



UNIVERSIDAD TÉCNICA PARTICULAR DE LOJA

La Universidad Católica de Loja

ÁREA TÉCNICA

TÍTULO DE INGENIERO EN INFORMÁTICA

**Análisis e implementación de aplicaciones mediante el framework Django
en un entorno de desarrollo para una arquitectura a gran escala.**

TRABAJO DE TITULACIÓN

AUTOR: Pincay Castro, Dany Wilson

DIRECTOR: Elizalde Solano, René Rolando, Mgs.

CENTRO UNIVERSITARIO MACHALA

2017



Esta versión digital, ha sido acreditada bajo la licencia Creative Commons 4.0, CC BY-NY-SA: Reconocimiento-No comercial-Compartir igual; la cual permite copiar, distribuir y comunicar públicamente la obra, mientras se reconozca la autoría original, no se utilice con fines comerciales y se permiten obras derivadas, siempre que mantenga la misma licencia al ser divulgada. <http://creativecommons.org/licenses/by-nc-sa/4.0/deed.es>

2017

APROBACIÓN DEL DIRECTOR DEL TRABAJO DE TITULACIÓN

Magister.

René Rolando Elizalde Solano

DOCENTE DE LA TITULACIÓN

De mi consideración:

El presente trabajo de titulación: “Análisis e implementación de aplicaciones mediante el framework Django en un entorno de desarrollo para una arquitectura a gran escala” realizado por Dany Wilson Pincay Castro, ha sido orientado y revisado durante su ejecución, por cuanto se aprueba la presentación del mismo.

Loja, septiembre de 2017

f)

DECLARACIÓN DE AUTORÍA Y CESIÓN DE DERECHOS

“Yo, Dany Wilson Pincay Castro declaro ser autor (a) del presente trabajo de titulación: “Análisis e implementación de aplicaciones mediante el framework Django en un entorno de desarrollo para una arquitectura a gran escala”, de la Titulación de Informática, siendo René Rolando Elizalde Solano director (a) del presente trabajo; y eximo expresamente a la Universidad Técnica Particular de Loja y a sus representantes legales de posibles reclamos o acciones legales. Además, certifico que las ideas, conceptos, procedimientos y resultados vertidos en el presente trabajo investigativo, son de mi exclusiva responsabilidad.

Adicionalmente declaro conocer y aceptar la disposición del Art. 88 del Estatuto Orgánico de la Universidad Técnica Particular de Loja que en su parte pertinente textualmente dice: “Forman parte del patrimonio de la Universidad la propiedad intelectual de investigaciones, trabajos científicos o técnicos y tesis de grado o trabajos de titulación que se realicen con el apoyo financiero, académico o institucional (operativo) de la Universidad”

f.

Autor: Dany Wilson Pincay Castro

Cédula: 0704252006

DEDICATORIA

El presente trabajo de fin de titulación, lo dedico con mucho amor a mis padres Deysi Castro y Wilson Pincay, quienes estuvieron siempre a mi lado brindándome su apoyo incondicional y fortaleza.

A mi querido hijo Estéfano, que se convirtió en la motivación para poder alcanzar este objetivo académico, a mi amada esposa Estefanía quien me ayudó y me alentó a lograr esta meta, y a mis queridos hermanos Rudy y Sandy por haber estado presentes cuando necesité su ayuda.

Dany Wilson Pincay Castro

AGRADECIMIENTO

A la Universidad Técnica Particular de Loja, y su personal docente, por haberme transmitido los conocimientos académicos, que me han permitido formarme como profesional.

Al Mgs. René Rolando Elizalde Solano, director de mi trabajo de fin de titulación, por el asesoramiento y guía en el desarrollo del presente proyecto de tesis.

Dany Wilson Pincay Castro

ÍNDICE DE CONTENIDOS

APROBACIÓN DEL DIRECTOR DEL TRABAJO DE TITULACIÓN	II
DECLARACIÓN DE AUTORÍA Y CESIÓN DE DERECHOS.....	III
DEDICATORIA.....	IV
AGRADECIMIENTO.....	V
ÍNDICE DE CONTENIDOS.....	VI
ÍNDICE DE FIGURAS.....	XI
ÍNDICE DE TABLAS.....	XIV
RESUMEN	1
ABSTRACT	2
INTRODUCCIÓN	3
OBJETIVOS	5
GLOSARIO.....	6
CAPÍTULO I. MARCO TEÓRICO.....	9
1.1 ANTECEDENTES	10
1.2 DISEÑO DE ARQUITECTURA TECNOLÓGICAS A GRAN ESCALA	10
1.2.1 <i>Arquitectura de la información</i>	11
1.2.2 <i>Arquitectura tecnológica</i>	11
1.2.3 <i>¿Qué es un Clúster?</i>	12
1.2.4 <i>Elementos que forman parte de un clúster</i>	12
1.2.5 <i>Características y Servicios de los Clúster</i>	13
1.2.5.1 Alto Rendimiento (High performance):.....	13
1.2.5.2 Alta Disponibilidad (High availability):	13
1.2.5.2.1 Configuración del Clúster de Alta disponibilidad.	13
1.2.5.2.2 Funcionamiento de un clúster de alta disponibilidad.....	13
1.2.5.3 Escalabilidad	14
1.2.5.4 Balanceo de Carga.....	14
1.2.5.4.1 Balanceo de carga por hardware	15
1.2.5.4.2 Balanceo de carga por software	15
1.2.5.4.3 Balanceo de Carga sobre el servidor Web	16
1.2.5.4.4 Balanceo de Carga de Base de Datos	17
1.2.5.4.5 Balanceo de Carga de alta disponibilidad	17
1.2.5.4.6 Algoritmos de balanceo de carga.....	17
1.3 ARQUITECTURA APLICACIÓN WEB	18
1.3.1 <i>Modelo Vista Controlador (MVC)</i>	18
1.3.1.1 Elementos del MVC	20
1.3.1.1.1 Modelo.....	20
1.3.1.1.2 Vista.....	20
1.3.1.1.3 Controlador	21
1.3.1.2 Ventajas del MVC.....	21
1.3.1.3 Desventajas del MVC.....	22
1.3.2 <i>Framework</i>	22
1.3.3 <i>Framework desarrollados con MVC</i>	22
1.3.3.1 Ruby on Rails (Ruby)	23
1.3.3.2 Struts (Java / J2ee)	23

1.3.3.3	Catalyst (Perl).....	23
1.3.3.4	Symfony (PHP).....	23
1.3.3.5	Java (Frameworks).....	24
1.3.3.5.1	Hibernate	24
1.3.3.5.2	Jsf.....	24
1.3.3.5.3	Struts	24
1.3.3.6	PHP (Frameworks).....	24
1.3.3.7	Ruby (Frameworks)	25
1.3.3.8	Python (Frameworks)	25
1.3.3.8.1	Django.....	26
1.3.3.8.2	Grok.....	27
1.3.3.8.3	TurboGears 2.....	27
1.3.3.8.4	Web2py.....	27
1.3.3.9	Selección del Framework Django para el desarrollo de este proyecto	27
1.3.4	<i>Introducción a Python – Django</i>	28
1.3.4.1	Lenguaje de programación en Python	28
1.3.4.2	Historia	29
1.3.4.3	Características.....	30
1.3.4.4	Elementos del lenguaje.....	30
1.3.4.5	Variables.....	30
1.3.4.6	Tipos de datos.....	31
1.3.4.7	Funciones.....	31
1.3.4.8	Módulos.....	32
1.3.4.9	Framework Web Django para Python.....	32
1.3.4.9.1	Descripción.....	32
1.3.4.9.2	Características	32
1.3.4.9.3	Arquitectura	33
1.3.4.9.4	Soporte de base de datos.....	34
1.3.4.9.5	Servidor Web.....	35
1.3.4.9.6	Requerimientos	35
1.4	SISTEMA OPERATIVO.....	35
1.4.1	<i>Software Libre</i>	35
1.4.1.1	Windows.....	36
1.4.1.2	Linux.....	36
1.4.1.3	Mac Os.....	36
1.5	BASE DE DATOS.....	40
1.5.1	<i>Sistema de Gestión de base de datos</i>	40
1.5.2	<i>Mysql</i>	41
1.5.3	<i>Características</i>	41
1.5.4	<i>Replicación Mysql</i>	42
1.5.4.1	Replicación maestro / esclavo.....	42
1.5.4.2	Replicación maestro / maestro	43
1.6	SOFTWARE Y HERRAMIENTAS PARA CLÚSTER.....	43
1.6.1	<i>OpenMosix</i>	43
1.6.2	<i>Piranha</i>	43
1.6.3	<i>Haproxy</i>	44
1.6.3.1	Características.....	45
1.6.3.2	Haproxy en clúster de base de datos.....	47
1.6.3.2.1	Ventajas de Haproxy en el equilibrio de carga.....	47
1.6.3.2.2	Chequeo de salud	48
1.6.3.2.3	Redundancia Haproxy	48
1.6.3.2.4	Detección de fallos y conmutación por error	49
1.6.3.2.5	Descripción de la configuración principal de Haproxy.....	49
1.6.4	<i>Tabla comparativa entre software para clúster</i>	50

1.6.5	<i>Heartbeat</i>	52
1.7	PRUEBAS DE CARGA Y RENDIMIENTO DE SOFTWARE	52
1.7.1	<i>Pruebas de carga</i>	52
1.7.2	<i>Pruebas de rendimiento</i>	53
1.7.3	<i>Pruebas de estrés</i>	53
1.7.4	<i>Apache Jmeter</i>	53
1.8	TRABAJOS RELACIONADOS	54
CAPÍTULO II. SOLUCION A LA PROBLEMÁTICA		61
2.1	INTRODUCCIÓN	62
2.2	ARQUITECTURA TECNOLÓGICA A GRAN ESCALA PROPUESTA	63
2.2.1	<i>Alta disponibilidad</i>	64
2.2.2	<i>Balanceo de carga</i>	64
2.2.2.1	Algoritmo de Balanceo de carga.....	64
2.2.3	<i>Detalle de Arquitectura Tecnológica a utilizar</i>	65
2.2.4	<i>Software empleado en la Arquitectura</i>	66
2.3	CONFORMACIÓN DE CLÚSTER PARA LA ARQUITECTURA TECNOLÓGICA	67
2.3.1	<i>Clúster web server</i>	70
2.3.2	<i>Clúster base de datos</i>	71
2.4	INSTALACIÓN Y CONFIGURACIÓN DE CLÚSTER WEBSERVER	71
2.4.1	<i>Configuración servidores de balanceo de carga (Haproxyweb1 y Haproxyweb2)</i>	71
2.4.1.1	Instalación Haproxy y Heartbeat	73
2.4.1.2	Configuración Haproxy	73
2.4.1.3	Configuración de Heartbeat.....	75
2.4.2	<i>Configuración servidores web reales (Webserver1 y Webserver2)</i>	76
2.4.2.1	Instalación Apache	77
2.5	INSTALACIÓN Y CONFIGURACIÓN CLÚSTER DE BASE DE DATOS	77
2.5.1	<i>Configuración servidores de base de datos mysql – replicación (Bdserver1 y Bdserver2)</i>	77
2.5.1.1	Instalar y configurar Mysql en Bdserver1.....	78
2.5.1.2	Instalar y configurar Mysql en Bdserver2.....	80
2.5.1.3	Finalizar configuración de réplica en bdserver1	82
2.5.1.4	Prueba De Replicación Master Master	82
2.5.2	<i>Configuración servidores de balanceo de carga(HaproxyBd1 y HaproxyBd2)</i>	82
2.5.2.1	Configurar servidores Mysql (Bdserver1 y Bdserver2).....	84
2.5.2.2	Instalación Mysql Client (HaproxyBd1 y HaproxyBd2)	84
2.5.2.3	Instalación y Configuración Haproxy (HaproxyBd1 y HaproxyBd2).....	85
2.5.2.4	Configuración De Heartbeat.....	87
2.5.2.5	Prueba de Balanceo de Carga.....	88
CAPÍTULO III. PRUEBAS A LA SOLUCIÓN		90
3.1	APLICACIÓN WEB	91
3.1.1	<i>Análisis de requerimientos</i>	91
3.1.1.1	Requerimientos funcionales	91
3.1.1.2	Requerimientos no funcionales.....	91
3.1.2	<i>Diseño de aplicación</i>	92
3.1.2.1	Diagrama de clases.....	92
3.1.2.2	Diagrama de base de datos.....	95
3.1.2.3	Diagrama de vistas	97
3.1.2.4	Diagrama de templates y menú	98

3.1.3	<i>Datos de la base de datos</i>	99
3.2	PRUEBAS DE CARGA Y RENDIMIENTO A LA ARQUITECTURA TECNOLÓGICA	99
3.2.1	<i>Encendido de los servidores del clúster web y base de datos</i>	99
3.2.2	<i>Prueba 1 realizada con herramienta Jmeter con 1 servidor web</i>	101
3.2.2.1	Revisión de estadísticas de sesiones y balanceo de carga previo a las pruebas	102
3.2.2.2	Configuración de plan de pruebas con Jmeter	104
3.2.2.2.1	Plan de Pruebas	104
3.2.2.2.2	Parámetros.....	104
3.2.2.3	Ejecución del plan de pruebas Jmeter	106
3.2.2.4	Resultados del plan de pruebas con Jmeter.....	107
3.2.2.4.1	Resultados en view results tree	107
3.2.2.4.2	Resultados en vista en tabla	110
3.2.2.4.1	Resultado gráfico	112
3.2.2.5	Revisión de las estadísticas de sesiones y balanceo de carga posterior a las pruebas con Jmeter	113
3.2.3	<i>Prueba 2 realizada con herramienta Jmeter con 2 servidores web</i>	116
3.2.3.1	Revisión de estadísticas de sesiones y balanceo de carga previo a las pruebas	117
3.2.3.2	Configuración de plan de pruebas con Jmeter	119
3.2.3.2.1	Plan de Pruebas	119
3.2.3.2.2	Parámetros.....	119
3.2.3.3	Ejecución del plan de pruebas Jmeter	121
3.2.3.4	Resultados del plan de pruebas con Jmeter.....	121
3.2.3.4.1	Resultados en view results tree	121
3.2.3.4.2	Resultados en view results in table	124
3.2.3.4.1	Resultado gráfico	126
3.2.3.5	Revisión de las estadísticas de sesiones y balanceo de carga posterior a las pruebas con Jmeter	127
3.2.4	<i>Análisis de resultados entre las 2 pruebas con Jmeter</i>	129
3.2.5	<i>Prueba 3 realizada con javascript desde la consola del navegador con 1 servidor web</i>	131
3.2.5.1	Revisión de estadísticas de sesiones y balanceo de carga previo a las pruebas	131
3.2.5.2	Código javascript utilizado en el navegador	133
3.2.5.3	Resultados de la prueba	133
3.2.5.4	Revisión de las estadísticas de sesiones y balanceo de carga posterior a las pruebas con javascript ..	135
3.2.6	<i>Prueba 4 realizada con javascript desde la consola del navegador con 2 servidores web</i>	137
3.2.6.1	Revisión de estadísticas de sesiones y balanceo de carga previo a las pruebas	137
3.2.6.2	Script utilizado en el navegador	140
3.2.6.3	Resultados de la prueba	140
3.2.6.4	Revisión de las estadísticas de sesiones y balanceo de carga posterior a las pruebas con javascript ..	142
3.2.7	<i>Análisis de resultados entre las 2 pruebas con Javascript</i>	144
	CONCLUSIONES	146
	RECOMENDACIONES	148
	BIBLIOGRAFÍA	149
	ANEXOS	154
1.1	CODIFICACIÓN DE APLICACIÓN WEB (CONFIGURACIÓN)	155
1.1.1	<i>settings.py</i>	155
1.1.2	<i>models.py</i>	157
1.1.3	<i>url.py</i>	159
1.1.4	<i>wsgi.py</i>	160
1.2	CÓDIGO APLICACIÓN WEB (VISTAS)	160
1.2.1	<i>historia_clinica.py</i>	160
1.2.2	<i>turno</i>	163
1.2.3	<i>afinidad.py</i>	166
1.2.4	<i>agendador.py</i>	168

1.2.5	<i>ciudad.py</i>	170
1.2.6	<i>consultorio.py</i>	173
1.2.7	<i>enfermedad.py</i>	176
1.2.8	<i>especialidad.py</i>	179
1.2.9	<i>grupo_cultural.py</i>	182
1.2.10	<i>instruccion_educativa.py</i>	184
1.2.11	<i>medico.py</i>	187
1.2.12	<i>pais.py</i>	190
1.2.13	<i>report.py</i>	193

ÍNDICE DE FIGURAS

Figura 1: Elementos de un Clúster	12
Figura 2: Balanceo de Carga	15
Figura 3: Modelo-Vista-Controlador	19
Figura 4: Replicación Maestro-esclavo	41
Figura 5: Replicación Maestro-esclavo 2	43
Figura 6: Haproxy-esquema de funcionamiento	45
Figura 7: Mysql-check.....	48
Figura 8: Configuración Haproxy.....	50
Figura 9: Resultados Siege.....	56
Figura 10: Reporte de estadísticas Haproxy	56
Figura 11: Prueba alta disponibilidad	59
Figura 12: Prueba de fallo nodo servidor web	60
Figura 13: modelo convencional cliente-servidor	63
Figura 14: Modelo convencional 2 cliente-servidor.....	63
Figura 15: Granja de servidores en VirtualBox.....	66
Figura 16: Diagrama arquitectura tecnológica a gran escala.....	68
Figura 17: Diagrama de clases	94
Figura 18: Diagrama de base de datos	96
Figura 19: Diagrama de vistas	97
Figura 20: Diagrama de templates y menú	98
Figura 21: Base de datos llena	99
Figura 22: Servidores encendidos en VirtualBox.....	100
Figura 23: Servidores ejecutándose.....	100
Figura 24: Servidor Webserver2 apagado.....	102
Figura 25: Reporte estadísticas Haproxy previo a la prueba1	103
Figura 26: Configuración plan de pruebas	104
Figura 27: Número de usuarios concurrentes	105
Figura 28: Configuración Ip y path	105
Figura 29: Creación de listener	106
Figura 30: Ejecución de pruebas en Jmeter	107
Figura 31: Petición url	107
Figura 32: Resultados vista en árbol.....	108
Figura 33: Resultados vista en árbol.....	109
Figura 34: Resultados vista en árbol.....	109
Figura 35: Resultados vista en tabla	110

Figura 36: Resultados vista en tabla	111
Figura 37: Resultados vista en tabla	112
Figura 38: Resultados en gráfica	113
Figura 39: Reporte estadísticas Haproxy posterior a la prueba1	114
Figura 40: Clúster con todos los servidores ejecutándose	117
Figura 41: Reporte estadísticas Haproxy previo a la prueba2	118
Figura 42: Configuración grupo de usuarios (hilos)	119
Figura 43: Configuración de conexión.....	120
Figura 44: Creación de listener	120
Figura 45: Ejecución de prueba en Jmeter.....	121
Figura 46: Resultados vista en árbol.....	122
Figura 47: Resultados vista en árbol.....	123
Figura 48: Resultados vista en árbol.....	123
Figura 49: Resultados vista en tabla	124
Figura 50: Resultados vista en tabla	125
Figura 51: Resultados vista en tabla	126
Figura 52: Resultado gráfico	127
Figura 53: Reporte estadísticas Haproxy posterior a la prueba2	128
Figura 54: Reporte estadísticas haproxy previo a la prueba3.....	132
Figura 55: Código JavaScript para la prueba3	133
Figura 56: Resultados prueba3	134
Figura 57: Resultados prueba3	134
Figura 58: Resultados prueba3	135
Figura 59: Resultados estadísticas Haproxy posterior a la prueba3.....	136
Figura 60: Resultados estadísticas haproxy previo a la prueba4.....	139
Figura 61: Código JavaScript para la prueba4	140
Figura 62: Resultados prueba4	141
Figura 63: Resultados prueba4	141
Figura 64: Resultados prueba4	142
Figura 65: Reporte estadísticas Haproxy posterior a la prueba4	143
Figura 66: Vista historia clínica	160
Figura 67: Vista turno.....	163
Figura 68: Vista afinidad	166
Figura 69: Vista agendador.....	168
Figura 70: Vista ciudad	171
Figura 71: Vista consultorio.....	174
Figura 72: Vista enfermedad.....	176

Figura 73: Vista especialidad	179
Figura 74: Vista grupo cultural	182
Figura 75: Vista instrucción educativa.....	184
Figura 76: Vista médico	187
Figura 77: Vista país	190
Figura 78: Reporte citas y pacientes.....	193

ÍNDICE DE TABLAS

Tabla 1: Software de implementación clúster.....	16
Tabla 2: Características de Python	26
Tabla 3: Tipos de datos Python.....	31
Tabla 4: Comparación de diferentes sistemas operativos, ventajas y desventajas.....	37
Tabla 5: Tabla comparativa de software para clúster	50
Tabla 6: Computador donde se alojarán los servidores virtuales	65
Tabla 7: Clústeres de la arquitectura tecnológica.....	67
Tabla 8: Clúster WebServer.....	70
Tabla 9: Clúster base de datos	71
Tabla 10: Servidores disponibles en prueba1	101
Tabla 11: Servidores disponibles en prueba2	116
Tabla 12: Comparación de resultados de prueba1 y prueba2	130
Tabla 13: Análisis de resultados entre prueba3 y prueba4.....	144

RESUMEN

El presente trabajo de fin de titulación, está dirigido al análisis e implementación de una arquitectura tecnológica en un entorno de desarrollo a gran escala, misma que se fusionará con una aplicación web desarrollada con el framework Django. En este sentido la arquitectura tecnológica, tiene como objetivo la gestión de grandes volúmenes de información de acuerdo los modelos de las estructuras a gran escala.

La arquitectura tecnológica está conformada por una granja de servidores, dividida en dos clústeres (clúster web y clúster de base de datos), la cual brinda los servicios de balanceo de carga entre los servidores reales, alta disponibilidad, escalabilidad y persistencia de sesiones. Esta solución se realizará por medio de software de código abierto, específicamente con las herramientas Haproxy y Heartbeat.

Para evidenciar el éxito en cuanto al rendimiento, de nuestra arquitectura tecnológica a gran escala, implementada como solución a la problemática, se la someterá a pruebas de carga y pruebas estrés, a través de la herramienta de pruebas Apache Jmeter.

PALABRAS CLAVES: Granja de servidores, Clúster, Balanceo de carga, Arquitectura tecnológica, Alta disponibilidad, Escalabilidad, Persistencia de sesiones, Django, Haproxy, Heartbeat.

ABSTRACT

This work is aimed at the analysis and implementation of a technological architecture in a large scale development environment, which will be merged with a web application developed with the Django framework. In this sense the technological architecture, aims to manage large volumes of information according to the models of large-scale structures.

The technology architecture consists of a server farm, divided into two clusters (web cluster and database cluster), which provides load balancing services between real servers, high availability, scalability and persistence of sessions. This solution will be done using open source software, specifically with Haproxy and Heartbeat tools.

To demonstrate the success in terms of performance, our large-scale technological architecture, implemented as a solution to the problem, will be subjected to load testing and stress testing, through the Apache Jmeter testing tool.

KEYWORDS: Server farm, Cluster, Load balancing, Technological architecture, High availability, Scalability, Session persistence, Django, Haproxy, Heartbeat.

INTRODUCCIÓN

El crecimiento acelerado de las aptitudes y competencias tecnológicas exige día a día la gestión y manejo de cantidades de información cada vez más grandes, por parte de las empresas o instituciones que ofrecen servicios o productos de diferente índole, en distintas áreas de desempeño, con el objetivo de brindar resultados y productos de calidad para todos sus usuarios.

Partiendo de este principio, se evidencia que los grandes volúmenes de información que manejan las empresas no son gestionados de manera óptima y, no cumplen con características como accesibilidad, integridad, confiabilidad y rapidez a la misma, factores esenciales para la toma correcta decisiones empresariales; en vista de esta problemática, existe la necesidad de crear aplicaciones capaces de gestionar información de gran tamaño en tiempo real.

Los paradigmas convencionales para el trato de la información a través de sistemas informáticos, utilizando la web como medio de comunicación, en la actualidad no satisfacen las necesidades y requerimientos de procesamiento, disponibilidad, distribución y balanceo de carga en la Base de Datos y alojamiento Web, ya que prácticamente todo este conjunto de actividades y procesos los absorbe un solo servidor, que por más recursos técnicos que posea el mismo, no es capaz de soportar y tener un rendimiento óptimo trabajando con grandes volúmenes de información.

Estos paradigmas disponen de un único equipo, donde es instalada la app, el servidor web, y la base de datos; o en su defecto como máximo 2 servidores, uno para alojamiento web, y otro para alojamiento de la base de datos. En ambos casos se evidencian los mismos problemas y carencias de distribución de carga de trabajo, distribución de peticiones, equilibrio en la carga en la Base de Datos, y además en estos sistemas existe el problema de tener un único punto de fallo.

En este sentido el presente trabajo de fin de titulación está dirigido al *Análisis e implementación de aplicaciones mediante el framework Django en un entorno de desarrollo para una arquitectura a gran escala*, y está compuesto por cinco capítulos que se detallan a continuación:

El capítulo uno está dirigido a la búsqueda y documentación desde fuentes bibliográficas científicas, sobre la conceptualización que encierran las arquitecturas tecnológicas a gran

escala, y las características que la comprenden, como: balanceo de carga, alta disponibilidad, escalabilidad, persistencia de sesiones, clúster de servidores, servidores dedicados, servidores virtuales, framework Django, entre otros.

El capítulo dos contempla la solución a la problemática, misma que consiste en la implementación de una granja de servidores para el alojamiento de un aplicativo web desarrollado con el framework Django.

El capítulo tres se enfoca en las pruebas a la solución, ejecutando de esta manera, pruebas de carga, pruebas de rendimiento y pruebas de estrés a la arquitectura tecnológica implementada, todo esto para constatar el óptimo rendimiento de nuestra arquitectura.

El capítulo cuatro está orientados al análisis de las conclusiones obtenidas luego del desarrollo de la arquitectura tecnológico y de las pruebas realizadas a la misma, además se presentan las recomendaciones respectivas para el desarrollo e implementación de las arquitecturas en un entorno de desarrollo a gran escala.

El capítulo cinco corresponde a los anexos, dentro de los cuales se encuentra el código de la aplicación web.

OBJETIVOS

General

- Analizar e implementar una aplicación mediante el framework Django en un entorno de desarrollo para una arquitectura a gran escala.

Específicos

- Realizar una documentación científica – técnica sobre las arquitecturas a gran escala en frameworks de desarrollo basados en MVC.
- Construir una aplicación haciendo uso de balanceo de carga a nivel de aplicación y base de datos.
- Realizar pruebas de rendimiento de la aplicación en base a diferentes escenarios.

GLOSARIO

Open-source o código abierto: Es el software que es distribuido y desarrollado libremente, básicamente, se refiere al hecho de adquirir algún tipo de programa de manera gratis y con opción a modificaciones personales.

Java Enterprise: Es una plataforma de programación para desarrollar y ejecutar software de aplicaciones en Lenguaje de programación Java con arquitectura de niveles distribuida, basándose ampliamente en componentes de software modulares.

Perl (Practical Extracting and Reporting Language): Es un lenguaje de programación utilizado para construir aplicaciones CGI para el web. Además, este lenguaje de desarrollo es muy práctico para extraer información de archivos de texto y generar informes a partir del contenido de los ficheros

Prado (PHP Rapid Application Development Object-oriented): Es un framework de programación basado en componentes y eventos para el desarrollo de aplicaciones Web en PHP 5.

Máquina Virtual: es un software que emula a un ordenador o computadora real, dentro del sistema operativo. Las demandas de CPU, memoria, disco duro, red y otros recursos de hardware son gestionadas por una capa de virtualización que traduce estas solicitudes a la infraestructura de hardware físico subyacente.

Alta Disponibilidad: Dentro de las arquitecturas a gran escala, es sinónimo de que el sistema se encuentre siempre disponible y no haya pérdida de servicio.

Balanceo de Carga: Distribución equilibrada de carga entre los servidores que conforman un clúster.

Persistencia de Sesiones: Se realiza por medio de un backup de la sesión de cada usuario para migrarla a otro servidor en el caso de que falle el servidor al que ingresó en primera instancia.

Escalabilidad: Posibilidad de que un clúster incremente el número de servidores que lo conforman.

Servidor Real: Servidor de alojamiento web o de base de datos.

Servidor Balanceador de Carga: Servidor que cumple funciones de equilibrar la carga de trabajo entre los servidores reales.

API (Application Programming Interface): Son un conjunto de reglas (código) y especificaciones que las aplicaciones pueden seguir para comunicarse entre ellas: sirviendo de interfaz entre programas diferentes de la misma manera en que la interfaz de usuario facilita la interacción humano-software.

Enterprise JavaBean: Es una de las interfaces de programación de aplicaciones (API) que forman parte del estándar de construcción de aplicaciones empresariales

Mapeo-objeto-relacional (ORM): Es un modelo de programación que consiste en la transformación de las tablas de una base de datos, en una serie de entidades que simplifiquen las tareas básicas de acceso a los datos para el programador.

Programación orientada a objetos (POO): Es un paradigma de programación que usa objetos y sus interacciones para diseñar aplicaciones y programas de computadora. Está basado en varias técnicas, incluyendo herencia, modularidad, polimorfismo, y encapsulamiento.

SQL (Structured Query Language): Es un lenguaje de programación estándar e interactivo para la obtención de información desde una base de datos y para actualizarla.

HTML (HyperText Markup Language): Es un lenguaje que pertenece a la familia de los "lenguajes de marcado" y es utilizado para la elaboración de páginas web.

XHTML (Lenguaje de Marcado de Hipertexto Extensible): Es una versión más estricta y limpia de HTML, que nace precisamente con el objetivo de reemplazar a HTML ante su limitación de uso con las cada vez más abundantes herramientas basadas en XML

Facelet: Es un sistema de código abierto de plantillas web bajo la Licencia Apache y la tecnología de controlador de Java Server Faces (JSF)

Java Server Faces: es una tecnología que ayuda a los desarrolladores de software a crear páginas web dinámicas basadas en HTML y XML, entre otros tipos de documentos. JSP es similar a PHP, pero usa el lenguaje de programación Java

J2EE: Son las siglas de Java 2 Enterprise Edition que es la edición empresarial del paquete Java creada y distribuida por Sun Microsystems. Comprende un conjunto de especificaciones y funcionalidades orientadas al desarrollo de aplicaciones empresariales

Workflow: Es el estudio de aspectos operacionales de una actividad de trabajo, esto es, cómo se realizan y estructuran las tareas, cuál es su orden correlativo, cómo se sincronizan, cómo fluye la información y cómo se hace su seguimiento.

Micro-framework: Término usado para *framework* que se especializan en desarrollos de escala mínima o aplicaciones web minimalistas

Domain specific language (DSL): es un lenguaje de programación o especificación dedicado a resolver un problema en particular, representar un problema específico y proveer una técnica para solucionar una situación particular.

Multiplataforma: Es un atributo conferido a programas informáticos o métodos y conceptos de cómputo que son implementados e interoperan en múltiples plataformas informáticas

Zope Toolkit (ZTK): Son un conjunto de bibliotecas destinadas a ser reutilizadas por proyectos para desarrollar aplicaciones web o *framework* web.

Bytecode: Código intermedio entre el código fuente y el código máquina. Suele tratarse como un fichero binario que contiene un programa ejecutable similar a un módulo objeto.

GUI (*Graphical User Interface*): Es un entorno que gestiona la interacción con el usuario basándose en relaciones visuales como iconos, menús o un puntero.

CAPÍTULO I. MARCO TEÓRICO

1.1 Antecedentes

El gran volumen de información manejado actualmente por muchas empresas, no es gestionado de manera óptima lo que provoca que dicha información no sea de fácil accesibilidad, integridad y rapidez. Esto conlleva a la demanda de creación de aplicaciones web que permitan al usuario manejar eficazmente dicha información.

El aumento vertiginoso de usuarios de internet y la demanda de utilización de sistemas basados en la web exigen que las aplicaciones web sean desarrolladas de la manera más rápida y eficiente. "(...) la cual debe cumplir con la filosofía web (libertad de acceso, universalidad, y rapidez) siendo capaz de ser utilizada en cualquier tipo de browser (Internet Explorer, Mozilla Firefox, etc.) y en cualquier plataforma (Windows, Linux, Mac)" (Cumba & Barreno, 2012, pág. 26).

En la actualidad, existen diferentes lenguajes de programación que permiten desarrollar estas aplicaciones web tales como Java, PHP, Python, Ruby. Estos lenguajes de programación requieren de una arquitectura de software (*framework*) que faciliten la creación ágil de aplicaciones web. Los *framework* utilizan módulos de software que cuenta con procedimientos, librerías y clases de un lenguaje concreto.

Uno de los principales patrones de diseño utilizados es el Modelo Vista Controlador (MVC). (Fuertes & Guevara, 2010) en su trabajo establecen que el MVC es utilizado en aplicaciones que manejan gran cantidad de datos y transacciones complejas.

1.2 Diseño de arquitectura tecnológicas a gran escala

La implementación de un proyecto macro o a gran escala, se basa en que el desarrollo contemple muchos módulos o vertientes de información y que abarque lo necesario para ser usado por una estructura amplia. En consecuencia, un desarrollo web a gran escala es una metodología aplicada para realizar trabajos con grandes volúmenes de datos en un ambiente geográfico bastante amplio, con el uso de recursos tecnológicos adecuadamente agrupados y configurados para que puedan cumplir con los estándares que demanda el desarrollo e implementación de arquitecturas a gran escala.

La forma en que se determina un gran proyecto o desarrollo, es la forma en que se plantea la idea desde el inicio. Puede realizarse de manera segmentada ocasionando el uso de módulos primarios y avanzando mediante el requerimiento de expansión del sistema.

Para realizar un desarrollo a gran escala se toman en cuenta varios aspectos:

1.2.1 Arquitectura de la información

La arquitectura de la información se la define como la parte en cómo se presenta visualmente la información hacia el usuario, a través de un aplicativo web, es decir la estructura y diseño de una página web. Por medio de esta arquitectura se organiza y se muestra la información al usuario.

Para esta investigación es importante tomar en cuenta que se debe diseñar una buena arquitectura de la información ya que trae consigo beneficios para los usuarios finales, lo cual le hace ser más atrayente, como es una optimización en el tiempo de navegación, reduce costos y minimiza los costos en la búsqueda de información a través de medios primarios.

1.2.2 Arquitectura tecnológica

Cada bloque representa una parte importante del desarrollo sistemático de la web macro y puesto que contiene un alto volumen de información, es importante tomar en cuenta aspectos como los siguientes:

- **Arquitectura de sistemas:** Son los componentes o *hardware*, con características adecuadas para soportar la carga de información y transmisión de datos a través de los protocolos conjuntamente con el sistema operativo, los cuales ofrecerán los recursos para distintas aplicaciones y servicios web a gran escala.
- **Arquitectura de datos:** En conjunto con la base de datos y la información suministrada, la arquitectura de datos se encarga de manejar la manera en que se distribuye la información.
- **Arquitectura de almacenamiento:** Se encarga de las redes de almacenamiento, las cuales admiten almacenar información generada por diferentes aplicaciones web en diferentes sistemas informáticos.
- **Arquitectura de redes:** Es aquella que establece la información a través de las redes de comunicaciones, permitiendo el intercambio de datos en diferentes sistemas de información.
- **Arquitectura de software:** Permiten la interacción entre las aplicaciones, servicios y sistemas de información, dando facilidad a los usuarios para la comunicación a través de ellos.

Cada uno de los elementos mencionados están orientados a la construcción de un desarrollo a gran escala, teniendo en cuenta que cada elemento es una parte de información que se

necesita para la creación de la web. Cada arquitectura establece una tarea que se realiza de manera independiente, pero se juntan al momento de hacer funcionar al proyecto web macro.

1.2.3 ¿Qué es un Clúster?

El desarrollar de un clúster dentro de una infraestructura tecnológica nace de la necesidad de garantizar a los servidores que sus “recursos y aplicaciones de importancia inmersas en un determinado servicio, permanezcan disponibles para los usuarios. (...), y garanticen confiabilidad en los procesos que se efectúen en un momento determinado” (Zumba, 2011, p. 29).

Un clúster de acuerdo a (Buyya, 1999) citado por (Sinisterra, Diaz, & Ruiz, 2012) “es un conjunto de computadoras construidas mediante la utilización de componentes de hardware que se comportan como si fuesen una única computadora”.

1.2.4 Elementos que forman parte de un clúster

De acuerdo a (Gallardo, 2011) estos elementos son:

- Un nodo activo, donde corren los servicios
- Un nodo pasivo que funciona como respaldo (Backup).
- Servidores reales.
- Software de administración.
- Protocolos de comunicación y servicios.
- Conexiones de red.
- Ambientes de programación paralela.
- Middleware.

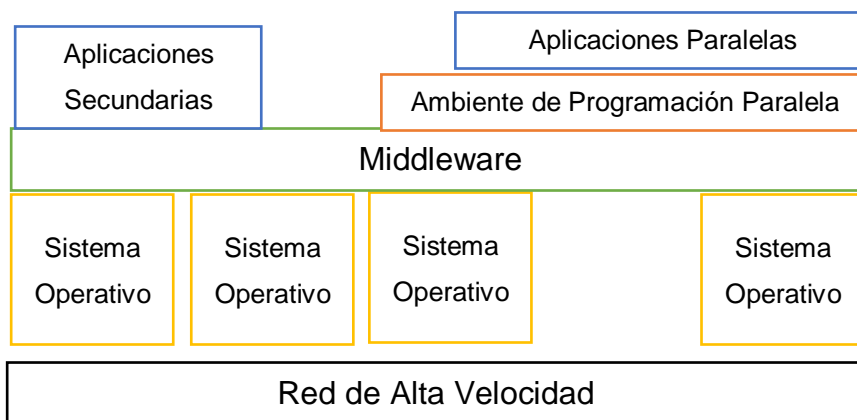


Figura 1: Elementos de un Clúster
Fuente: (Gallardo, 2011)
Elaborado por: El autor

1.2.5 Características y Servicios de los Clúster

La clasificación se realiza con base a sus características;

1.2.5.1 Alto Rendimiento (*High performance*):

Los clústeres de alto rendimiento utilizan los nodos para ejecutar cálculos simultáneos. Un clúster de alto rendimiento permite que las aplicaciones trabajen de forma paralela, mejorando así el rendimiento de éstas. Los clústeres de alto rendimiento son conocidos como clúster computacionales o computación de red (Jiménez & Medina, 2014, p. 17).

1.2.5.2 Alta Disponibilidad (*High availability*):

Los clústeres de alta disponibilidad proporcionan continua disponibilidad de los servicios a través de la eliminación de la falla por un único elemento y a través del proceso de recuperación en contra de fallos al trasladar el servicio desde el nodo de clúster erróneo a otro nodo completamente funcional (Jiménez & Medina, 2014, p. 16).

1.2.5.2.1 Configuración del Clúster de Alta disponibilidad.

- **Configuración Activo – Activo.**

(Bonilla, Vargas, & Meneses, 2009) destacan que se trata de un “Conjunto de componentes computacionales que se encuentran prestando su servicio activamente todo el tiempo”.

En este tipo de configuración, a los dos servidores les llegan solicitudes de servicio.

- **Configuración Activo – Pasivo**

“Conjunto de componentes computacionales que tienen un componente igual asociado, el cual es pasivo y monitorea al nodo activo para en caso de falla retoma los servicios del primero”. (Bonilla, Vargas, & Meneses, 2009)

1.2.5.2.2 Funcionamiento de un clúster de alta disponibilidad

“Los clústeres de alta disponibilidad tienen como propósito principal brindar la máxima disponibilidad de los servicios que ofrecen. Esto se consigue mediante software que monitoriza constantemente el clúster, detecta fallos y permite recuperarse frente a los mismos” (Cujano, 2011, p. 44).

1.2.5.3 Escalabilidad

La escalabilidad es la habilidad que tiene un sistema de adaptarse a un crecimiento de carga sin perder calidad en los servicios ofrecidos. Normalmente la escalabilidad se realiza hacia arriba, es decir, aumentando el tamaño y la potencia del sistema; pero también puede ser hacia abajo, utilizando solo los recursos necesarios en situaciones de baja demanda. (Godino, 2013, p. 23)

De acuerdo a (Zumba, 2011) “Para cuando la demanda de un determinado servicio aumenta, el sistema debe ofrecer la capacidad de ampliarse con la finalidad de atender todas las solicitudes sin que los tiempos de respuesta sean alterados” (p. 73)

Se distinguen dos maneras: “scale in” y “scale out”. El “scale in” o escalado vertical consiste en el incremento de recursos en un solo nodo, ya sea añadiendo memoria, capacidad de procesamiento. etc. El principal problema que existe con el escalado vertical es que es el coste de aumentar los recursos en un solo nodo es exponencial y siempre tiene un límite. El “scale out” o escalado horizontal consiste en el incremento de los nodos en un sistema, de tal manera que al aumentar la demanda de capacidad o de proceso, se añaden más nodos al sistema, sin tener que ampliar la capacidad de estos. (Godino, 2013, p. 23)

La escalabilidad debe formar parte del proceso de diseño porque no es una característica separada que se pueda agregar después. Al igual que con otras funciones de aplicación, las decisiones que se tomen durante las primeras fases de diseño y codificación determinarán en gran medida la escalabilidad de la aplicación. La escalabilidad de una aplicación requiere una pertenencia equilibrada entre dos dominios distintos, software y hardware. Puede avanzar grandes pasos que aumenten la escalabilidad de un dominio sólo para sabotearlos cometiendo errores en el otro. (Vargas & Abril, 2012)

1.2.5.4 Balanceo de Carga

El balanceo de carga permite la división de los procesos o trabajos que realiza un sistema en varios de ellos. (Zumba, 2011) concluye:

Es un clúster que permite que varios servidores compartan la carga de trabajo y de tráfico que se origina de parte de los clientes. Estos se encargan de distribuir las peticiones que reciba el clúster. Este proceso de dividir la carga de trabajo entre los servidores reales permite obtener un mejor tiempo de acceso a las aplicaciones y con ellos tener una mejor confiabilidad del sistema.

De esta forma, se beneficia al sistema permitiendo realizar el mismo trabajo en un tiempo mucho más corto, es decir, permite realizar más carga de trabajo en el mismo tiempo total.

Además, el balanceo de carga prevé el colapso de los sistemas al momento de recibir un volumen de información muy altos, entonces, cuando hablamos de arquitecturas a gran escala, en los cuales la información que se procesa es muy alta, es vital tener un balance de carga que permita manejar el entorno de datos. El factor de división de carga se puede definir, dando más o menos carga a cada uno de los sistemas implicados. Esta característica se llama carga asimétrica.

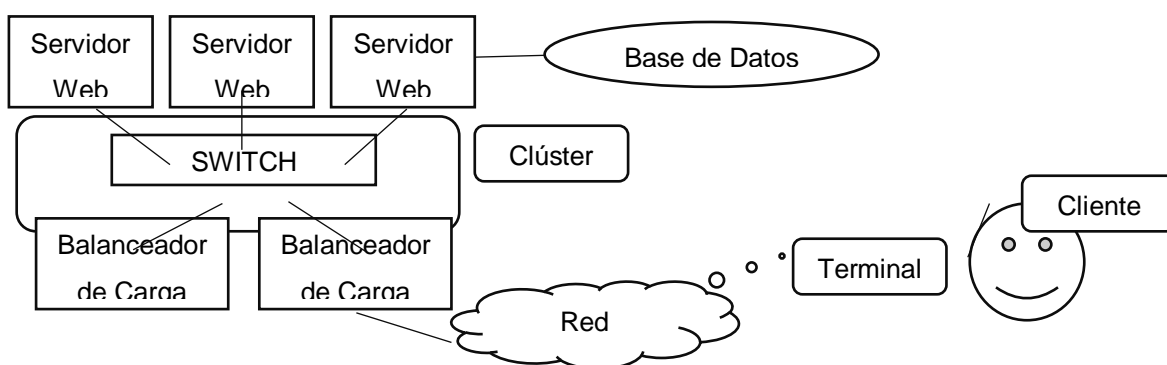


Figura 2: Balanceo de Carga
Fuente: El autor
Elaborado por: El autor

Existen varios tipos de Balanceo de Cargas, entre ellos:

1.2.5.4.1 Balanceo de carga por hardware

Este tipo de balanceo se establece cuando el valor de las rutas toma un recorrido que posee el mismo costo, es decir, cuando se utiliza a varios componentes para compartir el tráfico en partes iguales entre cada camino. De esta manera, este tipo de balanceo puede ser considerado como un clúster basado en redes Ethernet.

Las rutas alternativas están garantizadas para estar libre de bucles, ya que poseen el mismo valor en la ruta principal. Sin embargo, cuando las cargas no están al mismo valor ocurre que el tráfico de información es mucho más complicado y no podría lograrse el balance de carga.

1.2.5.4.2 Balanceo de carga por software

La implementación por medio de *software* facilita varios aspectos como despliegue y mantenimiento, y posee un rendimiento similar al balance de carga por medio de *hardware*.

Algunos *softwares* que realizan este proceso son los presentados en la Tabla 1.

Tabla 1: Software de implementación clúster

	Función	Software
Software para clúster	Alta disponibilidad	Heartbeat Keepalived
	Balanceo de carga	Ultra Monkey Pirahna Open Mosix Haproxy Nginx Linux Virtual Server

Fuente: (Gallardo, 2011)

Elaborado por: El autor

El balanceo de carga puede ser aplicado a:

1.2.5.4.3 Balanceo de Carga sobre el servidor Web

El balanceo de carga sobre el servidor web, consiste básicamente en un método de equilibrio de carga entre varios servidores que componen un clúster, el acceso al clúster se lo realiza por medio del servidor balanceador de carga, el cual recibe todas peticiones de los usuarios, y las distribuye según el algoritmo de balanceo de carga que posee, hacia los servidores reales.

El Frontend o balanceador de carga, puede ser uno o varios, para evitar un único punto de fallo dentro de la granja de servidores, entre sus características se puede anotar las siguientes:

- ✓ Dependiendo del algoritmo de equilibrio de carga que tenga configurado, el balanceador, puede distribuir las cargas equitativamente, por pesos, por distribución según lo ocupado que esté el servidor.
- ✓ Tolerancia a fallos de los nodos servidores HTTP.
- ✓ Chequeo del estado de "salud" de los servidores.
- ✓ Se puede incluir funciones de firewall.

El servidor web apache posee varios complementos para el balanceo de carga, con la condición de que cada módulo que se planea usar debe estar debidamente cargado. Entre

algunas características básicas del servidor apache tenemos la cache, comprensión, reescritura de la URL, y procesamiento de cabeceras. Por otra parte, los algoritmos destacados para el uso del balanceo de carga son:

- ✓ Conteo de solicitud.
- ✓ Trafico ponderado de conteo.
- ✓ Solicitud pendiente de conteo

1.2.5.4.4 Balanceo de Carga de Base de Datos

Las bases de datos con mucha frecuencia tienden a sobre cargarse con información por lo que ocasiona el cambio del equipo, por uno de mejor características generando costos elevados, por este motivo el usar un clúster con las bases de datos se hace simple y económico para crear una solución a ciertas problemáticas.

1.2.5.4.5 Balanceo de Carga de alta disponibilidad

(Jiménez & Medina, 2014) manifiestan:

Los clústeres de alta disponibilidad proporcionan continua disponibilidad de los servicios a través de la eliminación de la falla por un único elemento y a través del proceso de recuperación en contra de fallos al trasladar el servicio desde el nodo de clúster erróneo a otro nodo completamente funcional.

Generalmente, los servicios en los clústeres de alta disponibilidad leen y escriben datos a través de la lectura y escritura a un sistema de archivos montado. Así, un clúster de alta disponibilidad debe mantener la integridad de los datos cuando un nodo recibe el control del servicio desde otro nodo. (p. 16)

Actualmente, cuando se realizan proyectos web a gran escala, se necesita mucho rendimiento y disponibilidad del manejo de la información, esto hace que sea indispensable balancear la carga de procesos. Uno de los propósitos de los sistemas distribuidos es permitir aplicaciones y procesos de servicio para proceder concurrentemente sin competir por los mismos recursos y aprovechar los recursos computacionales disponibles.

1.2.5.4.6 Algoritmos de balanceo de carga

- **Round Robin**

(Zhang, 2000) señala que “El algoritmo de programación Round-Robin dirige las conexiones de red a los diferentes servidores de la manera round-robin. Se trata a

todos los servidores reales como iguales independientemente del número de conexiones o del tiempo de respuesta”

- **Weighted Round Robin**

De acuerdo a (Zhang, 2000) “La programación de round-robin ponderada puede tratar a los servidores reales de diferentes capacidades de procesamiento. A cada servidor se le puede asignar un peso, un número entero que indica su capacidad de procesamiento, el peso predeterminado es 1”.

- **Least-Connection Scheduling**

El algoritmo de programación de conexión mínima dirige las conexiones de red al servidor con el menor número de conexiones activas. Este es uno de los algoritmos de programación dinámica, ya que necesita contar las conexiones activas para cada servidor dinámicamente. En un servidor virtual en el que hay una colección de servidores con un rendimiento similar, la programación de conexión mínima es buena para distribuir suavemente cuando la carga de las solicitudes varía mucho, porque todas las solicitudes largas no se dirigirán a un solo servidor. (Zhang, 2000)

- **Weighted Least-Connection Scheduling**

La programación de conexión mínima ponderada es un superconjunto de la programación de conexión mínima, en la que se puede asignar un peso de rendimiento a cada servidor. Los servidores con un valor de peso más alto recibirán un porcentaje mayor de conexiones activas en cualquier momento. El administrador virtual del servidor puede asignar un peso a cada servidor real y las conexiones de red están programadas para cada servidor en el que el porcentaje del número actual de conexiones activas para cada servidor es una relación con su peso. (Zhang, 2000)

1.3 Arquitectura aplicación web

La arquitectura Web es la encargada de estructurar, organizar y etiquetar el contenido de la información en el proyecto a gran escala, como también establecer los puntos de acceso, sistemas de búsqueda y recuperación de información de cualquier aplicación soportada en la Web misma, con el fin de que el usuario cumpla con los objetos y que su experiencia sea óptima.

1.3.1 Modelo Vista Controlador (MVC)

(Fernández & Díaz, 2012), concluyen:

Fue diseñado para reducir el esfuerzo de programación necesario en la implementación de sistemas múltiples y sincronizados de los mismos datos. Sus características principales están dadas por el hecho de que, el Modelo, las Vistas y los Controladores se tratan como entidades separadas; esto hace que cualquier cambio producido en el Modelo se refleje automáticamente en cada una de las Vistas. Este modelo de arquitectura se puede emplear en sistemas de representación gráfica de datos, donde se presentan partes del diseño con diferente escala de aumento, en ventanas separadas. (p. 48)

La arquitectura MVC es un patrón que separa el Modelo (objeto del negocio), la Vista (interfaz con el usuario y otro sistema) y el Controlador (lógica interna).

De acuerdo con (Fernández & Díaz, 2012)

El Modelo es el objeto que representa los datos del programa. Maneja los datos y controla todas sus transformaciones. (...) La Vista es el objeto que maneja la presentación visual de los datos representados por el Modelo. Genera una representación visual del Modelo y muestra los datos al usuario. (...) El Controlador es el objeto que proporciona significado a las órdenes del usuario, actuando sobre los datos representados por el Modelo, centra toda la interacción entre la Vista y el Modelo. (p. 49)

La característica principal de este modelo es que permite separar la lógica del, interfaz del usuario y de la lógica del trabajo.

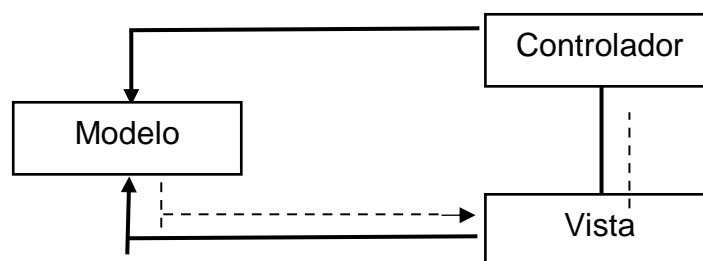


Figura 3: Modelo-Vista-Controlador
Fuente: (Fernández & Díaz, 2012)
Elaborado por: El autor

En la Ilustración 1 se puede visualizar que, en el MVC, tanto el controlador como la vista dependen directamente del modelo, pero este a su vez no depende ni de la vista ni del controlador.

1.3.1.1 Elementos del MVC

1.3.1.1.1 Modelo

“Este modelo es aquel que es realizado por el desarrollador y que contiene todos los datos, es decir toda la información y la funcionalidad del programa” (Molina, Loja, Zea, & Loaiza, 2016).

El modelo normalmente no depende de cómo el programa recoja los resultados o como los presente, de manera que es totalmente independiente. (Fuertes & Guevara, 2010) explican el modelo a través del siguiente ejemplo:

Si se quiere hacer un juego de ajedrez, todas las reglas del ajedrez son totalmente independientes de cómo se va a dibujar el tablero en 3D o plano, con figuras blancas y negras tradicionales o cualquier otro tipo de figura. Este código constituiría el modelo. En el juego de ajedrez el modelo podría ser una clase (o conjunto de clases) que mantengan un registro de piezas, que permita mover dichos segmentos verificando que los movimientos son legales, que detecte los jaques, jaque mate, tablas, etc. De hecho, las metodologías orientadas a objeto nos introducen en este tipo de clases, a las que llaman clases del negocio. (p. 22)

Las principales características del modelo son:

- Consiste en un conjunto de clases y objetos correspondientes al Modelo del Negocio; para la aplicación en esta investigación, estados y funcionalidad.
- Tiene un bajo acoplamiento con vistas y controladores.
- En el MVC se definen métodos para realizar consultas (informar el estado), comandos (modificar el estado) y mecanismos de notificación (para informar a los observadores / vistas).

1.3.1.1.2 Vista

“Una vista es aquella que permite gestionar como los datos se presentarán, es decir como interactúa el usuario final con la interfaz, la cual debe ser amigable para el cliente” (Molina, Loja, Zea, & Loaiza, 2016). En la codificación de las interfaces gráficas, por ejemplo, en el ajedrez, se presentan en 3D o en planos.

