

INTERFAZ PARA EVALUAR EL RENDIMIENTO DE UNA APLICACIÓN WEB

Hernández-Rueda Karen¹, Martínez Vargas Martha Patricia², Damián Casillas Mario³

^{1,2}University of Guadalajara-CUCEA, ³Mitratech, INC

^{1,2}Information Systems Department, ³Cloud Services

Periférico Norte N° 799, Núcleo Universitario Los Belenes, C.P. 45100, Zapopan, Jalisco, México.

Tel. ^{1,2} +52 (33) 3770 3300, ³ +52 (33) 1987 9053

¹karen.hrueda@academicos.udg.mx, ²martha.mvargas@academicos.udg.mx, ³mario.damian@gmail.com

RESUMEN.

Este trabajo presenta una implementación en código abierto de una Interfaz de Programación de Aplicación para evaluar el rendimiento de una aplicación web por medio del registro de variables de la interfaz de sincronización de navegación. La evaluación se realiza con registros de tiempos de cinco usuarios que acceden a diferentes aplicaciones web. La percepción de satisfacción depende del tiempo de carga, entre mayor sea el tiempo, los usuarios tendrán una percepción negativa. El objetivo del trabajo es proporcionar una opción para valorar el rendimiento de cualquier aplicación web de forma sencilla al integrar la interfaz en la plataforma del navegador web. Los resultados muestran los tiempos de retraso que tiene un usuario y esos coinciden con lo indicado en la literatura. La utilidad de contar con esta interfaz es la posibilidad de registrar estadísticos durante el acceso y mejorar la experiencia del usuario para garantizar su permanencia.

Palabras Clave: API, rendimiento web, aplicaciones web, métricas.

ABSTRACT.

This work presents an open-source implementation of an Application Programming Interface to evaluate the performance of a web application by registering variables from the navigation synchronization interface. The evaluation is carried out with time records of five users who access different web applications. The perception of satisfaction depends on the loading time, the longer the time, the users will have a negative perception. The objective of the work is to provide an option to evaluate the performance of any web application in a simple way by integrating the interface into the web browser platform. The results show the delay times that a user has and those coincide with what is indicated in the literature. The usefulness of having this interface is the possibility of registering statistics during access and improving the user experience to guarantee their permanence.

Keywords: API, web performance, web applications, metrics

1. INTRODUCCIÓN

Si un usuario percibe un tiempo de respuesta lento cuando accede a una aplicación o página web entonces lo más probable es que abandone esa actividad, y si el problema es recurrente, es posible que no continúe en la compañía que le proporcione el servicio. El término de aplicaciones o páginas web se considera como sitios web con funcionalidad y elementos interactivos. Las aplicaciones web desempeñan un papel importante para cualquier actividad de un negocio e intercambio de información, es una carta de presentación para los clientes, por lo que no es suficiente satisfacer las necesidades funcionales de la aplicación o página, sino que también se debe poner atención en lo que experimentan

los usuarios para determinar su éxito o fracaso [1]. Si una página o aplicación web tarda mucho en cargarse entonces el usuario se cansará y dejará el acceso, y es probable que no la visite nuevamente. Por lo que el tiempo de carga de una aplicación o página web es importante para cualquier negocio, según [2], Shospzilla redujo su tiempo de carga de 7 a 2 segundos y su publicidad creció un 112%, Amazon calculó que 0.1 segundos de retraso le implican una pérdida del 1% en sus ingresos, e Insideline tuvo un 17% más de visitas a su página cuando disminuyó su tiempo de carga de 9 a 1.4% segundos, y para reducir la carga lenta recomienda optimizar las imágenes, reducir el número de consultas del sitio a una Base de Datos (DB en inglés), desinstalar plugin innecesarios, entre otros. Asimismo, recomienda las herramientas PageSpeed Insights de Google, YSlow de Yahoo, GTmetrix, WebPageTest y Pindom Tools para medir la velocidad de carga, pero todas tienen un costo de uso a excepción de PageSpeed Insights y WebPageTest, aunque el primero tiene limitación de conexión y no permite pruebas personalizadas, por otro lado, GTmetrix da un informe más completo de rendimiento [3]. En [1] se registra la actividad de cinco usuarios que acceden a 10 aplicaciones web para probar que el tiempo de respuesta del usuario disminuye a medida que disminuye el tiempo de respuesta del sistema, con base en los tiempos de respuesta de 0.1 segundos, 1 segundos y 10 segundos de [4] y usa el analizador PageSpeeds Insights que mide el rendimiento de una red de 0 a 100 puntos, entre más puntos, mejor el rendimiento. Además, en [5] se indica que un usuario de una aplicación web empieza a percibir retraso en el acceso entre un rango de 100-300 ms. Por lo que es deseable garantizar un tiempo de acceso menor por parte de un proveedor de servicios de aplicaciones basadas en el navegador web y se debe medir los tiempos que toma cargar y ejecutar la aplicación que proporciona, es decir, valorar el rendimiento que tienen sus aplicaciones para identificar y generar estrategias que den solución a los problemas particulares. Además, de las herramientas mencionadas como YSlow y GTmetrix, existen diversas soluciones comerciales para el análisis de las variables clave en el rendimiento de una aplicación web, como por ejemplo, las aplicaciones DataDog, Microsoft Application Insight y New Relic, que ofrecen servicios de monitoreo de componentes y aplicaciones, aunque estas consideran más variables o métricas de análisis, también tienen costo, están orientadas a arquitecturas de tenencia múltiple (una sola instancia de una aplicación que se ejecuta en el servidor sirve a múltiples clientes) y es complejo su uso para tenencia simple

