UNIVERSIDAD POLITÉCNICA SALESIANA
SEDE QUITO

CARRERA:
INGENIERÍA DE SISTEMAS

Trabajo de titulación previo a la obtención del título de:
INGENIERO DE SISTEMAS

TEMA:
ANÁLISIS, DISEÑO, CONSTRUCCIÓN E IMPLEMENTACIÓN DE UN GEOPORTAL CON LAS HERRAMIENTAS MONGO DB, JSON Y GEOJSON PARA EL PROYECTO IDE-UPS.

AUTORES:
COBA DE LA TORRE CARLOS ALEJANDRO
GÓMEZ GÓMEZ PABLO MAURICIO

DIRECTOR:
NAVAS RUILOVA GUSTAVO ERNESTO

Quito, septiembre del 2014

DECLARATORIA DE RESPONSABILIDAD Y AUTORIZACIÓN DE USO DEL TRABAJO DE TITULACIÓN

Nosotros, autorizamos a la Universidad Politécnica Salesiana la publicación total o parcial de este trabajo de titulación y su reproducción sin fines de lucro.

Además, declaramos que los conceptos, análisis desarrollados y las conclusiones del presente trabajo son de exclusiva responsabilidad de los autores.

Quito, septiembre de 2014

Carlos Alejandro Coba De la Torre	Pablo Mauricio Gómez Gómez
CI. 1720908167.	CI. 1721970752.

DEDICATORIA

A Dios por la vida, por darme la fuerza e iluminar mi mente y mis pasos en el camino que he recorrido, hasta llegar a lograr mi meta soñada.

A mis padres, por ser el apoyo incondicional, por darme la fuerza con la que me han impulsado para seguir adelante, con su ejemplo y con su inmenso amor, acompañándome en muchas noches sin dormir y en contables horas de trabajo, enseñándome que si se tiene la confianza en Dios y en la Virgen Santísima, nada es imposible.

A mi hermana, por ser mi mayor inspiración para superarme y seguir adelante ante cualquier obstáculo que se me presente, con su ejemplo de lucha y perseverancia desde muy niña hasta ahora que se ha convertido en una hermosa señorita y es la persona que me brinda la fuerza para seguir superándome.

A mis abuelitos y abuelitas, presentes y ausentes que con sus bendiciones y oraciones han hecho posible llegar a cumplir un sueño y un anhelo deseado.

A mis tíos y primos, que por medio de sus consejos han sabido guiarme en esta etapa, para ser una mejor persona, humilde y de bien.

Carlos Coba De la Torre

DEDICATORIA

A ustedes querida familia, que nunca dejaron que me rindiese, que siempre me han apoyado en todo, desde económicamente hasta lo sentimental, que me brindan su apoyo incondicional y supieron guiarme para que pueda terminar mi carrera universitaria.

En especial a mis queridos padres, que siempre me dieron todo lo mejor que pudieron y que desde el cielo miran con orgullo el presente trabajo.

A mi amada Evelyn, quien con su paciencia y amor, lograron que no flaqueara en la elaboración de mi trabajo de titulación. Gracias por estar siempre a mi lado.

A mí amado hijo, para que yo pueda ser un ejemplo y motivación para él.

Pablo Gómez Gómez

AGRADECIMIENTO

A la universidad por brindarnos y abrirnos las puertas al conocimiento y siempre teniendo en cuenta las palabras de Don Bosco, de ser “Buenos Cristianos y Honrados Ciudadanos”.

A los Ingenieros, que con sus conocimientos han sabido impartirnos dentro de un salón de clases y han sembrado la semilla de sus enseñanzas, las mismas que nos han servido de impulso a seguir adelante en la adquisición de nuevos conocimientos, que nos servirán para poner en práctica a futuro y que serán de gran beneficio para la sociedad.

Al Ing. Gustavo Navas, por ser nuestro tutor y la persona que nos ha brindado su ayuda incondicional en los momentos que más hemos necesitado de sus conocimientos y experiencia, para así llegar a culminar este proyecto, queremos darle gracias por su paciencia, sus consejos, su apoyo, su amistad y su deseo de que sigamos adelante, brindándonos su ayuda en el transcurso y desarrollo del trabajo de titulación.

Carlos Coba De la Torre

Pablo Gómez Gómez

ÍNDICE

INTRODUCCIÓN 1
CAPÍTULO 1 6
MARCO TEÓRICO 6
1.1 Geoportales 6
1.2 MongoDB 7
1.3 Funcionamiento de MongoDB 8
1.4 Dónde utilizar MongoDB 10
1.5 Dónde no usar MongoDB 11
1.6 Diferencia en la estructura de las consultas 14
1.7 JSON 14
1.8 GeoJSON 15
1.9 Java Server Faces (JSF) 17
1.10 PrimeFaces 17
1.11 Metodología SCRUM 18
1.12 Recolección de información. 20
1.13 Planes para la ejecución del proyecto 21
CAPÍTULO 2 24
FASE INICIAL Y DE DEFINICIÓN 24
2.1 Fase inicial 24
2.1.1 Definición del proyecto. 24
2.1.2 Análisis y requerimientos. 24
2.1.3 Viabilidad técnica. 28
2.1.4 Búsqueda de una solución técnica. 29
2.1.5 Viabilidad financiera. 31
2.2 Fase de definición 32
2.2.1 Definición de las actividades. 32
2.2.2 Diagrama de casos de uso. 33
2.2.3 Caso de uso descripción. 34
2.2.4 Diagrama de actividades. 36
2.2.5 Diagrama de clases. 37
2.2.6 Diagrama de base de datos PostgreSQL. 38
2.2.7 Diagrama de base de datos MongoDB. 38
2.2.8 Creación de los planes para la ejecución. 42
CAPÍTULO 3 43

FASE DE EJECUCIÓN Y ENTREGA 43
3.1 Fase de ejecución 43
3.1.1 Desarrollo 43
3.1.2 Integración del producto. 64
3.1.3 Diagrama de implementación. 66
3.1.4 Prueba del producto. 66
3.2 Fase de entrega 73
3.2.1 Entrega del producto. 73
CAPÍTULO 4 78
FASE DE SOPORTE Y CIERRE DEL PROYECTO 78
4.1 Fase de soporte y mantenimiento 78
4.1.1 Fase del producto para el soporte. 78
4.1.1.1 Instalación de MongoDB en CentOS 78
4.1.1.2 Instalación de JBoss en CentOS. 82
4.2 Fase de cierre del producto 85
4.2.1 Formalización del cierre del proyecto. 85
4.2.1.1 Control del documento 86
4.2.1.2 Matriz de estados. 87
CONCLUSIONES 90
RECOMENDACIONES 91
LISTA DE REFERENCIAS 92
GLOSARIO DE TÉRMINOS 95
ANEXOS 958

ÍNDICE DE FIGURAS

Figura 1. Formato del documento en MongoDB. 7
Figura 2. Estructura de datos MySQL (Relacional) y MongoDB (No Relacional). .. 14
Figura 3. Ejemplo de sintaxis de GeoJSON. 16
Figura 4. Ejemplo de sintaxis de un punto. 16
Figura 5. Ejemplo de sintaxis de un lineString. 16
Figura 6. Ciclo principal de SCRUM que es incremental e iterativo. 18
Figura 7. Fases en la metodología SCRUM. 19
Figura 8. Proceso de la metodología adaptado para este trabajo de titulación. 20
Figura 9. Visualizador y estilos. 25
Figura 10. Visualizador y búsquedas. 26
Figura 11. Muestra de información. 26
Figura 12. Muestra de información. 27
Figura 13. Ejemplo de inicio de base de datos en consola. 29
Figura 14. Ejemplo de test. 30
Figura 15. Diagrama de casos de uso. 33
Figura 16. Diagrama de Clases. 37
Figura 17. Diagrama de PostgreSQL. 38
Figura 18. Importaciones de la librería mongo-2.10.1.jar. 43
Figura 19. Entidad de la Clase Conexión. 44
Figura 20. Constructor de la Clase Conexión. 44
Figura 21. Método de la Clase Conexión. 45
Figura 22. Método de impresión de la Clase Conexión. 46
Figura 23. Clase Obra. 47
Figura 24. Clase Casa. 48
Figura 25. Clase de ServicioCasa. 49
Figura 26. Clase de ServicioObra. 50
Figura 27. Variables de la clase ControladorObra. 51
Figura 28. Fragmento de método de impresión de la Clase Conexión. 52
Figura 29. Método para obtener los datos. 52
Figura 30. Variables de la clase ControladorObra. 53
Figura 31. Controlador de la clase ControladorObra. 54
Figura 32. Método para obtener los datos. 54
Figura 33. Variable de la clase Puntos. 55
Figura 34. Constructor de la clase Puntos. 56
Figura 35. Método de consulta del documento y visualización. 56
Figura 36. Método para guardar el documento en un archivo GeoJSON. 58
Figura 37. Entidad de la clase MultipoligonoClass 58
Figura 38. Método para graficar un multipolígono. 60
Figura 39. Método visualizar el método mapaPunto. 62
Figura 40. Método de división de polígonos a puntos. 63
Figura 41. Método para definir el formato de visualización. 64
Figura 42. Modelo de implementación entra Bases y Clases. 66
Figura 43. Página inicial del visualizador. 67
Figura 44. Resultado del JMeter. 68
Figura 45. Resultado del JMeter. 68

Figura 46. Resultado del JMeter. 69
Figura 47. Página de visualización de puntos. 69
Figura 48. Resultado del JMeter. 70
Figura 49. Resultado del JMeter. 70
Figura 50. Resultado del JMeter. 71
Figura 51. Página de visualización de MultiPolígonos. 71
Figura 52. Resultado del JMeter. 72
Figura 53. Resultado del JMeter. 72
Figura 54. Resultado del JMeter. 73
Figura 55. Creación del repositorio. 78
Figura 56. Edición del repositorio de 64 bits. 79
Figura 57. Vista del repositorio. 79
Figura 58. Creación del repositorio de 32 bits. 79
Figura 59. Instalación del repositorio. 80
Figura 60. Proceso de instalación del repositorio. 80
Figura 61. Comando de arranque del servicio de MongoDB. 81
Figura 62. Verificación que el servicio se está ejecutando. 81
Figura 63. Inicio de consola a MongoDB. 81
Figura 64. Consola de inicio de MongoDB. 81
Figura 65. Página de descarga JBoss. 82
Figura 66. Crear usuario. 82
Figura 67. Prepara ficheros. 82
Figura 68. Descarga del Zip de JBoss-7.1.1. 83
Figura 69. Imagen en servidor descompresión de JBoss. 83
Figura 70 Prepara ficheros para el inicio, parada y reinicio. 83
Figura 71 Editar ficheros de configuración. 84
Figura 72. Editar ficheros en la configuración. 84
Figura 73. Editar ficheros para creación de usuarios. 84
Figura 74. Error y cambio de usuario para la consola de administración. 85
Figura 75. Path de configuración. 85
Figura 76. Pantalla de inicio. 96
Figura 77 Pantalla de selección. 96
Figura 78. Pantalla de selección de casa. 96
Figura 79. Pantalla de selección de obra. 96
Figura 80. Pantalla de vista de generar multipolígono. 96
Figura 81. Pantalla de vista de multipolígono. 96
Figura 82. Pantalla de vista de puntos. 96
Figura 83. Pantalla de selección de puntos. 96
Figura 84. Pantalla de visualización de puntos. 96

ÍNDICE DE TABLAS

Tabla 1. Formato de consultas entre SQL y MongoDB. 8
Tabla 2. Tabla relacional entre NoSQL y SQL. 12
Tabla 3. Tabla de fases de metodología SCRUM. 19
Tabla 4. Roles de la metodología SCRUM. 20
Tabla 5. Tabla de plan de ejecución. 21
Tabla 6. Pila de producto. 28
Tabla 7. Caso de uso gestión de la base de datos. 34
Tabla 8. Caso de uso gestión de descarga. 35
Tabla 9. Caso de uso gestión de interfaz. 35
Tabla 10. Diagrama de base de datos en MongoDB. 38
Tabla 11. Diccionario de datos de la base PostgreSQL 40
Tabla 12. Diccionario de datos de la base PostgreSQL. 41
Tabla 13. Diccionario de la base de MongoDB. 41
Tabla 14. Pila de entregables. 73
Tabla 15. Pila de entregables. 77
Tabla 16. Información del documento. 86
Tabla 17. Historial del documento. 86
Tabla 18. Aprobación. 86
Tabla 19. Matriz de estados. 87

RESUMEN

En la actualidad el uso de geoportales ha ido en aumento y más aún con la aparición de las bases de datos NoSQL. El cuál permite almacenar masivamente los archivos JSON facilitando el uso de estos archivos y usados en otros sistemas de georeferenciación.

Los geoportales se han convertido en poderosas herramientas para la búsqueda y ubicación geográfica, el presente trabajo de titulación está se orienta a realizar una investigación con análisis, diseño, construcción e implementación de un geoportal, con una base de datos NoSQL

La herramienta MongoDB es una base de datos NoSQL que maneja una estructura de datos en forma de documentos llamados BSON, que son similares a JSON y por este motivo también se puede usar datos de tipo GeoJSON, permitiendo así que la integración de los datos en ciertas aplicaciones sea más fácil y rápida.

Para llevar a cabo el desarrollo de este trabajo de titulación, se realizó una investigación exhaustiva en el ámbito de la base de Datos NoSQL y la relación con los lenguajes de programación a utilizar, de este modo el geoportal constara de una base de datos NoSQL como es MongoDB, su capa de negocios con JSF y su capa de presentación con PrimeFaces.

Así se concluye, que con la implementación del geoportal este aportará beneficios a la Universidad Politécnica Salesiana y a sus áreas de investigación, permitiendo identificar si una base de datos SQL es mejor que una NoSQL en el manejo de la información geográfica siendo este el primer geoportal a ser desarrollado con dichas herramientas en el Ecuador.

ABSTRACT

Today the use of geoportal has increased and even more so with the appearance of the NoSQL database. Which allows the massive storage of JSON files, facilitating the use of these files and allowing them to be used in other georeferencing systems.

Geoportals have turned into powerful tools for searching and locating geographical locations. The current job is oriented in carrying out an investigation with analysis, design, construction and the implementation of a geoportal, with a NoSQL database.

MongoDB tool is a database that manages a NoSQL data structure in the form of documents called BSON, which are similar to JSON and for this reason you also can use data of type GeoJSON, this allowing the integration of the data in certain applications faster and easier.

To carry out the development of this thesis, he conducted an exhaustive investigation in the field of database NoSQL and the relationship with the programming languages to use, in this way the Geoportal will consist of a database as is NoSQL MongoDB, your business layer with JSF and its presentation layer with PrimeFaces.

It is concluded, that with the implementation of the Geoportal this will bring benefits to the Universidad Politecnica Salesiana and their research areas, making it possible to identify if a SQL database is better than a NoSQL in the handling of geographic information and this be the first to be Geoportal developed with such tools in Ecuador.

INTRODUCCIÓN

Este documento describe la implementación de un geoportal utilizando la metodología SCRUM en el desarrollo del trabajo de titulación. Incluye junto con la descripción de este ciclo de vida interactivo e incremental para el desarrollo del trabajo de titulación, las herramientas y/o documentos con los que se gestionaran las tareas de desarrollo, requisitos, monitorización y seguimiento de los avances.

De la misma forma se usará la información relacionada sobre el funcionamiento de los geoportales, así como sus ámbitos en la actualidad y el uso de estos, su forma de ser desarrollados, sus métodos de consulta y de investigación. Ya que los geoportales son variados, extensos y los beneficios que se obtiene al utilizar las herramientas geográficas de los que se puede nombrar unos cuantos:

- Fácil ubicación de un sitio en el globo terráqueo
- Información detallada del sitio de interés
- Compatibilidad con diversos dispositivos móviles
- Actualización de la información a través del uso de una base de datos
- Compartir lugares en las redes sociales
- Vinculación con las páginas web, etc.

También se puede nombrar varios campos sociales en los cuales los geoportales son herramientas claves para dar solución, entre los que se mencionaran:

- Turismo
- Educación
- Ciencia y naturaleza
- Aplicaciones móviles
- Universidades
- Arte y cultura, etc.

En la Universidad Politécnica Salesiana se encuentra el Centro de Investigación en Modelamiento Ambiental (CIMA), el cual lleva una labor de trabajo de investigación de más de 3 años. Dentro de las investigaciones que se llevan a cabo en este centro se

tiene: los sistemas de información geográfica (SIG.) y los geoportales. De estos últimos, el CIMA cuenta con su propia plataforma tecnológica que se puede encontrar en www.ide.ups.edu.ec.

Así mismo, se encuentran otros 6 geoportales (hasta 2013) desarrollados en el CIMA, como son:

- Ruta de los Yumbos
- Ideuquito
- Sitios de interés sur de Quito
- Panamericana norte desde Quito a Ibarra
- Geoportal para Unidades Educativas Salesianas
- Geoportal para la Comunidad Salesiana

Estos geoportales han sido desarrollados enteramente por estudiantes de la Universidad Politécnica Salesiana, con Sede en Quito en el Campus Sur como trabajo de titulación y que han servido de ayuda al crecimiento de la comunidad Salesiana. Para la elaboración de estos proyectos se han usado herramientas de Georeferenciación muy poderosas, como Quantum Gis, base de datos relacionales como PostgreSQL.

En los últimos años los geoportales han tenido un crecimiento muy importante en el ámbito de las búsquedas y ubicación de sitios de interés y estos se ha desarrollado con herramientas como:

- Google maps: Permite ver los mapas a nivel mundial.
- Open Layers: Crea visualizadores desde JavaScript con la facilidad de hacer mapas de múltiples fuentes.
- Geo Server: Servidor de software escrito en java que permite al usuario compartir y editar datos geográficos.
- Postgis: Base de datos relacional para objetos espaciales correspondiente a la base de datos PostgreSQL, siendo estas las herramientas más utilizadas en el desarrollo de los geoportales.

Lo dicho anteriormente, obliga a que los nuevos Ingenieros de Sistemas, no se pueden quedar atascados, ni atados a las herramientas que se posee en la actualidad, sino que este encaminado a seguir evolucionando de la mano de la tecnología y a ser usuarios activos de las nuevas herramientas que evolucionan constantemente.

Así, el presente trabajo de titulación se enfocará en la investigación y elaboración de un geoportal con la utilización de nuevas herramientas de Georreferenciación.

El uso de una potente base de datos llamada MongoDB, de libre distribución (Open Source), es una herramienta de almacenamiento de información cuya base de datos es líder en la implementación del concepto "No Relacional", un concepto que anteriormente no tenía cabida en el desarrollo de portales web, pero que en la actualidad ofrece muchos beneficios.

El concepto que se tenía en las bases de datos relacionales al poseer tablas, filas y columnas queda descartado, con la estructura de una base de datos no relacional que maneja el formato de documentos llamados BSON.

Además, esta base de datos ofrece beneficios a los usuarios programadores, pudiendo tener una base que puede poseer escalabilidad y alto rendimiento, además de ser dinámica, se podrán hacer cambios en la base, pudiendo añadir nuevos datos, relaciones y demás. (Escudero, 2013)

También se usaran herramientas que ayudaran a procesar la información obtenida en la recolección de datos de campo, como JSON, que es un formato de intercambio de datos, ofreciendo una ventaja de distribución de la información entre las distintas tecnologías

También se utilizará el formato para la codificación de una variedad de estructuras de datos geográficos, como es GeoJSON y todos estos documentos procesados se almacenaran en la base de datos MongoDB.

Es un reto poder realizar este trabajo de titulación, puesto que son herramientas que están siendo utilizadas con una mayor frecuencia en proyectos de proporciones empresariales, pero que a pesar de ofrecer grandes beneficios, es poca la información que se puede obtener de éstas.

Definición del proyecto

El siguiente trabajo de titulación para la obtención del título de Ingeniero de Sistemas, elaborado por estudiantes de la Universidad Politécnica Salesiana Sede Quito, Campus Sur, con la utilización de herramientas libres y aplicando los conocimientos adquiridos dentro y fuera de la universidad.

El trabajo de titulación plantea el diseño y construcción de un geoportal para el centro de investigación y modelamiento ambiental CIMA, utilizando una nueva herramienta para el almacenamiento de los datos geográficos, como es MongoDB.

Objetivo general

Analizar, diseñar, construir e implementar un geoportal con las herramientas Mongo DB, JSON y GeoJSON para el proyecto IDE-UPS.

Objetivos específicos

- Investigar la base de datos NoSQL Mongo DB y sus documentos JSON y GeoJSON y determinar su funcionalidad para el almacenamiento de datos geográficos.
- Investigar si se puede desarrollar un geoportal con el uso de herramientas como MongoDB, GeoJSON, JSF y PrimeFaces.
- Desarrollar el geoportal utilizando SCRUM como metodología de desarrollo ágil y basado en la administración de proyectos.
- Determinar si el desarrollo es óptimo, ágil y eficiente durante la investigación.
- Demostrar la viabilidad técnica del desarrollo del proyecto y sus posibles soluciones y adaptaciones a otros entornos.
- Realizar pruebas de rendimiento sobre el geoportal y tomar en cuenta los resultados obtenidos

Alcance

El uso de los geoportales se ha convertido en una herramienta primordial en la búsqueda y ubicación geográfica de sitios, permitiendo así realizar una búsqueda de un lugar por medio de un visualizador con relación a una base de datos.

Los geoportales se han desarrollado con un fin, el de mostrar información de un lugar por medio de un visualizador de mapas. Así, es el caso de los geoportales realizados por IDE-UPS perteneciente a CIMA UPS, IGM y MAGAP como instituciones que han desarrollado geoportales permitiendo ver información y ubicación de una búsqueda deseada.

Es por todo lo dicho anteriormente, que se vio en la necesidad de seguir investigando todos los avances con respecto a la georeferenciación. Ya que como es lógico, la evolución en las herramientas tecnológicas nos brindan la oportunidad de realizar cambios que pueden ser: mejoramiento en los tiempos de respuesta, visualización atractiva, consultas más efectivas, etc.

En el presente trabajo de titulación se hace referencia a la implementación de una primera versión de un geoportal, construido con nuevas herramientas de georeferenciación, en cuanto al almacenamiento de datos y consultas de los mismos. Por lo tanto el presente sistema de georeferenciación solo dará alcance a la investigación de dichas herramientas y a la construcción de un portal web con los datos recolectados en el campo. Y este se lo desarrollara con una base de datos NoSQL como es MongoDB y el entorno de desarrollo JSF + PrimeFaces de Java.

De este modo se podrá definir la viabilidad técnica, su funcionalidad y si trabajan en coordinación óptima, en el caso de que no se pueda tener una viabilidad técnica positiva se presentarán posibles soluciones dentro del documento para las próximas versiones.

CAPÍTULO 1
MARCO TEÓRICO

1.1 Geoportales

> El geoportal es un sitio Web cuyo objetivo es ofrecer al usuario, de forma práctica e integrada, el acceso a una serie de recursos y servicios basados en información geográfica. Así, dentro de una infraestructura de datos espaciales, los geoportales resuelven la conexión física y funcional entre los almacenes de datos geográficos y los usuarios de información geográfica. (Moya Honduvilla, Bernabé Poveda, & Manrique Sancho, 2007)

También se puede decir de una manera más formal, que un geoportal es un portal web que utiliza el geo posicionamiento y la navegación con la ayuda de mapas, para mostrar el contenido almacenado en una base de datos

Desde los primeros geoportales desarrollados en su mayoría, se han utilizado lenguajes como PHP, Punto net, etc. Y el almacenamiento de la información en bases de datos como PostgreSQL.

Los geoportales han avanzado a pasos gigantescos, ofreciendo una gran cantidad de ventajas y en variados campos de aplicación.

Los geoportales en la actualidad tienen una gran aceptación por parte de los usuarios, ya sea que éstos accedan desde sus casas o a través de sus dispositivos inteligentes, para localizar un sitio de interés y visualizar su información. Así, el mercado está teniendo un crecimiento de geoportales con diferentes herramientas y para diferentes fines.

Estos geoportales se los están creando con fines: educativo, comercial, de prevención de riesgos, de distracción, de ubicación, etc.

"La información geográfica que la base de datos contendrá puede ser muy variable en cantidad, calidad y diversidad. La cantidad vendrá determinada por las necesidades que el SIG. Abra de satisfacer." (Moreno Jiménez, 2008)

1.2 MongoDB

"MongoDB es una potente base de datos de propósito general, flexible y escalable. Combina la capacidad de escalar con la función como índices secundarios, rangos, clasificar, agregaciones e índices geoespaciales." (Chodorow, 2010)

MongoDB es una base de datos orientada a documentos. Esto quiere decir que en lugar de guardar los datos en registros, guarda los datos en documentos. Estos documentos son almacenados en BSON, que es una representación binaria de JSON.

Una de las diferencias más importantes con respecto a las bases de datos relacionales, es que no es necesario seguir un esquema. Los documentos de una misma colección poseen un concepto similar a una tabla de una base de datos relacional, ya que pueden tener esquemas diferentes.

Se tiene una colección a la que se llama GeoMongoDB. Un documento podría almacenar los datos necesarios de la siguiente manera:


```
{
    "_id" : "9PP",
    "Cod" : "AZYUMUZ01",
    "Nam" : "COMUNIDAD SALESIANA DE UZHUPUD ",
    "Denom" : "OBRA SALESIANA ENCARGADA DE BRINDAR EDUCACIÓN AGROPECUARIA A LA COMUNIDAD ",
    "Pais" : "ECUADOR",
    "Region" : "SUR",
    "Provincia" : "AZUAY",
    "Canton" : "PAUTE",
    "Parroquia" : "CHICAN",
    "Sector" : "UZHUPUD",
    "Responsabl" : "PD. ANGEL LOPEZ",
    "Dir" : "UZHUPUD ",
    "Fono" : "2125120 ",
    "Horario" : "08:00 A 13:30 Y 14:00 A 16:30 ",
    "Email" : "pauteuzhupud@salesianos.org.ec ",
    "Tipo" : "PASTORAL, SOCIAL, EDUCATIVA ",
    "Campo" : "CAPACITACIÓN OCUPACIONAL, JOVENES SALESIANOS, EDUCACIÓN PRIMARIA BÁSICA Y BACHILLERATO ",
    "No_Per" : 6,
    "Benefic" : "UNIDAD EDUCATIVA, GRANJA ",
    "Influencia" : "CUENCA, PAUTE, SEVILLA DE ORO, YUMACAY, UZHUPUD, GUALACEO, GUACHADALA, ZIGZIG, CHORDELEC ",
    "Informacio" : "LA CASA SALESIANA SE ENCARGA DE ENSEÑAR EL OFICIO DE LA AGRICLATURA, CON ENSEÑANZAS DE
    DON BOSCO Y HACIENDO JÓVENES DE BIEN PARA LA SOCIEDAD ",
    "Nam_E" : "VILMA CUZCO ",
    "Email_E" : "vilma_1202lucia@yahoo.es",
    "Tel_En" : "0992649597 ",
    "Fecha" : "23/05/2014",
    "Nam_R" : "PABLO GÓMEZ, CARLOS COBA ",
    "geometry" : {
        "type" : "Point",
        "coordinates" : [
            -78.76622311732071,
            -2.832526258595398
        ]
    }
}
```


Figura 1. Formato del documento en MongoDB.
Elaborado por: Coba Carlos & Gómez Pablo.

