v. 2, n. 9, 2022

Scientific Journal of Applied Social and Clinical Science

# INTERFAZ DE PROGRAMACIÓN DE APLICACIONES PARA EVALUAR EL RENDIMIENTO DE UNA APP WEB

***Karen Hernández Rueda***
Departamento de Sistemas de Información. Universidad de Guadalajara
https://orcid.org/0000-0002-7209-2907

***Martha Patricia Martínez Vargas***
Departamento de Sistemas de Información. Universidad de Guadalajara
https://orcid.org/0000-0002-0085-2567

***Mario Damián Casillas***
Cloud Services. Mitratech, INC.
https://orcid.org/0000-0001-7035-2755

***Sandra Elizabeth Hidalgo Pérez***
Departamento de Sistemas de Información. Universidad de Guadalajara
https://orcid.org/0000-0002-9781-331X

**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.

## INTRODUCCIÓN

El término de aplicaciones o páginas web se usa para 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 (PALACIOS; GUAMÁN, CONTENTO, 2018). 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 (RUIZ, 2020). Por lo que el tiempo de carga de una aplicación web es importante para cualquier negocio. En (ECHEVERRÍA, 2016) se menciona "un ingeniero de Google recalca que un retraso de 400ms produce un descenso del 0.44 el posicionamiento orgánico" y que "el 80% de los usuarios deja de ver un video si este se para durante su producción". Además, Echeverría (2016) señala que Google Maps incrementó en un 30% sus peticiones cuando redujo el tamaño de sus ficheros en un 30%, que Yahoo sufre caídas entre 5 y 9% del tráfico cuando presenta atrasos de 0.4 segundos, que Amazon calculó que 0.1 segundos de retraso le implican una pérdida del 1% en sus ingresos, y que Facebook tiene caídas de tráfico del 3% cuando sus páginas son 0.5 segundos más lentas. Asimismo, Echeverría (2016) 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, y el primero tiene limitación de conexión y no permite pruebas personalizadas. GTmetrix da un informe más completo de rendimiento (SUAZO; SMITH, HALSEY, MANDAL, 2020) respecto del resto de las herramientas.

Por otro lado, en (VÁZQUEZ, 2018) se señala que, para reducir los tiempos de cargas de una página web, se pueden optimizar las imágenes y ajustarlas al tamaño que se usan, utilizar conexiones seguras, reducir los recursos que sean posibles, disminuir el número de consultas del sitio a una Base de Datos (DB en inglés), entre otros. En (ECHEVERRIA, 2016) 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, esto se hace con base en los tiempos de respuesta de 0.1 segundos, 1 segundos y

10 segundos según Nielsen (1993) y se usa el analizador PageSpeeds Insights que mide el rendimiento de una red de 0 a 100 puntos, entre más puntos, mejor el rendimiento. En (GRIGORIK, 2013) 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 (Representacional State Transfer) que usa código abierto. En la Figura 1 se muestra un ejemplo de funcionamiento de una API, esta 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 recoleta 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.

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. Posteriormente los resultados y, por último, se dan las conclusiones con información de trabajo futuro.

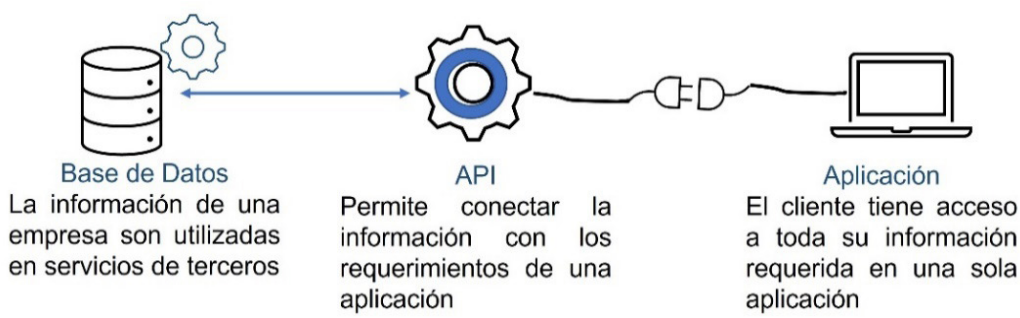


Figura 1. Ejemplo de funcionamiento de una API. Fuente (VINIEGA, 2020).

