REPÚBLICA BOLIVARIANA DE VENEZUELA

UNIVERSIDAD RAFAEL URDANETA

FACULTAD DE INGENIERÍA

ESCUELA DE INGENIERÍA DE COMPUTACIÓN



**DISEÑO DE UN COMPONENTE DISTRIBUIDO BASADO EN WEBSERVICES PARA EL ACCESO A BASES DE DATOS**

Trabajo Especial de Grado presentado ante la

Universidad Rafael Urdaneta para optar al título de:

INGENIERO DE COMPUTACIÓN

Autor: Ing. ALBERTO BONSANTO

Tutor: MSc. Júbert Pérez

Maracaibo, Julio de 2015

**DISEÑO DE UN COMPONENTE DISTRIBUIDO BASADO EN WEBSERVICES PARA EL ACCESO A BASES DE DATOS**

**DISEÑO DE UN COMPONENTE DISTRIBUIDO BASADO EN WEBSERVICES PARA EL ACCESO A BASES DE DATOS**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Bonsanto Pocaterra, Alberto Andrés

C.I. 19.623.747

Calle 75 Av. 13 Edif. Caroní Apto. 3B

Telf.: (0416) 5159700

alberto.bonsanto@hotmail.com

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Pérez Zabala, Júbert José

Tutor Académico

**APROBACIÓN**

Este jurado aprueba el Trabajo Especial de Grado titulado: “**DISEÑO DE UN COMPONENTE DISTRIBUIDO BASADO EN WEBSERVICES PARA EL ACCESO A BASES DE DATOS**”, presentado por el bachiller: **ALBERTO A. BONSANTO P**, portador de la C.I: V-19.623.747, en cumplimiento con los requisitos establecidos para optar por el título de INGENIERO DE COMPUTACIÓN.

**ÍNDICE GENERAL**

[RESUMEN 10](#_Toc425411114)

[ABSTRACT 11](#_Toc425411115)

[INTRODUCCIÓN 12](#_Toc425411116)

[CAPÍTULO I EL PROBLEMA 13](#_Toc425411117)

[1.1 PLANTEAMIENTO DEL PROBLEMA 13](#_Toc425411119)

[1.2 FORMULACIÓN DEL PROBLEMA 15](#_Toc425411120)

[1.3 OBJETIVOS 16](#_Toc425411121)

[1.3.1 OBJETIVO GENERAL 16](#_Toc425411122)

[1.3.2 OBJETIVOS ESPECÍFICOS 16](#_Toc425411123)

[1.4 JUSTIFICACIÓN 17](#_Toc425411124)

[1.5 DELIMITACIÓN DE LA INVESTIGACIÓN 18](#_Toc425411125)

[1.5.1 DELIMITACIÓN ESPACIAL 18](#_Toc425411126)

[1.5.2 DELIMITACIÓN TEMPORAL 18](#_Toc425411127)

[1.5.3 DELIMITACIÓN CIENTÍFICA 19](#_Toc425411128)

[CAPÍTULO II marco teórico 20](#_Toc425411129)

[2.1 ANTECEDENTES 20](#_Toc425411131)

[2.2 BASES TEÓRICAS 23](#_Toc425411132)

[2.2.1 JAVA 23](#_Toc425411133)

[2.2.1.1 ANOTACIONES 24](#_Toc425411134)

[2.2.1.2 SERVLET 25](#_Toc425411135)

[2.2.2 HTTP 26](#_Toc425411136)

[2.2.3 FORMATOS DE SALIDA 27](#_Toc425411137)

[2.2.3.1 XML 27](#_Toc425411138)

[2.2.3.2 JSON 29](#_Toc425411139)

[2.2.3.3 DATA SET 29](#_Toc425411140)

[2.2.3.4 CSV 30](#_Toc425411141)

[2.2.4 SISTEMA DE GESTIÓN DE BASE DE DATOS 31](#_Toc425411142)

[2.2.5 OBJETOS DISTRIBUIDOS 33](#_Toc425411143)

[2.2.6 SOA 34](#_Toc425411144)

[2.2.7 REST 35](#_Toc425411145)

[2.2.8 WEB SERVICES 37](#_Toc425411146)

[CAPÍTULO III marco metodológico 39](#_Toc425411147)

[3.1 TIPO DE INVESTIGACIÓN 39](#_Toc425411149)

[3.2 DISEÑO DE LA INVESTIGACIÓN 41](#_Toc425411150)

[3.3 TÉCNICAS DE RECOLECCIÓN DE DATOS. 42](#_Toc425411151)

[3.4 FASES DE LA INVESTIGACIÓN 43](#_Toc425411152)

[CAPÍTULO IV analisis de resultados 46](#_Toc425411153)

[4.1 PLANIFICACIÓN 46](#_Toc425411155)

[4.2 ANÁLISIS PRELIMINAR 47](#_Toc425411156)

[4.2.1 CARACTERÍSTICAS, PRESTACIONES Y FUNCIONAMIENTO 47](#_Toc425411157)

[4.2.2 EVALUACIÓN DE POOL DE CONEXIONES 62](#_Toc425411158)

[4.2.3 ESTRUCTURAS XML 63](#_Toc425411159)

[4.3 DESARROLLO DEL SISTEMA 65](#_Toc425411160)

[4.3.1 SELECCIÓN DEL POOL DE CONEXIONES 65](#_Toc425411161)

[4.3.2 ESTRUCTURA DE LOS ARCHIVOS XML 67](#_Toc425411162)

[4.3.3 DESARROLLO DEL SISTEMA 70](#_Toc425411163)

[4.3.4 PRUEBAS AL SISTEMA 75](#_Toc425411164)

[4.3.4.1 PRUEBAS DE UNIDAD E INTEGRACIÓN 75](#_Toc425411165)

[4.3.4.2 PRUEBAS DE ESTRÉS 76](#_Toc425411166)

[CONCLUSIONES 93](#_Toc425411167)

[RECOMENDACIONES 95](#_Toc425411168)

[REFERENCIAS BIBLIOGRÁFICAS 96](#_Toc425411169)

**ÍNDICE DE FIGURAS**

[Figura 2.1 Ejemplo de un archivo csv 30](#_Toc425410482)

[Figura 4.1 Diagrama de clase del Dispatcher (Dispatcher.java) 51](#_Toc425410483)

[Figura 4.2 Diagrama de clase para DataTable (DataTable.java) 51](#_Toc425410484)

[Figura 4.3 Diagrama de clase para JSON (JSON.java) 51](#_Toc425410485)

[Figura 4.4 Diagrama de clase para JSONWriter (JSONWriter.java) 52](#_Toc425410486)

[Figura 4.5 Diagrama de clase para CSVWriter (CSVWriter.java) 52](#_Toc425410487)

[Figura 4.6 Diagrama de clase de LogHandler (LogHandler.java) 53](#_Toc425410488)

[Figura 4.7 Diagrama de clase de DBConnection (DBConnection.java) 54](#_Toc425410489)

[Figura 4.8 Diagrama de clase de SettingsReader (SettingsReader.java) 54](#_Toc425410490)

[Figura 4.9 Diagrama de clase de ConnectionsReader (ConnectionsReader.java) 55](#_Toc425410491)

[Figura 4.10 Diagrama de clase de QueriesReader (QueriesReader.java) 55](#_Toc425410492)

[Figura 4.11 Diagrama de clase de Query (Query.java) 56](#_Toc425410493)

[Figura 4.12 Diagrama de clase desde la perspectiva del paquete core 57](#_Toc425410494)

[Figura 4.13 Diagrama de clase desde la perspectiva del paquete dependencies 57](#_Toc425410495)

[Figura 4.14 Diagrama de clase del sistema 57](#_Toc425410496)

[Figura 4.15 Diagrama de actividades para la actividad de arranque 59](#_Toc425410497)

[Figura 4.16 Diagrama de actividades para la el método queryJSON 60](#_Toc425410498)

[Figura 4.17 Diagrama de despliegue del sistema 61](#_Toc425410499)

[Figura 4.18 Estructura de un archivo XML 63](#_Toc425410500)

[Figura 4.19 Estructura de un archivo XML representado como un árbol 64](#_Toc425410501)

[Figura 4.20. Archivo settings.xml 67](#_Toc425410502)

[Figura 4.21. Archivo connections.xml 68](#_Toc425410503)

[Figura 4.22. Archivo connections.xml con etiquetas opcionales 69](#_Toc425410504)

[Figura 4.23. Archivo queries.xml 70](#_Toc425410505)

[Figura 4.24. Fragmento de la clase Dispatcher 71](#_Toc425410506)

[Figura 4.25 Fragmento del método main de la clase Dispatcher 72](#_Toc425410507)

[Figura 4.26 Fragmentos de la clase DBConnection 73](#_Toc425410508)

[Figura 4.27 Fragmentos de la clase JSON 74](#_Toc425410509)

[Figura 4.28. Contenido de muestra del archivo endpoint.properties 77](#_Toc425410510)

[Figura 4.29. Contenido de muestra del archivo queries.properties 77](#_Toc425410511)

[Figura 4.30. Fragmento de la clase Test 78](#_Toc425410512)

[Figura 4.31. Fragmento de la clase Starter 79](#_Toc425410513)

[Figura 4.32. Contenido de muestra del archivo endpoints.properties 79](#_Toc425410514)

[Figura 4.33. Diagrama simplificado de eventos durante la prueba 81](#_Toc425410515)

[Figura 4.34. Resultados en ms para el nodo 0 82](#_Toc425410516)

[Figura 4.35. Resultados en ms para el nodo 1 83](#_Toc425410517)

[Figura 4.36. Resultados en ms para el nodo 2 84](#_Toc425410518)

[Figura 4.37. Resultados en ms para el nodo 3 85](#_Toc425410519)

[Figura 4.38. Resultados en ms para el nodo 4 86](#_Toc425410520)

[Figura 4.39. Resultados en ms para el nodo 5 87](#_Toc425410521)

[Figura 4.40. Resultados en ms para el nodo 6 88](#_Toc425410522)

[Figura 4.41. Resultados en ms para el nodo 7 89](#_Toc425410523)

[Figura 4.42. Resultados en ms para el nodo 8 90](#_Toc425410524)

[Figura 4.43. Resultados en ms para el nodo 9 91](#_Toc425410525)

**ÍNDICE DE TABLAS**

Tabla 4.1. Paquetes con sus respectivas clases…………………………………….57

BONSANTO P., Alberto Andrés, **“DISEÑO DE UN COMPONENTE DISTRIBUIDO BASADO EN WEBSERVICES PARA EL ACCESO A BASES DE DATOS”**. Trabajo Especial de Grado para optar al Título de Ingeniero de Computación; Maracaibo – Venezuela: Universidad Rafael Urdaneta, Facultad de Ingeniería, Escuela de Ingeniería de Computación, 2015.

## RESUMEN

El presente trabajo de investigación tuvo como objetivo central, el diseño y desarrollo de un componente distribuido para el acceso a base de datos mediante la utilización de la tecnología de Webservices a través del protocolo SOAP. Para la realización de este componente se realizaron diferentes análisis, que permitieron determinar, el lenguaje de programación más apropiado para el desarrollo del mismo, las características mínimas requeridas para el pool de conexiones a bases de datos, la arquitectura necesaria para cohesionar los módulos que forman al componente, el mecanismo para el manejo de excepciones y errores durante la ejecución de la aplicación y el módulo que mantenimiento de registros temporales del comportamiento de la aplicación (logs). Metodológicamente la investigación se clasificó como proyectiva, ya que contempló el desarrollo de un proyecto, a su vez el diseño quedó enmarcado dentro de la clasificación de multivariable y cuasi-experimental, debido a que requirió un proceso iterativo de análisis, desarrollo e implementación que conllevó a alcanzar los objetivos planteados. Por otro lado, entre los beneficios del componente desarrollado están, la reducción del tiempo requerido por los desarrolladores para la elaboración de sistemas empresariales, la facilidad de integración con múltiples plataformas más detalladamente sistemas operativos basados en UNIX o Windows garantizando la versatilidad al permitir conectarse con varios manejadores de bases de datos de forma remota o local.

**Palabras clave:** Webservices, componente distribuido, SOAP, bases de datos.

BONSANTO P., Alberto Andrés, **“DESIGN OF A DISTRIBUTED COMPONENT BASED ON WEBSERVICES TO ACCESS DATABASES”**. Special Degree work to qualify for the title of Computing Engineer, Maracaibo – Venezuela: Rafael Urdaneta University, Engineering Faculty, School of Computing Engineering, 2015.

## ABSTRACT

The present researching project, had as central objective the design and development of a distributed component to access databases by using the technology of Webservices and the SOAP protocol. To carry out the development of this component some different analyses had to be done, allowing to choose, the most appropriate programming language to develop the system, minimal characteristics required for the database connection pool, the architecture needed to unite the modules which form the component, the mechanism to manage the exceptions and errors during the application run and the module that maintains the temporal records of the application behavior (logs). Methodologically this research was classified as projective, since the developing of a project was contemplated, likewise de design was established as multivariable and quasi-experimental, because it required an iterative process of analysis, developing and implementation that led to reach the previously defined objectives. On the other hand, as benefits of the developed component are: the reduction of time required by the developers to elaborate enterprise applications, ease integration with multiple platforms meaning operating systems based on UNIX or Windows, granting the versatility by allowing the remote or local connection with several relational database systems.

**Keywords:** Webservices, distributed component, SOAP, database.

## INTRODUCCIÓN

En la actualidad, las grandes empresas poseen sistemas empresariales muy grandes que escalan en magnitud de manera constante, requiriendo cada vez más y más sistemas de almacenamiento de información y en consecuencia más bases de datos a las que sus subsistemas básicos se tienen que conectar, por ello las grandes empresas necesitan componentes de software versátiles y que sean de código abierto, con el objeto de permitir modificaciones o incorporaciones de código útiles para cada empresa en particular.

A partir de la búsqueda de la eficiencia en el desarrollo de sistemas, se presentan casos en los cuales los componentes o sistemas desarrollados están sumamente acoplados, y en algunos casos son incompatibles entre sí, generando zonas aisladas de procesamiento, en los que cada subsistema trabaja de forma independiente, produciendo numerosos problemas a la hora de realizar operaciones que requieren información de diferentes lugares o bases de datos. Por lo tanto hay que emplear alguna estrategia que permita a estas aplicaciones acceder a bases de datos remotas o locales y facilitar que múltiples clientes se puedan conectar a los servicios provistos por un servicio, el cual funciona en realidad como un puente entre los clientes y las bases de datos.

Por lo tanto al desarrollar un sistema que cubra las necesidades de permitir a múltiples bases de datos, localizadas de manera remota o local, y pudiendo ser de manejadores diferentes facilitaría a las empresas en general la implementación de sistemas robustos, distribuidos e interoperables, que reducirían enormemente el tiempo y la complejidad de desarrollo de los subsistemas básicos de las empresas.

CAPÍTULO I

EL PROBLEMA

1.1 PLANTEAMIENTO DEL PROBLEMA

En el inicio del desarrollo de las aplicaciones empresariales, éstas se caracterizaban por ser monolíticas, es decir que la aplicación tenía combinados aspectos que en la actualidad suelen estar separados y protegidos; entre los problemas más comunes de este tipo de aplicaciones se observaban que todas las entradas del usuario, la lógica de validación, el programa del negocio y el acceso a datos estaban juntos y combinados en un solo programa (Liu, 2004).

De tal manera, en este período de la computación, era común encontrarse un computador personal de escritorio donde se almacenaba toda la aplicación, con lo cual se observaban aplicaciones que poseían toda la funcionalidad en una gran pieza de software, y debido a su proporción era muy difícil de mantener y mejorar, porque cualquier pequeño cambio podía afectar a otras piezas del programa y en consecuencia incluir errores o incluso dejar dicha aplicación inoperante.

Por lo tanto, la tarea de depurar y mejorar una aplicación de este tipo era muy complicada, y al pasar de los años se volvió más imperiosa la necesidad de diseñar una arquitectura capaz de permitir el desarrollo de aplicaciones escalables, multiplataforma, de fácil mantenimiento, que permitiesen la división del programa en subsistemas con cierto grado de independencia, con capacidad de comunicarse con otros programas o sistemas que estuvieran ubicados en otra locación distante, que a su vez se estuviesen ejecutando en una plataforma o sistema operativo distinto a la aplicación principal y que tuvieran la suficiente robustez para mantener un alto grado de seguridad y estabilidad a lo largo del tiempo

En la actualidad, el modelo de computación compuesta es la dirección en la cual la computación se ha ido avocando desde que las redes de transmisión de datos se volvieron baratas y fáciles de utilizar. Por lo que en vez de tratar de construir aplicaciones que requieran computadores más potentes, se intenta ensamblar componentes de software más pequeños que interactúen entre sí a través de redes muchos computadores ubicados a millas de distancia (Chappell & Jewell, 2002).

Como propuesta de desarrollo surgieron los sistemas interoperables. La interoperabilidad que los caracteriza se puede observar cuando dos o más sistemas o componentes que independientemente de la ubicación física, la plataforma operativa o el lenguaje de programación, poseen la capacidad de comunicarse a través del intercambio de datos mediante la utilización de diferentes vías de comunicación, permitiendo así la manipulación de la información intercambiada entre las diferentes partes.

Los sistemas distribuidos existen en su mayor parte como entidades débilmente acopladas que se comunican entre sí para resolver una tarea. Uno de los modelos más comúnmente usados en las aplicaciones distribuidas son las llamadas remotas a procedimientos, Remote Procedure Call (RPC), por sus siglas en inglés. Una razón de la popularidad de los sistemas basados en RPC es que ellos se asemejan a la sintaxis y a la semántica de la llamada a métodos y funciones a los cuales los programadores están familiarizados.

Una alternativa de este tipo, es la Arquitectura Orientada a Servicios, Service Oriented Architecture (SOA) por sus siglas en inglés, el cual es un paradigma para la realización y mantenimiento de procesos que requieren un conjunto de sistemas robustos y distribuidos. Este esquema de desarrollo está basado en tres conceptos básicos dirigidos a cumplir con: los servicios, la interoperabilidad y el bajo acoplamiento (Josuttis, 2007). En este ámbito arquitectónico se ubican los Servicios Web (Web Services) en inglés, que son una pieza de lógica del negocio, localizada en algún lugar de la Internet que son accesibles a través de los protocolos estandarizados como el Hypertext Transfer Protocol (HTTP), y Simple Mail Transfer Protocol (SMTP) (Chappell & Jewell, 2002).

Por lo tanto al diseñar un componente de código abierto que contribuya a evitar la elaboración de sistemas monolíticos, que permita la conexión con cualquier base de datos así sea remota, utilizando un estándar de conexión a través de un protocolo robusto y que en consecuencia favorezca el diseño del sistema con características de una arquitectura orientada a servicios, garantizaría a los diseñadores de sistemas empresariales, o sistemas que requieran un alto grado de interoperabilidad la implementación de un componente seguro, robusto, que permitiera la comunicación con cualquier lenguaje, en cualquier plataforma operativa y siendo dicho componente de código abierto.

1.2 FORMULACIÓN DEL PROBLEMA

Dada a la gran necesidad de acceso a sistemas de información mediante la utilización de base de datos, se plantearon un conjunto de cuestionamientos que contribuyeron directa o indirectamente a discernir como abordar el problema.

* ¿Cómo reducir significativamente la escritura de código en las aplicaciones distribuidas que poseen sistemas de acceso a bases de datos?
* ¿Qué conjunto de acciones se deben realizar para disminuir los niveles altos de latencia debido a la múltiple concurrencia en el acceso a base de datos?
* ¿Cómo permitir que varios clientes puedan acceder de forma simultánea a diferentes bases de datos?
* ¿Cómo reducir el fuerte acoplamiento en los sistemas actualmente desarrollados sin representar una tarea complicada para los desarrolladores?
* ¿Por qué se deben desarrollar esquemas de integración abiertos e interoperables para sistemas externos?

1.3 OBJETIVOS

1.3.1 OBJETIVO GENERAL

El siguiente objetivo engloba resumidamente el núcleo de la investigación.

* Desarrollar un componente interoperable para el acceso a bases de datos remotas basado en Web Services.

1.3.2 OBJETIVOS ESPECÍFICOS

A continuación se presentan el listado de objetivos que ser desarrollados para completar la investigación.