(una sola instancia de una aplicación que se ejecuta en el servidor que sirve a un cliente). Por lo que la propuesta que se presenta es una opción para arquitectura de tenencia múltiple más simple que las otras aplicaciones propuestas, por medio de una Interfaz de Programación de Aplicación (API o Application Programming Interface) de rastreo denominada API REST (Representational State Transfer) que usa código abierto. En la Figura 1 se muestra un ejemplo de funcionamiento de una API, permite que sus servicios o productos se comuniquen con otros. La interfaz identifica de manera única la aplicación y el usuario que usa la aplicación web a la que se desea evaluar su rendimiento. Esta API se integra en la aplicación web como un componente y al cliente o navegador web de forma más sencilla que otras interfaces, la interfaz recolecta las variables o métricas de la API de sincronización de navegación (Navigation Timing) y la API de rastreo las almacena, para que se pueda evaluar el rendimiento de la aplicación web en uso.



Figura 1. Ejemplo de funcionamiento de una API. Fuente [6]

En la siguiente sección se presenta el marco teórico que describe los términos básicos para que se entienda la propuesta. Luego se presenta la propuesta con el proceso de registro de las variables del rendimiento de una aplicación web entre un cliente y un servidor para realizar su evaluación, los resultados y, por último, se dan las conclusiones con información de trabajo futuro.

2. MARCO TEÓRICO

2.1. HTML

Una aplicación web se accede a través de un navegador (Firefox, Chrome, Safari, etc.) que se encarga de mostrar en una ventana la información que solicitó un usuario (cliente) a un servidor (conjunto de computadoras que atienden peticiones y devuelven una respuesta) de recursos web. El recurso comúnmente es un documento HTML (Hyper Text Markup Language o hiper texto), pero puede ser un archivo PDF o imagen, entre otros. HTML es un lenguaje estándar reconocido globalmente y normado por el organismo sin fines de lucro W3C (World Wide Web Consortium o Consorcio Mundial de la Red) que se utiliza para la creación de páginas web y permite que éstas puedan ser interpretadas de la misma forma por cualquier navegador web que se acceda por medio de Internet como Explorer, Mozilla o Chrome. En 1995 se estandarizó HTML como HTML 2.0, en 1997 se publicó la versión HTML 3.2 que tuvo como avances incrustar un programa en un documento HTML y texto que fluye alrededor de las imágenes [7]. En 1998, se publicó HTML 4.0 que introdujo las hojas de estilo en cascada o CSS (Cascading Style Sheet), la posibilidad de incluir pequeños programas (código de aplicaciones) o scripts (secuencia de comandos) en las páginas y

