UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

División de Ingeniería Eléctrica

Bases de datos NoSQL: solución en sistemas distribuidos y de alto desempeño.

TESIS PROFESIONAL
para obtener el título de
INGENIERO EN COMPUTACIÓN

PRESENTAN
IVÁN MOYA PÉREZ
VÍCTOR JESÚS RIVERA ESTRADA

DIRECTOR DE TESIS
ING. JORGE ALBERTO RODRÍGUEZ CAMPOS

Universidad Nacional
Autónoma de México

UNAM – Dirección General de Bibliotecas
Tesis Digitales
Restricciones de uso

DERECHOS RESERVADOS ©
PROHIBIDA SU REPRODUCCIÓN TOTAL O PARCIAL

Todo el material contenido en esta tesis esta protegido por la Ley Federal del Derecho de Autor (LFDA) de los Estados Unidos Mexicanos (México).

El uso de imágenes, fragmentos de videos, y demás material que sea objeto de protección de los derechos de autor, será exclusivamente para fines educativos e informativos y deberá citar la fuente donde la obtuvo mencionando el autor o autores. Cualquier uso distinto como el lucro, reproducción, edición o modificación, será perseguido y sancionado por el respectivo titular de los Derechos de Autor.

Agradecimientos.

A mis hermanos, Saulo e Isaí, mis padres, Araceli y Vicente, y a mi familia y amigos, sin cuyo amor, enseñanzas y apoyo, no habría sido posible este gran logro.

A mi compañero Víctor Rivera y a mi director de tesis, Jorge Rodriguez, por acompañarme en este arduo pero interesante proceso de investigación y desarrollo.

A Abigail, por todo el amor, el apoyo y la paciencia que he recibido en el tiempo que llevamos juntos.

A la Universidad Nacional Autónoma de Mexico y a la Facultad de Ingeniería, por haber sido parte importante en mi desarrollo académico, profesional y personal.

Iván Moya Pérez.

Gracias a Dios por todo lo que me ha concedido y todas las pruebas que ha puesto en mi camino.

A mi mamá, por ser el faro que nos ha guiado a todos como familia y por ser una gran mujer que desde pequeño me ha enseñado a ser un mejor ser humano día con día. Preguntarme a las 7 de la mañana para los exámenes rindió frutos. Este logro es tuyo, a través de mí espero puedas ver reflejado tu éxito como mamá y como mujer.

A mi papá, por ser quien guío mis pasos y ha fungido como mi mayor ejemplo a lo largo de mi vida. Este logro es tuyo porque siempre me has impulsado a ser mejor y a buscar la superación. Espero poder algún día lograr la mitad de lo que tú lograste con nosotros.

A mi hermana, por ser un ejemplo a seguir y por abrir camino antes que mí. Que este logro nos impulse a seguir adelante y nunca conformarnos.

A Alicia, por impulsarme, por apoyarme y por ser la mujer que ha dado todo por estar conmigo. G M que esto represente el inicio de nuestros planes y que nunca nos soltaremos.

A Iván y al profesor Jorge, gracias por hacer esto posible y fungir como aliados.

A la Universidad Nacional Autónoma de México y a la Facultad de Ingeniería.

Víctor J. Rivera Estrada.

Contenido.

Agradecimientos. 3
Contenido. 5
Introducción. 10
1. Panorama general: sistemas NoSQL. 15
1.1. Breve historia y definición del desarrollo de sistemas NoSQL. 15
1.1.1. La necesidad del uso de sistemas NoSQL. 15
1.2. Requerimientos no funcionales en NoSQL. 16
1.2.1. Sistemas de archivos específicos. 16
1.2.2. Transacciones en NoSQL. 18
1.2.3. Modelos de distribución. 27
1.2.4. Escalabilidad horizontal contra escalabilidad vertical. 29
1.3. Persistencia políglota. 30
1.4. Bases de datos orientadas al esquema clave-valor. 32
1.4.1. Características generales. 32
1.4.2. Principales implementaciones. 33
1.5. Bases de datos orientadas a documentos. 42
1.5.1. Características generales. 42
1.5.2. Principales implementaciones. 42
1.6. Bases de datos orientadas a grafos. 62
1.6.1. Características generales. 62
1.6.2. Principales implementaciones. 63
1.7. Comparativa de los principales productos NoSQL. 67
1.8. Casos de éxito con tecnologías NoSQL. 68
1.8.1. Sistemas de almacenamiento con bases de datos del tipo clave-valor (Cassandra). 69
1.8.2. Sistemas de almacenamiento con bases de datos orientadas a documentos (CouchDB). 71
1.8.3. Sistemas de almacenamiento con bases de datos orientadas a documentos (MongoDB). 72
2. Análisis de la problemática y requerimientos. 75

2.1. Definición del problema........ 75
2.2. Situación actual........ 76
2.2.1. Sistema de Atención de Incidencias: SAI. 76
2.3. Principales requerimientos funcionales........ 78
2.3.1. Administración de incidencias........ 78
2.3.2. Análisis de la información: generación de reportes. 84
2.4. Principales requerimientos no funcionales........ 86
2.4.1. Concurrencia. 86
2.4.2. Volumen........ 87
2.4.3. Transaccionalidad........ 89
2.5. Alcance. 90
3. Análisis y diseño de la arquitectura del sistema........ 95
3.1. Sistemas empresariales multicapa. 95
3.2. Principales frameworks. 97
3.3. Diseño de sistemas orientados a entidades........ 99
3.3.1. Identificación y diseño de las entidades del SAI. 100
3.4. Diseño de sistemas orientadas al desarrollo de interfaces. 113
3.4.1. Diseño de interfaces para el SAI. 113
3.5. Diseño de la capa de acceso a datos. 124
3.5.1. Modelo de datos basado en agregaciones. 125
3.5.2. Proceso de conversión del modelo relacional........ 127
3.5.3. Capacidades del modelo basado en agregaciones. 137
3.5.4. Limitaciones del modelo basado en agregaciones. 139
3.5.5. Diseño del modelo de datos (ER) del sistema empleando el modelo relacional. 141
3.5.6. Diseño del modelo de datos a través de una base de datos orientada a documentos. 146
3.5.7. Comparativa entre los modelos ER y el modelo orientado a documentos. 151
3.6. Selección de frameworks y estrategias de persistencia........ 151
3.6.1. ¿Por qué MongoDB? 152
3.7. Diseño de requerimientos no funcionales. 154

3.7.1. Características principales de la arquitectura de MongoDB. 154
3.7.2. Almacenamiento en MongoDB para el SAI... 156
3.7.3. Transaccionalidad en MongoDB para el SAI. 157
3.7.4. Concurrencia en MongoDB para el SAI. 157
4. Implementación de casos representativos. 166
4.1. Implementación del proceso. 166
4.2. Interacción básica con la base de datos: Operaciones CRUD......................... 166
4.2.1. Preparación de la información. 167
4.2.2. Conexión con el servidor de MongoDB. 173
4.2.3. Creación/Inserción de un documento. 174
4.2.4. Consulta de documentos. 178
4.2.5. Eliminación de documentos 187
4.2.6. Actualización de documentos. 188
4.2.7. Manejo de archivos con GridFS.. 193
4.2.8. Administración de bitácoras dentro del SAI. 199
4.3. Implementación y configuración de la estrategia para el manejo de transacciones. 202
4.4. Implementación de procesos de búsqueda y explotación de la información..... 209
4.4.1. Operaciones Map-Reduce. 210
4.5. Comparativa operacional entre los modelos NoSQL y relacional. 215
5. Pruebas y análisis de resultados.. 220
5.1. Diseño de pruebas unitarias e integrales. 220
5.1.1. Pruebas unitarias. 221
5.1.2. Pruebas integrales. 229
5.2. Diseño de pruebas de volumen, concurrencia y transaccionalidad 245
5.2.1. Prueba de volumen. 245
5.2.2. Prueba de concurrencia. 248
5.2.3. Prueba de transaccionalidad. 253
5.3. Análisis de Resultados 258
6. Conclusiones. 268
7. Bibliografía y mesografía... 277

7.1. Lista de figuras.. 281

Introducción

Introducción.

A lo largo de este trabajo de tesis se busca ofrecer una alternativa al Sistema de Administración de Incidencias (SAI) mediante la exploración de un relativamente nuevo paradigma en las bases de datos: el paradigma NoSQL.

El SAI forma parte de un entorno colaborativo de negocios en el que los problemas detectados en cualquier componente tecnológico de una empresa son reportados, monitoreados y finalmente solucionados o cerrados con el objetivo de llevar un control más eficiente y sencillo desde su administración hasta la interacción con el usuario final.

A lo largo de los seis capítulos que componen este trabajo de tesis se muestra la solución propuesta de desarrollo del SAI, comenzando por toda la información relacionada al paradigma NoSQL y concluyendo por el ciclo de pruebas que deben ejecutarse para asegurar que el SAI funciona cómo debe hacerlo y que su diseño ha sido el correcto.

En el capítulo 1 se documenta toda la información del paradigma NoSQL: dónde nació, quién lo formuló, cuáles son sus tecnologías, entre otros puntos. En este apartado se presentan las diversas herramientas y arquitecturas de los sistemas NoSQL actuales, además de una aproximación comparativa del modelo NoSQL con el modelo relacional de bases de datos y que resumen todas las características necesarias para contar con la base teórica que permitirá realizar un diseño e implementación óptima del SAI.

En el capítulo 2 se desarrolla el análisis de la situación actual en la cual el SAI debe ser implementado para solucionar un problema específico. En este apartado se define lo que será el SAI desde un punto de vista funcional así como también se analizan las principales características con las que deberá de contar.

En un ciclo de desarrollo de software, el capítulo 2 funge como la etapa inicial del ciclo. En él se analiza cualquier aspecto no técnico con el que SAI debe de interactuar en la operación del día a día y en él se define qué funciones debe cumplir el sistema.

El capítulo 3 de este trabajo se desarrolla el diseño que el SAI tendrá para poder cumplir con cada una de las características necesarias descritas en el capítulo anterior y con las cuales se define el flujo de su funcionamiento. En este capítulo se realiza el modelado de las entidades que formarán parte del SAI, así como las relaciones entre las mismas y se determinan las tecnologías empleadas en su desarrollo de acuerdo a los análisis realizados previamente para cubrir los requerimientos.

El desarrollo del capítulo 3 está basado en los casos de uso más representativos del SAI y en los cuales se observa un beneficio real del empleo de las bases de datos NoSQL así como los actores principales del sistema.

La implementación del SAI se encuentra detallada en el capítulo 4, en él se muestran ejemplos de código de la programación del sistema y se explica uno de las principales características del paradigma NoSQL: la agregación de entidades.

El capítulo 4 contiene el mayor tratamiento técnico de este trabajo, debido a que en él se muestra cómo se implementan los frameworks elegidos en el capítulo previo y la forma en la que se incorpora la tecnología necesaria para que el SAI pueda desempeñar cada una de sus funciones.

En este capítulo se muestran con ejemplos técnicos, código y configuraciones, que ilustran la forma en la que el SAI incorpora tecnología permitiendo que en próximos módulos como Business Intelligence (Inteligencia de Negocio en su traducción al español) y/o Big Data sean añadidos a la arquitectura actual de una forma transparente y sin mayores implicaciones técnicas.

Finalmente en el capítulo 5 se diseñan y ejecutan una serie de casos de prueba para verificar que los procesos que componen el funcionamiento del SAI desempeñan sus tareas de forma adecuada.

En el último capítulo de este trabajo de tesis se busca asegurar que el funcionamiento del SAI sea el esperado y reducir en la medida de lo posible, algún tipo de vicio o resultado poco coherente del sistema.

Es importante incluir en este trabajo, las conclusiones a las que se llegará con la implementación de un sistema diseñado con el modelo tradicional Entidad-Relación a un paradigma en constante crecimiento en nuestra área profesional como lo es NoSQL.

Introducción.

CAPÍTULO 1

Panorama general: sistemas NoSQL

1. Panorama general: sistemas NoSQL.

A lo largo de este trabajo los términos en inglés propios de las herramientas y conceptos teóricos, se mostrarán sólo en su primera mención con su significado en español pero la adopción generalizada será en su idioma original.

1.1. Breve historia y definición del desarrollo de sistemas NoSQL.

El término NoSQL fue desarrollado en 1990 por Carlo Strozzi pero su uso, estudio e implementación comenzaron a ejecutarse a partir del año 2009. El término NoSQL se puede interpretar como Not Only SQL (No Sólo SQL, en español) (Jayathilake, Sooriaarachchi, Gunawardena, Kulasuriya, & Dayaratne, 2012) (Vijaykumar & Saravanakumar, 2010) (Hang, E, Le, & Du, 2011).

1.1.1. *La necesidad del uso de sistemas NoSQL.*

El paradigma de las bases de datos NoSQL surgió por la necesidad de contar con bases de datos que almacenen grandes volúmenes de datos con un acceso más rápido que el de las bases de datos relacionales, basadas en la web, distribuidas y con escalabilidad horizontal (término que puede ser conceptualizado como el crecimiento físico de servidores de un entorno distribuido y referido en la sección 1.2.4 de este trabajo).

Saravanakumar y Vijakumar (2010) señalan en la Tabla 1-1 las equivalencias de algunos términos usados al inicio del desarrollo del paradigma de NoSQL con los términos usados actualmente.

Nombre tradicionalmente usado	Nuevo nombre adoptado
Archivo Hash	Almacén Clave-Valor
Archivo Jerárquico (HSAM,HDAM)	BigTable
Nodo Padre	Familia de Columna
Autonomía Local (los datos se almacenan donde son más consultados)	Tolerancia a la Partición (capacidad de distribuir el almacenamiento de una base de datos)
Partición Horizontal	Fragmentación
Propiedades (Atomic, Consistent, Isolated y Durable)	Propieades BASE (Basic Availability, Soft State y Eventually Consistent)

Tabla 1-1: Equivalencia de términos usados en NoSQL

1.2. Requerimientos no funcionales en NoSQL.

1.2.1. *Sistemas de archivos específicos.*

Algunos tipos de bases de datos NoSQL implementan un tipo de atributo específico para el manejo de archivos de datos de gran tamaño, lo que es conocido como sistemas de archivos, término que no es asociado de ninguna forma a la administración de archivos realizada por algún sistema operativo.

El funcionamiento de los sistemas de archivos se basa en tomar un archivo de gran tamaño (puede ir de GB a TB de información o datos) y, en la mayoría de los

casos, dividirlo en pequeños segmentos para su almacenamiento de forma independiente.

Algunos tipos de sistemas de archivos son el Sistema Distribuido de Archivos de Hadoop (HDFS por sus siglas en inglés), el cual es un sistema de archivos propietario de la herramienta Apache Hadoop que se orienta en la implementación de sistemas distribuidos.

Si bien HDFS no es un sistema de archivos propietario de alguna base de datos orientada a objetos, sí es compatible al menos con Apache Cassandra, base de datos que cuenta con su propio sistema de archivos llamado Sistema de Archivos de Cassandra (CFS por sus siglas en inglés) y que funge como puente entre Hadoop y Cassandra con la finalidad de obtener una integración completa de varias tecnologías.

Por su parte, MongoDB cuenta con el sistema de archivos llamado GridFS el cual le permite almacenar archivos de gran tamaño de una manera eficiente y cuya funcionalidad, características ventajas, se encuentran descritas en capítulos posteriores de este trabajo.

Así como existen sistemas de archivos especializados en el almacenamiento de grandes volúmenes de información, CouchDB no hace uso de alguno en particular, integrando el almacenamiento de archivos muy grandes dentro de su configuración y características.

Algunas implementaciones de bases de datos NoSQL complementan el uso de sistemas de archivos con la integración de herramientas de indexación de texto para las funcionalidades de búsqueda de información.

Por último, cabe mencionar que este tipo de sistemas de archivos se encuentran relacionados directamente con la posible integración de las tecnologías en bases de datos NoSQL con el nuevo paradigma de información llamado BigData.

1.2.2. *Transacciones en NoSQL.*

"Recuerda que el principal objetivo de las transacciones es asegurar la integridad de los datos guardados con el fin de mantener la consistencia de los mismos si se hacen consultas recurrentes" (Camacho, 2010).

Una transacción es definida como la ejecución de una operación (selección, actualización, inserción o borrado de datos) sobre la estructura de los datos, la cual puede ser modificada para que cierto grupo de operaciones puedan ser consideradas como una sola, es decir, todo un proceso de lectura y modificación de un dato puede ser considerado como una sola transacción.

El control de transacciones en los sistemas relacionales y los sistemas NoSQL suelen ser administrados de distintas formas: en los sistemas relacionales, el ambiente de ejecución y configuración es quien generalmente garantiza el correcto funcionamiento de la transaccionalidad en el sistema mediante una serie de propiedades; mientras que en los sistemas basados en NoSQL, el desarrollador de la aplicación que usa la base de datos o el administrador de la misma, es quien debe

asegurarse que la transaccionalidad funcione de manera adecuada a las propias necesidades.

1.2.2.1. *Impacto de las propiedades ACID en sistemas NoSQL.*

El modelo de datos empleado por las bases de datos tradicionales (relacionales) para asegurar el control de transacciones y la integridad de los datos, hace uso de las siguientes propiedades conocidas como ACID (Atomic, Consistent, Isolated y Durable):

- *Atomic (Atómico o atomicidad).* En una base de datos, se le denomina transacción al conjunto de operaciones que se ejecutan como unidad, la cual es indivisible y se trata como una sola. Una transacción es exitosa si cada uno de sus elementos es exitoso, de lo contrario cualquier cambio realizado es deshecho y se regresa al último estado exitoso de los datos.

- *Consistent (Consistente o Consistencia)*. Una transacción al completarse debe asegurar el estado consistente de la base de datos.

- *Isolated (Aíslado)*. Cada transacción es ejecutada individualmente y no deben afectarse entre ellas ni afectar el resultado de cada una de ellas.

- *Durable*. Los resultados de las transacciones concluidas permanecen aún después de casos de falla.

Gran parte de los diferentes modelos de datos usados por las tecnologías basadas en NoSQL, se encuentran orientados a esquemas distribuidos, por lo que las propiedades ACID no suelen ser asociadas a ellos debido que un estricto control de la

integridad de datos y de las transacciones resulta complejo y con un precio considerablemente alto debido a la cantidad de operaciones que se deben realizar para una sola transacción.

Mientras que los sistemas que implementan las propiedades ACID se enfocan en la alta integridad de los datos, los sistemas NoSQL utilizan otro tipo de propiedades que toman en consideración un conjunto distinto de restricciones.

Las propiedades que los modelos de datos del paradigma NoSQL utilizan son conocidas como propiedades BASE.

1.2.2.2. *Propiedades BASE.*

Los modelos de datos implementados por las bases de datos NoSQL suelen ser llamados como orientados a agregaciones, es decir, que la información contenida en ellos no se almacenan en tablas relacionadas unas con las otras en las cuales, para recabar toda la información relacionada con un solo dato, se tienen que realizar varias operaciones, conocidas como operaciones algebraicas en bases de datos, sobre los datos, pudiendo ser lecturas simples (select) hasta operaciones de asociación mediante atributos entre distintas tablas relacionadas (join).

Lo anterior flexibiliza las operaciones a realizar sobre los datos en una base de datos NoSQL y permite priorizar de manera diferente las características que se esperan en la operación de la base de datos, por lo que en una base de datos NoSQL es más importante un mejor desempeño a un alto control en las transacciones.

Bajo las premisas anteriores, las propiedades BASE son empleadas en NoSQL para el aseguramiento de distintas características del sistema, como lo enuncia el significado de sus siglas:

- *Basic Availability* (Disponibilidad Básica en su traducción al español).El sistema está disponible todo el tiempo a pesar de la existencia de múltiples fallas, haciendo uso de un enfoque de alta distributividad. En lugar de mantener una sola unidad de almacenamiento enfocada a la tolerancia de fallos, NoSQL propaga los datos a sistemas con un alto nivel de replicación. En caso de evento de falla en algún segmento de datos, esto no implicada una falla generalizada.

- *Soft-State* (sin traducción significativa al español). Indica que la base de datos no tiene que ser consistente *todo* el tiempo, sólo lo tiene que ser eventualmente en un estado perfectamente conocido (consistencia eventual). La consistencia será manejada por el desarrollador de la aplicación, no por la base de datos.

- *Eventual consistency* (Consistencia Eventual en su traducción al español). El único requerimiento es que la base de datos sea consistente en algún punto del futuro donde la consistencia convergerá. En este escenario las operaciones de lectura pueden ver escrituras, lo que puede causar lecturas sucias que en el modelo relacional son eliminadas a través del nivel de aislamiento de lecturas confirmadas. En bases de

datos replicadas un cambio se verá reflejado en todos los nodos tiempo después a la ejecución del cambio en otro nodo.

1.2.2.3. *El teorema de CAP.*

El teorema de CAP comenzó a ser formulado por Eric Brewer en el año 2000, estuvo orientado hacia servicios web y se planteaba como una compensación entre consistencia, disponibilidad y tolerancia a la partición (Consistency, Availability y Partitioning tolerance respectivamente por sus términos en inglés), por lo que fue formulado de la siguiente manera:

En una red de trabajo susceptible a fallos, es imposible para cualquier servicio web implementar memoria compartida de lectura/escritura atómica que garantice respuesta a cada solicitud. (Gilbert & Lynch, 2011)

Las propiedades mencionadas se describen de la siguiente manera:

- *Consistencia.* Este término corresponde a que si varios nodos de un clúster realizan una lectura a la misma información, obtendrán la misma respuesta teniendo una misma y única vista de los datos, es decir, corresponde con el concepto empleado en el modelo relacional.
- *Disponibilidad.* Para efectos del teorema inicial de CAP, la disponibilidad se resume a que cada solicitud obtenga respuesta de alguno de los nodos del clúster, habilitando de esta manera que un nodo en el clúster pueda ser accedido para leer y escribir datos y este, a su vez, pueda acceder a otro para leer y propagar los cambios en los integrantes del clúster.

- *Tolerancia a la partición.* Supóngase un sistema distribuido que reside en varios arreglos de servidores, la tolerancia a la partición se trata precisamente de poder distribuir en grupos al elemento que se encarga de otorgar las respuestas a cada solicitud y que no necesariamente tienen comunicación entre sí.

Gilbert y Lynch (2011) señalan que el teorema de CAP se interpreta como:

Es imposible para un sistema distribuido ser simultáneamente: Consistente, Disponible y Tolerante a la partición. En cualquier momento, sólo dos de estas tres propiedades deseables pueden ser alcanzadas.

El teorema de CAP debe tomarse en consideración cuando el ambiente distribuido a implementar se planea en una red que tiende a tener intermitencias en la comunicación de datos, por lo que, entre mayor confiabilidad se tenga en la red, menor es la aplicación de este teorema.

Considerando lo anterior y tomando en cuenta que uno de los objetivos de las bases de datos NoSQL es la escalabilidad horizontal, se necesita una red con tolerancia a la partición, por lo que se tiene que flexibilizar la consistencia o la disponibilidad.

En un sistema distribuido se pueden presentar los siguientes escenarios, en los cuales se justifica el planteamiento del teorema de CAP:

- Consistencia-Tolerante a la partición: el sistema ejecuta las operaciones de forma consistente aunque pierda comunicación entre los nodos, sin asegurar la respuesta.

- Disponibilidad-Tolerante a la partición: el sistema responde a las peticiones aún si se pierde la comunicación entre los nodos, pero no puede garantizar la consistencia.

- Consistencia-Disponibilidad: el sistema responde a las peticiones y la consistencia de los datos puede ser asegurada pero no se permiten ningún tipo de falla en la comunicación de los nodos del clúster.

1.2.2.4. *Propiedades CAE.*

Las propiedades CAE hacen referencia a tres características de las bases de datos NoSQL, las cuales son: Cost-Efficiency (Costo-eficiencia en su traducción al español), High-Availability (Alta Disponibilidad en su traducción al español) y Elasticity (Elasticidad en su traducción al español) y que provienen de los principios básicos del cómputo distribuido.

Silberschatz, Korth, & Sudarshan (2006) mencionan que para las bases de datos centralizadas, el costo-eficiencia de una operación se mide en la cantidad de accesos discos que esta requiere, mientras que para los sistemas distribuidos, adicionalmente a lo anterior, se toma en cuenta el costo de la transmisión de datos por la red y la posible ganancia en rendimiento al ejecutar un procesamiento en paralelo.

Por lo anterior, el término *Costo-eficiencia* en bases de datos NoSQL es interpretado como aquella relación existente entre la implementación de una característica o una operación en la base de datos y el costo (económico o en términos de esfuerzo) que conlleva.

El término puede ser ejemplificado con varias características que NoSQL puede implementar: escalabilidad, consultas y almacenamiento de grandes volúmenes de información.

Si es comparada con una base de datos relacional, la base de datos NoSQL es escalable a un costo (económico y/u operativo) menor debido a que es menos complejo su procesamiento y no requiere tiempos fuera de operación; de la misma forma es la ejecución de las consultas, si una base de datos se compone de muchas relaciones entre sus entidades, en el modelo relacional se requiere mayor complejidad en la recolección de datos (haciendo joins) lo cual se traduce en el uso de hardware de mayores capacidades técnicas, mientras que en NoSQL se puede recabar toda la información ligada a una búsqueda con una sola consulta simple con una demanda de recursos de hardware menor.

Así también Silberschatz, Korth, & Sudarshan (2006) definen alta disponibilidad en sistemas de bases de datos distribuidas como la característica que deben tener estas últimas para seguir seguir funcionando aunque el ambiente sufra distintos tipos de fallos.

De esta forma, la *Alta Disponibilidad,* tanto en bases de datos NoSQL como en cualquier otro tipo de sistemas, se refiere a que el sistema NoSQL debe ser diseñado

para ejecutarse sin ningún tipo de interrupción en su servicio bajo un enfoque de grandes volúmenes de solicitudes de lectura/escritura en un ambiente distribuido.

Regularmente la alta disponibilidad de un sistema suele ser medida por SLAs (Service Level Agreements por sus siglas en inglés y Acuerdo de Niveles de Servicio en su traducción al español) en una notación de “nueves”, p.e. un sistema que tiene tres nueves en su disponibilidad, permanece disponible 99.9% del tiempo medido.

Esta propiedad suele estar determinada directamente por la escalabilidad y replicación del sistema, conceptos detallados posteriormente en este trabajo.

Según (Dory, Mejías, & Van Roy) en su trabajo “Comparative elasticity and scalability measurements of cloud databases”, la *Elasticidad* se refiere a la capacidad que posee un sistema distribuido a adaptarse cuando se agregan nuevos nodos al arreglo o cuando se eliminan nodos.

Suele ser medida por el tiempo que le toma al clúster regresar a un estado estable, el cual se caracteriza por que no hay ningún tipo de movimiento de datos entre los nodos y estos, a su vez, se encuentran disponibles y atendiendo solicitudes.

La elasticidad también puede ser interpretada como la facilidad de adaptación que tienen las bases de datos NoSQL en su modelo de datos.

En las bases de datos relacionales se tiene que definir primero el esquema para poder ingresar los datos requeridos y esa estructura es rígida en caso que se requiera agregar un nuevo tipo de información, por lo que el diseño de la bases de datos se tiene que reajustar desde sus fundamentos; mientras que las bases de datos NoSQL

no implementan un esquema fijo y agregar nuevos tipos de datos y/o agregaciones es completamente viable sin necesidad de hacer algún cambio mayor en las mismas.

1.2.3. *Modelos de distribución.*

Una de las principales características de las bases de datos NoSQL es la de ejecutarse fácilmente en un clúster, lo cual habilita a la base de datos para manejar grandes volúmenes de datos y, a su vez, condiciona el que la base de datos pueda manejar alguna falla ajena a su funcionamiento interno, como puede ser una interrupción en el servicio de red de datos o una falla generalizada de software.

Existen dos modelos para distribuir una base de datos NoSQL: la replicación y el sharding (sin traducción al español).

La *replicación* realiza copias de los datos contenidos en la base de datos y las distribuye a través de los nodos del clúster, para lo cual puede emplear dos técnicas: maestro-esclavo o punto-punto.

a) Maestro-esclavo. Un nodo de clúster es nombrado como maestro y se encarga del control de las actualizaciones de los datos en todos los nodos. Es la técnica más recomendada si la lectura es la operación más ejecutada sobre los datos.

   En caso que el nodo maestro falle, los nodos esclavos o secundarios son capaces de seguir atendiendo solicitudes sólo de lectura debido a que el nodo maestro administra estas últimas operaciones.

Se puede implementar un nodo específico para replicar la configuración del nodo maestro para que, en caso de falla, la recuperación de la funcionalidad del sistema sea más rápida.

b) Punto-punto. En esta técnica todos los nodos del clúster cuentan con la misma información de configuración del mismo, así si un nodo deja de funcionar, los demás pueden continuar con la atención a las solicitudes.

   Con esta técnica se pueden añadir fácilmente nodos para mejorar el desempeño, además que es una técnica que mejora la atención en solicitudes de escritura pero que, a su vez, sensibiliza la consistencia de los datos.

El *sharding* divide una base de datos en pedazos y la distribuye en todos los nodos del clúster, lo cual facilita y se adapta de forma transparente al escalamiento horizontal.

La mayoría de las bases de datos NoSQL implementan esta técnica de forma natural, mientras que las bases de datos relacionales requieren de tiempo fuera de operación para reconfigurarse y poder dividirla en múltiples partes.

La replicación y el sharding son técnicas que pueden operar en conjunto, en donde se considera que cada pedazo o parte de la base de datos es su propio maestro que administra las solicitudes enviadas a sus datos contenidos.

1.2.4. *Escalabilidad horizontal contra escalabilidad vertical.*

Las bases de datos NoSQL desde su concepción, fueron pensadas y diseñadas para ser distribuidas y escalables de una forma sencilla y sin complicaciones.

La escalabilidad de una aplicación y/o base de datos comienza a ser necesaria cuando se requiere mayor capacidad de respuesta a solicitudes de usuarios o a un número mayor de éstos, cuando se requiere una administración de más volumen de datos, mayor disponibilidad, entre otros escenarios.

Existen dos posibles vías para cuando una aplicación y/o base de datos llega a alguno de los escenarios ejemplificados anteriormente: escalamiento vertical o escalonamiento horizontal.

El escalonamiento vertical es aquel que se realiza cuando se agregan más componentes físicos a un servidor, por ejemplo, más memoria, más CPUs y más discos (o cualquier otro método) de almacenamiento. Un servidor o servidores demasiado robustos y con capacidades muy altas, conllevan una administración compleja y con software adicional y de mejores prestaciones, por lo que ésta alternativa suele ser costosa.

Este tipo de escalonamiento se caracteriza por su centralización y por su poca tolerancia a fallos dentro de la arquitectura física de un sistema, es decir, si el servidor físico donde reside la aplicación y/o base de datos presenta un error o falla, afecta por completo al sistema, reduciendo las posibilidades de recuperación del

mismo; además, usualmente, implica cambios en el código fuente de las aplicaciones o en las operaciones en las bases de datos.

El escalonamiento horizontal es aquel que se lleva a cabo agregando servidores físicos conforme las necesidades de la aplicación y/o base de datos, el conjunto de servidores físicos o virtuales que pertenecen y colaboran para el funcionamiento del software con esa necesidad específica, se denomina clúster.

El escalonamiento horizontal implica una administración más fina por el número de servidores que puede alcanzar, pero presenta ventajas tales como que las aplicaciones demandan sus recursos como si se tratara de una sola unidad por la forma en que opera un clúster; de la misma forma, no hay que realiza algún tipo de modificación a la programación de la aplicación o a la base de datos. Adicionalmente en esta forma de escalonamiento la tolerancia a fallos es mayor debido a que la ejecución de tareas, así como la información, no residen en un solo servidor o equipo.

Las bases de datos NoSQL fueron diseñadas para operar bajo el esquema de escalanomiento horizontal por la facilidad de configuración, administración y operación, además de la característica que poseen de operar grandes volúmenes de datos.

1.3. Persistencia políglota.

En la actualidad, adaptar un sistema con una base de datos relacional a una base de datos NoSQL, no solo implicaría un trabajo de diseño, también debe tomarse en cuenta la implementación, la arquitectura y las capacidades del sistema. Es importante mencionar que cada sistema está enfocado a resolver un problema

diferente, por lo que muchas veces se opta por tener múltiples soluciones correspondientes. Este grupo de soluciones puede no adecuarse a un problema en específico, ya que cada aplicación y/o sistema puede manejar la información de distinta manera.

Es así como se usa la Persistencia Poliglota, esta se refiere al uso de distintos almacenes y modelos de datos, ya que debido a las características y necesidades de cada herramienta son útiles solo para un determinado problema, de ahí que se busque la forma de integrar distintas herramientas y resolver cada problema de manera óptima, este término engloba en sí, el uso de sistemas híbridos (bases de datos relacionales con bases de datos NoSQL.

Como se mencionó con anterioridad, las bases de datos NoSQL no son un sustituto para las bases de datos relacionales, sino que toman un problema desde otra perspectiva, mientras que un sistema NoSQL puede almacenar grandes cantidades de información, carece de funcionalidades (p.e. la administración de la transaccionalidad y algoritmos de bloqueo) que sí se implementa un sistema relacional, es así como se pueden utilizar ambos para un mejor manejo de las aplicaciones.

Las bases de datos relacionales tienen la ventaja de que pueden funcionar como sistemas analíticos (OLAP), transaccionales (OLTP) o híbridos (OLAP y OLTP), esto debido al nivel de relacionamiento de la información, mientras que los sistemas NoSQL no implementan un nivel de transaccionalidad, y de hacerlo es muy básico, el enfoque de estos, va más orientado a la problemática del tamaño de información que se almacena y no como lo hace, ni como se relaciona entre sí.

La complejidad de la aplicación que implemente una persistencia poliglota, dependerá entonces de las herramientas que se utilicen, es decir, comenzando por decidir que herramienta resolverá cada problema, como manejar los datos, que seguridad implementará y cómo se integrará con los demás sistemas, ya que aunque no sean del todo compatibles las soluciones NoSQL con las relacionales, estas deben interactuar de alguna u otra forma, si lo que se quiere es tener una aplicación consistente y confiable.

1.4. Bases de datos orientadas al esquema clave-valor.

1.4.1. *Características generales.*

Las bases de datos orientadas al esquema clave-valor, también conocidas como almacenes clave-valor, surgen cuando las aplicaciones, muchas de ellas disponibles a través de internet, distribuidas global o localmente comienzan a demandar una alta disponibilidad (24 horas / 7 días a la semana) pero se enfrentan a las limitantes descritas en el teorema de CAP, específicamente con la partición, por lo que las bases de datos tradicionales fueron restadas en su utilidad debido a que dan la más alta prioridad al aseguramiento de la consistencia en los datos, con una alta disponibilidad distribuida aceptable pero con tiempos de respuesta deficientes y con una administración compleja, además que ven limitado el manejo de grandes volúmenes de datos. . (Anderson, Li, Tucek, & Wylie)

1.4.2. *Principales implementaciones.*

1.4.2.1. *Cassandra.*

1.4.2.1.1. *Modelo de datos.*

El modelo de datos de Cassandra está enfocado en atender las necesidades específicas de las consultas a realizarse sobre los datos y está orientado a las columnas, es decir, que mientras en un modelo relacional de bases de datos, éste se encuentra enfocado en reducir entradas redundantes mediante su almacenamiento en tablas con datos normalizados, en el modelo de datos de bases de datos como Cassandra, no es necesario que todas las entradas tengan el mismo número de columnas y no hace uso de relaciones entre las mismas, suprimiendo el uso de llaves foráneas, por lo que las operaciones de unión (tradicionales y básicas en las bases de datos relacionales) al momento de la ejecución de consultas no pueden ser implementadas. Cada una de las entradas, debe tener una columna que cumpla con la función de identificador único de la entrada de datos (fila de columnas).

El lugar de almacenamiento de los datos en Cassandra se denomina keyspace (a lo largo de este trabajo se emplea el término en inglés debido a que no existe una traducción al español fiel al concepto de la palabra), el cual contiene todas las familias de columnas.

Como se menciona en párrafos anteriores, el modelo de datos de Cassandra utiliza como base, de manera análoga a las tablas en el modelo relacional de bases de

datos, las familias de columnas que son las que definen los metadatos de las columnas. Los dos tipos de familias de columnas son las siguientes:

- Familia de columnas estáticas: tipo de columnas básico de Cassandra y muy similares a las tablas de una base de datos relacional porque usan un conjuntos de columnas con estructura similar.
- Familia de columnas dinámicas: tipo de columnas definidas de acuerdo a las necesidades propias para el almacenamiento de datos lo cual convierte la recuperación de datos más eficiente. En esta familia, no se definen los metadatos para una columna en particular sino que los nombres y valores de las mismas son definidos por las aplicaciones al momento de su inserción. (Apache Software Foundation).

A su vez, las familias son formadas por los siguientes tipos de columnas:

- Estándar: Tiene una llave primaria
- Compuesta: Tiene más de una llave primaria.
- Expiradas: Aquellas que se borran durante la compactación.
- Contador: Aquellas usadas para contar las ocurrencias de un evento.
- Súper: Usadas para administrar filas muy grandes. (Apache Software Foundation).

El modelo de datos de Cassandra, está enfocado a ser explotado por las aplicaciones que lo usan, por lo que el diseño del modelado de datos debe comenzar partiendo de los resultados esperados cuando se ejecuten las consultas deseadas y cómo se desea acceder a los datos; así mismo al cambiar el paradigma de las bases

de datos relacionales, Cassandra se desempeña mejor cuando la mayor cantidad de datos puedan ser recuperados en una misma ejecución de una consulta por lo que al diseñar el modelo de datos a usarse, la normalización de los datos no es necesaria. (Apache Software Foundation).

1.4.2.1.2. *Consistencia de datos.*

La consistencia de datos en Cassandra se basa en cómo se encuentra actualizada y sincronizada una fila de datos en cada una de sus réplicas. En Cassandra se denomina como consistencia ajustable cuyo nivel puede ser establecido en cada operación de lectura o escritura.

El nivel de consistencia en las operaciones de escritura específica el número de escrituras exitosas en los nodos réplica que tienen que ser exitosas antes de regresar un estado de cumplimiento a la aplicación.

Cassandra cuenta con distintos niveles de consistencia, partiendo desde que sólo se escriba en un solo nodo (ANY) o que necesariamente se escriba en todos (ALL).

La consistencia de los datos en operaciones de lectura en Cassandra se basa en el número de nodos réplica que deben contestar a la petición antes de regresar algún resultado a la aplicación cliente.

En las consistencia de lectura existen distintos niveles siendo ONE (UNO) el de menor consistencia pero mayor disponibilidad y ALL (TODOS) el de mayor consistencia pero menor disponibilidad.

Como se ha revisado a lo largo de este punto, la consistencia de datos en Cassandra está sujeta a lo que la aplicación con la cual se usará demande, si no se desea latencia alguna, lo mejor es establecer la consistencia en ONE; si se desea que una operación de escritura jamás falle, el nivel de consistencia en operaciones de escritura deber ser ANY y si la consistencia es la primera prioridad, mediante el uso de la siguiente fórmula:

```
(nodes_written + nodes_read) > replication_factor
```

Cassandra habilita algunas características dentro que aseguran la consistencia a través de las réplicas, dichas características son las siguientes:

- Read Repair (Reparador de Lectura): Existen dos tipos de operaciones de lectura que un nodo coordinador envía a las réplicas: solicitud de lectura directa o solicitud de lectura de reparación de fondo. El número de réplicas consultadas en una lectura directa, está determinado por el nivel de consistencia, mientras que una solicitud de lectura de reparación de fondo es enviada a las réplicas adicionales a las que no se les envió ninguna solicitud de lectura directa. Esta herramienta compara los datos obtenidos de las réplicas consultadas más frecuentemente contra la información contenida en las réplicas a las que le pertenece la información solicitada que se encuentra de fondo o de respaldo, si los datos son inconsistentes, se escriben los datos en las réplicas fuera de línea el valor escrito más reciente. La herramienta de Read Repair puede ser configurada para cada familia de columnas y se encuentra habilitada por defecto.

- Anti-Entropy Node Repair (Reparación de Nodo Anti-Entrópico): Esta herramienta ayuda a asegurar la consistencia de datos en réplicas que han permanecido algún tiempo fuera de línea o en los datos leídos más frecuentemente.
- Hinted handoff (Transferencia indirecta): Esta herramienta obliga a que todos en todas las réplicas se escriban las filas insertadas sin importar el nivel de consistencia configurado; si una réplica se encuentra fuera de línea, es responsabilidad de las demás que cuando la réplica se encuentre nuevamente en línea, reciba una transferencia con las filas afectadas por la operación mientras estuvo fuera de línea.

1.4.2.1.3. *Concurrencia.*

Cassandra no implementa integralmente alguna de las propiedades ACID en las transacciones efectuadas por lo que no se pueden ejecutar uniones ni se contempla el uso de llaves foráneas, por lo que la atomicidad implementada en Cassandra es a nivel de fila, lo que significa que toda operación realizada sobre una sola fila será tratada como una sola.

Para determinar qué registro de la misma columna es el más reciente, Cassandra hace uso del timestamp proporcionado por el cliente de la aplicación, por lo que el registro con el timestamp más reciente es el que será desplegado en una consulta; así mismo en una transacción de actualización de una fila en una sesión multi-cliente, la fila con la actualización más reciente será la que persistirá.

En la versión 1.1 de Cassandra se pueden implementar dos tipos de aislamiento de los registros en las filas, el primero es uno parcial en el cual si una transacción se encuentra ejecutando varias escrituras, una consulta de datos solo podrá realizarse sobre una parte de los mismos. El aislamiento completo permite que mientras una transacción se encuentra realizando una escritura, ninguna consulta podrá tener acceso a la fila que se encuentra siendo utilizada por la transacción de escritura.

Cuando se realiza una escritura sobre una base de datos en Cassandra, ésta se guarda en memoria y en un commit log durante el periodo de tiempo en el que se confirma el éxito de la transacción. (Apache Software Foundation).

1.4.2.1.4. *Lenguajes Particulares e interacción con otros lenguajes de programación.*

En Cassandra 1.1 se implementa el Lenguaje de Consulta de Cassandra (CQL por sus siglas en inglés) el cual en su sintaxis es muy similar al Lenguaje Estándar de Consulta (SQL por sus siglas en inglés) pero con las restricciones propias del modelo de datos de Cassandra anteriormente descritas en este trabajo.

En la versión de Cassandra en la cual este trabajo se encuentra basado, la versión de CQL incluida por defecto es la 2 debido a que la versión 3 aún no es compatible con la versión de Cassandra. (Apache Software Foundation).

El API (Application Programming Interface o Interface de Programación de Aplicaciones) de Cassandra cuenta con manejadores o drivers para distintos lenguajes de programación, tales como Python, PHP; Ruby, Node.js y clientes que hacen uso de

JDBC, lo cual asienta que toda aplicación cliente basada en Java puede acceder y hacer uso de los comandos CQL usado en Cassandra. (Apache Software Foundation).

1.4.2.1.5. *Escalabilidad.*

El concepto de escalabilidad en Cassandra, está enfocado en la agregación de nuevos nodos a la arquitectura de clúster que es la más adoptada, en esta arquitectura una instancia de Cassandra se conforma de una colección de nodos independientes, pudiendo coexistir en diferente número de máquinas físicas en la cual es almacenada una porción del total del conjunto de la información. Al conjunto de nodos que forman un clúster, se denomina anillo.

Cuando se desea recuperar información del clúster que almacenan la base de datos, Cassandra hace uso de un protocolo llamado "gossip" con el fin de ubicar y conocer el estado de la información contenida en el mismo. El protocolo gossip es mediante el cual los nodos intercambian información acerca de ellos mismos y de nodos que tengan a su alrededor o de los que tengan conocimiento. (Neward, 2012) (Apache Software Foundation).

Al iniciar, los nodos miembros de un clúster deben verificar en su archivo de configuración el nombre del clúster al que pertenecen y qué nodos (nodos llamados semilla) deben contactar para obtener información acerca de otros nodos en el clúster, apegándose al funcionamiento del protocolo gossip.

Para evitar fallas en el funcionamiento del protocolo gossip, todos los nodos pertenecientes a un clúster deben contener la misma lista de nodos semilla en su archivo de configuración siendo el principal punto de atención cada ocasión que se

enciende por primera vez un nodo debido a que los nodos previamente integrados al clúster, por defecto, recordarán aquellos nodos con los que hayan ejecutado el mismo protocolo (Apache Software Foundation).

1.4.2.1.6. *Tolerancia a fallos.*

El método de detección de fallas de Cassandra se basa en los resultados obtenidos por su protocolo “gossip” mediante el cual se obtienen y registran los pulsos enviados por cada uno de los nodos del clúster ya sea de manera directa o indirecta. Cassandra no emplea un método rígido para determinar cuándo un nodo está fuera sino que hace uso de un método de detección que toma en cuenta las condiciones de la red de trabajo, carga de trabajo y otros factores que puedan afectar la detección del pulso de la red.

En caso de falla de un nodo, los demás nodos cercanos a él tratarán cada determinado lapso de tiempo de comunicarse con él hasta que la configuración haya cambiado, es decir, hasta que el estado del nodo fuera de línea haya sido actualizado mediante la herramienta administrativa “nodetool” de Cassandra.

En caso de falla en un nodo que ocasione que su estado sea fuera de línea, los demás nodos almacenarán las escrituras perdidas, correspondientes al nodo con falla, por un periodo de tiempo determinado y habilitarán las hinted handoff (transferencias indirectas), este almacenamiento temporal de información durará el tiempo que se encuentre configurado en la opción `max_hint_window_in_ms` que por defecto se encuentra establecida en una hora. (Apache Software Foundation).

A través de la característica hinted handoff, Cassandra asegura la tolerancia a fallos debido a que si existe una solicitud de escritura a un nodo detectado fuera de línea, se guarda la indicación de que dicha operación no se ha realizado y el nodo al que pertenece, por lo que cuando el nodo que estaba fuera de línea recupera su conexión al clúster, se escribe en él el dato de la fila perteneciente a la operación de escritura.

Cuando se usa la característica hinted handoff con Cassandra, se pueden presentar varios escenarios en lo que el software tomará algunos caminos predefinidos, como por ejemplo, si una aplicación exige a Cassandra que las operaciones de escritura siempre sean aceptadas sin importar el número de nodos del clúster que se encuentren fuera de línea, el nivel de consistencia garantizará la durabilidad de la transacción y la accesibilidad de los datos contenidos en la misma hasta que el nodo correspondiente recupere su estado en línea, descargando en él la información enviada al clúster mediante la operación de escritura.

La característica hinted handoff permite que, por diseño, Cassandra efectúe el mismo número de operaciones de escritura aunque el clúster se encuentre operando a su mínima capacidad. (Apache Software Foundation).

En un clúster de Cassandra, ninguno de sus nodos es un nodo maestro o administrador, el protocolo gossip hace que haya un control distribuido de la administración centrado la coordinación en los nodos disponibles, por lo que la tolerancia a fallos puede ser muy alta y confiable siempre y cuando el clúster se sustente de una manera adecuada.

1.5. Bases de datos orientadas a documentos.

1.5.1. *Características generales.*

Un documento en el ámbito de las bases de datos NoSQL, no se refiere a un documento creado por un procesador de texto o a un documento típico en algún formato habitual.

El documento usado por las bases de datos NoSQL es un conjunto de datos semiestructurados en parejas de llaves y valores, típicamente en formato JSON, BSON, entre otros.

Un documento puede considerarse como un agrupamiento de datos en forma de árbol, teniendo un elemento principal o inicial que contiene a su vez otra estructura de datos, pudiendo implementar varios niveles de profundidad.

Por lo anterior, las llaves pueden contener a su vez, otra estructura de datos.

Los documentos almacenados en una base de datos NoSQL orientada a documentos suelen ser similares entre sí, pero no son exactamente iguales, por lo que, analógicamente, las bases de datos orientadas a documentos almacenan documentos en la parte del valor en un almacén del tipo llave-valor.

1.5.2. *Principales implementaciones.*

1.5.2.1. *CouchDB.*

CouchDB (por sus siglas en inglés: Cluster Of Unreliable Commodity Hardware) comenzó a ser desarrollado por Damien Katz, entonces empleado de IBM, a principios

del 2005. El objetivo de esta herramienta, era crear un sistema enfocado a almacenes de objetos de tamaño considerable. Años adelante, este proyecto pasaría a ser parte de la Fundación de Software Apache, de manera que la comunidad tuviera acceso a la herramienta y se contribuyera de manera sustancial al mismo. Actualmente se encuentra disponible para las plataformas UNIX y Windows.

CouchDB es un manejador de bases de datos orientado a documentos, es decir, almacena la información sin necesidad de un esquema, lo cual permite almacenar información con diferentes estructuras, no limitándose a columnas y filas como el esquema normal de las bases de datos relacionales.

1.5.2.1.1. *Modelo de datos.*

Una característica de las bases de datos orientadas a documentos, es que prescinde de la creación de tablas para almacenar los diferentes objetos que puedan contener, siguiendo uno de los enfoques de las bases de datos NoSQL, permite que CouchDB almacene los documentos no en una tabla ni las estructuras correspondientes que son necesarias en el paradigma relacional, mientras que en este se crean tablas relacionadas entre sí por medio de identificadores y llaves, los datos son almacenados en su totalidad en un solo documento.

Un documento básico de CouchDB contiene un identificador, los metadatos, que indican algunas propiedades como cambios en el documento, y la propia información, almacenándola como son, sin necesidad de especificar el tipo de dato, su tamaño, etc. Los metadatos son importantes para manejar la integridad de la información dentro del

sistema, debido a que este es una propiedad del documento que CouchDB le otorga, por lo que no es posible modificar documentos simultáneamente.

Al no estar limitado por un esquema, permite que el desarrollo de aplicaciones no esté ligado o ajustado a un modelo predefinido, como lo es en el caso de las bases de datos relacionales, se puede introducir la información, sin previo declaración de una columna o restricciones más que las que el documentos indique, siendo un enfoque libre y dinámico, sin restricciones de tamaño de columnas, tipos de datos, o dependencia de otros datos, esto lo hace apegándose a las propiedades ACID (atomicidad, consistencia, aislamiento y durabilidad).

