UNIVERSIDAD CENTRAL DE VENEZUELA

FACULTAD DE CIENCIAS

ESCUELA COMPUTACIÓN

CENTRO DE COMPUTACIÓN DISTRIBUIDA Y PARALELA

SISTEMA DE EXTRACCIÓN MASIVA DE DATOS PARA LA CONSTRUCCIÓN DE UNA ESTRUCTURA DE RED SOCIAL A PARTIR DE FUENTES DE DATOS BASADAS EN INTERFACES DE PROGRAMACIÓN DE APLICACIONES (API)

Trabajo Especial de Grado presentado ante la ilustre
Universidad Central de Venezuela por

Br. Ashtar Márquez
C.I. 20.754.339

Br. Carla Navas
C.I. 20.026.343

Tutores: Jesús Lares y José Sosa.

Caracas, mayo 2017.

Acta

UNIVERSIDAD CENTRAL DE VENEZUELA
FACULTAD DE CIENCIAS
ESCUELA COMPUTACIÓN
ACTA DEL VEREDICTO

Quienes suscriben, Miembros del Jurado designado por el Consejo de la Escuela de Computación para examinar el Trabajo Especial de Grado, presentado por los bachilleres Ashtar Márquez C.I. 20.754.339 y Carla Navas C.I. 20.026.343 , con el título "Sistema de extracción masiva de datos para la construcción de una estructura de red social a partir de fuentes de datos basadas en interfaces de programación de aplicaciones (API)", a los fines de cumplir con el requisito legal para optar al título de Licenciado en Computación, dejan constancia de lo siguiente:

Leído el trabajo por cada uno de los Miembros del Jurado, se fijó el día 09 de junio de 2017, a las 10:00 a.m para que sus autores lo defendieran en forma pública, en el aula, lo cual estos realizaron mediante una exposición oral de su contenido, y luego respondieron satisfactoriamente a las preguntas que les fueron formuladas por el Jurado, todo ello conforme a lo dispuesto en la Ley de Universidades y demás normativas vigentes de la Universidad Central de Venezuela. Finalizada la defensa pública del Trabajo Especial de Grado, el jurado decidió aprobarlo.

En fe de lo cual se levanta la presente acta, en Caracas el 09 de Junio de 2017, dejándose también constancia de que actuó como Coordinador del Jurado el Profesor Tutor Jesús Lares.

Prof. Jesús Lares
Tutor

Prof. José Sosa
Co-Tutor

Prof. Hector Navarro
Jurado

Profa. Mercy Ospina
Jurado

Jurado Jurado

Dedicatoria

Por parte de Carla Navas,

Dedicado a mi madre Ana y mi padre Carlos.

Por parte de Ashtar Márquez,

Dedicado a mi madre María, a mi padre Richard, a mi hermano John y a mi segunda madre Carmen.

Agradecimientos

Primeramente a Dios, a esa fuerza divina que quita los obstáculos, mueve las piezas y nos trajo personas valiosas para transcurrir con éxito nuestra carrera universitaria.

A nuestras madres (Ana y María) y padres (Carlos y Richard) por la paciencia, por el apoyo incondicional, por vivir con nosotras algunas de las noches en vela y sobre todo por cada uno de sus esfuerzos para darnos el regalo de la educación en todos los niveles.

A nuestros tutores Jesús Lares y José Sosa, las mentes brillantes detrás de este trabajo especial de grado, quienes nos prestaron su visión para entender lo que aún no podíamos ver. Por la plena disposición para orientarnos, por la paciencia y por la confianza dispuesta sobre nosotras para lograr la realización del proyecto. Y sobre todo por a*post*ar a la excelencia académica y profesional dentro de nuestra escuela.

A la Universidad Central de Venezuela, a la Escuela de Computación y a todos y cada uno de los profesores que contribuyeron en nuestra formación académica, con un reconocimiento especial al Profesor Robinson Rivas por rescatarnos de los pasillos de nuestra facultad e impulsarnos a crecer como futuros profesionales a través de diversas actividades que desafiaron nuestras habilidades pero que sin duda aportaron un gran crecimiento, no solo a nivel profesional sino como mejores seres humanos.

A nuestros compañeros, Carlos Rodríguez, Sherezada Moubayyed y Juan Negretti, que fueron fundamentales en cada uno de los semestres, gracias por las horas de estudio y ayuda desinteresada. Donde entre estrés y risas logramos sacar con éxito cada una de nuestras materias. Hoy nos une el lazo inquebrantable de la amistad.

A Omar González que desarrollo un rol como tutor externo, guiándonos con su conocimientos en la lógica de negocio de nuestro proyecto, por la ayuda a nivel técnica con las herramientas de desarrollo utilizadas.

Por parte de Ashtar Márquez

Un agradecimiento especial a mi segunda madre Carmen Colmenares, por su entrega, confianza y apoyo en todo momento.

Por parte de Carla Navas

A mis hermanas Gabriela y Carolina por siempre impulsarme a ser mejor, por ser apoyo, por sus merienditas de madrugada y por su insistencia infinita en que culminará con esta etapa.

Y especialmente a Omar González por su apoyo incondicional en los momentos de angustia y frustración, siempre tuvo la paciencia y las palabras indicadas, gracias.

Resumen

Título: "Sistema de extracción masiva de datos para la construcción de una estructura de red social a partir de fuentes de datos basadas en interfaces de programación de aplicaciones (*API*)"

Autor: Ashtar Márquez y Carla Navas

Tutor: Jesús Lares y José Sosa

En la actualidad existe un interés creciente en encontrar valor detrás de los millones de datos que generan los usuarios diariamente en internet. Obtener los datos y almacenarlos de manera adecuada se torna como un gran desafío. Si bien las redes sociales pueden ser una mina de oro a la hora de obtener datos para generar información, es un reto extraer datos válidos y confiables. Una interfaz de programación de aplicaciones (API), es una fuente de datos externa a las redes sociales que permite realizar peticiones en nombre del usuario, tales como leer información personal, intereses o contactos. Sin embargo presentan una cantidad de restricciones en cuanto al acceso de los datos, número de consultas a un servicio, cantidad de datos que se descargan por consultas, y largos tiempos de espera para realizar de nuevo las peticiones. Es por ello que los estudios sobre las redes sociales se realizado sobre muestras de los datos ya que no se ha encontrado solución a las limitaciones de los API's y en algunos casos no se cuenta con la estructura correcta para almacenar grandes volúmenes de datos de este tipo. El objetivo de este trabajo especial de grado es diseñar la pieza que falta, una herramienta que permita extraer los datos de una red social para construir su estructura, a pesar de las limitaciones de las API para posteriormente almacenar estos grandes volúmenes de datos en una base de datos de grafos reduciendo los tiempos de respuesta con el uso de programación paralela y distribuida. Se utilizó la metodología de desarrollo Scrum con algunas variantes y se obtuvo como resultado un grafo social.

Palabras clave: Big Data, Grafo social, Redes sociales, Cómputo paralelo, API.

Índice general

Índice de figuras

Índice de tablas

CAPÍTULO 1

Introducción

La información se ha convertido en el petróleo del siglo XXI. En la actualidad existe un interés creciente en encontrar valor detrás de los millones de datos que generan los usuarios diariamente en internet. Diversas son las entidades interesadas, pueden ser educativas, estatales, empresas u organizaciones privadas. El objetivo es descubrir información y generar conocimiento de los datos. El propósito para estudiar los datos puede ser muy variado, Identificar grupos, aplicar métricas, estudiar patrones de comportamiento de los usuarios, realizar campañas de marketing, análisis de sentimiento, entre otros.

Cuando hablamos de procesamiento de datos, automáticamente se piensa en muchos números, muchas técnicas estadísticas, conclusiones cuantitativas, etc. Esto es así, sin embargo hay mundo más allá de los números. En menos de una década los medios sociales han revolucionado el mundo, y han dotado a los usuarios con la capacidad de estar en contacto con otros millones de usuarios ubicados en cualquier parte del mundo. Las redes se encuentran por todas partes, nos rodean y formamos parte de ellas, por lo tanto analizarlas puede ser de gran utilidad.

Se incrementa la necesidad del análisis de redes sociales o estructuras de grafos, esto no quiere decir que sea solo el estudio de contenido de *Social Media*. Es un estudio numérico, algebraico, de una representación de conocimiento en formato de grafo. Un campo que mezcla la sociología y las matemáticas (el álgebra de grafos) en el que hay actores o entidades que interactúan, pudiendo representar estas acciones a partir de un grafo.

En este punto, comienzan a surgir nuevos desafíos, ¿Cómo extraer los datos? La respuesta pudiese ser a partir de fuentes basadas en interfaces de programación de aplicaciones (API's), sin embargo no es tarea trivial ya que las API's tiene una serie de limitaciones de extracción lo que nos lleva a querer implementar nuevas estrategias.

Ahora, una vez que se tienen los datos ¿Cómo almacenar y manipular estos volúmenes de datos? Para ello llegan las bases de datos no relacionales y los sistemas Big Data.

En este trabajo especial de grado se plantea el desarrollo de una herramienta basada en 3 componentes, un módulo administrador, un módulo de clientes y un módulo coordinador. La interconexión de estos tres componentes permitirá consumir el API del caso de estudio superando sus limitaciones, para posteriormente almacenar estos datos en una base de datos orientada a grafos optimizando los tiempos de respuesta tanto en la descarga como en los accesos a la base de datos.

En el capítulo 2 se encuentra detallado en el planteamiento del problema, en el capítulo 3 un marco teórico donde se profundiza diversos conceptos, capítulo 4 la metodología planteada, capítulo 5 el marco aplicativo y finalmente el capítulo 6, las conclusiones asociadadas al presente trabajo especial de grado.

CAPÍTULO 2

Problema de investigación

2.1. Planteamiento del problema

Con la web 2.0 nació un fenómeno social surgido a partir del desarrollo de diversas aplicaciones en Internet que permiten interactuar con el resto de los usuarios o aportar contenido que enriquezca la experiencia de navegación. Millones de usuarios interconectados generan una cantidad interesante de datos diariamente. Estos datos son prioridad para distintas entidades. Algunas empresas acumulan gran cantidad de estos datos e intentan innovar con ellos sobre sus consumidores, con la convicción de que cuantos más datos se tengan, más completa será la visión que se tiene del consumidor, de sus intereses y gustos.

Sin embargo estos datos crecen de manera exponencial en corto tiempo y mucho de los entes interesados no son capaces de gestionarlos utilizando herramientas tradicionales tales como las ampliamente conocidas bases de datos relacionales ya que estas no fueron diseñadas para manejar estos volúmenes de datos.

Las empresas obtienen los datos y luego se enfrentan con una gran cantidad de problemas que hacen que no puedan comprenderlos, como por ejemplo medir lo que están haciendo, establecer servicios derivados de los datos o transformarlos en algo valioso que realmente les sirva para conectar con las audiencias.

Como consecuencia de los grandes volúmenes de datos los tiempos de acceso y manipulación son elevados, hay un desfase demasiado alto desde el momento en el que se produce el dato que deben analizar hasta el momento exacto en el que pueden realmente leerlo por lo que se ven obligados en realizar sus análisis sobre una muestra de los datos y no con la totalidad.

Para obtener los datos se utilizan diversas fuentes, de esta depende la calidad de los datos. Entre las fuentes de datos se encuentra cada vez con más auge las redes sociales, las cuales en menos de una década han revolucionado el mundo, con millones de usuarios, relaciones e interacciones.

Las empresas se dirigen a estas fuentes externas donde están involucrados sus clientes para realizar análisis de sentimiento, identificar patrones de comportamiento, hábitos de compra, utilización de servicios, gustos para campañas de publicidad y mercadeo [43]. Cada vez con mayor auge las API's se muestran como una fuente externa inagotable para obtener los datos de las redes sociales, estas incluyen una variedad de funciones que nos permiten tener datos de los usuarios pero a su vez presentan una serie de limitaciones en nombre seguridad para sus usuarios, aliviar la carga de la infraestructura de la red social o reducir el tiempo de interrupción del servicio.

Entonces, ¿Cómo deberían enfrentarse las diferentes entidades al desafío de obtener los datos y almacenarlos para su posterior análisis?

Este trabajo especial de grado se presenta una herramienta web que permite extraer y almacenar la estructura de una red social superando las limitaciones de los API's haciendo uso de computo paralelo y disminuyendo así los tiempos de respuesta. Se comenzó con un conjunto de cuentas de usuarios de una red social a la que se denominan *cuentas semillas* en este documento y a partir de estas cuentas comenzar a construir la estructura de la red en base a las relaciones de amistad de los usuarios.

2.1.1. Justificación

2.1.2. ¿Por qué es un problema?

Es un problema dado que las API's presentan una serie de limitaciones en cuanto al número de peticiones a los servicios para obtener amigos y seguidores, la cantidad de cuentas que puedes obtener en una consulta a un servicio y los tiempos de respuesta para descargar los datos. Esto trae como consecuencia que los estudios sobre las redes se realicen sobre muestras de datos ya que obtener la totalidad de los usuarios implica largos periodos de tiempo, meses e incluso años.

2.1.3. ¿Para quién es un problema?

En primera instancia se pudiese decir que este problema afecta en mayor medidas a las empresas, ya que un gran porcentaje de las ventas actuales se hacen a través de internet por lo que a las empresas se les ha despertado el interés de conocer

con mayor detalle a su cliente/consumidor para de esta manera aplicar herramientas de marketing personalizadas con la Idea de mantener la fidelidad de su cliente a largo plazo.

Sin embargo es un posible problema para cualquier usuario u organización que le interese y/o necesite realizar análisis de redes sociales que representan en su totalidad grandes volúmenes de datos.

2.1.4. ¿Desde cuándo es un problema?

Principalmente desde que comenzó a haber mayor penetración de Internet, el nacimiento de la WEB 2.0 combinado con uso excesivo de las redes sociales. El crecimiento exponencial de uso de teléfonos inteligentes y tabletas, y la generación sin control de millones de datos, aunado con la practicidad al momento de adquirir un producto o servicio a través de cualquier dispositivo con internet. Esto trajo como consecuencia que diversas entidades se interesaran en el estudio de estos datos.

2.2. Objetivos de la investigación

2.2.1. General

Desarrollar una herramienta web para la construcción de la estructura de una red social a partir de fuentes de datos externas, que reduzca los tiempos de obtención de los datos con el uso de programación paralela y distribuida.

2.2.2. Específicos

- Seleccionar la metodología para el desarrollo de la herramienta web que nos permita definir un marco de trabajo.
- Seleccionar una red social y su API como caso de estudio para particularizar el desarrollo.
- Desarrollar un módulo de administración que permita crear y eliminar *cuentas semillas* y proporcionar el *nivel de profundidad*.
- Desarrollar una aplicación *cliente* que se conecte con el API de caso de estudio.

- Implementar un coordinador que maneje una conexión con una base de datos de grafo, los clientes, paralelice las *cuentas semillas* y los *clientes.*
- Realizar pruebas de software que permitan verificar el funcionamiento correcto de la herramienta.

2.3. Alcance

El presente trabajo de investigación tiene como alcance:

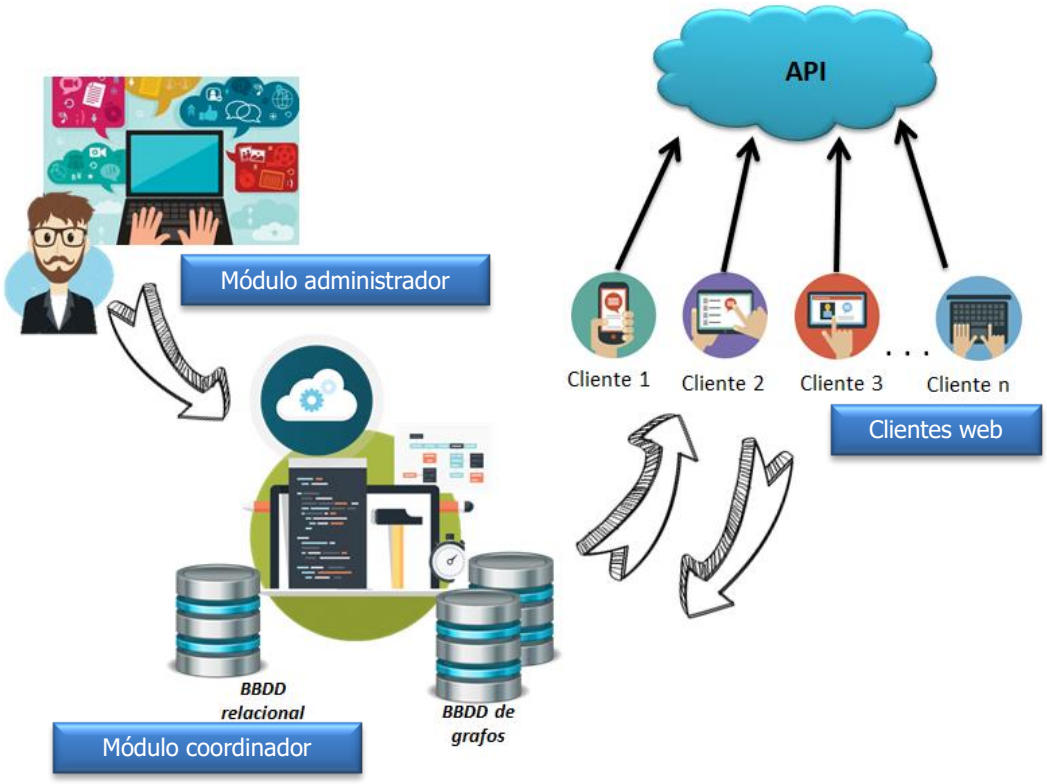
- Implementar el módulo de administración para crear semillas, definir nivel de profundidad y enviar semillas al coordinador para que sean distribuidas a los clientes.
- Implementar un cliente que se autentique, consuma los servicios del API del caso de estudio y se conecte con el coordinador para enviar los datos descargados.
- Implementar un coordinador que se conecte con el módulo de administración y los clientes, se encargue de recibir las semillas del usuario administrador, enviar semillas a los clientes, se conecte con una base de datos orientada a grafos, maneje nodos y relaciones.
- Construir el grafo de la red social a partir de n semillas y n clientes en paralelo
- Integrar los componentes de módulo de administración, clientes y coordinador.

No se contemplara en la realización de este trabajo especial de grado:

- Montar el sistema sobre una plataforma Hadoop
- Realizar análisis sobre el grafo obtenido
- Aplicar algoritmos de minería de datos sobre el grafo resultante.
- Visualizar el grafo mediante una interfaz gráfica especializada

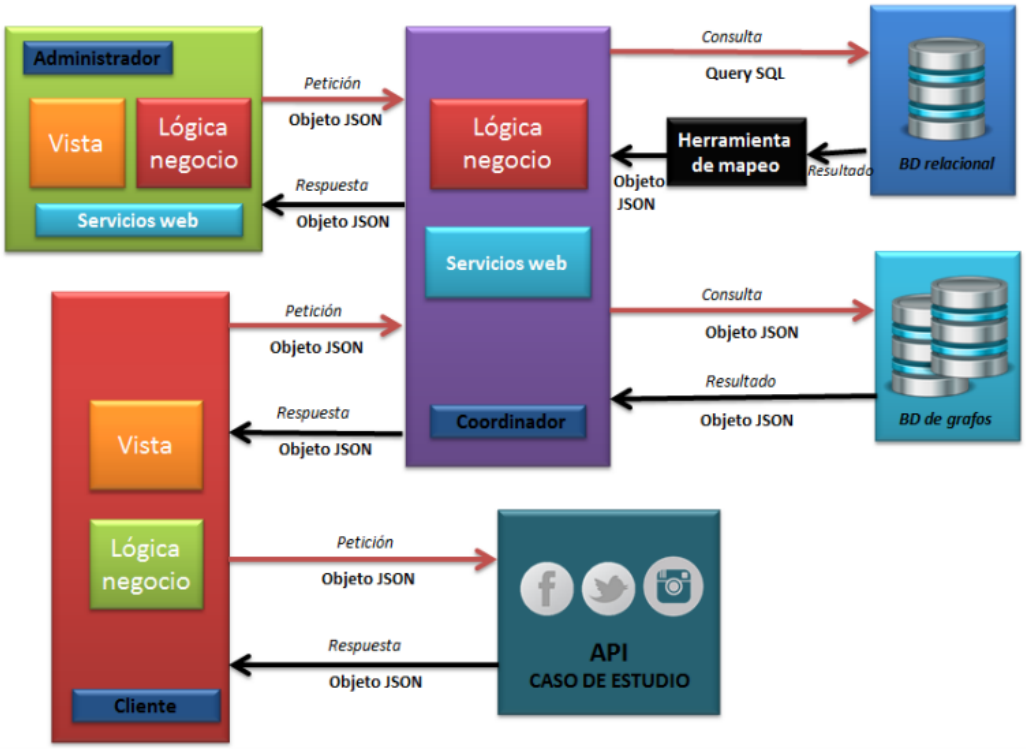
2.4. Arquitectura propuesta de la solución

Figura 1: Arquitectura genérica



En la **figura 1** se muestra una arquitectura orientada a servicios (SOA) compuesta por tres componentes un módulo administrador manejado por un usuario experto, un conjunto de clientes web y un módulo coordinador encargado de paralelizar y de almacenar los datos.

Figura 2: Arquitectura específica



La estrategia utilizada para superar las limitaciones de las API's es manejar distintos clientes web conectándose y descargando datos de manera paralela de una API. Estos clientes serán planificados por un módulo coordinador que se encarga distribuir de manera eficiente las *cuentas semillas* (estas también son paralelizadas) a cada uno de ellos.

Para obtener los datos de las *cuentas semillas** y el *nivel de profundidad*** contamos con un módulo administrador manipulado por un usuario experto que agrega o elimina *cuentas semillas* y determina el *nivel de profundidad*. En base a estos parámetros el módulo coordinador sabrá que datos enviar a los clientes y hasta que nivel de recursión se debe ejecutar para construir la red deseada.

***Cuenta semilla:** Se considera en este documento como cuentas semillas a un conjunto de cuentas de usuarios de una red social a partir de la cual se comenzará a construir la estructura de la red social.

****Nivel de profundidad:** Se considera en este documento lo siguiente:

Dado una cuenta semilla X se tiene que:

Nivel cero es equivalente a descargar los datos de los amigos (*friends*) y seguidores (*followers*) de la cuenta semilla X.

Nivel uno es equivalente a tomar por cuentas semillas a los amigos y seguidores obtenidos en el nivel cero y comenzar a descargar los amigos y seguidores de estos.

