Evaluación del rendimiento de una aplicación web

Evaluating the performance of a web application

DOI: 10.46932/sfjdv3n1-034

Received in: Dec 30st, 2021

Accepted in: Jan 1th, 2022

Karen Hernández-Rueda

PhD in Information Technologies

University of Guadalajara

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

E-mail: karen.hrueda@academicos.udg.mx

Martha Patricia Martínez Vargas

PhD in Information Technologies

University of Guadalajara

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

E-mail: martha.mvargas@academicos.udg.mx

Mario Damián Casillas

Master in Information Technologies

Mitrtech, INC

Av Adolfo López Mateos Nte 95, Italia Providencia, 44648 Guadalajara

E-mail: mariodamian@acm.org

RESUMEN

Este trabajo presenta una implementación de una aplicación llamada API REST para evaluar el rendimiento de una aplicación web por medio del registro de las variables de la interfaz de sincronización de navegación. La aplicación se base en código abierto como Mongo y Node.js. La evaluación del rendimiento considera el retardo de carga y ejecución de la aplicación en un navegador web, e implica una percepción de satisfacción por parte del usuario que depende del tiempo de carga, si transcurre entre 100 y 300 ms, la percepción se considera negativa y es posible que el usuario la abandone. Por lo que es importante para cualquier proveedor de servicios, considerar esos tiempos para mejorar su calidad y asegurar la permanencia del cliente. El objetivo del trabajo es proporcionar una opción para valorar el rendimiento de cualquier aplicación web de forma sencilla, al integrar la aplicación 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 implementation of an application called REST API to evaluate the performance of a web application by registering the variables of the navigation synchronization interface. The application is based on open source such as Mongo and Node.js. The performance evaluation considers the loading and execution delay of the application in a web browser, and implies a perception of satisfaction on the part of the user that depends on the load time, if between 100 and 300 ms elapses, the perception is

considered negative and the user may abandon it. So it is important for any service provider to consider these times to improve their quality and ensure customer retention. The aim of this work is to provide an option to evaluate the performance of any web application in a simple way, by integrating the application in 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 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. Según (Grigoriak, 2013) un usuario de una aplicación web empieza a percibir retraso en el acceso entre un rango de 100-300 ms, por lo que sería deseable garantizar un tiempo de acceso menor por parte de un proveedor de servicios de aplicaciones basadas en el navegador web, y eso implicaría 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 de rendimiento. Actualmente 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, pero 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 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 que usa código Mongo y Node. Js. 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 en la aplicación web, el cual recoleta las variables o métricas de la API de sincronización de navegación (Navigation Timing) y el API de rastreo las almacena, para que se pueda evaluar el rendimiento de la aplicación web en uso.

En la siguiente sección se describe la estructura de una página web y la sincronización de navegación que se necesitan para medir el rendimiento de una interfaz de programación de aplicaciones, y se presenta la propuesta de la interfaz que considera el proceso de registro las variables del rendimiento de una aplicación web entre un cliente y un servidor para realizar su evaluación. Luego, se proporcionan

las pruebas realizas y los resultados. Finalmente se presentas las conclusiones que consideran el posible trabajo futuro.

2 DESARROLLO

2.1 ESTRUCTURA DE UNA PÁGINA WEB

Las aplicaciones web modernas tienen una compleja dependencia entre hojas de estilo en cascada, el lenguaje de marcado de hiper texto y el código de las aplicaciones. Las hojas de estilo en cascada o CSS (Cascading Style Sheet), son un lenguaje de diseño gráfico que se usa para definir y crear la presentación de un documento de hiper texto. El hiper texto o HTML (Hyper Text Markup Language) es un lenguaje estándar que se utiliza para la creación de páginas web y permite que éstas puedan ser interpretada de la misma forma por cualquier navegador web que se acceda por medio de Internet como Explorer, Mozilla o Chrome. El código de las aplicaciones o 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. El análisis del documento HTML es lo que construye el modelo de objetos del documento o DOM (Documento Object Model) que es 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. Además, 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. Como se señala en (Grigoriak, 2013) y se muestra en la Figura 1, los modelos DOM y CSSOM se complementan para representar una página Web, por medio de un árbol de presentación 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 el lenguaje de programación JavaScript (Lenguaje de programación ligera que proporciona efectos y funciones complementarias), 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. Entonces, el rendimiento de la aplicación web depende directamente de cómo se resuelven las dependencias entre el marcado HTML, la hoja CSS y los scripts.