## MARCO TEÓRICO

### 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 (LUJÁN-MORA, 2002). 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 (LUJÁN-MORA, 2002). CSS es 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, (LUJÁN-MORA, 2002). 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 (ORACLE, 2016). 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 (ORÓS; NAVA, 2010).

### PARTES DE UN NAVEGADOR

Un navegador necesita de una interfaz de usuario (user interface), motor de búsqueda (browser engine), motor de renderización (rendering engine), red (networking), servidor de interfaz (UI backend), intérprete de JS (JavaScript interpreter) y almacenamiento de datos (data persistence) para que pueda mostrar una página web (WEBKIT, s.f.). Adicionalmente, cada navegador tiene características únicas. La interfaz de usuario 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 búsqueda coordina las acciones entre la interfaz de usuario y el motor de renderización. La red es responsable de las llamadas a la red, por ejemplo, hacer solicitudes HTTP (Hypertext Transfer Protocolo o Protocolo de Transferencia de Hypertexto) para acceder a una página web. El servidor de interfaz permite presentar dibujos, fuentes, ventas y cuadros combinados. El intérprete de JavaScript es el

que analiza y ejecuta código de JavaScript. El almacenamiento de datos guarda varios datos asociados con la sesión de navegación en el disco (marcadores, cookies, etc). En la figura 2 se muestra las partes del navegador y la relación entre ellos. El motor de renderizado 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). 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 una el WebKit (WEBKIT, s.f.). Este último se usa en la plataforma Linux, pero se modificó por Apple para hacerlo compatible con Mac y Windows.

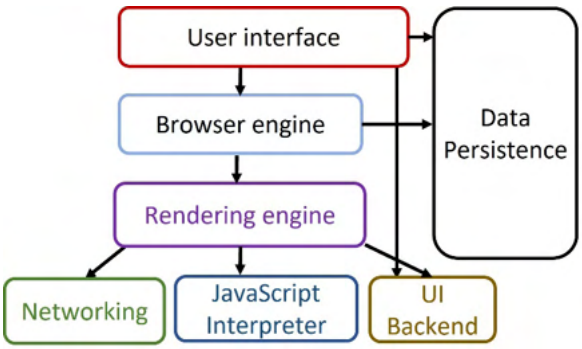


Figura 2. Partes de un navegador.

Fuente (GROSSKURTH; GODFREY, 2006).

## 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 (ORÓS; NAVA, 2010). 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 3. Flujo WebKit. Fuente (ORÓS; NAVA, 2010). se puede ver el flujo de un motor WebKit (WEBKIT, s.f.). 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.

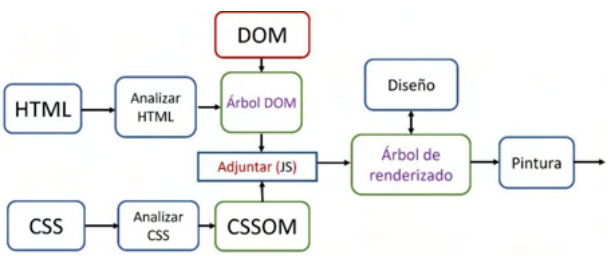


Figura 3. Flujo WebKit.

Fuente (ORÓS; NAVA, 2010).

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. Por esto, el rendimiento de una aplicación

depende directamente de cómo se resuelven las dependencias entre el marcado HTML, las hojas de estilo y el JS.

## RENDIMIENTO DE UNA PÁGINA WEB

El rendimiento de una página web es el tiempo que se demora en cargarse desde su petición. Entre mayor sea el rendimiento más fluido es el acceso a la página, con menor tiempo y más agradable para los usuarios (PALACIOS; GUAMÁN, CONTENTO, 2018). 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 3. Flujo WebKit. Fuente (ORÓS; NAVA, 2010).. En (BEASLY, 2013) 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) (MDN, 2021). De acuerdo con (GRIGORIK, 2013), 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. Por lo que 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 4. Estructura de la API de sincronización. Fuente (GRIGORIK, 2013). 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 repuesta, 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