El documento anterior es un clásico documento JSON. Tiene strings, arrays, subdocumentos y números. En la misma colección se guardaran infinitas cantidades de documentos, con diferentes esquemas que se muestra en la figura 1.

Desde su primera llave es el inicio del documento, con un diccionario de datos elaborado con los parámetros designados, en la palabra geometry, se abre un nuevo sub documento donde almacena el tipo y las coordenadas, en las coordenadas se crea un vector donde se reciben los parámetros de "latitud" y "longitud" en un formato double.

"Esto que es algo impensable en una base de datos relacional, es algo totalmente válido en MongoDB." (Rubenfa, 2014)

1.3 Funcionamiento de MongoDB

MongoDB está escrito en C++, aunque las consultas se hacen pasando objetos JSON como parámetro. Es algo bastante lógico, dado que los propios documentos se almacenan en BSON.

Tabla 1. Formato de consultas entre SQL y MongoDB.

CONSULTAS EN SQL	CONSULTA EN MongoDB
SELECT * **FROM** users	**db**.**users**.**find**()
SELECT id, user_id, status **FROM** users	**db**.**users**.**find**({ }, { **user_id**: 1, **status**: 1 })
SELECT user_id, status **FROM** users	**db**.**users**.**find**({ }, { **user_id**: 1, **status**: 1, **_id**: 0 })
SELECT * **FROM** users **WHERE** status = "A"	**db**.**users**.**find**({ **status**: "A" })

Continúa…

Tabla 1. Formato de consultas entre SQL y MongoDB. (Continuación...)

|SELECT user_id, status **FROM** users **WHERE** status = "A"	**db**.**users**.**find**({ **status**: "A" }, { **user_id**: 1, **status**: 1, **_id**: 0 })
SELECT * **FROM** users **WHERE** status != "A"	**db**.**users**.**find**({ **status**: { **$ne**: "A" } })
SELECT * **FROM** users **WHERE** status = "A" **AND** age = 50	**db**.**users**.**find**({ **status**: "A", **age**: 50 })
SELECT * **FROM** users **WHERE** status = "A" **OR** age = 50	**db**.**users**.**find**({ **$or**: [{ **status**: "A" } , { **age**: 50 }] })
SELECT * **FROM** users **WHERE** age > 25	**db**.**users**.**find**({ **age**: { **$gt**: 25 } })
SELECT * **FROM** users **WHERE** age < 25	**db**.**users**.**find**({ **age**: { **$lt**: 25 } })
SELECT * **FROM** users **WHERE** user_id **like** "%bc%"	**db**.**users**.**find**({ **user_id**: /bc/ })
SELECT * **FROM** users **WHERE** user_id **like** "bc%"	**db**.**users**.**find**({ **user_id**: /^bc/ })
SELECT * **FROM** users **WHERE** status = "A" **ORDER BY** user_id **ASC**	**db**.**users**.**find**({ **status**: "A" }).**sort**({ **user_id**: 1 })
SELECT * **FROM** users **WHERE** status = "A" **ORDER BY** user_id **DESC**	**db**.**users**.**find**({ **status**: "A" }).**sort**({ **user_id**: −1 })
SELECT COUNT(*) **FROM** users	**db**.**users**.**count**() or **db**.**users**.**find**().**count**()

Continúa...

Tabla 1. Formato de consultas entre SQL y MongoDB. (Continuación...)

SELECT COUNT(user_id) **FROM** users	**db**.**users**.**count**({ **user_id**: { **$exists**: **true** } }) or **db**.**users**.**find**({ **user_id**: { **$exists**: **true** } }).**count**()
SELECT COUNT(*) **FROM** users **WHERE** age > 30	**db**.**users**.**count**({ **age**: { **$gt**: 30 } }) or **db**.**users**.**find**({ **age**: { **$gt**: 30 } }).**count**()
SELECT * **FROM** users **LIMIT** 1	**db**.**users**.**findOne**() or **db**.**users**.**find**().**limit**(1)
SELECT * **FROM** users **LIMIT** 5 SKIP 10	**db**.**users**.**find**().**limit**(5).**skip**(10)
EXPLAIN SELECT * **FROM** users **WHERE** status = "A"	**db**.**users**.**find**({ **status**: "A" }).**explain**()

Elaborado por: Coba Carlos & Gómez Pablo.

Las consultas anteriores permite entender como son las similitudes en MongoDB con una referencia a las base de datos SQL.

“MongoDB se ejecuta bajo consola, esta consola está construida sobre JavaScript, por lo que las consultas se realizan utilizando ese lenguaje. Además de las funciones de MongoDB, se puede utilizar muchas de las funciones propias de JavaScript.” (Rubenfa, 2014)

1.4 Dónde utilizar MongoDB

Aunque suele decirse que las bases de datos NoSQL tienen un ámbito de aplicación reducido, MongoDB se puede utilizar en muchos de los proyectos que se desarrollan en la actualidad.

Cualquier aplicación que necesite almacenar datos semiestructurados puede usar MongoDB. Es el caso de las típicas aplicaciones CRUD o de muchos de los desarrollos web actuales.

Eso sí, aunque las colecciones de MongoDB, no necesitan definir un esquema, es importante que se diseñe en nuestra aplicación para seguir un comportamiento ordenado. Se debe tener en cuenta si se normalizar los datos, de normalizarlos o utilizar una aproximación híbrida. Estas decisiones pueden afectar al rendimiento de nuestra aplicación. En definitiva el esquema lo definen las consultas que se vayan a realizar con más frecuencia.

"MongoDB es especialmente útil en entornos que requieran escalabilidad. Con sus opciones de replicación y sharding, que son muy sencillas de configurar, se puede conseguir un sistema que escale horizontalmente sin demasiados problemas." (Rubenfa, 2014)

1.5 Dónde no usar MongoDB

En esta base de datos no existen las transacciones. Aunque la aplicación puede utilizar alguna técnica para simular las transacciones, MongoDB no tiene esta capacidad. Solo garantiza operaciones atómicas a nivel de documento. Si las transacciones son algo indispensable en el desarrollo, se deberá pensar en otro sistema.

> Tampoco existen los JOINS. Para consultar datos relacionados en dos o más colecciones, se tiene que hacer más de una consulta. En general, si los datos pueden ser estructurados en tablas, y utilizar las relaciones, es mejor que optemos por un RDBMS clásico. (Rubenfa, 2014)

> Para almacenar los documentos, MongoDB utiliza una serialización binaria de JSON, llamada BSON, que es una lista ordenada de elementos simples. El núcleo de la base de datos es capaz de interpretar su contenido, de modo que lo que a simple vista parece un contenido binario, realmente es un documento que contiene varios elementos. Estos datos están limitados a un tamaño máximo de 16 MB; para tamaños superiores se requiere del uso de GridFS. (Requena, 2010)

La estructura principal y fundamental que maneja MongoDB se le presenta de forma equivalente a una base de datos relacional y de orden jerárquico.

Tabla 2. Tabla relacional entre NoSQL y SQL.

MongoDB – NoSQL	PostgreSQL - SQL
Server	Servidor
Data base	Base de Datos
Collection	Tabla
Document (BSON; JSON)	Fila
Fields	Columna

Elaborado por: Coba Carlos & Gómez Pablo.

> MongoDB al ser una base de datos no relacional posee características principales. Es decir que las consultas son a nivel de colecciones (sin joins), los índices se generan en las colecciones, los documentos tendrán un ID única y su atomicidad es a nivel de documentos son de tipo JSON como esquema dinámico, haciendo que la integración de datos en ciertas aplicaciones sea más fácil y rápida. (Aulet, 2011)

Las características comunes entre las implementaciones de bases de datos distribuidas no relacionales o NoSQL son las siguientes:

- **Consistencia:** No se implementan mecanismos rígidos de consistencia como los presentes en las bases de datos relacionales, donde la confirmación de un cambio implica una comunicación del mismo a todos los nodos que lo repliquen. Las bases de datos NoSQL son ACID, esto significa que una transacción cumple lo siguiente:

- **Atomicidad:** Es la propiedad que asegura que la operación se ha realizado o no, y por lo tanto ante un fallo del sistema no puede quedar a medias.

- **Consistencia:** Es la propiedad que asegura que sólo se empieza aquello que se puede acabar. Por lo tanto se ejecutan aquellas operaciones que no van a romper la reglas y directrices de integridad de la base de datos.

- **Aislamiento:** Es la propiedad que asegura que una operación no puede afectar a otras. Esto asegura que la realización de dos transacciones sobre la misma información sean independientes y no generen ningún tipo de error.

- **Durabilidad:** Es la propiedad que asegura que una vez realizada la operación, ésta persistirá y no se podrá deshacer aunque falle el sistema.

- **Estructura distribuida:** Generalmente se distribuyen los datos mediante mecanismos de tablas de hash distribuidas como las redes P2P.

- **Escalabilidad Horizontal:** La implementación típica se realiza en muchos nodos de capacidad de procesado limitado, en vez de utilizar grandes 'mainframes'.

- **Tolerancia a fallos** (debido a la estructura ACID), redundancia y sin cuellos de botella.

La indexación en MongoDB también es posible ya que permite que cualquier campo en un documento de MongoDB pueda ser indexado, al igual que es posible la creación de índices secundarios.

> Cada base de datos en MongoDB consiste en colecciones que son equivalentes a una base de datos RDBMS (Sistema de gestión de base de datos relacionales SGBDR) o relacionales que consiste en tablas SQL. Cada base de datos, almacena los datos de recogida en la forma de documentos que es equivalente a las tablas que almacenan datos en filas. (Mario, 2014)

Mientras que almacena los datos de fila en el conjunto de columnas, un documento tiene una estructura JSON (conocido como BSON en MongoDB). Por último, la forma en que se tienen las filas en una fila de SQL, que tiene campos en MongoDB.

Figura 2. Estructura de datos MySQL (Relacional) y MongoDB (No Relacional).
Fuente: (Aulet, 2011)

1.6 Diferencia en la estructura de las consultas

Como se puede suponer a simple vista, al tener la estructura de almacenamiento diferente entre una base de datos relacional y una no relacional, existe una gran diferencia en la estructura de las consultas (query). Las consultas en la base de datos MongoDB son rápidas ahorrando recursos de procesamiento debido a la forma en que se almacenan los datos.

1.7 JSON

Es un formato de texto que es totalmente independiente del lenguaje que se esté utilizando, pero utiliza convenciones que son ampliamente conocidos por los programadores de la familia de lenguajes C, incluyendo C, C++, C#, Java, JavaScript, Perl, Python, y muchos otros. (Json, 2012).

JSON (JavaScript Object Notation) es un formato para el intercambios de datos, básicamente JSON describe los datos con una sintaxis dedicada que se usa para identificar y gestionar los datos. JSON nació como una alternativa a XML, el fácil uso en JavaScript ha generado un gran número de seguidores de esta alternativa. Una de las mayores ventajas que tiene el uso de JSON es que puede ser leído por cualquier lenguaje de programación. Por lo tanto, puede ser usado para el intercambio de información entre distintas tecnologías. (Rodriguez, 2013)

Estas características entre otras, hacen que la utilización de JSON sea una herramienta muy importante al momento del intercambio de datos.

JSON está constituido por dos estructuras:

- Una colección de pares de nombre/valor. En varios lenguajes esto es conocidos como un objeto, registro, estructura, diccionario, tabla hash, lista de claves o un arreglo asociativo.
- Una lista ordenada de valores. En la mayoría de los lenguajes, esto se implementa como arreglos, vectores, listas o secuencias. (Json, 2012)

Al usar estas estructuras, los lenguajes de programación de cierta forma las terminan interpretando, ya que al ser una herramienta de intercambio que no depende del lenguaje, es lógico que contenga estas estructuras.

Sintaxis

El primer elemento de JSON es el objeto u object, este se conforma de una llave de apertura, el nombre del objeto entre comillas, dos puntos, el valor dado al objeto y una llave de cierre.

Tipos de datos

El valor puede tomar cualquiera de los siguientes tipos de datos:

- **String** (Cadenas de texto)
- **number** (números)
- **object** (Objetos)
- **char** (caracteres unicode válidos)
- **array** (Un arreglo o colección de valores)
- **null** (nulo)
- **boolean** (valores true o false)

1.8 GeoJSON

GeoJSON es un formato para la codificación de una variedad de estructuras de datos geográficos. Un objeto GeoJSON puede representar una geometría, una función o un conjunto de características. GeoJSON soporta los siguientes tipos de geometría: puntos, líneas, Polígonos, Multipuntos, Multilíneas, Multipolígonos y GeometryCollection. Características de GeoJSON contienen

un objeto de geometría y propiedades adicionales, y una colección de fenómenos representan una lista de características. (Requena, 2010)

Con el fin de los datos de índice GeoJSON, debe almacenar los datos en un campo de ubicación que usted nombre. El campo de dirección contiene un subdocumento con un tipo de campo que especifica el tipo de objeto GeoJSON y coordenadas campo especificando las coordenadas del objeto. Siempre guarde las coordenadas de longitud, latitud orden. (Requena, 2010)

Utilice la siguiente sintaxis:

```
{ < ubicación de campo > : { tipo : "<GeoJSON type>" ,
                    coordenadas : < coordenadas >
} }
```

Figura 3. Ejemplo de sintaxis de GeoJSON.
Elaborado por: Coba Carlos & Gómez Pablo.

El ejemplo siguiente almacena un GeoJSON Point:

```
{ loc : { tipo : "Punto" , coordenadas : [ 40 , 5 ] } }
```

Figura 4. Ejemplo de sintaxis de un punto.
Elaborado por: Coba Carlos & Gómez Pablo.

El ejemplo siguiente almacena un GeoJSON LineString:

```
{ loc : { tipo : "LineString" ,
coordenadas : [ [ 40 , 5 ] , [ 41 , 6 ] ]
} }
```

Figura 5. Ejemplo de sintaxis de un lineString.
Elaborado por: Coba Carlos & Gómez Pablo.

Una estructura de datos GeoJSON completa es siempre un objeto (en términos JSON). En GeoJSON, un objeto compuesto por una colección de pares nombre / valor – también llamados miembros. Para cada miembro, el nombre es siempre una cadena. Valores miembros son o bien una cadena, número, objeto, matriz o uno de los literales: "true", "falso" y "nulo". Una matriz se compone de elementos, donde cada elemento es un valor como se describe anteriormente. (Howard, y otros, 2008)

1.9 Java Server Faces (JSF)

Java Server Faces (JSF) es una tecnología y framework para aplicaciones Java basadas en web que simplifica el desarrollo de interfaces de usuario en aplicaciones Java EE. JSF usa Java Server como la tecnología que permite hacer el despliegue de las páginas, pero también se puede acomodar a otras tecnologías como XUL (acrónimo de XML – based User - interface Language, lenguaje basado en XML para la interfaz de usuario). (Tong, 2009)

JSF en la actualidad de uno de los principales recursos en el desarrollo web con el uso de JSP permitiendo de este modo una mejor funcionalidad con los servidores relacionados a esta plataforma y estándar. De este modo JSF es el encargado de crear las interfaces de usuario por lado del servidor.

JSF se basa en la arquitectura Modelo – Vista – Controlador (MVC). El modelo MVC representa los datos de la aplicación, y se lleva a cabo típicamente usando Plain Old Java Objects (POJOs) basado en la API de JavaBeans. El punto de vista de MVC representa las interfaces de usuario, si la solicitud y es responsable de representar los datos y utiliza los controles de la interfaz para el usuario. El controlador de MVC representes un objeto que responde a los eventos de interfaz de usuario y se ocupa de consultar o modificar el modelo. (Hlavats, 2009)

1.10 PrimeFaces

Es una librería de componentes para Java Server Faces (JSF) de código abierto que cuenta con un conjunto de componentes enriquecidos que facilitan la creación de las aplicaciones web. PrimeFaces está bajo la licencia de Apache License V2. Una de las ventajas de utilizar PrimeFaces, es que permite la integración con otros componentes como por ejemplo RichFaces. (Çivici, 2014)

Algunas características de PrimeFaces

Se enumera algunas de las cosas que ofrece PrimeFaces:

- Un interesante conjunto de componentes (editor HTML, autocompletado, gráficas,…)
- Soporte para Ajax, basándose en el estándar JSF 2.0 Ajax API
- Sin dependencias, ni configuraciones, además de ser muy ligero (1802Kb en su versión 3.5)
- Soporte para interfaces de usuario sobre dispositivos móviles, provee de un kit para este menester.
- Múltiples temas de apariencia, listos para usar.
- La documentación, para mi forma de entender, está muy currada y organizadita
- Amplia difusión del framework, con lo cual existe una comunidad que respalda al proyecto. (Calendamaia, 2013)

1.11 Metodología SCRUM

Es una metodología ágil y flexible para gestionar el desarrollo de software, cuyo principal objetivo es maximizar el retorno de la inversión para su empresa (ROÍ). Se basa en construir primero la funcionalidad de mayor valor para el cliente y en los principios de inspección continua, adaptación, auto-gestión e innovación. (Palacios, 2006)

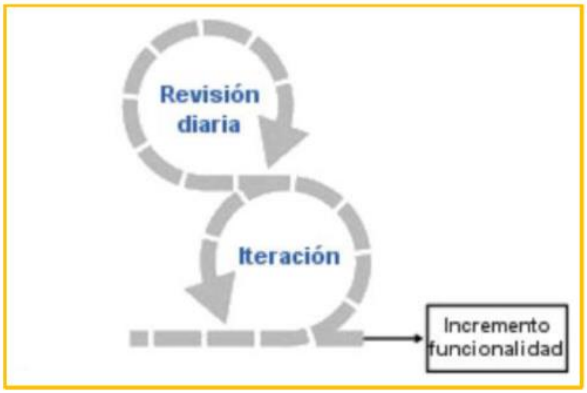


Figura 6. Ciclo principal de SCRUM que es incremental e iterativo.
Fuente: *(Palacios, 2006)*

Las fases de la metodología SCRUM permiten evaluar constantemente el avance de nuestro producto, como se ha mencionado, es una metodología incremental e iterativa,

por lo que se le considera una metodología ágil e ideal para alcanzar el desarrollo del presente trabajo de titulación de grado. Es decir la metodología SCRUM servirá para tener un control más específico de cada una de las tareas, y formarán el denominado ciclo de vida de un proyecto.

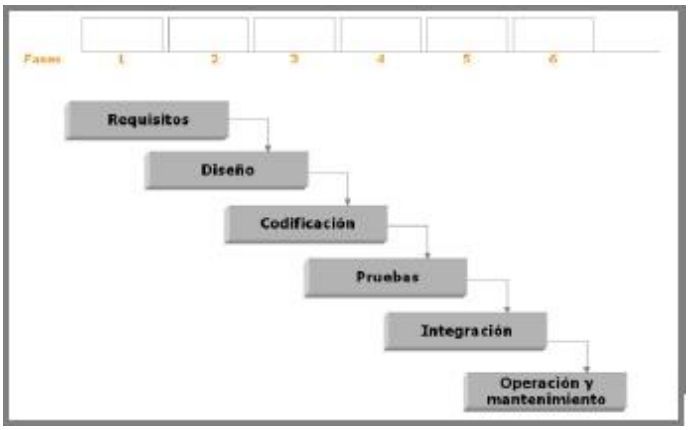


Figura 7. Fases en la metodología SCRUM.
Fuente: *(Sumosa, 2011)*.

En el contexto de la metodología SCRUM se va a definir las siguientes partes a tomar en consideración la efectuación y seguimiento del desarrollo del proyecto con la metodología mencionada anterior mente los cuales consta de los siguientes pasos:

Tabla 3. Tabla de fases de metodología SCRUM.

Fase Inicial			
Análisis y Requerimientos	Viabilidad técnica	Búsqueda de una solución técnica	Viabilidad financiera
Fase de Definición			
Definición de las actividades	Creación de una diagrama de actividades	Crear los planes para la ejecución	
Fase de Ejecución			
Desarrollo	Integración del producto	Prueba del producto	
Fase de Entrega			
Entrega del producto			
Fase de Soporte y Mantenimiento			
Desarrollo de productos para el soporte			
Fase Cierre del Proyecto			
Formalización del cierre proyecto y las lecciones aprendidas			

Elaborado por: Coba Carlos & Gómez Pablo.

Las principales razones del uso de un ciclo de desarrollo iterativo e incremental de tipo SCRUM para la ejecución de este proyecto son:

- Sistema modular. Las características del visualizador geográfico permiten desarrollar una base funcional mínima y sobre ella ir incrementando las funcionalidades o modificando el comportamiento de las ya implantadas.
- Entregas frecuentes y continuas al tutor de trabajo de titulación de los módulos terminados, de forma que puede disponer de una funcionalidad básica en un tiempo mínimo y a partir de ahí un incremento y mejora continua del sistema.

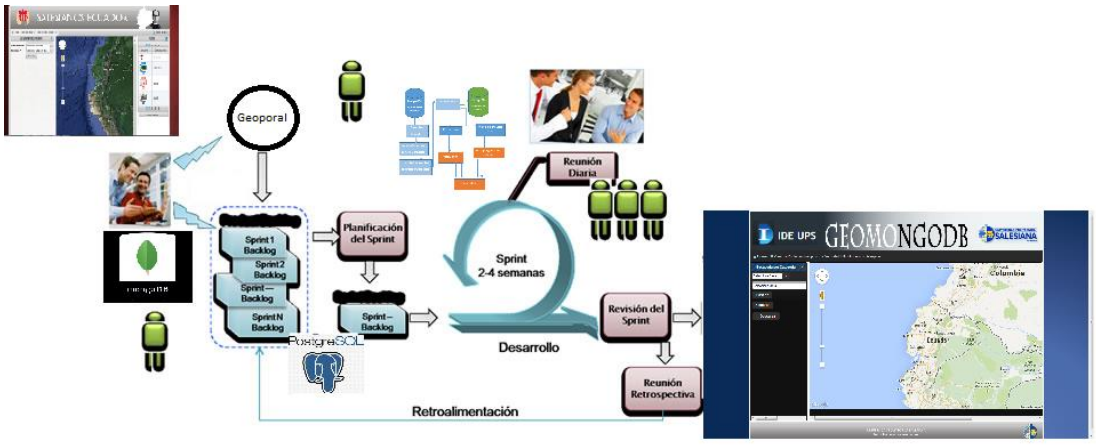


Figura 8. Proceso de la metodología adaptado para este trabajo de titulación.
Elaborado por: Coba Carlos & Gómez Pablo.

Personas y roles del proyecto

Tabla 4. Roles de la metodología SCRUM.

Persona	Contacto	Rol
Gustavo Navas	gnavas@ups.edu.ec	Tutor
Carlos Coba	alejo12088@gmail.com	Desarrollador y Documentador
Pablo Gómez	pGómezghds@gmail.com	Desarrollador y Documentador

Elaborado por: Coba Carlos & Gómez Pablo.

1.12 Recolección de información.

La recolección de información de Georeferenciación para el uso del presente trabajo de titulación, se la realizó en las ciudades de Cuenca, Paute y Machala, las que tienen una Casa Salesiana en cada cantón y de las que se derivan en obras salesianas. Así, se pudo conocer las diferentes obras que constan en cada cantón como es el caso del cantón Paute que da el servicio de estudio en su Colegio Agropecuario Salesiano, en

Machala en sus iglesias parroquiales y en Cuenca en la Universidad Politécnica Salesiana, Colegio Técnico Salesiano y en la Editorial Don Bosco.

Para recolectar los datos se usó un GPS Garmin proporcionado por el CIMA, para luego estos datos obtenidos poderlos manipular en sistemas de georeferenciación, como el QGIS (anteriormente llamado Quantum GIS)

1.13 Planes para la ejecución del proyecto

Se ha elaborado el siguiente plan para el control y seguimiento del proyecto de acuerdo a la metodología.

Cabe recalcar que los tiempos expuestos en el siguiente plan, son tiempos estimados de ejecución, pero siempre tratando de cumplir con los entregables en el rango de las fechas establecidas.

Tabla 5. Tabla de plan de ejecución.

Plan de ejecución del proyecto				
Objetivo	**Actividad**	**Entregable**	**Responsable**	**Fecha**
Analizar los requerimientos	Recolectar información.	Plan de trabajo de titulación	Carlos Coba Pablo Gómez	02/09/2013
	Revisar el plan de trabajo de titulación.	Tema, objetivos, plan de trabajo de titulación	Tutor de trabajo de titulación	05/09/2013
	Manejar geoportales desarrollados.	Esquema de funcionalidades	Carlos Coba Pablo Gómez	02/09/2014
	Entregar documentación a la cima.	Proyectos desarrollados en el cima	Tutor de trabajo de titulación	20/09/2013
	Investigar sobre base de datos NoSQL.	Documentos en línea.	Carlos Coba Pablo Gómez	25/09/2013
	Relación de la base MongoDB con lenguajes de programación.	Documentos en línea, foros y blog.	Carlos Coba Pablo Gómez	30/09/2013
Recolectar información de campo.	Viajar a las ciudades de Paute y Machala.	Archivo obtenido en el GPS.	Carlos Coba Pablo Gómez	10/06/2014
	Elaborar cuadro de visitas.	Plan de recolección de datos.	Tutor de trabajo de titulación	01/06/2014
	Cargar los datos shapefile.	Archivos shapefile de las casas salesianas.	Carlos Coba Pablo Gómez	10/06/2014

Continúa…

Tabla 5. Tabla de plan de ejecución. (Continuación...)

	Crear los shapefiles.	Diccionario shapefile de las casas salesianas.	Pablo Gómez	20/06/2014
	Armar los polígonos de las casas salesianas.	Archivo generado por el quantum gis.	Pablo Gómez	20/06/2014
	Revisión y aprobación de los datos recolectados.		Tutor de trabajo de titulación	21/06/2014
	Guardar la información en la base	Crear la base de datos	Carlos Coba	21/06/2014
Cargar la información al geoportal	Realizar pruebas de consistencia	Probar la base de datos	Carlos Coba	21/06/2014
	Crear la estructura del documento de trabajo de titulación	Plan de trabajo de titulación	Carlos Coba Pablo Gómez	10/10/2013
Desarrollar el contenido de la trabajo de titulación	Investigar la documentación	Capítulo 1	Carlos Coba Pablo Gómez	10/10/2013
	Documentar los errores	Capítulo 1	Carlos Coba Pablo Gómez	31/01/2014
	Buscar herramientas de desarrollo	Capítulo 2	Pablo Gómez Carlos Coba	18/02/2014
Viabilidad técnica	Manejar las herramientas	RoboMongo instalado	Carlos Coba	21/03/2014
	Probar las herramientas en Linux	U mongo para Linux	Carlos Coba Pablo Gómez	25/03/2014
	Analizar errores	Log	Carlos Coba	26/03/2014
	Crear una base de datos	Base de datos en MongoDB para usar	Carlos Coba Pablo Gómez	13/02/2014
Ejecución de las herramientas	Analizar varios geoportales	Comparativa de las herramientas	Carlos Coba Pablo Gómez	21/08/2014
Comparar la funcionalidad de las herramientas	Documentar la comparación de análisis.	Viabilidad técnica	Carlos Coba Pablo Gómez	23/08/2014
	Documentar los pasos ejecutados	Documento de soporte para nuevas versiones	Pablo Gómez	18/08/2014

Continúa…

Tabla 5. Tabla de plan de ejecución. (Continuación...)