Fuente Grigoriak, 2013.

2.2 ESTRUCTURA DE SINCRONIZACIÓN DE NAVEGACIÓN

Según (Beasley, 2013) la complejidad de la tarea o actividad que debe realizarse, el tipo de negocio y las expectativas del usuario, son los criterios primordiales que se consideran para determinar el rendimiento o la velocidad del procesamiento de una página web. Además, como se menciona en la introducción, en (Grigoriak, 2013) se indica que, a partir de un segundo, el usuario rompe la actividad que esté realizando por el atraso al acceso y con seguridad, después de 10 segundos, el usuario abandona la actividad si no se le da una retroalimentación sobre el progreso de su actividad. Por lo tanto, para que un usuario tenga la sensación de acceso instantáneo a una aplicación web es necesario que ésta se ejecute en el orden de los 100 ms como máximo. Una opción para monitorear los tiempos de acceso es 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 o página web por medio de las variables o métricas que contiene su estructura y que puede verse en la

Figura 2, las variables que se muestran 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* (Wang, 2012) (Grigoriak, 2013).

Fuente Grigoriak, 2013.

En este trabajo se calcula el tiempo total de la carga (*loadEventEnd*- *StartTime*), de solicitud (*responseStart*-*requestStart*) y de respuesta (*responseEnd*- *responseStart*) para medir el rendimiento del tiempo de la carga y el tiempo de ejecución de la página.

2.3 PROPUESTA PARA LA MEDICIÓN DEL RENDIMIENTO

La propuesta considera la integración de componentes de terceros y de elaboración propia que se usan para analizar las métricas del cliente que accede a la aplicación web. El análisis usa un modelo de consumo de datos no síncrono (no requiere una conexión al mismo tiempo), una analítica del cliente que implica el registro de sus eventos. La distinción de las aplicaciones web se realiza con la generación de un identificador único tanto para la aplicación como para el usuario.

El registro de la medición de las métricas de una aplicación web se realiza con un sistema de mensajería no síncrona. El sistema usa un patrón de mensajería conocido en arquitectura de software como publish/subscribe messaging que implica un remitente (publisher o editor), un receptor (subscriber o suscriptor) y un mensaje (message). Los remitentes de mensajes no programan los mensajes que se enviarán directamente a receptores específicos, sino que clasifican los mensajes publicados en clases sin saber cuál es el remitente. Igualmente, los remitentes expresan interés en una o más clases y solamente reciben mensajes que son de interés sin conocer al remitente (Shapira et al, 2017).

La analítica del cliente implica una recopilación o recolección y análisis de las interacciones de los usuarios con los sistemas, éstas se registran como eventos (notificación de un suceso), como por ejemplo las métricas de la estructura de la sincronización de navegación de la

Figura 2, por ejemplo, *loadEventEnd* y *loadEventStart*. La recolección de las métricas se realiza con una API de transferencia de estado representativa o REST (Representational State Transfer) que en este trabajo se nombra API de rastreo y permite a los usuarios conectarse e interactuar con servicios en la nube. La API de rastreo que se propone es una interfaz entre sistemas que usan el protocolo de transferencia de hipertexto o HTTP (Hypertext Transfer Protocol) para acceder a las páginas web a través de Internet. Las métricas se capturan y adecuan con el formato de notación de objetos de JavaScript o JSON (JavaScript Object Notation) que es un formato sencillo para el intercambio de datos. La API REST se usa en alta demanda por medio de un servidor de base de datos o DB (Data Base) 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 hacerlos en tablas. La API REST tiene como objetivos:

- Desacoplar el rastreo del servidor de aplicación principal habilitando un servicio dedicado para este fin.
- Proveer persistencia 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, el cual es 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 Javascript y actualizados mediante Mongoose la cual es una librería de Node.js para almacenar documentos en Mongo DB. La arquitectura del API REST propuesta puede verse en la Figura 3, contiene rutas donde se definen los puntos finales o end points de cada uno de los recursos de servicio, controladores que realizan la lógica necesaria para cumplir con la ruta solicitada, acciones que son las operaciones que se ejecutaran en los modelos, y los modelos son los mecanismos para interactuar con los datos persistentes.

Figura 3. Arquitectura de la solución de rastreo.

Fuente propia

Las interacciones de los componentes del API REST se pueden ver en la

Figura 4, se usan componentes individuales para cada usuario y se identifican como `models/Users.js`, `action/UserActions.js` y `controllors/Users.js`. que registran la ruta, las acciones y los controles de los usuarios que se implementan en `/routes/index.js` y allí mismo se archivan. 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. Los eventos se manejan con la estructura del patrón API de rastreo, usa el componente `models/Events`, `actions/eventsActions.js` y `controller/Events.js` de forma individual que también se implementan en las rutas `/rouerts/index.js`, y contiene la estructura que servirá para almacenar los datos del evento.

Figura 4. Secuencia de componentes del API para la recolección de métricas.

Las métricas de rendimiento también usan la arquitectura API de rastreo, y los componentes individuales de `models/Metrics.js`, `actions/metricsActions.js` y `controller/Metricis.js` que corresponden al modelo de la métrica que contiene la estructura de toda la información disponible del rendimiento, a las acciones de las métricas que obtienen la petición del registro de la métrica, y el controlar de las métricas donde se organizan las variables obtenidas del Navigation Timing API respectivamente, estas se implementan en las rutas `/routers/index.js` para el servicio API de rastreo como se muestra en la

Figura 4.

La secuencia de recolección de métricas se muestra en la

Figura 5, en la que se puede ver la interacción entre componentes para realizar el registro de las métricas. Primero, se debe hacer una transacción previa del registro del cliente para que el servidor de aplicaciones almacene al cliente, y luego se realizan peticiones con el API de rastreo por medio de un POST (solicita la inserción de la entidad enviada) que es una instrucción para envío de HTTP.

Figura 5. Secuencia de recolección de métricas

Por otra parte, para que se pueda evaluar el rendimiento es necesario que se inicie una sesión que permita el acceso a la aplicación web, por lo que es importante identificar esos accesos. Independientemente de que dos accesos a la aplicación web fueran de un mismo usuario o de usuarios diferentes, se puede distinguir cada acceso porque el sistema genera un UUID (Universally Unique Identifier) distinto. La API de rastreo registra ese UUID. La identificación de los accesos se hace con la generación del identificador del usuario durante el inicio de sesión mediante varios componentes, tanto

del lado del cliente como del servidor de aplicación, así como en el servidor de rastreo, como puede verse en la Figura 3 donde se representa un conjunto de clientes web que pueden iniciar sesión y con ello se hace un rastreo de usuario. La interacción de los componentes se ilustra en la Figura 6, si un cliente inicia sesión entonces el servidor de aplicaciones web genera el identificador de la aplicación, así como el identificador del usuario de la aplicación. Luego se hace una llamada al API de rastreo que da una respuesta con un UUID que es una cadena de texto que se almacena en el servidor web para su posterior uso.