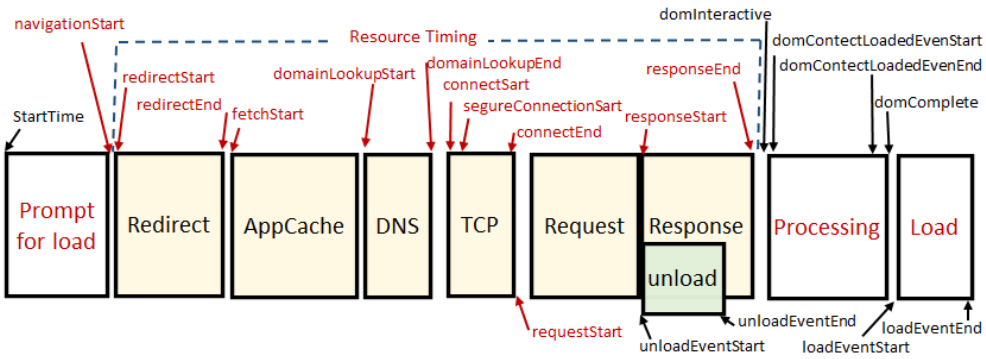


Figura 4. Estructura de la API de sincronización. Fuente (GRIGORIK, 2013).

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, 2013) (GRIGORIK, 2013).

## RECUPERACIÓN DE LAS MÉTRICAS

Es necesario el registro de la medición de las métricas mostradas en la Figura 4. Estructura de la API de sincronización. Fuente (GRIGORIK, 2013). 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 (NARKHEDE; SHAPIRA, PALINO, 2017). 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 o API de rastreo que permite a los usuarios conectarse e interactuar con servicios en la nube. La API REST que se propone (Figura 5) es una interfaz entre sistemas que usan HTTP 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 NodeJS, 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.

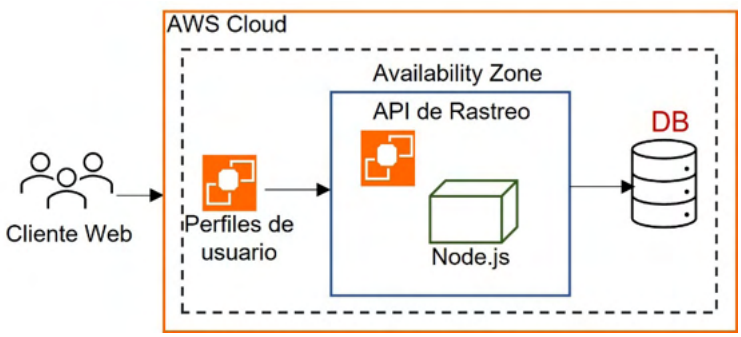


Figura 5. Arquitectura propuesta. Fuente propia.

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. Los datos persistentes o modelos que se usan para la recolección son el modelo de usuarios, el modelo de métricas y el modelo de eventos. En el modelo de usuario se define la información con el tipo de datos que será almacenada en la DB. El modelo de métricas contiene información de los tiempos totalRequest, totalReponse y totalRequestResponse que ayudan a calcular el rendimiento. El modelo de evento registra el acceso de un usuario.

## METODOLOGÍA

Primero se realizó una investigación es exploratoria para conocer trabajos similares y propuestas para medir el rendimiento de sitios web. Luego se aplicó un estudio cuasi experimental en la que se realizó una propuesta que considera el diseño de la API de rastreo, los parámetros para hacer la evaluación y la prueba de usuario enfocado en tiempo de respuesta. A continuación, se explica la solución propuesta, la implementación realizada y la etapa de prueba.

### API DE RASTREO PARA REGISTRAR LAS MÉTRICAS

La figura 6 muestra la arquitectura de la solución de rastreo. La API de rastreo contiene la librería Node.js que integra los elementos de ruta, controladores, acciones y modelos. En la ruta se definen los puntos finales donde están definidos cada uno de los recursos de servicio que se ofrece. Los controladores se encargan de realizar la lógica necesaria para cumplir con la ruta solicitada. Las acciones son las operaciones que efectúan los modelos, y los modelos son los mecanismos para interactuar con los datos persistentes.