Ejemplo:

Sea una cuenta semilla X= @maria

@maria tiene 3 seguidores {@juan, @jesus, @luisa}

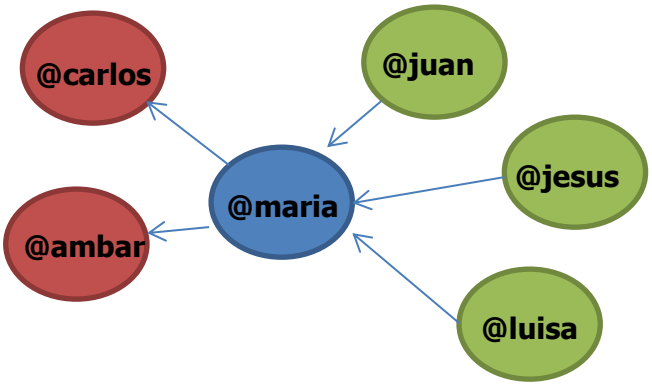
@maria sigue a 2 usuarios {@carlos @ambar}

Nivel=0 :

Cuentas semillas {@maria}

Se obtienen datos de {@juan, @jesus, @luisa,@carlos @ambar}

Figura 3: Ejemplo de nivel de profundidad igual a cero

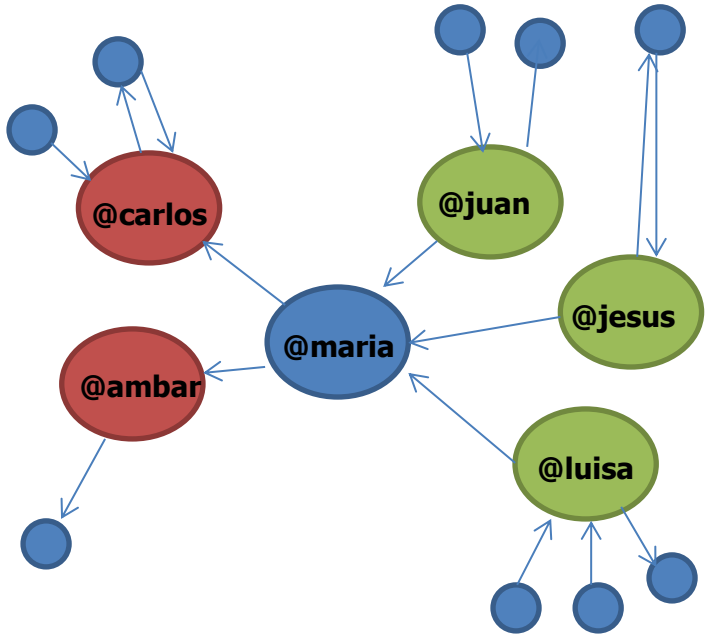


Nivel=1

Cuentas semillas {@juan, @jesus, @luisa, @carlos @ambar}

Se obtienen datos de los amigos y seguidores de {@juan @jesus, @luisa, @carlos @ambar}

Figura 4: Ejemplo de nivel de profundidad igual a cero



Capítulo 3

Marco Teórico

3.1. Ciencia de datos

La Ciencia de Datos (*Data Science*) es el estudio de la extracción de conocimiento generalizable a partir de datos. Al referirnos a este término se involucran conocimientos de uno o más dominios (por ejemplo finanzas, medicina, geología, entre otros) tomando en cuenta por supuesto, aspectos computacionales. Incluye técnicas científicas tales como la prueba de hipótesis y la validación de los resultados confiables. Adicional a cualquier enfoque involucra matemáticas y estadísticas. Incluye el aprendizaje automatizado (*machine learning*), inteligencia artificial o algoritmos de descubrimiento de conocimiento (*knowledge discovery*) y técnicas de procesamiento paralelo y distribuido. Según los requerimientos implica la visualización y creación rápida de prototipos para el desarrollo de *software* que permitan predecir o detectar patrones sobre los datos almacenados y apoyar en el proceso de toma de decisiones. [2]

Pudiésemos decir, que hace referencia a las técnicas necesarias para manipular y tratar la información desde un punto de vista estadístico/matemático. La ciencia de datos está basada en algoritmos, aplicados al problema de Big Data entre otros. Incorporar la figura del científico de datos (*Data Scientist*) en la organización implica dejar atrás las conocidas limitaciones de la minería de datos (*data mining*). Esta evolución traspasa las fronteras de una simple consulta (*query*), hallando correlaciones, aplicando algoritmos más complejos y proporcionando unos niveles de visibilidad que transforman por completo el contacto de la empresa con su entorno, otorgándole la capacidad , por primera vez, de descubrir y estudiar su oportunidades.

3.1.1. Grandes volúmenes de datos (Big Data)

El concepto Big Data hace referencia a los sistemas que manipulan grandes conjuntos de datos, también conocido s como conjuntos de datos (*data sets*). Entre sus principales *cualidades* se encuentran la heterogeneidad y la volatilidad, como datos que son. Sin embargo, son su volumen y su velocidad de generación las que plantean

mayores dificultades a la hora de trabajar a un nivel Big Data en el entorno empresarial. Los retos tienen que ver con:

- Captura de datos.
- Almacenamiento de tales volúmenes de información.
- Capacidad de realizar búsquedas eficientes.
- Compartición.
- Posibilidad de llevar a cabo análisis efectivos.
- Visualización de los datos.

Cuando nos referimos a grande volúmenes de datos, debemos resaltar que va más allá de las bases de datos relacionales y los almacenes de datos (*data warehouses*), involucra cómputo distribuido en múltiples *servidores* donde se entremezclan gestión y procesamiento de datos y en definitiva permite obtener resultados que no estaban disponibles con los enfoques anteriores de manejo de datos, o que en su defecto llevarían sustancialmente mucho más tiempo (tiempo de ejecución o latencia). [3]

Al hablar del término Big Data, se tienen relacionados distintos fenómenos los cuales ayudan a explicar mejor el concepto de Big Data. Estos fenómenos están separados en cinco variables, mejor conocidas como las 5 V de Big Data:

- **Volumen:** se refiere a la cantidad de data generada cada segundo. Dependiendo de las capacidades de procesamiento, grandes volúmenes pueden ser Terabytes, como para otros grandes volúmenes pueden ser Zettabytes. Para el procesamiento de estas cantidades no se pueden utilizar bases de datos tradicionales debido a que su rendimiento es deficiente y no proveen técnicas para el particionamiento de estas.
- **Velocidad**: está relacionado con la velocidad a la cual es generada la data. Existen herramientas las cuales permiten el análisis de estos datos sin tener siquiera que almacenarlos.
- **Variedad**: se refiere a los diferentes tipos de dato que se pueden utilizar. Los datos aparte de tener diferentes tipos, también pueden ser estructurados, semi-estructurados y no estructurados.
- **Veracidad**: este término es afín con la credibilidad y lo correcto de los datos. Algunas veces, dependiendo de la fuente de los datos, la calidad y certeza de los datos no pueden ser controlados y estos casos para el momento de su

análisis podrían entregar valores erróneos. Normalmente los datos erróneos son creados gracias al poco control que se puede tener sobre un humano.

- **Valor**: relacionado con el valor oculto en los datos. Así se tengan grandes volúmenes de datos, si estos no poseen valor alguno, no se podría finalmente obtener información valiosa de estos.

3.1.2. Campos en los que es utilizado:

Los científicos e investigadores han analizado datos desde ya hace mucho tiempo, como lo hemos explicado con anterioridad, lo que ahora representa un gran reto, es la escala en la que estos datos son generados.

Esta explosión de "grandes datos" está transformando la manera en que se conduce una investigación adquiriendo habilidades en el uso de Big Data para resolver problemas complejos relacionados con el descubrimiento científico, investigación ambiental y biomédica, educación, salud, seguridad nacional, matemáticas, computación, estadística, ingeniería de datos, modelos probabilísticos, entre otros. [4]

3.1.3. Fuentes de datos

Las fuentes para obtener los datos se pueden clasificar en:

- **Internas:** Son las fuentes que se encuentran dentro de la empresa. Ejemplos de fuentes internas pueden ser las bases de datos internas (que, por ejemplo, permiten obtener información referente a los clientes), los estados financieros (que, por ejemplo, permiten obtener información referente a la situación financiera de la empresa), los registros de inventarios, registros de ventas, registros de costos, el personal de la empresa, etc.
- **Externas:** Son las fuentes que se encuentran fuera de la empresa. Ejemplos de fuentes externas pueden ser Internet (páginas web de organismos gubernamentales, páginas web de la competencia, etc.), oficinas de gobierno, locales de la competencia, proveedores, distribuidores, clientes, diarios, revistas, publicaciones, redes sociales, etc.; en donde se puede obtener información referente a estadísticas, tendencias, preferencias, etc. [52]

3.2. Medios sociales

Los medios de comunicación sociales o simplemente medios sociales (social media en inglés), son plataformas de comunicación en línea donde el contenido es creado por los propios usuarios mediante el uso de las tecnologías de la WEB 2.0, que facilitan la edición, la publicación y el intercambio de información.

Los profesores Kaplan y Haenlein definen medios sociales como: «Un grupo de aplicaciones basadas en Internet que se desarrollan sobre los fundamentos Ideológicos y tecnológicos de la WEB 2.0, y que permiten la creación y el intercambio de contenidos generados por el usuario»

Los medios sociales son ricos en la influencia y la interacción entre pares y con una audiencia pública que es cada vez más «inteligente» y participativa. El medio social es un conjunto de plataformas digitales que amplía el impacto del boca a boca y también lo hace medible y, por tanto, rentabilizable por medio de la mercadotecnia de medios sociales y el CRMsocial. [5]

3.2.1. Aspecto conversacional

Los medios sociales utilizan la tecnología como plataforma pero su calidad y proyección dependen principalmente de las interacciones entre las personas, de la riqueza de las conversaciones, de la fecundidad del diálogo y la calidad de la colaboración entre los participantes, para construir sentido compartido de lo que se está realizando.

Los servicios de los medios sociales crean oportunidad es para el uso de la inteligencia colectiva de sus usuarios. Los reclamos o denuncias de mal uso son rápida mente atendidas por la comunidad de forma que esos debates son públicos y compartido s por todos los participantes.

La velocidad de los medios sociales, la amplitud, el alcance y la profundidad que pueden alcanzar se pueden apreciar al ver cómo se maneja un caso de mal uso del medio por algún usuario. La inteligencia colectiva actúa como el sistema inmunológico del cuerpo humano y es usada para validar, corregir, mejorar o autenticar a los usuarios y sus diferentes comentarios y declaraciones. Un lenguaje directo y franco, una voz humana es una parte importante del estilo y del lenguaje de los medios sociales de comunicación.

Sin embargo, los medios sociales no son una panacea[2], pero el hecho de que el público pueda participar raíz mente en ellos - de hecho participan - a través de agregar comentarios, mensajes instantáneos o incluso la complementación del contenido con historias y experiencias propias le dan su principal poder y atractivo.

2 La palabra panacea proviene de la voz griega *panakos* y significa 'remedio para todo'. [5]

3.2.2. Clasificación

- **Medios de comunicación:**

  Es el tipo de medio donde toda la información se genera de los usuarios, tal como explicamos anteriormente. En esta categoría se encuentran las redes sociales (Facebook), los blogs (*Blogger*), los foros, *microblogging* (Twitter), sistemas de agregación y redes sociales basadas en localización (*Foursquare*).

- **Medios de colaboración:**

Es una red basad en la colaboración de un conjunto de personas, que aportan trabajo intelectual a un proyecto con un objetivo común al grupo. El concepto es simple "más cerebros piensan más que menos". Ejemplo de este tipo de medios son las wiki (W*ikipedia*), marcadores sociales (*Google Reader*), noticias, sitios de opinión y sitios de comercio social.

- **Medios multimedia:**

Son el tipo de redes don el usuario puede hacer uso interactivo de la multimedia tradicional (textos e imágenes, animación, sonidos, v*ideo*, etc). El usuario tiene libre control acerca de lo que desea ver y cuando. Por ejemplo cuando se comparten imágenes (*Instagram*), se comparten v*ideo*s (*YouTube*), emisiones en directo (*Skype*) o compartir música (*SoundCloaud*).

- **Medios de entretenimiento:**

Se conoce como entretenimiento al conjunto de *actividad*es que permite a los seres humanos emplear su tiempo libre para divertirse, evadiendo temporalmente sus preocupaciones, tales como mundos virtuales (*Second Life*), juegos en línea (*Wordl of Warcraft*), entre otros. [5]

3.2.3. Redes Sociales

Ya hemos mencionado que las redes sociales son un tipo de medio de comunicación social, sin embargo en esta parte ampliaremos este término en pro de en *post*eriori dar respuesta a la siguiente pregunta ¿Cuál es la función de aplicar minería de datos a las redes sociales?

A lo largo de la historia los humanos hemos estado rodeados de redes sociales en el mundo no virtual, ya que, por naturaleza el ser humano ha buscado el apoyo de las personas que le rodean, éstas le pueden resultar de ayuda para satisfacer sus necesidades básicas, de seguridad, protección y/o afecto.

Como seres humanos vivimos inmersos en un conjunto de vínculos interpersonales que afectan nuestra vida. Así, el comportamiento de cada individuo afecta y a su vez se ve afectado por las interacciones sociales en las que participa.

El problema de las redes sociales en el mundo no virtual es que la mayoría de las conexiones entre las personas están ocultas, la red puede tener un gran potencial pero sólo es tan valiosa como las conexiones y las personas que una persona puede ver. El verdadero valor de las conexiones es explotado por las redes sociales en internet.

Las redes sociales en internet son sitios WEB donde las personas crean un perfil (una página WEB personal) para *post*eriormente agregar el perfil de sus amigos, personas a quienes invita u otros usuarios que ya pertenecen a la red social. De esta manera permite así la interacción entre personas que no necesariamente se conocen pero que comparten intereses, preocupaciones o necesidades. De donde se puede deducir que las redes sociales no sólo sirven para mostrar fotografías o documentos.

Las redes se pueden dividir en tres categorías:

- **Generales**: Son aquellas que se dirigen de forma indiscriminada a un público objetivo muy numeroso, tales como Facebook. En esta categoría también se incluyen las generales-sectoriales que abarcan a personas con perfiles, intereses, aficiones, etc. más concretas, cada una de ellas se centra en una temática diferente. Para la empresa, el uso eficaz de ambas redes es importante.

- **Profesionales**: Son las redes que pueden facilitar la contratación de personal, así como la búsqueda de nuevos clientes y socios, si son correctamente utilizadas por la compañía. La idea principal es ayudarle a multiplicar sus contactos, gracias a los contactos de sus contactos. Es decir, los amigos de sus amigos son mis amigos. Este tipo de requieren de ganar credibilidad y calidad de sus contactos y sobre todo la participación. Ejemplo este tipo de redes en *LinkedIn*.
- **Temáticas**: Son las redes especializadas en un solo tema, adicional a las características de las redes incorporando herramientas adicionales de información especializada tales como la posibilidad de comprar comida en línea en las redes sociales de alimentos, la posibilidad de armar equipos de fútbol Idea les creando también sus colores en las redes sociales dedicadas al fútbol o la posibilidad de comprar y escuchar música en las redes sociales orientadas a la música. [5]

3.2.4. ¿Por qué estudiar las redes sociales?

Muchas empresas comenzaron a darse cuenta de que usar diversas técnicas (Minería de datos por ejemplo) para obtener información a partir de lo que mostramos de nosotros mismos en las redes sociales es sinónimo de mejoras sustanciales en temas como publicidad, contratación, ventas, etc. Gran parte se basa en la Idea de incrementar la satisfacción de los usuarios, quienes reciben ofertas de mayor interés al haber sido diseñadas previamente en función de los datos obtenidos. Los usuarios pueden conectarse con personas similares a ellos, les llegan promociones más acorde a sus gustos, necesidades y posibilidades, lo que mejora el servicio en varios aspectos.

Aunque el mayor auge dentro de la minería de datos aplicada a redes sociales está en el campo de la publicidad no es la única. Esta técnica está siendo utilizada desde hace algunos años atrás para conocer más acerca de sus futuros empleados. En una entrevista personal el experto en recursos humanos intenta hacerse una Idea de cómo es el entrevistado y de cómo podría responder ante el trabajo. Adicionalmente la empresa encarga al experto técnico analizar cuáles son conocimientos y capacidad es.

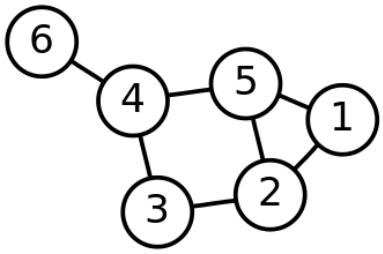
Adicionalmente ahora la empresa les facilita a los expertos un sistema que complementará el proceso, la minería de datos aplicada a las redes sociales. La mayor parte de la investigación en este tópico ese enfoca en la caracterización del cliente y el

estudio de sus aplicaciones más directas. Incluso las herramientas no son estáticas, las empresas apuntan a tener un grupo de trabajo que genere nuevas técnicas en la minería de datos y análisis de las redes para generar conocimiento estratégico que contribuya a la caracterización del caso de estudio (cliente, aspirante a empleo, etc). En otras palabras, las empresas tomarán decisiones basadas en "dime con quién andas y te diré quién eres".

3.3. Teoría de grafos

En matemáticas y en ciencias de la computación, la teoría de grafos (también llamada teoría de las gráficas) estudia las propiedades de los grafos (también llamadas gráficas). Un grafo es un conjunto, no vacío, de objetos llamados vértices (o nodos) y una selección de pares de vértices, llamados aristas (edges en inglés) que pueden ser orientados o no. Típicamente, un grafo se representa mediante una serie de puntos (los vértices) conectados por líneas (las aristas).[45]

Figura 5: Diagrama de un grafo con 6 vértices y 7 aristas



3.3.1. Composición de un grafo

- **Aristas**: Son las líneas con las que se unen los vértices de un grafo.
- **Aristas adyacentes**: 2 aristas son adyacentes si convergen en el mismo vértice.
- **Aristas paralelas**: Son dos aristas conjuntas si el vértice inicial y final son el mismo.
- **Arista cíclicas**: Es la arista que parte de un vértice para entrar en sí mismo.

- **Cruce**: Son 2 aristas que cruzan en un mismo punto.
- **Vértices o nodos**: Los vértices son los elementos que forman un grafo. Cada uno lleva asociada una valencia característica según la situación, que se corresponde con la cantidad de aristas que confluyen en dicho vértice.
- **Camino**: Se denomina camino de un grafo a un conjunto de vértices interconectados por aristas. Dos vértices están conectados si hay un camino entre ellos.

3.3.2. Tipos de grafos

- **Grafo simple o simplemente grafo:** es aquel que acepta una sola arista uniendo dos vértices cualesquiera. Esto es equivalente a decir que una arista cualquiera es la única que une dos vértices específicos. Es la definición estándar de un grafo.
- **Multigrafo**: Es el que acepta más de una arista entre dos vértices. Estas aristas se llaman múltiples o lazos (*loops*). Los grafos simples son una subclase de esta categoría de grafos. También se le llama grafo general.
- **Pseudografo**: Si incluye algún lazo.
- **Grafo orientado:** grafo dirigido o digrafo, son grafos en los cuales se ha añadido una orientación a las aristas, representada gráficamente por una flecha.
- **Grafo etiquetado**: Grafos en los cuales se ha añadido un peso a las aristas (número entero generalmente) o un etiquetado a los vértices.
- **Grafo aleatorio:** Grafo cuyas aristas están asociadas a una probabilidad.
- **Hipergrafo:** Grafos en los cuales las aristas tienen más de dos extremos, es decir, las aristas son incidentes a 3 o más vértices.
- **Grafo infinito:** Grafos con conjunto de vértices y aristas de cardinal infinito.
- **Grafo plano:** Los grafos planos son aquellos cuyos vértices y aristas pueden ser representados sin ninguna intersección entre ellos.
- **Grafo regular:** Un grafo es regular cuando todos sus vértices tienen el mismo grado de valencia.

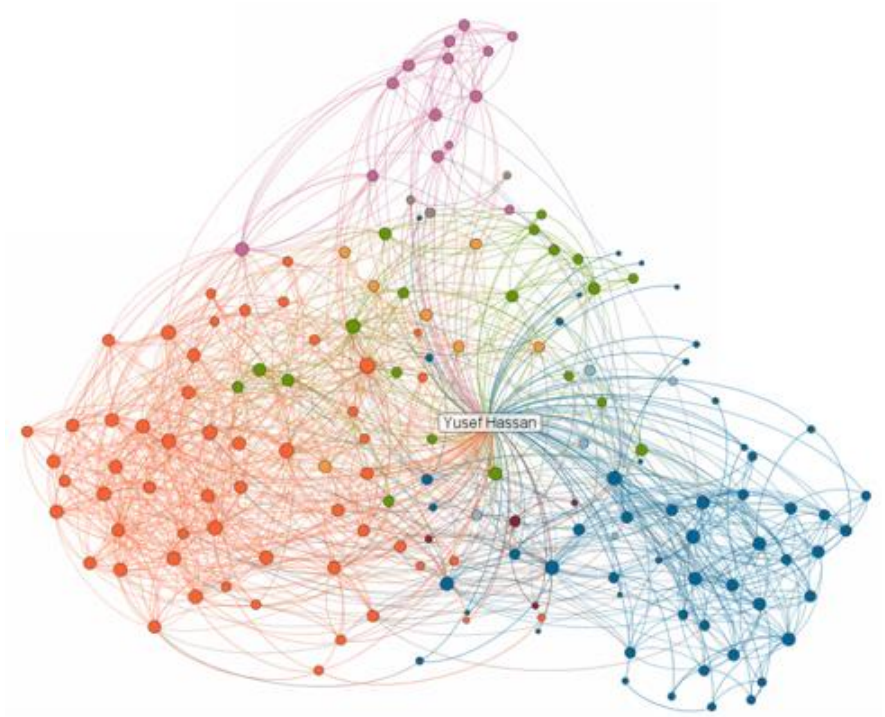
3.3.3. Grafo social

El grafo social es un concepto abstracto que describe las relaciones entre las personas en línea, en contraposición a la idea de una red social, que describe las relaciones en el mundo real. Los dos conceptos son muy similares, pero existen

algunas diferencias de menor importancia. Por ejemplo, el gráfico social es digital y, más importante aún, se define explícitamente por todas las conexiones en cuestión.

Grafo social es un término acuñado por Mark Zuckerberg, fundador de Facebook, que originalmente se refería a la red social de relaciones entre los usuarios del servicio de redes sociales proporcionados por Facebook. Este término se ha descrito como «el mapa global de todo el mundo y cómo están relacionados». [48]

Figura 6: Ejemplo de un grafo social [49]



3.3.3.1. Interés para analizar un grafo social

El interés por estudiar los patrones y estructura que esconden esta representación de nodos y aristas ha crecido en los últimos años a medida que ha aumentado la relación entre agentes. Es decir, a medida que han crecido las redes sociales, ha crecido la influencia de una persona en otra para comprar (ya que una persona confían más en la reputación de sus amigos que en la publicidad de las marcas), las redes de proveedores y clientes han aumentado sustancialmente (por la globalización

de la economía y la interconexión internacional), etc., crece el interés por estudiar qué patrones pueden descubrirse para incrementar la inteligencia del negocio.

¿Y por qué esto de interés ahora? En la medida en que un problema dado pueda ser modelado mediante un grafo y resuelto mediante algoritmos específicos de la teoría de grafos, la información que podemos obtener es muy relevante. Esto es algo que los topógrafos (cómo enlazar las estaciones del metro de Nueva York de la manera más eficiente para todas las variables a optimizar -distancia, coste, satisfacción usuario, etc.-) o los antropólogos (cómo se han relacionado las especies y los efectos producidos unos en otros) llevan muchos años ya explotando. Ahora, da el salto al mundo del consumo, la sanidad, la educación, etc.

¿Qué nos puede aportar un grafo, una red social, y su análisis a nuestros intereses? Las redes sociales se han definido anteriormente en el apartado 3.2.3 como un conjunto bien delimitado de actores que pueden ser individuos, grupos, organizaciones, comunidades, sociedades globales, entre otros. Están vinculados unos a otros a través de una relación o un conjunto de relaciones sociales. El análisis de estos vínculos puede ser empleado para interpretar comportamientos sociales de los implicados. Esto es lo que ha venido a denominarse el Análisis de Redes Sociales o ARS (*Social Network Analysis*, o SNA).

3.3.3.2. Sociometría:

Dentro del Análisis de Redes Sociales (ARS), uno de los conceptos clave es la Sociometría. Su fundador, Jacob Levy Moreno, la describió como:

"La sociometría tiene por objeto el estudio matemático de las propiedades psicológicas de las poblaciones; con este fin utiliza una técnica experimental fundada sobre los métodos cuantitativos y expone los resultados obtenidos por la aplicación de estos métodos. Persigue así una encuesta metódica sobre la evolución y la organización de los grupos y sobre la posición de los individuos en los grupos".

