UNIVERSIDAD NACIONAL JORGE BASADRE GROHMANN

Facultad de Ingeniería

Escuela Profesional de Ingeniería en Informática y Sistemas

ANÁLISIS COMPARATIVO DE RENDIMIENTO ENTRE SERVIDORES WEB NODE.JS Y APACHE UTILIZANDO DATOS DE UN CHAT DE TWITCH DEL CANAL XQC, 2023

TESIS

Presentada por:

Bach. Cristian Bladimir Peralta Segura

Para Optar el Título Profesional de:

INGENIERO EN INFORMÁTICA Y SISTEMAS

TACNA – PERÚ

2024

Acta de Sustentación de Tesis

En Laboratorio "A" de la ESIS, siendo las 09:40 horas del día 24 de mayo de 2024, y cumpliendo lo señalado en el Reglamento de Grados y Títulos de la Facultad de Ingeniería, se reunió el Jurado Calificador integrado por los docentes:

- Mag. Gianfranco Alexey Málaga Tejada — Presidente
- MSc. Hugo Manuel Barraza Vizcarra — Secretario
- Dr. Edwin Antonio Hinojosa Ramos — Vocal
- Mag. Luis Johnson Paul Mori Sosa — Accesitario

Designados mediante Resolución de Facultad N° 08333-2023-FAIN/UNJBG de fecha 28 de diciembre de 2023, para evaluar la Tesis: "Análisis comparativo de rendimiento entre servidores Web Node.js y apache utilizando datos de un chat de Twitch del canal XQC, 2023", presentado por el Bachiller Cristian Bladimir Peralta Segura, para optar el Título Profesional de Ingeniero en Informática y Sistemas. Le asesoró, el Dr. Edwin Antonio Hinojosa Ramos (R.F. N° 08035-2023-FAIN/UNJBG).

Ante la ausencia del presidente del jurado por una reunión propia de sus funciones como Director ESIS, asumió el Mag. Luis Johnson Paul Mori Sosa como accesitario.

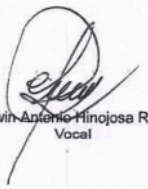
Dicho acto de sustentación se desarrolló en dos etapas: exposición y absolución de preguntas; procediéndose luego a la evaluación por parte de los miembros del jurado.

Habiendo absuelto las preguntas que le fueron formuladas por los miembros del Jurado Calificador, y de conformidad con las respectivas disposiciones reglamentarias, procedieron a deliberar y calificar, declarándolo APROBADO por unanimidad, con el calificativo 18 (DIECIOCHO).

Siendo las 10:40 horas del día 24 de mayo del 2024, los miembros del Jurado Calificador firman la presente Acta en señal de conformidad.

Mag. Luis Johnson Paul Mori Sosa
Presidente

MSc. Hugo Manuel Barraza Vizcarra
Secretario



Dr. Edwin Antonio Hinojosa Ramos
Vocal

CERTIFICADO DE SIMILITUD N° 003-2023-EAHR

Yo, **EDWIN ANTONIO HINOJOSA RAMOS**, en mi condición de asesor de tesis acreditado por la Resolución de Facultad N° 08035-2023-FAIN/UNJBG de la tesis titulada: *"ANÁLISIS COMPARATIVO DE RENDIMIENTO ENTRE SERVIDORES WEB NODE.JS Y APACHE UTILIZANDO DATOS DE UN CHAT DE TWITCH DEL CANAL XQC, 2023"* presentado por el BACH. CRISTIAN BLADIMIR PERALTA SEGURA (2013-39069) para optar el Título Profesional de Ingeniero en Informática y Sistemas. Habiendo cumplido con lo establecido en el reglamento de originalidad y de similitud de trabajos de investigación y producción intelectual, considerando que según la revisión, evaluación y análisis realizado a través del software de similitud textual TURNITIN cuenta con el nivel de similitud permitido cuyo porcentaje es 8%. Por lo que **CERTIFICO QUE LA SIMILARIDAD** de la tesis enunciada líneas arriba, la cual está expedita para continuar con los trámites para la obtención del Título Profesional de Ingeniero en Informática y Sistemas, según corresponda consiguientemente la publicación en el repositorio institucional.

Tacna, 05 de diciembre del 2023

ASESOR:
DR. EDWIN ANTONIO HINOJOSA RAMOS
DNI 00488610

Huella Digital

TESISTA:
BACH. CRISTIAN BLADIMIR PERALTA SEGURA
DNI 77686609

Huella Digital

DEDICATORIA

A mi padre, maestro incansable desde nuestra infancia; a mi amada madre, que aunque ya no está físicamente con nosotros, su amor y sacrificio siguen inspirándome y guiándome en cada uno de mis pasos; a mi hermano, cómplice en cada aventura y apoyo constante a lo largo de los años. Y a mi fiel perrito, compañero leal que ilumina mis días con alegría constante.

AGRADECIMIENTO

A mi querida familia, pilar de mi fortaleza, y a mi amada madre en el cielo, cuya luz sigue guiándome. Sus sacrificios y amor han sido mi mayor inspiración.

A mi querida casa académica, la Universidad Nacional Jorge Basadre Grohmann, y en especial a la Escuela Profesional de Ingeniería de Informática y Sistemas, así como al dedicado cuerpo docente y mi asesor, el Dr. Edwin Antonio Hinojosa Ramos, les agradezco por crear un entorno propicio y por su orientación, fundamentales para mi aprendizaje y crecimiento.

Por último, a todas las personas que, de alguna manera, brindaron su aliento, conocimientos o simplemente su presencia en mi vida académica. Espero que este trabajo refleje no solo mi esfuerzo sino también la influencia positiva de aquellos que han sido parte de mi viaje.

CONTENIDO

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

RESUMEN

El objetivo de esta investigación fue evaluar la diferencia del rendimiento entre los servidores web Node.js y Apache utilizando datos de un chat de Twitch del canal xQc.

El diseño de la investigación es no experimental, descriptiva-comparativa, con un enfoque específico en la capacidad de gestión de solicitudes y el uso de recursos del servidor. La población de interés comprende 643493 registros de mensajes correspondientes a 54668 usuarios, mientras que la muestra seleccionada consistió en 643 registros.

Los resultados obtenidos muestran que en términos de capacidad de gestión de solicitudes, Node.js exhibió un tiempo de respuesta significativamente mejor en comparación con Apache. La tasa de transferencia mostró diferencias no significativas, y las solicitudes fallidas fueron irrelevantes para ambos servidores. Asimismo en cuanto al uso de recursos, Node.js destacó con un uso de CPU significativamente menor, mientras que no se encontraron diferencias significativas en el uso de memoria.

Se concluye que debido a estos resultados que respaldan la hipótesis principal de que existe una diferencia significativa en el rendimiento entre Node.js y Apache en el contexto específico de este estudio, Node.js se posiciona como la opción preferida para este escenario particular. Finalmente se recomienda la búsqueda de configuraciones óptimas, especialmente para Apache en situaciones de alta concurrencia, y se sugiere la consideración de tecnologías como sockets y la evaluación de la escalabilidad de los servidores web en futuras investigaciones. Además, se insta a realizar pruebas con cargas más pesadas, como el trading en tiempo real, monitoreo de redes, eventos de redes sociales y juegos en línea.

PALABRAS CLAVE: rendimiento**,** servidor web, Node.js, Apache, tiempo de respuesta, uso de CPU.

ABSTRACT

The objective of this research was to assess the performance difference between Node.js and Apache web servers using data from the Twitch chat of the xQc channel.

The research design is non-experimental, descriptive-comparative, with a specific focus on the server's request management capacity and resource usage. The population of interest includes 643493 message records from 54668 users, while the selected sample consisted of 643 records.

The obtained results indicate that, in terms of request management capacity, Node.js exhibited a significantly better response time compared to Apache. The transfer rate showed non-significant differences, and the failed requests were irrelevant for both servers. Additionally, concerning resource usage, Node.js stood out with significantly lower CPU usage, while no significant differences were found in memory usage.

In conclusion, these results, which support the main hypothesis of a significant difference in performance between Node.js and Apache in the specific context of this study, position Node.js as the preferred option for this particular scenario. Finally, it is recommended to explore optimal configurations, especially for Apache in high-concurrency situations. Consideration of technologies such as sockets and the assessment of web server scalability are also suggested for future research. Additionally, conducting tests with heavier loads, such as real-time trading, network monitoring, social media events, and online gaming, is strongly encouraged.

KEYWORDS: performance, web server, Node.js, Apache, response time, CPU usage.

INTRODUCCIÓN

En la actual era de crecimiento exponencial de usuarios en internet, los servidores web se enfrentan a la apremiante tarea de gestionar una avalancha constante de millones de solicitudes en lapsos breves, provenientes de diversos usuarios concurrentes. Esta abrumadora demanda presenta desafíos considerables, ya que los servidores web no solo deben afrontar estas elevadas magnitudes de concurrencia, sino también cumplir con las expectativas de los usuarios que buscan experiencias web rápidas e instantáneas.

En este contexto, nuestra investigación cobra relevancia al sumergirse en el análisis comparativo de rendimiento entre dos destacados servidores web: Node.js y Apache. La esencia de este estudio reside en comprender cómo estos servidores abordan la tarea de procesar múltiples solicitudes, especialmente en un escenario tan particular como el chat en vivo del canal xQc en Twitch durante una transmisión del 2022.

A medida que exploramos las complejidades de estos servidores, no solo buscamos discernir cuál de ellos se destaca en términos de capacidad de gestión de solicitudes y uso de recursos que abarcan indicadores diversos como el tiempo de respuesta, tasa de transferencia, solicitudes fallidas, uso de CPU y memoria, sino que también aspiramos a proporcionar recomendaciones prácticas para configuraciones óptimas y los escenarios de uso más propicios para siguientes investigaciones.

En el primer capítulo, se aborda la problemática de la investigación, planteando sus objetivos y formulando las hipótesis. En el segundo capítulo, se ofrecen antecedentes relevantes, se establecen las bases teóricas y se define el marco conceptual. El tercer capítulo detalla el tipo y diseño de la investigación, la población y muestra de estudio, la operacionalización de las variables, así como los materiales e instrumentos utilizados, y presenta la metodología para el procesamiento y análisis de datos. El cuarto capítulo expone los resultados obtenidos, tanto a nivel descriptivo como inferencial, incluyendo las discusiones correspondientes a las hipótesis y objetivos. Finalmente, se ofrecen las conclusiones, recomendaciones, referencias bibliográficas y anexos.

CAPÍTULO I
PLANTEAMIENTO DEL PROBLEMA

1.1. Antecedentes del problema a investigar.

Se han reconocido antecedentes fundamentales en relación al rendimiento de los servidores web y su creciente importancia para las compañías que operan en el ámbito digital.

Dentro del ámbito de la investigación científica, a medida que las empresas se adentran en el vasto mundo en línea, los servidores web están tomando un papel cada vez más relevante en las operaciones de aquellas compañías cuya base está en el mundo virtual. Esto fue concluido por Ramirez Azanza en su investigación de 2019 llamada "Análisis comparativo de rendimiento a servidores web de distribución libre utilizando Apache Benchmark" (Ramirez Azanza, 2019). Este aumento en la importancia de los servidores web se refleja en el panorama global empresarial, donde su expansión en los últimos años marca una tendencia de crecimiento. En este escenario, el rendimiento de los servidores web se convierte en una preocupación crítica para aquellos sitios que enfrentan un gran flujo de solicitudes en la actualidad.

Adicionalmente, es relevante destacar que hasta agosto de 2023, se registra una impresionante cifra de más de “1,5 mil millones de páginas web” (Internet Live Stats, s.f.) en el ámbito digital, de acuerdo a la información presentada por esta página web. El incremento en la presencia en línea crea la demanda de que los servidores web puedan gestionar la concurrencia, dado que, como apunta Vilhelmsson (2021), "un usuario que quiera acceder a uno de estos sitios deben ser atendidos por su contenido y funciones mediante un servidor web" (p. 1).

En el contexto de una era digital en constante evolución, Choi et al. (2005) ya preveía la importancia fundamental de potenciar el rendimiento de los servidores web para hacer frente al creciente empleo de servicios en línea. A través de su análisis, Choi y demás integrantes de su investigación destacan tres “tácticas esenciales con el propósito de optimizar el desempeño de los servidores web: la implementación de mejoras en el software, el incremento de la capacidad del hardware y la adopción de estrategias de escalabilidad mediante el uso de clústeres” (Choi et al., 2005, p. 1). De acuerdo con estos investigadores, “estas tácticas tienen como objetivo primordial

maximizar la eficiencia de los servidores web y disminuir la latencia en el acceso a disco, permitiendo una respuesta más ágil a las solicitudes de los usuarios" (Choi et al., 2005, p. 1). Cabe mencionar, no obstante, que la implementación de cualquiera de estas alternativas conlleva un costo y demanda un conocimiento profundo para seleccionar la opción más pertinente, con el fin de alcanzar un rendimiento óptimo sin incurrir en un consumo excesivo de recursos.

Desde 1998, Hu et al. (1998) señalaban en su artículo presentado en la 17ª Conferencia Anual Conjunta de la IEEE Computer and Communications Societies, que la "importancia de los servidores web de alto rendimiento es esencial para satisfacer las crecientes demandas de internet y las intranets a gran escala" (Hu et al., 1998, p. 1). Un año después, Pai et al. (1999) expresaron una idea similar en su artículo titulado "Flash: An efficient and portable web server," presentado en la conferencia técnica anual USENIX de 1999, donde proponían una nueva arquitectura de servidores web denominada AMPED.

La exploración de la arquitectura de los servidores web también ha sido un enfoque clave en la investigación. Gokhale et al. (2006) afirmaron que "la arquitectura de un servidor web ejerce una influencia significativa en su rendimiento y fiabilidad" (Gokhale et al., 2006, p. 1). De hecho, estos autores ya reconocían que las opciones primordiales para el modelo de procesamiento en servidores web eran las que se fundamentaban en procesos, en hilos o una combinación híbrida de ambas.

En conjunto, estas investigaciones resaltan la importancia crítica del rendimiento de los servidores web en el contexto actual y proporcionan un marco sólido para comprender los antecedentes del problema abordado en la investigación científica "Análisis comparativo de rendimiento entre servidores web Node.js y Apache utilizando datos de un chat de Twitch del canal xQc, 2023".

1.2. Descripción del problema.

En el mundo de la tecnología actual, que es amplio y complejo, existe un problema importante en cómo elegimos y usamos tecnologías como los servidores web para crear aplicaciones y sistemas computacionales. Especialmente hoy en día, las tecnologías web son fundamentales y cada vez más necesarias para nosotros. Los usuarios actuales también son más exigentes en términos de lo que esperan de las aplicaciones. La decisión de qué tecnologías usar afecta mucho la experiencia de los

usuarios y el rendimiento de las aplicaciones en situaciones donde mucha gente las usa al mismo tiempo y en tiempo real. Pero no es solo un problema técnico; es un desafío grande que enfrentamos cuando diseñamos y creamos sistemas modernos.

Un ejemplo concreto que aclara esta problemática es el chat de una transmisión en vivo, como el chat de un canal en la plataforma de streaming en tiempo real muy reconocida llamada Twitch. En este ambiente, donde miles de usuarios interactúan al mismo tiempo, la capacidad del servidor web para responder de manera rápida se convierte en algo crucial. Para comprender realmente cómo los servidores web se comportan en este escenario de alto rendimiento y concurrencia, es esencial simular un caso como este. Por ejemplo, se puede tomar en cuenta una simulación que abarque 643493 registros de mensajes de 54668 usuarios durante 25 horas de transmisión de xQc en Twitch.

Este tipo de aplicaciones en tiempo real, donde la interacción en vivo es esencial, comparten similitudes con otras aplicaciones contemporáneas de la misma categoría. Aplicaciones de mensajería instantánea, redes sociales y plataformas de colaboración en línea también se enfrentan a desafíos similares, en términos de la gestión de cargas de trabajo simultáneas y las demandas de rendimiento. Aunque existen numerosas investigaciones sobre el rendimiento de servidores web en función de su arquitectura, se presenta una brecha en la literatura cuando se trata de aplicar comparaciones a situaciones reales y, sobre todo, actuales.

El problema principal radica en las elecciones de implementación que deben realizar tanto los desarrolladores individuales como las empresas tecnológicas. Elegir una tecnología y un servidor web con características particulares puede influir significativamente en el resultado positivo o negativo de una aplicación. Esto va más allá de una simple optimización técnica y se convierte en un elemento que impacta directamente en la experiencia del usuario final, los gastos operativos y la capacidad competitiva de una empresa en el contexto actual del mercado.

Este estudio, que compara los servidores web Node.js y Apache en un entorno de aplicación en tiempo real como el chat de Twitch, busca proporcionar información esencial para abordar esta problemática. Al evaluar cómo estas dos arquitecturas responden a las demandas de alto rendimiento y concurrencia, se aporta un entendimiento profundo que va más allá de lo técnico, permitiendo a los actores de la

industria tomar decisiones informadas y estratégicas para optimizar la experiencia del usuario y el rendimiento de sus aplicaciones.

1.3. Formulación del problema.

Problema general

¿Cuál es la diferencia entre el rendimiento de los servidores web Node.js y Apache utilizando datos de un chat de Twitch del canal xQc, 2023?

Problemas derivados

a) ¿Cuál es la diferencia en la capacidad de gestión de solicitudes entre los servidores web Node.js y Apache utilizando datos de un chat de Twitch del canal xQc, 2023?
b) ¿Cuál es la diferencia en uso de recursos entre los servidores web Node.js y Apache utilizando datos de un chat de Twitch del canal xQc, 2023?

1.4. Objetivos de la investigación

A continuación, presentaremos los objetivos de la investigación.

1.4.1. Objetivo general

Evaluar la diferencia del rendimiento entre servidores web Node.js y Apache utilizando datos de un chat de Twitch del canal xQc, 2023.

1.4.2. Objetivos específicos

a) Evaluar la capacidad de gestión de solicitudes de los servidores web Node.js y Apache utilizando datos de un chat de Twitch del canal xQc, 2023.
b) Medir el uso de recursos entre servidores web Node.js y Apache utilizando datos de un chat de Twitch del canal xQc, 2023.

1.5. Justificación e importancia de la investigación.

La justificación de esta investigación radica en la necesidad imperante de comprender y evaluar el rendimiento de servidores web en aplicaciones de alta concurrencia y tiempo real, como el chat de Twitch. Con la creciente dependencia de estas aplicaciones en la sociedad digital actual, es esencial abordar los desafíos técnicos y prácticos que surgen al implementar diferentes tecnologías y servidores. A pesar de la abundante investigación en el ámbito de arquitecturas de servidores, existe una carencia de estudios que apliquen estas comparaciones a situaciones reales y actuales. Esta

investigación busca llenar esta brecha al analizar de manera empírica y contextualizada cómo los servidores Node.js y Apache responden a las demandas de rendimiento en un entorno de alto tráfico y concurrencia. Los resultados de este estudio contribuirán a una toma de decisiones más informada por parte de los desarrolladores y las empresas, en su elección de tecnologías y servidores web, para garantizar una experiencia de usuario óptima y una aplicación exitosa en el competitivo panorama digital.

1.6. Limitaciones.

Limitación en el entorno de simulación: Dado que las simulaciones se realizaron en un solo computador, se reconocerá que ciertos aspectos del rendimiento del servidor y su interacción con la red podrían no reflejar completamente. Para contrarrestar esta limitación, se realizará una evaluación exhaustiva de cómo los servidores web Node.js y Apache responden a la carga simulada en un entorno controlado. Se indicará la necesidad de complementar estos resultados con estudios posteriores que consideren factores externos.

1.7. Viabilidad del estudio.

La presente investigación demuestra ser viable en base a diversos factores:

a) Disponibilidad de datos: La investigación se basa en un conjunto de datos reales de chat de Twitch, lo que garantiza la disponibilidad de información auténtica y actualizada para llevar a cabo las simulaciones y el análisis comparativo.
b) Acceso a herramientas y tecnologías: Las herramientas y tecnologías necesarias para simular el rendimiento de los servidores web Node.js y Apache están ampliamente disponibles y documentadas, lo que permite realizar la investigación de manera precisa y detallada.

1.8. Formulación de hipótesis.

A continuación, presentaremos las hipótesis de la investigación.

1.8.1. Hipótesis general

H0: No existe diferencia significativa en el rendimiento entre servidores web Node.js y Apache utilizando datos de un chat de Twitch del canal xQc, 2023.

H1: Existe diferencia significativa en el rendimiento entre servidores web Node.js y Apache utilizando datos de un chat de Twitch del canal xQc, 2023.

1.8.2. Hipótesis derivadas o secundarias

Primera hipótesis derivada

H0: No existe diferencia significativa en la capacidad de gestión de solicitudes entre servidores web Node.js y Apache utilizando datos de un chat de Twitch del canal xQc, 2023.

H1: Existe diferencia significativa en la capacidad de gestión de solicitudes entre servidores web Node.js y Apache utilizando datos de un chat de Twitch del canal xQc, 2023.

Segunda hipótesis derivada

H0: No existe diferencia significativa en uso de recursos entre servidores web Node.js y Apache utilizando datos de un chat de Twitch del canal xQc, 2023.

H1: Existe diferencia significativa en uso de recursos entre servidores web Node.js y Apache utilizando datos de un chat de Twitch del canal xQc, 2023.

1.9. Variables.

"Una variable es una propiedad que puede fluctuar y cuya variación es susceptible de medirse u observarse" (Hernández Sampieri et al., 2014, p.105). La única variable identificada en esta investigación es el rendimiento del servidor web, por lo cual no existe variable independiente ni dependiente, por lo que es univariable.

Caracterización de la variable:

Entonces nuestra única variable, "rendimiento del servidor web" se podría caracterizar de la siguiente manera:

- Por su naturaleza: Cuantitativa
- Por su escala de medición: Intervalo

1.10. Operacionalización de variables.

Definición conceptual: Siguiendo la perspectiva delineada en el estudio de Hu et al. (1998), el rendimiento del servidor web se define como la capacidad del servidor para gestionar eficientemente solicitudes web y entregar respuestas rápidas a los usuarios.

Definición operacional: Las medidas de rendimiento incluyen el tiempo de respuesta y la tasa de servicio, el uso de la memoria, la utilización de la CPU, entre otros. (Kunda et al., 2017, p. 1).

En el anexo 2, se proporciona una operacionalización exhaustiva de la variable, detallando minuciosamente cada elemento y procedimiento realizado para garantizar la integridad y precisión en la medición de dicha variable en el contexto de nuestra investigación.

CAPÍTULO II
MARCO TEÓRICO

2.1. Antecedentes del trabajo de investigación

A continuación, presentaremos los antecedentes internacionales, nacionales y locales de la investigación.

2.1.1. Antecedentes internacionales

En el estudio "A performance comparison of an Event-Driven Node.js web server and Multi-Threaded web servers" realizado por Vilhelmsson (2021) en Estocolmo, se llevó a cabo una comparación de rendimiento entre servidores web Node.js, Apache, Internet Information Services (IIS) y Go. Se evaluaron situaciones intensivas en E/S y cálculos, usando pruebas de cálculo de Fibonacci y consultas a una base de datos. Los resultados mostraron que el servidor Go tuvo el mejor rendimiento y menor consumo de memoria en todas las pruebas. Según Vilhelmsson (2021), “hubo un aumento promedio del 26 % en el rendimiento y una disminución promedio del 66 % en el consumo de memoria en comparación con los segundos mejores servidores”. Además, indica que el servidor IIS se ubicó con mayor frecuencia en el segundo lugar. Contrario a otros estudios anteriores Vilhelmsson (2021) señala que, “Node.js obtuvo un rendimiento inferior a Apache en pruebas intensivas en E/S”. Finalmente Vilhelmsson denota que los resultados indicaron que el servidor Go superó en rendimiento y consumo de memoria a los servidores Apache, IIS y Node.js en situaciones de E/S y cálculos. (Vilhelmsson, 2021).

En el estudio de Nguyen (2017) que se realizó en Hanoi, Vietnam, . Mediante análisis empíricos, demuestra que NginX, con su enfoque híbrido de procesos y eventos, supera en concurrencia y rendimiento a NodeJS y Apache. Nguyen (2017) señala que “los resultados indican hasta un 42 % más de concurrencia y un 16 % de aumento en el rendimiento en comparación con Apache”. Además, el tiempo de respuesta de NginX es el doble de rápido que el de Apache. Finalmente Nguyen (2017), concluye

que “el diseño híbrido basado en procesos y eventos puede generar un rendimiento superior en comparación con los demás enfoques”.

En la investigación de Ramírez Azanza (2019) realizada en Machala, Ecuador, se evaluaron servidores web de distribución libre, incluyendo Apache, Nginx y OpenLiteSpeed, mediante Apache Benchmark. Ramirez Azanza (2019), determina en sus resultados que señalan a “OpenLiteSpeed como altamente eficiente, con tiempos de conexión de 13 ms a 243 ms en situaciones de alta concurrencia”. Finalmente esta investigadora concluye que, OpenLiteSpeed es uno de los mejores servidores web para alojar una página web, “ya que su entorno se basa en procesos no en eventos, ahorrando recursos, logrando tener un impacto en el rendimiento del equipo, así como en la velocidad de despacho de la web” (Ramirez Azanza, 2019, p. 19).

2.1.2. Antecedentes nacionales

La investigación de Cubas Fernández (2019) realizada en Chiclayo, Perú, examinó el rendimiento de servidores web mediante pruebas de carga y estrés, específicamente IIS (Internet Information Service) y NGINX. Se utilizaron métricas como la media del tiempo de respuesta, tiempo mínimo y máximo de respuesta, y desviación estándar, con enfoque en cargas alta, media y baja. Donde Cubas Fernandez (2019) apunta que el servidor web IIS mostró resultados más óptimos en estos aspectos, con tiempos cercanos a 0 en las métricas que estaba evaluando en su investigación. Las pruebas de carga y estrés destacaron la capacidad del servidor IIS para soportar hasta 10000 usuarios concurrentes, mientras que el servidor NGINX presentó tiempos más bajos en estas condiciones. Estos hallazgos ayudan a una elección informada de servidores web en entornos de alto rendimiento.

2.1.3. Antecedentes locales

En el ámbito local, se ha encontrado una investigación relevante realizada por Arcaya Arhuata (2012), titulada "Sistema de informacion cliente/servidor con tecnología web para los procesos de matrículas y trámites de certificación de la escuela nacional de estadística e informática del INEI - Tacna - 2011". Aunque su enfoque difiere del presente estudio, dado que se centra en la implementación de un sistema de información basado en tecnología web para la gestión de matrículas y

trámites de certificación, esta investigación local proporciona un ejemplo de cómo la elección de servidores web puede estar influenciada por las características de la aplicación abordada.

En el caso de Arcaya Arhuata (2012), se observa que se optó por la combinación de Apache y PHP para el desarrollo del sistema. Esta elección podría atribuirse, en parte, a la familiaridad y estabilidad que Apache ofrecía en ese contexto temporal. Dado que Node.js era una tecnología emergente en ese momento, la preferencia por Apache, una opción consolidada y ampliamente utilizada, es comprensible.

Este antecedente local resalta la importancia de considerar el contexto y las necesidades específicas de la aplicación al seleccionar un servidor web. En el presente estudio, "Análisis comparativo de rendimiento entre servidores web Node.js y Apache utilizando datos de un chat de Twitch del canal xQc, 2023", se busca proporcionar un análisis objetivo y actualizado de las capacidades y el rendimiento de servidores web en un contexto específico, permitiendo así una toma de decisiones informada basada en datos empíricos y actuales.

2.2. Bases teóricas

A continuación, procederemos a describir las bases teóricas relacionadas con la investigación.

2.2.1. Servidor web

Según Liu et al. (2018) “un servidor web es un software de servidor que proporciona servicio a clientes, procesa las solicitudes entrantes de los clientes a través del protocolo HTTP”. (p. 2).

Por otro lado Luján Mora (2002, p. 49), define un servidor web como “un programa que se encuentra esperando permanentemente las solicitudes de conexión mediante el protocolo HTTP por parte de los clientes Web”. (p. 49).

Mientras que MDN (s.f.) menciona que para definir un servidor web podemos hacerlo de acuerdo al hardware o software, incluso de otra forma cuando trabajan juntos.

MDN (s.f.) describe a un servidor web basándonos en el hardware de la siguiente manera:

> En cuanto a hardware, un servidor web es una computadora que almacena el software de servidor web, y los archivos que componen un sitio web (por ejemplo, documentos HTML, imágenes, hojas de estilos CSS y archivos JavaScript). Un servidor web -hardware- se conecta a internet y mantiene el intercambio de datos con otros dispositivos conectados a la web (MDN, s.f.).

Y en cuanto a software de esta otra forma:

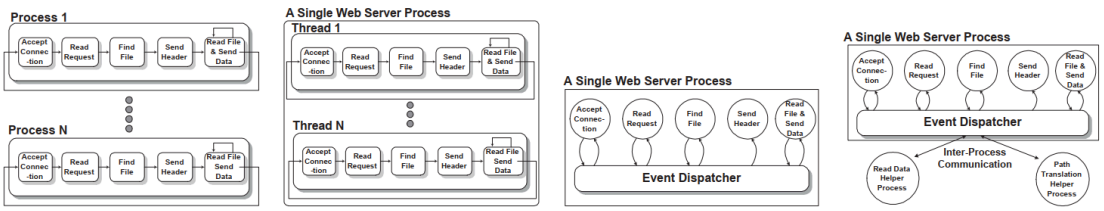
> Un servidor web tiene muchas partes que controlan cómo los usuarios de la web obtienen acceso a los archivos alojados en el servidor; es decir, mínimamente, un servidor HTTP. Un servidor HTTP es una pieza de software capaz de comprender URLs (direcciones web) y HTTP (el protocolo que tu navegador usa para obtener las páginas web). Un servidor HTTP puede ser accedido a través de los nombres de dominio de los sitios web que aloja, y entrega el contenido de esos sitios web alojados al dispositivo del usuario final (MDN, s.f.).

2.2.2. Arquitecturas de servidores web

Choi et al. (2005) en su paper donde propuso una nueva arquitectura para servidores múltiples procesadores llamada PIPELINED, detallo las arquitecturas de servidor existentes en ese contexto temporal para sistemas de un solo CPU como las que están en la Figura 1.

Figura 1

Arquitecturas de servidores web



(a) Multi-Process (MP) Model

(b) Multi-Thread (MT) Model

(c) Single-Process Event-Driven (SPED) Model

(d) Asynchronous Multi-Process Event-Driven (AMPED) Model

Nota: Esta figura muestra las arquitecturas de servidores web uniprocesador. De "A Multi-Threaded PIPELINED web server architecture for SMP/SoC machines" por Gyu Sang Choi et al., 2005. En WWW '05: Actas de la 14a Conferencia Internacional sobre la World Wide Web (pp. 730-739). Los derechos de autor son mantenidos por el Comité Internacional de la Conferencia World Wide Web (IW3C2).

“Todos estos modelos de servidores originalmente fueron propuestos para sistemas de un solo CPU”. (Choi et al., 2005, p. 3).

A continuación, describiremos las arquitecturas de los servidores que vamos a comparar: el modelo Multiprocesos (MP), en el cual se basa Apache; y el modelo Single Process Event Driven (SPED), que corresponde al servidor Node.js.

