UNIVERSIDAD DE LAS CIENCIAS INFORMÁTICAS

FACULTAD 1

Trabajo de Diploma para Optar por el Título de
Ingeniero en Ciencias Informáticas

Subsistema de detección de roles de usuario dentro de comunidades de interés de Twitter.

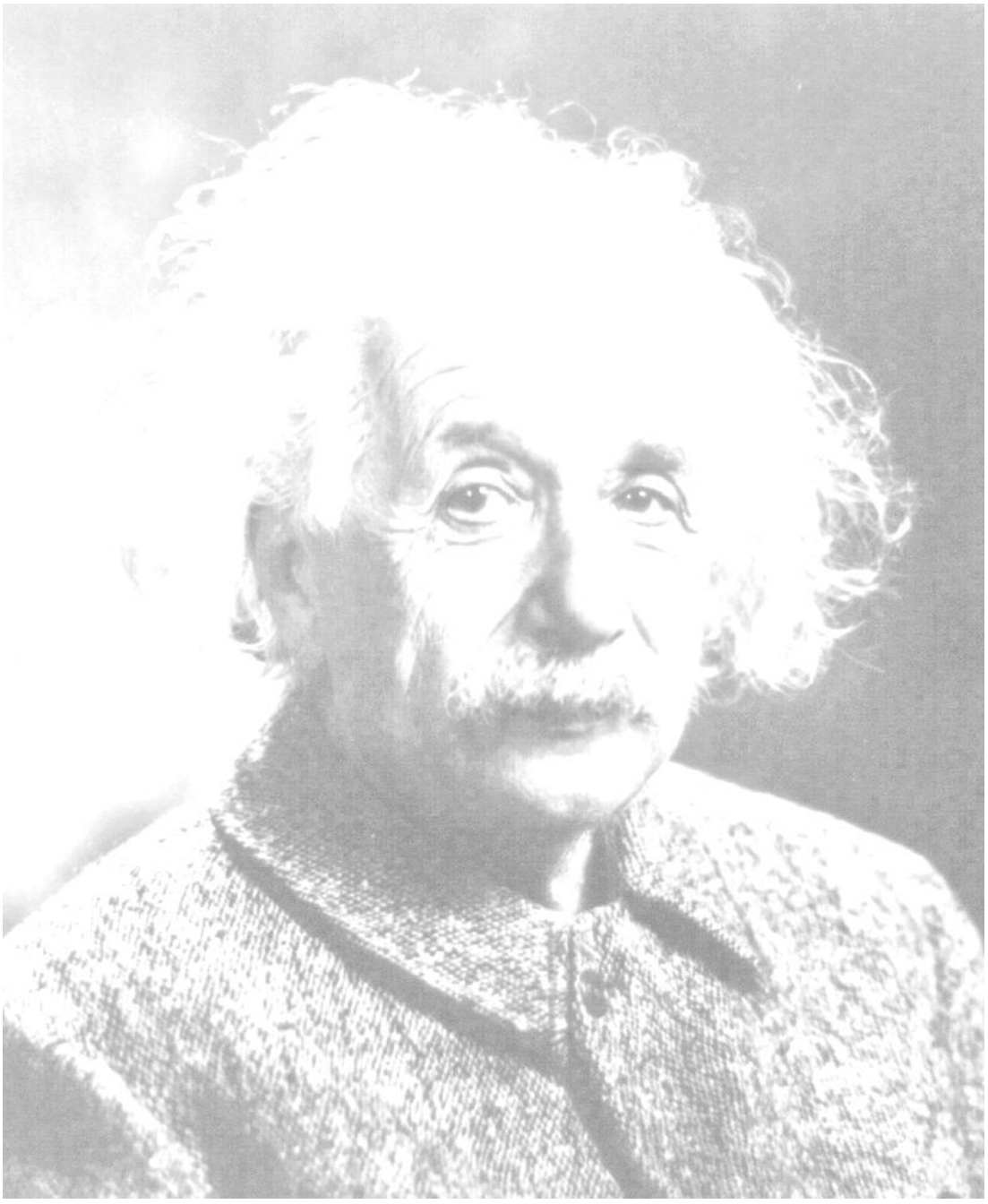
Autor: Carlos Julio Figueiras Carrera.

Tutores: Ing. José Gabriel Espinosa Ramirez.

Ing. Odelkis Rodriguez Irsula.

La Habana

Diciembre de 2014



Soy lo suficientemente artista como para dibujar libremente sobre mi imaginación. La imaginación es más importante que el conocimiento. El conocimiento es limitado. La imaginación circunda el mundo.

-Albert Einstein

DECLARACIÓN DE AUTORÍA

Declaro ser el único autor de la presente tesis y concedo a la Universidad de las Ciencias Informáticas los derechos patrimoniales de la misma, con carácter exclusivo.

Para que así conste firmo la presente a los ____ días del mes de _____ del año _____.

Carlos Julio Figueiras Carrera

_____

Firma del Autor

Ing. José G. Espinosa Ramirez

_____

Firma del Tutor

Ing. Odelkis Rodriguez Irsula

_____

Firma del Tutor

A mis padres por hacer de mí la persona que soy; por ser los máximos responsables de mi formación, por su dedicación y comprensión.

A mis tutores por su tiempo y ayuda.

A Rubi, Miguel y Luis por todo el cariño durante estos años, por estar ahí siempre que lo necesité.

A toda mi familia que siempre estuvo pendiente de mí.

A mis viejos amigos de estudios Raúl y Ángel que a pesar de las distancias siempre estuvieron presentes.

A todos mis compañeros de estudio y convivencia durante estos 5 años por compartir tantos momentos conmigo.

A todos los entrenadores del movimiento ACM-ICPC en la UCI que de una forma u otra contribuyeron a mis resultados.

A mis compañeros del movimiento ACM-ICPC en la UCI que compitieron junto a mí depositando su confianza.

RESUMEN

Con el desarrollo de Internet, las redes sociales se han convertido en catalizadores para movimientos sociales de diferentes tendencias políticas, en especial el servicio de Twitter, en el cual los usuarios expresan sus intereses entorno a diversos temas formando comunidades que, de forma explícita o implícita, permiten que sus usuarios se desenvuelvan en determinados roles. La identificación de los usuarios y sus roles, es una tarea del campo del análisis de redes sociales y es un elemento crítico a la hora de diseñar campañas de divulgación en Twitter. El análisis de redes sociales cada vez más se utiliza como una manera estructurada de estimar el grado de relación informal entre las personas. Es un área del conocimiento que puede verse limitado por las cantidades enormes de información acumulada y uno de sus primeros enfoques es medir el poder, la influencia, u otras características individuales de las personas basado en sus patrones de conexión. Para ello se utilizan métodos de medición conocidos como "métricas de centralidad". El Centro de Ideoinformática (CIDI), enfrenta el reto de difundir la verdad sobre Cuba y defender sus principios y causas haciendo uso del servicio Twitter, por lo que necesita integrar y optimizar sus actividades sobre la red social. Enfocado en ese sentido, y luego de realizar un análisis sobre las principales herramientas para análisis de redes sociales, se propone desarrollar un subsistema propio, usando tecnologías web, para la detección de roles de usuario en comunidades de interés.

Palabras claves: análisis de redes sociales, comunidades, métricas, redes sociales, roles, Twitter.

INTRODUCCIÓN

El hombre es un ser social por naturaleza que constantemente incorpora experiencias y establece lazos con el entorno y sus semejantes formando un sistema complejo conocido como red social. Así, (Lozares, 1996) define a las redes sociales como “un conjunto bien delimitado de actores -individuos, grupos, organizaciones, comunidades, sociedades globales, etc.- vinculados unos a otros a través de una relación o un conjunto de relaciones sociales”.

Las redes sociales, con el desarrollo de Internet, adquieren una nueva dimensión y se está viviendo un importante auge, que en poco tiempo ha provocado que el volumen de participación de los internautas se haya multiplicado exponencialmente.

Ejemplo de ello es Twitter, red social de microblogging que en pocos años ha evolucionado más allá de ser una simple red social; se ha transformado en un barómetro de los estados de opinión, en una importante plataforma para los movimientos sociales, en una sala de prensa global y en un medio para la globalización.

En el seno de la red social Twitter, los usuarios expresan sus opiniones en torno a diversos temas de interés formando comunidades. La detección de estas comunidades tiene como objetivo identificar, utilizando la topología de un grafo, grupos de usuarios densamente conectados entre sí y que compartan características comunes. El principal problema es que no existe una definición de comunidad universalmente aceptada más allá de la noción de que debe haber más enlaces entre los usuarios de una comunidad que con los usuarios de otras (Fortunato, 2010).

Cada una de estas comunidades, de forma explícita o implícita, permiten que sus usuarios se desenvuelvan en determinados roles en dependencia de la función que ejercen y la interacción con el resto de los usuarios. La identificación de los usuarios y sus roles dentro de una determinada comunidad es un elemento crítico a la hora de diseñar campañas de divulgación en Twitter, mejorar la comprensión de la dinámica interna y potenciar a la red social como un canal de comunicación institucional eficiente.

Entender las redes sociales y las comunidades virtuales es vital en el mundo actual. Sin embargo, la gran cantidad de información disponible hoy día en Internet hace a menudo imposible poner las cosas en contexto.

El Centro de Ideoinformática (CIDI), centro adjunto a la Facultad 1 de la Universidad de las Ciencias Informáticas (UCI), haciendo uso del servicio Twitter, enfrenta el reto de realizar acciones de vital importancia para la Universidad tales como; condenar el bloqueo financiero que mantiene los EE.UU. contra Cuba y los tuitazos gigantes por la liberación de los cinco y contra el terrorismo. Actualmente, casi todo este trabajo en el centro se realiza de forma manual repercutiendo negativamente sobre la ecuación «Resultados» / «Tiempo», lo que a opinión de los especialistas del centro se traduce en; mayor necesidad de personal, mayor esfuerzo, menor efectividad y afectación sobre otras tareas.

Por tal motivo, se necesita integrar y automatizar las actividades del centro sobre la red social de tal forma que se eviten redundancias y se aprovechen mejor los recursos tecnológicos y humanos involucrados. Enfocado en ese sentido, CIDI está dando los primeros pasos en la construcción de una plataforma propia para la interacción con la red social Twitter y otros canales de comunicación relacionados llamada “Plataforma para la Red Social Twitter” (PRST). De la forma que se realiza el trabajo hoy día en el centro, resulta prácticamente imposible realizar los análisis propuestos, por lo que se valora el posible desarrollo de un subsistema web para la detección de roles de usuario dentro de comunidades de interés de Twitter para su posterior integración a la PRST.

Por lo antes planteado, respondiendo a las necesidades descritas, se determina como **problema a resolver** la siguiente interrogante: ¿Cómo identificar los roles de usuario dentro de una comunidad de interés de Twitter?

Para ello se identifica como **objeto de estudio,** los sistemas de interacción y análisis de comunidades en redes sociales, centrándose en el **campo de acción** de la identificación de roles de usuario en la red social Twitter. Para dar respuesta a la interrogante, se traza como **objetivo general** desarrollar una aplicación para la identificación de los roles de usuario dentro de comunidades de interés de Twitter, desglosándose en los **objetivos específicos** que a continuación se enumeran:

1. Valorar el marco teórico conceptual y el estado del arte respecto a las tecnologías actuales para el desarrollo de herramientas para la interacción con la Interfaz de Programación de Aplicaciones (*Application Programming Interface*, API por sus siglas en inglés) de Twitter.

2. Diseñar una aplicación que permita la interacción indirecta con Twitter y la identificación de los roles de usuario dentro de comunidades de interés.
3. Implementar una aplicación que permita la interacción indirecta con Twitter y la identificación de los roles de usuario dentro de comunidades de interés.
4. Validar el correcto funcionamiento de la aplicación.

Como guía de la investigación se define como **idea a defender** que el desarrollo de una aplicación, con el uso de tecnologías web, que identifique los roles de usuario en comunidades de interés, permitirá automatizar el trabajo de CIDI sobre la red social Twitter.

En el transcurso de la investigación se utilizan **métodos de investigación científicos**; los métodos teóricos siguientes:

Histórico-Lógico: Permite entender el surgimiento, las características actuales, conceptos, términos y vocabularios propios de las redes sociales específicamente de la red social Twitter.

Analítico-Sintético: Se emplea con el fin de hacer un correcto, amplio y sintetizado análisis de la bibliografía existente posibilitando extraer las ideas esenciales y relevantes relacionadas con el objeto de estudio. A través de su empleo se definen los conceptos y principios relacionados con la red social Twitter.

Modelación: Se utiliza con el objetivo de reproducir la interacción de los objetos en la vida real mediante la creación de modelos.

El presente trabajo investigativo se desglosa en tres capítulos fundamentales abordando los siguientes temas:

CAPÍTULO 1: INTRODUCCIÓN AL ANÁLISIS DE REDES SOCIALES. En este capítulo se incluye el estudio del estado del arte de interacción directa o indirecta con Twitter y la detección de roles en redes sociales. Se muestra un estudio de las metodologías que se tuvieron en cuenta para guiar el proceso de desarrollo, lenguaje de programación, lenguaje de modelado, marco de trabajo, entorno de desarrollo, gestor de bases de datos y herramientas de ingeniería de software.

CAPÍTULO 2: CARACTERÍSTICAS Y DISEÑO DEL SUBSISTEMA. En este capítulo se definen las principales características que debe cumplir el software a desarrollar en términos de requisitos funcionales y no funcionales los cuales son representados mediante diagramas de casos de uso en los que se describen las relaciones existentes de los actores que se involucran.

CAPÍTULO 3: IMPLEMENTACIÓN Y PRUEBAS. En este capítulo se implementan las funcionalidades descritas con anterioridad y se describen las pruebas funcionales y no funcionales con sus resultados.

Tareas de investigación

1. Valoración del estado del arte de la interacción con Twitter.
2. Valoración del estado del arte de la detección de roles de usuario en comunidades de interés de Twitter.
3. Definición de las metodologías a utilizar en el desarrollo de la aplicación.
4. Identificación de las necesidades de abstracción, requisitos funcionales y no funcionales.
5. Selección de las bibliotecas y lenguaje de programación.
6. Diseño e implementación de las funcionalidades para la detección de roles de usuario en comunidades de interés y otras.
7. Diseño e implementación de los mecanismos de interacción indirecta con Twitter.
8. Documentación de los artefactos generados durante los procesos de análisis, diseño e implementación de la aplicación.
9. Aplicación de las pruebas de funcionalidad a la aplicación.

CAPÍTULO 1: INTRODUCCIÓN AL ANÁLISIS DE REDES SOCIALES

En este capítulo se incluye el estudio del estado del arte de interacción directa o indirecta con Twitter y la detección de roles en redes sociales. Se muestra un estudio de las metodologías que se tuvieron en cuenta para guiar el proceso de desarrollo, lenguaje de programación, lenguaje de modelado, marco de trabajo, entorno de desarrollo, gestor de bases de datos y herramientas de ingeniería de software.

1.1. Análisis de redes sociales

El análisis de redes sociales (ARS) cada vez más se utiliza como una manera estructurada de estimar el grado de relación informal entre las personas, los equipos, departamentos, o incluso las organizaciones. El ARS hace visibles estos patrones de interacción de otro modo invisibles a fin de identificar grupos importantes con el objetivo de facilitar la colaboración eficaz (Umadevi, 2013).

Uno de los primeros enfoques del ARS es medir el poder, la influencia, u otras características individuales de las personas basado en sus patrones de conexión. Para ello se utilizan métodos de medición conocidos como "métricas de centralidad".

Una métrica de centralidad puede ser considerada a nivel de nodo o nivel de enlace, y por lo general se interpreta como la "conexión" o la importancia de un nodo o enlace. La centralidad mide según un cierto criterio la contribución de un nodo según su ubicación en la red, independientemente de si se esté evaluando su importancia, influencia, relevancia o prominencia. Téngase en cuenta que las métricas no determinan si un usuario aplica a cierto rol, sino que brinda un valor indicando en qué medida el usuario cumple con un rol, pues dicho valor puede variar en dependencia del tamaño de la red.

Las métricas de centralidad se pueden agrupar en dos categorías: métricas radiales y mediales (Borgatti, y otros, 2006). Las primeras toman como punto de referencia un nodo dado que inicia o termina recorridos por la red, mientras que las segundas toman como referencia los recorridos que pasan a través de un nodo dado (Jimeng, y otros, 2011). Las métricas radiales a su vez se pueden clasificar en métricas de volumen y de longitud, según el tipo de recorridos que consideran. Las primeras miden el volumen (o el número) de recorridos limitados a dicha longitud prefijada, en tanto que las segundas

miden la longitud de los recorridos necesarios para alcanzar un volumen prefijado (Jimeng, y otros, 2011).