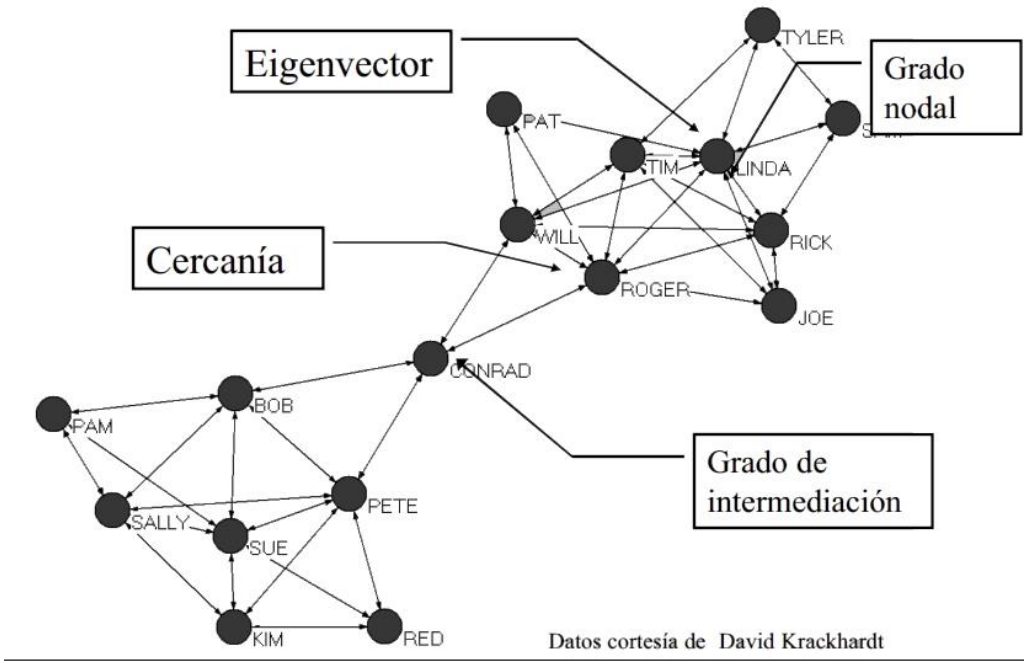
Haciendo uso de herramientas tecnológicas interactivas se puede hacer sociometría, (como por ejemplo la herramienta *Gephi)*, con lo cual se puede visualizar, explorar y analizar toda clase de redes y sistemas complejos, grafos jerárquicos y dinámicos. Una herramienta de este tipo nos permitirá obtener diferentes métricas, que podemos clasificar en tres niveles:

- **Nivel global de un grafo social**
    - **Coeficiente de agrupamiento**: nivel de agrupamiento de los nodos, para saber cómo de cohesionados o integrados están los agentes/actores.
    - **Camino característico**: mide el grado de separación de los nodos, para determinar lo contrario al punto anterior: cómo de separados o alejados están, y poder buscar así medidas para juntar más la relación entre agentes/actores.
    - **Densidad**: un grafo puede ser denso (cuando tiene muchas aristas) o disperso (muy pocas aristas). En este sentido, se puede interpretar como que hay mucha o poca conexión.
    - **Diámetro**: es el máximo de las distancias entre cualquier par de nodos. De esta manera, sabemos cómo de "alejados" o "próximos" están en agregado a la hora de comparar varios grafos.
    - **Grado medio**: número de vecinos (conexiones a otros nodos) medio que tiene un grado. Indicará cuál es la media de conexiones que tiene un nodo, de manera que se puede saber su popularidad..
    - **Centralidad**: permite realizar un análisis para indicar aquellos nodos que poseen una mayor cantidad de relaciones y por ende, los influyentes dentro del grupo. De esta manera, sabemos su "popularidad", lo que nos puede dar mucha información para saber la importancia de un nodo dentro del total.
- **Nivel comunidad** (grupos de nodos dentro de un grafo)
    - **Comunidades**: instrumento para conocerse a sí mismo, para conocer a los otros, al grupo concreto que vive su momento, y en general a los grupos que viven procesos similares. De esta manera, podemos agrupar a los nodos por patrones de similtud.
    - **Puentes entre comunidades**: ¿cómo se conectan estas comunidades? ¿cómo de comunicables son esas comunidades? Para trazar planes de actuación o de marketing.
    - **Centros locales vs. periferia**: para saber, dentro de las comunidades, los nodos que son más centrales o críticos, frente a los que no lo son.
- **Nivel nodo** (propiedades de un influenciador dado)
    - **Centralidad**: es una métrica de poder. El valor 0.522 para la centralidad de un nodo indica que si para cada par de influenciadores buscamos el camino

más corto en el grafo, el 52.2% de estos caminos pasa por ese influenciador. Mide su popularidad, y el algoritmo de Google, por ejemplo, funcionó durante mucho tiempo así, siendo cada nodo, una página web o recurso en Internet.

- **Modularidad**: la modularidad es una medida de la estructura de las redes o grafos. Fue diseñado para medir la fuerza de la división de una red en módulos (también llamados grupos, agrupamientos o comunidades). Las redes con alta modularidad tienen conexiones sólidas entre los nodos dentro de los módulos, pero escasas conexiones entre nodos en diferentes módulos.
- **Intermediación**: se puede enfocar como la capacidad que inviste el nodo en ocupar una posición intermediaria en las comunicaciones entre el resto de los influenciadores. Aquellos, con mayor intermediación tienen un gran liderazgo, debido a que controlan los flujos de comunicación. Y esto, de nuevo, da mucha inteligencia a un negocio.
- **Pagerank**: algoritmo que permite dar un valor numérico (ranking) a cada nodo de un grafo que mide de alguna forma su conectividad. Es el famoso pagerank que utilizó Google (de hecho, el algoritmo fue diseñado por los creadores de Google, que es de donde viene su pasado matemático).
- **Closeness**: cuán fácil es llegar a los otros vértices. Indicará, por lo tanto, cómo de cerca queda ese influenciador para llegar a contactar con otros. Esto, permite saber cuán importante es ese nodo dentro de la red de influencia para eventuales comunicaciones o relaciones con otros nodos.

Figura 7: Métricas de un nodo en una red [50]



Todo esto ya se está empleando en campos tan diversos como:

- El marketing digital para la detección de influenciadores entre los seguidores, de especial importancia en la economía colaborativa, donde la reputación online es clave de consumo.
- El fútbol: para la detección que Xavi y Xabi Alonso fueron la clave en el campeonato de Sudáfrica para que España ganase, en lugar de Iniesta, autor del gol),
- Optimización de rutas de distribución: por ejemplo, la de un taxi, en tiempo real, para evitar congestiones en ciudades como Pekín o New York)
- conocer los tipos de conversaciones: que se mantienen con tu audiencia/comunidad (muy usado en televisión y Twitter)
- Saber cuáles son los controladores de compra que más afectan a las decisiones de consumo de tus clientes, etc.

Entonces, las matemáticas, además de la sociología, a disposición de la inteligencia de un negocio. Eso es el análisis de redes sociales. [51]

3.4. API's

Conjunto de subrutinas, funciones y procedimientos (o métodos, en la programación orientada a objetos) que ofrece cierta biblioteca para ser utilizado por otro *software* como una capa de abstracción. Una *API* representa la capacidad de comunicación entre componentes de *software*. Se trata del conjunto de llamadas a ciertas bibliotecas que ofrecen acceso a ciertos servicios desde los procesos y representa un método para conseguir abstracción, generalmente (aunque no necesariamente) entre los niveles o capas inferiores y los superiores del *software*. [6]

3.4.1. Twitter

La Plataforma de Twitter permite conectar un sitio WEB o aplicaciones con el mundo de las conversaciones en Twitter.

¿Qué ofrecen los* API's*?

Twitter ofrece tres API's: Streaming API, REST *API* y Search API aplicables a necesidades diferentes.

- **Streaming API:** proporciona un subconjunto (*subset)* de *tweets* en casi tiempo real. Se establece una conexión permanente por usuario con los *servidores* de Twitter y mediante una petición *HTTP* se recibe un flujo continuo de *tweets* en formato *JSON*. Se puede obtener una muestra aleatoria (*statuses/sample*), un filtrado (*statuses/filter*) por palabras claves o por usuarios. Sin embargo, los métodos más interesantes cómo obtener todo el caudal de *tweets* (*statuses/firehose*) o sólo los *tweets* que tienen enlaces (*statuses/links*) o los *tweets* con re*tweets* (*statuses/retweet*) generalmente no se encuentran disponibles.
- **Search API:** suministra los *tweets* con una *profundidad* en el tiempo de 7 días que se ajustan a la consulta (*query)* solicitada. Es posible filtrar por, cliente utilizado, lenguaje y localización. No requiere autenticación y los *tweets* se obtienen en formato *JSON* o *ATOM*.
- **El REST API:** ofrece a los desarrolladores el acceso al *core* de los datos de Twitter. Todas las operaciones que se pueden hacer vía web son posibles realizarlas desde el API. Dependiendo si la operación requiere o no autenticación, con el mismo criterio que en el acceso web. Soporta los formatos: *XML*, *JSON*, *RSS*, *ATOM*. [7]

Podemos destacar que el *Search* API ofrece una información más limitada del *tweet*, en concreto, sobre los datos del autor solo indica identificador (ID*)*, el nombre de usuario (*screen_name)* y la URL de su avatar. Los otros dos API's si ofrecen el perfil completo del autor en el momento de la escritura del *tweet*.

Limitaciones

En el *Streaming* API el flujo es continuo y la velocidad de recepción de *tweets* tendrá fluctuaciones que dependerán del ancho de banda de los dos extremos de la conexión y la sobrecarga de los *servidores* de Twitter. En el *Search* API y en el *REST* API existe una limitación de 150 peticiones a la hora por usuario o por IP si la llamada no está autenticada. Es importante saber cómo realizar la paginación de las peticiones de una manera óptima para sacarle el máximo partido.

API	Petición	Máx. Tamaño de la pagina	Max. Total
SEARCH	*search*	200 *tweets*	1500 *tweets*
REST	*statuses*	200 *tweets*	3.200 *tweets*
REST	*friends/*id	5.000 id user	Los que haya
REST	*followers/*id	5.000 id user	Los que haya

Tabla 1: Limitaciones de las peticiones según el tipo de *API*

¿Qué persistencia tienen los tweets?

Aunque todos los *tweets* residan en las BBDD de Twitter hay una limitación temporal para obtenerlos.

API	Limitación temporal	Limitación de tamaño
Streaming	Solo tiempo real	-
Search	7 días	1.500 últimos *tweets*
REST	NO	3.200 últimos *tweets*

Tabla 2: Persistencia de los datos según cada una de las *API*'s

3.4.2. Facebook

Es un conjunto de servicios, herramientas y productos que son utilizados por los desarrolladores para obtener datos de la red social Facebook, como información de perfiles y conexiones de los usuarios. El propósito de dicha plataforma es crear una interfaz entre una aplicación y los servicios que Facebook provee. [9]

¿Qué ofrecen los API?

El *Graph* API, el núcleo de la plataforma, permite a los desarrolladores leer y escribir datos en Facebook.

La información del *Graph* API está compuesta por:

- **Nodos**: Son los objetos básicos, como un usuario, una foto, una página, un comentario.
- **Aristas:** son las conexiones entre estos objetos, por ejemplo los "me gusta" que no son representados por un nodo si no por la relación entre el usuario y el objeto del que el gusto.
- **Campos:** Es la información sobre los objetos, como nombre de un usuario o página, fecha de creación de una foto, entre otros.

La *Graph* API está basado en *HTTP*, por lo que funciona con cualquier lenguaje que contenga una librería *HTTP* como *curl*, *urllib.* La mayoría de las solicitudes al API necesitan un *access tokens* que se genera implementado el Facebook *Login*.

La información se lee a través de solicitudes GET a los nodos o aristas, la mayoría de las peticiones se realizan a través de *Graph.Facebook.com* a excepción de los *videos* que se suben a través de Graph-*video*.Facebook.com.

Versiones disponibles del *API*: 2.5, 2.4, 2.3, 2.2, 2.1, 2.0.

3.4.3. *Instagram*

La Plataforma de *Instagram* posee un conjunto de herramientas, que se pueden utilizar para construir aplicaciones y servicio no automatizado, autentico y de alta calidad .

¿Qué ofrece el API?

- **Puntos de acceso (*Endpoints*):** Los puntos de acceso representan un conjunto de servicio web REST que permiten acceder a todas las funcionalidades que *Instagram* brinda. Los puntos de acceso brindados por *Instagram* son los siguientes:
    - **Usuarios (*Users*):** Los puntos de acceso de los usuarios permite buscar usuarios por nombre, buscar información básicas de ellos y ver las personas que siguen (*newsfeed*) al igual que los *post* publicados y *likes*. Algunas de estas características para ser accedidas deben tener

una autenticación específica de usuario y otras pueden ser usadas por cualquier desarrollador.

- **Relaciones (*Relations*):** Los puntos de acceso de las relaciones permite traer una lista de todos los usuarios una persona sigue y quienes lo siguen a él, así como responder a las peticiones de relación y realizar cambios en las relaciones.

- **Media**: Permite obtener información de las fotos o *video*s de Instagram. También proporciona capacidad es de *geosearch* para buscar contenido de un tiempo y lugar en específico, de igual forma permite extraer los mensajes más populares.
- **Comentarios, me gusta y etiquetas** (***Comments, Likes y Tags)*:** Como su nombre lo indica, permite enumerar los comentarios, me gusta (*like*s), y etiquetas (*tags*) para cada video o foto publicada, y si es un *video* la cantidad de reproducciones que este tuvo.
- **Autenticación (*Autentication*):** Una vez registrado como desarrollador en *Instagram* se pueden realizar una variedad de solicitudes al *API* utilizando solo una clave de acceso (*Access key)*. Sin embargo por motivos de seguridad y obtener información privada del usuario *Instagram* utiliza AUTH 2.0

  La Autenticación por defecto permite realizar operaciones de lectura, pero *Instagram* permite al usuario agregar otras permisologías como:

  - **Comentarios**: para crear o eliminar comentarios en nombre de un usuario.
  - **Relaciones**: para seguir y dejar de seguir en nombre de un usuario.
  - ***Me gusta (Like*s)**: para me gusta (*like)* o no me gusta (un*like)* a una publicación en nombre de un usuario.

- **Servicios en tiempo real:** Estos servicios permiten monitorear las *actividad*es en vivo de los usuarios, etiquetas y ubicaciones y GPS, los servicios en tiempo real son:

  - **Usuarios *(Users)***: recibe notificaciones cuando los usuarios registrados con la aplicación publiquen fotos, esto requiere autenticación tu aplicación.
  - **Etiquetas (*Tags)***: recibe notificaciones cuando una foto es publicada con etiquetas que el usuario haya elegido.

- **Localización (*Locations)***: recibe notificaciones cuando una foto es publicada y etiquetada con una ubicación específica.

Plataformas que lo soportan:

Al *API* se puede acceder cualquier plataforma que use servicios REST, *Instagram* provee bibliotecas para phyton y php.

Limitaciones

Como cualquier *API*, este posee limitaciones de frecuencia, a la que el usuario debe adherirse, por ejemplo solo se pueden realizar 5000 solicitudes por hora. [10]

3.5. Base de datos

Una base de datos es un "almacén" que nos permite guardar grandes cantidades de información de forma organizada para que luego podamos encontrar y utilizar fácilmente. Estos datos son recolectados y explotados por los sistemas de información de una empresa o negocio en particular. [11]

3.5.1. Base de datos no relacional

Se puede decir que la aparición del término NoSQL aparece con la llegada de la WEB 2.0 ya que hasta ese momento sólo subían contenido a la red aquellas empresas que tenían un portal, pero con la llegada de aplicaciones como Facebook, Twitter o Youtube, cualquier usuario podía subir contenido, provocando así un crecimiento exponencial de los datos. Es en este momento cuando empiezan a aparecer los primeros problemas de la gestión de toda esa información almacenada en bases de datos relacionales. En un principio, para solucionar estos problemas de accesibilidad, las empresas optaron por utilizar un mayor número de máquinas pero pronto se dieron cuenta de que esto no solucionaba el problema, además de ser una solución muy cara. La otra solución era la creación de sistemas pensados para un uso específico que con el paso del tiempo han dado lugar a soluciones robustas, apareciendo así el movimiento NoSQL. Por lo tanto hablar de bases de datos NoSQL es hablar de estructuras que nos permiten almacenar información en aquellas situaciones en las que las bases de datos relacionales generan ciertos problemas debido principalmente a problemas de escalabilidad y rendimiento de las bases de datos relacionales donde se dan cita miles de usuarios concurrentes y con millones de consultas diarias. Además de lo comentado anteriormente, las bases de datos NoSQL son sistemas de

almacenamiento de información que no cumplen con el esquema entidad –relación. Tampoco utilizan una estructura de datos en forma de tabla donde se van almacenando los datos sino que para el almacenamiento hacen uso de otros formatos como clave–valor, mapeo de columnas o grafos (ver epígrafe 'Tipos de bases de datos NoSQL'). [12]

3.5.2. Ventajas de los sistemas No SQL

Esta forma de almacenar la información ofrece ciertas ventajas sobre los modelos relacionales. Entre las ventajas más significativas podemos destacar:

- **Se ejecutan en máquinas con pocos recursos**: Estos sistemas, a diferencia de los sistemas basados en SQL, no requieren de apenas computación, por lo que se pueden montar en máquinas de un coste más reducido.
- **Escalabilidad horizontal**: Para mejorar el rendimiento de estos sistemas simplemente se consigue añadiendo más nodos, con la única operación de indicar al sistema cuáles son los nodos que están disponibles.
- **Pueden manejar gran** cantidad **de datos:** Esto es debido a que utiliza una estructura distribuida, en muchos casos mediante tablas Hash.
- **No genera cuellos de botella**: El principal problema de los sistemas SQL es que necesitan transcribir cada sentencia para poder ser ejecutada, y cada sentencia compleja requiere además de un nivel de ejecución aún más complejo, lo que constituye un punto de entrada en común, que ante muchas peticiones puede ralentizar el sistema. [12]

3.5.3. Principales diferencias con bases de datos SQL

Algunas de las diferencias más destacables que nos podemos encontrar entre los sistemas NoSQL y los sistemas SQL están:

- **No utilizan SQL como lenguaje de consultas**: La mayoría de las bases de datos NoSQL evitan utilizar este tipo de lenguaje o lo utilizan como un lenguaje de apoyo. Por poner algunos ejemplos, Cassandra utiliza el lenguaje CQL, MongoDB utiliza *JSON* o BigTable hace uso de GQL.

- **No utilizan estructuras fijas como tablas para el almacenamiento de los datos:** Permiten hacer uso de otros tipos de modelos de almacenamiento de información como sistemas de clave–valor, objetos o grafos.
- **No suelen permitir operaciones JOIN:** Al disponer de un volumen de datos tan extremadamente grande suele resultar deseable evitar los JOIN. Esto se debe a que, cuando la operación no es la búsqueda de una clave, la sobrecarga puede llegar a ser muy costosa. Las soluciones más directas consisten en desnormalizar los datos, o bien realizar el JOIN mediante *software*, en la capa de aplicación.
- **Arquitectura distribuida**: Las bases de datos relacionales suelen estar centralizadas en una única máquina o bien en una estructura máster–esclavo, sin embargo en los casos NoSQL la información puede estar compartida en varias máquinas mediante mecanismos de tablas Hash distribuidas. [12]

3.5.4. Tipos de base de datos NoSQL

Dependiendo de la forma en la que almacenen la información, nos podemos encontrar varios tipos distintos de bases de datos NoSQL. Veamos los tipos más utilizados.

3.5.4.1. Bases de datos clave – valor:

Son el modelo de base de datos NoSQL más popular, además de ser la más sencilla en cuanto a funcionalidad. En este tipo de sistema, cada elemento está Identificado por una llave única, lo que permite la recuperación de la información de forma muy rápida, información que habitualmente está almacenada como un objeto binario (BLOB). Se caracterizan por ser muy eficientes tanto para las lecturas como para las escrituras. Algunos ejemplos de este tipo son Cassandra, BigTable o HBase.

3.5.4.2. Bases de datos documentales:

Este tipo almacena la información como un documento, generalmente utilizando para ello una estructura simple como *JSON* o *XML* y donde se utiliza una clave única para cada registro. Este tipo de implementación permite, además de realizar búsquedas por clave–valor, realizar consultas más avanzadas sobre el contenido del documento. Son las bases de datos NoSQL más versátiles. Se pueden utilizar en gran cantidad de proyectos, incluyendo muchos que tradicionalmente funcionarían sobre bases de datos relacionales. Algunos ejemplos de este tipo son MongoDB o CouchDB.

3.5.4.3. Bases de datos en grafos:

En este tipo de bases de datos, la información se representa como nodos de un grafo y sus relaciones con las aristas del mismo, de manera que se puede hacer uso de la teoría de grafos para recorrerla. Para sacar el máximo rendimiento a este tipo de bases de datos, su estructura debe estar totalmente normalizada, de forma que cada tabla tenga una sola columna y cada relación dos. Este tipo de bases de datos ofrece una navegación más eficiente entre relaciones que en un modelo relacional. Algunos ejemplos de este tipo son Neo4j, InfoGrid, OrientDB o Virtuoso.

3.5.4.4. Bases de datos columnares:

Están organizados de columna por columna en lugar de la fila: es decir, todos los casos de un solo elemento de datos (por ejemplo, Nombre de cliente) se almacenan de modo que se puede acceder como una unidad. Esto los hace especialmente eficaz en las consultas analíticas, como la lista de selecciones, que a menudo lee unos pocos elementos de datos, pero necesitamos ver todas las instancias de estos elementos. En contraste, una convencional base de datos relacional almacena los datos por filas, por lo que toda la información de un registro (fila) es inmediatamente accesible. Esto tiene sentido para las consultas transaccionales, que suelen referirse a un registro a la vez.

Hoy los sistemas columnares combinan su estructura columnar con técnicas que incluyen la indexación, compresión y paralelización.

3.5.5. Ejemplos de base de datos NoSQL

- ***Cassandra:***

Se trata de una base de datos creada por Apache del tipo clave–valor. Dispone de un lenguaje propio para realizar consultas CQL (Cassandra Query Language). Cassandra es una aplicación Java por lo que puede correr en cualquier plataforma que cuente con la JVM.

- ***Redis:***

Se trata de una base de datos creada por Salvatore Sanfilippo y Pieter Noordhuis y está apoyado por VMWare. Se trata de una base de datos del tipo clave–valor. Se puede imaginar como un arreglo gigante en memoria para almacenar datos, datos que pueden ser cadenas, hashes, conjuntos de datos o listas. Tiene la ventaja de que sus operaciones son atómicas y persistentes. Por ponerle una limitante, Redis no

permite realizar consultas, sólo se puede insertar y obtener datos, además de las operaciones comunes sobre conjuntos (diferencia, unión e inserción). Creado en ANSI C, por lo tanto es compatible y funciona sin problemas en sistemas Unix, Linux y sus derivados, Solaris, OS/X sin embargo no existe soporte oficial para plataformas Windows.

- **MongoDB**:

Se trata de una base de datos creada por 10gen del tipo orientada a documentos, de esquema libre, es decir, que cada entrada puede tener un esquema de datos diferente que nada tenga que ver con el resto de registros almacenados. Es bastante rápido a la hora de ejecutar sus operaciones ya que está escrito en lenguaje C++. Para el almacenamiento de la información, utiliza un sistema propio de documento conocido con el nombre BSON, que es una evolución del conocido *JSON* pero con la peculiaridad de que puede almacenar datos binarios. En poco tiempo, MongoDB se ha convertido en una de las bases de datos NoSQL favoritas por los desarrolladores.

- ***CouchDB***:

Se trata de un sistema creado por Apache y escrito en lenguaje *Erlang* que funciona en la mayoría de sistemas POSIX, incluyendo GNU/LINUX y OSX, pero no así en sistemas Windows. Como características más importantes cabe destacar el uso de Restfull *HTTP API* como interfaz y JavaScript como principal lenguaje de interacción. Para el almacenamiento de los datos se utiliza archivos *JSON*. Permite la creación de vistas, que son el mecanismo que permite la combinación de documentos para retornar valores de varios documentos, es decir, CouchDB permite la realización de las operaciones JOIN típicas de SQL.

- ***OrientBD:***

Es una base de datos multi-modelo, escrita en Java, de apoyo gráfico , documento , clave/valor, y el objeto modelos, [1] pero las relaciones se gestionan como en bases de datos de gráficos con conexiones directas entre los registros. Es compatible con los modos de esquema mezclado sin esquema, el esquema completo y. Cuenta con un sistema de perfiles de seguridad fuerte basada en usuarios y roles y apoyos consultar con Gremlin junto con SQL extendido por recorrido del grafo. OrientDB utiliza un nuevo algoritmo de indexación llamada MVRB-Árbol, derivado del árbol rojo-negro y desde el árbol B + . [12]

- ***Neo4J:***

Neo4j es un *software* libre de Base de datos orientada a grafos, implementado en Java.1 2 Sus desarrolladores la describen como un motor de persistencia embebido, basado en disco, implementado en Java, completamente transaccional, que almacena datos estructurados en grafos en lugar de en tablas. [41]

3.5.6. Entonces, ¿SQL o NoSQL?

Algunas de las razones que nos pueden llevar a decantarnos por el uso de las bases de datos NoSQL en lugar de las clásicas SQL son:

- Cuando el volumen de los datos crece muy rápida mente en momentos puntuales, pudiendo llegar a superar el Terabyte de información.
- Cuando la escalabilidad de la solución relacional no es viable tanto a nivel de costes como a nivel técnico.
- Cuando tenemos elevados picos de uso del sistema por parte de los usuarios en múltiples ocasiones.
- Cuando el esquema de la base de datos no es homogéneo, es decir, cuando en cada inserción de datos la información que se almacena puede tener campos distintos.