2.4 IMPLEMENTACIÓN DE LA INTERFAZ PARA EVALUAR EL RENDIMIENTO

La interfaz para evaluar el rendimiento es una API de rastreo e integra varios componentes, como la parte de identificación de usuario y aplicación, las acciones de las métricas y su controlador, así como el registro de las métricas y la petición para obtenerlas, para que se pueda evaluar el rendimiento, A continuación, se describen cada una de ellas y se muestran los códigos respectivos.

Figura 6. Secuencia de generación de identificadores de rastreo de usuario.

Fuente propia.

La función para generar el identificador de usuario se muestra en la

Figura 7 inciso A), ésta usa una función callAPI que está escrita en procesador de hipertexto o PHP (Hypertext Preprocessor) utilizada del lado del servidor, para hacer llamadas a la API REST. La API REST a su vez usa una llamada POST con la estructura JSON que se muestra en la

Figura 7 inciso B) al API de rastreo. La respuesta del API de rastreo es una UUID como se muestra en la

Figura 7 inciso C).

Figura 7. Código para identificación de usuario y aplicación.


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

```

A) Función para obtener identificador del usuarios

```
{
    "app_id": "identificador de la aplicacion",
    "user_id": "identificador del usuario"
}
```

B) Llamada para la creación del usuario

B) Formato UUID

99bdb232-7dd6-4aa6-9080-758e6e7bcc91

Fuente propia.

En la Figura 8 inciso A) se muestran los códigos de las acciones de las métricas y en el inciso B) del 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 DB. Así es posible hacer peticiones de registro de clientes o usuarios, eventos y métricas.

Figura 8. Códigos de las acciones y controlador de las 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
};
```

A) Acciones de las métricas

```
//Location: controllers/Metrics.js
const { reportMetric, getAllDataMetrics } = require("../actions/metricsActions");

const createMetric = (req, res) => {
    let totalRequest = req.body.responseStart - req.body.requestStart;
    let totalResponse = req.body.responseEnd - req.body.requestStart;
    let totalRequestResponse = req.body.responseEnd - req.body.requestStart;
    let total = req.body.unloadEventEnd - req.body.connectStart;
    req.body.totalRequest = totalRequest;
    req.body.totalResponse = totalResponse;
    req.body.totalRequestResponse = totalRequestResponse;
    req.body.total = total;
    req.body.date = Date.now();
    reportMetric(req.body).then((event) => {
        res.status(201).json(event);
    }).catch(e => res.status(400).json(e));
}

module.exports = {
    createMetric,
    getAllMetrics,
    getMetrics
};
```

B) Controlador de las métricas

Fuente propia.

Las métricas se integran a los componentes del cliente mediante el script de JavaScript con una función denominada métrica, es decir, se integra el script a cada recurso que se desee medir, dentro del cuerpo de HTML, que hace la llamada al 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 9 inciso A), genera una petición a la API de rastreo para que se agreguen las variables de rendimiento al cliente, que recibe una estructura JSON como se muestra por ejemplo en la Figura 9 inciso B), que tiene el registro de las métricas solicitadas, que se almacena y está contenida en el objeto window.performance.timing,

que tiene todas las variables para la evaluación del rendimiento que se obtiene del Navigation Timing API. Una vez procesado el método, este confirma la transacción de manera exitosa con un código 200 Ok.

Figura 9. Función métrica para petición del registro de las métricas de rendimiento. Fuente propia.

```
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));
}
```

A) Función métrica

```
{
  "connectEnd": 1554217642805,
  "connectStart": 1554217642805,
  "domComplete": 1554217642933,
  "domContentLoadedEventEnd": 1554217642929,
  "domContentLoadedEventStart": 1554217642928,
  "domInteractive": 1554217642927,
  "domLoading": 1554217642816,
  "domainLookupEnd": 1554217642805,
  "domainLookupStart": 1554217642805,
  "fetchStart": 1554217642805,
  "loadEventEnd": 1554217642934,
  "loadEventStart": 1554217642933,
  "navigationStart": 1554217642805,
  "redirectEnd": 0,
  "redirectStart": 0,
  "requestStart": 1554217642805,
  "responseEnd": 1554217642805,
  "responseStart": 1554217642805,
  "secureConnectionStart": 1554217642805,
  "unloadEventEnd": 1554217642817,
  "unloadEventStart": 1554217642816,
  "urlApp": "recurso",
  "uuid": "172b58b6-90c7-4ae9-b601-9d21974cf4b7"
}
```