Las principales características de este elemento son:

- Administra la visualización y presentación de la información.

- Observa al modelo para actualizar los cambios.
- Al definirse en el modelo una interfaz clara y estable, es fácil implementar múltiples Vistas para un mismo modelo.
- Altamente dependiente del dispositivo y tecnología de visualización.
- Muy dependiente del modelo, es decir debe conocerlo (Fuentes & Guevara, 2010, p. 23).

1.3.1.1.3 Controlador

“En este aspecto toda la información requerida es enviada al gestor de base de datos para ser guardada, es decir controla el acceso a los datos y de esta manera el contenido es de forma estática y dinámica” (Molina, Loja, Zea, & Loaiza, 2016)

Es aquel que maneja las solicitudes realizadas por los usuarios, de manera que pueda responder y procesar información. Si es necesario puede cambiar o modificar el modelo.

Esta codificación no define las ventanas visuales ni las reglas del modelo. En el ejemplo del juego del ajedrez formarían parte de este el algoritmo que se usa para definir los movimientos y jugadas.

Las principales características del controlador son:

- Responsable de definir el comportamiento de la aplicación.
- Recibe los eventos del usuario y decide que es lo que se debe hacer, mapeándolos en comandos (mensajes) hacia el Modelo.
- Altamente dependiente de los dispositivos y mecanismos de interacción del usuario.
- Muy dependiente del Modelo, es decir debe conocerlo (Fuentes & Guevara, 2010, p. 24).

1.3.1.2 Ventajas del MVC

- La implementación se realiza a partir de un conjunto de reglas establecidas en el modelo.
- No se requiere de actualización para el interfaz de vistas debido a que este proceso se realiza automáticamente en la aplicación.
- Las modificaciones hechas al dominio solo implicarían cambios al modelo y a las interfaces del mismo con las vistas.
- Los cambios a las vistas no afectan al modelo-dominio, solo modifica la manera en que se interpreta la información.

- El modelo-vista-control demuestra estar bien diseñado debido a que las aplicaciones que usan este patrón presentan una amplia y estable codificación lo cual las convierte en únicas y la diferencia de otras aplicaciones que usan otros patrones.

1.3.1.3 Desventajas del MVC

Como desventajas se tiene:

- Se requiere mayor dedicación en el inicio del desarrollo ya que se necesita un mayor número de clases que, en otros patrones, no son necesarios. Se diferencia que en la etapa de mantenimiento es mucho más estable, extensible y modificable.
- Se necesita de una arquitectura inicial que contenga por lo menos un mecanismo de eventos, una clase modelo, una clase vista y una clase controlador genérico que realicen las tareas de comunicación, notificación y actualización que posteriormente serán utilizadas para el desarrollo de la aplicación.
- Resulta costosa y difícil su implementación en lenguajes diferentes a patrones de diseño orientado a objetos.

1.3.2 Framework

Con base a la definición de (Acosta, Greiner , Dapozo, & Estayno, 2012) un *framework* puede ser considerado como una aplicación incompleta y configurable a la que se le puede agregar elementos para desarrollar una aplicación concreta.

Esta caracteriza por facilitar el desarrollo de aplicaciones, dependiendo de la programación que se utiliza, normalmente hace un buen uso de las bibliotecas, plantillas y funciones.

1.3.3 Framework desarrollados con MVC

Un *framework* desarrollado con MVC posee una filosofía de trabajo basada en la separación de funciones en la estructura configurable. Cuando se usa este diseño se obtienen muchos beneficios, ya que consiste principalmente en obtener un avance ordenado con códigos reutilizables, aportando un desarrollo funcional y rápido.

En este aspecto, existe una gran cantidad de opciones de *framework* que usan el modelo MVC como estructura base para sus proyectos. A continuación, se muestra una lista de los *framework* más importantes que usan el modelo MVC y el lenguaje de programación que utilizan.

1.3.3.1 Ruby on Rails (Ruby)

Este lenguaje de programación realizado en Ruby, básicamente, es un *framework* para desarrollo de aplicaciones web que acceden a datos a un nivel menos complejo, ya que permite escribir menos códigos, realizando más que muchos otros lenguajes y *framework*. También conocido como *Rails*, este software dogmático aumenta la productividad debido a que su paradigma se centra en que siempre existe una mejor manera de hacer las cosas y está diseñado para fomentar ese tipo de trabajo. (<http://rubyonrails.org/>)

1.3.3.2 Struts (Java / J2ee)

Es un *framework*, el cual ha sido desarrollado por la mano de muchos programadores. Este *Open-source* se ha convertido en una herramienta de desarrollo web, muy amplio trabajado en Java usando el patrón MVC, siendo muy elegante y ajustable.

Este *framework* está diseñado para simplificar el ciclo de desarrollo completo de aplicaciones web, desde la construcción, pasando por la implementación y mantenimiento de aplicaciones a lo largo del tiempo. Además de ser compatible con todas las plataformas de *Java Enterprise*, lo convierte en una herramienta muy potente. (<https://struts.apache.org/>)

1.3.3.3 Catalyst (Perl)

Es un *framework Open-source* basado en *Ruby on Rails*, creado en *Pearl* con estructura MVC para el desarrollo de aplicaciones web fomentando un rápido y limpio diseño. Esta herramienta da robustez y escalabilidad, ya que no hay jerarquías complicadas. (<http://www.catalystframework.org/>)

1.3.3.4 Symfony (PHP)

(Potencier & Zninotto, 2012) Symfony es un completo *framework* diseñado para optimizar, gracias a sus características, el desarrollo de las aplicaciones web. Para empezar, separa la lógica de negocio, la lógica de servidor y la presentación de la aplicación web. Proporciona varias herramientas y clases encaminadas a reducir el tiempo de desarrollo de una aplicación web compleja. Además, automatiza las tareas más comunes, permitiendo al desarrollador dedicarse por completo a los aspectos específicos de cada aplicación. El resultado de todas estas ventajas es que no se debe reinventar la rueda cada vez que se crea una nueva aplicación web. (p. 13)

Es una herramienta diseñada para trabajar con PHP, es compleja debido a que se debe poseer conocimientos amplios en lo que respecta al lenguaje de programación usado en PHP y de la programación orientada a objeto. Este *framework* está inspirado en *django*, *perl*, *rails*, *prado*, entre otros.

1.3.3.5 Java (Frameworks)

Java fue introducido por primera vez en 1995 por *Sun Microsystems*. En este desarrollador los programas se ejecutan en una máquina virtual y posee líneas de clases lo que lo hace que se trate de un lenguaje altamente portátil, es muy apto para una red con ordenadores y sistemas operativos tan heterogéneos como internet.

1.3.3.5.1 Hibernate

Es una iniciativa que utiliza el modelo Mapeo objeto-relacional (ORM) en la plataforma *Java*, siendo esto una aplicación o herramienta. Es de software libre.

1.3.3.5.2 Jsf

Es un *framework* MVC que contiene un conjunto de APIs, que proporcionan un conglomerado de componentes en forma de etiquetas definidas en páginas XHTML, a través del *framework Facelets* (Universidad de Alicante, 2014).

1.3.3.5.3 Struts

Es un *framework* basado en el MVC bajo la plataforma J2EE (*Java 2, Enterprise Edition*), lo que permite que sea compatible con todas las plataformas en donde Java Enterprise esté disponible.

Es un software libre y separa la gestión del *workflow* de la aplicación, del modelo de objetos de negocio y de la generación de interfaz, lo que permite reducir el tiempo de desarrollo de la aplicación.

1.3.3.6 PHP (Frameworks)

PHP es un tipo de programación o lenguaje el cual ha ido evolucionando por el trabajo de muchos desarrolladores, siendo este lenguaje muy popular por ser incrustable en HTML. La herramienta fue desarrollada en el año 1994 por *Rasmus Lerdorf*.

- KumbiaPHP
- Zend Framework
- Kohana

1.3.3.7 Ruby (Frameworks)

Ruby es un lenguaje de programación bastante balanceado debido a que Yukihiro “Matz” Matsumoto, su creador, quiso mezclar partes de sus lenguajes más utilizados, desarrollando una nueva forma de programación que incorpora tanto la funcionalidad como la interactividad. Fue liberado en el año 1995, y desde la fecha ha acumulado muchos desarrolladores que usan esta herramienta para sus proyectos, formando grupos activos con buenas capacidades y llenando las conferencias relacionadas con Ruby.

- Ruby on Rails
- Rack
- Lotus
- Sinatra

1.3.3.8 Python (Frameworks)

Python es uno de los lenguajes de programación que permite desarrollar aplicaciones de manera ágil, clara y sencilla. Fue creado en los años 90 y cuyo nombre está inspirado en el grupo inglés cómico “*Monty Python*”.

Python se trata de un lenguaje que de acuerdo a (Molina, Loja, Zea, & Loaiza, 2016, p. 202) es considerado “de código abierto, utilizable en múltiples plataformas, niveles de organización: código, funciones, clases, entre otras.”

El lenguaje interpretado o de script hace referencia que para su ejecución requiere del uso de un programador intermedio llamado intérprete, en lugar de compilar a lenguaje máquina que pueda comprender y ejecutar directamente el programa. Por su lado, el tipado dinámico se refiere a que no es importante declarar el tipo de dato que va a contener cierta variable. El fuertemente tipado, quiere decir que una variable no puede ser tratada como si fuera de un tipo distinto al que tiene, para ello es necesario convertirla en forma explícita. En cuanto a la multiplataforma quiere decir que está disponible en multitud de plataformas, por lo que no requiere de librerías específicas para que el programa corra en todos los sistemas; y la orientación de objetos es programar de acuerdo a los conceptos del mundo real relevantes para el problema, ya sea en clases u objetos del programa.

Una característica importante de esta herramienta de desarrollo es ser de uso libre, es decir, no se tiene que cubrir costo de licencia alguno. (Molina, Loja, Zea, & Loaiza, 2016) establecen como principales características de este lenguaje, las siguientes:

Tabla 2: Características de Python

CARACTERÍSTICAS	
Legible y elegantes	<ul style="list-style-type: none"> • Imposible escribir código ofuscado.
Simple y poderoso	<ul style="list-style-type: none"> • Soporta objetos y estructuras de datos de alto nivel: strings, listas, diccionarios, etc. • Múltiples niveles de organización código: funciones, clases, módulos y paquetes. • Incluye librerías que contiene un sinfín de clases de utilidad.
Scripting	<ul style="list-style-type: none"> • No tiene que declarar constantes y variables antes de utilizarlas. • No requiere paso de compilación. • Alta velocidad de desarrollo y buen rendimiento.
Sripting	<ul style="list-style-type: none"> • No requiere paso de compilación. • Alta velocidad de desarrollo y buen rendimiento.
Código interoperable	<ul style="list-style-type: none"> • Se puede utilizar en múltiples plataformas (más aún que Java). • Ejecutar Python dentro de una JVM (Jython)
Open Source	<ul style="list-style-type: none"> • Razón por la cual Python sigue creciendo y creciendo.
Propósito general	<ul style="list-style-type: none"> • Puedes hacer en Python todo lo que pueden hacer con C# o Java.

Fuente: (Molina, Loja, Zea, & Loaiza, 2016)

Elaborado por: (Molina, Loja, Zea, & Loaiza, 2016)

1.3.3.8.1 Django

Su objetivo principal es facilitar la creación de sitios web complejos. “Es un framework web de código abierto escrito en Python que permite construir aplicaciones web más rápido y con menos códigos” (Cumba y Barreno, 2012, pág. 38). Su arquitectura está basada en el MVC, no obstante Django utiliza en realidad un paradigma llamado Modelo-Vista-Template.

De acuerdo a (Cumba & Barreno, 2012) las características de Django son:

- Un mapeador objeto-relacional. ORM propio.
- Es un sistema cuya arquitectura es multiplataforma.
- Aplicaciones “enchufables” que pueden instalarse en cualquier página gestionada con Django.
- Una API de base de datos robusta.
- Un sistema incorporado de “vistas genéricas” que ahorra tener que escribir la lógica de ciertas tareas comunes.

1.3.3.8.2 Grok

“Grok es un framework de aplicación web escrito en el lenguaje de programación Python. Actualmente, hay un montón de opciones finas disponibles para los marcos de desarrollo web para Python, y Grok podría ser uno de los menos conocidos.” (De la Guardia, 2010)

1.3.3.8.3 TurboGears 2

(TutorialsPoint, 2016) citado por (Molina, Loja, Zea, & Loaiza, 2016) señalan:

TurboGears es un framework de aplicaciones web de Python, que se compone de varios módulos. Está diseñado alrededor de la arquitectura MVC que son similares a Ruby on Rails o Strut. TurboGears están diseñados para hacer que el desarrollo rápido de aplicaciones web en Python más fácil y más soportable. TurboGears es un framework de aplicaciones web escrito en Python. TurboGears sigue el paradigma Modelo Vista Controlador al igual que los marcos web más modernos como Rails, Django, puntales, etc.

1.3.3.8.4 Web2py

(Molina, Loja, Zea, & Loaiza, 2016) establecen que Web2py es un Framework de código abierto “(...) para el desarrollo de aplicaciones web seguras que están conectadas a bases de datos y a su vez están programadas en Python, estos a su vez contiene todos los componentes necesarios para su desarrollo.”

1.3.3.9 Selección del Framework Django para el desarrollo de este proyecto

El Framework Django, es del tipo de Framework web de alto nivel, ideal para el desarrollo de grandes proyectos, es decir desarrollo de proyectos a gran escala. Pero cuál es la razón o razones de por qué utilizarlo, a continuación, se detalla cuáles son estos motivos:

- Se enfoca en la re-utilización de componentes, además que todo su lenguaje está escrito en Python.
- Posee su propia librería de mapeo Objeto-Relacional, que conoce a fondo como realizar las configuraciones entre la base de datos y el código para reducir al mínimo el tener que escribir sentencias SQL como sea posible.
- Cuenta con un archivo de configuración “settings.py”, el cual se ajusta perfectamente al entorno de desarrollo que maneja nuestro proyecto a gran escala. Este punto es muy rescatable ya que, dentro de la arquitectura tecnológica, la app web recibe las peticiones o carga de trabajo a través de los servidores balanceadores de carga, y

estos trabajan con una dirección ip Virtual (Ip Virtual 192.168.0.21, para Load Balancer Base de Datos,), para distribuir equitativamente la carga hacia los servidores Backend. De esta manera la Ip Virtual se la define en el archivo “settings.py”, así como la plataforma de Base de Datos y el Wsgi.

- Bibliotecas HTTP, las cuales conocen como analizar las solicitudes web entrantes.
- Bibliotecas de enrutamiento que permiten definir exactamente que URL mapear con que parte de nuestro código.
- Posee un sistema de plantillas que permite a los no programadores escribir HTML mezclado con datos y permitirles generar código con la cantidad justa de lógica de presentación.
- Es un Framework usado para aplicaciones de alto rendimiento.

En definitiva, se concluye que el Framework que está realmente especializado en aplicaciones de alto rendimiento es Django, mismo que tiene como lenguaje base a Python. Este Framework maneja plazos de programación muy intensos, además de utilizar un archivo de configuración “settings.py” (para todo el aparato de conexión con el Servidor Web y Servidor de Base de Datos), y por otro lado la reutilización de componentes; lo que ocasiona que tenga un diseño más limpio y pragmático. Adicional a esto, se menciona, que es un Framework Opensource fácilmente configurable en sistemas operativos libres, y el adquirir librerías o componentes es totalmente gratis. Todas estas bondades brindadas por el Framework Django ayudaron de gran manera al desarrollo del proyecto.

1.3.4 Introducción a Python – Django

1.3.4.1 Lenguaje de programación en Python

(Cumba & Barreno, 2012) citado por (Molina, Loja, Zea, & Loaiza, 2016) señalan:

Un lenguaje de programación es un lenguaje que puede ser utilizado para controlar el comportamiento de una máquina, particularmente una computadora. Consiste en un conjunto de reglas sintácticas y semánticas que definen su estructura y el significado de sus elementos, respectivamente.

(Chavez & Buñay, 2008) citado por (Molina, Loja, Zea, & Loaiza, 2016) concluyen “Python es un lenguaje de programación interpretado e interactivo, capaz de ejecutarse en una gran cantidad de

plataformas. Se desarrolla como un proyecto de código abierto, administrado por PYTHON Software Foundation.”

(Molina, Loja, Zea, & Loaiza, 2016) manifiestan:

Es un lenguaje de programación de código abierto que permite la ejecución en diversas plataformas, los usuarios que utilizan este lenguaje lo consideran el más elegante y a su vez amigable para la programación web, el principal objetivo de este lenguaje es buscar la factibilidad tanto para la lectura como el diseño, al ser un lenguaje multiparadigma brinda innumerables beneficios al permitir al usuario trabajar bajo varios estilos: programación orientada a objetos, programación funcional, entre otros. Otro aspecto importante a considerar es que permite la facilidad de extensión esto quiere decir que se puede escribir nuevos módulos de manera fácil en bajo lenguaje como C o C++ y se puede incluir para aplicaciones que necesiten una interfaz programable.

1.3.4.2 Historia

De acuerdo a (Challenger, Díaz, & Becerra, 2014)

Python fue creado por Guido van Rossum, un programador holandés a finales de los 80 y principio de los 90 cuando se encontraba trabajando en el sistema operativo Amoeba. Primariamente se concibe para manejar excepciones y tener interfaces con Amoeba como sucesor del lenguaje ABC.

El 16 de octubre del 2000 se lanza Python 2.0 que contenía nuevas características como completa recolección de basura y completo soporte a Unicode. Pero el mayor avance lo constituye que este comenzó a ser verdaderamente desarrollado por la comunidad, bajo la dirección de Guido.

El Python 3.0 es una versión mayor e incompatible con las anteriores en muchos aspectos, que llega después de un largo período de pruebas el 3 de diciembre del 2008. Muchas de las características introducidas en la versión 3 han sido compatibilizadas en la versión 2.6 para hacer de forma más sencilla la transición entre estas. (p. 3)

1.3.4.3 Características

Con una metodología multiparadigmática, no obliga al programador adoptar un estilo de programación, sino, que éste se ajusta a varios tipos de lenguaje de programación. Usa un tipado dinámico, realiza un conteo de informes para la gestión de memoria y, además, realiza una resolución dinámica de nombres.

Conjuntamente, posee otras características apreciables como:

- Es bastante simple de usar
- Comprende una metodología muy sencilla de aprender
- Es un *Open-source*
- Usa funciones y lenguaje de alto nivel
- Fácilmente descargable
- Su interpretado no es robusto
- Está orientado a objeto
- Siempre se está desarrollando
- Es incrustable a otras aplicaciones
- Posee librerías externas

1.3.4.4 Elementos del lenguaje

Diseñado como un lenguaje de programación de alto nivel, *Python* fue creado para ser leído con facilidad, cambiando algunos símbolos por operadores lógicos, usando el contenido de los bloques de códigos antes de cada línea de ordenes pertenecientes al mismo, no dando límites a los tabuladores para el indexo de caracteres; y lo más curioso de este lenguaje es que se rige por leyes que permiten conocer el interior de los códigos, siendo este un lenguaje filosófico y reflexivo.

1.3.4.5 Variables

Una variable es básicamente una forma de guardar datos que pueden ser modificados. Las variables se definen de forma dinámica, cada una posee características particulares con un propósito específico, para definir una variable en *Python* se usa el "=" seguido de un valor. Por ejemplo:

```
x = 0
```

x = 'texto' # los dos tipos de datos se pueden manejar gracias a que la variable que se usa en Python son del tipo dinámica

1.3.4.6 Tipos de datos

Python posee una gran variedad de tipos de datos, entre ellos tenemos:

Tabla 3: Tipos de datos Python

Tipo	Clase	Notas	Ejemplo
Str	Cadena	Inmutable	'Cadena'
unicode	Cadena	Versión Unicode de str	u 'Cadena'
List	Secuencia	Mutable, puede contener objetos de diversos tipos	[4,0, 'Cadena', True]
Tuple	Secuencia	Inmutable, puede contener objetos de diversos tipos	[4,0, 'Cadena', True]
Set	Conjunto	Mutable, sin orden, no contiene duplicados	set([4,0, 'Cadena', True])
frozenset	Conjunto	Inmutable, sin orden, no contiene duplicados	Frozenset([4,0, 'Cadena', True])
Dict	Mapping	Grupo de pares clase valor	{'key1':1,0, 'key2':False}
Int	Numero entero	Precisión fija, convertido en <i>long</i> en caso de <i>overflow</i>	42
Long	Numero entero	Precisión arbitraria	42L o 456966786151987643L
Float	Numero decimal	Coma flotante de doble precisión	3.1415927
Bool	Booleano	Valor booleano verdadero o falso	True o False

Fuente: (Cumba & Barreno, 2012)

Elaborado por: El autor

1.3.4.7 Funciones

En *Python* las funciones son definidas con una palabra clave "def" en conjunto de la función lógica y los parámetros de ella. Se puede escribir también con la palabra clave "lambda", aunque su uso es menor que el de la palabra "def".

```
def estoes_mifuncion ():
```

```
    # Este es el Algoritmo
```

Para que la función pueda ejecutarse, debemos hacer el llamado del código.

```
def estoes_mifuncion ():
```

```
    print "Hola Mundo"
```

```
función_1()
```

Hay varios aspectos de las funciones cuando se le hace retorno de datos, cuando una función hace retorno puede ser añadida como una variable.

```
def función_1():
```

```
    return "Hola Mundo"
```

```
palabra = función_1()
```

`print` palabra

1.3.4.8 Módulos

Los módulos son mini códigos o mini comandos que existen como recursos para el desarrollador, constan de muchas propiedades que se pueden añadir al lenguaje importando módulos que llaman a los recursos del sistema. Una forma de agregar los códigos es escribiendo “*import*” seguida del nombre del módulo que se usara.

Los códigos en Python para apagar, reiniciar o cerrar la sesión en el sistema operativo son:

`Import subprocess` # se llama al módulo o función del sistema

```
# Apagar el sistema operativo
subprocess.call ("shutdown -s")
# Reiniciar el sistema operativo
subprocess.call ("shutdown -r")
# Cerrar sesión del sistema operativo
subprocess.call ("shutdown -l")
```

Sin embargo, para hacer el llamado a módulos del sistema operativo no solo puede usarse el “subprocess” también, podemos usar “os” y el “sys”.

1.3.4.9 Framework Web Django para Python

1.3.4.9.1 Descripción

Django es un *Framework* escrito en *Python*, el cual es un *Open-source* usado para la creación de páginas web. Este posee una gran cantidad de componentes que ayudan al programador a crear sus proyectos web fácilmente; además *Django* usa el Modelo-Vista-Controlador (MVC).

La meta fundamental de *Django* es facilitar la creación de sitios web complejos. Django pone énfasis en el re-uso, la conectividad y extensibilidad de componentes, el desarrollo rápido y el principio “No lo repitas” (*DRY*, del inglés *Don't Repeat Yourself*). *Python* es usado en todas las partes del framework, incluso en configuraciones, archivos, y en los modelos de datos. (Cumba & Barreno, 2012, p. 58)

1.3.4.9.2 Características

Los orígenes de Django en la administración de páginas de noticias son evidentes en su diseño, ya que proporciona una serie de características que facilitan el desarrollo rápido de páginas orientadas a contenidos. Por ejemplo, en lugar de requerir que los desarrolladores escriban controladores y vistas para las áreas de administración de la página, *Django* proporciona una aplicación incorporada para administrar los contenidos, que puede incluirse

como parte de cualquier página hecha con *Django* y que puede administrar varias páginas hechas con *Django* a partir de una misma instalación; la aplicación administrativa permite la creación, actualización y eliminación de objetos de contenido, llevando un registro de todas las acciones realizadas sobre cada uno, y proporciona una interfaz para administrar los usuarios y los grupos de usuarios (incluyendo una asignación detallada de permisos). (Cumba & Barreno, 2012, p. 58)

Características importantes de *Django*:

- Es para el desarrollo Web
- Es un *Open-source*, por lo que su código está abierto
- Facilita la creación de páginas web.
- Usa menos códigos que muchos otros *Frameworks*.
- *DRY (Don't Repeat Yourself)*.
- Usa un código muy fácil de interpretar. Es muy legible.
- Es Multiplataforma.
- Posee un gran sistema de plantillas
- Tiene aplicaciones adaptables a cualquier plataforma desarrollada en *Django*.
- Incluye traducciones incorporadas.
- Posee toda su documentación en la nube, es descargable y de fácil acceso.

1.3.4.9.3 Arquitectura

Está basada en el patrón MVC; sin embargo, *Django* define una estructura un tanto diferente llamada *Modell-View-Template*.

Django ejecuta el patrón MVC completamente siguiendo toda su estructura y aplicando su metodología. Por lo tanto, es llamado *Framework* con patrón MVC; aunque someramente, la M, V y C se separan en Django de la siguiente manera:

- M: información de acceso a la base de datos
- V: selecciona los datos a mostrar y cómo mostrarlos, es manejada principalmente por la vista y las plantillas.
- C: esta delega a la vista dependiendo de la información de entrada que proporcione el usuario.

Puesto que la “C” es controlado por el mismo *Framework*. *Django* es conocido también como un Framework del tipo MVT. En el patrón de diseño MVT.

- M significa "*Model*" (Modelo), la capa de acceso a la base de datos.
- T significa "*Template*" (Plantilla), la capa de presentación.
- V significa "*View*" (Vista), la capa de la lógica de negocios.

Como se puede apreciar *Django*, usa otro esquema de modelo porque es visto como un *Framework* de MVC. Básicamente usa una forma de patrón creada con base del MVC, este patrón se diferencia por pequeñas características y por ello que aún es considerado como un *Framework* MVC y no MVT. Aunque eso lo caracteriza y da particularidad a esta herramienta de desarrollo tan confiable y práctica.

1.3.4.9.4 Soporte de base de datos

Django tiene la particularidad que cada uno de sus modelos de datos tiene su comunicación en una tabla de base de datos. Esto hace que el Framework soporte los requerimientos del mercado actual simplemente cambiando variables. *DATABASE_ENGINE* en el *settings.py*.

Django admite cuatro gestores de base de datos:

- *PostgreSQL*
- *SQLite3*
- *MySQL*
- *Oracle*

Además, de estas bases de datos que son oficiales también están otras las cuales fueron configuradas por terceros para que el Django pueda trabajar con ellas.

- *Sybase SQL Anywhere*
- *IBM DB2*
- *Microsoft SQL Server 2005*
- *Firebird*
- *ODBC*

Hay que tener en cuenta que las bases de datos solo funcionan para algunas versiones del *Django* y dependiendo el ORM. Si no se consideran estos factores, pueden dar error al momento de su uso.

1.3.4.9.5 Servidor Web

Django incluye un servidor web ligero que se puede usar mientras se está desarrollando el sitio. Incluimos este servidor para que puedas desarrollar tu sitio rápidamente, sin tener que lidiar con configuraciones de servidores web de producción (i.e., Apache) hasta que estés listo para la producción. Este servidor de desarrollo vigila tu código a la espera de cambios y se reinicia automáticamente, ayudándote a hacer algunos cambios rápidos en tu proyecto sin necesidad de reiniciar nada. El servidor de desarrollo es extremadamente útil para, bueno, desarrollar, resiste la tentación de usar este servidor en cualquier entorno parecido a producción. El servidor de desarrollo puede manejar fiablemente una sola petición a la vez, y no ha pasado por una auditoría de seguridad de ningún tipo. (Cumba & Barreno, 2012, p. 62)

1.3.4.9.6 Requerimientos

- Instalar el Lenguaje de Programación Python
- Instalar Django
- Instalar un motor de Base de Datos
- Instalar editor de código SublimeText2

1.4 Sistema Operativo

(Silberschatz, Galvin, & Gagne, 2005) establecen que “Un sistema operativo es un programa que administra el hardware de una computadora, también proporciona las bases para los programas de aplicación y actúa como un intermediario entre el usuario y el hardware de la computadora”.

Sistema operativo es la conformación de algunos programas necesarios para administrar los recursos del computador, smartphone, Tablet, etc., posibilitando entre otras cosas, la comunicación con el usuario y de esta manera recibir órdenes de él; así como lectura y escritura en el hard disk, memoria usb, etc., ejecutar software instalado, recibir órdenes de impresión; controlar el uso de la memoria, entre otras cosas.

1.4.1 Software Libre

(Stallman, 2004) “Con software libre nos referimos a la libertad de los usuarios para ejecutar, copiar, distribuir, estudiar, cambiar y mejorar el software” (p. 45).

(Da Rosa & Heinz, 2007) señalan:

Para ser considerado libre, un programa debe ser distribuido de tal modo que el usuario pueda, entre otras cosas, estudiar el modo de funcionamiento del programa, adaptarlo

a sus necesidades y distribuir, bajo las mismas condiciones, programas derivados. Para que estas libertades sean practicables, no basta con que la licencia del programa las permita. Además, es necesario que el código fuente del programa esté a disposición del usuario, ya que de lo contrario las tareas de comprender, adaptar y mejorar el programa se vuelven tan complicadas que es casi lo mismo que si estuvieran prohibidas. (p. 25)

1.4.1.1 Windows

Microsoft Windows se constituye como un sistema operativo, dicho en otras palabras, un grupo de programas que permite la gestión de los recursos de un computador ó en su defecto un dispositivo inteligente. Este sistema comienza a operar apenas se enciende el equipo, administrando los recursos del mismo, entre software y hardware. Es un sistema operativo de tipo propietario.

1.4.1.2 Linux

Definido como un sistema operativo de Open Source, que surge como una alternativa de código abierto frente al muy conocido y poderoso Windows, posee gran cantidad de distribuciones, y es muy potente en sus versiones para servidor.

1.4.1.3 Mac Os

Sus siglas en inglés Macintosh Operating System, es un sistema operativo de tipo propietario perteneciente a la familia de las Apple.

Tabla comparativa de los sistemas operativos

En la tabla 3 se detalla la comparación de varios sistemas operativos, características, ventajas y desventajas para el desarrollo de diferentes aplicaciones y sustentabilidad.

Tabla 4: Comparación de diferentes sistemas operativos, ventajas y desventajas

Sistemas operativos		
Características		
Windows	Linux	Mac OS X
<ul style="list-style-type: none"> ▪ Interfaz gráfica con menús desplegados, ventanas en cascada y soporte para mouse. ▪ Gráficos de pantalla e impresora independientes del dispositivo. ▪ Multitarea cooperativa entre las aplicaciones Windows. ▪ Ventanas traslapadas ▪ Archivos PIF para aplicaciones DOS ▪ Modo estándar (286), con soporte de memoria grande (largo, menor). ▪ Modo Mejorado 386, con memoria grande y soporte de múltiples sesiones DOS. 	<ul style="list-style-type: none"> ▪ Linux es uno de los sistemas operativos más robustos, estables y rápidos ▪ El manejo de la memoria de Linux evita que los errores de las aplicaciones detengan el núcleo de Linux ▪ Linux es multitarea y multiusuario: Esta característica imprescindible está en Unix desde su concepción, pero le llevó a Microsoft más de 20 años ofrecerlo en su sistema operativo de consumo ▪ Linux soporta gran variedad de entornos gráficos (KDE, GNOME, XFCE...) ▪ Hay miles de programas libres para Linux, adaptados a muy diversos propósitos y disponibles en internet para usarlos con GNU/Linux ▪ Linux permite navegar por Internet y conectar máquinas en red de manera natural (los protocolos TCP/IP o PPP por ejemplo, están incluidos como un módulo del básico del núcleo) ▪ Casi cualquier aplicación Unix puede usarse bajo Linux 	<ul style="list-style-type: none"> ▪ En las disqueteras también se han acercado al mundo PC, y aceptan los formatos de 720Kb y 1,4Mb propios de dichas máquinas, así como el original del Mac de 800 Kb. ▪ Con los monitores otro tanto de lo mismo, aceptan sin ningún rubor monitores VGA estándar. ▪ Las memorias son DIMM de 168 contactos, y los discos de tecnología SCSI II. ▪ Cuentan con un ratón de un sólo botón. ▪ Las impresoras también son específicas de esta plataforma, aunque algunas soportan tanto a una como a otra (como algunas EPSON de inyección de tinta).

	<ul style="list-style-type: none"> ▪ Para Linux existe gran cantidad de documentación libre, aunque no siempre está traducida ▪ Las libertades de copia y modificación permiten usar GNU/Linux para facilitar servicios sin depender de terceros ▪ Al poder descargarse Linux de internet, el precio de las distribuciones debe mantenerse competitivo con el hágalo usted mismo y por lo tanto resulta un precio justo ▪ Pero no solamente el precio de adquisición de Linux es menor, el de implantación (debido a la posibilidad de emplearlo en tantas máquinas como se desee) también lo es, así como el Coste Total de Propiedad de Linux 	
Ventajas		
Windows	Linux	Mac OS X
<ul style="list-style-type: none"> ▪ Es más conocido ▪ Es el que tiene más software desarrollado... 	<ul style="list-style-type: none"> ▪ Linux es muy robusto, estable y rápido: Ideal para servidores y aplicaciones distribuidas. A esto se añade que puede funcionar en máquinas humildes: Linux puede correr servicios en un x86 a 200 MHz con calidad ▪ Linux es libre: Esto implica no sólo la gratuidad del software, sino también que Linux es modificable y que Linux tiene una gran cantidad de aplicaciones libres en Internet. Todo ello arropado por la inmensa documentación de Linux que puede encontrarse en la Red ▪ Linux ya no está restringido a personas con grandes conocimientos de informática: Los desarrolladores de Linux han hecho un gran esfuerzo por dotar al sistema de asistentes de configuración y ayuda, además de un 	<ul style="list-style-type: none"> ▪ Mejor interfaz gráfica del mercado ▪ Ideal para diseño gráfico. ▪ Es muy estable

	sistema gráfico muy potente. Distribuciones Linux como Red Hat/Fedora tienen aplicaciones de configuración similares a las de Windows	
Desventajas		
Windows	Linux	Mac OS X
<ul style="list-style-type: none"> ▪ El costo es muy alto ▪ Las nuevas versiones requieren muchos recursos ▪ La mayoría de los virus están hechos para win ▪ Puedes tener errores de compatibilidad en sistemas nuevos. ▪ Históricamente es más inestable de los 3 	<ul style="list-style-type: none"> ▪ Windows es incompatible con Linux: Este punto es difícil de explicar: no quiere decir que no podamos tener instalados ambos Sistemas (que es relativamente fácil de hacer) ▪ Uno de los problemas es que desde Windows no podremos escribir en particiones Linux o que desde Linux no podremos escribir (en sentido amplio) en particiones NTFS (Windows XP, 2000...) aunque esto último se está investigando ▪ En la mayoría de distribuciones Linux hay que conocer nuestro Hardware a la hora de instalar ▪ Sin embargo, distribuciones de Linux como Knoppix reconocen todo el sistema a lo Windows ▪ No sólo eso, en este sentido se está trabajando mucho por hacer esta tarea simple 	<ul style="list-style-type: none"> ▪ Costoso (aunque viene incluido con la maquina) ▪ Existe poco software para este sistema operativo. ▪ Es más complicado encontrar gente que la pueda arreglar en caso de fallas

Fuente: (Gallardo, 2011)
 Elaborado por: El autor

La presente indica la factibilidad de usar Linux como sistema operativo base para el proyecto, ya que el sistema operativo Linux posee grandes ventajas muy remarcadas, y en diferentes áreas. La velocidad de rendimiento es por mucho superior debido a los recursos mínimos que solicita Linux para su uso, además, posee una alta seguridad para virus, malware o spyware. Es bastante estable como unidad de trabajo, no es fácil que presente errores de sistema o de programas.

Para este proyecto se usó una de la distro más conocidas de Linux y más utilizadas, la cual es Ubuntu; posee una pantalla visual muy amigable, además de ser robusta, segura y muy sencilla de usar. Ubuntu es OpenSource por lo cual no tiene ningún costo al usarla, además posee gran compatibilidad con Python – Django, nos proporciona alta seguridad de la información en la BD.

1.5 Base de datos

Una colección compartida de datos lógicamente relacionados, junto con una descripción de estos datos, que están diseñados para satisfacer las necesidades de información de una organización. (Conolly, 2005)

Otra definición indica que:

Una base de datos de un SI es la representación integrada de los conjuntos de entidades instancia correspondientes a las diferentes entidades tipo del SI y de sus interrelaciones. Esta representación informática (o conjunto estructurado de datos) debe poder ser utilizada de forma compartida por muchos usuarios de distintos tipos. (Camps Paré, 2007, p. 8)

(Camps Paré, 2007) concluye que: “una base de datos es un conjunto estructurado de datos que representa entidades y sus interrelaciones. La representación será única e integrada, a pesar de que debe permitir utilizaciones varias y simultáneas”. (p. 8)

1.5.1 Sistema de Gestión de base de datos

Un sistema software que permite a los usuarios definir, crear, mantener y controlar acceso a la base de datos. El SGBD es el software que interactúa con los programas de aplicación del usuario y con la base de datos. (Conolly, 2005)

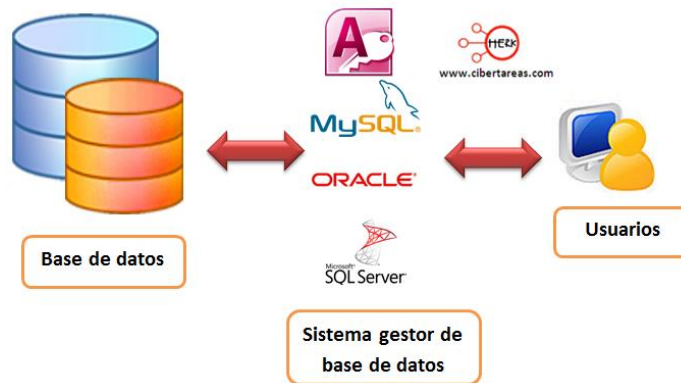


Figura 4: Replicación Maestro-esclavo
 Fuente: <https://sites.google.com/site/softwaredeaejecutiva/1-6-bases-de-datos>

1.5.2 Mysql

MySQL es un sistema de administración de base de datos relacionales rápido, sólido y flexible. Es ideal para crear bases de datos con acceso desde páginas dinámicas, para la creación de sistemas de transacciones on-line o para cualquier otra solución profesional que implique almacenar datos, teniendo la posibilidad de realizar múltiples y rápidas consultas. (Cobo, Gómez, Pérez, & Rocha, 2005, p. 339)

1.5.3 Características

Al principio, MySQL presentaba carencia de algunos componentes que son característicos y fundamentales en las bases de datos relacionales, como por ejemplo integridad referencial y transacciones. No obstante, ha ganado popularidad y acogida dentro de los programadores debido a la sencillez y facilidad de uso de este gestor de base de datos.

MySQL está incorporando características que antes no poseía, trabajo realizado por desarrolladores internos, y por desarrolladores de software open source. Dentro de estas características podemos mencionar las siguientes:

A consideración de (Oracle Corporation, 2014), estas son:

- Escrito en C y en C++
- Probado con un amplio rango de compiladores diferentes
- Funciona en diferentes plataformas.
- Usa GNU Automake, Autoconf, y Libtool para portabilidad.

- APIs disponibles para C, C++, Eiffel, Java, Perl, PHP, Python, Ruby, y Tcl. Consulte
- Uso completo de multi-threaded mediante threads del kernel. Pueden usarse fácilmente multiple CPUs si están disponibles.
- Proporciona sistemas de almacenamiento transaccionales y no transaccionales.
- Usa tablas en disco B-tree (MyISAM) muy rápidas con compresión de índice.
- Relativamente sencillo de añadir otro sistema de almacenamiento. Esto es útil si desea añadir una interfaz SQL para una base de datos propia. (p. 6)

1.5.4 Replicación Mysql

La replicación permite que los datos de un servidor de base de datos MySQL (el maestro) se copien a uno o más servidores de bases de datos MySQL (los esclavos). La replicación es asíncrona de forma predeterminada; Los esclavos no necesitan estar conectados permanentemente para recibir actualizaciones del maestro. Dependiendo de la configuración, puede replicar todas las bases de datos, bases de datos seleccionadas o incluso las tablas seleccionadas dentro de una base de datos. (Oracle Corporation, 2017)

1.5.4.1 Replicación maestro / esclavo

En este tipo de replicación todas las transacciones que se realicen en el nodo Maestro, son replicadas automáticamente en el Esclavo. Este tipo de replicación es útil cuando necesitamos realizar operaciones de selección, inserción, actualización o eliminación de datos en el Maestro, mientras que el servidor Esclavo está pensado únicamente para realizar operaciones de selección. Si en el Esclavo realizáramos cualquier otra operación que no fuera de selección de datos, estos cambios no se verían reflejados en el Maestro, cosa que sí ocurre a la inversa.

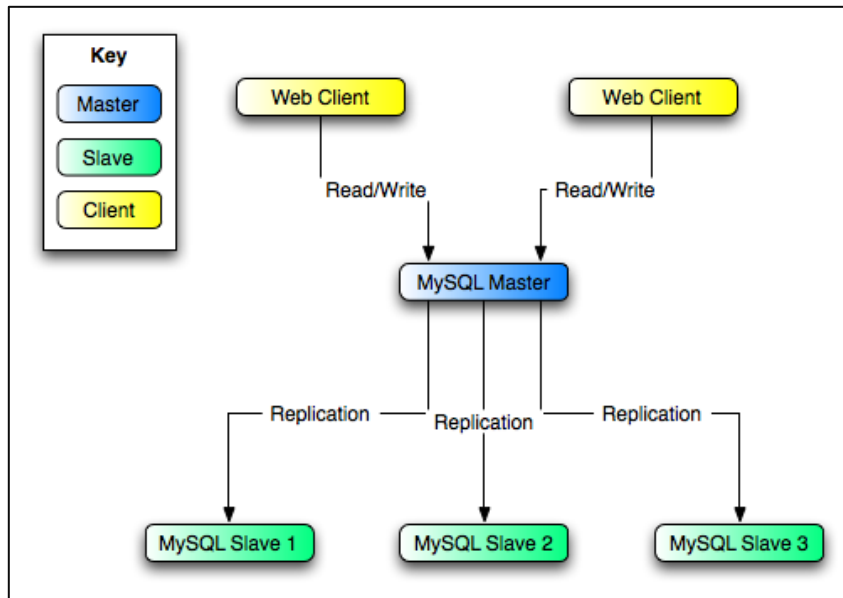


Figura 5: Replicación Maestro-esclavo 2
 Fuente: <https://www.boksar.info/?p=423>
 Elaborado por: <https://www.boksar.info/?p=423>

1.5.4.2 Replicación maestro / maestro

Este tipo de réplica es muy similar a la maestro/esclavo, con la diferencia que ambos nodos cumplen 2 funciones de maestro y esclavo.

1.6 Software y Herramientas para clúster

1.6.1 OpenMosix

“OpenMosix es un software que extiende el kernel linux para que los procesos puedan migrar de forma transparente entre las diferentes máquinas dentro de un clúster con el fin de distribuir de manera más uniforme la carga de trabajo”. (Sloan, 2004, p. 65)

1.6.2 Piranha

(Wangsmo, 1999) citado por (Zhang, 2000) establece:

Es la agrupación de productos de Red Hat Inc., que se distribuye en Red Hat Linux 6.1 distribución o después. Piranha incluye el parche del kernel LVS, el clúster Demonios de monitor y herramientas administrativas GUI.

El nanny daemon debe monitorear los nodos del servidor y servicios correspondientes en

el clúster y el daemon de pulso es controlar los nanny daemons y manejar la conmutación por error entre dos equilibradores de carga. El daemon de impulsos se bifurca Un nanny daemon para cada servidor real. Cuando el nanny daemon no puede recibir la respuesta del servidor nodo y el servicio en el tiempo especificado, se eliminará el servidor de la tabla de programación en el kernel. El nanny Daemon también adapta el peso del servidor con respecto a los nodos del servidor con el fin de evitar que el servidor está sobrecargado. Cuando nanny encuentra que la carga del servidor es mayor que el valor normal, disminuirá el peso del servidor en la tabla de programación en el kernel, para que el servidor reciba menos conexiones, viceversa.

1.6.3 Haproxy

(Tarreau, 2012) citado por (Rodríguez Castellanos & Rodríguez Castiblanco, 2015) establece que “Es una aplicación gratuita y de código abierto Linux, ofrece alta disponibilidad, balanceo de carga. Proxy para TCP y aplicaciones HTTP. Fue especialmente diseñado para sitios web de muy alto tráfico”.

Según (López, 2016) Haproxy,

Es un servidor proxy de alta disponibilidad para servicios TCP y HTTP. Su uso más común es para mejorar el rendimiento y la fiabilidad de un entorno servidor mediante la distribución de la carga de trabajo a través de múltiples servidores (por ejemplo en aplicaciones de web, correo electrónico, base de datos). (p. 15)

Haproxy es un potente software de código abierto, que cuenta con capacidades de alta disponibilidad, balanceo de carga y proxy para trabajar con conexiones TCP y HTTP, ideal para para aplicaciones web que demandan una alta carga de trabajo y alto rendimiento en su desempeño.

Esta herramienta HAProxy (High Availability Proxy), se ha convertido en un popular software open source que se puede ejecutar en Linux, Solaris y FreeBSD. Entre sus principales características está la de mejorar el rendimiento y la fiabilidad de un entorno de servidor mediante la repartición de la carga de trabajo a través de varios servidores (web, aplicación, base de datos). Es utilizado en varios entornos de alto rendimiento, entre ellos: Twitter, GitHub, Imgur, Instagram.

La ejecución y forma de actuar de Haproxy, hacen que esta herramienta pueda integrarse de manera sencilla con las diferentes arquitecturas que hay para desarrollos a gran escala, brindando seguridad a los servidores web delicados, hacia la web.

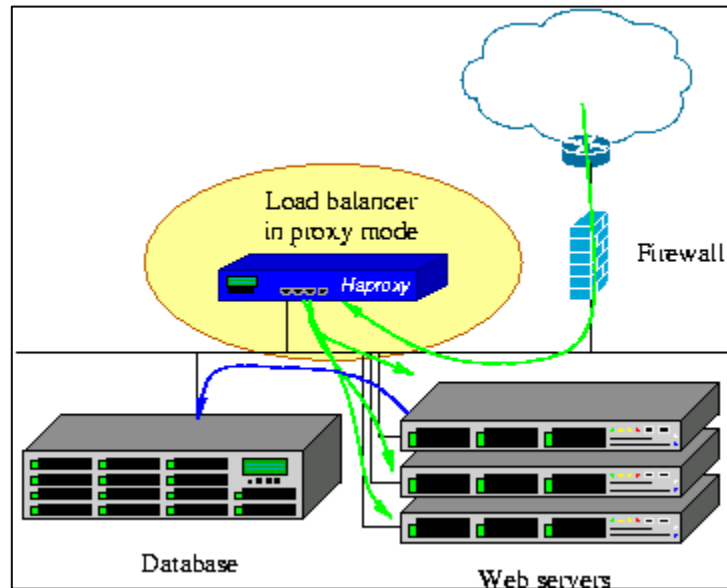


Figura 6: Haproxy-esquema de funcionamiento
Fuente: <http://www.haproxy.org/>
Elaborado por: El autor

1.6.3.1 Características

Como características HAProxy incorpora muchos atributos que utilizan las arquitecturas de Sistemas Operativos, lo cual se refleja en el desempeño y rendimiento de este balanceador de carga.

De acuerdo a (López, 2016), estas son las características:

- Proxy TCP: puede aceptar conexiones TCP a través de un socket a la escucha, conectarse a otro servidor y permitir que la información fluya entre los dos servidores.
- Proxy inverso HTTP: se presenta como un servidor que recibe peticiones HTTP a través de conexiones aceptadas y pasa las solicitudes a los servidores.
- SSL *terminator/initiator/offloader*: para aligerar el trabajo de los nodos servidores.

- Normalizador TCP: sirve para proteger las pilas TCP de los ataques y optimizar los parámetros de conexión con los clientes.
- Normalizador HTTP: sirve para que solo las solicitudes válidas pasen a través del servidor evitando que se produzcan ataques basados en protocolo HTTP.
- Herramienta de fijación HTTP: sirve para modificar/fijar/añadir/quitar/reescribir la dirección URL o el encabezado de cualquier petición o respuesta.
- Encaminamiento basado en el contenido: permite considerar cualquier elemento de la solicitud para decidir a qué servidor se le remitirá la información o la conexión.
- Equilibrador de carga: equilibra la carga de las peticiones, tanto TCP como HTTP, entre los nodos servidores.
- Regulador del tráfico: protege a los nodos servidores contra sobrecargas, ajusta las prioridades de tráfico basado en el contenido e incluso transmite dicha información a las capas inferiores y a los componentes de la red exterior mediante el marcado de los paquetes.
- Protección contra ataques de denegación de servicio (DDoS) y abuso de servicio: mantiene una gran cantidad de información por dirección IP, URL, cookies, etc. para detectar posibles ataques y actuar en consecuencia.
- Punto de observación para solucionar problemas de red: debido a la precisión de la información almacenada en los registros, se utiliza a menudo para detectar y resolver problemas relacionados con la red.
- HTTP compression offloader: puede comprimir las respuestas que no se comprimieron por el servidor, reduciendo así el tiempo de carga de la web para los clientes con conectividad pobre o con alta latencia, como por ejemplo las redes móviles. (p. 15)

1.6.3.2 Haproxy en clúster de base de datos

Tomando en cuenta que Haproxy es un gran equilibrador carga del tipo de herramientas de código abierto, haciéndolo a través de TCP / HTTP, equilibrando la carga de trabajo a través de un grupo de servidores para incrementar y repotenciar el rendimiento del clúster, se destaca que su utilización puede ser aprovechada tanto en un clúster web como en un clúster de base de datos, o simultáneamente en ambos.

En una aplicación de front-end que cuenta con una base de datos (Mysql, Galera Cluster, Mariadb) en su backend fácilmente puede congestionar la base de datos, al tener muchas conexiones de ejecución concurrentes al mismo tiempo. En este punto es donde HAProxy brinda como solución la cola y el direccionamiento de conexiones hacia uno o varios servidores de base de datos evitando de esta manera que un único servidor se sature con tantas peticiones. Esta distribución de carga se la realiza a través del algoritmo de balanceo de carga configurado.

1.6.3.2.1 Ventajas de Haproxy en el equilibrio de carga

- Las peticiones realizadas a través de cualquiera de las aplicaciones, tienen acceso al clúster por medio de una única dirección IP (se configura una IP virtual).
- El balance de carga se realiza entre los servidores de base de datos activos y disponibles, los nodos que se encuentren caídos, no serán tomados en cuenta.
- En Mysql cuando se llega al número de conexiones máximas permitidas en la base de datos, Haproxy toma las nuevas conexiones y las ubica en cola de espera. De esta forma se protege a la base de datos de saturaciones limitando las solicitudes de conexión a la misma.
- Haproxy brinda la posibilidad de adicionar o quitar servidores de base de datos dentro del clúster sin necesidad de realizar cambios en la configuración.

Es importante mencionar que HAProxy ha sido diseñado para que posea métodos de comprobación de la salud, lo que permite que varios servicios balanceen la carga en una sola instancia.

1.6.3.2.2 Chequeo de salud

Las comprobaciones de salud de Haproxy hacia los servidores backend, es un punto de vital importancia, ya que es aquí donde se determina si un nodo web o BD, se encuentra disponible para tomar las solicitudes de acceso.

Existen algunos métodos de comprobación de salud entre los cuales tenemos:

- **Mysql-check** se trata en una verificación sencilla pero muy efectiva y útil, la cual permite determinar el estado de los servidores Mysql. Funciona enviando un paquete Mysql de identificación de usuario y otro paquete Mysql para el cierre de sesión.

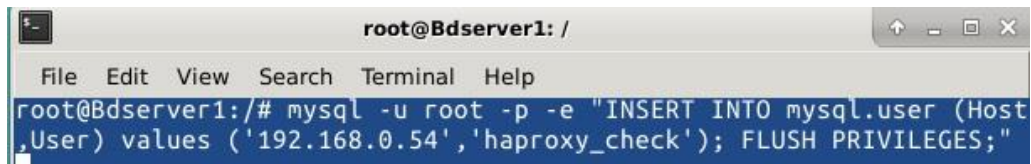
A screenshot of a terminal window titled "root@Bdserver1: /". The terminal shows a MySQL command being executed: "mysql -u root -p -e 'INSERT INTO mysql.user (Host,User) values ('192.168.0.54','haproxy_check'); FLUSH PRIVILEGES;'". The command is highlighted in blue. The terminal window has a menu bar with "File", "Edit", "View", "Search", "Terminal", and "Help".

Figura 7: Mysql-check
Fuente: El autor
Elaborado por: El autor

- **Tcp-check** se realiza chequeos de estado de salud por medio de la secuencia TCP
- **Httpcheck** se realiza chequeos de estado de salud para balancear carga con conexiones HTTP

1.6.3.2.3 Redundancia Haproxy

Dentro de un clúster de servidores en un entorno de desarrollo a gran escala, no puede existir un único punto de fallo, es decir, siempre debe haber por lo menos 1 servidor de respaldo que asuma, ante la caída del servidor principal.

Y en el clúster para que un usuario pueda conectarse a un servidor real, sea web o de base de datos, tiene que hacerlo primero por medio del servidor balanceador de carga Haproxy, de esta manera deben existir mínimo 2 servidores balanceadores de carga Haproxy. El primero estará configurado en modo activo y el segundo en modo pasivo, y este último se levantará ante la caída del primero.

Esta funcionalidad se la puede implementar en conjunto con otra herramienta de chequeo de salud, como Keppalived o Heartbeat.

1.6.3.2.4 Detección de fallos y conmutación por error

Dentro de las funciones de Haproxy se encuentra la detección de fallos, es decir si dentro del clúster se sufre la pérdida o baja de un servidor real, Haproxy detecta inmediatamente esta situación y redirige las peticiones hechas al servidor caído hacia un servidor sano y activo. Este proceso lo realizará en los tiempos definidos en la configuración inicial.

1.6.3.2.5 Descripción de la configuración principal de Haproxy

Listen: aquí en este grupo se definen las configuraciones principales de Haproxy, se coloca el nombre del clúster

Bind: en esta línea se indica el puerto que se utilizará, en el caso del clúster de base de datos, se utiliza el puerto 3306.