Finalizar el programa	Crear conclusiones y recomendaciones	Conclusiones y recomendaciones	Pablo Gómez Carlos Coba	20/08/2014
	Entregar el programa	Geoportal terminado	Pablo Gómez Carlos Coba	01/09/2014
	Revisar la documentación y el programa	Proyecto aprobado	Tutor de trabajo de titulación	01/09/2014

Elaborado por: Coba Carlos & Gómez Pablo.

CAPÍTULO 2
FASE INICIAL Y DE DEFINICIÓN

2.1 Fase inicial

En esta fase se definen los principios que se siguieron en el desarrollo de la trabajo de titulación, como también se identifica su viabilidad técnica, tomando en cuenta los requerimientos iniciales para la finalización de este proyecto.

2.1.1 Definición del proyecto.

El proyecto está definido como un primer visualizador a desarrollarse por primera vez dentro del país, con las herramientas que se han ido explicando y describiendo una por una. Es, así que se han implementado las diferentes fases de desarrollo, como también el uso de la base de datos NoSQL MongoDB, JSF y PrimeFaces y servidor JBoss.

2.1.2 Análisis y requerimientos.

Los geoportales que actualmente están desarrollados y se encuentran en los servidores del CIMA, funcionan con una base de datos relacional de libre distribución, como es PostgreSQL, la cual ofrece multitud de funcionalidades, pero en la actualidad el crecimiento de las bases con otra visión no relacional como es el caso de NoSQL, ha provocado que la velocidad de procesamiento de las mismas sea mejor, es por este motivo que se decidió trabajar con una base de datos NoSQL (MongoDB) para hacer un análisis de las ventajas que ésta presenta.

En el departamento de CIMA, se observó y analizó que la velocidad de procesamiento de la información dentro de los servidores al realizar una consulta a la base de datos poseía tiempos muy inferiores a los deseados, lo que provoca que el sistema se vuelva pesado y que no agrade al usuario final, se planteó realizar un sistema que involucre la investigación de herramientas que den una solución óptima a esta problemática.

El presente trabajo de titulación trata, sobre el desarrollo de un sistema de Georeferenciación para la Universidad Politécnica Salesiana, la que contará con una herramienta automatizada sobre algunos puntos obtenidos y recopilados por un GPS.

El sistema está desarrollado sobre un ambiente web, que se ejecuta sobre un servidor JBoss, su capa de negocios es JSF y su capa de presentación sobre PrimeFaces,

permitiendo así una vista amigable del geoportal. Así, como también el uso de la base de datos NoSQL, como es Mongo DB en su capa de Datos.

De este modo se podrán utilizar datos recopilados durante un trabajo de campo, realizado en la Comunidad Salesiana de Machala y Paute. Permitiendo probar estos datos en el visualizador a ser desarrollado, para luego hacer una comparación en los tiempos de respuesta dentro del procesamiento de consulta, utilizando una base de datos NoSQL vs la base de datos relacional con las que cuenta el CIMA.

Los geoportales desarrollados anteriormente servirán de guía para el análisis de los puntos recolectados, interfaces y ubicación de los nuevos sitios a ser publicados en el geoportal a ser desarrollado en este trabajo de titulación. Así, se podrá observar que el uso de esta herramienta como es MongoDB, permite el manejo de datos en JSON y GeoJSON a pesar de ser una base de datos NoSQL, con el visualizador a ser desarrollado en Java.

Figura 9. Visualizador y estilos.
Fuente: (Cofre & Toledo, 2014)

Figura 10. Visualizador y búsquedas.
Fuente: (Cofre & Toledo, 2014)

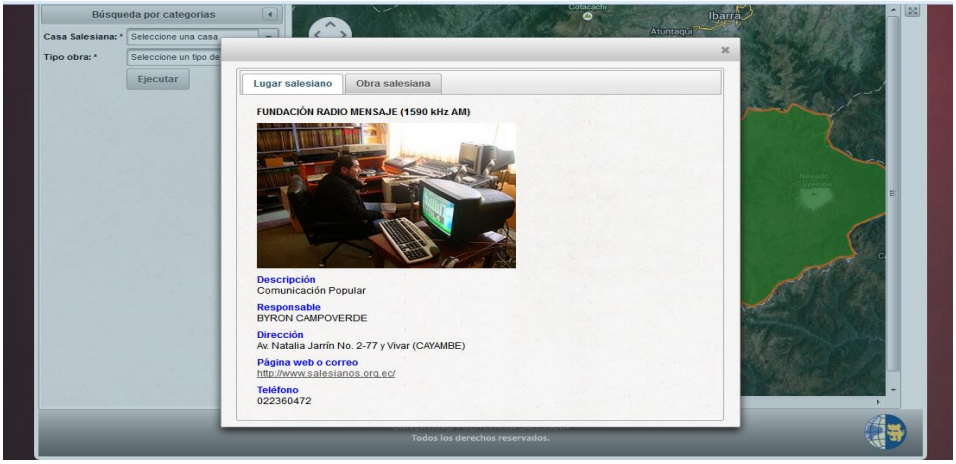


Figura 11. Muestra de información.
Fuente: (Cofre & Toledo, 2014)

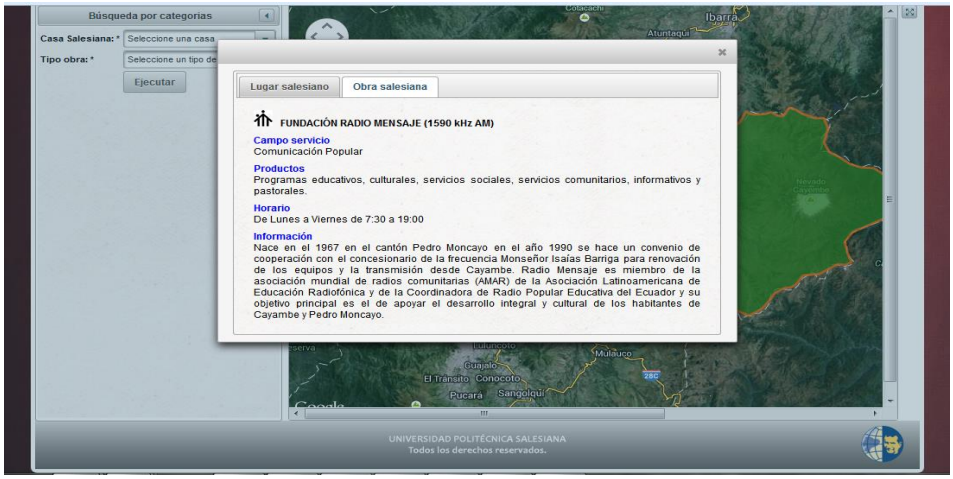


Figura 12. Muestra de información.
Fuente: (Cofre & Toledo, 2014)

Entre los requerimientos funcionales pertinentes al software en desarrollo es que permitan ver e identificar obras salesianas pertenecientes a Ecuador, para eso se ha implementado el uso de una base de datos NoSQL ya que durante la investigación, esto permitirá determinar si una base de datos no relacional obtiene datos de forma más rápida durante una consulta.

Con el propósito de que el tema se cumpla de una mejor manera se utilizan herramientas de licencia libre o GLP, y el uso de un sistema operativo robusto como es Centos OS. Así, también el uso de herramientas de visualización de datos recolectados con GPS y modificados con QGIS Desktop 2.2.0

El uso de la base de datos NoSQL como Mongo DB pertenecientes al grupo de licencias libres, permite tener una mayor ventaja en el uso de esta herramienta. El servidor web que se utilizará para la ejecución de nuestra plataforma web es JBoss, ya que es un servidor de aplicación de software libre y que trabaja bajo la plataforma Java EE.

La interfaz gráfica que se diseñará para la presentación de la información geográfica, se lo realizará obteniendo los datos recolectados por medio de consultas realizadas a la base de datos y esto se lo desarrollara en Java con la utilización de los PrimeFaces y JSF.

En el desarrollo del proyecto de trabajo de titulación se podrá identificar posibles errores que puedan efectuarse entre las diferentes capas que son la de datos, negocio y presentación. Así, también encontrar las soluciones correctas en el funcionamiento para próximas versiones que se podrán realizar.

Tabla 6. Pila de producto.

Pila de Producto: GeoMongoDB			
Requerimientos	**Prioridad**	**Estimación de valor**	**Estimación de fuerza inicial (Horas)**
Gestionar datos de una base de datos NoSQL.	1	10	192
Gestionar datos de una base de datos SQL.	2	9	192
Gestionar la interacción entra base de datos SQL y NoSQL.	3	8	192
Consumir datos en formato GeoJSON	4	7	192
Visualizar datos geográficos desde una base de datos NoSQL.	5	6	192
Revisar, corregir, integrar y probar cada módulo de gestión.	6	5	192
			2112

Elaborado por: Coba Carlos & Gómez Pablo.

2.1.3 Viabilidad técnica.

Para desarrollar el geoportal web, se han tomado en cuenta tanto el hardware como el software a utilizarse en la elaboración de la misma.

En cuanto a software se utilizaran sistemas de libre distribución, como:

- Sistema Operativo: CentOS
- Base de datos: MongoDB

Y en cuanto a hardware se puede decir que los servidores con los que actualmente cuenta el CIMA, se acoplan e integran a la tecnología con la que se va a desarrollar el geoportal. Así mismo se utilizarán computadoras que permitan y garanticen el correcto desarrollo del sistema.

2.1.4 Búsqueda de una solución técnica.

En el transcurso de la elaboración del trabajo de titulación, se evaluará si al realizar una consulta a MongoDB, este puede recuperar de forma rápida los resultados deseados, ya que por su modelo de consultas a documentos y sin la dependencia de la relación entre tablas, el resultado a probar es que esta base es veloz en procesamiento de la información.

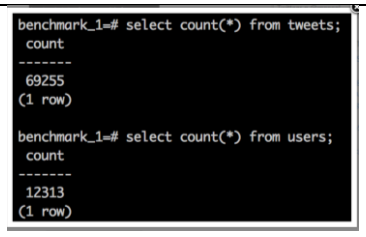
Para identificar este se ha tomado un ejemplo de una página amiga que permite ver las diferencias que se indican a (Continuación...):

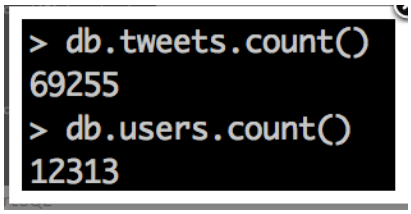
> La parte que a (Continuación...) se presenta, está sacada de una página web, tal como lo menciona Correa, 2013: "Bueno, después de algún tiempo de escuchar comentarios como "MySQL no te sirve para un sitio pequeño, PostgreSQL es mucho mejor", o "MongoDB no es bueno", me decidí por probar por mí mismo la velocidad de cada uno de estos motores. (Correa, 2013)

Preparando la DB

"Empecé colocando en cada uno, una base de datos real de tuits, que he estado recolectando desde hace unos dos años con un agente (@cuxibamba ; utiliza MySQL, y, de momento, tiene **69255** tuits y **12313** usuarios)" (Correa, 2013)

Inicio de la base de datos con el comando count





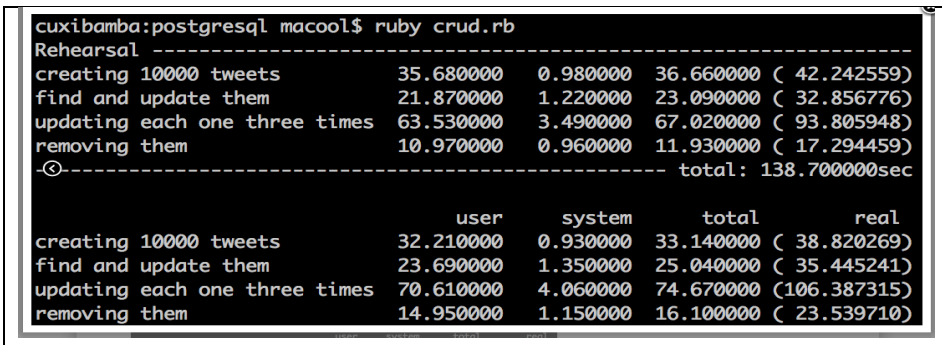
PostgreSQL	Mongo DB

Figura 13. Ejemplo de inicio de base de datos en consola.
Fuente: *(Correa, 2013)*

Pasos a seguir para el test a realizar:

1. Crear 10000 (diez mil) tuits.
2. Encontrarlos (por su id único) y actualizar su texto (guardándolo en DB nuevamente).

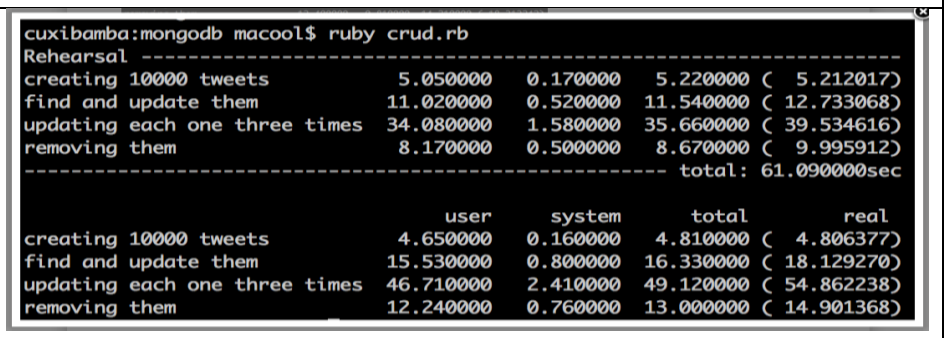
3. Encontrar nuevamente los 10000 tuits y actualizar tres veces el contenido de cada uno, guardando en DB cada vez que se actualice. (Es decir, tres veces por tuit)
4. Encontrar cada uno de los 10000 tuits (por su id) y eliminarlo



```
cuxibamba:postgresql macool$ ruby crud.rb
Rehearsal ------------------------------------------------------------------
creating 10000 tweets          35.680000   0.980000  36.660000 ( 42.242559)
find and update them           21.870000   1.220000  23.090000 ( 32.856776)
updating each one three times  63.530000   3.490000  67.020000 ( 93.805948)
removing them                  10.970000   0.960000  11.930000 ( 17.294459)
--------------------------------------------------- total: 138.700000sec

                                   user     system      total        real
creating 10000 tweets          32.210000   0.930000  33.140000 ( 38.820269)
find and update them           23.690000   1.350000  25.040000 ( 35.445241)
updating each one three times  70.610000   4.060000  74.670000 (106.387315)
removing them                  14.950000   1.150000  16.100000 ( 23.539710)
```

PostgreSQL



```
cuxibamba:mongodb macool$ ruby crud.rb
Rehearsal ------------------------------------------------------------------
creating 10000 tweets          5.050000   0.170000   5.220000 (  5.212017)
find and update them          11.020000   0.520000  11.540000 ( 12.733068)
updating each one three times 34.080000   1.580000  35.660000 ( 39.534616)
removing them                  8.170000   0.500000   8.670000 (  9.995912)
--------------------------------------------------- total: 61.090000sec

                                   user     system      total        real
creating 10000 tweets          4.650000   0.160000   4.810000 (  4.806377)
find and update them          15.530000   0.800000  16.330000 ( 18.129270)
updating each one three times 46.710000   2.410000  49.120000 ( 54.862238)
removing them                 12.240000   0.760000  13.000000 ( 14.901368)
```

MongoDB

Figura 14. Ejemplo de test.
Fuente: *(Correa, 2013)*

Resultados:

PostgreSQL:

- 38.82 segundos para almacenar diez mil tuits.
- 35.45 segundos para encontrar y actualizar diez mil tuits.
- 106.39 segundos para encontrar los tuits y actualizar tres veces cada uno.
- 23.54 segundos para eliminar los tuits.
- **Total**: 204.2 segundos.

MongoDB:

- 4.81 segundos para almacenar diez mil tuits.
- 18.13 segundos para encontrar y actualizar diez mil tuits.
- 54.86 segundos para encontrar los tuits y actualizar tres veces cada uno.
- 14.9 segundos para eliminar diez mil tuits.
- **Total:** 92.7 segundos (casi la mitad de MySQL, menos de la mitad de PostgreSQL)

Conclusiones:

- MongoDB podría llegar a ser el doble de rápido comparado con PostgreSQL.
- A PostgreSQL le costó bastante actualizar la información de cada tuit."

En la (Continuación...) del trabajo de titulación se irán identificando los posibles errores que se emitan desde la instalación de las herramientas a utilizar, con el transcurso de la construcción del geoportal y sus respectivas fases de pruebas y de finalización. Así, también se documentarán las respectivas soluciones técnicas para el buen funcionamiento del sistema a la finalización del trabajo de titulación.

2.1.5 Viabilidad financiera.

Al utilizar herramientas libres se puede aprovechar al máximo el uso de las misma, de igual forma se invertirán en cursos de desarrollo de la herramienta a usar, pertenecientes a la familia Java como son los PrimeFaces, conexión a base de datos NoSQL con el lenguaje en cuestión y la debida instalación del servidor web como lo es JBoss.

Se tomará en cuenta el uso de artículos de investigación y de equipos que ayudarán en la recopilación de datos como el GPS. Así, como también la implementación de máquinas virtuales, para la construcción e integración del sistema sobre la plataforma de CentOS.

En la fase se contemplará y se explicará cómo se organizó el manejo de la recolección de la información, los diagramas de clases y de base de datos.

2.2 Fase de definición

En la fase de definición se presentarán los diagramas que se han utilizado, como es el de casos de usos, de actividades, de clases y de base de datos. Así, también las actividades y tiempos a seguir durante el desarrollo del trabajo de titulación.

2.2.1 Definición de las actividades.

- Recolectar la información del manejo de la metodología SCRUM.
- Investigar sobre la base de datos NoSQL y la instalación de MongoDB.
- Definir el código de programación para la búsqueda por documentos.
- Viaje a Machala y Paute para la recolección de datos geográficos con el GPS.
- Integración de los datos recolectados con la programación.
- Definir interfaz para el visualizador.
- Presentación de la interfaz con los datos recolectados.
- Implementación del sistema en los servidores del CIMA.
- Realizar pruebas de funcionalidad, en comparación con los anteriores sistemas desarrollados con otras herramientas.
- Conclusiones y recomendaciones

2.2.2 Diagrama de casos de uso.

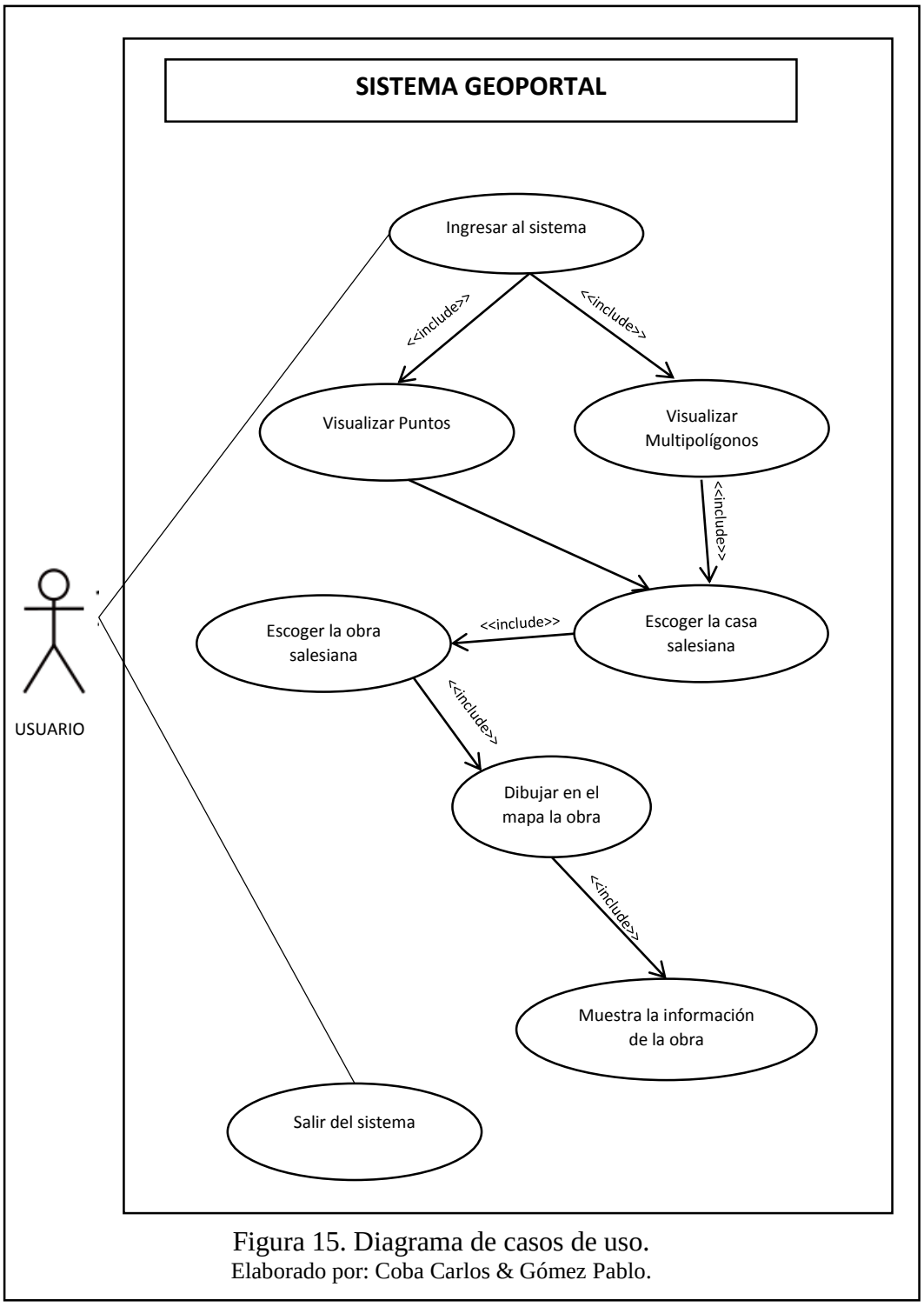


Figura 15. Diagrama de casos de uso.
Elaborado por: Coba Carlos & Gómez Pablo.

2.2.3 Caso de uso descripción.

Tabla 7. Caso de uso gestión de la base de datos.

Caso de uso	**Gestión de base de datos**
Escenarios relacionados	Gestión de interfaz
Actores	Administrador Desarrollador
Camino principal **Administrador**	1. Obtiene el archivo shapefile 2. Crea un objeto GeoJSON 3. Crea una colección en la base de datos 4. Ingresa la información en la colección
Desarrollador	1. Crea un nuevo botón para ingresar directamente la casas 2. Conexión a la base para ingresar la información
Camino Secundario	1.1 No se puede almacenar la información 1.2 .1 Verificar la integridad del archivo GeoJSON 1.2.2 Volver a generar el archivo 1 Información incompleta 1.1 Llenar todos los datos obligatorios 1.2 No ingresar con caracteres especiales
Precondiciones	Manejar la información obtenida de un gps en un sistema georreferenciar
Pos condiciones	Dibujar las coordenadas del archivo generado

Elaborado por: Coba Carlos & Gómez Pablo.

Tabla 8. Caso de uso gestión de descarga.

Caso de uso	**Gestión de descargas**
Escenarios relacionados	Gestión de interfaz Gestión de base de datos
Actores	Administrador Desarrollador
Camino principal **Administrador**	1 Crear el archivo GeoJSON 2 Cargar la información 3 Descargar
Desarrollador	1 Crea un nuevo botón para descargar el archivo consultado 2 Conexión a la base para descargar el archivo seleccionado
Camino Secundario	1. No se puede descargar el archivo 2. Verificar que el archivo se encuentre almacenado 3. Volver a generar el archivo 4. Volver a subir el archivo
Precondiciones	Crear un archivo GeoJSON Almacenar el archivo en una dirección accesible
Pos condiciones	Descargar el archivo GeoJSON con la información consultada

Elaborado por: Coba Carlos & Gómez Pablo.

Tabla 9. Caso de uso gestión de interfaz.

Caso de uso	**Gestión de interfaz**
Escenarios relacionados	Gestión de base de datos
Actores	Administrador Desarrollador
Camino principal **Administrador**	1. Creación de choice 2. Creación de tablas 3. Creación de botones 4. Colocar información de las consultas
Desarrollador	1. Estructurar el código 2. Desarrollar relaciones entre los datos consultados 3. Crear conexiones seguras 4. Reutilización del código
Camino Secundario	1.1 Interfaz no agradable 1.1.1 Mapas que no se dibujan a causa de la no conexión con los datos de la base
Precondiciones	Base de datos normalizada
Pos condiciones	Consultar casa, obra y dibujarla en el mapa

Elaborado por: Coba Carlos & Gómez Pablo.

2.2.4 Diagrama de actividades.

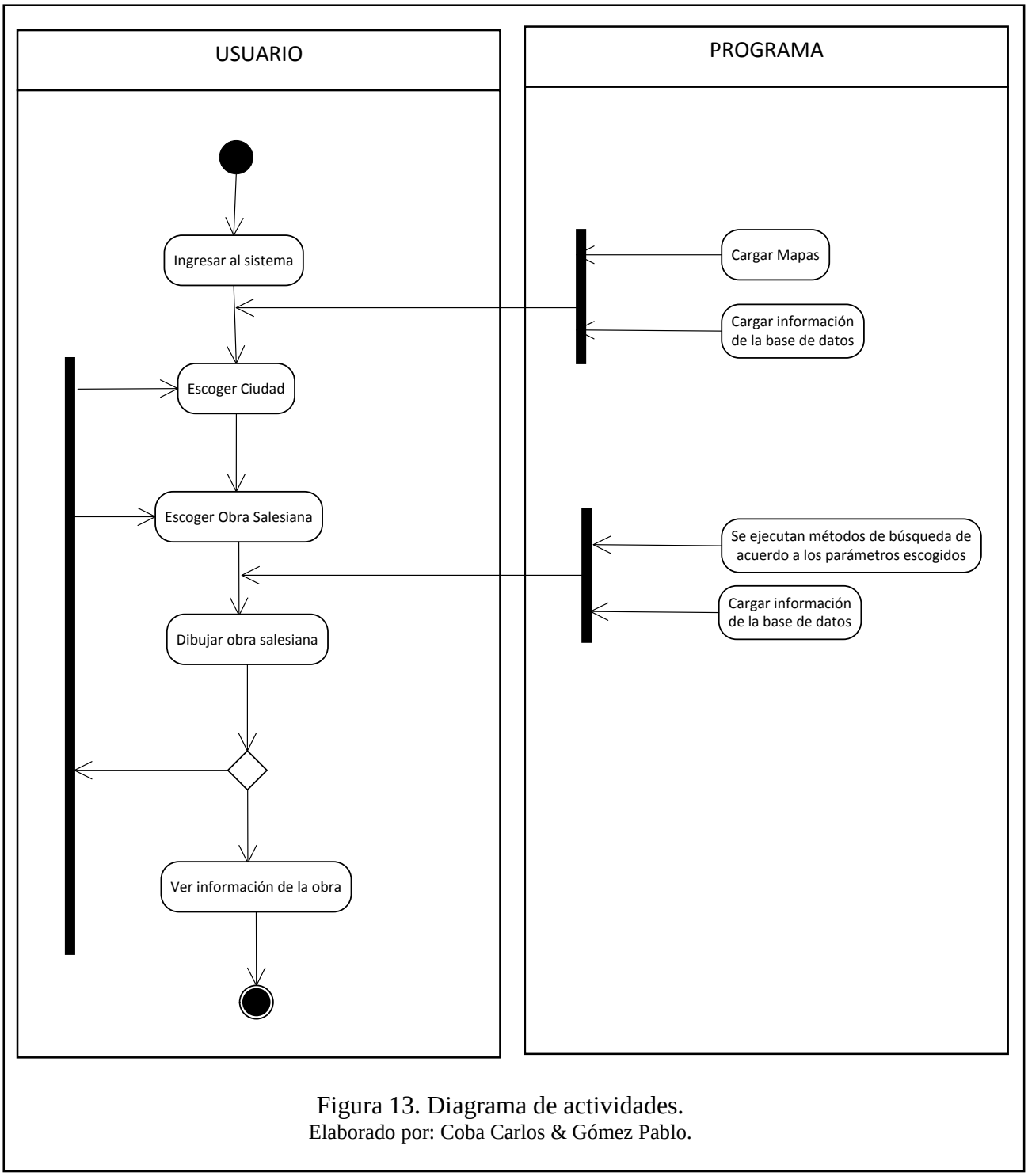


Figura 13. Diagrama de actividades.
Elaborado por: Coba Carlos & Gómez Pablo.