Desde la formulación realizada por (Bavelas, 1948), se han propuesto diversas métricas de centralidad de un nodo. Existen cuatro de estas métricas que son ampliamente usadas en análisis de redes sociales; Centralidad de Grado, Centralidad de Intermediación, Centralidad de Cercanía, y Centralidad Eigenvector. La primera y la última son medidas radiales de volumen. La segunda es una medida radial de longitud, y la tercera una medida medial (Jimeng, y otros, 2011). Adicionalmente, se puede distinguir entre las medidas absolutas de centralidad, que indican un valor no comparable y aquellas que están normalizadas, denominadas medidas relativas de centralidad.

1.1.1. Centralidad de Grado

La Centralidad de Grado es la primera y más simple de las medidas de centralidad. Corresponde al número de enlaces que posee un usuario con los demás. Para grafos dirigidos, como es el caso de una comunidad Twitter, se pueden definir dos medidas de centralidad de grado diferentes; grado de entrada y grado de salida. En el sentido de relaciones interpersonales, el primero puede interpretarse como una medida de popularidad, mientras que el segundo como una de sociabilidad, aunque las interpretaciones pueden ser muchas (Tsvetovat, 2011).

En la imagen (ver Figura 1.1), se muestra una red de cinco usuarios con varios enlaces entre ellos. Calculando ambos valores de grado, quedaría de la siguiente forma (ver Tablas 1.1 y 1.2).

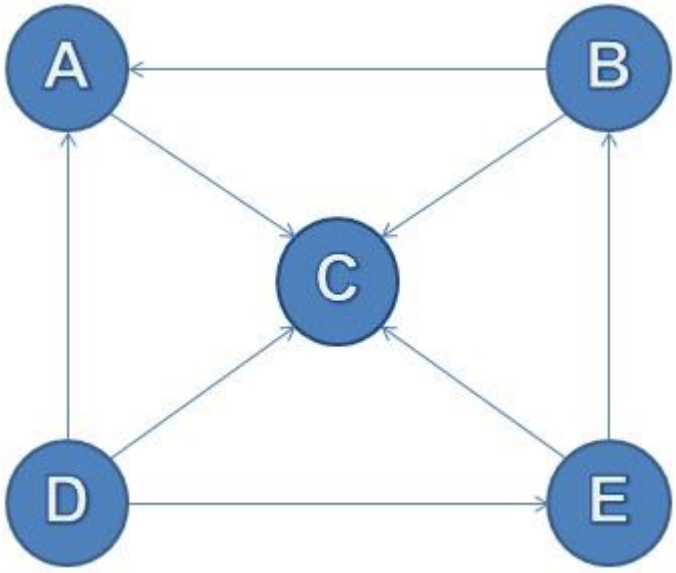


Figura 1.1. Ejemplo de red de usuarios (Elaboración propia).

En la Tabla 1.1 se muestra el resultado de calcular el grado de entrada para el caso de estudio ejemplificado en la figura anterior. El usuario **C** destaca por tener el mayor valor, por lo que puede señalarse como el más popular.

A	*B*	*C*	*D*	*E*
2	*1*	*4*	*0*	*1*

Tabla 1.1. Ejemplo de cálculo del grado de entrada para el caso de estudio (Elaboración propia).

Así mismo, en la Tabla 1.2, se muestra el resultado de calcular el grado de salida. En este caso, destaca el usuario **D** como el de mayor valor, por lo que puede señalarse como el más sociable.

A	*B*	*C*	*D*	*E*
1	*2*	*0*	*3*	*2*

Tabla 1.2. Ejemplo de cálculo del grado de salida para el caso de estudio (Elaboración propia).

Existen otras generalizaciones de esta medida de centralidad, tales como: **Centralidad de Camino-K**, **Centralidad de Katz**, **Centralidad de Bonacich** y **Centralidad de**

Hubbell. Estas últimas, permiten obtener resultados muy similares en una complejidad computacional igual o peor en algunos casos, por lo que no se tuvieron en cuenta para su aplicación.

1.1.2. Centralidad de Intermediación

La Centralidad de Intermediación se basa en el número de caminos más cortos que pasan a través de un usuario. Los usuarios con una alta intermediación juegan el rol de conectar a los diferentes grupos; son los agentes y conectores que mantienen juntos a los demás. Los individuos con alta intermediación son los pivotes en el flujo de conocimiento de la red. Cuando un usuario con gran valor de intermediación es retirado, tiende a ocurrir un incremento en la distancia típica entre los demás (Tsvetovat, 2011). En el ejemplo (ver Figura 1.2), **D** es un ejemplo de usuario intermediario en la red representada.

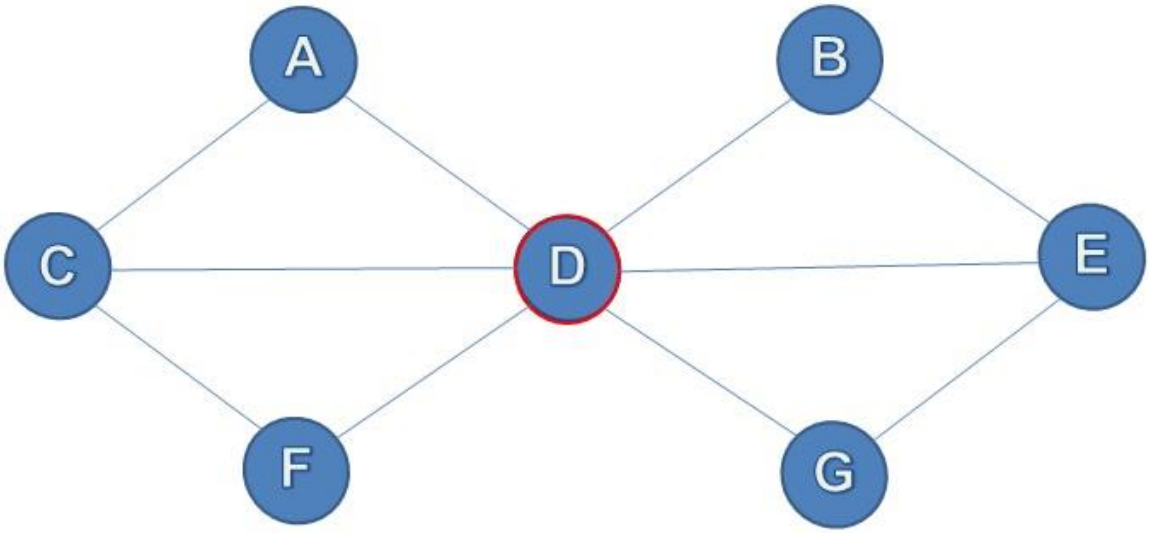


Figura 1.2. El usuario **D** es el de mayor valor de intermediación en la red (Elaboración propia).

El siguiente algoritmo describe de manera sencilla y simplificada en un lenguaje natural cómo calcular esta métrica (Tsvetovat, 2011).

1. Computar el camino más corto entre cada par de nodos de la red usando el algoritmo de Dijkstra.
2. Para cada nodo **N** en la red, contar el número de caminos más cortos a los que pertenece.

3. Normalizar los valores para obtener un resultado en el rango **[0,1]**.

Como alternativa, (Newman, 2005) propone una versión basada en considerar caminos aleatorios de la red, y no exclusivamente los más cortos. La idea es tomar en cuenta todos los caminos posibles, y calcular la medida de acuerdo a los elegidos aleatoriamente. El principal problema consiste en que posee una complejidad computacional significativamente superior a la Intermediación y para redes muy grandes, como es el caso, no aplica.

1.1.3. Centralidad de Cercanía

Expresa la distancia de un nodo con respecto a todos los demás en la red centrándose en la distancia geodésica. La cercanía puede ser considerada como una medida de cuánto tiempo tomará para que la información se propague desde un nodo determinado a los otros nodos en la red.

El algoritmo para su cálculo es simple aunque posee una complejidad computacional alta y se resume en los siguientes pasos: (Tsvetovat, 2011)

1. Computar el camino más corto entre cada par de nodos de la red usando el algoritmo de Dijkstra.
2. Para cada nodo **N** en la red:

    2.1. Calcular la distancia promedio al resto de los nodos de la red.

    2.2. Calcular el inverso del valor obtenido en el paso anterior.

El Resultado es un valor normalizado en el intervalo **[0,1],** donde a mayor valor, mayor cercanía.

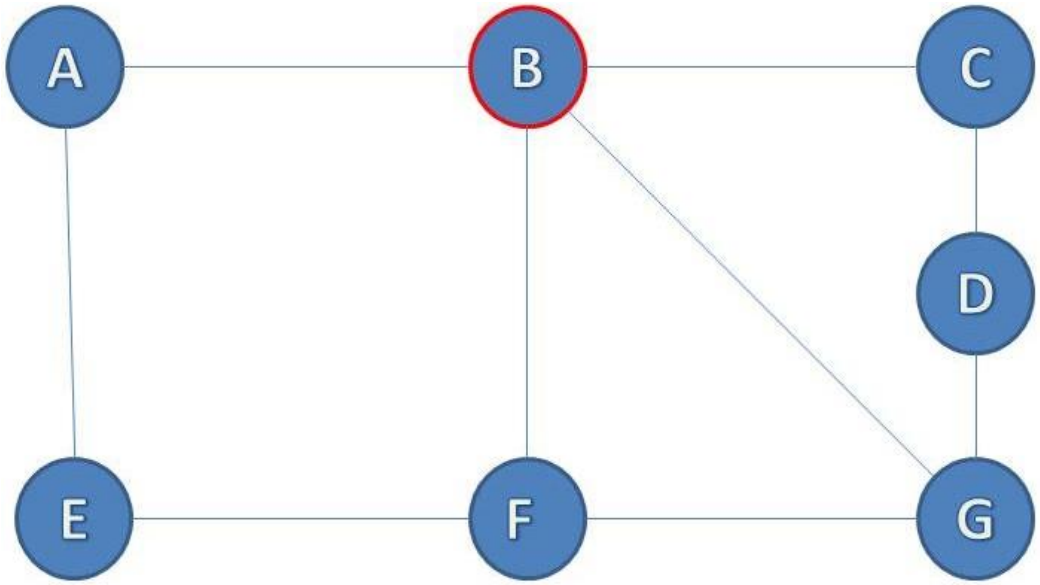


Figura 1.3. El usuario **B** es el de mayor valor de cercanía en la red (Elaboración propia).

Para el ejemplo (ver Figura 1.3), al calcular la cercanía para cada uno de los usuarios (ver Tabla 1.3), se puede observar al usuario **B** como el de mayor valor. Una posible interpretación del resultado es que **B** será el usuario con mayor efectividad para propagar un mensaje al resto de la red.

A	*B*	*C*	*D*	*E*	*F*	*G*
0.55	*0.75*	*0.55*	*0.50*	*0.50*	*0.66*	*0.66*

Tabla 1.3. Cálculo de cercanía para la red de la Figura 1.3 (Elaboración propia).

1.1.4. Centralidad de Eigenvector

La Centralidad de Eigenvector es una medida de la influencia que tiene un usuario en una red. Asigna puntuaciones relativas a todos los usuarios de la red basado en el principio de que las conexiones a los usuarios de alta puntuación contribuyen más a su puntuación que conexiones a los de baja puntuación. En general, las conexiones con las personas que son en sí mismos influyentes van a prestar a una persona más influencia que las conexiones a las personas menos influyentes (Tsvetovat, 2011).

El algoritmo para su cálculo es simple aunque posee una complejidad computacional alta y se resume en los siguientes pasos:

1. Asignar un valor de 1 a todos los nodos ($\mathbf{v_i = 1}$ para todo nodo **i** en la red).

2. Recalcular los valores de cada nodo como una suma con pesos de la centralidad de todos sus nodos vecinos.

$$v_i = \sum_{j \in N} x_{i,j} * v_j$$

3. Normalizar **v** dividiendo cada valor por el mayor valor.

4. Repetir pasos 2 y 3 hasta que los valores de **v** dejen de cambiar.

Aunque existen diversas alternativas para la Centralidad de Eigenvector tales como la **Centralidad Alfa** en la que los nodos están sujetos a distinta importancia dependiendo de factores externos (Bonacich, y otros, 2001), se usará **PageRank** en su lugar, pues es un algoritmo que permite obtener resultados similares, pero escala mucho mejor para redes muy grandes y que cambian con el tiempo como es el caso.

1.1.5. PageRank

Fue desarrollado originalmente para las páginas web de indexación, pero se puede aplicar a las redes sociales, así, siempre y cuando se trate de redes con una representación unidireccional, por ejemplo, una red de Twitter. PageRank es un proceso iterativo, que en un momento dado devolverá un resultado, pero luego de pocas iteraciones, la calidad de los resultados mejora grandemente. En la primera iteración, se obtendrá un resultado muy impreciso, este resultado mejorará rápidamente con iteraciones hasta alcanzar un punto de estabilidad (convergencia) o hasta que se decida. PageRank es similar a la Centralidad Eigenvector, pero el algoritmo escala mucho mejor para redes muy grandes y que cambian con el tiempo (Tsvetovat, 2011).

Como resultando se tendrá un valor en la escala **[0,1]** y representa la probabilidad de que una persona sigua un enlace llegando hasta una página o persona en particular. Una probabilidad de 0.5 es comúnmente interpretada como un 50% de chance de que ocurra un evento. Sin embargo, un PageRank de 0.5 significa que existe un 50% de chance que una persona presionando un enlace al azar vaya directamente a la página con 0.5 PageRank. Los pasos para su cálculo son sencillos (Tsvetovat, 2011):

1. Inicialmente asigna igual valor de probabilidad a todos los nodos de la red (ver Figura 1.4).

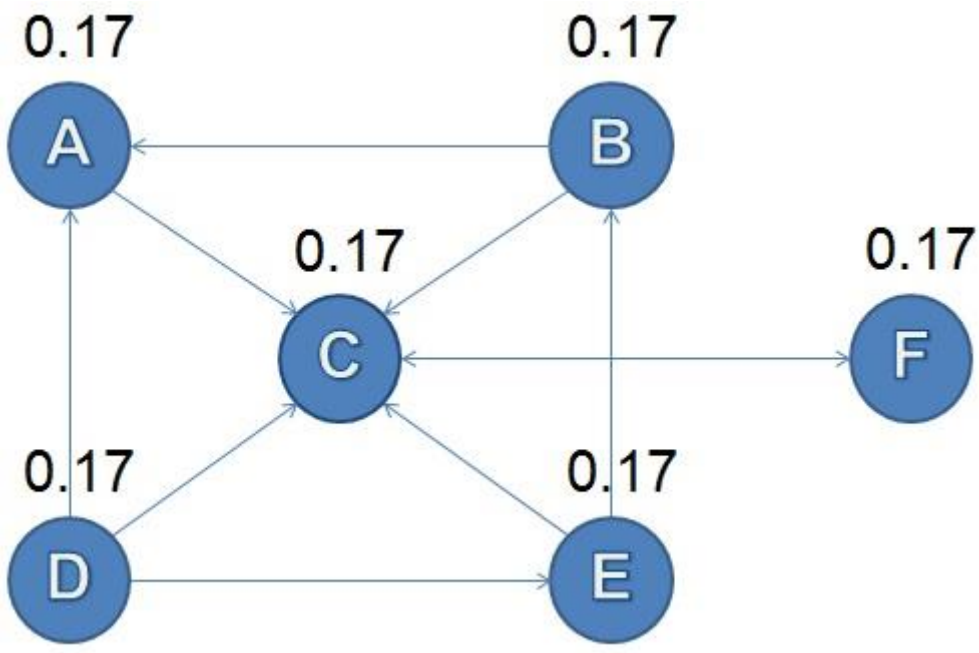


Figura 1.4. Asignación inicial para una red de seis usuarios (Elaboración propia).

2. En caso general, para todo nodo **N** de la red, PageRank será calculado por la función **Pr(N) = ∑ (Pr(i) / out_degree(i)),** para todo nodo **i** para el que existe al menos una arista hacia nodo **N**, donde **out_degree(i)** es el grado de salida del nodo **i** (ver Figura 1.5).

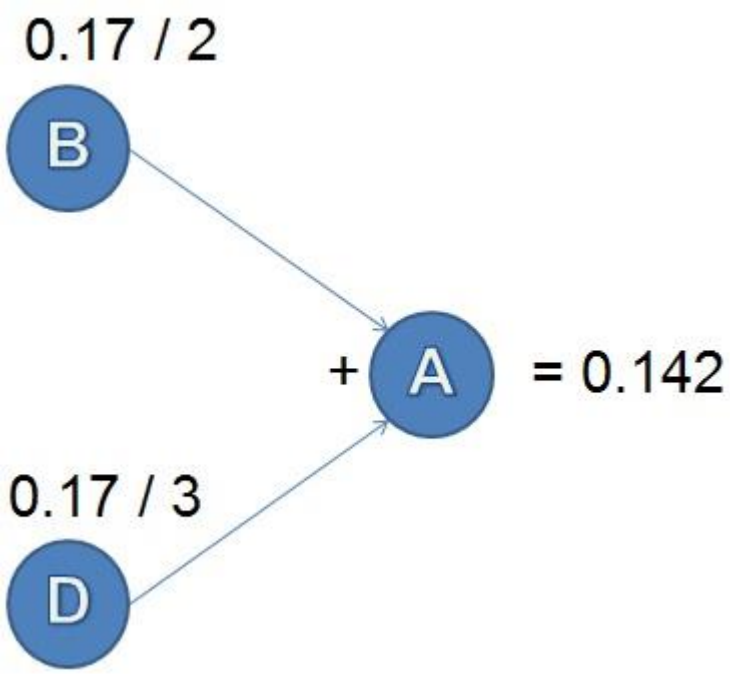


Figura 1.5. Cálculo del PageRank para usuario **A** en la 1ra iteración (Elaboración propia).