Mode: se define el modo http para web o el modo tcp para base de datos.

Balance: se define el algoritmo de balanceo de carga que se utilizará, como por ejemplo: roundrobin, leastconn, etc.

Server: son los servidores reales a los cuales se enviará las peticiones equilibrando la carga de las mismas.


```

root@HaproxyBd1: /
File Edit View Search Terminal Help
GNU nano 2.2.6 Archivo: etc/haproxy/haproxy.cfg

listen mysql-cluster
  bind 0.0.0.0:3306
  mode tcp
  option mysql-check user haproxy_check
  balance roundrobin
  server Bdserver1 192.168.0.60:3306 check
  server Bdserver2 192.168.0.61:3306 check

listen stats
  bind 192.168.0.21:8080
  mode http
  stats enable
  stats uri /
  stats realm Strictly\ Private

```

Figura 8: Configuración Haproxy
Fuente: El autor
Elaborado por: El autor

1.6.4 Tabla comparativa entre software para clúster

Tabla 5: Tabla comparativa de software para clúster

	Ventajas	Desventajas
Nginx	<ul style="list-style-type: none"> • Multifuncional, ya no solo brinda características de balanceador de carga. • Trabaja simultáneamente como servidor web y equilibrador de carga. 	<ul style="list-style-type: none"> • Métricas limitas (7) para estadísticas. • Poca compatibilidad para complementar con otras herramientas de monitoreo
Linux Virtual Server	<ul style="list-style-type: none"> • Facilidad de uso. • Funcionamiento multiprotocolo. • Ofrece buen alcance en las configuraciones. 	<ul style="list-style-type: none"> • Rendimiento afectado por la cantidad de servidores que se implementen. • Congestión en el flujo de red cuando la cantidad de servidores es grande
Piranha	<ul style="list-style-type: none"> • Implementación e instalación sencilla. 	<ul style="list-style-type: none"> • No siempre puede migrar todos los procesos.

	<ul style="list-style-type: none"> • Para el manejo y gestión, posee una interfaz web. • Soporte constante por parte de RedHat. 	<ul style="list-style-type: none"> • Presenta inconvenientes con memoria compartida.
Haproxy	<ul style="list-style-type: none"> • Muy completo en cuanto a métricas (61 métricas de estadísticas). • El reporte de estadísticas es muy completo y detallado, con información útil para el análisis. • Amplia gama de configuraciones. • La implementación con software de terceros es muy fácil. • Es muy potente y utilizado por varios entornos de alto rendimiento como: Twitter, GitHub, Imgur, Instagram. • A diferencia de otras herramientas buenas como Nginx, se centra únicamente en la función de balanceo de carga, por lo cual lo hace de manera eficaz y eficiente. • Es una herramienta multirpocolo. 	<ul style="list-style-type: none"> • No brinda otras funciones como Nginx.

Fuente: (López, 2016) & (Rodríguez Castellanos & Rodríguez Castiblanco, 2015)
 Elaborado por: El autor

De acuerdo a la tabla comparativa de software para clúster, se demuestra que Haproxy es una herramienta muy potente, que realiza exclusivamente balanceo de carga dentro del clúster,

debido a ello, todo su desempeño y capacidad está dirigido a la distribución de carga entre los nodos que conforman el clúster. Por tal motivo se elige a este software de código abierto para realizar el balanceo de carga en la arquitectura tecnológica que se está implementando en este proyecto.

1.6.5 Heartbeat

Traducido al español heartbeat significa latidos de corazón, por lo tanto su modo de funcionamiento consiste en el envío de latidos de corazón hacia el servidor primario para determinar, si éste, se encuentra activo o no. Dicho en otras palabras heartbeat envía ping hacia el servidor principal, y al haber ausencia de respuesta pasado un determinado tiempo, la herramienta comprueba que dicho servidor está inactivo, levantando automáticamente el servidor secundario o pasivo, para que asuma las funciones del principal.

Es importante complementar Heartbeat, con una herramienta de administración de recursos de clúster, que tiene la tarea de activar o inactivar servidores o servicios dentro del mismo.

1.7 Pruebas de carga y rendimiento de software

(Zapata & Cardona, 2011) Las pruebas de carga llevan al límite una aplicación Web, mediante la emulación de determinado número de usuarios con diversos periodos de conexión simultánea y distintas funcionalidades específicas en ejecución. Estas pruebas se relacionan directamente con las pruebas de rendimiento, debido a que éste se afecta a medida que la aplicación de software llega a su límite. (p. 143)

1.7.1 Pruebas de carga

(Cardona, 2009) establece:

Las pruebas de carga son aquellas en las cuales se lleva al límite una aplicación web mediante la emulación de determinado número de usuarios con diversos periodos de conexión simultánea y distintas funcionalidades específicas en ejecución. Las pruebas de carga se relacionan directamente con las pruebas de rendimiento debido a que este tiende a afectarse a medida que el aplicativo llega a su límite, es por esto que el monitoreo del software y los sistemas que apoyan los aplicativos web toma relevancia al realizar este tipo de pruebas. (p. 58)

1.7.2 Pruebas de rendimiento

(Deitel & Deitel, 2008) citados por (Cardona, 2009) señalan:

Su objetivo principal es desarrollar estrategias eficaces para la mejorar el rendimiento del sistema. En las pruebas de rendimiento se recopila y analiza información mediante un proceso de medición en el que se recogen datos para predecir cuando los niveles de carga agotarán los recursos del sistema. (p. 1025)

1.7.3 Pruebas de estrés

Evalúan el comportamiento de los sistemas cuando éstos son llevados más allá de sus límites operacionales (esto puede ser muy por encima de los requisitos no funcionales). Se evalúa las respuestas del sistema y de la aplicación a periodos de mayor volumen de actividad que superen las limitaciones del sistema. El objetivo principal de las pruebas de estrés es determinar si un sistema se bloquea o se recupera en dichas condiciones. Las pruebas de estrés deben ser diseñadas para llevar los límites de los recursos del sistema hasta el punto en que los puntos débiles de la aplicación queden expuestos. (Deitel & Deitel, 2008, p. 1025) citados por (Cardona, 2009)

1.7.4 Apache Jmeter

A consideración de (Nevedrov, 2006) establece:

Apache JMeter es una herramienta que puede usarse para probar aplicaciones que utilizan servidores HTTP o FTP. Se basa en Java y es altamente extensible a través de una API proporcionada. Una prueba típica JMeter implica la creación de un bucle y un grupo de hilos. El bucle simula peticiones secuenciales al servidor con un retardo predeterminado. Un grupo de hilos está diseñado para simular una carga concurrente. JMeter proporciona una interfaz de usuario. También expone una API que le permite ejecutar pruebas basadas en JMeter desde una aplicación Java. Para crear una prueba de carga en JMeter, construya un plan de prueba, que es esencialmente una secuencia de operaciones que JMeter ejecutará. El plan de prueba más simple normalmente incluye el siguientes elementos:

Thread group: estos elementos se utilizan para especificar el número de subprocesos en ejecución y un período de subida. Cada hilo simula un usuario y el período de ramp-up especifica el tiempo para crear todos los hilos. Por ejemplo con 5 hilos y 10 segundos de

tiempo de rampa, tardará 2 segundos entre cada hilo creación. El recuento de bucles define el tiempo de ejecución de un subproceso. El programador también le permite inicio y fin del tiempo de ejecución.

Samplers: estos elementos son solicitudes configurables para las solicitudes HTTP, FTP o LDAP del servidor.

Listeners: Estos elementos se utilizan para publicar los datos de solicitud de proceso. Por ejemplo, puede guardar datos archivo o ilustrar los resultados con un gráfico. En este momento, el gráfico de JMeter no proporciona opciones de configuración; Sin embargo es extensible y siempre es posible agregar una visualización extra o grupo de subprocesos.
(p. 1)

1.8 Trabajos relacionados

Es importante revisar y hacer referencia a los trabajos que tienen relación a este proyecto de tesis, realizado por otros autores y cuyo éxito ha sido comprobado y verificado.

Estos trabajos han sido desarrollados bajo la misma línea del presente proyecto, es decir que han implementado una arquitectura tecnológica a gran escala, con las características propias de estos sistemas, como son: alta disponibilidad y balanceo de carga.

A continuación, se mencionan los trabajos relacionados:

1. Hurtado, A. (2011). ***Propuesta de un servidor Web Apache 2 sobre un Cluster en Linux.*** Santa Clara: Universidad Central “Marta Abreu” de Las Villas.

En este trabajo, el autor desarrolla un proyecto dirigido a la implementación de un clúster en Linux, en el cual se maneje principalmente, balanceo de carga y alta disponibilidad, y que pueda garantizar la respuesta a todas las solicitudes enviadas por los usuarios a este sistema.

Base técnica

- ❖ **Hardware**, los computadores utilizados tienen estas características: Procesador Pentium 3.00Ghz, Memoria Ram 512MB, Disco Duro 40 GB. El clúster está formado por 4 computadores: 2 balanceadores de carga y 2 servidores web
- ❖ **Sistema Operativo**, Linux Debian
- ❖ **Software para Clúster**, Haproxy (balanceo de carga) y Keepalived (chequeo de estado de salud)
- ❖ **Algoritmo de Balanceo de Carga**, el método o algoritmo que utiliza el autor para realizar el equilibrio de carga es Round Robin.

Resultados de las pruebas

(Hurtado, 2011), referente a las pruebas señala:

En esta prueba se realizan un total de 15000 peticiones a través de Siege a la dirección Web <http://cluster.dps.vcl.sld.cu/index.php>.

En la Figura 26 se muestran los datos recopilados por Siege durante el proceso de simulación. Se observa que ocurrieron un total de 15000 transacciones en un tiempo aproximado de 38.45 segundos, se transfirió 1.36 MB de datos, las transacciones ocurrieron a una razón de 390.12 trans/seg, el Throughput fue de 0.04 MB/seg y que las 15000 transacciones se realizaron exitosamente. (p. 45)

En la Figura 27 se presentan las estadísticas recopiladas por la interfaz Web que presenta el software HaProxy, se debe observar que cada servidor Web (cluster1 y cluster2) recibe un total de 7500 sesiones, lo cual evidencia un balanceo de carga casi perfecto en el sistema.

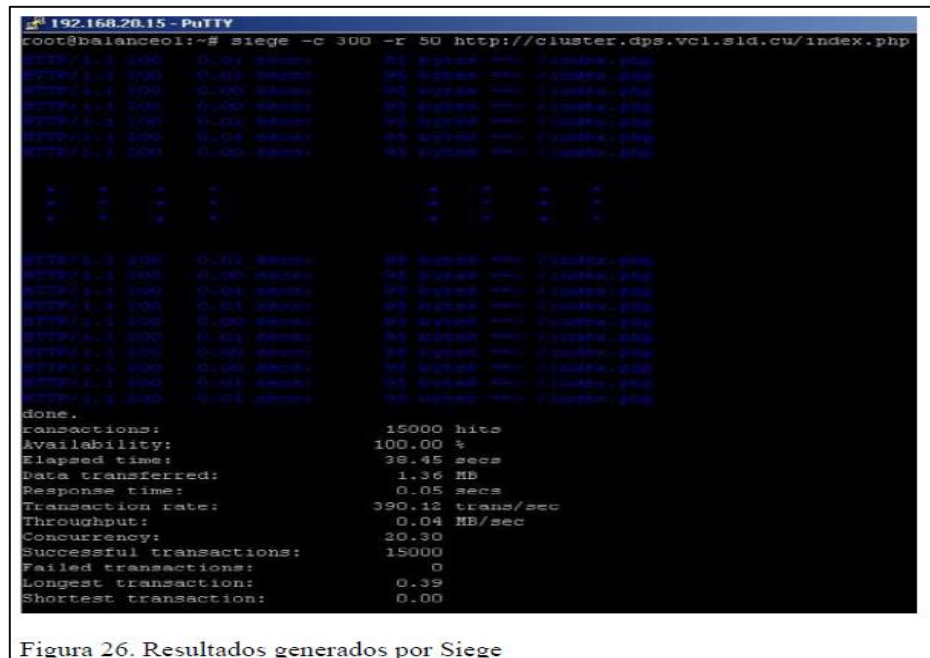


Figura 26. Resultados generados por Siege

Figura 9: Resultados Siege
Fuente: (Hurtado, 2011)
Elaborado por: (Hurtado, 2011)

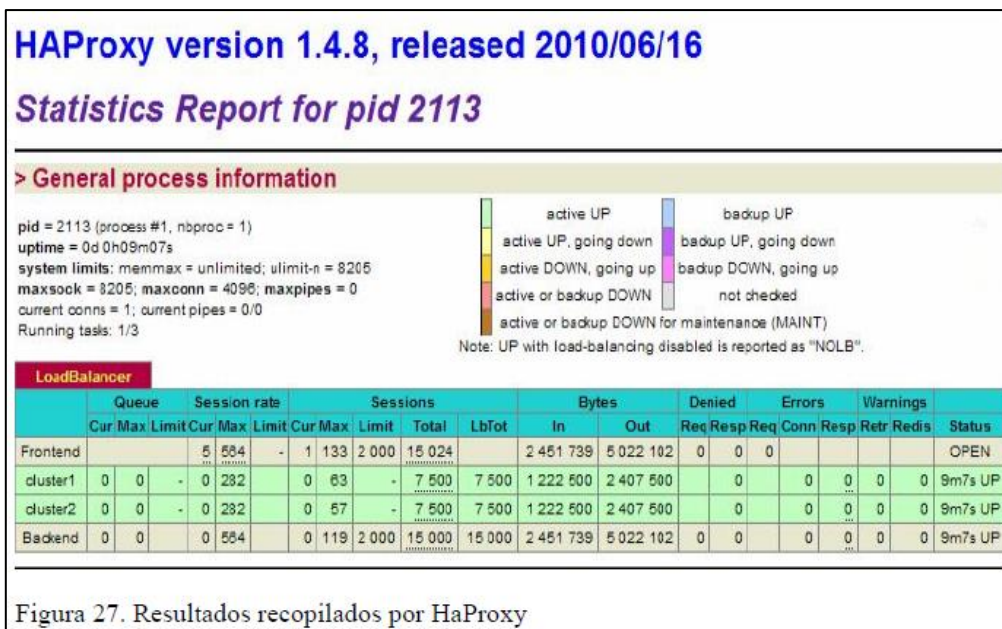


Figura 27. Resultados recopilados por HaProxy

Figura 10: Reporte de estadísticas Haproxy
Fuente: (Hurtado, 2011)
Elaborado por: (Hurtado, 2011)

2. López, M. (2016). *Instalación, configuración y evaluación de un servidor web de alta disponibilidad con equilibrado de carga*. Valencia: Universidad Politécnica de Valencia.

El autor de este trabajo, tiene como objetivo conseguir la instalación, configuración y verificación de funcionamiento, de una arquitectura que brinde alta disponibilidad y balanceo de carga.

Base técnica

❖ Recursos utilizados

En cuanto a los recursos utilizados (López, 2016) indica:

El clúster se ha desplegado utilizando un entorno de virtualización. Si bien un clúster real estaría compuesto de máquinas físicas, un entorno virtualizado ofrece una gran flexibilidad para la realización de pruebas de una forma muy económica. En concreto, se utilizará una infraestructura virtual basada en hipervisores de Oracle VM VirtualBox (versión 5.0.20).

El sistema operativo usado para la instalación de los nodos que constituyen el clúster de este trabajo será de la familia Ubuntu, en concreto, se instalará *Ubuntu Server 14.04.4 LTS (Long term Support)*, versión estable y gratuita.

Las máquinas virtuales se lanzarán desde un portátil con el programa VirtualBox. El host anfitrión es un portátil que tiene las siguientes características:

- **SO:** Windows 10 Home (64 bits)
- **RAM:** 8 GB a 1333 MHz
- **Disco duro:** 500 GB
- **Microprocesador:** Intel i3-3110M a 2.40 GHz

Estas máquinas se ejecutan en el portátil debido a que no se dispone del hardware para el montaje de un clúster real. Por ese motivo, debido a la limitación de recursos, el clúster implementado incluirá un número reducido de nodos servidores. Este clúster virtual se podría luego migrar a un clúster

real con un mayor número de nodos servidores sin demasiado esfuerzo.
(p. 11)

(López, 2016) establece:

El sistema estará formado por 8 nodos distribuidos de la siguiente manera:

- **master y standby:** nodos maestros. Están duplicados para conseguir alta disponibilidad, componiendo un servicio activo/pasivo. Estos nodos forman el *front-end* del sistema. Hacia el exterior (Internet), ofrecen el servicio web en el puerto 80 y en el 443 de una dirección IP virtual, asociada al nodo que en cada momento esté activo. Además de las funciones propias del nodo maestro de un clúster, también se ocupará de distribuir la carga, repartiendo las peticiones HTTP y HTTPS recibidas entre los servidores reales del *back-end*.
 - **server1, server2 y server3:** servidores web reales. Estos nodos forman el *back-end* del sistema. Tienen instalado un servidor apache que atiende las peticiones que les deriva el nodo maestro activo.
 - **NFS:** servidor de almacenamiento. Es un servidor NFS. Las páginas web proporcionadas por los servidores reales están almacenadas aquí.
 - **bdserver1 y bdserver2:** nodos de bases de datos replicados en forma de maestro/esclavo que ofrecen acceso a los servidores web.
- (p. 9)

❖ **Sistema Operativo**, Linux Ubuntu 14.04

❖ **Software para Clúster**, Haproxy (balanceo de carga) y Keepalived (chequeo de estado de salud)

- ❖ **Algoritmo de Balanceo de Carga**, el método o algoritmo que utiliza el autor para realizar el equilibrio de carga es Round Robin.

Resultados de las pruebas

(López, 2016), referente a las pruebas señala:

Vamos a proceder con las pruebas. En la primera prueba comprobaremos si se garantiza la alta disponibilidad, desconectando la interfaz externa del nodo maestro (interfaz “eth0”). La prueba consiste en lanzar un par de peticiones, desconectar la interfaz de red y lanzar una nueva petición.

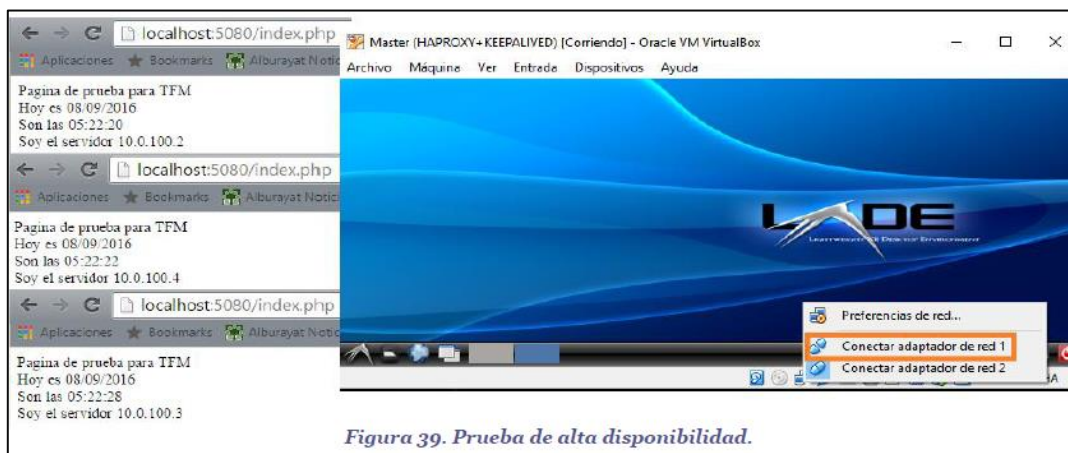


Figura 39. Prueba de alta disponibilidad.

Figura 11: Prueba alta disponibilidad

Fuente: (López, 2016)

Elaborado por: (López, 2016)

(López, 2016), indica, Como vemos en la figura anterior el servidor responde satisfactoriamente las peticiones lanzadas. Aquí el nodo standby ha adquirido la labor de nodo maestro. Sin embargo, debe hacerse constar que el cambio no es instantáneo, la carga de la web falla un par de veces.

En la segunda prueba simularemos la caída de uno de los nodos servidores. Por ejemplo, el nodo server1 (cluster2) falla. (p. 44)

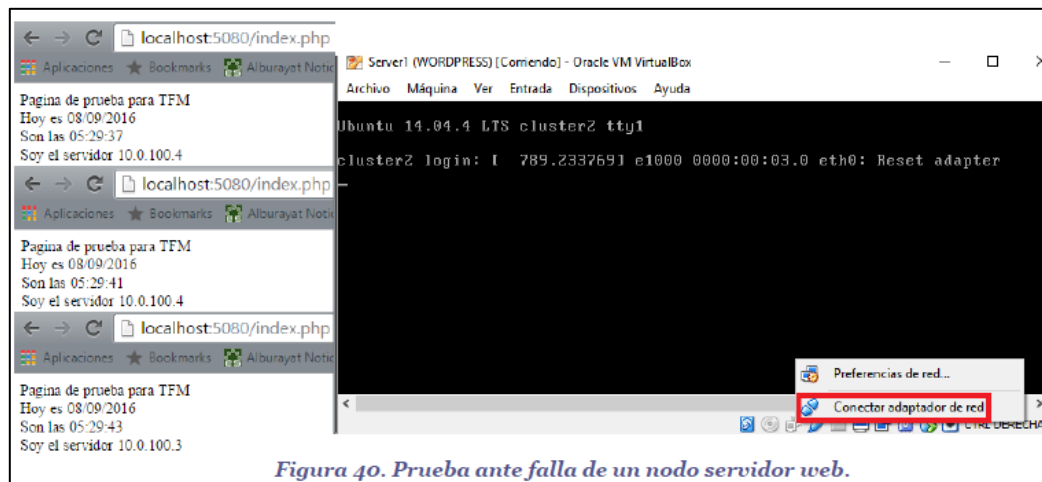


Figura 40. Prueba ante falla de un nodo servidor web.

Figura 12: Prueba de fallo nodo servidor web

Fuente: (López, 2016)

Elaborado por: (López, 2016)

“Vemos en la figura anterior que la web se sigue sirviendo por parte de los nodos supervivientes: server2 y server3. Si cayera otro de los nodos servidores, las peticiones serían respondidas por el nodo que quedase”. (López, 2016, p. 45)

CAPÍTULO II. SOLUCION A LA PROBLEMÁTICA

2.1 Introducción

El uso de grandes cantidades de información en una aplicación web, puede generar problemas de rendimiento y funcionamiento para los usuarios finales. En ese sentido se pretende buscar la optimización de estos sistemas, mediante conceptos como balanceo de carga, a nivel de servidores de aplicación y de base de datos, en vista que los modelos tradicionales para la gestión de la información a través de sistemas informáticos, no cubren todas las necesidades y requerimientos de procesamiento, gestión, disponibilidad, equilibrio de carga, ya que todos los procesos son absorbidos por un solo servidor.

De esta manera se implementará una arquitectura tecnológica en un entorno de desarrollo a gran escala la cual se fusionará posteriormente con una aplicación web desarrollada con el framework Django.

En este sentido la arquitectura tecnológica a gran escala que se implementará, deberá brindar los servicios de: balanceo de carga, alta disponibilidad, escalabilidad, persistencia de sesiones y replicación de los datos gestionados en la base de datos.

En tal virtud se menciona que la solución que se propone, se la implementará a través herramientas de código abierto, diseñadas para la gestión de granjas de servidores en entornos de desarrollo a gran escala, tomando como base los estándares y paradigmas propios de este tipo de estructuras.

En la figura 1, se puede apreciar gráficamente un caso de un sistema informático que obedece a los paradigmas tradicionales, en el cual se instala todo en un mismo servidor, es decir app, servidor web, base de datos, y cuando este único punto de fallo cae, el sistema o aplicación web queda sin funcionar.

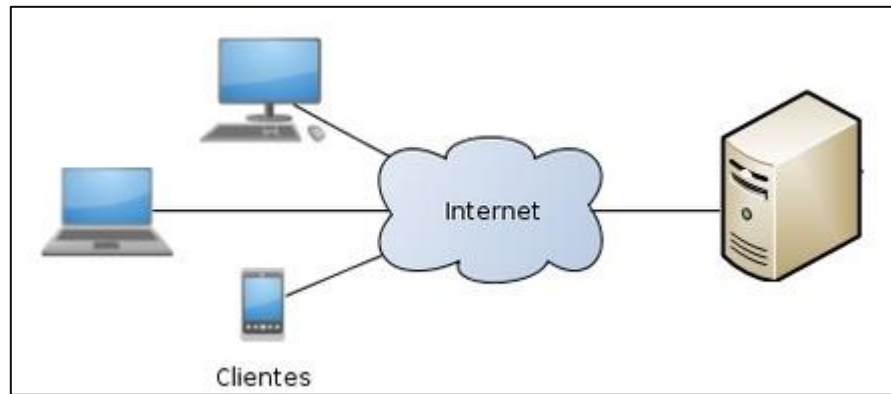


Figura 13: modelo convencional cliente-servidor
 Fuente: El autor
 Elaborado por: El autor

Ahora bien, en la figura 14, se aprecia que este modelo puede mejorar de cierto modo, dividiendo el trabajo en 2 equipos, un servidor para la aplicación web y otro para el alojamiento de la Base de Datos. Así se ha dividido hasta cierto punto, la carga de trabajo. Pero este modelo sigue teniendo un único punto de fallo sin servidores de redundancia o respaldo, es decir si el servidor web falla, cae todo el sistema, o si el servidor de Base de Datos falla, de igual manera cae todo el sistema.

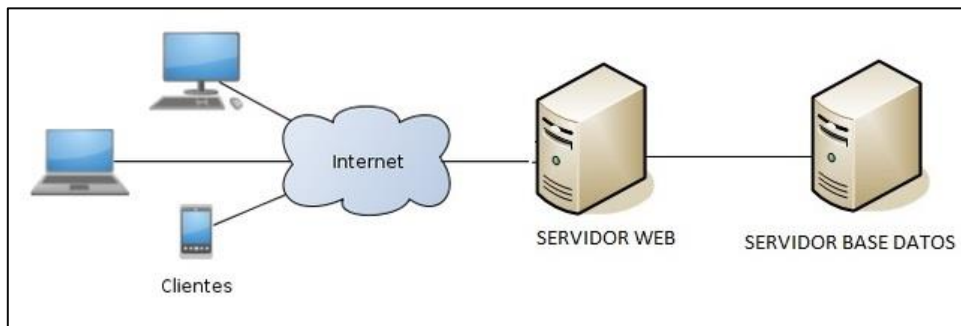


Figura 14: Modelo convencional 2 cliente-servidor
 Fuente: El autor
 Elaborado por: El autor

2.2 Arquitectura Tecnológica a Gran Escala propuesta

En vista de la problemática en cuestión se origina y se toma como solución a las aplicaciones web en un entorno de desarrollo para una arquitectura a gran escala.

En este sentido el presente trabajo de fin de titulación está dirigido al *Análisis e implementación de aplicaciones mediante el framework Django en un entorno de desarrollo para una arquitectura a gran escala.*

De esta manera se propone como solución a la problemática, el diseño e implementación de una arquitectura tecnológica a gran escala, basándose en los principios que rigen esta arquitectura, principalmente los atributos fuertes de este modelo como son:

2.2.1 Alta disponibilidad

En el alojamiento web, se cuenta con 2 servidores principales (uno de ellos es el Balanceador de carga y otro es el servidor web), y 2 servidores secundarios o de respaldo cumpliendo exactamente las mismas funciones que los primeros. En caso de la caída de uno de los servidores principales, la aplicación web no se verá afectada ya que seguirá funcionando con los servidores secundarios, demostrando así alta disponibilidad, el sistema nunca deja de funcionar.

El mismo concepto se manejará en la parte de Base de Datos, se contará con 2 servidores principales (Servidor Load Balancer y Servidor de Base de Datos), y 2 servidores secundarios cumpliendo las mismas funciones.

2.2.2 Balanceo de carga

En la parte web se tiene 2 servidores para realizar balanceo de carga, un servidor principal y otro de respaldo que responderá y se levantará automáticamente y de inmediato ante la caída del servidor principal, es decir, la arquitectura cuenta con 1 servidor activo y 1 servidor pasivo. En cualquier caso, estos balanceadores de carga tienen la función de balancear o distribuir la carga equitativamente, entre los 2 servidores reales (servidores web), evitando así la saturación de un solo servidor.

El mismo modelo se maneja en la parte de Base de Datos, teniendo 2 servidores Balanceadores de carga, cumpliendo funciones de distribución equitativa de carga entre los servidores reales de Base de Datos.

2.2.2.1 Algoritmo de Balanceo de carga

El método o algoritmo que se utilizará para realizar el equilibrio de carga es Round Robin con pesos iguales para cada servidor.

2.2.3 Detalle de Arquitectura Tecnológica a utilizar

El objetivo que busca conseguir esta arquitectura, es brindar alta disponibilidad, balanceo de carga y redundancia tanto en los Servidores Web como en los Servidores Base de Datos, utilizando mínimo 2 servidores en cada punto de la arquitectura, de esta manera si ocurre un fallo en el servidor primario, automáticamente se levanta el servidor secundario, reflejando que no existe un punto único de fallo que haga caer el sistema.

La implementación de la infraestructura tecnológica se realiza a través de servidores virtuales, con la utilización de la herramienta Virtual Box, misma que se encuentra instalada en 1 computador portátil con las siguientes características:

Tabla 6: Computador donde se alojarán los servidores virtuales

EQUIPO	TIPO	MARCA	PROCESADOR	MEMORIA RAM	SISTEMA OPERATIVO	PLATAFORMA
Computador	Laptop	Toshiba	Core i7	16 GB	Windows 10	64 bits

Fuente: El autor

Elaborado por: El autor

Una vez instalado Virtual Box en el computador portátil destinado para diseñar la implementación de la arquitectura tecnológica, se procede a instalar cada uno de los servidores virtuales, sumando un total de 8 servidores dentro del Virtual Box.

Es importante mencionar que, para efectos de pruebas en la implementación, se utilizará el computador portátil antes mencionado, mismo que es un equipo de uso doméstico, para instalar en él, todos los servidores virtuales, de esta manera, los 8 servidores en realidad funcionarán a través del mismo procesador lo tanto, no se tiene la capacidad de ni el rendimiento de 8 servidores independientes, no obstante igual se reflejará y quedará claramente demostrado, las características de la arquitectura a gran escala, como el balanceo de carga y alta disponibilidad, mejorando el rendimiento del clúster, la aplicación web y la base de datos que se alojará en este. En la ilustración 4, se aprecia el clúster ya instalado dentro de la herramienta Virtual Box.

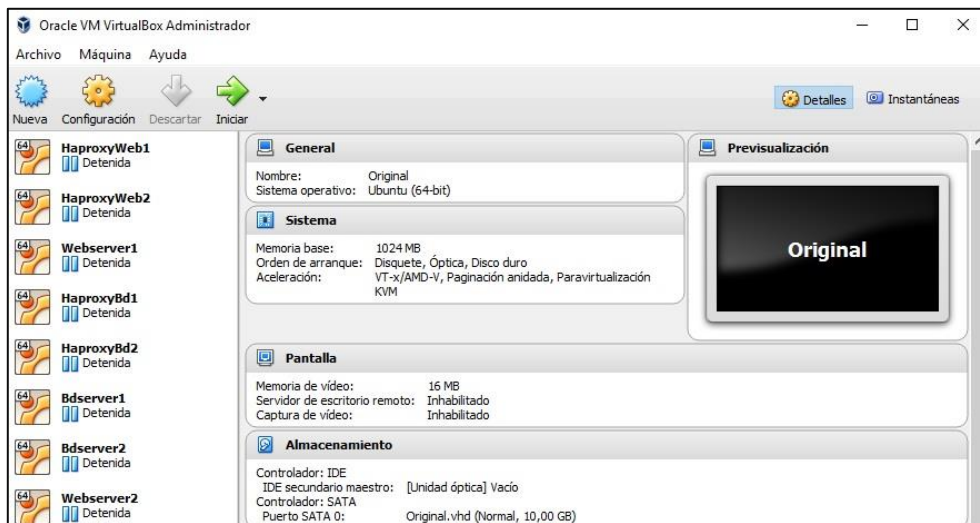


Figura 15: Granja de servidores en VirtualBox

Fuente: El autor

Elaborado por: El autor

2.2.4 Software empleado en la Arquitectura

En la solución a la problemática, se propone, implementar la arquitectura a gran escala por medio de software, con la utilización de herramientas Open Source, a través de las cuales se implementará los modelos de la arquitectura a gran escala, Alta Disponibilidad y Balanceo de Carga.

De esta manera para cumplir con estas funciones de Distribución y equilibrio de carga, y los atributos de alta disponibilidad se empleará básicamente 2 herramientas que son: Haproxy y Heartbeat.

Haproxy se constituye como una herramienta Open Source multiprotocolo que sirve para implementar balanceo de carga dentro de un clúster de servidores. En nuestra solución se lo utilizará tanto en el clúster de alojamiento web a través del protocolo HTTP, como en el clúster de servidores de Base de Datos a través del protocolo TCP.

Heartbeat es un servicio utilizado en los modelos de clúster de servidores, el cual tiene la función de constatar si un nodo se encuentra activo o inactivo dentro de ese clúster, una vez constatado el estado de dicho nodo se envía una señal al clúster permitiendo que el nodo o servidor de respaldo se active de manera automática.

Ambas herramientas se instalarán en los 2 servidores Load Balancer que se encuentran en el clúster web y, en los 2 servidores Load Balancer que se encuentran en el clúster de Base de Datos.

2.3 Conformación de clúster para la arquitectura tecnológica

La arquitectura tecnológica para la implementación de la app, está conformada en total por 8 servidores, divididos en 2 clúster repartidos de la siguiente manera:

Tabla 7: Clústeres de la arquitectura tecnológica

NOMBRE	CLÚSTER WEB		CLÚSTER BASE DATOS	
	BALANCEADOR CARGA WEB	SERVIDOR REAL WEB	BALANCEADOR CARGA BASE DATOS	SERVIDOR REAL BASE DATOS
HaproxyWeb1 (activo)	1			
HaproxyWeb2 (pasivo)	1			
Webserver1 (activo)		1		
Webserver2 (activo)		1		
HaproxyBd1 (activo)			1	
HaproxyBd2 (pasivo)			1	
Bdserver1 (replicando)				1
Bdserver2 (replicando)				1

Fuente: El autor

Elaborado por: El autor

La arquitectura tecnológica propuesta se puede apreciar gráficamente en la figura 19.

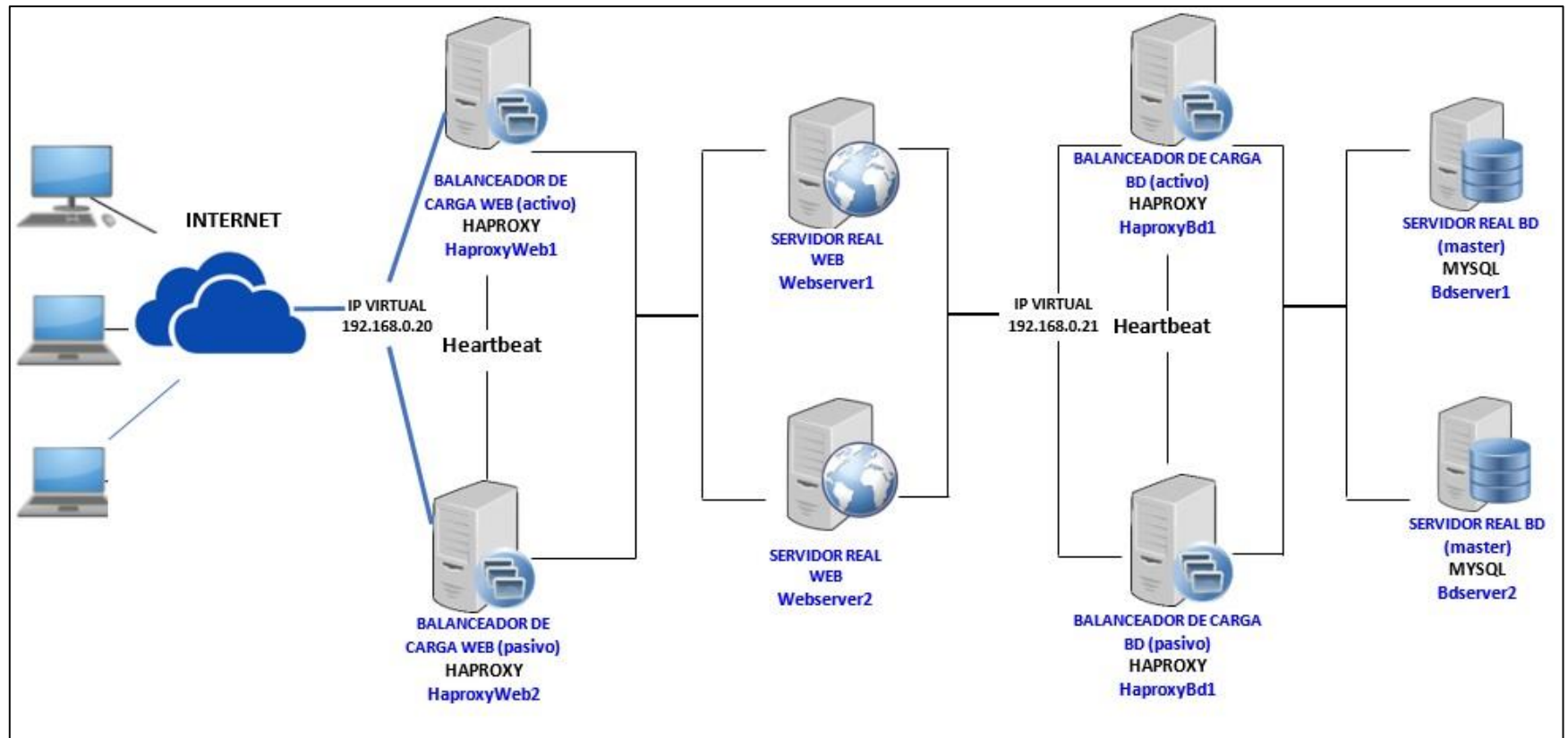


Figura 16: Diagrama arquitectura tecnológica a gran escala
 Fuente: El autor
 Elaborado por: El autor

Ahora bien, cuál es el modo de ejecución y funcionamiento de la arquitectura tecnológica; se indica a continuación:

1. El ingreso a la aplicación Web se realizará a través de los Servidores Frontend Balanceadores de Carga (HaproxyWeb1 y HaproxyWeb2).
2. Siendo así, la dirección IP de acceso a la Aplicación Web es la IP Virtual: 192.168.0.20
3. Esta IP virtual redirigirá la petición hacia los Servidores Reales Web, en este caso, la Ip 192.168.0.40 que es el Webserver1, o en su defecto a la Ip 192.168.0.41 que es el Webserver2 (donde se encuentra instalado, Apache, Python y Django).
4. Una vez hecha la petición de acceso al aplicativo web, sea al Webserver1 o Webserver2, si el usuario ejecuta su Login, crea, edita o elimina algún registro, quiere decir que se ha llamado a la Base de Datos, entonces para ejecutar el login, crear, editar, eliminar o sacar algún reporte, el Webserver1 o Webserver2 a través de Django tendrá que redirigir esa petición al Balanceador de Carga de Base de Datos (Esta redirección o conexión, Django la realiza a través del archivo de configuración “settings.py”).
5. El Balanceador de carga Base de Datos puede ser el HaproxyBd1 o HaproxyBd2 a través de la IP virtual 192.168.0.21. Y éste redirigirá la petición a los Servidor Reales Backend de Base de Datos (Bdserver1 o Bdserver2), que es donde está instalado el Mysql con la Base de Datos de la Aplicación Web.

De acuerdo a esta explicación, se evidencia que:

- **El código fuente** de la aplicación web, se encuentra en los servidores web (Webserver1 y Wbeserver2), que es donde está instalado el Framework Django.
- Django entre sus puntos fuertes, cuenta con el archivo de configuración “**settings.py**”, por ende, dicho archivo se encuentra en (Webserver1 y Webserver2) apuntando a los balanceadores de carga de base de datos.
- A través del “**settings.py**”, la app web recibe las solicitudes de acceso a la base de datos, por medio de los servidores de equilibrio de carga, mismos que trabajan con una dirección ip Virtual (Ip Virtual 192.168.0.21), para distribuir equitativamente la carga hacia los servidores Backend. De esta manera la Ip Virtual se la define en el archivo “settings.py”, así como la plataforma de Base de Datos y el Wsgi.

Qué se consigue con esto, una gestión muy ágil, práctica y eficiente, para la gestión de la base de datos, con la utilización del clúster de base de datos a través de la redirección de las peticiones con el archivo “settings.py”.

- **La Base de Datos** de la aplicación web se encuentra alojada en los Servidores Reales de Base de Datos Bdserver1 y Bdserver2, los cuales se encuentran en modo de replicación constante.

2.3.1 Clúster web server

Este clúster se encuentra conformado por 4 equipos, de los cuales, 2 son Balanceadores de Carga y 2 son Servidores Reales, teniendo lo siguiente:

Tabla 8: Clúster WebServer

NOMBRE	TIPO	IP ADDRESS	MÁSCARA	IP VIRTUAL	SISTEMA OPERATIVO	SOFTWARE INSTALADO
HaproxyWeb1	Load Balancer (Activo)	192.168.0.30	255.255.255.0	192.168.0.20	Linux Ubuntu Server 14.0.4	Haproxy Heartbeat
HaproxyWeb1	Load Balancer (Pasivo)	192.168.0.31	255.255.255.0	192.168.0.20	Linux Ubuntu Server 14.0.4	Haproxy Heartbeat
WebServer1	Servidor Real	192.168.0.40	255.255.255.0	-	Linux Ubuntu Server 14.0.4	Python 2.7 Django 1.10 Apache
WebServer2	Servidor Real	192.168.0.41	255.255.255.0	-	Linux Ubuntu Server 14.0.4	Python 2.7 Django 1.10 Apache

Fuente: El autor

Elaborado por: El autor

- Los balanceadores de carga, HaproxyWeb1 se encuentra en modo activo y HaproxyWeb2 se encuentra en modo pasivo. Cuando se presente el fallo de HaproxyWeb1, HaproxyWeb2 automáticamente asume modo activo, y el sistema sigue disponible.
- Los servidores web, Webserver1 y Webserver2, ambos en modo activo, se reparten equitativamente las cargas de trabajo, y en el caso de fallo de cualquiera de los 2, el nodo que queda en funcionamiento, absorbe toda la carga de trabajo.

2.3.2 Clúster base de datos

Este clúster se encuentra conformado por 4 equipos, de los cuales, 2 son Balanceadores de Carga y 2 son Servidores Reales de Base de Datos, teniendo lo siguiente:

Tabla 9: Clúster base de datos

NOMBRE	TIPO	IP ADDRESS	MÁSCARA	IP VIRTUAL	SISTEMA OPERATIVO	SOFTWARE INSTALADO
HaproxyBd1	Load Balancer (Activo)	192.168.0.50	255.255.255.0	192.168.0.21	Linux Ubuntu Server 14.0.4	Haproxy Heartbeat Mysql Client
HaproxyBd2	Load Balancer (Pasivo)	192.168.0.51	255.255.255.0	192.168.0.21	Linux Ubuntu Server 14.0.4	Haproxy Heartbeat Mysql Client
Bdserver1	Servidor Real	192.168.0.60	255.255.255.0	-	Linux Ubuntu Server 14.0.4	Mysql Server
Bdserver2	Servidor Real	192.168.0.61	255.255.255.0	-	Linux Ubuntu Server 14.0.4	Mysql Server

Fuente: El autor

Elaborado por: El autor

- Los balanceadores de carga, HaproxyBd1 se encuentra en modo activo y HaproxyBd2 se encuentra en modo pasivo. Cuando se presente el fallo de HaproxyBd1, HaproxyBd2 automáticamente asume modo activo, y el sistema sigue disponible.
- Los servidores de base de datos, Bdserver1 y Bdserver2, ambos en modo activo, se reparten equitativamente las cargas de trabajo, y en el caso de fallo de cualquiera de los 2, el nodo que queda en funcionamiento, absorbe toda la carga de trabajo.

2.4 Instalación y configuración de clúster webserver

2.4.1 Configuración servidores de balanceo de carga (Haproxyweb1 y Haproxyweb2)

Primero se realiza la configuración IP en los servidores, entonces se edita el archivo *etc/network/interfaces*, utilizando el comando para edición de archivos *nano*.

```
nano etc/network/interfaces
```

A continuación, se configura Ip Address e Ip Virtual en HaproxyWeb1, de esta manera:

```
auto lo
iface lo inet loopback

# IP ADDRESS
auto eth0
iface eth0 inet static
address 192.168.0.30
netmask 255.255.255.0
network 192.168.0.0
broadcast 192.168.0.255
gateway 192.168.0.1

# IP VIRTUAL
auto eth0:1
iface eth0:1 inet static
address 192.168.0.20
netmask 255.255.255.0
```

Luego se configura Ip Address e Ip Virtual en HaproxyWeb2, de esta manera

```
auto lo
iface lo inet loopback

# IP ADDRESS
auto eth0
iface eth0 inet static
address 192.168.0.31
netmask 255.255.255.0
network 192.168.0.0
broadcast 192.168.0.255
gateway 192.168.0.1

# IP VIRTUAL
auto eth0:1
iface eth0:1 inet static
address 192.168.0.20
netmask 255.255.255.0
```

Ahora se ejecuta el siguiente comando, para que los 2 balanceador de carga se puedan iniciar sin problemas

```
Echo "net.ipv4.ip_nonlocal_bind = 1" >> /etc/sysctl.conf
```

2.4.1.1 Instalación Haproxy y Heartbeat

En los servidores de Balanceo de Carga de la app (HaproxyWeb1 y HaproxyWeb2), se debe instalar y configurar las herramientas HAPROXY (permite realizar balanceo de carga) y HEARTBEAT (ofrece comunicación y monitoreo de los servidores load balancer que integran el clúster)

Con el comando apt-get install se procede a la instalación de HAPROXY y HEARTBEAT

```
apt-get install haproxy
apt-get install heartbeat
```

2.4.1.2 Configuración Haproxy

En este caso HAPROXY, se configura el balanceo de carga Roundrobin, además se establecen los nodos que cumplen la función de Servidores Reales (Webserver) y sean conocidos por el software. Esta configuración se la realiza en */etc/haproxy/haproxy.cfg*

Configuración global

```
global
    log /dev/log      local0
    log /dev/log      local1 notice
    chroot /var/lib/haproxy
    user haproxy
    group haproxy
    daemon
```

Configuración modo http

```
defaults
    log      global
    mode     http
    option   httplog
    option   dontlognull
    contimeout 5000
    clitimeout 50000
    srvtimeout 50000
```


Configuración Ip Virtual (Frontend) de ingreso al cluster web, y definición de servidores reales (webserver) a donde se distribuirán las peticiones que se hacen en el Frontend.

```
listen      web-cluster      192.168.0.20:80
            mode http
            stats enable
            stats auth admin:password # usuario y contraseña
            balance rounrobin
            option httpclose
            option forwardfor
            cookie JSESSIONID prefix
            server web1 192.168.0.40:80 cookie A check
            server web2 192.168.0.41:80 cookie B check
```

Luego se configura la herramienta HAPROXY para que se ejecute con el script de inicio

```
nano /etc/default/haproxy
```

Se encuentra

```
ENABLED = 0
```

Se cambia a

```
ENABLED = 1
```

Ahora se ejecuta el script sin ningún parámetro para comprobar si el cambio se ha realizado de manera correcta.

```
service haproxy
usage: /etc/init.d/haproxy {start|stop|reload|restart|status}
```

Por último, se inicia los servicios de Haproxy.

```
/etc/init.d/haproxy start
```

2.4.1.3 Configuración de Heartbeat

Luego se configura Heartbeat a través del siguiente script *etc/ha.d/ha.cf*, en el cual se indica la Ip del otro servidor balanceador de carga y se colocan los nombres de todos los servidores Frontend que hagan la función de balanceador de carga.

HaproxyWeb1

```
keeplive      2
deadtime     10
udpport      694
bcast        eth0
mcast        eth0 225.0.0.1 694 1 0
ucast        eth0 192.168.0.31
logfacility    local0
auto_failback on
node         HaproxyWeb1
node         HaproxyWeb2
```

HaproxyWeb2

```
keeplive      2
deadtime     10
udpport      694
bcast        eth0
mcast        eth0 225.0.0.1 694 1 0
ucast        eth0 192.168.0.30
logfacility    local0
auto_failback on
node         HaproxyWeb1
node         HaproxyWeb2
```

Se configura el siguiente archivo *etc/ha.d/authkeys* para permitir a los nodos miembros identificar o autenticarse dentro del cluster

```
auth 3
1 crc
2 sha1 yourpasswoedhere
3 md5 yourpasswordhere
```

Luego se cambia los permisos

```
chmod 600 etc/ha.d/authkeys
```

Finalmente se indica a HEARTBEAT la Ip Virtual del Servidor, configurando el archivo *etc/ha.d/haresources*

```
Haproxy1 \  
IPaddr :: 192.168.0.20/24/eth0
```

2.4.2 Configuración servidores web reales (Webserver1 y Webserver2)

A continuación, se configura Ip Address, de esta manera:

```
root@Webserver1/# nano etc/network/interfaces
```

```
auto lo  
iface lo inet loopback  
  
# IP ADDRESS  
auto eth0  
iface eth0 inet static  
address 192.168.0.40  
netmask 255.255.255.0  
network 192.168.0.0  
broadcast 192.168.0.255  
gateway 192.168.0.1
```

```
root@Webserver2/# nano etc/network/interfaces
```

```
auto lo  
iface lo inet loopback  
  
# IP ADDRESS  
auto eth0  
iface eth0 inet static  
address 192.168.0.41  
netmask 255.255.255.0  
network 192.168.0.0  
broadcast 192.168.0.255
```

```
gateway 192.168.0.1
```

2.4.2.1 Instalación Apache

La instalación de apache se la realiza en los Servidores reales Webserver1 y Webserver2.

```
root@Webserver1/# apt-get install apache2
```

```
root@Webserver2/# apt-get install apache2
```

2.5 Instalación y configuración clúster de base de datos

La instalación y configuración del Clúster de Base de Datos se divide en 2 partes: Instalación y configuración de servidores reales Mysql, realizando redundancia y, por otro lado la instalación y configuración de los servidores balanceadores de carga que se integrarán con los servidores Mysql.

2.5.1 Configuración servidores de base de datos mysql – replicación (Bdserver1 y Bdserver2)

El modelo de replicación Mysql que se utilizará en los servidores de Base de Datos, será Master-Master, que a diferencia de la replicación típica Maestro-Esclavo, permite que ambos servidores tengan capacidades de lectura y escritura,

Con la configuración Master-Master se incrementa la eficiencia y rendimiento al acceder a la Base de Datos desde cualquier Servidor BD, realizando gestiones RW y replicando la transacción hacia el otro servidor, lo que ayuda a proporcionar un verdadero balanceo de carga.

A continuación, se configura Ip Address, de esta manera:

```
root@Bdserver1/# nano etc/network/interfaces
```

```
auto lo
iface lo inet loopback

# IP ADDRESS
auto eth0
iface eth0 inet static
address 192.168.0.60
netmask 255.255.255.0
network 192.168.0.0
broadcast 192.168.0.255
gateway 192.168.0.1
```

```
root@Bdserver2/# nano etc/network/interfaces
```

```
auto lo
iface lo inet loopback

# IP ADDRESS
auto eth0
iface eth0 inet static
address 192.168.0.61
netmask 255.255.255.0
network 192.168.0.0
broadcast 192.168.0.255
gateway 192.168.0.1
```

2.5.1.1 Instalar y configurar Mysql en Bdserver1

Se procede a instalar mysql-server y mysql-client, a través del siguiente comando:

```
root@Bdserver1/# apt-get install mysql-server mysql-client
```

Ahora se edita el archivo ubicado en etc/mysql/my.cnf, para cambiar ciertos ajustes que son necesarios para realizar la replicación entre los bdserver, dicho archivo lo encontraremos así:

```
#server-id = 1
#log_bin = /var/log/mysql/mysql-bin.log
#binlog_do_db = include_database_name
bind-address = 127.0.0.1
```

Se debe modificar la configuración y tener lo siguiente:

Línea 1 es la identificación del servidor, la línea 2 muestra el archivo donde se registra cambios en la base de datos mysql, la línea 3 se ubica el nombre de la base de datos que se va a replicar entre los bdserver. Y la línea 4 le indica a Mysql que permita conexiones de internet al estar comentado la escucha por el localhost.

```
server-id = 1
log_bin = /var/log/mysql/mysql-bin.log
binlog_do_db = hospital
# bind-address = 127.0.0.1
```

Luego se reinicia Mysql de la siguiente manera:

```
root@Bdserver1/# sudo service mysql restart
```

Ahora hay que realizar unas configuraciones a través del Shell, ingresando con el usuario root mysql y la respectiva contraseña.

```
root@Bdserver1/# mysql -u root -p
```

Una vez hecho esto, se debe crear un usuario que servirá para realizar la réplica entre los servidores, seguido de la contraseña.

```
root@Bdserver1/# create user 'replicador'@'%' identified by 'password';
```

Luego conceder permisos al usuario creado para realizar las réplicas.

```
root@Bdserver1/# grant replication slave on *.* to 'replicador'@'%';
```

Posterior a esto, hay que obtener datos sobre el estado actual del servidor bdserver1.

```
root@Bdserver1/# show master status;
```

```

+-----+-----+-----+-----+
| File           | Position | Binlog_Do_DB | Binlog_Ignore_DB |
+-----+-----+-----+-----+
| mysql-bin.000063 |      1553 | hospital      |                    |
+-----+-----+-----+-----+
1 row in set (0.00 sec)

```

Entonces nos presentará el archivo y su ubicación

2.5.1.2 Instalar y configurar Mysql en Bdserver2

Al igual que en Bdserver1, se instala mysql-server y mysql-client

```
root@Bdserver2/# apt-get install mysql-server mysql-client
```

Luego de haber instalado ambos paquetes, nos dirigimos a configurar el archivo ubicado en etc/mysql/my.cnf.

```
root@Bdserver2/# nano etc/mysql/my.cnf
```

Ahora se debe modificar la configuración de manera similar que se hizo en Bdserver1, teniendo como valores predeterminados lo siguiente:

```
#server-id = 1
#log_bin = /var/log/mysql/mysql-bin.log
#binlog_do_db = include_database_name
bind-address = 127.0.0.1
```

Realizando el cambio se registra lo siguiente, tomando en cuenta que el valor de server-id es 2.

```
server-id = 2
log_bin = /var/log/mysql/mysql-bin.log
```

```
binlog_do_db = hospital
# bind-address = 127.0.0.1
```

Procedemos a reiniciar mysql.

```
root@Bdserver2/# sudo service mysql restart
```

Una vez hecho esto, ingresamos al shell para realizar otras configuraciones.

```
root@Bdserver2/# mysql -u root -p
```

De la misma manera en el servidor Bdserver1, se crea un usuario que servirá para realizar la replicación.

```
root@Bdserver2/# create user 'replicador'@'%' identified by 'password';
```

Seguido a esto, concedemos los permisos necesarios de usuario de replicación.

```
root@Bdserver2/# grant replication slave on *.* to 'replicador'@'%';
```

Ahora se registra la información del master status que se obtuvo en el servidor Bdserver1, ubicándola en la siguiente configuración a través del shell, haciendo esto, la replicación comienza a ejecutarse.

```
root@Bdserver2/# slave stop;
CHANGE MASTER TO MASTER_HOST = '192.168.0.60', MASTER_USER =
'replicador', MASTER_PASSWORD = 'password', MASTER_LOG_FILE = 'mysql-
bin.000063', MASTER_LOG_POS = 1553;
slave start;
```

Ahora se debe obtener el master status de Bdserver2, para registrarlo en Bdserver1.

```
root@Bdserver2/# show master status;
```



```

+-----+-----+-----+-----+
| File           | Position | Binlog_Do_DB | Binlog_Ignore_DB |
+-----+-----+-----+-----+
| mysql-bin.000051 |      1553 | hospital     |                   |
+-----+-----+-----+-----+
1 row in set (0.00 sec)

```

2.5.1.3 Finalizar configuración de réplica en bdserver1

Para finalizar la configuración de replicación, se debe registrar el master status de Bdserver2, ejecutando lo siguiente en Bdserver1.

```

root@Bdserver1/# slave stop;
CHANGE MASTER TO MASTER_HOST = '192.168.0.61', MASTER_USER =
'replicador', MASTER_PASSWORD = 'password', MASTER_LOG_FILE = 'mysql-
bin.000051', MASTER_LOG_POS = 1553;
slave start;

```

2.5.1.4 Prueba De Replicación Master Master

Se crea una Base de Datos en Bdserver2.

```

root@Bdserver2/# create database hospital;

```

Y la replicación se ejecuta creando la base de datos en Bdserver2 y Bdserver1.