* Determinar las características y requerimientos que deben ser cumplidos por el componente.
* Analizar detalladamente las especificaciones del componente.
* Planificar las actividades requeridas para el desarrollo del componente para el acceso a bases de datos.
* Diseñar los componentes que conforman a la librería.
* Codificar los componentes diseñados de la librería.
* Realizar pruebas de unidad, integración y estrés a los componentes que conforman al sistema.
* Desarrollar una aplicación demostrativa que implemente dicho componente.

1.4 JUSTIFICACIÓN

Actualmente las compañías de desarrollo de software han observado un aumento relevante de su importancia en la economía global y esto se debe al crecimiento de la demanda de software, a su vez las grandes empresas que poseen sistemas o plataformas enormes de información enfrentan el problema de mantener de forma organizada sus sistemas para la gestión de base de datos, además como cualquier compañía buscan la maximización de la productividad de sus procesos y el incremento de la eficiencia en sus tecnologías, de cierta forma en éste ámbito se ubica esta investigación, ya que presenta contribuciones para mejorar la forma en que los sistemas grandes están diseñados en especial para aquellos cuya plataforma sea distribuida.

A pesar de los avances en el desarrollo de software, es común que las empresas del sector industrial posean sistemas que son fuertemente acoplados, y que en algunos casos poseen múltiples bases de datos incompatibles y que deben estar localizadas cerca de la ubicación donde está el servidor encargado de interpretar el contenido de las múltiples bases de datos, produciendo que sus sistemas internos sean más lentos de lo esperado y en otros casos impidiendo que algunos de las bases de datos usadas por el sistema empresarial no puedan ser distribuidas dificultando a su vez la escalabilidad de éste.

Por lo tanto esta investigación proporciona aportes valiosos y beneficios a los arquitectos y desarrolladores de software, ya que contribuye a solventar algunos de los problemas típicos ligados a la administración de sistemas empresariales, en especial para aquellos sistemas que requieren consultar múltiples y diferentes bases de datos; y al ser software libre podrá ser utilizado por cualquier interesado. Para lograr esto se debe seleccionar un lenguaje de programación lo suficientemente maduro, escalable y con capacidad de soporte para sistemas remotos, Java cumple con dichas características además es de naturaleza código abierto y permite auditar proyectos de software de alta envergadura y que sin éste dificultaría el desarrollo de esta aplicación.

1.5 DELIMITACIÓN DE LA INVESTIGACIÓN

1.5.1 DELIMITACIÓN ESPACIAL

El presente trabajo especial de grado se llevó cabo dentro de las instalaciones de la Universidad Rafael Urdaneta, ubicada en la nueva sede “Vereda del Lago”, ubicada específicamente en la entrada sir de la Vereda del Lago, Av. 2 “El Milagro” con la Calle 86 (La Calzada). Maracaibo, Venezuela.

1.5.2 DELIMITACIÓN TEMPORAL

La investigación tuvo una duración de 9 meses desde Octubre del 2014 hasta Julio del 2015.

1.5.3 DELIMITACIÓN CIENTÍFICA

El presente trabajo especial de grado se desarrolló dentro de la Ingeniería de Computación, específicamente dentro del área de la programación y desarrollo de aplicaciones.

CAPÍTULO II

MARCO TEÓRICO

2.1 ANTECEDENTES

En el año 2011 Anaya L. Emilio, realizó el trabajo especial de grado para obtener el grado de maestro en ciencias en informática, titulado “**IMPLEMENTACIÓN DE CONTROL DE SEGURIDAD EN ARQUITECTURA ORIENTADAS A SERVICIOS (SOA) PARA SERVICIOS WEB**”, en el Instituto Politécnico Nacional, Sección de Estudios de Postgrado e Investigación, México, D. F, Estados Unidos Mexicanos.

El objetivo general de esta investigación fue el de implementar controles de seguridad que permitieran y garantizaran la autorización, autenticación, no repudio de la información, confidencialidad e integridad en los servicios web, con la intención de generar una guía práctica que ofreciera una solución a los problemas de seguridad que se presentan cuando se utilizan servicios web para la integración de aplicaciones y servicios.

La investigación tuvo como finalidad proporcionar una solución práctica a los problemas relacionados con la seguridad en los servicios web al momento de intercambiar información entre dos o más sistemas, planteando un caso práctico en el que se desarrolla una propuesta de solución y el procedimiento detallado para resolver los problemas haciendo uso de los estándares derivados del lenguaje extensible de marcas, cuyo nombre original en inglés es Extensible Markup Language (XML). Más específicamente se analizaron estándares como WS Security, XML Signature, XML Encryption y Security Assertion Markup Language (SAML).

El resultado de dicha investigación determina que el uso de estándares de seguridad basados en archivos con formato XML es una alternativa segura, robusta y confiable con muy pocos problemas de seguridad, problemas a los que encuentran expuestos los servicios web, por lo tanto este proyecto contribuyó a esclarecer que entre los formatos de datos que permiten comunicar diferentes aplicaciones está el XML, además de detallar sobre diferentes elementos entre los que se incluyen la seguridad y el cifrado de los archivos en formato XML.

En el año 2009 Jesica Zavala, Daniel Martínez y Dante Rivera realizaron la tesis de pregrado titulada “**DESARROLLO DE UN DIRECTORIO USANDO UN SERVICIO WEB**”, la cual fue realizada en el Instituto Politécnico Nacional, Escuela Superior de Ingeniería Mecánica y Eléctrica, Ciudad de México D.F, Estados Unidos Mexicanos.

El objetivo general de la investigación consistió en diseñar una aplicación que utilizara servicios web para consultar, agregar y modificar datos de profesores de la base de datos del directorio del departamento de horarios de ESIME Zacatenco, por su parte los objetivos específicos consistieron en: 1) Diseño y desarrollo de un directorio usando un servicio web que permita a un usuario acceder a la información (base de datos de los horarios de profesores) del directorio del departamento de horarios de ESIME Zacatenco; 2) Diseñar y crear una interfaz gráfica de usuario de la aplicación web; 3) Diseñar y crear la base de datos para la aplicación web; 4) Diseñar los servicios web que permitan consultar y modificar la base de datos.

El proyecto de investigación pretendió cubrir la necesidad de la falta de un medio que permitiera a los alumnos de ESIME tener información de sus profesores para poder localizarlos y colocar información relacionada con las actividades académicas, ya que los estudiantes de dicha institución experimentaban la necesidad de tener un medio o mecanismo de comunicación que permitiera conocer el horario de atención de los profesores de diferentes asignaturas.

Esta investigación se enmarca en el diseño y desarrollo de sistemas de información mediante la utilización de Web Services, y como tal presenta información relevante sobre los problemas típicos que tiene el desarrollador a la hora de diseñar un sistema que presente conexiones a base de datos y mecanismos interacción con el usuario; entre éstos problemas están: la reescritura del código de referente a las base de datos y problemas en la escalabilidad de la aplicación.

En el año 2009 Ponce S. Hugo C. realizó el trabajo de grado para obtener maestría, titulado “**ABCSIS: ARQUITECTURA BASADA EN COMPONENTES DE SOFTWARE PARA LA INTEGRACIÓN DE SERVICIOS**”, en la Universidad de Colima, Facultad de Telemática, Colima, Estados Unidos Mexicanos.

El objetivo general de esta investigación fue el de crear una metodología de desarrollo de software basado en SOA para el desarrollo del Sistema Integral Automatizado de Bibliotecas de la Universidad de Colima (SIABUC) y los objetivos específicos fueron: 1) Comprender el funcionamiento de los servicios Web y sus estándares XML relacionados; 2) Investigar acerca de la arquitectura SOA y su implementación con los servicios Web; 3) Entender el funcionamiento de los conceptos de SOA en el modelo de programación Windows Communication Foundation (WCF); 4) Realizar un análisis en SIABUC para identificar los servicios que pueden ser extendidos con la arquitectura propuesta; 5) Crear un prototipo tomando como base la arquitectura propuesta, el cual estará conformado por un conjunto de servicios; 6) Probar los servicios para detectar posibles fallas en la implementación posterior; 7) Invocar un servicio dentro de la aplicación prototipo.

La investigación consistió en la realización del diseño e implementación de servicios usando como arquitectura base SOA, tomando en cuenta las áreas de oportunidad más relevantes de las Bibliotecas de la Universidad de Colima, utilizando como prueba y demostración un prototipo que muestra la interacción entre el servicio de reservación, alojado en el servidor web para información de internet Internet Information Server (IIS), y el cliente fue desarrollado en lenguaje de programación PHP, para demostrar la independencia entre los lenguajes de programación.

Esta investigación presenta información importante referente a la descripción del funcionamiento de los sistemas basados en Web Services bajo la arquitectura SOA y se muestra su aplicabilidad en la resolución de un problema específico en este caso resolver problemas de automatización del sistema de reservación de las bibliotecas de la Universidad de Colima, México.

2.2 BASES TEÓRICAS

2.2.1 JAVA

Java generalmente hace referencia a la combinación de tres aspectos a la vez: el lenguaje de programación Java (un lenguaje de programación orientado a objetos y de alto nivel); la máquina virtual de Java o Java Virtual Machine (JVM), por su siglas en inglés, es una máquina virtual de alto desempeño que se encarga de ejecutar bytecode en una plataforma de computación específica; y la plataforma de Java que está conformada por una JVM que ejecuta bytecodes de Java compilados, que normalmente utilizan un conjunto de bibliotecas estándar como las que son provistas por la edición estándar o Java SE (Java Standard Edition) y la edición empresarial Java EE (Java Enterprise Edition), pero a pesar de que están acoplados por diseño, el lenguaje no implica la utilización forzada de la JVM y viceversa.

Java el lenguaje es un lenguaje de programación orientado a objetos influenciado de varias formas por C, C++ y Smalltalk, con ideas prestadas de otros lenguajes. Su sintaxis fue diseñada para ser familiar para aquellos que trabajaran con C, pero con principios a la orientación a objeto más robustos que los de C++, objetos fuertemente tipados, un sistema rígido para el manejo de excepciones y además posee un colector de basura o garbage collector, que permite liberar memoria mediante la eliminación de objetos que ya no se requieren en memoria.

Uno de los aspectos controversiales sobre Java ampliamente aceptados en el momento en que se creó Java pero que ha aumentado en el número de críticas es su incompleta orientación a objetos. Específicamente, las primitivas de Java como int, char, boolean, entre otros no son objetos, y requieren un tratamiento totalmente diferente por parte del desarrollador: como int no es una clase, no se puede declarar clases hijas ni nuevos métodos para él., para solventarlo se crearon las clases Integer que permitieron empaquetar ciertas funcionalidades adicionales.

A pesar de las críticas, Java es generalmente considerado como el lenguaje de computación multipropósito más popular de la actualidad. Es ampliamente usado para la programación empresarial, ya que permiten la ejecución en las plataformas operativas más grandes (Linux, Windows, Solaris y Mac).

2.2.1.1 ANOTACIONES

Las anotaciones, son una forma de metadato que proveen información de un programa que no es parte de la aplicación en sí, las anotaciones no tienen efecto sobre las operaciones del código que anotan. Las anotaciones poseen diversos usos, entre las cuales se pueden encontrar:

* Proveer de información al compilador, las anotaciones pueden ser usadas por el compilador para detectar errores o suprimir advertencias.
* Procesamiento en tiempo de compilación y de despliegue, las herramientas de software pueden procesar la información provista por las anotaciones para generar código, archivos XML, entre otros.
* Procesamiento en tiempo de ejecución, algunas anotaciones están disponibles y pueden ser examinadas durante el tiempo de ejecución de la aplicación.
* Permitir al programador declarar el comportamiento del programa dentro de su código fuente, lo que permite incluir características de la programación declarativa al lenguaje procedimental característico de Java.

Las Anotaciones en Java fueron presentadas en el 2002 y aprobadas en el 2004, éstas empezaron a estar disponibles a partir de la versión 1.5 del Java Development Kit (JDK) y la característica fue añadida oficialmente al lenguaje de programación a partir de dicha especificación.

La declaración de una anotación es una variación de las etiquetas, y toman la forma de una declaración de interfaz mediante la utilización de un carácter @ delante del nombre de dicha anotación.

2.2.1.2 SERVLET

Los Servlets son módulos escritos en Java que se utilizan en un servidor (pudiendo ser o no un servidor Web), con el objetivo de extender sus capacidades de respuesta a los clientes al utilizar las potencialidades de Java. Los Servlets son para los servidores lo que los applets para los navegadores, aunque los servlets no tienen una interfaz gráfica.

Los servlets pueden ser incluidos en servidores que soporten la Application Programming Interface (API) de Servlet. La API no realiza suposiciones sobre el entorno que se utiliza, como tipo de servidor o plataforma, ni del protocolo a utilizar, aunque existe una API especial para HTTP.

Los Servlets son un reemplazo efectivo para los Common Gateway Interface (CGI) en los servidores que los soporten ya que proporcionan una forma de generar documentos dinámicos utilizando las ventajas de la programación en Java como conexión a alguna base de datos, manejo de peticiones concurrentes, programación distribuida, etc. Por ejemplo, un servlet podría ser responsable de procesar los datos desde un formulario en HTML como registrar la transacción, actualizar una base de datos, contactar algún sistema remoto y retornar un documento dinámico o redirigir a otro servlet u alguna otra cosa. Entre las características más resaltantes de un servlet están:

* Manejo de sesiones, por lo que permiten hacer seguimiento a los usuarios a través de la creación y destrucción de sesiones.
* Utilización de cookies: los cookies son pequeños datos en texto plano que pueden ser guardados en el cliente (navegador), siempre mediante la utilización de la API de lo servlets.
* Programación en Java, se obtiene beneficios de la programación en este lenguaje como el acceso a JDBC, RMI, etc.
* Permiten la implementación de Web Services basados en la arquitectura SOA o el estilo de arquitectura Representational State Transfer (REST).

2.2.2 HTTP

El nombre HTTP es un acrónimo en inglés para Hypertext Transfer Protocol, y representa a un conjunto de reglas y lineamientos que permiten la transferencia de archivos (texto, imágenes en diversos formatos, sonido, video, y otros archivos multimedia) a través de la World Wide Web cuyo acrónimo es WWW, lo que implica que sea un protocolo de aplicaciones que se ejecuta sobre Transmission Control Protocol (TCP) e Internet Protocol (IP), que a su vez suelen denominarse familia de protocolos de internet o TCP/IP. Cada vez que un usuario común, abre su navegador Web, el usuario está haciendo uso del protocolo estándar HTTP de forma indirecta, lo que permite reconocer la importancia de dicho protocolo en el desarrollo de aplicaciones Web en la actualidad.

Los conceptos del HTTP incluyen la idea de que los archivos pueden contener referencias a otros archivos que cuya selección requeriría la transferencia adicional de solicitudes. Los servidores para aplicaciones Web contienen en adición a las páginas que sirven un componente HTTP, que representa a un programa que es diseñado para esperar solicitudes estandarizadas como el GET o POST, para proveerles de un servicio y manipular dichas solicitudes, a medida que éstas son recibidas.

Un navegador Web como Firefox, Chrome o Internet Explorer son en realidad clientes HTTP, que envían solicitudes a los servidores, por lo tanto cuando un usuario solicita una página web, el navegador construye una solicitud HTTP y la envía a través del protocolo IP el cual es provisto por la URL, a su vez el servidor recibe la solicitud envía de vuelta el recurso solicitado.

2.2.3 FORMATOS DE SALIDA

2.2.3.1 XML

El nombre XML proviene de las siglas Extensible Markup Language, que es un lenguaje jerárquico, basado en etiquetas al igual que HTML, con la diferencia de que XML es totalmente extensible, lo que permite describir contenido que es específico y versátil para cada aplicación haciéndolo de una forma estandarizada, mediante el cumplimiento de reglas y sin la necesidad de que los diseñadores del lenguaje hubieran diseñado ese contenido.

Entre las características más importantes de XML están:

* Los archivos XML son usados para datos estructurados, la data estructurada incluye libros de direcciones, parámetros de configuración, transacciones financieras, dibujos técnicos, entre otros. Para ello XML instituye un conjunto de normas (guías o convenciones) que permitan diseñar formatos para estructurar data. XML no es un lenguaje de programación y por lo tanto su uso o aprendizaje no se limita a los programadores. XML facilita la generación de datos, lectura de datos, y el aseguramiento que la data leída no es ambigua, evitando así las faltas típicas a partir del mismo diseño.
* XML es similar a HTML, manteniendo algunas diferencias; mientras HTML especifica que representa cada etiqueta y cada atributo, y usualmente como el texto entre ellos va a observarse en un navegador, XML usa las etiquetas solo para delimitar piezas de data, dejando el trabajo de interpretación de la data a la aplicación.
* Los archivos XML están estructurados mediante etiquetas que sirven para delimitar la data por lo que suelen ser un poco más voluminosos que sus representaciones binarias, siendo esta característica prevista por los diseñadores. El uso de etiquetas también provoca que los ficheros de este tipo sean más extensos que la competencia.
* Los archivos XML no requieren de licencia, ya que son un estándar de código abierto, además de que son independientes a la plataforma operativa y son soportados por muchos lenguajes de programación.

2.2.3.2 JSON

Un JavaScript Object Notation (JSON) es un formato de texto que facilita el intercambio de data estructurada entre todos los lenguajes de programación. JSON utiliza las llaves, corchetes, comas entre otros símbolos que son útiles para muchos contextos y aplicaciones. JSON fue inspirado por los objetos literales de JavaScript o EcmaScript como fue definido en la tercera edición de la especificación del lenguaje EcmaScript. Este formato no intenta imponer su sintaxis a otros lenguajes.

JSON desconoce acerca de los números, en cualquier lenguaje de programación puede existir un variado número de tipos con diferentes capacidades y características, de punto decimal, enteros o binarios, lo que puede dificultar el intercambio entre diferentes lenguajes de programación. Esta notación ofrece una representación de números que los humanos pueden comprender fácilmente, una secuencia de dígitos. Todos los lenguajes de programación saben cómo darle sentido a secuencias de números incluso si tienen desacuerdos en su representación interna.

Debido a la simplicidad del formato y que no se prevé cambios en la gramática de JSON, esto hace que sea una notación fundacional, tremendamente estable. El formato JSON fue presentado inicialmente al mundo a través de JSON.org en el 2001.

2.2.3.3 DATA SET

Un data set es una estructura de información correspondiente al contenido de una única tabla de base de datos, o una única matriz de datos estadística, donde cada columna representa una variable particular y cada fila corresponde a un dado miembro del conjunto de datos en cuestión. Un data set enlista valores para cada variable, así como el peso y tamaño de un objeto, cada miembro se suele conocer como dato.

Diversas características definen la estructura y las propiedades de un conjunto de datos, incluyendo el número y tipos de los atributos o variables, y algunas medidas estadísticas aplicables a ellos, como la desviación estándar. Los valores pueden ser números tanto reales o enteros, por ejemplo la representación de la altura de una persona en centímetros, pero también la fecha de nacimiento el cual sería otro tipo de datos, y así sucesivamente; mientras que la cantidad de elementos del data set representa la altura o tamaño de dicha estructura de datos.

2.2.3.4 CSV

Según el Shafranovich (2005), el formato Comma Separated Values (CSV) posee diversas especificaciones e implementaciones, pero no existe una especificación formal en existencia, lo que permite que exista una gran cantidad de interpretaciones a cerca del este formato.

https://dl.dropboxusercontent.com/u/6280514/ShareX/2015/07/2015-07-22_16-46-35.png

Figura 2.1 Ejemplo de un archivo csv

En la figura 2.1 se observa un ejemplo de este tipo de archivos y a continuación se muestran las normas más comunes para implementar un archivo de este tipo:

* Cada registro está localizado en una línea separada, delimitada por un carácter de fin de línea, que en inglés se denomina Carriage Return Line Feed (CRLF).
* El último registro en el archivo puede tener o no el carácter de fin de línea.
* También se puede utilizar una cabecera opcional que solo debe aparecer en la primera línea del archivo con el mismo formato de las demás líneas de registros. Esa cabecera puede contener los nombres que corresponden a los campos en el archivo y debe poseer el mismo número de campos que los registros en el resto del archivo, de lo contrario el archivo estaría malformado.
* Entre la cabecera y cada registro, debe existir uno o más campos, separados por comas. Cada línea debe contener al menos el mismo número de campos a lo largo del archivo. Los espacios son considerados parte del campo y no deben ser ignorados. El último registro no debe estar seguido por una coma.
* Cada campo puede o no estar rodeado por comillas dobles. Si los campos no están encerrados entre comillas dobles, entonces las comillas dobles podrían no aparecer dentro de los campos.
* Los campos que contienen caracteres de fin de línea, comillas dobles y comas deben estar encerrados entre comillas dobles.
* Si las comillas dobles son usadas para encerrar campos, entonces una comilla doble dentro del campo debe ser finalizada colocando otra comilla doble que la preceda.