3. Repetir a partir del paso 1 hasta que el valor estabilice (ver Figura 1.6).

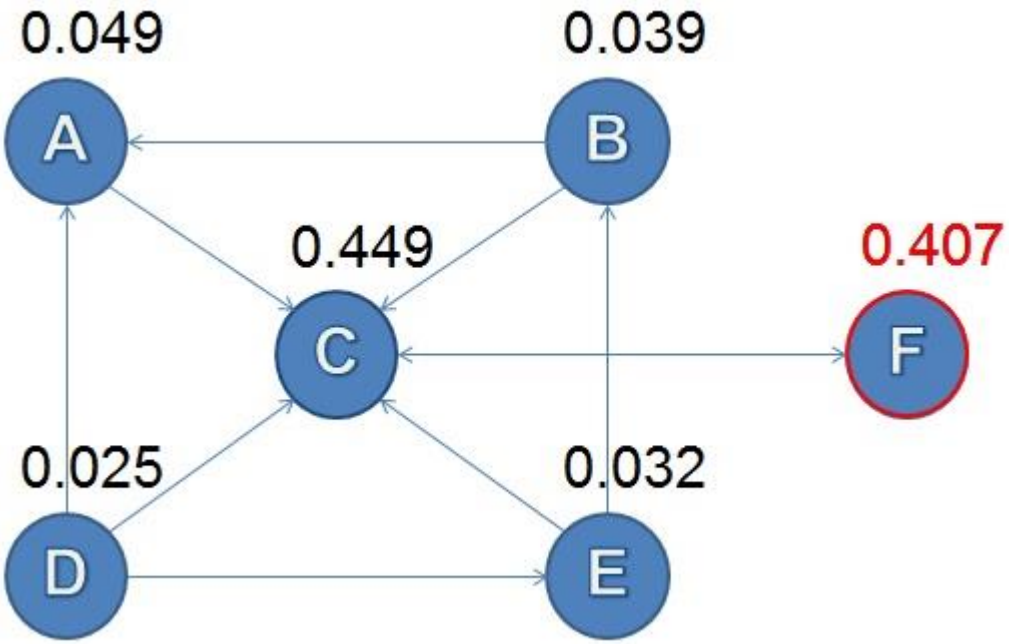


Figura 1.6. PageRank resultante luego de estabilizar los valores (Elaboración propia).

En la Figura 1.6, haciendo una interpretación del resultado final, se puede apreciar como el usuario **F** sólo tiene vínculo con **C**, pero dado que **C** es el de mayor influencia, convierte a **F** en influyente. Para esta red, el usuario **F** al no tener enlaces con el resto de los usuarios, para el resto de las métricas tendría valores despreciables, lo que no ocurre para PageRank.

1.2. Referentes teóricos-metodológicos

Para el ARS existe una gran variedad de software que permiten realizar análisis mediante valores numéricos o representaciones visuales, facilitando resultados tanto cuantitativos como cualitativos de las métricas requeridas. Aunque la mayoría de ellos responden a los mismos propósitos, están orientados a distintos tipos de redes por sus características estructurales y tamaño, no todos ofrecen las mismas funcionalidades y algunos poseen licencia comercial. Entre dichos software, se encuentran **Node XL Excel 2007**, **Pajek** y **UCINET,** los cuales se detallan a continuación.

Node XL Excel 2007. Es una aplicación complementaria a Excel que permite la interacción directa con Twitter. Construye una red visual a partir de ciertos parámetros de interés partiendo de los mensajes y comentarios realizados en la red social. Permite visualizar, filtrar y utilizar métricas de centralidad conocidas a la red (L. Hansen, y otros, 2011).

Permite importar fácilmente listas-redes no sólo de Twitter, sino también de Youtube, Flirck o Emails. Es gratuito.

Pajek. Es un software para Windows para análisis de grandes redes. Es gratuito y su uso se limita a fines no comerciales. Sus principales objetivos son soportar la abstracción mediante factorialización (recursiva) de grandes redes en otras de inferior tamaño que son susceptibles de ser tratadas con métodos más sofisticados, proporcionar herramientas potentes para visualizar estas redes e implementar una selección de algoritmos eficientes para análisis de grandes redes.

UCINET 6. Es un software diseñado para un ambiente Windows y es una de las herramientas computacionales con más difusión en el ARS. Es un paquete de herramientas que cumple diferentes roles, posee una alta manejabilidad, contiene el cálculo de los principales indicadores así como una aplicación de gráficos con muchas

opciones. Por otro lado no es gratuito, la licencia caduca, y aunque sigue funcionando se cierra cada vez que se realiza una operación (Borgatti, y otros, 2002).

Como se puede apreciar, son herramientas limitadas a entornos Windows y en algunos casos restringidos por licencias comerciales por lo que se hace necesaria la construcción de un sistema propio. Para ello se investigaron las principales bibliotecas para el análisis de redes sociales arrojando los siguientes resultados.

Gephi. Plataforma para la visualización interactiva y la exploración de todo tipo de redes, sistemas complejos y grafos dinámicos y jerárquicos. Permite importar, exportar, manipular, analizar, filtrar, representar, detectar comunidades y exportar grandes grafos y redes. Gephi está construido sobre la plataforma NetBeans y programado en Java y OpenGL. Se distribuye bajo una licencia dual GNU GPL v3 y Common Development and Distribution License (CDDL-1.0). Es, por tanto, una aplicación de código abierto y multiplataforma (Cherven, 2013).

igraph. Es una colección de herramientas para el análisis de redes sociales con énfasis en la eficiencia, la portabilidad y la facilidad de uso. Es de código abierto y libre, distribuido bajo licencia GPL v2 o superior. Actualmente se encuentra disponible en los lenguajes de programación GNU R, Python y C/C++. Posee algoritmos para grafos simples y análisis de redes, puede ser utilizado para grandes grafos muy bien y provee funciones para generar grafos aleatorios, visualización, cálculos de centralidad y detección de comunidades (igraph, 2013).

NetworkX. Es una biblioteca escrita en el lenguaje de programación Python para la creación, manipulación y el estudio de la estructura, dinámica y función de complejas redes. Con NetworkX se pueden gestionar redes en formatos de datos estándar y no estándar generando diferentes tipos de redes aleatorias y clásicas, analizar la estructura de la red, construir modelos de redes, diseñar nuevos algoritmos sobre redes y visualizarlas (Hagberg, y otros, 2014).

Entre los objetivos que persigue NetworkX se destacan: brindar una herramienta para el estudio de la estructura y dinámica de redes sociales, biológicas e infraestructura, convertirse en un entorno de desarrollo ágil para proyectos colaborativos y

multidisciplinarios, ser una interfaz estándar de programación y una implementación de grafo útil para muchas aplicaciones (Hagberg, y otros, 2014).

1.3. Métodos, herramientas y técnicas

Una de los pasos claves en el desarrollo, es definir correctamente los métodos y herramientas a utilizar, así como las técnicas para su uso, de ello dependerá en gran medida el éxito del proyecto.

1.3.1. Metodologías de desarrollo de software

Una metodología de desarrollo de software es un conjunto de pasos y procedimientos que deben seguirse para desarrollar software. Una metodología define cómo dividir un proyecto en etapas, qué tareas se llevan a cabo en cada etapa, qué restricciones deben aplicarse, qué técnicas y herramientas se emplean y cómo se controla y gestiona un proyecto (Hernando, 2009).

Para el proceso de desarrollo de software, existen dos tipos de metodologías, por un lado, se encuentran las conocidas como pesadas o tradicionales que centran su atención en llevar una documentación exhaustiva y en cumplir con un plan de proyecto definido en un inicio, pues se basan en la idea de que el éxito del producto se puede lograr si se tiene todo correctamente documentado.

Este tipo de metodología es más eficaz en la medida en que el proyecto a realizar sea mayor, pues requieren de gran organización. Algunas de sus principales desventajas son los altos costos al implementar un cambio y no ofrecer una buena solución para proyectos donde el entorno es volátil. Por otro lado están las metodologías ágiles que defienden la idea de que el proceso de desarrollo de software se centra en el software como tal y no en la documentación alrededor de este, por lo que definen solo la necesaria y de forma sencilla. La filosofía ágil se resume en las siguientes valoraciones:

Al individuo y las interacciones del equipo de desarrollo sobre el proceso y las herramientas. La gente es el principal factor de éxito de un proyecto software. Si se sigue un buen proceso de desarrollo, pero el equipo falla, el éxito no está asegurado; sin embargo, si el equipo funciona, es más fácil conseguir el objetivo final, aunque no se tenga un proceso bien definido. No se necesitan desarrolladores brillantes, sino desarrolladores que se adapten bien al trabajo en equipo. Así mismo, las herramientas (compiladores, depuradores, control de versiones, etc.) son importantes para mejorar el

rendimiento del equipo, pero el disponer más recursos que los estrictamente necesarios también pueden afectar negativamente.

En resumen, es más importante construir un buen equipo que construir el entorno. Muchas veces se comete el error de construir primero el entorno y esperar que el equipo se adapte automáticamente. Es mejor crear el equipo y que éste configure su propio entorno de desarrollo en base a sus necesidades (H. Canós, y otros, 2003).

Desarrollar software que funciona más que conseguir una buena documentación. Aunque se parte de la base de que el software sin documentación es un desastre, la regla a seguir es "no producir documentos a menos que sean necesarios de forma inmediata para tomar un decisión importante". Estos documentos deben ser cortos y centrarse en lo fundamental. Si una vez iniciado el proyecto, un nuevo miembro se incorpora al equipo de desarrollo, se considera que los dos elementos que más le van a servir para ponerse al día son: el propio código y la interacción con el equipo (H. Canós, y otros, 2003).

La colaboración con el cliente más que la negociación de un contrato. Las características particulares del desarrollo de software hacen que muchos proyectos hayan fracasado por intentar cumplir unos plazos y unos costes preestablecidos al inicio del mismo, según los requisitos que el cliente manifestaba en ese momento. Por ello, se propone que exista una interacción constante entre el cliente y el equipo de desarrollo. Esta colaboración entre ambos será la que marque la marcha del proyecto y asegure su éxito (H. Canós, y otros, 2003).

Responder a los cambios más que seguir estrictamente un plan. La habilidad de responder a los cambios que puedan surgir a los largo del proyecto determina también el éxito o fracaso del mismo. Por lo tanto, la planificación no debe ser estricta puesto que existen muchas variables en juego, debe ser flexible para poder adaptarse a los cambios que puedan surgir. Una buena estrategia es hacer planificaciones detalladas para unas pocas semanas y planificaciones mucho más abiertas para unos pocos meses (H. Canós, y otros, 2003).

Dentro del desarrollo de software, la creciente necesidad de que los proyectos lleguen al éxito y obtener un producto de gran valor para el cliente, genera grandes cambios en los requisitos iniciales del proyecto.

Esta es la razón de la importancia de una metodología que ajustada en un equipo cumpla con sus metas y satisfaga más allá de las necesidades definidas al inicio del proyecto pues el éxito del producto depende en gran parte de la metodología escogida (G. Figueroa, y otros, 2008).

En el proceso de valoración de la metodología a escoger, se deben tener en cuenta ciertas características deseables en una metodología tales como; existencia de reglas predefinidas, cobertura total del ciclo de desarrollo, planificación y control, utilización sobre un abanico amplio de proyectos, soporte al mantenimiento, soporte de la reutilización de software, entre otras.

Para elegir el tipo de metodología se ha de tener en cuenta ciertas diferencias que las distinguen. La Tabla 1.4 recoge esquemáticamente estas diferencias que no se refieren sólo al proceso en sí, sino también al contexto de equipo y organización que es más favorable a cada uno de estas filosofías de procesos de desarrollo de software (Letelier, y otros, 2010).

Metodologías Ágiles	Metodologías Tradicionales
Basadas en heurísticas provenientes de prácticas de producción de código	Basadas en normas provenientes de estándares seguidos por el entorno de desarrollo
Especialmente preparados para cambios durante el proyecto	Cierta resistencia a los cambios
Impuestas internamente (por el equipo de desarrollo)	Impuestas externamente
Proceso menos controlado, con pocos principios	Proceso mucho más controlado, con numerosas políticas/normas
No existe contrato tradicional o al menos es bastante flexible	Existe un contrato prefijado
El cliente es parte del equipo de desarrollo	El cliente interactúa con el equipo de desarrollo mediante reuniones
Grupos pequeños (<10 integrantes) y	Grupos grandes y posiblemente

trabajando en el mismo sitio	distribuidos
Pocos artefactos	Más artefactos
Pocos roles	Más roles
Menos énfasis en la arquitectura del software	La arquitectura del software es esencial y se expresa mediante modelos

Tabla 1.4. Comparación metodologías ágiles y tradicionales (Letelier, y otros, 2010).

De acuerdo con lo antes expuesto, se determina utilizar una metodología de tipo ágil para el proceso de desarrollo del subsistema de detección de roles de usuario dentro de comunidades de interés de Twitter, teniendo como punto de partida las características del equipo de trabajo, el cual es reducido, y que el tiempo de desarrollo es corto. En el proceso de elección de la metodología ágil a utilizar, se valoran dos variantes por su popularidad y utilidad en proyectos similares; Proceso Abierto Unificado (*Open Unified Process*, OpenUP por sus siglas en inglés) y Programación Extrema (*eXtreme Programming,* XP por sus siglas en inglés).

1.3.1.1. Proceso Abierto Unificado

OpenUp es una metodología de desarrollo de software ágil, que contiene el conjunto mínimo de prácticas que ayudan a los equipos a ser más eficaces en el desarrollo de software. OpenUp abraza una filosofía pragmática y ágil que se centra en la naturaleza colaborativa de desarrollo de software. Es un proceso iterativo que es mínimo, completo y extensible que puede utilizarse tal cual o ampliarse para tratar una amplia variedad de tipos de proyecto. Se caracteriza por ser iterativo e incremental, estar centrado en la arquitectura y guiado por los casos de uso. Está organizada dentro de cuatro áreas principales de contenido: Comunicación y Colaboración, Intención, Solución y Administración (Balduino, 2007).

Está organizado en dos dimensiones diferentes pero interrelacionadas: el método y el proceso. El contenido del método es donde los elementos del método (roles, tareas, artefactos y lineamientos) son definidos, sin tener en cuenta como son utilizados en el ciclo de vida del proyecto. El proceso es donde los elementos del método son aplicados de forma ordenada en el tiempo. Muchos ciclos de vida para diferentes proyectos

pueden ser creados a partir del mismo conjunto de elementos del método (Balduino, 2007).

OpenUP se caracteriza por cuatro principios básicos que se soportan mutuamente:

- Colaboración para alinear los intereses y un entendimiento compartido.
- Balance para confrontar las prioridades (necesidades y costos técnicos) para maximizar el valor para los *stakeholders* (persona o entidad que es afectada por las actividades de una organización).
- Enfoque en articular la arquitectura para facilitar la colaboración técnica, reducir los riesgos y minimizar excesos y trabajo extra.
- Evolución continúa para reducir riesgos y demostrar resultados.

Principales características de la metodología OpenUp:

- Metodología de desarrollo de software de código abierto diseñado para pequeños equipos organizados, quienes quieren tomar una aproximación ágil del desarrollo.
- Proceso iterativo e incremental que es Mínimo, Completo y Extensible.
- Se valora la colaboración y el aporte de los *stakeholders* sobre los entregables y las formalidades innecesarias.
- Practicantes de desarrollo de software (desarrolladores, administradores de proyectos, analistas y probadores) trabajan juntos como un equipo de proyecto.
- No define un modelo de negocio ni de dominio necesario.
- Permite detectar errores tempranos a través de un ciclo iterativo.
- Evita la elaboración de documentación, diagramas e iteraciones innecesarios.
- Por ser una metodología ágil tiene un enfoque centrado al cliente y con iteraciones cortas.

1.3.1.2. Programación Extrema

La Metodología de desarrollo de software **XP** se considera una metodología exitosa utilizada para proyectos de corto plazo y equipos pequeños; ha generado gran interés por su creciente número de casos de éxito en la industria, requiriendo de gran disciplina.

