# Filtrado Colaborativo de Aplicaciones Móviles Basado en la información Social y Contextual del Usuario


Universidad del Cauca

ANEXOS

**Darío Fernando Chamorro Vela**
**Pablo Esteban Calvache López**

Director: PhD. Ing. Juan Carlos Corrales

Asesor: Ing. Luis Antonio Rojas Potosí

*Universidad del Cauca*
**Facultad de Ingeniería Electrónica y Telecomunicaciones**
**Departamento de Telemática**
**Línea de Investigación Aplicaciones y Servicios sobre Internet**
Popayán, Junio de 2013

**Filtrado Colaborativo de Aplicaciones móviles Basado en la Información Social y Contextual del usuario**

Chamorro Vela, Darío Fernando
Calvache López, Pablo Esteban

Director: PhD Ing. Juan Carlos Corrales Muñoz
Asesor: Ing. Luis Antonio Rojas Potosí

Fecha de sustentación: 23 de Agosto del 2013
Lugar: Salón 228, Facultad de Ingeniería Electrónica y Telecomunicaciones
Concepto: Aprobado
Jurados:
Ing. Javier Alexander Hurtado
Ing. Gustavo Ramírez

Facultad de Ingeniería Electrónica y Telecomunicaciones
Programa de Ingeniería Electrónica y Telecomunicaciones
Departamento de Telemática
Grupo de Ingeniería Telemática
Línea de investigación Aplicaciones y Servicios sobre Internet
Universidad del Cauca
Popayán – Cauca, 2013

# ANEXOS

**Pág.**

ANEXO A. Modelo de Diseño y Despliegue del Sistema…….....1
A.1. Diagrama de Paquetes del Sistema…….2
A.2. Diagrama de Clases del Sistema…….8
A.3. Diagrama de Navegabilidad…….31
A.4. Diagrama de Despliegue del Sistema…….....36
ANEXO B. Diagrama entidad relación……37
ANEXO C. Diagrama de flujo de las técnicas de recomendación……40
C.1. Técnica Tradicional……41
C.2. Técnica Contextual…….43
C.3. Técnica Social-Contextual…….46
ANEXO D. Intercambio de Mensajes…….....49
ANEXO E. Términos y Condiciones…….58
ANEXO F. Articulo ……...…..63
ANEXO F. Imagen y Banner VANILLA ……...…..67

# ANEXO A

## MODELO DE DISEÑO DEL SISTEMA Y MODELO DE DESPLIEGUE

# ANEXO A

## A.1. Diagrama de Paquetes

### A.1.1. Servidor central

En la Figura 1, se representa el diagrama de paquetes del servidor web central, encargado de proveer recomendaciones, de la recepción y procesamiento de la información que se transmiten desde los móviles, entre otras funciones.

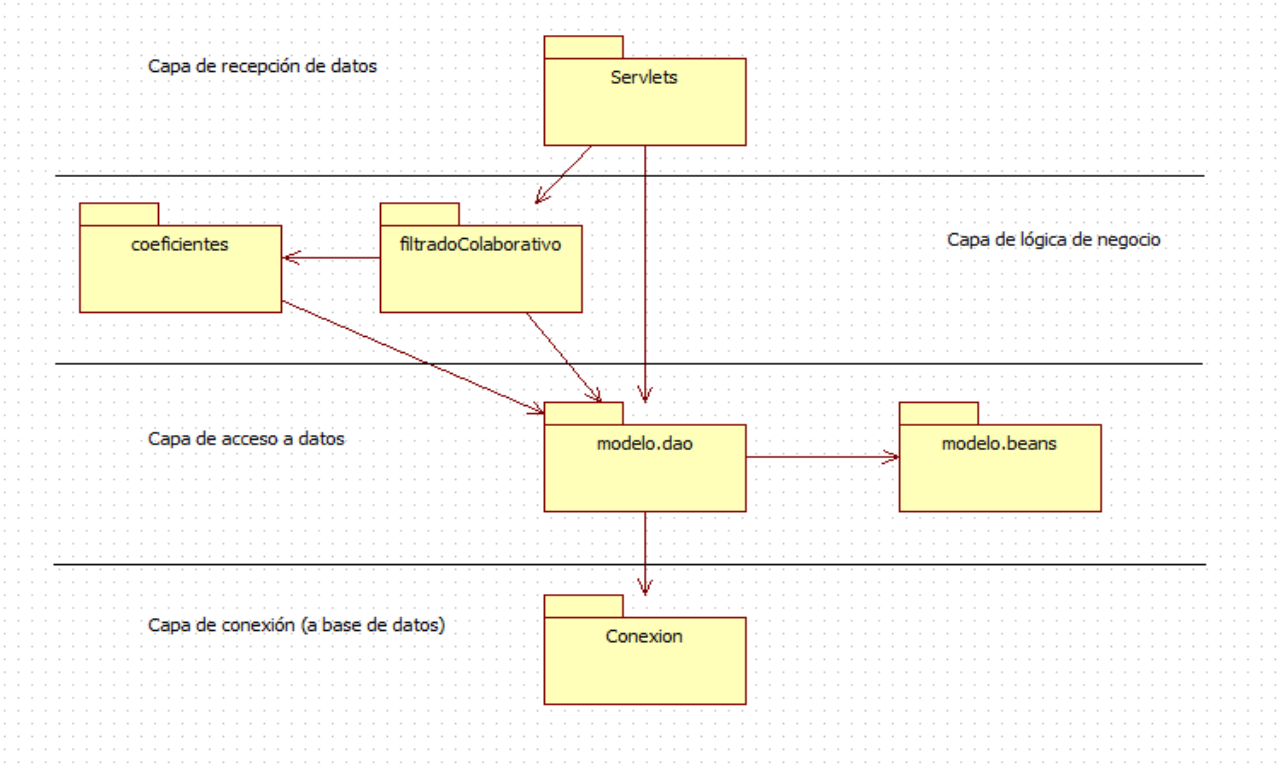


**Figura 1**. Diagrama de paquetes del aplicativo del servidor central.

## Capa de recepción de datos

Capa que implementa los siguientes paquetes.

***Servlets:*** Paquete que contiene todas las clases que implementan Servlets, encargadas de recibir y procesar la información que los móviles transmitan en formato JSON.

## Capa lógica de negocio

Capa que implementa los siguientes paquetes.

***filtradoColaborativo*** Paquete que contiene las clases encargadas de ejecutar los algoritmos de filtrado colaborativo en el sistema (3 en total).

***Coeficientes:*** Paquete que contiene clases que soportan las operaciones requeridas por las clases de *filtradoColaborativo.*

## Capa de acceso a datos

Capa que implementa los paquetes que contienen las clases sugeridas por el modelo de programación DAO (*Data Access Object*).

***modelo.beans:*** Paquete que contiene las clases entidades, que representan a las tablas de la base de datos central, en la aplicación.

***modelo.dao:*** Paquete que contiene las clases de control de las transacciones (insertar, actualizar y *merge*), por cada entidad.

## Capa de conexión

Capa que implementa el siguiente paquete:

***conexión:*** Paquete que contiene la clase encargada de establecer las conexiones con la base de datos central.

## A.1.2. Proceso Demonio

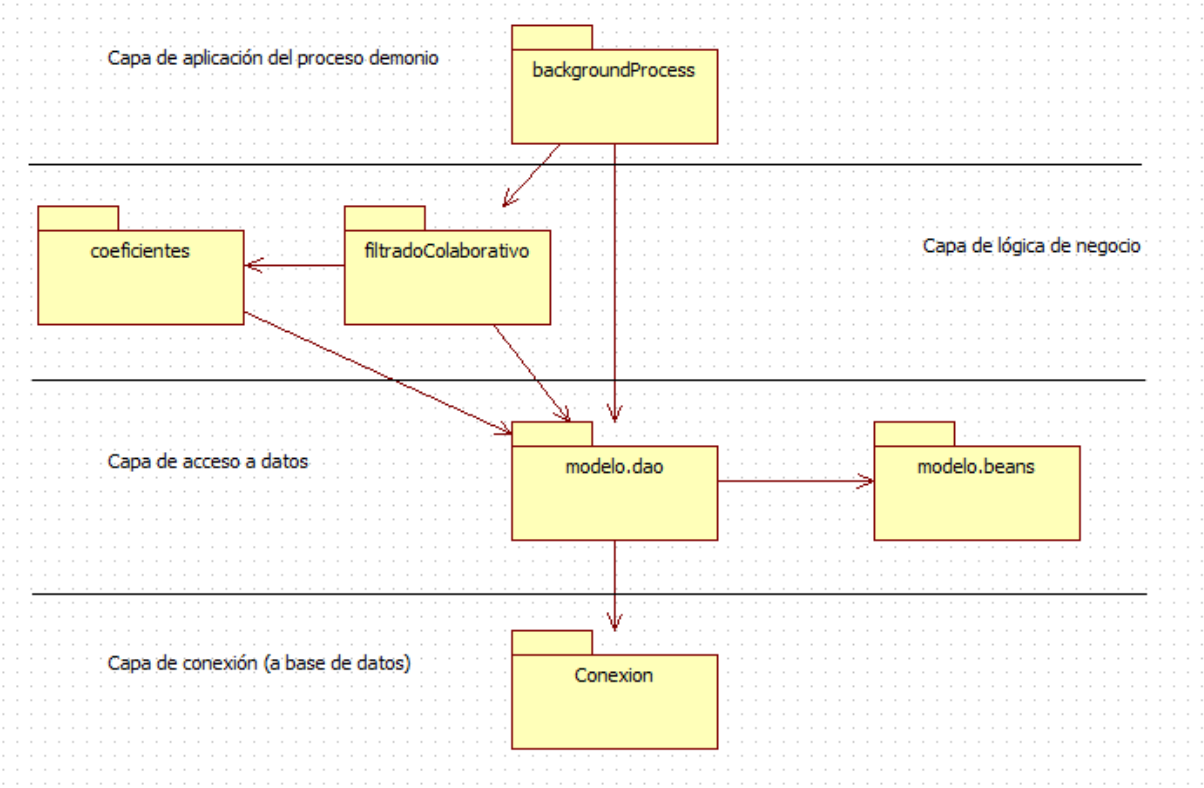


**Figura 2.** Diagrama de paquetes del aplicativo de proceso demonio.

## Capa de aplicación del proceso demonio

Capa que implementa los siguientes paquetes.

***backgroundProcess:*** Paquete que contiene todas las clases encargadas de ejecutar los procesos y algoritmos de apoyo para mejorar la eficiencia del sistema. Entre las funciones realizadas por estas clases se encuentran: actualización de la red social basada en contactos, cálculo de usuarios expertos, cálculo de calificaciones de aplicaciones a partir del consumo, *trigger* temporizado para la ejecución de las funciones mencionadas, etc.

## Capa lógica de negocio

Capa que implementa los siguientes paquetes.

***filtradoColaborativo*** Paquete que contiene las clases encargadas de los algoritmos de filtrado colaborativo en el sistema (3 en total).

***Coeficientes:*** Paquete que contiene clases que soportan las operaciones requeridas por las clases de *filtradoColaborativo.*

## Capa de acceso a datos

Capa que impleméntalo paquetes que contienen las clases sugeridas por el modelo de programación DAO (*Data Access Object*).

***modelo.beans:*** Paquete que contiene las clases entidades, que representan a las tablas de la base de datos central, en la aplicación.

***modelo.dao:*** Paquete que contiene las clases de control de las transacciones (insertar, actualizar y *merge*), por cada entidad.

## Capa de conexión

Capa que implementa el siguiente paquete:

***conexión:*** Paquete que contiene la clase encargada de establecer las conexiones con la base de datos central.

# A.1.3. Colector de Información

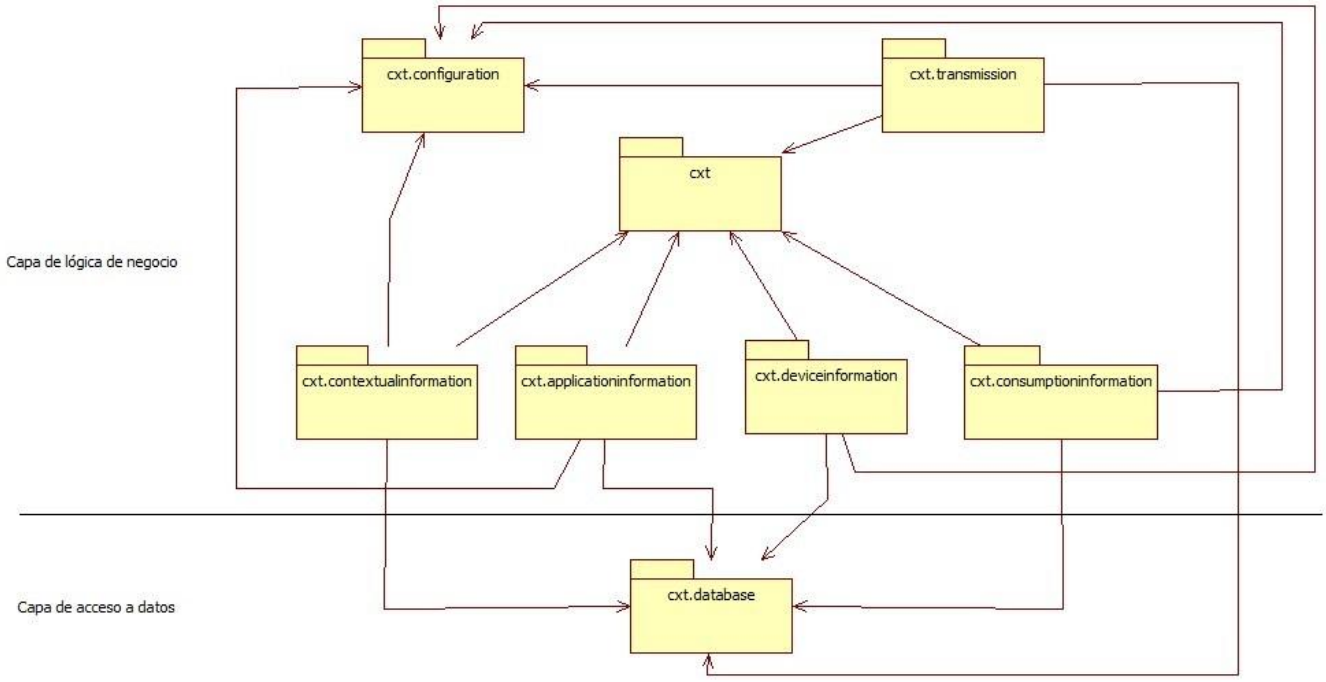


**Figura 3.** Diagrama de paquetes del colector de información.

## Capa de lógica de negocio

Capa que implementa lo siguientes paquetes

**cxt**: Paquete que contiene las clases que representan la situación contextual actual del usuario y las interfaces que contienen todas las variables explicitas necesarias en la aplicación.

**configuration**: Contiene las clases encargadas de configurar la recolección de información y la transmisión de esta, entre las que encontramos la clase para cargar la configuración de un archivo JSON y la clase que permite editar y leer las configuraciones.

**transmission**: Contiene las clases para la transmisión de información al servidor. Entre las funciones realizadas por estas clases se encuentran: enviar la información periódicamente según la configuración establecida y armar los mensajes JSON para enviar la información.

**Contextualinformation**: Contiene las clases para la recolección de la información contextual, entre las que se encuentra: información asociada al contexto físico (sensores, localización relativa, clima, etc.), información asociada al contexto cognitivo (estado de ánimo) e información asociada al contexto social (contactos, emisor-destinatario llamadas, emisor-destinatario de SMS).

**aplicationinformation:** Contiene las clases para la recolección de la información de las aplicaciones móviles, como nombre, paquete, instalación y desinstalación.

**deviceinformation:** Contiene las clases para la recolección de la información del dispositivo móvil, como versión del sistema operativo, fabricante, modelo, batería, pantalla.

**consumptioninformation:** Contiene las clases para la recolección de la información asociada al consumo de aplicaciones móviles.

## Capa de Acceso a Datos

Capa que implementa lo siguientes paquetes

**Database**: Contiene todas las clases para establecer la conexión a la base de datos y para realizar el control de transacciones (insertar, actualizar y eliminar)

### A.1.4. Cliente Móvil

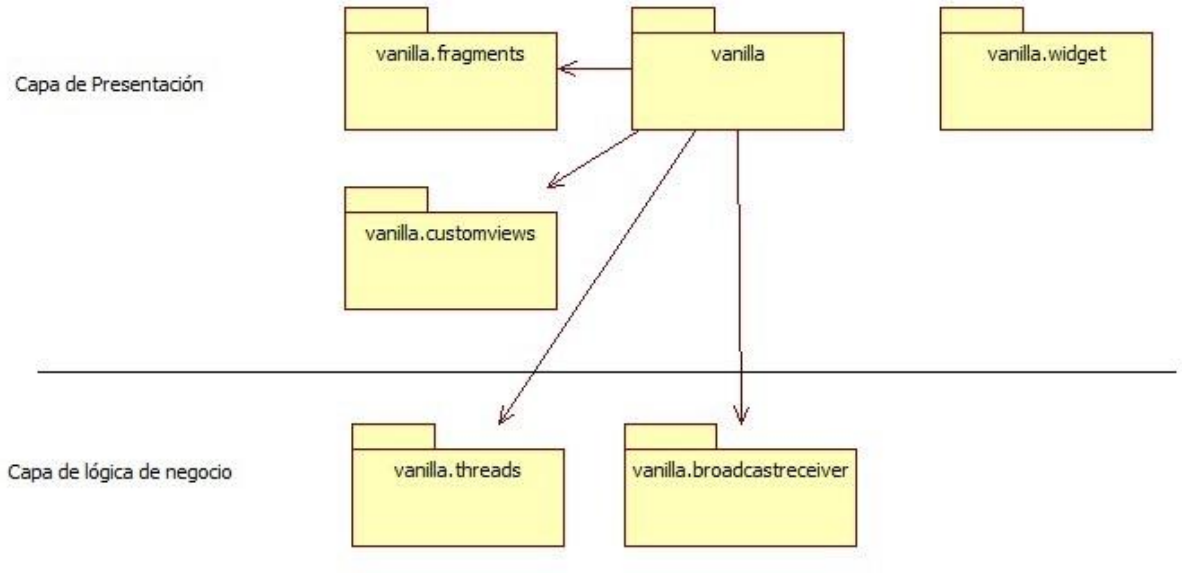


**Figura 4.** Diagrama de paquetes del cliente movil.

## Capa de Presentación

**vanilla**: Contiene todas las clases Activity para la interfaz grafica de usuario.

**fragments**: Contiene todas las clases Fragment para la interfaz grafica de usuario.

**custoviews**: Contiene todas las clases de vistas personalizadas para la interfaz grafica de usuario.

**Widget**: contiene las clases de configuración y administración de la aplicación widget de Vanilla.

## Capa de Lógica de negocio

**threads**: Contiene todas las clases de procesos en background, entre las funciones realizadas por estas clases se encuentra: procesamiento del JSON de recomendación y carga de las imágenes como iconos y banner de las aplicaciones.

