorEleCompartidos` y la línea 32 agrega ítems del otro usuario en el vector `vectorEleCompartidos`.

```
22            foreach (KeyValuePair<string, double> pair in
23            dictionaryelemento1)
24            {
25              if (dictionaryelemento2.ContainsKey(pair.Key))
26              {
27              lstelemento1Vector.Add(pair.Value);
28              lstelemento2Vector.Add(dictionaryelemento2[pair.Key]);
29              }
30            }
31            vectorEleCompartidos.Add(lstelemento1Vector.ToArray());
32            vectorEleCompartidos.Add(lstelemento2Vector.ToArray());
33            return vectorEleCompartidos;
34            }
```

Código 2.4. Método `ObtenerVectorElementosSimilares` (parte 2 de 2)

El Código 2.5 muestra el método para el cálculo del coeficiente de correlación de Pearson en `CorrelacionPearson`. La línea 14 hace una comparación entre el tamaño de los vectores de los usuarios a comparar. Dichos vectores siempre tienen que ser iguales, ya que solo deben contener ítems que coinciden entre ellos, variando solo en las calificaciones que cada usuario dio al ítem. En las líneas 16 a la 31 se realizan los cálculos de las expresiones del numerador y del denominador de la Ecuación 1.1.

La línea 34 se usa en caso de que durante el cálculo el denominador arroje un valor de cero. Si esto es así, significa que la similitud entre los usuarios es nula y directamente el coeficiente de correlación es cero. Si no es así, la línea 35 calcula finalmente el coeficiente de correlación y se retorna dicho valor.

```
12  public static double CorrelacionPearson(double[] a, double[] b)
13    {
14    if (a.Length != b.Length)
15    throw new Exception("El tamaño de los vectores no es igual");
16  …
30    double num = pSum - (suma1 * suma2 / n);
31    double den = System.Math.Sqrt((suma1cuad -
32    System.Math.Pow(suma1, 2) / n) * (suma2cuad -
33    System.Math.Pow(suma2, 2) / n));
34    if (den == 0) return 0;
35    return num / den;
36    }
```

Código 2.5. Método `CorrelacionPearson`

2.2.3. Servicio WCF

Para implementar el servicio WCF se empleó el IDE de Visual Studio 2013 y el lenguaje de programación C#. Se creó un proyecto de biblioteca de servicios WCF, cuyo espacio de nombre es `ServicioWCF` como se muestra en la Figura 2.23. En este proyecto se creó la interfaz `IOperaciones`, y la clase `Operaciones`. El directorio de este proyecto, contiene

los archivos necesarios (como `Web.config`, y archivos de extensión `.cs` y `.svc`) del servicio. El código completo del servicio WCF se encuentra en el Anexo IV.

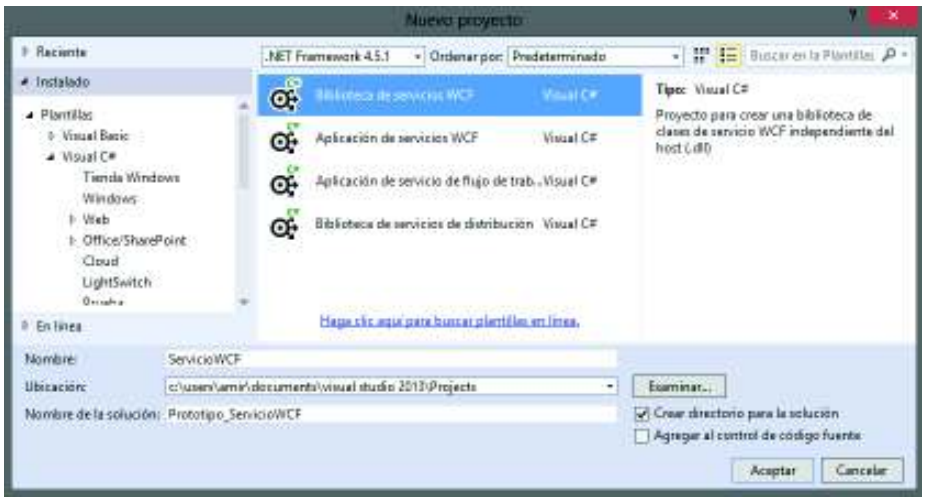


Figura 2.23. Creación de un proyecto para el servicio WCF en Visual Studio 2013

Para manipular datos almacenados en tablas, se creó una biblioteca de clases que permite representar datos como objetos. En el Anexo III se encuentra el código completo de la biblioteca de clases. En el Código 2.6, se presenta como ejemplo la clase `Usuario`. Las clases que conforman esta biblioteca son: `Publicidad`, `Locales`, `Usuario`, `UsuarioCalificacion`, y `Categorias`. Las variables definidas (líneas 3-6) representan las columnas de una tabla. Estas variables deben ser encapsuladas (líneas 10-14) para su correcta manipulación. Además, se define un constructor (línea 8) para la creación de objetos.

```
1   public class Usuario
2   {
3       private string nombre;
4       private string correo;
5       private string ocupacion;
6       private int contador;
7
8       public Usuario() ...
9
10      public int Contador ...
11      public string Nombre ...
13      public string Correo ...
14      public string Ocupacion ...
15  }
```

Código 2.6. Clase `Usuario` del Servicio WCF

En el Código 2.7 se muestra la interfaz del servicio WCF. En la línea 1 se define el atributo `[ServiceContract]` que indica que la interfaz es un contrato de servicio, la cual contendrá los métodos que expondrá el servicio WCF. En la línea 2 se define la interfaz.

```
1       [ServiceContract]
2       public interface IOperaciones
        ...
```

Código 2.7. Interfaz `IOperaciones` del Servicio WCF

El Código 2.8 muestra la firma del método `Registrar` de la interfaz `IOperaciones` del servicio WCF. El resto de firmas que forman parte de la interfaz son similares. En la línea 1 se define el atributo `[OperationContract]` para indicar que el método será expuesto por el servicio WCF. La línea 2 especifica que el método responde a peticiones HTTP POST, en la línea 3 se especifica el formato del URI[30], en la línea 4 se especifica el formato de pedido, y en la línea 5 el formato de respuesta, en este caso el formato es JSON[31]. En la línea 6 se define el método `Registrar` que acepta tres argumentos de tipo `string`, uno para el nombre de usuario, uno para su correo y uno para su ocupación, y retorna un valor de tipo `int` que indica el resultado de la operación.

```
1  [OperationContract]
2  [WebInvoke(Method          =  "POST",
3             UriTemplate     =  "Registrar/{nombre}/{correo}/{ocupacion}",
4             RequestFormat   =  WebMessageFormat.Json,
5             ResponseFormat  =  WebMessageFormat.Json)]
6  int Registrar(string nombre, string correo, string ocupacion);
```

Código 2.8. Firma del método `Registrar`

En el Código 2.9 se presenta parte de la clase `Operaciones`. Como se observa en la línea 1, esta clase implementa la interfaz `IOperaciones`. En la línea 4 se define la cadena de conexión que será utilizada para trabajar con la base de datos. A partir de la línea 8 se implementan todos los métodos definidos en la interfaz del servicio WCF y otros métodos utilizados para trabajar con la base de datos, y para generar recomendaciones.

```
1   public class Operaciones : Ioperaciones
2   {
4      private static string cadenaConexion =  ConfigurationManager.
5                                       ConnectionStrings["conexion"].
6                                       ConnectionString;
7      //Método para obtener informacion de usuarios registrados:
8      public List<Usuario> ObtenerUsuarios() ...
9      //Método para registrar un usuario en la Base de Datos:
10     int Registrar(string nombre, string correo, string ocupacion);
       ...
```

Código 2.9. Clase `Operaciones`

[30] URI (*Uniform Resource Identifier*): es una cadena de caracteres usados para identificar un recurso.
[31] JSON (*JavaScript Object Notation*): es un formato de archivos que emplea texto legible por humanos para transmitir objetos que están constituidos de pares atributo–valor y arreglos de tipos de datos.

A continuación, se presentan los métodos `EjecutarSentencia`, `Registrar`, y `EnviarCalificaciones`, el resto de métodos que conforman la clase `Operaciones` fueron implementados de forma similar con las peculiaridades de cada caso.

En el Código 2.10 se muestra el método `EjecutarSentencia`, el cual se utiliza para manipular la base de datos. En la línea 1, se puede observar que este método tiene un argumento de tipo `string`, el cual puede contener sentencias SQL *non query*, como `UPDATE`, `INSERT`, o `DELETE`.

En la línea 3, se inicializa la conexión con la base de datos, haciendo uso de la cadena de conexión, y en la línea 4 se inicializa un nuevo comando con la sentencia SQL. A continuación se abre la conexión con la base de datos (línea 11), y se ejecuta el comando (línea 12), finalmente se cierra la conexión con la base de datos (línea 20) y se retorna el resultado de ejecutar la sentencia SQL (línea 22).

```
1   public int EjecutarSentencia(string sentenciaSql)
2   {
3       SqlConnection conexion = new SqlConnection(cadenaConexion);
4       SqlCommand cmd = new SqlCommand(sentenciaSql, conexion);
        ...
7       int resultado = 0;
8
9       Try
10      {
11          conexion.Open();                    //Abrir conexión
12          resultado = cmd.ExecuteNonQuery(); //Ejecutar sentencia
13      }
14      catch (SqlException e)
15      {
16          resultado = 0;
17      }
18      Finally
19      {
20          conexion.Close();                   //Cerrar conexión
21      }
22      return resultado;
23  }
```

Código 2.10. Método `EjecutarSentencia`

En el Código 2.11 se muestra el método `Registrar` que forma parte de la interfaz del servicio WCF, que será expuesto para que un usuario pueda registrarse en el sistema. Este método acepta tres argumentos `string` para el nombre, el correo y la ocupación del usuario.

De la línea 3 a la 7, se construye la sentencia SQL *non query* utilizando los argumentos del método como los valores a ser almacenados en la tabla `Usuario`. En la línea 9, se retorna el resultado de usar el método `EjecutarSentencia`, con la sentencia SQL como argumento.

```
1   public int Registrar(string nombre, string correo, string ocupacion)
2   {
3     string sentenciaSql = "INSERT INTO Usuario ";
4            sentenciaSql += "(idUsuario, correo, ocupacion) VALUES ('";
5            sentenciaSql += nombre + "', '";
6            sentenciaSql += correo + "', '";
7            sentenciaSql += ocupacion + "')";
8
9     return EjecutarSentencia(sentenciaSql);
10  }
```

Código 2.11. Método `Registrar`

El método `EnviarCalificaciones` es presentado en el Código 2.12. En las líneas 1 y 2 se observa que este método acepta como argumentos dos objetos de tipo `string`, uno que contiene el nombre de usuario y otro que contiene sus calificaciones a locales comerciales, y con base en estos argumentos retorna una lista de recomendaciones de publicidad.

En la línea 6, se utiliza el método `ActualizarLogin`, el cual toma como argumentos el nombre de usuario y el valor entero uno, para actualizar la propiedad `contador` del usuario registrado en la base de datos.

El `string` que contiene las calificaciones realizadas a locales comerciales, se encuentra en formato JSON, estructurado como una colección de pares. Cada par contiene la calificación, y el identificador del local comercial.

Para manipular esta información contenida en el `string`, se procede a remover el formato JSON. Cada par en el `string`, se encuentra separado por los caracteres "`},{`", los cuales se reemplazan por el carácter barra vertical "`|`" (línea 8), de tal forma que mediante este carácter, cada par pueda ser separado y almacenado en un objeto tipo `var` (línea 13).

Se crea un objeto tipo `List` (líneas 15-16), para almacenar la categoría y subcategoría a la que pertenece un local comercial. Esta información va ser requerida por el algoritmo de recomendaciones.

Por cada par almacenado en objeto tipo `var` (línea 18), se obtienen dos valores, que corresponden al identificador del local, y a la calificación que realizó un usuario a dicho local. Para contener estos valores, en cada iteración se crea un objeto tipo `List` (línea 21).

Se procede a almacenar esta información en la base de datos mediante el método `IngresarCalificaciones` (línea 35), cuyos argumentos son el nombre del usuario, el identificador del local comercial calificado, y el valor de la calificación. Este método, es parecido al método `Registrar`, y difiere en los argumentos que acepta y en la sentencia

SQL utilizada para almacenar los datos. Se crea un objeto tipo `Categoria` (línea 35), y se procede a obtener la categoría y subcategoría a la que pertenece (líneas 37-38). Esta información se almacena en un objeto tipo `List` (línea 39).

```
1   public List<Publicidad> EnviarCalificaciones(string nombre,
2                                                 string calificaciones)
3   {
        ...
6       ActualizarLogin(nombre, 1);
7
8       calificaciones=calificaciones.Replace(@"},{","|");
        ...
13      var calificacionesVector = calificaciones.Split('|');
14
15      List<Categorias> categoriasCalificadas =
16      new List<Categorias>();
17
18      foreach (var parCalificacion in calificacionesVector)
19      {
            ...
21          List<int> dupla = new List<int>();
            ...
35          IngresarCalificaciones(nombre, dupla[0], dupla[1]);
36          Categorias interes = new Categorias();
37          interes.categoria = ObtenerInteres(dupla[0])[0].Categoria;
38          interes.subcategoria = ObtenerInteres(dupla[0])[0].Subcategoria;
39          categoriasCalificadas.Add(interes);
40      }
```

Código 2.12. Método `EnviarCalificaciones` (parte 1 de 3)

Las siguientes líneas muestran la implementación del algoritmo de recomendaciones. Previamente se obtuvo las categorías y subcategorías de los locales calificados por un usuario, de tal forma que cada par categoría/subcategoría representa un grupo, en el cual se va a realizar la recomendación.

En la línea 41 se define un objeto tipo `List<int>` que va a contener todos los identificadores de los locales recomendados por el algoritmo. Por cada par categoría/subcategoría (líneas 43-44), se crea un objeto tipo `List<UsuarioCalificacion>` que contendrá las calificaciones de los usuarios (líneas 46-47). A continuación, se obtiene de la base de datos, las calificaciones de los usuarios respecto a una categoría y subcategoría en específico, y se añaden a un objeto tipo `List<UsuarioCalificacion>` (líneas 49-53).

Se crea un objeto tipo `Formato` (línea 55) y a través de su método `ListaUso`, cuyos argumentos son las calificaciones de los usuarios y el nombre del usuario que requiere las recomendaciones, se obtienen los identificadores de locales comerciales recomendados por el algoritmo para una categoría y subcategoría en específico (líneas 57-58).

```
41      List<int> idsLocalesRecomendados = new List<int>();
42
43      foreach (KeyValuePair<string, int> categoria in
44                                          categoriasCalificadas)
45      {
46          List<UsuarioCalificacion> calificacionesCategoria = new
47                                    List<UsuarioCalificacion>();
48
49          foreach (UsuarioCalificacion pubC in
50          ObtenerCalificacionesInteres(categoria.Key, categoria.Value))
51          {
52              calificacionesCategoria.Add(pubC);
53          }
54
55          Formato obj = new Formato();
56
57          List<int> recomendaciones = obj.ListaUso(calificacionesCategoria,
58                                                   nombre);
59          ...
64      }
```

Código 2.12. Método `EnviarCalificaciones` (parte 2 de 3)

En la línea 66, se define un objeto tipo `List<Locales>` que contendrá los locales recomendados. A partir de los identificadores de locales comerciales, se obtiene una lista con información completa de locales comerciales (líneas 68-70).

En la línea 71, se define un objeto tipo `List<Publicidad>` que contendrá las recomendaciones de publicidad. Con base en la información de los locales comerciales, se busca en la base de datos la publicidad correspondiente a cada local (líneas 73-78) y se retorna una lista con recomendaciones de publicidad (línea 79).

```
66      List<Locales> localesRecomendados = new List<Locales>();
67      foreach (int id in idsLocalesRecomendados)
68      {
69          localesRecomendados.Add(ObtenerLocales(id));
70      }
71      List<Publicidad> recomendacionesPublicidad = new List<Publicidad>();
72      foreach (Locales local in localesRecomendados)
73      {
74          foreach (Publicidad publicidad in ObtenerPublicidad(local))
75          {
76              recomendacionesPublicidad.Add(publicidad);
77          }
78      }
79      return recomendacionesPublicidad;
80  }
```

Código 2.12. Método `EnviarCalificaciones` (parte 3 de 3)

A continuación, se muestran los cambios más importantes realizados al archivo `Web.config` para configurar el servicio WCF. Este archivo se encuentra en el Anexo III.

En el Código 2.13 se muestra la cadena de conexión utilizada para comunicarse con la base de datos. En la línea 2, se asigna el nombre de la cadena de conexión. En las líneas

3 a la 5 se asigna el motor de base de datos y en la línea 6, se especifica que el proveedor de datos .NET es `System.Data.SqlClient`.

```
1  <connectionStrings>
2        <add name             = "conexion"
3             connectionString = "Data Source=AMIR_MAIN\TESIS_SERVER;Initial
4                                 Catalog=InformacionUsuarios;
5                                 Integrated Security=True"
6             providerName     = "System.Data.SqlClient" />
7  </connectionStrings>
```

Código 2.13. Etiqueta `connectionString` del archivo `Web.config`

En el Código 2.14 se muestra la configuración de la etiqueta `services`. En la línea 1 se especifica el nombre de la clase que implementa la interfaz del servicio WCF, este nombre debe estar precedido por el espacio de nombres y un punto. En la línea 2 se especifica el nombre del comportamiento que tendrá el servicio.