2.2.4 SISTEMA DE GESTIÓN DE BASE DE DATOS

Un Sistema de Gestión de Base de datos (SGBD), consiste en una colección de datos interrelacionados y un conjunto de programas para acceder a los mismos.

Un SGBD permite el almacenamiento, manipulación y consulta de datos pertenecientes a una base de datos organizada en uno o varios ficheros. En el modelo más extendido (base de datos relacional) la base de datos consiste, de cara al usuario, en un conjunto de tablas entre las cuales se establecen relacione.

Las características más importantes de un SGBD son:

* El método de almacenamiento y el programa de gestión de datos (servidor) son independientes del programa desde el cual se hacen las consultas (cliente).
* En lugar de primarse la visualización de toda la información, el objetivo fundamental es permitir consultas complejas, cuya resolución está optimizada y expresadas mediante un lenguaje formal (SQL).
* El almacenamiento de los datos se hace de forma eficiente aunque oculta para el usuario y normalmente tiene poca relación con la estructura que los datos representan para el usuario.
* El acceso concurrente de múltiples usuarios autorizados a los datos, realizando operaciones de actualización y consulta de los mismos garantizando la ausencia de problemas de seguridad (debido a accesos no autorizados) o integridad (pérdida de datos por el intento de varios usuarios de acceder al mismo fichero al mismo tiempo).

El programa servidor suele activarse al arrancar la computadora y podría compararse a un bibliotecario que recibe peticiones (consultas) de diferentes programas clientes de base de datos, consulta la base de datos y entra al cliente el resultado de la consulta realizada. Si dos usuarios solicitan al mismo tiempo una modificación de los datos, el programa servidor se encarga de hacerlas ordenadamente para evitar perder datos.

El diseño de una base de datos implica codificar en formato digital ciertos aspectos del mundo real. Esta codificación implica tres fases de modelado, el modelo conceptual el modelo lógico y el modelo digital.

2.2.5 OBJETOS DISTRIBUIDOS

La idea de los objetos distribuido provino como una extensión del concepto de llamadas remotas a procedimientos, en un sistema de llamadas remotas como (Sun RPC, DCE RPC, Java RMI), el código es ejecutado remotamente a través de una llamada a un procedimiento remoto. La unidad de distribución es el procedimiento también conocido como función o método.

En un sistema para objetos distribuidos, la unidad de distribución es el objeto, por lo tanto, un cliente que importa algo permite al cliente acceder al objeto remoto como si éste fuera parte del programa original del cliente asegurando la transparencia para los clientes involucrados en el sistema. Un cliente en realidad importa una clase de java y ésta pasa a formar parte del conjunto de clases disponible en la máquina del cliente. Así por ejemplo, el cliente puede definir subclases a partir de la clase que importó, sobrecargar métodos, entre otras opciones.

Adicionalmente, los sistemas de objetos distribuidos poseen capacidades adicionales, como el descubrimiento de servicios, que permiten a los clientes localizar los objetos que requieren, también provee de servicios de seguridad, entre otros.

Algunas tecnologías de objetos distribuidos son:

* Modelo de objetos distribuidos comunes, Distributed Common Object Model (DCOM), desarrollado por Microsoft pero disponible en otras plataformas.
* Arquitectura común para solicitudes de objetos, Common Object Request Broker Architecture (CORBA), arquitectura definida por un consorcio industrial. CORBA se encuentra disponible para la mayoría de los sistemas operativos.

2.2.6 SOA

La arquitectura SOA es un enfoque usado para crear una arquitectura basada en el uso de servicios. Los servicios como REST (Transferencia de Estado Representacional) llevan a cabo algunas pequeñas funciones, como la producción de datos, validación de clientes o proveer de servicios de análisis de datos.

Adicionalmente a la construcción y exposición de servicios, SOA tiene la habilidad de aprovechar estos servicios muchas veces mediante aplicaciones (conocidas como aplicaciones compuestas). SOA une estos servicios de manera organizada o aprovecha un servicio de forma individual. Por lo tanto, SOA es realmente la fijación de arquitecturas existentes abordando la mayoría de los principales servicios como servicios y la abstracción de dichos servicios mediante dominios únicos que forman soluciones.

Una de las claves de la aceptación de SOA es que las interacciones ocurren con servicios débilmente acoplados y que operan de forma independiente. También la Arquitectura SOA permite la reutilización de servicios haciendo innecesario desarrollar desde cero la aplicación cuando es requerida una modificación, lo que es beneficioso para las empresas ya que garantiza el ahorro de tiempo y dinero.

Por lo tanto, al ser tan simple en su concepción SOA también en la mejor práctica para reparar arquitecturas rotas, con un amplio uso de estándares como los servicios WEB. Actualmente SOA está siendo promocionado como la mejor forma de proporcionar agilidad arquitectónica a las aplicaciones empresariales, siempre que se utilice SOA de forma a apropiada.

2.2.7 REST

REST fue propuesto por Roy Fielding en el año 2000. Evade la complejidad y los gastos de procesamiento característicos de los protocolos de servicios web mediante el uso del protocolo http. Un concepto relevante sobre REST es el de recurso que representa una pieza de información que tiene un único identificador llamado Uniform Resource Identifier (URI), en inglés. Como ejemplo se puede considerar un servicio de predicción del clima que se encuentra disponible de forma pública a través de http://www.weather.com, los recursos disponibles incluyen:

* El tiempo actual para el código postal 15213.
* Predicción del clima para mañana en la ciudad de Pittsburgh.
* Predicción del clima para los siguientes 10 días para el código postal 15213.
* Temperaturas promedio para el mes de octubre en la ciudad de Pittsburgh.

En este ejemplo, hay tres tipos de recursos: clima actual, predicción climática y temperaturas promedio. Siendo esto así, se puede estructurar la URI del recurso basado en estos tres tipos, por otro lado los parámetros puede ser representados por elementos en una ruta o patrón jerárquico o a través de los pares llave = valor, para estos ejemplos las URIs correspondientes podrían ser:

* http://www.weather.com/current/zip/15213
* http://www.weather.com/forecast/tomorrow/city/Pittsburgh
* http://www.weather.com/forecast/tenday/zip/15213
* http://www.weather.com/avg/city/Pittsburgh?month=10

REST se basa del protocolo http para permitir la interacción entre los usuarios y los proveedores del servicio. Este protocolo posee cuatro operaciones básicas: POST, GET, PUT y DELETE. Que en un diseño REST corresponden a las operaciones de crear, proporcionar, actualizar o borrar, que son comúnmente usados en los sistemas de información.

Por lo tanto si el usuario del servicio envía una solicitud POST a http://www.weather.com/current/zip/15213, lo que en realidad hace es solicitar al proveedor del servicio la creación de un dato con la información correspondiente al clima actual para la región correspondiente al código postal 15213; si el usuario envía una solicitud GET a la misma URI en realidad solicita que el proveedor del servicio proporcione la data en forma de respuesta para la zona geográfica denotada en la URI; una solicitud del tipo PUT indica que el proveedor del servicio debe remplazar su información con la información enviada con la data enviada durante la solicitud; una solicitud DELETE indica que el usuario del servicio quiere borrar dicha información.

El protocolo http también define los códigos referidos al estatus de la solicitud y pueden ser: 200 para OK, 201 para creado, 401 para acceso no autorizado, y así sucesivamente.

Una característica única de REST es que el predefine una interfaz uniforme, lo que quiere decir que el servicio es expuesto como fuentes de información a través de un conjunto de operaciones prediseñadas, en vez de un conjunto de métodos con diferentes parámetros. Por lo tanto en una solución REST para cada recurso se debe definir una representación. En la mayoría de los casos, un XML básico es el formato a utilizar, recordando que los servicios REST no poseen estados por lo que ellos no pueden guardar el estado de múltiples solicitudes de múltiples usuarios.

Los servicios Web y REST representan diferentes paradigmas de implementación de SOA. Uno se concentra en las operaciones a ser ejecutadas por el proveedor del servicio, mientras la otra se concentra en el acceso a recursos.

2.2.8 WEB SERVICES

Un servicio web también llamado Web Service en inglés es una interfaz que describe una colección de operaciones que son accesibles a través de una red mediante mensajería estandarizada XML. Un Web Service es descrito usando una notación estandarizada y formal de XML, denominada descripción del servicio, esta notación describe todos los detalles necesarios para interactuar con los servicios incluyendo formatos de mensajería (que detallan la operación), protocolos de transporte y ubicación.

La interfaz esconde los detalles de la implementación del servicio, permitiéndole que se le utilice independientemente de la plataforma de hardware y software en la que será implementada e independiente del lenguaje de programación mediante el cual el código fue escrito. Esto permite que las aplicaciones basadas en Web Services sean débilmente acopladas, orientadas a componentes e implementaciones de tecnologías cruzadas (es decir que ambas plataformas tecnológicas no requieren ser compatibles directamente).

La arquitectura para Web Services está basada en las interacciones entre tres roles diferente que son: el proveedor de servicios, el registro de servicios y el solicitante de servicios. Las interacciones envuelven la publicación, búsqueda y enlazamiento de operaciones. Juntos, estos roles y operaciones actúan a través de artefactos que incluyen al módulo y la descripción del Web Service.

En un escenario típico un proveedor de servicios hospeda un módulo accesible a través de una red, dicho proveedor define una descripción del servicio web y lo publica a un solicitante de servicios o un servicio de registro, el solicitante de servicios usa una operación de búsqueda para recuperar el servicio de descripción de forma local o desde el registro de servicios e invoca o interactúa con la implementación del Web Service y los roles del proveedor de servicios y del solicitante de servicios pueden exhibir características de ambos.

Los roles en una arquitectura de Web Service son:

* El proveedor de servicios: desde el punto de vista empresarial representa al propietario del servicio; desde la perspectiva arquitectónica es la plataforma que hospeda el acceso al servicio.
* El solicitante de servicios: desde un punto de vista empresarial este es un negocio que requiere ciertas funcionalidades que deben ser satisfechas, desde un punto de vista arquitectónico representa la aplicación que está buscando, invocando o inicializando una interacción con un servicio. El rol de solicitante puede ser jugado por un navegador manejado por una persona o un programa sin una interfaz de usuarios, como por ejemplo otro Web Service.
* El servicio de registro: Este es un registro de búsqueda para descripciones de servicios donde los proveedores de servicios publican sus descripciones de servicios

CAPÍTULO III

MARCO METODOLÓGICO

En este capítulo se estableció el esquema metodológico que fue usado en el trabajo de investigación, sabiendo que la metodología se encarga de estudiar las formas o maneras en las cuales un proyecto de investigación puede ser realizado, por lo tanto estudia todo lo relacionado con estrategias, métodos, procedimientos, tácticas y técnicas que pueden ser empleados durante el proceso de la investigación y en consecuencia para realizar los objetivos planteado, por lo tanto se dedicó el capítulo para identificar el tipo de investigación, diseñarla, describir sus fases y la metodología utilizada para su desarrollo y finalización.

3.1 TIPO DE INVESTIGACIÓN

La investigación científica es definida según Arias (2006, p. 22) como “un proceso metódico y sistemático dirigido a la solución de problemas o preguntas científicas, mediante la producción de nuevos conocimientos, los cuales constituyen la solución o respuesta a tales interrogantes”. Por otro lado el tipo de investigación, considerando a éste como el nivel o profundidad de la investigación, el mismo autor Arias (2006, p. 23) establece que “el nivel de investigación se refiere al grado de profundidad con que se aborda un fenómeno u objeto de estudio”.

En la actualidad existen numerosos trabajos documentados en los que se establecen diferentes tipos o niveles de profundidad de la investigación, los autores más reconocidos en esta área no enlistan de la misma forma estos niveles, por lo que no existe acuerdo entre ellos, una de las razones es que cada investigación es diferente y utiliza metodologías que difirieren en naturaleza y objetivos.

La categoría que se ajusta más a la metodología empleada para el desarrollo de esta investigación, es la proyectiva que fue definida por Hurtado (2010) como aquella que:

Consiste en la elaboración de una propuesta, un plan, un programa, un procedimiento, un aparato,…, como solución a un problema o necesidad de tipo práctico,…, en un área particular de conocimiento, a partir de un diagnóstico preciso de las necesidades del momento, de los procesos explicativos involucrados y de las tendencias futuras. … abarca el campo de la tecnología, pues esta aborda problemas prácticos, se centra en aplicaciones concretas, en dar respuesta a cómo hacer las cosas, inspirada en los procesos de investigación. (p. 567).

Adicionalmente Hurtado (2010, p.167) plante que: “Se pueden ubicar como proyectivas, las investigaciones que conducen a inventos, programas, diseños o creaciones dirigidas a cubrir una determinada necesidad… las investigaciones proyectivas se ocupan de la invención”.

Es importante mencionar que en un trabajo de investigación realizado por Parcero y Zambrano (2013, p.44) la investigación de carácter proyectiva está conformada por varias etapas, éstas están constituidas por la etapa del planteamiento del problema, la fundamentación teórica de la propuesta, la cual está comprendida por el establecimiento de actividades, la identificación de los recursos necesarios para su ejecución, la elaboración de un análisis y también las conclusiones sobre la viabilidad del proyecto y por último la fase de ejecución de la propuesta.

A su vez Hurtado (2010, p.584) reseña que “… la propuesta o el programa que se va a diseñar, no es el evento de estudio, por lo tanto no se operacionaliza como parte del proceso de construcción de los instrumentos”, por lo que en una investigación del tipo proyectiva deben seleccionarse el diseño de la investigación, escoger y describir las unidades de estudio, determinar las técnicas y construir los elementos de recolección de datos, así como establecer las fases de la investigación, dichas etapas serán tratadas en otros puntos.

3.2 DISEÑO DE LA INVESTIGACIÓN

En cuanto al diseño de la investigación los autores: Hernández, Fernández-Collado y Baptista (2006, p. 157) exponen que “con el propósito de responder a las preguntas de investigación planteadas y cumplir con los objetivos de estudio, el investigador debe seleccionar o desarrollar un diseño de investigación específico”. También plantean que: “El término diseño se refiere al plan o estrategia concebida para obtener la información que se desea” (Hernández, Fernández-Collado y Baptista 2006, p. 158).

Otro autor, en este caso Hurtado (2010, p. 691) plantea lo siguiente “el diseño de investigación es el conjunto de decisiones estratégicas que toma el investigador, relacionadas con el dónde, cuándo y cómo recoger los datos, y con el tipo de datos a recolectar, para garantizar la validez interna de su investigación”.

Al igual que con respecto al nivel de profundidad de la investigación, los autores y especialistas de las materias relacionadas con la metodología de la investigación, no tienen clasificaciones totalmente homogéneas y poseen definiciones divergentes y en algunos casos contradictorias, pero para encontrar un diseño que se adecuara al de esta definición se consultaron numerosos autores, y se analizaron las diferentes clasificaciones.

Según Hurtado (2010, p. 735) “los diseños multivariables son los que se ocupan del estudio de múltiples eventos de manera simultánea en la misma investigación”. Luego establece que estos diseños pueden ser de rasgo o de caso, por lo que se considera que este trabajo se acopla alde caso, debiéndose a que estos diseños “son estudios profundos o exhaustivos de una o de muy pocas unidades de estudios, cuyo objetivo es obtener un conocimiento detallado de ellas” (Hurtado 2010, p. 737).

A su vez Hurtado (2010, p.756) establece que en los diseños experimentales éstos “se aproximan a la investigación experimental, pero no cumplen con todas las condiciones de rigurosidad que tienen los diseños experimentales. En este tipo de diseño el investigador puede manipular la variable independiente, pero no hay selección al azar o rigurosa de la muestra”.

Contemplando las definiciones anteriores y la metodología de realización del sistema, se puede concluir que esta investigación queda enmarcada en las categorías de multivariable, y cuasi-experimental.

3.3 TÉCNICAS DE RECOLECCIÓN DE DATOS.

Sobre las técnicas de recolección de información Hurtado (2010, p. 771) expone que:

El proceso de recolección de datos requiere del empleo de técnicas e instrumentos que permitan acceder a la información necesaria durante la investigación. Las técnicas comprender procedimientos y actividades que le permiten al investigador obtener información necesaria para dar respuesta a su pregunta de investigación. Estas técnicas se pueden clasificar según el proceso utilizado para acceder a dicha información, y también con base en el área de conocimiento donde se aplican.

Existen numerosas técnicas o maneras en que los datos pueden ser recolectados una de las más importantes es la observación, que según Hurtado (2010, p. 834) “permite que el investigador se ubique en el marco de referencia de las personas observadas y tenga mayor acceso a su forma de ver el mundo. La observación directa se utiliza especialmente cuando el abordaje es endógeno”.

Por lo tanto la observación facilita la recolección de datos de manera íntegra ya que no requiere intermediarios; así pues luego del diseño y programación del sistema será necesaria la observación del comportamiento de éste con el objetivo de determinar si el funcionamiento del mismo cumple con las características especificadas en el planteamiento del problema.

3.4 FASES DE LA INVESTIGACIÓN

Para la realización de los objetivos específicos propuestos, se establecieron un conjunto de fases que abordan de forma sistemática la realización de los objetivos. La culminación de cada objetivo se logró culminando las fases de manera secuencial, realizando actividades que juntas lograban el cumplimiento de un objetivo específico. A continuación se presentan las fases de la investigación:

**FASE I: ANÁLISIS PRELIMINAR**

En esta fase de la investigación se contempla la definición del conjunto de características que debe poseer el sistema a desarrollar y se determinan las herramientas necesarias para el desarrollo del sistema.

Las actividades realizadas para la culminación de esta fase incluyeron:

* Determinar las características, prestaciones y funcionamiento que debe poseer el componente.
* Contemplar el diseño o utilización de un pool de conexiones como subcomponente que garantice la estabilidad del sistema y que sea lo suficientemente flexible y configurable para que el usuario lo pueda configurar fácilmente.
* Investigar el estándar que describe la forma en que los archivos XML son esquematizados.

**FASE II: DESARROLLO DEL COMPONENTE**

Esta fase de la investigación involucró la recolección de información bibliográfica y la programación de la totalidad de los componentes.

Las actividades realizadas para la culminación de esta fase fueron:

* Seleccionar el pool de conexiones a utilizar, analizando la robustez y las potencialidades del mismo.
* Diseñar la estructura del archivo XML que define el conjunto de bases de datos a las cuales se permitirá acceder por parte del sistema.
* Escribir el archivo con el conjunto de sentencias SQL que podrán ser ejecutadas, contemplando la posibilidad de variar los parámetros de la sentencia y la posibilidad de adicionar más sentencias SQL.
* Programar la clase que lee los parámetros que definen al pool de conexiones, escritos en el archivo XML que fue diseñado en la fase de análisis.
* Escribir la clase que inicializa el pool de conexiones a partir de los parámetros definidos en el archivo XML, contemplando todas las posibles configuraciones.
* Desarrollar la clase que escribe el resultado de la solicitud a base de datos, en los formatos: JSON.
* Integrar en el programa todas las clases definidas anteriormente, junto con los archivos de configuración.
* Definir la clase de iniciación del sistema mediante el uso un servidor embebido; dicha clase gestiona todos los procesos y facilita la utilización.

**FASE III: IMPLEMENTACIÓN Y PRUEBAS**

En esta fase de la investigación se integraron todos los archivos anteriores desarrollados y se implementaron en conjunto para luego realizar las diferentes pruebas para la evaluar el sistema desarrollado.

Las pruebas realizadas para la culminación de esta fase fueron:

* Pruebas de unidad.
* Pruebas de aceptación.
* Pruebas de integración.
* Pruebas de estrés.

CAPÍTULO IV

ANÁLISIS DE RESULTADOS

En el presente capítulo se describen de forma detallada los procedimientos y estudios realizados en el desarrollo de la investigación, así como las pruebas realizadas al sistema haciendo especial énfasis en el análisis a los resultados de dichas pruebas.