**broadcastreceiver**: Contiene todas las Clases BroadcastReceiver para el manejo de eventos como el cambio de versión de aplicación y el reinicio del Smartphone.

## A.2. Diagrama de Clases

### A.2.1. Diagrama de clases aplicativo web

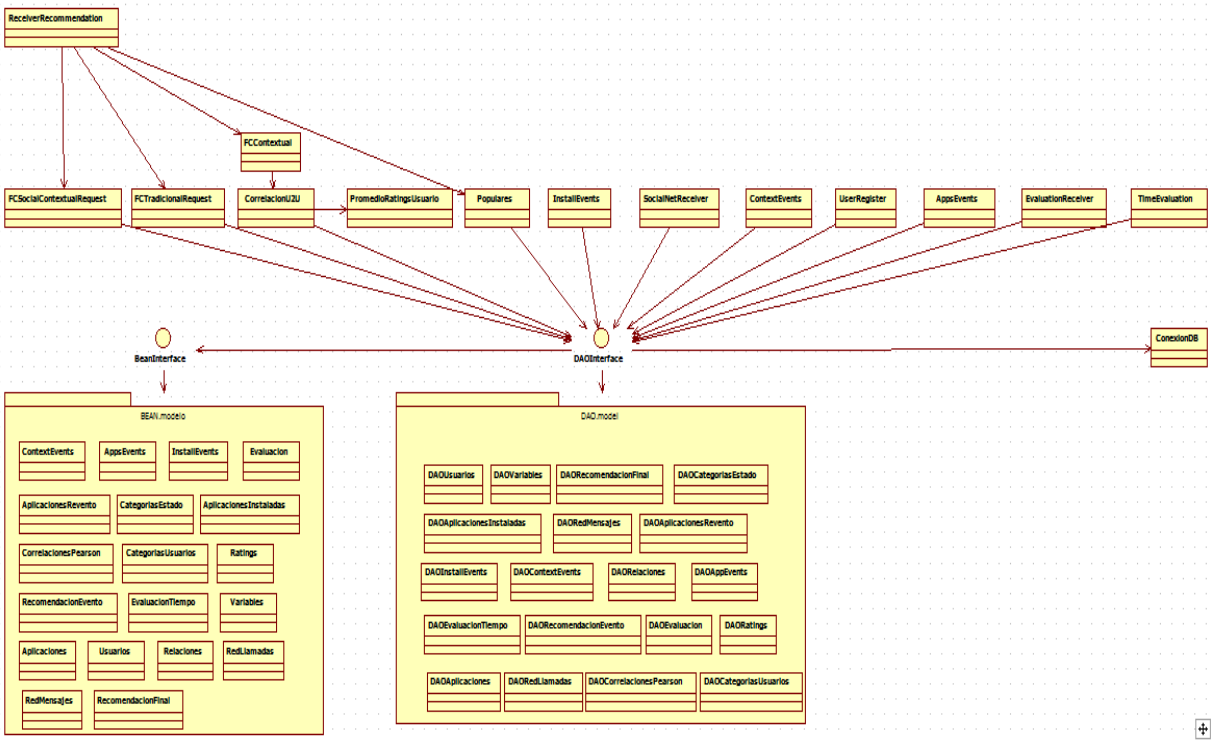


**Figura 5.** Diagrama de clases aplicativo web.

**Tabla 1.** Descripción de la clase *ReceiverRecommendation.*

| **Nombre** |
|---|
| *ReceiverRecommendation* |
| **Descripción** |
| Clase que extiende la clase HttpServlet. Esta se encarga de recibir y procesar las solicitudes de recomendación y la información correspondiente. |
| **Atributos** |
| *dateFinal*: Atributo tipo *SimpleDateFormat* cuya utilidad es establecer el formato correcto para almacenar registros tipo *TimeStamp*, en la base de datos. |
| **Métodos** |
| *doGet*(HttpServletRequest request, HttpServletResponse response)<br><br>*doPost*(HttpServletRequest request, HttpServletResponse response): Método que se encarga de recibir y procesar la solicitud de recomendación de cualquier |

| |
|---|
| usuario. En el mismo se invocan todos los métodos con el objetivo de generar la recomendación correspondiente (response).<br>*guardarRecomendacion*(String conjuntoApps, String movilUsuario): Método que almacena el evento que representa cada recomendación que se genere. |

**Tabla 2.** Descripción de la clase *InstallEvents.*

| **Nombre** |
|---|
| *InstallEvents* |
| **Descripción** |
| Clase que extiende la clase HttpServlet. Esta se encarga de recibir, procesar y almacenar la información correspondiente a los eventos de instalación de aplicaciones de cada usuario registrado. |
| **Atributos** |
| n/a |
| **Métodos** |
| *doGet*(HttpServletRequest request, HttpServletResponse response)<br><br>*doPost*(HttpServletRequest request, HttpServletResponse response): Método que se encarga de recibir, procesar y almacenar la información que se relaciona con los eventos de instalación de cada usuario. |

**Tabla 3.** Descripción de la clase *SocialNetReceiver.*

| **Nombre** |
|---|
| *SocialNetReceiver* |
| **Descripción** |
| Clase que extiende la clase HttpServlet. Esta se encarga de recibir, procesar y almacenar la información correspondiente a los contactos, eventos de llamadas y sms's de cada usuario. |
| **Atributos** |
| n/a |
| **Métodos** |
| *doGet*(HttpServletRequest request, HttpServletResponse response)<br><br>doPost(HttpServletRequest request, HttpServletResponse response): Método que se encarga de recibir, procesar y almacenar la información que se relaciona con los contactos, eventos de llamadas y sms's de cada usuario. |

**Tabla 4.** Descripción de la clase *ContextEvents*.

| **Nombre** |
|---|
| *ContextEvents* |
| **Descripción** |
| Clase que extiende la clase HttpServlet. Esta se encarga de recibir, procesar y almacenar la información correspondiente a los cambios de contexto de cada usuario. |
| **Atributos** |
| n/a |
| **Métodos** |
| *doGet*(HttpServletRequest request, HttpServletResponse response)<br><br>*doPost*(HttpServletRequest request, HttpServletResponse response): Método que se encarga de recibir, procesar y almacenar la información que se relaciona con los cambios de contexto de cada usuario. |

**Tabla 5.** Descripción de la clase *UserRegister*.

| **Nombre** |
|---|
| *UserRegister* |
| **Descripción** |
| Clase que extiende la clase HttpServlet. Esta se encarga de recibir, procesar y almacenar la información correspondiente a los nuevos usuarios que instalan la aplicación. |
| **Atributos** |
| n/a |
| **Métodos** |
| *doGet*(HttpServletRequest request, HttpServletResponse response)<br><br>*doPost*(HttpServletRequest request, HttpServletResponse response): Método que se encarga de recibir, procesar y almacenar la información que se relaciona con el registro de los nuevos usuarios. |

**Tabla 6.** Descripción de la clase *AppEvents*.

| **Nombre** |
|---|
| *AppEvents* |
| **Descripción** |
| Clase que extiende la clase HttpServlet. Esta se encarga de recibir, procesar y almacenar la información correspondiente a los eventos de consumo de aplicaciones de cada usuario registrado. |
| **Atributos** |
| n/a |

| **Métodos** |
| --- |
| *doGet*(HttpServletRequest request, HttpServletResponse response)<br><br>doPost(HttpServletRequest request, HttpServletResponse response): Método que se encarga de recibir, procesar y almacenar la información que se relaciona con los eventos de consumo de cada usuario. |

**Tabla 7.** Descripción de la clase *EvaluationReceiver*.

| **Nombre** |
| --- |
| *EvaluationReceiver* |
| **Descripción** |
| Clase que extiende la clase HttpServlet. Esta se encarga de recibir, procesar y almacenar la información correspondiente a la información de feedback respecto a las aplicaciones seleccionadas por recomendación. |
| **Atributos** |
| n/a |
| **Métodos** |
| *doGet*(HttpServletRequest request, HttpServletResponse response)<br><br>doPost(HttpServletRequest request, HttpServletResponse response): Método que se encarga de recibir, procesar y almacenar la información que se relaciona con la información de feedback respecto a las aplicaciones seleccionadas por recomendación. |

**Tabla 8.** Descripción de la clase *TimeEvaluation*.

| **Nombre** |
| --- |
| *TimeEvaluation* |
| **Descripción** |
| Clase que extiende la clase HttpServlet. Esta se encarga de recibir, procesar y almacenar la información correspondiente la información de feedback respecto a al tiempo que cada usuario emplea en la observación de las páginas de recomendación. |
| **Atributos** |
| n/a |
| **Métodos** |
| *doGet*(HttpServletRequest request, HttpServletResponse response)<br><br>doPost(HttpServletRequest request, HttpServletResponse response): Método que se encarga de recibir, procesar y almacenar la información de feedback respecto a al tiempo que cada usuario emplea en la observación de las páginas de recomendación. |

**Tabla 9.** Descripción de la clase *FCSocialContextualRequest*.

| **Nombre** |
|---|
| *FCSocialContextualRequest* |
| **Descripción** |
| Clase que retorna el grupo de aplicaciones que se recomiendan usando la técnica social-contextual. |
| **Atributos** |
| Atributos tipo *String: movilUsuario, contextoSQL, tramaFinal, notInSocial, tramaInsertPruebas*; que representan el móvil del usuario que solicita la recomendación, el contexto actual del usuario al solicitar la recomendación en formato SQL, la recomendación de aplicaciones dada en el formato JSON en el presente algoritmo, las aplicaciones recomendadas que no se deben tener en cuenta en los siguientes algoritmos de recomendación, cadena procesada en formato SQL para almacenar las aplicaciones recomendadas en la base de datos, respectivamente.<br>Atributo *conn* tipo ConexionDB que corresponde a la instancia de la clase de conexión a la base de datos.<br>Atributo *contexto* tipo *Map* que representa el contexto actual del usuario en una colección (con el objetivo de reducir la complejidad del código fuente).<br>Atributo *sumatoriaPesos* tipo *double*, que representa la sumatoria de los pesos asignados a las variables contextuales por el usuario.<br>Atributo *flag* tipo boolean, que es una variable de control para mejorar la eficiencia del procesamiento. |
| **Métodos** |
| *obtenerRecomendacion():* Método que extrae la recomendación de este tipo (social-contextual), del repositorio, teniendo en cuenta que la técnica de recomendación social-contextual, se calcula en un proceso demonio cada 6 horas (no al momento de la solicitud). |

**Tabla 10.** Descripción de la clase *FCTradicionalRequest*.

| **Nombre** |
|---|
| *FCTradicionalRequest* |
| **Descripción** |
| Clase que retorna el grupo de aplicaciones que se recomiendan usando la técnica *tradicional*. |
| **Atributos** |
| Atributos tipo *String: movilUsuario, tramaFinal, notInTradicional, tramaInsertPruebas*; que representan el móvil del usuario que solicita la recomendación, la recomendación de aplicaciones dada en el formato JSON en el presente algoritmo, las aplicaciones recomendadas que no se deben tener en cuenta en los siguientes algoritmos de recomendación, cadena procesada en formato SQL para almacenar las aplicaciones recomendadas en la base de datos, respectivamente.<br>Atributo *conn* tipo ConexionDB que corresponde a la instancia de la clase de conexión a la base de datos. |

| |
|---|
| **Métodos** |
| *obtenerRecomendacion():* Método que extrae la recomendación de este tipo (*tradicional*), del repositorio, teniendo en cuenta que la técnica de recomendación tradicional (junto con la *social-contextual*), se calcula en un proceso demonio cada 6 horas (no al momento de la solicitud). |

**Tabla 11.** Descripción de la clase *FCContextual*.

| **Nombre** |
|---|
| *FCContextual* |
| **Descripción** |
| Clase que encargada de calcular y proveer las recomendaciones utilizando la técnica contextual. |
| **Atributos** |
| Atributos tipo *String: movilUsuario, vecindadContextualSQL, aplicacionesSQL, contextoSQL, tramaFinal, notInContextual, tramaInsertPruebas*; que representan el móvil del usuario que solicita la recomendación, la vecindad del usuario según su contexto dada en un formato SQL, las aplicaciones candidatas a ser recomendadas en el contexto actual del usuario en un formato SQL, el contexto actual del usuario al solicitar la recomendación, la recomendación de aplicaciones dada en el formato JSON en el presente algoritmo, las aplicaciones recomendadas que no se deben tener en cuenta en los siguientes algoritmos de recomendación, cadena procesada en formato SQL para almacenar las aplicaciones recomendadas en la base de datos, respectivamente.<br>Atributo *conn* tipo ConexionDB que corresponde a la instancia de la clase de conexión a la base de datos.<br>Atributo *vecindadContextual* tipo *List* que representa la vecindad del usuario según su contexto en una lista (con el objetivo de reducir la complejidad del código fuente).<br>Atributo *ratingsOrdenados* tipo *List* que representa las aplicaciones que se van a recomendar utilizando el presente algoritmo, de forma ordenada (de mayor a menor rating). |
| **Métodos** |
| *algoritmoFCContextual()*: Método que implementa el algoritmo de FC para la predicción de forma contextual (FC contextual).<br>*armarTramaFinal():* Método que construye en formato JSON, la recomendación calculada a través de la técnica contextual.<br>*organizarApps(String paquete, Double rating)*: organiza las aplicaciones a las cuales se les aplica el FC contextual, de acuerdo a su respectivo rating (de mayor a menor).<br>*descubrirAplicacionesMasConsumidasPorContexto():* Extrae las aplicaciones consumidas con mayor score (calificación basada en consumo) en el contexto dado por el usuario,<br>*seleccionarVecindad():* Método que implementa el algoritmo para calcular la vecindad contextual del usuario. |

**Tabla 12.** Descripción de la clase *CorrelacionU2U*.

| **Nombre** |
|---|
| *CorrelacionU2U* |
| **Descripción** |
| Clase que calcula la similitud entre dos usuarios. |
| **Atributos** |
| Atributos *promRatingsUsuario, promRatingsVecino* tipo *PromedioRatingsUsuario*, que representan el objeto que contiene el promedio total de calificaciones del usuario y el objeto que contiene el promedio total de calificaciones de un vecino, respectivamente.<br>Atributos *coeficienteCorrelacion double* que representa el factor de correlación o similitud entre dos usuarios.<br>Atributos *movilUsuario, contextoSQL* tipo *String* que representan el móvil del usuario que solicita la recomendación, el contexto actual del usuario al solicitar la recomendación en formato SQL.<br>Atributo *contexto* tipo *Map* que representa el contexto actual del usuario en una colección (con el objetivo de reducir la complejidad del código fuente). |
| **Métodos** |
| *determinarCorrelacion ():* Método que calcula el factor de similitud entre el usuario y un vecino utilizando la correlación de Pearson tradicional.<br>*determinarCorrelacionContextual():*Método que calcula el factor de similitud entre el usuario y un vecino utilizando la correlación de Pearson contextual. |

**Tabla 13.** Descripción de la clase *PromedioRatingsUsuarios*.

| **Nombre** |
|---|
| *PromedioRatingsUsuarios* |
| **Descripción** |
| Clase que calcula el promedio de scores de cualquier usuario. |
| **Atributos** |
| Atributos *movilUsuario, contextoSQL* tipo *String* que representan el móvil del usuario que solicita la recomendación, el contexto actual del usuario al solicitar la recomendación en formato SQL.<br>Atributos *contexto, items* tipo *Map* que representan el contexto actual del usuario en una colección (con el objetivo de reducir la complejidad del código fuente) y los scores de los ítems consumidos por el usuario, respectivamente.<br>Atributos *sumatoriaFactores, promedio* tipo *double*, que representan la sumatoria de los pesos asignados a las variables contextuales por el usuario y al promedio de score del usuario (sea contextual o general).<br>Atributo *conn* tipo ConexionDB que corresponde a la instancia de la clase de conexión a la base de datos. |
| **Métodos** |
| *calcularPromedio(String movilUsuario):* Calcula el promedio del score del usuario de acuerdo a su consumo general.<br>*consultarAppCalificadas(String movilUsuario)*: Extrae las aplicaciones consumidas por el usuario, junto con sus respectivos scores (generales).<br>*calcularPromedioContextual(String movilUsuario):* Calcula el promedio del score |

| |
|---|
| del usuario de acuerdo a su consumo dado por el contexto definido.<br>*consultarAplicacionesConsumidasContexto(String movilUsuario):* Extrae las aplicaciones consumidas por el usuario, y calcula sus respectivos scores (contextuales). |

**Tabla 14.** Descripción de la clase *DAOInterface*.

| **Nombre** |
|---|
| *DAOInterface* |
| **Descripción** |
| Interfaz que representa todos los controladores de las entidades definidas de acuerdo a las tablas presentes en la base de datos (modelo de programación DAO). |
| **Atributos** |
| n/a |
| **Métodos** |
| *insert(ArrayList<Bean> bean)):* Inserta nuevos registros de la entidad/tabla correspondiente en la base de datos.<br>*update(ArrayList<Bean> bean)*: Inserta registros existentes de la entidad/tabla correspondiente en la base de datos.<br>*merge(ArrayList<Bean> bean):* Inserta nuevos registros en el caso de que estos no existan en la base de datos aún, o los actualiza en el caso contrario (se implementa este método debido a que el motor de base de datos MySQL, no soporta esta operación, a diferencia de otros como ORACLE) |

**Tabla 15.** Descripción de la clase *BeanInterface*.

| **Nombre** |
|---|
| *BeanInterface* |
| **Descripción** |
| Interfaz que representa todas las entidades definidas de acuerdo a las tablas presentes en la base de datos (modelo de programación DAO). |
| **Atributos** |
| Por cada entidad que extienda de Bean, se definen los atributos correspondientes a los campos en las tablas que se desee representar, de la base de datos |
| **Métodos** |
| "*Getters*" and "*Setters*" de cada atributo según sea la entidad que se represente. |