2.5.2 Configuración servidores de balanceo de carga(HaproxyBd1 y HaproxyBd2)

Una vez configurados y levantados los servidores de replicación de Base de Datos, se procede a la configuración de los servidores de balanceo de carga.

Se realiza configuración IP en los servidores se edita el archivo *etc/network/interfaces*, utilizando el comando para edición de archivos *nano*.

```

root@HaproxyBd1/# nano etc/network/interfaces

```

A continuación, se configura Ip Address e Ip Virtual en Haproxybd1, de esta manera:

```
auto lo
iface lo inet loopback

# IP ADDRESS
auto eth0
iface eth0 inet static
address 192.168.0.50
netmask 255.255.255.0
network 192.168.0.0
broadcast 192.168.0.255
gateway 192.168.0.1

# IP VIRTUAL
auto eth0:1
iface eth0:1 inet static
address 192.168.0.21
netmask 255.255.255.0
```

Luego se configura Ip Address e Ip Virtual en Haproxybd2, de esta manera:

```
root@HaproxyBd2/# nano etc/network/interfaces
```

```
auto lo
iface lo inet loopback

# IP ADDRESS
auto eth0
iface eth0 inet static
address 192.168.0.51
netmask 255.255.255.0
network 192.168.0.0
broadcast 192.168.0.255
gateway 192.168.0.1

# IP VIRTUAL
auto eth0:1
iface eth0:1 inet static
address 192.168.0.21
netmask 255.255.255.0
```

2.5.2.1 Configurar servidores Mysql (Bdserver1 y Bdserver2)

La siguiente configuración debe ser ejecutada con la ip de los 2 balanceadores de carga BD, en los 2 servidores Mysql.

Se crean 2 usuarios adicionales para Haproxy, en los cuales, el primero tiene la función de verificar en qué estado se encuentra el servidor.

```
root@Bserverd1/# mysql -u root -p -e "INSERT INTO mysql.user  
(Host,User) values ('192.168.0.50','haproxy_check'); FLUSH PRIVILEGES;"
```

```
root@Bserverd1/# mysql -u root -p -e "INSERT INTO mysql.user  
(Host,User) values ('192.168.0.51','haproxy_check'); FLUSH PRIVILEGES;"
```

Ahora se requiere un usuario Mysql, al cual se le debe otorgar privilegios de root, para que tenga acceso desde Haproxy hacia el clúster de Base de Datos, de esta manera se tendrá un usuario independiente con todos estos privilegios

```
root@Bserverd2/# mysql -u root -p -e "GRANT ALL PRIVILEGES ON *.* TO  
'haproxy_root'@'10.0.0.50 IDENTIFIED BY 'password' WITH GRANT OPTION;  
FLUSH PRIVILEGES"
```

```
root@Bserverd2/# mysql -u root -p -e "GRANT ALL PRIVILEGES ON *.* TO  
'haproxy_root'@'10.0.0.51 IDENTIFIED BY 'password' WITH GRANT OPTION;  
FLUSH PRIVILEGES"
```

2.5.2.2 Instalación Mysql Client (HaproxyBd1 y HaproxyBd2)

En este punto se debe instalar Mysql Cliente en los Servidores de Balanceo de Carga BD para las conexiones respectivas.

```
root@HaproxyBd1/# apt-get install mysql-client
```

```
root@HaproxyBd2/# apt-get install mysql-client
```

Luego ejecutamos la siguiente línea de código para comprobar el funcionamiento de Mysql Client.

```
root@HaproxyBd1/# mysql -h 192.168.0.60 -u haproxy_root -p -e "SHOW
DATABASES"
```

```
root@HaproxyBd1/# mysql -h 192.168.0.61 -u haproxy_root -p -e "SHOW
DATABASES"
```

```
root@HaproxyBd2/# mysql -h 192.168.0.60 -u haproxy_root -p -e "SHOW
DATABASES"
```

```
root@HaproxyBd2/# mysql -h 192.168.0.61 -u haproxy_root -p -e "SHOW
DATABASES"
```

En todos los casos debe de mostrarse el listado de las Bases de Datos existentes en los servidores Bdserver1 y Bdserver2.

2.5.2.3 Instalación y Configuración Haproxy (HaproxyBd1 y HaproxyBd2)

Se procede a instalar Haproxy en los servidores HaproxyBd1 y HaproxyBd2.

```
root@HaproxyBd1/# apt-get install haproxy
```

```
root@HaproxyBd1/# apt-get install haproxy
```

Posteriormente se configura la herramienta HAPROXY para que se ejecute con el script de inicio.

```
nano /etc/default/haproxy
```

Se encuentra:

```
ENABLED = 0
```

Se cambia a:

```
ENABLED = 1
```

Ahora se ejecuta el script sin ningún parámetro para comprobar si el cambio se ha realizado de manera correcta.

```
service haproxy
usage: /etc/init.d/haproxy {start|stop|reload|restart|status}
```

Ahora se debe configurar el archivo `/etc/haproxy/haproxy.cfg`

Configuración global

```
global
    log 127.0.0.1 local2
    user haproxy
    group haproxy
```

```
defaults
    log global
    option dontlognull
    timeout connect 3000
    timeout server 5000
    timeout client 5000
```

Modo Tcp

```
listen mysql-cluster
    bind 0.0.0.0:3306
    mode tcp
    option mysql-check user haproxy_check
    balance rounrobin
    server Bdserver1 192.168.0.60:3306 check
    server Bdserver2 192.168.0.61:3306 check
```

Por último, se inicia los servicios de Haproxy.

```
/etc/init.d/haproxy start
```

2.5.2.4 Configuración De Heartbeat

Luego se configura HEARTBEAT a través del siguiente script *etc/ha.d/ha.cf*, en el cual se indica la Ip del otro servidor balanceador de carga y se colocan los nombres de todos los servidores Frontend que hagan la función de balanceador de carga.

HaproxyWeb1

```
keepalive      2
deadtime       10
udpport        694
bcast          eth0
mcast          eth0 225.0.0.1 694 1 0
ucast          eth0 192.168.0.51
logfacility     local0
auto_failback  on
node           HaproxyWeb1
node           HaproxyWeb2
```

HaproxyWeb2

```
keepalive      2
deadtime       10
udpport        694
bcast          eth0
mcast          eth0 225.0.0.1 694 1 0
ucast          eth0 192.168.0.50
logfacility     local0
auto_failback  on
node           HaproxyWeb1
node           HaproxyWeb2
```

Se configura el siguiente archivo *etc/ha.d/authkeys* para permitir a los nodos miembros identificar o autenticarse dentro del cluster.

```
auth 3
1 crc
2 sha1 yourpasswoedhere
3 md5 yourpasswordhere
```

Luego se cambia los permisos.

```
chmod 600 etc/ha.d/authkeys
```

Finalmente se indica a HEARTBEAT la Ip Virtual del Servidor, configurando el archivo *etc/ha.d/haresources* de ambos servidores balanceadores de carga.

```
HaproxyBd1 \  
IPaddr :: 192.168.0.21/24/eth0
```

```
HaproxyBd2 \  
IPaddr :: 192.168.0.21/24/eth0
```

2.5.2.5 Prueba de Balanceo de Carga

Una vez realizadas todas las configuraciones del clúster de Base de Datos, se procede a verificar que el balancero de carga se esté ejecutando de acuerdo a lo previsto. Para esto, se consulta la variable `server_id`, de esta manera.

```
root@HaproxyBd1/# mysql -h 127.0.0.1 -u haproxy_root -p -e "show  
variables like 'server_id'"
```

Se obtiene este resultado, indicando la activación de `Bdserver1`.

```
+-----+-----+  
| Variable_name | Value |  
+-----+-----+  
| server_id     | 1     |  
+-----+-----+
```

Luego se realiza por segunda vez la consulta.

```
root@HaproxyBd1/# mysql -h 127.0.0.1 -u haproxy_root -p -e "show  
variables like 'server_id'"
```

Obteniendo este resultado, indicando la activación de `Bdserver2`.

```
+-----+-----+  
| Variable_name | Value |
```

```
+-----+-----+
| server_id      | 2      |
+-----+-----+
```

Por tercera vez se realiza la consulta.

```
root@HaproxyBd1/# mysql -h 127.0.0.1 -u haproxy_root -p -e "show
variables like 'server_id'"
```

Obteniendo este resultado, indicando la activación de Bdserver1 .

```
+-----+-----+
| Variable_name  | Value  |
+-----+-----+
| server_id      | 1      |
+-----+-----+
```

De esta manera se demuestra que el balanceo de carga de Base de Datos, está funcionando correctamente, apuntando las peticiones equitativamente entre ambos servidores Bdserver1 y Bdserver2.

CAPÍTULO III. PRUEBAS A LA SOLUCIÓN

3.1 Aplicación web

Los grandes volúmenes de información que tienen que manejar actualmente las aplicaciones de escritorio o web; y mucho más las apps web, exigen a las mismas que dicha información sea tratada bajo ciertos estándares y parámetros que puedan facilitar la obtención de resultados de calidad, lo cual se ve reflejado en un mejor servicio o producto para el usuario.

De esta manera, se crea una aplicación web que pueda ser capaz de complementarse con la arquitectura tecnológica que se está implementando en este proyecto, y así puedan desempeñarse con las características propias de las estructuras a gran escala, como son alta disponibilidad, balanceo de carga, persistencia de sesiones, replicación en base de datos, redundancia de servidores, etc.

De acuerdo a estos fundamentos se desarrolla una aplicación web mediante el framework Django, para el manejo de un sistema hospitalario de historias clínicas y agendamiento de citas médicas.

3.1.1 Análisis de requerimientos

3.1.1.1 Requerimientos funcionales

- El Sistema debe entregar información de Historias Clínicas y Agendamiento de citas médicas de los pacientes del Hospital.
- Se manejarán 3 usuarios dentro del sistema (Administrador, Agendador y Médico).
- El usuario (médico, agendador) a su ingreso al sistema de Historias Clínicas, lo hará una vez sea autenticado y validado en el sistema.
- El Sistema permitirá reportes y consultas secuenciales.
- El Sistema brindará un menú de opciones para el registro y edición de Historias Clínicas y Agendamiento.

3.1.1.2 Requerimientos no funcionales

❖ Interfaz de usuario

La estructura de los formularios debe tener características intuitivas y sencillas para el usuario, de tal manera que se presente una interfaz amigable y fácil de utilizar.

❖ **Disponibilidad**

El sistema debe estar operativo y disponible 24/7, sin presentar un único punto de fallo, tanto en hardware como en software, es decir tener Alta Disponibilidad.

❖ **Requisitos de desempeño**

- Los procesos, transacciones o consultas en el sistema deben tener un tiempo de respuesta inmediato, en peticiones comunes y en peticiones concurrentes.
- Para tener un óptimo rendimiento en el procesamiento de los datos, se debe realizar balanceo de carga en la Aplicación y en la Base de Datos.
- Debe poseer atributos de persistencia de sesiones.

❖ **Seguridad**

Para resguardar la información, se debe contar con un sistema de replicación automática de la Base de Datos en un Server Backup.

❖ **Calidad**

La app debe ser diseñada en base a estándares de flexibilidad para agregar requerimientos al desarrollo.

❖ **Escalabilidad**

El sistema debe tener la capacidad de escalar sin problemas a futuro.

3.1.2 Diseño de aplicación

3.1.2.1 Diagrama de clases

Clases

Grupo_Cultural: definición étnica del paciente

Instrucción_Educativa: nivel académico del paciente

Pais: país de origen del paciente

Provincia: provincia del paciente
Ciudad: ciudad paciente
Afinidad: parentesco de la persona con el paciente
Pariente: datos básicos de familiar del paciente
Paciente: Información de paciente
Usuario: datos de usuario de la app web
Agendador: datos de usuario agendador
Especialidad: especialidad del médico
Medico: datos del médico
Consultorio: datos del consultorio
Enfermedad: datos de la enfermedad
Diagnostico: datos del diagnóstico
Turno: datos del turno del paciente
Historia_Clinica: información de historia clínica del paciente

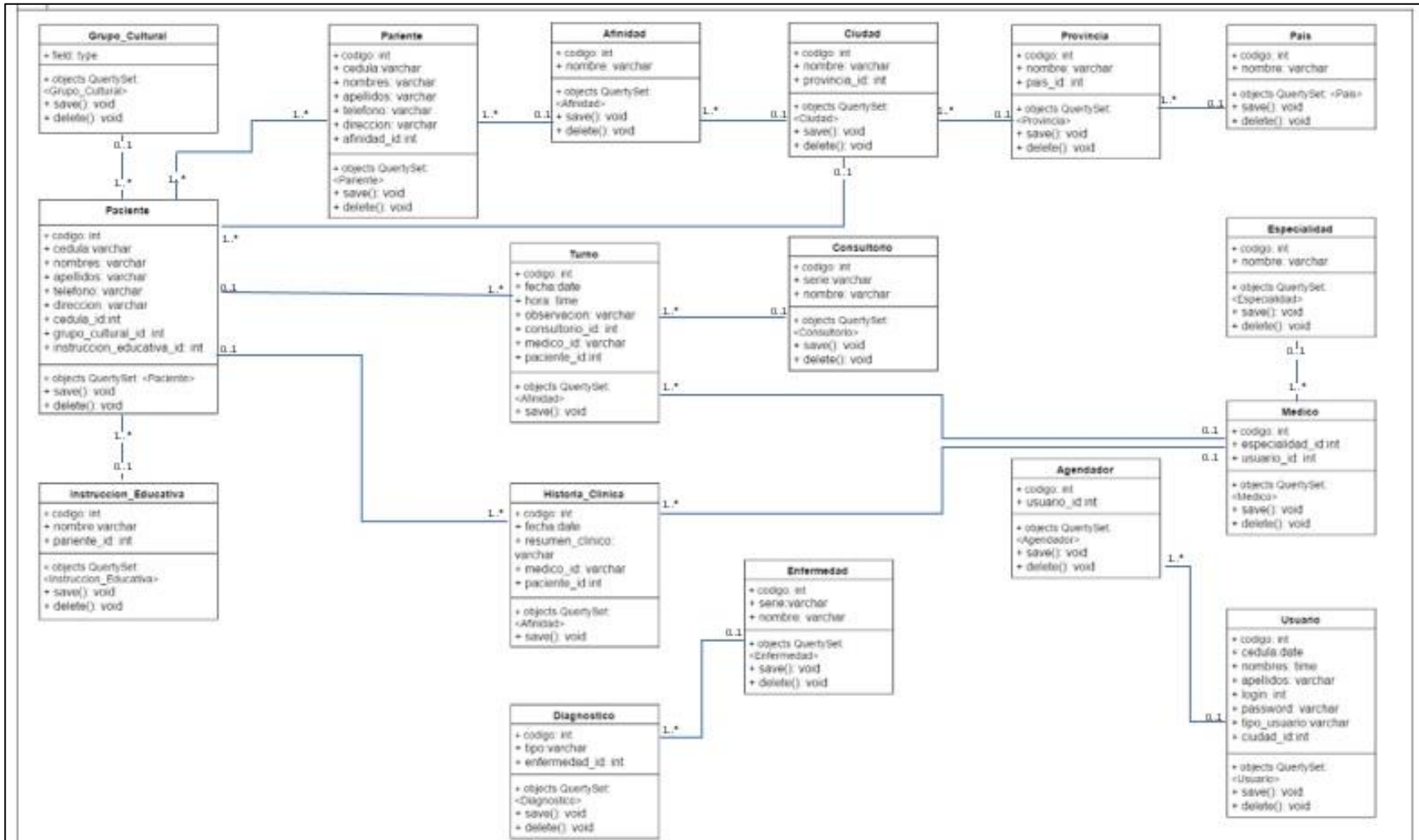


Figura 17: Diagrama de clases
 Fuente: El autor
 Elaborado por: El autor

3.1.2.2 Diagrama de base de datos

Grupo_Cultural: Registra información étnica del paciente

Instrucción_Educativa: Registra información sobre nivel académico del paciente

Pais: Registra código y nombre del país de origen del paciente

Provincia: Registra código y nombre de provincia del paciente

Ciudad: Registra código y nombre de la ciudad del paciente

Afinidad: Registra información sobre el parentesco de la persona con el paciente

Pariente: Registra datos básicos de familiar del paciente

Paciente: Registra toda la información del paciente

Usuario: Registra códigos y datos de usuario de la app web

Agendador: Registra datos de usuario agendador

Especialidad: Registra código y nombre de la especialidad del médico

Medico: Registra datos del médico

Consultorio: Registra datos del consultorio

Enfermedad: Registra datos de la enfermedad

Diagnostico: Registra datos del diagnóstico

Turno: Registra datos del turno del paciente

Historia_Clinica: Registra información de historia clínica del paciente

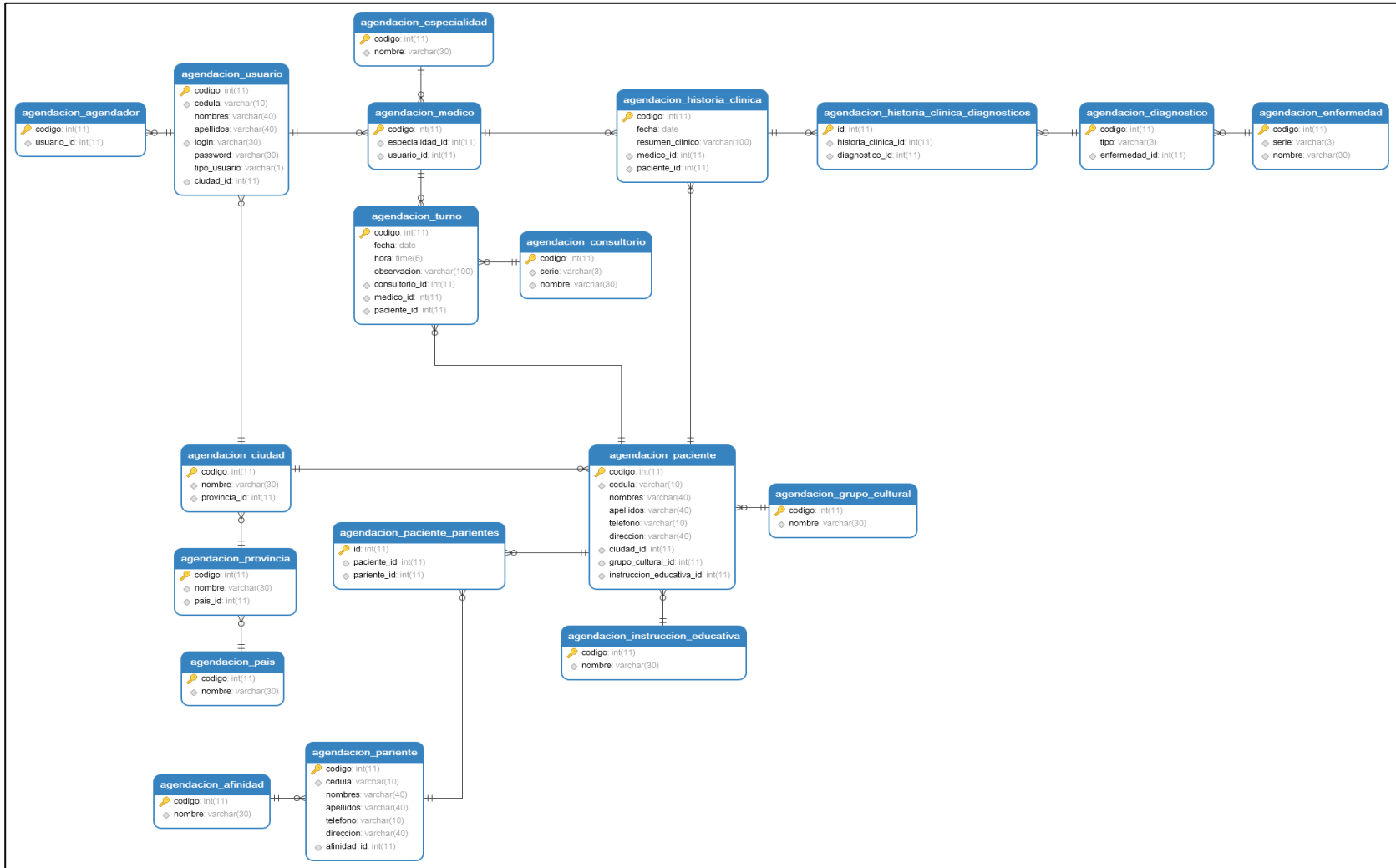


Figura 18: Diagrama de base de datos

Fuente: El autor

Elaborado por: El autor

3.1.2.3 Diagrama de vistas

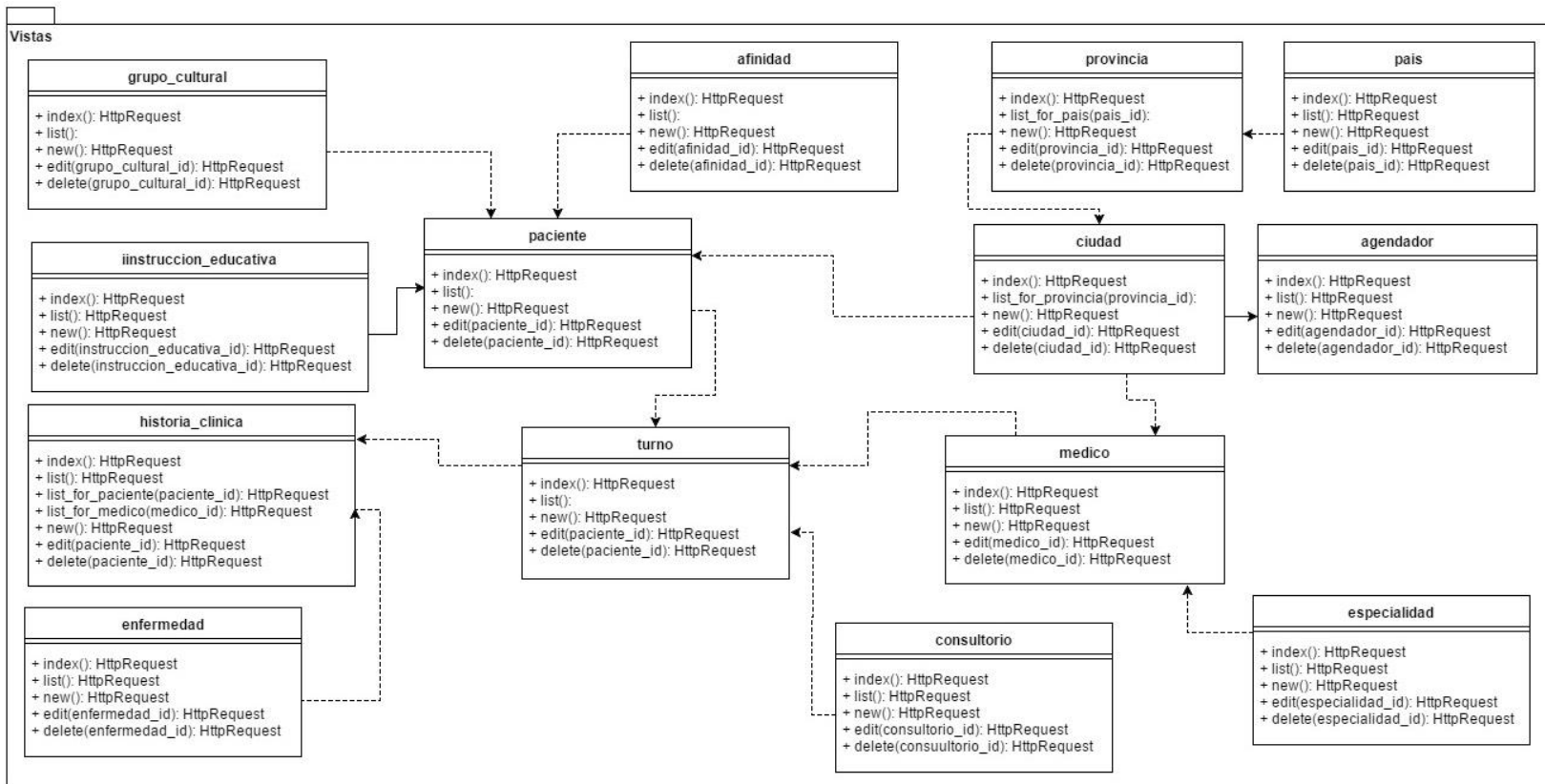


Figura 19: Diagrama de vistas

Fuente: El autor

Elaborado por: El autor

3.1.2.4 Diagrama de plantillas y menú

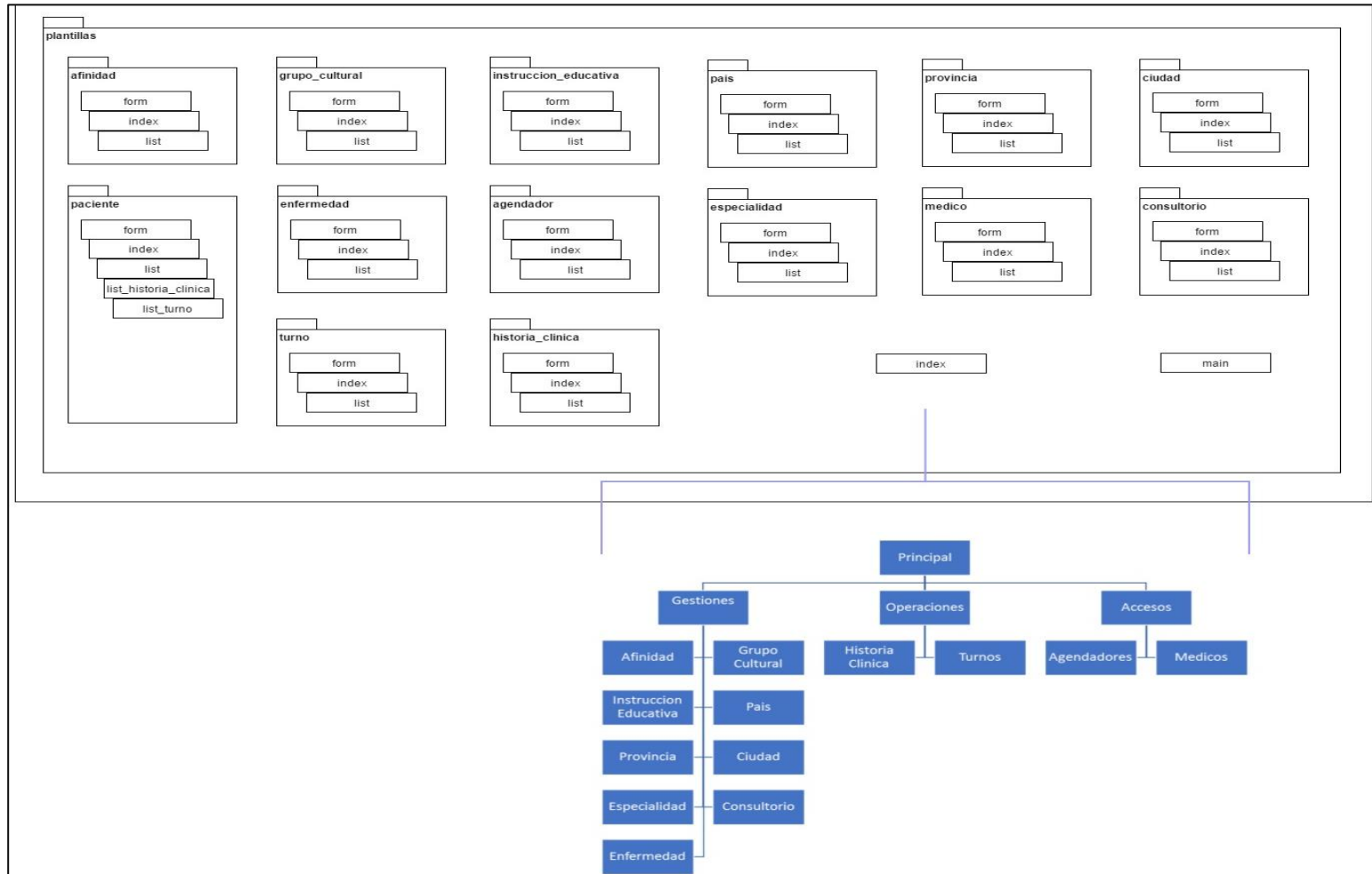


Figura 20: Diagrama de plantillas y menú
 Fuente: El autor
 Elaborado por: El autor

3.1.3 Datos de la base de datos

Cédula	Apellidos	Nombres
0790289746	GALLEGOS LUCERO	MARIA LUZMILA
0791278178	ORTEDA PILCO	MARIA ROSARIO
0790289837	TANCAZO DIAZ	FLABERTO ROSALINDO
0796915195	CASTRO GRANDA	RICKY ANDREE
0792229522	NOROA LIMA	JACINTO
9801283557	BRAYO	JOSE NICOMEDES
1303428896	DELGADO CAICEDO	BLANCA ANGELA
0792928854	LAMBERT ARMijos	ROSA ISABEL
0791637886	FLORES PARRALES	LUISA ISABEL
0792986188	CALDERIN CALDERON	ADOLFINA DEL CISNE

Página 1 de 2396 --- 23954 Resultado(s)

NICOMEDES

A ANGELA

SABEL

SABEL

INA DEL CISNE

Página 1 de 2396 --- 23954 Resultado(s)

Figura 21: Base de datos llena

Fuente: El autor

Elaborado por: El autor

Para realizar la comprobación del balanceo de carga entre los servidores reales web y de base de datos, se procedió a realizar el llenado de la base de datos, contando con más de 23.000 registros en la tabla pacientes.

3.2 Pruebas de carga y rendimiento a la arquitectura tecnológica

3.2.1 Encendido de los servidores del clúster web y base de datos

Se realiza el encendido de todos los servidores virtuales que se encuentran instalados dentro del VirtualBox, como se muestra en la siguiente figura.

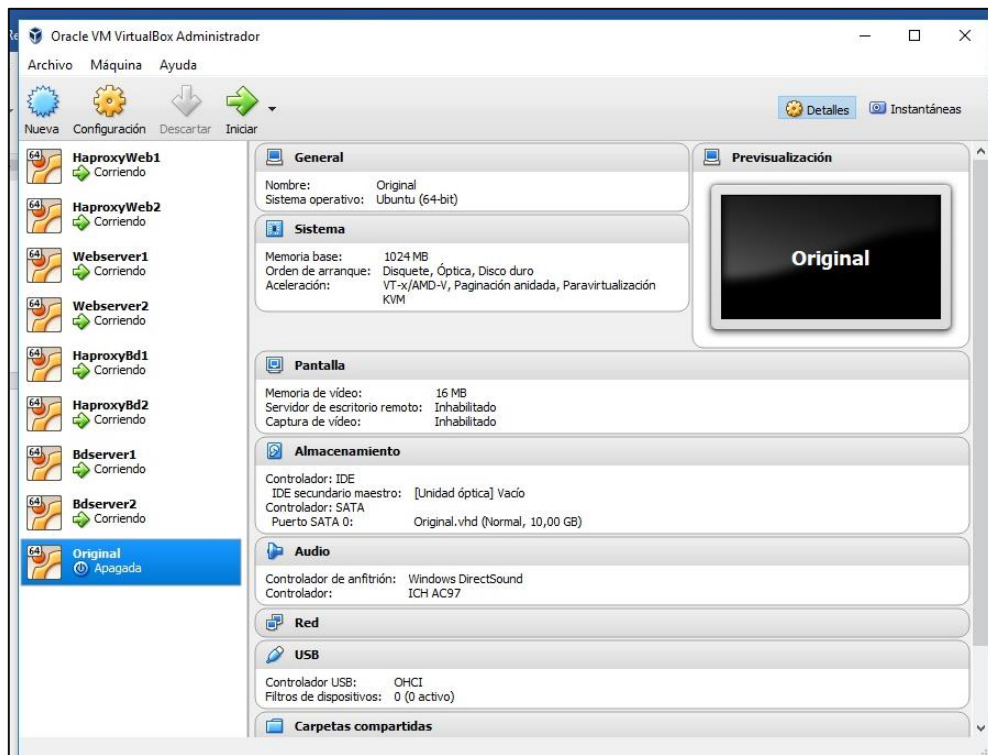


Figura 22: Servidores encendidos en VirtualBox
 Fuente: El autor
 Elaborado por: El autor

Aquí se muestran los 8 servidores encendidos y ejecutándose.

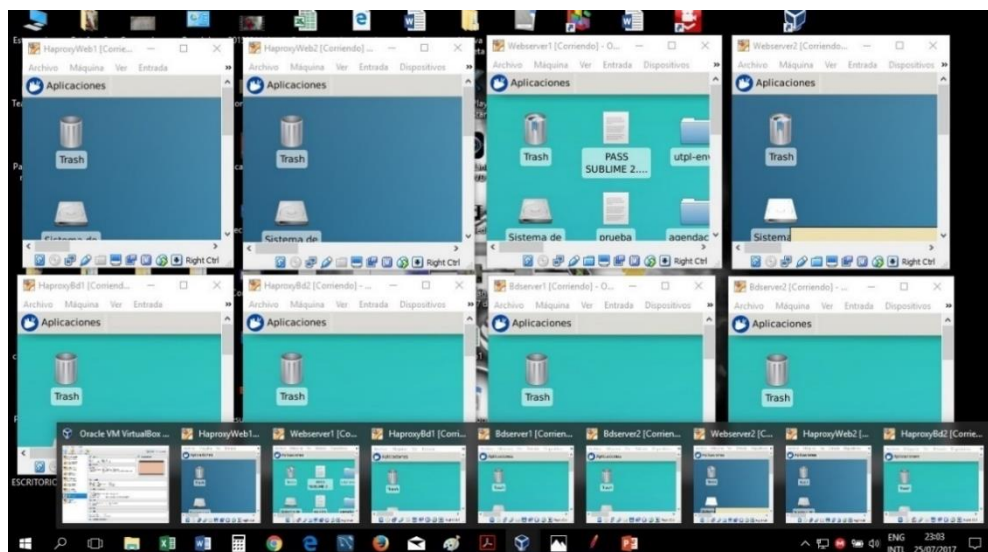


Figura 23: Servidores ejecutándose
 Fuente: El autor
 Elaborado por: El autor

De acuerdo a la figura 23 El número de peticiones para el Webserver 1 es 1034, y para el Webserver 2 es 1035

3.2.2 Prueba 1 realizada con herramienta Jmeter con 1 servidor web

Esta prueba se realiza con 7 de los 8 servidores que componen nuestra granja de servidores, específicamente de la siguiente manera:

Tabla 10: Servidores disponibles en prueba1

NOMBRE	CLÚSTER WEB		CLÚSTER BASE DATOS	
	BALANCEADOR CARGA WEB	SERVIDOR REAL WEB	BALANCEADOR CARGA BASE DATOS	SERVIDOR REAL BASE DATOS
HaproxyWeb1	✓			
HaproxyWeb2	✓			
Webserver1		✓		
Webserver2		X		
HaproxyBd1			✓	
HaproxyBd2			✓	
Bdserver1				✓
Bdserver2				✓

Fuente: El autor

Elaborado por: El autor

El objetivo de ejecutar la prueba con 1 solo servidor real web (Webserver1), es observar el rendimiento del clúster al recibir toda la carga de peticiones concurrentes de usuarios (1000 usuarios), en un único servidor. Posteriormente se realizará esta misma prueba, pero con los 2 servidores reales web y se hará la respectiva comparación de rendimiento y balanceo de carga.

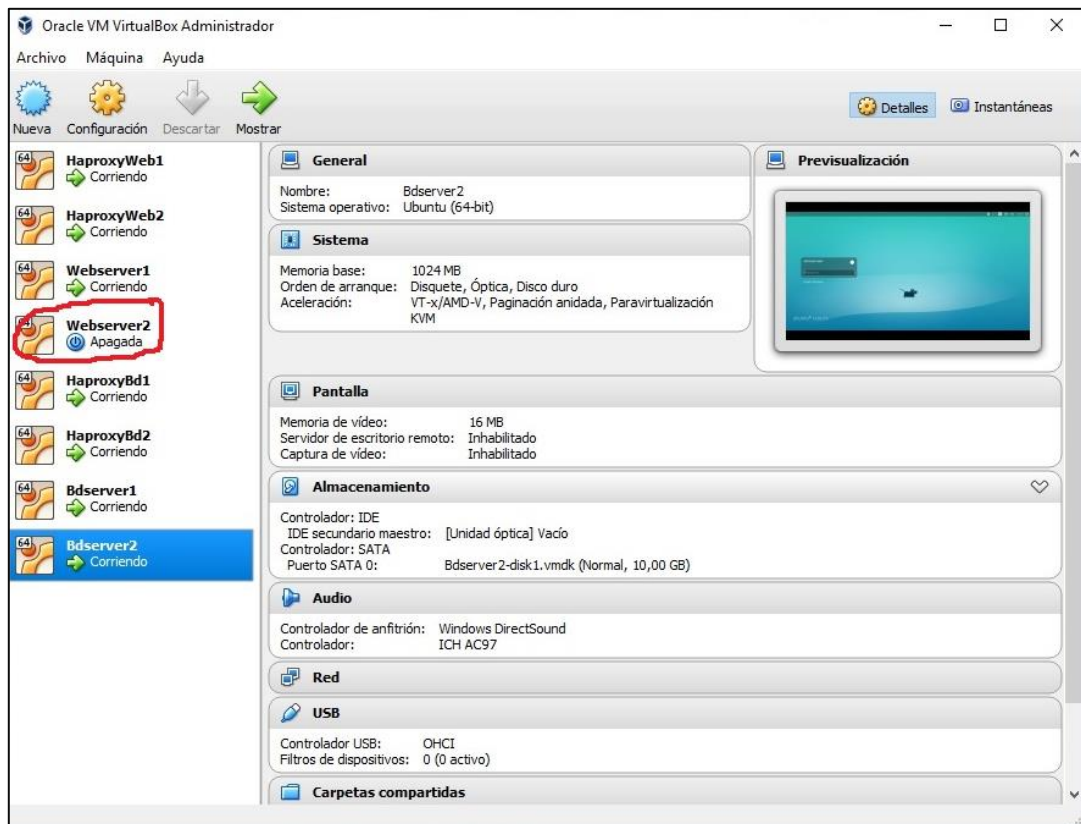


Figura 24: Servidor Webserver2 apagado

Fuente: El autor

Elaborado por: El autor

3.2.2.1 Revisión de estadísticas de sesiones y balanceo de carga previo a las pruebas

Antes de realizar las pruebas de balanceo de carga y pruebas de estrés a la aplicación web, se procede a revisar las estadísticas de peticiones y balanceo de carga, para verificar la cantidad de peticiones que tiene cada servidor web. Para ingresar a dichas estadísticas, lo hacemos por a través de la siguiente dirección:

<http://192.168.0.20/haproxy?stats>

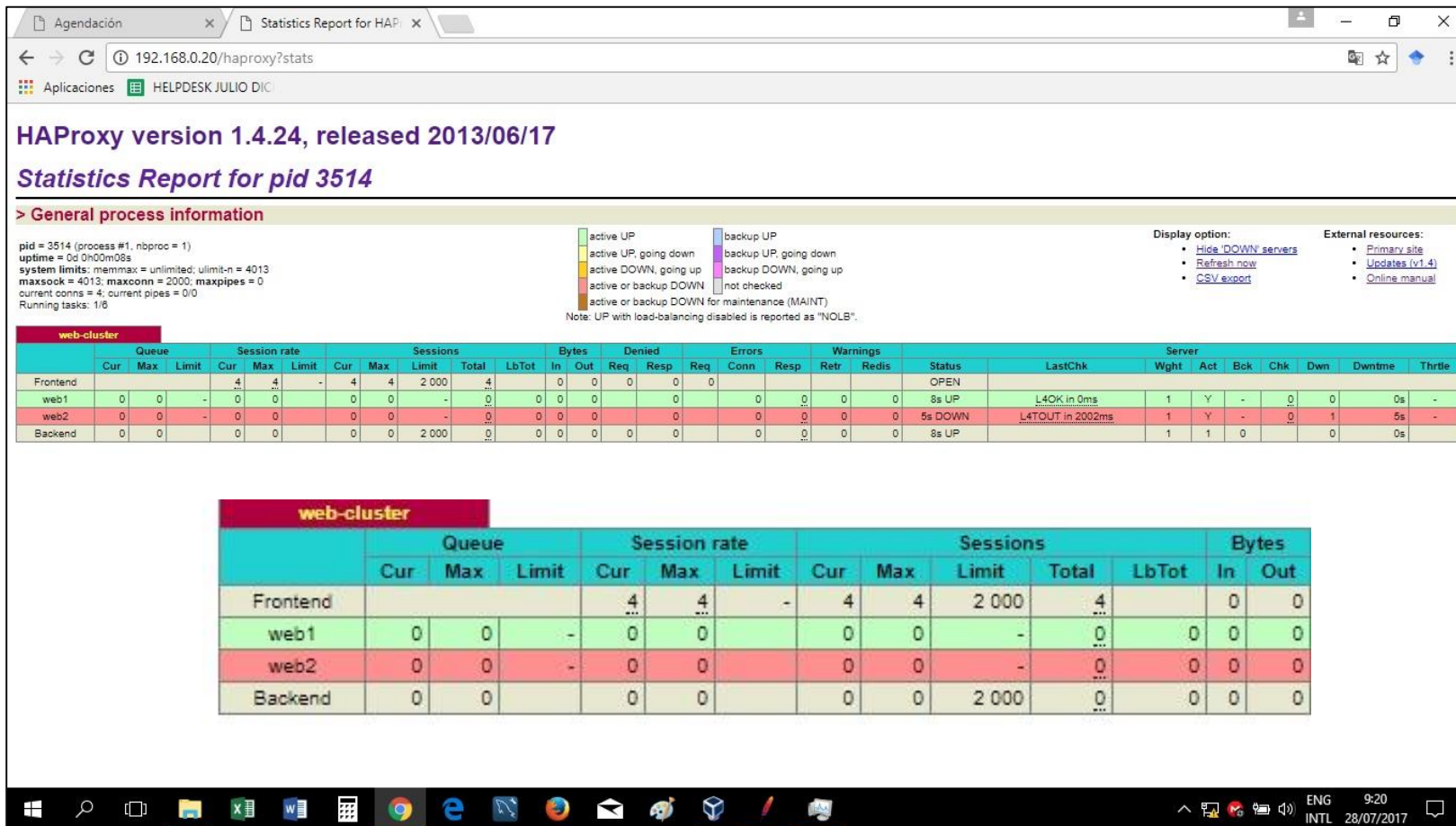


Figura 25: Reporte estadísticas Haproxy previo a la prueba1
 Fuente: El autor
 Elaborado por: El autor

3.2.2.2 Configuración de plan de pruebas con Jmeter

3.2.2.2.1 Plan de Pruebas

- **Grupo de usuarios:** se crea 1 Thread Group
- **Transacciones:** se genera 1 transacción para ingreso a la misma (192.168.0.20/agendacion/medico/).

3.2.2.2.2 Parámetros

- **Cantidad de usuarios:** 1000 usuarios
- **Duración de período de arribos:** 1 segundo
- **Cantidad de transacciones:** 1 transacción realizada por 1000 hilos o usuarios.
- **Protocolo:** Protocolo Http

Básicamente se crea un grupo de usuarios en Jmeter llamado Hospital, dentro del cual se establecen la cantidad de usuarios (1000 usuarios) que harán las peticiones concurrentes a la app; éstas peticiones serán enviadas por Jmeter y recibidas por nuestra arquitectura tecnológica en 1 segundo, haciendo esta operación 1 vez. Lo cual se aprecia en la figura a continuación:

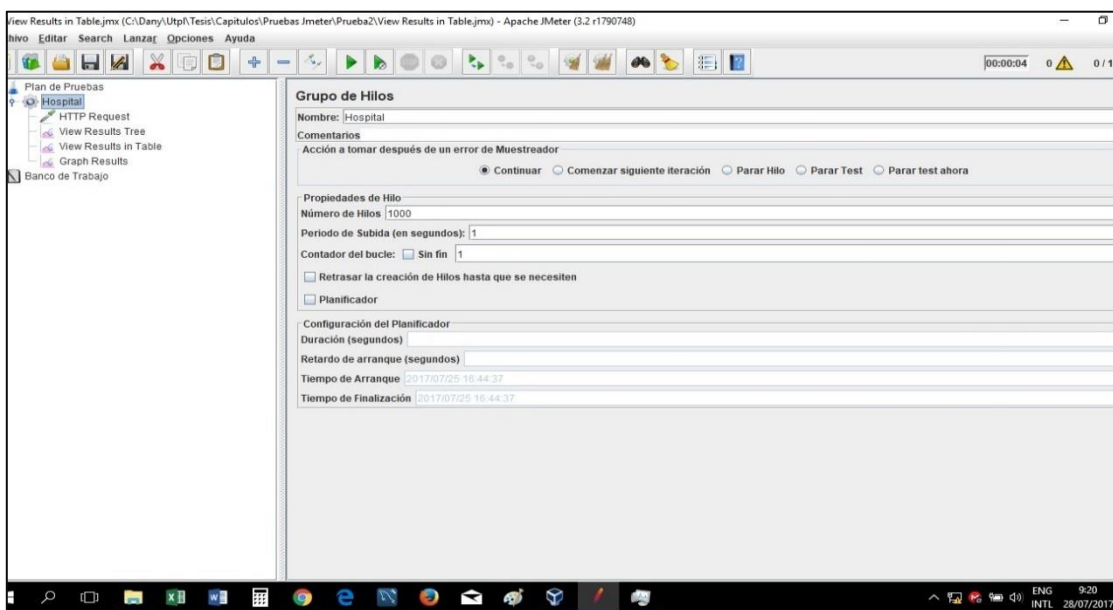


Figura 26: Configuración plan de pruebas

Fuente: El autor

Elaborado por: El autor

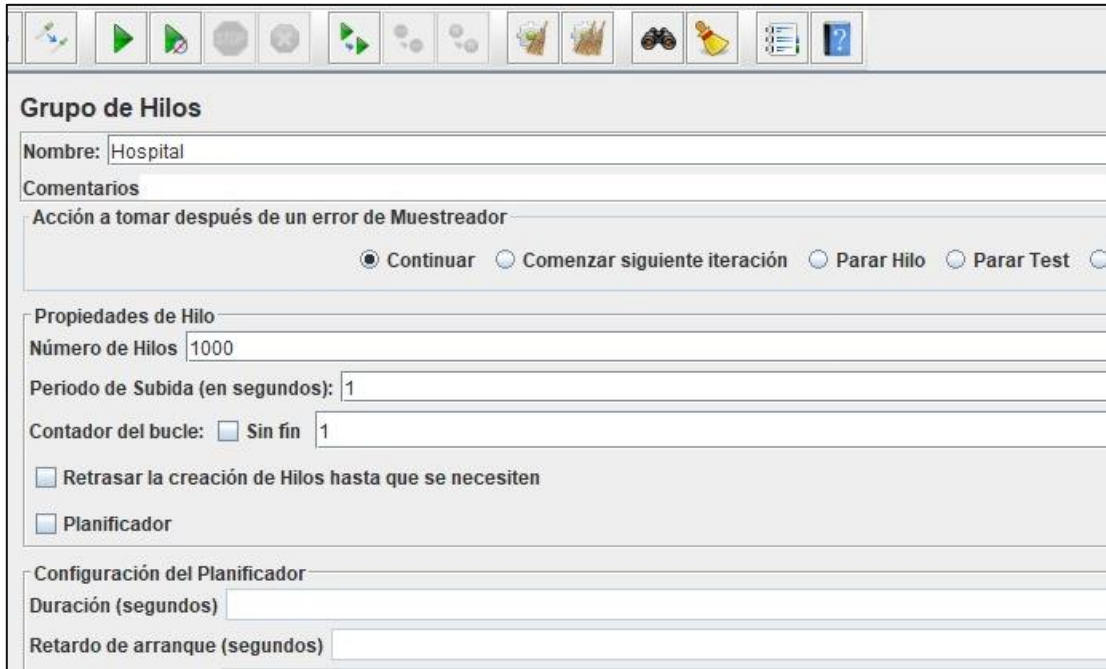


Figura 27: Número de usuarios concurrentes
Fuente: El autor
Elaborado por: El autor

A continuación, se crea una conexión Http, donde se configura la dirección ip, 192.168.0.20 que es la ip virtual para acceder al Frontend de la app, y a la que apuntan las peticiones y la url o path específico.

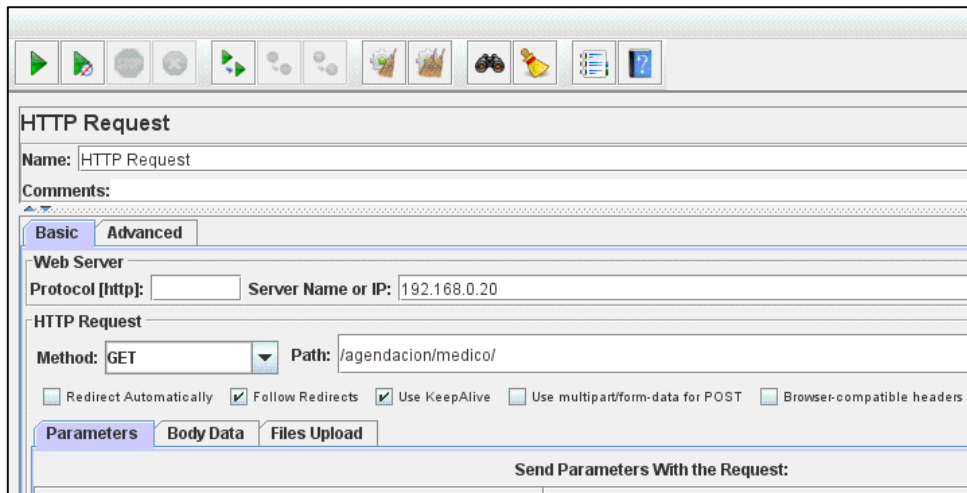


Figura 28: Configuración Ip y path
Fuente: El autor
Elaborado por: El autor

Posteriormente, se crean los *Listener*, estos son las vistas en las que se presentarán los resultados obtenidos luego de realizar la prueba de estrés a la aplicación.

Estas vistas son:

- View results tree
- View results in table
- Graph Results

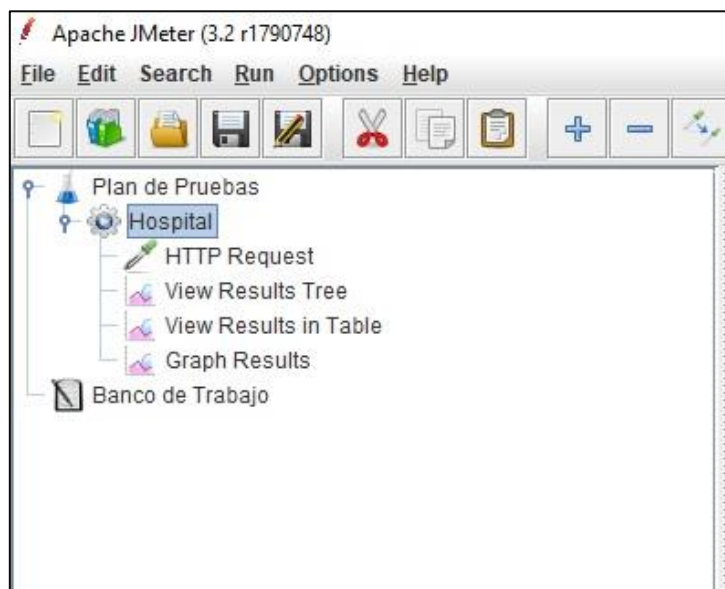


Figura 29: Creación de listener
Fuente: El autor
Elaborado por: El autor

3.2.2.3 Ejecución del plan de pruebas Jmeter

Una vez configurado el plan de pruebas en la herramienta Jmeter, se procede a la ejecución del mismo, haciendo clic en el ícono verde Start.