4.1 PLANIFICACIÓN

La metodología habitual para desarrollar aplicaciones, librerías y APIs, suele involucrar al cliente o al usuario interesado en utilizar la aplicación, como parte del equipo de desarrollo, tanto para el diseño, el desarrollo de la aplicación y para la obtención de una opinión objetiva sobre los resultados preliminares de cada iteración del proceso de desarrollo. No obstante, el presente trabajo de investigación no es realizado para cumplir las expectativas de un cliente, sino con objetivos investigativos, se decidió seleccionar a un desarrollador experimentado con experiencia en el diseño de aplicaciones empresariales, para servir de guía o asesor en las diferentes etapas de la investigación contribuyendo a la investigaciones a través de sus opiniones y sugerencias.

A pesar de que las metodologías manejan varias técnicas para cumplir con las diferentes de las etapas o fases del desarrollo, en este presente proyecto de investigación se optó por cumplir de forma secuencial con el conjunto de actividades señaladas en las fases de la investigación, dichos puntos serán explicados y desarrollados en puntos subsiguientes.

4.2 ANÁLISIS PRELIMINAR

4.2.1 CARACTERÍSTICAS, PRESTACIONES Y FUNCIONAMIENTO

En el presente punto se enumeran de forma sencilla las características preliminares del sistema. En primer lugar, el sistema está conformado por un endpoint basado en webservices, a través de una implementación del estándar SOAP, que sin el uso de dependencias externas adicionales, es capaz de conectarse a los siguientes cuatro manejadores de base de datos:

* MySQL.
* Oracle DataBase.
* PostgreSQL.
* SQL Server.

A su vez, el sistema tiene la capacidad de ejecutarse directamente desde la consola o terminal del Sistema Operativo, mediante la utilización de comandos de java, sin la utilización de contenedores o servidores web de terceros como Apache Tomcat, JBoss o Jetty; adicionalmente la aplicación debe poder ejecutarse en los sistemas operativos Windows y Linux.

Durante el inicio del sistema, éste se encarga de leer un archivo de configuración que posee indicadores que permiten inferir la información que se muestra a continuación.

* Rango de direcciones IP que pueden acceder al endpoint.
* Nombre del servicio del endpoint.
* Nivel mínimo de logs, que serán mostrados en el archivo log generado por la aplicación.

Así mismo, después de leer dicho archivo se debe leer otro, el cual debe proveer la información que determina la locación de las diferentes bases de datos a las cuales se va a conectar y acceder el endpoint, el diseño preliminar de este archivo implica que contenga la siguiente información:

* Identificador de la base de datos, este valor le permite a la aplicación y al usuario de la misma, determinar de forma exacta a cual base de datos se le enviarán las solicitudes.
* URL de conexión a base de datos, la cual es precisada por el JDBC de la aplicación, este dato determina en su estructura la IP donde está ubicada la base de datos, el puerto que está escuchando el sistema de gestión de base de datos, y el nombre de la base de datos.
* Nombre del usuario y palabra clave, ambos permiten que el endpoint se identifique y en consecuencia se conecte a la base de datos.
* Cantidad inicial de conexiones disponibles del pool de conexiones.
* Cantidad máxima de conexiones concurrentes (simultáneas) que el pool de conexiones va a manejar.

Luego de la lectura del archivo de conexiones, la siguiente fase en la secuencia consiste en leer un archivo de solicitudes, en el cual deben enumerarse las diferentes sentencias (queries) que se podrán hacer a cada base de datos, para ello se concibió que dicho archivo tuviera las características que se enlistan seguidamente.

* Sentencia SQL que represente la solicitud que será enviada a la base de datos, dicha sentencia debe estar en concordancia con el diseño de la tabla de la base de datos.
* Identificador de base de datos, de tal manera que el sistema pueda distinguir a cual base de datos se le asignará el listado de solicitudes.
* Identificador único para la solicitud, que permita al sistema reconocer cual solicitud será enviada.

Posteriormente, con toda la información provista anteriormente se crean conexiones hacia las bases de datos, seguidamente a la inicialización del sistema éste pasa a escuchar solicitudes de los clientes SOAP y a partir del nombre de los métodos ejecutados por los clientes y los parámetros enviados por éstos, se envían estructuras de información desde el servidor a los clientes.

A su vez, como el componente se encarga de realizar solicitudes a las bases de datos y de responder a dichas solicitudes en un formato en específico que depende del tipo de requerimiento efectuado por el cliente, se ideó en una estrategia que permitiera informar al cliente que las operaciones que fueron solicitadas se efectuaron correctamente (INSERT, UPDATE y DELETE) y que también facilitara el procesamiento de los datos provistos por la base de datos (SELECT), para ello las estructuras básicas de información que se seleccionaron para dicha tarea fueron:

* JSON
* DataTable

Es importante mencionar que cuando se hace referencia a un JSON, en realidad se hace referencia a una cadena de caracteres con dicha estructura, mientras que la clase de DataTable se asemeja de cierta forma a la clase ResultSet, pero garantizando la serialización de los datos contenidos en ésta.

Para poder representar de manera formal a dicha clase se muestra un diagrama de clases; según (Fowler & Scott, 1999) este tipo de esquema se convirtió en la técnica central de diagramación en la programación orienta a objetos, tanto así que además se ser ampliamente usada, también posee un gran rango de conceptos de modelado.

Profundizando en la explicación, (Fowler & Scott, 1999) establecen que un diagrama de clases describe los diferentes tipos de objetos que se utilizaron en el sistema, junto a los tipos variados de relaciones estáticas que existen entre las clases, cabe destacar que dichos autores establecen dos tipos de relaciones estáticas fundamentales:

* Asociaciones.
* Subtipos.

Adicionalmente, los autores (Fowler & Scott, 1999) exponen que, los diagramas de clase también deben mostrar los atributos y operaciones (métodos) de una clase, además de representar las restricciones que aplican en la forma en que los objetos están conectados.

A partir de esta información, se procedió a realizar el diagrama de clases para la clases Dispatcher (figura 4.1), DataTable (figura 4.2), y JSON (figura 4.3) mostrando las características más importantes incluyendo de las propiedades, métodos y mostrando algunas de las relaciones.

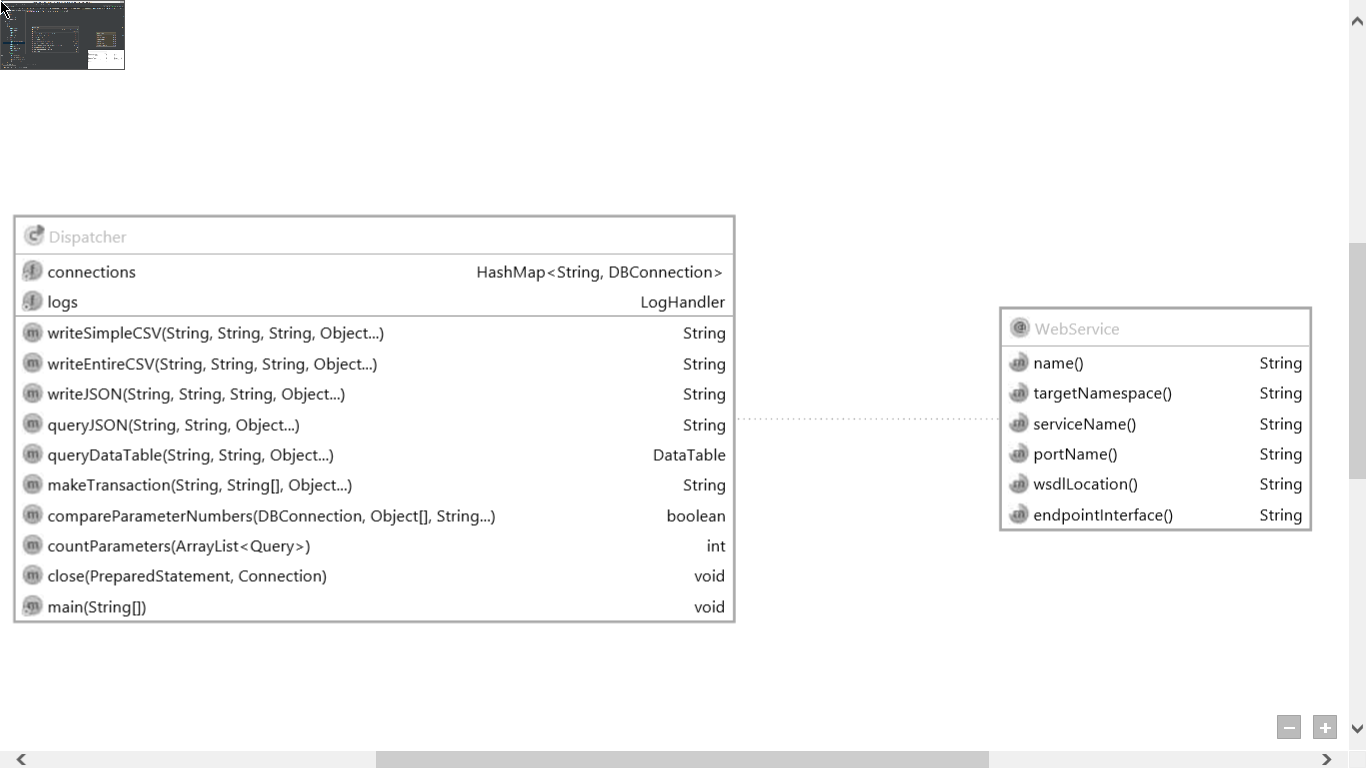


Figura 4.1 Diagrama de clase del Dispatcher (Dispatcher.java)

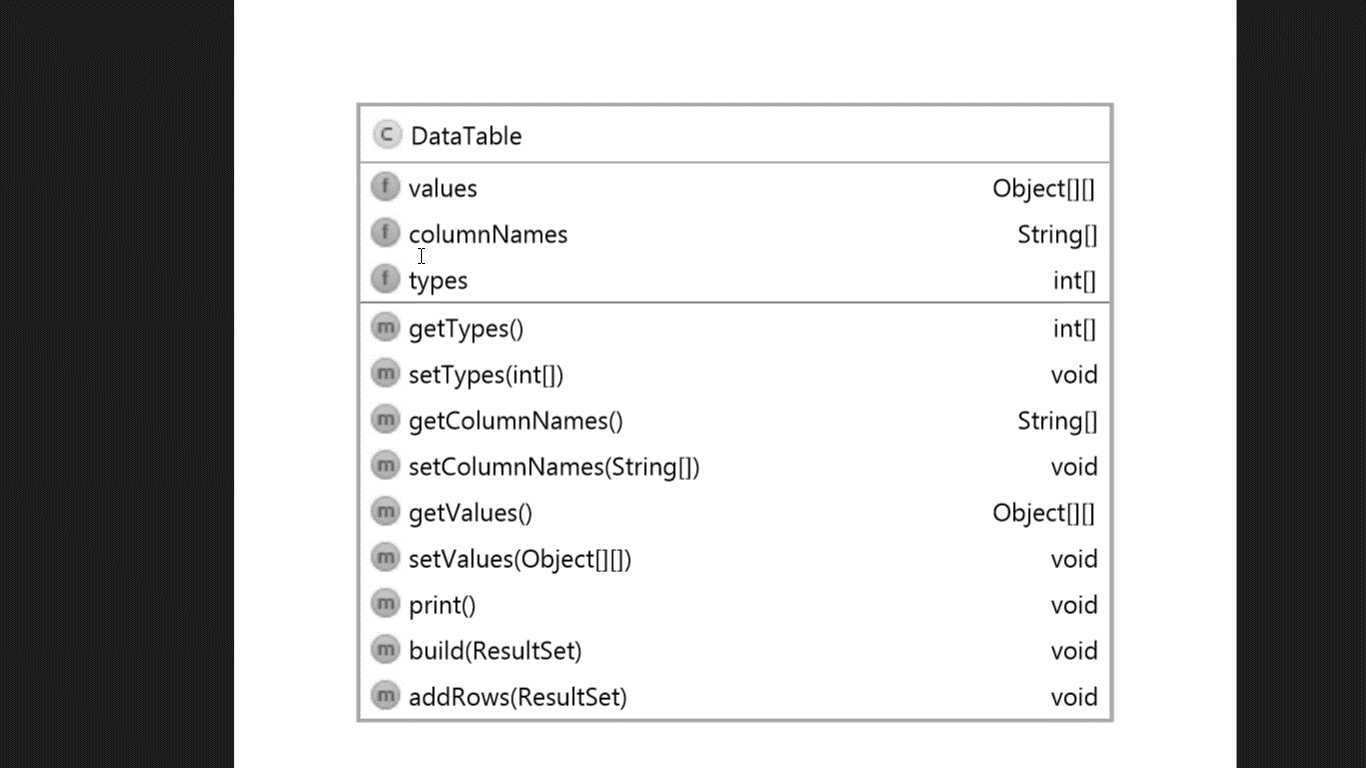


Figura 4.2 Diagrama de clase para DataTable (DataTable.java)

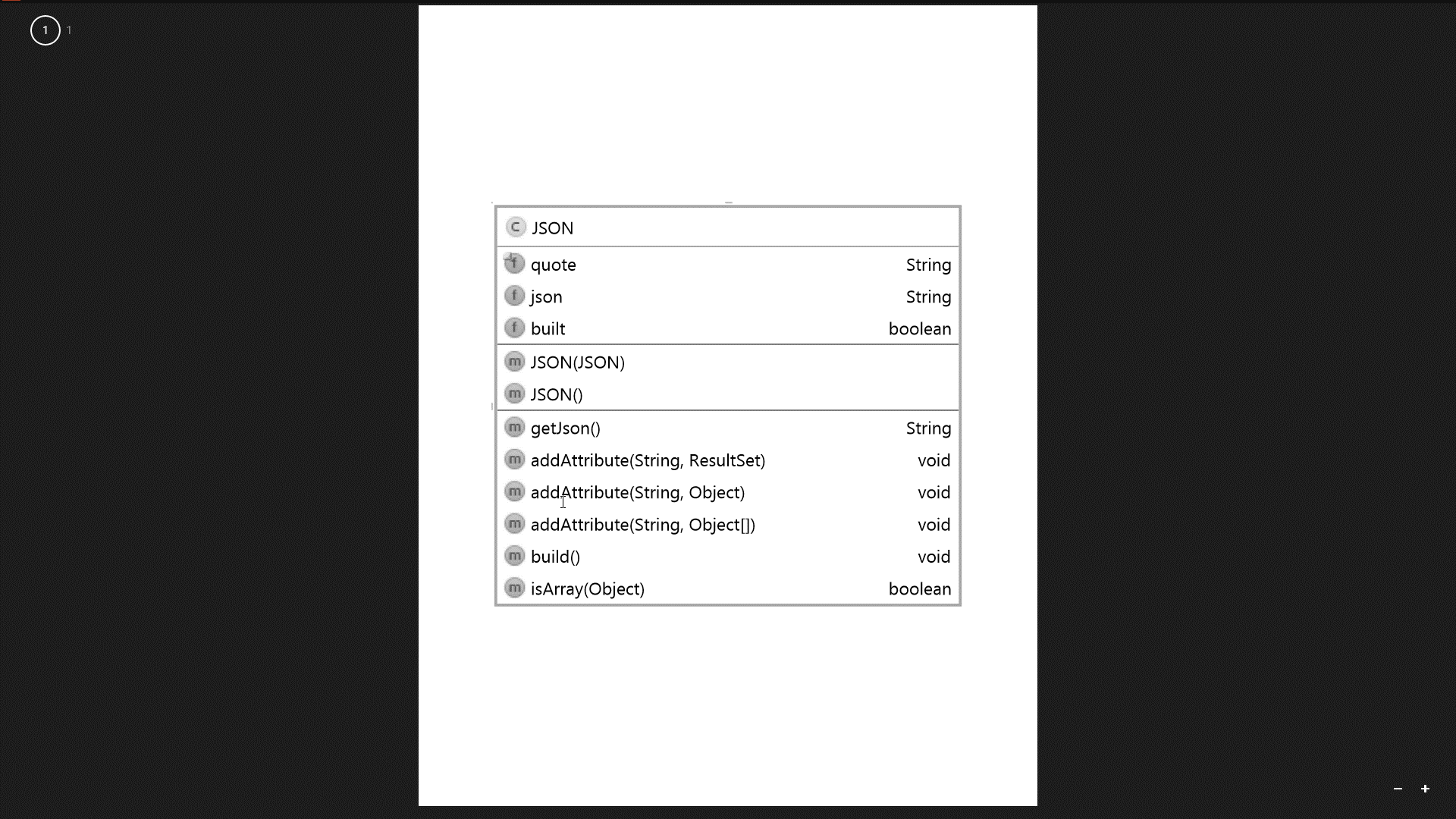


Figura 4.3 Diagrama de clase para JSON (JSON.java)

Adicionalmente a las funcionalidades anteriormente expuestas hay otras que aún no han sido descritas, específicamente otra característica que proporciona el sistema es la capacidad de escribir el resultado de una solicitud en los formatos CSV o JSON dependiendo de la necesidad del usuario, la escritura de estos archivos son provistas por las clases CSVWriter y JSONWriter, los diagramas de clases se muestran en las figuras 4.4 y 4.5 respectivamente.

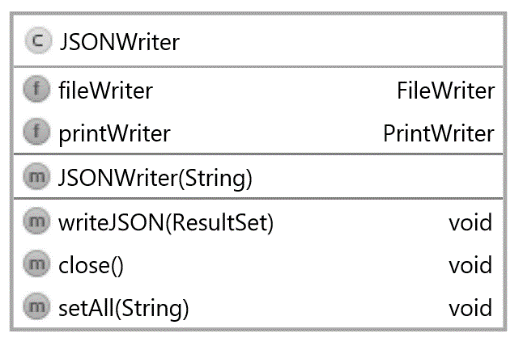


Figura 4.4 Diagrama de clase para JSONWriter (JSONWriter.java)

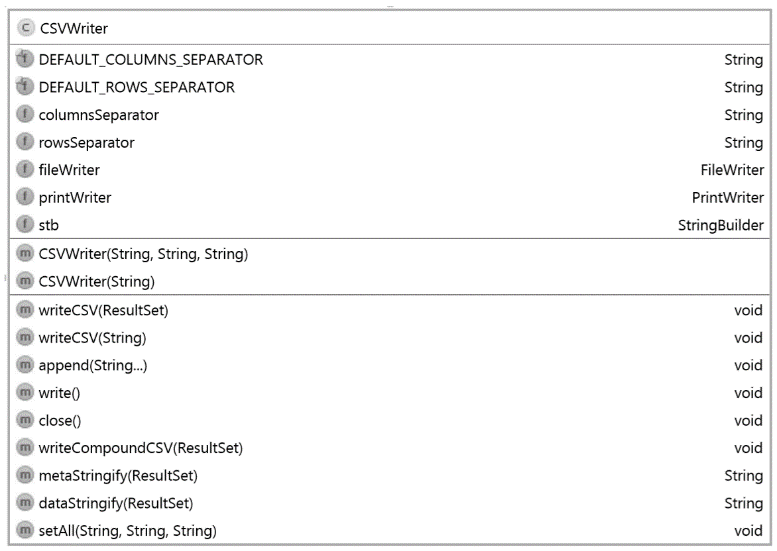


Figura 4.5 Diagrama de clase para CSVWriter (CSVWriter.java)

Algo esencial para el sistema es el manejo de errores, específicamente el sistema provee un sistema de log, el cual permite mantener un registro de los eventos históricos que han ocurrido dentro del sistema, dichos registros dependen del nivel mínimo de información que el usuario del sistema quiera mantener.

Con el objeto de proveer dicha funcionalidad, fue necesario diseñar e implementar un objeto que se encargara de esta tarea, dicho objeto se puede observar dentro de la clase Dispatcher como la propiedad logs (ver figura 4.1), y el diagrama de clase que describe las propiedades y métodos del objeto puede observarse en la figura 4.6.



Figura 4.6 Diagrama de clase de LogHandler (LogHandler.java)

Adicionalmente, una de las clases esenciales para el sistema es la que le permite mantener varias conexiones a bases de datos, esta funcionalidad está provista por el atributo connections de la clase Dispatcher (figura 4.1), el cual es un HashMap, lo que permite albergar muchas conexiones como valores (instancias de la clase DBConnection) utilizando como llave el nombre de la conexión base de datos, en la figura 4.7 se observa un detalle de su diagrama de clase de la clase DBConnection.