mejora de la accesibilidad de las páginas diseñadas [7]. CSS son un lenguaje de diseño gráfico que se usa para definir y crear la presentación de un documento de hiper texto. Los scripts se usan para instituir aplicaciones, manejar datos, e introducir información a páginas web, y se usan tanto por el cliente, es decir, cualquier usuario que quiera acceder a una aplicación web, como por el servidor, que es donde se ubica la aplicación web. En 2014, se presentó HTML5 para el desarrollo de aplicaciones multiplataformas (Linux, Windows, Android, iOS, etc) que integra HTML junto a JavaScript (JS, lenguaje de scripts o lenguaje de programación interpretado) y CSS3 (versión 3 de CSS) [7]. JS se usa como complemento de HTML y CSS para crear páginas webs debido a que funciona en los navegadores de forma nativa, es decir, se ejecuta directamente [8]. Normalmente, los contenidos y el diseño de la página web son responsabilidad de diferentes personas para que sea más fácil su mantenimiento, por lo que HTML se puede ver como una parte de contenido más presentación, es decir, XHTML (eXtensible Hyper Text Markup Language) o XML (eXtensible Markup Language) más CSS [9].

2.2. Partes de un navegador

Un navegador necesita de una interfaz de usuario, motor de búsqueda, motor de renderización, red, servidor de interfaz, intérprete de JS y almacenamiento de datos para que pueda mostrar una página web [10]. Adicionalmente, cada navegador tiene características únicas. La interfaz de usuario (user interface) incluye todas las partes visibles del navegador (barra de direcciones, botón de avance, etc.) a excepción de la ventana principal que contiene la página solicitada. El motor de renderizado (rendering engine) se encarga de mostrar el contenido solicitado (es el responsable de analizar el código HTML, CSS y mostrar el contenido analizado en la pantalla si se accede a una página web). El motor de búsqueda (browser engine) coordina las acciones entre la interfaz de usuario y el motor de renderización. Existen dos motores de renderización básicos para navegadores de código abierto, para Firefox se usa Gecko que es un motor propio de Mozilla y para Chrome o Safari se usa el WebKit [10]. Este último se usa en plataforma Linux, pero se modificó por Apple para hacerlo compatible con Mac y Windows.

2.3. Acceso a una página web

Una vez que se solicita la búsqueda de alguna página, el motor de renderización comienza a analizar el documento HTML y construye el modelo de objetos del documento o DOM (Document Object Model), es decir, convierte las etiquetas en un árbol de contenido o una API para documentos HTML. El DOM define la estructura lógica de los documentos y el modo en que se accede y manipula el contenido de documentos HTML. Luego analiza las hojas de estilo (CSS externos como los elementos de estilo), necesita el CSS que muestra el diseño visual de los documentos web, y utiliza el modelo de objetos CSS o CSSOM (CSS Object Model) de manera paralela para definir las APIs que se usan para consultas de medios, selectores y el propio CSS [9]. Las CSS junto con las instrucciones visuales del código HTML se usan para crear el árbol de renderizado (presentación), este

contiene rectángulos con atributos visuales como color y dimensiones, organizados en el orden que deben aparecer en la pantalla. Después se inicia el proceso de diseño (layout), a cada nodo se le asignan coordenadas exactas del lugar en que deben aparecer en la pantalla y, por último, se hace la pintura (painting) que consiste en recorrer el árbol de renderizado y pintar cada nodo con la capa de servidor de la interfaz de usuario. En la Figura 2 se puede ver el flujo de un motor WebKit [10]. Los modelos DOM y CSSOM se complementan para representar una página web, por medio de un árbol de renderizado diseñado para ubicar la posición y tamaño de su contenido con un formato que permite la visualización del contenido en pantalla. La página web se accede al ejecutar los scripts, escritos con en JS, que pueden modificar la estructura del DOM. Por lo que la construcción de objetos DOM no se realiza si no se ejecutan los scripts y estos sólo se ejecutan si el CSSOM está disponible.

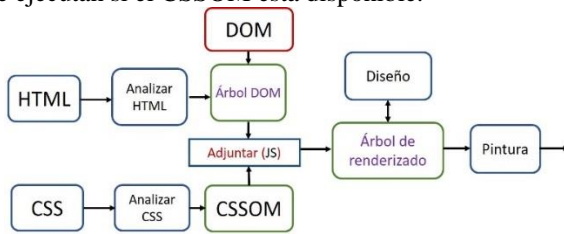


Figura 2. Flujo WebKit. Fuente [9]