Como se muestra a continuación en la figura 30.

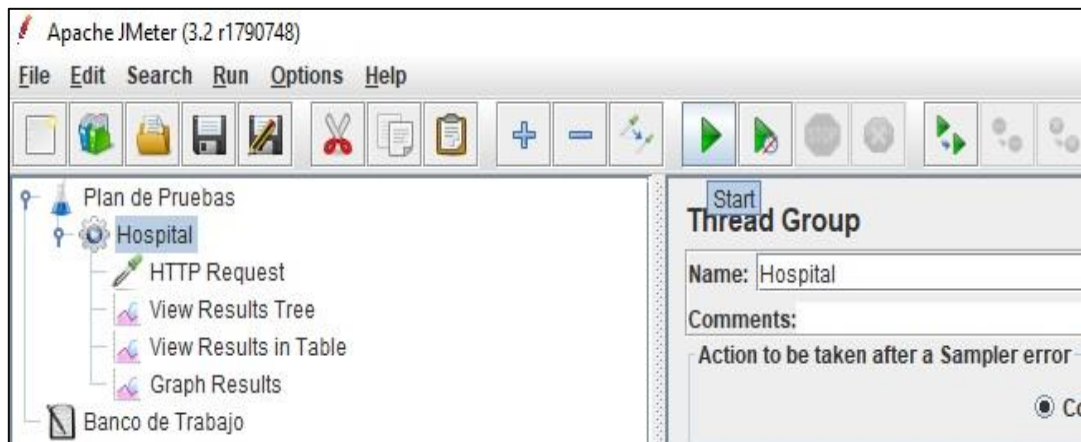


Figura 30: Ejecución de pruebas en Jmeter
 Fuente: El autor
 Elaborado por: El autor

Recordando que la prueba consiste en la ejecución de peticiones concurrentes de 1000 usuarios durante 1 segundo.

3.2.2.4 Resultados del plan de pruebas con Jmeter

Una vez ejecutada la prueba, Jmeter presenta los resultados a través de los 3 listener que se configuró al inicio (View Results Tree, View Results in Table, Graph Results)

3.2.2.4.1 Resultados en view results tree

En esta vista de resultados, se muestra de manera individual a manera de árbol, las peticiones realizadas, cada una con el detalle de la solicitud de acceso a la aplicación web, como, por ejemplo: Número de petición, inicio petición, tiempo de carga, latencia, tamaño de la petición, errores, etc.

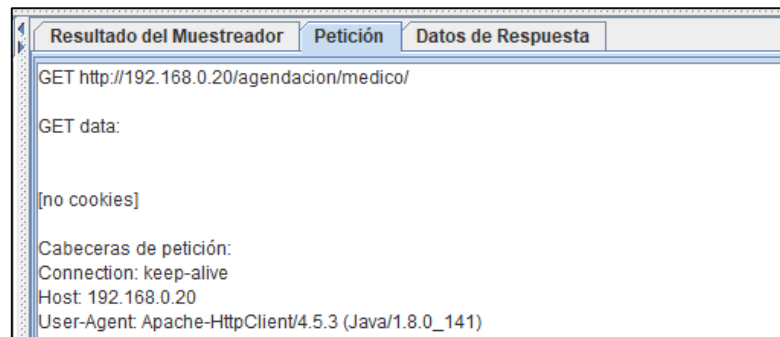


Figura 31: Petición url
 Fuente: El autor
 Elaborado por: El autor

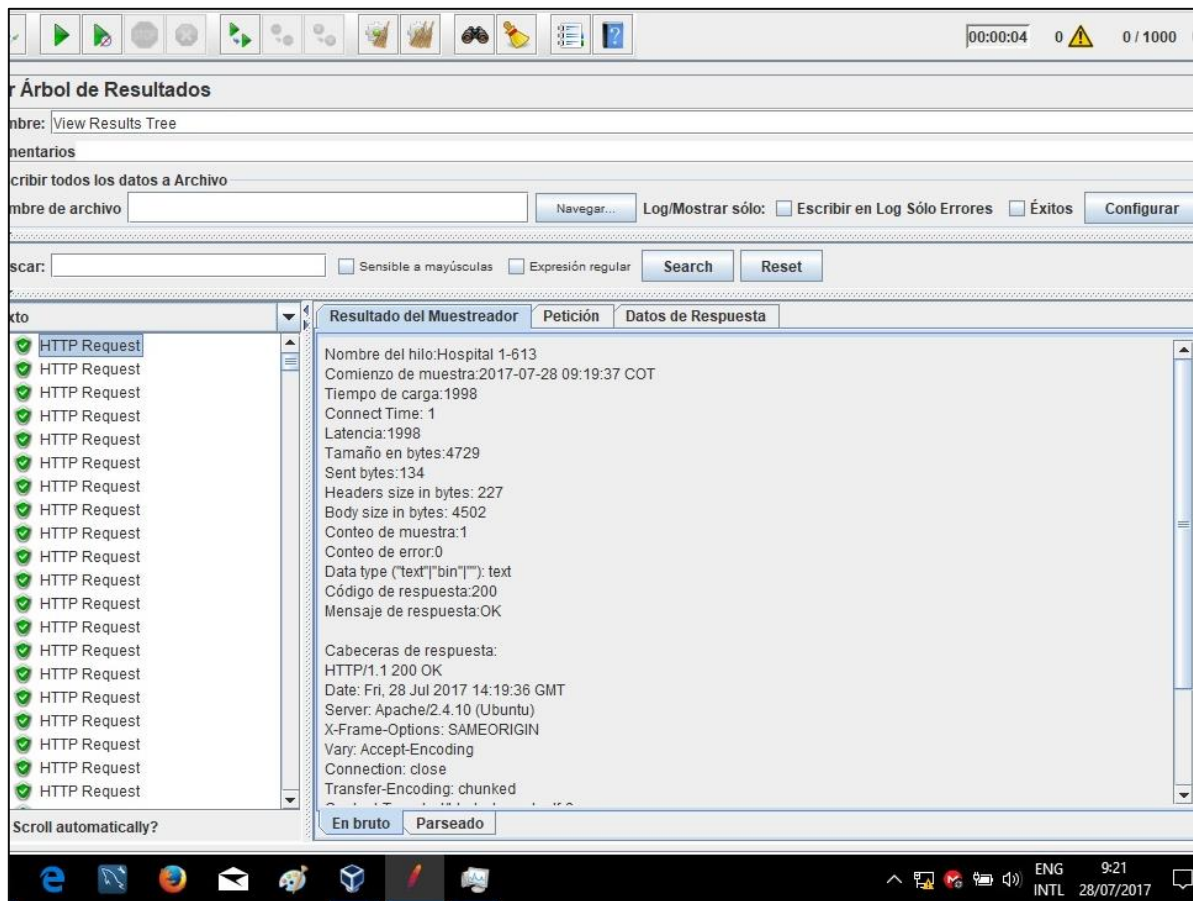


Figura 32: Resultados vista en árbol
Fuente: El autor
Elaborado por: El autor

La figura 32, evidencia que la primera petición en ingresar a la arquitectura tecnológica fue recibida correctamente sin errores a las 09h19:37, tiene un tamaño de 4729 bytes, latencia de 1998, bytes enviados 134.

Luego en la figura 33, se aprecia que la petición 476, tiene estado de respuesta correcto a las 09h19:37, tamaño en bytes 4729, latencia 3300 y 134 bytes enviados.

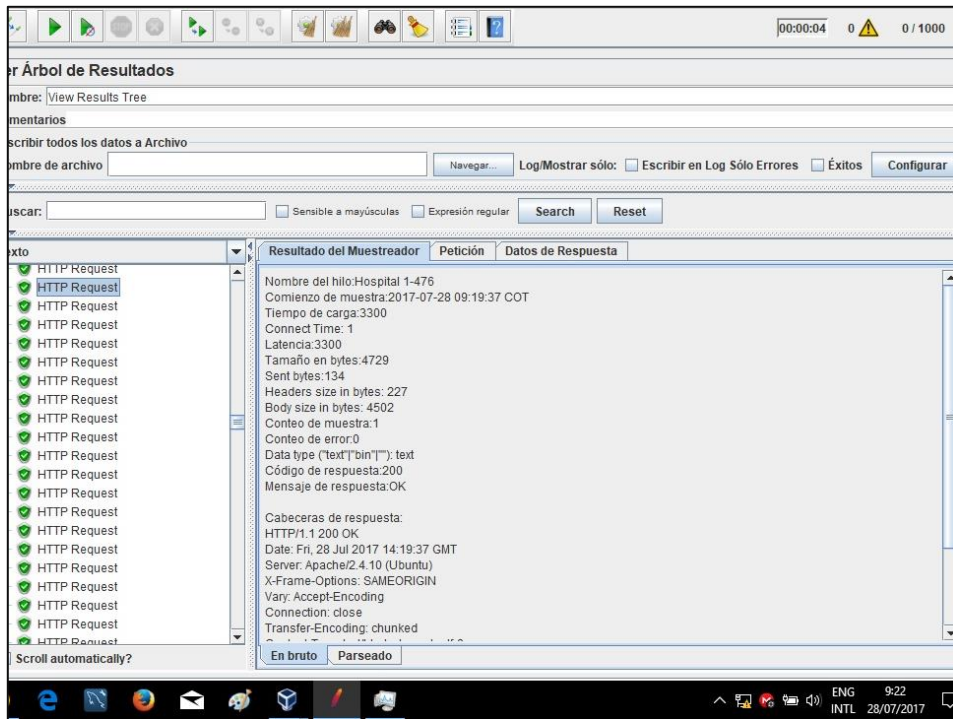


Figura 33: Resultados vista en árbol

Fuente: El autor

Elaborado por: El autor

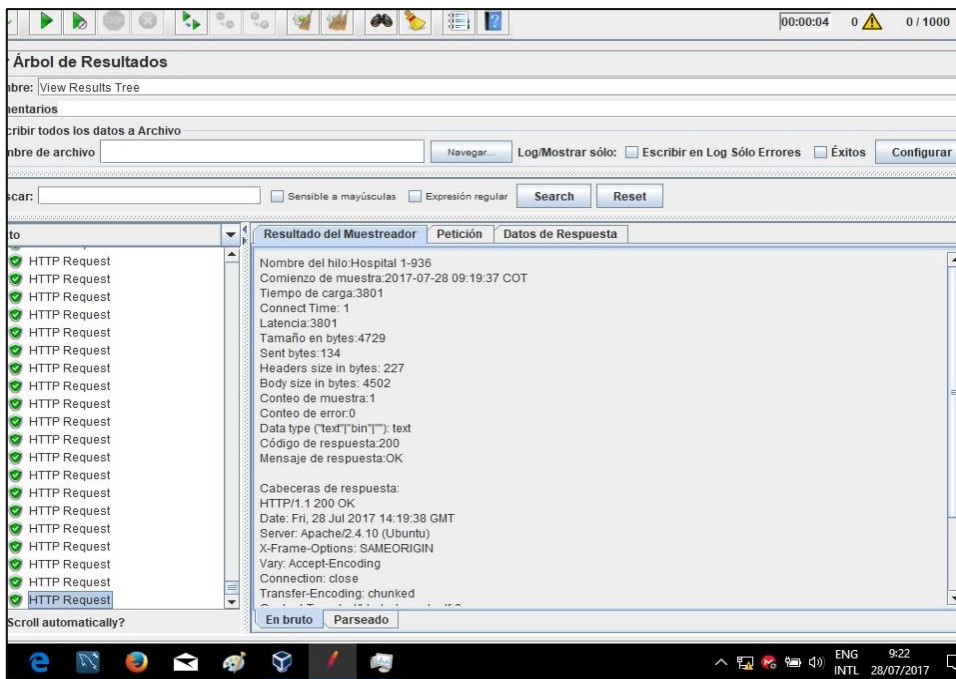


Figura 34: Resultados vista en árbol

Fuente: El autor

Elaborado por: El autor

Por último, en la figura 34 se demuestra, que la petición 936, tiene estado de respuesta correcto a las 09h19:37, tamaño en bytes 4729, latencia 3801 y 134 bytes enviados.

3.2.2.4.2 Resultados en vista en tabla

En esta vista los resultados se muestran en forma grupal en una tabla, detallando el estado de cada una de las peticiones realizadas por los 1000 usuarios.

Se puede apreciar en el reporte que las 1000 peticiones fueron procesadas completas, sin ningún error, en un tiempo total de 4 segundos.

	Tiempo de com...	Nombre del hilo	Etiqueta	Tiempo de Mue...	Estado	Bytes	Sent Bytes	Latency	Connect Time(...)
1	09:19:36.593	Hospital 1-2	HTTP Request	246	✓	4729	134	246	1
2	09:19:36.642	Hospital 1-40	HTTP Request	197	✓	4704	134	197	1
3	09:19:36.598	Hospital 1-7	HTTP Request	242	✓	4729	134	242	1
4	09:19:36.610	Hospital 1-16	HTTP Request	231	✓	4729	134	231	1
5	09:19:36.604	Hospital 1-12	HTTP Request	237	✓	4729	134	237	1
6	09:19:36.599	Hospital 1-8	HTTP Request	243	✓	4729	134	242	1
7	09:19:36.617	Hospital 1-20	HTTP Request	225	✓	4729	134	225	1
8	09:19:36.620	Hospital 1-24	HTTP Request	223	✓	4729	134	223	2
9	09:19:36.624	Hospital 1-25	HTTP Request	219	✓	4729	134	219	1
10	09:19:36.615	Hospital 1-18	HTTP Request	229	✓	4729	134	229	2
11	09:19:36.623	Hospital 1-28	HTTP Request	221	✓	4729	134	221	2
12	09:19:36.615	Hospital 1-21	HTTP Request	229	✓	4704	134	229	1
13	09:19:36.615	Hospital 1-19	HTTP Request	229	✓	4704	134	229	1
14	09:19:36.607	Hospital 1-13	HTTP Request	237	✓	4704	134	237	2
15	09:19:36.603	Hospital 1-10	HTTP Request	242	✓	4704	134	242	1
16	09:19:36.613	Hospital 1-17	HTTP Request	232	✓	4729	134	232	0
17	09:19:36.625	Hospital 1-27	HTTP Request	220	✓	4729	134	220	3
18	09:19:36.617	Hospital 1-22	HTTP Request	228	✓	4729	134	228	0
19	09:19:36.628	Hospital 1-32	HTTP Request	219	✓	4704	134	219	1
20	09:19:36.603	Hospital 1-11	HTTP Request	245	✓	4729	134	245	1
21	09:19:36.620	Hospital 1-23	HTTP Request	228	✓	4704	134	228	2
22	09:19:36.609	Hospital 1-15	HTTP Request	240	✓	4704	134	240	1
23	09:19:36.608	Hospital 1-14	HTTP Request	241	✓	4704	134	241	2
24	09:19:36.594	Hospital 1-3	HTTP Request	322	✓	4723	134	322	1
25	09:19:36.624	Hospital 1-26	HTTP Request	348	✓	4729	134	348	1
26	09:19:36.625	Hospital 1-30	HTTP Request	350	✓	4729	134	350	3
27	09:19:36.624	Hospital 1-29	HTTP Request	366	✓	4729	134	366	2

Child samples? No. de Muestras 1000 Última Muestra 3801 Media 2172 Desviación 1159

Figura 35: Resultados vista en tabla

Fuente: El autor

Elaborado por: El autor

Según la figura 35, se observa que la primera petición enviada por el usuario 2, se realizó básicamente en un tiempo de 246 ms, latencia 246 y bytes enviados 134, siendo procesada a las 09h19:36:593.

La figura 36, indica que la petición enviada por el usuario 441 se realizó en un tiempo de 1916 ms, latencia 1916, y 134 bytes enviados. siendo procesada a las 09h19:37:240

#	Tiempo de com...	Nombre del hilo	Etiqueta	Tiempo de Mue...	Estado	Bytes	Sent Bytes	Latency	Connect Time(...)
441	09:19:37.240	Hospital 1-555	HTTP Request	1916	✓	4729	134	1916	1
442	09:19:37.803	Hospital 1-701	HTTP Request	1354	✓	4729	134	1354	2
443	09:19:36.873	Hospital 1-241	HTTP Request	2290	✓	4729	134	2290	1
444	09:19:37.130	Hospital 1-459	HTTP Request	2034	✓	4729	134	2034	8
445	09:19:37.200	Hospital 1-521	HTTP Request	1967	✓	4729	134	1967	2
446	09:19:36.871	Hospital 1-240	HTTP Request	2297	✓	4729	134	2296	1
447	09:19:37.425	Hospital 1-585	HTTP Request	1764	✓	4729	134	1764	0
448	09:19:37.230	Hospital 1-546	HTTP Request	1964	✓	4729	134	1964	1
449	09:19:37.193	Hospital 1-514	HTTP Request	2014	✓	4729	134	2013	1
450	09:19:36.868	Hospital 1-236	HTTP Request	2341	✓	4729	134	2340	4
451	09:19:37.455	Hospital 1-612	HTTP Request	1764	✓	4729	134	1764	1
452	09:19:37.430	Hospital 1-589	HTTP Request	1793	✓	4704	134	1792	0
453	09:19:37.803	Hospital 1-899	HTTP Request	1428	✓	4729	134	1428	2
454	09:19:36.875	Hospital 1-242	HTTP Request	2356	✓	4729	134	2356	7
455	09:19:37.018	Hospital 1-364	HTTP Request	2225	✓	4729	134	2224	2
456	09:19:37.452	Hospital 1-609	HTTP Request	1792	✓	4729	134	1791	1
457	09:19:37.426	Hospital 1-586	HTTP Request	1820	✓	4729	134	1820	0
458	09:19:37.248	Hospital 1-562	HTTP Request	2006	✓	4729	134	2005	0
459	09:19:37.224	Hospital 1-540	HTTP Request	2031	✓	4729	134	2031	0
460	09:19:36.865	Hospital 1-208	HTTP Request	2390	✓	4729	134	2390	4
461	09:19:37.435	Hospital 1-594	HTTP Request	1820	✓	4729	134	1820	0
462	09:19:37.202	Hospital 1-522	HTTP Request	2054	✓	4729	134	2054	2
463	09:19:36.877	Hospital 1-245	HTTP Request	2383	✓	4729	134	2383	5
464	09:19:36.865	Hospital 1-234	HTTP Request	2397	✓	4729	134	2397	4
465	09:19:37.021	Hospital 1-366	HTTP Request	2249	✓	4729	134	2249	1
466	09:19:37.216	Hospital 1-533	HTTP Request	2054	✓	4729	134	2054	1
467	09:19:37.204	Hospital 1-523	HTTP Request	2060	✓	4729	134	2060	1

Summary: No. de Muestras 1000, Última Muestra 3801, Media 2172, Desviación 1159

Figura 36: Resultados vista en tabla
 Fuente: El autor
 Elaborado por: El autor

Se evidencia un aumento en el tiempo de proceso de la petición lo cual obedece únicamente a la sobrecarga del procesador del computador donde se encuentra el servidor web, mismo que aloja a los 8 servidores virtuales, y no es un servidor de alto rendimiento sino, un computador portátil de uso doméstico.

La figura 37, indica que la petición enviada por el usuario 936 se realizó en un tiempo de 3801 ms, latencia 3801, y 134 bytes enviados, siendo procesada a las 09h19:37:860

#	Tiempo de com...	Nombre del hilo	Etiqueta	Tiempo de Mue...	Estado	Bytes	Sent Bytes	Latency	Connect Time(...)
974	09:19:37.789	Hospital 1-773	HTTP Request	3809	✓	4729	134	3809	5
975	09:19:37.788	Hospital 1-780	HTTP Request	3811	✓	4729	134	3811	5
976	09:19:37.789	Hospital 1-775	HTTP Request	3812	✓	4729	134	3812	5
977	09:19:37.788	Hospital 1-764	HTTP Request	3816	✓	4729	134	3815	6
978	09:19:37.806	Hospital 1-897	HTTP Request	3805	✓	4729	134	3805	2
979	09:19:37.788	Hospital 1-781	HTTP Request	3823	✓	4729	134	3823	5
980	09:19:37.805	Hospital 1-702	HTTP Request	3806	✓	4729	134	3806	3
981	09:19:37.805	Hospital 1-705	HTTP Request	3809	✓	4729	134	3809	3
982	09:19:37.805	Hospital 1-710	HTTP Request	3810	✓	4729	134	3810	3
983	09:19:37.806	Hospital 1-901	HTTP Request	3812	✓	4729	134	3812	5
984	09:19:37.864	Hospital 1-940	HTTP Request	3763	✓	4736	134	3763	2
985	09:19:37.868	Hospital 1-950	HTTP Request	3759	✓	4729	134	3759	0
986	09:19:37.784	Hospital 1-817	HTTP Request	3843	✓	4729	134	3843	6
987	09:19:37.867	Hospital 1-953	HTTP Request	3770	✓	4729	134	3770	0
988	09:19:37.806	Hospital 1-697	HTTP Request	3831	✓	4729	134	3831	2
989	09:19:37.869	Hospital 1-955	HTTP Request	3776	✓	4729	134	3776	1
990	09:19:37.860	Hospital 1-947	HTTP Request	3787	✓	4729	134	3786	1
991	09:19:37.865	Hospital 1-951	HTTP Request	3783	✓	4729	134	3783	2
992	09:19:37.862	Hospital 1-949	HTTP Request	3786	✓	4729	134	3786	4
993	09:19:37.805	Hospital 1-708	HTTP Request	3845	✓	4729	134	3845	3
994	09:19:37.856	Hospital 1-944	HTTP Request	3795	✓	4729	134	3794	2
995	09:19:37.862	Hospital 1-945	HTTP Request	3789	✓	4729	134	3789	4
996	09:19:37.857	Hospital 1-935	HTTP Request	3794	✓	4729	134	3794	1
997	09:19:37.805	Hospital 1-703	HTTP Request	3846	✓	4729	134	3846	3
998	09:19:37.863	Hospital 1-939	HTTP Request	3789	✓	4729	134	3789	3
999	09:19:37.791	Hospital 1-760	HTTP Request	3861	✓	4729	134	3861	5
1000	09:19:37.860	Hospital 1-936	HTTP Request	3801	✓	4729	134	3801	1

Automatically? Child samples? No. de Muestras 1000 Última Muestra 3801 Media 2172 Desviación 1159

Figura 37: Resultados vista en tabla
 Fuente: El autor
 Elaborado por: El autor

3.2.2.4.1 Resultado gráfico

Nos presenta gráficamente los resultados de la prueba, en los cuales se puede observar el número de muestras o peticiones (1000 peticiones), tiempo de la última petición, una media de 2172, y un rendimiento de 11.836,654 peticiones procesadas por minuto, tal como se muestra en la figura 38.

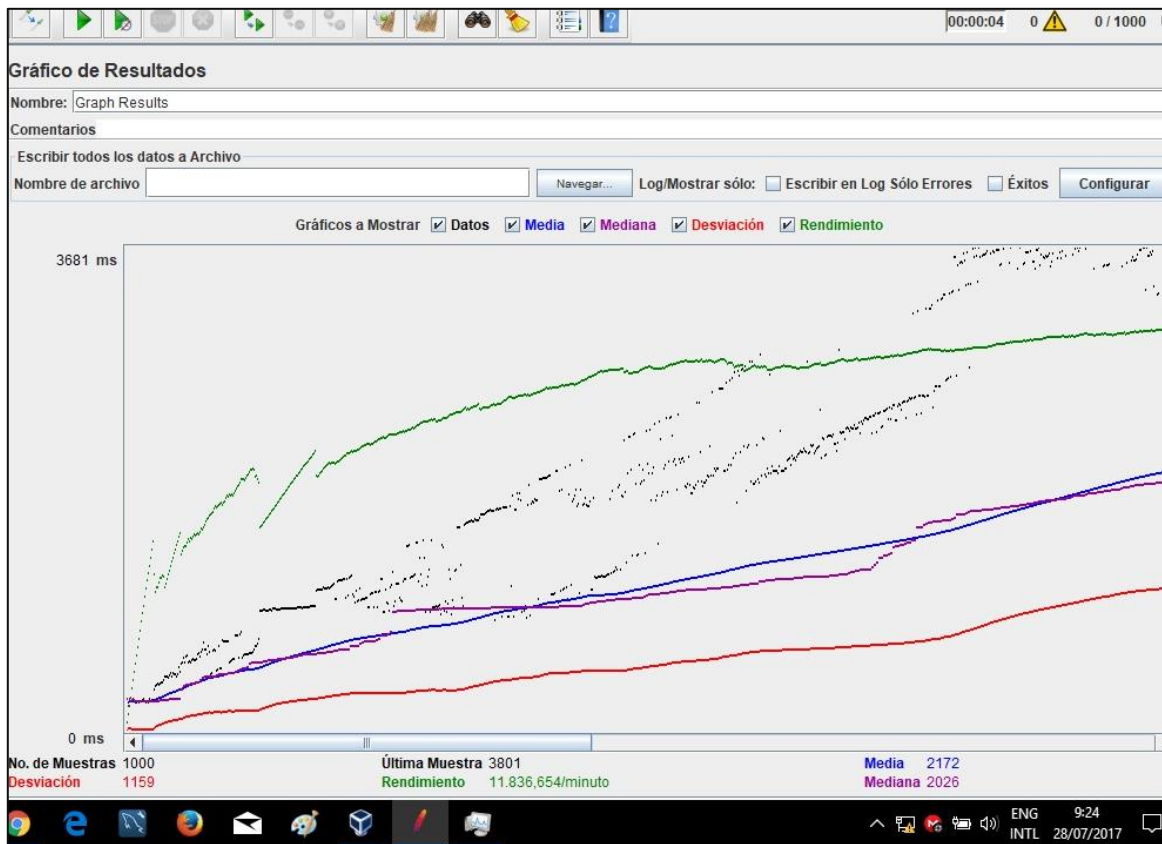


Figura 38: Resultados en gráfica
Fuente: El autor
Elaborado por: El autor

3.2.2.5 Revisión de las estadísticas de sesiones y balanceo de carga posterior a las pruebas con Jmeter

Una vez realizadas las pruebas de estrés a la aplicación y de balanceo de carga, a través de Jmeter. y obtenidos los resultados que nos entrega esta herramienta, se procede a verificar las estadísticas de peticiones y sesiones que ha recibido el Frontend de nuestra arquitectura tecnológica, y además de comprobar el balanceo de carga realizado por la herramienta Haproxy, tal como se aprecia en la figura 39.

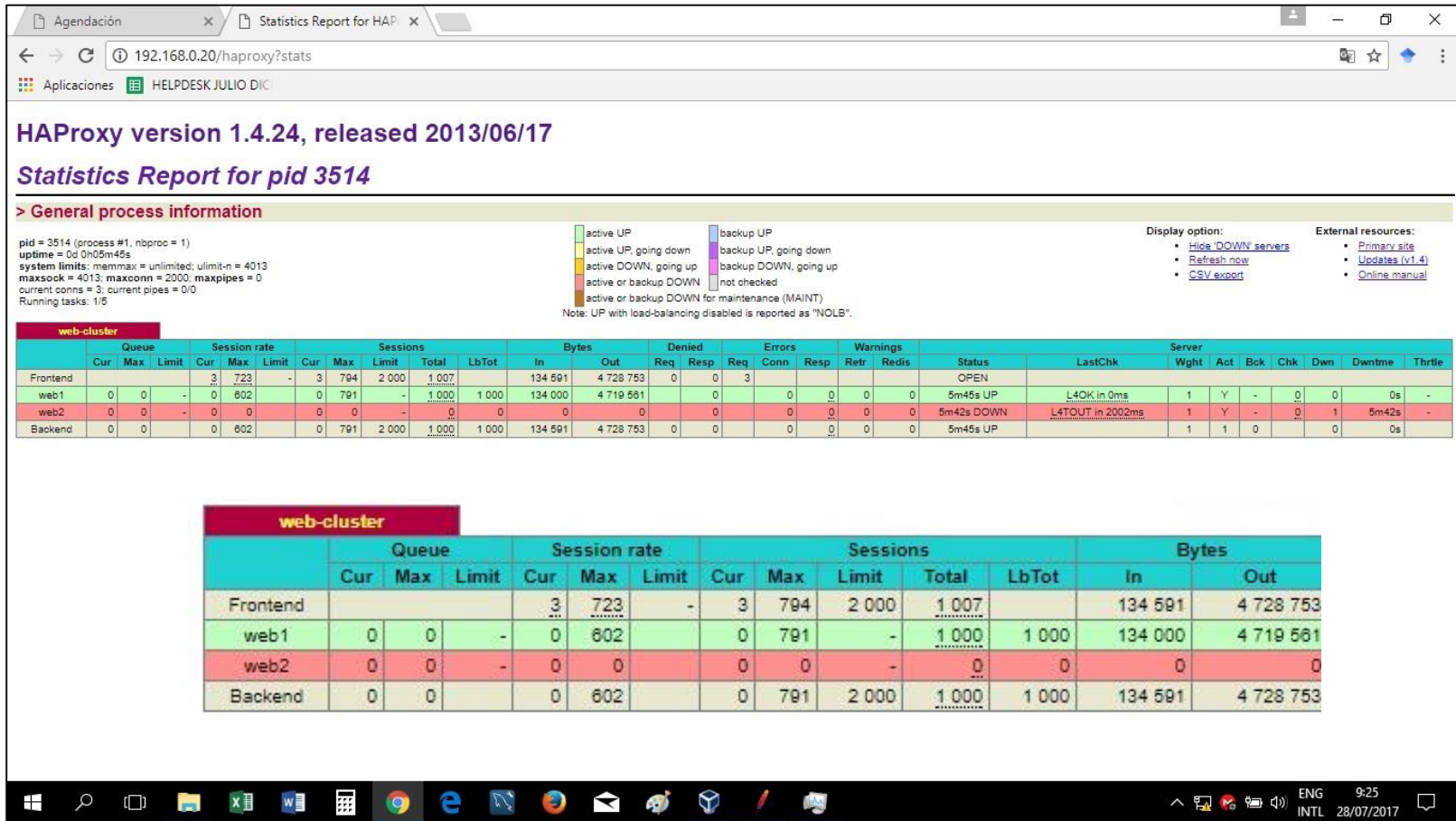


Figura 39: Reporte estadísticas Haproxy posterior a la prueba1
 Fuente: El autor
 Elaborado por: El autor

Podemos recordar que los valores en cuanto al números de sesiones, en las estadísticas previo a las pruebas realizadas con Jmeter, fueron:

Webserver1 = 0 peticiones

Webserver2 = 0 peticiones (**servidor apagado**)

Luego de las pruebas, según lo demostrado en la figura 39, se observa:

Webserver1 = 1000 peticiones

Webserver2 = 0 peticiones (**servidor apagado**)

De acuerdo a los resultados obtenidos en el reporte de estadísticas, se puede evidenciar que; enviadas las peticiones concurrentes de 1000 usuarios, hacia la url de la aplicación web “192.168.0.20/agendacion/medico/”, y en la que cada petición equivale a 1 sesión enviada hacia el servidor real Webserver1, éste servidor ha recibido la totalidad de la carga de las peticiones de los 1000 usuarios, ya que el servidor real Webserver2 se encuentra fuera de servicio.

3.2.3 Prueba 2 realizada con herramienta Jmeter con 2 servidores web

Esta prueba se realiza con 8 de los 8 servidores que componen nuestra granja de servidores, utilizando la herramienta para pruebas de estrés y balanceo de carga JMETER, a continuación, se detallan los servidores involucrados:

Tabla 11: Servidores disponibles en prueba2

NOMBRE	CLÚSTER WEB		CLÚSTER BASE DATOS	
	BALANCEADOR CARGA WEB	SERVIDOR REAL WEB	BALANCEADOR CARGA BASE DATOS	SERVIDOR REAL BASE DATOS
HaproxyWeb1	✓			
HaproxyWeb2	✓			
Webserver1		✓		
Webserver2		✓		
HaproxyBd1			✓	
HaproxyBd2			✓	
Bdserver1				✓
Bdserver2				✓

Fuente: El autor

Elaborado por: El autor

Esta prueba se ejecuta con los 2 servidores reales web (Webserver1 y Webserver2), para observar el rendimiento del clúster al equilibrar la carga de peticiones concurrentes de usuarios (1000 usuarios), entre los 2 Web servers. Luego se realizará la respectiva comparación de rendimiento y balanceo de carga entre ambas pruebas realizadas.

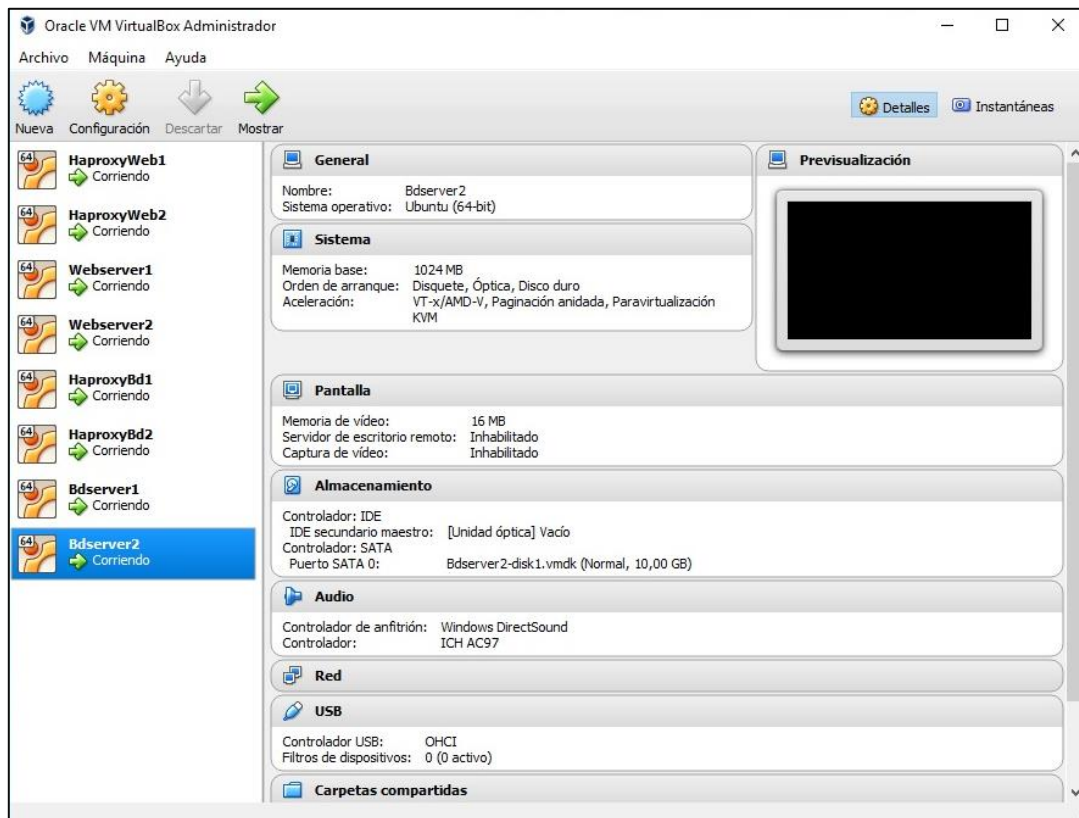


Figura 40: Clúster con todos los servidores ejecutándose

Fuente: El autor

Elaborado por: El autor

3.2.3.1 Revisión de estadísticas de sesiones y balanceo de carga previo a las pruebas

Se revisa el reporte de estadísticas de número de sesiones y balanceo de carga antes de realizar la prueba.

La dirección para obtener las estadísticas es:

<http://192.168.0.20/haproxy?stats>

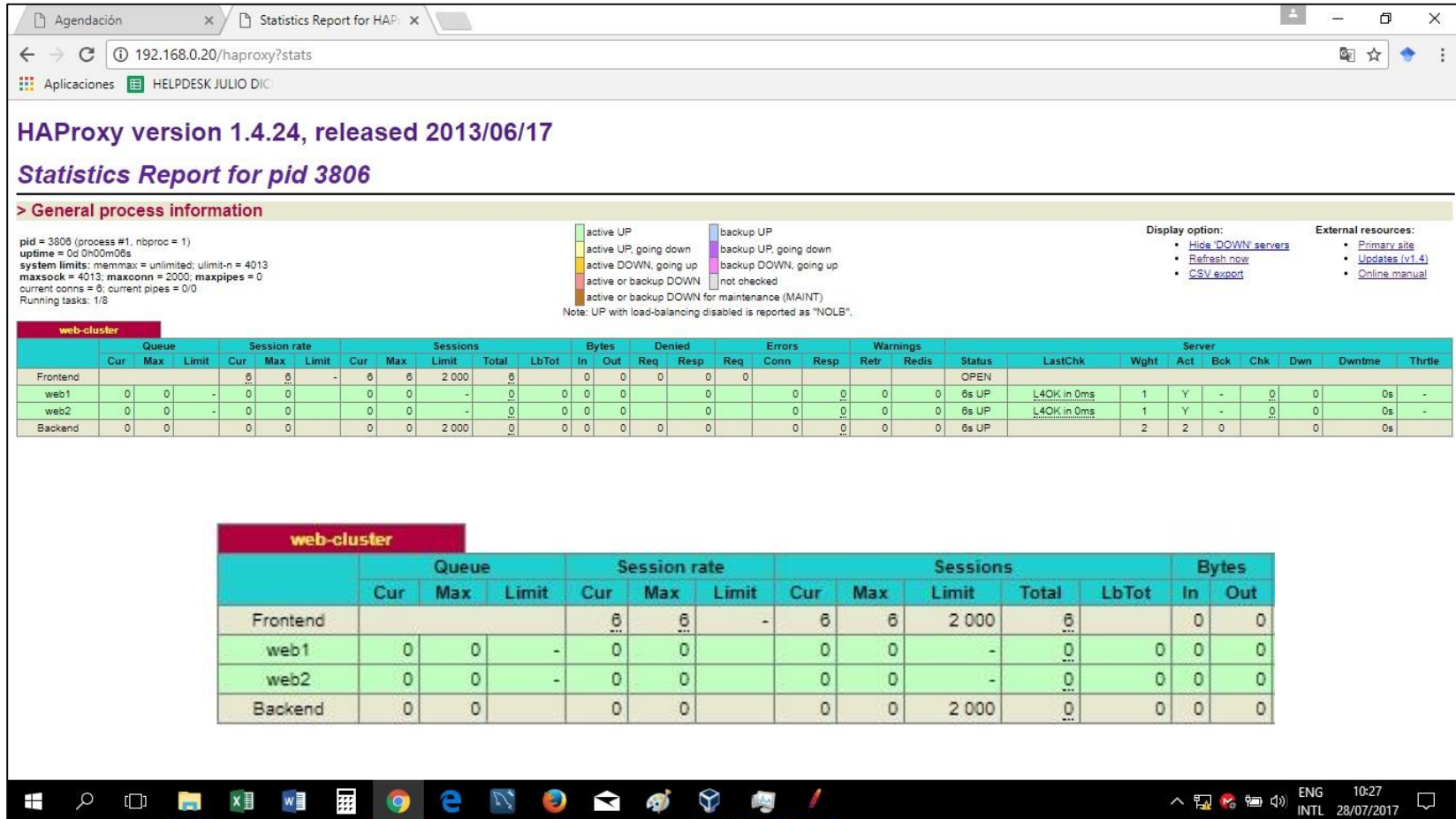


Figura 41: Reporte estadísticas Haproxy previo a la prueba2
 Fuente: El autor
 Elaborado por: El autor

3.2.3.2 Configuración de plan de pruebas con Jmeter

3.2.3.2.1 Plan de Pruebas

- **Grupo de usuarios:** se crea 1 Thread Group
- **Transacciones:** se genera 1 transacción para ingreso a la misma (192.168.0.20/agendacion/medico/).

3.2.3.2.2 Parámetros

- **Cantidad de usuarios:** 1000 usuarios
- **Duración de período de arribos:** 1 segundo
- **Cantidad de transacciones:** 1 transacción realizada por 1000 hilos o usuarios.
- **Protocolo:** Protocolo Http.

En el grupo de usuarios en Jmeter llamado Hospital, se establece la cantidad de usuarios (1000 usuarios) que harán las peticiones concurrentes a la app; el período de tiempo en el que realizará estas peticiones será de 1 segundo, haciendo esta operación 1 vez. Lo cual se aprecia en la figura a continuación:



The image shows the 'Thread Group' configuration window in JMeter. The window title is 'Grupo de Hilos'. The 'Nombre' field is set to 'Hospital'. The 'Comentarios' field is empty. The 'Acción a tomar después de un error de Muestreador' section has three radio buttons: 'Continuar' (selected), 'Comenzar siguiente iteración', and 'Parar'. The 'Propiedades de Hilo' section includes: 'Número de Hilos' set to 1000, 'Periodo de Subida (en segundos):' set to 1, and 'Contador del bucle:' with 'Sin fin' selected and a value of 1. There are two unchecked checkboxes: 'Retrasar la creación de Hilos hasta que se necesiten' and 'Planificador'.

Figura 42: Configuración grupo de usuarios (hilos)

Fuente: El autor

Elaborado por: El autor

A continuación, se crea una conexión Http, donde se configura la dirección ip, 192.168.0.20 que es la ip virtual para acceder al Frontend de la aplicación web.

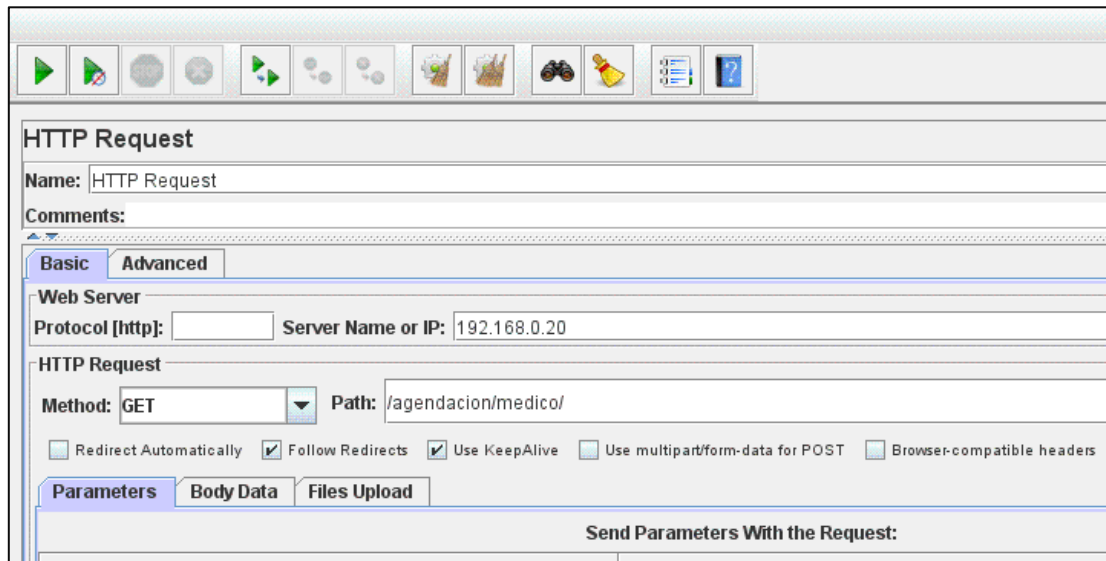


Figura 43: Configuración de conexión
Fuente: El autor
Elaborado por: El autor

Luego, se crean las vistas en las que se presentarán los resultados obtenidos luego de realizar la prueba de estrés a la aplicación.

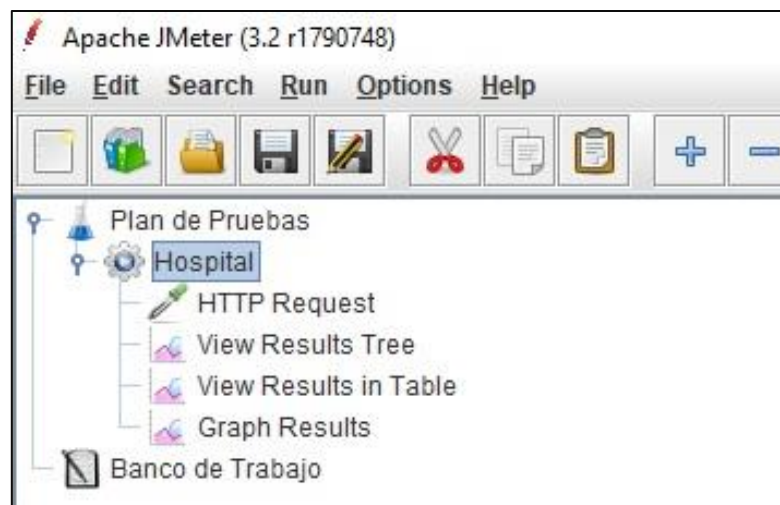


Figura 44: Creación de listener
Fuente: El autor
Elaborado por: El autor

3.2.3.3 Ejecución del plan de pruebas Jmeter

Una vez configurado el plan de pruebas en la herramienta Jmeter, se procede a la ejecución del mismo, haciendo clic en el ícono verde Start.

Como se muestra a continuación en la figura 45.

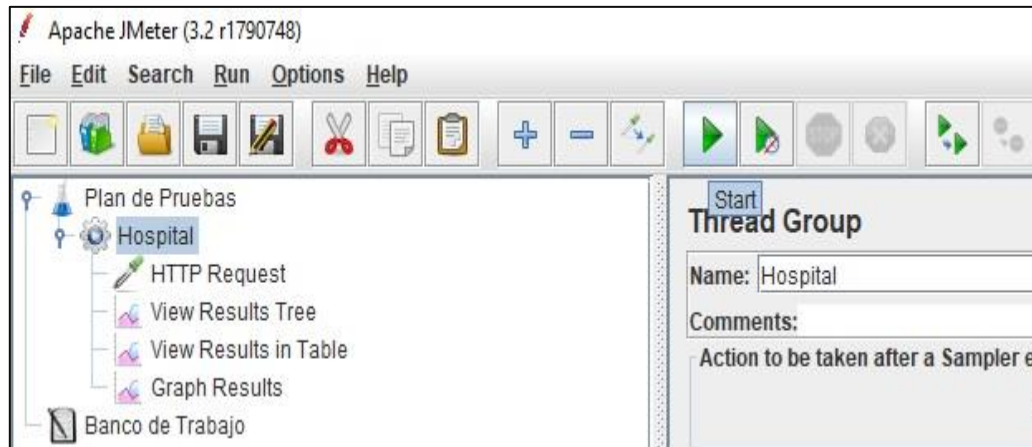


Figura 45: Ejecución de prueba en Jmeter

Fuente: El autor

Elaborado por: El autor

Recordando que la prueba consiste en la ejecución de peticiones concurrentes de 1000 usuarios durante 1 segundos

3.2.3.4 Resultados del plan de pruebas con Jmeter

Una vez ejecutada la prueba, Jmeter presenta los resultados a través de los 3 listener que se configuró al inicio (View Results Tree, View Results in Table, Graph Results)

3.2.3.4.1 Resultados en view results tree

En esta vista de resultados, se muestra de manera individual a manera de árbol, las peticiones realizadas, cada una con el detalle de la solicitud de acceso a la aplicación web, como, por ejemplo: Número de petición, inicio petición, tiempo de carga, latencia, tamaño de la petición, errores, etc.

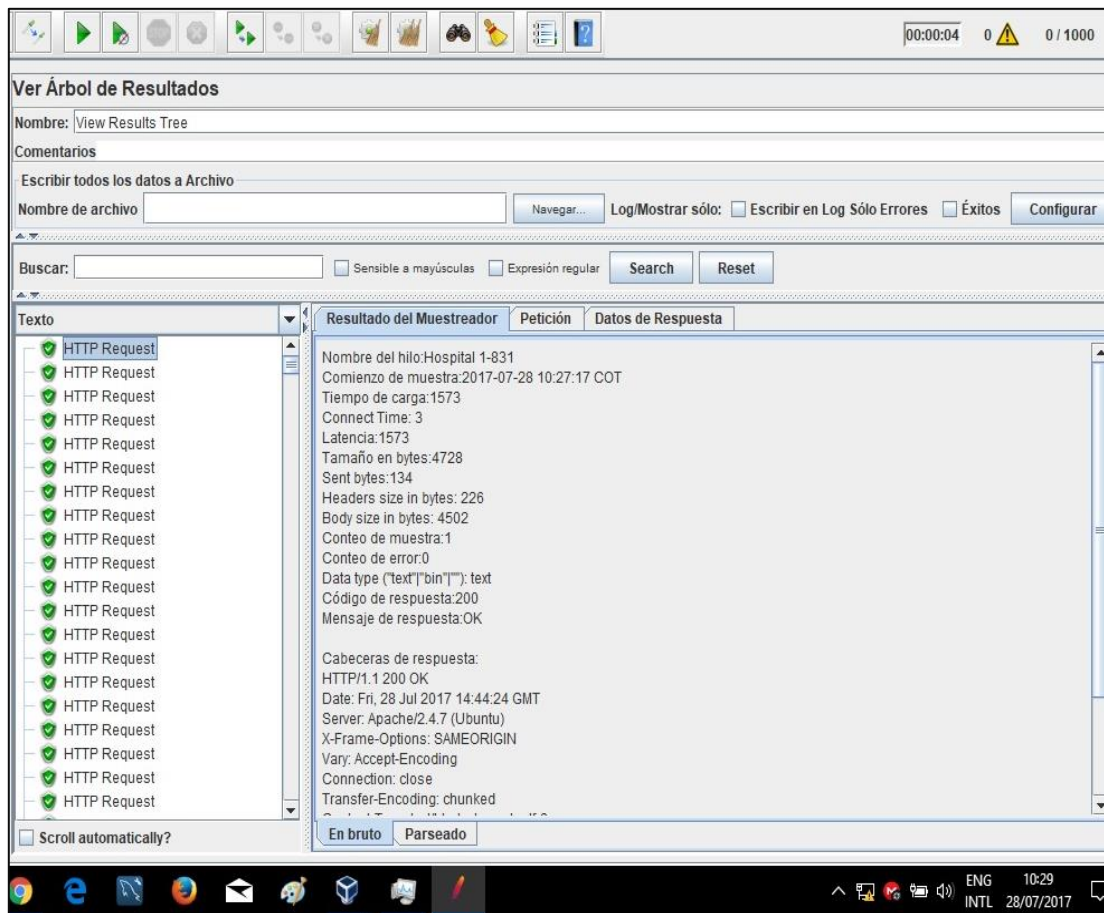


Figura 46: Resultados vista en árbol
Fuente: El autor
Elaborado por: El autor

La figura 46, evidencia que la primera petición en ingresar a la arquitectura tecnológica fue recibida correctamente sin errores a las 10h27:17, tiene un tamaño de 4728 bytes, latencia de 1573, bytes enviados 134.

Luego en la figura 47, se aprecia que la petición 694, tiene estado de respuesta correcto a las 10h27:17, tamaño en bytes 4722, latencia 2344 y 134 bytes enviados.

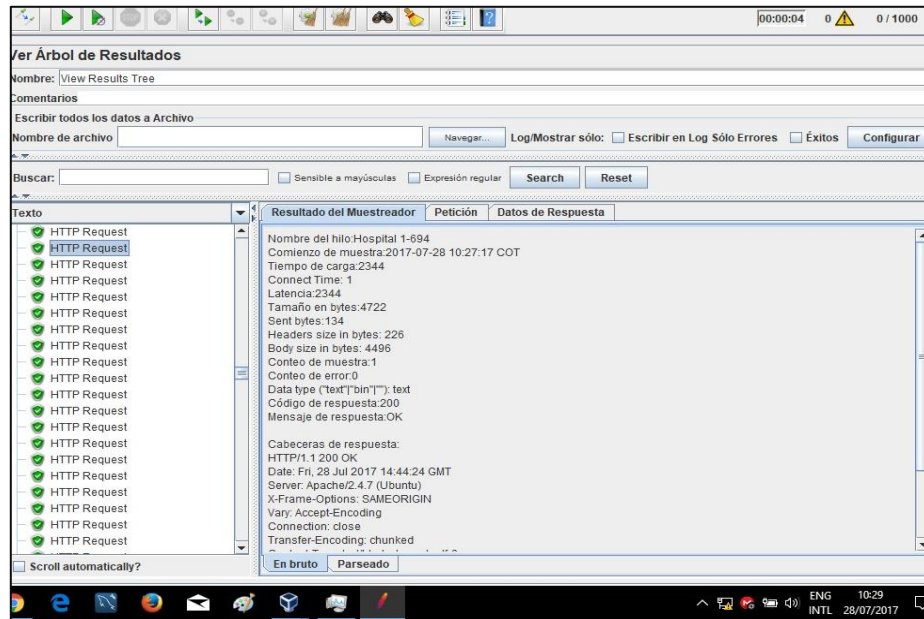


Figura 47: Resultados vista en árbol
 Fuente: El autor
 Elaborado por: El autor

Por último, la petición 975, tiene estado de respuesta correcto a las 10h27:17, tamaño en bytes 4722, latencia 3101 y 134 bytes enviados, lo cual se observa en la figura 48.

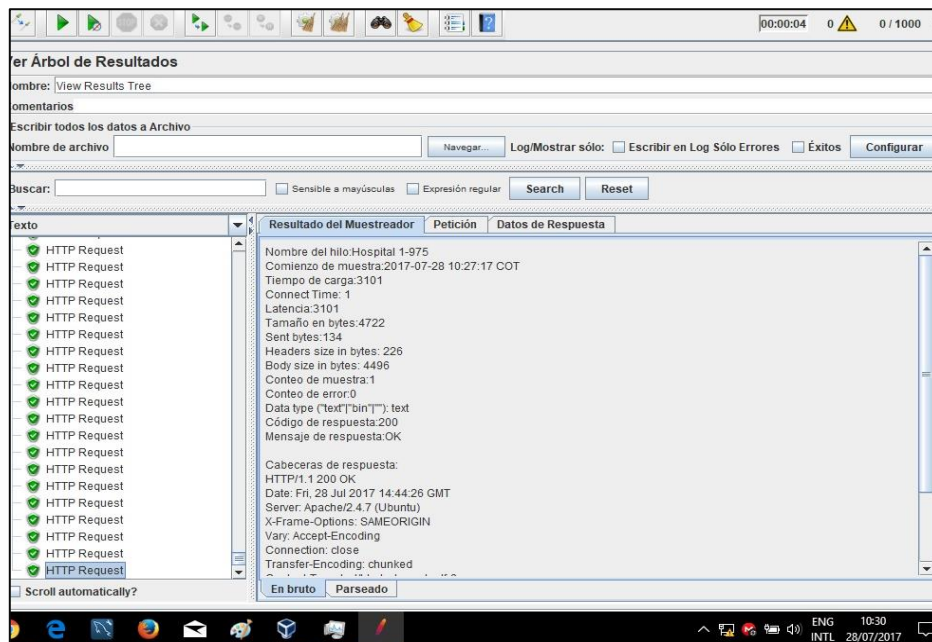


Figura 48: Resultados vista en árbol
 Fuente: El autor
 Elaborado por: El autor

3.2.3.4.2 Resultados en view results in table

Se muestran los resultados en forma grupal en una tabla, detallando el estado de cada una de las peticiones realizadas por los 1000 usuarios.

