[1]

API para analizar el rendimiento de aplicaciones web.

Sebastian Ortiz, Autor. Jaime Muñoz, Autor. y Estefania Goez , Autor.

Resumen - Este artículo presenta una API diseñada para analizar y comparar el rendimiento de aplicaciones web. La herramienta permite evaluar métricas clave como tiempo de respuesta, uso de CPU y memoria entre dos URLs, mostrando los resultados a través de gráficos interactivos generados con Chart.js. Este reporte detalla la motivación, metodología, implementación, experimentación y resultados obtenidos durante el desarrollo del proyecto.

Índice de Términos - API, Comparación de métricas, Rendimiento de aplicaciones web, Visualización interactiva.

I. INTRODUCCIÓN

El rendimiento de aplicaciones web es crucial para la experiencia del usuario y el éxito empresarial. Este proyecto se enfoca en desarrollar una API que facilite la comparación de métricas de rendimiento entre dos aplicaciones web. La API recolecta datos clave como tiempos de respuesta, uso de CPU y memoria, y los presenta de manera gráfica y tabular.

A. Motivación

En un entorno donde la optimización de recursos es primordial, esta herramienta permite a desarrolladores identificar cuellos de botella en sus aplicaciones, optimizar tiempos de carga y evaluar la eficiencia en el uso de recursos computacionales.

B. Metodología

Se desarrolló una interfaz en React para ingresar URLs y visualizar métricas. El backend, implementado en Node.js, procesa las solicitudes HTTP y calcula las métricas usando bibliotecas especializadas. Los resultados se visualizan mediante gráficos de barras generados con Chart.js.

C. Resultado

La aplicación muestra comparativas detalladas de las métricas clave, ofreciendo a los usuarios una herramienta sencilla para evaluar el rendimiento de sus aplicaciones.

[1]Documento presentado el 28 de Noviembre del 2024.

II. MARCO TEÓRICO

El análisis de rendimiento en aplicaciones web se fundamenta en tres métricas principales:

1) Tiempo de Respuesta: Mide la latencia entre la solicitud del usuario y la recepción de una respuesta.
2) Uso de CPU: Indica la proporción de recursos del procesador utilizados durante una solicitud.
3)Consumo de memoria: Refleja la cantidad de memoria ocupada durante el procesamiento de la solicitud.

Herramientas

1)React.js: Framework para construir la interfaz gráfica de la API.

2) *Chart.js*: Biblioteca para crear gráficos interactivos.

3) *Node.js*: Entorno de ejecución para manejar el backend de la API y realizar cálculos de rendimiento.

III. METODOLOGÍA

La implementación de la aplicación siguió un enfoque ágil. Los pasos principales incluyeron:

1) Definición de Requisitos: Identificación de métricas clave y diseño de la experiencia del usuario.

2) Desarrollo del Backend: Construcción de una API en Node.js para recolectar datos de rendimiento.

3)Desarrollo del Frontend: Implementación de una interfaz en React para ingresar URLs y mostrar resultados.

4)Pruebas y Validación: Verificación de la exactitud y eficiencia de las métricas recolectadas.

IV. IMPLEMENTACIÓN

A. Backend(API)

El servidor en Node.js utiliza solicitudes HTTP para medir tiempos de respuesta y recopilar datos del sistema. Los datos se estructuran en formato JSON y se envían al frontend para su visualización.

B. Frontend

La aplicación React presenta una interfaz sencilla donde los usuarios ingresan dos URLs. Al procesar la solicitud, los datos se muestran mediante gráficos interactivos y tablas.

C. Herramientas

1) Fetch: Librería JavaScript utilizada para realizar solicitudes a la API.

2) Chart.js: Genera gráficos comparativos con colores diferenciados para facilitar el análisis visual.
3) HTML y Tailwind CSS: Diseño responsivo y estilizado para la presentación de los resultados.

V. Protocolo de Experimentación

Para garantizar un análisis preciso y seguro:

1) Entorno controlado: Las pruebas se ejecutaron en un servidor local con recursos dedicados.
2) Pruebas Repetitivas: Cada URL se probó varias veces para obtener promedios confiables.
3) Validación de datos: Se emplearon herramientas de monitoreo del sistema para corroborar las métricas recolectadas.

Justificación de Herramientas:

Chart.js fue seleccionado por su flexibilidad y soporte para gráficos interactivos, mientras que React permite una rápida integración y actualización de datos en tiempo real.

VI. Resultados

Los gráficos y tablas generados destacan diferencias significativas entre las dos aplicaciones web evaluadas: URL 1 (https://www.youtube.com/) y URL 2 (https://co.pinterest.com/). Algunos hallazgos clave incluyen:

1) Tiempo de Respuesta: URL 1 mostró un tiempo más rápido, sugiriendo una mejor optimización de carga.