2.4. Rendimiento de una página web

El rendimiento de una página web es el tiempo que se demora en cargarse desde su petición y entre mayor sea el rendimiento más fluido es el acceso a la página, con menor tiempo y más agradable para los usuarios [11]. El rendimiento de la aplicación depende directamente de cómo se resuelven las dependencias entre el marcado HTML, la hoja CSS y los scripts de acuerdo con la Figura 2. En [12] se señala que el rendimiento o velocidad del procesamiento de una página web dependen de la complejidad de la tarea o actividad que debe realizarse, el tipo de negocio y las expectativas del usuario. Por lo que el rendimiento implica una medición objetiva (tiempo de carga y respuesta, cuadros por segundo) y la experiencia percibida por el usuario del tiempo de carga y el tiempo de ejecución (tiempo interactivo y receptivo, así como la fluidez del contenido con las interacciones del usuario) [13]. De acuerdo con [5], si el tiempo de carga es de un segundo entonces el usuario deja de hacer la actividad por el retardo de la carga, y si pasan 10 segundos sin que se le mande un mensaje al usuario sobre el progreso, abandona el acceso. Entonces es importante medir los tiempos de acceso y estos se pueden monitorear a través de la Navigation Timing API que propuso el grupo de trabajo de rendimiento Web o W3W (Web Performance Working Group), esta API es soportado por la mayoría de los navegadores modernos y tiene como objetivo proporcionar información de sincronización de navegadores para medir el rendimiento de una aplicación web por medio de las variables o métricas que contiene su estructura. En la Figura 3 se muestran las variables que se usan para registrar el tiempo de los eventos de todos los recursos de una petición Web. Por ejemplo, la métrica *navigationStart* registra el momento en que comenzó la

navegación, *loadEventEnd* guarda el tiempo a la que regresa el controlador de eventos de carga, *requestStart* proporciona la hora a la que inició la solicitud, *responseEnd* da el tiempo en que se terminó de recibir la respuesta, entre otros. Si se quiere determinar por ejemplo el tiempo total de carga de la página Web entonces es común determinar la diferencia entre las variables *loadEventEnd* y *navigationStart*, o si se quiere calcular el tiempo de respuesta de la solicitud se hace una diferencia entre las variables *responseEnd* y *requestStart* [14] [5].

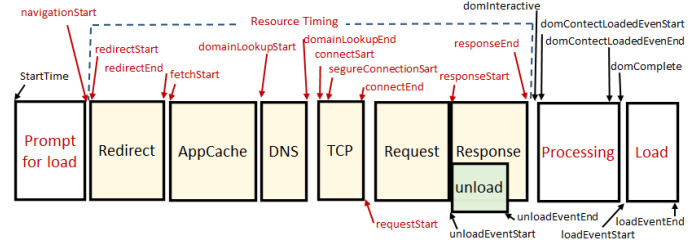


Figura 3. Estructura de la API de sincronización. Fuente [5]

3. PROPUESTA

3.1. Recuperación de las métricas

Es necesario el registro de la medición de las métricas mostradas en la Figura 3 para que se realice la medición del rendimiento y eso se hace con un sistema de mensajería asíncrona (no requiere una conexión al mismo tiempo). El sistema usa un modelo de mensajería conocido como publicación/suscripción (pub/sub messaging) de servicio a servicio sin contar con un servidor e implica editores, suscriptores y mensajes. Los editores (remitentes) de mensajes clasifican los mensajes de alguna manera, pero no lo dirigen específicamente a un suscriptor (receptor) porque desconocen su existencia. Asimismo, los suscriptores se registran para recibir ciertas clases de mensajes sin conocimiento de los editores [15]. Además, se usa la analítica del cliente, que implica el registro de sus eventos, para determinar las interacciones de los usuarios con los sistemas y recopilar datos de las métricas. Las métricas se registran con la ocurrencia de los eventos (un evento puede ser el uso de una aplicación o recurso) y se recuperan al momento de su carga y ejecución. La recolección de las métricas (*loadEventEnd* y *loadEventStart*) se realiza con una API REST que permite a los usuarios conectarse e interactuar con servicios en la nube. La API REST (API de rastreo) que se propone es una interfaz entre sistemas que usan HTTP (Hypertext Transfer Protocol o Protocolo de Transferencia de Hipertexto) para acceder a las páginas web por medio de Internet. Las métricas se capturan y adecuan con el formato JSON (JavaScript Object Notation o Notación de Objetos de JavaScript) que es un formato sencillo para el intercambio de datos. La API REST se usa en alta demanda por medio de un servidor de DB (Data Base o Base de Datos) llamado Mongo DB, que es un sistema de DB orientado a documentos de código abierto, es decir, los datos se guardan en estructuras de datos denominada colecciones en lugar de hacerlo en tablas. La API REST se encarga de desacoplar el rastreo del servidor de aplicación principal para habilitar un servicio dedicado para este fin, proveer persistencias de los mensajes de rendimiento y permitir escalabilidad de manera