3.5.7. Grandes compañías que utilizan este tipo de base de datos:

Son muchas las grandes empresas que hacen uso de este tipo de bases de datos no relacionales, como:

- Cassandra: Facebook, Twitter…
- HBase: Yahoo, Adobe…
- Redis: Flickr, *Instagram*, Github…
- Neo4j: Infojobs…
- MongoDB: FourSquare, SourceForge, CERN… [12]

3.6. Tecnologías del lado del servidor

3.6.1. Java

Java es un lenguaje de programación de propósito general, concurrente, orientado a objetos que fue diseñado específicamente para tener tan pocas dependencias de implementación como fuera posible. Su intención es permitir que los desarrolladores de aplicaciones escriban el programa una vez y lo ejecuten en cualquier dispositivo (conocido en inglés como WORA, o "*write once, run anywhere*"), lo que quiere decir que el código que es ejecutado en una plataforma no tiene que ser recompilado para correr en otra. [13]

Características:

Es un lenguaje simple ya que viene de la misma estructura de de C y C++.

- **Orientado Objeto:** Java soporta las tres características propias del paradigma orientada a objetos, encapsulación herencia y polimorfismo.
- **Distribuido:** Permite, establecer y aceptar conexiones con los *servidores* o clientes remotos; proporciona una colección de clases para aplicaciones en red lo que facilita la creación de aplicaciones distribuida s.
- **Robusto**:
    - Realiza verificaciones en busca de problemas tanto en tiempo de compilación como en tiempo de ejecución.
    - La comprobación de tipos en Java ayuda a detectar errores, lo antes posible, en el ciclo de desarrollo.
    - Obliga a la declaración explícita de métodos, reduciendo así las posibilidades de error.
    - Maneja liberación y corrupción de memoria
    - Comprobación de puntero.
    - Comprobación de límites de arreglos
    - Excepciones
    - Verificación de *bytes-code*
- **Seguro:** Se implementaron barreras de seguridad en el lenguaje como la eliminación de punteros o casting implícitos, evitando así el acceso ilegal a la memoria y en el sistema de ejecución en tiempo real es decir ningún objeto se

crea y se almacena en memoria, sin que se validen los privilegios de acceso. [14]

- **Indiferente a la arquitectura:** Java está diseñado para soportar aplicaciones que serán ejecutadas en los más variados entornos de red,(Unix, Windows Nt, MAC entre otros), sobre arquitecturas distintas y con sistemas operativos diversos. Para cumplir con esto el compilador genera un formato intermedio indiferente a la arquitectura, diseñado para transportar el código de manera eficiente a múltiples plataformas de hardware y *software*. El resto de los problemas los soluciona el intérprete de Java. [15]
- **Portable:** Por ser indiferente a la arquitectura sobre la cual está trabajando, esto hace que su portabilidad sea muy eficiente, sus programas son iguales en cualquiera de las plataformas, ya que java especifica tamaños básicos, esto se conoce como la máquina virtual de java. [14]
- **Interpretado y compilado a la vez**: Java puede ser compilado e interpretado en tiempo real, ya que cuando se construye el código fuente este se transforma en una especie de código de máquina. [14]
- **Multihilos:** Permitir la ejecución simultánea de muchas funciones en tiempo real. [14]
- **Dinámico:** Java es muy dinámico en la fase de enlazado, aprovecha al máximo la tecnología orientada a objetos es decir enlaza los módulos a medida que sean requeridos o necesitados, lo que permite que los enlaces se puedan incluir desde fuentes muy variadas o desde la red. [14]

3.6.2. C

C es un lenguaje de programación de propósito general que ofrece economía sintáctica, control de flujo y estructuras sencillas, es sencillo de medio nivel pero con muchas características de bajo nivel y no está especializado en ningún tipo de aplicación lo que lo hace un lenguaje potente con un campo de aplicación ilimitado, es muy útil para escribir compiladores y sistemas operativos, aunque de igual forma se puede desarrollar cualquier tipo de aplicación. Es un lenguaje de tipos de datos estáticos, débilmente tipificado.

Los compiladores suelen ofrecer extensiones al lenguaje que posibilitan mezclar código en ensamblador con código C o acceder directamente a memoria o dispositivos periféricos. [16]

Características

- C es un lenguaje de nivel medio ya que combina ya que combina los elementos de alto nivel con la funcionalidad del ensamblador.
- Es portable, es decir, es independiente del hardware. Los programas escritos en C son fácilmente transportables a otros sistemas.
- Es estructurado, es decir, el programa se divide en módulos (funciones) independientes entre sí.
- Inicialmente fue creado para la programación de Sistemas operativos, Intérpretes, Editores, Ensambladores, Compiladores. [17]

3.6.3. PHP

PHP es un lenguaje de programación de uso general de código del lado del servidor originalmente diseñado para el desarrollo WEB de contenido dinámico. Fue uno de los primeros lenguajes de programación del lado del servidor que se podían incorporar directamente en el documento HTML en lugar de llamar a un archivo externo que procese los datos. El código es interpretado por un servidor WEB con un módulo de procesador de PHP que genera la página WEB resultante. PHP actualmente incluye una interfaz de línea de comandos que puede ser usada en aplicaciones gráficas independientes. Puede ser usado en la mayoría de los *servidores* WEB al igual que en casi todos los sistemas operativos y plataformas sin ningún costo.

PHP es considerado uno de los lenguajes más flexibles, potentes y de alto rendimiento conocido s hasta el día de hoy, lo que ha atraído el interés de múltiples sitios con gran demanda de tráfico, como Facebook, para optar por el mismo como tecnología de servidor.

Características:

- Orientado al desarrollo de aplicaciones WEB dinámicas con acceso a información almacenada en una base de datos.
- Es considerado un lenguaje fácil de aprender, ya que en su desarrollo se simplificaron distintas especificaciones, como es el caso de la definición de las variables primitivas.

- El código fuente escrito en PHP es invisible al navegador WEB y al cliente, ya que es el servidor el que se encarga de ejecutar el código y enviar su resultado HTML al navegador.
- Capacidad de conexión con la mayoría de los motores de base de datos que se utilizan en la actualidad, destaca su conectividad con MySQL y *Post*greSQL.
- Capacidad de expandir su potencial utilizando módulos (llamados ext's o extensiones).
- Posee una amplia documentación en su sitio WEB oficial, entre la cual se destaca que todas las funciones del sistema están explicadas y ejemplificadas en un único archivo de ayuda.
- Es libre, por lo que se presenta como una alternativa de fácil acceso para todos.
- Permite aplicar técnicas de programación orientada a objetos.
- No requiere definición de tipos de variables aunque sus variables se pueden evaluar también por el tipo que estén manejando en tiempo de ejecución.
- Tiene manejo de excepciones (desde PHP5).
- Si bien PHP no obliga a quien lo usa a seguir una determinada metodología a la hora de programar, aun haciéndolo, el programador puede aplicar en su trabajo cualquier técnica de programación o de desarrollo que le permita escribir código ordenado, estructurado y manejable. Un ejemplo de esto son los desarrollos que en PHP se han hecho del patrón de diseño Modelo Vista Controlador (MVC), que permiten separar el tratamiento y acceso a los datos, la lógica de control y la interfaz de usuario en tres componentes independientes.
- Debido a su *flexibilidad* ha tenido una gran acogida como lenguaje base para las aplicaciones WEB de manejo de contenido, y es su uso principal.

Inconvenientes*:*

- Como es un lenguaje que se interpreta en ejecución, para ciertos usos puede resultar un inconveniente que el código fuente no pueda ser ocultado. La ofuscación es una técnica que puede dificultar la lectura del código pero no necesariamente impide que el código sea examinado.
- Debido a que es un lenguaje interpretado, un script en PHP suele funcionar considerablemente más lento que su equivalente en un lenguaje de bajo nivel, sin embargo este inconveniente se puede minimizar con técnicas de caché tanto en archivos como en memoria.

- En las versiones previas a la 7, las variables no son tipificadas, lo cual dificulta a los diferentes IDEs ofrecer asistencias para el tipificado del código, aunque esto no es realmente un inconveniente del lenguaje en sí. Esto es solventado por algunos IDEs añadiendo un comentario con el tipo a la declaración de la variable.

3.6.4. Ruby

Es un lenguaje de programación dinámico y de código abierto, es interpretado, reflexivo y orientado a objetos que combina una sintaxis inspirada en Python y en Perl.

Características

- Lenguaje de guiones interpretado:
  - Posibilidad de realizar directamente llamadas al sistema operativo
  - Potentes operaciones sobre cadenas de caracteres y expresiones regulares
  - Retroalimentación inmediata durante el proceso de desarrollo
- Rápido y sencillo:
  - Son innecesarias las declaraciones de variables
  - Las variables no tienen tipo
  - La sintaxis es simple y consistente
  - La gestión de la memoria es automática
- Programación orientada a objetos:
  - Todo es un objeto
  - Clases, herencia, métodos.
  - Métodos singleton
  - Mixins por módulos
  - Iteradores y cierres
- También maneja:
  - Enteros de precisión múltiple
  - Modelo de procesamiento de excepciones
  - Carga dinámica
  - Hilos

3.6.5. Phyton

Es un lenguaje de script independiente de la plataforma y orientado objeto, con este lenguaje se pueden realizar cualquier tipo de programas, desde aplicaciones

Windows a *servidores* de red o incluso, páginas WEB. Es un lenguaje interpretado, lo que significa que no se necesita compilar el código fuente para poder ejecutarlo, lo que ofrece ventajas como la r*api*dez de desarrollo e inconvenientes como una menor velocidad. [18]

Características

- **Propósito General:** Como se mencionó anteriormente se pueden crear cualquier tipo de programas.
- **Multiplataforma:** Originalmente se desarrolló para Unix, aunque cualquier sistema es compatible con el lenguaje siempre y cuando exista un intérprete programado para el.
- **Interactivo:** Python dispone de un intérprete por línea de comandos en el que se pueden introducir sentencias. Cada sentencia se ejecuta y produce un resultado visible, que es útil probar los resultados de la ejecución de porciones de código rápida mente.
- Usa **tipiado** dinámico y conteo de referencias para la administración de memoria.
- **Incrustarle:** Se puede insertar lenguaje Python dentro un programa C/C++ y de esta manera ofrecer las *facilidad*es del scripting. [19]
- **Sintaxis Clara:** Por último, destacar que Python tiene una sintaxis muy visual, gracias a una notación identada (con márgenes) de obligado cumplimiento. En muchos lenguajes, para separar porciones de código, se utilizan elementos como las llaves o las palabras clave *begin* y *end*. Para separar las porciones de código en Python se debe tabular hacia dentro, colocando un margen al código que iría dentro de una función o un bucle. Esto ayuda a que todos los programadores adopten unas mismas notaciones y que los programas de cualquier persona tengan un aspecto muy similar. [20]

3.6.6. R

Es un lenguaje de programación orientado a objetos y entorno de programación para análisis estadísticos y gráficos. Es una implementación libre, independiente, código abierto (*open-source)* del lenguaje de programación S actualmente conocido como S-Plus ,S es un lenguaje utilizado para análisis estadísticos.

Características:

- Proporciona una amplia gama de herramientas para el análisis estadístico.
- Posee herramientas que permiten generar gráficos de alta calidad
- Permite que los usuarios lo extiendan definiendo sus propias funciones, aunque gran parte de las funciones de R están escrito en el mismo lenguaje, para algoritmos computacionales exigentes es posible desarrollar bibliotecas en C , C++ o Fortan que se cargan dinámicamente.
- Los usuarios con mayor experiencia pueden manipular los objetos de R directamente desarrollado en C.
- Capacidad de conectarse con diferentes bases datos y existen bibliotecas que facilitan su utilización desde lenguajes de programación como Perl y Python.
- Posee su propio formato para la documentación basado en LaTex
- R posee herramientas que trabajan de manera eficaz para el cálculo numérico lo que son útiles para la minería de datos.

3.7. Tecnologías lado del cliente

3.7.1. HTML

Es un lenguaje de marcas de hipertexto, que sirve de referencia para la elaboración de páginas WEB en sus diferente versiones, define una estructura básica y un código denominado código HTML, para la definición de contenido de una página WEB, como textos imágenes, *vid*e*o*s entre otros. Está a cargo de la W3C, organización dedicada a la estandarización de casi todas las tecnologías ligadas a la WEB.

Se considera el lenguaje WEB más importante siendo su invención crucial en la aparición, desarrollo y expansión de la Web. Es el estándar que se ha impuesto en la visualización de páginas WEB y es el que todos los navegadores actuales han adoptado.

Características

- Lenguaje de etiquetas
- Es distribuido: La información está repartida en páginas no muy grandes enlazadas entre sí.
- Es hipertexto y de fácil navegabilidad
- Compatible con todo tipo de ordenadores y con todo tipo de sistemas operativos

- Es dinámico: el proceso de cambiar y actualizar información es ágil y rápida [21

3.7.2. JavaScript

JavaScript es un lenguaje de programación interpretado, por lo que no es necesario compilar los programas para ejecutarlos. En otras palabras, los programas escritos con JavaScript se pueden probar directamente en cualquier navegador sin necesidad de procesos intermedios, los programas escritos en él se llaman script.

Se utiliza principalmente para crear páginas WEB dinámicas, es decir que incorpore efectos, animaciones entre otras cosas en las páginas WEB. Con JavaScript no pueden construirse programas independientes, sólo pueden escribirse scripts que funcionarán en el entorno de una página WEB

A pesar de su nombre, JavaScript no guarda ninguna relación directa con el lenguaje de programación Java

Características

- Java Script es un lenguaje de secuencias de comandos basado en objetos e interpretado. [22]
- Maneja objetos dentro de la WEB lo que facilita la programación de páginas interraíz s, sobre dichos objetos de pueden definir diferentes eventos.
- Es dinámico, es decir, responde a eventos en tiempo real, por ejemplo presionar un botón y que se genere una acción. [23]
- Es limitado, es decir no se pueden realizar aplicaciones independientes en JavaScript y la capacidad de lectura y escritura de archivos es mínima [22]
- Las secuencias de comandos de Java Script sólo pueden ejecutarse con un intérprete, que bien puede estar en un servidor WEB o en un explorador de WEB.
- No se declaran tipo de variables

3.7.3. Applet Java

Es una manera incluir código a ejecutar en los clientes que visualizan una página WEB. Se trata de pequeños programas hechos en Java, que se transfieren con las páginas WEB y que el navegador ejecuta en el espacio de la página.

Los *applets* de Java están programados en Java y precompilados, es por ello que la manera de trabajar de éstos varía un poco con respecto a los lenguajes de script como Javascript, estos son más difíciles de programar que los scripts y requieren conocimientos básicos o medios del lenguaje Java.

La principal ventaja de utilizar *applets* consiste en que son mucho menos dependientes del navegador que los scripts, incluso independientes del sistema operativo del ordenador donde se ejecutan. Además, Java es más potente que Javascript, por lo que el número de aplicaciones de los *applets* podrá ser mayor.

Como desventajas en relación con Javascript cabe señalar que los *applets* son más lentos de procesar y que tienen espacio muy delimitado en la página donde se ejecutan, es decir, no se mezclan con todos los componentes de la página ni tienen acceso a ellos. Es por ello que con los *applets* de Java no podremos hacer directamente cosas como abrir ventanas secundarias, controlar marcos (*Frames*), formularios, capas, etc. [24]

3.8. Tecnologías Push

La tecnología push es una forma de comunicación a través de internet en la que la petición de envío tiene origen en el servidor, por oposición a la tecnología pull, en la que la petición tiene origen en el cliente.

Los servicios push están basados, a menudo, en preferencias de información a medida. Es decir, un modelo publicador/suscriptor. Un cliente deberá suscribirse a varios canales de información. Cuando el nuevo contenido está disponible en uno de estos canales, el servidor deberá enviar la información al usuario.

3.8.1. *HTTP server push*

HTTP server push (conocido como HTTP streaming) es un mecanismo de envío de datos desde un servidor web hacia un navegador web HTTP server push puede ser realizado a través de muchos mecanismos.

Generalmente el servidor web no termina la respuesta después que los datos han sido enviados al cliente. El servidor web deja el canal abierto para que si un evento ocurre, éste pueda ser enviado inmediatamente a uno o a muchos clientes. De otra forma, los datos deberían ser puestos en una cola hasta que se reciba la próxima petición del cliente

3.8.2. *Java pushlet*

Un pushlet es una técnica desarrollada originalmente para aplicaciones web sobre Java aunque las mismas técnicas pueden ser empleadas, en otros frameworks web. En esta técnica, el servidor toma ventaja de las conexiones persistentes HTTP y deja la respuesta al cliente "abierta" (es decir, no la termina), dejando al browser en un continuo estado de "carga" en el momento que la página inicial debería haber sido completada. El servidor, luego, utiliza periódicamente rutinas de javascript para actualizar el contenido de la página, consiguiendo de esta manera, la capacidad de push. Usando esta técnica, el cliente no necesita applets Java u otros plug-ins para mantener una conexión abierta con el servidor. Los clientes pueden ser notificados automáticamente de nuevos eventos, puestos por el servidor. Una importante desventaja de este método, sin embargo, es la pérdida del control que tiene el servidor sobre los tiempos de expiración. Un refresco de la página siempre es necesario si la página expira o el browser es cerrado.

3.8.3. *Long polling*

Long polling es una técnica donde el servidor no tiene información disponible para el cliente de manera inmediata, por ende en vez de enviar una respuesta vacía, el servidor guarda la petición y espera a que alguna información esté disponible. Una vez la información está disponible (o después de un tiempo establecido), se envía una respuesta completa al cliente. Entonces el cliente normalmente realizará un re-pedido de información al servidor, para que éste siempre tenga un pedido en espera, que puede ser usado para responder a un evento [39]

3.9. *Framework*s y herramientas

3.9.1. Servidor

- ***NodeJs***

Node.js es un entorno en tiempo de ejecución multiplataforma, de código abierto, para la capa del servidor (pero no limitándose a ello) basado en el lenguaje de programación ECMAScript, asíncrono, con I/O de datos en una arquitectura orientada a

eventos y basado en el motor V8 de Google. Fue creado con el enfoque de ser útil en la creación de programas de red altamente escalables, como por ejemplo, *servidores*WEB.2 Al contrario que la mayoría del código JavaScript, no se ejecuta en un navegador, sino en el servidor. Nodjs incluye un entorno REPL para depuración interraíz.

Aspectos técnicos

- **Paralelismo:** Node.js funciona con un modelo de evaluación de un único hilo de ejecución, usando entradas y *salida*s asíncronas las cuales pueden ejecutarse concurrentemente en un número de hasta cientos de miles sin incurrir en costos asociados al cambio de contexto. Este diseño de compartir un único hilo de ejecución entre todas las solicitudes atiende a necesidades de aplicaciones altamente concurrentes, en el que toda operación que realice entradas y *salida*s debe tener una función callback. Un inconveniente de este enfoque de único hilo de ejecución es que Node.js requiere de módulos adicionales como cluster7 para escalar la aplicación con el número de núcleos de procesamiento de la máquina en la que se ejecuta.
- **V8**: es el ambiente de ejecución para JavaScript, creado por Google para Google Chrome, está escrito en C++ y compila el código fuente JavaScript en código de maquina en lugar de interpretado en tiempo real, el cuerpo de operaciones de base de Node.js está escrito en JavaScript con métodos de soporte escritos en C++.

  Node.js contiene libuv para manejar eventos asíncronos. Libuv es una capa de abstracción de funcionalidad es de redes y sistemas de archivo en sistemas Windows y sistemas basados en POSIX como Linux, Mac OS X y Unix.

  - **Módulos**: Node.js incorpora varios "módulos básicos" compilados en el propio binario, como por ejemplo el módulo de red, que proporciona una capa para programación de red asíncrona y otros módulos fundamentales, como por ejemplo *Path*, *FileSystem*, *Buffer*, *Timers* y el de propósito más general *Stream*. Es posible utilizar módulos desarrollados por terceros, ya sea como archivos ".node" precompilados, o como archivos en Javascript plano.

- **Desarrollo homogéneo entre cliente y servidor:** Node.js puede ser combinado con una base de datos documental (por ejemplo, MongoDB o CouchDB) y *JSON* lo que permite desarrollar en un ambiente de desarrollo JavaScript unificado. Con la adaptación de los patrones para desarrollo del lado del servidor tales como MVC y sus variantes MVP, MVVM, etc. Node.js facilita la reutilización de código del mismo modelo de interfaz entre el lado del cliente y el lado del servidor.
- **Lazo de eventos:** Node.js se registra con el sistema operativo y cada vez que un cliente establece una conexión se ejecuta un callback. Dentro del ambiente de ejecución de Node.js, cada conexión recibe una pequeña asignación de espacio de memoria dinámico, sin tener que generar un hilo de trabajo. A diferencia de otros *servidores* dirigidos por eventos, el lazo de manejo de eventos de Node.js no es llamado explícitamente sino que se raíz al final de cada ejecución de una función de *callback*. El lazo de manejo de eventos se termina cuando ya no quedan eventos por atender. [25]
- **Sprint:** Es un *framework* para el desarrollo de aplicaciones y contenedor de inversión de control, de código abierto para la plataforma Java. Si bien las características fundamentales de *Spring Framework* pueden ser usadas en cualquier aplicación desarrollada en Java, existen variadas extensiones para la construcción de aplicaciones WEB sobre la plataforma Java EE. A pesar de que no impone ningún modelo de programación en particular, este *framework* se ha vuelto popular en la comunidad al ser considerado una alternativa, sustituto, e incluso un complemento al modelo EJB (Enterprise JavaBean). [42]

3.9.2. Cliente

- ***AngularJS:***

Es un *framework* de JavaScript de código abierto, mantenido por Google, que se utiliza para crear y mantener aplicaciones WEB de una sola página. Su objetivo es aumentar las aplicaciones basadas en navegador con capacidad de Modelo Vista Controlador (MVC). La biblioteca lee el HTML que contiene atributos de las etiquetas personalizadas adicionales, entonces obedece a las

directivas de los atributos personalizados, y une las piezas de entrada o *salida* de la página a un modelo representado por las variables estándar de JavaScript. Los valores de las variables de JavaScript se pueden configurar manualmente, o recuperados de los recursos *JSON* estáticos o dinámicos. [26]

AngularJS se combina con el entorno en tiempo de ejecución Node.js, el *framework* para servidor Express.js y la base de datos MongoDB para formar el conjunto MEAN.

3.10. Arquitectura orientada a Servicios (SOA)

Es un paradigma de arquitectura para diseñar y desarrollar sistemas distribuidos. Las soluciones SOA han sido creadas para satisfacer los objetivos de negocio las cuales incluyen *facilidad* y *flexibilidad* de integración con sistemas legados, alineación directa a los procesos de negocio reduciendo costos de implementación, innovación de servicios a clientes y una adaptación ágil ante cambios incluyendo reacción temprana ante la competitividad.

Permite la creación de sistemas altamente escalables que reflejan el negocio de la organización, a su vez brinda una forma bien definida de exposición e invocación de servicios (comúnmente pero no exclusivamente servicios WEB), lo cual facilita la interacción entre diferentes sistemas propios o de terceros.

SOA proporciona una metodología y un marco de trabajo para documentar las capacidades de negocio y puede dar soporte a las *actividad*es de integración y consolidación. [30]

3.10.1. SOA desde el punto del negocio

La arquitectura SOA, desde el punto de vista del negocio, ayuda a resolver los siguientes requerimientos, largamente reclamados por el área de negocio:

- Mejorar la *flexibilidad* y *agilidad* de los sistemas.
- Proporcionar una visión integrada de los distintos "silos" de la organización.
- Mejorar la cobertura de las necesidades de negocio.
- Reducir el impacto de la evolución de la tecnología en las aplicaciones de negocio.

La arquitectura SOA permite a las organizaciones satisfacer las cambiantes necesidades de la empresa mediante la implantación de procesos de negocio que utilizan los servicios proporcionados por los sistemas actuales. La arquitectura garantiza la interoperabilidad de los sistemas a pesar de que, en gran parte, hayan sido construidos en distintos momentos, con diferentes intenciones, plataformas y niveles de servicio, y a pesar del hecho de que ahora se encuentren en distintos ciclos de mantenimiento, mejora y presupuesto. Anteriores estrategias de integración entraban en conflicto con estas realidades, pero ahora la arquitectura SOA ofrece un modo de enfrentarse mejor a ellas y de aumentar los niveles de *agilidad* y *flexibilidad*.

La arquitectura SOA posibilita la ruptura de los silos internos de una organización (a menudo causados por sistemas que pertenecen a departamentos individuales y que no pueden conectarse fácilmente), y prepara el terreno para dar soporte a procesos de negocio automatizado e integrado que eliminan dichos silos. Descubrir las capacidades de negocio en forma de servicios y después conectar y consumir esos servicios (por ejemplo, a través de aplicaciones compuestas, procesos automatizados o empresas virtuales) son factores que pueden simplificar enormemente los procesos de una organización, al facilitar la ubicación y reutilización de una misma funcionalidad en diferentes partes del negocio.