Se evidencia en el reporte que las 1000 peticiones fueron procesadas completas, sin ningún error, en un tiempo total de 4 segundos.

Muestra #	Tiempo de com...	Nombre del hilo	Etiqueta	Tiempo de Mue...	Estado	Bytes	Sent Bytes	Latency	Connect Time...
1	10:27:16.320	Hospital 1-1	HTTP Request	36	✓	4730	134	36	1
2	10:27:16.323	Hospital 1-3	HTTP Request	154	✓	4723	134	154	1
3	10:27:16.343	Hospital 1-20	HTTP Request	226	✓	4704	134	226	1
4	10:27:16.345	Hospital 1-22	HTTP Request	228	✓	4728	134	228	1
5	10:27:16.330	Hospital 1-9	HTTP Request	260	✓	4729	134	260	2
6	10:27:16.351	Hospital 1-27	HTTP Request	245	✓	4728	134	244	0
7	10:27:16.325	Hospital 1-5	HTTP Request	277	✓	4735	134	277	2
8	10:27:16.339	Hospital 1-17	HTTP Request	274	✓	4728	134	274	1
9	10:27:16.371	Hospital 1-31	HTTP Request	232	✓	4704	134	232	0
10	10:27:16.367	Hospital 1-40	HTTP Request	247	✓	4704	134	246	2
11	10:27:16.374	Hospital 1-34	HTTP Request	241	✓	4704	134	241	1
12	10:27:16.329	Hospital 1-8	HTTP Request	291	✓	4728	134	291	1
13	10:27:16.337	Hospital 1-15	HTTP Request	286	✓	4735	134	286	1
14	10:27:16.326	Hospital 1-6	HTTP Request	317	✓	4723	134	317	2
15	10:27:16.335	Hospital 1-13	HTTP Request	321	✓	4728	134	320	1
16	10:27:16.331	Hospital 1-10	HTTP Request	326	✓	4730	134	326	1
17	10:27:16.415	Hospital 1-70	HTTP Request	245	✓	4703	134	245	1
18	10:27:16.340	Hospital 1-18	HTTP Request	326	✓	4723	134	326	1
19	10:27:16.338	Hospital 1-16	HTTP Request	332	✓	4723	134	331	1
20	10:27:16.396	Hospital 1-51	HTTP Request	279	✓	4729	134	279	1
21	10:27:16.396	Hospital 1-66	HTTP Request	291	✓	4730	134	291	1
22	10:27:16.333	Hospital 1-12	HTTP Request	355	✓	4723	134	355	1
23	10:27:16.359	Hospital 1-33	HTTP Request	335	✓	4722	134	335	1
24	10:27:16.344	Hospital 1-21	HTTP Request	350	✓	4728	134	350	3
25	10:27:16.332	Hospital 1-11	HTTP Request	370	✓	4735	134	370	1
26	10:27:16.348	Hospital 1-25	HTTP Request	355	✓	4728	134	355	1
27	10:27:16.369	Hospital 1-30	HTTP Request	340	✓	4729	134	340	1

Poll automatically? Child samples? **No. de Muestras 1000** **Última Muestra 3101** **Media 1834** **Desviación 756**

Figura 49: Resultados vista en tabla

Fuente: El autor

Elaborado por: El autor

Según la figura 49, se observa que la primera petición enviada por el primer usuario se realizó básicamente en un tiempo de 36 ms, latencia 36 y bytes enviados 134, siendo procesada a las 10h27:36:620

La figura 50, indica que la petición enviada por el usuario 500 se realizó en un tiempo de 1520 ms, latencia 1520, y 134 bytes enviados, siendo procesada a las 10h27:17:367

Muestra #	Tiempo de com...	Nombre del hilo	Etiqueta	Tiempo de Mue...	Estado	Bytes	Sent Bytes	Latency	Connect Time(...)
499	10:27:17.280	Hospital 1-807	HTTP Request	1530	✓	4728	134	1530	1
500	10:27:17.367	Hospital 1-874	HTTP Request	1520	✓	4728	134	1520	1
501	10:27:17.314	Hospital 1-831	HTTP Request	1573	✓	4728	134	1573	3
502	10:27:17.193	Hospital 1-734	HTTP Request	1694	✓	4729	134	1694	1
503	10:27:16.918	Hospital 1-507	HTTP Request	1969	✓	4704	134	1969	1
504	10:27:17.204	Hospital 1-742	HTTP Request	1683	✓	4729	134	1683	1
505	10:27:17.197	Hospital 1-732	HTTP Request	1691	✓	4723	134	1691	1
506	10:27:17.202	Hospital 1-743	HTTP Request	1686	✓	4730	134	1686	1
507	10:27:17.222	Hospital 1-759	HTTP Request	1666	✓	4729	134	1666	1
508	10:27:17.170	Hospital 1-716	HTTP Request	1719	✓	4704	134	1719	1
509	10:27:17.083	Hospital 1-642	HTTP Request	1806	✓	4723	134	1806	4
510	10:27:17.366	Hospital 1-873	HTTP Request	1523	✓	4728	134	1523	1
511	10:27:17.267	Hospital 1-795	HTTP Request	1622	✓	4728	134	1622	1
512	10:27:17.035	Hospital 1-597	HTTP Request	1854	✓	4728	134	1854	1
513	10:27:17.048	Hospital 1-541	HTTP Request	1841	✓	4729	134	1841	1
514	10:27:16.890	Hospital 1-481	HTTP Request	2000	✓	4704	134	1999	2
515	10:27:17.466	Hospital 1-954	HTTP Request	1424	✓	4703	134	1423	1
516	10:27:17.467	Hospital 1-956	HTTP Request	1423	✓	4703	134	1422	1
517	10:27:16.945	Hospital 1-529	HTTP Request	1945	✓	4704	134	1944	104
518	10:27:17.048	Hospital 1-535	HTTP Request	1842	✓	4704	134	1841	2
519	10:27:17.047	Hospital 1-542	HTTP Request	1843	✓	4729	134	1842	2
520	10:27:17.234	Hospital 1-769	HTTP Request	1656	✓	4735	134	1655	2
521	10:27:17.293	Hospital 1-817	HTTP Request	1597	✓	4722	134	1597	2
522	10:27:17.459	Hospital 1-947	HTTP Request	1447	✓	4735	134	1447	1
523	10:27:16.789	Hospital 1-399	HTTP Request	2115	✓	4728	134	2115	3
524	10:27:17.430	Hospital 1-921	HTTP Request	1467	✓	4728	134	1467	1
525	10:27:17.350	Hospital 1-861	HTTP Request	1544	✓	4728	134	1544	1

Auto-scan automatically? Child samples? **No. de Muestras 1000** **Última Muestra 3101** **Media 1834** **Desviación 756**

Figura 50: Resultados vista en tabla

Fuente: El autor

Elaborado por: El autor

La figura 51, indica que la petición enviada por el usuario 1000 se realizó en un tiempo de 3101 ms, latencia 3101, y 134 bytes enviados. Siendo procesada a las 10h27:17:487.

Sample #	Time of com...	Nombre del hilo	Etiqueta	Timeo de Mue...	Estado	Bytes	Sent Bytes	Latency	Connect Time(...)
974	10:27:17.478	Hospital 1-900	HTTP Request	3055	✓	4729	134	3055	1
975	10:27:17.474	Hospital 1-962	HTTP Request	3059	✓	4729	134	3059	2
976	10:27:17.487	Hospital 1-974	HTTP Request	3049	✓	4729	134	3049	1
977	10:27:17.506	Hospital 1-991	HTTP Request	3032	✓	4729	134	3032	0
978	10:27:17.485	Hospital 1-973	HTTP Request	3053	✓	4729	134	3053	0
979	10:27:17.510	Hospital 1-993	HTTP Request	3028	✓	4730	134	3028	2
980	10:27:17.475	Hospital 1-965	HTTP Request	3064	✓	4729	134	3064	1
981	10:27:17.409	Hospital 1-909	HTTP Request	3139	✓	4728	134	3139	0
982	10:27:17.514	Hospital 1-997	HTTP Request	3036	✓	4736	134	3035	1
983	10:27:17.475	Hospital 1-963	HTTP Request	3077	✓	4728	134	3077	1
984	10:27:17.400	Hospital 1-901	HTTP Request	3152	✓	4728	134	3152	1
985	10:27:17.453	Hospital 1-938	HTTP Request	3109	✓	4735	134	3109	1
986	10:27:17.477	Hospital 1-967	HTTP Request	3085	✓	4728	134	3085	1
987	10:27:17.416	Hospital 1-913	HTTP Request	3151	✓	4722	134	3151	3
988	10:27:17.415	Hospital 1-914	HTTP Request	3152	✓	4722	134	3152	2
989	10:27:17.421	Hospital 1-918	HTTP Request	3147	✓	4729	134	3147	2
990	10:27:17.409	Hospital 1-910	HTTP Request	3160	✓	4735	134	3160	1
991	10:27:17.407	Hospital 1-906	HTTP Request	3162	✓	4728	134	3162	1
992	10:27:17.466	Hospital 1-957	HTTP Request	3104	✓	4729	134	3103	1
993	10:27:17.403	Hospital 1-893	HTTP Request	3173	✓	4722	134	3173	1
994	10:27:17.417	Hospital 1-915	HTTP Request	3159	✓	4728	134	3159	1
995	10:27:17.479	Hospital 1-968	HTTP Request	3104	✓	4728	134	3104	1
996	10:27:17.490	Hospital 1-978	HTTP Request	3093	✓	4735	134	3093	1
997	10:27:17.484	Hospital 1-971	HTTP Request	3099	✓	4735	134	3099	1
998	10:27:17.489	Hospital 1-976	HTTP Request	3097	✓	4728	134	3096	1
999	10:27:17.504	Hospital 1-989	HTTP Request	3084	✓	4728	134	3084	0
1000	10:27:17.487	Hospital 1-975	HTTP Request	3101	✓	4722	134	3101	1

Summary: No. de Muestras 1000, Última Muestra 3101, Media 1834, Desviación 756

Figura 51: Resultados vista en tabla
Fuente: El autor
Elaborado por: El autor

3.2.3.4.1 Resultado gráfico

Nos presenta gráficamente los resultados de la prueba, en los cuales se puede observar el número de muestras o peticiones (1000 peticiones), tiempo de la última petición 3101 ms, una media de 1834 ms, y un rendimiento de 14.058,107 peticiones procesadas por minuto, tal como se muestra en la figura 52.

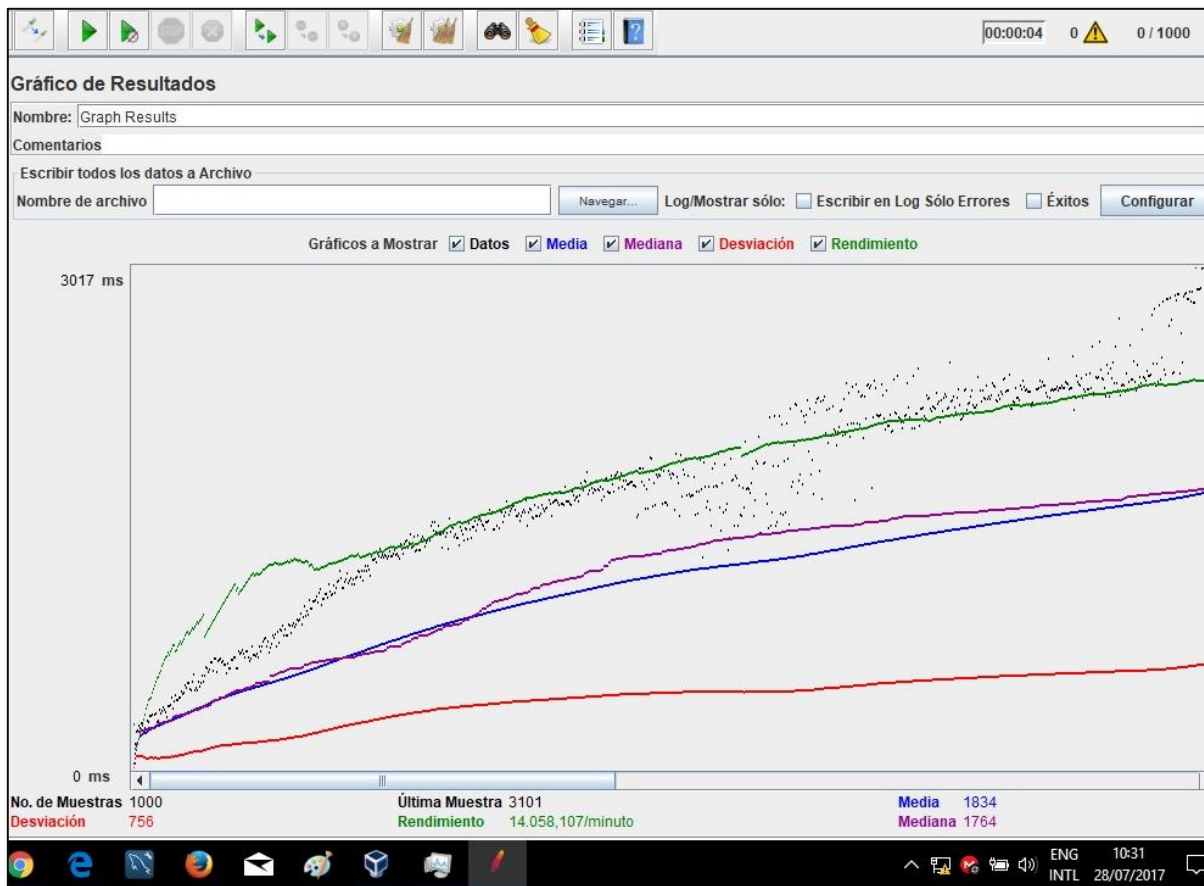


Figura 52: Resultado gráfico

Fuente: El autor

Elaborado por: El autor

3.2.3.5 Revisión de las estadísticas de sesiones y balanceo de carga posterior a las pruebas con Jmeter

Una vez realizadas las pruebas de estrés a la aplicación y de balanceo de carga, y obtenidos los resultados que nos entrega esta herramienta, se procede a verificar las estadísticas de peticiones y sesiones que ha recibido el Frontend de nuestra arquitectura tecnológica, y además de comprobar el balanceo de carga realizado por la herramienta Haproxy.

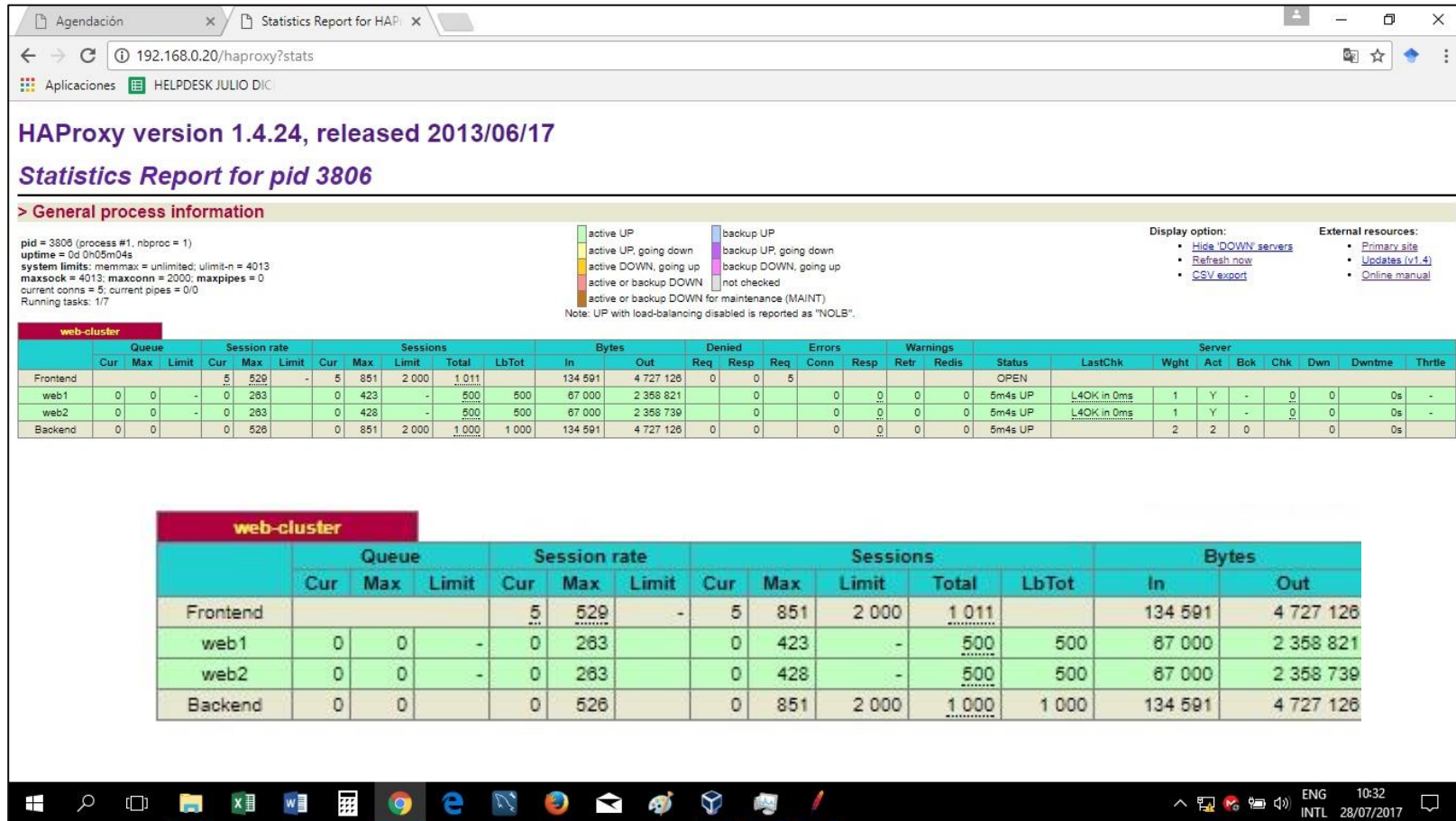


Figura 53: Reporte estadísticas Haproxy posterior a la prueba2
 Fuente: El autor
 Elaborado por: El autor

Se recuerda que los valores en cuanto al números de sesiones, en las estadísticas previo a las pruebas realizadas con Jmeter, fueron:

Webserver1 = 0 peticiones

Webserver2 = 0 peticiones

Luego de las pruebas, según lo demostrado en la figura 53, se observa:

Webserver1 = 500 peticiones

Webserver2 = 500 peticiones

De acuerdo a los resultados obtenidos en el reporte de estadísticas, se puede evidenciar que; enviadas las peticiones concurrentes de 1000 usuarios, hacia la url de la aplicación web “192.168.0.20/agendacion/medico/”, estas peticiones se han repartido equitativamente entre los servidores reales Webserver1 y Webserver2, 500 peticiones para cada uno, tal como lo indica el algoritmo utilizado Round Robin, y se refleja en el reporte de estadísticas Haproxy en la figura 53.

3.2.4 Análisis de resultados entre las 2 pruebas con Jmeter

Luego de haber realizado 2 pruebas de carga y rendimiento utilizando la herramienta Jmeter de Apache, la primera de ellas con 1 solo servidor real web, y la segunda con los 2 servidores reales web, es pertinente realizar el respectivo análisis de ambas pruebas para determinar el rendimiento y efectividad de la arquitectura tecnológica que está presentando como solución a la problemática.

Para esto, se elabora la siguiente tabla comparativa, en la cual se observan datos como la media, peticiones recibidas, rendimiento (peticiones/tiempo) y desviación, entre ambas pruebas, a continuación, se detalla:

Tabla 12: Comparación de resultados de prueba1 y prueba2

	PRUEBA JMETER CON 1 SERVIDOR WEB	PRUEBA JMETER CON 2 SERVIDORES WEB	
Peticiones recibidas	Webserver1	Webserver1	Webserver2
	1000	500	500
Media (tiempo)	2172 ms	1834 ms	
Rendimiento (peticiones/tiempo)	11.836,654/minuto	14.058,107/minuto	
Desviación	1159	756	

Fuente: El autor

Elaborado por: El autor

De acuerdo a la comparación realizada en la tabla 12, en el análisis se concluye lo siguiente:

- **Alta disponibilidad**, en la prueba 1 se evidencia que la arquitectura tecnológica posee alta disponibilidad en el servicio, ya que ante la caída del servidor Webserver2, el servicio sigue disponible con el servidor Webserver1, obviamente el rendimiento disminuye, pero no se sufre pérdida del servicio.
- **Balanceo de carga**, Se evidencia en la prueba 2, el correcto funcionamiento del balanceo de carga entre ambos servidores web (Webserver1 y Webserver2) y un aumento considerable en el rendimiento (peticiones/tiempo), en comparación a la prueba 1 (solo con Webserver1).
- **Rendimiento y escalabilidad**, queda demostrado que la arquitectura tecnológica está apta para el aumento de más servidores, y mientras se va aumentando servidores, esto se ve reflejado en el aumento del rendimiento en las transacciones y peticiones que recibe.
- **Persistencia de sesiones**, en la prueba 1, ante la caída del servidor Webserver2, la aplicación ejecutó el código de almacenamiento y traspaso de sesiones hacia el servidor disponible, en este caso el Webserver1.

3.2.5 Prueba 3 realizada con javascript desde la consola del navegador con 1 servidor web

Esta prueba consiste en ejecutar un código javascript dentro de la consola del navegador web, concretamente se ejecuta un for, el cual hará un bucle de 1000 repeticiones hacia, realizando 1000 peticiones hacia la arquitectura tecnológica.

Esta primera prueba con javascript, se realizará con un solo servidor real web funcionando, ya que el otro servidor web, se encuentra apagado. De esta manera se podrá evidenciar el rendimiento del clúster cuando tiene fuera de servicio a uno de los dos servidores web.

3.2.5.1 Revisión de estadísticas de sesiones y balanceo de carga previo a las pruebas

Se revisa el reporte de estadísticas de número de sesiones y balanceo de carga antes de realizar la prueba.

Las estadísticas de los servidores balanceadores de carga HaproxyWeb1 y HaproxyWeb2, son reiniciadas, por lo tanto se presentarán en cero.

La dirección para obtener las estadísticas es:

<http://192.168.0.20/haproxy?stats>

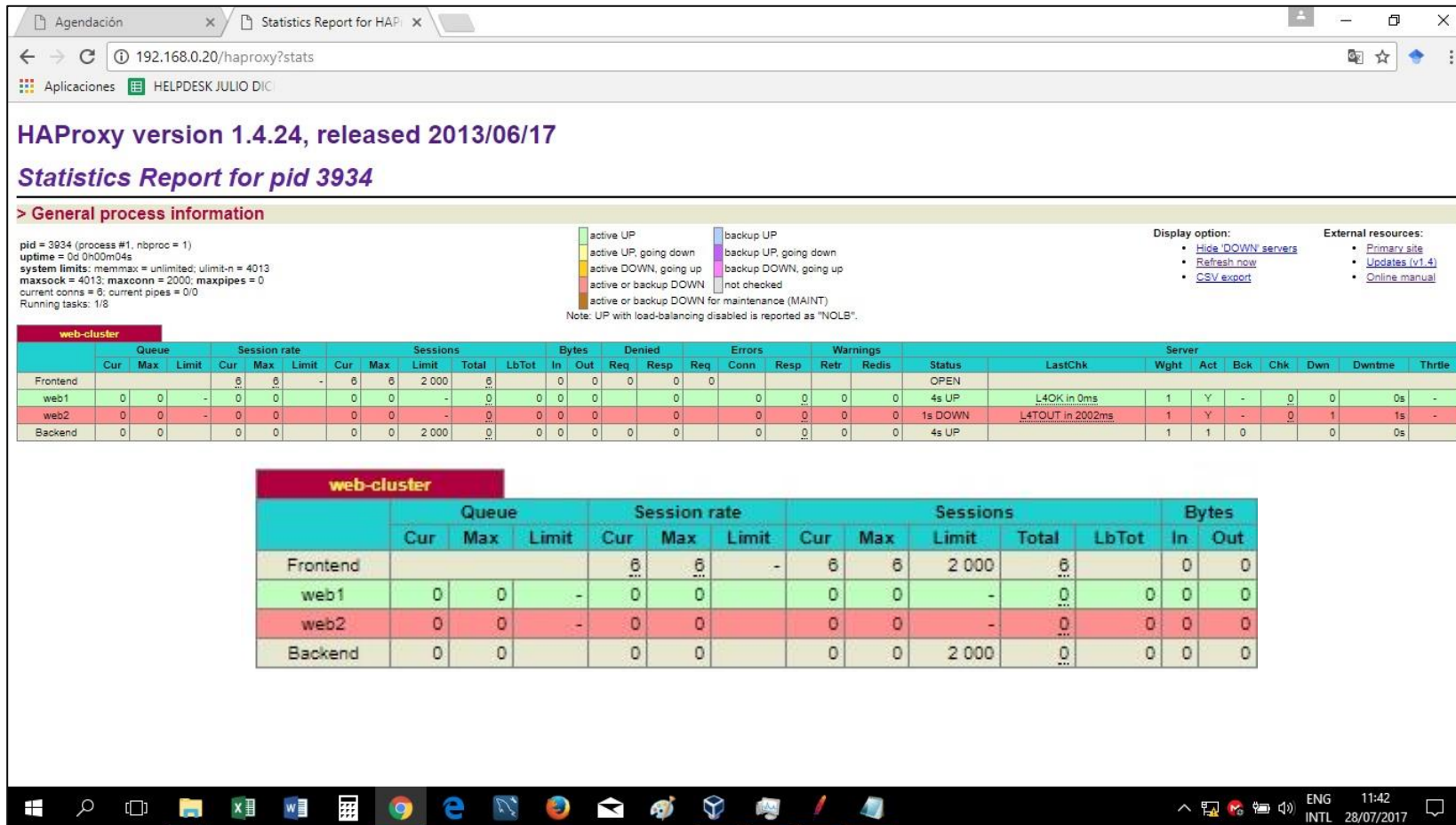


Figura 54: Reporte estadísticas haproxy previo a la prueba3

Fuente: El autor

Elaborado por: El autor

3.2.5.2 Código javascript utilizado en el navegador

Consiste básicamente en un For el cual realizará 1000 peticiones, pero a diferencia de las pruebas realizadas con la herramienta JMETER donde las peticiones las realizan con 1000 usuarios diferentes, con el For las 1000 peticiones las realizará un único usuario.

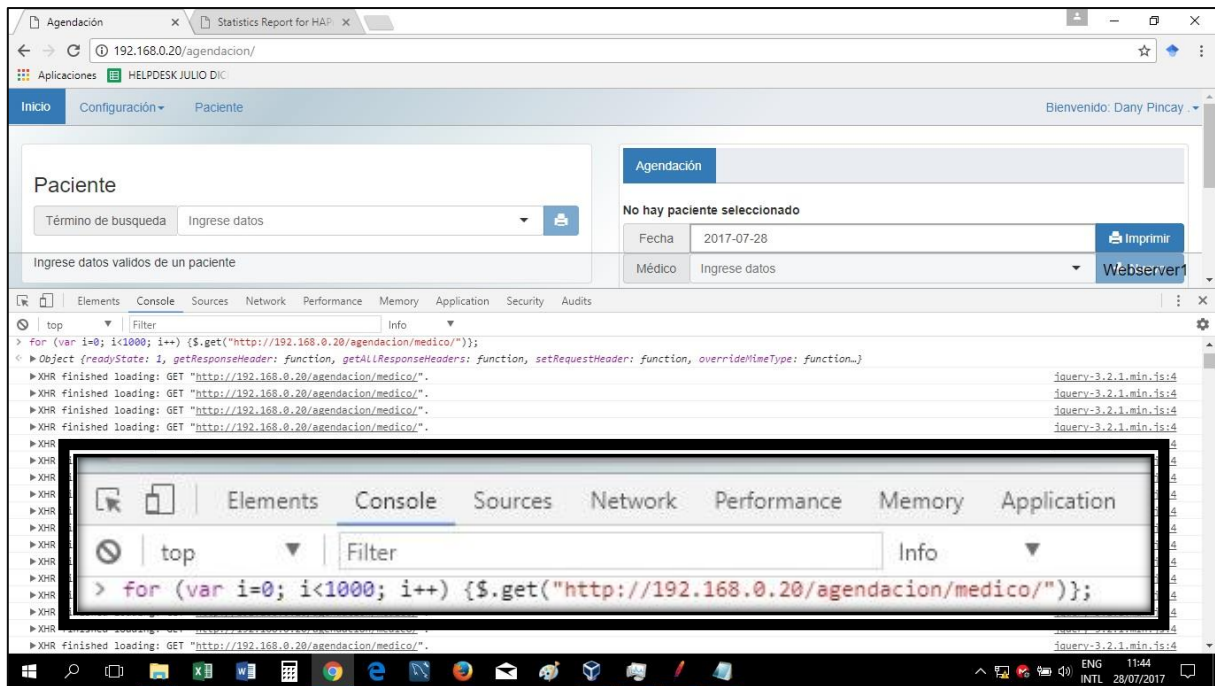


Figura 55: Código JavaScript para la prueba3

Fuente: El autor

Elaborado por: El autor

3.2.5.3 Resultados de la prueba

Una vez ejecutada la prueba, se puede apreciar los resultados obtenidos en número tiempo y peso de cada petición

En la figura 56, se observa que la primera petición tiene como datos: tiempo 18 ms y tamaño 4.6kb.

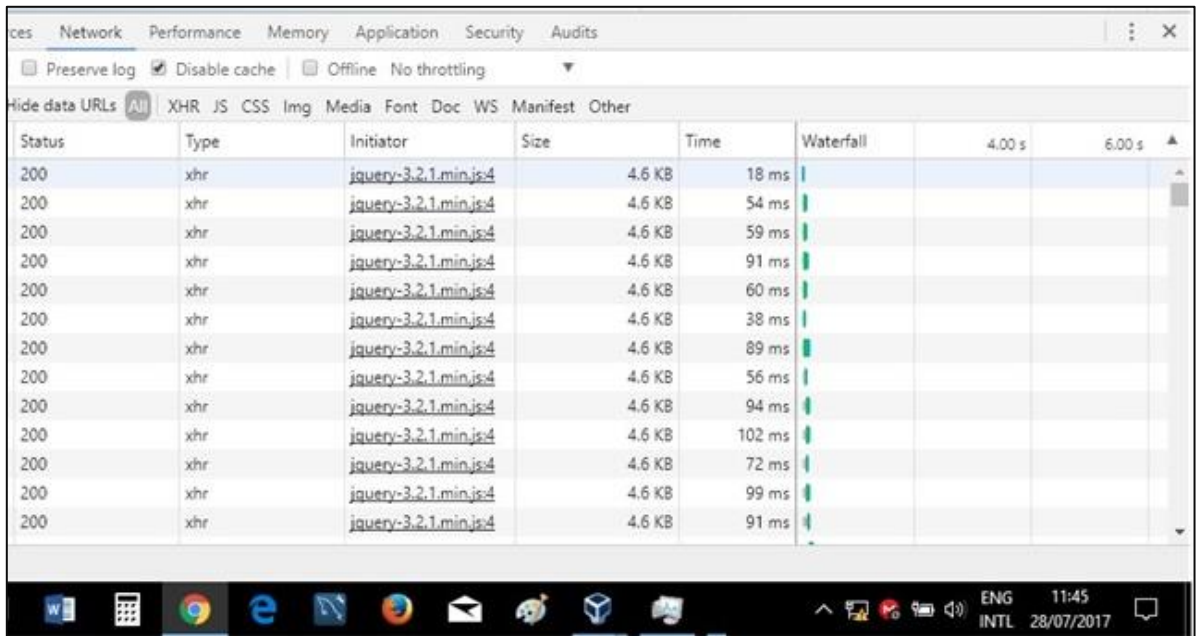


Figura 56: Resultados prueba3
 Fuente: El autor
 Elaborado por: El autor

En la figura 57, se observa que las peticiones a mitad de la tabla (aproximadamente petición 500) tiene como datos: tiempo 2.34 ms y tamaño 4.6kb.

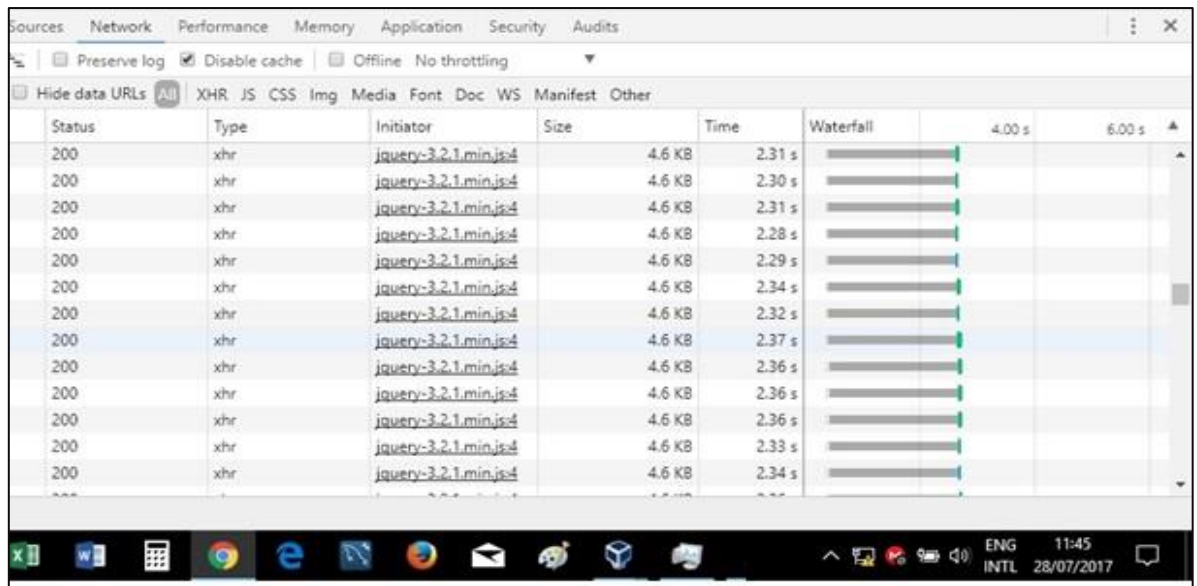


Figura 57: Resultados prueba3
 Fuente: El autor
 Elaborado por: El autor

Mientras que la figura 58, indica que la última petición tiene como datos: tiempo 5.72 ms y tamaño 4.6 Kb.

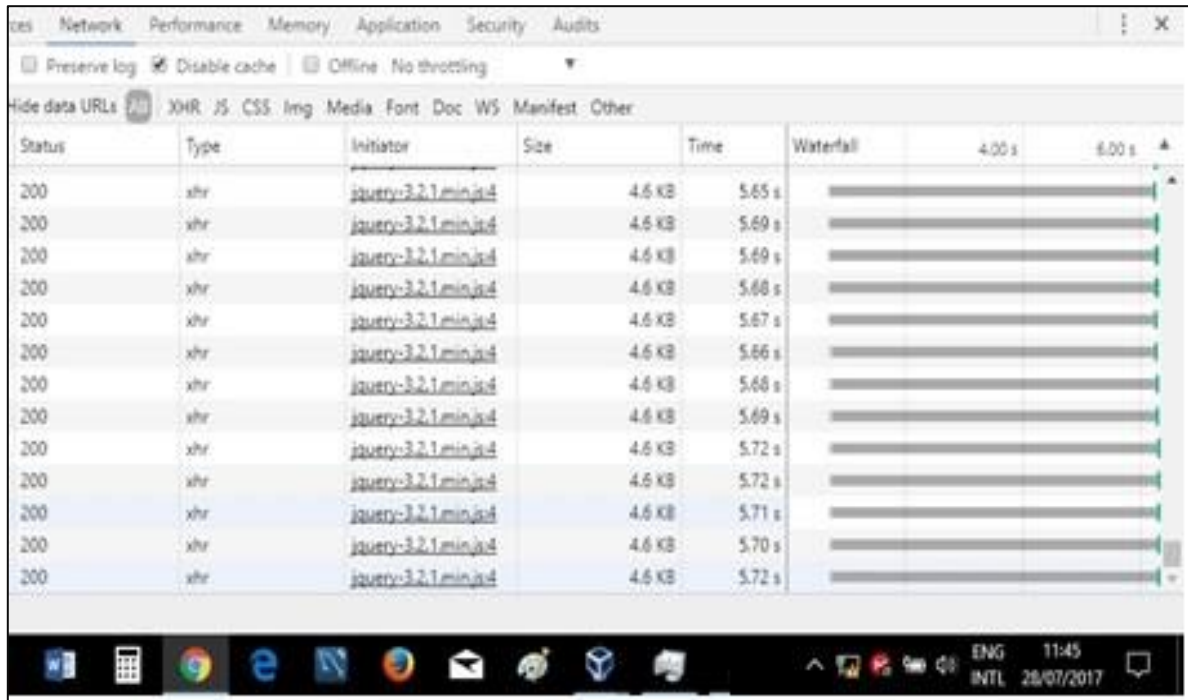


Figura 58: Resultados prueba3
Fuente: El autor
Elaborado por: El autor

3.2.5.4 Revisión de las estadísticas de sesiones y balanceo de carga posterior a las pruebas con javascript

Una vez realizadas las pruebas de estrés a la aplicación y de balanceo de carga, y obtenidos los resultados que nos entrega esta herramienta, se procede a verificar las estadísticas de peticiones y sesiones que ha recibido el Frontend de nuestra arquitectura tecnológica, y además de comprobar el balanceo de carga realizado por la herramienta Haproxy.

Agendación x Statistics Report for HAP x

192.168.0.20/haproxy?stats

Aplicaciones HELPDESK JULIO DIC

HAProxy version 1.4.24, released 2013/06/17

Statistics Report for pid 3934

> General process information

pid = 3934 (process #1, nbproc = 1)
 uptime = 0d 0h03m48s
 system limits: memmax = unlimited; ulimit-n = 4013
 maxsock = 4013; maxconn = 2000; maxpipes = 0
 current conns = 6; current pipes = 0
 Running tasks: 1/8

active UP backup UP
 active UP, going down backup UP, going down
 active DOWN, going up backup DOWN, going up
 active or backup DOWN not checked
 active or backup DOWN for maintenance (MAINT)

Note: UP with load-balancing disabled is reported as "NOLB".

Display option:
 • Hide 'DOWN' servers
 • Refresh now
 • CSV export

External resources:
 • Primary site
 • Updates (v1.4)
 • Online manual

web-cluster																														
	Queue			Session rate			Sessions				Bytes		Denied		Errors		Warnings		Server											
	Cur	Max	Limit	Cur	Max	Limit	Cur	Max	Limit	Total	LbTot	In	Out	Req	Resp	Req	Conn	Resp	Retr	Redis	Status	LastChk	Wght	Act	Bck	Chk	Dwn	Downtime	Thrtle	
Frontend				6	171	-	6	9	2 000	1 014		578 615	1 654 255	0	0	0	0	0	0	0	0	OPEN								
web1	0	0	-	0	171		0	7	-	1 002	1 002	578 024	1 644 687	0	0	0	0	0	0	0	0	3m48s UP	L4OK in 0ms	1	Y	-	0	0	0s	-
web2	0	0	-	0	0		0	0	-	0	0	0	0	0	0	0	0	0	0	0	0	3m45s DOWN	L4TOUIT in 2001ms	1	Y	-	0	1	3m45s	-
Backend	0	0		0	171		0	7	2 000	1 002	1 002	578 615	1 654 255	0	0	0	0	0	0	0	0	3m48s UP		1	1	0	0	0	0s	

web-cluster													
	Queue			Session rate			Sessions				Bytes		
	Cur	Max	Limit	Cur	Max	Limit	Cur	Max	Limit	Total	LbTot	In	Out
Frontend				6	171	-	6	9	2 000	1 014		578 615	1 654 255
web1	0	0	-	0	171		0	7	-	1 002	1 002	578 024	1 644 687
web2	0	0	-	0	0		0	0	-	0	0	0	0
Backend	0	0		0	171		0	7	2 000	1 002	1 002	578 615	1 654 255

Windows taskbar: 11:46 28/07/2017

Figura 59: Resultados estadísticas Haproxy posterior a la prueba3
 Fuente: El autor
 Elaborado por: El autor

Se recuerda que los valores en cuanto al números de sesiones, en las estadísticas previo a las pruebas realizadas con Javascript desde la consola del navegador, fueron:

Webserver1 = 0 peticiones

Webserver2 = 0 peticiones (**servidor apagado**)

Luego de las pruebas, según lo demostrado en la figura 59, se observa:

Webserver1 = 1002 peticiones

Webserver2 = 0 peticiones (**servidor apagado**)

De acuerdo a los resultados obtenidos en el reporte de estadísticas, se puede evidenciar que; enviadas las peticiones concurrentes de 1000 usuarios, hacia la url de la aplicación web “192.168.0.20/agendacion/medico/”, el servidor Webserver1 ha recibido la totalidad de la carga de las peticiones de los 1000 usuarios, ya que el servidor real Webserver2 se encuentra fuera de servicio

3.2.6 Prueba 4 realizada con javascript desde la consola del navegador con 2 servidores web

Esta prueba consiste en ejecutar un código javascript dentro de la consola del navegador web, concretamente se ejecuta un for, el cual hará un bucle de 1000 repeticiones hacia la aplicación web.

Esta primera prueba con javascript, se realizará con un solo servidor real web funcionando, ya que el otro servidor web, se encuentra apagado. De esta manera se podrá evidenciar el rendimiento del clúster cuando tiene fuera de servicio a uno de los dos servidores web.

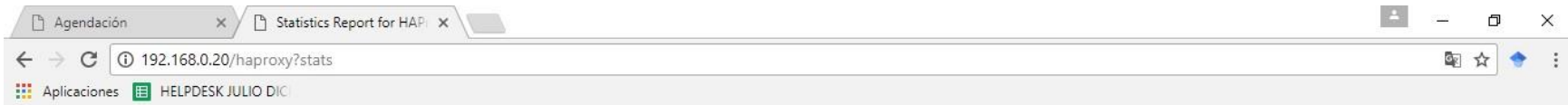
3.2.6.1 Revisión de estadísticas de sesiones y balanceo de carga previo a las pruebas

Se revisa el reporte de estadísticas de número de sesiones y balanceo de carga antes de realizar la prueba.

Las estadísticas de los servidores balanceadores de carga HaproxyWeb1 y HaproxyWeb2, son reiniciadas, por lo tanto, se presentarán en cero.

La dirección para obtener las estadísticas es:

<http://192.168.0.20/haproxy?stats>



HAProxy version 1.4.24, released 2013/06/17

Statistics Report for pid 3879

> General process information

pid = 3879 (process #1, nbproc = 1)
 uptime = 0d 0h01m06s
 system limits: memmax = unlimited; ulimit-n = 4013
 maxsock = 4013; maxconn = 2000; maxpipes = 0
 current conns = 8; current pipes = 0/0
 Running tasks: 1/8

active UP
 active UP, going down
 active DOWN, going up
 active or backup DOWN
 active or backup DOWN for maintenance (MAINT)

backup UP
 backup UP, going down
 backup DOWN, going up
 not checked

Note: UP with load-balancing disabled is reported as "NOLB".

Display option:

- Hide 'DOWN' servers
- Refresh now
- CSV export

External resources:

- Primary site
- Updates (v1.4)
- Online manual

web-cluster																														
	Queue			Session rate			Sessions				Bytes		Denied		Errors			Warnings		Server										
	Cur	Max	Limit	Cur	Max	Limit	Cur	Max	Limit	Total	LbTot	In	Out	Req	Resp	Req	Conn	Resp	Retr	Redis	Status	LastChk	Wght	Act	Bck	Chk	Dwn	Downtme	Thrtt	
Frontend				6	6	-	6	6	2 000	12		591	9 568	0	0	5					OPEN									
web1	0	0	-	0	0		0	0	-	0	0	0	0	0	0	0	0	0	0	0	1m9s UP	L4OK in 0ms	1	Y	-	2	0	0s	-	
web2	0	0	-	0	0		0	0	-	0	0	0	0	0	0	0	0	0	0	0	1m9s UP	L4OK in 0ms	1	Y	-	1	0	0s	-	
Backend	0	0		0	0		0	0	2 000	0	0	591	9 568	0	0	0	0	0	0	0	1m9s UP		2	2	0		0	0s		

web-cluster													
	Queue			Session rate			Sessions				Bytes		
	Cur	Max	Limit	Cur	Max	Limit	Cur	Max	Limit	Total	LbTot	In	Out
Frontend				6	6	-	6	6	2 000	12		591	9 568
web1	0	0	-	0	0		0	0	-	0	0	0	0
web2	0	0	-	0	0		0	0	-	0	0	0	0
Backend	0	0		0	0		0	0	2 000	0	0	591	9 568



Figura 60: Resultados estadísticas haproxy previo a la prueba4

Fuente: El autor

Elaborado por: El autor

3.2.6.2 Script utilizado en el navegador

Consiste en un For el cual realizará 1000 peticiones, pero a diferencia de las pruebas realizadas con la herramienta JMETER donde las peticiones las realizan con 1000 usuarios diferentes, con el For las 1000 peticiones las realizará un único usuario.

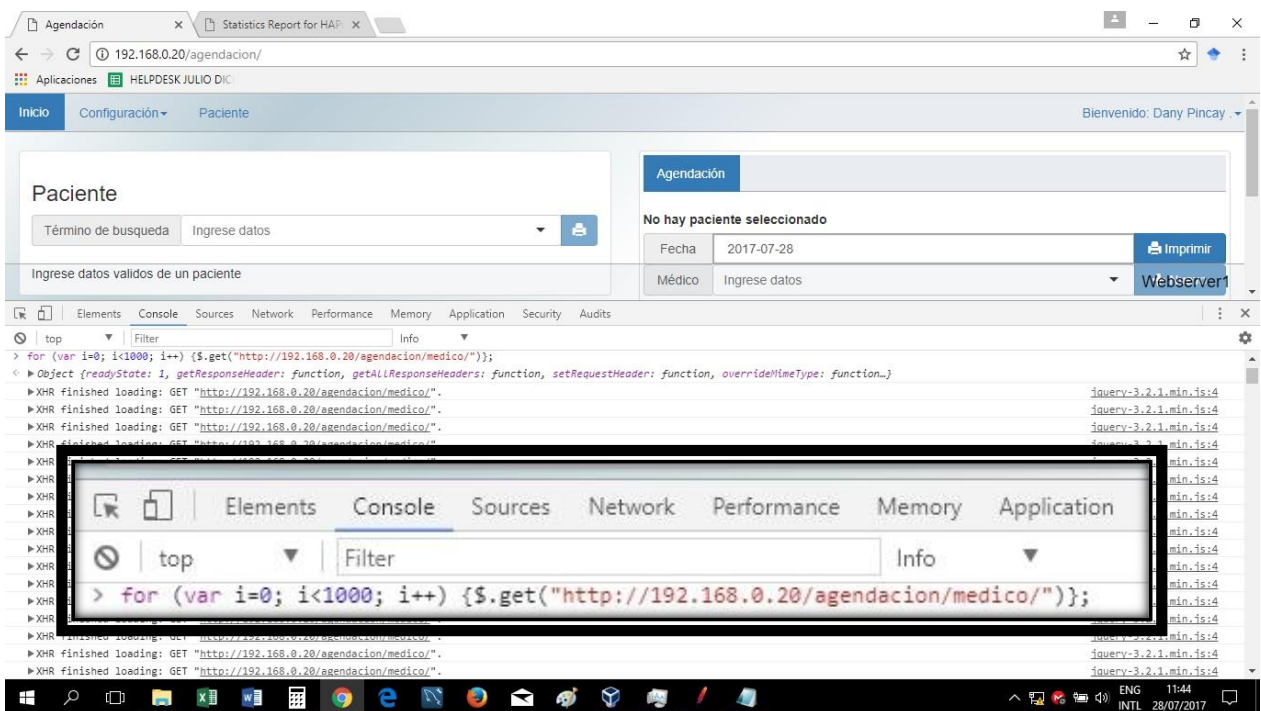


Figura 61: Código JavaScript para la prueba4

Fuente: El autor

Elaborado por: El autor

3.2.6.3 Resultados de la prueba

Una vez ejecutada la prueba, se puede apreciar los resultados obtenidos en número tiempo y peso de cada petición

En la figura 62, se observa que la primera petición tiene como datos: tiempo 40 ms y tamaño 4.6kb.

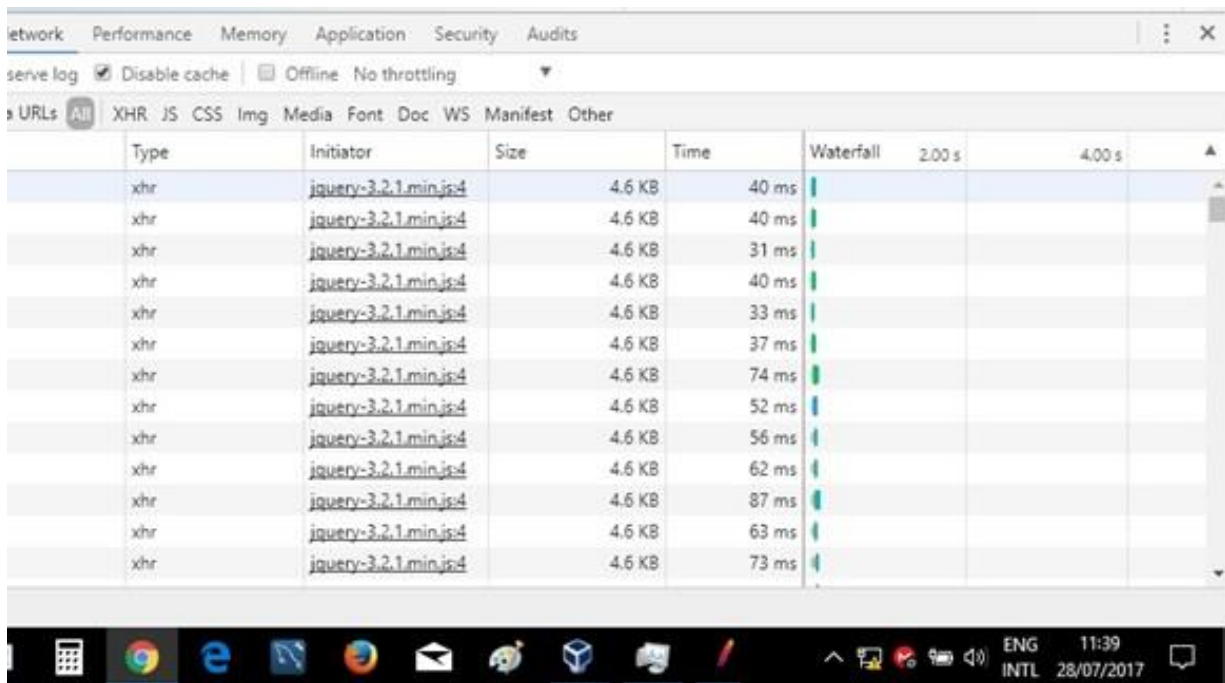


Figura 62: Resultados prueba4

Fuente: El autor

Elaborado por: El autor

En la figura 63, se observa que las peticiones a mitad de la tabla (aproximadamente petición 500) tiene como datos: tiempo 1.98 ms y tamaño 4.6kb.

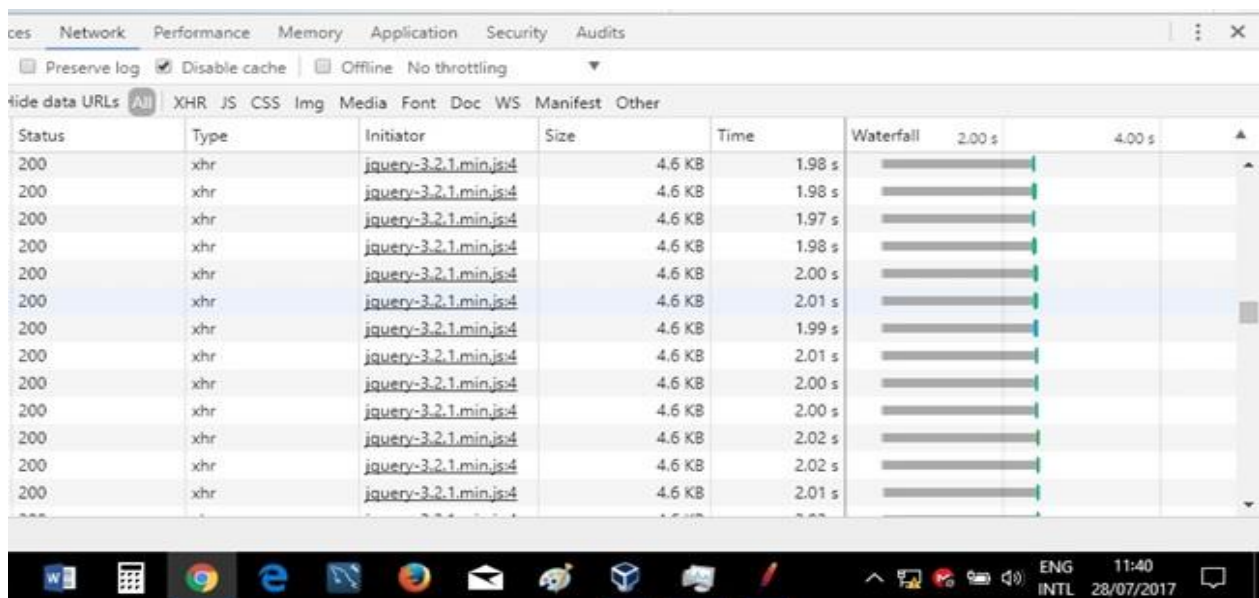


Figura 63: Resultados prueba4

Fuente: El autor

Elaborado por: El autor

Mientras que la figura 64, indica que la última petición tiene como datos: tiempo 3.83 ms y tamaño 4.6 Kb.