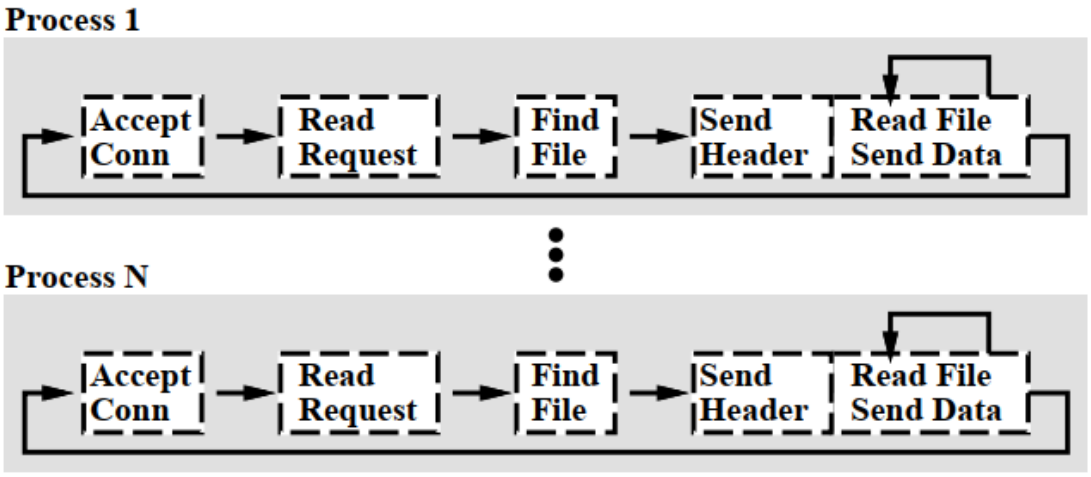
2.2.2.1. Multi Process (MP)

Pai et al. (1999), en su paper donde propuso la arquitectura para servidores web llamada AMPED, detallo otras arquitecturas existentes, en este caso la arquitectura multiproceso (MP), de la siguiente manera:

> En la arquitectura multiproceso (MP), se asigna un proceso para ejecutar los pasos básicos asociados con la atención secuencial de una solicitud de cliente. El proceso realiza todos los pasos relacionados con una solicitud HTTP antes de aceptar una nueva solicitud. Dado que se emplean varios procesos (por lo general, de 20 a 200), muchas solicitudes HTTP pueden ser atendidas de manera simultánea. Se produce de manera natural una superposición de actividad de disco, procesamiento de CPU y conectividad de red, ya que el sistema operativo cambia a un proceso en ejecución cuando el proceso actualmente activo se bloquea.
>
> Dado que cada proceso tiene su propio espacio de direcciones privado, no es necesario realizar sincronización para manejar el procesamiento de diferentes solicitudes HTTP. Sin embargo, puede ser más difícil realizar optimizaciones en esta arquitectura que dependan de información global, como una caché compartida de URL válidas (Pai et al.,1999, p. 2–3).

Figura 2

Arquitectura Multi Process (MP)



Nota: Esta figura muestra cómo funciona la arquitectura multiprocesos, cada proceso del servidor maneja una solicitud a la vez. Los procesos ejecutan las etapas de procesamiento de manera secuencial. De "Flash: An efficient and portable Web server" por Vivek S. Pai et al., 1999. En las Actas de la Conferencia Técnica Anual USENIX de 1999. (pp. 730-739). Los derechos de autor 1999 por The USENIX Association.

En nuestro estudio, Apache se alza como el representante de esta arquitectura (MP), y a través de la comparación de rendimiento, se pondrán de manifiesto las particularidades de su funcionamiento.

2.2.2.2. Single Process Event Driven (SPED)

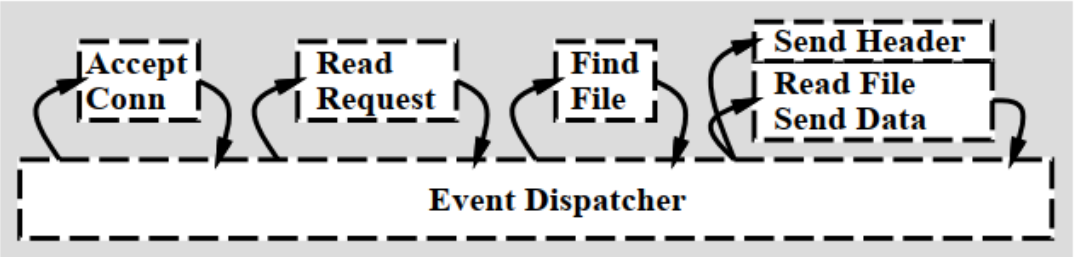
Pai et al. (1999), también describe la arquitectura Single Process Event Driven (SPED) y de la siguiente manera:

> La arquitectura de proceso único y dirigido por eventos (SPED, por sus siglas en inglés) utiliza un único proceso de servidor dirigido por eventos para realizar el procesamiento concurrente de múltiples solicitudes HTTP. El servidor utiliza llamadas al sistema no bloqueantes para llevar a cabo operaciones de E/S asincrónicas.
>
> Un servidor SPED puede considerarse como una máquina de estados que realiza un paso básico asociado con la atención de una solicitud HTTP a la vez, entrelazando así los pasos de procesamiento relacionados con muchas solicitudes HTTP. En cada iteración, el servidor realiza una operación de selección (select) para verificar eventos de E/S completados (nuevas conexiones entrantes, operaciones de archivo completadas, sockets de cliente que han recibido datos o tienen espacio en sus búferes de envío). Cuando un

evento de E/S está listo, se completa el paso básico correspondiente y se inicia el siguiente paso asociado con la solicitud HTTP, si es apropiado (Pai et al.,1999, p. 3).

Figura 3

Arquitectura Single Process Event Driven (SPED)



Nota: Esta figura muestra cómo funciona la arquitectura Single Process Event Driven (SPED), SPED utiliza un solo proceso para llevar a cabo todo el procesamiento de clientes y la actividad de disco de manera orientada a eventos. De "Flash: An efficient and portable Web server" por Vivek S. Pai et al., 1999. En las Actas de la Conferencia Técnica Anual USENIX de 1999. (pp. 730-739). Los derechos de autor 1999 por The USENIX Association.

En nuestra investigación, Node.js se presenta como el exponente de esta arquitectura (SPED), y al analizar su rendimiento, se expondrán las características específicas de su operación.

2.2.3. Rendimiento de servidores web

Pai et al. (1999) afirman lo siguiente:

> Un servidor web de alto rendimiento debe entrelazar los pasos secuenciales asociados con la atención de múltiples solicitudes para superponer el procesamiento de la CPU con los accesos a disco y la comunicación en red. La arquitectura del servidor determina qué estrategia se utiliza para lograr este entrelazado (Pai et al. 1999, p. 2).

Mientras que Kunda et al. (2017) aseguran que, "el rendimiento del servidor web es fundamental para la comunicación de información efectiva y eficiente" (p. 1). Y además hace mención de las medidas de rendimiento que son: "el tiempo de respuesta y la tasa de servicio, el uso de la memoria, la utilización de la CPU entre otros" (Kunda et al., 2017, p. 1). En este paper hace una revisión de varios estudios que indican una comparación cercana entre los diferentes servidores web que incluyeron Apache, IIS, Nginx y Lighttpd entre otros. Kunda et al. (2017) asegura que "los resultados de varios estudios indican que el tiempo de respuesta, la

utilización de la CPU y el uso de la memoria variaron con los diferentes servidores web según el modelo utilizado" (p. 1).

2.2.4. Métricas de rendimiento de servidores web

Según Kunda et al. (2017) "las medidas de rendimiento incluyen el tiempo de respuesta y la tasa de servicio, el uso de la memoria, la utilización de la CPU, entre otros" (p. 1).

Por otro lado Nguyen (2017) coincide con que las medidas de rendimiento de un servidor web son: Utilización de recursos, como el CPU y el disco local, así como el tiempo de respuesta de las solicitudes.

Y las métricas más usadas para medir el rendimiento de un servidor web según la investigación de Jader et al. (2019) son:

- Throughput: Rendimiento (o capacidad) de transferencia.
- Response time: Tiempo de respuesta.
- CPU utilization: Utilización de la CPU.
- Memory utilization: Utilización de memoria.
- Request per second: Solicitudes por segundo.
- Failed requests: Solicitudes fallidas.
- Concurrency: Concurrencia.

Y Wells et al. (2001), menciona que "existen varios documentos sobre el análisis de rendimiento de servidores web, y muchos de estos se centran en las caracterización de la carga de trabajo, o en la medición de la utilización de recursos y tiempo de respuesta" (p. 2).

Así que para nuestra investigación agrupamos en dos dimensiones las métricas para medir el rendimiento de los servidores web: Capacidad de procesamiento y uso de recursos:

1) Capacidad de gestión de solicitudes:
    a) Tiempo de respuesta.
    b) Tasa de transferencia.
    c) Solicitudes fallidas.
2) Uso de recursos:
    a) Uso de memoria.
    b) Uso de CPU.

2.2.5. Apache

"Apache es un servidor web gratuito y de código abierto basado en Unix, desarrollado por la Fundación Apache Software". (Laurie & Laurie, 2003).

En la encuesta de servidores web de julio del 2023 realizada por la empresa Netcraft, Apache representa un 20,81 % de la cuota de mercado de servidores web. (Netcraft, 2023).

Nguyen (2017), confirma que "el diseño de Apache se basa en hilos, donde se llama a un proceso principal (Módulos de Procesamiento Múltiple - MPM) al inicio, y se bifurcan procesos/hilos secundarios (módulos) para manejar solicitudes de manera concurrente" (p. 3). Y acorde con Laurie & Laurie (2003), "Apache puede actuar como un modelo multihilo, multiproceso o ambos, lo cual puede ser especificado por el MPM. "httpd" es el módulo central en Apache que implementa el procesamiento de solicitudes/respuestas HTTP" (p. 46).

Configuración de servidor Apache:

Para optimizar la configuración del servidor Apache y hacer frente a la carga específica que abordamos en esta investigación, es crucial considerar los límites de conexiones y otros detalles relevantes para cargas concurrentes. Así como también es importante destacar que utilizamos el MPM prefork de Apache. Por lo que basándonos en la misma documentación oficial de Apache, vamos a describir el módulo de procesamiento que usamos y las directivas para la configuración del propio servidor web Apache..

Apache MPM prefork: El Módulo de Procesamiento Múltiple (MPM) establece un servidor web que no utiliza hilos y opera mediante el modelo prefork. Cada proceso del servidor tiene la capacidad de manejar solicitudes entrantes, y un proceso principal supervisa el tamaño del conjunto de servidores. Asimismo, se destaca por ser el MPM más efectivo en el aislamiento de cada solicitud, asegurando que cualquier problema con una solicitud en particular no tenga impacto en otras. (The Apache Software Foundation, s.f.).

ServerLimit: Por definición según The Apache Software Foundation (s.f.), es el límite máximo en el número configurable de procesos. Para el MPM prefork, esta directiva establece el valor máximo configurado para MaxRequestWorkers durante la vida del proceso Apache httpd.

StartServers: Esta directiva vendría a ser el número de procesos hijos creados al iniciar el servidor. Y como explica The Apache Software Foundation (s.f.), dado que la cantidad de procesos se controla dinámicamente según la carga, generalmente hay poco motivo para ajustar este parámetro.

MinSpareServers: Esta directiva se refiere al mínimo de procesos hijos inactivos. Y como explica The Apache Software Foundation (s.f.), un proceso inactivo es aquel que no está manejando una solicitud. Si hay menos de MinSpareServers inactivos, el proceso principal creará nuevos hijos: generará uno, esperará un segundo, luego generará dos, esperará un segundo, luego generará cuatro, y continuará de manera exponencial hasta que esté generando 32 hijos por segundo.

MaxSpareServers: Esta directiva se refiere al máximo de procesos hijos inactivos. y a diferencia de MinSpareServers como indica The Apache Software Foundation (s.f.), si hay más de MaxSpareServers inactivos, el proceso principal eliminará los procesos excedentes.

MaxRequestWorkers: Esta directiva se refiere al número máximo de conexiones que se procesarán simultáneamente. Y para nuestro tipo de módulo del servidor (es decir, prefork), como The Apache Software Foundation (s.f.) explica, MaxRequestWorkers se traduce en el número máximo de procesos hijos que se lanzarán para atender solicitudes. Su valor predeterminado es 256 y para aumentarlo, también debe aumentar ServerLimit.

MaxConnectionsPerChild: Esta directiva se refiere al límite en el número de conexiones que un servidor hijo individual maneja durante su vida. Tal cual The Apache Software Foundation (s.f.) señala, después de MaxConnectionsPerChild conexiones, el proceso hijo morirá, además que si MaxConnectionsPerChild es 0, entonces el proceso nunca caducará.

KeepAlive: Esta directiva cuando está en su valor “On”, básicamente habilita las conexiones persistentes de HTTP.

KeepAliveTimeout: Esta directiva se refiere a la cantidad de tiempo que el servidor esperará para solicitudes subsiguientes en una conexión persistente.

MaxKeepAliveRequests: Esta directiva se refiere al número de solicitudes permitidas en una conexión persistente. Según The Apache Software Foundation (s.f.), la directiva MaxKeepAliveRequests limita el número de solicitudes permitidas por conexión cuando KeepAlive está habilitado por ende si se establece en 0, se

permitirán solicitudes ilimitadas. Dada la configuración del servidor, que cuenta con un procesador Intel Core i7-7700HQ con 4 núcleos y 11 GB de memoria RAM disponible, la siguiente configuración para el servidor Apache 2.4 se propone para manejar 643 peticiones HTTP durante 60 segundos:

ServerLimit: Establecido en 100 para permitir suficientes procesos de servidor para manejar las solicitudes, considerando la capacidad del procesador y la memoria disponible. Un valor más alto podría agotar los recursos disponibles.

StartServers: Configurado en 5 para inicializar un número moderado de servidores al inicio, aprovechando la capacidad del procesador para manejar cargas iniciales.

MinSpareServers y MaxSpareServers : Mantenidos entre 5 y 10 para tener un número suficiente de servidores inactivos disponibles para manejar nuevas solicitudes, evitando un tiempo de respuesta lento.

MaxRequestWorkers: Limitado a 100 para evitar consumir demasiados recursos y garantizar un rendimiento óptimo en el entorno de 4 núcleos de CPU.

MaxConnectionsPerChild : Establecido en un valor moderado (1000) para reiniciar los procesos hijos después de un número específico de conexiones, evitando problemas de pérdida de memoria a largo plazo.

KeepAlive: Habilitado para permitir la persistencia de conexión, reduciendo el tiempo de carga de la página al permitir que varias solicitudes utilicen la misma conexión.

KeepAliveTimeout: Establecido en 5 segundos para cerrar la conexión KeepAlive después de 5 segundos de inactividad, evitando conexiones inactivas prolongadas.

MaxKeepAliveRequests: Limitado a 100 para evitar que una conexión Keep-Alive se utilice indefinidamente, garantizando la eficiencia en la gestión de recursos.

2.2.6. Node.js

Node.js, como se describe en la documentación oficial de OpenJS Foundation and Node.js contributors (s.f.), es un "entorno de ejecución para JavaScript construido con V8, motor de JavaScript de Chrome." Desde su creación en 2009, gracias a la visión de Ryan Dahl (DelBono, 2017), ha evolucionado hasta convertirse en un componente vital del desarrollo web.

Uno de los aspectos distintivos de Node.js es su enfoque en la eficiencia y el rendimiento. A diferencia de muchos entornos modernos, Node.js opera bajo un modelo de ejecución asincrónica de E/S, como señalan Tilkov & Vinoski (2010).

Esto significa que, "a diferencia de los enfoques basados en multihilo, un proceso Node no depende de la multihilo para soportar la ejecución concurrente de la lógica empresarial" (Tilkov & Vinoski, 2010 , p. 1). En su lugar, se basa en un modelo de eventos asincrónicos, permitiendo que múltiples operaciones se realicen sin bloquear el flujo de ejecución.

La capa de red de Node.js es digna de atención en este análisis comparativo. Como menciona Herron (2020), "Los módulos HTTP de Node.js permiten la creación de servidores HTTP y clientes con apenas unas pocas líneas de código" (p. 12).

La arquitectura de Node.js es crucial para entender su funcionamiento. Como resalta Herron (2020), "la arquitectura de Node.js se basa en el envío de operaciones de bloqueo a un bucle de eventos de un solo hilo" (p. 10). Este enfoque garantiza que las operaciones de bloqueo no interfieran con el flujo de trabajo, ya que se manejan de manera asincrónica y se entregan como eventos a los manejadores correspondientes.

Estas características de Node.js lo clasifican dentro de la arquitectura SPED (Single-Thread, Event-Driven) ya que se basa en un modelo de ejecución de un solo hilo y eventos asíncronos. Este manejo de eventos permite una ejecución no bloqueante y eficiente, resultando particularmente beneficiosa para tareas que implican operaciones de lectura y escritura.

En la encuesta de servidores web de julio del 2023 realizada por la empresa Netcraft, Node.js posee una cuota de mercado de servidores web baja, ya que forma parte de los "otros" con un 23 % compartiendo con otros servidores webs de baja cuota. (Netcraft, 2023).

Según el informe de Gelbmann (2022), "Node.js se utiliza en el 2,4 % de todos los sitios web cuyos servidores web conocemos". Por ejemplo, algunos de los sitios web populares que usan Node.js son: Twitter, Netflix, Github, Vimeo, Adobe, Spotify, etc.

Configuración del servidor Node.js:

Dada la arquitectura Single-Process Event-Driven (SPED) de Node.js, su configuración por defecto es adecuada para manejar eficientemente la carga de 643 solicitudes en 60 segundos. Gracias a su capacidad para gestionar eventos de forma asincrónica y no bloqueante, Node.js proporciona una respuesta eficiente y

escalabilidad sin necesidad de ajustes detallados en esta carga moderada. La arquitectura SPED de Node.js se adapta eficazmente a escenarios con múltiples conexiones simultáneas, asegurando un rendimiento óptimo.

2.2.7. Apache Jmeter

Córdova Molina (2017), mencionaba que, "es importante definir los parámetros utilizados para determinar el desempeño de un servidor web". Además aclaraba que "varias herramientas son utilizadas para este fin como Funkload, Httperf, Apachebenchmak, Locust dando como resultados de sus pruebas el número de conexiones exitosas entre el varios clientes simulados y un servidor web dentro de un intervalo de tiempo" (Córdova Molina, 2017, p. 8).

Según la misma página oficial de Apache Jmeter describe esta herramienta de la siguiente manera:

> La aplicación Apache JMeter™ es un software de código abierto, una aplicación Java 100 % pura diseñada para probar la carga del comportamiento funcional y medir el rendimiento. Fue originalmente diseñada para probar aplicaciones web, pero desde entonces se ha expandido a otras funciones de prueba. (The Apache Software Foundation, s.f.).

Usaremos esta herramienta para medir algunas de las métricas del rendimiento de los servidores web, específicamente el tiempo de respuesta, la tasa de transferencia y la cantidad de solicitudes fallidas de los tests donde utilizaremos los datos de un chat de Twitch del canal xQc.

2.2.8. Monitor de rendimiento de Windows

En la página oficial de IBM (2023), específicamente en la sección donde presenta la documentación de DB2 en su versión 11.5 indica que, "el monitor de rendimiento de Windows es una de las herramientas administrativas de Windows, muestra una representación gráfica del rendimiento del sistema" (IBM, 2023).

Y desde la misma documentación de Microsoft (2012) indica que, "el monitor de rendimiento de Windows le permite realizar un seguimiento del impacto en el rendimiento de las aplicaciones y servicios, y generar alertas o tomar medidas cuando se superan los umbrales definidos por el usuario para un rendimiento óptimo" (Microsoft, 2012).

Usaremos esta herramienta para medir las dos métricas del rendimiento de los servidores web, específicamente el uso del CPU y uso de memoria durante los tests donde utilizaremos los datos del chat de Twitch del canal xQc.

2.2.9. Monitor de rendimiento de Linux

Para monitorear el rendimiento de los servidores web utilizamos el programa“top” de linux. Según Warner (s.f.), El programa top ofrece una perspectiva en tiempo real y dinámica de un sistema operativo en funcionamiento. Esta herramienta presenta información resumida del sistema y una lista de las tareas que están siendo actualmente administradas por el kernel de Linux.

Para monitorear el uso de CPU y memoria nos fijamos en las siguientes columnas del resultado del programa top:

- **% CPU (Uso de CPU):** La parte de la CPU utilizada por la tarea está expresada como un porcentaje del tiempo total de la CPU.
- **% MEM (Uso de memoria):** La parte de la memoria física disponible actualmente utilizada por la tarea y expresada en porcentaje.

2.2.10. Chat de Twitch

Para empezar vamos a dar contexto de los live streaming, o como en español lo llamamos, transmisiones en vivo.

“Live-streaming o streaming, como término, puede referirse a muchos aspectos de la tecnología y la cultura digitales, que van desde la transferencia de datos puramente técnica, hasta el fenómeno cultural del streaming” (Raman et al., 2018, p. 1).

Según Needleman (2015) “Twitch es actualmente la plataforma social de streaming de vídeo más popular en Europa y EE. UU., y ofrece un servicio para la actividad cultural y descentralizada de streaming y atiende a cientos de millones de espectadores”. Y en cuanto a los creadores de contenido que participan en Twitch, “a menudo se les llama streamers, y a los consumidores de contenido en Twitch se les llama espectadores. (Sjöblom et al., 2019, p. 3). Estos espectadores, como dice Hamilton (2014) “se comunican con el streamer y otros espectadores a través del chat” (p. 1).

Ahora VoD, son las siglas de Video on Demand, que traducido a español significa videos a la carta. “Si bien gran parte del énfasis está en la transmisión en

vivo, Twitch.tv también incluye videos bajo demanda (VoD) que permiten a los usuarios transmitir eventos en vivo anteriores" (Claypool, 2015, p. 2). Es decir los usuarios pueden reproducir los videos de las transmisiones anteriores. Junto a estos videos, Twitch también incluye la sección del chat, donde se pueden observar los mensajes que fueron enviados durante la transmisión en vivo.

Twitch dispone de una API que en su misma página de documentación (Twitch Developers, s.f.) indica lo siguiente, "la API de Twitch proporciona las herramientas y los datos utilizados para desarrollar integraciones de Twitch".

Dusk (2022) recopiló los registros de mensajes de la transmisión en vivo del canal xQc, que tuvo lugar el 11 de agosto de 2022. Utilizando las herramientas de la API de Twitch, estos registros fueron obtenidos y posteriormente compartidos públicamente en Kaggle.

Estos datos serán empleados como la representación de los registros de los mensajes del chat en aquella transmisión en vivo, subrayando su importancia en el análisis comparativo de rendimiento entre los servidores web Node.js y Apache . Utilizamos estos datos con el fin de obtener una representación más precisa y realista de un entorno de aplicación en línea en tiempo real. Estos datos desempeñarán un papel central en la evaluación y la obtención de conclusiones sólidas sobre la eficiencia de los servidores en un contexto de tráfico y cargas reales.

2.3. Definiciones conceptuales.

Node.js: Plataforma de ejecución para JavaScript en el lado del servidor que permite crear servidores web mediante sus módulos y está basado en SPED (Single Process, Event Driven).

Apache: Servidor web basado en el modelo MP (Multi Process).

SPED: (Single Process, Event Driven) Modelo proceso único, basado en eventos. Modelo de procesamiento con un solo proceso que atiende eventos.

MP: (Multi Process), Modelo multiproceso con múltiples procesos independientes

VoD: (Video on Demand), Video en demanda (videos grabados).

Twitch: Plataforma de transmisiones en vivo.

xQc: Creador de contenido de Twitch.

E/S: Entrada/Salida.

CPU: (Central Process Unit) Unidad central de procesamiento.

Jmeter: Software de código abierto,para probar la carga del comportamiento funcional y medir el rendimiento.

CAPÍTULO III
MARCO METODOLÓGICO

3.1. Planteamiento metodológico

A continuación describimos el tipo, nivel y diseño de la investigación.

3.1.1. Tipo y nivel de la investigación

Tal como lo plantea Vara Horna (2012), una investigación del tipo descriptivo-comparativo, "tienen como objetivo lograr la identificación de diferencias o semejanzas con respecto a la aparición de un evento en dos o más grupos" (p. 209). Este estudio se enmarca en un nivel de investigación descriptivo-comparativo.

Según Hernández Sampieri et al., (2014) el enfoque cuantitativo utiliza la recolección de datos para probar hipótesis con base en la medición numérica y el análisis estadístico, con el fin de establecer pautas de comportamiento y probar teorías. En coherencia con esta perspectiva, el presente estudio adopta un enfoque de investigación cuantitativo, enfocado en la recopilación y análisis de datos numéricos. Específicamente, se busca evaluar el rendimiento de los servidores web Node.js y Apache. Este análisis se realiza utilizando datos de chat de Twitch, particularmente extraídos de una transmisión del canal xQc.

En términos temporales, esta investigación es de naturaleza transversal o transeccional, ya que su objetivo es describir variables y analizar su relación e incidencia en un punto específico en el tiempo (Hernández Sampieri et al., 2014, p.154).

3.1.2. Diseño de la investigación

Según Hernández Sampieri et al. (2014), cuando en una investigación no se alteran las variables y se observa el evento para su análisis, se considera una investigación no experimental. El enfoque de diseño utilizado en este estudio es no experimental, dado que no se llevó a cabo ninguna intervención deliberada.

3.2. Población y muestra.

Según Hernández Sampieri et al., (2014) “una población es el conjunto de todos los casos que concuerdan con una serie de especificaciones” (p.174). En virtud de esto, la población que comprenderá la presente investigación abarcará un total de 643493 registros de mensajes generados por 54668 usuarios en el chat de Twitch (transmisión de 25 horas de xQc), la cual fue publicada el 11 de agosto de 2022.

De acuerdo a la afirmación de Hernández Sampieri et al., (2014) “la muestra es, en esencia, un subgrupo de la población” (p.175). Además el mismo autor indica que “en las muestras no probabilísticas, la elección de los elementos no depende de la probabilidad, sino de causas relacionadas con las características de la investigación o los propósitos del investigador”. (p.176)

Nuestra muestra se determinó usando muestreo por conveniencia, seleccionando el minuto con más registros dentro de la hora de máxima actividad en la transmisión. Extrajimos los mensajes de ese período para conformar la muestra que es un total de 643. Este enfoque se eligió debido a su idoneidad para la investigación y eficiencia en obtener datos representativos en una situación específica. Esto permitirá un análisis profundo de la comunicación en el chat de Twitch durante el pico de participación.

Hicimos el cálculo de esta muestra usando comandos de python, en el anexo 6 se podrá ver el código para calcular este dato.

3.3. Equipos y Materiales

Tabla 1

Características del hardware del servidor web

Nombre	Características	Descripción
Lenovo Legion Y520	Procesador: Intel(R) Core(TM) i7-7700HQ CPU @ 2,80GHz Tarjeta de video: Nvidia 1050p Memoria RAM: 16 GB Disco sólido: 256 GB, Disco duro: 2TB	Utilizado como servidor web, tanto para Apache como Node.js.

Nota: La tabla muestra las especificaciones de un equipo Lenovo Legion Y520. Este equipo se utilizará como servidor web para Node.js y Apache.

3.4. Procedimiento de las pruebas

Las pruebas se realizaron con el propósito de capturar datos relevantes relacionados con las dimensiones "capacidad de gestión de solicitudes" y "gestión de recursos".

En la realización de las pruebas, se presta especial atención a las versiones específicas del software empleado para garantizar la consistencia y fiabilidad de los resultados. El servidor web Node.js se implementó utilizando la versión 18.18.2, mientras que el servidor Apache con la versión 2.4.52. Es importante señalar que, dada la constante evolución del software, se optó por versiones disponibles en el momento de la investigación para mantener la coherencia en el análisis del rendimiento.

Se utilizaron herramientas específicas, como Apache JMeter y comandos de Linux, para recopilar información esencial.

A continuación, se explica el proceso:

1) Herramienta Apache JMeter

   Utilizamos Apache JMeter, una herramienta de código abierto ampliamente reconocida para realizar pruebas de rendimiento y carga. Con Apache JMeter, capturamos datos relacionados con la dimensión "capacidad de gestión de solicitudes". Estos datos incluyeron el tiempo de respuesta de cada solicitud, la tasa de transferencia y el número total de solicitudes fallidas. Apache JMeter proporcionó información valiosa sobre cómo se gestionan las solicitudes.

2) Uso de comandos de Linux

   Para obtener datos relacionados con la dimensión "gestión de recursos", empleamos comandos de Linux, siendo el principal comando "top". Estos comandos se utilizan para monitorear el uso de memoria y CPU durante las pruebas. Se realizaron las siguientes operaciones con comandos:

   Para Apache:

   - top -b -d 1 -n 70 | sed -e '/^[\t]*$/d' | awk '{print $1 "," $2 "," $3 "," $4 "," $5 "," $6 "," $7 "," $9 "," $10}' > informe_apache2.csv

   Para Node.js:

   - top -b -d 1 -n 70 | sed -e '/^[\t]*$/d' | awk '{print $1 "," $2 "," $3 "," $4 "," $5 "," $6 "," $7 "," $9 "," $10 "," $12}' > informe_node.csv

Estos comandos capturaron datos esenciales en tiempo real, proporcionando información detallada sobre el rendimiento de los servidores web durante las pruebas.

Estos comandos son una secuencia de comandos en Linux que se usa para monitorear el sistema y filtrar la información obtenida para generar un archivo CSV (formato de valores separados por comas) con datos específicos de top. Desglosamos cada parte del comando:

- **top:** Comando que muestra información dinámica sobre procesos en ejecución, uso de CPU, memoria, etc.
- **-b:** Indica que top se ejecuta en modo batch, lo que significa que producirá resultados para un solo ciclo y luego saldrá.
- **-d 1:** Especifica que top actualiza la información cada 1 segundo.
- **-n 70:** Limita top para que se ejecute 70 veces, es decir, durante 70 segundos (1 segundo * 70 veces).
- **|:** Tubo (pipe), se utiliza para enviar la salida del comando top el siguiente comando.
- **sed -e '/^[\t]*$/d':** sed es un editor de flujo que se utiliza aquí para eliminar líneas vacías o que contienen solo espacios o tabulaciones del resultado de top.
- **awk '{print $1 "," $2 "," $3 "," $4 "," $5 "," $6 "," $7 "," $9 "," $10," $12}':** awk es una herramienta de procesamiento de texto. Aquí, se seleccionan columnas específicas del resultado de top (por sus posiciones) y se imprimen con comas entre ellas para formar una línea de datos en formato CSV.
- **> informe_(apache2/node).csv:** Redirige la salida final (el CSV generado por awk) a un archivo llamado informe_(apache2/node).csv.

3) Análisis de los datos del chat de Twitch

Previo a las pruebas, se analizaron los registros del chat de la transmisión del canal de xQc en Twitch. Esta transmisión tiene una duración de 25 horas y un total de 643493 mensajes. Para facilitar la investigación, identificamos la hora de la transmisión con la mayor cantidad de mensajes. A continuación, identificamos los 10 minutos con mayor actividad dentro de esa hora.

Con los registros correspondientes a los 10 minutos más activos durante la hora con mayor cantidad de mensajes, realizamos pruebas utilizando Apache JMeter y "top". Los datos obtenidos, como el tiempo de respuesta y la tasa de transferencia, se compararon con pruebas de un minuto realizadas en el minuto con mayor cantidad de mensajes dentro de esos 10 minutos. La similitud de los valores demostró que representaban adecuadamente el flujo de datos de la transmisión.