La sencillez interna proporciona a la organización la *agilidad* necesaria para crear nuevos productos y servicios de una forma más fácil y rápida, y le permite así diferenciarse en el mercado. La diferenciación competitiva resulta esencial para la mayoría de los sectores, y la arquitectura SOA proporciona los elementos necesarios para que las organizaciones alcancen con éxito el alto rendimiento.

3.10.2. SOA desde el punto de vista tecnológico

La arquitectura orientada a servicios, desde el punto de vista tecnológico, es el resultado de la constante evolución hacia un mayor desacoplamiento de las capas de una aplicación (presentación, orquestación de procesos y servicios de negocio) y a un mayor nivel de estandarización/ interoperabilidad de cada una de estas capas.

Con un total desacoplamiento de las capas de las aplicaciones y un muy alto nivel de interoperabilidad, los beneficios desde el punto de vista tecnológico son claros:

Favorece la reutilización y la reducción del "*time to market*":

- Aumenta el grado de reutilización al desacoplar las capas de una aplicación.

- Permite reutilizar las aplicaciones existentes mediante la encapsulación en servicios.
- Permite la utilización de servicios de terceros.
- Permite reaprovechar las plataformas existentes.

Aumenta la *flexibilidad*:

- Simplifica la adaptación de los sistemas existentes.
- Evita el desarrollo de interfaces punto a punto entre los sistemas.
- Aumenta la interoperabilidad entre sistemas, permitiendo tanto la externalización como la prestación de servicios.

Mejora la productividad de los procesos:

- Aumenta el nivel de automatización de los procesos, reduciendo el número de *actividad*es manuales.
- Permite monitorizar la *actividad* del negocio (cuadros de mando).
- Permite realizar un análisis estadístico de los flujos de negocio reales en base a indicadores clave de negocio, permitiendo la identificación de puntos de mejora a optimizar.
- Permite evaluar el impacto y beneficio de variantes en los procesos mediante simulación.

Mejora el proceso de construcción de *software*:

- Favorece la industrialización.
- Mejora la especificación de los requerimientos de negocio.
- Proporciona una filosofía de desarrollo común a todos los negocios y canales.
- Mejora la calidad.
- Desacopla el desarrollo de servicios y de procesos.
- Mejora el mantenimiento (procesos autodocumentados).

Mejora la usabilidad de las aplicaciones:

- Permite presentar al usuario la información dispersa en distintos sistemas y de forma integrada.
- Permite alcanzar un mayor nivel de automatismo en las aplicaciones en procesos complejos del flujo de trabajo (*workflow)*.
- Permite utilizar tecnologías de presentación avanzadas como WEB 2.0.

3.10.3. Beneficios de SOA

- La arquitectura SOA ayuda a mejorar la *agilidad* y *flexibilidad* de las organizaciones
- La arquitectura SOA permite una "personalización masiva" de las tecnologías de la información
- La arquitectura SOA permite la simplificación del desarrollo de soluciones mediante la utilización de estándares de la industria y capacidad es comunes de industrialización
- La arquitectura SOA permite aislar mejor a los sistemas frente a los cambios generados por otras partes de la organización (protección de las inversiones realizadas)
- La arquitectura SOA permite alinear y acercar las áreas de tecnología y negocio [31]

CAPÍTULO 4

Marco metodológico

En la programación es de vital importancia antes de comenzar a implementar la solución de un problema tener un orden y establecer distintas *prioridad*es, para esto se hace uso de una metodología de desarrollo de software.

Este trabajo especial de grado no es la excepción ya que para construir la estructura de una red social es de gran ayuda tener un conjunto de actividades estructuradas y metódicas.

Específicamente se utilizó la metodología SCRUM con algunas variantes que explicaremos más adelante en este documento en el capítulo 5, apartado 5.3. Se decidió usar esta metodología ya que es de desarrollo ágil, orientada a la obtención de resultados, a pesar de que el entorno sea cambiante y/o los requisitos estén sujetos a cambios o no estén muy bien definidos. Está basada iteraciones con entregas parciales y regulares, cada una de ellas trabajan juntos el cliente y el equipo de desarrollo lo que permite un buen alineamiento entre ellos.

4.1. Roles principales

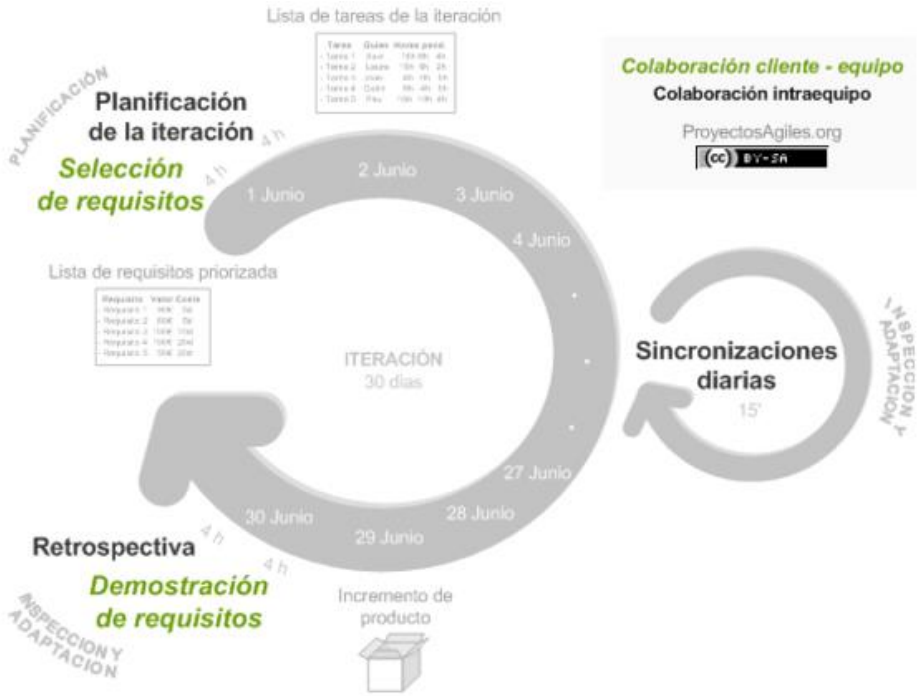
- **Cliente *(Product Owner):*** Representa la voz del cliente y del resto de interesados no implicados directamente en el proyecto. Es el encargado de definir los objetivos del proyecto y de garantizar que el equipo trabaja del modo adecuado para alcanzar dichos objetivos.
- **Facilitador *(Scrum Master):*** Encargado de asegurar que el resto del equipo no tiene problemas para abordar sus funciones y tareas. Guía y ayuda al *Scrum Team* para garantizar el cumplimiento de objetivos.
- **Equipo *(Scrum Team):*** Es el equipo encargado de desarrollar y entregar el producto. Su trabajo es imprescindible: se trata de una estructura horizontal auto-organizada capaz de auto-gestionarse a sí misma.
- **Interesados *(Stakeholders):*** Este grupo comprende aquellos perfiles interesados en el producto: directores, dueños, comerciales. Se trata de perfiles

que si bien no forman parte del equipo (Scrum Team) deben ser tenidos en cuenta.

4.2. El proceso

En Scrum un proyecto se ejecuta en bloques temporales cortos y fijos (iteraciones que normalmente son de 2 semanas, aunque en algunos equipos son de 3 y hasta 4 semanas, límite máximo de intercambio (*feedback)* y reflexión). Cada iteración tiene que proporcionar un resultado completo, un incremento de producto final que sea susceptible de ser entregado con el mínimo esfuerzo al cliente cuando lo solicite.

Figura 8: Proceso de la metodología Scrum [43]



El proceso parte de la lista de objetivos/requisitos priorizada del producto, que actúa como plan del proyecto. En esta lista el cliente prioriza los objetivos balanceando

el valor que le aportan respecto a su coste y quedan repartidos en iteraciones y entregas.

Las actividades que se llevan a cabo en Scrum son las siguientes:

4.2.1. Planificación de la iteración

4.2.1.1. Requisitos del producto (Product Backlog):

Para comenzar debemos elaborar los requisitos del producto. Recoge el conjunto de tareas, los requerimientos y las funcionalidad es requeridas por el proyecto. El único con autoridad para agregar *prioridades* es el Cliente (*Product Owner*), responsable del documento. El equipo pregunta al cliente las dudas que surgen y selecciona los requisitos más prioritarios que se compromete a completar en la iteración, de manera que puedan ser entregados si el cliente lo solicita.

4.2.1.2. Iteración (Sprint)

Los requisitos ya elaborados los dividimos en entregas parciales o fases, en función de la *prioridad* o el valor que se le dé a cada uno de los requisitos por parte del cliente. Cada uno de estos periodos recibe el nombre de **Sprint** y tienen definida una fecha de comienzo y una fecha de fin. Suelen tener una duración entre 2 y 4 semanas, el objetivo es entregar algo que funcione y vaya evolucionando conforme el usuario vaya probando, de modo que se puedan introducir los cambios necesarios, antes de que sea demasiado tarde. La estimación de esfuerzo se hace de manera conjunta y los miembros del equipo se autoasignan las tareas.

4.2.1.3. Requisitos de la iteración (Sprint Backlog)

Es el documento que recoge las tareas a entregar a la finalización del **Sprint**, junto con las horas de trabajo que van a suponer, quién las realiza y el coste. Todas las acciones que realicemos han de tener un control.

4.2.2. Ejecución de la iteración

Cada día el equipo realiza una reunión de sincronización (15 minutos máximos). Cada miembro del equipo inspecciona el trabajo que el resto está realizando (dependencias entre tareas, progreso hacia el objetivo de la iteración, obstáculos que pueden impedir este objetivo) para poder hacer las adaptaciones necesarias que permitan cumplir con el compromiso adquirido. En la reunión cada miembro del equipo responde a tres preguntas:

- ¿Qué he hecho desde la última reunión de sincronización?
- ¿Qué voy a hacer a partir de este momento?
- ¿Qué impedimentos tengo o voy a tener?

Durante la iteración el Facilitador (Scrum Master) se encarga de que el equipo pueda cumplir con su compromiso y de que no se merme su productividad.

- Elimina los obstáculos que el equipo no puede resolver por sí mismo.
- Protege al equipo de interrupciones externas que puedan afectar su compromiso o su productividad.

Durante la iteración, el cliente junto con el equipo refinan la lista de requisitos (para prepararlos para las siguientes iteraciones) y, si es necesario, cambian o replanifican los objetivos del proyecto para maximizar la utilidad de lo que se desarrolla y el retorno de inversión.

4.2.3. Inspección y adaptación

Al final del ciclo Sprint, se llevarán a cabo dos reuniones más:

4.2.3.1. Reunión de revisión de la iteración (Sprint Review Meeting):

Esta reunión es para revisar el trabajo que fue completado (o no completado) y hacer una demo de lo entregado. . En función de los resultados mostrados y de los cambios que haya habido en el contexto del proyecto, el cliente realiza las adaptaciones necesarias de manera objetiva, ya desde la primera iteración, replanificando el proyecto.

4.2.3.2. Retrospectiva de la iteración (Sprint Retrospective):

El equipo analiza cómo ha sido su manera de trabajar y cuáles son los problemas que podrían impedirle progresar adecuadamente, mejorando de manera continúa su productividad. El Facilitador se encargará de ir eliminando los obstáculos Identificado s.

4.3. Beneficios de la metodología SCRUM

Los beneficios que se obtienen son amplios y afectan a todos los actores del proyecto:

- Existe una mayor **adaptación y *flexibilidad*** ante entornos y requisitos cambiantes, permitiendo la reducción del tiempo de desarrollo de los productos.

- **Mayor control y transparencia** por parte de los *Stakeholders* sobre el proyecto. El cliente realiza un seguimiento más cercano sin tener que esperar a la entrega del producto, minimizando el riesgo de que el resultado final no le convenza.
- **Fomento del trabajo en equipo**. Se trata de un modelo basado en la autodisciplina y la autogestión repercutiendo en la responsabilidad y en la comunicación entre los distintos miembros del equipo.

CAPÍTULO 5

MARCO APLICATIVO

Tal como se describe en capítulos anteriores, existe un interés creciente en encontrar valor detrás de los millones de datos que generan los usuarios diariamente. Dependiendo de la entidad interesada ya sea educativa, estatal, empresas u organizaciones privadas o públicas el objetivo de descubrir información y generar conocimiento de los datos puede ser muy variado, identificar grupos, aplicar métricas, estudiar patrones de comportamiento de los usuarios, realizar campañas de marketing, análisis de sentimiento, etc.

Las redes sociales pueden ser una mina de oro a la hora de obtener datos para generar información, sin embargo esto presenta un gran desafío para las herramientas computacionales actuales al momento de manejar grandes volúmenes de datos.

En este trabajo especial de grado nos enfocaremos en dar respuesta a los siguientes desafíos y limitaciones:

Tabla 3: Desafíos para construir la estructura de una red social

Desafíos	Solución planteada
¿Cómo obtener los datos?	Se decidió utilizar las fuentes de datos basadas en interfaces de programación de aplicaciones (API) del caso de estudio
¿Qué forma tienen?	Según el comportamiento de una red social, el modelo de datos que se adapta es un grafo.
¿Dónde almacenarlos?	Dado su forma, una base de datos relacional no es viable, por lo que es necesario hacer uso de una base de datos NoSQL orientada a grafos.

En base a las soluciones planteadas en la **Tabla 3** se tomaron las siguientes decisiones:

5.1. Caso de estudio

Seleccionamos como caso de estudio la Red social de microblogging Twitter, se adapta al desarrollo ya que cuenta con una comunidad compuesta por más de 319 millones de usuarios activos al mes, para el primer trimestre del 2017 [44], cuyos gustos, géneros, y patrones de comportamiento son altamente variados y está destinada a la publicación de mensajes cortos no mayores a 140 caracteres de longitud.

Para esta primera fase del proyecto segmentamos el problema, enfocándonos en obtener datos de usuarios para construir la estructura de red social solo en Venezuela.

La extracción de datos se realizó a partir de una muestra de cuentas representativas en Venezuela, a dichas cuentas les haremos referencia en este documento como "*cuentas semillas*". Los seguidores de estas *cuentas semillas* se asumen como venezolanos o con una estrecha relación con esta nación.

5.1.1. Limitaciones del *API* Twitter

Como se indica el en capítulo 3 de buena parte del potencial de las redes sociales se encuentra en sus API, Twitter no es la excepción, y ofrece un generoso conjunto de funciones y procedimientos que permiten extender sus prestaciones originales en todo tipo de aplicaciones, esto indica lo permisiva que puede ser el API sin embargo nos revela también las limitaciones, por lo que busca aprovechar las ventajas y remediar las limitaciones .

Twitter es una de las redes sociales que por su naturaleza no hay muchas restricciones de privacidad, es decir, la gran mayoría de usuarios tiene público su perfil y por lo tanto también sus tweets. Por esta razón, Twitter permite el acceso a toda esta información a través de su API. Las restricciones que presenta es que existen límites de peticiones en un periodo de tiempo, es decir, existen intervalos de 15 minutos en donde se puede hacer un número máximo de peticiones. Cabe mencionar que estos límites son por usuario, no por aplicación, de esta forma cada usuario es controlado de forma independiente. Dependiendo del tipo de recurso que sea solicitado, existen dos tipos de restricciones principales: 15 peticiones cada 15 minutos

y 180 peticiones cada 15 minutos. Adicionalmente, no se permite recuperar información histórica, es decir, si se ejecuta una búsqueda sólo es posible obtener información generada 7 días atrás como máximo.

En caso de exceder el número máximo de peticiones en un periodo de tiempo, se obtiene un código de respuesta en donde se provee información acerca del recurso restringido temporalmente y el tiempo de espera para que el recurso se encuentre nuevamente disponible. Otra particularidad de Twitter, es que cuenta con streaming, lo que permite obtener información en tiempo real sin restringir el número de peticiones por un periodo de tiempo.

Para construir la estructura de la red social, es necesario obtener por cada *cuenta semilla*, sus seguidores y las personas seguidas. Estos usuarios se obtienen consumiendo los servicios "*friends/list*" y "*followers/list*". Estos servicios tienen una cantidad de peticiones permitidas y la descarga de los datos es paginada.

Tabla 4: Limitaciones del *API* Twitter en cuando a número de consultas y cantidad de datos extraídos

Limitaciones del *API* Twitter	Número
Servicio "friends/list"	15 consultas
Servicio "followers/list"	15 consultas
Paginación	200 cuentas
Tiempo de espera	15 minutos

Según la **Tabla 4** la cantidad de consultas en un instante de tiempo es proporcional a la cantidad de clientes conectados, por lo que el acceso a los datos se realizó de forma paralela por n clientes, estos a su vez descargan datos de m semillas disponibles. Así logramos minimizar sustancialmente el tiempo en espera, y el tiempo de respuesta total de la aplicación en construir el resultado final del grafo que involucre todos los usuarios y sus relaciones.

La **Tabla 5** muestra posibles escenarios

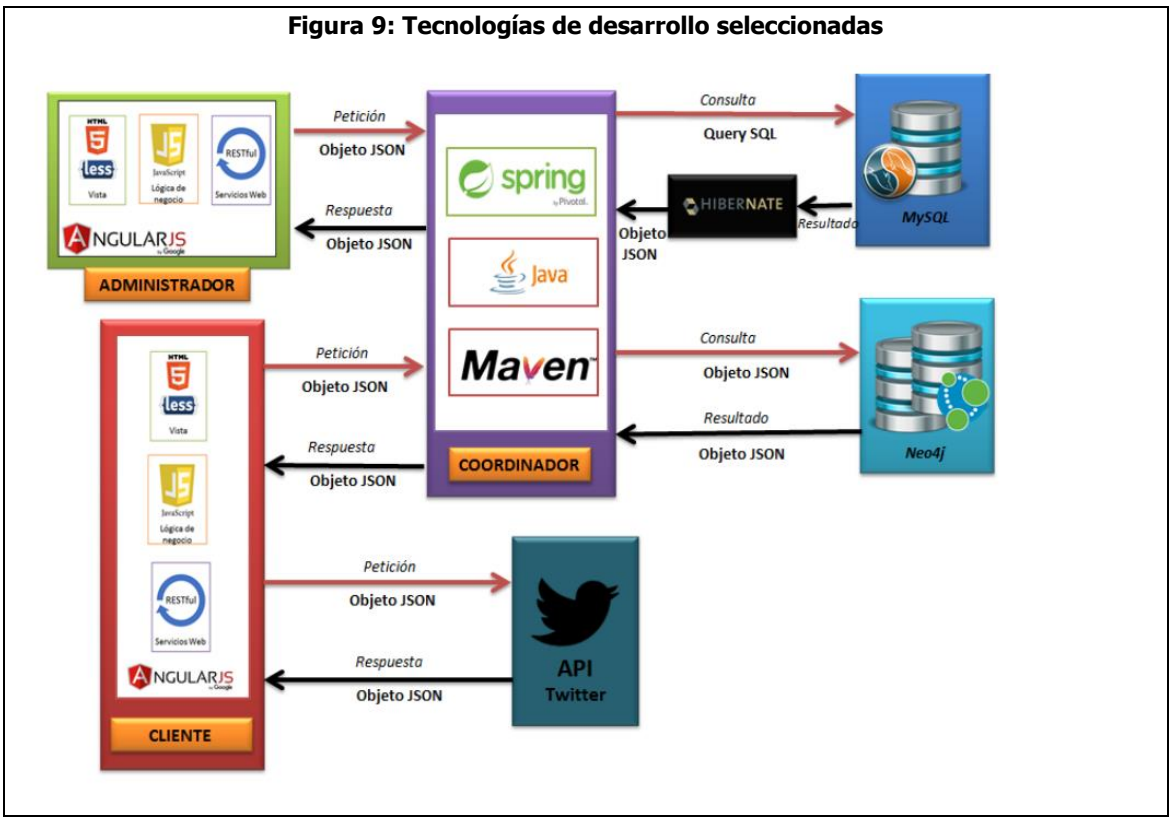
Tabla 5: Número de consultas y cantidad de datos según el número de clientes involucrados

Clientes conectados al *API*	Consultas friends/list followers/list	Cantidad de usuarios obtenidos en 15 minutos
1	15	3.000
10	150	30.000
100	1500	300.000
1000	15.000	3.000.000

De modo que, la forma de circunvalar las limitaciones del API Twitter es incrementando la cantidad de clientes que realicen descargas de datos de forma paralela de las *cuentas semillas*.

5.2. Tecnologías utilizadas

Figura 9: Tecnologías de desarrollo seleccionadas



La **figura 9** muestra las tecnologías seleccionadas según lo expuesto en el capítulo 3. Se desarrolló la herramienta de este Trabajo Especial de Grado, bajo tecnologías clientes como HTML5, LESS y AngularJs, y del lado del servidor, Java,

entorno de trabajo (*framework*) Spring*, base de datos MySQL y Neo4j** Hibernate***.

Para el control de versiones utilizamos Git y el servicio de alojamiento Bitbucket.

(*) Seleccionamos el *framework* Spring por su robustez, al poseer una gran cantidad de dependencias que facilitan el manejo servicios WEB, conexiones con base de datos relacionales y de grafos, e incluso para trabajos futuros módulo de integración con social media y Hadoop.

(**) Seleccionamos la base de datos Neo4j de entre las demás bases de datos orientadas a grafos por su alto desempeño y sobre todo por la cantidad de documentación disponible lo cual facilita el desarrollo.

(***) *Hibernate* es una herramienta de mapeo objeto-relacional (ORM) para la plataforma Java que facilita el mapeo de atributos entre una base de datos relacional tradicional y el modelo de objetos de una aplicación, mediante archivos declarativos (XML) de las entidades que permiten establecer estas relaciones.

5.3. Aplicación de la metodología

Como se indica en el capítulo 4 el método de desarrollo *SCRUM,* es la metodología que mejor se adaptó a las necesidades del proyecto, y al ser un método de desarrollo ágil, el proceso se hizo iterativo e incremental y de esta manera, se obtuvo una mejora continua en el producto. Sin embargo, para el desarrollo del sistema se realizaron adaptaciones a la metodología que se explican a continuación.

En *SCRUM* se identifican 4 roles, como se expone en el apartado 4.1 de este documento no obstante, para el desarrollo de este trabajo especial de grado estan involucrados solo 3 de ellos, como se muestra en la siguiente tabla.

Tabla 6: Asignación de roles

Roles	Representado por
Cliente	Tutores Jesús Lares y José Sosa
Facilitador	Carla Navas
Equipo	Ashtar Márquez y Carla Navas
Interesados	No aplica

Las reuniones diarias del equipo de desarrollo para mostrar las nuevas versiones del producto, no se realizaron. Fueron reemplazadas por dos reuniones a la semana con el quipo los días martes y jueves.

En la definición de *SCRUM* se sugiere equipos de entre 5 y 9 personas que trabajan juntos en la misma localización física para poder maximizar la comunicación entre ellos mediante conversaciones cara a cara, diagramas en pizarras blancas, etc [45]. Sin embargo, para el desarrollo de este trabajo especial de grado, no todas las reuniones ni el trabajo del equipo fueron realizados cara a cara y en la misma ubicación, sino que fueron realizados vía *online*. En los tiempos establecidos anteriormente.

5.3.1. Planificación de la iteración

5.3.1.1. Requisitos del producto (Product Backlog):

Se planteó la solución basada en el diseño de una arquitectura orientada a servicio (SOA) compuesta por 3 módulos como se muestra en la siguiente tabla que implementen los siguientes requerimientos:

Tabla 7: Módulos y requerimientos de la herramienta implementada

Módulo	**Descripción**	Funcionalidad **es asociadas**
Administrador	Este módulo permite a un usuario experto miembro de la organización que quiere realizar el estudio tener una cuenta personal para definir las cuentas que semillas que desea utilizar y el nivel de profundidad. Este es un valor entero que indica la cantidad de iteraciones que se deben realizar en la aplicación para construir el grafo. Fue desarrollado con tecnologías HTML, JavaScript y MySQL.	▪ Iniciar sesión ▪ Crear semillas ▪ Enviar semillas
Coordinador		

	Este módulo es el *core* del proyecto, contiene la lógica de negocio para paralelizar tanto los clientes como las semillas. Esto módulo recibe del módulo administración un conjunto de cuentas semillas, que denotamos nivel 0, a partir de las cuales se construirá el grafo en cuestión. Estas semillas serán distribuidas en los n clientes disponibles que descargan la información de Twitter. Una vez recibe los datos de los clientes, arma la estructura de la red almacenando sin duplicación los usuarios en Neo4j. Será Desarrollado con tecnología Java, bajo el *Framework* Sprint, y base de datos orientada a grafos Neo4j.	▪ Recibir semillas ▪ Enviar semillas ▪ Crear conexión con Neo4j ▪ Verificas semillas ▪ Crear nodo ▪ Actualizar relaciones ▪ Manejar conexión con el módulo de clientes ▪ Implementar servicio de *Long Polling** ▪ Paraleliza clientes ▪ Paralelizar semillas
Clientes consultores	Este módulo es el encargado de la extracción masiva de los datos, realiza la conexión con el *API* de Twitter, descarga los datos según las semillas recibidas por el coordinador. Esta implementado como un cliente WEB para superar las restricciones que presenta el *API* en cuanto a la cantidad de peticiones por unidad de tiempo. Fue desarrollado con tecnología HTML y AngularJs.	▪ Autenticar con *API* de Twitter ▪ Consumir servicios del *API* ▪ Conectar con el módulo coordinador

* El *long polling* en nuestro caso es que cada *cliente* inicia una conexión con el *coordinador* para indicar que está disponible para recibir las *cuentas semillas*. Este *cliente* se quede esperando por *cuentas semillas* por un tiempo limitado. En caso de que se agote el tiempo establecido el *coordinador* envía una señal de tiempo agotado (*timeout)* y el *cliente* debe iniciar de nuevo la conexión.