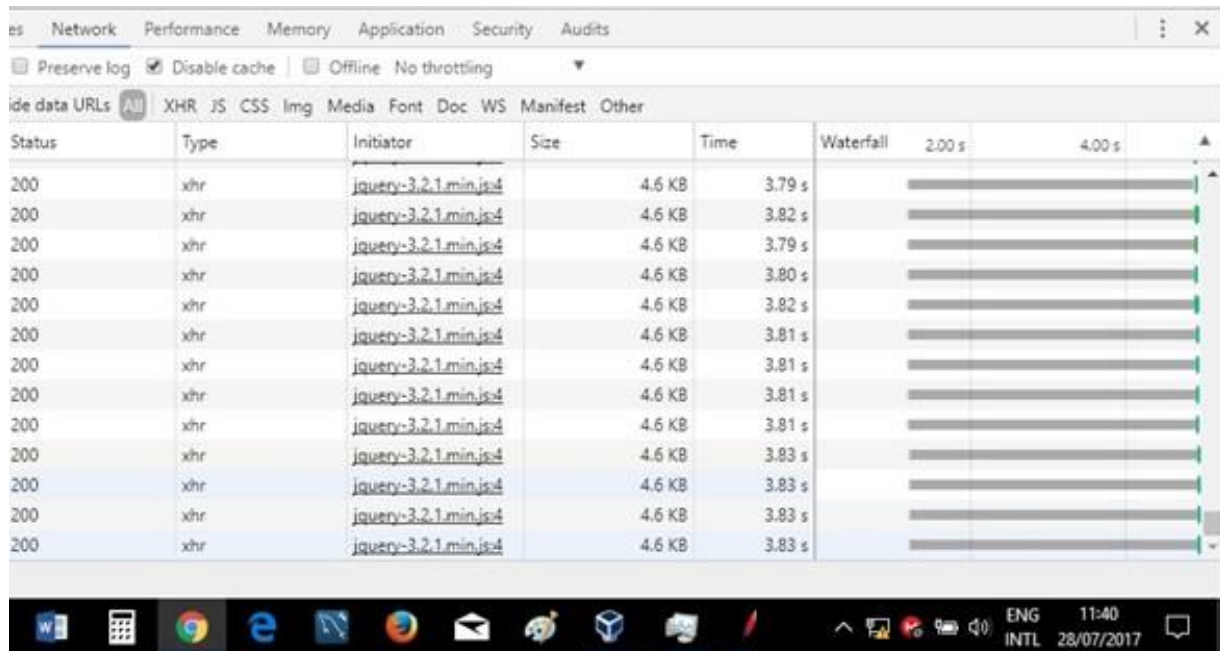


Figura 64: Resultados prueba4
Fuente: El autor
Elaborado por: El autor

3.2.6.4 Revisión de las estadísticas de sesiones y balanceo de carga posterior a las pruebas con javascript

Una vez realizadas las pruebas de estrés a la aplicación y de balanceo de carga, y obtenidos los resultados que nos entrega esta herramienta, se procede a verificar las estadísticas de peticiones y sesiones que ha recibido el Frontend de nuestra arquitectura tecnológica, y además de comprobar el balanceo de carga realizado por la herramienta Haproxy.



HAProxy version 1.4.24, released 2013/06/17

Statistics Report for pid 3879

> General process information

pid = 3879 (process #1, nbproc = 1)
 uptime = 0d 0h00m12s
 system limits: memmax = unlimited; ulimit-n = 4013
 maxsock = 4013; maxconn = 2000; maxpipes = 0
 current conns = 5; current pipes = 0/0
 Running tasks: 1/7

active UP
 active UP, going down
 active DOWN, going up
 active or backup DOWN
 active or backup DOWN for maintenance (MAINT)

backup UP
 backup UP, going down
 backup DOWN, going up
 not checked

Note: UP with load-balancing disabled is reported as "NOLB".

- Display option:
- Hide DOWN servers
 - Refresh now
 - CSV export
- External resources:
- Primary site
 - Updates (v1.4)
 - Online manual

	Queue			Session rate			Sessions				Bytes		Denied		Errors			Warnings			Server										
	Cur	Max	Limit	Cur	Max	Limit	Cur	Max	Limit	Total	LbTot	In	Out	Req	Resp	Req	Conn	Resp	Retr	Redis	Status	LastChk	Wght	Act	Bck	Chk	Dwn	Dwntme	Thrtie		
Frontend	0	0	-	5	221	-	5	9	2 000	1 019		579 206	1 663 385	0	0	10		0	0	0	OPEN										
web1	0	0	-	0	110		0	5	-	501	501	289 000	857 838	0	0	0	0	0	0	0	6m12s UP	L4OK in 0ms	1	Y	-	2	0	0s	-		
web2	0	0	-	0	110		0	5	-	501	501	289 024	788 350	0	0	0	0	0	0	0	6m12s UP	L4OK in 0ms	1	Y	-	1	0	0s	-		
Backend	0	0	0	220		0	7	2 000	1 002	1 002		579 206	1 663 385	0	0		0	0	0	0	6m12s UP		2	2	0		0	0s			

	Queue			Session rate			Sessions				Bytes		
	Cur	Max	Limit	Cur	Max	Limit	Cur	Max	Limit	Total	LbTot	In	Out
Frontend				5	221	-	5	9	2 000	1 019		579 206	1 663 385
web1	0	0	-	0	110		0	5	-	501	501	289 000	857 838
web2	0	0	-	0	110		0	5	-	501	501	289 024	788 350
Backend	0	0		0	220		0	7	2 000	1 002	1 002	579 206	1 663 385



Figura 65: Reporte estadísticas Haproxy posterior a la prueba4
 Fuente: El autor
 Elaborado por: El autor

Se recuerda que los valores en cuanto al números de sesiones, en las estadísticas previo a las pruebas realizadas con Javascript desde la consola del navegador, fueron:

Webserver1 = 0 peticiones

Webserver2 = 0 peticiones

Luego de las pruebas, según lo demostrado en la figura 65, se observa:

Webserver1 = 501 peticiones

Webserver2 = 501 peticiones

De acuerdo a los resultados obtenidos en el reporte de estadísticas, se puede evidenciar que; enviadas las peticiones concurrentes de 1000 usuarios, hacia la url de la aplicación web “192.168.0.20/agendacion/medico/”, y en la que cada petición equivale a 1 sesión enviada hacia los servidores reales Webserver1 y Webserver2.

3.2.7 Análisis de resultados entre las 2 pruebas con Javascript

Luego de haber realizado 2 pruebas de carga y rendimiento código javascript desde la consola del navegador, la primera de ellas con 1 solo servidor real web, y la segunda con los 2 servidores reales web, se procede a realizar el respectivo análisis de ambas pruebas para determinar el rendimiento y efectividad de la arquitectura tecnológica que está presentando como solución a la problemática.

Para esto, se elabora la siguiente tabla comparativa:

Tabla 13: Análisis de resultados entre prueba3 y prueba4

	PRUEBA JMETER CON 1 SERVIDOR WEB	PRUEBA JMETER CON 2 SERVIDORES WEB	
Peticiones recibidas	Webserver1	Webserver1	Webserver2
	1002	501	501
Tiempo total	5.72 segundos	3.83 segundos	

Fuente: El autor

Elaborado por: El autor

De acuerdo a la comparación realizada en la tabla 13, en el análisis se concluye lo siguiente:

- **Alta disponibilidad**, en la prueba 1 realizada directamente desde el navegador de internet se observan características de alta disponibilidad en el servicio, ya que ante la caída del servidor Webserver2, no se sufre pérdida del servicio, ya que éste sigue en funcionamiento el servidor Webserver1, el rendimiento se ve disminuido ya que todas las peticiones las absorbe 1 solo servidor web.
- **Balanceo de carga**, Se evidencia en la prueba 2, el correcto funcionamiento del balanceo de carga entre ambos servidores web (Webserver1 y Webserver2) y un aumento considerable en el rendimiento (peticiones/tiempo), en comparación a la prueba 1 (solo con Webserver1).
- **Rendimiento y escalabilidad**, se deja en evidencia que la arquitectura tecnológica está apta para el aumento de más servidores, y mientras el incremento sucede, esto se ve reflejado en el aumento del rendimiento en las transacciones y peticiones que recibe.
- **Persistencia de sesiones**, en la prueba 1, ante la caída del servidor Webserver2, la aplicación ejecutó el código de almacenamiento y traspaso de sesiones hacia el servidor disponible, en este caso el Webserver1.

CONCLUSIONES

En virtud de haber logrado alcanzar los objetivos propuestos en este trabajo de fin de titulación, se exponen las siguientes conclusiones:

- De acuerdo a las fuentes investigadas, el manejo de grandes volúmenes de información, requiere la gestión y trato correspondiente que merece la misma, es decir, utilizando arquitecturas de gestión de información a gran escala, tanto en hardware como en software.
- El uso de arquitecturas tecnológicas a gran escala, a través de estándares y paradigmas de alto nivel, proporcionan y facilitan el manejo de gran cantidad de información, de manera rápida, ágil, precisa y oportuna, resguardando siempre la integridad y confiabilidad de la misma.
- La implementación como solución, de una arquitectura tecnológica a gran escala (granja de 8 servidores, divididos en 2 clúster), para resolver la problemática del presente proyecto, a través de las herramientas informáticas de código abierto, como son: Haproxy y Heartbeat. Brindan a los sistemas de manejo de información, los atributos y beneficios propios de este tipo de estructuras, entre los cuales se menciona: Alta Disponibilidad, Balanceo de Carga, Escalabilidad, Replicación de datos, Persistencia de Sesiones, como los más destacables; todo esto reflejado en un óptimo rendimiento de la solución planteada e implementada.
- El uso de balanceadores de carga en cada uno de los clústeres, ayuda de forma considerable, a reducir y equilibrar la carga de trabajo a los servidores reales, repartiéndolas de manera equitativa entre todos los servidores que conforman el clúster. De esta manera el rendimiento de la arquitectura en general se refleja en un servicio hacia el usuario mucho más rápido, ágil y confiable.
- En lo referente a la aplicación web; el uso de uno de los Framework de mayor peso, dentro de los del tipo de código abierto, que es Django, facilitó sustancialmente, el éxito del rendimiento de la granja de servidores desarrollada. Ya que las características que posee Django, se ajustan perfectamente a los modelos de desarrollo a gran escala.

- El diseño de la aplicación web, por medio del Framework Django, de código abierto, y con su lenguaje base Python; tuvo un desarrollo exitoso, ya que este aplicativo web, se lo fusionó con la arquitectura tecnológica, a través de una de las características más representativas de Django, que es el “settings.py”. Por medio del cual se pudo redirigir las peticiones concurrentes de los usuarios, hacia los servidores balanceadores de carga, y estos a su vez, distribuían la carga a los servidores reales de Base de Datos.
- Las bondades que nos ofrece Django, entre las cuales tenemos: la reutilización de código y componentes, el uso de una librería propia de mapeo Objeto-Relacional, bibliotecas Http, el archivo de configuración “settings.py”, el servidor de desarrollo Django, entre otras agilitan el proceso de desarrollo de la aplicación, para su pronta puesta en producción.
- Dentro de la gestión de la data, por medio del gestor de Base de Datos Mysql, se pudo realizar satisfactoriamente el manejo de la gran cantidad de información registrada en la Base de Datos. Además de resguardar la integridad de la misma, se realizó implementación y configuración de Replicación de datos, entre los Servidores de Bases de Datos.
- Los resultados obtenidos en las pruebas de carga y estrés ejecutadas a través de la herramienta de código abierto Apache Jmeter, fueron altamente satisfactorios, evidenciándose un aumento en el rendimiento entre las pruebas 1 y pruebas 2 (revisar capítulo III Pruebas a la Solución), y de la misma forma se constata el balanceo de carga realizado por arquitectura tecnológica.
- Los resultados obtenidos en las pruebas de carga, rendimiento y estrés (revisar capítulo III Pruebas a la solución) realizadas y ejecutadas a través de código Javascript, desde la consola del navegador, confirmaron los resultados y éxito, obtenidos con las pruebas realizadas a la arquitectura tecnológica, a través de Apache Jmeter.

RECOMENDACIONES

- Para el análisis e implementación de una arquitectura tecnológica en un entorno de desarrollo a gran escala, se hace primordial, la absorción de conocimientos, conceptualización, métodos y técnicas inherentes a estos modelos de estructuras.
- Se debe definir desde el inicio y con claridad, los atributos y características que se desean que posea la arquitectura tecnológica a implementar, es decir, si debe brindar alta disponibilidad, balanceo de carga, persistencia de sesiones, alto rendimiento o escalabilidad; de estas características pueden ser algunas o todas, dependiendo de los objetivos planteados y de acuerdo a lo que se necesite.
- La elección de la herramienta que plasmará en nuestra arquitectura tecnológica a gran escala, los servicios antes mencionados, dependerá de los objetivos planteados al inicio. Y si no se cuenta con recursos económicos para la adquisición de una herramienta pagada, se puede elegir una herramienta de código abierto, como es el caso de Haproxy, la cual es utilizada por varios entornos de alto rendimiento.
- Es importante la búsqueda y utilización de una herramienta para la ejecución de pruebas de carga, pruebas de rendimiento y pruebas de estrés, que lleven al límite al aplicativo web y a toda la arquitectura tecnológica implementada, de esta manera se evidenciará el éxito en cuanto a conseguir los servicios mencionados en los objetivos del proyecto, que son: balanceo de carga, alta disponibilidad, rendimiento, etc.

BIBLIOGRAFÍA

- Acosta, J., Greiner, C., Dapozo, G., & Estayno, M. (2012). *Medición de atributos POO en frameworks de desarrollo PHP*. Buenos Aires.
- Aguilera, P., & Gómez, E. (2016). *Clustering y grid computing en sistemas de alta disponibilidad para servicios web y mail*. Obtenido de https://www.academia.edu/11855427/CLUSTERING_Y_GRID_COMPUTING_EN_SISTEMAS_DE_ALTA_DISPONIBILIDAD_PARA_SERVICIOS_WEB_Y_MAIL
- Barchéin, M. (2012). *Balanceo dinámico de servidores web basado en la carga de los nodos*. Obtenido de Balanceo dinámico de servidores web basado en la carga de los nodos: <http://blog.inteligencia.com/2012/03/balanceo-dinamico-de-servidores-web.html>
- Bonilla, M., Vargas, L., & Meneses, E. (2009). *Arquitecturas Distribuidas Como Solución a Sistemas Demandantes*. Memorias de la ULA.
- Buyya, R. (1999). *High Performance Cluster Computing: Architectures and Systems*. NJ: Prentice Hall.
- Camps Paré, R. (2007). *Introducción a las bases de datos*. Barcelona.
- Cardona, C. (2009). *Propuesta Metodológica para la Realización de Pruebas de Software en un Ambiente Productivo*. Medellín: Universidad Nacional de Colombia.
- Challenger, I., Díaz, Y., & Becerra, R. (2014). El lenguaje de programación Python/The programming language Python. *Revista Trimestral Ciencias Holguín*.
- Chavez, C., & Buñay, G. (2008). *ESTUDIO COMPARATIVO DE LAS TECNOLOGIAS PYTHON Y PERL PARA DESARROLLAR APLICACIONES WEB IMPLEMENTADO AL PROGRAMA DE ALFABETIZACION DEL CONSEJO PROVINCIAL DE CHIMBORAZO*. Riobamba: Bachelor's thesis.
- Clavijo, P. (2012). *Clústers de alta disponibilidad (ha)*. Obtenido de Clústers de alta disponibilidad (ha): <http://www.lintips.com/?q=node/119>

- Cobo, A., Gómez, P., Pérez, D., & Rocha, R. (2005). *PHP y MYSQL Tecnologías para el desarrollo de aplicaciones web*. Díaz de Santos.
- Conolly, T. (2005). *Sistemas de Base de Datos*. Madrid.
- Cujano, S. F. (2011). *Análisis e implementación de alta disponibilidad mediante clustering en sistemas de call center basados en voip*. Ribamba.
- Cumba, P., & Barreno, B. (2012). *"Análisis de Python con Django frente a Ruby on Rails para desarrollo ágil de aplicaciones web. Caso práctico: DECH"*. Riobamba.
- Da Rosa, F., & Heinz, F. (2007). *Guía práctica sobre software libre*.
- De la Guardia, C. (2010). *Grok 1.0 Web Development*. Packt Publishing Ltd.
- Deitel, P., & Deitel, H. (2008). *Rich Internet Applications, and Web Development for programmers*. Prentice Hall Press Upper Saddle River.
- Fernández, Y., & Díaz, Y. (2012). Patrón Modelo-Vista-Controlador. *Revista Telem@tica*, 47-57.
- Fuertes, V., & Guevara, J. C. (2010). *Análisis del patrón modelo vista controlador implementado en lenguajes de software libre para el desarrollo de aplicaciones web. Caso práctico: Liceo de talentos Stephen Hawking*. Obtenido de Análisis del patrón modelo vista controlador implementado en lenguajes de software libre para el desarrollo de aplicaciones web. Caso práctico: Liceo de talentos Stephen Hawking: <http://dspace.esPOCH.edu.ec/bitstream/123456789/550/1/18T00441.pdf>
- Gallardo, F. (2011). *Diseño de una solución para servidores de alta disponibilidad y balanceo de carga con Open Source*. Obtenido de Diseño de una solución para servidores de alta disponibilidad y balanceo de carga con Open Source: <http://es.scribd.com/doc/57937293/33/Funcionamiento-de-un-clúster>
- Godino, F. (2013). *Diseño de una arquitectura escalable y de alta disponibilidad para un sistema middleware*. Obtenido de Diseño de una arquitectura escalable y de alta disponibilidad para un sistema middleware: http://oa.upm.es/32647/1/PFC_FERNANDO_GODINO_GONZALEZ.pdf

- haproxy*. (2017). Obtenido de <http://www.haproxy.org/>
- Hurtado, A. (2011). *Propuesta de un servidor Web Apache 2 sobre un Cluster en Linux*. Santa Clara: Universidad Central “Marta Abreu” de Las Villas.
- Jiménez, D., & Medina, A. (2014). Clúster de Alto Rendimiento. *Journal Innovación y Tecnología*, 16-27.
- López, M. (2016). *Instalación, configuración y evaluación de un servidor web de alta disponibilidad con equilibrado de carga*. Valencia: Universidad Politécnica de Valencia.
- Molina, J., Loja, N., Zea, M., & Loaiza, E. (2016). Evaluación de los Frameworks en el Desarrollo de Aplicaciones Web con Python. *Revista Latinoamericana de Ingeniería de Software*, 201-207.
- Montes de Oca Sánchez de Bustamante, A. (2004). Arquitectura de información y usabilidad: nociones básicas para los profesionales de la información. *ACIMED*, 1-62.
- MySQL – Replicación master/escalvo y master/master|MySQL – master/slave and master/master replication*. (13 de 07 de 2013). Obtenido de <http://www.boksar.info/?p=423>
- Nevedrov, D. (2006). Using JMeter to Performance Test Web Services. 1-11.
- Oracle Corporation. (2014). *MySQL 5.0 Reference Manual*.
- Oracle Corporation. (2017). *Manual de referencia de MySQL 5.7*. Obtenido de <https://dev.mysql.com/doc/refman/5.7/en/replication.html>
- Potencier, F., & Zninotto, F. (2012). *Symfony la guía definitiva*. Saatavissa.
- Rodríguez Castellanos, N., & Rodríguez Castiblanco, H. (2015). Implementación de un balanceador de carga. *Revista de Investigación de la Facultad de Ingeniería EAM*.
- Silberschatz, A., Galvin, P., & Gagne, G. (2005). *Fundamentos de Sistemas Operativos*.

- Sinisterra, M., Diaz, T., & Ruiz, E. (2012). Clúster de balanceo de carga y alta disponibilidad para servicios web y mail. *Informador Técnico Colombia* , 93-102. Obtenido de Clúster de balanceo de carga y alta disponibilidad para servicios web y mail: <https://es.scribd.com/document/317725491/Dialnet-ClusterDeBalanceoDeCargaYAltaDisponibilidadParaSer-4364562-pdf>
- Sloan, J. (2004). *High Performance Linux Clusters with OSCAR, Rocks, OpenMosix, and MPI: A Comprehensive Getting-Started Guide*. High Performance Linux Clusters with OSCAR, Rocks, OpenMosix, and MPI: A Comprehensive Getting-Started Guide.
- Stallman, R. M. (2004). *Software libre para una sociedad libre*. Traficantes de sueños.
- Tarreau, W. (2012). *Haproxy configuration manual*.
- TutorialsPoint. (17 de febrero de 2016). *TurboGears*. Obtenido de <http://www.tutorialspoint.com/turbogears/turbogears>
- Universidad de Alicante. (2014). *Introducción a JavaServer Faces*. Obtenido de Introducción a JavaServer Faces: <http://www.jtech.ua.es/j2ee/publico/jsf-2012-13/sesion01-apuntes.html#Caracter%C3%ADsticas+de+JSF>
- Vargas , D., & Abril, A. (2012). *Introducción al Balanceo de Carga en Aplicaciones Web*. Obtenido de Introducción al Balanceo de Carga en Aplicaciones Web: <https://redesocialespaldava.files.wordpress.com/2012/11/articulo.pdf>
- Wangsmo, M. (1999). *Piranha load-balanced web and ftp clusters*. Obtenido de <http://www.redhat.com/support/wpapers/piranha/>
- Zapata, C., & Cardona, C. (2011). Comparación de las características de algunas herramientas de software para pruebas de carga. *Revista Avances en Sistemas e Informática*, 143-154.
- Zhang, W. (2000). *Linux Virtual Server for Scalable Network Services*.
- Zumba, C. P. (2011). *Diseño y desarrollo de un prototipo de un cluster en linux de alta disponibilidad para satisfacer la demanda de acceso web en la carrera de*

ingeniería en sistemas computacionales y el balanceo de carga de los servicios.
Obtenido de <http://repositorio.ug.edu.ec/handle/redug/6750>

ANEXOS

1.1 Codificación de aplicación web (Configuración)

1.1.1 settings.py

```
import os

# Build paths inside the project like this: os.path.join(BASE_DIR, ...)
import django

BASE_DIR = os.path.dirname(os.path.dirname(os.path.abspath(__file__)))

# Quick-start development settings - unsuitable for production
# See https://docs.djangoproject.com/en/1.10/howto/deployment/checklist/

# SECURITY WARNING: keep the secret key used in production secret!
SECRET_KEY = 'd3mb7&n^46iq@))4iiqf0q%6s$7oz%i_52yaweb)clbh=q7@_8'

# SECURITY WARNING: don't run with debug turned on in production!
DEBUG = True

ALLOWED_HOSTS = ['localhost']

# Application definition

INSTALLED_APPS = [
    'agendacion.apps.AgendacionConfig',
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
]

MIDDLEWARE = [
    'django.middleware.security.SecurityMiddleware',
    'django.contrib.sessions.middleware.SessionMiddleware',
    'django.middleware.common.CommonMiddleware',
    'django.middleware.csrf.CsrfViewMiddleware',
    'django.contrib.auth.middleware.AuthenticationMiddleware',
    'django.contrib.messages.middleware.MessageMiddleware',
    'django.middleware.clickjacking.XFrameOptionsMiddleware',
]

ROOT_URLCONF = 'HistoriaClinica.urls'

TEMPLATES = [
    {
```

```

'BACKEND': 'django.template.backends.django.DjangoTemplates',
'DIRS': [],
'APP_DIRS': True,
'OPTIONS': {
    'context_processors': [
        'django.template.context_processors.debug',
        'django.template.context_processors.request',
        'django.contrib.auth.context_processors.auth',
        'django.contrib.messages.context_processors.messages',
    ],
},
]

WSGI_APPLICATION = 'HistoriaClinica.wsgi.application'

# Database
# https://docs.djangoproject.com/en/1.10/ref/settings/#databases

DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.mysql',
        'HOST': '192.168.0.21',
        'NAME': 'hospital',
        'USER': 'haproxy_root',
        'PASSWORD': 'utpl'
    }
}

# Password validation
# https://docs.djangoproject.com/en/1.10/ref/settings/#auth-password-validators
AUTH_PASSWORD_VALIDATORS = [
    {
        'NAME': 'django.contrib.auth.password_validation.UserAttributeSimilarityValidator',
    },
    {
        'NAME': 'django.contrib.auth.password_validation.MinimumLengthValidator',
    },
    {
        'NAME': 'django.contrib.auth.password_validation.CommonPasswordValidator',
    },
    {
        'NAME': 'django.contrib.auth.password_validation.NumericPasswordValidator',
    },
]

# Internationalization
# https://docs.djangoproject.com/en/1.10/topics/i18n/

LANGUAGE_CODE = 'es-ec'

```

```

TIME_ZONE = 'UTC'
USE_I18N = True
USE_L10N = True
USE_TZ = True
# Static files (CSS, JavaScript, Images)
# https://docs.djangoproject.com/en/1.10/howto/static-files/
STATIC_URL = '/static/'

STATIC_ROOT = os.path.join(BASE_DIR, "agendacion", "static")
"""
STATICFILES_DIRS = [
    os.path.join(BASE_DIR, "static")
]

```

1.1.2 models.py

```

from __future__ import unicode_literals
from django.db import models

class Grupo_Cultural(models.Model):
    codigo = models.AutoField(primary_key=True)
    nombre = models.CharField(max_length=30, unique=True, null=False)

class Instruccion_Educativa(models.Model):
    codigo = models.AutoField(primary_key=True)
    nombre = models.CharField(max_length=30, unique=True, null=False)

class Pais(models.Model):
    codigo = models.AutoField(primary_key=True)
    nombre = models.CharField(max_length=30, unique=True, null=False)

    def __unicode__(self):
        return self.nombre

class Provincia(models.Model):
    codigo = models.AutoField(primary_key=True)
    nombre = models.CharField(max_length=30, unique=True, null=False)
    pais = models.ForeignKey(Pais, on_delete=models.CASCADE)

    def __unicode__(self):
        return self.nombre

class Ciudad(models.Model):
    codigo = models.AutoField(primary_key=True)
    nombre = models.CharField(max_length=30, unique=True, null=False)
    provincia = models.ForeignKey(Provincia, on_delete=models.CASCADE)

    def __unicode__(self):
        return self.nombre

```

```

class Afinidad(models.Model):
    codigo = models.AutoField(primary_key=True)
    nombre = models.CharField(max_length=30, unique=True, null=False)

    def __unicode__(self):
        return self.nombre

class Pariente(models.Model):
    codigo = models.AutoField(primary_key=True)
    cedula = models.CharField(max_length=10, null=False)
    nombres = models.CharField(max_length=40, null=False)
    apellidos = models.CharField(max_length=40, null=False)
    telefono = models.CharField(max_length=10)
    direccion = models.CharField(max_length=80)
    afinidad = models.ForeignKey(Afinidad, on_delete=models.CASCADE)

class Paciente(models.Model):
    codigo = models.AutoField(primary_key=True)
    cedula = models.CharField(max_length=10, unique=True, null=False)
    nombres = models.CharField(max_length=40, null=False)
    apellidos = models.CharField(max_length=40, null=False)
    telefono = models.CharField(max_length=10)
    direccion = models.CharField(max_length=120)
    sexo = models.CharField(max_length=1, choices= (('m', 'Masculino'), ('f', 'Femenino')))
    ciudad = models.ForeignKey(Ciudad, on_delete=models.CASCADE)
    grupo_cultural = models.ForeignKey(Grupo_Cultural, on_delete=models.CASCADE)
    instruccion_educativa = models.ForeignKey(Instruccion_Educativa,
on_delete=models.CASCADE)
    parientes = models.ManyToManyField(Pariente)

    @property
    def nombres_completos(self):
        "Returns the person's full name."
        return '%s%s' % (self.nombres, self.apellidos)

class Usuario(models.Model):
    codigo = models.AutoField(primary_key=True)
    cedula = models.CharField(max_length=10, unique=True, null=False)
    nombres = models.CharField(max_length=40, null=False)
    apellidos = models.CharField(max_length=40, null=False)
    login = models.CharField(max_length=30, unique=True, null=False)
    password = models.CharField(max_length=30, null=False)
    ciudad = models.ForeignKey(Ciudad, on_delete=models.CASCADE)
    tipo_usuario = models.CharField(max_length=1, choices= (('a', 'Agendador'), ('m', 'Medico'),
('s', 'Administrador')))

class Agendador(models.Model):
    codigo = models.AutoField(primary_key=True)

```

```

usuario = models.ForeignKey(Usuario, on_delete=models.CASCADE)

class Especialidad(models.Model):
    codigo = models.AutoField(primary_key=True)
    nombre = models.CharField(max_length=30, unique=True, null=False)

class Medico(models.Model):
    codigo = models.AutoField(primary_key=True)
    usuario = models.ForeignKey(Usuario, on_delete=models.CASCADE)
    especialidad = models.ForeignKey(Especialidad, on_delete=models.CASCADE)

class Consultorio(models.Model):
    codigo = models.AutoField(primary_key=True)
    serie = models.CharField(max_length=3, unique=True, null=False)
    nombre = models.CharField(max_length=30, unique=True, null=False)

class Enfermedad(models.Model):
    codigo = models.AutoField(primary_key=True)
    cie10 = models.CharField(max_length=5, unique=True, null=False)
    nombre = models.CharField(max_length=255, unique=True, null=False)

class Diagnostico(models.Model):
    codigo = models.AutoField(primary_key=True)
    enfermedad = models.ForeignKey(Enfermedad)
    tipo = models.CharField(max_length=3)

class Turno(models.Model):
    codigo = models.AutoField(primary_key=True)
    fecha = models.DateField()
    hora = models.TimeField()
    observacion = models.CharField(max_length=100)
    medico = models.ForeignKey(Medico, on_delete=models.CASCADE)
    paciente = models.ForeignKey(Paciente, on_delete=models.CASCADE)
    consultorio = models.ForeignKey(Consultorio, on_delete=models.CASCADE)

class Historia_Clinica(models.Model):
    codigo = models.AutoField(primary_key=True)
    fecha = models.DateField()
    diagnosticos = models.ManyToManyField(Diagnostico)
    resumen_clinico = models.CharField(max_length=100)
    medico = models.ForeignKey(Medico, on_delete=models.CASCADE)

```

1.1.3 url.py

```

from django.conf.urls import url, include
from django.contrib import admin
from agendacion import urls
from django.views.generic.base import RedirectView

```

```
urlpatterns = [
    url(r'^$', RedirectView.as_view(url='/agendacion/')),
    url(r'^agendacion/', include(urls)),
    url(r'^admin/', admin.site.urls),
```

1.1.4 wsgi.py

```
import os

from django.core.wsgi import get_wsgi_application

os.environ.setdefault("DJANGO_SETTINGS_MODULE", "HistoriaClinica.settings")

application = get_wsgi_application()
```

1.2 Código aplicación web (vistas)

1.2.1 historia_clinica.py

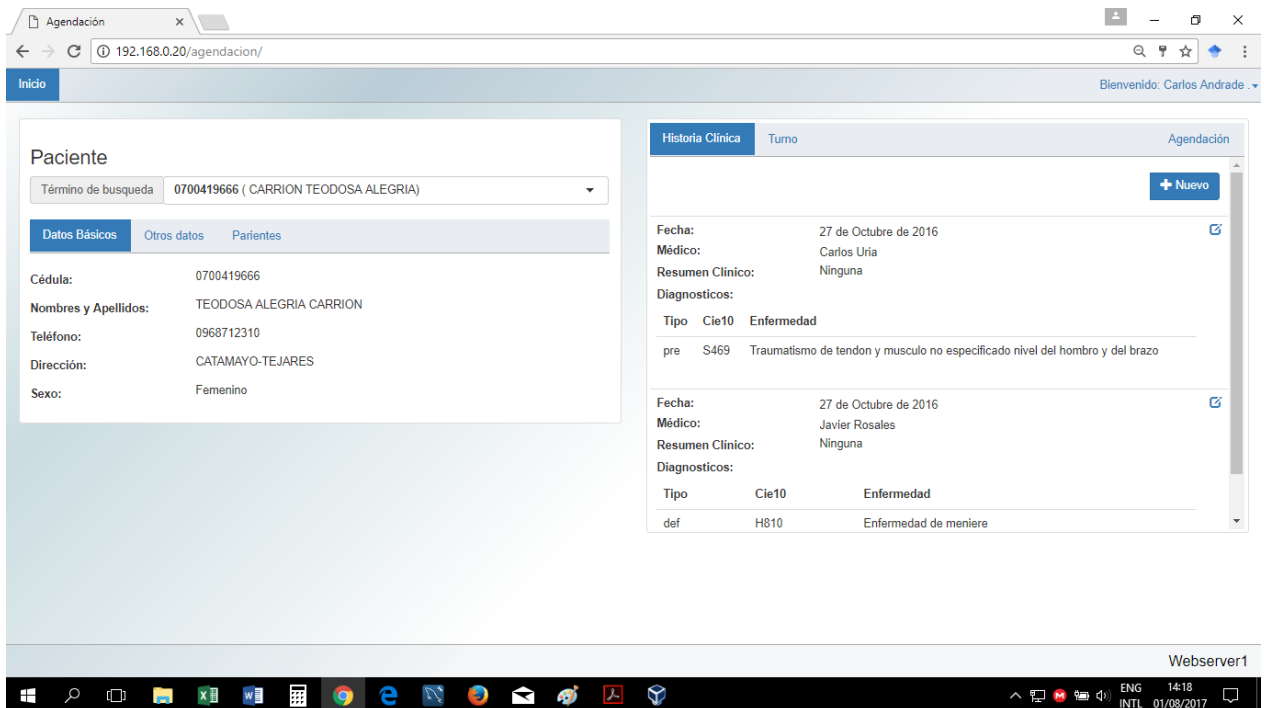


Figura 66: Vista historia clínica

Fuente: El autor

Elaborado por: El autor

```
import __builtin__
```

```

from django.http.response import JsonResponse
from django.shortcuts import render, get_object_or_404

from agendacion.models import Historia_Clinica, Medico, Enfermedad, Paciente, Diagnostico

#
# Carga la vista principal de la gestion de Historia Clinica
#
def index(request):
    return render(request, "paciente/historia_clinica/index.html")

#
# Carga el listado de historias clinicas segun un cedula de paciente
#
def list(request):
    cedula = request.GET.get('cedula', "").encode("utf_8")
    hl = Historia_Clinica.objects.filter(paciente__cedula=cedula).order_by('-fecha')
    historia_clinica_list = __builtin__.list()
    for h in hl:
        historia_clinica_list.append({
            'codigo': h.codigo,
            'fecha': h.fecha,
            'diagnosticos': h.diagnosticos.all(),
            'resumen_clinico': h.resumen_clinico,
            'medico': h.medico,
            'paciente': h.paciente
        })

    return render(request, "paciente/historia_clinica/list.html", {'historia_clinica_list':
historia_clinica_list, 'cedula': cedula})

#
# Muestra un formulario para la creacion de un registro de Historia_Clinica
#
def new(request):
    historia_clinica = {'medico': None}

    try:
        historia_clinica['medico'] = get_object_or_404(Medico,
usuario__pk=request.session['user_id'])
    except:
        None

    return render(request, "paciente/historia_clinica/form.html", {'historia_clinica': historia_clinica,
'is_edit': 1})

#
# Muestra un formulario para el edicion de un registro de Historia_Clinica segun codigo de
historia clinica proporcionado
#

```



```

def edit(request, historia_clinica_id):
    historia_clinica = get_object_or_404(Historia_Clinica, pk=historia_clinica_id)
    diagnosticos = historia_clinica.diagnosticos.all()

    return render(request, "paciente/historia_clinica/form.html", {'historia_clinica': historia_clinica,
'diagnosticos': diagnosticos, 'is_edit': 2})
#
# Elimina y guarda o modifica los registros de Historia_Clinica segun datos recibidos
#
def save(request):
    if len(request.POST) > 0:
        is_edit = request.POST['is_edit']

        if is_edit == '1':
            historia_clinica = Historia_Clinica()
        else:
            historia_clinica = get_object_or_404(Historia_Clinica, pk= int(request.POST['codigo']) )

        #try:
        historia_clinica.fecha = request.POST['fecha'].encode("utf_8")
        historia_clinica.resumen_clinico = request.POST['resumen_clinico'].encode("utf_8")
        historia_clinica.medico = Medico.objects.get(pk=request.POST['medico'].encode("utf_8"))
        historia_clinica.paciente =
Paciente.objects.get(cedula=request.POST['cedula'].encode("utf_8"))
        historia_clinica.save()
        historia_clinica.diagnosticos.all().delete()
        length = len(request.POST.getlist('diagnostico_codigo'))
        if (length > 0):

            for i in range(length):
                historia_clinica.diagnosticos.add(
                    Diagnostico.objects.create(
                        tipo=request.POST.getlist('diagnostico_tipo')[i],

enfermedad=Enfermedad.objects.get(pk=request.POST.getlist('diagnostico_enfermedad')[i].enc
ode("utf_8")),
                    )
                )

            return JsonResponse({'success': True})

        else:
            return JsonResponse({'success': False, 'msg': 'No existen datos'})

```

1.2.2 turno

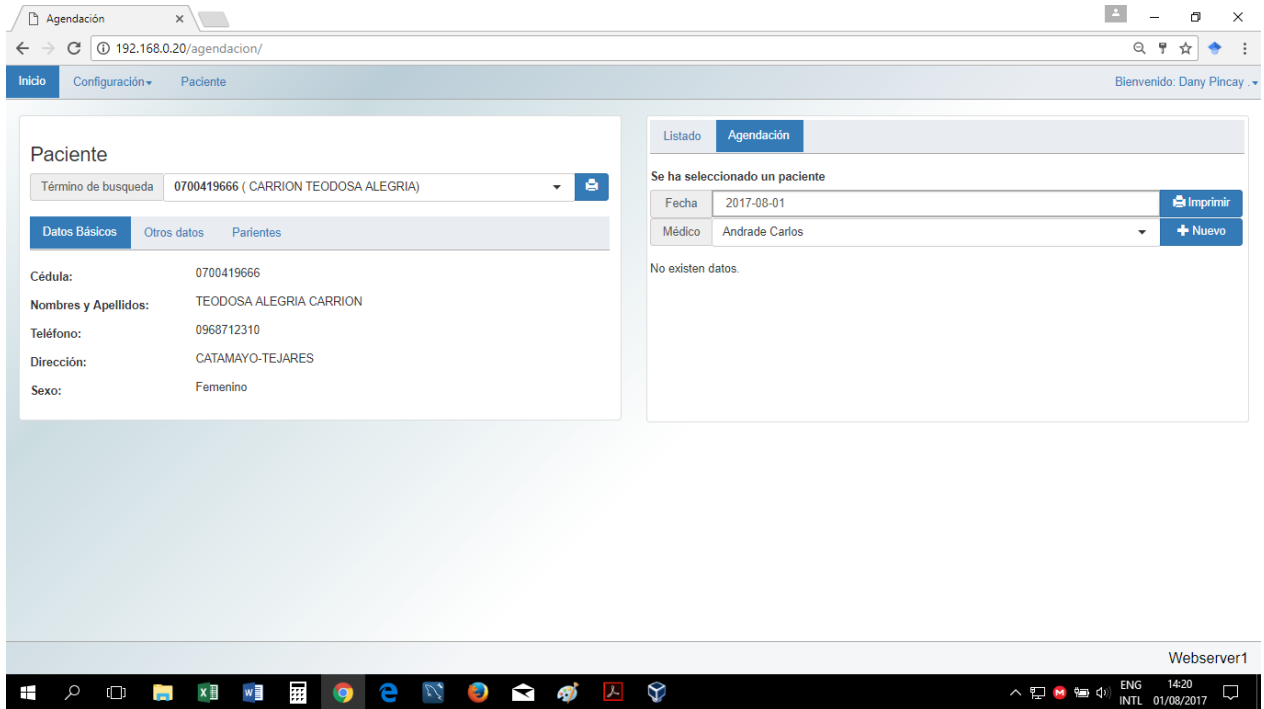


Figura 67: Vista turno
Fuente: El autor
Elaborado por: El autor

```
from datetime import datetime
from django.db.models.query_utils import Q
from django.http.response import JsonResponse, HttpResponse
from django.shortcuts import render, get_object_or_404

from agendacion.models import Ciudad, Medico, Paciente, Consultorio, Usuario
from ..models import Turno
#
# Carga la vista principal de la gestion de Turno
#
def index(request):
    medico_list = Medico.objects.all().order_by('usuario__apellidos')

    if request.session.get('user_id', False):
        usuario = get_object_or_404(Usuario, pk=request.session['user_id'])

        if usuario.tipo_usuario == 'a':
```

```

        return render(request, "turno/indexAgendador.html", {'medico_list': medico_list})
    elif usuario.tipo_usuario == 'm':
        return render(request, "turno/indexMedico.html", {'medico_list': medico_list})
    else:
        return HttpResponse("")

else:
    return HttpResponse("")

#
# Carga el listado de turnos segun un parametro de busqueda
#
def list(request):
    fecha = request.GET.get('fecha', "").encode("utf_8")
    medico = request.GET.get('medico', "").encode("utf_8")

    try:
        turno_list = Turno.objects.filter(fecha=fecha, medico__codigo=medico).order_by('hora')
    except:
        turno_list = []

    if request.session.get('user_id', False):
        usuario = get_object_or_404(Usuario, pk=request.session['user_id'])

        if usuario.tipo_usuario == 'a':
            return render(request, "turno/listAgendador.html", {'turno_list': turno_list, 'fecha': fecha,
            'medico': medico})
        elif usuario.tipo_usuario == 'm':
            return render(request, "turno/listMedico.html", {'turno_list': turno_list, 'fecha': fecha,
            'medico': medico})
        else:
            return HttpResponse("")

    else:
        return HttpResponse("")

#
# Muestra un formulario para la creacion de un registro de Turno
#
def new(request):
    try:
        turno = Turno()
        turno.paciente = get_object_or_404(Paciente, cedula=request.GET['cedula'])
        turno.medico = get_object_or_404(Medico, pk=request.GET['medico'])
        turno.fecha = datetime.strptime(request.GET['fecha'], "%Y-%m-%d").date()
        consultorio_list = Consultorio.objects.all().order_by('nombre')
        return render(request, "turno/form.html", {'turno': turno, 'consultorio_list': consultorio_list,
        'is_edit': 1})
    except:

```

```

        return HttpResponseRedirect("Los parametros recibidos son incorrectos")
#
# Muestra un formulario para el edicion de un registro de Turno segun codigo de turno
proporcionado
#
def edit(request, turno_id):
    turno = get_object_or_404(Turno, pk=turno_id)
    consultorio_list = Consultorio.objects.all().order_by('nombre')
    return render(request, "turno/form.html", {'turno': turno, 'consultorio_list': consultorio_list,
'is_edit': 2})
#
# Elimina un registro de Turno segun codigo de turno proporcionado
#
def delete(request, turno_id):
    turno = get_object_or_404(Turno, pk=turno_id)

    try:
        if(request.GET['confirm'] == 'true'):
            turno.delete()
            return JsonResponse({'success': True})
        else:
            return JsonResponse({'success': False, 'msg': 'No se ha confirmado la eliminacion'})
    except:
        return JsonResponse({'success': False, 'msg': 'Registro no pudo ser eliminado'})
#
# Guarda o modifica un registro de Turno segun datos recibidos
#
def save(request):
    if len(request.POST) > 0:
        is_edit = request.POST['is_edit']

        if is_edit == '1':
            turno = Turno()
        else:
            turno = get_object_or_404(Turno, pk= int(request.POST['codigo']))

        try:
            turno.fecha = request.POST['fecha'].encode("utf_8")
            turno.hora = request.POST['hora'].encode("utf_8")
            turno.observacion = request.POST['observacion'].encode("utf_8")
            turno.medico = Medico.objects.get(pk=request.POST['medico'].encode("utf_8"))
            turno.paciente = Paciente.objects.get(pk=request.POST['paciente'].encode("utf_8"))
            turno.consultorio =
Consultorio.objects.get(pk=request.POST['consultorio'].encode("utf_8"))
            turno.save()

            return JsonResponse({'success': True})
        except:
            return JsonResponse({'success': False, 'msg': 'Registro no pudo ser guardado'})
    else:

```

```
return JsonResponse({'success': False, 'msg': 'No existen datos'})
```

1.2.3 afinidad.py

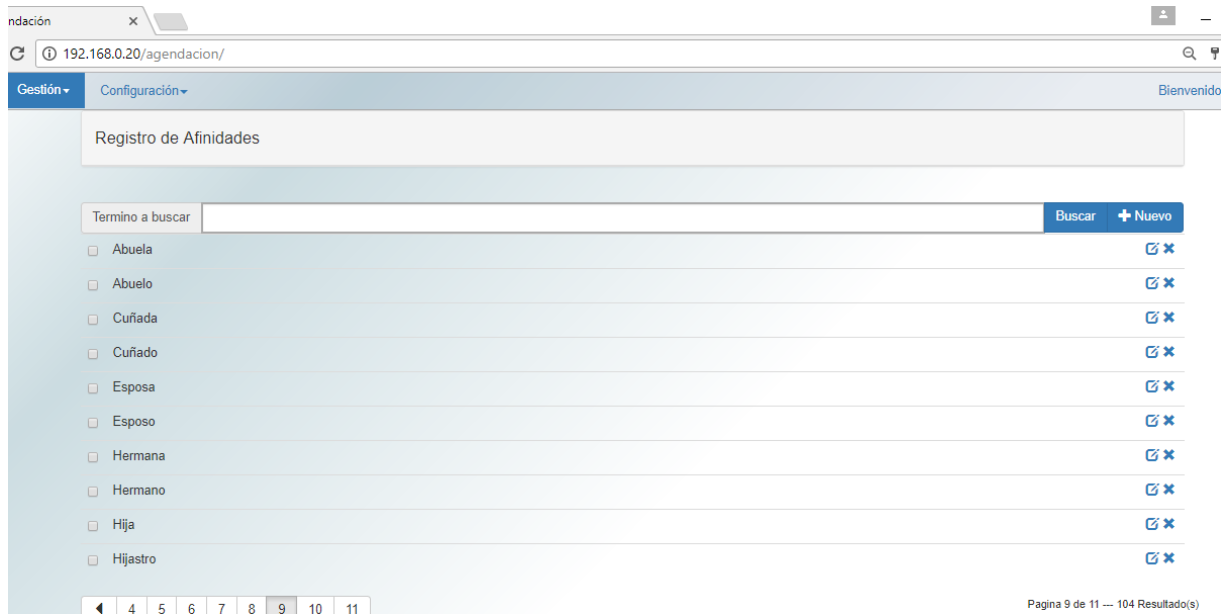


Figura 68: Vista afinidad

Fuente: El autor

Elaborado por: El autor

```
from django.db.utils import IntegrityError
from django.http.response import JsonResponse
from django.shortcuts import render, get_object_or_404

from ..models import Afinidad

#
# Carga la vista principal de la gestion de Afinidad
#
def index(request):
    return render(request, "afinidad/index.html")
#
# Carga el listado de afinidades segun un parametro de busqueda
#
def list(request):
    page = int(request.GET.get('page', 1))
    query = request.GET.get('query', "").encode("utf_8")

    inicio = (page - 1) * 10
    fin = page * 10

    queryset = Afinidad.objects.filter(nombre__icontains=query)
    count = queryset.count()
```

```

afinidad_list = queryset.order_by('nombre')[inicio:fin]

total_pages = count / 10 + 1

return render(request, "afinidad/list.html", {'afinidad_list': afinidad_list, 'query': query, 'count':
count, 'range_pages': range(total_pages), 'page': page})
#
# Muestra un formulario para la creacion de un registro de Afinidad
#
def new(request):
    return render(request, "afinidad/form.html", {'afinidad': {}, 'is_edit': 1})
#
# Muestra un formulario para el edicion de un registro de Afinidad segun codigo de afinidad
proporcionado
#
def edit(request, afinidad_id):
    afinidad = get_object_or_404(Afinidad, pk=afinidad_id)
    return render(request, "afinidad/form.html", {'afinidad': afinidad, 'is_edit': 2})
#
# Elimina un registro de Afinidad segun codigo de afinidad proporcionado
#
def delete(request, afinidad_id):
    afinidad = get_object_or_404(Afinidad, pk=afinidad_id)

    try:
        if(request.GET['confirm'] == 'true'):
            afinidad.delete()
            return JsonResponse({'success': True})
        else:
            return JsonResponse({'success': False, 'msg': 'No se ha confirmado la eliminacion'})
    except:
        return JsonResponse({'success': False, 'msg': 'Registro no pudo ser eliminado'})
#
# Guarda o modifica un registro de Afinidad segun datos recibidos
#
def save(request):
    if len(request.POST) > 0:
        is_edit = request.POST['is_edit']
        if is_edit == '1':
            afinidad = Afinidad()
        else:
            afinidad = get_object_or_404(Afinidad, pk= int(request.POST['codigo']))
    try:
        afinidad.nombre = request.POST['nombre'].encode("utf_8")
        afinidad.save()
        return JsonResponse({'success': True})
    except IntegrityError as e:
        return JsonResponse({'success': False, 'msg': 'Registro ya existe'})
    except:
        return JsonResponse({'success': False, 'msg': 'Registro no pudo ser guardado'})

```

```

else:
    return JsonResponse({'success': False, 'msg': 'No existen datos'})

```

1.2.4 agendador.py

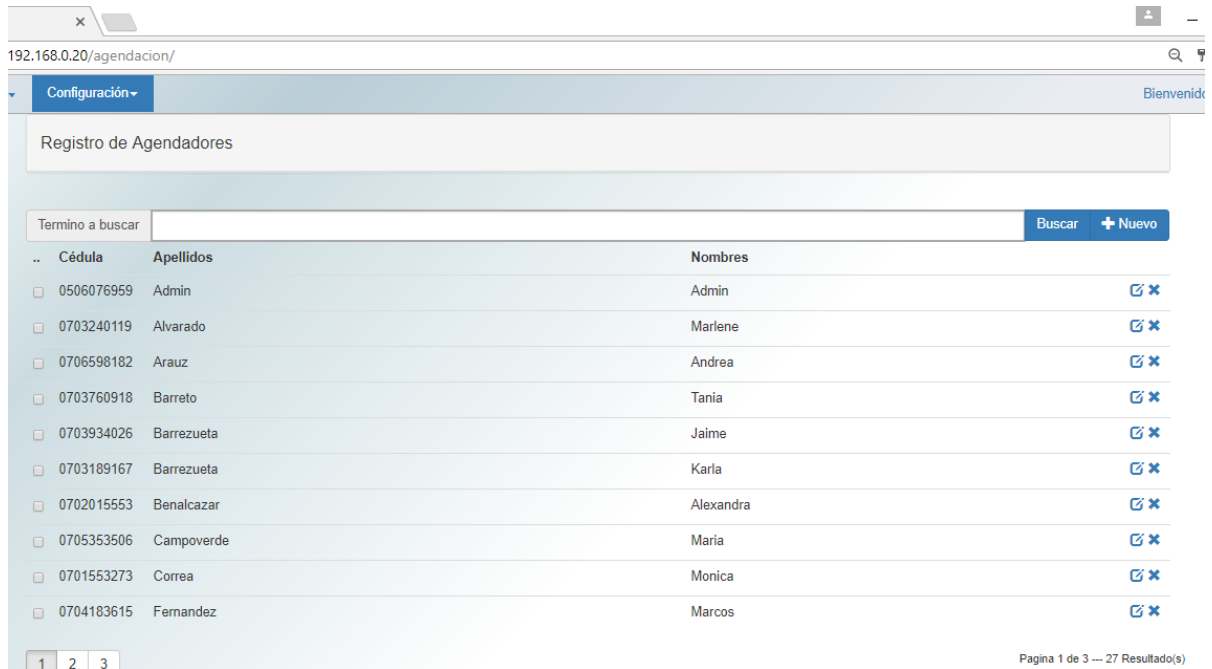


Figura 69: Vista agendador

Fuente: El autor

Elaborado por: El autor

```

from django.db.models.query_utils import Q
from django.db.utils import IntegrityError
from django.http.response import JsonResponse
from django.shortcuts import render, get_object_or_404

from agendacion import md5, combine_arguments
from agendacion.models import Ciudad, Usuario
from ..models import Agendador

#
# Carga la vista principal de la gestion Agendador
#
def index(request):
    return render(request, "agendador/index.html")

#
# Carga el listado de agendadores segun un parametro de busqueda
#
def list(request):

```

```

page = int(request.GET.get('page', 1))
query = request.GET.get('query', "").encode("utf_8")

inicio = (page - 1) * 10
fin = page * 10

queryset = Agendador.objects.filter(combine_arguments('usuario__apellidos__icontains',
'usuario__nombres__icontains', query) | Q(usuario__cedula__icontains=query))
count = queryset.count()
agendador_list = queryset.order_by('usuario__apellidos')[inicio:fin]

total_pages = count / 10 + 1

return render(request, "agendador/list.html", {'agendador_list': agendador_list, 'query':query,
'count': count, 'range_pages': range(total_pages), 'page': page})
#
# Muestra un formulario para la creacion de un registro de Agendador
#
def new(request):
    ciudad_list = Ciudad.objects.all().order_by('nombre')
    return render(request, "agendador/form.html", {'agendador': {}, 'ciudad_list': ciudad_list,
'is_edit': 1})
#
# Muestra un formulario para el edicion de un registro de Agendador segun codigo de
agendador proporcionado
#
def edit(request, agendador_id):
    agendador = get_object_or_404(Agendador, pk=agendador_id)
    ciudad_list = Ciudad.objects.all().order_by('nombre')
    return render(request, "agendador/form.html", {'agendador': agendador, 'ciudad_list':
ciudad_list, 'is_edit': 2})

#
# Elimina un registro de Agendador segun codigo de agendador proporcionado
#
def delete(request, agendador_id):
    agendador = get_object_or_404(Agendador, pk=agendador_id)

    try:
        if(request.GET['confirm'] == 'true'):
            agendador.usuario.delete()
            agendador.delete()
            return JsonResponse({'success': True})
        else:
            return JsonResponse({'success': False, 'msg': 'No se ha confirmado la eliminacion'})
    except:
        return JsonResponse({'success': False, 'msg': 'Registro no pudo ser eliminado'})

#
# Guarda o modifica un registro de Agendador segun datos recibidos