Un proceso ligero de bajo riesgo, flexible, predecible, científico y divertido de desarrollar. Su utilidad se mide en valores como la simplicidad, comunicación, realimentación y coraje. La metodología consiste en una programación rápida o extrema, cuya particularidad es tener como parte del equipo, al usuario final, pues es uno de los requisitos para llegar el éxito del proyecto (Robles, y otros, 2002).

XP es una metodología ágil centrada en potenciar las relaciones interpersonales como clave para el éxito en el desarrollo del software, promoviendo el trabajo en equipo, preocupándose por el aprendizaje de los desarrolladores y proporcionando un buen clima de trabajo. Se basa en la retroalimentación continúa entre el cliente y el equipo de desarrollo, comunicación fluida entre todos los participantes, simplicidad en las soluciones implementadas y coraje para enfrentar los cambios (Robles, y otros, 2002).

Según la valoración realizada, es seleccionada OpenUp como metodología de desarrollo de software, pues proporciona un marco ideal para el desarrollo de proyectos pequeños con escasos recursos y equipos de trabajo mínimos como es el caso, define todos los artefactos necesarios, es la metodología utilizada por excelencia en CIDI y tiene referente en los módulos para la PRST ya desarrollados.

1.3.2. Lenguaje de modelado

Un lenguaje permite la comunicación sobre un tema. En el desarrollo, el tema incluye los requisitos y el sistema. Sin un lenguaje, es difícil para los miembros del equipo de comunicación y colaboración desarrollar con éxito un sistema (Alhir, 2003).

Lenguaje Unificado de Modelado (*Unified Modeling Language,* UML por sus siglas en inglés), es un lenguaje visual para el modelado y la comunicación acerca de los sistemas mediante el uso de diagramas y texto de apoyo (Alhir, 2003).

UML es ante todo un lenguaje. Un lenguaje proporciona un vocabulario y unas reglas para permitir una comunicación. En este caso, este lenguaje se centra en la representación gráfica de un sistema, indica cómo crear y leer los modelos, pero no dice cómo crearlos. Esto último es el objetivo de las metodologías de desarrollo (Orallo, 2002).

Los objetivos de UML son muchos, pero se pueden sintetizar sus funciones: (Orallo, 2002)

- **Visualizar:** UML permite expresar de una forma gráfica un sistema de forma que otro lo puede entender.
- **Especificar:** UML permite especificar cuáles son las características de un sistema antes su construcción.
- **Construir:** a partir de los modelos especificados se pueden construir los sistemas diseñados.
- **Documentar:** los propios elementos gráficos sirven como documentación del sistema desarrollado que pueden ser utilizados para su futura revisión.

1.3.3. Herramientas de Ingeniería de Software Asistida por Computadoras

Las Herramientas de Ingeniería de Software Asistida por Computadoras (*Computer Aided Software Engineering*, CASE por sus siglas en inglés), son diversas aplicaciones informáticas destinadas a aumentar la productividad en el desarrollo de software reduciendo los costos en términos de tiempo y de dinero. Estas herramientas pueden ayudar en todos los aspectos del ciclo de vida de desarrollo del software en tareas como el proceso de realizar un diseño del proyecto, cálculo de costos, implementación de parte del código automáticamente con el diseño dado, compilación automática, documentación o detección de errores, entre otras (Pressman, 2009).

Según (Sommerville, 2005), “comprende un amplio abanico de diferentes tipos de programas que se utilizan para ayudar a las actividades del proceso del software, como análisis de requerimientos, el modelado de sistemas, la depuración y las pruebas. En la actualidad, todos los métodos vienen con tecnología CASE asociada, como los editores para las notaciones utilizadas en el método y módulos de análisis que verifican el modelo del sistema según las reglas del método y generadores de informes que ayudan a crear la documentación del sistema. Las herramientas CASE también incluyen un generador de código que automáticamente genera código fuente a partir del modelo del sistema y de algunas guías de procesos para los ingenieros de software”.

Visual Paradigm es una herramienta profesional que soporta UML y el ciclo de vida completo del desarrollo de software: análisis y diseño orientados a objetos, construcción, pruebas y despliegue. El software de modelado ayuda a la construcción

rápida de aplicaciones de calidad, mejores y a un menor coste, además de permitir el dibujo de todos los tipos de diagramas de clases, código inverso, generar código desde diagramas y generar documentación (Visual Paradigm, 2010).

Es seleccionada Visual Paradigm en su versión 8.0, pues está destinada para todo tipo de usuario interesado en el desarrollo de software a pequeña y gran escala usando un enfoque orientado a objetos. Además cuenta con licencia dual (gratuita y comercial) y soporte en diferentes plataformas (Windows, Linux) lo que le ofrece ventajas sobre otras herramientas CASE. Proporciona abundantes tutoriales, demostraciones y proyectos UML. Como característica atractiva para los desarrolladores, presenta una interfaz de uso intuitiva y con muchas facilidades a la hora de modelar los diagramas.

1.3.4. Lenguaje de programación

Un lenguaje de programación es un lenguaje artificial utilizado para la comunicación hombre-máquina a través del cual se pueden describir procedimientos que permiten dar solución a un problema determinado. Permite especificar de manera precisa sobre qué datos debe operar una computadora, cómo deben ser almacenados o transmitidos y qué acciones tomar ante determinada circunstancia.

Python (versión 2.7) es un lenguaje de propósito general creado por Guido van Rossum a principios de los años 90 cuyo nombre está inspirado en el grupo de cómicos ingleses “Monty Python”. Se trata de un lenguaje interpretado o de script, con tipado dinámico, multiplataforma y orientado a objetos (González Duque, 2008).

Es uno de los lenguajes de alto nivel más completo, caracterizado por ser muy legible y elegante además de simple y poderoso. Parte del concepto de que todo aquello que es innecesario no debe escribirse, soporta objetos y estructuras de datos de alto nivel, y posee una alta densidad. A pesar de no ser la rapidez de ejecución una de sus características, puede integrarse con módulos escritos en C/C++ para aquellas tareas que lo requieran (González Duque, 2008).

Actualmente es uno de los lenguajes de programación más utilizados en el mundo para proyectos de todo tipo y alcance. Una de sus mayores ventajas es que es un proyecto de código abierto razón por la cual la “Python Library” sigue creciendo constantemente. La selección de Python como lenguaje de programación, viene dado porque la

aplicación será integrada a la plataforma PRST la cual está concebida en Python usando Django como marco de trabajo.

1.3.5. Marco de trabajo

Es una estructura conceptual y tecnológica de soporte definida, normalmente con artefactos o módulos de software concretos, en base a la cual otro proyecto de software puede ser organizado y desarrollado. Típicamente, puede incluir soporte de programas, bibliotecas y un lenguaje interpretado entre otros programas para ayudar a desarrollar y unir los diferentes componentes de un proyecto (Framework, 2013).

Representa una arquitectura de software que modela las relaciones generales de las entidades del dominio. Provee una estructura y una metodología de trabajo la cual extiende o utiliza las aplicaciones del dominio. Son diseñados con la intención de facilitar el desarrollo de software permitiendo a los diseñadores y programadores pasar más tiempo identificando requerimientos de software que tratando con los tediosos detalles de bajo nivel de proveer un sistema funcional (Framework, 2013).

Django (versión 1.6) es un marco de trabajo web de código abierto sobre Python que permite desarrollar rápidamente aplicaciones web. El objetivo es poder montar sitios nuevos y añadir contenidos de forma muy rápida y dinámica. Perfecto para el manejo del crecimiento continuo de trabajo de manera fluida sin perder calidad en los servicios (escalable). Pone énfasis en el re-uso, la conectividad y extensibilidad de componentes, del desarrollo rápido y sigue el principio “No te repitas” eliminando redundancias resultando en un código más corto y legible (Infante, 2012).

1.3.6. Sistema Gestor de Bases de Datos

Un Sistema Gestor de Base de Datos (SGBD) es un sistema de software que permite la definición de base de datos; así como la elección de las estructuras de datos necesarios para el almacenamiento y búsqueda de los datos, ya sea de forma interactiva o a través de un lenguaje de programación. Un SGBD relacional es un modelo de datos que facilita a los usuarios describir los datos que serán almacenados en la base de datos junto con un grupo de operaciones para manejar los datos (SGBD, 2012).

Los SGBD relacionales son una herramienta efectiva que permite a varios usuarios acceder a los datos al mismo tiempo. Brindan facilidades eficientes y un grupo de funciones con el objetivo de garantizar la confidencialidad, la calidad, la seguridad y la

integridad de los datos que contienen, así como un acceso fácil y eficiente a los mismos (SGBD, 2012).

PostgreSQL (versión 9.3) es un SGBD objeto-relacional, funciona muy bien con grandes cantidades de datos y una alta concurrencia de usuarios accediendo a la vez al sistema, está distribuido bajo licencia BSD y su código fuente disponible libremente.

PostgreSQL utiliza un modelo cliente/servidor y usa multiprocesos en vez de multihilos para garantizar la estabilidad del sistema. Un fallo en uno de los procesos no afectará el resto; el sistema continuará funcionando.

Disponible para Linux y UNIX en todas sus variantes (AIX, BSD, HP-UX, SGI IRIX, Mac OS X, Solaris, Tru64) y Windows 32/64bit (PostgreSQL, 2010).

Todas estas características, lo convierte en un candidato ideal para cualquier tipo de aplicación, ajustable perfectamente a las necesidades de software y del entorno de desarrollo a utilizar.

1.4. Consideraciones parciales del capítulo

El estudio de los referentes teóricos-metodológicos arrojó las primeras pistas en relación al análisis de redes sociales, posibilitó conocer las herramientas y los métodos, así como los primeros requisitos funcionales. Como resultado, se llegó necesidad de construir un subsistema web propio pues los existentes están desarrollados para plataformas privativas, poseen licencia comercial para su uso y la mayoría no interactúa con Twitter. Por otra parte, las herramientas y metodologías seleccionadas, brindan un marco de trabajo completo y propicio para un buen desarrollo.

CAPÍTULO 2: CARACTERÍSTICAS Y DISEÑO DEL SUBSISTEMA

En el presente capítulo se definen las principales características que debe cumplir el software a desarrollar en términos de requisitos funcionales y no funcionales los cuales son representados mediante diagramas de casos de uso en los que se describen las relaciones existentes con los actores involucrados.

2.1. Descripción del subsistema a desarrollar

El subsistema consiste de un módulo web que será integrado posteriormente a la PRST. Su funcionamiento se puede resumir en una aplicación que reciba una colección de tuits, previamente descargada y almacenada en un fichero en formato Notación de Objetos de JavaScript (*JavaScript Object Notation*, JSON por sus siglas en inglés), a partir de la cual diseñará la estructura de la red. Como salida tendrá, para cada usuario, el valor de cada una de las métricas elegidas para la investigación.

2.1.1. Notación de Objetos de JavaScript

JSON, es un formato ligero de intercambio de datos. Leerlo y escribirlo es simple para humanos, mientras que para las máquinas es simple interpretarlo y generarlo. Está basado en un subconjunto del Lenguaje de Programación JavaScript, Estándar ECMA-262 3ra Edición – Diciembre 1999. JSON es un formato de texto que es completamente independiente del lenguaje pero utiliza convenciones que son ampliamente conocidos por los programadores de la familia de lenguajes C, incluyendo C, C++, C#, Java, JavaScript, Perl, Python, y muchos otros. Estas propiedades hacen que JSON sea un lenguaje ideal para el intercambio de datos (JSON, 2014).

JSON está constituido por dos estructuras: (JSON, 2014)

- Una colección de pares de nombre/valor. En varios lenguajes esto es conocido como un objeto, registro, estructura, diccionario, tabla hash, lista de claves o un arreglo asociativo.
- Una lista ordenada de valores. En la mayoría de los lenguajes, esto se implementa como arreglos, vectores, listas o secuencias.

Estas son estructuras universales; virtualmente todos los lenguajes de programación las soportan de una forma u otra. Es razonable que un formato de intercambio de datos que es independiente del lenguaje de programación se base en estas estructuras (JSON, 2014).

2.2. Estructura de la red

Cada tuit consta de una extensa estructura en formato JSON con cantidades de datos innecesarios para la actual propuesta de solución, por lo que es necesario definir sólo aquellos que de alguna forma establecen algún tipo de relación entre dos usuarios. A continuación, se resume mediante un ejemplo, la estructura con los datos relevantes para la investigación.

```
{
        "in_reply_to_screen_name": "A",
        "entities": {
        "user_mentions": [ { "screen_name": "B", }],
        },
        "contributors": [ { "screen_name": "C", } ],
        "retweeted_status": {
                "in_reply_to_screen_name": "C",
        "entities": { "user_mentions": [ ] },
                "contributors": [ { "screen_name": "A" } ],
        "user": { "screen_name": "B", },
        },
        "user": { "screen_name": "D" },
}
```

Con estos datos, es posible construir cuatro tipos de relaciones entre usuarios: Mención, Réplica, Contribución y Retuit, que hacen referencia, por ese mismo orden, a cuando un usuario “Menciona a”, “Replica a”, “Contribuye a” o “Retuitea a” otro usuario.

La estructura construida a partir del caso de ejemplo es la siguiente (ver Figura 2.1).

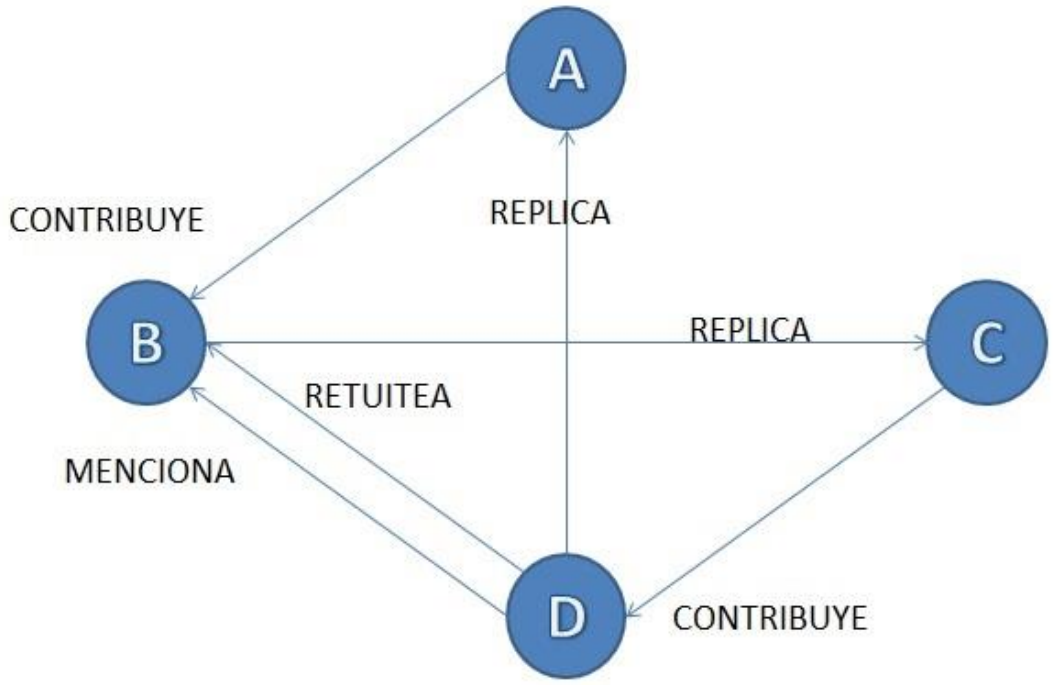


Figura 2.1. Estructura de la red para el caso de ejemplo (Elaboración propia).

2.3. Modelo conceptual

El modelo conceptual resulta una herramienta de comunicación fundamental que obliga y permite a desarrolladores a pensar formalmente en el problema y a validar su comprensión del mismo. Mediante este modelo se establece además un vocabulario propio del problema; y junto a los requerimientos constituye la entrada más importante para el diseño. Es importante aclarar que por lo general se requieren varias iteraciones para obtener un buen modelo (Kruchten, y otros, 2003).

El modelo conceptual, como el modelo de dominio es una representación visual estática que se realiza con el objetivo de lograr mejor comprensión del contexto para la obtención de requisitos del sistema.

Básicamente, de manera visual no es más que un diagrama de clase UML que modela los conceptos básicos asociados al dominio del problema, sus propiedades más importantes y las relaciones que resultan imprescindibles para contextualizar dichos conceptos, a fin de poder brindar elementos que permitan la identificación de los requisitos del sistema. Además, es válido para cualquier metodología de desarrollo de

software. Este es una representación de conceptos en el dominio del problema, representa cosas del mundo real, no componentes del software. Este es una representación de conceptos en el dominio del problema, representa cosas del mundo real, no componentes del software (Larman, 1999).

2.3.1. Diagrama del modelo conceptual

En el seno de la red social **Twitter**, los **usuarios** expresan sus opiniones en torno a diversos **temas** de interés formando **comunidades**. Cada una de estas comunidades, de forma explícita o implícita, permiten que sus usuarios se desenvuelvan en determinados **roles** en dependencia de la función que ejercen y la interacción con el resto de los usuarios.