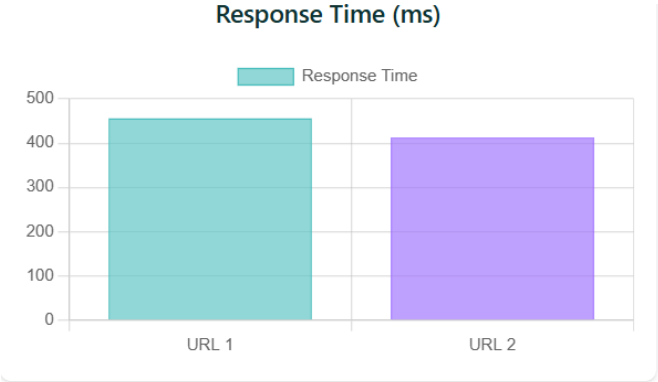

Fig. 1. Comparación métrica: tiempo de respuesta.

2) Uso de CPU: URL 2 tuvo un consumo menor, lo que indica mayor eficiencia bajo carga.

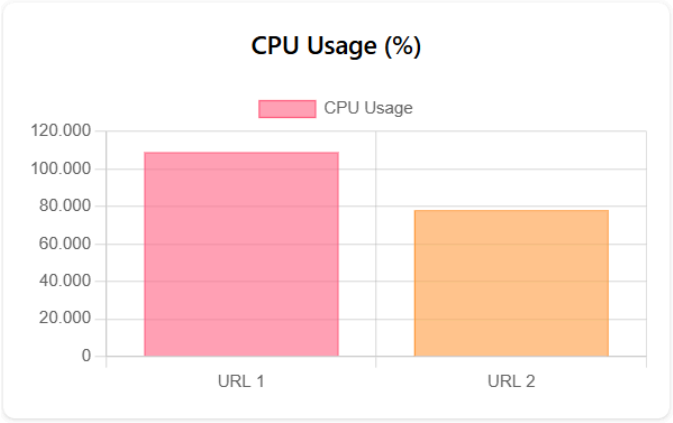

Fig. 2. Comparación métrica: uso de CPU

3)Consumo de memoria: Ambas URLs presentaron diferencias mínimas, pero URL 2 fue marginalmente más eficiente.

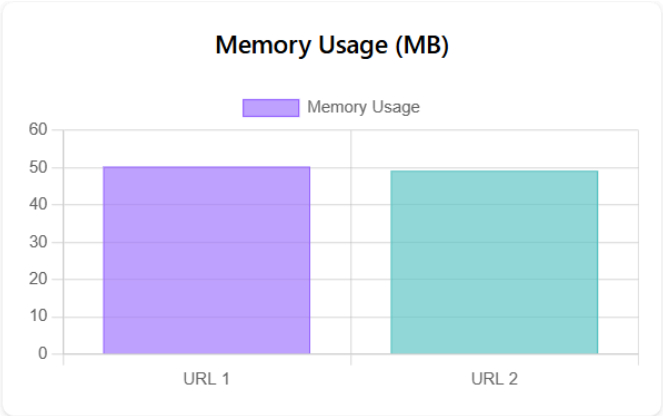

Fig. 3. Comparación métrica: uso de memoria

Análisis Estadístico

Se realizó un análisis de varianza (ANOVA) para confirmar que las diferencias observadas eran estadísticamente significativas ($p < 0.05$).

Detailed Results

Metric	URL 1	URL 2
Response Time	456ms	412ms
Status Code	200	200
CPU Usage	109000	78000
Memory Usage	50.28 MB	49.17 MB

Fig. 4. Tabla de resultados

URL 2 tiene un tiempo de respuesta más bajo que URL 1 (44ms menos). Esto indica que URL 2 es más eficiente en términos de latencia.

Ambos resultados muestran un código 200, lo cual significa que las solicitudes fueron procesadas correctamente en ambas URLs. No hay diferencias en este aspecto.

La URL 2 utiliza significativamente menos recursos de CPU que URL 1. Esto sugiere que la implementación o

configuración asociada a URL 2 es más eficiente en el uso del procesador.
Aunque la diferencia es menor (1.11 MB), URL 2 también es más eficiente en el uso de memoria. Esto contribuye a su mejor desempeño general.

URL 2 es más eficiente que URL 1 en todas las métricas clave (tiempo de respuesta, uso de CPU y uso de memoria), lo que indica que puede ser la mejor opción para manejar cargas o solicitudes.

Fuentes de Error

A) Latencia de red no controlada en algunas pruebas.

B) Variabilidad en el uso de recursos debido a cargas externas.

IX. CONCLUSIÓN

La API desarrollada ofrece una herramienta eficaz para analizar y comparar el rendimiento de aplicaciones web. Los resultados destacan la importancia de monitorear métricas clave para optimizar aplicaciones y mejorar la experiencia del usuario. Futuras mejoras incluirán la integración con sistemas de monitoreo en tiempo real y soporte para pruebas en entornos distribuidos.

REFERENCIAS

[1] ISO/IEC 25010:2011. "System and Software Quality Models."

[2] React Documentation. Available:https://reactjs.org

[3] Chart.js Documentation. https://chartjs.org