```



```

#
def save(request):
    if len(request.POST) > 0:
        is_edit = request.POST['is_edit']

        if is_edit == '1':
            agendador = Agendador()
            agendador.usuario = Usuario()
            agendador.usuario.password = md5(request.POST['cedula'].encode("utf_8"))
        else:
            agendador = get_object_or_404(Agendador, pk= int(request.POST['codigo']) )

        try:
            agendador.usuario.cedula = request.POST['cedula'].encode("utf_8")
            agendador.usuario.nombres = request.POST['nombres'].encode("utf_8")
            agendador.usuario.apellidos = request.POST['apellidos'].encode("utf_8")
            agendador.usuario.login = request.POST['login'].encode("utf_8")
            agendador.usuario.ciudad =
Ciudad.objects.get(pk=request.POST['ciudad'].encode("utf_8"))
            agendador.usuario.tipo_usuario = "a"
            agendador.usuario.save()
            agendador.usuario_id = agendador.usuario.pk
            agendador.save()
            return JsonResponse({'success': True})
        except IntegrityError as e:
            return JsonResponse({'success': False, 'msg': 'Cedula o login se encuentra registrada'})
        except:
            return JsonResponse({'success': False, 'msg': 'Hubo un error al guardar'})
    else:
        return JsonResponse({'success': False, 'msg': 'No existen datos'})

```

1.2.5 ciudad.py

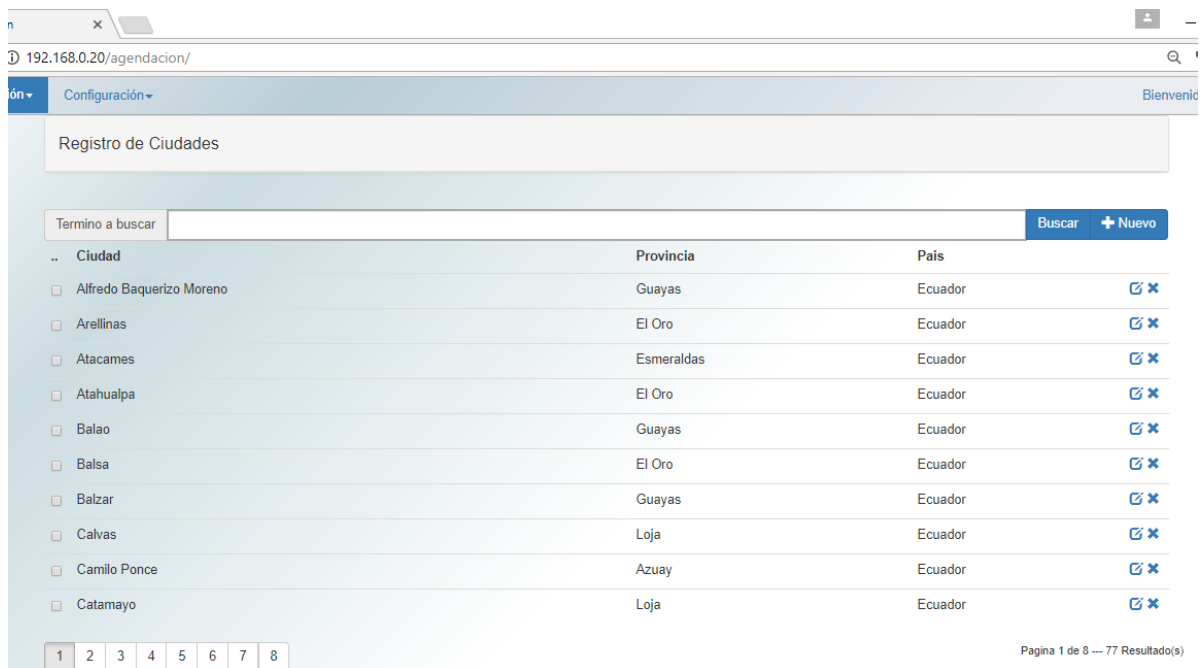


Figura 70: Vista ciudad
Fuente: El autor
Elaborado por: El autor

```

from json.encoder import JSONEncoder

from django.db.models.query_utils import Q
from django.db.utils import IntegrityError
from django.http.response import JsonResponse
from django.shortcuts import render, get_object_or_404

from ..models import Ciudad, Provincia
#
# Carga la vista principal de la gestion de Ciudad
#
def index(request):
    return render(request, "ciudad/index.html")
#
# Carga el listado de ciudades segun un parametro de busqueda
#
def list(request):
    page = int(request.GET.get('page', 1))
    query = request.GET.get('query', "").encode("utf_8")

    inicio = (page - 1) * 10
    fin = page * 10

    queryset = Ciudad.objects.filter(nombre__icontains=query)
    count = queryset.count()

```

```

ciudad_list = queryset.order_by('nombre')[inicio:fin]

total_pages = count / 10 + 1

return render(request, "ciudad/list.html", {'ciudad_list': ciudad_list, 'query': query, 'count': count,
'range_pages': range(total_pages), 'page': page})

#
# Carga el listado de ciudades en formato JSON segun un parametro de busqueda
#
def search(request):
    query = request.GET.get('query', "").encode("utf_8")

    ciudad_filter = Ciudad.objects.filter(Q(nombre__icontains=query) |
Q(provincia__nombre__icontains=query) |
Q(provincia__pais__nombre__icontains=query)).order_by('nombre')[0:20]

    ciudad_list = []

    for c in ciudad_filter:
        ciudad_list.append({
            "nombre": c.nombre,
            "provincia_nombre": c.provincia.nombre,
            "pais_nombre": c.provincia.pais.nombre,
            "codigo": c.codigo
        })

    return JsonResponse(ciudad_list, encoder=JSONEncoder, safe=False)

#
# Muestra un formulario para la creacion de un registro de Ciudad
#
def new(request):
    provincia_list = Provincia.objects.all().order_by('nombre')
    return render(request, "ciudad/form.html", {'ciudad': {}, 'provincia_list': provincia_list, 'is_edit':
1})

#
# Muestra un formulario para el edicion de un registro de Ciudad segun codigo de ciudad
proporcionado
#
def edit(request, ciudad_id):
    ciudad = get_object_or_404(Ciudad, pk=ciudad_id)
    provincia_list = Provincia.objects.all().order_by('nombre')
    return render(request, "ciudad/form.html", {'ciudad': ciudad, 'provincia_list': provincia_list,
'is_edit': 2})

#
# Elimina un registro de Ciudad segun codigo de ciudad proporcionado

```

```

#
def delete(request, ciudad_id):
    ciudad = get_object_or_404(Ciudad, pk=ciudad_id)

    try:
        if(request.GET['confirm'] == 'true'):
            ciudad.delete()
            return JsonResponse({'success': True})
        else:
            return JsonResponse({'success': False, 'msg': 'No se ha confirmado la eliminacion'})
    except:
        return JsonResponse({'success': False, 'msg': 'Registro no pudo ser eliminado'})

#
# Guarda o modifica un registro de Ciudad segun datos recibidos
#
def save(request):
    if len(request.POST) > 0:
        is_edit = request.POST['is_edit']

        if is_edit == '1':
            ciudad = Ciudad()
        else:
            ciudad = get_object_or_404(Ciudad, pk= int(request.POST['codigo']) )

        try:
            ciudad.provincia = Provincia.objects.get(pk=request.POST['provincia'].encode("utf_8"))
            ciudad.nombre = request.POST['nombre'].encode("utf_8")
            ciudad.save()

            return JsonResponse({'success': True})

        except IntegrityError as e:
            return JsonResponse({'success': False, 'msg': 'Registro ya existe'})

        except:
            return JsonResponse({'success': False, 'msg': 'Registro no pudo ser guardado'})

    else:
        return JsonResponse({'success': False, 'msg': 'No existen datos'})

```

1.2.6 consultorio.py

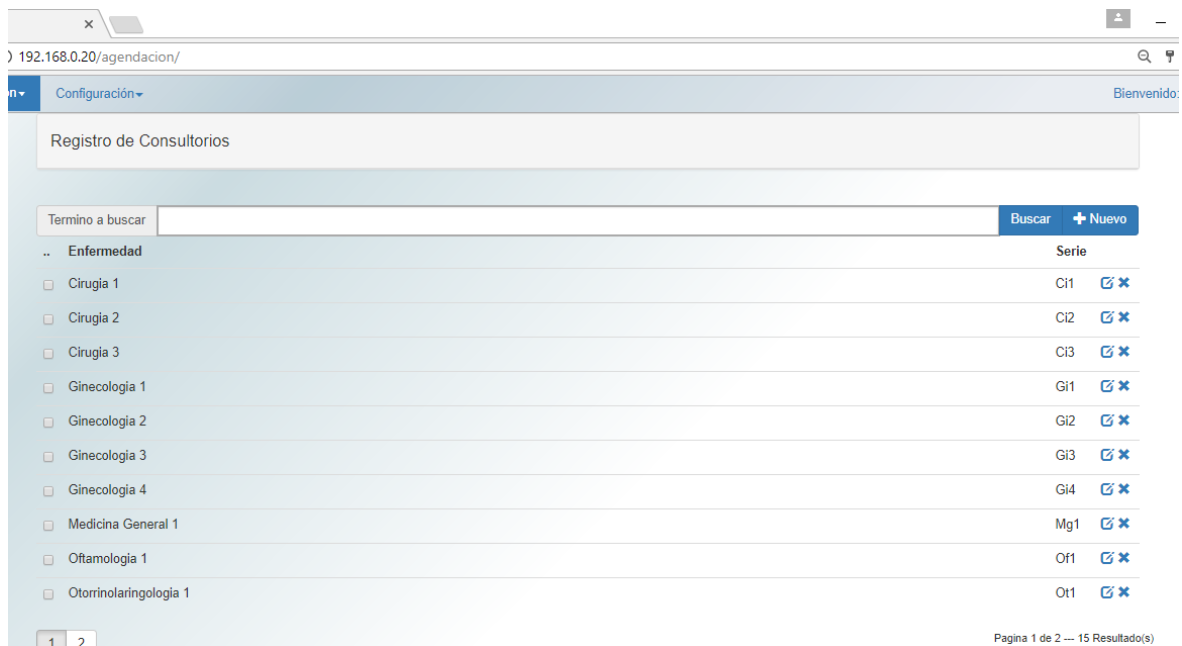


Figura 71: Vista consultorio

Fuente: El autor

Elaborado por: El autor

```

from django.db.utils import IntegrityError
from django.http.response import JsonResponse
from django.shortcuts import render, get_object_or_404

from ..models import Consultorio

#
# Carga la vista principal de la gestion de Consultorio
#
def index(request):
    return render(request, "consultorio/index.html")

#
# Carga el listado de consultorios segun un parametro de busqueda
#
def list(request):
    page = int(request.GET.get('page', 1))
    query = request.GET.get('query', "").encode("utf_8")

    inicio = (page - 1) * 10
    fin = page * 10

    queryset = Consultorio.objects.filter(nombre__icontains=query)

```

```

count = queryset.count()
consultorio_list = queryset.order_by('nombre')[inicio:fin]

total_pages = count / 10 + 1

return render(request, "consultorio/list.html", {'consultorio_list': consultorio_list, 'query':query,
'count': count, 'range_pages': range(total_pages), 'page': page})

#
# Muestra un formulario para la creacion de un registro de Consultorio
#
def new(request):
    return render(request, "consultorio/form.html", {'consultorio': {}, 'is_edit': 1})

#
# Muestra un formulario para el edicion de un registro de Consultorio segun codigo de
consultorio proporcionado
#
def edit(request, consultorio_id):
    consultorio = get_object_or_404(Consultorio, pk=consultorio_id)
    return render(request, "consultorio/form.html", {'consultorio': consultorio, 'is_edit': 2})

#
# Elimina un registro de Consultorio segun codigo de consultorio proporcionado
#
def delete(request, consultorio_id):
    consultorio = get_object_or_404(Consultorio, pk=consultorio_id)

    try:
        if(request.GET['confirm'] == 'true'):
            consultorio.delete()
            return JsonResponse({'success': True})
        else:
            return JsonResponse({'success': False, 'msg': 'No se ha confirmado la eliminacion'})
    except:
        return JsonResponse({'success': False, 'msg': 'Registro no pudo ser eliminado'})

#
# Guarda o modifica un registro de Consultorio segun datos recibidos
#
def save(request):
    if len(request.POST) > 0:
        is_edit = request.POST['is_edit']

        if is_edit == '1':
            consultorio = Consultorio()
        else:

```

```

consultorio = get_object_or_404(Consultorio, pk= int(request.POST['codigo']))

try:
    consultorio.nombre = request.POST['nombre'].encode("utf_8")
    consultorio.serie = request.POST['serie'].encode("utf_8")
    consultorio.save()

    return JsonResponse({'success': True})

except IntegrityError as e:
    return JsonResponse({'success': False, 'msg': 'Serie o nombre se encuentra registrada'})

except:
    return JsonResponse({'success': False, 'msg': 'Registro no pudo ser guardado'})

else:
    return JsonResponse({'success': False, 'msg': 'No existen datos'})

```

1.2.7 enfermedad.py

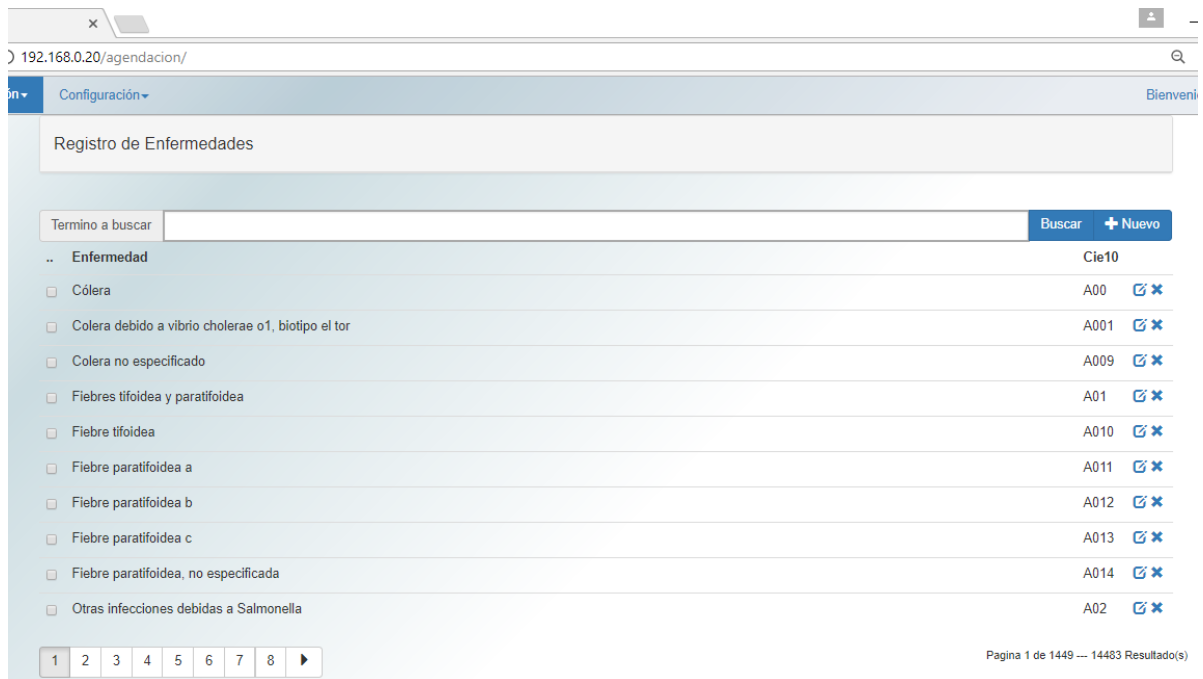


Figura 72: Vista enfermedad

Fuente: El autor

Elaborado por: El autor

```
from json.encoder import JSONEncoder
```

```
from django.db.models.query_utils import Q
from django.db.utils import IntegrityError
```

```

from django.http.response import JsonResponse
from django.shortcuts import render, get_object_or_404

from ..models import Enfermedad

#
# Carga la vista principal de la gestion de Enfermedad
#
def index(request):
    return render(request, "enfermedad/index.html")

#
# Carga el listado de enfermedades segun un parametro de busqueda
#
def list(request):
    page = int(request.GET.get('page', 1))
    query = request.GET.get('query', "").encode("utf_8")

    inicio = (page - 1) * 10
    fin = page * 10

    queryset = Enfermedad.objects.filter(nombre__icontains=query)
    count = queryset.count()
    enfermedad_list = queryset.order_by('cie10')[inicio:fin]

    total_pages = count / 10 + 1

    return render(request, "enfermedad/list.html", {'enfermedad_list': enfermedad_list,
'query':query, 'count': count, 'range_pages': range(total_pages), 'page': page})

#
# Carga el listado de enfermedades en formato JSON segun un parametro de busqueda
#
def search(request):
    query = request.GET.get('query', "").encode("utf_8")

    enfermedad_filter = Enfermedad.objects.filter(Q(nombre__icontains=query) |
Q(cie10__icontains=query)).order_by('cie10')[0:20]

    enfermedad_list = []

    for e in enfermedad_filter:
        enfermedad_list.append({
            "cie10": e.cie10,
            "nombre": e.nombre,
            "codigo": e.codigo
        })

    return JsonResponse(enfermedad_list, encoder=JSONEncoder, safe=False)

```



```

#
# Muestra un formulario para la creacion de un registro de Enfermedad
#
def new(request):
    return render(request, "enfermedad/form.html", {'enfermedad': {}, 'is_edit': 1})

#
# Muestra un formulario para el edicion de un registro de Enfermedad segun codigo de
enfermedad proporcionado
#
def edit(request, enfermedad_id):
    enfermedad = get_object_or_404(Enfermedad, pk=enfermedad_id)
    return render(request, "enfermedad/form.html", {'enfermedad': enfermedad, 'is_edit': 2})

#
# Elimina un registro de Enfermedad segun codigo de enfermedad proporcionado
#
def delete(request, enfermedad_id):
    enfermedad = get_object_or_404(Enfermedad, pk=enfermedad_id)

    try:
        if(request.GET['confirm'] == 'true'):
            enfermedad.delete()
            return JsonResponse({'success': True})
        else:
            return JsonResponse({'success': False, 'msg': 'No se ha confirmado la eliminacion'})
    except:
        return JsonResponse({'success': False, 'msg': 'Registro no pudo ser eliminado'})

#
# Guarda o modifica un registro de Enfermedad segun datos recibidos
#
def save(request):
    if len(request.POST) > 0:
        is_edit = request.POST['is_edit']

        if is_edit == '1':
            enfermedad = Enfermedad()
        else:
            enfermedad = get_object_or_404(Enfermedad, pk= int(request.POST['codigo'])) )

    try:
        enfermedad.nombre = request.POST['nombre'].encode("utf_8")
        enfermedad.cie10 = request.POST['cie10'].encode("utf_8")
        enfermedad.save()

```

```

return JsonResponse({'success': True})

except IntegrityError as e:
    return JsonResponse({'success': False, 'msg': 'Nombre o CIE10 ya se encuentra
registrado'})

except:
    return JsonResponse({'success': False, 'msg': 'Registro no pudo ser guardado'})

else:
    return JsonResponse({'success': False, 'msg': 'No existen datos'})

```

1.2.8 especialidad.py

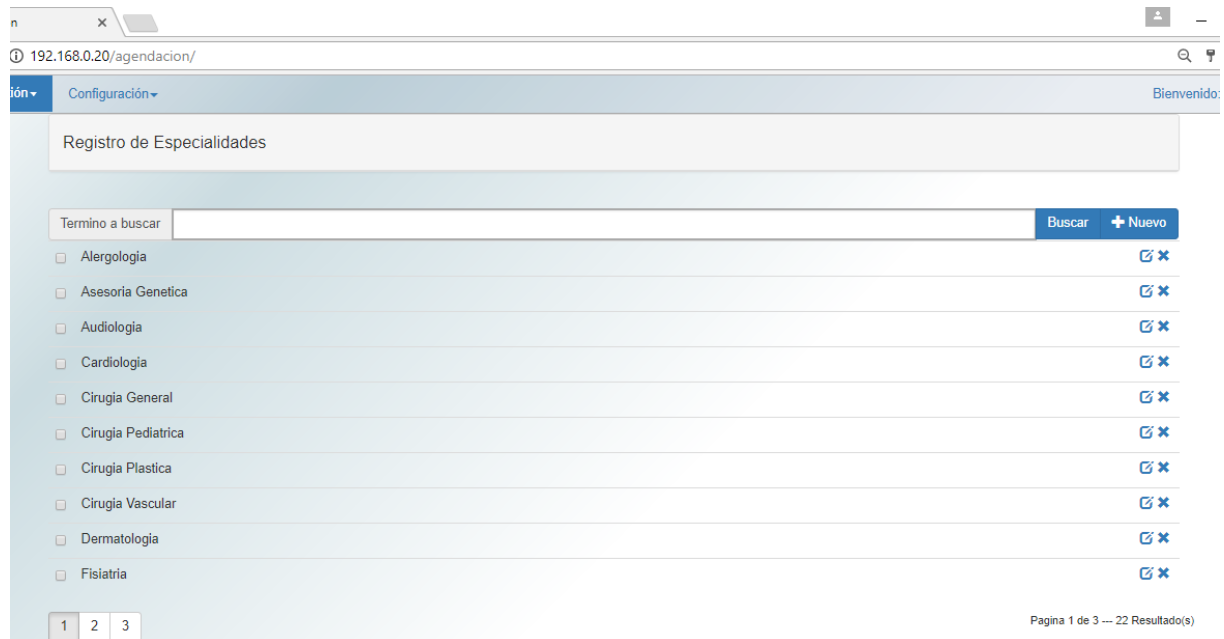


Figura 73: Vista especialidad

Fuente: El autor

Elaborado por: El autor

```

from django.db.utils import IntegrityError
from django.http.response import JsonResponse
from django.shortcuts import render, get_object_or_404

from ..models import Especialidad

#
# Carga la vista principal de la gestion de Especialidad
#
def index(request):

```

```

return render(request, "especialidad/index.html")

#
# Carga el listado de especialidades segun un parametro de busqueda
#
def list(request):
    page = int(request.GET.get('page', 1))
    query = request.GET.get('query', "").encode("utf_8")

    inicio = (page - 1) * 10
    fin = page * 10

    queryset = Especialidad.objects.filter(nombre__icontains=query)
    count = queryset.count()
    especialidad_list = queryset.order_by('nombre')[inicio:fin]

    total_pages = count / 10 + 1

    return render(request, "especialidad/list.html", {'especialidad_list': especialidad_list,
'query':query, 'count': count, 'range_pages': range(total_pages), 'page': page})

#
# Muestra un formulario para la creacion de un registro de Especialidad
#
def new(request):
    return render(request, "especialidad/form.html", {'especialidad': {}, 'is_edit': 1})

#
# Muestra un formulario para el edicion de un registro de Especialidad segun codigo de
especialidad proporcionado
#
def edit(request, especialidad_id):
    especialidad = get_object_or_404(Especialidad, pk=especialidad_id)
    return render(request, "especialidad/form.html", {'especialidad': especialidad, 'is_edit': 2})

#
# Elimina un registro de Especialidad segun codigo de especialidad proporcionado
#
def delete(request, especialidad_id):
    especialidad = get_object_or_404(Especialidad, pk=especialidad_id)

    try:
        if(request.GET['confirm'] == 'true'):
            especialidad.delete()
            return JsonResponse({'success': True})
        else:
            return JsonResponse({'success': False, 'msg': 'No se ha confirmado la eliminacion'})

```

```

except:
    return JsonResponse({'success': False, 'msg': 'Registro no pudo ser eliminado'})

#
# Guarda o modifica un registro de Especialidad segun datos recibidos
#
def save(request):
    if len(request.POST) > 0:
        is_edit = request.POST['is_edit']

        if is_edit == '1':
            especialidad = Especialidad()
        else:
            especialidad = get_object_or_404(Especialidad, pk= int(request.POST['codigo']) )

        try:
            especialidad.nombre = request.POST['nombre'].encode("utf_8")
            especialidad.save()

            return JsonResponse({'success': True})

        except IntegrityError as e:
            return JsonResponse({'success': False, 'msg': 'Registro ya existe'})

    except:
        return JsonResponse({'success': False, 'msg': 'Registro no pudo ser guardado'})

else:
    return JsonResponse({'success': False, 'msg': 'No existen datos'})

```

1.2.9 grupo_cultural.py

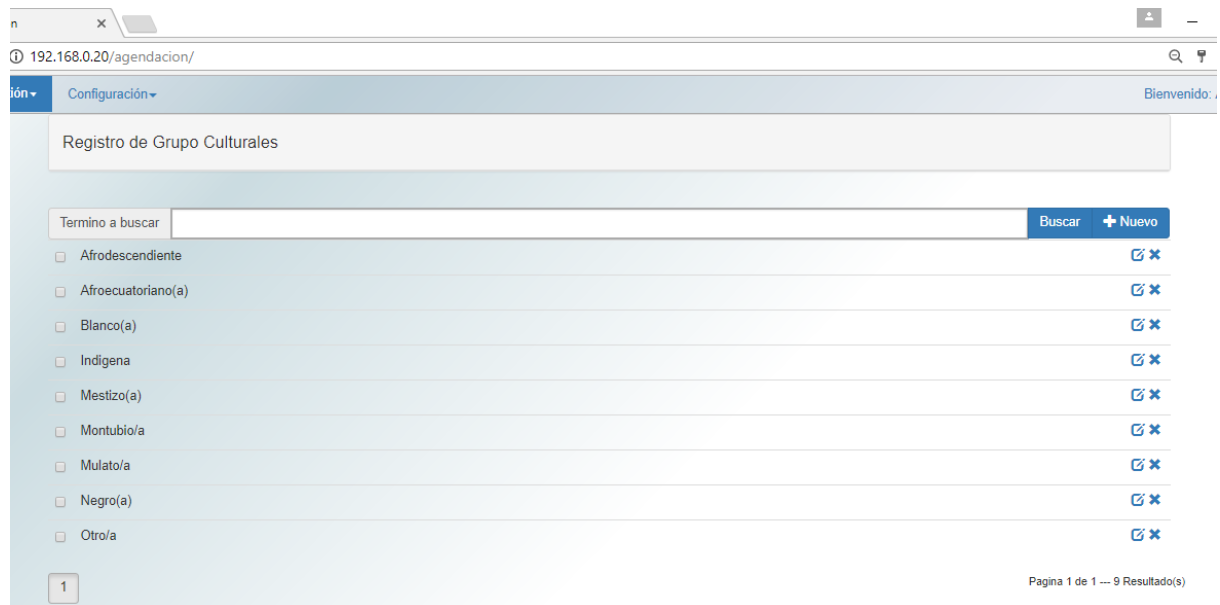


Figura 74: Vista grupo cultural

Fuente: El autor

Elaborado por: El autor

```
from django.db.utils import IntegrityError
from django.http.response import JsonResponse
from django.shortcuts import render, get_object_or_404

from ..models import Grupo_Cultural

#
# Carga la vista principal de la gestion de Grupo_Cultural
#
def index(request):
    return render(request, "grupo_cultural/index.html")

#
# Carga el listado de grupos culturales segun un parametro de busqueda
#
def list(request):
    page = int(request.GET.get('page', 1))
    query = request.GET.get('query', "").encode("utf_8")

    inicio = (page - 1) * 10
    fin = page * 10

    queryset = Grupo_Cultural.objects.filter(nombre__icontains=query)
```

```

count = queryset.count()
grupo_cultural_list = queryset.order_by('nombre')[inicio:fin]

total_pages = count / 10 + 1

return render(request, "grupo_cultural/list.html", {'grupo_cultural_list': grupo_cultural_list,
'query':query, 'count': count, 'range_pages': range(total_pages), 'page': page})

#
# Muestra un formulario para la creacion de un registro de Grupo_Cultural
#
def new(request):
    return render(request, "grupo_cultural/form.html", {'grupo_cultural': {}, 'is_edit': 1})

#
# Muestra un formulario para el edicion de un registro de Grupo_Cultural segun codigo de
grupo cultural proporcionado
#
def edit(request, grupo_cultural_id):
    grupo_cultural = get_object_or_404(Grupo_Cultural, pk=grupo_cultural_id)
    return render(request, "grupo_cultural/form.html", {'grupo_cultural': grupo_cultural, 'is_edit': 2})

#
# Elimina un registro de Grupo_Cultural segun codigo de grupo cultural proporcionado
#
def delete(request, grupo_cultural_id):
    grupo_cultural = get_object_or_404(Grupo_Cultural, pk=grupo_cultural_id)

    try:
        if(request.GET['confirm'] == 'true'):
            grupo_cultural.delete()
            return JsonResponse({'success': True})
        else:
            return JsonResponse({'success': False, 'msg': 'No se ha confirmado la eliminacion'})
    except:
        return JsonResponse({'success': False, 'msg': 'Registro no pudo ser eliminado'})

#
# Guarda o modifica un registro de Grupo_Cultural segun datos recibidos
#
def save(request):
    if len(request.POST) > 0:
        is_edit = request.POST['is_edit']

        if is_edit == '1':
            grupo_cultural = Grupo_Cultural()
        else:

```

```

grupo_cultural = get_object_or_404(Grupo_Cultural, pk= int(request.POST['codigo']))

try:
    grupo_cultural.nombre = request.POST['nombre'].encode("utf_8")
    grupo_cultural.save()

    return JsonResponse({'success': True})

except IntegrityError as e:
    return JsonResponse({'success': False, 'msg': 'Registro ya existe'})

except:
    return JsonResponse({'success': False, 'msg': 'Registro no pudo ser guardado'})

else:
    return JsonResponse({'success': False, 'msg': 'No existen datos'})

```

1.2.10 instruccion_educativa.py

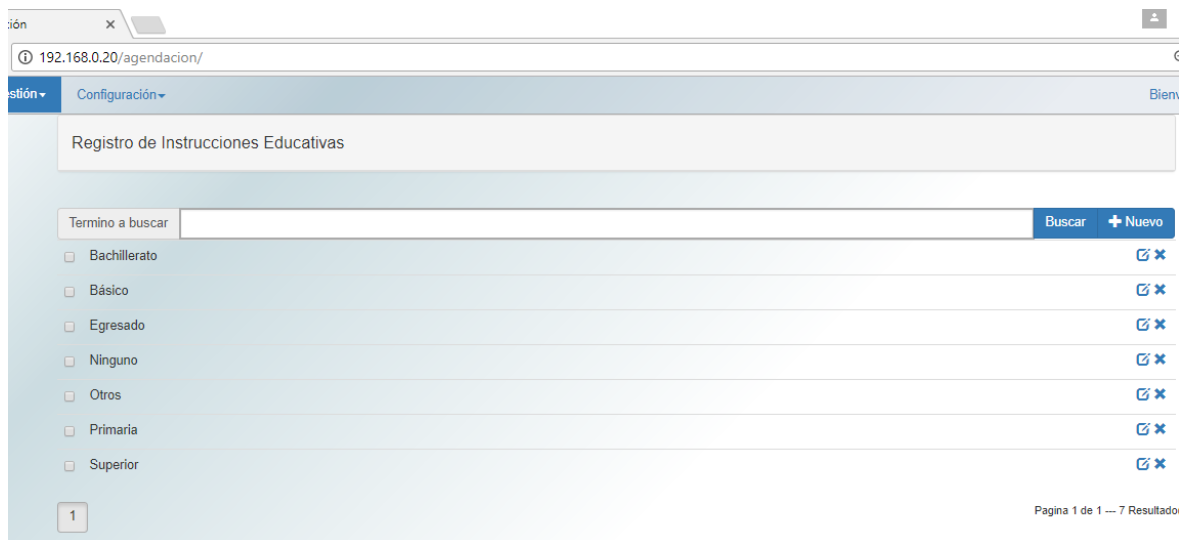


Figura 75: Vista instrucción educativa

Fuente: El autor

Elaborado por: El autor

```

from django.db.utils import IntegrityError
from django.http.response import JsonResponse
from django.shortcuts import render, get_object_or_404

from ..models import Instruccion_Educativa

#
# Carga la vista principal de la gestion de Instruccion_Educativa

```

```

#
def index(request):
    return render(request, "instruccion_educativa/index.html")

#
# Carga el listado de instrucciones educativas segun un parametro de busqueda
#
def list(request):
    page = int(request.GET.get('page', 1))
    query = request.GET.get('query', "").encode("utf_8")

    inicio = (page - 1) * 10
    fin = page * 10

    queryset = Instruccion_Educativa.objects.filter(nombre__icontains=query)
    count = queryset.count()
    instruccion_educativa_list = queryset.order_by('nombre')[inicio:fin]

    total_pages = count / 10 + 1

    return render(request, "instruccion_educativa/list.html", {'instruccion_educativa_list':
instruccion_educativa_list, 'query':query, 'count': count, 'range_pages': range(total_pages),
'page': page})

#
# Muestra un formulario para la creacion de un registro de Instruccion_Educativa
#
def new(request):
    return render(request, "instruccion_educativa/form.html", {'instruccion_educativa': {}, 'is_edit':
1})

#
# Muestra un formulario para el edicion de un registro de Instruccion_Educativa segun codigo
de instruccion educativa proporcionado
#
def edit(request, instruccion_educativa_id):
    instruccion_educativa = get_object_or_404(Instruccion_Educativa,
pk=instruccion_educativa_id)
    return render(request, "instruccion_educativa/form.html", {'instruccion_educativa':
instruccion_educativa, 'is_edit': 2})

#
# Elimina un registro de Instruccion_Educativa segun codigo de instruccion educativa
proporcionado
#
def delete(request, instruccion_educativa_id):

```



```

instruccion_educativa = get_object_or_404(Instruccion_Educativa,
pk=instruccion_educativa_id)

try:
    if(request.GET['confirm'] == 'true'):
        instruccion_educativa.delete()
        return JsonResponse({'success': True})
    else:
        return JsonResponse({'success': False, 'msg': 'No se ha confirmado la eliminacion'})
except:
    return JsonResponse({'success': False, 'msg': 'Registro no pudo ser eliminado'})

#
# Guarda o modifica un registro de Instruccion_Educativa segun datos recibidos
#
def save(request):
    if len(request.POST) > 0:
        is_edit = request.POST['is_edit']

        if is_edit == '1':
            instruccion_educativa = Instruccion_Educativa()
        else:
            instruccion_educativa = get_object_or_404(Instruccion_Educativa, pk=
int(request.POST['codigo']) )

        try:
            instruccion_educativa.nombre = request.POST['nombre'].encode("utf_8")
            instruccion_educativa.save()

            return JsonResponse({'success': True})

        except IntegrityError as e:
            return JsonResponse({'success': False, 'msg': 'Registro ya existe'})

        except:
            return JsonResponse({'success': False, 'msg': 'Registro no pudo ser guardado'})

    else:
        return JsonResponse({'success': False, 'msg': 'No existen datos'})

```

1.2.11 medico.py

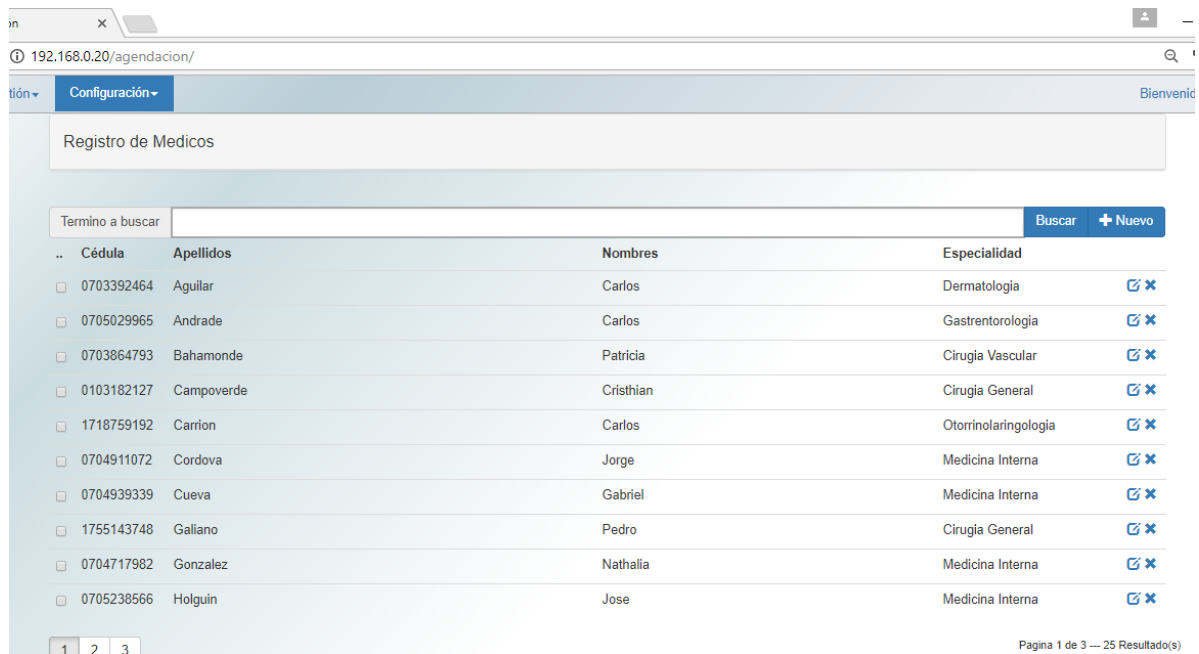


Figura 76: Vista médico

Fuente: El autor

Elaborado por: El autor

```
from json.encoder import JSONEncoder

from django.db.models.query_utils import Q
from django.db.utils import IntegrityError
from django.http.response import JsonResponse
from django.shortcuts import render, get_object_or_404

from agendacion import md5, combine_arguments
from agendacion.models import Ciudad, Usuario, Especialidad, Medico

#
# Carga la vista principal de la gestion de Medico
#
def index(request):
    return render(request, "medico/index.html")

#
# Carga el listado de medicos segun un parametro de busqueda
#
def list(request):
    page = int(request.GET.get('page', 1))
    query = request.GET.get('query', "").encode("utf_8")
```

```

inicio = (page - 1) * 10
fin = page * 10

queryset = Medico.objects.filter(combine_arguments('usuario__apellidos__icontains',
'usuario__nombres__icontains', query) | Q(usuario__cedula__icontains=query))
count = queryset.count()
medico_list = queryset.order_by('usuario__apellidos')[inicio:fin]

total_pages = count / 10 + 1

return render(request, "medico/list.html", {'medico_list': medico_list, 'query':query, 'count':
count, 'range_pages': range(total_pages), 'page': page})

#
# Carga el listado de medicos en formato JSON segun un parametro de busqueda
#
def search(request):
    query = request.GET.get('query', "").encode("utf_8")

    medico_filter = Medico.objects.filter(combine_arguments('usuario__apellidos__icontains',
'usuario__nombres__icontains', query) |
Q(usuario__cedula__icontains=query)).order_by('usuario__apellidos')[0:20]

    medico_list = []

    for m in medico_filter:
        medico_list.append({
            "cedula": m.usuario.cedula,
            "nombres": m.usuario.nombres,
            "apellidos": m.usuario.apellidos,
            "codigo": m.codigo
        })

    return JsonResponse(medico_list, encoder=JSONEncoder, safe=False)

#
# Muestra un formulario para la creacion de un registro de Medico
#
def new(request):
    ciudad_list = Ciudad.objects.all().order_by('nombre')
    especialidad_list = Especialidad.objects.all().order_by('nombre')

    return render(request, "medico/form.html", {'medico': {}, 'ciudad_list': ciudad_list,
'especialidad_list': especialidad_list, 'is_edit': 1})

#
# Muestra un formulario para el edicion de un registro de Medico segun codigo de medico

```

```

proporcionado
#
def edit(request, medico_id):
    medico = get_object_or_404(Medico, pk=medico_id)

    ciudad_list = Ciudad.objects.all().order_by('nombre')
    especialidad_list = Especialidad.objects.all().order_by('nombre')

    return render(request, "medico/form.html", {'medico': medico, 'ciudad_list': ciudad_list,
'especialidad_list': especialidad_list, 'is_edit': 2})

#
# Elimina un registro de Medico segun codigo de medico proporcionado
#
def delete(request, medico_id):
    medico = get_object_or_404(Medico, pk=medico_id)

    try:
        if(request.GET['confirm'] == 'true'):
            medico.usuario.delete()
            medico.delete()
            return JsonResponse({'success': True})
        else:
            return JsonResponse({'success': False, 'msg': 'No se ha confirmado la eliminacion'})
    except:
        return JsonResponse({'success': False, 'msg': 'Registro no pudo ser eliminado'})

#
# Guarda o modifica un registro de Medico segun datos recibidos
#
def save(request):
    if len(request.POST) > 0:
        is_edit = request.POST['is_edit']

        if is_edit == '1':
            medico = Medico()
            medico.usuario = Usuario()
            medico.usuario.password = md5(request.POST['cedula'].encode("utf_8"))
        else:
            medico = get_object_or_404(Medico, pk= int(request.POST['codigo']) )

    try:
        medico.usuario.cedula = request.POST['cedula'].encode("utf_8")
        medico.usuario.nombres = request.POST['nombres'].encode("utf_8")
        medico.usuario.apellidos = request.POST['apellidos'].encode("utf_8")
        medico.usuario.login = request.POST['login'].encode("utf_8")
        medico.usuario.ciudad = Ciudad.objects.get(pk=request.POST['ciudad'].encode("utf_8"))
        medico.usuario.tipo_usuario = "m"

```

```

medico.usuario.save()

medico.especialidad =
Especialidad.objects.get(pk=request.POST['especialidad'].encode("utf_8"))
medico.usuario_id = medico.usuario.pk
medico.save()

return JsonResponse({'success': True})

except IntegrityError as e:
    return JsonResponse({'success': False, 'msg': 'Cedula o login se encuentra registrada'})

except:
    return JsonResponse({'success': False, 'msg': 'Hubo un error al guardar'})

else:
    return JsonResponse({'success': False, 'msg': 'No existen datos'})

```

1.2.12 pais.py

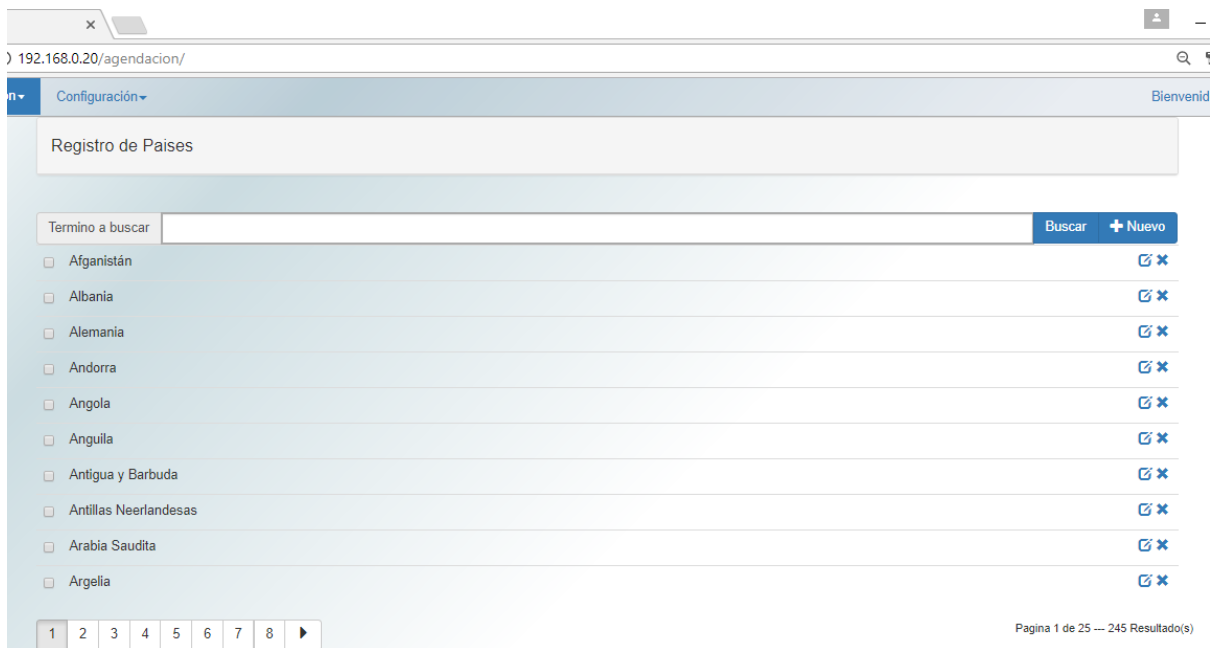


Figura 77: Vista país
Fuente: El autor
Elaborado por: El autor

```

from json.encoder import JSONEncoder

from django.db.utils import IntegrityError
from django.http.response import JsonResponse

```

```

from django.shortcuts import render, get_object_or_404

from ..models import Pais

#
# Carga la vista principal de la gestion de Pais
#
def index(request):
    return render(request, "pais/index.html")

#
# Carga el listado de paises segun un parametro de busqueda
#
def list(request):
    page = int(request.GET.get('page', 1))
    query = request.GET.get('query', "").encode("utf_8")

    inicio = (page - 1) * 10
    fin = page * 10

    queryset = Pais.objects.filter(nombre__icontains=query)
    count = queryset.count()
    pais_list = queryset.order_by('nombre')[inicio:fin]

    total_pages = count / 10 + 1

    return render(request, "pais/list.html", {'pais_list': pais_list, 'query': query, 'count': count,
'range_pages': range(total_pages), 'page': page})

#
# Carga el listado de paises en formato JSON segun un parametro de busqueda
#
def search(request):
    query = request.GET.get('query', "").encode("utf_8")

    pais_filter = Pais.objects.filter(nombre__icontains=query).order_by('nombre')[0:20]

    pais_list = []

    for p in pais_filter:
        pais_list.append({
            "nombre": p.nombre,
            "codigo": p.codigo
        })

    return JsonResponse(pais_list, encoder=JSONEncoder, safe=False)

```

```

#
# Muestra un formulario para la creacion de un registro de Pais
#
def new(request):
    return render(request, "pais/form.html", {'pais': {}, 'is_edit': 1})

#
# Muestra un formulario para el edicion de un registro de Pais segun codigo de pais
proporcionado
#
def edit(request, pais_id):
    pais = get_object_or_404(Pais, pk=pais_id)
    return render(request, "pais/form.html", {'pais': pais, 'is_edit': 2})

#
# Elimina un registro de Pais segun codigo de pais proporcionado
#
def delete(request, pais_id):
    pais = get_object_or_404(Pais, pk=pais_id)

    try:
        if(request.GET['confirm'] == 'true'):
            pais.delete()
            return JsonResponse({'success': True})
        else:
            return JsonResponse({'success': False, 'msg': 'No se ha confirmado la eliminacion'})
    except:
        return JsonResponse({'success': False, 'msg': 'Registro no pudo ser eliminado'})

#
# Guarda o modifica un registro de Pais segun datos recibidos
#
def save(request):
    if len(request.POST) > 0:
        is_edit = request.POST['is_edit']

        if is_edit == '1':
            pais = Pais()
        else:
            pais = get_object_or_404(Pais, pk= int(request.POST['codigo']))

        try:
            pais.nombre = request.POST['nombre'].encode("utf_8")
            pais.save()

            return JsonResponse({'success': True})

```

```

except IntegrityError as e:
    return JsonResponse({'success': False, 'msg': 'Registro ya existe'})

except:
    return JsonResponse({'success': False, 'msg': 'Registro no pudo ser guardado'})

else:
    return JsonResponse({'success': False, 'msg': 'No existen datos'})

```

1.2.13 report.py

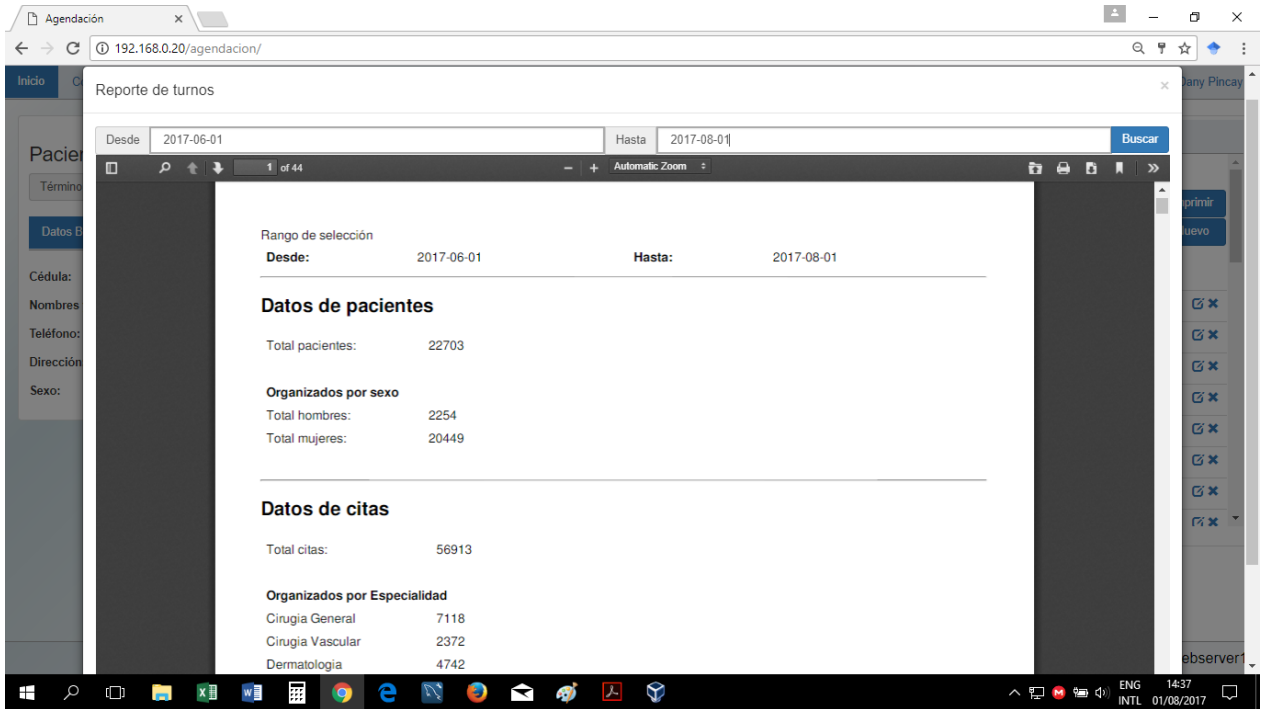


Figura 78: Reporte citas y pacientes

Fuente: El autor

Elaborado por: El autor

```

from django.db.models.aggregates import Count
from django.http.response import HttpResponseRedirect
from django.shortcuts import get_object_or_404, render
from django.template.loader import render_to_string
import pdfkit

```

```

from agendacion.models import Paciente, Turno, Historia_Clinica

```

```

#

```

```

# Muestra un archivo pdf en el que se carga la informacion de un paciente, asi como de las citas agendadas segun cedula

```



```

# del paciente
#
def reporte_form001(request):
    cedula = request.GET.get('cedula', "").encode("utf_8")

    try:
        paciente = get_object_or_404(Paciente, cedula=cedula)
        pariente_list = paciente.parientes.all()
        turno_list = Turno.objects.filter(paciente__cedula=cedula).order_by('-fecha')

        html_string = render_to_string('report/form001.html', {'paciente': paciente, 'pariente_list':
pariente_list, 'turno_list': turno_list})

    except:
        html_string = ""

    pdf = pdfkit.from_string(html_string, False)

    response = HttpResponse(pdf, content_type='application/pdf')
    response['Content-Disposition'] = 'attachment; filename="mypdf.pdf"'
    return response

#
# Muestra un formulario para la generacion del reporte de pacientes, turnos y diagnosticos
emitidos de acuerdo a un
# lapso de tiempo establecido
#
def general(request):
    return render(request, "report/general.html")

#
# Muestra un archivo pdf con el reporte de pacientes, turnos y diagnosticos emitidos de
acuerdo a un
# lapso de tiempo establecido
#
def general_print(request):
    desde = request.GET.get('desde', "").encode("utf_8")
    hasta = request.GET.get('hasta', "").encode("utf_8")

    try:
        turno_list = Turno.objects.filter(fecha__range=(desde, hasta)).order_by('-fecha')

        paciente_list = turno_list.values('paciente__cedula',
'paciente__sexo').annotate(total=Count('paciente__cedula')).order_by('paciente__cedula')
        paciente_hombre_list = paciente_list.filter(paciente__sexo='m')
        paciente_mujer_list = paciente_list.filter(paciente__sexo='f')

```

```

turno_especialidad_list =
turno_list.values('medico__especialidad__nombre').annotate(total=Count('medico__especialidad
__nombre')).order_by('medico__especialidad__nombre')

historia_clinica_list = Historia_Clinica.objects.filter(fecha__range=(desde,
hasta)).order_by('-fecha')
historia_clinica_enfermedad_list =
historia_clinica_list.values('diagnosticos__enfermedad__nombre').annotate(total=Count('diagnos
ticos__enfermedad__nombre')).order_by('diagnosticos__enfermedad__nombre')

html_string = render_to_string('report/general_print.html', {
    'desde': desde, 'hasta': hasta,
    'paciente_list': paciente_list, 'paciente_hombre_list': paciente_hombre_list,
'paciente_mujer_list': paciente_mujer_list,
    'turno_list': turno_list,
    'turno_especialidad_list': turno_especialidad_list,
    'historia_clinica_enfermedad_list': historia_clinica_enfermedad_list
})

except:
    html_string = ""

pdf = pdfkit.from_string(html_string, False)

response = HttpResponse(pdf, content_type='application/pdf')
response['Content-Disposition'] = 'attachment; filename="general.pdf"'
return response

```