2.2.5 Diagrama de clases.

Figura 16. Diagrama de Clases.
Elaborado por: Coba, Carlos & Gómez Pablo.

2.2.6 Diagrama de base de datos PostgreSQL.

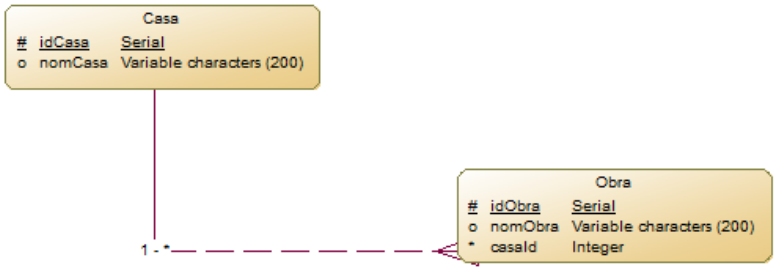


Figura 17. Diagrama de PostgreSQL.
Elaborado por: Coba Carlos & Gómez Pablo.

2.2.7 Diagrama de base de datos MongoDB.

MongoDB no maneja estándares gráficos como es, en el caso de las bases de datos relacionales, este es un estándar Elaborado independientemente con la ayuda de los Ingenieros de la UPS para la base de datos NoSQL. A, más de eso MongoDB no tiene una interfaz gráfica con la que se puede visualizar de mejor manera, para esto se utilizó la herramienta GUI, que es RoboMongo.

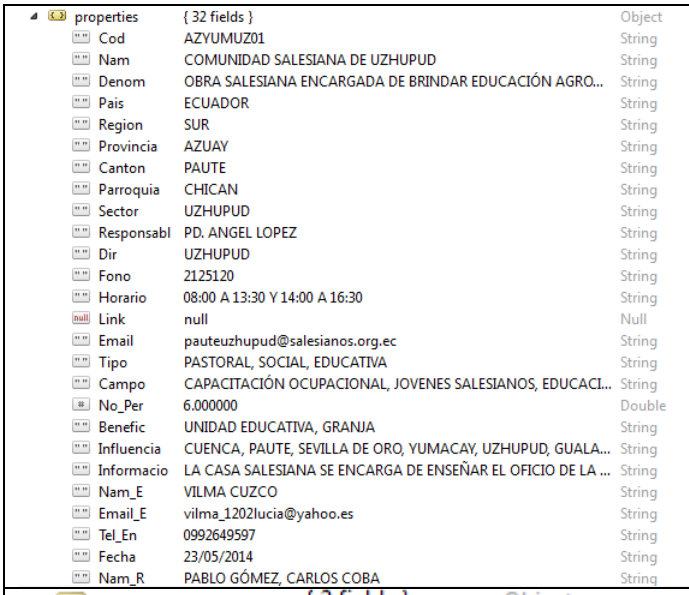
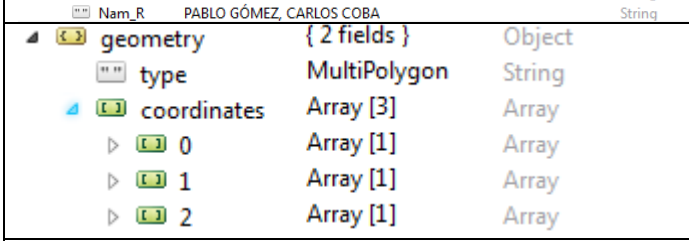
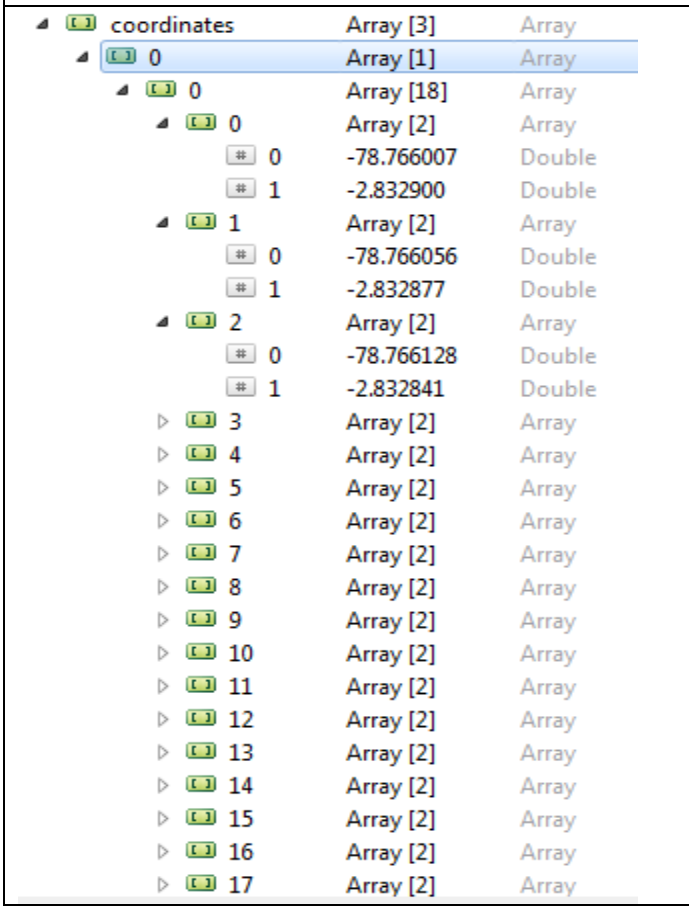
Permitiendo entender de mejor manera el manejo de los niveles de documentación de la base de datos en MongoDB.

Tabla 10. Diagrama de base de datos en MongoDB.

Forma gráfica representada con Robo Mongo un GUI de MongoDB	**Niveles de profundidad del documento JSON**
Key / Value / Type (1) 5L — { 3 fields } — Object _id — 5L — String type — FeatureCollection — String	**Primer Nivel** Nivel de definición de tipo y de ID.
features — Array [1] — Array 0 — { 4 fields } — Object type — Feature — String id — 0.000000 — Double	**Segundo Nivel - features** Nivel de tipo y de segundo id.

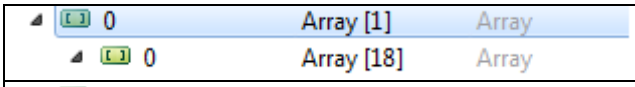
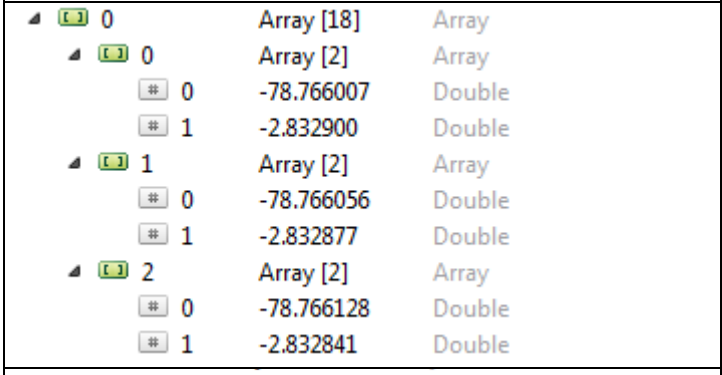
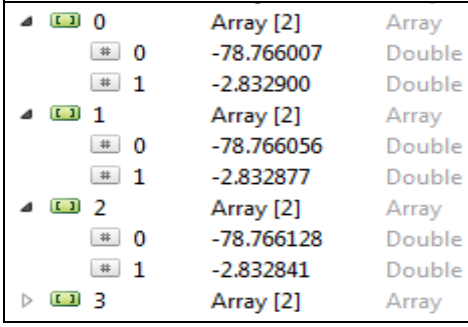
Continúa…

Tabla 10. Diagrama de base de datos en MongoDB. (Continuación...)

	**Tercer Nivel - Propiedades** Se encuentra el Diccionario de datos de la recolección de información.
	**Cuarto Nivel – Geometrías.** Tienes dos sub niveles que el tipo de la geometría y las coordenadas que pertenecen a la geometría.
	**Quito Nivel – Coordenadas.** Se encuentra toda la información recolectada en modo de vectores iniciando desde un Multipolígonos.

Continúa…

Tabla 10. Diagrama de base de datos en MongoDB. (Continuación...)

	**Sexto Nivel – Primer Vector del Multipolígonos.**
	**Séptimo Nivel – Vector de Polígonos.**
	**Octavo Nivel – Vector de Puntos**

Elaborado por: Coba Carlos & Gómez Pablo.

La base de datos NoSQL como es MongoDB, al no manejar relaciones, esta se puede leer de mejor manera y las consultas pueden ser más óptimas, ya que solo se maneja con documentos.

Diccionario de la base de datos de PostgreSQL

El siguiente diccionario de datos, muestra la forma en la que está compuesta la base de datos desarrollada de PostgreSQL.

Tabla 11. Diccionario de datos de la base PostgreSQL

Tabla casa		**Fecha: 28/07/2014**		**V 1.1**	
Tabla	**Descripción**	**Relación**	**Campo(s) primario(s)**	**Clave(s) foránea(s)**	
Casa	Nombre de la casa salesiana	Tabla obra	Id_casa		
Columna	**Tipo de datos**	**Descripción**	**Tamaño**	**Relaciones**	**Clave**
Id_casa	Integer	El id único que identifica a la casa salesiana, autoincremental	Variable	Tabla obra	Primary key
Nom_casa	Character	Se guarda la información de la casa, esto es el nombre	200	No	No

Elaborado por: Coba Carlos & Gómez Pablo

Tabla 12. Diccionario de datos de la base PostgreSQL.

Tabla obra		**Fecha: 28/07/2014**		**V 1.1**	
Tabla	**Descripción**	**Relación**	**Campo(s) primario(s)**	**Clave(s) foránea(s)**	
Obra	Nombre de la obra salesiana	Tabla casa	Id_obra	Id_casa	
Columna	**Tipo de datos**	**Descripción**	**Tamaño**	**Relaciones**	**Clave**
Id_obra	Integer	El id único que identifica a la obra salesiana, autoincremental	Variable	Tabla casa	Primary key
Nom_obra	Character	Se guarda la información de la obra, esto es el nombre	200	No	No
Casa_id	Integer	El id único que identifica a la casa salesiana, se vincula con la tabla casa	Auto	No	Foreign key

Elaborado por: Coba Carlos & Gómez Pablo

Diccionario de la base de datos realizada en MongoDB

El presente diccionario de datos muestra la estructura de cómo está armada la base de datos en MongoDB, al no poseer una estructura relacional, todos los datos están identificados por un ID que representa cada dato de la base, así no existe un ID único o Primary Key.

Tabla 13. Diccionario de la base de MongoDB.

Campo.	**Tipo de dato.**	**Descripción.**
Cod	Texto	Código único de la obra salesiana basada en las 2 primeras consonantes de la provincia, 3 primeras letras del sector, 1 letra de alguna descripción y numeración, por ejemplo provincia pichincha (pc), sector ups (ups), descripción campus sur (s), obra 1 (01), código pcupss01
Nam	Texto	Nombre de la casa salesiana
Denom	Texto	Denominación de la obra salesiana
País	Texto	País donde se encuentra la obra
Región	Texto	Región donde se encuentra la obra: costa, sur, oriente
Provincia	Texto	Provincia donde se encuentra la obra
Cantón	Texto	Cantón donde se encuentra la obra
Parroquia	Texto	Parroquia donde se encuentra la obra
Sector	Texto	Sector donde se encuentra la obra, si no se conoce puede ser el mismo que la parroquia
Responsable	Texto	Responsable de la obra
Dir	Texto	Dirección
Fono	Texto	Teléfonos
Horario	Texto	Horario de atención

Continúa...

Tabla 13. Diccionario de la base de MongoDB. (Continuación...)

Link	Texto	Link del portal web
Email	Texto	Correo electrónico obra salesiana
Tipo	Texto	Pastoral, social, salud, medios de comunicación, auto gestionada, educativa, y otros
Campo	Texto	Campo de servicio pastoral: capacitación ocupacional, jóvenes campesinos y en riesgo de exclusión, educación primaria, básica y bachillerato, catequesis, comunicación popular, promoción humana, pastoral, atención sacramental: bautismos , primeras comuniones, confirmaciones, matrimonios, enfermos,
No_Per	Entero	Número de colaboradores
Benefic	Texto	Beneficiarios: agrupados por categorías. P ej. obra parroquia:
Influencia	Texto	Área de influencia: recabar las áreas sobre la cual la obra tiene influencia como por ejemplo límites geográficos por calles, barrios, parroquias.
Información	Texto	Información de la obra en (100 líneas)
Nam_E	Texto	Nombre del encuestado
Email_E	Texto	Correo electrónico encuestado
Tel_En	Texto	Teléfono encuestado
Fecha	Date	Fecha encuesta
Type	Texto	Define el tipo de dato geométrico que hacer referencia ej. polígono, multipolígono, puntos, etc...
Coordinates	Double	Es un vector que almacena datos de formato doublé ya que es el valor numérico de la longitud y de la latitud

Elaborado por: Coba Carlos & Gómez Pablo.

2.2.8 **Creación de los planes para** la **ejecución.**

Se han definido varías tareas y/o actividades que se deben desarrollar a lo largo de un año de trabajo. Éstas deberán ser cumplidas en los tiempos establecidos y deberán seguir la metodología aplicada.

En el siguiente cronograma, se puede visualizar el tiempo que tomará desarrollar dichas tareas y/o actividades:

CAPÍTULO 3
FASE DE EJECUCIÓN Y ENTREGA

3.1 Fase de ejecución

En esta fase de describirá el código que se utilizó para el desarrollo del programa, las consultas que se han utilizado, las definiciones y el manejo de la información en las capas de desarrollo.

3.1.1 Desarrollo.

En el desarrollo se explicarán los métodos que se utilizaron para la creación del producto visual de la trabajo de titulación, se mostrará parte del código en Java que se desarrolló y se utilizó para realizar las diferentes consultas entre el sistema, la base de datos y la presentación.

Clase conexión - Librerías para el desarrollo

La importación de la librerías y el uso de estas permite conectar a sus diferentes clases que se manejan internamente, dentro del uso de las librerías que se utilizan para el desarrollo del programa, se puede enunciar que son las últimas versiones, tanto para MongoDB con el de PrimeFaces y su "template", los cuales son:

```
3  import java.net.UnknownHostException;
4  import com.mongodb.BasicDBObject;
5  import com.mongodb.DB;
6  import com.mongodb.DBCollection;
7  import com.mongodb.DBCursor;
8  import com.mongodb.Mongo;
9  import com.mongodb.MongoException;
10
```

Figura 18. Importaciones de la librería mongo-2.10.1.jar.
Elaborado por: Coba Carlos & Gómez Pablo.

- PrimeFaces-4.0.jar
- Dot-luv-10.jar
- Mongo 2.10.1.jar
- Gson 2.4.4.jar

Clase conexión - Variables

En la clase conexión sus variables permiten hacer referencia a las diferentes propiedades de las librerías a usar, en la clase conexión.

```
11  public class Conexion {
12      private Mongo conn;
13      private DBCollection coll;
14      private DB db;
15      private BasicDBObject doc;
16      private DBCursor cursor;
```

Figura 19. Entidad de la Clase Conexión.
Elaborado por: Coba Carlos & Gómez Pablo.

Línea 11. Se inicia la clase.

Línea 12. Se define la variable con el nombre (conn), que es de tipo Mongo que pertenece a librería de mongo-2.10.1.jar.

Línea 13. Se define la variable con el nombre coll, que es de tipo DBCollection que se refiere a la colección a consultar.

Línea 14. Se define la variable con el nombre db, que es de tipo DB que se refiere al nombre de la base a conectarnos.

Línea 15. Se define la variable con el nombre doc, que es de tipo BasicDBObject que se refiere al documento de impresión desde MongoDB.

Línea 16. Se define la variable con el nombre cursos, que es de tipo DBCursor, que se refiere a la función find de MongoDB.

Clase conexión - Constructor para la conexión a la base de datos

El constructor inicializa las variables, consumiendo parámetros de entrada que hacen referencia a las propiedades de la base de datos.

```
58      // constructor de la clase conexion
59      public Conexion() {
60          try {
61              db = conectar("geomap");
62              System.out.println("conection base");
63              coll = db.getCollection("GeoMongoDB");
64              System.out.println("conection a la colecion");
65              doc = new BasicDBObject();
66          } catch (MongoException e) {
67              e.printStackTrace();
68              System.out.println("error de conexion" + e);
69          }
70      }
```

Figura 20. Constructor de la Clase Conexión.
Elaborado por: Coba Carlos & Gómez Pablo.

Línea 61. Se usa la variable db, que se definió anteriormente y se escribe el nombre de la base con la que se va a conectar.

Línea 62. Se imprime un mensaje indicando que la conexión a la base se realizó.

Línea 63. Se usa la variable coll, donde se asigna la colección que se utiliza y se almacenan los documentos.

Línea 64. Se imprime un mensaje indicando que se ha realizado la conexión a la colección.

Línea 65. Se usa la doc, donde se transforma la consulta de tipo documento a tipo DBObject, o lo que se entendería como un objeto propio de Java.

Clase conexión - Método de la conexión

El método conexión hace referencia a la IP o localhost y el puerto para hacer una conexión estable con la base de datos de MongoDB.

```
72      // metodo dw conexion
73      public DB conectar(String nombd) {
74          db = null;
75          try {
76              conn = new Mongo("localhost", 27017);
77              db = conn.getDB(nombd);
78          } catch (UnknownHostException e) {
79              // TODO Auto-generated catch block
80              e.printStackTrace();
81          }
82          return db;
83      }
```

Figura 21. Método de la Clase Conexión.
Elaborado por: Coba Carlos & Gómez Pablo.

Línea 74. La variable db se la inicia como nula en caso de que no tenga colecciones creadas, de este modo permite crea una colección nueva.

Línea 76. La variable conn, es la que permite direccionar por medio de nuestra librería, asignando nuestra IP o local hosts y el puerto a utilizar, MongoDB trabaja en el puerto 27017.

Línea 77. La variable db, reconoce el nombre de la base de datos anteriormente ubicada y permite que se cree el llamado a la base de datos a utilizar.

Clase conexión - Método de consulta por id

El método de consulta hace referencia a una simulación a una consulta directa en MongoDB, recibiendo como parámetro el id, nom, provincia, etc. Así, pudiendo hacer una consulta con cualquier parámetro de entrada de tipo String.

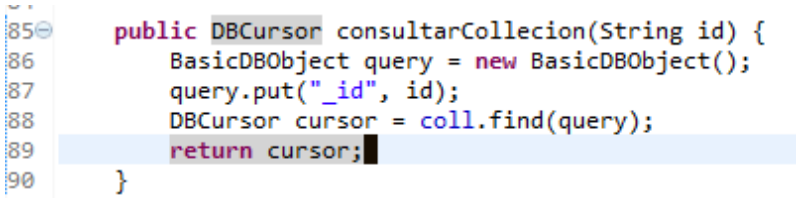


Figura 22. Método de impresión de la Clase Conexión.
Elaborado por: Coba Carlos & Gómez Pablo.

Línea 85. Se define un método de tipo DBCursor, en el que se recibirá un parámetro de entrada, que en este caso es el Id.

Línea 86. Se define un objeto de tipo query, que permite realizar la consulta a la base de datos.

Línea 87. En el objeto query, utilizando la palabra reservada put, este permitirá recibir un parámetro de tipo String id, con el cual se podrá definir el parámetro a consultar.

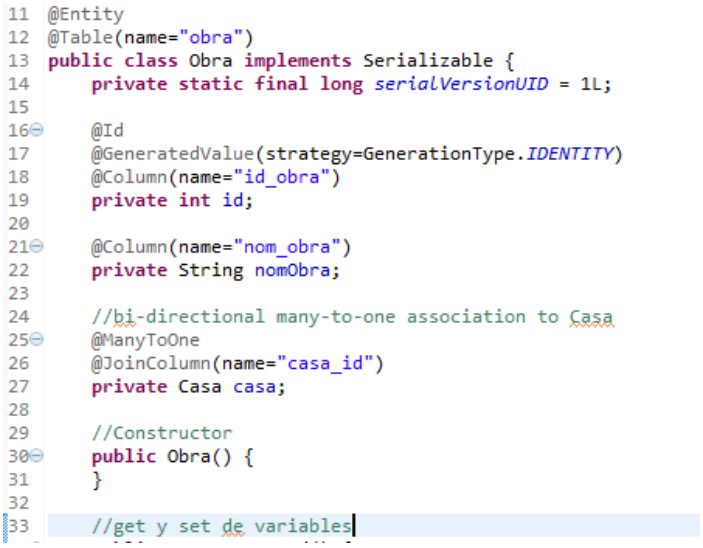
Línea 88. Se invoca a la variable cursor y se inicializa el llamando a la colección con la variable coll, permitiendo imprimir con la palabra reservada find, el cual recibe el parámetro del objeto query.

Línea 89. Retorna la consulta que se desea con el parámetro de entrada, que es el Id.

Clase Obra – Entidades

La clase entidad es generada con un JPA, de este modo esta clase puede hacer una referencia a una persistencia y una conexión a la base de PostgreSQL, por medio de la consola de JBoss con la persitencia.xml y la generación de la librería persistencia.jar.

De este modo se crean todos los parámetros de la conexión a PostgreSQL, para la debida interacción entre los datos de la tabla y de las clases, con una referencia que facilita la absorción de los datos a utilizar. De este modo, nos permite el ahorro de memoria, dado que los datos se consumen desde memoria cache y no desde la misma base de datos.



```
11 @Entity
12 @Table(name="obra")
13 public class Obra implements Serializable {
14     private static final long serialVersionUID = 1L;
15
16     @Id
17     @GeneratedValue(strategy=GenerationType.IDENTITY)
18     @Column(name="id_obra")
19     private int id;
20
21     @Column(name="nom_obra")
22     private String nomObra;
23
24     //bi-directional many-to-one association to Casa
25     @ManyToOne
26     @JoinColumn(name="casa_id")
27     private Casa casa;
28
29     //Constructor
30     public Obra() {
31     }
32
33     //get y set de variables
```

Figura 23. Clase Obra.
Elaborado por: Coba Carlos & Gómez Pablo.

Línea 11. Se define la clase con la anotación @Entity, con la se especifica que la clase es una entidad.

Línea 12. Se define la anotación @Table, con la se especifica la relación entre la clase y la tabla de PostgreSQL

Línea 16. Se define la anotación @Id, con la que se identifica que la primera variable es un id.

Línea 17. Se define la anotación @GeneratedValue, con la que se especifica el tipo identity de auto generación.

Línea 18, 21. Se crea la relación entre la columna de la base de datos y el nombre de su campo.

Línea 19, 22. Se define la variable con el mismo tipo de datos, con que la base se creó.

Línea 25. Se define la anotación @ManyToOne, con la permite hacer la relación de Varios a Uno, que pertenece a la base de datos.

Línea 26. Se define la anotación @JoinColumn, con la que se crea un relación de tipo Joins, con el campo de la base de datos**.**

Línea 30. Se crea el constructor de la clase con sus Get y Set.

Clase Casa – Entidades

```java
/**
 * The persistent class for the casa database table.
 *
 */
@Entity
@Table(name = "casa")
public class Casa implements Serializable {
    private static final long serialVersionUID = 1L;

    @Id
    @GeneratedValue(strategy=GenerationType.IDENTITY)
    @Column(name="id_casa")
    private int id;

    @Column(name="nom_casas")
    private String nomCasas;

    //bi-directional many-to-one association to Obra
    @OneToMany(mappedBy="casa")
    private List<Obra> obras;

    //contrsuctor
    public Casa() {
    }

    //get y set de variables
```

Figura 24. Clase Casa.
Elaborado por: Coba Carlos & Gómez Pablo.

Línea 12. Se define la clase con la anotación @Entity, con la se especifica que la clase es una entidad.

Línea 13. Se define la anotación @Table, con la se especifica la relación entre la clase y la tabla de PostgreSQL

Línea 17. Se define la anotación @Id, con la que se identifica que la primera variable es un id.

Línea 18. Se define la anotación @GeneratedValue, con la que se especifica el tipo identity de auto generación.

Línea 19, 22. Se crea la relación entre la columna de la base de datos y el nombre de su campo.

Línea 20, 23. Se define la variable con el mismo tipo de datos, con que la base se creó.

Línea 26. Se define la anotación @OneToMany, con la permite hacer la relación de Uno a Varios, que pertenece a la base de datos.

Línea 30, 33. Se crea el constructor de la clase con sus Get y Set.

Clase Servicio Casa

Se usa la librería persistencia.jar, que permite hacer una consulta de la misma forma que se hace en PostgreSQL y también ingresar y buscar por un parámetro de entrada.

```
12 @Stateless
13 public class ServicioCasa {
14
15     @PersistenceContext
16     private EntityManager em;
17
18     public void insertarCasa(Casa casa){
19         em.persist(casa);
20     }
21
22     @SuppressWarnings("unchecked")
23     public List<Casa> recuperarTodo(){
24         Query consulta = em.createQuery("select c from Casa c");
25         return consulta.getResultList();
26     }
27
28     public Casa buscarCasaPorId(Integer id){
29         System.out.println("ID CASA A BUSCAR "+id);
30         Query consulta = em.createQuery("select c from Casa c where c.id = :id");
31         consulta.setParameter("id", id);
32         return(Casa) consulta.getSingleResult();
33     }
34 }
```

Figura 25. Clase de ServicioCasa.
Elaborado por: Coba Carlos & Gómez Pablo.

Línea 12. Se define el tipo de protocolo de comunicación @Stateless de sesiones.

Línea 15. Se define el tipo de protocolo @PersistenceContext, para la generación de persistencia.

Línea 16. Se define la variable con el nombre em, que es de tipo Administrador de entidades, que genera ingresos y consultas.

Línea 22. Se define el protocolo @SuppressWarnigs, que desactiva las advertencias de compilación.

Línea 23. Se inicia el método de recuperación de todos los datos.

Línea 24. Se crear una variable de tipo Query, donde se escribe la consulta a realizar en la base de datos.