B) Registro de las métricas

3 RESULTADOS

Con la finalidad de verificar la implementación de la interfaz de rastreo para evaluar el rendimiento de una aplicación web, se realizaron pruebas de acceso de cinco clientes a una aplicación administrativa ofrecida como servicio por parte de una empresa privada. La interfaz se agregó a la parte del servidor de aplicaciones de la empresa. Los cinco clientes usaron un navegador de su elección para iniciar sesión desde sus respectivos lugares de trabajo a la plataforma de Windows. No se registró el proveedor de servicio ni la velocidad que les proporcionaba. El tiempo registrado para obtener el tiempo total para la carga de la aplicación por cada cliente o usuario se puede ver en el Cuadro 1, éste fue obtenido desde el navegador web y enviados a la API de rastreo, que se promedió con base en 1 hora de servicio.

Cuadro 1. Evaluación de rendimiento de la aplicación Web de los cinco usuarios.

Usuario	Promedio 1 (0-100ms)	Promedio 2 (100-300ms)	Promedio 3 (300-1000ms)	Promedio 4 (1000-10000ms)
25c9f14a-45d0-11ea-a83e-686ff684b0e6	50.2	207.75	684.033333	5683.45
534a397c-45d0-11ea-a83e-686ff684b0e6	55.5	195.675	620.566667	5914.1
59e37a64-45d0-11ea-a83e-686ff684b0e6	74.8	210.575	625.1	4893.45
5ea5645e-45d0-11ea-a83e-686ff684b0e6	63.4	187.7	663.4	5819.05
63033d0a-45d0-11ea-a83e-686ff684b0e6	54.4	184.4	651.4	4858.2

El promedio de total con base en el tiempo transcurrido coincide con lo que se indica en la literatura (Grigoriak, 2013) y que se muestra en el Cuadro 2, en la columna de acceso en términos de porcentaje.

Cuadro 2. Evaluación de rendimiento de la aplicación Web

Retardo	Percepción del cliente	Acceso
0-100ms	Instantáneo	10%

100-300 ms	Percepción pequeña de retraso	40%
300-1000ms	Está trabajando la máquina	30%
1000 + ms	Probable cambio de contexto	20%
10000 + ms	La tarea es abandonada	0%

Como se muestra en el Cuadro 2, basada en el retardo y la percepción del cliente que se presenta en (Grigoriak, 2013), el 90% de los accesos se considera que tienen un rendimiento aceptable debido a que 40% de las peticiones estuvieron dentro del rango de una percepción de retraso (Cuadro 1 promedio 4), el 30% dentro del rango de que la máquina estaba trabajando (Cuadro 1 promedio 3)) y el 20% en el rango de probable cambio de contexto y olvido de lo que estaba haciendo (Cuadro 1 promedio 2). Sólo el 10% de los accesos tuvo un rendimiento adecuado porque el acceso se percibió de forma instantánea (Cuadro 1 promedio 1).