En el siguiente modelo conceptual (ver Diagrama 2.1), se relacionan haciendo uso del lenguaje de modelado UML los conceptos presentes a manera de comprender el flujo de trabajo del negocio.

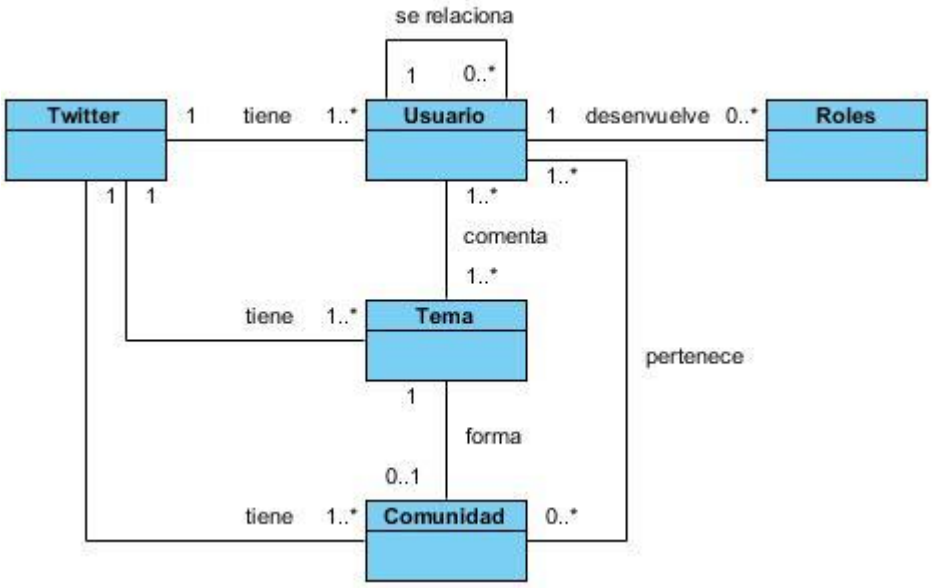


Diagrama 2.1. Diagrama del modelo conceptual (Elaboración propia).

2.4. Modelo de casos de uso del sistema

El modelo de casos de uso del sistema especifica los actores que interactúan con el sistema, los casos de uso y las relaciones que existen entre ellos. Representa un esquema donde se recopilan las funcionalidades a automatizar y se determina cómo

será utilizado desde el punto de vista del usuario (actor), pues se construye sobre la base de sus necesidades (Jacobson, 2000).

2.4.1. Actores del negocio

El actor representa una entidad externa que interactúa con el sistema. Las entidades externas podrían ser personas u otros sistemas. Es importante resaltar que los actores son abstracciones de papeles o roles y no necesariamente tienen una correspondencia directa con personas (Pow Sang Portillo, 2003).

Para el caso de estudio, se definen tres actores que son los encargados de interactuar con el subsistema, los cuales se detallan a continuación en la Tabla 2.2.

Actor del sistema	Descripción
Usuario	Usuario anónimo al sistema
Usuario autenticado	Usuario una vez completada la autenticación correctamente
Administrador	Usuario autenticado encargado de la gestión del sistema

Tabla 2.2. Descripción de los actores del sistema (Elaboración propia).

2.4.2. Caso de uso

A diferencia del actor, el caso de uso hace referencia al sistema a construir, detallando su comportamiento, el cual se traduce en resultados que pueden ser observados por el actor. Los casos de uso describen las cosas que los actores quieren que el sistema haga, por lo que un caso de uso debería ser una tarea completa desde la perspectiva del actor (Pow Sang Portillo, 2003).

Partiendo de lo planteado y de los requerimientos del sistema se identifican los siguientes casos de uso. Para ello se utiliza la notación (CU), refiriendo a “Caso de Uso”.

CU 1. Autenticar usuario.

CU 2. Gestionar usuarios del sistema.

CU 3. Analizar comunidad de interés.

CU 4. Listar usuarios de Twitter.

CU 5. Buscar usuario de Twitter.

CU 6. Mostrar detalles de usuario de Twitter.

CU 7. Ordenar usuarios de Twitter por métricas.

2.4.3. Modelo de casos de uso

Los actores y los casos de uso forman un modelo al que se le denomina “Modelo de Casos de Uso”. Dicho modelo muestra el comportamiento del sistema desde la perspectiva del usuario y servirá como producto de entrada para el análisis y diseño del sistema (Pow Sang Portillo, 2003).

En el Modelo de Casos de Uso(ver Diagrama 2.2), se muestran los actores y los casos de uso identificados, así como, las relaciones entre ellos.

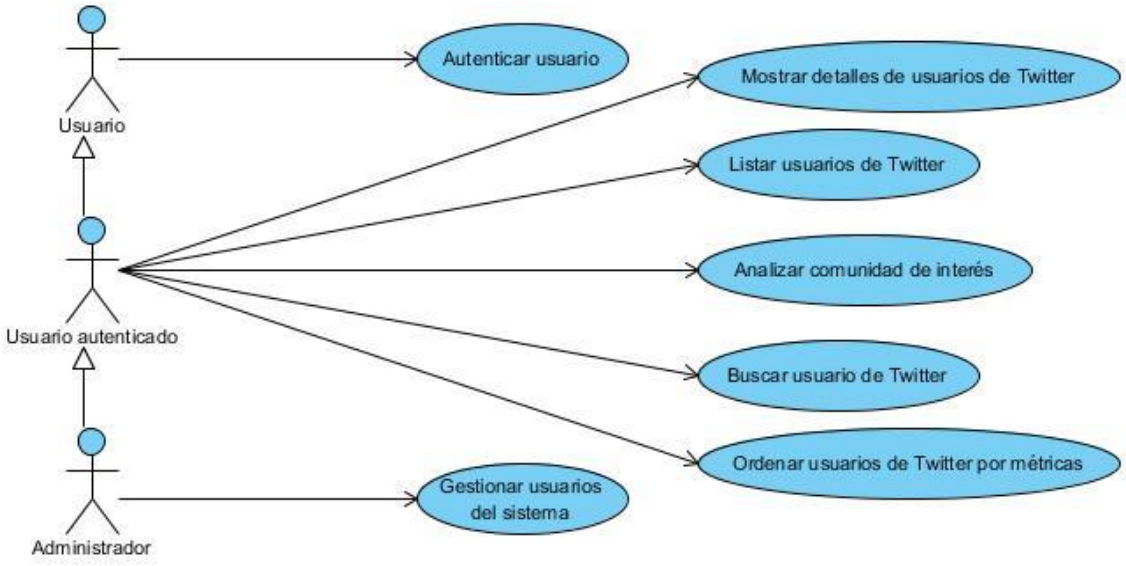


Diagrama 2.2. Modelo de Casos de Uso del subsistema (Elaboración propia).

2.4.4. Descripción de casos de uso del subsistema

En el siguiente epígrafe se describen los casos de uso del subsistema mediante el artefacto “Especificación de Casos de Uso” (ver Tablas 2.3, 2.4, 2.5, 2.6, 2.7, 2.8, 2.9).

Objetivo	Autenticar un usuario al sistema
Actores	Usuario.
Resumen	Inicia con la solicitud del Usuario de autenticarse. Finaliza con el usuario autenticado en el sistema.

Complejidad	Alta
Prioridad	Baja
Precondiciones	El usuario está registrado en el sistema.
Postcondiciones	
Flujo de eventos	
Flujo básico "Autenticar usuario"	

	Actor	Sistema
1.	El usuario solicita autenticarse.	
2.		El sistema muestra un formulario de autenticación.
3.	El usuario entra su nombre de usuario y contraseña.	
4.		El sistema recibe los datos, los valida y autentica el usuario.
5.		Termina el caso de uso.

Flujos alternos	
Nº 4 Los datos son incorrectos	

	Actor	Sistema
1.		Muestra un mensaje indicando que los datos son incorrectos.
2.		Comienza por el Nº 2 del flujo normal de eventos.

Requisitos no funcionales	
Asuntos	

pendientes	

Tabla 2.3. Descripción del caso de uso “Autenticar usuario” (Elaboración propia).

Objetivo	Adicionar, modificar o eliminar usuarios del sistema.	
Actores	Administrador.	
Resumen	Este caso se inicia cuando el Administrador adiciona, modifica o elimina un usuario del sistema. El caso de uso termina con la creación, modificación o eliminación de un usuario del sistema o la cancelación de cualquier operación de las anteriores.	
Complejidad	Alta	
Prioridad	Baja	
Precondiciones	El usuario está autenticado en el sistema. El usuario tiene los permisos de administración.	
Postcondiciones	Quedan registrados los datos de la operación.	
Flujo de eventos		
Flujo básico “Gestionar usuarios”		
	Actor	**Sistema**
1.	El Administrador entra al panel de administración y lista los usuarios del sistema.	
2.		El sistema muestra un listado con todos los usuarios del sistema y una opción “Adicionar usuario”.
3.	El Administrador escoge una de las opciones: • Adicionar usuario, presionando	

	en “Adicionar usuario”. Sección “Adicionar usuario”. • Eliminar usuario, presionando en la opción “Eliminar” correspondiente al usuario que desea eliminar. Sección “Eliminar usuario”. • Modificar usuario, presionando en la sobre el nombre de usuario que desea modificar. Sección “Modificar usuario”.	
4.		Termina el caso de uso.
Sección 1: Adicionar usuario.		
Flujo básico “Adicionar usuario”		
	Actor	**Sistema**
1.	El Administrador presiona la opción “Adicionar usuario”.	
2.		El sistema muestra un formulario para llenar los datos del nuevo usuario del sistema.
3.	El Administrador rellena los datos y presiona “Salvar”.	
4.		El sistema valida que los datos del nuevo usuario sean correctos.
5.		El sistema valida que el usuario no haya sido creado previamente.
6.		El sistema adiciona el nuevo usuario del sistema a la base de datos.
Flujos alternos		

Nº 4 Los datos del usuario son incorrectos		
	Actor	**Sistema**
3.		Muestra un mensaje indicando que los datos son incorrectos.
4.		Comienza por el Nº 2 del flujo normal de eventos.
Nº 5 El usuario ya ha sido creado		
	Actor	**Sistema**
1.		Muestra un mensaje indicando que el usuario ya existe.
2.		Comienza por el Nº 2 del flujo normal de eventos.
Sección 2: Eliminar usuario.		
Flujo básico “Eliminar usuario”		
	Actor	**Sistema**
1.	El Administrador selecciona la opción “Eliminar” del usuario deseado.	
2.		El sistema muestra un mensaje preguntando si desea eliminar el usuario seleccionado.
3.	El Administrador selecciona la opción “Aceptar”.	
4.		El sistema borra el usuario de la base de datos.
Flujos alternos		
Nº 3 El Administrador cancela la petición		
	Actor	**Sistema**

1.	El Administrador selecciona la opción “Cancelar”.	
Sección 3: Modificar usuario.		
Flujo básico “Modificar usuario”		
	Actor	**Sistema**
1.	El Administrador selecciona el usuario deseado.	
2.		El sistema muestra un formulario con los datos actuales del usuario.
3.	El Administrador modifica los datos deseados y presiona en “Salvar”.	
4.		El sistema actualiza los datos del usuario.
Requisitos no funcionales		
Asuntos pendientes		

Tabla 2.4. Descripción del caso de uso “Gestionar usuarios del sistema” (Elaboración propia).

Objetivo	Importar una colección de tuits representando una comunidad de interés desde un fichero en formato JSON para determinar los usuarios y sus interacciones haciendo el análisis de las métricas de centralidad.
Actores	Usuario autenticado.
Resumen	Inicia con la solicitud del Usuario de analizar una nueva comunidad de interés. Finaliza con el cálculo de las métricas de centralidad sobre la comunidad que define la colección.
Complejidad	Alta

Prioridad	Alta
Precondiciones	Previa autenticación. Haber descargado colección de tuits en un fichero con correcto formato JSON.
Postcondiciones	

Flujo de eventos		
Flujo básico "Análisis de comunidad de interés"		
	Actor	**Sistema**
1.	El Usuario solicita analizar una nueva comunidad.	
		El sistema muestra un formulario que permita subir un fichero.
	El Usuario presiona "Examinar", selecciona un fichero y presiona "Aceptar".	
		El sistema recibe el fichero, lo valida, recoge los datos de los usuarios de Twitter, construye los enlaces y calcula las métricas de centralidad.
		Termina el caso de uso.
Flujos alternos		
Nº 3 El fichero no es válido		
	Actor	**Sistema**
1.		Muestra un mensaje indicando que ocurrió un error.
2.		Comienza por el Nº 2 del flujo normal de eventos.

Requisitos no funcionales	
Asuntos pendientes	

Tabla 2.5. Descripción del caso de uso “Analizar comunidad de interés” (Elaboración propia).

Objetivo	Listar todos los usuarios Twitter detectados durante el análisis de la comunidad de interés mostrando el valor de cada una de las métricas para cada usuario.	
Actores	Usuario.	
Resumen	Inicia con la solicitud del Usuario de listar los usuarios de Twitter. Finaliza cuando el sistema lista los usuarios de Twitter.	
Complejidad	Alta	
Prioridad	Alta	
Precondiciones	Previa autenticación. Haber analizado una comunidad de interés de Twitter.	
Postcondiciones		
Flujo de eventos		
Flujo básico “Listar usuarios Twitter”		
	Actor	**Sistema**
1.	El Usuario solicita listar los usuarios Twitter.	
2.		El sistema muestra una vista con un listado de los usuarios de Twitter detectados durante el último análisis.

3.		Termina el caso de uso.
Requisitos no funcionales		
Asuntos pendientes		

Tabla 2.6. Descripción del caso de uso “Listar usuarios de Twitter” (Elaboración propia).

Objetivo	Listar todos los usuarios de Twitter que coincidan con un criterio de búsqueda.	
Actores	Usuario.	
Resumen	Inicia con el ingreso por parte del Usuario de un criterio de búsqueda. Finaliza cuando el sistema lista todos los usuarios de Twitter que coinciden con el criterio de búsqueda.	
Complejidad	Alta	
Prioridad	Alta	
Precondiciones	Previa autenticación.	
Postcondiciones		
Flujo de eventos		
Flujo básico “Buscar usuario de Twitter”		
	Actor	**Sistema**
1.		El sistema muestra una vista con un listado de los usuarios de Twitter detectados durante el último análisis y un campo de texto.
2.	El Usuario ingresa en el campo el criterio de búsqueda y presiona “Buscar”.	

3.		El sistema muestra un listado de los usuarios de Twitter que coinciden con el criterio de búsqueda.
4.		Termina el caso de uso.
Requisitos no funcionales		
Asuntos pendientes		

Tabla 2.7. Descripción del caso de uso “Buscar usuario de Twitter” (Elaboración propia).

Objetivo	Mostrar para un usuario los valores para cada una de las métricas de centralidad así como los enlaces que posee.	
Actores	Usuario.	
Resumen	Inicia con la solicitud del Usuario de mostrar los detalles del usuario de Twitter que desee. Finaliza cuando el sistema muestra los valores de centralidad y lista los enlaces del usuario de Twitter deseado.	
Complejidad	Alta	
Prioridad	Alta	
Precondiciones	Previa autenticación.	
Postcondiciones		
Flujo de eventos		
Flujo básico “Mostrar detalles de usuario de Twitter”		
	Actor	**Sistema**
1.		El sistema muestra una vista con un listado de los usuarios de Twitter detectados durante el último análisis.

2.	El Usuario elige un usuario de Twitter y presiona sobre el nombre de usuario.	
3.		El sistema muestra los valores de centralidad y lista los enlaces del usuario de Twitter deseado.
4.		Termina el caso de uso.
Requisitos no funcionales		
Asuntos pendientes		

Tabla 2.8. Descripción del caso de uso “Mostrar detalles de usuario de Twitter” (Elaboración propia).

Objetivo	Ordenar el listado de usuarios de Twitter por una métrica de centralidad.	
Actores	Usuario.	
Resumen	El usuario presiona sobre la métrica deseada y la lista es reordenada por ese parámetro en orden decreciente.	
Complejidad	Alta	
Prioridad	Alta	
Precondiciones	Previa autenticación.	
Postcondiciones		
Flujo de eventos		
Flujo básico “Ordenar usuarios de Twitter por métricas”		
	Actor	**Sistema**
1.		El sistema muestra una vista con un

		listado de los usuarios de Twitter detectados durante el último análisis.
2.	El Usuario elige una métrica y presiona en el encabezado de la columna correspondiente.	
3.		El sistema reordena el listado de usuarios de Twitter en orden decreciente por el valor de la métrica escogida.
4.		Termina el caso de uso.
Requisitos no funcionales		
Asuntos pendientes		

Tabla 2.9. Descripción del caso de uso “Ordenar usuarios de Twitter por métricas” (Elaboración propia).