Línea 25. Retorna el valor generado de la consulta de la línea anterior.

Línea 28. Se inicia el método definiendo el nombre buscarCasaPorId, que es de la clase Casa y que recibe un parámetro de entrada.

Línea 30. Se crea una variable de tipo Query, donde se escribe la consulta a realizar en la base de datos.

Línea 31. Se define la variable consulta, donde recibe el set de un parámetro, que identifica el id con el dato del parámetro de entrada.

Línea 32. Retorno el tipo Casa y que devuelve el valor de la consulta generada.

Clase Servicio Obra

```
12  @Stateless
13  public class ServicioObra {
14
15      @PersistenceContext
16      private EntityManager em;
17
18      public void insertarObra(Obra obra){
19          em.persist(obra);
20      }
21
22      @SuppressWarnings("unchecked")
23      public List<Obra> recuperarTodo(){
24          Query consulta = em.createQuery("select o from Obra o");
25          return consulta.getResultList();
26      }
27
28      public Obra buscarObraPorId(Integer id){
29          Query consulta = em.createQuery("select o from Obra o  where o.id = :id");
30          consulta.setParameter("id", id);
31          return (Obra) consulta.getSingleResult();
32      }
33  }
34
```

Figura 26. Clase de ServicioObra.
Elaborado por: Coba Carlos & Gómez Pablo.

Línea 12. Se define el tipo de protocolo de comunicación @Stateless de sesiones.

Línea 15. Se define el tipo de protocolo @PersistenceContext, para la generación de persistencia.

Línea 16. Se define la variable con el nombre em, que es de tipo administrador de entidades, que genera ingresos y consultas.

Línea 22. Se define el protocolo @SuppressWarnigs, que desactiva las advertencias de compilación.

Línea 23. Se inicia el método de recuperación de todos los datos.

Línea 24. Se crear una variable de tipo Query, donde se escribe la consulta a realizar en la base de datos.

Línea 25. Retorna el valor generado de la consulta de la línea anterior.

Línea 28. Se inicia el método definiendo el nombre buscarObraPorId, que es de la clase Obra y que recibe un parámetro de entrada.

Línea 30. Se crea una variable de tipo Query, donde se escribe la consulta a realizar en la base de datos.

Línea 31. Se define la variable consulta, donde recibe el set de un parámetro, que identifica el id con el dato del parámetro de entrada.

Línea 32. Retorno el tipo Obra y que devuelve el valor de la consulta generada.

Clase Controlador Casa - Variable

Esta clase hace uso de la persistencia y la función del EJB (Enterprise JavaBeans) y el @ManagedBean del PrimeFaces. Así, permite hacer referencia a la clase servicio y la conexión con las páginas del xhtml.

```
19 @ManagedBean
20 @ViewScoped
21 public class ControladorCasa {
22
23     private Casa entidadCasa;
24     private List<Casa>casas;
25     private Casa seleccionarCasa;
26     private Obra obra;
27     private List<Obra>obras;
28     private int idCasa;
29
30     @EJB
31     private ServicioCasa servicioCasa;
32     @EJB
33     private ServicioObra servicioObra;
```

Figura 27. Variables de la clase ControladorObra.
Elaborado por: Coba Carlos & Gómez Pablo.

Línea 19. Se define al gestionador de Bean con el @ManagedBean.

Línea 20. Se define el tipo de sesión a usar con el @ViewScoped.

Línea 23. Se define la variable con el nombre entidadCasa, que es de tipo Casa y permite el manejo de las entidades.

Línea 24. Se define la variable con el nombre casas, que es una lista de tipo Casa y que permite almacenar las entidades de la clase en la memoria.

Línea 25. Se define la variable con el nombre seleccionarCasa, que es de tipo Casa.

Línea 26. Se define la variable con el nombre obra, que es de tipo Obra y permite el manejo de las entidades.

Línea 27. Se define la variable con el nombre obras, que es una lista de tipo Obra y que permite almacenar las entidades de la clase en la memoria

Línea 28. Se define el nombre de la variable idCasa, que es de tipo entero.

Línea 30, 32. Se define el ámbito de interfaz remota de las clases servicios

Línea 31. Se define la variable con el nombre servicioCasa, que es de tipo ServicioCasa, para utilizar los métodos de la clase servicio.

Línea 33. Se define la variable con el nombre servicioObra, que es de tipo ServicioObra, para utilizar los métodos de la clase servicio.

Nota. Al Finalizar la creación de las variables, generar los get y set.

Clase controlador Casa – Constructor.

El constructor inicializa las variables que se crearon, haciendo referencia a las clases que se encuentran dentro del proyecto.

```
35    public ControladorCasa() {
36        entidadCasa = new Casa();
37        seleccionarCasa = new Casa();
38        obra = new Obra();
39        casas = new ArrayList<Casa>();
40        obras = new ArrayList<Obra>();
41    }
```

Figura 28. Fragmento de método de impresión de la Clase Conexión.
Elaborado por: Coba Carlos & Gómez Pablo.

Línea 35. Se inicia el constructor de la clase ControladorCasa.

Línea 36 - 39. Se inicializa las variables de la clase, con sus tipos de datos.

Clase controlador Casa - Método de recuperación

El método usa el PostConstruct de la persistencia y de la clase servicio con la conexión del @EJB y de los @Statless de las clases servicios.

```
43    @PostConstruct
44    public void recuperarTodo(){
45        casas = servicioCasa.recuperarTodo();
46        obras = servicioObra.recuperarTodo();
47    }
```

Figura 29. Método para obtener los datos.
Elaborado por: Coba Carlos & Gómez Pablo.

Línea 43. Se define el llamado de los constructores de la clase ServicioCasa.

Línea 44. Se inicia el método de recuperación de información.

Línea 45. Se asigna el valor devuelto del método recuperarTodo, por medio de la entidad servicioCasa, que hace el llamado a la Clase ServicioCasa y se almacena en la variable de tipo lista casas.

Línea 46. Se asigna el valor devuelto del método recuperarTodo, por medio de la entidad servicioObra, que hace el llamado a la Clase ServicioObra y se almacena en la variable de tipo lista obras.

Clase Controlador Obra – Variables

```
19 @ManagedBean
20 @ViewScoped
21 public class ControladorObra {
22
23     private Casa entidadCasa;
24     private List<Casa>casas;
25     private int idCasa;
26     private Obra entidadObra;
27     private List<Obra>obras;
28     private Obra seleccionarObra;
29
30     @EJB
31     private ServicioObra servicioObra;
32     @EJB
33     private ServicioCasa servicioCasa;
34     //get y set
```

Figura 30. Variables de la clase ControladorObra.
Elaborado por: Coba Carlos & Gómez Pablo.

Línea 19. Se define al gestionador de Bean con el @ManagedBean.

Línea 20. Se define el tipo de sesión a usar con el @ViewScoped.

Línea 23. Se define la variable con el nombre entidadCasa, que es de tipo Casa y permite el manejo de las entidades.

Línea 24. Se define la variable con el nombre casas, que es una lista de tipo Casa y que permite almacenar las entidades de la clase en la memoria.

Línea 25. Se define el nombre de la variable idCasa, que es de tipo entero.

Línea 26. Se define la variable con el nombre entidadObra, que es de tipo Obra y permite el manejo de las entidades.

Línea 27. Se define el nombre de la variable obras, que es una lista de tipo Obra.

Línea 28. Se define la variable con el nombre seleccionarObra, que es de tipo Obra.

Línea 27. Se define la variable con el nombre obras, que es una lista de tipo Obra y que permite almacenar las entidades de la clase en la memoria.

Línea 30, 32. Se define el ámbito de interfaz remota de las clases servicios.

Línea 31. Se define la variable con el nombre servicioObra, que es de tipo ServicioObra, para utilizar los métodos de la clase servicio.

Línea 33. Se define la variable con el nombre servicioCasa, que es de tipo ServicioCasa, para utilizar los métodos de la clase servicio.

Nota. Al Finalizar la creación de las variables, generar los get y set de cada una de ellas.

Clase controlador Obra – Constructor

```
35
36      public ControladorObra() {
37          entidadObra = new Obra();
38          entidadCasa = new Casa();
39          casas = new ArrayList<Casa>();
40          obras = new ArrayList<Obra>();
41      }
```

Figura 31. Controlador de la clase ControladorObra.
Elaborado por: Coba Carlos & Gómez Pablo.

Línea 35. Se inicia el constructor de la clase ControladorCasa.

Línea 36 - 39. Se inicializa las variables de la clase, con sus tipos de datos.

Clase controlador Obra - Método de recuperación

```
43      @PostConstruct
44      public void recuperarTodo(){
45          obras = servicioObra.recuperarTodo();
46      }
47
```

Figura 32. Método para obtener los datos.
Elaborado por: Coba Carlos & Gómez Pablo.

Línea 43. Se define el llamado de los constructores de la clase ServicioCasa.

Línea 44. Se inicia el método de recuperación de información.

Línea 45. Se asigna el valor devuelto del método recuperarTodo, por medio de la entidad servicioObra, que hace el llamado a la Clase ServicioObra y se almacena en la variable de tipo lista obras.

Clase Puntos – Variables

En esta clase se definirán los diferentes tipos de variables, que permitirán realizar las consultas a la base de MongoDB, extrayendo los documentos BSON y de PostgreSQL devolviendo el parámetro de entrada, para que de esta manera se pueda visualizar el punto en Google Map, con las variables de tipo simpleModel y Marker.

```
32 @ManagedBean
33 public class Puntos {
34     /**
35      * @param args
36      */
37
38     private Conexion con = new Conexion();
39     private DBCursor cursor;
40     private List<DBObject> trg;
41     private String impresion;
42     private String punto;
43     private String longitud;
44     private String latitud;
45     private MapModel simpleModel;
46     private Marker marker;
47     private String lugar;
48     private String title;
49     private String informacion;
50     private Obra namObra;
51     private String obraSeleccionada;
52     //private int idObraSeleccionada;
53     private String nombreObraSinBlanco;
```

Figura 33. Variable de la clase Puntos.
Elaborado por: Coba Carlos & Gómez Pablo.

Línea 32. Se define el gestionador de Bean con el @ManagedBean.

Línea 38. Se define una variable con el nombre (con), que es de tipo Conexión y este es de la clase conexión y se la inicializa.

Línea 39. Se define una variable con el nombre cursor, que es de tipo DBCursos,

Línea 40. Se define una variable con el nombre trg, que es una lista de tipo DBObject.

Línea 41. Se define una variable con el nombre impresión, que es de tipo String.

Línea 42. Se define una variable con el nombre punto, que es de tipo String.

Línea 43. Se define una variable con el nombre longitud, que de tipo String.

Línea 44. Se define una variable con el nombre latitud, que es de tipo String.

Línea 45. Se define una variable con el nombre simpleModel, que es de tipo MapModel, este permite graficar en el mapa los puntos obtenidos.

Línea 46. Se define una variable con el nombre marker, que es de tipo Marker, con el que permite graficar una marca sobre las coordenas del simpleModel.

Línea 47. Se define una variable con el nombre lugar, que es de tipo String.

Línea 48. Se define una variable con el nombre title, que es de tipo String.

Línea 49. Se define una variable con el nombre información, que es de tipo String.

Línea 50. Se define una variable con el nombre namObra, que es de tipo Obra.

Línea 51. Se define una variable con el nombre obraSeleccionada, que es de tipo String.

Línea 53. Se define una variable con el nombre nombreObraSinBlancos, que es de tipo String.

Clase Puntos – Constructor

En el constructor se inicializan las variables que pertenece a LatLng y marker, que son de la librería de PrimeFaces y se utiliza en la función gmap.

```
174    public Puntos() {
175        //namObra = new Obra();
176        //System.out.println("NAMIBRA   ..." + namObra);
177        // mapaPunto(namObra);
178        LatLng coordmar = new LatLng(0, 0);
179        marker = new Marker(coordmar);
180    }
```

Figura 34. Constructor de la clase Puntos.
Elaborado por: Coba Carlos & Gómez Pablo.

Línea 174. Se inicia el constructor de la clase Puntos.

Línea 178. Se inicializa la variable LatLng con el nombre coordmar.

Línea 179. Se inicializa la variable marker y se asigna a la variable coordmar.

Clase Puntos – Método de visualización de puntos

El método de visualización de puntos, recibe un documento y por medio de las variables, esta ira desfragmentando al documento en formato JSON y GeoJSON con el parámetro de entrada. Así, dividiendo los puntos necesarios, para luego almacenarlos en la variable simpleModel y de esta forma graficar en la extensión del gmap.

```
186    public void mapaPunto(String nombreObra) {
187        System.out.println("CODIGO DE LA OBRA ------------------------------------
188        simpleModel = new DefaultMapModel();
189        System.out.println("CODIGO DE LA OBRA ------------------------------------
190        cursor = con.consultarCollecionPorNam(nombreObra.trim());
191        System.out.println("CURSOR ---------------> "+cursor);
192        while (cursor.hasNext()) {  //No está entrando al While... xq?
193            impresion = cursor.next().toString();
194            System.out.println("IMPRESION :" + impresion);
195            String loc = impresion.toString();
196            System.out.println("LOC :" + loc);
197            punto = loc.substring(loc.indexOf("\"coordinates\"") + 18,
198                    loc.indexOf("]}}"));
199            System.out.println("punto :" + punto);
200            longitud = punto.substring(0, punto.indexOf(" , "));
201            latitud = punto.substring(punto.indexOf(" , ") + 3, punto.length());
202            Double lon = Double.valueOf(longitud).doubleValue();
203            Double lat = Double.valueOf(latitud).doubleValue();
204            LatLng coord1 = new LatLng(lat, lon);
205            simpleModel.addOverlay(new Marker(coord1, nombreObra.trim(),
206                    "imagenes/comunidadPaute.jpg"));
207        }
208    }
```

Figura 35. Método de consulta del documento y visualización.
Elaborado por: Coba Carlos & Gómez Pablo.

Línea 186. Se inicia el método de visualización que recibe un parámetro de tipo String, con el nombre nombreObra.

Línea 188. Se inicializa la variable simpleModel, con su tipo de dato.

Línea 190. Se asigna en la variable cursor, el resultado obtenido por la variable (con), que hace el llamado al método consultarColleccion por Nam de la clase Conexión, donde recibe el parámetro del método y con la función trim, elimina los espacios en blanco.

Línea 192. Se inicia la función While, donde se condicionar que la variable cursor vaya recorriendo documento por documento.

Línea 193. Se asigna el resultado de la consulta en la variable impresión, cambiando el dato obtenido de un formato de documento, a un tipo String con el toString.

Línea 195. Se define la variable con el nombre (loc), que es de tipo String, donde se almacenara el valor obtenido de la variable impresión.

Línea 197. Se agrega los datos obtenidos de la variable (loc) a la variable punto, al que se realiza un recorrido con el substring indicando cual es el inicio, con la palabra reservada indexof del documento de MongoDB, hasta la finalización del recorrido del documento de la base.

Línea 200. Se agrega el dato en la variable longitud, que es almacenada en la variable punto con el recorrido de la función substring, hasta delimitar la separación dentro del documento obtenido y modificado.

Línea 201. Se agrega el dato en la variable latitud, que es almacenada en la variable punto con el recorrido de la función substring, hasta delimitar la separación por medio de una coma hasta la finalización con la palabra length, que define el tamaño completo del documentó.

Línea 202. Se define una variable con el nombre (lon), que es de tipo Double, con el que se convierte a la variable longitud de tipo String a Double.

Línea 203. Se define una variable con el nombre (lat), que es de tipo Double, con el que se convierte a la variable latitud de tipo String a Double.

Línea 204. Se define una variable con el nombre coord1, que es de tipo LatLng, con el que se inicializa y se agrega los valores de lon y lat para graficar en el mapa.

Línea 205. Se define la variable simpleModel, donde la función Mark es agregada e inicializada con los puntos, el nombre y la imagen a ser demostrada en el visualizador web.

Clase Puntos – Método de descarga en formato GeoJson

Este método permitirá descargar el documento de la consulta en formato GeoJSON, desde MongoDB y poder utilizarlo en cualquier otro programa.

```
226    public void guardarArchivo() {
227        try {
228            FileWriter fw = new FileWriter("D:\\fichero1.geojson");
229            fw.write(impresion);
230            fw.close();
231        } catch (IOException e) {
232            System.out.println("Error E/S: " + e);
233        }
234    }
```

Figura 36. Método para guardar el documento en un archivo GeoJSON.
Elaborado por: Coba Carlos & Gómez Pablo.

Línea 226. Se inicia el método guardarArchivo.

Línea 227. Se inicia el bloque del manejo de excepción Try/Catch.

Línea 228. Se inicializa la variable fw con la función FileWrite y definiendo la localización del almacenamiento.

Línea 229. Se agrega la variable impresión para que se ejecute la variable fw.

Clase MultipoligonoClass – Entidades

```
22 @ManagedBean
23 @ViewScoped
24 public class MultipoligonoClass {
25
26     private Conexion con = new Conexion();
27     private DBCursor cursor;
28     private DBCursor cursorTotal;
29     private List<DBObject> trg;
30     private String impresion;
31     private String impresion1;
32     private String EstMultipoligono;
33     private List<String> listaPoligonos = new ArrayList<String>();
34     private String EstPoligono;
35     private String estPoligonoTemporal;
36     private String longitud;
37     private String latitud;
38     private String coorGlob;
39     private List<String> coordenadasList = new ArrayList<String>();
40     private List<LatLng> coord1 = new ArrayList<LatLng>();
41     private double lat;
42     private double lng;
43     private LatLng coord;
44     private MapModel polygonModel;
45     private String obraSeleccionada;
46     Polygon polygon = new Polygon();
47
48     //set y get
```

Figura 37. Entidad de la clase MultipoligonoClass
Elaborado por: Coba Carlos & Gómez Pablo.

Línea 22. Se define al gestionador de Bean con el @ManagedBean.

Línea 23. Se define el tipo de sesión Bean con el @ViewScoped.

Línea 26. Se define la variable con el nombre (con) que es de tipo conexión y se inicializa.

Línea 27. Se define la variable con el nombre cursor, que es de tipo DBCursos.

Línea 28. Se define la variable con el nombre cursorTotal, que es de tipo DBCursos.

Línea 29. Se define la variable con el nombre trg, que es de tipo lista de tipo DBObject.

Línea 30. Se define la variable con el nombre impresión, que es de tipo String.

Línea 31. Se define la variable con el nombre impresión1, que es de tipo String.

Línea 32. Se define la variable con el nombre EstMultipoligono, que es de tipo String.

Línea 33. Se define la variable con el nombre listaPoligonosCreamos, que es de lista de tipo String y se lo inicializa.

Línea 34. Se define la variable con el nombre EstPoligono, que es de tipo String.

Línea 35. Se define la variable con el nombre estPoligonoTemporal, que es de tipo String.

Línea 35. Se define la variable con el nombre simpleModel, este permitirá graficar en el mapa los puntos obtenidos y pertenece a la librería que es de tipo MapModel.

Línea 46. Se define la variable con el nombre de (marker), este permitirá graficar la marca sobre las coordenadas del simpleModel y es de tipo Marker.

Línea 48. Se generan los Set y Get de las variables creadas.

Clase MultipoligonoClass – Método de impresión de multipolígonos

Este método recibe por medio de la consulta el documento completo de la base MongoDB con el cual se va a desfragmentar al documento por estructuras, iniciando por la estructura del multipolígono, y guardando en una lista temporal y luego agruparlos en una estructura de polígonos.

Para que de este modo se pueda ir diferenciando entre cada estructura y como está se va descomponiendo en cada método hasta llegar a los puntos requeridos para la graficación y visualización en el geoportal.

```
232     //METODO DE IMPRESION DEL MULTIPOLIGONO
233     public void mapaMultiPoligono(String namObra) {
234         // TODO Auto-generated method stub
235         cursor = con.consultarCollecion(namObra.trim());
236         impresion = cursor.next().toString();
237         polygonModel = new DefaultMapModel();
238         System.out.println("consulta de multipoligono desde base :" + impresion);
239 //      System.out.println("*****************************************************************************************
240         String loc = impresion.toString();
241         EstMultipoligono = loc.substring(loc.indexOf("[ [ [ "), loc.indexOf("}}]}"));
242 //          System.out.println("estructura del multipoligono :" + EstMultipoligono);
243 //          System.out.println("-----------------------------------------------------------------------------------
244         EstPoligono = EstMultipoligono.substring(2, EstMultipoligono.length() - 1);
245 //          System.out.println("estructura del poligono :" + EstPoligono);
246 //          System.out.println("+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
247 //          int cont = 0;
248
249
250         // while de los poligonos
251         while (EstPoligono.indexOf("[ [ [ ") != -1) {
252 //              cont++;
253             estPoligonoTemporal = EstPoligono.substring(6,EstPoligono.indexOf("]]]"));
254 //              System.out.println("estructura del poligono temporal :" + estPoligonoTemporal);
255 //              System.out.println("////////////////////////////////////////////////////////////////////////////////
256             listaPoligonos.add(estPoligonoTemporal);
257             EstPoligono = EstPoligono.substring(EstPoligono.indexOf("]]] , [ [ [ ") + 6,EstPoligono.length());
258         }
259 //          System.out.println("&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&
260         listaPoligonos.add(estPoligonoTemporal);
261
262         for (String poligonoLista : listaPoligonos) {
263             //LLAMADA DEL METODO PARA DIVIDIR EN POLIGONOSOS
264             mapaPoligono(poligonoLista);
265         }
266     }
```

Figura 38. Método para graficar un multipolígono.
Elaborado por: Coba Carlos & Gómez Pablo.

Línea 233. Se inicia el método mapaMultiPoligono que recibe como parámetro el dato a consultar.

Línea 235. En la variable cursor se asignara el valor devuelto por la clase conexión por medio de la variable (con), recibiendo el parámetro namObra y con trim se eliminan los espacios en blanco.

Línea 236. En la variable impresión se almacena lo que se tiene en la variable cursor y usando el next y el toString lo convierte a un formato String.

Línea 237. Se inicializa la función pygonModel, de la librería PrimeFaces.

Línea 240. Se define una variable con el nombre loc, que es tipo String, donde se agrega lo almacenado en la función impresión y se almacena solo con la estructura del subdocumento de coorditanes de este modo se tiene la estructura del multipolígono.

Línea 241. En la variable EstMultipoligono se asignara el multipoligono devuelto por el documento de MongoDB, al realizar el subString este permitirá hacer un barrido sobre el documento, indicando el inicio del barrido con el indexof y de igual manera su final.

Línea 244. En la variable EstPoligono, se asignara lo de la variable EstMultipoligono y con la función subString se inicia el barrido, señalando a la posición 2 como inicio y su final con la palabra length.

Línea 251. Se inicia la función while con la condición, que si la variable EstPoligono inicia con tres corchetes y que este seda diferente a -1.

Línea 253. En la variable estPoligonoTemporal, se asignará de forma temporal la estructura del polígono, y usando la variable EstPoligono, con la función subString se iniciara un nuevo barrido de información, indicando el inicio en la posición 6 hasta el cierre de tres corchetes.

Línea 256. En la variable listaPoligono, se asigna la variable de la estructura temporal.

Línea 257. Con la variable EstPoligono se iniciará a dividir la estructura del multipoligono en polígonos, teniendo en cuenta la separación de la estructura por el uso de "]]] , [[[".

Línea 258. Se cierra la función de condicionamiento while.

Línea 260. En la variable lista lsitaPoligonos, se asignara lo de la variable estPoligonoTemporal.

Línea 262. Se inicia la función for each, definiendo que la variable de nombre poligonoLista, es de tipo String y que recorrerá a la lista de listaPoligono.

Línea 264. Se llama al método mapaPoligono y se define la variable creada en el for each como parámetro de entrada.

Línea 265. Se finaliza el for each.

Clase MultipoligonoClass – Método mapaPolígono

El método mapaPoligono recibe la estructura del polígono ya desfragmentado anteriormente, así lo que se hace es dividir el polígono en una lista temporal, dividiendo los puntos, para después reagruparlos y limpiar la lista del polígono.

El método en si maneja ya un fragmento del sub documento que contiene los vectores, es decir que en este método ya se ha desfragmentado la estructura inicial, y la estructura del multipolígono, dejando así una estructura sencilla.

Que facilitara el manejo de esta información, para su siguiente desfragmentación la que divide el polígono a puntos.

```
270     //METODO PARA DIVIDIR EL MULTIPOLIGONO EN POLIGNOS
271     public void mapaPoligono(String poligonoLista) {
272
273 //      System.out.println("lista de poligono agregados :" + poligonoLista);
274         while (poligonoLista.indexOf("] , [") != -1) {
275             coorGlob = poligonoLista.substring(0,poligonoLista.indexOf("] , ["));
276             poligonoLista = poligonoLista.substring(poligonoLista.indexOf("] , [") + 6, poligonoLista.length());
277             coordenadasList.add(coorGlob);
278 //          System.out.println("coordenadas globales :" + coorGlob);
279         }
280         coordenadasList.add(coorGlob);
281 //      System.out.println("lista de coordenadas :" + coordenadasList);
282 //      System.out.println("##################################################################################");
283         for (String cl : coordenadasList) {
284 //          System.out.println(cl);
285             //LLAMADA DEL METODO PARA DIVIDIR EN PUNTOS
286             mapaPunto(cl);
287         }
288         formato();
289         coordenadasList = new ArrayList<String>();
290     }
```

Figura 39. Método visualizar el método mapaPunto.
Elaborado por: Coba Carlos & Gómez Pablo.

Línea 271. Se inicia el método mapaPoligono que recibe como parámetro de entrada el poligonoLista, que es de tipo String.

Línea 274. Se inicia la función de condicionamiento while, en que la condición poligonLista tenga como parámetro "] , [" y que sea diferente de -1.

Línea 275. En la variable coorGlob se asignará lo de poligonolista y con el subString indicando la posición 0 para iniciar el barrido, hasta encontrar "] , [".

Línea 276. Con la variable poligonoLista, se inicia un nuevo barrido a poligonoLista con subString indicando el inicio con "] , [", hasta que se cumpla la función length.

Línea 277. En la variable coordenadaList, se agregará, lo que se generó en la variable coordGlob.

Línea 279. Se cierra la función de comparación while.

Línea 280. En la variable coordenadaLista se añade la variable coordGlob, después de finalizar la función while.

Línea 283. Se crea la función de tipo for each, definiendo la variable cl, que es de tipo String y que vaya recorriendo la lista de coordenadaList.

Línea 286. Se llama al método mapaPunto y se define a la variable cl del for each como parámetro de entrada.

Línea 287. Se cierra la función for each.

Línea 288. Se llama al método formato.

Línea 289. Se inicializa y se limpia la lista de coordenadaLista.

Clase MultipoligonoClass – Método de división de polígonos a puntos

El método mapaPunto, obtiene los datos generados por lista del anterior método, de esta forma divide los puntos y los va agregando en las variables latitud y longitud, para después unir y almacenar en la variable coord. Así, con la función polygon poder graficar en el gmap de la página xhtml.

```
292     //METODO PARA DIVIDIR LOS POLINOGONOS EN PUNTOS
293     public void mapaPunto(String cl) {
294         longitud = cl.substring(0, cl.indexOf(" , "));
295 //      System.out.println("longitud :" + longitud);
296         latitud = cl.substring(cl.indexOf(" , ") + 3, cl.length());
297 //      System.out.println("latitud :" + latitud);
298 //      System.out.println("_____");
299         lat = Double.valueOf(latitud).doubleValue();
300         lng = Double.valueOf(longitud).doubleValue();
301         coord = new LatLng(lat, lng);
302         polygon.getPaths().add(coord);
303     }
```

Figura 40. Método de división de polígonos a puntos.
Elaborado por: Coba Carlos & Gómez Pablo.