Dentro de la etiqueta `services` se realiza la configuración de los *endpoints* que exponen el servicio WCF (líneas 4-11). En la línea 4 se especifica la URI del servicio, como se observa, es una cadena vacía, ya que se realizará el *hosting* con IIS. En la línea 5 se especifica el nombre de la interfaz que expone el servicio WCF, precedido por el espacio de nombres y un punto. En la línea 6 se define el tipo de *binding* que se utilizará para comunicarse con el *endpoint*. En la línea 7 se define el nombre del comportamiento que tendrá el *binding*, y en la línea 8 se define el nombre del comportamiento que tendrá el *endpoint*.

```
1   <services>
2         <service name="ServicioWCF.Operaciones"
3                  behaviorConfiguration="restBehaviour">
4           <endpoint address=""
5                     contract="ServicioWCF.IOperaciones"
6                     binding="webHttpBinding"
7                     bindingConfiguration="restLargeBinding"
8                     behaviorConfiguration="myWebEndPointBehaviour" />
9           <endpoint
10                    ...
11                    />
12        </service>
13  </services>
```

Código 2.14. Etiqueta `services` del archivo `Web.config`

2.2.4. Hosting del Servicio WCF

El *hosting* para el servicio WCF se realizó mediante el IIS de Windows. Para lo cual se creó un sitio web como se muestra en la Figura 2.24. En esta ventana se define el nombre del sitio web, el grupo de aplicaciones al que pertenece, la ruta de acceso física al directorio

que contiene los archivos del servicio WCF, y las características del enlace como el tipo, dirección IP y puerto. En nombre de *host* no es necesario configurar.

Figura 2.24. Configuración de un sitio web en IIS

En este caso, el sitio web es asignado al grupo de aplicaciones `DefaultAppPool`. Este grupo debe tener los permisos necesarios para acceder y manipular los archivos del servicio WCF. Para lo cual, en las propiedades de seguridad del directorio que contiene los archivos del servicio WCF, se agrega este grupo de aplicaciones y se da el permiso de control total, como muestra la Figura 2.25.

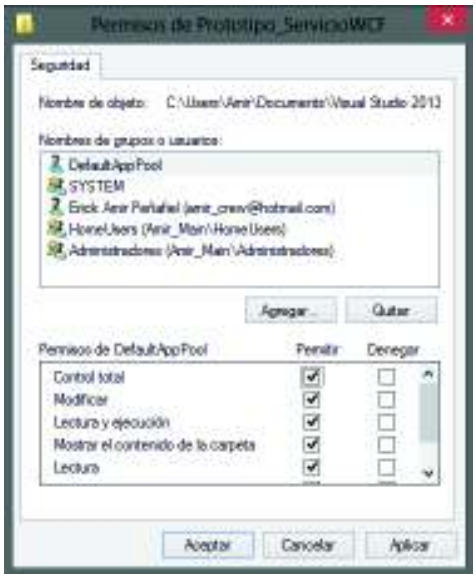


Figura 2.25. Permisos de seguridad de un directorio

Otro punto a considerar, es que el grupo de aplicaciones requiere permisos para comunicarse con la base de datos. Para lo cual, mediante SQL Server 2014 Management Studio, se realizaron las siguientes configuraciones al motor de base de datos:

1. Se creó un nuevo inicio de sesión para el grupo `DefaultAppPool`.
2. Se asignó el rol público al inicio de sesión.
3. Se proporcionaron permisos de lectura y escritura al inicio de sesión.

2.2.5. Cliente Android

La implementación del cliente Android fue realizada en Android Studio. Para la lógica del cliente, se empleó el lenguaje de programación Java, y para el desarrollo de la interfaz gráfica se utilizó el IDE de Android Studio. La solución del cliente Android se encuentra en el Anexo V.

a. Interfaz gráfica del cliente Android

Con el uso del editor de diseño de Android Studio, se realizó el diseño de los *layouts* de las actividades, y de las listas que presentarán datos recibidos del servidor. La Figura 2.26 muestra una captura de pantalla de este editor, en donde se observa la esquina superior izquierda, con todos los *widgets* disponibles para ser arrastrados y colocados en la posición requerida. En la esquina inferior izquierda, se muestra la jerarquía de vistas del diseño, y en lado derecho, se observa el diseño del *layout*.

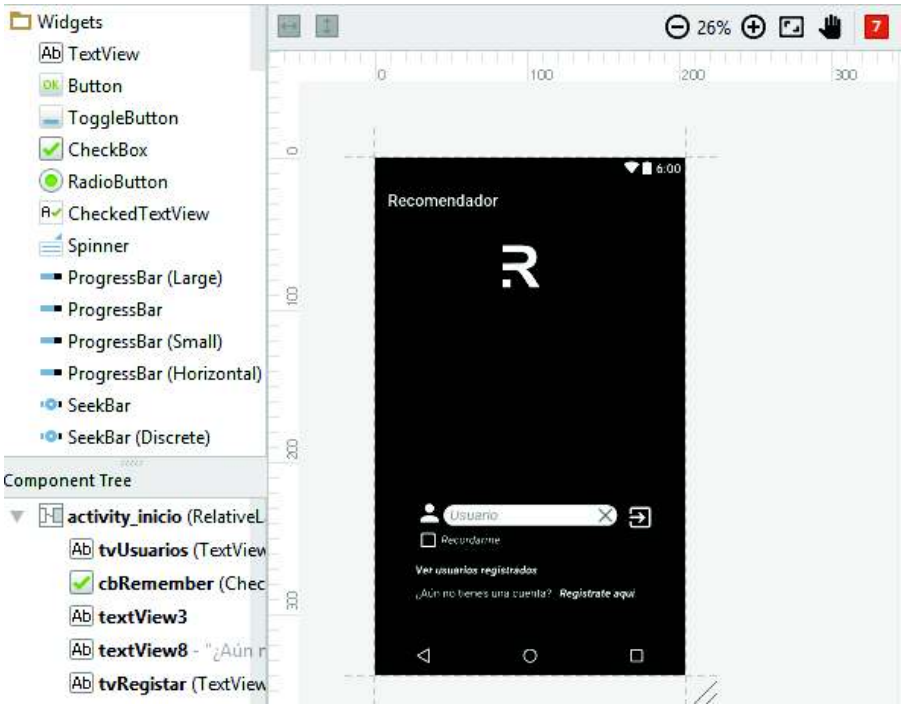


Figura 2.26. Editor de diseño de Android Studio

A continuación se presenta la implementación de la actividad `Recomendaciones`, como ejemplo de todas las actividades que conforman el cliente Android. En esta actividad, se

mostrará una lista de recomendaciones de publicidad. Todos los archivos de los *layouts* se encuentran en el Anexo V.

En el Código 2.15 se presenta parte del contenido del *layout* `Recomendaciones`. En las líneas 2-13, se define la etiqueta `RelativeLayout` con ciertos atributos, como el identificador del *layout* (línea 12), la orientación de la pantalla (línea 13), entre otros.

Se define el elemento `ListView` (línea 15), que será utilizado para mostrar una lista de datos. En la línea 28, se define el elemento `TextView` para mostrar texto en la pantalla, y en la línea 40 se define el elemento `FloatingActionButton`, el cual es un botón flotante, que mantiene una posición fija, mientras se desplaza a lo largo de una lista.

Dentro de las etiquetas de cada elemento se definen atributos propios de cada elemento; uno de los atributos comunes entre elementos, es el identificador de elemento.

```
1   <?xml version="1.0" encoding="utf-8"?>
2   <RelativeLayout
        ...
12      android:id="@+id/activity_recomendaciones"
13      android:screenOrientation="portrait">
14
15      <ListView...>
28      <TextView...>
40      <android.support.design.widget.FloatingActionButton...>
50
51  </RelativeLayout>
```

Código 2.15. *Layout* `Recomendaciones`

En la Figura 2.27 se muestra el *layout* `Recomendaciones`. En el editor de diseño de Android Studio, se colocaron los elementos definidos en el archivo XML del *layout*.

Figura 2.27. *Layout* `Recomendaciones`

Los datos que se muestran en un `ListView`, tienen su propio formato. Es decir, cada ítem de una lista, está definido en un *layout*. A continuación se presenta el *layout* `lista_publicidad`, que será empleado para mostrar un ítem de publicidad.

En el Código 2.16 se presenta parte del contenido del *layout* `lista_publicidad`. Se define el elemento `ImageView` (línea 7) para mostrar una imagen y varios elementos `TextView` para describir la publicidad.

```
1    <?xml version="1.0" encoding="utf-8"?>
2    <RelativeLayout ...>
6
7        <ImageView...>
20       <TextView...>
33       <TextView...>
44       <TextView...>
54       <TextView...>
67       <TextView...>
78       <TextView...>
89       <TextView...>
101
102  </RelativeLayout>
```

Código 2.16. *Layout* `lista_publicidad`

En la Figura 2.28 se muestra el *layout* `lista_publicidad`. Este es el formato de un ítem de una lista de recomendaciones de publicidad que se presentará al usuario. Los elementos de este ítem serán poblados mediante un adaptador.

Figura 2.28. *Layout* `lista_publicidad`

b. Lógica del cliente Android

En el Código 2.17 se muestra parte del contenido del archivo `AndroidManifest.xml`. Este archivo provee información esencial de la aplicación al sistema Android. Se define la etiqueta `manifest` (línea 2), que es el elemento raíz de este archivo. Dentro de esta etiqueta se establecen los permisos de la aplicación Android (líneas 5-14). Estos permisos son: abrir conexiones de red (líneas 5-6), acceder a información acerca de redes (líneas 7-8), acceder a información acerca de redes Wi-Fi (líneas 9-10), acceder a una ubicación aproximada (línea 12) y acceder a una ubicación precisa (líneas 13-14).

Se define la etiqueta `application` (líneas 16-25), en donde se establecen ciertos atributos del cliente Android, como: permitir respaldo (línea 17), el icono de la aplicación (línea 18), su nombre (línea 19), entre otros. Aquí también se define la etiqueta `meta-data` (líneas 23-25), para establecer la versión de *Google Play Services* que usará el cliente.

Las actividades que conforman el cliente se encuentras contenidas en la etiqueta `application`, como se puede se observar en las líneas 27-56.

```
1   <?xml version="1.0" encoding="utf-8"?>
2   <manifest ...>
5       <uses-permission
6       android:name="android.permission.INTERNET" />
7       <uses-permission
8       android:name="android.permission.ACCESS_NETWORK_STATE" />
9       <uses-permission
10      android:name="android.permission.ACCESS_WIFI_STATE" />
11      <uses-permission
12      android:name="android.permission.ACCESS_COARSE_LOCATION" />
13      <uses-permission
14      android:name="android.permission.ACCESS_FINE_LOCATION" />
15
16      <application
17          android:allowBackup="true"
18          android:icon="@drawable/r_icon"
19          android:label="@string/app_name"
20          android:theme="@style/AppTheme">
            ...
23          <meta-data
24             android:name="com.google.android.gms.version"
25             android:value="@integer/google_play_services_version" />>
27          <activity...>
36          <activity...>
40          <activity...>
44          <activity...>
48          <activity...>
52          <activity...>
56          <activity...>
57
58      </application>
59  </manifest>
```

Código 2.17. Contenido del archivo `AndroidManifest.xml`

En el Código 2.18 se presenta la etiqueta `activity` de la actividad `Inicio` en el archivo `AndroidManifest`. Se definen los parámetros de la actividad como: nombre de la actividad (línea 2), descripción (línea 3) y la orientación de la pantalla (línea 4).

Se define la etiqueta `intent-filter` (líneas 6-9), para indicar que es la actividad principal del cliente (línea 7) y que esta es la actividad que debe ser lanzada desde otra actividad del sistema cuando se pulse el icono del cliente (línea 8).

```
1   <activity
2      android:name=".Inicio"
3      android:label=""
4      android:screenOrientation="portrait">
5
6      <intent-filter>
7          <action android:name="android.intent.action.MAIN" />
8          <category android:name="android.intent.category.LAUNCHER" />
9      </intent-filter>
10  </activity>
```

Código 2.18. Etiqueta `activity` del `AndroidManifest`

A continuación, se explican las clases: `Local`, `Datos`, `SQLiteAdministrador`, `AdaptadorListaLocales`, `Holder`, `LocalesComerciales`, `EnviarCalificaciones`, e `Inicio`. Todas las clases del cliente Android se encuentran en el Anexo V.

En el Código 2.19, se presenta parte de la clase `Local`. Esta clase se utiliza para representar los datos de locales comerciales recibidos del servidor, como objetos. Se definen las variables que representan los datos (líneas 3-8) y son encapsuladas para su correcta manipulación (líneas 10-31). Se define un constructor para la creación de objetos (línea 9).

```
1   public class Local
2   {
3       public int idLocal;
4       public String descripcion;
5       public int calificacion;
6       public String imagenLocal;
7       public String categoria;
8       public int subcategoria;
9       public Local() {}
10      public  int getIdLocal() {return idLocal;}
11      public void setIdLocal(String imagen) {this.idLocal = idLocal;}

        ...

30      public void setSubcategoria(int subcategoria)
31      {this.subcategoria = subcategoria;}
32  }
```

Código 2.19. Clase `Local`

Las clases `Local` y `Publicidad` son utilizadas para el mismo propósito. La diferencia está en que la clase `Publicidad` se utiliza para representar datos de publicidad, y la clase `Local` se usa para representar datos de locales comerciales enviados por el servidor.

Las actividades se encargan de realizar peticiones HTTP, cuyas respuestas son demasiado grandes (superiores a 500 KB) para ser pasadas de una actividad a otra a través de un `intent`[32]. Por esta razón se realizan las clases `SQLiteAdminstrador` y `Datos`.

En el Código 2.20 se presenta parte de la clase `Datos`, la cual se utiliza para representar como objetos los datos almacenados en la base de datos SQLite del dispositivo. Se definen las variables que representan las columnas de una tabla de la base de datos (líneas 3-4) y se encapsulan estas variables para su correcta manipulación (líneas 13-18). Se definen dos constructores para la creación de objetos; uno sin argumentos (línea 6) y otro con dos argumentos `string`, para inicializar las variables (líneas 7-11).

```
1    public class Datos
2    {
3        public String informacion;
4        public String actividad;
5
6        public Datos(){}
7        public Datos(String actividad, String informacion)
8        {
9            this.setInformacion(informacion);
10           this.setActividad(actividad);
11       }
12
13       public String getInformacion() {return informacion;}
14       public void setInformacion(String informacion)
15       {this.informacion = informacion;}
16       public String getActividad() {return actividad;}
17       public void setActividad(String actividad)
18       {this.actividad = actividad}
19   }
```

Código 2.20. Clase `Datos`

En el Código 2.21 se presenta parte de la clase `SQLiteAdministrador`. Esta clase se emplea para manipular los datos de la base de datos SQLite del dispositivo. Para lo cual, se define la versión de base de datos (línea 3), el nombre de la base de datos (línea 4), nombre de una tabla (línea 5) y los nombres de las columnas que conformarán la tabla (líneas 6-7). Se define un constructor (línea 9) que recibe como argumento un objeto tipo `context`, el cual es el contexto de la actividad, en donde se creará el objeto que manejará la base de datos. Como se observa en la línea 1, esta clase se deriva de

[32] `Intent`: es una descripción de una operación a ser realizada.

`SQLiteOpenHelper`, por lo cual se implementan los métodos `onCreate` (líneas 14-22) y `onUpgrade` (líneas 24-25).

El método `onCreate` se emplea para crear una tabla, con las columnas definidas en las variables de la clase, este método acepta como argumento un objeto tipo `SQLiteDatabase`, que representa una base de datos SQLite. Se define la sentencia SQL para crear una tabla (líneas 16-20), y se ejecuta la sentencia en la base de datos (línea 21). El método `onUpgrade` se utiliza en caso que se requiera actualizar la base de datos.