2.5. Diagramas de secuencia

El diagrama de la secuencia de un sistema es una representación que muestra, en determinado escenario de un caso de uso, los eventos generados por actores externos, su orden y los eventos internos del sistema. A todos los sistemas se les trata como una caja negra; los diagramas se centran en los eventos que trascienden las fronteras del sistema y que fluyen de los actores a los sistemas (Larman, 1999).

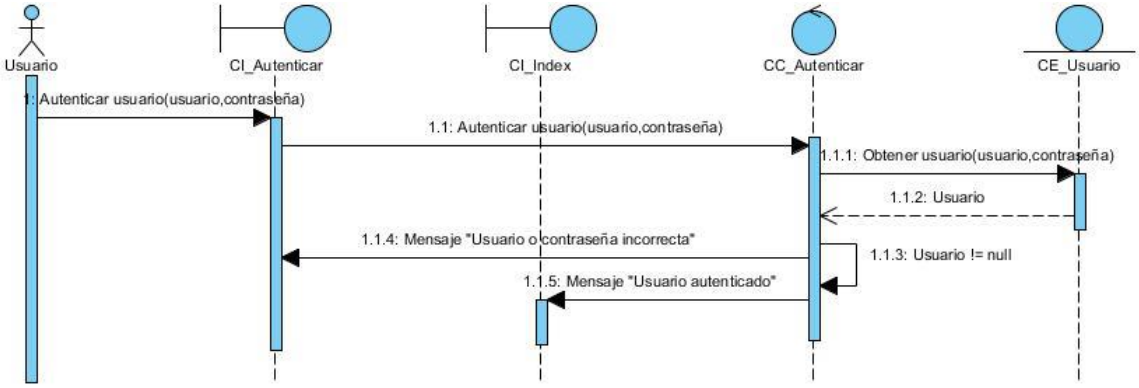


Diagrama 2.3. Diagrama de secuencia “Autenticar usuario” (Elaboración propia).

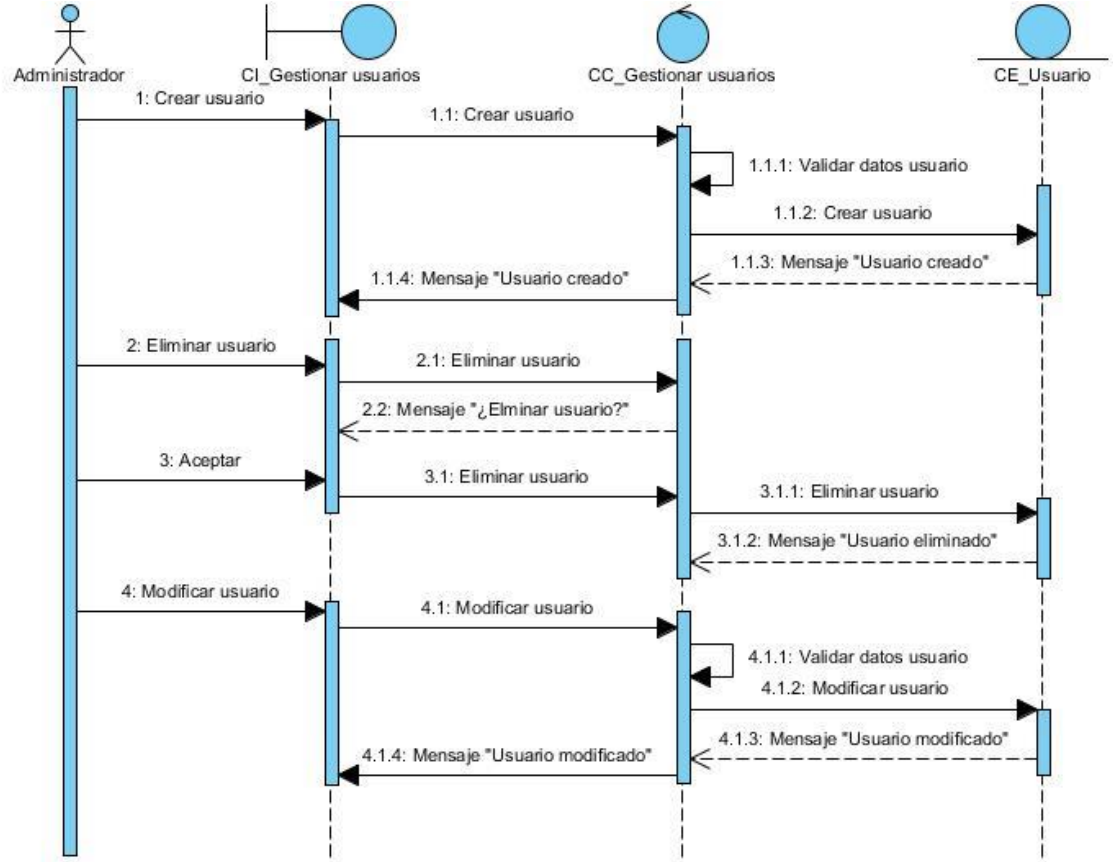


Diagrama 2.4. Diagrama de secuencia “Gestionar usuarios del sistema” (Elaboración propia).

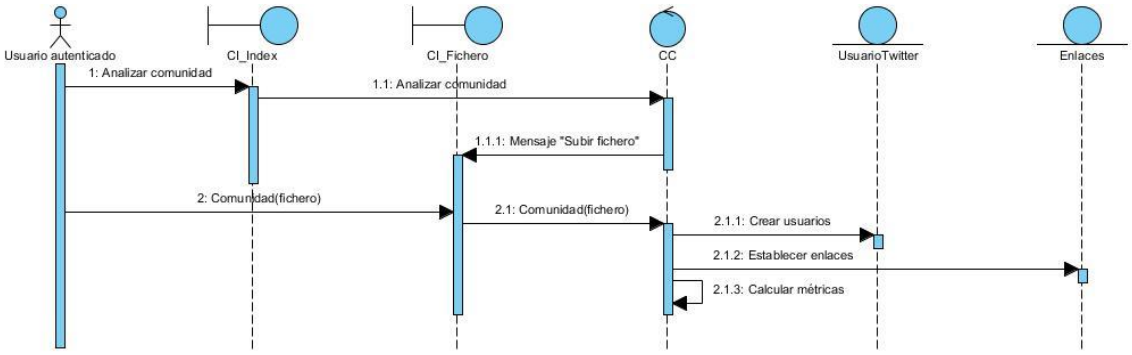


Diagrama 2.5. Diagrama de secuencia “Analizar comunidad de interés” (Elaboración propia).

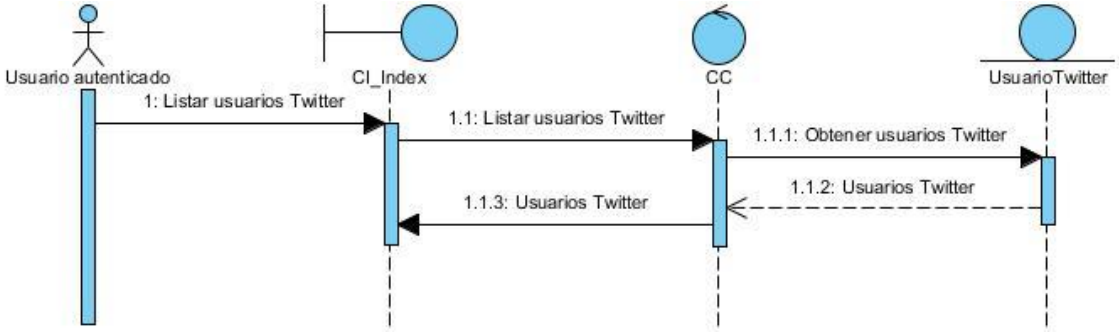


Diagrama 2.6. Diagrama de secuencia “Listar usuarios de Twitter” (Elaboración propia).

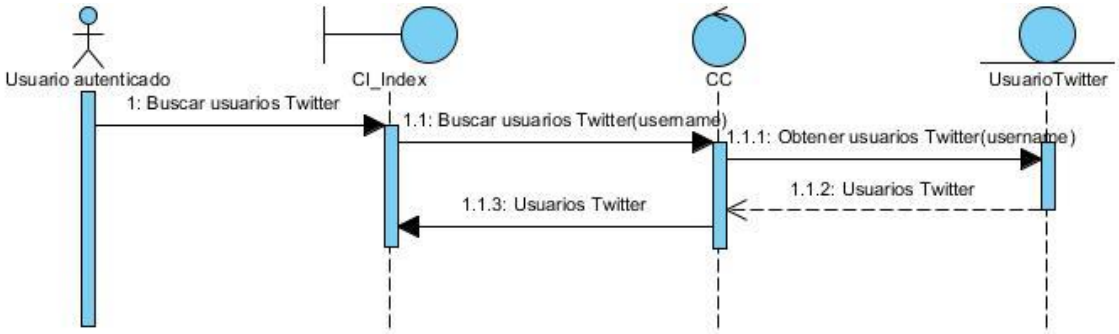


Diagrama 2.7. Diagrama de secuencia “Buscar usuario de Twitter” (Elaboración propia).

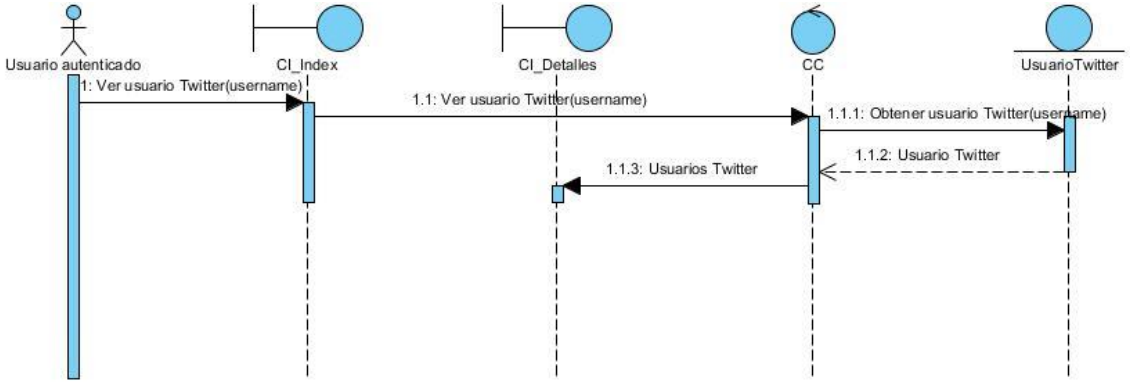


Diagrama 2.8. Diagrama de secuencia “Mostrar detalles de usuario de Twitter” (Elaboración propia).

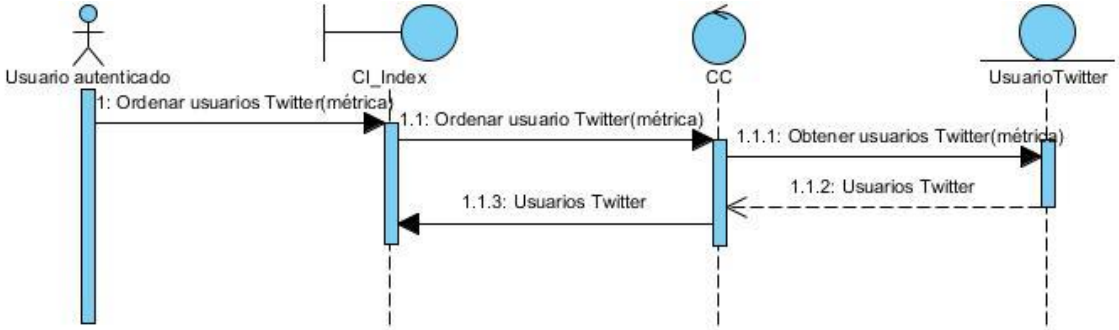


Diagrama 2.9. Diagrama de secuencia “Ordenar usuarios de Twitter por métricas” (Elaboración propia).

2.6. Levantamiento de requisitos

Los requerimientos de software son las características y cualidades que el sistema debe tener. El proceso de captura de requisitos, da inicio a la interacción con el cliente, trazando como meta fundamental satisfacer sus necesidades las cuales son analizadas e investigadas derivando en un estudio exhaustivo de los referentes teóricos-metodológicos. A partir de este punto, se describen las condiciones que se necesitan para dar cumplimiento a los objetivos planteados. Una vez definidos e identificados los mismos, se tiene la visión general de lo que se quiere hacer en el sistema.

2.6.1. Requisitos funcionales

Los requisitos funcionales se definen como las condiciones o capacidades que el sistema debe cumplir. Responden a ¿Qué debe hacer el sistema? (Pressman, 2009).

Luego de todo el estudio realizado se identificaron los requerimientos funcionales con los que debe contar el subsistema, los cuales se listan a continuación. Para ello se utiliza la notación (RF), refiriendo a “Requisito Funcional”.

RF 1. Autenticar usuario del sistema.

RF 2. Crear usuario del sistema.

RF 3. Eliminar usuario del sistema.

RF 4. Modificar usuario del sistema.

RF 5. Recibir colección de tuits desde un fichero en formato json.

RF 6. Detectar usuarios de Twitter.

RF 7. Construir enlaces entre los usuarios de Twitter.

RF 8. Calcular Centralidad de Grado.

RF 9. Calcular Centralidad de Intermediación.

RF 10. Calcular Centralidad de Cercanía.

RF 11. Calcular PageRank.

RF 12. Listar usuarios de Twitter.

RF 13. Mostrar detalles de usuario de Twitter.

RF 14. Ordenar usuarios por valor de Centralidad de Grado.

RF 15. Ordenar usuarios por valor de Centralidad de Intermediación.

RF 16. Ordenar usuarios por valor de Centralidad de Cercanía.

RF 17. Ordenar usuarios por valor de PageRank.

En la Tabla 2.1 se clasifican los requisitos funcionales de acuerdo a la prioridad del cliente.

Prioridad	Cantidad de requisitos funcionales
Alta	7
Media	10

Baja	0
Total	**17**

Tabla 2.1. Requisitos funcionales (Elaboración propia).

2.6.2. Requisitos no funcionales

Los requisitos no funcionales se definen como las cualidades o propiedades que el software debe tener. Debe pensarse en estas propiedades como las características que hacen al producto atractivo, usable, funcional, rápido o confiable. Normalmente están vinculados a los requerimientos funcionales, es decir, una vez que se conozca lo que el sistema debe hacer, podemos determinar cómo ha de comportarse. Se definen entonces los siguientes requerimientos no funcionales. Para ello se utiliza la notación (RnF), refiriendo a “Requisito no Funcional”.

2.6.2.1. Interfaz.

RnF 1. Interfaz sencilla, intuitiva y amigable para sus usuarios.

RnF 2. Implementar la ejecución de acciones de una manera rápida, minimizando los pasos a ejecutar en cada proceso.

RnF 3. La interfaz deberá garantizar la distinción visual entre los elementos.

2.6.2.2. Requerimientos de hardware

RnF 4. Para que el servidor realice todas las funcionalidades especificadas se requiere:

- 4GB de memoria RAM como mínimo.
- Procesador familia INTEL u otro con una velocidad mínima de 2.40 GHz.

2.6.2.3. Software

RnF 5. Debe ser multiplataforma para garantizar compatibilidad.

2.6.2.4. Accesibilidad

RnF 6. Debe ser accesible desde los navegadores web Mozilla Firefox (versión 24.0 o superior) y Google Chrome (versión 30.0 o superior).

2.6.2.5. Integración

RnF 7. Módulo con bajo acoplamiento y alta cohesión que facilite la integración.

2.7. Consideraciones parciales del capítulo

Con el trabajo realizado en el capítulo, se detectaron los requisitos necesarios para el cliente, se planteó un desarrollo simple y potente capaz de cumplir con todas las exigencias. Como resultado, la propuesta de solución queda lista para su implementación y proceso de pruebas.

CAPÍTULO 3: IMPLEMENTACIÓN Y PRUEBAS

En el capítulo se describe el proceso de implementación de los elementos identificados durante la realización del diseño. Además se incluyen las pruebas funcionales y no funcionales realizadas y sus resultados.

3.1. Estilos de código

Los estilos son normas usadas para escribir código y que incluye una gran gama de aspectos dentro del proceso de codificación. Un buen estilo de programación debe aportar a la eficiencia del proceso de desarrollo, logrando que los programas sean robustos y comprensibles (Muñoz Caro, y otros, 2002).

Uno de los aspectos a tener en cuenta para lograr mayor calidad en los productos de software es codificar de una manera clara y legible.

El mejor estilo es el que más aporte a la eficiencia del proceso de desarrollo y el que logre los programas más robustos y fáciles de usar. Todo lo que vaya en contra de esto es incorrecto, sobre todo porque para cumplir con este principio no se entra en contradicción con la propia lógica del problema.