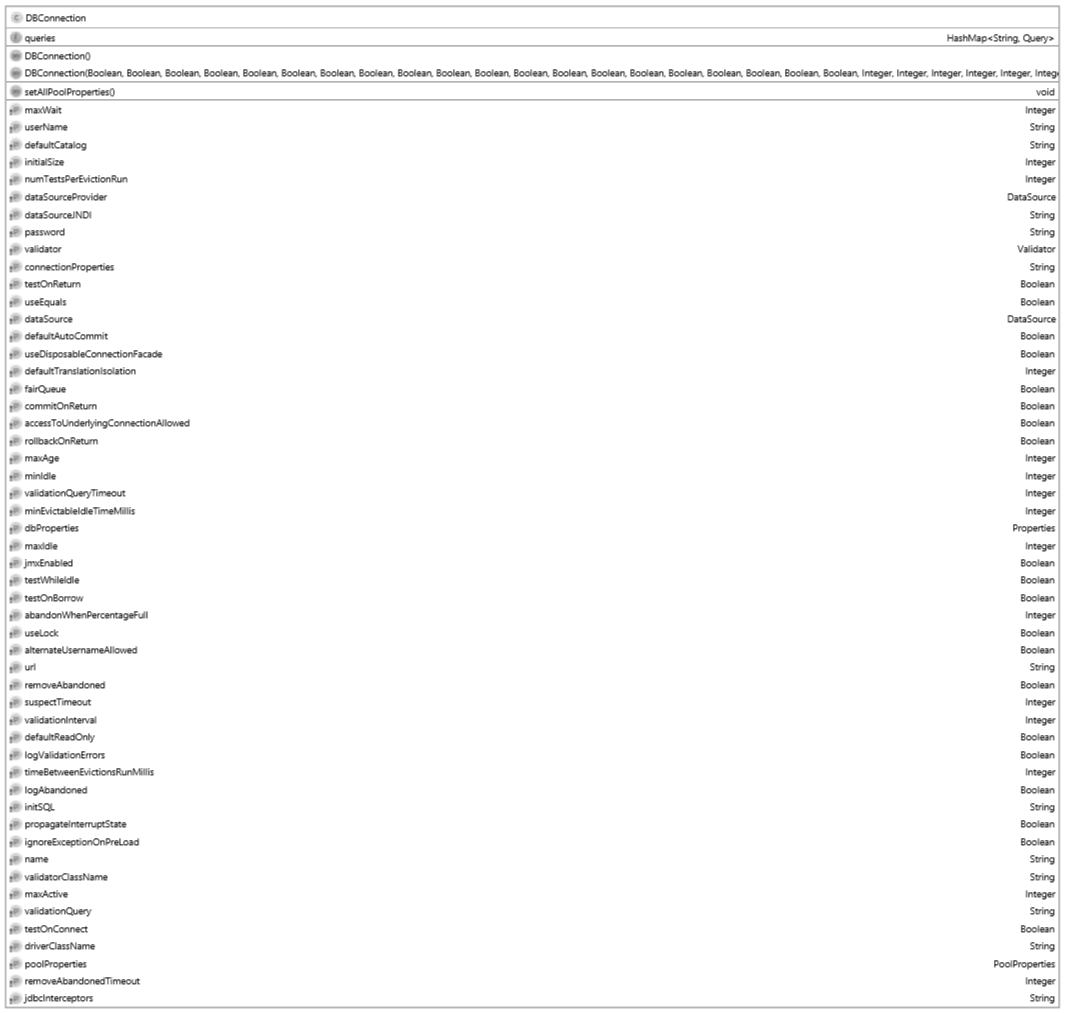


Figura 4.7 Diagrama de clase de DBConnection (DBConnection.java)

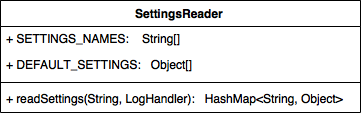


Figura 4.8 Diagrama de clase de SettingsReader (SettingsReader.java)

Además, hay otras clases relevantes, aquellas relacionadas con la lectura de los archivos que fueron definidos en el principio de esta sección, estas clases se encargan de leer las configuraciones del servidor, las conexiones de bases de datos y las sentencias SQL habilitadas para ser ejecutadas para cada conexión, los diagramas de clases están representados a través de las figuras 4.8, 4.9 y 4.10 respectivamente.

Así mismo hay otra clase que termina de proporcionar las funcionalidades faltantes, la clase Query (ver figura 4.11) que es utilizada dentro de QueriesReader, su función principal es la de envolver la sentencia SQL y facilita conocer la cantidad de parámetros que recibe una sentencia.

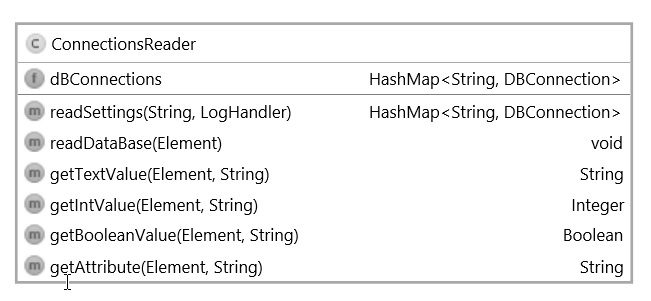


Figura 4.9 Diagrama de clase de ConnectionsReader (ConnectionsReader.java)

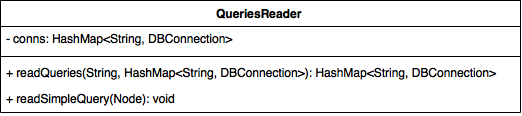


Figura 4.10 Diagrama de clase de QueriesReader (QueriesReader.java)

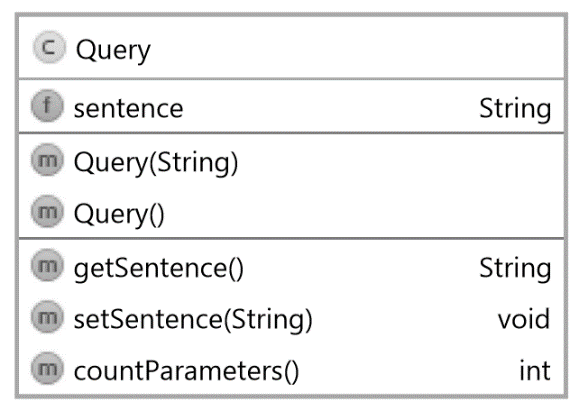


Figura 4.11 Diagrama de clase de Query (Query.java)

A partir de las clases individuales se construyeron los diagramas de clases por cada paquete del proyecto, en sí este tiene tres paquetes que se enlistan en la tabla 4.1. Asimismo, la representación de los diagramas de clases para los paquetes core, dependencies pueden observarse en las figuras 4.12, 4.13 respectivamente (se obvió el paquete server por su simplicidad), mientras que el diagrama de clases del sistema está representado en la figura 4.14 (éste no incluye las librerías estándar de java).

Tabla 4.1 Paquetes con sus respectivas clases

|  |  |
| --- | --- |
| Paquete | Clases |
| core | ConnectionReader  DBConnection  QueriesReader  Query  SettingsReader |
| dependencies | CSVWriter  DataTable  JSON  JSONWriter  LogHandler |
| server | Dispatcher |

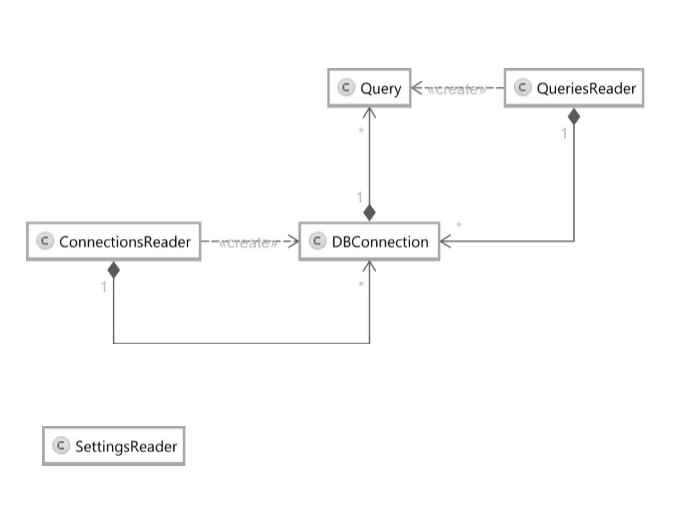


Figura 4.12 Diagrama de clase desde la perspectiva del paquete core

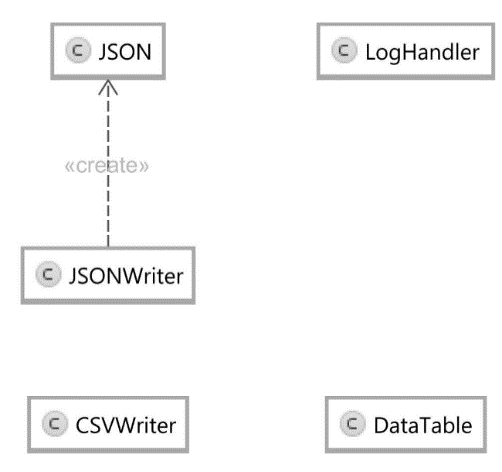


Figura 4.13 Diagrama de clase desde la perspectiva del paquete dependencies

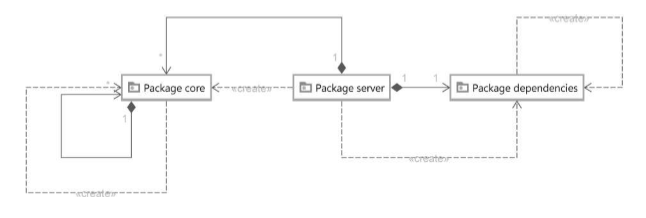


Figura 4.14 Diagrama de clase del sistema

A continuación se explica la manera en la cual el funcionamiento del sistema fue concebido, haciendo especial énfasis en las fases cruciales de la operación del sistema.

Una forma sencilla de representar el funcionamiento de un componente, es mediante la utilización de diagrama de actividades. Cabe destacar que dicha técnica de modelado está descrita en el estándar Unified Modeling Language (UML) lenguaje de modelado unificado en inglés.

Según (Fowler & Scott, 1999) esta técnica de modelado no tiene claros orígenes, pero combina ideas de muchas técnicas diferentes como lo son: los diagramas de eventos, la técnica de modelado de estados de SDL, el modelado de flujo de trabajo y las redes de Petri, a su vez estos diagramas son muy útiles para mostrar los distintos flujos de trabajo y en la representación del comportamiento de aplicaciones que tienen un alto nivel de procesos paralelizados.

A su vez, (Fowler & Scott, 1999) enfatizan la idea anterior exponiendo que, los diagramas de actividades describen la secuencia de acciones, manteniendo soporte para los comportamientos condicionales o paralelos, un diagrama de actividades es una variante de un diagrama de estados, pero donde los estados son sustituidos por el estado de las actividades.

Por otro lado, (Fowler & Scott, 1999) establecen que las actividades pueden ser fraccionadas en sub-actividades, esta característica es muy similar al funcionamiento de los diagramas de estado, donde existen súper-estados y sub-estados; otra ventaja de los diagramas de actividades es que permiten mostrar un súper-estado y su comportamiento interno, o utilizar un diagrama subsidiario permitiendo desacoplar el funcionamiento de ambas actividades y mostrar su funcionamiento de forma independiente.

En la figura 4.15 se presenta el diagrama de actividades que describe el arranque aplicación, dicho evento ocurre cuando se ejecuta el método main de la aplicación (ver figura 4.1), obsérvese que se encuentra en concordancia con las características enumeradas con anterioridad.

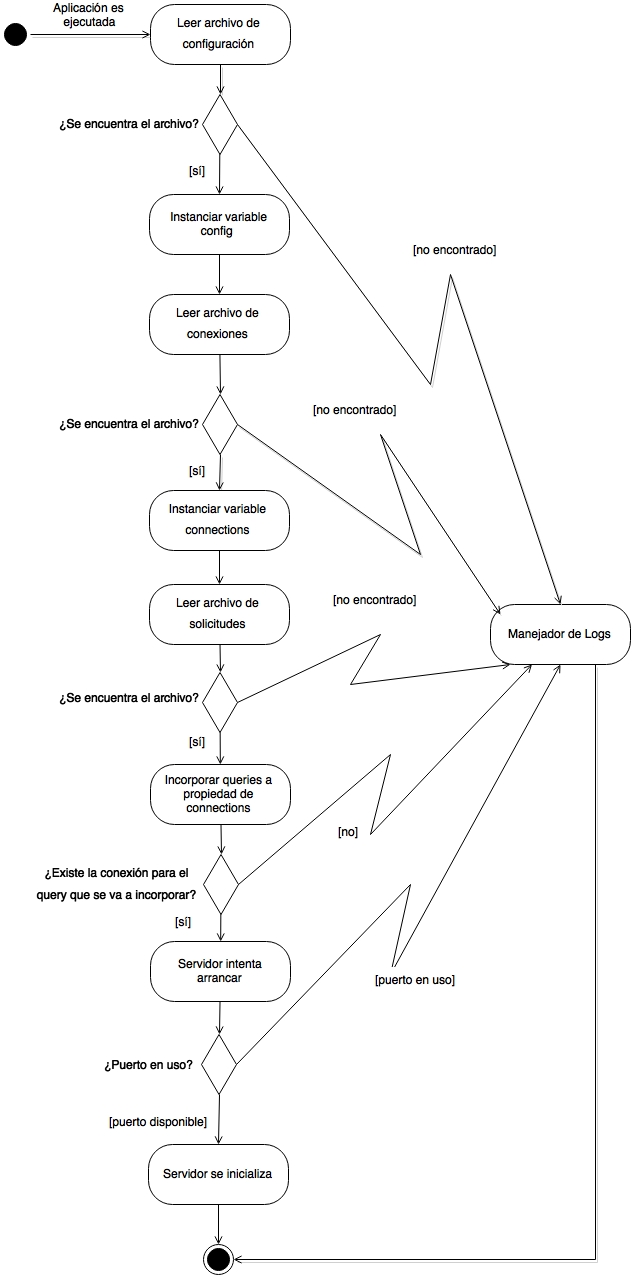


Figura 4.15 Diagrama de actividades para la actividad de arranque de la aplicación

Por otro lado, también es importante mostrar el funcionamiento de los métodos del servicio web que son ejecutados en el servidor al momento a partir de una llamada efectuada por un cliente del servicio web, para ello se realizó el diagrama de actividades para el método queryJSON del Dispatcher (ver figura 4.16), cuyo funcionamiento es muy similar al de los demás métodos disponibles del servicio.

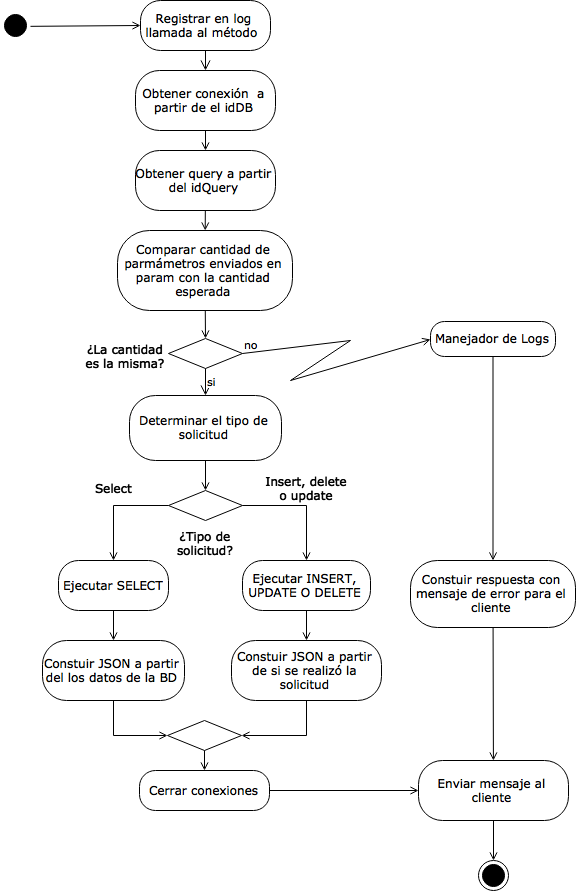


Figura 4.16 Diagrama de actividades para la el método queryJSON

Para poder representar la distribución de las partes que integran al sistema, se utiliza un diagrama de despliegue. Como explican (Fowler & Scott, 1999) un diagrama de despliegue muestra las relaciones físicas entre los componentes de software y hardware en un sistema, este tipo de diagrama es un buen lugar para mostrar como los componentes y objetos se encuentran enlazados, además de representar la manera en que se mueven a lo largo del sistema distribuido.

Adicionalmente (Fowler & Scott, 1999) exponen que cada nodo en un diagrama de despliegue representa algún tipo de unidad computación en muchos casos, es decir una pieza de hardware. El hardware puede ser un dispositivo sencillo o sensor, o un computador. En la figura 4.17 se presenta el diagrama de despliegue del sistema, donde se representa una simplificación de la interacción de los componentes que integran al sistema distribuido, en éste un servidor se presenta un servidor que funciona como despachador y que se conecta a múltiple servidores de base de datos y los otros dispositivos que se integran son los clientes del despachador.

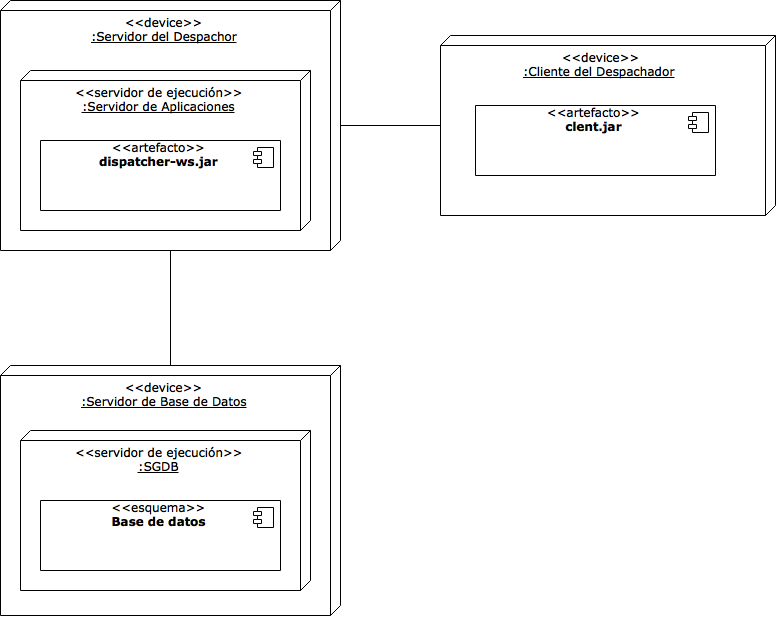


Figura 4.17 Diagrama de despliegue del sistema

4.2.2 EVALUACIÓN DE POOL DE CONEXIONES

En la presente sección se muestra la evaluación o análisis que fue efectuado, en el que se determinó si era imprescindible usar o no un pool de conexiones, y además si era necesario el desarrollo de uno o por el contrario si es más factible la utilización de uno ya desarrollado.

En relación con esquemas los sistemas empresariales para Sznajdleder (2013, p. 259) establece que:

En un esquema de tres capas físicas, es el server quien se ocupa de establecer la conexión con la base de datos. Los clientes se conectan al server e invocan sus servicios y es este [sic] quien, de ser necesario, accede a la base de datos para modificar y/o recuperar información. Como potencialmente son varios los clientes que podrían conectarse a la vez, si en el server manejamos una única conexión con la base de datos, esta [sic] será compartida entre los diferentes clientes que accedan concurrentemente.

Posteriormente Sznajdleder (2013, p. 260) señala que “La verdadera solución al problema de la concurrencia y la escalabilidad de la aplicación consiste en implementar un pool de conexiones”. Luego Sznajdleder (2013, p. 260) define lo que es un pool de conexiones estableciendo que:

Un pool de conexiones no es más que un conjunto de n conexiones preinstanciadas. Cuando un cliente necesita una conexión simplemente la pide al pool, la utiliza y luego la devuelve. Esa misma conexión, más tarde, podría ser “prestada” a otro cliente quien luego de utilizarla, también la devolverá.