Línea 293. Se inicia el método mapaPunto, que recibe un parámetro con el nombre cl que es de tipo String.

Línea 294. En la variable longitud, se realizará un barrido con subString, indicando la posición 0 y su finalización al usar una coma de este modo se dividirá la estructura del polígono en puntos de longitud para ser graficados en el mapa.

Línea 296. En la variable latitud, se realizará un barrido con subString, indicando la posición inicial al usar una coma y su finalización con la función length, de este modo se dividirá la estructura del polígono en puntos de latitud para ser graficados en el mapa.

Línea 299. En la variable lat, se agregan los puntos de la variable latitud.

Línea 300. En la variable lng, se agregan los puntos de la variable longitud.

Línea 301. Se inicializara la variable coord, que recibe como parámetro a la función LatLng que permite crear los puntos en el visualizador.

Línea 302. En la variable polygon, se agrega un path en el que se añade los datos almacenados en la variable coord.

Clase MultipoligonoClass – Método formato

En este método se pones lo parámetros o el formato deseado a ser representado en el gmap, del visualizador.

```
305     //METODO PARA GRAFICAR EN EL MAPRA
306     public void formato() {
307         polygon.setStrokeColor("MAGENTA");
308         polygon.setFillColor("MAGENTA");
309         polygon.setStrokeOpacity(0.7);
310         polygon.setFillOpacity(0.7);
311         polygonModel.addOverlay(polygon);
312         polygon = new Polygon();
313     }
```

Figura 41. Método para definir el formato de visualización.
Elaborado por: Coba Carlos & Gómez Pablo

Línea 306. Se inicia el método formato.

Línea 307. Con el setStrokeColor, se define el color del polígono.

Línea 308. Con el setFillColor, se define el color del filo del polígono.

Línea 309. Con el setStrokeOpacity, se define el nivel de sombra del polígono.

Línea 310. Con el setFillOpacida, se define el nivel de sombra del filo del polígono.

Línea 311. Se define a la variable polygon, como parámetro de entrada a polygonModel.addOverlay, permitiendo así graficar en el visualizador.

Línea 312. Se inicializa la variable polygon, para que cada polígono graficado se independiente del multipolígono.

3.1.2 Integración del producto.

El desarrollo del trabajo de titulación consta de diferentes ámbitos, tanto en el desarrollo como en el manejo de la información desde la base de datos.

En el desarrollo del proyecto, está separado el sistema en las siguientes capas:

- Capa de datos.
- Capa de negocios.
- Capa de presentación.

Capa de datos

En el trabajo de titulación se usan dos tipos de base de datos, una de tipo relacional como es el caso de PostgreSQL y la otra de tipo NoSQL como es MongoDB.

En la base de datos PostgreSQL se tiene dos tablas, en la que se almacenará la información de una casa y de una obra, estas tienen relación entre sí, el motivo para

utilizar esta base es demostrar en nuestro visualizador la carga de la información en un selectOneMenu de PrimeFaces.

Capa de negocios

Esta capa brindará el manejo de entidades y persistencias dentro de la clase entidad, servicio, controlador y después capturar la información consultada y realizar una nueva consulta en la base de datos en MongoDB.

En la base de MongoDB se almacena todo el diccionario de los datos recolectados, así como su información geográfica, la que al ser capturada en la consulta de PostgreSQL, direccionara la variable adquirida a MongoDB, de esta manera con el uso de las clases, Puntos y MultipoligonoClass se podrá consultar su parte geográfica a MongoDB y representar en la interfaz de nuestro visualizador web.

Capa de presentación

Esta implementación se la realizó con éxito, al integrar la base de datos PostgreSQL, la base NoSQL MongoDB, su capa de negocio con JSF y su capa de interfaz con PrimeFaces.

3.1.3 Diagrama de implementación.

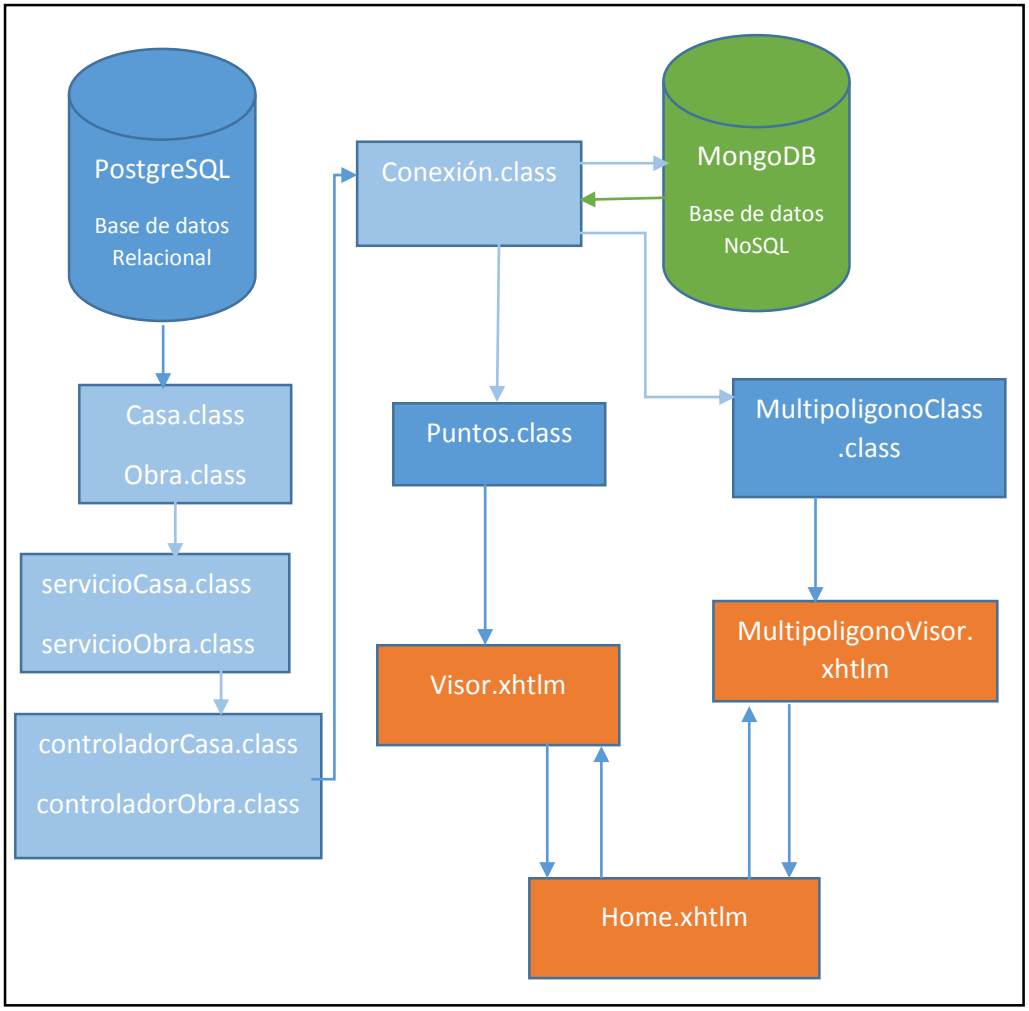


Figura 42. Modelo de implementación entra Bases y Clases.
Elaborado por: Coba Carlos & Gómez Pablo.

Conclusión de la implementación

La integración se la realizó favorablemente, convirtiéndose en el primer trabajo de titulación en realizar un proyecto, con la interacción entre una base de datos SQL y una NoSQL, con un lenguaje de programación de JSF en java y la libre PrimeFaces.

3.1.4 Prueba del producto.

Se realizaron pruebas de estrés a cada una de las páginas web del visualizador, con la herramienta JMeter de apache y que es de código libre, obteniendo los siguientes resultados:

Los valores están calculados, tomando el tiempo total en que se ha tardado la prueba (la ejecución de 50 usuarios), por lo tanto, mientras más muestras se realizan, más amplio es el tiempo de ejecución que influye en cada línea de la tabla. Para cada línea (petición) se tiene:

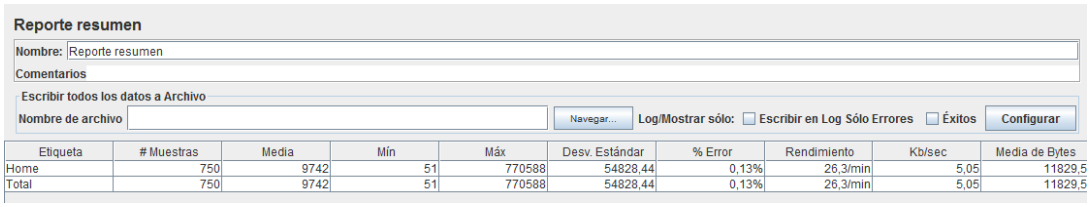
- El máximo de tiempo invertido por una petición (columna Max).
- El mínimo de tiempo invertido por una petición (columna Min).
- La media de tiempo invertido por una petición (columna Media).
- La mediana de tiempo invertido por una petición: significa que el 50% de las muestras tardaron menos del valor reflejado.
- El tanto por ciento de respuestas con error.
- El rendimiento (thoughput): número de peticiones procesadas en una unidad de tiempo, que puede ser segundos, minutos y horas.
- El rendimiento en Kb/segundo: igual que la anterior pero con cantidad de datos en lugar de peticiones.

Home.xhtml

Figura 43. Página inicial del visualizador.
Elaborado por: Coba Carlos & Gómez Pablo.

Reporte resumen

El reporte resumen, se obtuvo haciendo las pruebas a la página de inicio, donde se puede observar que sufrió un error del 0.13% y un rendimiento del 26.3 minutos, por una cantidad de 50 peticiones cada 15 segundos.

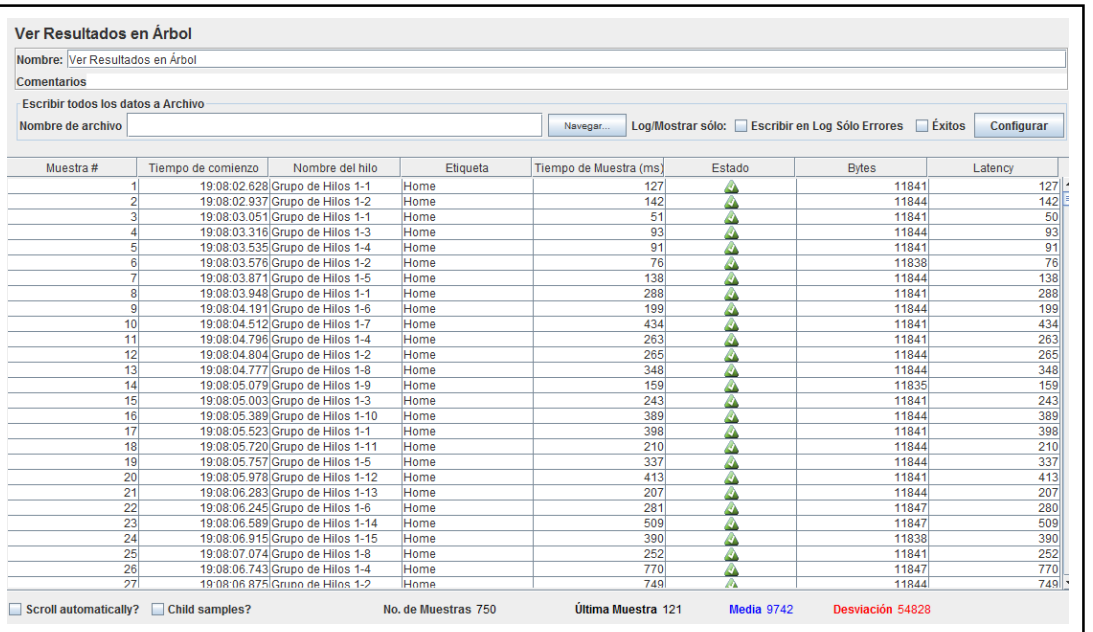


Etiqueta	# Muestras	Media	Mín	Máx	Desv. Estándar	% Error	Rendimiento	Kb/sec	Media de Bytes
Home	750	9742	51	770588	54828,44	0,13%	26,3/min	5,05	11829,5
Total	750	9742	51	770588	54828,44	0,13%	26,3/min	5,05	11829,5

Figura 44. Resultado del JMeter.
Elaborado por: Coba Carlos & Gómez Pablo.

Resultado en árbol

Este resultado muestra el estado de color verde por cada hilo, que simula una petición, donde permite observar que la latencia es aceptable y no sufrió errores en el trascurso de la prueba.



Muestra #	Tiempo de comienzo	Nombre del hilo	Etiqueta	Tiempo de Muestra (ms)	Estado	Bytes	Latency
1	19:08:02.628	Grupo de Hilos 1-1	Home	127		11841	127
2	19:08:02.937	Grupo de Hilos 1-2	Home	142		11844	142
3	19:08:03.051	Grupo de Hilos 1-1	Home	51		11841	50
4	19:08:03.316	Grupo de Hilos 1-3	Home	93		11844	93
5	19:08:03.535	Grupo de Hilos 1-4	Home	91		11841	91
6	19:08:03.576	Grupo de Hilos 1-2	Home	76		11838	76
7	19:08:03.871	Grupo de Hilos 1-5	Home	138		11844	138
8	19:08:03.948	Grupo de Hilos 1-1	Home	288		11841	288
9	19:08:04.191	Grupo de Hilos 1-6	Home	199		11844	199
10	19:08:04.512	Grupo de Hilos 1-7	Home	434		11841	434
11	19:08:04.796	Grupo de Hilos 1-4	Home	263		11841	263
12	19:08:04.804	Grupo de Hilos 1-2	Home	265		11844	265
13	19:08:04.777	Grupo de Hilos 1-8	Home	348		11844	348
14	19:08:05.079	Grupo de Hilos 1-9	Home	159		11835	159
15	19:08:05.003	Grupo de Hilos 1-3	Home	243		11841	243
16	19:08:05.389	Grupo de Hilos 1-10	Home	389		11844	389
17	19:08:05.523	Grupo de Hilos 1-1	Home	398		11841	398
18	19:08:05.720	Grupo de Hilos 1-11	Home	210		11844	210
19	19:08:05.757	Grupo de Hilos 1-5	Home	337		11844	337
20	19:08:05.978	Grupo de Hilos 1-12	Home	413		11841	413
21	19:08:06.283	Grupo de Hilos 1-13	Home	207		11844	207
22	19:08:06.245	Grupo de Hilos 1-6	Home	281		11847	280
23	19:08:06.589	Grupo de Hilos 1-14	Home	509		11847	509
24	19:08:06.915	Grupo de Hilos 1-15	Home	390		11838	390
25	19:08:07.074	Grupo de Hilos 1-8	Home	252		11841	252
26	19:08:06.743	Grupo de Hilos 1-4	Home	770		11847	770
27	19:08:06.875	Grupo de Hilos 1-2	Home	749		11844	749

Figura 45. Resultado del JMeter.
Elaborado por: Coba Carlos & Gómez Pablo.

Árbol de resultados

En este informe se muestran los datos de la manera en que son consumidos en cada petición y que devuelve la estructura de la página en un formato xml.

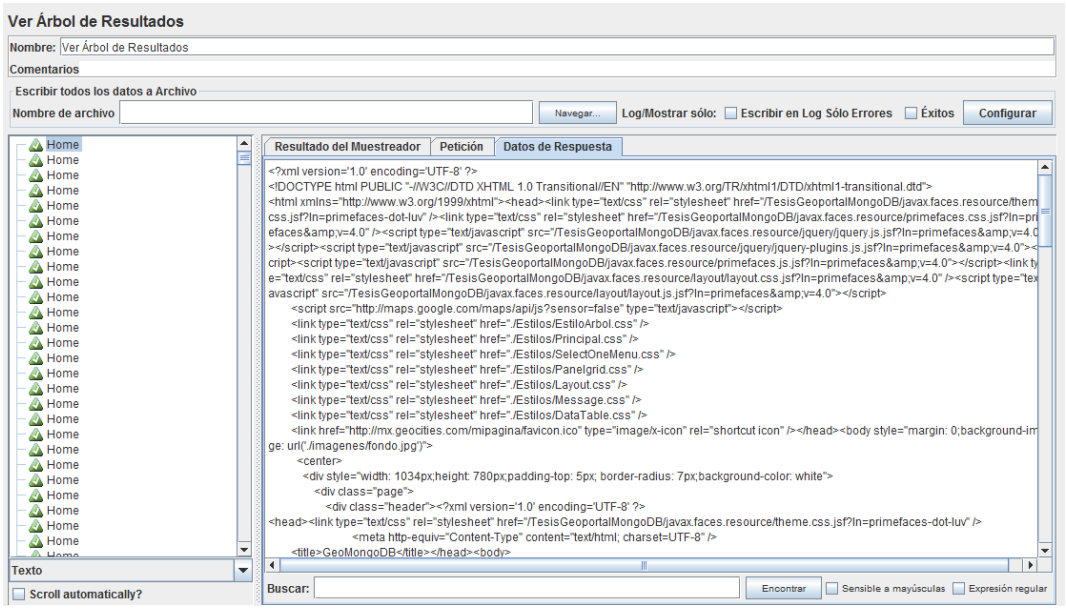


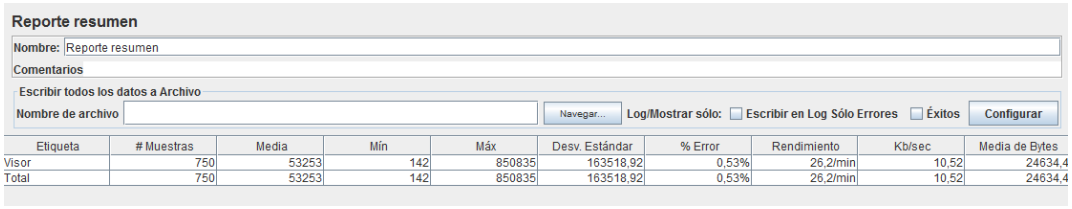
Figura 46. Resultado del JMeter.
Elaborado por: Coba Carlos & Gómez Pablo.

Visor.xhtml

Figura 47. Página de visualización de puntos.
Elaborado por: Coba Carlos & Gómez Pablo.

Reporte resumen

En la página visor, es la que permite hacer consultas y presentar los puntos de la información de la casa y obra salesiana, en donde se muestra que se dio un 0.53% de error y que posee un rendimiento de 26.2 minutos por cada petición, en el trascurso de 142 minutos.

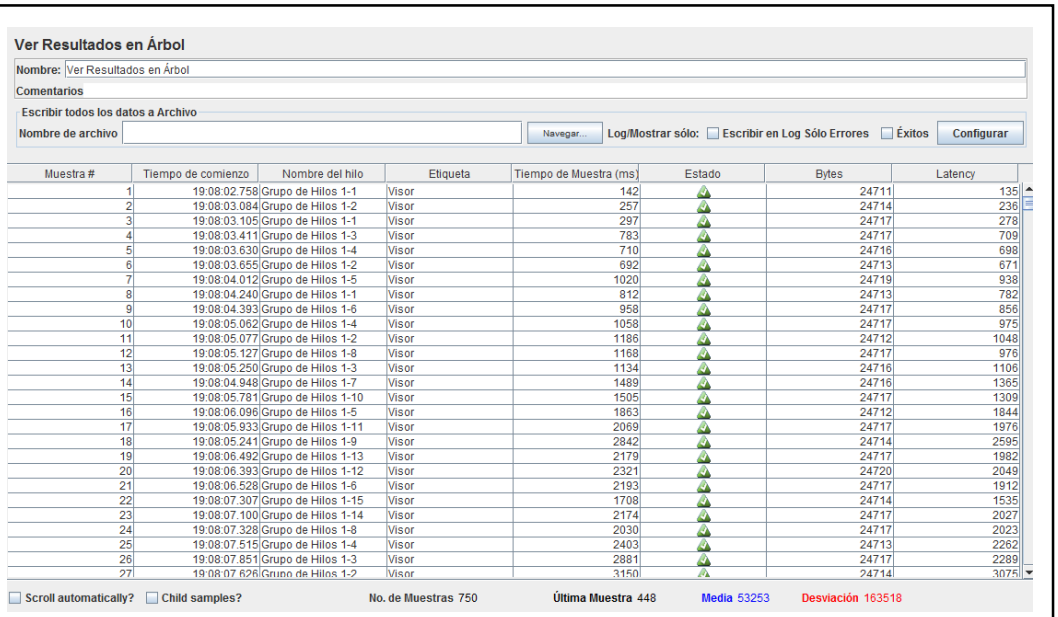


Etiqueta	# Muestras	Media	Mín	Máx	Desv. Estándar	% Error	Rendimiento	Kb/sec	Media de Bytes
Visor	750	53253	142	850835	163518,92	0,53%	26,2/min	10,52	24634,4
Total	750	53253	142	850835	163518,92	0,53%	26,2/min	10,52	24634,4

Figura 48. Resultado del JMeter.
Elaborado por: Coba Carlos & Gómez Pablo.

Resultado en árbol

El estado en la página fue muy aceptable, sufriendo un manejo de la latencia muy variable en cada petición, con un tiempo de muestra en 750 ms y sufriendo una media de 53253 en el consumo de Bytes, por las 50 peticiones.



Muestra #	Tiempo de comienzo	Nombre del hilo	Etiqueta	Tiempo de Muestra (ms)	Estado	Bytes	Latency
1	19:08:02.758	Grupo de Hilos 1-1	Visor	142		24711	135
2	19:08:03.084	Grupo de Hilos 1-2	Visor	257		24714	236
3	19:08:03.105	Grupo de Hilos 1-1	Visor	297		24717	278
4	19:08:03.411	Grupo de Hilos 1-3	Visor	783		24717	709
5	19:08:03.630	Grupo de Hilos 1-4	Visor	710		24716	698
6	19:08:03.655	Grupo de Hilos 1-2	Visor	692		24713	671
7	19:08:04.012	Grupo de Hilos 1-5	Visor	1020		24719	938
8	19:08:04.240	Grupo de Hilos 1-1	Visor	812		24713	782
9	19:08:04.393	Grupo de Hilos 1-6	Visor	958		24717	856
10	19:08:05.062	Grupo de Hilos 1-4	Visor	1058		24717	975
11	19:08:05.077	Grupo de Hilos 1-2	Visor	1186		24712	1048
12	19:08:05.127	Grupo de Hilos 1-8	Visor	1168		24717	976
13	19:08:05.250	Grupo de Hilos 1-3	Visor	1134		24716	1106
14	19:08:04.948	Grupo de Hilos 1-7	Visor	1489		24716	1365
15	19:08:05.781	Grupo de Hilos 1-10	Visor	1505		24717	1309
16	19:08:06.096	Grupo de Hilos 1-5	Visor	1863		24712	1844
17	19:08:05.933	Grupo de Hilos 1-11	Visor	2069		24717	1976
18	19:08:05.241	Grupo de Hilos 1-9	Visor	2842		24714	2595
19	19:08:06.492	Grupo de Hilos 1-13	Visor	2179		24717	1982
20	19:08:06.393	Grupo de Hilos 1-12	Visor	2321		24720	2049
21	19:08:06.528	Grupo de Hilos 1-6	Visor	2193		24717	1912
22	19:08:07.307	Grupo de Hilos 1-15	Visor	1708		24714	1535
23	19:08:07.100	Grupo de Hilos 1-14	Visor	2174		24717	2027
24	19:08:07.328	Grupo de Hilos 1-8	Visor	2030		24717	2023
25	19:08:07.515	Grupo de Hilos 1-4	Visor	2403		24713	2262
26	19:08:07.851	Grupo de Hilos 1-3	Visor	2881		24717	2289
27	19:08:07.626	Grupo de Hilos 1-2	Visor	3150		24714	3075

Figura 49. Resultado del JMeter.
Elaborado por: Coba Carlos & Gómez Pablo.

Árbol de resultados

Se observara que el manejo de las consultas que recibe un parámetro de la base de PostgreSQL, es almacenado para luego hacer la búsqueda en MongoDB. Así, se puede observar que se hace una petición por cada uno de los parámetros llamados y representados en la página de visor.

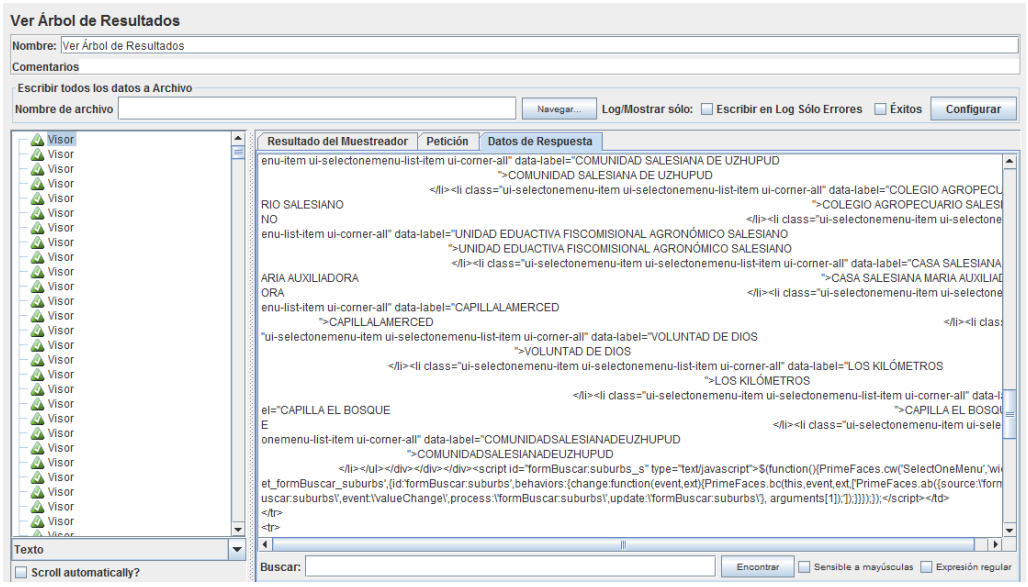


Figura 50. Resultado del JMeter.
Elaborado por: Coba Carlos & Gómez Pablo.