El resto de métodos fueron codificados de forma similar al método `AgregarDatos`, que se presenta en el Código 2.22. `ObtenerDatos` (línea 42) se utiliza para obtener las filas de una tabla, `ActualizarDatos` (línea 51) se utiliza para actualizar la fila de una tabla, y `BorrarTabla` (línea 61) se utiliza para borrar una tabla de la base de datos.

```
1   public class SQLiteAdministrador extends SQLiteOpenHelper
2   {
3       private static final int version=1;
4       private static final String baseDeDatos="dataActivity";
5       private static final String tabla="dataActTable";
6       private static final String actividad="actividad";
7       private static final String informacion="informacion";
8
9       public  SQLiteAdministrador(Context context)
        {...}
13      @Override
14      public void onCreate(SQLiteDatabase db)
15      {
16           String CREATE_TABLE= "CREATE TABLE " + tabla +
17                                "("              + actividad +
18                               " TEXT PRIMARY KEY, " +
19                                 informacion + " TEXT"+")" ;
20
21          db.execSQL(CREATE_TABLE);
22      }
23      @Override
24      public void onUpgrade(SQLiteDatabase db, int oldVersion,
25                            int newVersion)
        {...}
31      public void AgregarDatos(Datos datos)
        {...}
42      public Datos ObtenerDatos(String actividadNombre)
        {...}
51      public int ActualizarDatos(Datos datos)
        {...}
61      public void BorrarTabla()
        {...}
70  }
```

Código 2.21. Clase `SQLiteAdministrador`

En el Código 2.22 se presenta el método `AgregarDatos` (líneas 31-41), el cual acepta como argumento un objeto tipo `Datos`, el cual contiene la información para almacenar en

la base de datos. Para esto, se obtiene una referencia de la base de datos que será usada (línea 33), se define la fila a insertarse en una tabla (líneas 34-37), se inserta la fila en la tabla (línea 38) y se cierra la base de datos (línea 40).

```
31      public void AgregarDatos(Datos datos)
32      {
33          SQLiteDatabase bdd = this.getWritableDatabase();
34          ContentValues valores = new ContentValues();
36          valores.put(actividad,datos.actividad);
37          valores.put(informacion,datos.informacion);
39          bdd.insert(tabla,null,valores);
40          bdd.close();
41      }
```

Código 2.22. Método `AgregarDatos`

Para poblar un `ListView` con datos de una lista, se hace uso de un adaptador que instanciará filas hasta que el `ListView` haya sido poblado completamente con los datos de la lista. Cuando el usuario se desplaza a lo largo lista, cada fila que deja la pantalla es guardada en memoria para ser usada posteriormente, y cada fila nueva que ingresa en pantalla, reutiliza una vieja fila guardada en memoria. Esto se conoce como reciclaje de filas [44].

La clase `AdaptadorListaLocales` se utiliza para poblar un `ListView` definido en el *layout* `Recomendaciones` (ver Código 2.15), con datos de una lista de locales comerciales. Las clases `AdaptadorListaLocales` y `AdaptadorListaExpandible`, son similares ya que utilizan el concepto de reciclaje de filas, pero difieren en los datos que utilizan para mostrar en las filas.

En el Código 2.23 se presenta parte de la clase `AdaptadorListaLocales`. Como se observa en la línea 1, esta clase se deriva de `ArrayAdapter`[33] cuyo parámetro es un objeto tipo `Local`, esto indica que se utilizará una lista de objetos tipo `Local` para poblar un `ListView`. Al derivarse de `ArrayAdapter` es necesario implementar el método `getView` (línea 21).

Se definen variables para indicar: el contexto de la actividad en donde se creará el objeto de esta clase (línea 3), el identificador del *layout* de la actividad (línea 4) y el identificador del *layout* de cada fila (línea 5), además define una lista para almacenar la información de locales comerciales (línea 7). Se define un constructor (líneas 11-14) para inicializar las variables definidas anteriormente.

[33] `ArrayAdapter`: retorna una vista para cada objeto en una colección de datos. Puede ser usado en un *widget* basado en una lista, como un `ListView` o un `Spinner`.

Cada vez que se realice un desplazamiento a lo largo de un `ListView` se llamará frecuentemente al método `findViewByID` (usado en la implementación del método `getView`), lo que puede disminuir el rendimiento, por lo que se crea la clase interna `Holder` (línea 120) que soluciona este inconveniente mediante el uso del patrón de diseño *View Holder* [45].

```
 1   public class AdaptadorListaLocales extends ArrayAdapter<Local>
 2   {
 3       private Context context;
 4       int layout;
 5       int layoutResourceId;
 6
 7       ArrayList<Local> locales = new ArrayList<Local>();
         ...
10
11       public AdaptadorListaLocales(Context context,
12                                    int layoutResourceId,
13                                    ArrayList<Local> locales,
14                                    int layout)
         {...}
20       @Override
21       public View getView(int position, View convertView,
22                                          ViewGroup parent)

         ...
120      public class Holder {...}
129 }
```

Código 2.23. Clase `AdaptadorListaLocales`

En el Código 2.24 se presenta la clase `Holder`. En esta clase se define un conjunto de vistas (líneas 122-126), definidas en el *layout* `lista_publicidad` (ver Código 2.16). Se hace referencia a este *layout*, dado que será utilizado por el adaptador para poblar las filas del `ListView`.

```
120  public class Holder
121  {
122      ImageView image;
123      TextView desc;
124      RatingBar ratingBarSend;
125      RatingBar ratingBar;
126      TextView cat;
127  }
```

Código 2.24. Clase `Holder`

El método `getView` se encarga de realizar el traslado de un objeto de una lista a una vista para ser mostrado en pantalla. Este método retorna una vista, que representa una fila en una posición específica dentro del `ListView`. A continuación se explica este método.

En el Código 2.25 se presenta el método `getView`. Se crea un objeto tipo `Holder` (línea 24) para almacenar una fila. A continuación se verifica si una fila está siendo reutilizada, de lo contrario se infla la fila con datos (líneas 26-47).

Si se cumple la condición, en la que la fila no está siendo reutilizada (línea 26), se infla la fila. Para esto, se instancia el *layout* de la actividad en un objeto tipo `LayoutInflater`[34] (líneas 28-29), y se infla la fila, especificando el *layout* de la lista (línea 30).

Finalmente, se almacenan las referencias a los componentes del *layout* en el objeto tipo `Holder`, para ser reutilizadas después (líneas 31-42); esto permite mejorar el rendimiento al no tener que llamar al método `findViewByID` frecuentemente. Si una fila está siendo reutilizada (línea 44), se obtiene el objeto tipo `Holder` (líneas 46-47).

```
20  public View getView(int position, View row,
21                                      ViewGroup parent)
22  {
23
24      Holder holder= null;
25
26       if(row == null)
27       {
28           LayoutInflater inflater =
29           ((Activity)context).getLayoutInflater();
30           row = inflater.inflate(layoutResourceId, parent, false);
31
32           holder = new AdaptadorListaLocales.Holder();
33           holder.desc = (TextView) row.findViewById(R.id.Local);
34           holder.image = (ImageView)
35           row.findViewById(R.id.imagenLocal);
36           holder.ratingBarSend = (RatingBar)
37           row.findViewById(R.id.rbCalificacionEnviarLocal);
38           holder.ratingBar = (RatingBar)
39           row.findViewById(R.id.rbCalificacioNRecibida);
40           holder.cat =
41           (TextView) row.findViewById(R.id.tvCategoria);
42           row.setTag(layout,holder);
43       }
44       Else
45       {
46           holder =  (AdaptadorListaLocales.Holder)
47           row.getTag(layout);
48       }
```

Código 2.25. Método `getView` (parte 1 de 2)

De la lista que contiene la información de locales comerciales se obtienen los datos para llenar una fila en una posición dentro de un `ListView` (línea 50). Con estos datos se

[34] `LayoutInflater`: instancia un archivo XML de un *layout*, en sus correspondientes objetos.

llenan los componentes del *layout* de la lista a través del objeto tipo `Holder` (lineas 52-61). Finalmente, se retorna la fila para ser renderizada en pantalla (línea 63).

```
50        final Local localObj = locales.get(position);
51
52        holder.desc.setText(localObj.descripcion);
53        holder.ratingBar.setRating(localObj.calificacion);
54        holder.cat.setText
55        (Category(localObj.categoria,localObj.subcategoria));
          ...
61        holder.image.setImageBitmap(imagenS);
62
63         return row;
64    }
```

Código 2.25. Método `getView` (parte 2 de 2)

En el Código 2.26, se presenta parte de la clase `LocalesComerciales`, donde se observa la declaración de: variables para los componentes de la interfaz gráfica como un `ListView` (líneas 3) y un `Button` (línea 4), un `string` para almacenar información de locales comerciales (línea 5), un `ArrayList` para representar en forma de lista la información de locales comerciales (línea 6). Se define el método `onCreate` (línea 19), donde se inicializa la actividad, y se define la clase interna `EnviarCalificaciones` (líneas 300-301), para realizar peticiones HTTP.

```
1     public class LocalesComerciales extends AppCompatActivity
2     {
3         ListView lista;
4         Button enviar;
5         String locales;
6         ArrayList<Local> listaLocalesReducida;
          ...
18        @Override
19        protected void onCreate(Bundle savedInstanceState)
          {...}

300       class EnviarCalificaciones extends AsyncTask<String,
301                                                    Void, String>
          {...}
407
408   }
```

Código 2.26. Clase `LocalesComerciales`

El método `onCreate` se presenta en el Código 2.27, en donde se infla la interfaz gráfica de la actividad (línea 22), se inicializan las variables para interactuar con los *widgets* de la interfaz gráfica, como un `ListView` (línea 24) y un `Button` (línea 25). En un `string` se obtienen datos sobre locales comerciales de la base SQLite del dispositivo (líneas 32-35). Cabe recalcar que para llegar a la actividad `LocalesComerciales`, el usuario primero, tuvo que pasar por la actividad `Categorias`, en donde se realizó la selección de categorías

de publicidad. Esta selección se envía al servidor a través de una petición HTTP y como respuesta se obtiene información de locales comerciales en formato JSON, la cual se almacena en la base de datos SQLite, para luego ser recuperada en la actividad `LocalesComerciales` debido al tamaño de estos datos.

```
19   protected void onCreate(Bundle savedInstanceState)
20   {
          ...
22        setContentView(R.layout.activity_locales_comerciales);
23
24        lista = (ListView) findViewById(R.id.lvLocales);
25        enviar = (FloatingActionButton) findViewById(R.id.Next);
          ...
32        SQLiteAdministrador bdd = new
33        SQLiteAdministrador(getApplicationContext());
34
35        locales=bdd.ObtenerDatos("Categorias").informacion;
          ...
```

Código 2.27. Método `onCreate` (parte 1 de 4)

Se inicia un `ArrayList` de objetos tipo `Local` (línea 41) que se utilizará para poblar un `ListView` a través de un adaptador. A continuación se define un bloque `try` (líneas 45-62), en donde se extrae la información de locales comerciales contenida en el `string` y se coloca en el `ArrayList`. Para esto, se convierte el `string` en un arreglo de objetos JSON (línea 47), del cual se obtienen objetos tipo `Local` (líneas 50-54) y se agregan al `ArrayList` (línea 60).

```
41        listaLocalesReducida = new ArrayList<Local>();
          ...
45        Try
46        {
47            JSONArray jsonArray=new JSONArray(locales);
48            for(int i=0;i<jsonArray.length();i++)
49            {
50                JSONObject objeto = jsonArray.getJSONObject(i);
51                Local locales_clase = new Local();
52                locales_clase.idLocal = objeto.getInt("IdLocal");
53                locales_clase.descripcion = objeto.
54                                            getString("Descripcion");
                  ...

60                listaLocalesReducida.add(locales_clase);
61            }
62        }
```

Código 2.27. Método `onCreate` (parte 2 de 4)

Luego, se emplea el adaptador `AdaptadorListaLocales` para poblar un `ListView` de la actividad `LocalesComerciales`. Se inicializa un objeto tipo `AdaptadorListaLocales` (líneas 70-73), mediante su constructor que tiene como argumentos: el contexto de la actividad (línea 70), el identificador del *layout* de la actividad (línea 71), un `ArrayList` de objetos tipo `Local` (línea 72), y el identificador del *layout* del `ListView` que se va a poblar (línea 73). Se configura el `ListView` a través del adaptador (línea 74).

```
70        adapter = new AdaptadorListaLocales(LocalesComerciales.this,
71                                            R.layout.lista_locales,
72                                            listaLocalesReducida,
73                                            R.layout.lista_locales);
74        lista.setAdapter(adapter);
```

Código 2.27. Método `onCreate` (parte 3 de 4)

A continuación, se especifica la acción a realizarse cuando se presione sobre un botón de la actividad. En este caso al presionar el botón se ejecutará una tarea asíncrona. A través de un objeto que hace referencia a un `Button` de la interfaz gráfica, se asigna el *listener* `setOnClickListener` (línea 86), el cual ejecutará el código escrito dentro del método `onClick` al presionar el botón (líneas 89-113). Se inicializa un `ArrayList` de objetos tipo `Hashmap` (líneas 101-102), para almacenar las calificaciones de locales comerciales que realizó el usuario, y finalmente se ejecuta la tarea asíncrona que realiza la petición HTTP (líneas 112-113).

```
86        enviar.setOnClickListener(new View.OnClickListener()
87        {
88            @Override
89            public void onClick(View v)
90            {
                  ...
100               ArrayList<HashMap<String,Object>> tmp =
101               new ArrayList<HashMap<String, Object>>();
102               ...
111
112               new EnviarCalificaciones(nombre,tmp.toString())
113                                        .execute();
                  ...
114           }});
```

Código 2.27. Método `onCreate` (parte 4 de 4)

En el Código 2.28 se presenta el bloque `buildTypes` del archivo `build.gradle`. Aquí se configura en campo `buildConfigField` para especificar el URL del servidor al que se va a conectar el cliente Android, tanto para *debug* (líneas 16-17), como para *release* (líneas 20-21).

```
    ...
14  buildTypes {
15          debug{
16              buildConfigField "String", "URL_Servidor",
17                               '"http://192.168.1.9:8080/"'
18          }
19          release {
20              buildConfigField "String", "URL_Servidor",
21                               '"http://190.96.111.180:80/"'
22              ...
23          }
24      }
    ...
```

Código 2.28. Bloque `buildTypes`

En el Código 2.29 se presenta parte de la clase interna `EnviarCalificaciones`. Esta clase se deriva de `AsyncTask`, y tiene como argumentos un `String`, un `Void` y otro `String` (línea 300). El primer argumento de `AsyncTask` indica el tipo de objeto que se debe pasar al ejecutar el método, el segundo parámetro indica el tipo de las unidades de progreso que se publican durante la operación en segundo plano, y el tercer parámetro, es el tipo del resultado que retorna la tarea.

Se realiza la declaración de variables para el resultado de ejecutar la tarea asíncrona (línea 302), para almacenar las calificaciones del usuario (línea 303), su nombre (línea 304), y para la URL que se usará para realizar la petición HTTP (línea 305). Se define un constructor para inicializar las variables (línea 306), se define el método `onPreExecute` (líneas 315-316), que se invoca antes de ejecutar la petición, el método `doInBackground` (línea 318), que se invoca en segundo plano y `onPostExecute` (línea 370), que se invoca al finalizar la petición.

```
300 class EnviarCalificaciones extends AsyncTask<String, Void, String>
301 {
302       String status=null;
303       String calificaciones;
304       String nombre;
305       String urlString;
306       EnviarCalificaciones(String nombre, String calificaciones)
          {...}
314       @Override
315       protected void onPreExecute()
316       {super.onPreExecute();}
317       @Override
318       protected String doInBackground(String... connUrl)
          {...}
369       @Override
370       protected void onPostExecute(String result)
          {...}
    }
```

Código 2.29. Clase `EnviarCalificaciones`

En el Código 2.30 se presenta el método `doInBackground`. En este método se realiza la petición HTTP POST. Aquí se define la URL para consumir el método `EnviarCalificaciones` del servicio WCF (líneas 323-325). La URL está constituido por la dirección del servidor, la cual se obtiene del archivo `build.gradle` (línea 323), el *path* (línea 234) y los argumentos del método (línea 325). Se crea un objeto tipo `URL` (línea 327) y se establece una conexión (líneas 328-329).

A continuación se establece el campo `Content-Type`[35] de la cabecera HTTP, con el valor `application/json`, que indica que la petición está en formato JSON, se define el tipo de petición HTTP POST (línea 332), y se establece que la petición podrá contener un cuerpo en el pedido (línea 333) y que se recibirá la respuesta (línea 334), cuya longitud no se conoce previamente (línea 335).

```
318       Protected String doInBackground(String... connUrl)
319       {
320           Try
321           {
                  ...
323               urlString = BuildConfig.URL_Servidor +
324                           "Operaciones.svc/EnviarCalificaciones/"+
325                           nombre+"/"+calificaciones;
326
327               URL urlObj = new URL(urlString);
328               HttpURLConnection urlConnection =
329               (HttpURLConnection)urlObj.openConnection();
330               urlConnection.setRequestProperty("Content-Type",
331                                                "application/json");
332               urlConnection.setRequestMethod("POST");
333               urlConnection.setDoOutput(true);
334               urlConnection.setDoInput(true);
335               urlConnection.setChunkedStreamingMode(0);
                  ...
337               JSONObject jsonObject = new JSONObject();
338               jsonObject.put("nombre",nombre);
339               jsonObject.put("intereses",calificaciones);
340               JSONArray jsonArray = new JSONArray();
341               jsonArray.put(jsonObject);
```

Código 2.30. Método `doInBackground` (parte 1 de 3)

A continuación se crea un objeto que almacenará información en formato JSON (línea 337), para incluir los parámetros del pedido al servicio WCF (líneas 338-339), y se coloca el objeto en un arreglo JSON (línea 341). Luego se crea un objeto tipo `BufferedOutputStream` (línea 343-345), para obtener un flujo de salida que permite escribir sobre la conexión, en

[35] `Content-Type`: campo de la cabecera HTTP, que indica el tipo de contenido de la petición, que puede ser POST o PUT.

la cual se escriben los bytes del arreglo JSON (línea 346), se procede a forzar su envío (línea 347), y se cierra el flujo de salida (línea 348).

```
343                 OutputStream out = new
344                 BufferedOutputStream(urlConnection
345                                         .getOutputStream());
346                 out.write(jsonArray.toString().getBytes());
347                 out.flush();
348                 out.close()
```

Código 2.30. Método `doInBackground` (parte 2 de 3)

Se crea un objeto tipo `InputStream` (línea 350), para obtener un flujo de entrada que permite leer de la conexión establecida. Se crea un objeto tipo `InputStreamReader`, para leer los bytes del flujo de entrada y a partir de estos bytes obtener un conjunto de caracteres (línea 351), los cuales son leídos mediante un *buffer* de tipo `BufferedReader` (línea 352).