Algunas consideraciones en CouchDB y su manejo flexible de documentos no estructurados y sin esquema, es la dificultad para consultar volúmenes considerables de información, mientras que el agregar, modificar o eliminar un documento no representa un trabajo complicado, la misma libertad estructural requiere de un modo especial de acceder a los datos, para esta tarea es necesaria la creación de vistas, que permitan consultar determinada información. La forma en que dichas vistas se ejecutan, es por medio de funciones Java Script.

Aunque no se requiere una estructura, es ampliamente recomendable dar un formato a cada documento, debido a que las funciones de CouchDB manejan la información en formato JSON, permitiendo visualizar los documentos y facilitando el procesamiento de los datos.

Cada base de datos, contendrá un documento de diseño, este, a diferencia de la información almacenada dentro de la base de datos, define la lógica de los demás

documentos, es decir, si los documentos contienen una estructura similar, el documento de diseño permite “estructurarlos” de cierta manera, no solo organizarlos, sino realizar otros cálculos y operaciones. Las funcionalidades de los documentos de diseño, permiten también realizar búsquedas sobre un conjunto de documentos o uno en específico, mantener la consistencia de los datos, así como llevar un control de los cambios realizados.

No existe restricción alguna sobre estos documentos, lo que se busca es que faciliten a nivel de documento las operaciones que se realizan sobre estos, más allá de la lógica de la aplicación, estos ayudaran a un mejor desempeño a ambos niveles. Los documentos de diseño más utilizados son las vistas, están permiten consultar los documentos dentro de la base de datos, por medio de la característica Map/Reduce.

1.5.2.1.2. *MapReduce.*

MapReduce es una vista, que no es más que un conjunto de dos funciones Java Script que permiten representar los datos según requiera, con esto se pueden modelar los datos para su procesamiento, no sin antes adaptarlos con las funciones Map y Reduce, que tienen como objetivo permitir acceder a los datos de manera rápida y eficiente.

Map se encarga de mapear los datos en conjuntos llave-valor, de esta manera, se le otorga una estructura al documento similar a lo que son las relaciones en el paradigma de bases de datos relacional, permitiendo establecer pares uno a uno o uno a muchos, de manera que si existen datos que puedan ser agrupados en conjuntos llave-valores, reduzcan la estructura del documento para un manejo más eficiente.

Reduce por su parte, se encarga de agrupar los conjuntos generados por el mapeo de la función anterior, logrando de esta manera, reducir la forma en la que se procesa un documento. Las funciones de reducción de un documento deberán ser indicadas por la aplicación, por defecto, CouchDB contiene las funciones de conteo (`_count`), suma (`_sum`) y estadísticas (`_stats`), las cuales operan únicamente sobre los conjuntos llave-valor y no sobre los metadatos.

Sin embargo, estas funciones no son de mucha utilidad, cuando los datos de un conjunto llave-valor son de tamaño considerable, de manera que entre mayor sea el tamaño del valor o conjunto de valores el desempeño de las consultas resultará afectado, por ello, es recomendable utilizar esta funcionalidad en conjuntos llave-valor representativos dentro del documento.

1.5.2.1.3. *Consistencia de datos.*

CouchDB no cuenta con algún mecanismo que garantice la consistencia de la información, pero utiliza una aproximación diferente llamada Consistencia Eventual, representa un diferenciador importante de las bases de datos relacionales, ya que la información no se bloquea y podría considerarse como siempre disponible.

Este enfoque, permite realizar cambios en un sistema distribuido, otorgando permisos sobre un nodo para que después los datos sean replicados a los demás, sin embargo, si algún documento está siendo modificado simultáneamente por dos entidades (ya sea en una arquitectura distribuida o única), existirá un conflicto en la revisión de este (la revisión forma parte de los metadatos del documento), ya que la información disponible corresponderá al primero que aplique los cambios al documento,

mientras que la información en el otro documento editado, pertenece ahora a una versión anterior y por ende, obsoleta.

Cuando se presenta este caso, CouchDB notifica sobre las nuevas revisiones del documento, permitiendo cambios sobre la versión más actual a la otra parte, debido a que los documentos no se sobrescriben al modificarlos, solo se aplican los cambios y se modifica su revisión, de manera que se tiene un historial sobre todas las modificaciones que se han hecho en el documento, sin que afecte el desempeño o la integridad del documento ni de la base de datos.

Cabe destacar, que CouchDB se apega a las propiedades ACID que garantizan la ejecución confiable de las transacciones de la base de datos.

1.5.2.1.4. *Transaccionalidad.*

Las transacciones, desde una perspectiva relacional, no son soportadas por CouchDB, aunque este trabaja bajo las propiedades ACID, los documentos son procesados de acuerdo a su revisión, si las revisiones no coinciden, las transacciones fallarán y los cambios deberán volver a realizarse sobre la nueva versión del documento.

De esta forma, las revisiones son necesarias para las transacciones, aunque es posible reducir estas últimas respetando las primeras, lo cual permitirá un manejo quizás más sencillo, aunque aumentaría el nivel de complejidad de la lógica, al final, las modificaciones en el documento fallarán o tendrán éxito.

1.5.2.1.5. *Concurrencia.*

CouchDB versiona los documentos por medio del llamado Control de Concurrencia Multi-Versión (MVCC, por sus siglas en inglés) con el cual al modificar algún documento, éste es agregado nuevamente a la base de datos y se encuentra disponible inmediatamente, permitiendo así procesar incluso cantidades muy grandes de datos en tiempo real.

A diferencia de los sistemas relacionales, no se bloquea el documento, y si existe algún conflicto, siempre es posible notificar cuando una versión más reciente del documento existe, mientras este proceso de actualización se realiza, se permite la lectura de revisiones anteriores del documento, ya que puede considerarse que solo se está agregando información, no alterando su demás contenido.

1.5.2.1.6. *Lenguajes particulares e interacción con lenguajes de programación.*

CouchDB no cuenta con un lenguaje particular para sus operaciones, las cuales son realizadas por medio de peticiones HTTP y se manipulan los documentos como objetos JSON, es decir, tanto lectura como escritura en la base de datos son bajo este formato y mediante funciones con la arquitectura REST, que comprende las funciones POST, PUT, DELETE y GET, las cuales agregan, modifican, eliminar y consultan datos respectivamente.

Almacenar los documentos en formato JSON, permite un acercamiento más orientado a web en cuanto al manejo de datos, mientras que no se restringe a este nivel debido a que este formato es utilizado también en aplicaciones fuera de línea.

CouchDB cuenta con su propio API (Futon), la cual es una interfaz para la manipulación de datos desde un navegador web. También soporta cURL, una herramienta de línea de comandos que permite realizar y recibir peticiones HTTP.

Actualmente, muchos son los lenguajes que soportan JSON, ya sea de manera nativa o por medio software interprete, nativamente, CouchDB esta implementado en el lenguaje Erlang, el cual está enfocado a software de telecomunicaciones, ofreciendo soporte para computo distribuido, concurrente y tolerante a fallos.

En sí, cualquier lenguaje que permita realizar peticiones HTTP y utilice el formato JSON podrá utilizar CouchDB, no limitándose únicamente a aplicaciones web, por lo tanto, lenguajes de programación como Java, C/C++, Python, Ruby, C#, entre otros, podrán conectarse a un servidor, esto desde el lado de las aplicaciones de escritorio, mientras que lenguajes como PHP, ASP y Erlang están enfocados a las aplicaciones web.

1.5.2.1.7. *Escalabilidad.*

No existe problema con la escalabilidad en CouchDB, según la información que sea manejada, es importante siempre monitorear el desempeño del sistema sin importar el tamaño de este, esto ayudara a una mejor planeación e implementación, buscando siempre una mejora óptima en la operación.

Con base a las necesidades de la aplicación, puede o debe escalarse de varias formas, ya sea crear o aumentar nodos de un servidor, aumentando la capacidad de computo del sistema (esto se conoce como escalabilidad horizontal), o aumentando la capacidad del servidor, aumentando el tamaño de memoria, un mayor espacio en disco o una CPU más veloz (a lo cual se le conoce como escalamiento vertical). El tiempo de respuesta y el rendimiento del sistema, son factores determinantes para saber qué tipo de escalamiento se debe implementar.

Como se menciona anteriormente en este trabajo, mejorar las características de procesamiento siempre mejorará la respuesta de la aplicación, mientras que al crear un clúster de servidores representara un aumento de trabajo en la aplicación, debido a algunos factores como la consistencia y concurrencia.

CouchDB está enfocado en la Disponibilidad y Tolerancia a Particiones (parte del Teorema de CAP), con lo cual se tiene una aplicación que permite lecturas y escrituras en tiempo real sobre un clúster o conjunto de servidores replicados, esto, sacrificando la Consistencia de datos, por una Consistencia Eventual de datos, es decir, los cambios realizados sobre un documento, no serán replicados al mismo tiempo en todos los nodos. Por lo tanto, habría que evaluar las necesidades contra las restricciones al momento de escalar el sistema.

1.5.2.1.8. *Tolerancia a fallos.*

La tolerancia a fallos es crucial en una arquitectura distribuida, CouchDB permite cambiar a otro servidor si alguno llegase a fallar, para ello, es importante que los datos replicados sean consistentes en ambas bases, aunque presentaría una baja en el

desempeño, debido a que el servidor de reemplazo no solo debe procesar las peticiones, sino que también, debe monitorear y replicar los cambios aplicados a otros servidores.

CouchDB no soporta un clúster de manera nativa, solo por medio de un software de terceros (como CouchDB Lounge), por lo tanto no puede separar la carga de trabajo entre dos servidores, por lo que es necesaria que la replicación de datos siempre sea monitoreada para que en caso de emergencia, la operación no se vea afectada de manera considerable.

En caso de que se presente alguna falla, la disponibilidad se verá disminuida y la concurrencia de datos podría presentar conflictos, ya que los servidores no pueden sincronizarse.

Las replicaciones en CouchDB se realizan de manera incremental, es decir, solo se transmiten aquellos documentos que presentaron cambios, utilizando el control de revisiones.

La replicación puede ser unidireccional o bidireccional, es decir, si se hacen cambios en una base A, estos se verán reflejados en una base B, igual manera, con una configuración bidireccional, si algún documento en la base B es modificado, también se replicaran en la base A, en el mejor de los casos, ningún documento presentaría conflictos de revisiones, en caso de que estos existan, CouchDB notifica esto y corresponde a la aplicación manejar y operar sobre el nuevo documento.

1.5.2.1.9. *Alta disponibilidad.*

La disponibilidad de los datos en CouchDB es óptima si se tiene un servidor, pero esta puede incrementar o disminuir en un sistema distribuido, en el peor de los casos, la afectación puede deberse a inconsistencia de datos o a fallos en alguno de los nodos, por mencionar los más críticos, sin embargo, con una correcta replicación en un sistema totalmente sincronizado, no representaría problema alguno.

Debido al uso del MVCC (mencionado con anterioridad en este trabajo), el documento siempre estará disponible en su revisión más actual, por lo tanto, un documento en un servidor único o en servidores distribuidos siempre estará disponible para acceder a él, dependiendo de cómo está configurada la base y la forma en la que opera una aplicación, se pueden permitir modificaciones en un documento sin necesidad de replicarlo en tiempo real, debido a que es posible operar bajo el esquema de Consistencia Eventual, tarde o temprano los cambios serán replicados, mientras que en un servidor único siempre será consistente.

1.5.2.2. *MongoDB.*

MongoDB surgió a partir de la necesidad de un almacén de datos sin problemas de rendimiento y cuyo desarrollo se ubica en el año de 2007. Originalmente fue planeada como una herramienta con funcionalidad PaaS (por sus siglas en inglés, Platform as a Service) debido a que muchas empresas debían atender la misma problemática con sus datos.

10gen fue la empresa encargada de proveer una solución de almacenamiento en la nube, desarrollando al mismo tiempo el motor de bases de datos y la aplicación que la utilizaría.

Las versiones de desarrollo pre-MongoDB estaba escrito principalmente en JavaScript (esto en el lado del servidor), estas versiones incluían tanto una aplicación como el motor de bases de datos, debido a la escasa recepción de la aplicación, decidió separarse el motor de la aplicación, surgiendo así MongoDB.

1.5.2.2.1. *Modelo de datos.*

El modelo de datos de MongoDB es orientado a un conjunto de pares llave-valor, ordenadas y estructuradas libremente y no restringidas a un esquema o estructura definida, permitiendo la creación de colecciones adecuadas a las necesidades y restricciones de la aplicación.

Normalmente, los documentos contienen solo la información relevante, de manera que al modelar los datos a representar, se adecuen a las consultas y operaciones que se vayan a realizar, esto para obtener un mejor desempeño en la aplicación.

Los documentos son después correlacionados en colecciones, de ésta manera, las relaciones entre documentos son agrupadas de una manera similar a como lo estaría normalmente en un manejador de base de datos relacional.

El objetivo de las colecciones, es agrupar los documentos para que al momento de desarrollar o gestionar la base de datos, faciliten diferenciar los documentos de

interés de otros que generalmente tienen una estructura diferente, esto permite la creación de índices para que el acceso a los documentos sea más eficiente y no representa un doble trabajo al sistema.

También se cuenta con las características de crear subcolecciones, estas sirven para que dentro de una colección, se reduzca o filtre de alguna manera, los documentos contenidos en una colección queden organizados dentro de aquellos sobre los que se pueden operar.

Si bien MongoDB no requiere de un esquema o estructura al momento de manipular documentos, es recomendable que se manejen adecuadamente las colecciones, tanto en diseño como en implementación, deben apegarse lo más posible a las reglas de negocio de la aplicación, esto permitirá un desarrollo más entendible, y le permitirá al sistema operar de manera óptima.

JSON es el formato utilizado por MongoDB (al igual que CouchDB) para representar los documentos en un conjunto de arreglos de pares llave-valor, sin embargo, a diferencia de CouchDB, son almacenados en formato BSON (JSON binario).

Como es mencionado anteriormente en este trabajo, estos pares son la representación de objetos, por lo que no existe como tal una limitante sobre cuánto, que y como se introduce la información, solo las mismas restricciones de caracteres que son los que utiliza MongoDB para su interpretación.

1.5.2.2.2. *Consistencia de datos.*

MongoDB puede manejar la consistencia eventual o totalmente consistente, debido a que maneja siempre una arquitectura maestro-esclavo, al configurarse de manera totalmente consistente, solo se permiten lecturas a documentos que no están siendo modificados, y únicamente a la última versión, ya que MongoDB no almacena un histórico de los documentos.

La forma en que las escrituras y lecturas son realizadas son bastante parecidas en ambas configuraciones, pero las escrituras solo se aplican en el nodo maestro o primario, para después replicarse a los demás nodos esclavos o secundarios. Las lecturas son permitidas en cualquier nodo, pero la información que se tenga en ellas puede o no ser la última versión del documento, aunque también puede limitarse en su totalidad, todo esto debido al nivel de configuración que la aplicación requiera.

1.5.2.2.3. *MapReduce.*

MongoDB cuenta también con la funcionalidad MapReduce, al igual que CouchDB, esta necesita dos funciones en lenguaje JavaScript map y reduce, que se encarguen de devolver un conjunto de documentos ordenados por llave-valor.

Es posible especificar parámetros en las funciones MapReduce, las cuales permiten manipular delimitar, ordenar y procesar los documentos de una manera mayormente orientada a la programación que a nivel base de datos.

Por medio de estas funciones es posible realizar: conteos, agrupamientos y distinciones de documentos, funcionalidades que no es posible obtener directamente

de la base de datos, ya que la manera en que se consulta entre una base orientada a documentos y una base de datos relacional difiere enormemente, tanto a nivel estructural, como a nivel de datos.

MapReduce en funcionamiento, es el equivalente a las funciones de agregación en un modelo relacional, presentando la información en vistas especializadas sobre una colección de documentos en específico, siempre en múltiples documentos con estructura llave-valor.

Es importante mencionar que únicamente es operable sobre una colección a la vez, de tal manera, que si es requerido realizar una de estas funciones sobre mas colecciones de datos, resulta imposible ya que indica un grave error de diseño de la base de datos.

1.5.2.2.4. *Transaccionalidad.*

MongoDB provee un nivel de transaccionalidad bastante básico, debido a que las operaciones son realizadas a nivel de documento, uno a la vez, entonces, no todas las convenciones de ACID no aplican a este nivel y pasa ser manejado a nivel de aplicación, esto representa un problema en una arquitectura distribuida, debido a que las transacciones y la consistencia en una base de datos son esenciales para el correcto funcionamiento del sistema.

La durabilidad de una escritura en una transacción, es permitida siempre y cuando se haya habilitado la característica en MongoDB.

1.5.2.2.5. *Concurrencia.*

MongoDB permite que un documento sea leído por múltiples aplicaciones, pero solo una de ellas puede escribir a la vez, esta característica se le llama lectores-escritor, permitiéndoles acceder a un documento consistente y garantizando que esté libre de errores.

De manera que cuando un documento está modificando, este se bloquea, permitiendo solo acceso de lectura y solo será desbloqueado cuando los cambios hayan sido aplicados. El nivel en que este bloqueo se revisa es a nivel base de datos, debido a que los documentos no tienen esquema definido y están almacenados como tal.

Las operaciones que bloquean la base de datos se listan en la Tabla 1-2.

Operación	Tipo de bloqueo
Consulta	Bloqueo de lectura.
Uso de cursores	Bloqueo de lectura.
Inserción	Bloqueo de escritura.
Eliminación	Bloqueo de escritura.
Actualización	Bloqueo de escritura.

MapReduce	Bloqueo de lectura y escritura, a no ser que sea especificada la operación como no atómica. Algunas partes de los trabajos en MapReduce pueden ejecutarse concurrentemente.
Creación de índices	La construcción de índices bloquea la base de datos por periodos extendidos de tiempo.
Función db.eval()	Bloqueo de escritura. Bloquea además cualquier otro proceso JavaScript.
Función eval()	Bloqueo de escritura. Si se utiliza con la opción de bloqueo *nolock,* la opción de bloqueo de escritura es ignorada, pero no es posible la escritura de datos en la base.
Función aggregate()	Bloqueo de lectura.

Tabla 1-2: Operaciones que bloquean una base de datos en MongoDB

La concurrencia en un sistema fragmentado (Sharding), distribuye las colecciones de datos a través de los distintos nodos existentes, de manera que los bloqueos pueden afectar a múltiples bases de datos. También es importante mencionar, que a pesar de estar en un sistema distribuido, las escrituras siempre se aplicaran sobre el nodo maestro.

1.5.2.2.6. *Lenguajes particulares e interacción con lenguajes de programación.*

MongoDB utiliza un ambiente JavaScript en su totalidad, ya sea por medio de Shell o un cliente que lo soporte, de manera que todas las operaciones y funciones disponibles nativamente, son ejecutadas en este lenguaje.

El Shell permite ejecutar tareas administrativas y ejecutar procedimientos desde la línea de comandos, adaptándose a JavaScript y estrictamente al formato JSON, es decir, todas las operaciones se ejecutan, no limitándose únicamente a las instrucciones indicadas, provee de todo un ambiente para utilizarlo como se haría normalmente en una página web, por decirlo de algún modo, se pueden declarar objetos, funciones, etc.

Es importante mencionar, que aunque los documentos son mostrados y procesados en formato JSON, MongoDB los almacena en BSON, internamente se encarga de la conversión de los formatos.

Cualquier lenguaje de programación que cuente con un intérprete de JavaScript puede ser utilizado con MongoDB, si bien este provee un Shell para realizar operaciones, tanto en los datos como gestionarlas, estas tareas pueden realizarse desde cualquier cliente compatible con esta tecnología.

Existen una gran cantidad de conectores para MongoDB, entre los principales se encuentran los C, C++, C#, Java, PHP, Python, Ruby; independientemente del lenguaje de programación elegido por la aplicación, todas las instrucciones son procesadas en JavaScript, permitiendo una gran facilidad al momento de desarrollar aplicaciones web

o de escritorio, por lo que incluso por medio de formularios web básicos se podría manipular una base de datos, sin embargo, esto no explotaría todo el potencial de MongoDB.

1.5.2.2.7. *Escalabilidad.*

MongoDB fue creado con la intención de ser altamente escalable, de manera que pueda distribuirse la carga de procesamiento entre los diferentes nodos en un sistema distribuido. La escalabilidad es principalmente horizontal, es decir, se agregan servidores “esclavos”, según se requieran, cada uno de ellos, almacenara una parte de los datos, a diferencia de otro sistema donde la información es replicada es la misma, esta aproximación permite separar la información almacenada a través de los demás nodos, a este proceso se le conoce como Sharding o fragmentación/particionamiento de datos.

El particionamiento de datos es invisible en el servidor y puede ser ejecutado manual y automáticamente, generalmente el primer es altamente configurable, pero a diferencia del segundo, requiere mantenimiento constante. La concurrencia se ve beneficiada en gran medida, debido a que ahora puede estar en un nodo que contiene menor volumen de datos y por ende, las operaciones realizadas sobre los documentos son generalmente más rápidas.

1.5.2.2.8. *Tolerancia a fallos.*

MongoDB ofrece una gran tolerancia a fallos, esto debido a las opciones de replicación que se tienen, si bien es una configuración de maestro/esclavos, cada

replica contiene una copia de los datos, si en cambio se está utilizando el Sharding o fragmentación, se cuenta con información separada y dependiendo de la configuración, replicada, esto permite que en caso de que algún nodo falle, otro u otros nodos se encarguen de procesar las instrucciones correspondientes.

Debido al diseño y al manejo de nodos como maestro/esclavos, en caso de que el nodo maestro falle, las escrituras no podrían ser realizadas en su totalidad, si se utiliza el particionamiento de datos, existen opciones para configurar que nodo suplanta al principal, ya que la aplicación distribuida se encarga de gestionar todos los sistemas como uno solo.

1.5.2.2.9. *Alta disponibilidad.*

La disponibilidad de datos es MongoDB es alta, debido a que al momento de replicar los datos, ya sea por medio de una estructura jerárquica o particionamiento, todos o algunos datos, son replicados en los demás sistemas, siendo el nodo maestro el único en el cual se permite la escritura de datos, dejando a los nodos secundarios únicamente la opción de lectura.

Si se configura el equipo como maestro/esclavos, siempre se tendrá una alta disponibilidad, debido a que cada nodo esclavo contiene el mismo conjunto de datos que el nodo maestro, sin embargo, los cambios realizados en el nodo maestro deben aplicarse a los nodos esclavos para que la información sea consistente, lo cual, no representa una tarea complicada para MongoDB.

Si se tiene una arquitectura maestro/esclavos, en principio, la disponibilidad seguirá siendo alta, debido a que se crea redundancia de datos en los nodos, distribuyendo información consistente a través del sistema, otorgándole prioridad a los fragmentos de información a través del sistema y manejando un nivel de consistencia eventual, garantizando así, una alta disponibilidad, e internamente funciona de manera similar a la aproximación anterior.

1.6. Bases de datos orientadas a grafos.

1.6.1. *Características generales.*

Una base de datos orientada a grafos es aquella que también representan entidades y sus propiedades pero, adicionalmente, también representan cómo se relacionan las entre sí.

En este tipo de bases de datos las entidades se llaman nodos que se relacionan entre sí por medio de relaciones conocidas como aristas, las cuales tienen significado direccional. A su vez, tanto nodos como aristas pueden tener propiedades.

Al conjunto de nodos y relaciones, se le denomina grafo.

Estas bases de datos suelen ser implementadas en entornos que tengan relaciones complejas, como las redes sociales, y en los cuales se requieran hacer análisis de estructuras complejas y encontrar patrones dentro de las mismas.

1.6.2. *Principales implementaciones.*

1.6.2.1. *Neo4j.*

1.6.2.1.1. *Modelo de datos.*

Las bases de datos NoSQL usualmente manejan un modelo de datos en el que la base es la orientación clave-valor. Las bases de datos orientadas a grafos agregan un elemento más a su modelo: nodos, relaciones y propiedades.

Un nodo es usualmente usado para representar alguna entidad. (Neo Technology, 2013).

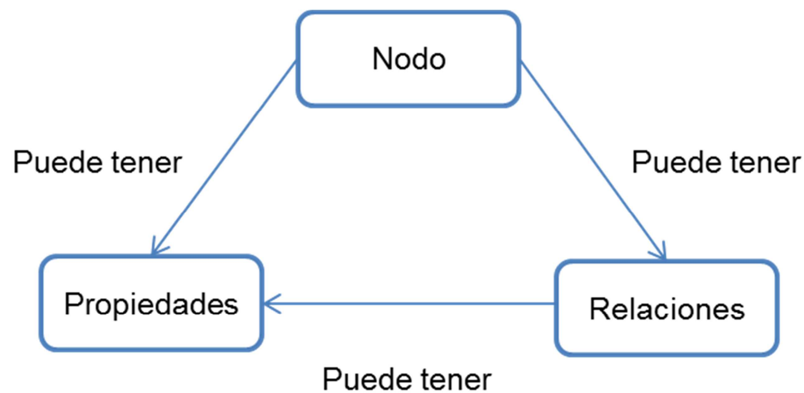


Figura 1-1: Representación de un nodo

Las relaciones entre los nodos facilitan las siguientes acciones:

a) Encontrar datos relacionados,

b) Conectar dos nodos y

c) Garantizar tener nodos de inicio y de fin,

Así mismo, las relaciones pueden ser en dirección de entrada y salida de un nodo, además que siempre son bidireccionales y pueden representar que un nodo tenga relación consigo mismo. (Neo Technology, 2013)

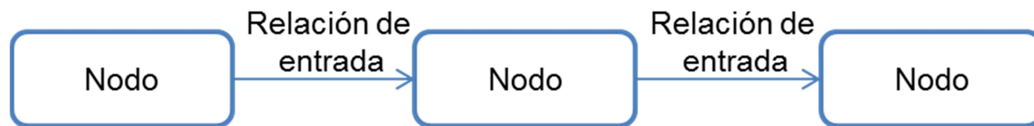


Figura 1-2: Representación de nodos y relaciones

Las propiedades son conjuntos de datos llave-valor donde la llave es una tipo de dato cadena y los valores pueden ser tipos de datos simples o arreglos.

Una etiqueta es un grafo nombrado que es usado para agrupar nodos en conjuntos: todos los nodos etiquetados con la misma etiqueta pertenecen al mismo conjunto.

Para consultar un grafo, se hace uso de los *recorridos* con los cuales se navega a través de los nodos desde un nodo inicial hacia sus nodos relacionados mediante algoritmos.

1.6.2.1.2. *Concurrencia.*

Por defecto, una operación de lectura leerá el último valor finalizado correctamente a menos que una modificación local dentro de la transacción exista, es decir, las operaciones de lectura no bloquean por lo que las lecturas no repetibles pueden suceder. Es posible alcanzar un nivel de aislamiento muy alto mediante los bloqueos manuales de lectura y escritura. (Neo Technology, 2014).

1.6.2.1.3. *Transaccionalidad.*

El comportamiento de bloqueo en Neo4j puede ser resumido en los siguientes casos (Neo Technology, 2014):

- Cuando se añade, cambia o remueve una propiedad en un nodo o relación, un bloqueo de escritura se implementa en el nodo específico o relación.
- Cuando se crea o borra un nodo, un bloqueo de escritura se implementa para el nodo específico.
- Cuando se crea o borra una relación, un bloqueo de escritura se implementa en la relación específica y en ambos nodos.

1.6.2.1.4. *Lenguajes particulares.*

El lenguaje nativo utilizado por Neo4j es llamado Cypher, el cual es un lenguaje declarativo que permite operaciones expresivas y eficientes de consulta y actualización de datos.

Cypher se compone de distintos componentes de otros lenguajes: las palabras clave basadas en SQL, la coincidencia de patrones se basa en la aproximación de patrones de SPARQL y algunas semánticas se basan en lenguajes como Python.

Las consultas en Cypher son construidas usando cláusulas SQL, son ligadas juntas y alimentan conjuntos de resultados intermedios entre ellas.

Algunas cláusulas de lectura de grafos son:

`MATCH`: Para buscar coincidencias de cierto patrón de grafos.

`WHERE`: No es una cláusula en sí pero forma parte de `MATCH`, `OPTIONAL`, `MATCH` and `WITH`. Agregan restricciones a un patrón o filtra los resultados intermedios con `WITH`.

`RETURN`: Resultado de la consulta.

Algunos ejemplos de cláusulas que son usadas para actualizar un grafo:

`CREATE` (y `DELETE`): Crea (y borra) nodos y relaciones.

`SET` (y `REMOVE`): Establece valores a las propiedades y agrega etiquetas en los nodos usando `SET` y `REMOVE` las borra.

`MERGE`: Une los nodos existentes o crea nuevos nodos y patrones. Es muy útil en conjuntos con restricciones de unicidad. (Neo Technology, 2014)

1.6.2.1.5. *Escalabilidad.*

La escalabilidad en Neo4j es conocida como un paquete llamada Alta Disponibilidad (HA, por las siglas en inglés de High Availability), lo cual habilita tres características (Montag, 2013):

a) Redundancia total de datos.

b) Tolerancia a fallos del servicio.

c) Escalabilidad linear de lectura.

El modo HA de Neo4j habilita una arquitectura mayormente de lectura de escalabilidad horizontal que permite al sistema administrar una mayor carga que una sola instancia de base de datos. Dicha arquitectura de bases de datos es tolerante a fallos donde varias bases de datos esclavas, pueden ser configuradas como réplicas exactas de una base de datos maestra.

Neo4j es una base de datos orientada a grafos que privilegia su uso en entornos distribuidos y en el cual enfoca todas sus características.

1.7. Comparativa de los principales productos NoSQL.

Base de datos	Modelo de datos	Transacción (teorema de CAP)	Consulta	Licencia	Índices	Sharding	Concurrencia
Cassandra	Columna-familia	PA	Hive-PIG, JavaAPI, MapReduce	Software libre (Apache)	No	Sí	N/A
CouchDB	Documento	PA	Lucene, Cloudant, REST API, JAVA API, MapReduce	Software libre (Apache)	Sí	No	Control de Concurrencia de Muiltiversiones
MongoDB	Documento	PA	Formato basado en BSON, REST	GNU	Sí	Sí	N/A

			API, JAVA API MapReduce				
Neo4j	Grafos	CA	Cypher, REST API, JAVA API	OpenSource y Comercial	No	No	Bloqueo

Tabla 1-3: Tabla comparativa entre los distintos tipos de bases de datos NoSQL. (Indrawan-Santiago, 2012) (Hecht & Jablonski, 2011)

1.8. Casos de éxito con tecnologías NoSQL.

La ventaja que ha tenido el auge de las tecnologías NoSQL se debe en gran medida, a que las compañías desarrolladoras liberan su producto para que sus clientes lo adapten a sus necesidades. Claro está, que si una herramienta está adaptada perfectamente a lo que se requiere.

Las necesidades de cada sistema desarrollado son muy diferentes entre sí, no solo los diferentes tipos se enfocan a resolver una cierta necesidad, por lo que también permite acoplarse con otras tecnologías, logrando así no solo un sistema enfocado a un único problema, sino a varios, permitiéndole al usuario o al cliente hacer un mejor uso de sus recursos.

Tomando las problemáticas que NoSQL pretende solucionar, como lo son la escalabilidad y el manejo de un alto volumen de datos, las múltiples herramientas disponibles cumplen estos requisitos sin mayor problema, sin embargo, dependiendo del diseño de los sistemas, de la arquitectura a utilizar y de los tipos de operaciones a

procesar, cada una ofrecerá un diferenciador, el cual puede ser crucial al momento de implementar y dar mantenimiento, puesto que la forma en las que cada una ofrece solucionar un problema, hace de la elección de la herramienta correcta una tarea imprescindible.

Para efectos de este trabajo, se enlistarán a continuación implementaciones exitosas de bases de datos NoSQL del tipo clave-valor y orientada a documentos.

1.8.1. *Sistemas de almacenamiento con bases de datos del tipo clave-valor (Cassandra).*

1.8.1.1. *BlueRunner.*

BlueRunner es una aplicación de correo en la nube creado por IBM, utiliza Apache Cassandra como backend debido a que soporta una considerable cantidad de consultas sobre un gran volumen de datos, además de que esta soporta computo en la nube y el uso de una arquitectura distribuida, logrando así una alta disponibilidad e integración con una aplicación de correo.

Debido a las propiedades de Cassandra y a la naturaleza de la aplicación, esta permite un alto grado de escalabilidad, la consistencia eventual es bastante útil, debido a que los correos o la información no es compartida, y por ende, permite tener un procesamiento óptimo, además de que el modelado de datos con Cassandra es relativamente simple.

1.8.1.2. *Spotify.*

Spotify es un servicio de distribución de audio, tanto en plataformas de escritorio como dispositivos móviles, por lo que ha tenido un gran auge y un buen recibimiento, debido a esto, ha optado por utilizar Cassandra, guardando información sobre el contenido (en este caso, para el audio, todas las propiedades de las pistas, como lo pueden ser nombre de la canción, autor, disco, etc.), además de las configuraciones de los usuarios y la información y estadísticas de estos.

Utiliza principalmente las características de replicación, lo que permite ofrecer un servicio ininterrumpido, la consistencia eventual y la escalabilidad.

1.8.1.3. *Twitter.*

Twitter es una red social de microblogging, es decir, permite a los usuarios compartir información de manera sencilla. Cassandra no se utiliza para almacenar esta información, la utilidad es para almacenar las estadísticas que Twitter realiza sobre esta, como la información geoespacial, los Trending Topics (informacion con mayor mención en un determinado tiempo) tanto local como global en tiempo real, cabe destacar que la información compartida es enorme, ya que esta red social es bastante popular.

Nuevamente, se explotan las características de replicación y almacenamiento, sin que se afecte el rendimiento, debido a que la información que Cassandra almacena no es crucial para el funcionamiento del sitio, si lo es para sus características y para llevar una minería de datos más efectiva.

1.8.2. *Sistemas de almacenamiento con bases de datos orientadas a documentos (CouchDB).*

1.8.2.1. *Arbit.*

Arbit es una herramienta básica para administración de proyectos, permite dar seguimiento al código y a los problemas que surjan de este, por medio de una interfaz sencilla, además de utilizar una wiki (una página colaborativa para almacenar contenido relevante sobre un tópico) para documentar todos los procesos.

CouchDB es utilizado para almacenar los problemas sobre los proyectos, de manera que sea fácil darle seguimiento, el principal beneficio que aporta a Arbit es la capacidad de replicación y su tolerancia a fallos.

1.8.2.2. *TapirWiki.*

TapirWiki es una aplicación que permite gestionar tanto el contenido como la presentación de las páginas por medio del navegador web, todo esto de manera sencilla y rápida. Es necesario tener instalado CouchDB, ya que es utilizado para almacenar los datos. Para agregar contenido (páginas web), basta con crearla y guardarla, la aplicación la almacenara de manera automática en CouchDB.

TapirWiki utiliza la replicación y el almacén de datos de CouchDB, pudiendo migrar fácilmente contenido de una wiki a otra.

1.8.3. *Sistemas de almacenamiento con bases de datos orientadas a documentos (MongoDB).*

1.8.3.1. *Foursquare.*

Es una red social que se basa en datos geo-espaciales, permitiéndoles a los usuarios registrar su posición en lo que se denomina como “lugares de interés” desde su dispositivo móvil.

Se utiliza MongoDB debido a que la aplicación ha crecido enormemente, de manera que se optó por mudarse de un modelo relacional a un NoSQL, además de que utiliza el Sharding o fragmentación, la capacidad de replicación de datos, la facilidad de escalar la solución, la característica de manejo de datos geo-espaciales y permite utilizar un modelo de datos mucho más simple a comparación del modelo relacional inicial de FourSquare.

1.8.3.2. *Viber.*

Es una aplicación para dispositivos móviles que permite el intercambio de mensajes y llamadas telefónicas entre usuarios por medio de redes inalámbricas almacenando la información generada entre mensajes, como propia de los usuarios.

MongoDB es utilizado debido a su enorme base de datos de usuarios y la carga que estos generan por su capacidad de almacenamiento y manejo de datos, además de que es una solución fácilmente escalable para una aplicación que tiene una inmensa carga de datos y una base de usuarios en crecimiento exponencial.

INGENIERIA
F I

CAPÍTULO 2

Análisis de la problemática y requerimientos

2. Análisis de la problemática y requerimientos.

En este capítulo se muestra la definición, diseño y análisis del Sistema de Atención de Incidencias (SAI), el cual permitirá mostrar las diferencias en las etapas de diseño e implementación entre el modelo relacional y el modelo NoSQL, mientras que la operatividad del sistema permanece intacta, de manera que la eficiencia del sistema solo se vería alterada acorde a las metodologías y tecnologías empleadas para su implementación.

2.1. Definición del problema.

Con el auge tecnológico actual en el que se involucran todas las áreas de negocio de una empresa, el contar con una herramienta que permita la administración de las fallas o errores que se presentan en la infraestructura informática y en el catálogo de software de la organización es necesaria para las actividades diarias, por lo que la administración correcta de los recursos disponibles para solucionar los incidentes mencionados debe ser meticulosa y eficiente. Dada la situación anterior se requiere de un sistema que permita levantar incidentes (tickets) para solucionar algún problema surgido en el flujo de negocio, este problema puede ser operativo, funcional y/o de infraestructura principalmente.

Debido a la naturaleza del negocio, se requiere de una pronta respuesta y solución a cada problema presentado, de manera que será necesario canalizar cada problema entre las dos partes, la parte usuario, que es la que presentó problemas, con la parte encargada correspondiente, las cuales intercambiarán información sobre cuál

problema se presentó y pueda darse seguimiento óptimo para la solución del problema lo antes posible.

2.2. Situación actual.

2.2.1. *Sistema de Atención de Incidencias: SAI.*

El sistema actual está dividido en cinco áreas:

- **Operativa.** Área que mantiene contacto directo con los usuarios y que funge como la capa más alta en el sistema.
- **Sistemas.** Área que desarrolla las aplicaciones para el negocio; se encarga de coordinar a las demás áreas para crear una aplicación que el área operativa utilizará.
- **Base de datos**. Área encargada del mantenimiento, creación y adecuación de las bases de datos; tendrá control sobre los accesos y permisos sobre las mismas.
- **Soporte.** Área encargada de la administración de la red de datos y el software en cada estación de trabajo.
- **Infraestructura.** Área encargada de la administración del hardware, tanto estaciones de trabajo como servidores.

El SAI permite ingresar información de los responsables, tanto como de los objetos con los cuales se cuenta dentro de cada sistema, como aplicaciones, bases de datos, servidores, entre otros; esta información ayudará a dar un mejor seguimiento

sobre el levantamiento de las incidencias y permitirá la correcta canalización cuando se presenta una.

Cada área (exceptuando el área operativa) tiene un flujo similar y almacena su información de forma independiente a las demás, acorde a sus funciones y necesidades. Cada área será responsable de la información que SAI administra y almacena y sólo ella puede gestionar y visualizar la información correspondiente, de manera que sea el sistema el que gestione de manera óptima cada incidencia reportada.

El funcionamiento del SAI se encuentra basado en formularios para el ingreso de la información de cada área, así como de la creación y seguimiento de las incidencias detectadas, cumpliendo con las funcionalidades de un directorio en su uso básico y en su uso avanzado como un administrador de incidencias.

Cada área será capaz de levantar un incidente, indicando únicamente la posible causa del mismo y el destino, permitiendo que el sistema utilice sólo la información necesaria para lograr un rendimiento óptimo al momento de solucionar cada incidencia.

Actualmente, el SAI permite la generación de reportes de incidencias con el objetivo de contar con una visión de la eficiencia alcanzada en la resolución de incidencias. Los reportes permiten aplicar correcciones sobre algunas áreas y contar con un historial sobre las aplicaciones y/o áreas para responder preguntas como: ¿Qué áreas presentan la mayor cantidad de fallos en cierto periodo de tiempo: semanal, mensual?, ¿cuáles se resuelven en menor tiempo?, ¿qué tipos de fallos son los más comunes?, entre otra información relevante.

2.3. Principales requerimientos funcionales.

2.3.1. *Administración de incidencias.*

El SAI crea una lista de incidencias, con la cual pueda darse seguimiento a errores u observaciones que se presenten en cada aplicación, tanto en el área operativa (los usuarios finales) hasta el área de desarrollo (sistemas), la cual funciona en conjunto con las demás y su interacción es necesaria para identificarlas posibles causas y ejecutar las soluciones del problema reportado por el usuario final.

Sobre la lista de incidencias se llevará el control del negocio y, en caso de requerirse, mantenimiento o correcciones por alguna falla o desperfecto. Por medio de la administración de incidencias llevada por el SAI, se pueda agilizar la solución a los errores o problemas detectados.

Para crear una incidencia, se seleccionará el tipo de la misma que va ligada al tipo de área responsable, también se debe ingresar una descripción detallada del problema de manera que los responsables puedan identificar la causa de la incidencia, la o las consecuencias y las posibles soluciones a aplicar. Como funcionalidad agregada se incluye la posibilidad de asignar etiquetas que representen de forma resumida la problemática reportada.

Una vez creada la incidencia, se creará el ticket notificando tanto al usuario como al responsable sobre el problema y mediante el SAI, se podrá registrar cada actividad realizada hasta que la incidencia sea cerrada.

Al modificar una incidencia, se podrá actualizar su estado, así como agregar las actividades que se han desarrollado a lo largo del proceso de solución del problema.

Los posibles estados de una incidencia podrán ser:

I. En revisión. Corresponde al análisis preliminar sobre las posibles causas del problema por el cual se ha ingresado la incidencia al SAI. En este punto se determinará si la incidencia continúa el flujo de solución establecido por la lógica propia del SAI, es cancelada por no ser propiamente una incidencia o porque su solución se encuentra fuera del alcance del software y/o hardware que se administran mediante el SAI.

II. Activa. Estado en el que la incidencia es clasificada dentro del alcance de SAI. En este estado se analiza a profundidad la problemática y las repercusiones de la misma dentro del área de negocio que la reportó. En este estado se agregarán las actividades que se realicen para la solución de la misma.

III. Cancelada. Estado en el que la incidencia se encuentra fuera del alcance de los componentes administrados por el SAI o fue cancelada por el usuario que originalmente manifestó la problemática.

IV. Cerrada. Estado último en el que la problemática de una incidencia es solucionada mediante los mecanismos pertinentes. Dicho estado sólo puede ser establecido por usuarios administradores del SAI. Cuando una

incidencia alcance el estado de Cerrada, se notificará vía correo electrónico al usuario que la registró.

Después de que una incidencia se encuentre Cerrada, se podrá almacenar como archivo histórico para generar reportes y establecer posibles soluciones para incidencia semejantes presentadas posteriormente.

El SAI será desarrollado para un ambiente web, permitiendo el acceso desde cualquier equipo con acceso a la red en la que sea implementado. La entrada que requiere el sistema será la información de los usuarios y del software y/o hardware con el que se cuente.

En el SAI no sólo será posible el levantamiento de incidencias, al contener información de usuarios, administradores, software, hardware, entre otros, también podrán consultarse las tareas como mantenimiento, desarrollo, ajustes y adecuaciones de software, teniendo siempre de forma disponible la información pertinente.

En lo que respecta a tareas de mantenimiento, desarrollo, ajustes y/o adecuaciones, el SAI la considerará un caso especial de incidencia debido a que sólo se agregarán una vez hayan sucedido y como referencia de cada componente.

La información referenciada en párrafos anteriores deberá especificarse antes para delimitar el alcance de cada área a nivel de usuario y establecer parámetros que ayuden a la gestión de los problemas presentados.

El proceso de administración de incidencias se describe en los siguientes pasos:

I. ***Análisis preliminar***. – Este proceso requiere información básica sobre la incidencia, con base a un análisis simple de la problemática presentada, se buscarán incidencias similares dentro del SAI, generando así un nuevo elemento o reabriendo uno ya existente.

II. ***Apertura/Reapertura***.- En este proceso se especificarán los detalles de la incidencia, así como el componente que presenta problemas e información relevante para su posterior análisis. En caso de ser una apertura, se generará un nuevo elemento en la base de datos. En caso de que se trate de una reapertura, la información de la incidencia anterior será copiada a un nuevo elemento, respetando la información anterior con fines de referencia.

III. ***Análisis completo***.- En esta fase del proceso, se analizarán las posibles causas de la incidencia, se notificará a los administradores del componente así como a los posibles usuarios afectados de alguna manera por la incidencia.

IV. ***Proceso de resolución***.- Una vez analizado el problema, en esta fase se definen los pasos a seguir para su posible solución. Este proceso será iterativo, de manera que hasta que no se resuelva el problema, se podrá seguir aportando información sobre el ciclo de vida de la incidencia, tanto como las observaciones, aciertos y errores a lo largo del proceso de resolución.

V. ***Cierre***.- En esta fase se entiende la problemática ya fue solucionada o que no se encontraron argumentos válidos para que el problema sea considerado una incidencia. Se notificará a los administradores del componente que la incidencia se ha cerrado o cancelado de acuerdo al administrador. Es posible también que en esta fase, se aporte información sobre cierres erróneos o cancelaciones de incidencias de acuerdo a los lineamientos establecidos para identificar incidencias.

De aquí, el proceso inicial corresponde a la identificación de problemas dentro del SAI, de manera que conforme crezca la base de datos de incidencias, se contarán con problemas más comunes, los cuales pueden presentarse nuevamente con el paso del tiempo.

Inicialmente un usuario proporciona cierta información al administrador del SAI, este último consultará las incidencias existentes buscando incidencias son problemas similares, y con base a los resultados obtenidos, se tomará la decisión de crear una nueva incidencia o bien, reabrirla.

Es importante establecer una manera en que las incidencias se manipulan, ya que una incidencia que se presenta nuevamente, puede o no provenir de las mismas causas que la vez anterior, o podría incluso aportar nuevos problemas al componente.

Para ello se realiza la reapertura de incidencias, con la cual se toma la información existente y se genera un nuevo documento al cual se le maneja como una nueva incidencia, con la diferencia de que la información y eventos previos se

mantienen con fines informativos. El proceso de reapertura es entonces un caso especial de la apertura.

Si el proceso durante una reapertura se resuelve con la misma información, se cierra nuevamente y se almacenará con los demás datos, sin embargo, en caso de que presente algunas variaciones en el proceso de resolución, es necesario modificar dicho proceso y seguir el mismo flujo como si se tratase de una nueva incidencia.

El Diagrama 2-1 ilustra el proceso de análisis preliminar:

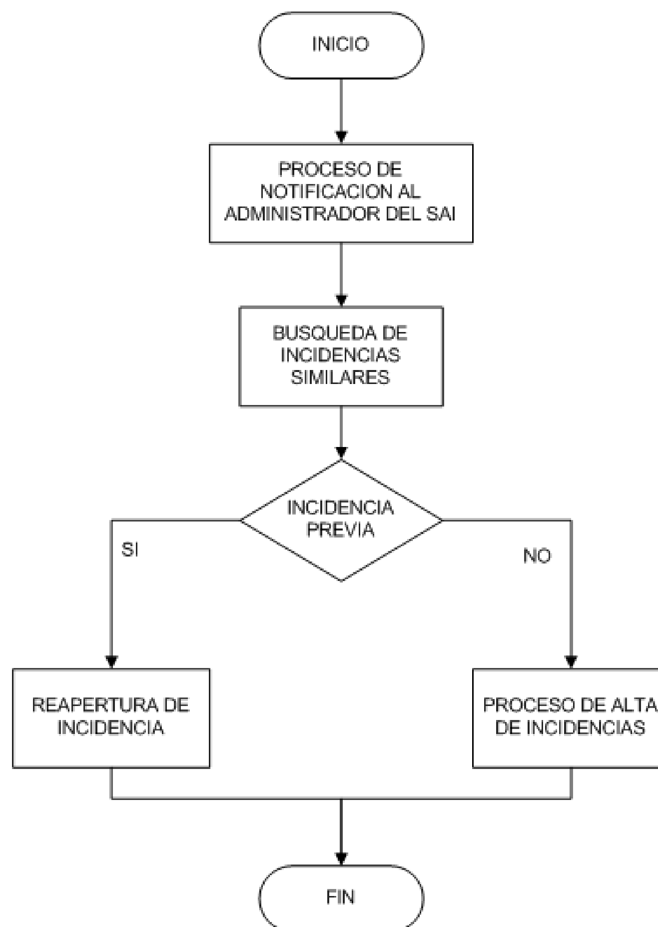


Diagrama 2-1: Proceso de análisis preliminar.

El Diagrama 2-2 muestra el proceso de apertura de incidencias.

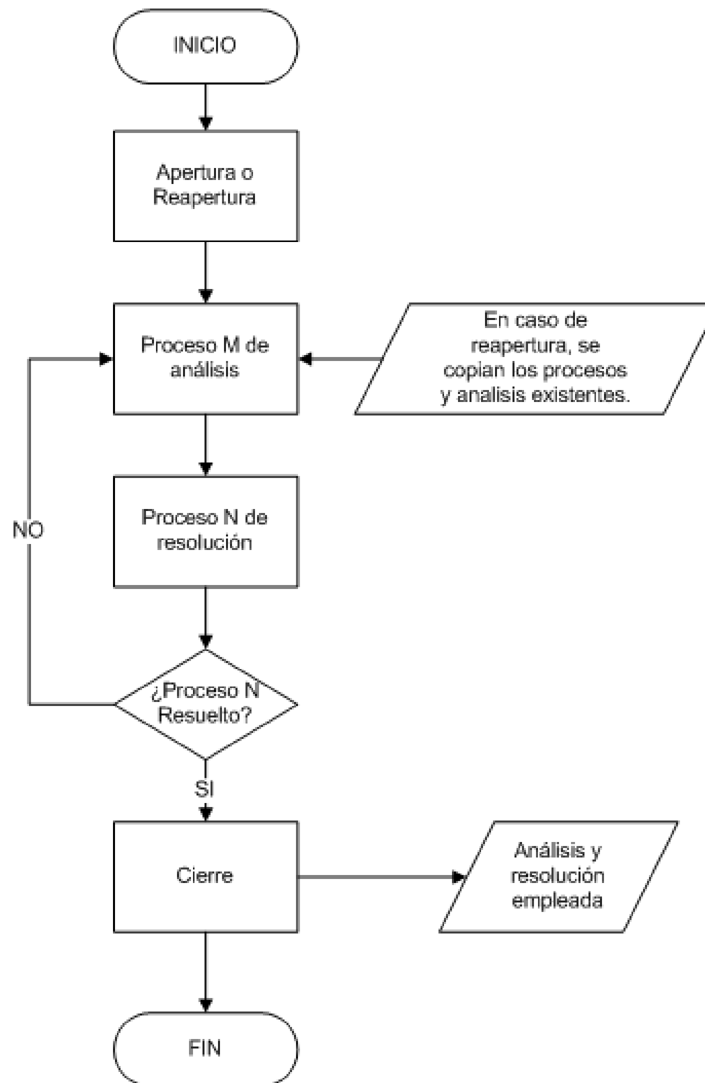


Diagrama 2-2: Proceso de apertura de incidencias

2.3.2. *Análisis de la información: generación de reportes.*

Uno de los principales objetivos del SAI es la explotación de la información de incidencias contenida en él, con la finalidad de encontrar patrones y generar estadísticas para establecer tareas de mantenimiento, monitoreo, entre otras, que

contribuyan a la toma de decisiones en lo que a los componentes administrados por el SAI se refiere.