Según las funcionalidades expuestas en la **Tabla 7** los requerimientos del producto (*Product Backlong*) son los siguientes:

Tareas generales:

- Seleccionar una base de datos NoSQL orientada a grafos
- Estudiar a detalle la estructura y funcionamiento de las redes sociales
- Estudiar las *API*s de las redes sociales en cuanto a sus comportamiento y limitaciones.
- Seleccionar un caso de estudio
- Configurar el *framework Spring*
- Configurar la base de datos relacional, MySQL
- Configurar la base de datos orientada a grafos, Neo4j

Módulo de administración:

- Diseñar vistas de la interfaz de usuario *(frontend)*
- Conectar servicios de autenticación con las vistas
- Implementar servicio de autenticación de usuario en el lado del servidor *(backend)*
- Implementar servicio de crear semillas en el lado del servidor *(backend)*
- Integrar servicio de crear semillas con la interfaz de usuario
- Implementar servicio de listar semillas según el usuario autenticado
- Implementar el servicio para enviar semillas al coordinador

Módulo de clientes:

- Realizar pruebas de conexión y descarga con el API del caso de estudio
- Implementar sincronización con el API del caso de estudio
- Implementar servicio de descarga de datos asociados a la cuenta semilla
- Implementar servicio de descarga de amigos (*Friends*) asociados a la cuenta semilla
- Implementar servicio de descarga de seguidores (*Followers*) asociados a la cuenta semilla

- Diseñar vista de la interfaz de usuario para autenticar con Twitter
- Integrar servicio de sincronizar usuario con la vista
- Integrar servicio de conexión de Long Polling con el coordinador
- Integrar servicio de envió de semillas al coordinador
- Manejar paginación en el proceso de descarga de las listas de amigos (*Friends*) y seguidores (*Followers*)

Módulo Coordinador

- Implementar servicio de Recibir semillas por parte del cliente
- Implementar servicio de Enviar semillas al cliente
- Crear conexión con Neo4j
- Implementar verificación de existencia de las cuentas semillas en la base de datos de grafo
- Verificar nivel de profundidad
- Manejar consistencia en la cola de cuentas de cuentas semillas pendientes
- Implementar servicio de *Long Polling*
- Implementar paralelización de clientes
- Implementar paralelización de semillas

5.3.1.2. Requisitos de la iteración

Los requisitos de cada iteración se explican con detalle en el apartado 5.3.2.

5.3.2. Ejecución de la iteración

La etapa de iteración se llevó a cabo en un lapso de 9 iteraciones *(spring).*

5.3.2.1. Iteración 0

Fecha de inicio: 16 de enero de 2017

Fecha fin: 26 de enero de 2017

***Requisitos de la iteración* (Spring Backlog):**

- Seleccionar una base de datos NoSQL orientada a grafos
- Estudiar a detalle la estructura y funcionamiento de las redes sociales
- Estudiar las *API*s de las redes sociales en cuanto a sus comportamiento y limitaciones.

- Seleccionar un caso de estudio
- Configurar el *framework* spring
- Configurar la base de datos relacional
- Configurar la base de datos orientada a grafos

Descripción de la solución:

- **Seleccionar una base de datos NoSQL orientada a grafos:** Una vez estudiadas las distintas base de datos orientados a grafos se tomó la decisión de utilizar Neo4j por tener buen desempeño y estar altamente documentada.
- **Estudiar a detalle la estructura y funcionamiento de las redes sociales:** Los detalles de esta tarea se encuentran en el capítulo 3 apartado 3.2
- **Estudiar las *API*s de las redes sociales en cuanto a sus comportamiento y limitaciones:** Los detalles de esta tarea se encuentran en el capítulo 3 apartado 3.3
- **Seleccionar un caso de estudio:** Fue seleccionado como caso de estudio la red de microblogging Twitter, los detalles se encuentran en este capítulo apartado 5.1.1
- **Configurar el *framework* spring:** Se instalaron las herramientas necesarias tales como el IDE Eclipse, Maven como manejador de dependencias para hacer uso del *framework*.
- **Configurar la base de datos relacional:** Se creó un usuario y contraseña asociados a la base de datos MySQL. Se crearon las tablas necesarias según el modelo plasmado en la **Figura 8***.*
- **Configurar la base de datos orientada a grafos:** Se realizó la descarga de la base de datos de la página oficial [46] se agregaron las dependencias requeridas por el *framework* spring.
- **Se decidió utilizar la base de datos orientada a grafos Neo4j**: El modelo de esta base de datos se detalla en la **Figura 9***.*

5.3.2.2. Iteración 1

Fecha de inicio: 27 de enero de 2017

Fecha fin: 12 de febrero de 2017

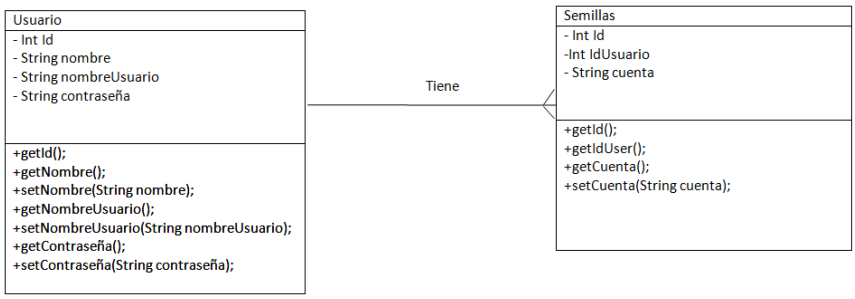
***Requisitos de la iteración* (*Spring Backlog*):**

- Implementar servicio de autenticación de usuario en el lado del servidor *(backend)*
- Implementar servicio de crear semillas en el lado del servidor *(backend)*
- Implementar servicio de listar semillas según el usuario autenticado

Descripción de la solución:

- **Implementar servicio de autenticación de usuario en el lado del servidor en el módulo de administración:** Para esta etapa se realizó manejo de sesiones a través de *token*, donde el usuario registrado recibe un *token* único que lo identifica, este es incluido en la cabecera de todos los servicios invocados.
- **Implementar servicio de crear semillas en el lado del servidor *(backend):*** Para esta tarea se creó un modelo semilla del lado del servidor, que hace relación con la tabla *semillas* en la base de datos según se muestra en la **Figura 10.** Este servicio que recibe del lado del cliente un objeto *JSON semillas* y lo almacena en la base de datos MSQL.

Figura 10: Modelo de base de dato del módulo administrador (MySQL)



- **Implementar servicio de listar semillas según el usuario autenticado:** Este es un servicio que corresponde al método GET y recibe como parámetro en la URL (*query String)* el ID del usuario que lo invocó, y retorna un objeto que contiene la lista de semillas creadas por este.

5.3.2.3. Iteración 2

Fecha de inicio: 13 de febrero de 2017

Fecha fin: 26 de febrero de 2017

***Requisitos de la iteración* (*Spring Backlog*):**

- Diseñar vistas de la interfaz de usuario *(frontend)*
- Conectar servicios de autenticación con las vistas
- Integrar servicio de crear semillas con la interfaz de usuario
- Implementar el servicio para enviar semillas al coordinador

Descripción de la solución:

- **Diseñar vistas de la interfaz de usuario *(frontend)*** para esta tarea se realizó la maquetación de la vista.
- **Conectar servicios de autenticación con las vistas:** En esta tarea se creó un servicio REST que corresponde al método *POST*, y envía un objeto *JSON* con los datos del el usuario introdujo (nombre de usuario y contraseña)
- **Integrar servicio de crear semillas con la interfaz de usuario:** En esta tarea se creó un servicio REST que corresponde al método *POST*, y envía un objeto *JSON* con los datos de la semilla (nombre de la semilla).
- **Implementar el servicio para enviar semillas al coordinador:** Este servicio corresponde a un método *POST* y recibe un objeto *JSON* que contiene la lista de semillas que el usuario selecciono para ser procesadas.

5.3.2.4. Iteración 3

Fecha de inicio: 27 de febrero de 2017

Fecha fin: 12 de marzo de 2017

***Requisitos de la iteración* (*Spring Backlog*):**

- Realizar pruebas de conexión y descarga con el *API* del caso de estudio
- Implementar sincronización con el *API* del caso de estudio
- Implementar servicio de descarga de datos asociados a la cuenta semilla

- Implementar servicio de descarga de amigos (*Friends*) asociados a la cuenta semilla
- Implementar servicio de descarga de seguidores (*Followers*) asociados a la cuenta semilla

Descripción de la solución:

- **Realizar pruebas de conexión y descarga con el *API* del caso de estudio:** se utilizó la cuenta personal del facilitador para realizar las pruebas de conexión y descargas. Fueron tareas sencillas.
- **Implementar sincronización con el *API* del caso de estudio (Twitter):** esta tarea se realizó mediante OAuth.1.0, autenticación provista por Twitter. Esta última, consume un servicio "*account/verify_credentials".* del *API* de Twitter y recibe como respuesta un *token*. Este *token* fue incluido en la cabecera de todos los servicios que consumen el *API* a lo largo del proyecto.
- **Implementar servicio de descarga de datos asociados a la cuenta semilla:** se hace uso del servicio "*users/show*" contenido en el *API*. Este retorna un objeto *JSON* con los datos del usuario que consumió el servicio.
- **Implementar servicio de descarga de amigos (*Friends*) asociados a la cuenta semilla:** se hace uso del servicio "*Friends/list*" contenido en el *API*. Recibe como parámetro en la URL *(query string)* el nombre de usuario *(username)* de la cuenta semilla raíz. Retorna un objeto *JSON* que contiene un arreglo de objetos de las cuentas de usuarios que son seguidos por la cuenta semilla.
- **Implementar servicio de descarga de seguidores (*Followers*) asociados a la cuenta semilla:** se hace uso del servicio "*Followers/list*" contenido en el *API*. Recibe como parámetro en la URL *(query string)* el nombre de usuario *(username)* de la cuenta semilla raíz. Retorna un objeto *JSON* que contiene un arreglo de objetos de las cuentas de usuarios que siguen a la cuenta semilla.

5.3.2.5. Iteración 4

Fecha de inicio: 13 de marzo de 2017

Fecha fin: 26 de marzo de 2017

***Requisitos de la iteración* (Spring Backlog):**

- Diseñar vista de la interfaz de usuario para autenticar con Twitter
- Integrar servicio de sincronizar usuario con la vista
- Manejar paginación en el proceso de descarga de las listas de amigos (*Friends*) y seguidores (*Followers*)

Descripción de la solución:

- **Diseñar vista de la interfaz de usuario para autenticar con Twitter:** para esta tarea se realizó la maquetación de la vista.
- **Integrar servicio de sincronizar usuario con la vista:** se implementó servicio REST "*connectTwitter"* que consume el servicio de autenticación implementado en la iteración 3 (*account/verify_credentials).*
- **Manejar paginación en el proceso de descarga de las listas de amigos (*Friends*) y seguidores (*Followers*):** en la iteración 3 se recibe un objeto *JSON* que con contiene un arreglo de objetos de las cuentas de usuarios que siguen a la cuenta semilla raíz, adicionalmente este objeto contiene un cursor que hace referencia a la siguiente página del arreglo recibido. Para obtener la cantidad total de cuentas de usuarios (seguidores y amigos) se llama de manera recursiva a la función que contiene el servicio *Followes/list o Friends/list* según sea el caso hasta que el valor del cursor sea igual a el valor 0, lo que implica que no existen más páginas que visitar.

5.3.2.6. Iteración 5

Fecha de inicio: 27 de marzo de 2017

Fecha fin: 09 de abril de 2017

***Requisitos de la iteración* (Spring Backlog):**

- Implementar paralelización de clientes
- Implementar paralelización de semillas

Descripción de la solución:

- **Implementar paralelización de clientes:** para esta tarea se utiliza una estructura de datos cola donde se agregan los clientes disponibles. Estos serán desecolados por el coordinador creando un hilo que una semilla cada cliente disponible.

- **Implementar paralelización de semillas:** para esta tarea se utiliza una estructura de datos cola donde se agregan las cuentas semillas que ya agotaron la cantidad de peticiones a los servicios de consulta de *Friends* y *followers* por un cliente determinado. Se guarda el cursor por donde quedo la paginación de ambas listas. Esta cuenta semilla será agregada en una cola de cuentas semillas pendientes para ser desagregada y entregada a otro cliente en disponible para que continúe su descargar a partir del página almacenada en el cursor.

5.3.2.7. Iteración 6

Fecha de inicio: 10 de abril 2017

Fecha fin: 23 de abril 2017

***Requisitos de la iteración* (Spring Backlog):**

- Integrar servicio de conexión de Long Polling con el coordinador
- Integrar servicio de envió de semillas al coordinador
- Implementar servicio de *Long Polling*

Descripción de la solución:

- **Implementar servicio de *Long Polling:*** este servicio consta de dos partes, registro del cliente y envió de los datos. Se implementa el uso de colas como estructura de datos para el manejo de concurrencia. Se opera con dos colas, una cola para manejar el conjunto de peticiones concurrentes de clientes disponibles para descargar datos del API y otra cola que contiene el conjunto de cuentas semillas pendientes por distribuir.

    - **Parte I - Registro de Cliente:** este servicio corresponde un método GET que recibe como parámetro en la URL un Identificar único (ID) del cliente que está haciendo la petición. El coordinador para manejar la concurrencia de peticiones agrega el ID del cliente solicitante de este modo se tiene en la cola el conjunto de clientes disponibles que esperan recibir una cuenta semilla.

    - **Parte II - Envío de datos:** este servicio corresponde a un método GET y recibe como parámetro en la URL el Identificado r (ID) del

cliente que está solicitando el envío de cuentas semillas. El coordinador abre una conexión constante con este cliente por un lapso de 2 minutos. Si en ese lapso de tiempo el coordinador dispone de cuentas en la cola cuentas semillas pendientes, desagrega una cuenta semilla y la envía al cliente. En caso que se agote el lapso de tiempo de 2 minutos el cliente debe reiniciar el proceso, volviendo a la parte I.

- **Integrar servicio de conexión de Long Polling con el coordinador:** este servicio corresponde un método GET que envía como parámetro en la URL el Identificador (ID) del cliente que solicitante tal como se detalla en el punto anterior.
- **Integrar servicio de envió de semillas al coordinador:** una vez descargado los datos (seguidores y amigos) de la cuenta semilla desde *API*, el cliente invoca a un servicio *POST* que le envía como parámetro un objeto *JSON* que contiene los datos de la cuenta semilla, una lista con sus seguidores y otra lista con sus amigos.

5.3.2.8. Iteración 7

Fecha de inicio: 24 de abril de 2017

Fecha fin: 07 de mayo de 2017

***Requisitos de la iteración* (Spring Backlog):**

- Implementar servicio de recibir la lista de amigos y seguidores de la cuenta semilla por parte del cliente
- Implementar servicio de enviar semillas al cliente
- Crear conexión con Neo4j

Detalles de la solución:

- **Implementar servicio de recibir la lista de amigos y seguidores de la cuenta semilla por parte del cliente:** Este servicio corresponde a un método *POST* que recibe como parámetro un objeto *JSON* que contiene datos de la cuenta semilla, una lista con sus seguidores y otra lista con sus amigos.

- **Implementar servicio de enviar semillas al cliente:** Este servicio corresponde a un método GET. Para esta tarea el coordinador verifica la cola de cuentas semilla pendientes y la cola de clientes disponibles, si hay semillas y existen clientes disponibles, desagrega el primer cliente de la lista y le envía una cuenta semilla desagregada de la cola de cuentas semillas pendientes.
- **Crear conexión con Neo4j:** en esta tarea se agregaron las dependencias asociadas a Neo4j.

5.3.2.9. Iteración 8

Fecha de inicio: 08 de mayo de 2017

Fecha fin: 15 de mayo de 2017

***Requisitos de la iteración* (*Spring Backlog*):**

- Implementar verificación de existencia de las cuentas semillas en la base de datos de grafo
- Verificar nivel de profundidad
- Manejar consistencia en la cola de cuentas de cuentas semillas pendientes

Detalles de la solución:

- **Implementar verificación de existencia de las cuentas semillas en la base de datos de grafo:** en esta tarea se recibe un objeto (El objeto se muestra en la *Figura 11* con los datos de la cuenta semilla raíz, una lista correspondiente a los amigos (*friends*), otra lista correspondiente a los seguidores (*followers*) y el nivel de profundidad que indicó el usuario administrador para trabajar.

  El flujo de esta tarea va de la siguiente manera:

  1. El coordinador crea un objeto auxiliar del tipo *objectTwitter,* este objeto contiene datos de la cuenta semilla raíz (ie, atributos del nodo *TwitterUser*, (el objeto se muestra en la *Figura 7*) y el nivel de profundad con el que se está trabajando.
  2. El coordinador verifica si la cuenta semilla raíz ya se encuentra almacenada en el grafo.
     a. Si ya está almacenada en el grafo, se obtiene la referencia a ese nodo y se almacena en una variable auxiliar del tipo *TwitterUser.*

i. Se verifican uno a uno si los elementos de las listas de seguidores y amigos se encuentran almacenados en el grafo.

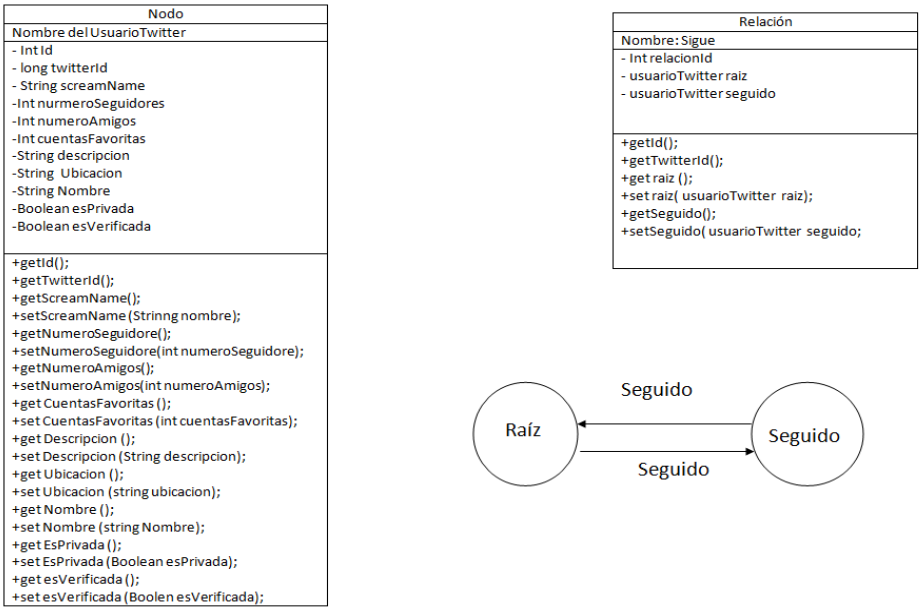
ii. Si ya están almacenados se verifica si existe la relación *Follows* que conecta la cuenta semilla raíz y la cuenta de ese amigo o seguidor seleccionado en la lista. Y se verifica el novel de profundidad*.

iii. Si no existe la relación antes mencionada la crea mediante un objeto *Follows.* Y se verifica el nivel de profundidad*.

b. Si la cuenta raíz no se encuentra almacenada se crea un objeto del tipo *TwitterUser* y se almacena en el grafo. Se realizan los pasos *a* y *b* para cada uno de los elementos de la lista de amigos (*Friends) y seguidores (Follower).*

( ) El nivel de* profundidad *proporcionado por el usuario administrador indica el número de recursiones del algoritmo para construir el grafo de la red. Esta funcionalidad se explica en el punto siguiente.*

Figura 11: Modelo de la BD orientada a grafos en Neo4j



- **Verificar nivel de profundidad:** según lo expuesto en el punto anterior se debe actualizar el valor entero que indica el nivel de profundidad del grafo en cada recursión.

  Esto sucede mediante el siguiente flujo:

  1. El coordinador verifica si el nivel de profundidad de la cuenta semilla raíz que ya está almacenada en el grafo, es menor o igual a cero.
     a. Si es menor o igual a cero, finaliza la recursión y no se agregan las listas de *friends* y *follower* en las cuentas semillas pendientes.
     b. Si el valor es mayor a cero se agrega en la cola de cuentas semillas pendientes y se realiza la verificación de consistencia, esta se explica con detalle en el siguiente ítem.

- **Manejar consistencia en la cola de cuentas de cuentas semillas pendientes:** para conservar la consistencia de las cuentas semillas pendientes se implementó un tipo de dato *set* de java, este no es más que una lista que no admite elementos repetidos. Este set contiene los mismos elementos de la cola de cuentas semillas pendientes. Al recibir una lista de cuentas semillas (amigos y seguidores) se va a comparar en primer lugar contra el set, para verificar su existencia en dicha cola:
     a. Si está, actualiza el valor del nivel de profundidad al mayor.
     b. Si no está agrega el objeto en el set y agrega en la cola de cuentas semillas pendientes y.

CAPÍTULO 6

PRUEBAS FUNCIONALIDAD

En este capítulo cumpliremos con la última fase de la metodología correspondiente a inspección y adaptación basadas en las pruebas del software.

Estas pruebas fueron realizadas en reuniones con el equipo a medida que se iba desarrollando las funcionalidades en cada iteración, en pro de verificar y comprobar la funcionalidad y continuar con la planificación y desarrollo de la siguiente iteración.

1. Probar servicio de crear semillas con la interfaz de usuario

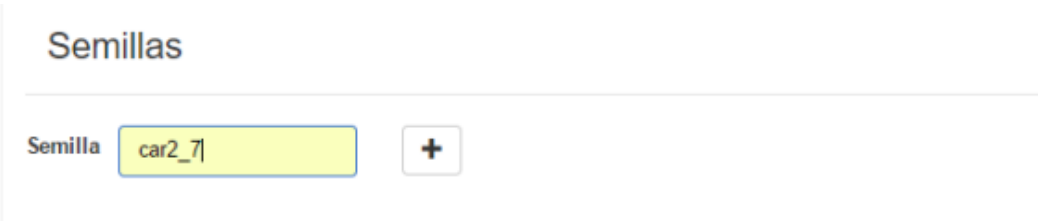
Descripción: Esta prueba consistió en verificar que la creación de semillas se realizó correctamente a través de la vista administrativa.

Caso de prueba: para probar el servicio se introdujo la semilla "carl2_7".

Respuesta esperada: Visualizar en la base de datos en la tabla *seed*, la cuenta "carl2_7"

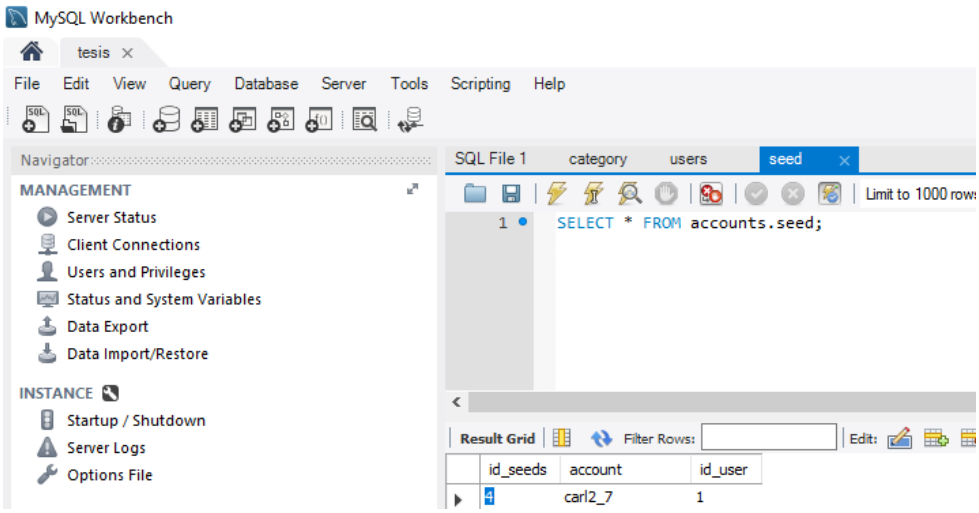
Resultado de la ejecución: En efecto insertó la cuenta en la base de datos.