**Tabla 16.** Descripción de la clase *ConexionDB*.

| **Nombre** |
|---|
| *ConexionDB* |
| **Descripción** |
| Clase que se encarga de establecer la conexión con la base de datos. |
| **Atributos** |
| Atributos *urlDB, user, password, driver* tipo *String* que representan los datos de validación para establecer una conexión exitosa con la base de datos.<br>Atributo *resultado* tipo *ResultSet* que representa el resultado que retorna la base de datos por cada consulta (API *JDBC*).<br>Atributo *connection* tipo *Connection* que representa la conexión que se mantiene en sesión por cada transacción (API *JDBC*). |
| **Métodos** |
| *iniciarConexion():* Método encargado de instanciar, por ende, iniciar una nueva conexión a la base de datos, antes de cualquier transacción.<br>*cerrarCx():* Método que cierra la conexión después de cada transacción.<br>*ejecutarSQL(String sql):* Método que ejecuta las operaciones indicadas en la base de datos (insert, update), sin necesidad de un *ResultSet* de retorno.<br>*consutar(String sql):* Método que ejecuta las operaciones indicadas en la base de datos (select), y que retorna un *ResultSet*.<br>*actualizar(String sql):* Método que ejecuta las operaciones indicadas en la base de datos (update), sin necesidad de un *ResultSet* de retorno.<br>*siguiente():* Método que apunta a la siguiente posición del *ResultSet* en sesión, retornado por la base de datos. |

## A.2.2. Diagrama de clases proceso demonio

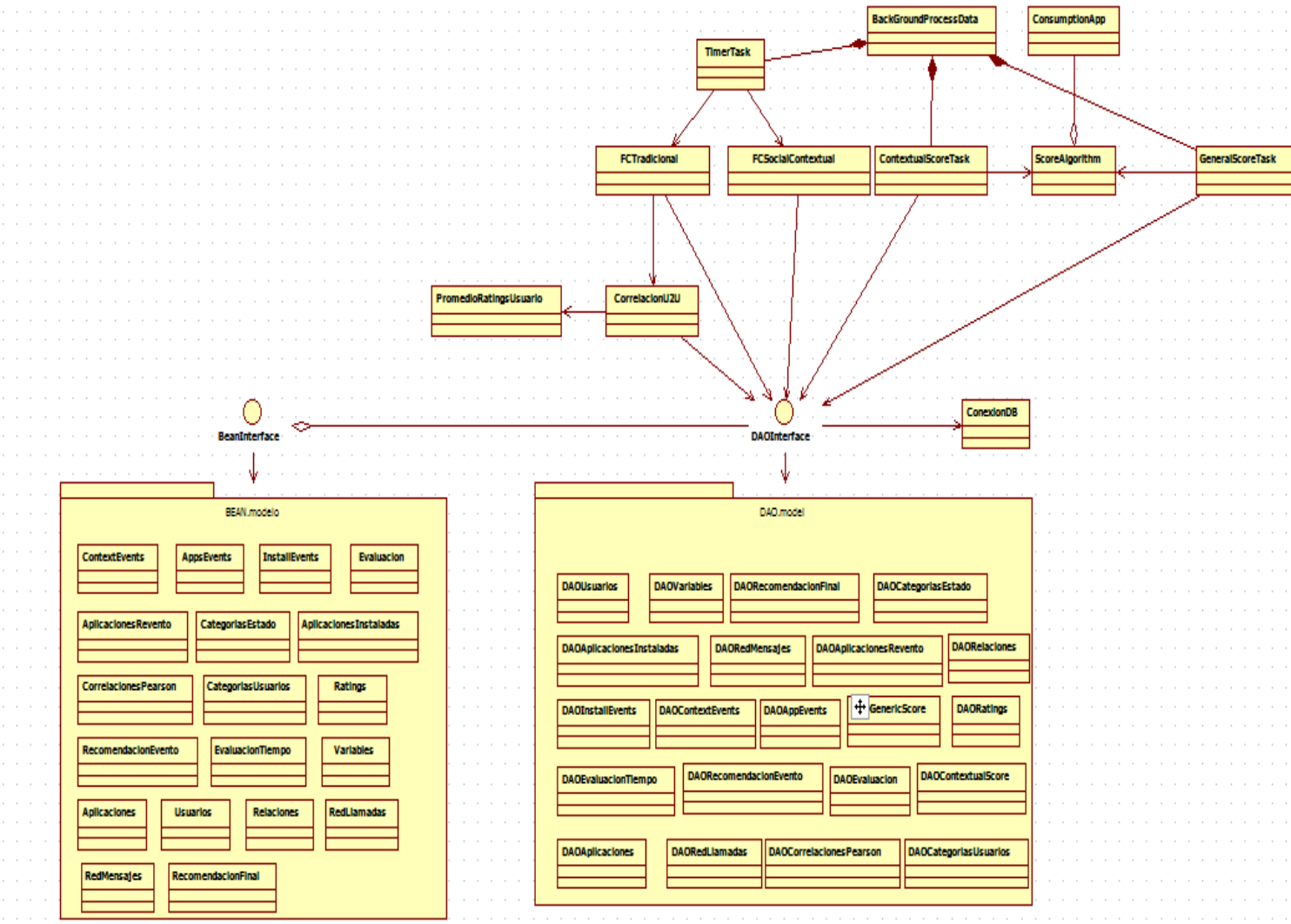


**Figura 6.** Diagrama de clases del proceso demonio implementado.

**Tabla 17.** Descripción de la clase *BackgroundProcessData*.

| **Nombre** |
|---|
| *BackgroundProcessData* |
| **Descripción** |
| Clase que se encarga de iniciar la ejecución del proceso demonio o en backGround, cada cierto tiempo de forma iterativa. |
| **Atributos** |
| n/a |
| **Métodos** |
| *main(String[] args)*: Método principal del ciclo de vida de una aplicación Java. Se |

| |
|---|
| encarga del manejo del proceso demonio controlando la ejecución de los métodos respectivos temporalmente. |

**Tabla 18.** Descripción de la clase *TimerTasks*.

| **Nombre** |
|---|
| *TimerTasks* |
| **Descripción** |
| Clase que extiende de *TimerTask,* en donde se encuentra la lógica principal. En esta clase se ejecutan todos los métodos que componen el proceso demonio (incluyendo las técnicas de recomendación *Tradicional* y *Social-contextual*). |
| **Atributos** |
| Atributo *conn* tipo ConexionDB que corresponde a la instancia de la clase de conexión a la base de datos.<br>Atributo *usuariosSistema* tipo *List* que representa todos los usuarios registrados en el sistema (con el objetivo de reducir la complejidad del código fuente).<br>Atributo *mensajesLlamadasUsuarios* tipo *Map* que representa todo el grafo social en el sistema (con el objetivo de reducir la complejidad del código fuente). |
| **Métodos** |
| *run():* Método que implementa toda la lógica de negocio del proceso demonio (de acuerdo a la documentación de *java.util.Timer*). Invoca los otros métodos correspondientes a las operaciones de actualización, cálculos e inferencia de nueva información en el sistema de recomendación.<br>*ejecutarAlgoritmosFC():* Método que invoca los métodos correspondientes a la implementación de las técnicas de recomendación *Tradicional* y *Social-Contextual.*<br>*actualizarDatosVecindad():* Método que calcula el número de aplicaciones que cada uno de los usuarios, consume en común con otro.<br>*actualizarDatosUsuariosExpertos():* Método que implementa la lógica para el cálculo de los usuarios expertos por categoría, en la red social del sistema.<br>*obtenerLlamadasMsjesBt(String numeroUsuario):* Método que extrae el número de llamdas y sms's entre los usuarios del sistema, de la base de datos.<br>*actualizarPesosRelaciones():*Método que implementa la lógica para actualizar la estructura del grafo o red social basada en contactos, llamadas y sms's en el sistema. |

**Tabla 19.** Descripción de la clase *FCTradicional*.

| **Nombre** |
|---|
| *FCTradicional* |
| **Descripción** |
| Clase que encargada de calcular y proveer las recomendaciones utilizando la técnica *Tradicional*. |
| **Atributos** |
| Atributos tipo *String: movilUsuario, vecindadSQL, aplicacionesMasConsumidasSQL*; que representan el móvil del usuario que |

| |
|---|
| solicita la recomendación, la vecindad tradicional del usuario dada en un formato SQL, las aplicaciones candidatas con mayor score (por parte de la vecindad tradicional) a ser recomendadas en un formato SQL, respectivamente.<br>Atributo *conn* tipo ConexionDB que corresponde a la instancia de la clase de conexión a la base de datos.<br>Atributo *vecindad* tipo *List* que representa la vecindad tradicional del usuario (con el objetivo de reducir la complejidad del código fuente).<br>Atributo *aplicacionesMasConsumidas* tipo *List* las aplicaciones candidatas con mayor score (por parte de la vecindad tradicional) a ser recomendadas (con el objetivo de reducir la complejidad del código fuente).<br>Atributo *aplicacionesRecomendadas* tipo *Map* que representa las aplicaciones con sus respectivos ratings, predichos a través de la técnica de recomendación *Tradicional.*<br>Atributos *flagVecindad, flagAppsDescubiertas* tipo *boolean* que son variables de control para mejorar la eficiencia del procesamiento. |
| **Métodos** |
| *algoritmoFCTradicional():* Método que implementa el algoritmo de FC *Tradicional.*<br>*mergeRecomendacionFinal():* Método que implementa la operación *merge,* a los registros de las aplicaciones que se van a recomendar (y sus ratings) en la base de datos.<br>*seleccionarVecindad():* Método que implementa los algoritmos para el cálculo de la vecindad tradicional de los usuarios.<br>*descubrirAplicaciones():* Método que extrae las aplicaciones candidatas a ser recomendadas, con mayor score. |

**Tabla 20.** Descripción de la clase *FCSocialContextual*.

| **Nombre** |
|---|
| *FCSocialContextual* |
| **Descripción** |
| Clase que encargada de calcular y proveer las recomendaciones utilizando la técnica *Social-Contextual*. |
| **Atributos** |
| Atributos tipo *String: movilUsuario, movilesContactos, contactosExpertos, categoriasVariablesSQL*; que representan el móvil del usuario que solicita la recomendación, la vecindad basada en la red social del usuario, el conjunto de usuarios expertos por categoría, respectivamente.<br>Atributo *conn* tipo ConexionDB que corresponde a la instancia de la clase de conexión a la base de datos.<br>Atributo *aplicacionesDescurbiertas* tipo *List* que representa las aplicaciones candidatas a recomendar, sugeridas por los usuarios expertos.<br>Atributos *categoriasVariables, usuariosExpertos, redSocial, aplicacionesRecomendadas* tipo *Map* que representan las categorías predominantes por cada estado contextual que experimente cada usuario, el conjunto de contactos expertos indexados por categorías, el grafo social o red social del sistema, las aplicaciones recomendadas con sus ratings respectivos y predichos por la técnica *Social-Contextual*; respectivamente (con el objetivo de |

| |
|---|
| reducir la complejidad del código fuente).<br>Atributos *flagContactosExpertos, flagAppsDescubiertas, flagEstado* tipo *boolean* que son variables de control para mejorar la eficiencia del procesamiento. |
| **Métodos** |
| *ejeutarFCSocialContextual():*Método que implementa el algoritmo de FC basado en la técnica *Social-Contextual.*<br>*mergeRecomendacionFinal():* Método que implementa la operación *merge,* a los registros de las aplicaciones que se van a recomendar (y sus ratings) en la base de datos.<br>*descubrirAplicaciones():* Método que extrae las aplicaciones candidatas a ser recomendadas, sugeridas por los usuarios expertos en las categorías dadas.<br>*usuariosExpertos():* Método que extrae los usuarios expertos correspondientes a las categorías dadas, de la base de datos.<br>*categoriasUsuarioPorEstado():* Método que calcula y extrae las categorías predominantes por cada estado que cada usuario experimente.<br>*mergeCategoriasUsuarioPorEstado():* Método que almacena en la base de datos, la información descrita en el anterior método (*categoriasUsuarioPorEstado()*). |

## A.2.3. Diagrama de colector de información

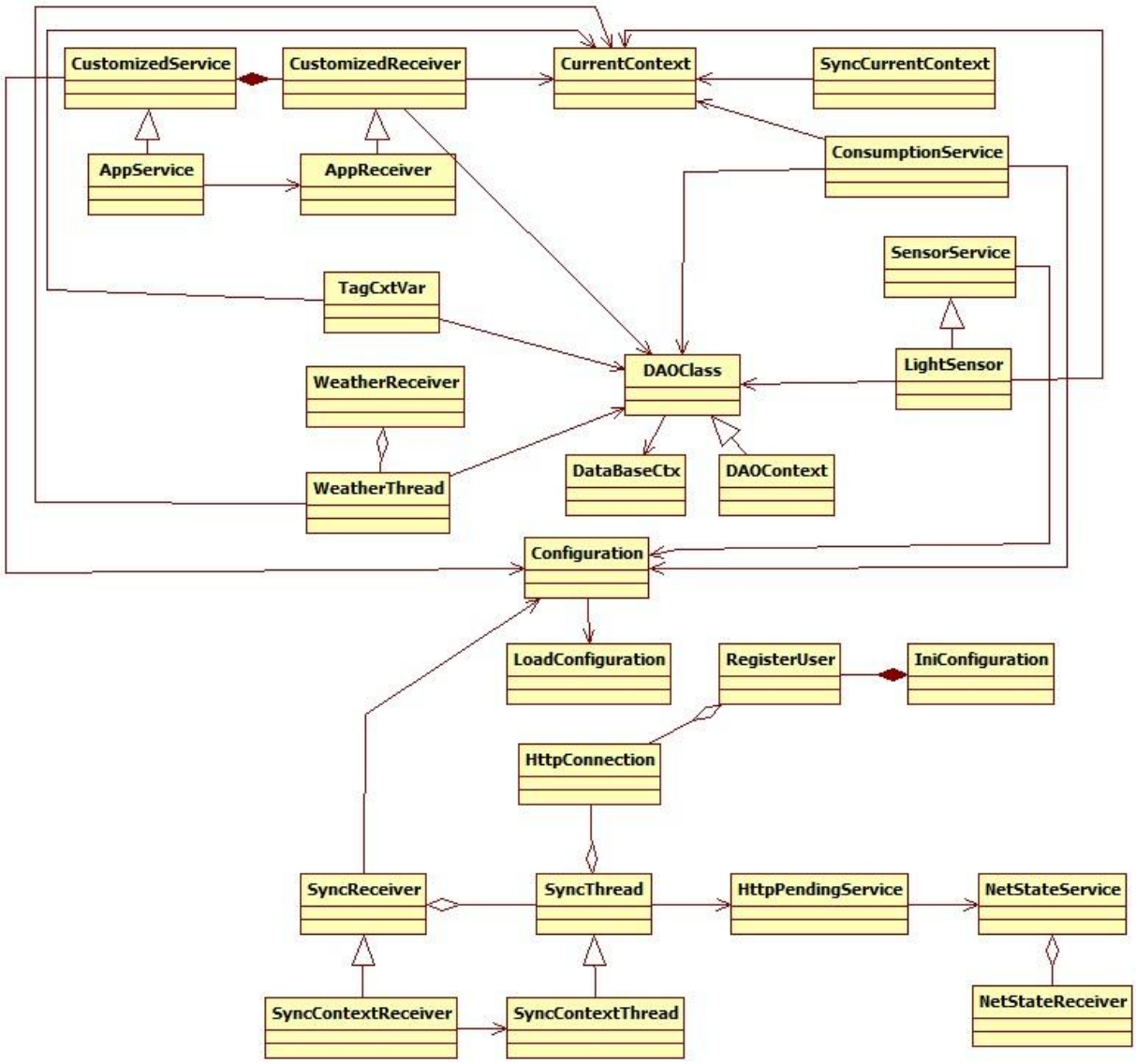


**Figura 7.** Diagrama de clases del colector de información.

**Tabla 21.** Descripción de la clase CurrentContext.

| **Nombre** |
|---|
| *CurrentContext* |
| **Descripción** |
| Clase que almacena los estados contextuales de cada dimensión contextual |
| **Atributos** |
| *CurrentContextList*: Atributo tipo *List<String>* que almacena las dimensiones contextuales activas . |
| **Métodos** |

| |
|---|
| *getCurrtenContext():* retorna la lista *CurrentContextList*<br>*getJsonDecriptor(Context context):* retorna el contexto actual en un formato JSON |

**Tabla 22.** Descripción de la clase CustomizedReceiver.

| **Nombre** |
|---|
| *CustomizedReceiver* |
| **Descripción** |
| Clase abstracta que hereda de BroadcastReceiver, esta recibe y procesa eventos y almacenados en la base de datos y en la clase *CurrentContext* |
| **Atributos** |
| No contiene atributos |
| **Métodos** |
| *onReceive(Context,Intent):* Método que se invoca al recibir un mensaje, el cual es procesado y almacenado<br>*getActions():* Método no implementado que retorna un vector de String para identificar la acción a procesar.<br>*processedInformation(String msg):* Método no implementado para procesar la información<br>*getDaoClass()*: Método no implementado que retorna la clase DAO correspondiente. |

**Tabla 23.** Descripción de la clase CustomizedService.

| **Nombre** |
|---|
| *CustomizedService* |
| **Descripción** |
| Clase abstracta que hereda de Service, la cual implementa el ciclo de vida del servicio y crea un Thread para el manejo del procesamiento |
| **Atributos** |
| *mServiceLooper:* Looper para asiganar un Thread alternativo<br>*mServiceHandler:* Handler que maneja los procedimientos dentro del hilo |
| **Métodos** |
| *getReceiver():* Metodo no implementado que retorna el BroadcastReceiver correspondiente.<br>*onCreate():* Metodo que se invoca cuando se crea el servicio, en este se inicializan los atributos<br>*onDestroy():* Metodo que se invoca cuando se destruye el servicio, en este se liberan los recursos y atributos. |