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 API de rastreo contiene rutas, controladores, acciones y modelos. 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. La identificación del acceso se hace con una llamada API de rastreo que da una respuesta con un UUID (Universally Unique IDentifier o Identificar Único Universal) que es una cadena de texto (ejemplo 99bdb232-7dd6-4aa6-

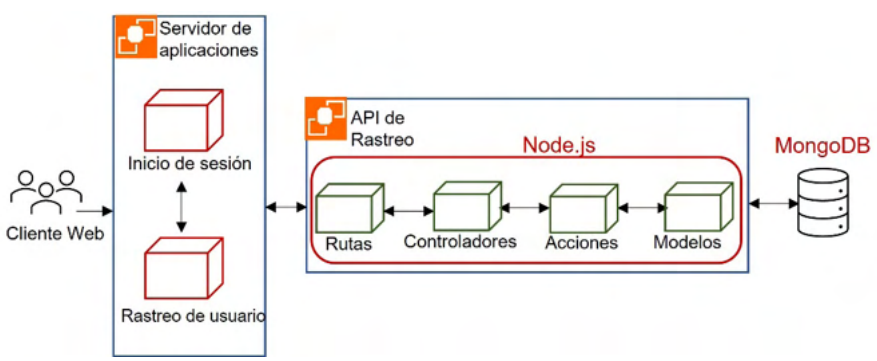


Figura 6. Arquitectura de la solución de rastreo. Fuente propia.

758e6e7bcc91) que se almacena en el servidor web. El sistema genera un UUID (Figura 7. Secuencia de recolección de métricas. Fuente propia.7, 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.

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/Metrcis.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 controlar organiza las variables obtenidas del Navigation Timing API, que se implementan en la misma ruta de los usuarios y eventos. En la Figura 7. Secuencia de recolección de métricas. Fuente propia., 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.

**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 8, 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 una UUID.

En la figura 9 se muestran los códigos de

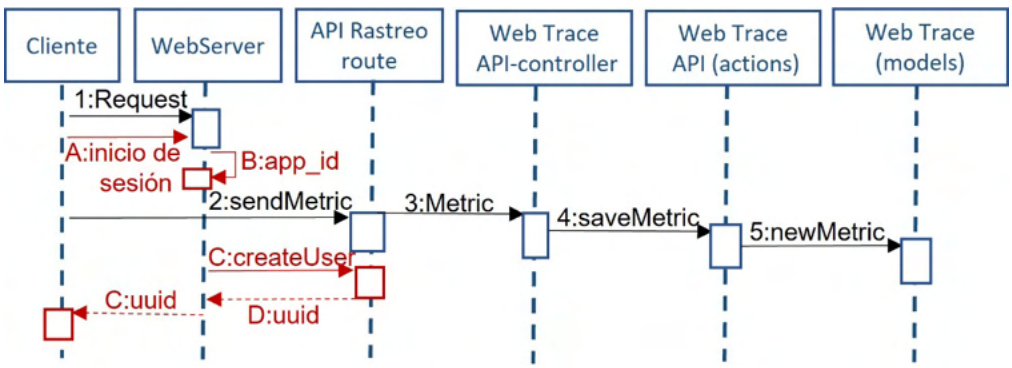


Figura 7. Secuencia de recolección de métricas. Fuente propia.

```
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;
 }
```

Figura 8. Código para identificación de usuario y aplicación.

```
//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 9. 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.date= Date.now();
     reportMetric(req.body).then(event) => {
        res.status(201).json(event);
     }).cath(e => res.status(400).json(e));
}
   module.exports = {
      createMetric,
      getAllMetrics,
      getMetrics
};
```

Figura 10. Código del controlador de las métricas.

las acciones de las métricas y en la figura 10 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.

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 11. 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 desee 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 12, 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.

### ETAPA DE PRUEBA

Las pruebas que se consideraron como parte de la usabilidad de la aplicación web fueron completar una transacción, evaluar el uso frecuente del producto, el aumento de la atención y la usabilidad de un producto crítico. Para la prueba una se espera que el usuario complete una compra, un registro o el reinicio de una contraseña, para la prueba dos se va a medir el tiempo de la tarea, para la prueba tres se monitoreará el número de interacciones con la página, para la prueba cuatro se va a solicitar que el usuario de valore en un rango si se completa o no la tarea o si se cumple o no. Las pruebas se realizaron a través de la interfaz de rastreo, este 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.

## RESULTADO

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. 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

```
//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",getAllUser);
     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 11. Rutas de la aplicación.