A continuación se crea un objeto tipo `StringBuilder` (línea 353), para juntar cada línea que fue leída por el objeto tipo `BufferedReader` (líneas 354-358), y una vez que se han adjuntado todas las líneas se convierte el objeto tipo `StringBuilder` a un objeto tipo `String` (línea 360). Finalmente se retorna este objeto tipo `String` que contiene la respuesta a la petición HTTP.

```
350                 InputStream is = urlConnection.getInputStream();
351                 InputStreamReader isr = new InputStreamReader(is);
352                 BufferedReader br = new BufferedReader(isr);
353                 StringBuilder sb = new StringBuilder();
354                 String line;
355                 while((line=br.readLine())!=null)
356                 {
357                     sb.append(line);
358                 }
359                 br.close();
360                 status = sb.toString();
361             }
362             catch(Exception ex)
363             {
364             }
365             return status;
366         }
```

Código 2.30. Método `doInBackground` (parte 3 de 3)

En el Código 2.31 se presenta el método `onPostExecute`. Este método recibe como argumento el resultado del método `doInBackground`. Si este resultado no es nulo (línea 372), se procede a la actividad `Recomendaciones`, para lo cual se crea un objeto tipo `Intent` (líneas 377-378) y se añaden datos al objeto tipo `Intent`, como el nombre del usuario (línea 381), entre otros.

El resultado del método `doInBackground` contiene la respuesta a la petición HTTP, la cual se debe pasar a otra actividad para ser presentada al usuario. Sin embargo, el tamaño de esta respuesta puede superar los 500 KB, tamaño que no es soportado por un `Intent`. Por esta razón, se crea un objeto tipo `SQLiteAdministrador` (líneas 384-385) para almacenar esta respuesta en la base de datos SQLite del dispositivo (líneas 386-389), y posteriormente obtenerla en otra actividad. A continuación, se lanza la actividad `Recomendaciones` (líneas 391-392). Si la respuesta a la petición HTTP es nula, se muestra un mensaje que indica que se no se pudo establecer correctamente la conexión (líneas 401-403).

```
370       protected void onPostExecute(String result)
371       {
372          if(result!=null)
373          {
374              Try
375              {
376                   Intent recomendacionIntent =
377                   new Intent(LocalesComerciales.this,
378                             Recomendaciones.class);
                     ...
381                  recomendacionIntent.putExtra("nombre",nombre);
                     ...
384                  SQLiteAdministrador bdd = new
385                  SQLiteAdministrador(getApplicationContext());
386                  Datos datos = new
387                  Datos("LocalesComerciales",result);
                     ...
389                  bdd.AgregarDatos(datos);
390
391                  LocalesComerciales.this
392                  .startActivity(recomendacionIntent);
393              }
394              catch (Exception ex)
395              {
396                  ex.printStackTrace();
397              }
398          }
399          Else
400          {
401               Toast.makeText(getApplicationContext()
402               ,"Fallo al concectar con el servidor!"
403               ,Toast.LENGTH_SHORT).show();
                  ...
406           }
407       }
```

Código 2.31. Método `onPostExecute`

En el Código 2.32 se presenta parte de la clase `Inicio`, en la cual se realiza la obtención de la ubicación del dispositivo a través de *Google Play Services*. Como se observa en las líneas 2 y 3, esta clase implementa `OnConnectionFailedListener` y `ConnectionCallbacks`, para hacer uso de la API de *Google Play Services*.

`ConnectionCallbacks` provee *callbacks*[36] que son llamados cuando el cliente se conecta o desconecta del servicio. `OnConnectionFailedListener` provee de *callbacks* para escenarios en los que el resultado de intentar conectarse es fallido.

A continuación se realiza la declaración de variables: `GoogleApiClient` (línea 55) que permitirá conectarse con el servicio de *Google Play*, `Location` (línea 56) para obtener la ubicación del dispositivo, y `String` (línea 57) para almacenar las coordenadas de latitud y longitud de la ubicación del dispositivo.

```
1    public class Inicio extends AppCompatActivity implements
2    ConnectionCallbacks,OnConnectionFailedListener
3    {
          ...
55        GoogleApiClient clienteGoogle;
56        Location ultimaUbicacion;
57        private String punto;
          ...
61        @Override
62        protected void onCreate(Bundle savedInstanceState)
63        {
              ...
85            buildGoogleApiClient();
86            clienteGoogle.connect();
              ...
215       }
216       @Override
217       public void onConnectionFailed(ConnectionResult arg0)
218       {   Toast.makeText(this, "Se falló al intentar conectar...",
219           Toast.LENGTH_SHORT).show();}
220
221       @Override
222       public void onConnected(Bundle arg0)
          {...}
239
240       @Override
241       public void onConnectionSuspended(int arg0)
242       {   Toast.makeText(this, "Conexión suspendida..",
243           Toast.LENGTH_SHORT).show();}
244
245       protected synchronized void buildGoogleApiClient()
          {...}
          ...
560  }
```

Código 2.32. Clase `Inicio`

En el método `onCreate` de la clase `Inicio`, se llama al método `buildGoogleApiCliente` (línea 85), el cual inicializa la conexión con el servicio, y será presentado en el Código 2.34 y se realiza la conexión con el servicio (línea 86).

[36] *Callback*: es un método que es llamado cuando un evento ocurre.

El método `onConnectionFailed` (líneas 217-219) se llama cuando existe un error al conectar el cliente con el servicio, y muestra un mensaje indicando este error (líneas 218-219).

El método `onConnected` (línea 222) se invoca de forma síncrona, cuando se ha completado exitosamente la conexión con el servicio. Este método se presenta en el Código 2.33, en donde se realiza la obtención de las coordenadas del usuario.

El método `onConnectionSuspended` (líneas 241-243) se llama cuando el cliente se encuentra desconectado del servicio temporalmente. Para indicar este suceso se muestra un mensaje en pantalla (líneas 242-243).

En el Código 2.33 se presenta el método `onConnected`. Aquí se verifica que el cliente tenga el permiso necesario para obtener la ubicación del dispositivo. Si se cumple la condición, en la que no tiene permiso (líneas 224-226) se solicita el permiso (líneas 228-229).

Caso contrario, se obtiene la ubicación del dispositivo en un objeto tipo `Location` (líneas 233-234). A partir de este objeto se obtienen las coordenadas de latitud y longitud (líneas 236-237).

```
222  public void onConnected(Bundle arg0)
223  {
224      if (ActivityCompat.checkSelfPermission(this,
225          Manifest.permission.ACCESS_FINE_LOCATION) !=
226          PackageManager.PERMISSION_GRANTED)
227      {
228          ActivityCompat.requestPermissions(this, new
229          String[]{Manifest.permission.ACCESS_FINE_LOCATION},1);
230      }
231      Else
232      {
233          ultimaUbicacion = LocationServices.FusedLocationApi
234                                  .getLastLocation(clienteGoogle);
235      }
236      punto = String.valueOf(ultimaUbicacion.getLatitude())+"|"
237              +String.valueOf(ultimaUbicacion.getLongitude());
238  }
```

Código 2.33. Método `onConnected`

En el Código 2.34 se presenta el método `buildGoogleApiClient`. Aquí se construye un objeto cliente que se conecta con la API de *Google* para obtener la ubicación del dispositivo. Se crea una instancia de `GoogleApiClient` (líneas 247-251), se registra un *listener* para recibir eventos de conexión (línea 248), se registra otro *listener* para evento de conexión fallida (línea 249), se especifica la API para servicios de ubicación requerida (línea 250), y se construye el objeto para comunicarse con la API de *Google* (línea 251).

```
245  protected synchronized void buildGoogleApiClient()
246  {
247                clienteGoogle =  new GoogleApiClient.Builder(this)
248               .addConnectionCallbacks(this)
249               .addOnConnectionFailedListener(this)
250               .addApi(LocationServices.API)
251               .build();
252  }
```

Código 2.34. Método `buildGoogleApiClient`

2.2.6. Instalación del cliente Android

Una vez terminada la codificación del cliente Android, se precedió a generar el archivo de extensión `.apk` en Android Studio. En la Figura 2.29 se presenta una captura de pantalla al momento de instalar el cliente en un dispositivo Android. Aquí se puede observar los permisos que fueron establecidos en el archivo `AndroidManifest.xml` (ver Código 2.17).

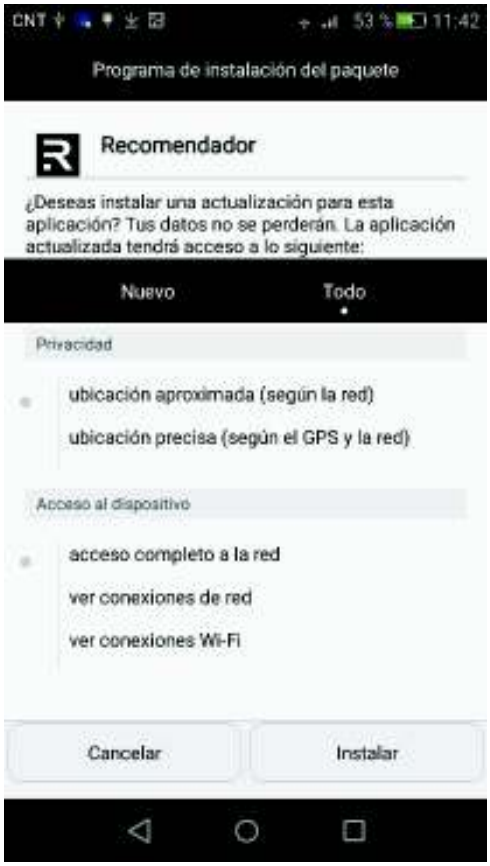


Figura 2.29. Instalación del cliente Android

A continuación se muestran ejemplos de la interfaz gráfica.

En la Figura 2.30 se presenta la actividad principal del cliente Android.

Figura 2.30. Actividad `Inicio`

En la Figura 2.31 se presenta la actividad `Registrar`, aquí un usuario puede ingresar sus datos para registrarse en el sistema.

Figura 2.31. Actividad `Registrar`

En la Figura 2.32 se presenta la actividad `Usuarios`, la cual presenta los usuarios registrados en el sistema.

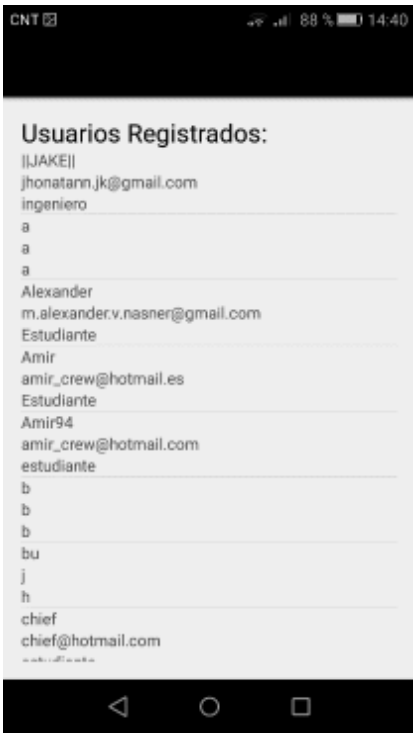


Figura 2.32. Actividad `Usuarios`

En la Figura 2.33 presenta la actividad `Categorias`, para seleccionar sus intereses de una lista de categorías de publicidad.

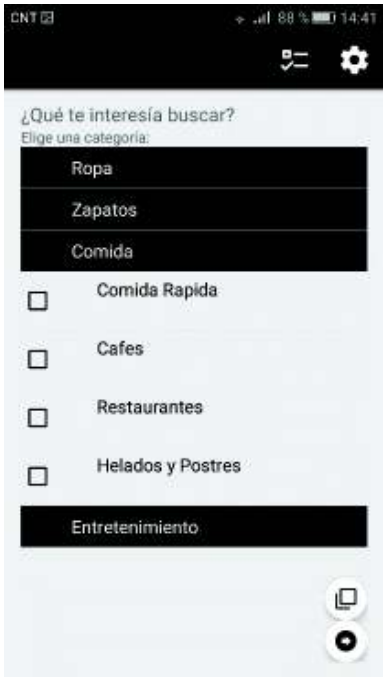


Figura 2.33. Actividad `Categorias`

En la Figura 2.34 se presenta la actividad `LocalesComerciales`. En esta actividad se presenta al usuario una lista de locales comerciales con base en su selección de intereses. Aquí el usuario puede calificar locales comerciales según sus experiencias previas.

Figura 2.34. Actividad `LocalesComerciales`

En la Figura 2.35 se presenta la actividad `Recomendaciones`, en la cual se presenta al usuario una lista de recomendaciones de publicidad con base en las calificaciones que realizó a locales comerciales.

Figura 2.35. Actividad `Recomendaciones`

En la Figura 2.36 se presenta la actividad `Ajustes`, el cual permite configurar el radio de búsqueda de locales comerciales.

Figura 2.36. Actividad `Ajustes`

3. RESULTADOS Y DISCUSIÓN

En este capítulo se presentan resultados a las pruebas realizadas al cliente Android en un dispositivo con el sistema operativo Android para verificar su funcionamiento, pruebas de conectividad entre el cliente Android y el servidor, y pruebas de funcionamiento tanto al servicio WCF como a la base de datos. Se probó el funcionamiento del algoritmo al realizar peticiones de recomendaciones y se observó que se generen recomendaciones apropiadas. Se realizaron 50 encuestas para validar la satisfacción de los usuarios respecto al prototipo. Para la realización de pruebas de conectividad se configuró una red inalámbrica con el direccionamiento de la Tabla 3.1.

Tabla 3.1. Direccionamiento IP de la Red Inalámbrica de pruebas

Dispositivo	Dirección IP
Access Point	192.168.1.1/24
Servidor	192.168.1.9/24
Dispositivo Android	192.168.1.4/24

3.1. Pruebas de funcionamiento de la base de datos

Para probar el correcto funcionamiento de la base de datos, se realizaron consultas sobre las tablas. A continuación, se presentan los resultados de las sentencias SQL ejecutadas, mediante SQL Server 2014 Management Studio. En la Figura 3.1 se presentan los primeros resultados de ejecutar la sentencia: `SELECT * FROM Usuario`. Como se puede observar, esta sentencia obtiene los usuarios registrados en la tabla `Usuario`.

idUsuario	correo	ocupacion	contador
‖JAKE‖	jhonatann.jk@gmail.com	ingeniero	0
Alexander	m.alexander.v.nasner@gmail.com	Estudiante	1
Amir	amir_crew@hotmail.es	Estudiante	1
Amir94	amir_crew@hotmail.com	estudiante	1
chief	chief@hotmail.com	estudiante	0

Figura 3.1. Resultado de la consulta a la tabla `Usuario`

En la Figura 3.2 se presentan los primeros resultados de ejecutar la sentencia: `SELECT * FROM Locales`, la cual obtiene los locales comerciales registrados en la tabla `Locales`.