MultipoligonoVisor.xhtml

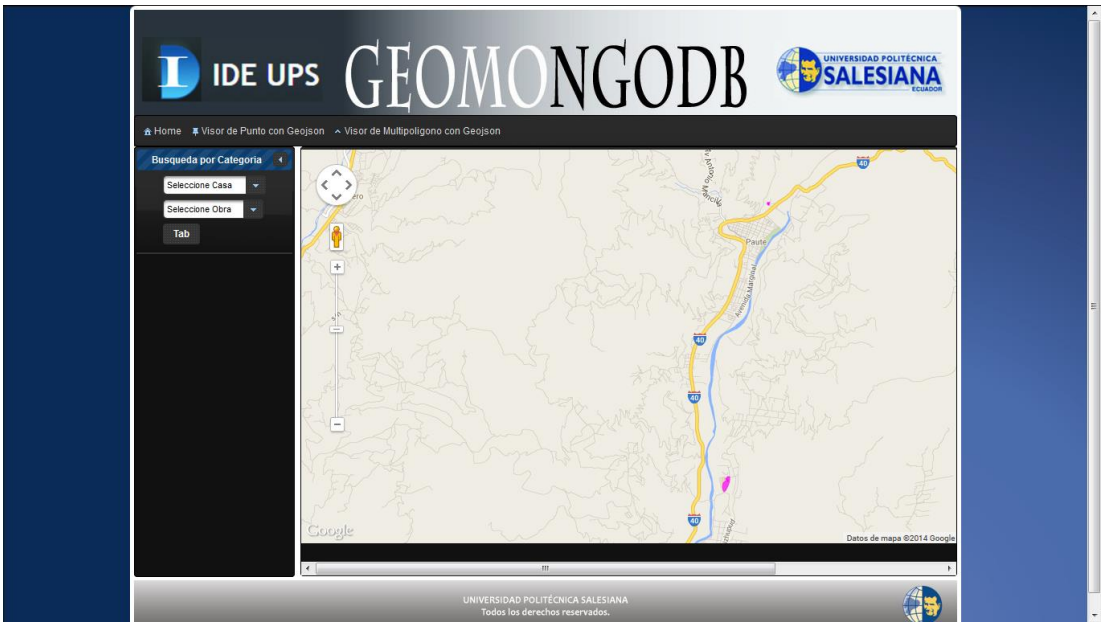
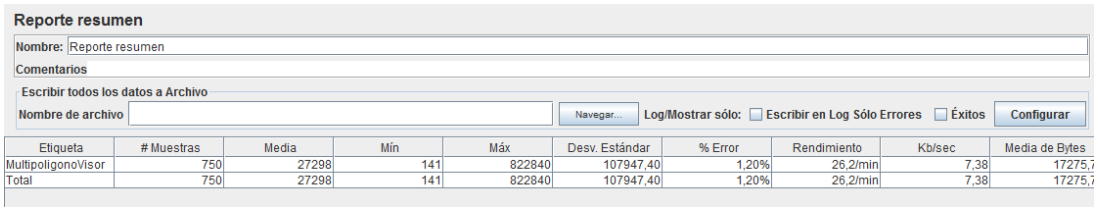


Figura 51. Página de visualización de MultiPolígonos.
Elaborado por: Coba Carlos & Gómez Pablo.

Reporte resumen

En el resultado de este reporte, se dan las pruebas de estrés que se hicieron en la página MulipoligonoVisor, sufriendo 1.20% de error con un rendimiento de 26.2 min en 141 min, y con una velocidad de 7.38 Kb/sec.

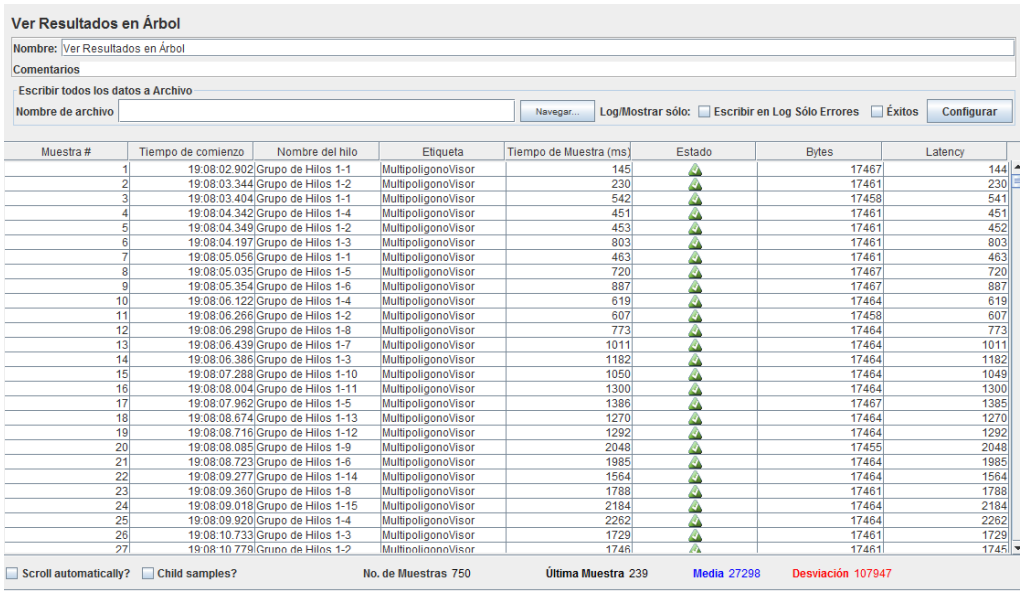


Etiqueta	# Muestras	Media	Mín	Máx	Desv. Estándar	% Error	Rendimiento	Kb/sec	Media de Bytes
MultipoligonoVisor	750	27298	141	822840	107947,40	1,20%	26,2/min	7,38	17275,7
Total	750	27298	141	822840	107947,40	1,20%	26,2/min	7,38	17275,7

Figura 52. Resultado del JMeter.
Elaborado por: Coba Carlos & Gómez Pablo.

Resultado en árbol

Como se observa en el estado, de igual manera es aceptable en las pruebas hechas, con una petición de 50 hilos y con un número de muestras de 750 con una media de 27299 y sufriendo una latencia variable.



Muestra #	Tiempo de comienzo	Nombre del hilo	Etiqueta	Tiempo de Muestra (ms)	Estado	Bytes	Latency
1	19:08:02.902	Grupo de Hilos 1-1	MultipoligonoVisor	145		17467	144
2	19:08:03.344	Grupo de Hilos 1-2	MultipoligonoVisor	230		17461	230
3	19:08:03.404	Grupo de Hilos 1-1	MultipoligonoVisor	542		17458	541
4	19:08:04.342	Grupo de Hilos 1-4	MultipoligonoVisor	451		17461	451
5	19:08:04.349	Grupo de Hilos 1-2	MultipoligonoVisor	453		17461	452
6	19:08:04.197	Grupo de Hilos 1-3	MultipoligonoVisor	803		17461	803
7	19:08:05.056	Grupo de Hilos 1-1	MultipoligonoVisor	463		17461	463
8	19:08:05.035	Grupo de Hilos 1-5	MultipoligonoVisor	720		17467	720
9	19:08:05.354	Grupo de Hilos 1-6	MultipoligonoVisor	887		17467	887
10	19:08:06.122	Grupo de Hilos 1-4	MultipoligonoVisor	619		17464	619
11	19:08:06.266	Grupo de Hilos 1-2	MultipoligonoVisor	607		17458	607
12	19:08:06.298	Grupo de Hilos 1-8	MultipoligonoVisor	773		17464	773
13	19:08:06.439	Grupo de Hilos 1-7	MultipoligonoVisor	1011		17464	1011
14	19:08:06.386	Grupo de Hilos 1-3	MultipoligonoVisor	1182		17464	1182
15	19:08:07.288	Grupo de Hilos 1-10	MultipoligonoVisor	1050		17464	1049
16	19:08:08.004	Grupo de Hilos 1-11	MultipoligonoVisor	1300		17464	1300
17	19:08:07.962	Grupo de Hilos 1-5	MultipoligonoVisor	1386		17467	1385
18	19:08:08.674	Grupo de Hilos 1-13	MultipoligonoVisor	1270		17464	1270
19	19:08:08.716	Grupo de Hilos 1-12	MultipoligonoVisor	1292		17464	1292
20	19:08:08.085	Grupo de Hilos 1-9	MultipoligonoVisor	2048		17455	2048
21	19:08:08.723	Grupo de Hilos 1-6	MultipoligonoVisor	1985		17464	1985
22	19:08:09.277	Grupo de Hilos 1-14	MultipoligonoVisor	1564		17464	1564
23	19:08:09.360	Grupo de Hilos 1-8	MultipoligonoVisor	1788		17461	1788
24	19:08:09.018	Grupo de Hilos 1-15	MultipoligonoVisor	2184		17464	2184
25	19:08:09.920	Grupo de Hilos 1-4	MultipoligonoVisor	2262		17464	2262
26	19:08:10.733	Grupo de Hilos 1-3	MultipoligonoVisor	1729		17461	1729
27	19:08:10.779	Grupo de Hilos 1-2	MultipoligonoVisor	1746		17461	1745

Figura 53. Resultado del JMeter.
Elaborado por: Coba Carlos & Gómez Pablo.

Árbol de resultados

En este informe se puede ver como se está consumido los datos geométricos, desde la base de MongoDB, realizando diferentes peticiones y de este modo como se desfragmenta el documento. Así, se puede fijar que está consumiendo los datos y que el código que se uso va realizando las funciones correctas, para dividir y asignar a las funciones propias para la visualización.

En otras palabras se puede observar la funcionalidad interna de lo que está haciendo la programación, o lo que es oculto para el usuario, ya que este solo mira los resultados y en esta prueba solo se observa el proceso, más no los resultados.

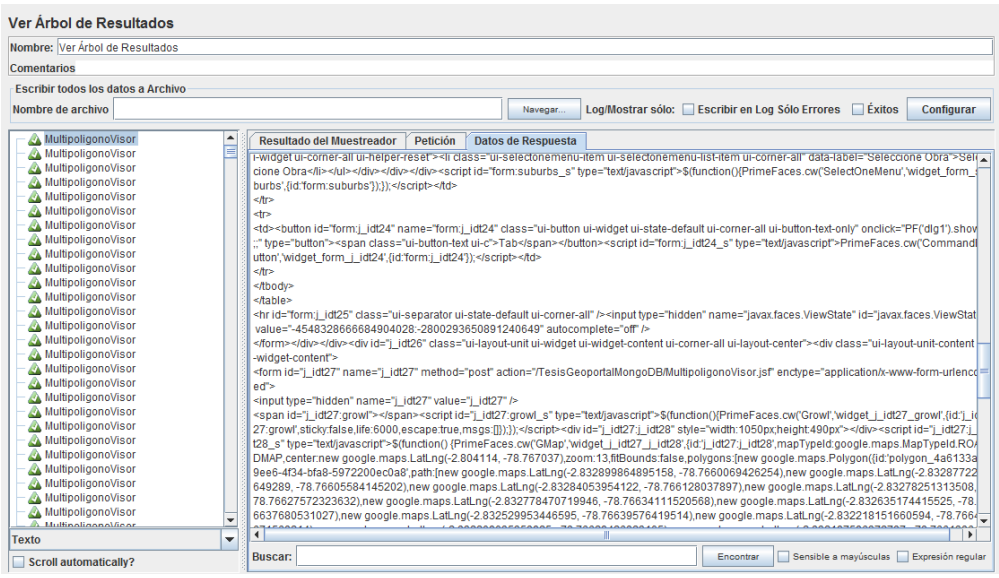


Figura 54. Resultado del JMeter.
Elaborado por: Coba Carlos & Gómez Pablo.

3.2 Fase de entrega

En la fase de entrega, se demostrará por medio de una pila de entrega, el manejo del sprint por cada clase a ser desarrollado para la integración de todas las clases, métodos y datos de la base. Así, también ver los avances y cambios realizados en el trascurso de los 11 meses de desarrollo, análisis e implementación del proyecto.

3.2.1 Entrega del producto.

Tabla 14. Pila de entregables.

Pila de Entrega: GeoMongoDB					**N° de esfuerzo estimado por mes**										
Requerimientos	**Sprint**	**Encargado**	**Apoyo**	**Estimación esfuerzo inicial (Horas)**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**
Gestionar datos de una base de datos NoSQL.	Investigar e instalar MongoDB.	Carlos Coba	Pablo Gómez	44	4	4	4	4	4	4	4	4	4	4	4

Continúa…

Tabla 14. Pila de entregables. (Continuación...)

	GUI de MongoDB.	Carlos Coba	Pablo Gómez	13	2	2	2	2	2	3	3	3	3	2,4	2,4
	Instalar RoboMongo.	Carlos Coba	Pablo Gómez	20	1	1	1	1	1	1	2	2	2	2.5	4
	Ingresar datos.	Carlos Coba	Pablo Gómez	37	0	0	0	0	0	0	5	5	4	5	3
	Consultar datos.	Carlos Coba	Pablo Gómez	27	2	2	2	2	2	3	3	3	3	2,4	2,4
	Manejo de los datos almacenados.	Carlos Coba	Pablo Gómez	20	1	1	1	1	1	1	2	2	2	2.5	4
	Definir consultas para la ejecución.	Carlos Coba	Pablo Gómez	27	0	0	0	0	0	0	5	5	4	5	3
Gestionar datos de una base de datos SQL.	Diagrama de base de datos.	Carlos Coba	Pablo Gómez	48	4	4	4	4	4	4	4	4	4	4	4
	Crear tablas.	Carlos Coba	Pablo Gómez	24	2	2	2	2	2	3	3	3	3	2,4	2,4
	Insertar datos.	Carlos Coba	Pablo Gómez	24	1	1	1	1	1	1	2	2	2	2.5	4
	Almacenar consultas a utilizar.	Carlos Coba	Pablo Gómez	12	0	0	0	0	0	0	5	5	4	5	3
Gestionar la interacción entra base de datos SQL y NoSQL.	Diagrama de caso de uso	Carlos Coba	Pablo Gómez	40	2	2	2	2	2	3	3	3	3	2,4	2,4
	Diseñar clase conexión a MongoDB.	Carlos Coba	Pablo Gómez	20	1	1	1	1	1	1	2	2	2	2.5	4
	Desarrollar clase conexión.	Carlos Coba	Pablo Gómez	45	0	0	0	0	0	0	5	5	4	5	3
	Diseño de proyecto JPA	Carlos Coba	Pablo Gómez	20	4	4	4	4	4	4	4	4	4	4	4
	Desarrollo de JPA	Carlos Coba	Pablo Gómez	22	2	2	2	2	2	3	3	3	3	2,4	2,4
	Copia clases JPA a Proyecto principal.	Carlos Coba	Pablo Gómez	20	1	1	1	1	1	1	2	2	2	2.5	4
	Diseño de paquete de clase entidad	Carlos Coba	Pablo Gómez	20	0	0	0	0	0	0	5	5	4	5	3

Continúa…

Tabla 14. Pila de entregables. (Continuación...)

	Desarrollo paquete clase entidad.	Carlos Coba	Pablo Gómez	45	2	2	2	2	2	3	3	3	3	2,4	2,4
	Diseño paquete clase servicio	Carlos Coba	Pablo Gómez	20	1	1	1	1	1	1	2	2	2	2.5	4
	Desarrollo paquete de clase servicio	Carlos Coba	Pablo Gómez	45	0	0	0	0	0	0	5	5	4	5	3
	Diseño de paquete de Clase control	Carlos Coba	Pablo Gómez	20	4	4	4	4	4	4	4	4	4	4	4
	Desarrollo de paquete de clase control	Carlos Coba	Pablo Gómez	45	2	2	2	2	2	3	3	3	3	2,4	2,4
	Diseño de la clase Punto	Carlos Coba	Pablo Gómez	20	1	1	1	1	1	1	2	2	2	2.5	4
	Desarrollo de la clase Punto	Carlos Coba	Pablo Gómez	45	0	0	0	0	0	0	5	5	4	5	3
	Diseño de la clase MultiPoligono	Carlos Coba	Pablo Gómez	20	0	0	0	0	0	0	5	5	4	5	3
	Desarrollo de la clase MultiPoligono	Carlos Coba	Pablo Gómez	45	2	2	2	2	2	3	3	3	3	2,4	2,4
	Diseño clase consulta	Carlos Coba	Pablo Gómez	20	1	1	1	1	1	1	2	2	2	2.5	4
	Desarrollo clase consulta	Carlos Coba	Pablo Gómez	45	0	0	0	0	0	0	5	5	4	5	3
Consumir datos en formato GeoJSON	Diseño de método de consulta	Carlos Coba	Pablo Gómez	64	4	4	4	4	4	4	4	4	4	4	4
	Modificar documento almacenado en MongoDB	Carlos Coba	Pablo Gómez	64	2	2	2	2	2	3	3	3	3	2,4	2,4
	Definir consultas para el consumo de documentos.	Carlos Coba	Pablo Gómez	64	1	1	1	1	1	1	2	2	2	2.5	4
Visualizar datos geográficos desde una base de datos NoSQL.	Diseño de la interfaz de usuario	Carlos Coba	Pablo Gómez	64	0	0	0	0	0	0	5	5	4	5	3
	Investigar la relación JSF y PrimeFaces con MongoDB	Carlos Coba	Pablo Gómez	61	2	2	2	2	2	3	3	3	3	2,4	2,4

Continúa…

Tabla 14. Pila de entregables. (Continuación...)

	Investigar herramientas y api de Google Map	Carlos Coba	Pablo Gómez	32	1	1	1	1	1	1	2	2	2	2.5	4
	Desarrollar las páginas xhtml	Carlos Coba	Pablo Gómez	45	0	0	0	0	0	0	5	5	4	5	3
	Extraer información desde capa de negocios	Carlos Coba	Pablo Gómez	32	4	4	4	4	4	4	4	4	4	4	4
	Visualizar con gmap de Google Map	Carlos Coba	Pablo Gómez	65	2	2	2	2	2	3	3	3	3	2,4	2,4
Revisar, corregir, integrar y probar cada módulo de gestión.	Revisar si el servicio es automático.	Carlos Coba	Pablo Gómez	25	1	1	1	1	1	1	2	2	2	2.5	4
	Revisar si la relación en eficiente.	Carlos Coba	Pablo Gómez	24	0	0	0	0	0	0	5	5	4	5	3
	Corregir error de relación.	Carlos Coba	Pablo Gómez	34	0	0	0	0	0	0	5	5	4	5	3
	Revisar diagrama de caso de uso.	Carlos Coba	Pablo Gómez	28	4	4	4	4	4	4	4	4	4	4	4
	Revisar clase conexión.	Carlos Coba	Pablo Gómez	23	2	2	2	2	2	3	3	3	3	2,4	2,4
	Revisar código y método de conexión	Carlos Coba	Pablo Gómez	23	1	1	1	1	1	1	2	2	2	2.5	4
	Revisión de JPA.	Carlos Coba	Pablo Gómez	23	0	0	0	0	0	0	5	5	4	5	3
	Revisar paquete de clase entidad.	Carlos Coba	Pablo Gómez	23	0	0	0	0	0	0	5	5	4	5	3
	Revisar paquete de clase servicio	Carlos Coba	Pablo Gómez	23	2	2	2	2	2	3	3	3	3	2,4	2,4
	Revisar paquete de clase control	Coba	Gómez	22	1	1	1	1	1	1	2	2	2	2.5	4
	Revisar método de consulta.	Carlos Coba	Pablo Gómez	23	0	0	0	0	0	0	5	5	4	5	3
	Revisar la compatibilidad entre JSF y MongoDB	Carlos Coba	Pablo Gómez	65	0	0	0	0	0	0	5	5	4	5	3
	Revisar diseño clase Punto	Carlos Coba	Pablo Gómez	33	4	4	4	4	4	4	4	4	4	4	4

Continúa…

Tabla 15. Pila de entregables. (Continuación...)

	Revisar desarrollo de Punto	Carlos Coba	Pablo Gómez	45	4	4	4	4	4	4	4	4	4	4	4
	Revisar clase MultiPoligono	Carlos Coba	Pablo Gómez	33	0	0	0	0	0	0	5	5	4	5	3
	Revisar desarrollo MultiPoligono	Carlos Coba	Pablo Gómez	24	0	0	0	0	0	0	5	5	4	5	3
	Probar consulta a MongoDB	Carlos Coba	Pablo Gómez	29	4	4	4	4	4	4	4	4	4	4	4
	Probar consultas en PostgreSQL	Carlos Coba	Pablo Gómez	29	2	2	2	2	2	3	3	3	3	2,4	2,4
	Consulta PostgreSQL y MongoDB	Carlos Coba	Pablo Gómez	45	1	1	1	1	1	1	2	2	2	2.5	4
	Revisar la integración	Carlos Coba	Pablo Gómez	40	0	0	0	0	0	0	5	5	4	5	3
	Integrar clases y consultas	Carlos Coba	Pablo Gómez	34	0	0	0	0	0	0	5	5	4	5	3
	Revisar la integración consulta clases	Carlos Coba	Pablo Gómez	50	2	2	2	2	2	3	3	3	3	2,4	2,4
	Integración final.	Carlos Coba	Pablo Gómez	70	1	1	1	1	1	1	2	2	2	2.5	4
				2112											

Elaborado por: Coba Carlos & Gómez Pablo.

CAPÍTULO 4
FASE DE SOPORTE Y CIERRE DEL PROYECTO

4.1 Fase de soporte y mantenimiento

Durante esta fase se expone, la instalación de las diferentes herramientas que se usa para el desarrollo del trabajo de titulación, así como también las posibles soluciones de mantenimiento que se puedan dar durante un problema con la base de datos y el programa a ser entregado y probado debidamente.

4.1.1 Fase del producto para el soporte.

En la fase de soporte para el producto, se mostrará la instalación de la base de datos, el desarrollo y también se describirá el comando a usar, en el caso de que sufra algún inconveniente la base de datos o el programa.

4.1.1.1 Instalación de MongoDB en CentOS.

MongoDB es una base de datos NoSQL, destinada al almacenamiento de grandes cantidades de datos, en la que está orientada a documentos con esquemas dinámicos. NoSQL se refiere a una base de datos, con un modelo de datos que no sea el formato de tabla utilizado en bases de datos relacionales como MySQL, PostgreSQL y Microsoft SQL. MongoDB dentro de sus características incluye: soporte completo de índice, la replicación de alta disponibilidad y auto-sharding. (Sauter & Purkis, 2011)

- Estas instrucciones son para la instalación de MongoDB en un solo nodo CentOS 6.
- Se va a trabajar desde un Core Liquid Web Gestionado CentOS 6.5 del servidor, y va a estar una sesión como root.

Pasó # 1: Agregar el repositorio MongoDB

```
vim /etc/yum.repos.d/mongodb.repo
```

Figura 55. Creación del repositorio.
Fuente: *(Sauter & Purkis, 2011)*

Opción A: Si está ejecutando un sistema de 64 bits, agregue la siguiente información en el archivo que has elaborado, utilizando la tecla insertar:

```
[mongodb]
name=MongoDB Repository
baseurl=http://downloads-distro.mongodb.org/repo/redhat/os/x86_64/
gpgcheck=0
enabled=1
```

Figura 56. Edición del repositorio de 64 bits.
Fuente: *(Sauter & Purkis, 2011)*

Luego salir y guardar el archivo con el comando ": wq". Se puede visualizar una salida similar a la siguiente imagen:

Figura 57. Vista del repositorio.
Fuente: *(Sauter & Purkis, 2011)*

Opción B: Si está ejecutando en un sistema de 32 bits, agregue la siguiente información en el archivo que has Elaborado, utilizando la tecla insertar.

```
[mongodb]
name=MongoDB Repository
baseurl=http://downloads-distro.mongodb.org/repo/redhat/os/i686/
gpgcheck=0
enabled=1
```

Figura 58. Creación del repositorio de 32 bits.
Fuente: *(Sauter & Purkis, 2011)*

Luego salir y guardar el archivo con el comando ": wq".

Pasó # 2: Instale MongoDB

En este punto, la instalación de MongoDB es tan simple como ejecutar un solo comando:

```
yum install mongo mongo-10gen-servidor 10gen
```

Figura 59. Instalación del repositorio.
Fuente: *(Sauter & Purkis, 2011)*

Cuando se le solicite está de acuerdo [Y/N]. Simplemente se escribe, Y luego la tecla enter. Se debe ver una salida similar a la siguiente imagen:

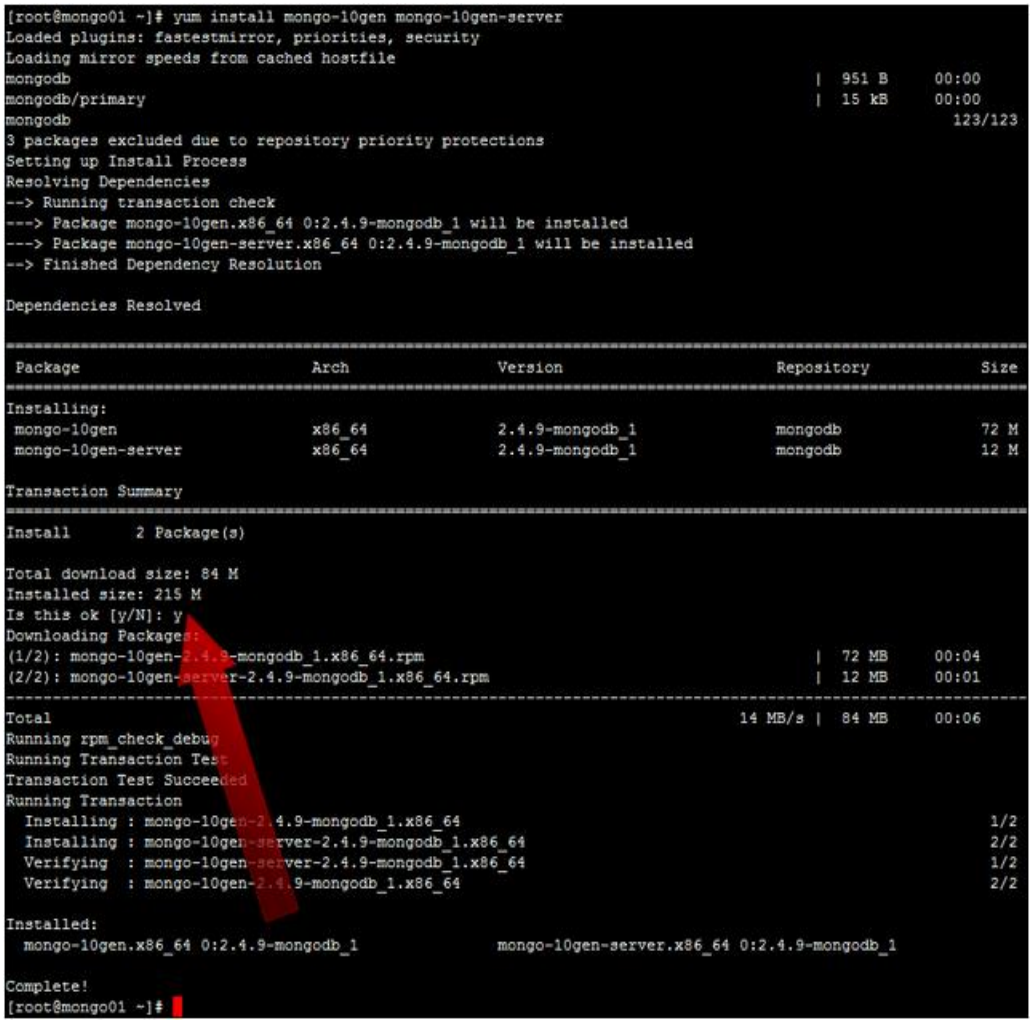


Figura 60. Proceso de instalación del repositorio.
Fuente: *(Sauter & Purkis, 2011)*

Pasó # 3: Arrancar MongoDB.