El funcionamiento del pool de conexiones es análogo al funcionamiento de una biblioteca. La biblioteca puede tener 10000 socios pero no por eso va a tener 10000 ejemplares de cada libro.

Luego de analizar la información presentada anteriormente se reconoce la importancia de la utilización de un pool de conexiones para cualquier sistema en el cual exista interacción con bases de datos y por lo tanto ya que el sistema permite conectarse a múltiples bases de datos, se decidió incluir un pool al sistema.

4.2.3 ESTRUCTURAS XML

Aunque en el marco teórico se hizo referencia a los archivos XML algunos aspectos quedaron inconclusos, por ese motivo en esta sección se hace especial énfasis y se muestra en qué áreas fueron utilizados.

Sobre la estructura del formato XML, Sznajdleder (2013, p. 286) establece lo siguiente “XML es un lenguaje de marcas (o tags) extensible que permite categorizar y organizar información de diferentes maneras. Los tags definen la estructura y la organización del documento y contienen la información propiamente dicha”. Un ejemplo de dicha estructura puede observarse en la figura 4.18

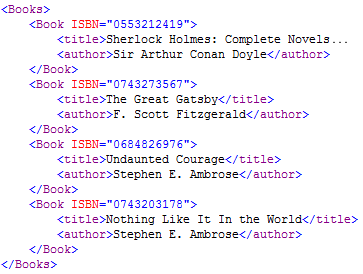


Figura 4.18 Estructura de un archivo XML

Por otro lado, al analizar meticulosamente la figura 4.18 se observa que la estructura es anidada en el que la etiqueta raíz de nombre Books engloba a tres etiquetas inscritas de nombre Book que poseen un atributo de nombre ISBN, y que internamente poseen dos etiquetas adicionales con los nombres title y author, todo con el objeto de representar información relevante para el programador. Esta estructura puede ser representada en forma de un árbol, como se observa en la figura 4.19.



Figura 4.19 Estructura de un archivo XML representado como un árbol

Posteriormente, Sznajdleder (2013, p. 286) señala que:

Llamamos “elemento”, “marca”, “etiqueta” o “tag” a cada palabra encerrada entre los signos “menor” y “mayor”. Los tags se abren para encerrar un determinado tipo de contenido (valores concretos u otros tags) y luego se cierran cuando aparece su nombre encerrado entre los signos “menor barra” y “mayor”…Un tag puede contener un valor concreto o puede contener subtags.

Luego, Sznajdleder (2013, p. 287) amplía su definición incluyendo la versatilidad de la estructura XML, como se presenta a continuaicón:

Como XML no está limitado a un conjunto de tags podemos crear tantos tags como necesitemos para describir y contener información de la mejor manera posible... Nuestro objetivo aquí es el de adquirir una mínima noción de XML para luego utilizar este “lenguaje” de tags en la estructuración y definición de los parámetros que volquemos en los archivos de configuración que utilicen nuestras aplicaciones.

Esta característica, es la permite que en el sistema a partir archivos con contenidos muy disímiles, puedan ser leídos si mantienen la estructura y en efecto terminen proporcionando funcionalidades similares al sistema, este tema será analizado a profundidad en otra sección.

4.3 DESARROLLO DEL SISTEMA

4.3.1 SELECCIÓN DEL POOL DE CONEXIONES

Después de haber comentado la teoría detrás del pool de conexiones en la sección 4.2.2, se procedió a analizar las dos alternativas.

La primera alternativa, consistía en la programación de un subsistema de pool de conexiones, esta opción aunque le hubiera proporcionado un valor agregado al trabajo de investigación, también hubiera aumentado el tiempo requerido para el desarrollo del mismo por más tiempo, además de haber requerido pruebas para certificar el funcionamiento apropiado.

La segunda opción, consistía en seleccionar un pool de conexiones ya desarrollado, implementado y probado, asimismo fuera lo suficientemente robusto para funcionar bajo alta concurrencia, también de código abierto y que además estuviera desarrollado en el lenguaje de programación JAVA o que permitiera la interoperabilidad con éste.

En el entorno de desarrolladores existen numerosas alternativas de pool de coenxiones, pero entre las dos más resaltantes están la librería “Commons DBCP” de Apache y la librería “JDBC Connection Pool” de Tomcat, se seleccionó la segunda opción por las razones que se señalan a continuación.

* “Commons DBCP” en su primera versión posee un hilo único. Y para ser seguro durante la ejecución de multitarea, la librería cierra el pool de conexiones por periodos de tiempo cortos, en los que la memoria se libera y se retornan los objetos.
* “Commons DBCP” puede ser lento. Como el número de unidades centrales de procesamiento y el número de hilos concurrentes que intentan obtener objetos (conexiones) crece, el desempeño de la librería sufre. Por lo que para sistemas altamente concurrentes puede ser significativo.
* “Commons DBCP” posee más de sesenta clases, mientras que la librería “JDBC Connection Pool” solo posee 8 clases, por lo consiguiente, las modificaciones que hagan a la librería de Tomcat implicarían menos modificaciones por parte de los desarrolladores.
* “Commons DBCP” utiliza interfaces estáticas, lo que significa que se deben utilizar las versiones correctas para una versión específica del Java Runtime Environment (JRE), o en cambio se podrían presentar excepciones del tipo “NoSuchMethodException”.
* No es pertinente reescribir alrededor de sesenta clases, cuando un pool de conexiones puede ser desarrollado con una implementación más sencilla.
* El pool de Tomcat implementa la capacidad de proveer conexiones de manera asíncrona, sin la incorporación de hilos adicionales a la librería en sí misma.

El listado completo de ventajas se puede hallar en la página web oficial de la librería ver (The Tomcat JDBC Connection, 2015).

4.3.2 ESTRUCTURA DE LOS ARCHIVOS XML

En la sección 4.2.1 se establecieron las características de los archivos de configuración, en sí son tres: settings.xml, connections.xml y queries.xml.

El archivo “settings.xml” posee una estructura muy simple, comprendida de una etiqueta raíz de nombre “settings”, que en su interior debe poseer cuatro subetiquetas con los nombres “ip”, “port”, “name” y “log” (ver figura 4.20), cada una de esas subetiquetas es imprescindible y tienen una función vital en la operación de inicio de la aplicación.

La subetiqueta “ip” determina el rango de direcciones IP que pueden acceder a los métodos remotos del WebService, por ejemplo cuando el valor de ésta es 0.0.0.0 significa que cualquier cliente puede solicitar los métodos del sistema, para detallar más, ver información delimitada por Raynolds y Poste (1994), la subetiqueta “port” es usada para indicar el puerto al cual van a estar atados los servicios y el usuario debe velar por evitar el solapamiento de servicios, a su vez la subetiqueta “name” establece el nombre del servicio web que va a ser inicializado, por último la subetiqueta “log” indica el grado mínimo de información que el sistema de logs registrará en el disco.

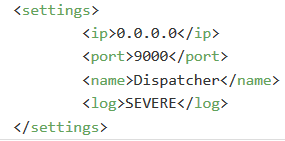


Figura 4.20. Archivo settings.xml

Por otro lado el archivo connections.xml es mucho más complejo ya que contiene información mucho más detallada que está relacionada con el modo en el que el pool de conexiones que fue seleccionado define cada una de las conexiones, en la figura 4.21 se puede observar la estructura mínima requerida.

Primeramente ésta posee una etiqueta raíz denominada connections y ésta posee varias subetiquetas, la subetiqueta sgbd indica el manejador de base de datos a la cual la base de datos pertenece, “url” representa la cadena de conexión para la base de datos, “driverClassName” determina el driver que el sistema utilizará para realizar solicitudes a la base de datos, “userName” y “password” hacen referencia al nombre del usuario (junto a su clave) que posee permisos para acceder a la base de datos, y las últimas dos subetiquetas indican algunas características del pool de conexiones, específicamente “maxActive” limita la cantidad a un máximo de cien conexiones concurrentes, mientras que “initialSize” indica el número de conexiones disponibles iniciales.



Figura 4.21. Archivo connections.xml

Adicionalmente, sobre el archivo “connections.xml”, cabe destacar que hay muchas más subetiquetas que son de carácter opcional (ver figura 4.22) y que le proveen funcionalidades que son proporcionadas por la librería de pool de conexiones, más detalles sobre estas funcionalidades pueden encontrarse en la página oficial del pool de conexiones ver (The Tomcat JDBC Connection, 2015).



Figura 4.22. Archivo connections.xml con etiquetas opcionales

Por último, el archivo queries.xml es bastante sencillo, contiene las sentencias SQL que se podrán ejecutar a cada base de datos de manera individual, una muestra de la estructura de este archivo se presenta en la figura 4.23.

Analizando la estructura del archivo queries.xml se observa que posee una etiqueta raíz “queries”, que posee una cantidad indeterminada de subetiquetas de nombre “db” que posee un atributo id, el cual permite identificar la base de datos a la cual se está haciendo referencia, a su vez cada subetiqueta “db” posee otras de nombre “query” que también poseen un id el cual identifica de manera única (dentro de la subetiqueta) a cada sentencia, y el valor encerrado es la sentencia SQL.

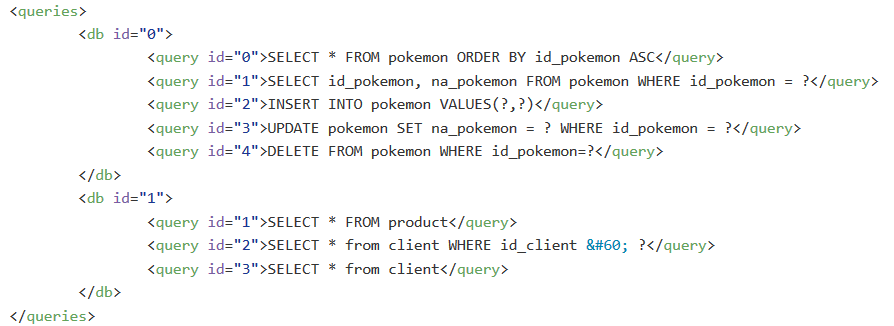


Figura 4.23. Archivo queries.xml

4.3.3 DESARROLLO DEL SISTEMA

En esta sección se hace referencia a la forma en que fue desarrollado el sistema. Como fue sutilmente reseñado en secciones anteriores, la aplicación fue desarrollada bajo el lenguaje de programación Java en su versión 8, utilizando la API de java para Web Services XML (JAX-WS) en la versión 2, utilizando el entorno de desarrollo IntelliJIDEA en su versión 14, como sistema de control de versiones se utilizó Git empleando como repositorio remoto Github en su modalidad gratuita, ya que la aplicación es de código abierto.

Con respecto al desarrollo, hay que recordar que los diagramas de clases al igual que los diagramas de actividades fueron reseñados en la sección 4.2.1, y debido a que el código de la aplicación es muy extenso, se decidió colocar solo fragmentos importantes del código. Hay que aclarar que el código en su totalidad está en el repositorio remoto albergado por Github en la dirección que se muestra a continuación <https://github.com/Bonsanto/easy-db-dispatcher>.

La clase principal es la clase “Dispatcher” (ver figura 4.24), esta es la clase que permite el arranque de la aplicación a través de la ejecución de su método “main”, también posee los diferentes métodos que pueden ser invocados remotamente. Para obtener esta capacidad, se determinó que la clase debía ser un servicio web, y se logró mediante la utilización de anotación “@WebService()” al principio de la clase, que junto a la utilización de la anotaciones “@WebMethod” sobre los métodos, indicaron que dichas operaciones pueden ser llamadas o invocadas remotamente por algún cliente del servicio.

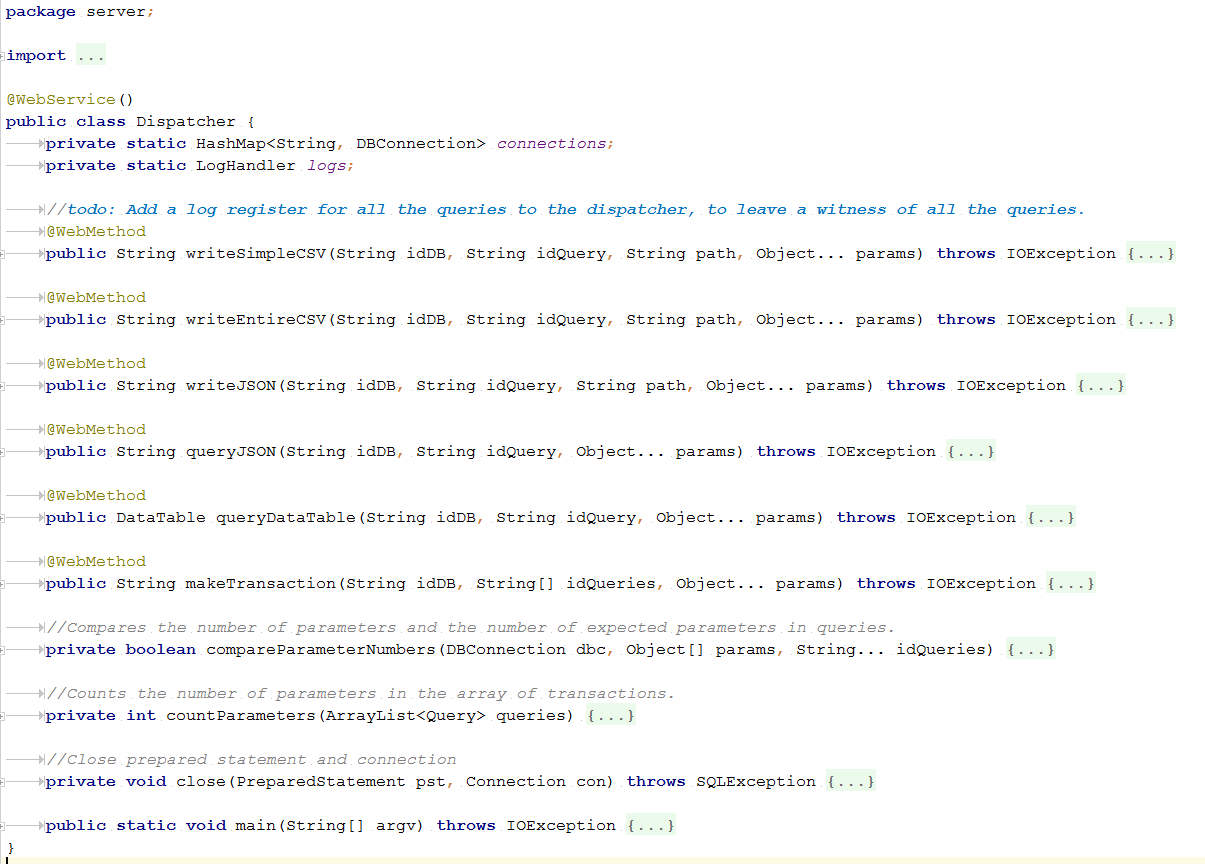


Figura 4.24. Fragmento de la clase Dispatcher

Para comprender como la aplicación se inicia, hay que ver el contenido del método “main” (ver figura 4.25), primeramente se observa que el método puede lanzar una excepción del tipo “IOException”, esto se debe a que el arranque del sistema involucra la lectura de tres archivos de configuración, archivos que son indispensables, así pues en caso de no hallarlos o no tener permisos de lectura para los mismos la aplicación no debería arrancar (ver figura 4.15).

Es importante mencionar que los archivos de configuración fueron dispuestos dentro de la carpeta “settings”, debido a que el sistema de archivos para Windows y UNIX utilizan separadores diferentes fue necesario pensar en una herramienta para captar el referido símbolo, esto se logró mediante la utilización de la clase estándar File.

Por otro lado, para captar la información registrada en los archivos, se emplea el objeto “settingsReader” para instanciar el objeto “config”, que después es usado para formar la cadena de caracteres que simboliza la URL a la que puede acceder el cliente para obtener los archivos wsdl y también involucra el puerto donde el servicio queda escuchando.

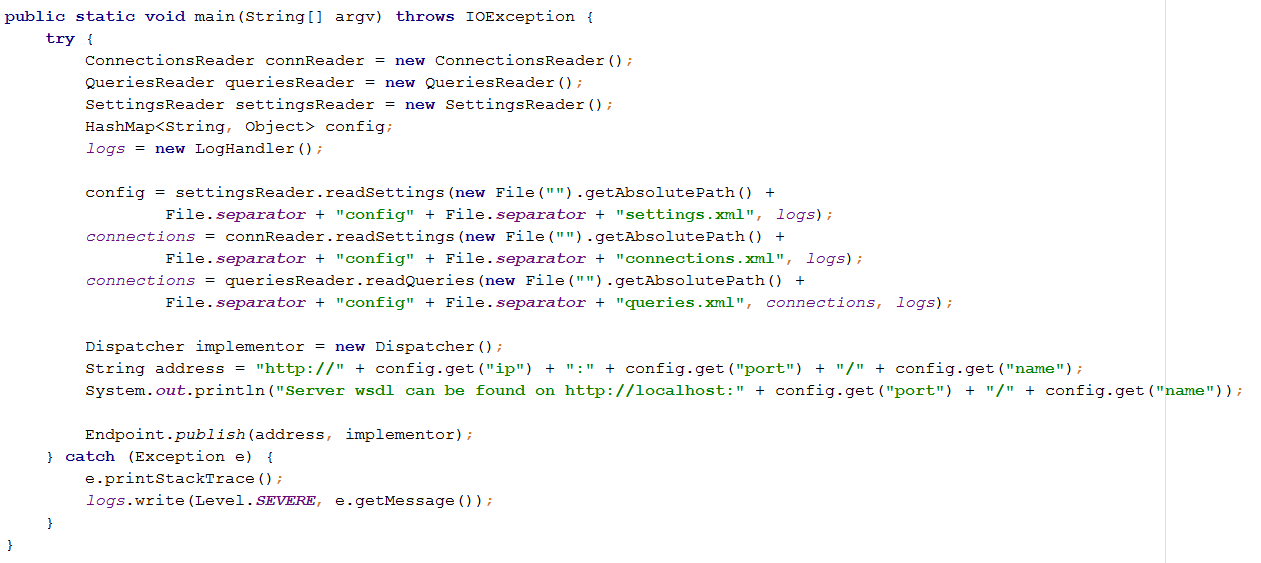


Figura 4.25 Fragmento del método main de la clase Dispatcher

Adicionalmente, la clase Dispatcher posee dos atributos: connections y logs, el primero encargado de contener las bases de datos a las cuales se puede conectar, de allí se observa que el objeto es un mapa cuya llave es el identificador de la conexión del tipo String y el valor es un objeto de la clase DBConnection.

Debido a que la clase DBConnection es sumamente extensa (posee más de seiscientas líneas de código), solo se hizo referencia a las características más resaltantes en la figura 4.26, eliminando muchas propiedades, métodos y constructores.

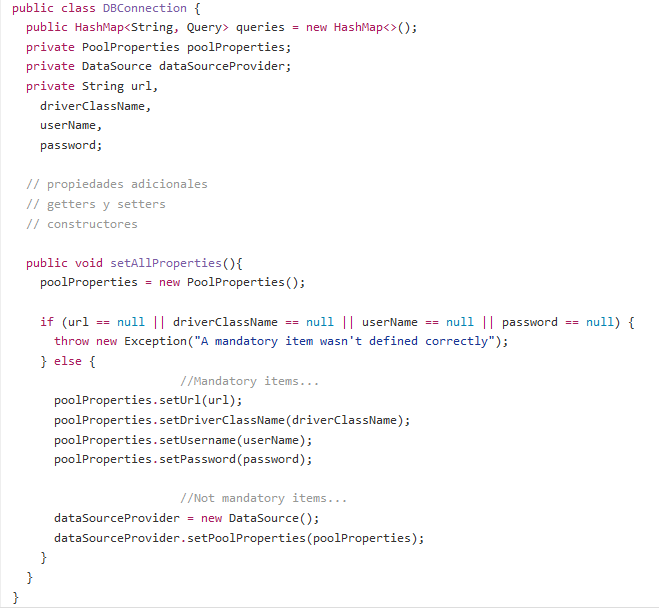


Figura 4.26 Fragmentos de la clase DBConnection

La clase DBConnection posee cuatro propiedades indispensables, que coinciden con las subetiquetas requeridas del archivo connections.xml (establecidas en la sección 4.3.2), pero el núcleo del funcionamiento está en las otras dos propiedades poolProperties y dataSourceProvider, la primera es la que se encarga de configurar la librería de pool de conexiones a partir de la información suministrada por el archivo connection.xml, y la segunda en realidad es el pool de conexiones el cual se instancia a partir de la información del objeto poolProperties.