IdLocal	Descripcion	Categoria	Subcategoria	Calificacion	Imagen
1	Carls jr	Comida	0	3	/9j/4AAQSkZJ
2	Supercines	Entretenimiento	0	3	/9j/4AAQSkZJ
3	Quito lunch	Comida	2	1	/9j/4AAQSkZJ
7	Papa jhones	Comida	0	4	/9j/4AAQSkZJ
9	Hansel & gretel	Comida	1	3	/9j/4AAQSkZJ

Figura 3.2. Resultado de la consulta a la tabla `Locales`

La Figura 3.3 presenta los primeros resultados de ejecutar la sentencia: `SELECT * FROM Publicidad`. Como se puede observar, esta sentencia obtiene la publicidad registrada en la tabla `Publicidad`.

idLocal	IdPublicidad	Descripcion	Imagen
1	1	Super duo. Western bacon + Famous star	/9j/4AAQSkZJRgABAQEA/
1	2	Hand breaded chicken tenders	/9j/4AAQSkZJRgABAQEA/
2	1	Salas de cine IMAX	/9j/4AAQSkZJRgABAQEA/
2	2	Cumpleaños de película: elige la película y nos ...	/9j/4AAQSkZJRgABAQEA/
7	1	Pizza familiar de especialidad	/9j/4AAQSkZJRgABAQEA/

Figura 3.3. Resultado de la consulta a la tabla `Publicidad`

En la Figura 3.4 se presentan los primeros resultados de ejecutar la sentencia: `SELECT * FROM UsuariosCalificacion`. Como se puede observar, esta sentencia obtiene las calificaciones de los usuarios a locales comerciales, registradas en la tabla `UsuariosCalificacion`.

idLocal	idUsuario	Calificacion
1	thomas	1
1	Perce	3
1	sofia	1
1	Jimmy	4

Figura 3.4. Resultado de la consulta a la tabla `UsuariosCalificacion`

3.2. Pruebas de funcionamiento del servicio WCF

Para probar el servicio WCF, se realizó el *hosting* en un servidor cuya dirección IP es 192.168.1.9/24 y el puerto por el que se accede al servicio es 8080. Se utilizó el software Fiddler[37], para realizar peticiones HTTP. En la Figura 3.5 se presenta la interfaz gráfica de Fiddler, donde se puede definir el método de la petición HTTP y el URL al cual se realizará la petición. En este caso el método de la petición es GET, y el método del servicio WCF que se va a consumir es `ObtenerUsuarios`. Este método obtiene los usuarios registrados en la base de datos.

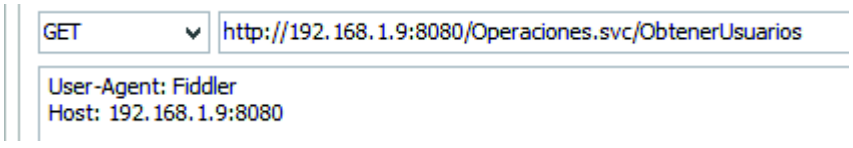


Figura 3.5. Interfaz de Fiddler

[37] Fiddler es una herramienta gratuita para depurar tráfico HTTP.

El resultado de esta petición se presenta en la Figura 3.6. Aquí se puede observar que el código de estado de respuesta de la petición es 200, que indica que la petición se realizó con éxito. Además se pueden observar otros parámetros como el protocolo, el *host*, el URL, y el tamaño del cuerpo de la respuesta en bytes.

#	Result	Protocol	Host	URL	Body
{js} 1	200	HTTP	192.168.1.9:8080	/Operaciones.svc/ObtenerUsuarios	857

Figura 3.6. Respuesta a una petición GET

El cuerpo de la respuesta a la petición se presenta en la Figura 3.7. Aquí se observa que la respuesta contiene los usuarios registrados en la base de datos, presentados en formato JSON.

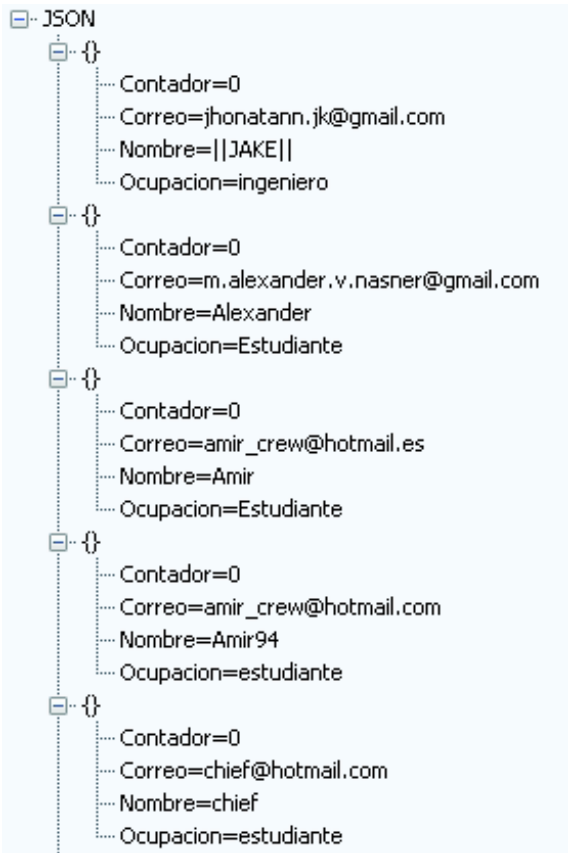


Figura 3.7. Cuerpo de la respuesta a una petición GET

Al comparar el resultado presentado en la Figura 3.1, en donde se obtienen todos los usuarios registrados consultado directamente en la base de datos, con los resultados presentados en la Figura 3.7, se puede concluir que el servicio WCF funciona correctamente, ya que ambos resultados presentan la misma información.

3.3. Pruebas de conectividad entre el cliente Android y el Servidor Web

Para determinar que hay conectividad entre el cliente Android y el Servidor, se ejecutó el comando PING[38] desde el servidor hacia el dispositivo Android. En la Figura 3.8 se presenta una captura de pantalla del dispositivo Android, en donde se puede observar que su dirección IPv4 es 192.168.1.4, como se estableció en la Tabla 3.1.

Figura 3.8. Información del dispositivo Android

En la Figura 3.9 se presenta la configuración IP de la tarjeta de red del servidor. Aquí se puede observar que su dirección IPv4 es 192.168.1.9, como se definió en la Tabla 3.1.

```
Adaptador de LAN inalámbrica Wi-Fi:

   Sufijo DNS específico para la conexión. . :
   Dirección IPv6 . . . . . . . . . . . . : 2800:370:88:da0::4
   Dirección IPv6 . . . . . . . . . . . . : 2800:370:88:da0:c07c:c530:c7e:4a6
   Dirección IPv6 . . . . . . . . . . . . : fda4:9947:96d9:1600:c07c:c530:c7e:4a6
   Dirección IPv6 temporal. . . . . . . . : 2800:370:88:da0:d966:7660:8b4a:515c
   Dirección IPv6 temporal. . . . . . . . : fda4:9947:96d9:1600:d966:7660:8b4a:515c
   Vínculo: dirección IPv6 local. . . . . : fe80::c07c:c530:c7e:4a6%3
   Dirección IPv4. . . . . . . . . . . . . . . . : 192.168.1.9
   Máscara de subred . . . . . . . . . . . . . . : 255.255.255.0
```

Figura 3.9. Configuración IP de la tarjeta de red inalámbrica del Servidor

En la Figura 3.10, se presenta el resultado de ejecutar el comando PING desde el servidor al dispositivo Android, a través de la red inalámbrica de pruebas. Como se puede observar el resultado de ejecutar el comando PING es exitoso, por lo que se concluye que ambos dispositivos pueden establecer una conexión.

[38] PING *Packet Internet Groper* es una herramienta usada para probar si un *host* es alcanzable en una red IP.

Figura 3.10. Prueba de conectividad

3.4. Pruebas de funcionamiento del Algoritmo de Recomendaciones

Las recomendaciones se realizan una por cada categoría. Para la realización de las pruebas se creó una aplicación de consola en el lenguaje de programación C#. Cada usuario posee un ID con el cual se identifica en la base de datos, ID de locales comerciales, que son únicos y representan el local dentro de una categoría y la calificación, que es la valoración del usuario respecto a ese local. El conjunto de entrenamiento consta de 25 usuarios (el usuario al que se le brinda la recomendación se denomina USR) y las calificaciones a locales comerciales son generadas de forma pseudo-aleatoria usando una distribución normal de tal manera que se pueda probar el modelo de recomendaciones y mostrar los resultados. Dicha información se encuentra en el Anexo VII.

Se realiza el cambio del formato de la información al de un diccionario, que no es más que una colección de claves (`Key`) y valores (`Values`), de tal manera que el objeto `Key` sea el ID de usuario y el objeto `Value` sea un grupo de dos elementos, ID de publicidad y calificación (como un diccionario anidado). Luego se filtran los locales comerciales que son similares a los del usuario al que se brinda la recomendación con sus respectivas calificaciones, para compararlos y encontrar la similitud entre usuarios.

3.4.1. Primera prueba

Cuando un usuario nuevo crea una cuenta, tiene que calificar al menos un local comercial de la o las categorías que seleccionó. Sin embargo, debe calificar más locales comerciales para que el momento de calcular el coeficiente de correlación de Pearson entre usuarios exista similitud entre ellos y pueda generarse una recomendación. Si esto no sucede, la recomendación será a través de la entrega de publicidad inesperada. La Tabla 3.2 muestra al usuario USR y sus calificaciones.

Tabla 3.2. Calificaciones del usuario USR

USR	**ID Local**	1	7	12
	Calificación	5	2	5

El usuario USR ha calificado 3 locales comerciales que se compararán con las calificaciones del resto de usuarios del Anexo VII. La Tabla 3.3 muestra los resultados de la similitud entre usuarios.

Tabla 3.3. Resultados de similitud entre usuarios

Usuarios comparados	Similitud
USR y Usuario A	0
USR y Usuario B	0
USR y Usuario C	0
USR y Usuario R	0
USR y Usuario S	0
USR y Usuario T	0
USR y Usuario U	0
USR y Usuario V	0
USR y Usuario W	0
USR y Usuario X	0
USR y Usuario Y	0

Al momento de depurar la aplicación de consola a través de la ventana de exploración de variables se puede apreciar el contenido de las distintas variables, donde en cada uno de los ejemplos de la Figura 3.11, el objeto `pair` corresponde al usuario que se compara con USR y `Similitud` corresponde al valor del coeficiente de correlación de Pearson.

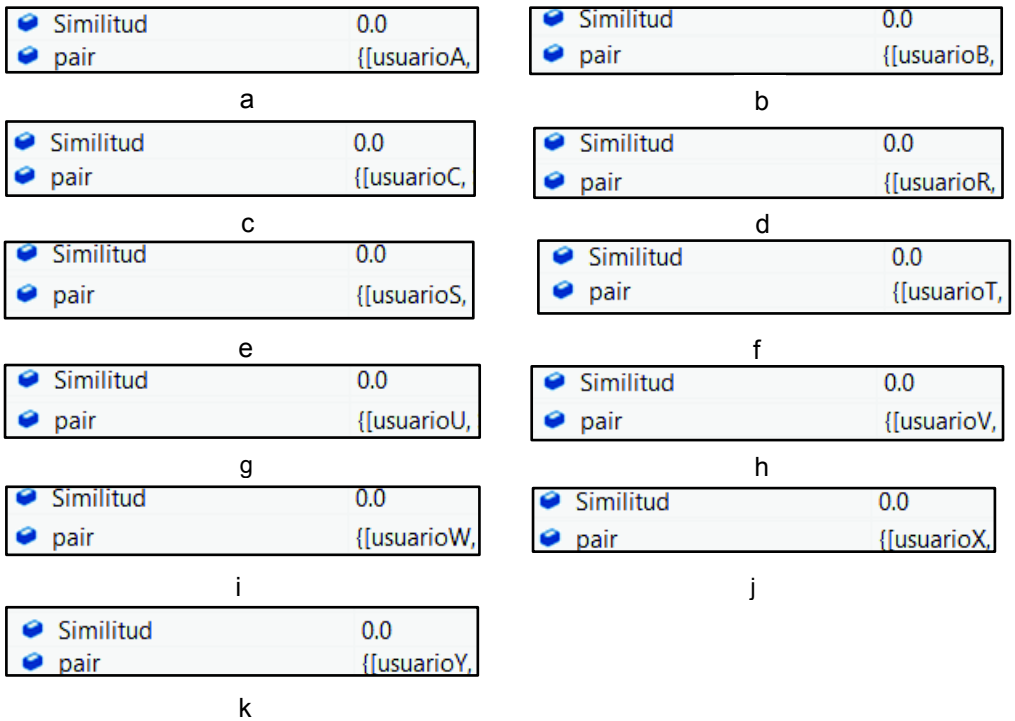


Figura 3.11. Similitud entre usuarios en depuración

Los valores de similitud son cero, por lo tanto, la recomendación se hará a través de la entrega de publicidad inesperada, dicha publicidad será aleatoria de locales comerciales aún no calificados por el usuario al que se brinda la recomendación. El resultado de este proceso se observa en la Figura 3.12. Los números mostrados en esta figura representan los identificadores de locales comerciales.

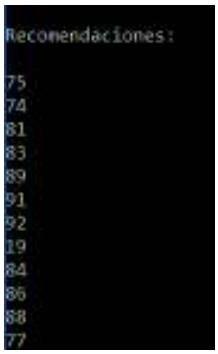


Figura 3.12. Identificadores de locales comerciales recomendados

La Figura 3.13 muestra las recomendaciones de los primeros 6 locales comerciales que se presentan en el cliente Android. Todos los resultados del cliente Android se encuentran en el Anexo VI.

ID Local 75: GoGreen

ID Local 74: Subway

ID Local 81: Pizza Hut

ID Local 83: KFC

ID Local 89: Stav

ID Local 91: Ceviches

Figura 3.13. Locales comerciales recomendados

No siempre son necesarias muchas calificaciones para que haya una similitud con algún usuario, pero es recomendable que haya mayor cantidad de calificaciones para que la recomendación sea más precisa.

3.4.2. Segunda prueba

Utilizando los mismos usuarios del Anexo VII se cambian las calificaciones del usuario USR para propósitos de prueba tal y como se muestra en la Tabla 3.4, con el propósito de mostrar el comportamiento del algoritmo de recomendaciones cuando el usuario al que se brindará la recomendación no tiene similitud alguna con el resto de usuarios a pesar de que tenga gran cantidad de calificaciones. USR ahora tiene 10 calificaciones que van a ser comparadas con el resto de usuarios. Luego se procede a aplicar la correlación de Pearson para encontrar la similitud entre ellos.

Tabla 3.4. Calificaciones a locales comerciales de USR

USR	**ID Local**	1	7	12	16	19	74	75	76	77	78
	Calificación	3	3	3	3	3	3	3	3	3	3

La Tabla 3.5 muestra los resultados de la similitud entre los usuarios y USR.

Tabla 3.5. Resultados de la similitud entre usuarios

Usuarios comparados	Similitud
USR y Usuario A	0
USR y Usuario B	0
USR y Usuario C	0
USR y Usuario D	0
USR y Usuario E	0
USR y Usuario F	0
USR y Usuario G	0
USR y Usuario H	0
USR y Usuario U	0
USR y Usuario V	0
USR y Usuario W	0
USR y Usuario X	0
USR y Usuario Y	0

Al momento de depurar la aplicación de consola a través de la ventana de exploración de variables se puede apreciar el contenido de las distintas variables, como se muestra en la Figura 3.14.

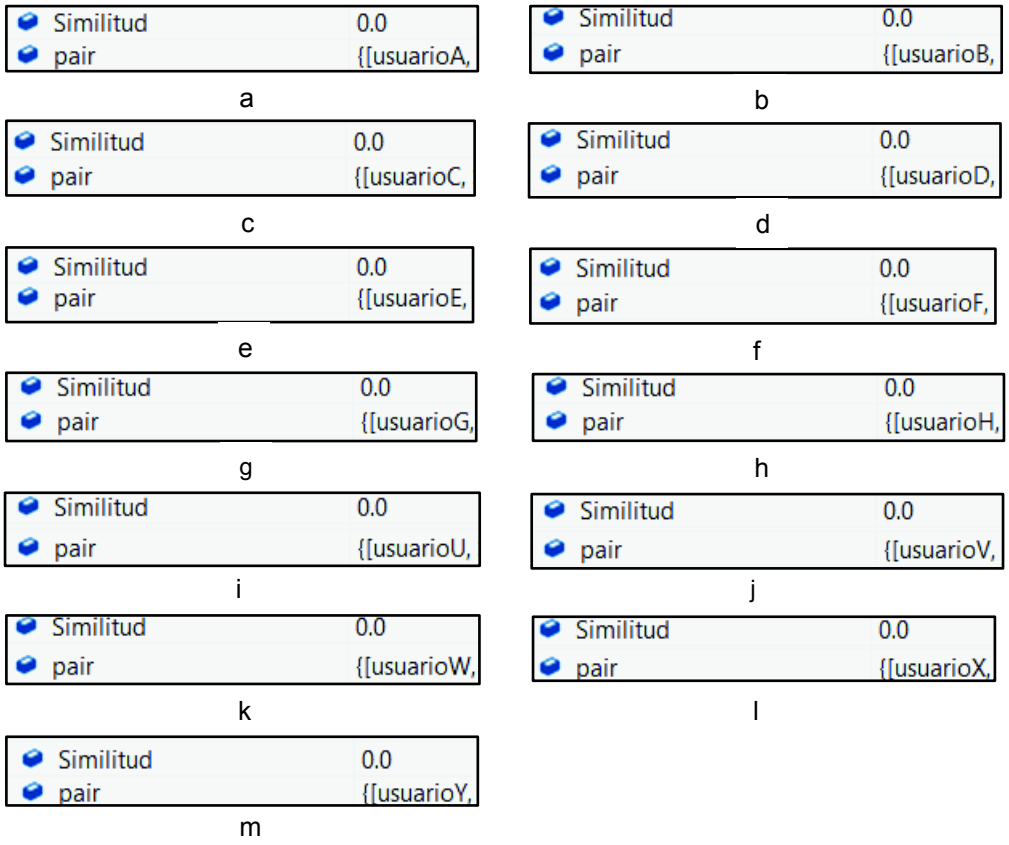


Figura 3.14. Resultados de similitud entre usuarios mostrado en depuración

Se aprecia que la similitud arroja resultados de cero. Entonces, la recomendación se hará a través de la entrega de publicidad inesperada, dicha publicidad será aleatoria de locales comerciales aún no calificados por el usuario al que se brinda la recomendación. El resultado de este proceso se observa en la Figura 3.15. Los números mostrados en esta figura representan los identificadores de locales comerciales.