4 CONCLUSIONES

Con la propuesta, se pudo obtener las métricas de rendimiento de la API de sincronización de navegación (respondeEnd, requestStart, loadEventEnd, domInteractive, StartTime, loadEventEnd, responseStart, responseEnd) de una aplicación web, a través de una función métrica y un API de rastreo tipo API REST. La medición de las métricas de tiempo de carga y ejecución, que se evalúa de acuerdo con el retardo transcurrido desde el inicio hasta el fin de la carga de la aplicación web. La aplicación web usada para este trabajo, muestra con los resultados obtenidos, que sólo el 10% de las veces que se accede su rendimiento es adecuado, pero el 90% de las veces se considera aceptable, por lo que puede presentar un posible riesgo de fracaso. Con eso, se puede verificar que es posible colocar una API REST dentro una aplicación web para brindar información estadística del rendimiento a su proveedor de servicio y también es posible determinar los problemas que origina ese rendimiento, quizá con la comparación de los parámetros de calidad del servicio que ofrece y la opinión del cliente que use la aplicación. Como se señala en (Erickson, 2011) es importante que un proveedor de aplicación web se preocupe más porque su cliente esté satisfecho con su servicio, y eso conlleva a que el proveedor sea más proactivo para evaluar el rendimiento del servicio que otorga y proporcionar una solución en corto plazo. La propuesta presentada proporciona la situación actual del servicio desde la perspectiva de la calidad porque se evalúa el rendimiento de la aplicación web y permite establecer estrategias para asegurar la calidad del servicio porque es posible identificar los problemas que lo originan. También es posible evaluar las métricas de rendimiento en un tiempo determinando a un cliente específico, que podría usarse cuando un cliente en particular se queja del servicio que proporcione el proveedor.

Por otra parte, como trabajo futuro, es posible realizar un análisis emocional del cliente que usa una aplicación web con ayuda de alguna API que registre sus variables físicas. Actualmente, se considera

relevante la experiencia del usuario cuando se diseña e implementa alguna aplicación para asegurar su éxito, así que las emociones de los usuarios juegan un papel importante (Tullis T. y Albert B., 2013). Por ejemplo, Microsoft Emotion API ofrece el reconocimiento de emociones basadas en expresiones faciales. Esta API acepta imágenes y regresa la emoción detectada en el rostro (Microsoft 2019) y mediante este tipo de servicio es posible detectar: ira, desprecio, asco, miedo, felicidad, neutral, tristeza y sorpresa, según (Zhang 2015). Por otro lado, el dispositivo Emotiv® EPOC emplea electroencefalografía, que registra las actividades bio eléctrica cerebral, y es útil en el uso de reconocimiento de emociones para sistemas de comercio electrónico y entretenimiento de acuerdo con (Pham y Tran, 2012).

REFERENCIAS

- Beasley, M. "Practical Web Analytics of User Experience: How Analytics Can Help You Understand Your Users". Copyright Elsevier, Inc. 2013.
- Erickson, L. B. "Web 2.0 and Social Networking for the Enterprise". Research Technology Management. Vol. 31, No. 1 - Vol. 54, 2011.
- Grigoriak, I. "High Performance Browser Networking". O'Reilly Media, Inc. 2013.
- Laurel, B. "Computers as a Theatre". Addison-Wesley Professional 2014. Copyright Pearson Education, Inc. 2014.
- Microsoft, Corp. "Cognitive Service". Consultado de internet en abril de 2019 de Microsoft Azure. Dirección de internet: <https://azure.microsoft.com/en-us/services/cognitive-services/>
- Pham, T. D., y Tran, D. "Emotion Recognition Using the Emotive EPOC Device". Neuronal Information Processing. Vol. 7667. Springer. 2012.
- Portugal, C. "Design, User-Experience and Teaching-Learning". International Conference of Design, user Experience and Usability. Springer. 2014.
- Shapira, G., Narkhede, N., y Palino, T. "Kafka: The Definitive Guide". California: O'Reilly Media. 2017.
- Tullis, T., y Albert, B. "Measuring the User Experience: Collecting, Analyzing, and Presenting usability Metrics". Morgan Kaufmann, 2013.
- Wang Zhiheng (Editor). W3C Recommendation "Navigation Timing". 17 December 2012. Available at <https://www.w3.org/TR/navigation-timing/>
- Yu, Z. y Zhang C. "Image based Static Facial Expression Recognition with Multiple Deep Network Learning". Proceedings of the 2015 ACM on International Conference on Multimodal Interaction. 2015.