- Inserción del lado del Administrador



B

- Base de Datos:



Falla: Al momento de probar se tuvo dificultad a la hora de acceder al servicio por parte del módulo administrador (*front-end*) debido a que el módulo coordinador (*back-end*) requería que en la cabecera de cada servicio se le enviara el identificador (token) de sesión que se le entregaba al administrador cuando un usuario iniciaba sesión.

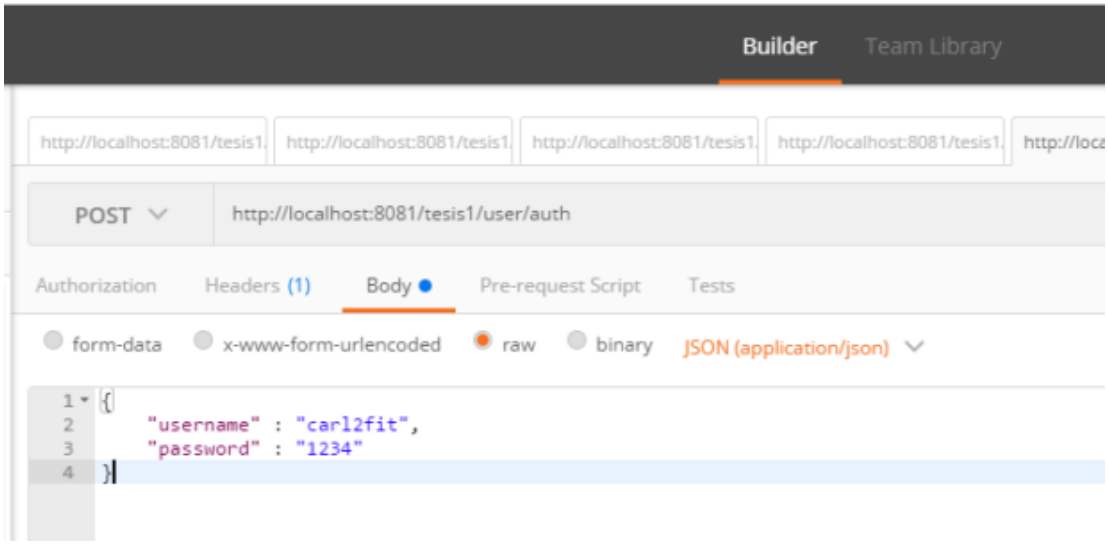
Solución:

Se colocó en la cabecera del servicio el toquen que requería, para verificar que en efecto se solucionó el problema volvimos a agregar la semilla antes mencionada y verificamos que en efecto se encontraba en la base de datos.

2. Listar semillas según el usuario autenticado

Descripción: Para probar este servicio del lado del coordinador (back-end) se usó postma, herramienta que permite hacer llamadas a servicios http, para ello se tuvo que autenticar al usuario a través del servicio http://localhost:8081/tesis1/user/auth

Caso de prueba: Para ello lo hicimos con el usuario carl2fit



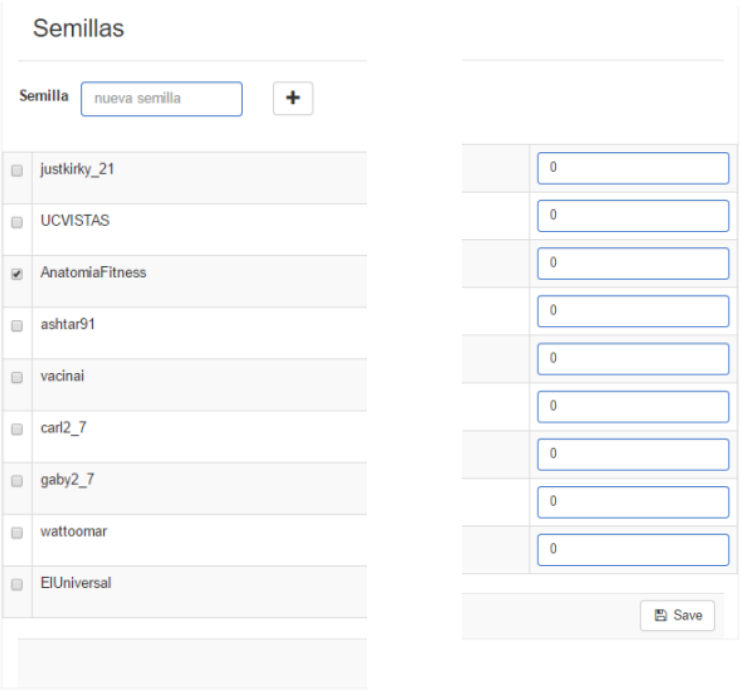
Una vez llamado al servicio autenticación, usamos el siguiente caso de prueba:

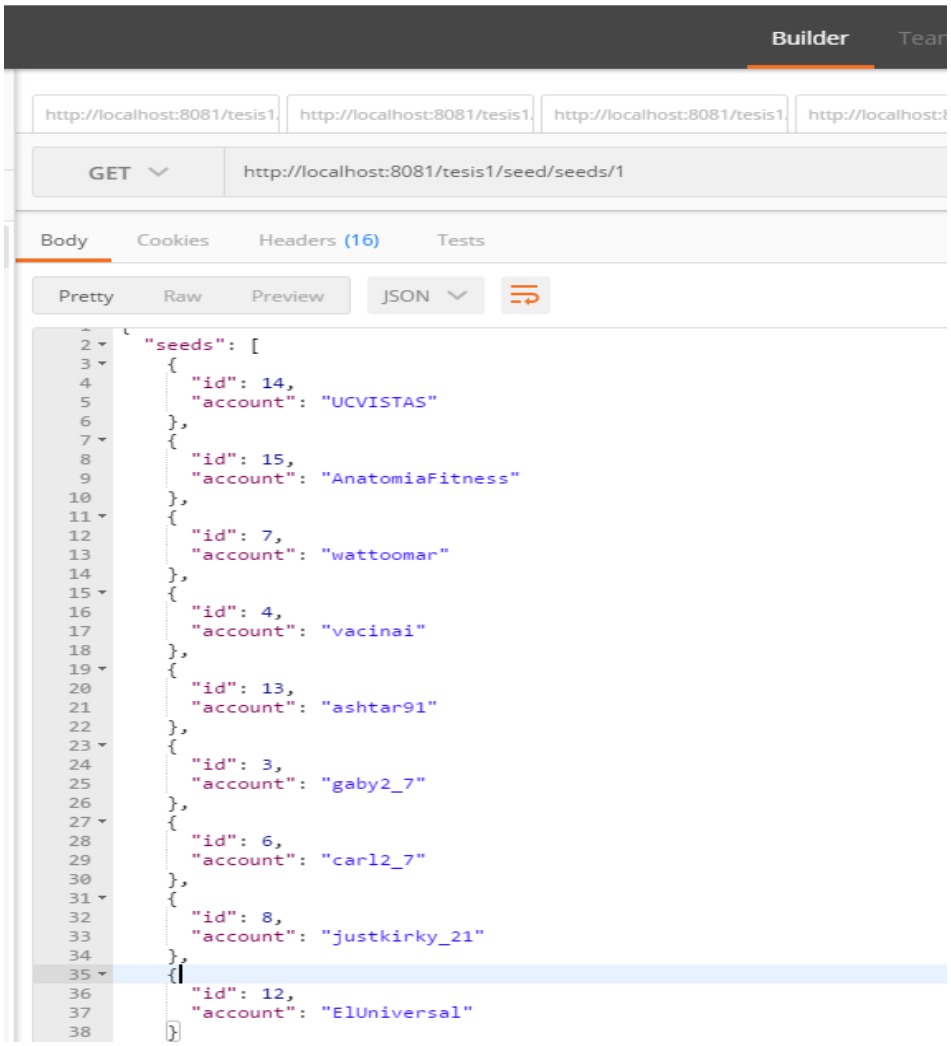
Usuario: carl2fit

URL : http://localhost:8081/tesis1/seed/seeds/1

Respuesta esperada: una lista con objetos semillas con las cuentas UCEVISTAS, carl2_7, AnatomiaFitness, wattoomar, ashtar91, gaby2_7,justkirty_21, Eluniversal, vcinai

Resultado de la ejecución: Se obtuvo la lista de semillas esperada





3. Verificar extracción de información, seguidores (*followers*) y amigos (*Friends*) de la semilla dada

Descripción: El objetivo de esta prueba es verificar que las consultas al api de twitter se realizaron de manera exitosa, y en efecto se creara el objeto que el coordinador debía recibir por parte del cliente

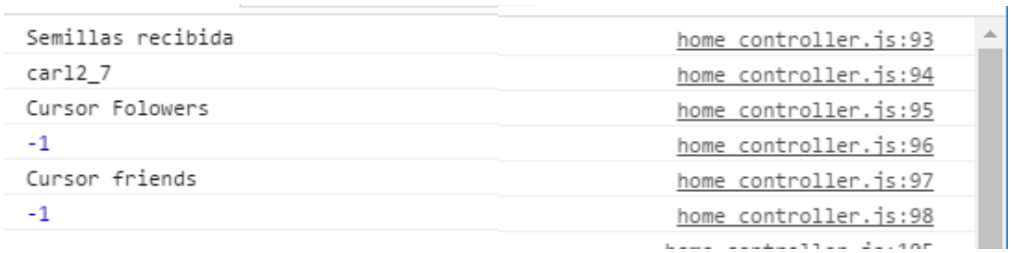
Caso de prueba: cuenta semilla: carl2_7 con nivel=0

Resultado esperado: Un objeto con información de la semilla y dos listas seguidores (*followers*) y amigos (friends)

Resultado de la ejecución: En efecto se visualizó el objeto esperado

- Semilla Enviada en el administrador

- Recepción del lado del cliente



- Resultado de la descarga

4. Verificación de cursores del lado del cliente

Descripción: Esta prueba consistió en verificar que al agotarse las 15 solicitudes que provee el API por cliente, para servicios "friends/list" y "followers/list" y aún hayan amigos o seguidores pendientes por descargar de la *cuenta semilla*, el *cliente* genere un objeto con información de *la cuenta semilla* con una lista parcial de los amigos y seguidores pendientes y los cursores correspondientes para que otro cliente pueda seguir descargando las cuentas restantes.

Caso de pruebas:

- **Semilla**: AnatomiaFitness cuenta con 4000 seguidores por lo cual deben usarse dos clientes para su descarga completa
- **Usuario de Twitter para el cliente**: carl2_7

Resultado esperado: Un objeto con información de la *cuenta semilla*, dos cursores *followersCursor* y *friendsCursor* y una lista parcial de sus amigos y seguidores pendientes por descargar. En este caso *followersCursor* debe ser mayor a cero (>0) lo que indica que esta *cuenta semilla* un cliente no pudo descargar la lista completa de seguidores.

Resultado de la ejecución: En efecto el objeto creado fue el correcto nótese que el cursor de followed es mayor a 0 por lo que significa que aún quedan seguidores por descargar para esa semilla.

5. Verificar que un segundo cliente reciba la semilla que no pudo ser totalmente descargada en la prueba anterior y que reciba el cursor adecuado

Descripción: Esta prueba es consecuencia del caso anterior, se realizó para verificar si otro cliente recibía la *cuenta semilla* con el cursor correcto para seguir descargando sus seguidores.

Caso de prueba:

- **Semilla**: AntomiaFitneess
- **Cliente:** zirneo

Respuesta esperada: Un objeto por parte del módulo coordinador con *followersCursor* = 1410582526713557200, cursor con el que terminó la prueba anterior.

Respuesta de la ejecución: se obtuvo el objeto esperado

6. Verificar que cuando el nivel de descarga del grafo sea mayor a cero los clientes reciban a los seguidores y amigos de la semilla raíz para ser descargados

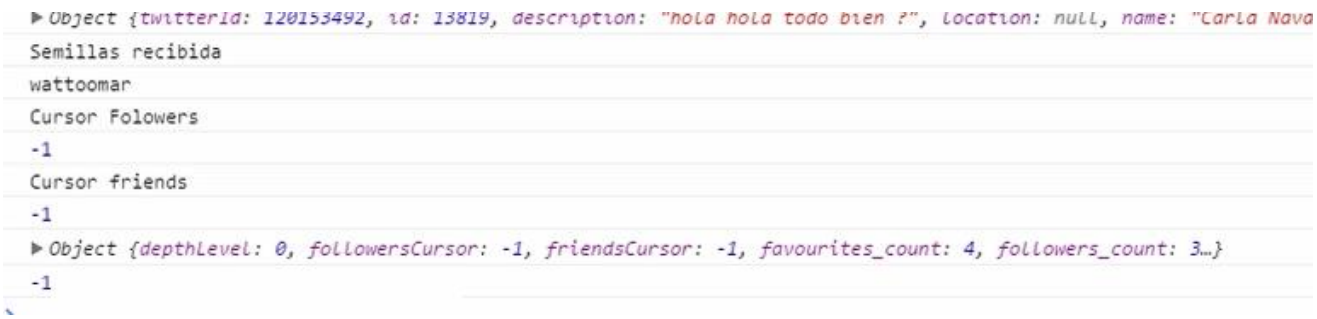
Descripción: Este caso se realizó para comprobar que los niveles de descarga del grafo se estaban realizando de manera adecuada.

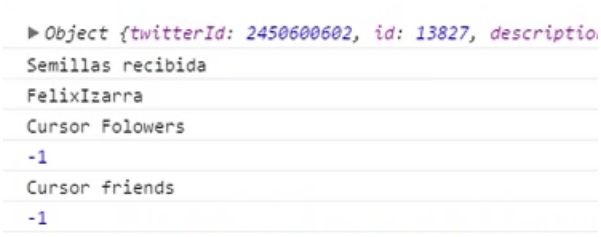
Caso de prueba:

- **Semilla base:** carl2_7
- **Clientes** : ashtarm91, carl2_7,zinero,gaby2_7

Respuesta esperada: por cada cliente autenticado recibir una semilla que corresponden a los seguidores (*followers)* o amigos (*friends)* de la *cuenta semilla* base, uno de ellos FelixIzarra, wattoomar, wallaceHamilto9

Resultado de la Ejecución: en efecto una vez descargados los datos de la *cuenta semilla* base, los clientes siguientes recibían de manera exitosa la cuentas de usuarios de sus amigos y seguidores.





7. Inserción de nodos nivel 0 y 1

Descripción: esta prueba consistió en verificar si dada una cuentasemilla totalmente descargada esta se almacena en el grafo y crea sus relaciones según sus niveles:

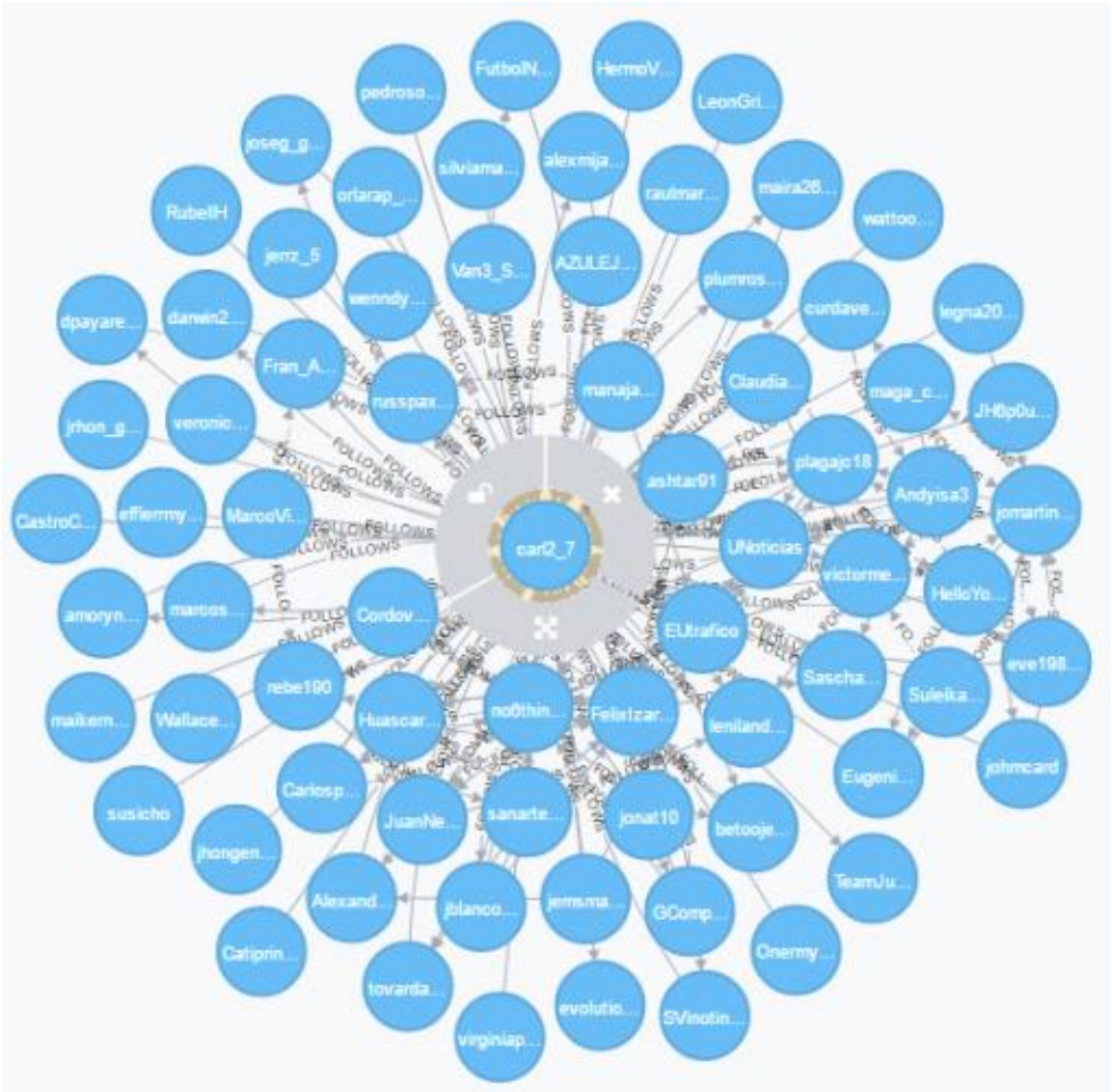
- **Semilla Base:** Carl2_7

Resultado Esperado:

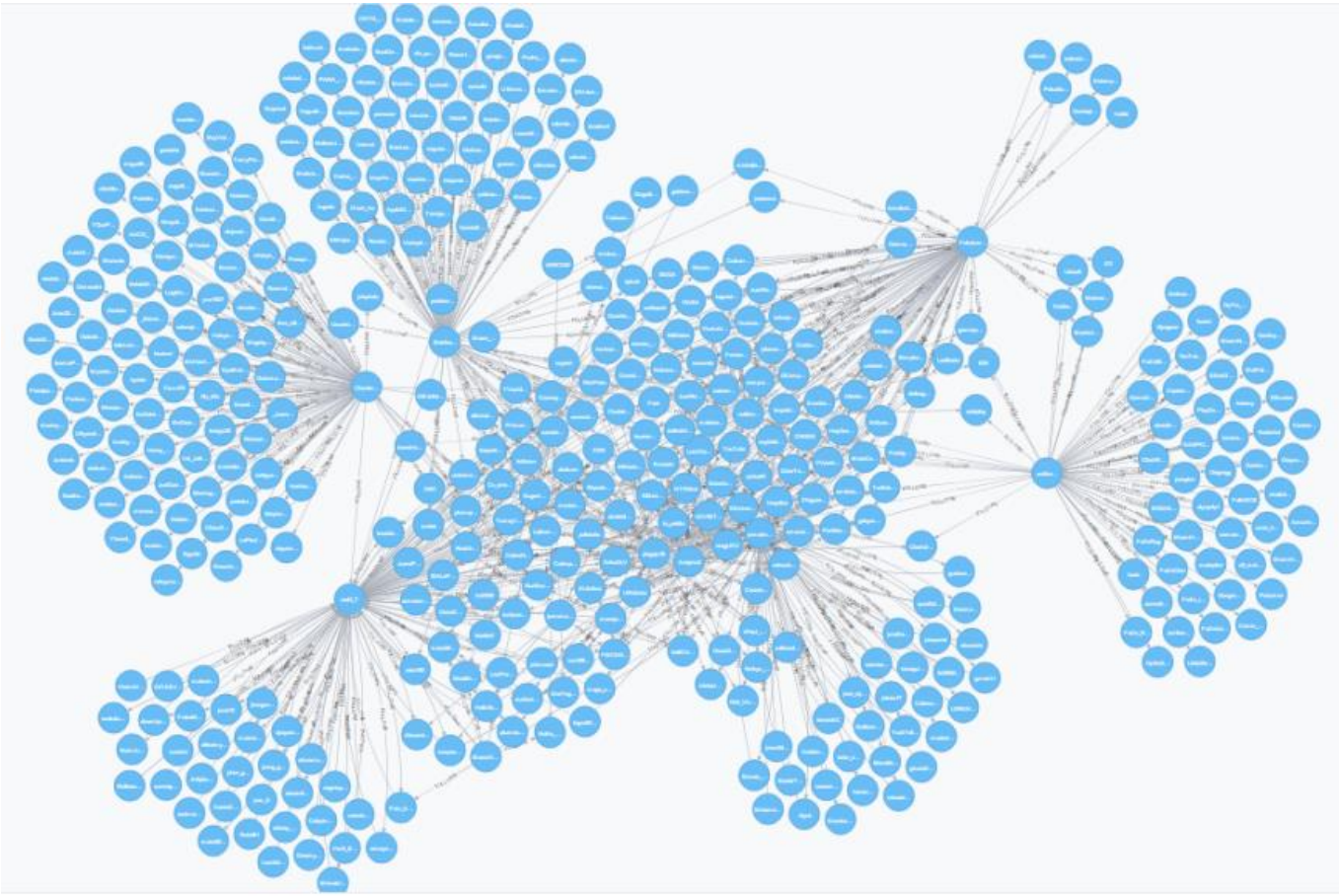
- **Para nivel 0:** Un grafo donde el nodo principal es Carl2_7 con sus relaciones a los nodos amigos y un grafo done los amigos de la semilla base tengan hijos
- **Para nivel 1:** Un grafo donde el nodo principal es Carl2_7 con sus relaciones a los nodos amigos y a los amigos de sus amigos

Resultado de la Ejecución: Se obtuvieron los grafos esperados

- Nivel 0



- Nivel 1



8. Verificar que si el algunos de los cursores es mayor a cero se agreguen de manera correcta en la cola de cuentas pendientes por descargar (*unFinishedTwitter)*

Descripción: Esta prueba consistió en verificar que las *cuentas semillas* se agreguen en la lista de *unFinishedTwitter*, cuya función es almacenar todas las *cuentas semillas* que tienen amigos y seguidores pendientes por descargar

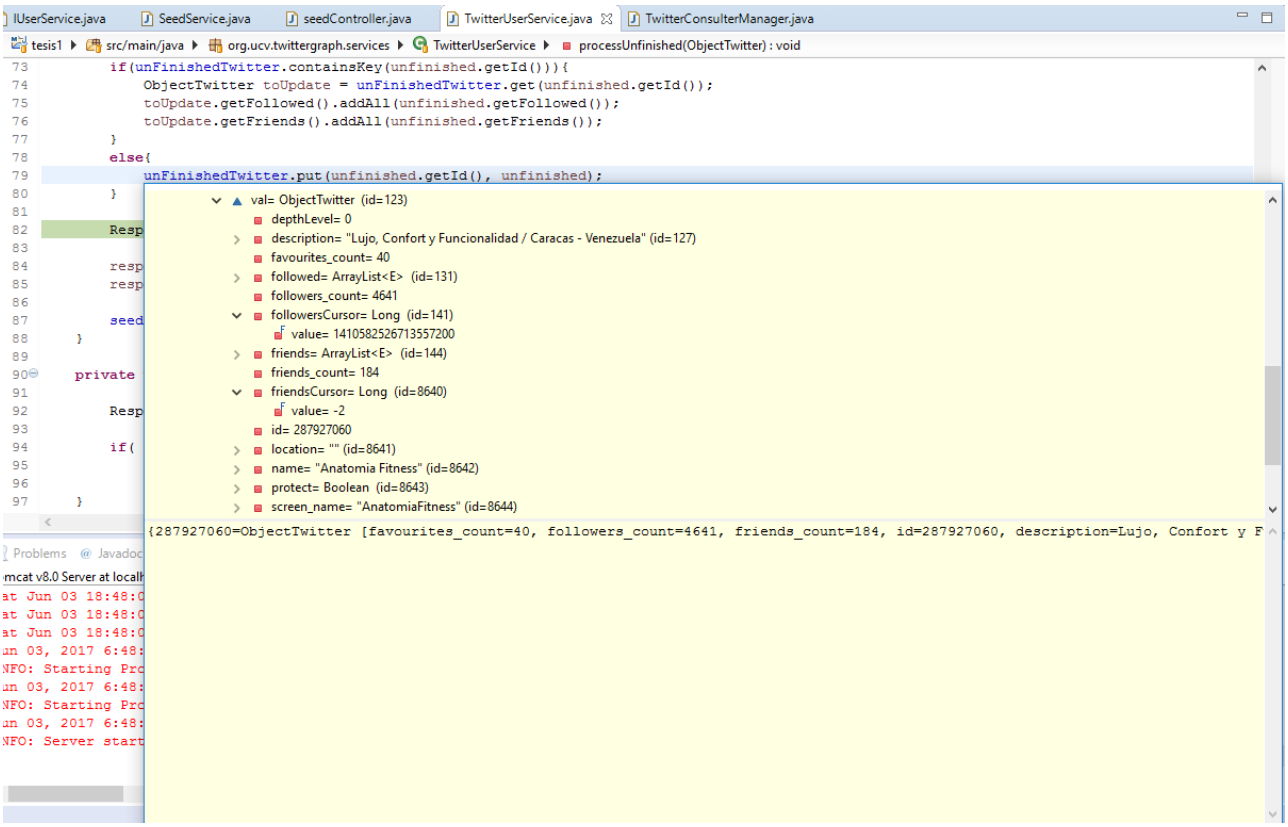
Semilla: AnatomiaFitness

Resultado Esperado: un objeto *semilla* encolado en la lista de unFinishedTwitter.