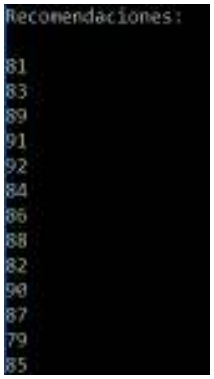
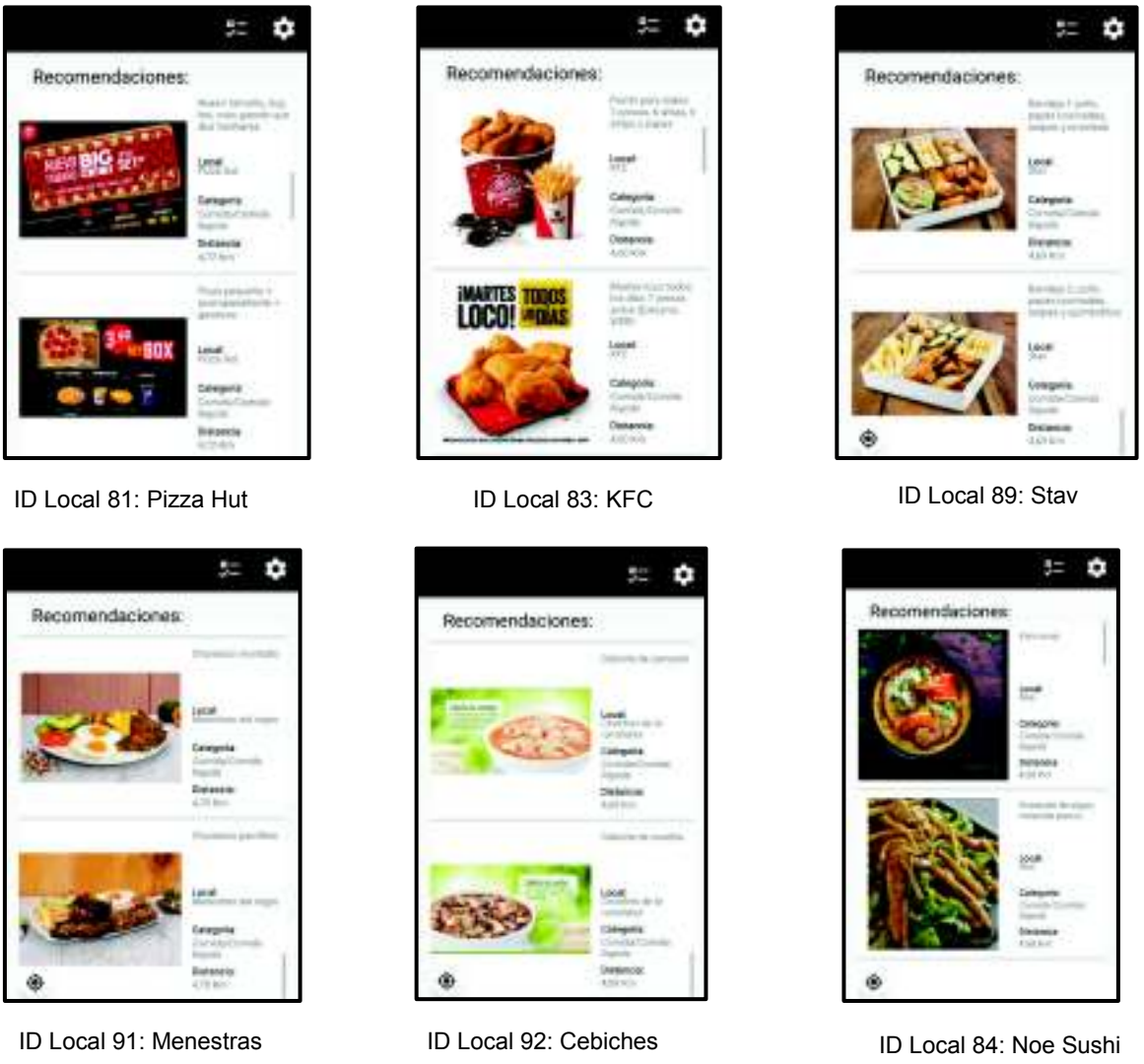


Figura 3.15. Recomendación con ítems inesperados

De igual forma, se obtiene como recomendación dichos locales comerciales en la aplicación. La Figura 3.16 muestra los 6 primeros resultados. Todos los resultados del cliente Android se encuentran en el Anexo VI.



ID Local 81: Pizza Hut | ID Local 83: KFC | ID Local 89: Stav

ID Local 91: Menestras | ID Local 92: Cebiches | ID Local 84: Noe Sushi

Figura 3.16. Locales comerciales recomendados al usuario

De estos resultados, se observa que se recomiendan locales comerciales variados, desde pizza hasta sushi o hamburguesas, esto debido a la aleatoriedad de la recomendación y solo se escogen las 10 primeras identificaciones de locales comerciales para evitar la sobrecarga a la aplicación y el exceso de información entregada al usuario.

3.4.3. Tercera prueba

Utilizando los mismos usuarios del Anexo VII se cambian las calificaciones del usuario USR para propósitos de prueba tal y como se muestra en la Tabla 3.6, con el propósito de mostrar el comportamiento del algoritmo de recomendaciones cuando el usuario al que se brindará la recomendación posee varias calificaciones y logra tener cierto nivel de similitud con otros usuarios. USR ahora tiene 10 calificaciones que van a ser comparadas con el resto de usuarios. Luego se procede a aplicar la correlación de Pearson para encontrar la similitud entre usuarios.

Tabla 3.6. Calificaciones a locales comerciales de USR

USR	ID Local	1	7	12	16	19	74	75	76	77	78
	Calificación	5	3	5	4	2	1	1	4	4	5

La Tabla 3.7 muestra los resultados de la similitud redondeados entre los usuarios y USR.

Tabla 3.7. Resultados del cálculo del coeficiente de correlación de Pearson

Usuarios comparados	Similitud
USR y Usuario A	-0.71
USR y Usuario B	-0.49
USR y Usuario C	-0.74
USR y Usuario D	0.61
USR y Usuario E	0.93
USR y Usuario F	0.96
USR y Usuario G	1.0
USR y Usuario H	1.0
USR y Usuario U	0
USR y Usuario V	-0.87
USR y Usuario W	-0.58
USR y Usuario X	-0.88
USR y Usuario Y	-0.94

La Figura 3.17 presenta la similitud entre usuarios al momento de depurar la aplicación de consola y mostrar las variables en la ventana de exploración de variables.

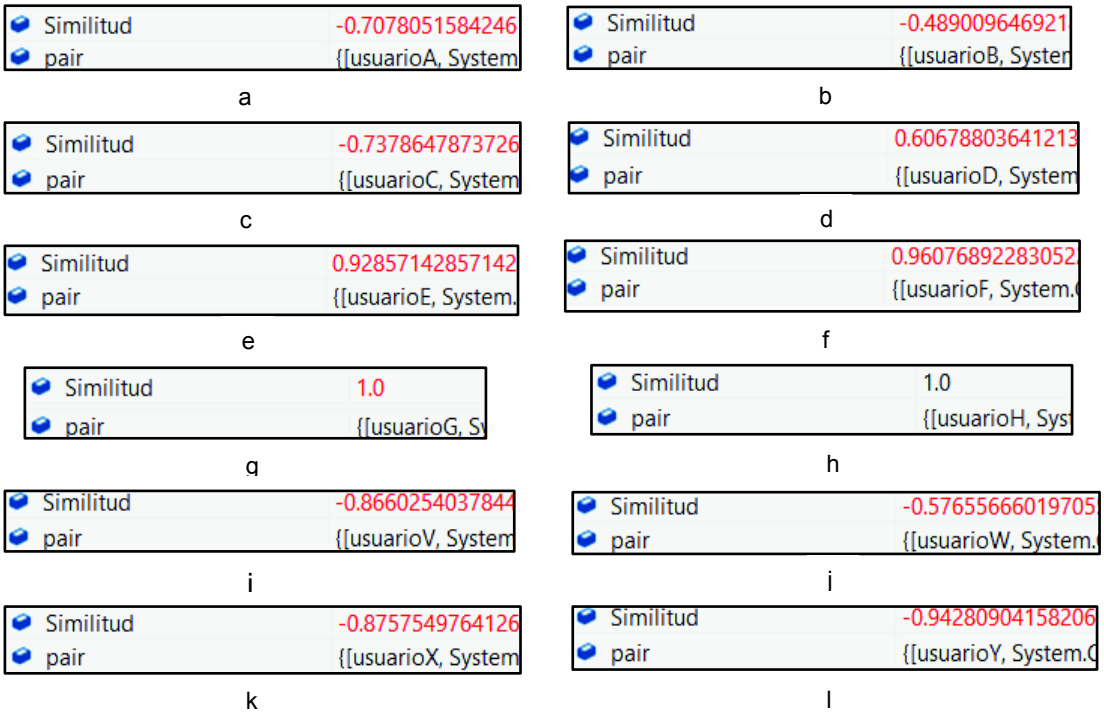


Figura 3.17. Resultados de similitud entre usuarios en depuración

USR tiene diferentes valores de similitud con los usuarios. El valor de similitud positivo y más cercano a uno indica un usuario con más relación con USR y por lo tanto se van a tomar los locales comerciales de estos usuarios como recomendación colocándolos de mayor a menor de acuerdo a dicho valor de similitud. La Figura 3.18 presenta el resultado de la recomendación:

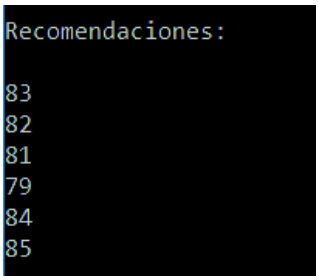


Figura 3.18. Resultados de la recomendación

La Figura 3.19 muestra la recomendación en el cliente Android.

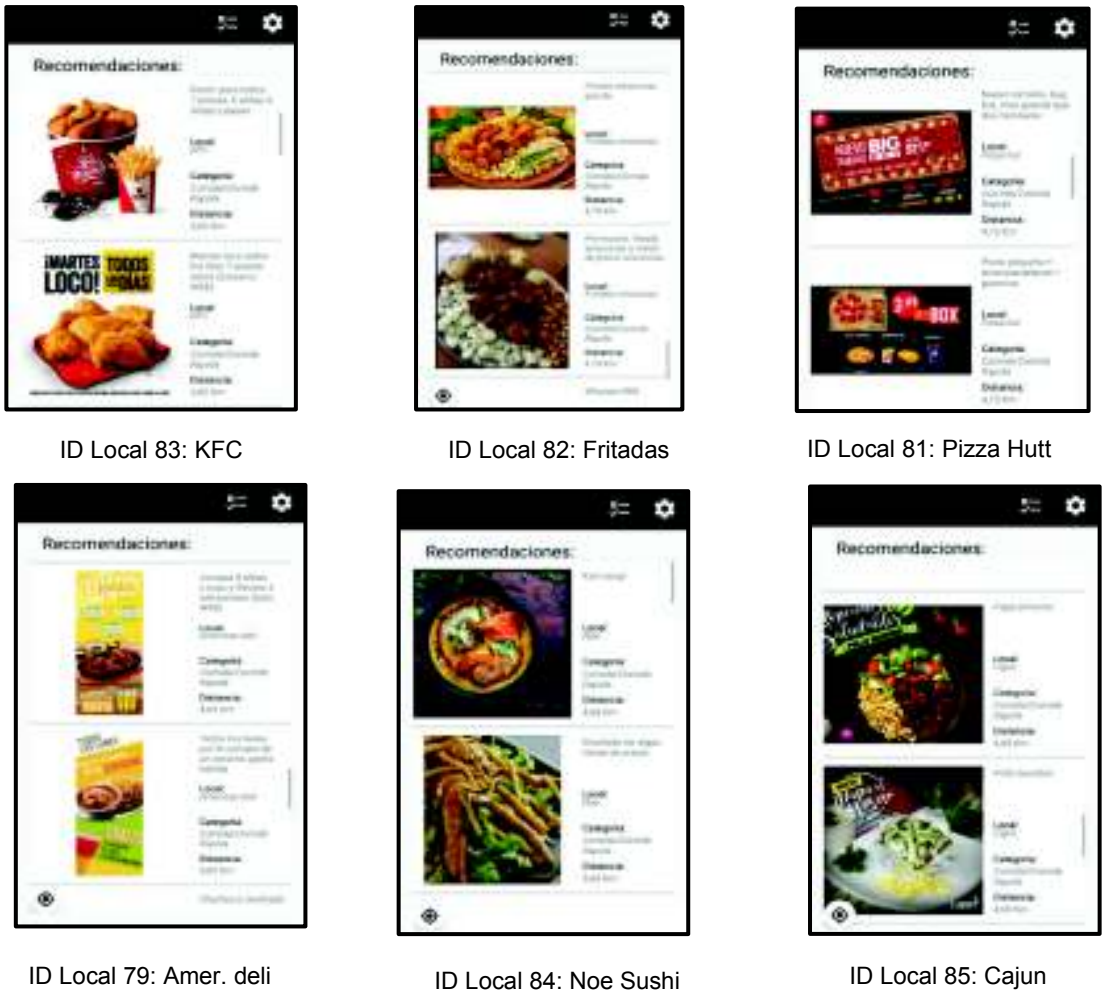


Figura 3.19. Recomendaciones brindadas al usuario

3.5. Pruebas de funcionamiento del cliente Android

En la Figura 3.20 se presenta la actividad `Usuarios`, donde se puede observar una lista de los usuarios registrados en el sistema. Al comparar con los resultados presentados en la Figura 3.1, se observa que se obtiene la misma información que al consultar directamente en la base datos. Se incluye un manual de usuario y el archivo de instalación `.apk` en el Anexo IX.

Figura 3.20. Prueba de funcionamiento a la actividad `Usuarios`

En la Figura 3.21 se muestra el flujo de actividades del cliente Android para realizar el registro de un usuario en el sistema. La primera actividad presentada es la actividad `Inicio`, en donde el usuario hace clic en el texto `Regístrate aquí` para dirigirse a la actividad `Registrar`. Aquí el usuario ingresa su información, y al hacer clic en el texto `verificar si el nombre es único`, se procede a realizar la verificación del nombre ingresado y se muestra un mensaje.

Finalmente, al hacer clic en el botón `Registrar`, se completa el proceso de registro y se regresa a la actividad `Inicio`, donde se muestra un mensaje indicando si el registro se realizó exitosamente.

Figura 3.21. Flujo de actividades para el registro de un usuario

En la Figura 3.22 se presenta el flujo de actividades de cliente Android para el ingreso de un usuario al sistema. Inicialmente se presenta la actividad `Inicio`, en donde se ingresa el nombre de usuario y al hacer clic en el botón de ingreso, se procede a verificar que el usuario esté registrado en el sistema. En este ejemplo, dado que es el usuario se encuentra registrado en el sistema, se procede a la actividad `Categorias`. Si el usuario no está registrado se mantiene en la actividad `Inicio`.

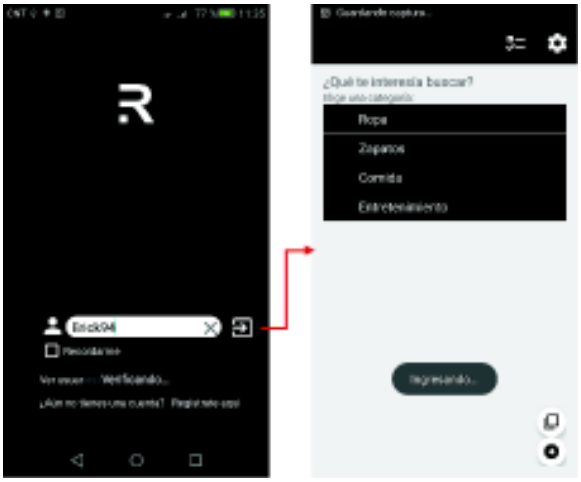


Figura 3.22. Flujo de actividades para el ingreso de un usuario

En la Figura 3.23 se presenta el flujo de actividades de cliente Android para la selección de categorías. Se presenta la actividad `Categorias`, donde el usuario elige de una lista las categorías de publicidad que le interesan. Una vez concluida la selección, al hacer clic en el botón para continuar, se procede a la actividad `LocalesComerciales`, en la cual se presentan locales comerciales que pertenecen a la selección de categorías realizada por el usuario.