Los reportes con los que el SAI contará desde el inicio de su operación, serán reportes estadísticos que reflejarán la actividad dentro del mismo y que permitirán una visión generalizada de las áreas y sus problemáticas, la cual podrá ser profundizada de acuerdo a la información relevante.

Los reportes iniciales del SAI son los siguientes:

I. *Reporte de línea de tiempo de incidencias*. Reporte que muestra el número de incidencias clasificadas por estado que se encuentran en el SAI, pudiendo seleccionar el intervalo de tiempo deseado así como el área en la que sea necesario profundizar.

II. *Reporte del promedio de tiempo de respuesta a incidencias por área*. Reporte que muestra el número de días promedio en el que las incidencias alcanzan el estado de cerrada por cada área del SAI, con la flexibilidad de delimitar el intervalo de tiempo deseado por año y/o mes para visualizar resultados objetivos y de utilidad para las áreas involucradas en la incidencia.

III. *Reporte de incidencias más comunes con base en etiquetas*. Reporte en el que se parte del área y un intervalo de tiempo para identificar las incidencias más comunes dentro del SAI haciendo uso de las etiquetas con mayor número de presencia y la posible relación entre ellas.

Los reportes anteriormente mencionados se pueden generar y visualizar en tiempo real a través de la interface web con la posibilidad de exportar los datos a hojas de cálculo convencionales.

2.4. Principales requerimientos no funcionales.

2.4.1. *Concurrencia.*

El SAI está concebido desde su origen para ser un sistema de administración centralizado en el que actualmente se da un servicio a una célula de negocio, sin embargo, a mediano plazo se contempla que su uso se expanda a nivel nacional para atender las distintas células que se encuentran en el interior del país.

Bajo el principio anterior, la concurrencia que debe mantener el SAI, está delimitada por el número de usuarios que gestionan (administradores) el alta, la modificación, cancelación, el cierre y el almacenamiento en histórico de las incidencias.

Inicialmente, el SAI se encuentra en un esquema de administración centralizada.

Dadas las características mencionadas con anterioridad, los recursos de hardware y software se encuentran mayormente disponibles en el tiempo para los solicitantes de información como lo son los reportes, las consultas habituales de un usuario, así como las del administrador, entre otras.

Se contempla que con el crecimiento y adopción del uso del SAI que todos y cada uno de los tipos de usuarios crezcan, por lo que es necesario que se cuente con

un mecanismo que garantice el acceso sin restricciones a la información y a los reportes a través de las distintas localidades que lo requieran.

El SAI contempla tener alta disponibilidad en el tiempo (24 horas/ 7 días) debido a la sensibilidad que puede ocasionar algún fallo en el software y/o hardware administrados a través de él, así como en la programación de ventanas de tiempo para mantenimiento y/o solución de incidencias.

Todas las operaciones de consulta sobre la información del SAI, deben ser atendidas de manera simultánea a cualquier otra operación, mientras que en las operaciones de edición, el primer usuario (administrador general o de componente) que inicie la edición, será quién tenga la prioridad hasta que finalice la tarea.

Dado el caso que dos operaciones de edición se soliciten al mismo tiempo, el SAI otorgará la más alta prioridad al usuario que pertenezca al grupo de administradores generales. Lo anterior debido a que el administrador general será quien tenga la visión más amplia sobre las incidencias, mientras que el administrador de componente, sólo puede delimitar el alcance de éste último.

2.4.2. *Volumen.*

Un requisito fundamental para el área de negocio y las demás áreas que dependen del SAI para la solución de sus incidencias, es que el sistema pueda procesar y almacenar archivos de gran tamaño tales como son las bitácoras o logs de cada componente administrado (Bases de Datos, Servidor, Aplicación, Software e Infraestructura).

El tamaño de los archivos mencionados puede ser desde 100MB hasta 1TB en su versión más robusta y deberán ser almacenados para su procesamiento de forma diaria, semanal y mensual.

Lo descrito previamente debe dar prioridad a la disponibilidad demandada al SAI y sin afectar de ninguna forma su uso, es decir, para cumplir con el procesamiento de las bitácoras o logs, no deben de haber interrupciones en el servicio ni grandes tiempos de respuesta, dando absoluta prioridad a la operación diaria del sistema.

Por lo anterior el SAI anexará dentro de la descripción de sus componentes el contenido de las bitácoras, sin importar su tamaño.

El objetivo de hacer uso de las bitácoras de todos los componentes administrados por el SAI, es hacer el análisis de la incidencia de una forma ágil y más eficiente sin la necesidad de salir del sistema, es decir, si se presenta una falla dentro de alguno de los componentes administrados.

Con lo anterior el SAI proporciona la funcionalidad de buscar dentro de las bitácoras del componente con la problemática, lo cual reduce la interacción con el administrador del componente y el tiempo de respuesta de las áreas administrativas otorgándole autonomía para la posible solución de incidencias.

Se requiere que el SAI pueda realizar búsquedas sobre su información con la finalidad de recabar todo lo relacionado a una incidencia, componente, log, usuario o cualquier entidad almacenada dentro del sistema.

De igual forma es relevante que se puedan realizar búsquedas específicas, es decir, búsquedas por periodos de tiempo, palabras clave o tags, número de identificación de la incidencia, entre otros parámetros.

El SAI facilitará las búsquedas sobre cualquier incidencia (siempre y cuando se tengan los permisos necesarios) o sobre una sola, no solamente sobre algún dato en específico (ej.: la descripción de la misma) sino que será posible recabar toda la información relacionada a ella: componente del SAI involucrado, personal administrativo y usuarios involucrados, áreas involucradas, fechas de creación, edición y/o cierre, bitácoras o logs, entre otra información disponible. Lo anterior sin ningún tipo de restricción en el tamaño o complejidad de la información solicitada.

Por lo anterior, el SAI será capaz de realizar búsquedas específicas y generales, para usuarios regulares o para usuarios expertos en temas computacionales o técnicos como pueden ser expresiones regulares.

2.4.3. *Transaccionalidad.*

El SAI al ser un sistema de atención a eventos y tener varios tipos de usuarios que puedan realizar modificaciones a sus elementos, no es considerado un sistema de alta transaccionalidad, sin embargo, debe mantener un mecanismo que garantice que la información sea consistente y se mantenga íntegra dentro del contexto del registro y actualización de una incidencia.

Al restringir las opciones de edición sobre incidencias a solamente los grupos de administradores (generales y de componente), dichas operaciones se estima que sean

en relación 2 a 1 contra las operaciones de consulta, por lo que el SAI debe garantizar que ambas peticiones se cumplan.

En lo que respecta a las operaciones de consulta, aunque éstas sólo pueden ser realizadas por el grupo de usuarios regulares, serán las que presenten mayor número de peticiones debido a la capacidad de generación de reportes proporcionada por el SAI.

Acorde a lo descrito con anterioridad y por la naturaleza de los eventos que son atendidos por el SAI, éste sólo debe garantizar que las peticiones de consulta se cumplan en su totalidad con un tiempo de respuesta mínimo, mientras que las peticiones de edición, deben ser atendidas en su totalidad conforme al orden en el que ésta fue solicitada.

2.5. Alcance.

Las tareas y características que el SAI desempeñará y cubrirá se encuentran enlistadas a continuación:

- El SAI es solamente un sistema de seguimiento a problemáticas de software y hardware que presenten en su operación las distintas áreas de negocio del entorno en donde sea instalado.

- Dentro de su almacenamiento, el SAI sólo registrará los elementos de software y hardware proporcionados por las áreas de negocio, no se realizará algún tipo de inventario.

- El SAI será capaz de recabar toda la información relacionada a una o un grupo de incidencias y de cualquier entidad existente en el sistema, es decir, si una incidencia está ligada a un componente, en una misma búsqueda se puede recabar la información de la incidencia y, simultáneamente, la información del componente asociado con la posibilidad de recabar las bitácoras del mismo.

- La primera fase del SAI (motivo de este trabajo) sentará las bases para que en fases futuras, el SAI explote su arquitectura y tecnologías aquí descritas.

- Los usuarios que sean agregados a algún grupo de usuarios predefinido en el SAI, será determinados por cada área de negocio.

- La clusterización o agregación de un nodo al SAI, se realizará conforme el crecimiento del volumen de la información

- La instalación y configuración del SAI contempla la instalación de la base de datos interna, puesta en marcha de interface web, integración de la información de componentes y usuarios, configuración del proceso de ingesta de bitácoras, creación de roles y privilegios para los distintos tipos de usuarios y cualquier ajuste interno del SAI.

- Se podrán integrar tantos componentes (Bases de Datos, Servidores, Aplicaciones e Infraestructura) como sean necesarios.

- El SAI será capaz de generar los siguientes tres reportes: Reporte de línea de tiempo de incidencias, reporte del promedio de tiempo de respuesta a incidencias por área y reporte de incidencias más comunes con base en etiquetas.

- Para la ingesta de las bitácoras o logs de cada uno de los componentes administrados por el SAI, se requiere que éstas sean transformadas y colocadas en un archivo de texto plano, en una ubicación lógica determinada por el administrador general del SAI o, de forma opcional para bitácoras o logs con un tamaño muy grande, comprimirlas en formato .zip, pudiendo el SAI descomprimirlas y procesarlas como cualquier otro archivo de texto plano.

- El SAI es capaz de incluir cualquier tipo de bitácoras siempre y cuando se encuentren en el formato solicitado.

- Dentro del proceso de ingesta de bitácoras, el SAI no es responsable de su transformación en el formato indicado.

- Las posibles tareas de mantenimiento, serán determinadas con base en las políticas propias de cada área de negocio, pero en la fase inicial del SAI (motivo de este trabajo) quedarán excluidas.

- Dentro de la instalación y configuración del SAI, ningún tipo de arreglo o adecuación de la red de datos disponible y sobre la que opere.

INGENIERIA
FI

Capítulo 3

Análisis y diseño de la arquitectura del sistema

3. Análisis y diseño de la arquitectura del sistema.

3.1. Sistemas empresariales multicapa.

Toda implementación de un sistema multicapa tiene como objetivo separar las operaciones que se realizan a través de él, delimitando los procesos de acuerdo a la lógica y la utilización de estas dentro del sistema.

La arquitectura mayormente utilizada es la de tres capas, en ella se dividen las tareas de presentación, la lógica de negocio y el acceso a datos. Cada una de estas áreas permite entonces procesar la información de acuerdo a la capa en la que se encuentre:

La capa de presentación se encarga únicamente de mostrar de manera visual la información a procesar. Los procesos de esta capa se encargan de mostrar al usuario la información del sistema de manera entendible y útil.

La capa de lógica de negocio valida que la información siga en todo momento las reglas y los lineamientos establecidos para un conjunto de datos determinado, así como su procesamiento.

Finalmente, la capa de acceso a datos se encarga del proceso de administración de los objetos a la base de datos o el método de almacenamiento elegido.

El objetivo de emplear la arquitectura multicapa es que el desarrollo del software se lleva a cabo en varios niveles y cuando un cambio es requerido, no es necesario

alterar toda la estructura del sistema, sino sólo la capa afectada. Lo anterior es posible debido a la delimitación específica de las tareas en cada capa.

El uso de la arquitectura multicapa mejora el mantenimiento de la aplicación y facilita el escalamiento cuando es necesario mejorar el desempeño, sin embargo separar en pocas o demasiadas capas puede agregar complejidad innecesaria y puede reducir el desempeño y la flexibilidad general.

La arquitectura multicapa facilita también la optimización del rendimiento de las capas individualmente sin afectar las capas adyacentes.

La adopción de una arquitectura multicapa puede añadir complejidad e incrementar el tiempo de desarrollo, pero con una correcta implementación se puede mejorar significativamente la sustentabilidad, extensibilidad y la flexibilidad de la aplicación.

En general, la ganancia en flexibilidad y sustentabilidad adquirida por la arquitectura multicapa es mucho mayor que la mejora marginal en rendimiento obtenida por una arquitectura que no es multicapa. (Microsoft Corporation, 2014)

Por medio de JEE (Java Enterprise Edition), se dispone entonces de un conjunto de estándares de desarrollo para la construcción de aplicaciones, tanto de escritorio como web, que permitan una integración sencilla, dinámica y transparente entre las distintas capas que componen a un sistema.

La especificación JEE incluye las herramientas necesarias para la resolución de problemas por medio de software, separando en cada capa las tecnologías necesarias disponibles para abordar dicha tarea de la manera más eficaz posible.

Las operaciones del SAI estaban contempladas inicialmente bajo el esquema relacional donde los componentes son representados por entidades y están relacionadas entre sí.

Existen actualmente múltiples mecanismos para gestionar la información desde las aplicaciones de distintas maneras, esta puede ser una simple relación de tabla a clase o de algún otro objeto, mientras que las más avanzadas permiten incluso reutilizar la misma clase para realizar las operaciones con base a un mapeo previamente configurado.

3.2. Principales frameworks.

La capa de presentación del SAI utilizara el patrón MVC (modelo-vista-controlador) por medio del framework Struts 2, de esta manera, es posible presentar en tiempo real la información contenida en la base de datos.

Por medio del patrón MVC, se pueden ligar los aspectos de la capa de negocio del SAI con la capa de modelo que Struts requiere para su funcionamiento, ya que esta capa comprende un conjunto de reglas y un flujo de mayor complejidad, de esta manera, es posible proporcionar al usuario de una aplicación web dinámica.

Struts incluye un API que permite la creación de interfaces graficas dentro de la aplicación web del SAI, mostrando un conjunto de páginas JSP (JavaServer Pages), que permiten que el usuario interactúe con el SAI.

La plataforma EJB (Enterprise JavaBeans) permite gestionar los componentes en la capa de lógica de negocio del SAI, encapsulando de manera separada el comportamiento de los procesos y los objetos.

De esta manera, la lógica de negocio separa los objetos y los procesos de acuerdo a un conjunto de reglas propias de la capa de negocio, permitiendo que la información sea procesada por un módulo intermedio entre la capa de presentación y la capa de datos, manipulándolos según sea requerido.

Utilizando el patrón de diseño DAO (Data Access Object), junto con los controladores de MongoDB para Java y la herramienta GSON, todos los objetos son en primer lugar convertidos y después procesados en la capa de datos.

Esta última capa permite la persistencia de datos dentro del SAI, evitando la necesidad que las capas de presentación y de lógica de negocio interactúen directamente con el origen de datos.

Esta capa realiza la función de intermediaria entre los orígenes de datos y la interfaz, de acuerdo a las necesidades del negocio, se puede modificar para operar de distintas maneras, permitiendo un esfuerzo menor de reprocesamiento y haciendo más flexible la persistencia de información dentro del SAI.

3.3. Diseño de sistemas orientados a entidades.

Una entidad es la representación de un objeto de la vida real y en el desarrollo de software es empleada para representar conceptualmente las características más relevantes de dicho objeto, los cuales, a su vez, tienen una identidad y un ciclo de vida.

En otras palabras, una entidad es una cosa o algo único capaz de ser cambiada continuamente en un periodo de tiempo y cuyos cambios pueden ser tan extensivos que el objeto pueda parecer muy distinto a su estado original pero que se identifica por su identidad, por lo que, se dice, que una entidad es más una entidad de comportamiento que una entidad de datos (Microsoft Corporation , 2009).

Conforme la entidad cambia, se puede rastrear cuándo, cómo y por quién fueron hechos los cambios.

Las entidades forman parte de la representación de diversos procesos de negocio cuyas actividades establecen las reglas que han de seguir tanto las entidades como el flujo de información, por lo que la interacción entre ellas es normal y lo que da forma a la lógica y procesos de negocio de un sistema.

Cuando la entidad forma parte central del desarrollo de software se debe a que se debe mantener su individualidad y el diferenciarla de todas las demás entidades dentro de un sistema, es parte fundamental de los requerimientos del mismo. (Vernon, 2013)

3.3.1. *Identificación y diseño de las entidades del SAI.*

El SAI al ser una abstracción de una realidad existente en un ambiente de tecnologías de la información de alguna área de negocio, está conformado por una serie de entidades que son necesarias para su operación y para las cuales es fundamental su correcto funcionamiento.

De acuerdo a lo descrito en capítulos anteriores en este trabajo, las principales entidades que conforman el SAI se enlistan a continuación:

i. `Incidencia.` Entidad principal que contiene los problemas o sucesos que impidan la operación adecuada de componentes y/o áreas correspondientes.

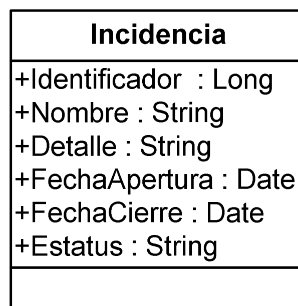


Diagrama 3-1. Entidad `Incidencia`

ii. `EventoIncidencia`. Entidad en la cual se registra cada una de las actividades realizadas sobre una incidencia y el usuario involucrado en la misma.

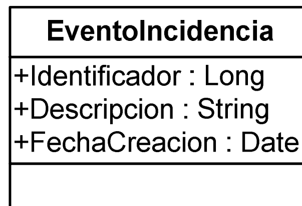


Diagrama 3-2: Entidad `EventoIncidencia`

iii. `ActividadEvento.` Entidad que describe el tipo de comportamiento que tiene un evento ligado a alguna incidencia.

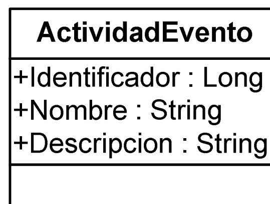


Diagrama 3-3: Entidad `ActividadEvento`

iv. `Area.` Entidad que representa la clasificación de las unidades de negocio del entorno empresarial en el cual el SAI se encuentra en operación.

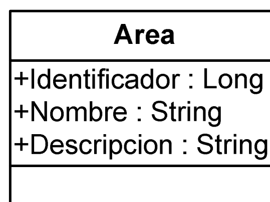


Diagrama 3-4: Entidad `Area`

v. `Usuario.` Entidad que representa a los actores dentro del SAI, los cuales llevan a cabo un conjunto de actividades de distinta importancia y funcionalidad.

Diagrama 3-5: Entidad `Usuario`

Esta entidad cuenta requiere una clasificación de roles para los distintos tipos de usuario que se encuentran en el SAI y que a continuación se muestran:

- Administrador General: es el usuario que se encarga del ciclo de vida de la incidencia, tiene acceso a todas ellas y a todos los objetos dentro del SAI, da de alta usuario y administradores de componente. Es el usuario con máximos privilegios y/o permisos dentro del SAI mostrado en el Diagrama 3-6.

Diagrama 3-6: Usuario tipo Administrador General

- Administrador de Componente: es el usuario que administra algún tipo de componente dentro del SAI. Solamente puede agregar comentarios, sugerencias e información de utilidad para la solución de las incidencias. Es el encargado de la administración de algún componente del SAI, mostrado en el Diagrama 3-7.

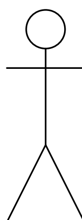


Diagrama 3-7: Usuario tipo Administrador Componente

- Usuario Regular: es el usuario que sólo puede consultar los distintos tipos de información contenida en el SAI tales como incidencias, reportes y la relación de componentes con sus administradores, mostrado en el Diagrama 3-8

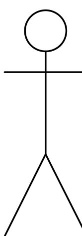


Diagrama 3-8: Usuario tipo Usuario Regular

vi. Componente. Entidad que representa los elementos administrados y monitoreados dentro del SAI y sobre los cuales las incidencias son levantadas.

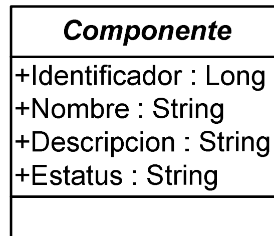


Diagrama 3-9: Entidad `Componente`

Los componentes se clasifican en los tipos que se muestran a continuación:

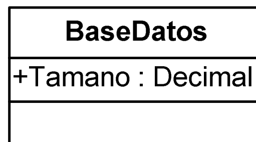


Diagrama 3-10: Subtipo `BaseDatos`

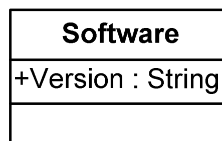


Diagrama 3-11: Subtipo `Software`

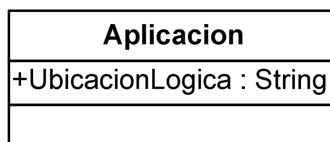


Diagrama 3-12: Entidad `Aplicacion`

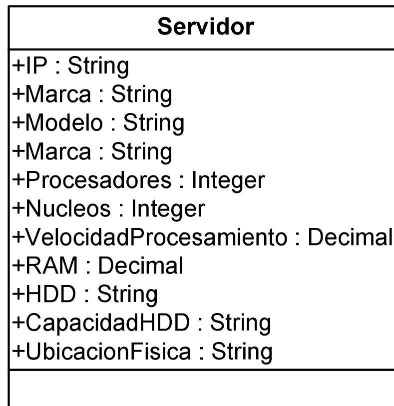


Diagrama 3-13: Entidad `Servidor`

Al igual que la entidad `Usuario` que cuenta con algunos subtipos de usuario, la entidad `Componente` cuenta con algunos subtipos que son `Servidor`, `BaseDatos`, `Software` y `Aplicacion`, los son representados en el Diagrama 3-14 y que se considera que heredan los atributos y métodos.

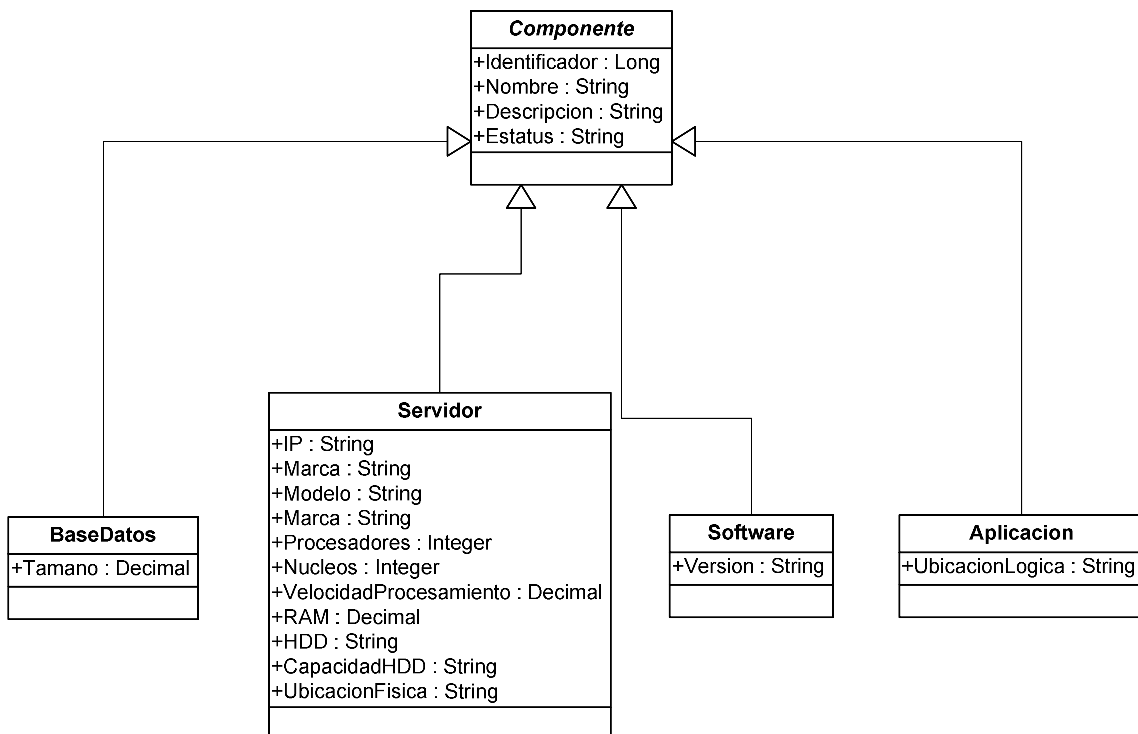


Diagrama 3-14: Generalizaciones de la entidad `Componente`

vii. `BitacoraComponente`. Entidad que representa el registro de la operación propio de cada componente existente en el SAI y que es generada de manera propietaria por el mismo. Son aquellas bitácoras que habitualmente genera cualquier software en las que se describe, con tanto detalle cómo se requiera, las acciones ejecutadas y los eventos generados en el mismo, ofreciendo información del funcionamiento en cualquier momento. Se almacenará con los fines descritos anteriormente en este trabajo.

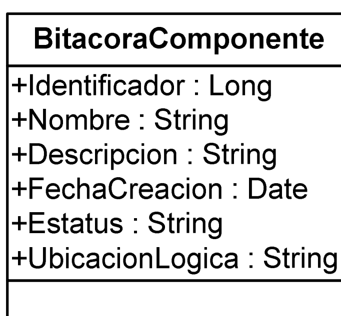


Diagrama 3-15: Entidad `BitacoraComponente`

Las entidades descritas hasta este punto fungen mayormente en los procesos de negocio en los cuales el SAI se ve involucrado, como catálogos propios del sistema debido a que representan componentes y/o actores principales del mismo.

3.3.1.1. *Identificación de relaciones entre entidades.*

Al encontrarse dentro de un sistema, las entidades del SAI interactúan de diversas formas para cumplir distintos objetivos y/o completar una parte de algún proceso, lo cual lleva a la formación de relaciones entre ellas.

Las principales relaciones entre las entidades del SAI se describen a continuación.

i. `Incidencia-EventoIncidencia`. Esta relación indica que cada vez que una incidencia sea levantada en el SAI, se creará una lista de eventos para la misma con el fin de registrar cada una de las acciones realizadas sobre una incidencia, por lo cual una incidencia puede ser relacionada a muchos eventos y un evento puede relacionarse sólo con una incidencia.

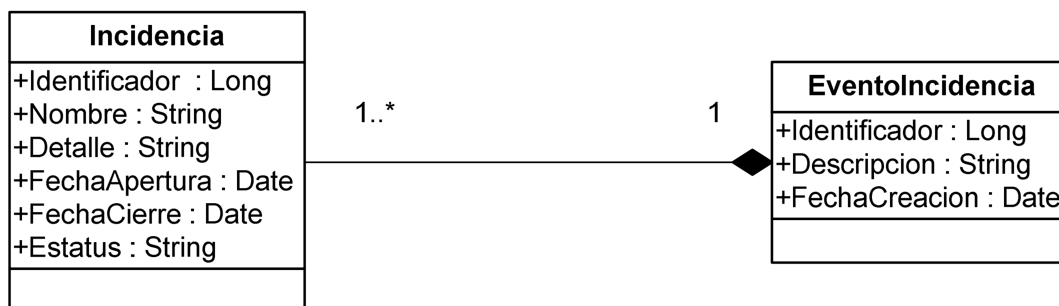


Diagrama 3-16: Relación `Incidencia-EventoIncidencia`

ii. `Incidencia-Componente`. Relación fundamental en el objetivo de negocio del SAI. Es la relación que mantiene una incidencia con el componente reportado, por lo que una incidencia debe tener un componente reportado y un componente puede tener de 1 a muchas incidencias o no tenerlas.

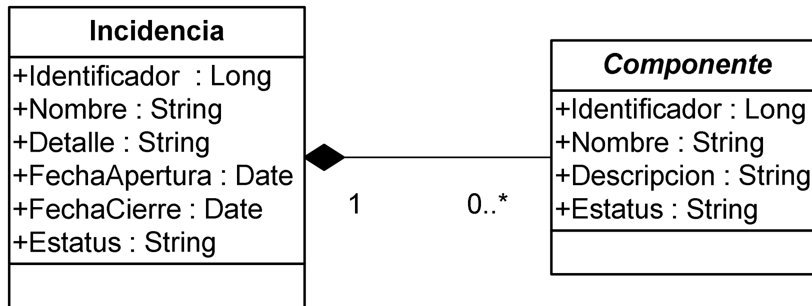


Diagrama 3-17: Relación `Incidencia-Componente`

iii. `Componente-BitacoraComponente`. Relación que implementa la integración de las bitácoras o logs de todos los componentes del SAI con los fines informativos y/o de análisis descritos con anterioridad, por lo que un componente puede tener ninguna o muchas bitácoras y una bitácora sólo pertenece a un componente (no hay bitácoras compartidas).

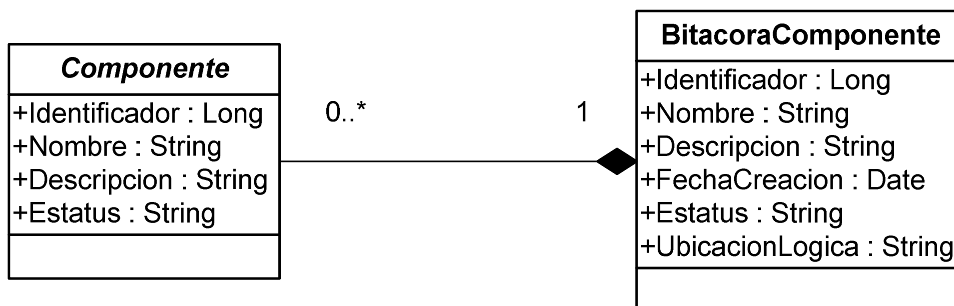


Diagrama 3-18: Relación `Componente-BitacoraComponente`

iv. `Usuario-Usuario`. Relación recursiva que indica que un usuario puede estar o no relacionado con otro usuario, el cual usualmente puede ser su jefe.

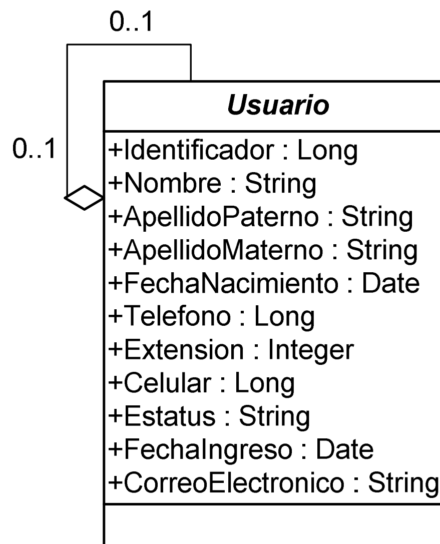


Diagrama 3-19: Relación `Usuario-Usuario`

v. `Usuario-Area`. Relación que establece que un usuario puede pertenecer a un área de negocio y que, a su vez, el área de negocio puede contener a muchos usuarios.

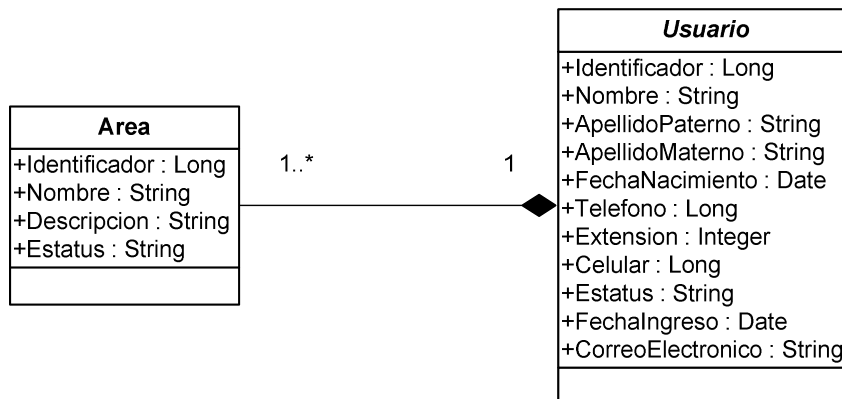


Diagrama 3-20: Relación `Usuario-Area`

vi. `Usuario-EventoIncidencia`. Relación que define que para cada evento ligado a una incidencia, se debe contar con un usuario de cualquier tipo,

mientras que un usuario puede o no encontrarse relacionado con muchos eventos de incidencias.

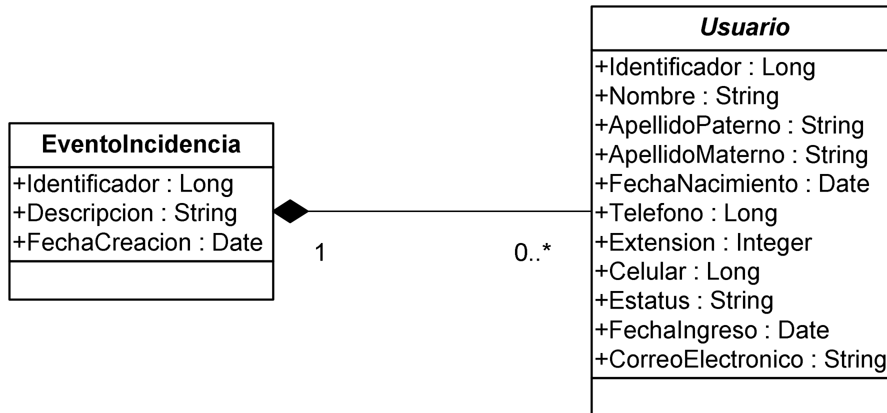


Diagrama 3-21: Relación `Usuario-EventoIncidencia`

vii. `Componente-Usuario`. Relación que denota que un componente siempre debe tener un usuario que lo administre, así como un usuario del subtipo Administrador Componente puede tener 1 o más componentes administrados por él.

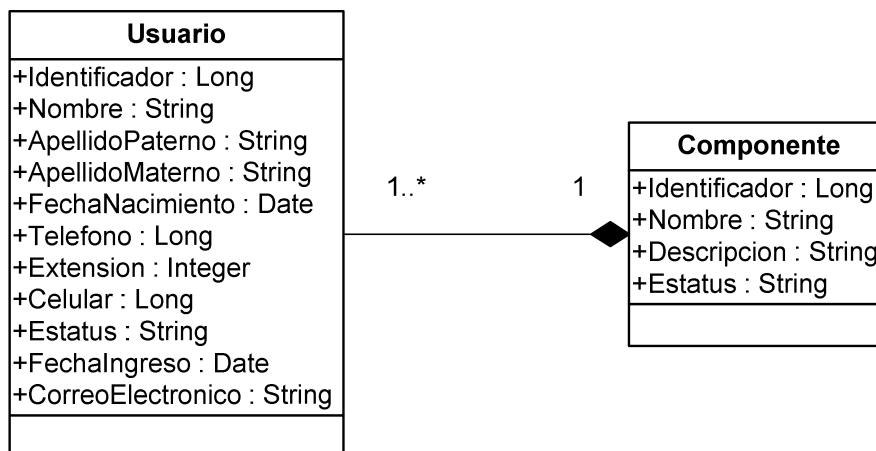


Diagrama 3-22: Relación `Componente-Usuario`

viii. `EventoIndicencia-ActividadIncidencia`. Relación a través de la que se establece el tipo de actividad que se ejerce sobre una incidencia registrada y en la cual se lleva el control de todas las operaciones realizadas sobre ella.

Diagrama 3-23: Relación `EventoIncidencia-ActividadIncidencia`

En el Diagrama 3-24 se muestra de forma general cómo se encuentran relacionadas todas las entidades del SAI.

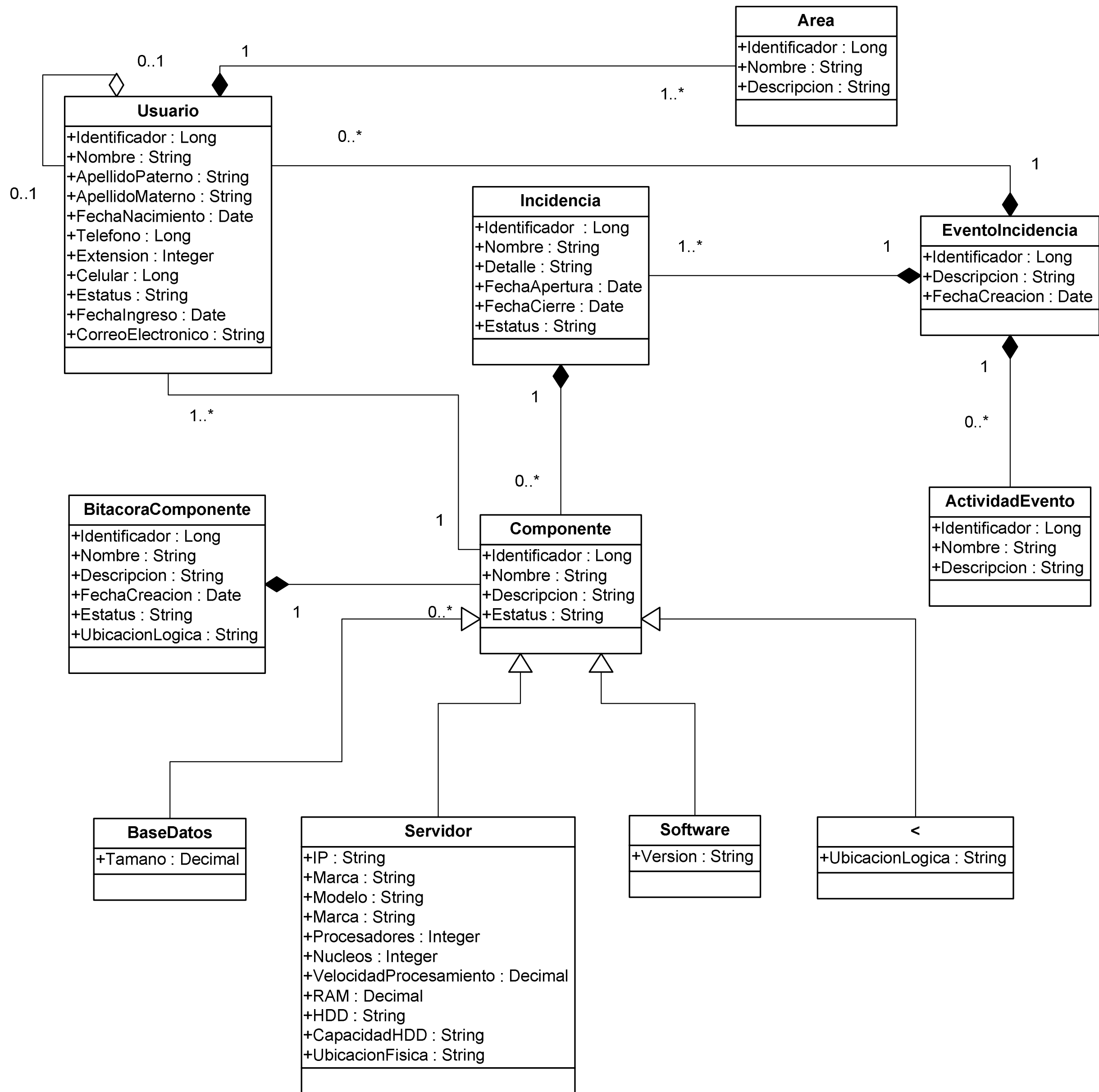


Diagrama 3-24. Visión general de las relaciones entre entidades del SAI.

3.4. Diseño de sistemas orientadas al desarrollo de interfaces.

3.4.1. *Diseño de interfaces para el SAI.*

El SAI al ser un sistema que tiene una alta interacción con usuarios, cuyos roles fueron definidos con anterioridad, lleva a cabo una serie de acciones básicas para su operación y con las cuales interactúa con sus componentes, mientras que estas ofrecen un conjuntos de funcionalidades para sus usuarios.

En el Diagrama 3-25 se visualizan las principales tareas que los usuarios realizan dentro del SAI, las cuales pueden catalogarse de acuerdo a la acción que realizan y a las entidades con las que se interactúa; dichas tareas se pueden clasificar de la siguiente forma:

i. Tareas de Incidencia.

ii. Tareas de Componente.

iii. Tarea de Reporte.

Por lo anterior, en el Diagrama 3-25 se muestran las tareas mencionadas como Casos de Uso representativos del SAI y que competen al alcance de este trabajo.

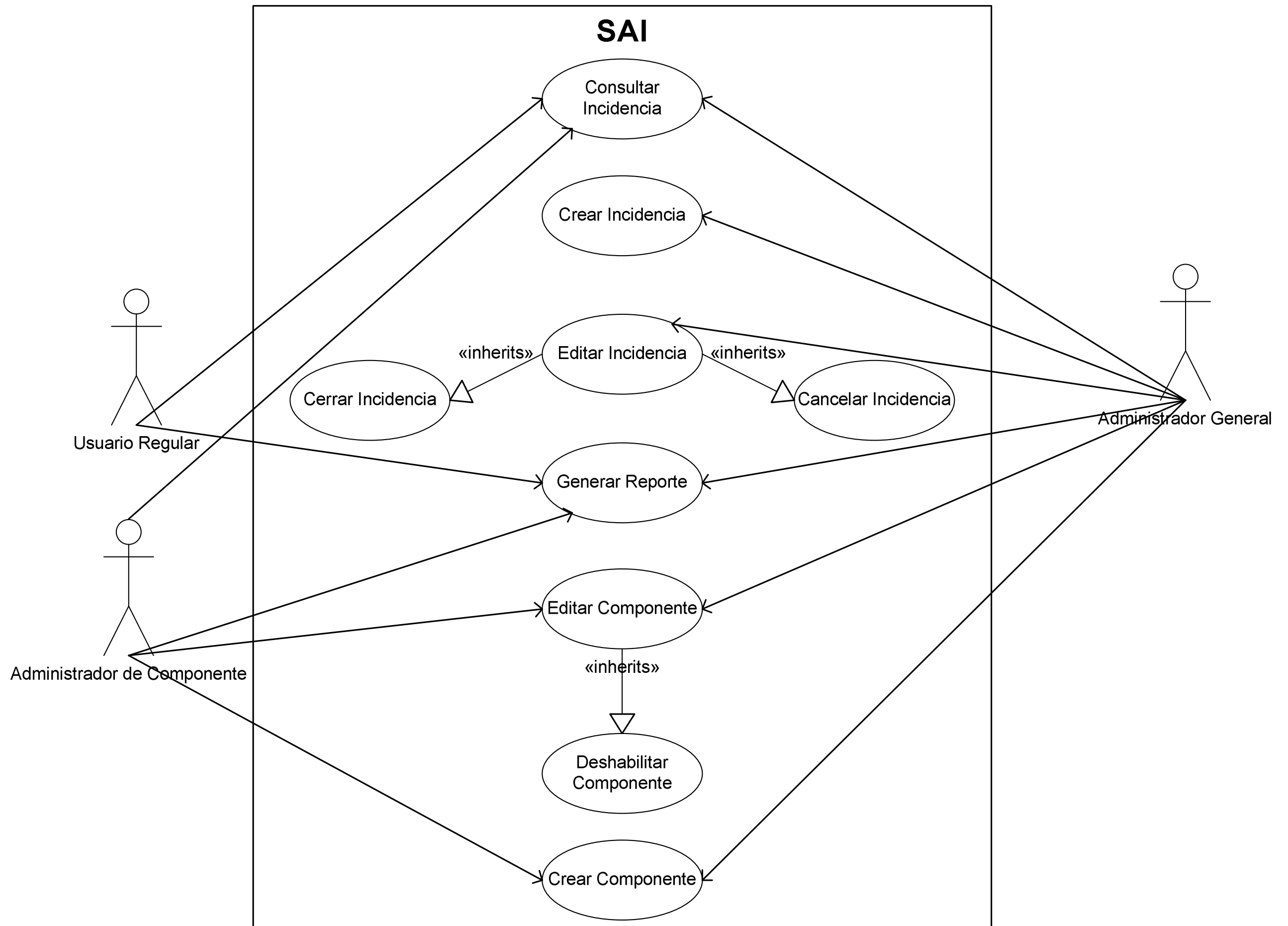


Diagrama 3-25: Casos de uso generales del SAI

Nombre:	**Crear Incidencia**
Resumen:	Se crea una incidencia
Actores:	Usuario Regular y/o Administrador General
Descripción:	El caso de uso puede iniciar de dos formas: el Usuario Regular reporta una incidencia al Administrador General y éste crea la incidencia o el Administrador General puede crear la incidencia directamente. Al momento en que se le indica al SAI que se crea una incidencia, el sistema crea internamente un registro nuevo que detalla el tipo de evento que se está realizando a la incidencia. Si la incidencia tiene una creación exitosa, devuelve mensajes de éxito al Administrador General y éste, a su vez, al Usuario Regular en caso de que se haya reportado la incidencia en esa secuencia.
Figura representativa	Diagrama de Secuencia 3-21

Tabla 3-1: Descripción de Caso de Uso Crear Incidencia.

Nombre:	**Editar Incidencia**
Resumen:	Una incidencia existente se modifica.
Actores:	Administrador General
Descripción:	El caso de uso inicia cuando el Administrador General accede a una incidencia para realizar alguna modificación. Este caso de uso funge como “padre” de los distintos tipos de edición que puede sufrir una incidencia, por lo que sus acciones son genéricas. Los casos de uso Cancelar Incidencia y Cerrar Incidencia son los subtipos del caso de uso Editar Incidencia, los cuales pueden ser iniciados únicamente por el Administrador General. Al momento de iniciar algunos de los distintos tipos de caso de uso de Editar Incidencia, se inicia un registro nuevo en que se detalla el tipo de actividad que fue ejecutada sobre la incidencia. Los casos de uso Cancelar Incidencia y Cerrar Incidencia inician únicamente devuelven un mensaje de éxito cuando la incidencia ha sido cancelada o cerrada satisfactoriamente.
Figura representativa	Diagrama de Secuencia 3-2

Tabla 3-2: Descripción de Caso de Uso Editar Incidencia.

Nombre:	**Consultar Incidencia**
Resumen:	Una incidencia existente se consulta.
Actores:	Usuario Regular, Administrador de Componente o Administrador General
Descripción:	El caso de uso inicia cuando cualquiera de los actores del sistema, requiere recuperar la información ligada a una incidencia especificando algunos parámetros al sistema para delimitar los resultados esperados.
Figura representativa	Diagrama de Secuencia 3-3

Tabla 3-3: Descripción de Caso de Uso Consultar Incidencia.

Nombre:	**Consultar Reportes**
Resumen:	Se genera un reporte nuevo de los establecidos
Actores:	Usuario Regular, Administrador de Componente o Administrador General
Descripción:	El caso de uso inicia cuando cualquiera de los actores del sistema, requiere generar algún reporte de los disponibles en el SAI. A su vez, el sistema inicia el proceso de recuperación de la información por medio de los parámetros especificados en el tipo reporte elegido.
Figura representativa	Diagrama de Secuencia 3-4

Tabla 3-4: Descripción de Caso de Uso Consultar Reportes.

Nombre:	**Crear Componente**
Resumen:	Se crea un componente nuevo en el SAI
Actores:	Administrador de Componente o Administrador General
Descripción:	El caso de uso inicia cuando un Administrador General o Administrador de Componente requieren dar de alta algún nuevo componente dentro del sistema. El sistema, a su vez, permite la adición de alguna bitácora o log al componente, invocando un proceso interno que liga los posibles archivos de bitácora a sus respectivos componentes.
Figura representativa	Diagrama de Secuencia 3-5

Tabla 3-5: Descripción de Caso de Uso Crear Componente.

Nombre:	**Editar Componente**
Resumen:	Caso de Uso que atiende las modificaciones generales que puede sufrir algún componente del SAI.
Actores:	Administrador de Componente o Administrador General
Descripción:	El caso de Uso inicia cuando un Administrador de Componente o Administrador General requieren establecer alguna nueva configuración de los componentes. El caso de Uso Deshabilitar Componente es un caso específico del caso de uso Editar Componente al tomar una deshabilitación de componente como un tipo especial de edición al dar de baja lógicamente un componente.
Figura representativa	Diagrama de Secuencia 3-6

Tabla 3-6: Descripción de Caso de Uso Editar Componente.

Diagramas de Secuencia del SAI.

Los diagramas de secuencia del SAI indican cómo el usuario interactúa de forma general con tareas específicas con el entorno del sistema, además de indicar en forma general cómo el SAI debe responder ante acciones específicas ejecutadas por el usuario.

A continuación se muestran las tareas más representativas del SAI y sus interacciones con los usuarios involucrados.

El Diagrama de Secuencia 3-1 muestra la primera operación básica del SAI: el Alta de Incidencia.

En el mencionado diagrama de secuencia se puede observar la interacción que tienen los actores del sistema con los distintas interfaces que lo conforman como `IncidenciaService` y `EventoIncidenciaService`.

Las interfaces mencionadas con anterioridad, son las encargadas de atender y dar respuesta a cada una de las solicitudes de los actores de acuerdo a la lógica que estas deben seguir y que se establece en el flujo y proceso de la información para cada uno de los casos.

Así también, cada interaccción entre los actores del SAI y las interfaces de servicio, emiten algún tipo de mensaje o solicitar alguna otra acción a otro módulo o interface del sistema hasta que las solicitudes del actor para la operación sean finalizadas o que la acción ejecutada sea finalizada.

El anterior, es el comportamiento general que se ilustra en los siguientes diagramas de secuencia que representan las operaciones básicas y más relevantes del SAI, los cuales tienen su respectiva tabla de descripción previamente detalladas.