Puesta en marcha de MongoDB.

```
service mongod start
```

Figura 61. Comando de arranque del servicio de MongoDB.
(Sauter & Purkis, 2011)

Se puede ver una salida similar a la siguiente imagen:

```
[root@mongo01 ~]# service mongod start
Starting mongod: about to fork child process, waiting until server is ready for connections.
forked process: 1423
all output going to: /var/log/mongo/mongod.log
child process started successfully, parent exiting
                                                           [  OK  ]
[root@mongo01 ~]#
```

Figura 62. Verificación que el servicio se está ejecutando.
Fuente: *(Sauter & Purkis, 2011)*

Escribir en la línea de comandos.

```
mongo
```

Figura 63. Inicio de consola a MongoDB.
Fuente: *(Sauter & Purkis, 2011)*

Se observar una salida similar a la siguiente imagen:

```
[root@mongo01 ~]# mongo
MongoDB shell version: 2.4.9
connecting to: test
>
```

Figura 64. Consola de inicio de MongoDB.
Fuente: *(Sauter & Purkis, 2011)*

Esto indica que se inició la consola de consultas de MongoDB

4.1.1.2 Instalación de JBoss en CentOS.

Paso 1: Descargar JBoss.

Para este ejemplo se usa

Jboss 7.1.1.Final.tar.gz

http://www.jboss.org/jboss-as/downloads/

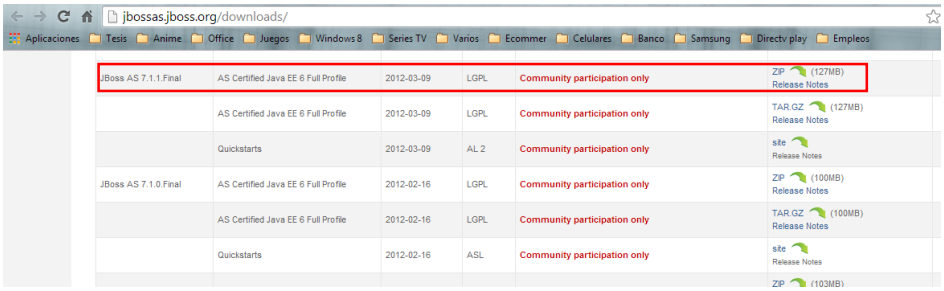


Figura 65. Página de descarga JBoss.
Fuente: *(Morochosv, 2013)*

Paso 2: Crear un usuario y un grupo.

Son los encargados de ejecutar JBoss.

```
# groupadd jboss
# useradd -g jboss -d /opt/jboss jboss
```

Figura 66. Crear usuario.
Fuente: *(Morochosv, 2013)*

Paso 3: Prepara ficheros

A (Continuación...) se extrae el archivo que se descargó y se copia todo su contenido en el directorio hogar del usuario elaborado anteriormente.

```
# tar -xzf jboss-as-7.1.1.Final.tar.gz
# useradd -g jboss -d /opt/jboss/jboss-as-7.1.1.Final jboss
# chown -R jboss.jboss /opt/jboss/jboss-as-7.1.1.Final
```

Figura 67. Prepara ficheros.
Fuente: *(Morochosv, 2013)*

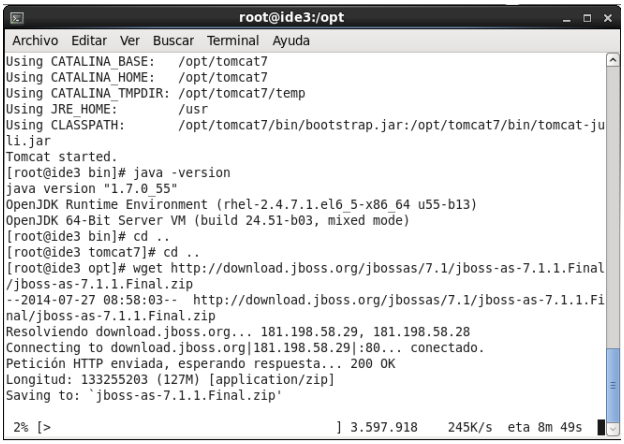


Figura 68. Descarga del Zip de JBoss-7.1.1.
Elaborado por: Coba Carlos & Gómez Pablo.

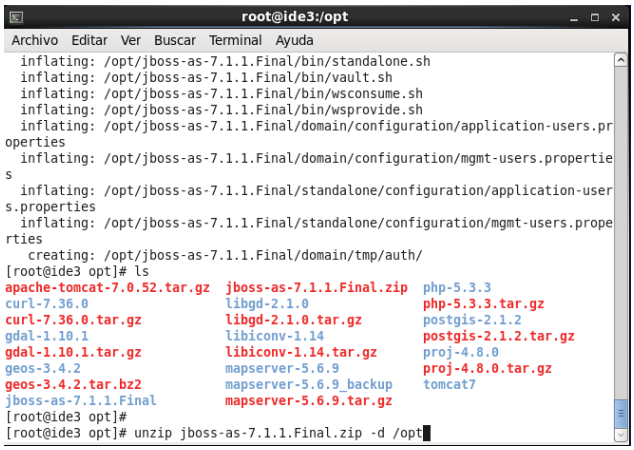


Figura 69. Imagen en servidor descompresión de JBoss.
Elaborado por: Coba Carlos & Gómez Pablo.

Paso 4: Configurar el script de inicio, parada y reinicio

```
cp /opt/jboss/jboss-as-7.1.1.Final/bin/init.d/jboss-as-standalone.sh /etc/init.d/jboss7
cp /opt/jboss/jboss-as-7.1.1.Final/bin/init.d/jboss-as.conf /etc/init.d/
```

Figura 70 Prepara ficheros para el inicio, parada y reinicio.
Fuente: *(Morochosv, 2013)*

Se edita el archivo de configuración /etc/init.d/jboss-as.conf:

```
vim /etc/init.d/jboss-as.conf

JBOSS_USER=jboss
JBOSS_HOME=/opt/jboss/jboss-as-7.1.1.Final
```

Figura 71 Editar ficheros de configuración.
Fuente: *(Morochosv, 2013)*

Se modifica la siguiente línea en /etc/init.d/jboss7 para especificar donde está el archivo de configuración:

```
vim /etc/init.d/jboss7

Load JBoss AS init.d configuration.
      if [ -z "$JBOSS_CONF" ]; then
      JBOSS_CONF="/etc/init.d/jboss-as.conf"
      fi
```

Figura 72. Editar ficheros en la configuración.
Fuente: *(Morochosv, 2013)*

Paso 5: Usuario Jboss.

Agregar un usuario con privilegios de administrador para la consola gráfica. Regresar a la consola y SE EJECUTA:

```
/opt/jboss/jboss-as-7.1.1.Final/bin/add-user.sh
```

Figura 73. Editar ficheros para creación de usuarios.
Fuente: *(Morochosv, 2013)*

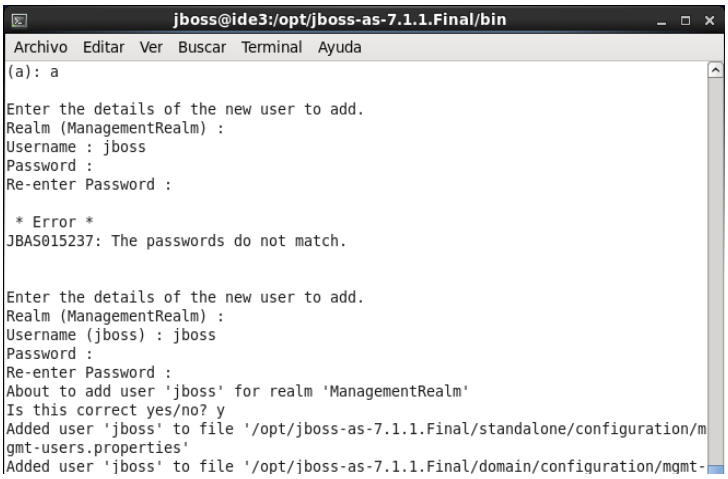


Figura 74. Error y cambio de usuario para la consola de administración.
Elaborado por: Coba Carlos & Gómez Pablo.

NOTA: Hay veces en que indica error de Java en la línea 66 y esto es porque no está exportado en el PATH el JAVA_HOME. Para esto realiza lo siguiente.

```
vim /opt/jboss/jboss-as-7.1.1.Final/bin/standalone.conf
DEFINIMOS EL PATH DEL JAVA

# Specify the location of the Java home directory.  If set then $JAVA will
# be defined to $JAVA_HOME/bin/java, else $JAVA will be "java".
#
JAVA_HOME="/opt/java"
```

Figura 75. Path de configuración.
Fuente: *(Morochosv, 2013)*

4.2 Fase de cierre del producto

El propósito de esta fase es la de evaluar el desempeño del proyecto y determinar si se han logrado cumplir los objetivos y en qué nivel de aceptabilidad se encuentran.

4.2.1 Formalización del cierre del proyecto.

Se definirá por medio de tablas lo realizado y los detalles del documento.

4.2.1.1 Control del documento.

Información del documento.

Tabla 16. Información del documento.

Identificación del documento	Trabajo de titulación para la obtención del titulo
Responsable del documento	Carlos Coba – Pablo Gómez
Fecha de emisión	2014 – 02 – 27
Fechas de última modificación	2014 – 30 – 07
Nombre del archivo	Trabajo de titulación Coba & Gómez

Elaborado por: Coba Carlos & Gómez Pablo.

Historial del documento.

Tabla 17. Historial del documento.

Versión	Fecha de versión	Modificación
Capítulo 1	04 – marzo - 2014	13 – marzo – 2014
Aprobación CAPÍTULO 1	20 – marzo – 2014	3 – abril – 2014
Capítulo 2	10 – abril – 2014	24 – abril – 2014
Aprobación CAPÍTULO 2	129 – abril – 2014	6 – mayo – 2014
Capítulo 3	15 – mayo – 2014	22 – mayo – 2014
Aprobación CAPÍTULO 3	29 – mayo – 2014	6 – junio – 2014
Capítulo 4	13 – junio – 2014	27 – junio – 2014
Aprobación CAPÍTULO 4	3 – julio – 2014	10 – julio – 2014
Conclusiones y Recomendación	17 – julio – 2014	24 – julio – 2014
Final 1.1	31 – julio – 2014	1 - agosto – 2014

Elaborado por: Coba Carlos & Gómez Pablo.

Aprobación.

Tabla 18. Aprobación.

Rol	Nombre	Firma	Fecha
Director de Trabajo de titulación	Ing. Gustavo Navas		31/julio/2014
Lector	Msc. Lina Zapata		16/septiembre/2014
Editor	Carlos Coba & Pablo Gómez		
Desarrollador	Carlos Coba & Pablo Gómez		

Elaborado por: Coba Carlos & Gómez Pablo.

4.2.1.2 Matriz de estados.

La matriz de estados indica el avance con el signo (+) cuando ya se han pasado por esas fase y (-) cuando no se ingresó en esa fase y al finalizar se indica que se ha completado cada una de sus partes a nivel de software y hardware.

Tabla 19. Matriz de estados.

Módulo y Clases	En espera	En progreso	Desechado	Realizado
Clase Conexión	+	+	-	Variables.
				Método de conexión.
				Constructor
				Consulta por query.
				Get y Set.
Clase Consulta	+	+	-	Variables.
				Método de barrido y extracción del formato GeoJSON.
				Constructor.
				Get y Set.
				Búsqueda por profundidad.
				Descarga del formato GeoJSON.
Clase Puntos	+	+	Consulta por profundidad.	Método de barrido y extracción del formato modificado del GeoJSON.
				Constructo.
				Variables.
				Get y Set.
Clase Casa	+	+	-	Variable de entidad.
				Constructor.
				Get y Set.
Clase Obra	+	+	-	Variables de entidad
				Get y Set.
				Constructor.
Clase ServicioCasa	+	+	-	Variable de persistencia.
				Método de inserción.
				Método de recuperar.
				Método de búsqueda por id.

Continúa…

Tabla 18. Matriz de estados. (Continuación...)

Clase ServicioObra	+	+	-	Variable de persistencia.
				Método de inserción.
				Método de recuperar.
				Método de búsqueda por id.
Clase ControladorCasa	+	+	-	Variable.
				EJB para servicio.
				Constructor.
				Método postConstrcut de recuperación.
				Método de inserción.
				Get y Set.
Clase ControladorObra	+	+	-	Variable.
				EJB para servicio.
				Constructor.
				Método postConstrcut de recuperación.
				Método de inserción.
				Get y Set.
Clase MultiPoligono	+	+	-	Variable.
				Get y Set.
				Método de recuperación de documento
				Método de barrido y definición de estructura del MultiPoligono.
				Método de barrido y definición de estructura del Polígono.
				Método de barrido y definición de estructura del Punto.
				Método de formato para la visualización del MultiPoligono.

Continúa…

Tabla 18. Matriz de estados. (Continuación...)

Visor.xhtml	+	+	-	Manejo de menús Búsqueda por selectOneMenu Manejo de botones Visualizar el punto en el Google Map Mostrar informaciones imagen de la obra
MultipolignoVisor.xhtml	+	+	-	Manejo de menús Búsqueda por selectOneMenu Manejo de botones Visualizar el punto en el Google Map
Home.xhtml	+	+	-	Manejo de menús Manejo de botones Visualizar información de las herramientas
Servidor	+	+	Tomcat 7.0	PostgreSQL 9.1 JBoss. 7.1.1 MongoDB

Elaborado por: Coba Carlos & Gómez Pablo.

CONCLUSIONES

- En la investigación se determinó que el almacenamiento de datos geográficos en MongoDB es muy efectiva, pero la falta de información y de consultas a profundidad es un limitante para el manejo de los datos geográficos con mayor eficiencia.
- Durante la investigación se observó que MongoDB se adapta a cualquier lenguaje de programación y como resultado se logró desarrollar el geoportal.
- El manejo de la metodología SCRUM es muy eficiente, en el caso de proyectos, ya que se puede hacer un avance significativo con la revisión del cliente y bajo la línea de requerimientos solicitados por este.
- En el sistema durante su desarrollo se pudo observar algunas dificultades ya que la falta de información en relación a MongoDB con JSF y PrimeFaces en la creación de geoportales es muy escasa.
- Como resultado de la investigación se determinó que la viabilidad técnica, con el lenguaje utilizado no es la mejor, ya que presenta una mejor aceptación con el uso de librerías como es POJO, Spring Data, API Google, GSON y lenguajes como PHP, JSF Maven, Java Script y Punto NET.
- Las pruebas realizadas durante las diferentes etapas de desarrollo y creación no llevaron a observar que la aplicación es muy estable y optima en el manejo de datos geográficos y es un gran almacenador de archivos planos.

RECOMENDACIONES

- Para próximos trabajo tomar en cuenta, el uso de consultas en la base de datos MongoDB a mayor nivel de profundidad en el caso del formato de GeoJSON.
- Investigar más profundamente en que aplicaciones es más factible el uso de las bases de datos NoSQL, ya que son muy útiles y muy eficientes en el manejo de datos.
- La utilización de otros lenguajes de programación como: el caso de PHP, JavaScript JSF Maven, Api's de Java y Punto Net, para un mejor desarrollo y solvencia de aplicaciones futuras con la base NoSQL.
- Que se dé mayor énfasis en el la utilización de la matriz de requerimientos, sprints y entregables que maneja la metodología SCRUM.

LISTA DE REFERENCIAS

Aulet, J. P. (9 de noviembre de 2011). *SINDIKOS*. Recuperado el 4 de abril de 2014, de http://www.sindikos.com/2011/11/introduccion-a-las-bases-de-datos-nosql-mongodb/

Avelis. (17 de mayo de 2013). *Stackoverflow*. Recuperado el 29 de junio de 2013, de GeoJSON y MongoDB: ¿Vale la pena para almacenar puntos como GeoJSON.Point?: http://stackoverflow.com/questions/16128851/geojson-and-mongodb-is-it-worth-it-to-store-points-as-geojson-point

Calendamaia. (14 de mayo de 2013). *Genbeta:dev*. Recuperado el 24 de julio de 2014, de http://www.genbetadev.com/frameworks/primefaces-framework-sobre-jsf-2-0-primeros-pasos

Chodorow, K. (2010). *MongoDB The Definitive Guia* (Segunda ed.). New York, Estados Unidos: O'REILLY. Recuperado el 28 de junio de 2013, de http://books.google.es/books?id=uGUKiNkKRJ0C&printsec=frontcover&dq=mongodb&hl=es&sa=X&ei=qMrTUfymCYGC9QTC7YHQDg

Çivici, Ç. (2014). *PrimeFaces user Guide 5.0* (Vol. 5.0). Croacia: JavaServer Faces Community. Recuperado el 02 de mayo de 2014, de http://www.primefaces.org/docs/guide/primefaces_user_guide_5_0.pdf

Cofre, V., & Toledo, S. (2014). *Tesis.* Recuperado el 21 de marzo de 2014

Correa, M. A. (04 de febrero de 2013). *bitácora de @macool*. Recuperado el 17 de septiembre de 2013, de http://macool.me/mysql-vs-postgresql-vs-mongodb-velocidad/04

Escudero, M. (2 de julio de 2013). *Xenode Systems*. Obtenido de http://blog.xenodesystems.com/2012/10/porque-amaras-mongodb-la-mejor-base-de.html

Hlavats, I. (2009). *Jsf 1.2 Components: Develop Advanced Ajax-Enabled Jsf Applications* (Vol. I). Birminghan, UK: Packt. Recuperado el 28 de junio de 2013, de http://books.google.com.ec/books/about/Jsf_1_2_Components.html?id=x2NfHGw5nUsC&redir_esc=y

Horowitz, E. (2 de abril de 2013). *mongoDB*. Recuperado el 28 de junio de 2013, de Introducción a MongoDB: http://www.mongodb.org/about/introduction/

Howard, B., Daly, M., Doyle, A., Gillies, S., Schaub, T., & Schmidt, C. (16 de junio de 2008). *The GeoJSON Format Specification*. Recuperado el 26 de junio de 2013, de http://geojson.org/geojson-spec.html

Json. (12 de noviembre de 2012). *JSON*. Recuperado el 01 de julio de 2013, de http://www.json.org/json-es.html

Mario. (17 de febrero de 2014). *Solvetic*. Obtenido de http://www.solvetic.com/tutorials/article/468-convertir-bases-de-datos-relacionales-y-sql-para-mongodb/

Moreno Jiménez, A. (2008). *Sistemas y análisis de la información geográfica.* México: Alfaomega. Recuperado el 29 de junio de 2013

Morochosv. (26 de julio de 2013). *Nexolinux*. Obtenido de http://www.nexolinux.com/instalacion-de-jboss-comunity-edition-7-1/

Moya Honduvilla, J., Bernabé Poveda, M., & Manrique Sancho, M. (30 de noviembre de 2007). *La usabilidad de los geoportales: Aplicación del Diseño Orientado a Metas (DOM).* Recuperado el 29 de junio de 2013, de http://www.orzancongres.com/administracion/upload/imgPrograma/N-033.pdf

Palacios, J. (16 de octubre de 2006). *El modelo SCRUM.* Recuperado el 30 de junio de 2013, de Navegapolis: http://www.navegapolis.net/files/s/NST-010_01.pdf

Requena, C. (10 de abril de 2010). *MongoDB.* Recuperado el 29 de junio de 2013, de NoSQL.es: http://www.nosql.es/blog/nosql/mongodb.html

Rodriguez, A. E. (13 de octubre de 2013). *GeekyTheory*. Obtenido de http://geekytheory.com/json-i-que-es-y-para-que-sirve-json/

Rubenfa. (03 de febrero de 2014). *Genbeta desarrollo y software*. Obtenido de http://www.genbetadev.com/bases-de-datos/mongodb-que-es-como-funciona-y-cuando-podemos-usarlo-o-no

Ruiz, M. (11 de octubre de 2006). *Infogeo*. Recuperado el 27 de junio de 2013, de Que es un SIG: http://www.infogeo.cl/index.php?option=com_content&view=article&id=26&Itemid=40

Sauter, C., & Purkis, M. (28 de julio de 2011). *Liquidweb*. Recuperado el 5 de mayo de 2014, de http://www.liquidweb.com/kb/how-to.install-mongodb-on-centos-6/

Sumosa, J. (08 de agosto de 2011). *Scribd*. Obtenido de http://es.scribd.com/doc/61815330/ScrumManager-Gestion-de-proyectos

Sun, P. F. (2011). *Web GIS Principles and applications.* California: Ingram. Recuperado el 28 de junio de 2013

Tong, K. K. (2009). *Beginning JSF 2 APIs y JBoss Seam.* New York: Appres. Recuperado el 29 de junio de 2013, de http://books.google.es/books?id=VA7mR9Gx4ysC&printsec=frontcover&dq=jsf&hl=es&sa=X&ei=HMXTUZeZFIr29gTfioAo&ved=0CDMQ6AEwAA#v=onepage&q=jsf&f=false

Trigas, M. (2012). *Metodologia SCRUM.* Catalunya: Universidad Oberta de Catalunya. Recuperado el 01 de julio de 2013, de

http://openaccess.uoc.edu/webapps/o2/bitstream/10609/17885/1/mtrigasTFC0612memoria.pdf

Varios. (23 de septiembre de 2013). *mongoBD*. Recuperado el 08 de noviembre de 2013, de http://docs.mongodb.org/manual/tutorial/install-mongodb-on-red-hat-centos-or-fedora-linux/

GLOSARIO DE TÉRMINOS

JSON: (JavaScript Object Notation & Notación de Objetos de JavaScript) es un formato ligero de intercambio de datos.

GeoJSON: Es un estándar abierto de formato para la codificación de las colecciones de las características geográficas simples junto con sus atributos no espaciales que utilizan JavaScript Object Notation.

BSON: Acrónimo de Binary JSON, es una serialización binaria codificada de documentos JSON& como. Al igual que JSON, BSON soporta la incorporación de documentos y matrices dentro de otros documentos y matrices.

MongoDB: (de la palabra en inglés "humongous" que significa enorme) es un sistema de base de datos NoSQL orientado a documentos, desarrollado bajo el concepto de código abierto.

GeoMongoDB: (Geoportal MongoDB) Es el nombre del geoportal que se le dio a nuestra trabajo de titulación.

NoSQL: ("no sólo SQL") es una amplia clase de sistemas de gestión de bases de datos que difieren del modelo clásico del sistema de gestión de bases de datos relacionales (RDBMS) en aspectos importantes, el más destacado que no usan SQL como el principal lenguaje de consultas.

SQL: El lenguaje de consulta estructurado o SQL (por sus siglas en inglés Structured Query Language) es un lenguaje declarativo de acceso a bases de datos relacionales que permite especificar diversos tipos de operaciones en ellas.

EJB: (también conocidos por sus siglas EJB) son una de las API que forman parte del estándar de construcción de aplicaciones empresariales J2EE.

RDBMS: (Sistema de Gestión de Base de Datos Relacional o). Tipo de SGBD para bases de datos relacionales (que emplea el modelo de datos) o sea, soporte de tablas relacionadas.

JBOSS: Es un servidor de aplicaciones Java EE de código abierto implementado en Java puro. Al estar basado en Java, JBoss puede ser utilizado en cualquier sistema operativo para el que esté disponible la máquina virtual de Java

ACID: (Atomicidad, Consistencia, Aislamiento y Durabilidad en español) En bases de datos se denomina a un conjunto de características necesarias para que una serie de instrucciones puedan ser consideradas como una transacción.

POJO: Es un acrónimo de Plain Old Java Object. El nombre se utiliza para enfatizar que un determinado objeto es un ordinario Java Object, no un objeto especial.

SCRUM: Es un modelo de desarrollo ágil caracterizado por adoptar una estrategia de desarrollo incremental, en lugar de la planificación y ejecución completa del producto, basar la calidad del resultado más en el conocimiento tácito de las personas en equipos auto organizados, que en la calidad de los procesos empleados y solapamiento de las diferentes fases del desarrollo, en lugar de realizar una tras otra en un ciclo secuencial o de cascada.

CRUD: Es el acrónimo de Crear, Obtener, Actualizar y Borrar, se usa para referirse a las funciones básicas en bases de datos o la capa de persistencia en un software.

JSF: JavaServer Faces (JSF) es una tecnología y framework para aplicaciones Java basadas en web que simplifica el desarrollo de interfaces de usuario en aplicaciones Java EE.

SIG: Un Sistema de Información Geográfica es un conjunto de herramientas que integra y relaciona diversos componentes que permiten la organización, almacenamiento, manipulación, análisis y modelización de grandes cantidades de datos procedentes del mundo real que están vinculados a una referencia espacial.

PHP: Es un lenguaje de programación de uso general de código del lado del servidor originalmente diseñado para el desarrollo web de contenido dinámico.

Anexo 1. Manual de usuario v 1.1

Introducción

El "Manual de Usuario", tiene como finalidad dar a conocer de una manera, visual y sencilla la forma de cómo usar el geoportal, desarrollado como proyecto de trabajo de titulación en la Universidad Politécnica Salesiana, Sede Quito, Campus Sur.

El manual está destinado a cualquier usuario para que pueda utilizar de una forma correcta el geoportal.

El geoportal fue diseñado para que el usuario final pueda, de una forma intuitiva y sin mayor capacitación encontrar el sitio salesiano que busca.

Requerimientos

- El geoportal necesita los siguientes requerimientos de software:
- Internet Explorer 7 o superior
- Mozila Firefox 3 o superior
- Google Chrome 4 o superior
- Se necesitan los siguientes requerimientos de hardware:
- CPU con conexión a internet

1.- Entrar al geoportal

Para comenzar a utilizar el geoportal y visualizar los diferentes sitios salesianos, se abrirá un navegador y colocar la siguiente dirección web: http://190.15.136.4:8090/TesisGeoportalMongoDB/Home.jsf

Figura 76. Pantalla de inicio.
Elaborado por: Coba Carlos & Gómez Pablo.

2.- Dentro del geoportal existen varios botones que se explican a (Continuación...):

Figura 77 Pantalla de selección.
Elaborado por: Coba Carlos & Gómez Pablo.

3.- Primero se escoge la casa salesiana que se desea buscar.

Figura 78. Pantalla de selección de casa.
Elaborado por: Coba Carlos & Gómez Pablo.

4.- Luego se elige la obra salesiana que se desea graficar en el mapa.

Figura 79. Pantalla de selección de obra.
Elaborado por: Coba Carlos & Gómez Pablo.

5.- Por último y para que la casa salesiana se dibuje en el mapa, se presiona el botón de “Generar”

Figura 80. Pantalla de vista de generar multipolígono
Elaborado por: Coba Carlos & Gómez Pablo.

6.- Si se desea descargar el archivo geojson se presiona en el botón de “Descargar” para proceder a genera una copia de la obra salesiana seleccionada:

Figura 81. Pantalla de vista de multipolígono.
Elaborado por: Coba Carlos & Gómez Pablo.

7.- Si lo que se quiere es dibujar en el mapa el punto donde se encuentra una obra salesiana lo que se hacer es lo siguiente:

Se escoge el botón “Visor de Punto con Geojson”

Figura 82. Pantalla de vista de puntos.
Elaborado por: Coba Carlos & Gómez Pablo.

8.- De igual manera se escoge la casa salesiana y luego la obra salesiana que se quiere que nos muestre en el mapa

Figura 83. Pantalla de selección de puntos.
Elaborado por: Coba Carlos & Gómez Pablo.

9.- Luego se da clic en el botón “Buscar” y se genera el punto en el mapa donde está ubicada la obra salesiana.

Figura 84. Pantalla de visualización de puntos.
Elaborado por: Coba Carlos & Gómez Pablo.

Anexo 2. Diagrama de GANT