4) Pruebas finales

Con base en la validación anterior, decidimos llevar a cabo las pruebas finales, cada una de 60 segundos de duración y con 643 registros. Durante cada prueba, se sincronizó la ejecución del comando "top" con la activación del test en Apache JMeter. Se realizó una ejecución controlada y precisa para capturar datos relevantes.

El enfoque de monitorización en tiempo real con el comando "top" y la ejecución de las pruebas en Apache JMeter permitieron obtener información valiosa sobre el rendimiento y la gestión de recursos en el contexto de la transmisión de Twitch. Cada prueba se ejecutó de manera cuidadosa y se registraron los datos necesarios para el análisis posterior.

3.5. Técnicas de recolección de datos

Según lo expresado por Vara Horna (2012), “las técnicas representan herramientas secundarias empleadas por los diseños como soporte, siendo específicas y poseyendo un enfoque instrumental” (p. 202). En esta investigación, la metodología seleccionada para recopilar los datos involucrará la observación, centrada en la comparación del rendimiento de dos servidores web.

Instrumento de recolección de datos

De acuerdo con Vara Horna (2012), "todo instrumento se desarrolla para medir o registrar una variable o conjunto de variables mediante una serie de preguntas, afirmaciones o indicadores (denominados ítems)" (p. 245). En esta investigación, se utilizará una guía de observación como instrumento para evaluar la variable única. A través de esta guía, se registrarán las observaciones relacionadas con los servidores Node.js y Apache.

En el anexo 3 se encuentra adjunta la guía de observación que se utilizará durante el proceso de investigación.

Para validar nuestro instrumento, se seguirá el enfoque de juicio de expertos, el cual, de acuerdo con Cabero Almenara y Llorente Cejudo (2013), "consiste en solicitar a un grupo de personas que emitan un juicio sobre un objeto, instrumento, material educativo u opinión acerca de un aspecto específico" (p. 14).

En el anexo 4 se presenta el documento para el proceso de validación del instrumento por juicio de expertos.

La validez de un instrumento se refiere a "la medida en que un instrumento de evaluación realmente captura lo que intenta medir o sirve para el propósito que ha sido diseñado" (Martín Arribas, 2004, p. 27). Para evaluar la validez de nuestro instrumento, utilizaremos la V de Aiken, propuesto por Aiken (1985). Este coeficiente permite cuantificar la relevancia de los ítems respecto a un dominio de contenido a partir de la valoración de N jueces o expertos. En este caso, hemos considerado la participación de 5 expertos.

Después de implementar la escala tipo Likert con las opciones de valoración "malo", "regular" y "bueno" en nuestro instrumento, asignamos valores numéricos en el rango de 1 a 3 respectivamente.

$$V = \frac{X - l}{k} \quad (1)$$

Donde:

V = Coeficiente V de Aiken.

X = Promedio de las calificaciones de todos los jueces.

l = Calificación mínima.

k = Diferencia de la calificación máxima menos la calificación mínima.

“El coeficiente resultante puede tener valores entre 0 y 1. Cuanto más el valor se acerque a 1, entonces tendrá una mayor validez de contenido” (Escurra Mayaute, 1988, p. 3).

Y las fórmulas de intervalos de confianza son:

$$L = \frac{2nkV + z^2 - z\sqrt{4nkV(1-V)+z^2}}{2(nk + z^2)} \quad (2)$$

$$U = \frac{2nkV + z^2 + z\sqrt{4nkV(1-V)+z^2}}{2(nk + z^2)} \quad (3)$$

Donde:

L = Límite inferior del intervalo de confianza.

U = Límite superior del intervalo de confianza.

Z = Valor en la distribución normal estándar, según nivel de confianza (para esta investigación será 95 % de confianza Z = 1,96).

V = Coeficiente V de Aiken.

k = Diferencia de la calificación máxima menos la calificación mínima.

n = Número de jueces.

Todas estas fórmulas están presentadas en el artículo de Penfield & Giacobbi (2004).

Cinco jueces expertos evaluaron los cinco ítems del instrumento (anexo 5), otorgando calificaciones en cuanto a su relevancia en relación a la escala propuesta, tal como se ilustra en la Tabla 2.

Tabla 2

Escala para validez de ficha de recolección de datos

MALO = 1	**REGULAR = 2**	**BUENO = 3**

Nota: La tabla presenta una escala de Likert utilizada para medir la relevancia de los ítems de la guía de observación.

Para evaluar si los ítems alcanzaron un nivel adecuado de validez de contenido, se aplicó el coeficiente V de Aiken con la participación de cinco jueces expertos. Este proceso se llevó a cabo con un nivel de confianza del 95 % y un criterio liberal, donde se estableció que el límite inferior debía ser igual o superior a 0,5 para que se considere válido el ítem.

La tabla muestra los cálculos correspondientes a los valores del coeficiente V de Aiken y sus respectivos intervalos de confianza para los cinco ítems incluidos en la guía de observación. En todos los casos, se confirmó la validez de los ítems.

Tabla 3
Coeficiente V de Aiken e intervalos de confianza al 95 %

Dimensión	Items	Promedio	V de Aiken	L. Inferior	L. Superior	Criterio Liberal V Aiken>=0.5
Capacidad de gestión de solicitudes	1	2.8	0.9	0.6	0.98	Es válido
	2	3	1	0.72	1	Es válido
	3	3	1	0.72	1	Es válido
Uso de recursos	4	3	1	0.72	1	Es válido
	5	3	1	0.72	1	Es válido

Nota: La tabla presenta el coeficiente V de Aiken e intervalos de confianza al 95 %.

3.6. Técnicas para el procesamiento de datos

A continuación, se detallan las principales técnicas que se aplicarán:

- Estadísticas descriptivas: Se utilizarán técnicas de estadísticas descriptivas para resumir y presentar los datos recopilados. Esto incluye el cálculo de medidas de tendencia central, medidas de dispersión, así como la creación de gráficos y visualizaciones que permitan una comprensión clara de la distribución de los datos.
- Contrastación de hipótesis: La contrastación de hipótesis se llevará a cabo específicamente para el indicador "tiempo de respuesta" dentro de la dimensión "capacidad de gestión de solicitudes". Esta técnica permitirá evaluar las afirmaciones específicas relacionadas con este indicador y determinar si existen diferencias significativas que respalden nuestras hipótesis.
- Prueba de normalidad: Dado que la contrastación de hipótesis se realizará para el indicador "tiempo de respuesta," se llevará a cabo una prueba de normalidad específicamente para este indicador. Esta prueba ayudará a determinar si los datos siguen una distribución normal, un supuesto importante en muchas pruebas estadísticas.

CAPÍTULO IV
RESULTADOS

4.1. Descripción de las pruebas experimentales

Se llevaron a cabo 10 pruebas experimentales, cada una compuesta por 643 solicitudes HTTP realizadas durante un período de 60 segundos cada una a una aplicación alojada en diferentes servidores web. Estas pruebas se basaron en la tasa de mensajes más alta registrada en el chat de un canal de Twitch llamado "xQc" durante un período de 25 horas. Para maximizar la eficiencia y representatividad de nuestras pruebas, seleccionamos el minuto con la mayor cantidad de mensajes dentro de la hora con la mayor actividad.

Nuestro enfoque en pruebas de 60 segundos, en lugar de pruebas de 25 horas, se basa en la similitud de la carga y en la viabilidad de ejecución. Realizar pruebas de larga duración resultaría en una carga de trabajo prácticamente idéntica, con la ventaja de una duración mucho más manejable. De las 10 pruebas, seleccionamos aleatoriamente una para llevar a cabo un análisis inferencial, específicamente enfocado en el indicador de “tiempo de respuesta”. Este indicador es fundamental para evaluar la capacidad de gestión de solicitudes de nuestros servidores web y, en última instancia, el rendimiento de los mismos.

4.2. Presentación y análisis de los resultados

Los resultados descriptivos se basan en los datos recopilados de las 10 pruebas realizadas, mientras que el análisis inferencial se llevará a cabo utilizando una selección al azar de entre las 10 pruebas existentes. Este último análisis se centrará en el indicador 'tiempo de respuesta'.

4.2.1. Resultados a nivel descriptivo

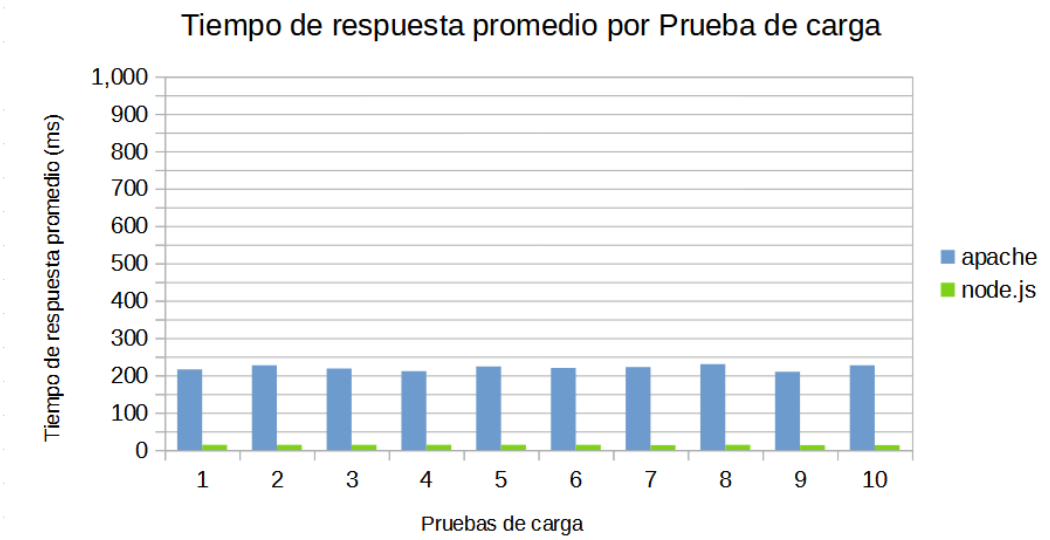
A continuación, se muestran los resultados logrados en términos del rendimiento de los servidores web, detallando sus dimensiones e indicadores respectivos.

Resultados para dimensión 1: Capacidad de gestión de solicitudes

Esta dimensión comprende tres indicadores: Tiempo de respuesta, tasa de transferencia y solicitudes fallidas.

Figura 4

Diagrama de barras del indicador Tiempo de respuesta

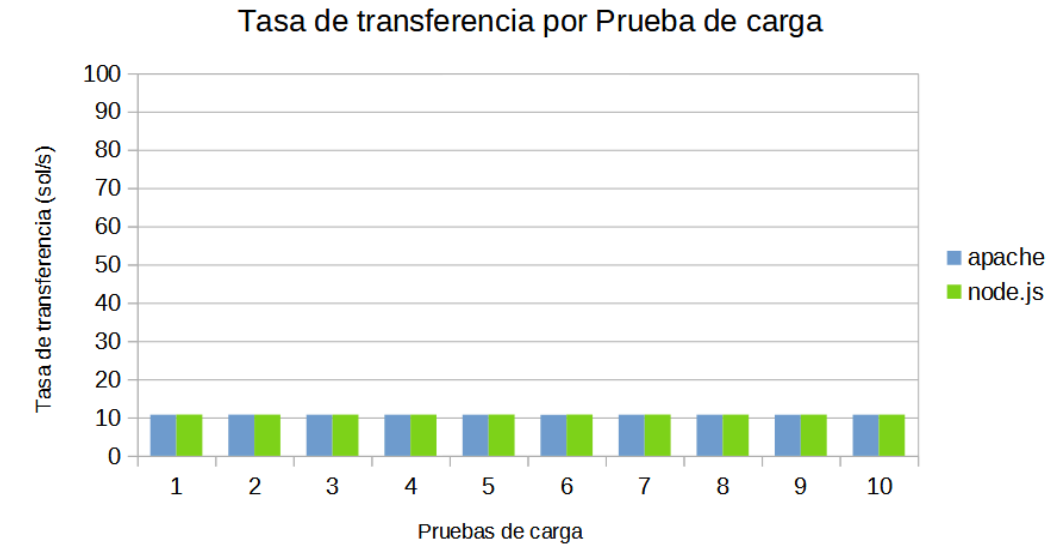


Nota: La figura presenta los promedios de tiempo de respuesta en varias pruebas de carga. La unidad de medida está representada por milisegundos (ms).

En la Figura 4, se destaca de manera significativa la disparidad en los tiempos promedio de respuesta de cada prueba de carga cuando se comparan los servidores Node.js y Apache.

Figura 5

Diagrama de barras del indicador tasa de transferencia

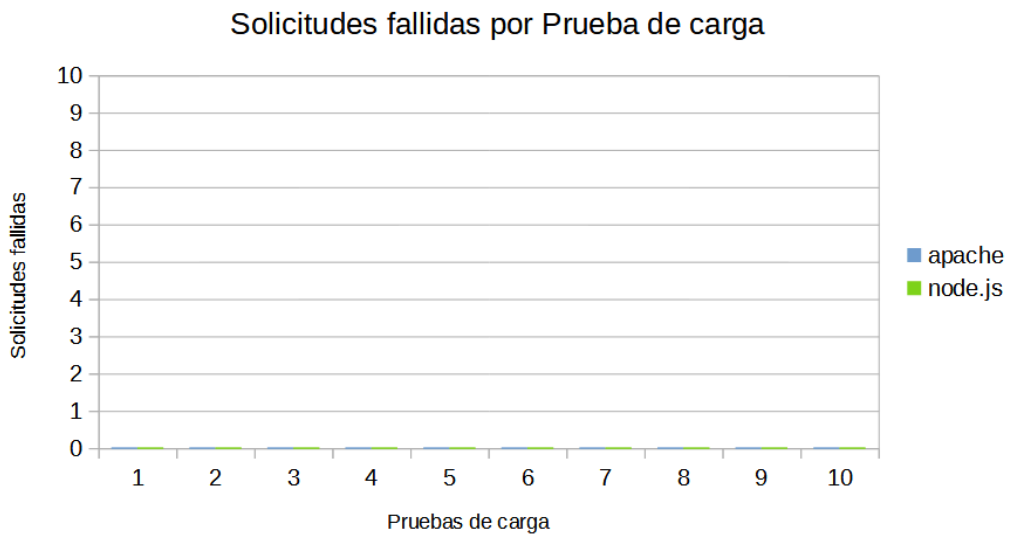


Nota: La figura presenta la tasa de transferencia en varias pruebas de carga. La unidad de medida está representada por el número de solicitudes por segundo (sol/s).

En la figura 5 se puede observar que la tasa de transferencia de nodejs es mejor en todas las pruebas de carga. Sin embargo la diferencia es muy poca si recalcamos que la unidad de medida es el total de número de solicitudes por segundo y la máxima diferencia entre la tasa de transferencia de Apache y Node.js es de 0,07 solicitudes por segundo.

Figura 6

Diagrama de barras del indicador solicitudes fallidas



Nota: La figura presenta el total de solicitudes fallidas en las pruebas de carga.

En la figura 6 se observa que en todas las pruebas de carga el total de solicitudes fallidas es nula para los dos servidores web. Por lo que se prevé que este indicador pasa a ser irrelevante para esta investigación en concreto.

Para entrar más a detalle veremos los datos estadísticos descriptivos que incluyen los valores mínimos, máximos, media y desviación estándar por cada servidor web.

Node.js

A continuación, exponemos los resultados obtenidos al aplicar el análisis estadístico descriptivo a los datos de las pruebas realizadas en el servidor Node.js.

Tabla 4

Resultados de estadísticos descriptivos en SPSS para la dimensión "capacidad de gestión de solicitudes" de las pruebas de carga a Node.js

Estadísticos descriptivos

	N	Mínimo	Máximo	Media	Desv. estándar
tiempo de respuesta (ms)	10	13.00	14.00	13.7000	.48305
tasa de transferencia (sol/c)	10	10.77	10.77	10.7700	.00000
solicitudes fallidas	10	.00	.00	.0000	.00000
N válido (por lista)	10				

Nota: Esta tabla muestra el mínimo, máximo, media y desviación estándar de cada uno de los indicadores de la dimensión "capacidad de gestión de solicitudes" para el servidor Node.js.

En la Tabla 4, se evidencia que el tiempo de respuesta promedio de Node.js en las pruebas de carga se mantiene constante, respaldado por el mínimo de 13 ms y el máximo de 14 ms. Esto se confirma aún más por la baja desviación estándar de 0,48305, especialmente considerando que estamos trabajando con unidades de milisegundos. Y ni mencionar la tasa de transferencia, que mantuvo un valor constante en todas las pruebas de carga. Además, el hecho de que el número total de solicitudes fallidas sea nulo en todas las pruebas demuestra que la cantidad de carga utilizada es fácilmente manejable por el servidor web Node.js.

Apache

A continuación, exponemos los resultados obtenidos al aplicar el análisis estadístico descriptivo a los datos de las pruebas realizadas en el servidor Apache.

Tabla 5

Resultados de estadísticos descriptivos en SPSS para la dimensión "capacidad de gestión de solicitudes" de las pruebas de carga a Apache

Estadísticos descriptivos

	N	Mínimo	Máximo	Media	Desv. estándar
tiempo de respuesta (ms)	10	210.00	230.00	220.5000	6.80278
tasa de transferencia (sol/s)	10	10.70	10.75	10.7420	.01751
solicitudes fallidas	10	.00	.00	.0000	.00000
N válido (por lista)	10				

Nota: Esta tabla muestra el mínimo, máximo, media y desviación estándar de cada uno de los indicadores de la dimensión "capacidad de gestión de solicitudes" para el servidor Apache.

En la Tabla 5, se observa que el tiempo de respuesta promedio de Apache en las pruebas de carga varía muy poco, reflejado en una desviación estándar de 6,80278, lo

cual es notable teniendo en cuenta que estamos trabajando con milisegundos. En lo que respecta a la tasa de transferencia, también muestra una baja desviación estándar de 0,1751, la cual es insignificante en esta escala. Además, el hecho de que el total de solicitudes fallidas sea nulo en todas las pruebas indica que la cantidad de carga aplicada es completamente manejable para el servidor web Apache.

Diferencias de los resultados entre los servidores web

Ahora hablaremos de las diferencias entre los resultados de estas pruebas de carga a los servidores web.

En las tablas 4 y 5, se puede apreciar que el **tiempo de respuesta** promedio para las 643 solicitudes HTTP en el servidor Node.js es de 13,70 ms, mientras que en el servidor Apache es de 220,50 ms. En promedio, la disparidad es de 206,80 ms, lo que sugiere de manera preliminar una notoria diferencia significativa en los tiempos de respuesta a favor de Node.js.

De igual manera, se observa que la **tasa de transferencia** promedio para las 643 solicitudes HTTP en el servidor Node.js es de 10,77 solicitudes por segundo, en contraste con el servidor Apache, que alcanza 10,74 solicitudes por segundo. En promedio, la discrepancia es de sólo 0,03 solicitudes por segundo, lo que sugiere preliminarmente que no hay una diferencia significativa en la tasa de transferencia entre ambos servidores web.

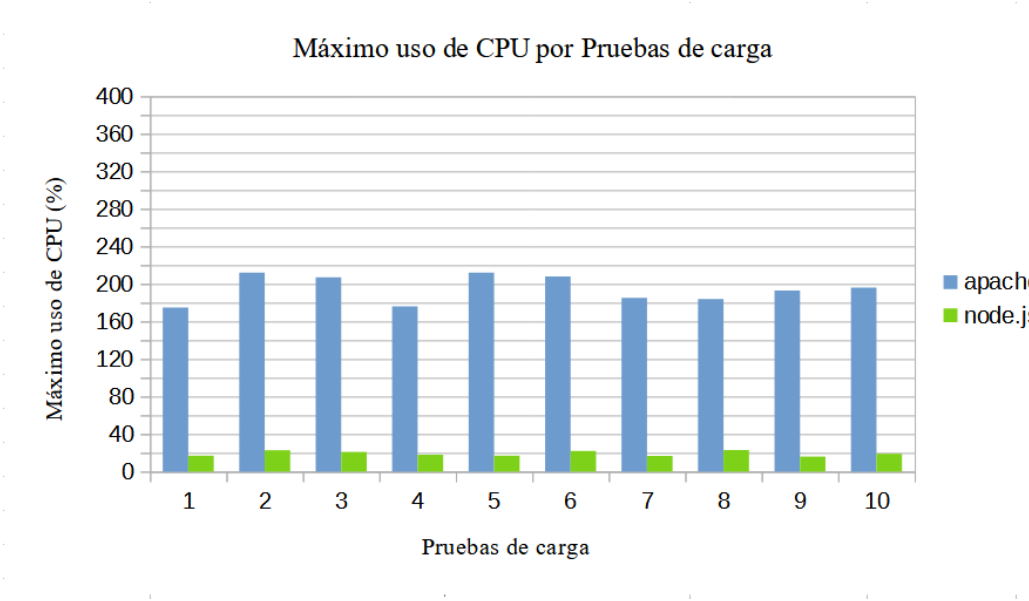
Por último, es importante destacar que el número total de **solicitudes fallidas** en cada prueba siempre se mantuvo en cero tanto para los servidores Node.js como para Apache. Por lo tanto, podemos concluir que este indicador carece de relevancia en esta comparación específica donde usamos los datos de los registros de mensajes del chat de una transmisión VoD del canal de xQc.

Resultados para dimensión 2: Uso de recursos

Esta dimensión comprende dos indicadores: Uso de memoria y uso de CPU. La métrica usada para estos indicadores es básicamente el máximo porcentaje de uso de cada recurso por test. En este caso se realizaron 10 tests para obtener una media para cada indicador.

Figura 7

Diagrama de barras del indicador máximo uso de CPU

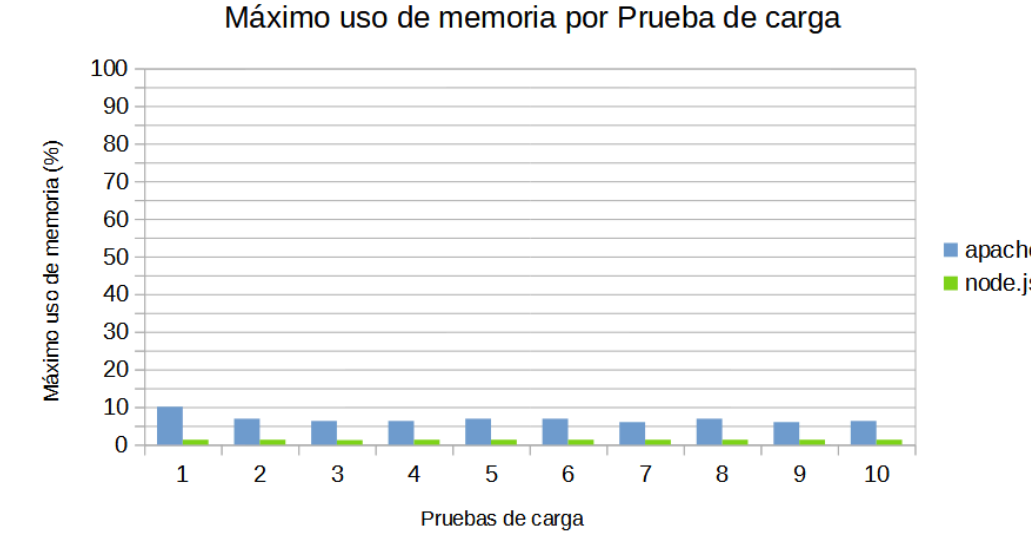


Nota: La figura presenta el máximo uso de CPU en varias pruebas de carga. La unidad de medida está representada por el porcentaje de uso de CPU utilizado (%).

La Figura 7 pone de manifiesto una diferencia relevante en el uso máximo de CPU en % entre los servidores web Node.js y Apache. En el caso de Apache, incluso supera el 100 %, lo que indica que está haciendo uso de múltiples núcleos del procesador.

Figura 8

Diagrama de barras del indicador Máximo uso de memoria



Nota: La figura presenta el máximo uso de memoria en varias pruebas de carga. La unidad de medida está representada por el porcentaje de uso de memoria (%).

La Figura 8 revela que Apache consume más memoria en comparación con Node.js, lo cual se atribuye a la arquitectura que utiliza este servidor web. Esta arquitectura, como se ilustra en la Figura 2, es conocida como MP (Multiprocesos), y esencialmente implica que, para cada solicitud, se genera un nuevo proceso que utiliza una cantidad fija de memoria.

Para entrar más a detalle veremos los datos estadísticos descriptivos que incluyen los valores mínimos, máximos, media y desviación estándar por cada servidor web.

Tabla 6

Resultados estadísticos descriptivos en SPSS para la dimensión "uso de recursos" de las pruebas de carga a Node.js

Estadísticos descriptivos

	N	Mínimo	Máximo	Media	Desv. estándar
uso de cpu (%)	10	16.00	23.00	19.2600	2.69988
uso de memoria (%)	10	1.20	1.30	1.2900	.03162
N válido (por lista)	10				

Nota: Esta tabla muestra el mínimo, máximo, media y desviación estándar de cada uno de los indicadores de la dimensión "uso de recursos" para el servidor Node.js.

En la Tabla 3, se puede apreciar que el uso máximo de CPU durante las pruebas de carga se mantiene en un rango entre el 16 % y el 23 %, lo cual resulta completamente manejable para la carga aplicada. En cuanto al consumo de memoria, es prácticamente insignificante, ya que no supera el 1,3 % del servidor. Este bajo uso de memoria es una característica distintiva de la arquitectura que emplea Node.js, conocida como SPED (Single Spread Event Driven), la cual se ilustra en la Figura 3. En esta arquitectura, Node.js utiliza un solo proceso que administra concurrentemente las solicitudes entrantes.

Tabla 7

Resultados estadísticos descriptivos en SPSS para la dimensión "uso de recursos" de las pruebas de carga a Apache

Estadísticos descriptivos

	N	Mínimo	Máximo	Media	Desv. estándar
uso de cpu (%)	10	175.00	212.00	194.8000	14.44376
uso de memoria (%)	10	6.00	10.10	6.8600	1.19833
N válido (por lista)	10				

Nota: Esta tabla muestra el mínimo, máximo, media y desviación estándar de cada uno de los indicadores de la dimensión "uso de recursos" para el servidor Apache.

En la Tabla 7, se aprecia que en todas las pruebas de carga, el uso máximo de CPU porcentual supera el 100 %, llegando incluso a alcanzar un 212 %. Como se mencionó previamente, esto indica que el servidor tuvo que emplear múltiples núcleos para gestionar la cantidad de solicitudes entrantes. Por otro lado, el máximo uso de memoria registrado fue del 10,10 %, lo cual no representa ningún problema para el servidor web Apache.

Diferencias de los resultados entre los servidores web

En las tablas 6 y 7, se puede notar que el máximo **uso de CPU** promedio en porcentaje es del 19,26 % para el servidor Node.js, mientras que para el servidor Apache es del 194,80 %. Esto resulta en una diferencia promedio de 175,54 %, lo que sugiere de manera preliminar una diferencia significativa en el uso de CPU a favor de Node.js.

Igualmente, es claro que el promedio del máximo **uso de memoria** en porcentaje es de 1,29 % para el servidor Node.js, en contraposición al servidor Apache, que registra un 6,86 %. En promedio, la discrepancia es de 5,57 %, lo que sugiere inicialmente que no existe una diferencia significativa en el uso de memoria a favor de Node.js. Esto se debe a que, en este contexto particular de carga de solicitudes HTTP, el uso de la CPU parece ser un factor más determinante para el soporte de la carga. Al menos para este tipo de flujo de peticiones que simula una aplicación de mensajería instantánea de texto, la memoria no representa un problema significativo, y por lo tanto, no se observa una diferencia relevante.

4.2.2. Resultados a nivel inferencial

En el marco de este estudio, se aborda la evaluación de diversas dimensiones relacionadas con el rendimiento de los servidores web. Entre estas dimensiones, "capacidad de gestión de solicitudes" se destaca como un elemento fundamental.

Se ha identificado un indicador crítico: el "tiempo de respuesta." Este indicador desempeña un papel esencial en esta investigación. Por lo tanto, en este análisis, hemos decidido centrarnos en la contrastación de hipótesis y la prueba de normalidad específicamente para el indicador "tiempo de respuesta," con el objetivo de proporcionar una evaluación más detallada y fundamentada en este aspecto particular de la "capacidad de gestión de solicitudes".

Prueba de normalidad

Se analizaron los datos con un nivel de confianza de 95 % y significancia de 5 % y se plantearon las siguientes hipótesis específicamente para los datos obtenidos de un test aleatorio:

Ho: Los datos provienen de una distribución normal.

Hi: Los datos no provienen de una distribución normal.

Según las palabras de Guisande González et al. (2013) el test de Kolmogorov-Smirnov "es la prueba adecuada para testar la normalidad de una muestra si el número de datos es grande (n >30), aunque se puede usar tanto para muestras grandes como pequeñas". (p.108)

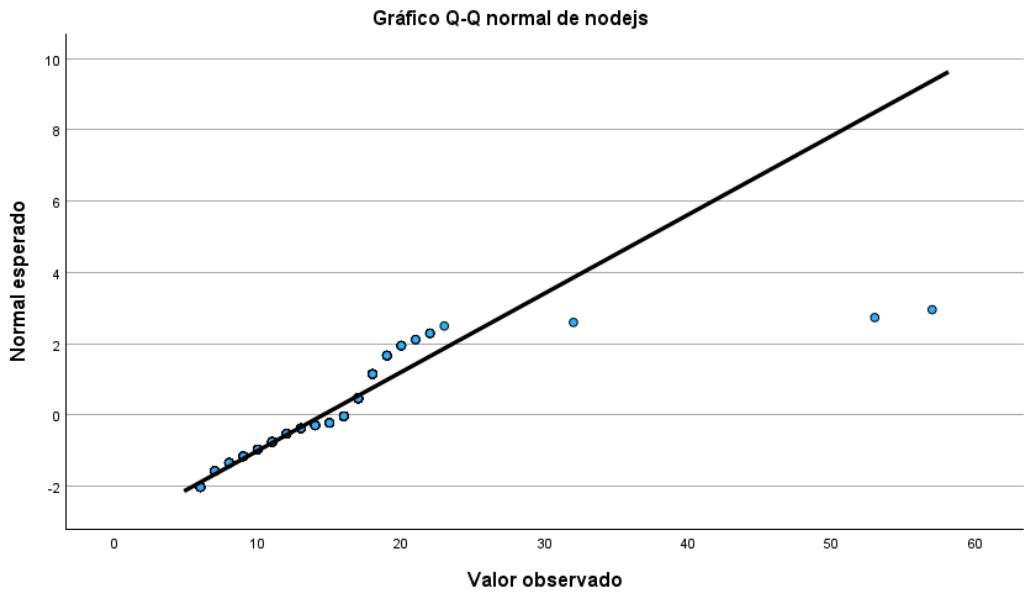
Se empleó la prueba de Kolmogorov-Smirnov para evaluar el indicador de tiempo de respuesta, ya que el tamaño de la muestra (643) superó los 30 valores requeridos

- Si $P \geq \alpha$, entonces se acepta Ho
- Si $P < \alpha$, entonces se acepta Hi

Se aplicó la prueba de normalidad y los resultados fueron los siguientes:

Figura 9

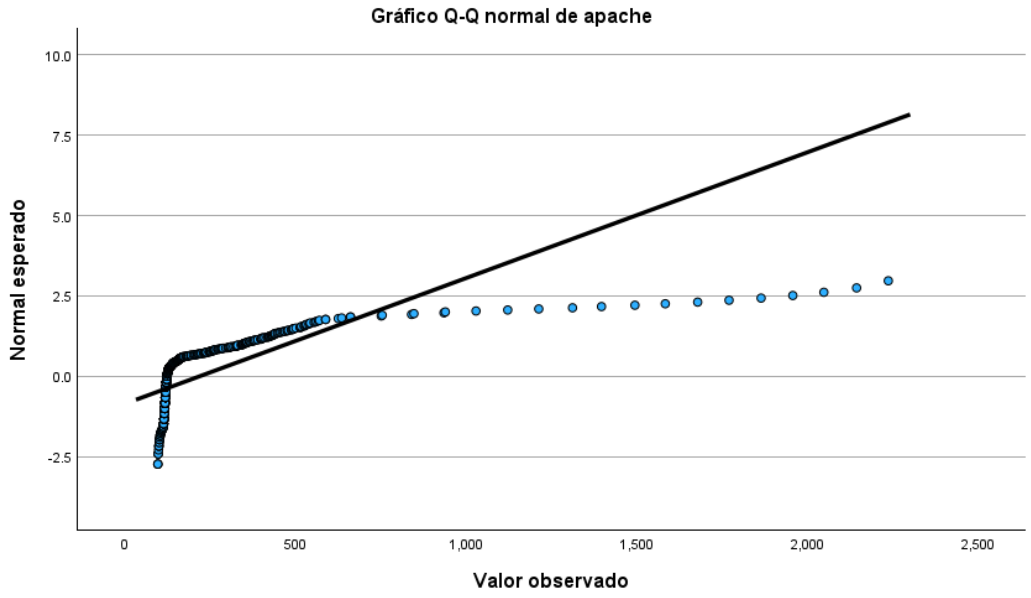
Distribución del tiempo de respuesta en SPSS para Node.js



Nota: Esta captura muestra la distribución de los tiempos de respuesta de las solicitudes que conforman parte de la prueba de carga para Node.js y la normal esperada.