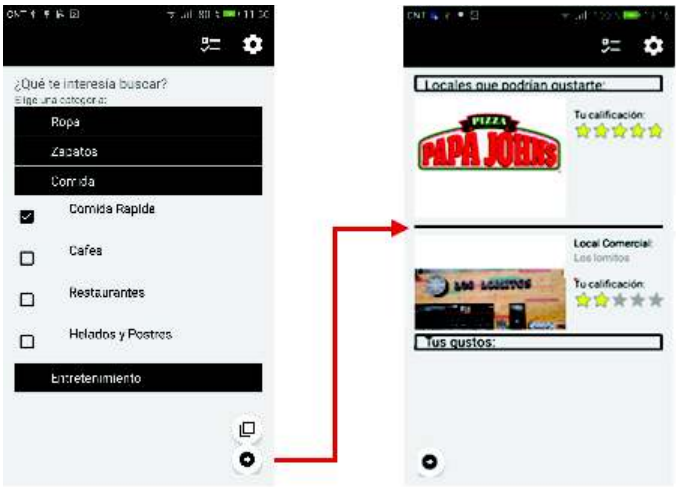


Figura 3.23. Flujo de actividades para la selección de categorías

En la Figura 3.24 se presenta el flujo de actividades de cliente Android para la calificación de locales comerciales. Se presenta inicialmente la actividad `LocalesComerciales`, que muestra una lista de locales comerciales, los cuales el usuario puede calificar con base a sus experiencias.

También se muestra un historial con las calificaciones que ha realizado previamente, en este caso se encuentra vacío ya que es un usuario que acaba de registrarse y no ha realizado calificaciones.

Una vez concluidas las calificaciones a locales comerciales, al presionar el botón de continuar, se procede a la actividad `Recomendaciones`. Aquí se presenta al usuario una lista de recomendaciones de publicidad con base en las calificaciones que realizó y a su ubicación.

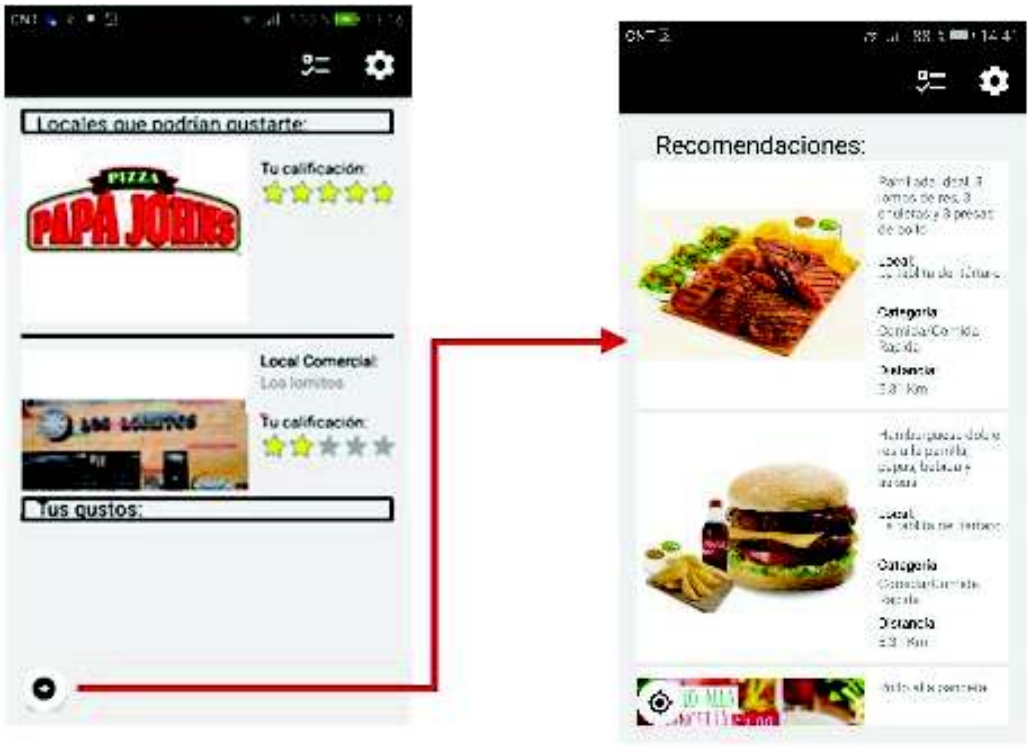


Figura 3.24. Flujo de actividades para la calificación de locales comerciales

3.6. Encuestas de Validación

Se distribuyó la Aplicación Android a 50 usuarios a los cuales se les pidió llenar una encuesta de validación luego de haber probado dicha aplicación. A continuación se muestran los resultados de la encuesta, la cual se encuentra en el Anexo VIII.

La primera pregunta pide calificar al usuario la forma en que la aplicación se presenta visualmente. Los resultados se muestran en la Figura 3.25 e indican que los encuestados encuentran que la interfaz gráfica de la aplicación es buena.

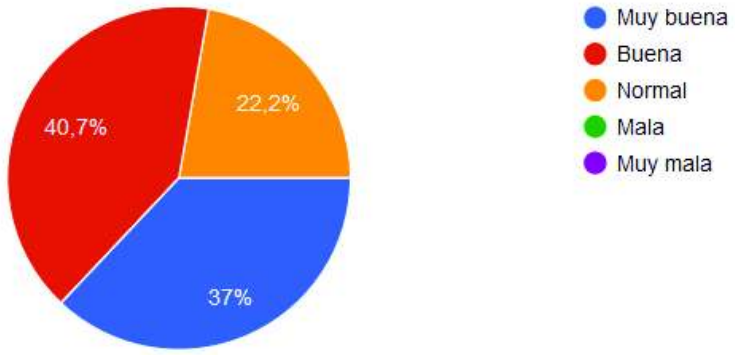


Figura 3.25. Respuestas a la primera pregunta: ¿Visualmente cómo calificaría la aplicación en forma general?

La segunda pregunta permite confirmar si al usuario le resultó intuitiva la navegación a través de la aplicación. Un 88.9% de los encuestados respondieron afirmativamente en que la navegación por la aplicación es intuitiva.

La tercera pregunta permite confirmar si el ingreso a la aplicación fue amigable, normal o malo. La Figura 3.26 muestran los resultados y se observa que un 57.4% de los encuestados les parece que la interfaz de ingreso a la aplicación es amigable y el restante 42.6% que es normal. Ninguno de los encuestados marcó el ingreso a la aplicación como malo.

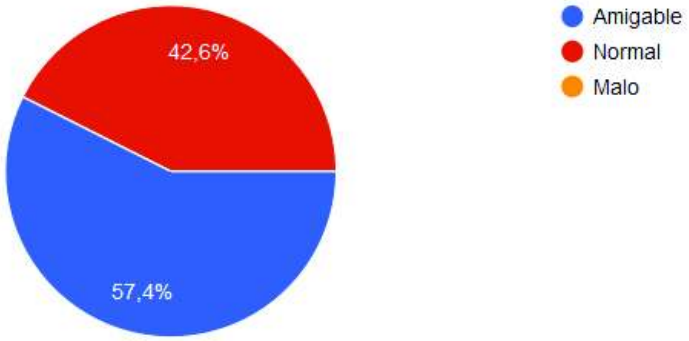


Figura 3.26. Respuestas a la tercera pregunta: ¿Cómo le pareció el ingreso a la aplicación?

La cuarta pregunta permite conocer si el tamaño de las imágenes con respecto a locales comerciales y publicidad es aceptable. Un 90.6% de los encuestados respondieron afirmativamente a la pregunta, es decir, no tienen problemas en identificar las imágenes de publicidad y locales comerciales mostrados en la aplicación.

La quinta pregunta permite conocer si es de utilidad el historial de locales comerciales presentado. La Figura 3.27 indica que el mayor porcentaje de los encuestados (50.9%) piensa que es útil el historial que muestra sus calificaciones realizadas a locales comerciales.

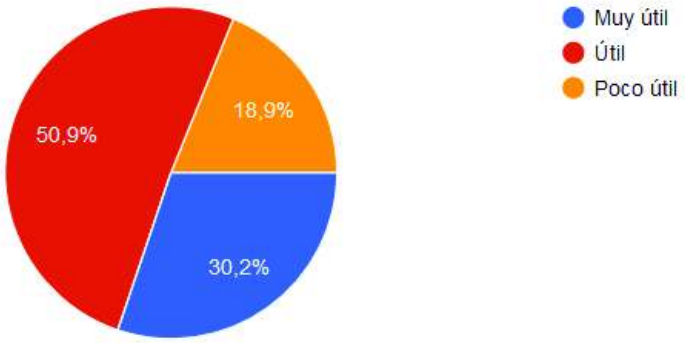


Figura 3.27. Respuestas a la quinta pregunta: ¿Cómo califica la presencia de un historial que muestra sus últimas calificaciones a locales comerciales?

La sexta pregunta permite confirmar si el número de locales comerciales mostrado es aceptable por el usuario. Se observa en la Figura 3.28 que un 63% de los encuestados les parece que la cantidad de comerciales mostrados es aceptable.

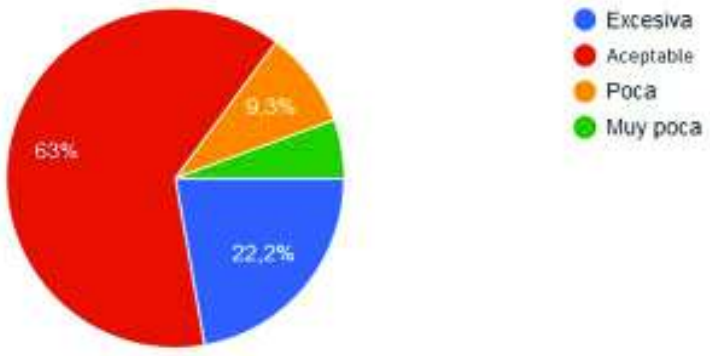


Figura 3.28. Respuestas a la sexta pregunta: ¿Cómo calificaría la cantidad de locales comerciales mostrada?

La séptima pregunta permite confirmar si la cantidad de recomendaciones es aceptable para el usuario. La Figura 3.29 indica que un 57.4% de los encuestados les parece que la cantidad de recomendaciones mostradas es aceptable.

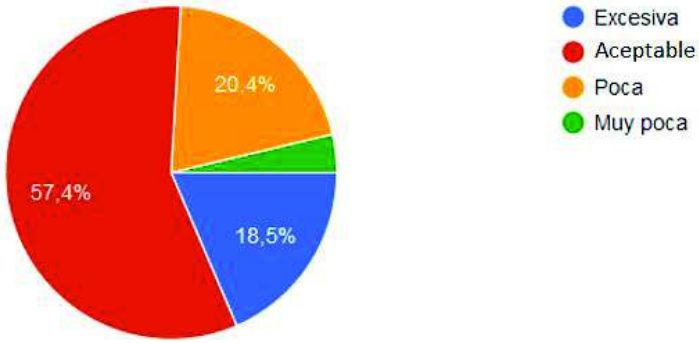


Figura 3.29. Respuestas a la séptima pregunta: ¿Cómo calificaría la cantidad de recomendaciones mostradas en la aplicación?

La octava pregunta permite confirmar si la forma en que se mostraron las recomendaciones es aceptable por el usuario. En los resultados de la Figura 3.30 se observa que un 51.9% de los encuestados piensa que fue buena.

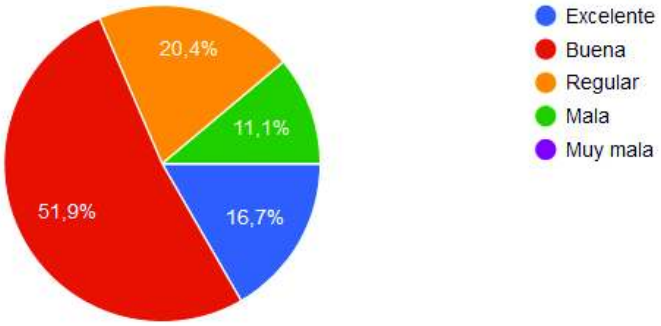


Figura 3.30. Respuestas a la octava pregunta: ¿Cómo calificaría la forma en la que se mostraron las recomendaciones de publicidad?

La novena pregunta permite confirmar si el alcance máximo de 100 metros establecido para buscar locales comerciales es aceptable por el usuario. Según la Figura 3.31 un 66.7% de los encuestados les parece que el alcance es lo suficiente para determinar los locales más cercanos.

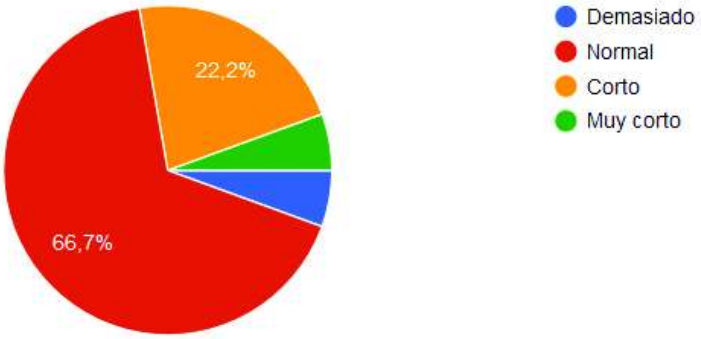


Figura 3.31. Respuestas a la novena pregunta: ¿Cómo calificaría el alcance máximo de 100 metros que tiene la aplicación para buscar locales comerciales?

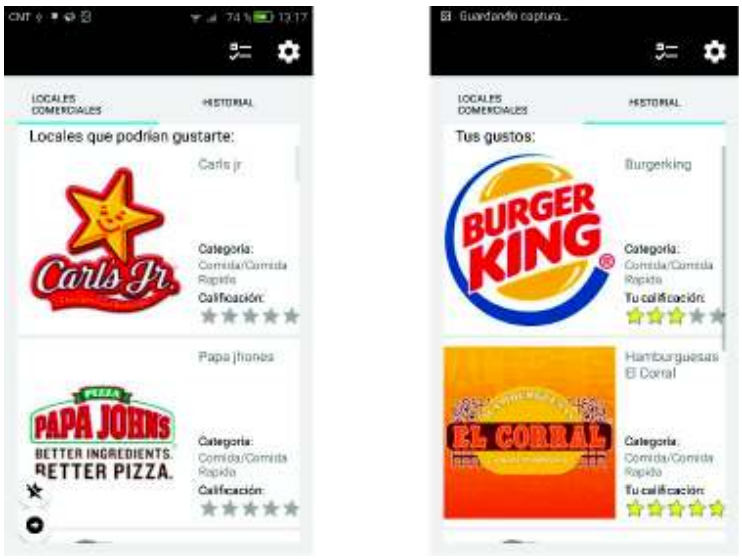
3.7. Correcciones a los componentes del prototipo

A continuación se presentan las correcciones realizadas a los componentes que conforman al prototipo de Sistemas de Recomendaciones, con base en los resultados de las encuestas de satisfacción de usuario de la sección 3.6.

3.7.1. Corrección al cliente Android

Respecto a la interfaz gráfica de usuario se realizaron cambios que ayudan a la navegación del usuario a través de la aplicación. En la Figura 3.32, se presenta la primera corrección a la actividad `LocalesComerciales`, en donde se separó a la lista de locales comerciales,

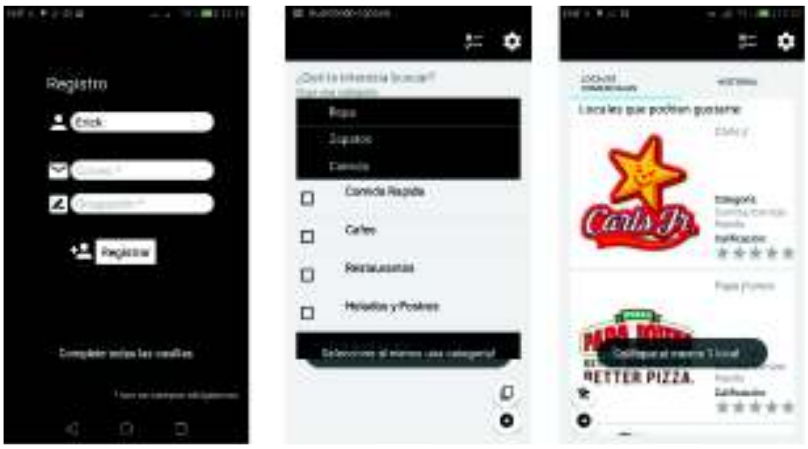
del historial de calificaciones del usuario. Esto se consiguió a través del uso del elemento `TabHost`.



a b

Figura 3.32. Cambios a la GUI del cliente Android

Se realizó un control de usuario para los elementos de la GUI del cliente Android como se presenta en la Figura 3.33. En el ejemplo, en la actividad `Registrar` se previene que el usuario deje espacios en blanco en los campos de escritura, mostrando un mensaje de advertencia y evitando que continúe. En la actividad `Categorias` se asegura que el usuario elija al menos una categoría, antes de continuar. Y en la actividad `LocalesComerciales` se asegura que el usuario califique al menos un local comercial antes de continuar.



a b c

Figura 3.33. Ejemplos de control de usuario en el cliente Android

4. CONCLUSIONES Y RECOMENDACIONES

4.1. Conclusiones

- El prototipo desarrollado permite recomendar publicidad de acuerdo a la localización y preferencias del usuario, por lo que se ha conseguido el objetivo general de este proyecto técnico. Para alcanzar este objetivo se utilizó Android Studio, *Windows Communitation Fundation*, MySQL y el algoritmo k-NN.