horizontal del sistema para soportar el crecimiento del flujo de datos. La construcción de rastreo se realiza con Node.js, un servidor de aplicaciones de código abierto que utiliza el lenguaje de programación ECMAScript (publicado por ECMA Internacional) que actualmente es el estándar ISO 16262 y está basado en JavaScript. La recolección de datos se realiza como una estructura de datos persistentes, es decir, siempre se preserva sus versiones anteriores después de ser modificados, son representados en JS y actualizados mediante Mongoose, que es una librería de Node.js para almacenar documentos en Mongo DB.

3.2. API de rastreo para registrar las métricas

La Figura 4 muestra la arquitectura de la solución de rastreo. Es necesario que el cliente inicie una sesión que permita el acceso a alguna aplicación web y el acceso se debe identificar para evaluar el rendimiento de esa aplicación. La identificación del acceso se hace con una llamada API de rastreo que da una respuesta con un UUID (Universally Unique Identifier o Identificador Único Universal) que es una cadena de texto (ejemplo 99bdb232-7dd6-4aa6-758e6e7bcc91) que se almacena en el servidor web. El sistema genera un UUID (Figura 5, sección roja) distinto sin importar si dos accesos son de un mismo usuario o diferente, durante el inicio de sesión tanto del lado del cliente (identificador del usuario) como del servidor de la aplicación (identificador de la aplicación), así como en el servidor de rastreo (clientes web) que hace un rastreo de usuario.

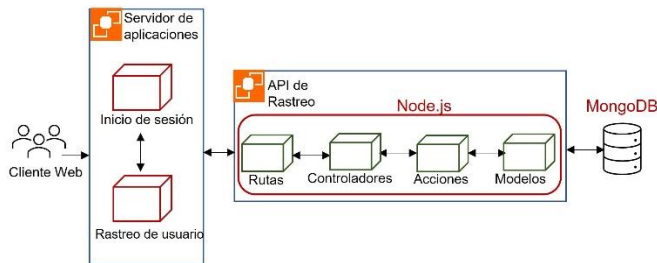


Figura 4. Arquitectura de la solución de rastreo. Fuente propia.

La API de rastreo contiene rutas, controladores, acciones y modelos. Las rutas definen los puntos finales de cada uno de los recursos de servicio, los controladores realizan la lógica necesaria para cumplir con la ruta solicitada, las acciones son las operaciones que se ejecutaran en los modelos, y los modelos son los mecanismos para interactuar con los datos persistentes. Las interacciones entre los componentes internos de la API inician con una ruta, cada ruta usa un controlador que utiliza una acción que se relaciona a un modelo y la acción, dependiendo de su uso, utiliza uno de los métodos del modelo que es implementado por el DOM. Los componentes de la API son usuario, eventos y métricas. Cada usuario integra componentes individuales que contiene modelo (models/Users.js), acción (action/userActions.js) y controlador(controllers/Users.j) que se implementan en la ruta /routes/index.js y allí mismo se archivan. Igualmente, los eventos

tienen el componente models/Events, actions/eventsActions.js y controller/Events.js de forma individual y se implementan en la misma ruta que contiene la estructura que servirá para almacenar los datos del evento. Asimismo, las métricas de rendimiento usan los componentes models/Metrics.js, actions/metricsActions.js y controller/Metrics.js individuales. En las métricas, el modelo contiene la estructura de toda la información disponible del rendimiento, las acciones obtienen la petición del registro de la métrica, y el controlador organiza las variables obtenidas del Navigation Timing API, que se implementan en la misma ruta de los usuarios y eventos. En la Figura 5, se muestra que debe hacerse una transacción previa del registro del cliente para que el servidor de aplicaciones almacene al cliente y luego se realicen peticiones con la API de rastreo.