Otra clase de suma relevancia es la cual permite convertir el resultado de una solitud a base de datos a una estructura del tipo JSON, esta labor es ejecutada por la clase JSON que se muestra en la figura 4.27.

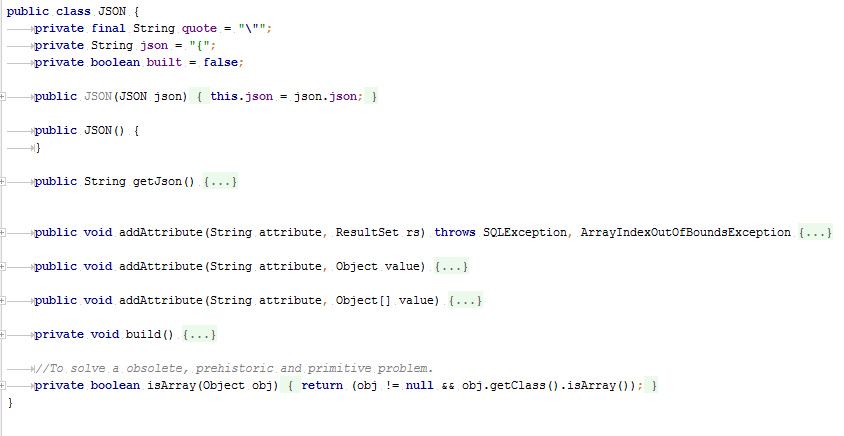


Figura 4.27 Fragmentos de la clase JSON

Como el lenguaje de programación Java no usa de forma nativa los objetos de JavaScript (JSON) se tuvo desarrollar un mecanismo para proveer una funcionalidad similar, desarrollando un analizador que convertía el resultado de la solicitud a una cadena de caracteres, para poder permitir el recorrido del resultado fue necesario colocar esta opción disponible.

4.3.4 PRUEBAS AL SISTEMA

4.3.4.1 PRUEBAS DE UNIDAD E INTEGRACIÓN

La realización de pruebas de unidad se estableció como objetivo específico, por lo tanto es importante conocer en qué consisten. En una prueba unitaria según Barrientos (2014, p. 1):

...se trata de descubrir errores nuevos o resultados de modificaciones a los componentes lógicos más pequeños ya existentes. Esta opción es la más económica ya que el costo de encontrar errores a medida que se avanza en las etapas del desarrollo de un proyecto es cada vez mayor... Además es necesario asegurar cierto grado de correctitud [sic] al nivel más pequeño para poder proceder con mayor confianza hacia las pruebas de integración funcional.

Posteriormente sobre las pruebas de integración, Barrientos (2014, p. 1) complementa su definición estableciendo que:

…en el caso de la programación orientada a objetos, el componente funcional más chico es la clase en particular, cada uno de sus métodos. Las pruebas de unidad se centran por lo tanto en estos componentes.

Partiendo de estas definiciones, se comprende que las pruebas de unidad son efectuadas de forma iterativa y a cada módulo, de tal manera que se garantice que las piezas del software funcionen bien de forma separada, reduciendo la probabilidad de hallar errores al final del desarrollo del sistema o durante su integración, con la subsiguiente disminución de la magnitud del costo de la reprogramación que habría que realizar al hallar un error en un módulo a la de todo un sistema.

También es importante conocer el funcionamiento de las pruebas de integración, sobre las pruebas de integración, Rouse (2008, traducción propia) plantean lo siguiente:

…es un proceso empleado durante desarrollo de software en el cual unidades del programa son combinadas en grupos y probadas de diferentes formas. En este contexto, una unidad se define como la porción más pequeña de una aplicación que puede ser testeada. La prueba de integración puede exponer problemas de las interfaces a lo largo de los componentes del programa, mucho antes que el problema ocurra durante la ejecución de un programa del mundo real…

Cada módulo y clase fue probado de forma separada durante la etapa de desarrollo, utilizando diferentes valores y parámetros para evaluar el funcionamiento correcto. La prueba de integración se efectuó al finalizar el desarrollo del componente, evaluando que el desempeño de la aplicación fuera estable, funcional y con un desempaño aceptable al ponerse a prueba.

4.3.4.2 PRUEBAS DE ESTRÉS

Las pruebas de estrés son pruebas que se hacen en diversos ámbitos profesionales, desde la ingeniería mecánica hasta la ingeniería de computación, teniendo implicaciones similares en algunos casos pero diferentes en otros, en el área del de software Molyneaux (2009, p. 39, traducción propia) establece que:

Una prueba de estrés provoca que la aplicación o que alguna parte de la infraestructura de soporte falle. El propósito de ésta es la de determinar los límites máximos y la capacidad de la infraestructura. Por lo tanto, una prueba de estrés continúa hasta que algo se rompa: los usuarios no puedan entrar al servicio, los tiempos de respuesta exceden los valores definidos como aceptables, o la aplicación pase a estar indisponible…los resultados de una prueba de estrés miden tanto la capacidad como el desempeño. Es importante conocer los límites máximos, en particular si se el crecimiento en el tráfico de la aplicación es impredecible.

Para poder ejecutar las pruebas de estrés que permitiera simular varios clientes utilizando los servicios del sistema de manera concurrente, fue necesario diseñar un sistema de prueba conformado por dos aplicaciones de prueba interdependientes, éstas se pueden hallar en detalle en Github <https://github.com/Bonsanto/easy-db-dispatcher>, específicamente dentro de las carpetas “test-client” y “test-starter”.

La primera aplicación “test-client”, consistió de un endpoint, el cual se encarga de ejecutar un conjunto de operaciones al sistema, se puede notar que al ser un servicio web este requeriría la URL que contiene tanto la ubicación del sistema, para proporcionarla se adicionó un archivo en formato de nombre endpoint.properties con dicha información (ver figura 4.28).

https://dl.dropboxusercontent.com/u/6280514/ShareX/2015/07/2015-07-22_20-40-17.png

Figura 4.28. Contenido de muestra del archivo endpoint.properties

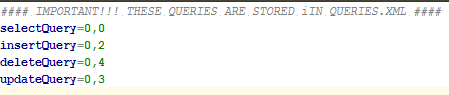


Figura 4.29. Contenido de muestra del archivo queries.properties

Adicionalmente para permitir cierta versatilidad en cuanto a las operaciones que la aplicación de prueba podría efectuar, se incorporó otro archivo de nombre queries.properties que contenía la información de cada operación que se realizaría al sistema, obsérvese que cada propiedad tiene dos valores separados por coma, el primero identifica a la base de datos y el segundo a la sentencia SQL que se efectuaría (ver figura 4.29).

Un fragmento del contenido de la clase Test se presenta en la figura 4.30, esta clase posee un método de nombre “startTest”, el cual puede ser ejecutado remotamente, siendo esta la característica que permite que el conjunto de pruebas dispuestas en su interior pueda ser ejecutado por un operador externo, específicamente el cliente de la aplicación “test-starter”.

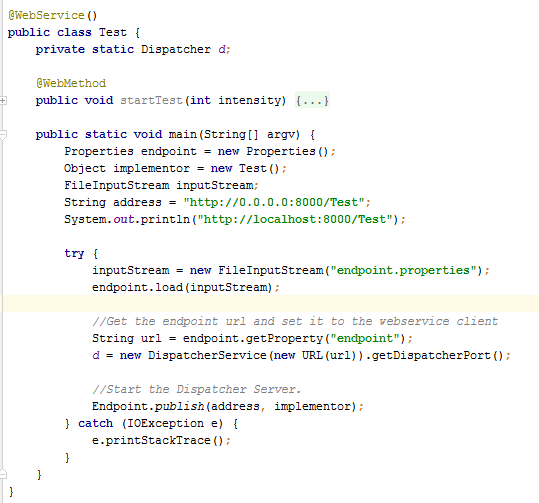


Figura 4.30. Fragmento de la clase Test

La segunda aplicación “test-starter”, consistió de dos clases principales “RequestThread” y “Starter”, la primera fue diseñada para permitir múltiples hilos (garantizando la múltiple concurrencia durante el momento de la prueba) y la segunda consistió en una clase con un “main” el cual contiene las llamadas a los nodos que realizarán la prueba (ver figura 4.31), al igual que en la otra aplicación fue necesario utilizar un archivo que contuviera la URL de todos los nodos que realizarían la prueba, para ello se escribió el archivo “endpoints.properties” el cual puede ser detallado en la figura 4.32.

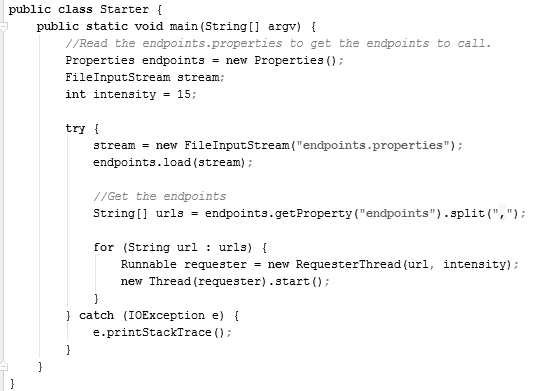


Figura 4.31. Fragmento de la clase Starter

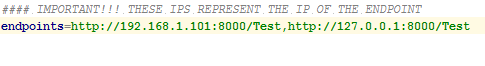


Figura 4.32. Contenido de muestra del archivo endpoints.properties

La prueba fue realizada bajo las siguientes circunstancias:

El sistema desarrollado fue ejecutado en un equipo con las siguientes características:

* Procesador Intel i5 3210M.
* Memoria 8GB de RAM DDR3 a 1600 MHZ.
* Disco duro de 500GB (5400 RPM).
* Sistema Operativo Windows 8.1.

El programa de prueba fue ejecutado por diez computadoras facilitadas por la dirección de escuela de computación de la Universidad Rafael Urdaneta (URU), las características incluían:

* Procesador Intel Dual Core.
* Memoria 4GB de RAM DDR2 a 800 MHZ.
* Disco duro de 500GB (7200 RPM).
* Sistema Operativo Windows 7.

En la figura 4.33 se observa un diagrama sencillo y resumido donde de forma simple se muestra la secuencia de eventos que ocurrió al realizarse la prueba. Cabe destacar que los resultados de los tiempos de ejecución de las pruebas fueron escritos por clase “CSVWriter” por cada cliente de prueba, en cada nodo donde se realizó la prueba.

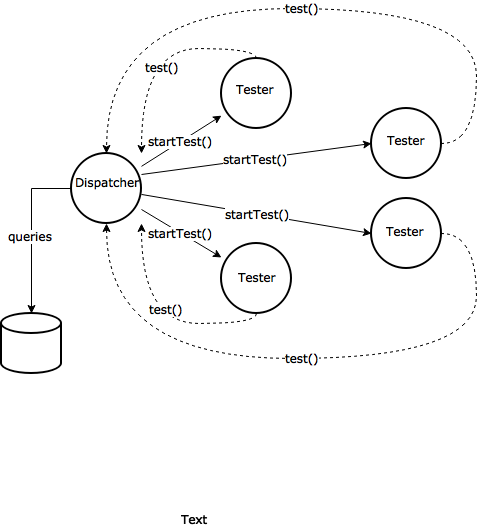


Figura 4.33. Diagrama simplificado de eventos durante la prueba

Las pruebas de estrés consistieron en la ejecución de una secuencia intensa de llamadas o solicitudes al “Dispatcher”, las primeras cuatro solicitudes fueron al método “queryJSON” enviando solicitudes del tipo selección, inserción, modificación y borrado, el siguiente método en la secuencia fue “writeJSON” (escribe un archivo en formato JSON en el disco), luego se ejecuta el método “writeEntireCSV” (escribe un archivo en formato CSV en el disco), el último método de la secuencia fue el “queryDataTable” (retorna un objeto de la clase “DataTable” con la información requerida).

Todos los resultados de las pruebas pueden ser hallados en el repositorio remoto dentro de la carpeta “test-analysis”, como archivos CSV. A partir de los datos obtenidos de la prueba, se procedió a realizar gráficos de barras par cada cliente que participaron en la prueba ver figuras 4.34 a la 4.43, diferenciando la intensidad de la prueba a partir del color de cada barra, la intensidad se refiere a la cantidad de veces que se repitieron las solicitudes de izquierda a derecha serían diez, quince, veinticinco y cincuenta veces respectivamente.