Figura 10

Distribución del tiempo de respuesta en SPSS para Apache



Nota: Esta captura muestra la distribución de los tiempos de respuesta de las solicitudes que conforman parte de la prueba de carga para Apache y la normal esperada.

En los figura 9 y 10, es evidente que tanto el servidor Node.js como el servidor Apache no muestran la tendencia de normalidad esperada. Esto se debe a que muchos

de los datos no se ajustan a la tendencia normal que se representa mediante una línea diagonal. No obstante, es importante destacar que la dispersión de datos es más notable en el caso de Apache.

Seguidamente obtenemos nuestro P-valor mediante el siguiente cuadro que nos proporciona SPSS aplicando la prueba de Kolmogorov-Smirnov.

Tabla 8

Resultados de la prueba de normalidad en SPSS para el indicador tiempo de respuesta

Pruebas de normalidad

	Kolmogorov-Smirnov[a]			Shapiro-Wilk		
	Estadístico	gl	Sig.	Estadístico	gl	Sig.
apache	.313	644	<.001	.455	644	<.001
nodejs	.201	644	<.001	.804	644	<.001

a. Corrección de significación de Lilliefors

Nota: Esta tabla muestra el valor de nuestro P-Valor representado por la columna Sig., específicamente de la sección de Kolmogorov Smirnov.

Mediante los resultados obtenidos en la prueba de normalidad de Kolmogorov, se concluye que al ser nuestro P-Valor = 0,001 < α = 0,05, entonces se acepta Hi, entonces los datos no provienen de una distribución normal tanto para los datos obtenidos de los tests de Apache y Node.js.

Prueba de hipótesis para el tiempo de respuesta

H0: No existe diferencia significativa en los tiempos de respuesta entre servidores web Node.js y Apache utilizando datos de un chat de Twitch del canal xQc - 2023.

H1: Existe diferencia significativa en los tiempos de respuesta entre servidores web Node.js y Apache utilizando datos de un chat de Twitch del canal xQc - 2023.

Para esta prueba se estableció un nivel de significancia α = 0,05 y como nuestros datos no siguen una distribución normal se utilizó el estadístico de **prueba U de Mann-Whitney** para comparar medianas,

Conforme a Ríos y Peña (2020), la prueba U de Mann-Whitney, un estadístico no paramétrico, es utilizado para comparar proporciones y medianas, o características equivalentes a la comparación de promedios, especialmente cuando no se cumple el supuesto de normalidad.

Aplicamos la prueba y obtenemos nuestro P-valor mediante el siguiente cuadro que nos proporciona SPSS.

Tabla 9

Resultados de la prueba de U de Mann-Whitney en SPSS para el indicador tiempo de respuesta

Estadísticos de prueba[a]

			tiempo_respuesta
U de Mann-Whitney			.000
W de Wilcoxon			207690.000
Z			-31.116
Sig. asin. (bilateral)			<.001
Sig. Monte Carlo (bilateral)	Sig.		<.001[b]
	Intervalo de confianza al 95%	Límite inferior	.000
		Límite superior	.002
Sig. Monte Carlo (unilateral)	Sig.		<.001[b]
	Intervalo de confianza al 95%	Límite inferior	.000
		Límite superior	.002

Nota: Esta tabla muestra el valor de nuestro P-Valor representado por la fila Sig. Monte Carlo (bilateral) > Sig.

Según los resultados obtenidos en la prueba U de Mann-Whitney en la figura 16, se concluye que al ser nuestro P-Valor = 0,000 < α = 0,05, entonces se acepta Hi, por lo tanto, existe diferencia significativa en los tiempos de respuesta entre servidores web Node.js y Apache, utilizando datos de un chat de Twitch del canal xQc - 2023.

4.3. Contrastación de hipótesis

En el contexto de esta investigación, nos centraremos en llevar a cabo un análisis de contraste de hipótesis específico. Nuestro objetivo es evaluar la dimensión "capacidad de gestión de solicitudes" de la variable "rendimiento del servidor web". Más concretamente, nos enfocaremos en el indicador de "tiempo de respuesta".

4.3.1 Contrastación de hipótesis para el indicador “tiempo de respuesta”

Ya que se realizó estadística inferencial a el indicador “tiempo de respuesta” que pertenece a la dimensión “capacidad de gestión de solicitudes”. Formulamos una hipótesis específica para este indicador.

a) Planteamiento de hipótesis

H1: Existe diferencia significativa en el tiempo de respuesta entre servidores web Node.js y Apache utilizando datos de un chat de Twitch del canal xQc - 2023.

b) Nivel de significancia

$\alpha = 0,05$ o 5 % de error

c) Prueba estadística

Prueba U de Mann-Whitney

d) Regla de decisión

- Si $P \leq \alpha$, entonces se acepta Hi
- Si $P > \alpha$, entonces se acepta Ho

e) Toma de decisión

Según los resultados obtenidos en la prueba U de Mann-Whitney en la figura 16, se concluye que al ser nuestro P-Valor ($< 0,001$) $< \alpha = 0,05$, entonces se acepta Hi, por lo tanto, existe diferencia significativa en los tiempos de respuesta entre servidores web Node.js y Apache utilizando datos de un chat de Twitch del canal xQc - 2023.

CAPÍTULO V
DISCUSIÓN

5.1. Aplicación de la tecnología encontrada

Los resultados revelan que Node.js supera a Apache en términos de capacidad de gestión de solicitudes y uso eficiente de recursos, especialmente en situaciones de alta concurrencia, como se evidenció en la carga del chat de Twitch del canal xQc. Estos descubrimientos son valiosos para guiar a ingenieros y desarrolladores en la selección y configuración de servidores para aplicaciones similares, donde la respuesta rápida a las solicitudes y la eficiencia en el uso de recursos son cruciales.

En resumen, la aplicación de los resultados de esta investigación se extiende más allá de la elección de Node.js o Apache. Proporciona conocimientos específicos para considerar las arquitecturas de cada servidor web según las demandas específicas, beneficiando a profesionales de IT, desarrolladores y diseñadores de plataformas de transmisión en tiempo real o aplicaciones similares.

5.2. Contraste con trabajos de investigación similares

Los resultados de nuestra investigación revelaron que tanto los servidores web Node.js como Apache fueron capaces de gestionar con éxito un total de 643 solicitudes HTTP en cada una de las 10 pruebas de carga realizadas, cada una con una duración de 60 segundos. Esto es significativo, ya que estas pruebas representan una muestra de un conjunto total de 643493 registros de mensajes procedentes de una transmisión en vivo en Twitch, específicamente del canal de xQc, que abarcó un período de 25 horas.

Comencemos discutiendo los resultados en relación con la dimensión "capacidad de gestión de solicitudes". Node.js demostró un tiempo de respuesta promedio de 13,70 milisegundos, mientras que Apache exhibió un tiempo promedio de 220.50 milisegundos. Esto indica una diferencia notoria en los tiempos de respuesta entre ambos servidores, con Node.js superando significativamente a Apache en este aspecto. En términos de tasa de transferencia, Node.js promedió 10,77 solicitudes por segundo, mientras que Apache alcanzó una media de 10,74 solicitudes por segundo, si bien Node.js mostró un rendimiento ligeramente superior, la diferencia es mínima. Es

importante destacar que ambos servidores web registraron un total de solicitudes fallidas nulas en todas las pruebas, lo que indica un alto nivel de fiabilidad en su funcionamiento.

En lo que respecta a la segunda dimensión, que se enfoca en la utilización de recursos, Node.js presentó un uso máximo de CPU con una media del 19,26 %, en contraste con Apache, que obtuvo una media de 194,8 %. En cuanto al uso de memoria, Node.js promedió un 1,29 %, mientras que Apache registró una media de 6,86 %. Estos resultados sugieren que Node.js utiliza de manera más eficiente tanto la CPU como la memoria en comparación con Apache, lo que podría indicar una ventaja en términos de uso de recursos.

Comencemos nuestra comparación con otras investigaciones afines, donde varios de estos estudios respaldan la tendencia que encontramos en nuestros resultados:

Vilhelmsson (2021) llevó a cabo pruebas que abarcaban dos categorías distintas: aquellas enfocadas en situaciones de intensidad computacional y otras dedicadas a evaluar la intensidad de E/S, que implica las operaciones de lectura y escritura en unidades de almacenamiento. Finalmente, realizó una combinación de ambas. Podemos contrastar los resultados específicos de las pruebas relacionadas con la intensidad de E/S realizadas por Vilhelmsson, ya que las aplicaciones alojadas en nuestros servidores web almacenan registros en bases de datos. Según los resultados de Vilhelmsson (2021), Apache mostró un mejor desempeño en términos de tasa de transferencia, tiempo de respuesta y uso de memoria. En nuestro caso, la tendencia es opuesta: Node.js exhibe un rendimiento superior en el uso de CPU y tiempo de respuesta, aunque no hay diferencias significativas en el uso de memoria y la tasa de transferencia.

En la investigación de Lei et al. (2014), que abordó un conjunto extenso de pruebas con hasta 10000 solicitudes por usuario y hasta 1000 usuarios, confirmamos y extendemos la conclusión de un rendimiento superior de Node.js en comparación con Apache. Resaltamos que estamos considerando las pruebas de consultas a la base de datos realizadas por Lei, donde evaluó situaciones de intensidad de E/S (entrada/salida), se refuerza nuestra comparación, ya que nuestras aplicaciones almacenan registros en una base de datos, lo que simula condiciones similares. Lei evaluó el rendimiento en términos de tiempo promedio por solicitud y solicitudes promedio por segundo. En los resultados de Lei et. al. (2014), Node.js demostró un rendimiento excepcional en la

gestión de solicitudes, fortaleciendo la posición de Node.js como una opción más eficiente frente a Apache en una variedad de situaciones y niveles de carga. Estos hallazgos así como los de la presente investigación respaldan y amplían la consistencia en la ventaja de rendimiento de Node.js sobre Apache en diferentes contextos de prueba.

Por otro lado Mao et. al. (2018) en su investigación realizó dos tipos de pruebas, uno para el cálculo del enésimo número fibonacci y consultas a la base de datos. Sus pruebas abarcaron condiciones de carga de hasta 200 solicitudes por usuario y un máximo de 100 usuarios. En los resultados de Mao et. al. (2018), Node.js exhibió un rendimiento notablemente mayor en la prueba donde se calculó el décimo número de Fibonacci, así como en la prueba de consultas a la base de datos, específicamente en términos de tiempo promedio por solicitud y solicitudes promedio por segundo. Estos resultados concuerdan con la presente investigación, reforzando la posición de Node.js como una opción más eficiente en comparación con Apache en diversas situaciones y cargas de trabajo.

Por otro lado Nguyen (2017) en su investigación abordó 20000 solicitudes concurrentes y midió el tiempo de respuesta, la tasa de transferencia, tasa de error, uso de CPU y uso de memoria. En los resultados de Nguyen (2017), Node.js destaca en la tasa de transferencia, tasa de error, uso de memoria y el uso de CPU en niveles altos de concurrencia por encima de Apache. Aunque en el caso del tiempo de respuesta ambos servidores web, Apache muestra mejores tiempos de respuesta, mientras aumenta la concurrencia, Node.js llega a superarlo. Estos resultados demuestran que la tendencia general del rendimiento de Node.js en situaciones de alta concurrencia persiste y supera a Apache, respaldando así la coherencia de los resultados a lo largo de diversas investigaciones como la presente.

En la investigación de Chhetri (2016), abordó un rango de 500 a 10000 solicitudes con concurrencias de 10 a 200, en los cuales midió el tiempo de respuesta, uso de CPU y memoria. Chhetri (2016) llevó a cabo pruebas de dos categorías distintas: aquellas enfocadas en situaciones de intensidad computacional y otras dedicadas a evaluar la intensidad de E/S, que implica las operaciones de lectura y escritura en unidades de almacenamiento. Además de que realizó estas pruebas en dos sistemas operativos diferentes: Windows 7 y Ubuntu 14.04.2. En los resultados de Chhetri (2016), en términos de tiempo de respuesta, tanto nuestra investigación como la de

Nimesh respaldan la superioridad de Node.js. La consistencia en este aspecto sugiere que Node.js mantiene un rendimiento más eficiente, independientemente de la variación en la carga de trabajo y la concurrencia. En cuanto al uso de CPU y memoria, nuestros resultados discrepan, según la prueba que realizó Nimesh referente al esfuerzo computacional, Apache demostró un mejor desenvolvimiento en el uso del CPU tanto en Windows y Ubuntu. Sin embargo en lo que refiere al uso de memoria, Apache gestiona mejor este recurso en el entorno de Ubuntu, mientras que en Windows no hay diferencia relevante.

CONCLUSIONES

Se llevó a cabo una evaluación comparativa entre los servidores web Node.js y Apache, centrándonos en la capacidad de gestión de solicitudes y el uso eficiente de recursos. Durante las pruebas bajo cargas reales, especialmente en el chat del stream de xQc en 2023, se concluyó que Node.js muestra un rendimiento significativamente superior. Por lo que concluimos lo siguiente:

Se evaluó la capacidad de gestión de solicitudes de ambos servidores web, aunque las diferencias en la tasa de transferencia y solicitudes fallidas no fueron significativas, destaca que la media del tiempo de respuesta de Node.js es aproximadamente un 1509 % mejor que el de Apache, evidenciando una diferencia sustancial a favor de Node.js.

Se midió el uso de recursos de ambos servidores web, donde se observó que Node.js gestiona de manera más eficiente la CPU en comparación con Apache. Aunque las diferencias en el uso de memoria no fueron considerables, la media del máximo uso de CPU por parte de Node.js fue aproximadamente un 911 % mejor que la de Apache. En consecuencia, se confirma una diferencia significativa a favor de Node.js en términos de eficiencia en la gestión de recursos.

Se evaluó el rendimiento de los servidores web Node.js y Apache utilizando datos de un chat de Twitch del canal xQc - 2023. Los resultados indican que Node.js supera a Apache en la capacidad de gestionar solicitudes web, especialmente en términos de tiempo de respuesta y uso de CPU. No se observaron diferencias significativas en la tasa de transferencia y el uso de memoria entre ambos servidores. En consecuencia, se confirma la hipótesis general planteada, demostrando una diferencia significativa en el rendimiento general entre Node.js y Apache. Estos hallazgos respaldan la elección de Node.js como una opción más eficiente en este escenario específico.

RECOMENDACIONES

Para unas futuras investigación del mismo tema se sugiere distintos puntos que vamos a enlistar a continuación::

a) Configuración óptima de Apache para alta concurrencia:

   Considerando los resultados obtenidos, se recomienda realizar investigaciones adicionales para determinar configuraciones óptimas de Apache que mejoren su rendimiento en situaciones de alta concurrencia. Explorar ajustes específicos en Apache podría contribuir a una mejora significativa en su capacidad de gestión de solicitudes.

b) Elección de tecnologías para aplicaciones en tiempo real:

   Con base en el rendimiento sobresaliente de Node.js en la gestión de solicitudes en tiempo real, se sugiere utilizar servidores web basados en la arquitectura SPED, como Node.js, para aplicaciones que requieren respuestas rápidas y eficientes bajo condiciones de alta concurrencia. La elección de tecnologías basadas en sockets puede ser especialmente beneficiosa para garantizar una comunicación eficiente entre el servidor y los clientes.

c) Explorar la escalabilidad de servidores web:

   Dada la importancia de la escalabilidad en entornos dinámicos, se recomienda evaluar a fondo la escalabilidad de los servidores web en diferentes contextos y cargas de trabajo. Comprender cómo escalan los servidores web bajo diversas condiciones puede proporcionar información valiosa para el diseño de arquitecturas que se adapten a las necesidades cambiantes de las aplicaciones.

d) Pruebas en otros tipos de aplicaciones de tiempo real:

   Para ampliar la aplicabilidad de los resultados, se sugiere realizar pruebas en otros tipos de aplicaciones en tiempo real que también involucran esfuerzo computacional. Esta diversificación en las pruebas permitirá obtener información más completa sobre el rendimiento de los servidores web en diversos escenarios y facilitará la toma de decisiones informada al seleccionar la tecnología adecuada para diferentes casos de uso.

e) Pruebas con escenarios de más carga de trabajo:

   Además de las cargas de trabajo derivadas de un chat de Twitch, se sugiere explorar escenarios que impliquen cargas más pesadas y demandantes

para los servidores web. Ejemplos incluyen aplicaciones de trading en tiempo real, monitoreo de redes, eventos de redes sociales o juegos en línea. Estas pruebas proporcionarán información valiosa sobre el rendimiento de los servidores web en situaciones que requieren un mayor esfuerzo computacional y pueden revelar sus límites y fortalezas en entornos más desafiantes.

REFERENCIAS

Aiken, L. R. (1980). Content validity and reliability of single items or questionnaires. *Educational and Psychological Measurement*, *40*(4). https://doi.org/10.1177/001316448004000419

Arcaya Arhuata, L. E. (2012). *Sistema de información clientes/servidor con tecnología web para los procesos de matrículas y trámites de certificación de la Escuela Nacional de Estadística e Informática del INEI – Tacna – 2011* [Tesis de Pregrado, Universidad Nacional Jorge Basadre Grohmann]. http://tesis.unjbg.edu.pe/handle/UNJBG/2431

Cabero Almenara, J., & Llorente Cejudo, C. (2013). La aplicación del juicio de experto como técnica de evaluación de las tecnologías de la información y comunicación (TIC). *Eduweb*.

Claypool, M., Farrington, D., & Muesch, N. (2015). Measurement-based analysis of the video characteristics of Twitch.tv. 2015 IEEE Games Entertainment Media Conference (GEM), 1-4. https://doi.org/10.1109/GEM.2015.7377227

Chhetri, N. (2016). A Comparative Analysis of Node.js (Server-Side JavaScript). *Culminating Projects in Computer Science and Information Technology*. https://repository.stcloudstate.edu/csit_etds/5

Choi, G. S., Kim, J.-H., Ersoz, D., & Das, C. R. (2005). *A multi-threaded PIPELINED Web server architecture for SMP/SoC machines*. https://doi.org/10.1145/1060745.1060851

Córdova Molina, A. F. (2017). Análisis de servicios web en redes SDN. *Santiago de Chile*, *6*.

Cubas Fernández, L. F. (2019). Análisis comparativo del rendimiento y el esfuerzo mediante pruebas de carga en servidores web. *Anuales de la Universidad de Chile*, *0*(13). https://repositorio.uss.edu.pe/handle/20.500.12802/6264

DelBono, E. (2017). *Node. Js Succinctly* (CreateSpace Independent Publishing Platform, Ed.). https://www.syncfusion.com/succinctly-free-ebooks/nodejs

Dusk. (2022). *Twitch | 25h xQc VOD Chat | Kaggle*. https://www.kaggle.com/datasets/dusk123/25h-xqc-vod-chat-2022-08-11

Escurra Mayaute, L. M. (1969). Cuantificación de la validez de contenido por criterio de jueces. *Revista de Psicología*, *6*(1-2), 103-111. https://doi.org/10.18800/psico.198801-02.008

Gelbmann, M. (2022). *Usage Statistics and Market Share of Node.js*. Web Technologies of the Year 2022. https://w3techs.com/technologies/details/ws-nodejs

Gokhale, S. S., Vandal, P. J., & Lu, J. (2006). Performance and reliability analysis of web server software. *Proceedings - 12th Pacific Rim International Symposium on Dependable Computing, PRDC 2006*. https://doi.org/10.1109/PRDC.2006.50

Guisande González, C., Vaamonde Liste, A., & Barreiro Felpeto, A. (2013). Tratamiento de datos con R, STATISTICA y SPSS. España: Díaz de Santos

Hamilton, W. A., Garretson, O., & Kerne, A. (2014). Streaming on twitch: Fostering participatory communities of play within live mixed media. *Conference on Human Factors in Computing Systems - Proceedings*. https://doi.org/10.1145/2556288.2557048

Hernández Sampieri R, Fernández Collado C, & Baptista Lucio M del P. (2015). Metodología de la Investigación -sampieri- 6ta EDICION. En *McGraw-Hill, editor. México; 2015. 634 p.*

Herron, D. (2020). Node.js Web Development: Server-side web development made easy with Node 14 using practical examples. *Amazone: fifth Edition.*

Hu, J. C., Mungee, S., & Schmidt, D. C. (1998). Techniques for developing and measuring high performance web servers over high speed networks. *Proceedings - IEEE INFOCOM*, *3*. https://doi.org/10.1109/infcom.1998.662936

IBM. (2023). *Monitoring performance using the Windows Performance Monitor - IBM Documentation*. Recuperado el 01 de septiembre del 2023, de https://www.ibm.com/docs/en/db2/11.5?topic=server-monitoring-performance-using-windows-performance-monitor

Internet Live Stats. (s. f.). *Total number of Websites*. Recuperado el 30 de agosto de 2023, de https://www.internetlivestats.com/total-number-of-websites/

Jader, O. H., Zeebaree, S. R. M., & Zebari, R. R. (2019). A state of art survey for web server performance measurement and load balancing mechanisms. *International Journal of Scientific and Technology Research*, *8*(12).

Kunda, D., Chihana, S., Muwanei, S., & Sinyinda, M. (2017). Web Server Performance of Apache and Nginx: A Systematic Literature Review. *Computer Engineering and Intelligent Systems*, *8*(2).

Laurie, B., & Laurie, P. (2003). *Apache: The Definitive Guide* (S. Laurent, Ed.; 3.ª ed.). O'Reilly Media, Inc.

Lei, K., Ma, Y., & Tan, Z. (2014). Performance Comparison and Evaluation of Web Development Technologies in PHP, Python, and Node.js. *2014 IEEE 17th International Conference on Computational Science and Engineering*, 661–668. https://doi.org/10.1109/CSE.2014.142

Liu, G., Xu, J., Wang, C., & Zhang, J. (2018). A performance comparison of HTTP servers in a 10G/40G network. *ACM International Conference Proceeding Series*. https://doi.org/10.1145/3220199.3220216

Luján Mora, S. (2002). *Programación de aplicaciones web historia, principios básicos y clientes web* (Editorial Club Universitario, Ed.). http://hdl.handle.net/10045/16995

Mao, X., Ludwig, S., Kong, J., Gao, Z., & Nygard, K. (2018). *Comparison between Symfony, ASP.NET MVC, And Node.js Express for Web Development*. https://library.ndsu.edu/ir/handle/10365/28191

Martín Arribas, M. C. (2004). Diseño y validación de cuestionarios. *Matronas Profesión*, *5*(17).

MDN. (s. f.). *¿Qué es un servidor WEB?* Aprende desarrollo web. Recuperado el 30 de Agosto de 2023, de https://developer.mozilla.org/es/docs/Learn/Common_questions/Web_mechanics/What_is_a_web_server

Microsoft. (2012). *Performance and Reliability Monitoring Getting Started Guide for Windows Server 2008*. Recuperado el 01 de septiembre del 2023, de https://learn.microsoft.com/es-es/previous-versions/windows/it-pro/windows-server-2008-r2-and-2008/cc771692(v=ws.10

Needleman, S. E. (2015). *Twitch's Viewers Reach 100 Million a Month*. The Wall Street Journal. Recuperado el 02 de octubre del 2023, de https://www.wsj.com/articles/BL-DGB-40239

Netcraft. (2023). *July 2023 Web Server Survey*. Web Server Survey. Recuperado el 14 de septiembre del 2023, de https://www.netcraft.com/blog/july-2023-web-server-survey/

Nguyen, V. N. (2017). Comparative Performance Evaluation of Web Servers. *VNU Journal of Science: Comp. Science & Com. Eng*, *31*(3).

OpenJS Foundation and Node.js contributors. (s. f.). *Node.js*. Recuperado el 30 de agosto de 2023, de https://nodejs.org/es

Pai, V. S., Druschel, P., & Zwaenepoel, W. (1999). Flash: An efficient and portable Web server. *Proceedings of the 1999 USENIX Annual Technical Conference*.

Penfield, R. D., & Giacobbi, Jr. , P. R. (2004). Applying a Score Confidence Interval to Aiken's Item Content-Relevance Index. Measurement in Physical Education and Exercise Science, 8(4), 213-225. https://doi.org/10.1207/s15327841mpee0804_3

Raman, A., Tyson, G., & Sastry, N. (2018). Facebook (A)Live? Are Live Social Broadcasts Really Broadcasts? *The Web Conference 2018 - Proceedings of the World Wide Web Conference, WWW 2018*. https://doi.org/10.1145/3178876.3186061

Ramírez Azanza, M. G. (2019). *Análisis comparativo de rendimiento a servidores web de distribución libre utilizando apache benchmark* [Universidad Técnica de Machala]. Recuperado el 05 de septiembre del 2023, de http://repositorio.utmachala.edu.ec/handle/48000/14567

Ríos, A. R., & Peña, A. M. P. (2020). Estadística inferencial. Elección de una prueba estadística no paramétrica en investigación científica. Horizonte de la Ciencia, 10(19), 191-208. https://doi.org/10.26490/uncp.horizonteciencia.2020.19.597

Sjöblom, M., Törhönen, M., Hamari, J., & Macey, J. (2019). The ingredients of Twitch streaming: Affordances of game streams. *Computers in Human Behavior*, *92*. https://doi.org/10.1016/j.chb.2018.10.012

The Apache Software Foundation. (s.f.). Apache HTTP Server Version 2.4 Documentation. Recuperado el 21 de noviembre del 2023, de https://httpd.apache.org/docs/2.4/

The Apache Software Foundation. (s.f.). Apache JMeter. Recuperado el 21 de noviembre del 2023, de https://jmeter.apache.org/

Tilkov, S., & Vinoski, S. (2010). Node.js: Using JavaScript to build high-performance network programs. *IEEE Internet Computing*, *14*(6). https://doi.org/10.1109/MIC.2010.145

Twitch Developers. (s. f.). *Twitch API*. Recuperado el 30 de agosto de 2023, de https://dev.twitch.tv/docs/api/

Vara Horna, A. A. (2012). Siete Pasos para una Tesis Exitosa: Desde la idea inicial hasta la sustentación. Un método efectivo para las ciencias empresariales. *Instituto de Investigación de la Facultad de Ciencias Administrativas y Recursos Humanos. Universidad de San Martín de Porres*, *January 2012*.

Vilhelmsson, I. (2021). *A Performance Comparison of an Event-Driven Node.js Web Server and Multi-Threaded Web Servers* [School of Electrical Engineering and Computer Science]. https://www.diva-portal.org/smash/record.jsf?pid=diva2%3A1605998&dswid=3725

Warner, J. (s.f.). top(1) - Linux man page. Recuperado el 20 de noviembre del 2023, de https://linux.die.net/man/1/top

Wells, L., Christensen, S., Kristensen, L. M., & Mortensen, K. H. (2001). Simulation based performance analysis of web servers. *Proceedings - 9th International Workshop on Petri Nets and Performance Models, PNPM 2001*. https://doi.org/10.1109/PNPM.2001.953356

ANEXOS

Anexo 1

Matriz de consistencia

Problema general **Principal** ¿Cuál es la diferencia entre el rendimiento de los servidores web Node.js y Apache utilizando datos de un chat de Twitch del canal xQc, 2023? **Problemas derivados** ¿Cuál es la diferencia en la capacidad de gestión de solicitudes entre los servidores web Node.js y Apache utilizando datos de un chat de Twitch del canal xQc, 2023? ¿Cuál es la diferencia en uso de recursos entre los servidores web Node.js y Apache utilizando datos de un chat de Twitch del canal xQc, 2023?	**Objetivos** **Objetivo general** Evaluar la diferencia del rendimiento entre servidores web Node.js y Apache utilizando datos de un chat de Twitch del canal xQc, 2023. **Objetivos específicos** Evaluar la capacidad de gestión de solicitudes de los servidores web Node.js y Apache utilizando datos de un chat de Twitch del canal xQc, 2023. Medir el uso de recursos entre servidores web Node.js y Apache utilizando datos de un chat de Twitch del canal xQc, 2023.	**Hipótesis.** **Hipótesis general** Existe diferencia significativa en el rendimiento entre servidores web Node.js y Apache utilizando datos de un chat de Twitch del canal xQc, 2023. **Hipótesis específicas.** Existe diferencia significativa en la capacidad de gestión de solicitudes entre servidores web Node.js y Apache utilizando datos de un chat de Twitch del canal xQc, 2023. Existe diferencia significativa en uso de recursos entre servidores web Node.js y Apache utilizando datos de un chat de Twitch del canal xQc, 2023.

Variables e indicadores Para demostrar y comprobar la hipótesis anteriormente formulada, la operacionalización, determinando las variables e indicadores que a continuación se mencionan: ***Variable**: Rendimiento del servidor web.* **Dimensión 1:** Capacidad de gestión de solicitudes ***Indicadores:*** *Tiempo de respuesta* *x1* *Tasa de transferencia* *x2* *Solicitudes fallidas* *x3* **Dimensión 2:** Utilización de recursos ***Indicadores:*** *Uso de CPU* *x4* *Uso de memoria* *x5*	**Metodología** **Nivel de la investigación**: Descriptivo comparativo. **Enfoque la investigación**: Cuantitativo **Diseño de la investigación**: No experimental **Población:** 643493 registros de mensajes de 54668 usuarios del chat de Twitch del canal xQc. **Muestreo:** Por conveniencia, 643 registros del chat de Twitch del canal xQc. **Técnicas de recolección de datos**: Observación. **Instrumentos de recolección de datos:** Guía de observación.