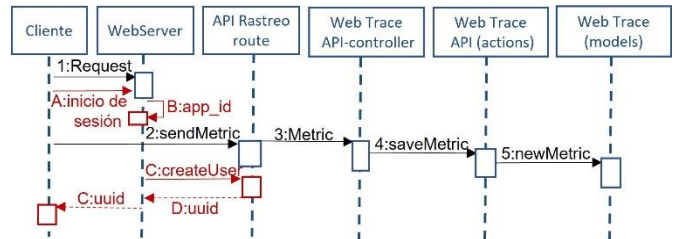


Figura 5. Secuencia de recolección de métricas. Fuente propia.

3.3. Implementación de la interfaz

La interfaz contiene la parte de identificación de usuario (user_id) y aplicación (app_id), las acciones de las métricas y su controlador, así como el registro de las métricas y la petición para obtenerlas con la idea de evaluar el rendimiento. La función para generar el identificador de usuario se muestra en la Figura 6, ésta usa una función callAPI que está escrita en PHP (Hypertext Preprocessor o Procesador de Hipertexto) del lado del servidor, para hacer llamadas a la API de REST. La API REST a su vez usa una llamada POST, solicita la inserción de la entidad enviada, con la estructura JSON (asigna el user_id y app_id) a la API de rastreo y la respuesta de la API de rastreo es un UUID.

```
function getUserTrackerUUID($app_id,$user_id,$url)
{
    $data_array=array(
        "app_id" => $app_id,
        "user_id" => $user_id,
    );
    $make_call=callAPI('POST', $url,json_encode($data_array));
    $response= json_decode($make_call,true);
    $errors=$response['response']['errors'];
    $data=$response['response']['data'][0];
    return $make_call;
}
```

Figura 6. Código para identificación de usuario y aplicación.

En la Figura 87 se muestran los códigos de las acciones de las métricas y en la Figura 8 el controlador de las métricas implementadas. Cuando se inicia una aplicación, el servicio se especifica en el puerto donde se está escuchando, se importan las rutas disponibles y el uso del archivo de configuración, para ubicar el servidor de MongoDB. Así es posible hacer peticiones de registro de clientes o usuarios, eventos y métricas.

```
//Location: actions/metricsActions.js
const Metric= require("../models/Metrics");
const reportMetric= (data)-> {
    return Metric.create(data);
}
const getAllDataMetrics= () => {
    return Metric.find({});
}
module.exports = {
    reportMetric,
    getAllDataMetrics
};
```

Figura 7. Código de las acciones de las métricas.

```
//Location: controllers/Metrics.js
const {reportMetric, getAllDataMetrics}= require("../actions/metricsActions");
const createMetric = (req,rest)-> {
    let totalRequest = req.body.responseStart - req.body.responseStart;
    let totalResponse = req.body.responseEnd - req.body.responseStart;
    let totalRequestResponse = req.body.responseEnd - req.body.requestStart;
    let total = req.body.unloadEventEnd - req.body.connectStart;
    req.body.totalRequest = totalRequest;
    req.body.totalResponse = totalRequestResponse;
    req.body.total= total;
    req.body.dale= Date.now();
    reportMetric(req.body).then(event) => {
        res.status(201).json(event);
    }).catch(e => res.status(400).json(e));
}
module.exports = {
    createMetric,
    getAllMetrics,
    getMetrics
};
```

Figura 8. Código del controlador de las métricas.

La integración de las rutas se hace con los métodos POST o GET en el archivo de rutas localizado en el archivo routes/index.js con el código de la Figura 9.

Las métricas se integran a los componentes del cliente mediante el script de JS con una función denominada métrica, es decir, se integra el script a cada recurso que se desea medir, dentro del cuerpo de HTML, que hace la llamada a la API de rastreo por medio de una llamada API por cada recurso o página web en donde se quiera medir el rendimiento. La función métrica se muestra en la Figura 10, genera una petición a la API de rastreo para que se agreguen las variables de rendimiento al cliente, que recibe una estructura JSON que tiene el registro de las métricas solicitadas con los parámetros especificados en el controlador de las métricas, esta se almacena y es contenida en el objeto window.performance.timing, que tiene todas las variables para la evaluación del rendimiento que se obtiene de la Navigation Timing API. Una vez procesado el método, este confirma la transacción de manera exitosa con un código 200 Ok.

```
//Location: routes/index.js
const express = require("express");
const {createUser, getAllUsers} = require("../controllers/Users");
const {createEvent} = require("../controllers/Events");
const {createMetric, getAllMetrics, getMetrics} = require("../controllers/Metrics");
const router = express.Router();
//User
router.get("/getallusers",getAllUsers);
router.post("/createuser",createUser);
//Events
router.post("/createevent",createEvent);
//Create Metric
router.post("/createmetric",createMetric);
router.get("/getallmetrics",getAllMetrics);
router.get("/getmetrics/:id",getMetrics);

module.exports = router;
```