**Tabla 24.** Descripción de la clase AppReceiver.

| **Nombre** |
|---|
| *AppReceiver* |
| **Descripción** |
| Hereda de la clase *CustomizedReceive,* implementa los métodos no implementados en la clase padre para recibir los eventos de instalación y desinstalación de las aplicaciones móviles en el Smartphone, Igual que esta clase también se hace la descripción para las clases de BatteryReceiver, ScreenReceiver, OrientationConfigReceiver, CallReceiver, SMSReceiver. |
| **Atributos** |
| *ACTION:* Identifica la acción a recibir en el método onReceive().<br>daoApp: Objeto de la clase DAOApp |
| **Métodos** |
| *getActions():* Método que retorna un vector de String para identificar la acción a procesar.<br>*processedInformation(String msg):* Método para procesar la información<br>*getDaoClass()*: Método que retorna la clase DAO correspondiente. |

**Tabla 25.** Descripción de la clase AppService.

| **Nombre** |
|---|
| *AppService* |
| **Descripción** |
| Hereda de la clase CustomizedService, implementa los métodos no implementados de la clase padre para establecer el servicio sobre el cual correrá el BroadcastReceiver correspondiente. Igual que esta clase también se hace la descripción para las clases BatteryService, ScreenService, OrientationConfigService, CallService, SMSService. |
| **Atributos** |
| *appReceiver:* Objeto de la clase AppReceiver. |
| **Métodos** |
| *getReceiver():* Metodo que retorna el BroadcastReceiver correspondiente. |

**Tabla 26.** Descripción de la clase *TagCxtVar.*

| **Nombre** |
|---|
| *TagCxtVar* |
| **Descripción** |
| Clase para el ingreso de información que es expresada por etiquetas como la localización relativa, estado de ánimo y clima. |
| **Atributos** |
| *LOCATION:* atributo estático que contiene la etiqueta para localización relativa |

| |
|---|
| *MOOD:* atributo estático que contiene la etiqueta para el estado de animo<br>*WEATHER:* atributo estático que contiene la etiqueta para el clima |
| **Métodos** |
| *inserValue(String tag, String value):* método para el ingreso de información en la base de datos con la etiqueta y el valor correspondiente. |

**Tabla 27.** Descripción de la clase *WheaterReceiver.*

| **Nombre** |
|---|
| *WheaterReceiver* |
| **Descripción** |
| Clase hereda de BroadcastReceiver, en la cual se recibe el evento de consultar clima y lanza el Thread para realizar la consulta correspondiente. |
| **Atributos** |
| *ACTION:* String estático que identifica el evento a procesar. |
| **Métodos** |
| *onReceive(Context,Intent):* Método que se invoca al recibir un mensaje o evento, el cual esta determinado por la acción declarada anteriormente, en este método se lanza el Thread del servicio web. |

**Tabla 28.** Descripción de la clase *WeatherThread.*

| **Nombre** |
|---|
| *WeatherThread* |
| **Descripción** |
| Clase que hereda de Thread, en la cual se consulta la posición actual del usuario y el clima mediante la clase *HttpConnecion* |
| **Atributos** |
| *woeid:* Atributo de tipo *String* que almacena la localización del usuario en formato Woeid.<br>*context:* Atributo de tipo *Context* que contiene el contexto del BroadcastReceiver.<br>*humidity:* Atributo de tipo *float* que almacena la humedad.<br>*temperature:* Atributo de tipo *float* que almacena la temperatura ambiente.<br>*wheather:* Atributo de tipo *String* que almacena el clima actual.<br>*pressure:* Atributo de tipo *float* que almacena la presión atmosférica. |
| **Métodos** |
| *WeatherThread(Context):* Constructor del la Clase.<br>*getGPS():* Método que retorna la localización del usuario en latitud y longitud.<br>*run():* Metodo que ejecuta el procesamiento principal del hilo, realiza la consulta del servicio web, lo almacena en la base de datos y lo coloca en la clase *CurrentContext.* |

**Tabla 29.** Descripción de la clase *ConsumptionService.*

| **Nombre** |
|---|
| *ConsumptionService* |
| **Descripción** |
| Clase que hereda de *Service*, en la cual se implementan los procedimientos para la captura de consumo de las aplicaciones móviles en el Smartphone. |
| **Atributos** |
| *mServiceLooper:* Looper para asiganar un Thread alternativo.<br>*mServiceHandler:* Handler que maneja los procedimientos dentro del hilo.<br>*appAct:* Atributo de tipo String que almacena el nombre del paquete de la aplicación que se encuentra corriendo.<br>*last:* Atributo de tipo String que almacena el nombre del paquete de la anterior aplicación que se encontraba corriendo.<br>homePackage: Atributo de tipo String que almacena el nombre del paquete del launcher del Smartphone. |
| **Métodos** |
| *getHomePackage:* Método que obtiene el nombre del paquete de la aplicación Launcher y lo almacena en el atributo *homePackage*.<br>*onCreate():* Metodo que se invoca cuando se crea el servicio, en este se inicializan los atributos.<br>*onDestroy():* Metodo que se invoca cuando se destruye el servicio, en este se liberan los recursos y atributos. |

**Tabla 30.** Descripción de la clase *SensorService.*

| **Nombre** |
|---|
| *SensorService* |
| **Descripción** |
| Clase abstracta que hereda de la clase *Service,* esta clase realiza el servicio en background para la lectura y procesamiento de los sensores del Smartphone |
| **Atributos** |
| *time:* Atributo tipo long que es usado para fijar el periodo de muestreo del sensor.<br>*valAnt:* vector de tipo float que es usado para almacenar el valor anterior del sensor.<br>*tagAnt:* Atributo de tipo String, que es usado para almacenar la etiqueta del valor anterior del sensor.<br>*sensorManager:* Objeto de SensorManager.<br>*sensorEventListener:* Objeto de SensorEventListener, en el cual se reciben los cambios en el sensor.<br>*mServiceLooper:* Looper para asiganar un Thread alternativo.<br>*mServiceHandler:* Handler que maneja los procedimientos dentro del hilo. |
| **Métodos** |
| *comparisonTag(String tagAct):* Método que compara la etiqueta anterior con la actual y retorna un booleano , true si son diferentes o false en caso contrario. |

| |
|---|
| *comparisonValue(float[] valueAct):*Método que compara el valor actual del sensor con el anterior y retorna un booleano , true si son diferentes o false en caso contrario.<br>*getSensorManager():* Método que retorna el objeto sensorManager inicializado.<br>*getName():* Método no implementado que retorna el nombre de la dimension contextual.<br>*getSensorType():* Método no implementado que retorna el tipo de sensor.<br>*getTag():* Método no implementado que retorna la etiqueta para un valor del sensor dado.<br>*getTime():* Método no implementado que retorna el tiempo de muestreo. |

**Tabla 31.** Descripción de la clase *LightSensor.*

| **Nombre** |
|---|
| *LightSensor* |
| **Descripción** |
| Clase que hereda de SensorService e implementa todos los métodos no implementados. Igual que esta clase también se hace la descripción para las clases de GravitySensor, GyroscopeSensor, HumiditySensor, AccelerometerSensor, etc. |
| **Atributos** |
| *NAME:* Atributo estático de tipo String que retorna el nombre de la dimensión contextual |
| **Métodos** |
| *getName():* Método que retorna el nombre de la dimension contextual .<br>*getSensorType():* Método que retorna el tipo de sensor.<br>*getTag():* Método que retorna la etiqueta para un valor del sensor dado.<br>*getTime():* Método que retorna el tiempo de muestreo. |

**Tabla 32.** Descripción de la clase *DataBaseCtx.*

| **Nombre** |
|---|
| *DataBaseCtx* |
| **Descripción** |
| Clase para crear y obtener la base de datos en la memoria externa. |
| **Atributos** |
| *DIRECTORY:* Atributo estático de tipo String que contiene la dirección donde se almacena la base de datos en la sdcard.<br>*db:* Objeto de tipo SQLiteDatabase para manipular la base de datos en la memoria externa |
| **Métodos** |
| *getDb():* Método para obtener la base de datos en la memoria externa de la sdcard.<br>*createTables():* Método para crear las tablas en la base de datos.<br>*closeDb():* Método para cerrar la base de datos. |

**Tabla 33.** Descripción de la clase *DAOClass.*

| **Nombre** |
|---|
| *DAOClass* |
| **Descripción** |
| Clase para la administración de las transacciones en la base de datos, para insertar, actualizar y eliminar los datos. |
| **Atributos** |
| No tiene Atributos |
| **Métodos** |
| insertData(String data): Método para insertar los datos en la tabla correspondiente.<br>updateData(String data, String conditional): Método para actualizar los datos en la tabla correspondiente.<br>deleteRegisters(int idIni, int idEnd): Método para borrar los registros desde un id inicial a un id final, en la tabla correspondiente.<br>readRegisters(int limit): Método que lee un número determinado de registros de la tabla correspondiente.<br>getNameTable(): Método no implementado que retorna la tabla correspondiente. |

**Tabla 34.** Descripción de la clase *SyncCurrentContext.*

| **Nombre** |
|---|
| *SyncCurrentContext* |
| **Descripción** |
| Clase que hereda de *Thread*, esta realiza la sincronización de la situación contextual actual del usuario para solicitar una recomendación de aplicaciones móviles. |
| **Atributos** |
| No tiene Atributos |
| **Métodos** |
| *SyncCurrentContext(HttpInterface, int, context)*: Método que envia la situación contextual actual al servidor y retorna el mensaje JSON de la recomendación. |

**Tabla 35.** Descripción de la clase *Configuration.*

| **Nombre** |
|---|
| *Configuration* |
| **Descripción** |
| Clase que permite ingresar y actualizar datos pares (Etiqueta - Valor) mediante la clase SharedPreference |
| **Atributos** |
| *PREFS_NAME:* Atributo estático de tipo String que contiene el nombre del archivo de preferencias de la clase SharedPreference. |

| |
|---|
| *settings:* Objeto de la clase SharedPreference.<br>*edit:* Objeto de la clase Editor, que permite ingresar valores y actualizarlos en el archivo de preferencias. |
| **Métodos** |
| *Configuration(Context context):* Constructor de la clase, que recibe como parámetro el objeto de la clase Context.<br>*getFloatConfig(String name):* Obtiene un valor de tipo *float* de la configuración para la etiqueta *name*.<br>*getIntConfig(String name):* Obtiene un valor de tipo *int* de la configuración para la etiqueta *name*.<br>*getLongConfig(String name):* Obtiene un valor de tipo *long* de la configuración para la etiqueta *name*.<br>*setFloatConfig(String name, float value):* Ingresa o actualiza un valor de tipo *float* para la etiqueta *name*.<br>*setIntConfig(String name, float value):* Ingresa o actualiza un valor de tipo *int* para la etiqueta *name*.<br>*setLongConfig(String name, float value):* Ingresa o actualiza un valor de tipo *long* para la etiqueta *name*. |

**Tabla 36.** Descripción de la clase *LoadConfiguration.*

| **Nombre** |
|---|
| *LoadConfiguration* |
| **Descripción** |
| Clase que carga la configuración inicial del colector de información a través de un JSON |
| **Atributos** |
| *ADDRESS:* Variable estática de tipo String que contiene la dirección de la base de datos en la sdcard.<br>*CONFIGURATION:* Variable estática que tiene el nombre de la etiqueta donde está alojada toda la configuración en el archivo JSON.<br>*STORAGE:* Variable estática que tiene el nombre de la etiqueta del segmento donde se establecen las configuraciones de almacenamiento en el archivo JSON.<br>*CONTEXTUAL_VAR:* Variable estática que tiene el nombre de la etiqueta del segmento donde se establecen las configuraciones de las dimensiones contextuales en el archivo JSON.<br>*SYNC:* Variable estática que tiene el nombre de la etiqueta del segmento donde se establecen las configuraciones de sincronización en el archivo JSON.<br>*preference:* Objeto de la clase SharedPreference.<br>*edit:* Objeto de la clase Editor que permite insertar y editar preferencias en el SharedPreference. |
| **Métodos** |
| *LoadConfiguration(Context):* Constructor de la clase.<br>*run():* Método que se llama al comenzar el hilo.<br>*loadJSON():* Método para cargar el JSON de configuración de la carpeta assets. |

| |
|---|
| *getConfigurationState():* Método para obtener el estado de la configuración, es decir, si se ah realizado o no.<br>*readContextualVar(JSONArray jArray):* Método para leer el segmento donde se establecen las configuraciones de las dimensiones contextuales en el archivo JSON.<br>*readStorageSettings (JSONArray jArray):* Método para leer el segmento donde se establecen las configuraciones de almacenamiento en el archivo JSON.<br>*readSyncSettings(JSONArray jArray):* Método para leer el segmento donde se establecen las configuraciones de sincronización en el archivo JSON.<br>*savePreference():*Método para guardar las preferencias en el SharedPreference. |

**Tabla 37.** Descripción de la clase *HttpConnection.*

| **Nombre** |
|---|
| *HttpConnection* |
| **Descripción** |
| Clase que realiza la conexión y comunicación con el servidor central |
| **Atributos** |
| *OK:* Atributo estático de tipo String, el cual contiene el valor de retorno OK del servidor, para confirmar que el envió de datos ha sido exitoso.<br>*PORT:* Atributo estático de tipo String que contiene le puerto del servidor.<br>*URL:* Atributo estático de tipo String que contiene la url del servidor.<br>*TIME_OUT_CONNECTION:* Atributo estático de tipo long, que contiene el tiempo máximo de espera para realizar la conexión.<br>*TIME_OUT_SOCKET:* Atributo estático de tipo long, que contiene el tiempo máximo de espera para el envió de un socket. |
| **Métodos** |
| *HttpConnection()*: Constructor de la clase.<br>*setServlet(String servlet):* Método para editar el Servlet al cual se enviaran los datos.<br>*sendWithPOST(String[][] data)*: Método que envía una matriz de datos mediante el método POST de http.<br>*processServerResponse(HttpResponse response)* : Método que procesa la respuesta del servidor. |

**Tabla 38.** Descripción de la clase *SyncContextReceiver.*

| **Nombre** |
|---|
| *SyncContextReceiver* |
| **Descripción** |
| Clase que recibe el evento de transmisión de datos y lanza el Thread correspondiente |
| **Atributos** |
| No tiene Atributos |
| **Métodos** |

| |
|---|
| *getSyncAction()*: Método que retorna un String que define la acción o evento a procesar.<br>*getSyncThread(Context context)* : Método que retorna un objeto tipo Thread que va a ejecutar el envío de los datos.<br>*onReceive(Context context, Intent intent)* : Método que recibe el mensaje o evento que lanza el Thread para el envío de los datos. |

**Tabla 39.** Descripción de la clase SyncThread.

| **Nombre** |
|---|
| *SyncThread* |
| **Descripción** |
| Clase que hereda de Thread , esta clase prepara la información de la base de datos para ser enviada al servidor mediante la clase HttpConnection |
| **Atributos** |
| *httpCx:* Objeto de la clase HttpConnection |
| **Métodos** |
| *getHttpCxt()*: Método que inicializa y retorna el objeto httpCx.<br>*getLimit()*: Método que retorna el límite de datos a transmitir.<br>*getJSONData(int limit)* : Método que arma el mensaje JSON con los datos correspondientes.<br>*getDAOClass()*: Método que retorna el objeto de DAOClass correspondiente.<br>*getServlet()*: Método que retorna el nombre del Servlet destino.<br>*run()*: Método que se ejecuta al lanzar el Thread, en el cual se realiza la creación del mensaje JSON y el envío del mensaje. |

## A.3. Modelo de Navegabilidad

A continuación se describe el diagrama de navegabilidad realizado para la implementación de la interfaz gráfica de usuario, para el cliente del sistema de recomendación, el cual, fue desarrollado mediante una aplicación móvil bajo la plataforma Android.

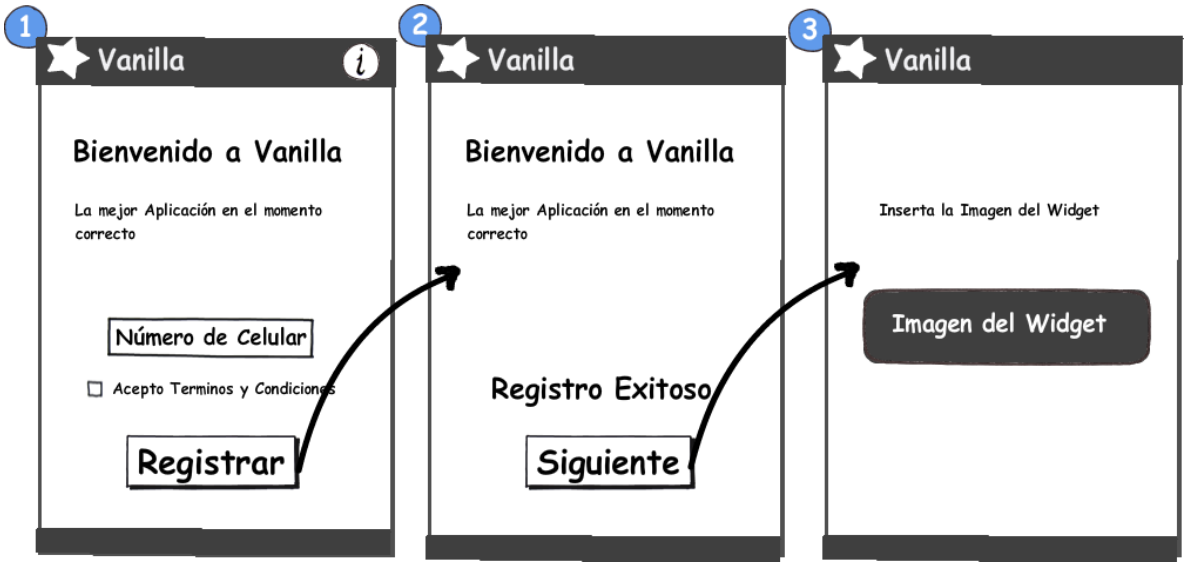


**Figura 8.** Registro en la Aplicación Móvil.

**Pantalla 1:**

Interfaz gráfica que solo se muestra en cuando se abre por primera vez la aplicación móvil, en la cual, se ingresa el número de celular para realizar el registro mediante la pulsación del botón Registrar. La Interfaz consta de un menú en el ActionView que permite ir a Términos y Condiciones (Figura 6).

**Pantalla 2:**

Interfaz gráfica que notifica si fue o no exitoso el registro de la aplicación, en caso de ser exitoso permite seguir a la siguiente interfaz, por el contrario notifica al usuario mendiante una alerta y permite realizar el registro nuevamente.