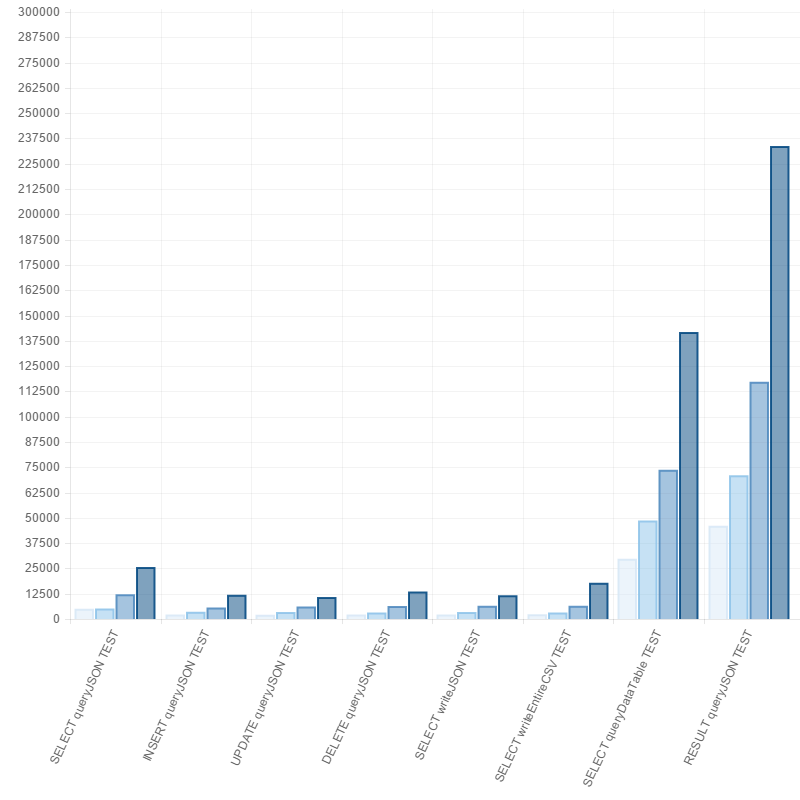


Figura 4.34. Resultados en ms para el nodo 0

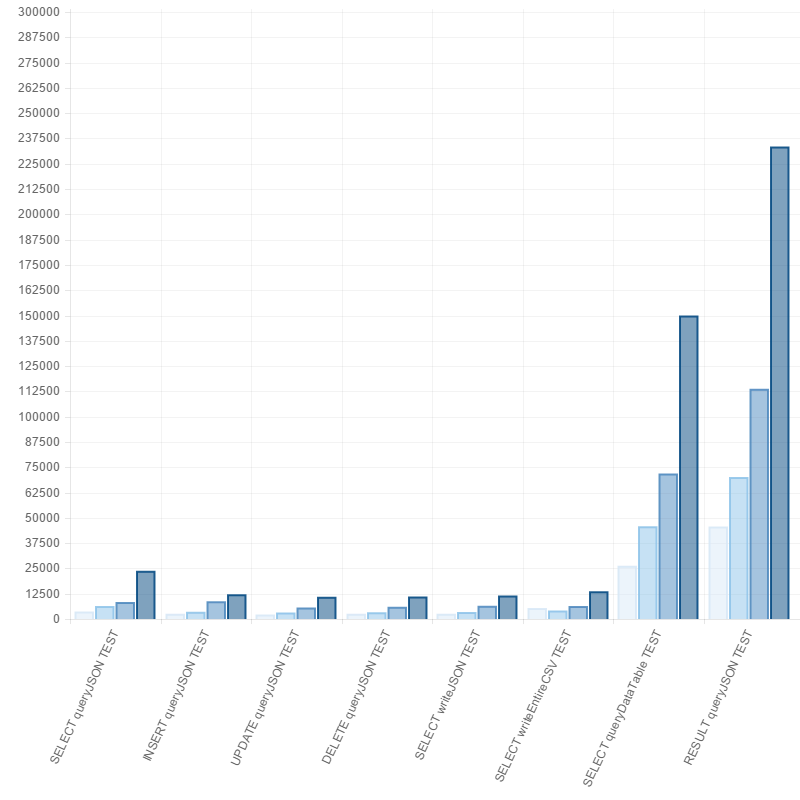


Figura 4.35. Resultados en ms para el nodo 1

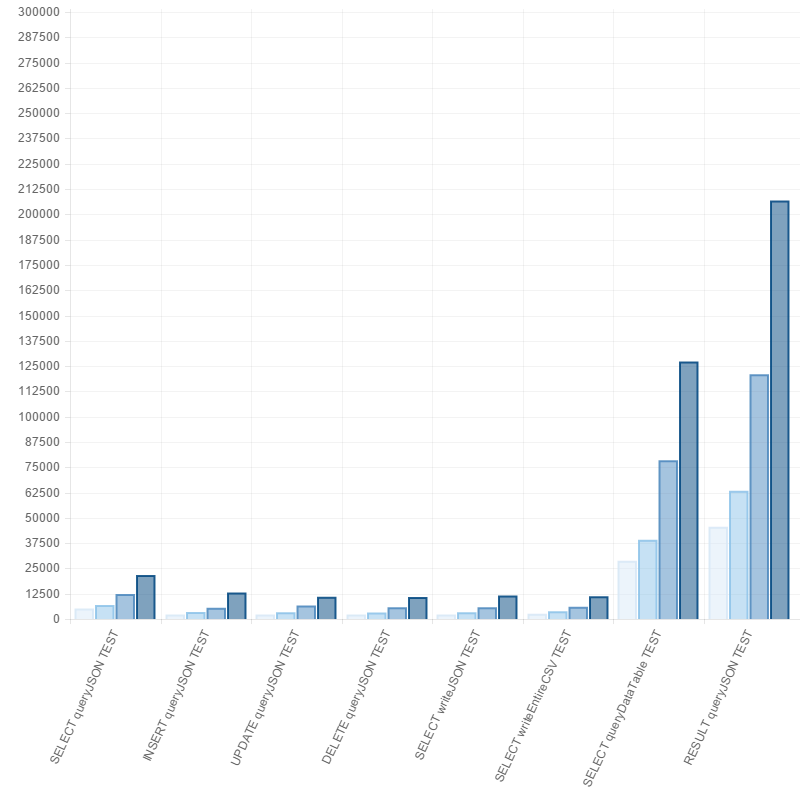


Figura 4.36. Resultados en ms para el nodo 2

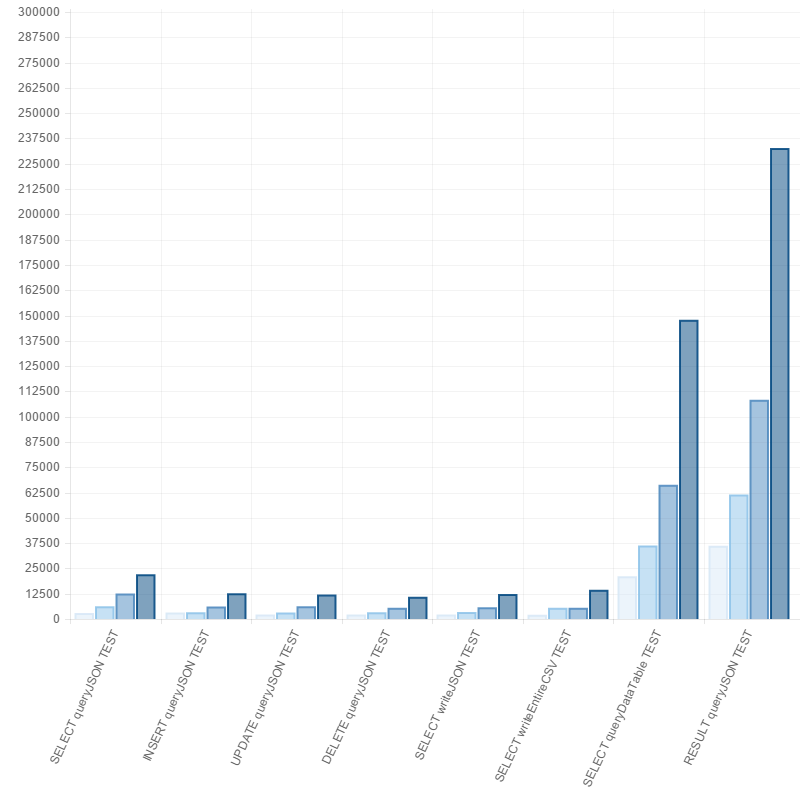


Figura 4.37. Resultados en ms para el nodo 3

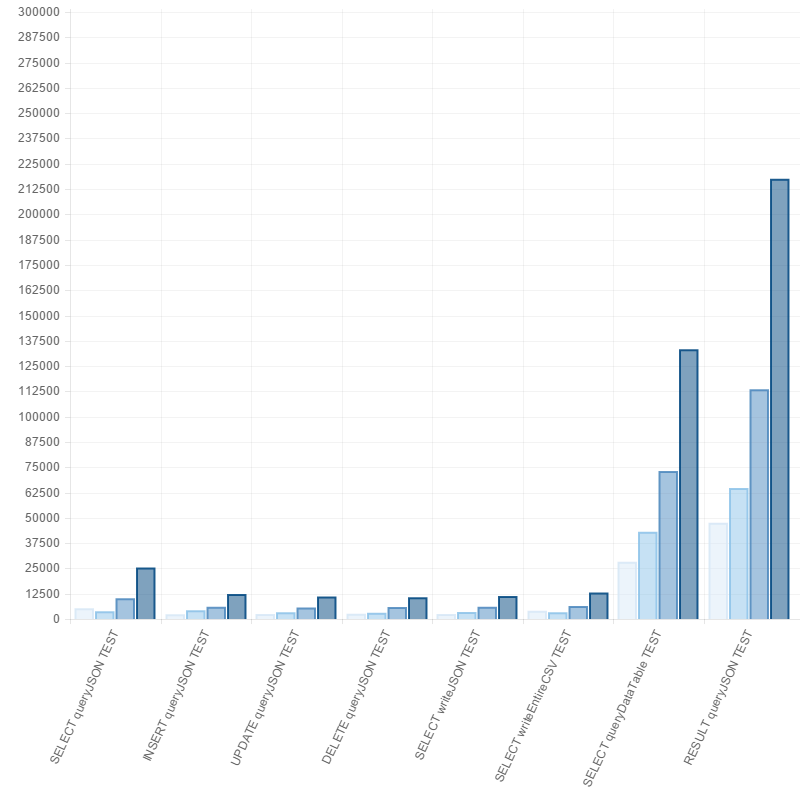


Figura 4.38. Resultados en ms para el nodo 4

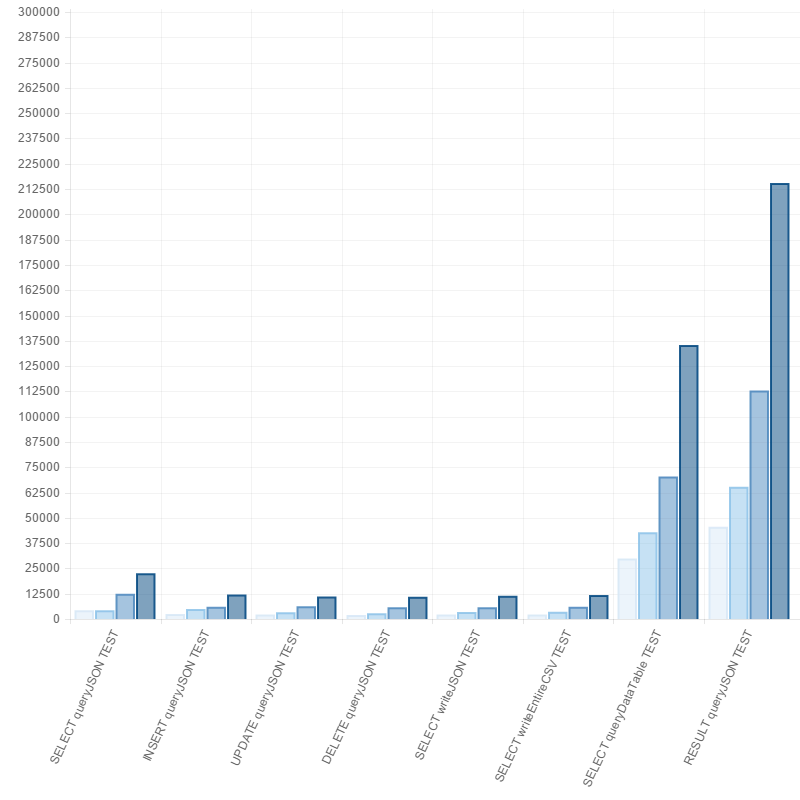


Figura 4.39. Resultados en ms para el nodo 5

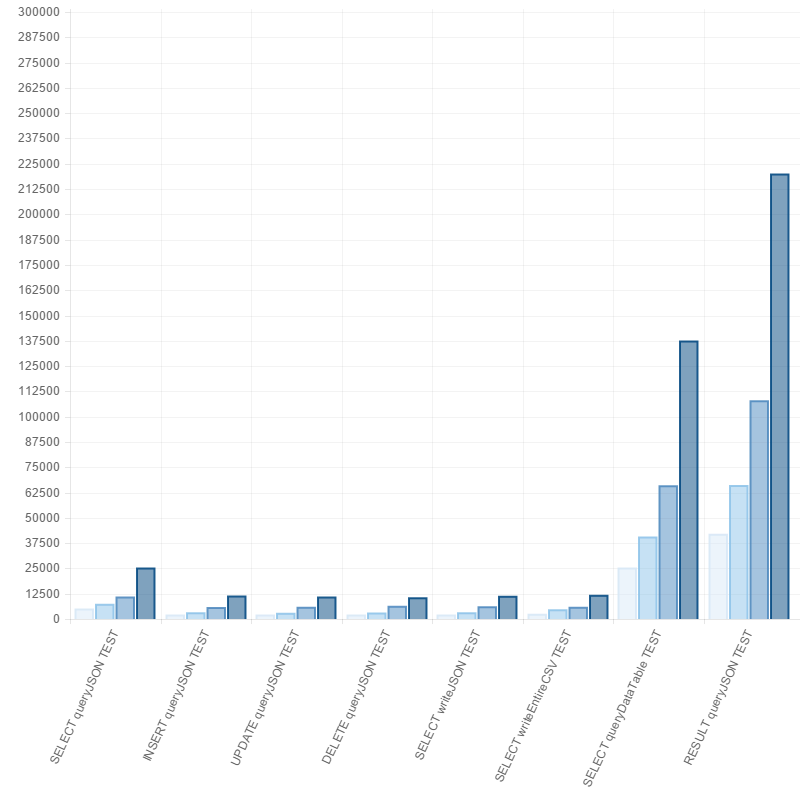


Figura 4.40. Resultados en ms para el nodo 6

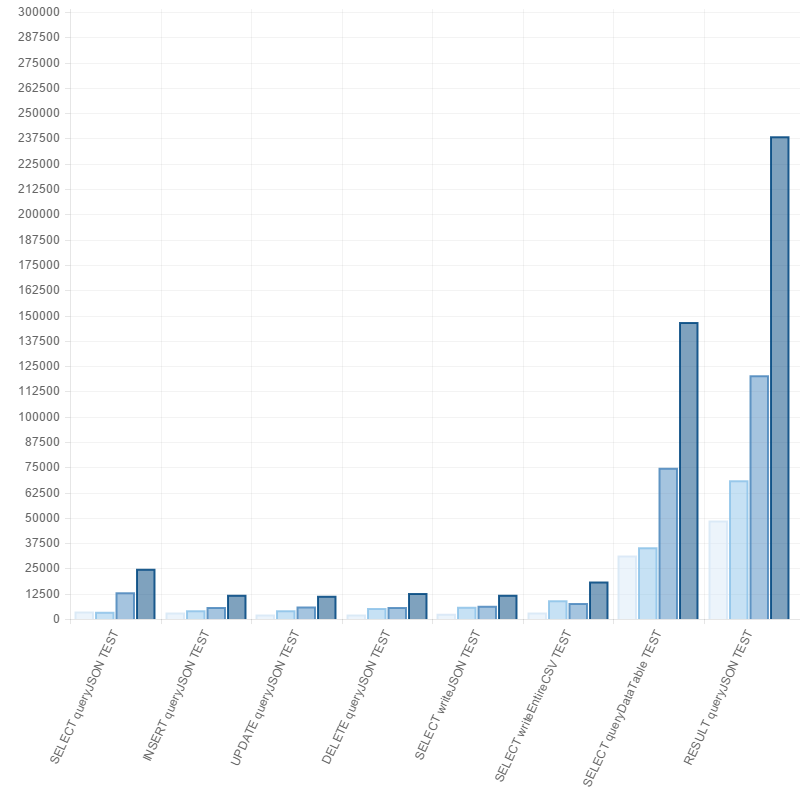


Figura 4.41. Resultados en ms para el nodo 7

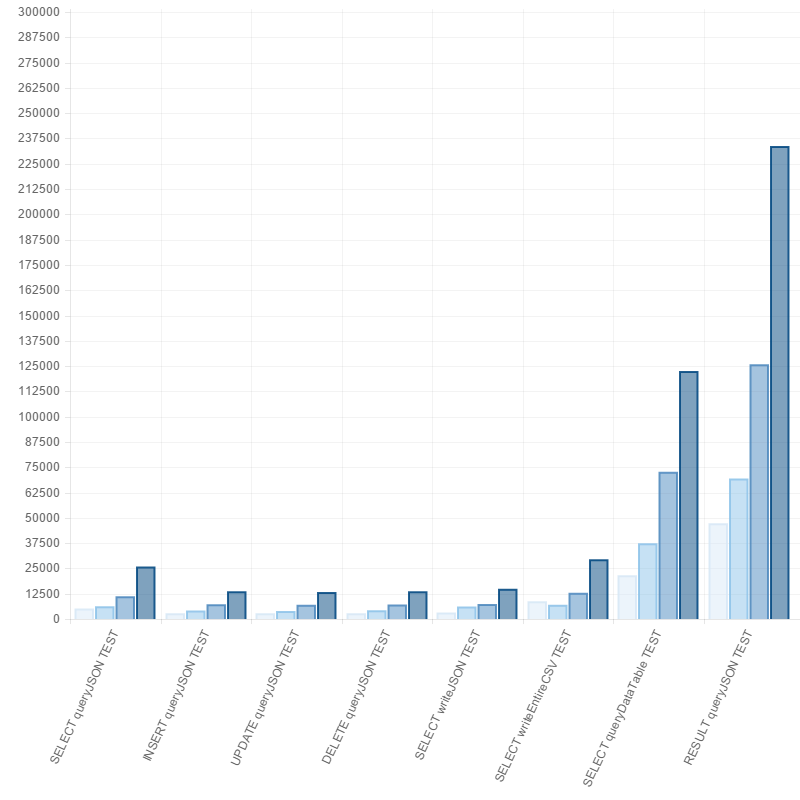


Figura 4.42. Resultados en ms para el nodo 8

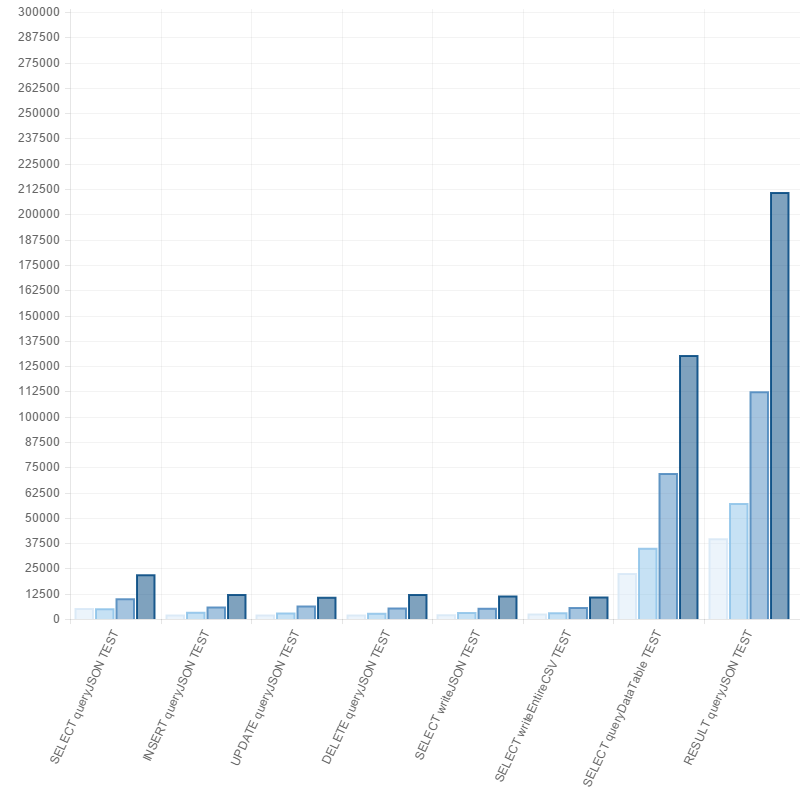


Figura 4.43. Resultados en ms para el nodo 9

Analizando el conjunto de gráficos se observan las siguientes características, que se cumplen para cada uno de ellos:

* El comportamiento para todos los nodos participantes es muy similar.
* El método que requiere más tiempo para ser completado es “queryDataTable”, a pesar de que éste no requiere de acceso al disco, aspecto que si requieren los métodos “writeJSON” y “writeEntireCSV”.
* El sistema se mantiene totalmente operativo y funcional a pesar del incremento de carga en un 500%.
* Por observación el desempeño del sistema es lineal con respecto al crecimiento de las solicitudes hechas.

## CONCLUSIONES

El desarrollo de este trabajo de investigación condujo a las siguientes conclusiones:

* El sistema desarrollado permitió reconocer la importancia de los sistemas distribuidos en el área de desarrollo de software, sobre todo por permitir un alto grado de versatilidad para el desarrollo de aplicaciones.
* El desarrollo de la aplicación permitió comprender la complejidad con los que muchos frameworks están desarrollados en la actualidad.
* El diseño de los diagramas UML, permitió primeramente abordar cierto grado de documentación, la cual es mundialmente aceptada en el desarrollo de software, así mismo también permitió aclarar muchísimos aspectos sobre la forma en que se debían programar y comunicar los diferentes módulos y componentes que integran al sistema.
* La programación de este sistema condujo a entender porque en el contexto del desarrollo de software es importante proveer soluciones lo más versátiles posibles, sobre todo porque las aplicaciones con esta característica son más valoradas ya que permiten ser empleadas en diversos sistemas operativos.
* La implementación de las diferentes pruebas (unidad, integración y estrés) facilitó vislumbrar la importancia que tiene ejecutar dichas pruebas en cualquier sistema que sea desarrollado, permitiendo hallar errores fácilmente con el subsiguiente aumento de la productividad del desarrollador.
* La prueba de estrés del sistema permitió determinar cómo se desempeña la aplicación en momentos de alto nivel de carga y concurrencia, percibiendo cuales métodos tienen un desempeño superior que otros.
* La culminación del desarrollo del sistema, permitió la obtención de un sistema para acceder a múltiples bases de datos de forma concisa y simple, de muy fácil implementación ya que el sistema está contenido dentro de un archivo JAR que solo requiere ser ejecutado en una JVM, también es muy versátil al haber sido desarrollado utilizando servicios web, y que al haber usado un pool de conexiones robusto permite un alto nivel de concurrencia, a su vez el sistema permite el acceso a las bases de datos más difundidas en la actualidad, sin requerir que el usuario provea las librerías de conexión.

## RECOMENDACIONES

Si bien este trabajo de investigación tuvo como varias metas la difusión de información referente sobre los servicios web, su objetivo principal era el desarrollo de un sistema que permitiera acceso a bases de datos tanto locales o remotas, y que permitiera el acceso a los servicios a clientes remotos. Por esta razón, se proporcionan una serie de recomendaciones que permitirán a los futuros investigadores una mejor comprensión del funcionamiento de este tipo de sistemas distribuidos.

* El análisis de las gráficas de desempeño, reveló el pobre rendimiento de la clase “DataTable” se sugiere la reprogramación de la misma, obviando la utilización de arreglos de datos primitivos para utilizar “ArrayLists”.
* Debido a la imposibilidad de realizar pruebas de estrés más sofisticadas, se sugiere realizar estas pruebas con una cantidad de nodos mayor, garantizando niveles de concurrencia elevados.
* Los objetivos de la investigación se limitaban a permitir el acceso a un número limitado de manejadores de bases de datos, con unos drivers que están implementados internamente. Sería conveniente encontrar un mecanismo para que el JAR que contiene la aplicación de ejecución del sistema tuviera la capacidad de cargar de forma dinámica los manejadores de conexiones, permitiendo así que el usuario de la aplicación pueda incorporar el soporte a más manejadores de bases de datos, lo que garantizaría una reducción del tamaño del JAR generado.

REFERENCIAS BIBLIOGRÁFICAS

* Kreger, H. (2001*). Web Services Conceptual Architecture (WSCA 1.0)*.
* Chappel, D. & Jewell, T. (2002). *Java Web Services.*
* Liu, M. I. (2004). *Computación Distribuida, fundamentos y aplicaciones*.
* Sarría, F. A (2006). *Sistemas de Información Geográfica.*
* Navarro R. (2006) *Modelado, Diseño e Implentación de Servicios Web.*
* Josuttis, N. M. (2007). *SOA in Practice*.
* Sznajdleder, P (2013). [*Java a fondo. Estudio del lenguaje y desarrollo de aplicaciones*](http://www.ietf.org/rfc/rfc4180.txt) *(2da Ed.)*. Buenos Aires, Argentina, Alfaomega.
* Reynolds, R & Postel J (October 1997). “Assigned Numbers”. Network Working Group (RFC 1400): 54.
* Shafranovich, Y. (October 2005). ["Common Format and MIME Type for CSV Files"](http://www.ietf.org/rfc/rfc4180.txt). *Network Working Group* ([RFC 4180](https://tools.ietf.org/html/rfc4180)): 1.
* Bianco P., Kotermanski Rick & Merson P. (2007) Evaluatinf a Service-Oriented Architecture.
* The Tomcat JDBC Connection (2015, Julio 1). Obtenido de: https://tomcat.apache.org/tomcat-7.0-doc/jdbc-pool.html#Introduction.
* ECMA International (2013). *The JSON Data Interchange Format*.
* W3C Communications Team (1999). *XML in 10 points*, [w3.org] de: http://www.w3.org/XML/1999/XML-in-10-points.
* Adamson C. (2006). *What is Java*. [onjava.com] de http://www.onjava.com/pub/a/onjava/2006/03/08/what-is-java.html#java-programming-language.
* Barrios, J. M. (2001). *Java Servlets* [users.dcc.uchile.cl] http://users.dcc.uchile.cl/~jbarrios/servlets/general.html.
* Oracle Corporation (2014) *The Java Tutorials, Lesson: Annotation.* [docs.oracle.com] https://docs.oracle.com/javase/tutorial/java/annotations/.
* Rouse M. (2006). *HTTP (Hypertext Transfer Protocol).* [techtarget.com]. http://searchwindevelopment.techtarget.com/definition/HTTP. Consultado el 28 de abril de 2014.
* Rouse, M. (2015). *What is integration testing or integration and testing (I&T) ? - Definition from WhatIs.com*. *SearchSoftwareQuality*. Obtenido el 22 Julio 2015 de http://searchsoftwarequality.techtarget.com/definition/integration-testing
* Rouse M. (2012) *Service Oriented Architecture (SOA)* [techtarget.com]. <http://searchsoa.techtarget.com/definition/service-oriented-architecture>.
* Barrientos, Pablo Andrés (25 de abril de 2014). [*Enfoque para pruebas de unidad basado en la generación aleatoria de objetos*](http://sedici.unlp.edu.ar/handle/10915/34969). p.101. Consultado el 28 de abril de 2014.
* Fowler, M., & Scott, K. (2000). *UML distilled*. Reading, Mass.: Addison Wesley.
* Molyneaux, I. (2009). *The art of application performance testing*. Sebastopol, Calif.: O'Reilly Media, Inc.