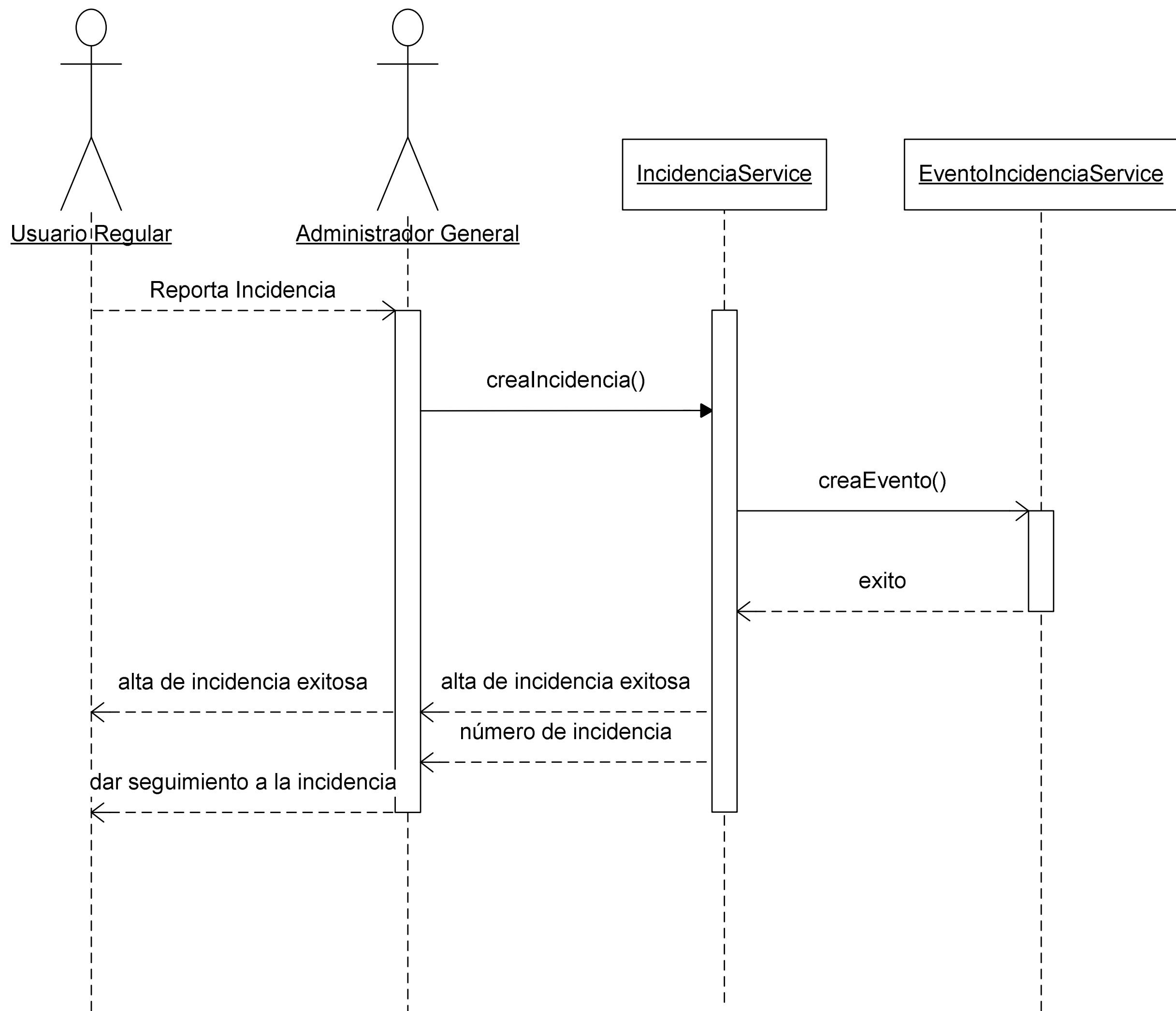


Diagrama de Secuencia 3-1: Alta de Incidencia.

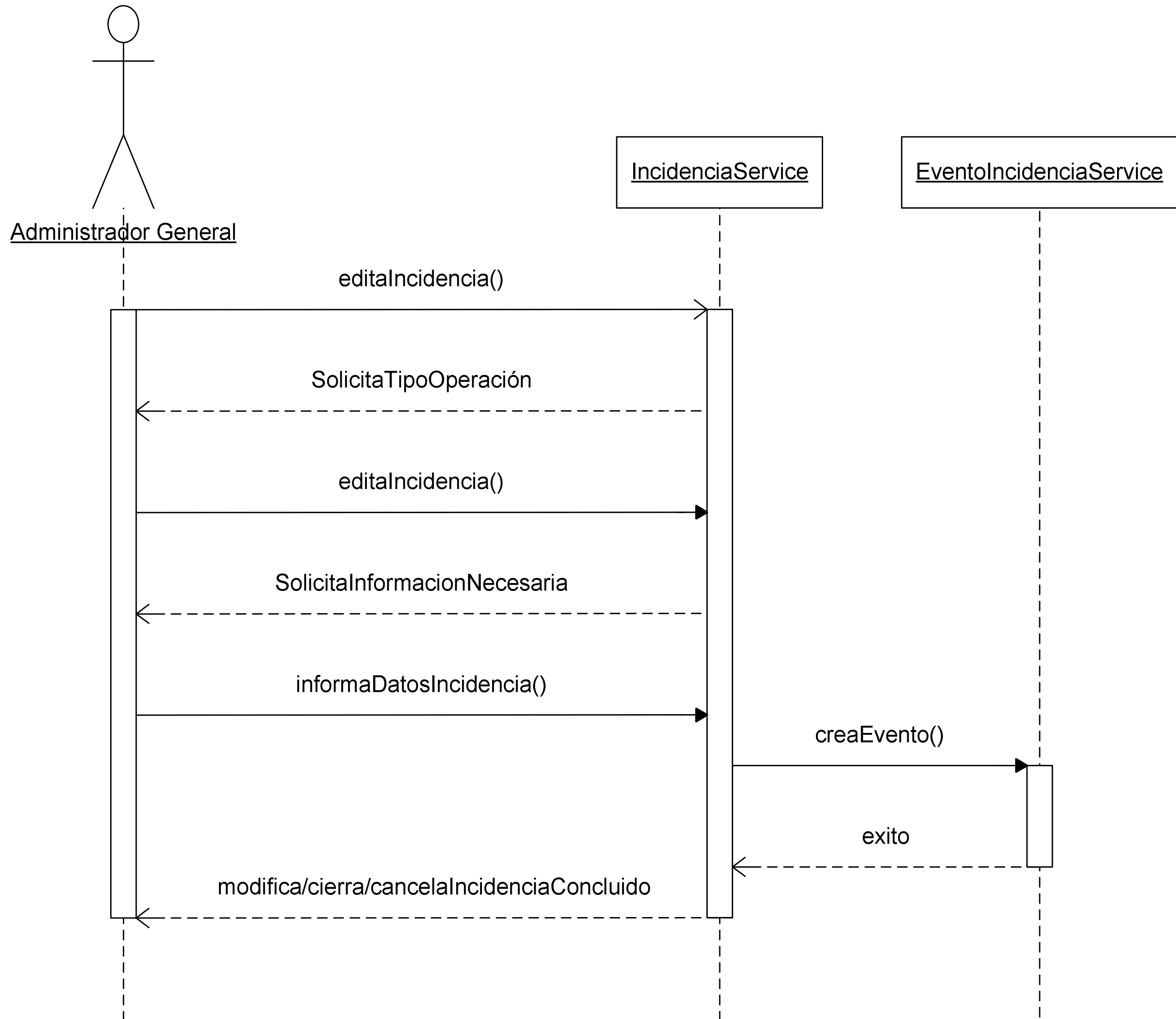


Diagrama de Secuencia 3-2: Edición de Incidencia.

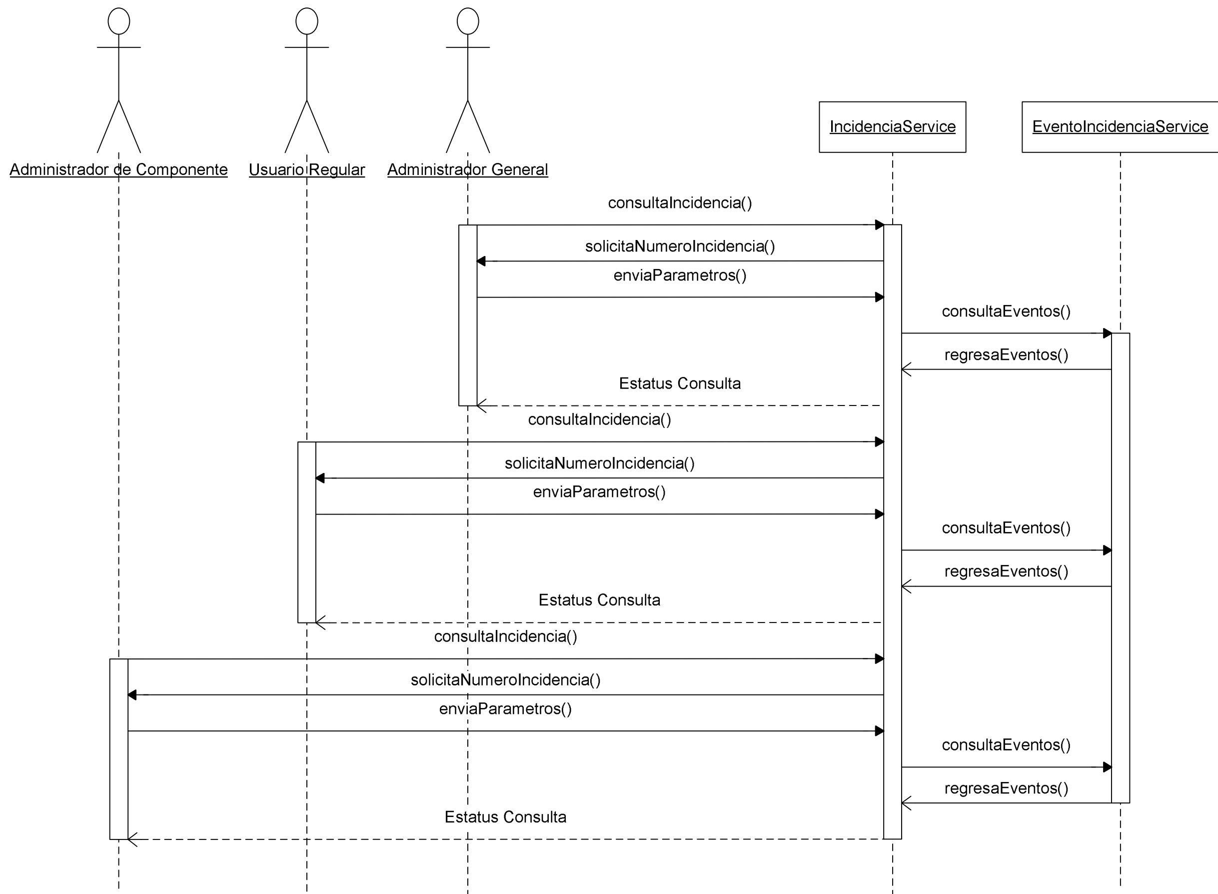


Diagrama de Secuencia 3-3: Consulta de Incidencia.

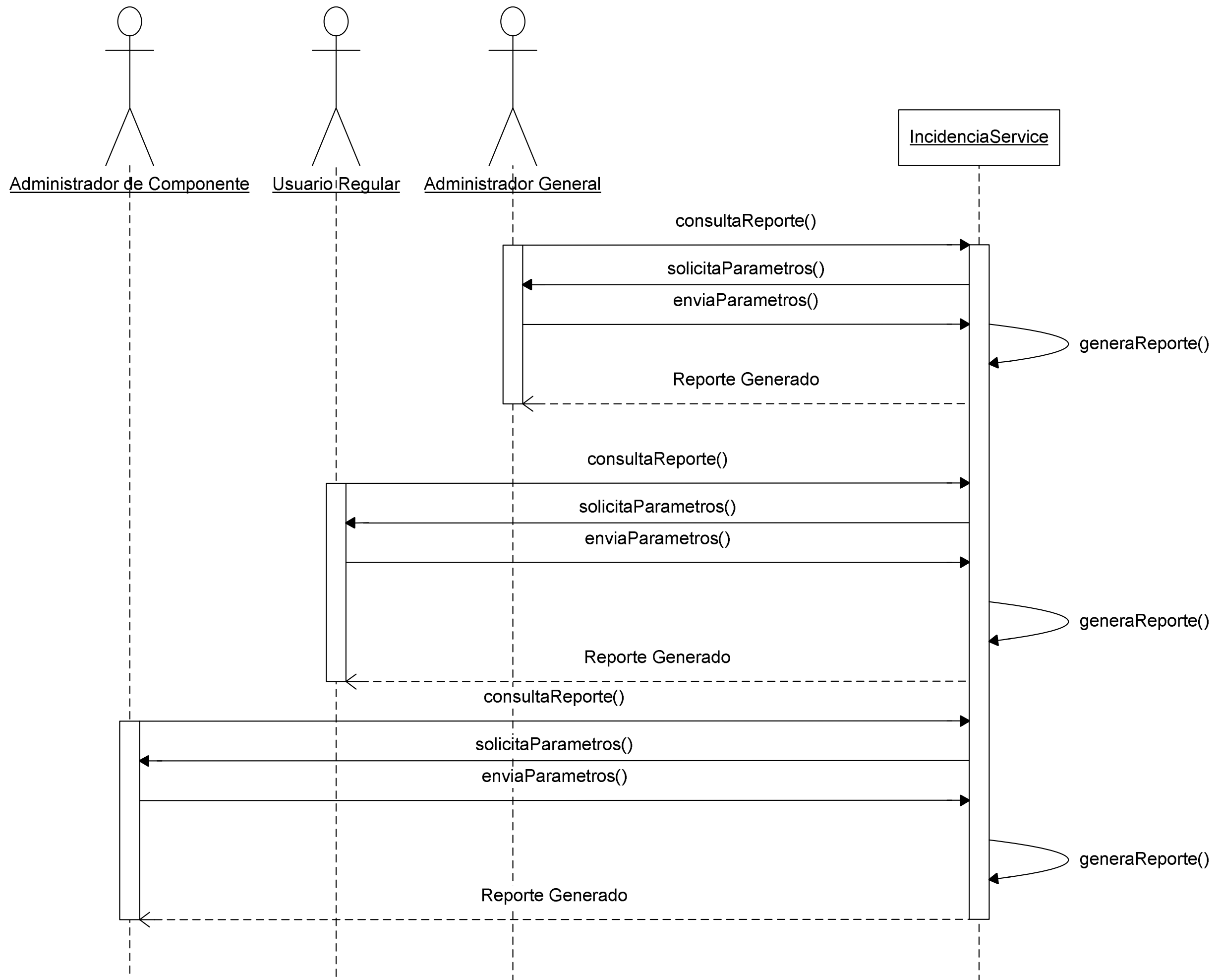


Diagrama de Secuencia 3-4: Consulta de Reportes.

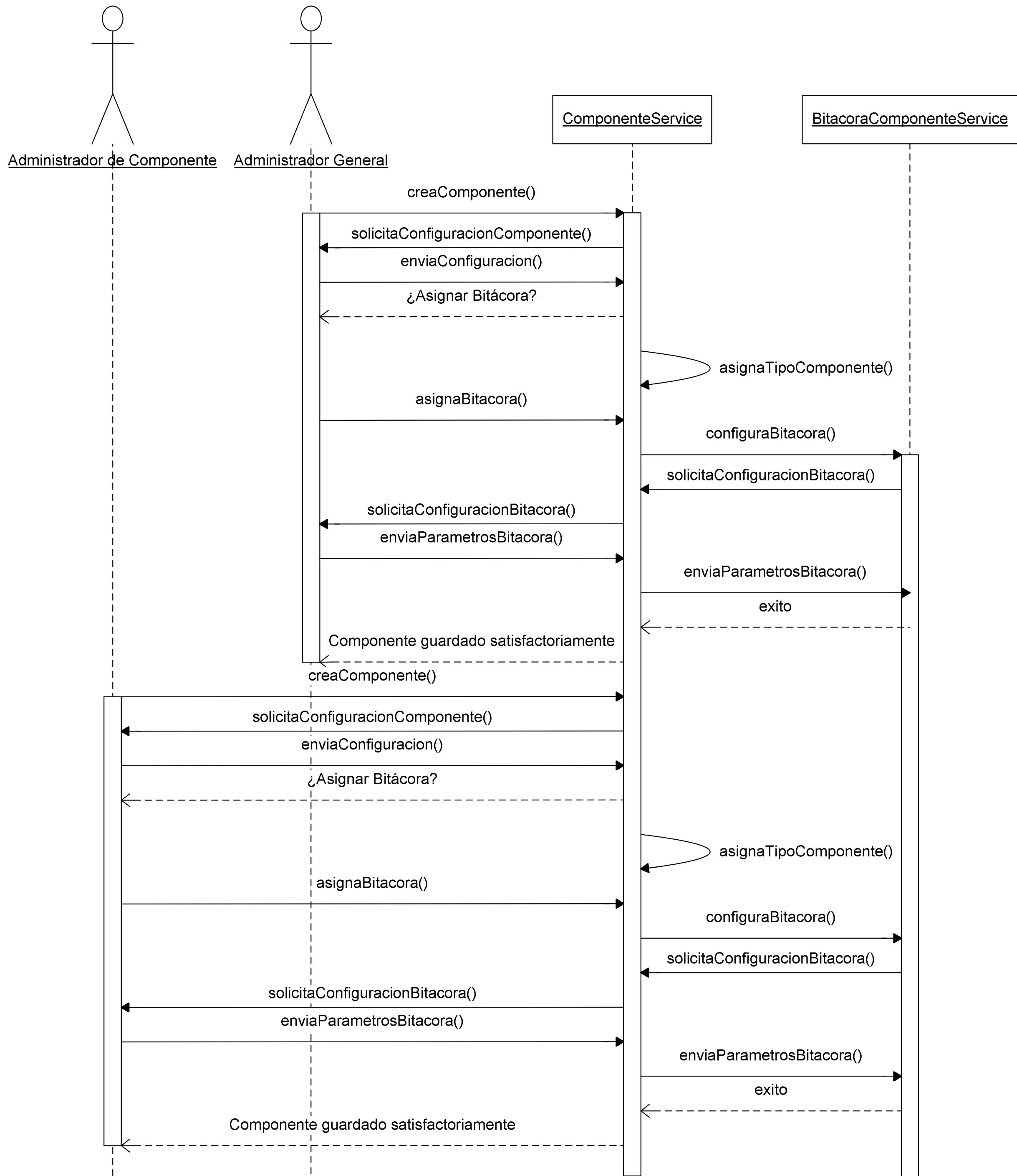


Diagrama de Secuencia 3-5: Alta de Nuevo Componente.

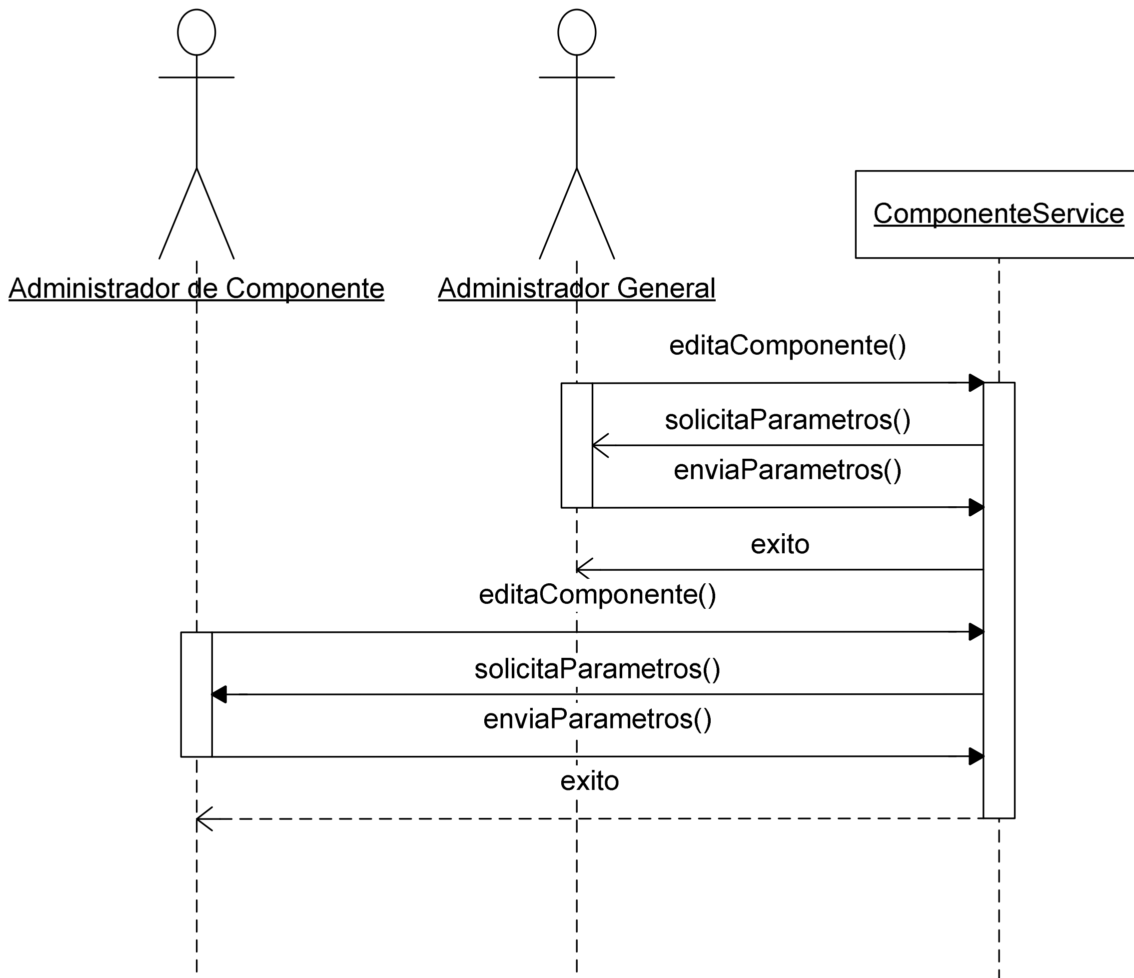


Diagrama de Secuencia 3-6: Edición de Componente

3.5. Diseño de la capa de acceso a datos.

La capa de acceso a datos e información almacenados en el SAI es la intermediaria entre las acciones solicitadas por el usuario (una vez procesadas por la capa superior donde residen todas las reglas de negocio que rigen al SAI) y el almacén de datos, por lo que una parte del desempeño del sistema reside en ella.

Es importante mencionar que los niveles de profundidad, así como la complejidad al implementar la capa de persistencia, dependen de varios factores como

son la lógica del negocio, la complejidad de la abstracción del modelo de datos, los esquemas de transaccionalidad y la granularidad de datos, entre otros.

El objetivo de la capa de acceso a datos es crear una relación entre los objetos del sistema y los objetos de la base de datos, de manera que puedan ser manipulados por la aplicación, para que la interacción entre el usuario y la información almacenada cuente con una capa intermedia que permita realizar la administración de datos de una manera sencilla y sin que la interacción se realice directamente sobre la base.

La finalidad de la capa de acceso a datos en cualquier sistema es facilitar a las bases de datos o sistemas de almacenamiento el aseguramiento de la integridad de la información en todo el sistema, como lo es el SAI.

Es pertinente recordar que toda la interacción entre capas se realiza a través de las interfaces que ejecutan servicios con la finalidad de distribuir sólo la información necesaria para cada capa, es decir, la capa de acceso a datos no conoce qué ejecuta la capa de negocio sino que sólo recibe y realiza peticiones a través de las interfaces los datos que almacenará en fuente de datos.

3.5.1. *Modelo de datos basado en agregaciones.*

Los modelos relacionales manipulan la información dentro de una tabla de forma bidimensional, a través de filas y columnas, esta representación de la información en últimas versiones, permite almacenar tipos de datos con mayor complejidad, como BLOBs (objetos binarios de gran tamaño), XML, entre otros.

Lo anterior impacta de manera drástica en el modelo, puesto que es necesario almacenar cierta información en distintas entidades para después relacionarla de múltiples maneras, lo que afecta el desempeño conforme el volumen de la información aumenta.

De acuerdo a las necesidades del negocio se crean entidades que se relacionan entre sí, esto permite realizar operaciones entre los distintos objetos en la base de datos de manera que las características más importantes están representadas y sus atributos adecuadamente separados entre sí, evitando la duplicación de datos cuando se trata de un modelo relacional normalizado.

NoSQL aborda esta limitante con las agregaciones, las cuales consisten en grupos de datos anidados que permiten a un elemento almacenar información más compleja, sin limitarse a cierto tipo de datos para una columna, como lo es el almacenamiento de estructuras.

En NoSQL no existe una restricción estructural para cada documento, es decir, que la estructura individual de cada documento puede ser adaptada a las necesidades específicas que se presenten pudiendo ser modificada en cualquier momento sin que los documentos existentes o futuros se vean afectados o requieran el mismo cambio.

Un modelo NoSQL, de manera distinta al modelo relacional, debe crearse con base en cómo se manipula esta información y en menor medida a como está constituida, es decir, un modelo de este tipo no se rige por sus entidades y relaciones, sino por una abstracción de estos; por lo tanto, una agregación de entidades conforman

una nueva entidad de datos estructurados partiendo de uno o más conjuntos de datos con cierta relación entre sí.

Se tiene entonces que el modelo es creado a partir de múltiples entidades y sus relaciones con otras entidades, haciendo que la nueva entidad contenga no sólo los atributos propios de una sola, sino que puede contener los de aquellas entidades con las cuales se relacionan, disminuyendo el número de objetos en el modelo e incorporando los atributos de las demás entidades.

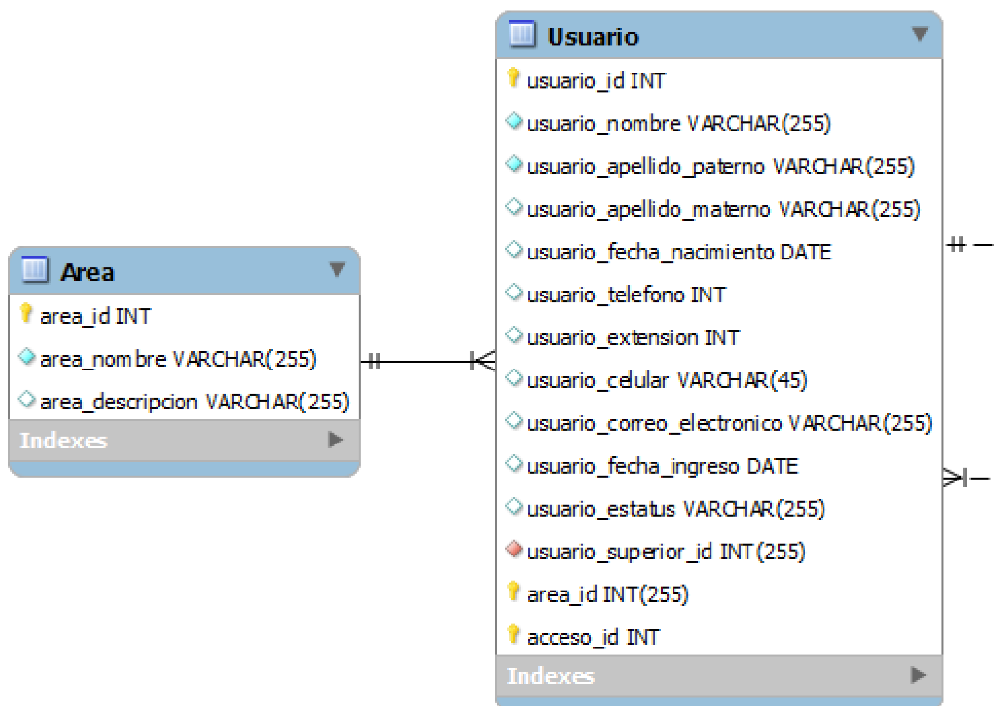
Al incorporar información entre entidades, se permite que la información sea redundante, según sean las necesidades de negocio y las funcionalidades que otorguen las entidades, pueden beneficiar o perjudicar al sistema. (Sadalage, 2013)

3.5.2. *Proceso de conversión del modelo relacional.*

Partiendo de la entidad básica del SAI llamada `Usuario` que debe pertenecer a una entidad `Area`, la relación entre ellas se ve representada en el siguiente Modelo Entidad-Relación 3-1.

De manera que varios usuarios pertenecen a un área, y cada usuario puede pertenecer a una única área a la vez.

En una consulta simple, donde se requieren conocer los usuarios y el área a las que pertenecen, es necesario consultar estos por su relación con la llave ***identificadorArea*** y sin esta llave no sería posible relacionarlas.



Modelo Entidad-Relación 3-1: Relación entre entidades `Usuario-Area`

La representación en JSON de estas clases se muestra a continuación:

```
usuario:
{
    identificador:null,
    nombre:null,
    apellidoPaterno:null,
    apellidoMaterno:null,
    fechaNacimiento:null,
    telefono:null,
    extension:null,
    celular:null,
    estatus:false,
    fechaIngreso:null,
    identificadorJefe:null,
    identificadorArea:null
}
```

```
area:
{
    identificadorArea:null,
    nombreArea:null,
    descripcionArea:null,
    estatusArea:false
}
```

Documento JSON 3-1: Documentos `usuario` y `area`

En el modelo basado en agregaciones, la entidad `Usuario` permitirá entonces contener a la entidad `Area` y se convierte en uno de los atributos, dada la relación que existe entre ellos por la llave ***identificadorArea***.

El Documento JSON 3-2, muestra la representación por agregación de una relación simple, sin embargo, el número y el tipo de relaciones entre entidades repercuten en el diseño del modelo de agregaciones.

Otras entidades presentan un relacionamiento más complejo, el cual puede ser abordado de distintas maneras en el modelo de agregaciones, así como la entidad `Usuario` pertenece a una entidad `Area` y al final agrega los atributos de esta.

Sin embargo, una entidad `Incidencia` presenta un conjunto de entidades `EventoIncidencia`, las cuales representan el registro de actividades que se han realizado con respecto a una Incidencia en específico, por lo tanto, una Incidencia está compuesta de un conjunto de estas.

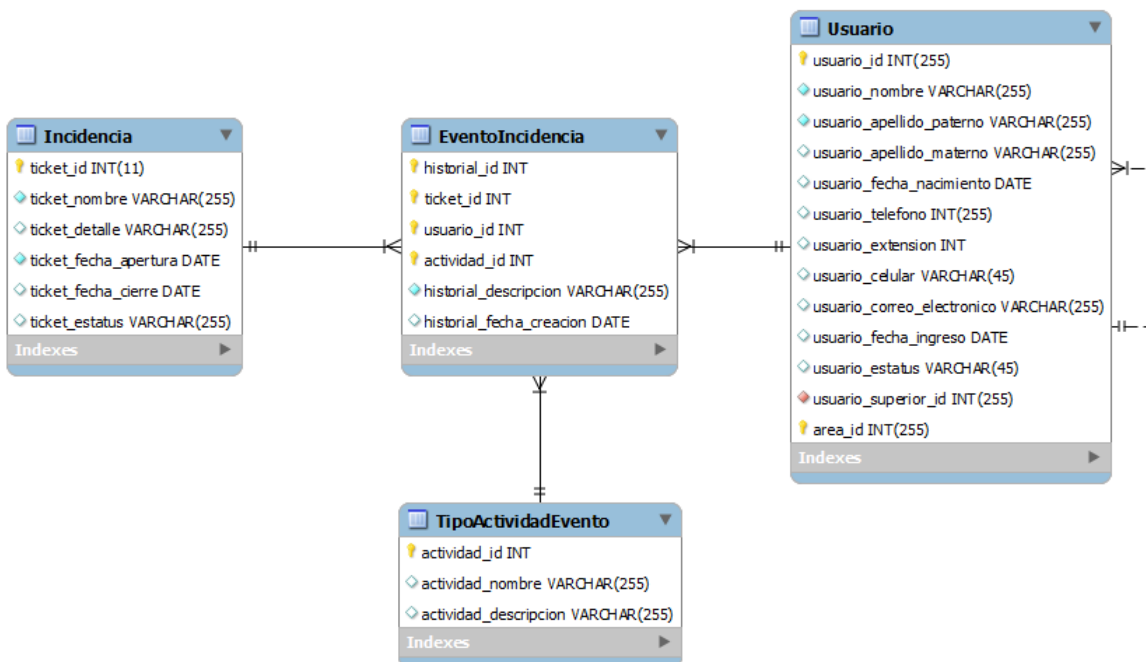
Además de este relacionamiento, `EventoIncidencia` está relacionado con dos entidades más, `Usuario`, la cual especifica el actor encargado del evento de acuerdo al tipo de actividad, especificado por `ActividadEvento`.

Las entidades quedan representadas como se muestran en el Modelo Entidad-Relación 3-2.

```
usuario:
{
    identificador:null,
    nombre:null,
    apellidoPaterno:null,
    apellidoMaterno:null,
    fechaNacimiento:null,
    telefono:null,
    extension:null,
    celular:null,
    estatus:false,
    fechaIngreso:null,
    identificadorJefe:null,
    area:
    {
        nombreArea:null,
        descripcionArea:null,
        estatusArea:false
    }
}
```

Documento JSON 3-2: Agregación de entidades `usuario` y `area`

De acuerdo al modelo basado en agregaciones realizado con anterioridad entre `Usuario` y `Área`, las relaciones de `EventoIncidencia` con `Usuario` y `TipoActividadEvento` definen que la primera puede heredar los atributos de las últimas. La representación de cada entidad en JSON se muestra en Documento JSON 3-3.



Modelo Entidad-Relación 3-2: Relacionamiento entre `Incidencia` y `EventoIncidencia`

```
usuario:
{
    identificador:null,
    nombre:null,
    apellidoPaterno:null,
    apellidoMaterno:null,
    fechaNacimiento:null,
    telefono:null,
    extension:null,
    celular:null,
    correoElectronico:null,
    estatus:null,
    fechaIngreso:null,
    identificadorJefe:null,
    identificadorArea:null,
    area:
    {
        identificadorArea:null,
        nombreArea:null,
        descripcionArea:null,
        estatusArea:null
    }
}
```

```
tipoActividad:
{
    identificador:null,
    nombre:null,
    descripcion:null
}
```

```
eventoIncidencia
{
    identificador:null,
    descripcion:null,
    fechaCreacion:null,
    identificadorIncidencia:null,
    identificadorUsuario:null,
    identificadorTipoActividad:null
}
```

Documento JSON 3-3: Documentos Usuario, `TipoActividad` y EventoIncidencia

Una entidad EventoIncidencia requiere de un solo elemento Usuario y un solo elemento TipoActividadEvento, por lo que sus atributos pasarían a formar parte de EventoIncidencia como se muestra a continuación:

```
eventoIncidencia
{
    identificador:null,
    descripcion:null,
    fechaCreacion:null,
    identificadorIncidencia:null,
    usuario:
    {
        identificadorUsuario:null,
        nombre:null,
        correoElectronico:null,
    },
    tipoActividad:
    {
        nombre:null,
        descripcion:null
    }
}
```

Documento JSON 3-4: Agregación para EventoIncidencia.

De esta forma los atributos de las entidades Usuario y TipoActividadEvento forman parte de los atributos de EventoIncidencia, si bien, en el caso de Usuario, solo los atributos más relevantes para la entidad EventoIncidencia se agregan, debido a que la entidad Usuario existe de manera independiente dentro del modelo. Más adelante se detalla el porqué de esta decisión.

A continuación se tienen dos entidades Incidencia y EventoIncidencia, de las cuales Incidencia tiene una relación uno a muchos.

Durante el proceso de gestión de las incidencias, existirán múltiples eventos que registran la actividad realizada sobre un elemento Incidencia, por lo cual, el

conjunto de elementos `EventoIncidencia` que la definen, pasan a formar parte de esta.

La representación básica JSON de la clase Incidencia es la siguiente:

```
incidencia:
{
    identificador:null,
    nombre:null,
    detalle:null,
    fechaApertura:null,
    fechaCierre:null,
    estatus:false,
    identificadorComponente:null
}
```

Documento JSON 3-5: Documento de incidencia.

Cada `EventoIncidencia` corresponde a un elemento `Incidencia`, por lo que existen uno más registros que trabajan sobre un mismo elemento `Incidencia`.

Por medio de las agregaciones, es posible que de los atributos a agregar a una entidad se almacenen en una estructura, permitiendo mayor flexibilidad sin la necesidad de relacionar las entidades, ya que todos los elementos forman parte de una estructura en esta.

En el Documento JSON 3-6 se muestra cómo es la agregación para el caso descrito anteriormente.

De acuerdo al modelo de agregación generado para la entidad Incidencia, esta contiene uno o más elementos `EventoIncidencia`, según se requieran, de esta manera, cada vez que se genera un nuevo evento sobre alguna Incidencia, se modifica

el conjunto de elementos `EventoIncidencia` dentro de esta, sin que los demás atributos de `Incidencia` sean alterados.

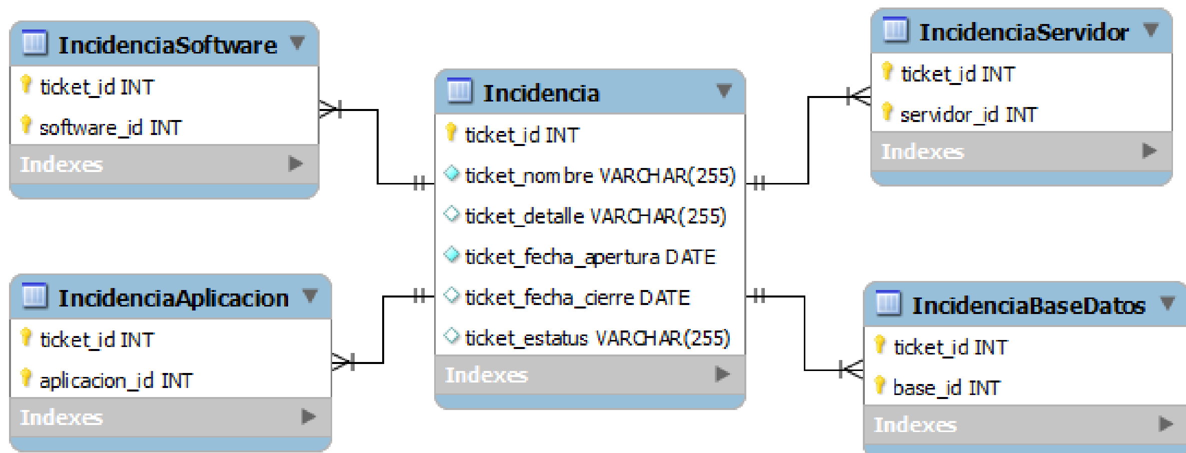
En el Documento JSON 3-6 se muestra el uso de una lista de objetos `EventoIncidencia`.

```
incidencia:
{
    identificador:null,
    nombre:null,
    detalle:null,
    fechaApertura:null,
    fechaCierre:null,
    estatus:null,
    identificadorComponente:null,
    eventoIncidencia:
    [{
        identificador:null,
        descripcion:null,
        fechaCreacion:null,
        identificadorIncidencia:null,
        usuario:
        {
            identificadorUsuario:null,
            nombre:null,
            correoElectronico:null,
        },
        tipoActividad:
        {
            nombre:null,
            descripcion:null
        }
    },
    {
        ...
    }]
}
```

Documento JSON 3-6: Agregación para incidencia.

Además de cada EventoIncidencia que se presenta en Incidencia, se tiene que esta se relaciona con los componentes de las cuatro áreas dentro del SAI, y sus respectivas entidades, estas son: Aplicacion, BaseDatos, Servidor y Software. Esta relación se implementa a través de una entidad débil con IncidenciaComponente que se genera para a cada una: IncidenciaAplicacion, IncidenciaBaseDatos, IncidenciaServidor e IncidenciaSoftware.

A su vez, cada una de estas entidades se relaciona con su respectivo componente. Esta relación permite a una Incidencia asignarle únicamente un Componente, por medio de la entidad, sin embargo, la lógica de cada una de las entidades que presentan incidencias es idéntica, la relación entre estas entidades es la siguiente:



Modelo Entidad-Relación 3-3: Relacionamiento entre entidades Incidencia e IncidenciaComponente.

La relación de cada `Incidencia` con su respectivo componente se constituye por medio de su identificador, por lo que el único atributo importante para este relacionamiento está dado por él mismo.

Finalmente la entidad `Incidencia` se representa de la siguiente forma:

```
incidencia:
{
    identificador:null,
    nombre:null,
    detalle:null,
    fechaApertura:null,
    fechaCierre:null,
    estatus:null,
    eventoIncidencia:
    [{
        ...
    }],
    aplicacion:
    {
        identificadorAplicacion:null,
        aplicacionNombre:null
    },
    baseDatos:
    {
        identificadorBaseDatos:null,
        baseDatosNombre:null
    },
    servidor:
    {
        identificadorServidor:null,
        servidorNombre:null
    },
    software:
    {
        identificadorSoftware:null,
        softwareNombre:null
    }
}
```

Documento JSON 3-7: Agregación final para la entidad `Incidencia`.

Es importante aclarar, que aunque todos los atributos de los demás componentes están especificados dentro de la nueva estructura de `Incidencia`, solo será permitido uno de ellos, es decir, si la `Incidencia` corresponde a una `Aplicacion`, se colocaran únicamente los atributos identificador y nombre, mientras que los demás elementos pueden omitirse en su totalidad o declararse como nulos, evitando así discrepancias con la información almacenada.

De igual manera que sucede con `EventoIncidencia` y `Usuario`, únicamente serán utilizados los atributos más relevantes de la entidad `Componente`, ya que las entidades `Incidencia` y `Aplicacion` pueden existir, si bien relacionadas, el modelo de agregación no requiere que una de estas sea reemplazada o contenida dentro de la otra.

3.5.3. *Capacidades del modelo basado en agregaciones.*

Partiendo de la primicia del modelado basado en agregaciones, el cual permite componer una entidad a partir de otras que se encuentran relacionadas con esta, es que la información ahora es contenida dentro de un solo elemento compuesto.

Al agregar una o más entidades, se deja de depender de llaves y entidades débiles, por lo que en primera instancia, las búsquedas pueden realizarse con mayor eficiencia, diferenciándose de un modelo relacional, cuya información puede representarse únicamente de manera bidimensional, la agregación permita la utilización de estructuras dentro de cada elemento, como se mencionó anteriormente en este trabajo, cada estructura tiene una finalidad, por lo que los atributos que se coloquen

dentro de cada una de estas, debe ser únicamente relevante y enfocada a como se utiliza la información y no como esta almacenada. (Vaish, 2013).

Ejemplificando estas características, se encuentra la agregación de `Usuario` y `Area`. Todos sus atributos se agregaron en el `Usuario`, puesto que estos últimos están identificados por su área y la requieren para operar dentro del SAI.

De igual manera, el número de atributos es de acuerdo a los campos que sean utilizados para cada agregación, por lo que únicamente se colocan aquellos atributos que son relevantes sin la necesidad de duplicar las entidades agregadas en su totalidad.

Cuando la información es almacenada, no es de interés si esta se encuentra duplicada, puesto que el fin del modelo basado en agregaciones es contener aquellos atributos que sean relevantes, minimizando la cantidad de entidades a las cuales debería accederse en un modelo relacional.

En la entidad `Incidencia`, se agregaron atributos relevantes de cada componente, como lo son el identificador y el nombre, pero no contienen información restante del mismo, puesto que no es necesario o relevante más allá de saber que componente se ve afectado por la incidencia.

La estructura de una entidad puede ser dinámica, es decir, un elemento puede o no contener la misma estructura que otro del mismo tipo, sin embargo, se maneja de manera similar o idéntica y no está totalmente limitado por los campos en común o la diferencia entre ellos.

El modelo relacional representa la abstracción de los objetos de negocio mientras que el modelo basado en agregaciones representa un flujo de datos de interés dentro del negocio.

Lo anterior es claramente visible en la entidad `Incidencia`, de acuerdo al componente con el cual esté relacionado, será el atributo que vendrá completo, por ejemplo, es posible tener dos incidencias, pero una está ligada a una aplicación y otra se encuentra ligada a un servidor, la única diferencia es el componente especificado, mientras que los demás atributos pueden ser omitidos sin que esto afecte la manera en que se opera sobre la incidencia.

Consultar la información bajo este modelado, permite consultar una única entidad, es por ello que los atributos contenidos representan la información necesaria y útil para que opere, sin la necesidad de realizar consultas entre dos o más entidades enlazadas por una llave.

3.5.4. *Limitaciones del modelo basado en agregaciones.*

Las problemáticas que pueden presentarse en este modelo, son que el flujo de datos dentro del negocio no sea claro, si bien el modelo es altamente flexible, una agregación mal realizada, impacta tanto en el desempeño como en la forma en que la información se almacena.

Cada modelo es generado de acuerdo al contexto en el que se manejan las entidades, por lo que existen múltiples formas en las que la misma información es

consultada, permitiendo así la creación de múltiples entidades que atienden la misma problemática, pero con desempeño variable.

Si en vez de utilizar una entidad `Incidencia`, se opta por colocar los atributos correspondientes dentro de cada uno de los componentes, al momento en que este sea eliminado, se eliminaran también todas las Incidencias de cada historial, mientras que si se maneja la entidad `Incidencia` aparte, los objetos permanecerán exista o no el componente, ya que la información se encuentra en sus atributos.

Por ende, cualquier consulta que se requiera sea presentada de distinta manera a la especificada en el modelo, puede que sea necesario incluir una nueva entidad con los atributos necesarios de las entidades existentes, sin que estas se vean afectadas por el nuevo requerimiento.

Por ejemplo, si se requiere establecer un nuevo requerimiento, en el cual sea necesario planificar tareas de mantenimiento, que serán aplicados a algunos de los componentes existentes dentro del SAI, es necesario indicar una nueva entidad de agregados sin que la tabla Incidencia se vea afectada.

Se puede optar por crear una nueva entidad `Mantenimiento` o que a cada componente se le agreguen atributos representativos de dicha tarea como un conjunto de `EventoMantenimiento`, partiendo de la analogía que existe entre `Incidencia` y `EventoIncidencia`, con la excepción, de que las tareas de Mantenimiento estarían dentro de cada componente, dictando que un Componente puede o no tener múltiples elementos de este tipo.

La implementación de esta nueva entidad puede ser ambigua, puesto que el requerimiento no existe en el modelo relacional, y por ende queda a libre interpretación las múltiples formas de abordarlo.

Finalmente, de acuerdo al flujo de datos mayormente utilizando, se crearán las entidades representativas de dicho flujo, el modelo basado en agregaciones permite que la base de datos sea flexible de acuerdo a la lógica del negocio y su correcta implementación permitirá una transición con un mayor entendimiento sobre la naturaleza de la información y como es que esta se mueve en el SAI.

3.5.5. *Diseño del modelo de datos (ER) del sistema empleando el modelo relacional.*

Las entidades de mayor importancia en el SAI cuentan con un identificador único que funciona como llave primaria, un nombre, descripción, estatus, así como otros campos con los atributos de mayor relevancia de acuerdo al objeto que están representando:

i) Incidencia.

ii) Usuario.

iii) Componente:

- a) Aplicación
- b) Base de Datos
- c) Servidor

d) Software

También se cuenta con catálogos que proporcionan información adicional a las entidades principales, estas son:

i) Area.

ii) Acceso.

iii) TipoEventoActividad.

Una vez establecidas las entidades principales, es necesario dictaminar mecanismos de relación entre ellas, además de proporcionar detalles adicionales sobre cada una de ellas:

i) EventoIncidencia.

ii) HistorialAcceso.

iii) AdministradorComponente.

a) AdministradorAplicacion.

b) AdministradorBaseDatos.

c) AdministradorServidor.

d) AdministradorSoftware.

iv) IncidenciaComponente:

a) IncidenciaAplicacion

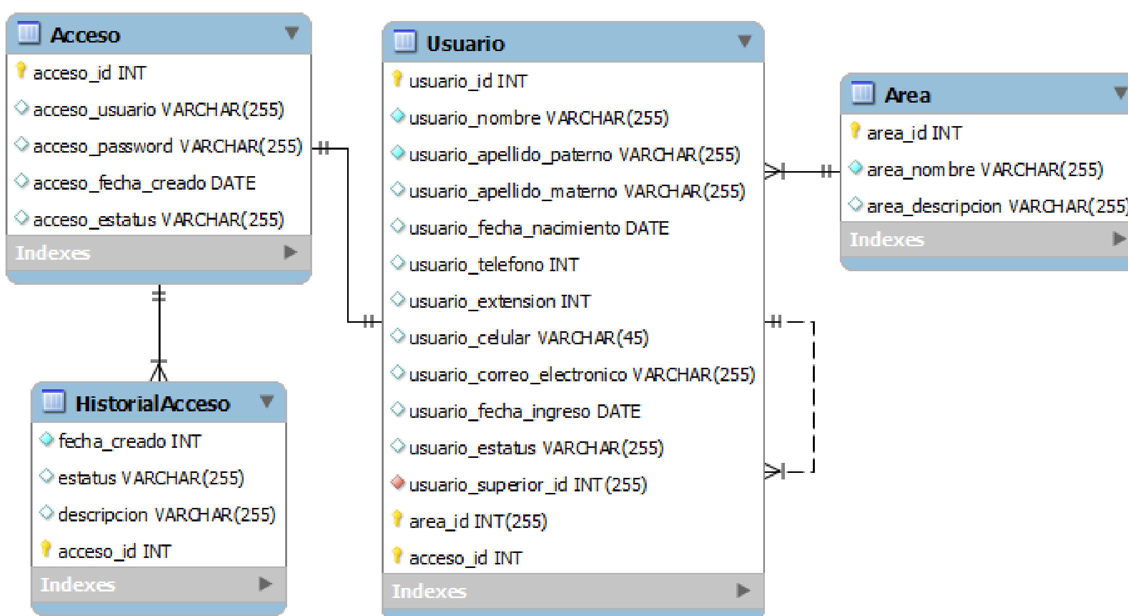
b) `IncidenciaBaseDatos.`

c) `IncidenciaServidor.`

d) `IncidenciaSoftware.`

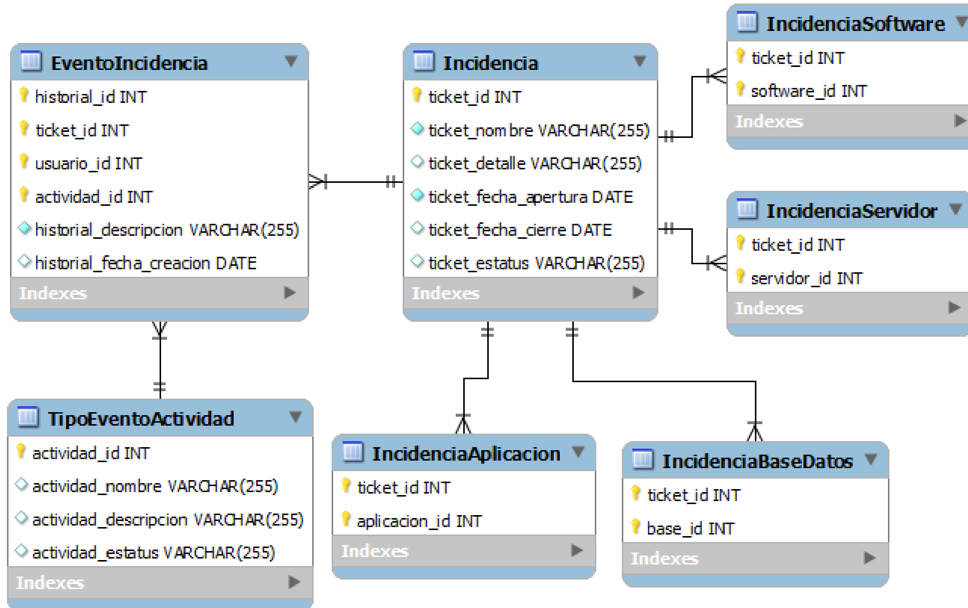
Es posible clasificar a los elementos de acuerdo a su contexto, con esto es posible representar las entidades de manera conjunta de acuerdo a la naturaleza de sus datos:

i) Control de usuarios y accesos.