```
function metric (url_api,uuid,url){
    var url=url_api;
    var data=window.performande.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 12. Función para agregar las métricas de rendimiento.

| **Cliente** | **Promedio 1**<br>0-100ms | **Promedio 2**<br>100-300ms | **Promedio 3**<br>300-1000ms | **Promedio 4**<br>+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 |
| **%** | **10** | **40** | **30** | **20** |

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

percibió de forma instantánea (Promedio 1 Tabla 1).

## CONCLUSIONES

El resultado verifica lo que indica Grigorik (2013); un 40% de los usuarios estuvo dentro del rango de una percepción pequeña de retraso, un 30% en el rango de que la máquina estaba trabajando y un 20% en el rango de que el usuario pudo cambiar de contexto y olvidar lo que estaba haciendo. Con estos datos se puede concluir que en el 90% de los tiempos de respuesta de los datos (con las métricas obtenidas), el usuario percibió el rendimiento lento o con retraso. 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. 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. 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.

## REFERENCIAS


BEASLY, M. (2013). Practical Web Analytics of User Experience: How Analytics Can Help You Understand Your Users. Waltham, MA, United States of America: Elsevier Inc.

ECHEVARRÍA, D. (2016). Tiempo de Respuesta y Experiencia de Usuario, Estudio Experimental. *Revista Latinoamericana de Ingeniería de Software, 4(5),* 231-234.

GRIGORIK, I. (2013). *High Performance Browser Networking* (vol. 53). O'Reilly Media Inc.

GROSSKURTH, A.; GODFREY, M. W (2006). Architecture and evolution of the modern web browser. Prepritn submitted to Elsevier Science, 12(6), 235-246.

LUJÁN-MORA, S. (2016). Programación de aplicaciones web: historia, principios básicos y clientes web. San Vicente, Alicante, España: Universitario, Editorial Club.

MDN. (2021). *MDN Web Documents*. Recuperado el 28 de junio de 2021, de https://developer.mozilla.org/es/docs/Learn/Performance.

NARKHEDE, N.; SHAPIRA, G., PALINO, T. (2017). *Kafka: The Definitive Guide Real-Time Data and Stream Processing at Scale*. (S. Cut, Ed.), Sebastopol, United State of America: O'Reilly Media, Inc., 2017.

NIELSEN, J. (1993). NN/g Nielsen Nomran Group. [En línea]. Available: https://www.nngroup.com/articles/response-times-3-important-limits/.

ORACLE. (2016). Java. Recuperado el 28 de junio de 2021, de https://www.java.com/es/download/help/java_javascript.html.

ORÓS J. C.; NAVA Martínez, M. N. (2010). *Diseño de páginas web con: XTML, JavaScript y CSS*. Ra-Ma. p. 376.

PALACIOS, D.; GUAMÁN, J., CONTENTO, S. (2018). Análisis del rendimiento de librerías de componentes Java Server Faces en el desarrollo de aplicaciones web. *NOVASINERGIA,* 2(1), 54-59.

RUIZ TAUSTE, P. (2020). *LMWP Lo Mejor de WordPress*. Recuperado 28 de junio 2020, de http://www.ipixelestudio.com/medir-mejorar-velocidad-carga-pagina-web/.

SUAZO, C. E.; SMITH, J., HALSEY, L., MANDAL, S. (2021). TrusRadious. Recuperado el 17 enero 2020, de https://www.trustradius.com/compare-products/google-pagespeed-insights-vs-gtmetrix.

VÁZQUEZ, S. A. (2018). *Optimización de Páginas Web: Visión teórica y análisis práctico* (Tesis de licenciatura de Tecnologás de las Telecomunciaciones, Universidad de Sevilla). Recuperado el 30 de agosto de 2021, de https://idus.us.es/bitstream/handle/11441/63749/TFG%20Adri%c3%a1n%20V%c3%a1zquez%20Sanisidro.pdf?sequence=1&isAllowed=y.

VINIEGA, J. S. (2020). *Blog de programación avanzada de software*. Recuperado 29 de junio de 2020, de https://proasw.wordpress.com/2020/09/12/estrategias-de-sincronizacion/.

WANG, Z. (2012). *W3C*. Recuperado el 29 de junio de 2021, de https://www.w3.org/TR/navigation-timing/.

WEBKIT. (s.f.). WebKit. Recuperado el 28 de junio de 2021, de https://webkit.org/.