**Pantalla 3:**

Notifica al usuario que debe insertar la aplicación Widget de Vanilla en el Home Screen del Smartphone.

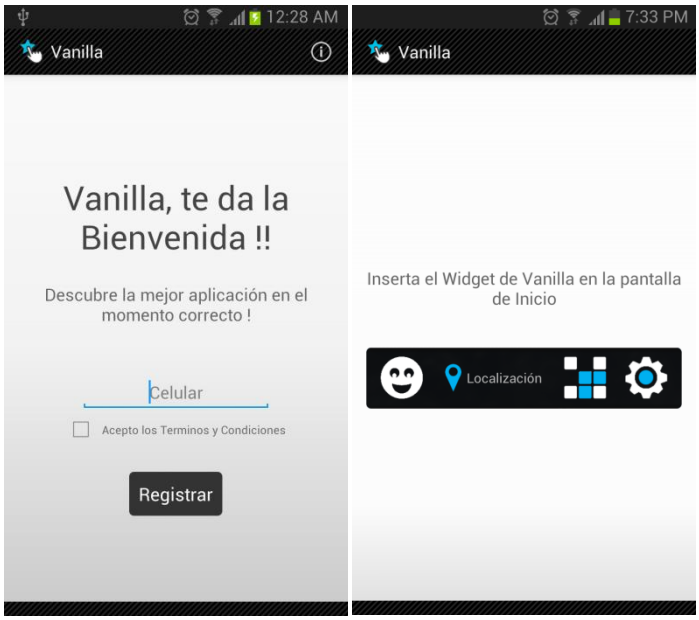


**Figura 9.** Interfaz Grafica Final (Pantalla 1 y Pantalla 3).

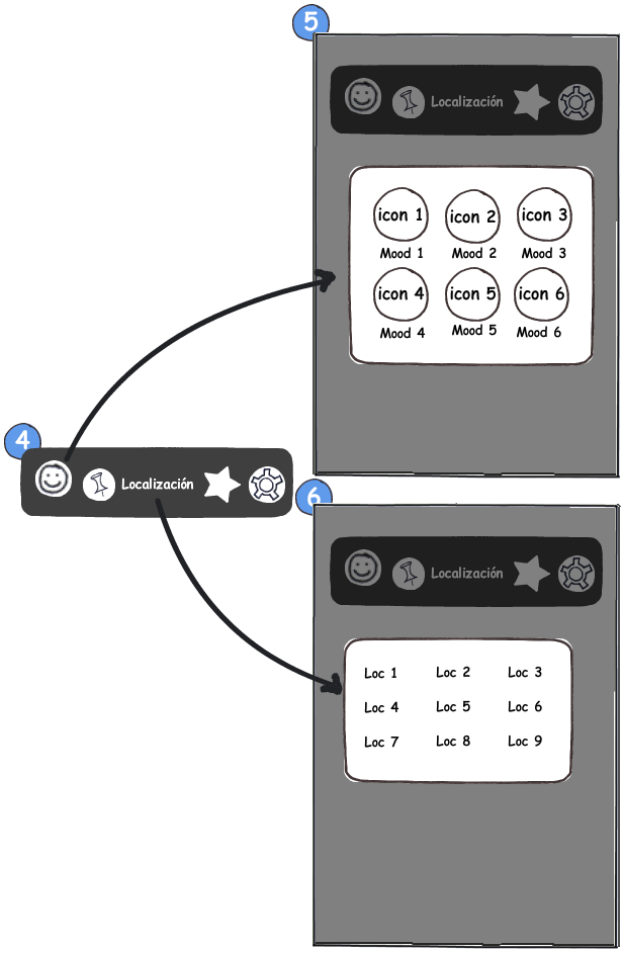


**Figura 10.** Registro en la Aplicación Móvil.

**Pantalla 4:**

Aplicación Widget para la pantalla de inicio de los Smartphone Android. Este Wdiget consta de cuatro botones:

- Ingreso de estado de animo
- Ingreso de localización Relativa
- Acceso rápido a Recomendación
- Acceso rápido a Configuración

**Pantalla 5:**

Ventana para el Ingreso rápido de los estados del ánimo, la cual contiene una etiqueta y un icono para cada estado.

**Pantalla 6:**

Ventana para el Ingreso rápido de la localización relativa, la cual solo tiene las etiquetas que representan a cada localización

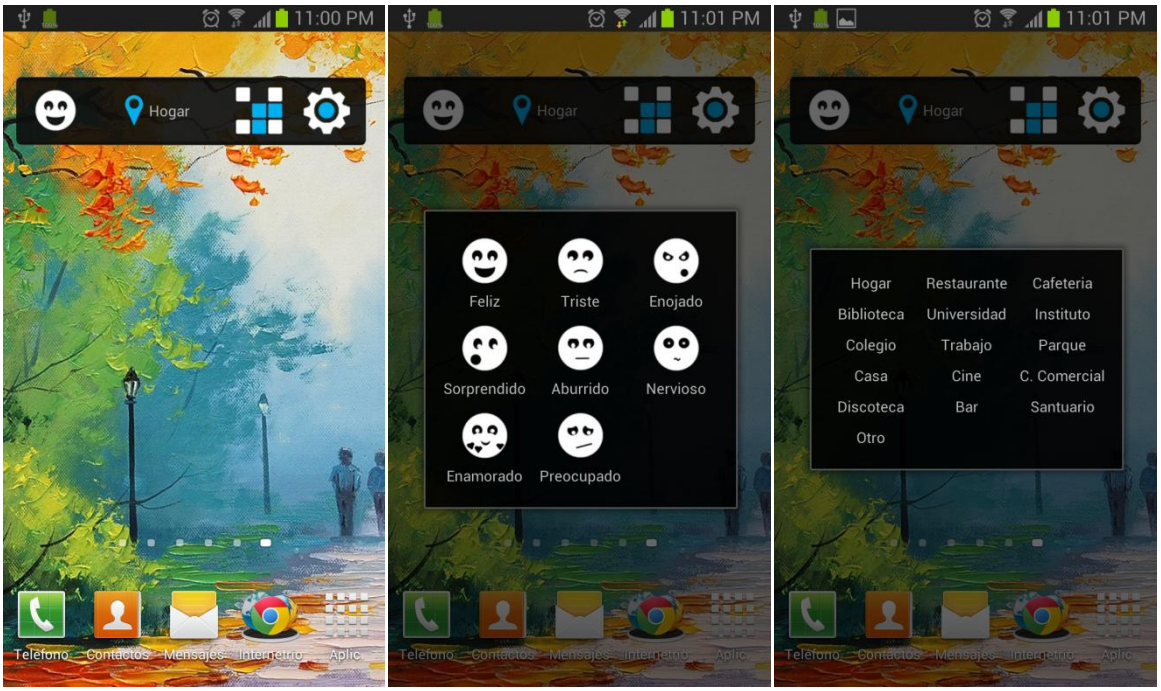


**Figura 11.** Interfaz Grafica Finales (Pantalla 4, Pantalla 5 y Pantalla 6).

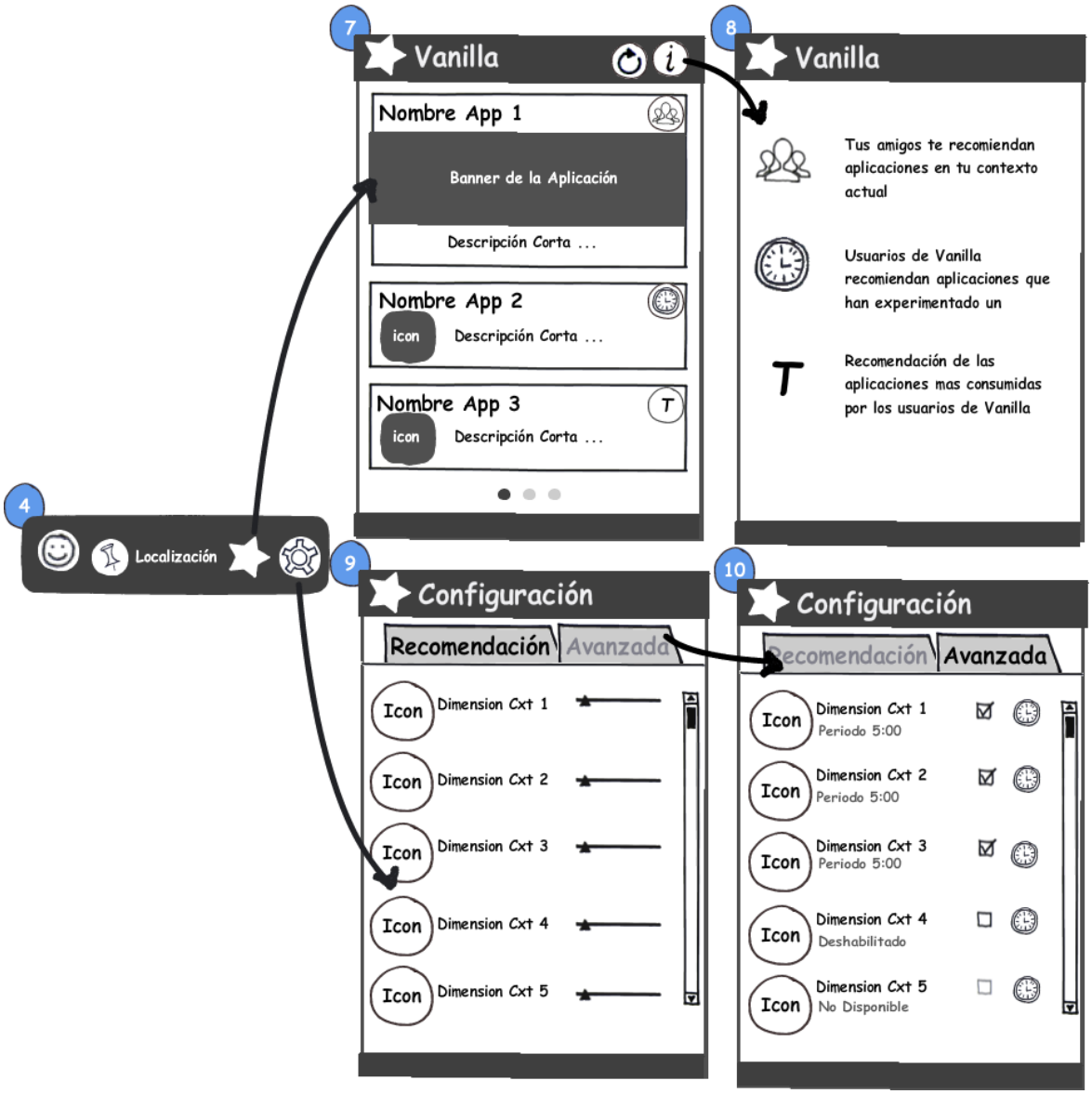


**Figura 12.** Registro en la Aplicación Móvil.

**Pantalla 7:**

Interfaz gráfica de recomendación de aplicaciones móviles, en la cual se presentan como máximo 9 aplicaciones, distribuidas en tres páginas, donde la primera aplicación de cada página ocupa la mayor porción de la página debido al banner. La pantalla consta de dos menús de opciones, el primero para actualizar la recomendación y el segundo para acceder a la pantalla de información.

**Pantalla 8:**

Interfaz gráfica de información de recomendación donde se explica los tipos de recomendación disponibles en la aplicación.

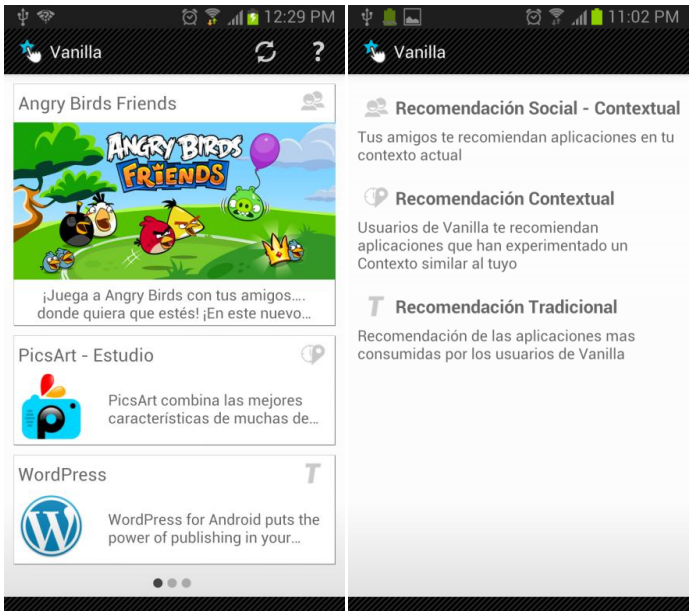


**Figura 13.** Interfaz Grafica Finales (Pantalla 7 y Pantalla 8).

**Pantalla 9:**

Interfaz de configuración de la importancia de los estados contextuales, en el cual, se determina el peso de cada variable contextual mediante un seekbar.

**Pantalla 10:**

Interfaz de configuración avanzada, donde se activan y desactivan las variables contextuales y se fijan los periodos de recolección de información.

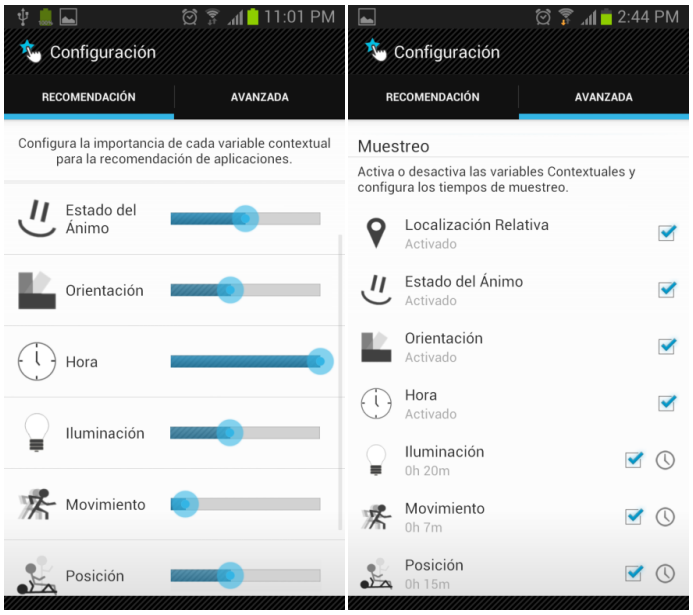


**Figura 14.** Interfaz Grafica Finales (Pantalla 9 y Pantalla 10).

## A.4. Diagrama de Despliegue

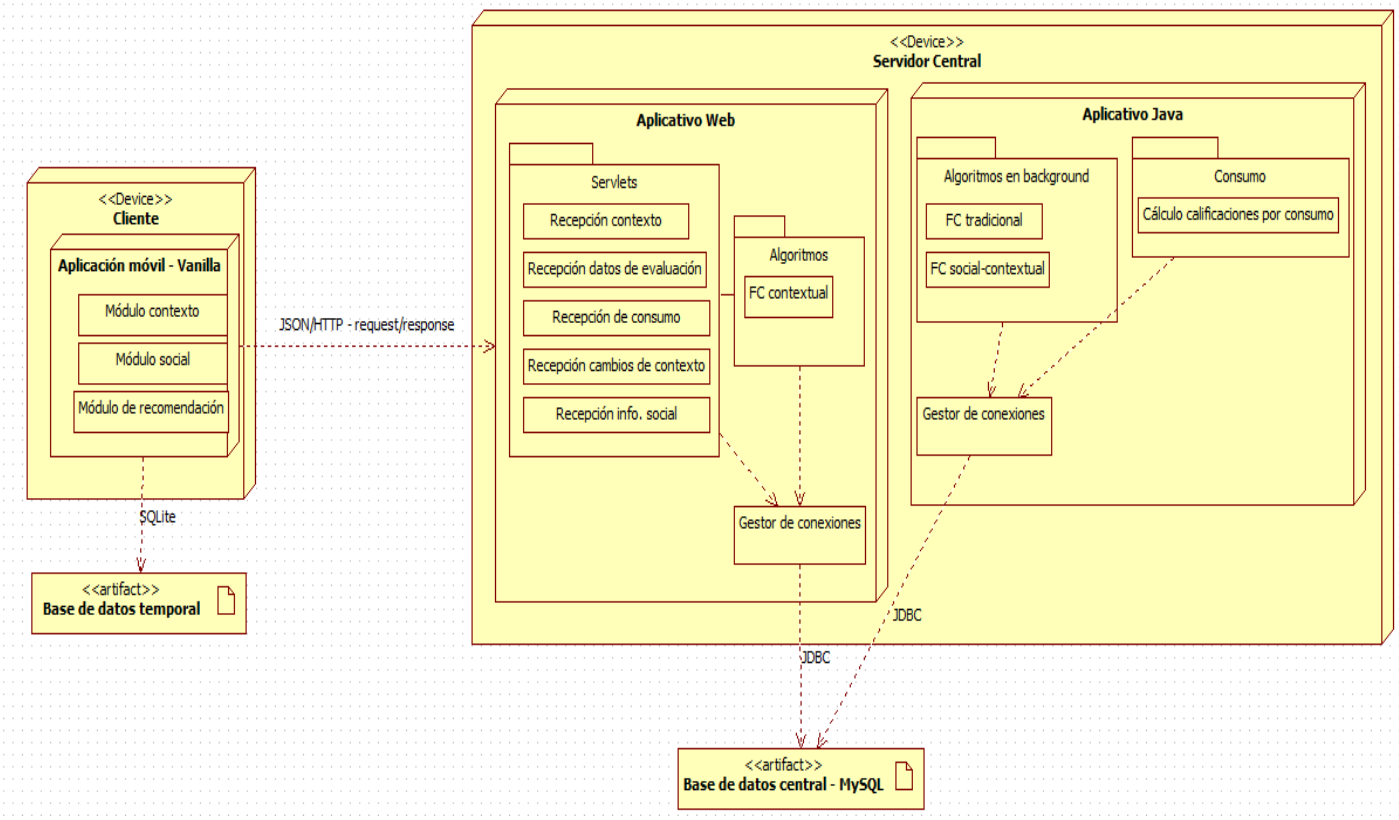


**Figura 15.** Diagrama de despliegue del sistema

El diagrama de despliegue del sistema, presentado en la Figura 1, ilustra el conjunto de nodos en los cuales se distribuyen físicamente los artefactos implementados para soportar su operación. Tal como se muestra en el diagrama, el sistema está dispuesto en una configuración típica de Cliente/Servidor, donde la lógica de negocia está distribuida en ambas partes.