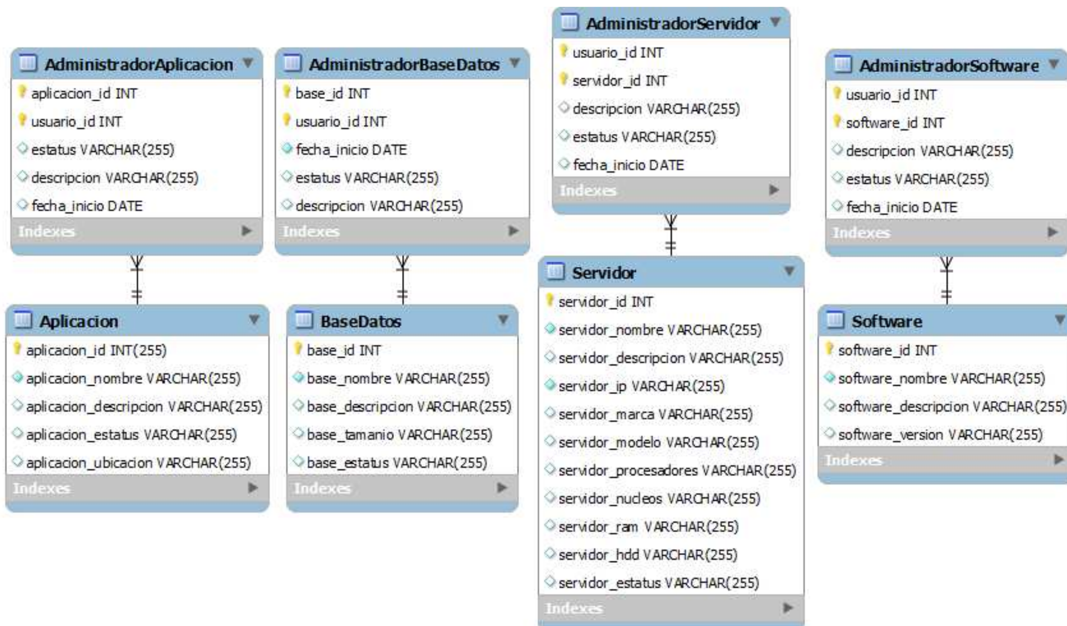
Modelo Entidad-Relación 3-4. Entidades de control de acceso para usuarios.

ii) Seguimiento de Incidencias.



Modelo Entidad-Relación 3-5: Entidades de administración de incidencias.

iii) Configuración de componentes.



Modelo Entidad-Relación 3-6: Entidades de Componente y Administración

3.5.5.1. *Identificación de oportunidades de mejora.*

El modelo anterior sirve de acuerdo a los requerimientos actuales bajos los cuales se creó el SAI, por lo que un nuevo requerimiento puede incluir alguna nueva entidad así como la formación de nuevas relaciones y flujos de datos que inicialmente no se contemplaron dentro del SAI.

Una oportunidad de mejora, es establecer un mecanismo de mantenimiento a cada uno de los componentes, de manera que sirva como agente de notificación a aquellos componentes que requieren tareas de mejora, implementación o corrección.

De igual manera, es posible que dicho mecanismo implemente planes de contingencia para que en caso de que se presente alguna incidencia, se tengan medidas precautorias que reduzcan el impacto que una incidencia pueda generar dentro de la operación del negocio.

Una mejora sustancial realizable dentro del SAI, es la implementación de tecnologías de Inteligencia de Negocio, que permitan explotar los datos que el SAI genera, esto permitirá tener una mejor visión sobre cómo es que se mueven las incidencias, realizando análisis con mayor complejidad que pueden ayudar a una mejor toma de decisiones.

Un requerimiento con mayor notoriedad, es el alertamiento, de manera que cuando un conjunto de incidencias no cumplan con ciertos requisitos, se envíen notificaciones a los responsables solicitando una pronta solución.

3.5.6. *Diseño del modelo de datos a través de una base de datos orientada a documentos.*

El modelo de una base de datos NoSQL orientada a documentos puede partir del modelo basado en agregaciones, puesto que la información que se va a almacenar en una base de datos de este tipo, tiene como ventaja que permite un acceso más rápido a los datos, así como la existencia un volumen considerable de los mismos.

Como el requerimiento de un modelo basado en agregaciones dictamina la estructura de las nuevas entidades con base a las consultas mayormente utilizadas, se tienen que los nuevos elementos se generan de la siguiente manera:

i) `Incidencia`. Mostrado en Documento JSON 3-8: Documento completo de `Incidencia`.

ii) `Usuario`. Mostrado en Documento JSON 3-9: Documento completo de `Usuario`.

iii) `Componente`:

   a) `Aplicacion`. Mostrado en Documento JSON 3-10: Documento completo de `Aplicacion`.

   b) `BaseDatos`. Mostrado en Documento JSON 3-11: Documento complete de `BaseDatos`.

   c) `Servidor`. Mostrado en Documento JSON 3-12: Documento completo de `Servidor`.

d) `Software`. Mostrado en Documento JSON 3-13: Documento completo de `Software`.

```
incidencia:
{
    identificador:null,
    nombre:null,
    detalle:null,
    fechaApertura:null,
    fechaCierre:null,
    estatus:null,
    eventoIncidencia:
    [{
        identificador:null,
        descripcion:null,
        fechaCreacion:null,
        identificadorIncidencia:null,
        usuario:
        {
            identificadorUsuario:null,
            nombre:null,
            correoElectronico:null,
        },
        tipoActividad:
        {
            nombre:null,
            descripcion:null
        }
    }]
    aplicacion:
    {
        identificadorAplicacion:null,
        aplicacionNombre:null
    },
    baseDatos:
    {
        identificadorBaseDatos:null,
        baseDatosNombre:null
    },
    servidor:
    {
        identificadorServidor:null,
        servidorNombre:null
    },
    software:
    {
        identificadorSoftware:null,
        softwareNombre:null
    }
}
```

Documento JSON 3-8: Documento completo de `Incidencia`.

```
usuario:
{
    identificador:null,
    nombre:null,
    apellidoPaterno:null,
    apellidoMaterno:null,
    fechaNacimiento:null,
    telefono:null,
    extension:null,
    celular:null,
    correoElectronico:null,
    estatus:null,
    fechaIngreso:null,
    identificadorJefe:null,
    area:
    {
        identificadorArea:null,
        nombreArea:null,
        descripcionArea:null,
        estatusArea:null
    }
    acceso:
    {
        accesoUsuario:null,
        accesoPassword:null,
        acceso:Estatus:null,
        acccesoFechaCreado:null,
        historialAcceso:
        [{
            estatus:null,
            descripcion:null,
            fechaCreracion:null
        }]
    }
}
```

Documento JSON 3-9: Documento completo de `Usuario`.

```
aplicacion:
{
    identificador:null,
    nombre:null,
    descripcion:null,
    estatus:null,
    ubicacionLogica:null,
    administrador:
    {
        identificadorUsuario:null,
        nombreUsuario:null,
        fechaInicio:null,
        estatusUsuario:null
    }
}
```

Documento JSON 3-10: Documento completo de `Aplicacion`.

```
baseDatos:
{
    identificador:null,
    nombre:null,
    descripcion:null,
    estatus:null,
    tamaño:null,
    administrador:
    {
        identificadorUsuario:null,
        nombreUsuario:null,
        fechaInicio:null,
        estatusUsuario:null
    }
}
```

Documento JSON 3-11: Documento complete de `BaseDatos`.

```
servidor:
{
    identificador:null,
    nombre:null,
    descripcion:null,
    estatus:null,
    ip:null,
    marca:null,
    modelo:null,
    procesadores:null,
    nucleos:null,
    velocidadProcesamiento:null,
    ram:null,
    hdds:null,
    hddCapacidad:null,
    ubicacion:null,
    administrador:
    {
        identificadorUsuario:null,
        nombreUsuario:null,
        fechaInicio:null,
        estatusUsuario:null
    }
}
```

Documento JSON 3-12: Documento completo de `Servidor`.

```
software:
{
    idenfiticador:null,
    nombre:null,
    descripcion:null,
    estatus:null,
    fechaCreacion:null,
    administrador:
    {
        identificadorUsuario:null,
        nombreUsuario:null,
        fechaInicio:null,
        estatusUsuario:null
    }
}
```

Documento JSON 3-13: Documento completo de `Software`.

3.5.7. *Comparativa entre los modelos ER y el modelo orientado a documentos.*

Estas únicas entidades prescinden tanto de los catálogos como de las entidades débiles, puesto que la información ahora se encuentra contenida dentro de cada una de ellas, lo que reduce la dependencia entre objetos, por lo que la dependencia de datos es únicamente informativa y no restrictiva.

Al no incorporar relaciones, es posible omitir algunos de los atributos, como aquellas que funcionan como llaves, tanto primarias como foráneas, reduciendo el tiempo de consulta.

Las nuevas estructuras permiten un mayor dinamismo, puesto que ya no dependen de un esquema, lo que permite modificar la estructura sin afectar a los demás documentos, por lo que es posible agregar nuevos campos o eliminar existentes sin que sea necesario actualizar los demás.

3.6. Selección de frameworks y estrategias de persistencia.

Para la implementación en un modelo NoSQL del SAI, es necesario utilizar herramientas que permitan tanto una implementación y manejo de manera sencilla y simple, sin que ésta deje de ser una solución robusta.

Como se mencionó con anterioridad, existen múltiples enfoques de la bases de datos NoSQL: llave-valor, grafos, documento, etc., Y la cantidad de herramientas existentes es bastante amplio: CouchDB, Cassandra, Neo4j, DynamoDB, BigTable, MongoDB, entre muchos.

Debido a que la complejidad de modelado a partir del modelo de datos orientado a documentos cumple mayormente con los requerimientos del SAI, reduce las herramientas a dos soluciones que se encuentran entre las más populares actualmente, estas son Apache CouchDB y MongoDB.

3.6.1. *¿Por qué MongoDB?*

MongoDB cumple mayormente con los requerimientos del SAI, esta herramienta esta mayormente enfocada a aplicaciones web, en cambio, CouchDB esta principalmente enfocada a aplicaciones de escritorio. Ambas soluciones utilizan el formato JSON para sus documentos.

Una de las ventajas que tiene MongoDB es una mayor simplicidad desde la instalación, la configuración hasta la puesta en marcha, ofreciendo opciones de configuración que van desde lo simple hasta lo más robusto. De esta manera se puede tener una solución funcionando en un tiempo relativamente reducido.

El cliente de MongoDB es bastante sencillo de utilizar, la interacción entre el servidor y el cliente es por medio de JavaScript. Con este lenguaje es posible realizar todas las operaciones CRUD sobre los documentos, así como gran parte de las funciones que JavaScript ofrece.

A diferencia de MongoDB, el cliente de CouchDB utiliza la arquitectura REST, las peticiones son realizadas a través de un cliente HTTP, por lo que podría utilizarse telnet para la interacción con la base de datos.

MongoDB utiliza colecciones de documentos sin la necesidad de un esquema, dicho de otro modo, utiliza esquemas polimórficos (Copeland, 2013), si bien una colección de documentos puede contener cierta información bajo algún contexto, no es necesario que todos estos documentos tengan la misma estructura o el mismo número de atributos, sino que es posible que tengan un mayor o un menor número de estos.

Una de las características de los modelos orientados a documentos, es que representan de algún modo ciertas entidades o un conjunto de ellas, que en un modelo relacional requiere de su normalización y separación para evitar duplicación de la información, de manera que la creación de un documento, es en cierto modo un mapeo de esta información.

Es una solución fácilmente escalable, permite la inclusión de nuevos nodos con tan solo insertar nuevos servidores dentro, por lo que la información puede distribuirse de manera más efectiva, sin la necesidad de manipular los actuales. El escalamiento horizontal de MongoDB y su facilidad de implementación son un gran diferenciador entre otros sistemas escalados verticalmente. (Islam, 2011)

Lenguajes de programación: Python y/o Java.

MongoDB cuenta con una alta variedad de herramienta para conectarse a la base de datos.

Entre las más populares se encuentran las de C++, C#, Java y Python. Un listado completo de todos los lenguajes, APIs y controladores soportados por MongoDB se encuentra en la página de MongoDB. (MongoDB I. , 2014).

El SAI utiliza el API de Java, debido a la facilidad de integración de ambas tecnologías, así como también el framework de Struts 2, el cual permite el desarrollo de aplicaciones web bajo la arquitectura conocida como Modelo-Vista-Controlador, de esta manera se manejan la información y los procesos dentro del SAI a nivel de interfaz, de operación y lógica de negocio y de base de datos.

3.7. Diseño de requerimientos no funcionales.

3.7.1. *Características principales de la arquitectura de MongoDB.*

La arquitectura de MongoDB elegida para el SAI se encuentra compuesta por conjuntos de réplicas, de esta manera, es posible escalar horizontalmente sin mayor complicación, permitiendo que se tenga una alta disponibilidad, redundancia y consistencia.

Por medio de los conjuntos (o sets) de réplicas, es posible establecer una red de hasta 12 instancias de MongoDB. Cada instancia puede estar compuesta por una red de equipos conectados y configurados adecuadamente.

Por medio de la replicación, es posible separar las instancias en primaria y secundaria, de esta manera, las lecturas serán posibles a todos aquellos conjuntos secundarios, mientras que las escrituras se realizaran únicamente en el primario, como se muestra en Diagrama 3-26.

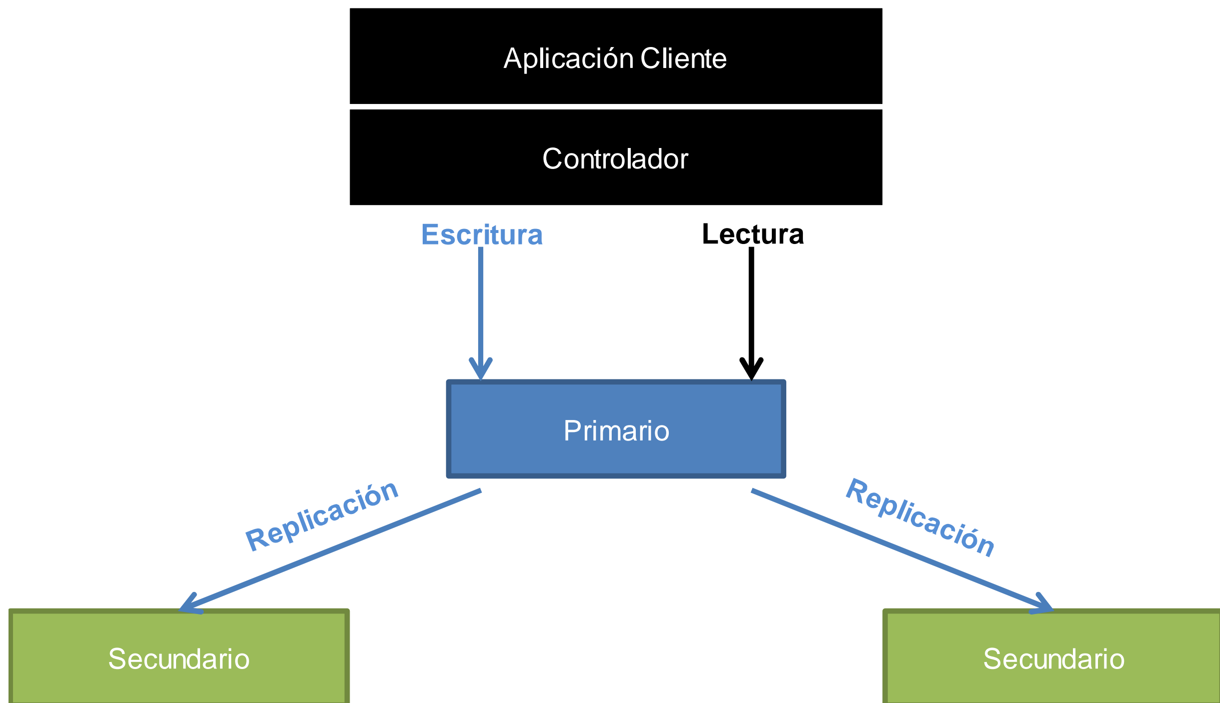


Diagrama 3-26. Arquitectura de Replicación para MongoDB.

Con la separación de primarios y secundarios, se obtiene una arquitectura con mayor tolerancia a fallos, es decir, el conjunto de réplicas escogen de manera automática un nuevo nodo primario.

En el caso del SAI, la replicación es utilizada mayormente para la lectura de información, mientras que las tareas más importantes como la administración de Incidencias y la ingesta de bitácoras ocuparan gran parte del nodo primario.

De igual manera, la información se encontrará con una alta disponibilidad en cada instancia secundaria del SAI, aunada a que mayormente se realizaran tareas de consulta por parte de los usuarios, no tiene mayor impacto sobre el SAI.

3.7.2. *Almacenamiento en MongoDB para el SAI.*

MongoDB crea múltiples archivos de datos por cada base dentro del directorio configurado. Estos archivos cuentan con una extensión **.0**, **.1**, **.2**, **.3**, **etc.**, además de un archivo **.ns**.

Los archivos de datos con extensión numérica contienen toda la información de la base de datos, mientras que el archivo **ns** almacena los namespaces de la base de datos.

Conforme se requiere espacio en la base de datos, MongoDB genera un nuevo archivo con el doble de tamaño del archivo de datos actual, esto es visible en Diagrama 3-27.

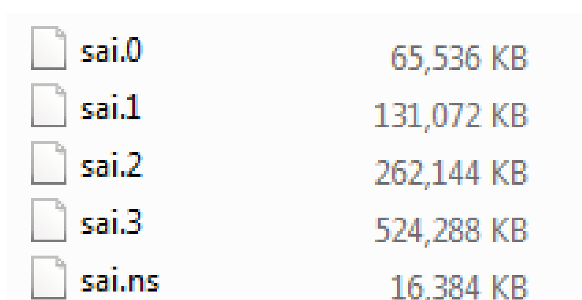


Diagrama 3-27. Archivos de datos para MongoDB

Es posible separar los archivos de bases de datos por carpetas específicas para cada una, se debe establecer el directorio donde se colocaran los respectivos archivos de datos y su archivo de namespace (MongoDB I. , Configuration File Options - MongoDB 2.4.9, 2014).

El SAI utilizará la configuración por defecto, puesto que únicamente existirá su propia base de datos y las colecciones de los componentes más relevantes.

3.7.3. *Transaccionalidad en MongoDB para el SAI.*

Debido a que MongoDB maneja un volumen considerable de información, darle seguimiento a cada inserción o modificación de los datos en la base de datos puede resultar una tarea complicada, debido a que cada operación debería cumplir con las características ACID, teniendo como resultado un tiempo de procesamiento alto, por consecuente, el nivel de transaccionalidad en MongoDB es bastante simple.

Se recomienda la implementación un mecanismo llamado *confirmación de dos fases* (two phase commit). Mediante este mecanismo es posible establecer en múltiples documentos los estados de la operación en la que se encuentran (MongoDB I. , Perform Two Phase Commits - MongoDB Manual 2.4.9, 2014)

Una confirmación de dos fases, permite realizar operaciones de confirmación (commit) y rollback de forma manual, por lo que debe recibir seguimiento en cada cambio entre estados, de manera que se garantice que la operación sobre la información concluya exitosamente, o en caso contrario, se maneje adecuadamente según el último estado de la transacción.

El SAI utilizará este mecanismo para la administración de Incidencias y para la ingesta de bitácoras, ya que se requiere un manejo seguro y eficaz sobre las operaciones de estas entidades.

3.7.4. *Concurrencia en MongoDB para el SAI.*

MongoDB permite las lecturas y escrituras simultáneas, sin embargo las operaciones de escritura tienen una mayor prioridad que las operaciones de lectura,

por lo que es necesario implementar un mecanismo de bloqueo debido a que MongoDB no cuenta con un sistema evolucionado para esta tarea.

En una arquitectura de nodos primario-secundarios, es posible especificar las preferencias de lectura y escritura, por medio de las funciones `setReadPreference(preference)` y `setWriteConcern(concern)` del objeto `MongoClient`.

El método `setReadPreference(preference)` permite especificar el modo en el que se realizarán las lecturas de datos, de acuerdo a la siguiente tabla.

Modo (preference)	Descripción
primary	Todas las operaciones de lectura se realizan sobre el nodo primario actual, permitiendo obtener la última versión de los documentos. **Este es el modo por defecto**.
primaryPreferred	En un caso excepcional de que el primario no se encuentre, se consultaran los secundarios.
secondary	Todas las operaciones de lectura se realizan sobre nodos secundarios.
secondaryPreferred	En un caso en el que los nodos secundarios no se encuentren disponibles, se consultara del nodo primerio.

nearest	Las operaciones de lectura se realizan con el nodo con menor latencia de red, sin importar si es primario o secundario.

Complementado el modo de lectura, el método `setWriteConcern(concern)` permite establecer el mecanismo con el cual se asegura que las escrituras en la base de datos se realizaron exitosamente, permitiendo que la información sea persistente además de obtener un rendimiento mejor para algunos casos. Las posibles configuraciones de las escrituras se muestran en la siguiente tabla.

Modo (concern)	Descripción
UNACKNOWLEDGED	En este nivel, no se intercambian mensajes de confirmación de escrituras, por lo que no es posible identificar si fue exitoso o erróneo.
ACKNOWLEDGED	En este nivel, existen mensajes de confirmación, por lo que es posible identificar errores de red, llaves duplicadas y otros errores. Únicamente aplica a escrituras en el nodo primario. **Este es nivel por defecto**.
JOURNALED	Este nivel envía mensajes de confirmación de escritura una vez realizado una escritura al *journal*, de manera

	que sea posible recuperar la información en caso de que el servidor sea apagado o se interrumpa el suministro eléctrico.
REPLICA_ACKNOWLEDGED	Es similar al ACKNOWLEDGED, con la mejora de que la confirmación de escritura viene por lo menos de dos o más nodos secundarios, según se configure.

El SAI utiliza la configuración por defecto, un `ReadPreference` *primary* y un `WriteConcern` *ACKNOWLEDGED*, de manera que se garantice que las lecturas se realicen sobre la última versión, a costo de rendimiento y las escrituras sean confirmadas, permitiendo identificar los errores que se presentan dentro del mismo.

Sin embargo, cada documento requiere editarse o manipularse de manera concurrente, a diferencia de los parámetros de configuración, se requiere de un mecanismo adicional con el cual sea posible manipular un documento.

Por medio del bloqueo pesimista, se pueden evitar ediciones simultáneas sin afectar las lecturas de un documento, esto gracias a los índices TTL (siglas en inglés para tiempo de vida, Time To Live, que presentan un tiempo asignado en el que el documento será eliminado de manera automática por MongoDB), es posible establecer una colección en la que se agrupen algunos documentos con ciertas características, y que a partir de ellos se puedan consultar o manipular los datos.

De esta manera, al terminar la edición del documento, se puede eliminar el documento respectivo con el índice TTL o bien esperar a que MongoDB lo elimine de manera automática.

El bloqueo pesimista para el SAI permitirá así mantener la concurrencia de lecturas afectando únicamente las escrituras concurrentes, sin embargo, es posible que dichas modificaciones a cada documento no afecten el mismo proceso del SAI, ya que un documento se edita generalmente para agregar información y no para modificar la existente.

Es posible que el documento permita lecturas sucias, sin embargo, estas lecturas serian realizadas mientras es editado el documento.

Una vez finalizada la escritura del documento, el documento de bloqueo se elimina automáticamente, sin importar el índice TTL, ya que con esto se libera el bloqueo y los cambios serian propagados desde el nodo primario hasta los nodos secundarios.

Por lo anterior, el SAI no requiere la implementación de algún otro método de bloqueo debido a que la concurrencia necesaria para administrar y persistir su información es mínima y la implementada de forma habitual es suficiente. Como una funcionalidad agregada, en el capítulo 5.2.2 se implementa un mecanismo adicional con un funcionamiento similar al bloqueo.

MongoDB se escala horizontalmente, de esta manera, según se requiera añadir equipos a la instancia basta con configurar cada nuevo equipo que se desee agregar y añadir la nueva instancia.

MongoDB recomienda un conjunto de replicación de tres miembros, es decir, tres instancias que funcionaran como una primaria y dos secundarias, de esta manera, en caso de fallo, una réplica secundaria ocuparía el lugar de una primaria.

El procedimiento para establecer un conjunto de réplicas, requiere que los equipos configurados sean visibles entre sí.

Cada miembro debe configurar el puerto 27107, la IP del equipo, la ruta donde se alojan los archivos de la base de datos en el archivo `mongodb.conf`, como se muestra en el Diagrama 3-28.

```
port = 27017
bind_ip = 10.8.0.10
dbpath = C:\mongodb\data\sai
fork = true
replSet = rs0
```

Diagrama 3-28. Archivo **mongodb.conf** para un miembro del conjunto de Replicas.

Algunas consideraciones que deben tenerse es que la carpeta especificada en `dbpath` donde se alojarán los archivos de datos debe existir y se debe contar con los permisos adecuados para la creación de archivos.

Con la propiedad `bind_ip` se especifica la IP de las aplicaciones las que MongoDB procesará.

Una vez configurados los archivos, cada instancia debe ser iniciada con el comando:

```
mongod --config C:\mongodb\mongodb.conf
```

Posteriormente, se debe abrir un shell de MongoDB en el miembro que se desea asignar como Primario y se debe ejecutar:

```
mongo
```

Una vez conectado al cliente, se debe iniciar la configuración de los conjuntos de réplicas por medio del comando:

```
rs.initiate()
```

Por último, se agregan los demás miembros de acuerdo a su nombre de red:

```
rs.add(equipo1.sai.com)

rs.add(equipo2.sai.com)
```

Cabe mencionar, que el nombre del primer miembro es `equipo0.sai.com`.

INGENIERIA
F I

Capítulo 4

Implementación de casos representativos

4. Implementación de casos representativos.

4.1. Implementación del proceso.

Previamente en el capítulo 3 de este trabajo, se hace mención de los procesos más representativos del SAI, como lo son: administración de incidencias y generación de reportes. Adicionalmente en este capítulo se mostrará la implementación del proceso de administración de componentes, enfocado en la administración de bitácoras.

4.2. Interacción básica con la base de datos: Operaciones CRUD.

Las operaciones básicas en MongoDB de Creación, Lectura, Actualización y Borrado (CRUD por sus siglas en ingles que representan Create, Read, Update & Delete) permiten al SAI manipular la información de manera más flexible que en un modelo relacional, cabe aclarar que la flexibilidad está altamente ligada con la estructura de los documentos de la colección seleccionada.

Una colección permite catalogar aquellos documentos que comparten similitudes en su estructura, además de proporcionar un mejor orden. Esta funcionalidad permite almacenar documentos en una colección, contraparte de los registros en las tablas en un enfoque relacional.

Para las operaciones CRUD, se parte del principio de documentos comunes y archivos. Estos últimos, de acuerdo a su tamaño, pueden almacenarse directamente en MongoDB, o puede utilizarse la herramienta de GridFS, la cual se detallará más adelante.

4.2.1. *Preparación de la información.*

Partiendo de una entidad básica del SAI, se tiene una representación de una incidencia en la clase `Incidencia` que permitirá, junto con los frameworks GSON y el API de Java, manipular con mayor facilidad y de manera casi transparente el flujo de la información de manera programática en MongoDB.

Para ejemplificar cada una de las operaciones, se creará un objeto de la clase `Incidencia` que contiene los atributos de la entidad, así como los métodos de acceso `get` y `set` para cada uno, de acuerdo a la convención de JavaBeans, como se muestra en el Código Java 4-1.

```java
package mx.mongo.bean;

import java.util.Date;

public class Incidencia
{
    private Long identificador;
    private String nombre;
    private String descripcion;
    private Date fechaApertura;
    private Date fechaCierre;
    private String estatus;
    private EventoIncidencia[] eventos;
    private String[] tags;
    private String coleccion;
    private Aplicacion aplicacion;
    private BaseDatos baseDatos;
    private Infraestructura infraestructura;
    private Software software;

    public Long getIdentificador() {
        return identificador;
    }
    public void setIdentificador(Long identificador) {
        this.identificador = identificador;
    }
```

Código Java 4-1: Clase `Incidencia`.

Mientras que este objeto `Incidencia` representa una incidencia y la información que contiene, debe pasar por un proceso para convertirse en un objeto permitido para su almacenamiento.

Por medio del framework GSON, es posible convertir cualquier clase en un objeto JSON válido, representado en el Código Java 4-2.

```
Gson gson = new GsonBuilder().setPrettyPrinting().create();
```

Código Java 4-2: Constructor del objeto GSON.

La diferencia entre los constructores de GSON se denota al imprimir una clase en su representación JSON: el segundo presenta mayor legibilidad.

En Código Java 4-3, se muestra el objeto con el constructor por defecto `Gson()`.

```
{"identificador":1396799296142,"nombre":"Incidencia","descripcion":"Actualizacio
n de Incidencia","fechaApertura":"Apr 6, 2014 10:48:16 AM","estatus":"Abierto","
eventos":[{"detalle":"Historial 1","descripcion":"Primer elemento de Incidencia"
,"fechaCreacion":"Apr 6, 2014 11:48:16 AM","identificadorUsuario":123456789}],"c
oleccion":"ticket"}
```

Código Java 4-3: Impresión de Objeto Incidencia con el constructor Gson().

Mientras que en Código Java 4-4 se muestra con el constructor `GsonBuilder().setPrettyPainting().create()`.

```
{
    "identificador" : 1375874191647 ,
    "nombre" : "Titulo de la incidencia 1375874191647" ,
    "descripcion" : "Descripcion de la incidencia 1375874191647" ,
    "estatus" : "Cancelada" ,
    "fechaApertura" : { "$date" : "2013-08-07T11:16:31.647Z"} ,
    "infraestructura" : { "identificador" : 7751} ,
    "coleccion" : "incidencia" ,
    "eventos" : [ {
        "identificador" : 0 ,
        "detalle" : "Detalle 1 de la Incidencia 1375874191647" ,
        "descripcion" : "Descripcion del evento 1" ,
        "fechaCreacion" : { "$date" : "2014-04-28T00:43:10.800Z"} ,
        "identificadorUsuario" : 26199
    } , {
        "identificador" : 1 ,
        "detalle" : "Detalle 2 de la Incidencia 1375874191647" ,
        "descripcion" : "Descripcion del evento 2" ,
        "fechaCreacion" : { "$date" : "2014-04-28T12:32:24.436Z"} ,
        "identificadorUsuario" : 63657} ,
    ]}
}
```


Código Java 4-4: Impresión del Objeto con el constructor GsonBuilder().

En el primer caso se observa que el objeto `Incidencia` se imprime en una sola línea, mientras que en el segundo, se le da una línea a cada atributo, si bien esto no impacta a nivel aplicativo, ayuda a que la información sea más legible.

La ventaja de este framework, es que permite la conversión de los objetos de manera sencilla, sin este, la clase debería mapearse para cada caso, como es representado en el Código Java 4-5.

```
BasicDBObject incidencia = new BasicDBObject();
incidencia.put("identificador", id);
incidencia.put("nombre", "Titulo de la incidencia " + id);
incidencia.put("descripcion", "Descripcion de la incidencia " + id);
incidencia.put("estatus", "Abierto");
incidencia.put("fechaApertura", cal.getTime());

cal.add(Calendar.HOUR, 1);
BasicDBObject[] eventos = new BasicDBObject[2];
eventos[0] = new BasicDBObject();
eventos[0].put("identificador", cal.getTimeInMillis());
eventos[0].put("identificadorUsuario", 12346789L);
eventos[0].put("detalle", "Apertura de Incidencia " + id);
eventos[0].put("descripcion", "Evento de apertura: " + id);
eventos[0].put("fechaCreacion", cal.getTime());

cal.add(Calendar.HOUR, 2);
eventos[1] = new BasicDBObject();
eventos[0].put("identificador", cal.getTimeInMillis());
eventos[1].put("identificadorUsuario", 12346789L);
eventos[1].put("detalle", "Modificacion de Incidencia " + id);
eventos[1].put("descripcion", "Informacion adicional: " + id);
eventos[1].put("fechaCreacion", cal.getTime());

incidencia.put("eventos", eventos);
```

Código Java 4-5: Implementación del objeto Incidencia con los tipos de datos del API.

Este objeto corresponde a un objeto válido para MongoDB, pudiendo ser un documento utilizando un `BasicDBObject` o una cadena de texto válida en JSON, como se muestra en Código Java 4-6.

```
{ "identificador" : 1396799636004 , "nombre" : "Titulo de la incidencia 13967996
36004" , "descripcion" : "Descripcion de la incidencia 1396799636004" , "estatus
" : "Abierto" , "fechaApertura" : { "$date" : "2014-04-06T15:53:56.004Z"} , "eve
ntos" : [ { "identificador" : 1396810436004 , "identificadorUsuario" : 12346789
, "detalle" : "Apertura de Incidencia 1396799636004" , "descripcion" : "Evento d
e apertuta: 1396799636004" , "fechaCreacion" : { "$date" : "2014-04-06T16:53:56.
004Z"}} , { "identificadorUsuario" : 12346789 , "detalle" : "Modificacion de Inc
idencia 1396799636004" , "descripcion" : "Informacion adicional: 1396799636004"
, "fechaCreacion" : { "$date" : "2014-04-06T18:53:56.004Z"}}]}
```

Código Java 4-6: Representación JSON de la Incidencia con los constructores del API.

La principal diferencia en el ejemplo anterior, es que en los campos de tipo fecha, la representación que otorga el API de MongoDB incluye el valor del elemento dentro de un sub elemento únicamente para hacer la distinción como tipo de dato fecha.

El manejo de fecha en GSON es menos eficiente, por lo cual, debe implementarse un mecanismo para que estas se serialicen y viceversa correctamente. Para ello, se crearon las clases `DateSerializer` (Código Java 4-7) y `DateDeserializer` (Código Java 4-8).

```java
public class DateSerializer implements JsonSerializer<Date>
{
    final SimpleDateFormat sdf =
            new SimpleDateFormat("yyyy-MM-dd'T'HH:mm:ss.SSS'Z'");

    @Override
    public JsonElement serialize(Date src, Type typeOfSrc,
            JsonSerializationContext context)
    {
        final JsonObject json = new JsonObject();
        json.addProperty("$date", sdf.format(src));

        return json;
    }

}
```

Código Java 4-7: Clase para serializar de fechas.

```
public class DateDeserializer implements JsonDeserializer<Date>
{
    private SimpleDateFormat sdf =
            new SimpleDateFormat("yyyy-MM-dd'T'HH:mm:ss.SSS'Z'");

    @Override
    public Date deserialize(JsonElement json, Type typeOfT,
            JsonDeserializationContext context) throws JsonParseException
    {
        try
        {
            String date = json.toString()
                    .replace("{\"$date\":\"", "")
                    .replace("\"}", "");

            return sdf.parse(date);
        }
        catch (ParseException e)
        {
            e.printStackTrace();
        }
        return null;
    }

}
```

Código Java 4-8: Clase para de serializar fechas.

Cada vez que se llame el constructor del objeto GSON, es necesario especificar la clase que se va a utilizar de acuerdo a la operación.

Como parámetros de entrada, el objeto **gson** debe incluir la clase `DateSerializer`, de manera que todas las fechas sean convertidas al tipo de dato adecuado en MongoDB.

```
Gson gson = new GsonBuilder()
    .registerTypeAdapter(Date.class, new DateSerializer())
    .setPrettyPrinting()
    .create();
```

Código Java 4-9: Constructor GSON con DateSerializer.

Si en cambio, se requiere convertir la fecha en un objeto de salida, se debe utilizar la clase `DateDeserializer` al declarar el objeto.

```
Gson gson = new GsonBuilder()
    .registerTypeAdapter(Date.class, new DateDeserializer())
    .setPrettyPrinting()
    .create();
```

Código Java 4-10: Constructor GSON con DateDeserializer.

4.2.2. *Conexión con el servidor de MongoDB.*

Para conectarse a MongoDB, son necesarios el nombre del servidor o su dirección IP, el número de puerto (`27017` es el puerto por defecto), el nombre de la base de datos, se indican nombre de usuario y contraseña, así como el nombre de la colección que se desea utilizar.

```
Mongo mongo = new MongoClient("localhost", 27017);
DB db = mongo.getDB("sai");
DBCollection collection = db.getCollection("ticket");
```

Código Java 4-11: Conexión a la bases de datos de MongoDB.

Se requiere posteriormente firmarse en el sistema, si las credenciales existen, la función de autenticación regresa verdadero, en caso contrario regresa falso.

```
if (db.authenticate("adminsai", "midna741".toCharArray()))
```

Código Java 4-12: Método de autenticación con MongoDB.

Una vez ejecutados correctamente los pasos, ya es posible realizar operaciones dentro de la base de datos y en la colección especificada.

4.2.3. *Creación/Inserción de un documento.*

Si se tiene un objeto de la clase `Incidencia`, se procede a llenar un objeto **incidencia** con información para su almacenado.

Uno de los atributos del objeto **incidencia** es `eventos`, dentro del cual se colocan objetos **evento**, los cuales a su vez, corresponden a la clase `EventoIncidencia`, la cual ofrece información sobre cada evento realizado durante la gestión de la incidencia.

Por último, el objeto GSON servirá para convertir los objetos **incidencia** e **evento** en texto JSON válido, ejemplificado en el Código Java 4-13.

```
Incidencia ticket = new Incidencia();
EventoIncidencia[] eventos = new EventoIncidencia[2];
EventoIncidencia evento = new EventoIncidencia();
Gson gson = new Gson();
```

Código Java 4-13: Objetos necesarios para la inserción de una Incidencia.

Cada objeto debe ser llenado con la información necesaria para poder insertar el documento.

```
BasicDBObject incidencia = new BasicDBObject();
incidencia.put("identificador", id);
incidencia.put("nombre", "Titulo de la incidencia " + id);
incidencia.put("descripcion", "Descripcion de la incidencia " + id);
incidencia.put("estatus", "Abierto");
incidencia.put("fechaApertura", cal.getTime());

cal.add(Calendar.HOUR, 1);
BasicDBObject[] eventos = new BasicDBObject[2];
eventos[0] = new BasicDBObject();
eventos[0].put("identificador", cal.getTimeInMillis());
eventos[0].put("identificadorUsuario", 12346789L);
eventos[0].put("detalle", "Apertura de Incidencia " + id);
eventos[0].put("descripcion", "Evento de apertura: " + id);
eventos[0].put("fechaCreacion", cal.getTime());
```

Código Java 4-14: Objetos básicos para crear un documento.

Se llena la información del objeto **incidencia** y se agregan dos elementos a **eventos**, cada uno con sus respectivos atributos. Una vez llenado el objeto **incidencia**, está listo para enviarse al método para insertar el documento.

```
@Override
public void creaIncidencia(Incidencia incidencia) throws RuntimeException
{
    String msg = validar(incidencia);

    if (msg.isEmpty())
        msg = incidenciaDAO.crearIncidencia(incidencia);

    if (!msg.isEmpty())
        throw new RuntimeException(msg);
}
```

Código Java 4-15: Llamada al método de inserción.

El método descrito en Código Java 4-16 recibe un objeto `Incidencia`, el cual debe convertirse a un objeto `DBObject`, el cual es el documento básico a insertar en MongoDB.

```java
public String creaIncidencia(Incidencia incidencia) throws RuntimeException
{
    String collection = incidencia.getColeccion();
    BasicDBObject[] eventos = null;
    DBCollection db = null;

    BasicDBObject incidenciaDBO = new BasicDBObject();
    incidenciaDBO.put("identificador", incidencia.getIdentificador());
    incidenciaDBO.put("nombre", incidencia.getNombre());
    incidenciaDBO.put("descripcion", incidencia.getDescripcion());
    incidenciaDBO.put("estatus", incidencia.getEstatus());
    incidenciaDBO.put("fechaApertura", incidencia.getFechaApertura());

    BasicDBObject componente = new BasicDBObject();
```

Código Java 4-16: Atributos Incidencia convertidos a DBObject.

De acuerdo al área del componente que presenta la incidencia, se obtendrá el identificador para dicho elemento, y se colocara en el objeto `Incidencia` convertido, como se muestra en Código Java 4-17.

```java
BasicDBObject componente = new BasicDBObject();

if (incidencia.getAplicacion() != null)
{
    componente.put("identificador", incidencia.getAplicacion().getIdentificador());
    incidenciaDBO.put("aplicacion", componente);
}
else if (incidencia.getBaseDatos() != null)
{
    componente.put("identificador", incidencia.getBaseDatos().getIdentificador());
    incidenciaDBO.put("baseDatos", componente);
}
else if (incidencia.getInfraestructura() != null)
{
    componente.put("identificador", incidencia.getInfraestructura().getIdentificador());
    incidenciaDBO.put("infraestructura", componente);
}
else if (incidencia.getSoftware() != null)
{
    componente.put("identificador", incidencia.getSoftware().getIdentificador());
    incidenciaDBO.put("software", componente);
}
```

Código Java 4-17: Conversión del atributo de componente en una Incidencia.

Es necesario que cada uno de los objetos `EventoIncidencia`, dentro del arreglo eventos sea convertido de igual manera a objetos DBObject, debido a que

tienen la misma estructura de un documento normal, para ello es necesario realizar la conversión de cada elemento en Código Java 4-18.

```
if (incidencia.getEventos() != null && incidencia.getEventos().length > 0)
{
    eventos = new BasicDBObject[incidencia.getEventos().length];
    int contador = 0;

    for (EventoIncidencia evento : incidencia.getEventos())
    {
        eventos[contador] = new BasicDBObject();
        eventos[contador].put("identificador", contador + 1);
        eventos[contador].put("identificadorUsuario", evento.getIdentificadorUsuario());
        eventos[contador].put("detalle", evento.getDetalle());
        eventos[contador].put("descripcion", evento.getDescripcion());
        eventos[contador].put("fechaCreacion", evento.getFechaCreacion());
        contador++;
    }

    incidenciaDBO.put("eventos", eventos);
}
```

Código Java 4-18: Conversión de los eventos de la Incidencia.

Finalmente, una vez que todo el objeto **incidencia** se haya convertido al tipo `BasicDBObject`, se inserta en la base de datos.

```
if (collection != null && !collection.isEmpty())
{
    try
    {
        db = getConexion(collection);

        WriteResult wr =
                db.insert(incidenciaDBO, WriteConcern.ACKNOWLEDGED);

        if (wr.getError() != null)
            throw new RuntimeException(wr.getError());
    }
    catch (Exception e)
    {
        throw new RuntimeException(e.getMessage());
    }
}
```

Código Java 4-19: Inserción del DBObject al final del método creaIncidencia.

El objeto **collection** corresponde a la colección seleccionada, independientemente de si existe dicha colección o si tiene algún documento, es únicamente para indicar donde se insertara el documento en cuestión.

El objeto **dbo** almacenará el documento en un formato legible para MongoDB, para ello se utiliza la clase JSON proporcionada por el API de MongoDB (Código Java 4-20).

Al final de la inserción, se muestra el resultado de la operación con un objeto de la clase `WriteResult`.

```
DBObject dbo = (DBObject)JSON.parse(json);

WriteResult wr = collection.insert(dbo);
```

Código Java 4-20: Código básico para insertar un documento.

```
{
      "serverUsed" : "/192.168.1.21:27017" ,
      "n" : 0 , "connectionId" : 4 ,
      "err" :  null  ,
      "ok" : 1.0
}
```

Documento JSON 4-1: Mensaje de respuesta de una inserción correcta.

4.2.4. *Consulta de documentos.*

La consulta de datos regresa un cursor el cual es un conjunto de documentos encontrados de acuerdo a los criterios de búsqueda seleccionados y bajo la proyección seleccionada, la cual está compuesta por los atributos de interés del mismo.

Tanto los criterios de búsqueda como la proyección son opcionales, sin embargo, se debe tener cuidado de no consultar altos volúmenes de datos, ya que sin los filtros adecuados, la consulta puede extraer toda la información existente en la colección consultada, dificultando su procesamiento y, en casos críticos, impactando gravemente el desempeño.

La plantilla de la consulta, requiere que los parámetros de criterio y proyección sean objetos JSON válidos.

La forma en que se consultan todos los datos de la colección **incidencia** es por medio de la función `db.find()`.

```
public String consultaIncidenciaJSON()
{
    DBCollection db = null;
    String collection = "incidencia";

    if (collection != null && !collection.isEmpty())
    {
        db = getConexion(collection);

        if (db != null)
        {
            return JSON.serialize(db.find());
        }
    }

    return "";
}
```

Código Java 4-21: Consultando toda la información de la colección **incidencia**.

La consulta representada en el Código Java 4-21, regresa una cadena de texto en formato JSON con todos los elementos de la colección **incidencia**.

Por medio de GSON, en conjunto con las clases para la serialización de fechas definidas anteriormente, se construye a partir del texto obtenido una lista de elementos de tipo `Incidencia`.

La consulta mostrada en el Código Java 4-22 obtiene todos los elementos de la colección **incidencia** y el criterio de búsqueda especificado es únicamente por el atributo `colección`, sin embargo, es posible especificar proyecciones más complejas, de acuerdo a los atributos y la estructura de los mismos.

El resultado de la consulta del Código Java 4-22 son todas las incidencias encontradas (Código Java 4-23).

Si en cambio, sólo se requiere consultar un conjunto determinado de objetos, deben establecerse aquellos atributos que se desean incluir con un valor de 1 a la función `db.find(ref, keys)`, donde `ref` corresponde a los parámetros de la consulta y `keys` a los atributos de interés.

Tanto **ref** como **keys**, son objetos de la clase `BasicDBObject`. La diferencia radica en que **ref** contiene los atributos por los cuales se restringe la búsqueda, mientras que **keys** almacenará los atributos de interés que regresarán con la consulta.

```java
public String consultaIncidencia(Incidencia incidencia,
        Date inicio, Date fin) throws RuntimeException
{
    BasicDBObject query = new BasicDBObject();
    DBCollection db = null;
    String json = "";

    if (incidencia.getIdentificador() != null && incidencia.getIdentificador() > 0)
        query.put("identificador", incidencia.getIdentificador());
    if (incidencia.getEstatus() != null && !incidencia.getEstatus().isEmpty())
        query.put("estatus", incidencia.getEstatus());
    if (incidencia.getFechaApertura() != null)
        query.put("fechaApertura", incidencia.getFechaApertura());
    if (incidencia.getFechaCierre() != null)
        query.put("fechaCierre", incidencia.getFechaCierre());
    if (incidencia.getTags() != null && incidencia.getTags().length > 0)
    {
        for (String tag : incidencia.getTags())
            query.put("etiquetas", tag);
    }

    try
    {
        db = getConexion(incidencia.getColeccion());

        json = JSON.serialize(db.find(query));
    }
    catch (Exception e)
    {
        throw new RuntimeException(e.getMessage());
    }

    if (json.isEmpty())
        throw new RuntimeException("No se encontraron incidencias.");;

    return json;
}
```

Código Java 4-22: Método de consulta de Incidencias en el DAO.

```
2014-03-22 14:33:02 INFO - 23   Incidencias encontradas.
2014-03-22 14:33:02 INFO - 1394903374712 - Sat Mar 15 17:09:34 CST 2014
2014-03-22 14:33:02 INFO - 1394903375952 - Sat Mar 15 17:09:35 CST 2014
2014-03-22 14:33:02 INFO - 1394903376973 - Sat Mar 15 17:09:36 CST 2014
2014-03-22 14:33:02 INFO - 1395076183785 - Mon Mar 17 17:09:43 CST 2014
2014-03-22 14:33:02 INFO - 1395076185025 - Mon Mar 17 17:09:45 CST 2014
2014-03-22 14:33:02 INFO - 1395248990377 - Wed Mar 19 17:09:50 CST 2014
2014-03-22 14:33:02 INFO - 1395248991607 - Wed Mar 19 17:09:51 CST 2014
2014-03-22 14:33:02 INFO - 1395248997399 - Wed Mar 19 17:09:57 CST 2014
2014-03-22 14:33:02 INFO - 1395248998629 - Wed Mar 19 17:09:58 CST 2014
2014-03-22 14:33:02 INFO - 1395248999650 - Wed Mar 19 17:09:59 CST 2014
2014-03-22 14:33:02 INFO - 1395421805752 - Fri Mar 21 17:10:05 CST 2014
2014-03-22 14:33:02 INFO - 1395421820615 - Fri Mar 21 17:10:20 CST 2014
2014-03-22 14:33:02 INFO - 1394994477133 - Sun Mar 16 12:27:57 CST 2014
2014-03-22 14:33:02 INFO - 1394994478476 - Sun Mar 16 12:27:58 CST 2014
2014-03-22 14:33:02 INFO - 1394994479508 - Sun Mar 16 12:27:59 CST 2014
2014-03-22 14:33:02 INFO - 1394994480539 - Sun Mar 16 12:28:00 CST 2014
2014-03-22 14:33:02 INFO - 1395167291036 - Tue Mar 18 12:28:11 CST 2014
2014-03-22 14:33:02 INFO - 1395340105064 - Thu Mar 20 12:28:25 CST 2014
2014-03-22 14:33:02 INFO - 1395340106385 - Thu Mar 20 12:28:26 CST 2014
2014-03-22 14:33:02 INFO - 1395340107406 - Thu Mar 20 12:28:27 CST 2014
2014-03-22 14:33:02 INFO - 1395340108430 - Thu Mar 20 12:28:28 CST 2014
2014-03-22 14:33:02 INFO - 1395340109462 - Thu Mar 20 12:28:29 CST 2014
2014-03-22 14:33:02 INFO - 1395340110502 - Thu Mar 20 12:28:30 CST 2014
```

Código Java 4-23: Impresión de los las Incidencias encontradas.