Independientemente de que las plataformas de desarrollo influyan en cierto modo a la formación de estilos de programación, es responsabilidad de cada equipo determinar el suyo. A la hora de confeccionar un estilo de código, existen aspectos fundamentales que se deben tener en cuenta; notación, comentarios, identación, espacios y líneas en blanco, longitud máxima de las líneas de caracteres, declaración de variables, declaración de constantes y parámetros en los métodos (Muñoz Caro, y otros, 2002).

Para el desarrollo, se usa el definido en la guía de estilo del código Python (Rossum, y otros, 2001).

3.2. Diagrama de componentes

Los diagramas de componentes describen los elementos físicos del sistema y sus relaciones. Se utilizan para modelar la vista estática de un sistema. Muestran la organización y las dependencias lógicas entre un conjunto de componentes de

software, sean estos componentes de código fuente, librerías, binarios o ejecutables (Booch, y otros, 2000).

El uso más importante de estos diagramas es mostrar la estructura de alto nivel del modelo de implementación especificando que:

- Muestran las organizaciones y las dependencias entre tipos de componentes.
- Organizan los subsistemas de implementación en capas.

El diagrama de componentes mostrado en el Diagrama 3.1, corresponde al subsistema para la detección de roles de usuario en comunidades de interés de Twitter.

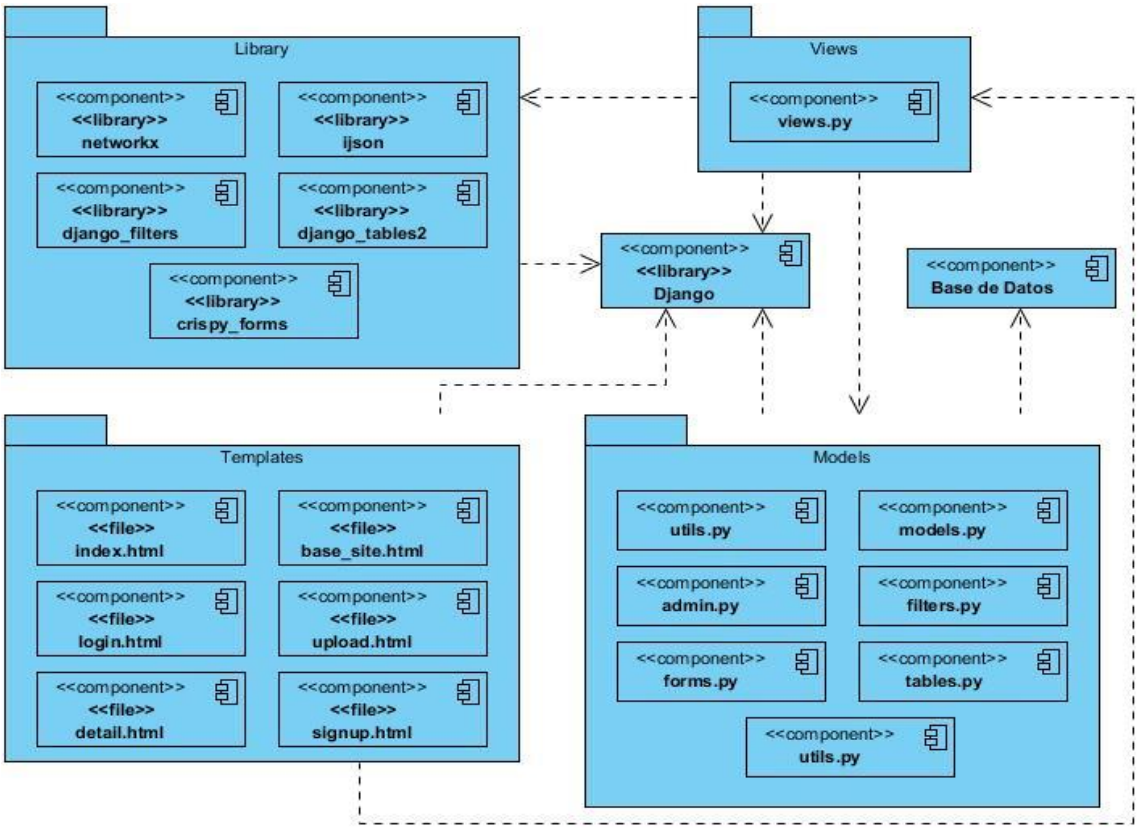


Diagrama 3.1. Diagrama de componentes del subsistema (Elaboración propia).

3.3. Patrones arquitectónicos

Un patrón arquitectónico define la estructura básica de una aplicación, provee un subconjunto de subsistemas predefinidos, incluyendo reglas, lineamientos para conectarlos y pautas para su organización y constituye una plantilla de construcción.

Entre las ventajas del uso de patrones, se pueden encontrar:

- Permitir la reutilización.
- Controlar la complejidad del diseño.
- Proporcionar un vocabulario común para la comunicación desarrollador-diseñador.

Para el desarrollo, se siguió el patrón arquitectónico Modelo-Vista-Controlador (MVC), pues la estructura actual de Django define claramente este patrón separando la lógica del negocio en tres componentes fundamentales; *Models, Views, Templates (MVT)*. El patrón MVT en Django hace referencia a Modelo-Vista-Plantilla, y es sencillo de entender, el modelo sigue siendo el modelo, en este caso, la vista no es una vista, sino que más bien es un controlador que se llama vista, y la plantilla es la vista del MVC.

En la Figura 3.1 se muestra, para mayor claridad, el MVT de Django. Las relaciones enumeradas explican su funcionamiento haciendo referencia a:

1. El navegador envía una solicitud.
2. La vista interactúa con el modelo para obtener datos.
3. La vista llama a la plantilla.
4. La plantilla renderiza la respuesta a la solicitud del navegador.

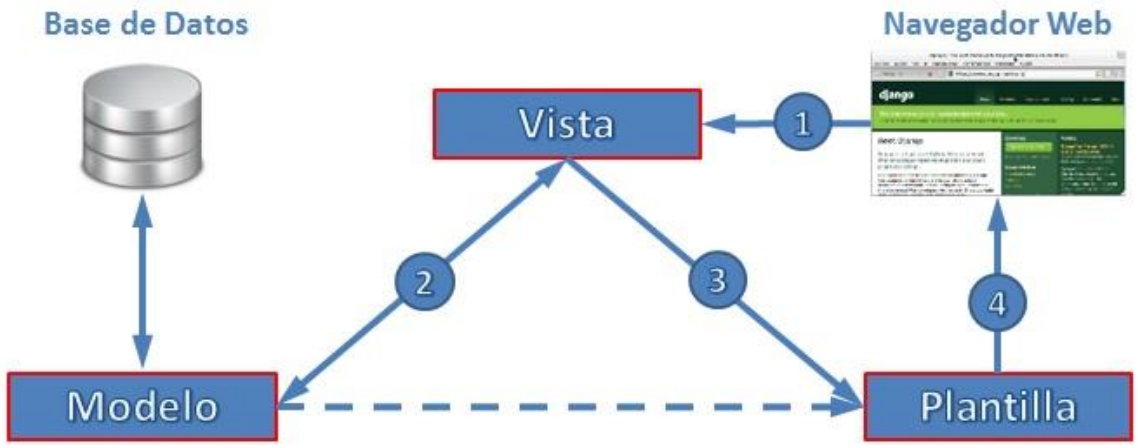


Figura 3.1. Patrón MVT en Django (Elaboración propia).

3.3.1. Patrón Modelo-Vista-Controlador

El Modelo-Vista-Controlador (MVC) es un patrón de arquitectura de software que separa los datos y la lógica de negocio de una aplicación de la interfaz de usuario y el módulo

encargado de gestionar los eventos y las comunicaciones. Para ello MVC propone la construcción de tres componentes distintos que son el **modelo**, la **vista** y el **controlador**, es decir, por un lado define componentes para la representación de la información, y por otro lado para la interacción del usuario (Reenskaug, y otros, 2009). Este patrón de arquitectura de software se basa en las ideas de reutilización de código y la separación de conceptos, características que buscan facilitar la tarea de desarrollo de aplicaciones y su posterior mantenimiento (yiiframework, 2013).

De manera genérica, los componentes de MVC se podrían definir como sigue:

- El **Modelo**: Es la representación de la información con la cual el sistema opera, por lo tanto gestiona todos los accesos a dicha información, tanto consultas como actualizaciones, implementando también los privilegios de acceso que se hayan descrito en las especificaciones de la aplicación (lógica de negocio). Envía a la 'vista' aquella parte de la información que en cada momento se le solicita para que sea mostrada (típicamente a un usuario). Las peticiones de acceso o manipulación de información llegan al 'modelo' a través del 'controlador' (Burbeck, 1992).
- El **Controlador**: Responde a eventos (usualmente acciones del usuario) e invoca peticiones al 'modelo' cuando se hace alguna solicitud sobre la información (por ejemplo, editar un documento o un registro en una base de datos). También puede enviar comandos a su 'vista' asociada si se solicita un cambio en la forma en que se presenta de 'modelo' (por ejemplo, desplazamiento o *scroll* por un documento o por los diferentes registros de una base de datos), por tanto se podría decir que el 'controlador' hace de intermediario entre la 'vista' y el 'modelo' (Burbeck, 1992).
- La **Vista**: Presenta el 'modelo' (información y lógica de negocio) en un formato adecuado para interactuar (usualmente la interfaz de usuario) por tanto requiere de dicho 'modelo' la información que debe representar como salida (Burbeck, 1992).

Aunque originalmente MVC fue desarrollado para aplicaciones de escritorio, ha sido ampliamente adaptado como arquitectura para diseñar e implementar aplicaciones web en los principales lenguajes de programación. Se han desarrollado multitud de marcos

de trabajo, comerciales y no comerciales, que implementan este patrón; estos marcos de trabajo se diferencian básicamente en la interpretación de como las funciones MVC se dividen entre cliente y servidor (Leff, y otros, 2001).

Los primeros marcos de trabajo MVC para desarrollo web planteaban un enfoque de cliente ligero en el que casi todas las funciones, tanto de la vista, el modelo y el controlador recaían en el servidor. En este enfoque, el cliente manda la petición de cualquier hiperenlace o formulario al controlador y después recibe de la vista una página completa y actualizada (u otro documento); tanto el modelo como el controlador (y buena parte de la vista) están completamente alojados en el servidor. Como las tecnologías web han madurado, ahora existen marcos de trabajo como JavaScriptMVC, Backbone o jQuery que permiten que ciertos componentes MVC se ejecuten parcial o totalmente en el cliente.

3.4. Patrones de diseño

Un patrón de diseño constituye un esquema para refinar subsistemas o componentes. Es una descripción de clases y objetos comunicándose entre sí adaptada para resolver un problema de diseño general en un contexto particular. Un patrón de diseño identifica: clases, instancias, roles, colaboraciones y la distribución de responsabilidades, además de que ayuda a construir clases y a estructurar sistemas de clases. Los patrones de diseño son la base para la búsqueda de soluciones a problemas comunes en el desarrollo de software y otros ámbitos referentes al diseño de interacción o interfaces (Gamma, y otros, 2002).

En la terminología de objetos, el patrón es una descripción de un problema y su solución que recibe un nombre y que puede emplearse en otro contextos; en teoría, indica la manera de utilizarlo en circunstancias diversas.

Los patrones de diseño son soluciones simples y elegantes basadas en la experiencia a problemas específicos y comunes del diseño orientado a objetos. Con el uso de patrones los diseños serán mucho más flexibles, modulares y reutilizables (Larman, 1999).

Resumiendo, un patrón de diseño es (Gamma, y otros, 2002):

- Una solución estándar para un problema común de programación.

- Una técnica para flexibilizar el código haciéndolo satisfacer ciertos criterios.
- Un proyecto o estructura de implementación que logra una finalidad determinada.
- Un lenguaje de programación de alto nivel.
- Una manera más práctica de describir ciertos aspectos de la organización de un programa.
- Conexiones entre componentes de programas.
- La forma de un diagrama de objeto o de un modelo de objeto.

Su uso, brinda indudables ventajas en muchos sentidos (Gamma, y otros, 2002):

- Proponen una forma de reutilizar la experiencia de los desarrolladores, para ello clasifica y describe formas de solucionar problemas que ocurren de forma frecuente en el desarrollo.
- Están basados en la recopilación del conocimiento de los expertos en desarrollo de software.
- Es una experiencia real, probada y que funciona.
- Facilitan la localización de los objetos que formarán el sistema, la determinación de la granularidad adecuada, el aprendizaje y la comunicación entre programadores.
- Especifican interfaces para las clases e implementaciones al menos parciales.

Para el desarrollo, se siguieron los patrones de diseño conocidos como Patrones Generales de Software para Asignar Responsabilidades (*General Responsibility Assignment Software Patterns*, GRASP pos sus siglas en inglés).

3.4.1. Patrones GRASP

Los patrones GRASP describen los principios fundamentales de la asignación de responsabilidades a objetos, expresados en forma de patrones (Larman, 1999).

Existen diversos patrones GRASP utilizados para guiar el proceso de desarrollo y cumplir con los requerimientos de una manera limpia y simplificada.

Experto es un patrón que se usa más que cualquier otro al asignar responsabilidades; es un principio básico que suele utilizarse en el diseño orientado a objetos. Con él no se pretende designar una idea oscura ni extraña; expresa simplemente la "intuición" de que los objetos hacen cosas relacionadas con la información que poseen.

Además, se conserva el encapsulamiento, ya que los objetos se valen de su propia información para hacer lo que se les pide. Esto contribuye a un bajo acoplamiento, lo que favorece al hecho de tener sistemas más robustos y de fácil mantenimiento, alentando con ello definiciones de clases más sencillas y cohesivas que son más fáciles de comprender y de mantener (Larman, 1999).

En el subsistema, cada una de las entidades tiene bien definida sus funciones, las tareas están asignadas a aquellas clases que cuentan con la información para cumplir con ellas.

El patrón **Creador** guía la asignación de responsabilidades relacionadas con la creación de objetos. Una clase B tiene la responsabilidad de crear un objeto de una clase A cuando la clase está contenida en la clase B, la clase B en una agregación o composición de la clase A, la clase B almacena a la clase A, la clase B inicializa los datos de la clase A o la clase B usa la clase A. Entre sus beneficios está que brinda soporte al bajo acoplamiento (Larman, 1999).

El **Bajo Acoplamiento** estimula asignar una responsabilidad de modo que su colocación no incremente el acoplamiento tanto que produzca los resultados negativos propios de un alto acoplamiento y soporta el diseño de clases más independientes, que reducen el impacto de los cambios, y también más reutilizables, que acrecientan la oportunidad de una mayor productividad. No puede considerarse en forma independiente de otros patrones como Experto o Alta Cohesión, sino que más bien ha de incluirse como uno de los principios del diseño que influyen en la decisión de asignar responsabilidades (Larman, 1999).

El acoplamiento tal vez no sea tan importante, si no se busca la reutilización, para dado que el sistema está previsto para su posterior integración a la plataforma PRST, es un aspecto a tener en cuenta durante el desarrollo.

Como el patrón Bajo Acoplamiento, también **Alta Cohesión** es un principio que debemos tener presente en todas las decisiones de diseño: es la meta principal que ha

de buscarse en todo momento. Es un patrón evaluativo que el desarrollador aplica al valorar sus decisiones de diseño.

La cohesión es una medida de cuán relacionadas están las responsabilidades de una clase. Una alta cohesión permite a las clases que están muy relacionadas no realizar un enorme trabajo. Las clases que presentan baja cohesión son difíciles de comprender, reutilizar y conservar (Larman, 1999).

3.5. Modelo de despliegue

El diagrama de despliegue permite indicar la situación física de los componentes lógicos desarrollados. Este modelo es utilizado para capturar los elementos de configuración del procesamiento y las conexiones entre dichos elementos. Cada Hardware es representado como un nodo; un nodo es un elemento donde se ejecutan los componentes, entre ellos existen relaciones que son representados como medios de comunicación tales como HTTPS o TCP/IP.

En el modelo de despliegue del sistema (ver Diagrama 3.2), se puede observar de forma gráfica los elementos que intervienen (Cliente, Servidor Web, Servidor de Base de Datos), el Cliente realiza la petición al Servidor Web mediante el protocolo HTTPS, este solicita los datos al Servidor de Base de Datos mediante el protocolo TCP/IP, luego los procesa y muestra al Cliente.

Diagrama 3.2. Modelo de despliegue (Elaboración propia).

3.6. Pruebas de software

Las pruebas de software consisten en la dinámica de la verificación del comportamiento de un programa en un conjunto finito de casos de prueba, debidamente seleccionados de por lo general infinitas ejecuciones de dominio, contra la del comportamiento esperado. Son una serie de actividades que se realizan con el propósito de encontrar los posibles fallos de implementación, calidad o usabilidad de un programa u ordenador; probando el comportamiento del mismo (Ecured, 2013).

El objetivo de las pruebas es comprobar la integración del sistema de información global, verificando el funcionamiento correcto de las interfaces entre los distintos subsistemas que lo componen y con los que se comunica. Durante la realización de las mismas se hace necesario verificar la cobertura de los requisitos, dado que su incumplimiento puede comprometer la aceptación de la solución desarrollada por el equipo de operación.