Anexo 2

Operacionalización de la variable rendimiento del servidor web

Variable	Definición conceptual	Definición operacional	Dimensión	Indicadores
Rendimiento del servidor web	Siguiendo la perspectiva delineada en el estudio de Hu et al. (1998), el rendimiento del servidor web se define como la capacidad del servidor para gestionar eficientemente solicitudes web y entregar respuestas rápidas a los usuarios.	"Las medidas de rendimiento incluyen el tiempo de respuesta y la tasa de servicio, el uso de la memoria, la utilización de la CPU, entre otros" (Kunda et al., 2017).	Uso de recursos	- Uso de CPU - Uso de memoria
			Capacidad de gestión de solicitudes	- Tiempo de respuesta - Tasa de transferencia - Solicitudes fallidas

Anexo 3

Instrumentos de recolección de datos: Guía de observación

Dimensión	Indicador	Métricas	Node.js	Apache
Capacidad de gestión de solicitudes	Tiempo de respuesta	Tiempo de respuesta (ms) (Solicitud 1)		
		Tiempo de respuesta (ms) (Solicitud 2)		
		…	…	…
		Tiempo de respuesta (ms) (Solicitud n)		
	Tasa de transferencia	Número de solicitudes por segundo. (solicitudes/s)		
	Solicitudes fallidas	Número de registros fallidos		
Uso de recursos	Uso de memoria	Máximo porcentaje de memoria utilizada (%)		
	Uso de CPU	Máximo porcentaje de CPU utilizada (%)		

Anexo 4

Validación del instrumento por juicio de expertos

NOMBRE DEL INSTRUMENTO: "GUÍA DE OBSERVACIÓN PARA LA EVALUACIÓN DE RENDIMIENTO DE LOS SERVIDORES WEB NODE.JS Y APACHE."

OBJETIVO: Registrar el tiempo de respuesta, tasa de transferencia, solicitudes fallidas, uso de CPU y memoria de las **solicitudes realizadas** a los servidores web Node.js y Apache durante los tests.

MALO = 1	REGULAR = 2	BUENO = 3

Escala de valoración:

Indicador	Métricas	Node.js	Apache	**Valoración**	Observación
Tiempo de respuesta	Tiempo de respuesta (ms) (Solicitud n)				
Tasa de transferencia	Número de solicitudes por segundo. (solicitudes/s)				
Solicitudes fallidas	Número de solicitudes fallidas				
Uso de memoria	Máximo porcentaje de memoria utilizada (%)				
Uso de CPU	Máximo porcentaje de CPU utilizada (%)				

Apellidos y nombres del juez evaluador: ……

Especialidad del juez evaluador: ……

FIRMA DEL JUEZ EVALUADOR

Anexo 5

Valoraciones de jueces al instrumento de recolección de datos

VALIDACIÓN DEL INSTRUMENTO POR JUICIO DE EXPERTOS

NOMBRE DEL INSTRUMENTO: "GUÍA DE OBSERVACIÓN PARA LA EVALUACIÓN DE RENDIMIENTO DE LOS SERVIDORES WEB NODE.JS Y APACHE."

OBJETIVO: Registrar el tiempo de respuesta, tasa de transferencia, solicitudes fallidas, uso de cpu y memoria de las **solicitudes realizadas** a los servidores web Node.js y Apache durante los tests.

Escala de valoración:

MALO = 1	REGULAR = 2	BUENO = 3

Indicador	Métricas	Node.js	Apache	**Valoración**	Observación
Tiempo de respuesta	Tiempo de respuesta (ms) (Solicitud n)			2	
Tasa de transferencia	Número de solicitudes por segundo. (solicitudes/s)			3	
Solicitudes fallidas	Número de solicitudes fallidas			3	
Uso de memoria	Máximo porcentaje de memoria utilizada (%)			3	
Uso de CPU	Máximo porcentaje de CPU utilizada (%)			3	

Apellidos y nombres del juez evaluador: Supo Gavancho, Karin Yanet.

Especialidad del juez evaluador: Ingeniera en Informática y Sistemas, Doctora en Ciencias de la Educación.

Dra. Karin Y. Supo Gavancho
Ingeniería y Sistemas
CIP. 75696

FIRMA DEL EVALUADOR

VALIDACIÓN DEL INSTRUMENTO POR JUICIO DE EXPERTOS

NOMBRE DEL INSTRUMENTO: "GUÍA DE OBSERVACIÓN PARA LA EVALUACIÓN DE RENDIMIENTO DE LOS SERVIDORES WEB NODE.JS Y APACHE."

OBJETIVO: Registrar el tiempo de respuesta, tasa de transferencia, solicitudes fallidas, uso de cpu y memoria de las **solicitudes realizadas** a los servidores web Node.js y Apache durante los tests.

Escala de valoración:

MALO = 1	REGULAR = 2	BUENO = 3

Indicador	Métricas	Node.js	Apache	**Valoración**	Observación
Tiempo de respuesta	Tiempo de respuesta (ms) (Solicitud n)			3	
Tasa de transferencia	Número de solicitudes por segundo. (solicitudes/s)			3	
Solicitudes fallidas	Número de solicitudes fallidas			3	
Uso de memoria	Máximo porcentaje de memoria utilizada (%)			3	
Uso de CPU	Máximo porcentaje de CPU utilizada (%)			3	

Apellidos y nombres del juez evaluador: Barraza Vizcarra, Hugo Manuel

Especialidad del juez evaluador: Ingeniero en Informática y Sistemas, Maestro en Ciencias con mención en Ingeniería de Sistemas e Informática

FIRMA DEL JUEZ EVALUADOR

VALIDACIÓN DEL INSTRUMENTO POR JUICIO DE EXPERTOS

NOMBRE DEL INSTRUMENTO: "GUÍA DE OBSERVACIÓN PARA LA EVALUACIÓN DE RENDIMIENTO DE LOS SERVIDORES WEB NODE.JS Y APACHE."

OBJETIVO: Registrar el tiempo de respuesta, tasa de transferencia, solicitudes fallidas, uso de cpu y memoria de las **solicitudes realizadas** a los servidores web Node.js y Apache durante los tests.

Escala de valoración:

MALO = 1	REGULAR = 2	BUENO = 3

Indicador	Métricas	Node.js	Apache	**Valoración**	Observación
Tiempo de respuesta	Tiempo de respuesta (ms) (Solicitud n)			3	
Tasa de transferencia	Número de solicitudes por segundo. (solicitudes/s)			3	
Solicitudes fallidas	Número de solicitudes fallidas			3	
Uso de memoria	Máximo porcentaje de memoria utilizada (%)			3	
Uso de CPU	Máximo porcentaje de CPU utilizada (%)			3	

Apellidos y nombres del juez evaluador: Acero Charaña, Carlos Hernan

Especialidad del juez evaluador: Ingeniero en informatica y Sistemas

FIRMA DEL JUEZ EVALUADOR

VALIDACIÓN DEL INSTRUMENTO POR JUICIO DE EXPERTOS

NOMBRE DEL INSTRUMENTO: "GUÍA DE OBSERVACIÓN PARA LA EVALUACIÓN DE RENDIMIENTO DE LOS SERVIDORES WEB NODE.JS Y APACHE."

OBJETIVO: Registrar el tiempo de respuesta, tasa de transferencia, solicitudes fallidas, uso de cpu y memoria de las **solicitudes realizadas** a los servidores web Node.js y Apache durante los tests.

Escala de valoración:

MALO = 1	REGULAR = 2	BUENO = 3

Indicador	Métricas	Node.js	Apache	**Valoración**	Observación
Tiempo de respuesta	Tiempo de respuesta (ms) (Solicitud n)			3	
Tasa de transferencia	Número de solicitudes por segundo. (solicitudes/s)			3	
Solicitudes fallidas	Número de solicitudes fallidas			3	
Uso de memoria	Máximo porcentaje de memoria utilizada (%)			3	
Uso de CPU	Máximo porcentaje de CPU utilizada (%)			3	

Apellidos y nombres del juez evaluador: Osco Mamani, Erbert Francisco

Especialidad del juez evaluador: Ingeniero de Sistemas, Magister en Informática
Doctor en Administración de la Educación

FIRMA DEL JUEZ EVALUADOR

Dr. Erbert F. Osco Mamani

VALIDACIÓN DEL INSTRUMENTO POR JUICIO DE EXPERTOS

NOMBRE DEL INSTRUMENTO: "GUÍA DE OBSERVACIÓN PARA LA EVALUACIÓN DE RENDIMIENTO DE LOS SERVIDORES WEB NODE.JS Y APACHE."

OBJETIVO: Registrar el tiempo de respuesta, tasa de transferencia, solicitudes fallidas, uso de cpu y memoria de las **solicitudes realizadas** a los servidores web Node.js y Apache durante los tests.

Escala de valoración:

MALO = 1	REGULAR = 2	BUENO = 3

Indicador	Métricas	Node.js	Apache	**Valoración**	Observación
Tiempo de respuesta	Tiempo de respuesta (ms) (Solicitud n)			3	
Tasa de transferencia	Número de solicitudes por segundo. (solicitudes/s)			3	
Solicitudes fallidas	Número de solicitudes fallidas			3	
Uso de memoria	Máximo porcentaje de memoria utilizada (%)			3	
Uso de CPU	Máximo porcentaje de CPU utilizada (%)			3	

Apellidos y nombres del juez evaluador: Taya Acosta, Edgar Aurelio

Especialidad del juez evaluador: Ingeniero en Informática y Sistemas, Maestro en Ciencias con mención en Ingeniería de Sistemas e Informática Doctor en ciencias de la educación

FIRMA DEL JUEZ EVALUADOR

Anexo 6

Código fuente de python para seleccionar la muestra de la investigación y resultados

Figura 1

Código en python para obtener el total de mensajes por hora de la transmisión de xQc.

get_total_messages_per_hour.py

xqc > get_total_messages_per_hour.py

```python
import pandas as pd

# Cargar datos desde un archivo CSV llamado 'xqc25.csv' en un DataFrame
df = pd.read_csv('xqc25.csv')

# Convertir la columna 'delta' a un formato de tiempo (timedelta)
df['delta'] = pd.to_timedelta(df['delta'])

# Agrupar los registros por hora redondeando hacia abajo (floor) en la columna 'delta'
hourly_counts = df.groupby(df['delta'].dt.floor('H')).size().reset_index()

# Renombrar las columnas resultantes a 'Hour' y 'Count'
hourly_counts.columns = ['Hour', 'Count']

# Ordenar los resultados en orden descendente (mayor a menor) según la columna 'Count'
hourly_counts = hourly_counts.sort_values(by='Count', ascending=False)

# Imprimir el DataFrame con los recuentos de registros por hora
print(hourly_counts)
```

Figura 2

Resultado del código en python para obtener el total de mensajes por hora de la transmisión de xQc y ordenado de manera descendente.

```
cristian@DESKTOP-MDK9PDC:/mnt/d/code/python/xqc$ python3 get_total_messages_per_hour.py
              Hour  Count
3  0 days 03:00:00  34787
4  0 days 04:00:00  34064
0  0 days 00:00:00  32332
22 0 days 22:00:00  30798
2  0 days 02:00:00  29769
8  0 days 08:00:00  28761
7  0 days 07:00:00  28732
5  0 days 05:00:00  27093
24 1 days 00:00:00  26634
23 0 days 23:00:00  26607
6  0 days 06:00:00  26395
19 0 days 19:00:00  25575
1  0 days 01:00:00  25512
21 0 days 21:00:00  25238
17 0 days 17:00:00  23457
15 0 days 15:00:00  23357
10 0 days 10:00:00  22982
20 0 days 20:00:00  22697
18 0 days 18:00:00  22062
16 0 days 16:00:00  21941
9  0 days 09:00:00  21346
11 0 days 11:00:00  20382
14 0 days 14:00:00  20248
13 0 days 13:00:00  20236
12 0 days 12:00:00  17957
25 1 days 01:00:00   8273
```

Figura 3

Código en python para obtener el total de mensajes por cada 10 minutos dentro de la hora 3 de la transmisión de xQc.

get_total_messages_per_ten_minute.py

```python
import pandas as pd

# Cargar datos desde el archivo CSV 'hour_3_records.csv' en un DataFrame
df = pd.read_csv('hour_3_records.csv')

# Convertir la columna 'delta' a un formato de tiempo (timedelta)
df['delta'] = pd.to_timedelta(df['delta'])

# Calcular el número de registros en intervalos de 10 minutos
ten_minute_intervals = df.groupby(df['delta'].dt.floor('10T')).size().reset_index()
ten_minute_intervals.columns = ['Interval', 'Count']

print("Registros por intervalo de 10 minutos:")
print(ten_minute_intervals)

```

Figura 4

Resultado del código en python para obtener el total de mensajes por cada 10 minutos dentro de la hora 3 de la transmisión de xQc.

```
cristian@DESKTOP-MDK9PDC:/mnt/d/code/python/xqc$ python3 get_total_messages_per_ten_minute.py
Registros por intervalo de 10 minutos:
         Interval  Count
0 0 days 03:00:00   5816
1 0 days 03:10:00   5862
2 0 days 03:20:00   5775
3 0 days 03:30:00   5896
4 0 days 03:40:00   5763
5 0 days 03:50:00   5675
```

Figura 5

Código en python para obtener el total de mensajes por minuto dentro de la hora 3:30 hasta 3:40 de la transmisión de xQc.

get_total_messages_per_minute.py

```python
import pandas as pd

# Cargar datos desde el archivo CSV 'minute_30_to_40_records.csv' en un DataFrame
df = pd.read_csv('minute_30_to_40_records.csv')

# Convertir la columna 'delta' a un formato de tiempo (timedelta)
df['delta'] = pd.to_timedelta(df['delta'])

# Calcular el número de registros en intervalos de 1 minuto
one_minute_intervals = df.groupby(df['delta'].dt.floor('1T')).size().reset_index()
one_minute_intervals.columns = ['Interval', 'Count']

print("\nRegistros por intervalo de 1 minuto:")
print(one_minute_intervals)
```

Figura 6

Resultado del código en python para obtener el total de mensajes por minuto dentro de la hora 3:30 hasta 3:40 de la transmisión de xQc.

```
cristian@DESKTOP-MDK9PDC:/mnt/d/code/python/xqc$ python3 get_total_messages_per_minute.py

Registros por intervalo de 1 minuto:
         Interval  Count
0 0 days 03:30:00    573
1 0 days 03:31:00    600
2 0 days 03:32:00    583
3 0 days 03:33:00    550
4 0 days 03:34:00    627
5 0 days 03:35:00    582
6 0 days 03:36:00    574
7 0 days 03:37:00    577
8 0 days 03:38:00    643
9 0 days 03:39:00    587
```

Figura 7

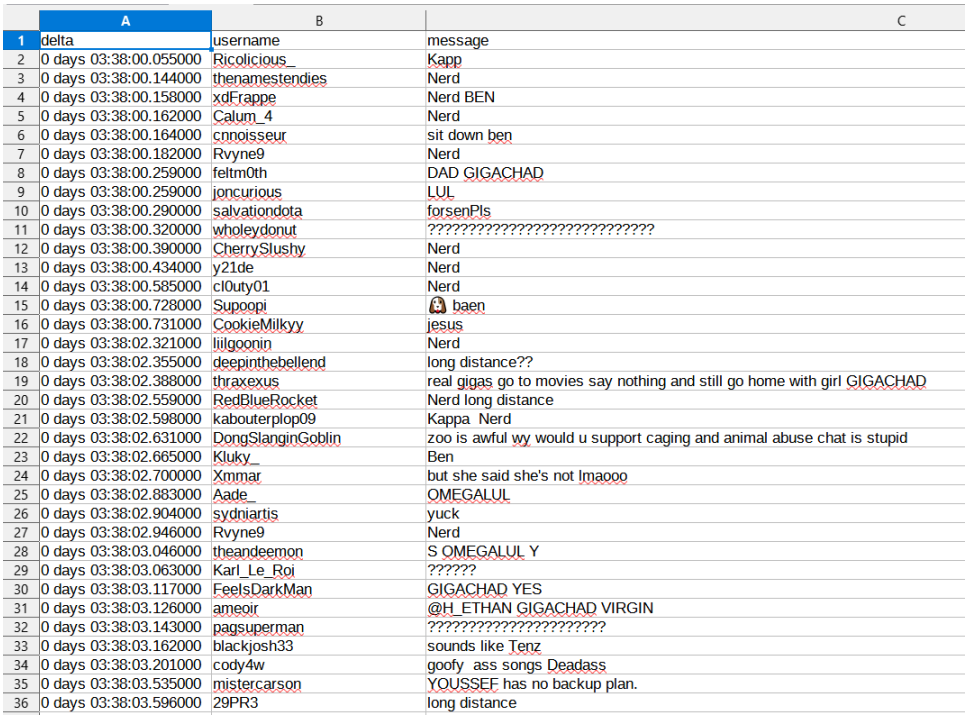
Código en python para exportar los mensajes por minuto de la hora 3:38 - 3:39 de la transmisión de xQc.

1_minute.py

```python
import pandas as pd

# Leer datos desde el archivo CSV 'minute_30_to_40_records.csv' en un DataFrame
df = pd.read_csv('minute_30_to_40_records.csv')

# Convertir la columna 'delta' a un formato de tiempo (timedelta)
df['delta'] = pd.to_timedelta(df['delta'])

# Extraer el minuto de la columna 'delta'
df['minute'] = df['delta'].dt.components['minutes']

# Agrupar por minuto y contar registros
minute_counts = df.groupby('minute').size().reset_index()
minute_counts.columns = ['Minute', 'Count']

# Encontrar el minuto con el mayor número de mensajes
max_minute = minute_counts['Minute'].iloc[minute_counts['Count'].idxmax()]

# Filtrar registros del minuto con mayor mensajes
max_minute_records = df[df['minute'] == max_minute]

# Eliminar la columna 'minute' antes de exportar
max_minute_records = max_minute_records.drop(columns=['minute'])

# Calcular la cantidad de usuarios únicos durante cada minuto
unique_users_per_minute = df.groupby('minute')['username'].nunique().reset_index()
unique_users_per_minute.columns = ['Minute', 'Unique Users']

# Imprimir una tabla con cada minuto, la cantidad de mensajes y usuarios únicos
minute_info = pd.merge(minute_counts, unique_users_per_minute, on='Minute')
print("Minuto | Cantidad de Mensajes | Usuarios Únicos")
print(minute_info.to_string(index=False))
```

Figura 8

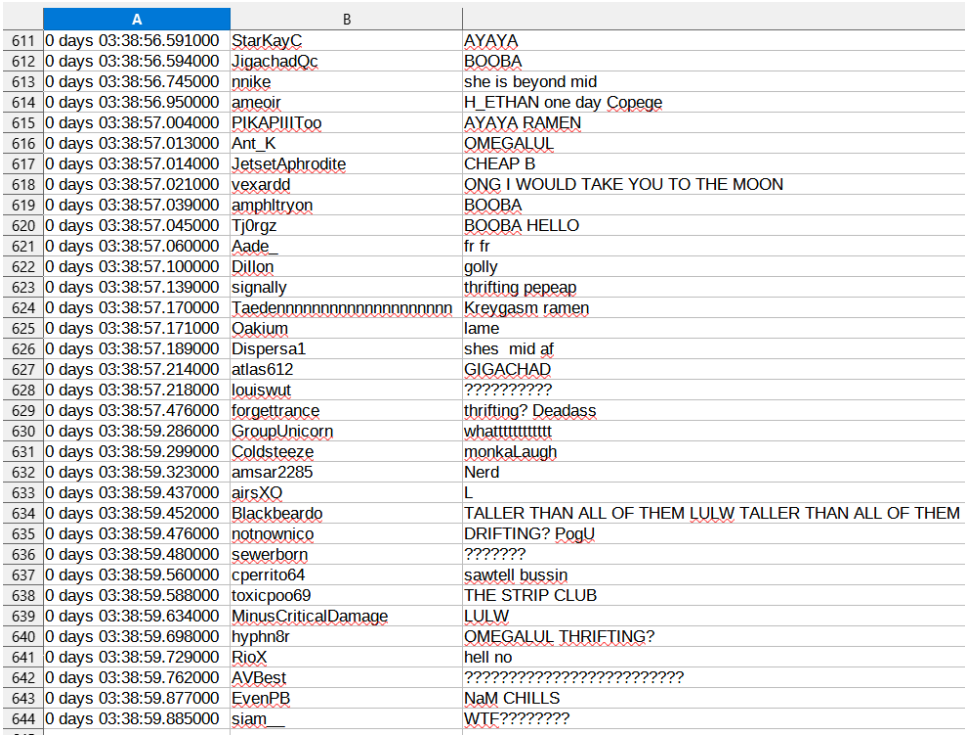
Primeros mensajes exportados del minuto de la hora 3:38 - 3:39 de la transmisión de xQc.



	A	B	C
1	delta	username	message
2	0 days 03:38:00.055000	Ricolicious_	Kapp
3	0 days 03:38:00.144000	thenamestendies	Nerd
4	0 days 03:38:00.158000	xdFrappe	Nerd BEN
5	0 days 03:38:00.162000	Calum_4	Nerd
6	0 days 03:38:00.164000	cnnoisseur	sit down ben
7	0 days 03:38:00.182000	Rvyne9	Nerd
8	0 days 03:38:00.259000	feltm0th	DAD GIGACHAD
9	0 days 03:38:00.259000	joncurious	LUL
10	0 days 03:38:00.290000	salvationdota	forsenPls
11	0 days 03:38:00.320000	wholeydonut	????????????????????????????????
12	0 days 03:38:00.390000	CherrySlushy	Nerd
13	0 days 03:38:00.434000	y21de	Nerd
14	0 days 03:38:00.585000	cl0uty01	Nerd
15	0 days 03:38:00.728000	Supoopi	baen
16	0 days 03:38:00.731000	CookieMilkyy	jesus
17	0 days 03:38:02.321000	lilgoonin	Nerd
18	0 days 03:38:02.355000	deepinthebellend	long distance??
19	0 days 03:38:02.388000	thraxexus	real gigas go to movies say nothing and still go home with girl GIGACHAD
20	0 days 03:38:02.559000	RedBlueRocket	Nerd long distance
21	0 days 03:38:02.598000	kabouterplop09	Kappa Nerd
22	0 days 03:38:02.631000	DongSlanginGoblin	zoo is awful wy would u support caging and animal abuse chat is stupid
23	0 days 03:38:02.665000	Kluky_	Ben
24	0 days 03:38:02.700000	Xmmar	but she said she's not lmaooo
25	0 days 03:38:02.883000	Aade_	OMEGALUL
26	0 days 03:38:02.904000	sydniartis	yuck
27	0 days 03:38:02.946000	Rvyne9	Nerd
28	0 days 03:38:03.046000	theandeemon	S OMEGALUL Y
29	0 days 03:38:03.063000	Karl_Le_Roi	??????
30	0 days 03:38:03.117000	FeelsDarkMan	GIGACHAD YES
31	0 days 03:38:03.126000	ameoir	@H_ETHAN GIGACHAD VIRGIN
32	0 days 03:38:03.143000	pagsuperman	??????????????????????
33	0 days 03:38:03.162000	blackjosh33	sounds like Tenz
34	0 days 03:38:03.201000	cody4w	goofy ass songs Deadass
35	0 days 03:38:03.535000	mistercarson	YOUSSEF has no backup plan.
36	0 days 03:38:03.596000	29PR3	long distance

Figura 9

Últimos mensajes exportados del minuto de la hora 3:38 - 3:39 de la transmisión de xQc, un total de 643 registros.



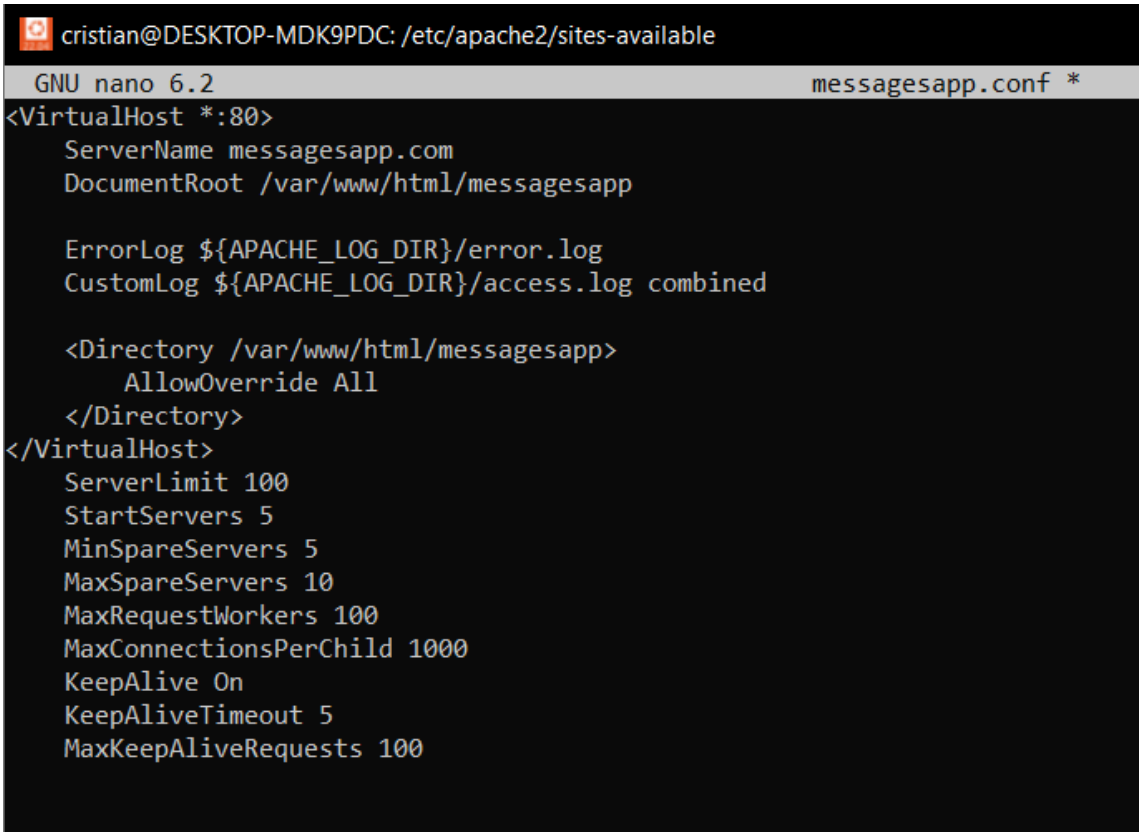
	A	B	
611	0 days 03:38:56.591000	StarKayC	AYAYA
612	0 days 03:38:56.594000	JigachadQc	BOOBA
613	0 days 03:38:56.745000	nnike	she is beyond mid
614	0 days 03:38:56.950000	ameoir	H_ETHAN one day Copege
615	0 days 03:38:57.004000	PIKAPIIITOO	AYAYA RAMEN
616	0 days 03:38:57.013000	Ant_K	OMEGALUL
617	0 days 03:38:57.014000	JetsetAphrodite	CHEAP B
618	0 days 03:38:57.021000	vexardd	ONG I WOULD TAKE YOU TO THE MOON
619	0 days 03:38:57.039000	amphltryon	BOOBA
620	0 days 03:38:57.045000	Tj0rgz	BOOBA HELLO
621	0 days 03:38:57.060000	Aade_	fr fr
622	0 days 03:38:57.100000	Dillon	golly
623	0 days 03:38:57.139000	signally	thrifting pepeap
624	0 days 03:38:57.170000	Taedennnnnnnnnnnnnnnnnnn	Kreygasm ramen
625	0 days 03:38:57.171000	Oakium	lame
626	0 days 03:38:57.189000	Dispersa1	shes mid af
627	0 days 03:38:57.214000	atlas612	GIGACHAD
628	0 days 03:38:57.218000	louiswut	??????????
629	0 days 03:38:57.476000	forgettrance	thrifting? Deadass
630	0 days 03:38:59.286000	GroupUnicorn	whatttttttttt
631	0 days 03:38:59.299000	Coldsteeze	monkaLaugh
632	0 days 03:38:59.323000	amsar2285	Nerd
633	0 days 03:38:59.437000	airsXO	L
634	0 days 03:38:59.452000	Blackbeardo	TALLER THAN ALL OF THEM LULW TALLER THAN ALL OF THEM
635	0 days 03:38:59.476000	notnownico	DRIFTING? PogU
636	0 days 03:38:59.480000	sewerborn	???????
637	0 days 03:38:59.560000	cperrito64	sawtell bussin
638	0 days 03:38:59.588000	toxicpoo69	THE STRIP CLUB
639	0 days 03:38:59.634000	MinusCriticalDamage	LULW
640	0 days 03:38:59.698000	hyphn8r	OMEGALUL THRIFTING?
641	0 days 03:38:59.729000	RioX	hell no
642	0 days 03:38:59.762000	AVBest	??????????????????????????
643	0 days 03:38:59.877000	EvenPB	NaM CHILLS
644	0 days 03:38:59.885000	siam__	WTF????????

Anexo 7

Configuración del servidor Apache

Figura 1

Configuración del servidor web Apache.



Anexo 8

Apache Jmeter, configuración y resultados

Figura 1

Interfaz inicial de Apache Jmeter para posteriormente configurar las pruebas de carga.

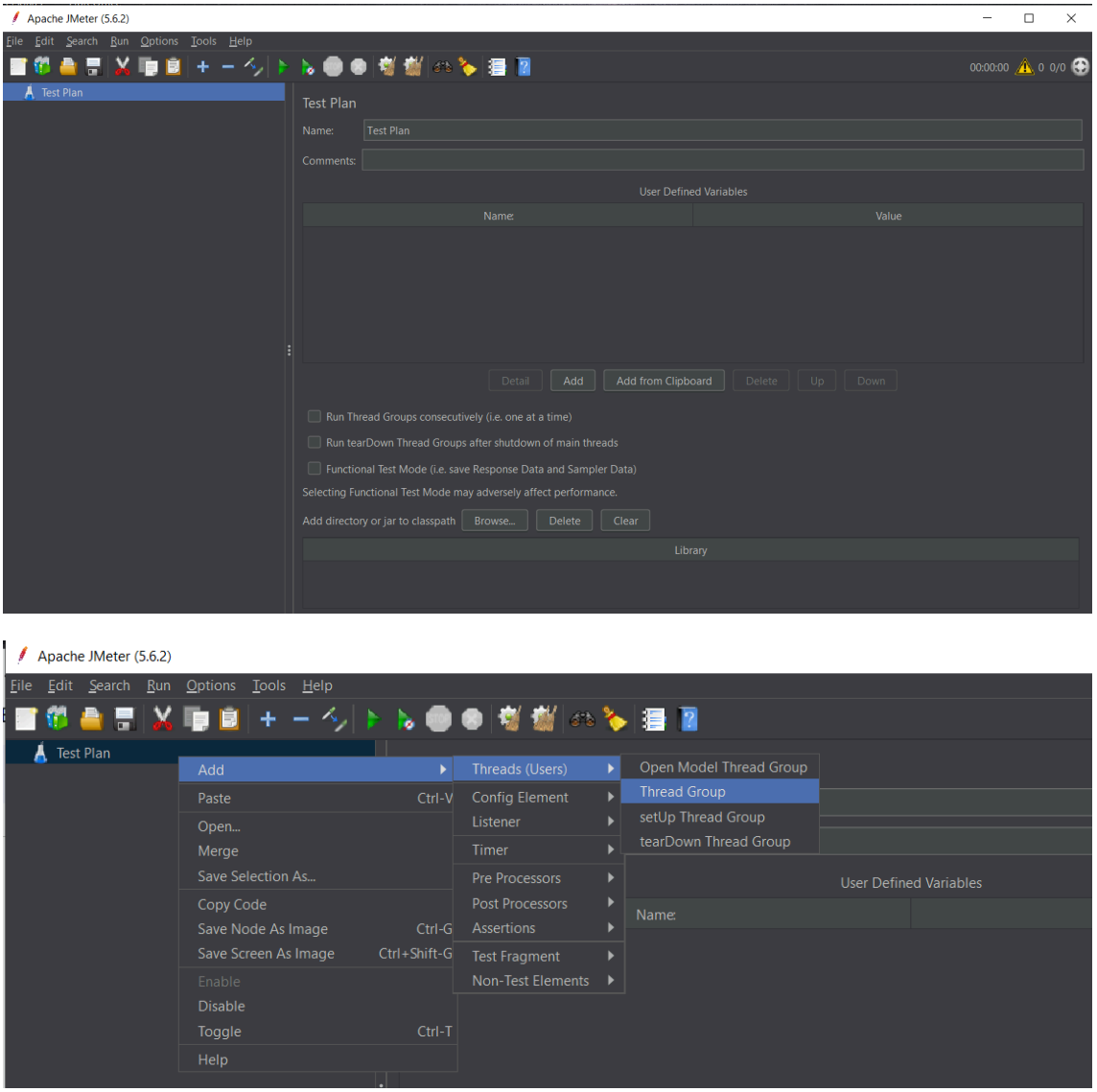


Figura 2. Creación de un Thread Group en Apache Jmeter para las pruebas de carga.