En el Código Java 4-24, **ref** restringe los objetos cuyo atributo `fechaApertura` se encuentre entre la fecha actual (**end**) y tres días anteriores (**start**), estos valores se colocan en el objeto `BasicDBObject` **ref**, delimitándolos por medio de los operadores mayor o igual que (`$gte`) y menor o igual que (`$lte`).

```java
public String consultaIncidenciaFecha()
{
    Calendar cal = Calendar.getInstance();
    BasicDBObject ref = new BasicDBObject();
    BasicDBObject keys = new BasicDBObject();
    Date end = new Date();
    Date start = null;
    DBCollection db = null;
    String collection = "incidencia";

    keys.put("identificador", 1);
    keys.put("fechaApertura", 1);

    cal.setTime(end);
    cal.add(Calendar.DAY_OF_YEAR, -3);
    start = cal.getTime();

    db = getConexion(collection);

    if (db != null)
    {
        ref.put("fechaApertura",
            new BasicDBObject("$gte", start).append("$lte", end));

        return JSON.serialize(db.find(ref, keys));
    }

    return "";
}
```

Código Java 4-24: Consulta de Incidencias por fecha.

Mientras que al objeto **keys**, se le agregarán los atributos `identificador` y `fechaApertura` con un valor numérico de 1, lo que indica que son los únicos atributos a regresar de cada objeto `Incidencia` encontrado (Código Java 4-25).

Estas consultas son realizadas sobre los atributos propios de la clase `Incidencia`, sin embargo, es capaz también de consultar en aquellos elementos dentro de una estructura.

```
2014-03-22 14:34:16 INFO - 8    Incidencias encontradas.
2014-03-22 14:34:16 INFO - 1395421805752 - Fri Mar 21 17:10:05 CST 2014
2014-03-22 14:34:16 INFO - 1395421820615 - Fri Mar 21 17:10:20 CST 2014
2014-03-22 14:34:16 INFO - 1395340105064 - Thu Mar 20 12:28:25 CST 2014
2014-03-22 14:34:16 INFO - 1395340106385 - Thu Mar 20 12:28:26 CST 2014
2014-03-22 14:34:16 INFO - 1395340107406 - Thu Mar 20 12:28:27 CST 2014
2014-03-22 14:34:16 INFO - 1395340108430 - Thu Mar 20 12:28:28 CST 2014
2014-03-22 14:34:16 INFO - 1395340109462 - Thu Mar 20 12:28:29 CST 2014
2014-03-22 14:34:16 INFO - 1395340110502 - Thu Mar 20 12:28:30 CST 2014
```

Código Java 4-25: Incidencias delimitadas por un intervalo de fecha.

En Código Java 4-26, se observa que la Incidencia contiene una lista de objetos `EventoIncidencia`, dentro de este se almacena un atributo `identificadorUsuario` así como otros elementos, todos estos pueden ser asignados como parámetros en la consulta.

Por medio de estos parámetros es posible consultar información dentro de las entidades a través no solo de los atributos, sino también por medio de las estructuras y los mismos atributos de estas que residen en cada documento.

```
{
  "identificador": 1396801355719,
  "nombre": "Incidencia",
  "descripcion": "Prueba de Incidencia",
  "fechaApertura": {
    "$date": "2014-04-06T11:22:35.719Z"
  },
  "estatus": "Abierto",
  "eventos": [
    {
      "identificador": 13579,
      "detalle": "Evento 1",
      "descripcion": "Descripcion del evento.",
      "fechaCreacion": {
        "$date": "2014-04-06T12:22:35.719Z"
      },
      "identificadorUsuario": 123
    },
    {
      "identificador": 24680,
      "detalle": "Evento 2",
      "descripcion": "Descripcion nueva.",
      "fechaCreacion": {
        "$date": "2014-04-06T12:22:35.719Z"
      },
      "identificadorUsuario": 987
    }
  ],
  "coleccion": "incidencia"
}
```

Código Java 4-26: Representación de una Incidencia en JSON.

Para realizar una búsqueda dentro de cada estructura, es necesario establecer como llave **eventos.identificadorUsuario**.

La consulta en Código Java 4-27, regresa aquellas Incidencias en las que `identificadorUsuario` en el atributo `eventos` es igual a 13955343710502L y que se muestran en el Código Java 4-28.

```java
public String consultaIncidenciaUsuario() throws RuntimeException
{
    BasicDBObject ref = new BasicDBObject();
    BasicDBObject keys = new BasicDBObject();
    DBCollection db = null;
    String collection = "incidencia";
    String json = "";

    keys.put("identificador", 1);
    keys.put("fechaApertura", 1);
    keys.put("eventos", 1);

    db = getConexion(collection);

    ref.append("eventos.IdentificadorUsuario", 1395343710502L);

    if (db != null)
    {
        json = JSON.serialize(db.find(ref, keys));

        if (json.isEmpty())
            throw new RuntimeException("Error al consultar la base de datos.");
    }

    return json;
}
```

Código Java 4-27: Consulta de un elemento estructurado dentro de una `Incidencia`.

```
2014-03-22 15:05:52 INFO - 1     Incidencias encontradas.
2014-03-22 15:05:52 INFO - 1395340110502 - Thu Mar 20 12:28:30 CST 2014
```

Código Java 4-28: Incidencias delimitadas por **eventos.identificadorUsuario**.

Es posible anidar elementos cuando se requiere navegar a través de los diferentes niveles de la estructura de un elemento, por ejemplo, para la función `db.find()` en Documento JSON 4-2, la representación JSON de acuerdo a sus parámetros de búsqueda, contiene aquel elemento que se va a buscar, en este caso, el valor de `identificadorUsuario` se encuentra dentro de `eventos`.

```
db.incidencia.find({
    eventos:
    {
        $elemMatch
        {
            identificadorUsuario : 1395343710502
        }
    }
},
{
    identificador : 1,
    eventos : 1
})
```

Documento JSON 4-2: Representación JSON de la consulta de una Incidencia.

4.2.5. *Eliminación de documentos*

Para eliminar los documentos de MongoDB es necesario especificar el documento que se quiere eliminar mediante la creación de un objeto de la clase `BasicDBObject`, y especificando algún atributo con el que se delimitan los objetos a eliminar (Código Java 4-29).

Es posible eliminar todos los documentos de una colección por medio de las funciones `db.collection.drop()` y `db.collection.remove(new BasicDBObject())`.

En el caso de la función `db.remove(new BasicDBObject())`, se le envía como parámetro un documento vacio, debido a que no existen atributos por los cuales delimitar los resultados. Toda la colección se elimina por completo, por lo que es recomendable abstenerse de utilizarlos.

```java
public void eliminaIncidencia(Incidencia incidencia)
        throws RuntimeException
{
    BasicDBObject obj = new BasicDBObject();
    DBCollection db = null;
    String collection = incidencia.getColeccion();

    obj.put("identificador", incidencia.getIdentificador());

    db = getConexion(collection);

    if (db != null)
    {
        WriteResult wr = db.remove(obj);

        if (wr.getError() != null)
            throw new RuntimeException(wr.getError());
    }
}
```

Código Java 4-29: Eliminación de la incidencia por identificador.

4.2.6. *Actualización de documentos.*

Para actualizar un solo documento son necesarios dos objetos de la clase `BasicDBObject`, los cuales almacenarán los parámetros de búsqueda, delimitando los resultados sobre los cuales los nuevos valores, especificados en el segundo objeto van a actualizarse.

Se requiere que el atributo **estatus** de la `Incidencia` en Documento JSON 4-3, sea actualizado a "Cerrado", por lo que es necesario especificar un objeto **query**, que llevará el atributo **identificador** con el identificador del documento, y un objeto **object** que llevará el campo **estatus** con el nuevo valor.

```
{
  "identificador": 1395537188115,
  "nombre": "Incidencia",
  "descripcion": "Actualizacion de Incidencia",
  "fechaApertura": {
    "$date": "2014-03-22T18:51:24.125Z"
  },
  "estatus": "Abierto",
  "coleccion": "incidencia"
}
```

Documento JSON 4-3: Objeto básico de Incidencia en JSON en la base de datos.

En Código Java 4-30, se utiliza un objeto auxiliar **set**, el cual es necesario ya que al no especificar la operación `$set`, se sustituye el documento por el especificado en **object**. Este método actualiza únicamente un documento.

```java
public void actualizaIncidencia(Incidencia incidencia)
        throws RuntimeException
{
    BasicDBObject query = new BasicDBObject();
    BasicDBObject object = new BasicDBObject();
    BasicDBObject set = new BasicDBObject();
    DBCollection db = null;
    String collection = incidencia.getColeccion();

    query.put("identificador", incidencia.getIdentificador());
    object.put("estatus", "Cerrado");
    set.put("$set", object);

    db = getConexion(collection);

    if (db != null)
    {
        WriteResult wr = db.update(query, set);

        if (wr.getError() != null)
            throw new RuntimeException(wr.getError());
    }
}
```

Código Java 4-30: Método de actualización de la Incidencia.

```
{
  "identificador": 1395537188115,
  "nombre": "Incidencia",
  "descripcion": "Actualizacion de Incidencia",
  "fechaApertura": {
    "$date": "2014-03-22T18:51:24.125Z"
  },
  "estatus": "Cerrado",
  "coleccion": "incidencia"
}
```

Código Java 4-31: Actualización del campo `estatus` de una `Incidencia`.

Si se requiere actualizar múltiples documentos a la vez, es necesario utilizar el método `db.updateMulti(query, object)`.

Para modificar la estructura de un documento, ya sea agregando atributos simples o complejos, o agregando un nuevo elemento en un arreglo, pueden utilizarse las funciones `$set` y `$push`.

La función `$set`, en primera instancia, actualiza un atributo con un nuevo valor, sin embargo, si este atributo no se encuentra actualmente dentro de la estructura del documento, se agrega de manera automática, posteriormente, si se vuelve a utilizar, solo se realizará una actualización normal.

```java
public void actualizaMultiplesIncidencias()
        throws RuntimeException
{
    BasicDBObject query = new BasicDBObject();
    BasicDBObject object = new BasicDBObject();
    BasicDBObject set = new BasicDBObject();
    DBCollection db = null;
    String collection = "incidencia";

    query.put("estatus", "Abierto");
    object.put("estatus", "Cancelado");
    set.put("$set", object);

    db = getConexion(collection);

    if (db != null)
    {
        WriteResult wr = db.updateMulti(query, set);

        if (wr.getError() != null)
            throw new RuntimeException(wr.getError());
    }
}
```

Código Java 4-32: Utilizando la función $set.

La función `$push`, realiza las mismas operaciones que `$set`, la diferencia es que agrega un nuevo elemento en el arreglo cada vez, mientras que `$set` únicamente lo actualiza.

```
public int actualizarPush()
{
    BasicDBObject query = new BasicDBObject();
    BasicDBObject object = new BasicDBObject();
    BasicDBObject evento = new BasicDBObject();
    BasicDBObject set = new BasicDBObject();
    DBCollection db = null;
    String collection = "incidencia";
    Long id = 1395537188115L;

    query.put("identificador", id);
    set.put("$push", object);

    evento = new BasicDBObject();
    evento.put("identificador", 27856287L);
    evento.put("identificadorUsuario", 987654321L);
    evento.put("detalle", "Evento  2");
    evento.put("descripcion", "Nueva entrada.");
    evento.put("fechaCreacion", new Date());

    object.put("eventos", evento);

    db = getConexion(collection);

    if (db != null)
    {
        WriteResult wr = db.update(query, set);

        logger.info(wr);

        return 0;
    }

    return -1;
}
```

Código Java 4-33: Utilizando la función $push.

Para eliminar atributos de un documento, basta con especificar un objeto para consulta **query** y otro objeto para almacenar el atributo que se va a eliminar, esto se logra con la función `$unset`.

```java
public void actualizaIncidenciaUnset(Incidencia incidencia)
    throws RuntimeException
{
    BasicDBObject query = new BasicDBObject();
    BasicDBObject object = new BasicDBObject();
    DBCollection db = null;
    String collection = incidencia.getColeccion();

    query.put("identificador", incidencia.getIdentificador());
    object.put("$unset", new BasicDBObject("eventos", ""));

    db = getConexion(collection);

    if (db != null)
    {
        WriteResult wr = db.update(query, object);

        if (wr.getError() != null)
            throw new RuntimeException(wr.getError());
    }
}
```

Código Java 4-34: Utilizando la función $unset.

4.2.7. *Manejo de archivos con GridFS*

El SAI permite adjuntar archivos durante la gestión de las incidencias, permitiendo que se inserten archivos como lo son imágenes, documentos de texto, así como bitácoras de las aplicaciones que lo requieran, otorgando un conjunto de información adicional durante el proceso de administración de cada incidencia.

En el caso de las bitácoras, es necesario especificar su estructura de manera explícita, debido a que su contenido debe ser insertado de acuerdo a esta. El SAI requiere que se especifiquen las columnas que componen a la bitácora, un carácter separador de texto y un alias para crear una colección donde se almacenará el contenido de la bitácora.

De acuerdo a la configuración establecida, se crearan en la colección especificada por alias, los documentos con los atributos especificados para cada registro, de manera que una bitácora puede tener diferente estructura de otra y no requiere una tabla o configuración adicional.

Para el manejo de archivos dentro de la base de datos, MongoDB utiliza la herramienta GridFS. El principal diferenciador entre el sistema de archivos nativo de MongoDB y GridFS radica en el tamaño permitido de los documentos y el tamaño del archivo.

Mientras que el tamaño máximo de un documento BSON en la versión 2.6.1 de MongoDB es de 16MB, un archivo puede ser de un mayor tamaño, por lo que será necesario gestionarlo por medio de GridFS.

Esta herramienta se encarga de separar el archivo en conjuntos de documentos de cierto tamaño, sea el valor por defecto de 255KB o algún valor configurado, siempre y cuando no se exceda el máximo permitido de 16MB o el especificado por la versión de MongoDB en curso. Es importante mencionar que el tiempo de respuesta es inversamente proporcional al tamaño del pedazo utilizado.

Para insertar el archivo, se debe especificar un prefijo **bucket**, el cual indica el nombre de la colección que será utilizada específica para almacenar archivos, dentro de la cual son creados los documentos **files** y **chunks** para cada archivo.

Estos nombres son propios de MongoDB, mientras que **bucket** puede llamarse diferente, por defecto, el nombre del **bucket** es **fs**, para finalmente

almacenar el archivo en los documentos **bucket.files** y **bucket.chunks**. (MongoDB I. , GridFS Reference - MongoDB Manual 2.6.1, 2014)

El documento **files** contiene los metadatos del archivo Documento JSON 4-4, atributos como el identificador del objeto, nombre, fecha de procesamiento y tamaño, entre otros. Este documento es único para cada archivo.

```
{
  "_id" : <ObjectId>,
  "length" : <num>,
  "chunkSize" : <num>,
  "uploadDate" : <timestamp>,
  "md5" : <hash>,
  "filename" : <string>,
  "contentType" : <string>,
  "aliases" : <string array>,
  "metadata" : <dataObject>,
}
```

Documento JSON 4-4: Estructura básica del documento files.

El conjunto de documentos **chunks** corresponde a los pedazos en los que el archivo fue separado de acuerdo al tamaño del documento especificado Documento JSON 4-5. La información que contiene cada uno de estos documentos, es el identificador del archivo, el número de pedazo, la información binaria del archivo, entre otros.

```
{
  "_id" : <ObjectId>,
  "files_id" : <ObjectId>,
  "n" : <num>,
  "data" : <binary>
}
```

Documento JSON 4-5: Estructura básica del documento chunks.

La gestión de los documentos **files** y **chunks** se realiza de manera transparente para el SAI, ya que MongoDB se encarga de reconstruir internamente el archivo a partir de los documentos en los cuales fue segmentado, consultando únicamente el documento **files**.

En el método mostrado en Código Java 4-35, se solicita la colección **bucket** para el archivo **filename** y se inserta en la base de datos. El archivo se almacena entonces en los documentos **bucket.files** y **bucket.chunks**.

```
public void insertaArchivo(String bucket, String filename)
    throws RuntimeException
{
    try
    {
        File file = new File(filename);

        if (file.exists())
        {
            DB db = getDB();

            GridFS gfs = new GridFS(db, bucket);
            GridFSInputFile gfsFile = gfs.createFile(file);
            gfsFile.setFilename(filename);
            gfsFile.save();
        }
    }
    catch (Exception e)
    {
        throw new RuntimeException(e);
    }
}
```

Código Java 4-35: Método de inserción de archivo con GridFS.

Para consultar el archivo, es necesario especificar la colección **bucket** dentro de la cual fue almacenado el archivo con nombre **fileIn**, por último, este será reconstruido en el directorio y con el nombre especificado por **fileOut**.

GridFS realiza una consulta dentro de la colección **bucket**, buscando el archivo con nombre **fileIn**, una vez encontrado, genera el archivo **fileOut**. Código Java 4-36.

```
public void generaArchivo(String bucket, String fileIn, String fileOut)
    throws RuntimeException
{
    try
    {
        BasicDBObject query = new BasicDBObject();
        DB db = getDB();

        query.put("filename", fileIn);

        GridFS gfs = new GridFS(db, bucket);
        GridFSDBFile output = gfs.findOne(query);
        output.writeTo(fileOut);
    }
    catch (Exception e)
    {
        throw new RuntimeException(e.getMessage());
    }
}
```

Código Java 4-36: Generación de archivo almacenado con GridFS.

Para eliminar el archivo, se requieren especificar la colección **bucket** y el nombre del archivo **filename**, con el cual se consultara primero el archivo para después eliminarlo de la colección.

```java
public void eliminaArchivo(String bucket, String filename)
    throws RuntimeException
{
    try
    {
        BasicDBObject query = new BasicDBObject();
        DB db = getDB();

        query.put("filename", filename);

        GridFS gfs = new GridFS(db, bucket);
        gfs.remove(query);
    }
    catch (Exception e)
    {
        throw new RuntimeException(e.getMessage());
    }
}
```

Código Java 4-37: Eliminación de un archivo con GridFS.

Es importante remarcar que no es necesario consultar directamente los documentos **files** y **chunks**, ya que GridFS se encarga de relacionarlos de manera automática, además de que internamente se manipulan los dos como un solo conjunto de información, evitando así inconstancias de datos.

Dentro de los procesos de consulta y eliminación de archivos, es necesario que el nombre del archivo sea el mismo al momento de su inserción, ya que puede llevar la ruta de origen, tal como se detalla en Documento JSON 4-6.

```
{
    "_id" : ObjectId("532e9fc312a632210485498b"),
    "chunkSize" : NumberLong(262144),
    "length" : NumberLong(104479261),
    "md5" : "86078d5024b33ee02ed35dbddcfb40d1",
    "filename" : "c:\\logs\\log_reportes2012.log",
    "contentType" : null,
    "uploadDate" : ISODate("2014-03-23T08:48:03.140Z"),
    "aliases" : null
}
```

Documento JSON 4-6: Ejemplo de un documento files.

Sin embargo, al utilizar un objeto **BasicDBObject**, es posible especificar parámetros adicionales de consulta, aunque no se recomienda el uso de esta funcionalidad, ya que pueden eliminar bitácoras con nombres idénticos.

4.2.8. *Administración de bitácoras dentro del SAI.*

Como se mencionó anteriormente, SAI permite almacenar bitácoras para su procesamiento posterior, de esta forma cada bitácora será almacenada ya no como un archivo, sino como un conjunto de documentos dentro de una colección personalizada y única.

Al adjuntar una bitácora, serán solicitados los siguientes parámetros:

a) Una colección temporal **alias** o cualquier otro nombre, donde serán almacenados los registros de la bitácora. Por cada registro existente en la bitácora, se crea un documento en la colección **alias**. Esta colección es independiente de las colecciones utilizadas por el SAI, es decir, estas colecciones son totalmente dinámicas y creadas a partir de las bitácoras a petición del usuario. Ejemplo: “*bitácora123*”.

b) Los nombres de las columnas. Estos deben ser valores alfanuméricos únicos comenzando con letras y una longitud máxima de 24 caracteres. Por medio de estos valores, se especifica la estructura de cada documento creado dentro de la colección **alias**, creando un prototipo de documento que será llenado con los valores de los registros.. Ejemplo: "*a01, b, campo3*".

c) Un carácter separador, con el cual se delimitarán las columnas y las celdas dentro de la bitácora, de esta manera, cada valor obtenido se almacena en uno de los atributos del documento a insertar de acuerdo a las columnas especificadas. Por ejemplo: "|", ",", "*\t*".

Con estas configuraciones adicionales, serán creados un conjunto de *n* documentos dentro de la colección *temporal* especificada, todos los documentos contendrán los atributos especificados por las columnas. Se creará un documento por cada registro o renglón de la bitácora, el cual dividirá las columnas de acuerdo al separador y serán colocadas dentro de los atributos de cada documento.

Los parámetros de configuración permitirán convertir cada registro de la bitácora en un documento válido de MongoDB, ejemplificado al convertir la información de la bitácora en la Ilustración 4-1 a la colección de documentos en Documento JSON 4-7.

```
fecha   texto   numero
10-10-2013 23:40:12     cadena de texto 001
10-10-2013 23:42:15     bitacora de prueba      002
10-10-2013 23:52:10     separado por tabulaciones       003
```

Ilustración 4-1: Datos de muestra de un archivo de bitácora sencillo.

```
{
    "fecha" : "10-10-2013 23:40:12",
    "texto" : "cadena de texto",
    "numero" : "001"
}

{
    "fecha" : "10-10-2013 23:42:15",
    "texto" : "bitacora de prueba",
    "numero" : "002"
}

{
    "fecha" : "10-10-2013 23:52:10",
    "texto" : "separado por tabulaciones",
    "numero" : "003"
}
```

Documento JSON 4-7: Documentos creados a partir de una bitácora.

De esta manera, los registros de la bitácora ya se encuentran almacenados dentro del SAI con la estructura especificada, sin embargo, es importante mencionar, que debido a la naturaleza de este procesamiento, SAI considera todos los campos como cadenas de texto, por lo que los documentos que se crearon a partir de la bitácora deben ser consultados como tal.

4.3. Implementación y configuración de la estrategia para el manejo de transacciones.

La transaccionalidad dentro el SAI, así como en cualquier sistema, garantiza que los datos sean consistentes después de una operación de inserción, eliminación o actualización. De esta forma, es posible regresar a un estado posterior a la operación en caso de cualquier error.

El nivel de transaccionalidad por defecto en MongoDB, va de acuerdo al nivel de configuración especificado para el cliente, tal como se indica en el apartado 4.6.4 del presente trabajo.

De acuerdo al nivel configurado, es posible tener o no respuesta después de la ejecución de alguna operación de escritura de algún documento, por lo que la transaccionalidad es totalmente atómica.

```java
public void insertaDocumento(String collection, DBObject dbObj)
{
    try
    {
        DBCollection db = getConexion(collection);

        WriteResult wr =
                db.insert(dbObj, WriteConcern.ACKNOWLEDGED);

        if (wr.getError() != null)
            throw new RuntimeException(wr.getError());
    }
    catch (Exception e)
    {
        throw new RuntimeException(e.getMessage());
    }
}
```

Código Java 4-38: Código para un documento genérico especificando nivel de escritura.

Sin embargo, debido a que este mecanismo funciona a nivel documento, si se requiere manipular más de un documento a la vez, es necesario utilizar un mecanismo que permita llevar un control sobre los mismos. El mecanismo utilizado es conocido como Confirmación de dos Fases o Two-Phase Commit.

Tal y como se especifica en el apartado 4.6.3, es necesario contar con un elemento que almacene el estatus de las operaciones, de tal forma que se pueda enviar una confirmación (commit) o ejecutar una restauración de datos (rollback).

El SAI utiliza un objeto **trans** de la clase `Transaccion`, el cual almacenará el estado de la transacción durante todo el proceso de escritura de datos. El objeto **trans** contiene un identificador, fecha de creación y su fecha de actualización, así como el estatus de la transacción y su respectiva colección, será especificado por los valores en una enumeración dentro de la misma clase. Código Java 4-39.

```
public class Transaccion
{
    private Long identificador;
    private Date creado;
    private Date actualizado;
    private Estatus estatus;
    private String coleccion;

    public Transaccion()
    {
        creado = new Date();
        identificador = creado.getTime();
        estatus = Estatus.procesando;
        coleccion = "transacciones";
    }

    public enum Estatus
    {
        procesando,
        fallido,
        terminado
    }
```

Código Java 4-39: Atributos de la clase Transaccion.

Cada una de las transacciones generadas, será insertada dentro de la colección **transacciones**, y la estructura de cada documento se muestra en Documento JSON 4-8.

```
{
    "_id" : ObjectId("537e0f8fc502d7fc707655b2"),
    "identificador" : NumberLong("1400770447487"),
    "creado" : ISODate("2014-05-22T14:54:07.487Z"),
    "estatus" : "procesando"
}
```

Documento JSON 4-8: Documento inicial de transacción.

El documento **trans** creado, es apenas la mitad de la operación, mientras que este documento será actualizado, es necesario especificar en el grupo de documentos a manipular, un campo *transacción*, el cual almacena el identificador de la transacción creada.

Si se desea insertar un conjunto de objetos **Incidencia**, es necesario modificar la clase, agregando un campo numérico *transacción*, en el cual se almacenara el identificador de la transacción que será utilizada. Código Java 4-40.

```
public class Incidencia
{
    private Long identificador;
    private String nombre;
    private String descripcion;
    private Date fechaApertura;
    private Date fechaCierre;
    private String estatus;
    private EventoIncidencia[] eventos;
    private String[] tags;
    private String coleccion;
    private Aplicacion aplicacion;
    private BaseDatos baseDatos;
    private Infraestructura infraestructura;
    private Software software;
    private Long transaccion;
```

Código Java 4-40: Modificación a la clase Incidencia para permitir transacciones.

Una vez preparados el documento `Transaccion` y el grupo de documentos `Incidencia`, la operación debe seguir la secuencia del método en Código Java 4-41.

```java
@Override
public void creaIncidenciasTrans(Incidencia[] incidencias)
    throws RuntimeException
{
    Transaccion transaccion = new Transaccion();
    String coleccion = incidencias[0].getColeccion();

    try
    {
        incidenciaDAO.creaTransaccion(transaccion);

        for (Incidencia inc : incidencias)
        {
            inc.setTransaccion(transaccion.getIdentificador());
            incidenciaDAO.creaIncidencia(inc);
        }

        transaccion.setEstatus(Estatus.terminado);
        incidenciaDAO.actualizaTransaccion(transaccion);
    }
    catch (Exception e)
    {
        ejecutaRollback(coleccion, transaccion);
        transaccion.setEstatus(Estatus.fallido);
        incidenciaDAO.actualizaTransaccion(transaccion);
        throw new RuntimeException(e.getMessage());
    }
}
```

Código Java 4-41: Método de inserción transaccional de un grupo de incidencias.

En primer lugar, se debe crear un objeto de la clase `Transaccion`, para insertarse posteriormente a la base de datos. Este documento contiene la fecha en la que se creó, un estatus inicial *procesando* y un identificador numérico, el cual va ligado a la fecha de creación de la transacción. Estos valores se asignan al

momento de instanciar el objeto para después insertarse en la base de datos. Código Java 4-42.

Este documento transacción, es requerido en la base de datos, ya que el identificador de este es referenciado por el conjunto de incidencias a insertar.

```
public void creaTransaccion(Transaccion transaccion)
    throws RuntimeException
{
    String collection = transaccion.getColeccion();
    DBCollection db = null;

    BasicDBObject transBO = new BasicDBObject();
    transBO.put("identificador", transaccion.getIdentificador());
    transBO.put("creado", transaccion.getCreado());
    transBO.put("estatus", transaccion.getEstatus().toString());

    try
    {
        db = getConexion(collection);

        WriteResult wr = db.insert(transBO);

        if (wr.getError() != null)
            throw new RuntimeException(wr.getError());
    }
    catch (Exception e)
    {
        throw new RuntimeException(e.getMessage());
    }
}
```

Código Java 4-42: Método de creación del objeto Transaccion dentro de MongoDB.

Una vez insertado el documento de la clase `Transaccion`, se asignara a cada una de las incidencias el identificador de este en su respetivo atributo.

Se procede con la operación, en este caso, el proceso realizar inserciones de cada uno de los documentos.

En caso de que por alguna razón, el proceso falle durante la inserción de algún documento, se ejecuta la función de rollback en Código Java 4-43.

```
@Override
public int ejecutaRollback(String coleccion, Transaccion transaccion)
    throws RuntimeException
{
    try
    {
        return incidenciaDAO.ejecutaRollback(coleccion, transaccion);
    }
    catch (RuntimeException e)
    {
        throw new RuntimeException(e.getMessage());
    }
}
```

Código Java 4-43: Método de rollback implementado

En caso de error, este método elimina todas las incidencias dentro de la colección, en las que el identificador de la transacción corresponde al identificador del objeto creado. El método del DAO se muestra en Código Java 4-44.

```java
public int ejecutaRollback(String coleccion, Transaccion transaccion)
    throws RuntimeException
{
    try
    {
        BasicDBObject query = new BasicDBObject();
        DBCollection db = getConexion(coleccion);

        query.put("transaccion", transaccion.getIdentificador());

        WriteResult wr = db.remove(query);

        if (wr.getError() != null)
            throw new RuntimeException(wr.getError());

        return wr.getN();
    }
    catch (Exception e)
    {
        throw new RuntimeException(e.getMessage());
    }
}
```

Código Java 4-44: Método de rollback dentro del DAO.

En caso de que el proceso sea exitoso o erróneo, es necesario actualizar el objeto `Transaccion` utilizado, especificando el estatus de acuerdo al resultado de la operación. Si la operación fue exitosa, se debe especificar un estatus *terminado*, mientras que si la operación presentó errores, deberá especificar un estatus *fallido*. Código Java 4-45.

```java
public void actualizaTransaccion(Transaccion transaccion)
    throws RuntimeException
{
    String collection = transaccion.getColeccion();
    BasicDBObject query = new BasicDBObject();
    BasicDBObject obj = new BasicDBObject();
    DBCollection db = null;

    query.put("identificador", transaccion.getIdentificador());

    obj.put("actualizado", new Date());
    obj.put("estatus", transaccion.getEstatus().toString());

    try
    {
        db = getConexion(collection);

        WriteResult wr = db.update(query, obj);

        if (wr.getError() != null)
            throw new RuntimeException(wr.getError());
    }
    catch (Exception e)
    {
        throw new RuntimeException(e.getMessage());
    }
}
```

Código Java 4-45: Método de actualización de Transaccion.

Mientras que la ejecución de un *rollback* es necesaria en caso de un error durante el procesamiento, un *commit* no es necesario, debido a que cada documento realiza una confirmación, siempre y cuando, se tenga un nivel de respuesta de escritura adecuado.

4.4. Implementación de procesos de búsqueda y explotación de la información.

En MongoDB para ejecutar una búsqueda sobre la información almacenada basta con especificar los atributos y valores que delimitarán los documentos de una colección determinada.

Se requiere un procesamiento complejo sin la necesidad de consultar todos los documentos, para ello es necesario implementar funciones Map-Reduce, las cuales permiten realizar funciones de agregación sobre un conjunto de documentos.

Por lo anterior, la función Map-Reduce es empleada para la generación de reportes y explotación de la información.

4.4.1. *Operaciones Map-Reduce.*

Map-Reduce está formada por un conjunto de métodos aplicables sobre un conjunto de documentos, la cual realiza tareas complejas de agregación, que son el equivalente a las funciones de agregación **count**, **distinct** o **group** en el modelo relacional. (MongoDB Documentation Project, 2014)

Todas las operaciones Map-Reduce se realizan a través de funciones JavaScript.

Las funciones **map** y **reduce** se encargan de manipular los documentos para presentar un conjunto de elementos como salida. La estructura de la función se muestra en el Documento JSON 4-9.

```
{
    mapReduce: <collection>,
    map: <function>,
    reduce: <function>,
    out: <output>,
    query: <document>,
    sort: <document>,
    limit: <number>,
    finalize: <function>,
    scope: <document>,
    jsMode: <boolean>,
    verbose: <boolean>
}
```

Documento JSON 4-9: Estructura de la función map-reduce.

El atributo **mapReduce** específica el nombre de la colección sobre la cual se realizará la operación.

La función **map** se encarga de *realizar el mapeo de* los documentos en conjuntos llave-valor, para su procesamiento indicado por la función **reduce**.

Estos tres atributos son requeridos para una operación map-reduce básica.

El documento **out** específica la ubicación de los resultados de la operación. Es posible enviar los resultados directamente a una colección.

Los documentos se delimitan por los atributos y valores especificados en el objeto **query**.

El resultado del procesamiento de los documentos será ordenado de acuerdo a los atributos del documento **sort**, mientras que **limit** permitirá establecer un número máximo de documentos de la colección que se procesarán.

El atributo **finalize** permite dar seguimiento a la función declarada en **reduce**, mientras que el atributo **scope** permite establecer variables globales que son accesibles en las funciones **map**, **reduce** y **finalize**.

Por medio del atributo **verbose** es posible especificar información sobre el tiempo de procesamiento de la función map-reduce, por defecto, este valor es verdadero (true).

Una función **map** devuelve un conjunto llave-valor con base a los atributos de un documento. En el Código JavaScript 4-1 se devuelve la fecha en formato `dd/MM/yyyy` y un valor numérico *conteo* de 1, estos valores son regresados por medio de `emit` cada vez que la función `map` es invocada.

```
function ()
{
    var date = this.fechaApertura;
    var fecha = ('0' + date.getDate()).slice(-2)+'/';
    fecha += ('0' + (date.getMonth() + 1)).slice(-2)+'/';
    fecha += date.getFullYear();
    emit(fecha, {conteo:1} );"
}
```

Código JavaScript 4-1: Función map para el atributo fechaApertura

La función **reduce** se encarga de procesar los pares que recibe de **map**, agrupándolos de acuerdo al valor de *fecha* y regresando el valor respectivo de *total*, el cual, almacena la suma de elementos agrupados por *fecha*, como se muestra en Código JavaScript 4-2.

```
function (key, values)
{
    var total = 0;
    for (var i in values) {
        total += values[i].conteo;
    }

    return {conteo:total};
}
```

Código JavaScript 4-2: Función reduce para los pares fecha-conteo.

La operación map-reduce que realiza estas funciones, permite agrupar las fechas de las incidencias de acuerdo a la fecha *fechaApertura*. Código Java 4-46

```
MapReduceCommand cmd =
            new MapReduceCommand(collection, map, reduce,
        null, MapReduceCommand.OutputType.REPLACE, query);

MapReduceOutput out = collection.mapReduce(cmd);
```

Código Java 4-46: Método para ejecutar la función map-reduce.

La función map-reduce contempla todos los documentos de la colección **incidencia** y regresa el número total de documentos agrupados por el atributo `fechaApertura`.

```
{ "_id" : "2014/03/15" , "value" : { "conteo" : 3.0}}
{ "_id" : "2014/03/16" , "value" : { "conteo" : 4.0}}
{ "_id" : "2014/03/17" , "value" : { "conteo" : 2.0}}
{ "_id" : "2014/03/18" , "value" : { "conteo" : 1.0}}
{ "_id" : "2014/03/19" , "value" : { "conteo" : 5.0}}
{ "_id" : "2014/03/20" , "value" : { "conteo" : 4.0}}
{ "_id" : "2014/03/21" , "value" : { "conteo" : 2.0}}
{ "_id" : "2014/03/22" , "value" : { "conteo" : 2.0}}
```

Documento JSON 4-10: Resultados de la operación map-reduce.

Es posible delimitar los documentos especificando un objeto **query** de la clase `BasicDBObject`, el cual lleva los atributos por los cuales se desea filtrar los documentos. En el Código Java 4-47 se procesarán aquellos documentos cuya fecha se apertura sea 20 de marzo del 2014 y hasta 3 días posteriores.

```
Date start = null;
Date end = null;

cal.clear();
cal.set(2014, 2, 20, 0, 0, 0);
start = cal.getTime();
cal.add(Calendar.DAY_OF_YEAR, 3);
end = cal.getTime();

openDate.append("$gte", start).append("$lt", end);
query.put("fechaApertura", openDate);

if (db.authenticate("adminsai", "midna741".toCharArray()))
{
    collection = db.getCollection("incidencia");

    MapReduceCommand cmd =
                new MapReduceCommand(collection, map, reduce,
             null, MapReduceCommand.OutputType.REPLACE, query);

    MapReduceOutput out = collection.mapReduce(cmd);

    for (DBObject o : out.results())
        logger.info(o.toString());
}
```

Código Java 4-47: El objeto query delimita los documentos a procesar.

De acuerdo a los nuevos parámetros en la función map-reduce, los nuevos conjuntos llave-valor obtenidos son los mostrados en el Documento JSON 4-11

```
{ "_id" : "2014/03/20" , "value" : { "conteo" : 4.0}}
{ "_id" : "2014/03/21" , "value" : { "conteo" : 2.0}}
{ "_id" : "2014/03/22" , "value" : { "conteo" : 2.0}}
```

Documento JSON 4-11: Resultados del procesamiento map-reduce delimitado.

Es posible especificar la colección de destino donde se almacenarán los documentos que regresa la función map-reduce, para ello se especifica la colección **conteos** y el tipo `MapReduceCommand.OutputType.REPLACE`. Código Java 4-48

```
MapReduceCommand cmd =
            new MapReduceCommand(collection, map, reduce,
        "conteos", MapReduceCommand.OutputType.REPLACE, query);
```

Código Java 4-48: Almacenando los resultados de la consulta.

De esta manera, cada vez que se ejecute la función, se reemplazaran los documentos existentes en la colección por los nuevos obtenidos.

4.5. Comparativa operacional entre los modelos NoSQL y relacional.

Dentro de las operaciones CRUD realizadas, se hace evidente una de las mayores diferencias entre el modelo NoSQL y el modelo relacional, esta diferencia es la cantidad de tablas requeridas en el NoSQL, puesto que únicamente se maneja una colección de documentos, en vez de registros distribuidos en múltiples entidades.

La mayor mejora que puede encontrarse, es que al no requerir de múltiples tablas a través de los cuales pueden distribuirse las consultas, es posible disminuir el tiempo de respuesta, ya sean operaciones de lectura u operaciones de escritura. Solo sería necesario establecer bajo que parámetros se realizarían estas operaciones al especificar un prototipo de documento, el cual tendría una

estructura tan flexible y dinámica, como los documentos de la colección lo permitan.

El modelo NoSQL si bien no cuenta con un diseño que podría llamarse normalizado, si requiere un análisis similar, puesto que al no requerir de relacionamiento entre entidades, o colecciones para su caso, si requiere que el documento contenga toda la información relevante que pudiera necesitarse o que el mismo negocio necesita, ya que con esto, es posible meter datos redundantes de utilidad, es decir, datos distribuidos en múltiples colecciones, pero utilizados en ambos.

Todas las funciones y operaciones que se realizan normalmente en una base de datos relacional pueden realizarse en MongoDB aunque de una manera diferente y únicamente sobre una colección a la vez.

Una consulta permite especificar campos que pueden existir o no en ciertos documentos, así como una inserción o actualización, permitirán tanto agregar, como modificar e incluso anexar datos, si es que el atributo lo permite.

Muchas otras funciones se realizan especificando el tipo de operaciones posibles para cada tipo de datos posible dentro de un documento, donde los tipos numéricos, fechas y texto siguen conservando y a su vez añaden funcionalidades que representan un tiempo y una carga de procesamiento mayor en comparación a un modelo relacional.

El manejo de archivos es más transparente por medio de **GridFS**, a diferencia de una base de datos relacional, en donde el archivo se almacena de un CLOB, BLOB, entre otros, los archivos se seccionan en pedazos más pequeños y se van insertando en documentos binarios, sin que dejen de tratarse como una unidad.

Estos documentos binarios se almacenan dentro de una misma colección *bucket* designada específicamente a datos binarios y a metadatos, sin que sea requerida configuración extra, más que la misma configuración del tamaño de los documentos en MongoDB.

La cuestión radica en algunas cuestiones de transaccionalidad, ya que MongoDB no tiene dichas funcionalidades aun maduras, aunque por medio de programación, es posible diseñar e implementar mecanismos diferentes al utilizando en el presente trabajo, que es la confirmación de dos fases, por lo que en algunos aspectos siguen haciendo fuertes al modelo relacional.

INGENIERIA
F I

Capítulo 5

Pruebas y análisis de resultados

5. Pruebas y análisis de resultados.

El ciclo de pruebas de un software es un proceso, o serie de procesos, diseñado para asegurarse que los componentes de un sistema codificado realizan la tarea específica para la que fueron diseñados y que no hacen alguna tarea que no está prevista; además de asegurar que el sistema de software sea predecible, consistente y sin comportamientos inesperados para los usuarios. (Myers, Sandler, & Badgett, 2012)

> *“Realizar pruebas es el proceso de ejecutar un programa con la intención de encontrar errores.”* (Myers, Sandler, & Badgett, 2012)

Un caso de prueba es un conjunto de entradas de prueba, condiciones de ejecución y resultados esperados para un objetivo en particular como puede ser explorar los posibles caminos particulares que puede seguir un programa o verificar el cumplimiento de algún requerimiento específico (IEEE, 1990).

5.1. Diseño de pruebas unitarias e integrales.

Antes de iniciar con el funcionamiento del SAI, debe ser introducida la información de los catálogos, en primer lugar se debe insertar los usuarios, junto con la área a la que corresponden.

Posteriormente se deben cargar los componentes en las cuatro áreas del SAI, y ligarse con el usuario responsable del componente. Opcionalmente se pueden agregar las bitácoras de la aplicación, este paso se detallara más adelante.

Todas las pruebas realizadas en este capítulo se realizarán en el siguiente equipo:

- Servidor: Toshiba Satellite L745
- Procesador: Intel Core i3 a 2.3 GHz
- Memoria RAM de 4 GB.
- Disco duro de 300GB
- Servidor de Aplicaciones: Tomcat 7.0
- Servidor de base de datos: MongoDB 2.6.0 de 64 bits.

5.1.1. *Pruebas unitarias.*

En un proceso de desarrollo de software, las pruebas unitarias plantean probar la unidad más pequeña de un sistema asociado a una unidad de código que representa un requerimiento, adicionalmente a la comparación de sus funciones contra las especificaciones funcionales o interfaces que lo definen.

Las unidades que hacen uso o llaman a otras unidades se encuentran en un nivel más alto del programa, mientras que los que son llamadas por otra unidad en un nivel más alto, son llamadas submódulos.

Existen dos tipos metodologías para realizar pruebas unitarias:

a) Black-Box Testing (Pruebas de Caja Negra por su traducción al español). También conocidas como pruebas funcionales, son el tipo de pruebas que ignoran los mecanismos internos de un sistema o componente y se enfocan solamente en los resultados y en las condiciones de ejecución(IEEE, 1990).

b) White-Box Testing (Pruebas de Caja Blanca por su traducción al español). También conocidas como pruebas estructurales, son el tipo de pruebas que consideran al mecanismo interno de un sistema o componente por lo que permite tener visibilidad completa del funcionamiento interno del software, especialmente de la lógica y la estructura del código. (IEEE, 1990).

Para la realización de las pruebas unitarias del SAI, se emplea la metodología White-Box Testing debido a que se conoce y es necesario analizar los mecanismos propios de cada módulo del sistema para determinar si los resultados obtenidos, son los resultados esperados.

Para las pruebas unitarias, se hará uso de una entidad de negocio que contendrá los datos de prueba. Dicho objeto se empleará para validar el correcto funcionamiento de una unidad de código. Código Java 5-1.

El objeto incidencia es utilizado en cada método en las pruebas unitarias, de manera que será omitida la declaración de dicho objeto en métodos subsecuentes, con la finalidad de obtener una mejor visibilidad del código.

```
Incidencia incidencia = new Incidencia();
EventoIncidencia evento = new EventoIncidencia();
Aplicacion aplicacion = new Aplicacion();

incidencia.setColeccion("incidencia");
incidencia.setIdentificador((new Date()).getTime());
incidencia.setNombre("Objeto Incidencia");
incidencia.setDescripcion("Objeto para pruebas unitarias.");
incidencia.setEstatus("Cerrado");
incidencia.setFechaApertura(new Date());
incidencia.setFechaCierre(new Date());
incidencia.setTags(new String[]{"pruebas"});

evento.setIdentificador(1L);
evento.setDetalle("Evento de prueba");
evento.setDescripcion("Descripcion del evento de prueba.");
evento.setFechaCreacion(new Date());
evento.setIdentificadorUsuario(1234567L);

incidencia.setEventos(new EventoIncidencia[]{evento});

aplicacion.setIdentificador(987654L);

incidencia.setAplicacion(aplicacion);
```

Código Java 5-1: Objeto básico a utilizar en las pruebas unitarias.

Durante el proceso de pruebas unitarias se utilizaran tanto el objeto incidencia, como su representación en JSON. Fue utilizado el Código Java 5-2 para realizar la conversión del objeto a una cadena valida JSON, junto con las clases DateSerializer y DateDeserializer para una conversión apropiada de fechas.

```java
Gson gson = new GsonBuilder()
.setPrettyPrinting()
.registerTypeAdapter(Date.class, new DateSerializer())
.registerTypeAdapter(Date.class, new DateDeserializer())
.create();
String json = "";

logger.info("Conversion de objeto Incidencia a JSON.");

json = gson.toJson(incidencia, Incidencia.class);

Assert.assertTrue(!json.isEmpty());

logger.info(json);
```

Código Java 5-2: Conversión del objeto incidencia a una cadena válida JSON.

Finalmente, Documento JSON 5-1 muestra la representación JSON del objeto incidencia que será insertado en la base de datos.


```json
"identificador": 1399009852509,
"nombre": "Objeto Incidencia",
"descripcion": "Objeto para pruebas unitarias.",
"fechaApertura": {
    "$date": "2014-05-02T00:50:52.509Z"
},
"fechaCierre": {
    "$date": "2014-05-02T00:50:52.509Z"
},
"estatus": "Cerrado",
"eventos": [
{
    "identificador": 1,
    "detalle": "Evento de prueba",
    "descripcion": "Descripcion del evento de prueba.",
    "fechaCreacion": {
        "$date": "2014-05-02T00:50:52.509Z"
    },
    "identificadorUsuario": 1234567
    }
],
"tags": [
    "pruebas"
],
"coleccion": "incidencia",
"aplicacion": {
    "identificador": 987654
}
```


Documento JSON 5-1: Documento JSON del objeto incidencia para pruebas.

Para la realización de las pruebas unitarias, será utilizado JUnit, este framework permite realizar pruebas y desarrollar código de manera simultánea, es decir, se pueden implementar los métodos y probarlos según sean requeridos.

5.1.1.1. *Prueba unitaria PU01 - Validación de Incidencia.*

Identificador	PU01
Titulo	Validar Incidencia
Funcionalidad	Validación de un objeto Incidencia
Descripción	Se evalúan los atributos requeridos del objeto incidencia.

Caso de Prueba 5-1: Prueba Unitario 01, Validación de los atributos de una Incidencia.

Previo a cualquier operación dentro del SAI, es necesario que objeto **incidencia** contenga los atributos mínimos requeridos para poder ser utilizado como tal en la base de datos, en caso contrario, se envía error.

Para iniciar este proceso, es necesario asignar un valor a todos los atributos de un objeto de la clase `Incidencia`, este objeto almacenara tanto sus propios atributos, como los de sus elementos más completos, como los eventos y componentes, representados por las clases `EventoIncidencia` y `Aplicacion`.

Los atributos más relevantes de cada uno de los objetos **incidencia**, **evento** y **aplicacion** son validados, y en caso de que alguno de ellos no cumple con la validación, se enviara una excepción y la prueba habrá fallado.

```java
@Test
public void validaIncidencia()
{
    Incidencia incidencia = new Incidencia();
    EventoIncidencia evento = new EventoIncidencia();
    Aplicacion aplicacion = new Aplicacion();

    incidencia.setColeccion("incidencia");
    incidencia.setIdentificador((new Date()).getTime());
    incidencia.setNombre("Objeto Incidencia");
    incidencia.setDescripcion("Objeto para pruebas unitarias.");
    incidencia.setEstatus("Cerrado");
    incidencia.setFechaApertura(new Date());
    incidencia.setFechaCierre(new Date());
    incidencia.setTags(new String[]{"pruebas"});

    evento.setIdentificador(1L);
    evento.setDetalle("Evento de prueba");
    evento.setDescripcion("Descripcion del evento de prueba.");
    evento.setFechaCreacion(new Date());
    evento.setIdentificadorUsuario(1234567L);

    incidencia.setEventos(new EventoIncidencia[]{evento});

    aplicacion.setIdentificador(987654L);

    incidencia.setAplicacion(aplicacion);

    Assert.assertFalse(incidencia.getIdentificador() == null);
    Assert.assertFalse(incidencia.getNombre() == null);
    Assert.assertFalse(incidencia.getDescripcion() == null);
    Assert.assertFalse(incidencia.getEstatus() == null);
    Assert.assertFalse(incidencia.getFechaApertura() == null);

    if (incidencia.getFechaCierre() != null)
    {
        Assert.assertFalse(incidencia.getFechaApertura()
                .before(incidencia.getFechaCierre()));
    }

    if (incidencia.getEventos() != null)
    {
        for (EventoIncidencia evt : incidencia.getEventos())
        {
            Assert.assertFalse(evt == null);
            Assert.assertFalse(evt.getIdentificador() == null);
            Assert.assertFalse(evt.getDetalle() == null);
            Assert.assertFalse(evt.getDescripcion() == null);
            Assert.assertFalse(evt.getFechaCreacion() == null);
        }
    }
}
```

Código Java 5-3: Código de validación del objeto incidencia.