Figura 9. Rutas de la aplicación.

```
function metric (url_api,uuid,url){
    var url=url_api;
    var data=window.performance.timing.toJSON();
    data.uuid=uuid;
    data.url=url;
    fetch(url, {
        method: 'POST',
        body: JSON.stringify(data),
        headers:{
            'Content-Type':'application/json'
        }
    }).then (res=> res.json())
    .catch(error=>console.error('Error:',error))
    .then (response=> console.log('Success:', response));
}
```

Figura 10. Función para agregar las métricas de rendimiento.

4. RESULTADO

La interfaz de rastreo se agregó a la parte del servidor de aplicaciones de una empresa privada y se realizó pruebas de acceso de cinco clientes a una aplicación web administrativa ofrecida por la empresa para evaluar su rendimiento, donde se tomó el 30% de los módulos más utilizados. Los clientes usaron un navegador de su elección para iniciar sesión desde sus respectivos lugares de trabajo a la plataforma de Windows con su respectivo proveedor de servicio de Internet. El tiempo de carga de la aplicación por cada cliente se puede ver en el Tabla 1, fue obtenido desde el navegador web y enviado a la API de rastreo, que se promedió con base en 1 hora de servicio. El cliente (número decimal) representa a un UUID para reducir espacio en la tabla. La evaluación del rendimiento se realizó con la percepción de estos usuarios, con base en sus acciones dentro de la página y el retardo de respuesta dentro de un rango.

Tabla 1. Tiempo de carga de la aplicación web por cliente.

Cliente	Promedio 1 0-100ms	Promedio 2 100-300ms	Promedio 3 300-1000ms	Promedio 4 +1000ms
1	50.2	207.75	684.033333	5683.45
2	55.5	195.675	620.566667	5914.1
3	74.8	210.575	625.1	4893.45
4	63.4	187.7	663.4	5819.05
5	54.4	184.4	651.4	4858.2

El 90% de los accesos (Promedios 2, 3 y 4, Tabla 1) tuvo una percepción de rendimiento aceptable. Por ejemplo, 40% de los accesos (Promedio 2 Tabla 1) percibieron retraso en las peticiones, que se registró con la actualización del navegador, el 20% (Promedio 4 Tabla 1) percibieron probable cambio de contexto u olvido de lo que estaba haciendo, registrado por acciones innecesarias, y cuando el tiempo de carga estuvo entre 100 y 300 ms (Promedio 4 Tabla 1), en promedio, 30% los accesos percibieron que estaba trabajando aplicación. Nada más el 10% de los accesos tuvo un rendimiento adecuado, es decir, se percibió de forma instantánea (Promedio 1 Tabla 1).

Por otro lado, se realizó la evaluación del servidor de aplicaciones de un servidor Apache HTTP (Tabla 2) con el uso de servicios de AWS (Amazon Web Services) para obtener métricas que sirven a nivel de negocio. El resultado muestra que la totalidad de los recursos de librerías tanto de terceros como de la aplicación están contenidas en cada aplicación y cada aplicación contiene todos los

archivos de la aplicación incluidas las librerías de terceros y propias. La evaluación de advertencias es del 0% y a la detección de errores es del 70%. Estos resultados ayudan a identificar recursos duplicados y optimizar la entrega de recursos en un siguiente trabajo.

Tabla 2. Análisis del servidor Apache HTTP.