- Una vez concluido este trabajo se cuenta con un prototipo de sistema de recomendaciones, el cual está conformado por una base de datos que almacena los usuarios, sus calificaciones, información de locales comerciales y su publicidad, un algoritmo (k-NN) que recomienda locales comerciales con base en las preferencias del usuario y la similitud entre usuarios es calculada por medio del coeficiente de correlación de Pearson, un cliente Android que permite recolectar información acerca del usuario como sus preferencias y ubicación, y un servicio WCF que permite al usuario obtener recomendaciones de publicidad.

- Las características de los componentes que conforman al prototipo fueron diseñadas principalmente con base en los resultados obtenidos de las encuestas de requerimientos de usuarios. Sin embargo, el algoritmo de recomendaciones influye en el diseño del resto de componentes, como por ejemplo, el cliente Android tiene que ser diseñado acorde al algoritmo para la recolección de datos del usuario y la presentación de recomendaciones.

- Al comparar los modelos de recomendaciones *Collaborative Filtering*, *Content-Based recomendation*, *Knowledge-Based recomendation* y *Hybrid recomendation*, se escogió *Collaborative Filtering*, dado que por sus características permite hacer uso de las colaboraciones de las calificaciones de varios usuarios para identificar aquellos cuyos gustos sean similares y generar la recomendación.

- Considerando que en este caso la cantidad de datos que se tiene es pequeña, no se utilizó k-NN sino su variante modelo k-NN, en la cual no es necesario establecer el valor de k, dado que previamente los datos son agrupados y el número de muestras en cada grupo puede ser visto como el valor óptimo de k.

- La agrupación de datos es realizada al momento en el que el usuario hace la selección de categorías acorde a sus preferencias, de este modo se aplica el algoritmo k-NN en cada grupo categorías seleccionadas. Caso contrario, el tratar de generar una recomendación agrupando todas las categorías puede provocar que se

recomiende un local comercial dentro de categorías que ni siquiera el usuario ha escogido.

- Para poder realizar una recomendación de un local comercial específico se necesita que más de un usuario tenga un nivel de similitud lo suficientemente alto (cercano a uno) para que sus calificaciones se tomen en cuenta dentro de la recomendación. Esto muestra el problema de *cold-start* y es la razón por la cual se pobló la base de datos con varios usuarios para evitar un fallo al generar recomendaciones.

- Al realizar el diseño de los componentes del prototipo, el primer elemento que fue diseñado fue la base de datos, puesto que a partir de la información de locales comerciales y usuarios depende el diseño del resto de componentes. De tal forma que el siguiente diseño fue el algoritmo de recomendaciones, luego el servicio WCF y finalmente el cliente Android.

- Se empleó la arquitectura REST, la cual permite que desde el cliente Android se generen las peticiones utilizando el protocolo HTTP.

- Para enviar datos de una actividad a otra se hace uso de un `intent`. Sin embargo, en este prototipo dado que el tamaño de los datos supera los 500KB, no es posible utilizar un `intent` para pasar datos de un lado a otro, por lo cual se utilizó la base de datos SQLite, en la cual en una actividad se guarda la información y en otra actividad se lee la información requerida.

- El uso de clases internas en las actividades, permitió separar la lógica del cliente Android, de la lógica de comunicación, de tal forma que las clases internas solamente se encarguen de realizar las peticiones y obtener las respuestas, y las clases que las contienen se encarguen de manejar la interfaz gráfica.

- Dado que en ocasiones el usuario no entregó toda su información para brindar la recomendación usando el algoritmo de recomendaciones (ya sea porque es un usuario nuevo o porque el usuario no tiene similitud con el resto de usuarios) es necesario disponer de un mecanismo para resolver dicha situación. Por esta razón, se decidió utilizar el método conocido como recomendación inesperada el cual consiste en recomendar publicidad aleatoria de locales comerciales aún no calificados por el usuario al que se brinda la recomendación.

- Los resultados de las encuestas de validación permitieron realizar ajustes al prototipo, como la cantidad de publicidad que el usuario desea que se le muestre y mejoras a la interfaz gráfica del cliente Android.

4.2. Recomendaciones

- Se recomienda explorar el rendimiento del sistema cambiando el algoritmo de recomendaciones por ejemplo utilizando *Knowledge Discovery in Databases* o *Web mining*, considerando que se cuente con la cantidad de datos suficientes para aplicar estos algoritmos.

- Una alternativa para pasar datos de una actividad a otra, es el uso de fragmentos lo cual se sugiere se emplee en otros trabajos.

- Se recomienda que la interfaz de la aplicación sea mejorada de tal forma que sea de mayor agrado y más intuitiva para los usuarios. También que se agreguen más características como una sección de comentarios o la adición de enlaces a las diferentes redes sociales que cada local comercial posee.

- El servicio WCF no implementa seguridad, dado que esto no está contemplado en el alcance de este proyecto, sin embargo se recomienda que a posteriori se empleen las medidas de seguridad que ofrece WCF.

- El proyecto desarrollado permite recomendar publicidad, sin embargo se requiere que manualmente exista una persona que recolecte la información de publicidad y la almacene en la base de datos, se podría continuar en el desarrollo de este proyecto automatizando la recolección y almacenamiento de la información.

- Dar continuidad al desarrollo de este proyecto, agregando nuevas funcionalidades, como permitir al usuario agregar locales comerciales que encuentre y sean de su gusto para que contribuyan con información para generar mejores recomendaciones, o perfeccionar las ya implementadas, por ejemplo, el uso de variantes del método de recomendaciones como el filtrado colaborativo ítem a ítem que es destinado para el manejo de gran cantidad de usuarios.

- La implementación del servicio WCF basada en la arquitectura REST permitió realizar de una forma sencilla las peticiones en el cliente Android. Sin embargo, dado que REST soporta únicamente HTTP, se recomienda explorar el uso de SOAP en caso de que se requiera trabajar con diferentes plataformas.

5. REFERENCIAS BIBLIOGRÁFICAS

[1] F. Isinkaye, Y. Folajimi, y B. Ojokoh, “Recommendation systems: Principles, methods and evaluation”, *Egypt. Informatics J.*, vol. 16, núm. 3, pp. 261–273, 2015.

[2] M. Martín, “Nuevas tendencias en la publicidad del siglo XXI”, *Comun. Soc. ediciones y publicaciones. Sevilla - Zamora*, vol. Primera ed, p. 150, 2007.

[3] J. Herlocker, J. Konstan, L. Terveen, y J. Riedl, “Evaluating collaborative filtering recommender systems”, *ACM Trans. Inf. Syst.*, vol. 22, núm. 1, pp. 5–53, 2004.

[4] A. Almaraz y J. Goddard, “Sistema de recomendación”, *Komputer Sapiens*, pp. 1–40, 2015.

[5] Universitat Pompeu Fabra, *Hipertext.net.* Universidad Pompeu Fabra.

[6] D. Jannach, M. Zanker, A. Felfernig, y G. Friedrich, *Recommender systems: an introduction*, vol. 40. 2011.

[7] J. Aguilar, P. Valdiviezo-Díaz, y G. Riofrio, “A general framework for intelligent recommender systems”, *Appl. Comput. Informatics*, vol. 13, núm. 2, pp. 147–160, 2017.

[8] G. Guo, “LibRec - A Java Library for Recommender Systems”. [En línea]. Disponible en: https://www.librec.net/. [Consultado: 15-may-2017].

[9] T. Bogers y A. Van Den Bosch, *Collaborative and content-based filtering for item recommendation on social bookmarking websites*, vol. 532. 2009.

[10] C. Aggarwal, “Recommender Systems”, pp. 1–29, 2016.

[11] M. Balanovic, Y. Shoham, M. Balabanović, y Y. Shoham, “Content-Based, Collaborative Recommendation”, *Commun. Acm*, vol. 40, núm. 3, pp. 66–72, 1997.

[12] P. Lops, M. de Gemmis, y G. Semeraro, “Content-based Recommender Systems: State of the Art and Trends”, en *Recommender Systems Handbook*, Springer Science Business Media, 2011.

[13] B. Mobasher, “Data mining for web personalization”, *Adapt. Web Lect. Notes Comput. Sci.*, vol. 4321, pp. 90–135, 2007.

[14] R. Burke, “Hybrid Web Recommender Systems”, *Adapt. Web*, pp. 377–408, 2007.

[15] U. Fayyad, G. Piatetsky-Shapiro, y P. Smyth, “The KDD process for extracting useful knowledge from volumes of data”, *Commun. ACM*, vol. 39, núm. 11, pp. 27–34, 1996.

[16] B. Sarwar, G. Karypis, J. Konstan, y J. Riedl, “Application of Dimensionality Reduction in Recommender System - A Case Study”, *Architecture*, vol. 1625, pp. 264–8, 2000.

[17] G. Frawley, William y C. J. Matheus, “Knowledge Discovery in Databases: An Overview”, vol. 13, núm. 3, pp. 57–70, 1992.

[18] J. Saldaña, “El proceso de descubrimiento de conocimiento en bases de datos”, *Ingenierías*, vol. VIII, núm. 26, pp. 37–47, 2005.

[19] A. Kusiak, “Feature transformation methods in data mining”, *IEEE Trans. Electron. Packag. Manuf.*, vol. 24, núm. 3, pp. 214–221, 2001.

[20] B. Liu, B. Mobasher, y O. Nasraoui, “Web Data Mining”, *Web Data Min. Explor. Hyperlinks, Contents, Usage Data*, pp. 449–483, 2007.

[21] P. Bari y P. M. Chawan, “Web Usage Mining”, *J. Eng. Comput. Appl. Sci.*, vol. 2, núm. 6, pp. 34–38, 2013.

[22] S. Thirumuruganathan, “A Detailed Introduction to K-Nearest Neighbor (KNN) Algorithm | God, Your Book Is Great !!”, 2010. [En línea]. Disponible en: https://saravananthirumuruganathan.wordpress.com/2010/05/17/a-detailed-introduction-to-k-nearest-neighbor-knn-algorithm/. [Consultado: 05-jul-2017].

[23] J. Brownlee, “Tutorial To Implement k-Nearest Neighbors in Python From Scratch”, 2014. [En línea]. Disponible en: https://machinelearningmastery.com/tutorial-to-implement-k-nearest-neighbors-in-python-from-scratch/. [Consultado: 15-jul-2017].

[24] H. Ivan Chang Miranda, D. Viejo Luis Alejandro, F. Daniel Ruiz Moncayo, y P. Fabricio Echeverria Briones, “Sistema De Prediccion Y Recomendacion Personalizada Basada En Ranking De Items Homogeneos Usando Filtrado Colaborativo”, 2006.

[25] I. Triguero, J. Maillo, J. Luengo, S. Garcia, y F. Herrera, “From Big Data to Smart Data with the K-Nearest Neighbours Algorithm”, *Proc. - 2016 IEEE Int. Conf. Internet Things; IEEE Green Comput. Commun. IEEE Cyber, Phys. Soc. Comput. IEEE Smart Data, iThings-GreenCom-CPSCom-Smart Data 2016*, pp. 859–864, 2017.

[26] A. Hassanat, M. Abbadi, y A. Alhasanat, “Solving the Problem of the K Parameter in the KNN Classifier Using an Ensemble Learning Approach”, *Int. J. Comput. Sci. Inf. Secur.*, vol. 12, núm. 8, pp. 33–39, 2014.

[27] A. Lund y M. Lund, “Pearson Product-Moment Correlation - When you should run this test, the range of values the coefficient can take and how to measure strength of

association.” [En línea]. Disponible en: https://statistics.laerd.com/statistical-guides/pearson-correlation-coefficient-statistical-guide.php. [Consultado: 04-ene-2018].

[28] Graddle Inc., “Gradle Features”. [En línea]. Disponible en: https://gradle.org/features/. [Consultado: 04-ene-2018].

[29] Universidad Carlos III de Madrid, “Interfaces de usuario (View, Layouts) - Software de Comunicaciones”. [En línea]. Disponible en: https://sites.google.com/site/swcuc3m/home/android/api/librerias-basicas-interfaces-usuario. [Consultado: 04-ene-2018].

[30] Android Developers, “Adapter”, 2018. [En línea]. Disponible en: https://developer.android.com/reference/android/widget/Adapter. [Consultado: 04-ene-2018].

[31] Oracle, “Oracle”. [En línea]. Disponible en: https://www.oracle.com/mysql/index.html. [Consultado: 18-jul-2017].

[32] Tutorials Point, “MySQL Quick Guide”. [En línea]. Disponible en: https://www.tutorialspoint.com/mysql/mysql-quick-guide.htm. [Consultado: 18-jul-2017].

[33] SQLite, “About SQLite”. [En línea]. Disponible en: https://sqlite.org/about.html. [Consultado: 19-jul-2017].

[34] SQLite, “Distinctive Features Of SQLite”. [En línea]. Disponible en: https://sqlite.org/different.html. [Consultado: 19-jul-2017].

[35] K. Tharun, M. Prudhvi, y S. Reddy, “Advantages of WCF Over web services”, *Int. J. Comput. Sci. Mob. Comput.*, vol. 2, núm. 4, p. 6, 2013.

[36] Tutorials Point, “WCF Hosting wcf service”. [En línea]. Disponible en: https://www.tutorialspoint.com/wcf/wcf_hosting_service.htm. [Consultado: 20-jul-2017].

[37] M. Stopper y B. Gastermann, “Service-oriented communication concept based on WCF .NET for industrial applications”, *Proccedings Int. Multi Conf. Eng. Comput. Sci.*, vol. III, 2010.

[38] P. Tasgave y A. Dravid, “Friendbook : A Lifestyle based Friend”, vol. 3, núm. Iii, pp. 556–559, 2015.

[39] Q. Liao, B. Wang, Y. Ling, J. Zhao, y X. Qiu, “Improved Recommendation System

Using Friend Relationship in SNS", en *Transactions on Computational Collective Intelligence XIX - Volume 9380*, 2015, pp. 17–31.

[40] V. Adhikari *et al.*, "Unreeling Netflix : Understanding and Improving Multi-CDN Movie Delivery", *Infocom*, pp. 1620–1628, 2012.

[41] R. Bell y Y. Koren, "Lessons from the Netflix prize challenge", *ACM SIGKDD Explor. Newsl.*, vol. 9, núm. 2, p. 75, 2007.

[42] R. Smit, "Blog | The Intent extras size limit". [En línea]. Disponible en: https://www.neotechsoftware.com/blog/android-intent-size-limit. [Consultado: 22-mar-2018].

[43] Android Developers, "Parcelables and Bundles". [En línea]. Disponible en: https://developer.android.com/guide/components/activities/parcelables-and-bundles. [Consultado: 22-mar-2018].

[44] I. Krotzky, "Using an ArrayAdapter with ListView". [En línea]. Disponible en: https://github.com/codepath/android_guides/wiki/Using-an-ArrayAdapter-with-ListView. [Consultado: 05-abr-2018].

[45] Android Developers, "Making ListView Scrolling Smooth". [En línea]. Disponible en: https://developer.android.com/training/improving-layouts/smooth-scrolling. [Consultado: 04-may-2018].

[46] G. Linden, B. Smith, y J. York, "Amazon.com recommendations: Item-to-item collaborative filtering", *IEEE Internet Comput.*, vol. 7, núm. 1, pp. 76–80, 2003.

6. ANEXOS

ANEXO I: Encuestas de requerimientos de usuario

ANEXO II. Scripts para la creación de tablas

ANEXO III. Solución del algoritmo de recomendaciones (Código Fuente)

ANEXO IV. Solución del servicio WCF (Código Fuente)

ANEXO V. Solución del cliente Android (Código Fuente)

ANEXO VI. Resultados de recomendaciones del cliente Android

ANEXO VII. Calificaciones de clientes para pruebas del algoritmo de recomendaciones

ANEXO VIII. Encuestas de validación

ANEXO IX. Manual de usuario

ANEXO X. Diagramas de clases detallados de los componentes del prototipo

* Dada la extensión, todos los anexos se han incluido en el disco compacto adjunto a este documento.

ORDEN DE EMPASTADO