3.6.1. Pruebas unitarias

Comienzan con la prueba de cada módulo. Una prueba unitaria es una forma de probar el correcto funcionamiento de un módulo de código. Esto sirve para asegurar que cada uno de estos módulos funcione correctamente por separado. El objetivo que persiguen es aislar cada parte del programa y mostrar que las partes individuales son correctas. Proporcionan un contrato escrito que el fragmento de código debe satisfacer. Estas pruebas aisladas proporcionan cinco ventajas básicas: fomentan el cambio, simplifican la integración, documentan el código, separan la interfaz del código y hacen que los errores estén más acotados y sean fáciles de localizar.

A lo largo del desarrollo del componente se realizaron pruebas unitarias para ir comprobando el funcionamiento del software. Su ejecución, por parte del desarrollador, permite aprovechar las ventajas de compilación, paso a paso, que brinda el entorno de desarrollo.

Las pruebas se evaluarán en base al resultado que arrojen, siguiendo la escala que a continuación se presenta:

- **Prueba satisfactoria**: cuando el resultado de la prueba es exactamente el esperado por el equipo de desarrollo.
- **Prueba parcialmente satisfactoria**: cuando el resultado no es completamente el esperado por el equipo de desarrollo pues muestra resultados erróneos o fuera de contexto.
- **Prueba insatisfactoria:** cuando el resultado de la prueba realizada genera un error de codificación en la aplicación.

El proceso se realizó en tres iteraciones (ver Figura 3.2) obteniendo resultados no satisfactorios en las dos primeras. Para la tercera y última iteración, los resultados fueron satisfactorios en un 100% por lo que se considera cumplido el proceso de validación para las pruebas unitarias.

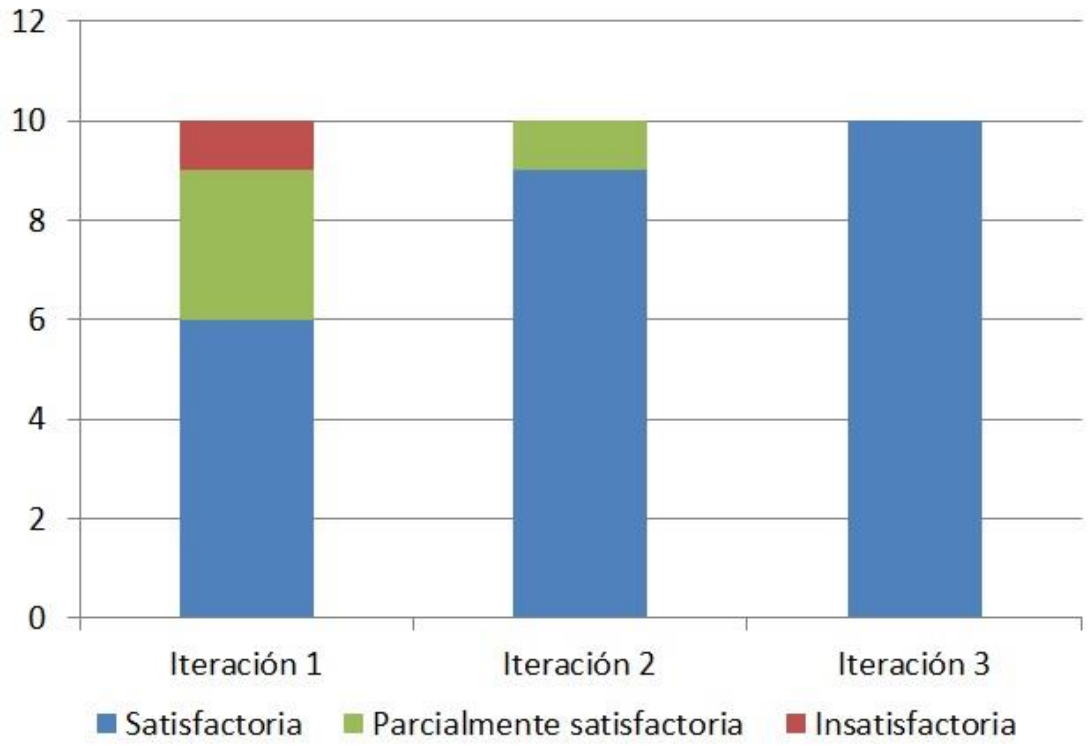


Figura 3.2. Resultado de las pruebas de aceptación (Elaboración propia).

3.6.2. Pruebas de carga y estrés

Las pruebas de estrés son una particularidad de las pruebas no funcionales y sirven para verificar el comportamiento de una aplicación bajo una demanda excesiva.

El objetivo es poder generar una gran cantidad de peticiones a la aplicación y verificar su comportamiento, y de esta manera poder garantizar el número máximo de peticiones bajo las cuales la aplicación, mantiene su funcionamiento normal. Para poder realizar este tipo de pruebas se deben contemplar muchas variables y se vuelve un tanto complejo su implementación y análisis, por lo que se deberá tener un plan bien definido de la arquitectura de servidores / computadoras que las realizarán y un método de recolección y análisis centralizado.

Para realizar las pruebas, se utilizó un servidor con un microprocesador corei3 a 2.40 GHz con 6 GB de memoria RAM. Para elaborar el plan de prueba (ver Tabla 3.1), se tuvo en cuenta aquellos casos de uso críticos para el cliente y se elaboraron los casos de prueba usando datos reales.

Escenario	Escenario de la selección	Carga	Descripción	Tiempo de respuesta esperado	Tiempo de respuesta alcanzado
1. Analizar comunidad de Twitter.	1.1 Recibir colección de tuits en formato JSON.	100	Hacer petición de análisis para una colección con 100 tuits.	0:00:05.000	0:00:04.944
		1000	Hacer petición de análisis para una colección con 1000 tuits.	0:00:50.000	0:00:44.687
		10000	Hacer petición de análisis para una colección con 10000 tuits.	0:08:20.000	0:07:04.853
		20000	Hacer petición de análisis para una colección con 20000 tuits.	0:16:40.000	0:14:32.105
		38460	Hacer petición de análisis para una colección con 38460 tuits.	0:35:00.000	0:33:24.182

Tabla 3.1. Plan de pruebas de carga y estrés (Elaboración propia).

3.7. Consideraciones parciales del capítulo

El proceso de implementación, guiado por la investigación y los artefactos definidos con anterioridad, se desarrolló sin inconvenientes y con fluidez, obteniendo un producto de software listo para las pruebas de funcionamiento. Luego de las pruebas realizadas, han sido detectadas una serie de irregularidades en el funcionamiento del sistema las cuales han sido corregidas obteniendo un producto de software estable y fiable.

CONCLUSIONES

En la presente investigación, se describe un proceso completo para el desarrollo del subsistema de detección de roles de usuario dentro de comunidades de interés de Twitter. Para el mismo, se realizó un estudio del marco teórico en el que se sustenta la investigación centrado en las características deseadas en un sistema para el análisis de redes sociales.

Con el uso de la metodología de desarrollo OpenUp, se generó la documentación necesaria para cumplir con los objetivos y sirve de base para la implementación de versiones posteriores.

La utilización de Django como marco de trabajo para el desarrollo, permitió la reducción de tiempo y esfuerzo por parte del equipo de desarrollo traduciéndose en mayor productividad. Además, se dio cumplimiento a los objetivos propuestos inicialmente obteniendo los siguientes resultados:

- Con el estudio del estado del arte se identificaron los principales requerimientos funcionales y sirvió de base para toda la investigación.
- El proceso de elección de las herramientas y metodologías brindó los resultados esperados.
- Se generaron los artefactos necesarios relacionados con el diseño y la implementación del sistema.
- Se realizaron pruebas al sistema desarrollado de acuerdo a las estrategias definidas, obteniendo resultados satisfactorios.

Hasta el momento, el problema resuelto por la investigación era imposible realizarlo manualmente. Una vez integrada en la PRST, la solución web permitirá realizar los análisis propuestos, reduciendo esfuerzo, tiempo y recursos. Por todo lo antes expuesto, se consideran cumplidos los objetivos trazados en la investigación.

RECOMENDACIONES

Luego de haber hecho las consideraciones finales de todo el trabajo realizado hasta aquí, solo resta recomendar aquellas cuestiones que faciliten materializar el trabajo hecho e incorporar nuevas utilidades relacionadas con el mismo, las cuales se relacionan a continuación:

1. Estudiar otras posibles aplicaciones del análisis de redes sociales que puedan ser aplicadas para automatizar el trabajo sobre Twitter del centro.
2. Desarrollar la Plataforma para la Red Social Twitter para la integración del subsistema.
3. Crear las condiciones finales para el despliegue del sistema.

REFERENCIAS BIBLIOGRÁFICAS

1. **Alhir, Sinan Si. 2003.** *Learning UML.* EE.UU. : O'Reilly & Associates, Inc, 2003.
2. **Balduino, Ricardo. 2007.** *Introduction to Open UP.* 2007.
3. **Bavelas, Alex. 1948.** *A mathematical model for group structures.* s.l. : Human Organization, 1948.
4. **Beck, Kent. 1999.** *Extreme Programming Explained.* 1999.
5. **Bonacich, Phillip y Lloyd, Paulette. 2001.** *Eigenvector-like measures of centrality for asymmetric relations.* s.l. : Social Networks, 2001.
6. **Booch, Grady, Jacobson, Ivar y Rumbaugh, James. 2000.** *El proceso unificado de desarrollo de software.* Madrid : s.n., 2000. ISBN: 84-7829-036-2.
7. **Borgatti, S.P. y Everett, M.G. 2006.** *A graph-theoretic perspective on centrality.* s.l. : Social networks, 2006.
8. **Borgatti, S.P., Everett, M.G. y Freeman, L.C. 2002.** *Ucinet for Windows: Software for Social Network Analysis.* Harvard, MA : Analytic Technologies, 2002.
9. **Burbeck, Steve. 1992.** *How to use Model-View-Controller (MVC).* 1992.
10. **Buschmann, Frank. 1996.** *Pattern-Oriented Software Arquitecture.* EE.UU. : West Sussex : John Wiley & Sons Ltd, 1996. 0-471-95889-7.
11. **Cherven, K. 2013.** *Network Graph Analysis and Visualization with Gephi.* s.l. : Packt Publishing Ltd, 2013.
12. **Fortunato, Santo. 2010.** *Community Detection in Graphs.* s.l. : Physics Reports, 2010.
13. **Framework. 2013.** Ecured. [En línea] 29 de Mayo de 2013. [Citado el: 18 de Octubre de 2014.] http://www.ecured.cu/index.php/Framework.
14. **G. Figueroa, Roberth, J. Solís, Camilo y A. Cabrera, Armando. 2008.** *METODOLOGÍAS TRADICIONALES VS. METODOLOGÍAS ÁGILES.* 2008.

15. **Gamma, Erich, y otros. 2002.** *Patrones de Diseño. Elementos de software orientado a objetos reutilizables.* España : ADDISON-WESLEY, 2002. ISBN: 9788478290598.

16. **González Duque, Raúl. 2008.** mundogeek.net. *Python para todos.* [En línea] 2008. [Citado el: 23 de Septiembre de 2014.] http://mundogeek.net/tutorial-python/.

17. **H. Canós, José, Letelier, Patricio y Carmen Penadés, María. 2003.** *Métodologías Ágiles en el Desarrollo de Software.* 2003.

18. **Hagberg, Aric, Schult, Dan y Swart, Pieter. 2014.** *NetworkX Reference. Release 1.9.1.* 2014.

19. **Hernando, Roberto. 2009.** Blog personal de Roberto Hernando. *rhernando.net.* [En línea] 2009. [Citado el: 21 de Septiembre de 2014.] http://www.rhernando.net/modules/tutorials/doc/ing/met_soft.html.

20. **igraph. 2013.** igraph. *The network analysis package.* [En línea] 2013. [Citado el: 2 de Noviembre de 2014.] http://igraph.org/python/.

21. **Infante, Sergio. 2012.** *Curso Django. El framework para detallistas con deadlines.* 2012.

22. **Jacobson, Ivar. 2000.** *Escribir casos de uso efectivos.* EE.UU. : Alistair Cockburn, 2000.

23. **Jimeng, Sun y Jie, Tang. 2011.** *A survey of models and algorithms for social influence analysis.* Nueva York : Springer, 2011. ISBN 978-4419-8461-6.

24. **JSON. 2014.** json.org. [En línea] 2014. [Citado el: 2 de 12 de 2014.] http://www.json.org/json-es.html.

25. **Kruchten, Philippe y Kroll, Per. 2003.** *The Rational Unified Process Made Easy: A Practitioners Guide to the RUP.* Amsterdam : Addison-Wesley, 2003. ISBN: 9780321166098.

26. **L. Hansen, Derek, Shneiderman, Ben y A. Smith, Marc. 2011.** *ANALYZING SOCIAL MEDIA NETWORKS WITH NODEXL.* Burlington : Elsevier Inc, 2011.

27. **Larman, Craig. 1999.** *UML y Patrones. Introducción al análisis y diseño orientado a objetos.* México : PRENTICE HALL, 1999. ISBN: 970-17-0261-1.

28. **Leff, Avraham y T. Rayfield, James. 2001.** *Web-Application Development Using the Model/View/Controller Design Pattern.* s.l. : IEEE Enterprise Distributed Object Computing Conference, 2001.

29. **Letelier, Patricio y Penadés, M. Carmen. 2010.** *Métodologías ágiles para el desarrollo de software: eXtreme Programming (XP).* Valencia : s.n., 2010.

30. **Lozares, Carlos. 1996.** *La teoría de redes sociales.* Barcelona : s.n., 1996. 08193.

31. **Mañas, José A. 2014.** Prueba de Programas. [En línea] 12 de noviembre de 2014. http://www.lab.dit.upm.es/~lprg/material/apuntes/pruebas/testing.htm.

32. **Muñoz Caro, C., Niño, A. y Vizcaíno Barceló, A. 2002.** *Introducción a la programación con orientación a objetos.* s.l. : Prentice-Hall, 2002.

33. **Newman, Mark. 2005.** *A measure of betweenness centrality based on random walks.* s.l. : Social Networks, 2005.

34. **Orallo, Enrique Hernández. 2002.** *El lenguaje Unificado de Modelado (UML).* Madrid : PEARSON EDUCACIÓN, S. A., 2002.

35. **Visual Paradigm. 2010.** Visual Paradigm for UML. [En línea] 2010. [Citado el: 19 de Octubre de 2014.] http://www.visual-paradigm.com/.

36. **PostgreSQL. 2010.** PostgreSQL. [En línea] 2010. [Citado el: 19 de Octubre de 2014.] http://www.postgresql.org.es/sobre_postgresql.

37. **Pow Sang Portillo, José Antonio. 2003.** *La Especificación de Requisitos con Casos de Uso: Buenas y Malas Prácticas.* Lima-Perú : II Simposio Internacional de Sistemas de Información e Ing. de Software en la Sociedad del Conocimiento-SISOFT 2003, 2003.

38. **Pressman, Roger. 2009.** *Ingenieria de Software: Un enfoque práctico.* Nueva York : McGraw-Hill, 2009.

39. **Ecured. 2013.** Pruebas de software. [En línea] 29 de Mayo de 2013. [Citado el: 19 de Noviembre de 2014.] http://www.ecured.cu/index.php/Pruebas_de_software.

40. **Reenskaug, Trygve y O. Coplien, James. 2009.** The DCI Architecture: A New Vision of Object-Oriented Programming. [En línea] 20 de Marzo de 2009. [Citado el: 02 de 12 de 2014.] http://www.artima.com/articles/dci_vision.html.

41. **Robles, Gregorio y Ferrer, Jorge. 2002.** *Programación eXtrema y Software Libre.* 2002.

42. **Rossum, Guido van y Warsaw, Barry. 2001.** *Guía de estilo del código Python.* 2001.

43. **SGBD. 2013.** Ecured. [En línea] 19 de Diciembre de 2013. [Citado el: 20 de Octubre de 2014.] http://www.ecured.cu/index.php/Sistema_Gestor_de_Base_de_Datos..

44. **Sommerville, Ian. 2005.** *Ingeniería del Software. Séptima edición.* Madrid : PEARSON EDUCACIÓN, S. A, 2005.

45. **Tsvetovat, Maksim. 2011.** *Social Network Analysis for Startups: Finding connections on the social web.* California, EE.UU. : O'Reilly Media, 2011.

46. **Umadevi, V. 2013.** *ESTUDIO DE CASO: MEDIDAS DE CENTRALIDAD EN EL ANÁLISIS DE LA RED DE CO-AUTORES.* Bangalore, Karnataka, India : s.n., 2013.

47. **yiiframework. 2013.** Best MVC Practices. *yiiframework.* [En línea] 2013. [Citado el: 02 de 12 de 2014.] http://www.yiiframework.com/doc/guide/1.1/en/basics.best-practices.