Tipo de contenido	Tenencia simple	Resultado
Librería de terceros	Se almacena en la aplicación del cliente	100%
Librerías propias	Se almacena en la aplicación del cliente	100%
Registro y evaluación de advertencias	Se analizan y evalúan	0%
Errores	Se analizan y evalúan	70%

5. CONCLUSIONES

En este trabajo se presentó la implementación de una interfaz basada en una API REST para evaluar el rendimiento de una aplicación web con la obtención de las métricas de rendimiento de la API de sincronización de navegación. La medición de las métricas de tiempo de carga y ejecución se evaluó con el tiempo transcurrido desde el inicio hasta el fin de la carga de la aplicación web. La evaluación se realizó con cinco usuarios desde diferentes accesos y sistemas operativos, y el resultado verifica lo que indica la referencia [5]. Con los datos obtenidos se puede concluir que en el 90% de las métricas obtenidas existe una percepción de retraso en la entrega de información del sistema y en un 100% de los casos las librerías externas están presentes en cada aplicación. Con la implementación de esta interfaz es posible mejorar el servicio al cliente porque se pueden identificar problemas de rendimiento que afectan directamente al usuario final y se puede guardar estadística para revisar los parámetros de calidad del servicio, así como la opinión del cliente. Como trabajo futuro, es posible hacer una modificación en la interfaz para medir el rendimiento de una aplicación web a un cliente en particular que presente quejas del servicio para determinar la problemática y darle una solución de forma rápida o a corto plazo. Es necesario que un proveedor de servicio web esté al pendiente de las quejas de sus clientes para que garantice la permanencia y satisfacción de estos, así como la estabilidad de sus negocios. Con la evaluación de la aplicación web realizada se pueden establecer estrategias de calidad de servicio al cliente porque se pueden determinar los problemas que afectan el rendimiento.

6. REFERENCIAS

- [1] D. Echeverría, «Tiempo de Respuesta y Experiencia de Usuario, Estudio Experimental,» *Revista Latinoamericana de Ingeniería de Software*, vol. 5, n° 4, pp. 231-234, 2016.
- [2] P. Ruiz Tauste, «LMWP Lo Mejor de WordPress,» 18 noviembre 2020. [En línea]. Available: <http://www.ipixelestudio.com/medir-mejorar-velocidad-carga-pagina-web/>. [Último acceso: 28 junio 2021].
- [3] C. E. Suazo, J. Smith, L. Halsey y S. Mandal, «TrusRadious,» 17 enero 2020. [En línea]. Available: <https://www.trustradius.com/compare-products/google-pagespeed-insights-vs-gtmetrix>. [Último acceso: 28 junio 2021].
- [4] J. Nielsen, «NN/g Nielsen Nomran Group,» 1993. [En línea]. Available: <https://www.nngroup.com/articles/response-times-3-important-limits/>. [Último acceso: 28 junio 2021].
- [5] I. Grigorik, *High Performance Browser Networking*, vol. 53, O'Reilly Media Inc, 2013, p. 54.
- [6] J. S. Viniegra, «Blog de programación avanzada de software,» 12 septiembre 2020. [En línea]. Available: <https://proasw.wordpress.com/2020/09/12/estrategias-de-sincronizacion/>. [Último acceso: 29 junio 2021].
- [7] S. Luján-Mora, *Programación de aplicaciones web: historia, principios básicos y clientes web.*, San Vicente, Alicante: Universitario, Editorial Club, 2002, p. 354.
- [8] Oracle, «Java,» 2016. [En línea]. Available: https://www.java.com/es/download/help/java_javascript.html. [Último acceso: 28 junio 2021].
- [9] J. C. Orós y M. N. Nava Martínez, *Diseño de páginas web con: XHTML, JavaScript y CSS*, Ra-Ma, 2010, p. 376.
- [10] «WebKit,» [En línea]. Available: <https://webkit.org/>. [Último acceso: 28 junio 2021].
- [11] D. Palacios, J. Guamán y S. Contenido, «Análisis del rendimiento de librerías de componentes Java Server Faces en el desarrollo de aplicaciones web,» *NOVASINERGIA*, vol. 2, n° 1, pp. 54-59, 2018.
- [12] M. Beasley, *Practical Web Analytics of User Experience: How Analytics Can Help You Understand Your Users*, Waltham, MA: Elsevier Inc, 2013.
- [13] MDN, «MDN Web Documents,» 22 junio 2021. [En línea]. Available: <https://developer.mozilla.org/es/docs/Learn/Performance>. [Último acceso: 28 junio 2021].
- [14] Z. Wang, «W3C,» 17 diciembre 2012. [En línea]. Available: <https://www.w3.org/TR/navigation-timing/>. [Último acceso: 29 junio 2021].
- [15] N. Narkhede, G. Shapira y T. Palino, *Kafka: The Definitive Guide Real-Time Data and Stream Processing at Scale*, S. Cut, Ed., Sebastopol: O'Reilly Media, Inc., 2017, p. 322.