Figura 3

Configuración del Thread Group en Apache Jmeter para las pruebas de carga.

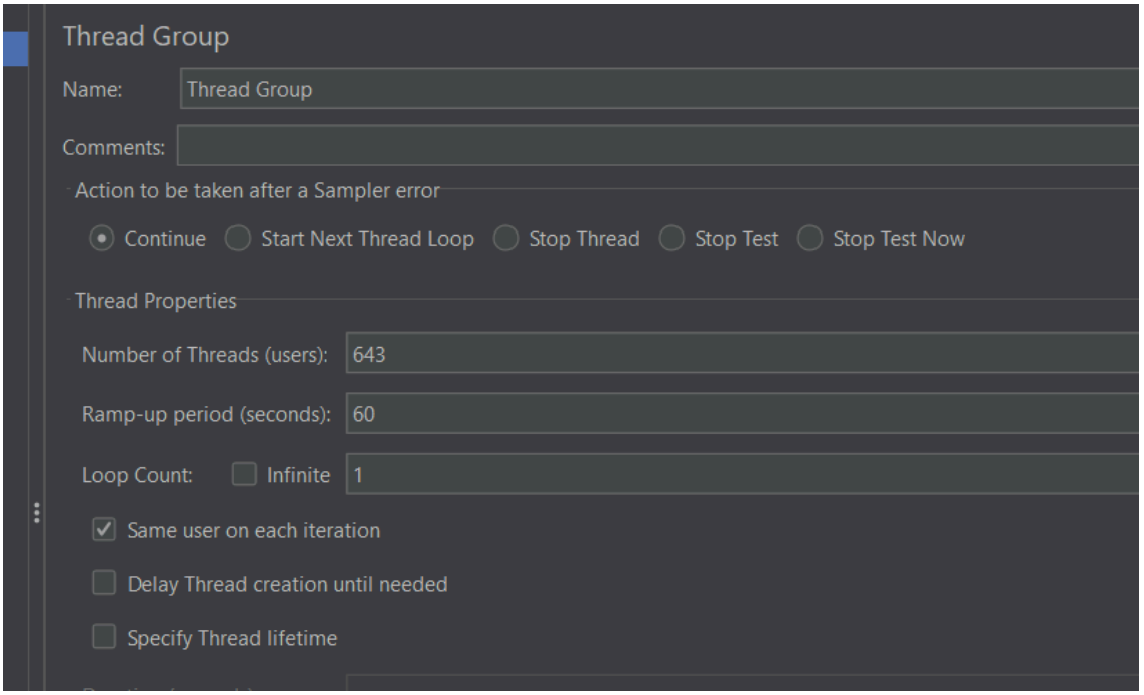


Figura 4

Creación del Sampler HTTP Request en Apache Jmeter para enviar peticiones http a los servidores web.

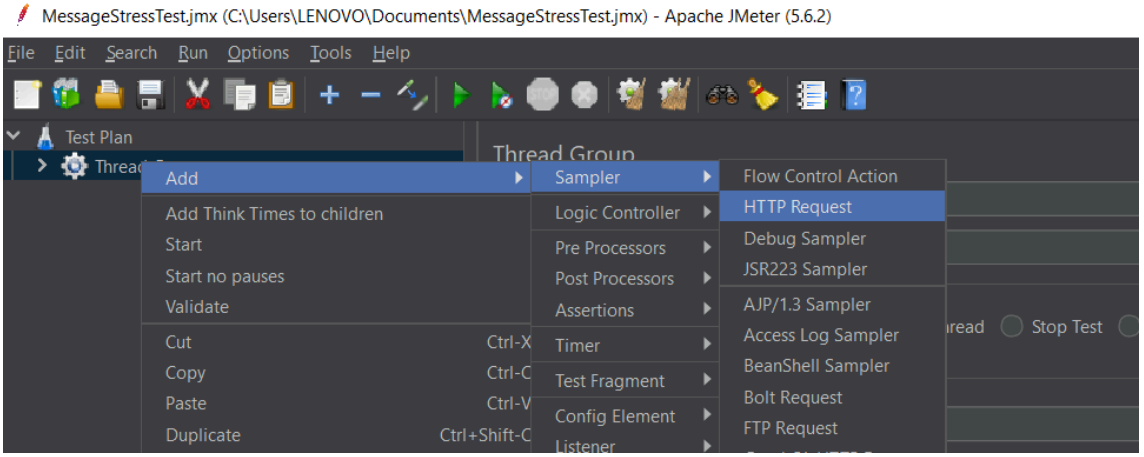


Figura 5

Configuración del Sampler HTTP Request en Apache Jmeter para enviar peticiones http al servidor Node.js.

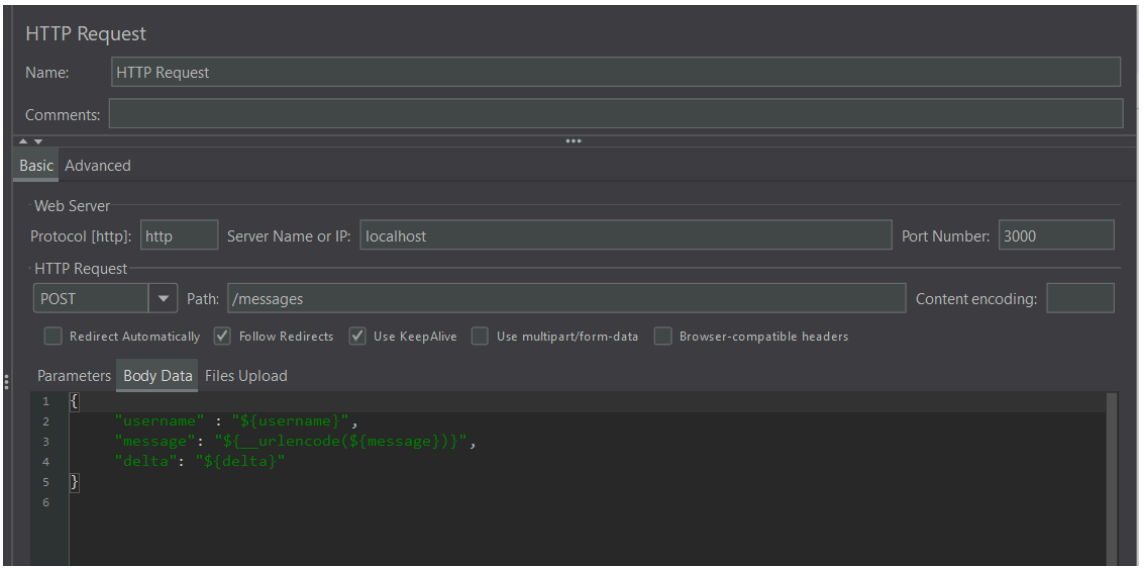


Figura 6

Configuración del Sampler HTTP Request en Apache Jmeter para enviar peticiones http al servidor Apache.

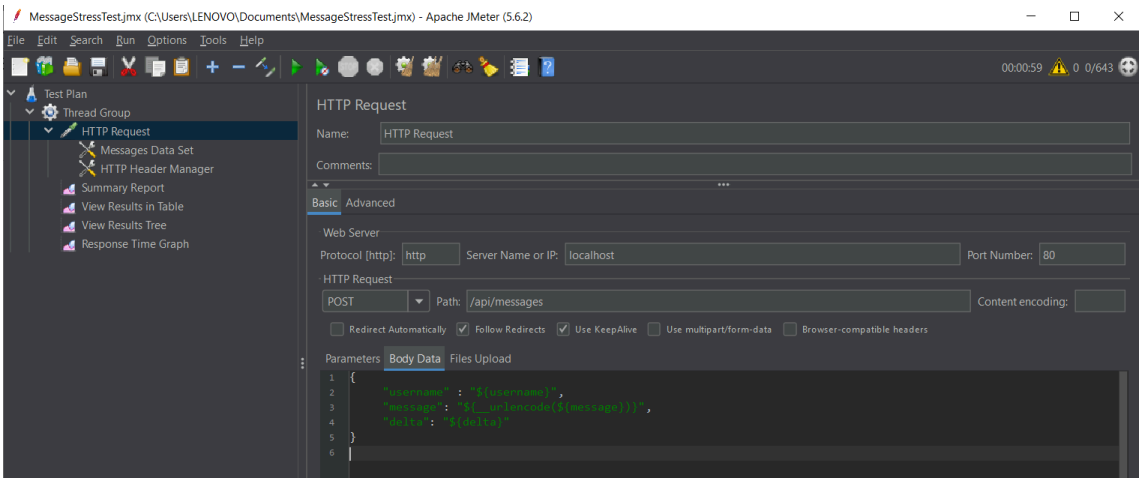


Figura 7

Implementación de Listeners para las pruebas de carga.

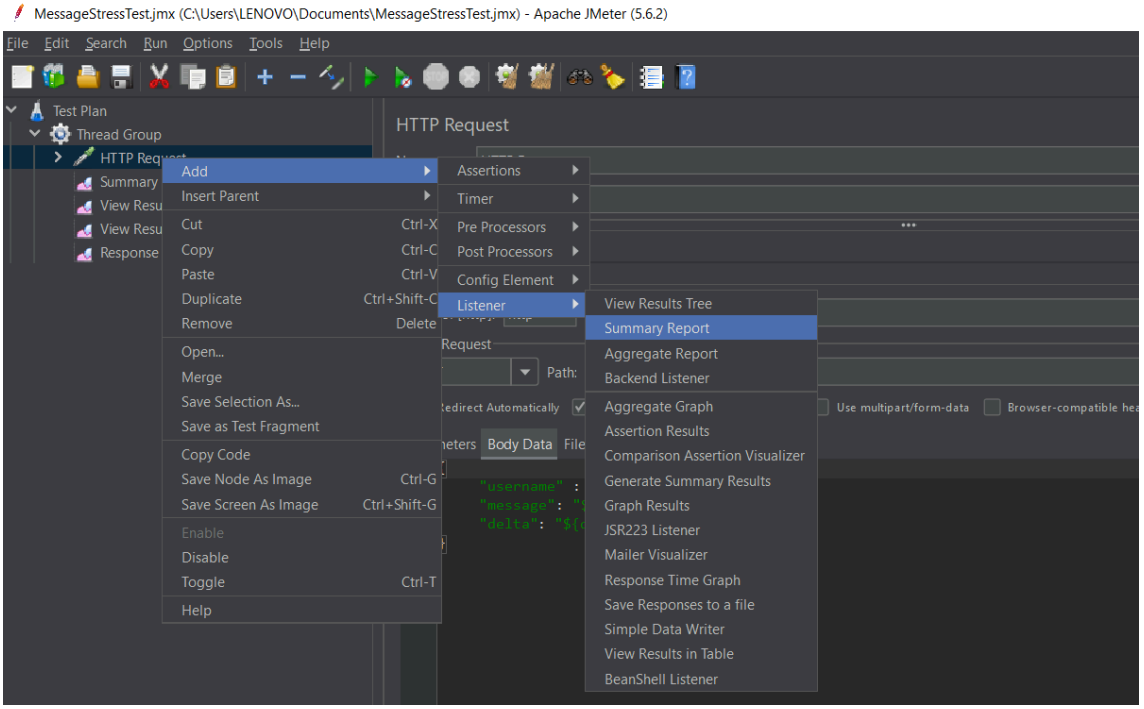


Figura 8

Lista de Listeners para las pruebas de carga, donde se ven todos los datos de cada petición http.

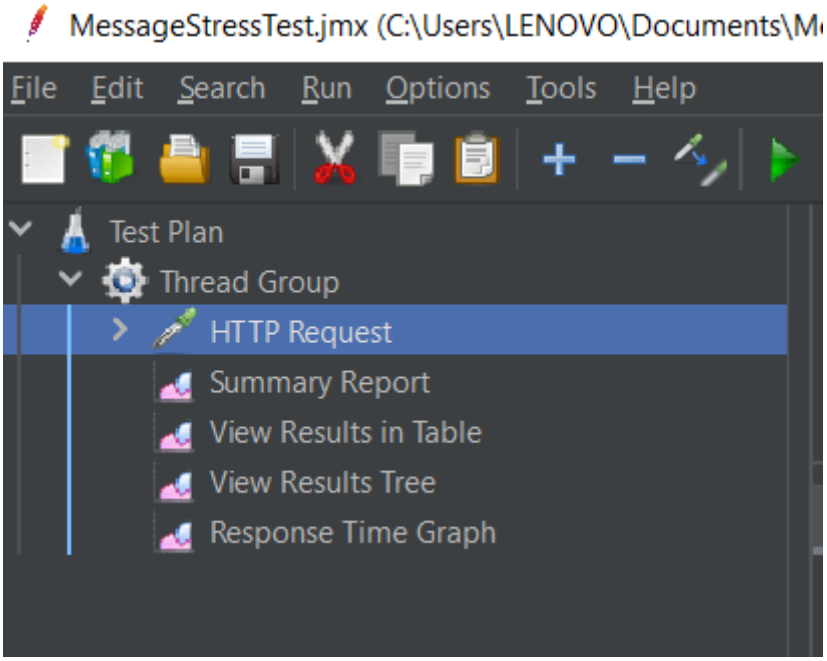


Figura 9

Creación de CSV Data Set Config y HTTP Header Manager para cargar datos desde un archivo CSV de Python y gestionar encabezados en solicitudes HTTP.

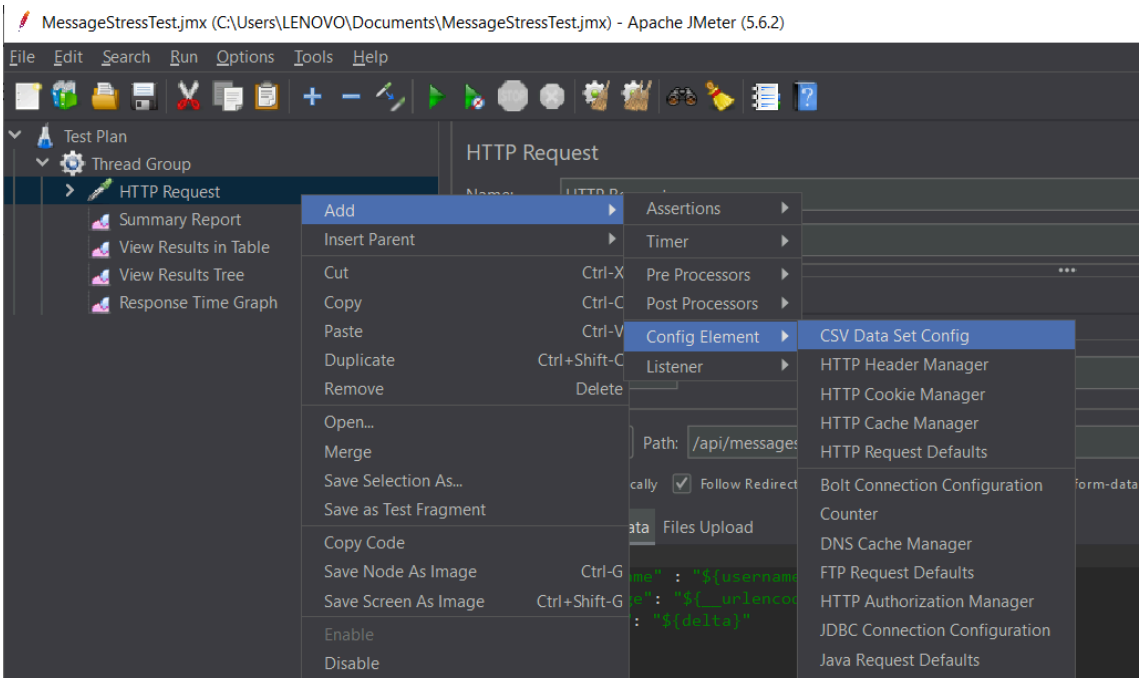


Figura 10

Configuración del elemento CSV Data Set Config para leer datos desde el csv exportado de python y crear peticiones http.

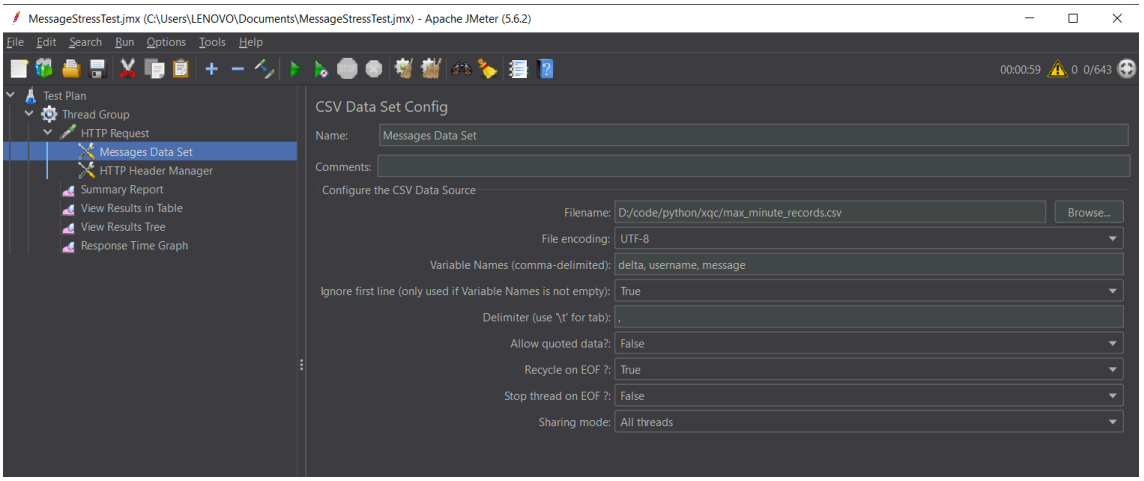


Figura 11

Creación del elemento CSV Data Set Config para leer datos desde el csv exportado de python y crear peticiones http.

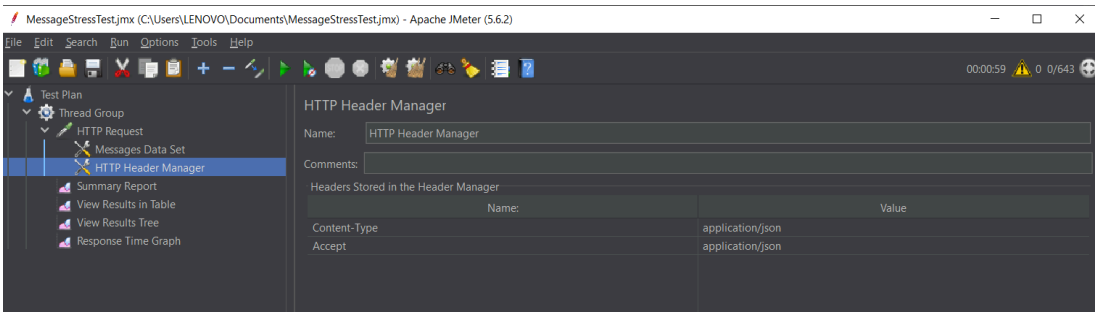


Figura 12

Resultados recuperados del listener Summary Report de la prueba de carga para el servidor Apache.

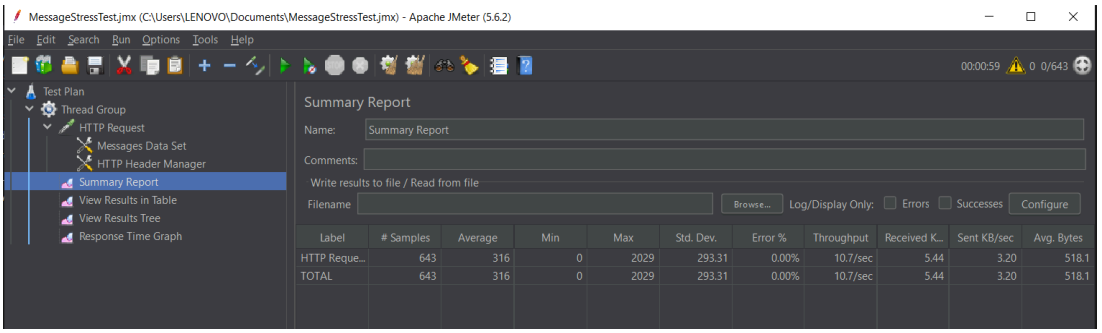


Figura 13

Resultados recuperados del listener View Results in Table de la prueba de carga para el servidor Apache.

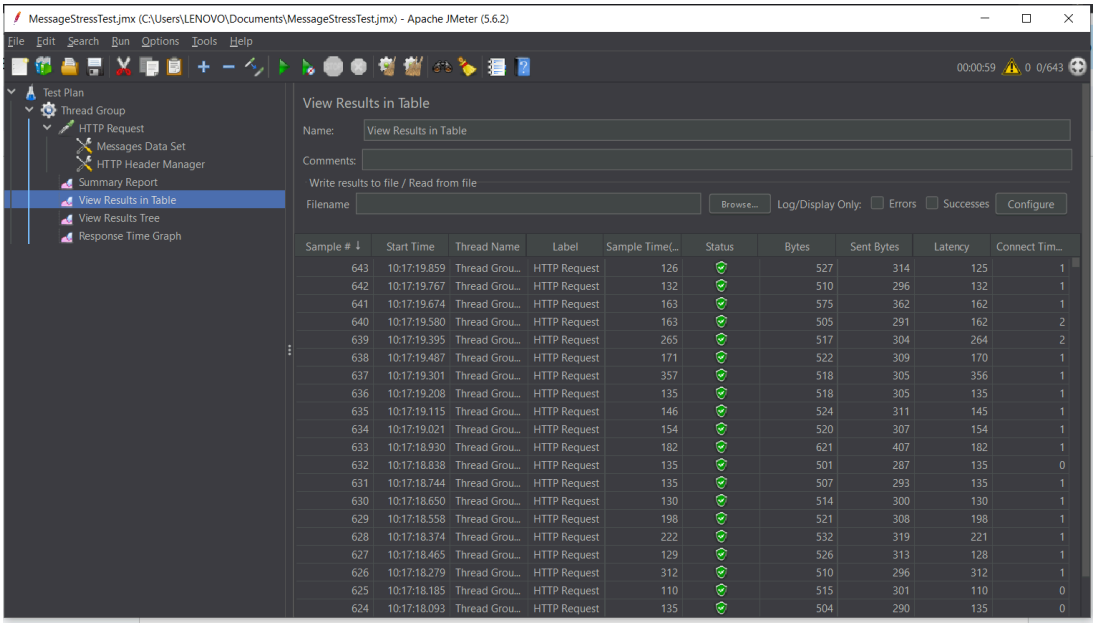


Figura 14

Resultados recuperados del listener View Results Tree de la prueba de carga para el servidor Apache.

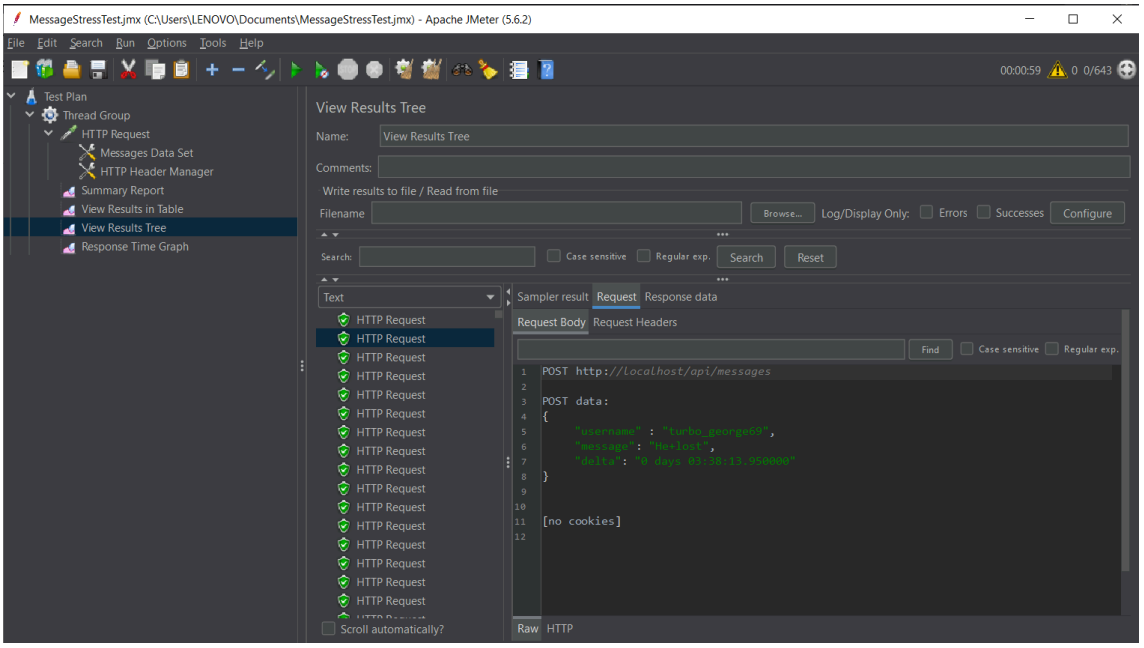


Figura 15

Resultados recuperados del listener Response Time Graph de la prueba de carga para el servidor Apache.

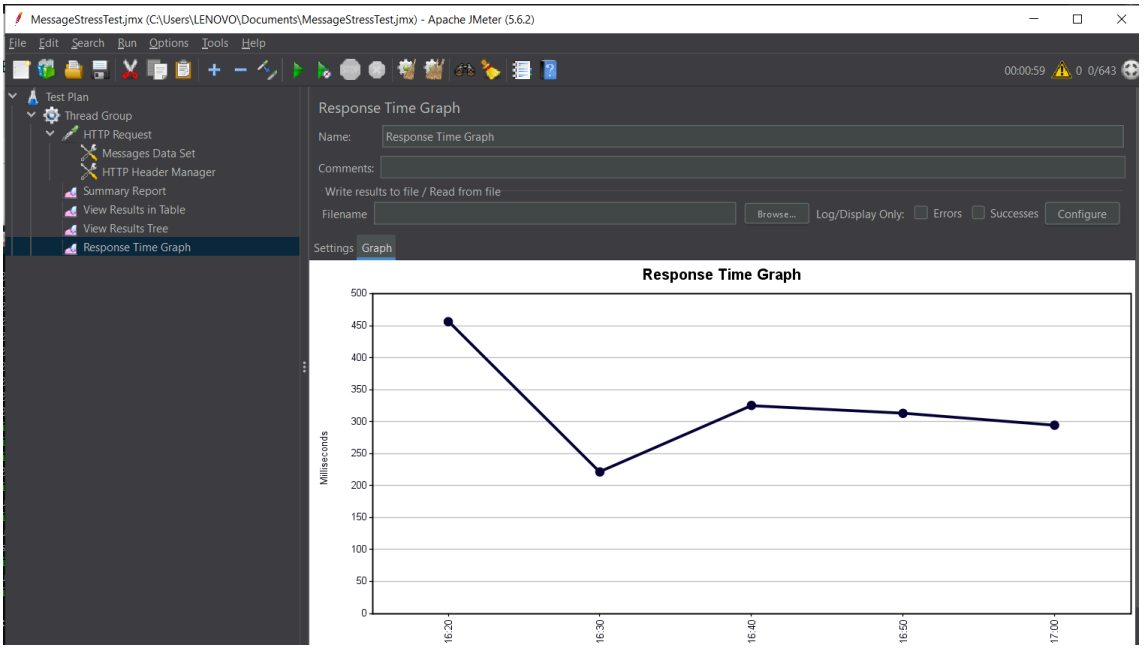


Figura 16

Resultados recuperados del listener Summary Report de la prueba de carga para el servidor Node.js.

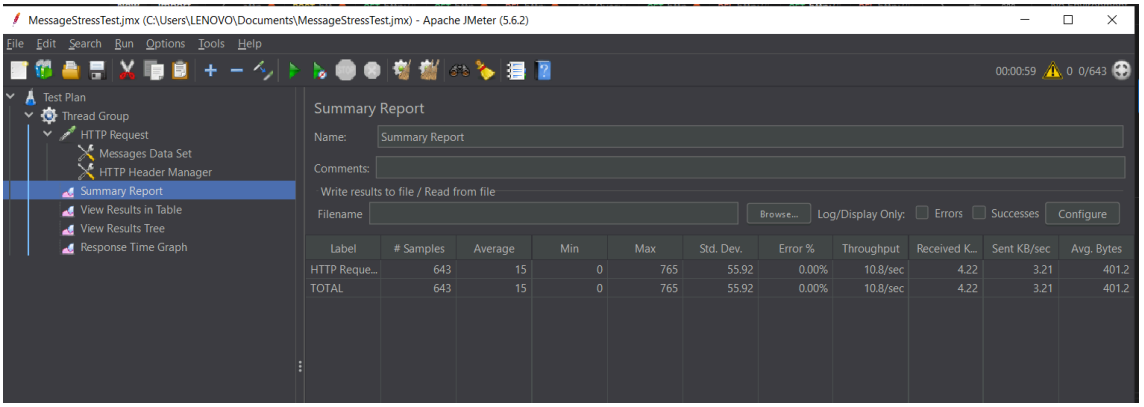


Figura 17

Resultados recuperados del listener View Results in Table de la prueba de carga para el servidor Node.js.