5.1.1.2. *Prueba unitaria PU02 - Inserción de Incidencia.*

Identificador	PU02
Titulo	Insertar Incidencia
Funcionalidad	Inserción de un objeto Incidencia
Descripción	Se inserta el objeto incidencia dentro de la base de datos.

Caso de Prueba 5-2: Prueba Unitaria 02, Inserción del objeto incidencia.

El método de inserción en Código Java 5-4 se encarga de enviar el objeto incidencia al DAO, convirtiéndolo a JSON para finalmente antes de ser insertarlo.

Es requerido un objeto `Incidencia`, además de un objeto `EventoIncidencia` y un componente asociado, en este caso, un objeto `Aplicacion`.

Se asignan valores a cada uno de los atributos de todos los objetos, para finalmente, agregar al objeto **incidencia**, los objetos **evento** y **aplicacion**.

Nuevamente, se especifican los valores utilizados para cada uno de los objetos en la prueba anterior.

```java
@Test
public void insertaIncidencia() throws RuntimeException
{
    Incidencia incidencia = new Incidencia();
    EventoIncidencia evento = new EventoIncidencia();
    Aplicacion aplicacion = new Aplicacion();

    incidencia.setColeccion("incidencia");
    incidencia.setIdentificador((new Date()).getTime());
    incidencia.setNombre("Objeto Incidencia");
    incidencia.setDescripcion("Objeto para pruebas unitarias.");
    incidencia.setEstatus("Cerrado");
    incidencia.setFechaApertura(new Date());
    incidencia.setFechaCierre(new Date());
    incidencia.setTags(new String[]{"pruebas"});

    evento.setIdentificador(1L);
    evento.setDetalle("Evento de prueba");
    evento.setDescripcion("Descripcion del evento de prueba.");
    evento.setFechaCreacion(new Date());
    evento.setIdentificadorUsuario(1234567L);

    incidencia.setEventos(new EventoIncidencia[]{evento});

    aplicacion.setIdentificador(987654L);

    incidencia.setAplicacion(aplicacion);

    Gson gson = new GsonBuilder()
    .setPrettyPrinting()
    .registerTypeAdapter(Date.class, new DateSerializer())
    .registerTypeAdapter(Date.class, new DateDeserializer())
    .create();
    String msg = null;
    Result wr = null;

    incidenciaCtrl = new IncidenciaServiceImpl();

    incidenciaCtrl.creaIncidencia(incidencia);

    wr = gson.fromJson(msg, Result.class);

    if (wr.getErr() != null)
        throw new RuntimeException(wr.getErr());
}
```

Código Java 5-4: Inserción del objeto incidencia.

5.1.2. *Pruebas integrales.*

Las pruebas integrales pueden interpretarse como la extensión de las unitarias, debido a que se prueba la unión entre una o más unidades previamente probadas de manera unitaria. Con este tipo de pruebas se identifican los posibles problemas cuando las unidades son combinadas.

Existen tres estrategias comunes para realizar pruebas integrales:

- La estrategia Top-Down (Arriba-Abajo en su traducción al español) comienza por el módulo que se encuentra en el nivel más alto, es decir, el módulo de mayor nivel es integrado y probado primero. Para probar el siguiente módulo, la única consideración a tomar es que el siguiente módulo debe tener al menos un submódulo que ya debió haber sido probado.

  La principal ventaja de esta estrategia es que se enfoca en el primer nivel del sistema que suele ser uno de los más importantes del mismo: la interface de usuario, con lo cual se puede establecer la lógica de alto nivel y el flujo de datos que pueden ser probados tempranamente en el proceso.

  Esta estrategia puede requerir la implementación de muchos subsistemas de prueba, especialmente si los niveles más bajos del sistema contiene un número alto de unidades funcionales; así también si los niveles más bajos se crean mientras los niveles altos

se completan, los cambios relevantes que puedan mejorar la funcionalidad del sistema mediante la modificación de las funcionalidades core o esenciales, contenidas en su mayoría en los niveles más bajos, pueden ser ignoradas.

Finalmente cuando los niveles más bajos del sistema son añadidos, las capas superiores necesitan ser probadas nuevamente.

- La estrategia Bottom-Up (Abajo-Arriba en su traducción al español) comienza con los módulos terminales de un programa, que son aquellos que ya no llaman a otros módulos, siendo la única regla para seguir el proceso de prueba a otros módulos, es que el módulo a ser elegido debe tener todos sus módulos subordinados probados con anterioridad.

  Esta estrategia tiene la principal desventaja que la interface de usuario es probada hasta el final del proceso, lo cual puede provocar desviaciones en el flujo de datos, sin embargo, proporciona una visión más concreta cuando las fallas del sistema se dan en los niveles más bajos del sistema, lo que facilita la corrección de cambios mayores en la parte sustancial del mismo en una etapa temprana de desarrollo.

- La tercera estrategia requiere pruebas a través de los datos funcionales y de los caminos de control de flujo. En esta estrategia las entradas de las funciones son integradas por el patrón Bottom-Up

y los resultados para cada función son integrados por el patrón Top-Down.

En las pruebas integrales del SAI, se empleará la estrategia Top-Down debido a la planeación de la codificación del mismo, que parte de módulos de alto nivel, estos a su vez van llamando submódulos que realizan especializaciones de las operaciones solicitadas.

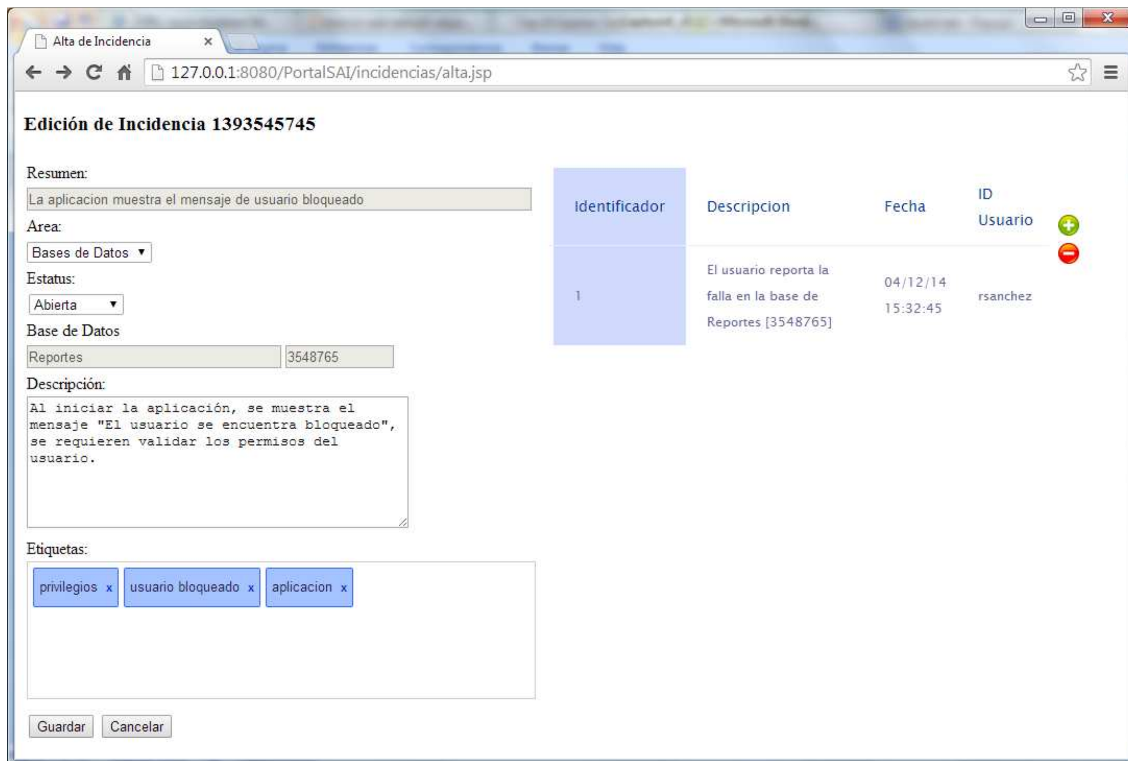
Aunque la forma en que el SAI fue desarrollado es ampliamente flexible para utilizar cualquiera de las estrategias descrita, se ha elegido la Top-Down porque todos sus módulos se encuentran creados y operando en coordinación, lo cual favorecerá la prueba general del sistema.

5.1.2.1. *Prueba integral PI01 - Creación de una incidencia.*

Identificador	PI01
Titulo	Crear Incidencia
Condiciones preliminares	El usuario Administrador General debe existir en el SAI y contar con una sesión iniciada con sus credenciales dentro del sistema.
Funcionalidad	Crear una incidencia en el SAI.
1. Seleccionar el menú Incidencia	
2. Seleccionar Alta de Incidencia	
3. Proporcionar la información requerida para el levantamiento de la Incidencia (nombre, descripción, usuario que reporta, entre otros)	
4. Guardar por medio del botón Aceptar la incidencia dentro del SAI	
Resultado	Se crea una nueva incidencia en conjunto a los eventos correspondientes.

Caso de Prueba 5-3: Prueba Integral 01

En la Interfaz de Usuario 5-1 se muestra cómo el usuario puede dar de alta una incidencia de acuerdo a los parámetros solicitados. En el Documento JSON 5-2 es posible observar cómo el sistema ha procesado la nueva incidencia.



Interfaz de Usuario 5-1: Alta de una incidencia.

```
"identificador" : 1397451226000,
"nombre" : "La aplicacion muestra el mensaje usuario bloqueado",
"descripcion" : "Al iniciar la aplicación, se muestra el mensaje \"El usuario se encuen
"coleccion" : "incidencia",
"estatus" : "Abierta",
"fechaApertura" : {
    "$date" : "2014-04-13T14:34:54.678Z"
},
"eventos" : [
    {
        "identificador" : 1397451232000,
        "detalle" : "Alta",
        "descripcion" : "El usuario rsanchez reporte la falla en la base de reportes",
        "identificadorUsuario" : 475,
        "fechaCreacion" : {
            "$date" : "2014-04-13T14:35:26.365Z"
        }
    }
],
"baseDatos" : {
    "identificador" : 2534678,
    "nombre" : "BDReportes"
}
```


Documento JSON 5-2: Documento de Incidencia creado.

Como resultado de la PI01 se obtiene que el SAI crea con éxito la incidencia nueva.

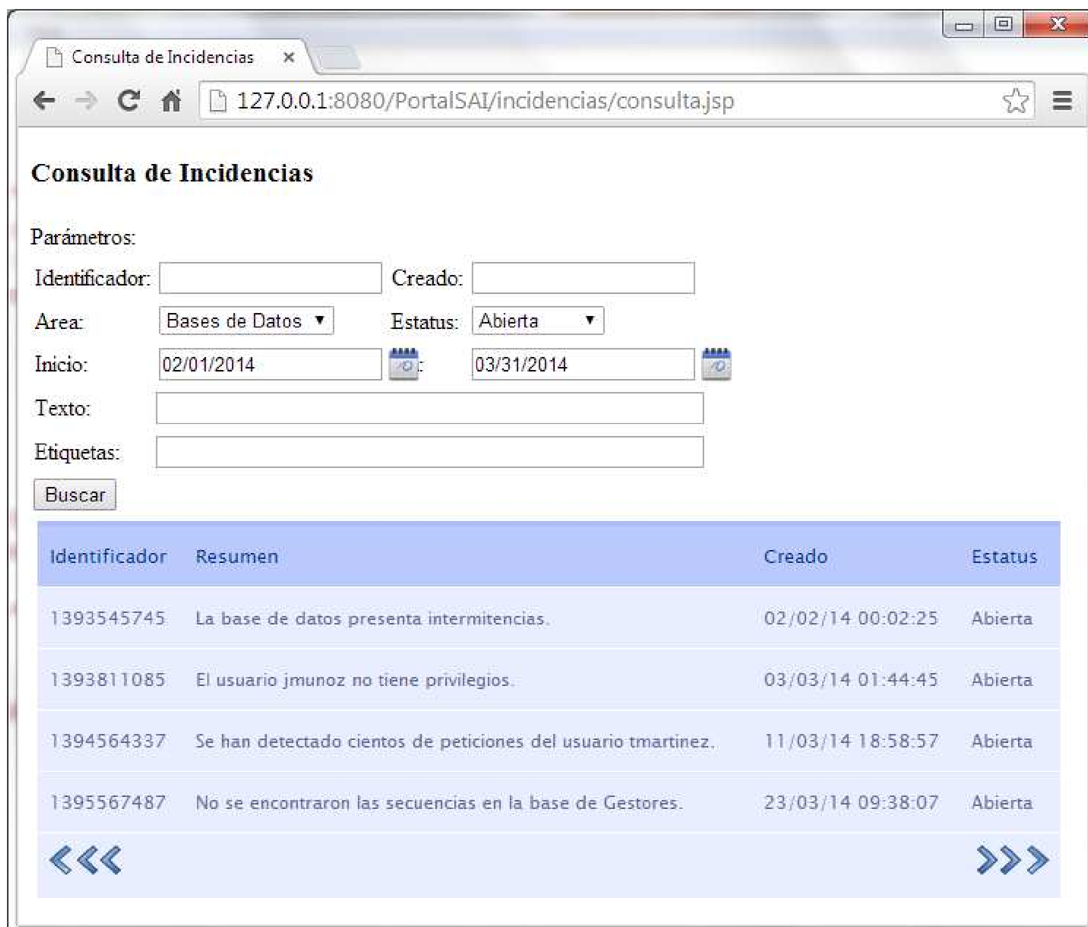
5.1.2.2. *Prueba integral PI02 - Consulta de Incidencia.*

Identificador	PI02
Titulo	Consultar Incidencias
Condiciones preliminares	Debe existir alguna incidencia previa en el SAI. El usuario que ejecute la consulta, debe contar con una sesión iniciada con sus credenciales dentro del sistema.
Funcionalidad	Consulta de incidencia.
1. Seleccionar el menú Incidencia	
2. Seleccionar Consulta de Incidencia	
3. Proporcionar la información necesaria para la recabación de la o las incidencias requeridas.	
4. Si se encuentra la o las incidencias solicitadas, se despliega un resumen con la información más importante de las mismas.	
5. Si se desea mayor información, el usuario puede seleccionar la incidencia de su interés.	
6. Si es seleccionada alguna incidencia en particular, se muestra el detalle de la	

misma, si no es así, el usuario puede cerrar su sesión del SAI.	
Resultado	Se despliega una lista con la o las incidencias encontradas dentro del SAI.

Caso de Prueba 5-4: Prueba Integral 02

En la Interfaz de Usuario 5—2 se muestra cómo el usuario tiene que completar la información necesaria para que el SAI recupere la información deseada, por lo que en el Documento JSON 5-3 se puede observar el tratamiento interno que MongoDB a la consulta, debido a que este documento se muestran los filtros que fueron establecidos por el usuario a través de la interfaz de usuario.



Interfaz de Usuario 5-2: Consulta de incidencias.

```
{
    "coleccion" : "incidencia",
    "estatus" : "Abierta",
    "baseDatos" : { "$exists" : true }
}
```

Documento JSON 5-3: Filtros para realizar la consulta.

Como resultado de la PU02 se obtiene el listado esperado de las incidencias que se encuentran con un estatus de “Abierta” pertenecientes a los componentes “Bases de datos” del SAI.

Los filtros no son restrictivos a los dos incluidos en el caso de prueba.

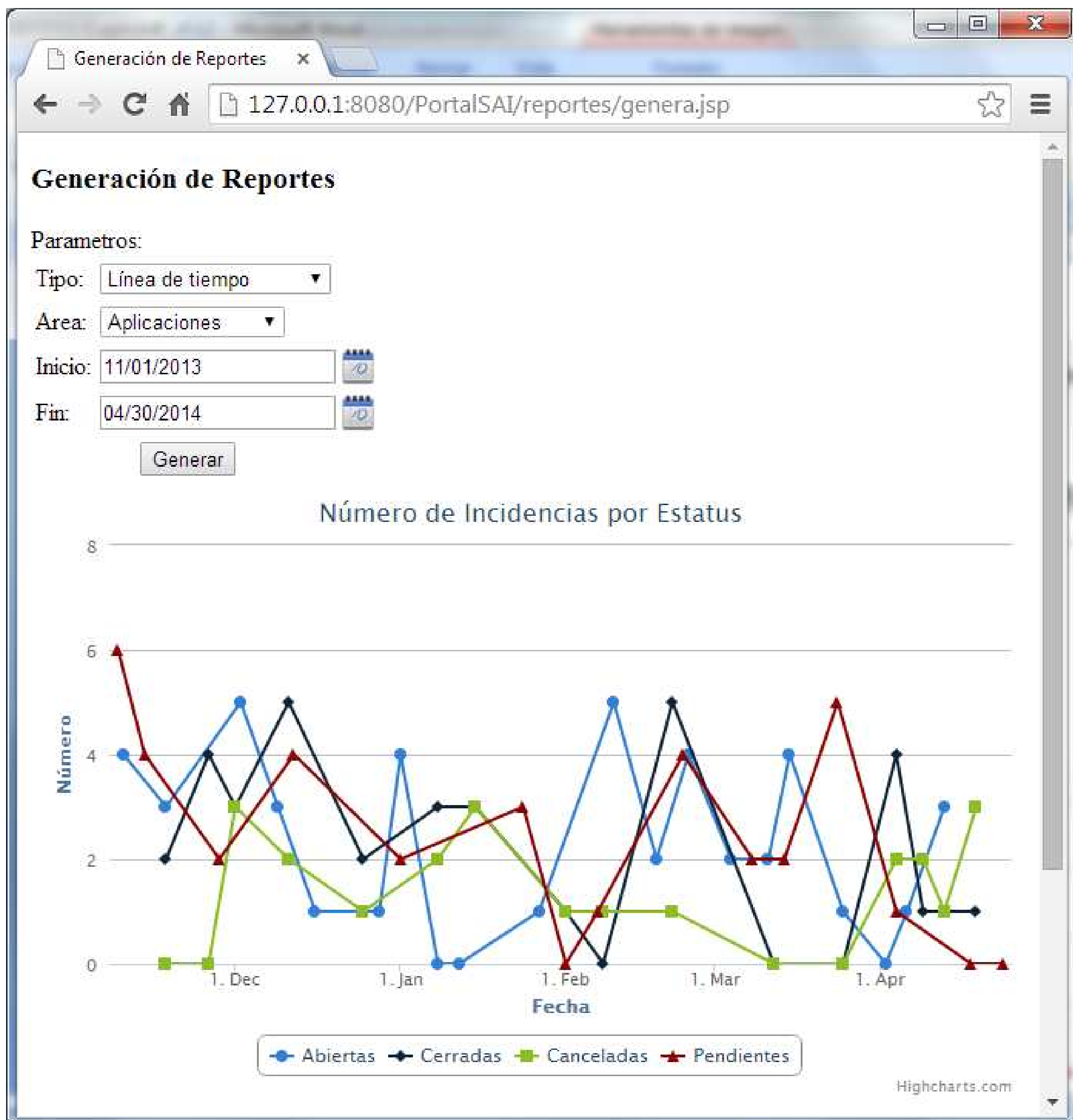
5.1.2.3. *Prueba unitaria PU03.*

<table>
<tr><th>Identificador</th><th>PU03</th></tr>
<tr><td>Titulo</td><td>Generar Reporte</td></tr>
<tr><td>Condiciones preliminares</td><td>Debe existir un conjunto de incidencias previas en el SAI.
El usuario que ejecuta la generación del reporte, debe contar con una sesión iniciada con sus credenciales dentro del sistema.</td></tr>
<tr><td>Funcionalidad</td><td>Generación de Reporte</td></tr>
<tr><td colspan="2">1. Seleccionar el menú Reporte</td></tr>
<tr><td colspan="2">2. Seleccionar el tipo de reporte (1 de 3) que se desea generar</td></tr>
<tr><td colspan="2">3. Proporcionar los parámetros necesarios para la generación del reporte como pueden ser: rango de fechas, área de interés y/o etiquetas</td></tr>
<tr><td colspan="2">4. Generar reporte por medio del botón Generar</td></tr>
<tr><td colspan="2">5. Despliegue de la información válida para el reporte solicitado</td></tr>
<tr><td colspan="2">6. Si el usuario lo requiere, el reporte se puede exportar a una hoja de cálculo utilizando el botón de Exportar o imprimirlo con el botón de Imprimir, de lo contrario, el usuario puede cerrar sesión del SAI.</td></tr>
<tr><td>Resultado</td><td>Se muestra el contenido del reporte requerido con las opciones de Exportar o Imprimir el mismo.</td></tr>
</table>

Caso de Prueba 5-5: Prueba Unitaria 03

Para la generación de reportes se emplea una sola interfaz de usuario en la cual se elige el tipo de reporte solicitado.

En la Interfaz de usuaro 5--3se muestra el resultado arrojado por el primer tipo de reporte, el cual indica el número de incidencias a través del tiempo.



Interfaz de Usuario 5-3: Reporte por línea de tiempo de incidencias.

El Código JavaScript 5-1 muestra el manejo interno que el SAI da al tipo de reporte elegido, empleando funciones map-reduce específicas para cada tipo de gráfica.

```
//funcion map
function ()
{
        var date = this.fechaApertura;
        var fecha = ('0' + date.getDate()).slice(-2)+'/';
        fecha += ('0' + (date.getMonth() + 1)).slice(-2)+'/';
        fecha += date.getFullYear();

        emit(fecha, {conteo:1} );
};

//funcion reduce
function(keys, values)
{
        var total = 0;

        for (var i in values)
        {
                total += values[i].conteo;
        }

        return {conteo:total};
};
```

Código JavaScript 5-1: Funciones map/reduce para generar reporte por estatus y área.

El siguiente tipo de reporte, promedio de días de resolución, muestra el número de incidencias en promedio resueltas por cada área. En la Interfaz de Usuario 5-4 los resultados de este reporte pueden ser observados.

El Código JavaScript 5-2 representa el tratamiento que el SAI realiza para la generación del tipo de reporte específico

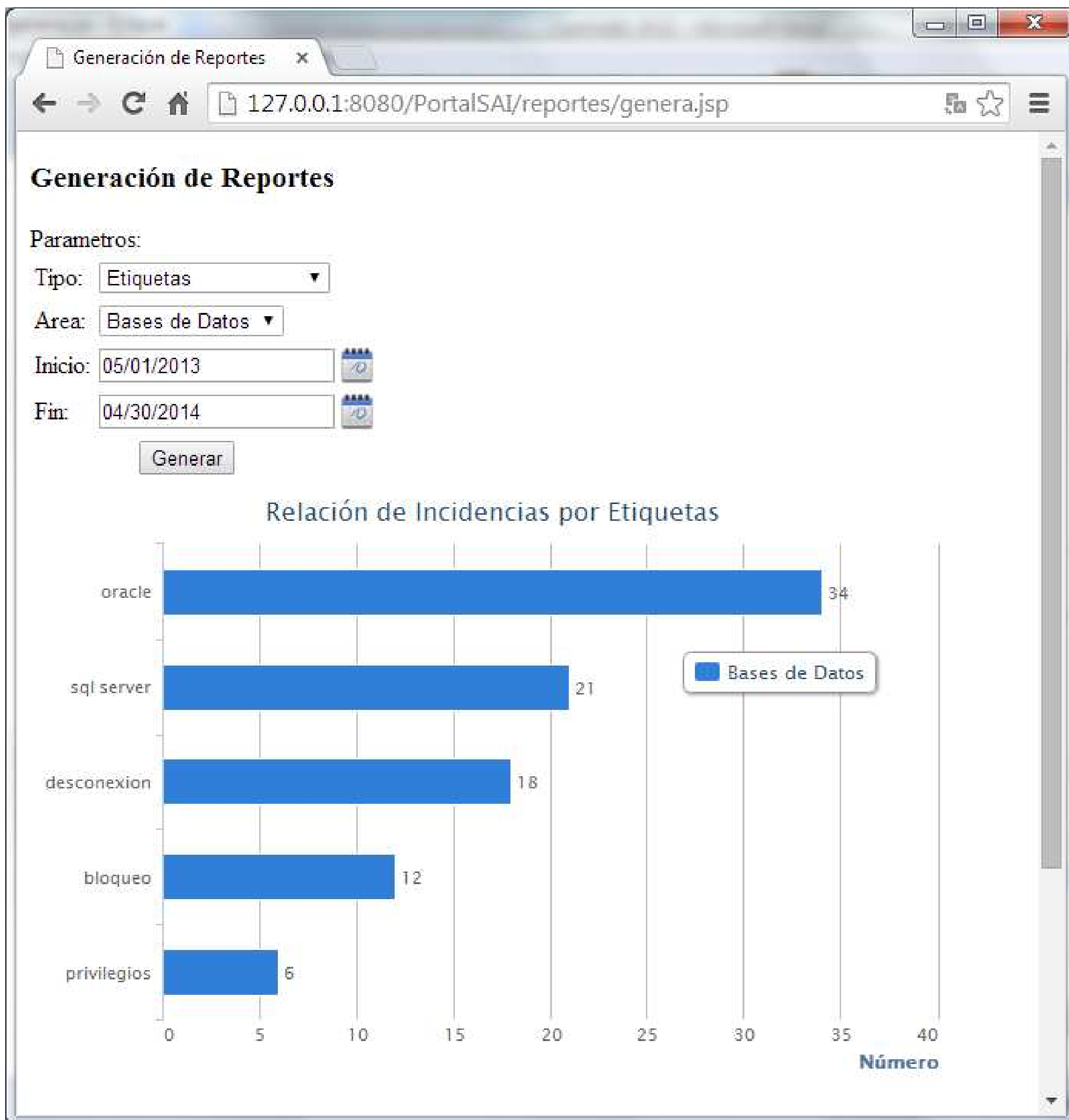
Interfaz de Usuario 5-4: Reporte por promedio de días de resolución.

```
//funcion map
function ()
{
        var apertura = parseDate(this.fechaApertura);
        var cierre = parseDate(this.fechaCierre);
        var promedio = (cierre - apertura) / (1000 * 60 * 60 * 24);
        var mes = ('0' + (apertura.getMonth() + 1)).slice(-2);

        emit(mes, {suma : promedio} );
};

//funcion reduce
function(keys, values)
{
        var total = 0;

        for (var i in values)
        {
                total += values[i].conteo;
        }

        return {conteo:total};
};
```

Código JavaScript 5-2: Funciones map/reduce para generar reporte por promedio de resolución.

En la Interfaz de Usuario 5-5 se observa el resultado de la generación del reporte por etiquetas, en el cuál se obtiene el número de etiquetas en determinado periodo de tiempo proporcionado por el usuario.

El Código JavaScript 5-3 representa el tratamiento que el SAI le da a la petición del reporte por etiquetas.



Interfaz de Usuario 5-5: Reporte por etiquetas.

```
//funcion map
function ()
{
    if (!this.etiquetas)
    {
        return;
    }

    for (indice in this.etiquetas)
    {
        emit(this.etiquetas[indice], 1);
    }
};

//funcion reduce
function(anterior, actual)
{
    var conteo = 0;

    for (indice in actual)
    {
        conteo += actual[indice];
    }

    return conteo;
};
```

Código JavaScript 5-3: Funciones map-reduce para generar reporte por etiquetas.

5.1.2.4. *Prueba integral PI03 - Edición de incidencia.*

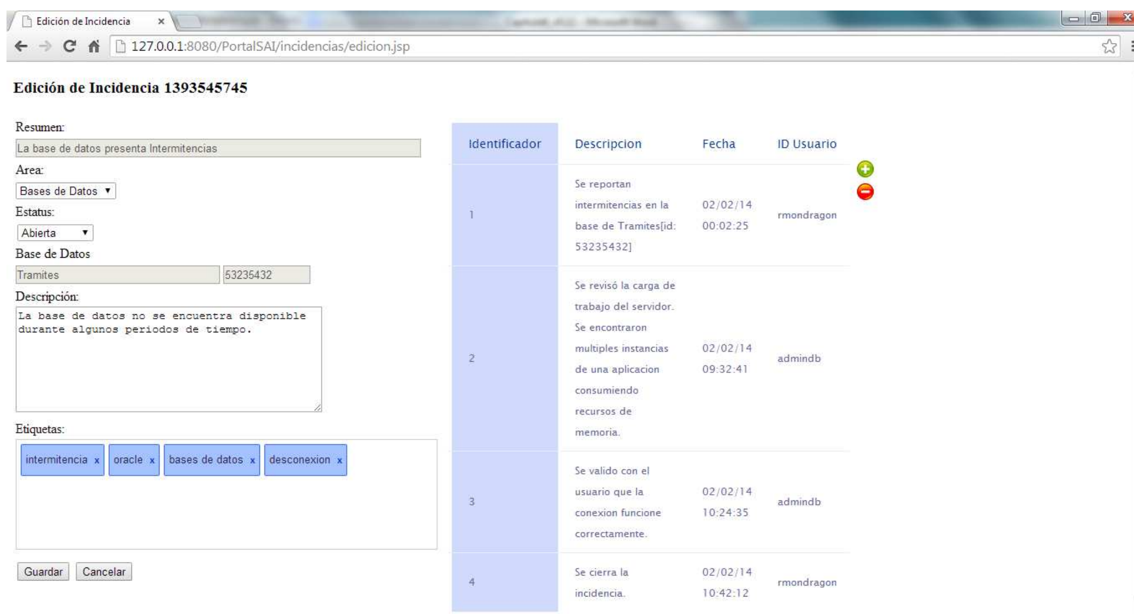
Identificador	PI03
Titulo	Editar Incidencia
Condiciones preliminares	Debe existir al menos una incidencia previa en el SAI. El Administrador General debe contar con una sesión iniciada con sus credenciales dentro del sistema.
Funcionalidad	Edición de incidencia
1. Seleccionar el menú Incidencia	
2. Seleccionar el submenú Edición de Incidencia	
3. Especificar el identificador de la Incidencia que se desea editar.	
4. Se despliega la información relacionada de la Incidencia	
5. El SAI agrega nueva información relevante como un nuevo Evento de Incidencia de acuerdo a la operación que se ejecute sobre la incidencia (Cancelación, Cierre o agregar información a la misma) agregando detalles como el usuario que la	

<table>
<tr><td colspan="2">ejecuto, fecha, hora y descripción del evento.</td></tr>
<tr><td colspan="2">6. Guardar todos los cambios realizados por medio del botón Guardar.</td></tr>
<tr><td colspan="2">7. El usuario puede cerrar sesión del SAI.</td></tr>
<tr><td>Resultado</td><td>La incidencia contiene nuevos elementos asociados que aportan información de las operaciones realizadas sobre la misma.
Puede resultar en el cierre o cancelación de la incidencia.</td></tr>
</table>

Caso de Prueba 5-6: Prueba de Integración 03

La edición de una Incidencia es bastante sencilla, únicamente son editables algunos atributos, como lo son: Estatus, Descripción de la Incidencia y Eventos. Al agregar un evento, se deben especificar un Usuario y una Descripción, el identificador y la fecha serán agregados de manera automática al agregar un evento.

En Interfaz de Usuario 5-6 se muestra la interfaz para editar una incidencia. Los elementos propios de la incidencia se especifican en la parte izquierda, mientras que los eventos se detallan en la parte izquierda de la página.



Interfaz de Usuario 5-6: Edición de incidencia.

5.1.2.5. *Prueba Integral PI04 - Edición de componente*

Identificador	PI04
Titulo	Editar Componente
Condiciones preliminares	Debe existir al menos un componente previo en el SAI. Se debe contar con acceso a la ubicación lógica de la bitácora que debe encontrarse en el formato requerido por el SAI. El Administrador General y/o Administrador de Componente debe contar con una sesión iniciada con sus credenciales dentro del sistema.
Funcionalidad	Edición de componente con agregación de bitácora
1. Seleccionar el menú Componente	
2. Seleccionar el submenú Edición de Componente	
3. Especificar el identificador del Componente que se desea editar.	
4. Se despliega la información relacionada del Componente	
5. Agregar bitácora por medio del botón Bitácora Nueva.	
6. Indicar la ubicación lógica del archivo de bitácora.	
7. Actualizar alguna otra información del Componente si es requerido.	
8. Guardar todos los cambios realizados por medio del botón Guardar.	
9. Comienza el proceso interno de ingesta de la bitácora	

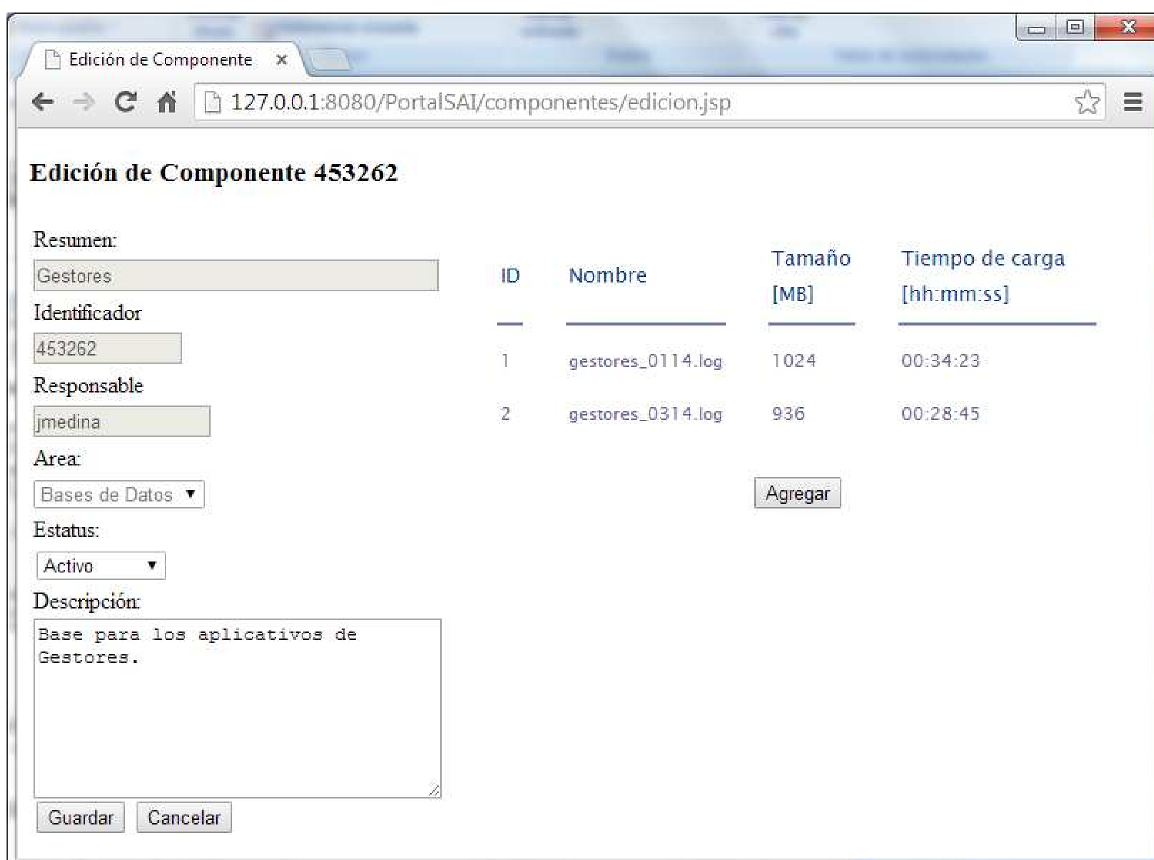
10. Se debe esperar el mensaje de éxito o falla del proceso de ingesta.	
11. El usuario puede cerrar sesión del SAI.	
Resultado	El Componente contará con una o más bitácoras asociadas y con la edición completa de alguna característica requerida.

Caso de Prueba 5-7: Prueba de Integración 02

Para editar componentes, la página Interfaz de Usuario 5-7 se muestra la información más relevante, únicamente podrán modificarse los atributos de Estatus, Descripción y agregar Bitácoras

Para el componente *Gestores*, se encuentran actualmente dos archivos de bitácoras almacenados en la base del SAI, junto con atributos adicionales al momento de la carga, como el tamaño del archivo y el tiempo de carga.

Si se desea dar seguimiento al proceso de carga de bitácoras, se deberá seleccionar la opción adecuada en la ventana emergente. Esta última funcionalidad, corresponde a la transaccionalidad, detallada más adelante.



Interfaz de Usuario 5-7: Edición de componente.

5.2. Diseño de pruebas de volumen, concurrencia y transaccionalidad

5.2.1. *Prueba de volumen.*

Las pruebas de volumen son aquellas en las que se somete a las funcionalidades del sistema a manejar cantidades muy grandes de información con la finalidad de verificar su desempeño y comportamiento.

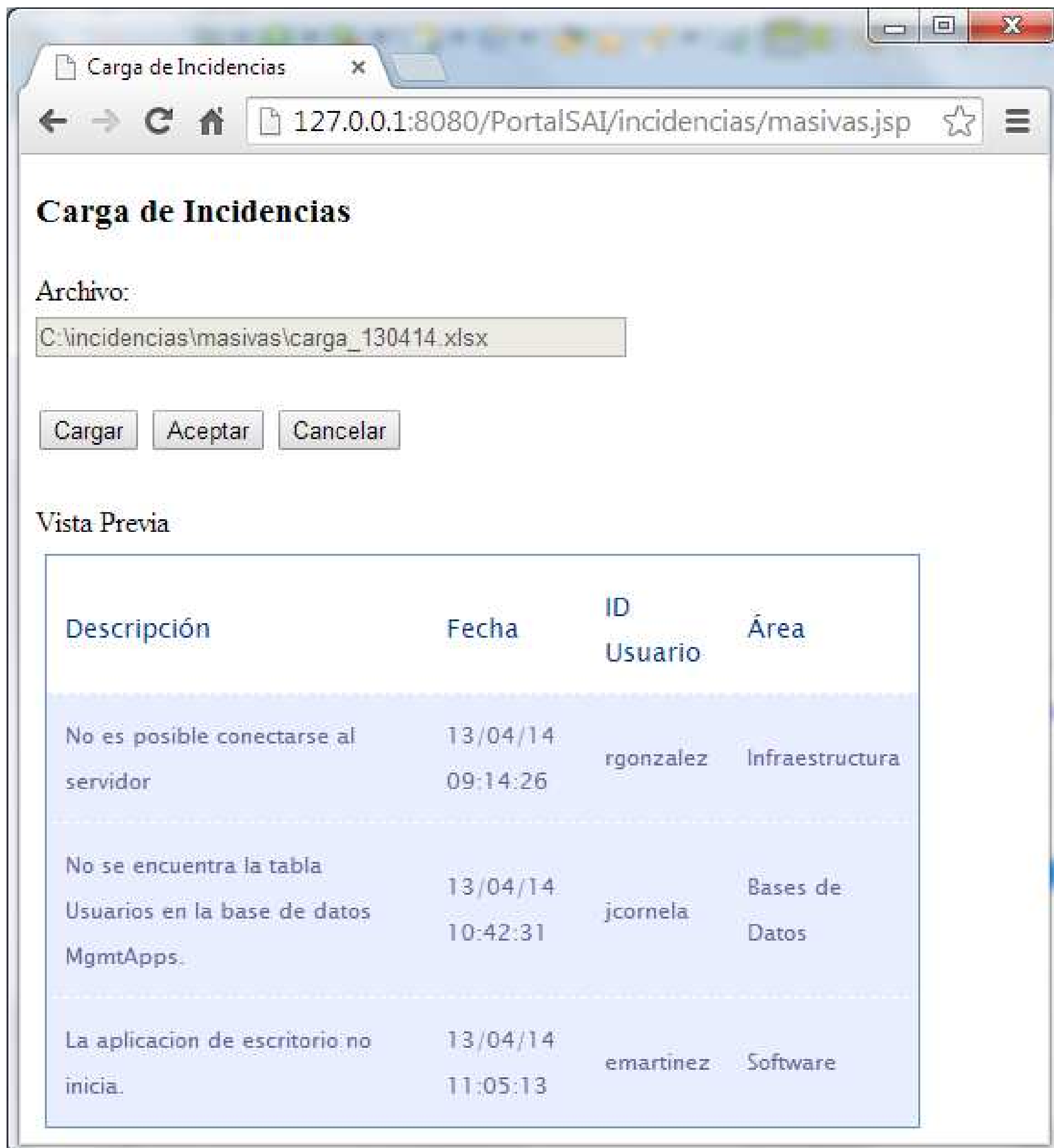
Identificador	PV01
Condiciones preliminares	Se debe contar con una serie de incidencias no registradas en el SAI en una hoja de cálculo. Este archivo solamente puede contener incidencias de un

	mismo día y solamente especificar la incidencia, el componente y el usuario que la reporta. El Administrador General debe contar con una sesión iniciada con sus credenciales dentro del sistema.
Funcionalidad	Creación masiva de Incidencias.
1. Seleccionar el menú Incidencia	
2. Seleccionar el submenú Carga de Incidencias	
3. Especificar la ubicación lógica del archivo con el conjunto de incidencias a ser cargadas en el SAI	
4. El proceso de carga se ejecuta por medio del botón Aceptar	
5. Se despliega el resumen de las incidencias procesadas.	
6. El usuario puede cerrar sesión del SAI.	
Resultado	El SAI crea un conjunto de incidencia en una sola operación sin necesidad de crearlas manualmente.

Caso de Prueba 5-8: Prueba de Volumen 01

Es posible realizar la carga de incidencias desde un archivo, sin embargo, la información que puede almacenarse en el SAI no es completa, hasta que se agreguen eventos a cada una de las incidencias, la plantilla únicamente requiere los atributos Descripción, Fecha, Usuario y Área, como se muestra en Interfaz de Usuario 5-8.

Para la carga de incidencia, se insertaron desde cien hasta un millón de registros, utilizando ambos modelos, el NoSQL y el relacional, con resultados bastantes diferentes, estos resultados se detallan en la sección 6.3, perteneciente al análisis de resultados.



Interfaz de Usuario 5-8: Carga de incidencias.

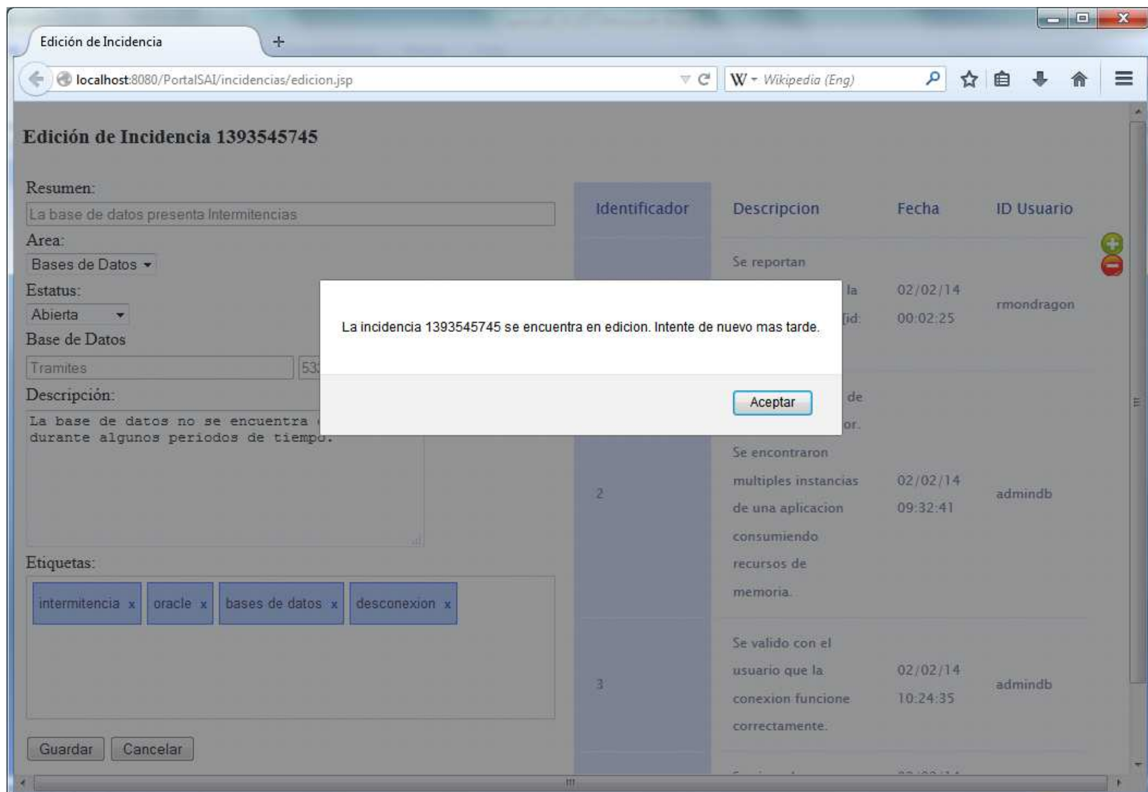
5.2.2. *Prueba de concurrencia.*

Este tipo de pruebas son empleadas para probar la estrategia usada por el SAI para asegurar que las solicitudes sean revisadas y ejecutadas.

Identificador	PC01
Condiciones preliminares	Se debe contar con una incidencia creada en el SAI. El Administrador General debe contar con una sesión iniciada con sus credenciales dentro del sistema. El Administrador de Componente debe contar con una sesión iniciada con sus credenciales dentro del sistema.
Funcionalidad	Dos operaciones de escritura simultáneas de una misma incidencia.
Administrador General	*Administrador de Componente*
1. Seleccionar el menú Incidencia	1. Seleccionar el menú Incidencia
2. Seleccionar el submenú Edición de Incidencia	2. Seleccionar submenú Edición de Incidencia
3. Especificar el identificador de la Incidencia que se desea editar.	3. Especificar el identificador de la Incidencia que se desea editar.
4. Se despliega la información relacionada de la Incidencia.	4. Se despliega la información relacionada de la Incidencia.
5. El SAI agrega nueva información relevante como un nuevo Evento de Incidencia de acuerdo a la operación que se ejecute sobre la incidencia (Cancelación, Cierre o agregar información a la misma) agregando detalles como el usuario que la ejecuto, fecha, hora y descripción del evento.	5. El SAI agrega nueva información relevante como un nuevo Evento de Incidencia de acuerdo a la operación que se ejecute sobre la incidencia (Cancelación, Cierre o agregar información a la misma) agregando detalles como el usuario que la ejecuto, fecha, hora y descripción del evento.
6. Guardar todos los cambios realizados por medio del botón Guardar.	6. Guardar todos los cambios realizados por medio del botón Guardar.
Resultado	Al ejecutarse dos escrituras simultáneas para una misma incidencia, alguna de las dos obtendrá un error en la escritura de la misma debido a que la incidencia se encontrará bloqueada en su edición por la operación que haya sido privilegiada con el primer turno.

Caso de Prueba 5-9: Prueba de Concurrencia 01

El SAI implementa el mecanismo por defecto de bloqueo de MongoDB, el cual bloquea las escrituras concurrentes sobre un documento, sin bloquear las lecturas, hasta que la información sea actualizada. Este bloqueo se realiza con un documento adicional que funciona como un candado sobre el documento.



Interfaz de Usuario 5-9: Mensaje de error en la interfaz gráfica.

De esta manera, la primera conexión que realiza la escritura será privilegiada y sólo hasta que se envían las modificaciones y se recibe el mensaje de confirmación de escritura, se levanta el bloqueo sobre el documento. Esta misma, se encargara de crear el candado para evitar que conexiones adicionales intenten editar su información, denegando la solicitud como en Interfaz de Usuario 5-9 y Código Java 5-5.

```
java.lang.RuntimeException: La incidencia se encuentra bloqueada: 1234567890
	at mx.mongo.dao.IncidenciaDAO.editaIncidencia(IncidenciaDAO.java:426)
	at mx.mongo.demo.Principal.insertaCandado(Principal.java:402)
	at mx.mongo.demo.Principal.main(Principal.java:61)
```

Código Java 5-5: Mensaje de error a nivel programación.

El mecanismo de bloqueo adicional que implementa el SAI, almacena la fecha en que comenzó a editarse la incidencia, su identificador dentro de un documento en la colección **locks**, este documento de bloqueo, se crea con la función en Código Java 5-6.

```java
public void creaCandado(Long identificador)
    throws RuntimeException
{
    Candado lock = new Candado();
    String collection = lock.getColeccion();
    DBCollection db = null;

    lock.setIdentificadorIncidencia(identificador);

    BasicDBObject transBO = new BasicDBObject();
    transBO.put("identificadorIncidencia", lock.getIdentificadorIncidencia());
    transBO.put("identificador", lock.getIdentificador());
    transBO.put("fechaCreacion", lock.getFechaCreacion());

    try
    {
        db = getConexion(collection);

        WriteResult wr = db.insert(transBO);

        if (wr.getError() != null)
            throw new RuntimeException(wr.getError());
    }
    catch (Exception e)
    {
        throw new RuntimeException(e.getMessage());
    }
}
```

Código Java 5-6: Creación del candado en un documento incidencia.

Para que el candado sea eliminado, la primera escritura debe finalizar, ya sea que se envíen los cambios o que se cancele la operación, después de lo cual es posible realizar nuevas operaciones de escritura. Sin embargo, si el usuario no libera el candado, será necesario utilizar índices TTL para asegurar que el documento de bloqueo sea eliminado.

Por medio de índices TTL (acrónimo en inglés para *Time To Live*), es posible especificar que un documento expira de acuerdo a una fecha de vigencia del documento, como se muestra en Documento JSON 5-4.

```
{
    "identificador": 1401645514309,
    "identificadorIncidencia": 1234567890,
    "fechaCreacion": {
        "$date": "2014-06-01T12:58:34.309Z"
    },
    "expireAt": {
        "$date": "2014-06-01T13:06:34.309Z"
    },
    "coleccion": "locks"
}
```

Documento JSON 5-4: Documento de bloqueo con la póliza expireAt:

Especificando el campo `expireAt`, al pasar el tiempo de creación, el documento de bloqueo es eliminado automáticamente por MongoDB, sin la necesidad de interacción con el usuario

Adicionalmente, es posible especificar un índice sobre un campo fecha, donde después de ciertos segundos sean eliminados aquellos documentos que

cumplan su tiempo de vida dentro de la base de datos ejecutando la función en Documento JSON 5-5.

```
db.locks.ensureIndex(
   { "fechaCreacion": 1 },
   { expireAfterSeconds: 480 } )
```

Documento JSON 5-5: Índice de tiempo de vida en segundos sobre el atributo fechaCreacion.