Resultado: en efecto se agregó la semilla para un próximo procesamiento. Para verificar este resultado se colocó un *break point* en la línea donde se hace el procesamiento y se verificó que en efecto se encontraba la semilla.

Objeto que debe encolar

Objeto encolado



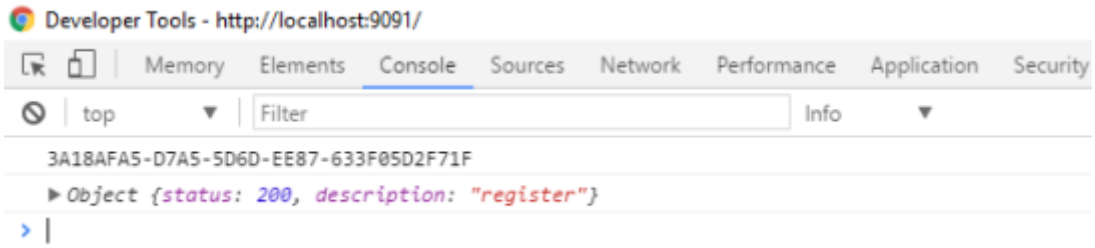
9. Verificar que los clientes sean agregados de manera correcta en la cola clientes disponibles para descargar (*idleConsulters)*

Descripción: Esta prueba consiste en verificar que dado un cliente registrado este se encole de manera correcta en la cola de clientes y así el módulo coordinador sepa que clientes están disponible para enviarle información que deba descargar:

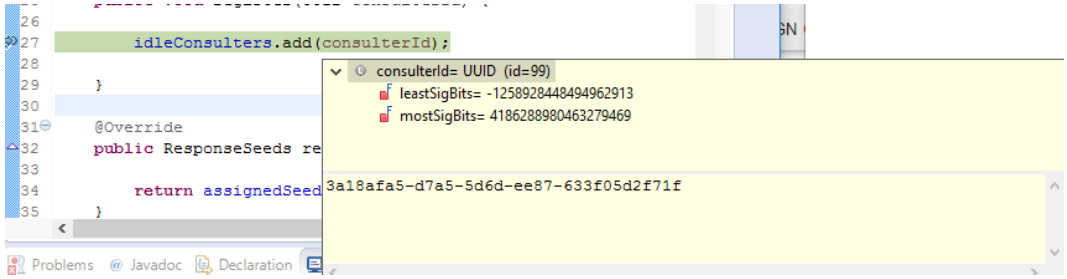
Respuesta esperada: Una cola con el identificador del cliente registrado

Respuesta de la ejecución: En efecto se encolo el identificador de manera correcta

- **Identificador del cliente**



- **Información Almacenada en el coordinador**



10. Verificar que las semillas recibidas por parte del módulo administrador se inserten en la cola cuentas semillas pendientes por descargar (pendingProcess)

Descripción: Esta prueba consistió en verificar que toda cuenta semilla que llegue del sistema administrador se agregue de manera correcta en la cola *pendingProcess*, para que estas puedan ser asignadas a un cliente para su posterior procesamiento.

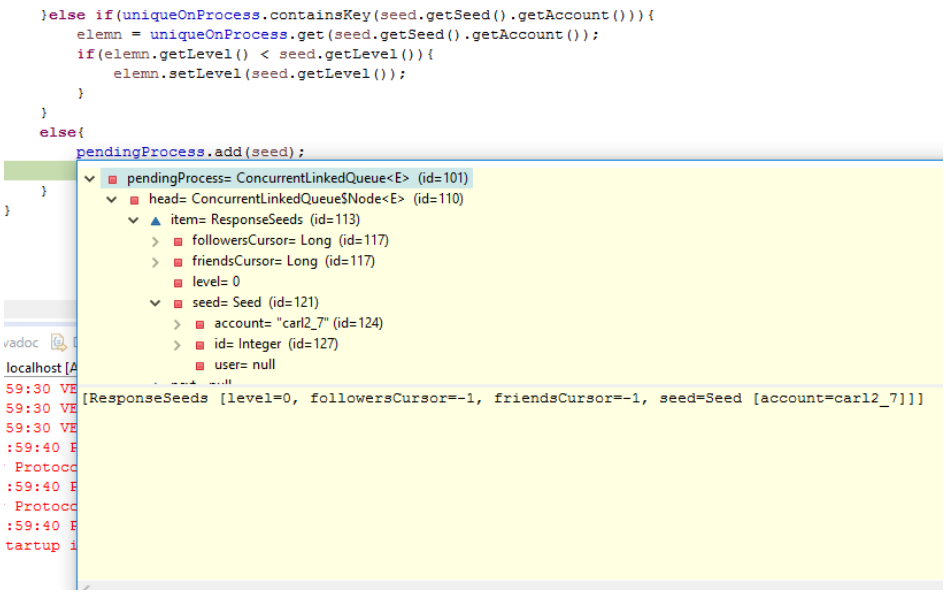
Caso de Prueba: Seleccionar la semilla carl2_7 en el sistema de administración

Resultado esperado: Objeto semilla con atributo account = "carl2_7" encolado en la lista de *pendingProcess*.

Resultado Obtenido: Al chequear la cola de *pendingProcess* en efecto se encontraba la semilla esperada.

- Vista Administrativa

- Recepción de la semilla en el Coordinador



7.1. Retrospectiva de la iteración (Sprint Retrospective):

El equipo en cada reunión semanal analizó cómo era la manera de trabajar y cuáles habían sido los problemas que les impedían progresar adecuadamente. Entre ellos los siguientes desafíos:

- Falta de entendimiento del problema, lo que implicó que en cada reunión con el cliente (Product Owner), surgieran nuevas dudas por parte del equipo y re replanteara la solución.
- Falta de experiencia con el manejo del *framework* y la base de datos Neo4j, lo cual implicó una curva de aprendizaje elevada antes de poder implementar la lógica de negocio que involucraba estas herramientas.
- La mayoría de las iteraciones implicaron un trabajo arduo para el entendimiento de la lógica de negocio a aplicar.
- Mala distribución del tiempo de las horas de trabajo por parte del equipo.
- Al momento de momento de realizar las pruebas finales nos enfrentamos a desafíos en cuanto a los recursos necesarios a nivel de hardware para realizar la inserción de los nodos en la base de datos de grafos Neo4j.

CAPÍTULO 7

CONCLUSIONES

Se logró desarrollar todo lo planteado en el alcance del trabajo, los componentes se interconectan de manera correcta. Se utilizó la metodología de desarrollo Scrum con algunas variantes lo que permitió definir un marco de trabajo. Se seleccionó como caso de estudio la red social Twitter y se enfocó el desarrollo en superar las limitaciones específicas de su API. Se desarrolló un módulo *administrador* que permite crear y eliminar *cuentas semillas* y proporcionar el *nivel de profundidad* al grafo que se desea construir. Se desarrolló una aplicación *cliente* que se conecta con el API de Twitter para descargar los datos. Se implementó un coordinador que maneja una conexión con la base de datos Neo4j, también se conecta con los clientes disponibles para descarga de datos y tiene como tarea principal paralelizar las *cuentas semillas* y los *clientes.* Se realizaron pruebas de software que permitieron garantizar el funcionamiento correcto de la herramienta.

Se obtuvo como resultado la estructura de la red social de Twitter sin duplicación en los datos superando las limitaciones presentadas por el API disminuyendo los tiempos de respuesta como podemos ver en la *tabla 8*. Aunque en principio el proyecto fue pensado para trabajar dos niveles de profundidad (nivel 0 y nivel 1) el sistema está diseñado para construir el grafo para cualquier nivel indicado por el usuario administrador.

Tabla 8: Comparación de tiempos en función de la cantidad de clientes

Número Amigos y seguidores	1 Cliente	2 Clientes	10 Clientes	100 clientes	1000 clientes
3.000	**15 minutos**	7,5 minutos	1,5 minutos	9 segundos	0,9 segundos
50.000	**4.1 horas**	2.08 horas	25 minutos	2,5 minutos	15 segundos
100.000	**8.3 horas**	4.1 horas	50 minutos	5 minutos	30 segundos
1.000.000	**3.4 días**	1.7 días	8.3 horas	50 minutos	5 minutos

El grafo generado permite ser la entrada para cualquier otro sistema que realice los análisis pertinentes asociados a la ciencia de datos.

7.1. Contribución

Este trabajo de investigación contribuye principalmente con la Escuela de Computación de la Facultad de Ciencias de la Universidad Central de Venezuela, en el área computación paralela y distribuida específicamente en la rama Big Data, ya que abre camino a una infinidad de futuras investigaciones donde sirva como entrada el grafo social generado.

7.2. Recomendaciones

Para el uso del sistema es necesario contar con equipo que cuente con los siguientes requerimientos:

- Base de datos MySQL para que pueda funcionar de manera correcta el módulo administrador y se puedan almacenar las *cuentas semillas*
- Servidor web para alojar la herramienta y pueda ser consultada por diversos clientes.
- Base de datos Neo4j, ya que es la base de datos de grafos donde se almacena la estructura de la red social
- Java porque es el lenguaje base del framework Spring en el que fue desarrollada la herramienta.
- Un IDE para facilitar el entendimiento y manejo del código a los desarrolladores que deseen implementar trabajos futuros.

7.3. Trabajos futuros

Sobre el trabajo realizado existen distintas modificaciones que pueden ser hechas para mejorar su utilidad.

7.3.1. Implementar el módulo cliente en el lado del servidor:

Se propone implementar la búsqueda de semillas en un módulo Java, ya que actualmente los clientes son web, manejados en el lado del cliente y esto puede implicar un consumo total de la memoria del computador al ejecutar la aplicación. Este módulo a desarrollar debe ser instalado en cada una de las maquinas que correrán el sistema.

7.3.2. Integración con "Spring for Apache Hadoop" para implementar sobre una plataforma Big Data

Por otro lado al no tener los recursos disponibles, no fue posible montar el sistema sobre una plataforma hadoop, sin embargo esta tarea puede realizarse de manera sencilla haciendo uso del módulo para hadoop que provee el *framework Spring "Spring for Apache Hadoop".*

"Spring for Apache Hadoop" simplifica el desarrollo sobre Apache Hadoop proporcionando un modelo de configuración unificada y *API*s fácil de usar para el uso de HDFS, MapReduce, Pig, y Hive. También proporciona integración con otros proyectos del Ecosistema Spring como Spring Integration y Spring Batch lo que le permite desarrollar soluciones para grandes volúmenes de datos y orquestación de flujo de trabajo en Hadoop.

- **Características**
  - Apoyo a la creación de aplicaciones de Hadoop que se configuran mediante la inyección de dependencias y se ejecuten Como una aplicación Estándar de java vs utilizando Hadoop utilidades de línea de comandos.
  - Integración con Spring boot para simplificar la creaciones de aplicaciones Spring que se conectan a HDFS b para leer y escribir datos.
  - Permite la creación y configuración de aplicaciones que utilizan Java MapReduce, Streaming, hive , pig , o HBase
  - Extensiones a Spring Batch para apoyar la creación de flujos de trabajo basados en Hadoop para cualquier tipo de operación HDFS Hadoop de las operaciones de escritura de HDFS utilizando cualquier lenguaje de programación basado en JVM.

- Cree fácilmente aplicaciones basadas en la *Spring Boot* personalizadas que se pueden implementar para ejecutar en el hilo.
- apoyo DAO (Plantilla y devoluciones de llamada) para HBase.
- Soporte para Hadoop SegurIDad.

- **Versiones y Apoyo a la Distribución**

Spring para Apache Hadoop es compatible con distribuciones comerciales de Pivotal, Hortonworks y Cloudera.

Las distribuciones soportadas varían según la versión de lanzamiento. La integración continúa para las versiones más compatibles.

- **¿Cómo integrar?**

El método recomendado para empezar a utilizar *spring-hadoop* en su proyecto es con un sistema de gestión de la dependencia - el fragmento a continuación se puede copiar y pegar en su construcción. [40]

Figura 12: Dependencias para utilizar *Spring-hadoop*

```
<dependencies>
   <dependency>
      <groupId>org.springframework.data</groupId>
      <artifactId>spring-data-hadoop</artifactId>
      <version>2.4.0.RELEASE</version>
   </dependency>
</dependencies>
```

- **Implementación para el registro de clientes en el coordinador**

```
@RestController
@RequestMapping("/twitterConsulter")
public class TwitterConsulterController {
    ResponsePolling resp = new ResponsePolling();

    @Autowired
    private ITwitterConsulterManager consulterManager;

    @Autowired
    private ITwitterUserService twitterUserService;


    private static final Long  TIMEOUT = 1200001;

    private static final Long WAIT_TIME = 20001;

    @RequestMapping(value = "/{consulterId}" ,method = RequestMethod.GET)
    public ResponsePolling register(@PathVariable(name="consulterId") UUID consulterId) {
        resp.setStatus(200);
        resp.setDescription("register");
        consulterManager.register(consulterId);

        return resp;
    }
```

Figura 13: Implantación para el registro de clientes en el coordinador

- **Implementación del Long Polling**

```
@RequestMapping(value = "/work/{consulterId}", method = RequestMethod.GET)
public DeferredResult<ResponseEntity<ResponseSeeds>> askWork(@PathVariable("consulterId") UUID consulterId) {


    DeferredResult<ResponseEntity<ResponseSeeds>> defResult = new DeferredResult<>(TIMEOUT);

    CompletableFuture.runAsync(() -> {
        ResponseSeeds r;
        while(!defResult.isSetOrExpired()){
            r = consulterManager.requestWork(consulterId);
            if(r != null){

                defResult.setResult(ResponseEntity.ok(r));
            } else
                try {
                    Thread.sleep(WAIT_TIME);
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
        }
    });
    defResult.onTimeout(()->{

        defResult.setResult(ResponseEntity.status(HttpStatus.REQUEST_TIMEOUT).body(null));
    });

    return defResult;
}
```

Figura 14: Implementación del Long Polling

- **Implementación de almacenamiento de cuenta semilla raíz y procesamiento de la lista de amigos (*Friends*) y seguidores (*Followers*)**

```
public TwitterUser processTwitterUser(ObjectTwitter twitterObject) {

    //Find and store First User
    TwitterUser firstUser = findORCreate(twitterObject);

    List<ResponseSeeds> seedsToQueue = new LinkedList<>();

    seedsToQueue.addAll(persistFollowers(firstUser, twitterObject.getFollowed(), twitterObject.getDepthLevel()));
    seedsToQueue.addAll(persistFriends(firstUser, twitterObject.getFriends(), twitterObject.getDepthLevel()));

    seedService.sendSeed(seedsToQueue);

    return firstUser;
}

@Transactional
private List<ResponseSeeds> persistFollowers(TwitterUser root, List<ObjectTwitter> followers, int level){

    TwitterUser follower;

    level--;

    List<ResponseSeeds> followerSeeds = new LinkedList<>();

    for(ObjectTwitter tu : followers){

        follower = findORCreate(tu);

        createRelationship(follower, root);

        if(level >= 0){
            followerSeeds.add(ResponseSeeds.fromSeedAndLevel(tu.toSeed(), level));
        }

    }

    return followerSeeds;
```

Figura 15: Implementación de almacenamiento de cuenta semilla raíz y procesamiento de la lista de amigos (*Friends*) y seguidores (*Followers*)

Bibliografía

[1] Puro Marketing *HTTP*://www.puromarketing.com/16/16626/horas-redes-sociales-estadisticas-sorprendentes.html 11/04/2016 - 10:16am

[2] Power Data, Especialistas en gestión de datos: ¿Qué significa Hadoop en el mundo del Big Data? Un contenido para perfiles técnicos *HTTP*://cdn2.hubspot.net/hub/239039/file-884064500-pdf/docs/PWD_-_BIG_DATA_-_hadoop_-_que_significa_haddop_en_el_mundo_del_big_data.pdf 28/03/2016 – 1:24am

[3] *HTTP*://sg.com.mx/revista/43/definiendo-ciencia-datos#.VrVlXfnhDct 28/02/2016 - 11:28pm

[4] *HTTP*s://www.ibm.com/developerworks/ssa/local/im/que-es-big-data/ 27/03/2016 – 0:43am

[5] *HTTP*s://es.wikipedia.org/wiki/Medios_sociales 09/03/2015 – 10:06 pm

[6] *HTTP*s://es.wikipedia.org/wiki/Interfaz_de_programaci%C3%B3n_de_aplicaciones 04/03/2016 – 2:41am

[7] *HTTP*s://dev.Twitter.com/overview/documentation 04/03/2016 – 2:45am

[8] *HTTP*://www.barriblog.com/2010/07/lo-que-siempre-quiso-saber-del-*API*-de-Twitter-y-nunca-se-atrevio-a-preguntar/ 04/03/2016 – 2:49am

[9] *HTTP*s://en.wikipedia.org/wiki/Facebook_Platform 28/03/2016 - 3:07pm

[10] *HTTP*://code.tutsplus.com/tutorials/introduction-to-the-*Instagram-API*--cms-23608 07/04/2016 - 10:25pm

[11] *HTTP*://www.maestrosdelWEB.com/que-son-las-bases-de-datos/ 10/03/2016 – 9:23pm

[12] *HTTP*://www.acens.com/wp-content/images/2014/02/bbdd-nosql-wp-acens.pdf 10/03/2016 – 9:23pm

[13] *HTTP*://www.WEBtaller.com/manual-java/caracteristicas-java.php 08/03/2016 – 2:12pm

[14] *HTTP*://sheyla88.blogspot.es/ 08/03/2016 – 2:21pm

[15]*HTTP*s://es.wikibooks.org/wiki/Programaci%C3%B3n_en_Java/Caracter%C3%ADsticas_del_lenguaje 08/03/2016 – 2:23pm.

[16] *HTTP*://www.monografias.com/trabajos4/lenguajec/lenguajec.shtml 08/03/2016 – 3:45pm.

[17] *HTTP*://decsai.ugr.es/~jfv/ed1/c/cdrom/cap1/f_cap12.htm 08/03/2016 – 4:05pm.

[18] *HTTP*://www.desarrolloWEB.com/articulos/1325.php 08/03/2016 – 5:13pm.

[19] *HTTP*://www.desarrolloWEB.com/articulos/1325.php 08/03/2016 – 5:13pm.

[20] *HTTP*://www.desarrolloWEB.com/articulos/1325.php 08/03/2016 – 5:13pm.

[21] *HTTP*://ibiblio.org/pub/linux/docs/LuCaS/Manuales-LuCAS/doc-curso-html/doc-curso-html/x38.html 08/03/2016 – 5:13pm.

[22] *HTTP*://edumatica.ing.ula.ve/edumatica/Teleclases/TecniWEB/Ingenieria%20WEB/Teleclase/Ejecucion/Practicas/JavaScript/Paginas/CaracteristicasGenerales.htm 08/03/2016 – 5:25pm.

[23] *HTTP*://www.uazuay.edu.ec/estudios/sistemas/lenguaje_iii/MAnualJavaScript/caracteristicas.htm 08/03/2016 – 5:36pm.

[24] *HTTP*://www.adelat.org/media/docum/nuke_publico/lenguajes_del_lado_servidor_o_cliente.html 08/03/2016 – 5:41pm.

[25] *HTTP*s://es.wikipedia.org/wiki/Node.js 08/03/2016 – 8:00pm.

[26] *HTTP*s://es.wikipedia.org/wiki/AngularJS 08/03/2016 – 8:00pm.

[27] Encuesta de forrester *software* Q4,2013 28/03/2016 00:55am

[28] ¿Qué significa Hadoop en el mundo del Big Data? Un contenido para perfiles técnicos / powerdata (pdf) 28/03/2016 00:55am

[29] *HTTP*s://es.wikipedia.org/wiki/Arquitectura_orientada_a_servicios 09/04/2016 6:00pm

[30]*HTTP*s://www.accenture.com/t20150527T210642__w__/es-es/_acnmedia/Accenture/Conversion-Assets/DotCom/Documents/Local/es-es/PDF_2/Accenture-SOA.pdf 09/04/2016 7:30pm

[31] *HTTP*s://es.wikipedia.org/wiki/Desarrollo_%C3%A1gil_de_*software* 09/04/2016 8:01pm

[32]*HTTP*://blog.leanmonitor.com/es/que-son-las-metodologias-agiles/ 09/04/2016 8:34pm

[33]*HTTP*://www.i2btech.com/blog-i2b/tech-deployment/5-beneficios-de-aplicar-metodologias-agiles-en-el-desarrollo-de-*software*/ 09/04/2016 8:36pm

[34]*HTTP*://issi.dsic.upv.es/archives/f-1069167248521/actas.pdf 09/04/2016 8:54pm

[35] *HTTP*s://es.wikipedia.org/wiki/Kanban_(desarrollo) 09/04/2016 9:32pm

[36] *HTTP*://blog.leanmonitor.com/es/que-son-las-metodologias-agiles/ 09/04/2016 10:57pm

[37] *HTTP*s://es.wikipedia.org/wiki/Programaci%C3%B3n_extrema 09/04/2016 00:30am

[38] *HTTP*://gravitar.biz/bi/base-datos-columnar/

[39] *HTTP*s://es.wikipedia.org/wiki/Tecnolog%C3%ADa_Push , 14/05/2017 10:23pm

[40] *HTTP*://projects.spring.io/spring-hadoop/ 13/05/2017 9:48pm

[41] *HTTP*://www.silicon.es/grandes-volumenes-datos-pequenas-decisiones-negocio-2321872

[42] HTTPs://es.wikipedia.org/wiki/Spring_Framework

[43] HTTPs://proyectosagiles.org/que-es-scrum/ 13/05/2017

[44] HTTPs://www.cnet.com/es/noticias/Twitter-319-millones-usuarios-febrero-2017-q4-2016/

[45] HTTPs://proyectosagiles.org/equipo-team/

[46] https://neo4j.com/

[47] https://es.wikipedia.org/wiki/Teor%C3%ADa_de_grafos

[48] https://es.wikipedia.org/wiki/Grafo_social

[49] http://yusef.es/blog/2011/01/visualizacion-de-redes-sociales-linkedin-inmaps/

[50] http://historiapolitica.com/redhistoria/imagenes/ndos/larrosa4.jpg

[51] https://blogs.deusto.es/bigdata/analisis-de-redes-sociales-el-poder-de-la-teoria-de-grafos/

[52] http://www.crecenegocios.com/fuentes-de-informacion/