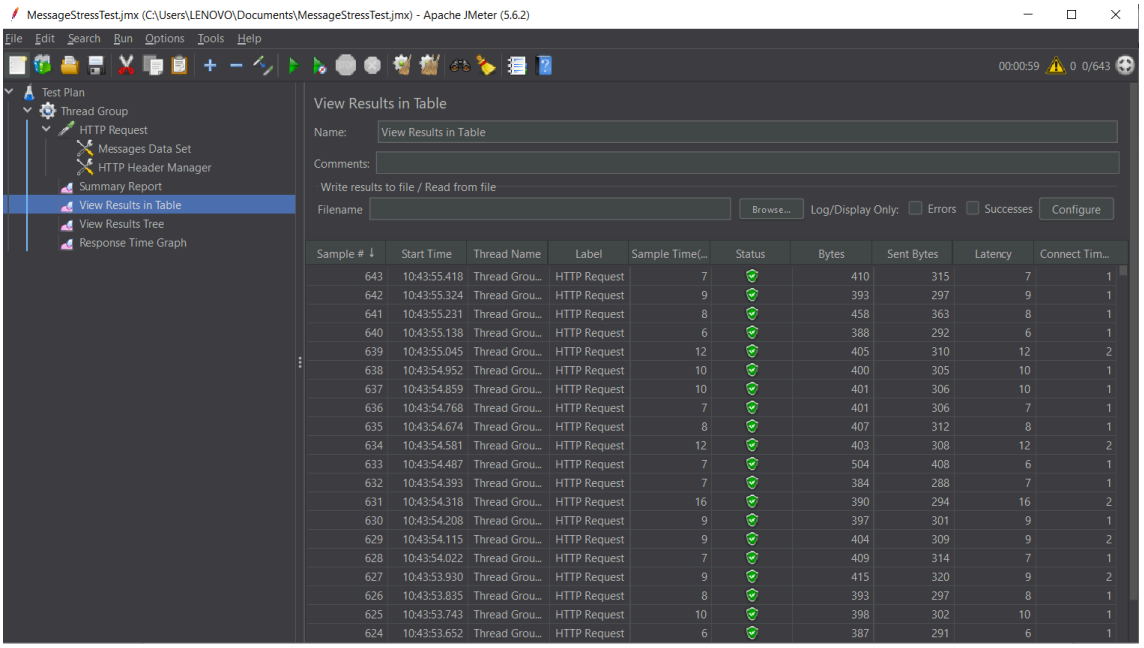


Figura 18

Resultados recuperados del listener View Results Tree de la prueba de carga para el servidor Node.js.

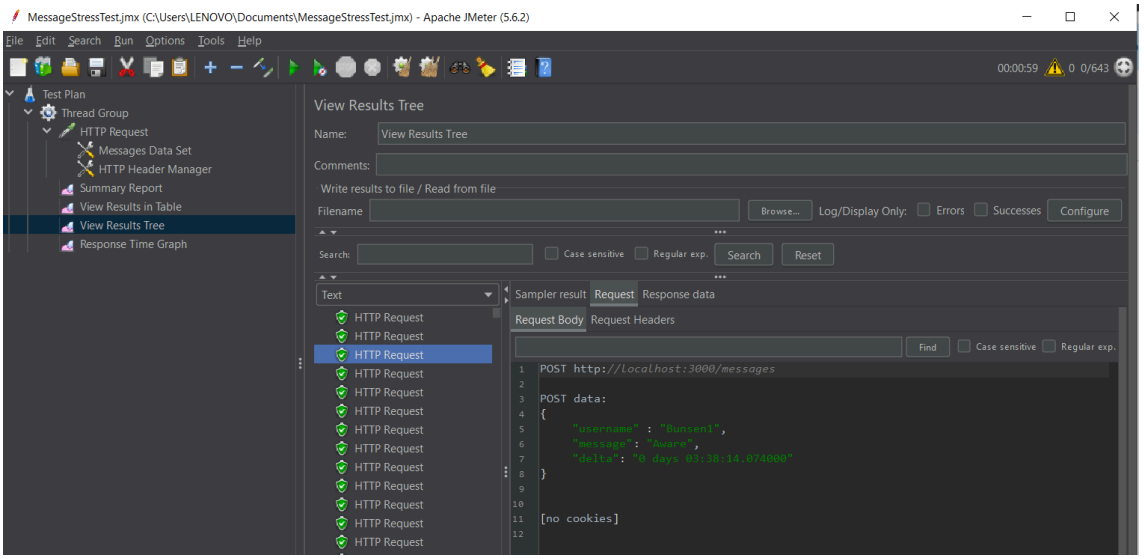
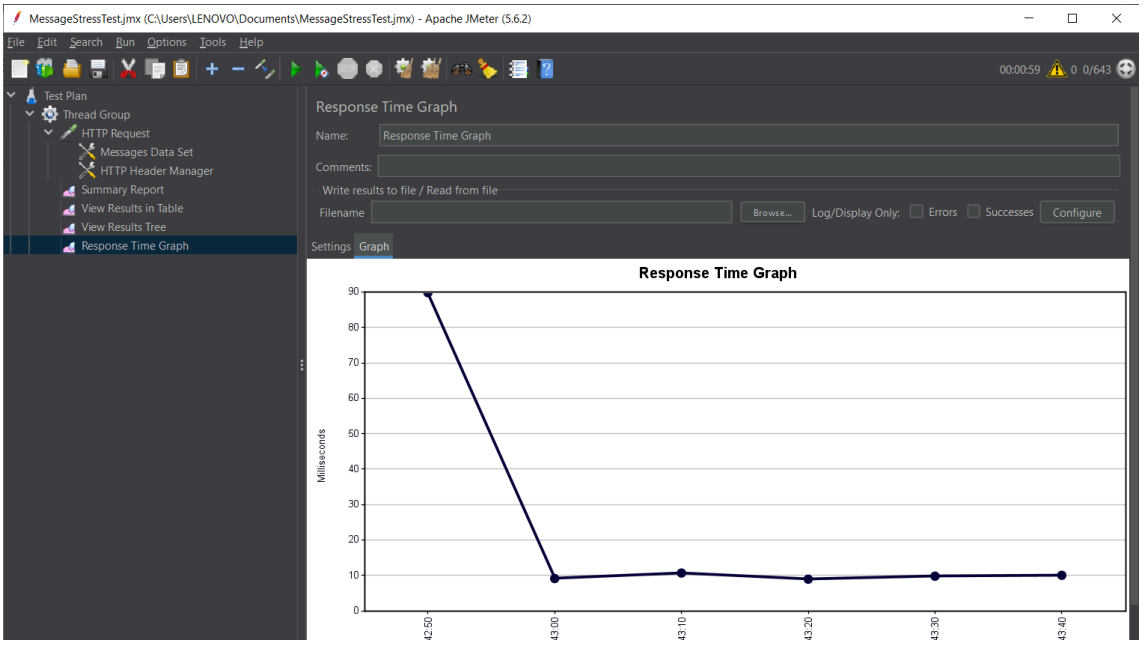


Figura 19

Resultados recuperados del listener Response Time Graph de la prueba de carga para el servidor Node.js.



Anexo 9

Comandos de linux para monitorear el uso de recursos de los servidores web y sus resultados

Figura 1

Comandos de linux para monitorear el uso de recursos del servidor Apache por cada prueba de carga y exportarlos en un csv.

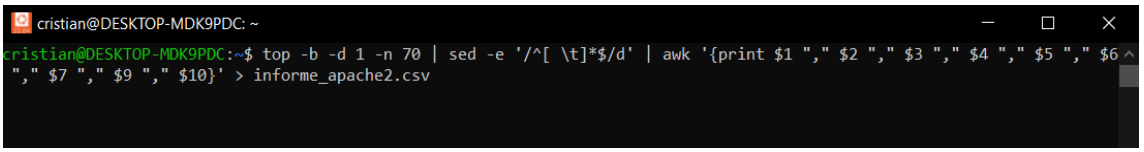
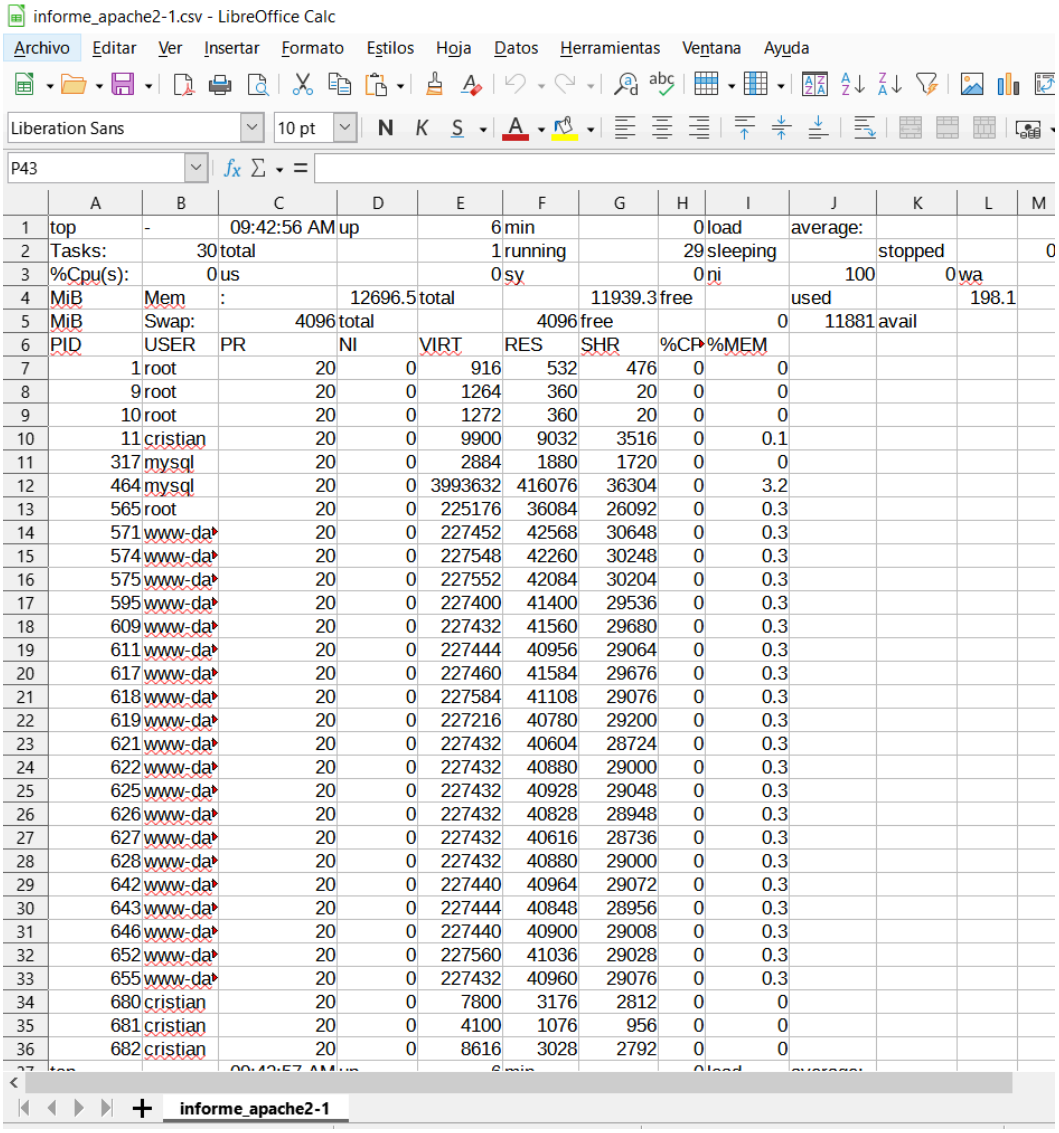


Figura 2

Csv exportado como resultado de comandos de linux para monitorear el uso de recursos del servidor Apache por cada prueba de carga.



	A	B	C	D	E	F	G	H	I	J	K	L	M
1	top	-	09:42:56 AM	up	6	min		0	load	average:			
2	Tasks:	30	total		1	running		29	sleeping		stopped		0
3	%Cpu(s):	0	us		0	sy		0	ni	100	0	wa	
4	MiB	Mem	:	12696.5	total		11939.3	free		used		198.1	
5	MiB	Swap:	4096	total		4096	free		0	11881	avail		
6	PID	USER	PR	NI	VIRT	RES	SHR	%CP	%MEM				
7	1	root	20	0	916	532	476	0	0				
8	9	root	20	0	1264	360	20	0	0				
9	10	root	20	0	1272	360	20	0	0				
10	11	cristian	20	0	9900	9032	3516	0	0.1				
11	317	mysql	20	0	2884	1880	1720	0	0				
12	464	mysql	20	0	3993632	416076	36304	0	3.2				
13	565	root	20	0	225176	36084	26092	0	0.3				
14	571	www-da	20	0	227452	42568	30648	0	0.3				
15	574	www-da	20	0	227548	42260	30248	0	0.3				
16	575	www-da	20	0	227552	42084	30204	0	0.3				
17	595	www-da	20	0	227400	41400	29536	0	0.3				
18	609	www-da	20	0	227432	41560	29680	0	0.3				
19	611	www-da	20	0	227444	40956	29064	0	0.3				
20	617	www-da	20	0	227460	41584	29676	0	0.3				
21	618	www-da	20	0	227584	41108	29076	0	0.3				
22	619	www-da	20	0	227216	40780	29200	0	0.3				
23	621	www-da	20	0	227432	40604	28724	0	0.3				
24	622	www-da	20	0	227432	40880	29000	0	0.3				
25	625	www-da	20	0	227432	40928	29048	0	0.3				
26	626	www-da	20	0	227432	40828	28948	0	0.3				
27	627	www-da	20	0	227432	40616	28736	0	0.3				
28	628	www-da	20	0	227432	40880	29000	0	0.3				
29	642	www-da	20	0	227440	40964	29072	0	0.3				
30	643	www-da	20	0	227444	40848	28956	0	0.3				
31	646	www-da	20	0	227440	40900	29008	0	0.3				
32	652	www-da	20	0	227560	41036	29028	0	0.3				
33	655	www-da	20	0	227432	40960	29076	0	0.3				
34	680	cristian	20	0	7800	3176	2812	0	0				
35	681	cristian	20	0	4100	1076	956	0	0				
36	682	cristian	20	0	8616	3028	2792	0	0				

Figura 3

Comandos de linux para monitorear el uso de recursos del servidor Node.js por cada prueba de carga y exportarlos en un csv.

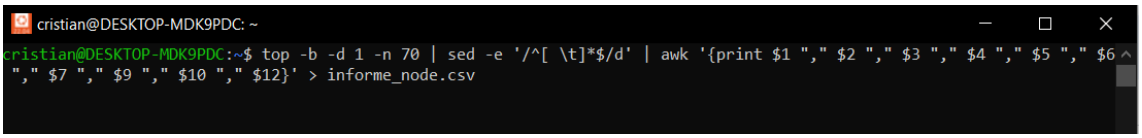
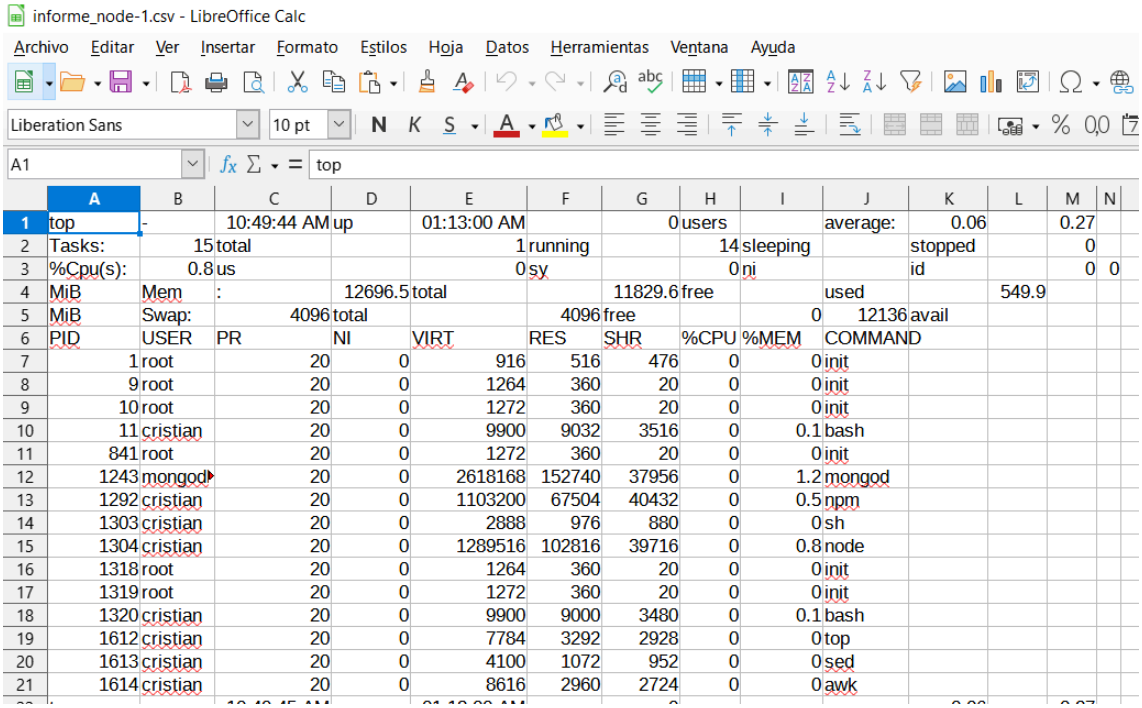


Figura 4

Csv exportado como resultado de comandos de linux para monitorear el uso de recursos del servidor Node.js por cada prueba de carga.



informe_node-1.csv - LibreOffice Calc

Archivo Editar Ver Insertar Formato Estilos Hoja Datos Herramientas Ventana Ayuda

Liberation Sans 10 pt

A1 top

	A	B	C	D	E	F	G	H	I	J	K	L	M	N
1	top	-	10:49:44 AM	up	01:13:00 AM		0	users		average:	0.06		0.27	
2	Tasks:	15	total		1	running		14	sleeping		stopped		0	
3	%Cpu(s):	0.8	us		0	sy		0	ni		id		0	0
4	MiB	Mem	:	12696.5	total		11829.6	free		used		549.9		
5	MiB	Swap:	4096	total		4096	free		0	12136	avail			
6	PID	USER	PR	NI	VIRT	RES	SHR	%CPU	%MEM	COMMAND				
7	1	root	20	0	916	516	476	0	0	init				
8	9	root	20	0	1264	360	20	0	0	init				
9	10	root	20	0	1272	360	20	0	0	init				
10	11	cristian	20	0	9900	9032	3516	0	0.1	bash				
11	841	root	20	0	1272	360	20	0	0	init				
12	1243	mongod	20	0	2618168	152740	37956	0	1.2	mongod				
13	1292	cristian	20	0	1103200	67504	40432	0	0.5	npm				
14	1303	cristian	20	0	2888	976	880	0	0	sh				
15	1304	cristian	20	0	1289516	102816	39716	0	0.8	node				
16	1318	root	20	0	1264	360	20	0	0	init				
17	1319	root	20	0	1272	360	20	0	0	init				
18	1320	cristian	20	0	9900	9000	3480	0	0.1	bash				
19	1612	cristian	20	0	7784	3292	2928	0	0	top				
20	1613	cristian	20	0	4100	1072	952	0	0	sed				
21	1614	cristian	20	0	8616	2960	2724	0	0	awk				

Anexo 10

Código duende de la aplicación alojada en el servidor web Apache

Figura 1

Código de configuración de las variables de entorno y declaración de las credenciales de la base de datos mysql que está asociada a la aplicación alojada en Apache.

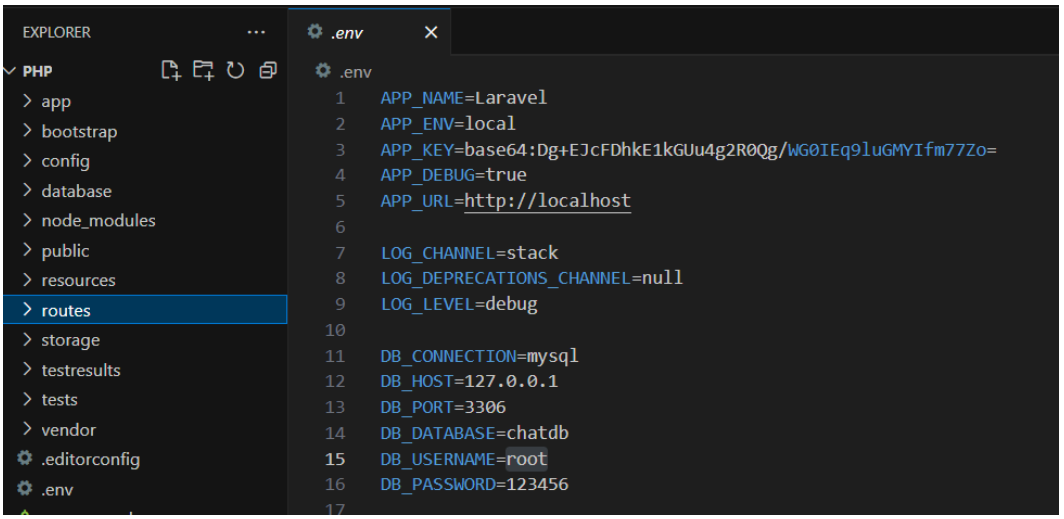


Figura 2

Código para crear la tabla "messages" en la base de datos mysql que está asociada a la aplicación alojada en Apache.

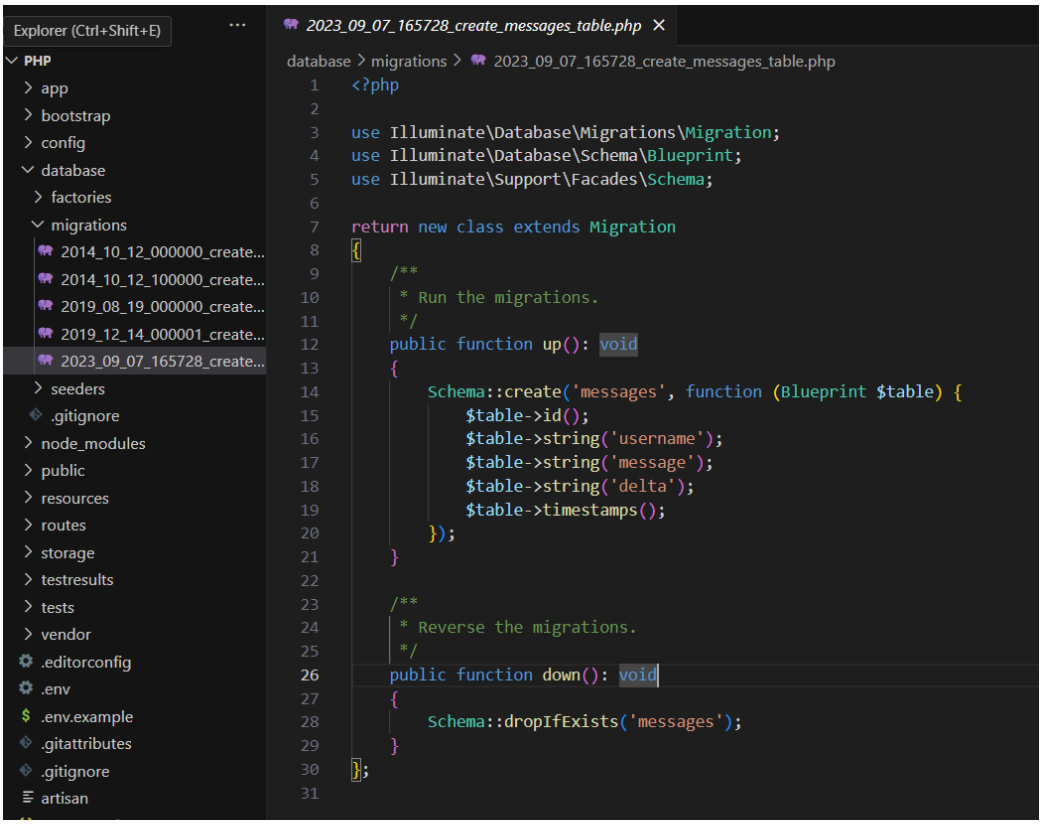


Figura 3

Código para crear el modelo "messages" de la aplicación alojada en Apache.

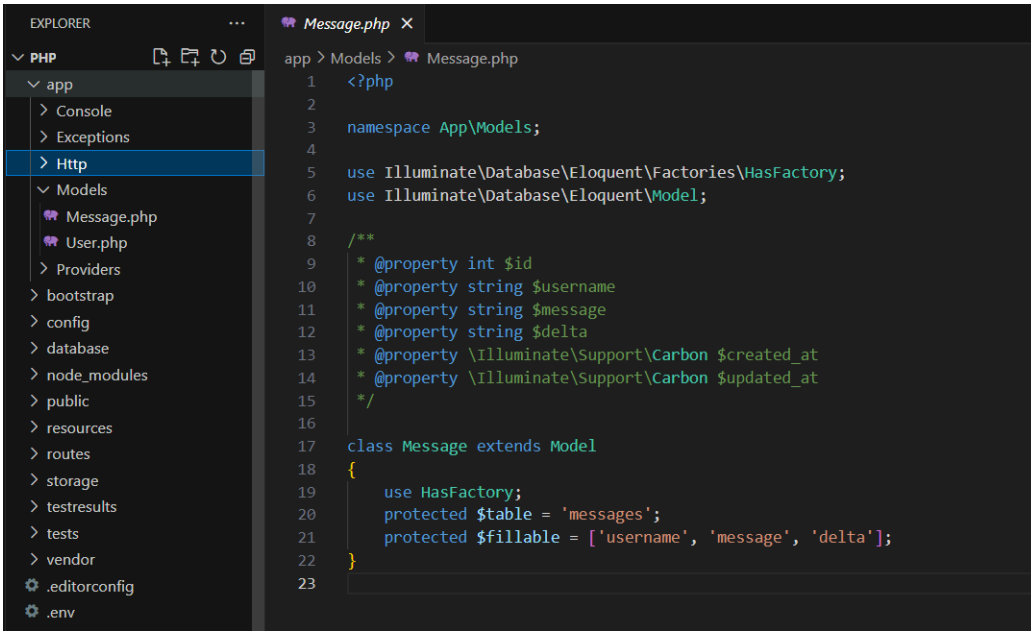


Figura 4

Código del controlador "MessageController" de la aplicación alojada en Apache.

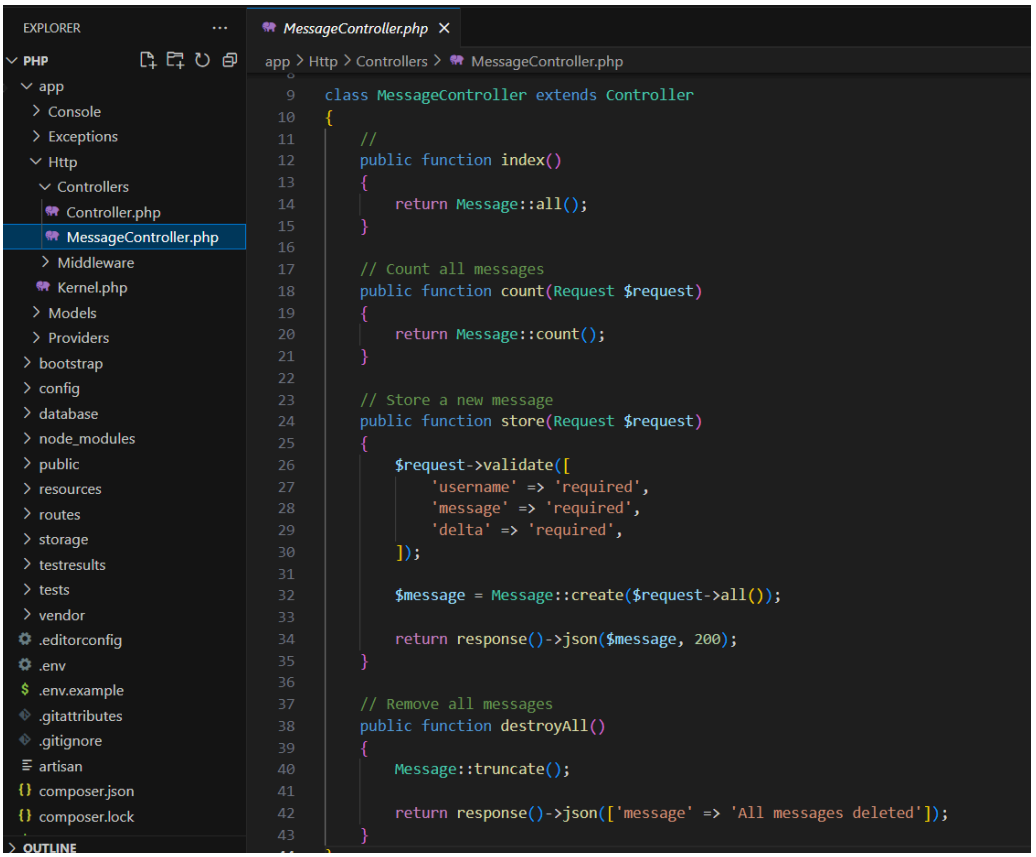
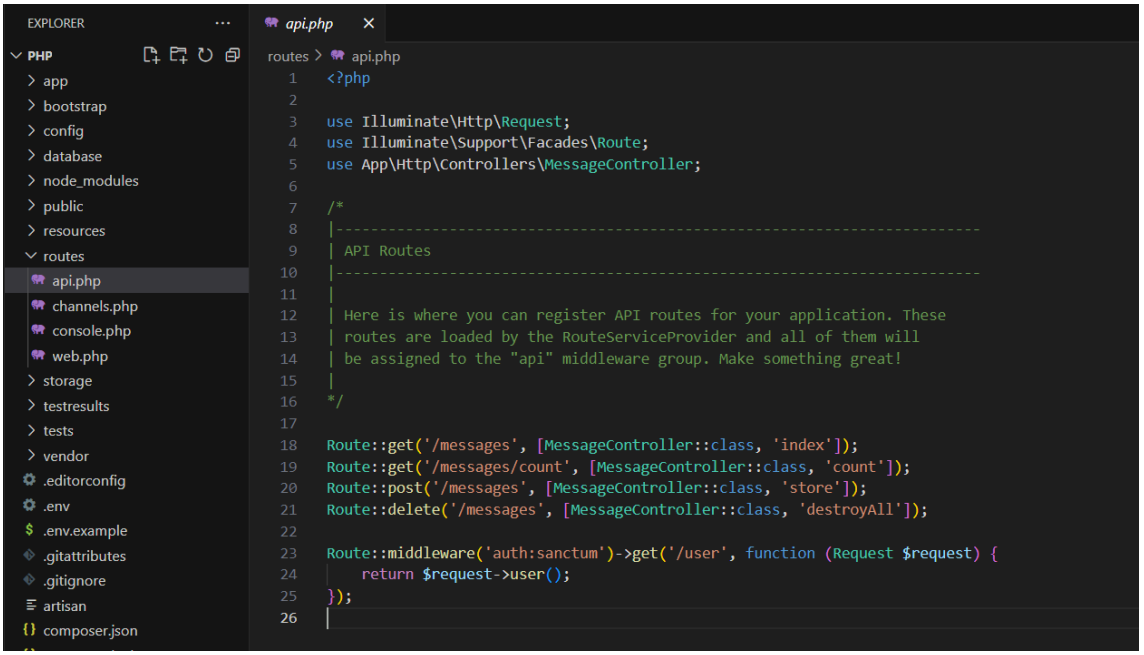


Figura 5

Código para crear los endpoints en las rutas de la API de la aplicación alojada en Apache.



Anexo 11

Código fuente de la aplicación alojada en el servidor web Node.js

Figura 1

Código de configuración de la aplicación en Node.js y conexión a la base de datos.