Básicamente el nodo del cliente, aparte de actuar como intermediario, es un receptor de toda la información que se captura del usuario (información almacenada temporalmente hasta que sea transmitida), consecuentemente el nodo del servidor central se encarga del procesamiento de los algoritmos de recomendación y de procesar la información que recibe del nodo móvil, para almacenarla en la base de datos central.

# ANEXO B

## DIAGRAMA ENTIDAD RELACIÓN

## Diagrama entidad relación base de datos central

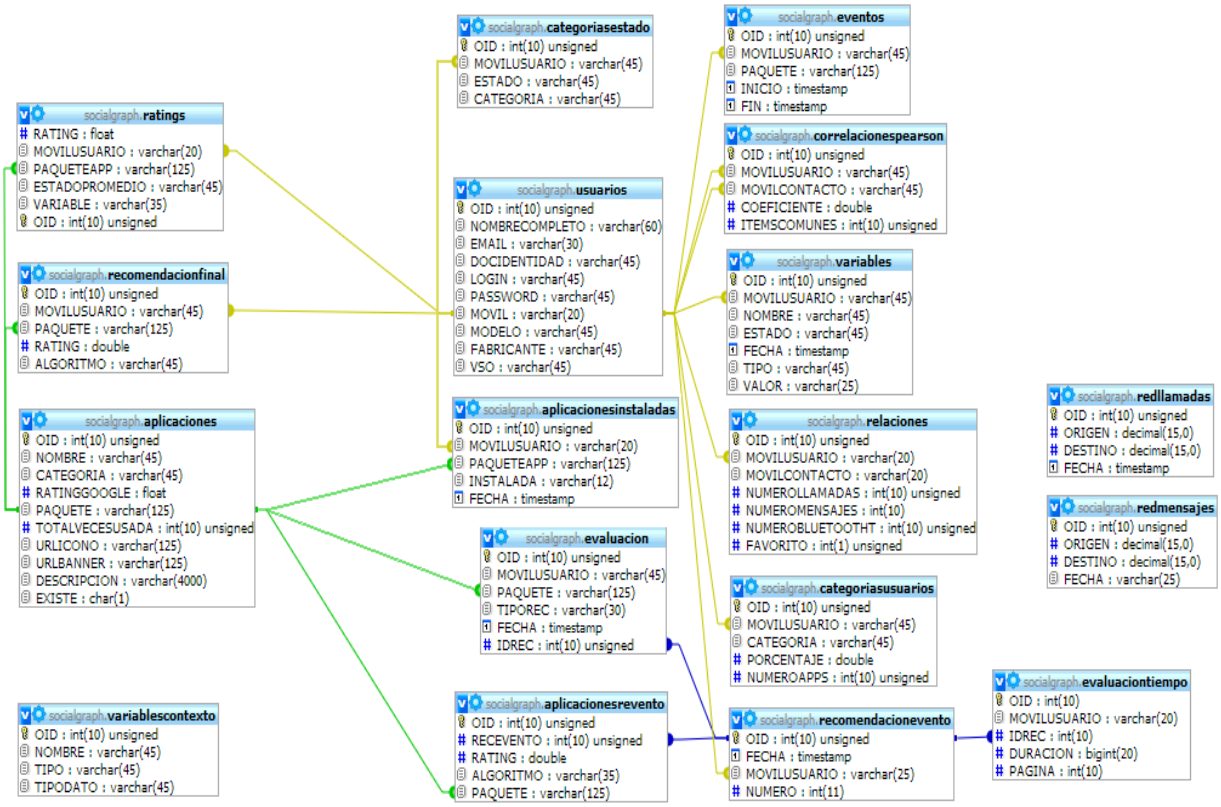


**Figura 16.** Modelo entidad relación.

**Tabla 40.** Descripción de tablas de la base de datos central.

| Nombre de tabla | Descripción |
|---|---|
| ***Aplicaciones*** | Repositorio de aplicaciones móviles, detectadas desde los móviles de todos los usuarios. |
| ***Usuarios*** | Usuarios registrados en el sistema. |
| ***Ratings*** | Scores de los usuarios sobre las aplicaciones que estos consumen ya sea por estado contextual o general. |
| ***Recomendacionfinal*** | Ratings predichos de las aplicaciones a recomendar por usuarios y por técnica de recomendación. |
| ***Categoriasestado*** | Categorías predominantes por cada estado contextual que cada usuario experimenta. |
| ***Aplicacionesinstaladas*** | Registro de las aplicaciones instaladas en |

| | |
|---|---|
| | los dispositivos móviles de los usuarios. |
| ***Relaciones*** | Registro de las relaciones entre usuarios en el sistema (grafo social). |
| ***Redllamadas*** | Registro de los eventos de llamadas de los usuarios del sistema. |
| ***Redmensajes*** | Registro de los eventos de sms's de los usuarios del sistema. |
| ***Eventos*** | Registro de los eventos de consumo de cada usuario registrado en el sistema. |
| ***Correlacionespearson*** | Registro de las aplicaciones consumidas entre los usuarios en el sistema (tabla implementada con el fin de mejorar el procesamiento del sistema). |
| ***Variables*** | Registro de los eventos de cambios de contexto de cada usuario registrado en el sistema. |
| ***Categoriasusuarios*** | Registro del número de aplicaciones consumidas por cada categoría de cada usuario. |
| ***Recomendacionevento*** | Registro de las recomendaciones provistas por el sistema. |
| ***Evaluación*** | Registro de las aplicaciones que cada usuario selecciona por cada recomendación que solicita. |
| ***Aplicacionesrevento*** | Registro de las aplicaciones mostradas por cada recomendación que cada usuario solicite. |
| ***Evaluaciontiempo*** | Registro del tiempo que cada usuario emplea en observar cada página de la interfaz de recomendación. |
| ***Variablescontexto*** | Registro de las dimensiones contextuales del modelo contextual planteado. |

# ANEXO C

## DIAGRAMA DE FLUJO DE LAS TECNICAS DE RECOMENDACIÓN

## C.1. Técnica Tradicional

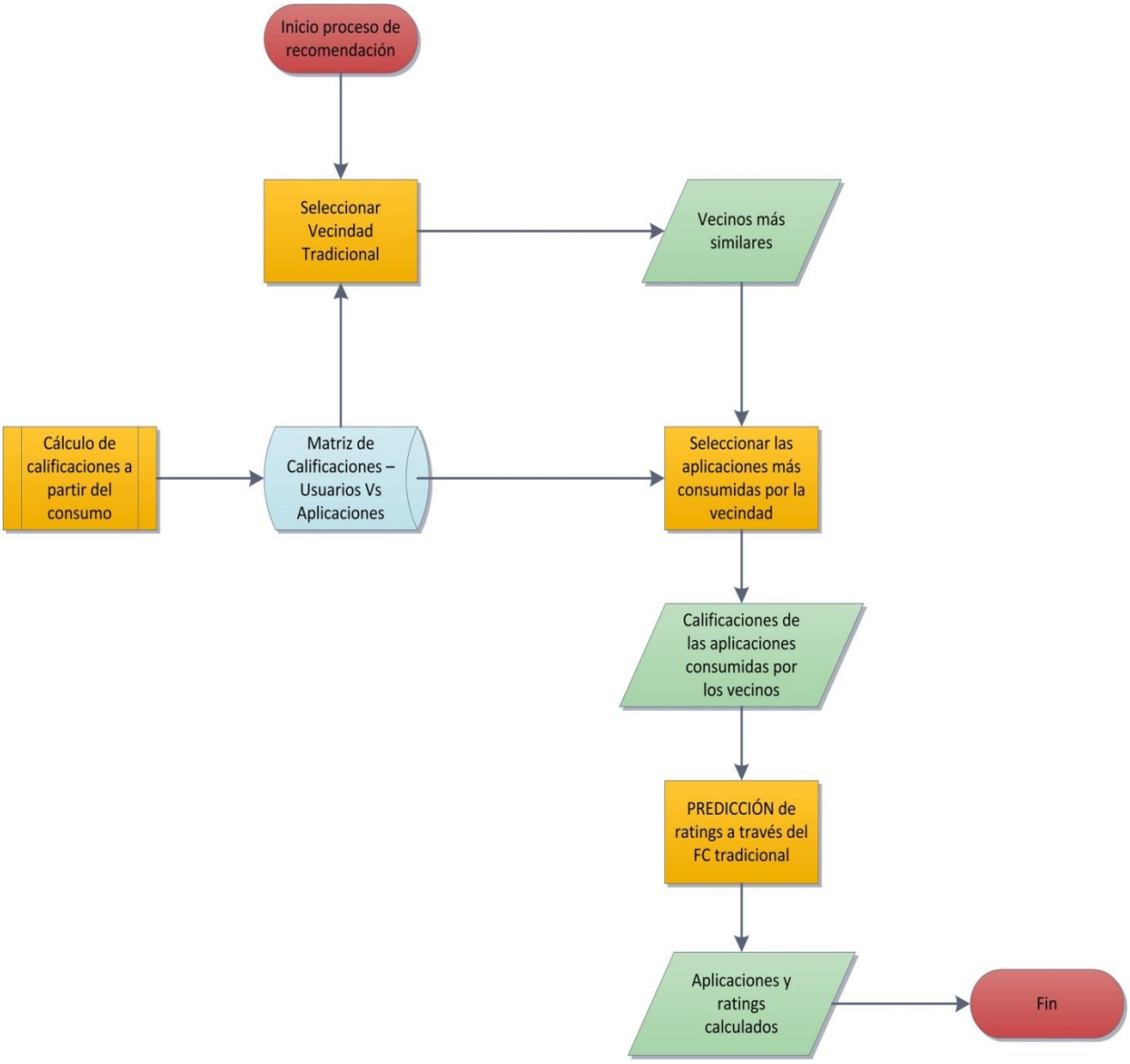


**Figura 17.** Flujo del algoritmo definido para la recomendación tradicional de aplicaciones móviles

**Tabla 41.** Descripción del flujo de la Técnica Tradicional.

| Repositorios | |
|---|---|
| **Nombre** | **Descripción** |
| Matriz de calificaciones – Usuarios vs Aplicaciones | Repositorio en donde se encuentra la información relacionada con las calificaciones y aplicaciones consumidas, de todos los usuarios registrados en el sistema |

| **Datos** | |
|---|---|
| **Nombre** | **Descripción** |
| Vecinos más similares: | Lista o arreglo que contiene a los 40 vecinos que tienen más aplicaciones consumidas en común con el usuario (cálculo de *N*). |
| Calificaciones de las aplicaciones consumidas por los vecinos | Colección o lista indexada, donde se almacenan los usuarios pertenecientes a la vecindad calculada, con sus respectivas **calificaciones** de las aplicaciones que más hayan consumido |
| Aplicaciones y ratings calculados | Colección o lista indexada, donde se almacenan las aplicaciones recomendadas, junto con los **ratings** predichos a través del algoritmo de filtrado colaborativo implementado para este tipo de recomendación. |
| **Procesos** | |
| **Nombre** | **Descripción** |
| ***Seleccionar vecindad tradicional*** | **Entradas:** Matriz de calificaciones – Usuarios vs Aplicaciones (Repositorio). |
| | **Salidas:** Vecinos más similares (Lista). |
| | **Descripción:** Este proceso se encarga de filtrar el conjunto total de personas registradas en el sistema, con el objetivo de generar la vecindad del usuario de acuerdo al número de aplicaciones que han sido consumidas en común (la intersección entre el conjunto de aplicaciones consumidas por el usuario u, con el conjunto de su vecino u'). De acuerdo a lo que sugieren trabajos como (Liu, y otros, 2010), se seleccionan las 40 personas que más aplicaciones tengan en común con el usuario. |
| ***Seleccionar las aplicaciones más consumidas por la vecindad*** | **Entradas:** Vecinos más similares (Lista), Matriz de calificaciones – Usuarios vs Aplicaciones (Repositorio). |
| | **Salidas:** Calificaciones de las aplicaciones más consumidas por los vecinos (Arreglo en sesión). |
| | **Descripción:** Este proceso consiste en la selección de las aplicaciones más consumidas por la vecindad del usuario. Estas corresponden a las que cuentan con la mayor calificación |
| ***Ejecutar filtrado colaborativo usuario – usuario para la aplicación i*** | **Entradas:** Calificaciones totales de la Aplicación i (Colección en sesión). |
| | **Salidas:** Aplicaciones y ratings calculados (Variable en sesión, que se va adjuntando a la Colección de aplicaciones y ratings calculados). |
| | **Descripción:** Proceso encargado de predecir el rating de las aplicaciones correspondientes, mediante la implementación del filtrado colaborativo usuario – usuario basado en Promedio ponderado o "Weighted averaging". El factor de similitud para este algoritmo de predicción, corresponde al factor de correlación de Pearson tradicional descrito en el capítulo 5. |
| **Sub-procesos** | |
| **Nombre** | **Descripción** |

| | |
|---|---|
| ***Cálculo de predicciones a partir del consumo*** | **Entradas: "**Eventos de consumo" |
| | **Salidas:**. Matriz de calificaciones – Usuarios vs Aplicaciones (Repositorio). |
| | **Descripción:** Subproceso encargado calcular las calificaciones de las aplicaciones que los usuarios consumen de acuerdo a los eventos de consumo registrados. Sub-proceso que sigue los lineamientos definidos en el capítulo 4. |

## C.2. Técnica Contextual

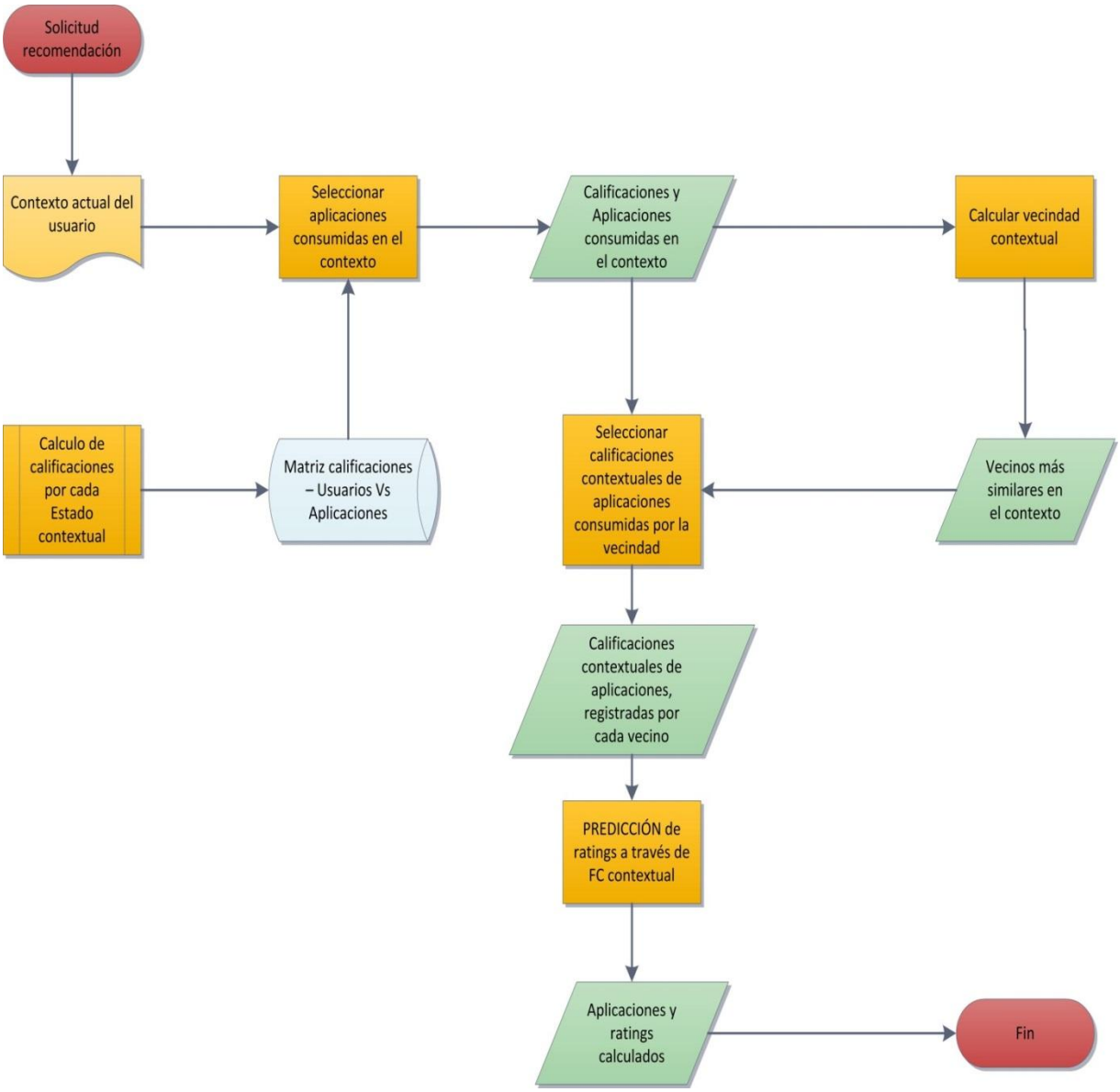


**Figura 18.** Flujo del algoritmo definido para la recomendación contextual de aplicaciones móviles.

**Tabla 42.** Descripción del flujo de la Técnica Contextual.

| **Documentos** | |
|---|---|
| **Nombre** | **Descripción** |
| ***Contexto actual del usuario*** | **Descripción:** Documento que representa de manera formal el contexto actual del usuario |
| **Repositorios** | |
| **Nombre** | **Descripción** |
| Matriz de calificaciones – Usuarios vs Aplicaciones | Repositorio en donde se encuentra la información relacionada con las calificaciones y aplicaciones consumidas, de todos los usuarios registrados en el sistema |
| **Datos** | |
| **Nombre** | **Descripción** |
| ***Calificaciones y aplicaciones consumidas en el contexto*** | **Descripción:** Colección o lista indexada, que se compone de TODOS los usuarios del sistema y de las aplicaciones que por lo menos estos hayan consumido alguna vez en cualquiera de los estados contextuales dados, respectivamente. |
| ***Vecinos más similares en el contexto*** | **Descripción:** Lista que contiene a los 40 vecinos que tienen más aplicaciones consumidas en común con el usuario en el contexto dado (la justificación de que sean 40 vecinos, se da en el proceso que genera este *dato* como salida). |
| ***Calificaciones contextuales de aplicaciones, registradas por cada vecino*** | **Descripción:** Colección o lista indexada, que contiene los VECINOS más similares en el contexto del usuario, junto con las calificaciones contextuales de las aplicaciones correspondientes. Dichas calificaciones se infieren teniendo en cuenta lo estipulado en el capítulo 4 referente al cálculo de calificaciones contextuales |
| ***Aplicaciones y ratings calculados*** | **Descripción:** Colección o lista indexada, donde se almacenan las aplicaciones recomendadas, junto con los ratings predichos a través del algoritmo de filtrado colaborativo contextual |
| **Procesos** | |
| **Nombre** | **Descripción** |
| ***Seleccionar aplicaciones consumidas en el contexto*** | **Entradas:** Contexto actual del usuario (Documento)**,** Matriz de calificaciones – Usuarios vs Aplicaciones (Repositorio). |
| | **Salidas:** Calificaciones y aplicaciones consumidas en el contexto (Colección). |
| | **Descripción:** Este proceso selecciona todas las aplicaciones que hayan sido consumidas por TODOS los usuarios del sistema, en cualquier estado contextual dado. |
| ***Calcular vecindad contextual*** | **Entradas:** Calificaciones y aplicaciones consumidas en el contexto (Repositorio). |
| | **Salidas:** Vecinos más similares en el contexto (Arreglo en sesión). |
| | **Descripción:** Este proceso se encarga de filtrar el conjunto total de personas registradas en el sistema al generar una vecindad de acuerdo al número de aplicaciones que han sido consumidas en común en los estados contextuales que se han definido (la |