Dichos índices TTL no pueden ser utilizados en conjunto con identificadores de objetos _id asignados por MongoDB, por lo que se utilizará el identificador del de la incidencia.

Debido a que el SAI se encuentra en un solo nodo primario, las escrituras realizadas sobre los documentos pueden verse reflejadas en tiempo real, esto en una arquitectura maestro-esclavos puede requerir un ligero retraso de acuerdo al nivel de escritura configurado con el cliente de MongoDB, aunque dicho retraso puede considerarse despreciable debido a la rapidez con la que MongoDB realiza las escrituras.

Se recomiendan los niveles de bloqueo por defecto en MongoDB de acuerdo al nivel de configuración escritura `WriteConcern`, descrita en el apartado 4.6.4:

a) *ACKNOWLEDGED.* Es la configuración por defecto y sirve para sistemas de un solo servidor. Al realizar una escritura, este envía una notificación cuando se haya realizado correctamente, en caso contrario, envía una excepción.

b) *REPLICA_ACKNOWLEDGED*. Es utilizado en sistemas con múltiples servidores. Se encarga de enviar confirmación una vez que la escritura de datos se haya propagado a dos o más nodos secundarios, según se configure el cliente del SAI.

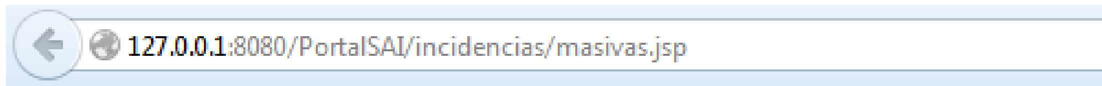
5.2.3. *Prueba de transaccionalidad.*

Con las pruebas de transaccionalidad se busca verificar la estrategia que el sistema emplea para asegurar la consistencia que las operaciones de manejo de datos pueden alcanzar, así como el proceso por el cual las operaciones a ser agrupadas en una transacción, garantizando la inserción todos o ningún documento.

Identificador	PT01
Condiciones preliminares	El Administrador General y/o Administrador de Componente debe contar con una sesión iniciada con sus credenciales dentro del sistema. Se debe contar con el archivo que contiene las incidencias a insertar.
Funcionalidad	Inserción de incidencias con mecanismo de transacciones.
1. Seleccionar el menú Incidencia.	
2. Seleccionar el submenú Carga Masiva.	
3. Indicar la ubicación lógica el archivo a cargar con el botón Cargar	
4. Se despliega la información de cada Incidencia.	
5. Presionar el botón Aceptar para insertar el contenido.	
6. Comienza el proceso interno de ingesta del archivo.	
7. Esperar respuesta exitosa o fallida del proceso.	
8. El usuario puede cerrar sesión del SAI.	
Resultado	Si el proceso es exitoso, se muestra un mensaje con el número de inserciones realizadas. Si el proceso es erróneo, se eliminan los registros insertados.

Caso de Prueba 5-10: Prueba de Transaccionalidad 01

En el proceso masivo de inserción, se utiliza la confirmación de dos fases, por lo que en caso de presentarse algún error durante el proceso, se maneja un método de *rollback* manual. Debido a que cada inserción es totalmente aislada y atómica, no se requiere la ejecución de un *commit*.



Carga de Incidencias

Archivo:

C:\incidencias\masivas\carga_130414.xlsx

Cargar | Aceptar | Cancelar

Vista Previa

Descripción	Fecha	ID Usuario	Área
No es posible conectarse al servidor	13/04/14 09:14:26	rgonzalez	Infraestructura
No se encuentra la tabla Usuarios en la base de datos MgmtApps.	13/04/14 10:42:31	jcornela	Bases de Datos
La aplicacion de escritorio no inicia.	13/04/14 11:05:13	emartinez	Software

Interfaz de Usuario 5-10: Inserción masiva de incidencias.

```
@Test
public void insertaIncidenciaTrans()
{
    Incidencia[] incidencias = new Incidencia[5];
    Transaccion transaccion = new Transaccion();
    String coleccion = "incidencia";
    String msg = null;
    int count = 1;

    Gson gson = new GsonBuilder()
    .setPrettyPrinting()
    .registerTypeAdapter(Date.class, new DateSerializer())
    .registerTypeAdapter(Date.class, new DateDeserializer())
    .create();

    for (int i = 0; i < incidencias.length; i++)
    {
        incidencias[i] = new Incidencia();
        incidencias[i].setColeccion(coleccion);
        incidencias[i].setIdentificador((long) (123456789 + i));
        incidencias[i].setTransaccion(transaccion.getIdentificador());
        incidencias[i].setNombre("Objeto Incidencia " + i);
        incidencias[i].setDescripcion("Objeto para pruebas integrales.");
        incidencias[i].setEstatus("Cerrado");
        incidencias[i].setFechaApertura(new Date());
        incidencias[i].setFechaCierre(new Date());
    }

    try
    {
        incidenciaCtrl = new IncidenciaServiceImpl();

        incidenciaCtrl.creaTransaccion(transaccion);

        for (Incidencia inc : incidencias)
        {
            incidenciaCtrl.creaIncidencia(inc);
            logger.info("Insertando incidencia: \n" + gson.toJson(inc));
            count++;

            if (count == 3)
                throw new RuntimeException("Error en la insercion.");
        }

        transaccion.setEstatus(Estatus.terminado);
        incidenciaCtrl.actualizaTransaccion(transaccion);
    }
    catch (RuntimeException e)
    {
        transaccion.setEstatus(Estatus.fallido);
        incidenciaCtrl.actualizaTransaccion(transaccion);
        logger.error("Al insertar el documento " + count + ". Ejecutando rollback.");
        count = incidenciaCtrl.ejecutaRollback(coleccion, transaccion);
        logger.info("Rollback ejecutado a " + count + " documentos.");
        Assert.assertNull(msg);
    }
}
```

Código Java 5-7: Método de prueba de inserción masiva con transacciones.

En el método en Código Java 5-7, se detalla cómo se insertan las incidencias dentro de una transacción. Partiendo de un conjunto de incidencias, se crea un objeto **transaccion** de la clase `Transaccion`, el cual contiene información sobre el inicio del procesamiento, como lo son la fecha y hora, el estado de la transacción, etc. Documento JSON 5-6.


```
2014-05-22 09:58:00 INFO - Insertando transaccion:
{
  "identificador": 1400770680314,
  "creado": {
    "$date": "2014-05-22T09:58:00.314Z"
  },
  "estatus": "procesando",
  "coleccion": "transacciones"
}
```


Documento JSON 5-6: Documento inicial de transacción.

En la prueba realizada, se envía un error cuando el contador llegue a tres, por lo que las dos incidencias insertadas anteriormente, serian eliminadas mediante la operación de rollback, actualizando el estado de la transacción y colocando un atributo con la fecha y hora en la que el documento fue actualizado.


```
2014-05-22 09:58:00 INFO - Actualizando transaccion:
{
  "identificador": 1400770680314,
  "creado": {
    "$date": "2014-05-22T09:58:00.314Z"
  },
  "actualizado": {
    "$date": "2014-05-22T09:58:00.509Z"
  },
  "estatus": "fallido",
  "coleccion": "transacciones"
}
```


Documento JSON 5-7: Actualización de la transacción a un estado fallido.

El método de *rollback* se encarga de eliminar todas las incidencias ligadas al objeto **transaccion** de la base de datos.

En Documento JSON 5-8, se muestra la salida del método de prueba, desde la inserción de los primeros dos documentos hasta la ejecución del método de *rollback*.

```
2014-05-22 09:58:00 INFO - Insertando incidencia:
{
  "identificador": 123456789,
  "nombre": "Objeto Incidencia 0",
  "descripcion": "Objeto para pruebas integrales.",
  "fechaApertura": {
    "$date": "2014-05-22T09:58:00.354Z"
  },
  "fechaCierre": {
    "$date": "2014-05-22T09:58:00.354Z"
  },
  "estatus": "Cerrado",
  "coleccion": "incidencia",
  "transaccion": 1400770680314
}
2014-05-22 09:58:00 INFO - Insertando incidencia:
{
  "identificador": 123456790,
  "nombre": "Objeto Incidencia 1",
  "descripcion": "Objeto para pruebas integrales.",
  "fechaApertura": {
    "$date": "2014-05-22T09:58:00.354Z"
  },
  "fechaCierre": {
    "$date": "2014-05-22T09:58:00.354Z"
  },
  "estatus": "Cerrado",
  "coleccion": "incidencia",
  "transaccion": 1400770680314
}
2014-05-22 09:58:00 ERROR - Al insertar el documento 3. Ejecutando rollback.
2014-05-22 09:58:00 INFO - Rollback ejecutado a 2 documentos.
```

Documento JSON 5-8: Salida de la ejecución del método de prueba.

El método de *rollback* requiere especificar una colección y un objeto de la clase `Transaccion`, del cual se tomara el identificador para eliminar aquellos documentos en los que el atributo *transacción* coincide con este identificador. Código Java 5-8.

```
public int ejecutaRollback(String coleccion, Transaccion transaccion)
    throws RuntimeException
{
    try
    {
        BasicDBObject query = new BasicDBObject();
        DBCollection db = getConexion(coleccion);

        query.put("transaccion", transaccion.getIdentificador());

        WriteResult wr = db.remove(query);

        if (wr.getError() != null)
            throw new RuntimeException(wr.getError());

        return wr.getN();
    }
    catch (Exception e)
    {
        throw new RuntimeException(e.getMessage());
    }
}
```

Código Java 5-8: Método para realizar rollback.

5.3. Análisis de Resultados

Las pruebas unitarias permiten establecer los módulos de mediano y bajo nivel que son la base del funcionamiento del SAI, corresponden a las funcionalidades básicas más representativas durante el flujo de operación.

Se requiere que cada objeto sea válido, es decir, que la información que sea requerida a cada paso dentro de la operación sea consistente para evitar errores dentro del SAI.

Una de las ventajas de la implementación en una base de datos orientada a documentos como lo es NoSQL es la libertad de modificar la estructura de un documento, ya no se inserta un registro en una tabla y se liga con otra, ahora el mismo objeto almacena las modificaciones sin que la demás información se vea comprometida.

Una de las diferencias mayormente visibles es la diferencia entre los tiempos de respuesta durante la inserción de incidencias, donde se insertaron desde cien registros, hasta un millón, con una diferencia enorme de tiempos entre el modelo NoSQL y el modelo relacional, las medidas se muestran en Tabla 5-1.

Incidencias	MongoDB(segundos)	MySQL(segundos)
50	2	3
100	2	18
500	3	26
1000	3	38
5000	6	401
10000	6	494
50000	20	2360
100000	55	4830
500000	156	15715
1000000	456	31043

Tabla 5-1: Medidas de registros y tiempo de respuesta en segundos.

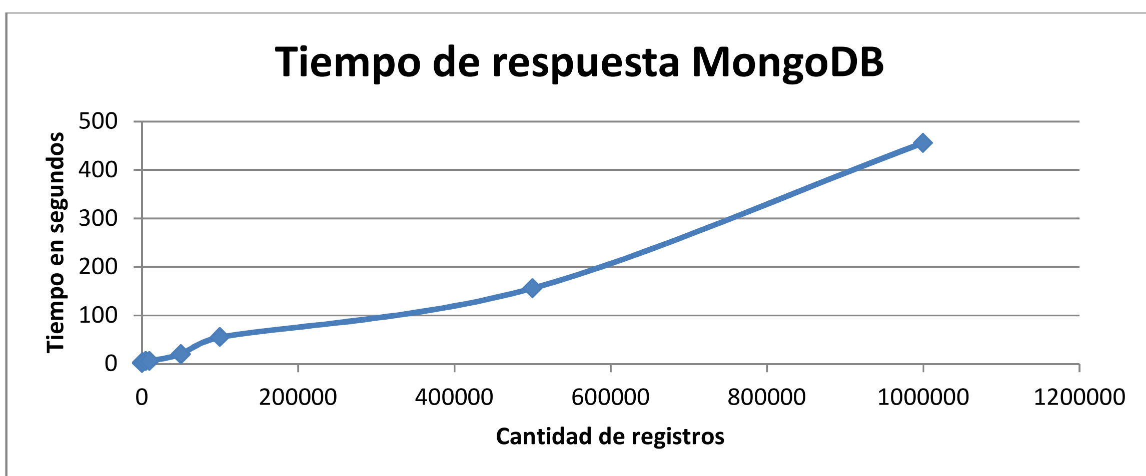
La diferencia entre los tiempos de carga se debe a la cantidad de entidades interactuando entre si durante el proceso de inserción, entre mayor sea el número

de entidades utilizadas, mayor tiempo de procesamiento, debido a que en el modelo NoSQL únicamente se tiene un documento este tiempo es mucho menor en comparación al modelo relacional, el cual utiliza las entidades: `Incidencia`, `EventoIncidencia`, `Componente` e `IncidenciaComponente`.

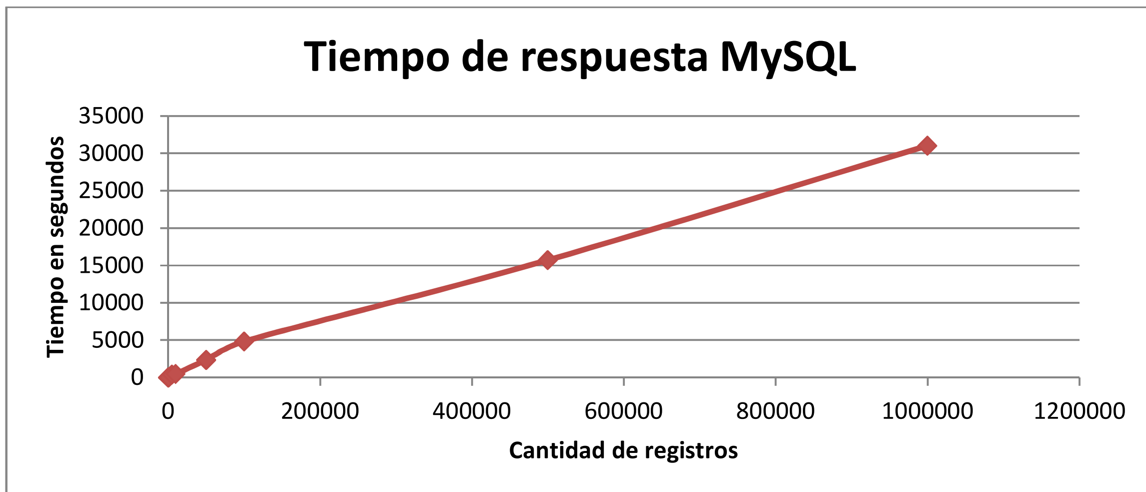
Con esta razón de 4 entidades a 1, que se tiene en el modelo NoSQL, permite que toda la información relacionada con el documento se almacene en un único elemento.

Es importante mencionar, que mientras el proceso de inserción en MongoDB fue menor a 10 minutos, en MySQL tuvo un tiempo aproximado de 13 horas, durante los cuales fue necesario relanzar el proceso debido a desbordamientos de memoria.

Como se observa en Gráfica 5-1 y Gráfica 5-2, los tiempos de inserción varían enormemente, debido a la optimización que se tiene en cuanto al modelo orientado a documentos, evitando la necesidad de relacionamiento y de llaves.



Gráfica 5-1: Tiempo de carga de incidencias en MongoDB.



Gráfica 5-2: Tiempo de carga de incidencias en MySQL.

El tiempo de lectura para las consultas también se ve mejorado en el modelo NoSQL, de igual manera que las escrituras, tanto la consulta de información como la ejecución de funciones map-reduce, tienen un tiempo de procesamiento menor al modelo relacional.

Mientras que en NoSQL solo se deben especificar parámetros con los cuales se especificaran los documentos dentro de la colección que serán procesados, evitando así, el relacionamiento por medio de llaves principales y foráneas, ya que toda la información relevante e encuentra en una sola unidad como lo es el documento.

Al no requerir de un esquema, los documentos pueden consultarse de acuerdo a sus atributos, sin embargo, es necesario colocarlos en una colección adecuada, donde los atributos, si bien no requieren ser iguales, deben presentar

similitud y muchas veces, lo único que cambia, es el número de elementos de cierto tipo, o algún valor que permita identificar qué tipo de documento es.

Este último caso, se tiene que una Incidencia tiene un Componente, utilizando el identificador de este como uno de sus atributos, no para relacionarlo con estos documentos, sino para permitir la redundancia de información y poder realizar un relacionamiento básico de manera interna.

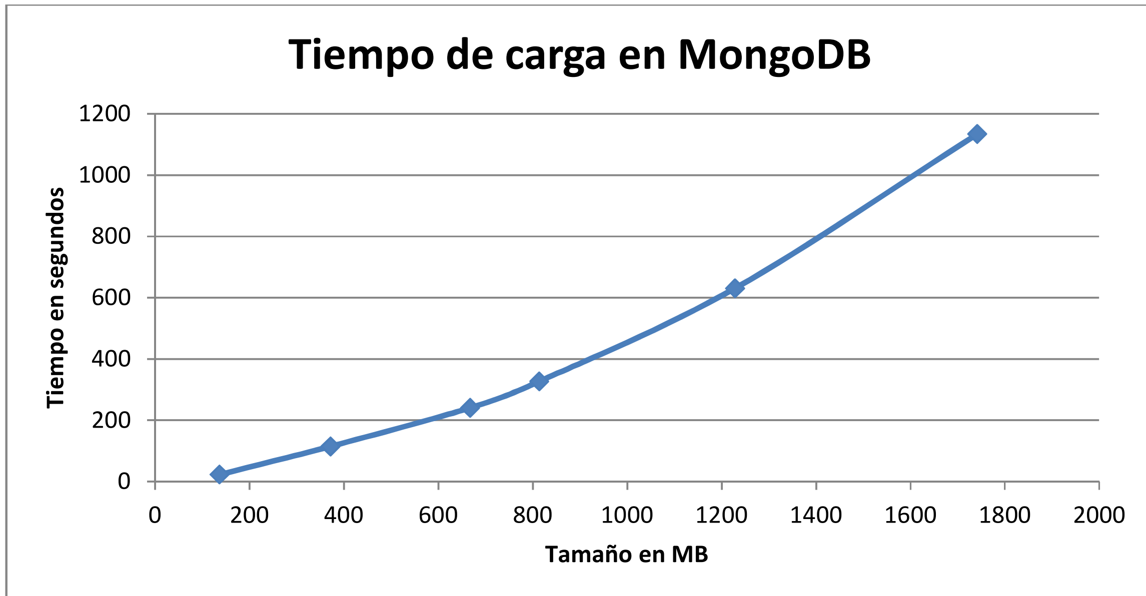
De acuerdo al área de este componente, se agrega uno de cuatro posibles atributos que permita identificar que componente es el que presenta la problemática, que servirá para identificar en primera instancia, del tipo de incidencia de la cual se trata.

En cuanto al procesamiento de archivos de bitácoras, el desempeño del SAI con la base en MongoDB es ligeramente mejor que el de MySQL; Este último requirió configuración adicional para permitir enviar un alto volumen de datos.

En cambio, GridFS se encarga de segmentar un archivo e insertarlo dentro de las respectivas colecciones, de esta manera, no se hace una inserción de un único objeto de tamaño considerable, sino que se divide en múltiples pedazos y se almacenan en una colección diseñada para ello.

De acuerdo a la Gráfica 5-3 se nota una ligera curva y la relación entre el tiempo de carga de cada uno de los archivos. Mientras que un archivo de alrededor de 100MB tomo aproximadamente 20 segundos, un archivo de casi 2GB, con aproximadamente 10 millones de registros, tardo alrededor de 18

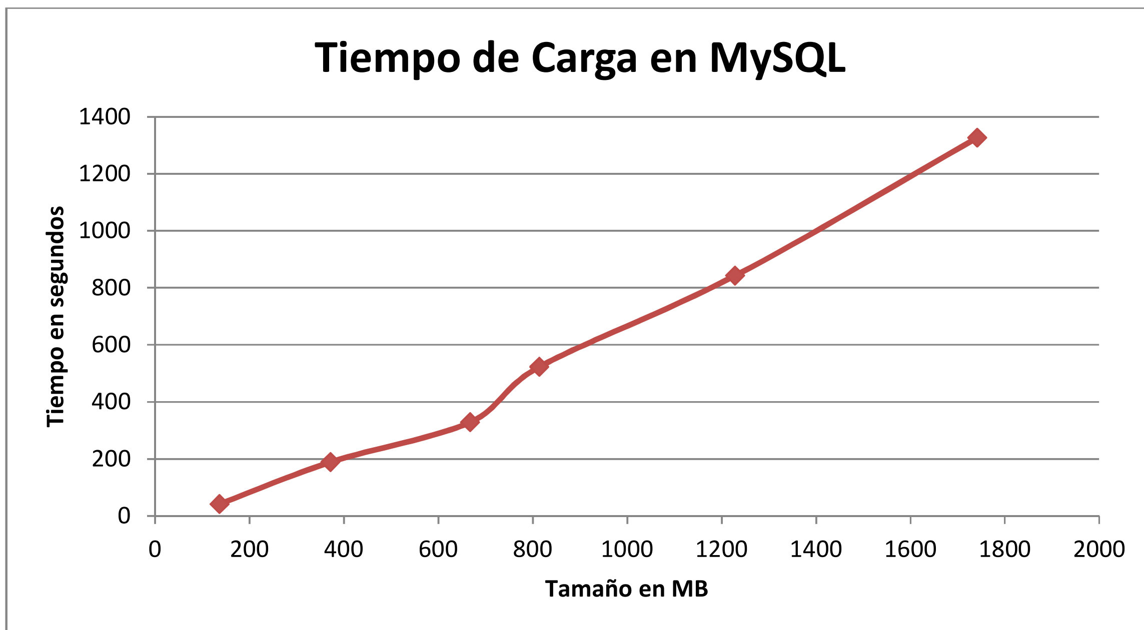
minutos, es importante mencionar que se insertó el archivo en su totalidad, es decir, no fue analizado su contenido.



Gráfica 5-3: Tiempo de carga de archivos en MongoDB (MB vs segundos)

De acuerdo a los tiempos de carga de los mismos archivos en MySQL, se nota un tiempo de respuesta mayor.

Como se observa en Gráfica 5-3, los tiempos fueron similares, sin tomar en cuenta, que una carga de un archivo en esta herramienta consume demasiada memoria, y fue necesario reiniciar el servidor de MySQL en múltiples ocasiones, por lo que en realidad las cargas tomaron un tiempo mayor que no fue cuantificable debido a que las pruebas fueron erróneas.



Gráfica 5-4: Tiempo de carga de archivos en MySQL (MB vs segundos).

Para terminar, los resultados de las pruebas fueron satisfactorios, puesto que en cada uno de los casos, los requerimientos entre el SAI con el modelo relacional y el modelo orientado a documentos es totalmente transparente.

La diferencia es notable con tiempo de respuesta menores, una programación mayor y robusta, sin ser tan compleja; De esta manera es posible visualizar con mejor detalle, el ciclo de vida de la información, desde la creación, edición, hasta la consulta de las mismas sin la necesidad de consultas complejas, restricciones de esquema o consideraciones especiales para el manejo de archivo.

En cuanto a transaccionalidad, MongoDB la implementa de manera nativa, pero muy simple y a nivel documento, por lo que es posible gestionar cada uno de manera individual, no siendo posible manipular más de un documento a la vez.

Sin embargo, al requerir un mayor nivel de transaccionalidad sobre un conjunto de documentos, surgió la necesidad de implementar el mecanismo de confirmación de dos fases, la cual se encarga de encapsular cada una de las respuestas que los documentos generan al durante las lecturas y escrituras dentro de la base de datos.

La confirmación de dos fases de cierto modo funciona como una secuencia de lecturas y escrituras, en las que si una de ellas presenta errores, las operaciones anteriores son eliminadas y las operaciones posteriores se omiten. Si viene es un mecanismo bastante simple, cumple su propósito y otorga una mejor visibilidad sobre cómo funcionan las transacciones dentro de MongoDB, abriendo paso a mejoras que pueden implementarse en un futuro al manejador.

INGENIERIA
F I

Capítulo 6

Conclusiones

6. Conclusiones.

La computación es una de las ciencias que más avance demuestra a través del tiempo por lo que la implementación de nuevas tecnologías que explotan nuevos aspectos de los datos y la información es algo común para el desarrollo de nuevos sistemas.

A su vez, las bases de datos han permitido que el manejo de los datos y su explotación se conviertan en un área sumamente especializada de la computación, estableciendo nuevos retos en su funcionamiento, cumpliendo necesidades específicas en diversas áreas de negocios y generando nuevas de mayor complejidad y que requieren mayor dedicación de recursos y personal.

Al ser un entorno evolutivo, las tecnologías existentes necesitan ser flexibles en su adaptación para cumplir el objetivo con el que fueron diseñadas y construidas, pero, tarde o temprano, llega un límite en el cual ya no pueden seguir evolucionado. Tal caso se puede ejemplificar desde la eminente extinción de la PC, a la cual la movilidad y la expansión de las tecnologías móviles han comenzado a rezagarlas a tareas monótonas y sin movilidad física, hasta lenguajes de programación que han cambiado distintos paradigmas como lo fue, en su momento, la evolución de la programación estructurada a la programación orientada a objetos.

Las bases de datos relacionales no podían permanecer alejadas a esta tendencia de mejora continua. Si bien hay una distancia considerable entre las primeras bases de datos autónomas que tardaban horas en ejecutar alguna instrucción o que hacían uso de una gran cantidad de recursos físicos, y las más actuales que

pueden operar en entornos distribuidos, manejar grandes volúmenes de datos, mantener una disponibilidad 24 horas los 7 días de la semana, además de administrar una gran cantidad de transacciones, aún existen fronteras que las bases de datos relacionales o clásicas no han podido rebasar.

Para poder satisfacer necesidades específicas en las cuales las bases de datos relacionales resultan poco útiles, es que surgen las bases de datos NoSQL.

A lo largo de este trabajo de tesis se observó qué es, cómo operan y qué tipos de bases de datos NoSQL existen actualmente y con base en ese análisis se logró realizar la implementación del SAI de una forma reveladora.

El paradigma NoSQL, en su primer vistazo, pareciera ser la solución para aquellos problemas específicos donde los manejadores de datos relacionales actuales parecen comenzar a trastabillar, por ejemplo en el manejo de grandes volúmenes de información donde la capacidad de las bases de datos relacionales para manejar "grandes" volúmenes de datos resulta no ser tan "grande" porque la demanda ha sido mayor.

Así mismo cuando la complejidad de administrar una base de datos ha crecido tanto que se prefiere duplicar información porque es más sencillo que actualizar cierto registro del que se desprende una actualización a múltiples tablas o simplemente al momento de escalar la base de datos resulta sumamente complejo y oneroso porque no pueden existir ventanas de mantenimiento en los cuales todo el sistema se encuentra fuera de línea, sumado al costo de los recursos físicos asignados.

Bajo la perspectiva anterior, NoSQL parece ser una opción sencilla en donde se decide cambiar de base de datos o adquirir más equipo y contar con gente más preparada para hacer más contar con una base de datos más robusta que la base de datos actual, pero la respuesta más sensata a este punto es que depende de las necesidades particulares de cada sistema.

Al desarrollar el SAI en conjunto con una base de datos NoSQL orientada a documentos, se obtiene una ventaja a nivel programación considerable, ya que los elementos trabajan de forma más transparente entre la base de datos y el sistema sin mayor dificultad, siendo transparente a nivel aplicación, el SAI está implementado en ambos modelos.

Las principales características de MongoDB, están representadas modularmente por el SAI, desde las colecciones de documentos, el manejo de archivos, los documentos sin esquema, hasta las funciones de agrupamiento map/reduce, entre otras, son utilizadas para realizar las operaciones más comunes.

Mientras que en el modelo relacional se tiene un conjunto de entidades con una estructura estrictamente idéntica, las bases de datos NoSQL orientadas a documentos omite esta restricción al proveer colecciones dentro de las cuales se almacenan documentos estructuralmente similares, con la facilidad de que son modificables sin afectar la información existente.

Esta es una gran ventaja, ya que de requerir un cambio en la estructura dentro de un modelo relacional, la demás información se ve afectada, sin mencionar la

problemática que derivaría el modificar una nueva entidad, mientras que en un documento, basta con agregar los elementos requeridos sin mayor complicación.

Debido a que la información se encuentra en un solo documento y no a través de múltiples tablas, la velocidad de respuesta es notable cuando la cantidad de documentos aumenta y la estructura de los mismos se vuelve más compleja.

Las funciones map/reduce reducen considerablemente el tiempo de procesamiento que una función de agrupación tarda normalmente en un modelo relacional, puesto que además de especificar filtros y parámetros, se realizan análisis completos de las entidades, reduciendo el tiempo de respuesta en sistemas con altos volúmenes de datos.

Map/reduce ofrecen herramientas bastante útiles, sin embargo, requiere que la lógica de las funciones map y reduce sean las óptimas, de esta manera se tendrán análisis más completos sobre mayor información de manera más rápida y eficaz.

El manejo de archivos es bastante sencillo, la habilidad de MongoDB para utilizar GridFS con los archivos se hace de manera transparente, no requieren una especificación compleja, además de utilizar el propio GridFS, ya que en algunos manejadores de datos actualmente requieren un cierto tipo de dato, como el BLOB, el CLOB, entre otros.

De igual manera, MongoDB es ligeramente más rápido en comparación con MySQL, mientras que en este último, el sistema presentaba interrupciones y

desbordamientos de memoria al insertar los archivos, MongoDB no presentó problemas ni antes ni después de la inserción de archivos.

Una de las desventajas al implementar el SAI, fue la necesidad que ACID brinda a los modelos relacionales, de cierto modo, cada operación en MongoDB cumple con las características ACID, sin embargo, no son tan completas como lo serian en el modelo relacional.

No se cuenta con mecanismos transaccionales completos, así como no es posible utilizar un mecanismo para la concurrencia, ni configuración alguna durante todo el proceso de implementación del SAI, fue necesario que se implementaran dichos mecanismos a nivel programación.

Sin embargo, y dependiendo de cada sistema desarrollado, estas características pueden o no pasarse por alto, tomando en cuenta que MongoDB maneja la información de distinta manera a que las bases de datos convenciones que existen actualmente, muchos de los paradigmas se han visto renovados de nuevos requerimientos y funcionalidades gracias a las nuevas tecnologías.

El desarrollo del SAI fue un reto bastante interesante, debido a que los requerimientos de un sistema basado inicialmente en un modelo relacional, requiere un completo análisis al migrar a una base de datos orientada a documentos.

Se requiere de análisis de información, de estructura, de tiempos, de configuraciones, de topologías de red, etc., todo un conjunto de características que debe ser desarrollada a la par de la mejor manera, puesto que una vez implementado

un sistema y puesto en marcha, un pequeño detalle puede impactar de manera negativa en el negocio.

Pero, ¿de qué tipo de necesidades depende la decisión de quedarse en el paradigma relacional o mudarse al NoSQL? Los siguientes puntos pueden contribuir con la toma de decisión:

- Si se requiere que un sistema sea altamente transaccional y con una alta integridad de datos, el paradigma relacional sigue siendo una excelente opción.

- Si se requiere un sistema altamente disponible, distribuido y a un bajo costo, NoSQL resuelve fácilmente este tema.

- Si se requiere un sistema que pueda manejar muy grandes volúmenes de datos y estar inherentemente preparados para tecnologías como Big Data, NoSQL es una opción altamente viable.

- Si se requiere mejor utilización de recursos físicos locales y en un sistema distribuido, además de una administración relativamente sencilla y no onerosa, NoSQL es la mejor aproximación a este ideal.

- Si el requerimiento es un sistema fácilmente escalable, ambos paradigmas son muy flexibles y aptos para ello, pero NoSQL lo hace de forma sencilla y a un costo relativamente bajo.

En este trabajo de tesis se demostró a través de la implementación de Sistema de Administración de Incidencias cómo NoSQL mediante el uso de MongoDB trae consigo beneficios, pero también trae ciertas incertidumbres.

El SAI nos deja como lección que ninguno de los dos paradigmas cumple completamente con los requerimientos generales ni de este sistema ni de la mayoría en general, porque la transaccionalidad que puede ser alta tiene un mejor manejo por los sistemas de bases de datos relacionales, pero la escalabilidad y el manejo de archivos de gran tamaño es resuelto por NoSQL.

En síntesis, el SAI puede funcionar muy bien con NoSQL, pero puede funcionar mejor como un sistema híbrido que mezcle lo mejor de ambos paradigmas para alcanzar un nivel de madurez y que resuelva necesidades específicas.

INGENIERIA
F I

Capítulo 7

Bibliografía y mesografía

7. Bibliografía y mesografía

- IEEE. (1990). IEEE Standard Glossary of Software Engineering Terminology. IEEE.
- Microsoft Corporation. (Mayo de 2012). Qué es unit test. Recuperado el 13 de Abril de 2014, de Microsoft Developer Network: http://msdn.microsoft.com/es-es/library/jj130731.aspx
- Microsoft Corporation. (s.f.). Integration Testing. Recuperado el 13 de Abril de 2014, de Microsoft Developer Network: http://msdn.microsoft.com/en-us/library/aa292128%28v=vs.71%29.aspx
- Microsoft Corporation. (s.f.). Unit Testing. Recuperado el 13 de Abril de 2014, de Microsoft Developer Network: http://msdn.microsoft.com/en-us/library/aa292197%28v=vs.71%29.aspx
- Myers, G., Sandler, C., & Badgett, T. (2012). The Art of Software Testing. New Jersey: John Wiley & Sons, Inc. .
- Runeson, P. (2006). A survey of unit testing practices. IEEE Journals & Magazines, 22-29.
- MongoDB Documentation Project. (2014). MongoDB Documentation Release 2.4.9.
- MongoDB, I. (2014). GridFS Reference - MongoDB Manual 2.6.1. Recuperado el 23 de Marzo de 2014, de sitio web de MongoDB, Inc: http://docs.mongodb.org/manual/reference/gridfs/#gridfs-files-collection
- Microsoft Corporation. (2014). Recuperado el 24 de Abril de 2014, de Microsoft Developer Network: http://msdn.microsoft.com/en-us/library/ee658109.aspx

- MongoDB, I. (2014). Configuration File Options - MongoDB 2.4.9. Retrieved Abril 5, 2014, from http://docs.mongodb.org/manual/reference/configuration-options/
- MongoDB, I. (2014). MongoDB Drivers and Clients Libraries. Retrieved Abril 05, 2014, from http://docs.mongodb.org/ecosystem/drivers/
- MongoDB, I. (2014). Perform Two Phase Commits - MongoDB Manual 2.4.9. Retrieved Marzo 23, 2014, from sitio web de MongoDB, Inc.: http://docs.mongodb.org/manual/tutorial/perform-two-phase-commits/
- Sadalage, P. J. (2013). NoSQL Distilled, A Brief Guide to the Emerging World of Polyglot Persistence. Crawfordsville, Indiana.: Pearson Education, Inc.
- Vaish, G. (2013). Getting Started with NoSQL: Your guide to the world and technology of NoSQL. Birmingham, UK: Packt Publishing Ltd.
- Vernon, V. (2013). Implementing Domain-Driven Design. Westford, Massachusetts: Pearson Education, Inc.
- Apache Software Foundation. (n.d.). Apache Cassandra 1.1 Documentation: About Column Families. Retrieved Enero 21, 2013, from http://www.datastax.com/docs/1.1/ddl/column_family
- Apache Software Foundation. (n.d.). Apache Cassandra 1.1 Documentation: About Data Consistency in Cassandra. Retrieved Enero 25, 2013, from Datastax: http://www.datastax.com/docs/1.1/dml/data_consistency
- Apache Software Foundation. (s.f.). Apache Cassandra 1.1 Documentation: About Internode Communications (Gossip). Recuperado el 31 de Enero de 2013, de Apache Cassandra 1.1 Documentation: http://www.datastax.com/docs/1.1/cluster_architecture/gossip

- Apache Software Foundation. (n.d.). Apache Cassandra 1.1 Documentation: The Cassandra Data Model. Retrieved Enero 21, 2013, from http://www.datastax.com/docs/1.1/ddl/about-data-model
- Camacho, E. (Mayo-Julio de 2010). NoSQL la evolución de las bases de datos. Recuperado el 22 de Marzo de 2014, de Software Gurú: http://sg.com.mx/revista/42/nosql-la-evolucion-las-bases-datos#.Uy416IXJZUl
- Hecht, R., & Jablonski, S. (2011). NoSQL Evaluation: A Use Case Oriented Survey. Cloud and Service Computing (CSC) (págs. 336-341). IEEE Conference Publications.
- Indrawan-Santiago, M. (2012). Database Research: Are We At a Crossroad? Reflection on NoSQL. Network-Based Infomation Systems (NBiS) (págs. 45-51). IEEE Conference Publications.
- Montag, D. (Enero de 2013). Neotechnology, graphs are everywhere. Recuperado el 4 de Abril de 2014, de http://info.neotechnology.com/rs/neotechnology/images/Understanding%20Neo4j%20Scalability(2).pdf
- Neo Technology. (2013). Neo4j the graph database. Recuperado el 20 de Julio de 2013, de The Neo4j Manual: http://docs.neo4j.org/chunked/milestone/graphdb-neo4j-nodes.html
- Neo Technology. (2013). Neo4j the graph database. Recuperado el 20 de Julio de 2013, de The Neo4j Manual: http://docs.neo4j.org/chunked/milestone/graphdb-neo4j-relationships.html

- Neo Technology. (20 de Febrero de 2014). Neo4j the graph database. Recuperado el 3 de Abril de 2014, de The Neo4j Manual v2.1.0: http://docs.neo4j.org/chunked/milestone/index.html
- Neo Technology. (2014). Neo4j the graph database. Recuperado el 5 de Abril de 2014, de The Neo4j Manual: http://docs.neo4j.org/chunked/milestone/transactions-locking.html
- Dory, T., Mejías, B., & Van Roy, P. (s.f.). Comparative elasticity and scalability measurements of cloud databases.
- MongoDB, I. (2014). GridFS Reference - MongoDB Manual 2.6.1. Retrieved Marzo 23, 2014, from sitio web de MongoDB, Inc: http://docs.mongodb.org/manual/reference/gridfs/#gridfs-files-collection
- Silberschatz, A., Korth, H., & Sudarshan, S. (2006). Fundamentos de bases de datos. Quinta Edición. Madrid: McGraw-Hill/Interamericana de España.

7.1. Lista de figuras.

Caso de Prueba 5-1: Prueba Unitario 01, Validación de los atributos de una Incidencia. 225
Caso de Prueba 5-2: Prueba Unitaria 02, Inserción del objeto incidencia........ 227
Caso de Prueba 5-3: Prueba Integral 01........ 231
Caso de Prueba 5-4: Prueba Integral 02........ 234
Caso de Prueba 5-5: Prueba Unitaria 03........ 235
Caso de Prueba 5-6: Prueba de Integración 03........ 242
Caso de Prueba 5-7: Prueba de Integración 02........ 244
Caso de Prueba 5-8: Prueba de Volumen 01........ 246
Caso de Prueba 5-9: Prueba de Concurrencia 01........ 248
Caso de Prueba 5-10: Prueba de Transaccionalidad 01........ 253
Código Java 4-1: Clase Incidencia........ 167
Código Java 4-2: Constructor del objeto GSON........ 168
Código Java 4-3: Impresión de Objeto Incidencia con el constructor Gson()........ 168
Código Java 4-4: Impresión del Objeto con el constructor GsonBuilder()........ 169
Código Java 4-5: Implementación del objeto Incidencia con los tipos de datos del API. 170
Código Java 4-6: Representación JSON de la Incidencia con los constructores del API. 170
Código Java 4-7: Clase para serializar de fechas........ 171
Código Java 4-8: Clase para de serializar fechas........ 172
Código Java 4-9: Constructor GSON con DateSerializer........ 172
Código Java 4-10: Constructor GSON con DateDeserializer........ 173
Código Java 4-11: Conexión a la bases de datos de MongoDB........ 173
Código Java 4-12: Método de autenticación con MongoDB........ 173
Código Java 4-13: Objetos necesarios para la inserción de una Incidencia........ 174
Código Java 4-14: Objetos básicos para crear un documento........ 175
Código Java 4-15: Llamada al método de inserción........ 175
Código Java 4-16: Atributos Incidencia convertidos a DBObject........ 176
Código Java 4-17: Conversión del atributo de componente en una Incidencia........ 176
Código Java 4-18: Conversión de los eventos de la Incidencia........ 177
Código Java 4-19: Inserción del DBObject al final del método creaIncidencia........ 177
Código Java 4-20: Código básico para insertar un documento........ 178
Código Java 4-21: Consultando toda la información de la colección **incidencia**........ 179
Código Java 4-22: Método de consulta de Incidencias en el DAO........ 181
Código Java 4-23: Impresión de los las Incidencias encontradas........ 182
Código Java 4-24: Consulta de Incidencias por fecha........ 183
Código Java 4-25: Incidencias delimitadas por un intervalo de fecha........ 184

Código Java 4-26: Representación de una Incidencia en JSON. 185
Código Java 4-27: Consulta de un elemento estructurado dentro de una Incidencia. 186
Código Java 4-28: Incidencias delimitadas por **eventos.identificadorUsuario**. 186
Código Java 4-29: Eliminación de la incidencia por identificador. 188
Código Java 4-30: Método de actualización de la Incidencia. 189
Código Java 4-31: Actualización del campo estatus de una Incidencia. 190
Código Java 4-32: Utilizando la función $set. 191
Código Java 4-33: Utilizando la función $push. 192
Código Java 4-34: Utilizando la función $unset. 193
Código Java 4-35: Método de inserción de archivo con GridFS. 196
Código Java 4-36: Generación de archivo almacenado con GridFS. 197
Código Java 4-37: Eliminación de un archivo con GridFS. 198
Código Java 4-38: Código para un documento genérico especificando nivel de escritura. 202
Código Java 4-39: Atributos de la clase Transaccion.. 203
Código Java 4-40: Modificación a la clase Incidencia para permitir transacciones. 204
Código Java 4-41: Método de inserción transaccional de un grupo de incidencias. ... 205
Código Java 4-42: Método de creación del objeto Transaccion dentro de MongoDB. 206
Código Java 4-43: Método de rollback implementado... 207
Código Java 4-44: Método de rollback dentro del DAO. 208
Código Java 4-45: Método de actualización de Transaccion. 209
Código Java 4-46: Método para ejecutar la función map-reduce. 213
Código Java 4-47: El objeto query delimita los documentos a procesar. 214
Código Java 4-48: Almacenando los resultados de la consulta. 215
Código Java 5-1: Objeto básico a utilizar en las pruebas unitarias. 223
Código Java 5-2: Conversión del objeto incidencia a una cadena válida JSON. 224
Código Java 5-3: Código de validación del objeto incidencia. 226
Código Java 5-4: Inserción del objeto incidencia. 228
Código Java 5-5: Mensaje de error a nivel programación. 250
Código Java 5-6: Creación del candado en un documento incidencia. 250
Código Java 5-7: Método de prueba de inserción masiva con transacciones. 255
Código Java 5-8: Método para realizar rollback. 258
Código JavaScript 4-1: Función map para el atributo fechaApertura 212
Código JavaScript 4-2: Función reduce para los pares fecha-conteo. 213
Código JavaScript 5-1: Funciones map/reduce para generar reporte por estatus y área. 237
Código JavaScript 5-2: Funciones map/reduce para generar reporte por promedio de resolución. 239
Código JavaScript 5-3: Funciones map-reduce para generar reporte por etiquetas. .. 241
Diagrama 2-1: Proceso de análisis preliminar. 83

Diagrama 2-2: Proceso de apertura de incidencias 84
Diagrama 3-1. Entidad Incidencia 100
Diagrama 3-2: Entidad EventoIncidencia 101
Diagrama 3-3: Entidad ActividadEvento... 101
Diagrama 3-4: Entidad Area 101
Diagrama 3-5: Entidad Usuario 102
Diagrama 3-6: Usuario tipo Administrador General.. 102
Diagrama 3-7: Usuario tipo Administrador Componente 103
Diagrama 3-8: Usuario tipo Usuario Regular.. 103
Diagrama 3-9: Entidad Componente 104
Diagrama 3-10: Subtipo BaseDatos 104
Diagrama 3-11: Subtipo Software 104
Diagrama 3-12: Entidad Aplicacion 104
Diagrama 3-13: Entidad Servidor 105
Diagrama 3-14: Generalizaciones de la entidad Componente 105
Diagrama 3-15: Entidad BitacoraComponente 106
Diagrama 3-16: Relación Incidencia-EventoIncidencia 107
Diagrama 3-17: Relación Incidencia-Componente 108
Diagrama 3-18: Relación Componente-BitacoraComponente 108
Diagrama 3-19: Relación Usuario-Usuario 109
Diagrama 3-20: Relación Usuario-Area.. 109
Diagrama 3-21: Relación Usuario-EventoIncidencia 110
Diagrama 3-22: Relación Componente-Usuario.. 110
Diagrama 3-23: Relación EventoIncidencia-ActividadIncidencia................................... 111
Diagrama 3-24. Visión general de las relaciones entre entidades del SAI. 112
Diagrama 3-25: Casos de uso generales del SAI 114
Diagrama 3-26. Arquitectura de Replicación para MongoDB. 155
Diagrama 3-27. Archivos de datos para MongoDB 156
Diagrama 3-28. Archivo **mongodb.conf** para un miembro del conjunto de Replicas. 162
Diagrama de Secuencia 3-1: Alta de Incidencia. 119
Diagrama de Secuencia 3-2: Edición de Incidencia 120
Diagrama de Secuencia 3-3: Consulta de Incidencia.. 121
Diagrama de Secuencia 3-4: Consulta de Reportes 122
Diagrama de Secuencia 3-5: Alta de Nuevo Componente 123
Diagrama de Secuencia 3-6: Edición de Componente.. 124
Documento JSON 3-1: Documentos usuario y area 128
Documento JSON 3-2: Agregación de entidades usuario y area 130
Documento JSON 3-3: Documentos Usuario, TipoActividad y EventoIncidencia 131
Documento JSON 3-4: Agregación para EventoIncidencia... 132
Documento JSON 3-5: Documento de incidencia 133

Documento JSON 3-6: Agregación para incidencia 134
Documento JSON 3-7: Agregación final para la entidad Incidencia 136
Documento JSON 3-8: Documento completo de Incidencia 147
Documento JSON 3-9: Documento completo de Usuario 148
Documento JSON 3-10: Documento completo de Aplicacion. 149
Documento JSON 3-11: Documento complete de BaseDatos 149
Documento JSON 3-12: Documento completo de Servidor. 150
Documento JSON 3-13: Documento completo de Software 150
Documento JSON 4-1: Mensaje de respuesta de una inserción correcta. 178
Documento JSON 4-2: Representación JSON de la consulta de una Incidencia. 187
Documento JSON 4-3: Objeto básico de Incidencia en JSON en la base de datos. 189
Documento JSON 4-4: Estructura básica del documento files. 195
Documento JSON 4-5: Estructura básica del documento chunks. 196
Documento JSON 4-6: Ejemplo de un documento files. 199
Documento JSON 4-7: Documentos creados a partir de una bitácora. 201
Documento JSON 4-8: Documento inicial de transacción. 204
Documento JSON 4-9: Estructura de la función map-reduce. 211
Documento JSON 4-10: Resultados de la operación map-reduce. 213
Documento JSON 4-11: Resultados del procesamiento map-reduce delimitado. 214
Documento JSON 5-1: Documento JSON del objeto incidencia para pruebas. 224
Documento JSON 5-2: Documento de Incidencia creado. 233
Documento JSON 5-3: Filtros para realizar la consulta. 235
Documento JSON 5-4: Documento de bloqueo con la póliza expireAt: 251
Documento JSON 5-5: Índice de tiempo de vida en segundos sobre el atributo fechaCreacion. 252
Documento JSON 5-6: Documento inicial de transacción. 256
Documento JSON 5-7: Actualización de la transacción a un estado fallido. 256
Documento JSON 5-8: Salida de la ejecución del método de prueba. 257
Figura 1-1: Representación de un nodo 63
Figura 1-2: Representación de nodos y relaciones 64
Ilustración 4-1: Datos de muestra de un archivo de bitácora sencillo. 201
Interfaz de Usuario 5-1: Alta de una incidencia. 232
Interfaz de Usuario 5-2: Consulta de incidencias. 234
Interfaz de Usuario 5-3: Reporte por línea de tiempo de incidencias. 236
Interfaz de Usuario 5-4: Reporte por promedio de días de resolución. 238
Interfaz de Usuario 5-5: Reporte por etiquetas. 240
Interfaz de Usuario 5-6: Edición de incidencia. 243
Interfaz de Usuario 5-7: Edición de componente. 245
Interfaz de Usuario 5-8: Carga de incidencias. 247
Interfaz de Usuario 5-9: Mensaje de error en la interfaz grafica. 249

Interfaz de Usuario 5-10: Inserción masiva de incidencias. 254
Modelo Entidad-Relación 3-1: Relación entre entidades Usuario-Area 128
Modelo Entidad-Relación 3-2: Relacionamiento entre Incidencia y EventoIncidencia 131
Modelo Entidad-Relación 3-3: Relacionamiento entre entidades Incidencia e IncidenciaComponente.. 135
Modelo Entidad-Relación 3-4. Entidades de control de acceso para usuarios............ 143
Modelo Entidad-Relación 3-5: Entidades de administración de incidencias................ 144
Modelo Entidad-Relación 3-6: Entidades de Componente y Administración............... 144
Tabla 1-1: Equivalencia de términos usados en NoSQL 16
Tabla 1-2: Operaciones que bloquean una base de datos en MongoDB 58
Tabla 1-3: Tabla comparativa entre los distintos tipos de bases de datos NoSQL. (Indrawan-Santiago, 2012) (Hecht & Jablonski, 2011) 68
Tabla 3-1: Descripción de Caso de Uso Crear Incidencia... 115
Tabla 3-2: Descripción de Caso de Uso Editar Incidencia. 115
Tabla 3-3: Descripción de Caso de Uso Consultar Incidencia 116
Tabla 3-4: Descripción de Caso de Uso Consultar Reportes 116
Tabla 3-5: Descripción de Caso de Uso Crear Componente 116
Tabla 3-6: Descripción de Caso de Uso Editar Componente 117
Tabla 5-1: Medidas de registros y tiempo de respuesta en segundos. 259

Bibliografía y mesografía.