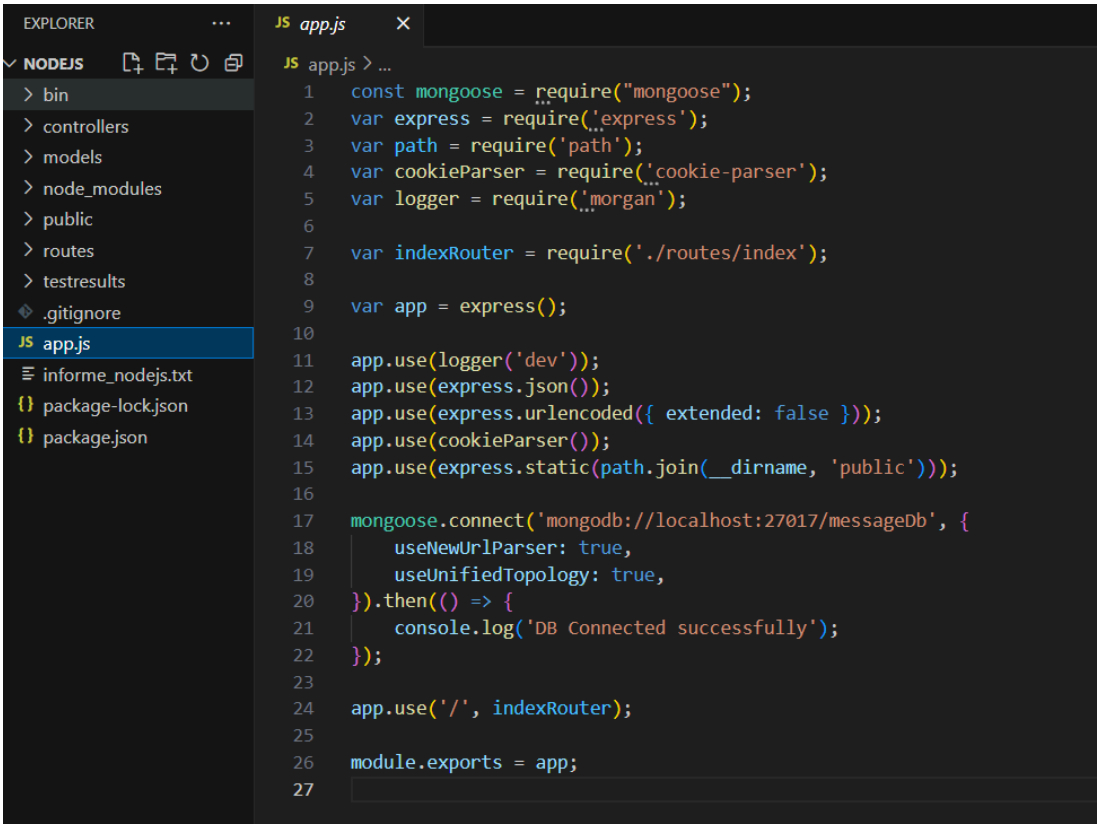


Figura 2

Código para la creación del modelo "Message" de la aplicación alojada en Node.js.

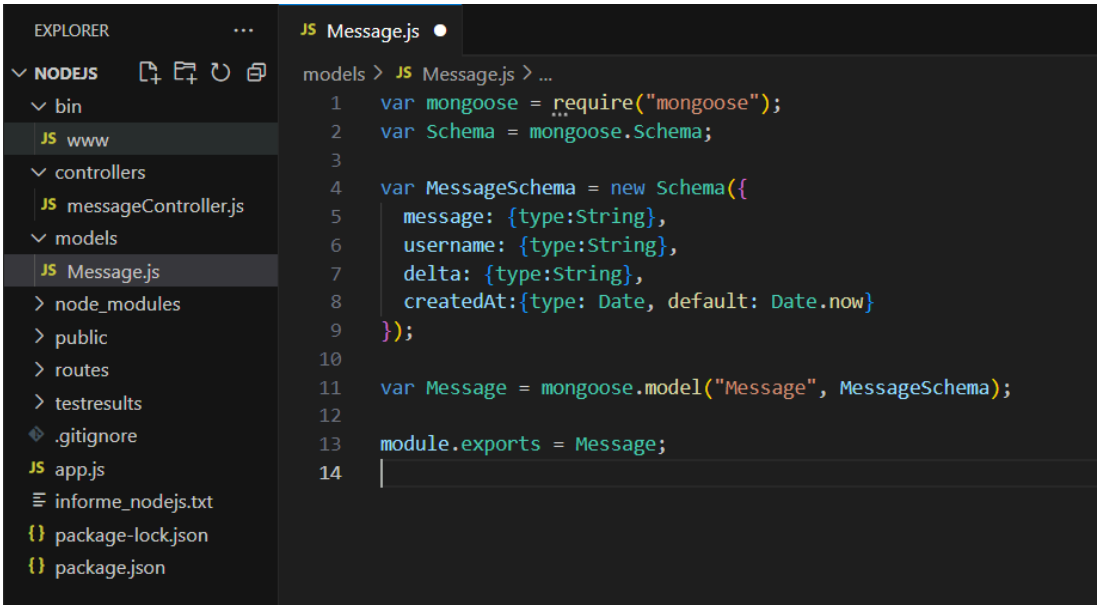


Figura 3

Código para la creación del controlador "messageController" de la aplicación alojada en Node.js.

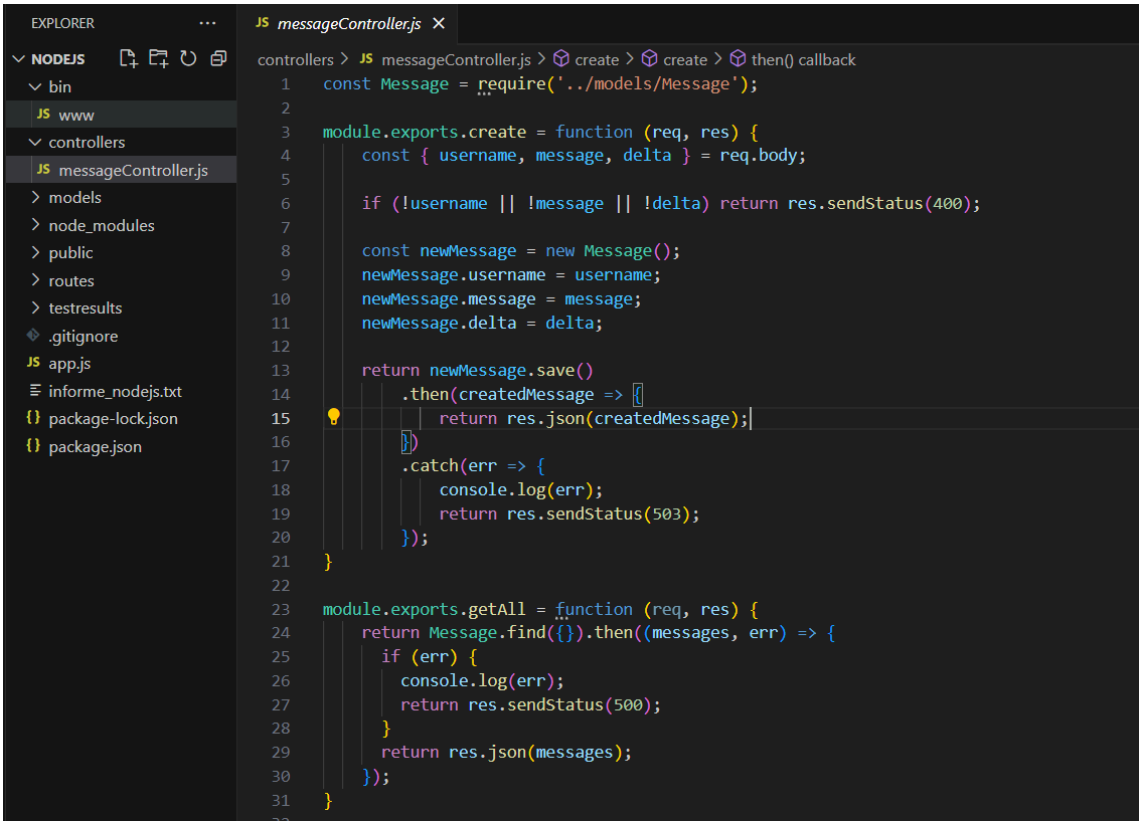
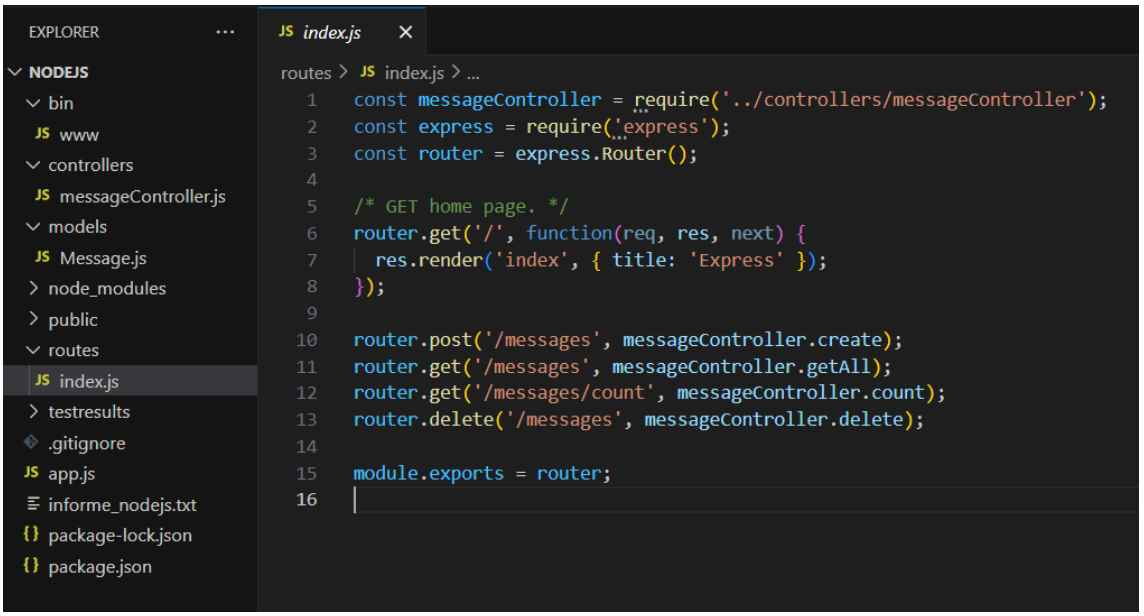


Figura 4

Código para crear los endpoints en las rutas de la API de la aplicación alojada en Node.js.



Anexo 12

Registros de los mensajes en las bases de datos

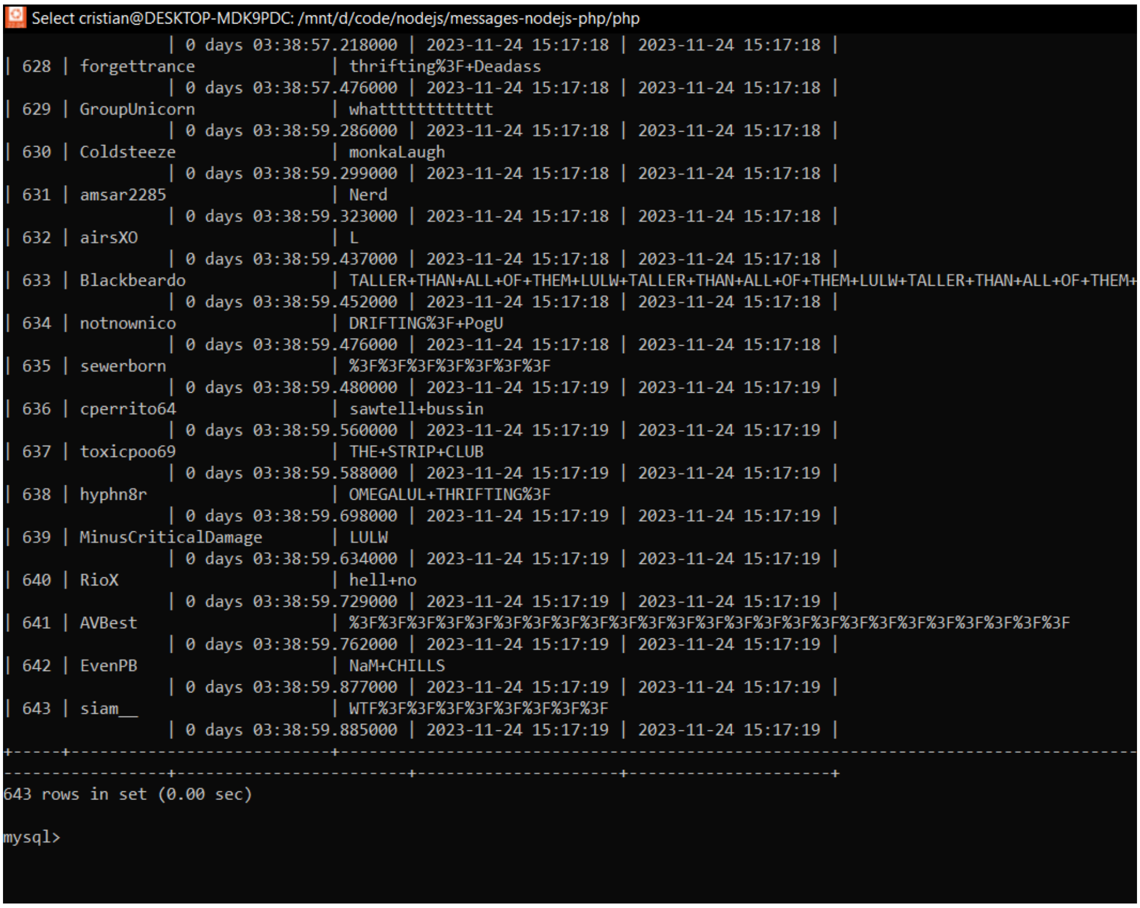
Figura 1

Primeros registros de la base de datos mysql después de una prueba de carga realizada contra la aplicación alojada en el servidor Apache.

```
mysql> SELECT * FROM messages;
+-----+-------------------------+------------------------------------------------------------------------------------------------------------
----------------+-----------------------+---------------------+---------------------+
| id  | username                | message
                | delta                 | created_at          | updated_at          |
+-----+-------------------------+------------------------------------------------------------------------------------------------------------
----------------+-----------------------+---------------------+---------------------+
|   1 | Rvyne9                  | Nerd
                | 0 days 03:38:00.182000 | 2023-11-24 15:16:21 | 2023-11-24 15:16:21 |
|   2 | Calum_4                 | Nerd
                | 0 days 03:38:00.162000 | 2023-11-24 15:16:21 | 2023-11-24 15:16:21 |
|   3 | cnnoisseur              | sit+down+ben
                | 0 days 03:38:00.164000 | 2023-11-24 15:16:21 | 2023-11-24 15:16:21 |
|   4 | joncurious              | LUL
                | 0 days 03:38:00.259000 | 2023-11-24 15:16:21 | 2023-11-24 15:16:21 |
|   5 | xdFrappe                | Nerd+BEN
                | 0 days 03:38:00.158000 | 2023-11-24 15:16:21 | 2023-11-24 15:16:21 |
|   6 | salvationdota           | forsenPls
                | 0 days 03:38:00.290000 | 2023-11-24 15:16:21 | 2023-11-24 15:16:21 |
|   7 | Ricolicious_            | Kapp
                | 0 days 03:38:00.055000 | 2023-11-24 15:16:21 | 2023-11-24 15:16:21 |
|   8 | thenamestendies         | Nerd
                | 0 days 03:38:00.144000 | 2023-11-24 15:16:21 | 2023-11-24 15:16:21 |
|   9 | feltm0th                | DAD+GIGACHAD
                | 0 days 03:38:00.259000 | 2023-11-24 15:16:21 | 2023-11-24 15:16:21 |
|  10 | wholeydonut             | %3F%3F%3F%3F%3F%3F%3F%3F%3F%3F%3F%3F%3F%3F%3F%3F%3F%3F%3F%3F%3F%3F%3F%3F%3F%3F%3F%3F
                | 0 days 03:38:00.320000 | 2023-11-24 15:16:22 | 2023-11-24 15:16:22 |
|  11 | thraxexus               | real+gigas+go+to+movies+say+nothing+and+still+go+home+with+girl+GIGACHAD
                | 0 days 03:38:02.388000 | 2023-11-24 15:16:22 | 2023-11-24 15:16:22 |
|  12 | CherrySIushy            | Nerd
                | 0 days 03:38:00.390000 | 2023-11-24 15:16:22 | 2023-11-24 15:16:22 |
|  13 | cl0uty01                | Nerd
                | 0 days 03:38:00.585000 | 2023-11-24 15:16:22 | 2023-11-24 15:16:22 |
|  14 | y21de                   | Nerd
                | 0 days 03:38:00.434000 | 2023-11-24 15:16:22 | 2023-11-24 15:16:22 |
|  15 | Supoopi                 | %F0%9F%90%B6+baen
                | 0 days 03:38:00.728000 | 2023-11-24 15:16:22 | 2023-11-24 15:16:22 |
|  16 | CookieMilkyy            | jesus
                | 0 days 03:38:00.731000 | 2023-11-24 15:16:22 | 2023-11-24 15:16:22 |
|  17 | liilgoonin              | Nerd
                | 0 days 03:38:02.321000 | 2023-11-24 15:16:22 | 2023-11-24 15:16:22 |
|  18 | deepinthebellend        | long+distance%3F%3F
                | 0 days 03:38:02.355000 | 2023-11-24 15:16:22 | 2023-11-24 15:16:22 |
```

Figura 2

Últimos registros de la base de datos mysql después de una prueba de carga realizada contra la aplicación alojada en el servidor Apache.



```
Select cristian@DESKTOP-MDK9PDC: /mnt/d/code/nodejs/messages-nodejs-php/php
                 | 0 days 03:38:57.218000 | 2023-11-24 15:17:18 | 2023-11-24 15:17:18 |
| 628 | forgettrance                | thrifting%3F+Deadass
                 | 0 days 03:38:57.476000 | 2023-11-24 15:17:18 | 2023-11-24 15:17:18 |
| 629 | GroupUnicorn                | whattttttttttttt
                 | 0 days 03:38:59.286000 | 2023-11-24 15:17:18 | 2023-11-24 15:17:18 |
| 630 | Coldsteeze                  | monkaLaugh
                 | 0 days 03:38:59.299000 | 2023-11-24 15:17:18 | 2023-11-24 15:17:18 |
| 631 | amsar2285                   | Nerd
                 | 0 days 03:38:59.323000 | 2023-11-24 15:17:18 | 2023-11-24 15:17:18 |
| 632 | airsXO                      | L
                 | 0 days 03:38:59.437000 | 2023-11-24 15:17:18 | 2023-11-24 15:17:18 |
| 633 | Blackbeardo                 | TALLER+THAN+ALL+OF+THEM+LULW+TALLER+THAN+ALL+OF+THEM+LULW+TALLER+THAN+ALL+OF+THEM+
                 | 0 days 03:38:59.452000 | 2023-11-24 15:17:18 | 2023-11-24 15:17:18 |
| 634 | notnownico                  | DRIFTING%3F+PogU
                 | 0 days 03:38:59.476000 | 2023-11-24 15:17:18 | 2023-11-24 15:17:18 |
| 635 | sewerborn                   | %3F%3F%3F%3F%3F%3F%3F
                 | 0 days 03:38:59.480000 | 2023-11-24 15:17:19 | 2023-11-24 15:17:19 |
| 636 | cperrito64                  | sawtell+bussin
                 | 0 days 03:38:59.560000 | 2023-11-24 15:17:19 | 2023-11-24 15:17:19 |
| 637 | toxicpoo69                  | THE+STRIP+CLUB
                 | 0 days 03:38:59.588000 | 2023-11-24 15:17:19 | 2023-11-24 15:17:19 |
| 638 | hyphn8r                     | OMEGALUL+THRIFTING%3F
                 | 0 days 03:38:59.698000 | 2023-11-24 15:17:19 | 2023-11-24 15:17:19 |
| 639 | MinusCriticalDamage         | LULW
                 | 0 days 03:38:59.634000 | 2023-11-24 15:17:19 | 2023-11-24 15:17:19 |
| 640 | RioX                        | hell+no
                 | 0 days 03:38:59.729000 | 2023-11-24 15:17:19 | 2023-11-24 15:17:19 |
| 641 | AVBest                      | %3F%3F%3F%3F%3F%3F%3F%3F%3F%3F%3F%3F%3F%3F%3F%3F%3F%3F%3F%3F%3F%3F%3F%3F%3F%3F%3F%3F
                 | 0 days 03:38:59.762000 | 2023-11-24 15:17:19 | 2023-11-24 15:17:19 |
| 642 | EvenPB                      | NaM+CHILLS
                 | 0 days 03:38:59.877000 | 2023-11-24 15:17:19 | 2023-11-24 15:17:19 |
| 643 | siam__                      | WTF%3F%3F%3F%3F%3F%3F%3F%3F%3F
                 | 0 days 03:38:59.885000 | 2023-11-24 15:17:19 | 2023-11-24 15:17:19 |
+-----+-----------------------------+----------------------------------------------------------------------------------
-----------------+------------------------+---------------------+---------------------+
643 rows in set (0.00 sec)

mysql>
```

Figura 3

Primeros registros de la base de datos mongodb después de una prueba de carga realizada contra la aplicación alojada en el servidor Node.js, realizada a través del terminal mongosh.

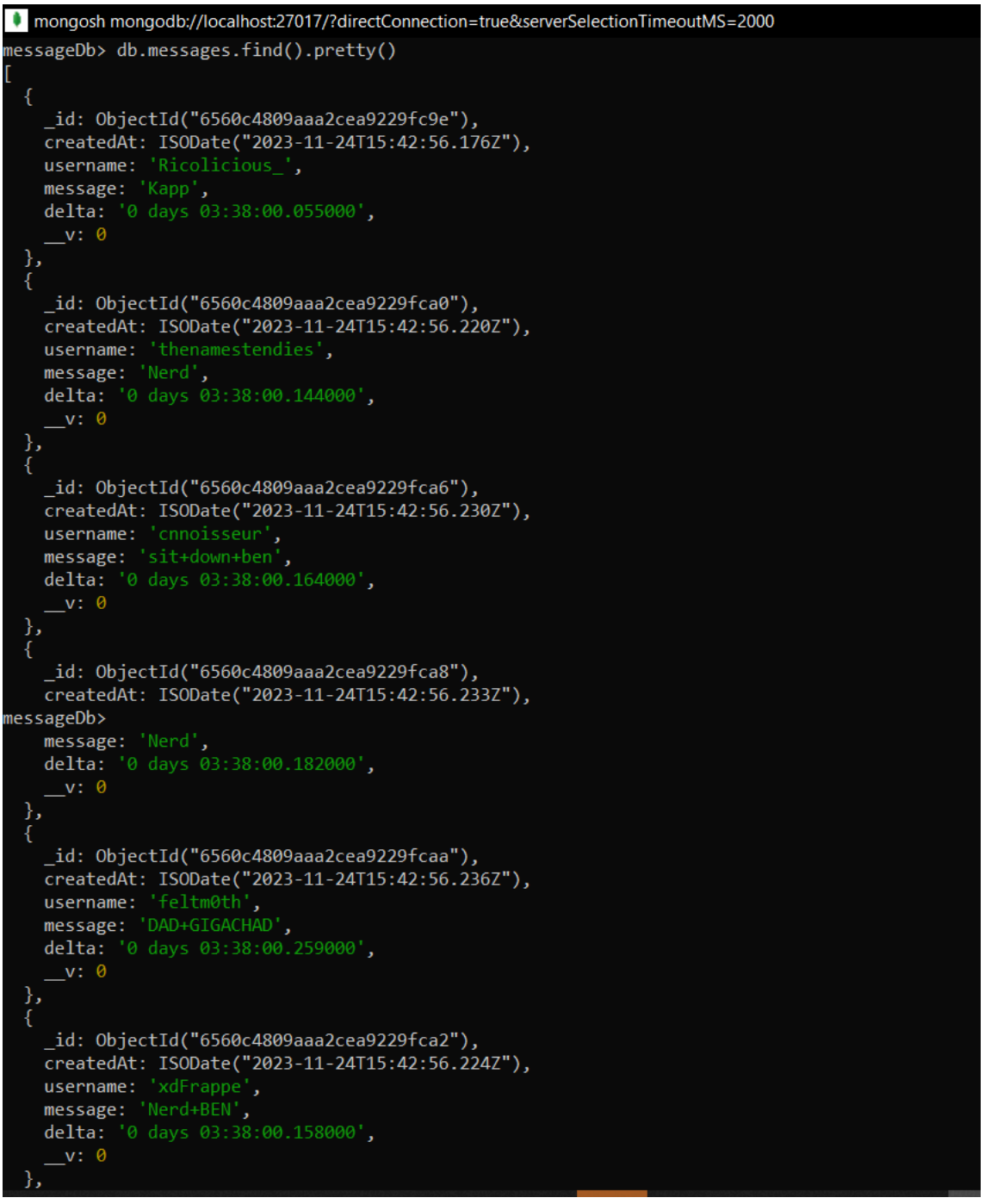


Figura 4

Últimos registros de la base de datos mongodb después de una prueba de carga realizada contra la aplicación alojada en el servidor Node.js, realizada a través del terminal mongosh.

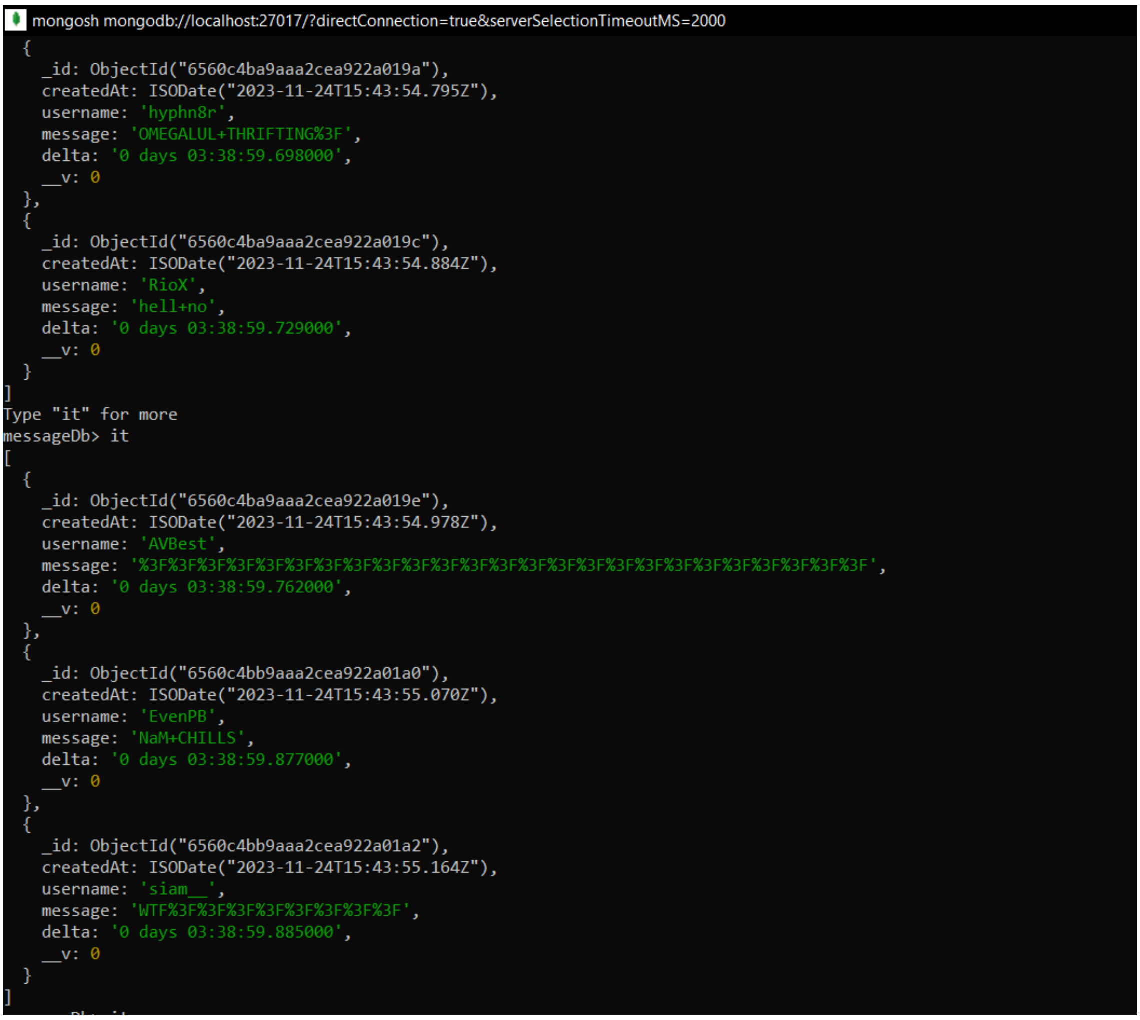


Figura 5

Primeros registros de la base de datos mongodb después de una prueba de carga realizada contra la aplicación alojada en el servidor Node.js, realizada a través del programa Mongodb Compass.

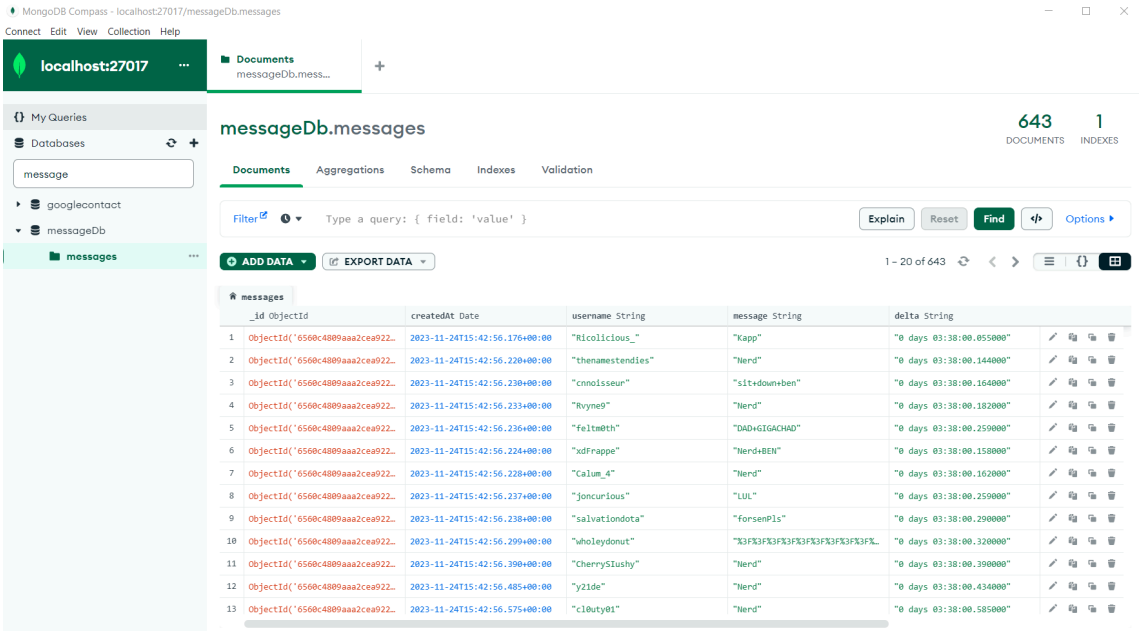


Figura 6

Últimos registros de la base de datos mongodb después de una prueba de carga realizada contra la aplicación alojada en el servidor Node.js, realizada a través del programa Mongodb Compass.