| | |
|---|---|
| | intersección entre el conjunto de aplicaciones consumidas por el usuario u, con el conjunto de su vecino u' en el contexto dado). De manera similar a la recomendación tradicional, el tamaño de la vecindad es de 40 personas. |
| ***Seleccionar calificaciones contextuales de aplicaciones consumidas por la vecindad*** | **Entradas:** Vecinos más similares en el contexto, Calificaciones y aplicaciones consumidas en el contexto. |
| | **Salidas:** Calificaciones contextuales de las aplicaciones de las aplicaciones consumidas por vecinos. |
| | **Descripción:** Proceso encargado de seleccionar las calificaciones contextuales de la vecindad, |
| ***Predicción de ratings a través de FC contextual*** | **Entradas:** Calificaciones contextuales de las aplicaciones de las aplicaciones consumidas por vecinos (colección). |
| | **Salidas:** Aplicaciones y ratings calculados (Colección). |
| | **Descripción:** Proceso encargado de predecir el rating de las aplicaciones correspondientes, mediante la implementación del filtrado colaborativo contextual usuario – usuario denominada Promedio ponderado o "Weighted averaging". El FC contextual sigue la misma filosofía del FC tradicional, con la única diferencia de que las calificaciones con las que se trabaja, corresponden a las del contexto dado, al igual que en el factor de similitud contextual explicado en el capítulo 5 |
| **Sub-procesos** | |
| **Nombre** | **Descripción** |
| ***Cálculo de calificaciones por cada estado contextual*** | **Entradas:** "Eventos de consumo". |
| | **Salidas:** Matriz de calificaciones – Usuarios vs Aplicaciones. |
| | **Descripción:** Sub-proceso encargado calcular las calificaciones de las aplicaciones que los usuarios consumen de acuerdo a los eventos de consumo registrados y a los estados contextuales que experimenten. Sub-proceso que sigue los lineamientos definidos en el capítulo 4. |

## C.3 Técnica Social-Contextual

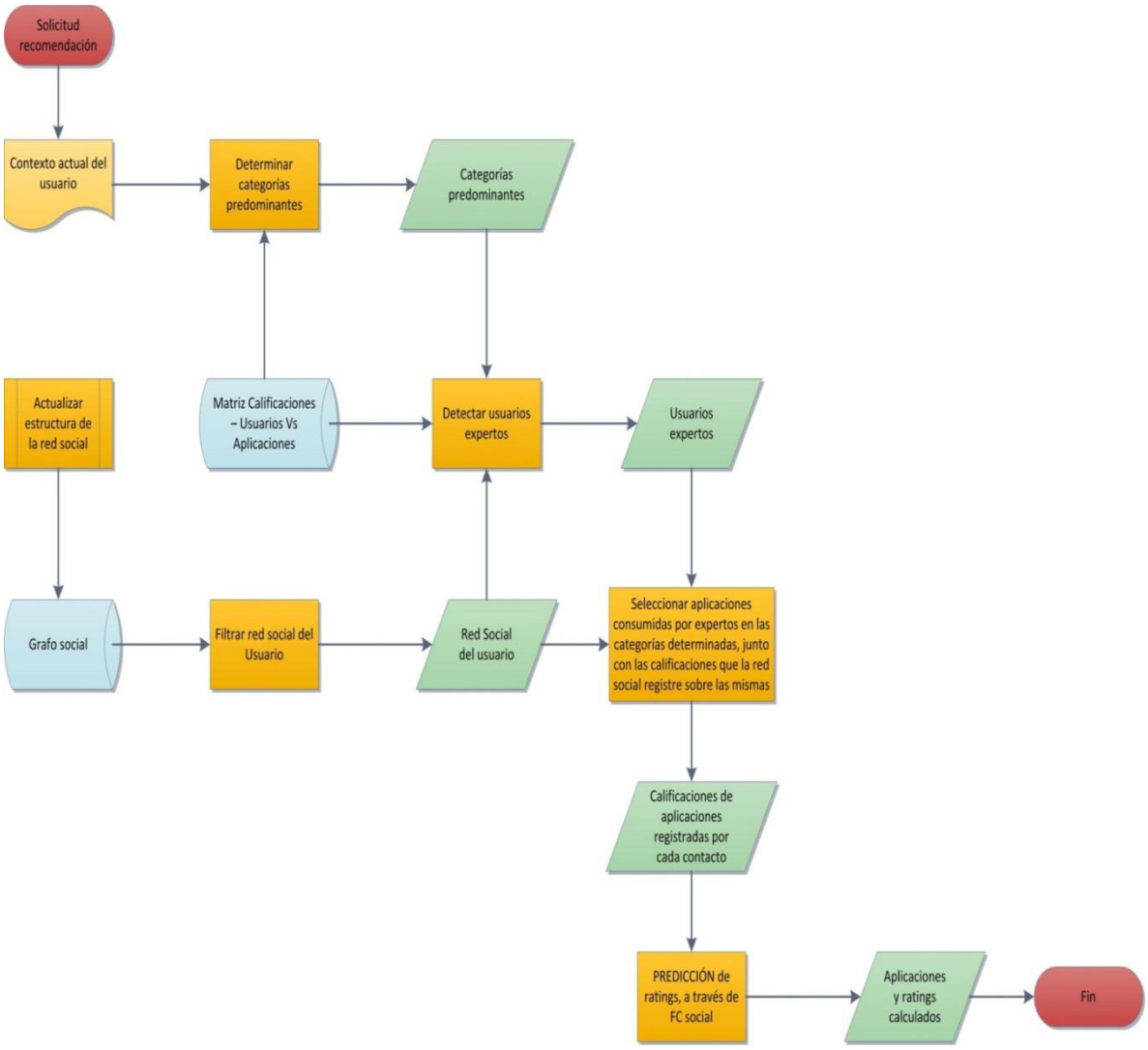


**Figura 19.** Flujo del algoritmo definido para la recomendación social-contextual de aplicaciones móviles.

**Tabla 43.** Descripción del flujo de la Técnica Social-Contextual.

| **Documentos** | |
|---|---|
| **Nombre** | **Descripción** |
| ***Contexto actual del usuario*** | **Descripción:** Documento que representa de manera formal el contexto actual del usuario |
| **Repositorios** | |
| **Nombre** | **Descripción** |
| **Matriz de calificaciones – Usuarios vs Aplicaciones** | Repositorio en donde se encuentra la información relacionada con las calificaciones y aplicaciones consumidas, de todos los usuarios registrados en el sistema |
| ***Grafo social*** | **Descripción:** Repositorio en donde está almacenada toda la |

| | |
|---|---|
| | información referente a las interacciones que todos los usuarios del sistema mantienen entre sí. Dicha información incluye llamadas, sms's y la selección de contactos favoritos [Capítulo 5]. |
| **Datos** | |
| **Nombre** | **Descripción** |
| ***Categorías predominantes*** | **Descripción:** Arreglo o lista indexada que contiene las categorías que el usuario tiende a consumir en el contexto de entrada [Hipótesis 2] [capítulo 4, detección de categoría por estado]. |
| ***Red social del usuario*** | **Descripción:** Colección o lista indexada que contiene a todos los contactos del usuario, junto con el número de interacciones que se han mantenido entre estos (favoritismo, número de llamadas y sms's). |
| ***Usuarios expertos*** | **Descripción:** Colección o lista indexada, que contiene de manera conjunta a las categorías y a los contactos (integrantes de la red social del usuario) que se considera son expertos en dichas categorías de aplicaciones |
| ***Calificaciones de aplicaciones registradas por cada contacto*** | **Descripción:** Colección o lista indexada, que contiene a los todos contactos del usuario, junto con sus respectivas calificaciones de las aplicaciones seleccionadas bajo los criterios de los *usuarios expertos*. |
| ***Aplicaciones y ratings calculados*** | **Descripción:** Colección o lista indexada donde se almacenan las aplicaciones recomendadas junto con los ratings predichos a través del algoritmo de filtrado colaborativo social. |
| **Procesos** | |
| **Nombre** | **Descripción** |
| ***Filtrar red social del usuario*** | **Entradas:** Grafo social. |
| | **Salidas:** Red social del usuario. |
| | **Descripción:** Este proceso se encarga de filtrar el conjunto total de personas registradas en el sistema, de acuerdo a las relaciones y número de interacciones existentes con el usuario. |
| ***Determinar categorías predominantes*** | **Entradas:** Contexto actual del usuario. |
| | **Salidas:** Categorías predominantes. |
| | **Descripción:** Proceso que consiste en determinar la(s) categoría(s) que el usuario tiende a consumir en su contexto actual [Bohmer-capitulo1]. De acuerdo a lo especificado en el capítulo 4 respecto al cálculo de categorías predominantes por contexto, se toma el término relevancia (valor que oscila entre 0 y 1) como el indicador del número de aplicaciones que se deben recomendar por categoría respectivamente. De esta forma se define el número de aplicaciones recomendadas de la categoría predominante $m$, como el producto entre su relevancia y el número total de aplicaciones que se muestran en la recomendación social-contextual. |

| | |
|---|---|
| | $NumAppsRec_{C_m} = NumAppsRecSocial.\left(\frac{\sum_{n=1}^{N1} W_{V_n,C_m}}{W}\right)$<br><br>Se resalta el hecho de que el factor dentro del paréntesis en la expresión de arriba, corresponde a la ecuación N, para la detección de categorías predominantes. |
| ***Detectar usuarios expertos:*** | **Entradas:** Red social del usuario, categorías predominantes, Matriz calificaciones – Usuarios vs Aplicaciones. |
| | **Salidas:** Usuarios expertos. |
| | **Descripción:** Proceso que calcula el número de aplicaciones que cada contacto del usuario, ha consumido en cada categoría predominante. En este sentido, se seleccionan a los contactos que cuentan con mayor número de aplicaciones consumidas, para conformar el grupo de los usuarios expertos (el número máximo de contactos expertos, se ha establecido como 5). |
| ***Seleccionar las aplicaciones consumidas por expertos, junto con las calificaciones que la red social registre sobre las mismas*** | **Entradas:** Red social del usuario, Usuarios expertos. |
| | **Salidas:** Calificaciones de aplicaciones registradas por cada contacto). |
| | **Descripción:** Proceso encargado de obtener las aplicaciones sugeridas por los usuarios expertos de acuerdo a las categorías predominantes inferidas. De igual forma, su función también se centra en obtener todas las calificaciones de dichas aplicaciones, que los contactos del usuario tengan registradas. |
| ***Ejecutar filtrado colaborativo usuario – usuario para la aplicación i:*** | **Entradas:** Colección de las calificaciones totales de la Aplicación i (Colección en sesión). |
| | **Salidas:** Rating Aplicación i (Variable en sesión, que se va adjuntando a la Colección de aplicaciones y ratings calculados). |
| | **Descripción:** Proceso encargado de predecir el rating de las aplicaciones correspondientes, mediante la implementación del filtrado colaborativo social basada en Promedio ponderado o “Weighted averaging”. A diferencia del FC tradicional, el FC social implementa el factor de similitud basado en relaciones [Capítulo 5, factor de similitud social |
| **Sub-procesos** | |
| **Nombre** | **Descripción** |
| ***Ejecutar filtrado colaborativo usuario – usuario para la aplicación i*** | **Entradas:** “Eventos relacionados con interacciones entre contactos” |
| | **Salidas:** Grafo Social. |
| | **Descripción:** Sub-proceso cuya función se centra en modelar y actualizar la estructura del grafo social de contactos, teniendo en cuenta los eventos registrados que se relacionan con las llamadas o sms’s realizados. |

# ANEXO D

## Intercambio de mensajes

## D.1. Intercambio de Mensajes

A continuación se describe los mensajes que se envían del cliente móvil al servidor central.

**Registro de Usuario**

```
{ "type":"user_register",
  "user":"numero de celular",
  "model":"modelo del celular",
  "manufacturer":"fabricante del celular",
  "v_so":"version de Android SDK"
}
```

**Figura 20.** JSON registro de usuario

Mensaje enviado desde la aplicación movil al servidor central, para efectuar el registro del usuario en el sistema de recomendación. Entre la informacion transmitida se encuentran:

- **type:** Indentifica el tipo de mensaje.
- **user:** Numero telefonico con el cual se idenfica al usuario.
- **model:** Modelo de Smartphone a traves del cual se realizo el registro.
- **manufacturer:** Fabricante del Smartphone.
- **v_so:** Version de Android SDK.

El servidor central retorna un **101** en caso de ser correcto el registro, por el contrario retorna un **404**.

**Sincronización de Aplicaciónes instaladas**

```
{ "type":"Applications",
  "user":"numero de celular",
  "data":[
          {"package":"nombre del paquete",
           "state":"installed/uninstalled",
           "datetime":"YYYY-MM-DD HH:mm:ss"
          },
          .
          .
          .
          {"package":"nombre del paquete n",
           "state":"installed/uninstalled",
           "datetime":"YYYY-MM-DD HH:mm:ss"
          }
]
}
```

**Figura 21.** Sincronización de aplicaciones instaladas.

Mensaje enviado desde la aplicación movil al servidor central, para sincronizar los datos relacionados a instalacion, desinstalacion y actualizacion de las aplicaciones moviles en el Smartphone del usuario. Entre la informacion transmitida se encuentran:

- **type:** Indentifica el tipo de mensaje.
- **user:** Numero de telefonico con el cual se idenfica al usuario.
- **package:** Nombre del paquete principal de la aplicación movil, este nombre es unico e identifica a cada aplicación movil en el GooglePlay.
- **datetime:** Fecha y hora en las cuales se realizo el evento.

El servidor central retorna un **101** en caso de ser correcto la sincronización, por el contrario retorna un **404**.

**Sincronización de Consumo de Aplicaciones Moviles**

```
{ "type":"Consumption",
  "user":"numero de celular",
  "data":[
    {"package":"nombre del paquete",
     "times":[
                {"time_start":"YYYY-MM-DD hh:mm:ss",
                 "time_end":"YYYY-MM-DD hh:mm:ss"
                },
                .
                .
                .
              ]
    },
    .
    .
    .
    {"package":"nombre del paquete n",
     "times":[
                {"time_start":"YYYY-MM-DD hh:mm:ss",
                 "YYYY-MM-DD hh:mm:ss"},
                .
                .
                .
              ]
    }
]
}
```

**Figura 22.** JSON eventos de consumo de aplicaciones móviles.

Mensaje enviado desde la aplicación movil al servidor central, para sincronizar los datos relacionados al consumo de aplicaciones moviles en el Smartphone del usuario. Entre la informacion transmitida se encuentran:

- **type:** Indentifica el tipo de mensaje.
- **user:** Numero telefonico con el cual se idenfica al usuario.

- **package:** Nombre del paquete principal de la aplicación movil, este nombre es unico e identifica a cada aplicación movil en el GooglePlay.
- **times:** Tiempos de inicio y fin en el uso de una aplicación movil, el uso esta determinado al tiempo en que una aplicación esta activa en la pantalla.

El servidor central retorna un **101** en caso de ser correcto la sincronización, por el contrario retorna un **404**.

### Sincronizacion del Historial Contextual del Usuario

```
{ "type":"Context", "user":"numero de celular", "data":[
    {"variable":"Dimension Contextual",
     "states":[
                {"name":"Estado Contextual",
                 "values":"valor 1|valor 2|valor 3",
                 "timestamp":"YYYY-MM-DD hh:mm:ss"
                },
                .
                .
                .
                {"name":"Estado Contextual N",
                 "values":"valor 1|valor 2|valor 3",
                 "timestamp":"YYYY-MM-DD hh:mm:ss"
                }
              ]
    },
    .
    .
    .
    {"variable":"Dimension Contextual N",
     "states":[
                .
                .
                .
                {"name":"Estado Contextual N",
                 "values":"valor 1|valor 2|valor 3",
                 "timestamp":"YYYY-MM-DD hh:mm:ss"
                }
              ]
    }
]
}
```

**Figura 23. JSON eventos de cambios en el contexto de un usuario.**

Mensaje enviado desde la aplicación movil al servidor central, para sincronizar los datos relacionados al historial contextual del usuario. Entre la informacion transmitida se encuentran:

- **type:** Indentifica el tipo de mensaje.
- **user:** Numero telefonico con el cual se idenfica al usuario.
- **variable:** Nombre de la Dimension contextual, por ejemplo : Iluminación, Localizacion relativa, clima, etc.
- **name:** Nombre del estado contextual correspondiente a la dimension contextual.

- **values:** Valores numericos del estado contextual, solo en caso de tenerlos.
- **datetime:** Fecha y hora en las cuales se realizo el evento.

El servidor central retorna un **101** en caso de ser correcto la sincronización, por el contrario retorna un **404**.

**Sincronización de Contactos**

```
{ "type":"Contacts",
  "user":"numero de celular",
  "data":[
    {"phone":"telefono del contacto 1"},
    .
    .
    .
    {"phone":"telefono del contacto n"}
]
}
```

**Figura 24.**JSON contactos de un usuario.

Mensaje enviado desde la aplicación movil al servidor central, para sincronizar los datos relacionados a los contactos usuario. Entre la informacion transmitida se encuentran:

- **type:** Indentifica el tipo de mensaje.
- **user:** Numero telefonico con el cual se idenfica al usuario.
- **phone:** Numero de celular correspondiente a cada contacto del usuario.

El servidor central retorna un **101** en caso de ser correcto la sincronización, por el contrario retorna un **404**.

**Sincronizacion de Interacciones Sociales**

```
{ "type":"Telephony",
  "user":"numero de celular de usuario",
  "data":[
          {"origin":"numero de origen",
           "destination":"numero de destino",
           "type":"tipo",
           "datetime":"YYYY-MM-DD hh:mm:ss"},
          .
          .
          .
          {"origin":"numero de origen N",
           "destination":"numero de destino N",
           "type":"tipo",
           "datetime":"YYYY-MM-DD hh:mm:ss"},
        ]
}
```

**Figura 25.** JSON registro de llamadas y sms's de un usuario.

Mensaje enviado desde la aplicación movil al servidor central, para sincronizar los datos relacionados a las interacciones del usuario con sus contactos. Entre la informacion transmitida se encuentran:

- **type:** Indentifica el tipo de mensaje.
- **user:** Numero telefonico con el cual se idenfica al usuario.
- **origin:** Numero telefonico origen o el que inicio la interaccion.
- **destination:** Numero telefonico destino de la interacción.
- **type:** Tipo de interaccion, esta puede ser llamada o SMS.
- **datetime:** Fecha y hora en las cuales se realizo la interacción.

El servidor central retorna un **101** en caso de ser correcto la sincronización, por el contrario retorna un **404**.

**Solicitud de Recomendación**

```
{"type":"current_context",
 "user":"numero de celular",
 "data":[
        {"state":"Estado contextual",
         "dimension":"Dimension contextual",
         "weight":"Importancia de la Dimension contextual"
        },
        .
        .
        .
        {"state":"Estado contextual N",
         "dimension":"Dimension contextual N",
         "weight":"Importancia de la Dimension contextual N"
        },
        ]
}
```

**Figura 26.** JSON solicitud de recomendación.

Mensaje enviado desde la aplicación movil al servidor central, para solicitar una recomendación de aplicaciones moviles, donde se envia la situacion contextual actual del usuario. Entre la informacion transmitida se encuentran:

- **type:** Indentifica el tipo de mensaje.
- **user:** Numero telefonico con el cual se idenfica al usuario.
- **dimension:** Dimension contextual.
- **state:** Estado contextual actual, correspondiente a la Dimension contextual.
- **weight:** Peso o importancia de la dimension contextual para el usuario.

El servidor central envial el siguiente mensaje como respuesta a la solicitud.

```
{"type":"recommendation",
 "user":"numero de celular",
 "state":"0|1|2",
 "idrecommendation":"indentificador unico de la recomendacion",
 "recommendations":[
        {"algorithim":"tradicional", "state":"0|1", "apps":[
                {"package":"nombre del paquete",
                 "rating"="calificacion de la aplicacion",
                 "description"="descripcion de la aplicacion",
                 "name"="nombre de la aplicacion",
                 "icon"="icono de la aplicacion",
                 "banner"="banner de la aplicacion"},
                ...
            ]
        },
        {"algorithim":"socialContextual", "state":"0|1", "apps":[
                ...
            ]
        },
        {"algorithim":"contextual", "state":"0|1", "apps":[
                ...
            ]
        }

    ]
}
```

**Figura 27.** JSON lista de aplicaciones recomendadas.

Mensaje enviado desde servidor central a la aplicación movil, con el contenido de la recomendación. Entre la informacion transmitida se encuentran:

- **type:** Indentifica el tipo de mensaje.
- **user:** Numero telefonico con el cual se idenfica al usuario.
- **state:** Estado de toda la recomendación, el cual puede tomar tres valores: 0 para informar que la recomendación no esta disponible, 1 notifica normalidad en la recomendación y 2 informa que la recomendación solo contiene aplicaciones populares debido a la falta de información del consumo del usuario.
- **idrecommendation:** Indentificacion unica del evento de la recomendación.
- **algorithim:** Nombre del algoritmo usado en cada recomendación.
- **state:** Estado de cada recomendación, el cual puede tomar dos valores: 0 para notificar que no contiene recomendaciones del tipo correspondiente y 1 para notificar normalidad en la recomendación correspondiente.
- **package:** Nombre del paquete principal de la aplicación movil, este nombre es unico e identifica a cada aplicación movil en el GooglePlay.
- **name:** Nombre de la aplicación móvil.
- **rating:** Calificación resultante del algoritmo de recomendación correspondiente.
- **description:** Descripción corta de la aplicación móvil.
- **icon:** Url de la imagen del icono de la aplicación móvil.
- **banner:** Url de la imagen del banner de la aplicación móvil.

### Sincronización de Evaluación de la Recomendación

```
{ "type":"Evaluation",
  "user":"numero de celular",
  "data":[
            {"package":"nombre del paquete",
             "type":"tecnica de recomendacion",
             "datetime":"YYYY-MM-DD hh:mm:ss"},
            .
            .
            .
            {"package":"nombre del paquete n",
             "type":"tecnica de recomendacion",
             "datetime":"YYYY-MM-DD hh:mm:ss"}
          ]
}
```

**Figura 28.** JSON evaluación de recomendación.

Mensaje enviado desde la aplicación movil al servidor central, para sincronizar la evaluacion de la recomendación en el usuario, esta detalla que aplicación fue seleccionada dentro de todo el grupo de aplicaciones recomendadas. Entre la informacion transmitida se encuentran:

- **type:** Indentifica el tipo de mensaje.
- **user:** Numero telefonico con el cual se idenfica al usuario.
- **package:** Nombre del paquete principal de la aplicación movil, este nombre es unico e identifica a cada aplicación movil en el GooglePlay.
- **type:** Nombre del algoritmo de recomendación usado para la aplicación seleccionada.
- **datetime:** Fecha y hora en las cuales se selecciono la aplicación movil.

El servidor central retorna un **101** en caso de ser correcto la sincronización, por el contrario retorna un **404**.

### Sincronización de la evaluacion en la interfaz de recomendación

```
{"type":"evaluation_view",
 "user":"numero de celular de usuario",
 "data":[
        {"idrecommendation":"id de la Recomendacion",
         "totalpage":"Número de paginas",
         "datapage":[
                        {"page":"Indice de Pagina","duration":"duracion en milis"},
                        .
                        .
                        .
                        {"page":"Indice de Pagina N","duration":"duracion en milis"}
                   ]
        },
        .
        .
        .
        {"idrecommendation":"id de la Recomendacion N",
         "totalpage":"Número de paginas",
         "datapage":[
                        {"page":"Indice de Pagina","duration":"duracion en milis"},
                        .
                        .
                        .
                        {"page":"Indice de Pagina N","duration":"duracion en milis"}
                   ]
        },
}
```

**Figura 29.** JSON evaluación en la interfaz de recomendación.

Mensaje enviado desde la aplicación movil al servidor central, para sincronizar la evaluacion de la interfaz grafica de recomendación en el usuario, esta detalla los tiempos en los que permanece el usuario en cada pagina de recomendación. Entre la informacion transmitida se encuentran:

- **type:** Indentifica el tipo de mensaje.
- **user:** Numero telefonico con el cual se idenfica al usuario.
- **idrecommendation:** ID de la recomendación solicitada.
- **totalpage:** Numero total de paginas, el numero de paginas depende del numero de aplicaciones a recomendar.
- **page:** Indice de la pagina.
- **duration:** Duración de permanencia del usuario en la pagina correspondiente, esta duracion esta dada en milisegundos.

# ANEXO E

## Términos y Condiciones de VANILLA

# TERMINOS Y CONDICIONES DE VANILLA

El uso de nuestros servicios, está sujeto a los presentes Términos y Condiciones de uso (en adelante indistintamente, “los términos de uso”); te rogamos leerlos con detención.

**Declaración previa.**

VANILLA es un sistema que provee recomendaciones de aplicaciones móviles de forma personalizada, que proviene del cruce de información relacionada con las aplicaciones móviles consumidas, con la información que el Smartphone de cada usuario, sea capaz de brindar.

**Aceptación de los Términos de uso**

Al instalar la aplicación y/o utilizarla de cualquier forma, aceptas todos los términos y las condiciones contenidas en las presentes ("Condiciones de uso"), que también incorporan la Política de privacidad y licencias y demás reglas de operación, políticas y procedimientos que puedan ser publicados de vez en cuando.

**Acceso y registro**

Para acceder a la aplicación y disfrutar de nuestro servicio, bastará con instalar la aplicación e ingresar el número de celular, otorgando los permisos requeridos previos a la instalación.

**Contenido**

Para efectos de estos Términos y Condiciones de uso, el término "Contenido" incluye, entre otras cosas, cualquier información de, anuncios, información, datos, texto, fotografías, software, gráficos y características interactivas generadas, provistas o de cualquier otra forma suministradas por Vanilla. Todo el contenido emitido o enviado o cuyo origen sea el usuario es de su exclusiva responsabilidad.

**Contenido de VANILLA**

El Servicio incluye contenido específicamente provisto por Google Play del cual no tiene una vinculación ni relación directa con VANILLA.

**Contenidos del usuario**

Respecto de todo contenido que tú autorices proveer,  incluyendo, eventos de uso de aplicaciones, eventos de interacciones con tus contactos, información detectada por los sensores de tu dispositivo móvil. Sin constituir una lista taxativa sino que a modo ejemplar, cualesquiera palabras, imágenes, fotografías, nombre, características, comentarios, videos, sonidos, etc. Otorgas a VANILLA una licencia de uso con el fin de ejecutar los algoritmos que requieran dicha información.

VANILLA no garantiza la veracidad, propiedad, exactitud o cualquier otra característica del contenido suministrado o cuyo origen sea un usuario incluyendo su identidad.

Al eliminar el contenido no necesariamente queda eliminado de las plataformas de VANILLA las que pueden ser mantenidas por mandato legal o a voluntad de VANILLA en atención a la licencia que se señala en este párrafo.

**Uso de los servicios y plataforma**

Todo uso no autorizado del servicio está prohibido y eres exclusivamente responsable como usuario de lo que realices a través de él.

A título ejemplar no está autorizado el uso del servicio, el sitio web o las plataformas de VANILLA para: violar o infringir la propiedad intelectual ajena, de cualquier tipo o naturaleza, violar, infringir o afectar otros derechos exclusividad de terceros, afectar la privacidad, la imagen o intereses legítimos de terceros, usarlos para fines no autorizados, afectar las buenas costumbres, la moral o la ley, quedando el derecho discrecional de VANILLA de tomar todas las medidas conducentes que estime necesario en contra del usuario infractor.

No deberás: (i) hacer nada que imponga o pueda imponer (según lo determinado por VANILLA a su exclusiva discreción) una carga irrazonable o desproporcionadamente grande en la infraestructura de VANILLA (o de sus proveedores de terceros); (ii) interferir o intentar interferir con el buen funcionamiento del Servicio o de cualquier otra actividad realizada en el Servicio; (iii) evitar medidas que VANILLA pueda utilizar para impedir o restringir el acceso al Servicio (u otros sistemas informáticos o redes conectadas al Servicio); (iv) ejecutar cualquier forma de autorespuesta o "spam" en el Servicio; (v) recoger o extraer Contenido del Servicio o (vi) proceder de alguna forma en violación de las directrices y las políticas de VANILLA.

No debes (directa o indirectamente): (i) descifrar, descompilar, desensamblar, realizar ingeniería inversa o intentar derivar un código fuente o ideas subyacentes o algoritmos de cualquier parte de la aplicación (incluidos entre otros, cualquier aplicación o widget), excepto que en cierta medida las leyes aplicables prohíban específicamente dicha restricción, (ii) modificar, traducir o crear trabajos derivados de cualquier parte del Servicio o (iii) copiar, alquilar, arrendar, distribuir o transferir alguno de los derechos que recibes en el presente documento. Deberás cumplir con todas las normas y leyes locales, estatales, nacionales e internacionales.

VANILLA se reserva el derecho de (i) quitar, suspender, editar o modificar el Contenido a su discreción. VANILLA también se reserva el derecho de acceder, leer, preservar y revelar cualquier información que VANILLA razonablemente crea necesaria para (i) cumplir con las leyes, normas, procesos legales o requerimientos gubernamentales, (ii) hacer cumplir estas Condiciones de uso, incluida la investigación de posibles violaciones a las mismas, (iii) detectar, prevenir, o de cualquier otra forma resolver asuntos relacionados con el fraude, la seguridad o problemas técnicos, (iv) responder a solicitudes de asistencia de los usuarios o (v) proteger los derechos, propiedad o seguridad de VANILLA, sus usuarios y el público.

**Terminación**

Se reserva el derecho a los administradores de la aplicación, de suspender el acceso y/o servicio en cualquier momento, con o sin motivo, con o sin previo aviso, con efecto inmediato, lo que puede causar la pérdida y destrucción de toda la información asociada con tu membresía. Si deseas cancelar tu servicio, puedes hacerlo desinstalando la aplicación. Todas las disposiciones de estas Condiciones de uso que por su naturaleza deberían

sobrevivir a la terminación seguirán vigentes, incluyendo, entre otras cosas, las disposiciones de propiedad, renuncias de garantía, indemnización y limitaciones de responsabilidad.

**Exención de garantía**

Salvo en la medida requerida por ley, VANILLA no tiene ninguna relación especial o deber fiduciario contigo. Reconoces que VANILLA no tiene control ni está obligado a tomar medidas con respecto a: los usuarios que acceden a la aplicación; el Contenido al que accedes a través de la Aplicación; los efectos que pueda causarte el Contenido; cómo se pueda interpretar o utilizar el Contenido o qué medidas puedas tomar como resultado de haber estado expuesto al Contenido.

Eximes a VANILLA de toda responsabilidad por haber adquirido o no Contenido a través de la aplicación. El Servicio puede dirigirte a aplicaciones que contengan información que algunas personas podrían considerar ofensiva o inadecuada. VANILLA no hace declaraciones respecto del Contenido incluido o accesible a través del Servicio, y VANILLA no se hace responsable por la exactitud, cumplimiento de los derechos de autor, legalidad o decencia del material incluido o accedido a través del Servicio.

VANILLA, Y SUS DIRECTORES, EMPLEADOS, AGENTES, REPRESENTANTES, PROVEEDORES, SOCIOS Y PROVEEDORES DE CONTENIDO NO GARANTIZAN QUE: (A) EL SERVICIO SERÁ SEGURO O ESTARÁ DISPONIBLE EN CUALQUIER MOMENTO O LUGAR; (B) TODO DEFECTO O ERROR SERÁ CORREGIDO.

**Modificación de las Condiciones de uso**

VANILLA se reserva el derecho, a su entera discreción, de modificar o sustituir cualquiera de estas Condiciones de uso, o de modificar, alterar, suspender o interrumpir el Servicio (incluida entre otras cosas, la disponibilidad de alguna característica, base de datos o contenido) en cualquier momento mediante la publicación de un aviso en el Sitio o mediante el envío de un aviso a través del Servicio o vía email.

**Sensores e información que proveas**

Nos permitirás utilizar la información dada por los sensores de tu teléfono móvil, eventos de tu consumo, eventos de tus interacciones con tus contactos y la información que entregues de forma explícita (estado de ánimo, localización relativa, entre otros.) particularmente para la ejecución del servicio, como se ha suscrito en la sección de las licencias. Si por alguna razón, alguna vez decides desinstalar VANILLA, tu información no será eliminada de nuestras bases de datos, debido a que es necesaria para un óptimo desempeño del sistema. Sin embargo, dicha información NO será publicada ni compartida con tus contactos, con otros usuarios o terceros.

**Sensores**

VANILLA te permitirá determinar que sensores de tu teléfono móvil estarán disponibles o no, por lo que tendrás el control sobre qué es lo que podremos censar.

**Interacciones**

Nos permitirás recolectar la información relacionada con tu línea telefónica, lista de contactos y las interacciones que hayas mantenido con ellos, incluyendo, fechas de llamadas, fechas

de sms's, línea de destino de llamadas o sms's. EN NINGÚN CASO Y BAJO NINGUNA CIRCUSNTANCIA, SE GRABARAN LAS LLAMADAS QUE HAGAS O RECIBAS, O SE ACCEDERÁ AL CONTENIDO DE LOS SMS'S REGISTRADOS EN TU TELÉFONO.

**Ley aplicable**

Los presentes términos se regirán por la ley Colombiana fijándose como domicilio la comuna y ciudad de Popayán.

Notificaciones y procedimiento para formular reclamaciones por presunta infracción de los derechos de propiedad intelectual.

Las reclamaciones por infracción de derechos de propiedad intelectual deberán enviarse al siguiente correo electrónico vanillarec@gmail.com

Para procesar los reclamos Ud debe cumplir con las siguientes condiciones que establece la ley aplicable:

a) Enviar el aviso de la supuesta infracción en forma electrónica o de otra forma escrita del titular de los derechos o de su representante;

b) El titular de los derechos o su representante deberá tener domicilio o residencia en Colombia y, en su caso, contar con poder suficiente para ser emplazado en juicio, en representación del titular;

c) Se identifiquen los derechos supuestamente infringidos, con indicación precisa de la titularidad de éstos y la modalidad de la infracción;

d) Se identifique el material infractor y su localización en las redes o sistemas del prestador de servicios a quien se envía la comunicación, a través del URL o sus equivalentes, y

e) El reclamo debe contener datos que permitan al prestador de servicios identificar al usuario proveedor del supuesto material infractor.

Reserva de derechos de autor y marcas comerciales.

Todo el contenido de la Aplicación Móvil o Sitio Web puede estar protegido por derechos de autor en lo que fuere aplicable.

# ANEXO F

## Articulo

# ANEXO G

## Icono y Banner de VANILLA

## Icono y Banner de Vanilla

A continuación se describen el diseño utilizado para el icono y para el banner de la aplicación en el Google Play.

| Icono |
| --- |
|  |
| **Banner** |
|  |